

Чару Аггарвал

Нейронные сети и глубокое обучение

Учебный курс

Нейронные сети и глубокое обучение

Учебный курс

Charu C. Aggarwal

Neural Networks and Deep Learning

A Textbook

 Springer

Чару Аггарвал

Нейронные сети и глубокое обучение

Учебный курс



Москва · Санкт-Петербург
2020

ББК 32.973.26-018.2.75

A23

УДК 681.3.07

Компьютерное издательство “Диалектика”

Перевод с английского канд. хим. наук А.Г. Гузиковича

Под редакцией В.Р. Гинзбурга

По общим вопросам обращайтесь в издательство “Диалектика” по адресу:

info@dialektika.com, <http://www.dialektika.com>

Аггарвал, Чару

A23 Нейронные сети и глубокое обучение: учебный курс. : Пер. с англ. — СПб. :
ООО “Диалектика”, 2020. — 752 с. — Парал. тит. англ.

ISBN 978-5-907203-01-3 (рус.)

ББК 32.973.26-018.2.75

Все названия программных продуктов являются зарегистрированными торговыми марками соответствующих фирм.

Никакая часть настоящего издания ни в каких целях не может быть воспроизведена в какой бы то ни было форме и какими бы то ни было средствами, будь то электронные или механические, включая фотокопирование и запись на магнитный носитель, если на это нет письменного разрешения издательства Springer.

Authorized Russian translation of the English edition of *Neural Networks and Deep Learning: A Textbook* (ISBN 978-3-319-94462-3) © Springer International Publishing AG, part of Springer Nature 2018

This translation is published and sold by permission of Springer, which owns or controls all rights to sell the same.

All rights reserved. No part of this book may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording, or by any information storage or retrieval system, without the prior written permission of the copyright owner and the Publisher.

Научно-популярное издание

Чару Аггарвал

**Нейронные сети и глубокое обучение:
учебный курс**

ООО “Диалектика”, 195027, Санкт-Петербург, Магнитогорская ул., д. 30, лит. А, пом. 848

ISBN 978-5-907203-01-3 (рус.)

ISBN 978-3-319-94462-3 (англ.)

© 2020 ООО “Диалектика”

© Springer International Publishing AG,
part of Springer Nature 2018

Оглавление

Предисловие	17
Глава 1. Введение в нейронные сети	23
Глава 2. Машинное обучение с помощью мелких нейронных сетей	101
Глава 3. Обучение глубоких нейронных сетей	179
Глава 4. Обучение глубоких сетей способности к обобщению	271
Глава 5. Сети радиально-базисных функций	345
Глава 6. Ограниченные машины Больцмана	369
Глава 7. Рекуррентные нейронные сети	423
Глава 8. Сверточные нейронные сети	485
Глава 9. Глубокое обучение с подкреплением	569
Глава 10. Дополнительные вопросы глубокого обучения	637
Библиография	695
Предметный указатель	737

Содержание

Предисловие	17
Об авторе	20
Ждем ваших отзывов!	21
Глава 1. Введение в нейронные сети	23
1.1. Введение	23
1.1.1. Человек и компьютер: расширение пределов возможностей искусственного интеллекта	26
1.2. Базовая архитектура нейронных сетей	29
1.2.1. Одиночный вычислительный слой: перцептрон	29
1.2.2. Многослойные нейронные сети	46
1.2.3. Многослойная нейронная сеть как вычислительный граф	51
1.3. Тренировка нейронной сети с помощью алгоритма обратного распространения ошибки	52
1.4. Практические аспекты тренировки нейронных сетей	57
1.4.1. Проблема переобучения	58
1.4.2. Проблемы затухающих и взрывных градиентов	63
1.4.3. Трудности со сходимостью	64
1.4.4. Локальные и ложные оптимумы	64
1.4.5. Вычислительные трудности	65
1.5. Преимущества композиции функций	66
1.5.1. Важность нелинейной активации	69
1.5.2. Снижение требований к параметрам с помощью глубины	71
1.5.3. Нестандартные архитектуры нейронных сетей	73
1.6. Распространенные архитектуры нейронных сетей	76
1.6.1. Имитация базового машинного обучения с помощью мелких моделей	76
1.6.2. Сети радиально-базисных функций	77
1.6.3. Ограниченные машины Больцмана	78
1.6.4. Рекуррентные нейронные сети	79
1.6.5. Сверточные нейронные сети	82
1.6.6. Иерархическое конструирование признаков и предварительное обучение моделей	84

1.7. Дополнительные темы	87
1.7.1. Обучение с подкреплением	87
1.7.2. Отделение хранения данных от вычислений	88
1.7.3. Генеративно-сопоставительные сети	89
1.8. Два показательных эталонных теста	90
1.8.1. База данных рукописных цифр MNIST	90
1.8.2. База данных ImageNet	92
1.9. Резюме	93
1.10. Библиографическая справка	94
1.10.1. Видеолекции	96
1.10.2. Программные ресурсы	97
1.11. Упражнения	98
Глава 2. Машинное обучение с помощью мелких нейронных сетей	101
2.1. Введение	101
2.2. Архитектуры нейронных сетей для моделей бинарной классификации	104
2.2.1. Повторное рассмотрение перцептрона	105
2.2.2. Регрессия по методу наименьших квадратов	107
2.2.3. Логистическая регрессия	113
2.2.4. Метод опорных векторов	116
2.3. Архитектуры нейронных сетей для мультиклассовых моделей	119
2.3.1. Мультиклассовый перцептрон	119
2.3.2. SVM Уэстона — Уоткинса	121
2.3.3. Мультиномиальная логистическая регрессия (классификатор Softmax)	123
2.3.4. Иерархическая функция <i>Softmax</i> для случая многих классов	125
2.4. Использование алгоритма обратного распространения ошибки для улучшения интерпретируемости и выбора признаков	126
2.5. Факторизация матриц с помощью автокодировщиков	127
2.5.1. Автокодировщик: базовые принципы	128
2.5.2. Нелинейные активации	135
2.5.3. Глубокие автокодировщики	138
2.5.4. Обнаружение выбросов	142
2.5.5. Когда скрытый слой шире входного	142
2.5.6. Другие применения	144
2.5.7. Рекомендательные системы: предсказание значения для индекса строки	147
2.5.8. Обсуждение	151

2.6. Word2vec: применение простых архитектур нейронных сетей	151
2.6.1. Нейросетевые вложения в рамках модели непрерывного мешка слов	153
2.6.2. Нейросетевые вложения в рамках модели скип-грамм	157
2.6.3. Модель SGNS — это логистическая матричная факторизация	164
2.6.4. Простая модель скип-грамм — это мультиномиальная матричная факторизация	168
2.7. Простые архитектуры нейронных сетей для вложений графов	169
2.7.1. Обработка произвольных значений счетчиков ребер	171
2.7.2. Мультиномиальная модель	171
2.7.3. Связь с моделями DeepWalk и node2vec	172
2.8. Резюме	172
2.9. Библиографическая справка	173
2.9.1. Программные ресурсы	175
2.10. Упражнения	175
Глава 3. Обучение глубоких нейронных сетей	179
3.1. Введение	179
3.2. Алгоритм обратного распространения ошибки:	
подробное рассмотрение	182
3.2.1. Анализ обратного распространения ошибки с помощью абстракции вычислительного графа	182
3.2.2. На выручку приходит динамическое программирование	187
3.2.3. Обратное распространение ошибки с постактивационными переменными	189
3.2.4. Обратное распространение ошибки с преактивационными переменными	193
3.2.5. Примеры обновлений для различных активаций	196
3.2.6. Обособленное векторное представление процесса обратного распространения ошибки	198
3.2.7. Функции потерь на нескольких выходных и скрытых узлах	201
3.2.8. Мини-пакетный стохастический градиентный спуск	203
3.2.9. Приемы обратного распространения ошибки для обработки разделяемых весов	206
3.2.10. Проверка корректности вычисления градиентов	207
3.3. Настройка и инициализация сети	208
3.3.1. Тонкая настройка гиперпараметров	208
3.3.2. Предварительная обработка признаков	210
3.3.3. Инициализация	214

3.4. Проблемы затухающих и взрывных градиентов	215
3.4.1. Геометрическая интерпретация эффекта отношений градиентов	217
3.4.2. Частичное устранение проблем за счет выбора функции активации	219
3.4.3. Мертвые нейроны и “повреждение мозга”	221
3.5. Стратегии градиентного спуска	222
3.5.1. Регулирование скорости обучения	223
3.5.2. Метод импульсов	224
3.5.3. Скорости обучения, специфические для параметров	227
3.5.4. Овраги и нестабильность более высокого порядка	232
3.5.5. Отсечение градиентов	234
3.5.6. Производные второго порядка	235
3.5.7. Усреднение Поляка	246
3.5.8. Локальные и ложные минимумы	247
3.6. Пакетная нормализация	249
3.7. Практические приемы ускорения вычислений и сжатия моделей	254
3.7.1. Ускорение с помощью GPU	255
3.7.2. Параллельные и распределенные реализации	258
3.7.3. Алгоритмические приемы сжатия модели	260
3.8. Резюме	265
3.9. Библиографическая справка	265
3.9.1. Программные ресурсы	267
3.10. Упражнения	268
Глава 4. Обучение глубоких сетей способности к обобщению	271
4.1. Введение	271
4.2. Дилемма смещения — дисперсии	278
4.2.1. Формальное рассмотрение	280
4.3. Проблемы обобщаемости модели при ее настройке и оценке	284
4.3.1. Оценка точности с помощью отложенного набора данных и перекрестной проверки	287
4.3.2. Проблемы с крупномасштабной тренировкой	288
4.3.3. Как определить необходимость в сборе дополнительных данных	289
4.4. Регуляризация на основе штрафов	289
4.4.1. Связь регуляризации с внедрением шума	292
4.4.2. L_1 -регуляризация	293
4.4.3. Что лучше: L_1 - или L_2 -регуляризация?	294
4.4.4. Штрафование скрытых элементов: обучение разреженным представлениям	295

4.5. Ансамблевые методы	296
4.5.1. Бэггинг и подвыборка	297
4.5.2. Выбор и усреднение параметрических моделей	299
4.5.3. Рандомизированное отбрасывание соединений	300
4.5.4. Дропаут	301
4.5.5. Ансамбли на основе возмущения данных	305
4.6. Ранняя остановка	306
4.6.1. Ранняя остановка с точки зрения дисперсии	307
4.7. Предварительное обучение без учителя	308
4.7.1. Разновидности неконтролируемого предварительного обучения	313
4.7.2. Контролируемое предварительное обучение	314
4.8. Обучение с продолжением и поэтапное обучение	316
4.8.1. Обучение с продолжением	318
4.8.2. Поэтапное обучение	318
4.9. Разделение параметров	319
4.10. Регуляризация в задачах обучения без учителя	321
4.10.1. Штрафы на основе значений: разреженные автокодировщики	321
4.10.2. Внедрение шума: шумоподавляющие автокодировщики	322
4.10.3. Штрафование на основе градиентов: сжимающие автокодировщики	324
4.10.4. Скрытая вероятностная структура: вариационные автокодировщики	328
4.11. Резюме	338
4.12. Библиографическая справка	338
4.12.1. Программные ресурсы	341
4.13. Упражнения	341
Глава 5. Сети радиально-базисных функций	345
5.1. Введение	345
5.1.1. Когда следует использовать RBF-сети	349
5.2. Тренировка RBF-сети	349
5.2.1. Тренировка скрытого слоя	350
5.2.2. Тренировка выходного слоя	352
5.2.3. Ортогональный метод наименьших квадратов	354
5.2.4. Полностью контролируемое обучение	355
5.3. Разновидности и специальные случаи RBF-сетей	357
5.3.1. Классификация с использованием критерия перцептрона	358
5.3.2. Классификация с кусочно-линейной функцией потерь	358

5.3.3. Пример линейной разделимости, обеспечиваемой RBF-функциями	359
5.3.4. Применение в задачах интерполяции	360
5.4. Связь с ядерными методами	362
5.4.1. Ядерная регрессия как специальный случай RBF-сетей	362
5.4.2. Ядерный метод SVM как специальный случай RBF-сетей	363
5.4.3. Замечания	364
5.5. Резюме	365
5.6. Библиографическая справка	365
5.7. Упражнения	366
Глава 6. Ограниченные машины Больцмана	369
6.1. Введение	369
6.1.1. Исторический экскурс	370
6.2. Сети Хопфилда	371
6.2.1. Оптимальные конфигурации состояний обученной сети	373
6.2.2. Обучение сети Хопфилда	376
6.2.3. Создание экспериментальной рекомендательной системы и ее ограничения	377
6.2.4. Улучшение выразительных возможностей сети Хопфилда	379
6.3. Машина Больцмана	380
6.3.1. Как машина Больцмана генерирует данные	382
6.3.2. Обучение весов машины Больцмана	383
6.4. Ограниченная машина Больцмана	386
6.4.1. Обучение RBM	389
6.4.2. Алгоритм контрастной дивергенции	391
6.4.3. Практические рекомендации и возможные видоизменения процедуры	392
6.5. Применение ограниченных машин Больцмана	394
6.5.1. Снижение размерности и реконструирование данных	394
6.5.2. Применение RBM для коллаборативной фильтрации	398
6.5.3. Использование RBM для классификации	403
6.5.4. Тематическое моделирование с помощью RBM	407
6.5.5. Использование RBM для машинного обучения с мультимодальными данными	410
6.6. Использование RBM с данными, не являющимися бинарными	412
6.7. Каскадные ограниченные машины Больцмана	413
6.7.1. Обучение без учителя	416

6.7.2. Обучение с учителем	417
6.7.3. Глубокие машины Больцмана и глубокие сети доверия	417
6.8. Резюме	418
6.9. Библиографическая справка	419
6.10. Упражнения	421
Глава 7. Рекуррентные нейронные сети	423
7.1. Введение	423
7.1.1. Выразительная способность рекуррентных нейронных сетей	427
7.2. Архитектура рекуррентных нейронных сетей	428
7.2.1. Пример языкового моделирования с помощью RNN	432
7.2.2. Обратное распространение ошибки во времени	435
7.2.3. Двухнаправленные рекуррентные сети	439
7.2.4. Многослойные рекуррентные сети	442
7.3. Трудности обучения рекуррентных сетей	444
7.3.1. Послойная нормализация	448
7.4. Эхо-сети	450
7.5. Долгая краткосрочная память (LSTM)	453
7.6. Управляемые рекуррентные блоки	458
7.7. Применение рекуррентных нейронных сетей	460
7.7.1. Автоматическое аннотирование изображений	461
7.7.2. Обучение “последовательность в последовательность” и машинный перевод	463
7.7.3. Классификация на уровне предложений	469
7.7.4. Классификация на уровне токенов с использованием лингвистических признаков	470
7.7.5. Прогнозирование и предсказание временных рядов	472
7.7.6. Временные рекомендательные системы	474
7.7.7. Предсказание вторичной структуры белка	478
7.7.8. Сквозное распознавание речи	479
7.7.9. Распознавание рукописного текста	479
7.8. Резюме	480
7.9. Библиографическая справка	481
7.9.1. Программные ресурсы	482
7.10. Упражнения	483

Глава 8. Сверточные нейронные сети	485
8.1. Введение	485
8.1.1. Исторический обзор и биологические предпосылки	486
8.1.2. Общие замечания относительно сверточных нейронных сетей	488
8.2. Базовая структура сверточной сети	489
8.2.1. Дополнение	495
8.2.2. Шаговая свертка	498
8.2.3. Типичные параметры	499
8.2.4. Слой ReLU	500
8.2.5. Пулинг	501
8.2.6. Полносвязные слои	503
8.2.7. Чередование слоев	504
8.2.8. Нормализация локального отклика	508
8.2.9. Иерархическое конструирование признаков	508
8.3. Тренировка сверточной сети	510
8.3.1. Обратное распространение ошибки через свертки	511
8.3.2. Обратное распространение ошибки как свертка с инвертированным/транспонированным фильтром	512
8.3.3. Свертка и обратное распространение ошибки как матричное умножение	514
8.3.4. Аугментация данных	517
8.4. Примеры типичных сверточных архитектур	518
8.4.1. Сеть AlexNet	519
8.4.2. Сеть ZFNet	523
8.4.3. Сеть VGG	524
8.4.4. Сеть GoogLeNet	528
8.4.5. Сеть ResNet	531
8.4.6. Эффекты глубины	536
8.4.7. Предварительно обученные модели	537
8.5. Визуализация и обучение без учителя	539
8.5.1. Визуализация признаков обученной сети	540
8.5.2. Сверточные автокодировщики	548
8.6. Применение сверточных сетей	555
8.6.1. Извлечение изображений на основе содержимого	555
8.6.2. Локализация объектов	555
8.6.3. Обнаружение объектов	558
8.6.4. Распознавание естественного языка и обучение последовательностей	559
8.6.5. Классификация видео	560

8.7. Резюме	561
8.8. Библиографическая справка	562
8.8.1. Программные ресурсы и наборы данных	565
8.9. Упражнения	566
Глава 9. Глубокое обучение с подкреплением	569
9.1. Введение	569
9.2. Алгоритмы без запоминания состояний: многорукие бандиты	572
9.2.1. Наивный алгоритм	573
9.2.2. Жадный алгоритм	574
9.2.3. Методы верхней границы	574
9.3. Базовая постановка задачи обучения с подкреплением	575
9.3.1. Трудности обучения с подкреплением	578
9.3.2. Простой алгоритм обучения с подкреплением для игры крестики-нолики	579
9.3.3. Роль глубокого обучения и алгоритм “соломенного пугала”	580
9.4. Бутстрэппинг для обучения функции оценки	583
9.4.1. Модели глубокого обучения как аппроксиматоры функций	585
9.4.2. Пример: нейронная сеть для игр на платформе Atari	589
9.4.3. Методы, привязанные и не привязанные к стратегии: SARSA	590
9.4.4. Моделирование состояний, а не пар “состояние — действие”	592
9.5. Градиентный спуск по стратегиям	596
9.5.1. Метод конечных разностей	598
9.5.2. Методы относительного правдоподобия	599
9.5.3. Сочетание обучения с учителем с градиентными методами моделирования стратегий	602
9.5.4. Методы “актор — критик”	602
9.5.5. Непрерывное пространство действий	605
9.5.6. Преимущества и недостатки градиентного спуска по стратегиям	606
9.6. Поиск по дереву методом Монте-Карло	606
9.6.1. Использование в бутстрэппинге	608
9.7. Типовые примеры	609
9.7.1. AlphaGo: достижение чемпионского уровня в игре го	609
9.7.2. Самообучающиеся роботы	616
9.7.3. Создание разговорных систем: глубокое обучение чат-ботов	621
9.7.4. Беспилотные автомобили	624
9.7.5. Предложение нейронных архитектур с помощью обучения с подкреплением	627

9.8. Практические вопросы безопасности	629
9.9. Резюме	630
9.10. Библиографическая справка	631
9.10.1. Программные ресурсы и испытательные системы	633
9.11. Упражнения	634
Глава 10. Дополнительные вопросы глубокого обучения	637
10.1. Введение	637
10.2. Механизмы внимания	639
10.2.1. Рекуррентные модели визуального внимания	642
10.2.2. Механизмы внимания для машинного перевода	646
10.3. Нейронные сети с внешней памятью	651
10.3.1. Воображаемая видеоигра: обучение сортировке на примерах	652
10.3.2. Нейронные машины Тьюринга	655
10.3.3. Дифференциальный нейронный компьютер: краткий обзор	663
10.4. Генеративно-сопоставительные сети	664
10.4.1. Тренировка генеративно-сопоставительной сети	666
10.4.2. Сравнение с вариационным автокодировщиком	670
10.4.3. Использование GAN для генерирования данных изображений	671
10.4.4. Условные генеративно-сопоставительные сети	673
10.5. Соревновательное обучение	679
10.5.1. Векторная квантизация	680
10.5.2. Самоорганизующиеся карты Кохонена	681
10.6. Ограничения нейронных сетей	685
10.6.1. Амбициозная задача: разовое обучение	686
10.6.2. Амбициозная задача: энергетически эффективное обучение	688
10.7. Резюме	690
10.8. Библиографическая справка	691
10.8.1. Программные ресурсы	693
10.9. Упражнения	693
Библиография	695
Предметный указатель	737

Предисловие

*Любой ИИ, достаточно сообразительный для того,
чтобы пройти тест Тьюринга, сможет также
сообразить, как провалить этот тест.*

Иен Макдональд

Нейронные сети были разработаны с целью имитации нервной системы человека для решения задач машинного обучения на основе вычислительных элементов, работа которых напоминала бы действие человеческих нейронов. Концептуальной задачей, стоящей перед теми, кто разрабатывает нейронные сети, является создание искусственного интеллекта (ИИ) на основе машин, архитектура которых была бы способна имитировать процессы, происходящие в нервной системе человека. Совершенно очевидно, что задача такого масштаба не может иметь простого решения, поскольку вычислительная мощность даже самых быстрых современных компьютеров составляет лишь ничтожную долю возможностей человеческого мозга. Первые сообщения о нейронных сетях появились вскоре после прихода в нашу жизнь компьютеров в 50-60-х годах прошлого столетия. Алгоритм перцептрона Розенблатта был воспринят в качестве панацеи нейронных сетей, что поначалу вызвало небывалый ажиотаж относительно перспектив искусственного интеллекта. Но после того как первоначальная эйфория пошла на спад, наступил период разочарований, на протяжении которого завышенные запросы в отношении требуемых объемов данных и вычислительных ресурсов нейронных сетей считались основной помехой на пути их дальнейшего развития. В конечном счете на рубеже тысячелетия повышение доступности данных и увеличение вычислительных мощностей обеспечили дальнейший прогресс нейронных сетей, и эта область возродилась под названием “глубокое обучение”. И хотя мы по-прежнему все еще далеки от того дня, когда искусственный интеллект достигнет уровня человеческих возможностей, существуют такие специфические области, как распознавание образов или игры, в которых ИИ сравнялся или даже превзошел человека. Точно так же трудно предсказать, на что будет способен искусственный интеллект в ближайшем будущем. Например, два десятилетия назад лишь немногие эксперты

могли всерьез думать о том, что автоматизированная система сумеет решать такие задачи, требующие применения интуиции, как более точная категоризация изображений, чем та, на которую способен человек.

Нейронные сети *теоретически* могут обучиться вычислению любой математической функции при условии предоставления им достаточного количества обучающих (тренировочных) данных, и известно, что некоторые их варианты наподобие рекуррентных сетей обладают *свойством полноты по Тьюрингу* (являются тьюринг-полными). Полнота по Тьюрингу означает, что нейронная сеть способна имитировать любой алгоритм обучения, если ей предоставить *достаточный объем тренировочных (обучающих) данных*. Камнем преткновения является тот факт, что объем данных, которые требуются для обучения даже простым задачам, часто оказывается непомерно большим, что приводит к увеличению времени обучения. Например, время обучения распознаванию образов, что для человека является простой задачей, может исчисляться неделями даже на высокопроизводительных системах. Кроме того, существуют проблемы технического характера, обусловленные нестабильностью процесса тренировки нейронных сетей, которые приходится решать даже в наши дни. Тем не менее, если со временем быстрое действие компьютеров, как ожидается, возрастет, а на горизонте уже просматриваются значительно более мощные технологии наподобие квантовых вычислений, то вычислительные проблемы в конечном счете могут оказаться не столь критичными, как нам кажется сейчас.

Несмотря на то что проведение параллелей между нейронными сетями и биологическими системами чрезвычайно обнадеживает и граничит с идеями из области научной фантастики, математическое обоснование нейронных сетей требует выполнения более рутинной работы. Абстракцию нейронной сети можно рассматривать как разновидность модульного подхода к созданию обучающих алгоритмов на основе непрерывной оптимизации в рамках вычислительного графа зависимостей между входом и выходом. Справедливости ради следует отметить, что этот подход не слишком отличается от той работы, которая ведется в традиционной теории управления. В действительности некоторые из методов, применяемых для оптимизации в теории управления, разительно напоминают большинство фундаментальных алгоритмов, используемых в нейронных сетях (и исторически им предшествовали). Однако повышение доступности данных в последние годы наряду с увеличением вычислительных мощностей позволило экспериментировать с более глубокими архитектурами подобных вычислительных графов, чем было возможно ранее. Результирующий успех изменил оценку потенциала глубокого обучения.

Книга имеет следующую структуру.

1. *Основы нейронных сетей*. В главе 1 обсуждаются базовые принципы проектирования нейронных сетей. Суть многих традиционных моделей машинного обучения можно понять, рассматривая их как частные случаи

нейронных сетей. Понимание соотношения между традиционным машинным обучением и нейронными сетями является первым шагом к изучению последних. Имитация различных моделей машинного обучения рассматривается в главе 2. Это даст читателям знание того, как нейронные сети преодолевают ограничения традиционных алгоритмов машинного обучения.

2. *Фундаментальные понятия нейронных сетей.* В то время как в главах 1 и 2 дается общий обзор методов обучения нейронных сетей, более подробная информация о проблемах обучения приведена в главах 3 и 4. В главах 5 и 6 рассмотрены сети радиально-базисных функций (RBF) и ограниченные машины Больцмана.
3. *Дополнительные вопросы нейронных сетей.* Своими недавними успехами глубокое обучение в значительной мере обязано специализированным архитектурам, таким как рекуррентные и сверточные нейронные сети, которые обсуждаются в главах 7 и 8. Главы 9 и 10 посвящены более сложным темам, таким как глубокое обучение с подкреплением, нейронные машины Тьюринга и генеративно-состязательные сети.

Автор позаботился о том, чтобы не упустить из виду некоторые из “забытых” архитектур наподобие RBF-сетей или самоорганизующихся карт Кохонена, поскольку они по-прежнему имеют значительный потенциал во многих задачах. Книга предназначена для студентов старших курсов, исследователей и специалистов-практиков. В конце каждой главы приведены упражнения, решение которых поможет студентам закрепить материал, изученный ими на лекциях. Там, где это возможно, автор обращает особое внимание на прикладные аспекты нейронных сетей, чтобы читатель чувствовал себя увереннее в этой сфере.

В книге для обозначения векторов и многомерных точек данных используются строчные или прописные буквы с чертой над ними, например \vec{X} или \vec{y} . Операция скалярного (точечного) произведения векторов обозначается точкой между ними, например $\vec{X} \cdot \vec{Y}$. Матрицы обозначаются прописными буквами без надчеркивания, например R . По всей книге матрица размером $n \times d$, соответствующая полному обучающему набору данных, обозначается через D , где n — число документов, а d — число измерений. Поэтому индивидуальные точки данных в D являются d -мерными вектор-строками. С другой стороны, векторы с одной составляющей для каждой точки данных обычно являются n -мерными вектор-столбцами. В качестве примера можно привести n -мерный вектор-столбец \vec{y} переменных, включающих n точек данных. Чтобы отличить его от наблюдаемого значения y_i , предсказываемое значение \hat{y}_i обозначается символом циркумфлекса (“крышка”) поверх переменной.



Чару Аггарвал — заслуженный научный сотрудник (DRSM) исследовательского центра IBM имени Томаса Дж. Уотсона в г. Йорктаун Хайтс, штат Нью-Йорк. Базовое образование в области компьютерных наук получил в Индийском технологическом институте в г. Канпур, который окончил в 1993 г., а в 1996 г. получил степень доктора философии в Массачусетском технологическом институте. Вся его деятельность связана с обширными исследо-

ваниями в области интеллектуального анализа данных. Он автор свыше 350 публикаций и более чем 80 зарегистрированных патентов. Выпустил 18 книг, в том числе учебники по анализу данных и рекомендательным системам. Ввиду коммерческой ценности его патентов он трижды получал звание заслуженного изобретателя компании IBM. Чару Аггарвал был удостоен награды *IBM Corporate Award* (2003) за работы по обнаружению биотеррористических угроз в потоках данных, награды *IBM Outstanding Innovation Award* (2008) за научный вклад в технологию защиты конфиденциальности данных, а также двух наград *IBM Outstanding Technical Achievement Award* (2009, 2015) за работы по потокам многомерных данных. Он также лауреат награды *EDBT 2014 Test of Time Award*, присужденной за работы по интеллектуальному анализу данных с сохранением их конфиденциальности. Кроме того, он был удостоен награды *IEEE ICDM Research Contributions Award* (2015), являющейся одной из двух наивысших наград, которые присуждаются за выдающийся вклад в области анализа данных.

Чару Аггарвал был сопредседателем на конференции *IEEE Big Data* (2014) и сопредседателем на конференциях *ACM CIKM* (2015), *IEEE ICDM* (2015) и *ACM KDD* (2016). С 2004 по 2008 годы был заместителем редактора журнала *IEEE Transactions on Knowledge and Data Engineering*. В настоящее время заместитель редактора журналов *IEEE Transactions on Big Data*, *Data Mining and Knowledge Discovery Journal* и *Knowledge and Information Systems Journal*, а также главный редактор журналов *ACM Transactions on Knowledge Discovery from Data* и *ACM SIGKDD Explorations*. Член консультативного комитета по выпуску книжной серии *Lecture Notes on Social Networks* (издательство Springer). Ранее занимал пост вице-президента инициативной группы *SIAM Activity Group on Data Mining*, а в настоящее время — член комитета *SIAM Industry*.

Ждем ваших отзывов!

Вы, читатель этой книги, и есть главный ее критик. Мы ценим ваше мнение и хотим знать, что было сделано нами правильно, что можно было сделать лучше и что еще вы хотели бы увидеть изданным нами. Нам интересны любые ваши замечания в наш адрес.

Мы ждем ваших комментариев и надеемся на них. Вы можете прислать нам электронное письмо либо просто посетить наш сайт и оставить свои замечания там. Одним словом, любым удобным для вас способом дайте нам знать, нравится ли вам эта книга, а также выскажите свое мнение о том, как сделать наши книги более интересными для вас.

Отправляя письмо или сообщение, не забудьте указать название книги и ее авторов, а также свой обратный адрес. Мы внимательно ознакомимся с вашим мнением и обязательно учтем его при отборе и подготовке к изданию новых книг.

Наши электронные адреса:

E-mail: info@dialektika.com

WWW: <http://www.dialektika.com>

Глава 1

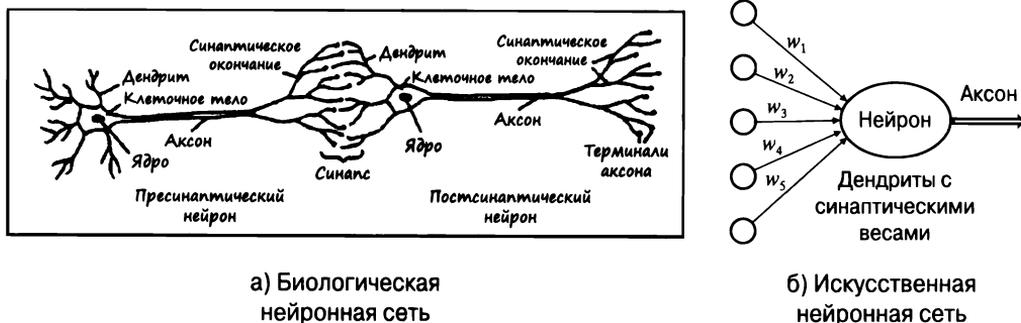
Введение в нейронные сети

Не сотвори машину, выдающую себя за человека.

Фрэнк Герберт

1.1. Введение

Термин *искусственные нейронные сети* охватывает совокупность популярных методов машинного обучения, имитирующих механизмы обучения, которые действуют в биологических организмах. Нервная система человека содержит *нейроны* — специфические клетки, предназначенные для приема, обработки, хранения и передачи поступающей информации с помощью электрических и химических сигналов. Нейроны могут соединяться друг с другом посредством *аксонов* и *дендритов*, а структуры, связывающие дендриты одного нейрона с аксонами других нейронов, называются *синапсами* (рис. 1.1, а). Сила синаптических связей часто изменяется в ответ на внешнее возбуждение. Именно эти изменения лежат в основе механизма обучения живых организмов.



а) Биологическая нейронная сеть

б) Искусственная нейронная сеть

Рис. 1.1. Синаптические связи между нейронами; изображение слева взято из курса “The Brain: Understanding Neurobiology Through the Study of Addiction” [598]

Описанный биологический механизм имитируют *искусственные* нейронные сети, которые содержат вычислительные элементы, также получившие название *нейроны*. На протяжении всей книги мы будем использовать термин “нейронные сети”, подразумевая искусственные нейронные сети, а не биологические. Вычислительные элементы соединяются между собой посредством связей, имеющих определенные веса, которые играют ту же роль, что и силы синаптических связей в биологических организмах. Каждый вход нейрона масштабируется в соответствии с его весом, что влияет на результат вычисления функции в каждом элементе (рис. 1.1, б). Искусственная нейронная сеть вычисляет нелинейную функцию активации от взвешенной суммы входов, распространяя вычисленные значения от входных нейронов к выходным, причем веса играют роль промежуточных параметров. Процесс обучения происходит путем изменения весов связей, соединяющих нейроны. Точно так же, как для обучения биологических систем нужны внешние стимулы, в искусственных нейронных сетях внешним стимулом являются тренировочные данные, содержащие пары значений “вход — выход” той функции, которой должна обучиться сеть. *Тренировочными (учебными) данными* могут быть пиксельные представления изображений (вход) и их аннотационные метки (например, *морковь, банан*), представляющие ожидаемый выход. Нейронная сеть использует тренировочные данные, передаваемые ей в качестве входных обучающих примеров, для предсказания (прогнозирования) выходных меток. Тренировочные данные обеспечивают обратную связь, позволяющую корректировать веса в нейронной сети в зависимости от того, насколько хорошо выход, предсказанный для конкретного входа (например, вероятность того, что на картинке изображена морковь), соответствует ожидаемой аннотационной выходной метке, определенной в составе тренировочных данных. Ошибки, допущенные нейронной сетью в процессе вычисления функции, можно уподобить неприятным ощущениям, испытываемым биологическим организмом, которые управляют изменением силы синаптических связей. Аналогично этому в нейронной сети веса связей между нейронами подстраиваются так, чтобы уменьшить ошибку предсказания. Целью изменения весов является изменение вычисляемой функции в направлении, обеспечивающем получение более точных предсказаний на будущих итерациях. Настройка весов, уменьшающая ошибку вычислений для конкретного входного примера, осуществляется в соответствии с определенной математически обоснованной процедурой. Благодаря согласованной настройке весов связей между нейронами на множестве пар “вход — выход” функция, вычисляемая нейронной сетью, постепенно улучшается настолько, что выдает все более точные предсказания. Поэтому, если для тренировки нейронной сети используется набор различных изображений бананов, то она в конечном счете

приобретает способность правильно распознавать бананы на изображениях, которые до этого ей не встречались. Способность точно вычислять функции незнакомых входов после прохождения тренировки (обучения) на конечном наборе пар “вход — выход” называется *обобщением модели* (model generalization). Именно способность нейронной сети к обобщению знаний, приобретенных с помощью предоставленных ей тренировочных данных, на незнакомые примеры, является основным фактором, определяющим полезность любой модели машинного обучения.

Попытки сравнения моделей искусственных нейронных сетей с биологическими моделями нередко подвергаются критике на том основании, что они являются не более чем жалким подобием действительных процессов, происходящих в человеческом мозге. Тем не менее принципы нейронаук часто используются в качестве ориентиров при проектировании архитектур нейронных сетей. Другая точка зрения состоит в том, что нейронные сети строятся как высокоуровневые абстракции классических моделей, широко применяемых в машинном обучении. Действительно, идея базовых вычислительных элементов в нейронных сетях была подсказана традиционными алгоритмами машинного обучения, такими как *регрессия по методу наименьших квадратов* и *логистическая регрессия*. Нейронные сети обретают свою мощь благодаря объединению многих вычислительных элементов и совместному обучению их весов для минимизации ошибки предсказания. С этой точки зрения нейронную сеть можно рассматривать как *вычислительный граф* элементов, повышение вычислительной мощности которых достигается за счет их объединения определенными способами. Если нейронная сеть используется в своей простейшей форме, без организации взаимодействия многочисленных вычислительных элементов между собой, то алгоритм обучения часто сводится к классическим моделям машинного обучения (об этом мы поговорим в главе 2). Реальное превосходство нейронной модели по сравнению с классическими подходами раскрывается в полную мощь лишь в случае объединения разрозненных вычислительных элементов в единое целое, когда веса моделей согласованно обучаются с учетом зависимостей между ними. Комбинируя множество вычислительных элементов, можно расширить возможности модели до уровня, обеспечивающего обучение более сложным функциям, чем те, которые свойственны элементарным моделям простого машинного обучения (рис. 1.2). Способы объединения этих элементов также оказывают влияние на вычислительные возможности архитектуры и требуют тщательного изучения и адекватной оценки аналитиков. К тому же для обучения возросшего количества весов в таких расширенных вычислительных графах требуются достаточно большие объемы тренировочных данных.

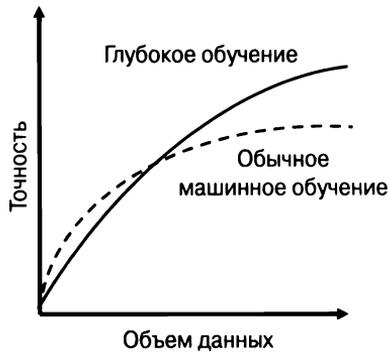


Рис. 1.2. Иллюстративное сравнение точности типичного алгоритма машинного обучения и точности большой нейронной сети. Преимущества методов глубокого обучения по сравнению с обычными методами проявляются главным образом при наличии достаточно больших объемов данных и вычислительных мощностей. В последние годы доступ к этим ресурсам значительно упростился, что привело к “Кембрийскому взрыву” в глубоком обучении

1.1.1. Человек и компьютер: расширение пределов возможностей искусственного интеллекта

Люди и компьютеры в силу самой своей природы приспособлены для решения задач разного типа. Например, для компьютера извлечение кубического корня из большого числа не составит труда, тогда как человеку справиться с этим нелегко. С другой стороны, такая задача, как распознавание объектов на изображении, для человека — сущий пустяк, тогда как для алгоритмов машинного обучения она долгое время оставалась трудным орешком. Лишь в последние годы при решении некоторых задач такого типа глубокое обучение продемонстрировало точность, недоступную для человека. Действительно, последние результаты, демонстрирующие превосходство алгоритмов глубокого обучения над человеком [184] при решении специализированных задач по распознаванию образов, еще каких-то десять лет назад считались бы невероятными большинством специалистов в области компьютерного зрения.

Многие архитектуры глубокого обучения, продемонстрировавшие столь необычайную эффективность, создавались вовсе не путем неструктурированного объединения вычислительных элементов. Исключительная эффективность *глубоких нейронных сетей* отражает тот факт, что истоки возможностей биологических нейронных сетей также кроются в их глубине. Более того, способы связывания биологических нейронных сетей между собой нам еще не до конца понятны. В тех немногих случаях, когда нам удалось в какой-то степени достигнуть понимания биологических структур, применение их в качестве

прототипов при проектировании искусственных нейронных сетей обеспечило значительный прорыв в этом направлении. В качестве классического примера архитектуры такого типа можно привести использование *сверточных нейронных сетей* для распознавания образов. Их архитектура была подсказана результатами экспериментов по исследованию организации нейронов в зрительной коре кошек, полученными Хубелем и Визелем в 1959 году [212]. Предшественником сверточной нейронной сети был *неокогнитрон* [127], архитектура которого основана непосредственно на этих результатах.

Структура нейронных связей человека совершенствовалась в процессе эволюции на протяжении миллионов лет, обеспечивая выживание вида. Выживание тесно связано с нашей способностью объединять чувства и интуицию в единое целое способом, который в настоящее время невозможно реализовать с помощью машин. Биологические нейронауки [232] все еще находятся на младенческой стадии развития, и мы располагаем лишь ограниченным набором сведений относительно того, как на самом деле работает человеческий мозг. Поэтому есть все основания полагать, что когда мы будем знать об этом больше, успех сверточных нейронных сетей, идея которых была подсказана биологическими моделями, будет повторен в других конфигурациях искусственных нейронных сетей [176]. Ключевым преимуществом нейронных сетей по сравнению с традиционным машинным обучением является то, что они обеспечивают возможность высокоуровневого абстрактного выражения семантики внутренних связей между данными посредством выбора вариантов архитектурных решений в вычислительном графе. Другое преимущество нейронных сетей заключается в том, что они позволяют очень легко регулировать сложность модели, добавляя или удаляя нейроны из архитектуры в соответствии с доступностью тренировочных данных или вычислительных ресурсов. Значительная доля недавних успехов нейронных сетей объясняется облегчением доступа к данным и ростом вычислительной мощности современных компьютеров, что позволило преодолеть ограничения традиционных алгоритмов машинного обучения, которые не в состоянии в полной мере воспользоваться всеми преимуществами доступных возможностей (см. рис. 1.2). В некоторых случаях традиционные алгоритмы машинного обучения работают лучше для небольших наборов данных в силу наличия большего количества вариантов выбора, более простых возможностей интерпретации моделей и распространенной тенденции вручную настраивать интерпретируемые признаки, вбирающие в себя специфические для данной предметной области внутренние закономерности. При ограниченных объемах данных лучшая из широкого разнообразия моделей машинного обучения обычно будет работать лучше, чем одиночный класс моделей (типа нейронных сетей). В этом и кроется одна из причин того, почему потенциал нейронных сетей в первые годы не был по-настоящему осознан.

Начало эпохи больших данных стало возможным благодаря прогрессу в технологиях сбора данных. В настоящее время практически вся информация о совершаемых нами действиях, включая покупку товаров, звонки по телефону или посещение сайтов, кем-то собирается и записывается. К тому же разработка мощных графических процессоров сделала возможным резкое повышение эффективности обработки огромных объемов данных. Эти достижения в значительной мере объясняют недавние успехи глубокого обучения, достигнутые с использованием алгоритмов, которые представляют собой лишь незначительно видоизмененные версии алгоритмов, доступных уже два десятилетия тому назад. Кроме того, эти недавние модификации алгоритмов стали возможными благодаря возросшей скорости вычислений, что создало благоприятные условия для тестирования моделей (и последующей корректировки алгоритмов). Если для тестирования алгоритма требуется месяц, то на одной аппаратной платформе за год может быть протестировано не более двенадцати его вариаций. Ситуации подобного рода исторически ограничивали интенсивность экспериментов, необходимых для настройки алгоритмов обучения нейронных сетей. Быстрые темпы прогресса во всем, что касается доступности данных, скорости вычислений и возможностей проведения экспериментов, усилили оптимизм в отношении будущих перспектив глубокого обучения. Ожидается, что к концу нынешнего столетия мощность компьютеров возрастет до такой степени, что станет возможным тренировать нейронные сети с количеством нейронов, сопоставимым с количеством нейронов в человеческом мозге. И хотя очень трудно предсказать истинные возможности нейронных сетей, которыми они будут обладать к тому времени, опыт разработок в области компьютерного зрения убеждает нас в том, что мы должны быть готовы к любым неожиданностям.

Структура главы

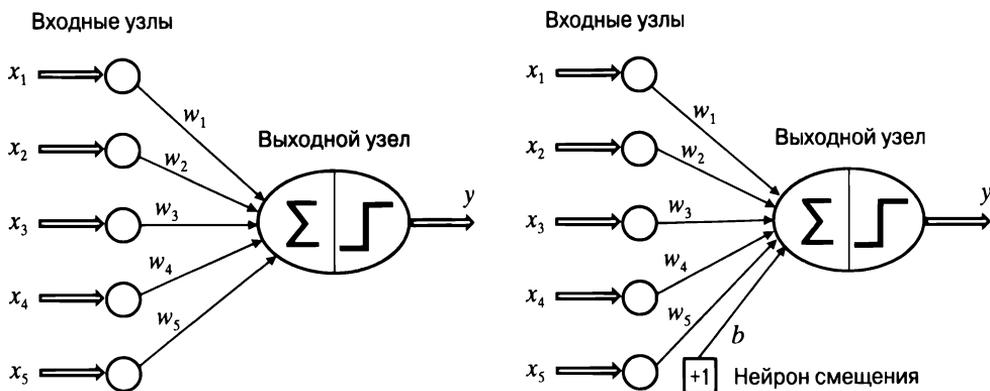
В следующем разделе вы познакомитесь с принципами работы одно- и многослойных сетей. Будут обсуждаться различные типы функций активации, выходных узлов и функций потерь. Рассмотрению алгоритма обратного распространения ошибки посвящен раздел 1.3. Практические аспекты тренировки нейронных сетей обсуждаются в разделе 1.4. Ключевые моменты, касающиеся того, каким образом нейронные сети обретают эффективность за счет выбора функций активации, описаны в разделе 1.5. Типичные архитектуры, используемые при проектировании нейронных сетей, обсуждаются в разделе 1.6. Дополнительные вопросы глубокого обучения рассмотрены в разделе 1.7, а некоторые показательные эталонные тесты — в разделе 1.8. Резюме главы приведено в разделе 1.9.

1.2. Базовая архитектура нейронных сетей

В этом разделе рассмотрены одно- и многослойные нейронные сети. В однослойной сети набор входов непосредственно транслируется на выход путем использования обобщенного варианта линейной функции. Этот простой вариант реализации нейронной сети также называют *перцептроном*. В многослойных нейронных сетях нейроны располагаются слоями, причем между входным и выходным слоями располагается группа так называемых *скрытых слоев*. Нейронные сети с такой многослойной архитектурой также называют *сетями прямого распространения* (feed-forward network). В этом разделе мы обсудим как однослойные, так и многослойные сети.

1.2.1. Одиночный вычислительный слой: перцептрон

Простейшая нейронная сеть, получившая название *перцептрон*, состоит из одного входного слоя и одного выходного. Базовая структура перцептрона приведена на рис. 1.3, а. Рассмотрим ситуацию, когда каждый тренировочный пример имеет вид (\bar{X}, y) , где каждый вектор $\bar{X} = [x_1 \dots x_d]$ содержит d переменных признаков, а $y \in \{-1, +1\}$ содержит наблюдаемое значение переменной бинарного класса. “Наблюдаемым” мы будем называть значение, предоставляемое нейронной сети в составе тренировочных данных, и наша цель заключается в том, чтобы научиться предсказывать значения переменной класса для случаев, в которых за ней не осуществлялось наблюдение. Например, в приложении, обнаруживающем поддельные кредитные карты, признаками могут служить различные свойства набора операций с кредитными картами (такие, как объемы и частота операций), а переменная класса может указывать на то, является ли



а) Перцептрон без смещения

б) Перцептрон со смещением

Рис. 1.3. Базовая архитектура перцептрона

данный набор операций законным или незаконным. Ясно, что в приложениях такого типа необходимо располагать информацией о предыдущих случаях, в которых наблюдалась данная переменная класса, и о других (текущих) случаях, в которых переменная класса еще не наблюдалась, но должна быть предсказана.

Входной слой содержит d узлов, которые передают выходному узлу d признаков $\bar{X} = [x_1 \dots x_d]$ с весовыми коэффициентами $\bar{W} = [w_1 \dots w_d]$. Сам входной слой не выполняет сколь-нибудь существенных вычислений. Линейная функция $\bar{W} \cdot \bar{X} = \sum_{i=1}^d w_i x_i$ вычисляется на выходном узле. Впоследствии знак этого реального значения используется для предсказания значения зависимой переменной \bar{X} . Поэтому предсказанное значение \hat{y} определяется следующей формулой:

$$\hat{y} = \text{sign}\{\bar{W} \cdot \bar{X}\} = \text{sign}\left\{\sum_{j=1}^d w_j x_j\right\}. \quad (1.1)$$

Функция *sign* переводит реальное значение в $+1$ или -1 , что вполне подходит для бинарной классификации. Обратите внимание на символ циркумфлекса над переменной y , указывающий на то, что это предсказанное, а не наблюдаемое значение. Поэтому ошибка предсказания $E(\bar{X}) = y - \hat{y}$ может иметь одно из значений: $\{-2, 0, +2\}$. В тех случаях, когда ошибка $E(\bar{X})$ имеет ненулевое значение, веса нейронной сети должны быть оптимизированы за счет обновления в направлении, противоположном направлению градиента ошибки. Как будет показано далее, этот процесс аналогичен тому, который используется в различных типах линейных моделей в машинном обучении. Несмотря на сходство перцептрона с традиционными моделями машинного обучения, его весьма полезно интерпретировать как вычислительный элемент, поскольку это позволяет объединять множество таких элементов для создания гораздо более мощных моделей, чем те, которые доступны в традиционном машинном обучении.

Архитектура перцептрона представлена на рис. 1.3, *a*, на котором одиночный входной слой передает признаки выходному узлу. Связи, соединяющие вход с выходом, характеризуются весами $w_1 \dots w_d$, после умножения на которые признаки суммируются на выходном узле. Далее функция *sign* преобразует агрегированное значение в метку класса. Функция *sign* играет роль *функции активации*. Для имитации различных типов моделей, используемых в машинном обучении, можно выбирать различные функции активации, приводящие к таким моделям, как *регрессия по методу наименьших квадратов с числовыми целевыми значениями*, *метод опорных векторов* или *классификатор на основе логистической регрессии*. Большинство базовых моделей машинного обучения можно легко представить как различные архитектуры простой нейронной сети. Моделирование всевозможных методов традиционного машинного обучения как нейронных архитектур — занятие весьма полезное, поскольку это позволяет прояснить кар-

тину того, как глубокое обучение обобщает традиционное машинное обучение. Эта точка зрения детально исследуется в главе 2. Следует обратить внимание на то, что в действительности перцептрон содержит два слоя, хотя входной слой не выполняет никаких вычислений и лишь передает значения признаков. Поэтому при подсчете количества слоев в нейронной сети входной слой не учитывается. Поскольку перцептрон содержит только *один* вычислительный слой, он считается однослойной сетью.

Во многих случаях предсказание содержит инвариантную часть, так называемое *смещение* (*bias*). Рассмотрим, например, эксперимент, в котором переменные признаков центрированы на среднем значении, но среднее значение предсказания бинарных классов из набора $\{-1, +1\}$ не равно нулю. Обычно такое встречается в ситуациях, в которых распределение бинарных классов в значительной степени разбалансировано. Тогда вышеупомянутого подхода оказывается недостаточно для выполнения предсказаний. В этом случае мы должны включать в рассмотрение дополнительную переменную смещения b , вбирающую в себя инвариантную часть предсказания:

$$\hat{y} = \text{sign}\{\bar{W} \cdot \bar{X} + b\} = \text{sign}\left\{\sum_{j=1}^d w_j x_j + b\right\}. \quad (1.2)$$

Смещение можно включить в качестве веса связи, используя *нейрон смещения*. Это достигается за счет добавления нейрона, который всегда передает значение 1 выходному узлу. Тогда вес связи, соединяющей нейрон смещения с выходным узлом, представляет собой переменную смещения. Пример нейрона смещения представлен на рис. 1.3, б. Другой подход, который хорошо работает в однослойных архитектурах, основан на *трюке с конструированием признаков*, предполагающем создание дополнительного признака с постоянным значением 1. Коэффициент при этом признаке представляет смещение, и тогда можно работать с уравнением 1.1. В книге мы не будем использовать смещения в явном виде, поскольку они могут быть учтены путем добавления нейронов смещения. При этом детали тренировочных алгоритмов остаются прежними за счет того, что мы рассматриваем нейрон смещения как любой другой нейрон с фиксированным значением активации, равным 1. Поэтому далее мы будем использовать для предсказаний уравнение 1.1, в котором смещения не фигурируют в явном виде.

В те времена, когда Розенблатт предложил алгоритм перцептрона [405], оптимизация весов осуществлялась эвристическими методами на аппаратном уровне и не представлялась в терминах формализованных задач оптимизации в машинном обучении (как это общепринято в наши дни). Однако конечной целью всегда ставилась минимизация ошибки предсказания, даже если эта задача и не формулировалась на формальном уровне. Поэтому эвристическая модель

перцептрона была спроектирована так, чтобы минимизировать число случаев ошибочной классификации, и были предложены способы доказательства сходимости процесса, гарантирующие корректность алгоритма обучения в упрощенных постановках экспериментов. Таким образом, мы все же можем сформулировать (обоснованную эвристически) цель алгоритма перцептрона в форме метода наименьших квадратов по отношению ко всем тренировочным примерам в наборе данных \mathcal{D} , содержащем пары “признак — метка”:

$$\text{Минимизировать}_{\bar{w}} L = \sum_{(\bar{X}, y) \in \mathcal{D}} (y - \hat{y})^2 = \sum_{(\bar{X}, y) \in \mathcal{D}} (y - \text{sign}\{\bar{W} \cdot \bar{X}\})^2.$$

Целевые функции минимизации такого типа также называют *функциями потерь* (loss functions). Как будет показано далее, почти все алгоритмы обучения нейтральной сети формулируются с использованием понятия функции потерь. В главе 2 будет показано, что эта функция во многом подобна функции, используемой в регрессии по методу наименьших квадратов. Однако последняя определена для целевых переменных, имеющих непрерывные значения, и соответствующие потери являются гладкой и непрерывной функцией своих переменных. С другой стороны, в случае нашей целевой функции в форме метода наименьших квадратов функция *sign* не является дифференцируемой и имеет скачки в некоторых точках. К тому же она принимает постоянные значения на протяжении больших участков области изменения переменных и поэтому имеет нулевой градиент в тех точках, где она дифференцируема. В результате поверхность функции потерь имеет ступенчатую форму, непригодную для применения метода градиентного спуска. Алгоритм перцептрона использует (неявно) гладкую аппроксимацию градиента этой целевой функции по отношению к каждому примеру:

$$\nabla L_{\text{сглаженная}} = \sum_{(\bar{X}, y) \in \mathcal{D}} (y - \hat{y}) \bar{X}. \quad (1.3)$$

Обратите внимание на то, что вышеприведенный градиент не является истинным градиентом ступенчатой поверхности (эвристической) целевой функции, которая не предоставляет пригодных для использования градиентов. Поэтому “лестница” сглаживается и принимает форму наклонной поверхности, определяемой *критерием перцептрона*. Свойства критерия перцептрона будут описаны в разделе 1.2.1.1. Следует отметить, что такие понятия, как “критерий перцептрона”, были предложены уже после выхода статьи Розенблатта [405] для объяснения эвристики шагов градиентного спуска. А пока что мы будем полагать, что алгоритм перцептрона оптимизирует некоторую неизвестную гладкую функцию с помощью градиентного спуска.

Несмотря на то что вышеприведенная целевая функция определена на всем диапазоне тренировочных данных, в процессе создания предсказания \hat{y} обучающим алгоритмом нейронных сетей каждый пример входных данных \bar{X} передается сети поочередно (или в составе небольших пакетов). После этого, исходя из значения ошибки $E(\bar{X}) = (y - \hat{y})$, выполняется обновление весов. В частности, при передаче сети точки данных \bar{X} вектор весов \bar{W} обновляется в соответствии с такой формулой:

$$\bar{W} \leftarrow \bar{W} + \alpha(y - \hat{y})\bar{X}. \quad (1.4)$$

Параметр α регулирует *скорость обучения* (learning rate) нейронной сети. Алгоритм перцептрона повторяет циклические проходы по всем тренировочным примерам в случайном порядке и итеративно настраивает веса до тех пор, пока не будет достигнута сходимость. Прохождение одиночной точки данных может циклически повторяться множество раз. Каждый такой цикл называется *эпохой*. Формулу обновления весов по методу градиентного спуска можно переписать в терминах ошибки $E(\bar{X}) = (y - \hat{y})$ в следующем виде:

$$\bar{W} \leftarrow \bar{W} + \alpha E(\bar{X})\bar{X}. \quad (1.5)$$

Базовый алгоритм перцептрона можно рассматривать как метод *стохастического градиентного спуска* (stochastic gradient-descent), который неявно минимизирует квадрат ошибки предсказания, выполняя обновления методом градиентного спуска по отношению к случайно выбираемым тренировочным точкам. При этом суть основного допущения заключается в том, что нейронная сеть в процессе тренировки циклически обходит точки в случайном порядке и изменяет веса, стремясь уменьшить ошибку предсказания в данной точке. Как следует из уравнения 1.5, ненулевые обновления весов возможны только в том случае, если $y \neq \hat{y}$, т.е. только тогда, когда предсказание было выполнено с ошибкой. В методе *мини-пакетного стохастического градиентного спуска* вышеупомянутые обновления (1.5) реализуются на случайно выбранном поднаборе тренировочных точек S :

$$\bar{W} \leftarrow \bar{W} + \alpha \sum_{\bar{X} \in S} E(\bar{X})\bar{X}. \quad (1.6)$$

Преимущества использования мини-пакетного стохастического градиентного спуска обсуждаются в разделе 3.2.8. Интересной особенностью перцептрона является то, что скорость обучения α может быть установлена равной 1, поскольку она масштабирует лишь веса.

Модель, предложенная перцептроном, относится к *линейному типу*, для которого уравнение $\bar{W} \cdot \bar{X} = 0$ определяет линейную гиперплоскость. Здесь $\bar{W} = (w_1 \dots w_d)$ — d -мерный вектор, направленный вдоль нормали к гиперплоскости.

Кроме того, значение $\bar{W} \cdot \bar{X}$ положительно для значений \bar{X} по одну сторону от гиперплоскости и отрицательно для значений \bar{X} по другую ее сторону. Такой тип моделей особенно хорошо работает в случае *линейно разделимых* данных. Примеры линейно разделимых и линейно неразделимых данных приведены на рис. 1.4.

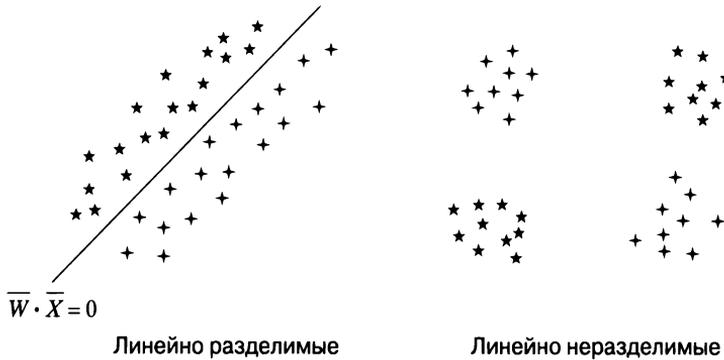


Рис. 1.4. Примеры линейно разделимых и неразделимых данных двух классов

Алгоритм перцептрона хорошо работает при классификации наборов данных, подобных тому, который представлен на рис. 1.4, *слева*, когда данные поддаются *линейному разделению*. С другой стороны, он плохо справляется с этой задачей в случае наборов данных вроде того, который представлен на рис. 1.4, *справа*. Этот пример демонстрирует присущие перцептрону ограничения, что заставляет прибегать к более сложным нейронным архитектурам.

Поскольку оригинальный механизм перцептрона был предложен в качестве метода эвристической минимизации ошибок классификации, было особенно важно показать, что данный алгоритм сходится к разумным решениям в некоторых специальных случаях. В этом контексте было доказано [405], что условием сходимости алгоритма перцептрона к нулевой ошибке на тренировочных данных является линейная разделимость этих данных. Однако алгоритм перцептрона не гарантирует сходимость для примеров, в которых данные не являются линейно разделимыми. По причинам, которые обсуждаются в следующем разделе, в случае данных, не являющихся линейно разделимыми, алгоритм перцептрона иногда может приводить к очень плохим решениям (по сравнению со многими другими алгоритмами обучения).

1.2.1.1. Какой целевой функции соответствует оптимизация перцептрона

Как уже упоминалось, в оригинальной статье Розенблатта, в которой он описал перцептрон [405], функция потерь формально не вводилась. В те годы модели реализовывались аппаратными средствами. Оригинальный *перцептрон Mark I*, реализованный на компьютере, название которого и дало ему имя, изначально задумывался как машина, а не как алгоритм, в связи с чем для него было специально создано соответствующее электронное устройство (рис. 1.5).

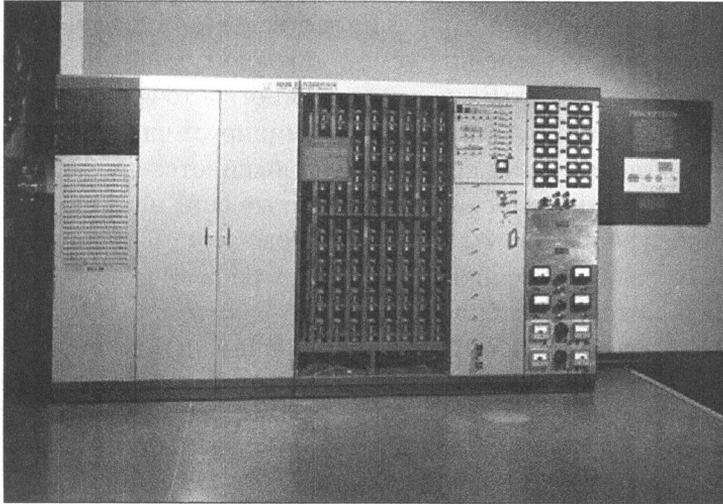


Рис. 1.5. Алгоритм перцептрона первоначально был реализован в виде электронного устройства; на приведенной фотографии изображен компьютер Mark I, созданный в 1958 году специально для реализации перцептрона (публикуется с разрешения Смитсоновского института)

Общая задача заключалась в минимизации количества ошибок классификации с помощью эвристического процесса обновления (реализованного аппаратными средствами), который изменялся в “правильном” направлении, если возникала ошибка. Это эвристическое обновление очень напоминало градиентный спуск, но оно получалось не по методу градиентного спуска. В алгоритмической постановке задачи градиентный спуск определен лишь для гладких функций потерь, тогда как аппаратный подход основывался на эвристических принципах и был рассчитан на *бинарные выходы*. Многие из принципов построения аппаратных моделей с бинарными выходами были заимствованы из модели нейрона Маккаллока — Питтса [321]. К сожалению, оптимизация по непрерывным переменным неприменима к бинарным сигналам.

Можем ли мы найти такую гладкую функцию потерь, чтобы ее градиент описывал обновление перцептрона? Количество ошибок классификации для бинарной задачи может быть записано в форме *двоичной функции потерь* (0/1 loss function) для тренировочной точки данных (\bar{X}_i, y_i) в следующем виде:

$$L_i^{(0/1)} = \frac{1}{2} (y_i - \text{sign}\{\bar{W} \cdot \bar{X}_i\})^2 = 1 - y_i \cdot \text{sign}\{\bar{W} \cdot \bar{X}_i\}. \quad (1.7)$$

Здесь правая часть целевой функции упрощена за счет замены значением 1 как члена y_i^2 , так и члена $\text{sign}\{\bar{W} \cdot \bar{X}_i\}^2$, поскольку они получаются возведением в квадрат значения, извлекаемого из множества $\{-1, +1\}$. Однако эта целевая функция не является дифференцируемой, поскольку она имеет ступенчатую форму, особенно после ее суммирования по многим точкам. Обратите внимание на то, что в двоичной функции потерь доминирует член $-y_i \cdot \text{sign}\{\bar{W} \cdot \bar{X}_i\}$, в котором функция *sign* является источником большинства проблем, связанных с недифференцируемостью. Поскольку нейронные сети определяются оптимизацией на основе градиента, мы должны определить гладкую целевую функцию, которая отвечала бы за обновления перцептрона. Можно показать [41], что обновления перцептрона неявно оптимизируют *критерий перцептрона*. Эта целевая функция получается опусканием функции *sign* в приведенном выше выражении для двоичной функции потерь и подстановкой нулевого значения вместо отрицательных значений, чтобы все корректные значения обрабатывались унифицированным способом и без потерь:

$$L_i = \max\{-y_i(\bar{W} \cdot \bar{X}_i), 0\}. \quad (1.8)$$

Читателям предлагается самостоятельно использовать дифференциальное исчисление для проверки того, что градиент этой сглаженной целевой функции приводит к обновлению перцептрона, и это обновление по сути описывается формулой $\bar{W} \leftarrow \bar{W} - \alpha \nabla_w L_i$. Видоизмененную функцию потерь, обеспечивающую возможность вычисления градиента недифференцируемой функции, также называют *сглаженной суррогатной функцией потерь* (smoothed surrogate loss function). Подобные функции используются почти всеми непрерывными методами обучения на основе оптимизации с дискретными выходами (такими, как метки классов).

Несмотря на то что вышеупомянутый критерий перцептрона был получен методами обратного инжиниринга, т.е. реконструирован по обновлениям перцептрона, природа этой функции потерь обнаруживает некоторые слабые стороны обновлений в оригинальном алгоритме. Критерий перцептрона имеет интересную особенность: можно установить \bar{W} равным нулевому вектору, *независимо от тренировочного набора*, и получить оптимальное значение потерь, равное нулю. Несмотря на это, обновления сходятся к отчетливому разделителю между двумя классами в случае их линейной делимости, и в конце

концов разделитель классов также обеспечивает нулевое значение потерь. Однако в случае данных, не разделяемых линейно, такое поведение становится достаточно произвольным, а иногда результирующее решение вообще не является хорошим приближением к разделителю классов. Непосредственная чувствительность потерь к величине вектора весов может размыть цель разделения классов. Возможны даже ситуации, когда количество случаев ошибочной классификации значительно увеличивается, хотя потери при этом уменьшаются. Это является ярким примером того, как суррогатные функции потерь иногда не в состоянии обеспечить решение поставленных перед ними задач. В силу указанных причин описанный подход отличается нестабильностью и может приводить к решениям, качество которых меняется в широких пределах.

В связи с этим были предложены различные варианты алгоритма обучения для неразделимых данных, при этом естественный подход состоит в том, чтобы всегда отслеживать, какое решение является наилучшим в смысле наименьшего количества случаев ошибочной классификации [128]. Такой подход, когда наилучшее решение всегда держится наготове, получил название *карманного алгоритма* (pocket algorithm). Другой в высшей степени эффективный вариант, включающий понятие *зазоров* для функции потерь, приводит к созданию алгоритма, *идентичного линейному методу опорных векторов* (Support Vector Machine — SVM). По этой причине линейный метод опорных векторов также называют *перцептроном оптимальной стабильности*.

1.2.1.2. Взаимосвязь с методом опорных векторов

Критерий перцептрона — это смещенная версия функции *кусочно-линейных потерь* (hinge-loss), используемой в методе опорных векторов (глава 2). Кусочно-линейные потери еще более похожи на двоичный критерий потерь (см. уравнение 1.7) и определяются следующей формулой:

$$L_i^{svm} = \max\{1 - y_i(\bar{W} \cdot \bar{X}_i), 0\}. \quad (1.9)$$

Обратите внимание на отсутствие постоянного члена 1 в правой части варианта уравнения 1.7, соответствующего перцептрону, тогда как в случае кусочно-линейных потерь данная константа включается в максимизируемую функцию. Это изменение не сказывается на алгебраическом выражении для градиента, но влияет на принятие решений о том, какие точки не вносят вклад в потери и не должны обновляться. Связь между критерием перцептрона и кусочно-линейными потерями показана на рис. 1.6. Это сходство становится особенно очевидным, если переписать обновления перцептрона, определяемые уравнением 1.6, в следующем виде:

$$\bar{W} \leftarrow \bar{W} + \alpha \sum_{(\bar{x}, y) \in S^+} y \bar{x}. \quad (1.10)$$

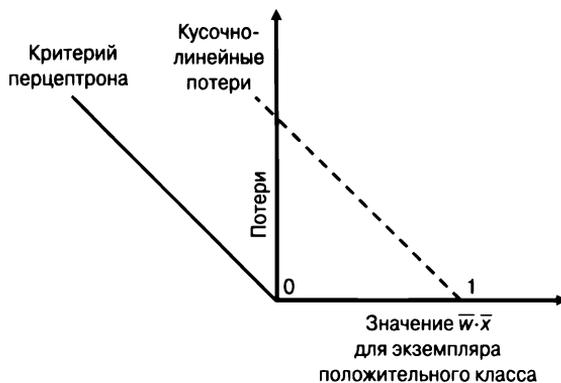


Рис. 1.6. Критерий перцептрона и кусочно-линейные потери

Здесь S^+ определен как набор всех неправильно классифицированных тренировочных точек $\bar{X} \in S$, которые удовлетворяют условию $y(\bar{W} \cdot \bar{X}) < 0$. Очевидно, что это обновление несколько отличается от обновления для перцептрона, поскольку перцептрон использует для обновления ошибку $E(\bar{X})$, которая в вышеприведенном уравнении заменяется на y . Здесь ключевым является то обстоятельство, что (целочисленная) ошибка $E(\bar{X}) = (y - \text{sign}\{\bar{W} \cdot \bar{X}\}) \in \{-2, +2\}$ никогда не может иметь нулевое значение для ошибочно классифицированных точек в S^+ . Поэтому для *ошибочно классифицированных точек* $E(\bar{X}) = 2y$, и тогда, предварительно включив множитель 2 в скорость обучения, $E(\bar{X})$ в обновлениях можно заменить на y . Это обновление идентично тому, которое применялось в первоначальном алгоритме метода опорных векторов (SVM) [448], за исключением того, что в перцептроне обновления вычисляются только для ошибочно классифицированных точек, тогда как SVM дополнительно учитывает *зазоры* для корректных точек вблизи границ принятия решений относительно обновлений. Обратите внимание на то, что SVM использует для определения S^+ условие $y(\bar{W} \cdot \bar{X}) < 1$ (вместо $y(\bar{W} \cdot \bar{X}) < 0$) — одно из основных различий между этими двумя алгоритмами. Отсюда следует, что перцептрон по сути не слишком отличается от таких хорошо известных алгоритмов, как метод опорных векторов, хотя в его основе и лежат другие предпосылки. Фройнд и Шапир предоставили отличное объяснение роли зазоров в улучшении стабильности перцептрона и его связи с методом опорных векторов [123]. Получается, что многие традиционные модели машинного обучения можно рассматривать как незначительные вариации мелких нейронных архитектур, таких как перцептрон. Связь между классическими моделями машинного обучения и мелкими нейронными сетями подробно описана в главе 2.

1.2.1.3. Выбор функции активации и функции потерь

Выбор функции активации — критическая часть процесса проектирования нейронной сети. В случае перцептрона выбор функции *sign* в качестве функции активации обусловлен тем фактом, что предсказания касаются бинарных меток классов. Однако возможны ситуации другого типа, требующие предсказания других целевых переменных. Если, скажем, предсказываемая целевая переменная имеет вещественные значения, то имеет смысл использовать тождественную функцию активации, и в этом случае результирующий алгоритм совпадает с алгоритмом регрессии по методу наименьших квадратов. Если желательно предсказывать вероятность бинарного класса, то для активации выходного узла целесообразно использовать *сигмоиду*, чтобы прогнозное значение \hat{y} указывало на вероятность того, что наблюдаемое значение y зависимой переменной равно 1. Исходя из предположения, что y кодируется из множества значений $\{-1, 1\}$, в качестве функции потерь используется отрицательный логарифм $|y/2 - 0,5 + \hat{y}|$. Если \hat{y} — это вероятность того, что y равен 1, то $|y/2 - 0,5 + \hat{y}|$ — это вероятность предсказания корректного значения. В том, что это действительно так, можно легко убедиться, проверив два случая, когда y равно 0 или 1. Можно показать, что эта функция потерь представляет отрицательное логарифмическое правдоподобие тренировочных данных (раздел 2.2.3). Нелинейность функций активации приобретает особое значение при переходе от однослойного перцептрона к многослойным архитектурам, о которых речь пойдет далее. В различных слоях могут использоваться различные типы нелинейных функций, такие как *знаковая* (сигнум), *сигмоида* и *гиперболический тангенс*. Мы будем использовать для функции активации обозначение Φ :

$$\hat{y} = \Phi(\bar{W} \cdot \bar{X}). \quad (1.11)$$

Таким образом, в действительности нейрон вычисляет в пределах узла две функции, и именно поэтому наряду с символом суммирования S мы включаем в нейрон символ функции активации Φ . Разбиение выполняемых нейроном вычислений на два отдельных значения показано на рис. 1.7.

По отношению к значениям, вычисляемым до применения к ним функции активации $\Phi(\cdot)$, мы будем использовать термин *преактивационное значение*, тогда как по отношению к значениям, вычисленным после применения функции активации, — термин *постактивационное значение*. Выходом нейрона всегда является постактивационное значение, хотя преактивационные переменные также часто используются в различных типах анализа, таких, например, как алгоритм *обратного распространения ошибки* (backpropagation algorithm), к обсуждению которого мы еще вернемся далее. Происхождение пред- и постактивационных значений показано на рис. 1.7.

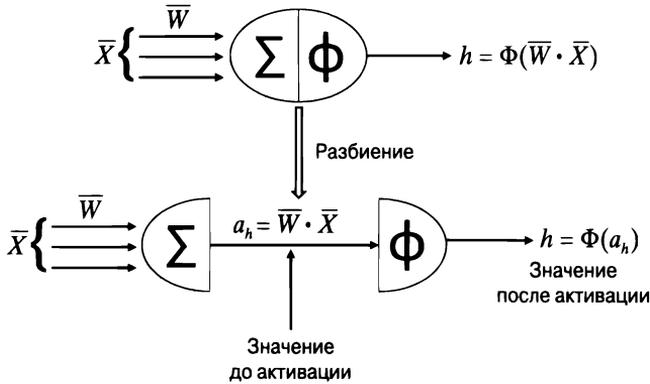


Рис. 1.7. Значения, вычисляемые в нейроне до и после активации

Простейшей функцией активации $\Phi(\cdot)$ является тождественная или линейная функция, вообще не вносящая никакой нелинейности:

$$\Phi(v) = v.$$

Линейная активационная функция часто используется в выходных узлах в тех случаях, когда целевое значение является вещественным числом. Ее используют даже для дискретных выходов, если необходимо задать сглаженную суррогатную функцию потерь.

К числу классических функций активации, которые применялись на ранних этапах разработки нейронных сетей, относятся знаковая функция, или сигнум (*sign*), сигмоида (*sigmoid*) и гиперболический тангенс (*tanh*):

$$\Phi(v) = \text{sign}(v) \quad (\text{сигнум}),$$

$$\Phi(v) = \frac{1}{1 + e^{-v}} \quad (\text{сигмоида}),$$

$$\Phi(v) = \frac{e^{-2v} - 1}{e^{-2v} + 1} \quad (\text{гиперболический тангенс}).$$

Несмотря на то что функцию активации *sign* можно применять для трансляции значений на бинарные выходы на этапе предсказания, тот факт, что она не является дифференцируемой функцией, препятствует ее использованию для создания функции потерь на этапе тренировки. Например, перцептрон задействует функцию *sign* для предсказаний, тогда как критерий перцептрона требует использовать во время тренировки только линейную активацию. Выходное значение сигмоиды принадлежит к интервалу $[0, 1]$, что удобно в тех случаях, когда результаты вычислений должны интерпретироваться как вероятностные значения. Это также оказывается полезным при создании вероятностных выходов и конструировании функций потерь, получаемых из моделей

максимального правдоподобия. Гиперболический тангенс имеет примерно ту же форму, что и сигмоида, за исключением того, что функция \tanh масштабирована вдоль горизонтальной оси, а также масштабирована и сдвинута вдоль вертикальной оси таким образом, что ее значения принадлежат к интервалу $[-1, 1]$. Связь между гиперболическим тангенсом и сигмоидой описывается следующей формулой (см. упражнение 3):

$$\tanh(v) = 2 \cdot \text{sigmoid}(2v) - 1.$$

Гиперболический тангенс более предпочтителен, чем сигмоида, в тех случаях, когда желательно, чтобы результаты вычислений могли иметь как положительные, так и отрицательные значения. Кроме того, его центрированность на среднем значении и больший (из-за более сжатой формы функции) градиент по сравнению с сигмоидой упрощают тренировку модели. Ранее для введения нелинейности в нейронные сети в качестве функций активации традиционно использовали сигмоиду или гиперболический тангенс. Однако в последние годы все большую популярность приобретают различные кусочно-линейные функции активации наподобие тех, которые приведены ниже.

$$\begin{aligned} \Phi(v) &= \max\{v, 0\} && \text{(полулинейный элемент [ReLU]),} \\ \Phi(v) &= \max\{\min[v, 1], -1\} && \text{(спрямленный гиперболический тангенс).} \end{aligned}$$

В современных нейронных сетях функции активации ReLU (Rectified Linear Unit) и спрямленный гиперболический тангенс в значительной степени вытеснили сигмоиду и гиперболический тангенс, поскольку их использование упрощает тренировку многослойных нейронных сетей.

Вышеупомянутые функции активации представлены на рис. 1.8. Заслуживает внимания тот факт, что все они являются монотонными функциями. Кроме того, за исключением тождественной функции, в своем большинстве¹ они *насыщаются* при больших абсолютных значениях аргумента, т.е. не изменяются существенно при дальнейшем увеличении его абсолютного значения.

Как будет показано далее, подобные нелинейные функции активации находят широкое применение в многослойных нейронных сетях, поскольку они помогают создавать более мощные композиции функций различного типа. Многие из этих функций называют функциями *сжатия*, поскольку они транслируют выходные значения из произвольного интервала на ограниченный интервал значений. Использование нелинейной активации играет фундаментальную роль в увеличении моделирующей способности сети. Если сеть использует только линейные виды активации, то она не сможет выйти за пределы возможностей однослойной нейронной сети. Эта проблема обсуждается в разделе 1.5.

¹ Функция активации ReLU демонстрирует асимметричное насыщение.

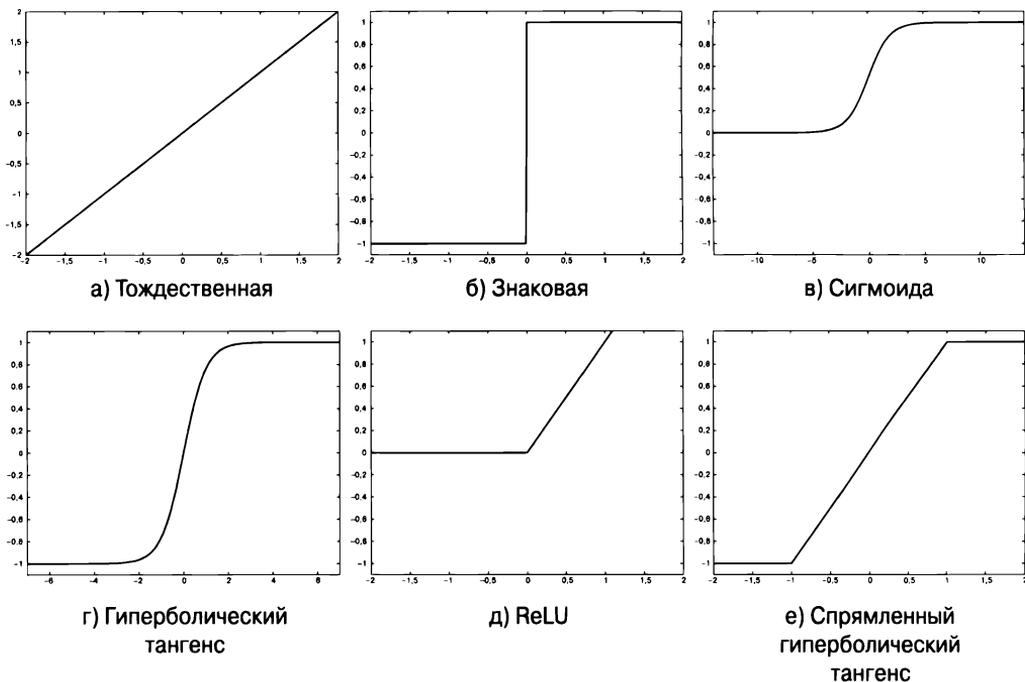


Рис. 1.8. Разновидности функции активации

1.2.1.4. Выбор выходных узлов и их количества

Выбор выходных узлов и их количества в значительной степени зависит от выбора функции активации, что, в свою очередь, определяется конкретной задачей. Например, в случае k -арной классификации могут использоваться k выходных значений $\bar{v} = [v_1 \dots v_k]$ с применением функции активации *Softmax* в узлах данного слоя. В частности, функция активации для выхода определяется следующим образом:

$$\Phi(\bar{v})_i = \frac{\exp(v_i)}{\sum_{j=1}^k \exp(v_j)} \quad \forall i \in \{1 \dots k\}. \quad (1.12)$$

Эти k значений полезно рассматривать как выходные значения k узлов, входными значениями которых являются $v_1 \dots v_k$. Пример функции *Softmax* с тремя выходами приведен на рис. 1.9, на котором также представлены соответствующие значения v_1 , v_2 и v_3 . Обратите внимание на то, что три выхода соответствуют трем классам, и они преобразуют выходные значения последнего скрытого слоя в вероятности с помощью функции *Softmax*. В последнем скрытом слое часто используют линейную (тождественную) функцию активации, если его

выход передается в качестве входа слою Softmax. Кроме того, со слоем Softmax не связаны никакие веса, поскольку он лишь преобразует вещественные значения в вероятности. Использование функции активации *Softmax* с единственным скрытым слоем линейной активации в точности реализует модель так называемой *мультиномиальной логистической регрессии* (multinomial logistic regression) [6]. Аналогичным образом с помощью нейронных сетей могут быть легко реализованы многие вариации методов наподобие мультиклассовых SVM. В качестве еще одного типичного примера использования множественных выходных узлов можно привести *автокодировщик*, в котором каждая входная точка данных полностью реконструируется выходным слоем. Автокодировщики могут применяться для реализации методов матричной факторизации наподобие *сингулярного разложения* (singular value decomposition). Эта архитектура будет подробно обсуждаться в главе 2. Изучение простейших нейронных сетей, имитирующих базовые алгоритмы машинного обучения, может быть весьма поучительным, поскольку они занимают промежуточное положение между традиционным машинным обучением и глубокими сетями. Исследование этих архитектур позволяет лучше понять связь между традиционным машинным обучением и нейронными сетями, а также преимущества, обеспечиваемые последними.

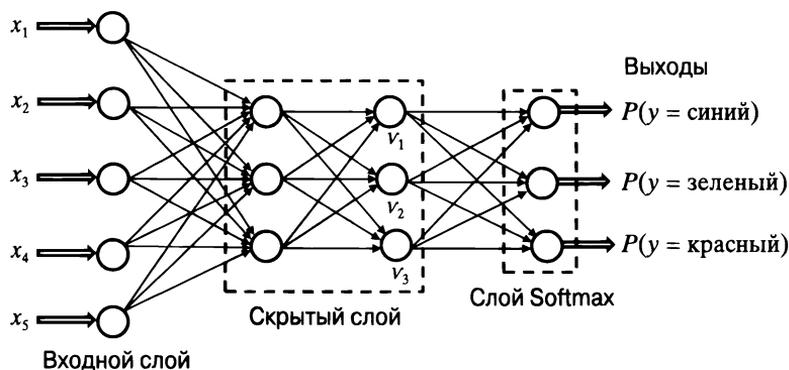


Рис. 1.9. Пример множественных выходов для категориальной классификации с использованием слоя *Softmax*

1.2.1.5. Выбор функции потерь

При определении выходов, наиболее подходящих для данной задачи, очень многое зависит от правильного выбора функции потерь. Например, регрессия по методу наименьших квадратов с числовыми выходами требует использования простой квадратичной функции потерь вида $(y - \hat{y})^2$ для простого тренировочного примера с целевым значением y и предсказанием \hat{y} . Можно задейство-

вать и другие типы потерь наподобие кусочно-линейной функции $y \in \{-1, +1\}$ и вещественного предсказания \hat{y} (с тождественной функцией активации):

$$L = \max\{0, 1 - y \cdot \hat{y}\}. \quad (1.13)$$

Кусочно-линейные потери можно применять для реализации метода обучения, известного как *метод опорных векторов* (Support Vector Machine — SVM).

Для множественных предсказаний (таких, как предсказание идентификаторов слов или одного из нескольких классов) особенно полезен выход Softmax. Однако выход такого типа — вероятностный и поэтому требует использования функции потерь другого типа. В действительности для вероятностных предсказаний используют два различных типа функций потерь, в зависимости от того, является ли предсказание бинарным или многоарным.

- 1. Бинарные целевые значения (логистическая регрессия).** В этом случае предполагается, что наблюдаемое значение y извлекается из набора $\{-1, +1\}$, а предсказание \hat{y} является произвольным числовым значением при использовании тождественной функции активации. В подобных ситуациях функция потерь для одиночного примера с наблюдаемым значением y и предсказанием \hat{y} в виде вещественного числа (с тождественной активацией) определяется как

$$L = \log(1 + \exp(-y \cdot \hat{y})). \quad (1.14)$$

Функция потерь этого типа реализует фундаментальный метод машинного обучения, известный как *логистическая регрессия*. Другой возможный подход заключается в использовании сигмоиды для получения выхода $y \in \{0, 1\}$, определяющего вероятность того, что наблюдаемое значение y равно 1. Далее, отрицательный логарифм $|y / 2 - 0,5 + \bar{y}|$ представляет потери в предположении, что y кодируется из набора $\{-1, 1\}$. Это обусловлено тем, что $|y / 2 - 0,5 + \bar{y}|$ определяет вероятность того, что предсказание корректно. Данное наблюдение иллюстрирует тот факт, что для достижения одного и того же результата можно использовать различные комбинации функции активации и функции потерь.

- 2. Категориальные целевые значения.** В данном случае, если $\hat{y}_1 \dots \hat{y}_k$ — вероятности k классов (в уравнении 1.9 используется функция активации *Softmax*) и r -й класс — истинный, то функция потерь для одиночного примера определяется следующим образом:

$$L = -\log(\hat{y}_r). \quad (1.15)$$

Функция потерь этого типа реализует мультиномиальную логистическую регрессию и известна под названием *кросс-энтропийные потери*

(cross-entropy loss). Заметьте, что бинарная логистическая регрессия идентична мультиномиальной логистической регрессии при $k = 2$.

Очень важно не забывать о том, что выбор выходных узлов, функции активации и функции потерь определяется конкретной задачей. Кроме того, эти варианты выбора также взаимосвязаны. Несмотря на то что перцептрон часто рассматривают как квинтэссенцию однослойных сетей, он представляет лишь одну из многочисленных возможностей. На практике критерий перцептрона редко используется в качестве функции потерь. В случае дискретных выходных значений обычно используют активацию *Softmax* с кросс-энтропийными потерями, в случае вещественных — линейную активацию с квадратичными потерями. Как правило, оптимизировать кросс-энтропийные потери легче, чем квадратичные.

1.2.1.6. Некоторые полезные производные функций активации

В большинстве случаев обучение нейронных сетей главным образом связано с градиентным спуском и функциями активации. По этой причине производные функций активации многократно используются в книге, и их обсуждение в одном месте будет полезным. В этом разделе подробно описаны производные функций потерь. В последующих главах мы будем неоднократно обращаться к приведенным здесь результатам.

1. *Линейная и знаковая функции активации.* Производная линейной функции активации везде равна 1. Производная функции $sign(v)$ равна 0 при всех значениях v , отличных от точки $v = 0$, где она разрывна и не имеет производной. Из-за нулевого значения градиента этой функции и ее недифференцируемости ее редко применяют в функции потерь, даже если она и используется для получения предсказаний на этапе тестирования. Производные линейной и знаковой функций активации представлены в графическом виде на рис. 1.10, *a* и *б*, соответственно.
2. *Сигмоида.* Производная сигмоиды выглядит особенно просто, будучи выраженной в терминах выхода сигмоиды, а не входа. Обозначим через o выход сигмоиды с аргументом v :

$$o = \frac{1}{1 + \exp(-v)}. \quad (1.16)$$

Тогда производную функции активации можно записать в следующем виде:

$$\frac{\partial o}{\partial v} = \frac{\exp(-v)}{(1 + \exp(-v))^2}. \quad (1.17)$$

Очень важно, что данное выражение можно переписать в более удобной форме в терминах выходов:

$$\frac{\partial o}{\partial v} = o(1 - o). \quad (1.18)$$

Производную сигмоиды часто используют в виде функции выхода, а не входа. График производной сигмоиды приведен на рис. 1.10, в.

3. *Функция активации tanh*. Как и в случае сигмоиды, активацию *tanh* часто используют в качестве функции выхода o , а не входа v :

$$o = \frac{\exp(2v) - 1}{\exp(2v) + 1}. \quad (1.19)$$

Тогда градиент этой функции можно выразить в следующем виде:

$$\frac{\partial o}{\partial v} = \frac{4 \cdot \exp(2v)}{(\exp(2v) + 1)^2}. \quad (1.20)$$

Это выражение можно переписать в терминах выхода o :

$$\frac{\partial o}{\partial v} = 1 - o^2. \quad (1.21)$$

График производной функции активации в виде гиперболического тангенса приведен на рис. 1.10, г.

4. *Функции активации ReLU и спрямленный гиперболический тангенс*. Производная функции активации ReLU равна 1 для неотрицательных значений аргумента и нулю в остальных случаях. Производная спрямленного гиперболического тангенса равна 1 для значений аргумента в интервале $[-1, +1]$ и нулю в остальных случаях. Графики производных функции ReLU и спрямленного гиперболического тангенса приведены на рис. 1.10, д и е, соответственно.

1.2.2. Многослойные нейронные сети

Перцептрон содержит входной и выходной слои, из которых вычисления выполняются только в выходном слое. Входной слой передает данные выходному, и все вычисления полностью видны пользователю. Многослойные нейронные сети содержат несколько вычислительных слоев. Дополнительные промежуточные слои (расположенные между входом и выходом) называют *скрытыми слоями*, поскольку выполняемые ими вычисления остаются невидимыми для пользователя. Базовую архитектуру многослойных нейронных сетей называют *сетями прямого распространения* (feed-forward networks), поскольку в сетях этого типа данные последовательно передаются от одного слоя к другому в прямом направлении от входа к выходу. В используемой по умолчанию архитектуре

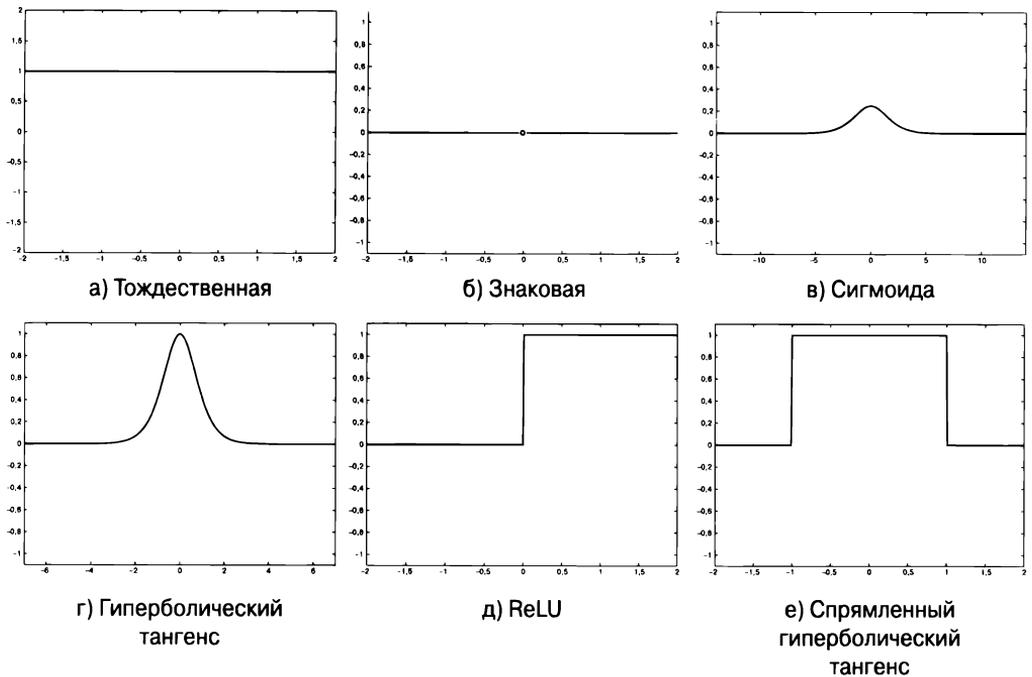


Рис. 1.10. Производные различных функций активации

сетей прямого распространения предполагается, что каждый узел одного слоя связан с каждым узлом следующего слоя. Поэтому архитектура такой сети почти полностью определяется количеством слоев и количеством/типом узлов в каждом слое. Единственное, что остается дополнительно определить, — это функцию потерь, которая оптимизируется в выходном слое. И хотя в алгоритме перцептрона используется критерий перцептрона, это не единственный вариант выбора. Очень часто используют выходы Softmax с кросс-энтропийной функцией потерь для предсказания дискретных значений и линейные выходы с квадратичной функцией потерь для предсказания вещественных значений.

Как и в случае однослойных сетей, в скрытых и выходных слоях многослойных сетей можно использовать нейроны смещения. Примеры многослойных нейронных сетей с нейронами смещения и без них приведены на рис. 1.11, а и б, соответственно. В каждом из этих случаев нейронная сеть содержит три слоя. Обратите внимание на то, что входной слой часто не учитывают при подсчете общего количества слоев, поскольку он просто передает данные следующему слою, не выполняя самостоятельно никаких вычислений. Если нейронная сеть содержит элементы $p_1 \dots p_k$ в каждом из своих k слоев, то представления их выходов $\hat{h}_1 \dots \hat{h}_k$ в виде векторов (столбцов) имеют размерности $p_1 \dots p_k$. Поэтому количество элементов в каждом слое называют *размерностью (размером)* данного слоя.

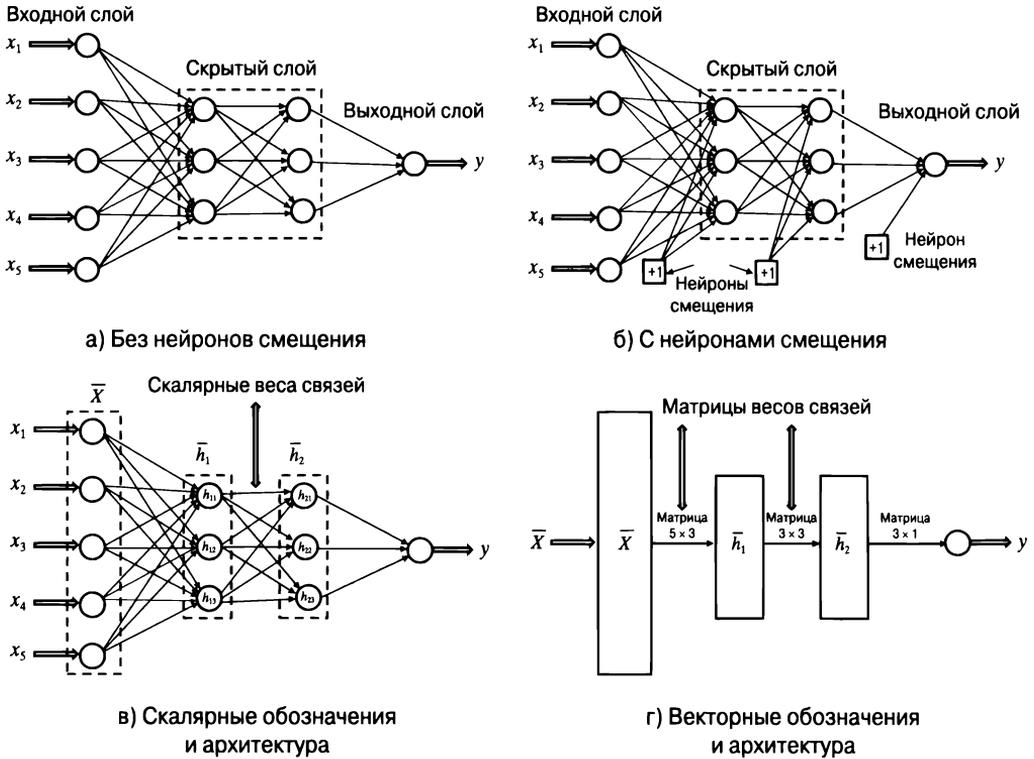


Рис. 1.11. Базовая архитектура сети прямого распространения с двумя скрытыми слоями и одним выходным слоем. Несмотря на то что каждый элемент содержит одну скалярную переменную, всю совокупность элементов одного слоя часто представляют в виде одного векторного элемента. Векторные элементы часто изображают в виде прямоугольников с указанием матриц весов связей на соединениях между ними

Веса связей, соединяющих входной слой с первым скрытым слоем, содержатся в матрице W_1 размера $d \times p_1$, тогда как веса связей между r -м и $(r+1)$ -м скрытыми слоями — в матрице W_r размера $p_r \times p_{r+1}$. Если выходной слой содержит o узлов, то заключительная матрица W_{k+1} имеет размер $p_k \times o$. d -мерный входной вектор x преобразуется в выходы с использованием следующих рекурсивных уравнений:

$$\begin{aligned}
 \bar{h}_1 &= \Phi(W_1^T \bar{x}) && \text{[входной со скрытыми]}, \\
 \bar{h}_{p+1} &= \Phi(W_{p+1}^T \bar{h}_p) \quad \forall p \in \{1 \dots k-1\} && \text{[скрытый со скрытыми]}, \\
 \bar{o} &= \Phi(W_{k+1}^T \bar{h}_k) && \text{[скрытый с выходным]}.
 \end{aligned}$$

В этих уравнениях функции активации наподобие сигмоиды применяются к своим векторным аргументам *поэлементно*. Однако для некоторых функций активации, таких как *Softmax* (которые, как правило, используются в выходных

слоях), естественными являются векторные аргументы. И хотя каждый элемент нейронной сети содержит одну переменную, на многих архитектурных диаграммах эти элементы объединяются в один слой для создания единого векторного элемента, обозначаемого *прямоугольником*, а не *кружочком*. Так, архитектурная диаграмма, приведенная на рис. 1.11, в (со скалярными элементами), преобразована в векторную нейронную архитектуру, приведенную на рис. 1.11, з. Обратите внимание на то, что теперь соединения между векторными элементами представлены матрицами. Кроме того, неявно предполагается, что в векторных нейронных архитектурах во всех элементах данного слоя используется одна и та же функция активации, применяемая поэлементно ко всему слою. Обычно это ограничение не является проблемой, поскольку в большинстве нейронных архитектур вдоль всей цепочки вычислений используется одна и та же функция активации, с единственным исключением из этого правила, обусловленным природой выходного слоя. На протяжении всей книги элементы нейронных архитектур, содержащие векторные переменные, будут изображаться в виде прямоугольников, тогда как скалярным переменным будут соответствовать кружочки.

Имейте в виду, что вышеприведенные рекуррентные уравнения и векторные архитектуры справедливы лишь для сетей прямого распространения, структура которых представлена слоями, и не всегда могут использоваться в случае нестандартных архитектурных решений. К таким нестандартным решениям относятся любые архитектуры с входами, включенными в промежуточные слои, или топологиями, допускающими соединения между несмежными слоями. Кроме того, функции, вычисляемые в узле, не всегда могут представлять собой некоторую комбинацию линейной функции и функции активации. Для выполнения вычислений в узлах могут использоваться любые функции.

На рис. 1.11 был представлен строго классический тип архитектур сетей, однако он может варьироваться самыми разными способами, такими, например, как введение нескольких выходных слоев. Возможные варианты выбора часто диктуются условиями конкретной задачи (такой, например, как классификация или снижение размерности). Классическим примером задачи снижения размерности является автокодировщик, цель которого — воссоздание выходов на основании входов. В этом случае количество выходов совпадает с количеством входов (рис. 1.12). Расположенный посередине скрытый слой выводит уменьшенное представление каждого примера. В результате этого ограничения часть информации теряется, что обычно сопровождается появлением шума в данных. Выходы скрытых слоев соответствуют представлению данных, имеющему меньшую размерность. В действительности можно показать, что мелкий вариант этой схемы с математической точки зрения эквивалентен такому хорошо известному методу снижения размерности данных, как *сингулярное разложение*.

Как вы узнаете из главы 2, увеличение глубины сети позволяет еще существеннее снижать размерность данных.

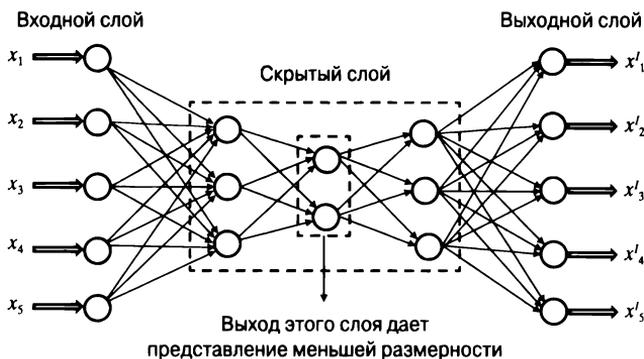


Рис. 1.12. Пример автокодировщика с несколькими выходами

Несмотря на то что полносвязная архитектура хорошо подходит для многих задач, ее эффективность часто удается повысить за счет исключения некоторых связей или их разделения тщательно продуманным способом. Как правило, понимание того, как это следует делать, приходит в результате анализа данных с учетом специфики конкретной задачи. Классическим примером такого исключения и разделения весов может служить *сверточная нейронная сеть* (глава 8), архитектура которой тщательно спроектирована таким образом, чтобы она согласовывалась с типичными свойствами данных изображения. Такой подход минимизирует риск *переобучения* путем учета специфики свойств данных этого типа (смещения). Как будет подробно обсуждаться в главе 4, переобучение — распространенная проблема нейронных сетей, проявляющаяся в том, что сеть очень хорошо работает на тренировочных данных, но плохо *обобщается* на неизвестные тестовые данные. Эта проблема возникает тогда, когда количество свободных параметров (обычно равное количеству взвешенных связей) слишком велико по сравнению с объемом тренировочных данных. В подобных случаях многочисленные параметры запоминают специфические нюансы тренировочных данных, но теряют способность распознавать статистически значимые закономерности и использовать их для классификации неизвестных тестовых данных. Совершенно очевидно, что увеличение количества узлов в нейронной сети повышает ее склонность к переобучению. В последнее время интенсивно ведется работа по совершенствованию как архитектур нейронных сетей, так и вычислений, выполняемых в каждом узле, направленная на то, чтобы минимизировать риски переобучения. К тому же на качество конечного решения оказывает влияние также способ, используемый для тренировки сети. За последние годы для повышения качества обучения был предложен ряд эффективных методов, таких как *предварительное обучение* (pretraining), о котором

будет рассказано в главе 4. Все эти современные методы тренировки подробно обсуждаются в данной книге.

1.2.3. Многослойная нейронная сеть как вычислительный граф

Нейронную сеть полезно рассматривать в качестве *вычислительного графа*, который строится путем объединения в единое целое многих базовых параметрических моделей. Нейронные сети обладают значительно более мощными возможностями, чем их строительные блоки, поскольку параметры этих моделей обучаются *совместному* созданию в высшей степени оптимизированной композиционной функции моделей. Обычное использование термина “перцептрон” в отношении базового элемента нейронной сети может вводить в заблуждение, поскольку разработаны многочисленные вариации этого базового элемента, являющиеся более предпочтительными в тех или иных ситуациях. В действительности в качестве строительных блоков таких моделей чаще встречаются логистические элементы (с сигмоидной функцией активации).

Многослойная сеть оценивает результат композиций функций, вычисляемых в индивидуальных узлах. Путь длиной 2 в нейронной сети, в которой вслед за функцией $f(\cdot)$ вычисляется функция $g(\cdot)$, можно рассматривать как сложную функцию $f(g(\cdot))$. Кроме того, если $g_1(\cdot)$, $g_2(\cdot)$, ..., $g_k(\cdot)$ — это функции, вычисляемые в слое m , а отдельный узел слоя $(m + 1)$ вычисляет функцию $f(\cdot)$, то сложная функция, вычисляемая узлом слоя $(m + 1)$ в терминах входов слоя m , может быть записана как $f(g_1(\cdot) \dots g_k(\cdot))$. Ключом к повышению вычислительной мощности множества слоев является использование нелинейной функции активации. Можно показать, что если во всех слоях используется *тождественная* (линейная) функция активации, то такая многослойная сеть сводится к линейной регрессии. Было доказано [208], что сеть, состоящая из одного скрытого слоя нелинейных элементов (с широким выбором *функций сжатия* наподобие сигмоиды) и одного (линейного) выходного слоя, способна вычислить почти любую “разумную” функцию. Вследствие этого нейронные сети часто называют *универсальными аппроксиматорами функций* (universal function approximators), хотя это теоретическое заявление не всегда легко реализовать на практике. Основная трудность состоит в том, что количество необходимых для этого скрытых элементов довольно велико, что увеличивает количество параметров, подлежащих обучению. В результате на практике при обучении сети с использованием ограниченного объема данных возникают проблемы. Фактически глубокие сети нередко оказываются более предпочтительными, поскольку они позволяют уменьшить количество скрытых элементов в каждом слое, а вместе с этим и общее количество параметров.

Использование выражения “строительный блок” применительно к многослойным нейронным сетям представляется особенно уместным. Очень часто готовые библиотеки², предназначенные для создания нейронных сетей, предоставляют аналитикам доступ к таким строительным блокам. Аналитику достаточно указать количество и тип элементов в каждом слое, а также готовую или нестандартную функцию потерь. Глубокую нейронную сеть, содержащую десятки слоев, часто можно описать несколькими строками кода. Обучение весов осуществляется автоматически с помощью алгоритма обратного распространения ошибки, который использует динамическое программирование для формирования сложных этапов обновления параметров базового вычислительного графа. Аналитик не должен тратить время и усилия на самостоятельное прохождение этих этапов, благодаря чему испытание различных типов архитектур будет восприниматься им как относительно безболезненный процесс. Создание нейронной сети с помощью готовых программных средств часто сравнивают со сборкой игрушек из деталей детского конструктора. Каждый блок подобен элементу (или слою элементов) с определенным типом активации. Значительное упрощение обучения нейронных сетей достигается за счет использования алгоритма обратного распространения ошибки, который избавляет аналитика от ручного проектирования всех этапов по обновлению параметров, являющегося на самом деле лишь частью чрезвычайно сложного процесса оптимизации. Прохождение этих этапов зачастую является наиболее сложной частью большинства алгоритмов машинного обучения, и важная роль парадигмы нейронных сетей заключается в привнесении идеи модульности в машинное обучение. Другими словами, модульность структуры нейронной сети транслируется в модульность процесса обучения ее параметров. Для последнего типа модульности существует специальное название — *обратное распространение ошибки* (backpropagation). При таком подходе характер проектирования нейронных сетей изменяется, и вместо занятия для математиков оно становится задачей для (опытных) инженеров.

1.3. Тренировка нейронной сети с помощью алгоритма обратного распространения ошибки

Процесс обучения однослойной нейронной сети отличается сравнительной простотой, поскольку ошибка (или функция потерь) может быть вычислена как непосредственная функция весов, что дает возможность легко вычислять градиент. В случае многослойных нейронных сетей проблема обусловлена тем, что потери представляют собой сложную композиционную функцию весов,

² В качестве примера можно привести библиотеки Torch [572], Theano [573] и TensorFlow [574].

относящихся к предыдущим слоям. Градиент композиционной функции вычисляется с помощью алгоритма обратного распространения ошибки. Этот алгоритм использует цепное правило дифференциального исчисления, в соответствии с которым градиенты вычисляются путем суммирования произведений локальных градиентов вдоль различных путей от узла к выходу. Несмотря на то что эта сумма включает экспоненциально возрастающее количество компонент (путей), она поддается эффективному вычислению с помощью *динамического программирования*. Алгоритм обратного распространения ошибки является непосредственным результатом динамического программирования и включает две фазы: *прямую* и *обратную*. Прямая фаза необходима для вычисления выходных значений и локальных производных в различных узлах, а обратная — для аккумуляции произведений этих локальных значений по всем путям, ведущим от данного узла к выходу.

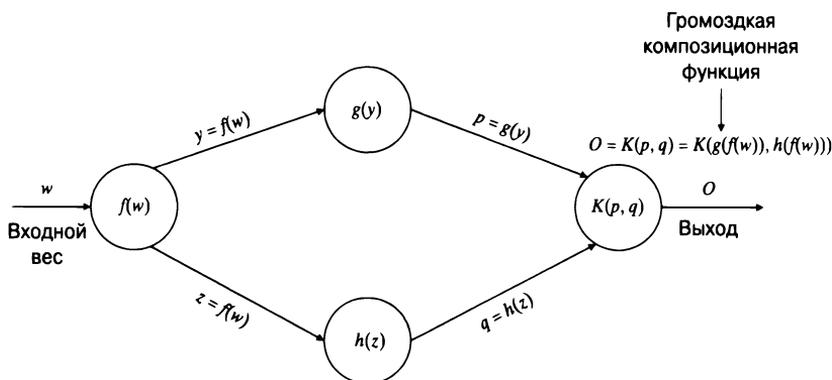
1. *Прямая фаза.* На протяжении этой фазы нейронной сети передаются входные данные тренировочного примера. Результатом является каскад вычислений, выполняемых поочередно в каждом из слоев в порядке их следования с использованием текущего набора весов. Предсказанный окончательный выход сравнивается с тренировочным примером, что позволяет вычислить для данного выхода производную функции потерь. Эта производная вычисляется по весам во всех слоях на протяжении обратной фазы.
2. *Обратная фаза.* Основной задачей обратной фазы является обучение градиенту функции потерь по различным весам посредством использования цепного правила дифференциального исчисления. Результирующие градиенты используются для обновления весов. Поскольку процесс тренировки (обучения) градиентов осуществляется в обратном направлении, его называют обратной фазой. Рассмотрим последовательность скрытых элементов h_1, h_2, \dots, h_k , за которой следует выход o , относительно которого вычисляется функция потерь L . Кроме того, предположим, что вес соединения, связывающего скрытый элемент h_r с элементом h_{r+1} равен $w_{(h_r, h_{r+1})}$. Тогда в случае, если существует единственный путь, соединяющий узел h_1 с выходом o , градиент функции потерь по любому из этих весов может быть вычислен с помощью цепного правила в соответствии со следующей формулой:

$$\frac{\partial L}{\partial w_{(h_{r-1}, h_r)}} = \frac{\partial L}{\partial o} \cdot \left[\frac{\partial o}{\partial h_k} \prod_{i=r}^{k-1} \frac{\partial h_{i+1}}{\partial h_i} \right] \frac{\partial h_r}{\partial w_{(h_{r-1}, h_r)}} \quad \forall r \in 1 \dots k. \quad (1.22)$$

В этом выражении предполагается, что в сети существует *только один* путь, ведущий от h_1 к o , тогда как в действительности число таких путей

может экспоненциально возрастать с ростом размера сети. Для расчета градиента в вычислительном графе, в котором существует не один такой путь, используют обобщенный вариант цепного правила, так называемое *многомерное цепное правило* (multivariable chain rule). Это достигается добавлением композиций вдоль каждого из путей, ведущих от узла h_r к o . Пример, иллюстрирующий применение цепного правила в вычислительном графе с двумя путями, приведен на рис. 1.13. Поэтому представленное выше выражение обобщают на случай, когда существует набор P путей, ведущих от узла h_r к o :

$$\frac{\partial L}{\partial w_{(h_{r-1}, h_r)}} = \frac{\partial L}{\partial o} \cdot \underbrace{\left[\sum_{\{h_r, h_{r+1}, \dots, h_k, o\} \in P} \frac{\partial o}{\partial h_k} \prod_{i=r}^{k-1} \frac{\partial h_{i+1}}{\partial h_i} \right]}_{\substack{\text{Алгоритм обратного распространения} \\ \text{ошибки вычисляет } \nabla(h_r, o) = \frac{\partial L}{\partial h_r}}} \frac{\partial h_r}{\partial w_{(h_{r-1}, h_r)}}. \quad (1.23)$$



$$\begin{aligned} \frac{\partial o}{\partial w} &= \frac{\partial o}{\partial p} \cdot \frac{\partial p}{\partial w} + \frac{\partial o}{\partial q} \cdot \frac{\partial q}{\partial w} = && \text{[многомерное цепное правило]} \\ &= \frac{\partial o}{\partial p} \cdot \frac{\partial p}{\partial y} \cdot \frac{\partial y}{\partial w} + \frac{\partial o}{\partial q} \cdot \frac{\partial q}{\partial z} \cdot \frac{\partial z}{\partial w} = && \text{[одномерное цепное правило]} \\ &= \underbrace{\frac{\partial K(p, q)}{\partial p} \cdot g'(y) \cdot f'(w)}_{\text{Первый путь}} + \underbrace{\frac{\partial K(p, q)}{\partial q} \cdot h'(z) \cdot f'(w)}_{\text{Второй путь}}. \end{aligned}$$

Рис. 1.13. Цепное правило в вычислительных графах. Произведения специфических для узлов частных производных вдоль путей от веса w к выходу o агрегируются. Результирующее значение дает производную выхода o по весу w . В данном простом примере существуют лишь два пути, ведущих от выхода к входу

Вычисление члена $\frac{\partial h_r}{\partial w_{(h_{r-1}, h_r)}}$ в правой части, к которому мы еще вернемся (уравнение 1.27), не представляет трудностей. Однако член $\nabla(h_r, o) = \frac{\partial L}{\partial h_r}$ агрегируется по экспоненциально возрастающему (с ростом длины пути) количеству путей, что, на первый взгляд, кажется непреодолимым препятствием. В данном случае ключевую роль играет тот факт, что вычислительный граф нейронной сети не имеет циклов, что делает принципиально возможным вычисление подобной агрегации в обратном направлении путем вычисления сначала членов $D(h_k, o)$ для узлов h_k , ближайших к o , а затем рекурсивного вычисления этих значений для узлов в предыдущих слоях по отношению к последующим. Кроме того, значение $D(o, o)$ для каждого выходного узла инициализируется следующим образом:

$$\Delta(o, o) = \frac{\partial L}{\partial o}. \quad (1.24)$$

Этот тип динамического программирования часто используют для эффективного вычисления всех типов функций, центрированных на путях в направленных ациклических графах, что в противном случае потребовало бы экспоненциально возрастающего количества операций. Рекурсию для $D(h_k, o)$ можно легко получить с помощью многомерного цепного правила:

$$\Delta(h_r, o) = \frac{\partial L}{\partial h_r} = \sum_{h: h_r \Rightarrow h} \frac{\partial L}{\partial h} \frac{\partial h}{\partial h_r} = \sum_{h: h_r \Rightarrow h} \frac{\partial h}{\partial h_r} \Delta(h, o). \quad (1.25)$$

Поскольку все фигурирующие здесь узлы h содержатся в слоях, следующих за слоем, в котором находится узел h_r , то к моменту вычисления члена $D(h_k, o)$ член $D(h, o)$ оказывается уже вычисленным. Однако для вычисления уравнения 1.25 нам все еще нужно вычислить член $\frac{\partial h}{\partial h_r}$. Рассмотрим ситуацию, в которой связь, соединяющая узел h_r с узлом h , имеет вес $w_{(h_r, h)}$, и пусть a_h — значение, вычисленное в скрытом элементе h непосредственно *перед* применением функции активации $\Phi(\cdot)$. Иными словами, мы имеем соотношение $h = \Phi(a_h)$, где a_h — линейная комбинация входов элемента h , связанных с элементами предыдущего слоя. Тогда в соответствии с одномерным цепным правилом мы получаем для члена следующее выражение:

$$\frac{\partial h}{\partial h_r} = \frac{\partial h}{\partial a_h} \cdot \frac{\partial a_h}{\partial h_r} = \frac{\partial \Phi(a_h)}{\partial a_h} \cdot w_{(h_r, h)} = \Phi'(a_h) \cdot w_{(h_r, h)}.$$

Это значение $\frac{\partial h}{\partial h_r}$ используется в уравнении 1.25, которое рекурсивно повторяется в обратном направлении, начиная с выходного узла. Соответствующие обновления в обратном направлении можно записать следующим образом:

$$\Delta(h_r, o) = \sum_{h: h_r \Rightarrow h} \Phi'(a_h) \cdot w_{(h_r, h)} \cdot \Delta(h, o). \quad (1.26)$$

Поэтому градиенты последовательно аккумулируются при проходе в обратном направлении, и при этом каждый узел обрабатывается ровно один раз. Обратите внимание на то, что для вычисления градиента по всем весам связей уравнение 1.25 (число операций в котором пропорционально числу исходящих связей) должно быть вычислено для каждой входящей связи узла. Наконец, уравнение 1.23 требует вычисления члена $\frac{\partial h_r}{\partial w_{(h_{r-1}, h_r)}}$, что можно легко сделать в соответствии со следующей формулой:

$$\frac{\partial h_r}{\partial w_{(h_{r-1}, h_r)}} = h_{r-1} \cdot \Phi'(a_{h_r}). \quad (1.27)$$

Здесь основным градиентом, распространяющимся в обратном направлении, является производная по активациям слоя, а градиент по весам легко вычисляется для любой входящей связи соответствующего элемента.

Следует отметить, что динамическая рекурсия (уравнение 1.26) может вычисляться множеством способов, в зависимости от того, какие переменные используются в качестве промежуточных при формировании цепочек. Все подобные рекурсии эквивалентны в плане получения окончательного результата обратного распространения ошибки. Далее мы представим альтернативную версию такой рекурсии, которая чаще встречается в учебниках. Обратите внимание на то, что в уравнении 1.23 переменные в скрытых слоях используются в качестве “цепочечных” переменных для динамической рекурсии. Для цепного правила также можно использовать преактивационные переменные, которые в нейроне получают после применения линейного преобразования (но до применения активации) в качестве промежуточных переменных. Преактивационное значение скрытой переменной $h = \Phi(a_h)$ равно a_h . Различия между преактивационным и постактивационным значениями в нейроне показаны на рис. 1.7. Поэтому вместо уравнения 1.23 можно использовать следующее цепное правило:

$$\frac{\partial L}{\partial w_{(h_{r-1}, h_r)}} = \frac{\partial L}{\partial o} \cdot \underbrace{\Phi'(a_o) \cdot \left[\sum_{\{h_r, h_{r+1}, \dots, h_k, o\} \in P} \frac{\partial a_o}{\partial a_{h_k}} \prod_{i=r}^{k-1} \frac{\partial a_{h_{i+1}}}{\partial a_{h_i}} \right]}_{\text{Алгоритм обратного распространения}} \underbrace{\frac{\partial a_{h_r}}{\partial w_{(h_{r-1}, h_r)}}}_{h_{r-1}}. \quad (1.28)$$

ошибки вычисляет $\delta(h_r, o) = \frac{\partial L}{\partial a_{h_r}}$

Записывая это рекурсивное уравнение, мы использовали обозначение $\delta(h_r, o) = \frac{\partial L}{\partial a_{h_r}}$ вместо $\Delta(h_r, o) = \frac{\partial L}{\partial h_r}$. Значение $\delta(o, o) = \frac{\partial L}{\partial a_o}$ инициализируется следующим образом:

$$\delta(o, o) = \frac{\partial L}{\partial a_o} = \Phi'(a_o) \cdot \frac{\partial L}{\partial o}. \quad (1.29)$$

Затем можно использовать многомерное цепное правило для записи аналогичной рекурсии:

$$\delta(h_r, o) = \frac{\partial L}{\partial a_{h_r}} = \sum_{h: h_r \Rightarrow h} \frac{\overbrace{\frac{\partial L}{\partial a_h}}^{\delta(h, o)}}{\underbrace{\frac{\partial a_h}{\partial a_{h_r}}}_{\Phi'(a_{h_r})w_{(h_r, h)}}} = \Phi'(a_{h_r}) \sum_{h: h_r \Rightarrow h} w_{(h_r, h)} \cdot \delta(h, o). \quad (1.30)$$

Именно в такой форме условие рекурсии чаще всего приводится в учебниках при обсуждении механизма обратного распространения ошибки. Зная $d(h_r, o)$, мы можем вычислить частную производную функции потерь по весу с помощью следующей формулы:

$$\frac{\partial L}{\partial w_{(h_{r-1}, h_r)}} = \delta(h_r, o) \cdot h_{r-1}. \quad (1.31)$$

Как и в случае однослойной сети, процесс обновления узлов осуществляется эпохами, на протяжении каждой из которых выполняется обработка всех тренировочных данных, пока не будет достигнута сходимость. Алгоритм обратного распространения ошибки подробно рассматривается в главе 3. В данной главе мы ограничимся лишь кратким обсуждением этой темы.

1.4. Практические аспекты тренировки нейронных сетей

Несмотря на то что нейронные сети пользуются репутацией универсальных аппроксиматоров функций, по-прежнему существуют значительные трудности, связанные с их фактической тренировкой, которая обеспечивала бы такой уровень их функциональности. Эти трудности в основном обусловлены некоторыми практическими проблемами процесса тренировки, наиболее важной из которых является проблема *переобучения* (overfitting).

1.4.1. Проблема переобучения

Суть проблемы переобучения заключается в том, что подгонка модели под конкретный тренировочный набор данных еще не гарантирует, что она будет хорошо предсказывать тестовые данные, которые ей прежде не встречались, даже если на тренировочных данных она идеально с этим справлялась. Иными словами, между качеством работы сети с тренировочными и тестовыми данными всегда существует брешь, размеры которой увеличиваются в случае сложных моделей и небольших наборов данных.

Чтобы понять природу описанного явления, рассмотрим простую однослойную сеть, для обучения которой вещественным целевым значениям из набора данных, имеющих пять атрибутов, мы будем использовать тождественную (линейную) функцию активации. Эта архитектура почти идентична той, которая представлена на рис. 1.3, если отвлечься от того, что в данном случае предсказываются вещественные значения. Поэтому сеть пытается обучить следующую функцию:

$$\hat{y} = \sum_{i=1}^5 w_i \cdot x_i. \quad (1.32)$$

Рассмотрим ситуацию, в которой наблюдаемое целевое значение является вещественным числом и всегда в два раза превышает значение первого атрибута, тогда как значения остальных атрибутов никак не связаны с целевым значением. В то же время мы имеем в своем распоряжении лишь четыре тренировочных примера, т.е. их количество на единицу меньше количества признаков (свободных параметров). Для конкретности предположим, что мы работаем со следующим набором тренировочных примеров:

x_1	x_2	x_3	x_4	x_5	y
1	1	0	0	0	2
2	0	1	0	0	4
3	0	0	1	0	6
4	0	0	0	1	8

В данном случае, учитывая известное соотношение между первым признаком и целевым значением, корректным вектором параметров будет вектор $\vec{W} = [2, 0, 0, 0, 0]$. Тренировочный набор обеспечивает нулевую ошибку для данного решения, хотя сеть должна *обучиться* ему на примерах, поскольку оно не задано нам априори. Однако при этом мы сталкиваемся с той проблемой, что количество тренировочных точек меньше количества параметров, а это означает, что в данном случае можно найти бесчисленное количество решений с нулевой ошибкой. Например, набор параметров $[0, 2, 4, 6, 8]$ также обеспечивает

нулевую ошибку на *тренировочных данных*. Но если мы применим это решение к неизвестным тестовым данным, то сеть, вероятнее всего, будет работать очень плохо, поскольку мы просто удачно подобрали параметры для одного случая и они не будут хорошо *обобщаться* на новые точки, т.е. обеспечивать целевое значение, в два раза превышающее значение первого атрибута, при произвольных значениях остальных атрибутов. К этому типу ложного вывода приводит малочисленность тренировочных данных, когда в модели кодируются случайные нюансы. В результате решение плохо обобщается на неизвестные тестовые данные. Эта ситуация почти аналогична обучению путем простого запоминания, которое характеризуется высокой предсказательной силой для тренировочных данных, но неспособно предсказывать правильный результат для неизвестных данных. Увеличение количества тренировочных примеров улучшает обобщающую способность модели, тогда как повышение сложности модели снижает ее способность к обобщению. В то же время, если набор тренировочных данных достаточно велик, то маловероятно, что слишком простой модели удастся уловить сложные соотношения, существующие между признаками и целевыми значениями. Простое практическое правило заключается в том, что количество тренировочных точек данных должно превышать количество параметров по крайней мере в 2-3 раза, хотя точное количество необходимых примеров определяется спецификой конкретной задачи. В общем случае о моделях с большим количеством параметров говорят как о моделях *большой мощности*, и для того, чтобы такая модель хорошо обобщалась на неизвестные тестовые данные, объемы данных, используемых для ее тренировки, также должны быть большими. Понятие переобучения тесно связано с достижением компромисса между смещением и дисперсией алгоритма машинного обучения. *Смещение* (bias) алгоритма является показателем того, насколько хорошо он аппроксимирует зависимость между данными и ответами, тогда как *дисперсия* (variance) алгоритма — это показатель степени разброса ответов в зависимости от изменения выборки данных. Суть вывода, который следует из анализа баланса этих показателей, заключается в том, что использование более мощных (т.е. менее *смещенных*) моделей не всегда обеспечивает выигрыш при работе с ограниченными объемами тренировочных данных в силу более высокой дисперсии таких моделей. Например, если вместо тренировочных данных, приведенных в таблице выше, мы используем другой набор, состоящий из четырех точек данных, то, скорее всего, получим в результате обучения сети совершенно другой набор параметров (основанный на случайных различиях между этими точками). Применение новой модели к *тому же тестовому примеру* приведет, скорее всего, к предсказаниям, которые будут отличаться от прежних, полученных при использовании первого тренировочного набора. Этот тип разброса предсказаний для одного и того же тестового примера и есть проявление *дисперсии модели*,

которая вносит свой вклад в ошибку. В конце концов, два разных предсказания, полученных для одного и того же тестового набора, не могут быть корректными одновременно. Недостатком сложных моделей является то, что они видят ложные закономерности в случайных нюансах, что становится особенно заметным при недостаточности тренировочных данных. Принимая решения относительно сложности модели, необходимо держаться золотой середины. Все эти вопросы подробно рассматриваются в главе 4.

Нейронные сети всегда считались инструментом, способным, по крайней мере теоретически, аппроксимировать любую функцию [208]. Но в случае недостаточности данных сеть может работать плохо; это и есть одна из причин того, почему нейронные сети приобрели широкую известность лишь в последнее время. Облегчение доступа к данным обнаружило преимущества нейронных сетей по сравнению с традиционным машинным обучением (см. рис. 1.2). В целом же нейронные сети должны тщательно проектироваться для минимизации отрицательных эффектов переобучения даже при наличии доступа к большим объемам данных. В этом разделе дается обзор некоторых методов проектирования, используемых для уменьшения влияния переобучения.

1.4.1.1. Регуляризация

Поскольку увеличение количества параметров повышает вероятность переобучения, вполне естественно ограничиваться использованием меньшего количества ненулевых параметров. Если в предыдущем примере ограничиться вектором \bar{W} , имеющим лишь одну ненулевую компоненту из пяти возможных, то это обеспечит получение корректного решения $[2, 0, 0, 0, 0]$. Уменьшение абсолютных значений параметров также способствует ослаблению эффектов переобучения. Ввиду трудностей ограничения значений параметров прибегают к более мягкому подходу, заключающемуся в добавлении штрафного члена $\lambda \|\bar{W}\|^p$ в используемую функцию потерь. Обычно p задают равным 2, что соответствует *регуляризации Тихонова*. В общем случае в целевую функцию добавляется квадрат значения каждого параметра (умноженный на параметр регуляризации $\lambda > 0$). На практике это изменение приводит к тому, что из обновления параметра w_i вычитается величина, пропорциональная lw_i . В качестве примера ниже приведена регуляризованная версия уравнения 1.6 для мини-пакета S и шага обновления $\alpha > 0$:

$$\bar{W} \leftarrow \bar{W}(1 - \alpha\lambda) + \alpha \sum_{\bar{X} \in S} E(\bar{X})\bar{X}. \quad (1.33)$$

Здесь $E(\bar{X})$ представляет текущую ошибку $(y - \hat{y})$, т.е. разницу между наблюдаемым и предсказанным значениями для тренировочного примера \bar{X} . Такой тип штрафования можно рассматривать как своего рода *ослабление веса* в

процессе обновления. Регуляризация особенно важна в условиях ограниченности доступного объема данных. Хорошей аналогией регуляризации из области биологии может служить процесс постепенного забывания, заключающийся в том, что “менее важные” (т.е. шум) связи между данными не сохраняются. В общем случае часто целесообразно использовать более сложные модели с регуляризацией, а не простые модели без регуляризации.

Попутно заметим, что общая форма уравнения 1.33 используется многими регуляризованными моделями машинного обучения, такими как *регрессия по методу наименьших квадратов* (глава 2), где вместо $E(\bar{X})$ подставляется функция ошибки, специфическая для данной модели. Интересен тот факт, что в однослойном перцептроне ослабление весов используют лишь в редких случаях³, поскольку иногда это может чрезмерно ускорять процесс забывания, если в векторе весов доминирует небольшое количество неверно классифицированных тренировочных точек. Основная проблема состоит в том, что критерий перцептрона (в отличие от линейно-кусочной или квадратичной функций потерь) уже сам по себе является вырожденной функцией потерь с минимальным значением 0 при $\bar{W} = 0$. Эта особенность является следствием того факта, что однослойный перцептрон первоначально был определен в терминах обновлений, имитирующих биологические нейроны, а не в терминах тщательно продуманной функции потерь. За исключением линейно-разделимых классов, о гарантиях сходимости к оптимальному решению речь никогда не шла. Обычно для однослойного перцептрона используются другие методы регуляризации, которые обсуждаются ниже.

1.4.1.2. Архитектура нейрона и разделение параметров

Наиболее эффективный подход к построению нейронной сети заключается в том, чтобы приступать к разработке ее архитектуры лишь после тщательного изучения особенностей базовых данных. Знание этих особенностей позволяет создавать специализированные архитектуры с меньшим количеством параметров. Помимо этого, многие параметры могут разделяться. Например, сверточная нейронная сеть использует один и тот же набор параметров для обучения характеристикам локального блока изображения. Последние достижения в разработке *рекуррентных и сверточных нейронных сетей* наглядно демонстрируют плодотворность такого подхода.

1.4.1.3. Раннее прекращение обучения

Другой распространенной формой регуляризации является *раннее прекращение обучения*, когда процесс градиентного спуска останавливают после

³ Обычно в однослойных моделях, а также в многослойных моделях с большим количеством параметров ослабление весов используется с другими функциями потерь.

нескольких итераций. Один из способов определения точки останова заключается в том, чтобы удерживать часть тренировочных данных, а затем тестировать ошибку модели на этом удержанном наборе. Процесс градиентного спуска прекращается, когда ошибка на удержанном наборе начинает возрастать. Раннее прекращение обучения снижает размер пространства параметров до меньшей окрестности в пределах начальных значений параметров. С этой точки зрения раннее прекращение обучения действует в качестве регуляризатора, поскольку оно в конечном счете ограничивает пространство параметров.

1.4.1.4. Достижение компромисса между шириной и глубиной

Как ранее уже обсуждалось, при наличии в скрытом слое большого количества скрытых элементов двухслойная нейронная сеть может использоваться в качестве универсального аппроксиматора функций [208]. Оказывается, что в случае нейронных сетей с еще большим количеством слоев (т.е. обладающих *большой глубиной*) обычно требуется меньшее количество элементов, приходящихся на один слой, поскольку композиционные функции, создаваемые последовательными слоями, повышают мощность нейронной сети. Увеличение глубины сети — это разновидность регуляризации, поскольку признаки в последующих слоях вынуждены подчиняться конкретному типу структуры, навязываемому предшествующими слоями. Усиление ограничений уменьшает мощность сети, что приносит пользу, если существуют ограничения на объем доступных данных. Краткое объяснение такого поведения приведено в разделе 1.5. Обычно количество элементов в каждом слое может быть уменьшено до такого уровня, что нередко глубокая сеть будет иметь гораздо меньшее количество параметров даже при увеличении количества слоев. Это наблюдение стало причиной бума исследований в области *глубокого обучения* (deep learning).

Несмотря на то что глубокие сети позволили снизить остроту проблемы переобучения, они привнесли проблемы другого рода, осложняющие обучение сети. В частности, производные функции потерь по весам в разных слоях имеют, как правило, значительно различающиеся величины, что затрудняет выбор наиболее оптимальных размеров шагов. Это нежелательное поведение порождает проблемы так называемых *затухающих* и *взрывных* градиентов. Кроме того, для достижения сходимости в глубоких сетях часто требуется недопустимо длительное время. Эти проблемы еще будут обсуждаться в этой и последующих главах.

1.4.1.5. Ансамблевые методы

Для повышения обобщающей способности модели используют различные ансамблевые методы, такие как метод *параллельной композиции алгоритмов*,

или *бэггинг* (англ. “bagging” от “bootstrap aggregating” — улучшенное агрегирование). Эти методы применимы не только к нейронным сетям, но и к любому типу алгоритмов машинного обучения. Однако в последние годы был предложен ряд ансамблевых методов, предназначенных для нейронных сетей. К их числу относятся метод исключения, или *дропаут* (dropout), и метод *прореживания связей* (dropconnect). Эти методы могут сочетаться со многими архитектурами нейронных сетей, обеспечивая дополнительное повышение точности около 2% во многих реальных конфигурациях. Однако повышение точности зависит от типа данных и природы базовой тренировки. Например, нормализация активации в скрытых слоях может снизить эффективность методов исключения, но при этом сама активация может обеспечить результирующий выигрыш. Ансамблевые методы обсуждаются в главе 4.

1.4.2. Проблемы затухающих и взрывных градиентов

В то время как увеличение глубины сети во многих случаях позволяет уменьшить количество параметров, это приводит к появлению ряда затруднений практического характера. Слабой стороной использования метода обратного распространения ошибки с помощью цепного правила в сетях с большим количеством слоев является нестабильность обновлений. В частности, в некоторых типах архитектур обновления в начальных слоях нейронной сети могут иметь либо пренебрежимо малую (затухающий градиент), либо непомерно большую (взрывной градиент) величину. В основном это обусловлено поведением члена в виде произведения в уравнении 1.23, который может либо экспоненциально возрасти, либо экспоненциально убывать с увеличением длины пути. Чтобы понять природу этого явления, рассмотрим ситуацию, когда имеется многослойная сеть с одним нейроном в каждом слое. Можно показать, что каждая локальная производная вдоль пути представляет собой произведение веса и производной функции активации. Общая производная для обратного распространения равна произведению этих значений. Если каждое значение подчиняется случайному распределению с математическим ожиданием меньше 1, то произведение этих производных в уравнении 1.23 будет экспоненциально затухать с увеличением длины пути. Если же математические ожидания индивидуальных значений вдоль пути больше 1, то в типичных случаях это приводит к “взрывному” поведению градиента, т.е. его безудержному росту. Даже если значения локальных производных подчинены случайному распределению с математическим ожиданием, в точности равным единице, то общая производная, как правило, будет демонстрировать нестабильное поведение, в зависимости от фактического распределения указанных значений. Другими словами, возникновение проблем затухающих и взрывных градиентов — довольно естественное явление для глубоких сетей, что делает процесс их тренировки нестабильным.

Для преодоления этих проблем было предложено не одно решение. Так, проблема затухающих градиентов характерна для сигмоиды, поскольку ее производная меньше 0,25 при всех значениях аргумента (упражнение 7) и имеет предельно малое значение в области насыщения. Элемент активации ReLU менее подвержен проблеме затухающего градиента, поскольку его производная всегда равна 1 для положительных значений аргумента. Более подробно эти вопросы обсуждаются в главе 3. Помимо использования ReLU, для улучшения сходимости градиентного спуска существует множество других способов. В частности, во многих случаях для этого применяют *адаптивные скорости обучения* и *методы сопряженных градиентов*. Кроме того, для преодоления некоторых из описанных выше проблем можно воспользоваться *пакетной нормализацией* (batch normalization). Обо всем этом речь пойдет в главе 3.

1.4.3. Трудности со сходимостью

Добиться достаточно быстрой сходимости процесса оптимизации в случае очень глубоких сетей довольно трудно, поскольку увеличение глубины сети повышает ее сопротивляемость процессу тренировки, нарушая плавное перетекание градиента по сети. Эта проблема в некотором смысле родственна проблеме затухающего градиента, но имеет собственные уникальные характеристики. Специально для таких случаев в литературе был предложен ряд “трюков”, включая использование *управляемых (вентильных) сетей* (gating networks) и *остаточных сетей* (residual networks) [184]. Эти методы обсуждаются в главах 7 и 8 соответственно.

1.4.4. Локальные и ложные оптимумы

Оптимизируемая функция нейронной сети является в высшей степени нелинейной и имеет множество локальных оптимумов. В случае большого пространства параметров с многочисленными локальными оптимумами имеет смысл потратить некоторое время на выбор начальных точек. Одним из способов улучшения инициализации параметров нейронной сети является *предварительное обучение* (pretraining). Основная идея заключается в обучении (с учителем или без учителя) *мелких подсетей* исходной сети для создания начальных значений весов. Этот тип предварительного обучения применяется *последовательно в жадной манере*, т.е. все слои тренируются поочередно для обучения начальным значениям своих параметров. Такой подход позволяет избежать попадания начальных точек в явно неподходящие области пространства параметров. Кроме того, предварительное обучение без учителя часто способствует устранению проблем переобучения. Основная идея заключается в том, что некоторые минимумы функции потерь являются ложными оптимумами, поскольку они проявляются лишь на тренировочных данных и не проявляются на тестовых. Как

правило, обучение без учителя смещает начальную точку в сторону бассейнов аттракции “хороших” оптимумов для тестовых данных. Этот вопрос также связан с обобщаемостью модели. Методы предварительного обучения обсуждаются в разделе 4.7.

Интересно отметить, что понятие ложного минимума часто рассматривают сквозь призму способности модели нейронной сети к обобщению. Ход связанных с этим рассуждений отличается от того, который принят в традиционных методах оптимизации. Традиционные методы оптимизации фокусируются не на различиях в функциях потерь тренировочных и тестовых данных, а только на форме функции потерь тренировочных данных. Как это ни удивительно, проблема локальных оптимумов (с точки зрения традиционных подходов) в нейронных сетях в действительности не столь серьезна, как мы могли бы ожидать от таких нелинейных функций. В большинстве случаев нелинейность порождает проблемы, связанные с собственно процессом тренировки (например, отсутствие сходимости), а не с “застреванием” в локальном минимуме.

1.4.5. Вычислительные трудности

Важным фактором, который следует учитывать при проектировании нейронной сети, является время, необходимое для ее тренировки. В области распознавания текстов и изображений сети нередко тренируют неделями. Достигнутый в последние годы прогресс в разработке такого оборудования, как графические процессоры (GPU), способствовал значительному прорыву в данном направлении. GPU — это специализированные процессоры, позволяющие резко ускорить выполнение операций, типичных для нейронных сетей. В этом смысле особенно удобными оказываются такие алгоритмические фреймворки, как *Torch*, поскольку они предоставляют поддержку GPU, тесно интегрированную в платформу.

Несмотря на то что прогресс в разработке алгоритмов сыграл свою роль в возбуждении интереса к глубокому обучению, даже прежние алгоритмы способны дать нам гораздо больше, если они выполняются на современном оборудовании. В свою очередь, более скоростное оборудование стимулирует разработку алгоритмов, поскольку этот процесс изначально требует многократного выполнения интенсивных в вычислительном отношении тестов. Например, последний вариант такой модели, как *долгая краткосрочная память* (long short-term memory), претерпел лишь умеренные изменения [150] с тех пор, как в 1997 году был предложен ее первоначальный вариант [204]. И все же потенциальные возможности этой модели были распознаны лишь недавно благодаря повышению вычислительной мощности современных компьютеров и появлению алгоритмических новинок, разработка которых стала возможной в силу создания лучших условий для проведения экспериментов.

Удобным свойством преобладающего большинства моделей нейронных сетей является то, что большая часть тяжелой вычислительной работы переносится в фазу тренировки, тогда как для фазы предсказания нередко характерна более высокая эффективность в вычислительном отношении, поскольку она требует выполнения меньшего количества операций (в зависимости от количества слов). Это очень важный момент, поскольку фактор времени нередко более критичен для фазы предсказания, чем для фазы тренировки. Например, очень важно иметь возможность классифицировать изображение в режиме реального времени (с помощью предварительно созданной модели), даже если для фактического создания этой модели потребуется несколько недель обрабатывать миллионы изображений. Также были разработаны методы сжатия обученных сетей, обеспечивающие возможность их развертывания на мобильных и миниатюрных устройствах. Эти вопросы обсуждаются в главе 3.

1.5. Преимущества композиции функций

Несмотря на всю соблазнительность попыток привлечения биологических метафор для того, чтобы попытаться понять на интуитивном уровне, в чем кроется секрет удивительных вычислительных возможностей нейронных сетей, они не позволяют нам получить полную картину ситуаций, в которых нейронные сети хорошо работают. На самом низком уровне абстракции нейронную сеть можно рассматривать как вычислительный граф, комбинирующий простые функции для получения более сложной функции. Мощь глубокого обучения во многом объясняется тем фактом, что *повторная* композиция многочисленных нелинейных функций обладает значительной выразительной силой. Несмотря на то что в работе [208] было показано, что однократная композиция большого количества *функций сжатия* позволяет аппроксимировать почти любую функцию, такой подход требует использования чрезвычайно большого количества элементов (а значит, и параметров) сети. Это увеличивает емкость сети, что приводит к переобучению, избавиться от которого можно только предельным увеличением размера набора данных. Мощь глубокого обучения в значительной степени проистекает от того, что *многократная композиция определенных типов функций повышает репрезентативную способность сети, тем самым уменьшая размеры пространства параметров, подлежащих обучению*.

Не все базовые функции в одинаковой степени пригодны для достижения этой цели. В действительности нелинейные функции сжатия, используемые в нейронных сетях, не выбираются произвольно, а тщательно конструируются с учетом определенных типов свойств. Представьте, например, ситуацию, когда во всех слоях используется тождественная функция активации, так что вычисляются только линейные функции. В этом случае результирующая нейронная сеть будет не мощнее однослойной линейной сети.

Теорема 1.5.1. *Многослойная сеть, во всех слоях которой используется только тождественная функция активации, сводится к однослойной сети, выполняющей линейную регрессию.*

Доказательство. Рассмотрим сеть, которая содержит k скрытых слоев, а значит, в общей сложности $(k + 1)$ вычислительных слоев (включая выходной слой). Соответствующие $(k + 1)$ матриц весов связей между последовательными слоями обозначим как $W_1 \dots W_{k+1}$. Пусть \bar{x} — d -мерный вектор-столбец, соответствующий входу, $\bar{h}_1 \dots \bar{h}_k$ — вектор-столбцы, соответствующие скрытым слоям, а \bar{o} — m -мерный вектор-столбец, соответствующий выходу. Тогда для многослойных сетей справедливо следующее условие рекурсии:

$$\begin{aligned}\bar{h}_1 &= \Phi(W_1^T \bar{x}) = W_1^T \bar{x}, \\ \bar{h}_{p+1} &= \Phi(W_{p+1}^T \bar{h}_p) = W_{p+1}^T \bar{h}_p \quad \forall p \in \{1 \dots k-1\}, \\ \bar{o} &= \Phi(W_{k+1}^T \bar{h}_k) = W_{k+1}^T \bar{h}_k.\end{aligned}$$

Во всех вышеприведенных случаях в качестве функции активации $\Phi(\cdot)$ устанавливается тождественная функция. Путем исключения переменных скрытых слоев нетрудно получить следующее соотношение:

$$\begin{aligned}\bar{o} &= W_{k+1}^T W_k^T \dots W_1^T \bar{x} = \\ &= \underbrace{(W_1 W_2 \dots W_{k+1})^T}_{W_{xo}^T} \bar{x}.\end{aligned}$$

Заметьте, что матрицу $W_1 W_2 \dots W_{k+1}$ можно заменить новой матрицей W_{xo} размера $d \times m$ и обучать коэффициенты матрицы W_{xo} , а не коэффициенты всех матриц $W_1 W_2 \dots W_{k+1}$, не теряя выразительности. Другими словами, получаем следующее соотношение:

$$\bar{o} = W_{xo}^T \bar{x}.$$

Однако это условие в точности идентично условию линейной регрессии с множественными выходами [6]. В действительности обучать сеть многочисленными матрицам $W_1 W_2 \dots W_{k+1}$, а не матрице W_{xo} , — плохая идея, поскольку это увеличивает количество обучаемых параметров, но при этом никак не увеличивает мощность сети. Таким образом, многослойная нейронная сеть с тождественными функциями активации не дает никакого выигрыша в смысле выразительности по сравнению с однослойной сетью. Мы сформулировали этот результат для модели линейной регрессии с числовыми целевыми переменными. Аналогичный результат также справедлив для бинарных целевых переменных. В том специальном случае, когда во всех слоях используется тождественная функция активации, а в последнем слое для предсказаний используется единственный

выход со знаковой функцией активации, многослойная нейронная сеть сводится к перцептрон.

Лемма 1.5.1. *Рассмотрим многослойную сеть, во всех скрытых слоях которой используется тождественная активация, а единственный выходной узел использует критерий перцептрона в качестве функции потерь и знако-вую функцию для получения предсказаний. Тогда такая нейронная сеть сводит-ся к однослойному перцептрон.*

Данное утверждение доказывается почти так же, как и предыдущее. Факти-чески, при условии что скрытые слои являются линейными, введение допол-нительных слоев не дает никакого выигрыша.

Этот результат показывает, что в целом использовать глубокие сети имеет смысл только тогда, когда функции активации в промежуточных слоях являют-ся нелинейными. В качестве типичных примеров *функций сжатия* (squashing functions), которые “втискивают” выход в ограниченный интервал значений и градиенты которых максимальны вблизи нулевых значений, можно привести сигмоиду и гиперболический тангенс. При больших абсолютных значениях своих аргументов эти функции, как говорят, достигают *насыщения* в том смыс-ле, что дальнейшее увеличение абсолютной величины аргумента не приводит к сколь-нибудь существенному изменению значения функции. Этот тип функ-ций, значения которых не изменяются существенно при больших абсолютных значениях аргументов, разделяется другим семейством функций, так называе-мыми *гауссовскими ядрами* (Gaussian kernels), которые обычно используются в непараметрических оценках плотности распределения:

$$\Phi(v) = \exp(-v^2 / 2). \quad (1.34)$$

Единственным отличием гауссовских ядер является то, что они насыщают-ся до нуля при больших значениях своих аргументов, в то время как сигмоида и гиперболический тангенс могут насыщаться до значений +1 и -1. В литера-туре, посвященной оценкам плотности, хорошо известен тот факт, что сумма многих небольших гауссовских ядер может быть использована для аппрокси-мации любой функции плотности [451]. Функции плотности имеют специаль-ную неотрицательную структуру, в которой экстремумы распределения данных всегда насыщаются до нулевой плотности, и поэтому то же поведение демон-стрируют и базовые ядра. В отношении функций сжатия действует (в более об-щем смысле) аналогичный принцип, в соответствии с которым любую функцию можно аппроксимировать некоторой линейной комбинацией небольших функ-ций активации. В то же время функции сжатия не насыщаются до нуля, что дает возможность обрабатывать произвольное поведение при экстремальных зна-чениях. Универсальная формулировка возможностей аппроксимации функций

нейронными сетями [208] гласит, что использование линейных комбинаций сигмоидных элементов (и/или большинства других разумных функций сжатия) в одном скрытом слое обеспечивает удовлетворительную аппроксимацию любой функции. Обратите внимание на то, что такая линейная комбинация может формироваться в одиночном выходном узле. Таким образом, при условии что количество скрытых элементов довольно велико, для этого вполне достаточно использовать двухслойную сеть. Однако при этом, чтобы сохранить возможность моделирования любых особенностей поведения произвольной функции, активационная функция всегда должна предоставлять базовую нелинейность в той или иной форме. Чтобы разобраться в том, почему это так, заметим, что все одномерные функции могут аппроксимироваться суммой масштабированных/смещенных ступенчатых функций, а большинство функций активации, которые обсуждались в этой главе (например, сигмоида), весьма напоминают ступенчатые функции (см. рис. 1.8). Эта базовая идея выражает суть универсальной теоремы об аппроксимации функций с помощью нейронных сетей. В действительности доказательство способности функций сжатия аппроксимировать любую функцию концептуально аналогично такому же доказательству в отношении гауссовских ядер (по крайней мере на интуитивном уровне). В то же время требуемое количество базовых функций, обеспечивающее высокую точность аппроксимации, в обоих случаях может быть очень большим, тем самым повышая требования, относящиеся к данным, до уровня, делающего их невыполнимыми. По этой причине мелкие сети сталкиваются с извечной проблемой переобучения. Универсальная теорема об аппроксимируемости функций утверждает о возможности удовлетворительной аппроксимации функции, неявно заданной тренировочными данными, но не дает никаких гарантий относительно того, что эта функция будет обобщаться на неизвестные тестовые данные.

1.5.1. Важность нелинейной активации

В предыдущем разделе было предоставлено непосредственное доказательство того факта, что увеличение количества слоев в нейронной сети, в которой используются только линейные функции активации, не дает никакого выигрыша. В качестве примера рассмотрим двухклассовый набор данных, представленный на рис. 1.14 в двух измерениях, обозначенных как x_1 и x_2 . Этот набор данных включает два экземпляра, A и B, с координатами (1, 1) и (-1, 1) соответственно, принадлежащих к классу, обозначенному символом *. Также существует единственный экземпляр B с координатами (0, 1), принадлежащий к классу, обозначенному символом +. Нейронная сеть, использующая только линейные активации, ни при каких условиях не сможет идеально классифицировать тренировочные данные, поскольку между точками, относящимися к разным классам, невозможно провести разделяющую их прямую линию.

С другой стороны, рассмотрим ситуацию, в которой скрытые элементы используют активацию ReLU и обучаются двум новым признакам h_1 и h_2 следующим образом:

$$h_1 = \max\{x_1, 0\},$$

$$h_2 = \max\{-x_1, 0\}.$$

Обратите внимание на то, что обе эти цели могут быть достигнуты за счет использования подходящих весов связей, соединяющих вход со скрытым слоем, и применения активации ReLU. Последняя из этих мер устанавливает порог, обрезающий отрицательные значения до нуля. Соответствующие веса связей указаны на рис. 1.14.

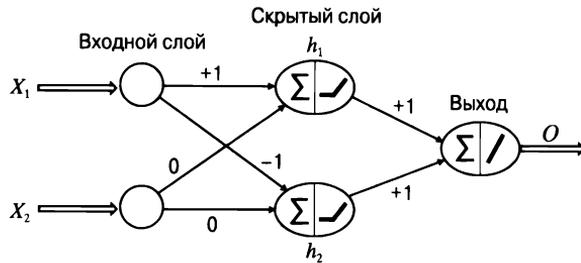
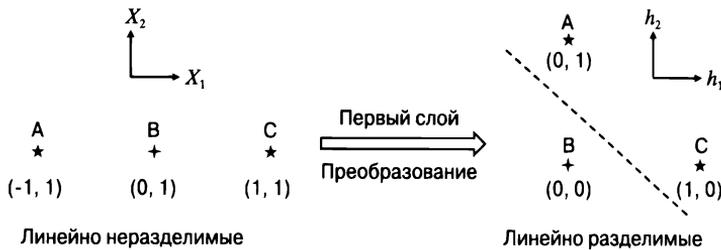


Рис. 1.14. Преобразование линейно неразделимого набора данных в линейно разделимый с помощью нелинейных функций активации

Мы отобразили те же данные на том же рисунке, используя координаты h_1 и h_2 . Координаты трех точек в двухмерном скрытом слое равны $\{(1, 0), (0, 1), (0, 0)\}$. Это сразу делает очевидным тот факт, что в терминах нового скрытого представления два класса становятся линейно разделимыми. В определенном смысле задача первого слоя заключалась в том, чтобы обеспечить обучение представлению, позволяющему получить решение с помощью линейного классификатора. Поэтому, если мы добавим в нейронную сеть единственный линейный выходной слой, то он будет идеально классифицировать наши тренировочные примеры. Функции активации выполняют нелинейное преобразование,

обеспечивающее отображение точек данных в линейно разделимый набор. Фактически, если установить веса обеих связей, соединяющих скрытый слой с выходным, равными 1, и применить линейную функцию активации, то выход O будет определяться следующей формулой:

$$O = h_1 + h_2. \quad (1.35)$$

Эта простая линейная функция разделяет два класса, поскольку она всегда принимает значение 1 для обеих точек, обозначенных символом $*$, и значение 0 для точки, обозначенной символом $+$. Поэтому мощь нейронных сетей в значительной степени заключается в использовании функций активации. *Обучение весов*, приведенных на рис. 1.14, должно осуществляться в ходе процесса, *управляемого данными*, хотя существует много других возможных вариантов выбора весов, способных сделать скрытое представление линейно разделимым. Поэтому веса, полученные в результате фактического выполнения тренировки, могут отличаться от тех, которые приведены на рис. 1.14. Тем не менее в случае перцептрона нельзя надеяться на существование таких весов, которые обеспечили бы идеальную классификацию этого тренировочного набора данных, поскольку он не разделяется линейно в исходном пространстве. Другими словами, функции активации делают возможными такие нелинейные преобразования данных, мощность которых увеличивается с увеличением количества слоев. Последовательность нелинейных преобразований диктует определенный тип структуры обучаемой модели, мощность которой увеличивается с увеличением глубины последовательности (т.е. количества слоев нейронной сети).

Другим классическим примером является функция *XOR*, в которой две точки $\{(0, 0), (1, 1)\}$ относятся к одному классу, а другие две точки $\{(1, 0), (0, 1)\}$ — к другому. Для разделения этих двух классов также можно использовать активацию ReLU, хотя в этом случае потребуются нейроны смещения (см. уравнение 1). Функция *XOR* обсуждается в оригинальной статье, посвященной обратному распространению ошибки [409], поскольку она была одним из факторов, стимулирующих разработку многослойных сетей и способов их тренировки. Функция *XOR* считается лакмусовой бумажкой, позволяющей определить принципиальную возможность предсказания линейно неразделимых классов с помощью конкретного семейства нейронных сетей. Несмотря на то что выше для простоты использовалась функция активации ReLU, для достижения тех же целей можно использовать большинство других активационных функций.

1.5.2. Снижение требований к параметрам с помощью глубины

Основой глубокого обучения служит идея о том, что повторная композиция функций часто может снижать требования к количеству базовых функций

(вычислительных элементов), обеспечивая его экспоненциальное уменьшение с увеличением количества слоев в сети. Поэтому, даже если количество слоев в сети увеличивается, количество параметров, необходимых для аппроксимации той же функции, резко уменьшается. В результате увеличивается и обобщающая способность сети.

Идея углубления архитектур основана на тех соображениях, что это обеспечивает более эффективное использование повторяющихся закономерностей в распределении данных для уменьшения количества вычислительных элементов, а значит, и для обобщения результатов обучения даже на области пространства данных, не охватываемые предоставленными примерами. Во многих случаях нейронные сети обучаются этим закономерностям путем использования весов в качестве базисных векторов иерархических признаков. И хотя подробное доказательство [340] этого факта выходит за рамки книги, мы предоставим пример, проливающий определенный свет на эту точку зрения. Рассмотрим ситуацию, в которой одномерная функция определяется с помощью 1024 повторяющихся ступеней, имеющих одну и ту же ширину и высоту. Мелкой сети с одним скрытым слоем и ступенчатой функцией активации для моделирования этой функции потребуется 1024 элемента. Однако многослойная сеть будет моделировать 1 ступень в первом слое, 2 ступени в следующем, 4 ступени в третьем и 2^r ступеней в r -м слое (рис. 1.15). Обратите внимание на то, что простейшим признаком является шаблон в виде одной ступени, поскольку он повторяется 1024 раза, тогда как шаблон в виде двух ступеней — более сложный. Поэтому признаки (и изучаемые сетью функции) в последовательных слоях иерархически связаны между собой. В данном случае для моделирования присоединения этих двух шаблонов из предыдущего слоя потребуется в общей сложности 10 слоев и небольшое количество постоянных узлов в каждом слое.

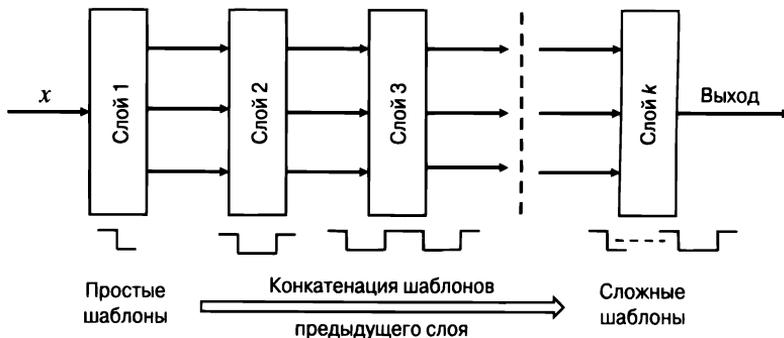


Рис. 1.15. Более глубокие сети могут обучаться более сложным функциям за счет композиции функций, которым были обучены предыдущие слои

Еще один способ, позволяющий подойти к этому вопросу с другой стороны, заключается в следующем. Рассмотрим одномерную функцию, которая принимает одно из значений, 1 и -1 , в чередующихся интервалах, и это значение переключается 1024 раза через регулярные интервалы аргумента. Единственная возможность имитировать такую функцию с помощью ступенчатых функций активации (содержащих лишь один переключатель в значении) — это использовать 1024 таких функции (или небольшое кратное от этого количества). Однако в нейронной сети с 10 скрытыми слоями и всего лишь 2 элементами в каждом слое существует $2^{10} = 1024$ пути от источника к выходу. Если функция, которой должна обучиться сеть, обладает той или иной регулярностью, то часто существует возможность обучить параметры слоев таким образом, чтобы эти 1024 пути смогли вобрать в функцию всю сложность 1024 различных переключателей значений. Ранние слои обучаются более детальным шаблонам, более поздние — более высокоуровневым шаблонам. Поэтому общее количество требуемых узлов *на порядок меньше* того, которое потребовалось бы для однослойной сети. Это означает, что объем необходимых для обучения данных также на порядок меньше. Причина заключается в том, что многослойная сеть неявно *ищет повторяющиеся регулярности и обучается им* на меньшем объеме данных, чем если бы она пыталась явно обучаться каждому повороту и изгибу целевой функции. Такое поведение становится интуитивно очевидным при использовании сверточных нейронных сетей для обработки изображений, когда начальные слои моделируют простые признаки наподобие линий, средний слой — элементарные формы, а последующий — сложные формы наподобие лица. С другой стороны, одиночный слой испытывал бы трудности при моделировании каждой черты лица. Это наделяет более глубокую модель улучшенной способностью к обобщению, а также способностью к обучению на меньших объемах данных.

Однако увеличение глубины сети не лишено недостатков. Более глубокие сети часто труднее поддаются обучению и подвержены всем типам нестабильного поведения, в том числе проблемам затухающих и взрывных градиентов. Глубокие сети также проявляют явную нестабильность в отношении выбора параметров. Эти проблемы чаще всего удается устранить за счет тщательного продумывания функций, вычисляемых в узлах, а также использования процедур предварительного обучения для повышения производительности сети.

1.5.3. Нестандартные архитектуры нейронных сетей

Выше был дан обзор наиболее распространенных способов конструирования и организации типичных нейронных сетей. Однако существует много вариаций этих способов, которые обсуждаются ниже.

1.5.3.1. Размытие различий между входными, скрытыми и выходными слоями

Вообще говоря, при обсуждении структуры нейронных сетей основное внимание уделяется главным образом сетям прямого распространения с послойной организацией и последовательным расположением входных, скрытых и выходных слоев. Другими словами, все входные узлы передают данные первому скрытому слою, скрытые слои последовательно передают данные друг другу, а последний скрытый слой передает данные выходному слою. Вычислительные элементы часто определяют как *функции сжатия* (squashing functions), применяемые к линейным комбинациям входа. Обычно скрытый слой не принимает входные данные, а функция потерь обычно вычисляется без учета значений в скрытых слоях. Из-за этого смещения акцентов легко забыть о том, что нейронная сеть может быть определена в виде *параметризованного вычислительного графа любого типа*, не требующего этих ограничений для того, чтобы работал механизм обратного распространения ошибки. Вообще говоря, вполне возможно, чтобы промежуточные слои включали входы или принимали участие в вычислении функции потерь, хотя такие подходы менее распространены. Например, была предложена [515] сеть, идея которой была подсказана *случайными лесами* [49] и которая допускает наличие входов в разных слоях. Один из примеров такой сети приведен на рис. 1.16. В данном случае размытие различий между входными и скрытыми слоями становится очевидным.

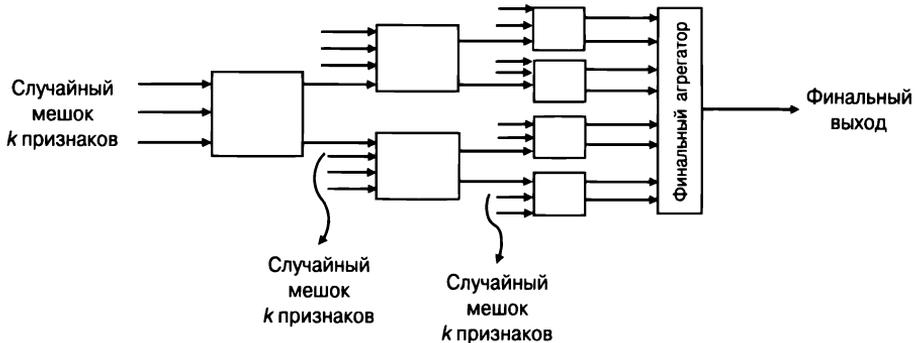


Рис. 1.16. Пример нестандартной архитектуры, в которой входы передают данные слоям, отличным от первого скрытого слоя. При условии что нейронная сеть не содержит циклов (или может быть преобразована в ациклическое представление), веса базового вычислительного графа могут обучаться с использованием динамического программирования (обратного распространения ошибки)

В других вариациях базовой архитектуры прямого распространения функции потерь вычисляются не только в выходных узлах, но и в скрытых. Вклады в скрытых узлах часто принимают форму *штрафов*, действующих в качестве ре-

гуляризаторов. Например, методы этого типа используются для обучения разреженным признакам посредством штрафования скрытых узлов (главы 2 и 4). В данном случае размываются различия между скрытыми и выходными слоями.

В качестве еще одного примера предложенных недавно вариантов проектирования нейронных сетей можно привести архитектуру с *замыкающими соединениями* (skip connections) [184], в которой входы одного слоя могут напрямую соединяться с другими слоями, расположенными далее слоя, непосредственно следующего за данным. Такой подход порождает по-настоящему глубокие модели. Например, 152-слойная архитектура, получившая название *ResNet* [184], обеспечила эффективность работы сети на уровне человека в задачах распознавания образов. И хотя эта архитектура не размывает различия между входными, скрытыми и выходными словами, ее структура отличается от структуры традиционных сетей прямого распространения, в которых допустимы только соединения между смежными последовательными слоями. В сетях этого типа используется итеративный подход к *конструированию признаков* (feature engineering) [161], в соответствии с которым признаки в поздних слоях представляют собой итеративные уточнения признаков в ранних слоях. В противоположность этому традиционный подход к конструированию признаков — иерархический, т.е. признаки в более поздних слоях представляют собой все более абстрактные представления признаков из предыдущих слоев.

1.5.3.2. Необычные операции и сети сумм и произведений

Некоторые нейронные сети наподобие сетей с ячейками *долгой краткосрочной памяти* (long short-term memory — LSTM) и сверточных нейронных сетей определяют различные типы операций мультипликативного “забывания”, свертки и пулинга, выполняемые над переменными, существующими не строго в тех формах, которые обсуждались в этой главе. В настоящее время такие архитектуры настолько интенсивно используются при обработке текста и изображения, что они больше не рассматриваются как необычные.

Еще одним уникальным типом архитектуры является *сеть сумм и произведений* (sum-product network) [383]. В данном случае узлы являются либо узлами суммирования, либо узлами умножения. Узлы суммирования аналогичны традиционным линейным преобразованиям, выполняемым над набором взвешенных ребер вычислительных графов. Однако веса могут иметь только положительные значения. Узлы умножения просто умножают свои входы без использования весов. Следует отметить, что существует несколько вариаций этих узлов, различающихся способом вычисления произведения. Например, если входы — это два скаляра, то можно вычислить их произведение. Если входы — это два вектора равной длины, то можно вычислить их поэлементное произведение. Некоторые библиотеки глубокого обучения поддерживают вычисление произведений такого типа. Вполне естественно, что для максимизации

выразительной способности сети слои суммирования могут чередоваться со слоями умножения.

Сети сумм и произведений довольно выразительны, и часто возможно создание их глубоких вариаций с высокой степенью выразительности [30, 93]. Очень важно то, что любая математическая функция может быть аппроксимирована полиномиальной функцией своих входов. Поэтому почти любая функция может быть выражена с помощью архитектуры сумм и произведений, хотя более глубокие архитектуры допускают моделирование с использованием более развитых структур. В отличие от традиционных нейронных сетей, в которых нелинейность встраивается с помощью функций активации, в сетях сумм и произведений ключом к нелинейности является операция умножения.

Проблемы обучения

Используя различные типы вычислительных операций в узлах, часто целесообразно проявлять гибкость и выходить за рамки известных преобразований и функций активации. Кроме того, соединения между узлами не обязательно должны структурироваться послойно, а узлы в скрытых слоях могут включаться в вычисление функции потерь. Если базовый вычислительный граф является ациклическим, то можно легко обобщить алгоритм обратного распространения ошибки на любой тип архитектуры и любые вычислительные операции. В конце концов, алгоритм динамического программирования (наподобие обратного распространения ошибки) можно использовать практически на любом типе направленного ациклического графа, в котором для инициализации динамической рекурсии можно использовать несколько узлов. Очень важно иметь в виду, что архитектуры, спроектированные на основе понимания особенностей конкретной предметной области, часто могут обеспечивать результаты, превосходящие результаты, полученные методами черного ящика, в которых используются полностью связанные сети прямого распространения.

1.6. Распространенные архитектуры нейронных сетей

Существует несколько типов нейронных архитектур, применяемых в задачах машинного обучения. В этом разделе дается краткий обзор архитектур, которые будут более подробно обсуждаться в следующих главах.

1.6.1. Имитация базового машинного обучения с помощью мелких моделей

Большинство базовых моделей машинного обучения, таких как линейная регрессия, классификация, метод опорных векторов, логистическая регрессия,

сингулярное разложение и матричная факторизация, можно имитировать с помощью мелких нейронных сетей, содержащих не более одного-двух слоев. Базовые архитектуры такого типа заслуживают изучения, поскольку это будет косвенной демонстрацией возможностей нейронных сетей. Большую часть того, что мы знаем о машинном обучении, можно сымитировать с помощью относительно простых моделей! Кроме того, многие базовые модели нейронных сетей наподобие *модели обучения Уидроу — Хоффа* (Widrow–Hoff learning model) самым непосредственным образом связаны с традиционными моделями машинного обучения, такими как дискриминант Фишера, хотя они и были предложены независимо. Следует отметить один важный момент: глубокие архитектуры часто создаются путем продуманного образования стеков более простых моделей. Нейронные модели для базовых моделей машинного обучения обсуждаются в главе 2. Там же будут рассматриваться приложения для работы с текстом и рекомендательные системы.

1.6.2. Сети радиально-базисных функций

Сети *радиально-базисных функций* (radial basis function — RBF) представляют одну из забытых архитектур, которыми богата история нейронных сетей. Эти архитектуры редко встречаются в наши дни, хотя и обладают значительным потенциалом применительно к некоторым типам задач. Одним из лимитирующих факторов является то, что они не обладают глубиной и, как правило, насчитывают всего лишь два слоя. Первый строй конструируется с расчетом на самообучение, тогда как второй тренируется с использованием методов обучения с учителем. Эти сети фундаментально отличаются от сетей прямого распространения, и источник их возможностей кроется в большом количестве узлов в слое обучения без учителя. Базовые принципы использования RBF-сетей сильно отличаются от принципов использования сетей прямого распространения в том смысле, что усиление их возможностей достигается за счет увеличения размера пространства признаков, а не глубины. Этот подход базируется на *теореме Ковера о разделимости образов (шаблонов)* [84], которая утверждает, что задачи классификации образов линейризуются легче после перевода их в многомерное пространство с помощью нелинейного преобразования. Каждый узел второго слоя сети содержит прототип, и активация определяется на основе сходства входных данных и прототипа. Затем эти активации комбинируются с обученными весами следующего слоя для создания окончательного прогноза. Такой подход очень напоминает классификацию по методу ближайших соседей, за исключением того, что веса во втором слое обеспечивают дополнительный уровень управления процессом обучения. Другими словами, данный подход представляет собой *контролируемый метод ближайших соседей* (supervised nearest-neighbor method).

В частности, известно, что реализации метода опорных векторов представляют собой контролируемые варианты классификаторов по методу ближайших соседей, в которых ядерные функции комбинируются с обучаемыми весами для взвешивания соседних точек в окончательном прогнозе [6]. Сети радиальных базисных функций могут использоваться для имитации ядерных методов, таких как метод опорных векторов. В некоторых типах задач, таких как классификация, эти архитектуры могут работать эффективнее готовых ядер. Причина заключается в большей общности этих моделей, что предоставляет больше возможностей для экспериментов. Кроме того, иногда удается получить дополнительный выигрыш за счет увеличенной глубины обучаемых слоев. Полный потенциал сетей радиальных базисных функций все еще остается не до конца исследованным в литературе, поскольку эта архитектура была в значительной степени забыта, и исследователи сосредоточили основное внимание на методах прямого распространения. Сети радиально-базисных функций обсуждаются в главе 5.

1.6.3. Ограниченные машины Больцмана

В ограниченных машинах Больцмана (restricted Boltzmann machine — RBM) для создания архитектур нейронных сетей, предназначенных для моделирования данных на основе самообучения, используется понятие энергии, которая подлежит минимизации. Эти методы особенно полезны при создании генеративных моделей и тесно связаны с вероятностными графическими моделями [251]. Ограниченные машины Больцмана берут свое начало в *сетях Хопфилда* [207], которые могут использоваться для сохранения ранее полученной информации. Результатом обобщения стохастических вариантов этих сетей явились *машины Больцмана*, в которых скрытые слои моделировали генеративные аспекты данных.

Ограниченные машины Больцмана часто применяются в обучении без учителя и для снижения размерности данных, хотя они также могут использоваться для обучения с учителем. В то же время, поскольку они не были изначально приспособлены для обучения с учителем, тренировке с учителем часто предшествовала фаза тренировки без учителя. Это естественным образом привело к идее предварительного обучения, которая оказалась чрезвычайно плодотворной для обучения с учителем. RBM были одними из первых моделей, которые начали применяться в глубоком обучении, особенно без учителя.

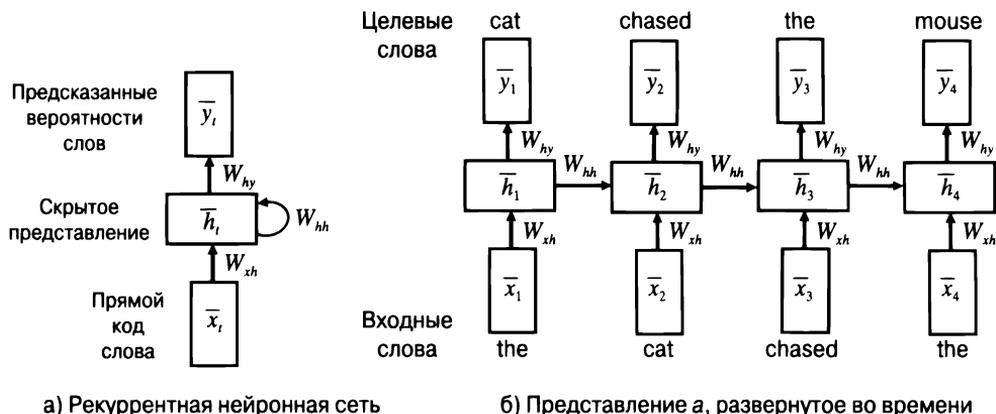
В конечном счете подход, основанный на предварительном обучении, был перенесен на другие типы моделей. Поэтому архитектура RBM имеет еще и историческое значение в том смысле, что она инициировала разработку некоторых методологий тренировки для глубоких моделей. Процесс тренировки ограниченной машины Больцмана заметно отличается от процесса тренировки сетей

прямого распространения. В частности, эти модели нельзя тренировать с использованием механизма обратного распространения ошибки, и для их тренировки необходимо прибегать к семплированию данных по методу Монте-Карло. Обычно для тренировки RBM используют конкретный алгоритм, а именно так называемый *алгоритм контрастивной дивергенции* (contrastive divergence algorithm). Ограниченные машины Больцмана обсуждаются в главе 6.

1.6.4. Рекуррентные нейронные сети

Рекуррентные нейронные сети предназначены для работы с последовательными данными, такими как текстовые цепочки, временные ряды и другие дискретные последовательности. В таких случаях вход имеет форму $\bar{x}_1 \dots \bar{x}_n$, где \bar{x}_t — d -мерная точка в момент времени t . Например, вектор x_t может содержать d значений, соответствующих t -му отсчету (моменту времени t) многомерного ряда (с d различными рядами). При работе с текстом вектор \bar{x}_t будет содержать закодированное с помощью *прямого кодирования* слово, соответствующее t -му отсчету. При прямом кодировании мы имеем вектор, длина которого равна размеру словаря и компонента которого, соответствующая данному слову, равна 1. Все остальные компоненты равны нулю.

В случае последовательностей важно то, что последовательные слова зависят друг от друга. Поэтому целесообразно получать конкретный вход x_t лишь *после* того, как более равные входы уже были получены и преобразованы в скрытое состояние. Традиционный тип сетей прямого распространения, в котором все входы передаются первому слою, не обеспечивает достижения этой цели. Поэтому рекуррентная нейронная сеть позволяет входу \bar{x}_t непосредственно взаимодействовать со скрытым состоянием, созданным из входов, соответствующих предыдущим временным отсчетам. Базовая архитектура рекуррентной нейронной сети показана на рис. 1.17, *a*. Ключевой аспект заключается в том, что в каждый момент времени существует вход \bar{x}_t и скрытое состояние \bar{h}_t , которое изменяется, как только поступает новая точка данных. Для каждого отсчета также имеется выходное значение \bar{y}_t . Например, в случае временного ряда выходом \bar{y}_t может быть прогнозируемое значение \bar{x}_{t+1} . Если речь идет о работе с текстом, когда предсказывается следующее слово, то такой подход называют *моделированием языка* (language modeling). В некоторых приложениях мы выводим значения \bar{y}_t не для каждого отсчета, а лишь в конце последовательности. Например, если мы пытаемся классифицировать тональность предложения как “положительная” или “отрицательная”, то выходное значение будет выведено только для последнего отсчета.



а) Рекуррентная нейронная сеть

б) Представление а, развернутое во времени

Рис. 1.17. Рекуррентная нейронная сеть и ее развернутое во времени представление

Скрытое состояние в момент времени t определяется функцией входного вектора в момент времени t и скрытого вектора в момент времени $(t - 1)$:

$$\bar{h}_t = f(\bar{h}_{t-1}, \bar{x}_t). \quad (1.36)$$

Для обучения выходным вероятностям на основе скрытых состояний используется отдельная функция $\bar{y}_t = g(\bar{h}_t)$. Обратите внимание на то, что функции $f(\cdot)$ и $g(\cdot)$ остаются одними и теми же для каждого отсчета. Здесь скрыто неявное предположение о том, что временные ряды демонстрируют некоторый уровень *стационарности*, т.е. базовые свойства не меняются с течением времени. Несмотря на то что в реальных задачах это условие не выполняется со всей строгостью, такое предположение достаточно разумно для того, чтобы использовать его в целях регуляризации.

В данном случае ключевую роль играет изображенный на рис. 1.17, а, цикл, наличие которого приводит к тому, что скрытое состояние нейронной сети изменяется каждый раз при появлении на входе нового значения \bar{x}_t . На практике мы имеем дело с последовательностями конечной длины, и поэтому имеет смысл развернуть этот цикл в сеть с временными слоями, которая выглядит как сеть прямого распространения (рис. 1.17, б). Заметьте, что в данном случае у нас имеются отдельные узлы для каждого скрытого состояния, соответствующего отдельной временной метке, и цикл был развернут в сеть прямого распространения. Это представление математически эквивалентно тому, которое приведено на рис. 1.17, а, но оно гораздо понятнее в силу его сходства с традиционной сетью. Обратите внимание на то, что, в отличие от сетей прямого распространения, в этой развернутой сети входы встречаются также в промежуточных слоях. Матрицы весов соединений *совместно используются многими соединениями* в развернутой во времени сети, тем самым гарантируя, что в каждый момент вре-

мени применяется одна и та же функция. Совместное использование весов играет ключевую роль в обучении сети специфическим для конкретных данных внутренним закономерностям. Алгоритм обратного распространения ошибки учитывает это и временную длину при обновлении весов в процессе обучения. Этот специальный тип обратного распространения ошибки называют *обратным распространением во времени* (backpropagation through time — BPTT). В силу рекурсивной природы уравнения 1.36 рекуррентная сеть обладает *способностью вычислять функцию входов переменной длины*. Другими словами, рекурсию уравнения 1.36 можно развернуть, чтобы определить функцию для \bar{h}_t в терминах t входов. Например, начиная с члена \bar{h}_0 , значение которого обычно фиксируется равным некоторому постоянному вектору, мы имеем $\bar{h}_1 = f(\bar{h}_0, \bar{x}_1)$ и $\bar{h}_2 = f(f(\bar{h}_0, \bar{x}_1), \bar{x}_2)$. Обратите внимание на то, что \bar{h}_1 — это функция только \bar{x}_1 , тогда как \bar{h}_2 — функция как \bar{x}_1 , так и \bar{x}_2 . Поскольку выход \bar{y}_t — функция \bar{h}_t , то эти свойства наследуются также и \bar{y}_t . В общем случае мы можем записать следующее соотношение:

$$\bar{y}_t = F_t(\bar{x}_1, \bar{x}_2, \dots, \bar{x}_t). \quad (1.37)$$

Заметьте, что функция $F_t(\cdot)$ меняется с изменением значения t . Такой подход особенно полезен в случае входов переменной длины наподобие текстовых предложений. Более подробно о рекуррентных нейронных сетях речь пойдет в главе 7, и там же будет обсуждаться применение рекуррентных нейронных сетей в различных областях.

Интересным теоретическим свойством рекуррентных нейронных сетей является их *полнота по Тьюрингу* [444]. Это означает, что при наличии достаточно большого количества данных и вычислительных ресурсов рекуррентная нейронная сеть может имитировать любой алгоритм. Однако на практике это теоретическое свойство не приносит никакой пользы ввиду значительных практических проблем с обобщаемостью рекуррентных сетей на длинные последовательности. Требуемые для этого объем данных и размер скрытых состояний возрастают с увеличением длины последовательности настолько быстро, что их практическая реализация становится невозможной. Кроме того, возникают практические трудности с выбором оптимальных параметров, обусловленные проблемами затухающих и взрывных градиентов. Вследствие этого были предложены специальные варианты рекуррентных нейронных сетей, например такие, в которых используется долгая краткосрочная память. Эти более сложные архитектуры также обсуждаются в главе 7. Кроме того, с помощью усовершенствованных вариантов рекуррентной архитектуры, таких как нейронные машины Тьюринга, в некоторых приложениях удалось получить лучшие результаты, чем с помощью рекуррентных нейронных сетей.

1.6.5. Сверточные нейронные сети

Сверточные нейронные сети, идея которых была почерпнута из биологии, применяются в компьютерном зрении для классификации изображений и обнаружения объектов. В основу этой идеи легли результаты Хьюбела и Визеля [212], исследовавших работу зрительной коры головного мозга кошек, в соответствии с которыми определенные участки зрительного поля возбуждают определенные нейроны. Этот широкий принцип был использован при проектировании разреженной архитектуры сверточных нейронных сетей. Первой базовой архитектурой, основанной на этой биологической модели, был *неокогнитрон*, впоследствии обобщенный до архитектуры *LeNet-5* [279]. В архитектуре сверточной нейронной сети каждый слой является 3-мерным и имеет пространственную протяженность и глубину, которая соответствует количеству признаков. Понятие “глубины” слоя сверточной нейронной сети отличается⁴ от понятия глубины в смысле количества слоев. Во входном слое упомянутые признаки соответствуют цветовым каналам типа RGB (т.е. красному, зеленому и синему цветам), тогда как в скрытых каналах эти признаки представляют карты скрытых признаков, кодирующие различные типы форм в изображении. Если входы задаются в градациях серого (как в *LeNet-5*), то входной слой будет иметь глубину 1, но последующие слои по-прежнему будут трехмерными. Такая архитектура содержит слои двух типов, которые называют *сверточным слоем* (convolution layer) и *слоем субдискретизации* (subsampling layer) соответственно.

Для сверточных слоев определяется операция *свертки* (convolution), в которой для трансляции активаций из одного слоя в следующий используется фильтр. Операция свертки использует трехмерный фильтр весов той же глубины, что и текущий слой, но меньшей пространственной протяженности. Значение скрытого состояния в следующем слое (после применения функции активации наподобие ReLU) определяется скалярным (точечным) произведением между всеми весами фильтра и любой выбранной в слое пространственной областью (того же размера, что и фильтр). Данная операция между фильтром и пространственными областями в слое выполняется в каждой возможной позиции для определения следующего слоя (в котором сохраняются пространственные соотношения между активациями из предыдущего слоя).

Сверточная нейронная сеть характеризуется высокой разреженностью соединений, поскольку любая активация отдельного слоя является функцией только небольшой пространственной области предыдущего слоя. Все слои, за исключением последнего набора из двух-трех слоев, сохраняют свою

⁴ Это перегрузка терминологии сверточных нейронных сетей. Смысл слова “глубина” выводится из контекста, в котором оно используется.

пространственную структуру, что обеспечивает возможность пространственной визуализации того, какие именно части изображения влияют на определенные части активаций в слое. Признаки в слоях более низкого уровня захватывают линии и другие примитивные формы, тогда как признаки в слоях более высокого уровня захватывают более сложные формы наподобие замкнутых петель (которые обычно встречаются в начертаниях многих цифр). Поэтому более поздние слои могут создавать цифры путем композиции форм, сохраненных в этих интуитивных признаках. Это классический пример того, как понимание внутренних семантических зависимостей, существующих среди данных определенного типа, используется для проектирования более “умных” архитектур. В дополнение к этому субдискретизирующий слой просто усредняет значения в локальных областях размером 2×2 , тем самым снижая размерность слоев по каждому пространственному измерению в 2 раза. Архитектура сети LeNet-5 приведена на рис. 1.18. Ранее сеть LeNet-5 использовалась несколькими банками для распознавания рукописных цифр на чеках.

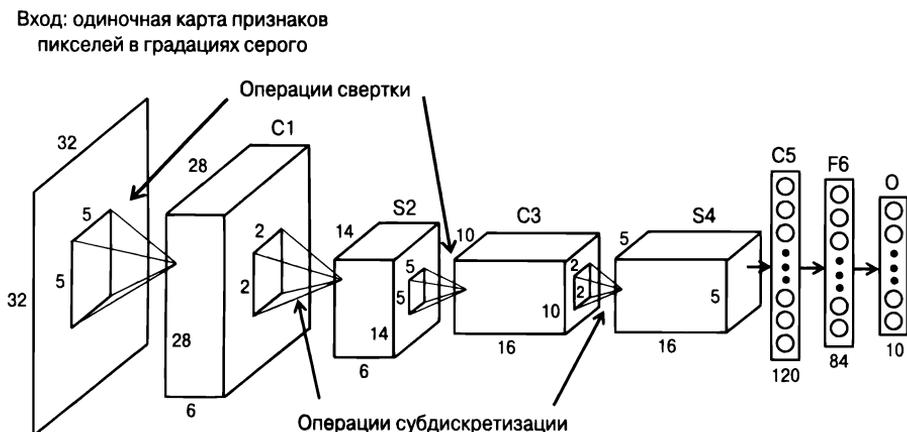


Рис. 1.18. LeNet-5: одна из первых сверточных нейронных сетей

Сверточные нейронные сети исторически оказались наиболее успешными из всех типов нейронных сетей. Они широко применяются для распознавания образов, обнаружения/локализации объектов и даже для обработки текста. При решении задач классификации изображений эти сети недавно продемонстрировали производительность, превосходящую возможности человека [184]. Пример сверточных нейронных сетей хорошо подтверждает тот факт, что варианты архитектурных решений нейронной сети должны выбираться с учетом семантических особенностей данных конкретной предметной области. В частном случае сверточной нейронной сети понимание этих особенностей было достигнуто на основе наблюдений за тем, как работает зрительная кора головного мозга кошек, а также за счет интенсивного использования пространственных

соотношений между пикселями. Кроме того, этот факт служит дополнительным свидетельством того, что дальнейшие достижения нейронауки также могут оказаться полезными для разработки методов искусственного интеллекта.

Сверточные нейронные сети, предварительно обученные на таких публично доступных ресурсах, как ImageNet, часто предлагаются в готовом для применения в других приложениях виде. Это достигается за счет использования в сверточной сети большинства предварительно обученных весов без каких-либо изменений, за исключением последнего классифицирующего слоя. Веса в этом слое обучаются на данных конкретной решаемой задачи. Тренировка последнего слоя необходима по той причине, что метки классов в данной задаче могут отличаться от тех, которые используются в ImageNet. Тем не менее веса в ранних слоях все еще остаются полезными, поскольку они обучены различным типам форм в изображениях, которые могут пригодиться в задачах классификации практически любого типа. Кроме того, активации признаков в предпоследнем слое могут быть использованы в приложениях даже в целях обучения без учителя. Например, можно создать многомерное представление произвольного набора изображений, пропуская каждое изображение через сверточную нейронную сеть и извлекая активации предпоследнего слоя. Впоследствии к этому представлению можно применить любой тип индексирования для изображений, аналогичных предъявленному целевому изображению. Такой подход часто демонстрирует на удивление хорошие результаты при извлечении изображений в силу семантической природы признаков, которым обучается сеть. Следует отметить, что использование предварительно обученных сверточных сетей стало настолько популярным, что их тренировка с нуля выполняется лишь в редких случаях. Сверточные нейронные сети подробно обсуждаются в главе 8.

1.6.6. Иерархическое конструирование признаков и предварительное обучение моделей

Многие углубленные архитектуры, охватывающие архитектуры прямого распространения, включают несколько слоев, в которых последовательные преобразования входов от предыдущего слоя приводят к представлениям данных все более увеличивающейся сложности. Значения в каждом скрытом слое, соответствующие отдельному входу, включают преобразованное представление входной точки данных, содержащее все более полную информацию о целевом значении, которому мы пытаемся обучить сеть, по мере приближения к выходному узлу. Как было показано в разделе 1.5.1, преобразованные подходящим образом представления признаков лучше соответствуют простым типам предсказаний в выходном слое. Это усложнение является результатом применения нелинейных активаций в промежуточных слоях. Ранее в качестве функций активации чаще всего выбирались сигмоида и гиперболический тангенс, однако в последнее

время все большую популярность завоевывает активация ReLU, использование которой облегчает устранение проблем затухающего и взрывного градиента (раздел 3.4.2). В задачах классификации последний слой может рассматриваться в качестве простого прогнозного слоя, который содержит всего один выходной нейрон в случае регрессии и представляет собой сигмоиду/знаковую функцию в случае бинарной классификации. Более сложные выходы могут требовать использования большего количества узлов. Одна из точек зрения на такое разделение труда между скрытыми слоями и последним прогнозирующим слоем заключается в том, что ранние слои создают представление признаков, которое более всего соответствует данной задаче. Затем последний слой использует все преимущества этого представления изученных признаков. Подобное разделение труда показано на рис. 1.19. Важно то, что признаки, которым сеть обучилась в скрытых слоях, часто (но не всегда) обобщаются на другие наборы данных и задачи, относящиеся к той же предметной области (например, обработка текста, изображений и т.п.). Этой особенностью можно воспользоваться самыми разными способами, просто заменив выходной узел (узлы) предварительно обученной сети выходным слоем, учитывающим специфику конкретного приложения (например, используя слой линейной регрессии вместо слоя сигмоидной классификации) в отношении данных и решаемой задачи. В дальнейшем обучение новым данным может потребоваться лишь для весов этого нового замещающего слоя при фиксированных весах остальных слоев.

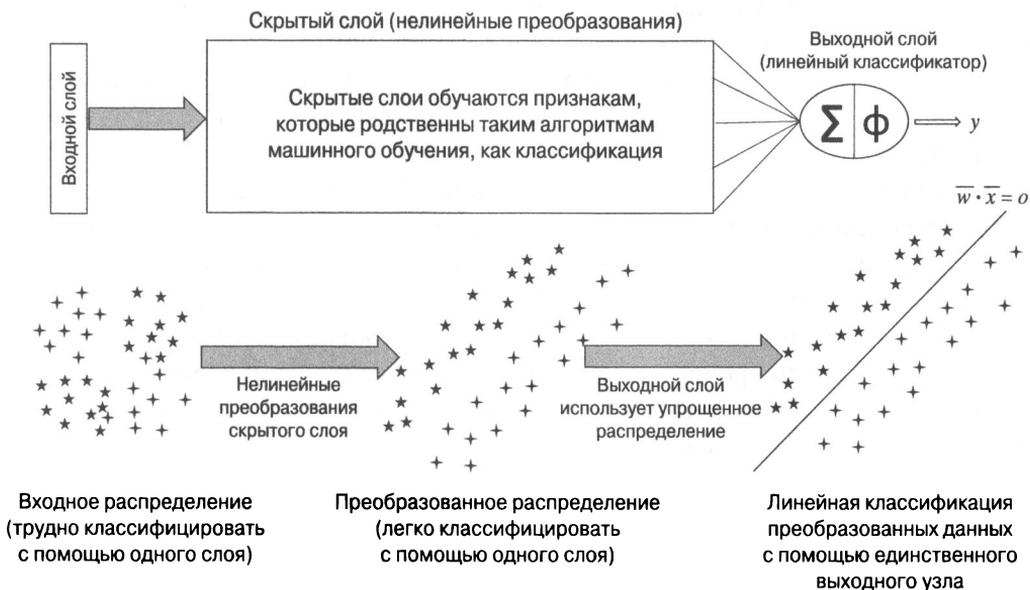


Рис. 1.19. Роль скрытых слоев в конструировании признаков

Выходом каждого скрытого слоя является преобразованное представление признаков данных, размерность которого определяется количеством элементов в данном слое. Этот процесс можно рассматривать в качестве своеобразного процесса построения иерархии признаков, когда признаки в ранних слоях представляют примитивные характеристики данных, в то время как признаки в более поздних слоях представляют сложные характеристики с семантической значимостью для меток классов. Данные, представленные в терминах признаков в более поздних слоях, часто ведут себя лучше (например, являются линейно разделимыми) в силу того, что они отражают семантическую природу признаков, изученных путем преобразования. В некоторых задачах, например в случае сверточных нейронных сетей для обработки изображений, такой тип поведения проявляется в наглядной, визуально интерпретируемой форме. В сверточных нейронных сетях признаки в ранних слоях захватывают в наборе изображений детальные, но примитивные формы наподобие линий или границ однородности изображений. С другой стороны, признаки в более поздних слоях захватывают более сложные формы наподобие шестиугольников, сот и т.п., в зависимости от типа изображений, предоставленных в качестве тренировочных данных. Обратите внимание на то, что такие семантически интерпретируемые формы зачастую тесно коррелируют с метками классов в задачах обработки изображений. Например, почти любое изображение будет содержать линии или границы, тогда как для изображений, принадлежащих к некоторым определенным классам, более характерным будет наличие в них шестиугольников или сот. Эта особенность упрощает задачу классификации представлений более поздних слоев с помощью простых моделей наподобие линейных классификаторов. Признаки, полученные в более ранних слоях, используются в качестве строительных кирпичиков для создания более сложных признаков. Этот общий принцип “сведения воедино” простых признаков для создания более сложных признаков лежит в основе успехов, достигнутых с помощью нейронных сетей. Использование этого способа тщательно продуманными способами оказалось полезным и для предварительного обучения моделей. Практику использования предварительно обученных моделей также называют *переносимым обучением* (transfer learning).

Суть одного специфического типа переносимого обучения, обычно применяемого в нейронных сетях, заключается в том, что данные и структура, доступные для некоторого набора данных, используются с целью обучения признакам, относящимся к данной предметной области в целом. Классическим примером могут служить текстовые или графические данные. В случае текстовых данных представления слов создаются с использованием стандартизированных эталонных наборов данных, таких как *Википедия* [594], и моделей наподобие *word2vec*. Такие данные и модели могут быть использованы практически в любом приложении для обработки текста, поскольку природа

текстовых данных не изменяется существенно при переходе от одного приложения к другому. Аналогичный подход часто применяется и в отношении изображений, когда набор данных *ImageNet* (раздел 1.8.2) используется для предварительного обучения сверточных нейронных сетей и предоставляет готовые к использованию признаки. Любой набор изображений можно преобразовать в многомерное представление, загрузив предварительно обученную сверточную нейронную сеть и пропустив через нее изображения. Кроме того, при наличии дополнительных данных, специфических для приложения, можно регулировать уровень переносимого обучения в зависимости от объема имеющихся данных. Это достигается за счет тонкой настройки подмножества слоев в предварительно обученной сети с помощью этих дополнительных данных. Если доступен лишь небольшой объем данных, специфических для приложения, то можно зафиксировать веса ранних слоев на их предварительно обученных значениях и настроить лишь последние несколько слоев нейронной сети. Ранние слои часто содержат примитивные признаки, которые легче обобщаются на произвольные приложения. Например, в сверточной нейронной сети ранние слои обучаются таким примитивным признакам, как границы областей изображения с однородными характеристиками, которые встречаются в самых разных изображениях, от моркови до грузовика. С другой стороны, поздние слои содержат сложные признаки, которые могут зависеть от особенностей коллекции изображений (например, колесо грузовика или срез верхушки моркови). В подобных случаях имеет смысл настраивать только последние слои. Если же объем доступных данных, специфических для приложения, достаточно велик, то можно настраивать большее количество слоев. Поэтому глубокие сети обеспечивают значительную гибкость в отношении способов реализации переносимого обучения с использованием предварительно обученных моделей нейронных сетей.

1.7. Дополнительные темы

Ряд тем глубокого обучения вызывает постоянно возрастающий интерес, и в этих областях недавно были достигнуты заметные успехи. Несмотря на то что некоторые из методов ограничены доступными в настоящее время вычислительными возможностями, они обладают значительным потенциалом.

1.7.1. Обучение с подкреплением

В обычных приложениях искусственного интеллекта нейронные сети должны обучаться тому, чтобы предпринимать те или иные действия в постоянно изменяющихся и динамически развивающихся ситуациях. Примерами могут служить обучающиеся роботы и беспилотные автомобили. Во всех подобных случаях критическое значение имеет предположение о том, что обучаемой

системе ничего заранее не известно о том, какая последовательность действий будет наиболее подходящей, и все обучение, так называемое *обучение с подкреплением* (reinforcement learning), проводится посредством присуждения вознаграждения за правильность совершаемых системой действий. Эти типы обучения соответствуют динамической последовательности действий, которую трудно моделировать, используя традиционные методы машинного обучения. В данном случае ключевым является предположение о том, что такие системы слишком сложны для явного моделирования, но достаточно просты для того, чтобы за каждое правильное действие обучаемой системы ей можно было назначить вознаграждение.

Представьте ситуацию, когда требуется с нуля обучить систему видеоигре, правила которой ей не известны заранее. Видеоигры представляют собой великолепную тестовую площадку для методов обучения с подкреплением. Как и в реальных ситуациях, количество возможных *состояний* (т.е. уникальных позиций в игре) может быть настолько большим, что уже один его подсчет становится нереальным, и в этих условиях выбор оптимального хода критически зависит от знания того, какая информация о конкретном состоянии игры действительно важна для модели. Кроме того, поскольку невозможно вести игру, вообще не зная правил, обучаемая система должна собрать необходимые сведения, совершая какие-то действия, как это делает мышь, исследуя лабиринт, чтобы изучить его структуру. Поэтому собранные данные испытывают значительное смещение в результате пользовательских действий, что создает неблагоприятные условия для обучения. Успешная тренировка методов обучения с подкреплением — вот путь, ведущий к *самообучающимся* системам, Святому Граалю искусственного интеллекта. И хотя поле обучения с подкреплением было исследовано независимо от поля нейронных сетей, их взаимодополняемость свела их вместе. Методы глубокого обучения могут быть полезными для обучения представлениям признаков на многомерных сенсорных входах (например, это могут быть пиксели видеоигры или пиксели поля “зрения” робота). Кроме того, методы обучения с подкреплением часто применяются для поддержки различных типов алгоритмов нейронных сетей, например *механизмов внимания*. Методы обучения с подкреплением обсуждаются в главе 9.

1.7.2. Отделение хранения данных от вычислений

Важным аспектом нейронных сетей является тесная интеграция хранения данных и вычислений. Например, состояния нейронной сети могут рассматриваться как некоторый тип временной памяти, во многом напоминающий своим поведением регистры центрального процессора компьютера. Но что если мы хотим построить нейронную сеть, в которой можно контролировать, откуда читать данные и куда их записывать? Эта цель достигается за счет использования

внимания и внешней памяти. Механизмы внимания могут применяться в различных задачах, таких как обработка изображений, где внимание последовательно фокусируется на небольших участках изображения. Эти методики также используются в машинном переводе. Нейронные сети, имеющие возможность строго контролировать доступ по чтению и записи к внешней памяти, называются *нейронными машинами Тьюринга* (neural Turing machines) [158] или *сетями памяти* (memory networks) [528]. И хотя они являются не более чем усовершенствованными вариантами рекуррентных нейронных сетей, они демонстрируют значительное превосходство над своими предшественниками в отношении круга задач, с которыми способны справиться. Эти методы обсуждаются в главе 10.

1.7.3. Генеративно-сопоставительные сети

Генеративно-сопоставительные сети (generative adversarial network, GAN) — это модель порождения данных, фактически построенная на основе двух моделей, состязющихся между собой в своеобразной игре. Этими двумя “игроками” являются генератор и дискриминатор. Генератор принимает гауссовский шум в качестве входа и выдает на выходе *сгенерированный* образец, похожий на пример из базовой обучающей выборки данных. В роли дискриминатора обычно выступает какой-нибудь вероятностный классификатор, например на основе логистической регрессии, который должен научиться отличать реальные образцы из базового набора данных от сгенерированных. Генератор пытается генерировать как можно более реальные образцы. Его задача состоит в том, чтобы обмануть дискриминатор, тогда как задача дискриминатора — идентифицировать поддельные образцы, несмотря на все попытки генератора обмануть его. Суть процесса можно понять, рассматривая его как сопоставительную игру между генератором и дискриминатором, тогда как ее формальной оптимизационной моделью является обучение задаче минимакса. Получение конечной обученной модели обеспечивается достижением *равновесия Нэша* в этой минимаксной игре. В типичных случаях точкой равновесия является та, в которой дискриминатору не удается отличать поддельные образцы от реальных.

Подобные методы позволяют создавать реалистично выглядящие фантазийные образцы на основе базового набора данных и обычно используются в задачах обработки изображений. Например, если использовать для тренировки набор данных, содержащий изображения спален, то сеть породит реалистичные изображения спален, которые на самом деле не являются частью базового набора. Поэтому такой подход вполне подходит для использования в художественных или творческих целях. Также возможна тренировка этих методов на специфических типах контекста, таких как метки, текстовые подписи или изображения с отсутствующими деталями. Во всех случаях используются пары тренировочных объектов. Типичным примером такой пары может служить подпись к рисунку

(контекст) и изображение (базовый объект). Точно так же подобными парами могут быть эскизы объектов и их фактические фотографии. Поэтому, начав с набора данных, который содержит изображения различных типов животных, снабженные подписями, можно создать фантазийное изображение, не являющееся частью базового набора данных, по контекстной подписи “синяя птица с острыми когтями”. Или же, начав со сделанного художником наброска дамской сумочки, можно получить ее реалистичное цветное изображение. Генеративно-состязательные сети обсуждаются в главе 10.

1.8. Два показательных эталонных теста

Среди обсуждаемых в литературе по нейронным сетям результатов эталонных тестов доминируют данные, относящиеся к области компьютерного зрения. Нейронные сети могут тестироваться на наборах данных традиционного машинного обучения, таких как репозиторий UCI [601], однако наблюдается тенденция использовать для этой цели преимущественно хорошо визуализируемые наборы данных. Несмотря на наличие широкого разнообразия наборов данных в виде текста и изображений, два из них занимают особое положение, поскольку ссылки на них повсеместно встречаются в статьях по глубокому обучению. И хотя оба этих набора связаны с компьютерным зрением, первый из них достаточно прост, чтобы его можно было использовать для тестирования обычных приложений, не имеющих отношения к зрению.

1.8.1. База данных рукописных цифр MNIST

База данных *MNIST* (от *Modified National Institute of Standards and Technology*) — это обширная база данных рукописных цифр [281]. Как следует из самого названия, набор данных был создан путем изменения оригинальной базы рукописных цифр, созданной Национальным институтом стандартов и технологий США (NIST). Набор содержит 60 000 тренировочных и 10 000 тестовых изображений. Каждое изображение получено сканированием рукописных цифр от 0 до 9, различия между которыми являются результатом того, что цифры были написаны различными людьми. Для этого были привлечены сотрудники Бюро переписи населения США и студенты американских вузов. Исходные черно-белые изображения, полученные от NIST, были нормализованы таким образом, чтобы они умещались в пределах окна размером 20×20 пикселей с соблюдением первоначального форматного соотношения, а затем центрированы в изображение размером 28×28 пикселей путем вычисления центра масс пикселей и его переноса в центр поля размером 28×28 . Каждый из этих 28×28 пикселей имеет значение от 0 до 255, в зависимости от того, какую позицию на серой шкале он занимает. С каждым значением связана метка, соответствующая одной из десяти цифр.

Примеры цифр, содержащихся в базе данных MNIST, приведены на рис. 1.20. Это довольно небольшой набор данных, содержащий только простые объекты в виде цифр. Поэтому кое-кто мог бы назвать базу данных MNIST “игрушечной”. Однако небольшой размер и простота набора одновременно являются его преимуществом, поскольку его можно легко использовать в качестве лаборатории для быстрого тестирования алгоритмов машинного обучения. Кроме того, дополнительная простота этого набора, обусловленная (приблизительным) центрированием цифр, облегчает тестирование алгоритмов, не связанных с компьютерным зрением. Алгоритмы компьютерного зрения требуют использования специальных предположений, таких как предположение о трансляционной инвариантности. Простота данного набора делает подобные предположения излишними. Как метко заметил Джефф Хинтон [600], исследователи в области нейронных сетей используют базу данных MNIST во многом подобно тому, как биологи используют дрозофилу для быстрого получения предварительных результатов (прежде чем выполнять серьезные тесты на более сложных организмах).



Рис. 1.20. Примеры рукописных цифр, хранящихся в базе данных MNIST

Несмотря на то что матричное представление каждого изображения подходит для сверточной нейронной сети, его также можно преобразовать в многомерное представление, имеющее $28 \times 28 = 784$ измерений. При таком преобразовании часть пространственной информации теряется, но эта потеря не сказывается значительно на результатах (по крайней мере, в случае набора данных MNIST ввиду его относительной простоты). И действительно, применение простого метода опорных векторов к 784-мерному представлению демонстрирует впечатляющий результат с ошибкой всего лишь 0,56%. Применение простой двухслойной нейронной сети для обработки многомерного представления

(без использования пространственной структуры изображения) обычно приводит к худшим результатам по сравнению с методом опорных векторов в широком диапазоне выбора значений параметров! Глубокая нейронная сеть без какой-либо сверточной архитектуры обеспечивает снижение ошибки до уровня 0,35% [72]. Более глубокие нейронные сети и сверточные нейронные сети (учитывающие пространственную структуру изображений) могут снизить ошибку до уровня 0,21% за счет использования ансамбля из пяти сверточных сетей [402]. Таким образом, даже в случае столь простого набора данных видно, что относительная производительность нейронных сетей по сравнению с традиционными методами машинного обучения чувствительна к специфике используемой архитектуры.

Наконец, следует отметить, что 784-мерное представление без учета пространственной структуры изображений набора MNIST используется для тестирования всех типов алгоритмов нейронных сетей, помимо тех, которые применяются в области компьютерного зрения. И хотя 784-мерное (плоское) представление не подходит для задач, связанных со зрением, его можно использовать для тестирования общей эффективности универсальных (т.е. не ориентированных на задачи, связанные со зрением) алгоритмов нейронных сетей. Например, базу данных MNIST часто используют для тестирования типичных, а не только сверточных автокодировщиков. Даже если для реконструкции изображения с помощью автокодировщика используется его непространственное представление, результаты по-прежнему можно визуализировать, используя информацию об исходных позициях реконструируемых пикселей, тем самым создавая впечатление, будто алгоритм работает непосредственно с данными. Такое визуальное исследование часто помогает ученым пролить свет на особенности, которые не удастся заметить при работе с произвольными наборами данных наподобие тех, которые получают из репозитория UCI Machine Learning Repository [601]. В этом смысле набор данных MNIST имеет более широкую применимость, чем многие другие типы наборов данных.

1.8.2. База данных ImageNet

ImageNet [581] — это огромная база данных, которая содержит свыше 14 миллионов изображений, образующих примерно 1000 различных категорий. Ее покрытие классов настолько широко, что она охватывает большинство типов изображений того, что может встретиться в повседневной жизни. Эта база данных организована в соответствии с иерархией существительных, используемой в электронном тезаурусе *WordNet* [329] — базе данных, содержащей соотношения между словами английского языка. Ее основной словарной единицей является не слово, а так называемый синонимический ряд — *синсет* (от англ. *synonym set*). Иерархия *WordNet* успешно использовалась для решения

задач машинного обучения в области обработки естественного языка, и поэтому построение набора изображений на основе используемых в ней соотношений между словами представляется вполне естественным.

Проект ImageNet известен проводимыми под его эгидой ежегодными соревнованиями *ImageNet Large Scale Visual Recognition Challenge* (ILSVRC) [582]. Эти соревнования высоко ценятся сообществом специалистов по обработке изображений, и в них принимают участие большинство групп, занимающихся исследованиями в области компьютерного зрения. На этих соревнованиях были впервые представлены многие из современных передовых архитектур, ориентированных на распознавание образов, в том числе методы, продемонстрировавшие превосходство над человеком при решении некоторых узких задач, таких как классификация объектов [184]. Ввиду широкой доступности известных результатов, полученных с использованием этого набора данных, сравнение с ними является популярной альтернативой эталонному тестированию. Некоторые из современных алгоритмов, которые были представлены на соревнованиях ImageNet, будут рассмотрены в главе 8 при обсуждении сверточных нейронных сетей.

Другим немаловажным фактором, обуславливающим ценность набора данных ImageNet, является то, что ввиду своего большого размера и широкого разнообразия содержащихся в нем изображений он достаточно полно представляет все наиболее важные визуальные концепции в области обработки изображений. Как следствие, его часто используют для тренировки сверточных нейронных сетей, после чего предобученную сеть можно использовать для извлечения признаков из любого изображения. Это представление изображения определяется скрытыми активациями в предпоследнем слое нейронной сети. Такой подход позволяет создавать новые многомерные представления наборов данных изображений, пригодных для использования традиционными методами машинного обучения. Его можно рассматривать как своего рода переносимое обучение, где визуальные концепции, скрытые в наборе данных ImageNet, переносятся на неизвестные объекты, с которыми работают другие приложения.

1.9. Резюме

Несмотря на то что искусственную нейронную сеть можно рассматривать как имитацию процесса обучения в живых организмах, для более непосредственного понимания принципов работы этих сетей лучше представлять их в виде вычислительных графов. Такие вычислительные графы осуществляют рекурсивную композицию простых функций для обучения более сложным функциям. Поскольку вычислительные графы параметризованы, обычно задача сводится к обучению параметров графа посредством оптимизации функции потерь. Простейшими типами нейронных сетей часто являются базовые модели

машинного обучения, такие как регрессия по методу наименьших квадратов. Реальная мощь нейронных сетей проявляется в полной мере при использовании более сложных комбинаций базовых функций. Обучение параметров таких сетей ведется с использованием алгоритма динамического программирования, известного как метод обратного распространения ошибки. С обучением моделей нейронных сетей связаны определенные трудности, такие как переобучение и нестабильность процесса тренировки. Достижения последних лет в области разработки алгоритмов позволили несколько снизить остроту этих проблем. Проектирование методов глубокого обучения в таких специфических областях, как обработка текста и изображений, требует использования тщательно продуманных архитектур. Примерами таких архитектур могут служить рекуррентные и сверточные нейронные сети. В случае задач с динамическими условиями, требующими обучения принятию последовательности решений, могут быть полезными такие методы, как обучение с подкреплением.

1.10. Библиографическая справка

Понимание принципов работы нейронных сетей требует хорошего знания алгоритмов машинного обучения, особенно линейных моделей на основе градиентного спуска. Элементарное изложение методов машинного обучения содержится в [2, 3, 40, 177]. Многочисленные обзоры и сведения общего характера относительно использования нейронных сетей в различных контекстах приведены в [27, 28, 198, 277, 345, 431]. Классическими книгами по нейронным сетям для распознавания образов являются [41, 182], тогда как изложение более современных взглядов на перспективы глубокого обучения можно найти в [147]. В вышедшей недавно книге по интеллектуальному анализу текста [6] также обсуждаются последние достижения в области глубокого обучения. Обзоры взаимосвязи глубокого обучения и вычислительной нейробиологии содержатся в [176, 239].

Алгоритм перцептрона был предложен Розенблаттом [405]. Для устранения проблем нестабильности были предложены карманный алгоритм [128], алгоритм Маховега [523] и другие методы на основе понятия зазоров [123]. К числу других ранних алгоритмов аналогичного характера относятся алгоритмы Уидроу — Хоффа [531] и Уинноу [245]. Алгоритм Уинноу использует мультипликативные обновления вместо аддитивных и особенно полезен в тех случаях, когда многие признаки являются несущественными. Первоначальная идея обратного распространения ошибки основывалась на идее дифференцирования композиции функций, разработанной в теории управления [54, 237]. Использование динамического программирования для градиентной оптимизации переменных, связанных между собой посредством направленного ациклического графа, стало стандартной практикой с 60-х годов прошлого столетия. Однако

возможностей использования этих методов для тренировки нейронных сетей в то время еще не существовало. В 1969 году Минский и Пейперт опубликовали книгу по перцептронну [330], в которой высказали крайне негативную точку зрения в отношении возможности надежной тренировки многослойных нейронных сетей. В книге было продемонстрировано, что выразительные возможности одиночного перцептрона ограничены, а о том, как тренировать несколько слоев перцептрона, никто ничего не знал. Минский был влиятельной фигурой в мире искусственного интеллекта, и отрицательный тон его книги внес свой вклад в наступление длительного застоя в этой области. Адаптация методов динамического программирования к методу обратного распространения ошибки в нейронных сетях была впервые предложена Полом Вербосом в его докторской диссертации в 1974 году [524]. Однако работе Вербоса не удалось преодолеть сильного предубеждения против нейронных сетей, которое к тому времени уже успело укорениться. Алгоритм обратного распространения ошибки был вновь предложен Румельхартом и др. в 1986 году [408, 409]. Работа Румельхарта примечательна изяществом своего представления, и в ней были даны ответы на некоторые из вопросов, поднятых Минским и Пейпертом. Это одна из причин того, почему статья Румельхарта рассматривается как значимое событие для метода обратного распространения ошибки, хотя она и не была первой, в которой этот метод был предложен. Обсуждение истории развития метода обратного распространения ошибки можно найти в книге Пола Вербоса [525].

К этому времени интерес к нейронным сетям возродился лишь частично, поскольку все еще оставались проблемы, связанные с их тренировкой. Тем не менее целые группы исследователей продолжали работы в этом направлении и еще до 2000-го года успели разработать большинство известных нейронных архитектур, включая сверточные нейронные сети, рекуррентные нейронные сети и сети LSTM. Эти методы все еще демонстрировали довольно умеренную точность ввиду ограниченности доступных данных и вычислительных возможностей. Кроме того, механизм обратного распространения ошибки оказался менее эффективным в отношении тренировки глубоких сетей по причине затухающих и взрывных градиентов. Однако к тому времени рядом авторитетных исследователей уже была высказана гипотеза о том, что с увеличением объемов доступных данных и вычислительных мощностей и связанного с этим сокращения длительности экспериментов с алгоритмами производительность существующих алгоритмов должна улучшиться. Появление фреймворков больших данных и мощных GPU в конце 2000-х годов положительно сказалось на исследованиях в области нейронных сетей. Уменьшение длительности экспериментов, обусловленное возросшими вычислительными мощностями, создало условия для применения таких уловок, как предварительное обучение моделей [198]. Очевидное для широкой публики воскрешение нейронных сетей произошло после

2011 года и ознаменовалось рядом ярких побед [255] на соревнованиях глубоких сетей по классификации изображений. Неизменные победы алгоритмов глубокого обучения на этих соревнованиях заложили фундамент для взрыва их популярности, свидетелями чего мы являемся в наши дни. Примечательно то, что отличия выигранных архитектур от тех, которые были разработаны двумя десятилетиями ранее, можно охарактеризовать лишь как умеренные (но имевшие существенное значение).

Пол Вербос, пионер рекуррентных нейронных сетей, предложил оригинальную версию алгоритма обратного распространения ошибки во времени [526]. Основные идеи сверточных нейронных сетей были предложены в контексте *неокогнитрона* в [127]. Затем эта идея была обобщена до сети LeNet-5 — одной из первых сверточных нейронных сетей. Способность нейронных сетей выступать в качестве универсального аппроксиматора функций обсуждается в [208]. Благотворное влияние глубины на уменьшение количества параметров сети обсуждается в [340].

Теоретическая универсальность нейронных сетей была признана еще на ранних этапах их развития. Например, в одной из ранних работ было показано, что нейронная сеть с единственным скрытым слоем может быть использована для аппроксимации любой функции [208]. Дополнительный результат заключается в том, что некоторые нейронные архитектуры наподобие рекуррентных сетей обладают полнотой по Тьюрингу [444]. Последнее означает, что нейронные сети потенциально могут имитировать любой алгоритм. Разумеется, существуют многочисленные практические проблемы, связанные с тренировкой нейронной сети, объясняющие, почему столь впечатляющие теоретические результаты не переходят в реальную производительность. Самой главной из них является жадная к данным природа мелких архитектур, что проявляется в меньшей степени с увеличением глубины. Увеличение глубины можно рассматривать как своего рода регуляризацию, в которой нейронную сеть вынуждают идентифицировать повторяющиеся закономерности среди точек данных и обучаться им. Однако увеличение глубины нейронной сети затрудняет ее тренировку с точки зрения оптимизации. Обсуждение некоторых из этих проблем можно найти в [41, 140, 147]. Экспериментальная оценка, демонстрирующая преимущества более глубоких архитектур, предоставлена в [267].

1.10.1. Видеолекции

Существует большое количество бесплатных видеокурсов по теме глубокого обучения, доступных на таких сайтах, как YouTube и Coursera. Одним из двух наиболее авторитетных ресурсов является видеокурс Джеффа Хинтона на сайте Coursera [600]. Там вы найдете многочисленные предложения по глубокому обучению, а также рекомендации, касающиеся группы курсов, относящихся

к этой области. На момент выхода книги к этим предложениям был добавлен курс Эндрю Энга. Курс по сверточным нейронным сетям, разработанный в Стэнфордском университете, бесплатно доступен на сайте YouTube [236]. Стэнфордский курс, который ведут Карпаты, Джонсон и Фай-Фай [236], также посвящен нейронным сетям, однако его охват более широкой тематики нейронных сетей не может похвастаться достаточной полнотой. В начальных частях курса рассматриваются простейшие нейронные сети и методы тренировки.

Многие темы машинного обучения [89] и глубокого обучения [90] охватывают лекции Нандо Де Фрейтаса, доступные на сайте YouTube. Еще один интересный курс по нейронным сетям предложил Хьюго Ларошель из Шербрукского университета [262]. Также доступен курс, разработанный Али Ходзи из университета Ватерлоо [137]. Видеолекции Кристофера Маннинга по методам обработки естественного языка опубликованы на YouTube [312]. Кроме того, доступен курс Дейвида Силвера по обучению с подкреплением [619].

1.10.2. Программные ресурсы

Глубокое обучение поддерживают многочисленные программные фреймворки, такие как *Caffe* [571], *Torch* [572], *Theano* [573] и *TensorFlow* [574]. Также доступны расширения *Caffe* для Python и MATLAB. Библиотека *Caffe*, написанная на C++, была разработана в Калифорнийском университете в Беркли. Она реализует высокоуровневый интерфейс, с помощью которого можно задать архитектуру сети, и позволяет конструировать нейронные сети, используя небольшой по размеру код и относительно простые сценарии. Основным недостатком *Caffe* — ограниченный объем доступной документации. Библиотека *Theano* [35], препроцессор которой написан на Python, предоставляет в качестве интерфейсов такие высокоуровневые пакеты, как *Keras* [575] и *Lasagne* [576]. В ее основе лежит понятие вычислительных графов, и большинство предоставляемых ею возможностей основано на явном использовании этой абстракции. Библиотека *TensorFlow* [574], предложенная компанией Google, также в значительной степени ориентирована на вычислительные графы. Библиотека *Torch* [572] написана на высокоуровневом языке Lua и имеет относительно дружелюбный пользовательский интерфейс. В последние годы *Torch* несколько укрепила свои позиции по сравнению с остальными фреймворками. Поддержка GPU тесно интегрирована в *Torch*, что упрощает развертывание приложений на основе *Torch* на компьютерах с GPU. Многие из этих фреймворков содержат предварительно обученные модели компьютерного зрения и интеллектуального анализа данных, которые могут быть использованы для извлечения признаков. Многие готовые инструменты глубокого обучения доступны в репозитории *DeepLearning4j* [590]. Платформа *PowerAI* компании IBM предлагает фреймворки машинного и глубокого обучения, работающие поверх серверов

IBM Power Systems [599]. Следует отметить, что на момент выхода книги также предлагался бесплатный вариант этой платформы, доступный для использования в определенных целях.

1.11. Упражнения

1. Пусть имеется функция XOR, в которой две точки $\{(0, 0), (1, 1)\}$ принадлежат к одному классу, а две другие точки $\{(1, 0), (0, 1)\}$ — к другому. Покажите, как разделить два этих класса, используя функцию активации ReLU, аналогично тому, как это было сделано в примере на рис. 1.14.
2. Покажите, что для функции активации в виде сигмоиды и гиперболического тангенса (обозначенной как $\Phi(\cdot)$ в каждом из этих случаев) справедливы приведенные ниже равенства.

А. Сигмоида: $\Phi(-v) = 1 - \Phi(v)$.

Б. Гиперболический тангенс: $\Phi(-v) = -\Phi(v)$.

В. Спрямоленный гиперболический тангенс: $\Phi(-v) = -\Phi(v)$.

3. Покажите, что гиперболический тангенс — это сигмоида, масштабированная вдоль горизонтальной и вертикальной осей и смещенная вдоль вертикальной оси:

$$\tanh(v) = 2\text{sigmoid}(2v) - 1.$$

4. Пусть имеется набор данных, в котором две точки $\{(-1, -1), (1, 1)\}$ принадлежат к одному классу, а две другие точки $\{(1, -1), (-1, 1)\}$ — к другому. Начните со значений параметра перцептрона $(0, 0)$ и выполните несколько обновлений по методу стохастического градиентного спуска с $\alpha = 1$. В процессе получения обновленных значений циклически обходите тренировочные точки в любом порядке.

А. Сходится ли алгоритм в том смысле, что изменения значений целевой функции со временем становятся предельно малыми?

Б. Объясните, почему наблюдается ситуация, описанная в п. А.

5. Определите для набора данных из упражнения 4, в котором двумя признаками являются (x_1, x_2) , новое одномерное представление z , обозначаемое как

$$z = x_1 \cdot x_2.$$

Является ли набор данных линейно разделимым в терминах одномерного представления, соответствующего z ? Объясните важность нелинейных преобразований в задачах классификации.

6. Реализуйте перцептрон на привычном для вас языке программирования.

7. Покажите, что значение производной сигмоиды не превышает 0,25, независимо от значения аргумента. При каком значении аргумента сигмоида имеет максимальное значение?
8. Покажите, что значение производной функции активации \tanh не превышает 1, независимо от значения аргумента. При каком значении аргумента функция активации \tanh имеет максимальное значение?
9. Пусть имеется сеть с двумя входами, x_1 и x_2 . Сеть имеет два слоя, каждый из которых содержит два элемента. Предположим, что веса в каждом слое устанавливаются таким образом, что в верхнем элементе каждого слоя для суммирования его входов применяется сигмоида, а в нижнем — функция активации \tanh . Наконец, в единственном выходном узле для суммирования двух его входов применяется активация ReLU. Запишите выход этой нейронной сети в замкнутой форме в виде функции x_1 и x_2 . Это упражнение должно дать вам представление о сложности функций, вычисляемых нейронными сетями.
10. Вычислите частную производную по x_1 функции в замкнутой форме, полученной в предыдущем упражнении. Практично ли вычислять производные для градиентного спуска в нейронных сетях, используя выражения в замкнутой форме (как в традиционном машинном обучении)?
11. Пусть имеется двухмерный набор данных, в котором все точки с $x_1 > x_2$ принадлежат к положительному классу, а все точки с $x_1 \leq x_2$ — к отрицательному. Истинным разделителем для этих двух классов является линейная гиперплоскость (прямая линия), определяемая уравнением $x_1 - x_2 = 0$. Создайте набор тренировочных данных с 20 точками, сгенерированными случайным образом в положительном квадранте единичного квадрата. Снабдите каждую точку меткой, указывающей на то, превышает или не превышает ее первая координата x_1 вторую координату x_2 .
 - А. Реализуйте алгоритм перцептрона без регуляризации, обучите его на полученных выше 20 точках и протестируйте его точность на 1000 точках, случайно сгенерированных в единичном квадрате. Используйте для генерирования тестовых точек ту же процедуру, что и для тренировочных.
 - Б. Замените критерий перцептрона на кусочно-линейную функцию потерь в своей реализации тренировки и повторите определение точности вычислений на тех же тестовых точках, которые использовали перед этим. Регуляризация не используется.
 - В. В каком случае и почему вам удалось получить лучшую точность?
 - Г. Как вы считаете, в каком случае классификация тех же 1000 тестовых точек не изменится значительно, если использовать другой набор из 20 тренировочных точек?

Глава 2

Машинное обучение с помощью мелких нейронных сетей

Простота — высшая степень сложности.

Леонардо да Винчи

2.1. Введение

В традиционном машинном обучении для тренировки параметризованных моделей часто используют методы оптимизации и градиентного спуска. Примерами таких моделей могут служить линейная регрессия, метод опорных векторов, логистическая регрессия, снижение размерности и факторизация матриц. Нейронные сети также представляют собой параметризованные модели, которые обучаются с помощью непрерывных методов оптимизации. В этой главе будет показано, как *очень простые* архитектуры нейронных сетей, содержащие один или два слоя, способны охватить широкий круг методов, основанных на оптимизации. В действительности нейронные сети можно рассматривать как более мощные версии этих простых моделей, достигающие своей мощности за счет объединения базовых моделей в комплексную нейронную архитектуру (т.е. вычислительный граф). Целесообразно с самого начала обратить ваше внимание на эти параллели, поскольку так вам будет легче понять структуру глубокой сети как композиции базовых элементов, которые часто применяются в машинном обучении. Кроме того, демонстрация этой взаимосвязи прояснит специфические отличия традиционного обучения от нейронных сетей и позволит понять, при каких обстоятельствах лучше использовать нейронные сети. Во многих случаях внесение незначительных изменений в эти простые архитектуры нейронных сетей (соответствующие традиционным методам машинного обучения) позволяет получить полезные вариации моделей машинного обучения, которые еще никем не были изучены. Как бы то ни было, количество

способов, которыми можно комбинировать различные элементы вычислительного графа, намного превышает количество моделей, изучаемых в традиционном машинном обучении, даже если использовать только мелкие модели.

В случае небольших объемов доступных данных использование комплексных или глубоких нейронных архитектур нередко будет перебором. Кроме того, в условиях дефицита данных проще оптимизировать традиционные модели машинного обучения, поскольку они легче поддаются интерпретации. С другой стороны, с ростом доступных объемов данных преимущество переходит к нейронным сетям в силу того, что они предоставляют большую гибкость в отношении моделирования более сложных функций при добавлении нейронов в вычислительный граф. Эта ситуация отражена на рис. 2.1 (который просто повторяет рис. 1.1).

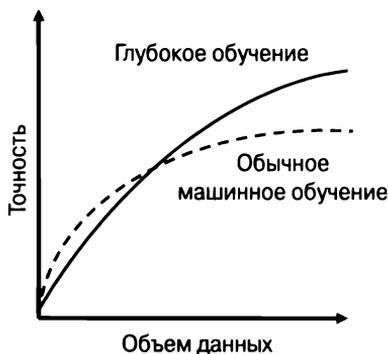


Рис. 2.1. Влияние увеличения объема доступных данных на точность

Одним из способов анализа моделей глубокого обучения является их представление в виде стека более простых моделей наподобие логистической или линейной регрессии. Связывание линейного нейрона с сигмоидной активацией приводит к *логистической регрессии*, которая будет подробно обсуждаться в этой главе. Для построения сложных нейронных сетей также интенсивно использовалось связывание линейного элемента с сигмоидной активацией¹. Поэтому вполне естественно задаться следующим вопросом [312]:

Не является ли глубокое обучение обычным стеком более простых моделей наподобие логистической или линейной регрессии?

Несмотря на то что многие нейронные сети могут рассматриваться подобным образом, такая точка зрения не позволяет полностью охватить всю сложность моделей глубокого обучения и не отражает необходимый для их

¹ В последние годы сигмоидные элементы теряют свою предпочтительность, уступая место ReLU.

понимания стиль мышления. Например, некоторые модели (такие, как рекуррентные или сверточные нейронные сети) формируют стеки особым способом, учитывающим *специфические для конкретной предметной области особенности* входных данных. Кроме того, иногда некоторые параметры совместно используются различными элементами, чтобы тем самым вынудить решение подчиняться специфическим типам свойств. Способность сводить воедино базовые элементы тщательно продуманным способом — важнейший архитектурный навык, которым должны обладать те, кто занимается глубоким обучением. Однако изучение свойств базовых моделей имеет не менее важное значение, поскольку они повсеместно используются в глубоком обучении в качестве элементарных вычислительных модулей. Поэтому базовые модели также рассматриваются в данной главе.

Следует обратить ваше внимание на существование тесной взаимосвязи между некоторыми ранними нейронными сетями (такими, как модели перцептрона и Уидроу — Хоффа) и традиционными моделями машинного обучения (такими, как метод опорных векторов и дискриминант Фишера). В некоторых случаях эта взаимосвязь оставалась незамеченной в течение нескольких лет, поскольку соответствующие модели были независимо предложены различными сообществами. В качестве конкретного примера можно привести функцию потерь в методе опорных векторов с L_2 -регуляризацией, которая была предложена Хинтоном [190] в контексте нейронной архитектуры в 1989 году. В случае использования регуляризации результирующая нейронная сеть будет вести себя так же, как в методе опорных векторов. Для сравнения заметим, что спустя несколько лет вышла статья Кортеса и Вапника, в которой они предложили метод опорных векторов с линейной функцией потерь [82]. Такие параллели не должны удивлять, поскольку наилучшие способы определения мелкой нейронной сети часто тесно связаны с известными алгоритмами машинного обучения. Именно поэтому исследование элементарных нейронных моделей имеет важное значение для выработки интегрального взгляда на нейронные сети и традиционное машинное обучение.

В этой главе в основном обсуждаются два класса моделей машинного обучения.

1. *Модели обучения с учителем, или модели контролируемого обучения (supervised models)*. Обсуждаемые в этой главе модели обучения с учителем в основном соответствуют линейным моделям и их вариантам. Они включают такие методы, как регрессия по методу наименьших квадратов, метод опорных векторов и логистическая регрессия. Также будут изучены мультиклассовые варианты этих моделей.
2. *Модели обучения без учителя, или модели неконтролируемого обучения (unsupervised models)*. Обсуждаемые в этой главе модели обучения без

учителя в основном соответствуют методам снижения размерности и факторизации матриц. Такие традиционные методы, как анализ главных компонент, также могут быть представлены архитектурами простых нейронных сетей. Незначительные вариации этих моделей способны уменьшить количество самых разнообразных свойств, что далее также будет обсуждаться. Проведение аналогий с инфраструктурой нейронной сети позволяет лучше понять, как соотносятся между собой различные методы обучения без учителя, такие как линейное и нелинейное снижение размерности или обучение разреженным признакам, тем самым предоставляя возможность увидеть цельную картину алгоритмов традиционного машинного обучения.

В этой главе предполагается, что читателю уже известно кое-что о классических моделях машинного обучения. Тем не менее, учитывая интересы неподготовленных читателей, ниже будет приведен краткий обзор каждой из этих моделей.

Структура главы

В следующем разделе обсуждаются базовые модели классификации и регрессии, такие как регрессия по методу наименьших квадратов, бинарный дискриминант Фишера, метод опорных векторов и логистическая регрессия. Мультиклассовые варианты этих моделей будут обсуждаться в разделе 2.3, методы отбора признаков для нейронных сетей — в разделе 2.4, использование автокодировщиков для матричной факторизации — в разделе 2.5. В качестве конкретного приложения простых нейронных архитектур в разделе 2.6 обсуждается метод *word2vec*. С простыми методами создания вложений узлов в графах вы познакомитесь в разделе 2.7. Резюме главы приведено в разделе 2.8.

2.2. Архитектуры нейронных сетей для моделей бинарной классификации

В этом разделе мы обсудим некоторые базовые архитектуры моделей машинного обучения, такие как регрессия по методу наименьших квадратов и классификация. Как мы увидим, соответствующие нейронные архитектуры представляют собой незначительные вариации модели перцептрона в машинном обучении. Основное различие между ними заключается в выборе функции активации, используемой в последнем слое, и функции потерь, используемой в этих выходах. К этой теме мы будем постоянно возвращаться в данной главе, в которой будет показано, что небольшие изменения нейронной архитектуры могут приводить к моделям, отличающимся от моделей традиционного машинного обучения. Представление моделей традиционного машинного обучения в

виде нейронных архитектур также помогает осознать истинную близость различных моделей машинного обучения.

На протяжении всего раздела мы будем работать с однослойной сетью, имеющей d входных узлов и единственный выходной узел. Весовые коэффициенты связей, соединяющих d входных узлов с выходным узлом, обозначены как $\bar{W} = (w_1 \dots w_d)$. Кроме того, смещение не будет указываться явным образом, поскольку его можно моделировать в виде весового коэффициента дополнительного фиктивного входа с постоянным значением 1.

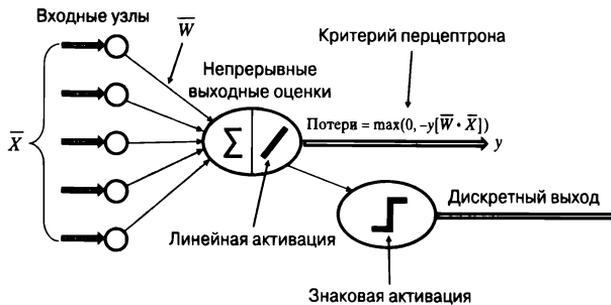


Рис. 2.2. Расширенная архитектура перцептрона с дискретными и непрерывными прогнозными значениями

2.2.1. Повторное рассмотрение перцептрона

Пусть (\bar{X}_i, y_i) — тренировочный пример, в котором наблюдаемое значение y_i прогнозируется на основе значений переменных признаков \bar{X}_i с использованием следующего соотношения:

$$\hat{y}_i = \text{sign}(\bar{W} \cdot \bar{X}_i), \quad (2.1)$$

где \bar{W} — d -мерный вектор коэффициентов, которым обучается перцептрон. Обратите внимание на символ циркумфлекса (“крышка”) в обозначении \hat{y}_i , указывающий на то, что это предсказанное, а не наблюдаемое значение. Вообще говоря, цель тренировки заключается в том, чтобы гарантировать максимальную близость предсказания \hat{y}_i к наблюдаемому значению y_i . Шаги градиентного спуска перцептрона направлены на снижение количества неверных классификаций, и поэтому обновления пропорциональны разнице $(y_i - \hat{y}_i)$ между наблюдаемым и прогнозным значениями на основании уравнения 1.33:

$$\bar{W} \leftarrow \bar{W}(1 - \alpha\lambda) + \alpha(y_i - \hat{y}_i)\bar{X}_i. \quad (2.2)$$

Пропорциональность обновлений градиентного спуска разнице между наблюдаемым и предсказанным значениями естественным образом вытекает из квадратичности функции потерь, такой, например, как $(y_i - \hat{y}_i)^2$. Поэтому одной

из возможностей является использование квадратичных потерь между предсказанным и наблюдаемым значениями в качестве функции потерь. Архитектура этого типа с дискретным выходным значением представлена на рис. 2.3, а. Однако проблема в том, что эта функция потерь дискретна, поскольку она имеет лишь значения 0 или 4. Такая функция не является дифференцируемой из-за скачков в виде ступенек.

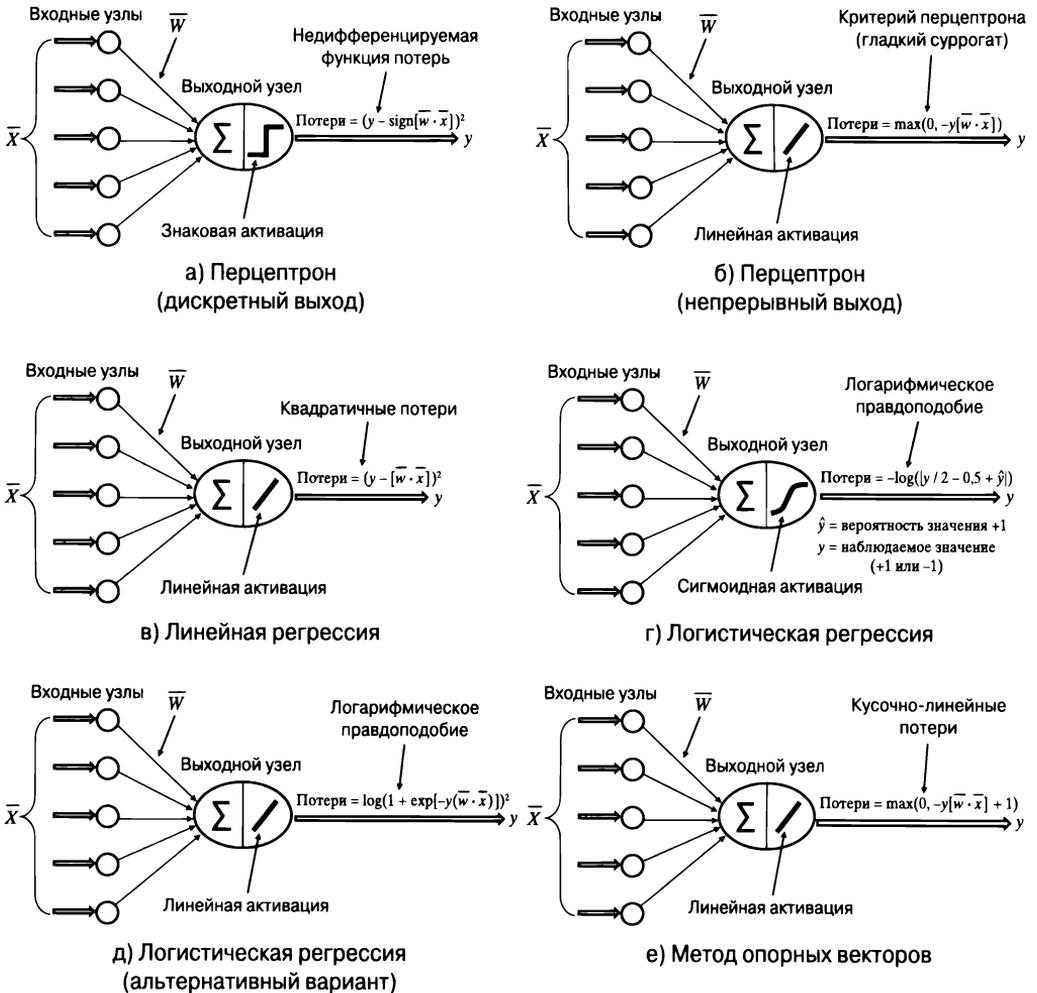


Рис. 2.3. Различные варианты перцептрона

Перцептрон — одна из немногих моделей обучения, в которой градиентные обновления исторически были предложены еще до того, как была предложена функция потерь. Какую же дифференцируемую функцию в действительности оптимизирует перцептрон? Ответ на этот вопрос вы найдете в разделе 1.2.1.1,

заметив, что обновления выполняются лишь для неверно классифицированных тренировочных примеров (для которых $y_i \hat{y}_i < 0$) и могут быть записаны с использованием индикаторной функции $I(\cdot) \in \{0, 1\}$, которая имеет значение 1, если удовлетворится условие, указанное в ее аргументе:

$$\bar{W} \leftarrow \bar{W}(1 - \alpha\lambda) + \alpha y_i \bar{X}_i [I(y_i \hat{y}_i < 0)]. \quad (2.3)$$

При переходе от уравнения 2.2 к уравнению 2.3 был использован тот факт, что $y_i = (y_i - \hat{y}_i) / 2$ для неверно классифицированных точек, а постоянный множитель 2 может быть включен в скорость обучения. Можно показать, что это правило обновления согласуется с приведенной ниже функцией потерь L_i (специфической для i -го тренировочного примера):

$$L_i = \max\{0, -y_i(\bar{W} \cdot \bar{X}_i)\}. \quad (2.4)$$

Эту функцию потерь называют *критерием перцептрона* (рис. 2.3, б). Заметьте, что на рис. 2.3, б, для вычисления непрерывной функции потерь используются *линейные* активации, но при этом дискретные предсказания для данного тестового примера по-прежнему вычисляются с использованием активаций в виде функции *sign*. Во многих задачах с дискретными переменными предсказаний выход часто представляет собой оценку предсказания (например, вероятность класса или значение $\bar{W} \cdot \bar{X}_i$), которая затем преобразуется в дискретное значение. Тем не менее окончательное предсказание не всегда должно преобразовываться в дискретное значение, и можно просто выводить соответствующую оценку для класса (что, во всяком случае, часто используется для вычисления функции потерь). Активация *sign* редко применяется в большинстве реализаций нейронных сетей, поскольку в большинстве случаев предсказания классовых переменных представляют собой непрерывные оценки. В действительности для перцептрона можно создать расширенную архитектуру (см. рис. 2.2), выходом которой являются как дискретные, так и непрерывные значения. Однако, поскольку дискретная часть несущественна для вычисления функции потерь, а большинство выходов так или иначе выводятся как оценки, этот тип расширенного представления используется крайне редко. Поэтому на протяжении оставшейся части книги активация в выходных узлах будет ориентирована на оценочный выход (и на то, как вычисляется функция потерь), а не на предсказания в виде дискретных значений для тестовых примеров.

2.2.2. Регрессия по методу наименьших квадратов

В регрессии по методу наименьших квадратов обучающие данные содержат n различных тренировочных пар $(\bar{X}_1, y_1) \dots (\bar{X}_n, y_n)$, где \bar{X}_i — это d -мерное представление точек данных, а y_i — вещественное целевое значение. Тот факт, что целевое значение — вещественное, важен, поскольку в этом случае базовая

задача называется *регрессией*, а не *классификацией*. Регрессия по методу наименьших квадратов — это старейшая из всех задач обучения, а методы градиентного спуска, предложенные Тихоновым и Арсениным в 1970-х годах [499], очень тесно связаны с правилами обновления методом градиентного спуска Розенблатта [405] для алгоритма перцептрона. В действительности, как мы далее увидим, регрессию по методу наименьших квадратов можно использовать также по отношению к бинарным целевым значениям, “сделав вид”, что эти значения — вещественные. Результирующий подход эквивалентен алгоритму обучения Уидроу — Хоффа, известному в литературе по нейронным сетям как второй алгоритм обучения, предложенный после перцептрона.

В регрессии по методу наименьших квадратов целевая переменная связана с переменными признаков следующим соотношением:

$$\hat{y}_i = \bar{W} \cdot \bar{X}_i. \quad (2.5)$$

Обратите внимание на символ циркумфлекса в обозначении \hat{y}_i , указывающий на то, что это предсказанное значение. В уравнении 2.5 смещение отсутствует. На протяжении всего раздела будет предполагаться, что один из признаков в тренировочных данных имеет постоянное значение 1, а коэффициент при таком фиктивном признаке равен смещению. Это стандартный прием конструирования признаков, заимствованный из традиционного машинного обучения. В нейронных сетях смещение часто представляется нейроном смещения (см. раздел 1.2.1) с постоянным выходом 1. И хотя нейрон смещения почти всегда используется в реальных задачах, мы не будем отображать его в явном виде ради простоты.

Ошибка предсказания, e_i , дается выражением $e_i = (y_i - \hat{y}_i)$. Вектор $\bar{W} = (w_1 \dots w_d)$ — это d -мерный вектор коэффициентов, подлежащий обучению таким образом, чтобы минимизировать общую квадратичную ошибку на тренировочных данных, равную $\sum_{i=1}^n e_i^2$. Часть потерь, обусловленная i -м тренировочным примером, дается следующим выражением:

$$L_i = e_i^2 = (y_i - \hat{y}_i)^2. \quad (2.6)$$

Эту функцию потерь можно имитировать с помощью архитектуры, аналогичной перцептрону, с тем лишь исключением, что квадратичная функция потерь дополняется тождественной функцией активации. Эта архитектура представлена на рис. 2.3, в, тогда как архитектура перцептрона — на рис. 2.3 а. Перед перцептроном и регрессией по методу наименьших квадратов стоит одна и та же задача — минимизировать ошибку предсказания. В то же время, поскольку функция потерь в задаче классификации по своей сути дискретная, алгоритм перцептрона использует гладкую аппроксимацию желаемого

целевого значения. Результатом этого является *сглаженный критерий перцептрона*, представленный на рис. 2.3, б. Как будет показано далее, правило обновления методом градиентного спуска в регрессии по методу наименьших квадратов очень напоминает аналогичное правило для перцептрона, с тем основным отличием, что в регрессии используются ошибки в виде вещественных значений, а не дискретные ошибки из набора значений $\{-2, +2\}$.

Как и в алгоритме перцептрона, шаги стохастического градиентного спуска определяются путем вычисления градиента e^2 по \bar{W} , когда нейронной сети предъявляется тренировочная пара (\bar{X}_i, y_i) . Этот градиент можно вычислить с помощью следующей формулы:

$$\frac{\partial e_i}{\partial \bar{W}} = -e_i \bar{X}_i. \quad (2.7)$$

Поэтому обновления \bar{W} по методу градиентного спуска вычисляются с использованием приведенного выше градиента и размера шага α :

$$\bar{W} \leftarrow \bar{W} + \alpha e_i \bar{X}_i.$$

Это обновление можно переписать в следующем виде:

$$\bar{W} \leftarrow \bar{W} + \alpha (y_i - \hat{y}_i) \bar{X}_i. \quad (2.8)$$

Получаемые в ходе градиентного спуска обновления для регрессии по методу наименьших квадратов можно видоизменить таким образом, чтобы они включали факторы забывания. Добавление регуляризации эквивалентно штрафованию функции потерь классификации по методу наименьших квадратов дополнительным членом, пропорциональным $\lambda \cdot \|\bar{W}\|^2$, где $\lambda > 0$ — параметр регуляризации. С учетом регуляризации обновление приобретает следующий вид:

$$\bar{W} \leftarrow \bar{W} (1 - \alpha \cdot \lambda) + \alpha (y_i - \hat{y}_i) \bar{X}_i. \quad (2.9)$$

Обратите внимание на то, что обновление в этой форме выглядит почти так же, как и обновление для перцептрона в уравнении 2.2. Однако эти обновления не идентичны, поскольку предсказываемое значение \hat{y}_i вычисляется по-разному в обоих случаях. В случае перцептрона для вычисления бинарного значения \hat{y}_i к произведению $\bar{W} \cdot \bar{X}_i$ применяется функция *sign*, вследствие чего ошибка $(y_i - \hat{y}_i)$ может иметь только одно из двух значений: $\{-2, +2\}$. В регрессии по методу наименьших квадратов предсказание \hat{y}_i представляет собой вещественное значение, к которому функция *sign* не применяется.

В связи с этим может возникнуть естественный вопрос: а что если применить регрессию по методу наименьших квадратов непосредственно для минимизации квадрата расстояния *вещественного* прогнозного значения \hat{y}_i от наблюдаемых бинарных целевых значений $y_i \in \{-1, +1\}$? Непосредственное

применение регрессии по методу наименьших квадратов к бинарным целевым значениям называют *классификацией по методу наименьших квадратов*. Выражение для обновления методом градиентного спуска совпадает с тем, которое указано в уравнении 2.9 и внешне похоже на аналогичное выражение для перцептрона. Однако классификация по методу наименьших квадратов не приводит к тем же результатам, что и алгоритм перцептрона, поскольку ошибки обучения в виде *вещественных значений* ($y_i - \hat{y}_i$) в первом случае вычисляются иначе, нежели *целочисленные* ошибки ($y_i - \hat{y}_i$) во втором. Непосредственное приложение регрессии по методу наименьших квадратов к бинарным целевым выходам называют *методом обучения Уидроу — Хоффа*.

2.2.2.1. Метод обучения Уидроу — Хоффа

В 1960 году Уидроу и Хофф расширили правила обучения перцептрона. Однако их метод не является принципиально новым, поскольку он представляет собой непосредственное применение регрессии по методу наименьших квадратов к бинарным целевым значениям. Несмотря на то что вещественные значения *предсказаний для неизвестных тестовых образцов* преобразуются в бинарные выходы с помощью функция *sign*, ошибка для *обучающих* примеров вычисляется (в отличие от перцептрона) непосредственно с использованием вещественных предсказаний. Поэтому данный метод также называют *классификацией по методу наименьших квадратов* или *линейным методом наименьших квадратов* [6]. Примечательно то, что, казалось бы, не имеющий к этому отношения метод, предложенный в 1936 году и известный как дискриминант Фишера, также сводится к методу обучения Уидроу — Хоффа в специальном случае бинарных целевых выходов.

Дискриминант Фишера формально определяется как направленный вектор \bar{W} , вдоль которого максимизируется отношение междуклассовой дисперсии спроецированных данных к внутриклассовой дисперсии. Выбирая скаляр b для определения гиперплоскости $\bar{W} \cdot \bar{X} = b$, можно моделировать разделение двух классов. Для классификации используется именно эта гиперплоскость. Несмотря на то что дискриминант Фишера, на первый взгляд, отличается от регрессии/классификации по методу наименьших квадратов, примечательно то, что дискриминант Фишера для бинарных целевых выходов идентичен регрессии по методу наименьших квадратов, примененной к бинарным целям (т.е. классификации по методу наименьших квадратов). Как данные, так и целевые значения должны быть центрированными на среднем значении, что позволяет установить переменную смещения равной нулю. В литературе приведено несколько доказательств этого результата [3, 6, 40, 41].

Нейронная архитектура для классификации методом Уидроу — Хоффа показана на рис. 2.3, в. Шаги градиентного спуска как для перцептрона, так и для

метода Уидроу — Хоффа даются уравнением 2.8, если не считать различий в способе вычисления ошибки $(y_i - \hat{y}_i)$. В случае перцептрона она всегда будет равна одному из значений $\{-2, +2\}$. В случае метода Уидроу — Хоффа ошибки могут иметь произвольные вещественные значения, поскольку \hat{y}_i устанавливается равным $\bar{W} \cdot \bar{X}_i$ без применения функции *sign*. Это отличие играет важную роль, поскольку алгоритм перцептрона никогда не штрафует точки данных положительного класса для “слишком правильных” (т.е. больших, чем 1) значений $\bar{W} \cdot \bar{X}_i$, в то время как использование вещественных предсказанных значений для вычисления ошибки сопровождается неблагоприятным эффектом в виде штрафования таких точек. Такое неуместное штрафование за превышение целевых значений является ахиллесовой пятой метода обучения Уидроу — Хоффа и дискриминанта Фишера [6].

Следует подчеркнуть, что регрессия/классификация по методу наименьших квадратов, метод обучения Уидроу — Хоффа и дискриминант Фишера были предложены в разное время разными сообществами исследователей. В действительности дискриминант Фишера, старейший из этих методов, который датируется далеким 1936 годом, часто рассматривается как метод нахождения направлений, характеризующихся хорошим разделением спроецированных классов, а не как классификатор. Однако его можно применять и в качестве классификатора, используя результирующее направление \bar{W} для создания линейного предсказания. Полная несхожесть исходных предпосылок и, казалось бы, различная мотивация создания этих методов делают эквивалентность обеспечиваемых ими решений еще более приметной. Правилу обучения Уидроу — Хоффа соответствует модель *Adaline*, название которой является сокращением от “*adaptive linear neuron*” (адаптивный линейный нейрон). Его также называют *дельта-правил*ом. Вспомним, что в случае бинарных целевых значений $\{-1, +1\}$ к правилу обучения, выраженному уравнением 2.8, применим ряд альтернативных названий: “классификация по методу наименьших квадратов”, “алгоритм наименьших средних квадратов” (least mean-squares algorithm — LMS), “дискриминантный классификатор Фишера”², “правило обучения Уидроу — Хоффа”, “дельта-правило” и “Adaline”. Это является отражением того факта, что семейство методов классификации по принципу наименьших квадратов заново открывалось в литературе несколько раз под разными названиями и с различной мотивацией.

² Чтобы получить в точности то же направление, что и в методе Фишера с уравнением 2.8, очень важно центрировать вокруг среднего значения как переменные признаки, так и бинарные целевые значения. Поэтому каждое бинарное целевое значение будет одним из двух вещественных значений с различными знаками. Вещественное значение будет выражать долю примеров, принадлежащих к другому классу. Альтернативный вариант заключается в использовании нейрона смещения, поглощающего постоянный сдвиг.

Учитывая бинарность ее откликов, функцию потерь метода Уидроу — Хоффа можно переписать в несколько видоизмененном по сравнению с регрессией по методу наименьших квадратов виде:

$$\begin{aligned} L_i &= (y_i - \hat{y}_i)^2 = \underbrace{y_i^2}_{1} (y_i - \hat{y}_i)^2 = \\ &= (\underbrace{y_i^2}_{1} - \hat{y}_i y_i)^2 = (1 - \hat{y}_i y_i)^2. \end{aligned}$$

Этот тип кодирования возможен в том случае, если целевая переменная y_i извлекается из набора $\{-1, +1\}$, поскольку тогда мы можем воспользоваться равенством $y_i^2 = 1$. Целевую функцию метода Уидроу — Хоффа полезно преобразовать в эту форму, так как в этом случае ее будет легче связать с другими целевыми функциями других моделей, таких как перцептрон и метод опорных векторов. Например, функцию потерь метода опорных векторов можно получить, подправив приведенную выше функцию потерь таким образом, чтобы превышение целевых значений не штрафовалось. Функцию потерь можно скорректировать, заменив целевую функцию на $[\max\{(1 - \hat{y}_i y_i), 0\}]^2$, что дает L_2 -функцию потерь метода опорных векторов (SVM) Хинтона с L_2 -регуляризацией [190]. Можно показать, что почти все модели бинарной классификации, обсуждаемые в этой главе, тесно связаны с функцией потерь Уидроу — Хоффа вследствие того, что корректируют функции потерь различными способами, дабы избежать штрафования значений, превышающих целевое.

Обновления с помощью градиентного спуска (см. уравнение 2.9) в регрессии по методу наименьших квадратов можно несколько видоизменить для обучения по методу Уидроу — Хоффа, приняв во внимание бинарность переменных отклика:

$$\begin{aligned} \bar{W} &\leftarrow \bar{W}(1 - \alpha \cdot \lambda) + \alpha(y_i - \hat{y}_i)\bar{X} \quad [\text{для числовых и бинарных откликов}], \\ &= \bar{W}(1 - \alpha \cdot \lambda) + \alpha y_i(1 - y_i \hat{y}_i)\bar{X} \quad [\text{только для бинарных откликов, поскольку } y_i^2 = 1]. \end{aligned}$$

Вторая форма обновления полезна в плане установления ее взаимосвязи с выражениями обновлений для перцептрона и SVM, в каждом из которых вместо члена $(1 - y_i \hat{y}_i)$ фигурирует индикаторная переменная, являющаяся функцией $y_i \hat{y}_i$. Этот вопрос будет обсуждаться в следующем разделе.

2.2.2.2. Решения в замкнутой форме

Специальным случаем регрессии и классификации по методу наименьших квадратов является решение в замкнутой форме (без градиентного спуска), получаемое путем *псевдообращения* матрицы обучающих данных D размера $n \times d$, строки которой — векторы $\bar{X}_1 \dots \bar{X}_n$. Обозначим n -мерный вектор-столбец зави-

симых переменных как $\bar{y} = [y_1 \dots y_n]^T$. Псевдообращение матрицы D определяется следующим образом:

$$D^+ = (D^T D)^{-1} D^T. \quad (2.10)$$

Далее вектор-строка \bar{W} определяется с помощью следующего соотношения:

$$\bar{W}^T = D^+ \bar{y}. \quad (2.11)$$

Если используется регуляризация, то вектор коэффициентов \bar{W} приобретает следующий вид:

$$\bar{W}^T = (D^T D + \lambda I)^{-1} D^T \bar{y}, \quad (2.12)$$

где $\lambda > 0$ — параметр регуляризации. В то же время обращение матриц наподобие $(D^T D + \lambda I)$ обычно осуществляют с помощью численных методов, в любом случае требующих использования градиентного спуска. Однако к фактическому обращению таких больших матриц, как $D^T D$, прибегают редко. В действительности обновления метода Уидроу — Хоффа обеспечивают эффективный способ решения этой задачи без использования решения в замкнутой форме.

2.2.3. Логистическая регрессия

Логистическая регрессия — это вероятностная модель, классифицирующая экземпляры в терминах вероятностей. В силу вероятностного характера такой классификации вполне естественно требовать, чтобы оптимизация параметров гарантировала получение как можно большего значения вероятности предсказания наблюдаемого класса для каждого примера. Это достигается за счет обучения параметров модели с привлечением понятия *оценки максимального правдоподобия*. Правдоподобие тренировочных данных определяется как произведение вероятностей наблюдаемых меток каждого тренировочного примера. Очевидно, что чем больше значение этой целевой функции, тем лучше. Используя отрицательный логарифм данного значения, мы получаем функцию потерь для задачи минимизации. Поэтому в выходном узле в качестве функции потерь используется отрицательное *логарифмическое правдоподобие*. Эта функция потерь заменяет квадратичную ошибку, используемую в методе Уидроу — Хоффа. В таком случае выходной слой может использовать сигмоидную функцию активации, что является общепринятой практикой при проектировании нейронных сетей.

Обозначим через $(\bar{X}_1, y_1), (\bar{X}_2, y_2), \dots, (\bar{X}_n, y_n)$ набор из n тренировочных пар, в котором \bar{X}_i содержит d -мерные признаки, а $y_i \in \{-1, +1\}$ — бинарная переменная класса. Как и в случае перцептрона, используется однослойная архитектура с весами $\bar{W} = (w_1 \dots w_d)$. Вместо того чтобы применять к произведению

$\bar{W} \cdot \bar{X}_i$ ступенчатую активацию *sign* для предсказания значений y_i , в логистической регрессии используется сглаженная сигмоидная функция активации, применение которой к произведению $\bar{W} \cdot \bar{X}_i$ дает оценку *вероятности* того, что y_i равно 1:

$$\hat{y}_i = P(y_i = 1) = \frac{1}{1 + \exp(-\bar{W} \cdot \bar{X}_i)}. \quad (2.13)$$

Принадлежность к определенному классу может быть предсказана для тех образцов, прогнозируемая вероятность которых превышает значение 0,5. Обратите внимание на то, что $P(y_i = 1)$ равно 0,5, если $\bar{W} \cdot \bar{X}_i = 0$, а \bar{X}_i лежит на разделяющей гиперплоскости. Смещение \bar{X}_i в любом направлении от этой гиперплоскости приводит к изменениям знака $\bar{W} \cdot \bar{X}_i$ и соответствующим смещениям значений вероятности. Поэтому знак произведения $\bar{W} \cdot \bar{X}_i$ также дает то же предсказание, что и выбор класса в соответствии со значением вероятности, превышающим 0,5.

Приступим к описанию того, как установить функцию потерь, соответствующую оценке правдоподобия. Эта методология очень важна, поскольку широко применяется во многих нейронных моделях. В случае положительных образцов тренировочных данных мы хотим максимизировать величину $P(y_i = 1)$, в случае отрицательных — величину $P(y_i = -1)$. Для положительных образцов, удовлетворяющих условию $y_i = 1$, мы максимизируем \hat{y}_i , а для отрицательных, удовлетворяющих условию $y_i = -1$, — величину $1 - \hat{y}_i$. Эти две разные формы максимизации можно записать в виде одного консолидированного выражения: $|y_i / 2 - 0,5 + \hat{y}_i|$. Теперь мы должны максимизировать следующую функцию правдоподобия, которая представляет собой произведение вероятностей, вычисленных для каждого из тренировочных примеров:

$$\mathcal{L} = \prod_{i=1}^n |y_i / 2 - 0,5 + \hat{y}_i|. \quad (2.14)$$

Поэтому функция потерь устанавливается равной $L_i = -\log(|y_i / 2 - 0,5 + \hat{y}_i|)$ для каждого тренировочного примера, что позволяет преобразовать максимизируемую функцию из мультипликативной формы в аддитивную:

$$\mathcal{L}\mathcal{L} = -\log(\mathcal{L}) = \sum_{i=1}^n \underbrace{-\log(|y_i / 2 - 0,5 + \hat{y}_i|)}_{L_i}. \quad (2.15)$$

Аддитивные формы целевой функции особенно удобны для обновлений стохастического градиентного спуска, распространенных в нейронных сетях. Общая архитектура и функция потерь этой сети приведены на рис. 2.3, з. Предсказанная для каждого тренировочного примера вероятность \hat{y}_i вычисляется путем

его пропускания через нейронную сеть, а соответствующий ему градиент определяется с помощью функции потерь.

Обозначим функцию потерь для i -го тренировочного примера через L_i , как она и обозначена в уравнении 2.15. Тогда градиент L_i по весам \bar{W} можно вычислить с помощью следующей формулы:

$$\begin{aligned} \frac{\partial L_i}{\partial \bar{W}} &= -\frac{\text{sign}(y_i / 2 - 0,5 + \hat{y}_i)}{|y_i / 2 - 0,5 + \hat{y}_i|} \frac{\partial \hat{y}_i}{\partial \bar{W}} = \\ &= -\frac{\text{sign}(y_i / 2 - 0,5 + \hat{y}_i)}{|y_i / 2 - 0,5 + \hat{y}_i|} \cdot \frac{\bar{X}_i}{1 + \exp(-\bar{W} \cdot \bar{X}_i)} \cdot \frac{1}{1 + \exp(\bar{W} \cdot \bar{X}_i)} = \\ &= \begin{cases} -\frac{\bar{X}_i}{1 + \exp(\bar{W} \cdot \bar{X}_i)}, & \text{если } y_i = 1; \\ \frac{\bar{X}_i}{1 + \exp(-\bar{W} \cdot \bar{X}_i)}, & \text{если } y_i = -1. \end{cases} \end{aligned}$$

Заметьте, что приведенный выше градиент можно переписать в более сжатом виде:

$$\frac{\partial L_i}{\partial \bar{W}} = -\frac{y_i \bar{X}_i}{1 + \exp(y_i \bar{W} \cdot \bar{X}_i)} = -[\text{Вероятность ошибки для } (\bar{X}_i, y_i)] (y_i \bar{X}_i). \quad (2.16)$$

Поэтому обновления методом градиентного спуска для логистической регрессии даются следующим выражением (включая регуляризацию):

$$\bar{W} \leftarrow \bar{W}(1 - \alpha\lambda) + \alpha \frac{y_i \bar{X}_i}{1 + \exp[y_i (\bar{W} \cdot \bar{X}_i)]}. \quad (2.17)$$

Точно так же, как в алгоритмах перцептрона и Уидроу — Хоффа для вычисления обновлений используются *величины* ошибок, в методе регрессии для этой цели используются *вероятности* ошибок. Это вполне естественное расширение вероятностной природы функции потерь на обновления.

2.2.3.1. Альтернативные варианты выбора функции активации и функции потерь

Ту же модель можно реализовать, используя другие варианты выбора функции активации и функции потерь в выходном узле, коль скоро они комбинируются для достижения того же результата. Вместо того чтобы использовать сигмоидную активацию для создания выхода $\hat{y}_i \in (0, 1)$, мы можем применить

тождественную функцию активации для создания выхода $\hat{y}_i \in (-\infty, +\infty)$, а затем задействовать следующую функцию потерь:

$$L_i = \log(1 + \exp(-y_i \cdot \hat{y}_i)). \quad (2.18)$$

Альтернативная архитектура для логистической регрессии приведена на рис. 2.3, д. Окончательные предсказания для тестовых образцов можно получать, применяя функцию $sign$ к \hat{y}_i , что эквивалентно отнесению их к тем классам, для которых вероятность больше 0,5. Этот пример демонстрирует возможность реализации той же модели с помощью различных комбинаций функций активации и функций потерь, если они комбинируются для достижения того же результата.

Благоприятной стороной использования тождественной функции активации для определения \hat{y}_i является то, что это согласуется со способом определения функций потерь в других моделях, таких как перцептрон и модель обучения Уидроу — Хоффа. Кроме того, функция потерь, определяемая уравнением 2.18, содержит произведение y_i и \hat{y}_i , как в других моделях. Это обеспечивает возможность непосредственного сравнения функций потерь в различных моделях, что и послужит предметом нашего исследования в этой главе.

2.2.4. Метод опорных векторов

Функции потерь, используемые в методе опорных векторов и логистической регрессии, тесно взаимосвязаны. Однако в методе опорных векторов вместо гладкой функции (как в уравнении 2.18) используется кусочно-линейная функция потерь.

Рассмотрим набор тренировочных данных, который состоит из n примеров, обозначенных как (\bar{X}_1, y_1) , (\bar{X}_2, y_2) , ..., (\bar{X}_n, y_n) . Нейронная архитектура метода опорных векторов аналогична архитектуре классификатора методом наименьших квадратов (модель Уидроу — Хоффа). Основное отличие заключается в выборе функции потерь. Как и в случае классификации методом наименьших квадратов, предсказание \hat{y}_i для тренировочной точки \bar{X}_i получают, применяя тождественную функцию активации к произведению $\bar{W} \cdot \bar{X}_i$. Здесь $\bar{W} = (w_1 \cdot w_d)$ содержит вектор, компонентами которого являются d весов, соответствующих d различным входам однослойной сети. Поэтому выходом нейронной сети для вычисления функции потерь является $\hat{y}_i = \bar{W} \cdot \bar{X}_i$, хотя предсказания для тестовых образцов получают путем применения к выходу функции $sign$.

Функция потерь L_i для i -го тренировочного примера в методе опорных векторов определяется так:

$$L_i = \max\{0, 1 - y_i \hat{y}_i\}. \quad (2.19)$$

Эти потери называют *кусочно-линейными* (hinge-loss), а соответствующая нейронная архитектура приведена на рис. 2.3, е. Суть общей идеи, лежащей в основе этой функции потерь, заключается в том, что положительные тренировочные примеры должны штрафовать лишь в случае значений, меньших 1, а отрицательные — лишь в случае значений, больших, чем -1 . В обоих случаях величина штрафа ведет себя линейно, а при превышении порога ведет себя как константа. Это свойство оказывается полезным при сравнении данной функции потерь со значением потерь $(1 - y_i \hat{y}_i)^2$ в методе Уидроу — Хоффа, в котором предсказания штрафуются за отличие от целевых значений. Как будет показано далее, это отличие является важным преимуществом метода опорных векторов по сравнению с функцией потерь Уидроу — Хоффа.

Различия между функциями потерь перцептрона, метода Уидроу — Хоффа, логистической регрессии и метода опорных векторов проиллюстрированы на рис. 2.4, на котором приведены значения потерь для одного положительного

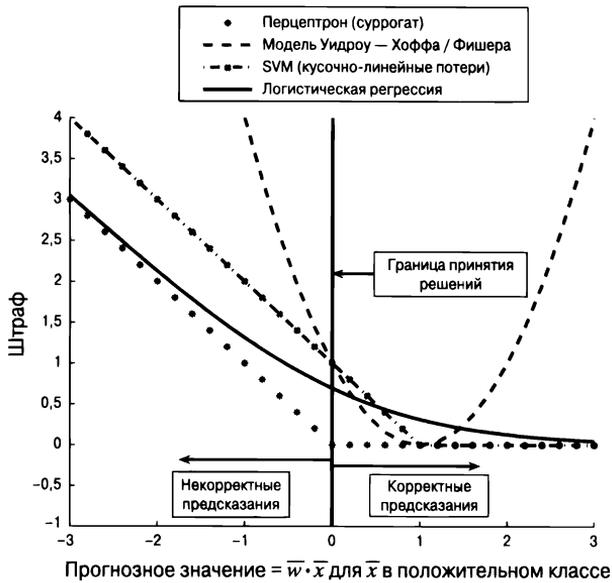


Рис. 2.4. Функции потерь в различных вариантах перцептрона. Можно отметить следующие важные моменты: а) функция потерь SVM смещена ровно на одну единицу вправо относительно (суррогатной) функции потерь перцептрона; б) логистическая функция потерь — это сглаженный вариант функции потерь SVM; в) функция потерь в методе Уидроу — Хоффа — единственный случай, когда размер штрафа увеличивается для “слишком верно” классифицированных точек (т.е. точек \bar{X}_i , относящихся к положительному классу, для которых значение $\bar{W} \cdot \bar{X}_i$ превышает $+1$), причем корректировка этой функции путем ее установки в 0 при $\bar{W} \cdot \bar{X}_i > 1$ приводит к квадратичной функции потерь SVM [190]

тренировочного примера при различных значениях $\hat{y}_i = \bar{W} \cdot \bar{X}_i$. В случае перцептрона отображена лишь сглаженная *суррогатная* функция потерь (см. раздел 1.2.1.1). Поскольку целевое значение равно +1, то улучшение, обеспечиваемое штрафованием функции потерь в случае логистической регрессии, уменьшается при превышении произведением $\bar{W} \cdot \bar{X}_i$ значения +1. В случае метода опорных векторов кусочно-линейная функция потерь сохраняет постоянное значение за пределами этой точки. Другими словами, штрафуются только неверно классифицируемые точки или точки, расположенные в непосредственной близости от границы принятия решений $\bar{W} \cdot \bar{X}_i = 0$. Критерий перцептрона имеет ту же форму, что и кусочно-линейная функция потерь, но смещен на одну единицу влево. Метод Уидроу — Хоффа единственный, в котором положительная тренировочная точка штрафуются за слишком большие положительные значения $\bar{W} \cdot \bar{X}_i$. Иными словами, метод Уидроу — Хоффа штрафует точки в случае “чрезмерно” верной классификации. Эта особенность, вследствие которой хорошо разделяемые точки становятся источниками проблем в процессе тренировки, несколько затрудняет использование целевой функции Уидроу — Хоффа.

Метод стохастического градиентного спуска предполагает вычисление градиентов функций потерь L_i по элементам \bar{W} для семплируемых точек. Градиент вычисляется согласно следующей формуле:

$$\frac{\partial L_i}{\partial \bar{W}} = \begin{cases} -y_i \bar{X}_i, & \text{если } y_i \hat{y}_i < 1, \\ 0 & \text{в противном случае.} \end{cases} \quad (2.20)$$

Таким образом, в методе стохастического градиентного спуска для каждой семплированной точки проверяется справедливость неравенства $y_i \hat{y}_i < 1$. Если это условие выполняется, то обновление пропорционально $y_i \bar{X}_i$:

$$\bar{W} \leftarrow \bar{W}(1 - \alpha\lambda) + \alpha y_i \bar{X}_i [I(y_i \hat{y}_i < 1)], \quad (2.21)$$

где $I(\cdot) \in \{0, 1\}$ — индикаторная функция, которая имеет значение 1 в случае выполнения условия, заданного ее аргументом. Этот подход представляет собой простейшую версию первоначального обновления для SVM [448]. Читателю предлагается самостоятельно убедиться в том, что обновление в этой форме совпадает с обновлением для (регуляризованного) перцептрона (см. уравнение 2.3), за исключением того, что в случае перцептрона условие этого обновления имеет вид $y_i \hat{y}_i < 0$. Поэтому перцептрон выполняет обновление только для неверно классифицированных точек, тогда как в методе опорных векторов обновляются также точки, классифицированные верно, хотя и не со стопроцентной долей вероятности. В основе схожести этих функций лежит тот факт, что функция потерь, применяемая в критерии перцептрона (см. рис. 2.4), про-

сто смещена относительно кусочно-линейной функции потерь SVM. Чтобы облегчить сравнительный анализ функций стоимости, используемых различными методами, они сведены в представленной ниже таблице.

Модель	Функция потерь L_i для (\bar{X}_i, y_i)
Перцептрон (сглаженный суррогат)	$\max\{0, -y_i \cdot (\bar{W} \cdot \bar{X}_i)\}$
Метод Уидроу — Хоффа / Фишера	$(y_i - \bar{W} \cdot \bar{X}_i)^2 = \{1 - y_i \cdot (\bar{W} \cdot \bar{X}_i)\}^2$
Логистическая регрессия	$\log(1 + \exp[-y_i(\bar{W} \cdot \bar{X}_i)])$
Метод опорных векторов (кусочно-линейные потери)	$\max\{0, 1 - y_i \cdot (\bar{W} \cdot \bar{X}_i)\}$
Метод опорных векторов (функция стоимости Хинтона с L_2 -регуляризацией) [190]	$[\max\{0, 1 - y_i \cdot (\bar{W} \cdot \bar{X}_i)\}]^2$

Следует подчеркнуть, что все обновления, приведенные в этом разделе, обычно соответствуют обновлениям по методу стохастического градиентного спуска, которые встречаются как в традиционном машинном обучении, так и в нейронных сетях. Обновления остаются одними и теми же независимо от того, используем ли мы нейронную архитектуру для представления моделей, соответствующих этим алгоритмам. Основная цель вышеприведенных рассуждений — показать, что элементарные специальные случаи нейронных сетей представляют собой реализацию алгоритмов, хорошо известных в литературе по машинному обучению. Важный вывод заключается в том, что с ростом объема доступных данных появляется возможность включения дополнительных узлов и увеличения глубины сети для увеличения емкости модели, что и объясняет причины превосходства нейронных сетей при работе с крупными наборами данных (см. рис. 2.1).

2.3. Архитектуры нейронных сетей для мультиклассовых моделей

Каждая из моделей, рассмотренных нами до сих пор, предназначена для бинарной классификации. В этом разделе мы обсудим возможности проектирования моделей мультиклассовой классификации путем незначительного изменения архитектуры перцептрона и включения в нее множественных выходных узлов.

2.3.1. Мультиклассовый перцептрон

Рассмотрим задачу с k различными классами. Каждый тренировочный пример $(\bar{X}_i, c(i))$ содержит d -мерный вектор признаков \bar{X}_i и индекс $c(i) \in \{1 \dots k\}$

наблюдаемого класса. В данном случае мы хотим найти одновременно k различных разделительных гиперплоскостей (разделителей) $\bar{W}_1 \dots \bar{W}_k$, таких, что значение $\bar{W}_{c(i)} \cdot \bar{X}_i$ больше, чем значение $\bar{W}_r \cdot \bar{X}_i$, для каждого класса $r \neq c(i)$. Это объясняется тем, что для примера \bar{X}_i всегда предсказывается принадлежность к тому классу r , который имеет наибольшее значение $\bar{W}_r \cdot \bar{X}_i$. Поэтому в случае мультиклассового перцептрона функция потерь для i -го тренировочного примера определяется следующей формулой:

$$\max_{r:r \neq c(i)} \max(\bar{W}_r \cdot \bar{X}_i - \bar{W}_{c(i)} \cdot \bar{X}_i, 0). \quad (2.22)$$

Архитектура мультиклассового перцептрона показана на рис. 2.5, а. Как и во всех моделях нейронных сетей, для определения обновлений можно использовать метод градиентного спуска. Для корректно классифицированных примеров градиент всегда равен нулю, и они не нуждаются в обновлении. Для неверно классифицированного примера градиент выражается следующей формулой:

$$\frac{\partial L_i}{\partial \bar{W}_r} = \begin{cases} -\bar{X}_i, & \text{если } r = c(i), \\ \bar{X}_i, & \text{если } r \neq c(i) \text{ — наиболее неверное предсказание,} \\ 0 & \text{в противном случае.} \end{cases} \quad (2.23)$$

Поэтому процедура применения метода стохастического градиентного спуска выглядит следующим образом. Каждый тренировочный пример передается сети. Если наибольшее выходное значение $\bar{W}_r \cdot \bar{X}_i$ получает корректный класс $r = c(i)$, то обновление не требуется. В противном случае каждый разделитель \bar{W}_r обновляется со скоростью обучения $\alpha > 0$ в соответствии со следующими формулами:

$$\bar{W}_r \leftarrow \bar{W}_r + \begin{cases} \alpha \bar{X}_i, & \text{если } r = c(i), \\ -\alpha \bar{X}_i, & \text{если } r \neq c(i) \text{ — наиболее неверное предсказание,} \\ 0 & \text{в противном случае.} \end{cases} \quad (2.24)$$

Каждый раз обновляются только два разделителя. В том специальном случае, когда $k = 2$, эти градиентные обновления сводятся к перцептрону, поскольку оба разделителя, \bar{W}_1 и \bar{W}_2 , будут связаны соотношением $\bar{W}_1 - \bar{W}_2$, если спуск начат в точке $\bar{W}_1 = \bar{W}_2 = 0$. Другая особенность, специфическая для нерегуляризованного перцептрона, заключается в возможности использования скорости обучения $\alpha = 1$, не влияя на процесс обучения, поскольку значение α лишь масштабирует вес, если начинать с точки $\bar{W}_j = 0$ (см. упражнение 2). Однако другие линейные модели этим свойством не обладают, поскольку в них значение α влияет на процесс обучения.

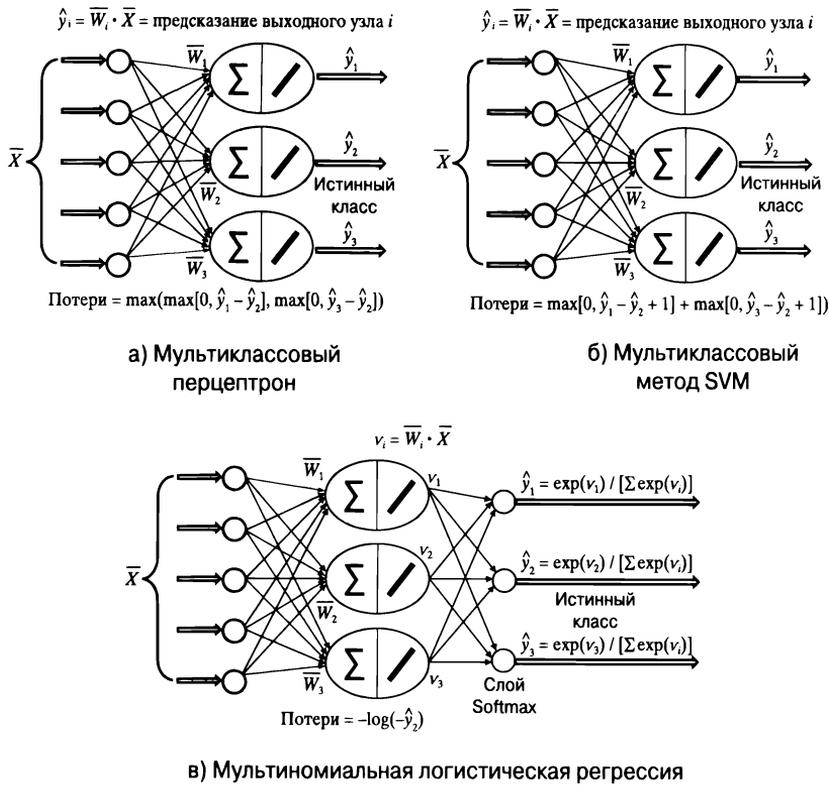


Рис. 2.5. Мультиклассовые модели; во всех случаях предполагается, что истинным является класс 2

2.3.2. SVM Уэстона — Уоткинса

SVM Уэстона — Уоткинса [529] изменяет многослойный перцептрон в двух отношениях.

1. В мультиклассовом перцептрон обновляется лишь линейный разделитель класса, который предсказан *наиболее* неправильно, а также линейный разделитель истинного класса. С другой стороны, в SVM Уэстона — Уоткинса обновляется разделитель *любого* класса, для которого предсказание более предпочтительно, чем для истинного класса. В обоих случаях разделитель наблюдаемого класса обновляется на ту же агрегированную величину, что и некорректные классы (но в противоположном направлении).
2. В SVM Уэстона — Уоткинса разделитель обновляется не только в случае неверной классификации, но и в тех случаях, когда некорректный класс получает предсказание, “неприлично близкое” к истинному классу. Этот подход основан на понятии *зазора* между классами.

Как и в случае мультиклассового перцептрона, обозначим i -й тренировочный пример как $(\bar{X}_i, c(i))$, где \bar{X}_i содержит d -мерные переменные признаков, а $c(i)$ содержит индекс класса, получаемый из набора значений $\{1 \dots k\}$. Мы хотим обучить d -мерные коэффициенты $\bar{W}_1 \dots \bar{W}_k$ линейных разделителей таким образом, чтобы для индекса класса r с наибольшим значением $\bar{W}_r \cdot \bar{X}_i$ предсказывался корректный класс $c(i)$. В методе SVM Уэстона — Уоткинса функция потерь L_i для i -го тренировочного примера $(\bar{X}_i, c(i))$ определяется следующим выражением:

$$L_i = \sum_{r:r \neq c(i)} \max(\bar{W}_r \cdot \bar{X}_i - \bar{W}_{c(i)} \cdot \bar{X}_i + 1, 0). \quad (2.25)$$

Нейронная архитектура SVM Уэстона–Уоткинса приведена на рис. 2.5, б. Представляет интерес сравнить между собой целевые функции, используемые в SVM Уэстона — Уоткинса (уравнение 2.25) и мультиклассовом перцептроне (уравнение 2.22). Во-первых, для каждого класса $r \neq c(i)$ потери начисляются в том случае, если значение предсказания $\bar{W}_r \cdot \bar{X}_i$ отличается от аналогичного значения для истинного класса в меньшую сторону не более чем на величину зазора, равную 1. Кроме того, потери для всех подобных классов $r \neq c(i)$ суммируются, а не берется максимальная из них. Учет этих двух факторов формализует отмеченные выше два различия между обеими моделями.

Чтобы определить обновления по методу градиентного спуска, можно найти градиенты функции потерь по каждому \bar{W}_r . В тех случаях, когда функция потерь L_i равна 0, ее градиент также равен 0. Поэтому, если тренировочный пример корректно классифицирован с достаточным зазором по отношению ко второму по качеству классификации классу, то обновление не требуется. Но если функция потерь имеет ненулевое значение, то мы имеем дело либо с неверным, либо с “едва корректным” предсказанием, когда лучший и второй по качеству классификации классы разделены недостаточно надежно. В подобных случаях градиент функции потерь имеет ненулевое значение. Функция потерь, описываемая уравнением 2.25, создается путем суммирования вкладов $(k - 1)$ разделителей, принадлежащих к некорректным классам. Обозначим через $\delta(r, \bar{X}_i)$ индикаторную функцию 0/1, которая имеет значение 1, если разделитель r -го класса вносит положительный вклад в функцию потерь, определяемую уравнением 2.25. В этом случае градиент функции потерь вычисляется следующим образом:

$$\frac{\partial L_i}{\partial \bar{W}_r} = \begin{cases} -\bar{X}_i \left[\sum_{j \neq r} \delta(j, \bar{X}_i) \right], & \text{если } r = c(i), \\ \bar{X}_i [\delta(r, \bar{X}_i)], & \text{если } r \neq c(i). \end{cases} \quad (2.26)$$

Это приводит к следующим шагам стохастического градиентного спуска для r -го сепаратора \bar{W}_r со скоростью обучения α :

$$\bar{W}_r \leftarrow \bar{W}_r(1 - \alpha\lambda) + \alpha \begin{cases} \bar{X}_i \left[\sum_{j \neq r} \delta(j, \bar{X}_i) \right], & \text{если } r = c(i), \\ -\bar{X}_i [\delta(r, \bar{X}_i)], & \text{если } r \neq c(i). \end{cases} \quad (2.27)$$

Можно показать, что для тренировочных примеров \bar{X}_i , в которых функция потерь L_i равна нулю, приведенное выше обновление упрощается до регулирующего обновления каждой гиперплоскости \bar{W}_r :

$$\bar{W}_r \leftarrow \bar{W}_r(1 - \alpha\lambda). \quad (2.28)$$

Регуляризация использует параметр $\lambda > 0$. Регуляризация рассматривается как существенный фактор надлежащего функционирования метода опорных векторов.

2.3.3. Мультиномиальная логистическая регрессия (классификатор Softmax)

Мультиномиальную логистическую регрессию можно рассматривать как мультиклассовое обобщение логистической регрессии в полной аналогии с тем, как SVM Уэстона — Уотсона — это мультиклассовое обобщение бинарной SVM. Мультиномиальная логистическая регрессия использует функцию потерь в виде логарифмической функции правдоподобия, взятой с отрицательным знаком, и поэтому является вероятностной моделью. Как и в случае мультиклассового перцептрона, предполагается, что входом модели является тренировочный набор данных, содержащий пары вида $(\bar{X}_i, c(i))$, где $c(i) \in \{1 \dots k\}$ — индекс класса d -мерной точки данных \bar{X}_i , а в качестве метки точки данных \bar{X}_i , как и в случае предыдущих двух моделей, предсказывается класс, имеющий наибольшее значение $\bar{W}_r \cdot \bar{X}_i$. Однако на этот раз существует дополнительная вероятностная интерпретация величины $\bar{W}_r \cdot \bar{X}_i$ в терминах апостериорной вероятности $P(r | \bar{X}_i)$ того, что точка данных \bar{X}_i получает метку r . Эту оценку можно естественным образом получить с помощью функции активации *Softmax*:

$$P(r | \bar{X}_i) = \frac{\exp(\bar{W}_r \cdot \bar{X}_i)}{\sum_{j=1}^k \exp(\bar{W}_j \cdot \bar{X}_i)}. \quad (2.29)$$

Иными словами, эта модель предсказывает принадлежность к классу в терминах вероятностей. Функция потерь L_i для i -го тренировочного примера определяется с помощью кросс-энтропии, которая представляет собой

отрицательный логарифм вероятности истинного класса. Нейронная архитектура классификатора Softmax приведена на рис. 2.5, в.

Кросс-энтропийные потери могут быть выражены либо в терминах входных признаков, либо в терминах преактивационных значений $v_r = \bar{W}_r \cdot \bar{X}_i$, передаваемых функции *Softmax*:

$$L_i = -\log \left[P(c(i) | \bar{X}_i) \right] = \quad (2.30)$$

$$= -\bar{W}_{c(i)} \cdot \bar{X}_i + \log \left[\sum_{j=1}^k \exp(\bar{W}_j \cdot \bar{X}_i) \right] = \quad (2.31)$$

$$= -v_{c(i)} + \log \left[\sum_{j=1}^k \exp(v_j) \right]. \quad (2.32)$$

Поэтому частную производную L_i по v_r можно вычислить следующим образом:

$$\frac{\partial L_i}{\partial v_r} = \begin{cases} - \left(1 - \frac{\exp(v_r)}{\sum_{j=1}^k \exp(v_j)} \right), & \text{если } r = c(i), \\ \left(\frac{\exp(v_r)}{\sum_{j=1}^k \exp(v_j)} \right), & \text{если } r \neq c(i); \end{cases} \quad (2.33)$$

$$= \begin{cases} -(1 - P(r | \bar{X}_i)), & \text{если } r = c(i), \\ P(r | \bar{X}_i), & \text{если } r \neq c(i). \end{cases} \quad (2.34)$$

Градиент функции потерь i -го тренировочного примера по разделителю r -го класса вычисляется с помощью цепного правила дифференциального исчисления в терминах его преактивационного значения $v_j = \bar{W}_j \cdot \bar{X}_i$:

$$\frac{\partial L_i}{\partial \bar{W}_r} = \sum_j \left(\frac{\partial L_i}{\partial v_j} \right) \left(\frac{\partial v_j}{\partial \bar{W}_r} \right) = \frac{\partial L_i}{\partial v_r} \underbrace{\frac{\partial v_r}{\partial \bar{W}_r}}_{\bar{X}_i}. \quad (2.35)$$

В процессе упрощения приведенного выше выражения мы использовали тот факт, что v_j имеет нулевой градиент по \bar{W}_r для $j \neq r$. Подставив в уравнение 2.35 значение $\frac{\partial L_i}{\partial v_r}$ из уравнения 2.34, мы получаем следующий результат:

$$\frac{\partial L_i}{\partial v_r} = \begin{cases} -\bar{X}_i(1 - P(r | \bar{X}_i)), & \text{если } r = c(i), \\ \bar{X}_i P(r | \bar{X}_i), & \text{если } r \neq c(i). \end{cases} \quad (2.36)$$

Представление выражения для градиента через вероятности (определяемые уравнением 2.29) не только делает его более компактным, но и облегчает понимание взаимосвязи градиента с вероятностью совершения ошибок того или иного типа. Каждый из членов $[1 - P(r | \bar{X}_i)]$ и $P(r | \bar{X}_i)$ выражает вероятность совершения ошибки для экземпляра с меткой $c(i)$ в отношении предсказаний для r -го класса. Учтя фактор регуляризации по аналогии с другими моделями, мы можем записать обновление разделителя для r -го класса в следующем виде:

$$\bar{W}_r \leftarrow \bar{W}_r(1 - \alpha\lambda) + \alpha \begin{cases} \bar{X}_i \cdot (1 - P(r | \bar{X}_i)), & \text{если } r = c(i), \\ -\bar{X}_i \cdot P(r | \bar{X}_i), & \text{если } r \neq c(i). \end{cases} \quad (2.37)$$

где α — скорость обучения, а λ — параметр регуляризации. В отличие от мультиклассового перцептрона и SVM Уэстона — Уоткинса, в которых для каждого тренировочного примера обновляется лишь небольшой поднабор разделителей (или гиперплоскостей, не являющихся разделителями), классификатор Softmax обновляет все k разделителей для каждого тренировочного примера. Это является следствием вероятностного моделирования, в котором корректность классификации определяется сглаженным способом.

2.3.4. Иерархическая функция *Softmax* для случая многих классов

Рассмотрим задачу классификации с предельно большим количеством классов. В этом случае обучение чрезмерно замедляется ввиду большого количества разделителей, которые необходимо обновлять для каждого тренировочного примера. Подобные ситуации могут встречаться в таких задачах, как интеллектуальный анализ текста, где предсказанием является целевое слово. Прогнозирование целевых слов особенно распространено в *нейронных языковых моделях*, которые пытаются предсказывать слова на основании непосредственно предшествующего контекста. В подобных случаях количество классов обычно измеряется числами порядка 10^5 . Использование иерархической функции *Softmax* — это способ улучшения эффективности обучения путем

иерархического разложения задачи классификации на ряд подзадач. Основная идея заключается в переходе от множественной классификации на k классов к иерархическому группированию классов в древовидную бинарную структуру с последующей $\log_2(k)$ -бинарной классификацией в направлении от корня дерева к листу. И хотя иерархическая классификация может приводить к определенному снижению точности, результирующее повышение производительности может быть значительным.

Как работает иерархия классов? Наивный подход предполагает создание случайной иерархии. Однако выбор конкретного способа группирования оказывает влияние на производительность. Как правило, группирование сходных классов обеспечивает повышение производительности. Улучшение качества построения этой иерархии возможно за счет использования специфических особенностей конкретной предметной области. Например, если целью предсказания является слово, то для управления группированием можно воспользоваться иерархией *WordNet* [329]. Далее может потребоваться последующая реорганизация [344], поскольку иерархия *WordNet* — это не в точности бинарное дерево. Другая возможность заключается в том, чтобы использовать кодирование Хаффмана для создания бинарного дерева [325, 327]. За дополнительными ссылками обратитесь к библиографическим примечаниям.

2.4. Использование алгоритма обратного распространения ошибки для улучшения интерпретируемости и выбора признаков

В отношении нейронных сетей часто можно услышать замечания, касающиеся недостаточной интерпретируемости результатов, получаемых с их помощью [97]. Однако оказывается, что, используя механизм обратного распространения ошибки, можно определить признаки, вклад которых в классификацию определенного тестового примера наибольший. Это дает исследователю понимание того, какую роль играет каждый из признаков в классификации. Такой подход имеет также то полезное свойство, что его можно использовать для выбора признаков [406].

Рассмотрим тестовый пример $\bar{X} = (x_1, \dots, x_d)$, для которого множественными выходными метками для нейронной сети являются $o_1 \dots o_k$. Кроме того, пусть среди всех k выходов наиболее выигрышным в отношении предсказания класса будет выход o_m , где $m \in \{1 \dots k\}$. Нашей целью является идентификация признаков, оказывающих наибольшее влияние на классификацию этого тестового примера. В целом нам нужно определить для каждого атрибута x_i чувствительность выхода o_m к x_i . Очевидно, что для классификации конкретного примера более всего важны признаки, характеризующиеся большими *абсолютными*

значениями параметра чувствительности. Для этого нам нужно вычислить абсолютную величину $\frac{\partial o_m}{\partial x_i}$. Признаки с наибольшей по абсолютной величине частной производной оказывают наибольшее влияние на отнесение примера к преобладающему (имеющему наивысшую оценку) классу. Знак производной сообщает о том, приведет ли незначительное увеличение x_i по отношению к его текущему значению к увеличению или уменьшению оценки преобладающего класса. Частная производная предоставляет также возможность составить некоторое впечатление о чувствительности и в отношении других классов, но эта информация менее важна, особенно если количество классов велико. Значение $\frac{\partial o_m}{\partial x_i}$ можно вычислить путем непосредственного применения алгоритма обратного распространения ошибки, но не останавливая его на первом скрытом слое, а продолжая его вплоть до входного слоя.

Этот же подход можно использовать для выбора признаков путем агрегирования абсолютного значения градиента по всем классам и корректно классифицированным тренировочным примерам. Наиболее релевантны признаки с наибольшей агрегированной чувствительностью по всему набору тренировочных данных. Строго говоря, нет нужды агрегировать это значение по всем классам; достаточно использовать лишь преобладающий класс для корректно классифицированных тренировочных примеров. Однако в оригинальной работе [406] предлагается агрегировать это значение по всем классам и всем примерам.

Аналогичные методы интерпретации эффектов влияния различных частей входа используются в компьютерном зрении с применением сверточных нейронных сетей [466]. Обсуждение некоторых из этих методов приведено в разделе 8.5.1. В случае компьютерного зрения визуальные эффекты салиентного анализа этого типа иногда производят сильное впечатление. Например, если речь идет об изображении собаки, то анализ сообщит, какие признаки (т.е. пиксели) играют определяющую роль в том, что данное изображение рассматривается как изображение собаки. Как следствие, мы можем создать салиентное черно-белое изображение, в котором та часть, которая соответствует собаке, выделяется белым цветом на темном фоне (рис. 8.12).

2.5. Факторизация матриц с помощью автокодировщиков

Автокодировщики представляют фундаментальную архитектуру, которая используется для различных типов обучения без учителя, включая факторизацию матриц, анализ главных компонент и снижение размерности. Естественные архитектурные вариации автокодировщика также могут применяться для матричной факторизации неполных данных с целью создания рекомендательных

систем. Кроме того, некоторые недавние методы *конструирования признаков* (feature engineering) в области распознавания естественного языка, такие как *word2vec*, также могут рассматриваться как видоизмененные варианты автокодировщиков, которые выполняют нелинейную факторизацию терм-контекстных матриц. Нелинейность обеспечивается функцией активации в выходном слое, что при традиционной матричной факторизации обычно не используется. Поэтому одной из наших задач будет демонстрация того, как внесение незначительных изменений в базовые строительные блоки нейронных сетей позволяет реализовать сложные вариации заданного семейства методов. Это особенно удобно для аналитика, цель которого — лишь провести эксперименты с небольшими вариациями архитектуры для тестирования различных типов моделей. В традиционном машинном обучении это потребовало бы приложения больших усилий, поскольку оно не позволяет воспользоваться преимуществами таких абстракций обучения, как алгоритм обратного распространения ошибки. Прежде всего мы начнем с простой имитации традиционного метода факторизации матриц с помощью мелкой нейронной архитектуры. Затем мы обсудим, каким образом эта простая модель может быть обобщена на нелинейные методы снижения размерности путем добавления слоев и/или нелинейных активационных функций.

Таким образом, цель данного раздела заключается в том, чтобы продемонстрировать следующие два момента.

1. Классические методы снижения размерности, такие как сингулярное разложение и анализ главных компонент, представляют собой специальные случаи нейронных архитектур.
2. Путем добавления в эту базовую архитектуру элементов различного уровня сложности можно генерировать сложные нелинейные представления данных. Несмотря на то что нелинейные представления доступны и в машинном обучении, нейронные архитектуры предоставляют беспрецедентные возможности управления свойствами этих представлений путем внесения различных типов архитектурных изменений (позволяя алгоритму обратного распространения ошибки самостоятельно позаботиться о необходимых изменениях базовых алгоритмов обучения).

Мы также обсудим ряд задач, таких как рекомендательные системы и обнаружение выбросов (аномалий).

2.5.1. Автокодировщик: базовые принципы

Базовая идея автокодировщика заключается в использовании выходного слоя той же размерности, что и входы. Суть этой идеи — попытаться в точности реконструировать каждую размерность путем передачи ее через сеть. Автокоди-

ровщик *реплицирует* данные из входа в выход, поэтому его иногда называют *репликаторной нейронной сетью*. И хотя, на первый взгляд, реконструирование данных может показаться не более чем обычным копированием данных из одного слоя в другой в прямом направлении, это невозможно в случае ограниченного количества элементов в промежуточных слоях. Другими словами, обычно каждый промежуточный слой содержит меньше элементов, чем входной (или выходной). Как следствие, эти элементы хранят редуцированное представление данных, в результате чего последний слой не может полностью реконструировать данные. Поэтому данному типу реконструкции изначально присущи потери. Функция потерь такой нейронной сети использует сумму квадратов разностей между входом и выходом для того, чтобы принудительно довести выход до уровня, максимально близкого к уровню входа. Это общее описание автокодировщика проиллюстрировано на рис. 2.6, а, на котором приведена архитектура с тремя ограниченными слоями. Следует отметить, что представление данных в самом внутреннем слое будет иерархически связано с их представлениями в двух других скрытых слоях. Поэтому автокодировщик способен выполнять иерархическое уменьшение размерности данных.

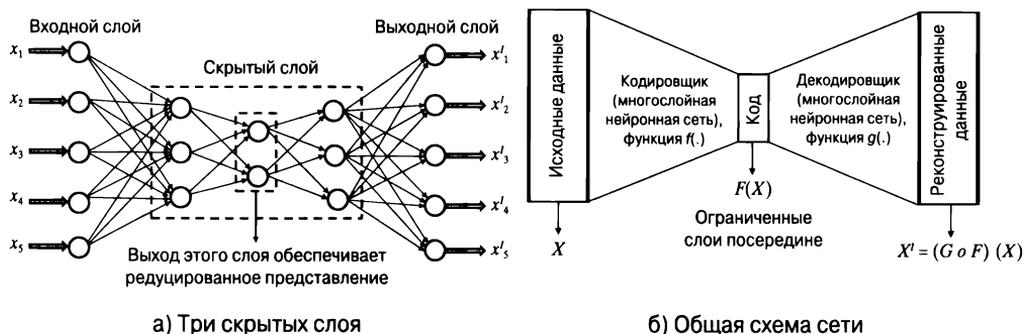


Рис. 2.6. Базовое схематическое представление автокодировщика

Нередко (но не всегда) часть архитектуры M -слоеного автокодировщика между входом и выходом обладает симметрией в том смысле, что количество элементов в k -м слое совпадает с их количеством в $(M - k + 1)$ -м слое. Кроме того, нередко значение M — нечетное, в результате чего $(M + 1)/2$ -й слой часто является наиболее ограниченным (содержащим наименьшее количество элементов). В данном случае мы считаем (невыхислительный) входной слой первым слоем, и поэтому минимальное количество слоев в автокодировщике — 3, что соответствует входному, ограниченному и выходному слоям. Как будет показано далее, эта простейшая форма автокодировщика используется в традиционном машинном обучении для *сингулярного разложения матриц* (singular value decomposition — SVD). Такая симметрия архитектуры часто распространяется и на веса: во многих архитектурах веса исходящих соединений k -го слоя часто

привязываются к входящим весам $(M - k)$ -го слоя. Ради простоты мы не будем прибегать к этому предположению на данном этапе. Кроме того, симметрия никогда не бывает абсолютной из-за влияния нелинейных функций активации. Если, например, в выходном слое используется нелинейная функция активации, то вам никак не удастся зеркально отобразить этот факт во входном слое.

Редуцированное представление данных иногда называют *кодом*, а количество элементов в этом слое — *размерностью* данного представления. Начальную часть нейронной архитектуры, предшествующую этому “бутылочному горлышку”, называют *кодировщиком* (поскольку она создает редуцированный код), а завершающую часть архитектуры — *декодировщиком* (поскольку она реконструирует данные на основе кода). Структура автокодировщика представлена в общем виде на рис. 2.6, б.

2.5.1.1. Автокодировщик с одним скрытым слоем

Ниже описана простейшая версия автокодировщика, которая используется в целях матричной факторизации. В этом автокодировщике имеется всего один скрытый слой, состоящий из $k \ll d$ элементов и располагающийся между входным и выходным слоями, содержащими каждый по d элементов. В целях нашего обсуждения предположим, что имеется матрица D размера $n \times d$, которую мы хотим факторизовать с помощью матрицы U размером $n \times k$ и матрицы V размера $d \times k$:

$$D \approx UV^T, \quad (2.38)$$

где k — ранг факторизации. Матрица U содержит редуцированное представление данных, а матрица V — базовые векторы. Факторизация матриц — один из наиболее широко исследованных методов в области обучения с учителем, который применяется для снижения размерности данных, кластеризации и предсказательного моделирования в рекомендательных системах.

В традиционном машинном обучении эта задача решается посредством минимизации нормы Фробениуса остаточной матрицы $(D - UV^T)$. Квадрат нормы Фробениуса матрицы — это сумма квадратов матричных элементов. Поэтому целевая функция задачи оптимизации может быть записана в следующем виде:

$$\text{Минимизируемая функция } J = \|D - UV^T\|_F^2,$$

где $\|\cdot\|_F$ — норма Фробениуса. Матрицы параметров U и V должны быть обучены тому, чтобы минимизировать ошибку. Такая целевая функция имеет бесконечное количество минимумов, одному из которых соответствуют взаимно ортогональные базисные векторы. Это частное решение называют *усеченным сингулярным разложением* (truncated singular value decomposition). Несмотря на относительную простоту шагов градиентного спуска [6] в этой оптимизацион-

ной задаче (когда задумываться о нейронных сетях вообще не приходится), в данном случае мы хотим справиться с ней именно с помощью нейронной архитектуры. Выполнение этого упражнения позволит продемонстрировать, что SVD — это специальный случай архитектуры автокодировщика, который послужит нам фундаментом для понимания преимуществ, обеспечиваемых более сложными автокодировщиками.

Пример нейронной архитектуры SVD, в котором скрытый слой содержит k элементов, приведен на рис. 2.7. Строки матрицы D — это вход автокодировщика, тогда как k -мерные строки матрицы U — это активации скрытого слоя. Матрицей весов декодировщика, имеющей размер $k \times d$, является матрица V^T . Как уже обсуждалось во введении в многослойные нейронные сети в главе 1, вектор значений в определенном слое сети можно получить, умножив вектор значений в предыдущем слое на матрицу весов соединений, связывающих эти два слоя (с использованием линейной активации). Поскольку активации скрытого слоя содержатся в матрице U , а веса декодировщика — в матрице V^T , то реконструированный выход содержится в строках матрицы UV^T . Автокодировщик минимизирует сумму квадратов разностей между входом и выходом, что эквивалентно минимизации $\|D - UV^T\|^2$. Поэтому та же задача решается методом сингулярного разложения.

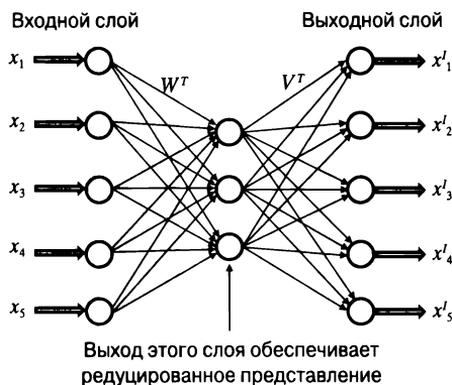


Рис. 2.7. Базовый автокодировщик с одним слоем

Обратите внимание на то, что этот подход можно использовать для получения редуцированного представления неизвестных образцов данных, *не входящих в учебную выборку* и, таким образом, не включенных в исходную матрицу D . Для этого нужно просто подать строки с новыми данными в качестве входа, и тогда активации скрытого слоя обеспечат получение желаемого представления пониженной размерности. Понижение размерности неизвестных образцов данных оказывается особенно полезным в случае предназначенных для этого нелинейных методов, поскольку традиционным методам машинного обучения труднее свертывать такие образцы.

Веса кодировщика

Как показано на рис. 2.7, веса кодировщика содержатся в матрице размера $k \times d$, обозначенной как W . Каким образом эта матрица связана с матрицами U и V ? Заметьте, что автокодировщик создает реконструированное представление $DW^T V^T$ исходной матрицы данных. Поэтому он пытается оптимизировать задачу минимизации $\|DW^T V^T - D\|^2$. Наилучшее решение получается тогда, когда матрица W является псевдообращением матрицы V , которое определяется следующей формулой:

$$W = (V^T V)^{-1} V^T. \quad (2.39)$$

Можно легко доказать, что этот результат справедлив по крайней мере в невырожденных случаях, когда строки матрицы D охватывают полный ранг d измерений (см. упражнение 14). Разумеется, окончательное решение, найденное тренировочным алгоритмом автокодировщика, может не соответствовать этому условию, поскольку оно может не являться точным решением или матрица D может иметь меньший ранг.

Из самого определения псевдообратной матрицы следует, что $WV = I$ и $V^T W^T = I$, где I — единичная матрица размера $k \times k$. Умножая уравнение 2.38 справа на матрицу W^T , получаем следующее уравнение:

$$DW^T \approx U \underbrace{(V^T W^T)}_I = U. \quad (2.40)$$

Иными словами, умножение каждой строки матрицы D на матрицу W^T размера $d \times k$ дает редуцированное представление этого примера, которым является соответствующая строка матрицы U . Кроме того, повторное умножение этой строки U на матрицу V^T дает реконструированную версию исходной матрицы D .

Обратите внимание на то, что существует много других оптимумов для матриц W и V , но для того, чтобы была возможна указанная реконструкция (т.е. минимизация функции потерь), обученная матрица W всегда будет представлять собой (приближенно) матрицу, псевдообратную к матрице V , а на столбцы V всегда будет натянуто³ некоторое k -мерное подпространство, определяемое задачей SVD-оптимизации.

³ Это подпространство определяется первыми k сингулярными векторами метода сингулярного разложения. Однако задача оптимизации не налагает ограничений ортогональности, и поэтому для представления данного подпространства столбец V может использовать другую, неортогональную базисную систему.

2.5.1.2. Связь с сингулярным разложением

Архитектура однослойного автокодировщика тесно связана с *методом сингулярного разложения* (singular value decomposition — SVD). Этот метод находит факторизацию UV^T , в которой строки V ортогональны. Функция потерь такой нейронной сети совпадает с функцией потерь метода сингулярного разложения, а решение V , в котором столбцы V ортонормальны, всегда будет одним из *возможных* оптимумов, получаемых в результате обучения нейронной сети. В то же время, поскольку эта функция потерь допускает альтернативные оптимумы, возможно нахождение оптимального решения, в котором столбцы V не обязательно взаимно ортогональны или масштабированы до единичной нормы. SVD определяется ортонормальной базисной системой. Тем не менее подпространство, натянутое на k столбцов V , будет совпадать с пространством, натянутым на первые k базисных векторов SVD. *Анализ главных компонент* (principal component analysis — PCA) идентичен методу сингулярного разложения, за исключением того, что он применяется к матрице D , центрированной на среднем значении. Поэтому данный подход также может быть использован для нахождения подпространства, натянутого на первые k главных компонент. Однако каждый столбец D должен центрироваться на среднем путем заблаговременного вычитания среднего значения из него. Можно получить ортонормальную базисную систему, еще более близкую к SVD и PCA, за счет разделения некоторых весов кодировщиком и декодировщиком. Данный подход обсуждается в следующем разделе.

2.5.1.3. Использование общих весов в кодировщике и декодировщике

В соответствии с приведенным выше обсуждением существует множество альтернативных решений для матриц W и V , где W — матрица, псевдообратная к матрице V , что позволяет дополнительно уменьшить количество параметров без значительной⁴ потери точности реконструкции входных данных. При построении автокодировщиков обычной практикой является совместное использование весов кодировщиком и декодировщиком. Такой подход также называют *связыванием весов*. В частности, автокодировщикам присуща симметричная структура, в которой веса кодировщика и декодировщика принудительно устанавливаются равными в симметрично расположенных слоях. В случае мелкой

⁴ В некоторых особых случаях, таких как обсуждаемый нами однослойный автокодировщик, потери точности реконструкции отсутствуют даже для тренировочных данных. В других случаях потеря точности наблюдается только для тренировочных данных, в то время как неизвестные данные лучше реконструируются вследствие эффектов регуляризации, обусловленных уменьшением количества параметров.

архитектуры кодировщик и декодировщик разделяют веса с использованием следующего соотношения:

$$W = V^T. \quad (2.41)$$

Эта архитектура (рис. 2.8) совпадает с архитектурой, приведенной на рис. 2.7, за исключением связанных весов. Иными словами, сначала матрица весов V размера $d \times k$ используется для преобразования d -мерной точки данных X в k -мерное представление. Затем матрица весов V^T используется для реконструкции первоначального представления данных.

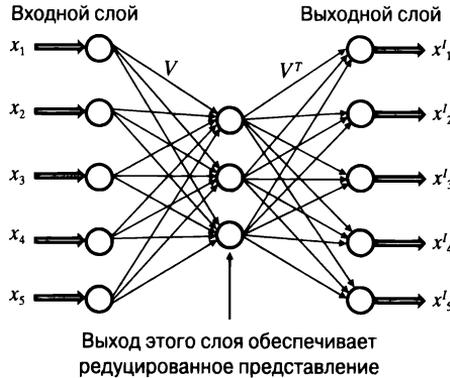


Рис. 2.8. Базовый автокодировщик с одним слоем; обратите внимание на использование в данном случае (в отличие от автокодировщика, представленного на рис. 2.7) связанных весов

Связывание весов по сути означает, что V^T — матрица, псевдообратная к матрице V (см. упражнение 14). Иными словами, мы имеем $V^T V = I$, поэтому столбцы V взаимно ортогональны. В результате благодаря связыванию весов теперь возможна точная имитация метода SVD, в котором различные базисные векторы должны быть взаимно ортогональны.

В этом конкретном примере архитектуры с одним скрытым слоем связывание весов выполняется только для пары весовых матриц. В общем случае мы будем иметь дело с нечетным количеством слоев и четным количеством весовых матриц. общепринятая практика заключается в том, чтобы согласовывать матрицы весов симметричным образом относительно середины. В этом случае симметрично расположенные слои должны содержать одинаковое количество элементов. Даже в тех случаях, когда веса не разделяются частями архитектуры, соответствующими кодировщику и декодировщику, такая мера позволяет вдвое уменьшить количество параметров. В этом есть свои преимущества и с точки зрения ослабления эффектов переобучения. Иными словами, данный подход обеспечивает улучшенную реконструкцию неизвестных данных. Еще одним

преимуществом связывания матриц весов кодировщика и декодировщика является то, что это приводит к автоматической нормализации столбцов V к одинаковым значениям. Например, если мы не связываем матрицы весов кодировщика и декодировщика, то разные столбцы V могут иметь разные нормы. По крайней мере, в случае линейных активаций связывание матриц весов заставляет все столбцы V иметь одинаковые нормы. Это полезно и с точки зрения улучшения нормализации встроенного представления. В случае использования нелинейных активаций в вычислительных слоях свойства нормализации и ортогональности уже не соблюдаются строго. Но даже в этом случае связывание весов обеспечивает значительные преимущества в плане получения улучшенных решений.

Разделение весов требует внесения некоторых изменений в алгоритм обратного распространения в процессе тренировки. Однако сделать это вовсе не трудно. Для этого нужно всего лишь не учитывать тот факт, что веса связаны, при вычислении градиентов в процессе обычного обратного распространения ошибки. После этого градиенты по различным экземплярам одного и того же веса суммируются для вычисления шагов градиентного спуска. Логика обработки разделяемых весов таким способом обсуждается в разделе 3.2.9.

2.5.1.4. Другие методы факторизации матриц

Простой трехслойный автокодировщик можно видоизменить таким образом, чтобы имитировать другие типы методов матричной факторизации, такие как *неотрицательное матричное разложение* (non-negative matrix factorization), *вероятностный латентно-семантический анализ* (probabilistic latent semantic analysis) и *логистическое матричное разложение* (logistic matrix factorization). Различные методы логистического матричного разложения обсуждаются в следующем разделе, разделе 2.6.3 и упражнении 8. Методы неотрицательного матричного разложения и вероятностного латентно-семантического анализа обсуждаются в упражнениях 9 и 10. Будет поучительно исследовать, как соотносятся между собой различные варианты данных методов, поскольку это даст возможность продемонстрировать, как изменение простых нейронных архитектур позволяет получать результаты с весьма различающимися свойствами.

2.5.2. Нелинейные активации

До сих пор наше обсуждение фокусировалось на имитации методов сингулярного разложения с использованием нейронной архитектуры. Очевидно, что такой подход нельзя считать многообещающим, поскольку для сингулярного разложения уже существует множество готовых инструментов. Однако в реальной мощи автокодировщиков вы сможете убедиться тогда, когда начнете использовать нелинейные активации и многослойные нейронные сети. Рассмотрим, к примеру, ситуацию, когда матрица D — бинарная. В этом случае

применима та же нейронная архитектура, что и представленная на рис. 2.7, но теперь для предсказания выхода можно использовать в выходном слое сигмоиду. Этот сигмоидный слой комбинируется с отрицательными логарифмическими потерями. Поэтому для бинарной матрицы $B = [b_{ij}]$ модель предполагает следующее:

$$B \sim \text{сигмоида}(UV^T). \quad (2.42)$$

Здесь сигмоидная функция активации применяется к каждому элементу по отдельности. Обратите внимание на использование знака \sim вместо \approx в приведенном выше выражении для указания того, что бинарная матрица B получена путем извлечения случайных значений из распределений Бернулли с соответствующими параметрами, содержащимися в аргументе сигмоиды. Можно показать, что результирующая факторизация эквивалентна *логистическому матричному разложению*. Суть идеи состоит в том, что (i, j) -й элемент UV^T — это параметр распределения Бернулли, а элементы b_{ij} генерируются из распределения Бернулли с этими параметрами. Поэтому матрицы U и V обучаются с использованием логарифмической функции потерь этой *генеративной* модели. Функция логарифмических потерь неявно пытается найти такие матрицы параметров U и V , чтобы максимизировать вероятность матрицы B , генерируемой с помощью этих параметров.

Логистическое матричное разложение было предложено лишь недавно [224] в качестве улучшенного метода матричной факторизации для бинарных данных, который может быть полезным для рекомендательных систем с *неявными пользовательскими рейтинговыми оценками* (implicit feedback ratings). Под этим термином подразумеваются бинарные действия пользователей, такие как покупка или отказ от покупки определенных товаров. Может показаться, что методология получения решений, изложенная в недавней работе по логистическому матричному разложению [224], значительно отличается от SVD и не связана с подходом, основанным на нейронных сетях. Однако для того, кто работает с нейронными сетями, переход от SVD-модели к логистическому матричному разложению представляет лишь незначительное изменение, затрагивающее только последний слой нейронной сети. Именно модульная природа нейронных сетей делает их столь привлекательными для инженеров и поощряет проведение экспериментов любого рода. В действительности, как показывает тщательное изучение этого вопроса, одним из вариантов популярного подхода к конструированию признаков текста на основе модели *word2vec* [325, 327] является метод логистического матричного разложения. Интересно отметить, что модель *word2vec* была предложена раньше, чем метод логистического матричного разложения в традиционном машинном обучении [224], хотя эквивалентность обоих методов и не была продемонстрирована в оригинальной работе. Впервые это было сделано в [6], и доказательство этого результата также будет приведено

далее. Что касается многослойного автокодировщика, то его точного аналога в традиционном машинном обучении не существует. Все это указывает на то, что понять, как работают сложные алгоритмы машинного обучения, зачастую легче, работая с применением модульного подхода к конструированию многослойных нейронных сетей. Заметьте, что данный подход можно использовать даже для факторизации матриц с элементами в виде вещественных значений из интервала $[0, 1]$, при условии, что функция логистических потерь соответствующим образом видоизменена для обработки дробных значений (см. упражнение 8). Логистическое матричное разложение — это тип *ядерного матричного разложения* (kernel matrix factorization).

Вместо того чтобы применять нелинейные активации в выходном слое (или в дополнение к этому), их также можно использовать в скрытом слое. Налагаемая ограниченность в виде неотрицательности значений с помощью нелинейности в скрытом слое, можно имитировать неотрицательное матричное разложение (см. упражнения 9 и 10). Рассмотрим автокодировщик с единственным скрытым слоем, содержащим сигмоидные элементы, и линейным выходным слоем. Обозначим матрицы весов соединений, связывающих входной слой со скрытым слоем и скрытый слой с выходным слоем, как W^T и V^T соответственно. В этом случае матрица W уже не будет псевдообратной к матрице V ввиду нелинейности активации в скрытом слое.

Если U — выход скрытого слоя, в котором используется нелинейная функция активации $\Phi(\cdot)$, то

$$U = \Phi(DW^T). \quad (2.43)$$

Если выход линейный, то общее разложение по-прежнему имеет следующий вид:

$$D \approx UV^T. \quad (2.44)$$

Обозначим через $U' = DW^T$ линейную проекцию исходной матрицы D . Тогда факторизация может быть записана в следующем виде:

$$D \approx \Phi(U')V^T, \quad (2.45)$$

где U — линейная проекция D . В данном случае мы имеем дело с другим типом нелинейной матричной факторизации [521, 558]. Несмотря на то что рассматриваемая нами конкретная форма нелинейности (т.е. сигмоида) может показаться слишком простой по сравнению с теми, которые типичны для ядерных методов, в действительности для обучения более сложным формам снижения размерности данных используются сети с несколькими скрытыми слоями. Кроме того, нелинейность в скрытых слоях может сочетаться с нелинейностью в выходном слое. Методы нелинейного снижения размерности позволяют отображать данные на пространства гораздо меньшей размерности (с хорошими

характеристиками восстановления), чем это возможно в случае использования таких методов, как РСА. Пример набора данных, распределенного вдоль нелинейной спирали, приведен на рис. 2.9, а. Снижение размерности этих данных с помощью метода РСА (не приводящее к значительному увеличению ошибок их восстановления) невозможно. В то же время использование нелинейных методов снижения размерности позволяет сплюснуть эту нелинейную спираль до двухмерного представления (рис. 2.9, б).

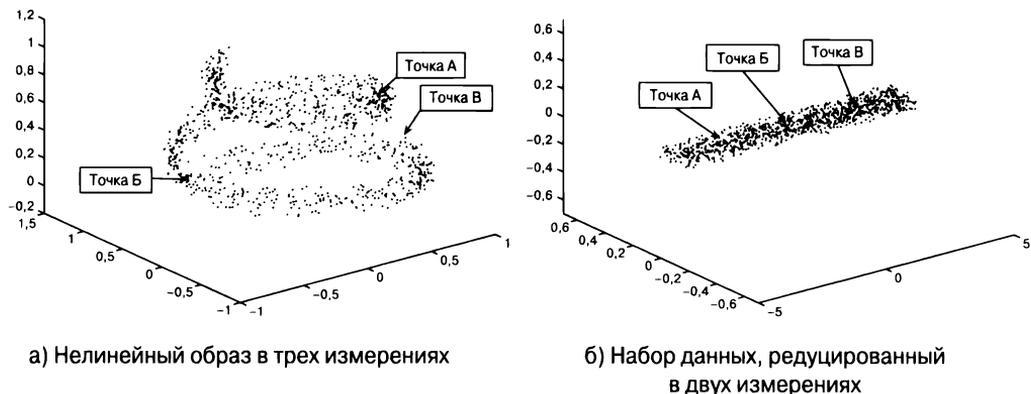


Рис. 2.9. Эффект нелинейного снижения размерности; данный рисунок приведен исключительно в иллюстративных целях

Нелинейные методы снижения размерности часто требуют использования более глубоких сетей, что обусловлено повышенной сложностью преобразований, осуществляемых с помощью комбинации нелинейных элементов. Преимущества глубоких сетей обсуждаются в следующем разделе.

2.5.3. Глубокие автокодировщики

Реальная мощь автокодировщиков в нейронных сетях проявляется лишь в случае использования их глубоких вариантов. Пример автокодировщика с тремя скрытыми слоями приведен на рис. 2.10. Увеличение количества промежуточных слоев нейронной сети позволяет дополнительно повысить ее репрезентативную мощь. Необходимо отметить, что для увеличения репрезентативной мощи сети существенное значение имеет использование нелинейных активаций в некоторых слоях глубокого автокодировщика. В соответствии с леммой 1.5.1, увеличение количества слоев в многослойной сети при условии, что используются только линейные активации, не дает никакого выигрыша в мощи. И хотя этот результат приводился в главе 1 в связи с задачей классификации, он справедлив для более широкого круга многослойных нейронных сетей (включая автокодировщики).

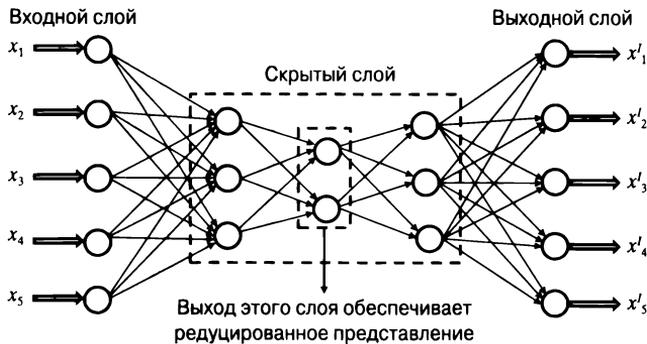


Рис. 2.10. Пример автокодировщика с тремя скрытыми слоями; сочетание нелинейных активаций с несколькими скрытыми слоями увеличивает репрезентативную мощность сети

Глубокие сети со многими слоями обладают беспрецедентной репрезентативной мощностью. Последовательность многих слоев такой сети обеспечивает *иерархическое* редуцирование представления данных. В случае некоторых типов данных, таких как изображения, этот иерархический процесс представляется особенно естественным. Обратите внимание на то, что точного аналога модели этого типа в традиционном машинном обучении не существует, и механизм обратного распространения ошибки избавляет нас от трудностей, связанных с вычислением шагов градиентного спуска. Нелинейные методы снижения размерности способны транслировать множество произвольных форм в редуцированное представление. И хотя в машинном обучении также известно несколько методов нелинейного снижения размерности, нейронные сети имеют ряд преимуществ по сравнению с этими методами.

1. Многие нелинейные методы снижения размерности плохо справляются с редуцированием представлений новых точек данных, если эти точки не входили в тренировочный набор. С другой стороны, можно сравнительно легко вычислить редуцированное представление таких точек, пропуская их через сеть.
2. Нейронные сети обеспечивают большие возможности и гибкость нелинейного снижения размерности данных за счет варьирования количества и типов слоев, используемых на промежуточных стадиях. Кроме того, путем выбора конкретных типов активационных функций для использования в различных слоях можно приспособливать характер редукиции данных к их свойствам. Например, в случае бинарного набора данных имеет смысл использовать логистический выходной слой с логарифмической функцией потерь.

Использование этого подхода позволяет добиться предельно компактного снижения размерности. Например, в [198] показано, как преобразовать 784-мерное

пиксельное представление изображения в редуцированное 6-мерное представление, используя глубокие автокодировщики. Значительное снижение размерности всегда достигается за счет использования нелинейных элементов, которые неявно отображают искривленные многообразия на линейные гиперплоскости. В подобных случаях достижение исключительно высокой степени снижения размерности становится возможным благодаря тому, что обеспечить прохождение через большое количество точек с помощью криволинейной поверхности гораздо легче, чем с помощью линейной. Это свойство нелинейных автокодировщиков часто используют для двумерной визуализации данных путем создания глубокого автокодировщика, в котором наиболее компактный скрытый слой имеет всего два измерения. Затем эти два измерения отображаются на плоскость для визуализации точек. Во многих случаях этот способ обеспечивает проявление классовой структуры данных в терминах отчетливо разделенных кластеров.

На рис. 2.11 приведен иллюстративный пример типичного поведения реального распределения данных, в котором двумерное отображение, созданное автокодировщиком, отчетливо разделяется на два различных класса. С другой стороны, отображение, созданное PCA, не обеспечивает удовлетворительного разделения этих классов. Причины такого поведения проясняет рис. 2.9, на котором было представлено нелинейное спиральное распределение данных, отображенное на линейную гиперплоскость. Во многих случаях распределения данных могут содержать переплетающиеся спирали (или другие формы), принадлежащие к разным классам. Линейные методы снижения размерности не позволяют добиться отчетливого разделения классов, поскольку нелинейно переплетающиеся формы не являются линейно разделимыми. С другой стороны, глубокие автокодировщики с нелинейностью обладают гораздо более мощными возможностями и способны распутывать такие формы. В некоторых случаях глубокие автокодировщики могут использоваться в качестве альтернативы другим робастным методам визуализации, таким как *стохастическое вложение соседей с t -распределением* (*t-distributed stochastic neighbor embedding* — t-SNE) [305]. Несмотря на то что метод t-SNE нередко позволяет повысить производительность⁵ визуализации (поскольку он ориентирован конкретно на визуализацию данных, а не на снижение их размерности), автокодировщики имеют по

⁵ Метод t-SNE работает по принципу, суть которого заключается в том, что для вложения низкой размерности невозможно сохранить один и тот же уровень точности для всех случаев попарного сходства и различия признаков. Поэтому, в отличие от методов снижения размерности и автокодировщиков, которые пытаются обеспечить надежное восстановление данных, его функция потерь асимметрична в отношении обработки сходств и различий признаков. Асимметричные функции потерь такого типа оказываются особенно полезными для выделения различных многообразий в процессе визуализации. Поэтому при визуализации метод t-SNE может работать лучше, чем автокодировщики.

сравнению с ним то преимущество, что они легче обобщаются на неизвестные данные. При получении новых точек данных их можно просто пропускать через ту часть автокодировщика, которая ответственна за кодирование данных, для добавления к текущему набору визуализируемых точек. Конкретный пример визуализации многомерной коллекции документов с помощью автокодировщика приведен в [198].

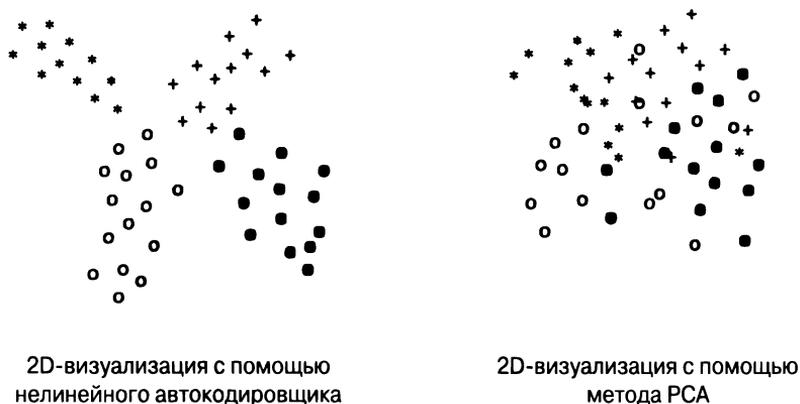


Рис. 2.11. Типичные различия между вложениями, созданными с помощью нелинейных автокодировщиков и по методу анализа главных компонент (PCA). Нелинейные и глубокие автокодировщики часто способны разделять переплетенные классовые структуры, содержащиеся в базовых данных, что невозможно в условиях ограничений линейных преобразований наподобие PCA. Это происходит по той причине, что отдельные классы часто заполняются в исходном пространстве на таких искривленных многообразиях, которые, если при этом не искривлять само пространство, выглядят смешанными при рассмотрении любого двухмерного сечения данных. Данный рисунок приведен исключительно в иллюстративных целях и не представляет никакого конкретного распределения данных

Однако при этом очень важно не переборщить, чтобы не создать представления, которые окажутся бесполезными. Например, можно сжать многомерную точку данных в одно измерение, обеспечивающее удовлетворительное восстановление тренировочных данных, но приводящее к большим ошибкам для тестовых данных. Иными словами, нейронная сеть может суметь найти способ запомнить набор данных, но при этом она окажется неспособной создавать представления сниженной размерности для точек, которые еще не встречались. Поэтому даже в таких задачах обучения без учителя, как снижение размерности, важно придержать некоторые точки в качестве *валидационного набора*. Точки, входящие в валидационный набор, не используются в процессе тренировки. После этого следует количественно оценить разницу в ошибках восстановления данных между тренировочным и валидационным наборами. Большое различие

между этими ошибками является индикатором эффектов переобучения. Еще одна проблема заключается в том, что глубокие сети труднее поддаются тренировке, и поэтому важное значение приобретают такие приемы, как *предварительное обучение* (pretraining). Эти приемы обсуждаются в главах 3 и 4.

2.5.4. Обнаружение выбросов

Снижение размерности тесно связано с обнаружением *выбросов*, поскольку точки выброса трудно кодировать и декодировать, не потеряв при этом существенную информацию. Как хорошо известно, если матрица D факторизуется в виде $D \approx D' = UV^T$, то низкоранговая матрица D' — это обесшумленное представление данных. В конце концов, сжатое представление U улавливает лишь регулярности в данных и неспособно улавливать аномальные вариации, проявляющиеся в отдельных точках. Как следствие, при восстановлении данных до состояния D' все эти необычные точки упускаются.

Абсолютные значения элементов матрицы $(D - D')$ представляют оценки выбросов соответствующих матричных элементов. Поэтому данный подход можно использовать для нахождения элементов, представляющих выбросы, или же, суммируя квадраты оценок элементов в каждой строке матрицы D , для нахождения оценки выброса данной строки. Тем самым можно идентифицировать точки выбросов. Кроме того, суммирование квадратов таких оценок в каждом столбце матрицы D позволяет обнаруживать выбросы признаков. Это полезно для приложений, связанных с отбором признаков при кластеризации, когда признак с большой оценкой выброса можно удалить, поскольку он привносит шум в кластеризацию. Несмотря на то что в приведенном выше описании мы опирались на матричную факторизацию, оно применимо к автокодировщикам любого типа. В действительности построение *шумоподавляющих (обесшумливающих) автокодировщиков* — это самостоятельное и быстро развивающееся направление. Для получения дополнительной информации воспользуйтесь ссылками, приведенными в разделе “Библиографическая справка”.

2.5.5. Когда скрытый слой шире входного

До сих пор мы рассматривала случаи, когда скрытый слой содержит меньше элементов, чем входной. В этом есть смысл, если вас интересует сжатое представление данных. Ограниченный в размерах скрытый слой навязывает снижение размерности, и функция потерь конструируется таким образом, чтобы избежать утери информации. Такие представления называются *неполными* и соответствуют традиционным случаям использования автокодировщиков.

А что если количество элементов в скрытом слое превышает размерность входа? Эта ситуация соответствует случаю *сверхполных* представлений. Увеличение количества скрытых элементов сверх количества элементов на входе

предоставляет скрытому слою возможность обучиться тождественной функции (с нулевыми потерями). Очевидно, что простое копирование входа в слои не дало бы ничего особенно полезного. Однако на практике (в процессе обучения весов) этого не происходит, особенно если в скрытом слое применяется регуляризация и налагаются *ограничения разреженности* (sparsity constraints). Даже если эти ограничения не налагаются, но для обучения используется стохастический градиентный спуск, то обусловленная им вероятностная регуляризация является достаточной гарантией того, что скрытое представление всегда будет перемешивать вход, прежде чем реконструировать его на выходе. Это происходит потому, что стохастический градиентный спуск добавляет шум в процесс обучения, и в результате невозможно обучить веса так, чтобы они просто копировали вход на выход, действуя как тождественная функция в каждом слое. Кроме того, вследствие некоторых особенностей процесса тренировки нейронная сеть почти никогда не использует свои возможности моделирования в полную силу, что приводит к возникновению зависимостей между весами [94]. Скорее будет создано сверхполное представление, хотя оно может и не обладать свойством разреженности (которое требует явной поддержки). Способы поддержки разреженности обсуждаются в следующем разделе.

2.5.5.1. Обучение разреженным признакам

В случае наложения явных ограничений разреженности результирующий автокодировщик называют *разреженным*. Разреженным представлением d -мерной точки является k -мерная точка, где $k \gg d$, при этом большинство значений в разреженном представлении являются нулями. Обучение разреженным представлениям имеет огромное значение для многих задач, таких как обработка изображений, когда обучаемые признаки нередко легче интерпретировать с учетом специфики конкретной задачи. Кроме того, точки с переменной степенью информативности будут естественным образом иметь переменное количество признаков с ненулевыми значениями. Свойства этого типа присущи некоторым входным представлениям, таким как документы: более информативные документы будут иметь большее количество ненулевых признаков (частоты слов) в многомерном формате представления. Но если имеющийся вход не является разреженным, то часто целесообразно создать разреженное преобразованное представление, обеспечивающее такую гибкость. Разреженные представления позволяют использовать определенные типы алгоритмов, производительность которых значительно зависит от степени разреженности данных. Существует множество способов принудительного наложения ограничений на скрытый слой для создания разреженности. Один из таких подходов заключается в добавлении смещений в скрытый слой, чтобы стимулировать создание многих элементов с нулевыми значениями. Несколько примеров приведено ниже.

1. Наложение L_1 -штрафов на активации в скрытом слое поддерживает разреженную активацию. Применение L_1 -штрафов для создания разреженных решений (в терминах весов или в терминах скрытых элементов) обсуждается в разделах 4.4.2 и 4.4.4. В этом случае механизм обратного распространения ошибки должен распространять в обратном направлении также градиент этого штрафа. Как это ни удивительно, подобная естественная альтернатива используется лишь в редких случаях.
2. Альтернативный вариант состоит в том, чтобы разрешить ненулевые значения лишь для первых r активаций в скрытом слое, где $r \leq k$. В этом случае механизм обратного распространения ошибки реализуется лишь через активированные элементы. Такой подход называют *r-разреженным автокодировщиком* (*r-sparse autoencoder*) [309].
3. Также возможен вариант автокодировщика, работающего по принципу “победитель получает все” [310], в котором из всего набора тренировочных данных активации разрешены лишь для некоторой доли f скрытых элементов. В этом случае первые r активаций вычисляются на всем наборе тренировочных примеров, тогда как в предыдущем случае первые r активаций вычисляются в скрытом слое для одного тренировочного примера. Поэтому специфические для узлов пороги должны оцениваться с использованием статистики мини-пакетов. Алгоритм обратного распространения ошибки должен распространять градиент лишь через активированные элементы.

Обратите внимание на то, что реализация состязательных механизмов почти эквивалентна активациям ReLU с адаптивными порогами. Для получения более подробной информации об этих механизмах воспользуйтесь ссылками, приведенными в разделе “Библиографическая справка”.

2.5.6. Другие применения

Автокодировщики — рабочая лошадка обучения без учителя в области нейронных сетей. Они находят множество применений, о чем еще пойдет речь далее. Завершив тренировку автокодировщика, вовсе не обязательно использовать как кодировщик, так и декодировщик. Например, если перед вами стоит задача снижения размерности, можно использовать только кодировщик для создания редуцированного представления данных. В этом случае восстановление данных с помощью декодировщика может вообще не потребоваться.

Несмотря на то что автокодировщик (как и почти любой другой метод снижения размерности) естественным образом устраняет шум, существуют способы улучшения его возможностей в отношении удаления шумов определенного типа. В случае *шумоподавляющего (обесшумливающего) автокодировщика*

(de-noising autoencoder) применяется особый тип тренировки. Во-первых, прежде чем пропускать данные через сеть, к ним добавляют шум. Распределение добавляемого шума отражает точку зрения аналитика на естественные виды шумов, свойственные конкретному типу данных. Однако потери вычисляются по отношению к *исходным* примерам тренировочных данных, а не к их зашумленным версиям. Исходные тренировочные данные относительно чисты, хотя и можно ожидать, что тестовые данные будут содержать шумы. Поэтому автокодирущик учится восстанавливать чистые представления из испорченных данных. Обычный подход к добавлению шума заключается в случайной установке нулевых значений для определенной доли f входов [506]. Этот подход особенно эффективен в случае бинарных входов. Параметр f регулирует уровень искажения входных данных. Значение данного параметра можно задать фиксированным или разрешить ему изменяться случайным образом для различных тренировочных примеров. В некоторых случаях, когда входы представляют собой вещественные значения, также добавляется гауссовский шум. Более подробная информация относительно шумоподавляющего автокодирущика приведена в разделе 4.10.2. Тесно связанный с ним *контрактивный автокодирущик* (contractive autoencoder) обсуждается в разделе 4.10.3.

Одним из интересных применений автокодирущиков является создание стилизованных художественных изображений с помощью только его *кодирующей* компоненты. В основе этой идеи лежит понятие *вариационных автокодирущиков* (variational autoencoders) [242, 399], в которых функция потерь видоизменяется таким образом, чтобы навязать специальную структуру скрытому слою. Например, в функцию потерь можно добавить член, усиливающий эффект того, что скрытые переменные извлекаются из гауссовского распределения. Далее можно повторно извлекать образцы данных из этого гауссовского распределения и использовать только декодирующую компоненту сети для порождения примеров исходных данных. Сгенерированные примеры часто представляют реалистичные выборки из оригинального распределения данных.

Родственной моделью являются *генеративно-сопоставительные сети* (generative adversarial network — GAN), ставшие особенно популярными в последние годы. В этой модели обучение декодирующей сети сочетается с обучением конкурирующего дискриминатора для создания порожденных образцов набора данных. Генеративно-сопоставительные сети часто применяются для обработки изображений, видео и текстовых данных и способны генерировать изображения и видео в стиле результатов работы ветви ИИ под названием “мечтание” (dreaming). Эти методы также полезны для преобразования изображений в изображения. Вариационные автокодирущики подробно обсуждаются в разделе 4.10.4, а генеративно-сопоставительные сети — в разделе 10.4.

Автокодировщики могут использоваться для вложения мультимодальных данных в присоединенное латентное пространство. Мультимодальные данные — это, по сути, данные с гетерогенными входными признаками. Например, изображение с описательными метками можно рассматривать как мультимодальные данные. Последние создают трудности для приложений *интеллектуального анализа данных*, поскольку различные признаки требуют разных способов обработки. Вложение гетерогенных атрибутов в унифицированное пространство позволяет устранить этот источник трудностей для интеллектуального анализа. С целью вложения гетерогенных данных в присоединенное пространство можно использовать автокодировщик. Соответствующий пример приведен на рис. 2.12. Представленный на этом рисунке автокодировщик содержит только один слой, хотя в общем случае их может быть несколько [357, 468]. Подобные присоединенные пространства могут быть очень полезны для целого ряда задач.

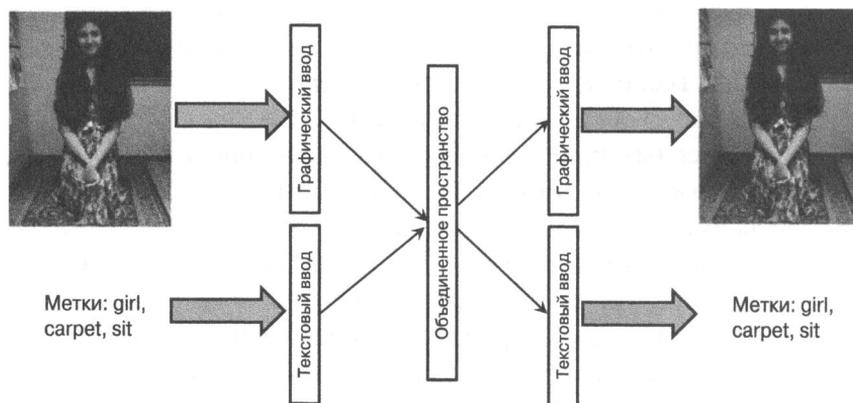


Рис. 2.12. Мультимодальное вложение с помощью автокодировщиков

Наконец, автокодировщики используются для улучшения процесса обучения в нейронных сетях. Конкретным примером может служить *предварительное обучение* (pretraining), в котором автокодировщик применяется для инициализации весов нейронной сети. Базовая идея заключается в том, что обучение разнородной структуре набора данных также может оказаться полезным для таких задач обучения с учителем, как классификация. Это объясняется тем, что признаки, определяющие разнородный набор данных, часто могут быть более информативными в терминах их взаимосвязи с различными классами. Методы предварительного обучения обсуждаются в разделе 4.7.

2.5.7. Рекомендательные системы: предсказание значения для индекса строки

Одним из наиболее интересных применений матричной факторизации является проектирование нейронных архитектур для рекомендательных систем. Рассмотрим матрицу рейтинговых оценок D размера $n \times d$, где n — количество пользователей, а d — количество оцениваемых объектов. (i, j) -й элемент матрицы — это оценка пользователя i , присвоенная им объекту j . Однако большинство элементов данной матрицы не определено, что создает трудности в случае использования традиционной архитектуры автокодировщика. Это обусловлено тем, что традиционные автокодировщики предназначены для работы с полностью определенными матрицами, когда элементом входных данных является целая строка. С другой стороны, рекомендательные системы приспособлены для поэлементного обучения, при котором может быть доступен только очень небольшой поднабор оценок из всей строки. На практике вход рекомендательной системы можно рассматривать в виде набора триплетов следующего вида, содержащих идентификатор строки, идентификатор столбца и рейтинговую оценку:

$$\langle \text{RowId} \rangle, \langle \text{ColumnId} \rangle, \langle \text{Rating} \rangle$$

Как и в случае традиционных видов матричной факторизации, матрица рейтингов D задается выражением UV^T . Однако имеется и отличие, суть которого в том, что обучение матриц U и V должно проводиться с использованием входных данных на основе триплетов указанного выше вида, поскольку наблюдаются не все элементы D . Поэтому естественный подход заключается в создании архитектуры, в которой входы не испытывают влияния из-за отсутствия элементов и могут быть однозначно определены. Входной слой содержит n входных элементов, количество которых совпадает с количеством строк (пользователей). Однако входом является кодированный напрямую индекс идентификатора строки. Поэтому лишь один элемент входа имеет значение 1, тогда как остальные получают нулевые значения. Скрытый слой содержит k нейронных элементов, где k — ранг факторизации. Наконец, выходной слой содержит d нейронных элементов, где d — количество столбцов (оцениваемых объектов). Выходом является вектор, содержащий d оценок (даже если наблюдениями является лишь небольшой их поднабор). Наша цель — обучить нейронную сеть с помощью неполной матрицы данных D таким образом, чтобы она представляла на выходе все рейтинги, соответствующие индексу строки с прямым кодированием после его подачи на вход. Данный подход должен обеспечить восстановление данных путем обучения рейтингам, связанным с каждым индексом строки.

Пусть U — матрица весов соединений размера $n \times k$, связывающих вход со скрытым слоем, а V^T — аналогичная матрица размера $k \times d$, связывающая скрытый слой с выходом. Обозначим элементы матрицы U как u_{iq} , а элементы матрицы V — как v_{jq} . Предположим, что все функции активации — линейные. Кроме того, пусть \bar{e}_r — входной вектор (строка) для r -го пользователя в представлении прямого кодирования. Этот вектор-строка имеет n измерений, из которых лишь r -я компонента равна 1, тогда как все остальные равны нулю. Функция потерь представляет собой сумму квадратов ошибок в выходном слое. Однако в силу отсутствия некоторых компонент не все выходные узлы имеют наблюдаемое выходное значение, и обновляются лишь известные компоненты. Общая архитектура этой нейронной сети приведена на рис. 2.13. Для любого конкретного входа в виде строки мы в действительности тренируем нейронную сеть, которая является подмножеством этой базовой сети, в зависимости от того, какие элементы входа заданы. Тем не менее *предсказания* можно получить для всех выходов сети (даже при том, что функция потерь не может быть вычислена для отсутствующих элементов на входе). Учитывая, что матричные умножения выполняются сетью с линейными активациями, нетрудно увидеть, что вектор d выходов для r -го пользователя задается выражением $\bar{e}_r UV^T$. По сути, умножение на вектор \bar{e}_r извлекает r -ю строку из матрицы UV^T . Эти значения появляются в выходном слое и представляют предсказания рейтингов объектов для r -го пользователя. Поэтому значения всех признаков восстанавливаются за один раз.

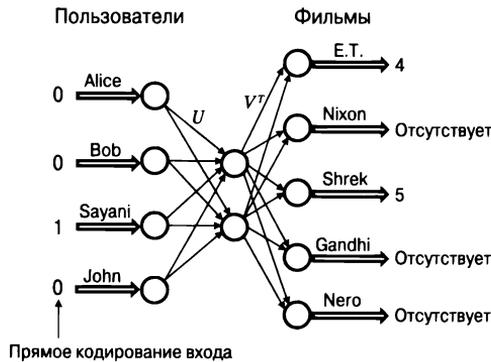


Рис. 2.13. Кодировщик, преобразующий индекс строки в значение для матричной факторизации с отсутствующими значениями

Как происходит обучение в этом случае? Данная структура в основном привлекательна тем, что позволяет выполнять тренировку как построчно, так и поэлементно. Если тренировка проводится построчно, то входом является индекс строки с прямым кодированием, и тогда для вычисления потерь используются все *заданные* элементы строки. Алгоритм обратного распространения

ошибки начинает выполняться лишь в тех узлах, для которых определены значения. С точки зрения теории каждая из строк тренируется с использованием своей сети, включающей лишь некоторое подмножество базовых выходных узлов (в зависимости от того, какие элементы представляют наблюдаемые значения), но все эти сети совместно используют веса. Данная ситуация представлена рис. 2.14, где показаны две сети рекомендательной системы фильмов, соответствующие рейтинговым оценкам Боба (Bob) и Саяни (Sayani). Например, Боб не присвоил оценку фильму *Шрек* (Shrek), в результате чего соответствующий выходной узел отсутствует. В то же время, поскольку фильм *Инопланетянин* (E. T.) был оценен обоими пользователями, k -мерные скрытые факторы для этого фильма в матрице V будут обновляться в процессе обратного распространения ошибки при обработке данных Боба или Саяни. Этой возможностью выполнять тренировку с использованием лишь подмножества выходных узлов иногда пользуются для оптимизации производительности путем сокращения времени тренировки даже в тех случаях, когда заданы все выходы. Такие ситуации часто встречаются в бинарных рекомендательных наборах данных, также называемых *наборами данных с неявными пользовательскими рейтинговыми оценками*, когда преобладающее большинство выходов имеет нулевые значения. В подобных случаях для тренировки с использованием методов матричной факторизации семплируется лишь подмножество нулей [4]. Эта методика называется *отрицательное семплирование* (negative sampling). Конкретным примером такого подхода являются нейронные модели для обработки естественного языка, такие как *word2vec*.

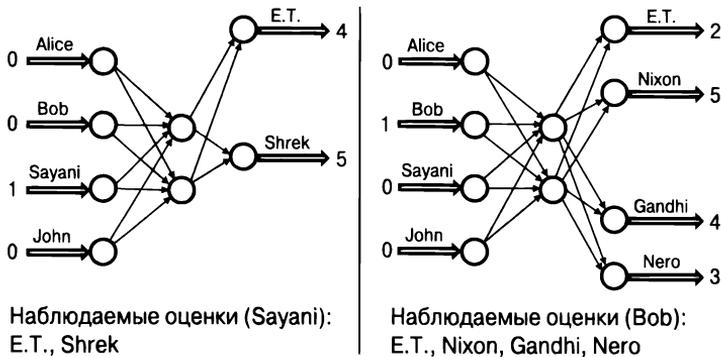


Рис. 2.14. Отбрасывание выходных узлов на основе отсутствующих значений. Выходные узлы исключаются лишь на время тренировки. В создании прогнозов участвуют все узлы. Аналогичные результаты также могут быть получены с помощью архитектуры RBM (рис. 6.5)

Также возможна поэлементная тренировка, когда входом является отдельный триплет. В таком случае потери вычисляются лишь для индекса одного

столбца, указанного в триплете. Рассмотрим случай, когда индексом строки является i , а индексом столбца — j . В этом конкретном случае, когда единственной ошибкой, вычисляемой в выходном слое, является $y - \hat{y} = e_{ij}$, алгоритм обратного распространения ошибки по сути обновляет веса вдоль всех k путей, ведущих от узла j выходного слоя к узлу i входного слоя. Эти k путей проходят через k узлов скрытого слоя. Нетрудно показать, что процесс обновления вдоль q -го такого пути описывается следующими выражениями:

$$u_{iq} \leftarrow u_{iq}(1 - \alpha\lambda) + \alpha e_{ij} v_{jq},$$

$$v_{jq} \leftarrow v_{jq}(1 - \alpha\lambda) + \alpha e_{ij} u_{iq},$$

где α — размер шага, а λ — параметр регуляризации. Эти обновления идентичны тем, которые применяются в методе стохастического градиентного спуска для матричной факторизации в рекомендательных системах. Однако важным преимуществом нейронной архитектуры (по сравнению с традиционной матричной факторизацией) является то, что мы можем изменять ее множеством различных способов для усиления тех или иных свойств. Например, в случае матриц с бинарными данными мы можем использовать на выходе сети логистический слой. Такой подход приводит к *логистической матричной факторизации*. Мы можем создавать более мощные модели, включая в сеть несколько скрытых слоев. В случае матриц с категориальными элементами (и связанными весами на основе счетчиков) в самом конце сети можно задействовать слой Softmax. Такой подход приводит к *мультиномиальной матричной факторизации*. На сегодняшний день мы не располагаем сведениями о формальном описании мультиномиальной матричной факторизации в традиционном машинном обучении, и тем не менее эта простая модификация нейронной архитектуры используется (неявно) в рекомендательных системах. Вообще говоря, ввиду модульной структуры нейронных архитектур в процессе работы с ними нередко приходится сталкиваться со сложными моделями. Коль скоро эмпирические результаты подтверждают робастность нейронной архитектуры, пытаться искать связь между ней и какой-либо обычной моделью машинного обучения вовсе не обязательно. Например, две вариации (в высшей степени успешные) скип-грамм-модели *word2vec* [325, 327] соответствуют логистической и мультиномиальной разновидностям факторизации матриц контекстов слов. Однако этот факт, похоже, не был замечен⁶ ни авторами оригинальной модели *word2vec* [325, 327], ни сообществом энтузиастов. В традиционном машинном обучении модели наподобие логистической

⁶ В работе [287] было обращено внимание на существование ряда *неявных* взаимосвязей подобного рода с матричной факторизацией, но при этом не были замечены более прямые связи, указанные в данной книге. Некоторые из этих взаимосвязей также были отмечены в работе [6].

матричной факторизации считаются относительно экзотическими методиками, которые были предложены лишь недавно [224]. Тем не менее эти усложненные модели представляют сравнительно простые нейронные архитектуры. В целом абстракция нейронной сети упрощает специалистам-практикам (не обладающим солидной математической подготовкой) работу с усложненными методами машинного обучения, позволяя переложить все заботы о деталях оптимизации на фреймворк обратного распространения ошибки.

2.5.8. Обсуждение

Основная цель данного раздела состояла в демонстрации преимуществ модульной природы нейронных сетей при обучении без учителя. В нашем конкретном примере мы начали с простой имитации SVD, а затем показали, как внесение незначительных изменений в нейронную архитектуру обеспечивает достижение самых разных целей в интуитивно понятных задачах. В то же время с архитектурной точки зрения объем усилий со стороны аналитика, требуемых для перехода от одной архитектуры к другой, часто измеряется несколькими строками кода. Это объясняется тем, что современное программное обеспечение для построения нейронных сетей часто предоставляет шаблоны, позволяющие описывать структуру сети, в которой каждый слой определяется независимо от других. В некотором смысле нейронная сеть строится из хорошо известных типов блоков машинного обучения подобно тому, как ребенок собирает игрушку из элементов конструктора. Обо всех деталях оптимизации заботится алгоритм обратного распространения ошибки, избавляющий пользователя от необходимости выполнения всех сложных действий. Рассмотрим важные математические отличия в конкретных деталях SVD и логистической матричной факторизации. Переход от линейного выходного слоя к сигмоидному (одновременно со сменой функции потерь) может потребовать изменения буквально нескольких строк кода, тогда как большая часть остального кода (которая и сама не очень большая) может остаться нетронутой. Модульность такого типа позволяет решать самые разнообразные задачи. Кроме того, автокодировщики связаны с методом обучения без учителя другого типа, известным под названием ограниченной машины Больцмана (restricted Boltzmann machine — RBM). Эти методы также могут применяться для создания рекомендательных систем, которые обсуждаются в разделе 6.5.2.

2.6. Word2vec: применение простых архитектур нейронных сетей

Методы нейронных сетей используются для обучения *векторным (распределенным) представлениям слов* (word embeddings) текстовых данных (также

называемых *вложениями*). Вообще говоря, для создания векторных представлений слов и документов можно использовать и такие методы, как SVD (сингулярное разложение). Для этого в SVD создается матрица размера $n \times d$, представляющая количество вхождений каждого слова в каждом документе. Затем эта матрица факторизуется в виде $D \approx UV$, где U и V — матрицы размера $n \times k$ и $k \times d$ соответственно. Строки U содержат вложения документов, а столбцы V — вложения слов. Обратите внимание на то, что мы несколько изменили обозначения по сравнению с предыдущим разделом (используя запись UV вместо UV^T в выражении, описывающем факторизацию), поскольку эта форма более удобна для целей данного раздела.

В то же время SVD — это метод, в котором документ рассматривается просто как *мешок слов* (bag of words). В данном случае нас интересует факторизация, в которой для создания вложений используются порядковые номера слов во всей их совокупности, применяемой для создания вложений. Здесь основной целью является создание вложений слов, а не вложений документов. Семейство моделей *word2vec* хорошо приспособлено для создания вложений слов. Это семейство включает две модели, ориентированные на решение следующих двух задач.

1. *Предсказание целевых слов исходя из контекста.* Эта модель пытается предсказать i -е слово, w_i , в предложении, используя окно шириной t , охватывающее данное слово. Таким образом, для предсказания целевого слова w_i используются слова $w_{i-t} w_{i-t+1} \dots w_{i-1} w_{i+1} \dots w_{i+t-1} w_{i+t}$. Эту модель также называют *моделью непрерывного мешка слов* (continuous bag-of-words — CBOW).
2. *Предсказание контекстов для целевых слов.* Эта модель пытается предсказать контекст $w_{i-t} w_{i-t+1} \dots w_{i-1} w_{i+1} \dots w_{i+t-1} w_{i+t}$, окружающий i -е слово в предложении, которому соответствует обозначение w_i . Такую модель называют *скип-граммой* (skip-gram). Однако предсказания подобного рода могут делаться с помощью двух разных моделей. Одна из них — *мультиномиальная модель*, которая предсказывает одно слово из d возможных исходов. Вторая — *модель Бернулли*, которая предсказывает, присутствует ли каждый контекст для конкретного слова. Для повышения производительности и точности предсказаний вторая модель использует *отрицательное семплирование контекстов* (negative sampling of contexts).

В данном разделе мы обсудим оба этих метода.

2.6.1. Нейросетевые вложения в рамках модели непрерывного мешка слов

В модели *непрерывного мешка слов* (continuous bag-of-words — CBOW) все тренировочные пары — это пары “контекст — слово”, в которых входом является окно контекстных слов, а предсказанием — целевое слово. Контекст содержит $2t$ слов, из которых t слов предшествуют целевому слову, а остальные t слов следуют за ним. Чтобы упростить обозначения, определим длину контекста как $m = 2t$. Таким образом, входом системы является набор из m слов. Не теряя общности, введем для пронумерованных слов контекста обозначения $w_1 \dots w_m$, а для целевого слова (выхода) — обозначение w . Обратите внимание на то, что w можно рассматривать как категориальную переменную, имеющую d возможных значений, где d — размер словаря. Целью создания вложений (векторных представлений) слов является вычисление вероятности $P(w | w_1 w_2 \dots w_m)$ и максимизация произведения этих вероятностей по всем тренировочным примерам.

Общая архитектура такой модели представлена на рис. 2.15. В этой архитектуре имеется один входной слой, содержащий $m \times d$ узлов, скрытый слой, содержащий p узлов, и выходной слой, содержащий d узлов. Узлы входного слоя объединены в m различных групп, каждая из которых состоит из d элементов. Каждая такая группа из d входных элементов представляет собой входной вектор для одного из m слов контекста, моделируемых в рамках модели CBOW, заданный в представлении прямого кодирования. Лишь один из этих d входов будет равен 1, в то время как все остальные входы будут равны нулю. Поэтому для входов можно использовать обозначение x_{ij} , где индексы конкретизируют позицию в контексте и идентификатор слова. Таким образом, $x_{ij} \in \{0, 1\}$, где $i \in \{1 \dots m\}$ — позиция в контексте, а $j \in \{1 \dots d\}$ — идентификатор слова.

Скрытый слой содержит p элементов, где p — размерность скрытого слоя в модели *word2vec*. Обозначим через h_1, h_2, \dots, h_p выходы узлов скрытого слоя. Обратите внимание на то, что для каждого из d слов, входящих в состав словаря, имеется m различных представителей во входном слое, соответствующих m различным контекстным словам, но все эти m соединений имеют одинаковые веса. Такие веса называют *разделяемыми*. Разделение весов — обычный прием, используемый в нейронных сетях в целях регуляризации на основании эмпирически установленных закономерностей поведения, присущих конкретным данным. Обозначим через u_{jq} разделяемый вес каждого соединения, связывающего j -е слово из словаря с q -м узлом скрытого слоя. Заметьте, что каждая из m групп во входном слое соединяется со скрытым слоем посредством связей, веса которых определяются одной и той же матрицей U размера $d \times p$ (рис. 2.15).

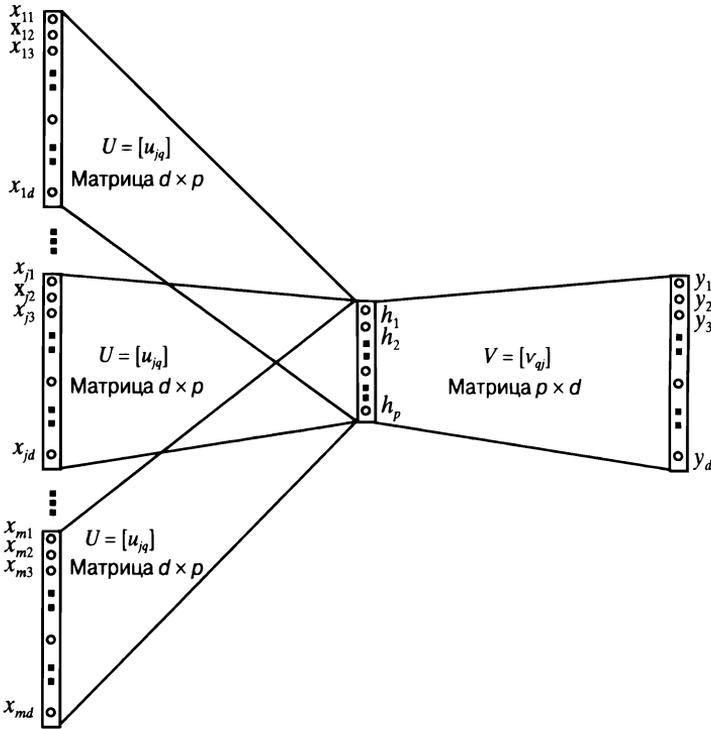


Рис. 2.15. Word2vec: модель CBoW. Обратите внимание на то, в чем этот рисунок сходен, а в чем отличается от рис. 2.13, соответствующего единственному набору входов и линейному выходному слою. Того же результата можно было бы добиться, свернув t наборов, состоящих из d входных узлов каждый, в единственный набор, состоящий из d входов, и агрегировав t входов в представлении прямого кодирования в одном окне. В этом случае вход уже не будет представлен прямыми кодами

Отметим, что вектор $\bar{u}_j = (u_{j1}, u_{j2}, \dots, u_{jp})$ можно рассматривать как p -мерное вложение j -го входного слова на всем корпусе текста, а $h = (h_1, \dots, h_p)$ представляет вложение конкретного экземпляра входного контекста. Тогда выход скрытого слоя получается путем усреднения вложений слов, присутствующих в контексте. Другими словами, имеем следующее соотношение:

$$h_q = \sum_{i=1}^m \left[\sum_{j=1}^d u_{jq} x_{ij} \right] \quad \forall q \in \{1 \dots p\}. \quad (2.46)$$

Во многих описаниях используют дополнительный фактор t в знаменателе с правой стороны, хотя такой тип мультипликативного масштабирования (с помощью константы) и не является существенным. Это соотношение может быть записано в векторной форме:

$$\bar{h} = \sum_{i=1}^m \sum_{j=1}^d \bar{u}_j x_{ij}. \quad (2.47)$$

По сути, прямые коды входных слов агрегируются, а это означает, что порядок следования слов в пределах окна размером m не влияет на выход модели. Это и есть причина того, почему модель получила название *непрерывный мешок слов*. Однако последовательный характер информации все же учитывается за счет отнесения предсказания лишь к области окна контекста.

На основе векторного представления $(h_1 \dots h_p)$ с использованием функции активации *Softmax* для каждого из d слов вычисляется выходная оценка вероятности того, что данное слово является целевым. Веса в выходном слое параметризуются с помощью матрицы $V = [v_{qj}]$ размера $p \times d$. Обозначим j -й столбец матрицы V как \bar{v}_j . После применения функции активации *Softmax* на выходе создаются d выходных значений $\hat{y}_1 \dots \hat{y}_d$, представляющих собой вещественные числа в интервале $(0, 1)$. Сумма этих значений равна 1, и они могут интерпретироваться как вероятности. Для заданного тренировочного примера истинное значение должно быть равно 1 только для одного из выходов $y_1 \dots y_d$, тогда как все остальные значения должны быть равны нулю. Это условие можно записать в следующем виде:

$$y_j = \begin{cases} 1, & \text{если целевым словом } w \text{ является } j\text{-е слово,} \\ 0 & \text{в противном случае.} \end{cases} \quad (2.48)$$

Функция *Softmax* вычисляет вероятность $P(w | w_1 w_2 \dots w_m)$ того, что выход y_j в представлении прямого кодирования является истинным значением, следующим образом:

$$\hat{y}_j = P(y_j = 1 | w_1 \dots w_m) = \frac{\exp\left(\sum_{q=1}^p h_q v_{qj}\right)}{\sum_{k=1}^d \exp\left(\sum_{q=1}^p h_q v_{qk}\right)}. \quad (2.49)$$

Заметьте, что эта вероятностная форма предсказания базируется на слое *Softmax* (см. раздел 1.2.1.4). Для конкретного целевого слова $w = r \in \{1 \dots d\}$ функция потерь задается как $L = -\log[P(y_r = 1 | w_1 \dots w_m)] = -\log(\hat{y}_r)$. Использование отрицательного логарифма превращает произведение вероятностей по различным тренировочным примерам в аддитивную функцию потерь.

Обновления определяются с помощью алгоритма обратного распространения ошибки по мере пропуска тренировочных примеров по одному через нейронную сеть. Прежде всего, производная вышеупомянутой функции потерь может быть использована для обновления градиентов матрицы весов V в выходном слое. Затем алгоритм обратного распространения ошибки может быть использован для обновления матрицы U весов связей между входным и скрытым слоями. Уравнения обновления со скоростью обучения α выглядят так:

$$\begin{aligned}\bar{u}_i &\leftarrow \bar{u}_i - \alpha \frac{\partial L}{\partial \bar{u}_i} \quad \forall i, \\ \bar{v}_j &\leftarrow \bar{v}_j - \alpha \frac{\partial L}{\partial \bar{v}_j} \quad \forall j.\end{aligned}$$

Входящие в эти выражения частные производные легко вычисляются [325, 327, 404].

Вероятность совершения ошибки предсказания для j -го слова словаря определяется как $|y_j - \hat{y}_j|$. Однако мы будем использовать ошибки *со знаком*, e_j , когда ошибка положительна только для корректного слова с $y_j = 1$ и отрицательна для всех остальных слов словаря. Это достигается за счет отбрасывания знака модуля:

$$\varepsilon_j = y_j - \hat{y}_j. \quad (2.50)$$

Можно показать, что ошибка e_j равна отрицательной производной кросс-энтропийных потерь по j -му входу Softmax ($\bar{h} \cdot \bar{v}_j$). Этот результат продемонстрирован в разделе 3.2.5.1 и может быть использован для вычисления обновлений в процессе обратного распространения ошибки. Тогда обновления⁷ для конкретного входного контекста и выходного слова принимают следующий вид:

$$\begin{aligned}\bar{u}_i &\leftarrow \bar{u}_i + \alpha \sum_{j=1}^d \varepsilon_j \bar{v}_j \quad [\forall \text{ слова } i, \text{ присутствующие в окне контекста}], \\ \bar{v}_j &\leftarrow \bar{v}_j + \alpha \varepsilon_j \bar{h} \quad [\forall j \text{ в словаре}],\end{aligned}$$

где $\alpha > 0$ — скорость обучения. Повторения одного и того же слова i в окне контекста запускают многократные обновления \bar{u}_i . Примечательно, что входные

⁷ При записи операций сложения для векторов \bar{u}_i и \bar{v}_j в приведенных выше выражениях для обновлений допущена неточность. Дело в том, что \bar{u}_i — вектор-строка, тогда как \bar{v}_j — вектор-столбец. В этом разделе мы опускаем явные указания на транспонирование одного из этих векторов, чтобы не загромождать нотацию, поскольку природа обновлений понятна на интуитивном уровне.

вложения контекстных слов агрегируются в обоих обновлениях, учитывая тот факт, что \bar{h} агрегирует входные вложения в соответствии с уравнением 2.47. Этот тип агрегации оказывает сглаживающее влияние на модель SBOW, что особенно полезно в случае небольших объемов данных.

Тренировочные примеры пар “контекст — цель” предоставляются по одному, и веса тренируются до сходимости. Необходимо подчеркнуть, что модель *word2vec* предоставляет не одно, а два различных вложения, соответствующих p -мерным строкам матрицы U и p -мерным столбцам матрицы V . Первый тип вложения слов называется *входным*, а второй — *выходным*. В модели SBOW входное вложение представляет контекст, и поэтому имеет смысл использовать выходное вложение. Однако входное вложение (или сумма/конкатенация входного и выходного вложений) также может быть полезным для многих задач.

2.6.2. Нейросетевые вложения в рамках модели скип-грамм

В модели скип-грамм целевые слова используются для предсказания m слов контекста. Поэтому в данном случае мы имеем один вход и m выходов. Одной из проблем при работе с моделью SBOW является то, что усреднение входных слов в окне контекста (на основе чего создается скрытое представление) приводит к (благоприятному) сглаживающему эффекту в случае данных небольшого объема, но не обеспечивает в полной мере использование всех преимуществ, предоставляемых большими объемами данных. В тех случаях, когда доступных данных много, более предпочтительна модель скип-грамм (N-грамм с пропусками).

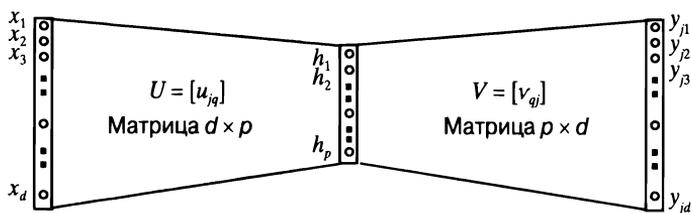
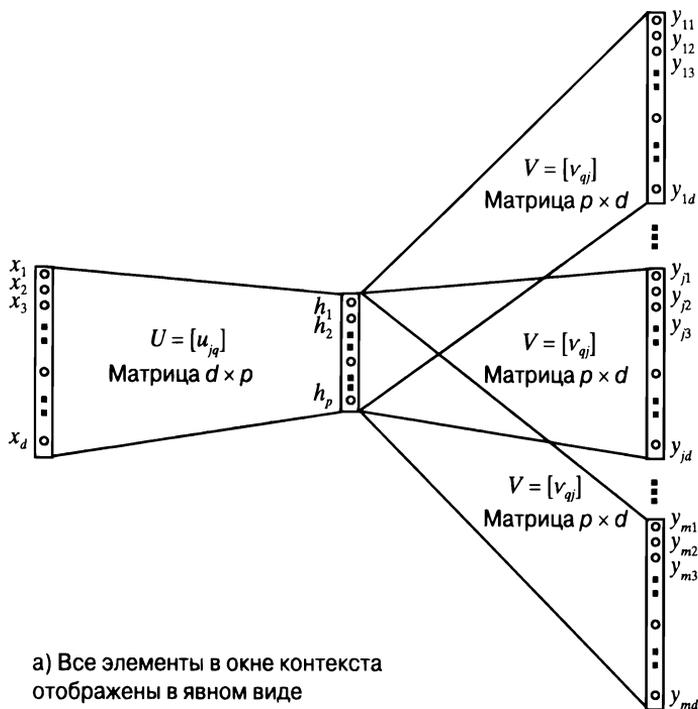
В модели скип-грамм входом является целевое слово w , а выходами служат m контекстных слов $w_1 \dots w_m$. Поэтому задача заключается в оценке величины $P(w_1 w_2 \dots w_m | w)$, которая отличается от величины $P(w | w_1 w_2 \dots w_m)$, оцениваемой в модели SBOW. Как и в случае модели непрерывного мешка слов, в модели скип-грамм можно использовать прямое кодирование (категориального) входа и выходов. После выполнения такого кодирования модель скип-грамм будет иметь бинарные входы $x_1 \dots x_d$, соответствующие d возможным значениям единственного входного слова. Аналогичным образом выход каждого тренировочного примера кодируется в виде $m \times d$ значений $y_{ij} \in \{0, 1\}$, где i изменяется в пределах от 1 до m (размер окна контекста), а j — от 1 до d (размер словаря). Каждое значение $y_{ij} \in \{0, 1\}$ указывает на то, принимает ли i -е слово контекста j -е возможное значение для данного тренировочного примера. Однако на выходе (i, j) -го элемента с помощью функции *Softmax* вычисляется вероятностное значение $\bar{y}_{ij} = P(y_{ij} = 1 | w)$. Поэтому суммирование вероятностей \bar{y}_{ij} в выходном слое при фиксированном индексе i и переменном индексе j дает 1, поскольку i -ю позицию в контексте занимает ровно одно из d слов. Скрытый слой содержит p элементов с выходами $h_1 \dots h_p$. Каждый вход x_j соединен со всеми скрытыми узлами посред-

ством матрицы U размера $d \times p$. Кроме того, p скрытых узлов связаны с каждой из m групп, включающих по d выходных узлов, с помощью того же набора разделяемых весов. Этот набор разделяемых весов связей между p скрытыми узлами и d выходными узлами каждого из слов контекста определяется матрицей V размера $p \times d$. Обратите внимание на то, что структура “вход — выход” модели скип-грамм представляет собой обращенную структуру “вход — выход” модели SVOW. Нейронная архитектура модели скип-грамм приведена на рис. 2.16, а. Однако в случае модели скип-грамм можно свернуть m идентичных выходов в один и получить те же результаты, используя определенный тип мини-пакетирования в процессе стохастического градиентного спуска. В частности, все элементы одного окна контекста принудительно включаются в один и тот же мини-пакет. Эта архитектура приведена на рис. 2.16, б. Поскольку значение m невелико, использование данного специфического типа формирования мини-пакетов оказывает лишь весьма ограниченное влияние на результаты, и упрощенной архитектуры, изображенной на рис. 2.16, б, достаточно для описания модели, независимо от использования того или иного специфического типа мини-пакетирования. В дальнейшем мы будем использовать архитектуру, приведенную на рис. 2.16, а.

Выход скрытого слоя можно вычислить на основании входного слоя с помощью матрицы весов $U = [u_{jq}]$ размера $d \times p$, связывающей входной и выходной слои:

$$h_q = \sum_{j=1}^d u_{jq} x_j \quad \forall q \in \{1 \dots p\}. \quad (2.51)$$

В силу использования прямого кодирования входного слова w в терминах $x_1 \dots x_d$ вышеприведенное уравнение интерпретируется очень просто. Если выходное слово w является r -м словом, то u_{rq} просто копируется в q -й узел скрытого слоя для каждого $q \in \{1 \dots p\}$. Иными словами, r -я строка \bar{u}_r матрицы U копируется в скрытый слой. Как уже обсуждалось выше, скрытый слой связан с m группами, включающими по d выходных узлов, каждый из которых связан со скрытым слоем с помощью матрицы $V = [v_{qj}]$ размера $p \times d$. Каждая из этих групп, включающих по d выходных узлов, вычисляет вероятности различных слов для конкретного слова контекста. j -й столбец матрицы V , \bar{v}_j , представляет выходное вложение j -го слова. Выход \hat{y}_{ij} — это вероятность того, что слово в i -й позиции контекста получает j -е слово словаря. Однако, поскольку одна и та же матрица V разделяется всеми группами, нейронная сеть предсказывает для каждого из слов контекста одно и то же мультиномиальное распределение. Поэтому имеем следующее соотношение:



Мини-пакетирование m d -мерных выходных векторов в каждом окне контекста в процессе стохастического градиентного спуска. Отображенные выходы y_k соответствуют j -му из m выходов.

б) Все элементы в окне контекста не отображены в явном виде

Рис. 2.16. Word2vec: модель на основе skip-грамм. Обратите внимание на сходство с рис. 2.13, соответствующим одиночному набору линейных выходов. Кроме того, для получения того же результата можно было бы свернуть m наборов, включающих по d выходных узлов (а), в один набор из d выходов, и сформировать мини-пакет из m примеров, содержащихся в одном окне контекста, в процессе стохастического градиентного спуска. Все элементы, входящие в состав мини-пакета, представлены явно (а), тогда как элементы мини-пакета представлены неявно (б). В то же время, если принять во внимание природу мини-пакетирования, эти два варианта эквивалентны

$$\hat{y}_{ij} = P(y_{ij} = 1 | w) = \frac{\exp\left(\sum_{q=1}^p h_q v_{qj}\right)}{\sum_{k=1}^d \exp\left(\sum_{q=1}^p h_q v_{qk}\right)} \quad \forall i \in \{1 \dots m\}. \quad (2.52)$$

не зависит от контекстной позиции i

Обратите внимание на то, что вероятность \hat{y}_{ij} остается одной и той же для всех значений i при фиксированном значении j , поскольку правая часть этого уравнения не зависит от точной позиции i в окне контекста.

Функцией потерь для алгоритма обратного распространения ошибки является отрицательная сумма логарифмического правдоподобия правильных значений $y_{ij} \in \{0, 1\}$ тренировочного примера (обучающей выборки). Эта функция потерь L определяется следующим выражением:

$$L = -\sum_{i=1}^m \sum_{j=1}^d y_{ij} \log(\hat{y}_{ij}). \quad (2.53)$$

Заметим, что множитель при логарифме — это истинное бинарное значение, тогда как величина под знаком логарифма — это предсказанное (вероятностное) значение. Поскольку y_{ij} представлено прямым кодом для фиксированного значения i и переменного значения j , то целевая функция имеет лишь m ненулевых членов. Для каждого тренировочного примера эта функция потерь используется в сочетании с алгоритмом обратного распространения ошибки для обновления весов соединений между узлами. Правила обновления для обучения со скоростью α будут выглядеть так:

$$\begin{aligned} \bar{u}_i &\leftarrow \bar{u}_i - \alpha \frac{\partial L}{\partial \bar{u}_i} \quad \forall i, \\ \bar{v}_j &\leftarrow \bar{v}_j - \alpha \frac{\partial L}{\partial \bar{v}_j} \quad \forall j. \end{aligned}$$

Ниже мы детализируем эти выражения для обновлений, после того как введем некоторые дополнительные обозначения.

Вероятность совершения ошибки при предсказании j -го слова словаря для i -й позиции контекста определяется разностью $|y_{ij} - \bar{y}_{ij}|$. Однако мы используем ошибки со знаком e_{ij} , когда положительную вероятность имеют только предсказанные слова (положительные примеры). Этого можно достигнуть, опустив знак модуля:

$$\varepsilon_{ij} = y_{ij} - \hat{y}_{ij}. \quad (2.54)$$

Тогда выражения для обновления отдельного входного слова r и его выходного контекста могут быть переписаны в следующем виде:

$$\bar{u}_r \leftarrow \bar{u}_r + \alpha \sum_{j=1}^d \left[\sum_{i=1}^m \varepsilon_{ij} \right] \bar{v}_j \quad [\text{только для входного слова } r],$$

$$\bar{v}_j \leftarrow \bar{v}_j + \alpha \left[\sum_{i=1}^m \varepsilon_{ij} \right] \bar{h} \quad [\text{для всех слов } j \text{ в словаре}],$$

где $\alpha > 0$ — скорость обучения. p -мерные строки матрицы U используются в качестве вложений слов. Иначе говоря, принято использовать входные вложения в строках матрицы U , а не выходные вложения в столбцах матрицы V . Как утверждается в [288], добавление входных и выходных вложений может пригодиться в некоторых задачах (но навредить в других). Также может быть полезной конкатенация этих двух вложений.

Практические проблемы

Существуют определенные проблемы практического характера, связанные с точностью и производительностью фреймворка *word2vec*. Достижение компромисса между смещением и дисперсией обеспечивается размерностью вложения, определяемой количеством узлов в скрытом слое. Увеличение размерности вложений улучшает распознавательные свойства, но требует большего количества данных. В общем случае вложения имеют размерность порядка нескольких сотен, хотя в случае очень больших коллекций она может измеряться тысячами. Размер окна контекста обычно варьирует от 5 до 10, причем для модели скип-грамм используются более крупные окна по сравнению с моделью SBOW. Неявным эффектом использования окон случайного размера является предоставление большего веса близко расположенным словам. Модель скип-грамм более медленная, но работает лучше для нечасто встречающихся слов и более крупных наборов данных.

Суть другой проблемы заключается в том, что доминирующее влияние на результаты могут оказывать часто упоминаемые слова и части речи (такие, например, как артикль “the”). Поэтому общий подход заключается в *прореживании* (downsampling) часто упоминаемых слов, что приводит к улучшению как точности, так и производительности. Учтите, что неявным следствием этого является увеличение размера окна контекста, поскольку отбрасывание слова, расположенного в промежутке между двумя словами, сближает эти слова. Слова с опечатками встречаются сравнительно редко, и было бы трудно создавать для них значимые вложения, избегая переобучения. Поэтому такие слова игнорируются.

С вычислительной точки зрения обновление выходных вложений обходится очень дорого. Причина этого — применение функции *Softmax* ко всем d словам, входящим в состав словаря, что требует обновления каждого \bar{v}_j . Поэтому функция *Softmax* применяется иерархически с использованием кодирования Хаффмана для повышения производительности. Для получения более подробной информации отсылаем читателя к [325, 327, 404].

Скип-граммы с отрицательным семплированием

Эффективной альтернативой иерархической технике *Softmax* является метод *скип-грамм с отрицательным семплированием* (*skip-gram with negative sampling* — SGNS) [327], в котором для тренировки используются как имеющиеся, так и отсутствующие пары “слово — контекст”. В соответствии с этим методом отрицательные примеры контекстов искусственно генерируются путем семплирования слов пропорционально их встречаемости в корпусе текста (т.е. из распределения униграмм). В основе целевой функции, оптимизируемой при таком подходе и отличающейся от той, которая используется в модели скип-грамм, лежат соображения, связанные с *контрастивной оценкой шума* (*noise contrastive estimation*) [166, 333, 334].

Основная идея заключается в том, что вместо непосредственного предсказания каждого из m слов в окне контекста мы пытаемся предсказывать, присутствует ли в нем каждое из d слов словаря. Иными словами, последний слой на рис. 2.16 — это не *Softmax*-предсказание, а сигмоидный слой Бернулли. Выходной элемент для каждого слова в каждой позиции контекста — это сигмоида, предоставляющая значение вероятности того, что данная позиция содержит данное слово. Поскольку правильные значения также доступны, можно использовать логистическую функцию потерь, вычисляемую по всем словам. При таком подходе даже задача предсказания определяется по-другому. Разумеется, с вычислительной точки зрения пытаться делать бинарные предсказания для всех d слов крайне неэффективно. Поэтому в подходе SGNS используются все слова в окне контекста из положительных примеров и *выборка* слов из отрицательных примеров. Количество отрицательных примеров по отношению к количеству положительных определяется параметром выборки k . В этой видоизмененной задаче предсказания отрицательное семплирование играет существенную роль, позволяя избежать обучения тривиальных весов, которые предсказывают для всех примеров значение 1. Иными словами, мы не можем полностью отказаться от использования отрицательных примеров (т.е. не можем положить $k = 0$).

Как генерируются отрицательные выборки? Простейшее униграммное распределение обеспечивает отбор слов пропорционально частотам их относительной встречаемости $f_1 \dots f_d$ в корпусе. Лучшие результаты удается получить [327], семплируя слова пропорционально $f_j^{3/4}$, а не f_j . Как и в случае всех

других моделей *word2vec*, предположим, что U — матрица размера $d \times p$, представляющая входное вложение, а V — матрица размера $p \times d$, представляющая выходное вложение. Обозначим через u_i p -мерную строку матрицы U (входное вложение i -го), а через v_j — p -мерный столбец матрицы V (выходное вложение j -го слова). Пусть \mathcal{P} — набор положительных пар слов “цель — контекст” в окне контекста, а \mathcal{N} — набор аналогичных отрицательных пар слов, созданных путем семплирования. Поэтому размер \mathcal{P} равен размеру окна контекста m , а размер \mathcal{N} равен $m \cdot k$. Тогда (минимизируемая) целевая функция для каждого окна контекста получается путем суммирования логистических потерь по m положительным примерам и $m \cdot k$ отрицательным примерам:

$$O = - \sum_{(i,j) \in \mathcal{P}} \log(P[\text{Предсказание 1 для } (i,j)]) - \sum_{(i,j) \in \mathcal{N}} \log(P[\text{Предсказание 0 для } (i,j)]) = \quad (2.55)$$

$$= - \sum_{(i,j) \in \mathcal{P}} \log\left(\frac{1}{1 + \exp(-\bar{u}_i \cdot \bar{v}_j)}\right) - \sum_{(i,j) \in \mathcal{N}} \log\left(\frac{1}{1 + \exp(-\bar{u}_i \cdot \bar{v}_j)}\right). \quad (2.56)$$

Эта модифицированная целевая функция используется в модели скип-грамм с отрицательным семплированием (SGNS) для обновления весов U и V . Модель SGNS математически отличается от базовой модели скип-грамм, которую мы обсуждали прежде. Модель SGNS не только более эффективна, но и обеспечивает получение лучших результатов среди всех других вариантов моделей скип-грамм.

Какова фактическая нейронная архитектура SGNS

Несмотря на то что модель SGNS рассматривается в оригинальной статье по *word2vec* лишь как способ оптимизации модели скип-грамм, в ней используется существенно иная архитектура, отличающаяся применением функции активации в последнем слое. К сожалению, в оригинальной статье этот факт не подчеркнут явным образом (в ней лишь предоставляется измененная целевая функция), что приводит к некоторой путанице.

Вот что собой представляет модифицированная нейронная архитектура SGNS. В реализации SGNS слой Softmax уже не используется. Вместо этого каждое наблюдаемое значение y_{ij} (см. рис. 2.16) *независимо* обрабатывается как *бинарный* исход, а не как мультиномиальный, когда вероятностные предсказания различных исходов в позиции контекста зависят друг от друга. Вместо использования функции *Softmax* для создания предсказания \hat{y}_{ij} , в SGNS с помощью сигмоидной активации создаются вероятностные предсказания \hat{y}_{ij} того, будет ли каждый y_{ij} равен 0 или 1. После этого, суммируя логарифмические потери \hat{y}_{ij}

относительно наблюдаемых значений y_{ij} по всем $m \cdot d$ возможным значениям (i, j) , можно создать полную функцию потерь для окна контекста. Однако это непрактично, поскольку количество нулевых значений y_{ij} очень велико, а нулевые значения — это, как бы то ни было, шум. Поэтому SGNS применяет отрицательное семплирование для аппроксимации *модифицированной* целевой функции. Это означает, что для каждого окна контекста алгоритм обратного распространения ошибки выполняется лишь для некоторого поднабора $m \cdot d$ выходов, изображенных на рис. 2.16. Размер этого поднабора равен $m + m \cdot k$. Именно это обеспечивает повышение производительности. В то же время, поскольку в последнем слое используются бинарные предсказания (с помощью сигмоиды), архитектура SGNS существенно отличается от простой модели сkip-грамм даже в терминах применяемой базовой нейронной сети (логистическая активация вместо активации Softmax). Различия между моделью SGNS и простой моделью сkip-грамм аналогичны различиям между моделью Бернулли и мультиномиальной моделью в задаче наивной байесовской классификации (с применением отрицательного семплирования только модели Бернулли). Очевидно, что в обоих этих случаях первая из моделей не может рассматриваться просто как результат оптимизации производительности второй.

2.6.3. Модель SGNS — это логистическая матричная факторизация

Как уже отмечалось, в [287] показана *неявная* взаимосвязь между моделью *word2vec* и матричной факторизацией, однако ниже мы продемонстрируем более непосредственную связь между ними. Архитектура моделей сkip-грамм подозрительно напоминает архитектуры, используемые для предсказания значения по индексу строки в рекомендательных системах (см. раздел 2.5.7). Применение алгоритма обратного распространения ошибки лишь для поднабора наблюдаемых выходов аналогично идее отрицательного семплирования, за исключением того, что отбрасывание выходов при отрицательном семплировании выполняется с целью повышения производительности. В то же время, в отличие от линейных выходов, представленных на рис. 2.13, для моделирования бинарных предсказаний в модели SGNS используются логистические выходы. Модель SGNS *word2vec* можно сымитировать с помощью логистической матричной факторизации. Понять природу сходства с задачей, рассмотренной в разделе 2.5.7, будет легче, если мы подробнее рассмотрим предсказания для конкретного окна контекста с использованием следующих триплетов:

<Идентификатор_слова>, <Идентификатор_контекста>, <0/1>.

Каждое окно контекста производит $m \cdot d$ таких триплетов, хотя для отрицательного семплирования задействуются только $m \cdot k + m$ триплетов, из которых в процессе тренировки формируются *мини-пакеты*. Мини-пакетирование — еще один источник различий между рис. 2.13 и 2.16, где на последнем имеется m различных групп выходов, количество которых совпадает с количеством m положительных примеров. Однако эти различия относительно несущественны, и для представления базовой модели по-прежнему можно использовать логистическую матричную факторизацию.

Пусть $B = [b_{ij}]$ — бинарная матрица, в которой (i, j) -е значение равно 1, если слово j встречается по крайней мере один раз в контексте слова i в наборе данных, и 0 в противном случае. Вес c_{ij} для любого слова (i, j) , встречающегося в корпусе, определяется количеством вхождений слова j в контексте слова i . Веса нулевых элементов матрицы B определяются следующим образом. Для каждой строки i в B мы семплируем $k \sum_j b_{ij}$ различных элементов из строки i , выбирая их среди элементов, для которых $b_{ij} = 0$, и частота, с которой семплируется j -е слово, пропорциональна $f_j^{3/4}$. Это отрицательные примеры, а веса c_{ij} для отрицательных примеров (т.е. таких, для которых $b_{ij} = 0$) устанавливаются равными количеству семплирований каждого элемента. Как и в модели *word2vec*, p -мерные вложения i -го слова и j -го контекста обозначаются как u_i и v_j соответственно. Простейший способ факторизации — использовать взвешенную матричную факторизацию B с нормой Фробениуса:

$$\text{Минимизация}_{u, v} \sum_{i, j} c_{ij} (b_{ij} - \bar{u}_i \cdot \bar{v}_j)^2. \quad (2.57)$$

Несмотря на то что матрица B имеет размер порядка $O(d^2)$, ее разложение содержит лишь ограниченное количество ненулевых элементов в целевой функции, для которых $c_{ij} > 0$. Эти веса зависят от значений счетчиков совместной встречаемости, но некоторые нулевые элементы также имеют положительный вес. Поэтому шаги градиентного спуска могут фокусироваться лишь на элементах с $c_{ij} > 0$. Каждый цикл стохастического градиентного спуска ведет себя *линейно* относительно количества ненулевых элементов, как и в реализации SGNS модели *word2vec*.

Однако эта целевая функция также выглядит несколько иначе, чем целевая функция *word2vec*, применяемая в логистической форме. Аналогично тому, как в случае обучения с учителем бинарных целей рекомендуется заменять линейную регрессию логистической, этот же прием можно использовать в случае матричного разложения бинарных матриц [224]. Член квадратичной ошибки можно заменить уже знакомым вам членом L_{ij} , выражающим правдоподобие, который используется в логистической регрессии:

$$L_{ij} = \left| b_{ij} - \frac{1}{1 + \exp(-\bar{u}_i \cdot \bar{v}_j)} \right|. \quad (2.58)$$

Значение L_{ij} всегда находится в пределах интервала $(0, 1)$, и более высоким значениям соответствует большее правдоподобие (что приводит к максимизации целевой функции). Операция взятия по модулю в приведенном выше выражении изменяет знак лишь для отрицательных примеров, в которых $b_{ij} = 0$. Теперь можно оптимизировать следующую целевую функцию в форме минимизации:

$$\text{Минимизация}_{U, V} J = - \sum_{i,j} c_{ij} \log(L_{ij}). \quad (2.59)$$

Основное отличие от целевой функции *word2vec* (см. уравнение 2.56) заключается в том, что эта целевая функция — глобальная по всем матричным элементам, а не локальная, которая вычисляется на отдельном окне контекста. Использование мини-пакетного стохастического градиентного спуска в матричной факторизации (с подходящим образом выбранными мини-пакетами) делает такой подход почти идентичным обновлениям, выполняемым с помощью алгоритма обратного распространения *word2vec*.

Как интерпретировать этот тип матричной факторизации? Вместо соотношения $B \approx UV$ мы имеем соотношение $B \approx f(UV)$, где $f(\cdot)$ — сигмоида. Если говорить точнее, в данном случае мы имеем дело с вероятностной факторизацией, в соответствии с которой сначала вычисляется произведение матриц U и V , а затем к нему применяется сигмоида для получения параметров распределения Бернулли, из которого генерируется B :

$$P(B_{ij} = 1) = \frac{1}{1 + \exp(-\bar{u}_i \cdot \bar{v}_j)} \left[\begin{array}{l} \text{аналог логистической регрессии} \\ \text{в матричной факторизации} \end{array} \right].$$

Используя уравнение 2.58, нетрудно проверить, что L_{ij} равно $P(b_{ij} = 1)$ для положительных примеров и $P(b_{ij} = 0)$ — для отрицательных. Поэтому целевая функция факторизации принимает форму максимизируемой функции логарифмического правдоподобия. Этот тип логистической матричной факторизации обычно используется в рекомендательных системах с бинарными данными [224].

Градиентный спуск

Также представляет интерес исследовать более подробно шаги градиентного спуска при факторизации. Производные J по входным и выходным вложениям можно вычислить с помощью следующих формул:

$$\begin{aligned}
\frac{\partial J}{\partial \bar{u}_i} &= - \sum_{j:b_j=1} \frac{c_{ij} \bar{v}_j}{1 + \exp(\bar{u}_i \cdot \bar{v}_j)} + \sum_{j:b_j=0} \frac{c_{ij} \bar{v}_j}{1 + \exp(-\bar{u}_i \cdot \bar{v}_j)} = \\
&= - \underbrace{\sum_{j:b_j=1} c_{ij} P(b_{ij} = 0) \bar{v}_j}_{\text{Положительные ошибки}} + \underbrace{\sum_{j:b_j=1} c_{ij} P(b_{ij} = 1) \bar{v}_j}_{\text{Отрицательные ошибки}} \\
\frac{\partial J}{\partial \bar{v}_j} &= - \sum_{i:b_{ij}=1} \frac{c_{ij} \bar{u}_i}{1 + \exp(\bar{u}_i \cdot \bar{v}_j)} + \sum_{i:b_{ij}=0} \frac{c_{ij} \bar{u}_i}{1 + \exp(-\bar{u}_i \cdot \bar{v}_j)} = \\
&= - \underbrace{\sum_{i:b_{ij}=1} c_{ij} P(b_{ij} = 0) \bar{u}_i}_{\text{Положительные ошибки}} + \underbrace{\sum_{i:b_{ij}=1} c_{ij} P(b_{ij} = 1) \bar{u}_i}_{\text{Отрицательные ошибки}}
\end{aligned}$$

Процедура оптимизации использует градиентный спуск, пока не будет достигнута сходимость:

$$\begin{aligned}
\bar{u}_i &\leftarrow \bar{u}_i - \alpha \frac{\partial J}{\partial \bar{u}_i} \quad \forall i, \\
\bar{v}_j &\leftarrow \bar{v}_j - \alpha \frac{\partial J}{\partial \bar{v}_j} \quad \forall j.
\end{aligned}$$

Необходимо подчеркнуть, что производные могут быть выражены в терминах вероятности совершения ошибок при предсказании b_{ij} . Такой подход часто применяется в градиентном спуске при оптимизации логарифмического правдоподобия. Также следует отметить, что производная целевой функции SGNS (см. уравнение 2.56) приводит к аналогичной форме градиента. Единственное отличие состоит в том, что производная целевой функции SGNS выражается с использованием меньшего пакета примеров, определяемых окном контекста. Мы также можем решить задачу вероятностной матричной факторизации с помощью мини-пакетного стохастического градиентного спуска. При подходящем выборе мини-пакета стохастический градиентный спуск матричной факторизации становится идентичным обновлению с помощью алгоритма обратного распространения SGNS. Единственное отличие состоит в том, что SGNS семплирует отрицательные записи для каждого набора обновлений на лету, тогда как в случае матричной факторизации отрицательные примеры заранее фиксируются. Разумеется, семплирование на лету можно задействовать также и в случае обновлений при матричной факторизации. Сходство между SGNS и матричной факторизацией также можно проследить, заметив, что архитектура, приведенная на рис. 2.16, б, почти совпадает с архитектурой матричной факторизации для рекомендательной системы, представленной на рис. 2.13. Как и в случае

рекомендательных систем, в SGNS имеются отсутствующие (отрицательные) записи. Это обусловлено тем, что отрицательное семплирование использует лишь подмножество нулевых значений. Единственное различие между двумя случаями состоит в том, что в выходном слое архитектуры SGNS используются сигмоидные элементы, тогда как в рекомендательных системах применяется линейный выходной слой. Однако в рекомендательных системах с неявной обратной связью используется *логистическая матричная факторизация* [224], аналогично тому, как это делается в модели *word2vec*.

2.6.4. Простая модель скип-грамм — это мультиномиальная матричная факторизация

Поскольку мы уже продемонстрировали, что улучшение модели скип-грамм в виде SGNS есть не что иное, как логистическая матричная факторизация, возникает вполне естественный вопрос: можно ли переформулировать в виде метода матричной факторизации также и оригинальную модель скип-грамм? Оказывается, что простую модель скип-грамм можно переформулировать как *мультиномиальную матричную факторизацию*, поскольку в самом конце сети используется слой Softmax.

Пусть $C = [c_{ij}]$ — матрица совместной встречаемости “слово — контекст” размера $d \times d$, где c_{ij} — количество вхождений слова j в контексте слова i . Пусть U — матрица размера $d \times p$, строки которой содержат входные вложения, а V — матрица размера $p \times d$, столбцы которой содержат выходные вложения. Тогда в рамках модели скип-грамм создается в целом модель, в которой вектор частотности в r -й строке C — это эмпирическая инстанциализация вероятностей, получаемых путем применения активации Softmax к r -й строке UV .

Пусть \bar{u}_i — p -мерный вектор, соответствующий i -й строке матрицы U , а \bar{v}_j — p -мерный вектор, соответствующий j -му столбцу матрицы V . Тогда функция потерь вышеупомянутой факторизации имеет следующий вид:

$$O = - \sum_{i=1}^d \sum_{j=1}^d c_{ij} \log \underbrace{\left(\frac{\exp(\bar{u}_i \cdot \bar{v}_j)}{\sum_{q=1}^d \exp(\bar{u}_i \cdot \bar{v}_q)} \right)}_{P(\text{слово}_j | \text{слово}_i)} \quad (2.60)$$

Функция потерь записывается в форме минимизации. Заметьте, что эта функция потерь идентична той, которая применяется в простой модели скип-грамм, за исключением того, что в последнем случае используется мини-пакетный стохастический градиентный спуск, в котором группируются m слов заданного контекста. Этот специфический тип мини-пакета не вносит никаких существенных отличий.

2.7. Простые архитектуры нейронных сетей для вложений графов

Большие сети получили широкое распространение благодаря их повсеместному применению в веб-приложениях, ориентированных на социальные сети. Графы — это структурные записи, содержащие информацию об узлах и соединяющих их ребрах. Например, каждый участник социальной сети — это узел, а дружественная связь между двумя участниками — это ребро. В данной конкретной ситуации мы рассматриваем случай очень больших сетей наподобие Интернета, социальной или коммуникационной сети. Необходимо вложить эти узлы в векторы признаков таким образом, чтобы граф отражал связи между узлами. Ради простоты мы ограничимся ненаправленными графами, хотя организовать обработку направленных графов со взвешенными ребрами, внося лишь несколько изменений в приведенное ниже описание, не составит большого труда.

Рассмотрим присоединенную матрицу $B = [b_{ij}]$ размера $n \times n$ для графа с n узлами. Элемент b_{ij} равен 1, если узлы i и j соединены ненаправленным ребром. Кроме того, матрица B симметрична, поскольку для ненаправленного графа $b_{ij} = b_{ji}$. Чтобы определить вложение, мы должны определить две матрицы U и V размера $n \times p$ каждая, таких, чтобы матрицу B можно было выразить в виде функции UV^T . В простейшем случае матрица B может быть установлена равной UV^T , что ничем не отличается от метода традиционной матричной факторизации, применяемого для факторизации графов [4]. Однако в случае бинарных матриц можно поступить лучше и использовать вместо этого логистическую матричную факторизацию. Иными словами, каждый элемент матрицы B генерируется с помощью матрицы параметров Бернулли в $f(UV^T)$, где $f(\cdot)$ — сигмоида, применяемая к каждому элементу матрицы, указанной в качестве ее аргумента:

$$f(x) = \frac{1}{1 + \exp(-x)}. \quad (2.61)$$

Поэтому, если \bar{u}_i — i -я строка матрицы U , а \bar{v}_j — j -я строка матрицы V , то имеем следующее соотношение:

$$b_{ij} \sim \text{распределение Бернулли с параметрами } f(\bar{u}_i \cdot \bar{v}_j). \quad (2.62)$$

Этот тип генеративной модели обычно реализуют с использованием модели логарифмического правдоподобия. Кроме того, эту задачу можно переформулировать в виде эквивалентной ей логистической матричной факторизации, применяемой в SGNS-модели word2vec.

Заметим, что модели word2vec — это логистические/мультиномиальные варианты модели, представленной на рис. 2.13, которая транслирует индексы строк в значения с линейной активацией. Чтобы прояснить этот момент, на рис. 2.17 приведена нейронная архитектура небольшого графа, включающего 5 узлов.

Входом является индекс строки матрицы B (т.е. узла) в представлении прямого кодирования, а выходом — список всех значений 0/1 всех узлов сети. В данном конкретном случае мы представили вход для узла 3 и его соответствующий выход. Поскольку узел 3 имеет трех соседей, выходной вектор содержит три компоненты со значением 1. Эта архитектура мало отличается от той, которая представлена на рис. 2.13, за исключением того, что в ее выходном слое используется сигмоидная (а не линейная) активация. Кроме того, поскольку количество нулей на выходе обычно намного превышает⁸ количество единиц, многие нули могут быть отброшены за счет применения отрицательного семплирования. Такой тип отрицательного семплирования создаст ситуацию, аналогичную той, которая представлена на рис. 2.14. Шаги градиентного спуска для этой нейронной архитектуры совпадают с аналогичными шагами для SGNS-модели *word2vec*. Основное отличие состоит в том, что узел может появиться в качестве соседа другого узла самое большее один раз, тогда как слово может появиться в контексте другого слова несколько раз. Разрешение счетчикам ребер принимать произвольные значения устраняет это различие.

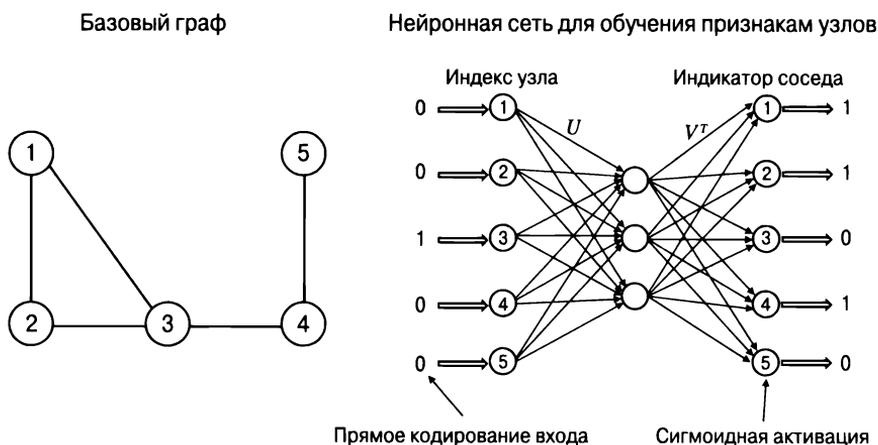


Рис. 2.17. Граф, включающий пять узлов, показан вместе с нейронной архитектурой для отображения индексов строк на индикаторы соседей. Приведенные на рисунке вход и выход представляют узел 3 и его соседей. Обратите внимание на сходство с рис. 2.13. Основное отличие состоит в том, что в данном случае нет отсутствующих значений, а количество входов совпадает с количеством выходов для квадратной матрицы. Как вход, так и выходы — бинарные векторы. Но если вместе с сигмоидной активацией используется отрицательное семплирование, то большинство выходных узлов с нулевыми значениями могут быть опущены

⁸ В небольшом примере, представленном на рис. 2.17, этот факт не сразу очевиден. На практике степень узла графа составляет лишь небольшую долю от общего количества узлов. Например, у человека может быть 100 друзей в социальной сети, насчитывающей миллионы узлов.

2.7.1. Обработка произвольных значений счетчиков ребер

В приведенном выше обсуждении предполагалось, что вес каждого ребра является бинарным (т.е. может иметь значения 0 или 1). Рассмотрим случай, когда с ребром (i, j) связывается произвольное значение счетчика c_{ij} . В этом случае необходимо использовать как положительное, так и отрицательное семплирование. Первый шаг заключается в семплировании ребра (i, j) из сети с вероятностью, пропорциональной c_{ij} . Поэтому входом является вектор узла, находящегося в одной из концевых точек (скажем, i) данного ребра, в представлении *прямого* кодирования. Выходом является аналогичное представление узла j . По умолчанию как вход, так и выход являются n -мерным вектором. Но если используется отрицательное семплирование, то существует возможность снизить размерность выходного вектора до $(k + 1)$. Здесь $k \ll n$ — частота выборки. В общей сложности семплируется k отрицательных узлов с вероятностями, пропорциональными их (взвешенным) степеням⁹, а выходами этих узлов являются нули. Функцию потерь в виде логарифмического правдоподобия можно вычислить, рассматривая каждый выход как исход опыта Бернулли, параметром которого является выходное значение сигмоиды. Градиентный спуск выполняется с использованием этой функции потерь. Этот вариант представляет собой почти полную имитацию варианта SGNS модели *word2vec*.

2.7.2. Мультиномиальная модель

Простая модель скип-грамм *word2vec* — это мультиномиальная модель, которая также может использоваться для создания вложений. Единственное различие состоит в том, что в последнем слое нейронной сети, представленной на рис. 2.17, должна применяться функция активации *Softmax* (а не сигмоида). Кроме того, в мультиномиальной модели не задействуется отрицательное семплирование, а входной и выходной слои содержат ровно по n узлов каждый. Как и в модели SGNS, любое одиночное ребро (i, j) семплируется с вероятностью, пропорциональной c_{ij} , для создания пары “вход — выход”. Вход — это узел i в представлении прямого кодирования, а выход — это узел j , тоже в представлении прямого кодирования. Также можно применять мини-пакетное семплирование ребер для улучшения производительности. Шаги стохастического градиентного спуска в этой модели почти аналогичны шагам, используемым в простой модели скип-грамм *word2vec*.

⁹ Взвешенная степень узла j равна $\sum_r c_{rj}$.

2.7.3. Связь с моделями *DeepWalk* и *node2vec*

Недавно предложенные модели *DeepWalk* [372] и *node2vec* [164] принадлежат к обсуждавшемуся выше семейству мультиномиальных моделей (со специальными шагами предварительной обработки). Основное отличие состоит в том, что в моделях *DeepWalk* и *node2vec* для (косвенного) генерирования c_{ij} используются проходы “сначала в глубину” или “сначала в ширину”. Сама модель *DeepWalk* — это предшественница (и специальный случай) модели *node2vec* в том, что касается выполнения случайных обходов узлов. В данном случае c_{ij} можно интерпретировать как число раз, когда узел j появлялся по соседству с узлом i , поскольку включался в обход “сначала в ширину” или “сначала в глубину”, начинающийся с узла i . В моделях, основанных на обходе узлов, значение c_{ij} может рассматриваться в качестве более надежной оценки родственности узлов i и j , чем простые веса в исходном графе. Разумеется, в отношении использования случайного обхода узлов для повышения надежности c_{ij} не существует никаких табу. Число возможных вариантов выбора способов генерирования значений сродства этого типа ничем не ограничено. Такие значения сродства генерируют все *методы предсказания связей* [295]. Например, *мера Каца* [295], которая тесно связана с числом случайных путей между парой узлов, является надежной мерой сродства узлов i и j .

2.8. Резюме

В этой главе обсуждался ряд нейронных моделей обучения как с учителем, так и без учителя. Одной из целей этого обсуждения являлась демонстрация того, что многие традиционные модели, используемые в машинном обучении, представляют собой реализации относительно простых нейронных моделей. Мы обсудили методы бинарной/мультиклассовой классификации. Кроме того, были приведены примеры применения этого подхода к рекомендательным системам и векторным представлением (вложениям) слов. В случае обобщения традиционной методики машинного обучения, такой, например, как сингулярное разложение, до нейронного представления результат такого обобщения зачастую оказывается менее эффективным по сравнению с его аналогом в традиционном машинном обучении. Однако нейронные модели имеют то преимущество, что обычно их можно обобщить до уровня более мощных нелинейных моделей. Кроме того, использование нейронных сетей упрощает проведение экспериментов с нелинейными вариантами традиционных моделей машинного обучения. В этой главе также обсуждались вопросы практического характера наподобие рекомендательных систем или систем обработки текста и графов.

2.9. Библиографическая справка

Алгоритм перцептрона был предложен Розенблаттом [405], а его подробное обсуждение можно найти в [405]. Алгоритм Уидроу — Хоффа был предложен в [531] и тесно связан с работой Тихонова и Арсенина [499]. Дискриминант Фишера был предложен Роналдом Фишером [120] в 1936 году и является специальным случаем семейства методов линейного дискриминантного анализа [322]. Несмотря на то что дискриминант Фишера использует целевую функцию, на первый взгляд отличающуюся от той, которая применяется в регрессии по методу наименьших квадратов, он представляет собой специальный случай регрессии по методу наименьших квадратов, в котором регрессантом служит бинарная переменная ответа [40]. Подробное обсуждение обобщенных линейных моделей предоставлено в работе [320]. В [178] обсуждается ряд таких процедур, как *обобщенное итеративное масштабирование, метод наименьших квадратов с итеративно обновляемыми весами и градиентный спуск для мультиномиальной логистической регрессии*. Открытие метода опорных векторов (SVM) обычно приписывают Кортесу и Вапнику [82], хотя первоначальный метод SVM с L_2 -регуляризацией был предложен Хинтоном [190] несколькими годами ранее! Этот подход исправляет недостатки функции потерь задачи классификации по методу наименьших квадратов, оставляя лишь одну половину квадратичной функции потерь и задавая остальную ее часть равной нулю, в результате чего она принимает форму сглаженной функции кусочно-линейных потерь (попробуйте сделать это, воспользовавшись рис. 2.4). Значимость этой работы не была оценена по достоинству, поскольку она просто затерялась в потоке литературы по нейронным сетям. В работе Хинтона также не было обращено особое внимание на регуляризацию в SVM, хотя о важности уменьшения шагов градиентного спуска в нейронных сетях было хорошо известно. В [82] SVM с кусочно-линейными потерями была основательно представлена с точки зрения дуализма и понятия максимального зазора, что несколько затушевало ее связь с классификацией по методу наименьших квадратов с регуляризацией. Связь SVM с классификацией по методу наименьших квадратов продемонстрирована более прозрачно в работах [400, 442], результаты которых говорят о том, что SVM с квадратичными и кусочно-линейными потерями являются естественными вариациями классификации с L_2 -регуляризацией (т.е. дискриминанта Фишера) и L_1 -регуляризацией, в которых бинарные переменные классов используются в качестве регрессионных ответов [139]. Мультиклассовая SVM Уэстона — Уоткинса была введена в [529]. Было показано [401], что эффективность обобщения на случай мультиклассовой классификации с использованием стратегии “один против всех” сопоставима с тесно интегрированными мультиклассовыми вариантами. Многие иерархические Softmax-методы обсуждаются в [325, 327, 332, 344].

Отличный обзор методов снижения размерности данных в нейронных сетях приведен в [198], хотя основным предметом ее рассмотрения является использование родственной модели, известной как ограниченная машина Больцмана (restricted Boltzmann machine — RBM). Наиболее раннее упоминание об автокодировщиках (в более общей форме) приведено в статье по обратному распространению ошибки [408]. В этой работе обсуждается задача перекодировки данных между входом и выходом. Используя подходящим образом подобранные структуры входа и выхода, можно показать, что как классификаторы, так и автокодировщики являются специальными случаями этой архитектуры. В указанной статье по обратному распространению ошибки [408] также обсуждается специальный случай, в котором вход перекодируется с помощью тождественного преобразования, что в точности соответствует сценарию автокодировщика. Более подробное обсуждение автокодировщика в первые годы его существования приведено в [48, 275]. Обсуждение однослойной модели обучения без учителя можно найти в [77]. Стандартным методом регуляризации автокодировщика является затухание весов, что соответствует L_2 -регуляризации. Разреженные автокодировщики обсуждаются в [67, 273, 274, 284, 354]. Другим способом регуляризации автокодировщика является штрафование производных в процессе градиентного спуска. Это гарантирует, что обученная функция не изменится слишком сильно при изменении входа. Такой метод называют *контрактивным автокодировщиком* (contractive autoencoder) [397]. Вариационные автокодировщики, которые обсуждаются в [106, 242, 399], могут кодировать сложные вероятностные распределения. Шумоподавляющий автокодировщик обсуждается в [506]. Многие из этих методов подробно обсуждаются в главе 4. Использование автокодировщиков для обнаружения выбросов исследуется в [64, 181, 564], а обзор их использования в кластеризации приведен в [8].

О применении методов снижения размерности в рекомендательных системах можно прочитать в [414], хотя в этом подходе используется ограниченная машина Больцмана, которая отличается от метода матричной факторизации, обсуждавшегося в этой главе. Автокодировщик на основе объектов обсуждается в [436], и этот подход представляет собой нейронный вариант обобщения регрессии по методу ближайших соседей на основе объектов [253]. Основным отличием является то, что регрессионные веса регуляризируются с помощью ограниченного скрытого слоя. Аналогичные работы с различными типами моделей “объект — объект”, в которых используются шумоподавляющие автокодировщики, обсуждаются в [472, 535]. Описание более непосредственного обобщения методов матричной факторизации можно найти в [186], хотя предложенный в ней подход несколько отличается от более простого подхода, представленного в этой главе. Включение содержимого в рекомендательные системы для глубокого обучения обсуждается в [513]. В [110] предложен подход к глубокому

обучению на основе нескольких представлений, который впоследствии [465] был расширен также на временные рекомендательные системы. Обзор методов глубокого обучения для рекомендательных систем приведен в [560].

Модель *word2vec* предложена в [325, 327], а ее подробное изложение вы найдете в [404]. Базовая идея была расширена на векторные представления (вложения) на уровне предложений и абзацев с помощью модели *doc2vec* [272]. Альтернативой *word2vec*, в которой используется другой тип матричной факторизации, является GloVe [371]. Многоязычные вложения представлены в [9]. Расширение *word2vec* на графы с вложениями на уровне узлов предоставляют модели *DeepWalk* [372] и *node2vec* [164]. Различные типы сетей, в которых используются векторные представления, обсуждаются в [62, 512, 547, 548].

2.9.1. Программные ресурсы

Модели машинного обучения наподобие линейной регрессии, SVM и логистической регрессии доступны в библиотеке *scikit-learn* [587]. DISSECT (Distributional Semantics Composition Toolkit) [588] — это набор инструментов, использующий счетчики совместной встречаемости слов для создания вложений (векторных представлений). Метод GloVe предоставляется Стэнфордской группой обработки естественного языка (Stanford NLP) [589] и доступен в библиотеке *gensim* [394]. Инструмент *word2vec* доступен на условиях лицензии Apache [591] и в виде версии *TensorFlow* [592]. Библиотека *gensim* содержит реализации *word2vec* и *doc2vec* для Python [394]. Версии *doc2vec*, *word2vec* и GloVe для Java можно найти в репозитории *DeepLearning4j* [590]. В некоторых случаях можно просто загрузить предварительно обученные (на большом корпусе общего текста, считающемся достаточно представительным) версии представлений и использовать их непосредственно в качестве удобной альтернативы тренировке на тексте, имеющемся под рукой. Программное обеспечение *node2vec* доступно от его автора [593].

2.10. Упражнения

1. Предположим, функция потерь для тренировочной пары (\bar{X}, y) имеет следующий вид:

$$L = \max \{0, \alpha - y(\bar{W} \cdot \bar{X})\}.$$

Результаты тестовых примеров предсказываются как $\hat{y} = \text{sign}\{\bar{W} \cdot \bar{X}\}$. Значение $\alpha = 0$ соответствует критерию перцептрона, а значение $\alpha = 1$ — SVM. Покажите, что любое значение $\alpha > 0$ приводит к SVM с неизменным оптимальным решением, если регуляризация не задействуется. Что произойдет в случае использования регуляризации?

2. Основываясь на результатах упражнения 1, сформулируйте обобщенную целевую функцию для SVM Уэстона — Уоткинса.
3. Рассмотрим нерегуляризируемое обновление перцептрона для бинарных классов со скоростью обучения α . Покажите, что изменение значения α ни на что существенно не влияет в том смысле, что это приводит лишь к масштабированию вектора весов с коэффициентом α . Покажите, что этот результат остается справедливым также для мультиклассового случая. Остаются ли результаты такими же в случае использования регуляризации?
4. Покажите, что в случае применения SVM Уэстона — Уоткинса к набору данных с $k = 2$ классами результирующие обновления эквивалентны бинарным обновлениям SVM, которые обсуждались в этой главе.
5. Покажите, что в случае применения мультиномиальной логистической регрессии к набору данных с $k = 2$ классами результирующие обновления эквивалентны обновлениям логистической регрессии.
6. Реализуйте классификатор Softmax, используя библиотеку глубокого обучения, с которой вы работаете.
7. В моделях ближайших соседей на основе линейной регрессии рейтинговая оценка объекта предсказывается в виде взвешенной комбинации оценок других объектов, оцененных тем же пользователем, где специфические для объектов веса обучаются с помощью линейной регрессии. Покажите, какую архитектуру автокодировщика вы использовали бы для создания модели этого типа. Обсудите связь этой архитектуры с архитектурой матричной факторизации.
8. **Логистическая матричная факторизация.** Рассмотрим автокодировщик, в котором имеется входной слой, один скрытый слой, содержащий представление пониженной размерности, и выходной слой с линейной активацией.
 - А. Задайте функцию потерь на основе отрицательного логарифма правдоподобия для случая, когда известно, что матрица входных данных содержит бинарные значения из набора $\{0, 1\}$.
 - Б. Задайте функцию потерь на основе отрицательного логарифма правдоподобия для случая, когда известно, что матрица входных данных содержит вещественные значения из интервала $[0, 1]$.
9. **Неотрицательная матричная факторизация с автокодировщиками.** Пусть D — матрица данных размера $n \times d$ с неотрицательными элементами. Покажите, как бы вы аппроксимировали матрицу D , факторизовав ее в виде $D \approx UV^T$, где U и V — две неотрицательные матрицы, используя архитектуру автокодировщика с d входами и выходами. (Подсказка: вы-

берите подходящую функцию активации для скрытого слоя и измените правило обновления для градиентного спуска.)

10. **Вероятностный латентный семантический анализ.** Определение вероятностного латентного семантического анализа дано в [99, 206]. Предложите видоизменение подхода, используемого в упражнении 9, которое можно было бы применить для вероятностного латентного семантического анализа. (Подсказка: какова связь между факторизацией неотрицательной матрицы и вероятностным латентным семантическим анализом?)
11. **Имитация комбинированного ансамбля моделей.** В машинном обучении комбинированный ансамбль моделей усредняет оценки многих узлов для создания более надежных классификационных оценок. Подумайте, как можно аппроксимировать усреднение линейного элемента Adaline и логистической регрессии с помощью двухслойной нейронной сети. Проанализируйте, чем отличается и в чем сходна эта архитектура с фактическим комбинированным ансамблем моделей, если для ее тренировки используется обратное распространение ошибки. Покажите, как следует видоизменить процесс тренировки, чтобы окончательный результат представлял собой тонкую настройку комбинированного ансамбля моделей.
12. **Имитация стекового ансамбля моделей.** В машинном обучении стековый ансамбль создает модели поверх признаков, обученных с помощью классификаторов первого уровня. Подумайте, как можно видоизменить архитектуру, рассмотренную в упражнении 11, чтобы классификаторы первого уровня соответствовали классификатору на основе Adaline и логистической регрессии, а классификатор более высокого уровня соответствовал методу опорных векторов. Выясните сходства и отличия этой архитектуры от фактического стекового ансамбля, если для ее тренировки используется обратное распространение ошибки. Покажите, как следует видоизменить процесс тренировки нейронной сети, чтобы окончательный результат представлял собой тонкую настройку стекового ансамбля.
13. Покажите, что обновления стохастического градиентного спуска для перцептрона, метода обучения Уидроу — Хоффа, SVM и логистической регрессии имеют вид $\bar{W} \leftarrow \bar{W}(1 - \alpha\lambda) + \alpha y[\delta(\bar{X}, y)]\bar{X}$. Здесь функция ошибки $\delta(\bar{X}, y)$ равна $1 - y(\bar{W} \cdot \bar{X})$ в случае классификации по методу наименьших квадратов, индикаторной переменной в случае перцептрона/SVM и вероятностному значению в случае логистической регрессии. Предположите, что α — скорость обучения, а $y \in \{-1, +1\}$. Запишите конкретные формы $\delta(\bar{X}, y)$ для каждого случая.
14. Линейный автокодировщик, который обсуждался в этой главе, применяется к каждой d -мерной строке набора данных D размером $n \times d$ для

создания k -мерного представления. Веса кодировщика содержатся в матрице W размера $k \times d$, а веса декодировщика — в матрице V размера $d \times k$. Поэтому реконструированное представление имеет вид $DW^T V^T$, и агрегированное значение потерь $\|DW^T V^T - D\|^2$ минимизируется по всему тренировочному набору данных.

- А.** Покажите, что для фиксированного значения V оптимальная матрица W должна удовлетворять соотношению $D^T D(W^T V^T V - V) = 0$.
- Б.** Используя результат *А*, покажите, что если матрица D размером $n \times d$ имеет ранг d , то $W^T V^T V = V$.
- В.** Используя результат *Б*, покажите, что $W = (V^T V)^{-1} V^T$. Предположите, что матрица $V^T V$ обратима.
- Г.** Повторите пп. *А–В* упражнения для случая, когда веса кодировщика и декодировщика связаны соотношением $W = V^T$. Покажите, что столбцы V должны быть ортонормированными.

Глава 3

Обучение глубоких нейронных сетей

Я проклинал каждую минуту тренировки, но говорил себе: не сдавайся. Потерпи сейчас — и проживешь всю жизнь чемпионом.

Мухаммед Али

3.1. Введение

Процедура тренировки нейронной сети с использованием алгоритма обратного распространения ошибки была кратко описана в главе 1. В данной главе это описание расширено в нескольких направлениях.

1. Наряду с более подробным рассмотрением алгоритма обратного распространения ошибки мы уделим внимание деталям реализации. Часть материала из главы 1 будет обсуждаться повторно ради полноты изложения, чтобы читателю не пришлось слишком часто возвращаться к предыдущим разделам.
2. В этой главе будут затронуты важные темы, касающиеся предварительной обработки и инициализации признаков.
3. Вы познакомитесь с процедурами, используемыми совместно с градиентным спуском. Будет изучено влияние глубины сети на стабильность процесса тренировки, а также будут представлены методы, обеспечивающие эту стабильность.
4. Мы обсудим вопросы, связанные с эффективностью тренировки. Будут представлены методы сжатия обученных моделей. Эти методы особенно полезны для развертывания предварительно обученных сетей на мобильных устройствах.

В прежние годы о методах обучения многослойных сетей ничего не было известно. В своей основополагающей книге [330] Минский и Пейперт весьма скептически оценили перспективы развития нейронных сетей, поскольку

тренировка многослойных сетей в то время представлялась невозможной. Поэтому интерес к изучению нейронных сетей был утерян вплоть до 80-х годов прошлого столетия. Первый значительный прорыв в этом направлении был совершен Румельхартом и др. [408, 409], предложившими¹ алгоритм обратного распространения ошибки. Это предложение возродило интерес к нейронным сетям. Однако использование данного алгоритма натолкнулось на ряд трудностей вычислительного характера, не говоря уже о проблемах стабильности процесса обучения и эффектах переобучения. Как следствие, исследования в этой области вновь отошли на задний план.

На рубеже столетий нейронные сети в очередной раз обрели популярность благодаря определенным достижениям. Не все из них были связаны с алгоритмами. Главную роль в возрождении интереса к нейронным сетям сыграло увеличение объема доступных данных и вычислительных мощностей. Но этому также сопутствовало внесение некоторых изменений в базовый алгоритм обратного распространения ошибки и разработка эффективных методов инициализации параметров сети, таких как *предварительное обучение* (pretraining). Кроме того, достигнутое в последние годы сокращение длительности циклов тестирования (обусловленное развитием вычислительной техники) облегчило проведение ресурсоемких экспериментов, необходимых для настройки алгоритмов. Поэтому расширение доступа к данным, улучшение вычислительных возможностей и сокращение длительности экспериментов (для тонкой настройки алгоритмов) шли рука об руку. Так называемая “тонкая настройка” играет очень важную роль; большинство этих очень важных алгоритмических усовершенствований будет рассмотрено в данной и следующей главах.

Ключевое значение имеет тот факт, что алгоритм обратного распространения ошибки ведет себя довольно *нестабильно* по отношению к незначительным изменениям алгоритмических установок, таких как начальная точка, используемая при таком подходе. Эта нестабильность особенно проявляется при работе с очень глубокими сетями. Стоит подчеркнуть, что оптимизация нейронной сети — это *задача оптимизации со многими переменными*. Такими переменными являются веса соединений в различных слоях. В задачах оптимизации со

¹ Несмотря на то что алгоритм обратного распространения ошибки был популяризирован в статьях Румельхарта и его соавторов [408, 409], он был исследован еще раньше в контексте теории управления. В незаслуженно забытой (и в конечном счете заново открытой) кандидатской диссертации Пола Вербоса, написанной им еще в 1974 году, обсуждались возможные способы использования методов обратного распространения ошибки в нейронных сетях. Это было задолго до появления в 1986 году статей Румельхарта, которые тем не менее сыграли значительную роль в силу того, что стиль изложения материала способствовал лучшему пониманию того, почему обратное распространение ошибки может работать в нейронных сетях.

многими переменными причиной частого возникновения проблем нестабильности является то, что шаги в различных направлениях должны совершаться с соблюдением “правильных” пропорций. Обеспечить это в области нейронных сетей особенно трудно, так что результаты шагов градиентного спуска могут быть довольно непредсказуемыми. Одна из проблем заключается в том, что *градиент определяет скорость изменения в каждом направлении лишь в пределах бесконечно малой окрестности текущей точки*, тогда как фактические шаги имеют конечные размеры. Чтобы оптимизация была действительно успешной, размер шагов должен тщательно подбираться. Трудность состоит в том, что в случае шагов конечной длины сами градиенты испытывают изменения, которые иногда достигают значительной величины. В этом отношении особенно предательски ведут себя сложные оптимизационные поверхности нейронных сетей, и неудачная организация процесса оптимизации (например, выбор начальной точки или способа нормализации входных признаков) лишь усугубляет эту проблему. Как следствие, легко вычисляемое направление скорейшего спуска зачастую является далеко не лучшим направлением, вдоль которого можно двигаться большими шагами. Использование шагов небольшого размера замедляет процесс оптимизации, тогда как использование шагов большего размера может приводить к непредсказуемым результатам. Все эти проблемы делают процесс оптимизации нейронных сетей более сложным, чем могло бы показаться на первый взгляд. В то же время многих из проблем удастся избежать, тщательно выбирая шаги градиентного спуска сообразно с природой оптимизационной поверхности. Рассмотрению соответствующих алгоритмов посвящена данная глава.

Структура главы

В следующем разделе будет дан обзор алгоритма обратного распространения ошибки, первоначально обсуждавшегося в главе 1. В этой главе он обсуждается более подробно, причем будет рассмотрено несколько его вариантов. Некоторые части алгоритма, которые уже обсуждались в главе 1, будут повторно рассмотрены в данной главе, чтобы содержащийся в ней материал был самодостаточным. Вопросам предварительной обработки и инициализации признаков посвящен раздел 3.3. Проблемы исчезающих и затухающих градиентов, с которыми обычно приходится сталкиваться в глубоких сетях, обсуждаются в разделе 3.4, где также будут представлены наиболее часто используемые способы решения этих проблем. Стратегии градиентного спуска для глубокого обучения обсуждаются в разделе 3.5. В разделе 3.6 читатели познакомятся с методами пакетной нормализации. В разделе 3.7 обсуждаются ускоренные варианты реализации нейронных сетей. Резюме главы приведено в разделе 3.8.

3.2. Алгоритм обратного распространения ошибки: подробное рассмотрение

В данном разделе мы вновь проанализируем алгоритм обратного распространения ошибки, но уже гораздо подробнее, чем это было сделано в главе 1. Наша цель — продемонстрировать, что цепное правило дифференцирования можно использовать несколькими способами. Для этого мы сначала исследуем стандартные правила обновления при обратном распространении ошибки в том виде, в каком их обычно представляют в большинстве учебников (как и в главе 1). Далее будет рассмотрено упрощенное представление обратного распространения ошибки, в котором линейные операции матричного умножения отделены от активационных слоев. Именно в таком представлении алгоритм реализуется в большинстве готовых систем.

3.2.1. Анализ обратного распространения ошибки с помощью абстракции вычислительного графа

Нейронная сеть — это *вычислительный граф*, элементами которого являются нейроны. Нейронные сети обладают существенно большими возможностями по сравнению со строительными блоками, из которых они состоят, поскольку *совместное* обучение параметров моделей обеспечивает создание в высшей степени оптимизированной композитной функции модели. Кроме того, использование нелинейных активаций между различными слоями дополнительно повышает выразительную мощь сети. Многослойная сеть вычисляет композитные функции путем вычисления индивидуальных функций узлов. Путь длиной 2 в нейронной сети, в которой за функцией $f(\cdot)$ следует функция $g(\cdot)$, можно рассматривать как композитную функцию $f(g(\cdot))$. Чтобы пояснить эту мысль, рассмотрим тривиальный вычислительный граф с двумя узлами, в каждом из которых ко входу с весом w применяется сигмоида. В этом случае вычисляемая функция принимает следующий вид:

$$f(g(w)) = \frac{1}{1 + \exp\left[-\frac{1}{1 + \exp(w)}\right]}. \quad (3.1)$$

Уже сейчас мы видим, насколько неудобно было бы вычислять производную этой функции по w . Кроме того, рассмотрим случай, когда в слое m вычисляются функции $g_1(\cdot)$, $g_2(\cdot)$, ..., $g_k(\cdot)$, и эти функции передаются некоторому узлу в слое $(m + 1)$, который вычисляет функцию $f(\cdot)$. В таком случае композитной функцией, вычисляемой узлом $(m+1)$ -го слоя в терминах входов слоя m , является сложная функция $f(g_1(\cdot) \dots g_k(\cdot))$. Как видите, эта композитная функция в многомерном представлении выглядит довольно неуклюже. Поскольку

функция потерь использует выходы в качестве своих аргументов, ее, как правило, можно выразить в виде рекурсивно вложенных функций весов, относящихся к предыдущим слоям. В случае нейронной сети с 10 слоями и всего лишь 2 узлами в каждом слое, в которой глубина рекурсивного вложения функций равна 10, это приведет к необходимости обработки 2^{10} рекурсивно вложенных членов в процессе вычисления частных производных. Поэтому для вычисления этих производных требуется привлечение некоего итеративного подхода. Таким подходом является *динамическое программирование*, а соответствующие обновления в действительности следуют *цепному правилу дифференциального исчисления*.

Чтобы понять, как работает цепное правило в вычислительном графе, обсудим два базовых варианта этого правила, которые следует знать. Простейшая версия цепного правила применяется к прямой композиции функций:

$$\frac{\partial f(g(w))}{\partial w} = \frac{\partial f(g(w))}{\partial g(w)} \cdot \frac{\partial g(w)}{\partial w}. \quad (3.2)$$

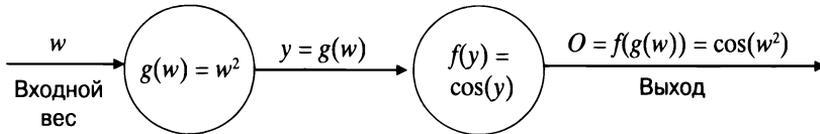
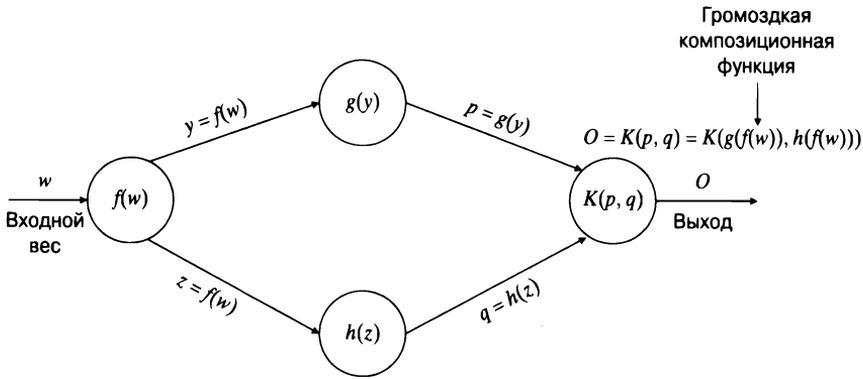


Рис. 3.1. Простой вычислительный граф с двумя узлами

Этот вариант называется *одномерным цепным правилом* (univariate chain rule). Обратите внимание на то, что каждый член справа от знака равенства представляет собой *локальный градиент*, поскольку в нем производная функции вычисляется по ее непосредственному аргументу, а не по аргументу, извлекаемому в процессе рекурсии. Основная идея заключается в том, что применение композиции функций ко входному весу w дает конечный выход, тогда как градиент конечного выхода дается произведением локальных градиентов вдоль этого пути.

При вычислении каждого локального градиента необходимо знать лишь его конкретные вход и выход, что упрощает вычислительный процесс. На рис. 3.1 приведен пример, в котором $f(y) = \cos(y)$, а $g(w) = w^2$. Поэтому композитная функция — это $\cos(w^2)$. Используя одномерное цепное правило, получаем следующее соотношение:

$$\frac{\partial f(g(w))}{\partial w} = \underbrace{\frac{\partial f(g(w))}{\partial g(w)}}_{-\sin(g(w))} \cdot \underbrace{\frac{\partial g(w)}{\partial w}}_{2w} = -2w \cdot \sin(w^2).$$



$$\begin{aligned}
 \frac{\partial o}{\partial w} &= \frac{\partial o}{\partial p} \cdot \frac{\partial p}{\partial w} + \frac{\partial o}{\partial q} \frac{\partial q}{\partial w} = && \text{[цепное правило, несколько переменных]} \\
 &= \frac{\partial o}{\partial p} \cdot \frac{\partial p}{\partial y} \cdot \frac{\partial y}{\partial w} + \frac{\partial o}{\partial q} \cdot \frac{\partial q}{\partial z} \cdot \frac{\partial z}{\partial w} = && \text{[цепное правило, сведение к одной переменной]} \\
 &= \underbrace{\frac{\partial K(p, q)}{\partial p} \cdot g'(y) \cdot f'(w)}_{\text{Первый путь}} + \underbrace{\frac{\partial K(p, q)}{\partial q} \cdot h'(z) \cdot f'(w)}_{\text{Второй путь}}.
 \end{aligned}$$

Рис. 3.2. Вариант рис. 1.13 применительно к вычислительным графам. Произведения специфических для узлов частных производных вдоль путей от входа с весом w к выводу o агрегируются. Результирующее значение дает производную выхода O по весу w . В этом упрощенном примере существуют лишь два пути между входом и выходом

Вычислительные графы в нейронных сетях не являются путями, что служит основной причиной, диктующей потребность в использовании алгоритма обратного распространения ошибки. Скрытый слой часто получает свой вход сразу от нескольких элементов, что приводит к возникновению нескольких путей от переменной w к выводу. Рассмотрим функцию $f(g_1(w) \dots g_k(w))$, в которой элемент, вычисляющий многомерную функцию $f(\cdot)$, получает свои входы от k элементов, вычисляющих функции $g_1(w) \dots g_k(w)$. В подобных случаях необходимо использовать многомерное цепное правило (multivariable chain rule), которое определяется следующим образом:

$$\frac{\partial f(g_1(w) \dots g_k(w))}{\partial w} = \frac{\partial f(g_1(w), \dots, g_k(w))}{\partial g_i(w)} \cdot \frac{\partial g_i(w)}{\partial w}. \quad (3.3)$$

Нетрудно увидеть, что многомерное цепное правило, приведенное в уравнении 3.3, есть не что иное, как обобщение правила, приведенного в уравнении 3.2. Важным следствием многомерного цепного правила является следующая лемма.

Лемма 3.2.1 (об агрегации вдоль путей). Рассмотрим направленный ациклический граф, в i -м узле которого содержится переменная $y(i)$. Локальная производная $z(i, j)$ для направленного ребра (i, j) графа определяется как $z(i, j) = \frac{\partial y(j)}{\partial y(i)}$. Предположим, существует ненулевое множество \mathcal{P} путей, ведущих в графе от переменной w к выходному узлу, содержащему переменную o . Тогда значение производной $\frac{\partial o}{\partial w}$ можно получить, вычислив произведение локальных градиентов вдоль каждого пути из множества \mathcal{P} и просуммировав эти произведения по всем путям:

$$\frac{\partial o}{\partial w} = \sum_{P \in \mathcal{P}} \prod_{(i, j) \in P} z(i, j). \quad (3.4)$$

В справедливости этой леммы легко убедиться, рекурсивно применяя уравнение 3.3. И хотя лемма 3.2.1 не используется где-либо в алгоритме обратного распространения ошибки, она позволяет разработать другой, экспоненциальный алгоритм явного вычисления производных. Такая точка зрения облегчает интерпретацию многомерного цепного правила как рекурсии динамического программирования для расчета величины, вычисление которой другими способами было бы слишком затратным. Рассмотрим пример, приведенный на рис. 3.2. В данном конкретном случае существуют два пути. В этом примере также демонстрируется применение цепного правила. Очевидно, что конечный результат получается путем вычисления произведений локальных градиентов вдоль каждого из этих двух путей и последующего их суммирования. На рис. 3.3 приведен более конкретный пример функции, которая вычисляется с помощью того же вычислительного графа.

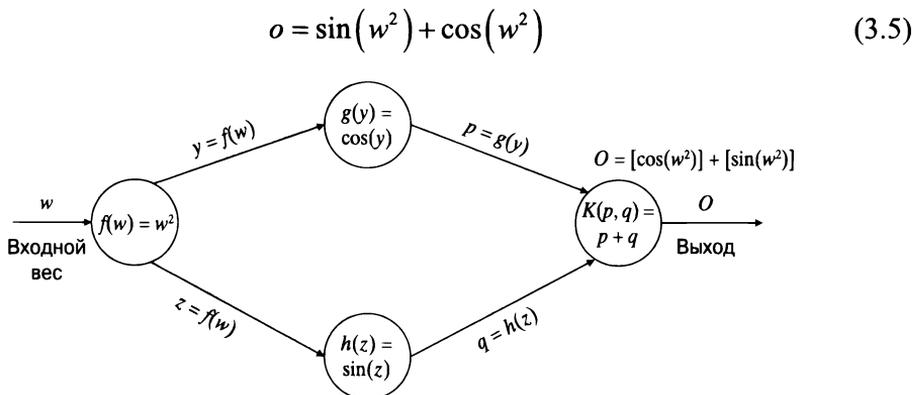


Рис. 3.3. Пример применения цепного правила на основании вычислительного графа, приведенного на рис. 3.2

На рис. 3.3 видно, что применение цепного правила в вычислительном графе позволяет корректно вычислить производную, равную $-2w \cdot \sin(w^2) + 2w \cdot \cos(w^2)$.

Экспоненциальный по времени алгоритм

Тот факт, что мы можем вычислить производную сложной функции как агрегацию произведений локальных производных вдоль путей вычислительного графа, приводит к следующему алгоритму экспоненциальной по времени выполнения сложности.

1. Использовать вычислительный граф для вычисления значения $y(i)$ каждого узла i в фазе прямого распространения.
2. Вычислить локальные частные производные $z(i, j) = \frac{\partial y(j)}{\partial y(i)}$ на каждом ребре вычислительного графа.
3. Пусть \mathcal{P} — множество всех путей, ведущих от входного узла со значением w к выходу. Для каждого $P \in \mathcal{P}$ вычислить произведение всех локальных производных $z(i, j)$ вдоль этого пути.
4. Суммировать эти значения по всем путям в \mathcal{P} .

В общем случае количество путей в вычислительном графе будет экспоненциально расти с увеличением глубины, а произведения локальных производных необходимо суммировать по всем путям. На рис. 3.4 приведен пример, в котором имеется пять слоев, каждый из которых содержит всего лишь по два элемента. Поэтому количество путей, ведущих от входа к выходу, составляет $2^5 = 32$. Элемент j слоя i обозначен как $h(i, j)$. Каждый скрытый элемент определяется в виде произведения своих входов:

$$h(i, j) = h(i-1, 1) \cdot h(i-1, 2) \quad \forall j \in \{1, 2\}. \quad (3.6)$$

В данном примере выход, равный w^{32} , можно выразить в замкнутой форме и легко продифференцировать по w . Однако мы используем экспоненциальный алгоритм, чтобы пояснить, как работают алгоритмы данного типа. Производные каждого $h(i, j)$ по каждому из его двух входов представляют собой значения комплементарных входов:

$$\frac{\partial h(i, j)}{\partial h(i-1, 1)} = h(i-1, 2), \quad \frac{\partial h(i, j)}{\partial h(i-1, 2)} = h(i-1, 1).$$

Согласно лемме об агрегации вдоль путей значение производной $\frac{\partial o}{\partial w}$ представляет собой произведение локальных производных (которые в данном случае равны значениям комплементарных входов) вдоль всех 32 путей, ведущих от входа к выходу:

$$\begin{aligned} \frac{\partial o}{\partial w} &= \sum_{j_1, j_2, j_3, j_4, j_5 \in \{1, 2\}^5} \underbrace{h(1, j_1)}_w \underbrace{h(2, j_2)}_{w^2} \underbrace{h(3, j_3)}_{w^4} \underbrace{h(4, j_4)}_{w^8} \underbrace{h(5, j_5)}_{w^{16}} = \\ &= \sum_{\text{По всем 32 путям}} w^{31} = 32w^{31}. \end{aligned}$$

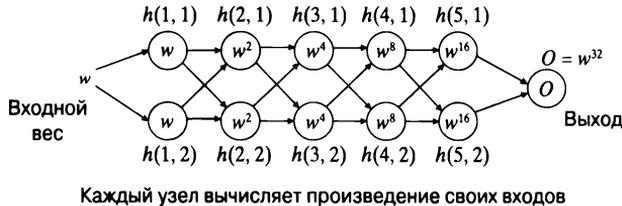


Рис. 3.4. Количество путей в вычислительном графе экспоненциально растет с увеличением глубины. В данном случае в соответствии с ценным правилом произведения локальных производных будут агрегироваться вдоль $2^5 = 32$ путей

Разумеется, этот результат совпадает с тем, который можно было бы получить непосредственным дифференцированием функции w^{32} по w . Важно, однако, то, что для вычисления производной таким способом в случае относительно простого графа требуется 2^5 агрегаций. Еще важнее то, что мы *повторно дифференцируем одну и ту же функцию, вычисляемую в каждом узле в целях агрегации*.

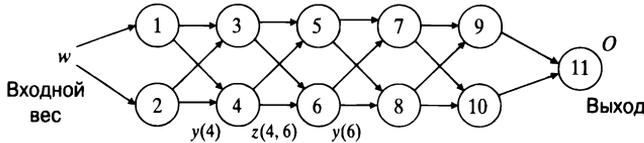
Неэффективность такого подхода к вычислению градиентов очевидна. В случае сети с тремя слоями, каждый из которых содержит по 100 узлов, мы будем иметь дело с миллионом путей. *Тем не менее это именно то, что мы делаем в традиционном машинном обучении, когда предсказываемая функция является сложной композитной функцией*. Это также объясняет, почему большинство традиционных методов машинного обучения эквивалентно мелкой нейронной модели (см. главу 2). Начиная с определенного уровня сложности выписывать вручную все необходимые выражения для сложной композитной функции слишком трудоемко и непрактично. И здесь идея динамического программирования относительно обратного распространения ошибки наводит порядок в хаосе, позволяя создавать модели, реализовать которые иными способами было бы невозможно.

3.2.2. На выручку приходит динамическое программирование

Несмотря на то что в суммировании, которое мы обсуждали выше, участвует экспоненциальное число компонент (путей), динамическое программирование обеспечивает эффективное вычисление данной суммы. В теории графов вычисление всех типов значений, агрегируемых вдоль путей, осуществляется с помощью динамического программирования. Рассмотрим направленный

ациклический граф, в котором значение $z(i, j)$ (интерпретируемое как локальная частная производная переменной в узле j по переменной в узле i) ассоциируется с ребром (i, j) . Пример такого вычислительного графа приведен на рис. 3.5. Мы хотим вычислить произведение $z(i, j)$ по каждому из путей $P \in \mathcal{P}$, ведущих от узла-источника w к выходу o , а затем сложить результаты.

$$\sum_{P \in \mathcal{P}} \prod_{(i, j) \in P} z(i, j). \quad (3.7)$$



Каждый узел i содержит $y(i)$, а каждое ребро между i и j содержит $z(i, j)$.
Пример: $z(4, 6) =$ частная производная $y(6)$ по $y(4)$

Рис. 3.5. Пример вычислительного графа с ребрами, соответствующими локальным частным производным

Пусть $A(i)$ — множество узлов в конечных точках ребер, исходящих из узла i . Агрегированное значение $S(i, o)$ для каждого промежуточного (находящегося между узлами w и o) узла i можно вычислить, используя следующее хорошо известное правило обновления в динамическом программировании:

$$S(i, o) \leftarrow \sum_{j \in A(i)} S(j, o) z(i, j). \quad (3.8)$$

Это выражение может быть вычислено в обратном направлении, начиная с узлов, непосредственно предшествующих узлу o , поскольку уже известно, что $S(o, o)$ равно 1. Обсуждавшийся выше алгоритм относится к числу наиболее широко используемых методов вычисления всех типов функций, основанных на путях в направленных ациклических графах, что в любом другом случае потребовало бы экспоненциального времени выполнения. Например, вариацию этого алгоритма можно использовать даже для нахождения самого длинного пути в направленном ациклическом графе (что, как известно, является *NP-сложной* задачей в случае обычных графов с циклами) [7]. Этот общий подход динамического программирования интенсивно применяется в направленных ациклических графах.

В действительности вышеупомянутое правило обновления динамического программирования в точности представляет собой многомерное цепное правило (уравнение 3.3), применяемое в обратном направлении, начиная с выходного узла с известным локальным градиентом. Это объясняется прежде всего тем, что выражение для градиента функции потерь в форме, агрегируемой вдоль путей (лемма 3.2.1), было получено нами с использованием цепного правила.

Главное отличие состоит в том, что мы применяем это правило в определенном направлении для минимизации объема вычислений. Этому можно дать следующую краткую формулировку:

Использование динамического программирования для эффективной агрегации произведений локальных градиентов вдоль экспоненциального множества путей в вычислительном графе приводит к обновлению в рамках динамического программирования, идентичному многомерному цепному правилу дифференциального исчисления.

Приведенное выше обсуждение относится к случаю типичных вычислительных графов. Как применить эти идеи к нейронным сетям? В случае нейронных сетей можно легко вычислить производную $\frac{\partial L}{\partial o}$ в терминах известного значения o (пропустив вход через сеть). Эта производная распространяется в обратном направлении с использованием локальных частных производных $z(i, j)$, в зависимости от того, какие переменные используются в нейронной сети в качестве промежуточных. Например, если узлами вычислительного графа считать постактивационные значения в узлах, то значение $z(i, j)$ будет представлять собой произведение веса ребра (i, j) и локальной производной функции активации в узле j . С другой стороны, если в качестве узлов вычислительного графа использовать преактивационные значения, то значение $z(i, j)$ будет представлять собой произведение локальной производной функции активации в узле i и веса ребра (i, j) . Понятия пред- и постактивационных переменных в нейронных сетях мы обсудим немного позже на конкретном примере (рис. 3.6). Можно даже создать вычислительный граф, содержащий как пред-, так и постактивационные переменные, чтобы отделить линейные операции от активационных функций. Все эти методы эквивалентны и будут обсуждаться в последующих разделах.

3.2.3. Обратное распространение ошибки с постактивационными переменными

В этом разделе реализация вышеупомянутого подхода демонстрируется на примере вычислительного графа, узлы которого содержат постактивационные переменные нейронной сети. Это те же переменные, что и скрытые переменные в различных слоях.

Алгоритм обратного распространения ошибки сначала использует *прямую фазу* для вычисления выхода и потерь. Поэтому прямая фаза инициализирует переменные для рекурсии динамического программирования, а также промежуточные переменные, которые потребуются для обратной фазы. Как обсуждалось в предыдущем разделе, обратная фаза использует рекурсию динамического программирования на основании многомерного цепного правила дифференцирования.

Прямая фаза. Во время прямой фазы конкретный входной вектор используется для вычисления значений каждого скрытого слоя на основании текущих значений весов. Название “прямая фаза” используется по той причине, что в данном случае вычисления естественным образом каскадируются в прямом направлении через все слои. Цель прямой фазы — вычислить значения всех промежуточных скрытых и выходных переменных для данного входа. Эти значения потребуются во время выполнения обратной фазы. В той точке, в которой вычисления заканчиваются, вычисляется значение выхода o , а также производная функции потерь L . Обычно при наличии нескольких узлов функция потерь является функцией всех выходов, поэтому производные вычисляются по всем выходам. Пока что для простоты ограничимся случаем одного выходного узла, а затем обсудим непосредственное обобщение решения на случай нескольких выходов.

Обратная фаза. Вычисляется градиент функции потерь по различным весам. Первый шаг состоит в том, чтобы вычислить производную $\frac{\partial L}{\partial o}$. Если сеть имеет несколько выходов, то это значение рассчитывается для каждого из них. Тем самым настраивается инициализация вычислений градиентов. После этого производные распространяются в обратном направлении с использованием многомерного цепного правила (см. уравнение 3.3).

Рассмотрим путь, образованный последовательностью скрытых элементов h_1, h_2, \dots, h_k , за которой следует выход o . Вес соединения, ведущего от скрытого элемента h_r к элементу h_{r+1} , обозначим как $w_{(h_r, h_{r+1})}$. Если бы в сети существовал только один такой путь, то распространить в обратном направлении производную функции потерь L по весам вдоль этого пути не составило бы никакого труда. В большинстве случаев в сети существует экспоненциально большое количество путей от любого узла h_r к выходному узлу o . В соответствии с леммой 3.2.1 мы можем вычислить частную производную, агрегируя произведения частных производных по всем путям, ведущим от h_r к o . Если существует множество \mathcal{P} путей от h_r к o , то функцию потерь можно записать в следующем виде:

$$\frac{\partial L}{\partial w_{(h_{r-1}, h_r)}} = \frac{\partial L}{\partial o} \cdot \underbrace{\left[\sum_{[h_r, h_{r+1}, \dots, h_k, o] \in \mathcal{P}} \frac{\partial o}{\partial h_k} \prod_{i=r}^{k-1} \frac{\partial h_{i+1}}{\partial h_i} \right]}_{\text{Алгоритм обратного распространения}} \cdot \frac{\partial h_r}{\partial w_{(h_{r-1}, h_r)}}. \quad (3.9)$$

Алгоритм обратного распространения вычисляет $\Delta(h_r, o) = \frac{\partial L}{\partial h_r}$

Вычисление значения $\frac{\partial h_r}{\partial w_{(h_{r-1}, h_r)}}$, фигурирующего справа от знака равенства,

используется для преобразования рекурсивно вычисленной частной производной по *активациям слоя* в частную производную по *весам*. Агрегированный по путям член (обозначен выше как $\Delta(h_r, o) = \frac{\partial L}{\partial h_r}$) очень напоминает величину $S(i, o) = \frac{\partial o}{\partial y_i}$, о которой шла речь в разделе 3.2.2. Как и в указанном разделе, идея состоит в том, чтобы сначала вычислить $\Delta(h_k, o)$ для узлов h_k , ближайших к o , а затем вычислить рекурсивно эти значения для узлов в предыдущих слоях, выразив их через значения в последующих слоях. Значение $\Delta(o, o) = \frac{\partial L}{\partial o}$ вычисляется в качестве начальной точки рекурсии. Далее это вычисление распространяется в обратном направлении с помощью обновлений динамического программирования (аналогично уравнению 3.8). Рекурсия для $\Delta(h_r, o)$ предоставляется непосредственно многомерным цепным правилом:

$$\Delta(h_r, o) = \frac{\partial L}{\partial h_r} = \sum_{h: h_r \Rightarrow h} \frac{\partial L}{\partial h} \frac{\partial h}{\partial h_r} = \sum_{h: h_r \Rightarrow h} \frac{\partial h}{\partial h_r} \Delta(h, o). \quad (3.10)$$

Поскольку каждый член h относится к более позднему слою, чем h_r , то к моменту вычисления величины $\Delta(h_r, o)$ величина $\Delta(h, o)$ оказывается уже вычисленной. Однако для вычисления уравнения 3.10 нам все еще нужно вычислить производную $\frac{\partial h}{\partial h_r}$. Рассмотрим ситуацию, когда ребро, соединяющее элементы h_r и h , имеет вес $w_{(h_r, h)}$, и пусть a_h — значение, вычисленное в скрытом элементе h непосредственно перед применением функции активации $\Phi(\cdot)$. Иными словами, имеем $h = \Phi(a_h)$, где a_h — линейная комбинация входов, полученных от элементов предыдущего слоя. Тогда в соответствии с одномерным цепным правилом получаем для производной $\frac{\partial h}{\partial h_r}$ следующее выражение:

$$\frac{\partial h}{\partial h_r} = \frac{\partial h}{\partial a_h} \cdot \frac{\partial a_h}{\partial h_r} = \frac{\partial \Phi(a_h)}{\partial a_h} \cdot w_{(h_r, h)} = \Phi'(a_h) \cdot w_{(h_r, h)}. \quad (3.11)$$

Это значение $\frac{\partial h}{\partial h_r}$ используется в уравнении 3.10 для получения следующего соотношения:

$$\Delta(h_r, o) = \sum_{h: h_r \Rightarrow h} \Phi'(a_h) \cdot w_{(h_r, h)} \cdot \Delta(h, o). \quad (3.12)$$

Описанная рекурсия повторяется в обратном направлении, начиная с выходного узла. Весь процесс линеен относительно количества ребер в сети. Обратите внимание на то, что уравнение 3.12 также можно было бы получить,

используя общий алгоритм вычислительного графа, приведенный в разделе 3.2.2, по отношению к постактивационным переменным. Для этого необходимо всего лишь заменить $z(i, j)$ в уравнении 3.8 произведением веса связи между узлами i и j и производной функции активации в узле j .

Процедуру обратного распространения ошибки можно резюмировать в следующем виде.

1. Использовать фазу прямого распространения для вычисления значений всех скрытых элементов, выхода o и функции потерь L для конкретной пары “вход — выход” (\bar{X}, y) .
2. Инициализировать $\Delta(o, o)$ значением $\frac{\partial L}{\partial o}$.
3. Использовать рекурсию 3.12 для вычисления каждого члена $\Delta(h_r, o)$ в обратном направлении. После этого вычислить градиенты по весам в соответствии со следующей формулой:

$$\frac{\partial L}{\partial w_{(h_{r-1}, h_r)}} = \Delta(h_r, o) \cdot h_{r-1} \cdot \Phi'(a_{h_r}). \quad (3.13)$$

Частные производные по смещениям можно вычислить, используя тот факт, что нейроны смещения всегда активируются со значением $+1$. Поэтому для вычисления частной производной функции потерь по смещению узла h_r мы просто полагаем значение h_{r-1} равным 1 в правой части уравнения 3.13.

4. Использовать вычисленные частные производные функции потерь по весам для выполнения стохастического градиентного спуска применительно к паре “вход — выход” (\bar{X}, y) .

Приведенное выше описание алгоритма обратного распространения ошибки в значительной степени упрощено, и фактически реализуемая процедура должна включать многочисленные изменения, направленные на обеспечение эффективной и стабильной работы данного алгоритма. Например, градиенты вычисляются сразу для многих тренировочных примеров, совокупность которых называется *мини-пакетом* (mini-batch). Обратное распространение выполняется сразу для всех этих примеров с последующим сложением их локальных градиентов и выполнением мини-пакетного стохастического градиентного спуска. Эта усовершенствованная процедура будет обсуждаться в разделе 3.2.8. Еще одним отличием является наше предположение о том, что сеть имеет только один выход. Однако многие типы сетей (например, мультиклассовые перцептроны) характеризуются наличием нескольких выходов. Описание, приведенное в этом разделе, можно легко обобщить на случай нескольких выходов, добавив вклады различных выходов в производные функции потерь (раздел 3.2.7).

Необходимо отметить следующее. Из уравнения 3.13 видно, что частная производная функции потерь по весу ребра, ведущего от узла h_{r-1} к узлу h_r , всегда содержит мультипликативный член h_{r-1} . Другой мультипликативный член в уравнении 3.13 рассматривается как “ошибка”, распространяющаяся в обратном направлении. В определенном смысле алгоритм выполняет рекурсивное обратное распространение ошибок и умножает их на значения узлов скрытого слоя непосредственно перед обновлением матрицы весов. Именно по этой причине обратное распространение трактуют как распространение ошибки.

3.2.4. Обратное распространение ошибки с преактивационными переменными

В приведенном выше обсуждении для вычисления цепного правила используются значения $h_1 \dots h_k$ вдоль пути. Однако для этого также можно использовать значения, предшествующие применению функции активации $\Phi(\cdot)$. Иными словами, градиенты вычисляются по преактивационным значениям скрытых переменных, а затем распространяются в обратном направлении. Именно такой альтернативный подход к обратному распространению ошибки излагается в большинстве учебников.

Пусть a_{h_r} — преактивационное значение скрытой переменной h_r . Тогда

$$h_r = \Phi(a_{h_r}). \quad (3.14)$$

Различие между пред- и постактивационными значениями показано на рис. 3.6. В данном случае мы можем переписать уравнение 3.9 в следующем виде:

$$\frac{\partial L}{\partial w_{(h_{r-1}, h_r)}} = \frac{\partial L}{\partial o} \cdot \Phi'(a_o) \cdot \underbrace{\left[\sum_{[h_r, h_{r+1}, \dots, h_k, o] \in \mathcal{P}} \frac{\partial a_o}{\partial a_{h_k}} \prod_{i=r}^{k-1} \frac{\partial a_{h_{i+1}}}{\partial a_{h_i}} \right]}_{\text{Алгоритм обратного распространения}} h_{r-1}. \quad (3.15)$$

Алгоритм обратного распространения вычисляет $\delta(h_r, o) = \frac{\partial L}{\partial a_{h_k}}$

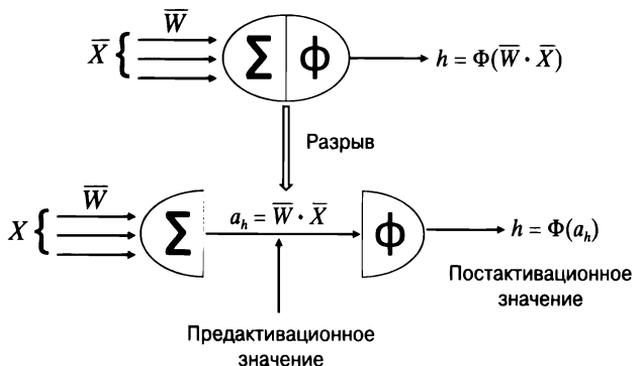


Рис. 3.6. Пред- и постактивационные значения в нейроне

В данном случае мы используем величину $\delta()$ в формуле рекурсии. Обратите внимание на то, что в рекурсии для $\Delta(h_r, o) = \frac{\partial L}{\partial h_r}$ в качестве промежуточных переменных в цепном правиле используются скрытые значения, получаемые *после* каждой активации, тогда как в рекурсии для $\delta(h_r, o) = \frac{\partial L}{\partial a_{h_r}}$ используются скрытые значения, получаемые *перед* активацией. Как и в случае уравнения 3.10, мы можем получить следующие рекуррентные уравнения:

$$\delta(h_r, o) = \frac{\partial L}{\partial a_{h_r}} = \sum_{h:h_r \Rightarrow h} \frac{\partial L}{\partial a_h} \frac{\partial a_h}{\partial a_{h_r}} = \sum_{h:h_r \Rightarrow h} \frac{\partial a_h}{\partial a_{h_r}} \delta(h, o). \quad (3.16)$$

Для вычисления производной $\frac{\partial a_h}{\partial a_{h_r}}$ в правой части уравнения 3.16 можно использовать цепное правило:

$$\frac{\partial a_h}{\partial a_{h_r}} = \frac{\partial a_h}{\partial h_r} \cdot \frac{\partial h_r}{\partial a_{h_r}} = w_{(h_r, h)} \cdot \frac{\partial \Phi(a_{h_r})}{\partial a_{h_r}} = \Phi'(a_{h_r}) \cdot w_{(h_r, h)}. \quad (3.17)$$

Подставляя вычисленное выражение для $\frac{\partial a_h}{\partial a_{h_r}}$ в правую часть уравнения 3.16, получаем следующее соотношение:

$$\delta(h_r, o) = \Phi'(a_{h_r}) \sum_{h:h_r \Rightarrow h} w_{(h_r, h)} \cdot \delta(h, o). \quad (3.18)$$

Уравнение 3.18 также можно получить, используя преактивационные переменные в обобщенном алгоритме вычислительного графа (см. раздел 3.2.2). Для этого нужно заменить $z(i, j)$ в уравнении 3.8 произведением весов ребер, связывающих узлы i и j , и производной функции активации в узле i .

Одним из преимуществ условия рекурсии в этой форме по сравнению с той, которая была получена с использованием постактивационных переменных, является то, что градиент функции активации вынесен за знак суммы, поэтому мы можем легко вычислить специфическую форму рекурсии для каждого типа активационной функции в узле h_r . Кроме того, поскольку градиент функции активации находится за пределами знака суммы, мы можем упростить вычисления, разделив эффекты активации и линейного преобразования в обновлениях обратного распространения ошибки. В упрощенном и *обособленном* представлении, которое будет более подробно обсуждаться в разделе 3.2.6, для целей динамического программирования используются как пред-, так и постактивационные переменные. Этот упрощенный подход показывает, как обратное распространение ошибки фактически реализуется в реальных системах. С точки

зрения реализации разделение линейного преобразования и функции активации весьма полезно, поскольку линейная часть — это обычное матричное умножение, тогда как часть, связанная с активацией, включает поэлементные операции умножения. Обе эти части могут эффективно реализовываться на любом оборудовании, выполняющем операции над матрицами на аппаратном уровне (таким, например, как графические процессоры).

Теперь пошаговый процесс обратного распространения ошибки может быть описан следующим образом.

1. Использовать проход в прямом направлении для вычисления значений всех скрытых элементов, выхода o и функции потерь L для конкретной пары “вход — выход” (\bar{X}, y) .
2. Инициализировать производную $\frac{\partial L}{\partial a_o} = \delta(o, o)$ значением $\frac{\partial L}{\partial o} \cdot \Phi'(a_o)$.
3. Использовать рекурсию 3.18 для вычисления каждого члена $\delta(h_r, o)$ в обратном направлении. После каждого такого вычисления рассчитать градиенты по весам в соответствии со следующей формулой:

$$\frac{\partial L}{\partial w_{(h_{r-1}, h_r)}} = \delta(h_r, o) \cdot h_{r-1}. \quad (3.19)$$

Частные производные по смещениям можно вычислить, используя тот факт, что нейроны смещения всегда активируются со значением +1. Поэтому для вычисления частной производной функции потерь по смещению узла h_r мы просто полагаем значение h_{r-1} равным 1 в правой части уравнения 3.19.

4. Использовать вычисленные частные производные функции потерь по весам для выполнения стохастического градиентного спуска применительно к паре “вход — выход” (\bar{X}, y) .

Основное отличие этого (более распространенного) варианта алгоритма обратного распространения ошибки — в способе записи рекурсии, поскольку преактивационные переменные уже были использованы для целей динамического программирования. С математической точки зрения пред- и пост-варианты обратного распространения эквивалентны (см. упражнение 9). Мы решили продемонстрировать оба варианта обратного распространения, чтобы подчеркнуть тот факт, что динамическое программирование можно использовать рядом способов, приводящих к эквивалентным уравнениям. Еще более упрощенное представление процесса обратного распространения ошибки, в котором используются как пред-, так и постактивационные переменные, описано в разделе 3.2.6.

3.2.5. Примеры обновлений для различных активаций

Одним из преимуществ уравнения 3.18 является то, что оно позволяет вычислять специфические типы обновлений для разных улов. Вот примеры реализации уравнения 3.18 для узлов различного типа:

$$\delta(h_r, o) = \sum_{h: h_r \Rightarrow h} w_{(h_r, h)}(h_r, o) \quad [\text{линейный}],$$

$$\delta(h_r, o) = h_r(1-h_r) \sum_{h: h_r \Rightarrow h} w_{(h_r, h)}(h_r, o) \quad [\text{сигмоида}],$$

$$\delta(h_r, o) = (1-h_r^2) \sum_{h: h_r \Rightarrow h} w_{(h_r, h)}(h_r, o) \quad [\text{гиперболический тангенс}].$$

Обратите внимание на то, что производную сигмоиды можно выразить через ее *выходное* значение h_r в виде $h_r(1-h_r)$. Аналогичным образом производную гиперболического тангенса можно записать в виде $(1-h_r^2)$. Производные функций активации различного типа обсуждались в разделе 1.2.1.6. Для функции ReLU значение $\delta(h_r, o)$ вычисляется так:

$$\delta(h_r, o) = \begin{cases} \sum_{h: h_r \Rightarrow h} w_{(h_r, h)}, & \text{если } 0 < a_{h_r}, \\ 0 & \text{во всех остальных случаях.} \end{cases}$$

Можно показать, что рекурсия для спрямленного гиперболического тангенса выглядит аналогично, за исключением того, что условие обновления немного отличается:

$$\delta(h_r, o) = \begin{cases} \sum_{h: h_r \Rightarrow h} w_{(h_r, h)}, & \text{если } -1 < a_{h_r} < 1, \\ 0 & \text{во всех остальных случаях.} \end{cases}$$

Следует отметить, что ReLU и гиперболический тангенс не являются дифференцируемыми функциями на граничных точках указанных интервалов. Однако на практике, когда вычисления выполняются с конечной точностью, это редко создает какие-либо трудности.

3.2.5.1. Особый случай активации Softmax

Функция активации *Softmax* представляет собой особый случай, поскольку она вычисляется не по одному, а по нескольким входам. Поэтому в отношении ее нельзя использовать тот же тип обновления, что и для других активационных функций. В соответствии с обсуждением уравнения 1.12 функция *Softmax* пре-

образует k вещественных значений предсказаний $v_1 \dots v_k$ в выходные вероятности $o_1 \dots o_k$ согласно следующему соотношению:

$$o_i = \frac{\exp(v_i)}{\sum_{j=1}^k \exp(v_j)} \quad \forall i \in \{1, \dots, k\}. \quad (3.20)$$

Обратите внимание на то, что если мы попытаемся использовать цепное правило для обратного распространения производной функции потерь L по $v_1 \dots v_k$, то нам придется вычислять каждую производную $\frac{\partial L}{\partial o_i}$ и каждую производную $\frac{\partial o_i}{\partial v_j}$. Обратное распространение в этом случае можно значительно упростить,

если принять во внимание следующие два фактора.

1. В выходном слое почти всегда используется функция *Softmax*.
2. Функция *Softmax* почти всегда используется совместно с *кросс-энтропийной функцией потерь* (cross-entropy loss). Пусть $y_1 \dots y_k \in \{0, 1\}$ — (наблюдаемые) выходы в представлении прямого кодирования для k взаимоисключающих классов. Тогда кросс-энтропийные потери определяются следующим образом:

$$L = -\sum_{i=1}^k y_i \log(o_i). \quad (3.21)$$

Ключевую роль играет тот факт, что в случае функции *Softmax* значение производной $\frac{\partial L}{\partial v_i}$ выражается особенно просто:

$$\frac{\partial L}{\partial v_i} = \frac{\partial L}{\partial o_j} \cdot \frac{\partial o_j}{\partial v_i} = o_i - y_i. \quad (3.22)$$

Мы предлагаем читателям самостоятельно получить приведенный выше результат; соответствующие выкладки трудоемки, но относительно просты. В процессе этого вывода используется тот факт, что, как несложно показать, значение производной $\frac{\partial o_j}{\partial v_i}$ в уравнении 3.22 равно $o_i(1 - o_i)$ при $i = j$ (как и в случае сигмоиды), тогда как в остальных случаях оно равно $-o_i o_j$ (см. упражнение 10).

Поэтому в случае функции *Softmax* сначала выполняется обратное распространение ошибки от выхода до слоя, содержащего $v_1 \dots v_k$. Далее обратное распространение можно продолжать в соответствии с правилами, которые обсуждались ранее. Обратите внимание на то, что мы отделили обновление активации

при обратном распространении от обратного распространения в остальной части сети, в которой операции матричного умножения всегда выполняются с включением функций активации в процессе обновления. В общем случае целесообразно создавать такое представление обратного распространения, которое позволяет “развязать” слои линейного матричного умножения и слои активации, поскольку это значительно упрощает вычисление обновлений. Обсуждению представления такого типа посвящен следующий раздел.

3.2.6. Обособленное векторное представление процесса обратного распространения ошибки

Выше мы обсудили два эквивалентных способа вычисления обновлений на основании уравнений 3.12 и 3.18. В каждом из этих случаев *обратное распространение осуществляется посредством одновременного выполнения линейных операций матричного умножения и вычисления функций активации*. От выбора порядка выполнения этих двух связанных вычислений зависит, получим ли мы уравнение 3.12 или 3.18. К сожалению, именно такой излишне усложненный подход к рассмотрению обратного распространения ошибки преобладает в статьях и учебниках. Частично это объясняется тем, что при определении слоев нейронных сетей в них традиционно объединялись вычисления, связанные с линейными преобразованиями и активацией элементов.

Однако во многих реальных реализациях линейные преобразования и активация обособляются в виде отдельных “слоев”, и в этом случае обратное распространение ошибки затрагивает два независимых слоя. При этом для описания слоев нейронной сети используют векторное представление, чтобы применяемые к слою операции можно было рассматривать как векторные, подобные матричному умножению в линейном слое (см. рис. 1.11, *з*). Такой подход значительно упрощает вычисления. Он позволяет создать сеть, в которой слои активации чередуются с линейными слоями (рис. 3.7). Обратите внимание на то, что в активационных слоях при необходимости можно использовать тождественное преобразование. Как правило, вычисления в активационных слоях сводятся к применению функции активации к каждой компоненте вектора, тогда как операции в линейных слоях означают умножение вектора на матрицу коэффициентов W . Тогда для каждой пары слоев матричного умножения и применения функции активации должны выполняться шаги распространения в прямом и обратном направлении, представленные на рис. 3.7.

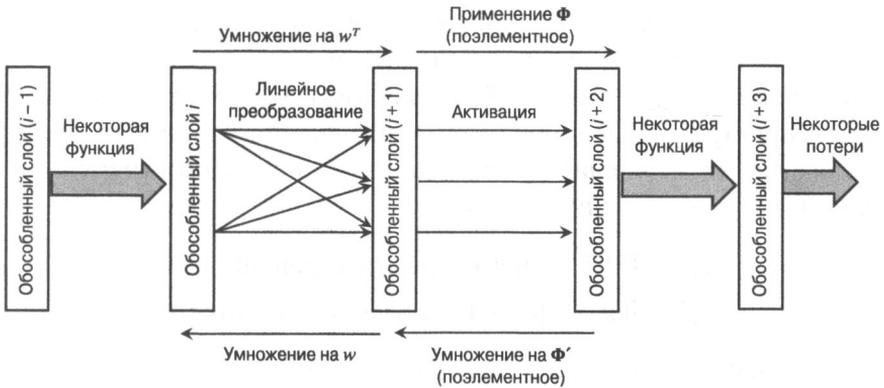


Рис. 3.7. Обособленное представление обратного распространения ошибки

Таблица 3.1. Примеры различных функций и их обновлений при обратном распространении ошибки между слоями i и $(i + 1)$. Скрытые значения и градиенты в слое i обозначены как \bar{z}_i и \bar{g}_i . В некоторых из этих вычислений в качестве бинарной индикаторной функции используется тождественное преобразование $I(\cdot)$

Функция	Тип связи	Прямое распространение	Обратное распространение
Линейная	Многие к многим	$\bar{z}_{i+1} = W^T \bar{z}_i$	$\bar{g}_i = W \bar{g}_{i+1}$
Сигмоида	Один к одному	$\bar{z}_{i+1} = \text{sigmoid}(\bar{z}_i)$	$\bar{g}_i = \bar{g}_{i+1} \odot \bar{z}_{i+1} = \odot (1 - \bar{z}_{i+1})$
Гиперболический тангенс	Один к одному	$\bar{z}_{i+1} = \tanh(\bar{z}_i)$	$\bar{g}_i = \bar{g}_{i+1} \odot (1 - \bar{z}_{i+1}) \odot \bar{z}_{i+1}$
ReLU	Один к одному	$\bar{z}_{i+1} = \bar{z}_i \odot I(\bar{z}_i > 0)$	$\bar{g}_i = \bar{g}_{i+1} \odot I(\bar{z}_i > 0)$
Спрямленный гиперболический тангенс	Один к одному	Установить в ± 1 ($\notin [-1, +1]$) Копировать ($\in [-1, +1]$)	Установить в 0 ($\notin [-1, +1]$) Копировать ($\in [-1, +1]$)
Max	Многие к одному	Максимальный из входов	Установить в 0 (не максимальные входы) Копировать (максимальный вход)
Произвольная функция $f_k(\cdot)$	Любой	$\bar{z}_{i+1}^{(k)} = f_k(\bar{z}_i)$	$\bar{g}_i = J^T \bar{g}_{i+1}$ J — якобиан (уравнение 3.23)

1. Пусть \bar{z}_i и \bar{z}_{i+1} — вектор-столбцы активации при распространении в прямом направлении, W — матрица линейного преобразования при переходе от слоя i к слою $(i + 1)$, а \bar{g}_i и \bar{g}_{i+1} — векторы градиентов в этих двух слоях, распространяемые в обратном направлении. Каждый элемент \bar{g}_i — это частная производная функции потерь по скрытой переменной в i -м слое. Тогда имеем следующие соотношения:

$$\begin{aligned}\bar{z}_{i+1} &= W^T \bar{z}_i \quad [\text{прямое распространение}], \\ \bar{g}_i &= W \bar{g}_{i+1} \quad [\text{обратное распространение}].\end{aligned}$$

2. Рассмотрим ситуацию, когда функция активации $\Phi(\cdot)$ применяется к каждому узлу в слое $(i + 1)$ для получения активаций в слое $(i + 2)$. В этом случае соответствующие соотношения приобретают следующий вид:

$$\begin{aligned}\bar{z}_{i+2} &= \Phi(\bar{z}_{i+1}) \quad [\text{прямое распространение}], \\ \bar{g}_{i+1} &= \bar{g}_{i+2} \odot \Phi'(\bar{z}_{i+1}) \quad [\text{обратное распространение}].\end{aligned}$$

Здесь $\Phi(\cdot)$ и ее производная $\Phi'(\cdot)$ применяются к своим векторным аргументам поэлементно. Операция поэлементного умножения обозначена символом \odot .

Обратите внимание на то, как разительно упростилась ситуация после отделения активации от операций матричного умножения в данном слое. Вычисления, выполняемые при распространении в прямом и обратном направлении, показаны на рис. 3.7. Кроме того, производные $\Phi'(\bar{z}_{i+1})$ часто можно выразить через выходы следующего слоя. Основываясь на материале, изложенном в разделе 3.2.5, можно показать, что для сигмоидных активаций выполняется следующее соотношение:

$$\begin{aligned}\Phi'(\bar{z}_{i+1}) &= \Phi(\bar{z}_{i+1}) \odot (1 - \Phi(\bar{z}_{i+1})) = \\ &= \bar{z}_{i+2} \odot (1 - \bar{z}_{i+2}).\end{aligned}$$

Примеры различных типов обновлений в процессе обратного распространения для различных функций, участвующих в прямом распространении, приведены в табл. 3.1. В данном случае мы использовали индексы слоев i и $(i + 1)$ как для линейных преобразований, так и для функций активации, вместо того чтобы использовать для функции активации индекс $(i + 2)$. Обратите внимание на предпоследнюю строку таблицы, которая соответствует функции максимизации. Функции этого типа полезны для выполнения операций пулинга по максимальному значению (max-pooling) в сверточных нейронных сетях. Поэтому процесс распространения в обратном направлении имеет тот же вид, что и в прямом. Зная вектор градиентов в некотором слое, можно получить градиенты

для предыдущего слоя, применив операции, приведенные в последнем столбце табл. 3.1.

Некоторые операции в нейронных сетях описываются более сложными функциями с типами связей “многие ко многим”, а не просто матричными умножениями. Такие ситуации обрабатываются в предположении, что k -я активация в слое $(i + 1)$ получается путем применения произвольной функции $f_k(\cdot)$ к вектору активации в слое i . Тогда элементы якобиана определяются следующим образом:

$$J_{kr} = \frac{\partial f_k(\bar{z}_i)}{\partial \bar{z}_i^{(r)}}, \quad (3.23)$$

где $\bar{z}_i^{(r)}$ — r -й элемент \bar{z}_i . Обозначим через J матрицу, элементами которой являются J_{kr} . Тогда обновление при обратном распространении ошибки от одного слоя к другому можно записать в следующем виде:

$$\bar{g}_i = J^T \bar{g}_{i+1}. \quad (3.24)$$

С точки зрения реализации запись уравнений обратного распространения ошибки с помощью операций матричного умножения часто обеспечивает такие преимущества, как ускорение вычислений за счет использования графических процессоров (раздел 3.7.1). Обратите внимание на то, что элементы вектора \bar{g}_i представляют градиенты функции потерь по активациям в i -м слое, поэтому для вычисления градиентов по весам требуется дополнительный шаг. Градиент функции потерь по весу связи между p -м элементом $(i - 1)$ -го слоя и q -м элементом i -го слоя получается умножением p -го элемента вектора \bar{z}_{i-1} на q -й элемент вектора \bar{g}_i .

3.2.7. Функции потерь на нескольких выходных и скрытых узлах

В предыдущем обсуждении мы упростили вычисление функции потерь, ограничившись единственным выходным узлом. Однако в большинстве приложений функция потерь вычисляется на множестве выходных узлов O . В таком случае единственное отличие состоит в том, что каждая производная $\frac{\partial L}{\partial a_0} = \delta(o, O)$ для $o \in O$ инициализируется значением $\frac{\partial L}{\partial o} \Phi'(o)$. После этого с помощью механизма обратного распространения ошибки для каждого скрытого узла h вычисляется значение $\frac{\partial L}{\partial a_h} = \delta(h, O)$.

В некоторых видах обучения на разреженных признаках функции потерь ассоциируются даже с выходами скрытых узлов. Этим часто пользуются для

того, чтобы стимулировать получение решений со специфическими свойствами, такими, например, как разреженный скрытый слой (разреженные автокодировщики) или скрытый слой со специфическим типом регуляризационного штрафа (сжимающие автокодировщики). Штрафы за разреженность обсуждаются в разделе 4.4.4, а сжимающие автокодировщики — в разделе 4.10.3. В подобных случаях алгоритм обратного распространения ошибки нуждается лишь в незначительных изменениях, в которых поток градиента в обратном направлении базируется на всех узлах, в которых вычисляются потери. Этого можно достигнуть простой агрегацией потоков градиентов, результирующих из различных потерь, что можно рассматривать как специальный тип сети, в которой скрытые узлы также выступают в качестве выходных узлов, а выходные узлы не ограничиваются последним слоем сети. На самом базовом уровне технология обратного распространения ошибки остается одной и той же.

Рассмотрим ситуацию, когда с каждым скрытым узлом h_r ассоциируется функция потерь L_{h_r} , а L — общая функция потерь, вычисляемая по всем узлам.

Пусть $\frac{\partial L}{\partial a_{h_r}} = \delta(h_r, N(h_r))$ — поток градиента от всех узлов $N(h_r)$, достижимых из узла h_r , с которым связана часть потерь. В этом случае множество узлов $N(h_r)$ может включать узлы как выходного слоя, так и скрытого (с которым связываются потери), при условии, что они достижимы из узла h_r . Поэтому множество $N(h_r)$ использует h_r в качестве аргумента. Обратите внимание на то, что множество $N(h_r)$ включает сам узел h_r . Тогда в качестве первого шага мы можем переписать выражение для обновления (см. уравнение 3.18) в следующем виде:

$$\delta(h_r, N(h_r)) \leftarrow \Phi'(a_{h_r}) \sum_{h: h_r \Rightarrow h} w_{(h_r, h)} \delta(h, N(h)). \quad (3.25)$$

Это выражение аналогично стандартному выражению для обновления в процессе обратного распространения ошибки. Однако в нынешнем виде оно не включает вклад узла h_r . Поэтому требуется выполнить *дополнительный шаг* для включения в величину $\delta(h_r, N(h_r))$ поправки, учитывающей вклад h_r в функцию потерь:

$$\delta(h_r, N(h_r)) \leftarrow \delta(h_r, N(h_r)) + \Phi'(h_r) \frac{\partial L_{h_r}}{\partial h_r}. \quad (3.26)$$

Важно иметь в виду, что общая функция потерь L отличается от функции потерь L_{h_r} , специфичной для узла h_r . Кроме того, поправка, введенная в поток градиента (3.26), имеет тот же алгебраический вид, что и значение инициализации для выходных узлов. Иными словами, потоки градиента, обусловленные скрытыми узлами, аналогичны потокам градиента выходных узлов. Единственное

отличие состоит в том, что вычисленное значение добавляется к существующему потоку градиента в скрытых узлах. Поэтому общая схема обратного распространения ошибки остается почти той же, с тем основным отличием, что алгоритм обратного распространения ошибки включает дополнительные вклады, обусловленные потерями в скрытых узлах.

3.2.8. Мини-пакетный стохастический градиентный спуск

Начиная с самой первой главы мы обновляли веса, аппроксимируя градиент функции стоимости значением, вычисляемым для отдельных точек данных, т.е. использовали метод *стохастического градиентного спуска* (stochastic gradient descent). Такой подход является общепринятым в глубоком обучении. В этом разделе мы предоставим обоснование такого выбора, а также рассмотрим его родственные варианты, такие как *мини-пакетный стохастический градиентный спуск* (mini-batch stochastic gradient descent). Кроме того, будут проанализированы преимущества и недостатки различных вариантов выбора.

Большинство задач машинного обучения удается переформулировать в виде задачи оптимизации тех или иных специфических целевых функций. Например, в случае нейронных сетей целевую функцию можно определить в терминах оптимизации функции потерь L , которую часто можно представить в виде *линейно разделимой суммы* функций потерь индивидуальных тренировочных точек данных. Так, в *задачах линейной регрессии* минимизируется сумма квадратов ошибок предсказания по всему набору тренировочных точек данных. В *задачах снижения размерности* минимизируется сумма квадратов ошибок представления реконструированных тренировочных точек данных. Функцию потерь для нейронной сети можно записать в следующем виде:

$$L = \sum_{i=1}^n L_i, \quad (3.27)$$

где L_i — вклад потерь, обусловленный i -й тренировочной точкой. Используя алгоритмы, описанные в главе 2, мы работали с функциями потерь, вычисляемыми на индивидуальных обучающих точках данных, а не с агрегированными потерями.

В методе градиентного спуска функция потерь нейронной сети минимизируется путем изменения параметров в направлении антиградиента. Например, в случае перцептрона параметрам соответствует вектор $\vec{W} = (w_1 \dots w_d)$. Поэтому функция потерь для базовой целевой функции вычисляется по всем точкам одновременно, и для нее выполняется градиентный спуск. Следовательно, шаги, выполняемые в процессе традиционного градиентного спуска, описываются следующей формулой:

$$\bar{W} \leftarrow \bar{W} - \alpha \left(\frac{\partial L}{\partial w_1}, \frac{\partial L}{\partial w_2} \dots \frac{\partial L}{\partial w_d} \right). \quad (3.28)$$

Производную этого типа можно представить в более компактном виде, используя векторную нотацию:

$$\bar{W} \leftarrow \bar{W} - \alpha \frac{\partial L}{\partial \bar{W}}. \quad (3.29)$$

В случае таких однослойных сетей, как перцептрон, градиентный спуск выполняется лишь по \bar{W} , тогда как в случае более крупных сетей все параметры должны обновляться с помощью процедуры обратного распространения ошибки. В крупномасштабных приложениях количество параметров может исчисляться многими миллионами, и для вычисления обновлений с помощью алгоритма обратного распространения ошибки необходимо выполнять *одновременно* все примеры в прямом и обратном направлении. Однако пропуск через сеть одновременно всех примеров для вычисления градиента по *всему набору данных* за один раз — практически неосуществимая задача. Не забывайте также о тех объемах памяти, которые понадобились бы для хранения всех промежуточных/окончательных прогнозных значений для каждого тренировочного примера, нуждающихся в поддержке градиентным спуском. В большинстве ситуаций, которые возникают на практике, эти объемы оказываются чрезмерно большими. В начале процесса обучения веса нередко могут быть такими, что даже небольшой выборки данных будет достаточно для получения отличной оценки направления градиента. Аддитивный эффект обновлений, созданных на основе таких выборок, часто может предоставить точное направление изменения параметров. Этот факт является практическим подтверждением успешности метода стохастического градиентного спуска и его вариантов.

Поскольку в большинстве задач оптимизации функцию потерь можно выразить в виде линейной суммы потерь, ассоциированных с отдельными точками (см. уравнение 3.27), нетрудно показать, что

$$\frac{\partial L}{\partial \bar{W}} = \sum_{i=1}^n \frac{\partial L_i}{\partial \bar{W}}. \quad (3.30)$$

В этом случае для обновления полного градиента с учетом всех точек необходимо суммировать их индивидуальные эффекты. Задачам машинного обучения свойственна высокая степень избыточности информации, предоставляемой различными тренировочными точками, так что во многих случаях процесс обучения может эффективно осуществляться с использованием обновлений стохастического градиентного спуска, вычисляемых для отдельных точек:

$$\bar{W} \leftarrow \bar{W} - \alpha \frac{\partial L_i}{\partial \bar{W}}. \quad (3.31)$$

Этот тип градиентного спуска называется *стохастическим*, поскольку предполагает циклический обход точек в некотором случайном порядке. Заметьте, что долгосрочный эффект таких повторных обновлений остается примерно тем же, хотя при стохастическом градиентном спуске каждое обновление может рассматриваться лишь как вероятностная аппроксимация. Каждый локальный градиент может вычисляться весьма эффективно, что повышает быстродействие стохастического градиентного спуска, хотя и за счет точности вычисления градиента. Стохастический градиентный спуск обладает тем интересным свойством, что даже в тех случаях, когда он справляется с тренировочными данными хуже, чем градиентный спуск, он способен обеспечивать сравнимые (а иногда даже лучшие) результаты при работе с тестовыми данными [171]. Как будет показано в главе 4, стохастический градиентный спуск оказывает косвенный эффект регуляризации. Однако иногда при неудачном порядке следования тренировочных точек он может приводить к очень плохим результатам.

В мини-пакетном стохастическом градиентном спуске обновление выполняется с использованием пакета тренировочных точек $B = \{j_1 \dots j_m\}$:

$$\bar{W} \leftarrow \bar{W} - \alpha \sum_{i \in B} \frac{\partial L_i}{\partial \bar{W}}. \quad (3.32)$$

Мини-пакетный стохастический градиентный спуск часто обеспечивает достижение наилучшего компромисса между требованиями стабильности, скорости и памяти. При этом выходами слоя являются матрицы, а не векторы, следовательно, процедура прямого распространения требует вычисления произведений матрицы весов и матрицы активаций. То же самое относится и к обратному распространению, которое в таком случае поддерживает матрицы градиентов. Поэтому реализация мини-пакетного стохастического градиентного спуска предъявляет высокие требования к памяти, что является главным фактором, ограничивающим размер мини-пакета.

В силу этих причин размер мини-пакета регулируется в зависимости от объема памяти, доступного на используемом оборудовании. Слишком малый размер мини-пакета также приводит к постоянным накладным расходам и неэффективен даже с вычислительной точки зрения. Увеличение размера пакета свыше некоторого порогового значения (обычно порядка нескольких сотен точек данных) не дает никакого дополнительного выигрыша в точности вычисления градиента. Обычно размер пакета выбирается кратным степеням 2, поскольку этот выбор часто обеспечивает наилучшие результаты на большинстве аппаратных архитектур; типичными значениями являются 32, 64, 128 и 256. Несмотря на то

что мини-пакетный стохастический градиентный спуск повсеместно применяется для обучения нейронных сетей, в данной книге обновления в большинстве случаев вычисляются по упрощенной схеме на основе индивидуальных точек (т.е. с использованием чисто стохастического градиентного спуска).

3.2.9. Приемы обратного распространения ошибки для обработки разделяемых весов

Весьма распространенным подходом к регуляризации нейронных сетей является использование *разделяемых весов* (shared weights). Базовая идея такова: если известно, что аналогичная функция будет вычисляться в других узлах сети, то веса, связанные с этими узлами, будут ограничиваться одними и теми же значениями. Приведем некоторые примеры.

1. В автокодировщике, имитирующем PCA (см. раздел 2.5.1.3), входной и выходной слои используют одни и те же веса.
2. В рекуррентной нейронной сети, предназначенной для обработки текста (см. главу 7), веса разделяются различными временными слоями, поскольку предполагается, что на каждом временном шаге *языковая модель* остается одной и той же.
3. В сверточной нейронной сети одна и та же сетка весов (соответствующая визуальному полю) используется на протяжении всей пространственной области нейронов (глава 8).

Разделение весов продуманным с точки зрения семантики способом — один из ключевых приемов построения успешных нейронных сетей. Если у вас есть все основания полагать, что в двух узлах будет вычисляться одна и та же функция, то для них имеет смысл использовать одни и те же веса.

На первый взгляд, вычисление градиента функции потерь по весам, разделяемым узлами, которые находятся в разных частях сети, может показаться трудоемкой задачей. Однако реализация обратного распространения ошибки с использованием разделяемых весов не представляет особых математических трудностей.

Пусть w — вес, разделяемый T различными узлами сети, $\{w_1 \dots w_m\}$ — соответствующие копии весов в этих узлах, а L — функция потерь. Тогда, используя цепное правило, можно показать, что

$$\begin{aligned} \frac{\partial L}{\partial w} &= \sum_{i=1}^T \frac{\partial L}{\partial w_i} \underbrace{\frac{\partial w_i}{\partial w}}_{=1} = \\ &= \sum_{i=1}^T \frac{\partial L}{\partial w_i}. \end{aligned}$$

Иными словами, все, что мы должны сделать, — это вычислить производные функции стоимости по весам и сложить их, действуя так, будто веса являются независимыми! Поэтому мы просто применяем алгоритм обратного распространения ошибки без внесения в него каких-либо изменений, а затем суммируем градиенты различных экземпляров разделяемого веса. Этим обстоятельством интенсивно пользуются при обучении нейронных сетей. Оно также служит основой алгоритма обучения рекуррентных нейронных сетей.

3.2.10. Проверка корректности вычисления градиентов

Алгоритм обратного распространения ошибки довольно сложен, и иногда может требоваться проверка корректности вычисления градиентов. Это легко делается с помощью численных методов. Рассмотрим конкретный вес w случайно выбранного ребра в сети. Обозначим через $L(w)$ текущее значение функции потерь. Вес данного ребра незначительно изменяется путем прибавления к нему небольшой величины $\varepsilon > 0$. Далее выполняется алгоритм прямого распространения с использованием измененного веса и вычисляется функция потерь $L(w + \varepsilon)$. Тогда для частной производной функции потерь по w справедливо следующее соотношение:

$$\frac{\partial L}{\partial w} \approx \frac{L(w + \varepsilon) - L(w)}{\varepsilon}. \quad (3.33)$$

Если частные производные не удовлетворяют данному условию с достаточной точностью, то это указывает на ошибку в вычислениях. Такую проверку достаточно выполнять лишь для двух-трех контрольных точек в процессе обучения. Однако проверку в этих контрольных точках рекомендуется выполнять для большого подмножества параметров. Одной из проблем является определение того, что именно следует считать *достаточной точностью*, особенно в отсутствие каких-либо предварительных данных о возможных абсолютных величинах значений. Эту проблему решают за счет использования относительных величин.

Обозначим через G_e производную, полученную в результате прямого распространения, а вышеприведенную оценку — через G_a . Тогда относительная погрешность r определяется следующим образом:

$$\rho = \frac{|G_e - G_a|}{|G_e + G_a|}. \quad (3.34)$$

В типичных случаях это отношение не должно превышать 10^{-6} , хотя для некоторых функций активации, таких как ReLU, существуют точки, в которых производные испытывают резкие изменения, и тогда вычисленный градиент будет отличаться от его численной оценки. В подобных случаях допускается, чтобы

относительная погрешность не превышала 10^{-3} . Эти численные оценки можно использовать для тестирования различных ребер и проверки корректности их градиентов. Если количество параметров исчисляется миллионами, то можно ограничиться тестированием выборки производных для быстрой проверки корректности вычислений. Кроме того, в процессе тренировки такую проверку рекомендуется выполнять в двух-трех точках, поскольку результаты проверки на начальных стадиях обучения могут соответствовать особым случаям, которые не обобщаются на произвольные точки в пространстве параметров.

3.3. Настройка и инициализация сети

Существует ряд важных моментов, касающихся настройки нейронной сети, предварительной обработки данных и инициализации параметров. Прежде всего необходимо выбрать значения *гиперпараметров* нейронной сети (скорость обучения, параметры регуляризации и др.). Предварительная обработка данных и инициализация параметров также имеют немаловажное значение. Как правило, нейронные сети характеризуются большим количеством параметров по сравнению с другими алгоритмами машинного обучения, что во многих отношениях повышает роль предварительной обработки данных и выбора начальных значений параметров. Базовые методы предварительной обработки и инициализации параметров обсуждаются ниже. Строго говоря, такие передовые методы, как *предварительное обучение* (pretraining), также могут рассматриваться как разновидность инициализации. Однако использование подобного подхода требует глубокого понимания проблем обобщаемости модели, связанных с процессом обучения нейронных сетей. Поэтому обсуждение данной темы будет продолжено в следующей главе.

3.3.1. Тонкая настройка гиперпараметров

Нейронные сети имеют большое количество так называемых *гиперпараметров*, таких как скорость обучения, вес регуляризации и т.п. Термин “гиперпараметр” относится к параметрам, регулирующим характеристики собственно модели, которые не следует смешивать с параметрами, представляющими веса связей в нейронной сети. При работе с байесовскими статистиками это понятие используется в отношении параметра, управляющего априорным распределением, но в данной книге мы трактуем термин “гиперпараметр” несколько более свободно. В некотором смысле можно говорить о двухуровневой организации параметров нейронной сети, при которой основные параметры модели наподобие весов оптимизируются с помощью механизма обратного распространения ошибки лишь после того, как зафиксированы значения гиперпараметров, которые подбираются либо вручную, либо с использованием фазы *настройки*. Как будет обсуждаться в разделе 4.3, для настройки гиперпараметров не следует

использовать те же данные, что и для градиентного спуска. Вместо этого часть данных резервируется в качестве так называемых *валидационных данных*, и работа модели тестируется на этом валидационном наборе при различных вариантах выбора значений гиперпараметров. Такой подход гарантирует, что процесс тонкой настройки модели не приведет к ее переобучению на тренировочном наборе данных (в результате чего модель будет демонстрировать плохие результаты на тестовых данных).

Как выбирать пробные значения гиперпараметров для тестирования? Наиболее известная методика — *сеточный поиск* (grid search), когда для каждого гиперпараметра заранее выбирается набор значений. В самой простой реализации сеточного поиска для определения наиболее оптимального набора гиперпараметров тестируются все возможные комбинации их значений. Трудности такого подхода обусловлены тем, что с увеличением количества узлов сетки количество гиперпараметров увеличивается по *экспоненциальному* закону. Так, в случае тестирования 5 гиперпараметров, для каждого из которых будут испытываться 10 значений, процедуру тренировки сети с целью выбора комбинации гиперпараметров, обеспечивающей наилучшую точность, потребуется повторить $10^5 = 100\,000$ раз. Даже если вы не будете каждый раз доводить тренировку до конца, уже само число необходимых тестовых прогонов настолько велико, что выполнить их все за разумное время просто невозможно. Обычный трюк состоит в том, чтобы начинать с грубой сетки. Сузив интервал до значений, представляющих интерес, можно использовать более мелкую сетку. Следует быть особенно внимательным, если оптимальное значение гиперпараметра располагается вблизи границы интервала сетки, поскольку в этом случае необходимо проверить, не найдется ли лучшее значение за пределами сетки.

Иногда даже описанная процедура тестирования с предварительным использованием грубой сетки перед переходом к тонкой настройке гиперпараметров с помощью мелкой сетки не позволяет снизить накладные расходы до приемлемого уровня. Как было указано в [37], подбор оптимальных гиперпараметров на основе сетки значений не обязательно является наилучшим выбором. В некоторых случаях имеет смысл использовать случайные выборки значений гиперпараметров, равномерно распределенных в пределах интервалов сетки. Как и в случае простого перебора сеточных значений, такие случайные выборки также могут формироваться с использованием различного разрешения. Под этим подразумевается, что на первых порах значения выбираются из полного диапазона, охватываемого сеткой. Затем создается новый набор интервалов сетки, размеры которых уменьшены в геометрической прогрессии по сравнению с предыдущими и которые центрированы на оптимальных значениях гиперпараметров, полученных с помощью предыдущих выборок. Процесс семплирования повторяется для уменьшенной сетки, а затем весь описанный процесс многократно повторяется для уточнения параметров.

Суть еще одного важного приема, используемого для семплирования значений гиперпараметров многих типов, заключается в том, что из однородного распределения семплируются не сами значения, а их логарифмы. Двумя примерами таких гиперпараметров являются степень регуляризации и скорость обучения. Например, вместо семплирования значения скорости обучения α в интервале 0,001–0,1 мы сначала семплируем значение $\log(\alpha)$ из равномерного распределения в интервале от -1 до -3 , а затем возводим его в степень 10. При поиске оптимальных значений гиперпараметров чаще всего используют логарифмическую шкалу, хотя для некоторых гиперпараметров необходимо использовать только линейную шкалу.

Наконец, важно иметь в виду, что в случае крупномасштабных задач описанные алгоритмы не всегда удается выполнить до полного завершения из-за чрезмерной длительности необходимой тренировки. Например, на однократное выполнение цикла обработки изображения с помощью сверточной нейронной сети может уйти две недели. Выполнить такой алгоритм для многих комбинаций значений параметров практически невозможно. Однако нередко разумную оценку долговременного поведения алгоритма удается получить за достаточно короткое время. Поэтому развитие процесса часто тестируют, выполняя алгоритм лишь на протяжении определенного количества эпох. В случае очевидно плохих результатов или в отсутствие сходимости вычисления можно немедленно прекратить. Во многих случаях можно использовать многопоточную обработку, последовательно завершая или добавляя новые потоки для проведения выборочных вычислений с различными значениями гиперпараметров. Полное обучение модели выполняется лишь для наилучшей комбинации гиперпараметров. Иногда процесс тренировки может доводиться до завершения для нескольких лучших комбинаций гиперпараметров, предсказания которых усредняются, как в ансамбле.

Математически обоснованный способ выбора гиперпараметров обеспечивает *байесовская оптимизация* [42, 306]. Однако в случае крупных нейронных сетей соответствующие методы работают слишком медленно, чтобы их можно было применять на практике, и пока что они сохраняют лишь статус любопытного предмета для исследований. Для работы с небольшими сетями можно использовать такие библиотеки, как *Hyperopt* [614], *Spearmint* [616] и *SMAC* [615].

3.3.2. Предварительная обработка признаков

Методы обработки признаков для обучения нейронных сетей не очень отличаются от тех, которые используются в других алгоритмах машинного обучения. Существуют два вида предварительной обработки признаков, применяемые в машинном обучении.

1. *Аддитивная предварительная обработка* (additive preprocessing) и *центрирование по среднему* (mean-centering). Центрирование по среднему значению позволяет исключать некоторые типы эффектов смещения. Многие алгоритмы традиционного машинного обучения (такие, как анализ главных компонент) также работают в предположении, что данные центрированы по среднему. В подобных случаях из каждой точки данных вычитается вектор-столбец средних значений. Центрирование по среднему часто сочетается со *стандартизацией*, которая обсуждается ниже.

Если желательно, чтобы все признаки имели неотрицательные значения, то используется другой тип предобработки. В этом случае к значениям признака в каждой точке данных добавляется абсолютная величина наибольшего (без учета знака) из его отрицательных значений. Как правило, этот способ предобработки сочетается с минимаксной нормализацией, о которой речь пойдет ниже.

2. *Нормализация признаков* (feature normalization). Распространенным типом нормализации является деление значений каждого признака на его стандартное отклонение. Если этот тип масштабирования признаков сочетается с центрированием по среднему, то говорят, что данные *стандартизируются*. В основе этой идеи лежит предположение о том, что данные извлекаются из стандартного нормального распределения с нулевым средним и единичной дисперсией.

В тех случаях, когда данные должны быть масштабированы так, чтобы их значения находились в пределах интервала (0, 1), полезным оказывается другой тип нормализации признаков. Обозначим через \min_j и \max_j максимальное и минимальное значения атрибута соответственно. Тогда каждое значение признака x_{ij} , соответствующее j -му измерению i -й точки данных, масштабируется с помощью минимаксной нормализации следующим образом:

$$x_{ij} \leftarrow \frac{x_{ij} - \min_j}{\max_j - \min_j}. \quad (3.35)$$

Нормализация признаков часто позволяет улучшить работу модели, поскольку их значения нередко различаются на порядок величины. В подобных случаях очень важно правильно подобрать скорость обучения, поскольку функция потерь обычно более чувствительна к некоторым параметрам по сравнению с другими. Как будет показано далее, этот тип неустойчивости влияет на работу градиентного спуска. Поэтому рекомендуется заранее масштабировать признаки.

Отбеливание

Существует еще одна разновидность предварительной обработки признаков, так называемое *отбеливание* (whitening), когда путем поворота системы координат создается новый набор *декоррелированных* признаков, каждый из которых масштабируется до единичной дисперсии. Обычно для достижения этой цели используют анализ главных компонент.

Анализ главных компонент (principal component analysis — PCA) можно рассматривать как применение сингулярного разложения *после* центрирования по среднему (т.е. вычитания среднего из каждого столбца) матрицы данных. Пусть D — матрица данных размера $n \times d$, уже центрированная по среднему, а C — ковариационная матрица D размера $d \times d$, элемент (i, j) которой — ковариация между компонентами i и j . Поскольку матрица D центрирована по среднему значению, имеем следующее соотношение:

$$C = \frac{D^T D}{n} \propto D^T D. \quad (3.36)$$

Собственные векторы (eigenvectors) ковариационной матрицы представляют декоррелированные направления в данных. Кроме того, *собственные значения* (eigenvalues) представляют значения дисперсии в каждом из этих направлений. Поэтому, если использовать первые k собственных векторов (т.е. векторов, которым соответствуют k наибольших собственных значений) ковариационной матрицы, то это позволит удержать большую часть дисперсии и удалить шум. Также можно выбрать $k=d$, но это часто приводит к тому, что ошибки вычислений начинают превышать значения дисперсии вдоль векторов, близких к нулю. Включать измерения, дисперсия которых обусловлена вычислительными ошибками, — плохая идея, поскольку эти измерения будут содержать мало полезной информации для обучения знаниям, специфичным для приложения. Кроме того, процесс отбеливания будет масштабировать признак до единичной дисперсии, что увеличит ошибки вдоль этих направлений. По крайней мере, рекомендуется установить некоторый порог величины собственных значений, равный, скажем, 10^{-5} . Поэтому на практике значение k лишь в редких случаях будет в точности равно d . Альтернативный вариант заключается в прибавлении 10^{-5} к каждому собственному значению с целью регуляризации, прежде чем масштабировать каждое измерение.

Пусть P — матрица размера $d \times k$, в каждом столбце которой содержится один из первых k собственных векторов. Тогда матрицу данных D можно преобразовать в k -мерную координатную систему, умножив ее справа на матрицу P . Результирующая матрица U размера $n \times k$, строки которой содержат преобразованные k -мерные точки данных, выражается следующей формулой:

$$U = DP. \quad (3.37)$$

Обратите внимание на то, что значения дисперсии в столбцах U — это соответствующие собственные значения, поскольку таково свойство декоррелирующего преобразования в анализе главных компонент. При отбеливании каждый столбец U масштабируется до единичной дисперсии путем деления на его стандартное отклонение (т.е. квадратный корень из соответствующего собственного значения). Преобразованные признаки передаются нейронной сети. Поскольку количество признаков может уменьшиться в процессе отбеливания, этот тип предобработки также может влиять на архитектуру сети ввиду уменьшения количества входов.

Важным аспектом отбеливания является то, что для оценки ковариационной матрицы большого набора данных вовсе не обязательно обрабатывать весь набор. В подобных случаях оценка ковариационной матрицы и средних значений столбцов может быть произведена на основе выборки данных. Сначала вычисляется матрица P собственных векторов размера $d \times k$, столбцы которой содержат первые k собственных векторов. После этого в отношении каждой точки данных выполняются следующие действия: 1) среднее значение каждого столбца вычитается из соответствующего признака; 2) каждый d -мерный вектор-строка, представляющий тренировочную (или тестовую) точку данных, умножается справа на матрицу P для создания k -мерного вектора-строки; 3) каждый признак этого k -мерного представления делится на квадратный корень из соответствующего собственного значения.

В основе отбеливания лежит допущение о том, что данные генерируются из независимых гауссовских распределений вдоль направлений главных компонент. При отбеливании предполагается, что каждое такое распределение является стандартным нормальным распределением и предоставляет равную значимость различным признакам. Обратите внимание на то, что диаграмма рассеяния данных будет иметь приблизительно сферическую форму, даже если исходные данные вытянуты в виде эллипса в произвольном направлении. Суть в том, что некоррелированные признаки теперь масштабированы до равной (априорной) значимости, и нейронная сеть может сама решать, какой из них заслуживает большего внимания в процессе обучения. Другое дело, когда различные признаки масштабируются по-разному, потому что в этом случае на начальной стадии обучения среди активаций и градиентов будут доминировать “крупные” признаки (если веса инициализированы случайным образом примерно одинаковыми значениями). Это может отрицательно сказаться на относительной скорости обучения некоторых важных весов в сети. Практические преимущества использования различных типов предварительной обработки и нормализации признаков обсуждаются в [278, 532].

3.3.3. Инициализация

Учитывая проблемы со стабильностью обучения нейронных сетей, их инициализация приобретает особенно важное значение. Как будет показано в разделе 3.4, проблемы со стабильностью в нейронных сетях проявляются в том, что при переходе от одного слоя к другому активации последовательно увеличиваются или уменьшаются. Этот эффект экспоненциально зависит от глубины сети и становится особенно заметным в очень глубоких сетях. Одним из способов смягчения данного эффекта является удачный выбор начальных точек таким образом, чтобы градиенты сохраняли стабильность в различных слоях.

Возможный подход к инициализации весов заключается в том, чтобы генерировать случайные значения из гауссовского распределения с нулевым средним и небольшим стандартным отклонением, равным, скажем, 10^{-2} . Обычно это приводит к случайным значениям небольшой величины, которые могут быть как положительными, так и отрицательными. Одной из проблем такой инициализации является то, что она не учитывает количество входов каждого конкретного нейрона. Например, если один нейрон имеет только 2 входа, а другой — 100, то выход второго намного более чувствителен к среднему весу ввиду аддитивного эффекта большего количества входов (что проявляется в намного большем градиенте). В общем случае можно показать, что дисперсия выходов нейрона изменяется пропорционально количеству входов, а стандартное отклонение — пропорционально квадратному корню из этого количества. Чтобы сбалансировать данный эффект, каждый вес инициализируется значением, извлеченным из гауссовского распределения со стандартным отклонением, равным $\sqrt{1/r}$, где r — количество входов данного нейрона. Нейроны смещения всегда инициализируются нулевым весом. Альтернативный вариант заключается в том, чтобы инициализировать веса случайными значениями, равномерно распределенными в интервале $[-1/\sqrt{r}, +1/\sqrt{r}]$.

Более сложные правила инициализации учитывают тот факт, что дисперсия выхода формируется за счет взаимодействия нейронов различных слоев. Обозначим через r_{in} и r_{out} количество входов и выходов конкретного нейрона. В соответствии с одним из таких правил, которое называют *инициализацией Ксавье* (Xavier initialization) или *инициализацией Глоро* (Glorot initialization), веса инициализируются случайными значениями, выбираемыми из гауссовского распределения со стандартным отклонением $\sqrt{2/(r_{in} + r_{out})}$.

При использовании рандомизированных методов очень важно *нарушать симметрию*. Если все веса инициализируются одним и тем же значением (таким, как 0), то все обновления в слое будут строго одними и теми же. В результате всеми нейронами слоя будут созданы идентичные признаки. Именно поэтому очень важно с самого начала вводить асимметрию между нейронами.

3.4. Проблемы затухающих и взрывных градиентов

При работе с глубокими нейронными сетями приходится сталкиваться с рядом проблем стабильности, связанных с их обучением. В частности, при обучении многослойных сетей могут возникать трудности, обусловленные тем, как связаны между собой градиенты в предыдущих и последующих слоях.

Чтобы понять суть проблемы, рассмотрим очень глубокую сеть, в каждом слое которой имеется всего один узел. Допустим, сеть содержит $(m + 1)$ слой, включая невычисляемый входной слой. Веса ребер, соединяющих различные слои, обозначим как w_1, w_2, \dots, w_m . Кроме того, предположим, что в каждом слое применяется сигмоидная функция активации $\Phi(\cdot)$. Пусть x — вход, $h_1 \dots h_{m-1}$ — скрытые значения в различных слоях, а o — конечный выход. Обозначим через $\Phi'(h_t)$ производную функции активации в скрытом слое t , а через $\frac{\partial L}{\partial h_t}$ — производную функции потерь по скрытой активации h_t . Соответствующая нейронная архитектура представлена на рис. 3.8. Используя правила обновления при обратном распространении ошибки, нетрудно показать, что в этом случае выполняется следующее соотношение:

$$\frac{\partial L}{\partial h_t} = \Phi'(h_{t+1}) \cdot w_{t+1} \cdot \frac{\partial L}{\partial h_{t+1}}. \quad (3.38)$$

Поскольку каждый узел имеет только один вход, предположим, что веса инициализируются из стандартного нормального распределения. Поэтому ожидаемое среднее значение для каждого веса w_t равно 1.

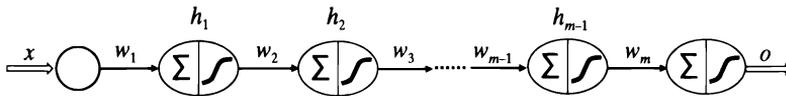


Рис. 3.8. Проблемы затухающих и взрывных градиентов

Исследуем поведение этой рекурсии в конкретном случае использования сигмоиды. Производная сигмоиды с выходом $f \in (0, 1)$ равна $f(1 - f)$. Это выражение достигает максимума при $f = 0,5$, поэтому значение $\Phi'(h_t)$ не может быть больше, чем 0,25. Так как ожидаемое значение w_{t+1} равно 1, значение $\frac{\partial L}{\partial h_t}$ при каждом обновлении веса будет составлять менее чем 0,25 от значения $\frac{\partial L}{\partial h_{t+1}}$. Поэтому после прохождения примерно r слоев это значение обычно будет меньше 0,25^r. Чтобы вы могли получить представление о порядке уменьшения этой величины, положим $r = 10$, тогда величина градиента спадет до 10^{-6} от своего первоначального значения! Таким образом, если использовать обратное распространение

ошибки, ранние слои будут обновляться очень незначительно по сравнению с поздними. Это явление известно как *проблема затухающих градиентов*. Заметьте, что мы могли бы попытаться решить проблему, используя функцию активации с большими градиентами и инициализируя веса большими значениями. Но при этом можно очень легко оказаться в прямо противоположной ситуации: вместо того чтобы уменьшаться, градиенты будут неограниченно возрастать, проявляя *взрывной* характер поведения. В общем случае величины частных производных будут вести себя крайне нестабильно, если только нам не удастся инициализировать вес каждого ребра таким образом, чтобы произведение веса и производной каждой активации не было равно ровно единице. Для большинства функций активации добиться этого на практике почти невозможно, поскольку величина производной функции активации будет варьироваться от одной итерации к другой.

Несмотря на то что мы использовали весьма упрощенный пример сети, в которой каждый слой имел только один узел, приведенные рассуждения легко обобщаются на ситуации, когда слои содержат более чем по одному узлу. В общем случае можно показать, что обновления при переходе от слоя к слою в процессе обратного распространения ошибки включают операции перемножения матриц (а не скаляров). Многократному перемножению матриц свойственна нестабильность того же типа, что и скалярам. В частности, производные функции потерь в слое $(i + 1)$ умножаются на матрицу, которая называется *якобианом* (см. уравнение 3.23). Элементами якобиана являются производные активаций в слое $(i + 1)$ по активациям в слое i . В некоторых случаях, таких как рекуррентные нейронные сети, якобиан представляет собой квадратную матрицу, для которой действительно можно определить условия стабильности относительно наибольшего собственного значения якобиана. Эти условия строго соблюдаются лишь в редких случаях, поэтому с проблемами затухающих и взрывных градиентов приходится сталкиваться довольно часто. Кроме того, сам характер поведения функций активации, таких как сигмоида, способствует возникновению проблемы затухающих градиентов. Этой проблеме можно дать следующую краткую формулировку.

Замечание 3.4.1. *Относительные величины частных производных по параметрам в различных частях сети могут заметно отличаться, что создает проблемы для методов градиентного спуска.*

В следующем разделе мы предоставим геометрическую интерпретацию естественных причин того, почему нестабильное поведение отношений градиентов порождает проблемы в большинстве задач многомерной оптимизации даже в относительно простых нейронных сетях.

3.4.1. Геометрическая интерпретация эффекта отношений градиентов

Проблемы затухающих и взрывных градиентов свойственны задачам оптимизации со многими переменными даже в условиях отсутствия локальных оптимумов. В действительности с незначительными проявлениями этих проблем приходится сталкиваться почти в любой задаче выпуклой оптимизации. Поэтому в данном разделе мы рассмотрим простейший из возможных случаев — выпуклую квадратичную целевую функцию чашеподобной формы с одним глобальным минимумом. В задачах с одной переменной траектория крутейшего спуска (которая является единственной траекторией спуска) всегда проходит через точку минимума чаши (т.е. через точку оптимума целевой функции). Однако, как только количество переменных задачи оптимизации возрастает от 1 до 2, это уже не так. Очень важно понимать, что, за очень немногими исключениями, траектория крутейшего спуска для большинства функций потерь является лишь мгновенным направлением движения в наилучшем направлении, а не подходящим направлением спуска в долгосрочной перспективе. Иными словами, в случае небольших шагов всегда требуется “грубая коррекция курса”. Если в задаче оптимизации проявляется проблема затухающего градиента, то это означает, что единственным способом достижения оптимума посредством обновлений по методу крутейшего спуска является использование *очень большого количества чрезвычайно малых обновлений и грубых поправок*, что, очевидно, весьма неэффективно.

Чтобы разобраться в этом более детально, рассмотрим две двумерные функции. На рис. 3.9 представлены контурные графики функции потерь, где каждая линия соответствует точкам плоскости XY , в которых функция потерь имеет одно и то же значение. Первая функция потерь описывается формулой $L = x^2 + y^2$, и ей соответствует идеально круглая чаша, если рассматривать высоту чаши как значение целевой функции. Переменные x и y входят в эту функцию симметричным образом. Вторая функция потерь описывается формулой $L = x + 4y^2$, и ей соответствует чаша эллиптической формы. Обратите внимание на то, что эта функция потерь более чувствительна к изменениям y по сравнению с изменениями x , хотя степень чувствительности зависит от расположения точки данных.

В случае круговой чаши (рис. 3.9, *а*) градиент направлен непосредственно в сторону оптимального решения, которое может быть достигнуто за один шаг при условии подходящего выбора его размера. Однако это совсем не так для функции потерь, изображенной на рис. 3.9, *б*, поскольку в этом случае при определении направления градиента направление y является более предпочтительным по сравнению с направлением x . Более того, градиент никогда не указывает в сторону оптимального решения, в результате чего в процессе спуска

приходится вводить множество грубых поправок. Бросается в глаза, что в u -направлении совершаются большие шаги, однако последующие шаги нивелируют эффект предыдущих. С другой стороны, продвижение в x -направлении все время осуществляется в правильную сторону, но небольшими шагами. Несмотря на то что ситуация, представленная на рис. 3.9, б, встречается почти в любой задаче оптимизации с использованием метода крутейшего спуска, затухающие градиенты представляют собой крайнюю форму проявления такого поведения². Тот факт, что простая квадратичная чаша (тривиальный случай по сравнению с типичной функцией потерь глубокой сети) демонстрирует столь малые осцилляции в методе крутейшего спуска, заслуживает интереса. В конце концов, повторная композиция функций (в соответствии с вычислительным графом) ведет себя крайне *нестабильно* в смысле чувствительности выхода к параметрам в различных частях сети. Проблема относительных различий производных крайне обостряется в реальных нейронных сетях, насчитывающих миллионы параметров и отношения градиентов, варьирующиеся на порядки величины. Кроме того, многие функции активации имеют небольшие производные, что способствует возникновению проблемы затухающих градиентов в процессе обратного распространения ошибки. Как следствие, параметры в поздних слоях с большими компонентами спуска часто осциллируют при больших величинах обновлений, тогда как параметры в ранних слоях обновляются небольшими порциями, но без осцилляций. Поэтому ни ранние, ни поздние слои не обеспечивают достаточного прогресса в отношении приближения к оптимальному решению. В результате могут возникать ситуации, когда даже длительная тренировка не позволяет заметно приблизиться к решению.

Различные типы проявления этого эффекта встречаются в тех случаях, когда ранние и поздние слои разделяют параметры. В подобных ситуациях результат обновления может быть крайне непредсказуемым ввиду комбинированного эффекта различных слоев. Подобные сценарии встречаются в рекуррентных нейронных сетях, в которых параметры, используемые в поздних слоях,

² Различные типы проявления этого эффекта встречаются при разделении параметров ранними и поздними слоями. В подобных случаях результат обновления может быть в высшей степени непредсказуемым ввиду комбинированного эффекта различных слоев. Подобные сценарии встречаются в рекуррентных нейронных сетях, в которых параметры, используемые в поздних слоях, привязываются к параметрам из ранних временных слоев. В таких случаях небольшие изменения параметров могут приводить к большим изменениям функции потерь в узко локализованных областях без каких-либо указаний на возможные проблемы с градиентом в прилежащих областях. Подобного типа топологические характеристики функции потерь называют утесами (раздел 3.5.4), и они создают еще более трудные проблемы для оптимизации, поскольку градиентный спуск может либо не дойти до оптимума, либо перескочить через него.

привязываются к параметрам из ранних временных слоев. В таких случаях небольшие изменения параметров могут приводить к большим изменениям функции потерь в узко локализованных областях без каких-либо указаний на возможные проблемы с градиентом в прилежащих областях. Подобного типа топологические характеристики функции потерь называют *утесами* (раздел 3.5.4), и они создают еще более трудные проблемы для оптимизации, поскольку градиентный спуск может либо не добраться до оптимума, либо переключиться через него.

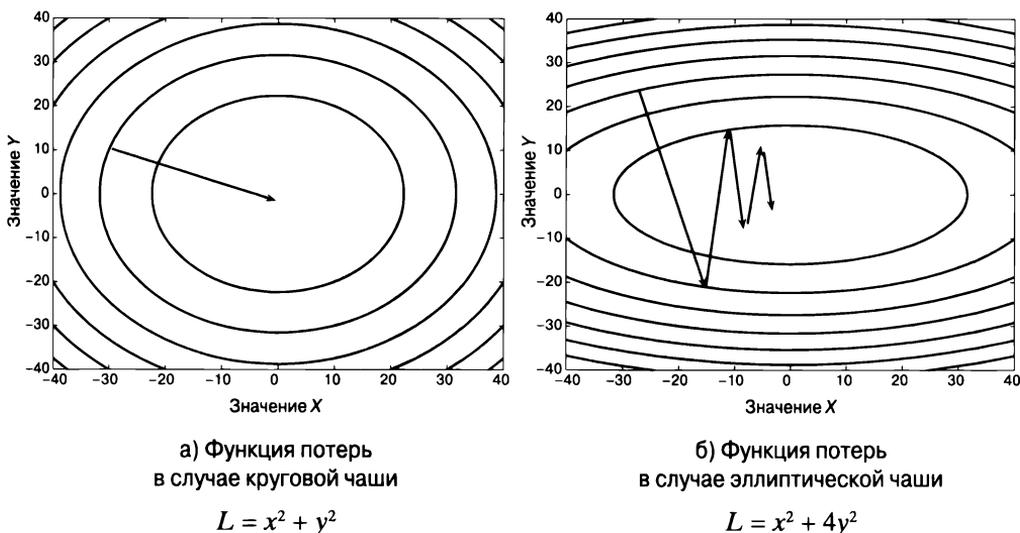


Рис. 3.9. Влияние формы функции потерь на направление крутейшего спуска

3.4.2. Частичное устранение проблем за счет выбора функции активации

Удачный выбор функции активации часто позволяет снизить остроту проблемы затухающих градиентов. Графики производных сигмоиды и гиперболического тангенса приведены на рис. 3.10, *а* и *б* соответственно. Градиент сигмоиды никогда не превышает значения 0,25 и поэтому подвержен проблеме затухающих градиентов. Кроме того, он насыщается (т.е. принимает почти нулевое значение) при больших значениях аргумента. В подобных случаях веса нейрона изменяются очень медленно. Наличие нескольких таких активаций в сети способно значительно повлиять на вычисление градиента. Гиперболический тангенс ведет себя несколько лучше, чем сигмоида, поскольку его градиент равен 1 вблизи начала координат, однако и он быстро насыщается при увеличении абсолютной величины аргумента. Следовательно, гиперболический тангенс также будет подвержен проблеме затухающих градиентов.

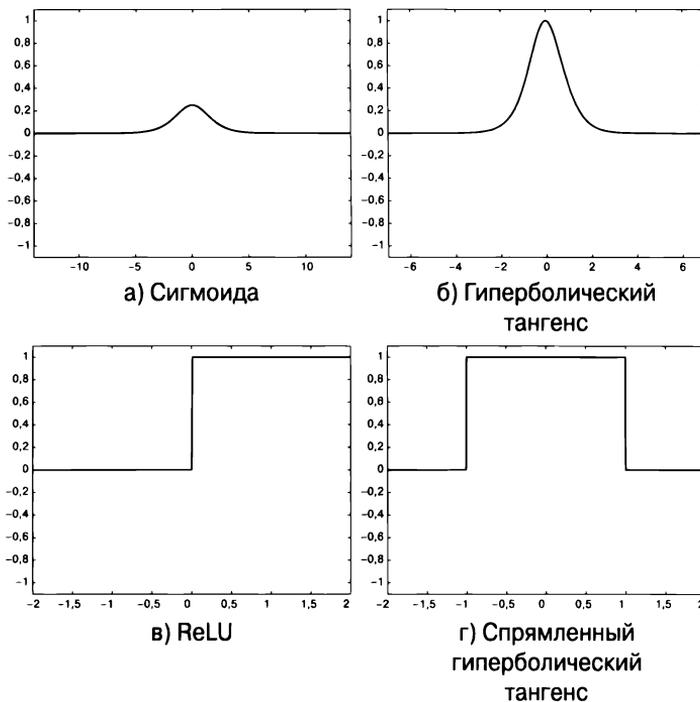


Рис. 3.10. Графики производных различных функций активации. Значения локальных градиентов кусочно-линейных функций равны 1

В последние годы сигмоида и гиперболический тангенс как функции активации были заметно потеснены функцией ReLU и спряmlенным гиперболическим тангенсом. Кроме того, процесс обучения с функцией ReLU проходит быстрее ввиду эффективности вычисления ее градиента. Графики производных ReLU и спряmlенного гиперболического тангенса приведены на рис. 3.10, в и г соответственно. Очевидно, что в некоторых интервалах аргумента производные этих функций равны 1. Правда, в других интервалах они могут иметь нулевые градиенты. Как следствие, если большинство элементов работает в пределах благоприятных интервалов, где градиент равен 1, то проблема затухающих градиентов будет возникать реже. В последние годы эти кусочно-линейные варианты становятся гораздо более популярными, чем их гладкие аналоги. Обратите внимание на то, что замена функции активации — это всего лишь полумера, поскольку операции матричного умножения в слоях по-прежнему обуславливают определенный уровень нестабильности. Кроме того, кусочно-линейные функции активации создают новые трудности — проблему так называемых *мертвых* нейронов (dead neurons).

3.4.3. Мертвые нейроны и “повреждение мозга”

На рис. 3.10, в, отчетливо видно, что градиент функции активации ReLU равен нулю для отрицательных значений аргумента. Такая ситуация может возникать по целому ряду причин. Рассмотрим, например, случай, когда вход нейрона всегда неотрицателен, но веса были инициализированы отрицательными значениями. Поэтому выход такого нейрона всегда будет нулевым. Еще одной причиной может быть завышенная скорость обучения. В таком случае преактивационные значения ReLU могут оказаться в диапазоне, в котором градиент равен нулю независимо от входа. Иными словами, высокие скорости обучения могут “выводить из игры” элементы ReLU. В подобных случаях элемент ReLU может вообще не запускаться ни для каких примеров данных. Как только нейрон достигает этой точки, градиент функции потерь по весам непосредственно перед применением к нему ReLU всегда будет равен нулю. Иными словами, веса данного нейрона уже никогда не будут обновляться в процессе обучения. К тому же при изменении входа его выход не будет изменяться, поэтому его участие в дальнейшем анализе данных будет вообще исключено. Такой нейрон можно считать *мертвым*, а создавшуюся ситуацию, говоря языком биологии, можно назвать “повреждением мозга”. Проблему умирающих нейронов можно несколько смягчить, используя умеренные скорости обучения. Для устранения описанной проблемы также можно попытаться использовать так называемые *ReLU с утечкой* (leaky ReLU), которые позволяют нейронам, находящимся за пределами активного интервала, передавать градиент в обратном направлении.

3.4.3.1. ReLU с утечкой

Определение ReLU с утечкой включает дополнительный параметр $\alpha \in (0, 1)$:

$$\Phi(v) = \begin{cases} \alpha \cdot v, & \text{если } v \leq 0, \\ 0 & \text{в противном случае} \end{cases} \quad (3.39)$$

Несмотря на то что α — гиперпараметр, выбираемый пользователем, он также может уточняться в процессе обучения. Поэтому и при отрицательных значениях v ReLU с утечкой может распространять градиент в обратном направлении, пусть даже это будет некоторая его доля, определяемая параметром $\alpha < 1$.

Применение ReLU с утечкой не гарантирует получения положительного результата, так что данное средство нельзя считать абсолютно надежным. Важно то, что мертвые нейроны не всегда создают проблемы, поскольку они представляют своего рода способ отсечения части нейронов, обеспечивающий точное управление структурой нейронной сети. Поэтому отбрасывание некоторой части нейронов можно рассматривать как часть процесса обучения. В конце

концов, наши возможности тонкой настройки количества нейронов в каждом слое ограничены. Мертвые нейроны выполняют за нас эту часть тонкой настройки. В действительности намеренное отсечение некоторых связей иногда используется в качестве стратегии регуляризации [282]. Разумеется, если доля мертвых нейронов в сети велика, то это также может создавать проблемы, поскольку в таком случае значительная часть сети станет неактивной. Кроме того, если модель не очень удачна, то исключение слишком большого количества нейронов на ранних стадиях обучения также нежелательно.

3.4.3.2. Сети Maxout

Одним из недавно предложенных решений описанных проблем является использование *maxout-сетей* [148]. В основе maxout-элементов лежит идея использования двух векторов коэффициентов, \bar{W}_1 и \bar{W}_2 , вместо одного. В дальнейшем в качестве активации используется функция $\max\{\bar{W}_1 \cdot \bar{X}, \bar{W}_2 \cdot \bar{X}\}$. В случае использования нейронов смещения в качестве maxout-активации берется функция $\max\{\bar{W}_1 \cdot \bar{X} + b_1, \bar{W}_2 \cdot \bar{X} + b_2\}$. Элемент *maxout* можно рассматривать как обобщение элемента ReLU, поскольку ReLU можно получить из него, установив один из векторов коэффициентов равным нулю. Можно показать, что даже ReLU с уткой является частным случаем элемента *maxout*, в котором $\bar{W}_2 = \alpha \bar{W}_1$ для $\alpha \in (0, 1)$. Как и ReLU, функция *maxout* — кусочно-линейная. Однако она вовсе не насыщается и линейна почти везде. Несмотря на то что эта функция линейна, в [148] было показано, что maxout-сети являются универсальными аппроксиматорами функций. Функция *maxout* обладает рядом преимуществ по сравнению с функцией ReLU и улучшает работу ансамблевых методов наподобие *Dropout* (раздел 4.5.4). Единственным недостатком является то, что их использование удваивает количество необходимых параметров.

3.5. Стратегии градиентного спуска

Самым распространенным методом обучения параметров в нейронных сетях является *метод крутейшего спуска*, в котором для обновления параметров используется градиент функции потерь. Фактически все обсуждения в предыдущих главах основывались на этом предположении. Как демонстрировалось в предыдущем разделе, иногда этот метод может вести себя непредсказуемым образом, поскольку он не всегда указывает наилучшее направление, если речь идет о шагах конечного размера. Направление крутейшего спуска является оптимальным лишь с точки зрения шагов бесконечно малого размера. Иногда направление крутейшего спуска становится направлением подъема после незначительного обновления параметров. Вследствие этого требуется вводить многочисленные грубые поправки. Конкретный пример, когда небольшие различия в чувствительности к определенным признакам могут вызывать

осцилляции алгоритма крутейшего спуска, был приведен в разделе 3.4.1. С проблемой осцилляции и зигзагообразного поведения постоянно приходится сталкиваться в тех случаях, когда направление крутейшего спуска совпадает с направлением *высокой кривизны* поверхности функции потерь. Крайние формы проявления такого поведения встречаются в случае неудачного выбора значений параметров, когда производные функции потерь по различным переменным оптимизации значительно отличаются по своей величине. В этом разделе мы обсудим эффективные стратегии обучения, которые работают удовлетворительно даже при неблагоприятных условиях.

3.5.1. Регулирование скорости обучения

Постоянная скорость обучения нежелательна по той причине, что это ставит перед аналитиком дилемму, суть которой заключается в следующем. Использование низкой скорости обучения на ранних стадиях приведет к тому, что даже для того, чтобы приблизиться к оптимальному решению, алгоритму придется затратить слишком много времени. С другой стороны, высокая скорость обучения позволит алгоритму достаточно быстро приблизиться к удовлетворительному решению. Однако сохранение постоянной скорости обучения в дальнейшем может приводить к осцилляциям алгоритма в окрестности оптимальной точки в течение длительного времени и препятствовать его сходимости. В любом случае сохранение постоянной скорости обучения не является идеальным вариантом. Естественный способ, позволяющий избежать этих трудностей, заключается в том, чтобы использовать скорость обучения, которой разрешено уменьшаться с течением времени.

Двумя наиболее распространенными функциями уменьшения скорости обучения являются *экспоненциальный спад* (exponential decay) и *обратный спад* (inverse decay). Скорость обучения α_t можно выразить в терминах начальной скорости обучения α_0 и порядкового номера эпохи t следующим образом:

$$\alpha_t = \alpha_0 \exp(-k \cdot t) \quad [\text{экспоненциальный спад}],$$

$$\alpha_t = \frac{\alpha_0}{1 + k \cdot t} \quad [\text{обратный спад}].$$

Параметр k контролирует скорость спада. Также существует метод, предполагающий *пошаговый спад* (step decay), когда скорость обучения уменьшается в определенное количество раз каждые несколько эпох. Например, скорость обучения может умножаться на 0,5 каждые 5 эпох. Обычный подход заключается в отслеживании функции потерь на зарезервированной части тренировочного набора данных и уменьшении скорости обучения сразу же после того, как функция потерь перестает улучшаться. В некоторых случаях аналитик может

самостоятельно присматривать за ходом процесса и использовать реализацию, позволяющую вручную изменять скорость обучения в зависимости от того, как оно проходит. Такой подход может применяться в простых реализациях градиентного спуска, хотя он и не в состоянии справиться со многими другими проблемами.

3.5.2. Метод импульсов

Различные приемы, основанные на методе импульсов, позволяют распознавать, что зигзагообразный характер траектории оптимизации является следствием выполнения противоречивых шагов, взаимно погашающих получаемые с их помощью результаты, и уменьшать эффективный размер шагов в корректном (в долгосрочной перспективе) направлении. Пример сценария подобного типа приведен на рис. 3.9, б. Простая попытка увеличить размер шага для перемещения на большее расстояние в корректном направлении фактически может увести текущее решение еще дальше от оптимума. С этой точки зрения гораздо больше смысла имеет перемещение в направлении, усредненном по последним нескольким шагам для сглаживания зигзагов.

Чтобы в этом разобраться, рассмотрим ситуацию, в которой градиентный спуск выполняется по отношению к вектору параметров \bar{W} . Обычная формула обновления при градиентном спуске для функции потерь L (определенной на мини-пакете примеров) выглядит так:

$$\bar{V} \leftarrow -\alpha \frac{\partial L}{\partial \bar{W}}; \bar{W} \leftarrow \bar{W} + \bar{V},$$

где α — скорость обучения. В случае градиентного спуска на основе импульса вектор \bar{V} модифицируется экспоненциальным сглаживанием:

$$\bar{V} \leftarrow \beta \bar{V} - \alpha \frac{\partial L}{\partial \bar{W}}; \bar{W} \leftarrow \bar{W} + \bar{V},$$

где $\beta \in (0, 1)$ — параметр сглаживания. Большие значения β способствуют выбору согласованных значений скорости \bar{V} в корректном направлении. Значение $\beta = 0$ соответствует непосредственному мини-пакетному градиентному спуску. Параметр β также называют *параметром импульса* или *параметром трения*. Ссылка на “трение” объясняется тем, что небольшие значения β действуют как “тормоза”, во многом подобно трению.

При градиентном спуске на основе импульса обучение ускоряется, поскольку в этом случае движение обычно осуществляется в направлении, которое часто ориентировано главным образом в сторону оптимального решения и подавляет бесполезное “рыскание”. Суть идеи заключается в том, чтобы отдавать

большее предпочтение согласованным направлениям движения на протяжении нескольких шагов, что имеет важное значение при спуске. Это позволяет использовать большие шаги в подходящем направлении без перескоков и чрезмерных отклонений в боковых направлениях. В результате обучение ускоряется. Как отчетливо видно на рис. 3.11, *а*, импульс увеличивает компоненту градиента в корректном направлении. Как это влияет на обновления, показано на рис. 3.11, *б* и *в*. Очевидно, что обновления на основе импульса позволяют достигать оптимального решения за меньшее количество шагов.

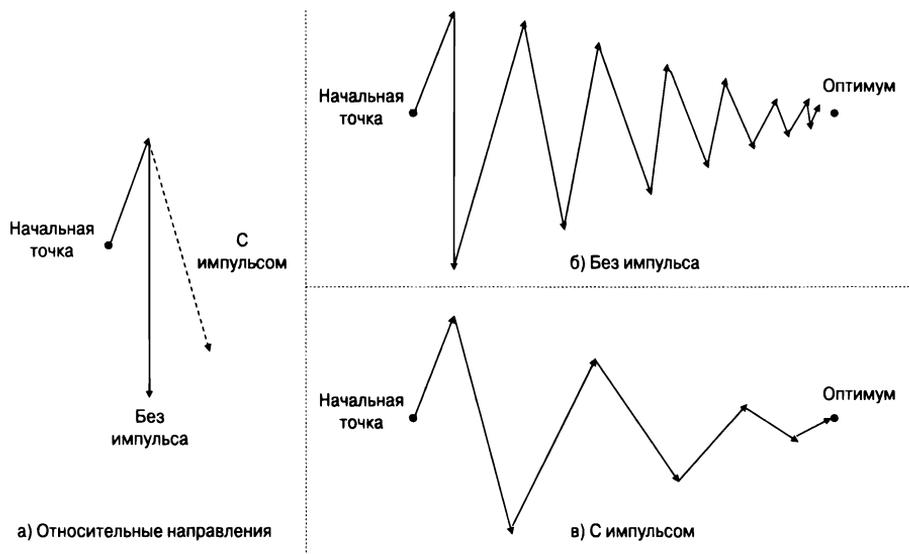


Рис. 3.11. Роль импульса в сглаживании зигзагообразных обновлений

Использование импульса часто приводит к незначительным перескокам через оптимальное решение в направлении выбранной скорости подобно тому, как шарик, скатывающийся по стенке округлой чаши, проскакивает дно и переходит на ее противоположную сторону. Однако при надлежащем выборе β ситуация все равно будет более благоприятная, чем в том случае, когда импульс не используется. Обычно метод на основе импульса будет работать лучше, поскольку шарик набирает скорость по мере скатывания по стенке чаши. Ускоренные темпы достижения оптимального решения с лихвой компенсируют недостатки возможных перескоков через цель. Перескоки желательны в той мере, в какой они помогают избежать попадания в локальные оптимумы. Эту мысль поясняет рис. 3.12, на котором изображен шарик, скатывающийся по поверхности функции потерь сложной формы (и постепенно набирающий скорость). Благодаря набору скорости шарик удается эффективно проходить плоские участки этой поверхности. Параметр β управляет силой трения, которую испытывает шарик со стороны поверхности. Несмотря на то что увеличение значения β помогает

избегать локальных оптимумов, это может привести к осцилляциям вокруг конечной точки. Описанная картина служит изящной интерпретацией метода импульсов в терминах физики движения шарика, скатывающегося по поверхности сложной функции потерь.



Рис. 3.12. Роль импульса в навигации по сложным поверхностям функции потерь. Импульс содействует сохранению постоянной скорости процесса оптимизации на плоских участках поверхности функции потерь и помогает избежать попадания в локальные оптимумы

3.5.2.1. Метод импульсов Нестерова

Метод импульсов Нестерова [353] — это модифицированный вариант традиционного метода импульсов, предполагающий вычисление градиентов в той точке, которая была бы достигнута в результате повторного применения β -версии предыдущего шага (т.е. части текущего шага, связанной с импульсом). Эта точка получается путем умножения предыдущего вектора \vec{V} на параметр трения β и последующего вычисления градиента в точке $\vec{W} + \beta\vec{V}$. Идея заключается в том, что скорректированный градиент учитывает, как изменятся градиенты благодаря части обновления, связанной с импульсом, и включает эту информацию в градиентную часть обновления. Тем самым при вычислении обновлений в некоторой степени учитывается прогнозная информация. Пусть $L(\vec{W})$ — функция потерь в точке, соответствующей текущему решению \vec{W} . В данном случае, учитывая способ вычисления градиента, важно явно указывать аргумент функции потерь. Поэтому обновление вычисляется следующим образом:

$$\vec{V} \leftarrow \beta\vec{V} - \alpha \frac{\partial L(\vec{W} + \beta\vec{V})}{\partial \vec{W}}; \vec{W} \leftarrow \vec{W} + \vec{V}.$$

Заметьте, что единственное отличие метода Нестерова от стандартного метода импульсов обусловлено членом, в котором вычисляется градиент. Использо-

вание значения градиента, вычисленного в точке, незначительно опережающей текущую в направлении предыдущего обновления, может ускорить сходимость. Возвращаясь к предыдущей аналогии с шариком, можно сказать, что этот подход начинает применять “тормоза” в процедуре градиентного спуска тогда, когда шарик приближается к дну чаши, поскольку опережающий просмотр “предупреждает” об обращении направления градиента.

Метод Нестерова хорошо работает только в случае мини-пакетного градиентного спуска с умеренными размерами пакетов. Использовать очень малые пакеты — плохая идея. Можно показать, что в подобных случаях метод Нестерова уменьшает ошибку до уровня $O(1/t^2)$ после t шагов, что можно сравнить со сложностью ошибки $O(1/t)$ в методе импульсов.

3.5.3. Скорости обучения, специфические для параметров

Базовая идея методов импульса, которые обсуждались в предыдущем разделе, — использование преимуществ *согласованности* направлений градиентов определенных параметров для ускорения процесса их обновления. Того же эффекта можно добиться более явным способом, назначая разным параметрам разные скорости обучения. Эта идея базируется на том наблюдении, что параметры с большими частными производными часто осциллируют и обновляются зигзагообразно, в то время как параметры с небольшими частными производными обычно ведут себя более согласованно, но при этом изменяются в одном и том же направлении. Одним из ранних методов, использовавших эту идею, был метод *delta-bar-delta* [217]. В нем отслеживается смена знака каждой частной производной. Если производная сохраняет свой знак, то это свидетельствует о том, что данное направление является подходящим, и тогда частная производная в этом направлении увеличивается. С другой стороны, если знак частной производной каждый раз меняется на противоположный, то частная производная уменьшается. Однако подход такого типа предназначен для простого, а не стохастического градиентного спуска, поскольку в последнем случае ошибки могут увеличиваться. В связи с этим был предложен ряд методов, способных работать даже тогда, когда используется мини-пакетный метод.

3.5.3.1. Алгоритм AdaGrad

В алгоритме *AdaGrad* [108] отслеживается агрегированная квадратичная величина частной производной по каждому параметру в процессе работы. Квадратный корень из этого значения *пропорционален* среднеквадратическому наклону для этого параметра (хотя его абсолютная величина будет увеличиваться с увеличением количества эпох ввиду последовательной агрегации).

Обозначим через A_i агрегированное значение для i -го параметра. Тогда на каждой итерации выполняется следующее обновление:

$$A_i \leftarrow A_i + \left(\frac{\partial L}{\partial w_i} \right)^2 \quad \forall i. \quad (3.40)$$

Обновление для i -го параметра w_i выглядит следующим образом:

$$w_i \leftarrow w_i - \frac{\alpha}{\sqrt{A_i}} \left(\frac{\partial L}{\partial w_i} \right) \quad \forall i.$$

Если хотите, используйте в знаменателе $\sqrt{A_i + \varepsilon}$ вместо $\sqrt{A_i}$, чтобы избежать деления на нуль. Здесь ε — небольшое положительное значение, например 10^{-8} .

Масштабирование производной в обратной пропорции к $\sqrt{A_i}$ является своего рода нормализацией отношения “сигнал/шум”, поскольку A_i измеряет только историческую величину градиента, а не его знак. Оно стимулирует более быстрое *относительное* перемещение вдоль направлений, характеризующихся умеренным уклоном с постоянным знаком градиента. Если компонента градиента вдоль i -го направления беспорядочно флуктуирует между значениями $+100$ и -100 , то этот тип нормализации штрафует данную компоненту в гораздо большей степени, чем другую, которая последовательно принимает значения в окрестности $(0, 1)$, но с постоянным знаком. Например, представленному на рис. 3.11 перемещению в осциллирующем направлении отдастся меньшее предпочтение, чем перемещению в систематически сохраняющемся направлении. Однако абсолютная величина перемещений вдоль всех компонент со временем будет уменьшаться, что является основной проблемой данного подхода. Это объясняется тем, что A_i представляет *агрегированное* значение, отражающее всю предысторию частных производных, что приводит к уменьшению значений масштабированной производной. Как следствие, продвижение алгоритма AdaGrad может преждевременно замедлиться и, наконец, вообще прекратиться. Другая проблема состоит в том, что агрегированные масштабные множители зависят от предыстории, которая в конечном счете будет содержать много устаревших данных. Использование далеко не актуальных значений масштабных множителей может вносить дополнительную неточность. Как будет показано далее, в большинстве других методов используется экспоненциальное усреднение, позволяющее решить обе проблемы.

3.5.3.2. Алгоритм RMSProp

Алгоритм *RMSProp* [194] использует ту же предпосылку, что и алгоритм AdaGrad, для нормализации отношения “сигнал/шум” с помощью абсолютной величины $\sqrt{A_i}$ градиентов. Однако, вместо того чтобы просто складывать ква-

драты градиентов для оценки A_i , в нем применяется экспоненциальное усреднение. Поскольку усреднение используется для нормализации, а не агрегирования значений, продвижение алгоритма не замедляется преждевременно постоянно увеличивающимся масштабным множителем A_i . Базовая идея заключается в том, чтобы использовать фактор затухания $\rho \in (0, 1)$ и наделять квадраты частных производных, отстоящих назад на t обновлений, весом ρ^t . Заметьте, что этого можно легко добиться, умножая текущий агрегированный квадрат (т.е. текущую оценку) на ρ и добавляя текущее (возведенное в квадрат) значение частной производной $(1 - \rho)$ раз. Текущая оценка инициализируется нулевым значением. Это приводит к некоторому (нежелательному) смещению на ранних итерациях, которое исчезает в долгосрочной перспективе. Поэтому, если A_i — экспоненциально усредненное значение i -го параметра w_i , то мы получаем следующее правило обновления A_i :

$$A_i \leftarrow \rho A_i + (1 - \rho) \left(\frac{\partial L}{\partial w_i} \right)^2 \quad \forall i. \quad (3.41)$$

Квадратный корень из этой величины для каждого параметра используется для нормализации его градиента. Тогда для (глобальной) скорости обучения α будет справедливо следующее соотношение:

$$w_i \leftarrow w_i - \frac{\alpha}{\sqrt{A_i}} \left(\frac{\partial L}{\partial w_i} \right) \quad \forall i.$$

Если хотите, используйте в знаменателе $\sqrt{A_i + \epsilon}$ вместо $\sqrt{A_i}$, чтобы избежать деления на нуль. Здесь ϵ — небольшое положительное значение, например 10^{-8} . Дополнительное преимущество алгоритма RMSProp по сравнению с алгоритмом AdaGrad состоит в том, что значимость исторических (т.е. устаревших) градиентов экспоненциально спадает со временем. Кроме того, этот подход допускает включение понятия импульса в вычислительный алгоритм (разделы 3.5.3.3 и 3.5.3.5). Недостатком алгоритма RMSProp является то, что текущая оценка A_i момента второго порядка смещается на ранних итерациях из-за его инициализации нулевым значением.

3.5.3.3. Комбинация алгоритма RMSProp и метода Нестерова

Алгоритм RMSProp также может сочетаться с методом импульсов Нестерова. Обозначим через A_i квадрат агрегированного значения i -го веса. В подобных случаях мы вводим дополнительный параметр $\beta \in (0, 1)$ и используем следующее правило обновления:

$$v_i \leftarrow \beta v_i - \frac{\alpha}{\sqrt{A_i}} \left(\frac{\partial L(\bar{W} + \beta \bar{V})}{\partial w_i} \right); \quad w_i \leftarrow w_i + v_i \quad \forall i.$$

Обратите внимание на то, что частная производная функции потерь вычисляется в смещенной точке, что присуще методу Нестерова. При вычислении частной производной функции потерь вес \bar{W} смещается на величину $\beta\bar{V}$. Вычисление A_i также осуществляется с использованием смещенных градиентов:

$$A_i \leftarrow \rho A_i + (1 - \rho) \left(\frac{\partial L(\bar{W} + \beta\bar{V})}{\partial w_i} \right)^2 \quad \forall i. \quad (3.42)$$

Несмотря на то что данный подход использует преимущества, проистекающие из добавления импульса в алгоритм RMSProp, он не вводит поправки на смещение инициализации.

3.5.3.4. Алгоритм AdaDelta

Алгоритм *AdaDelta* [553] использует то же правило обновления, что и алгоритм RMSProp, за исключением того, что он устраняет необходимость во введении глобального параметра скорости обучения и вычисляет его как функцию инкрементных обновлений на предыдущих итерациях. Обратимся к правилу обновления RMSProp, которое для удобства повторно приведено ниже:

$$w_i \leftarrow w_i - \underbrace{\frac{\alpha}{\sqrt{A_i}} \left(\frac{\partial L}{\partial w_i} \right)}_{\Delta w_i} \quad \forall i.$$

Покажем, каким образом можно заместить α значением, которое зависит от предыдущих инкрементных обновлений. При каждом обновлении значение w_i инкрементируется значением Δw_i . Как и в случае экспоненциально сглаженных градиентов A_i , мы удерживаем экспоненциально сглаженное значение δ_i значений Δw_i на предыдущих итерациях с тем же параметром затухания ρ :

$$\delta_i \leftarrow \rho \delta_i + (1 - \rho) (\Delta w_i)^2 \quad \forall i. \quad (3.43)$$

Значение δ_i для каждой итерации можно вычислить только на основании результатов предыдущих итераций, поскольку значение Δw_i пока что недоступно. С другой стороны, A_i можно вычислить, используя частные производные текущей итерации. Это тонкое различие в способах вычисления A_i и δ_i . В результате мы получаем следующее правило обновления AdaDelta:

$$w_i \leftarrow w_i - \underbrace{\sqrt{\frac{\delta_i}{A_i}} \left(\frac{\partial L}{\partial w_i} \right)}_{\Delta w_i} \quad \forall i.$$

Следует отметить, что параметр α , определяющий скорость обучения, отсутствует в этом правиле. Метод AdaDelta имеет некоторые общие черты с

другими методами второго порядка, поскольку отношение $\sqrt{\frac{\delta_i}{A_i}}$, входящее в правило обновления, является эвристическим суррогатом обратной величины производной функции потерь по w_i [553]. Как обсуждается в последующих разделах, скорость обучения не используется во многих методах второго порядка, таких как метод Ньютона.

3.5.3.5. Алгоритм Adam

Как и алгоритмы AdaGrad и RMSProp, алгоритм Adam использует аналогичную нормализацию отношения “сигнал/шум”, однако для включения импульса в обновления применяется экспоненциальное сглаживание градиента первого порядка. Кроме того, в этом алгоритме непосредственно решается проблема смещения, присущая экспоненциальному сглаживанию, если текущая оценка сглаженного значения нереалистически инициализируется нулевым значением.

Как и в случае алгоритма RMSProp, обозначим через A_i экспоненциально усредненное значение i -го параметра w_i . Это значение обновляется с использованием параметра затухания $\rho \in (0, 1)$ так же, как и в алгоритме RMSProp:

$$A_i \leftarrow \rho A_i + (1 - \rho) \left(\frac{\partial L}{\partial w_i} \right)^2 \quad \forall i. \quad (3.44)$$

Одновременно поддерживается и экспоненциально сглаженное значение градиента, i -ю компоненту которого мы обозначим через F_i . Это сглаживание осуществляется с помощью другого параметра, ρ_f :

$$F_i \leftarrow \rho_f F_i + (1 - \rho_f) \left(\frac{\partial L}{\partial w_i} \right) \quad \forall i. \quad (3.45)$$

Данный тип экспоненциального сглаживания градиента с помощью параметра ρ_f представляет собой разновидность метода импульсов, рассмотренного в разделе 3.5.2 (который параметризуется параметром трения β , а не параметром ρ_f). Затем на t -й итерации используется следующая скорость обучения α_t :

$$w_i \leftarrow w_i - \frac{\alpha_t}{\sqrt{A_i}} F_i \quad \forall i.$$

В данном случае мы имеем дело с двумя основными отличиями от алгоритма RMSProp. Во-первых, градиент заменяется его экспоненциально сглаженным значением для включения импульса. Во-вторых, скорость обучения α_t теперь зависит от номера итерации t и определяется следующим образом:

$$\alpha_t = \alpha \underbrace{\left(\frac{\sqrt{1 - \rho^t}}{1 - \rho_f^t} \right)}_{\text{Коррекция смещения}}. \quad (3.46)$$

Регулирование скорости обучения технически осуществляется с помощью поправочного множителя смещения, применяемого для учета нереалистической инициализации двух механизмов сглаживания, что особенно важно на ранних итерациях. Как F_i , так и A_i инициализируются нулевыми значениями, вызывающими смещение на ранних итерациях. Учитывая отношение, приведенное в уравнении 3.46, смещение по-разному влияет на эти две величины. Примечательно, что и ρ_i , и ρ_f^t сходятся к 0 для больших t , поскольку $\rho_f \in (0, 1)$. В результате приведенный в уравнении 3.46 множитель, корректирующий смещение инициализации, сходится к 1, а α_t сходится к α . В оригинальной статье [241] для параметров ρ_f и ρ были предложены значения по умолчанию, равные соответственно 0,9 и 0,999. Более подробно о других критериях (таких, как разреженность параметров), используемых для выбора значений ρ и ρ_f , см. в [241]. Подобно другим методам, алгоритм Adam использует $\sqrt{A_i + \varepsilon}$ (вместо $\sqrt{A_i}$) в знаменателе выражения для обновления, чтобы избежать деления на нуль. Алгоритм Adam чрезвычайно популярен, поскольку включает большинство преимуществ, предлагаемых другими алгоритмами, и нередко может составить конкуренцию лучшим из них [241].

3.5.4. Овраги и нестабильность более высокого порядка

До сих пор в главе обсуждались только производные первого порядка. Для некоторых поверхностей функции ошибок алгоритмы, использующие первые производные, могут продвигаться слишком медленно. Частично эта проблема обусловлена тем, что производные первого порядка предоставляют ограниченную информацию о поверхности функции ошибки, в результате чего обновления могут перепрыгивать через решение. Сложность поверхностей функции потерь многих нейронных сетей может приводить к непредсказуемому поведению обновлений, основанных на градиентах.

Пример поверхности функции потерь приведен на рис. 3.13. В данном случае существует участок поверхности с плавным уклоном, который резко переходит в отвесный овраг. Однако, если вычислять лишь частную производную первого порядка по переменной x , то мы увидим только плавный склон. В результате небольшое значение скорости обучения значительно замедлит процесс, тогда как увеличение этого значения может привести к быстрому переходу в точку, находящуюся далеко от оптимального решения. Рассмотренная

проблема обусловлена природой кривизны (т.е. изменения градиента), относительно которой градиент первого порядка не несет в себе никакой информации, необходимой для управления размером обновления. Во многих случаях скорость изменения градиента можно вычислить с помощью производной второго порядка, которая предоставляет полезную (дополнительную) информацию. В общем случае методы второго порядка аппроксимируют локальную поверхность функции потерь квадратичной поверхностью, более точной по сравнению с линейным приближением. Некоторым методам второго порядка, таким как метод Ньютона, требуется ровно одна итерация для того, чтобы найти оптимальное решение для квадратичной поверхности. Разумеется, типичные функции потерь нейронных моделей не являются квадратичными. Тем не менее этой аппроксимации часто достаточно для того, чтобы значительно ускорить работу методов градиентного спуска, по крайней мере, в тех случаях, когда отсутствуют слишком резкие или внезапные изменения градиента.

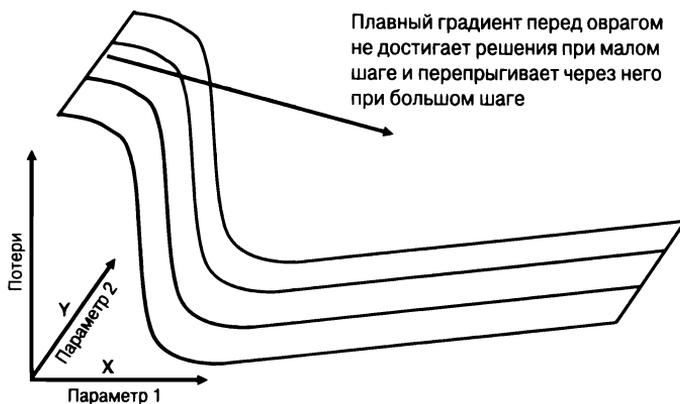


Рис. 3.13. Пример оврага на поверхности функции потерь

Овраги нежелательны, поскольку они приводят к определенному уровню нестабильности в поведении функции потерь. Это означает, что небольшие изменения некоторой части весов могут привести либо к незначительным, либо к резким изменениям функции потерь такой большой величины, что результирующее решение оказывается слишком далеким от истинного оптимума. Как будет показано в главе 7, все временные слои рекуррентной нейронной сети разделяют одни и те же параметры. В этом случае появление затухающих или взрывных градиентов означает неравномерную чувствительность функции потерь к параметрам ранних и поздних слоев (которые так или иначе связаны между собой). Поэтому небольшое изменение хорошо подобранного параметра может распространиться по каскаду слоев неустойчивым способом и либо приобрести взрывной характер поведения, либо оказать пренебрежимо малое влияние на значение функции потерь. Кроме того, нелегко найти такой способ

управления размером шага, который позволил бы избежать обоих этих событий. Описанное поведение типично вблизи оврагов. В результате можно легко пропустить оптимум на шаге градиентного спуска. Такое поведение можно объяснить тем, что разделение параметров несколькими слоями естественным образом порождает эффекты возмущения весов функции потерь второго порядка. Причиной является перемножение разделяемых весов различных слоев в процессе прогнозирования, и в этих условиях градиента первого порядка оказывается недостаточно для того, чтобы моделировать эффект кривизны функции потерь, являющейся мерой изменения градиента вдоль определенного направления. Для преодоления указанных проблем часто задействуют методику, основанную либо на отсечении градиента, либо на явном использовании кривизны (т.е. производных второго порядка) функции потерь.

3.5.5. Отсечение градиентов

Отсечение градиентов (gradient clipping) — методика, которую применяют в тех случаях, когда величины частных производных в различных направлениях могут значительно различаться между собой. Некоторые методы отсечения градиентов пытаются сделать поведение различных компонент частных производных более равномерным, используя принципы, аналогичные тем, с которыми вы познакомились при рассмотрении методов адаптивного изменения скорости обучения. Однако отсечение градиентов выполняется только на основе их текущих, а не предыдущих значений. Наиболее распространены два способа отсечения градиентов.

1. *Отсечение градиентов на основе значений* (value-based clipping). В этом методе для значений градиента устанавливаются минимальное и максимальное пороговые значения. Для всех частных производных ниже минимума устанавливается минимальное пороговое значение. Для всех частных производных, превышающих максимум, устанавливается максимальное пороговое значение.
2. *Отсечение градиентов на основе норм* (norm-based clipping). В этом методе вектор полного градиента нормализуется с использованием L_2 -нормы всего вектора. Обратите внимание на то, что данный метод отсечения не изменяет величин обновлений вдоль различных направлений. Однако для нейронных сетей, в которых параметры разделяются различными слоями (как в рекуррентных нейронных сетях), эффекты, оказываемые обоими типами отсечения, весьма сходны. Отсечение позволяет упорядочивать значения таким образом, что при переходе от одного мини-пакета к другому обновления остаются в целом примерно одинаковыми. Тем самым аномально большие значения градиента для отдельного мини-пакета не оказывают существенного влияния на решение.

Вообще говоря, эффекты отсечения градиентов довольно ограничены по сравнению с другими методами. Особенно эффективно применение этого метода в рекуррентных нейронных сетях для устранения проблемы взрывных градиентов (глава 7). В таких сетях параметры разделяются различными слоями, но производные по разделяемым параметрам в каждом слое вычисляются так, как если бы они были независимыми переменными. Эти производные являются временными компонентами общего градиента, и значения отсекаются еще до того, как они будут суммироваться для получения общего градиента. Геометрическая интерпретация проблемы взрывных градиентов приведена в [369], а подробное рассмотрение того, как работает отсечение градиентов, можно найти в [368].

3.5.6. Производные второго порядка

В последние годы был предложен ряд методов оптимизации с использованием вторых производных. Эти методы позволяют частично сгладить остроту проблем, обусловленных кривизной поверхности функции потерь.

Рассмотрим вектор параметров $\bar{W} = (w_1 \dots w_d)^T$, представленный в виде вектор-столбца³. Производные второго порядка функции потерь $L(\bar{W})$ вычисляются по следующей формуле:

$$H_{ij} = \frac{\partial^2 L(\bar{W})}{\partial w_i \partial w_j}.$$

Обратите внимание на то, что в знаменателе этого выражения попарно используются все параметры. Поэтому для нейронной сети с d параметрами мы получим *матрицу Гессе* H размера $d \times d$, элемент (i, j) которой обозначим как H_{ij} . Вторые производные функции потерь можно вычислить с помощью алгоритма обратного распространения ошибки [315], хотя на практике данный способ используют редко. Этот гессиан можно рассматривать как якобиан градиента.

Запишем квадратичное приближение для функции потерь в окрестности вектора параметров \bar{W}_0 , используя разложение в ряд Тейлора:

$$L(\bar{W}) \approx L(\bar{W}_0) + (\bar{W} - \bar{W}_0)^T [\nabla L(\bar{W}_0)] + \frac{1}{2} (\bar{W} - \bar{W}_0)^T H (\bar{W} - \bar{W}_0). \quad (3.47)$$

Обратите внимание на то, что гессиан H вычисляется как \bar{W}_0 . Здесь векторы параметров \bar{W} и \bar{W}_0 являются d -мерными вектор-столбцами, как и градиент функции потерь. Это квадратичное приближение, и мы можем просто положить градиент равным нулю, что даст нам следующее условие оптимума:

$$\begin{aligned} \nabla L(\bar{W}) &= 0 && \text{[градиент функции потерь]}, \\ \nabla L(\bar{W}_0) + H(\bar{W} - \bar{W}_0) &= 0 && \text{[градиент в приближении Тейлора]}. \end{aligned}$$

³ В этой книге мы в большинстве случаев работали с матрицей \bar{W} как с вектор-строкой. Однако в данном случае работать с ней как с вектор-столбцом явно удобнее.

Переформулировав это условие, мы получаем следующее обновление по методу Ньютона:

$$\bar{W}^* \leftarrow \bar{W}_0 - H^{-1}[\nabla L(\bar{W}_0)]. \quad (3.48)$$

Это обновление интересно тем, что оно получено непосредственно из условия оптимума, и поэтому в нем отсутствует скорость обучения. Иными словами, оно аппроксимирует функцию потерь квадратичной формой и обеспечивает перемещение непосредственно в минимум за один шаг (скорость обучения уже включена в него неявным образом). Вспомните пример, приведенный на рис. 3.9, в котором методы первого порядка осуществляют перемещение в направлениях высокой кривизны. Разумеется, минимум этой квадратичной формы не является минимумом истинной функции потерь, для достижения которого требуется многократное выполнение обновлений по методу Ньютона.

Основным отличием уравнения 3.48 от обновления по методу крутейшего градиентного спуска является предварительное умножение направления скорейшего спуска (которое дается выражением $[\nabla L(\bar{W}_0)]$) на обратную матрицу Гессе. Такое умножение играет определяющую роль в изменении направления крутейшего градиентного спуска, поэтому в данном направлении можно совершать более крупные шаги (приводящие к улучшению целевой функции), даже если *мгновенная* скорость изменения в этом направлении не столь велика, как при перемещении в направлении крутейшего градиентного спуска. Это возможно по той причине, что гессиан содержит информацию о том, насколько быстро изменяется градиент в каждом направлении. Изменение градиентов нежелательно для более крупных обновлений, поскольку можно непреднамеренно ухудшить целевую функцию, если многие компоненты градиента меняют свой знак при выполнении данного шага. Выгодно перемещаться в тех направлениях, в которых отношение градиента к изменению градиента велико, что позволяет выбрать большой размер шага без ущерба для оптимизации. Предварительное умножение на обратный гессиан обеспечивает достижение данной цели. Эффект предварительного умножения направления крутейшего спуска на обратный гессиан показан на рис. 3.14. Полезно сопоставить этот рисунок с примером, приведенным на рис. 3.9. В некотором смысле предварительное умножение на обратный гессиан смещает шаги обучения в направлениях меньшей кривизны. Для одного измерения шаг по методу Ньютона — это просто отношение первой производной (скорость изменения) ко второй (кривизна). В случае нескольких измерений направления низкой кривизны обычно преобладают вследствие умножения на обратный гессиан.

Эффект кривизны становится особенно очевидным в случае поверхностей функций потерь в виде отвесных или извилистых долин. Пример такой долины приведен на рис. 3.15. Для градиентного спуска долина — опасное место,

особенно если ее дно характеризуется крутой и резко изменяющейся поверхностью (что создает узкую долину). Разумеется, относительно простой пример, изображенный на рис. 3.15, не относится к описанной категории. Однако даже в этом случае направление крутейшего спуска часто будет приводить к прыжкам с одной стороны долины на другую и довольно медленному продвижению вниз по склону при неверном выборе шагов. В узких долинах метод градиентного спуска будет еще более интенсивно перемещаться вдоль крутых сторон долины, не продвигаясь в заметной степени в пологих направлениях, которые могут обещать выигрыш в *долгосрочной* перспективе. В подобных случаях направление, корректное в долгосрочной перспективе, может обеспечить лишь нормализация градиента с использованием информации о кривизне. Как правило, этот тип нормализации отдает предпочтение направлениям с низкой кривизной, подобным тем, которые представлены на рис. 3.15. Умножение направления крутейшего спуска на обратный гессиан в точности решает данную задачу.



Рис. 3.14. Эффект предварительного умножения направления крутейшего спуска на обратный гессиан

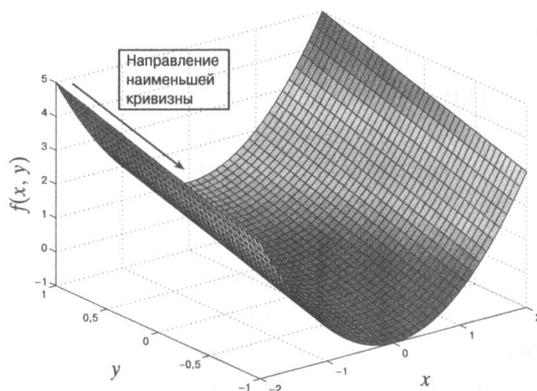


Рис. 3.15. Эффект кривизны в долинах

В случае большинства крупномасштабных нейронных сетей гессиан оказывается слишком большим, чтобы его можно было сохранить или вычислить явным образом. Нередко встречаются нейронные сети, насчитывающие миллионы параметров. Возможности современной вычислительной техники не позволяют вычислить матрицу, обратную к матрице Гессе размера 106×106 , за разумное время. В действительности трудно вычислить даже сам гессиан, не говоря уже о его обращении! В связи с этим были разработаны многие приближенные и видоизмененные варианты метода Ньютона. Примерами таких методов могут служить *оптимизация, не использующая гессиан* [41, 189, 313, 314] (или метод *сопряженных градиентов*), и квазиньютоновские методы, которые

аппроксимируют гессиан. Основной целью этих методов является вычисление обновлений второго порядка без точного вычисления гессиана.

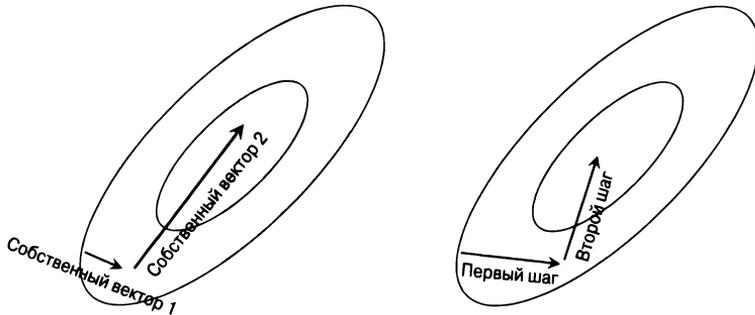
3.5.6.1. Сопряженные градиенты и оптимизация без использования гессиана

Метод сопряженных градиентов (conjugate gradient method) [189] требует выполнения d шагов для достижения оптимального решения в случае квадратичной функции потерь (вместо одного шага в методе Ньютона). Этот подход хорошо известен в классической литературе по нейронным сетям [41, 443] и недавно был переформулирован в виде модифицированного варианта под названием “оптимизация без гессиана” (Hessian-free optimization). Такое название обусловлено тем, что направление поиска может быть определено без явного вычисления гессиана.

Главной проблемой методов первого порядка является зигзагообразный характер процесса оптимизации, нивелирующий результаты предыдущих итераций. В методе сопряженных градиентов направления перемещения соотносятся между собой таким образом, чтобы не ухудшить результаты, достигнутые на предыдущей итерации (для квадратичной функции потерь). Это достигается за счет того, что изменение градиента на каком-либо шаге, если его спроектировать на вектор в направлении любого другого перемещения, всегда равно нулю. Кроме того, при подборе оптимального размера шага используется линейный поиск. *Так как оптимальный размер шага выбирается для каждого направления по отдельности и результаты, достигнутые в этом направлении, никогда не нивелируются последующими шагами, для достижения оптимума d -мерной функции требуется d линейно независимых шагов.* Поскольку нахождение таких направлений возможно только в случае квадратичных функций потерь, в первую очередь мы обсудим метод сопряженных градиентов в предположении, что функция потерь $L(\bar{W})$ квадратична.

Квадратичная выпуклая функция потерь $L(\bar{W})$ имеет эллипсоидный контурный график наподобие того, который приведен на рис. 3.16. Ортонормированные собственные векторы $q_0 \dots q_{d-1}$ симметричного гессиана представляют направления осей эллипсоидного контурного графика. Мы можем записать функцию потерь в новом пространстве координат, соответствующих собственным векторам. В системе осей, соответствующих собственным векторам, (трансформированные) переменные не взаимодействуют между собой в силу ориентации эллипсоидного контура функции потерь относительно системы осей. *Так происходит потому, что новый гессиан $H_q = QTHQ$, полученный в результате выражения функции потерь через преобразованные переменные, — диагональная матрица, где Q — матрица размера $d \times d$, столбцы которой содержат собственные векторы.* Поэтому оптимизация может выполняться независимо по

каждой переменной. Альтернативный вариант заключается в том, чтобы работать с исходными переменными, последовательно совершая наилучшие (спроектированные) шаги вдоль каждого собственного вектора, ведущие к минимизации функции потерь. Наилучшее перемещение вдоль определенного направления выполняется с использованием линейного поиска для выбора размера шага. Природа таких перемещений продемонстрирована на рис. 3.16, а. Обратите внимание на то, что перемещение вдоль j -го собственного вектора не ухудшает результаты, полученные до этого перемещением вдоль других собственных векторов, и потому d шагов достаточно для каждого оптимального решения.



а) Собственные векторы гессиана взаимно ортогональны:

$$\bar{q}_i^T \bar{q}_j = 0$$

б) Произвольная сопряженная пара векторов взаимно неортогональна:

$$\bar{q}_i^T H \bar{q}_j = 0$$

Рис. 3.16. Собственные векторы гессиана квадратичной функции представляют ортогональные оси квадратичного эллипсоида и также взаимно ортогональны. Собственные векторы гессиана представляют ортогональные сопряженные направления. Обобщенное определение сопряженности может приводить к неортогональным направлениям

Несмотря на то что вычисление собственных векторов гессиана трудно осуществимо на практике, имеются и другие эффективно вычисляемые направления, удовлетворяющие аналогичным свойствам. Ключевое свойство носит название *взаимной сопряженности* векторов. Обратите внимание на то, что два разных собственных вектора \bar{q}_i и \bar{q}_j гессиана удовлетворяют условию $\bar{q}_i^T \bar{q}_j = 0$ в силу ортогональности. Кроме того, поскольку \bar{q}_j — собственный вектор H , имеем $H \bar{q}_j = \lambda_j \bar{q}_j$ для некоторого скалярного собственного значения λ_j . Умножая обе части на \bar{q}_i^T , можно легко показать, что собственные векторы гессиана попарно удовлетворяют условию $\bar{q}_i^T H \bar{q}_j = 0$. Его называют *условием взаимной сопряженности* (mutual conjugacy condition), и оно эквивалентно утверждению о том, что гессиан $H_q = Q^T H Q$ в преобразованной системе осей в направлениях

$q_0 \dots q_{d-1}$ представляет собой диагональную матрицу. Фактически, если мы выберем любой набор (не обязательно ортогональных) векторов $q_0 \dots q_{d-1}$, удовлетворяющих условию взаимной сопряженности, то перемещение вдоль любого из этих направлений не возмущает проекции градиента на другие направления. Сопряженные направления, отличные от собственных векторов гессиана наподобие тех, которые приведены на рис. 3.16, б, могут не быть взаимно ортогональными. Если мы выразим квадратичную функцию потерь через координаты неортогональной системы осей сопряженных направлений, то получим аккуратно разделенные переменные с диагональным гессианом $H_q = Q^T H Q$. Однако H_q не является результатом истинной диагонализации матрицы H , поскольку $Q^T Q \neq I$. Тем не менее такие невзаимодействующие направления играют важную роль в устранении зигзагообразного характера продвижения процесса оптимизации.

Пусть \bar{W}_i и \bar{W}_{i+1} представляют соответствующие векторы параметров до и после перемещения в направлении \bar{q}_i . Изменение градиента $\nabla L(\bar{W}_{i+1}) - \nabla L(\bar{W}_i)$, вызванное перемещением в направлении \bar{q}_i , задает то же направление, что и $H\bar{q}_i$. Это объясняется тем, что результат умножения матрицы производных второго порядка (гессиана) на некоторый вектор пропорционален изменению производных первого порядка (градиента) при перемещении в направлении данного вектора. Это соотношение представляет собой конечно-разностную аппроксимацию для неквадратичных функций и строго выполняется для квадратичных функций. Поэтому проекция (или скалярное произведение) вектора этого изменения на любой другой вектор шага $(\bar{W}_{i+1} - \bar{W}_i) \propto \bar{q}_i$ описывается следующим выражением:

$$\underbrace{[\bar{W}_{i+1} - \bar{W}_i]^T}_{\text{Предыдущий шаг}} \underbrace{[\nabla L(\bar{W}_{i+1}) - \nabla L(\bar{W}_i)]}_{\text{Изменение текущего градиента}} \propto \bar{q}_i^T H \bar{q}_i = 0.$$

Это означает, что изменение градиента вдоль некоторого направления \bar{q}_i (в процессе всего обучения) происходит только при выполнении шага в данном направлении. Линейный поиск гарантирует, что окончательный градиент в этом направлении равен нулю. Выпуклые функции потерь имеют линейно независимые сопряженные направления (упражнение 7). Благодаря выполнению наилучших шагов в каждом из сопряженных направлений окончательный градиент будет иметь нулевые проекции на каждое из d линейно независимых направлений. Это возможно только в том случае, если окончательный градиент представляет собой нулевой вектор (упражнение 8), что является условием оптимума выпуклой функции. На практике число обновлений, необходимых для достижения решения, близкого к оптимальному, часто оказывается намного меньшим, чем d .

Как итеративно генерировать сопряженные направления? Очевидный подход требует отслеживания $O(d^2)$ векторных компонент всех предыдущих $O(d)$ сопряженных направлений, чтобы навязать следующему направлению сопряженность по отношению ко всем предыдущим (упражнение 11). Как это ни удивительно, если для итеративного генерирования направлений используются направления крутейшего спуска, то следующее направление можно генерировать с использованием лишь самого последнего сопряженного направления [359, 443]. Этот результат не является очевидным (упражнение 12). Поэтому направление \bar{q}_{t+1} определяется итеративно в виде линейной комбинации текущего направления крутейшего спуска $\nabla L(\bar{W}_{t+1})$ и *одного лишь* предыдущего сопряженного направления \bar{q}_t , умноженного на параметр β_t :

$$\bar{q}_{t+1} = -\nabla L(\bar{W}_{t+1}) + \beta_t \bar{q}_t. \quad (3.49)$$

Умножив обе части уравнения на $\bar{q}_t^T H$ и приняв во внимание тот факт, что в силу условия сопряженности левая часть становится равной нулю, получаем следующее выражение для β_t :

$$\beta_t = \frac{\bar{q}_t^T H [\nabla L(\bar{W}_{t+1})]}{\bar{q}_t^T H \bar{q}_t}. \quad (3.50)$$

Это приводит нас к следующей процедуре итеративного обновления, которая инициализируется значением $\bar{q}_0 = -\nabla L(\bar{W}_0)$ и позволяет итеративно вычислять \bar{q}_{t+1} для $t = 0, 1, 2, \dots, T$.

1. Обновить $\bar{W}_{t+1} \leftarrow \bar{W}_t + \alpha_t \bar{q}_t$. Здесь размер шага α_t вычисляется с помощью линейного поиска для минимизации функции потерь.
2. Установить $\bar{q}_{t+1} = -\nabla L(\bar{W}_{t+1}) + \left(\frac{\bar{q}_t^T H [\nabla L(\bar{W}_{t+1})]}{\bar{q}_t^T H \bar{q}_t} \right) \bar{q}_t$. Увеличить t на 1.

Можно показать [359, 443], что \bar{q}_{t+1} удовлетворяет условию сопряженности по отношению ко *всем* предыдущим \bar{q}_i . Систематическое представление этапов этого вы найдете в упражнении 12.

На первый взгляд, приведенные формулы для обновлений не соответствуют оптимизации без гессиана, поскольку в них входит матрица H . Однако базовые вычисления нуждаются лишь в *проекциях* гессиана на определенные направления. Как будет показано далее, их можно вычислить косвенно с помощью метода конечных разностей без явного вычисления элементов гессиана. Пусть \bar{v} — вектор направления, для которого требуется вычислить проекцию $H\bar{v}$. Приближенное значение этой проекции вычисляется методом конечных

разностей путем вычисления градиента функции потерь для текущего вектора параметров \bar{W} и вектора $\bar{W} + \delta\bar{v}$ при некотором небольшом значении δ :

$$H\bar{v} \approx \frac{\nabla L(\bar{W} + \delta\bar{v}) - \nabla L(\bar{W})}{\delta} \propto \nabla L(\bar{W} + \delta\bar{v}) - \nabla L(\bar{W}). \quad (3.51)$$

В правой части этого уравнения отсутствует гессиан. Данное уравнение строго выполняется для квадратичных функций. Альтернативные варианты обновлений, не требующих вычисления гессиана, обсуждаются в [41].

До сих пор мы рассматривали упрощенный случай квадратичных функций потерь, в которых матрица вторых производных (гессиан) постоянна (т.е. не зависит от текущего вектора параметров). Однако нейронные функции потерь не являются квадратичными, и в этом случае матрица Гессе зависит от текущего значения \bar{W}_t . Следует ли нам сначала создать квадратичное приближение в данной точке, а затем разрешить его, выполнив несколько итераций с помощью гессиана (квадратичное приближение), зафиксированного в этой точке, или же мы должны изменять гессиан на каждой итерации? Первый вариант называется *линейным методом сопряженных градиентов* (linear conjugate gradient method), второй — *нелинейным методом сопряженных градиентов*. Оба этих метода становятся эквивалентными в случае квадратичных функций потерь, которые почти никогда не встречаются в мире нейронных сетей.

В классической работе по нейронным сетям и глубокому обучению [41] преимущественно исследовано использование нелинейного метода сопряженных градиентов, тогда как в недавних работах [313, 314] пропагандируется использование линейных методов. В нелинейном методе взаимная сопряженность направлений со временем нарушается, что может непредсказуемым образом повлиять на общий прогресс даже после выполнения большого количества итераций. Частью этой проблемы является то, что процесс вычисления сопряженных градиентов приходится перезапускать каждые несколько шагов по мере ухудшения сопряженности. Если это ухудшение происходит слишком быстро, то сопряженность не даст существенного выигрыша. С другой стороны, каждая квадратичная аппроксимация в линейном методе сопряженных градиентов может быть точно разрешена и обычно (почти) разрешается за намного меньшее количество итераций, чем d . Несмотря на то что потребуется несколько таких аппроксимаций, прогресс в пределах каждой аппроксимации гарантирован, а количество требуемых аппроксимаций часто оказывается не очень большим. Превосходство линейных методов сопряженных градиентов было экспериментально продемонстрировано в [313].

3.5.6.2. Квазиньютоновские методы и алгоритм BFGS

Алгоритм Бройдена — Флетчера — Гольдфарба — Шанно (сокр. BFGS) — это приближение метода Ньютона. Вернемся к обновлениям по методу Ньютона. Типичное обновление в методе Ньютона записывается в следующем виде:

$$\bar{W}^* \leftarrow \bar{W}_0 - H^{-1}[\nabla L(\bar{W}_0)]. \quad (3.52)$$

В квазиньютоновских методах на различных шагах используется последовательность приближений к обратной матрице Гессе. Обозначим через G_t приближение к обратному гессиану на шаге t . Перед началом итераций G_t инициализируется тождественной матрицей, что означает перемещение в направлении крутейшего спуска. Эта матрица непрерывно обновляется от G_t до G_{t+1} с помощью низкоранговых обновлений. Непосредственное переформулирование правила обновления Ньютона в терминах обратного гессиана $G_t \approx H_t^{-1}$ приводит к следующему соотношению:

$$\bar{W}_{t+1} \leftarrow \bar{W}_t - G_t[\nabla L(\bar{W}_t)]. \quad (3.53)$$

Это обновление можно улучшить за счет оптимизации скорости обучения α_t для неквадратичных функций потерь, работая с приближением к (обратному) гессиану наподобие G_t :

$$\bar{W}_{t+1} \leftarrow \bar{W}_t - \alpha_t G_t[\nabla L(\bar{W}_t)]. \quad (3.54)$$

Оптимизированное значение скорости обучения α_t определяется с помощью линейного поиска. Линейный поиск не обязательно должен выполняться точно (как метод сопряженных градиентов), поскольку поддержание сопряженности направлений для него не является критичным. Тем не менее приближительная сопряженность начального набора направлений сохраняется, если процесс начинается с тождественной матрицы. Также возможна переустановка матрицы G_t в тождественную матрицу через каждые d итераций (хотя этим пользуются редко).

Нам осталось лишь обсудить, как получить приближение к матрице G_{t+1} на основании матрицы G_t . Для этих целей используется так называемое *квазиньютоновское условие*:

$$\underbrace{\bar{W}_{t+1} - \bar{W}_t}_{\text{Изменение параметра}} = G_{t+1} \underbrace{[\nabla L(\bar{W}_{t+1}) - \nabla L(\bar{W}_t)]}_{\text{Изменение первой производной}} \quad (3.55)$$

Вышеприведенная формула — это конечно-разностное приближение. Интуитивно понятно, что умножение матрицы вторых производных (т.е. гессиана) на изменение параметров (вектор) приближенно дает изменение градиента. Поэтому умножение приближения обращенного гессиана G_{t+1} на изменение градиента дает изменение параметров. Нашей целью является нахождение симметричной

матрицы G_{t+1} , удовлетворяющей уравнению 3.55, однако оно представляет недоопределенную систему уравнений с бесконечным числом решений. Из них BFGS выбирает ближайшую к текущей матрице G_t симметричную матрицу G_{t+1} и достигает этой цели за счет представления минимизируемой целевой функции $\|G_{t+1} - G_t\|_F$ в виде взвешенной нормы Фробениуса:

$$G_{t+1} \leftarrow (I - \Delta_t \bar{q}_t \bar{v}_t^T) G_t (I - \Delta_t \bar{v}_t \bar{q}_t^T) + \Delta_t \bar{q}_t \bar{q}_t^T. \quad (3.56)$$

Здесь векторы (столбцы) \bar{q}_t и \bar{v}_t представляют изменение параметров и изменение градиента. Скаляр $\Delta_t = 1 / (\bar{q}_t^T \bar{v}_t)$ — обратная величина скалярного произведения этих двух векторов.

$$\bar{q}_t = \bar{W}_{t+1} - \bar{W}_t; \quad \bar{v}_t = \nabla L(\bar{W}_{t+1}) - \nabla L(\bar{W}_t).$$

Обновление в уравнении 3.56 можно преобразовать к виду, более экономному в отношении расходования памяти, разложив его в ряд, чтобы пришлось хранить меньшее количество временных матриц. Относительно деталей реализации и вывода формул для этих обновлений отсылаем заинтересованных читателей к [300, 359, 376].

Несмотря на то что в алгоритме BFGS используется приближение к обратному гессиану, матрица G_t размера $O(d^2)$ не переносится из одной итерации в следующую. Алгоритм *BFGS с ограниченным использованием памяти* (limited memory BFGS — L-BFGS) резко снижает требования к памяти от $O(d^2)$ до $O(d)$, не перенося матрицу G_t из предыдущей итерации. В базовой версии метода L-BFGS матрица G_t заменяется тождественной матрицей в уравнении 3.56 для получения G_{t+1} . Улучшенный вариант предполагает хранение $m \approx 30$ последних векторов \bar{q}_t и \bar{v}_t . Тогда метод L-BFGS эквивалентен инициализации G_{t-m+1} тождественной матрицей с последующим рекурсивным применением уравнения 3.56 m раз для получения G_{t+1} . На практике реализация оптимизируется до прямого вычисления направления перемещения на основании векторов без явного хранения больших промежуточных матриц от G_{t-m+1} до G_t . Направления, найденные с помощью метода L-BFGS, в грубом приближении удовлетворяют условию взаимной сопряженности даже в случае использования приближенно-линейного поиска.

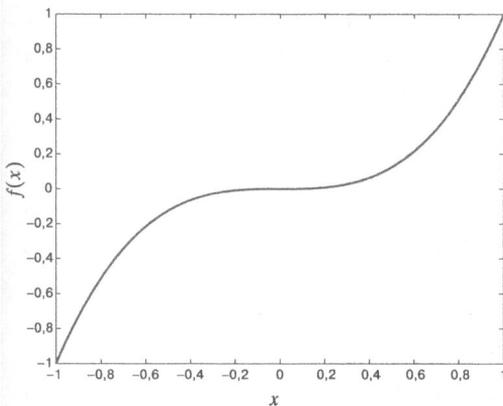
3.5.6.3. Проблемы матриц второго порядка: седловые точки

Методы второго порядка чувствительны к наличию седловых точек. *Седловая точка* — это стационарная точка метода градиентного спуска, поскольку градиент в ней равен нулю, но ей не соответствует минимум (или максимум). Это *точка перегиба*, которая может проявлять себя либо как точка минимума, либо как точка максимума, в зависимости от того, с какой стороны к ней приближаться. Поэтому квадратичная аппроксимация метода Ньютона будет по-

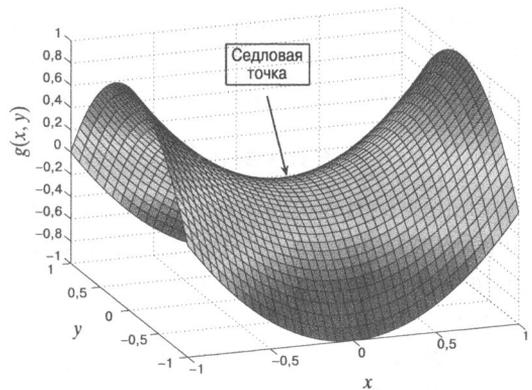
рождать большое разнообразие форм, в зависимости от направления подхода к седловой точке. Вот пример одномерной функции, имеющей седловую точку:

$$f(x) = x^3.$$

Эта функция, которая представлена на рис. 3.17, а, имеет точку перегиба при $x = 0$. Обратите внимание на то, что при $x > 0$ квадратичная аппроксимация обращена выпуклостью вниз, тогда как при $x < 0$ — выпуклостью вверх. Если же в процессе оптимизации будет достигнута точка $x = 0$, то в ней будут равны нулю как первая, так и вторая производная. Поэтому ньютоновское правило примет вид неопределенности $0/0$. С точки зрения численной оптимизации такая точка является вырожденной. Однако не каждая седловая точка является вырожденной, и не каждая вырожденная точка является седловой. В многомерных задачах такие вырожденные точки часто проявляют себя в виде протяженных плоских областей, не являющихся минимумом целевой функции. Это создает значительные проблемы для оптимизации. В качестве примера можно привести функцию $h(x, y) = x^3 + y^3$, которая является вырожденной в точке $(0, 0)$. Кроме того, прилегающая к этой точке область выглядит как плоское плато. Такого типа плато создают проблемы для алгоритмов обучения, поскольку алгоритмы первого порядка работают медленнее в этих областях, а алгоритмы второго порядка не распознают их как области ложных оптимумов. Следует отметить, что подобные седловые точки возникают лишь в случае алгебраических функций высокого (т.е. выше второго) порядка, что типично для задач оптимизации нейронных сетей.



а) 1-мерная седловая точка



б) 2-мерная седловая точка

Рис. 3.17. Седловые точки

Будет поучительно исследовать также случай седловой точки, не являющейся вырожденной. В качестве примера приведем 2-мерную функцию $g(x, y) = x^2 - y^2$ (рис. 3.17, б). Седловая точка имеет координаты $(0, 0)$. Нетрудно

заметить, что своей формой эта функция действительно напоминает седло. В данном случае при подходе к седловой точке вдоль оси x и вдоль оси y мы получаем совершенно разные квадратичные аппроксимации. В одном случае седловая точка будет казаться точкой минимума, в другом — точкой максимума. Кроме того, седловая точка $(0, 0)$ — стационарная с позиций правила обновления Ньютона, даже если она и не является точкой экстремума. Седловые точки часто встречаются в областях между холмами функции потерь и являются проблемной топографией для методов второго порядка. Интересно отметить, что методам первого порядка часто удается избегать попадания в седловые точки [146], поскольку траектории этих методов не притягиваются подобными точками. С другой стороны, метод Ньютона просто совершает “прыжок” непосредственно в такую точку.

К сожалению, наличие большого количества седловых точек характерно для функций потерь некоторых нейронных сетей. Поэтому методы второго порядка не всегда более предпочтительны по сравнению с методами первого порядка. Важную роль могут играть особенности топографии конкретных функций потерь. Преимущества методов второго порядка проявляются в случае функций потерь с участками сложной кривизны или оврагами. В случае других функций с седловыми точками преимущество переходит на сторону методов первого порядка. Обратите внимание на то, что уже одно сочетание вычислительных алгоритмов (таких, как Adam) с методами градиентного спуска первого порядка неявно включает в себя некоторые преимущества методов второго порядка. Поэтому в реальной практике часто отдают предпочтение использованию комбинаций методов первого порядка с такими вычислительными алгоритмами, как Adam. Недавно для решения проблемы седловых точек в случае использования методов второго порядка был предложен ряд способов [88].

3.5.7. Усреднение Поляка

Одной из привлекательных сторон методов второго порядка является то, что они позволяют избежать скачкообразного поведения процесса оптимизации в областях высокой кривизны. Такое же поведение может наблюдаться и при наличии оврагов на оптимизируемой поверхности (см. рис. 3.15). Одним из способов обеспечения определенной стабильности при использовании любого алгоритма обучения является усреднение параметров, экспоненциально затухающее во времени, целью которого является устранение скачков. Пусть $\vec{W}_1 \dots \vec{W}_t$ — последовательность параметров, найденная с помощью любого алгоритма обучения за T шагов. Простейший вариант усреднения Поляка сводится к вычислению обычного среднего \vec{W}_T^f по всему набору:

$$\bar{W}_T^f = \frac{\sum_{i=1}^T \bar{W}_i}{T}. \quad (3.57)$$

В случае этого простого усреднения мы должны вычислить \bar{W}_T^f только один раз в конце процесса, и от нас не требуется вычислять аналогичные значения для шагов $1 \dots T-1$.

Однако в случае экспоненциально затухающего среднего с параметром затухания $\beta < 1$ полезно вычислять эти значения итеративно и поддерживать скользящее среднее в процессе работы алгоритма:

$$\bar{W}_t^f = \frac{\sum_{i=1}^t \beta^{t-i} \bar{W}_i}{\sum_{i=1}^t \beta^{t-i}} \quad [\text{явная формула}],$$

$$\bar{W}_t^f = (1 - \beta) \bar{W}_t + \beta \bar{W}_{t-1}^f \quad [\text{рекуррентная формула}].$$

Обе эти формы приблизительно эквивалентны при больших значениях t . Вторая форма более удобна, поскольку она не требует хранения всей предыстории значений параметров. Экспоненциально затухающее усреднение снижает влияние давно пройденных точек. При простом усреднении на окончательный результат могут сильно влиять более ранние точки, являющиеся плохим приближением к корректному решению.

3.5.8. Локальные и ложные минимумы

Приведенные в предыдущих разделах примеры квадратичных поверхностей относятся к сравнительно простым задачам оптимизации, имеющим один глобальный минимум. Задачи такого типа, которые называют *выпуклыми задачами оптимизации*, представляют собой простейший случай оптимизации. Однако в общем случае целевые функции нейронных сетей не являются выпуклыми и склонны иметь многочисленные минимумы. В подобных ситуациях обучение может сходиться к неоптимальному решению. Несмотря на это, при достаточно хорошей инициализации параметров проблема локальных минимумов доставляет меньше проблем, чем можно было бы ожидать.

Локальные минимумы являются источником проблем лишь в том случае, когда значения целевой функции значительно превышают значение в локальном минимуме. Однако на практике с этим обычно не приходится сталкиваться в нейронных сетях. Результаты многих исследований [88, 426] говорят о том, что в реальных сетях значения целевых функций в локальных минимумах очень близки к значению в глобальном минимуме. В результате этого их наличие не создает столь острых проблем, как обычно считается.

Локальные минимумы часто порождают проблемы в контексте *обобщения моделей* в условиях ограниченности данных. Важно иметь в виду, что функция потерь всегда определяется на ограниченной выборке тренировочных данных и является лишь грубым приближением функции потерь в той форме, которую она принимает на истинном распределении неизвестных контрольных данных. Малочисленность тренировочных данных приводит к созданию ряда ложных глобальных или локальных минимумов. Эти минимумы не наблюдаются на (бесконечно большом) распределении неизвестных данных контрольных примеров, но проявляются в виде случайных артефактов конкретно выбранного набора тренировочных данных. Такие ложные минимумы часто проявляются в более заметной степени и служат более сильными центрами притяжения в ситуациях, когда функция потерь строится на небольших тренировочных выборках. В подобных случаях ложные минимумы действительно создают проблемы, поскольку они не обобщаются достаточно хорошо на неизвестные тестовые примеры. Эта проблема несколько отличается от обычной проблемы локальных минимумов в том смысле, в каком она понимается в традиционной оптимизации. *Локальные минимумы, соответствующие тренировочным данным, не обобщаются в полной мере на тестовые данные.* Иными словами, форма функции потерь не является одной и той же для тренировочных и тестовых данных, поэтому минимумы в этих двух случаях не совпадают. Важно понимать, что существуют фундаментальные различия между традиционной оптимизацией и методами машинного обучения, которые пытаются обобщать функцию потерь, полученную на ограниченном наборе данных, на бесконечное множество тестовых примеров. В данном случае мы имеем дело с понятием так называемой *минимизации эмпирического риска*, когда для алгоритма вычисляется (приближенный) *эмпирический риск*, поскольку истинное распределение примеров неизвестно. Если начинать со случайных точек инициализации, то это нередко приводит к “скатыванию” в один из этих ложных минимумов, если только заранее не переместить точку инициализации ближе к бассейнам аттракции истинного оптимума (с точки зрения обобщаемости модели). Одним из таких подходов является *предварительное обучение без учителя* (unsupervised pretraining), которое рассматривается в главе 4.

Для обучения нейронных сетей конкретная проблема ложных минимумов (обусловленная неспособностью к обобщению результатов с ограниченного набора тренировочных данных на неизвестные тестовые данные) является гораздо более серьезной, чем проблема локальных минимумов (с точки зрения традиционной оптимизации). Природа этой проблемы достаточно существенно отличается от общепринятого понимания локальных минимумов, поэтому она обсуждается в главе 4, посвященной обобщаемости моделей.

3.6. Пакетная нормализация

Пакетная нормализация (batch normalization) — недавно предложенный метод, призванный справиться с проблемами затухающих и взрывных градиентов, проявляющимися в виде уменьшения или увеличения активаций градиентов по мере прохождения последовательности слоев. Другой серьезной проблемой тренировки глубоких сетей является *внутренний ковариационный сдвиг* (internal covariate shift). Суть этой проблемы заключается в том, что по мере обучения параметры изменяются, а вместе с ними изменяются и активации скрытых переменных. Иными словами, скрытые входы по мере продвижения от ранних слоев к более поздним постоянно изменяются. Это изменение входов ухудшает сходимость в процессе обучения ввиду нестабильности обучающих данных для поздних слоев. Пакетная нормализация помогает ослабить данный эффект.

Идея пакетной нормализации заключается в добавлении дополнительных слоев нормализации между скрытыми слоями, которые подавляют такой тип поведения за счет создания признаков с примерно одинаковой дисперсией. Кроме того, каждый элемент слоя нормализации содержит два дополнительных параметра, β_i и γ_i , регулирующие точный уровень нормализации в i -м элементе; обучение этих параметров управляется данными. Суть базовой идеи в том, что выход i -го элемента будет иметь среднее β_i и стандартное отклонение γ_i для каждого мини-пакета тренировочных примеров. Казалось бы, в качестве одного из вариантов можно было бы просто установить каждый параметр β_i в 0, а каждый параметр γ_i — в 1, но это приведет к снижению представительной мощности сети. Например, если выполнить подобное преобразование, то сигмоидные элементы будут работать на линейных участках, особенно если выполнять нормализацию непосредственно перед активацией (рис. 3.18). Как отмечалось в главе 1, глубина многослойных сетей, если не использовать нелинейную активацию, не дает никакого выигрыша. Поэтому в предоставлении указанным параметрам возможности варьироваться в определенных пределах и их обучении на основе данных есть определенный смысл. Кроме того, параметр β_i играет роль обучаемой переменной смещения, в связи с чем нет никакой необходимости вводить дополнительные элементы смещения в этих слоях.

Мы предполагаем, что i -й элемент соединен со специальным типом узла BN_i , где BN — аббревиатура от “Batch Normalization”. Этот элемент содержит два параметра, β_i и γ_i , нуждающихся в обучении. Заметьте, что BN_i имеет только один вход, и его задача состоит в том, чтобы выполнять нормализацию и масштабирование. Данный узел соединяется со следующим слоем сети стандартным способом, используемым в нейронной сети для связывания с последующими слоями. В данном случае возможны следующие два способа связи.

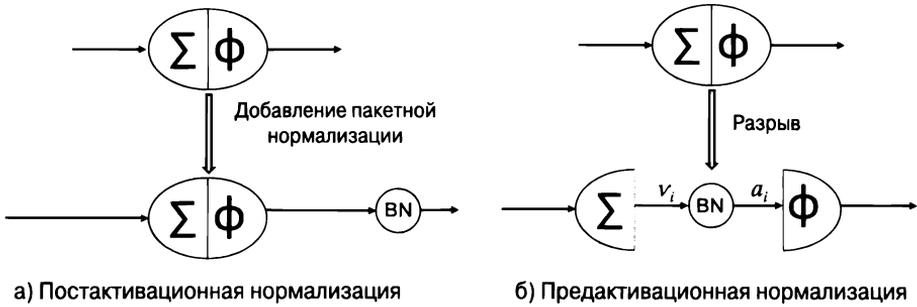


Рис. 3.18. Разные варианты пакетной нормализации

1. Нормализация может выполняться непосредственно после применения функции активации к линейно преобразованным входам. Это решение представлено на рис. 3.18, а. В результате нормализуются постактивационные значения.
2. Нормализация может выполняться после линейного преобразования входов. Эта ситуация представлена на рис. 3.18, б. В результате нормализуются преактивационные значения.

Как утверждается в [214], более предпочтительным является второй вариант. Его мы и выберем. Узел BN (см. рис. 3.18, б) ведет себя как любой другой вычислительный узел (хотя и обладает некоторыми особыми свойствами), поэтому к нему применим алгоритм обратного распространения ошибки.

Какие преобразования выполняет узел BN_i ? Пусть его вход $v_i^{(r)}$ соответствует r -му элементу пакета, поступающему в i -й элемент. Каждый вход $v_i^{(r)}$ является результатом линейного преобразования, определяемого вектором коэффициентов \bar{W} (и смещениями, если таковые имеются). Обозначим через $v_i^{(1)}, v_i^{(2)}, \dots, v_i^{(m)}$ значения m активаций для конкретного пакета, включающего m примеров. Первый шаг заключается в вычислении среднего μ_i и стандартного отклонения σ_i для i -го скрытого элемента. Затем эти величины масштабируются с использованием параметров β_i и γ_i с целью создания выходов для следующего слоя:

$$\mu_i = \frac{\sum_{r=1}^m v_i^{(r)}}{m} \quad \forall i, \quad (3.58)$$

$$\sigma_i^2 = \frac{\sum_{r=1}^m (v_i^{(r)} - \mu_i)^2}{m} + \varepsilon \quad \forall i, \quad (3.59)$$

$$\hat{v}_i^{(r)} = \frac{v_i^{(r)} - \mu_i}{\sigma_i} \quad \forall i, r, \quad (3.60)$$

$$a_i^{(r)} = \gamma_i \cdot \hat{v}_i^{(r)} + \beta_i \quad \forall i, r. \quad (3.61)$$

Небольшое значение ϵ добавлено к σ_i^2 для регуляризации тех случаев, когда все активации одинаковы, что приведет к нулевой дисперсии. Обратите внимание на то, что $a_i^{(r)}$ — это преактивационный выход i -го узла, когда через него пропускается r -й пример пакета. Если бы мы не применяли пакетную регуляризацию, то это значение было бы установлено равным $v_i^{(r)}$. Мы концептуально представляем данный узел как специальный узел BN_p , который выполняет дополнительную обработку (см. рис. 3.18, б). Поэтому алгоритм обратного распространения ошибки должен учесть дополнительный узел и гарантировать, что производные функции потерь слоев, предшествующих слою пакетной нормализации, учтут данное преобразование, предположительно осуществляемое этими новыми узлами. Важно отметить, что функция, применяемая в каждом из специальных узлов BN , специфична для используемого пакета. Такой тип вычислений необычен для нейронной сети, в которой градиенты представляют собой линейно разделимые суммы градиентов, соответствующих индивидуальным тренировочным примерам. В данном случае это не совсем так, поскольку слой пакетной нормализации вычисляет на основе пакета нелинейные метрики (такие, как стандартное отклонение). Поэтому активации зависят от того, как связаны между собой примеры в пределах одного пакета, что не свойственно большинству нейронных вычислений. Однако это специальное свойство узла BN не препятствует обратному распространению ошибки через вычисления, выполняемые таким узлом.

Ниже описаны изменения, которые должны быть внесены в алгоритм обратного распространения ошибки в связи с введением слоя нормализации. Наша основная цель — продемонстрировать, как выполняется обратное распространение через вновь добавленный слой узлов нормализации. Дополнительная цель — показать, как оптимизируются параметры β_i и γ_i . Для шагов градиентного спуска в отношении каждого из параметров β_i и γ_i нам нужны градиенты по этим параметрам. Предположим, что обратное распространение ошибки уже выполнено вплоть до выхода узла BN , поэтому мы имеем все необходимые производные $\frac{\partial L}{\partial a_i^{(r)}}$. Тогда производные по этим параметрам можно вычислить по следующим формулам:

$$\frac{\partial L}{\partial \beta_i} = \sum_{r=1}^m \frac{\partial L}{\partial a_i^{(r)}} \cdot \frac{\partial a_i^{(r)}}{\partial \beta_i} = \sum_{r=1}^m \frac{\partial L}{\partial a_i^{(r)}},$$

$$\frac{\partial L}{\partial \gamma_i} = \sum_{r=1}^m \frac{\partial L}{\partial a_i^{(r)}} \cdot \frac{\partial a_i^{(r)}}{\partial \gamma_i} = \sum_{r=1}^m \frac{\partial L}{\partial a_i^{(r)}} \cdot \hat{v}_i^{(r)}.$$

Нам также необходим способ вычисления $\frac{\partial L}{\partial v_i^r}$. Как только будет вычислено это значение, обратное распространение ошибки до преактивационных значений $\frac{\partial L}{\partial a_j^r}$ для всех узлов j предыдущего слоя использует непосредственное обновление обратного распространения, введенное ранее. Поэтому рекурсия динамического программирования будет завершена, поскольку в ней можно будет использовать значения $\frac{\partial L}{\partial a_j^r}$. Учитывая тот факт, что $v_i^{(r)}$ можно записать как функцию (нормализации), зависящую только от $\hat{v}_i^{(r)}$, среднего μ_i и дисперсии σ_i^2 , значение $\frac{\partial L}{\partial v_i^r}$ можно выразить через $\hat{v}_i^{(r)}$, μ_i и σ_i . Заметьте, что μ_i и σ_i трактуются не как константы, а как переменные, поскольку они зависят от обрабатываемого пакета. В итоге получаем следующие соотношения:

$$\frac{\partial L}{\partial v_i^{(r)}} = \frac{\partial L}{\partial \hat{v}_i^{(r)}} \frac{\partial \hat{v}_i^{(r)}}{\partial v_i^{(r)}} + \frac{\partial L}{\partial \mu_i} \frac{\partial \mu_i}{\partial v_i^{(r)}} + \frac{\partial L}{\partial \sigma_i^2} \frac{\partial \sigma_i^2}{\partial v_i^{(r)}} = \quad (3.62)$$

$$= \frac{\partial L}{\partial \hat{v}_i^{(r)}} \left(\frac{1}{\sigma_i} \right) + \frac{\partial L}{\partial \mu_i} \left(\frac{1}{m} \right) + \frac{\partial L}{\partial \sigma_i^2} \left(\frac{2(v_i^{(r)} - \mu_i)}{m} \right). \quad (3.63)$$

Мы должны вычислить каждую из трех частных производных в правой части уравнения, выразив их через величины, вычисляемые с помощью уже полученных посредством динамического программирования обновлений обратного распространения ошибки. Это позволяет создать рекуррентное уравнение для слоя пакетной нормализации. Учитывая тот факт, что $a_i^{(r)}$ связано с $\hat{v}_i^{(r)}$ константой пропорциональности γ_i , мы можем выразить первую из этих производных, $\frac{\partial L}{\partial \hat{v}_i^{(r)}}$, через производную функции потерь следующего слоя:

$$\frac{\partial L}{\partial \hat{v}_i^{(r)}} = \gamma_i \frac{\partial L}{\partial a_i^{(r)}} \left[\text{поскольку } a_i^{(r)} = \gamma_i \cdot \hat{v}_i^{(r)} + \beta_i \right]. \quad (3.64)$$

Таким образом, подставляя это значение $\frac{\partial L}{\partial \hat{v}_i^{(r)}}$ в уравнение 3.63, получаем следующее соотношение:

$$\frac{\partial L}{\partial v_i^{(r)}} = \frac{\partial L}{\partial a_i^{(r)}} \left(\frac{\gamma_i}{\sigma_i} \right) + \frac{\partial L}{\partial \mu_i} \left(\frac{1}{m} \right) + \frac{\partial L}{\partial \sigma_i^2} \left(\frac{2(v_i^{(r)} - \mu_i)}{m} \right). \quad (3.65)$$

Теперь остается вычислить только частную производную функции потерь по среднему и дисперсии. Частная производная функции потерь по дисперсии вычисляется по следующей формуле:

$$\frac{\partial L}{\partial \sigma_i^2} = \underbrace{\sum_{q=1}^m \frac{\partial L}{\partial \hat{v}_i^{(q)}} \cdot \frac{\partial \hat{v}_i^{(q)}}{\partial \sigma_i^2}}_{\text{Цепное правило}} = \underbrace{-\frac{1}{2\sigma_i^3} \sum_{q=1}^m \frac{\partial L}{\partial \hat{v}_i^{(q)}} (v_i^{(q)} - \mu_i)}_{\text{Используем уравнение 3.60}} = \underbrace{-\frac{1}{2\sigma_i^3} \sum_{q=1}^m \frac{\partial L}{\partial a_i^{(q)}} \gamma_i \cdot (v_i^{(q)} - \mu_i)}_{\text{Подставляем из уравнения 3.64}}.$$

Частная производная функции потерь по среднему вычисляется по следующей формуле:

$$\begin{aligned} \frac{\partial L}{\partial \mu_i} &= \underbrace{\sum_{q=1}^m \frac{\partial L}{\partial \hat{v}_i^{(q)}} \cdot \frac{\partial \hat{v}_i^{(q)}}{\partial \mu_i}}_{\text{Цепное правило}} + \frac{\partial L}{\partial \sigma_i^2} \frac{\partial \sigma_i^2}{\partial \mu_i} = \underbrace{-\frac{1}{\sigma_i} \sum_{q=1}^m \frac{\partial L}{\partial \hat{v}_i^{(q)}} - 2 \frac{\partial L}{\partial \sigma_i^2} \cdot \frac{\sum_{q=1}^m (v_i^{(q)} - \mu_i)}{m}}_{\text{Используем уравнения 3.60 и 3.59}} = \\ &= \underbrace{-\frac{\gamma_i}{\sigma_i} \sum_{q=1}^m \frac{\partial L}{\partial a_i^{(q)}}}_{\text{Уравнение 3.64}} + \left(\frac{1}{\sigma_i^3} \right) \cdot \left(\sum_{q=1}^m \frac{\partial L}{\partial a_i^{(q)}} \gamma_i \cdot (v_i^{(q)} - \mu_i) \right) \cdot \left(\frac{\sum_{q=1}^m (v_i^{(q)} - \mu_i)}{m} \right) \\ &\quad \underbrace{\hspace{10em}}_{\text{Подстановка вместо } \frac{\partial L}{\partial \sigma_i^2}} \end{aligned}$$

Вставляя частные производные функции потерь по среднему и дисперсии в уравнение 3.65, получаем полную рекурсию для $\frac{\partial L}{\partial v_i^{(r)}}$ (значение перед слоем пакетной нормализации) в терминах $\frac{\partial L}{\partial a_i^{(r)}}$ (значение после слоя пакетной нормализации). Это дает полную картину обратного распространения потерь через слой пакетной нормализации, соответствующий узлу *BN*. Другие аспекты обратного распространения остаются теми же, что и в традиционном случае. Пакетная нормализация ускоряет логический вывод, поскольку предотвращает возникновение таких проблем, как взрывные и затухающие градиенты (замедляющие обучение).

У читателей может возникнуть вполне естественный вопрос: как работает пакетная нормализация на стадии логического вывода (предсказания)? Поскольку параметры преобразования μ_i и σ_i зависят от пакета, то каким образом следует их вычислять в условиях тестирования, когда имеется *всего один* контрольный пример? В таком случае значения μ_i и σ_i вычисляются заранее на *полной* генеральной совокупности (тренировочных данных), а затем считаются константами в процессе тестирования. Также возможно использование экспоненциально взвешенного скользящего среднего в процессе тренировки. Поэтому в процессе логического вывода нормализация проявляет себя в виде простого линейного преобразования.

Пакетная нормализация обладает интересным свойством: она *также играет роль регуляризатора*. Обратите внимание на то, что одна и та же точка данных может приводить к несколько отличающимся обновлениям в зависимости от пакета, в который она включена. Этот эффект можно рассматривать как добавление своеобразного шума в процесс обновления. Регуляризацию часто осуществляют путем добавления небольшого шума в тренировочные данные. Экспериментальные результаты указывают на то, что, по-видимому, методы регуляризации наподобие *метода исключений* (раздел 4.5.4) не дают никакого выигрыша, если используется пакетная нормализация [184], хотя полного согласия относительно этого пока что не достигнуто. Установлено, что в случае рекуррентных сетей хорошо работает вариант пакетной нормализации, известный как *послойная нормализация*. Этот подход обсуждается в разделе 7.3.1.

3.7. Практические приемы ускорения вычислений и сжатия моделей

Алгоритмы обучения нейронных сетей чрезвычайно требовательны к вычислительным ресурсам в отношении как количества параметров модели, так и объемов данных, которые приходится обрабатывать. Существует несколько стратегий, направленных на ускорение и сжатие базовых реализаций. Наиболее распространенными из них являются следующие.

1. *Ускорение с помощью GPU*. Графические процессоры (Graphics Processor Unit — GPU) исторически применялись для визуализации видеоигр с интенсивной графикой в силу их высокой эффективности в задачах, требующих многократного выполнения матричных операций (например, при обработке пиксельной графики). Вскоре разработчики программного обеспечения для машинного обучения (а также компании-производители GPU) осознали, что подобные многократные операции приходится выполнять и в нейронных сетях, интенсивно использующих матричные

вычисления. Использование даже одного GPU способно значительно ускорить вычисления благодаря высокой пропускной способности памяти графических процессоров и многопоточности их многоядерной архитектуры.

2. *Параллельные вычисления.* Вычисления в нейронных сетях можно распараллеливать за счет использования нескольких GPU или CPU. Как модели нейронных сетей, так и данные могут разделяться различными процессорами. В подобных случаях говорят о *реализациях на основе параллелизма моделей и параллелизма данных* соответственно.
3. *Алгоритмические приемы для сжатия моделей при развертывании.* Важным фактором практического использования нейронных сетей является различие требований к вычислительным ресурсам в процессе обучения и развертывания моделей. В то время как тренировка модели на протяжении нескольких недель с использованием больших объемов памяти вполне допустима, ее обученный вариант может выполняться на мобильном телефоне в условиях жестких ограничений как по памяти, так и по вычислительной мощности. Поэтому был разработан ряд приемов для сжатия моделей на стадии тестирования. Этот тип сжатия часто позволяет улучшить производительность работы с кеш-памятью и повысить эффективность вычислений.

Обсудим некоторые из подобных методик ускорения и сжатия моделей.

3.7.1. Ускорение с помощью GPU

Первоначально GPU были разработаны для визуализации графики на экранах с использованием списков 3-мерных координат. Поэтому графические карты изначально проектировались для параллельного выполнения многочисленных операций матричного умножения с целью быстрого воспроизведения графики. GPU претерпели значительную эволюцию, в ходе которой их возможности расширились далеко за рамки первоначального предназначения как оборудования для визуализации графики. Как и графические приложения, реализации нейронных сетей требуют выполнения огромного количества операций перемножения матриц больших размеров, для чего и приспособлены GPU. В традиционной нейронной сети прямое распространение включает операции умножения матрицы на вектор, тогда как в сверточной нейронной сети умножаются две матрицы. Если используется мини-пакетный подход, то активации в традиционной нейронной сети становятся матрицами (вместо векторов). Поэтому процесс прямого распространения требует выполнения матричных умножений. То же самое справедливо и в отношении обратного распространения, в ходе которого матрицы перемножаются для распространения производных в обратном направлении.

Иными словами, большинство интенсивных вычислений включает операции над векторами, матрицами и тензорами. Даже один GPU неплохо справляется с распараллеливанием этих операций между своими ядрами за счет многопоточности [203], когда некоторые группы потоков, разделяющие один и тот же код, выполняются одновременно. Это так называемый *принцип SIMT* (Single Instruction Multiple Threads — одна инструкция, множество потоков). И хотя CPU также поддерживают параллельную обработку коротких векторных данных с помощью *инструкций SIMD* (Single Instruction Multiple Data — одна инструкция, множество данных), степень параллелизма в данном случае значительно ниже той, которая достигается в GPU. Использованию GPU сопутствуют дополнительные издержки по сравнению с CPU. GPU отлично справляются с многократно повторяющимися операциями, но сталкиваются с трудностями при выполнении операций ветвления наподобие тех, которые требуют инструкции типа *if-then*. Большинство ресурсоемких операций, выполняемых при обучении нейронных сетей, составляют повторные операции матричного умножения, выполняемые над данными различных тренировочных примеров, что соответствует назначению GPU. Несмотря на пониженную тактовую частоту одиночных инструкций в GPU по сравнению с традиционными CPU, эффективность параллельных вычислений с помощью GPU настолько высока, что это приносит огромный выигрыш в производительности.

Потоки GPU группируются в небольшие блоки — так называемые *варпы* (warps). Все потоки в блоке разделяют один и тот же код в каждом цикле, и это ограничение обеспечивает одновременное выполнение потоков. Реализация должна тщательно продумываться таким образом, чтобы снизить использование полосы пропускания памяти. Это достигается за счет *совмещения* операций записи и чтения из различных потоков в одной транзакции доступа к памяти. Рассмотрим типичную для нейронных сетей операцию умножения матриц. При умножении матриц за вычисление каждого элемента произведения отвечает отдельный поток. Предположим, например, что матрица размера 100×50 умножается на матрицу размера 50×200 . В этом случае для вычисления элементов результирующей матрицы будет запущено $100 \times 200 = 20\,000$ потоков. Как правило, эти потоки будут сгруппированы в несколько блоков (варпов), каждый из которых обеспечивает высокую степень параллелизма. Тем самым вычисления ускоряются. Выполнение операций матричного умножения с помощью GPU обсуждается в [203].

Основным лимитирующим фактором при высокой степени параллелизма часто является полоса пропускания памяти, определяющая скорость доступа процессора к месту хранения соответствующих параметров в памяти. Графические процессоры обеспечивают более высокую степень параллелизма и полосу пропускания памяти по сравнению с традиционными CPU. Подчеркнем,

что при недостаточно быстром доступе к параметрам, хранящимся в памяти, увеличение скорости выполнения операций не приведет к увеличению скорости вычислений. В подобных случаях скорость обмена данными с памятью не может сравниться со скоростью CPU- или GPU, и ядра CPU/GPU будут простаивать. GPU характеризуются достижением различных компромиссов между скоростями доступа к кешу, вычислений и доступа к оперативной памяти. CPU располагают гораздо большими объемами блоков кеш-памяти по сравнению с GPU и именно в ней сохраняют промежуточные результаты операций, такие, например, как результат умножения двух чисел. На извлечение вычисленного значения из кеша уйдет гораздо меньше времени, чем на повторное вычисление произведения, и в этом отношении CPU превосходят GPU. Однако это преимущество нивелируется в случае нейронных сетей, когда размеры матриц параметров и активаций часто оказываются настолько большими, что они не могут уместиться в кеш-памяти CPU. Несмотря на то что размеры кеша CPU значительно больше размеров кеша GPU, этого все равно недостаточно для обработки данных в тех масштабах, которые типичны для операций в нейронных сетях. В подобных ситуациях может выручить лишь широкая полоса пропускания памяти, и именно в этом GPU обладают явным преимуществом по сравнению с CPU. Кроме того, при работе с GPU часто оказывается так, что будет быстрее повторить вычисление, чем извлечь готовый результат из памяти (при условии, что он недоступен в кеше). Поэтому по своему исполнению GPU-реализации несколько отличаются от традиционных CPU-реализаций. В дополнение к этому получаемые преимущества могут быть чувствительны к выбору архитектуры нейронной сети, поскольку выигрыш, обеспечиваемый шириной полосы пропускания памяти и многопоточностью, может быть различным для различных архитектур.

После рассмотрения приведенных выше примеров у читателей могло сложиться впечатление, что использование GPU требует написания больших объемов низкоуровневого кода, и это действительно так, потому что создание GPU-ориентированного пользовательского кода для каждой архитектуры нейронной сети представляет собой чрезвычайно трудоемкую задачу. Учитывая это, такие компании, как NVIDIA, сделали модульным интерфейс между приложениями и GPU-реализацией. Ключевую роль здесь играет тот факт, что ускорение выполнения таких примитивов, как операции матричного умножения и свертки, может быть скрыто от пользователя за счет предоставления библиотеки операций для нейронных сетей, которая “за кулисами” обеспечивает ускоренное выполнение этих операций. Библиотека GPU тесно интегрируется с такими фреймворками глубокого обучения, как Caffe или Torch, чтобы можно было в полной мере использовать преимущества ускорения выполнения операций с помощью GPU. В качестве конкретного примера такой библиотеки

можно привести *NVIDIA CUDA Deep Neural Network Library* [643], которую для краткости называют библиотекой *cuDNN*. CUDA (Compute Unified Device Architecture) — это платформа для параллельных вычислений и модель программирования, которая работает на совместимых GPU-процессорах. Библиотека предлагает абстракцию и программный интерфейс, использование которых требует внесения в код лишь относительно небольших изменений. Библиотека *cuDNN* может интегрироваться со многими фреймворками глубокого обучения, такими как *Caffe*, *TensorFlow*, *Theano* и *Torch*. Изменения, которые требуется ввести в CPU-версию обучающего кода конкретной нейронной сети для его преобразования в GPU-версию, часто оказываются совсем небольшими. Например, в случае *Torch* пакет *CUDA Torch* включается в начало кода, а различные структуры данных (такие, как тензоры) инициализируются как CUDA-тензоры (а не как обычные тензоры). После внесения умеренных изменений описанного типа практически один и тот же код может выполняться в *Torch* на GPU, а не на CPU. Вышесказанное остается справедливым и в отношении других фреймворков глубокого обучения. Подход такого типа освобождает разработчиков от настройки низкоуровневой производительности, которую необходимо выполнять в GPU-фреймворках, поскольку включенные в библиотеку примитивы уже содержат код, учитывающий все низкоуровневые детали организации параллельных вычислений на GPU.

3.7.2. Параллельные и распределенные реализации

Существует возможность дальнейшего ускорения процесса обучения за счет использования нескольких CPU или GPU. Поскольку чаще применяют несколько GPU, мы сосредоточимся именно на этом варианте. Параллелизм не сводится к работе нескольких GPU одновременно, поскольку существуют накладные расходы, обусловленные необходимостью коммуникации различных процессоров между собой. Задержки, связанные с этими накладными расходами, недавно были уменьшены с помощью специализированных сетевых карт, обеспечивающих обмен данными между GPU. Кроме того, ускорение такой коммуникации может достигаться за счет использования таких алгоритмических приемов, как 8-битовые аппроксимации градиентов [98]. Существуют разные способы распределения работы между процессорами, а именно: параллелизм гиперпараметров, параллелизм моделей и параллелизм данных. Перейдем к рассмотрению этих методов.

Параллелизм гиперпараметров

Простейшим из возможных способов достижения параллелизма в процессе обучения, не требующим больших накладных расходов, является тренировка нейронных сетей с установкой различных параметров на разных процессорах.

Это исключает необходимость в обмене информацией между различными потоками выполнения, что позволяет избежать излишних накладных расходов. Как уже отмечалось ранее, циклы выполнения с неоптимальными гиперпараметрами часто преждевременно прекращаются задолго до их завершения. Тем не менее небольшое количество различных запусков с оптимизированными параметрами часто используют для создания ансамбля моделей. Тренировка различных ансамблевых составляющих может выполняться независимо на разных процессорах.

Параллелизм моделей

Параллелизм моделей особенно полезен в ситуациях, когда размеры одиночной модели настолько велики, что она не умещается в GPU. В подобных случаях скрытый слой распределяется по различным GPU. Различные GPU работают с одним и тем же пакетом тренировочных примеров, хотя разные GPU вычисляют разные части активаций и градиентов. Каждый GPU содержит лишь ту часть матрицы весов, которая умножается на скрытые активации, размещенные в GPU. При этом необходимость в обмене результатами активации с другими GPU все еще сохраняется. Точно так же сохраняется необходимость в получении производных, относящихся к скрытым элементам других GPU, для вычисления градиентов весов, которые включают смешанные производные, относящиеся к другим GPU. Это достигается за счет использования межпроцессорных связей GPU, что приводит к дополнительным накладным расходам. В некоторых случаях допускается исключение этих взаимосвязей для некоторого подмножества слоев с целью снижения накладных расходов (хотя результирующая модель уже не будет полностью той же, что и ее последовательная версия). Параллелизм моделей не слишком полезен при небольшом количестве параметров нейронной сети, и его следует применять лишь в крупных сетях. В качестве неплохого примера практической реализации параллелизма моделей можно привести сверточную нейронную сеть *AlexNet* (раздел 8.4.1). Последовательная и распределенная версии *AlexNet* представлены на рис. 8.9. Обратите внимание на то, что приведенная на этом рисунке версия не эквивалентна распределенной версии, включающей слои с исключенным межпроцессорным взаимодействием GPU. Обсуждение параллелизма моделей содержится, в частности, в [74].

Параллелизм данных

Параллелизм данных лучше всего работает в ситуациях, когда размеры модели достаточно невелики для того, чтобы она могла уместиться в каждом отдельном GPU, при большом объеме тренировочных данных. В этом случае параметры разделяются всеми GPU, а задачей обновления является использование различных процессоров с различными тренировочными примерами для

ускорения соответствующих вычислений. Проблема в том, что идеальная синхронизация обновлений может замедлить процесс, поскольку для этого будут задействованы механизмы блокировки. Ключевую роль здесь играет тот факт, что каждый процессор будет вынужден дожидаться, пока другие процессоры не завершат свои обновления. В результате самый медленный процессор создаст “узкое место”. В [91] был предложен метод, использующий асинхронный стохастический градиентный спуск. Суть идеи заключается в применении сервера параметров для организации разделения параметров различными GPU. Обновления осуществляются без использования каких-либо механизмов блокировки. Иными словами, каждый GPU может в любой момент времени читать разделяемые параметры, а после выполнения необходимых вычислений записывать их на сервер параметров, не заботясь о блокировках. В таком случае эффективность процесса все еще может снижаться за счет того, что один GPU может замедлять прогресс другого, но простой в ожидании возможности выполнения записи будут отсутствовать. В результате общая скорость процесса все равно будет выше, чем при использовании синхронизированного механизма. Метод распределенного асинхронного градиентного спуска является популярной стратегией параллелизма в крупномасштабных проектах промышленного уровня.

Использование компромиссных решений для организации гибридного параллелизма

Из приведенного выше обсуждения должно быть очевидно, что параллелизм на уровне моделей хорошо подходит для моделей с большим количеством параметров, в то время как параллелизм данных — для небольших моделей. Оказывается, существует возможность комбинировать оба типа параллелизма, используя их в различных частях сети. В некоторых типах сверточных нейронных сетей, включающих полносвязные слои, подавляющее большинство параметров относится к полносвязным слоям, тогда как более интенсивные вычисления выполняются в предыдущих слоях. В подобных случаях имеет смысл использовать в ранних слоях параллелизм данных, а в поздних — параллелизм моделей. Подобный подход называют *гибридным параллелизмом*. Обсуждение подходов такого типа содержится в [254].

3.7.3. Алгоритмические приемы сжатия модели

Обычно требования, предъявляемые к памяти и производительности во время тренировки нейронной сети и ее развертывания, различаются. Если тренировка нейронной сети в течение недели для распознавания лиц на фотографии может считаться вполне допустимой, то конечный пользователь хотел бы, чтобы сеть распознавала лица в течение нескольких секунд. Кроме того, модель может развертываться на мобильном устройстве с небольшим объемом памяти и

ограниченными вычислительными ресурсами. В подобных случаях обеспечение возможности эффективного использования обученной модели в условиях ограниченного размера свободной памяти приобретает первостепенное значение. Обычно при развертывании сети проблемы с производительностью не возникают, поскольку для предсказания тестового примера часто достаточно выполнить лишь простые операции матричного умножения в нескольких слоях. С другой стороны, нехватка памяти часто становится проблемой из-за большого количества параметров в многослойных сетях. Существует ряд технических приемов, которые применяются для сжатия модели в подобных ситуациях. В большинстве случаев крупная обученная нейронная сеть видоизменяется так, чтобы ей требовалось меньше места, путем аппроксимации некоторых частей модели. Кроме того, определенного улучшения производительности во время предсказания можно достигнуть за счет повышения производительности кеш-памяти и уменьшения количества выполняемых операций, хотя это не является основной целью. Интересно отметить, что иногда такая аппроксимация даже *улучшает* точность предсказаний для неизвестных образцов благодаря эффектам регуляризации, особенно если размер исходной модели завышен по сравнению с размером тренировочного набора данных.

Прореживание весов при тренировке сети

В нейронных сетях со связями ассоциируются веса. Если абсолютное значение какого-то веса невелико, то он не будет оказывать существенного влияния на модель. Тогда можно безболезненно отбросить (исключить) такие веса и выполнить тонкую настройку сети, начиная с текущих весов связей, которые еще не были отброшены. Выбрав большое пороговое значение, определяющее, какие веса должны быть отброшены, можно значительно уменьшить размер модели. В подобных случаях очень важно выполнить тонкую настройку оставшихся весов на протяжении последующих эпох тренировки. Отбрасывание связей можно стимулировать, используя L_1 -регуляризацию, о чем пойдет речь в главе 4. Если использовать L_1 -регуляризацию в процессе тренировки, то многие веса в любом случае будут иметь нулевые значения ввиду естественных математических свойств этой формы регуляризации. В то же время, как было показано в [169], преимуществом L_2 -регуляризации является более высокая точность. Поэтому в [169] используется именно L_2 -регуляризация, тогда как веса ниже определенного порога обрезаются.

Дальнейшие улучшения были описаны в работе [168], в которой данный подход был совмещен с кодированием Хаффмана и квантизацией для сжатия. Задача квантизации — уменьшить количество битов, представляющих каждое соединение. Благодаря использованию этого подхода необходимый для сети *AlexNet* [255] объем памяти удалось уменьшить в 35 раз, с 240 до 6,9 Мбайт, без

потери точности. Вместо того чтобы хранить такую модель во внепроцессорной динамической памяти DRAM, ее можно поместить во внутрипроцессорную статическую кеш-память SRAM, что благоприятно скажется на длительности предсказаний.

Использование избыточности весов

В [94] было показано, что преобладающая часть весов в нейронной сети является избыточной. Иначе говоря, любую матрицу весов W размера $m \times n$, которая связывает два слоя, содержащие m_1 и m_2 элементов соответственно, можно записать в виде $W \approx UV^T$, где матрицы U и V имеют размеры $m_1 \times k$ и $m_2 \times k$ соответственно. Кроме того, предполагается, что $k \ll \min\{m_1, m_2\}$. Это явление объясняется рядом особенностей процесса тренировки. Например, признаки и веса в нейронной сети стремятся *коадаптироваться*, поскольку различные части сети обучаются с различными скоростями. Поэтому более быстрые части сети часто адаптируются к более медленным. В результате в сети создается заметная избыточность как в терминах признаков, так и в терминах весов, и выразительная способность сети никогда не используется в полную силу. В этом случае можно заменить пару слоев (содержащих весовую матрицу W) тремя слоями с размерами m_1 , k и m_2 . Матрицей весов соединений, связывающих первую пару слоев, является матрица U , а матрицей весов соединений, связывающих вторую пару слоев, — матрица V^T . Несмотря на большую глубину новой матрицы, она лучше регуляризована, поскольку разность $W - UV^T$ содержит только шум. Кроме того, матрицы U и V требуют $(m_1 + m_2) \cdot k$ параметров, что меньше количества параметров матрицы W , при условии, что k меньше, чем половина гармонического среднего m_1 и m_2 :

$$\frac{\text{Параметры } W}{\text{Параметры } U, V} = \frac{m_1 \cdot m_2}{k(m_1 + m_2)} = \frac{\text{ГАРМОНИЧЕСКОЕ_СРЕДНЕЕ}(m_1, m_2)}{2k}.$$

Как показано в [94], более 95% параметров нейронной сети являются избыточными, поэтому низкого значения ранга k достаточно для аппроксимации.

Важно подчеркнуть, что замена матрицы W матрицами U и V должна осуществляться после завершения обучения W . Например, если мы заменим пару слоев, соответствующих матрице W , тремя слоями, содержащими две матрицы весов U и V^T , и обучим их с самого начала, то, возможно, не получим удовлетворительных результатов. Это произойдет в силу уже упомянутой коадаптации в процессе тренировки, и ранг результирующих матриц U и V станет еще меньше, чем k . В результате мы можем столкнуться с недообучением сети.

Наконец, можно обеспечить еще большее сжатие модели, заметив, что в одновременном обучении матриц U и V нет необходимости, поскольку они взаимно избыточны. Для любой матрицы U ранга k можно обучить матрицу V таким

образом, чтобы произведение UV^T имело то же значение. В связи с этим в [94] были предоставлены методы, предусматривающие фиксацию матрицы U с последующим обучением матрицы V .

Сжатие на основе хеширования

Количество параметров, подлежащих хранению, можно уменьшить, принудительно присваивая случайно выбранным элементам матрицы весов разделяемые значения параметров. Случайный выбор осуществляется путем применения хеш-функции к элементу (i, j) матрицы. В качестве примера рассмотрим матрицу весов размера 100×100 , имеющую 104 элемента. В этом случае можно хешировать каждый вес значением из интервала $\{1 \dots 1000\}$ для создания 1000 групп. Каждая из этих групп будет содержать среднее по 10 связям, разделяющим веса. Обратное распространение ошибки может обрабатывать разделяемые веса, используя подход, который обсуждался в разделе 3.2.9. При таком подходе для матрицы потребуется только 1000 ячеек памяти, что составляет 10% от первоначальных потребностей. Заметим, что ту же степень сжатия могло бы обеспечить использование матрицы размера 100×10 , однако существует одно важное обстоятельство, суть которого заключается в том, что использование разделяемых весов наносит меньший ущерб выразительной способности весов, чем тот, который нанесло бы *априорное* уменьшение размера матрицы весов. Более подробно этот подход обсуждается в [66].

Использование MIMIC-моделей

В [13, 55] получен ряд интересных результатов. В частности, существует возможность значительного сжатия модели путем создания нового тренировочного набора данных из обученной модели, который легче моделировать. Эти более легкие тренировочные данные можно использовать для обучения намного меньшей сети без заметной потери точности. Такую уменьшенную модель называют *моделью MIMIC* (Multiple Indicators and Multiple Causes). Для создания MIMIC-модели необходимо выполнить следующие действия.

1. На основе исходных тренировочных данных создается модель. Эта модель может иметь очень большие размеры и потенциально может создаваться на основе ансамбля различных моделей, что приведет к дальнейшему увеличению количества параметров. Использование такого подхода в условиях нехватки памяти было бы неуместно. Предполагается, что выходами модели являются Softmax-вероятности различных классов. Эту модель также называют *моделью-учителем*.
2. Создаются новые тренировочные данные путем пропускаемых неразмеченных примеров через обученную сеть. В качестве целевых значений вновь созданных тренировочных данных устанавливаются

Softmax-вероятностные значения выходов обученной модели для неразмеченных примеров. Поскольку неразмеченные данные часто имеются в избытке, этот способ позволяет создавать большие объемы тренировочных данных. Следует подчеркнуть, что новые тренировочные данные содержат вероятностные целевые значения, а не дискретные, как в исходных тренировочных данных, что играет значительную роль при создании сжатой модели.

3. Новый тренировочный набор данных (с искусственно сгенерированными метками) используется для тренировки намного меньшей и гораздо более узкой сети. Исходные тренировочные данные вообще не используются. Именно эта уменьшенная модель, которую называют *МIMIC-моделью* или *моделью-студентом* (student model), развертывается в условиях ограниченности объемов доступной памяти. Можно показать, что по точности МIMIC-модель, несмотря на свои небольшие размеры, не уступает значительно модели, обученной с помощью исходной нейронной сети.

У читателей может возникнуть вполне резонный вопрос: благодаря чему модель МIMIC, имеющая меньшую глубину и меньшее количество параметров, способна работать не хуже исходной модели? Попытки построить мелкую модель на исходных данных не позволяют добиться той точности, которую обеспечивает модель МIMIC. В литературе было предложено несколько возможных причин успешности модели МIMIC [13].

Если исходные тренировочные данные содержат ошибки, обусловленные неверной маркировкой признаков, то это может приводить к необоснованному усложнению обученной модели. Использование новых тренировочных данных в значительной степени снижает указанный риск.

Если в пространстве решений существуют сложные области, то модель-учитель упрощает их, предоставляя вероятностные оценки вместо дискретных меток. Сложность устраняется за счет фильтрации целевых значений через модель-учителя.

Исходные тренировочные данные содержат бинарные целевые значения 0/1, тогда как вновь создаваемые данные содержат более информативные вероятностные значения. Это особенно полезно в случае прямого кодирования целевых значений с множественными метками, когда между различными классами существуют отчетливые корреляции.

Исходные целевые значения могут зависеть от входов, которые недоступны в тренировочных данных. С другой стороны, метки, созданные моделью-учителем, зависят только от имеющихся входов. Это упрощает обучение модели и устраняет необъясненные сложности. Необъясненная сложность часто требует излишнего увеличения количества параметров и глубины.

Некоторые из перечисленных выше преимуществ можно рассматривать как своего рода эффекты регуляризации. Интересные результаты получены в [13], где показано, что глубокие сети не являются теоретически необходимыми для моделей MIMIC, хотя практическая необходимость в регуляризационном эффекте глубины существует при работе с исходными тренировочными данными. Модель MIMIC извлекает все выгоды из этого регуляризационного эффекта, но не за счет глубины, а за счет искусственно созданных целевых значений.

3.8. Резюме

В этой главе обсуждалась задача обучения глубоких нейронных сетей. Мы более подробно рассмотрели алгоритм обратного распространения ошибки и его подводные камни, тщательно проанализировали проблемы затухающих и взрывных градиентов, а также трудности, обусловленные неодинаковой чувствительностью функции потерь к различным переменным оптимизации. Для некоторых типов активаций, таких как ReLU, эти проблемы менее критичны. В то же время использование ReLU иногда может приводить к мертвым нейронам при неудачно подобранной скорости обучения. Для более эффективного функционирования сети также важно правильно выбрать тип градиентного спуска. Модифицированные методы стохастического градиентного спуска включают использование алгоритмов *AdaGrad*, *AdaDelta*, *RMSProp* и *Adam*, а также метода импульсов Нестерова. Все эти методы стимулируют ускорение процесса обучения.

Для устранения проблемы оврагов с крутыми склонами был предложен ряд подходов, основанных на использовании методов оптимизации второго порядка. В частности, подход, известный под названием “оптимизация без гессиана”, позволяет справиться со многими базовыми проблемами оптимизации. Представляет интерес один из недавно представленных методов, заключающийся в использовании пакетной нормализации. Преобразуя данные слой за слоем, пакетная нормализация обеспечивает оптимальное масштабирование различных переменных. Использование пакетной нормализации приобрело широкую популярность в различных типах глубоких сетей. Также был предложен ряд методов ускорения и сжатия алгоритмов нейронных сетей. Ускорение часто достигается за счет усовершенствования аппаратных средств, а сжатие — за счет использования различных алгоритмических ухищрений.

3.9. Библиографическая справка

Первоначальная идея обратного распространения ошибки базировалась на концепции дифференцирования композиции функций, разработанной в теории управления [54, 237] в рамках *автоматического дифференцирования*.

Адаптация этих методов для нейронных сетей была предложена Полом Вербосом в его кандидатской диссертации в 1974 году [524], хотя более современная форма алгоритмов была предложена Румельхартом и др. в 1986 году [408]. Обсуждение истории развития алгоритма обратного распространения ошибки можно найти в книге Пола Вербоса [525].

Обсуждение алгоритмов для оптимизации гиперпараметров в нейронных сетях и других алгоритмов машинного обучения содержится в [36, 38, 490]. Обсуждению метода случайного поиска для оптимизации гиперпараметров посвящена работа [37]. Использование *байесовской оптимизации* для настройки гиперпараметров обсуждается в [42, 306, 458]. Существуют специализированные библиотеки для байесовской настройки, такие как *Hyperopt* [614], *Spearmint* [616] и *SMAC* [615].

Правило, согласно которому начальные веса должны зависеть от количества входов (r_{in}) и выходов (r_{out}) узла в пропорции $2 / (r_{in} + r_{out})$, основано на результатах [140]. Анализ методов инициализации для линейно-кусочных сетей приведен в [183]. Оценки и анализ влияния предварительной обработки признаков на обучение нейронных сетей содержится в [278, 532]. Использование кусочно-линейных элементов для решения ряда проблем обучения обсуждается в [141].

Описание алгоритма Нестерова для градиентного спуска можно найти в [353]. Метод *delta-bar-delta* был предложен в [217]. Алгоритм *AdaGrad* был предложен в [108]. Алгоритм *RMSProp* обсуждается в [194]. Другой адаптивный алгоритм, *AdaDelta*, использующий стохастический градиентный спуск, обсуждается в [553]. Эти методы имеют кое-что общее с методами второго порядка и, в частности с методом, описанным в [429]. Алгоритм *Adam*, являющийся дальнейшим развитием идей в этом русле, описан в [241]. Практическая важность инициализации и учета импульсов в глубоком обучении обсуждается в [478]. Использование координатного спуска было предложено в [273]. Стратегия *усреднения Поляка* обсуждается в работе [380].

Некоторые сложные задачи, связанные с проблемами затухающих и взрывных градиентов, описаны в [140, 205, 368]. Идеи относительно методов инициализации параметров, позволяющих избежать этих проблем, обсуждаются в [140]. Правило отсечения градиентов приведено Миколовым в его кандидатской диссертации [324]. Обсуждение метода отсечения градиентов в контексте рекуррентных нейронных сетей содержится в [368]. Функция активации ReLU была введена в [167], а некоторые из ее интересных свойств исследуются в [141, 221].

Описание некоторых градиентных методов второго порядка (таких, как метод Ньютона) предоставлено в [41, 545, 300]. Базовые принципы метода сопряженных градиентов описаны в нескольких классических книгах и статьях [41, 189, 443], тогда как в [313, 314] обсуждается применение этих методов

в нейронных сетях. В [316] для ускорения градиентного спуска используется представление матрицы кривизны в виде произведения Кронекера. Другой способ аппроксимации метода Ньютона предлагают квазиньютоновские методы [273, 300], причем простейшей аппроксимацией является диагональный гессиан [24]. Аббревиатура BFGS составлена из первых букв фамилий авторов алгоритма Бройдена — Флетчера — Гольдфарба — Шанно. Вариант BFGS под названием LBFGS [273, 300] не требует использования больших объемов памяти. Еще одним популярным методом второго порядка является алгоритм Левенберга — Марквардта. Однако этот подход определен для квадрата функции потерь и не может применяться в отношении многих форм кросс-энтропийных и логарифмических функций потерь, столь обычных для нейронных сетей. Обзоры этого подхода содержатся в [133, 300]. Общее обсуждение различных типов методов нелинейного программирования содержится в [23, 39].

Стабильность нейронных сетей в отношении локальных минимумов обсуждается в [88, 426]. Недавно в [214] были введены методы пакетной нормализации. Метод, в котором для пакетной нормализации применяется отбеливание, обсуждается в [96], хотя данный подход не представляется практичным. Пакетная нормализация требует незначительной настройки в случае рекуррентных сетей [81], хотя применительно к рекуррентным сетям более эффективна *последовательная нормализация* [14]. В этом методе (раздел 7.3.1) используется нормализация всех элементов слоя с помощью одного и того же примера, а не мини-пакетная нормализация одного элемента. Такой подход хорошо работает в рекуррентных сетях. Близким аналогом пакетной нормализации является нормализация весов [419], в которой величина и направление векторов весов разделяются в процессе обучения. Трюки обучения подобного рода обсуждаются в [362].

Расширенное обсуждение методов ускорения алгоритмов машинного обучения с помощью GPU содержится в [644]. Различные типы методов параллелизации для GPU описаны в [74, 91, 254], а конкретное обсуждение применительно к сверточным нейронным сетям приведено в [541]. Сжатие моделей путем регуляризации обсуждается в [168, 169]. Похожий метод сжатия моделей предложен в [213]. Использование моделей MIMIC с целью сжатия рассматривается в [55, 13]. Сходный подход обсуждается в [202]. Использование избыточности параметров для сжатия нейронных сетей описано в [94], а сжатие нейронных сетей путем хеширования — в [66].

3.9.1. Программные ресурсы

Все алгоритмы обучения, которые рассматривались в этой главе, поддерживаются многочисленными фреймворками глубокого обучения, такими как *Caffe* [571], *Torch* [572], *Theano* [573] и *TensorFlow* [574]. Доступны расширения *Caffe* для Python и MATLAB. Все эти фреймворки предоставляют целый

ряд алгоритмов обучения, представленных в данной главе. Опции для пакетной нормализации доступны в этих фреймворках в виде отдельных слоев. Несколько библиотек предоставляют байесовскую оптимизацию гиперпараметров. В их число входят библиотеки *Hyperopt* [614], *Spearmint* [616] и *SMAC* [615]. Несмотря на то что указанные библиотеки предназначены для решения небольших задач машинного обучения, в некоторых случаях их удается использовать для более крупных задач. Ссылки на NVIDIA cuDNN приведены в [643], а различные фреймворки, поддерживаемые cuDNN, обсуждаются в [645].

3.10. Упражнения

1. Рассмотрим следующую рекурсию:

$$(x_{t+1}, y_{t+1}) = (f(x_t, y_t), g(x_t, y_t)), \quad (3.66)$$

где $f()$ и $g()$ — функции многих переменных.

- A. Представьте производную $\frac{\partial x_{i+2}}{\partial x_i}$ в виде выражения, включающего только x_i и y_i .
 - B. Можете ли вы начертить архитектуру нейронной сети, соответствующей приведенной выше рекурсии, если t принимает значения в интервале от 1 to 5? Предположите, что нейроны могут вычислять любую требуемую функцию.
2. Рассмотрим нейрон с двумя входами, который перемножает входы x_1 и x_2 для получения выхода o . Пусть L — функция потерь, вычисляемая в узле o . Предположим, вам известно, что $\frac{\partial L}{\partial o} = 5$, $x_1 = 2$, а $x_2 = 3$. Вычислите $\frac{\partial L}{\partial x_1}$ и $\frac{\partial L}{\partial x_2}$.
 3. Рассмотрим нейронную сеть с двумя слоями, включая входной слой. Первый (входной) слой содержит четыре входа: x_1 , x_2 , x_3 и x_4 . Второй слой имеет шесть скрытых элементов, соответствующих всем операциям парного умножения. Выходной узел o просто суммирует значения шести скрытых элементов. Пусть L — функция потерь выходного узла. Предположим, вам известно, что $\frac{\partial L}{\partial o} = 2$, $x_1 = 1$, $x_2 = 2$, $x_3 = 3$, а $x_4 = 4$. Вычислите $\frac{\partial L}{\partial x_i}$ для каждого i .
 4. Как бы вы выполнили предыдущее упражнение с измененным условием, при котором выход o вычисляется как максимальное из шести его значений, а не как их сумма?

5. В данной главе обсуждалось (см. табл. 3.1), как выполнить обратное распространение произвольной функции, используя умножение на якобиан. Объясните, почему необходимо быть внимательным при использовании подхода, основанного на матрицах. (Подсказка: вычислите якобиан для сигмоиды.)
6. Рассмотрим функцию потерь $L = x^2 + y^{10}$. Реализуйте простой алгоритм крутейшего спуска для графического построения координат по мере их изменения от начальной точки до оптимального значения 0. Рассмотрите две различные начальные точки (0,5; 0,5) и (2; 2) и отобразите на графике траектории в этих двух случаях при постоянной скорости обучения. Что можно сказать о поведении алгоритма в этих двух случаях?
7. Гессиан H сильно выпуклой квадратичной функции всегда удовлетворяет условию $\bar{x}^T H \bar{x} > 0$ при любом ненулевом векторе \bar{x} . Покажите, что в таких задачах все сопряженные направления являются линейно независимыми.
8. Покажите, что в случае, если скалярные произведения d -мерного вектора v и d линейно независимых векторов равны нулю, то v должен быть нулевым вектором.
9. В этой главе обсуждались два варианта обратного распространения ошибки, в которых для рекурсии динамического программирования используются преактивационные и постаktivационные значения соответственно. Покажите, что эти два варианта обратного распространения математически эквивалентны.
10. Рассмотрим функцию активации Softmax в выходном слое, в котором существенные числовые выходы $v_1 \dots v_k$ преобразуются в вероятности следующим образом (согласно уравнению 3.20):

$$o_i = \frac{\exp(v_i)}{\sum_{j=1}^k \exp(v_j)} \quad \forall i \in \{1, \dots, k\}.$$

- А. Покажите, что производная $\frac{\partial o_i}{\partial v_j}$ равна $o_i(1 - o_i)$, если $i = j$, и $-o_i o_j$, если $i \neq j$.
- Б. Используйте приведенный выше результат для доказательства справедливости уравнения 3.22:

$$\frac{\partial L}{\partial v_i} = o_i - y_i.$$

Предполагается, что мы используем кросс-энтропийную функцию потерь

$L = -\sum_{i=1}^k y_i \log(o_i)$, где $y_i \in \{0, 1\}$ — метки классов в представлении прямого кодирования для различных значений $i \in \{1 \dots k\}$.

11. В этой главе для итеративного генерирования сопряженных направлений использовались направления крутейшего спуска. Предположим, мы выбрали d произвольных линейно-независимых направлений $\bar{v}_0 \dots \bar{v}_{d-1}$. Покажите, что (при подходящем выборе β_n) мы можем начать с $\bar{q}_0 = \bar{v}_0$ и генерировать последовательные сопряженные направления в следующем виде:

$$\bar{q}_{t+1} = \bar{v}_{t+1} + \sum_{i=0}^t \beta_i \bar{q}_i.$$

Объясните, почему этот подход более дорогостоящий по сравнению с тем, который обсуждался в данной главе.

12. Определение β_i в разделе 3.5.6.1 гарантирует, что направление \bar{q}_i сопряжено с направлением \bar{q}_{t+1} . Данное упражнение демонстрирует систематическим образом, что *любое* направление \bar{q}_i для $i \leq t$ удовлетворяет условию $\bar{q}_i^T H \bar{q}_{t+1} = 0$. (Подсказка: докажите совместно B , V и Γ методом индукции по t , начав с A .)

A. Вспомните (см. уравнение 3.51), что $H \bar{q}_i = [\nabla L(\bar{W}_{i+1}) - \nabla L(\bar{W}_i)] / \delta_i$ для квадратичной функции потерь, где δ_i зависит от i -го размера шага. Объедините это условие с уравнением 3.49 для того, чтобы продемонстрировать справедливость следующего соотношения для всех $i \leq t$:

$$\delta_i [\bar{q}_i^T H \bar{q}_{t+1}] = -[\nabla L(\bar{W}_{i+1}) - \nabla L(\bar{W}_i)]^T [\nabla L(\bar{W}_{t+1})] + \delta_i \beta_i (\bar{q}_i^T H \bar{q}_i).$$

Также покажите, что $[\nabla L(\bar{W}_{t+1}) - \nabla L(\bar{W}_i)] \cdot \bar{q}_i = \delta_i \bar{q}_i^T H \bar{q}_i$.

B. Покажите, что $\nabla L(\bar{W}_{t+1})$ ортогонален каждому \bar{q}_i для $i \leq t$. (Доказательство для $i = t$ тривиально, поскольку градиент при завершении линейного поиска всегда ортогонален направлению поиска.)

B. Покажите, что градиенты функции потерь при $\bar{W}_0 \dots \bar{W}_{t+1}$ взаимно ортогональны.

Г. Покажите, что $\bar{q}_i^T H \bar{q}_{t+1} = 0$ для $i \leq t$ (случай $i = t$ тривиален).

Глава 4

Обучение глубоких сетей способности к обобщению

Все обобщения опасны, в том числе и это.
Александр Дюма

4.1. Введение

Нейронные сети — одаренные ученики, неоднократно доказавшие свою способность обучаться самым сложным функциям во многих областях. Однако необычайные возможности нейронных сетей одновременно являются их наиболее слабым местом. Если процесс обучения спроектирован недостаточно тщательно, они просто переобучаются на тренировочных данных. В переводе на язык практики термин *переобучение* (overfitting) означает, что сеть великолепно предсказывает тренировочные данные, на которых она была обучена, но плохо справляется с неизвестными ей контрольными примерами. Это обусловлено тем, что в процессе обучения сеть часто запоминает случайные артефакты тренировочных данных, которые плохо обобщаются на тестовые данные. Крайние формы проявления переобучения называют *заучиванием* (memorization). В данном случае уместна аналогия с ребенком, который способен решить любую аналитическую задачу из тех, с решениями которых он был ознакомлен ранее, однако не сможет решить новую для него задачу. Но если продолжать знакомить ребенка с решениями все большего и большего количества задач различного типа, то, вероятнее всего, он решит новую задачу, опираясь на шаблоны, многократно повторяющиеся в различных задачах и их решениях. Примерно то же самое происходит и в процессе машинного обучения, когда идентифицируются устойчивые закономерности, полезные для предсказаний. Так, если приложение для борьбы со спамом обнаруживает, что в электронных рассылках словосочетание “*Free Money!*” встречается тысячи раз, то обучаемая сеть обобщает

это правило для идентификации спама в электронных письмах, которые до этого ей не предоставлялись. С другой стороны, предсказания, базирующиеся на шаблонах, которые встречаются в совсем небольшом тренировочном наборе, включающем всего лишь два электронных письма, будут демонстрировать хорошие результаты для этих писем, но не для новой электронной почты. О возможностях обучаемой сети выдавать надежные предсказания для неизвестных образцов говорят как о ее способности к *обобщению* (generalization).

Способность сети обобщаться на неизвестные данные — свойство, которое имеет немаловажное практическое значение и по этой причине является своего рода чашей Грааля для всех приложений машинного обучения. Ведь если обучающие (тренировочные) примеры уже содержат метки, то их повторное предсказание не принесет никакой практической пользы. Например, если речь идет о приложении для создания подписей к изображениям, то мы заинтересованы в обучении сети тому, чтобы она была способна снабжать подписями те изображения, которые до этого ей еще не встречались.

Степень переобучения зависит от сложности модели и объема доступных данных. Сложность модели, определяемой нейронной сетью, зависит от количества базовых параметров. Параметры предоставляют дополнительные степени свободы, которые могут быть использованы для объяснения специфических тренировочных точек данных даже при плохой обобщаемости сети на неизвестные данные. Представим, например, ситуацию, в которой мы пытаемся предсказать значение переменной y на основании значения x , используя следующую формулу полиномиальной регрессии:

$$\hat{y} = \sum_{i=0}^d w_i x^i. \quad (4.1)$$

В этой модели для объяснения доступных нам пар (x, y) используются $(d + 1)$ параметров $w_0 \dots w_d$. Данная модель может быть реализована на базе нейронной сети с d входами, соответствующими x, x^2, \dots, x^d , и одиночным нейроном смещения, коэффициент которого равен w_0 . Функция потерь использует квадрат разности между наблюдаемым значением y и предсказанным значением \hat{y} . В общем случае, чем больше значение d , тем лучше улавливается нелинейность. В примере, приведенном на рис. 4.1, нелинейная модель с $d = 3$ будет согласовываться с данными лучше, чем модель с $d = 1$, при условии, что имеется *бесконечный (или очень большой) набор данных*. Однако в случае небольших наборов данных конечных размеров это не всегда будет так.

При наличии $(d + 1)$ или менее тренировочных пар (x, y) возможна точная подгонка данных с нулевой ошибкой *независимо от того, насколько хорошо эти пары отражают истинное распределение*. Рассмотрим в качестве примера ситуацию, когда доступны пять тренировочных точек. Можно показать, что



Рис. 4.1. Пример нелинейного распределения, для которого можно ожидать, что модель с $d = 3$ будет работать лучше, чем линейная модель с $d = 1$

точное согласование данных с нулевой ошибкой обеспечивается многочленом степени 4. Но это вовсе не означает, что нулевая ошибка будет обеспечена и для неизвестных данных. Пример такой ситуации приведен на рис. 4.2, где для трех наборов данных, каждый из которых содержит по пять случайно выбранных точек, представлены как линейная, так и полиномиальная модель. Совершенно очевидно, что линейная модель стабильна, хотя и неспособна точно моделировать кривизну истинного распределения данных. С другой стороны, несмотря на то что полиномиальная модель способна более точно моделировать истинное распределение данных, ее качество существенно меняется при переходе от одного набора к другому. Поэтому один и тот же тестовый пример при $x = 2$ (рис. 4.2) будет приводить к различным предсказаниям, получаемым с помощью полиномиальной модели для различных наборов тренировочных данных. Разумеется, с практической точки зрения такое поведение крайне нежелательно, поскольку нам хотелось бы, чтобы предсказания для конкретного тестового примера всегда совпадали, даже если используются разные выборки из набора тренировочных данных. Поскольку различные предсказания полиномиальной модели не могут быть корректными все одновременно, становится очевидным, что повышенная мощность полиномиальной модели по сравнению с линейной на самом деле увеличивает ошибку вместо того, чтобы уменьшать ее. Эта разница в предсказаниях, сделанных для одного и того же тестового примера (но взятого из различных тренировочных наборов данных), проявляется в виде дисперсии модели. Как следует из рис. 4.2, модели с высокой дисперсией стремятся запомнить артефакты тренировочных данных, что приводит к непоследовательности и неточности предсказаний на неизвестных тестовых примерах. Необходимо подчеркнуть, что полиномиальной модели высокой степени изначально присуща большая мощность по сравнению с линейной, поскольку значения

коэффициентов при более высоких степенях независимой переменной всегда могут быть установлены равными нулю. В то же время она не может развернуть свой потенциал в полную силу, если объем доступных данных ограничен. Проще говоря, наличие дисперсии, присущей ограниченным наборам данных, приводит к тому, что увеличение сложности модели становится контрпродуктивным. Данную ситуацию, требующую достижения разумного компромисса между мощностью модели и ее качеством, называют *дилеммой смещения — дисперсии* (bias — variance trade-off).

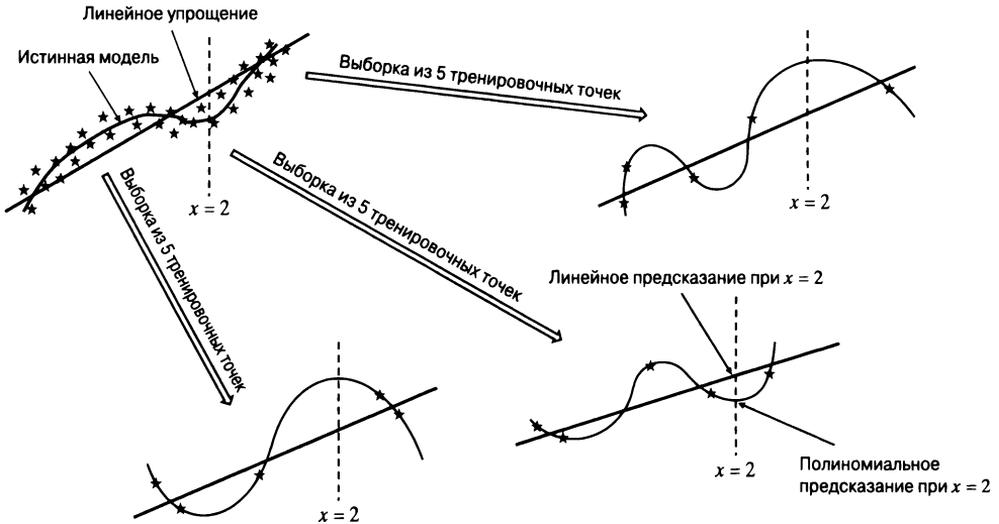


Рис. 4.2. Переобучение, обусловленное увеличением сложности модели. Линейная модель не испытывает существенных изменений с изменением тренировочных данных, тогда как в случае полиномиальной модели эти изменения проявляются в очень резкой форме. В результате непоследовательные предсказания полиномиальной модели при $x = 2$ часто оказываются более неточными, чем в случае линейной модели. Превосходство полиномиальной модели над линейной проявляется лишь в случае предоставления достаточно большого набора тренировочных данных

Существует несколько сигнальных индикаторов переобучения.

1. Если модель обучается на различных наборах данных, то для одного и того же тестового примера могут быть получены разные предсказания. Это служит признаком того, что модель запоминает нюансы конкретного тренировочного набора, а не обучается образам, которые обобщаются на неизвестные тестовые примеры. Обратите внимание на то, что три предсказания при $x = 2$ на рис. 4.2 заметно различаются для полиномиальной модели. Для линейной модели это не совсем так.

2. Расхождение между ошибками предсказания тренировочных и неизвестных примеров довольно велики. Заметьте, что предсказания в неизвестной тестовой точке $x = 2$ на рис. 4.2 часто более неточны в полиномиальной модели, чем в линейной. С другой стороны, ошибка обучения всегда равна нулю для полиномиальной модели, тогда как для нелинейной модели она всегда ненулевая.

Ввиду больших расхождений между ошибками обучения и тестирования модели часто тестируют на зарезервированной специально для этих целей части данных тренировочного набора. Эти неизвестные данные часто заранее удерживают и впоследствии используют для принятия различных типов алгоритмических решений, таких как настройка параметров. Такой набор точек называют *валидационным набором* (validation set). Окончательная точность тестируется на *отложенном*, или *внешнем*, наборе точек, которые не использовались ни для создания модели, ни для настройки параметров. Ошибку, полученную для такого набора данных, часто называют *ошибкой обобщения* (generalization error).

Нейронные сети характеризуются большими размерами и в сложных приложениях могут иметь миллионы параметров. Несмотря на эти трудности, существует ряд трюков, которые можно использовать для того, чтобы не допустить возникновения проблемы переобучения. Выбор соответствующего метода определяется конкретными условиями задачи и типом нейронной сети. К числу основных методов, позволяющих избежать проблемы переобучения в нейронных сетях, относятся следующие.

1. **Регуляризация на основе штрафов (penalty-based regularization).** Это наиболее распространенная методика, применяемая в нейронных сетях для предотвращения переобучения. Суть регуляризации этого типа состоит в наложении штрафов или других типов ограничений на параметры, благоприятствующих созданию более простых моделей. Например, в случае полиномиальной регрессии одно из возможных ограничений на параметры может заключаться в том, чтобы ненулевые значения могли иметь не более чем k различных параметров w_i . Это будет гарантировать создание упрощенных моделей. В то же время, поскольку налагодить такие ограничения в явном виде нелегко, предпринимается другой подход, заключающийся во введении более гибкого штрафа наподобие $\lambda \sum_{i=0}^d w_i^2$ и его добавлении к функции потерь. В общих чертах данный подход сводится к умножению каждого параметра w_i на фактор затухания вида $(1 - \alpha\lambda)$, где α — скорость обучения, перед каждым обновлением. Помимо штрафования параметров сети можно использовать штрафование активаций скрытых элементов. Такой подход часто приводит к разреженным скрытым представлениям.

2. **Типовые и настраиваемые ансамблевые методы.** Многие ансамблевые методы не относятся к числу разработанных специально для нейронных сетей и могут применяться для других задач машинного обучения. Мы обсудим два простейших ансамблевых метода, которые могут быть реализованы практически для любой модели или задачи обучения: *бэггинг* (англ. “bagging” от “bootstrap aggregating” — улучшенное агрегирование) и *подвыборка* (subsampling). Эти методы унаследованы от традиционного машинного обучения. Существует несколько ансамблевых методов, предназначенных специально для нейронных сетей. Прямолинейный подход предполагает усреднение предсказаний различных нейронных архитектур, полученных путем быстрой и грубой оптимизации гиперпараметров. Еще одной ансамблевой техникой, предназначенной для нейронных сетей, является метод *исключения*, или *дропаут* (dropout). В этом подходе используется селективное отбрасывание узлов для создания других нейронных сетей. Окончательный результат формируется на основе сочетания предсказаний различных сетей. Такое прореживание узлов ослабляет эффекты переобучения, косвенно действуя в качестве регуляризатора.
3. **Ранняя остановка.** Метод *ранней остановки* предполагает преждевременное прекращение итеративной оптимизации, не дожидаясь достижения сходимости к оптимальному решению на тренировочном наборе данных. Точка остановки определяется с использованием отложенной части тренировочных данных, на которой модель не обучалась. Обучение останавливается тогда, когда ошибка на отложенных данных начинает увеличиваться. Несмотря на то что этот подход не является оптимальным для тренировочных данных, он, похоже, хорошо справляется с тестовыми данными, поскольку точка остановки определяется на основе отложенных данных.
4. **Предварительное обучение (pretraining).** Это форма обучения, использующая жадный алгоритм для нахождения хороших начальных значений параметров. Веса в различных слоях нейронной сети последовательно тренируются в жадной манере. Эти обученные веса используются в качестве неплохой начальной точки для общего процесса обучения. Можно показать, что предварительное обучение представляет собой косвенную форму регуляризации.
5. **Методы обучения с продолжением (continuation learning) и поэтапного обучения (curriculum learning).** Эффективность этих методов достигается за счет первоначальной тренировки простых моделей с последующим переходом к обучению все более сложных моделей. Такой подход основан на тех соображениях, что более простые модели легче поддаются обучению без риска переобучению. Кроме того, начало обучения с точки,

оптимальной для более простой модели, создает хорошие предпосылки для тесно связанной с ней более сложной модели. Следует отметить, что некоторые из этих методов можно считать аналогичными предварительному обучению. Предварительное обучение также находит решения путем перехода от простого к сложному за счет разложения процесса обучения нейронной сети на процессы обучения набора мелких слоев.

6. **Разделение параметров на основе знания специфики конкретной предметной области.** При обработке некоторых видов данных, таких как текст или изображения, часто можно выдвинуть некоторые соображения относительно структуры пространства параметров, исходя из знания особенностей этих данных. В подобных случаях для некоторых параметров, относящихся к различным частям сети, можно установить одно и то же общее значение. Тем самым достигается снижение количества степеней свободы модели. Подобный подход используется в рекуррентных нейронных сетях (обрабатывающих последовательные данные) и сверточных нейронных сетях (обрабатывающих изображения). Разделение параметров наталкивается на собственные трудности, поскольку это требует внесения соответствующих изменений в алгоритм обратного распространения ошибки.

Эта глава начинается с обсуждения обобщаемости моделей с привлечением некоторых теоретических результатов, относящихся к дилемме смещения — дисперсии. Далее рассматриваются способы уменьшения рисков переобучения.

Интересно отметить, что некоторые виды регуляризации в грубом приближении эквивалентны введению шума либо во входные данные, либо в скрытые переменные. Например, можно показать, что многие регуляризаторы на основе штрафов эквивалентны добавлению шума [44]. Кроме того, даже использование *стохастического* градиентного спуска вместо градиентного может рассматриваться как своеобразное добавление шума в шаги алгоритма. Как следствие, стохастический градиентный спуск демонстрирует превосходную точность на тестовых данных, несмотря на то что на тренировочных данных он может уступать градиентному спуску. Кроме того, некоторые виды ансамблевой техники, такие как дропаут и возмущение данных, эквивалентны внедрению шума. Сходство между внедрением шума и регуляризацией будет обсуждаться там, где это необходимо, на протяжении всей главы.

Несмотря на то что естественный способ предотвращения переобучения заключается в построении меньших сетей (имеющих меньшее количество параметров), лучше создавать более крупные сети, а затем регуляризовать их. Это объясняется тем, что большие сети сохраняют *возможность* построения более сложных моделей в тех случаях, когда это действительно требуется. В то же время процесс регуляризации может сглаживать случайные артефакты, не

поддерживаемые достаточным количеством данных. Используя такой подход, мы предоставляем модели возможность выбирать необходимую степень сложности, а не навязываем ей заранее фиксированный вариант (что может привести даже к недообучению).

Обучение с учителем более подвержено риску переобучения по сравнению с обучением без учителя, и именно на нем большей частью концентрируется внимание в литературе, посвященной обобщаемости моделей. Чтобы это понять, необходимо учесть, что приложения, реализующие обучение с учителем, пытаются обучить одну целевую переменную, но при этом могут иметь сотни входных (объясняющих) переменных. Риск переобучения в случае отчетливо сфокусированной цели велик по той причине, что результат каждого тренировочного примера жестко контролируется (например, с помощью бинарной метки). С другой стороны, в задачах неконтролируемого (без учителя) обучения количество целевых переменных совпадает с количеством объясняющих переменных. В конце концов, мы пытаемся моделировать полный набор данных на нем самом. В последнем случае переобучение менее вероятно (но по-прежнему возможно), поскольку в одном тренировочном примере содержится большое количество битов информации. Тем не менее регуляризация все еще используется в задачах обучения без учителя, особенно если имеются намерения навязать определенную структуру обучаемым представлениям.

Структура главы

В следующем разделе обсуждается дилемма смещения — дисперсии. Практические следствия дилеммы обсуждаются в разделе 4.3, использование регуляризации на основе штрафов — в разделе 4.4, а суть ансамблевых методов — в разделе 4.5. Одни методы, такие как бэггинг, являются универсальными, другие (как дропаут) предназначены специально для нейронных сетей. Методы ранней остановки обсуждаются в разделе 4.6, а методы предварительного обучения без учителя — в разделе 4.7. Методы обучения с расширением и поэтапного обучения представлены в разделе 4.8, а методы разделения параметров — в разделе 4.9. Формы регуляризации на основе обучения без учителя описаны в разделе 4.10. Резюме главы приведено в разделе 4.11.

4.2. Дилемма смещения — дисперсии

Во введении был приведен пример приближения небольшого набора данных полиномиальной моделью, которая делает для новых тестовых данных более ошибочные предсказания, чем (более простая) линейная модель. Это обусловлено тем, что полиномиальная модель требует предоставления данных в больших объемах, чтобы не быть введенной в заблуждение артефактами тренировочного набора данных. Тот факт, что более мощные модели не всегда дают

выигрыш в отношении точности предсказаний в условиях ограниченного набора данных, является основным следствием, вытекающим из дилеммы смещения — дисперсии.

В соответствии с дилеммой смещения — дисперсии квадрат ошибки алгоритма обучения можно разделить на три составляющие.

- 1. Смещение.** Это ошибка, обусловленная использованием упрощенных допущений в модели, что приводит к ошибкам соразмерной величины, независимо от выбора тренировочного набора данных. Даже если у модели имеется доступ к неограниченному источнику тренировочных данных, это не устраняет смещения. Например, для приведенной на рис. 4.2 линейной модели смещение больше, чем для полиномиальной, поскольку она никогда не впишется точно в (незначительно искривленное) распределение данных, независимо от количества доступных данных. Предсказание для некоторого отложенного тестового примера при $x = 2$ всегда будет иметь ошибку в определенном направлении, если использовать линейную модель при любом выборе тренировочной выборки. Если предположить, что прямая и искривленная линии, отображенные в левой верхней части на рис. 4.2, оценивались на бесконечном объеме данных, то разница между ними при любых конкретных значениях x и есть смещение. Пример смещения при $x = 2$ приведен на рис. 4.2.
- 2. Дисперсия.** Обусловлена невозможностью обучить параметры модели статистически надежным способом, особенно если данные ограничены и число параметров модели велико. Большие значения дисперсии проявляются в переобучении модели на определенном тренировочном наборе данных. Поэтому для одного и того же примера, выбранного из различных тренировочных наборов, будут получены различные предсказания. Обратите внимание на то, что для $x = 2$ на рис. 4.2 предсказания линейной модели во всех случаях одинаковы, тогда как предсказания полиномиальной модели меняются в широких пределах в зависимости от выбора тренировочных данных. Во многих случаях варьирующиеся в широких пределах предсказания при $x = 2$ значительно отличаются от корректного значения, что и есть проявлением дисперсии. Поэтому полиномиальный предиктор на рис. 4.2 характеризуется более высокой дисперсией по сравнению с линейным.
- 3. Шум.** Обусловлен внутренними ошибками в данных. Например, все точки данных на диаграмме рассеяния, отображенной в левой верхней части на рис. 4.2, отклоняются от истинной модели. Если бы не было шума, то все точки данных укладывались бы на криволинейную линию, представляющую истинную модель.

Приведенное описание отражает качественную сторону дилеммы смещения — дисперсии. Более формальное математическое описание представлено ниже.

4.2.1. Формальное рассмотрение

Обозначим через \mathcal{B} базовое распределение, из которого генерируется тренировочный набор данных. На основе этого базового распределения можно сгенерировать набор данных \mathcal{D} :

$$\mathcal{D} \sim \mathcal{B}. \quad (4.2)$$

Данные могут извлекаться разными способами, например путем выбора только наборов данных определенного размера. На текущем этапе предположим, что у нас имеется четко определенный генеративный процесс, в соответствии с которым тренировочные наборы данных извлекаются из \mathcal{B} . Приведенный ниже анализ не привязан к какому-то определенному механизму извлечения тренировочных наборов данных из \mathcal{B} .

Доступ к базовому распределению \mathcal{B} эквивалентен доступу к бесконечному ресурсу тренировочных данных, поскольку базовое распределение можно использовать для генерации тренировочных наборов данных неограниченное число раз. На практике такие базовые распределения (т.е. бесконечные источники данных) отсутствуют. Что касается практической стороны вопроса, то аналитик, используя некий источник данных, формирует только *один конечный экземпляр* \mathcal{D} . Однако сама концепция существования базового распределения, из которого могут генерироваться другие наборы тренировочных данных, полезна для количественной оценки источников ошибки при обучении на этом конечном наборе данных.

А теперь представьте, что аналитик располагает набором из t тестовых примеров в d измерениях, которые мы обозначим как $\bar{Z}_1 \dots \bar{Z}_t$. Зависимые переменные этих тестовых примеров обозначим как $y_1 \dots y_t$. Для определенности предположим, что тестовые переменные и их зависимые переменные были сгенерированы из одного и того же базового распределения \mathcal{B} третьей стороной, однако аналитик получил доступ только к представлениям признаков $\bar{Z}_1 \dots \bar{Z}_t$, но не к зависимым переменным $y_1 \dots y_t$. Поэтому перед аналитиком стоит задача использовать один конечный экземпляр тренировочного набора данных \mathcal{D} для предсказания зависимых переменных $\bar{Z}_1 \dots \bar{Z}_t$.

Предположим, что соотношение между зависимой переменной y_i и представлением ее признака \bar{Z}_i определяется *неизвестной* функцией $f(\cdot)$ следующим образом:

$$y_i = f(\bar{Z}_i) + \varepsilon_i, \quad (4.3)$$

где ε_i — внутренний шум, который не зависит от используемой модели. Значение ε_i может быть как положительным, так и отрицательным, хотя предполагается, что $E[\varepsilon_i] = 0$. Если бы аналитику было известно, какая функция $f(\cdot)$ соответствует этому соотношению, то он мог бы применить ее к каждой тестовой точке \bar{Z}_i для аппроксимации зависимой переменной y_i , и тогда единственным оставшимся источником неопределенности был бы внутренний шум.

Проблема в том, что на практике функция $f(\cdot)$ неизвестна аналитику. Эта функция используется в генеративном процессе базового распределения \mathcal{B} , а весь процесс генерирования данных подобен оракулу, который недоступен аналитику. Аналитик имеет лишь примеры входа и выхода функции. Ясно, что аналитику придется разработать некий тип модели $g(\bar{Z}_i, \mathcal{D})$, используя тренировочные данные для управления *аппроксимацией* этой функции на основе данных.

$$\hat{y}_i = g(\bar{Z}_i, \mathcal{D}). \quad (4.4)$$

Обратите внимание на символ циркумфлекса (^) над переменной \hat{y}_i , указывающий на то, что это *предсказанное* значение, полученное с помощью определенного алгоритма, а не наблюдаемое (истинное) значение y_i .

Примером оценочной функции $g(\cdot, \cdot)$ может служить любая из предсказывающих функций, используемых при обучении моделей (включая нейронные сети). Существуют даже алгоритмы (такие, как линейная регрессия и перцептроны), которые удается выразить в сжатой и понятной форме:

$$g(\bar{Z}_i, \mathcal{D}) = \underbrace{\bar{W} \cdot \bar{Z}_i}_{\text{Обучение } \bar{W} \text{ с помощью } \mathcal{D}} \quad [\text{линейная регрессия}],$$

$$g(\bar{Z}_i, \mathcal{D}) = \underbrace{\text{sign}\{\bar{W} \cdot \bar{Z}_i\}}_{\text{Обучение } \bar{W} \text{ с помощью } \mathcal{D}} \quad [\text{перцептрон}].$$

Большинство нейронных сетей алгоритмически выражается в виде композиций многих функций, вычисляемых в разных узлах. Выбор вычислительной функции включает эффект настройки ее специфического параметра, такого как векторный коэффициент \bar{W} в перцептроне. Для полного обучения функции в сети с большим количеством параметров требуется большее количество параметров. Именно здесь возникает дисперсия на одном и том же тестовом примере. Модель с большим набором параметров \bar{W} будет обучаться разным значениям этих параметров при использовании разных наборов тренировочных данных. Следовательно, предсказания для одного и того же тестового примера также будут весьма разными для различных тренировочных наборов данных. Эта несогласованность вносит свой вклад в ошибку (см. рис. 4.2).

Смысл разрешения дилеммы смещения — дисперсии заключается в квантификации ожидаемой ошибки алгоритма обучения в терминах ее смещения, дисперсии и (специфического для данных) шума. Для общности обсуждения предположим, что целевая переменная имеет числовую форму, поэтому интуиция нам подсказывает, что ошибку можно количественно выразить в виде *среднеквадратического отклонения* предсказываемых значений \hat{y}_i от наблюдаемых значений y_i . Эта форма квантификации ошибки естественна в регрессии, хотя она находит применение и в регрессии в терминах вероятностных предсказаний тестовых примеров. Среднеквадратическая ошибка, или MSE (mean squared error), алгоритма обучения $g(\cdot, \mathcal{D})$ определяется для набора тестовых примеров $\bar{Z}_1 \dots \bar{Z}_t$ следующим образом:

$$MSE = \frac{1}{t} \sum_{i=1}^t (\hat{y}_i - y_i)^2 = \frac{1}{t} \sum_{i=1}^t (g(\bar{Z}_1 \dots \bar{Z}_t) - f(\bar{Z}_i) - \varepsilon_i)^2.$$

Наилучшим способом оценки ошибки, не зависящим от конкретного выбора тренировочного набора данных, является вычисление *ожидаемой* ошибки при различных выборах тренировочного набора данных:

$$\begin{aligned} E[MSE] &= \frac{1}{t} \sum_{i=1}^t E[(g(\bar{Z}_i, \mathcal{D}) - f(\bar{Z}_i) - \varepsilon_i)^2] = \\ &= \frac{1}{t} \sum_{i=1}^t E[(g(\bar{Z}_i, \mathcal{D}) - f(\bar{Z}_i))^2] + \frac{\sum_{i=1}^t E[\varepsilon_i^2]}{t}. \end{aligned}$$

Второе соотношение получено путем разложения квадратичной части в правой части первого уравнения и последующего использования того факта, что среднее значение ε_i для большого набора количества тестовых примеров равно нулю.

Правая часть вышеприведенного выражения допускает дальнейшее разложение путем добавления и вычитания $E[g(\bar{Z}_i, \mathcal{D})]$ в квадратных скобках в правой части:

$$\begin{aligned} E[MSE] &= \frac{1}{t} \sum_{i=1}^t E[\{(f(\bar{Z}_i) - E[g(\bar{Z}_i, \mathcal{D})]) + \\ &+ (E[g(\bar{Z}_i, \mathcal{D})] - g(\bar{Z}_i, \mathcal{D}))\}^2] + \frac{\sum_{i=1}^t E[\varepsilon_i^2]}{t}. \end{aligned}$$

Разложив квадрат в правой части, получаем следующее выражение:

$$\begin{aligned}
 E[MSE] &= \frac{1}{t} \sum_{i=1}^t E[\{f(\bar{Z}_i) - E[g(\bar{Z}_i, \mathcal{D})]\}^2] + \\
 &+ \frac{2}{t} \sum_{i=1}^t \{f(\bar{Z}_i) - E[g(\bar{Z}_i, \mathcal{D})]\} \{E[g(\bar{Z}_i, \mathcal{D})] - E[g(\bar{Z}_i, \mathcal{D})]\} + \\
 &+ \frac{1}{t} \sum_{i=1}^t E[\{E[g(\bar{Z}_i, \mathcal{D})] - g(\bar{Z}_i, \mathcal{D})\}^2] + \frac{\sum_{i=1}^t E[\varepsilon_i^2]}{t}.
 \end{aligned}$$

Второй член в правой части вышеприведенного выражения равен нулю, поскольку он содержит множители вида $E[g(\bar{Z}_i, \mathcal{D})] - E[g(\bar{Z}_i, \mathcal{D})]$. После упрощения получаем следующее соотношение:

$$\begin{aligned}
 E[MSE] &= \underbrace{\frac{1}{t} \sum_{i=1}^t \{f(\bar{Z}_i) - E[g(\bar{Z}_i, \mathcal{D})]\}^2}_{\text{Смещение}^2} + \underbrace{\frac{1}{t} \sum_{i=1}^t E[\{g(\bar{Z}_i, \mathcal{D}) - E[g(\bar{Z}_i, \mathcal{D})]\}^2]}_{\text{Дисперсия}} + \\
 &+ \underbrace{\frac{\sum_{i=1}^t E[\varepsilon_i^2]}{t}}_{\text{Шум}}.
 \end{aligned}$$

Иными словами, квадрат ошибки можно разложить на (возведенное в квадрат) смещение, дисперсию и шум. Основным членом, препятствующим обобщению сети, является дисперсия. В общем случае дисперсия будет больше в нейронных сетях с большим количеством параметров. С другой стороны, слишком малое количество параметров вызывает смещение ввиду отсутствия достаточного числа степеней свободы для моделирования всех сложностей распределения данных. Этот баланс между смещением и дисперсией показан на рис. 4.3. Очевидно, что существует точка оптимальной сложности модели, в которой производительность оптимальна. Кроме того, малочисленность тренировочных данных приводит к увеличению дисперсии. Однако тщательное проектирование модели способно уменьшить эффекты переобучения. Некоторые из этих возможностей обсуждаются далее.

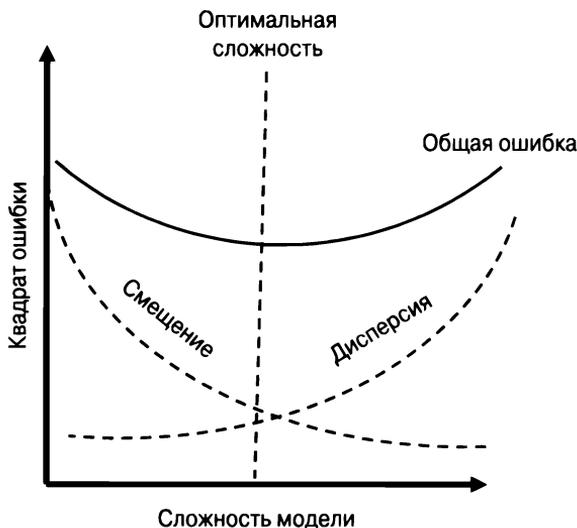


Рис. 4.3. Обычно достижением компромисса между смещением и дисперсией удается найти точку оптимальной сложности модели

4.3. Проблемы обобщаемости модели при ее настройке и оценке

При тренировке моделей нейронной сети возникает ряд вопросов практического характера, связанных с дилеммой смещения — дисперсии. Первый из них касается настройки и выбора гиперпараметров. Например, если настроить нейронную сеть на тех же данных, на которых она обучалась, то получение хороших результатов маловероятно ввиду эффектов переобучения. Поэтому гиперпараметры (т.е. параметры регуляризации) настраивают на отдельном отложенном наборе, отличающемся от того, на котором обучались параметры весов нейронной сети.

Располагая определенным набором размеченных данных, мы должны использовать этот ресурс для тренировки, настройки и тестирования точности модели. Очевидно, что использовать полностью весь набор размеченных данных для создания модели (т.е. для обучения ее весовых параметров) нельзя. Например, использование одного и того же набора данных для создания модели и тестирования приводит к значительной переоценке точности. Так происходит потому, что основной целью классификации является *обобщение* модели размеченных данных на новые тестовые примеры. Кроме того, часть набора данных, используемая для *отбора модели и настройки параметров*, также должна отличаться от данных, которые были использованы для создания модели.

Распространенной ошибкой является использование одного и того же набора данных как для настройки параметров, так и для окончательной оценки модели (тестирования). При таком подходе тренировочные и тестовые данные частично перемешиваются, и результирующая точность получается чрезмерно оптимистичной. Имеющийся набор данных всегда необходимо разделять на три части, определяемые в соответствии с их предназначением.

1. *Тренировочные данные.* Эта часть данных используется для построения тренировочной модели (создаваемой в процессе обучения весов нейронной сети). Здесь возможно несколько проектных решений. Нейронная сеть может использовать различные гиперпараметры для скорости обучения или регуляризации. Один и тот же тренировочный набор может испытываться множество раз при различном выборе гиперпараметров или с совершенно разными алгоритмами для создания модели множеством способов. Это позволяет оценивать относительную точность различных алгоритмических настроек. Такой подход включает стадию *отбора модели*, в процессе которой из различных моделей выбирается та, алгоритм которой оказался наилучшим. Однако фактическая *оценка* этих алгоритмов для отбора наилучшей модели выполняется не на тренировочных данных, а на отдельном валидационном наборе, чтобы переобученные модели не получали преимущество.
2. *Валидационный набор.* Эта часть данных используется для отбора модели и настройки параметров. Например, можно настроить скорость обучения путем создания нескольких моделей с использованием первой части набора данных (т.е. тренировочных данных), а затем использовать валидационный набор для оценки точности различных моделей. Как обсуждалось в разделе 3.3.1, для этого значения параметров меняют в определенном диапазоне, и различные комбинации этих значений используются для тестирования точности на валидационном наборе. Наилучшей считается комбинация, обеспечивающая достижение наибольшей точности. В определенном смысле валидационные данные можно рассматривать как своего рода тестовый набор данных, используемый для настройки параметров алгоритма (т.е. скорости обучения, количества слоев или элементов в каждом слое) или для выбора наилучшего проектного решения (например, активации в виде сигмоиды или гиперболического тангенса).
3. *Тестовые данные.* Эта часть данных используется для тестирования точности окончательной (настроенной) модели. Во избежание переобучения важно, чтобы тестовые данные вообще никак не использовались в процессе настройки параметров и отбора модели. *Тестовые данные задействуются лишь однажды в самом конце процесса.* Кроме того, если

аналитик использует результаты, полученные на тестовых данных, для той или иной настройки модели, то результаты будут искажены знаниями, приобретенными на этих данных. Идея, суть которой заключается в том, что к тестовым данным можно обратиться только один раз, выдвигается как чрезвычайно строгое (и очень важное) требование. И все же в реальных контрольных испытаниях это требование часто нарушается. А объясняется это всего-навсего тем, что соблазн использовать знания, приобретенные в процессе окончательной оценки точности, оказывается слишком большим.

Разделение помеченного набора данных на тренировочные, валидационные и тестовые данные показано на рис. 4.4. Строго говоря, валидационные данные также являются частью тренировочных данных, поскольку они влияют на окончательную модель (хотя к тренировочным обычно относят только ту часть данных, которая используется для построения модели). Еще с 1990-х годов общепринято делить исходный набор на указанные поднаборы в пропорции 2:1:1. Однако это правило не следует воспринимать как строгое. В случае очень больших наборов помеченных данных для оценки точности потребуется лишь умеренное количество примеров. При наличии большого набора данных имеет смысл использовать для построения модели как можно большую его часть, поскольку дисперсия, возникающая на стадиях валидации и оценки, часто довольно низкая. Использование постоянного количества примеров (например, менее нескольких тысяч) в валидационном и тестовом наборах данных достаточно для получения точных оценок. Поэтому деление в пропорции 2:1:1 — это мнемоническое правило, унаследованное от прошлых времен, когда наборы данных были небольшими. В наши дни, когда мы располагаем доступом к большим объемам данных, для тренировки используются почти все точки, а для тестирования — умеренное (постоянное) их количество. Поэтому нередко деление исходного набора осуществляется в пропорциях порядка 98:1:1.

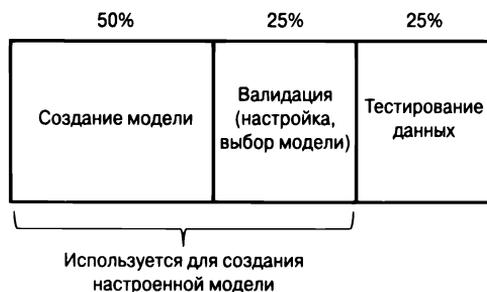


Рис. 4.4. Распределение набора размеченных данных для оценки

4.3.1. Оценка точности с помощью отложенного набора данных и перекрестной проверки

Приведенное выше разделение помеченных данных на три сегмента представляет собой неявное описание метода сегментирования помеченных данных под названием *отложенный набор*. Однако разделение данных на *три* части не выполняется за один раз. Вместо этого сначала тренировочные данные разделяются на *две* части, предназначенные для целей тренировки и тестирования. После этого часть, предназначенная для тестирования, тщательно скрывается от любого дальнейшего анализа *до самого конца, когда ее можно использовать только однажды*. Оставшаяся часть набора данных вновь подразделяется на тренировочную и валидационную. Этот тип рекурсивного деления представлен на рис. 4.5.

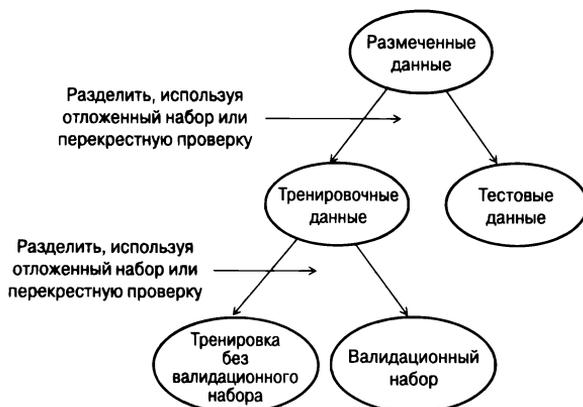


Рис. 4.5. Иерархическое деление данных на тренировочный, валидационный и тестовый наборы

Важно то, что типы деления данных на обоих уровнях иерархии концептуально идентичны. Далее мы будем последовательно придерживаться терминологии первого уровня и говорить о разделении данных на «тренировочный» и «тестовый» наборы, даже если тот же подход используется для второго уровня деления данных на части, ответственные за построение модели и ее валидацию. Это позволяет дать общее описание процессов оценки на обоих уровнях деления данных.

Метод отложенного набора

В методе отложенного набора часть примеров используется для построения тренировочной модели. Оставшиеся примеры, которые также называют *отложенными*, используются для тестирования. Затем точность предсказания меток удержанных примеров объявляется в качестве общей точности. Такой подход

гарантирует, что объявленная точность не является результатом переобучения на каком-то конкретном наборе данных, поскольку для тренировки и тестирования используются различные примеры. Однако этот подход недооценивает истинную точность. Рассмотрим случай, когда доля определенного класса в отложенных примерах выше, чем в размеченном наборе данных. Это означает, что средняя доля этого же класса в *оставленных* примерах всегда будет ниже, что приведет к несогласованности тренировочных и тестовых данных. Кроме того, это приведет к систематическому пессимистическому смещению оценки точности. Несмотря на эти недостатки, у метода отложенного набора имеются и свои преимущества — простота и эффективность, что сделало его популярным в крупномасштабных задачах. Для глубокого обучения, относящегося именно к задачам такого рода, это имеет большое значение.

Перекрестная проверка

В методе перекрестной проверки помеченные данные разделяются на q равных сегментов. Один из этих q сегментов применяется для тестирования, тогда как остальные $(q - 1)$ сегментов — для тренировки. Этот процесс повторяется q раз с использованием каждого из q сегментов в качестве тестового набора. В качестве точности объявляется средняя точность по q различным тестовым наборам. Отметим, что можно обеспечить оценку, близкую к истинной точности, если q велико. Особым является случай, когда q выбирается равным количеству помеченных точек данных, и поэтому для тестирования используется одна точка. Поскольку из обучающего набора исключается только одна точка, этот подход называют *перекрестной проверкой по отдельным точкам* (leave-one-out cross-validation). Несмотря на то что такой подход способен обеспечить очень близкую оценку точности, многократное обучение модели обычно оказывается слишком дорогостоящим. На практике перекрестная проверка редко используется в нейронных сетях из-за проблем с производительностью.

4.3.2. Проблемы с крупномасштабной тренировкой

Одной из проблем, с которыми приходится сталкиваться в конкретном случае нейронных сетей, являются большие размеры наборов тренировочных данных. Поэтому, в то время как в традиционном машинном обучении методы наподобие перекрестной проверки обучения твердо держат позиции наилучшего варианта выбора, их техническим превосходством часто жертвуют ради производительности. В общем случае фактор длительности тренировки играет настолько важную роль в моделировании нейронных сетей, что для обеспечения практической реализуемости моделей приходится идти на многие компромиссы.

Проблемы вычислительного характера часто возникают в контексте сеточного поиска гиперпараметров (см. раздел 3.3.1). Иногда на выбор даже одного

гиперпараметра уходит несколько дней, а сеточный поиск требует тестирования большого количества различных возможностей. Поэтому общая стратегия заключается в выполнении тренировочного процесса для каждой настройки в течение фиксированного количества эпох. Процесс тренировки многократно выполняется при различных вариантах выбора гиперпараметров в разных потоках выполнения. Выполнение вариантов, не обеспечивающих достижения удовлетворительного прогресса за фиксированное количество эпох, прерывается. В конце только нескольким вариантам ансамбля предоставляется возможность выполнения до полного завершения. Одной из причин того, почему такой подход хорошо работает, является то, что преобладающая доля прогресса часто обеспечивается ранними фазами тренировки. Этот процесс также описан в разделе 3.3.1.

4.3.3. Как определить необходимость в сборе дополнительных данных

Большая ошибка обобщения в нейронной сети может возникать по целому ряду причин. Во-первых, данные уже сами по себе могут содержать много шума, и в этом случае мало что можно сделать для улучшения точности. Во-вторых, нейронные сети нелегко поддаются обучению, и большая ошибка может быть обусловлена плохой сходимостью алгоритма. Ошибка также может вызываться большим смещением, что называют *недообучением* (underfitting). Наконец, значительная часть ошибки обобщения может быть обусловлена *переобучением* (overfitting), т.е. высокой дисперсией. В большинстве случаев ошибка возникает в результате сочетания нескольких факторов. В то же время переобучение для конкретного набора тренировочных данных можно обнаружить по разнице в точности тренировки и тестирования. Переобучение проявляется в виде *большого расхождения между этими величинами*. Нередко точность тренировки на небольшом наборе данных достигает 100%, даже если ошибка тестирования довольно невелика. Первое, что необходимо сделать для устранения этой проблемы, — собрать больше данных. С увеличением объема тренировочных данных точность тренировки будет уменьшаться, тогда как точность тестирования/валидации — увеличиваться. Однако в случае отсутствия дополнительных данных необходимо использовать другие способы улучшения обобщаемости модели, такие как регуляризация.

4.4. Регуляризация на основе штрафов

Регуляризация на основе штрафов — наиболее распространенный подход к снижению переобучения. Чтобы понять, почему это так, вновь обратимся к

примеру полиномиальной модели степени d . В этом случае предсказание \hat{y} для заданного значения x вычисляется по следующей формуле:

$$\hat{y} = \sum_{i=0}^d w_i x^i. \quad (4.5)$$

Для моделирования данного предсказания можно использовать однослойную нейронную сеть с d входами и одним нейроном смещения с весом w_0 . Значением входа i является x^i . В этой нейронной сети используются линейные активации, и квадратичную функцию потерь для набора тренировочных примеров (x, y) из набора \mathcal{D} можно определить следующим образом:

$$L = \sum_{(x, y) \in \mathcal{D}} (y - \hat{y})^2.$$

Как уже отмечалось при обсуждении примера, приведенного на рис. 4.2, большие значения d усиливают эффект переобучения. Одним из возможных решений проблемы является уменьшение d . Иными словами, *экономия параметров* приводит к более простой модели. Так, уменьшение значения d до 1 создает линейную модель, которая имеет меньше степеней свободы и обеспечивает одинаковое согласование с различными тренировочными образцами. Однако в случае действительно сложных шаблонов данных некоторая часть выразительности при этом теряется. Иными словами, чрезмерное упрощение модели приводит к уменьшению выразительной способности нейронной сети, в результате чего она теряет возможность достаточно хорошо подстраиваться под различные типы наборов данных.

Каким же образом можно сохранить выразительность сети и при этом предохранить ее от чрезмерного переобучения? Вместо того чтобы просто уменьшить количество параметров, можно ввести *мягкую* систему штрафов за их использование. Кроме того, большие (по абсолютной величине) значения параметров штрафуются более строго, чем небольшие, поскольку последние не оказывают на предсказания существенного влияния. Какой же тип штрафов следует использовать? Наиболее распространенный выбор — L_2 -регуляризация, известная под названием *регуляризация Тихонова*. В этом случае величина дополнительного штрафа определяется суммой квадратов значений параметров. Тогда можно использовать целевую функцию с параметром регуляризации $\lambda > 0$ следующего вида:

$$L = \sum_{(x, y) \in \mathcal{D}} (y - \hat{y})^2 + \lambda \cdot \sum_{i=0}^d w_i^2.$$

Увеличивая или уменьшая значение λ , мы можем регулировать величину штрафа. Одним из преимуществ такого параметризованного типа штрафования

является возможность настройки этого параметра для оптимизации производительности на той части тренировочного набора данных, которая не используется для обучения параметров. Такой подход называют *валидацией модели*. Он обеспечивает гораздо большую гибкость по сравнению с заблаговременной фиксацией экономичности модели. Рассмотрим случай полиномиальной регрессии, которая обсуждалась перед этим. Предварительное ограничение количества параметров жестко определяет форму обучаемого многочлена (например, сводя ее к линейной модели), тогда как гибкое штрафование позволяет управлять его формой в соответствии с данными. В целом, как показывают экспериментальные наблюдения, лучше использовать сложные модели (например, более крупные нейронные сети) с регуляризацией, чем простые модели без регуляризации. Кроме того, первый вариант обеспечивает большую гибкость за счет предоставления дополнительных элементов настройки (например, параметра регуляризации), значения которых выбираются под управлением данными. Такие элементы настройки обучаются на отложенной части данных тренировочного набора.

Каково влияние регуляризации на обновления нейронной сети? Для любого веса w_i связи в нейронной сети обновления определяются с помощью метода градиентного спуска (или его пакетной версии):

$$w_i \leftarrow w_i - \alpha \frac{\partial L}{\partial w_i},$$

где α — скорость обучения. Использование L_2 -регуляризации в общих чертах эквивалентно наложению затухания после каждого обновления параметра:

$$w_i \leftarrow w_i(1 - \alpha\lambda) - \alpha \frac{\partial L}{\partial w_i}.$$

Обратите внимание на то, что в приведенной выше формуле вес сначала умножается на фактор затухания $(1 - \alpha\lambda)$, а затем используется для обновления на основе градиента. Затуханию весов можно дать биологическую интерпретацию, если предположить, что начальные значения весов близки к нулю. Затухание весов можно рассматривать в качестве своего рода механизма забывания, смещающего веса ближе к их начальным значениям. Это гарантирует, что только повторные обновления могут оказывать существенное влияние на абсолютные величины весов. Механизм забывания не дает модели *запоминать* тренировочные данные, благодаря чему лишь наиболее значимые и повторные обновления будут отражаться на весах.

4.4.1. Связь регуляризации с внедрением шума

Добавление шума на входе можно связать с регуляризацией на основе штрафов. Можно показать, что добавление равных порций гауссовского шума на каждом входе эквивалентно регуляризации Тихонова в однослойной сети с тождественной функцией активации (для линейной регрессии).

В частности, один из способов продемонстрировать, что это действительно так, заключается в том, чтобы исследовать одиночный тренировочный пример (\bar{X}, y) , который после добавления шума с дисперсией λ к каждому признаку превращается в $(\bar{X} + \sqrt{\lambda}\bar{\varepsilon}, y)$. Здесь $\bar{\varepsilon}$ — случайный вектор, каждый элемент $\bar{\varepsilon}_i$ которого независимо извлекается из стандартного нормального распределения с нулевым средним и единичной дисперсией. Зашумленное предсказание \hat{y} на основании примера $\bar{X} + \sqrt{\lambda}\bar{\varepsilon}$ определяется следующей формулой:

$$\hat{y} = \bar{W} \cdot (\bar{X} + \sqrt{\lambda}\bar{\varepsilon}) = \bar{W} \cdot \bar{X} + \sqrt{\lambda}\bar{W} \cdot \bar{\varepsilon}. \quad (4.6)$$

Теперь исследуем вклад в квадратичную функцию потерь $L = (y - \hat{y})^2$, обусловленный одиночным примером. Мы вычислим *ожидаемое* значение функции потерь. Нетрудно показать, что для него справедливо следующее выражение:

$$\begin{aligned} E[L] &= E[(y - \hat{y})^2] = \\ &= E[(y - \bar{W} \cdot \bar{X} - \sqrt{\lambda}\bar{W} \cdot \bar{\varepsilon})^2]. \end{aligned}$$

Выражение в правой части можно разложить и привести к следующему виду:

$$\begin{aligned} E[L] &= (y - \bar{W} \cdot \bar{X})^2 - 2\sqrt{\lambda}(y - \bar{W} \cdot \bar{X}) \underbrace{E[\bar{W} \cdot \bar{\varepsilon}]}_0 + \lambda E[(\bar{W} \cdot \bar{\varepsilon})^2] = \\ &= (y - \bar{W} \cdot \bar{X})^2 + \lambda E[(\bar{W} \cdot \bar{\varepsilon})^2]. \end{aligned}$$

Второе выражение можно разложить, используя соотношения $\bar{\varepsilon} = (\varepsilon_1 \dots \varepsilon_d)$ и $\bar{W} = (w_1 \dots w_d)$. Кроме того, учитывая независимость случайных переменных ε_i и ε_j , можем представить любой член $E[\varepsilon_i \varepsilon_j]$ в виде $E[\varepsilon_i] \cdot E[\varepsilon_j] = 0$. Любой член вида $E[\varepsilon_i^2]$ равен 1, поскольку каждый элемент ε_i извлекается из стандартного нормального распределения. После разложения члена $E[(\bar{W} \cdot \bar{\varepsilon})^2]$ и выполнения приведенных выше подстановок получаем следующее соотношение:

$$E[L] = (y - \bar{W} \cdot \bar{X})^2 + \lambda \left(\sum_{i=1}^d w_i^2 \right). \quad (4.7)$$

Следует отметить, что *эта функция потерь приводит в точности к тому же результату, что и L_2 -регуляризация* одиночного примера.

Несмотря на то что эквивалентность между затуханием весов и добавлением шума строго соблюдается для линейной регрессии, в случае нейронных сетей с нелинейными активациями это не так. Тем не менее регуляризация на основе штрафов и в этих случаях продолжает напоминать добавление шума, хотя результаты могут качественно различаться. Ввиду этого сходства иногда пытаются выполнить регуляризацию путем непосредственного добавления шума. Одним из таких подходов является метод *возмущения данных* (data perturbation), в котором шум добавляется во входные тренировочные данные и точки данных предсказываются с добавленным шумом. Этот подход многократно повторяется с различными тренировочными наборами данных, создаваемыми повторным добавлением шума в духе метода Монте-Карло. Предсказания для одного и того же тестового примера усредняются по различным добавкам шума для получения улучшенных результатов. В этом случае шум добавляется только в тренировочные данные и не должен добавляться в тестовые. В случае явного добавления шума важно усреднять предсказание для одного и того же тестового примера по множеству компонент ансамбля для гарантии того, что решение адекватно представляет ожидаемое значение функции потерь (без добавленной дисперсии, обусловленной шумом). Этот подход описан в разделе 4.5.5.

4.4.2. L_1 -регуляризация

Использование штрафов на основе квадрата нормы (так называемая L_2 -регуляризация) — наиболее распространенный способ регуляризации. Однако возможны и другие способы штрафования параметров. Обычным подходом является L_1 -регуляризация, в которой квадратичный штраф заменяется штрафом на основе суммы абсолютных величин коэффициентов. Поэтому новая целевая функция принимает следующий вид:

$$L = \sum_{(x, y) \in \mathcal{D}} (y - \hat{y})^2 + \lambda \cdot \sum_{i=0}^d |w_i|.$$

Основная проблема этой функции обусловлена тем, что она содержит члены $|w_i|$, которые не имеют производной при нулевом значении. Это требует внесения определенных изменений в метод градиентного спуска, если w_i равно нулю. В случае ненулевых w_i можно непосредственно использовать обновление, вычисляемое с помощью частной производной. Дифференцируя приведенную выше целевую функцию, мы можем определить уравнение обновления по крайней мере для случаев, когда w_i отличны от нуля:

$$w_i \leftarrow w_i - \alpha \lambda s_i - \alpha \frac{\partial L}{\partial w_i}.$$

Величина s_i , являющаяся частной производной $|w_i|$ (по w_i), имеет следующие значения:

$$s_i = \begin{cases} -1, & \text{если } w_i < 0, \\ +1, & \text{если } w_i > 0. \end{cases}$$

Но мы должны установить значения частной производной $|w_i|$ также для случаев *точного* равенства w_i нулю. Одна из возможностей заключается в использовании *метода субградиента*, в котором w_i получает стохастические значения из набора $\{-1, +1\}$. Однако на практике этого не приходится делать. Компьютеры работают с конечной точностью, и вследствие вычислительных ошибок случаи строгого равенства w_i нулю будут крайне редкими. Поэтому ошибки вычислений фактически обеспечивают решение той задачи, для которой предназначено стохастическое семплирование. Кроме того, в тех редких случаях, когда значение w_i в точности равно нулю, можно опустить регуляризацию и просто положить производную s_i равной нулю. Этот тип приближения работает достаточно хорошо во многих ситуациях.

Одно из различий между уравнениями для обновлений при L_1 - и L_2 -регуляризации заключается в том, что в случае L_2 -регуляризации в качестве механизма затухания используется мультипликативное затухание, тогда как в случае L_1 -регуляризации в качестве механизма забывания используются аддитивные обновления. В обоих случаях регуляризационная часть обновлений смещает значения коэффициентов ближе к нулю. В то же время между типами решений, получаемых в обоих случаях, существуют определенные различия, которые рассматриваются в следующем разделе.

4.4.3. Что лучше: L_1 - или L_2 -регуляризация?

У вас может возникнуть вполне естественный вопрос относительно того, какую регуляризацию следует использовать: L_1 или L_2 . С точки зрения точности L_2 -регуляризация обычно превосходит регуляризацию L_1 . Именно по этой причине в большинстве реализаций предпочтение почти всегда отдается L_2 -регуляризации. При больших количествах входов и элементов различия в производительности невелики.

Однако с точки зрения интерпретируемости результатов у L_1 -регуляризации имеется своя специфическая область применения. L_1 -регуляризация обладает тем интересным свойством, что она создает *разреженные* решения, в которых подавляющее большинство значений w_i равно нулю (если игнорировать¹

¹ Вычислительные ошибки можно игнорировать, потребовав, чтобы ненулевыми считались w_i , для которых $|w_i|$ превышает по крайней мере значение 10^{-6} .

вычислительные ошибки). Если значение w_i равно нулю для входящего соединения на входе, то данный вход не оказывает никакого влияния на предсказание. Иными словами, такой вход можно отбросить, и тогда L_1 -регуляризатор действует в качестве селектора признаков. Поэтому L_1 -регуляризацию можно использовать для оценки того, какие признаки должны включаться в число предсказываемых для конкретного приложения.

А что можно сказать о связях скрытых слоев, веса которых устанавливаются в нуль? Эти соединения можно отбрасывать, что приводит к разреженной сети. Разреженные сети могут быть полезными в тех случаях, когда для обучения используется один и тот же тип набора данных, причем природа и более широкие характеристики этого набора не меняются существенным образом во времени. Поскольку разреженная нейронная сеть будет содержать только небольшую долю соединений исходной нейронной сети, она может более эффективно повторно обучаться при получении дополнительных тренировочных данных.

4.4.4. Штрафование скрытых элементов: обучение разреженным представлениям

Методы на основе штрафов, которые обсуждались до сих пор, штрафуют параметры нейронной сети. Другой возможный подход — штрафование *активаций* нейронной сети, чтобы для любого заданного примера данных активировалось лишь небольшое подмножество нейронов. Иными словами, даже в случае крупной и сложной нейронной сети лишь небольшая ее часть будет использоваться в процессе создания предсказания для любого заданного примера данных.

Простейшим способом достижения разреженности является наложение L_1 -штрафов на скрытые элементы. Поэтому исходная функция потерь L преобразуется в регуляризованную функцию потерь L' :

$$L' = L + \lambda \sum_{i=1}^M |h_i|, \quad (4.8)$$

где M — общее количество элементов в сети; h_i — значение i -го скрытого элемента; λ — параметр регуляризации. Во многих случаях регуляризуется один *слой* сети, поэтому разреженное представление признаков может быть извлечено из активаций данного слоя.

Как изменится алгоритм обратного распространения ошибки в результате указанного изменения целевой функции? Основное отличие состоит в том, что функция потерь агрегируется по узлам не только выходного, но и скрытого слоя. На фундаментальном уровне это изменение никак не влияет на общую динамику и принципы обратного распространения ошибки (см. раздел 3.2.7).

Алгоритм обратного распространения необходимо видоизменить таким образом, чтобы штрафной вклад регуляризации скрытого элемента включался в обратный поток градиента всех входящих соединений данного узла. Обозначим через $N(h)$ множество узлов, достижимых из любого конкретного узла h вычислительного графа (включая сам узел). Градиент $\frac{\partial L}{\partial a_h}$ функции потерь L также зависит от штрафных вкладов узлов из множества $N(h)$. В частности, для любого узла h_r с предактивационным значением a_{h_r} его поток градиента $\frac{\partial L}{\partial a_{h_r}} = \delta(h_r, N(h_r))$ в выходной узел увеличивается на $\lambda \Phi'(a_{h_r}) \cdot \text{sign}(h_r)$. Здесь поток градиента $\frac{\partial L}{\partial a_{h_r}} = \delta(h_r, N(h_r))$ определяется в соответствии с обсуждением, приведенным в разделе 3.2.7. Рассмотрим уравнение 3.25, в соответствии с которым вычисляется обратный поток градиента:

$$\delta(h_r, N(h_r)) = \Phi'(a_{h_r}) \sum_{h: h_r \Rightarrow h} h w_{(h_r, h)} \delta(h, N(h)). \quad (4.9)$$

Здесь $w_{(h_r, h)}$ — вес ребра, ведущего от h_r к h . Сразу же после выполнения этого обновления значение $\delta(h_r, N(h_r))$ подстраивается для учета регуляризационного члена в данном узле следующим образом:

$$\delta(h_r, N(h_r)) \leftarrow \delta(h_r, N(h_r)) + \lambda \Phi'(a_{h_r}) \cdot \text{sign}(h_r).$$

Обратите внимание на то, что приведенное выше обновление базируется на уравнении 3.26. Сразу же после изменения значения $\delta(h_r, N(h_r))$ в данном узле h_r , изменения автоматически распространяются в обратном направлении ко всем узлам, которые достигают h_r . Это единственное изменение, которого требует введение L_1 -регуляризации скрытых элементов. В некотором смысле включение штрафов на узлах промежуточных слоев не вносит никаких кардинальных изменений в алгоритм обратного распространения, за исключением того, что теперь скрытые узлы трактуются так же, как и выходные узлы в плане вклада в поток градиента.

4.5. Ансамблевые методы

Ансамблевые методы были введены с целью достижения компромисса между смещением и дисперсией. Одна из возможностей уменьшить ошибку классификации заключается в уменьшении смещения или дисперсии таким способом, чтобы это не повлияло на другие компоненты. Ансамблевые методы широко применяются в машинном обучении. В качестве примера можно привести методы *бэггинга* (bagging) и *бустинга* (boosting). Первый из них позволяет уменьшить дисперсию, второй — смещение.

Целью использования большинства ансамблевых методов в нейронных сетях является уменьшение дисперсии. Это объясняется тем, что ценность нейронных сетей заключается в возможности строить на их основе сколь угодно сложные модели, характеризующиеся относительно низким смещением. Однако в силу дилеммы смещения — дисперсии это почти всегда приводит к росту дисперсии, что проявляется в виде переобучения. Поэтому цель большинства ансамблевых методов в нейронных сетях — уменьшение дисперсии (улучшающее обобщаемость модели). Рассмотрим эти методы более подробно.

4.5.1. Бэггинг и подвыборка

Представьте, что вы располагаете неограниченным источником тренировочных данных, с помощью которого можете генерировать сколь угодно много тренировочных точек из базового распределения. Как использовать этот необычайно щедрый ресурс для того, чтобы избавиться от дисперсии? При наличии достаточно большого количества выборок дисперсия большинства типов статистических оценок в конце концов может быть асимптотически сведена к нулю.

В этом случае естественным подходом к снижению дисперсии было бы повторное создание различных наборов тренировочных данных и предсказание результата для одного и того же примера с использованием различных наборов. После этого окончательное предсказание можно было бы получить усреднением предсказаний, сделанных для различных наборов. Если использовать достаточно количество наборов тренировочных данных, то дисперсия предсказания будет уменьшена до нуля, хотя зависимость смещения от модели по-прежнему останется.

Описанный подход может быть использован только при наличии неограниченного источника данных. Тем не менее на практике мы располагаем лишь одним экземпляром доступных нам данных. Очевидно, что в подобных случаях описанная методология не может быть реализована. Однако оказывается, что даже несовершенная имитация этой методологии все еще будет иметь лучшие дисперсионные характеристики, чем в случае однократного выполнения модели на полном тренировочном наборе данных. Базовая идея состоит в том, чтобы генерировать новые наборы тренировочных данных из одной и той же базовой коллекции данных путем извлечения выборок. Выборки могут извлекаться с замещением или без замещения данных. Затем предсказания для отдельного тестового примера, полученные с помощью моделей, построенных с использованием различных тренировочных наборов, усредняются для получения окончательного предсказания. Можно усреднять либо предсказания в виде вещественных чисел (например, вероятностные оценки меток классов), либо дискретные предсказания. В случае вещественных предсказаний лучшие результаты иногда удается получить, используя медиану значений.

Для получения вероятностных предсказаний дискретных выходов общепринято использовать функцию *Softmax*. Если усредняются вероятностные предсказания, то обычно усредняют *логарифмы* этих значений. Это эквивалентно использованию *геометрических* средних вероятностей. В случае дискретных предсказаний используется *арифметически усредненное голосование*. Указанное различие в обработке дискретных и вероятностных предсказаний переносится на другие типы ансамблевых методов, которые требуют усреднения предсказаний. Это объясняется тем, что логарифмы вероятностей допускают интерпретацию в терминах логарифмического правдоподобия, которому присуща аддитивность.

Основное различие между методами бэггинга и подвыборок определяется тем, используется ли замещение при создании выборочных наборов тренировочных данных. Ниже приведено краткое описание этих методов.

1. *Метод бэггинга (bagging)*. В этом методе предполагается извлечение выборок тренировочных данных из полного тренировочного набора с замещением. Размер s выборки устанавливается в зависимости от размера n полного тренировочного набора, хотя нередко s принимается равным n . Если $s = n$, то некоторые данные могут быть повторены в каждой выборке, причем доля исходных данных, которые вообще не войдут в выборки, при больших n асимптотически составит $(1 - 1/n)^n \approx 1/e$, где e — основание натуральных логарифмов. На основе выборки тренировочных данных строится модель, с помощью которой получают предсказания для каждого тестового примера. Весь процесс ресемплинга и построения модели повторяется m раз. Каждая из этих m моделей применяется к тестовым данным заданного тестового примера. После этого предсказания различных моделей усредняются для получения окончательного предсказания. Несмотря на то что в методе бэггинга принято использовать размер выборки, равный размеру исходного тренировочного набора ($s = n$), наилучшие результаты часто удается получить, устанавливая s на много меньшим, чем n .
2. *Метод подвыборок (subsampling)*. Этот метод аналогичен бэггингу, за исключением того, что модели строятся на выборках тренировочных данных, извлекаемых без замещения. Предсказания различных моделей усредняются. В данном случае важно выбирать размер выборок таким, чтобы выполнялось неравенство $s < n$, поскольку при $s = n$ для различных членов ансамбля будет получаться один и тот же тренировочный набор, приводящий к одним и тем же результатам.

При наличии достаточно большого объема тренировочных данных более предпочтительным часто оказывается метод подвыборок. В то же время, если набор доступных данных ограничен, имеет смысл использовать метод бэггинга.

Следует подчеркнуть, что ни метод бэггинга, ни метод подвыборок не позволяет полностью устранить дисперсию, поскольку различные тренировочные выборки будут перекрываться на включенных точках данных. Поэтому между предсказаниями для тестовых примеров, полученными на основе различных выборок, будет существовать положительная корреляция. Среднее набора случайных положительно коррелированных переменных всегда будет иметь дисперсию, пропорциональную степени корреляции. В результате предсказания всегда будут содержать остаточную дисперсию. Наличие остаточной дисперсии является следствием того факта, что бэггинг и подвыборки представляют собой неидеальную имитацию извлечения тренировочных данных из базового распределения. Тем не менее при таком подходе дисперсия все равно будет ниже, чем в случае конструирования единственной модели на основе полного набора тренировочных данных. Основная трудность непосредственного использования бэггинга в нейронных сетях обусловлена необходимостью построения множества тренировочных моделей, что крайне неэффективно. Однако процесс построения различных моделей может быть полностью параллелизован и поэтому является идеальным кандидатом для выполнения тренировки с использованием нескольких GPU-процессоров.

4.5.2. Выбор и усреднение параметрических моделей

Одной из трудностей, возникающих при построении нейронных сетей, является необходимость выбора большого количества гиперпараметров, таких как глубина сети и количество нейронов в каждом слое. Кроме того, выбор функции активации также влияет на производительность. Наличие большого количества параметров создает трудности при построении моделей, поскольку функционирование последних может зависеть от того, какая конкретная конфигурация используется. Одна из возможностей заключается в резервировании определенной части тренировочных данных и испытании различных комбинаций параметров и вариантов выбора модели. Впоследствии для предсказаний используется тот вариант, который обеспечивает наивысшую точность на отложенном наборе данных. Разумеется, этот подход, носящий название *выбор модели*, является стандартом для настройки параметров любой модели машинного обучения. В определенном смысле выбор модели неразрывно связан с ансамблевым подходом, предполагающим выбор наилучшей модели из создаваемой их совокупности. Поэтому иногда на такой подход также ссылаются как на метод *“ведра моделей”* (bucket-of-models).

В задачах глубокого обучения основной проблемой является большое количество возможных конфигураций. Например, выбору подлежат количество слоев, количество элементов в каждом слое и вид функции активации. Число возможных комбинаций параметров очень велико, поэтому на практике приходится ограничиваться меньшим их количеством. Дополнительный подход, который может быть использован для уменьшения дисперсии, заключается в выборе k наилучших конфигураций с последующим усреднением предсказаний, полученных с их помощью. Такой подход приводит к более надежным предсказаниям, особенно если конфигурации значительно различаются между собой. Каждая конфигурация, взятая по отдельности, может быть не самой оптимальной, однако общее предсказание будет довольно надежным. В то же время в случае крупномасштабных приложений такой подход оказывается практически нереализуемым, поскольку для выполнения цикла вычислений с одной конфигурацией может потребоваться несколько недель. Поэтому чаще всего ограничиваются одной наилучшей конфигурацией в соответствии с подходом, описанным в разделе 3.3.1. Как и в случае бэггинга, учет большего количества конфигураций часто оказывается возможным лишь при условии использования нескольких GPU в процессе тренировки.

4.5.3. Рандомизированное отбрасывание соединений

Случайное отбрасывание соединений, связывающих различные слои в многослойной нейронной сети, часто приводит к разнообразию моделей, в которых для конструирования скрытых переменных используются различные комбинации признаков. Создаваемые при этом модели обладают, как правило, меньшей мощностью из-за введения ограничений в процесс построения модели. В то же время, поскольку исключаемые из разных моделей соединения выбираются случайным образом, предсказания различных моделей могут образовывать очень широкий спектр. Полученное на основе различных моделей усредненное предсказание часто характеризуется высокой точностью. Следует подчеркнуть, что в этом подходе веса различных моделей не разделяются, что отличает его от техники, известной как *дропаут* (dropout), или *метод исключения*.

Рандомизированное отбрасывание соединений может использоваться для создания предсказаний в задачах любого типа, а не только в задачах классификации. Например, подход на основе ансамблей автокодировщиков применялся для обнаружения выбросов [64]. Как обсуждалось в разделе 2.5.4, автокодировщики могут использоваться для обнаружения выбросов путем оценки ошибки воспроизведения данных в каждой точке. В [64] выбросы оцениваются с использованием нескольких автокодировщиков с рандомизированными соединениями, после чего полученные с их помощью оценки выбросов агрегируются для получения оценки одиночной точки данных. Однако предпочтение следует

отдавать использованию медианы, а не среднего [64]. В цитируемой работе было продемонстрировано, что такой подход позволяет повысить результирующую точность обнаружения выбросов. Следует подчеркнуть, что несмотря на кажущееся сходство этого подхода с методами *Dropout* и *DropConnect*, он отличается от них. Дело в том, что в методах наподобие *Dropout* и *DropConnect* различные компоненты ансамбля разделяют веса, тогда как в данном подходе никакого разделения весов между компонентами ансамбля не происходит.

4.5.4. Дропаут

Дропаут, или *исключение*, — это метод, в котором для создания ансамбля нейронных сетей используется семплирование узлов, а не ребер. В случае исключения некоторого узла вместе с ним исключаются и все его входящие и исходящие соединения. При этом семплируются только узлы входного и скрытых слоев сети. Отметим, что семплирование выходных узлов сделало бы невозможным создание предсказаний и вычисление функции потерь. В некоторых случаях входные узлы семплируются с вероятностью, отличающейся от вероятности семплирования скрытых узлов. Поэтому, если во всей нейронной сети содержится M узлов, то полное количество возможных семплированных сетей составляет 2^M .

Важно понимать, что рассматриваемый подход отличается от семплирования соединений, которое обсуждалось в предыдущем разделе, поскольку в данном случае *различные семплированные сети разделяют веса*. Таким образом, в методе дропаута семплирование узлов сочетается с разделением весов. В последующем процессе тренировки для обновления весов семплированной сети с помощью обратного распространения ошибки используется один пример. Процесс обучения продолжается с использованием описанных ниже действий, которые многократно повторяются в цикле по всем тренировочным точкам.

1. Семплируйте нейронную сеть из базовой сети. Каждый из входных узлов семплируется с вероятностью p_i , а каждый скрытый узел — с вероятностью p_h . Кроме того, все выборки являются независимыми. Если узел исключается из сети, то вместе с ним исключаются и все его входящие ребра.
2. Семплируйте одиночный тренировочный пример или мини-пакет тренировочных примеров.
3. Обновите веса оставшихся ребер, используя механизм обратного распространения ошибки для семплированного тренировочного примера или мини-пакета тренировочных примеров.

Общепринято исключать узлы с вероятностью 20–50%. Часто используют большие скорости обучения в сочетании с импульсом, которые подстраивают

путем ограничения максимальной нормы весов. Иными словами, вводится ограничение, в соответствии с которым L_2 -норма входных весов каждого узла не должна превышать небольшое постоянное значение, например 3 или 4.

Следует отметить, что для каждого небольшого мини-пакета тренировочных примеров используется другая нейронная сеть. Поэтому число семплированных нейронных сетей может быть довольно большим, в зависимости от размера тренировочного набора данных. В этом состоит отличие данного метода от большинства других ансамблевых методов наподобие бэггинга, в котором количество членов ансамбля редко превышает 25. В методе Dropout семплируются тысячи нейронных сетей с разделяемыми весами, и в каждом случае для обновления весов используется совсем небольшой тренировочный набор. Несмотря на то что семплируется большое количество нейронных сетей, доля семплированных сетей относительно базового количества возможных вариантов все равно остается крайне незначительной. Другое допущение, которое используется в данном классе нейронных сетей, заключается в том, что выходом является вероятностное значение. Это сказывается на способе формирования окончательного предсказания на основе совокупности предсказаний различных нейронных сетей.

Как создается предсказание для неизвестного тестового примера с помощью ансамбля нейронных сетей? Один из возможных способов заключается в создании предсказания для одного тестового примера с использованием всех семплированных нейронных сетей с последующим нахождением геометрического среднего всех вероятностей, предсказанных различными сетями. Использование геометрического среднего вместо арифметического обусловлено вышеупомянутым предположением о вероятностной природе выходов сети, а геометрическое среднее эквивалентно арифметическому усреднению логарифмической функции правдоподобия. Например, если нейронная сеть имеет k вероятностных выходов, соответствующих k классам, и j -й ансамбль создает выход $p_i^{(j)}$ для i -го класса, то оценка данного ансамбля для i -го класса вычисляется по следующей формуле:

$$p_i^{Ens} = \left[\prod_{j=1}^m p_i^{(j)} \right]^{1/m} \quad (4.10)$$

Здесь m — общее количество компонент ансамбля, которое в случае метода Dropout может быть довольно большим. Одной из проблем, связанных с получением этой оценки, является то, что использование геометрических средних приводит к ситуации, в которой сумма вероятностей по различным классам не равна 1. Поэтому значения вероятностей перенормируются так, чтобы их сумма равнялась 1:

$$p_i^{Ens} \leftarrow \frac{p_i^{Ens}}{\sum_{i=1}^k p_i^{Ens}}. \quad (4.11)$$

Главной проблемой такого подхода является слишком большое количество компонент ансамбля, что делает его неэффективным.

Важной особенностью метода Dropout является то, что для оценки предсказания вовсе не обязательно использовать все компоненты ансамбля. Вместо этого можно выполнить проход в прямом направлении лишь для базовой сети (без применения исключений), предварительно перемасштабировав веса. Базовая идея заключается в умножении исходящих весов каждого узла на вероятность семплирования данного элемента. Использование этого подхода позволяет оценить ожидаемое значение выхода данного элемента по всем подсетям. Это правило называют *правилом масштабирования весов на стадии вывода* (weight scaling inference rule). Его использование также гарантирует, что вход, поступающий в данный элемент, совпадает с ожидаемым значением входа для семплированной сети.

Указанное правило масштабирования весов строго выполняется для многих типов сетей с линейными активациями, однако в случае сетей с нелинейностями оно не всегда справедливо. Как показала практика, данное правило хорошо работает в случае самых разных сетей. Поскольку на практике в большинстве сетей используются нелинейные активации, это правило масштабирования весов в методе Dropout следует рассматривать как эвристическое, а не как теоретически обоснованный результат. Метод Dropout применялся в разнообразных моделях на основе распределенных представлений; он использовался с сетями прямого распространения, ограниченными машинами Больцмана и рекуррентными нейронными сетями.

Основной эффект применения метода Dropout сводится к встраиванию регуляризации в процедуру обучения. Исключая входные и скрытые элементы, метод Dropout эффективно внедряет шум во входные данные и скрытые представления. По своей природе этот шум может рассматриваться как маскирующий, когда некоторые входы и скрытые элементы устанавливаются в нуль. Добавление шума — это своеобразная форма регуляризации. В оригинальной статье, описывающей метод Dropout [467], продемонстрировано, что этот подход работает лучше других регуляризаторов, таких как *затухание весов* (weight decay). Метод Dropout предотвращает возникновение в скрытых элементах явления, известного как *коадаптация признаков* (feature co-adaptation). Поскольку эффект дропаута заключается в создании маскирующего шума, который удаляет некоторые из скрытых узлов, такой подход навязывает определенный уровень избыточности признаков, обучение которым происходит в различных скрытых элементах.

Эффективность метода Dropout объясняется тем, что каждая из семплированных подсетей тренируется на наименьшем из семплированных примеров. В результате работа, которая должна быть дополнительно выполнена, сводится к семплированию скрытых элементов. В то же время, поскольку Dropout — это метод регуляризации, он уменьшает выразительную способность сети. Поэтому для наиболее полного использования возможностей метода Dropout требуется использовать более крупные модели и большее количество скрытых элементов. Следствием этого являются дополнительные вычислительные расходы. Кроме того, если исходный тренировочный набор данных уже сам по себе достаточно большой для того, чтобы уменьшить вероятность переобучения, то дополнительные вычислительные преимущества метода Dropout могут оказаться небольшими, но все еще ощутимыми. Например, многие из сверточных нейронных сетей, обученных на больших репозиториях данных наподобие ImageNet [255], неизменно обеспечивают получение улучшенных результатов с точностью порядка 2%, используя метод Dropout. Вариацией метода Dropout является метод *DropConnect*, в котором аналогичный подход применяется к весам, а не узлам нейронной сети [511].

Замечание относительно коадаптации признаков

Чтобы понять причины успеха метода Dropout, полезно рассмотреть понятие *коадаптации признаков* (feature coadaptation). Было бы идеально, если бы скрытые слои нейронной сети создавали признаки, отражающие важные классификационные характеристики входа без сложных зависимостей от других признаков, если только эти другие признаки не являются действительно полезными. Чтобы разобраться в этом, рассмотрим ситуацию, в которой все входящие ребра 50% узлов в каждом слое фиксируются при своих случайных начальных значениях и не обновляются в процессе обратного распространения (даже если все градиенты вычисляются обычным способом). Интересно отметить, что даже в этом случае нейронная сеть часто сможет давать достаточно хорошие результаты путем адаптации других весов и признаков к влиянию этих случайно выбранных поднаборов весов (и соответствующих активаций). Разумеется, такая ситуация нежелательна, поскольку задача совместной работы признаков состоит в объединении возможностей, связанных с каждым существенным признаком, а не в адаптации одних признаков к пагубному влиянию на них других признаков. Коадаптация такого типа может происходить и при обычном обучении нейронной сети (в процессе которого обновляются все веса). Например, если обновления в определенных частях сети осуществляются недостаточно быстро, то некоторые признаки не будут приносить ощутимой пользы, в то время как другие будут адаптироваться к этим по сути бесполезным признакам. Такая ситуация с большой долей вероятности может возникнуть при обучении

нейронной сети, поскольку различные части сети стремятся обучаться с различной скоростью. Еще более опасный сценарий возникает, когда коадаптируемые признаки хорошо работают при предсказании тренировочных точек путем улавливания таких сложных зависимостей между ними, которые плохо обобщаются на тестовые точки, не входящие в тренировочный набор. Метод Dropout предотвращает коадаптацию такого типа за счет того, что он вынуждает нейронную сеть создавать предсказания с использованием лишь некоторого подмножества входов и активаций. Это заставляет сеть делать предсказания с определенной степенью избыточности, но одновременно стимулирует ее к тому, чтобы меньшие наборы обученных признаков обладали предсказательной силой. Иными словами, коадаптация происходит только тогда, когда для моделирования действительно важно, чтобы тем самым было вытеснено обучение случайным нюансам тренировочных данных. Конечно же, это является формой регуляризации. Кроме того, за счет обучения сети избыточным признакам метод Dropout усредняет предсказания избыточных признаков, что напоминает аналогичное усреднение в методе бэггинга.

4.5.5. Ансамбли на основе возмущения данных

Большинство ансамблевых методов, которые мы до сих пор обсуждали, относятся либо к ансамблям на основе семплирования тренировочных данных, либо к ансамблям на основе моделей. Метод Dropout можно рассматривать как ансамблевый, в котором к данным косвенным способом добавляется шум. Существуют также методы, в которых используется явное возмущение данных.

В простейшем случае во входные данные вводится небольшой шум, и веса обучаются на возмущенных данных. Этот процесс многократно повторяется, и предсказания для тестовой точки, полученные с помощью различных компонент ансамбля, усредняются. Подход такого типа является общим ансамблевым методом, а не методом, специфичным для нейронных сетей. Как будет обсуждаться в разделе 4.10, этот подход обычно применяется при обучении без учителя с использованием *шумоподавляющих автокодировщиков* (de-noising autoencoders).

Также возможно добавление шума в скрытый слой. Однако в этом случае шум должен тщательно калиброваться [382]. Следует подчеркнуть, что метод Dropout косвенно добавляет шум в скрытый слой путем случайного исключения узлов. Исключение узла аналогично маскированию шумом, устанавливающим активацию в данном узле равной нулю.

Возможны и другие типы аугментации набора данных. Например, в набор данных может быть добавлен повернутый или сдвинутый экземпляр имеющегося изображения. Тщательное проектирование схем аугментации данных часто позволяет значительно увеличить точность обучения за счет усиления

обобщающей способности. Однако, строго говоря, такие схемы не являются пертурбационными, поскольку аугментированные примеры создаются с использованием калиброванной процедуры на основании знания особенностей конкретной предметной области. Подобные методы повсеместно используются в сверточных нейронных сетях (подробно об этом — в разделе 8.3.4).

4.6. Ранняя остановка

Нейронные сети тренируются с использованием разновидностей метода градиентного спуска. В большинстве моделей оптимизации методы градиентного спуска выполняются до достижения сходимости. Однако выполнение градиентного спуска до сходимости оптимизирует функцию потерь на тренировочных данных, но не обязательно на тестовых данных, не входящих в состав тренировочного набора. Это объясняется тем, что последние несколько шагов часто приводят к переобучению сети специфическим нюансам тренировочных данных, которые могут плохо обобщаться на тестовые данные.

Естественным решением этой дилеммы является *ранняя остановка* (early stopping). В этом методе часть тренировочных данных откладывается в качестве *валидационного набора* (validation set). Обучение, основанное на обратном распространении, применяется только к той части тренировочных данных, которая не включает валидационный набор. При этом осуществляется непрерывный контроль ошибки модели на валидационном наборе. В некоторой точке эта ошибка начинает возрастать на валидационном наборе, даже если на тренировочном наборе она продолжает уменьшаться. Это та точка, начиная с которой дальнейшая тренировка приводит к переобучению. Поэтому такая точка может быть выбрана для прекращения выполнения. Важно отслеживать наилучшее (по результатам вычислений на валидационных данных) из решений, полученных на текущий момент в процессе обучения. Рекомендуется не прибегать к ранней остановке после незначительного увеличения ошибки на данных, не входящих в тренировочный набор (что может быть вызвано вариациями шума), а продолжать тренировку с целью удостовериться в том, что ошибка продолжает возрастать. Иными словами, точка остановки выбирается только после обретения полной уверенности в том, что ошибка на валидационном наборе продолжает возрастать и рассчитывать на улучшение ситуации с ошибкой на валидационном наборе не приходится.

Несмотря на то что исключение валидационного набора из тренировочных данных приводит к потере некоторой части тренировочных точек, влияние этой потери зачастую довольно незначительно. Это объясняется тем, что нейронные сети часто тренируются на чрезвычайно больших наборах данных, содержащих десятки миллионов точек. Валидационный набор не обязан содержать большое количество точек. Например, выборка размером 10 000 точек, используемая для

валидационного набора, может оказаться крайне малой по сравнению с полным набором данных. И хотя валидационный набор зачастую может включаться в тренировочные данные для повторного обучения сети в течение того же количества шагов (совершенного к моменту ранней остановки), результаты такого подхода иногда могут быть непредсказуемыми. Кроме того, этот подход может приводить к удвоению вычислительных затрат, поскольку нейронная сеть должна заново пройти всю процедуру обучения.

Одно из преимуществ ранней остановки состоит в том, что ее добавление в обучение нейронной сети не требует внесения существенных изменений в процедуру тренировки. Кроме того, методы наподобие затухания весов требуют испытания различных значений параметра регуляризации λ , что влечет за собой дополнительные вычислительные расходы. Учитывая легкость его сочетания с существующими алгоритмами, метод ранней остановки может использоваться совместно с другими регуляризаторами относительно непосредственным способом. Поэтому метод ранней остановки применяется почти всегда по той причине, что его добавление в процедуру обучения не сопровождается большими потерями.

Метод ранней остановки можно рассматривать как разновидность ограничения, налагаемого на процесс оптимизации. Ограничивая количество шагов градиентного спуска, можно эффективно ограничивать расстояние финального решения от точки инициализации. Введение ограничений в модель задачи машинного обучения часто представляет собой разновидность регуляризации.

4.6.1. Ранняя остановка с точки зрения дисперсии

Причина возникновения дилеммы смещения — дисперсии заключается в том, что для построения истинной функции потерь задачи оптимизации необходимо располагать бесконечно большим набором данных. Если имеющийся набор данных ограничен, то функция потерь, построенная на основе тренировочного набора, не будет совпадать с истинной. Пример истинной функции потерь и ее смещенного аналога, полученного с использованием тренировочных данных, приведен на рис. 4.6. Смещение функции потерь является косвенным проявлением дисперсии прогнозов, создаваемых на основе конкретного тренировочного набора данных. Использование различных тренировочных наборов будет приводить к непредсказуемым смещениям различной величины.

К сожалению, процедура обучения может выполнить градиентный спуск только для функции потерь, определенной на тренировочном наборе, поскольку истинная функция потерь неизвестна. Однако, если тренировочные данные достаточно представительны, то оптимальные решения в обоих случаях будут близки между собой. Как обсуждалось в главе 3, большинство процедур градиентного спуска выбирают рыскающий, отклоняющийся от кратчайшего путь к оптимальному решению. Во время окончательной стадии сходимости к

оптимальному решению (на тренировочных данных) градиентный спуск будет часто наталкиваться на лучшие решения по отношению к истинной функции потерь, прежде чем сойдется к лучшему решению по отношению к тренировочным данным. Эти решения будут обнаруживать себя улучшенной точностью на валидационном наборе и поэтому могут служить хорошей точкой ранней остановки. Пример подходящей точки ранней остановки приведен на рис. 4.6.

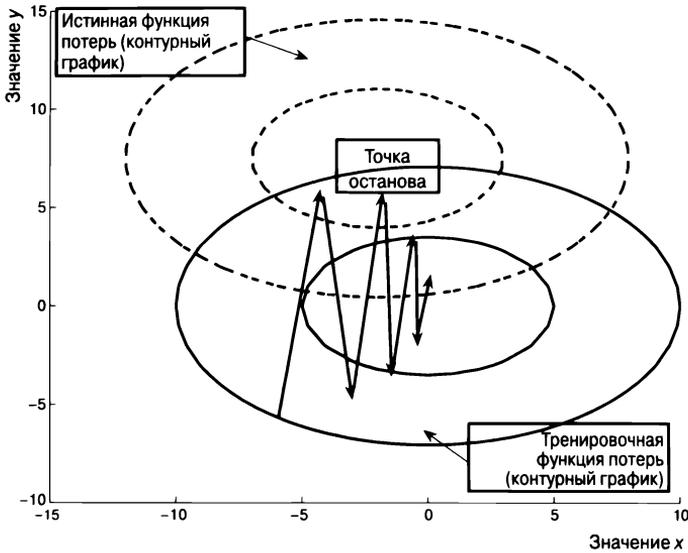


Рис. 4.6. Смещение функции потерь, обусловленное эффектами дисперсии и ранней остановки. Расхождение между истинной функцией потерь и функцией потерь, полученной на тренировочных данных, приводит к тому, что в случае выхода градиентного спуска за пределы определенной точки ошибка начинает возрастать. Для простоты обе функции, приведенные на рисунке, имеют одинаковую форму, хотя на практике они могут различаться

4.7. Предварительное обучение без учителя

Трудности обучения глубоких сетей обусловлены целым рядом причин, которые обсуждались в предыдущей главе. Одной из них является проблема взрывных и затухающих градиентов, из-за чего различные слои обучаются с разной скоростью. Существование большого количества слоев в нейронной сети приводит к искажениям градиента, что затрудняет их обучение.

Несмотря на то что глубина нейронной сети уже сама по себе создает трудности, связанные с этим проблемы в значительной мере зависят также от способа инициализации сети. Подходящим выбором точки инициализации часто удается решить многие из проблем, связанных с достижением хороших решений.

Прорыв в этом направлении обеспечило использование *предварительного обучения без учителя* (unsupervised pretraining), или *неконтролируемого обучения*, как способа нахождения надежных точек инициализации [196]. Такая инициализация достигается за счет жадной послойной тренировки сети. Первоначально данный подход был предложен в контексте *глубоких сетей доверия* (deep belief networks), но впоследствии был распространен на другие типы моделей, такие как автокодировщики [386, 506]. В этой главе мы изучим применение указанного подхода к автокодировщикам. Начнем с задачи снижения размерности, поскольку она не требует обучения с учителем, и в этом случае продемонстрировать, как работает неконтролируемая предварительная тренировка, будет несложно. В то же время, внося лишь незначительные изменения, неконтролируемую предварительную тренировку можно использовать также в таких задачах, как классификация, которые требуют обучения с учителем.

В случае предварительного обучения используется *жадный* подход, в соответствии с которым сеть тренируется по одному слою за раз путем обучения весов сначала внешних, а затем внутренних скрытых слоев. Результирующие веса используются в качестве начальных точек для финальной фазы традиционного обратного распространения ошибки в нейронных сетях с целью их тонкой настройки.

Рассмотрим архитектуры автокодировщика и классификатора. Поскольку эти архитектуры характеризуются наличием нескольких слоев, инициализация весов случайными значениями иногда может создавать проблемы. Однако можно создать неплохое начальное состояние, инициализируя веса слой за слоем в жадной манере. Мы начнем с описания процесса в контексте автокодировщика (рис. 4.7, а), хотя почти та же процедура применяется и в случае классификатора (рис. 4.7, б). Нейронные архитектуры в обоих случаях намеренно выбраны такими, чтобы скрытые слои содержали одинаковое количество узлов.

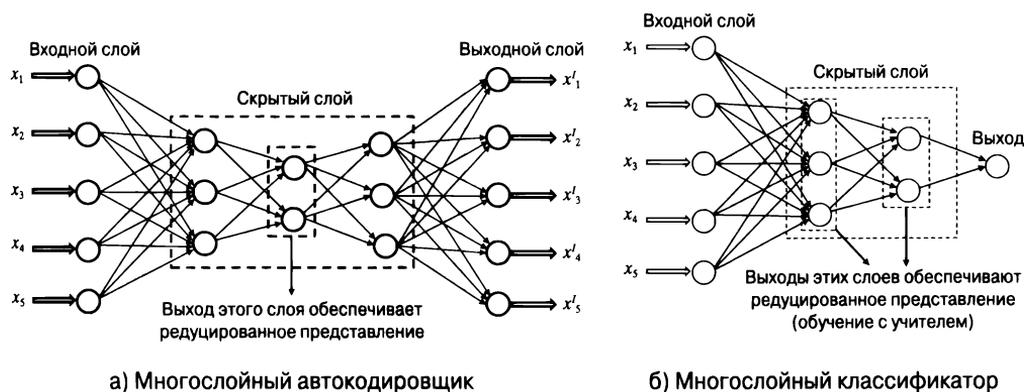


Рис. 4.7. Использование похожих процедур предварительной тренировки в многослойном автокодировщике и многослойном классификаторе

Процесс предварительной тренировки представлен на рис. 4.8. Мы предполагаем, что оба (симметричных) внешних скрытых слоя содержат редуцированное представление первого уровня большей размерности, а внутренний скрытый слой содержит редуцированное представление второго уровня меньшей размерности. Поэтому первым шагом является обучение редуцированного представления первого уровня соответствующим весам, связанным с внешними скрытыми слоями, с помощью упрощенной сети (рис. 4.8, а). В этой сети средний скрытый слой отсутствует, а два внешних скрытых слоя свернуты в один. При этом предполагается, что два внешних скрытых слоя связаны между собой симметричным образом, словно они образуют небольшой автокодировщик. На втором шаге редуцированное представление, полученное на первом шаге, используется для обучения редуцированного представления второго уровня (и весов) внутренних скрытых слоев. Поэтому внутренняя часть нейронной сети трактуется как небольшой автономный автокодировщик. Поскольку каждая из этих предварительно обучаемых подсетей имеет намного меньшие размеры, процесс обучения весов проходит легче. Затем этот начальный набор весов используется для тренировки всей нейронной сети с помощью обратного распространения ошибки. Учтите, что этот процесс может выполняться по-слойно даже в случае очень глубокой нейронной сети, содержащей любое количество скрытых слоев.

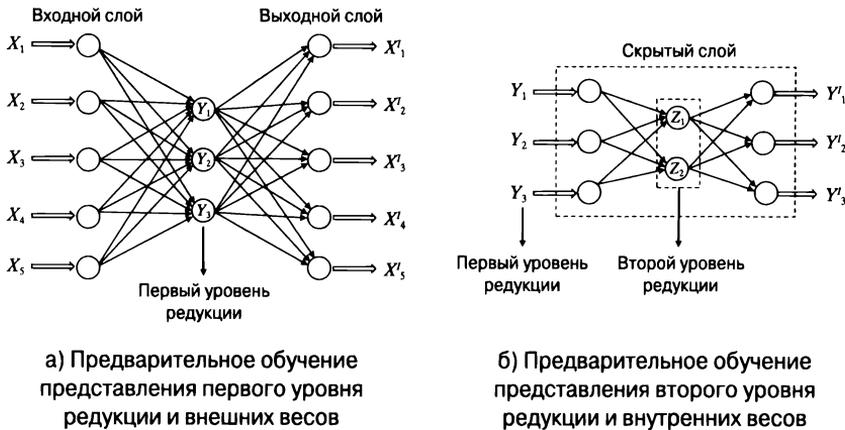


Рис. 4.8. Предварительная тренировка нейронной сети

До сих пор мы обсуждали лишь то, каким образом неконтролируемое предобучение может применяться в приложениях, ориентированных на обучение без учителя. Возникает вполне естественный вопрос: существует ли возможность использовать предобучение в приложениях, ориентированных на обучение с учителем? Рассмотрим многослойную архитектуру классификатора с единственным выходным слоем и k скрытыми слоями. На стадии предварительной

тренировки выходной слой исключается, а представление последнего скрытого слоя обучается без учителя. Это достигается за счет создания автокодировщика с $2k - 1$ скрытыми слоями, в котором средний слой является последним скрытым слоем в условиях обучения с учителем. Например, автокодировщик, используемый на рис. 4.7, б, представлен на рис. 4.7, а. Поэтому добавляются дополнительные $(k - 1)$ скрытых слоев, для каждого из которых имеется симметричный аналог в исходной сети. Эта сеть тренируется точно так же послойно, как и в случае архитектуры автокодировщика, которая обсуждалась перед этим. Для инициализации входящих весов всех скрытых слоев используется лишь часть нейронной сети, соответствующая автокодировщику. Веса соединений, связывающих последний скрытый слой с выходным слоем, также можно инициализировать, рассматривая последний скрытый слой и выходные узлы как однослойную сеть. Эта однослойная сеть получает редуцированные представления последнего скрытого слоя (созданные путем обучения автокодировщика на стадии предварительной тренировки). После обучения весов всех слоев выходные узлы заново присоединяются к последнему скрытому слою. Для тонкой настройки весов, полученных на стадии предварительной тренировки, к инициализированной описанным способом сети применяется алгоритм обратного распространения ошибки. Заметьте, что при таком подходе все начальные скрытые представления тренируются по методу обучения с учителем, и лишь входящие веса выходного слоя инициализируются с использованием меток. Поэтому предварительную тренировку в значительной мере все еще можно рассматривать как неконтролируемое обучение.

В первые годы предварительное обучение часто считалось более стабильным способом тренировки глубокой сети, благодаря которому различные слои имеют больше шансов инициализироваться одинаково эффективно. И хотя этот фактор действительно используется в качестве аргумента в пользу предварительного обучения, основная проблема часто проявляется в виде переобучения. Как обсуждалось в главе 3, веса в ранних слоях (по достижении ими окончательной сходимости) могут не испытывать заметных изменений по сравнению с их случайными начальными значениями, если в сети существует проблема затухающих градиентов. Даже если веса соединений в первых нескольких слоях являются случайными (вследствие неудачной тренировки), последующие слои по-прежнему имеют возможность достаточно эффективно адаптировать свои веса для получения нулевой ошибки на *тренировочных* данных. В данной ситуации случайные соединения в ранних слоях обуславливают почти случайные преобразования в поздних слоях, но поздние слои все еще могут переобучиться этим признакам и тем самым обеспечить очень низкую ошибку обучения. Иными словами, признаки в поздних слоях *адаптируются* к признакам в ранних слоях в результате неэффективности тренировки. Любая разновидность

коадаптации признаков, обусловленная недостатками тренировки, почти всегда приводит к переобучению. Поэтому, если применить такой подход к неизвестным тестовым данным, то переобучение станет очевидным, поскольку различные слои не адаптируются специфически к неизвестным тестовым данным. В этом смысле предварительное обучение является своеобразной формой регуляризации.

Интересно отметить, что предварительное неконтролируемое обучение оказывается полезным даже в тех случаях, когда количество тренировочных точек очень велико. Вероятно, это объясняется тем фактом, что предварительное обучение помогает справиться с другими проблемами помимо обобщаемости модели. В пользу этого свидетельствует, в частности, то, что в случае крупных наборов данных даже ошибка на тренировочных данных оказывается большой, если не использовать методы наподобие предварительного обучения. В подобных случаях веса в ранних слоях часто не изменяются сколь-нибудь существенно относительно своих первоначальных значений, и для обработки случайных преобразований (определяемых случайной инициализацией ранних слоев) используют лишь небольшое количество поздних слоев. Как следствие, обучаемая часть сети получается довольно мелкой с некоторыми дополнительными потерями, обусловленными случайным преобразованием. В этих ситуациях предварительная тренировка, кроме всего прочего, содействует реализации моделью всех преимуществ глубины, тем самым позволяя улучшить предсказательную способность на больших наборах данных.

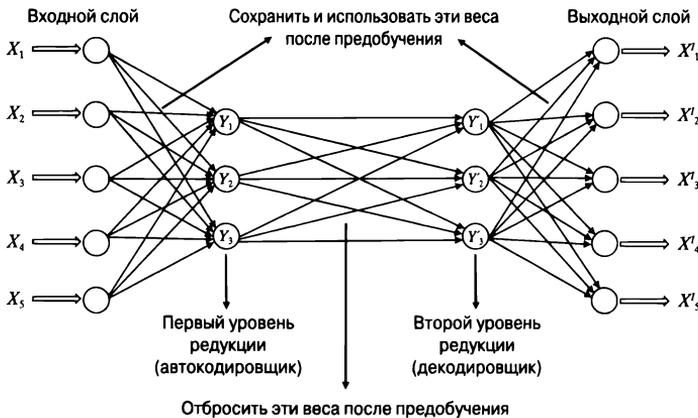


Рис. 4.9. Эта архитектура допускает значительные различия между представлениями первого уровня в кодировщике и декодировщике. Полезно сравнить ее с той, которая приведена на рис. 4.8, а

Еще одним достоинством предварительного обучения является то, что оно позволяет больше узнать о существующих среди данных скрытых устойчивых

закономерностей в виде повторяющихся шаблонов, которые есть не что иное, как признаки, улавливаемые из тренировочных точек данных. Например, автокодировщик может обучиться тому, что в цифрах имеются замкнутые кривые, а некоторые цифры содержат штрихи, искривленные определенным образом. Декодировщик реконструирует цифры, объединяя часто встречающиеся формы. Однако эти формы также обладают отличительными свойствами, используемыми для распознавания цифр. Выражение данных в терминах небольшого количества признаков помогает распознавать, каким образом признаки связаны с метками классов. Эти принципы подытожены Джеффом Хинтоном [192] в контексте классификации изображений в следующей фразе: *“Прежде чем распознавать формы, необходимо научиться генерировать изображения”*. Такой тип регуляризации осуществляет предварительную подготовку благоприятных условий для процесса тренировки, привязывая его начало к семантически близкой области пространства параметров, в которой несколько важных признаков уже изучены, так что в процессе дальнейшего обучения остается лишь выполнить тонкую настройку нескольких важных признаков и объединить их для создания предсказания.

4.7.1. Разновидности неконтролируемого предварительного обучения

Существует множество способов, позволяющих вводить различные вариации в процедуру неконтролируемого предварительного обучения. Например, вместо послойного предварительного обучения можно использовать тренировку одновременно нескольких слоев за один раз. В качестве конкретного применения такого подхода можно привести сеть VGG (она будет рассмотрена в разделе 8.4.3), в которой совместно тренировались одиннадцать слоев, входящих в состав еще более глубокой архитектуры. Действительно, в группировании как можно большего количества слоев для целей предварительного обучения есть свои преимущества, поскольку (успешная) процедура тренировки участков нейронной сети более крупного размера обеспечивает более эффективную инициализацию весов. С другой стороны, группирование слишком большого количества слоев в каждом предварительно обучаемом компоненте сети может приводить к возникновению в них всевозможных проблем (таких, как проблема затухающих и взрывных градиентов).

Следует также подчеркнуть, что в процедуре предварительной тренировки, представленной на рис. 4.8, предполагается, что автокодировщик работает полностью симметричным способом, т.е. в k -м слое кодировщика выполняется редукция (понижение размерности) примерно той же кратности, что и в зеркальном по отношению к нему слое декодировщика. На практике это допущение может стать ограничивающим, если в разных слоях используются разные

функции активации. Например, сигмоида, используемая в одном из слоев кодировщика, будет создавать лишь неотрицательные значения, тогда как функция активации в виде гиперболического тангенса, используемая в соответствующем слое декодировщика, может создавать как положительные, так и отрицательные значения. Другой подход заключается в использовании более свободной архитектуры предварительной тренировки, в которой представление на k -м уровне редукции в кодировщике и представление в зеркальном слое декодировщика обучаются по отдельности. Благодаря этому соответствующие редуцированные представления в кодировщике и декодировщике могут различаться. Для учета этих различий между соответствующими двумя слоями должен быть вставлен дополнительный слой весов, который отбрасывается после редукции, и сохраняются только веса связей “кодировщик — декодировщик”. Единственным расположением, в котором дополнительный слой весов не используется в процессе тренировки, является центральный редуцирующий слой, обработка которого осуществляется в соответствии с обсуждением, приведенным в предыдущем разделе (см. рис. 4.8, б). Пример подобной архитектуры редукции первого уровня приведен на рис. 4.9. Обратите внимание на то, что представления первого уровня редукции для кодировщика и декодировщика могут различаться, что обеспечивает дополнительную гибкость процесса предварительной тренировки. Если такой подход применяется для классификации, то могут использоваться лишь веса кодировщика, а окончательный редуцированный код может далее обрабатываться классификационным слоем в целях обучения.

4.7.2. Контролируемое предварительное обучение

До сих пор мы обсуждали только *неконтролируемую*, т.е. проводимую по методике обучения без учителя, предварительную тренировку, независимо от того, в задачах какого типа она применяется: обучения с учителем или обучения без учителя. Даже если базовое приложение было ориентировано на обучение с учителем, инициализация весов осуществлялась с использованием архитектуры неконтролируемого автокодировщика. Несмотря на принципиальную возможность выполнения контролируемой предварительной тренировки, было продемонстрировано, что этот вид предварительного обучения не приводит к таким же хорошим результатам, как и неконтролируемая тренировка, по крайней мере, в некоторых ситуациях [31, 113]. Это вовсе не означает, что контролируемое предварительное обучение совершенно бесполезно. Действительно, существуют случаи, когда глубина сети затрудняет тренировку самой сети. Например, сети, насчитывающие сотни слоев, чрезвычайно трудно поддаются тренировке из-за плохой сходимости и ряда других проблем. В подобных случаях ошибка оказывается высокой даже на *тренировочных* данных, что указывает на невозможность обеспечить эффективную работу алгоритма тренировки. *Эта*

проблема отличается от проблемы обобщаемости модели. Помимо контролируемого предварительного обучения были разработаны и другие методики, такие как создание *магистральных сетей* (highway networks) [161, 470], *вентильных сетей* (gating networks) [204] и *остаточных сетей* (residual networks) [184], позволяющие справиться с указанной проблемой. Однако эти решения не ориентированы на решение конкретной проблемы переобучения, в то время как контролируемое предварительное обучение, по всей видимости, позволяет справляться с обеими проблемами, по крайней мере, в некоторых типах сетей.

В процессе контролируемой предварительной тренировки [31] архитектура автокодировщика не используется для обучения входящих соединений скрытого слоя. На первой итерации сконструированная сеть содержит только первый скрытый слой, связанный со всеми узлами выходного слоя. На этом шаге происходит обучение весов соединений, связывающих вход со скрытым слоем, а веса выходного слоя игнорируются. В последующем выходы первого скрытого слоя используются в качестве нового представления тренировочных точек. Затем мы создаем другую нейронную сеть, содержащую первый и второй скрытые слои и выходной слой. Теперь первый скрытый слой трактуется как входной, входами которого являются преобразованные представления тренировочных точек, изученных на предыдущей итерации. Далее эти представления используются для обучения следующего слоя весов и их скрытых представлений. Описанный цикл действий повторяется вплоть до последнего слоя. И хотя этот подход действительно обеспечивает определенное улучшение результатов по сравнению с подходом, в котором не используется предварительная тренировка, он, по-видимому, работает хуже, чем неконтролируемая предварительная тренировка, по крайней мере, в некоторых случаях. В основном различия между этими двумя подходами проявляются в *ошибке обобщения* на неизвестные тестовые данные, в то время как на тренировочных данных они приводят к ошибкам примерно одного порядка [31]. Такое поведение почти всегда является надежным индикатором существования различий в степени переобучения сравниваемых методов.

Почему контролируемое предварительное обучение во многих случаях оказывается менее полезным по сравнению с неконтролируемым? Основной проблемой контролируемого предобучения является то, что оно работает в чересчур жадной манере, и ранние слои инициализируются представлениями, которые очень тесно связаны с выходами. Вследствие этого возможности глубины нейронной сети используются не в полной мере, что является просто другим типом переобучения. Важным фактором, объясняющим успешность неконтролируемой предварительной тренировки, является менее жесткая привязка обученных представлений к меткам классов, в результате чего последующему обучению становится легче изолировать эти представления

и выполнить тонкую настройку их важных характеристик. Поэтому предварительную тренировку можно рассматривать как своего рода *обучение с частичным привлечением учителя* (semi-supervised learning) (также *частичное обучение, полуавтоматическое обучение*), которое вынуждает начальные представления скрытых слоев укладываться в низкоразмерные многообразия примеров данных. Секрет успеха предварительной тренировки заключается в том, что на этих многообразиях удастся находить большее количество признаков, обладающих предсказательной силой на уровне классификации, чем в случайно выбираемых областях пространства данных. В конце концов, распределения классов испытывают лишь плавные изменения на базовых многообразиях данных. Поэтому расположения точек данных на этих многообразиях могут служить неплохими признаками для предсказания распределений классов. Таким образом, финальной фазе обучения остается лишь выполнить тонкую настройку и усилить эти признаки.

Существуют ли случаи, в которых неконтролируемая предварительная тренировка не даст никакого выигрыша? В [31] приведен пример многообразия, соответствующего распределению данных, которое не обнаруживает сколь-нибудь заметной связи с целевым объектом. Такие ситуации чаще встречаются в задачах регрессии, чем в задачах классификации. Было показано, что добавление контролируемой предварительной тренировки действительно может принести пользу в подобных случаях. Первый слой весов (связывающих входной и первый скрытый слой) тренируется с использованием градиентных обновлений значений, получаемых с помощью конструкции наподобие автокодировщика, в сочетании с жадной контролируемой тренировкой. Таким образом, обучение весов первого слоя является частично контролируемым. Последующие слои тренируются с использованием только автокодировщика. Включение контролируемого обучения на первом уровне весов автоматически встраивает определенную степень контроля и во внутренние слои. Описанный подход применяется для инициализации весов нейронной сети. После этого осуществляется тонкая настройка этих весов с использованием полностью контролируемого обратного распространения ошибки по всей сети.

4.8. Обучение с продолжением и поэтапное обучение

Из материала, изложенного в этой и предыдущей главах, отчетливо следует, что обучение параметров нейронной сети неразрывно связано с необходимостью решения задачи оптимизации, в которой оптимизируемая функция потерь имеет сложную топологическую форму. К тому же функция потерь, определяемая на

тренировочных данных, не совпадает с истинной функцией потерь, что приводит к ложным минимумам. Эти минимумы называют ложными, поскольку они могут располагаться вблизи оптимального минимума на тренировочных данных, но могут вообще оказаться не минимумами на тестовых примерах. Во многих случаях оптимизация сложной функции потерь приводит к таким решениям, обладающим слабой предсказательной силой.

Опыт предварительного обучения показывает, что упрощение задачи оптимизации (получение простых жадных решений без излишне тщательной оптимизации) часто смещает решение в направлении бассейнов лучших оптимумов на тестовых данных. Иными словами, вместо того чтобы пытаться сразу же решить сложную задачу, сначала получают упрощенные решения, а затем постепенно продвигаются в направлении сложных решений. Ниже описаны два таких подхода.

1. *Обучение с продолжением* (continuation learning). Обучение с продолжением начинают с решения упрощенной версии задачи оптимизации, после чего, используя это решение в качестве отправной точки, переходят к более сложному уточнению оптимизационной задачи и обновляют решение. Процесс повторяется до тех пор, пока не будет решена сложная задача оптимизации. Таким образом, обучение с продолжением основано на принципе перехода от простого к сложному применительно к моделям. Например, если функция потерь имеет множество локальных оптимумов, то ее можно сгладить до функции потерь с одним глобальным оптимумом и найти оптимальное решение. Дальнейшая работа продолжается путем постепенного перехода к более совершенным (более сложным) аппроксимациям до тех пор, пока не будет использоваться точная функция потерь.
2. *Поэтапное обучение* (curriculum learning). Этот тип обучения начинается с тренировки модели на простых примерах данных и продолжается с постепенным добавлением в тренировочные данные все более сложных примеров. Таким образом, поэтапное обучение основано на принципе перехода от простого к сложному применительно к данным, а не к моделям, как обучение с продолжением.

На методы обучения с продолжением и поэтапного обучения можно взглянуть с другой точки зрения, проведя аналогию с тем, как обучаются люди. Обычно мы начинаем изучение какого-либо предмета с простых понятий, а затем переходим к более сложным. Именно такое планомерное усложнение позволяет ускорить обучение ребенка. Похоже, что такой же принцип хорошо работает и в машинном обучении. Ниже методы обучения с продолжением и поэтапного обучения рассмотрены более подробно.

4.8.1. Обучение с продолжением

Эта форма обучения предполагает использование серии функций потерь $L_1 \dots L_r$, сложность оптимизации которых постепенно возрастает. Иными словами, каждую L_{i+1} оптимизировать труднее, чем L_i . Вся задача оптимизации определяется на одном и том же наборе параметров, поскольку они определяются для одной и той же сети. Сглаживание функции потерь является формой регуляризации. Каждую функцию потерь L_i можно рассматривать как сглаженную версию L_{i+1} . Оптимизация каждой L_i приближает решение к бассейну оптимальных решений с точки зрения ошибки обобщения.

Функции потерь для обучения с продолжением часто конструируют посредством *размытия* (blurring). Основная идея заключается в вычислении функции потерь в выборочных точках окрестности данной точки с последующим усреднением полученных значений для создания новой функции потерь. Так, для вычисления i -й функции потерь L_i можно использовать нормальное распределение со стандартным отклонением σ_i . Такой подход можно рассматривать как своего рода добавление шума в функцию потерь, что также является формой регуляризации. Степень размытия зависит от размера локальной области, используемой для размытия, который определяется значением σ_i . Если значение σ_i установлено слишком большим, то стоимость во всех точках будет близка и функция потерь не сможет уловить достаточное количество деталей целевой функции. Однако зачастую выполнить оптимизацию в этом случае будет очень просто. С другой стороны, задание степени размытия σ_i равной нулю позволит удерживать все детали функции потерь. Поэтому естественно начинать с большого значения σ_i , а затем уменьшать его для последующих функций потерь. Такой подход можно рассматривать как использование увеличенного количества шума для регуляризации на ранних итерациях с постепенным уменьшением уровня регуляризации по мере приближения алгоритма к решению. Подобные приемы добавления изменяющегося количества калиброванного шума во избежание локальных оптимумов постоянно встречаются во многих методах оптимизации, таких как алгоритм *имитация отжига* (simulated annealing) [244]. Основная проблема методов обучения с продолжением заключается в их дороговизне, связанной с необходимостью оптимизации серии функций потерь.

4.8.2. Поэтапное обучение

Ориентированные на *постепенное усложнение обучающих данных* методы поэтапного обучения преследуют те же цели, что и ориентированные на *постепенное усложнение модели* методы обучения с продолжением. Основная гипотеза состоит в том, что различные тренировочные наборы данных представляют различные уровни трудности для ученика. В методах поэтапного обучения ученику сначала предоставляются легкие примеры. Одним из критериев

трудного примера может служить то, что он оказывается на “неправильной” стороне границы принятия решений в случае перцептрона или SVM. Существуют и другие возможности, такие как использование байесовского классификатора. Базовая идея состоит в том, что трудные примеры часто зашумлены или представляют чрезвычайно трудные для обучения образы, которые запутывают ученика. Поэтому начинать тренировку с таких примеров не рекомендуется.

Иными словами, на начальных итерациях стохастического градиентного спуска для обучения параметров с целью “подводки” их к разумным значениям используют только легкие примеры. На более поздних итерациях к легким примерам подключают трудные. Важно включать на более поздних итерациях как легкие, так и трудные примеры, иначе произойдет переобучение на трудных примерах. Во многих случаях трудные примеры могут представлять собой исключительные образы в некоторых областях пространства или даже быть шумом. Если на поздних стадиях ученику предоставляются только трудные примеры, то общая точность не может быть высокой. Часто наилучшие результаты удается получить за счет случайного смешивания простых и трудных примеров на поздних итерациях. Доля трудных примеров по мере обучения должна увеличиваться до тех пор, пока ввод не будет представлять истинное распределение данных. Доказательства эффективности *стохастического поэтапного обучения* (stochastic curriculum learning) были приведены в литературе.

4.9. Разделение параметров

Естественной формой регуляризации, уменьшающей количество параметров модели, является разделение (совместное использование) параметров различными соединениями. Часто использование этого типа разделения параметров диктуется специфическими внутренними свойствами конкретной области. Для разделения параметров двумя узлами прежде всего требуется, чтобы вычисляемые в них функции были каким-то образом связаны между собой. Чтобы понять, так ли это, необходимо хорошо представлять себе роль конкретного вычислительного узла по отношению к входным данным. Примеры методов разделения данных приведены ниже.

1. *Разделение весов в автокодировщиках.* Симметричные веса кодировщика и декодировщика, образующие автокодировщик, часто разделяются. И хотя автокодировщик будет работать независимо от того, разделяются или не разделяются веса, разделение весов улучшает регуляризирующие свойства алгоритма. В однослойном автокодировщике с линейной активацией разделение весов навязывает ортогонализацию различных компонент матрицы весов. Это обеспечивает примерно то же уменьшение количества параметров, что и метод сингулярного разложения.

2. *Рекуррентные нейронные сети.* Они часто применяются для моделирования последовательных данных, таких как временные ряды и текст. Именно для обработки данных последнего типа чаще всего используются рекуррентные нейронные сети. В рекуррентных нейронных сетях создается разбитое по временным отрезкам представление сети, в котором нейронная сеть реплицируется по слоям, связанным с временными отметками. Поскольку предполагается, что в каждой временной отметке применяется одна и та же модель, параметры разделяются различными слоями. Рекуррентные сети подробно обсуждаются в главе 7.
3. *Сверточные нейронные сети.* Сверточные нейронные сети используются для распознавания и предсказания изображений. Соответственно входы во всех слоях сети организуются в виде прямоугольной сетки. Кроме того, веса смежных фрагментов сети обычно разделяются. Базовая идея состоит в том, что прямоугольный фрагмент изображения соответствует части визуального поля и должен интерпретироваться одинаково, независимо от того, где он располагается. Иными словами, морковь остается морковью, независимо от того, располагается ли она слева или справа на изображении. В сущности, уменьшение количества параметров, разделение весов и разреживание соединений в этих методах основано на знании внутренних семантических особенностей данных. Сверточные нейронные сети обсуждаются в главе 8.

Во многих случаях очевидно, что разделение параметров обеспечивается знанием специфических внутренних свойств данных, относящихся к конкретной области, а также пониманием того, каким образом вычислительная функция в узле связана с тренировочными данными. Об изменениях, которые необходимо внести в алгоритм обратного распространения ошибки для того, чтобы можно было использовать разделение весов, см. в разделе 3.2.9.

Дополнительным типом подобного разделения является так называемое *мягкое разделение весов* (soft weight sharing) [360], при котором веса не связываются полностью, а штрафуются за различия между ними. Например, если ожидается, что веса w_i и w_j должны быть аналогичными, то к функции потерь может быть добавлен штраф в размере $\lambda(w_i - w_j)^2 / 2$. В этом случае обновление w_i может быть увеличено на добавку $\alpha\lambda(w_j - w_i)$, а обновление w_j — на добавку $\alpha\lambda(w_i - w_j)$. Здесь α — скорость обучения. Эти типы изменений в обновлениях сближают веса.

4.10. Регуляризация в задачах обучения без учителя

Несмотря на то что переобучение случается и при обучении без учителя, это часто не является проблемой. В задачах классификации делается попытка обучить одиночный фрагмент информации, связанный с соответствующим примером, и поэтому использование параметров в количестве, превышающем количество примеров, может привести к переобучению. В случае приложений, ориентированных на обучение без учителя, в которых одиночный тренировочный пример содержит намного больше фрагментов информации, соответствующих различным измерениям, это не совсем так. В общем случае количество фрагментов информации будет зависеть от внутренней размерности набора данных. Поэтому в обучении без учителя жалобы относительно переобучения встречаются реже.

Тем не менее существует множество задач обучения без учителя, в которых полезно использовать регуляризацию. Распространенным является случай *сверхполного автокодировщика* (overcomplete autoencoder), в котором количество скрытых элементов превышает количество входных. Важной целью регуляризации в задачах обучения без учителя является подчинение обучаемых представлений некоторой структуре. У такого подхода могут быть различные, зависящие от приложения преимущества, такие как создание разреженных представлений или возможность очистки искаженных данных. Как и в случае моделей, основанных на обучении с учителем, для того чтобы навязать решению требуемые свойства, можно использовать знание особенностей данных в конкретной предметной области. В этом разделе показано, как, используя различные типы штрафов и ограничений, налагаемых на скрытые элементы, можно создать скрытые/реконструированные представления, обладающие полезными свойствами.

4.10.1. Штрафы на основе значений: разреженные автокодировщики

Штрафование разреженных скрытых элементов находит применение в таких приложениях обучения без учителя, как *разреженные автокодировщики* (sparse autoencoders). Они содержат намного большее количество скрытых элементов в каждом слое по сравнению с количеством входных элементов. При этом стимулируется приближение значений скрытых элементов к нулю за счет введения либо явного штрафования, либо ограничений. В результате большинство значений скрытых элементов будут иметь нулевые значения при достижении сходимости. Один из возможных подходов заключается в наложении L_1 -штрафов на скрытые элементы для создания скрытых представлений. Градиентный спуск с наложением L_1 -штрафов на скрытые элементы обсуждается в разделе 4.4.4.

Следует также отметить, что тема L_1 -регуляризации довольно редко обсуждается в литературе по автокодировщикам (хотя нет никаких причин, которые мешали бы ее использованию). Существуют и другие методы, основанные на ограничениях, например такие, в которых разрешена активация лишь первых k скрытых элементов. В большинстве подобных случаев ограничения выбираются таким образом, чтобы это не препятствовало внесению необходимых изменений в алгоритм обратного распространения ошибки. Например, если для активации выбраны только первые k элементов, то потокам градиента разрешено распространяться только через выбранные элементы. Методы, основанные на введении ограничений, являются всего лишь жестким вариантом методов на основе штрафов. Более подробную информацию о некоторых из этих методов обучения см. в разделе 2.5.5.1.

4.10.2. Внедрение шума: шумоподавляющие автокодировщики

Как обсуждалось в разделе 4.4.1, внедрение шума — это форма регуляризации весов на основе штрафов. Использование гауссовского шума на входе примерно эквивалентно L_2 -регуляризации в однослойных сетях с линейной активацией. В основе работы шумоподавляющего автокодировщика лежит внедрение шума, а не штрафование весов или скрытых элементов. Однако целью шумоподавляющего автокодировщика является реконструкция примеров из искаженных тренировочных данных. Поэтому тип шума должен калиброваться в соответствии с природой входа. Существует несколько типов шумов, которые можно добавлять.

1. *Гауссовский шум* (Gaussian noise). Шум такого типа подходит для входов с вещественными значениями. Добавленный шум имеет нулевое среднее и дисперсию $\lambda > 0$ для каждого входа. Здесь λ — параметр регуляризации.
2. *Маскирующий шум* (masking noise). Базовая идея заключается в искажении входов за счет задания нулевых значений для некоторой их доли f . Подход такого типа особенно полезен при работе с бинарными входами.
3. *Крпчатый черно-белый шум* (salt-and-pepper noise). В этом случае некоторая доля f входов устанавливается либо в минимальное, либо в максимальное из возможных значений по принципу “орла и решки”. Подход такого типа обычно используется для бинарных входов, для которых минимальное и максимальное значения равны 0 и 1 соответственно.

Шумоподавляющие автокодировщики полезны при обработке искаженных данных. Поэтому восстановление таких данных является их основной областью применения. Входами автокодировщика являются искаженные тренировочные

записи, выходами — восстановленные записи. Таким образом, автокодировщик учится распознавать тот факт, что входные данные искажены и нуждаются в восстановлении их истинного представления. В результате даже при наличии дефектов в тестовых данных (возникших по специфическим для приложения причинам) данный подход способен реконструировать их бездефектную версию. Следует отметить, что в тренировочные данные шум добавляется явным образом, тогда как в тестовых данных он уже имеется в силу ряда причин, специфических для приложения. Например, как показано на рис. 4.10, *вверху*, данный подход можно использовать для устранения из изображений размытия или другого рода шумов. Природа шума, добавляемого во входные тренировочные данные, должна выбираться исходя из специфики дефектов, имеющихся в тестовых данных. Поэтому, чтобы автокодировщик работал наилучшим образом, в тренировочные примеры обязательно нужно добавлять неискаженные данные. В большинстве случаев сделать это нетрудно. Например, если задача заключается в удалении шума из изображений, то тренировочные данные могут содержать высококачественные изображения в качестве выхода и искусственно размытые изображения в качестве входа. Для восстановления искаженных данных довольно часто используют сверхполные шумоподавляющие автокодировщики. Однако такой выбор зависит также от природы входа и количества добавляемого шума. Помимо того что добавление шума применяется при реконструкции изображений, оно также может использоваться в качестве великолепного регуляризатора, улучшающего результаты для данных, не входящих в тренировочный набор, даже если автокодировщик является *неполным*.

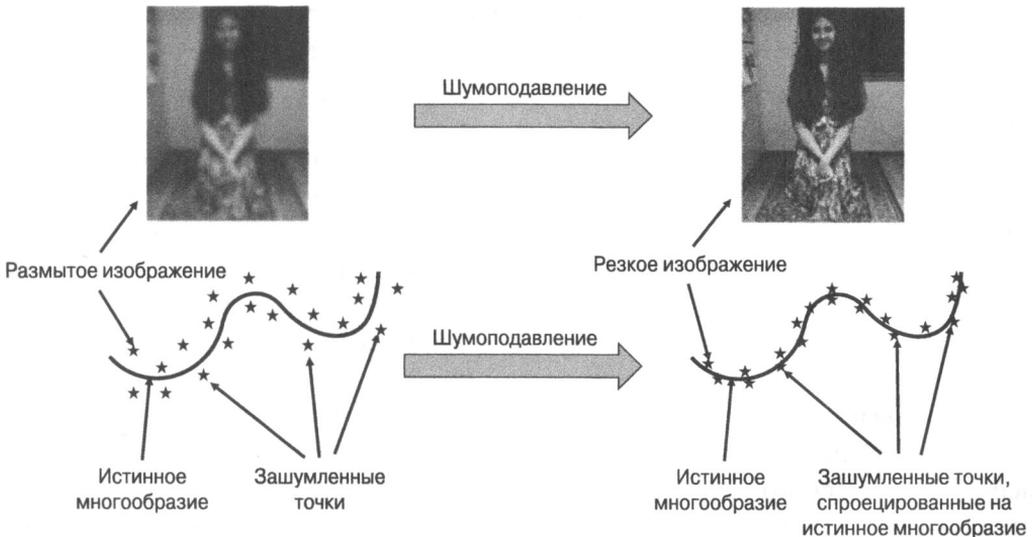


Рис. 4.10. Шумоподавляющий автокодировщик

В основе работы шумоподавляющего автокодировщика лежит использование шума во входных данных для обучения распознавания истинного многообразия, на котором распределены данные. Каждая искаженная точка проецируется на “ближайшую” к ней соответствующую точку истинного многообразия распределения данных. Ближайшая соответствующая точка — это ожидаемая позиция на многообразии, предсказываемая моделью как позиция, послужившая началом для зашумленной точки. Эта проекция приведена на рис. 4.10, *внизу*. Истинное многообразие является более компактным представлением данных, чем их зашумленная версия, и эта компактность — результат регуляризации, присущей добавлению шума на входе. Любая форма регуляризации стремится сделать базовую модель более компактной.

4.10.3. Штрафование на основе градиентов: сжимающие автокодировщики

Как и в случае шумоподавляющего автокодировщика, скрытое представление *сжимающего автокодировщика* (contractive autoencoder) часто является сверхполным, поскольку количество скрытых элементов превышает количество входных. Сжимающий автокодировщик — это автокодировщик со значительной степенью регуляризации, характеризующийся малой чувствительностью скрытого представления к небольшим изменениям входных значений. Очевидно, что следствием этого является меньшая чувствительность выхода по отношению ко входу. На первый взгляд, попытки создать автокодировщик, в котором выход менее чувствителен к изменениям на входе, могут показаться нелогичными, ведь назначение автокодировщика — точное восстановление данных. Поэтому может сложиться впечатление, что задачи регуляризации полностью расходятся с задачами той части функции потерь, которая обусловлена вкладом сжимающего автокодировщика.

В данной ситуации ключевым является то обстоятельство, что сжимающие автокодировщики должны быть робастными лишь по отношению к *небольшим* изменениям во входных данных. Кроме того, они, как правило, нечувствительны к тем изменениям, которые не согласуются со структурой многообразия данных. Иными словами, если внести во входные данные небольшое изменение, не укладывающееся в структуру многообразия входных данных, то сжимающий автокодировщик будет стремиться подавить это изменение в реконструированном представлении. Здесь важно понимать, что доля (случайно выбираемых) направлений в многомерных входных данных (с заметной долей низкоразмерного многообразия), которые окажутся приблизительно ортогональными к структуре многообразия, будет очень большой, следствием чего станет смена компонент изменения на этой структуре. В результате сжимающие автокодировщики стремятся удалить шум из входных данных подобно

шумоподавляющим автокодировщиком, однако с использованием другого механизма. Как будет показано далее, сжимающие автокодировщики штрафуют градиенты скрытых значений по входам. Низкие значения градиентов означают, что они не очень чувствительны к небольшим изменениям входных данных (хотя более существенные изменения или изменения, параллельные структуре многообразия, смогут заметно влиять на градиенты).

Для простоты рассмотрим случай, когда автокодировщик содержит только один скрытый слой. Обобщение на большее количество скрытых слоев не составляет труда. Пусть $h_1 \dots h_k$ — значения k скрытых элементов для входных переменных $x_1 \dots x_d$, а $\hat{x}_1 \dots \hat{x}_d$ — восстановленные значения в выходном слое. Поэтому целевая функция определяется взвешенной суммой функции потерь восстановления и вклада регуляризации. Функция потерь L для одиночного тренировочного примера имеет следующий вид:

$$L = \sum_{i=1}^d (x_i - \hat{x}_i)^2. \quad (4.12)$$

Вклад регуляризации формируется с использованием суммы квадратов частных производных всех скрытых переменных по всем входным измерениям. Для задачи с k скрытыми переменными $h_1 \dots h_k$ вклад регуляризации R можно записать в следующем виде:

$$R = \sum_{i=1}^d \sum_{j=1}^k \left(\frac{\partial h_j}{\partial x_i} \right)^2. \quad (4.13)$$

В оригинальной статье [397] в скрытом слое использовалась сигмоидная нелинейность. Можно показать (см. раздел 3.2.5), что в этом случае

$$\frac{\partial h_j}{\partial x_i} = w_{ij} h_j (1 - h_j) \quad \forall i, j, \quad (4.14)$$

где w_{ij} — вес соединения, связывающего входной элемент i со скрытым элементом j .

Общая целевая сумма для одиночного тренировочного примера задается взвешенной суммой функции потерь и вкладов регуляризации:

$$\begin{aligned} J &= L + \lambda \cdot R = \\ &= \sum_{i=1}^d (x_i - \hat{x}_i)^2 + \frac{\lambda}{2} \sum_{j=1}^k h_j^2 (1 - h_j)^2 \sum_{i=1}^d w_{ij}^2. \end{aligned}$$

Эта целевая функция содержит комбинацию весов и вкладов скрытых элементов в регуляризацию. Штрафы скрытых элементов обрабатываются в соответствии с обсуждением, приведенным в разделе 3.2.7. Обозначим через a_h зна-

чение преактивации для узла h_j . Обновления обратного распространения ошибки традиционно определяются в терминах преактивационных значений с использованием значения $\frac{\partial J}{\partial a_{h_j}}$, распространяемого в обратном направлении.

После вычисления значения $\frac{\partial J}{\partial a_{h_j}}$ с помощью динамического программирования обновления в процессе обратного распространения ошибки от выходного слоя его можно дополнительно обновить для включения эффекта регуляризации скрытого узла h_j :

$$\begin{aligned} \frac{\partial J}{\partial a_{h_j}} &\leftarrow \frac{\partial J}{\partial a_{h_j}} + \frac{\lambda}{2} \frac{\partial [h_j^2(1-h_j)^2]}{\partial a_{h_j}} \sum_{i=1}^d w_{ij}^2 = \\ &= \frac{\partial J}{\partial a_{h_j}} + \lambda h_j(1-h_j)(1-2h_j) \underbrace{\frac{\partial h_j}{\partial a_{h_j}}}_{h_j(1-h_j)} \sum_{i=1}^d w_{ij}^2 = \\ &= \frac{\partial J}{\partial a_{h_j}} + \lambda h_j^2(1-h_j)^2(1-2h_j) \sum_{i=1}^d w_{ij}^2. \end{aligned}$$

Производная $\frac{\partial h_j}{\partial a_{h_j}}$ равна $h_j(1-h_j)$ в силу того, что в данном случае используется сигмоида, и для других активаций она имела бы другое значение. Согласно цепному правилу, чтобы получить градиент функции потерь по w_{ij} , мы должны умножить $\frac{\partial J}{\partial a_{h_j}}$ на $\frac{\partial a_{h_j}}{\partial w_{ij}} = x_i$. Однако для получения полного градиента мы также должны в соответствии с *многомерным* цепным правилом прибавить к полученному результату производную регуляризатора по w_{ij} . Таким образом, если обозначить через R регуляризатор скрытого слоя, то полный градиент примет следующий вид:

$$\begin{aligned} \frac{\partial J}{\partial w_{ij}} &\leftarrow \frac{\partial J}{\partial a_{h_j}} \frac{\partial a_{h_j}}{\partial w_{ij}} + \lambda \frac{\partial R}{\partial w_{ij}} = \\ &= x_i \frac{\partial J}{\partial a_{h_j}} + \lambda w_{ij} h_j^2 (1-h_j)^2. \end{aligned}$$

Интересно отметить, что в случае использования линейного скрытого элемента вместо сигмоиды целевая функция совпадает с той, которая используется в автокодировщике с L_2 -регуляризацией. Поэтому описанный подход имеет смысл применять только с нелинейным скрытым слоем, поскольку линейный

скрытый слой можно обрабатывать гораздо более простым способом. Веса в кодировщике и декодировщике могут быть связанными или независимыми. В случае связанных весов необходимо суммировать градиенты, вычисленные по обоим экземплярам веса. В приведенном выше обсуждении предполагалось, что имеется всего один скрытый слой, однако полученные результаты легко обобщаются на большее количество скрытых слоев. В [397] показано, что использование более глубоких вариантов описанного подхода позволяет получать лучшие результаты.

Представляет интерес провести сравнительный анализ шумоподавляющих и сжимающих автокодировщиков. Шумоподавляющие автокодировщики достигают своих целей робастности стохастически за счет явного добавления шума, в то время как сжимающие автокодировщики достигают своих целей аналитическим способом за счет добавления вклада регуляризации. Добавление небольшого количества гауссовского шума в шумоподавляющий автокодировщик обеспечивает достижение примерно тех же целей, которые ставятся перед сжимающим автокодировщиком, если в скрытом слое используется линейная активация. В этом случае частная производная скрытого элемента по входу равна весу связи, и целевая функция сжимающего автокодировщика приобретает следующий вид:

$$J_{\text{линейная}} = \sum_{i=1}^d (x_i - \hat{x}_i)^2 + \frac{\lambda}{2} \sum_{i=1}^d \sum_{j=1}^k w_{ij}^2. \quad (4.15)$$

В подобных ситуациях оба типа автокодировщиков, как сжимающий, так и шумоподавляющий, становятся похожими на метод сингулярного разложения с L_2 -регуляризацией. Различие между шумоподавляющим и сжимающим автокодировщиком показано на рис. 4.11. Шумоподавляющий автокодировщик обучается направлениям вдоль истинного многообразия неискаженных данных, используя соотношение между искаженными данными на выходе и истинными данными на входе. В сжимающем автокодировщике эта цель достигается аналитическим способом, поскольку подавляющее большинство случайных возмущений приблизительно ортогонально многообразию, если размерность многообразия намного меньше размерности входных данных. В таком случае незначительное возмущение точки данных не вызывает сколь-нибудь существенного изменения скрытого представления. Штрафование частной производной скрытого слоя в равной степени вдоль всех направлений гарантирует, что частная производная имеет существенную величину лишь в небольшом количестве направлений вдоль истинного многообразия, тогда как частные производные вдоль подавляющего количества ортогональных направлений равны нулю. Иными словами, вариации, не являющиеся значимыми для распределения имеющегося набора тренировочных данных, подавляются, и остаются только значимые вариации.

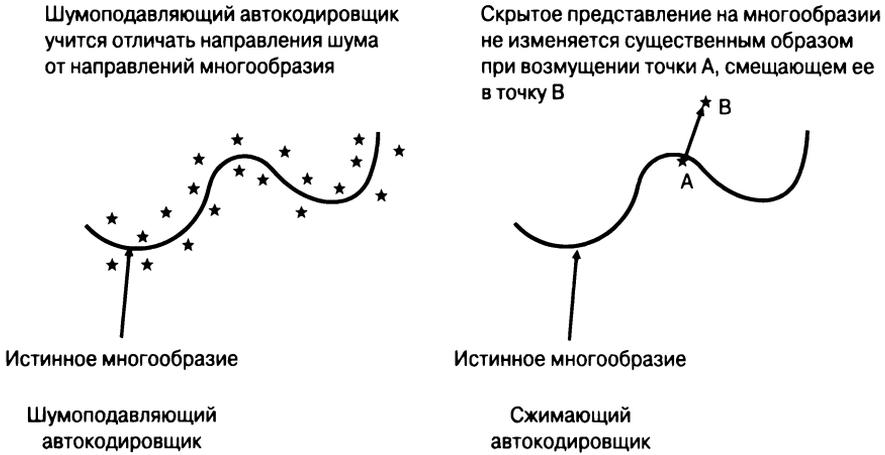


Рис. 4.11. Различие между шумоподавляющими и сжимающими автокодировщиками

Другое различие между этими двумя методами состоит в том, что шумоподавляющие автокодировщики разделяют ответственность за регуляризацию между кодировщиком и декодировщиком, тогда как сжимающий автокодировщик возлагает всю ответственность за это на кодировщика. Для извлечения признаков используется лишь кодировщик; по этой причине сжимающие автокодировщики более полезны для конструирования признаков.

В сжимающем автокодировщике градиенты детерминистичны, поэтому использовать в нем методы обучения второго порядка легче, чем в шумоподавляющем автокодировщике. С другой стороны, если используются методы обучения первого порядка, то шумоподавляющий автокодировщик создать легче (для этого достаточно внести лишь незначительные изменения в нерегуляризуемый автокодировщик).

4.10.4. Скрытая вероятностная структура: вариационные автокодировщики

Точно так же, как *разреженные кодировщики* (sparse encoders) налагают ограничение разреженности на скрытые элементы, *вариационные кодировщики* (variational encoders) подчиняют скрытые элементы определенной вероятностной структуре. Простейшее ограничение заключается в том, что для всех данных активации скрытых элементов должны извлекаться из стандартного гауссовского распределения (т.е. распределения с нулевым средним и единичной дисперсией в каждом направлении). Одним из преимуществ наложения ограничения этого типа является то, что оно позволяет отказаться от использования кодировщика и просто предоставлять декодировщику выборки из стандартного

нормального распределения для генерирования примеров тренировочных данных. Однако если все объекты генерируются из одного и того же распределения, то ни отличать различные объекты, ни восстанавливать их из заданного ввода было бы невозможно. В этом случае для получения различных распределений из стандартного нормального распределения можно использовать *условное* (по отношению к конкретному входному объекту) распределение активаций в скрытом слое. Даже если вклад регуляризации будет пытаться приблизить условное распределение к стандартному нормальному распределению, это может быть достигнуто только на распределении скрытых примеров из всего набора данных, а не на скрытых примерах из одиночного объекта.

Наложение ограничения на вероятностное распределение скрытых переменных — более сложный способ регуляризации по сравнению с теми регуляризаторами, которые мы до сих пор обсуждали. Здесь ключевое значение имеет использование подхода на основе *репараметризации*, в соответствии с которым кодировщик создает k -мерные векторы средних значений и стандартных отклонений условного гауссовского распределения, и из этого распределения семплируется скрытый вектор (рис. 4.12, а). К сожалению, подобная сеть должна включать составляющую, ответственную за семплирование. Веса такой сети не могут обучаться по механизму обратного распространения ошибки, поскольку стохастическая часть вычислений не позволяет использовать производные из-за наличия недифференцируемых членов. Пользователь может решить эту проблему, явно генерируя k -мерные примеры, каждая компонента которых извлекается из стандартного нормального распределения. Среднее значение и стандартное отклонение выхода кодировщика используются для масштабирования и преобразования входной выборки из гауссовского распределения. Эта архитектура представлена на рис. 4.12, б. В силу явного генерирования стохастической части входа результирующая архитектура становится полностью детерминированной, и ее веса могут обучаться методом обратного распространения ошибки. Кроме того, в обновлениях обратного распространения должны использоваться значения выборок, сгенерированных из стандартного нормального распределения.

Для каждого объекта \bar{X} кодировщик создает скрытые активации для среднего значения и стандартного отклонения. Обозначим через $\bar{\mu}(\bar{X})$ и $\bar{\sigma}(\bar{X})$ k -мерные активации для среднего значения и стандартного отклонения соответственно. Кроме того, из распределения $\mathcal{N}(0, I)$, где I — тождественная матрица, генерируется k -мерный пример \bar{z} , который обрабатывается пользователем как вход скрытого слоя. Скрытое представление $\bar{h}(\bar{X})$ создается масштабированием этого случайного входного вектора \bar{z} с использованием среднего значения и стандартного отклонения следующим образом:

$$\bar{h}(\bar{X}) = \bar{z} \odot \bar{\sigma}(\bar{X}) + \bar{\mu}(\bar{X}), \quad (4.16)$$

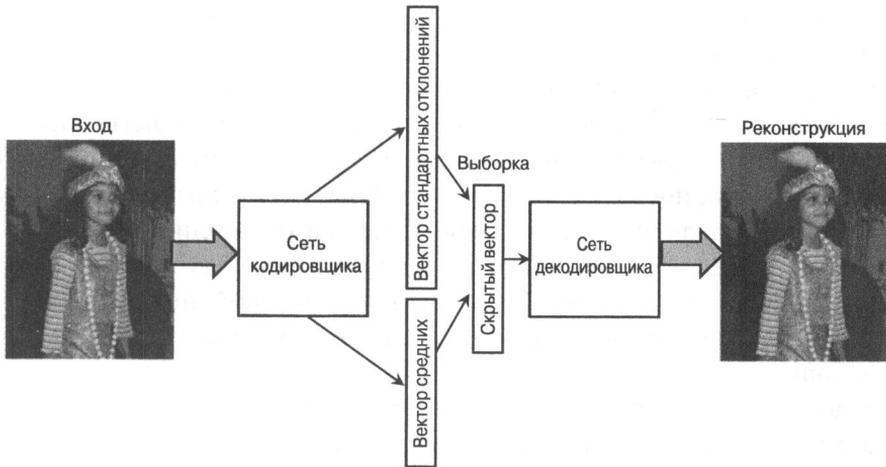
где символом \odot обозначена операция поэлементного умножения. На рис. 4.12, б, эти операции обозначены небольшими кружочками, содержащими операторы умножения и сложения. Очевидно, что элементы вектора $\bar{h}(\bar{X})$ для конкретного объекта будут отклоняться от стандартного нормального распределения, за исключением случая, когда векторы $\bar{\mu}(\bar{X})$ и $\bar{\sigma}(\bar{X})$ содержат только значения 0 и 1 соответственно. Такого не будет из-за компоненты функции потерь, соответствующей реконструкции объекта, которая вынуждает условные распределения скрытых представлений отдельных точек иметь другие средние значения и более низкие стандартные отклонения, чем в стандартном нормальном распределении (которое можно рассматривать как априорное распределение). Это распределение скрытого представления конкретной точки является апостериорным (условным по отношению к конкретной точке тренировочных данных) и поэтому будет отличаться от априорного гауссовского распределения. Общая функция потерь выражается в виде взвешенной суммы потерь реконструкции и потерь регуляризации. Существуют различные способы оценки ошибки реконструкции, но ради простоты мы будем использовать квадратичную функцию потерь, которая определяется следующим образом:

$$L = \|\bar{X} - \bar{X}'\|^2, \quad (4.17)$$

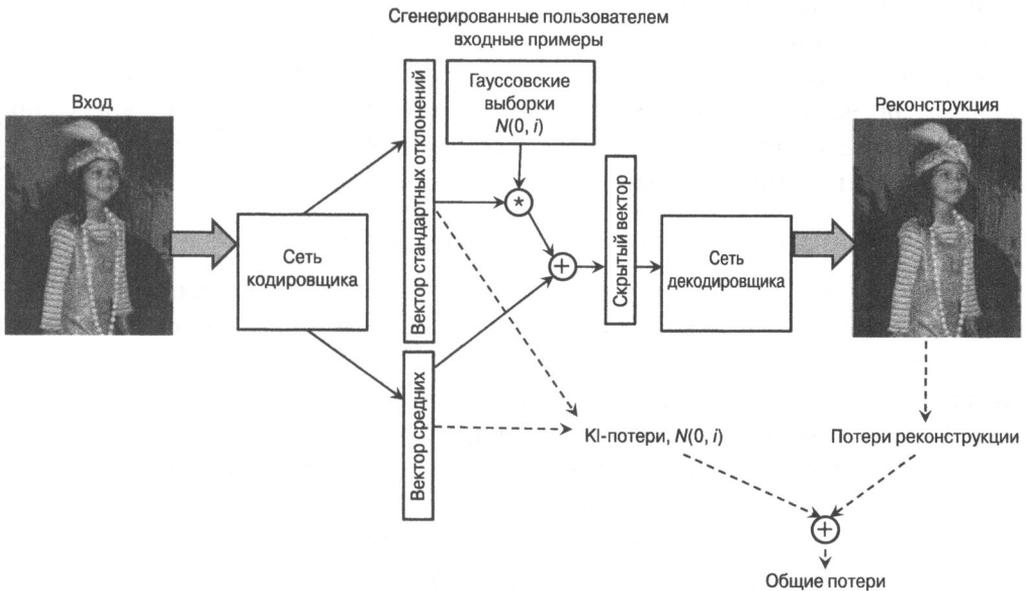
где \bar{X}' — реконструированная версия входной точки \bar{X} декодировщика. Потери регуляризации R — это просто расхождение (расстояние) Кульбака — Лейблера (KL) условного скрытого распределения с параметрами $(\bar{\mu}(\bar{X}), \bar{\sigma}(\bar{X}))$ относительно k -мерного распределения Гаусса с параметрами $(0, I)$. Данное расхождение вычисляется по следующей формуле:

$$R = \frac{1}{2} \left(\underbrace{\|\bar{\mu}(\bar{X})\|^2}_{\bar{\mu}(\bar{X})_i \Rightarrow 0} + \underbrace{\|\bar{\sigma}(\bar{X})\|^2 - 2 \sum_{i=1}^k \ln(\bar{\sigma}(\bar{X})_i)}_{\bar{\sigma}(\bar{X})_i \Rightarrow 1} - k \right) \quad (4.18)$$

Под некоторыми из членов формулы обозначены направления изменения параметров в результате воздействия этих членов. Постоянная величина k фактически несущественна, но она является частью функции расхождения Кульбака — Лейблера. Оказываемый этим постоянным членом эффект носит исключительно “косметический” характер и сводится к тому, что часть целевой функции, обусловленная вкладом регуляризации, уменьшается до нуля, если параметры $(\bar{\mu}(\bar{X}), \bar{\sigma}(\bar{X}))$ являются теми же, что и в изотропном распределении Гаусса с нулевым средним и единичной дисперсией во всех направлениях. Но это условие не будет выполняться для любой заданной точки данных из-за влияния части целевой функции, обусловленной вкладом реконструкции. Однако для всех точек тренировочных данных в целом распределение скрытого



а) Специфическое для каждой точки гауссовое распределение (недифференцируемые стохастические потери)



б) Специфическое для каждой точки гауссовое распределение (дифференцируемые детерминированные потери)

Рис. 4.12. Репараметризация вариационного автокодировщика

представления будет приближаться к стандартному гауссовскому распределению из-за влияния вклада регуляризации. Общая целевая функция J для точки данных \bar{X} определяется как взвешенная сумма компонент функции потерь, соответствующих реконструкции и регуляризации:

$$J = L + \lambda R, \quad (4.19)$$

где $\lambda > 0$ — параметр регуляризации. При небольших значениях λ предпочтение будет отдаваться точной реконструкции, и поведение модели будет напоминать поведение традиционного автокодировщика. Вклад регуляризации стимулирует стохастичность скрытых представлений, чтобы множество скрытых представлений генерировало почти одну и ту же точку. Это усиливает обобщающую способность модели, поскольку легче моделировать новое изображение, напоминающее (но не являющееся точным подобием) изображения из тренировочных данных в пределах стохастического диапазона скрытых значений. В то же время, поскольку распределения скрытых представлений аналогичных точек будут перекрываться, это дает нежелательные побочные эффекты. Например, в случае использования такого подхода для реконструкции изображений они могут размываться. Это является следствием усреднения значений по точкам, являющимся в некотором отношении близкими. В предельном случае, если значение λ выбрано чрезмерно большим, все точки будут иметь одно и то же скрытое распределение (а именно изотропное распределение Гаусса с нулевым средним и единичной дисперсией). Реконструкция может предоставить усреднение, выполненное по настолько большому количеству тренировочных точек, что в нем не будет никакого смысла. Размытие изображений, реконструируемых вариационным автокодировщиком, является недостатком моделей данного класса по сравнению с некоторыми другими родственными моделями, предназначенными для генеративного моделирования.

Обучение вариационного автокодировщика

Тренировка вариационного автокодировщика осуществляется сравнительно простым способом, поскольку стохастичность была выделена в качестве дополнительного входа. Алгоритм обратного распространения ошибки может применяться как в любой традиционной нейронной сети. Единственное различие состоит в том, что для обратного распространения используется необычная форма уравнения 4.16. Кроме того, в процессе обратного распространения ошибки необходимо учесть штрафы скрытого слоя.

Сначала можно распространить функцию потерь L в обратном направлении вплоть до скрытого состояния $\bar{h}(\bar{X}) = (h_1 \dots h_k)$, используя традиционные методы. Обозначим через $\bar{z} = (z_1 \dots z_k)$ вектор, содержащий k случайных выборок из распределения $\mathcal{N}(0, 1)$, которые используются в текущей итерации. Чтобы выполнить обратное распространение от $\bar{h}(\bar{X})$ до $\bar{\mu}(\bar{X}) = (\mu_1 \dots \mu_k)$ и $\bar{\sigma}(\bar{X}) = (\sigma_1 \dots \sigma_k)$, можно использовать следующее соотношение:

$$J = L + \lambda R, \quad (4.20)$$

$$\frac{\partial J}{\partial \mu_i} = \frac{\partial L}{\partial h_i} \underbrace{\frac{\partial h_i}{\partial \mu_i}}_{=1} + \lambda \frac{\partial R}{\partial \mu_i}, \quad (4.21)$$

$$\frac{\partial J}{\partial \sigma_i} = \frac{\partial L}{\partial h_i} \underbrace{\frac{\partial h_i}{\partial \sigma_i}}_{=z_i} + \lambda \frac{\partial R}{\partial \sigma_i}. \quad (4.22)$$

Значения под фигурными скобками — это оценки частных производных h_i по μ_i и σ_i соответственно. Обратите внимание на то, что равенства $\frac{\partial h_i}{\partial \mu_i} = 1$ и $\frac{\partial h_i}{\partial \sigma_i} = z_i$ получены дифференцированием уравнения 4.16 по μ_i и σ_i соответственно. Значение $\frac{\partial L}{\partial h_i}$ в правой части доступно из обратного распространения ошибки. Значения $\frac{\partial R}{\partial \mu_i}$ и $\frac{\partial R}{\partial \sigma_i}$ являются непосредственными производными KL-расхождения (уравнение 4.18). Последующее распространение ошибки от активаций для $\bar{\mu}(\bar{X})$ и $\bar{\sigma}(\bar{X})$ может происходить в соответствии с обычной работой алгоритма обратного распространения.

Архитектура вариационного автокодировщика считается фундаментально отличной от других типов автокодировщиков, поскольку она моделирует скрытые переменные стохастически. Но некоторые точки соприкосновения все-таки существуют. В шумоподавляющем автокодировщике на входе добавляется шум; в то же время на форму скрытого распределения не налагается никаких ограничений. В вариационном автокодировщике используется стохастическое скрытое представление, хотя стохастичность создается за счет использования дополнительного входа в процессе тренировки. Иными словами, шум добавляется в скрытое представление, а не во входные данные. Вариационный подход, поощряющий отображение каждого входа не на единственную точку, а на собственную стохастическую область в скрытом пространстве, улучшает обобщаемость модели. Поэтому небольшие изменения в скрытом представлении не приводят к существенным изменениям в восстановленной версии. То же самое относится и к сжимающему автокодировщику. Однако введение ограничения в виде предположения о гауссовской форме скрытого распределения действительно является наиболее существенным отличием вариационного автокодировщика от других типов преобразований.

4.10.4.1. Реконструкция и генеративное семплирование

Данный подход можно использовать как для создания представлений пониженной размерности, так и для генерирования выборок. В случае понижения размерности данных получается распределение Гаусса со средним $\bar{\mu}(\bar{X})$ и стандартным отклонением $\bar{\sigma}(\bar{X})$, которое используется в качестве распределения скрытых представлений.

Однако особенно интересным применением вариационных автокодировщиков является генерирование выборочных образцов из базового распределения данных. Точно так же, как в методах конструирования признаков используется только та часть автокодировщика, которая соответствует кодировщику (когда тренировка уже выполнена), вариационные автокодировщики используют только часть, соответствующую декодировщику. Базовая идея заключается в том, чтобы извлекать точку из распределения Гаусса и передавать ее скрытым элементам декодировщика. Результирующий “реконструированный” вывод декодировщика будет точкой, подчиняющейся тому же распределению, что и исходные данные. Вследствие этого сгенерированная точка будет служить реалистичным примером исходных данных. Архитектура для генерирования выборок представлена на рис. 4.13. Изображения приведены исключительно в иллюстративных целях и не отражают фактический выход вариационного автокодировщика (который обычно имеет несколько более низкое качество).

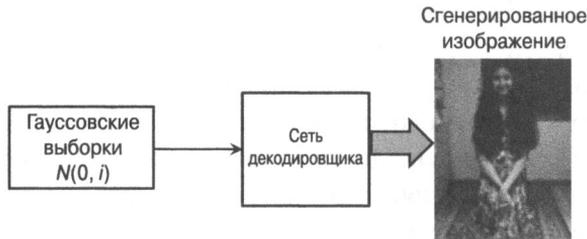


Рис. 4.13. Генерирование образцов с помощью вариационного автокодировщика. Изображения приведены исключительно в иллюстративных целях

Чтобы понять, почему вариационный автокодировщик способен генерировать изображения описанным способом, будет полезно рассмотреть распространенные типы распределенных представлений, или *вложений* (embeddings), создаваемых нерегуляризованным автокодировщиком, в сравнении с представлениями, создаваемыми с помощью других методов, таких как вариационный автокодировщик. На рис. 4.14, *слева*, приведен пример двухмерного распределенного представления тренировочных данных, созданных нерегуляризованным автокодировщиком распределения четырех классов (в данном случае четырех цифр набора MNIST). Очевидно, что в определенных областях латентного пространства существуют большие разрывы, и эти разреженные области могут не соответствовать значимым точкам. С другой стороны, регуляризационный член в вариационном автокодировщике стимулирует к тому, чтобы тренировочные точки распределялись (в общих чертах) в соответствии с гауссовским распределением, в результате чего размеры областей разрывов сокращаются (рис. 4.14, *справа*). Следовательно, семплирование из любой точки латентного

пространства будет создавать разумные реконструкции одного из четырех классов (т.е. одной из цифр набора MNIST). Кроме того, переход из одной точки латентного пространства в другую вдоль прямой линии во втором случае приведет к более гладкому преобразованию между классами. Например, переход из области, содержащей экземпляры '4', в область, содержащую экземпляры '7', в латентном пространстве набора данных MNIST будет сопровождаться плавным изменением стиля цифры '4' вплоть до точки перехода из одного класса в другой, в которой рукописная цифра может интерпретироваться либо как '4', либо как '7'. С подобными ситуациями можно столкнуться и в реальных условиях, поскольку в наборе данных MNIST содержатся также цифры с неразборчивым написанием. Кроме того, размещение различных цифр во вложении будет таким, что пары цифр, допускающих плавные переходы в неоднозначных точках (например, [4, 7] или [5, 6]), будут соседствовать в латентном пространстве.

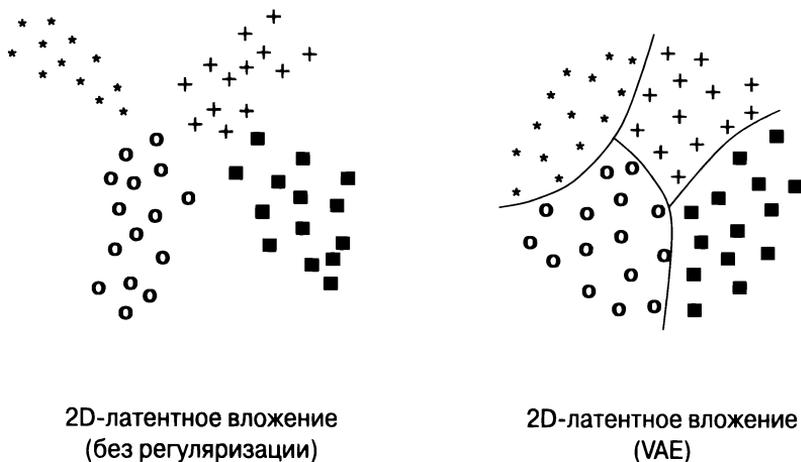


Рис. 4.14. Распределенные представления, созданные вариационным автокодировщиком и нерегуляризованной версией автокодировщика. Нерегуляризованная версия характеризуется наличием больших разрывов в латентном пространстве, которые могут не соответствовать значимым точкам. В случае вариационного автокодировщика гауссовское распределенное представление делает возможным семплирование

Важно понимать, что сгенерированные объекты часто напоминают, но не в точности совпадают с объектами, извлеченными из тренировочных данных. В силу своей стохастической природы вариационный автокодировщик способен исследовать различные модели процесса генерации, что открывает определенный простор для креативности перед лицом неоднозначности. Это свойство можно обратить во благо, модифицируя данный подход применительно к другому объекту.

4.10.4.2. Условные вариационные автокодировщики

Интересные результаты удается получать с помощью вариационных автокодировщиков, модифицированных в соответствии с заданными условиями [510, 463]. Принцип работы условных вариационных автокодировщиков основан на добавлении дополнительного условного входа, который обычно предоставляет родственный контекст. Например, контекстом может быть поврежденное изображение с отсутствующими участками, и задачей автокодировщика является его реконструкция. Как правило, предсказательные модели плохо справляются с такими задачами, поскольку уровень неоднозначности может быть довольно высоким, и реконструкция, усредненная по многим изображениям, может оказаться бесполезной. Во время фазы тренировки необходимо использовать пары поврежденных и исходных изображений, чтобы кодировщик и декодировщик могли научиться распознавать, как контексты соотносятся с изображениями, генерируемыми из тренировочных данных. Архитектура тренировочной фазы приведена на рис 4.15, *вверху*. Во всех остальных отношениях тренировка осуществляется так же, как и в случае условного вариационного автокодировщика. Во время фазы тестирования контекст предоставляется в виде дополнительного входа, и автокодировщик реконструирует отсутствующую часть разумным способом на основании модели, обученной на стадии тренировки. Архитектура фазы восстановления изображения приведена на рис. 4.15, *внизу*. Бросается в глаза ее простота. Изображения, приведенные на рисунке, являются исключительно иллюстративными; в реальных условиях сгенерированные изображения часто получаются размытыми, особенно в тех областях, где изображение отсутствовало. К рассмотрению такого подхода к трансляции изображений в изображения мы еще вернемся в главе 10 в контексте обсуждения генеративно-состязательных сетей.

4.10.4.3. Взаимосвязь с генеративно-состязательными сетями

Вариационные автокодировщики тесно связаны с другим классом моделей, так называемыми *генеративно-состязательными сетями* (generative adversarial network — GAN). Однако между ними также имеются различия. Подобно вариационным автокодировщикам, генеративно-состязательные сети могут использоваться для создания изображений, похожих на изображения из базового тренировочного набора. Кроме того, условные варианты обеих моделей могут быть полезными для восполнения отсутствующих данных, особенно в случаях, когда неоднозначность довольно велика и от креативного процесса требуется определенная степень креативности. В то же время результаты генеративно-состязательных сетей часто оказываются более реалистичными, поскольку декодировщики явным образом обучаются созданию хороших подделок. Это

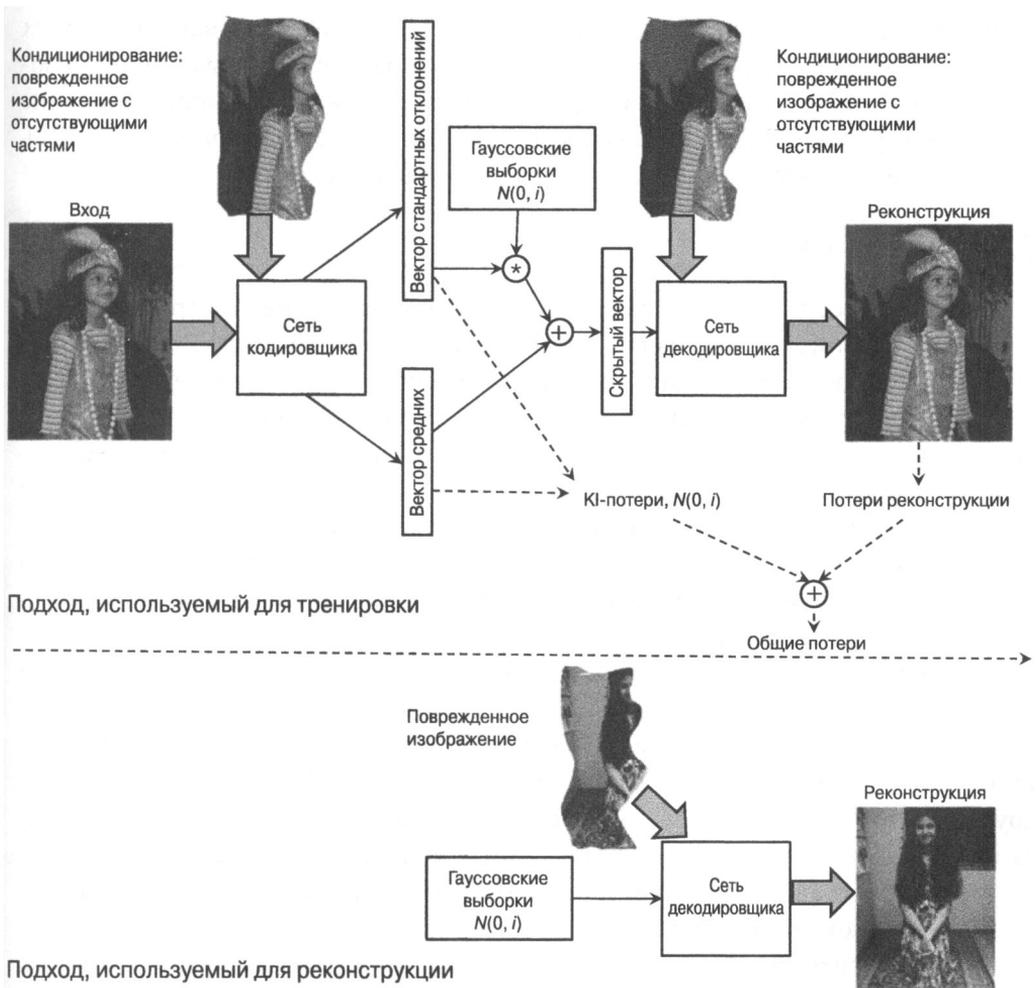


Рис. 4.15. Восстановление поврежденных изображений с помощью условного вариационного автокодировщика. Данные изображения использованы исключительно в иллюстративных целях

достигается за счет введения дискриминатора, который оценивает качество генерируемых объектов. Кроме того, для генерации объектов используется более креативный способ, поскольку исходные объекты из тренировочного набора вообще не предоставляются генератору, но его действия управляются таким образом, чтобы обмануть дискриминатор. В результате генеративно-сопоставительные сети обучаются созданию искусственных подделок. В случае некоторых типов данных, таких как изображения или видео, этот подход может обеспечивать замечательные результаты. В отличие от вариационных автокодировщиков, изображения не размываются. Тем самым можно создавать реалистичные изображения

и видео в стиле фэнтези. Подобная методика может использоваться в самых различных целях, скажем, для преобразования текста в изображения или изображений в изображения. Например, можно предоставить текстовое описание и получить фантазийное изображение, соответствующее этому описанию [392]. Генеративно-сопоставительные сети обсуждаются в разделе 10.4.

4.11. Резюме

Нейронные сети часто содержат большое количество параметров, что чревато переобучением. Одно из решений заключается в заблаговременном ограничении размеров сети. Однако такой подход часто приводит к неоптимальным решениям в случае сложных моделей и достаточно большого количества данных. Более гибкий подход заключается в использовании настраиваемой регуляризации, допускающей большое количество параметров. В подобных случаях ограничения на размеры пространства параметров, налагаемые регуляризацией, не являются жесткими. Наиболее общей формой регуляризации является регуляризация на основе штрафов. Общепринято налагать штрафы на параметры, хотя также возможно применение штрафов в отношении активаций скрытых элементов. Последний подход приводит к разреженным представлениям скрытых элементов. Общепринятым подходом, используемым для уменьшения дисперсии, является ансамблевое обучение, и некоторые ансамблевые методы, такие как *Dropout*, предназначены специально для нейронных сетей. К числу других часто используемых методов регуляризации относятся ранняя остановка и предварительное обучение. Последнее, используемое в качестве регуляризатора, представляет собой разновидность частичного обучения с учителем, осуществляемого по принципу “от простого к сложному” путем инициализации параметров в соответствии с простой эвристикой, а также путем обратного распространения ошибки для нахождения улучшенных решений. Кроме того, существует ряд сходных методов, таких как поэтапное обучение и обучение с продолжением, которые также действуют по принципу “от простого к сложному” для предоставления решений с небольшой ошибкой обобщения. И хотя в задачах обучения без учителя переобучение редко представляет собой серьезную проблему, обучаемые модели часто подчиняются определенной структуре с помощью различных типов регуляризации.

4.12. Библиографическая справка

Подробное обсуждение дилеммы смещения — дисперсии содержится в [177]. Эта дилемма возникла в области статистики, где она рассматривалась в контексте задачи регрессии. Обобщение на случай бинарной функции потерь в задачах классификации было предложено в [247, 252]. В [175, 282] для ослабления

эффектов переобучения были предложены методы, в которых количество параметров снижалось за счет удаления из сети несущественных весов. Было показано, что этот тип урезания сети обладает значительными преимуществами в отношении улучшения обобщающей способности. Также в одной из ранних работ было показано, что глубокие и узкие сети обычно обобщаются лучше, чем широкие и мелкие [450]. В первую очередь это объясняется тем, что глубина сети подчиняет данные определенной структуре и может представлять их с использованием меньшего количества параметров. Результаты недавнего исследования обобщающей способности моделей в нейронных сетях представлены в [557].

Начало исследованиям L_2 -регуляризации в регрессии было положено работами Тихонова и Арсенина [499]. Эквивалентность регуляризации по Тихонову и обучения с шумом была доказана Бишопом [44]. Использование L_1 -регуляризации подробно изучено в [179]. Также было предложено несколько методов регуляризации, специально предназначенных для нейронных архитектур. Например, в [201] предлагается техника регуляризации, которая ограничивает норму каждого слоя нейронной сети. Разреженные представления данных исследуются в [67, 273, 274, 284, 354].

Подробное обсуждение использования ансамблевых методов в задачах классификации можно найти в [438, 566]. Методы бэггинга и подвыборки обсуждаются в [50, 56]. В [515] предлагается ансамблевая архитектура, основанная на идее случайного леса. Эта архитектура представлена на рис. 1.16. Ансамбли этого типа хорошо подходят для задач с небольшими наборами данных, в которых, как известно, случайные леса работают весьма эффективно. Подход, основанный на случайном исключении ребер, был введен в контексте обнаружения выбросов [64], тогда как метод *Dropout* был представлен в [467]. В [567] обсуждается идея, в соответствии с которой объединение результатов нескольких лучших компонент ансамбля приводит к лучшему общему результату, чем объединение результатов всех компонент. Целью большинства ансамблевых методов является уменьшение дисперсии, хотя некоторые подходы, такие как *бустинг* (boosting) [122], предназначены и для уменьшения смещения. Бустинг также использовался в контексте обучения нейронных сетей [435]. Однако применение бустинга в нейронных сетях обычно ограничивается инкрементным добавлением скрытых элементов на основании характеристик ошибки. Важно знать, что бустинг, как правило, приводит к переобучению данных и поэтому подходит для моделей обучения, характеризующихся большим смещением, но не для моделей обучения, характеризующихся большой дисперсией. Обучению с помощью нейронных сетей свойственна высокая дисперсия. На связь бустинга с некоторыми типами нейронных архитектур указано в [32]. Использование методов возмущения данных в задачах классификации обсуждается в [63], однако эти методы главным образом имеют отношение к увеличению объема

имеющихся данных о классе, находящемся в меньшинстве, а потому их связь с методами уменьшения дисперсии не обсуждалась. Обсуждение возможности сочетания данного подхода с методами снижения дисперсии содержится в [5]. Ансамблевые методы для нейронных сетей предлагаются в [170].

В контексте нейронных сетей исследовались различные типы предобучения [31, 113, 196, 386, 506]. Ранние методы предварительной тренировки для обучения без учителя были предложены в [196]. Эта работа базировалась на вероятностных графических моделях (раздел 6.7) и впоследствии была распространена на обычные автокодировщики [386, 506]. По сравнению с предварительной тренировкой для обучения без учителя предварительная тренировка для обучения с учителем имеет лишь незначительный эффект [31]. Подробное обсуждение причин того, почему предварительная тренировка для обучения без учителя оказывается столь полезной для глубокого обучения, содержится в [113]. В этой работе постулируется, что предварительная тренировка при обучении без учителя действует в качестве регуляризатора и поэтому улучшает обобщаемость модели на неизвестные тестовые примеры. Это подтверждается также экспериментальными результатами [31], свидетельствующими о том, что различные варианты предварительной тренировки, ориентированной на обучение с учителем, оказывают гораздо меньший эффект, чем варианты, ориентированные на обучение без учителя. В этом смысле предварительная тренировка для обучения без учителя может рассматриваться как разновидность частичного обучения с учителем (*semi-supervised learning*), ограничивающего поиск параметров определенными областями пространства параметров, которые зависят от базового распределения имеющихся данных. Кроме того, в случае некоторых задач предварительная тренировка, по-видимому, приносит мало пользы [303]. Другая форма частичного обучения с учителем может использоваться в *лестничных сетях* (*ladder networks*) [388, 502], в которых исключение связей сочетается с архитектурой наподобие автокодировщика.

Работа методов поэтапного обучения и обучения с продолжением основана на переходе от простых моделей к сложным. Методы обучения с продолжением обсуждаются в [339, 536]. Ранее был предложен ряд методов [112, 422, 464], продемонстрировавших преимущества поэтапного обучения. Его базовые принципы обсуждаются в [238]. Связь между методами поэтапного обучения и обучения с продолжением исследуется в [33].

Многочисленные методы обучения без учителя предлагались для регуляризации. Обсуждение разреженных автокодировщиков содержится в [354]. Шумоподавляющие автокодировщики описаны в [506]. Сжимающий автокодировщик обсуждается в [397]. Использование шумоподавляющих автокодировщиков в рекомендательных системах описано в [472, 535]. Идеи, лежащие в основе работы сжимающего автокодировщика, напоминают *двойное обратное*

распространение (double backpropagation) [107], при котором небольшие изменения на входе не изменяют выхода. Сходные идеи рассматриваются применительно к *тангенциальному классификатору* (tangent classifier) [398].

Вариационный автокодировщик вводится в [242, 399]. Использование значимости весов для улучшения представлений, обучаемых вариационным автокодировщиком, рассматривается в [58]. Условные вариационные автокодировщики описаны в [463, 510]. Также имеется руководство по работе с вариационными автокодировщиками [106]. Генеративные варианты шумоподавляющих автокодировщиков приведены в [34]. Вариационные автокодировщики тесно связаны с генеративно-сопоставительными сетями, которые обсуждаются в главе 10. Тесно связанные с ними методы проектирования сопоставительных автокодировщиков рассматриваются в [311].

4.12.1. Программные ресурсы

Многочисленные ансамблевые методы доступны в библиотеках машинного обучения, таких как *scikit-learn* [587]. Большинство методов, основанных на затухании весов и штрафах, доступно в виде стандартизированных опций в библиотеках глубокого обучения. Однако такие методы, как *Dropout*, специфичны для приложений и должны реализовываться с нуля. Реализации ряда автокодировщиков различного типа можно найти на сайте GitHub [595]. Там же вы найдете и несколько реализаций вариационных автокодировщиков [596, 597, 640].

4.13. Упражнения

1. Рассмотрим две нейронные сети, используемые для регрессионного моделирования, которые имеют одинаковую структуру входного слоя и включают по 10 скрытых слоев, каждый из которых содержит по 100 элементов. В обоих случаях выходным узлом является одиночный элемент с линейной активацией. Единственным отличием является то, что в одной из сетей в скрытых слоях используются линейные активации, а в другой — сигмоидные. В какой модели дисперсия предсказаний будет больше?
2. Рассмотрим ситуацию, когда имеются четыре атрибута $x_1 \dots x_4$, а зависимая переменная y такова, что $y = 2x_1$. Создайте небольшой тренировочный набор данных, включающий 5 различных примеров, в котором линейная регрессионная модель будет иметь бесконечное количество решений для коэффициентов с $w_1 = 0$. Проанализируйте, как эта модель будет работать с данными, не входящими в тренировочный набор. Почему в данном случае пригодится регуляризация?

3. Реализуйте перцептрон с регуляризацией и без регуляризации. Протестируйте обе разновидности перцептрона на тренировочных данных и на данных, не входящих в тренировочные, используя для этого набор данных *Ionosphere* из хранилища UCI Machine Learning Repository [601]. Что можно сказать относительно влияния регуляризации в обоих случаях? Повторите этот эксперимент с меньшими примерами из тренировочных данных *Ionosphere* и запишите свои наблюдения.
4. Реализуйте автокодировщик с одним скрытым слоем. Реконструируйте входы для набора данных *Ionosphere* из предыдущего примера, используя два варианта: а) без добавления шума, но с регуляризацией весов, и б) с добавлением гауссовского шума, но без регуляризации весов.
5. В этой главе приводился пример сжимающего автокодировщика с сигмоидой. Рассмотрите сжимающий автокодировщик с одним скрытым слоем и активацией ReLU. Проанализируйте, как изменяются обновления в случае использования ReLU-активации.
6. Предположим, у вас есть модель, обеспечивающая точность около 80% как на тренировочном наборе данных, так и на тестовых данных, не входящих в тренировочный набор. Рекомендовали бы вы увеличение набора данных или настройку модели для повышения точности?
7. В этой главе было показано, что добавление гауссовского шума во входные признаки в линейной регрессии эквивалентно L_2 -регуляризации линейной регрессии. Подумайте, почему добавление гауссовского шума во входные данные шумоподавляющего автокодировщика с одним скрытым слоем и линейными элементами примерно эквивалентно L_2 -регуляризации сингулярного разложения.
8. Рассмотрим сеть с одним входным слоем, двумя скрытыми слоями и одним выходом, предсказывающим бинарную метку. Все скрытые слои используют сигмоиду, но регуляризация не используется. Входной слой содержит d элементов, а каждый скрытый слой — p элементов. Предположим, вы добавили дополнительный скрытый слой между двумя имеющимися скрытыми слоями, и этот дополнительный скрытый слой содержит q линейных элементов.
 - А. Объясните, почему мощность этой модели уменьшится, если $q < p$, несмотря на то что добавление скрытого слоя увеличивает количество параметров.
 - Б. Увеличится ли мощность модели, если $q > p$?
9. Боб разделил помеченные классифицированные данные на две части: одну для построения модели, другую — для ее валидации. Затем Боб

выполнил тестирование 1000 нейронных архитектур, обучая параметры (с помощью алгоритма обратного распространения ошибки) на части данных, предназначенной для построения модели, и тестируя точность на валидационной части данных. Подумайте, почему результирующая модель, скорее всего, продемонстрирует для тестовых данных, не входящих в тренировочный набор, худшую точность, чем для валидационных данных, несмотря на то что валидационные данные не использовались для обучения параметров. Имеются ли у вас какие-либо рекомендации для Боба относительно использования результатов его 1000 валидационных тестов?

10. Повышается ли точность классификации на тренировочных данных с увеличением их объема? А что можно сказать о поточечном усреднении функции потерь на тренировочных примерах? В какой точке полученные в процессе тренировки и тестирования значения точности сблизятся? Объясните свой ответ.
11. Какое влияние на точность тренировки и тестирования оказывает увеличение параметра регуляризации? В какой точке полученные в процессе тренировки и тестирования значения в точности сблизятся?

Глава 5

Сети радиально-базисных функций

Пока две птицы спорили за зернышко, прилетела третья и склевала его.

Африканская пословица

5.1. Введение

Сети *радиально-базисных функций* (radial basis function — RBF) представляют архитектуру, существенно отличающуюся от тех архитектур, с которыми мы познакомились в предыдущих главах. Во всех предыдущих главах использовалась сеть прямого распространения (feed-forward), в которой выходы создаются путем последовательной передачи входов из слоя в слой в прямом направлении. Сеть прямого распространения может иметь несколько слоев, и в типичных случаях нелинейность создается за счет повторной композиции функций активации. С другой стороны, RBF-сеть включает, как правило, только входной слой, одиночный скрытый слой (со специальным типом поведения, определяемым RBF-функциями) и выходной слой. Несмотря на принципиальную возможность замены выходного слоя несколькими слоями прямого распространения (как в сверточной сети), результирующая сеть все еще остается довольно мелкой, и ее поведение в значительной мере определяется природой специального скрытого слоя. Чтобы не усложнять обсуждение, мы будем работать только с одиночным выходным слоем. Как и в случае сетей прямого распространения, входной слой в действительности не является вычислительным и лишь передает входы в прямом направлении. Природа вычислений в скрытом слое сильно отличается от той, с которой мы до сих пор встречались в сетях прямого распространения. В частности, скрытый слой выполняет вычисления на основе сравнения с *вектором прототипов* (prototype vector), точного аналога которого в сетях прямого распространения не существует. Возможности RBF-сети определяются структурой специального скрытого слоя и выполняемыми в нем вычислениями.

Различия между функциями скрытого и выходного слоев заключаются в следующем.

1. Скрытый слой получает входные точки, структура классов которых может не быть линейно разделимой, и преобразует их в новое пространство, которое (нередко) допускает линейное разделение классов. Скрытый слой зачастую имеет более высокую размерность, чем входной, поскольку для обеспечения линейной разделимости часто требуется преобразование в многомерное пространство более высокой размерности. Этот принцип базируется на *теореме Ковера о разделимости образов* [84], которая утверждает, что нелинейное преобразование задач классификации образов в пространство более высокой размерности повышает вероятность линейной разделимости образов. Кроме того, большую вероятность линейной разделимости обеспечивают определенные типы преобразований, в которых признаки представляют небольшие локальные области пространства. Несмотря на то что размерность скрытого слоя обычно превышает размерность входного, она всегда меньше или равна количеству обучающих точек. Можно показать, что предельный случай, в котором размерность скрытого слоя совпадает с количеством обучающих точек, примерно эквивалентен обучению с использованием *ядерных методов* (kernel learners). Примерами таких моделей могут служить *ядерная регрессия* (kernel regression) и *ядерный метод опорных векторов* (kernel support vector machines).
2. В отношении входов, поступающих от скрытого слоя, выходной слой использует линейную классификацию или регрессионное моделирование. С соединениями, связывающими скрытый слой с выходным, ассоциируются веса. Вычисления в выходном слое выполняются точно так же, как и в стандартной сети прямого распространения. И хотя вместо выходного слоя могут использоваться несколько слоев прямого распространения, мы для простоты ограничимся случаем одного такого слоя.

Точно так же, как перцептрон является вариантом линейного метода опорных векторов, RBF-сети являются обобщением ядерных методов классификации и регрессии. Специальные варианты RBF-сетей могут использоваться для реализации ядерной регрессии, ядерной классификации по методу наименьших квадратов и ядерного метода опорных векторов. Различия между этими специальными случаями формулируются в терминах того, как структурированы скрытый слой и функция потерь. В сетях прямого распространения усиление нелинейности обеспечивается за счет увеличения глубины сети. Однако в RBF-сети, в силу ее особой структуры, для достижения требуемого уровня нелинейности обычно хватает одного скрытого слоя. Как и сети прямого распространения, RBF-сети являются универсальными аппроксиматорами функций.

Отдельные слои RBF-сети имеют следующее назначение.

1. Входной слой просто передает входные признаки скрытым слоям. Поэтому количество входных элементов в точности совпадает с размерностью данных d . Как и в случае сетей прямого распространения, во входных слоях не выполняются вычисления. Как и во всех сетях прямого распространения, все входные элементы связаны со всеми элементами скрытого слоя и передают свой вход в прямом направлении.
2. Выполняемые в скрытых слоях вычисления базируются на сравнениях с *вектором прототипов* (prototype vector). Каждый скрытый элемент содержит d -мерный вектор прототипов. Обозначим через $\bar{\mu}_i$ вектор прототипов i -го скрытого элемента, а через σ_i — полосу пропускания i -го элемента. Несмотря на то что векторы прототипов всегда специфичны для конкретных элементов, для полос пропускания различных элементов σ_i часто уравнивают одно и то же значение σ . Обычно векторы прототипов и полосы пропускания обучаются без учителя или под мягким контролем.

Затем для любой входной тренировочной точки \bar{X} и i -го скрытого элемента определяется функция активации $\Phi_i(\bar{X})$:

$$h_i = \Phi_i(\bar{X}) = \exp\left(-\frac{\|\bar{X} - \bar{\mu}_i\|^2}{2 \cdot \sigma_i^2}\right) \quad \forall i \in \{1, \dots, m\}, \quad (5.1)$$

где m — общее количество скрытых элементов. Каждый из этих скрытых элементов предназначен для того, чтобы оказывать существенное влияние на конкретный кластер точек, ближайших к его вектору прототипов. Поэтому m можно рассматривать как количество кластеров, используемых для моделирования, и считать его важным гиперпараметром, доступным для алгоритма. Для входов с низкой размерностью типичные значения m превышают размерность входа d , но меньше количества тренировочных точек n .

3. Пусть h_i — выход i -го элемента, определяемый для любой конкретной точки данных \bar{X} уравнением 5.1. Веса связей, соединяющих скрытые узлы с выходными, установлены в w_i . Тогда предсказание \hat{y} , создаваемое в выходном слое RBF-сети, определяется следующей формулой:

$$\hat{y} = \sum_{i=1}^m w_i h_i = \sum_{i=1}^m w_i \Phi_i(\bar{X}) = \sum_{i=1}^m w_i \exp\left(-\frac{\|\bar{X} - \bar{\mu}_i\|^2}{2 \cdot \sigma_i^2}\right).$$

Переменная \hat{y} выделена символом циркумфлекса, указывающим на то, что это предсказанное, а не наблюдаемое значение. Если наблюдаемое

целевое значение — вещественное, то можно ввести функцию потерь метода наименьших квадратов, имеющую в целом тот же вид, что и в случае сети прямого распространения. Веса $w_1 \dots w_m$ должны обучаться с использованием метода обучения с учителем.

Дополнительным нюансом является то, что скрытый слой нейронной сети содержит нейроны смещения. Отметим, что нейрон смещения может быть реализован единственным скрытым элементом, который всегда подключен к выходу. Этот тип нейрона также можно реализовать, создав скрытый элемент, для которого значение σ_i устанавливается равным ∞ . В любом случае до конца главы будет предполагаться, что этот скрытый элемент входит в число m скрытых элементов. Поэтому в использовании какого-либо особого способа для его обработки нет никакой необходимости. Пример RBF-сети приведен на рис. 5.1.

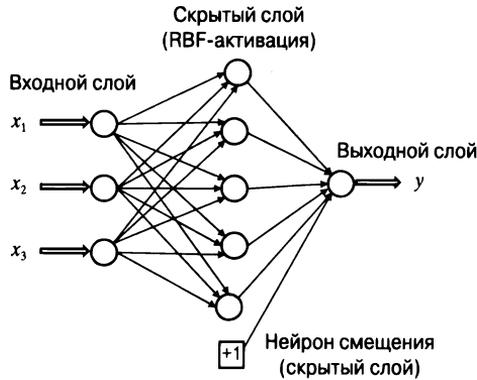


Рис. 5.1. RBF-сеть. Обратите внимание на то, что типичный скрытый слой шире входного, однако это не является обязательным требованием

В RBF-сети выполняются два вида вычислений, соответствующих скрытому и выходному слоям. Параметры $\bar{\mu}_i$ и σ_i скрытого слоя обучаются с учителем, тогда как обучение этих же параметров, относящихся к выходному слою, осуществляется без учителя посредством градиентного спуска, как и в случае сетей прямого распространения. Прототипы $\bar{\mu}_i$ могут либо семплироваться из данных, либо устанавливаться в значения m центроидов алгоритма m -кратной кластеризации. Последнее решение относится к числу наиболее часто используемых. Различные способы обучения нейронной сети обсуждаются в разделе 5.2.

Можно показать, что RBF-сети являются непосредственным обобщением класса ядерных методов. В первую очередь это обусловлено тем, что получаемое в выходном слое прогнозируемое значение, как можно показать, эквивалентно взвешенной оценке, получаемой методом ближайших соседей с весами, представляющими собой произведения коэффициентов w_i и показателей сходства

гауссовских RBF-функций с *прототипами*. Также можно показать, что функции предсказания почти всех ядерных методов эквивалентны функциям для получения взвешенных оценок по методу ближайших соседей, когда веса обучаются с учителем. Поэтому ядерные методы являются частным случаем RBF-методов, в которых количество скрытых узлов равно количеству тренировочных точек и все параметры σ_i имеют одно и то же значение. Это дает основания полагать, что по своим возможностям и гибкости RBF-сети превосходят ядерные методы (связь между ними подробно обсуждается в разделе 5.4).

5.1.1. Когда следует использовать RBF-сети

Важно понимать, что скрытый слой RBF-сети создается с использованием обучения без учителя, чтобы повысить ее устойчивость к шумам любого типа. Это свойство RBF-сетей разделяет и метод опорных векторов. В то же время существуют ограничения в отношении того, какой части структуры данных способна обучиться RBF-сеть. Глубокие сети прямого распространения доказали свою эффективность в обучении на данных с богатой структурой, поскольку наличие множества слоев нелинейности вынуждает данные следовать определенным типам шаблонов. Кроме того, настраивая структуру соединений в сетях прямого распространения, можно учесть особенности, специфические для конкретной предметной области. В качестве примера можно привести рекуррентные и сверточные нейронные сети. Наличие всего одного слоя в RBF-сети ограничивает долю структуры, которой можно обучиться. И хотя известно, что как RBF-сети, так и сети прямого распространения являются универсальными аппроксиматорами функций, они различаются своей способностью обобщаться на различные типы наборов данных.

Структура главы

В следующем разделе обсуждаются различные методы тренировки RBF-сетей. Об использовании RBF-сетей в задачах классификации и регрессии речь пойдет в разделе 5.3. Связь RBF-метода с ядерной регрессией и классификацией описывается в разделе 5.4. Резюме главы содержится в разделе 5.5.

5.2. Тренировка RBF-сети

Тренировка RBF-сети значительно отличается от тренировки сети прямого распространения, которая полностью интегрируется по различным слоям. В RBF-сети тренировка скрытого слоя обычно осуществляется по методике без учителя. Несмотря на принципиальную возможность использования обратного распространения для обучения векторов прототипов и полос пропускания, проблема заключается в том, что на поверхности функции потерь RBF-сетей

существует больше локальных минимумов по сравнению с сетями прямого распространения. Поэтому если скрытый слой и обучается с учителем, то это часто делается в сравнительно мягкой манере или же ограничивается лишь тонкой настройкой уже обученных весов. Тем не менее, поскольку переобучение является, по-видимому, распространенной проблемой в случае тренировки скрытого слоя по методике обучения с учителем, наше обсуждение будет ограничено методами обучения без учителя. Ниже мы сначала обсудим тренировку скрытого слоя RBF-сети, а затем скрытого слоя.

5.2.1. Тренировка скрытого слоя

Скрытый слой RBF-сети содержит ряд параметров, в том числе векторы прототипов $\bar{\mu}_1 \dots \bar{\mu}_m$ и полосы пропускания $\sigma_1 \dots \sigma_m$. Гиперпараметр m управляет количеством скрытых элементов. На практике для разных элементов устанавливаются не отдельные значения σ_i , а одно и то же значение полосы пропускания σ . Однако средние значения $\bar{\mu}_i$ для различных скрытых элементов различны, поскольку они определяют крайне важные векторы прототипов. Сложность модели регулируется количеством скрытых элементов и полосой пропускания. Сочетание небольшой полосы пропускания с большим количеством скрытых элементов увеличивает сложность модели и полезно в случае значительного объема данных. Для предотвращения переобучения в случае небольших наборов данных требуется меньшее количество элементов и более широкие полосы пропускания. Обычно значение m превышает размерность входных данных, но никогда не может быть больше количества тренировочных точек. Приравнивание значения m к количеству тренировочных точек и использование каждой тренировочной точки в качестве прототипа в скрытом узле делает этот подход эквивалентным традиционным ядерным методам.

Полоса пропускания также зависит от выбранных векторов прототипов $\bar{\mu}_1 \dots \bar{\mu}_m$. В идеальном случае полосы пропускания должны устанавливаться такими, чтобы на каждую точку могло оказывать ощутимое влияние лишь небольшое количество векторов прототипов, которые соответствуют его ближайшим кластерам. Установка слишком большой или слишком малой полосы пропускания по сравнению с расстоянием между прототипами приведет к недообучению или переобучению соответственно. Пусть d_{max} — максимальное расстояние между парами центров прототипов, а d_{ave} — среднее расстояние между ними. Получили распространение следующие два эвристических способа задания значения полосы пропускания:

$$\sigma = \frac{d_{max}}{\sqrt{m}},$$

$$\sigma = 2 \cdot d_{ave}.$$

С обоими вариантами выбора σ связана одна проблема, которая заключается в том, что полоса пропускания может быть различной в различных областях входного пространства. Например, полоса пропускания в плотной области пространства данных должна быть меньше полосы пропускания в разреженной области. Кроме того, полоса пропускания зависит также от распределения векторов прототипов в этом пространстве. Следовательно, одним из возможных решений является выбор такой полосы пропускания σ_i для i -го вектора прототипа, чтобы она была равна расстоянию, на котором он находится от его r -го ближайшего соседа среди прототипов. Здесь r — небольшое значение, равное, скажем, 5 или 10.

Однако это всего лишь эмпирические правила. Также возможна тонкая настройка значений с использованием отложенной части набора данных. Иными словами, вероятные значения σ генерируются в окрестности рекомендованных выше значений σ (которые служат начальной точкой отсчета). Затем с помощью этих значений-кандидатов σ конструируется несколько моделей (включая тренировку выходного слоя). Используется тот выбор σ , который обеспечивает наименьшую ошибку на отложенной части тренировочных данных. В подходе такого типа выбор обучения при выборе значения полосы пропускания в определенной степени контролируется, но без риска “застрять” в локальном минимуме. В то же время такое контролируемое обучение осуществляется довольно мягко, что особенно важно при работе с параметрами первого слоя RBF-сети. Следует отметить, что этот тип настройки полосы пропускания применяется также в случае использования гауссовского ядра в ядерном методе опорных векторов. Отмеченное сходство — не простое совпадение, поскольку ядерный метод опорных векторов является специальным случаем RBF-сетей (раздел 5.4).

Выбор векторов прототипов чуть более сложен. В частности, наиболее распространенными вариантами выбора являются следующие.

1. Векторы прототипов могут случайно семплироваться из n тренировочных точек. Для создания векторов прототипов используются $m < n$ тренировочных точек. Основной проблемой этого подхода является то, что он будет предоставлять слишком большое количество прототипов из плотных областей данных, тогда как разреженные области будут представлены недостаточно полно или вообще не представлены. В результате предсказательная точность будет страдать.
2. Для создания m кластеров можно использовать алгоритм кластеризации k -средних. Центроид каждого из этих m кластеров может использоваться в качестве прототипа. Алгоритм k -средних — наиболее распространенный вариант выбора способа обучения векторов прототипов.
3. Также используются варианты алгоритмов кластеризации, которые делят *пространство* данных (а не точки). В качестве конкретного примера

можно привести использование деревьев решений для создания прототипов.

4. Альтернативным методом тренировки скрытого слоя является использование *ортогонального алгоритма наименьших квадратов*. Этот подход предполагает частичное использование обучения с учителем. Векторы прототипов поочередно выбираются из тренировочного набора данных для минимизации остаточной ошибки на тестовом наборе, не входящем в тренировочные данные. Поскольку обсуждение этого подхода требует понимания того, как тренируется выходной слой, оно будет приведено в одном из последующих разделов.

Далее мы вкратце опишем применение алгоритма k -средних для создания прототипов, поскольку это наиболее распространенный выбор в приложениях. Алгоритм k -средних — это классическая методика, описанная в литературе по кластеризации. Прототипы кластеров используются в ней так же, как прототипы для скрытого слоя используются в методе RBF. В общих чертах алгоритм k -средних работает следующим образом. На стадии инициализации значения m кластерных прототипов устанавливаются в m случайных тренировочных точек. Впоследствии каждая из n точек данных приписывается к прототипу, от которого она удалена на минимальное евклидово расстояние. Назначенные каждому прототипу точки усредняются для создания нового центра кластера. Иными словами, центроид созданного кластера используется для замены его старого прототипа новым. Этот процесс итеративно повторяется до достижения сходимости. Сходимость достигается тогда, когда изменения назначаемых точек при переходе от одной итерации к другой становятся незначительными.

5.2.2. Тренировка выходного слоя

Тренировка выходного слоя осуществляется после того, как обучен скрытый слой. Она не представляет трудностей, поскольку ее объектом является один слой с линейной активацией. Чтобы упростить обсуждение, рассмотрим сначала случай, когда целевые значения выходного слоя являются вещественными. Другие варианты будут рассмотрены позже. Выходной слой содержит m -мерный вектор весов $\bar{W} = [w_1 \dots w_m]$, подлежащих обучению. Предположим, что вектор \bar{W} является вектор-строкой.

Рассмотрим ситуацию, когда тренировочный набор данных содержит n точек $\bar{X}_1 \dots \bar{X}_m$, создающих в скрытом слое представление $\bar{H}_1 \dots \bar{H}_m$. Таким образом, каждый \bar{H}_i — это m -мерный вектор. Совокупность этих n вектор-строк можно представить в виде матрицы H размера $n \times m$. Кроме того, наблюдаемые значения n тренировочных точек y_1, y_2, \dots, y_n можно записать в виде n -мерного вектор-столбца: $\bar{y} = [y_1 \dots y_n]^T$.

Предсказания для n тренировочных точек даются элементами n -мерного вектор-столбца $H\bar{W}^T$. В идеале нам хотелось бы, чтобы эти предсказания были как можно более близкими к вектору наблюдаемых значений \bar{y} . Следовательно, функция потерь L , которая будет использоваться для обучения весов выходного слоя, может быть записана в следующем виде:

$$L = \frac{1}{2} \|H\bar{W}^T - \bar{y}\|^2.$$

Для уменьшения эффектов переобучения можно добавить в целевую функцию регуляризацию Тихонова:

$$L = \frac{1}{2} \|H\bar{W}^T - \bar{y}\|^2 + \frac{\lambda}{2} \|\bar{W}\|^2, \quad (5.2)$$

где $\lambda > 0$ — параметр регуляризации. Вычисляя частную производную L по элементам вектора весов, получаем следующее соотношение:

$$\frac{\partial L}{\partial \bar{W}} = H^T (H\bar{W}^T - \bar{y}) + \lambda \bar{W}^T = 0.$$

В этом выражении использованы матричные обозначения, в соответствии с которыми запись $\frac{\partial L}{\partial \bar{W}}$ означает следующее:

$$\frac{\partial L}{\partial \bar{W}} = \left(\frac{\partial L}{\partial w_1} \dots \frac{\partial L}{\partial w_d} \right)^T. \quad (5.3)$$

Перегруппировав члены в приведенном выше условии, получаем

$$(H^T H + \lambda I) \bar{W}^T = H^T \bar{y}.$$

При $\lambda > 0$ матрица $H^T H + \lambda I$ является положительно определенной и поэтому имеет обратную матрицу. Иными словами, мы получаем для вектора весов следующее простое решение в замкнутой форме:

$$\bar{W}^T = (H^T H + \lambda I)^{-1} H^T \bar{y}. \quad (5.4)$$

Таким образом, для нахождения вектора весов достаточно вычислить некую обратную матрицу, и в этом случае необходимость в использовании алгоритма обратного распространения отпадает.

Однако реалии таковы, что попытка получения решения в замкнутой форме на практике наталкивается на значительные трудности из-за размера $m \times m$ матрицы $H^T H$, который может быть очень большим. Например, в ядерных методах мы устанавливаем $m = n$, что затрудняет обработку самой матрицы, не говоря о ее обращении. Поэтому на практике для обновления вектора весов приходится использовать стохастический градиентный спуск. В данном случае

обновления по методу градиентного спуска (для всех тренировочных точек) записываются в следующем виде:

$$\begin{aligned}\bar{W}^T &\leftarrow \bar{W}^T - \alpha \frac{\partial L}{\partial \bar{W}} = \\ &= \bar{W}^T (1 - \alpha \lambda) - \alpha H^T \underbrace{(H\bar{W}^T - \bar{y})}_{\text{Текущие ошибки}}.\end{aligned}$$

Другой возможный вариант заключается в использовании мини-пакетного градиентного спуска, при котором матрица H в приведенном выше выражении для обновлений заменяется случайным подмножеством ее строк H_r , соответствующих мини-пакету. Такой подход эквивалентен мини-пакетному стохастическому градиентному спуску, используемому в традиционных нейронных сетях. Однако в данном случае он применяется только к весам входящих соединений выходного слоя.

5.2.2.1. Выражение через псевдообратную матрицу

В том случае, когда параметр регуляризации λ устанавливается равным нулю, вектор весов \bar{W} определяется следующим образом:

$$\bar{W}^T = (H^T H)^{-1} H^T \bar{y}. \quad (5.5)$$

Матрица $(H^T H)^{-1} H^T$ называется *псевдообратной* к матрице H и обозначается как H^+ . Таким образом, вектор весов \bar{W}^T можно записать в следующем виде:

$$\bar{W}^T = H^+ \bar{y}. \quad (5.6)$$

Понятие псевдообращения матриц является обобщением понятия обращения матриц на случай несингулярных или прямоугольных матриц. В данном конкретном случае предполагается, что матрица $H^T H$ обращается, хотя псевдообратную к H матрицу можно вычислять даже в тех случаях, когда это не так. Если матрица H — квадратная, то ее псевдообратная матрица совпадает с обратной.

5.2.3. Ортогональный метод наименьших квадратов

Вернемся к рассмотрению фазы тренировки скрытого слоя. В подходе, который обсуждается в данном разделе, при выборе прототипов будут использоваться предсказания в выходном слое. Поэтому процесс тренировки скрытого слоя осуществляется контролируемо, т.е. по методике обучения с учителем, хотя этот контроль ограничивается итеративным выбором из исходных точек тренировочного набора. Ортогональный метод наименьших квадратов минимизирует

ошибку предсказаний, поочередно выбирая по одному вектору прототипов из тренировочных точек.

Данный алгоритм начинается с создания RBF-сети, содержащей один скрытый слой, и попыток использования каждой возможной тренировочной точки в качестве прототипа для вычисления ошибки предсказания. После этого выбирается прототип, который минимизирует ошибку предсказания. На следующей итерации выбранный прототип дополняется другим прототипом для создания RBF-сети с двумя прототипами. Как и на предыдущей итерации, в качестве кандидатов для добавления в текущий набор прототипов испытываются все оставшиеся $(n - 1)$ точек, критерием отбора которых в текущий набор является минимизация ошибки предсказания. Во время выполнения $(r + 1)$ -й итерации испытываются все $(n - r)$ оставшихся тренировочных точек, и в текущий набор прототипов добавляется та из них, которая минимизирует ошибку предсказания. Некоторые из точек резервируются и не используются для вычислений или в качестве кандидатов для включения в состав прототипов. Эти не входящие в обучающий набор точки используются для тестирования влияния добавляемого прототипа на ошибку. На определенном этапе ошибка на отложенном наборе начинает возрастать по мере добавления прототипов. В этот момент работа алгоритма прекращается.

Основной проблемой данного подхода является его крайняя неэффективность. На каждой итерации процедура тренировки должна выполняться n раз, что в случае крупных наборов тренировочных данных требует проведения чрезвычайно интенсивных вычислений. В этом отношении представляет интерес *ортогональный метод наименьших квадратов* (orthogonal least-squares algorithm) [65], доказавший свою эффективность. Данный алгоритм аналогичен вышеописанному в том смысле, что добавляемые векторы прототипов итеративно извлекаются из исходного тренировочного набора данных. Однако для добавления извлекаемых прототипов в их текущий набор используется намного более эффективная процедура. Из тренировочных точек формируется набор ортогональных векторов, которые принадлежат пространству, натянутому на активации скрытых элементов. Эти ортогональные векторы можно использовать для того, чтобы определить, какой именно прототип должен быть выбран из тренировочного набора данных.

5.2.4. Полностью контролируемое обучение

Ортогональный алгоритм наименьших квадратов представляет собой тип обучения с учителем, характеризующийся мягким контролем процесса обучения, в соответствии с которым в качестве вектора прототипов выбирается одна из точек тренировочного набора, а критерием отбора служит минимизация общей ошибки предсказания. Однако возможны и другие варианты, предполагающие

более строгий контроль над процессом обучения, в соответствии с которыми для обновления векторов прототипов и полосы пропускания используется алгоритм обратного распространения ошибки. Рассмотрим функцию потерь L , вычисляемую по всем тренировочным точкам:

$$L = \frac{1}{2} \sum_{i=1}^n (\bar{H}_i \cdot \bar{W} - y_i)^2, \quad (5.7)$$

где \bar{H}_i представляет m -мерный вектор активаций в скрытом слое для тренировочной точки \bar{X}_i .

Частная производная по каждой полосе пропускания σ_j вычисляется так:

$$\begin{aligned} \frac{\partial L}{\partial \sigma_j} &= \sum_{i=1}^n (\bar{H}_i \cdot \bar{W} - y_i) w_j \frac{\partial \Phi_j(\bar{X}_i)}{\partial \sigma_j} = \\ &= \sum_{i=1}^n (\bar{H}_i \cdot \bar{W} - y_i) w_j \Phi_j(\bar{X}_i) \frac{\|\bar{X}_i - \bar{\mu}_j\|^2}{\sigma_j^3}. \end{aligned}$$

Если для всех полос пропускания σ_j установлено одно и то же значение σ , как это принято в RBF-сетях, то для вычисления производной можно использовать тот же трюк, что и при обработке разделяемых весов:

$$\begin{aligned} \frac{\partial L}{\partial \sigma} &= \sum_{j=1}^m \frac{\partial L}{\partial \sigma_j} \cdot \underbrace{\frac{\partial \sigma_j}{\partial \sigma}}_{=1} = \\ &= \sum_{j=1}^m \frac{\partial L}{\partial \sigma_j} = \\ &= \sum_{j=1}^m \sum_{i=1}^n (\bar{H}_i \cdot \bar{W} - y_i) w_j \Phi_j(\bar{X}_i) \frac{\|\bar{X}_i - \bar{\mu}_j\|^2}{\sigma_j^3}. \end{aligned}$$

Мы также можем вычислить частную производную по каждому элементу вектора прототипов. Пусть μ_{jk} представляет k -й элемент $\bar{\mu}_j$, а x_{ik} — k -й элемент i -й тренировочной точки \bar{X}_i . Тогда частную производную по μ_{jk} можно вычислить следующим образом:

$$\frac{\partial L}{\partial \mu_{jk}} = \sum_{i=1}^n (\bar{H}_i \cdot \bar{W} - y_i) w_j \Phi_j(\bar{X}_i) \frac{(x_{ik} - \mu_{jk})}{\sigma_j^2}. \quad (5.8)$$

Используя эти частные производные, можно обновить полосу пропускания и векторы прототипов вместе с весами. К сожалению, такой тип строгого контроля процесса обучения не может считаться удачным, поскольку ему свойственны следующие недостатки.

1. Привлекательность RBF-сетей обусловлена эффективностью процесса их тренировки, когда используются методы обучения без учителя. Даже ортогональный метод наименьших квадратов может быть выполнен в течение разумного промежутка времени. Но как только мы прибегаем к полноценному обратному распространению ошибки, это преимущество теряется. В целом эффективным способом обучения RBF-сетей является их двухэтапная тренировка.
2. Поверхность функции потерь в RBF-сетях характеризуется наличием множества локальных минимумов. С точки зрения ошибок обобщения такой подход может часто приводить к “застреванию” в одном из локальных минимумов.

С учетом этих свойств RBF-сетей их тренировка по методике обучения с учителем применяется редко. В действительности было показано [342], что попытки контролировать процесс обучения приводят к увеличению полосы пропускания и получению слишком общих ответов. Если такой контроль и применяется, то он должен осуществляться очень тщательно и сопровождаться постоянным тестированием производительности на данных, которые не использовались в процессе тренировки, чтобы свести к минимуму риск переобучения.

5.3. Разновидности и специальные случаи RBF-сетей

Проведенное выше обсуждение относится лишь к случаю, когда обучение с учителем ориентировано на числовые целевые переменные. На практике целевые переменные могут быть бинарными. Одна из возможностей заключается в том, чтобы рассматривать бинарные метки классов $\{-1, +1\}$ как числовые ответы и использовать тот же подход к установке вектора весов в соответствии с уравнением 5.4:

$$\bar{W}_T = (H^T H + \lambda I)^{-1} H^T \bar{y}.$$

Как обсуждалось в разделе 2.2.2.1, это решение эквивалентно дискриминанту Фишера и методу Уидроу — Хоффа. Основное отличие состоит в том, что эти методы применяются к скрытому слою увеличенной размерности, что способствует получению лучших результатов для более сложных распределений. Представляет интерес исследовать и другие функции потерь, которые обычно используются в сетях прямого распространения для классификации.

5.3.1. Классификация с использованием критерия перцептрона

Используя обозначения, введенные в предыдущем разделе, мы можем записать предсказание для i -го тренировочного примера в виде $\bar{W} \cdot \bar{H}_i$. Здесь \bar{H}_i представляет m -мерный вектор активаций в скрытом слое для i -го тренировочного примера \bar{X}_i . Тогда (см. раздел 1.2.1.1) критерий перцептрона соответствует следующей функции потерь:

$$L = \max\{-y_i(\bar{W} \cdot \bar{H}_i), 0\}. \quad (5.9)$$

Кроме того, к функции потерь часто добавляют регуляризацию Тихонова с параметром $\lambda > 0$.

Пусть S представляет каждый мини-пакет тренировочных примеров, а S^+ — неверно классифицированные примеры. Под неверно классифицированными примерами понимаются такие, для которых функция потерь L не равна нулю. Для таких примеров применение функции sign к произведению $\bar{H}_i \cdot \bar{W}$ будет давать предсказание, знак которого противоположен знаку наблюдаемой метки y_i .

Тогда в каждом мини-пакете S тренировочных примеров в отношении классифицированных примеров S^+ используются следующие обновления:

$$\bar{W} \leftarrow \bar{W}(1 - \alpha\lambda) + \alpha \sum_{(\bar{H}_i, y_i) \in S^+} y_i \bar{H}_i, \quad (5.10)$$

где $\alpha > 0$ — скорость обучения.

5.3.2. Классификация с кусочно-линейной функцией потерь

Кусочно-линейная функция потерь (hinge loss) часто используется в методе опорных векторов. В действительности использование кусочно-линейных потерь в гауссовской RBF-сети можно рассматривать как обобщение метода опорных векторов. Кусочно-линейные потери — это смещенная версия критерия перцептрона:

$$L = \max\{-y_i(\bar{W} \cdot \bar{H}_i), 0\}. \quad (5.11)$$

Следствием сходства кусочно-линейной функции потерь и критерия перцептрона является близкое сходство соответствующих обновлений. Основное различие состоит в том, что в случае критерия перцептрона набор S^+ включает лишь неверно классифицируемые примеры, тогда как в случае кусочно-линейной функции потерь набор S^+ включает как неверно классифицированные точки, так и точки, верно классифицированные с учетом зазора. Это обусловлено тем, что S^+ определяется как множество точек, для которых функция потерь имеет ненулевое значение, но (в отличие от критерия перцептрона)

кусочно-линейная функция потерь имеет ненулевое значение даже для верно классифицированных точек с учетом зазора. В отношении такого набора S^+ используются следующие обновления:

$$\bar{W} \leftarrow \bar{W}(1 - \alpha\lambda) + \alpha \sum_{(\bar{H}_i, y_i) \in S^+} y_i \bar{H}_i, \quad (5.12)$$

где $\alpha > 0$ — скорость обучения, а $\lambda > 0$ — параметр регуляризации. Отметим, что аналогичные обновления несложно определить также для логистической функции потерь (см. упражнение 2).

5.3.3. Пример линейной разделимости, обеспечиваемой RBF-функциями

Основная задача скрытого слоя заключается в выполнении преобразования, способствующего линейной разделимости классов до такой степени, что даже линейные классификаторы могут хорошо работать на преобразованных данных. Известно, что как перцептрон, так и линейный метод опорных векторов плохо работают в случае линейно неразделимых классов. Гауссовский RBF-классификатор способен разделять классы, не являющиеся линейно разделимыми в пространстве входа, если в качестве функции потерь используется критерий перцептрона или кусочно-линейная функция потерь. Ключом к пониманию этой разделимости является локальное преобразование, создаваемое скрытым слоем. Важно то, что использование гауссовского ядра с небольшой полосой пропускания часто приводит к ситуациям, когда лишь небольшое количество скрытых элементов в отдельных локальных областях активируется до существенных ненулевых значений, в то время как другие элементы почти равны нулю. Это обусловлено экспоненциально затухающей природой гауссовской функции, которая принимает почти нулевые значения за пределами некоторой области. Идентификация прототипов с помощью центров кластеров часто делит пространство на локализованные области, в которых значительная ненулевая активация достигается лишь в небольших участках пространства. На практике каждой локальной области пространства присваивается собственный признак, соответствующий скрытому элементу, который активируется им наиболее интенсивно.

На рис. 5.2 приведены примеры двух наборов данных. Эти наборы были представлены в главе 1 в качестве иллюстрации случаев, в которых (традиционный) перцептрон может или не может обеспечить решение. Традиционный перцептрон способен найти решение для набора данных, представленного слева, но плохо справляется со своей задачей в отношении набора, представленного справа. Однако преобразование, используемое гауссовским RBF-методом, способно справиться с разделимостью для набора кластеризованных данных, представленного справа. Рассмотрим случай, когда каждый из центроидов

четырёх кластеров (рис. 5.2) используется в качестве прототипа. Это приведет к 4-мерному скрытому представлению данных. Обратите внимание на то, что размерность скрытого представления превышает размерность входа, что типично для таких ситуаций. При надлежащем выборе полосы пропускания лишь скрытый элемент, соответствующий идентификатору кластера, к которому принадлежит точка, будет активироваться наиболее сильно. Другие скрытые элементы будут активироваться слабо, оставаясь практически на нулевом уровне. Это приводит к довольно разреженному представлению. На рисунке приведено приближенное 4-мерное представление для точек каждого кластера. Значения a , b , c и d на рис. 5.2 будут различными для различных точек соответствующего кластера, хотя они всегда будут иметь существенно большие ненулевые значения по сравнению с другими координатами. Обратите внимание на то, что один из классов определен существенно ненулевыми значениями в первом и третьем измерениях, тогда как второй класс определяется существенно ненулевыми значениями во втором и четвертом измерениях. Как следствие, вектор весов $\bar{W} = [1, -1, 1, -1]$ обеспечит отличное нелинейное разделение двух классов. Важно понимать, что гауссовская RBF-функция создает локальные признаки, приводящие к разделимым распределениям классов. Именно так ядерный метод опорных векторов обеспечивает линейную разделимость.

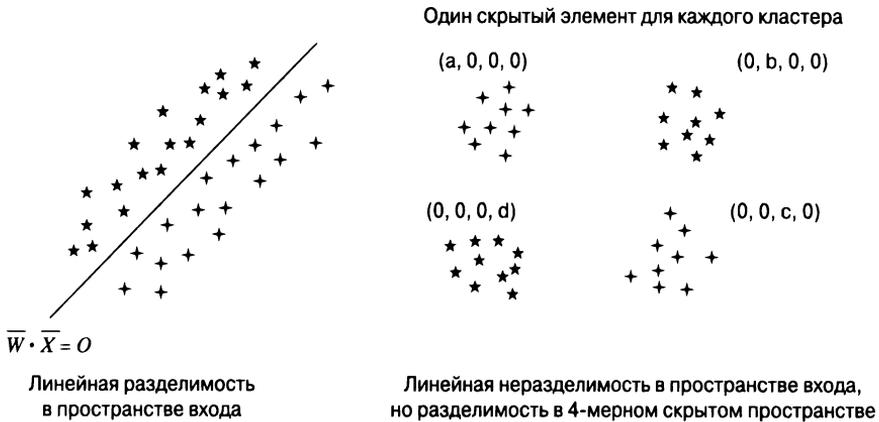


Рис. 5.2. Пересмотр рис. 1.4; гауссовские RBF-функции способствуют разделимости за счет преобразования данных в скрытое представление

5.3.4. Применение в задачах интерполяции

Одним из ранних применений гауссовской RBF-сети была интерполяция, т.е. нахождение промежуточных значений функции при заданном наборе известных значений. В данном случае речь идет о точной интерполяции, т.е. такой, когда результирующая функция проходит через все входные точки. Интерполя-

цию можно рассматривать как частный случай регрессии, в котором каждая тренировочная точка является прототипом, и поэтому количество весов m в матрице \bar{W} в точности равно количеству тренировочных примеров n . В подобных случаях можно найти n -мерный вектор весов \bar{W} с нулевой ошибкой. Пусть $\bar{H}_1 \dots \bar{H}_n$ — активации, представляющие n -мерные вектор-строки. Тогда матрица H , полученная наложением этих вектор-строк одна поверх другой, имеет размер $n \times n$. Обозначим через $\bar{y} = [y_1, y_2, \dots, y_n]^T$ n -мерный вектор-столбец наблюдаемых переменных.

В линейной регрессии для определения матрицы \bar{W} минимизируется функция потерь $\|H\bar{W}^T - \bar{y}\|^2$. Это обусловлено тем, что матрица H не является квадратной, в результате чего система уравнений $H\bar{W}^T = \bar{y}$ получается сверхполной. Однако в случае линейной интерполяции матрица H — квадратная, и система уравнений уже не является сверхполной. Благодаря этому можно найти точное решение (с нулевыми потерями), удовлетворяющее следующей системе уравнений:

$$H\bar{W}^T = \bar{y}. \quad (5.13)$$

Можно показать, что если тренировочные точки отличаются друг от друга, то данная система уравнений имеет единственное решение [323]. Тогда значение вектора весов \bar{W}^T можно вычислить следующим образом:

$$\bar{W}^T = H^{-1}\bar{y}. \quad (5.14)$$

Следует подчеркнуть, что это уравнение представляет собой специальный случай уравнения 5.6, поскольку матрица, псевдообратная к квадратной несингулярной матрице, — это то же самое, что обратная матрица. Если матрица H — сингулярная, то выражение для псевдообратной матрицы упрощается:

$$\begin{aligned} H^+ &= (H^T H)^{-1} H^T = \\ &= H^{-1} \underbrace{(H^T)^{-1} H^T}_I = \\ &= H^{-1}. \end{aligned}$$

Таким образом, линейная интерполяция является специальным случаем регрессии по методу наименьших квадратов. Иначе говоря, регрессия по методу наименьших квадратов — это вид интерполяции с шумом, когда точная подгонка функции, обеспечивающая ее прохождение через все тренировочные точки, невозможна из-за ограниченного числа степеней свободы в скрытом слое. Увеличение размерности скрытого слоя до размерности данных делает возможной точную интерполяцию. Это вовсе не обязательно является лучшим вариантом для вычисления значения функции в точках, не входящих в тренировочный набор, поскольку такая точность может быть следствием переобучения.

5.4. Связь с ядерными методами

RBF-сеть реализует свои возможности за счет отображения входных точек на многомерное скрытое пространство, в котором линейных моделей оказывается достаточно для моделирования нелинейностей. Это тот же принцип, который применяется в ядерных методах. В действительности можно показать, что некоторые специальные случаи RBF-сети сводятся к ядерной регрессии и ядерному методу опорных векторов.

5.4.1. Ядерная регрессия как специальный случай RBF-сетей

Вектор \bar{W} в RBF-сетях обучается минимизации квадратичных потерь для следующей функции предсказания:

$$\hat{y}_i = \bar{H}_i \bar{W}^T = \sum_{j=1}^m w_j \Phi_j(\bar{X}_i). \quad (5.15)$$

Рассмотрим случай, когда прототипы совпадают с тренировочными точками, и поэтому $\bar{\mu}_j = \bar{X}_j$ для каждого $j \in \{1 \dots n\}$. Обратите внимание на то, что данный подход совпадает с тем, который используется при интерполяции функций, когда прототипами служат все тренировочные точки. Кроме того, для всех полос пропускания устанавливается одно и то же значение σ . В этом случае приведенная выше функция предсказания может быть записана в следующем виде:

$$\hat{y}_i = \sum_{j=1}^m w_j \exp\left(-\frac{\|\bar{X}_i - \bar{X}_j\|^2}{2\sigma^2}\right). \quad (5.16)$$

Экспоненциальный член в правой части уравнения 5.16 можно записать в виде гауссовского ядерного сходства между точками \bar{X}_i и \bar{X}_j . Обозначим это сходство как $K(\bar{X}_i, \bar{X}_j)$. Тогда функция предсказания приобретает следующий вид:

$$\hat{y}_i^{kernel} = \sum_{j=1}^m w_j K(\bar{X}_i, \bar{X}_j). \quad (5.17)$$

Эта функция предсказания в точности совпадает с той, которая используется в ядерной регрессии с полосой пропускания σ , где функция предсказания \hat{y}_i^{kernel} определяется¹ в терминах *множителей Лагранжа* λ , а не весов w_j (см., например, [6]):

¹ Полное объяснение использования уравнения 5.18 для получения предсказаний в ядерной регрессии выходит за рамки книги, однако заинтересованные читатели могут ознакомиться с ним в [6].

$$\hat{y}_i^{kernel} = \sum_{j=1}^m \lambda_j y_j K(\bar{X}_i, \bar{X}_j). \quad (5.18)$$

К тому же в обоих случаях используется одна и та же (квадратичная) функция потерь. Следовательно, между гауссовскими RBF-решениями и решениями ядерной регрессии существует взаимно однозначное соответствие, так что установка для весов значений $w_j = \lambda_j y_j$ приводит к аналогичному значению функции потерь. Поэтому их оптимальные значения также будут совпадать. Иными словами, в том специальном случае, когда векторы прототипов устанавливаются в тренировочные точки, гауссовская RBF-сеть дает те же результаты, что и ядерная регрессия. Однако RBF-сеть обладает более широкими возможностями, поскольку допускает выбор различных векторов прототипов и позволяет моделировать случаи, с которыми ядерная регрессия не в состоянии справиться. В этом смысле полезно рассматривать RBF-сеть как гибкий нейронный вариант ядерных методов.

5.4.2. Ядерный метод SVM как специальный случай RBF-сетей

Подобно ядерной регрессии, ядерный метод опорных векторов (SVM) также является специальным случаем RBF-сетей. Как и в случае ядерной регрессии, векторы прототипов устанавливаются в тренировочные точки, а для всех полос пропускания устанавливается одно и то же значение σ . Кроме того, веса w_j обучаются с целью минимизации кусочно-линейной функции потерь предсказаний.

Можно показать, что в этом случае функция предсказаний RBF-сети принимает следующий вид:

$$\hat{y}_i = \text{sign} \left\{ \sum_{j=1}^m w_j \exp \left(-\frac{\|\bar{X}_i - \bar{X}_j\|^2}{2\sigma^2} \right) \right\}, \quad (5.19)$$

$$\hat{y}_i = \text{sign} \left\{ \sum_{j=1}^m w_j K(\bar{X}_i, \bar{X}_j) \right\}. \quad (5.20)$$

Полезно сравнить эту функцию с той, которая используется в ядерном методе SVM (см., например, [6]) с множителями Лагранжа λ_j :

$$\hat{y}_i^{kernel} = \text{sign} \left\{ \sum_{j=1}^m \lambda_j y_j K(\bar{X}_i, \bar{X}_j) \right\}. \quad (5.21)$$

Эта функция предсказания имеет примерно ту же форму, что и в ядерном методе SVM, за исключением небольшого различия в используемых переменных. В обоих случаях в качестве целевой функции используется кусочно-линейная функция потерь. Установив значения $w_j = \lambda_j y_j$, мы получаем в обоих случаях одно и то же значение функции потерь. Поэтому оптимальные решения в ядерном методе SVM и в RBF-сети также будут связаны между собой в соответствии с условием $w_j = \lambda_j y_j$. Иными словами, ядерный метод SVM также является специальным случаем RBF-сетей. Обратите внимание на то, что веса w_j могут рассматриваться в качестве коэффициентов соответствующих точек данных, если в ядерных методах используется *теорема о представителе* [6].

5.4.3. Замечания

Изменив функцию потерь, приведенную выше аргументацию можно распространить на другие линейные модели, такие как ядерный дискриминант Фишера и ядерная логистическая регрессия. В действительности ядерный дискриминант Фишера можно получить за счет использования бинарных переменных в качестве целевых значений и последующего применения ядерной регрессии. Однако, поскольку дискриминант Фишера работает в предположении центрирования данных, в выходной слой необходимо добавить элемент смещения, поглощающий любое смещение, происходящее от нецентрированных данных. Поэтому RBF-сеть позволяет имитировать практически любой ядерный метод, если выбрать подходящую функцию потерь. Важно то, что RBF-сеть предлагает гораздо более гибкие возможности по сравнению с ядерной регрессией или классификацией. Например, она допускает более гибкий выбор количества узлов в скрытом слое, равно как и количества прототипов. Удачный экономичный выбор прототипов позволяет повысить как точность, так и производительность сети. При выборе возможных вариантов приходится принимать ряд компромиссных решений.

1. Увеличение количества скрытых узлов увеличивает сложность моделируемой функции. Это может быть полезным для моделирования трудных функций, но может приводить и к переобучению, если моделируемая функция не является действительно сложной.
2. Увеличение количества скрытых элементов усложняет процесс обучения.

Один из возможных способов выбора количества скрытых элементов заключается в резервировании (откладывании) части данных и оценке точности модели на этом отложенном наборе при различном количестве скрытых элементов. Далее для количества скрытых элементов устанавливается значение, которое оптимизирует эту точность.

5.5. Резюме

В этой главе были введены сети радиально-базисных функций (RBF), представляющие собой существенно иной способ использования архитектуры нейронных сетей. В отличие от сетей прямого распространения скрытый и выходной слои обучаются немного по-разному. Тренировка скрытого слоя осуществляется по методике обучения без учителя, а выходного — по методике обучения с учителем. Обычно скрытый слой содержит больше узлов, чем входной. Основная идея заключается в преобразовании точек данных в многомерное пространство, где преобразованные точки становятся линейно разделимыми. Этот подход может использоваться для классификации, регрессии и линейной интерполяции путем изменения природы функции потерь. В задачах классификации используются различные типы функций потерь, такие как функция потерь Уидроу — Хоффа, кусочно-линейная функция потерь и логистическая функция потерь. Специальные случаи различных функций потерь приводят к хорошо известным ядерным методам, таким как ядерный метод SVM и ядерная регрессия. В последние годы RBF-сети используются редко и постепенно переходят в категорию нейронных архитектур. Однако они обладают значительным потенциалом для того, чтобы использоваться в любом сценарии, в котором может применяться ядерный метод. Кроме того, существует возможность комбинировать этот подход с архитектурами прямого распространения, используя после первого скрытого слоя многослойные представления.

5.6. Библиографическая справка

RBF-сети были предложены Брумхедом и Лоу [51] в контексте интерполяции функций. Разделимость многомерных преобразований представлена в работе Ковера [84]. Обзор RBF-сетей содержится в [363]. Эта тема также хорошо рассмотрена в книгах Бишопа [41] и Хайкина [182]. Обзор радиально-базисных функций предоставлен в [57]. Доказательство возможности универсального аппроксимирования функций с помощью RBF-сетей представлено в [173, 365]. Анализ аппроксимирующих свойств RBF-сетей приведен в [366].

Эффективные алгоритмы обучения для RBF-сетей описаны в [347, 423]. Алгоритм обучения расположением центров в RBF-сетях предложен в [530]. Использование деревьев решений для инициализации RBF-сетей обсуждается в [65]. Одни из первых результатов сравнения тренировки RBF-сетей по методикам обучения с учителем и без учителя представлены в [342]. Согласно проведенному в этой работе анализу использование полноценного обучения с учителем, по-видимому, увеличивает вероятность попадания сети в ловушки локальных минимумов. Некоторые идеи относительно улучшения обобщающей

способности RBF-сетей высказаны в работе [43]. Инкрементные RBF-сети обсуждаются в [125]. Подробное обсуждение взаимосвязи RBF-сетей и ядерных методов предоставлено в [430].

5.7. Упражнения

Некоторые из упражнений требуют знания тем из области машинного обучения, которые в данной книге не затрагивались. Для выполнения упражнений 5, 7 и 8 потребуются дополнительные сведения о ядерных методах, спектральной кластеризации и обнаружении выбросов.

1. Рассмотрим следующий вариант сети радиально-базисных функций, в котором скрытые элементы имеют значение 0 или 1. Скрытый элемент равен 1, если расстояние до вектора прототипов меньше σ . В противном случае он имеет нулевое значение. Изучите связь этого метода с RBF-сетями и его сравнительные преимущества и недостатки.
2. Предположим, для предсказания бинарной метки класса как вероятностного значения в выходном узле RBF-сети вы используете сигмоидную активацию в последнем слое. Задайте для этой задачи функцию потерь в виде отрицательного логарифмического правдоподобия. Получите обновления градиентного спуска для весов в последнем слое. Как этот подход соотносится с методом логистической регрессии, который обсуждался в главе 2? В каком случае такой подход будет работать лучше, чем логистическая регрессия?
3. Почему RBF-сеть представляет собой вариант классификатора по методу ближайших соседей, в котором используется обучение с учителем?
4. Подумайте, как расширить три многоклассовые модели, которые обсуждались в главе 2, до RBF-сетей. В частности, изучите варианты расширения следующих моделей: а) многоклассовый перцептрон, б) SVM Уэстона — Уоткинса, в) классификатор SoftMax. Почему результирующие модели обладают большими возможностями, чем те, которые обсуждались в главе 2?
5. Предложите способ расширения RBF-сетей до обучения с учителем с помощью автокодировщиков. Что именно вы будете реконструировать в выходном слое? Специальный случай вашего подхода должен обеспечивать грубую имитацию ядерного метода сингулярного разложения.
6. Предположим, вы изменяете свою RBF-сеть таким образом, чтобы оставлять в скрытом слое только первые k активаций, устанавливая остальные активации равными нулю. Почему такой подход обеспечит лучшую точность классификации на ограниченном объеме данных?

7. Объедините метод конструирования RBF-слоя с помощью первых k активаций (см. упражнение 6) с RBF-автокодировщиком из упражнения 5 для использования методики обучения без учителя. Почему создаваемые при таком подходе представления будут более пригодными для кластеризации? Какова связь этого метода со спектральной кластеризацией?
8. Суть рассмотрения выбросов с точки зрения многообразий заключается в том, что выбросы определяются как точки, которые не вписываются естественным образом в нелинейные многообразия тренировочных данных. Изучите возможность использования RBF-сетей для обнаружения выбросов с использованием методики обучения без учителя.
9. Предположим, что вместо использования RBF-функции в скрытом слое вы используете для активации скалярные произведения прототипов и точек данных. Покажите, что специальным случаем такой модели является линейный перцептрон.
10. Изучите возможность видоизменения RBF-автокодировщика из упражнения 5 для выполнения классификации, осуществляемой с использованием частичного обучения с учителем в случае большого объема неразмеченных данных и небольшого объема размеченных данных.

Глава 6

Ограниченные машины Больцмана

Основная ставка в борьбе за выживание и в процессе эволюции мира — доступ к запасам энергии.

Людвиг Больцман

6.1. Введение

Ограниченная машина Больцмана (restricted Boltzmann machine — RBM) в корне отличается от сети прямого распространения. Обычные нейронные сети транслируют вход на выход, т.е. преобразуют набор входов в набор выходов. RBM — это сети, в которых вероятностные состояния сети обучаются на наборе входов, что хорошо подходит для моделирования на основе обучения без учителя (неконтролируемого обучения). В то время как сеть прямого распространения минимизирует функцию потерь предсказаний (вычисляемую на основе *наблюдаемых* входов) по отношению к наблюдаемому выходу, ограниченная машина Больцмана моделирует совместное распределение вероятностей наблюдаемых атрибутов и некоторых скрытых атрибутов. Если традиционные сети прямого распространения описываются направленными (ориентированными) вычислительными графами, соответствующими потоку вычислений в направлении от входа к выходу, то сети RBM — *ненаправленные*, поскольку они обучаются вероятностным соотношениям, а не отображениям “вход — выход”. Следовательно, ограниченные машины Больцмана — это вероятностные модели, создающие латентные представления базовых точек данных. И хотя для конструирования латентных представлений также могут быть использованы автокодировщики, ограниченная машина Больцмана создает *стохастическое* скрытое представление каждой точки. Большинство автокодировщиков (за исключением вариационного) создает *детерминированные* представления точек данных. В связи с этим RBM требует совершенно иного подхода к ее обучению и использованию.

По своей сути RBM — это модели, обучаемые без учителя, которые генерируют представления латентных признаков точек данных. Тем не менее обученные представления могут сочетаться с традиционным обратным распространением ошибки в тесно связанной (с конкретной RBM) сети прямого распространения для создания приложений на основе обучения с учителем. Такое сочетание контролируемого и неконтролируемого видов обучения работает аналогично предварительному обучению, осуществляемому с помощью архитектуры традиционного автокодировщика (см. раздел 4.7). RBM сыграли свою роль в популяризации идеи предварительного обучения в последние годы. Вскоре эта идея была адаптирована к автокодировщикам, легче поддающимся обучению благодаря использованию детерминированных скрытых состояний.

6.1.1. Исторический экскурс

Ограниченные машины Больцмана эволюционировали из описанной в литературе по нейронным сетям классической модели под названием *сеть Хопфилда*. Сети этого типа состоят из узлов, содержащих бинарные состояния, которые представляют значения бинарных атрибутов тренировочных данных. Сеть Хопфилда создает *детерминированную* модель соотношений между различными атрибутами, используя весовые коэффициенты связей между узлами. Со временем сеть Хопфилда эволюционировала в *машину Больцмана*, которая использует *вероятностные* состояния для представления распределений Бернулли бинарных атрибутов. Машина Больцмана содержит как видимые, так и скрытые состояния. Видимые состояния моделируют распределения наблюдаемых точек данных, тогда как скрытые — распределение латентных (скрытых) переменных. Параметры связей между различными состояниями регулируют их совместное распределение. Задача заключается в обучении параметров модели таким образом, чтобы максимизировать правдоподобие модели. Машина Больцмана относится к семейству (ненаправленных) вероятностных графических моделей. В конечном счете машина Больцмана эволюционировала в *ограниченную машину Больцмана*, которая отличается от простой машины Больцмана в основном тем, что в ней разрешены лишь связи между скрытыми и видимыми узлами. Это упрощение необычайно полезно с практической точки зрения, поскольку позволяет проектировать более эффективные обучающие алгоритмы. RBM представляет собой частный случай вероятностной графической модели, известной как *марковское случайное поле* (Markov random field).

В первые годы модели RBM считались слишком медленными в обучении и поэтому не пользовались большой популярностью. Однако на рубеже столетий для этого класса моделей были предложены более быстрые алгоритмы. Кроме того, RBM приобрели определенную известность благодаря использованию в качестве одной из ансамблевых компонент сети [414], ставшей победителем

соревнований за приз компании Netflix [577]. Обычной сферой применения моделей RBM являются такие задачи, не требующие обучения с учителем, как факторизация матриц, моделирование латентных структур и снижение размерности, хотя имеется много способов их расширения на случаи, требующие обучения с учителем. Следует подчеркнуть, что модели RBM в своей наиболее естественной форме обычно работают с бинарными состояниями, хотя могут работать и с другими типами данных. В этой главе мы в основном ограничимся обсуждением элементов с бинарными состояниями. Успешное обучение глубоких сетей на основе RBM предшествовало успешному опыту обучения обычных нейронных сетей. Иными словами, использование стеков из нескольких слоев RBM для создания глубоких сетей и их эффективного обучения было продемонстрировано еще до того, как аналогичные идеи распространились на обычные сети.

Структура главы

В следующем разделе обсуждаются сети Хопфилда, предшествовавшие семейству моделей Больцмана. Машина Больцмана рассматривается в разделе 6.3. Описанию ограниченных машин Больцмана посвящен раздел 6.4, а рассмотрению возможных способов их применения — раздел 6.5. Использование RBM для работы с данными более общего типа, а не только с бинарными представлениями, рассматривается в разделе 6.6. Процесс создания глубоких сетей на основе стеков из нескольких слоев ограниченных машин Больцмана описан в разделе 6.7. Резюме главы приведено в разделе 6.8.

6.2. Сети Хопфилда

Сети Хопфилда были предложены в 1982 году [207] в качестве модели памяти для сохранения образов. Сеть Хопфилда — это ненаправленная (неориентированная) сеть, в которой d элементов (нейронов) пронумерованы значениями от 1 до d . Каждая связь между парой нейронов записывается в виде (i, j) , где i и j имеют значения от 1 до d . Связи (i, j) — ненаправленные, и каждая из них характеризуется весовым коэффициентом $w_{ij} = w_{ji}$. Предполагается, что каждый узел связан со всеми остальными узлами, однако допускается, что некоторые w_{ij} могут устанавливаться равными нулю, что равносильно исключению соответствующих связей (i, j) . Веса w_{ii} полагаются равными нулю, что исключает взаимодействие узлов с самими собой. С каждым нейроном i ассоциировано состояние s_i . Относительно сети Хопфилда делается важное предположение: каждое состояние s_i может иметь одно из двух значений, 0 и 1, хотя могут использоваться и другие значения, например -1 и $+1$. Кроме того, с каждым i -м узлом ассоциируется смещение b_i ; большие значения b_i повышают вероятность того,

что i -е состояние примет значение 1. Сеть Хопфилда — ненаправленная модель с симметричными связями, поэтому всегда выполняется соотношение $w_{ij} = w_{ji}$.

Каждое бинарное состояние в сети Хопфилда соответствует отдельному измерению в (бинарном) тренировочном наборе данных. Поэтому, если необходимо запомнить d -мерный набор данных, мы должны использовать сеть Хопфилда с d элементами. Состояние i сети соответствует i -му биту в конкретном тренировочном примере. Значения состояний представляют значения бинарных атрибутов из тренировочного примера. Веса в сети Хопфилда являются ее параметрами. Большие положительные веса связей между парами состояний указывают на высокую положительную корреляцию между их значениями, тогда как отрицательные веса — на высокую отрицательную корреляцию. Пример сети Хопфилда с соответствующим набором тренировочных данных приведен на рис. 6.1. В данном случае сеть Хопфилда — полносвязная и имеет шесть видимых состояний, соответствующих шести бинарным атрибутам тренировочных данных.

В сети Хопфилда для обучения весовых параметров используется оптимизационная модель, обеспечивающая улавливание весами положительных и отрицательных связей между атрибутами тренировочного набора данных. Целевым функционалом сети Хопфилда является так называемая *функция энергии сети* (energy function), которая аналогична функции потерь традиционной сети прямого распространения. Функция энергии сети Хопфилда определенным образом оптимизируется, и ее минимизация приводит к тому, что пары узлов, связанные между собой соединениями с большими положительными весами, стремятся иметь одинаковые состояния, а пары узлов с большими отрицательными весами связей — разные. Поэтому во время прохождения фазы тренировки сети Хопфилда веса связей обучаются минимизировать энергию таким образом, чтобы состояния сети фиксировались на значениях бинарных атрибутов отдельных тренировочных точек. Следовательно, процесс обучения весов сети Хопфилда неявно создает модель тренировочного набора данных по методике обучения без учителя. Энергия E конкретной комбинации состояний $s = (s_1 \dots s_d)$ сети Хопфилда определяется следующей формулой:

$$E = -\sum_i b_i s_i - \sum_{i,j:i < j} w_{ij} s_i s_j. \quad (6.1)$$

Член $-b_i s_i$ стимулирует активизацию нейронов с большими положительными смещениями. Точно так же член $-w_{ij} s_i s_j$ стимулирует состояния s_i и s_j принимать близкие значения, если $w_{ij} > 0$. Другими словами, положительные веса будут приводить к “притягиванию” состояний, отрицательные — к “отталкиванию”. Для тренировочных наборов данных небольшого объема результатом такого моделирования является запоминание состояний, что позволяет получать

точки данных тренировочного набора, исходя из точек данных аналогичных, но неполных или поврежденных наборов, путем исследования локальных минимумов функции энергии вблизи этих точек. Иначе говоря, обучение весов сети Хопфилда неявно приводит к запоминанию тренировочных примеров, хотя и существует относительно консервативный предел количества примеров, которые может запомнить сеть Хопфилда, состоящая из d элементов. Для этого предела также существует термин *емкость (мощность)* модели.

6.2.1. Оптимальные конфигурации состояний обученной сети

Обученная сеть Хопфилда характеризуется наличием многих локальных оптимумов функции энергии, каждый из которых соответствует либо запомненной точке данных тренировочного набора, либо точке, представляющей некую плотную область данных набора. Прежде чем перейти к рассмотрению процесса обучения весов сети Хопфилда, остановимся на методологии нахождения локальных минимумов энергии сети Хопфилда, когда веса уже обучены. Локальный минимум определяется как такая комбинация состояний, в которой обращение любого отдельного бита не приведет к дальнейшему уменьшению энергии. В процессе тренировки веса устанавливаются в такие значения, чтобы локальные минимумы соответствовали примерам из тренировочного набора данных.

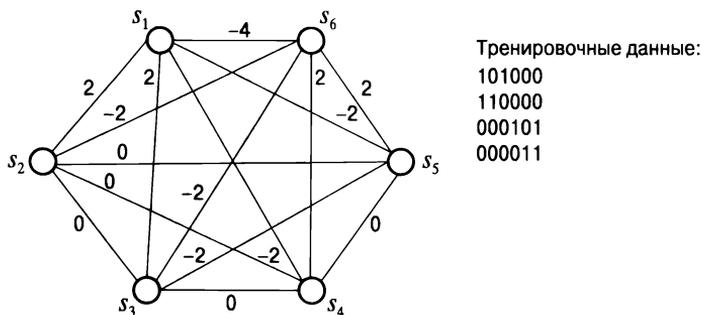


Рис. 6.1. Сеть Хопфилда с шестью видимыми состояниями, соответствующими 6-мерным тренировочным данным

Нахождение оптимальной конфигурации состояний позволяет сети Хопфилда “вспоминать” сохраненные (запомненные) образы. Сеть Хопфилда учится ассоциативному запоминанию, поскольку при заданном наборе входных состояний (т.е. при заданной входной конфигурации битов) она многократно обращает биты, улучшая целевую функцию до тех пор, пока не найдет конфигурацию, изменение которой уже не приводит к дополнительному улучшению целевого

функционала. Образ, соответствующий этому локальному минимуму, зачастую тесно связан с начальным образом (исходным набором состояний), отличаясь от него всего лишь несколькими битами. Кроме того, этим окончательным образом нередко оказывается один из членов тренировочного набора данных (поскольку веса обучались на этих данных). В определенном смысле сеть Хопфилда пролагает путь к *памяти с адресуемым содержимым*.

Если задана начальная комбинация состояний, то как можно обучиться ближайшему локальному минимуму, когда веса уже фиксированы? В этом случае можно обновлять каждое состояние сети для продвижения в направлении глобального минимума энергии, используя *правило обновления порога*. Чтобы разобраться в этом, сравним энергию сети в двух случаях: когда состояние сети s_i установлено в 1 и когда оно установлено в 0. Поочередно подставляя эти два значения s_i в уравнение 6.1 и вычитая один результат из другого, получаем следующее выражение для разницы энергий:

$$\Delta E_i = E_{s_i=0} - E_{s_i=1} = b_i + \sum_{j:j \neq i} w_{ij} s_j. \quad (6.2)$$

Чтобы переключение состояния s_i из 0 в 1 оказывало притягивающий эффект, эта разница должна быть положительной. Таким образом, получаем следующее правило обновления для каждого состояния s_i :

$$s_i = \begin{cases} 1, & \text{если } \sum_{j:j \neq i} w_{ij} s_j + b_i \geq 0; \\ 0 & \text{в противном случае.} \end{cases} \quad (6.3)$$

Это правило итеративно применяется к каждому из состояний s_i , которые в необходимых случаях переключаются в соответствии с данным условием. Зная веса и смещения для любого момента времени, можно найти локальный минимум энергии по отношению к состояниям, повторно применяя вышеприведенное правило обновления.

Локальные минимумы сети Хопфилда зависят от значений обученных весов. Следовательно, для того чтобы можно было “вспомнить” сохраненный в памяти образ, нужно лишь предоставить d -мерный вектор, близкий к тому, который сохранен в памяти, и сеть Хопфилда найдет локальный минимум, аналогичный данной точке, используя ее в качестве начального состояния. Ассоциативная память такого типа свойственна людям, которые часто вспоминают необходимую информацию посредством аналогичного процесса, основанного на ассоциациях. Также можно предоставить неполный вектор начальных состояний и использовать его для восстановления других состояний. Обратимся к сети Хопфилда, представленной на рис. 6.1. Обратите внимание на то, что веса заданы такими, что каждый из четырех тренировочных векторов, приведенных на

рисунке, будет иметь низкую энергию. Однако существуют также ложные минимумы, такие как 111000. Поэтому не гарантируется, что локальные минимумы будут всегда соответствовать точкам тренировочных данных. В то же время локальные минимумы соответствуют некоторым ключевым характеристиками тренировочных данных. В качестве примера рассмотрим ложный минимум, соответствующий образу 111000. Примечательно то, что в данном случае положительно коррелированы как первые, так и последние три бита. Как следствие, значение 111000, соответствующее данному локальному минимуму, отражает некий широкий образ среди тренировочных данных, хотя он и не представлен в них в явном виде. Также следует отметить, что веса в этой сети тесно связаны с образами тренировочных данных. Например, каждая тройка элементов, соответствующих первым и последним трем битам, положительно коррелирована в данной конкретной группе из трех битов. Кроме того, между этими двумя наборами элементов существуют отрицательные корреляции. Следовательно, ребра в каждом из наборов $\{s_1, s_2, s_3\}$ и $\{s_4, s_5, s_6\}$ стремятся иметь положительные веса, тогда как ребра, соединяющие эти наборы, — отрицательные. Настройка весов таким специфическим для данных образом является задачей фазы обучения (раздел 6.2.2).

В зависимости от того, каким был задан начальный вектор состояний, итеративное правило обновления приводит к одному из многих локальных минимумов энергии сети Хопфилда. Каждый из этих локальных минимумов может соответствовать одному из образов тренировочного набора данных, запомненных в процессе обучения, и в результате минимизации будет достигнут сохраненный в памяти образ, ближайший к начальному вектору состояний. Эти запомненные образы неявно хранятся в весах, обученных в процессе тренировки. Однако сеть Хопфилда может совершать ошибки, когда тесно связанные между собой обучающие образцы сливаются в один (более глубокий) минимум. Например, если тренировочные данные содержат образцы 1110111101 и 1110111110, то сеть Хопфилда может обучиться тому, что 1110111111 является локальным минимумом. Поэтому в некоторых запросах может восстанавливаться образец, отличающийся небольшим количеством битов от образца, фактически имеющегося среди тренировочных данных. Однако это не более чем форма обобщения модели, используемая сетью Хопфилда для сохранения репрезентативных кластерных центров вместо отдельных точек тренировочных данных. Другими словами, если объем данных превышает емкость модели, то, вместо того чтобы запоминать данные, она обобщает их. В конце концов, сеть Хопфилда создает модели, обучаясь на тренировочных данных без учителя.

Сеть Хопфилда может быть использована для извлечения данных из ассоциативной памяти, исправления поврежденных данных и заполнения атрибутов. Задачи извлечения данных из ассоциативной памяти и очистки поврежденных

данных близки между собой. В обоих случаях в качестве начального состояния используется испорченный вход (или целевой вход для ассоциативного извлечения данных), а конечное состояние используется в качестве очищенного (или востребованного) выхода. Если же речь идет о заполнении атрибутов, то вектор состояний инициализируется установкой известных значений для наблюдаемых состояний и случайных значений для ненаблюдаемых. В этом случае лишь ненаблюдаемые состояния обновляются до достижения сходимости. Когда сходимость достигнута, битовые значения этих состояний обеспечивают завершённое представление.

6.2.2. Обучение сети Хопфилда

При заданном наборе тренировочных данных нужно обучить веса таким образом, чтобы локальные минимумы этой сети располагались вблизи примеров (или областей их плотного расположения) тренировочного набора. Сети Хопфилда обучаются по Хеббу. Правила обучения Хебба основываются на данных нейробиологии, согласно которым синаптические связи усиливаются, когда между выходами нейронов по обе стороны синапса существует высокая корреляция. Пусть $x_{ij} \in \{0, 1\}$ представляет j -й бит i -й тренировочной точки. Предполагается, что количество тренировочных примеров равно n . Правила обучения Хебба устанавливают следующие значения весов:

$$w_{ij} = 4 \frac{\sum_{k=1}^n (x_{ki} - 0,5) \cdot (x_{kj} - 0,5)}{n}. \quad (6.4)$$

Суть этого правила можно понять, если заметить, что, когда два бита i и j в тренировочном наборе положительно коррелированы, выражение $(x_{ki} - 0,5) \cdot (x_{kj} - 0,5)$ обычно будет иметь положительное значение. Как следствие, для весов связей между соответствующими элементами также будут установлены положительные значения. С другой стороны, если два бита рассогласованы, то для весов будут установлены отрицательные значения. Это правило также можно применять без нормирующего знаменателя:

$$w_{ij} = 4 \sum_{k=1}^n (x_{ki} - 0,5) \cdot (x_{kj} - 0,5). \quad (6.5)$$

На практике часто желательно иметь инкрементные алгоритмы обучения для поточечных обновлений. Для обновления веса w_{ij} с использованием лишь k -й тренировочной точки данных применима следующая формула:

$$w_{ij} \leftarrow w_{ij} + 4(x_{ki} - 0,5) \cdot (x_{kj} - 0,5) \quad \forall i, j.$$

Смещения b_i можно обновлять, предположив, что существует одно фиктивное состояние, которое всегда активизировано, а само смещение представляет вес связи между этим фиктивным и i -м состояниями:

$$b_i \leftarrow b_i + 2(x_{ki} - 0,5) \quad \forall i.$$

В тех случаях, когда принимается соглашение о том, что компоненты векторов состояний могут иметь значения -1 или $+1$, приведенные выше правила приобретают следующий вид:

$$w_{ij} \leftarrow w_{ij} + x_{ki}x_{kj} \quad \forall i, j;$$

$$b_i \leftarrow b_i + x_{ki} \quad \forall i.$$

Широко применяются и другие правила обучения, такие как *правило обучения Сторки*. За дополнительной информацией обратитесь к библиографической справке.

Емкость сети Хопфилда

Каким должен быть размер тренировочного набора данных, чтобы сеть Хопфилда с d видимыми элементами могла сохранять образы и обеспечивать их безошибочное извлечение из ассоциативной памяти? Можно показать, что емкость памяти сети Хопфилда с d элементами составляет всего лишь $0,15 \cdot d$ тренировочных примеров. Поскольку каждый тренировочный пример содержит d бит, то сеть Хопфилда способна хранить всего лишь около $0,15 \cdot d^2$ бит. Такая форма хранения информации неэффективна, поскольку количество весов в сети равно $d(d-1)/2 = O(d^2)$. Кроме того, веса не являются бинарными, и можно показать, что для них требуется $O(\log(d))$ бит. При большом количестве тренировочных примеров возрастает вероятность ошибок (ассоциативного вспоминания), которые представляют обобщенные предсказания, сделанные на большом объеме данных. И хотя может показаться, что этот тип обобщения полезен для целей машинного обучения, существуют ограничения на использование сетей Хопфилда в приложениях подобного рода.

6.2.3. Создание экспериментальной рекомендательной системы и ее ограничения

Сети Хопфилда часто используют не в типичных приложениях машинного обучения, требующих обобщения данных, а в приложениях, ориентированных на запоминание данных. Поясним суть ограничений сети Хопфилда на примере бинарной коллаборативной фильтрации. Поскольку сети Хопфилда работают с бинарными данными, ограничимся случаем *неявной обратной связи с пользователями* (implicit feedback), когда с каждым пользователем ассоциируется набор бинарных атрибутов, указывающих на то, смотрел ли данный пользователь

соответствующие фильмы. Рассмотрим ситуацию, когда пользователь Боб смотрел фильмы *Шрек* и *Аладдин*, тогда как пользователь Алиса смотрела фильмы *Ганди*, *Нерон* и *Терминатор*. Не составляет труда сконструировать полносвязную сеть Хопфилда для множества всех фильмов, установив значение 1 для состояний просмотренных фильмов и значение 0 для всех остальных состояний. Эту конфигурацию можно использовать в каждой точке тренировочного набора для обновления весов. Разумеется, в случае очень большого базового количества состояний (фильмов) такой подход может стать чрезвычайно дорогостоящим. Для базы данных, содержащей 106 фильмов, модель имела бы 1012 ребер, большинство которых соединяло бы состояния с нулевыми значениями. Это связано с тем, что данные неявной обратной связи подобного типа часто оказываются разреженными и большинство состояний будут иметь нулевые значения.

Одним из способов разрешения этой проблемы является *отрицательное семплирование* (negative sampling). Этот подход предполагает создание для каждого пользователя отдельной сети Хопфилда, содержащей информацию о просмотренных им фильмах и небольшую выборку фильмов, которые он еще не просмотрел. Например, можно сформировать случайную выборку из 20 не просмотренных (Алисой) фильмов и создать сеть Хопфилда, содержащую $20 + 3 = 23$ состояния (включая просмотренные фильмы). Сеть Хопфилда для Боба будет содержать $20 + 2 = 22$ состояния, причем список не просмотренных им фильмов может быть совершенно другим. Однако для пар фильмов, общих для двух сетей, веса будут разделяться. В процессе тренировки веса всех ребер инициализируются нулями. Для обучения разделяемых весов можно использовать повторение итераций тренировки по различным сетям Хопфилда (с использованием одного и того же алгоритма, который уже обсуждался ранее). Основное отличие состоит в том, что итерирование по различным тренировочным точкам приведет к итерированию по различным сетям Хопфилда, каждая из которых содержит небольшое подмножество базовой сети. В типичных случаях в каждой из этих сетей будет представлено лишь небольшое подмножество этих 1012 ребер, причем большинство ребер не появятся ни в одной из этих сетей. Веса таких ребер будут неявно полагаться равными нулю.

А теперь предположим, что пользователь Мэри просмотрела фильмы *Инопланетянин* и *Шрек* и мы хотим порекомендовать ей фильмы для просмотра. Для этого мы используем полносвязную сеть Хопфилда, в которой представлены лишь ребра с ненулевыми весовыми коэффициентами. Состояния, соответствующие фильмам *Инопланетянин* и *Шрек*, инициализируются значением 1, а все остальные состояния — нулевым значением. Впоследствии мы разрешим обновление всех других состояний, чтобы найти минимум энергии конфигурации сети Хопфилда. Пользователю можно будет рекомендовать к просмотру те

фильмы, состояния которых в процессе обновления будут установлены равными нулю. Однако в идеальном случае нам хотелось бы, чтобы лучшие рекомендованные фильмы были *ранжированы* в соответствии с их рейтингом. Один из возможных способов упорядочения всех фильмов заключается в использовании *разницы в энергиях* между двумя состояниями каждого из них. Разница в энергиях вычисляется лишь после того, как найдена минимальная энергетическая конфигурация. Однако такой подход довольно наивен, поскольку окончательная конфигурация — детерминированная и содержит бинарные значения, тогда как экстраполированные значения могут оцениваться только в терминах вероятностей. Например, было бы гораздо естественнее использовать какую-либо функцию энергетического зазора (например, сигмоиду) для создания вероятностных оценок. Кроме того, было бы полезно иметь возможность захватывать коррелированные наборы фильмов с помощью латентных (скрытых) состояний. Очевидно, что для расширения выразительных возможностей сети Хопфилда мы должны привлечь другие методы.

6.2.4. Улучшение выразительных возможностей сети Хопфилда

Хотя это и не является стандартной практикой, для улучшения выразительных возможностей сети Хопфилда в нее можно добавлять скрытые элементы. Скрытые состояния предназначены для захвата латентной структуры данных. Веса соединений между скрытыми и видимыми элементами будут улавливать взаимосвязь латентной структуры с данными. В некоторых случаях становится возможным приближенное представление данных только в терминах небольшого количества скрытых состояний. Например, если данные содержат два тесно связанных кластера, то информация об этом может быть сохранена с помощью двух состояний. Рассмотрим случай, когда мы усиливаем сеть Хопфилда, приведенную на рис. 6.1, добавляя в нее два скрытых элемента. Результирующая сеть представлена на рис. 6.2. Для ясности ребра с весами, близкими к нулю, не показаны на рисунке. Несмотря на то что исходные данные определены в терминах шести битов, два скрытых элемента обеспечивают представление данных в терминах двух битов. Это скрытое представление является сжатой версией данных, которая сообщает определенную информацию относительно имеющегося образца. В сущности, все образцы данных сжимаются до одного из образцов 10 или 01, в зависимости от того, какие биты доминируют в обучающем образце: три первых или три последних. Если зафиксировать скрытые состояния сети Хопфилда на значении 10 и инициализировать видимые состояния случайными значениями, то при повторном применении к каждому состоянию правила обновления, выраженного уравнением 6.3, мы будем часто получать образец 111000. Если же начать со скрытого состояния 01, то в

качестве окончательной устойчивой точки мы будем получать образец 000111. Примечательно то, что образцы 000111 и 111000 представляют собой хорошее приближение к двум типам образцов данных, что и ожидается от методики сжатия. Если мы предоставим неполную версию видимых элементов, а затем будем итеративно обновлять другие состояния по правилу обновления, выражаемому уравнением 6.3, то будем часто получать либо образец 000111, либо образец 111000, в зависимости от распределения битов в неполном представлении. Если мы добавим в сеть Хопфилда скрытые элементы и позволим состояниям иметь вероятностные (не детерминированные) значения, то получим машину Больцмана. Именно по этой причине машины Больцмана можно рассматривать как стохастические сети Хопфилда со скрытыми элементами.

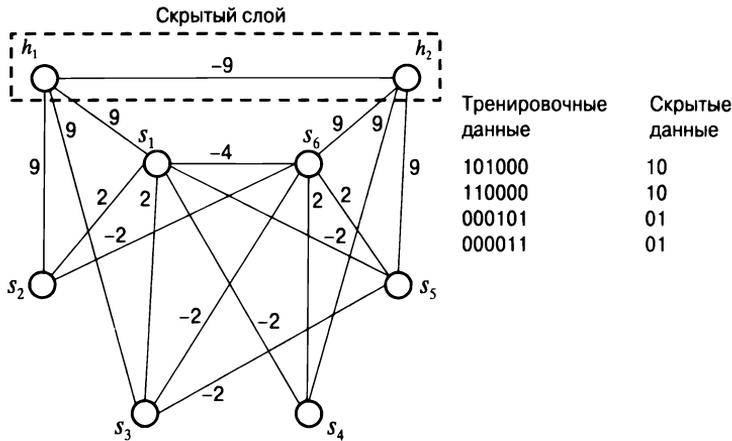


Рис. 6.2. Сеть Хопфилда с двумя скрытыми узлами

6.3. Машина Больцмана

В данном разделе мы будем полагать, что машина Больцмана в общей сложности содержит $q = (m + d)$ состояний, где d — количество видимых, а m — скрытых состояний. Конкретная конфигурация состояний определяется значением вектора состояний $s = (s_1 \dots s_q)$. Если желательно ввести явные обозначения для видимых и скрытых состояний в s , то вектор состояний можно записать в виде пары (v, h) , где v — набор видимых, а h — скрытых элементов. Состояния в (v, h) представляют в точности тот же набор, что и $s = (s_1 \dots s_q)$, за исключением того, что в первом варианте видимые и скрытые состояния обозначены явно.

Машина Больцмана — это вероятностное обобщение сети Хопфилда. Сеть Хопфилда детерминированно устанавливает каждое состояние s_i в 1 или 0, в зависимости от того, положительна или отрицательна величина энергетического зазора ΔE_i состояния s_i . Вспомните, что энергетический зазор i -го элемента

определяется как разница энергий двух конфигураций (при условии, что другие состояния зафиксированы на заранее определенных значениях):

$$\Delta E_i = E_{s_i=0} - E_{s_i=1} = b_i + \sum_{j:j \neq i} w_{ij} s_j. \quad (6.6)$$

Сеть Хопфилда детерминированно устанавливает значение s_i равным 1, если энергетический зазор положителен. С другой стороны, машина Больцмана назначает s_i вероятностное значение, зависящее от энергетического зазора. В случае положительного зазора присваиваются вероятностные значения, превышающие 0,5. Вероятность состояния s_i определяется путем применения сигмоиды к энергетическому зазору:

$$P(s_i | s_1, \dots, s_{i-1}, s_{i+1}, s_q). \quad (6.7)$$

Обратите внимание на то, что теперь состояние s_i является случайной переменной Бернулли, и нулевой энергетический зазор приводит к вероятности 0,5 для каждого бинарного выхода состояния.

Для конкретного набора параметров w_{ij} и b_i машина Больцмана определяет распределение вероятности по различным конфигурациям состояний. Энергия конкретной конфигурации $\bar{s} = (\bar{v}, \bar{h})$ обозначается как $E(\bar{s}) = E([\bar{v}, \bar{h}])$ и определяется так же, как и для сети Хопфилда:

$$E(\bar{s}) = -\sum_i b_i s_i - \sum_{i,j:i < j} w_{ij} s_i s_j. \quad (6.8)$$

Однако в случае машины Больцмана эти состояния известны лишь с вероятностной точки зрения (в соответствии с уравнением 6.7). Условное распределение следует более фундаментальному определению безусловной вероятности $P(s)$ конкретной конфигурации s :

$$P(\bar{s}) \propto \exp(-E(\bar{s})) = \frac{1}{Z} \exp(-E(\bar{s})). \quad (6.9)$$

Нормирующий множитель Z определяется так, чтобы вероятности всех возможных конфигураций в сумме составляли 1:

$$Z = \sum_{\bar{s}} \exp(-E(\bar{s})). \quad (6.10)$$

Нормирующий множитель Z также называют *статистической суммой* (partition function). В общем случае явное вычисление статистической суммы наталкивается на большие трудности, так как она содержит экспоненциальное количество членов, соответствующих всем возможным конфигурациям состояний. Ввиду неподатливости статистической суммы точное вычисление вероятности $P(\bar{s}) = P(\bar{v}, \bar{h})$ невозможно. Тем не менее вычисление многих типов

условных вероятностей, например $P(\bar{v} | \bar{h})$, оказывается возможным, поскольку они выражаются через отношения, и неподатливые нормирующие множители взаимно уничтожаются. К примеру, условная вероятность, выражаемая уравнением 6.7, подчиняется более фундаментальному определению вероятности конфигурации (см. уравнение 6.9) следующим образом:

$$\begin{aligned}
 P(s_i = 1 | s_1, \dots, s_{i-1}, s_{i+1}, s_q) &= \frac{P(s_1, \dots, s_{i-1}, \overset{s_i}{1}, s_{i+1}, s_q)}{P(s_1, \dots, s_{i-1}, \underset{s_i}{1}, s_{i+1}, s_q) + P(s_1, \dots, s_{i-1}, \underset{s_i}{1}, s_{i+1}, s_q)} = \\
 &= \frac{\exp(-E_{s_{i=1}})}{\exp(-E_{s_{i=1}}) + \exp(-E_{s_{i=0}})} = \frac{1}{1 + \exp(E_{s_{i=1}} - E_{s_{i=0}})} = \\
 &= \frac{1}{1 + \exp(-\Delta E_i)} = \text{сигмоида } (\Delta E_i).
 \end{aligned}$$

Это то же условие, что и в уравнении 6.9. Не сложно увидеть, что логистическая сигмоида связана с понятием энергии из статистической физики.

О преимуществах вероятностной трактовки рассматриваемых состояний говорит хотя бы тот факт, что, извлекая выборки из этих состояний, мы можем создавать новые точки данных, которые выглядят как оригинальные данные. В силу этого машины Больцмана являются скорее стохастическими моделями, чем детерминированными. Многие генеративные модели в машинном обучении (например, смешанные гауссовские модели кластеризации) используют последовательный процесс, в котором сначала из априорного распределения семплируются скрытые состояния, а затем генерируются видимые наблюдения в зависимости от скрытых состояний. В случае машин Больцмана, в которых зависимости между всеми парами состояний являются ненаправленными, это не так. Все видимые состояния зависят от скрытых состояний в той же мере, в какой скрытые состояния зависят от видимых. Вследствие этого генерирование данных с помощью машины Больцмана может быть более затруднительным, чем в случае многих других генеративных моделей.

6.3.1. Как машина Больцмана генерирует данные

Динамика генерирования данных в машине Больцмана осложняется существованием циклических зависимостей между состояниями в силу уравнения 6.7. Поэтому для генерирования выборок точек данных с помощью машины Больцмана нам потребуется итеративный процесс, позволяющий добиться того, чтобы уравнение 6.7 выполнялось для всех состояний. Машина Больцмана итеративно семплирует состояния, используя условное распределение, сгенерированное на основании значений состояний из предыдущей итерации, до тех пор

пока не будет достигнуто состояние теплового (термодинамического) равновесия. Понятие *теплового равновесия* означает, что мы начинаем со случайного набора состояний, используем уравнение 6.7 для вычисления их условных вероятностей, а затем семплируем значения состояний, вновь используя полученные вероятности. Обратите внимание на то, что мы можем итеративно генерировать s_i с помощью вероятностей $P(s_i | s_1, \dots, s_{i-1}, s_{i+1}, s_q)$ из уравнения 6.7. В результате выполнения этого процесса в течение достаточно длительного времени выборочные значения видимых состояний предоставят случайные выборочные значения сгенерированных точек данных. Период времени, необходимый для достижения теплового равновесия, называют *временем обжига* (burn-in time) данной процедуры. Описанный подход называется *семплированием по Гиббсу* (Gibbs sampling) или *семплированием методом Монте-Карло марковских цепей* (Markov Chain Monte Carlo sampling — MCMC).

В состоянии теплового равновесия сгенерированные точки будут представлять модель, захваченную машиной Больцмана. Заметим, что корреляция между сгенерированными точками данных будет зависеть от весов связей между различными состояниями. Для состояний с большими весовыми коэффициентами связей между ними будут наблюдаться тенденция к высокой взаимной корреляции. Например, в случае *приложений интеллектуального анализа данных*, в которых состояния соответствуют наличию или отсутствию слов, будет существовать корреляция между тематически близкими словами. Поэтому, если машина Больцмана была предварительно обучена на наборе соответствующих текстовых данных, она будет генерировать векторы, содержащие эти типы равновесных корреляций между словами, даже если состояния инициализируются случайным образом. Следует отметить, что генерирование набора точек данных с помощью машины Больцмана — более сложный процесс по сравнению со многими другими вероятностными моделями. Например, генерирование точек данных из смешанной гауссовской модели требует лишь непосредственного семплирования точек из распределения вероятности семплируемой смеси компонент. С другой стороны, ненаправленная природа машины Больцмана вынуждает нас выполнять процесс, обеспечивающий достижение теплового равновесия, лишь для того, чтобы генерировать выборки. Это еще более затрудняет задачу обучения весов связей между состояниями для заданного тренировочного набора данных.

6.3.2. Обучение весов машины Больцмана

В случае машины Больцмана мы хотим обучать веса таким образом, чтобы максимизировать логарифмическую функцию правдоподобия имеющегося набора данных. Логарифмические функции правдоподобия индивидуальных состояний вычисляются путем использования логарифма вероятностей,

определяемых уравнением 6.9. Взяв логарифм от обеих частей уравнения 6.9, мы получим следующее соотношение:

$$\log[P(\bar{s})] = -E(\bar{s}) - \log(Z). \quad (6.11)$$

Таким образом, вычисление частной производной $\frac{\partial \log[P(\bar{s})]}{\partial w_{ij}}$ требует вычисления отрицательной производной энергии, хотя при этом еще имеется дополнительный член, включающий статистическую сумму. Функция энергии (см. уравнение 6.8) линейно зависит от веса w_{ij} с коэффициентом $-s_i s_j$. Следовательно, частная производная энергии по весу w_{ij} равна $-s_i s_j$. Можно показать, что результирующее соотношение имеет следующий вид:

$$\frac{\partial \log[P(\bar{s})]}{\partial w_{ij}} = \langle s_i, s_j \rangle_{\text{данные}} - \langle s_i, s_j \rangle_{\text{модель}}. \quad (6.12)$$

Здесь член $\langle s_i, s_j \rangle_{\text{данные}}$ представляет усредненное значение $s_i s_j$, полученное во время выполнения генеративного процесса, описанного в разделе 6.3.1, когда видимые состояния фиксируются на значениях атрибутов в тренировочной точке. Усреднение выполняется по всем мини-пакетам тренировочных точек. Аналогичный член $\langle s_i, s_j \rangle_{\text{модель}}$ представляет усредненное значение $s_i s_j$ в состоянии теплового равновесия, полученное во время выполнения генеративного процесса без фиксирования видимых состояний на тренировочных точках. В этом случае усреднение производится по множеству примеров выполнения процесса до достижения теплового равновесия. Интуиция подсказывает, что для нас желательно увеличение весов связей, которые дифференцированно (по сравнению с неограниченной моделью) стремятся к совместной активизации в условиях, когда видимые состояния фиксируются на тренировочных точках данных. Это именно то, что достигается в процессе описанного выше обновления, в котором используется разница между значениями $\langle s_i, s_j \rangle$, получаемыми усреднением, центрированным на данных, и усреднением, центрированным на модели. Из приведенного обсуждения отчетливо следует, что для выполнения обновлений должны генерироваться два типа выборки.

- 1. Выборки на основе данных.** В выборках первого типа видимые состояния фиксируются на векторе, случайно выбираемом из тренировочного набора данных. Скрытые состояния инициализируются случайными значениями, извлекаемыми из распределения Бернулли с вероятностью 0,5. Затем вероятность каждого скрытого состояния вычисляется заново в соответствии с уравнением 6.7. Из этих вероятностей повторно генерируются выборочные значения скрытых состояний. Процесс повторяется в течение какого-то времени, пока не будет достигнуто состояние теплового

равновесия. Значения скрытых переменных в этой точке предоставляют требуемые выборки. Заметьте, что видимые состояния фиксируются на атрибутах вектора соответствующих данных и поэтому не должны семплироваться.

- 2. Выборки на основе модели.** Второй тип выборок не налагает никаких ограничений на состояния, и выборочные значения семплируются из неограниченной модели. Этот подход совпадает с описанным выше, за исключением того, что случайными значениями инициализируются как видимые, так и скрытые состояния, и обновления непрерывно выполняются до тех пор, пока не будет достигнуто тепловое равновесие.

Эти выборки позволяют создать правило обновления для весов. На основе выборок первого типа можно вычислить член $\langle s_i, s_j \rangle_{\text{данные}}$, который представляет корреляции между состояниями узлов s_i и s_j , когда видимые векторы фиксированы на векторе тренировочных данных \mathcal{D} , а скрытым состояниям разрешено меняться. Поскольку используется мини-пакет тренировочных векторов, мы получаем множество выборочных значений векторов состояний. Значение $\langle s_i, s_j \rangle_{\text{данные}}$ вычисляется как усредненное произведение всех подобных векторов состояний, получаемых семплированием по Гиббсу. Аналогичным образом можно оценить значение $\langle s_i, s_j \rangle_{\text{модель}}$, используя усредненное произведение s_i и s_j из центрированных на модели выборок, полученных семплированием по Гиббсу. Вычислив эти значения, мы можем использовать следующее правило обновления:

$$w_{ij} \leftarrow w_{ij} + \alpha \left(\underbrace{\langle s_i, s_j \rangle_{\text{данные}} - \langle s_i, s_j \rangle_{\text{модель}}}_{\text{Частная производная логарифмической вероятности}} \right) \quad (6.13)$$

Правило обновления для смещения имеет аналогичный вид, за исключением того, что состояние s_j устанавливается в 1. Этого можно добиться, используя фиктивный элемент смещения, который является видимым и связан со всеми состояниями:

$$b_i \leftarrow b_i + \alpha \left(\langle s_i, 1 \rangle_{\text{данные}} - \langle s_i, 1 \rangle_{\text{модель}} \right). \quad (6.14)$$

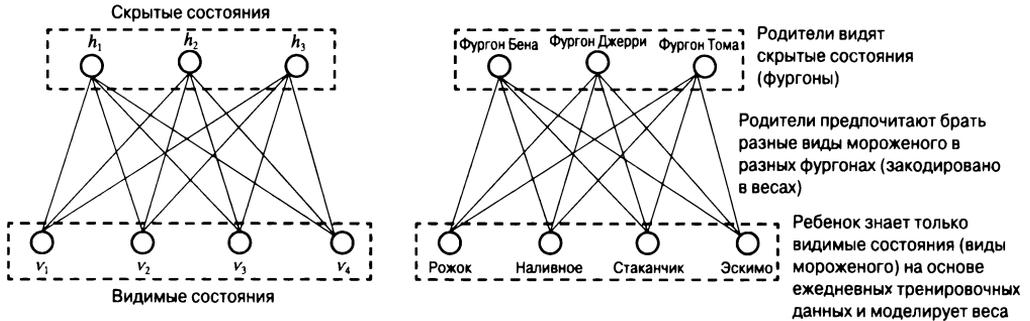
Обратите внимание на то, что значение $\langle s_i, 1 \rangle$ — это среднее выборочных значений s_i для мини-пакета тренировочных примеров, которые взяты из выборок, центрированных либо на данных, либо на модели.

Такой подход аналогичен правилу обновления Хебба для сети Хопфилда, но при этом мы также исключаем из обновлений влияние корреляции в выборках, основанных на модели. Исключение этих корреляций требуется для учета эффекта статистической суммы в уравнении 6.11. Основной проблемой вышеупомянутого правила обновления является то, что на практике оно работает очень

медленно. Это связано с тем, что для достижения теплового равновесия процедура семплирования методом Монте-Карло должна использовать большое количество выборочных значений. Для такого трудоемкого процесса существуют гораздо более быстрые приближения. В следующем разделе мы обсудим применение данного подхода в контексте упрощенной версии машины Больцмана, которую называют *ограниченной машиной Больцмана*.

6.4. Ограниченная машина Больцмана

В машине Больцмана связи между скрытыми и видимыми элементами могут быть произвольными. Например, ребрами могут быть соединены два скрытых состояния или же два видимых. Такое общее допущение создает излишние сложности. Естественным частным случаем машины Больцмана является *ограниченная машина Больцмана* (restricted Boltzmann machine — RBM), которая состоит из двух частей и в которой разрешены только связи между скрытыми и видимыми элементами. Пример ограниченной машины Больцмана приведен на рис. 6.3, а. В данном примере имеются три скрытых и четыре видимых узла. Каждое скрытое состояние связано с одним или несколькими видимыми состояниями, но связи между двумя скрытыми или двумя видимыми состояниями отсутствуют. Ограниченную машину Больцмана первоначально называли *фисгармония* (harmonium) [457].



а) Ограниченная машина Больцмана б) Фургоны с мороженым для Алисы

Рис. 6.3. Ограниченная машина Больцмана. Обратите внимание на ограничение, заключающееся в отсутствии взаимодействия скрытых элементов между собой и видимых элементов между собой

Обозначим скрытые и видимые элементы как $h_1 \dots h_m$ и $v_1 \dots v_d$ соответственно. Смещение, ассоциированное с видимым узлом v_i , обозначим через $b_i^{(v)}$, а смещение, ассоциированное со скрытым узлом h_j , — через $b_j^{(h)}$. Обратите внимание на верхние индексы, позволяющие различать смещения видимых и скрытых узлов. Вес связи между видимым узлом v_i и скрытым узлом h_j обозначим

через w_{ij} . В случае ограниченной машины Больцмана эти обозначения весов имеют несколько иной смысл (по сравнению с обычной машиной Больцмана), поскольку скрытые и видимые элементы индексируются по отдельности. Например, соотношение $w_{ij} = w_{ji}$ уже не будет выполняться, так как первый индекс, i , всегда относится к видимому узлу, а второй индекс, j , — к скрытому. При экстраполяции уравнений из предыдущего раздела очень важно помнить об этих отличиях.

Чтобы сделать пояснения более понятными, работа машины Больцмана будет рассмотрена в этом разделе на примере получения Алисой различных видов мороженого, которое родители ежедневно покупают для нее у разных продавцов (рис. 6.3, б). Представьте, что тренировочные данные соответствуют четырем битам, которые определяют виды десертного мороженого, ежедневно получаемого Алисой от родителей. В данном примере эти данные описывают видимые состояния. Таким образом, Алиса может собрать 4-мерные точки тренировочных данных, поскольку она каждый день получает десерты различного типа в количестве от 0 до 4. Однако родители покупают мороженое для Алисы у одного¹ или нескольких из трех продавцов, указанных на том же рисунке в качестве скрытых состояний. Идентификационные данные фургонов скрыты от Алисы, хотя она знает, что мороженое доставляется фургонами от трех разных продавцов (причем в один день может быть сформирован заказ, состоящий из нескольких видов мороженого). Родители Алисы — люди нерешительные, и их процесс принятия решений не совсем обычен, поскольку они могут изменить свое мнение относительно выбора мороженого после того, как выберут продавца, и наоборот. Вероятность выбора конкретного вида мороженого зависит как от выбора фургона, так и от весов фургонов. Точно так же вероятность выбора фургона зависит от того, какое именно мороженое хотят купить родители, и от тех же весов. Поэтому родители Алисы могут многократно изменять свое мнение относительно выбора мороженого после выбора фургона и выбора фургона после выбора мороженого, пока не придут к окончательному решению относительно сегодняшней покупки. Как видим, подобные циклические связи характерны для ненаправленных моделей, и процесс принятия решений, используемый родителями Алисы, напоминает семплирование по Гиббсу.

Ограничение, заключающееся в разделении сети на две части, значительно упрощает алгоритмы вывода в RBM, позволяя в то же время сохранить предсказательные возможности такого подхода. Если нам известны все значения

¹ В ситуациях, когда поставщики не выбираются, этот пример вносит излишние сложности в терминах семантической интерпретируемости. Но даже и в этом случае вероятности различных видов мороженого могут быть ненулевыми, в зависимости от смещения. Для объяснения подобных случаев можно ввести фиктивный фургон, который всегда выбирается.

видимых элементов (как это обычно бывает, когда предоставляется точка данных), то вероятности скрытых элементов можно вычислить за один шаг, не используя трудоемкий процесс семплирования по Гиббсу. Например, для каждого элемента вероятность того, что он примет значение 1, можно записать непосредственно в виде логистической функции значений видимых элементов. Иначе говоря, применив к ограниченной машине Больцмана уравнение 6.7, можно получить следующее соотношение:

$$P(h_j = 1 | \bar{v}) = \frac{1}{1 + \exp(-b_j^{(h)} - \sum_{i=1}^d v_i w_{ij})}. \quad (6.15)$$

Этот результат вытекает непосредственно из уравнения 6.7, которое связывает вероятности состояний с энергетическим зазором ΔE_j между состояниями $h_j = 0$ и $h_j = 1$. Значение ΔE_j равно $b_j + \sum_i v_i w_{ij}$, если видимые состояния являются наблюдаемыми. Это соотношение также можно использовать для снижения размерности представления тренировочных векторов после того, как обучены веса. В частности, в случае машины Больцмана с m скрытыми элементами значение j -го скрытого элемента можно установить равным вероятности, вычисляемой в соответствии с уравнением 6.15. Отметим, что такой подход обеспечивает получение редуцированного представления бинарных данных, выраженное вещественными значениями. Вышеприведенное уравнение также можно записать, используя сигмоиду:

$$P(h_j = 1 | \bar{v}) = \text{сигмоида} \left(b_j^{(h)} + \sum_{i=1}^d v_i w_{ij} \right). \quad (6.16)$$

Также можно использовать выборку скрытых состояний для того, чтобы сгенерировать точки данных за один шаг. Это возможно в силу того, что связи между видимыми и скрытыми элементами аналогичны связям между элементами в ненаправленной архитектуре RBM, разделенной на две части. Другими словами, мы можем использовать уравнение 6.7 для получения следующего соотношения:

$$P(v_i = 1 | \bar{h}) = \frac{1}{1 + \exp(-b_i^{(v)} - \sum_{j=1}^m h_j w_{ij})}. \quad (6.17)$$

Эту вероятность также можно выразить через сигмоиду:

$$P(v_i = 1 | \bar{h}) = \text{сигмоида} \left(b_i^{(v)} + \sum_{j=1}^m h_j w_{ij} \right). \quad (6.18)$$

Использование сигмоиды удобно, в частности, тем, что это нередко дает возможность создать близкую сеть прямого распространения с сигмоидной активацией элементов, в которой веса, обученные машиной Больцмана, используются для непосредственного вычисления отображений “вход — выход”. Затем веса этой сети настраиваются с помощью обратного распространения ошибки. Пример такого подхода будет предоставлен в соответствующем разделе.

Обратите внимание на то, что веса кодируют сродство между видимыми и скрытыми состояниями. Большой положительный вес связи между двумя состояниями означает, что они, вероятнее всего, встречаются вместе. Например, в ситуации, представленной на рис. 6.3, б, вполне может быть так, что родители предпочитают покупать рожки и эскимо у Бена, а наливное мороженое и мороженое в стаканчиках — у Тома. Эти предрасположенности закодированы в весах, которые регулируют выбор как видимых, так и скрытых состояний циклическим способом. *Циклическая* природа этих соотношений создает трудности, поскольку связь между выбором десерта и выбором продавца работает в обоих направлениях; это является разумным основанием для использования семплирования по Гиббсу. Хотя Алиса может и не знать, у какого именно продавца было куплено то или иное мороженое, она заметит результирующую корреляцию между битами тренировочных данных. В действительности, если Алисе известны веса RBM, она может использовать семплирование по Гиббсу для того, чтобы генерировать 4-битовые точки, представляющие “типичные” виды мороженого, которые она будет получать в будущие дни. Алиса даже может обучиться весам модели на примерах, что является сущностью генеративной модели обучения без учителя. Если при трех фигурирующих в этом примере скрытых состояниях (фургонах) задать достаточное количество 4-мерных тренировочных точек данных, то Алиса сможет обучиться соответствующим смещениям и весам связей между видимыми сортами мороженого и скрытыми продавцами. Алгоритм, реализующий этот процесс обучения, обсуждается в следующем разделе.

6.4.1. Обучение RBM

Вычисление весов RBM осуществляется с использованием правила обучения, аналогичного тому, которое применяется в случае машин Больцмана. В частности, возможно создание алгоритма, основанного на мини-пакетном подходе. Веса w_{ij} инициализируются небольшими значениями. Для текущего набора весов действуют следующие правила обновления.

- *Позитивная фаза.* Этот подход основан на использовании мини-пакета тренировочных примеров и вычислении вероятностей состояний каждого элемента всего за один шаг с помощью уравнения 6.15. Затем на

основании этих вероятностей для каждого скрытого состояния генерируется одно выборочное значение. Этот процесс повторяется для каждого элемента мини-пакета тренировочных примеров, после чего вычисляется корреляция между различными тренировочными примерами v_i и сгенерированными примерами h_j , которую мы обозначим как $\langle v_i, h_j \rangle_{pos}$. Эта корреляция, по сути, представляет собой усредненное значение произведения всех пар видимых (v_i) и скрытых (h_j) элементов.

- *Негативная фаза.* Во время негативной фазы работа алгоритма начинается с мини-пакета тренировочных примеров. Для каждого тренировочного примера выполняется фаза семплирования по Гиббсу, начинающаяся со случайно инициализированных состояний. Это достигается за счет многократного применения уравнений 6.15 и 6.17 для вычисления вероятностей видимых и скрытых элементов и использования полученных вероятностей для извлечения выборочных значений. Значения v_i и h_j при тепловом равновесии используются для вычисления величины $\langle v_i, h_j \rangle_{neg}$ тем же способом, что и при прохождении позитивной фазы.
- Последующие обновления осуществляются по тем же правилам, которые используются в случае машин Больцмана:

$$\begin{aligned} w_{ij} &\leftarrow w_{ij} + \alpha (\langle v_i, h_j \rangle_{pos} - \langle v_i, h_j \rangle_{neg}), \\ b_i^{(v)} &\leftarrow b_i^{(v)} + \alpha (\langle v_i, 1 \rangle_{pos} - \langle v_i, 1 \rangle_{neg}), \\ b_j^{(h)} &\leftarrow b_j^{(h)} + \alpha (\langle 1, h_j \rangle_{pos} - \langle 1, h_j \rangle_{neg}), \end{aligned}$$

где $\alpha > 0$ — скорость обучения. Каждая величина $\langle v_i, h_j \rangle$ оценивается путем усреднения произведений v_i и h_j по всему мини-пакету, хотя значения v_i и h_j вычисляются разными способами при прохождении позитивной и негативной фаз соответственно. Кроме того, величина $\langle v_i, 1 \rangle$ представляет усредненное по мини-пакету значение v_i , а величина $\langle 1, h_j \rangle$ — усредненное по мини-пакету значение h_j .

Приведенные выше формулы для обновлений полезно интерпретировать в терминах мороженого из примера, приведенного на рис. 6.3, б. Если между весами некоторых видимых битов (например, рожки и эскимо) существует значительная корреляция, то вышеприведенные формулы обновлений будут стремиться смещать веса в тех направлениях, в которых эти корреляции могут быть объяснены весами связей между фургонами и видами мороженого. Например, если корреляции между рожками и эскимо — сильные, тогда как все остальные — слабые, то это можно объяснить большими значениями весов связей

между этими двумя видами мороженого и одним фургоном. На практике корреляции будут гораздо более сложными, равно как и соотношения между базовыми весами.

С описанным выше подходом связана одна трудность, которая заключается в том, что для получения теплового равновесия и генерирования выборочных значений негативной фазы приходится в течение некоторого времени выполнять семплирование по методу Монте-Карло. Однако оказывается, что если *начинать с видимых состояний, зафиксированных на тренировочных точках данных из мини-пакета*, то семплирование по методу Монте-Карло достаточно выполнять лишь в течение короткого промежутка времени и при этом все еще получать хорошее приближение для градиента.

6.4.2. Алгоритм контрастивной дивергенции

В самом быстром варианте подхода на основе *контрастивной дивергенции* (contrastive divergence, CD) для генерирования выборочных значений видимых и скрытых состояний используется всего *одна* дополнительная (по отношению к позитивной фазе) итерация семплирования по методу Монте-Карло. Сначала, зафиксировав видимые элементы на тренировочной точке, генерируются скрытые состояния (что в любом случае делается во время позитивной фазы), вслед за чем, используя семплирование по методу Монте-Карло, на основе этих скрытых состояний вновь (но только однократно) генерируются видимые состояния. Результирующие значения видимых элементов используются в качестве выборочных состояний вместо тех, которые получаются при достижении теплового равновесия. Далее эти видимые элементы вновь используются для генерирования скрытых элементов. Таким образом, основным различием между позитивной и негативной фазами является лишь количество итераций, выполняемых при одних и тех же начальных значениях видимых состояний, зафиксированных на тренировочных точках. Во время позитивной фазы мы используем лишь половину итерации простого вычисления скрытых состояний. Во время негативной фазы мы используем по крайней мере одну *дополнительную* итерацию (для пересчета видимых состояний на основе скрытых и повторного генерирования скрытых состояний). Именно эта разница в количестве итераций приводит к контрастивной дивергенции (расхождению) между распределениями состояний в этих двух случаях. Интуиция подсказывает, что увеличение количества итераций приводит к тому, что распределение смещается (что и создает расхождение) от состояний, обусловленных данными, к состояниям, предлагаемым текущим вектором весов. Поэтому значение $(\langle v_i, h_j \rangle_{pos} - \langle v_i, h_j \rangle_{neg})$ в формуле обновления является количественной мерой контрастивной дивергенции. Этот самый быстрый вариант алгоритма контрастивной дивергенции называется CD_1 , поскольку для генерации выборочных значений во время

негативной фазы используется всего одна (дополнительная) итерация. Разумеется, использование такого подхода является лишь приближением к истинному градиенту. Точность контрастивной дивергенции можно повысить, увеличив количество дополнительных итераций до k , когда данные перевычисляются k раз. Такой вариант подхода получил название CD_k . Увеличение k улучшает градиенты за счет потери скорости вычислений.

Использование варианта CD_1 на ранних итерациях дает довольно неплохие результаты, однако может не обеспечивать дополнительного улучшения результатов на более поздних стадиях. Поэтому вполне естественным представляется подход, предполагающий постепенное увеличение значения k при использовании варианта CD_k в процессе тренировки. Процесс в целом можно описать следующим образом.

1. На начальной стадии градиентного спуска веса инициализируются небольшими значениями. На каждой итерации используется лишь один дополнительный шаг контрастивной дивергенции. На этом этапе достаточно одной итерации, поскольку на первых итерациях разница между весами весьма неточная и требуется лишь грубое определение направления спуска. Поэтому в большинстве случаев даже использование варианта CD_1 обеспечивает получение удовлетворительного направления градиентного спуска.
2. По мере приближения градиентного спуска к лучшему решению требуется более высокая точность. В связи с этим используют два-три шага контрастивной дивергенции (т.е. варианты CD_2 или CD_3). Вообще говоря, количество шагов алгоритма марковских цепей можно удваивать после выполнения некоего фиксированного количества шагов градиентного спуска. Другой подход, рекомендованный в [469], заключается в увеличении значения k в варианте CD_k на 1 после каждых 10000 шагов. Максимальное значение k , которое использовалось в [469], составляло 20.

Алгоритм контрастивной дивергенции может быть расширен на многие другие вариации RBM. Великолепное практическое руководство по обучению ограниченных машин Больцмана содержится в [193]. В нем обсуждаются такие вопросы, как инициализация, настройка и правила обновления переменных. Краткий обзор практических рекомендаций приведен в следующем разделе.

6.4.3. Практические рекомендации и возможные видоизменения процедуры

В процессе обучения RBM с помощью контрастивной дивергенции придется решать ряд вопросов технического характера. Несмотря на то что мы всегда предполагали, что процедура семплирования по методу Монте-Карло

генерирует бинарные выборочные значения, это не всегда так. На некоторых итерациях семплирования по методу Монте-Карло непосредственно используются *вычисленные* вероятности (см. уравнения 6.15 и 6.17), а не *семплированные* бинарные значения. Это делается для того, чтобы уменьшить уровень шума в процессе обучения, поскольку вероятностные значения удерживают больше информации по сравнению с бинарными выборочными значениями. Однако способы обработки скрытых и видимых состояний несколько различаются.

- *Импровизации в семплировании скрытых состояний.* В варианте CD_k на последней итерации скрытые состояния вычисляются для позитивных и негативных выборок как вероятностные значения в соответствии с уравнением 6.15. Поэтому значение h_j , используемое для вычисления $\langle v_i, h_j \rangle_{pos} - \langle v_i, h_j \rangle_{neg}$, всегда будет вещественным для позитивных и негативных выборок. Ввиду использования сигмоиды в уравнении 6.15 это вещественное значение является дробным.
- *Импровизации в семплировании видимых состояний.* Видоизменения в семплировании видимых состояний по методу Монте-Карло всегда связаны с вычислением $\langle v_i, h_j \rangle_{neg}$, а не $\langle v_i, h_j \rangle_{pos}$, поскольку видимые состояния всегда фиксируются на тренировочных данных. Для негативных выборочных значений процедура Монте-Карло *всегда* вычисляет вероятностные значения видимых состояний на *всех* итерациях в соответствии с уравнением 6.17, а не использует значения 0 и 1. В случае скрытых состояний, которые всегда остаются бинарными вплоть до самой последней итерации, это не так.

Итеративное использование вероятностных значений вместо семплированных бинарных значений с технической точки зрения не является корректным и не обеспечивает достижения корректного теплового равновесия. Однако алгоритм контрастной дивергенции в любом случае является приближенным, а подход такого типа в значительной степени снижает уровень шумов, хотя и за счет некоторой теоретической некорректности. Снижение уровня шумов является следствием того факта, что вероятностные выходы обеспечивают более близкое соответствие ожидаемым значениям.

Весы могут инициализироваться значениями из гауссовского распределения с нулевым средним и стандартным отклонением 0,01. Большие значения весов могут ускорить обучение, но также могут приводить к несколько худшей результирующей модели. Видимые смещения инициализируются значениями $\log(p_i / (1 - p_i))$, где p_i — доля точек данных, в которых i -е измерение принимает значение 1. Скрытые смещения инициализируются нулевыми значениями.

Размер мини-пакета должен лежать в пределах от 10 до 100. Очередность предоставления примеров должна рандомизироваться. В тех случаях, когда с

примерами связаны метки классов, мини-пакеты должны выбираться таким образом, чтобы доля меток в пакете была примерно такой же, как и во всем наборе данных.

6.5. Применение ограниченных машин Больцмана

В данном разделе мы рассмотрим несколько примеров применения ограниченных машин Больцмана. Описанные ниже методы доказали свою успешность в целом ряде приложений, основанных на методике обучения без учителя, хотя они также использовались в приложениях, требующих обучения с учителем. На практике часто возникает необходимость в отображении входов на выходы, тогда как RBM в своем простейшем виде предназначена лишь для обучения распределений вероятности. Привязку “вход — выход” часто обеспечивают, конструируя сеть прямого распространения с весами, взятыми из обученной RBM. Иными словами, исходная RBM часто служит для построения *ассоциированной с ней* традиционной нейронной сети.

В этой связи имеет смысл обсудить различия в понятиях *состояния* узла RBM и *активации* этого узла в ассоциированной нейронной сети. Состояние узла — это бинарное значение, семплируемое из распределений Бернулли в соответствии с уравнениями 6.15 и 6.17. С другой стороны, активация узла в ассоциированной нейронной сети — это вероятностное значение, полученное благодаря использованию сигмоиды в уравнениях 6.15 и 6.17. Многие приложения используют активации в узлах ассоциированной нейронной сети, а не состояния исходной RBM после тренировки. Обратите внимание на то, что на окончательном шаге алгоритма контрастивной дивергенции при обновлении весов также используются активации узлов, а не состояния. В практических ситуациях активации более информативны, а значит, более полезны. Использование активаций согласуется с традиционными архитектурами нейронных сетей, в которых может использоваться механизм обратного распространения ошибки. Использование финальной фазы обратного распространения является ключевым фактором, определяющим возможность применения описанного подхода в приложениях на основе обучения с учителем. В большинстве случаев главным предназначением RBM является обучение признакам без учителя. Поэтому роль RBM в приложениях, использующих обучение с учителем, часто ограничивается лишь предварительным обучением. В действительности предварительное обучение является одним из важнейших исторических вкладов RBM.

6.5.1. Снижение размерности и реконструирование данных

Самой базовой функцией RBM является *снижение размерности представления данных* (dimensionality reduction) и *автоматическое конструирование*

признаков (unsupervised feature engineering). Скрытые элементы RBM содержат редуцированное представление данных. Однако мы пока еще не обсуждали, каким образом можно реконструировать исходное представление за счет использования RBM (во многом подобно автокодировщику). Понимание сути этого процесса в первую очередь требует понимания эквивалентности ненаправленной RBM и направленных графических моделей [251], в которых вычисления осуществляются в определенном направлении. Материализация направленного вероятностного графа является первым шагом на пути к материализации традиционной нейронной сети (получаемой на основе RBM), в которой дискретное вероятностное семплирование из сигмоиды может быть заменено вещественными сигмоидными активациями.

Несмотря на то что RBM — ненаправленная графическая модель, ее можно “развернуть” и создать направленную модель, в которой процесс вывода осуществляется в определенном направлении. В общем случае можно показать, что ненаправленная RBM в общем эквивалента направленной графической модели с бесконечным количеством слоев. Развертывание особенно полезно, когда видимые элементы фиксируются на конкретных значениях, поскольку в этом случае развернутая сеть сводится до количества слоев, ровно вдвое превышающего количество слоев в исходной RBM. Кроме того, в результате замены дискретного вероятностного семплирования непрерывными сигмоидными элементами эта направленная модель функционирует как виртуальный автокодировщик, состоящий из двух частей: кодировщика и декодировщика. Несмотря на то что веса RBM обучались с использованием дискретного вероятностного семплирования, их также можно использовать в этой родственной нейронной сети, предварительно выполнив определенную тонкую настройку. Такой эвристический подход позволяет преобразовать веса, обученные в машине Больцмана, в инициализированные веса традиционной нейронной сети с сигмоидными элементами.

RBM можно рассматривать как ненаправленную графическую модель, которая использует для обучения \bar{h} на \bar{v} ту же матрицу весов, что и для обучения \bar{v} на \bar{h} . Если внимательно присмотреться к уравнениям 6.15 и 6.17, то можно увидеть, что они очень похожи. Основное различие состоит в том, что в этих уравнениях используются различные смещения и взаимно транспонированные матрицы весов. Другими словами, уравнения 6.15 и 6.17 можно переписать для некоторой функции $f(\cdot)$ в следующем виде:

$$\begin{aligned}\bar{h} &\sim f(\bar{v}, \bar{b}^{(h)}, W), \\ \bar{v} &\sim f(\bar{h}, \bar{b}^{(v)}, W^T).\end{aligned}$$

В бинарных RBM-машинах, которые составляют преобладающий вариант моделей данного класса, в качестве функции $f(\cdot)$ обычно определяют сигмоиду. Игнорируя смещения, ненаправленный граф RBM можно заменить двумя направленными связями (рис. 6.4, а). Обратите внимание на то, что матрицами весов в двух направлениях являются матрицы W и W^T соответственно. Однако, фиксируя видимые состояния на тренировочных точках, мы можем выполнить всего лишь две итерации этих операций для того, чтобы реконструировать видимые состояния с помощью *вещественных* приближений. Другими словами, мы аппроксимируем обученную RBM традиционной нейронной сетью, заменяя дискретное семплирование сигмоидными активациями с вещественными значениями (в качестве эвристики). Это преобразование представлено на рис. 6.4, б. Иначе говоря, вместо того чтобы использовать операцию семплирования \sim , мы заменяем выборочные значения вероятностными:

$$\bar{h} = f(\bar{v}, \bar{b}^{(h)}, W),$$

$$\bar{v}' = f(\bar{h}, \bar{b}^{(v)}, W^T).$$



а) Эквивалентность направленных и ненаправленных отношений



б) Дискретная графическая модель для аппроксимации нейронной сети вещественными значениями

Рис. 6.4. Использование обученной RBM для аппроксимации обученного автокодировщика

Обратите внимание на то, что \bar{v}' — это реконструированная версия \bar{v} , содержащая вещественные значения (в отличие от бинарных состояний, содержащихся в \bar{v}). В данном случае мы работаем с вещественными активациями, а не с дискретными выборочными значениями. Поскольку семплирование больше не используется и все вычисления выполняются в терминах ожидаемых значений, то для обучения редуцированному представлению нам нужно выполнить

всего одну итерацию уравнения 6.15. Кроме того, для обучения реконструированным данным требуется только одна итерация уравнения 6.17. Фаза предсказания работает лишь в одном направлении: от входной точки до реконструированных данных, что показано на рис. 6.4, б, *справа*. Чтобы определить состояния этой традиционной нейронной сети в виде вещественных значений, мы видоизменяем уравнения 6.15 и 6.17 следующим образом:

$$\hat{h}_j = \frac{1}{1 + \exp(-b_j^{(h)} - \sum_{i=1}^d v_i w_{ij})}. \quad (6.19)$$

В ситуациях, когда общее количество скрытых состояний $m \ll d$, редуцированное представление с вещественными значениями имеет вид $(\hat{h}_1 \dots \hat{h}_m)$. Этот первый шаг создания скрытых состояний эквивалентен той части автокодировщика, которая ответственна за кодирование, и данные значения являются ожидаемыми значениями бинарных состояний. Далее можно реконструировать видимые состояния, применив к этим *вероятностным значениям* уравнение 6.17 (без создания выборок по методу Монте-Карло):

$$\hat{v}_i = \frac{1}{1 + \exp(-b_i^{(v)} - \sum_j \hat{h}_j w_{ij})}. \quad (6.20)$$

Несмотря на то что \hat{h}_j представляет ожидаемое значение j -го элемента, повторное применение сигмоиды к этой версии \hat{h}_j с вещественными значениями дает лишь грубое приближение ожидаемого значения v_i . Тем не менее вещественное предсказание \hat{v}_i аппроксимирует реконструированное значение v_i . Обратите внимание, что для выполнения этой реконструкции мы применили операции, аналогичные тем, которые используются в традиционных нейронных сетях с сигмоидными элементами, а не с доставляющими хлопоты дискретными выборочными значениями вероятностных графических моделей. Таким образом, теперь мы можем использовать эту родственную нейронную сеть в качестве неплохой стартовой точки для тонкой настройки весов с помощью обратного распространения ошибки. Такой тип реконструкции данных аналогичен тому, который применяется в архитектуре автокодировщика, обсуждавшейся в главе 2.

На первый взгляд может показаться, что в обучении RBM нет большого смысла, если те же задачи могут быть решены с помощью традиционного автокодировщика. Однако описанный широкий подход к построению традиционной нейронной сети на основе обученной RBM оказывается особенно полезным при работе с каскадами (стеками) RBM (раздел 6.7). При обучении стеков RBM не приходится сталкиваться с трудностями, возникающими в случае глубоких сетей, особенно с теми, которые обусловлены проблемами затухающих и

взрывных градиентов. Подобно тому как простая RBM может служить отличной начальной точкой для мелкого автокодировщика, каскадные RBM могут стать отличной отправной точкой для глубокого автокодировщика [198]. Этот принцип привел к появлению идеи предварительного обучения с помощью RBM еще до того, как были разработаны общепринятые методы предварительного обучения, не использующие RBM. Как обсуждалось в данном разделе, RBM можно применять в других задачах, требующих снижения размерности данных, таких как коллаборативная фильтрация и тематическое моделирование.

6.5.2. Применение RBM для коллаборативной фильтрации

В предыдущем разделе демонстрировалось использование машин Больцмана в качестве альтернативы автокодировщику для моделирования на основе обучения без учителя и снижения размерности данных. Однако, как обсуждалось в разделе 2.5.7, методы снижения размерности находят еще и целый ряд родственных применений, таких как *коллаборативная фильтрация*. Ниже мы представим RBM-альтернативу методике создания рекомендательных систем, описанной в разделе 2.5.7. Этот подход базируется на методике, предложенной в [414], и использовался в качестве одной из ансамблевых компонент сети, ставшей победителем соревнований, проводимых компанией Netflix.

Одна из трудностей при работе с матрицами рейтингов заключается в том, что они не задаются полностью. Как правило, это затрудняет проектирование нейронной архитектуры для коллаборативной фильтрации по сравнению с традиционными методами снижения размерности. Возможно, вы вспомните из обсуждения в разделе 2.5.7, что аналогичные проблемы возникают и при моделировании подобных неполных матриц с помощью традиционной нейронной сети. Там было показано, каким образом можно создавать разные тренировочные примеры и разные нейронные сети для каждого пользователя, в зависимости от его предпочтений. Все эти различные нейронные сети разделяют веса. Точно такой же подход применяется в ограниченной машине Больцмана, когда для каждого пользователя определяется один тренировочный пример и одна RBM. Однако в случае RBM возникает дополнительная проблема, обусловленная бинарной природой элементов, тогда как рейтинги могут иметь значения от 1 до 5. Поэтому нам нужно найти какой-то способ для работы с этими ограничениями.

Чтобы справиться с указанной проблемой, в RBM разрешено использование скрытых элементов, рассчитанных на классификацию 5 классов в соответствии с тем фактом, что рейтинговые оценки имеют значения от 1 до 5. Другими словами, скрытые элементы определяются в форме прямого кодирования рейтингов. Прямые коды естественно моделировать с помощью функции активации

Softmax, которая определяет вероятности каждой возможной позиции. i -му *Softmax*-элементу соответствует i -й фильм, а вероятность определенного рейтинга, присваиваемого фильму, задается распределением *Softmax*-вероятностей. Поэтому, если учитываются d фильмов, то всего имеется d таких рейтинговых оценок в виде прямых кодов. Бинарные значения видимых элементов, кодируемые с помощью прямых кодов, обозначим как $v_i^{(1)} \dots v_i^{(5)}$.

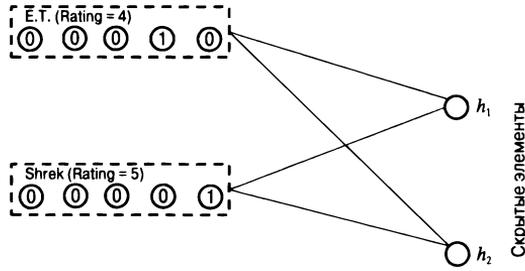
Обратите внимание на то, что только одно из значений $v_i^{(k)}$ может быть равно 1 при фиксированном i и переменном k . Предполагается, что скрытый слой содержит m элементов. Матрица весов имеет отдельный параметр для каждого из мультиномиальных исходов *Softmax*-элемента. Поэтому обозначим вес связи между видимым элементом i и скрытым элементом j для исхода k как $w_{ij}^{(k)}$. Кроме того, мы имеем 5 смещений для видимого элемента i , которые обозначим как $b_i^{(k)}$ для $k \in \{1 \dots 5\}$. Каждый скрытый элемент имеет только одно смещение, и мы введем для смещения j -го скрытого элемента обозначение b_j (без верхнего индекса). Архитектура RBM для коллаборативной фильтрации приведена на рис. 6.5. Данный пример включает $d = 5$ фильмов и $m = 2$ скрытых элемента. На рисунке представлены архитектуры RBM для двух пользователей: Алисы и Боба. Алиса оценила только два фильма, поэтому общее количество связей в данном случае равно $2 \times 2 \times 5 = 20$, хотя для того, чтобы не загромождать рисунок, на нем отображены только некоторые из них. Боб оценил четыре фильма, поэтому его сеть будет содержать $4 \times 2 \times 5 = 40$ связей. Обратите внимание на то, что фильм *Инопланетянин* (E.T.) оценен как Алисой, так и Бобом, и поэтому связи данного фильма со скрытыми элементами будут разделять веса между соответствующими RBM.

Состояния скрытых элементов, являющиеся бинарными, определяются с использованием сигмоиды:

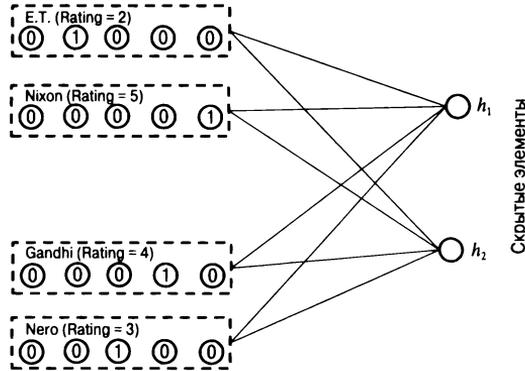
$$P(h_j = 1 | \bar{v}^{(1)} \dots \bar{v}^{(5)}) = \frac{1}{1 + \exp(-b_j - \sum_{i,k} v_i^{(k)} w_{ij}^{(k)})}. \quad (6.21)$$

Основное отличие от уравнения 6.15 заключается в том, что видимые элементы содержат верхний индекс, соответствующий различным рейтинговым исходам. Во всех остальных отношениях условия практически идентичны. Однако вероятности видимых элементов определяются иначе, чем в традиционной модели RBM. В данном случае видимые элементы определяются с использованием функции *Softmax*:

$$P(v_i^{(k)} = 1 | \bar{h}) = \frac{\exp(b_i^{(k)} + \sum_j h_j w_{ij}^{(k)})}{\sum_{r=1}^5 \exp(b_i^{(r)} + \sum_j h_j w_{ij}^{(r)})}. \quad (6.22)$$



а) Архитектура RBM для Алисы (наблюдаемые рейтинговые оценки: *E.T.* и *Shrek*)



б) Архитектура RBM для Боба (наблюдаемые рейтинговые оценки: *E.T.*, *Nixon*, *Gandhi* и *Nero*)

Рис. 6.5. Архитектуры RBM двух пользователей показаны на основе присвоенных ими рейтинговых оценок. Будет поучительно сравнить эти архитектуры с обычной нейронной архитектурой, приведенной на рис. 2.14. В обоих случаях веса разделяются сетями отдельных пользователей

Обучение осуществляется аналогично тому, как это делается в неограниченной машине Больцмана с семплированием по методу Монте-Карло. Основное отличие заключается в том, что видимые состояния генерируются из мультиномиальной модели. Поэтому семплирование МСМС должно генерировать негативные выборки из мультиномиальной модели уравнения 6.22 для создания каждого $v_i^{(k)}$. Соответствующие обновления для обучения весов выглядят так:

$$w_{ij}^{(k)} \leftarrow w_{ij}^{(k)} + \alpha \left(\langle v_i^{(k)}, h_j \rangle_{pos} - \langle v_i^{(k)}, h_j \rangle_{neg} \right) \quad \forall k. \quad (6.23)$$

Обратите внимание на то, что для одного тренировочного примера (т.е. пользователя) обновляются только веса связей *наблюдаемых* видимых элементов со всеми скрытыми элементами. Другими словами, машины Больцмана

используют для разных пользователей разные данные, хотя веса разделяются различными пользователями. Примеры машин Больцмана для двух различных тренировочных примеров приведены на рис. 6.5, причем архитектуры для Боба и Алисы отличаются. Однако веса для элементов, представляющих фильм *Инопланетянин* (E.T.), разделяются. Подход такого же типа уже встречался нам в традиционной нейронной архитектуре, описанной в разделе 2.5.7, где для каждого тренировочного примера использовалась своя нейронная сеть. Как обсуждалось в том разделе, традиционная нейронная архитектура эквивалентна методу матричной факторизации. Обычно предсказания рейтинговых оценок, получаемые с помощью машины Больцмана, несколько отличаются от предсказаний, получаемых методом матричной факторизации, хотя точность в обоих случаях примерно одинакова.

Прогнозирование

Обученные веса можно использовать в целях прогнозирования. Однако фаза предсказаний работает не с бинарными состояниями, а с вещественными активациями, во многом напоминая традиционную сеть, элементы которой активируются сигмоидой или функцией *Softmax*. Прежде всего, можно использовать уравнение 6.21 для обучения вероятностей скрытых элементов. Обозначим через \hat{p}_j вероятность того, что j -й скрытый элемент равен 1. Тогда вероятности *не наблюдаемых* видимых элементов можно вычислить с помощью уравнения 6.22. Основной проблемой уравнения 6.22 является то, что оно определено в терминах значений скрытых элементов, которые известны лишь в форме вероятностей, которые вычисляются в помощью уравнения 6.21. Однако для вычисления вероятностей видимых элементов можно просто подставить \hat{p}_j вместо h_j в уравнении 6.22. Получаемые при этом предсказания представляют вероятности возможных рейтинговых оценок для каждого фильма. Эти вероятности также можно использовать для вычисления ожидаемого значения рейтинговой оценки, если в этом возникнет необходимость. Несмотря на то что с теоретической точки зрения такой подход является приближенным, практика показала, что он работает вполне удовлетворительно и обеспечивает очень высокую скорость вычислений. Использование описанных вычислений с вещественными значениями обеспечивает эффективное преобразование RBM в традиционную нейронную архитектуру с логистическими элементами для скрытых слоев и *Softmax*-элементами для входного и выходного слоев. И хотя в оригинальной статье [414] об этом не упомянуто, при таком подходе существует даже возможность настройки весов этой сети с помощью алгоритма обратного распространения ошибки (упражнение 1).

Описанный подход на основе RBM работает так же хорошо, как и традиционная матричная факторизация, хотя и выдает предсказания другого типа. Такое

разнообразие возможностей открывает широкие перспективы для построения ансамблевых моделей. Получаемые результаты можно комбинировать с подходом на основе матричной факторизации, тем самым обеспечивая улучшение результатов, естественным образом связанное с использованием ансамблевого метода. Обычно ансамблевые методы демонстрируют лучшие результаты, если в них используются разные методы примерно одинаковой точности.

Условная факторизация: красивый трюк с регуляризацией

В работе [414], посвященной коллаборативной фильтрации на основе RBM, содержится описание красивого трюка с регуляризацией. Он не является специфичным для задач коллаборативной фильтрации, и его можно задействовать в любом приложении RBM. Этот подход не обязательно применять в традиционных нейронных сетях, в которых его можно имитировать, включив дополнительный скрытый слой, однако его использование в RBM оказывается полезным. Ниже этот подход описывается в более общей форме, без учета видоизменений, специфических для задач коллаборативной фильтрации. В некоторых приложениях с большим количеством скрытых и видимых элементов размер матрицы параметров $W = [w_{ij}]$ может быть очень большим. Например, в случае матрицы с количеством видимых элементов $d = 10^5$ и количеством скрытых элементов $m = 100$ мы будем иметь десять миллионов параметров. Поэтому, чтобы избежать переобучения, в данном случае потребуется более десяти миллионов тренировочных точек. Возможным выходом из этой ситуации могло бы стать понижение ранга параметрической структуры матрицы весов, являющееся одной из форм регуляризации. Суть идеи заключается в том, чтобы представить матрицу W в виде произведения двух низкоранговых матриц U и V , имеющих размеры $d \times k$ и $m \times k$ соответственно. Таким образом,

$$W = UV^T, \quad (6.24)$$

где k — ранг факторизации, который обычно намного меньше как d , так и m . После этого вместо обучения параметров матрицы W можно обучить параметры матриц U и V . Этот прием часто используется в различных приложениях машинного обучения, в которых параметры представляются матрицами. В качестве конкретного примера можно привести машины факторизации, которые применяются также для коллаборативной фильтрации [396]. В традиционных нейронных сетях подход такого типа не требуется, поскольку его можно симитировать, включив дополнительный линейный слой с k элементами между двумя слоями с матрицей W весов связей между ними. Матрицами весов этих двух слоев будут матрицы U и V^T соответственно.

6.5.3. Использование RBM для классификации

Самый распространенный способ применения RBM для классификации — предварительное обучение. Другими словами, сначала машина Больцмана используется для конструирования признаков без учителя, а затем развертывается в родственную архитектуру “кодировщик — декодировщик” в соответствии с подходом, описанным в разделе 6.5.1. Это традиционная нейронная сеть с сигмоидными элементами, веса которых берутся из обученной без учителя RBM, а не из алгоритма обратного распространения ошибки. Часть сети, ответственная за кодирование, завершается выходным слоем для предсказания классов. Затем веса этой нейронной сети настраиваются с помощью алгоритма обратного распространения. Такой подход можно использовать даже с *каскадными RBM* (раздел 6.7) для получения глубоких классификаторов. Эта методология инициализации (обычных) глубоких сетей с помощью RBM была одним из первых подходов, применяемых для предварительного обучения глубоких сетей.

Однако для классификации с помощью RBM существует альтернативный подход, предполагающий более тесную интеграцию тренировки RBM и получения выводов с процессом классификации. Такой подход отдаленно напоминает методологию коллаборативной фильтрации, о которой шла речь в предыдущем разделе. Задачу коллаборативной фильтрации также называют задачей *заполнения матриц* (matrix completion) ввиду того, что предсказываются значения наблюдений, отсутствующих в частично определенной матрице. Опыт использования RBM в рекомендательных системах дает полезные подсказки относительно возможностей их применения в целях классификации. Это объясняется тем, что классификацию можно рассматривать как упрощенную версию задачи заполнения матрицы, когда мы создаем единственную матрицу, которая включает как тренировочные, так и тестовые строки и в которой отсутствующие значения принадлежат к определенному столбцу матрицы. Этот столбец соответствует переменной класса. Кроме того, в случае классификации все отсутствующие значения находятся в тестовых строках, тогда как в случае рекомендательных систем они могут находиться в матрице где угодно. Указанная взаимосвязь между классификацией и типичной задачей заполнения матрицы проиллюстрирована на рис. 6.6. В случае классификации все наблюдаемые признаки содержатся в строках, соответствующих тренировочным точкам, что упрощает моделирование (по сравнению с коллаборативной фильтрацией, в которой обычно ни одна строка не содержит полный набор признаков).

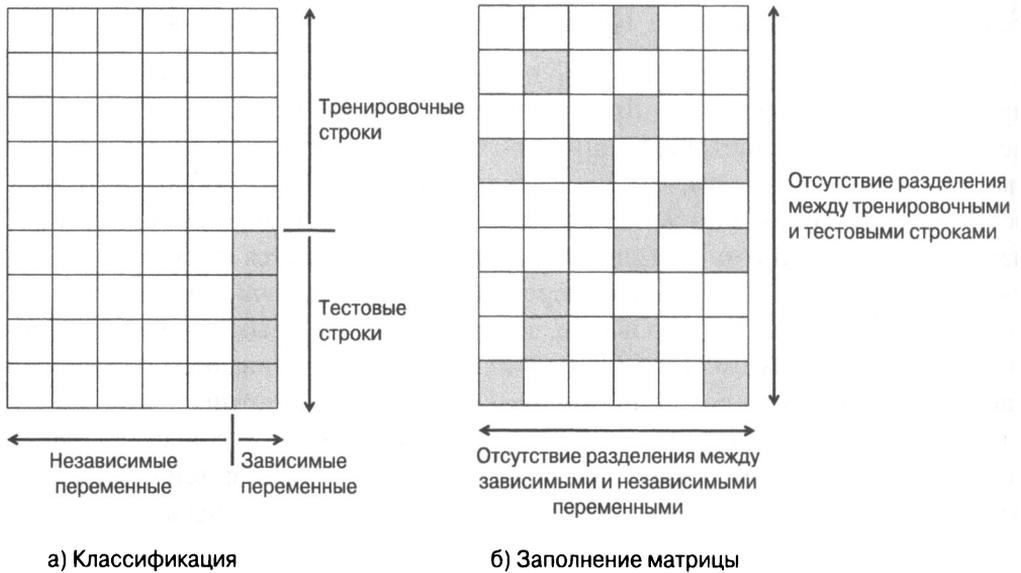


Рис. 6.6. Задача классификации является частным случаем задачи заполнения матрицы. Затенением выделены отсутствующие записи, которые необходимо предсказывать

Мы предполагаем, что входные данные содержат d бинарных признаков. Метка класса имеет k дискретных значений, соответствующих задаче многоклассовой классификации. Задачу классификации можно моделировать с помощью RBM, определив скрытые и видимые признаки в соответствии с приведенным ниже описанием.

1. Видимый слой содержит два типа узлов, соответствующих признакам и меткам классов соответственно. Имеется d бинарных элементов, соответствующих признакам, и k бинарных элементов, соответствующих меткам классов. Однако лишь один из этих бинарных элементов может принимать значение 1, которое соответствует прямому кодированию меток классов. Такое кодирование меток классов аналогично подходу, используемому для кодирования рейтинговых оценок в задаче коллаборативной фильтрации. Видимые элементы для признаков обозначим как $v_1^{(f)} \dots v_d^{(f)}$, а видимые элементы для меток классов — как $v_1^{(c)} \dots v_k^{(c)}$. Обратите внимание на то, что верхние индексы указывают, соответствуют ли видимые элементы признакам (f) или меткам классов (c).
2. Скрытый слой содержит m бинарных элементов. Обозначим их как $h_1 \dots h_m$.

Обозначим через w_{ij} вес связи между i -м видимым элементом $v_i^{(f)}$ и j -м скрытым элементом h_j . Это приводит к матрице весов связей $W = [w_{ij}]$ размером

$d \times m$. Вес связи между i -м видимым элементом $v_i^{(c)}$ и j -м скрытым элементом h_j обозначим через u_{ij} . Это приводит к матрице весов связей $U = [u_{ij}]$ размера $k \times m$. Отношения между различными типами узлов и матрицами для $d = 6$ признаков, $k = 3$ классов и $m = 5$ скрытых признаков отображены на рис. 6.7. Смещение для i -го видимого узла, соответствующего признаку, обозначим через $b_i^{(f)}$, а смещение для i -го видимого узла, соответствующего метке класса, — через $b_i^{(c)}$. Смещение для j -го скрытого узла обозначим через b_j (без верхнего индекса). Состояния скрытых узлов определяются в терминах всех видимых узлов с использованием сигмоид:

$$P(h_j = 1 | \bar{v}^{(f)}, \bar{v}^{(c)}) = \frac{1}{1 + \exp(-b_j - \sum_{i=1}^d v_i^{(f)} w_{ij} - \sum_{i=1}^k v_i^{(c)} u_{ij})}. \quad (6.25)$$

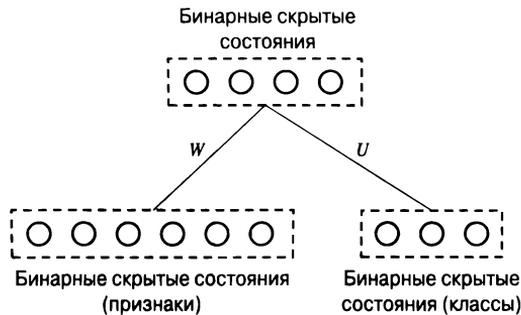


Рис. 6.7. Архитектура RBM для задач классификации

Отметим, что это стандартный подход к определению вероятностей скрытых элементов в машине Больцмана. Впрочем, способы определения вероятностей для видимых элементов, специфических для признаков и классов, несколько отличаются. В случае видимых элементов, специфических для признаков, соответствующее соотношение не очень отличается от того, которое используется в стандартной машине Больцмана:

$$P(v_i^{(f)} = 1 | \bar{h}) = \frac{1}{1 + \exp(-b_i^{(f)} - \sum_{j=1}^m h_j w_{ij})}. \quad (6.26)$$

Однако в случае элементов, соответствующих классам, существуют определенные отличия, поскольку вместо сигмоиды мы должны использовать функцию *Softmax*. Это обусловлено прямым кодированием классов. Поэтому имеем следующее соотношение:

$$P(v_i^{(c)} = 1 | \bar{h}) = \frac{\exp(b_i^{(c)} + \sum_j h_j u_{ij})}{\sum_{l=1}^k \exp(b_l^{(c)} + \sum_j h_j u_{lj})}. \quad (6.27)$$

В случае применения наивного подхода к обучению машины Больцмана использовалась бы модель, аналогичная генеративной модели, рассмотренной в предыдущих разделах. Генерирование видимых состояний $v_i^{(c)}$ для классов осуществляется с помощью мультиномиальной модели. Соответствующие обновления алгоритма контрастивной дивергенции выглядят следующим образом:

$$w_{ij} \leftarrow w_{ij} + \alpha (\langle v_i^{(f)}, h_j \rangle_{pos} - \langle v_i^{(f)}, h_j \rangle_{neg}), \text{ если } i \text{ — элемент признака};$$

$$u_{ij} \leftarrow u_{ij} + \alpha (\langle v_i^{(c)}, h_j \rangle_{pos} - \langle v_i^{(c)}, h_j \rangle_{neg}), \text{ если } i \text{ — элемент класса}.$$

Этот подход является прямым расширением коллаборативной фильтрации. Однако основная проблема заключается в том, что данный *генеративный* подход не выполняет полную оптимизацию, способную обеспечить точную классификацию. Для того чтобы создать нечто аналогичное автокодировщикам, во все не обязательно значительно улучшать снижение размерности (в смысле обучения с учителем) простым включением переменной класса в число входных данных. Над этим снижением размерности часто будут доминировать отношения между признаками, сформированные в процессе обучения без учителя. Вместо этого обучение должно полностью фокусироваться на оптимизации точности классификации. Поэтому для обучения RBM часто используется *дискриминантный* подход, в котором обучение весов основано на максимизации функции условного правдоподобия истинного класса. Обратите внимание на то, что при заданных видимых классах нетрудно установить условную вероятность переменной класса, используя вероятностные зависимости между скрытыми признаками и классами/признаками. Например, в традиционной ограниченной машине Больцмана мы максимизируем совместную вероятность переменных признака $v_i^{(f)}$ и переменных класса $v_i^{(c)}$. Однако в дискриминантном варианте, который мы рассматриваем, целевая функция настраивается для максимизации *условной* вероятности $P(v_y^{(c)} = 1 | \bar{v}^{(f)})$ переменной класса $y \in \{1 \dots k\}$. Такой подход в большей степени сфокусирован на максимизации точности классификации. Несмотря на возможность обучения дискриминантной ограниченной машины Больцмана с помощью контрастивной дивергенции, эта проблема упрощается, поскольку величину $P(v_y^{(c)} = 1 | \bar{v}^{(f)})$ можно оценить в замкнутой форме без использования итеративного подхода. Можно показать, что эта форма имеет следующий вид [263, 414]:

$$P(v_y^{(c)} = 1 | \bar{v}^{(f)}) = \frac{\exp(b_y^{(c)}) \prod_{j=1}^m [1 + \exp(b_j^{(h)} + u_{yj} + \sum_i w_{ij} v_i^{(f)})]}{\sum_{l=1}^k \exp(b_l^{(c)}) \prod_{j=1}^m [1 + \exp(b_j^{(h)} + u_{lj} + \sum_i w_{ij} v_i^{(f)})]}. \quad (6.28)$$

Располагая этой замкнутой дифференцируемой формой, можно без труда вычислить производную отрицательного логарифма вышеприведенного выражения для стохастического градиентного спуска. Если \mathcal{L} — указанный отрицательный логарифм, а θ — любой конкретный параметр (например, вес или смещение) машины Больцмана, то можно показать, что выполняется следующее соотношение:

$$\frac{\partial \mathcal{L}}{\partial \theta} = \sum_{j=1}^m \text{сигмоида}(o_{yj}) \frac{\partial o_{yj}}{\partial \theta} - \sum_{l=1}^k \sum_{j=1}^m \text{сигмоида}(o_{lj}) \frac{\partial o_{lj}}{\partial \theta}, \quad (6.29)$$

где $o_{yj} = b_j^{(h)} + u_{yj} + \sum_i w_{ij} v_i^{(f)}$. Приведенное выше выражение можно легко вычислить для каждой тренировочной точки и каждого параметра и выполнить процесс стохастического градиентного спуска. С помощью уравнения 6.28 можно относительно легко генерировать вероятностные предсказания для неизвестных тестовых примеров. Более подробно этот подход и его расширения обсуждаются в [263].

6.5.4. Тематическое моделирование с помощью RBM

Тематическое моделирование (topic modeling) — это разновидность снижения размерности, специфическая для текстовых данных. Самые ранние тематические модели, которые соответствуют *вероятностному латентно-семантическому анализу* (Probabilistic Latent Semantic Analysis — PLSA), были предложены в [206]. В PLSA базовые векторы не являются взаимно ортогональными, как это имеет место в SVD. С другой стороны, как базовые векторы, так и преобразованные представления ограничены неотрицательными значениями. Неотрицательность значений каждого преобразованного признака полезна с семантической точки зрения, поскольку она представляет относительную долю темы в конкретном документе. В контексте RBM эта доля соответствует вероятности того, что определенный скрытый элемент принимает значение 1, при условии, что соответствующие слова встречаются в определенном документе. Поэтому мы можем использовать вектор условных вероятностей скрытых состояний (когда видимые состояния фиксированы на словах документа) для создания редуцированного представления каждого документа. Предполагается, что размер словаря равен d , а количество скрытых элементов $m \ll d$.

Этот подход имеет некоторые общие черты с методикой, используемой для коллаборативной фильтрации, в которой для каждого пользователя (строки матрицы) создается отдельная RBM. В данном случае отдельные RBM создаются для каждого документа. Для каждого слова создается группа видимых элементов, поэтому количество групп видимых элементов равно количеству слов в документе. Ниже определено, каким образом видимые и скрытые состояния RBM фиксируются для описания того, как работает модель.

1. Для t -го документа, содержащего n_t слов, в общей сложности удерживается n_t Softmax-групп. Каждая Softmax-группа содержит d узлов, соответствующих d словам словаря. Следовательно, для каждого документа используется отдельная RBM, поскольку количество элементов зависит от длины документа. Однако все Softmax-группы в документе и во всей совокупности документов разделяют веса их связей со скрытыми элементами. Позиция i в документе соответствует i -й группе видимых Softmax-элементов. Обозначим эту группу как $v_i^{(1)} \dots v_i^{(d)}$. Смещение, ассоциированное с $v_i^{(k)}$, обозначим как $b^{(k)}$. Обратите внимание на то, что i -й видимый узел зависит только от k (идентификатор слова), а не от i (позиция слова в документе). Это объясняется тем, что в данной модели используется “мешок слов”, позиции слов в котором не играют никакой роли.
2. Имеется m скрытых элементов, которые обозначим как $h_1 \dots h_m$; смещение j -го скрытого элемента обозначим как b_j .
3. Каждый скрытый элемент связан с каждым из $n_t \times d$ видимых элементов. Все Softmax-группы в каждой отдельной RBM, а также во всей совокупности RBM (соответствующих различным элементам) разделяют один и тот же набор d весов. k -й скрытый элемент связан с группой из d Softmax-элементов вектором из d весов, который мы обозначим как $\bar{w}^{(k)} = (w_1^{(k)} \dots w_d^{(k)})$. Иными словами, k -й скрытый элемент связан с каждой из n_t групп, состоящих из d Softmax-элементов, одним и тем же набором весов $\bar{w}^{(k)}$.

Архитектура RBM приведена на рис. 6.8. Основываясь на ней, можно выразить вероятности, ассоциированные со скрытыми узлами, используя сигмоиду:

$$P(h_j = 1 | \bar{v}^{(1)} \dots \bar{v}^{(d)}) = \frac{1}{1 + \exp(-b_j - \sum_{i=1}^{n_t} \sum_{k=1}^d v_i^{(k)} w_j^{(k)})}. \quad (6.30)$$

Вероятности видимых состояний можно выразить, используя мультиномиальную модель:

$$P(v_i^{(k)} = 1 | \bar{h}) = \frac{\exp(b^{(k)} + \sum_{j=1}^m w_j^{(k)} h_j)}{\sum_{l=1}^d \exp(b^{(l)} + \sum_{j=1}^m w_j^{(l)} h_j)}. \quad (6.31)$$

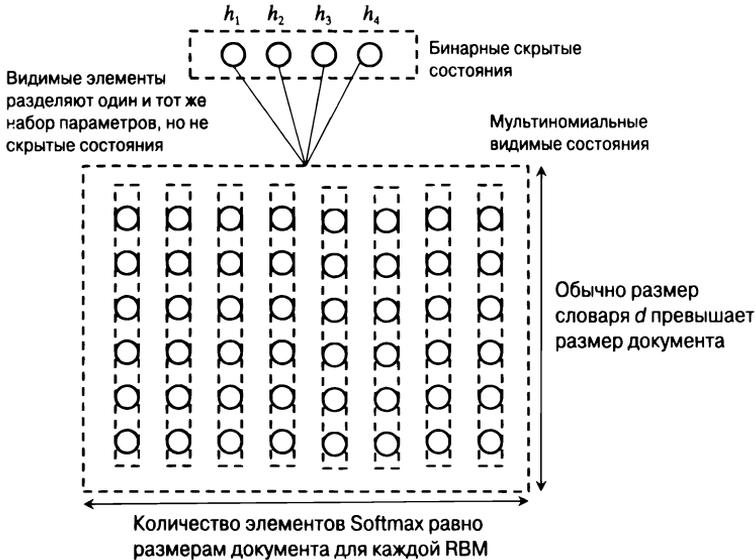


Рис. 6.8. Пример использования RBM для каждого документа. Количество видимых элементов равно количеству слов в каждом документе

Нормирующий множитель в знаменателе гарантирует, что сумма вероятностей видимых элементов по всем словам всегда будет равна 1. Кроме того, выражение в правой части этого уравнения не зависит от индекса i видимого элемента. Так происходит потому, что данная модель не зависит от позиции слов в документе, и в процессе моделирования документ рассматривается как “мешок слов”.

Располагая этими соотношениями, можно сгенерировать выборочные значения для алгоритма контрастивной дивергенции, применяя семплирование MCMC. Заметьте, что для разных документов используются разные RBM, несмотря на то что эти RBM разделяют веса. Как и в случае коллаборативной фильтрации, каждая RBM ассоциируется только с одним тренировочным примером, соответствующим документу. Для градиентного спуска используются те же правила обновления весов, что и для традиционных RBM. Единственное отличие состоит в том, что веса разделяются различными видимыми элементами. Этот подход аналогичен тому, который применяется в коллаборативной

фильтрации. Получение правил обновления весов мы оставляем для читателей в качестве упражнения (см. упражнение 5).

По завершении обучения редуцированное представление каждого документа вычисляется путем применения уравнения 6.30 к словам документа. Вещественные значения вероятностей скрытых элементов дают m -мерное редуцированное представление документа. Описанный в этом разделе подход представляет собой упрощение подхода, описанного в [469].

6.5.5. Использование RBM для машинного обучения с мультимодальными данными

Машины Больцмана также могут быть использованы для машинного обучения с *мультимодальными данными*. Этот термин относится к ситуациям, когда информация извлекается из точек данных, представленных в разнообразных видах и форматах — *модальностях* (modalities). Например, изображение с текстовым описанием может рассматриваться как мультимодальные данные. В этом конкретном случае объект данных имеет модальности в виде изображения и текста.

При обработке мультимодальных данных часто возникают трудности, связанные с использованием алгоритмов машинного обучения на таких гетерогенных признаках. Мультимодальные данные часто обрабатывают с использованием разделяемых представлений, в которых две моды отображаются на совместное пространство. В подобных случаях общепринято использовать подход на основе *разделяемой матричной факторизации* (shared matrix factorization). Многочисленные методы, в которых разделяемая матричная факторизация используется при работе с текстом и изображениями, обсуждаются в [6]. Поскольку во многих ситуациях RBM могут служить альтернативой методам матричной факторизации, вполне естественно исследовать вопрос о том, можно ли применять эту архитектуру для создания разделяемого латентного представления данных.

Описанный в [468] пример архитектуры для мультимодального моделирования приведен на рис. 6.9, *а*. В этом примере предполагается, что две моды соответствуют текстовым данным и изображениям. Оба вида данных используются для создания скрытых состояний, специфических для изображений и текста соответственно. Затем эти скрытые состояния передаются одному общему представлению. Такая архитектура удивительно похожа на архитектуру, приведенную на рис. 6.7. Это обусловлено тем, что обе архитектуры пытаются отобразить два типа признаков на один набор разделяемых скрытых состояний. Полученные скрытые состояния можно использовать для различных видов статистического вывода, например для использования разделяемого представления

в целях классификации. Как показано в разделе 6.7, существует даже возможность улучшения подобных представлений, полученных по методике обучения без учителя, за счет использования алгоритма обратного распространения ошибки для тонкой настройки описанного подхода. Эта модель позволяет генерировать отсутствующие модальности.



а) Простая RBM для мультимодальных данных



б) Мультимодальная RBM с добавленным скрытым слоем

Рис. 6.9. Архитектура RBM для обработки мультимодальных данных

Существует дополнительная возможность улучшения выразительной силы модели за счет использования глубины. Между видимыми состояниями и разделяемым представлением можно вставить дополнительный скрытый слой (рис. 6.9, б) или даже несколько скрытых слоев для создания глубокой сети. Однако пока что мы ничего не сказали о том, как фактически обучить многослойную RBM. Этот вопрос обсуждается в разделе 6.7.

Еще одна трудность использования мультимодальных данных обусловлена тем, что признаки зачастую не являются бинарными. Существует несколько способов решения этой проблемы. В случае текста (или модальностей данных с небольшим количеством дискретных атрибутов) можно использовать подход, аналогичный тому, который применялся в RBM для тематического моделирования, когда счетчик s дискретных атрибутов используется для создания s экземпляров атрибута с прямым кодированием. Эта проблема еще более обостряется, когда данные содержат произвольные вещественные значения. Одно из возможных решений — дискретизация данных, хотя такой подход может привести к потере полезной информации о данных. Еще одним решением может быть изменение функции энергии машины Больцмана. Обсуждение некоторых из указанных проблем содержится в следующем разделе.

6.6. Использование RBM с данными, не являющимися бинарными

До сих пор мы фокусировались на использовании RBM для бинарных типов данных. Действительно, подавляющее большинство RBM предназначено именно для таких данных. В случае некоторых типов данных, таких как категориальные или порядковые (например, рейтинговые оценки), можно использовать подход на основе функции *Softmax*, описанный в разделе 6.5.2. Например, использование *Softmax*-элементов в отношении данных о встречаемости слов в документе обсуждается в разделе 6.5.4. Если количество дискретных значений атрибута невелико, то подход на основе функции *Softmax* может быть видоизменен для работы с подобными упорядоченными атрибутами. Однако в случае данных с вещественными значениями эти методы не вполне эффективны. Одним из возможных решений является использование дискретизации для преобразования вещественных данных в дискретные, которые могут быть обработаны с помощью *Softmax*-элементов. Недостатком такого подхода является определенная потеря репрезентативной точности.

Подход, описанный в разделе 6.5.2, дает нам подсказки относительно того, как можно справиться с наличием данных различного типа. Например, категориальные и порядковые данные обрабатываются путем *изменения распределения вероятности* видимых элементов таким образом, чтобы оно лучше соответствовало текущей задаче. В общем случае может потребоваться изменение распределения вероятности не только видимых, но и скрытых элементов. Это обусловлено тем, что природа скрытых элементов зависит от видимых элементов.

В случае данных с вещественными значениями естественным решением является использование гауссовских видимых элементов. Кроме того, вещественные значения имеют и скрытые элементы, в отношении которых мы полагаем, что они содержат функцию активации ReLU. Энергия конкретной комбинации (\bar{v}, \bar{h}) видимых и скрытых элементов дается следующей формулой:

$$E(\bar{v}, \bar{h}) = \underbrace{\sum_i \frac{(v_i - b_i)^2}{2\sigma_i^2}}_{\text{Ограничивающая функция}} - \sum_j b_j h_j - \sum_{i,j} \frac{v_i}{\sigma_i} h_j w_{ij}. \quad (6.32)$$

Вклад смещений видимых элементов в энергию дается *параболической ограничивающей функцией* (parabolic containment function), задачей которой является удержание значения i -го видимого элемента вблизи b_i . Как и в случае других типов машин Больцмана, производные функции энергии по различным переменным также предоставляют производные логарифмических правдоподобий. Это обусловлено тем, что вероятности всегда определяются через экспоненты функции энергии.

С использованием этого подхода связан ряд проблем. Значительной проблемой является его нестабильность по отношению к выбору дисперсионного параметра σ . В частности, обновления видимого слоя становятся слишком малыми, тогда как обновления скрытого слоя — слишком большими. Естественным решением этой дилеммы является использование скрытых элементов в большем количестве, чем видимых. Также общепринято нормализовать входные данные до единичной дисперсии, чтобы значение стандартного отклонения σ видимых элементов можно было положить равным 1. Элементы ReLU модифицируются таким образом, чтобы создать зашумленную версию. В частности, к значениям элементов, еще до того, как они усекаются до неотрицательных значений, добавляется гауссовский шум с нулевым средним и дисперсией $\log(1 + \exp(v))$. Обоснованием использования столь необычной функции активации является то, что она, как это можно показать, эквивалентна *биномиальному элементу* [348, 495], способному кодировать больше информации, чем обычно используемый бинарный элемент. При работе с вещественными данными очень важно реализовать такую возможность. RBM с семплированием вещественных значений по Гиббсу аналогична бинарной RBM, что относится и к обновлениям весов, если генерируются выборки MCMC. Для устранения нестабильности важно поддерживать скорость обучения на низком уровне.

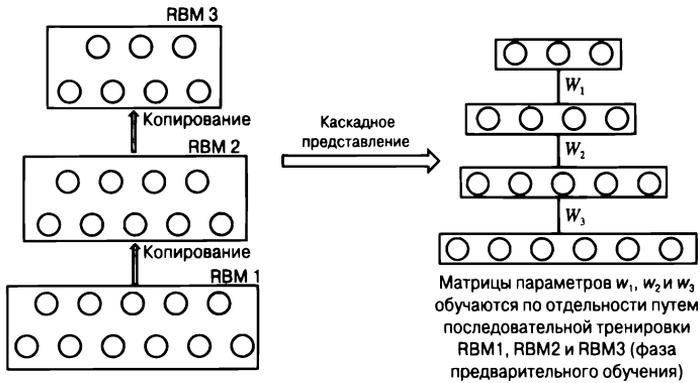
6.7. Каскадные ограниченные машины Больцмана

Большая часть возможностей обычных нейронных архитектур обусловлена наличием в них множества слоев элементов. Известно, что более глубокие сети обладают и большими возможностями. Возникает вполне естественный вопрос: а нельзя ли добиться такого же эффекта путем объединения нескольких RBM? В действительности RBM хорошо приспособлены для создания глубоких сетей и применялись для создания глубоких моделей с предварительным обучением даже *раньше*, чем обычные нейронные сети. Другими словами, RBM тренируются с использованием семплирования по Гиббсу, и результирующие веса внедряются в обычную нейронную сеть с непрерывными сигмоидными активациями (вместо дискретного семплирования на основе сигмоиды). Зачем вообще нужно обременять себя предварительным обучением RBM для тренировки обычной сети? Это объясняется тем фактом, что способ тренировки машин Больцмана кардинально отличается от подхода, основанного на обратном распространении ошибки, являющегося общепринятым в нейронных сетях. Для подхода на основе контрастивной дивергенции характерна совместная тренировка всех слоев, которая не страдает проблемами затухающих и взрывных градиентов, как в случае обычных нейронных сетей.

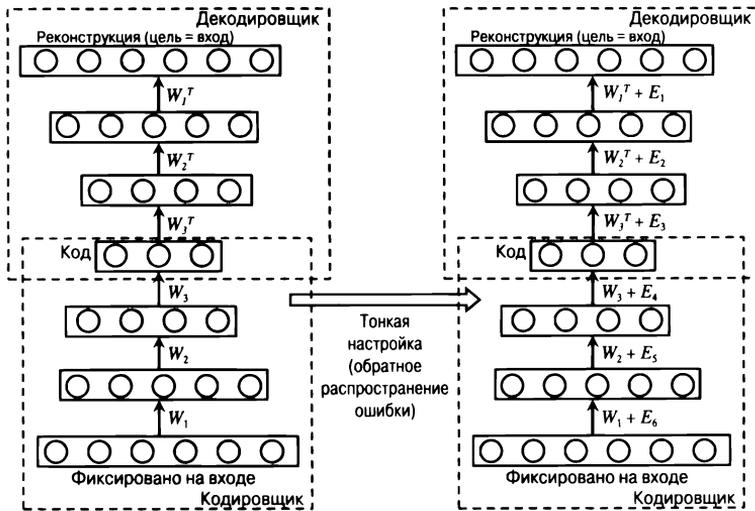
На первый взгляд, создание глубоких сетей из RBM может показаться довольно трудной задачей. Прежде всего, RBM нельзя полностью уподоблять элементам прямого распространения, которые выполняют вычисления в определенном направлении. RBM — симметричные модели, в которых видимые и скрытые элементы объединяются в форме ненаправленной графической модели. Следовательно, необходимо определить конкретный способ, в соответствии с которым несколько RBM взаимодействуют между собой. В этом контексте можно отметить следующее: даже несмотря на тот факт, что RBM — это симметричные и дискретные модели, обученные веса могут быть использованы в родственной нейронной сети, которая выполняет направленные вычисления в непрерывном пространстве активаций. Такие веса уже довольно близки к конечному решению, поскольку они прошли определенное обучение с помощью дискретного семплирования. Поэтому для их тонкой настройки потребуются сравнительно умеренные усилия с использованием традиционного обратного распространения ошибки. Чтобы понять эту точку зрения, рассмотрим однослойную RBM, представленную на рис. 6.4, на котором было показано, что даже однослойная RBM эквивалентна направленной графической модели бесконечной длины. Однако, как только видимые состояния зафиксированы, достаточно оставить только три слоя этого вычислительного графа и выполнять вычисления с непрерывными значениями, полученными от сигмоидных активаций. Такой подход уже сам по себе дает неплохое приближение к решению. Результирующая сеть — это традиционный автокодировщик, хотя его веса были (приблизительно) обучены довольно необычным способом. В этом разделе будет продемонстрировано, что аналогичный подход также может применяться к каскадам RBM.

Итак, что же собой представляет каскадный набор RBM? Рассмотрим набор данных, имеющий d измерений. Нашей целью является создание его редуцированного представления с размерностью, пониженной до m_1 измерений. Этого можно добиться, используя RBM, содержащую d видимых элементов и m_1 скрытых. Обучая эту RBM, можно получить m_1 -мерное представление исходного набора данных. А теперь рассмотрим вторую RBM, имеющую m_1 видимых элементов и m_2 скрытых. Мы можем просто скопировать m_1 выходов первой RBM и использовать их в качестве входов для второй RBM, которая имеет $m_1 \times m_2$ весов. Следовательно, появляется возможность обучения новой RBM для создания m_2 -мерного представления, используя выходы первой RBM в качестве входов. Заметьте, что этот процесс можно повторить k раз, так что последняя RBM будет иметь $(m_k - 1) \times m_k$ весов. Таким образом, мы последовательно обучаем каждую из этих RBM, копируя выход одной на вход другой.

Пример каскада RBM приведен на рис. 6.10, а, слева. Для представления этого типа RBM часто используют компактную диаграмму, приведенную на



а) На стадии предварительного обучения все RBM последовательно тренируются по отдельности



б) После предварительного обучения выполняется тонкая настройка с помощью алгоритма обратного распространения ошибки

Рис. 6.10. Обучение многослойной RBM

рис. 6.10, а, справа. Обратите внимание на то, что копирование данных между двумя RBM представляет собой простую операцию копирования между соответствующими узлами по принципу “один к одному”, поскольку выходной слой r -й RBM имеет в точности такое же количество узлов, как и входной слой $(r + 1)$ -й RBM. Результирующие представления получаются в результате обучения без учителя, поскольку они не зависят от какой-то конкретной цели. Еще один важный момент заключается в том, что машина Больцмана — ненаправленная модель. В то же время, каскадируя машины Больцмана, мы отходим от ненаправленной модели, поскольку верхние слои получают данные от нижних

слоев, но не наоборот. В действительности каждую машину Больцмана можно трактовать как одиночный вычислительный элемент со многими входами и выходами, а операцию копирования из одной машины в другую — как передачу данных между двумя вычислительными элементами. Если исходить из этой специфической точки зрения, рассматривая каскад машин Больцмана как вычислительный граф, и воспользоваться сигмоидными элементами для создания вещественных активаций вместо того, чтобы создавать параметры, необходимые для извлечения бинарных выборок, то становится возможным даже применение алгоритма обратного распространения ошибки. Несмотря на то что использование активаций в виде вещественных чисел является всего лишь приближением, это приближение уже оказывается очень неплохим благодаря способу обучения машины Больцмана. Начальный набор весов можно дополнительно настроить с помощью алгоритма обратного распространения ошибки. В конце концов, обратное распространение применимо к любому вычислительному графу независимо от природы вычисляемой им функции, при условии, что эта функция является непрерывной. Тонкая настройка с помощью алгоритма обратного распространения ошибки приобретает особенно важное значение в случае обучения с учителем, поскольку веса, полученные от машины Больцмана, всегда представляют собой результат обучения без учителя.

6.7.1. Обучение без учителя

Как правило, даже в случае обучения без учителя каскады RBM обеспечивают лучшее качество представлений пониженной размерности, чем одиночные RBM. Однако тренировка такой RBM требует тщательности, поскольку высокое качество результатов не является простым следствием совместного обучения всех слоев. Лучшие результаты удается получать за счет предварительного обучения. Три RBM, представленные на рис. 6.10, *а*, тренируются последовательно. Сначала тренируется RBM 1, которая использует предоставленный набор тренировочных данных в качестве значений видимых элементов. Затем выходы первой RBM используются для тренировки RBM 2. Такой же подход применяется для тренировки RBM 3. Обратите внимание на то, что этот подход обеспечивает возможность “жадной” тренировки любого нужного количества слоев. Предположим, что матрицами весов для трех обучаемых RBM являются W_1 , W_2 и W_3 соответственно. Завершив обучение всех трех матриц, можно сформировать пару “кодировщик — декодировщик” (рис. 6.10, *б*), которая использует эти матрицы. Три декодировщика используют матрицы W_1^T , W_2^T и W_3^T , поскольку выполняют операцию, обратную той, которую выполняют кодировщики. В результате мы получаем направленную сеть “кодировщик — декодировщик”, которую можно тренировать обратным распространением, подобно любой обычной нейронной сети. В этой сети состояния вычисляются с использованием направленной

ных вероятностных операций, а не семплируются с использованием методов Монте-Карло. Для тонкой настройки обучения можно применить обратное распространение ошибки по слоям. Обратите внимание на то, что матрицы весов справа на рис. 6.10, б, изменяются в результате такой тонкой настройки. Кроме того, после выполнения тонкой настройки матрицы весов кодировщика и декодировщика уже не будут симметричными. Подобные каскадированные RBM обеспечивают снижение размерности более высокого качества по сравнению с мелкими RBM [414], что аналогично поведению обычных нейронных сетей.

6.7.2. Обучение с учителем

Существует ли возможность организовать обучение весов таким образом, чтобы стимулировать машину Больцмана к получению выхода определенного типа, например меток классов? Предположим, вы хотите выполнить классификацию по k классам, используя каскадные машины Больцмана. Использование однослойной RBM для классификации уже обсуждалось в разделе 6.5.3, а соответствующая архитектура приведена на рис. 6.7. Эту архитектуру можно видоизменить, заменив одиночный скрытый слой стекком скрытых слоев. Затем последний слой скрытых признаков можно связать с видимым Softmax-слоем, выходом которого являются k вероятностей, соответствующих различным классам. При этом, как и в случае снижения размерности, предобучение оказывается очень полезным. Поэтому первая фаза осуществляется полностью по методике обучения без учителя и не использует метки классов. Иными словами, мы тренируем веса каждого скрытого слоя по отдельности. Это достигается за счет последовательной тренировки весов сначала нижних слоев, а затем более высоких, как в случае любой каскадной RBM. Установив начальные значения весов в соответствии с методикой обучения без учителя, можно выполнить начальную тренировку весов связей, соединяющих последний скрытый слой с видимым слоем Softmax-элементов. После этого можно создать направленный вычислительный граф с этими начальными весами, как в случае обучения без учителя. Далее для тонкой настройки обученных весов используется алгоритм обратного распространения ошибки.

6.7.3. Глубокие машины Больцмана и глубокие сети доверия

Различные слои RBM можно объединять в стеки самыми разными способами для достижения тех или иных целей. В некоторых формах каскадирования взаимодействие различных машин Больцмана является двунаправленным. Этот вариант получил название *глубокая машина Больцмана*. В других формах стеков одни слои однонаправленные, другие — двунаправленные. В качестве примера можно привести *глубокую сеть доверия*, в которой только верхняя RBM

является двунаправленной, тогда как нижние — однонаправленные. Можно показать, что некоторые из этих методов эквивалентны различным типам вероятностных графических моделей, таких как *сигмоидные сети доверия* [350].

Из них особо можно выделить глубокую машину Больцмана ввиду наличия в ней двунаправленных связей между каждой парой элементов. Тот факт, что копирование весов осуществляется обоими способами, означает возможность слияния узлов в смежных узлах двух RBM в один слой. Кроме того, заметьте, что RBM можно преобразовать в двудольный граф, поместив все нечетные слои в один набор, а четные — в другой. Иными словами, глубокая RBM эквивалентна одиночной RBM. Отличие от одиночной RBM состоит в том, что видимые элементы образуют лишь небольшое подмножество элементов одного слоя и все пары узлов не являются связанными. Учитывая последнее, узлы в верхних слоях стремятся получить меньшие веса, чем узлы в нижних слоях. В результате вновь возникает необходимость в предобучении, в процессе которого сначала тренируются нижние слои, а затем — верхние в “жадной” манере. Впоследствии все слои тренируются совместно для тонкой настройки метода. Для получения более подробной информации относительно этих усовершенствованных моделей обратитесь к библиографической справке.

6.8. Резюме

Самым ранним вариантом машины Больцмана была сеть Хопфилда — модель, основанная на понятии энергии, которая сохраняет примеры тренировочных данных в своих локальных минимумах. Сеть Хопфилда можно тренировать с использованием правила обучения Хебба. Машина Больцмана может рассматриваться как стохастический вариант сети Хопфилда, в котором для улучшения обобщающей способности используется вероятностная модель. Кроме того, скрытые состояния машины Больцмана хранят редуцированное представление данных. Машину Больцмана можно тренировать с помощью стохастического варианта правила обучения Хебба. В случае машины Больцмана основной трудностью является то, что она требует применения семплирования по Гиббсу, которое на практике может работать очень медленно. Ограниченная машина Больцмана допускает лишь связи между скрытыми и видимыми узлами, что упрощает процесс обучения. Для такой машины доступны более эффективные алгоритмы обучения. Ее можно использовать в качестве метода снижения размерности, а также в рекомендательных системах с неполными данными. Ограниченная машина Больцмана была обобщена на целочисленные, номинальные и вещественные данные. Однако в преобладающем большинстве случаев RBM по-прежнему строятся в предположении бинарных элементов. В последние годы был предложен ряд глубоких вариантов ограниченной машины

Больцмана, которые могут быть использованы в типичных задачах обычного машинного обучения, таких как классификация.

6.9. Библиографическая справка

Самым ранним вариантом семейства машин Больцмана была сеть Хопфилда [207]. Правило обучения Сторки было предложено в [471]. Самые ранние алгоритмы обучения машин Больцмана с использованием семплирования по методу Монте-Карло были предложены в [1, 197]. Обсуждение семплирования по методу Монте-Карло марковских цепей содержится в [138, 351], и многие из этих методов также применимы к машинам Больцмана. Первоначальный вариант RBM был предложен Смоленским и получил название *фисгармония*. Руководство по моделям на основе энергетического функционала содержится в [280]. Ограниченные машины Больцмана трудно обучать в силу независимой стохастической природы их элементов. Обучение ограниченной машины Больцмана также осложняется трудностью обработки статистической суммы. В то же время оценку статистической суммы можно получить с помощью *семплирования по значимости с использованием метода обжига* (annealed importance sampling) [352]. Одним из вариантов машины Больцмана является *машина Больцмана на основе аппроксимации среднего поля* (mean-field Boltzmann machine) [373], в которой используются вещественные детерминированные элементы, а не стохастические. Однако это эвристический подход, который трудно обосновать. Тем не менее аппроксимации на основе вещественных значений часто используются на стадии вывода. Иными словами, традиционные нейронные сети с вещественными активациями и весами, взятыми из обученной машины Больцмана, широко применяют для предсказаний. Другие вариации RBM, такие как авторегрессионные модели [265], можно рассматривать как автокодировщики.

Эффективный мини-пакетный алгоритм для ограниченных машин Больцмана описан в [491]. Алгоритм контрастивной дивергенции (CD), пригодный для RBM, описан в [61, 191]. Его вариант, известный как *персистентная контрастивная дивергенция*, предложен в [491]. Идея постепенного увеличения k в методе CD_k по мере прохождения тренировки была предложена в [61]. В этой работе показано, что даже одна итерация семплирования по Гиббсу (которое значительно уменьшает длительность приработки) приводит лишь к незначительному смещению конечного результата, которое может быть дополнительно уменьшено постепенным увеличением значения k в методе CD_k по мере прохождения тренировки. Это наблюдение послужило ключом к эффективной реализации RBM. Анализ смещения в алгоритме контрастивной дивергенции приведен, в частности, в [29]. Свойства сходимости RBM анализируются в [479]. В ней также показано, что алгоритм контрастивной дивергенции

является эвристическим и в действительности не оптимизирует никакую целевую функцию. Обсуждение и практические рекомендации по обучению машин Больцмана можно найти в [119, 193]. Свойства RBM как универсальных аппроксиматоров обсуждаются в [341].

RBM находят целый ряд применений, таких как снижение размерности, коллаборативная фильтрация, тематическое моделирование и классификация. Использование RBM для коллаборативной фильтрации обсуждается в [414]. Этот подход поучителен, поскольку демонстрирует возможности использования RBM применительно к категориальным данным, содержащим небольшое количество значений. Применение дискриминантных ограниченных машин Больцмана в целях классификации обсуждается в [263, 264]. Тематическое моделирование документов посредством машин Больцмана с Softmax-элементами (что обсуждалось в данной главе) основывается на [469]. Усовершенствованные RBM для тематического моделирования с использованием распределения Пуассона обсуждаются в [134, 538]. Основная проблема этих методов заключается в их неспособности работать с документами переменной длины. Обсуждению использования реплицируемых Softmax-элементов посвящена [199]. Этот подход тесно связан с идеями *семантического хеширования* [415].

Большинство RBM предложены для бинарных данных. Однако в последние годы RBM были обобщены на другие типы данных. Моделирование счетных данных с использованием Softmax-элементов обсуждалось в контексте тематического моделирования в [469]. Трудности, связанные с моделированием этого типа, приведены в [86]. Использование RBM для семейства экспоненциальных распределений описано в [522], а обсуждение вещественных данных можно найти в [348]. Введение биномиальных элементов для кодирования большого количества информации, чем это позволяют бинарные элементы, предложено в [495]. Было показано, что такой подход представляет собой зашумленную версию ReLU [348]. Замена бинарных элементов линейными, которые содержат меньше гауссовского шума, была впервые предложена в [124]. Моделирование документов с помощью глубоких машин Больцмана обсуждается в [469]. Машины Больцмана также использовались для мультимодального обучения с использованием изображений и текста [357, 468].

Тренировка глубоких версий машин Больцмана позволила получить первый алгоритм глубокого обучения, продемонстрировавший неплохие результаты [196]. Алгоритмы этого типа были первыми методами предварительного обучения, которые впоследствии были обобщены на другие типы нейронных сетей. Методы предварительного обучения подробно описаны в разделе 4.7. Глубокие машины Больцмана обсуждаются в [417], а соответствующие эффективные алгоритмы — в [200, 418].

Некоторые архитектуры, родственные машинам Больцмана, предоставляют возможности моделирования другого типа. Машина Гельмгольца и двухфазный алгоритм *wake-sleep* (“пробуждение — сон”) предложены в [195]. Можно показать, что RBM и их многослойные варианты эквивалентны различным типам вероятностных графических моделей, таких как сигмоидные сети доверия [350]. Подробное обсуждение сигмоидных графических моделей содержится в [251]. В машинах Больцмана более высокого порядка функция энергии определяется группами из k узлов, где $k > 2$. Например, машина Больцмана 3-го порядка будет содержать члены вида $w_{ijk} s_i s_j s_k$. Машины этого типа обсуждаются в [437]. Несмотря на то что указанные методы предлагают больше возможностей, чем традиционные машины Больцмана, они не пользуются особой популярностью ввиду больших объемов данных, которые требуются для их тренировки.

6.10. Упражнения

1. В этой главе обсуждалось использование машин Больцмана для коллаборативной фильтрации. Даже несмотря на то, что для обучения модели используется дискретное семплирование алгоритма контрастивной дивергенции, окончательная фаза вывода выполняется с использованием вещественных сигмоид и Softmax-активаций. Опишите, как бы вы воспользовались этим фактом для тонкой настройки обученной модели с помощью алгоритма обратного распространения ошибки.
2. Реализуйте алгоритм контрастивной дивергенции ограниченной машины Больцмана. Также реализуйте алгоритм вывода, позволяющий получить распределение вероятности скрытых элементов для заданного тестового примера. Используйте Python или любой другой язык программирования по своему выбору.
3. Рассмотрим машину Больцмана без двудольного ограничения (RBM), но с тем ограничением, что все элементы являются видимыми. Как это ограничение упрощает процесс тренировки машины Больцмана?
4. Предложите подход, позволяющий использовать RBM для обнаружения выбросов.
5. Получите обновления весов для тематического моделирования с использованием подхода на основе RBM, который обсуждался в этой главе. Используйте те же обозначения.
6. Предложите способ расширения RBM для задач коллаборативной фильтрации (см. раздел 6.5.2) с помощью дополнительных слоев.

7. В разделе 6.5.3 описана машина Больцмана для классификации. Однако этот подход ограничен бинарной классификацией. Предложите способ его расширения на задачи многоклассовой классификации.
8. Предложите способ расширения RBM для тематического моделирования (эта RBM обсуждалась в данной главе), обеспечивающий создание скрытого представления каждого узла, извлеченного из большого разреженного графа (наподобие социальной сети).
9. Покажите, каким образом можно улучшить модель из упражнения 8 для включения данных о неупорядоченном списке ключевых слов, связанных с каждым узлом.
10. Предложите способ улучшения RBM для тематического моделирования (эта RBM обсуждалась в данной главе) за счет использования нескольких слоев.

Глава 7

Рекуррентные нейронные сети

Демократия — это периодическое проявление веры в то, что в большинстве случаев правота на стороне большинства.

“Нью-Йоркер”, 3 июля 1944 г.

7.1. Введение

Все нейронные архитектуры, обсуждавшиеся в предыдущих главах, предназначены для работы с многомерными данными, в которых атрибуты в значительной мере зависят друг от друга. Однако некоторые типы данных, такие как временные ряды, текст и биологические данные, содержат последовательные зависимости между атрибутами. Соответствующие примеры приведены ниже.

1. В наборах данных временных рядов значения, соответствующие последовательным моментам времени, тесно связаны между собой. Если использовать эти значения просто как независимые признаки, то ключевая информация об отношениях между привязанными ко времени значениями теряется. Например, значение временного ряда в момент времени t тесно связано с его значениями в предыдущем окне. Однако в случае независимой обработки значений, относящихся к отдельным моментам времени, эта информация теряется.
2. Несмотря на то что текст часто обрабатывается как “мешок слов”, учет порядка следования слов позволяет извлечь из текста более богатую семантическую информацию. В подобных случаях важно конструировать модели, учитывающие информацию о том, в каком порядке стоят слова. Текстовые данные — наиболее распространенный случай использования рекуррентных нейронных сетей.
3. Биологические данные часто содержат последовательности, в которых символы могут соответствовать аминокислотам или одному из азотистых оснований, являющихся основными строительными компонентами ДНК.

Индивидуальные значения в последовательности могут быть либо вещественными, либо символическими. Последовательности с вещественными значениями также называют временными рядами. Рекуррентные сети могут использоваться для любого типа данных. На практике более распространены символические данные. Поэтому мы сфокусируемся преимущественно на символических данных вообще и текстовых в частности. На протяжении главы мы будем по умолчанию предполагать, что на вход рекуррентной сети подается текстовый фрагмент, в котором соответствующие символы последовательности являются идентификаторами слов, входящих в словарь. Однако наряду с этим мы исследуем и другие ситуации, например случаи, в которых индивидуальными элементами являются символы или вещественные значения.

Зачастую последовательные данные, такие как текст, обрабатываются как “мешки слов”. Подобный подход игнорирует порядок следования слов в документе и хорошо подходит для документов, имеющих разумные размеры. Однако в тех случаях, когда важна семантическая интерпретация предложений или когда размер текста относительно мал (например, одно предложение), такой подход не годится. Чтобы понять, почему это так, рассмотрим следующие два предложения:

The cat chased the mouse (Кот преследовал мышь).

The mouse chased the cat (Мышь преследовала кота).

Абсолютно очевидно, что это совершенно разные предложения (причем смысл второго довольно необычен). Однако в представлении “мешка слов” они будут считаться идентичными. Следовательно, несмотря на то что представления такого типа хорошо работают в простых задачах (таких, как классификация), в более сложных случаях, таких как *сентимент-анализ*, *машинный перевод* или *извлечение информации*, требуется учет лингвистических аспектов.

Одно из возможных решений состоит в том, чтобы избежать подхода на основе “мешка слов” и создать по одному входу для каждой позиции в предложении. Рассмотрим ситуацию, когда делается попытка использовать обычную нейронную сеть для выполнения сентимент-анализа предложений с одним входом для каждой позиции в предложении. Позитивную или негативную тональность текста можно определить с помощью бинарной метки. Первой проблемой, с которой придется столкнуться на этом пути, является различная длина предложений. Поэтому, если бы мы использовали нейронную сеть с пятью наборами входных слов в представлении прямого кодирования (рис. 7.1, а), то было бы невозможно ввести предложение с более чем пятью словами. Кроме того, любое предложение, содержащее менее пяти слов, будет содержать пропущенные входы (рис. 7.1, б). В некоторых случаях, таких как предложения из веб-журналов, длина входной последовательности может достигать сотен или

тысяч единиц. Что немаловажно, небольшие изменения в порядке следования слов могут приводить к семантически различным коннотациям, и *очень важно каким-то образом закодировать информацию о порядке следования слов более непосредственно в самой архитектуре сети*. Целью такого подхода должно было бы стать снижение требований к количеству параметров при увеличении длины последовательности. Рекуррентные нейронные сети служат отличным примером *экономичной* (в смысле использования параметров) *архитектуры сети* за счет использования знаний, специфических для конкретной предметной области. Таким образом, двумя основными недостающими элементами процесса обработки последовательностей являются: 1) возможность получать и обрабатывать входы в том же порядке, в каком они встречаются в последовательности, 2) обработка входов, соответствующих последовательным моментам времени, одинаковым по отношению к предыстории входов способом. Основная трудность заключается в том, чтобы суметь сконструировать нейронную сеть с фиксированным количеством параметров и при этом сохранить возможность обрабатывать переменное количество входов.

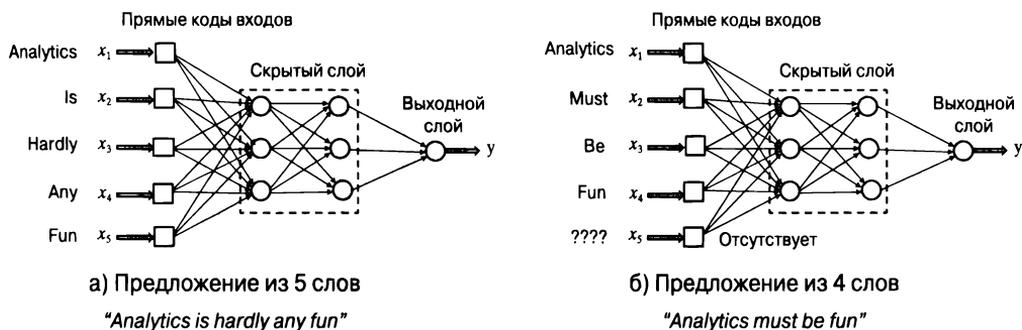


Рис. 7.1. Попытки использования обычной нейронной сети для сентимент-анализа наталкиваются на трудности обработки входов переменной длины. Архитектура сети также не содержит никакой полезной информации о зависимостях, обусловленных порядком следования слов

Отсутствие эти недостающих элементов естественным образом восполняется использованием *рекуррентных нейронных сетей* (recurrent neural network — RNN). В рекуррентной нейронной сети между ее слоями и конкретными позициями в последовательности существуют отношения "один к одному". Позицию в последовательности также называют *временной меткой* или *временным шагом*. В связи с этим вместо переменного количества входов в одном входном слое сеть содержит переменное количество слоев, и каждый слой имеет единственный вход, соответствующий данной временной метке. Поэтому входам разрешено непосредственно взаимодействовать с последующими скрытыми слоями, в зависимости от их позиций в последовательности. Чтобы

гарантировать следование одному и тому же способу моделирования в каждый момент времени, определяемый временной меткой, каждый слой задействует один и тот же набор параметров, что означает использование фиксированного количества параметров. Иными словами, с течением времени повторяется одна и та же архитектура слоев, потому сеть и называется *рекуррентной*. Рекуррентные нейронные сети также называют сетями прямого распространения со специфической структурой, основанной на временных слоях, что позволяет принимать *последовательность* входов и формировать последовательность выходов. Каждый временной слой может получать точку входных данных (в виде одиночного или множественного атрибута) и формировать, если это необходимо, многомерный выход. Подобные модели особенно полезны в таких задачах, требующих обучения одной последовательности на основе другой, как машинный перевод или предсказание следующего элемента последовательности. В качестве примера рассмотрим следующие задачи.

1. Входом может служить последовательность слов, а выходом — та же последовательность, смещенная на 1, так что мы предсказываем следующее слово в любой заданной точке. Это классическая *языковая модель*, в которой мы пытаемся предсказать следующее слово на основании предыстории слов в последовательности. Языковые модели имеют широкий спектр применений в области *интеллектуального анализа текста* (text mining) и извлечения информации [6].
2. В случае временных рядов с вещественными значениями задача обучения следующему элементу эквивалентна *авторегрессионному анализу*. Однако рекуррентная нейронная сеть может обучать гораздо более сложные модели, чем те, которые удается получить с помощью традиционных методов моделирования временных рядов.
3. Входом может служить предложение на одном языке, а выходом — то же предложение на другом языке. В этом случае можно объединить две рекуррентные сети для обучения моделей перевода с одного языка на другой. Возможно даже объединение рекуррентной сети с сетью другого типа (например, обычной нейронной сетью) для обучения захвата изображений.
4. Входом может служить последовательность (например, предложение), а выходом — вектор вероятностей классов, который запускается в конце предложения. Такой подход может применяться в задачах классификации предложений, таких как сентимент-анализ.

Из этих примеров хорошо видно, насколько велико разнообразие базовых архитектур, которые использовались или исследовались в рамках более широкой инфраструктуры RNN.

Обучение параметров рекуррентной нейронной сети значительно затруднено. В этом контексте одной из ключевых является проблема затухающих и взрывных градиентов. Она становится особенно заметной в контексте глубоких сетей наподобие RNN. Как следствие, был предложен целый ряд вариантов RNN, таких как долгая краткосрочная память (long short-term memory — LSTM) и вентильный рекуррентный блок (gated recurrent unit — GRU). Рекуррентные нейронные сети и их варианты использовались в контексте различных задач, таких как обучение “последовательность в последовательность”, захват изображений, машинный перевод и сентимент-анализ. Многочисленные примеры этих способов применения рекуррентных сетей также будут рассмотрены в данной главе.

7.1.1. Выразительная способность рекуррентных нейронных сетей

Известно, что рекуррентные нейронные сети обладают свойством *полноты по Тьюрингу* (Turing complete) [444]. Полнота по Тьюрингу означает, что при наличии достаточного количества данных и вычислительных ресурсов рекуррентная нейронная сеть может имитировать любой алгоритм [444]. Однако на практике это свойство не является особенно полезным, поскольку объемы данных и вычислительных ресурсов, требуемые для достижения этой цели, могут оказаться нереалистичными. Кроме того, в процессе тренировки рекуррентных нейронных сетей возникают трудности практического характера, такие как проблема затухающих и взрывных градиентов. С увеличением длины последовательности эти проблемы усугубляются, а более стабильные варианты сетей, такие как долгая краткосрочная память, обеспечивают лишь частичное разрешение указанных проблем. Нейронная машина Тьюринга, в которой стабильность процесса обучения нейронной сети достигается за счет использования внешней памяти, обсуждается в главе 10. Можно показать, что нейронная машина Тьюринга эквивалентна рекуррентной нейронной сети, и в качестве важного компонента, ответственного за принятие решений относительно предпринимаемых действий, в ней часто используется более традиционная рекуррентная сеть. Более подробно об этом речь пойдет в разделе 10.3.

Структура главы

В следующем разделе будет представлена базовая архитектура RNN вместе с соответствующим алгоритмом обучения. Трудности обучения рекуррентных нейронных сетей обсуждаются в разделе 7.3. Ввиду существования таких трудностей были предложены различные варианты архитектуры RNN. Некоторые из них также будут рассмотрены в данной главе. Эхо-сети представлены в разделе 7.4. Сети на основе долгой краткосрочной памяти обсуждаются

в разделе 7.5. Вентильный рекуррентный блок рассматривается в разделе 7.6. Рассмотрению применений рекуррентных нейронных сетей посвящен раздел 7.7. Резюме главы приведено в разделе 7.8.

7.2. Архитектура рекуррентных нейронных сетей

Несмотря на то что рекуррентные нейронные сети можно использовать практически в любой области, требующей обработки последовательных данных, их применение для обработки текста не только широко распространено, но и вполне естественно. Именно обработка текста будет рассмотрена в этом разделе, поскольку она позволит на интуитивно понятном уровне объяснить целый ряд важных понятий. Данная глава будет сфокусирована в основном на дискретных RNN как наиболее популярном варианте рекуррентных сетей. Отметим, что точно такую же нейронную сеть можно использовать для построения RNN, работающих как на уровне слов, так и на уровне символов, т.е. букв. Единственное различие между этими двумя случаями — набор базовых символов, используемых для определения последовательности. Для большей ясности изложения мы введем необходимые понятия и определения на примере RNN, работающих на уровне слов. Однако в этой главе не будут обойдены вниманием и другие варианты применения RNN.

Простейшая рекуррентная нейронная сеть представлена на рис. 7.2, а. Здесь ключевую роль играет наличие петли, что вызывает изменение скрытого состояния нейронной сети после появления на входе каждого слова последовательности. На практике работают только с последовательностями конечной длины, так что имеет смысл развернуть цикл в последовательность временных слоев, соответствующих различным меткам времени, что больше похоже на сеть прямого распространения (рис. 7.2, б). В данном случае у нас имеются отдельные узлы для скрытых состояний в каждый момент времени, и петля развертывается в сеть прямого распространения. Это представление математически эквивалентно тому, которое приведено на рис. 7.2, а, но оно гораздо более понятно в силу его сходства с традиционной сетью. Матрицы весов *разделяются* различными временными слоями, гарантируя тем самым, что для каждой метки времени используется одна и та же функция. Обозначения W_{xh} , W_{hh} и W_{hy} матриц весов на рис. 7.2, б, делают это разделение очевидным.

Следует отметить, что на рис. 7.2 представлен случай, в котором для каждой временной метки имеется свой вход, выход и скрытый элемент. На практике в любой момент времени входной или выходной элемент может отсутствовать. Примеры отсутствующих входов и выходов приведены на рис. 7.3. Выбор отсутствующих входов или выходов определяется спецификой конкретного приложения. Например, в задачах прогнозирования нам могут понадобиться

выходы для каждой временной отметки, чтобы можно было предсказать следующее значение во временном ряду. С другой стороны, в задачах классификации последовательностей может требоваться всего лишь одна выходная метка в конце последовательности, соответствующая ее классу. Вообще говоря, в зависимости от конкретики приложения может отсутствовать любое подмножество входов или выходов. В приведенном ниже обсуждении мы будем предполагать наличие всех входов и выходов, хотя эти же рассуждения можно легко обобщить на случаи, когда некоторые из входов или выходов отсутствуют, просто исключив соответствующие члены или уравнения.

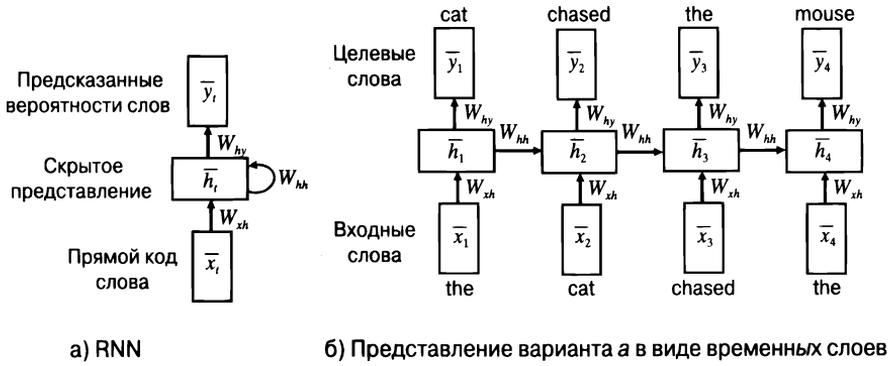


Рис. 7.2. Рекуррентная нейронная сеть и ее развертка во времени

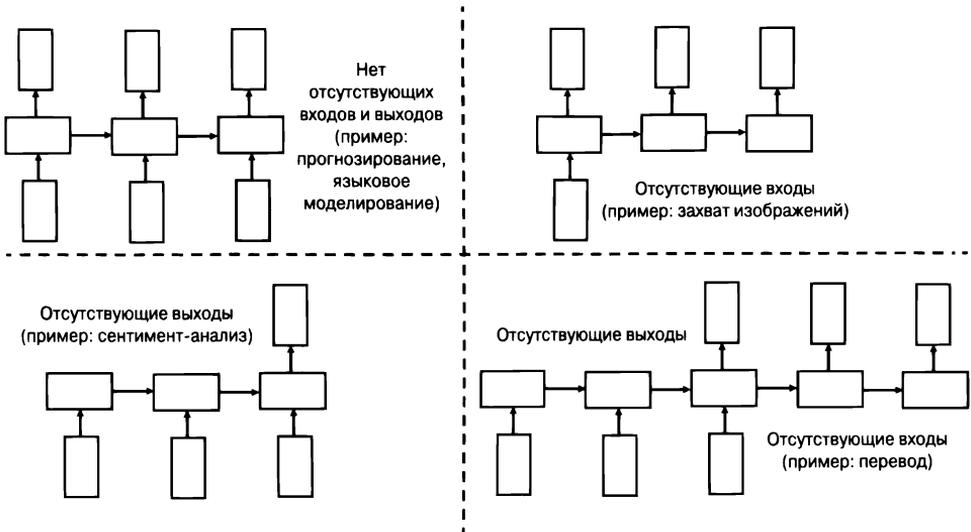


Рис. 7.3. Различные вариации рекуррентных сетей с отсутствующими входами и выходами

Архитектура, представленная на рис. 7.2, приспособлена для языкового моделирования. Понятие языкового моделирования хорошо известно специалистам в области обработки естественного языка, и его целью является предсказание следующего слова, если известна последовательность предыдущих слов. При заданной последовательности слов их прямые коды одновременно подаются на вход нейронной сети (рис. 7.2, а). Этот быстротекущий процесс эквивалентен подаче отдельных слов на входы в моменты времени, соответствующие временным меткам (рис. 7.2, б). Временные метки соответствуют позициям в последовательности, и их нумерация начинается с 0 (или 1) и увеличивается на 1 при переходе вперед на один элемент последовательности. В условиях языкового моделирования выходом является вектор вероятностей, предсказанных для следующего слова последовательности. В качестве примера рассмотрим следующую последовательность:

The cat chased the mouse.

Когда входом является слово “The”, выходом будет вектор вероятностей для всех слов словаря, включающий слово “cat”, а когда входом является слово “cat”, мы вновь получим вектор вероятностей, предсказывающий следующее слово. Разумеется, это классическое определение языкового моделирования, в котором вероятность слова оценивается на основе непосредственной предыстории следования предыдущих слов. В общем случае обозначим входной вектор в момент времени t (например, входной вектор t -го слова в представлении прямого кодирования) как \bar{x}_t , скрытое состояние в момент времени t — как \bar{h}_t , а выходной вектор в момент времени t , например предсказанные вероятности $(t + 1)$ -го слова, — как \bar{y}_t . В случае словаря размером d как \bar{x}_t , так и \bar{y}_t являются d -мерными векторами. Скрытый вектор \bar{h}_t — p -мерный, где p регулирует сложность вложенного представления. Для целей предстоящего обсуждения будем полагать, что все названные векторы являются вектор-столбцами. Во многих задачах, таких как классификация, выход не генерируется элементами для каждой отметки времени, а лишь запускается на последней отметке времени в конце последовательности. Несмотря на то что входные и выходные элементы могут соответствовать лишь подмножеству отметок времени, мы исследуем простейший случай, когда они имеются для всех отметок времени. Тогда скрытое состояние в момент времени t задается функцией входного вектора в момент времени t и скрытого вектора в момент времени $(t - 1)$:

$$\bar{h}_t = f(\bar{h}_{t-1}, \bar{x}_t). \quad (7.1)$$

Эта функция определяется с помощью матриц весов и функций активации (так как они используются для обучения всеми сетями), причем для каждой временной отметки используются одни и те же веса. Поэтому, несмотря на то

что скрытое состояние эволюционирует во времени, веса и базовая функция $f(\cdot, \cdot)$ остаются фиксированными для всех временных отметок (т.е. элементов последовательности) по завершении обучения нейронной сети. Для обучения выходных вероятностей на основе скрытых состояний применяется отдельная функция $\bar{y}_t = g(\bar{h}_t)$.

Конкретизируем функции $f(\cdot, \cdot)$ и $g(\cdot)$. Определим матрицу весов “вход — скрытый” W_{xh} размера $p \times d$, матрицу весов “скрытый — скрытый” W_{hh} размера $p \times p$ и матрицу весов “скрытый — выход” W_{hy} размера $d \times p$. Тогда уравнение 7.1 и условие для выходов запишутся в следующем виде:

$$\begin{aligned}\bar{h}_t &= \tanh(W_{xh}\bar{x}_t + W_{hh}\bar{h}_{t-1}), \\ \bar{y}_t &= W_{hy}\bar{h}_t.\end{aligned}$$

Здесь подразумевается, что функция активации \tanh применяется к p -мерному вектор-столбцу поэлементно, создавая новый p -мерный вектор-столбец, в котором значение каждого элемента принадлежит интервалу $[-1, 1]$. Именно такое поэлементное воздействие функций активации на векторы будет подразумеваться на протяжении всего раздела. Для самой первой отметки времени предполагается, что \bar{h}_{t-1} — некоторый заданный по умолчанию постоянный вектор (например, нулевой), поскольку в начале предложения от скрытого слоя не поступает никаких входных данных. Также возможно обучение этого вектора. Несмотря на то что скрытые состояния изменяются при переходе от одной отметки времени к другой, матрицы весов остаются постоянными для различных отметок. Заметьте, что выходной вектор \bar{y}_t — это набор непрерывных значений, размер которого совпадает с размером словаря. Поверх \bar{y}_t налагается слой Softmax, чтобы обеспечить возможность вероятностной интерпретации результатов. *p -мерный выход \bar{h}_t скрытого слоя в конце текстового сегмента из t слов создает его вложение, а p -мерные столбцы W_{xh} создают вложения отдельных слов.* Последние предоставляют альтернативу вложениям *word2vec* (см. главу 2).

Ввиду рекурсивной природы уравнения 7.1 *рекуррентная сеть способна вычислять функцию входов переменной длины.* Иными словами, рекурсию уравнения 7.1 можно разложить, чтобы определить функцию для \bar{h}_t в терминах t входов. Например, начиная с вектора \bar{h}_0 , в качестве которого обычно выбирают некий постоянный вектор (например, нулевой), мы получаем $\bar{h}_1 = f(\bar{h}_0, \bar{x}_1)$ и $\bar{h}_2 = f(f(\bar{h}_0, \bar{x}_1), \bar{x}_2)$. Обратите внимание на то, что \bar{h}_1 — функция, зависящая только от \bar{x}_1 , тогда как \bar{h}_2 зависит как от \bar{x}_1 , так и от \bar{x}_2 . В общем случае \bar{h}_t является функцией $\bar{x}_1 \dots \bar{x}_t$. Поскольку выход \bar{y}_t является функцией \bar{h}_t , он также наследует эти свойства. Обобщая, можно записать следующее соотношение:

$$\bar{y}_t = F_t(\bar{x}_1, \bar{x}_2, \dots, \bar{x}_t). \quad (7.2)$$

Отметим, что функция $F_t(\cdot)$ меняется с изменением t , хотя ее соотношение с непосредственно предшествующим состоянием всегда остается одним и тем же (на основании уравнения 7.1). Такой подход особенно полезен в случае входов переменной длины. Подобные ситуации часто встречаются во многих задачах, например при обработке текста, в котором длина предложений меняется. Например, в случае языкового моделирования функция $F_t(\cdot)$ определяет вероятность следующего слова с учетом предыдущих слов предложения.

7.2.1. Пример языкового моделирования с помощью RNN

Чтобы проиллюстрировать работу RNN, мы используем небольшой пример, определенный на словаре из четырех слов:

The cat chased the mouse.

В данном случае наш словарь состоит из четырех слов: {"the", "cat", "chased", "mouse"}. На рис. 7.4 отображено вероятностное предсказание следующего слова для каждой временной отметки от 1 до 4. В идеале нам хотелось бы, чтобы вероятность следующего слова правильно предсказывалась на основе вероятности предыдущих слов. Длина каждого вектора \bar{x}_t в представлении прямого кодирования равна четырем, причем только один бит равен 1, тогда как остальные равны нулю. В данном случае основным фактором гибкости является размерность p скрытого представления, которую мы установили равной 2. В результате матрица W_{xh} будет иметь размер 2×4 , поэтому она транслирует входной вектор в представлении прямого кодирования на скрытый вектор \bar{h}_t размера 2. В практическом отношении каждый столбец W_{xh} соответствует одному из четырех слов, и один из этих столбцов копируется выражением $W_{xh}\bar{x}_t$. Обратите внимание на то, что это выражение прибавляется к выражению $W_{hh}\bar{h}_t$, а затем преобразуется с помощью функции гиперболического тангенса для получения окончательного выражения. Конечный вывод \bar{y}_t определяется выражением $W_{hy}\bar{h}_t$. Матрицы W_{hh} и W_{hy} имеют размеры 2×2 и 4×2 соответственно.

В данном случае выходами являются непрерывные значения (не вероятности), причем большие значения указывают на большую вероятность того, что данное слово будет следующим. В конечном счете непрерывные значения преобразуются в вероятности с помощью функции *Softmax*, поэтому их можно рассматривать в качестве замены логарифмических вероятностей. В первый момент времени слово "cat" предсказывается со значением 1,3, но, как нетрудно заметить, его (некорректно) "опережает" слово "mouse", для которого соответствующее значение равно 1.7. Однако для следующей отметки времени слово "chased", как видим, предсказано корректно. Как и в случае любого алгоритма

обучения, нельзя надеяться на то, что все значения будут предсказаны точно, и подобные ошибки наиболее вероятны на ранних итерациях алгоритма обратного распространения. Но поскольку сеть периодически тренируется на протяжении многих итераций, алгоритм со временем допускает все меньше ошибок на тренировочных данных.

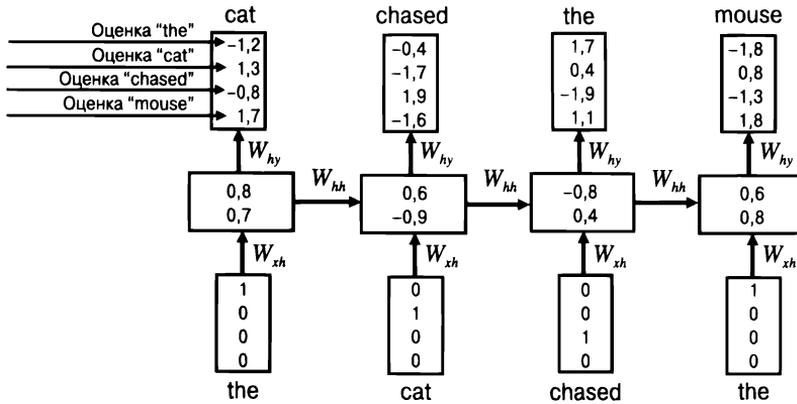


Рис. 7.4. Пример языкового моделирования с помощью рекуррентной нейронной сети

7.2.1.1. Генерирование образцов текста

Как только тренировка завершена, подобный подход можно использовать также для генерирования произвольных образцов текста. Каким же образом можно использовать языковую модель во время тестирования, если для каждого состояния требуется входное слово, тогда как в начале процесса генерирования текста такие слова отсутствуют? Вероятности лексем, соответствующих первой отметке времени, можно сгенерировать, используя токен <START> в качестве входа. Поскольку токен <START> имеется также среди тренировочных данных, модель обычно выбирает слово, с которого часто начинаются текстовые сегменты. Следовательно, суть идеи заключается в том, чтобы выбрать одну из лексем, сгенерированных (на основании предсказанной вероятности) для каждой отметки времени, и использовать ее в качестве входа для следующей отметки. Точность последовательного предсказания лексем можно улучшить, если использовать лучевой поиск (beam search) для выбора наиболее подходящих вероятностей, постоянно отслеживая b наилучших префиксов последовательности любой конкретной длины. Значение b является параметром, определяемым пользователем. Рекурсивно применяя эту операцию, можно сгенерировать произвольную текстовую последовательность, которая отражает свойства имеющегося тренировочного набора. Если предсказывается токен <END>, то это указывает на конец конкретного текстового сегмента. Несмотря на то что такой

подход часто приводит к синтаксически корректному тексту, этот текст может не иметь никакого смысла. Например, предложенная Карпаты, Джонсоном и Фей-Фей [233, 580] сеть RNN¹, работающая на уровне символов, тренировалась на пьесах Шекспира. RNN символьного уровня требует, чтобы нейронная сеть обучалась как синтаксису, так и орфографии. После пяти итераций обучения на полном наборе данных был получен следующий образец вывода:

KING RICHARD II:

Do cantant, -'for neight here be with hand her,-
Eptar the home that Valy is thee.

NORONCES:

Most ma-wrow, let himself my hispeasures;
An exmorbackion, gault, do we to do you comforr,
Laughter's leave: mire sucintracce shall have theref-Helt.

В этом тексте многие слова являются бессмысленными, а текст в целом — это сущая абракадабра. Однако после того, как тренировка была продолжена до 50 итераций, был получен следующий образец:

KING RICHARD II:

Though they good extremity if you damed;
Made it all their fripts and look of love;
Prince of forces to uncertained in conserve
To thou his power kindless. A brives my knees
In penitence and till away with re doom.

GLOUCESTER:

Between I must abide.

В сгенерированном отрывке текста в значительной степени соблюдены синтаксис и орфография архаичного английского языка шекспировских пьес, хотя в нем по-прежнему имеются очевидные ошибки. Кроме того, этот подход также обеспечивает выделение отступами и форматирование текста в манере, присущей текстам пьес, располагая соответствующим образом новые строки. Увеличение количества итераций тренировки делает вывод практически безошибочным, а некоторые впечатляющие примеры приведены в [235].

Разумеется, семантический смысл текста весьма ограничен, и у вас могут возникнуть сомнения относительно пользы от генерирования столь бессмысленных отрывков текста с точки зрения перспективы для задач машинного обучения.

¹ В данном случае использовалась сеть на основе долгой краткосрочной памяти (LSTM), являющаяся вариацией обычной сети RNN, которую мы обсуждали.

В этом отношении важно иное: путем передачи дополнительного *контекстуального* ввода, такого как нейронное представление изображений, можно добиться того, что сеть будет выдавать разумный вывод, например грамматически корректное описание изображения (снабжение его подписью). Другими словами, языковые модели лучше всего использовать, генерируя *условные* выходы.

Основной задачей RNN языкового моделирования является не создание произвольных текстовых последовательностей, а формирование архитектурной базы, которую можно модифицировать различными способами для встроеного учета влияния конкретного контекста. Например, в таких задачах, как машинный перевод или аннотирование изображений, языковая модель обучается *в привязке* к другому входу — предложению на исходном языке перевода или аннотируемому изображению соответственно. Поэтому конкретная структура RNN, зависящая от приложения, будет использовать те же общие принципы, что и RNN, предназначенная для языкового моделирования, но включать небольшие изменения этой базовой архитектуры, учитывающие влияние специфического контекста. Во всех подобных случаях важную роль играет выбор входных и выходных значений рекуррентных элементов разумным способом, обеспечивающим возможность применения алгоритма обратного распространения ошибки и обучения весов связей нейронной сети с учетом специфики приложения.

7.2.2. Обратное распространение ошибки во времени

Отрицательные логарифмы Softmax-вероятностей корректных слов в различные моменты времени агрегируются для создания функции потерь. Функция *Softmax* была описана в разделе 3.2.5.1, так что мы просто используем здесь соответствующие результаты. Если выходной вектор записан в виде $[\hat{y}_t^1 \dots \hat{y}_t^d]$, то сначала он преобразуется в вектор d вероятностей с помощью функции *Softmax*:

$$[\hat{p}_t^1 \dots \hat{p}_t^d] = \text{Softmax}([\hat{y}_t^1 \dots \hat{y}_t^d]).$$

Выражение для функции *Softmax* приведено в уравнении 3.20. Если j_t — истинное слово в тренировочных данных, соответствующее моменту времени t , то функция потерь L для всех T временных отметок вычисляется по следующей формуле:

$$L = -\sum_{t=1}^T \log(\hat{p}_t^{j_t}). \quad (7.3)$$

Эта функция потерь является непосредственным следствием уравнения 3.21. Производная функции потерь по необработанным выходам вычисляется в соответствии со следующей формулой (см. уравнение 3.22):

$$\frac{\partial L}{\partial \hat{y}_t^k} = \hat{p}_t^k - I(k, j_t), \quad (7.4)$$

где $I(k, j_i)$ — индикаторная функция, принимающая значение 1, если значения k и j_i совпадают, и 0 в противном случае. Начав с этой частной производной, можно непосредственно применить алгоритм обратного распространения ошибки, описанный в главе 3 (на примере неразвернутой временной сети), для вычисления градиентов по весам в различных слоях с целью их обновления. Главной проблемой является то, что разделение весов различными временными слоями будет влиять на процесс обновления. С точки зрения корректного применения цепного правила для обратного распространения ошибки (см. главу 3) важную роль играет допущение о различии весов в разных слоях, что обеспечивает относительную простоту обновления параметров. Однако, как обсуждалось в разделе 3.2.9, алгоритм обратного распространения ошибки можно легко приспособить для обработки разделяемых весов.

Здесь применяется трюк, в соответствии с которым на первом этапе полагаются, что параметры в различных временных слоях являются независимыми. С этой целью для временной отметки t вводятся временные переменные $W_{xh}^{(t)}$, $W_{hh}^{(t)}$ и $W_{hy}^{(t)}$. Сначала выполняется обычный алгоритм обратного распространения ошибки, основанный на независимости этих переменных. Затем вклады различных временных аватаров параметров весов суммируются с целью создания унифицированного обновления для каждого весового параметра. Этот специальный тип алгоритма называют *обратным распространением ошибки во времени* (backpropagation through time — ВРТТ).

Ниже суммированы отдельные шаги алгоритма ВРТТ.

1. Выполняем последовательный ввод в прямом направлении во времени и вычисляем ошибки (а также отрицательный логарифм функции потерь слоя Softmax) для каждой отметки времени.
2. Вычисляем градиенты весов связей в обратном направлении для развернутой сети, не обращая внимания на тот факт, что веса разделяются различными временными слоями. Иными словами, предполагается, что веса $W_{xh}^{(t)}$, $W_{hh}^{(t)}$ и $W_{hy}^{(t)}$ в момент времени t отличаются от весов, соответствующих другим отметкам времени. Как следствие, для вычисления $\frac{\partial L}{\partial W_{xh}^{(t)}}$, $\frac{\partial L}{\partial W_{hh}^{(t)}}$ и $\frac{\partial L}{\partial W_{hy}^{(t)}}$ можно использовать обычный алгоритм обратного распространения ошибки. Обратите внимание на то, что мы использовали обозначения матричного исчисления, где производная по матрице определяется соответствующей матрицей поэлементных производных.
3. Суммируем все (разделяемые) веса, соответствующие различным экземплярам связей во времени. Иными словами, имеем следующие соотношения:

$$\frac{\partial L}{\partial W_{xh}} = \sum_{t=1}^T \frac{\partial L}{\partial W_{xh}^{(t)}},$$

$$\frac{\partial L}{\partial W_{hh}} = \sum_{t=1}^T \frac{\partial L}{\partial W_{hh}^{(t)}},$$

$$\frac{\partial L}{\partial W_{hy}} = \sum_{t=1}^T \frac{\partial L}{\partial W_{hy}^{(t)}}.$$

Приведенные выше производные являются следствием непосредственного применения многомерного цепного правила. Как и в случае любого метода обратного распространения с разделяемыми весами (см. раздел 3.2.9), мы используем тот факт, что частная производная временной копии каждого параметра (например, элемента $W_{xh}^{(t)}$) по исходному экземпляру этого параметра (например, соответствующего элемента W_{xh}) может быть установлена равной 1. Здесь уместно подчеркнуть, что вычисление частных производных по временным копиям весов ничем не отличается от традиционного обратного распространения ошибки. Поэтому для получения уравнений обновления нам остается лишь обернуть обычный алгоритм обратного распространения агрегацией по времени. Алгоритм обратного распространения во времени был первоначально предложен Вербосом в 1990 году [526] задолго до того, как рекуррентные нейронные сети приобрели популярность.

Усеченное обратное распространение ошибки во времени

Одной из проблем тренировки рекуррентных сетей является то, что базовые последовательности могут быть очень длинными, в результате чего количество слоев в сети также может быть очень большим. Это может порождать проблемы с вычислениями, сходимостью и использованием памяти. Проблемы подобного рода разрешаются за счет использования алгоритма *усеченного обратного распространения ошибки во времени*. Эту методику можно рассматривать как аналог стохастического градиентного спуска для RNN. В данном подходе значения состояний корректно вычисляются в процессе прямого распространения, но обновления обратного распространения выполняются лишь для сегментов последовательности умеренной длины (скажем, 100). Иными словами, для нахождения градиентов и обновления весов используется лишь та часть потерь, которая вычисляется для соответствующего сегмента. Сегменты обрабатываются в том порядке, в каком они встречаются во входной последовательности. Фазу прямого распространения не обязательно выполнять за один раз, она также может быть выполнена для соответствующего сегмента, при условии, что значения в последнем временном слое этого сегмента используются для вычисления значений состояний в следующем сегменте слоев. Значения в последнем

слое текущего сегмента используются для вычисления значений в первом слое следующего сегмента. Поэтому алгоритм прямого распространения всегда способен точно обрабатывать значения состояний, хотя алгоритм обратного распространения использует лишь небольшую часть функции потерь. В данном случае мы описали усеченный алгоритм ВРТТ, в котором для простоты применяются непрерывающиеся сегменты. На практике для вычисления обновлений можно задействовать перекрывающиеся сегменты входов.

Проблемы практического характера

Элементы каждой матрицы весов инициализируются небольшими значениями из интервала $[-1/\sqrt{r}, 1/\sqrt{r}]$, где r — количество столбцов в данной матрице. Также возможна инициализация каждого из d столбцов входной матрицы весов W_{xh} вложениями *word2vec* соответствующего слова (см. главу 2). Этот подход представляет собой разновидность предварительного обучения. Конкретные преимущества использования предварительного обучения этого типа зависят от объема тренировочных данных. Такой тип инициализации может быть полезным при небольшом объеме доступных тренировочных данных. В конце концов, предварительное обучение является разновидностью регуляризации (см. главу 4).

Еще одной важной деталью является то, что тренировочные данные часто содержат специальные токены (маркеры) $\langle \text{START} \rangle$ и $\langle \text{END} \rangle$ в начале и в конце тренировочного сегмента. Токены этого типа помогают модели распознавать специфические блоки текста, такие как предложения, параграфы или начало определенного модуля текста. Распределение слов в начале текстового сегмента часто очень отличается от их распределения во всем тренировочном наборе. Поэтому, как только встретится токен $\langle \text{START} \rangle$, модель, вероятнее всего, выберет слова, которыми начинается определенный сегмент текста.

Существуют и другие подходы к принятию решения о том, следует ли заканчивать сегмент в определенной точке. В качестве конкретного примера можно привести использование бинарного выхода, на основании которого и принимается решение относительно того, должна ли последовательность быть продолжена в данной точке. Следует отметить, что бинарный выход является дополнением к другим выходам, специфическим для приложения. В типичных случаях для моделирования предсказания этого выхода используется сигмоидная активация, а в качестве функции потерь — кросс-энтропия. Такой подход полезен в случае последовательностей с вещественными значениями. Это обусловлено тем, что токены $\langle \text{START} \rangle$ и $\langle \text{END} \rangle$, по сути, предназначены для использования с символическими последовательностями. В то же время одним из недостатков данного подхода является то, что его применение сопровождается переходом от функции потерь в специфической для приложения форме к функции потерь,

которая обеспечивает баланс между предсказанием конца последовательности и потребностями, специфическими для предложения. Поэтому веса различных компонент функции потерь будут еще одним гиперпараметром, с которым придется работать.

С обучением RNN связан ряд проблем практического характера, которые диктуют необходимость внесения различных усовершенствований в архитектуру RNN. Также следует подчеркнуть, что во всех практических приложениях используется множество скрытых слоев (с улучшениями на основе долгой краткосрочной памяти), о чем пойдет речь в разделе 7.2.4. Однако для большей ясности мы будем использовать более простую однослойную модель. Обобщить эти приложения на усовершенствованные архитектуры не составляет труда.

7.2.3. Двухнаправленные рекуррентные сети

Одним из недостатков рекуррентных сетей является то, что состоянию на определенном временном шаге известны предыдущие входы вплоть до конкретного слова в предложении, но будущие состояния не известны. В качестве конкретного примера можно привести распознавание рукописного текста, когда важную роль играют знания как о прошлых, так и о будущих символах, поскольку это позволяет получить лучшее представление о базовом контексте.

В двухнаправленной рекуррентной сети имеются отдельные скрытые состояния $\bar{h}_i^{(f)}$ и $\bar{h}_i^{(b)}$ для прямого и обратного направлений. Скрытые состояния, соответствующие прямому направлению (“прямые” состояния), взаимодействуют лишь между собой, и то же самое относится к состояниям, соответствующим обратному направлению (“обратные” состояния). Основное различие заключается в том, что “прямые” состояния взаимодействуют в прямом направлении, тогда как “обратные” — в обратном. Однако как $\bar{h}_i^{(f)}$, так и $\bar{h}_i^{(b)}$ получают вход от одного и того же вектора \bar{x}_i (например, слова в представлении прямого кодирования) и взаимодействуют с одним и тем же выходным вектором \bar{y}_i . Пример двухнаправленной RNN с тремя временными слоями приведен на рис. 7.5.

Существует ряд задач, в которых требуется предсказывать свойства текущих лексем. В качестве примера можно привести распознавание символов в образце рукописного текста или частеречную разметку предложений, а также классификацию лексем естественной речи. Вообще говоря, применение этого подхода обеспечивает более эффективное предсказание любого свойства *текущего* слова, поскольку он использует как левый, так и правый контекст. Например, в некоторых языках порядок следования слов может меняться в зависимости от грамматической конструкции. Поэтому двухнаправленная рекуррентная сеть часто моделирует скрытые представления любой конкретной позиции в предложении более надежно за счет использования “прямых” и “обратных” состояний, независимо от специфики языка. И действительно, использование

двунаправленных рекуррентных сетей в различных задачах, связанных с обработкой естественного языка, таких как распознавание речи, становится все более популярным.

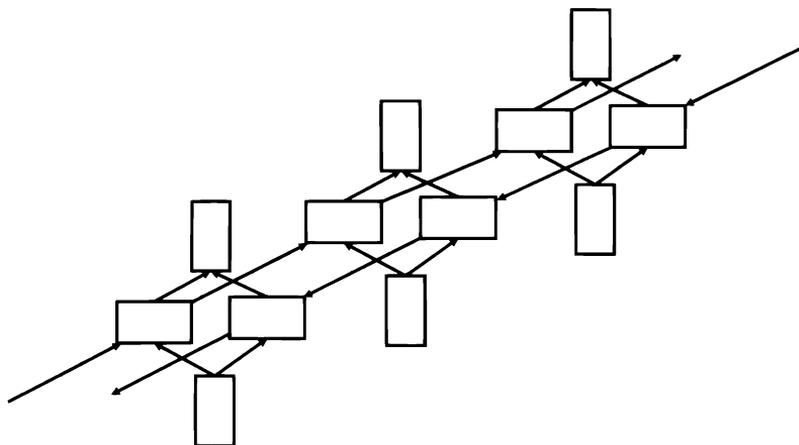


Рис. 7.5. Двунаправленная рекуррентная сеть с тремя временными слоями

В случае двунаправленной сети мы имеем отдельные матрицы параметров для прямого и обратного направлений. Обозначим матрицы взаимодействия между слоями “входной — скрытый”, “скрытый — скрытый” и “скрытый — выходной” в прямом направлении как $W_{xh}^{(f)}$, $W_{hh}^{(f)}$ и $W_{hy}^{(f)}$ соответственно. Аналогичные матрицы для обратного направления обозначим как $W_{xh}^{(b)}$, $W_{hh}^{(b)}$ и $W_{hy}^{(b)}$ соответственно.

Условия рекурсии можно записать в следующем виде:

$$\begin{aligned}\bar{h}_t^{(f)} &= \tanh\left(W_{xh}^{(f)}\bar{x}_t + W_{hh}^{(f)}\bar{h}_{t-1}^{(f)}\right), \\ \bar{h}_t^{(b)} &= \tanh\left(W_{xh}^{(b)}\bar{x}_t + W_{hh}^{(b)}\bar{h}_{t+1}^{(b)}\right), \\ \bar{y}_t &= W_{hy}^{(f)}\bar{h}_t^{(f)} + W_{hy}^{(b)}\bar{h}_t^{(b)}.\end{aligned}$$

Как нетрудно заметить, эти уравнения получены простым обобщением уравнений, используемых в случае одного направления. При этом предполагается, что общее количество временных шагов в представленной выше нейронной сети равно T , где T — длина последовательности. Под вопросом остаются только граничные условия при $t = 1$ для входа в прямом направлении и при $t = T$ для входа в обратном направлении, которые не определены. В обоих этих случаях можно по умолчанию использовать постоянное значение 0,5, однако определение таких значений может быть сделано частью процесса обучения.

Сразу же отметим, что скрытые состояния, соответствующие прямому и обратному направлениям, вообще не взаимодействуют между собой. Поэтому

сначала можно выполнить последовательность в прямом направлении и вычислить скрытые состояния, соответствующие этому направлению, а затем сделать то же самое для вычисления скрытых состояний, соответствующих обратному направлению. Теперь, когда скрытые состояния для обоих направлений найдены, можно вычислить выходные состояния.

За вычислением выходных состояний следует вычисление частных производных по различным параметрам с помощью алгоритма обратного распространения ошибки. Сначала вычисляются частные производные по выходным состояниям, поскольку состояния, соответствующие как прямому, так и обратному направлению, указывают на выходные узлы. Затем выполняется прогон обратного распространения только для скрытых состояний, соответствующих прямому направлению, начиная с $t = T$ и до $t = 1$. Далее выполняется прогон обратного распространения для скрытых состояний, соответствующих обратному направлению, начиная с $t = 1$ и до $t = T$. В завершение выполняется агрегирование частных производных по разделяемым параметрам. Таким образом, алгоритм ВРТТ легко видоизменяется для случая двунаправленных сетей. Описанные шаги можно суммировать в следующем виде.

1. Вычислить скрытые состояния, соответствующие прямому и обратному направлениям, путем выполнения для них по отдельности независимых прогонов алгоритма обратного распространения ошибки.
2. Вычислить выходные состояния на основании полученных скрытых состояний, соответствующих обоим направлениям.
3. Вычислить частные производные функции потерь по выходным состояниям и каждой копии выходных параметров.
4. Вычислить частные производные функции потерь независимо по состояниям, соответствующим прямому и обратному направлениям. Использовать полученные результаты для оценки частных производных по каждой копии параметров, соответствующих прямому и обратному направлениям.
5. Агрегировать частные производные по разделяемым параметрам.

Двунаправленные рекуррентные сети подходят для приложений, в которых предсказания не являются каузальными (т.е. не учитывают причинно-следственные связи на основании предыстории событий). Классическим примером каузальной задачи может служить поток символов, в котором событие предсказывается на основании предыдущих символов. Несмотря на то что приложения языкового моделирования формально считаются каузальными (т.е. базирующимися на предыстории следования слов), реальность такова, что заданное слово может быть предсказано с намного большей точностью посредством использования слов контекста по обе стороны от него. В общем случае двунаправленные RNN хорошо работают в тех задачах, в которых предсказания базируются

на двунаправленном контексте. Примерами таких задач могут служить распознавание рукописного текста и распознавание речи, когда свойства отдельных элементов последовательности зависят от элементов, располагающихся по обе стороны от них. Так, если рукописный текст выражен в терминах штрихов, то распознаванию определенного синтезируемого символа способствуют штрихи по обе стороны от конкретной позиции. Кроме того, вероятность того, что символы будут смежными, для одних символов больше, чем для других.

Двунаправленные рекуррентные сети позволяют добиваться результатов почти того же качества, что и использование ансамбля из двух отдельных рекуррентных сетей, в одной из которых вход представлен в исходной форме, а в другой вход обращен. Основное отличие состоит в том, что в данном случае параметры состояний для обоих направлений обучаются совместно. Однако такая интеграция довольно слабая ввиду отсутствия непосредственного взаимодействия этих двух типов состояний.

7.2.4. Многослойные рекуррентные сети

При описании всех вышеупомянутых задач для большей ясности использовалась архитектура однослойной RNN. Однако на практике применяется многослойная архитектура, позволяющая строить модели повышенной сложности. Кроме того, многослойную архитектуру можно использовать в сочетании с усовершенствованными вариантами RNN, такими как архитектура LSTM или вентильные рекуррентные элементы. Эти усовершенствованные архитектуры рассматриваются в последующих разделах.

Пример глубокой нейронной сети, содержащей три слоя, приведен на рис. 7.6. Обратите внимание на то, что узлы в вышележащих слоях получают входные данные от нижележащих. Соотношения между скрытыми слоями можно получить обобщением соотношений для однослойной сети. Прежде всего мы переписываем уравнение рекурсии скрытых слоев (для однослойных сетей) в формате, который может быть легко адаптирован для многослойных сетей:

$$\begin{aligned}\bar{h}_t &= \tanh(W_{xh}\bar{x}_t + W_{hh}\bar{h}_{t-1}) = \\ &= \tanh W \begin{bmatrix} \bar{x}_t \\ \bar{h}_{t-1} \end{bmatrix}.\end{aligned}$$

Здесь мы объединили две матрицы в одну большую матрицу $W = [W_{xh}, W_{hh}]$, которая включает столбцы W_{xh} и W_{hh} . Точно так же мы создали большой вектор-столбец, расположив в одном столбце вектор состояний в первом скрытом слое в момент времени $t - 1$ и входной вектор в момент времени t . Чтобы можно было различать скрытые узлы вышележащих слоев, добавим дополнительный верхний индекс для скрытого состояния и введем для вектора скрытых состоя-

ний на шаге t в слое k обозначение $\bar{h}_t^{(k)}$. Точно так же введем для матрицы весов k -го скрытого слоя обозначение $W^{(k)}$. Следует подчеркнуть, что веса сообщаются используются различными временными шагами (как в однослойной рекуррентной сети), но не различными слоями. Именно из этих соображений обозначение матрицы весов $W^{(k)}$ включает верхний индекс k , конкретизирующий слой. Первый скрытый слой — особенный, поскольку он получает входные данные как от входного слоя на текущем временном шаге, так и от смежного скрытого состояния на предыдущем временном шаге. Поэтому матрицы $W^{(k)}$ будут иметь размер $p \times (d + p)$ лишь для первого слоя (т.е. $k = 1$), где d — порядок входного вектора \bar{x}_t , а p — порядок скрытого вектора \bar{h}_t . Учтите, что в типичных случаях d не будет совпадать с p . Условие рекурсии для первого слоя легко получить из того, которое было приведено выше, положив в нем $W^{(1)} = W$. Поэтому сфокусируем внимание на всех скрытых слоях k для $k \geq 2$. Условие рекурсии для слоев с $k \geq 2$ также оказывается очень похожим на уже приведенное выше:

$$\bar{h}_t^{(k)} = \tanh W^{(k)} \begin{bmatrix} \bar{h}_t^{(k-1)} \\ \bar{h}_{t-1}^{(k)} \end{bmatrix}.$$

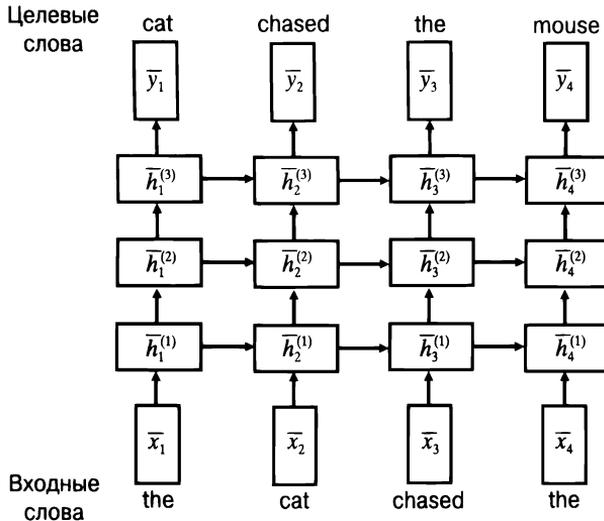


Рис. 7.6. Многослойная нейронная сеть

В данном случае размер матрицы $W^{(k)}$ равен $p \times (p + p) = p \times 2p$. Преобразование из входного слоя в выходной остается тем же, что и в однослойных сетях. Нетрудно заметить, что этот подход является непосредственным обобщением случая однослойных сетей на многослойные. Обычно в практических задачах используют два-три слоя. Чтобы задействовать большее количество слоев, важно иметь доступ к большому количеству тренировочных данных во избежание переобучения.

7.3. Трудности обучения рекуррентных сетей

Рекуррентные нейронные сети трудно поддаются обучению в силу того, что сеть с временными слоями является очень глубокой, особенно в случае длинных входных последовательностей. Иными словами, глубина временных слоев зависит от входа. Как и в случае любой глубокой сети, чувствительность функции потерь (т.е. ее градиенты) очень сильно варьируется для различных временных слоев. Кроме того, несмотря на то что величины градиентов функции потерь варьируются в широких пределах для различных слоев, различные временные слои совместно используют одни и те же матрицы параметров. Такое сочетание переменной чувствительности и разделяемых параметров в различных слоях может приводить к необычным эффектам нестабильности.

Основные трудности работы с рекуррентными нейронными сетями обусловлены *проблемами затухающих и взрывных градиентов* (см. раздел 3.4). В данном разделе мы повторно рассмотрим их в контексте рекуррентных нейронных сетей. Трудности, связанные с рекуррентными сетями, проще всего обсудить на примере рекуррентной сети, содержащей по одному элементу в каждом слое.

Рассмотрим набор из T последовательных слоев, между каждой парой которых применяется функция активации $\Phi(\cdot)$ в виде гиперболического тангенса (\tanh). Обозначим через w вес, разделяемый двумя скрытыми узлами. Пусть $h_1 \dots h_T$ — скрытые значения в различных слоях, а $\Phi'(h_t)$ — производная функции активации в скрытом слое t . Обозначим копию разделяемого веса w в t -м слое через w_t , чтобы можно было исследовать эффект обновления за счет обратного распространения ошибки. Пусть $\frac{\partial L}{\partial h_t}$ — производная функции потерь по скрытой активации h_t . Соответствующая нейронная архитектура представлена на рис. 7.7. Тогда для обновлений с помощью обратного распространения ошибки можно получить следующие уравнения:

$$\frac{\partial L}{\partial h_t} = \Phi'(h_{t+1}) \cdot w_{t+1} \cdot \frac{\partial L}{\partial h_{t+1}}. \quad (7.5)$$

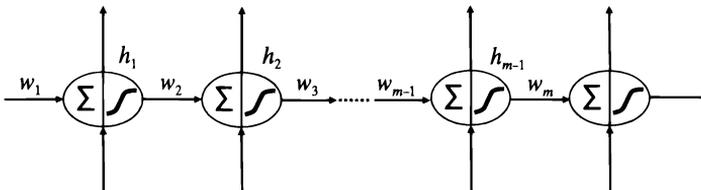


Рис. 7.7. Иллюстрация проблем затухающих и взрывных градиентов

Поскольку в различных временных слоях используются одни и те же разделяемые веса, градиент умножается на одну и ту же величину $w_t = w$ для каждо-

го слоя. Такое умножение будет всегда приводить к постоянному смещению в сторону затухания, если $w < 1$, и в сторону увеличения, если $w > 1$. Однако выбор функции активации также играет роль, поскольку в это произведение включается производная $\Phi'(h_{t+1})$. Например, в случае гиперболического тангенса, для которого производная $\Phi'()$ почти всегда меньше 1, увеличивается вероятность возникновения проблемы затухающих градиентов.

Несмотря на то что наши рассуждения охватывают лишь простейший случай скрытого слоя с одним элементом, эта аргументация легко обобщается на скрытый слой с несколькими элементами [220]. Можно показать, что в этом случае обновление градиента сводится к повторному умножению на одну и ту же матрицу A . В отношении этой матрицы справедливо следующее утверждение, сформулированное в форме леммы.

Лемма 7.3.1. *Пусть A — квадратная матрица, наибольшим собственным значением которой является λ . Тогда элементы матрицы A^t стремятся к 0 с увеличением t , если $\lambda < 1$. С другой стороны, элементы матрицы A^t монотонно возрастают до больших значений, если $\lambda > 1$.*

Этот результат можно доказать, диагонализировав матрицу $A = P\Delta P^{-1}$. Тогда, как нетрудно показать, $A^t = P\Delta^t P^{-1}$, где Δ — диагональная матрица. Величина наибольшего диагонального элемента матрицы Δ^t либо затухает с увеличением t , либо постоянно возрастает (по абсолютной величине), в зависимости от того, меньше или больше 1 собственное значение. В первом случае матрица A^t стремится к 0, и поэтому градиент затухает. Во втором случае градиент проявляет взрывное поведение. Разумеется, сюда еще не включены эффекты функции активации, а кроме того, можно установить порог для наибольшего собственного значения, чтобы задать условия для затухающих или взрывных градиентов. Например, наибольшее возможное значение производной сигмоиды равно 0,25; поэтому проблема затухающего градиента непременно возникнет, если наибольшее собственное значение меньше $1/0,25 = 4$. Конечно, можно комбинировать эффекты матричного умножения и функции активации в одной матрице Якоби (см. табл. 3.1) и протестировать ее собственные значения.

В конкретном случае рекуррентных нейронных сетей сочетание затухающего/взрывного градиента со связыванием параметров между различными слоями может привести к нестабильному поведению рекуррентной нейронной сети, в зависимости от размера шага градиентного спуска. Иначе говоря, если размер шага выбран слишком малым, то в результате влияния некоторых слоев сходимость процесса замедлится. С другой стороны, если размер шага выбран слишком большим, то в результате влияния некоторых слоев градиентный спуск “перепрыгнет” через оптимальную точку, порождая нестабильность. Здесь важно то, что градиент указывает наилучшее направление спуска лишь

для бесконечно малых шагов; в случае шагов конечной величины поведение обновлений будет существенно отличаться от того, что предсказывает градиент. В рекуррентных сетях оптимальные точки часто скрываются вблизи оврагов или других областей с непредсказуемым рельефом поверхности функции потерь, в результате чего наилучшие направления для *мгновенных* перемещений оказываются плохими предикторами лучших направлений для *конечных* перемещений. Поскольку для устойчивого продвижения в направлении оптимального решения любой практический алгоритм должен совершать конечные шаги разумной величины, это делает тренировку довольно затруднительной. Пример оврага представлен на рис. 7.8. Трудности, обусловленные оврагами, обсуждаются в разделе 3.5.4. Подробное обсуждение проблемы взрывных градиентов и ее геометрическая интерпретация содержится в [369].



Рис. 7.8. Повторение рис 3.13: пример оврага на поверхности функции потерь

Существует несколько способов решения проблем затухающих и взрывных градиентов, но не все из них одинаково эффективны. Так, простейшее решение заключается в использовании сильной регуляризации, которая уменьшает некоторые виды нестабильности, обусловленной проблемами затухающих и взрывных градиентов. Однако чрезмерно сильная регуляризация может привести к тому, что модель не сможет реализовать весь потенциал возможностей конкретной архитектуры нейронной сети. Второй способ, который обсуждался в разделе 3.5.5, — *отсечение градиентов* (gradient clipping), — хорошо подходит для решения проблемы взрывных градиентов. Наиболее часто используются два типа отсечения градиента. Первый из них основан на оценке значений, второй — на оценке нормы. В методе отсечения значений наибольшие временные компоненты градиента обрезаются, прежде чем будут добавлены. Эта первоначальная форма отсечения градиентов была предложена Миколовым в его докторской диссертации [324]. Второй тип отсечения предполагает отсечение по норме. Суть идеи заключается в том, что в тех случаях, когда норма всего

вектора градиента превышает установленный порог, он перенормируется до порогового значения. Оба типа отсечения работают аналогичным образом, и их анализ представлен в [368].

Следует отметить, что в случае рельефов с резко изменяющейся кривизной (например, крутые овраги) использования градиентов первого порядка обычно оказывается недостаточно для моделирования локальных поверхностей ошибок. В качестве естественного решения напрашивается использование градиентов более высоких порядков. Основной трудностью, которая возникает при этом, является значительное увеличение объема необходимых вычислений. Например, использование методов второго порядка (см. раздел 3.5.6) требует обращения матрицы Гессе. В случае сети, имеющей 106 параметров, это потребовало бы обращения матрицы размером 106×106 . Решить такую задачу с использованием имеющейся на сегодняшний день вычислительной техники практически невозможно. Однако недавно [313, 314] был предложен ряд остроумных способов реализации методов второго порядка без использования гессианов. Суть идеи заключается в отказе от точного вычисления матрицы Гессе и использовании только ее грубых приближений. Краткий обзор многих подобных методов см. в разделе 3.5.6. Эти методы позволили добиться определенных успехов и при обучении рекуррентных нейронных сетей.

Тип нестабильности, с которой сталкивается процесс оптимизации, зависит от специфики той точки на поверхности функции потерь, которая соответствует текущему решению. Поэтому очень многое зависит очень от выбора начальных точек. В [140] обсуждается ряд способов инициализации, позволяющих избегать возникновения нестабильностей в процессе обновления параметров. Использование методов на основе импульса (см. главу 3) также может помочь частично справиться с проблемами нестабильности. Возможности, обеспечиваемые надлежащей инициализацией параметров и использованием методов на основе импульса, обсуждаются в [478]. Надежные начальные состояния часто получают за счет использования упрощенных вариантов рекуррентных нейронных сетей, таких как *эхо-сети* (echo-state networks).

Другой полезный прием, к которому часто прибегают для устранения проблем затухающих и взрывных градиентов, заключается в использовании *пакетной нормализации* (batch normalization), хотя применение базового подхода к рекуррентным нейронным сетям требует внесения в него определенных изменений [81]. Методы пакетной нормализации обсуждаются в разделе 3.6. Однако применительно к рекуррентным сетям более эффективным оказывается один из их вариантов, известный как *послойная нормализация* (layer normalization). Методы послойной оптимизации оказались настолько успешными, что при использовании рекуррентных нейронных сетей или их вариантов они стали стандартной опцией.

Наконец, был предложен целый ряд вариантов рекуррентных нейронных сетей, специально предназначенных для снижения остроты проблем затухающих и взрывных градиентов. Первое упрощение заключается в использовании эхо-сетей, в которых матрицы весов “скрытый — скрытый” выбираются случайным образом, но обучаются только выходные слои. В первые годы, когда обучение рекуррентных нейронных сетей считалось трудной задачей, в качестве их жизнеспособной альтернативы использовались эхо-сети. Однако эти методы слишком просты, чтобы их можно было применять для решения сложных задач. Тем не менее они все еще используются для робастной инициализации рекуррентных нейронных сетей [478]. Более эффективный подход к решению проблем затухающих и взрывных градиентов состоит в наделении рекуррентной сети внутренней памятью, придающей большую стабильность состояниям сети. Использование *долгой краткосрочной памяти* (long short-term memory — LSTM) стало эффективным способом, позволяющим справиться с проблемами затухающих и взрывных градиентов. Этот подход вводит дополнительные состояния, которые можно интерпретировать как своего рода долгосрочную память. Долгосрочная память предоставляет состояния, стабильность которых со временем увеличивается, что повышает стабильность процесса градиентного спуска. Этот подход обсуждается в разделе 7.5.

7.3.1. Послойная нормализация

Методика пакетной нормализации, которая обсуждалась в разделе 3.6, предназначена для преодоления проблемы затухающих и взрывных градиентов в глубоких нейронных сетях. Несмотря на всю полезность такого подхода в большинстве типов нейронных сетей, его применение к рекуррентным нейронным сетям наталкивается на определенные трудности. Во-первых, пакетные статистики меняются от одного временного слоя нейронной сети к другому, и поэтому для различных временных шагов требуется поддерживать различные статистики. Кроме того, ряд слоев рекуррентной сети зависит от длины входной последовательности. В связи с этим, если длина тестовой последовательности превышает длину любой тренировочной последовательности, которая встречается в данных, мини-пакетные статистики могут оказаться недоступными для некоторых временных шагов. В общем случае мини-пакетные статистики, вычисленные для мини-пакетов в разных временных слоях, обладают разной степенью надежности (независимо от размера мини-пакета). Наконец, пакетную нормализацию нельзя применять к задачам онлайн-обучения. Одним из проблемных вопросов является то, что пакетная нормализация является не совсем обычной операцией для нейронной сети, поскольку активации элементов зависят от других тренировочных примеров пакета, а не только от текущего примера. И хотя пакетная нормализация может быть адаптирована для

рекуррентных сетей [81], более эффективным подходом является *последлойная нормализация* (layer normalization).

В этом случае нормализация выполняется лишь по одному тренировочному примеру, хотя для получения нормирующего множителя используются все текущие активации *данного слоя текущего примера*. Такой подход больше соответствует обычной работе сети, и при этом отпадает проблема поддержания мини-пакетных статистик. Вся информация, необходимая для вычисления активаций примера, может быть получена исключительно из данного примера!

Чтобы понять, как работает последлойная нормализация, вернемся к уравнению рекурсии “скрытый — скрытый”, которое приводилось в разделе 7.2:

$$\bar{h}_t = \tanh(W_{xh}\bar{x}_t + W_{hh}\bar{h}_{t-1}).$$

Поведение этой рекурсии подвержено нестабильности из-за мультипликативного эффекта временных слоев. Ниже будет показано, как следует видоизменить эту рекурсию для последлойной нормализации. Как и в случае обычной пакетной нормализации, описанной в главе 3, данный тип нормализации применяется к преактивационным значениям до применения активации \tanh . Поэтому преактивационное значение на t -м временном шаге вычисляется по следующей формуле:

$$\bar{a}_t = W_{xh}\bar{x}_t + W_{hh}\bar{h}_{t-1}.$$

Заметьте, что \bar{a}_t — это вектор, имеющий столько компонент, сколько элементов в скрытом слое (для количества которых в этой главе последовательно используется обозначение p). Математическое ожидание μ_t и стандартное отклонение σ_t преактивационных значений на шаге t вычисляются по следующим формулам:

$$\mu_t = \frac{\sum_{i=1}^p a_{ti}}{p}, \quad \sigma_t = \sqrt{\frac{\sum_{i=1}^p a_{ti}^2}{p} - \mu_t^2},$$

где a_{ti} обозначает i -ю компоненту вектора a_t .

Как и в случае пакетной нормализации, мы имеем дополнительные параметры обучения, связанные с каждым элементом. В частности, для p элементов t -го слоя имеем p -мерный вектор *параметров усиления* (gain parameters) $\bar{\gamma}_t$ и p -мерный вектор *параметров смещения* (bias parameters), обозначенный как $\bar{\beta}_t$. Эти параметры аналогичны параметрам γ_i, β_i , рассмотренным в разделе 3.6, где обсуждалась пакетная нормализация. Назначение параметров — перемасштабирование нормализованных значений и добавление смещений обучаемым способом. Поэтому скрытые активации \bar{h}_t следующего слоя вычисляются по формуле

$$\bar{h}_i = \tanh\left(\frac{\bar{\gamma}_i}{\sigma_i} \odot (\bar{a}_i - \bar{\mu}_i) + \bar{\beta}_i\right), \quad (7.6)$$

где \odot — операция поэлементного умножения, а $\bar{\mu}_i$ — вектор, содержащий p копий скаляра μ_i . Послойная нормализация гарантирует, что величины активаций не будут непрерывно расти или уменьшаться от одного временного шага к другому (что порождает проблемы затухающих и взрывных градиентов), хотя обучаемые параметры и обеспечивают определенную гибкость. В [14] было показано, что нормализация слоев в рекуррентных нейронных сетях обеспечивает лучшую производительность, чем пакетная нормализация. Некоторые родственные виды нормализации также могут использоваться для потокового и онлайн-обучения [294].

7.4. Эхо-сети

Эхо-сети представляют собой упрощенный вариант рекуррентных нейронных сетей. Они хорошо работают, когда размерность входа мала. Это обусловлено тем, что эхо-сети хорошо масштабируются с увеличением количества временных элементов, но не размерности входа. Поэтому сети данного типа являются удачным выбором для регрессионного моделирования одиночных или немногочисленных временных рядов с вещественными значениями в пределах сравнительно протяженного временного горизонта. В то же время такой выбор был бы неудачным в случае моделирования текста, когда размерность входа (в представлении прямого кодирования) определяется размером словаря документов. Тем не менее даже в этом случае эхо-сети приносят практическую пользу в качестве средства инициализации весов сети. Эхо-сети также называют *жидкими машинами* (liquid-state machines) [304], хотя для ясности следует заметить, что в жидких машинах используются *пиковые*, или *импульсные*, нейроны с бинарными выходами, тогда как в эхо-сетях применяются обычные функции активации, такие как сигмоида или гиперболический тангенс.

В эхо-сетях для связывания скрытых слоев между собой и даже с входным слоем используются *случайные веса*, хотя размерность скрытых состояний почти всегда намного превышает размерность входных состояний. В случае одиночных входных последовательностей нередко используют скрытые состояния с размерностью порядка 200. Поэтому обучается только выходной слой, в типичных случаях — с использованием линейного слоя для выходов с вещественными значениями. Отметим, что выходной слой просто агрегирует ошибки в различных выходных узлах, хотя веса в различных выходных узлах по-прежнему разделяются. Тем не менее целевая функция по-прежнему будет сводиться к случаю линейной регрессии, которую можно очень легко обучать, не прибегая

к обратному распространению ошибки. В результате обучение эхо-сети происходит очень быстро.

Как и в случае традиционных рекуррентных нейронных сетей, между скрытыми слоями используются нелинейные активации, такие как логистическая сигмоида, хотя возможны и такие активации, как гиперболический тангенс. В отношении инициализации весов между скрытыми элементами очень важно знать, что наибольший собственный вектор матрицы весов W_{hh} должен быть установлен в 1. Этого можно добиться путем симулирования весов матрицы W_{hh} случайным образом из стандартного нормального распределения и последующего деления каждого ее элемента на абсолютную величину наибольшего собственного значения $|\lambda_{max}|$ этой матрицы:

$$W_{hh} \leftarrow W_{hh} / |\lambda_{max}|. \quad (7.7)$$

В результате такой нормализации наибольшее собственное значение матрицы будет равно 1, что соответствует ее *спектральному радиусу* (spectral radius). Однако использование спектрального радиуса, равного 1, может быть слишком консервативным, поскольку в этом случае нелинейные активации будут оказывать ослабляющий эффект на значения состояний. Например, в случае использования сигмоидной активации наибольшим возможным значением частной производной сигмоиды всегда будет 0,25, поэтому использование спектрального радиуса, намного превышающего 4 (скажем, равного 10), вполне допустимо. В случае использования гиперболического тангенса в качестве функции активации разумным значением спектрального радиуса будет 2 или 3. Такой выбор все еще будет приводить к определенному ослаблению значений с течением времени, что фактически можно рассматривать как полезную регуляризацию, поскольку во временных рядах долговременные связи обычно намного слабее кратковременных. Также можно настроить спектральный радиус на основании производительности путем подбора различных значений масштабирующего множителя γ на отложенных данных для соотношения $W_{hh} = \gamma W_0$. Здесь W_0 — матрица, инициализированная случайными значениями.

В отношении связей между скрытыми слоями рекомендуется использовать разреженность, что нередко встречается в задачах, включающих преобразования со случайными проекциями. Можно семплировать ненулевые значения ряда соединений в W_{hh} , установив для других нулевые значения. Обычно количество таких соединений линейно растет с увеличением количества скрытых элементов.

Еще один действенный прием заключается в том, чтобы разбить скрытые элементы на группы, проиндексированные числами от 1 до K , и разрешить лишь связи между скрытыми состояниями, принадлежащими одной и той же группе. Можно показать, что такой подход эквивалентен тренировке ансамбля

эхо-сетей (см. упражнение 2). Еще одна проблема связана с настройкой матриц связей W_{xh} между входным и скрытым слоями. Масштабируя эту матрицу, необходимо проявлять осторожность, иначе влияние входов на каждом временном шаге может серьезно исказить информацию, переносимую в скрытые состояния из предыдущего временного шага. Поэтому матрица W_{xh} сначала устанавливается случайным образом в W_1 , а затем масштабируется различными значениями гиперпараметра β для определения окончательной матрицы $W_{xh} = \beta W_1$, которая приводит к наилучшей точности для отложенных данных.

В основе эхо-сетей лежит старая идея о том, что увеличение количества признаков набора данных за счет нелинейных преобразований часто способно увеличить выразительную мощность входного представления. Например, как сеть RBF (см. главу 5), так и ядерный метод опорных векторов обретают свою мощь за счет расширения базового пространства признаков в соответствии с теоремой Ковера о разделимости образов [84]. Единственным отличием является то, что эхо-сеть выполняет расширение признаков посредством случайной проекции. Такой подход не лишен прецедентов, поскольку различные типы случайных преобразований используются также в машинном обучении в качестве быстрых альтернатив ядерным методам [385, 516]. Следует отметить, что эффективность расширения признаков обусловлена главным образом применением нелинейных преобразований, а они обеспечиваются активациями в скрытых слоях. В определенном смысле эхо-сеть работает на основе использования принципов RBF-сети применительно к данным временных рядов, точно так же, как рекуррентную нейронную сеть можно рассматривать в качестве замены сетей прямого распространения для данных такого типа. Эхо-сеть нуждается лишь в небольшом обучении для извлечения признаков, в чем она сходна RBF-сетью, и опирается на рандомизированное расширение пространства признаков.

В случае применения к данным временных рядов этот подход демонстрирует превосходные результаты при предсказании значений, относящихся к далеко отстоящим будущим моментам времени. Здесь ключевым является выбор таких целевых выходных значений на шаге t , которые соответствуют входным значениям временного ряда на шаге $t + k$, где k — требуемое опережение для предсказания. Другими словами, эхо-сеть — отличная нелинейная авторегрессионная техника для моделирования временных рядов. Этот подход можно использовать даже для многомерных временных рядов, хотя в случае длинных временных рядов его не рекомендуется применять, поскольку размерность скрытых состояний, необходимых для моделирования, окажется слишком большой. Применение эхо-сетей для моделирования временных рядов подробно обсуждается в разделе 7.7.5. В этом же разделе данный подход сравнивается с традиционными предсказательными моделями для временных рядов.

Несмотря на то что в случае входов очень большой размерности (таких, как текст) описанный подход невозможно реализовать на практике, он хорошо подходит для инициализации параметров [478]. Суть идеи заключается в том, чтобы инициализировать рекуррентную сеть путем использования варианта эхо-сети для обучения выходного слоя. Кроме того, инициализированные значения W_{hh} и W_{xh} можно подвергнуть надлежащему масштабированию путем перебора различных значений масштабирующих множителей β и γ (в соответствии с приведенным выше обсуждением). Для дальнейшей тренировки рекуррентной сети используется традиционный алгоритм обратного распространения ошибки. Такой подход можно рассматривать как упрощенный вариант предварительного обучения рекуррентных сетей.

Нам осталось обсудить разреженность соединений. Должна ли матрица W_{hh} быть разреженной? Мнения по этому вопросу в значительной степени расходятся и довольно противоречивы. Несмотря на то что на первых порах [219] для эхо-сетей рекомендовалось использовать разреженные соединения, четкие доводы в пользу этого приведены не были. В оригинальной работе [219] утверждается, что разреженность соединений приводит к разрыву связей между индивидуальными подсетями, что способствует развитию их индивидуальной динамики. По-видимому, это может служить аргументом в пользу увеличения разнородности признаков, выучиваемых эхо-сетью. Если целью действительно является разрыв связей между подсетями, то имело бы больше смысла делать это явным образом, разбивая скрытые состояния на несвязанные группы. Такой подход допускает интерпретацию в терминах ансамблей. Для повышения эффективности вычислений часто рекомендуется увеличивать разреженность также в методах, включающих случайные проекции. Наличие плотных соединений может приводить к тому, что активации различных состояний будут внедряться в мультипликативный шум многочисленных гауссовских случайных переменных, затрудняющих выделение признаков.

7.5. Долгая краткосрочная память (LSTM)

Как обсуждалось в разделе 7.3, рекуррентные нейронные сети сталкиваются с проблемами, обусловленными затухающими и взрывными градиентами [205, 368, 369]. Это распространенная проблема, свойственная обновлениям параметров в нейронных сетях, когда последовательное умножение на матрицу $W^{(k)}$ вносит нестабильность, что приводит либо к затуханию градиентов в процессе обратного распространения ошибки, либо к их взрывному росту до больших значений в нестабильной манере. Такой тип нестабильности является прямым следствием последовательного умножения на (рекуррентную) матрицу весов на различных временных шагах. На проблему можно взглянуть с той точки зрения, что нейронная сеть, использующая лишь мультипликативные

обновления, хорошо обучается лишь на коротких последовательностях, и поэтому ей внутренне присуща хорошая краткосрочная память и плохая долгосрочная [205]. Решением проблемы является изменение уравнения рекурсии для вектора скрытых состояний путем включения в него эффектов *долгой краткосрочной памяти* (long short-term memory — LSTM). Операции LSTM обеспечивают контроль над данными, относящимися к коротким временным промежуткам и записанными в эту долгосрочную память.

Как и в предыдущих разделах, обозначение $\bar{h}_i^{(k)}$ представляет скрытые состояния k -го слоя многослойной LSTM. С целью упрощения нотации мы также примем для входного вектора \bar{x}_i обозначение $\bar{h}_i^{(0)}$ (хотя совершенно очевидно, что этот слой не является скрытым). Как и в случае рекуррентной сети, входной вектор \bar{x}_i — d -мерный, тогда как скрытые состояния — p -мерные. LSTM — это улучшенный вариант архитектуры рекуррентной нейронной сети, приведенной на рис. 7.6, в которой мы изменяем условия рекурсии, управляющие распространением состояний $\bar{h}_i^{(k)}$. Для реализации этой цели в нашем распоряжении имеется дополнительный скрытый вектор порядка p , который называется *вектором состояний ячеек* и обозначается как $\bar{c}_i^{(k)}$. Состояние ячеек можно рассматривать как своего рода долгосрочную память, которая удерживает по крайней мере часть информации о более ранних состояниях путем применения операций частичного “забывания” и “запоминания” к предыдущим состояниям. Было показано [233], что природа памяти $\bar{c}_i^{(k)}$ такова, что иногда она поддается интерпретации, если применяется к текстовым данным, таким как отрывки литературных произведений. Например, одно из p значений в $\bar{c}_i^{(k)}$ может обратить свой знак после того, как встретится открывающая кавычка, а затем, когда встретится закрывающая кавычка, восстановить его. В конечном итоге результирующая нейронная сеть обретает способность моделировать дальние языковые зависимости или даже некоторые образцы (такие, как цитаты), охватывающие большое количество лексем. Это достигается за счет использования мягкого подхода к обновлению состояний ячеек со временем, что повышает персистентность сохраняемой информации. Персистентность значений состояний позволяет избежать нестабильности того типа, который встречается в случае проблем исчезающих и взрывных градиентов. На интуитивном уровне это можно попытаться объяснить тем, что увеличение степени схожести состояний в различных временных слоях (посредством долгосрочной памяти) препятствует резкому изменению градиентов по весам входящих связей.

Как и в случае многослойной рекуррентной сети, матрица обновлений, обозначаемая как $W^{(k)}$, используется для умножения на вектор-столбец $[\bar{h}_i^{(k-1)}, \bar{h}_{i-1}^{(k)}]^T$. Однако эта матрица имеет размер $4p \times 2p$, поэтому ее умножение на вектор порядка

² В первом слое матрица $W^{(1)}$ имеет размер $4p \times (p + d)$, поскольку она умножается на вектор порядка $(p + d)$.

$2p$ дает вектор порядка $4p$. В данном случае обновления используют четыре промежуточные векторные переменные \bar{i} , \bar{f} , \bar{o} и \bar{c} , которым соответствует $4p$ -мерный вектор. Промежуточные переменные \bar{i} , \bar{f} и \bar{o} называются соответственно переменными *входа* (input), *забывания* (forget) и *выхода* (output), учитывая их роли в обновлении состояний ячеек и скрытых состояний. Определение вектора скрытых состояний $\bar{h}_i^{(k)}$ и вектора состояний ячеек $\bar{c}_i^{(k)}$ использует многоступенчатый процесс, начинающийся с вычисления этих промежуточных переменных и заканчивающийся вычислением скрытых переменных на их основе. Обратите внимание на различие между промежуточным переменным вектором \bar{c} и первичным состоянием ячеек $\bar{c}_i^{(k)}$, играющих совершенно разные роли. Обновления вычисляются следующим образом:

$$\begin{array}{l} \text{Входной вентиль:} \\ \text{Вентиль забывания:} \\ \text{Выходной вентиль:} \\ \text{Новое состояние ячеек:} \end{array} \quad \begin{array}{l} \left[\bar{i} \right] \\ \left[\bar{f} \right] \\ \left[\bar{o} \right] \\ \left[\bar{c} \right] \end{array} = \begin{array}{l} \left(\begin{array}{l} \text{сигмоида} \\ \text{сигмоида} \\ \text{сигмоида} \\ \text{tanh} \end{array} \right) W^{(k)} \begin{array}{l} \left[\bar{h}_i^{(k-1)} \right] \\ \left[\bar{h}_{i-1}^{(k)} \right] \end{array} \quad [\text{Установка вентиляей}]$$

$$\bar{c}_i^{(k)} = \bar{f} \odot \bar{c}_{i-1}^{(k)} + \bar{i} \odot \bar{c} \quad [\text{Селективное забывание и заполнение долгосрочной памяти}]$$

$$\bar{h}_i^{(k)} = \bar{o} \odot \tanh(\bar{c}_i^{(k)}) \quad [\text{Селективная утечка из долгосрочной памяти в скрытое состояние}]$$

Здесь символом \odot обозначена операция поэлементного умножения. Для каждого первого слоя (т.е. для $k = 1$) обозначение $\bar{h}_i^{(k-1)}$ в приведенном выше уравнении должно быть заменено обозначением \bar{x}_i , а матрица $W^{(1)}$ имеет размер $4p \times (p + d)$. В практических реализациях в вышеприведенных уравнениях используются также смещения³, хотя здесь они опущены для упрощения. Эти уравнения кажутся таинственными и потому нуждаются в дополнительных пояснениях.

В приведенной выше последовательности уравнений первым шагом является задание промежуточных векторных переменных \bar{i} , \bar{f} и \bar{o} , из которых первые три должны *концептуально* рассматриваться как бинарные значения, хотя в действительности они являются непрерывными значениями в интервале $(0, 1)$. Умножение пары бинарных значений подобно применению вентиля AND (И) к паре булевых значений. Поэтому далее мы будем называть такую операцию

³ Смещение, связанное с вентилями забывания, играет особенно важную роль. Обычно смещение вентиля забывания инициализируется значениями, превышающими 1 [228], поскольку это, по всей видимости, позволяет избежать проблем загухающих и взрывных градиентов на стадии инициализации.

управлением пропуском. Векторы \bar{i} , \bar{f} и \bar{o} называются входным вентиляем, вентиляем забывания и выходным вентиляем соответственно. В частности, эти векторы концептуально применяются в качестве булевых вентиляей, управляющих 1) добавлением информации в состояние ячейки, 2) забыванием состояния ячейки и 3) использованием информации из ячейки скрытым состоянием. Использование бинарной абстракции для этих векторных переменных способствует лучшему пониманию типов решений, принимаемых вентилями. На практике в этих переменных содержатся непрерывные значения из интервала $(0, 1)$, что позволяет интерпретировать эффект бинарных вентиляей на вероятностной основе, если рассматривать выход как вероятность. В условиях нейронной сети возможность работы с непрерывными функциями очень важна, поскольку обеспечивает дифференцируемость функций, необходимых для градиентных обновлений. Вектор \bar{c} содержит вновь предложенное содержимое состояния ячейки, хотя входной вентилю и вентилю забывания регулируют, в какой мере этой информации разрешается изменить предыдущее состояние ячейки (для поддержания долгосрочной памяти).

Четыре промежуточные векторные переменные \bar{i} , \bar{f} , \bar{o} и \bar{c} устанавливаются с использованием матриц весов $W^{(k)}$ для k -го слоя в первом из приведенных выше уравнений. Исследуем второе уравнение, которое обновляет состояние ячейки за счет использования некоторых из этих промежуточных переменных:

$$\bar{c}_t^{(k)} = \underbrace{\bar{f} \odot \bar{c}_{t-1}^{(k)}}_{\text{Сбросить?}} + \underbrace{\bar{i} \odot \bar{c}}_{\text{Инкрементировать?}}$$

Это уравнение имеет две части. Первая часть использует p битов забывания из вектора \bar{f} для принятия решения о том, какие из p состояний ячейки из предыдущего временного шага должны быть сброшены⁴ в нуль, тогда как вторая часть использует p входных битов из вектора \bar{i} для принятия решения о том, следует ли добавить соответствующие компоненты из \bar{c} в каждое из состояний ячейки. Обратите внимание на то, что подобные обновления состояний ячеек осуществляются в аддитивной форме, что позволяет избежать проблемы затухающих градиентов, обусловленной мультипликативными обновлениями. Вектор состояний ячеек можно рассматривать как непрерывно обновляемую память, в которой биты забывания и биты входа определяют соответственно, следует ли 1) сбросить состояния ячеек, относящиеся к предыдущим временным шагам, и забыть о предыстории, или 2) инкрементировать состояния ячеек

⁴ Здесь мы трактуем биты забывания как вектор бинарных битов, хотя он содержит вещественные значения в интервале $(0, 1)$, которые можно рассматривать как вероятности. Как ранее обсуждалось, бинарная абстракция облегчает понимание концептуальной природы этих операций.

из предыдущих шагов для включения новой информации в долгосрочную память из текущего слова. Этот вектор содержит p величин, которыми могут быть инкрементированы состояния ячеек, и их значения принадлежат к интервалу $[-1, +1]$, поскольку все они являются выходами функции активации в виде гиперболического тангенса.

Наконец, скрытые состояния $\bar{h}_i^{(k)}$ обновляются с использованием “утечки” информации из состояния ячейки. Скрытые состояния обновляются в соответствии со следующей формулой:

$$\bar{h}_i^{(k)} = \underbrace{\bar{o} \odot \tanh(\bar{c}_i^{(k)})}_{\text{Утечка из } \bar{c}_i^{(k)} \text{ в } \bar{h}_i^{(k)}}.$$

Здесь мы копируем функциональную форму каждого из p состояний ячейки, в зависимости от того, каково значение выходного вентиля (определяемого вектором \bar{o}): 0 или 1. Разумеется, в условиях нейронной сети, когда мы имеем дело с непрерывными значениями, происходит частичное пропускание сигналов, и только некоторая их доля копируется из каждого состояния ячейки в соответствующее скрытое состояние. Следует отметить, что в последнем уравнении не всегда используется функция гиперболического тангенса. Также возможна следующая альтернативная форма этого уравнения:

$$\bar{h}_i^{(k)} = \bar{o} \odot \bar{c}_i^{(k)}.$$

Как и в случае любой нейронной сети, для обучения применяется алгоритм обратного распространения ошибки.

Чтобы понять, почему LSTM-сети обеспечивают лучший поток градиента, чем простые сети RNN, рассмотрим обновление для простой сети LSTM с одним слоем и $p = 1$. В таком случае обновление ячейки упрощается и приобретает следующий вид:

$$c_i = c_{i-1} * f + i * c. \quad (7.8)$$

Соответственно, частная производная c_i по c_{i-1} равна f , а это означает, что обратные потоки градиента для c_i умножаются на значение вентиля забывания f . Учитывая поэлементный характер операций, этот результат обобщается на произвольные значения размерности состояния p . Для смещения вентиля забывания первоначально устанавливаются высокие значения, чтобы сделать затухание потока градиентов относительно меленным. Вектор забывания f также может быть разным на разных временных шагах, что снижает вероятность возникновения проблемы затухающих градиентов. Скрытые состояния могут быть выражены в терминах состояний ячеек в виде $h_i = o * \tanh(c_i)$, что позволяет вычислять частную производную по h_i с использованием одной производной гиперболического тангенса. Иными словами, долгосрочные состояния ячеек функционируют как супермагистраль для перетекания информации в скрытые состояния.

смещения в приведенных выше уравнениях, однако в практических реализациях они обычно включаются. Далее мы предоставим дополнительное объяснение этих уравнений и сравним их с аналогичными уравнениями для LSTM.

Точно так же, как для принятия решения о том, в какой мере информация из предыдущего временного шага должна быть использована на следующем шаге, в LSTM используются входной и выходной вентили, а также вентиль забывания, в GRU используются вентили обновления и сброса. GRU не имеет отдельной внутренней памяти и требует дополнительных вентилях для выполнения обновления от одного скрытого состояния к другому. В связи с этим возникает вполне естественный вопрос о точном определении ролей вентилях обновления и сброса. Вентиль обновлений \bar{r} решает, в какой мере скрытое состояние должно переноситься из предыдущего временного шага во время обновления посредством матричного преобразования (как в случае рекуррентной нейронной сети). Вентиль обновления \bar{z} принимает решение об относительной доле вкладов этого обновления на основе матрицы и более непосредственного вклада от вектора скрытых состояний $\bar{h}_t^{(k-1)}$ на предыдущем временном шаге. Благодаря участию непосредственной (частичной) копии скрытых состояний из предыдущего слоя в обновлении поток градиента становится более стабильным в процессе обратного распространения ошибки. Вентиль обновления GRU совмещает роли входного вентиля и вентиля забывания, используемых в LSTM, в форме \bar{z} и $1 - \bar{z}$ соответственно. Однако такое уподобление GRU и LSTM не является точным, поскольку обновляются непосредственно скрытые состояния (состояния ячеек отсутствуют). Как и вентили в LSTM, вентили обновления и сброса в GRU играют роль оперативной памяти.

Чтобы разобраться в том, за счет чего GRU обеспечивают лучшую производительность по сравнению с простыми RNN, рассмотрим GRU с одним слоем и одним состоянием размерности $p = 1$. В этом случае уравнение GRU может быть переписано в следующем виде:

$$h_t = z \cdot h_{t-1} + (1 - z) \cdot \tanh[v_1 \cdot x_t + v_2 \cdot r \cdot h_{t-1}]. \quad (7.9)$$

Обратите внимание на то, что ввиду наличия только одного слоя верхний индекс, указывающий на слой, отсутствует. Здесь v_1 и v_2 — два элемента матрицы V , имеющей размер 2×1 . Тогда, как нетрудно заметить,

$$\frac{\partial h_t}{\partial h_{t-1}} = z + (\text{аддитивные члены}). \quad (7.10)$$

Обратный поток градиента умножается на этот множитель. Здесь член $z \in (0, 1)$ содействует беспрепятственной передаче потока градиента и придает стабильность вычислениям. Кроме того, поскольку аддитивные члены в значительной степени зависят от величины $(1 - z)$, общий мультипликативный

фактор стремится быть ближе к 1, даже если z мало. Также следует отметить, что значение z и мультипликативный фактор $\frac{\partial h_t}{\partial h_{t-1}}$ различны для каждого временного шага, что уменьшает вероятность возникновения проблем затухающих и взрывных градиентов.

Несмотря на то что GRU — это упрощенный вариант LSTM, его не следует рассматривать как частный случай LSTM. Сравнительный анализ LSTM и GRU приведен в [71, 228]. Показано, что обе модели обеспечивают примерно одинаковую производительность, а относительная производительность зависит от специфики конкретной задачи. GRU проще и имеет незначительное преимущество в отношении легкости реализации и производительности. GRU обобщается несколько легче при меньших объемах данных, поскольку имеет гораздо меньше параметров [71], хотя с увеличением объема доступных данных LSTM становится более предпочтительной. Вопросы практической реализации LSTM обсуждаются в [228]. LSTM тестировалась гораздо интенсивнее, чем GRU, лишь потому, что ее архитектура была разработана раньше и обрела широкую популярность. В результате этого LSTM считается более безопасным решением, особенно при работе с длинными последовательностями и крупными наборами данных. Также было показано [160], что ни один из вариантов LSTM не способен всегда работать одинаково надежно, уступая в этом отношении модели LSTM, имеющей явную внутреннюю память и осуществляющей более полный контроль над обновлениями посредством вентиляей.

7.7. Применение рекуррентных нейронных сетей

Рекуррентные нейронные сети находят многочисленные применения в различных задачах машинного обучения, связанных с извлечением информации, а также распознаванием речи и рукописного текста. Основной областью применения RNN является обработка текстовых данных, но также имеются примеры их применения в вычислительной биологии. Большинство приложений RNN можно отнести к одной из двух категорий.

1. **Условное языковое моделирование.** Если выходом рекуррентной сети является языковая модель, то ее можно улучшить за счет учета контекстного окружения, чтобы модель лучше соответствовала контексту. В большинстве случаев контекстом является нейронный выход другой нейронной сети. В качестве одного из примеров можно привести аннотирование изображений подписями, когда контекстом является нейронное представление изображения, обеспечиваемое сверточной сетью, в то время как языковая модель предоставляет соответствующую текстовую аннотацию (подпись) к нему. В машинном переводе контекстом является

представление предложения на исходном языке (создаваемое другой RNN), а языковая модель целевого языка обеспечивает перевод.

- Использование выходов, специфических для лексем.** Выходы, относящиеся к различным лексемам, можно использовать для обучения свойствам, не охватываемым языковой моделью. Например, можно предусмотреть метки выхода на различных временных шагах, соответствующие свойствам лексем (таким, как отнесение к частям речи). При распознавании рукописного текста метки могут соответствовать символам. В некоторых случаях выходы на всех временных шагах могут отсутствовать, но маркер конца предложения может выводить метку для всего предложения. Такой подход, который носит название *классификация на уровне предложений* (sentence-level classification), часто используют в сентимент-анализе (анализе тональности текста). В некоторых из этих приложений используются двунаправленные рекуррентные сети, позволяющие учитывать двухсторонний контекст.

Излагаемый далее материал представляет собой обзор многочисленных применений рекуррентных сетей. Для простоты изложения и упрощения соответствующих графических иллюстраций мы в основном будем ссылаться на однослойную рекуррентную сеть. В то же время в большинстве случаев используется многослойная LSTM. В других случаях используется двунаправленная LSTM, поскольку она обеспечивает лучшую производительность. Замена однослойной RNN на многослойную/двунаправленную LSTM в любом из описанных ниже приложений осуществляется непосредственно. Нашей более широкой целью является демонстрация того, каким образом это семейство архитектур можно использовать для решения очерченных задач.

7.7.1. Автоматическое аннотирование изображений

В задачах аннотирования изображений тренировочный набор данных состоит из пар “изображение — подпись”. В качестве примера на рис. 7.9 слева приведено изображение взятое на сайте Национального управления по аэронавтике и космонавтике (NASA)⁶. Это изображение аннотировано словами “cosmic winter wonderland”. Таких пар “изображение — подпись” могут быть сотни тысяч. Эти пары используются для обучения весов нейронной сети. По завершении тренировки предсказываются аннотации для неизвестных тестовых примеров. Следовательно, такой подход является примером обучения “изображение — последовательность”.

⁶ https://www.nasa.gov/mission_pages/chandra/cosmic-winter-wonderland.html.

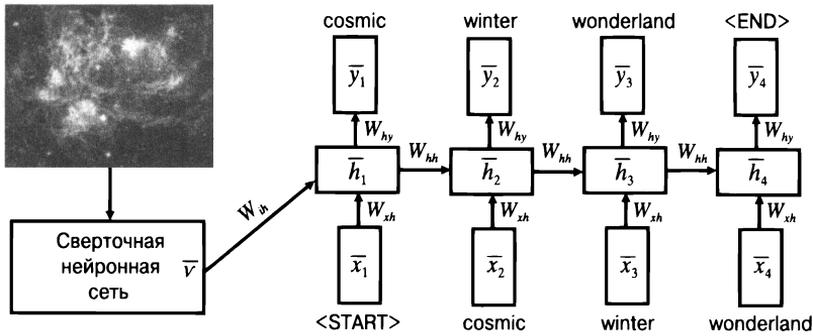


Рис. 7.9. Пример снабжения изображений подписями с помощью рекуррентной нейронной сети. Для обучения представлению изображений требуется дополнительная нейронная сеть. Изображение представляется вектором \bar{v} , являющимся выходом сверточной нейронной сети. Изображение во вставке публикуется с разрешения Национального управления по авиации и космонавтике (NASA)

Одна из проблем автоматического аннотирования изображений заключается в том, что обучение представлению изображений требует использования отдельной сети. Обычной архитектурой для обучения представлению изображений является *сверточная нейронная сеть* (CNN). Сверточные нейронные сети подробно обсуждаются в главе 8. Рассмотрим задачу, в которой CNN формирует q -мерный вектор \bar{v} в качестве выходного представления. Затем этот вектор используется в качестве входа нейронной сети, но только⁷ на первом временном шаге. Для учета этого дополнительного входа требуется еще одна матрица W_{ih} размером $p \times q$, которая транслирует представление изображения на скрытый слой. Таким образом, теперь уравнение обновления для различных слоев должно быть переписано в следующем виде:

$$\begin{aligned}\bar{h}_1 &= \tanh(W_{xh}\bar{x}_1 + W_{ih}\bar{v}), \\ \bar{h}_t &= \tanh(W_{xh}\bar{x}_t + W_{hh}\bar{h}_{t-1}) \quad \forall t \geq 2, \\ \bar{y}_t &= W_{hy}\bar{h}_t.\end{aligned}$$

Здесь важно то, что CNN и RNN тренируются не по отдельности. И хотя в целях инициализации их можно тренировать изолированно друг от друга, окончательные веса всегда тренируются совместно путем прогона каждого изображения через сеть и сопоставления предсказанной подписи с истинной. Другими словами, для каждой пары “изображение — подпись” в тех случаях, когда любой отдельный токен (лексема) подписи предсказывается неправильно, обнов-

⁷ В принципе, можно разрешить использование этого вектора в качестве входа на всех временных шагах, но, по всей видимости, это только ухудшает производительность.

ляются веса обеих сетей. На практике эти ошибки являются “размытыми”, поскольку токены в каждой точке предсказываются на вероятностной основе. Такой подход гарантирует, что обучаемое представление \bar{v} изображений будет чувствительным к специфике применения предсказываемых подписей.

Когда все веса обучены, тестовое изображение служит входом для всей системы и пропускается как через сверточную, так и рекуррентную сеть. Для рекуррентной сети входом на первом временном шаге является токен <START> и представление изображения. На более поздних временных шагах входом, вероятнее всего, будет токен, предсказанный на предыдущем временном шаге. Также возможно использование лучевого поиска с целью отслеживания b наиболее вероятных префиксов последовательности для расширения в каждой точке. Такой подход не слишком отличается от подхода на основе генерации языка, который обсуждался в разделе 7.2.1.1, за исключением того, что он привязан к представлению изображения, которое служит входом для модели на первом временном шаге рекуррентной сети. Результатом является предсказание подписи к изображению.

7.7.2. Обучение “последовательность в последовательность” и машинный перевод

Точно так же, как сверточную нейронную сеть можно объединить с рекуррентной для аннотирования изображений, можно объединить две рекуррентные сети для перевода текста с одного языка на другой. Такие методы часто называют обучением “последовательность в последовательность” (sequence-to-sequence learning), поскольку последовательность на одном языке сопоставляется с последовательностью в другом. Вообще говоря, обучение “последовательность в последовательность” может иметь и другие применения помимо машинного перевода. Например, даже системы “вопрос — ответ” (QA) могут рассматриваться как приложения обучения “последовательность в последовательность”.

Ниже мы опишем простое решение для машинного перевода с помощью рекуррентных сетей, хотя такие приложения редко реализуются с использованием непосредственно простых форм рекуррентных нейронных сетей. Вместо этого обычно применяется вариация рекуррентной нейронной сети, известная как модель LSTM (long short-term memory — долгая краткосрочная память). Такая модель ведет себя лучше при изучении долгосрочных зависимостей и поэтому может хорошо работать с длинными предложениями. Поскольку общий подход к использованию RNN также применим к LSTM, мы проведем обсуждение машинного перевода на примере (простой) RNN. Обсуждение LSTM было дано в разделе 7.5, а обобщение приложения машинного перевода на LSTM не составляет труда.

В приложении машинного перевода две различные RNN стыкуются друг с другом точно так же, как сверточная и рекуррентная нейронные сети стыкуются для аннотирования изображений. Первая рекуррентная сеть использует поступающие на вход слова исходного языка. На этих временных шагах не генерируются никакие выходы, и последовательные временные шаги аккумулируют знания об исходном предложении в скрытом состоянии. Далее встречается символ, обозначающий конец предложения, и вторая рекуррентная сеть начинает свою работу с вывода первого слова языка перевода (целевого языка). Следующий набор состояний во второй рекуррентной сети выводит слова предложения на языке перевода одно за другим. Эти состояния также используют слова языка перевода в качестве входа, которые доступны в случае тренировочных примеров, но не доступны в случае тестовых примеров (вместо которых используются предсказанные значения). Эта архитектура представлена на рис. 7.10.

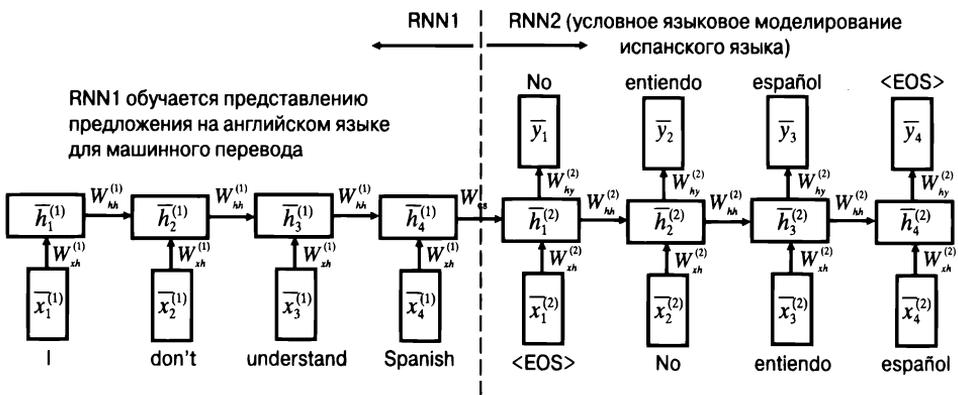


Рис. 7.10. Машинный перевод с использованием RNN. Обратите внимание на наличие двух отдельных рекуррентных сетей, каждая из которых имеет собственный набор разделяемых весов. Выход $\bar{h}_4^{(1)}$ является кодированным представлением фиксированной длины предложения из 4 слов на английском языке

Приведенная на рис. 7.10 архитектура аналогична архитектуре автокодировщика и даже может использоваться с парами идентичных предложений на одном и том же языке для создания представлений предложений фиксированной длины. Веса, используемые в этих двух сетях, обозначенных как RNN1 и RNN2, не совпадают. Например, матрица весов связей между двумя скрытыми узлами на последовательных временных шагах в RNN1 обозначена как $W_{hh}^{(1)}$, а соответствующая матрица в RNN2 — как $W_{hh}^{(2)}$. Матрица весов W_{es} , связывающая две нейронные сети, играет особую роль и может быть независимой от обеих сетей. Это необходимо, если векторы скрытых состояний в обеих RNN имеют разные порядки, поскольку размер матрицы W_{es} будет отличаться от размеров матриц $W_{hh}^{(1)}$

и $W_{hh}^{(2)}$. В целях упрощения⁸ можно использовать в обеих сетях скрытые векторы одного порядка и положить $W_{es} = W_{hh}^{(1)}$. Веса в RNN1 предназначены для обучения кодированию входа на исходном языке, а веса в RNN2 — для использования этого кодирования с целью создания выходного предложения на языке перевода. Эту архитектуру можно считать аналогичной архитектуре, используемой в задаче аннотирования изображений, за исключением того, что мы задействуем две рекуррентные сети вместо сверточной и рекуррентной. Выходом последнего скрытого узла RNN1 является кодировка исходного предложения, имеющая фиксированную длину. Поэтому, независимо от длины предложения, кодировка исходного предложения зависит от размерности скрытого представления.

Грамматика и длина предложения на исходном языке и на языке перевода могут отличаться. Чтобы предоставить грамматически правильный вывод на языке перевода, RNN2 должна обучиться его языковой модели. Следует отметить, что элементы в RNN2, ассоциируемой с языком перевода, имеют как входы, так и выходы, расположенные так же, как в RNN, моделирующей язык. В то же время выход RNN2 обусловлен входом, который она получает от RNN1, результатом чего в конечном счете и является перевод на другой язык. Для достижения этой цели используются тренировочные пары слов на исходном языке и на языке перевода. При таком подходе пары слов “исходный — целевой” пропускаются через архитектуру, приведенную на рис. 7.10, и параметры модели обучаются с использованием алгоритма обратного распространения ошибки. Так как выходы имеют лишь узлы в RNN2, то для тренировки весов в обеих сетях обратному распространению подвергаются лишь ошибки, допущенные при предсказании слов языка перевода. Обе сети обучаются совместно, поэтому ошибки в переводе на выходе RNN2 минимизируются путем оптимизации обеих сетей. С практической точки зрения это означает, что внутреннее представление исходного языка, которому обучается RNN1, в высшей степени оптимизировано для машинного перевода и значительно отличается от того представления, которому она обучилась бы, если бы использовалась для языкового моделирования предложений на исходном языке. Когда параметры будут обучены, перевод предложения на исходном языке начинается с его прогона через RNN1 с целью получения необходимого ввода для RNN2. Помимо этого контекстного входа дополнительным входом первого элемента RNN2 служит тег <EOS>, встретив который RNN2 выводит вероятность первого токена на языке перевода. Наиболее вероятный токен выбирается с помощью *лучевого поиска* (см. раздел 7.2.1.1) и используется в качестве входа элемента рекуррентной

⁸ По-видимому, именно такой подход был использован в оригинальной работе [478]. В системе Google Neural Machine Translation [579] этот вес исключен. В настоящее время эта система используется в веб-службе Google Translate.

сети на следующем временном шаге. Этот процесс применяется рекурсивно до тех пор, пока на выходе элемента RNN2 снова не появится тег <EOS>. Как и в разделе 7.2.1.1, мы генерируем предложение на языке перевода, используя подход на основе языкового моделирования, за исключением того, что конкретный вывод обуславливается внутренним представлением предложения на исходном языке.

Нейронные сети стали лишь сравнительно недавно использоваться для машинного перевода. Сложность моделей на основе рекуррентной нейронной сети намного превышает сложность традиционных моделей машинного обучения. В последнем классе методов часто используют машинное обучение, ориентированное на фразы, которое не настолько продвинуто, чтобы обучаться тонким различиям между грамматиками обоих языков. На практике для улучшения производительности применяют глубокие модели со многими слоями.

Одной из слабых сторон таких моделей перевода является то, что они, как правило, плохо работают с длинными предложениями. Для решения этой проблемы было создано множество решений. Одно из них заключается в том, чтобы предложение на исходном языке подавалось на вход в обратном порядке [478]. Такой подход обеспечивает большее сближение первых нескольких слов предложения в обоих языках в терминах их временных меток в архитектуре рекуррентной нейронной сети. Это приводит к тому, что первые несколько слов на языке перевода будут с большей вероятностью предсказаны корректно. Корректность прогнозирования первых нескольких слов также способствует прогнозированию последующих слов, которые тоже зависят от нейронной языковой модели на языке перевода.

7.7.2.1. Системы “вопрос — ответ”

Естественным применением обучения “последовательность в последовательность” являются системы “вопрос — ответ” (question-answering — QA). Эти системы часто проектируются для использования различных типов тренировочных данных. В частности, распространены два типа систем “вопрос — ответ”.

1. В системах первого типа ответы непосредственно выводятся на основании фраз и ключевых слов, содержащихся в вопросе.
2. В системах второго типа вопрос сначала преобразуется в запрос базы данных, с помощью которого затем опрашивается структурированная база знаний.

Обучение “последовательность в последовательность” может применяться в обеих ситуациях. Рассмотрим первую ситуацию, когда имеются тренировочные данные, содержащие пары вопросов и ответов наподобие следующих:

Как называется столица Китая? <EOQ> Столица Китая — Пекин. <EOA>

Эти типы тренировочных пар не очень отличаются от тех, которые доступны в случае машинного перевода, и в обоих случаях могут использоваться одни и те же методы. Однако заметьте, что между машинным переводом и системами “вопрос — ответ” существует одно важное различие: машинный перевод требует умозаключений более высокого уровня, чем тот, который обычно нужен для понимания отношений между различными сущностями (например, людьми, местами и организациями). Эта проблема тесно связана с архиважной проблемой *извлечения информации*. Поскольку вопросы часто относятся к именованным сущностям и различным типам отношений между ними, для извлечения информации применяют различные методики. Важность использования сущностей при извлечении информации становится особенно очевидной при поиске ответов на вопросы типа “что/кто/где/когда” (например, сущностно-ориентированный поиск), поскольку *именованные сущности* используются для представления персон, местоположений, организаций, дат и событий, а информацию о взаимодействии между сущностями получают путем *извлечения отношений* между ними. В качестве дополнительной входной информации процесса обучения могут внедряться мета-атрибуты токенов, такие как типы сущностей. Конкретные примеры таких входных элементов были приведены на рис. 7.12, хотя назначение этого рисунка — иллюстрация классификации на уровне токенов.

Важным различием между системами “вопрос — ответ” и системами машинного перевода является то, что последние используют в качестве входа большой корпус документов (например, большую базу данных наподобие Википедии). Процесс разрешения запроса можно считать разновидностью сущностно-ориентированного поиска. С точки зрения глубокого обучения серьезной проблемой QA-систем является то, что они требуют хранилищ знаний намного большей емкости, чем те, которые доступны в типичных рекуррентных нейронных сетях. Архитектурой глубокого обучения, одинаково хорошо работающей в этих ситуациях, являются *сети с памятью* (memory networks) [528]. Системы “вопрос — ответ” предлагают множество примеров предоставления тренировочных данных и способов ответа на вопросы и их оценки. В этом контексте в [527] обсуждается ряд шаблонных задач, которые могут быть полезными для оценки систем “вопрос — ответ”.

Несколько иной подход предполагает преобразование вопросов на естественном языке в запросы, надлежащим образом сформулированные в терминах сущностно-ориентированного поиска. В отличие от систем машинного перевода ответ на вопрос часто рассматривают как многостадийный процесс, в котором понимание предмета вопроса (в терминах подходящим образом сформулированного запроса) иногда вызывает большие затруднения, чем собственно ответ на запрос. В подобных случаях тренировочные пары будут

соответствовать неформальному и формальному представлениям вопросов. Например, может встретиться такая пара:

Как называется столица Китая? <EOQ1>	Столица (Китай, ?) <EOQ2>
<i>Вопрос на естественном языке</i>	<i>Формальное представление</i>

Выражение справа — структурированный вопрос, запрашивающий сущности различного типа, такие как персоны, местоположения и организации. Первым шагом будет преобразование вопроса во внутреннее представление, подобное приведенному выше, которое в большей степени подходит для ответа на запрос. Это преобразование может быть выполнено с использованием тренировочных пар вопросов и их внутренних представлений в связке с рекуррентной сетью. Как только вопрос понят в качестве запроса сущностно-ориентированного поиска, он может быть предъявлен индексированному корпусу, из которого, возможно, родственные соотношения были извлечены заранее. Поэтому в подобных случаях предварительно просматривается база знаний, и разрешение вопроса сводится к поиску соответствий между запросом и извлеченными отношениями. Следует отметить, что такой подход ограничен сложностью синтаксиса, с использованием которого сформулированы вопросы, а ответы могут состоять из одного слова. Поэтому подходы такого типа часто используют для более ограниченных предметных областей. В некоторых случаях прибегают к обучению тому, как можно упростить вопрос путем его перефразирования, прежде чем создавать представление запроса [115, 118]:

Как вы определите, больны ли вы гриппом? <EOQ1>	Каковы признаки гриппа?) <EOQ2>
<i>Сложный вопрос</i>	<i>Перефразированный вопрос</i>

Перефразированному вопросу можно обучить с помощью методики “последовательность в последовательность”, хотя в [118] такой подход, по-видимому, не использовался. Впоследствии это упростит преобразование вопроса в структурированный запрос. Другой вариант заключается в предоставлении вопроса в структурированной форме, с которой следует начать. Пример рекуррентной нейронной сети, которая поддерживает ответы на вопросы из тренировочных пар, описан в [216]. Однако в нем, в отличие от обучения “последовательность в последовательность”, в качестве входного представления используются разреженные деревья зависимостей вопросов. Поэтому часть формального понимания вопроса уже закодирована во входных данных.

7.7.3. Классификация на уровне предложений

В этой задаче каждое предложение трактуется как тренировочный (или тестовый) пример в целях классификации. Обычно классификация на уровне предложений представляет собой более трудную задачу, чем классификация документов, поскольку длина предложений ограничена и в представлении векторного пространства часто содержится недостаточно признаков информации для того, чтобы могла быть выполнена точная классификация. В то же время подход, основанный на последовательностях, обладает большей мощностью и часто может применяться для выполнения более точной классификации. Архитектура RNN для классификации на уровне предложений приведена на рис. 7.11. Обратите внимание на то, что единственным отличием от рис. 7.2, б, является то, что нам больше не нужно заботиться о выходах в каждом узле и мы можем отложить вывод класса до конца последовательности. Иными словами, метка класса предсказывается на самом последнем временном шаге последовательности и используется для обратного распространения ошибок предсказания класса.

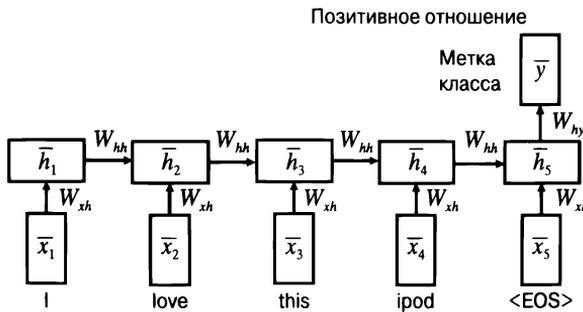


Рис. 7.11. Пример классификации на уровне предложений в сентимент-анализе с двумя классами: “позитивное отношение” и “негативное отношение”

Классификация на уровне предложений часто применяется в сентимент-анализе. В этой задаче делается попытка определить, насколько позитивно или негативно относятся пользователи к конкретным темам, путем анализа содержимого предложения [6]. Например, этот тип классификации можно использовать для определения того, выражает ли данное предложение позитивное отношение, посредством обработки тональности настроения как метки класса. В примере, приведенном на рис. 7.11, предложение отчетливо указывает на позитивное отношение. Заметьте, однако, что нельзя просто использовать векторное пространство представлений, содержащее слово “love” (любовь), чтобы сделать вывод о положительной тональности высказывания. Например, если перед словом “love” встречаются такие слова, как “don’t” (не) or

“hardly” (вряд ли), то тональность настроения сменится с позитивной на негативную. Такие слова называют *переключателями контекстной валентности* (contextual valence shifters) [377], и их влияние может быть смоделировано лишь при условии подходов на основе анализа последовательностей. Рекуррентные нейронные сети могут работать в таких условиях, поскольку используют факты, аккумулированные по конкретной последовательности слов для предсказания метки класса. Такой же подход можно использовать в отношении лингвистических признаков. В следующем разделе будет продемонстрировано, как применять лингвистические признаки для классификации на уровне токенов; аналогичные идеи также применимы в случае классификации на уровне предложений.

7.7.4. Классификация на уровне токенов с использованием лингвистических признаков

Одними из многочисленных применений классификации на уровне токенов являются *извлечение информации* и *сегментирование текста*. При извлечении информации идентифицируются конкретные слова или сочетания слов, которые соответствуют людям, местоположениям и организациям. Лингвистические признаки слова (прописные/строчные буквы, часть речи, орфография) в этих приложениях играют более важную роль, чем в типичных приложениях языкового моделирования или машинного перевода. Тем не менее обсуждаемые в данном разделе методы внедрения лингвистических признаков могут быть использованы в любой из задач, которые обсуждались в предыдущих разделах. Для конкретности рассмотрим *распознавание именованных сущностей* (named-entity recognition), когда каждая сущность должна быть классифицирована как одна из категорий, соответствующих персоне (Person, P), местоположению (Location, L) или другому классу (Other, O). В подобных случаях каждый токен в тренировочных данных снабжен одной из этих меток. В качестве примера можно привести следующее тренировочное предложение:

$$\underbrace{\text{William}}_P \underbrace{\text{Jefferson}}_P \underbrace{\text{Clinton}}_P \underbrace{\text{lives}}_O \underbrace{\text{in}}_O \underbrace{\text{New}}_L \underbrace{\text{York}}_L$$

На практике разметка часто оказывается более сложной, поскольку кодирует информацию о начале и конце смежных токенов с одной и той же меткой. Для тестовых примеров информация о метках токенов недоступна.

Рекуррентную нейронную сеть можно определить аналогично тому, как это делается в приложениях языкового моделирования, за исключением того, что выходы определяются тегами разметки, а не следующим набором слов. Входом на каждом временном шаге t служит токен в представлении прямого кодирова-

ния \bar{x}_t , а выходом \bar{y}_t — тег разметки. Кроме того, мы имеем дополнительный набор q -мерных лингвистических признаков \bar{f}_t , связанных с токенами на временном шаге t . Эти лингвистические признаки могут кодировать информацию о прописных/строчных буквах, орфографии и т.п. Поэтому скрытый слой получает два отдельных входа: от токенов и от лингвистических признаков. Соответствующая архитектура приведена на рис. 7.12.

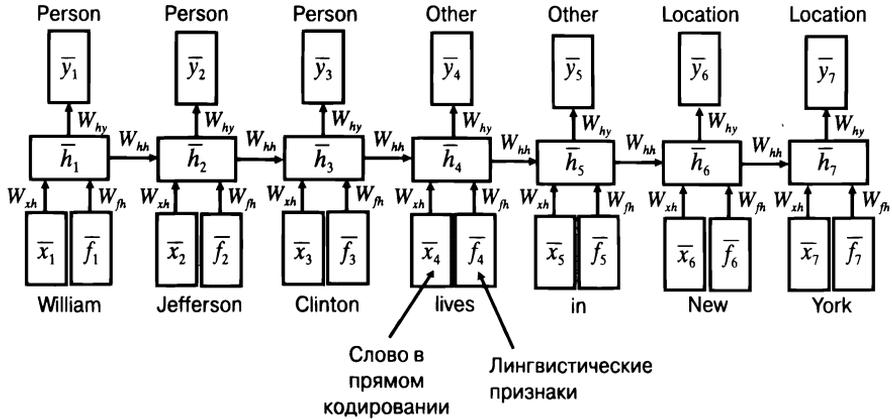


Рис. 7.12. Классификация на уровне токенов по лингвистическим признакам

Здесь мы имеем дополнительную матрицу W_{fh} размера $p \times q$, которая транслирует признаки \bar{f}_t на скрытый слой. Тогда условие рекурсии на временном шаге t может быть записано в следующем виде:

$$\bar{h}_t = \tanh(W_{xh}\bar{x}_t + W_{fh}\bar{f}_t + W_{hh}\bar{h}_{t-1}),$$

$$\bar{y}_t = W_{hy}\bar{h}_t.$$

Здесь основным элементом новизны является использование дополнительной матрицы весов для лингвистических признаков. Изменение типа выходного тега не оказывает значительного влияния на модель. В некоторых вариациях данного метода вместо сложения лингвистических признаков и признаков токенов может оказаться полезным их *конкатенация* в отдельный слой *распределенного представления*, или *вложения* (embedding). В [565] соответствующий пример приведен для случая рекомендательных систем, однако тот же принцип применим и в данном случае. Процесс обучения в целом также не испытывает заметных изменений. В случае классификации на уровне токенов иногда полезно использовать двунаправленные рекуррентные сети, в которых рекурсия осуществляется в обоих временных направлениях [434].

7.7.5. Прогнозирование и предсказание временных рядов

Рекуррентные нейронные сети являются вполне естественным вариантом выбора для *прогнозирования и предсказания* временных рядов. Основное отличие от обработки текста заключается в том, что входные элементы представляют собой векторы с вещественными значениями, а не векторы в представлении прямого кодирования с дискретными значениями. Для предсказания вещественных значений выходной слой всегда использует линейные активации, а не функцию *Softmax*. В тех случаях, когда выходом являются дискретные значения (например, идентификатор конкретного события), также возможно использование дискретных выходов с *Softmax*-активацией. Несмотря на принципиальную возможность применения любого варианта рекуррентной нейронной сети (например, LSTM или GRU), одной из распространенных проблем анализа временных рядов является то, что такие последовательности могут быть чрезвычайно длинными. И хотя LSTM и GRU обеспечивают определенный уровень защиты с увеличением длины временного ряда, существуют ограничения, связанные с производительностью. Это обусловлено деградацией LSTM и GRU в случае рядов, длина которых превышает определенный предел. Многие временные ряды имеют очень большое количество временных меток с различными типами кратко- и долгосрочных зависимостей. В таких случаях задачи прогнозирования представляют особые трудности.

В то же время, по крайней мере в тех случаях, когда количество временных рядов не очень велико, существует несколько полезных решений. Наиболее эффективное из них — использование эхо-сети (см. раздел 7.4), что позволяет эффективно прогнозировать и предсказывать наблюдения как с вещественными, так и дискретными значениями в случае *небольшого* количества временных рядов. Оговорка о небольшом количестве входов очень важна, поскольку работа эхо-сетей основана на рандомизированном расширении пространства признаков посредством скрытых элементов (см. раздел 7.4). Если количество исходных временных рядов очень велико, то расширение размерности скрытого пространства в степени, достаточной для конструирования признаков такого типа, может оказаться практически неосуществимым. Следует отметить, что преобладающее большинство моделей, описанных в литературе, посвященной временным рядам, фактически представляет собой модели с одной переменной. В качестве классического примера можно привести *авторегрессионную модель* (autoregressive model — AR), в которой для прогнозирования применяется непосредственное окно предыстории.

Эхо-сеть может использоваться для выполнения авторегрессии и прогнозирования временных рядов самым непосредственным образом. На каждом временном шаге входом является вектор, который включает d значений,

соответствующих d различным моделируемым временным рядам. Предполагается, что d временных рядов синхронизированы, что часто достигается за счет их предварительной обработки и интерполяции. На каждом временном шаге выходом является предсказанное значение. В случае прогнозирования предсказанным значением является просто значение (значения) различных временных рядов с опережением в k элементов. Такой подход можно рассматривать как аналогию языковых моделей с дискретными последовательностями на языке временных рядов. Также возможен выбор выхода, который соответствует временному ряду, отсутствующему в данных (например, предсказание одной биржевой цены на основании другой), или выбор выхода, соответствующего дискретному событию (например, сбою оборудования). Основное различие между всеми этими случаями заключается в специфическом выборе функции потерь для конкретного типа выхода. Можно показать, что в частном случае прогнозирования временных рядов между авторегрессионными моделями и эхо-сетями существует отчетливая взаимосвязь.

Связь с авторегрессионными моделями

Авторегрессионная модель моделирует значения временных рядов как линейную функцию своей непосредственной предыстории длины p . Коэффициенты этой модели в количестве p обучаются с помощью линейной регрессии. Можно показать, что эхо-сети тесно связаны с авторегрессионными моделями, в которых элементы матрицы весов, соответствующие соединениям “скрытый — скрытый”, семплируются особым способом. Дополнительные возможности эхо-сети по сравнению с авторегрессионными моделями проистекают из нелинейности, существующей между скрытыми слоями. Чтобы понять эту точку зрения, рассмотрим частный случай эхо-сети, в которой входы соответствуют одиночному временному ряду, а в соединениях между скрытыми словами используются линейные активации. Теперь представьте, что у нас имеется какая-то возможность выбирать связи между скрытыми слоями таким образом, чтобы значения скрытого состояния на каждом временном шаге в точности совпадали со значениями временных рядов на последних p отметках времени. Какой тип матрицы семплированных весов способен обеспечить достижение данной цели?

Во-первых, скрытое состояние должно иметь p элементов, поэтому матрица W_{hh} имеет размер $p \times p$. Нетрудно показать, что в результате воздействия матрицы весов W_{hh} , которая смещает скрытое состояние на один элемент и копирует входное значение в вакантное состояние, освободившееся вследствие сдвига, образуется скрытое состояние, в точности совпадающее с состоянием последнего окна из p точек. Другими словами, матрица W_{hh} будет иметь ровно $(p - 1)$ ненулевых элементов вида $(i, i + 1)$ для каждого $i \in \{1 \dots p - 1\}$. В результате ум-

ножение матрицы W_{hh} на любой p -мерный вектор-столбец \bar{h}_t сдвинет элементы \bar{h}_t на один нейронный элемент. В случае одномерного временного ряда элемент x_t представляет собой 1-мерный вход в t -е скрытое состояние эхо-сети, и поэтому W_{xh} имеет размер $p \times 1$. Результатом установки элемента $(p, 0)$ матрицы W_{xh} в 1, а всех остальных элементов в 0 будет копирование x_t в первый элемент \bar{h}_t . Матрица W_{hy} представляет собой матрицу обученных весов размера $1 \times p$, так что результатом умножения $W_{hy} \bar{h}_t$ будет предсказание \hat{y}_t для наблюдаемого значения y_t . В авторегрессионном моделировании значение y_t просто устанавливается в x_{t+k} для некоторого опережающего значения k , а значение k часто устанавливается равным 1. Следует отметить, что матрицы W_{hh} и W_{xh} фиксированы, и в обучении нуждается только матрица W_{hy} . Этот процесс приводит к разработке модели, идентичной авторегрессионной модели временных рядов [3].

Основным отличием авторегрессионной модели временных рядов от эхо-сети является то, что последняя фиксирует значения матриц W_{hh} и W_{xh} случайным образом и использует скрытые состояния намного большей размерности. Кроме того, в скрытых элементах используются нелинейные активации. При условии, что спектральный радиус W_{hh} (незначительно) меньше 1, случайный выбор матриц W_{hh} и W_{xh} с линейными активациями можно рассматривать как вариант авторегрессионной модели с затуханием. Это объясняется тем, что матрица W_{hh} выполняет лишь случайное (но с незначительным затуханием) преобразование предыдущего скрытого состояния. Интуитивно понятно, что использование затухающей случайной проекции предыдущего скрытого состояния позволяет достигать тех же целей, что и скользящее окно смещенной копии предыдущего состояния. Степень затухания регулируется точным спектральным радиусом матрицы W_{hh} . При достаточно большом количестве скрытых состояний матрица W_{hy} предоставляет достаточно степеней свободы для того, чтобы моделировать любую затухающую функцию недавней предыстории. Кроме того, надлежащее масштабирование W_{xh} гарантирует, что самый последний элемент не получает слишком большой или слишком малый вес. Отметим, что эхо-сети тестируют различные коэффициенты масштабирования матрицы W_{xh} для гарантии того, что эффект этого входа не вытеснит вклады скрытых состояний. Нелинейные активации в эхо-сети наделяют этот подход большей мощностью по сравнению с авторегрессионной моделью временных рядов. В определенном смысле эхо-сеть, в отличие от авторегрессионной модели в том виде, как она есть, способна моделировать сложную нелинейную динамику временных рядов.

7.7.6. Временные рекомендательные системы

В последние годы было предложено несколько решений для временного моделирования рекомендательных систем [465, 534, 565]. В некоторых из этих

методов используются временные аспекты пользователей и объектов. Следует подчеркнуть тот факт, что свойства объектов в большей степени склонны к фиксации значений во времени, чем свойства пользователей. Поэтому решений, использующих лишь временное моделирование на уровне пользователей, часто оказывается недостаточно. Однако некоторые методы [534] выполняют временное моделирование как на пользовательском, так и на объектном уровне.

Ниже мы обсудим упрощение модели, рассмотренной в [465]. Во временных рекомендательных системах в процессе выработки рекомендаций используются метки времени, ассоциированные с пользовательскими рейтинговыми оценками. Пусть r_{ijt} — наблюдаемая рейтинговая оценка, данная пользователем i объекту j в момент времени t . Для простоты предположим, что отметка времени t — это просто индекс оценки, присваиваемый в порядке очередности ее получения (хотя многие модели используют фактическое время). Поэтому последовательность, моделируемая RNN, представляет собой последовательность значений, ассоциируемых с базирующимися на содержании представлениями пользователей и объектов, к которым относятся рейтинговые оценки. Таким образом, мы хотим моделировать значение рейтинга как функцию входов, базирующихся на содержании, для каждой отметки времени.

Эти представления, базирующиеся на содержании, описаны ниже. Предполагается, что рейтинговая оценка r_{ijt} зависит от 1) статических признаков, ассоциированных с объектом, 2) статических признаков, ассоциированных с пользователем, и 3) динамических признаков, ассоциированных с пользователем. Статическими элементами, ассоциированными с объектом, могут быть названия или описания, но также возможно создание представления объекта в виде “мешка слов”. Статическими признаками, ассоциированными с пользователем, могут быть пользовательский профиль или фиксированная история посещения страницы, которая не изменяется в пределах набора данных. Статические признаки, ассоциированные с пользователями, в типичных случаях представляются в виде “мешка слов”, и существует даже возможность рассматривать пары “объект — рейтинг” как псевдоключевые слова, чтобы комбинировать указанные пользователем ключевые слова с активностью в присвоении рейтинговых оценок (рейтинговой активностью). В тех случаях, когда используется рейтинговая активность, для создания признаков всегда используется фиксированная история попыток доступа к объекту со стороны пользователя. Динамические пользовательские признаки представляют большой интерес, поскольку они базируются на динамически изменяющейся истории пользовательских попыток доступа. В этом случае в качестве псевдоключевых слов может использоваться краткая история пар “объект — рейтинг”, а для отметки времени t может быть создано представление в виде “мешка слов”.

В некоторых случаях явные рейтинговые оценки отсутствуют, однако доступны данные неявной обратной связи, которые соответствуют щелчкам, выполненным пользователем на ссылках. В случае неявной обратной связи необходимо применять отрицательное семплирование, когда пары “пользователь — объект”, для которых активность не наблюдалась, включаются в последовательность случайным образом. Такой подход можно рассматривать как некий гибридный вариант между подходом, основанным на содержании, и коллаборативными рекомендательными системами. Наряду с использованием в нем триплетов “пользователь — объект — рейтинг”, как в традиционной рекомендательной модели, входами для каждой отметки времени служат представления пользователей и объектов на основе содержания. Однако входы для различных отметок времени соответствуют различным парам “пользователь — объект”, благодаря чему одновременно применяются коллаборативные возможности образцов рейтинговых оценок, соответствующих различным пользователям и объектам.

Общая архитектура этой рекомендательной системы представлена на рис. 7.13. Она содержит три различных подсети, предназначенных соответственно для создания распределенных представлений признаков (вложений) на основе статических признаков объектов статических признаков пользователей и динамических признаков пользователей. Из них первые две — сети прямого распространения, тогда как последняя — рекуррентная нейронная сеть. На первой стадии распределенные представления из первых двух подсетей сливаются в одно с использованием либо конкатенации, либо поэлементного умножения. В последнем случае для статических и динамических признаков пользователей необходимо создавать распределенные представления одинаковой размерности. Затем это объединенное представление пользовательских признаков в момент времени t и статическое представление признаков объекта используются для предсказания рейтинговой оценки для отметки времени t . В случае использования данных неявной обратной связи можно предсказывать вероятности позитивной активности для отдельных пар “пользователь — объект”. Выбор функции потерь зависит от природы предсказываемой рейтинговой оценки. Обучающий алгоритм должен поочередно работать с последовательностями тренировочных триплетов (характеризующимися некоторым фиксированным размером мини-пакета) и распространять ошибку в обратном направлении одновременно в статическую и динамическую части сети.

Вышеупомянутое представление упростило некоторые аспекты процедуры тренировки, описанные в [465]. Например, предполагается, что в каждый момент времени получается только одна рейтинговая оценка и что фиксированного временного горизонта будет вполне достаточно для временного моделирования. В действительности различные условия могут требовать использования

различных уровней гранулярности при обработке временных аспектов. В связи с этим в [465] предложены методы, позволяющие справляться с изменяющимися уровнями гранулярности в процессе моделирования. Также возможна выработка рекомендаций исключительно в режиме коллаборативной фильтрации без какого-либо использования признаков, ориентированных на содержание. Например, рекомендательную систему, которая обсуждалась в разделе 2.5.7, можно⁹ адаптировать для этих целей, используя рекуррентную нейронную сеть (см. упражнение 3).



Рис. 7.13. Рекомендательная система на основе рекуррентной нейронной сети. Для каждой отметки времени входом служат статические/динамические признаки пользователей и статические признаки объектов, а рейтинговой оценкой является выход, соответствующий данной комбинации “пользователь — объект”

В другой недавней работе [565] эта задача трактуется как обработка триплетов “продукт — действие — время” на сайте электронной торговли. Суть идеи состоит в том, что сайт регистрирует последовательные действия, выполняемые каждым пользователем по отношению к различным продуктам, такие как переход на страницу продукта с главной страницы и фактическая покупка

⁹ Несмотря на то что адаптация системы из раздела 2.5.7 представляется наиболее естественной и очевидной, нигде в литературе она не встречалась. Поэтому читателю будет интересно реализовать такую адаптацию, что предлагается в упражнении 3.

продукта. Каждое действие характеризуется периодом длительности, указывающим продолжительность промежутка времени, затраченного пользователем на выполнение данного действия. Период длительности дискретизируется в набор интервалов, характеризующихся равномерной или геометрической шкалой, в зависимости от конкретного приложения. Имеет смысл дискретизировать время в интервалы, увеличивающиеся в геометрической прогрессии.

Для каждого пользователя формируется по одной последовательности в соответствии с выполняемыми им действиями. Обозначим r -й элемент такой последовательности как $(\bar{p}_r, \bar{a}_r, \bar{t}_r)$, где \bar{p}_r — продукт, \bar{a}_r — действие, а \bar{t}_r — дискретизированное значение интервала времени. Каждая из величин \bar{p}_r , \bar{a}_r и \bar{t}_r является вектором в представлении прямого кодирования. Для создания представления $\bar{e}_r = (W_p \bar{p}_r, W_a \bar{a}_r, W_t \bar{t}_r)$ использовался слой вложений с матрицами весов W_p , W_a и W_t . Эти матрицы были предварительно обучены с помощью инструмента *word2vec*, примененного к последовательностям, которые были извлечены на сайте электронной торговли. Впоследствии входом для рекуррентной нейронной сети служили элементы $\bar{e}_1 \dots \bar{e}_T$, которые использовались для предсказания выходов $\bar{o}_1 \dots \bar{o}_T$. Выход на момент времени t соответствует следующему действию пользователя в тот же момент времени. Обратите внимание на то, что слой вложения также подключается к рекуррентной сети и подвергается тонкой настройке в процессе обратного распространения ошибки (помимо его инициализации средствами *word2vec*). В оригинальной работе [565] также добавляется слой ослабления, хотя неплохие результаты удается получить даже без использования этого слоя.

7.7.7. Предсказание вторичной структуры белка

При предсказании структуры белка элементами последовательности являются символы, представляющие одну из 20 аминокислот. Эти 20 возможных аминокислот сродни словарю, используемому при распознавании текста. Поэтому прямое кодирование входа эффективно работает в подобной ситуации. Каждая позиция ассоциируется с меткой класса, соответствующего вторичной структуре белка. Этой вторичной структурой может быть альфа-спираль, бета-лист или полипролиновая спираль. В итоге задача сводится к классификации на уровне токенов. В выходном слое применяется классификация на три класса с использованием функции *Softmax*. В [20] для предсказания используется рекуррентная нейронная сеть. Это объясняется тем, что предсказание структуры белка — задача, в которой можно эффективно задействовать контекст по обе стороны от заданной позиции. В общем случае выбор между одно- и двусторонней сетями в значительной степени определяется тем, имеет ли предсказание каузальную связь с сегментом предыстории и зависит ли оно от двухстороннего контекста.

7.7.8. Сквозное распознавание речи

При *сквозном распознавании речи* (end-to-end speech recognition) делается попытка транскрибировать, насколько это возможно, необработанные аудио-файлы в последовательности символов через несколько промежуточных стадий. Чтобы сделать возможным представление данных в виде входной последовательности, все еще требуется предварительная обработка. Например, в [157] данные представляются в виде *спектрограмм*, полученных из необработанных аудиофайлов с использованием функции *specgram* из библиотеки *matplotlib*, входящей в состав инструментария Python. При этом использовалась ширина окна, равная 254 Фурье-окнам, с перекрыванием 127 кадров и 128 входами на один кадр. Выходом является символ в транскрипционной последовательности, которым может быть буквенный символ, знак пунктуации, пробельный символ и даже нуль-символ. Метки могут быть разными в зависимости от специфики приложения. Например, метками могут служить символы, фонемы или музыкальные ноты. Этим условиям в наибольшей степени отвечает двунаправленная рекуррентная нейронная сеть, поскольку использование контекста по обе стороны от символа способствует повышению точности.

В условиях такого типа одной из трудностей является необходимость выравнивания кадров представления аудиоинформации с транскрипционной последовательностью. Осуществить априорное выравнивание этого типа невозможно, и фактически оно является одним из выходов системы. Это порождает проблему циклической зависимости между сегментированием и распознаванием (так называемый *парадокс Сейре*). Данная задача решается за счет использования *коннекционной временной классификации* (connectionist temporal classification). В этом подходе для определения выравнивания, максимизирующего общую вероятность генерирования вероятностных выходов рекуррентной сети, используется сочетание алгоритма динамического программирования [153] с вероятностными Softmax-выходами рекуррентной сети. Для получения более подробной информации по этому вопросу обратитесь к [153, 157].

7.7.9. Распознавание рукописного текста

С распознаванием речи тесно связано распознавание рукописного текста [154, 156]. В этой задаче вход представляет собой последовательность координат (x, y) , описывающих позицию кончика пера для каждой отметки времени. Выход соответствует последовательности символов, написанных пером. Затем эти координаты используются для извлечения дополнительных признаков, таких как касание пером пишущей поверхности, углы между прилегающими сегментами линии, скорость письма и нормализованные значения координат. В [154] в общей сложности извлекаются 25 признаков. Очевидно, что

множество точек, определяемых координатами, формирует символ. Однако трудно заранее сказать, сколько координат потребуется для создания каждого символа, поскольку их количество может значительно меняться в зависимости от стиля письма различных авторов. Как и в случае распознавания речи, значительные трудности представляет определение наиболее подходящей сегментации. Это тот же парадокс Сейре, с которым приходится сталкиваться при распознавании речи.

В случае распознавания свободного рукописного текста результатом является набор штрихов, объединяя которые можно получать символы. Одна из возможностей состоит в том, чтобы заранее идентифицировать штрихи, а затем использовать их для формирования символов. Однако такой подход приводит к неточным результатам, поскольку идентификация границ штриха часто будет ошибочной. Поскольку ошибки, допущенные на различных стадиях этого процесса, суммируются, разбиение процесса на отдельные фазы — обычно не очень хорошая идея. На базовом уровне задача распознавания рукописного текста не слишком отличается от задачи распознавания речи. Единственное, чем они отличаются, так это спецификой способов представления входов и выходов. Как и в случае распознавания речи, здесь также применима коннекционная временная классификация, в которой подход, основанный на динамическом программировании, сочетается с Softmax-выходами рекуррентной нейронной сети. Поэтому максимизация вероятности того, что для конкретной входной последовательности генерируется конкретная выходная последовательность, достигается за счет того, что выравнивание и классификация по меткам выполняются одновременно с алгоритмом динамического программирования. За более подробной информацией отсылаем читателей к [154, 156].

7.8. Резюме

Рекуррентные нейронные сети (RNN) — это класс нейронных сетей, которые применяются для моделирования последовательностей. Они могут быть описаны как сети с временными слоями, в которых веса разделяются различными слоями. Рекуррентные нейронные сети трудно поддаются обучению, поскольку они подвержены проблемам затухающих и взрывных градиентов. С некоторыми из этих проблем можно справиться, используя усовершенствованные методы тренировки, о чем шла речь в главе 3. Однако существуют и более надежные способы тренировки RNN. В качестве конкретного примера, получившего признание, можно привести использование сетей на основе долгой краткосрочной памяти. Во избежание проблем затухающих и взрывных градиентов в сетях этого типа применяется более мягкий процесс обновления скрытых состояний. Рекуррентные нейронные сети и их варианты находят применение

во многих задачах, таких как аннотирование изображений, классификация на уровне токенов, классификация предложений, сентимент-анализ, распознавание речи, машинный перевод и вычислительная биология.

7.9. Библиографическая справка

Одной из самых ранних форм рекуррентной нейронной сети была сеть Элмана [111] — предшественница современных нейронных сетей. Оригинальная версия алгоритма обратного распространения ошибки во времени была предложена Вербосом [526]. Еще один ранний алгоритм обратного распространения в рекуррентных нейронных сетях был представлен в [375]. Большинство работ по рекуррентным сетям посвящено символическим данным, хотя определенная работа была проведена и в отношении временных рядов с вещественными значениями [80, 101, 559]. Регуляризация RNN обсуждается в [552].

Роль спектрального радиуса матрицы весов связей “скрытый — скрытый” в ослаблении проблем затухающих/взрывных градиентов описана в [220]. Подробное обсуждение проблемы взрывных градиентов и других проблем, связанных с RNN, содержится в [368, 369]. Рекуррентные нейронные сети (и их усовершенствованные варианты) стали более популярными после 2010 года в связи с появлением более производительного компьютерного оборудования, увеличением объема доступных данных и алгоритмическими ухищрениями, которые сделали эту архитектуру намного более привлекательной. Проблемы затухающих и взрывных градиентов в различных типах глубоких сетей, включая рекуррентные сети, описаны в [140, 205, 368]. Правило отсечения градиента рассматривалось Миколовым в его докторской диссертации [324]. Инициализация рекуррентных сетей, содержащих ReLU, обсуждается в [271].

К числу ранних вариантов RNN относится эхо-сеть [219], которую также называют “жидкой машиной” [304]. Эта парадигма также носит название *резервуарные вычисления* (reservoir computing). Обзор эхо-сетей в контексте принципов резервуарных вычислений приведен в [301]. Использование пакетной нормализации обсуждается в [214]. Методы *воздействия на учителя* (teacher forcing methods) описаны в [105]. Стратегии инициализации, снижающие остроту проблем затухающих и взрывных градиентов, рассматриваются в [140].

Модель LSTM была впервые предложена в [204], а ее использование в языковом моделировании обсуждается в [476]. Проблемы, связанные с тренировкой рекуррентных нейронных сетей, затронуты в [205, 368, 369]. Также было показано [326], что с некоторыми из проблем, связанных с затухающими и взрывными аргументами, можно справиться, налагая ограничения на матрицу весов связей “скрытый — скрытый”. В частности, для того чтобы обеспечить медленное обновление некоторых скрытых переменных аналогично медленному

обновлению памяти в LSTM, соответствующий блок этой матрицы ограничивается близостью к тождественной матрице. Вариации рекуррентных нейронных сетей и LSTM, предназначенные для языкового моделирования, обсуждаются в [69, 71, 151, 152, 314, 328]. Двухнаправленные нейронные сети предложены в [434]. В частности, описание LSTM в этой главе основано на результатах, приведенных в [151], а альтернатива управляемому рекуррентному блоку (GRU) представлена в [69, 71]. Суть рекуррентных нейронных сетей разъясняется в [233]. О дополнительном применении RNN для обработки последовательностей и естественных языков рассказывается в [143, 298]. LSTM-сети также используются для маркирования последовательностей [150], что может быть полезным при выполнении сентимент-анализа [578]. Использование комбинации CNN и RNN для аннотирования изображений описано в [225, 509]. Методы обучения “последовательность в последовательность”, предназначенные для машинного перевода, обсуждаются в [69, 231, 480]. Применение двухнаправленных нейронных сетей и LSTM-сетей для предсказания структуры белков, распознавания рукописного текста, машинного перевода и распознавания речи рассматривается в [20, 154, 155, 157, 378, 477]. В последние годы нейронные сети использовались также для временной коллаборативной фильтрации, впервые представленной в [258]. Многочисленные методы временной коллаборативной фильтрации обсуждаются в [465, 534, 560]. Генеративная модель для диалоговых систем с использованием рекуррентных сетей описана в [439, 440]. Использование рекуррентных нейронных сетей для распознавания действий обсуждается в работе [504].

Рекуррентные нейронные сети также были обобщены на рекурсивные нейронные сети, предназначенные для моделирования произвольных структурных отношений, существующих среди данных [379]. Эти методы обобщают рекуррентные нейронные сети на деревья (а не на последовательности) путем рассмотрения древовидного вычислительного графа. Их использование для обнаружения зависящих от задачи представлений описано в [144]. Указанные методы могут применяться в тех случаях, когда структуры данных рассматриваются в качестве входных данных нейронной сети [121]. Рекуррентные нейронные сети являются частным случаем рекурсивных нейронных сетей, в которых структура соответствует линейным цепочкам зависимостей. Использование рекурсивных нейронных сетей в различных приложениях, предназначенных для обработки естественного языка, обсуждается в [459, 460, 461].

7.9.1. Программные ресурсы

Рекуррентные нейронные сети и их варианты поддерживаются многими фреймворками, такими как *Caffe* [571], *Torch* [572], *Theano* [573] и *TensorFlow* [574]. Некоторые другие фреймворки наподобие *DeepLearning4j* предоставляют

реализации LSTM [617]. Реализации сентимент-анализа с помощью LSTM-сетей доступны в [578]. Этот подход основан на технике маркирования последовательностей, описанной в [152]. Читателю будет особенно поучительно ознакомиться с приведенным в [580] исходным кодом RNN, работающей на уровне символов. Концептуальное описание этого кода приведено в [233, 618].

7.10. Упражнения

1. Загрузите приведенную в [580] RNN, работающую на уровне символов, и обучите ее на наборе данных “*tiny Shakespeare*”, доступном по тому же адресу. Создайте выходы языковой модели после выполнения тренировки в течение 1) 5 эпох, 2) 50 эпох и 3) 500 эпох. Какие существенные различия наблюдаются между этими тремя выходами?
2. Рассмотрим эхо-сеть, в которой скрытые состояния разделены на K групп по p/K элементов каждая. Скрытым состояниям отдельной группы разрешается образовывать связи лишь в пределах собственной группы в следующий момент времени. Обсудите, как этот подход связан с ансамблевым методом, в котором создаются K независимых эхо-сетей и предсказания этих сетей усредняются.
3. Покажите, как можно видоизменить архитектуру сети прямого распространения, которая обсуждалась в разделе 2.5.7, для создания рекуррентной нейронной сети, способной работать во временных рекомендательных системах. Реализуйте этот адаптированный вариант и сравните его производительность с производительностью архитектуры прямого распространения на наборе данных конкурса Netflix.
4. Рассмотрим рекуррентную сеть, в которой скрытые состояния имеют размерность 2. Каждый элемент матрицы W_{hh} размером 2×2 , которая осуществляет преобразование между скрытыми состояниями, равен 3,5. Кроме того, между скрытыми состояниями различных временных слоев используется сигмоидная активация. Будет ли такая сеть подвержена проблемам затухающих и взрывных градиентов?
5. Предположим, вы располагаете большой базой биологических данных с информацией об азотистых основаниях, которая содержится в виде строк кодовых последовательностей, состоящих из символов $\{A, C, T, G\}$. (A — аденин, C — цитозин, T — тимин, G — гуанин.) Некоторые из этих строк содержат необычные мутации, представляющие изменения в азотистых основаниях. Предложите метод обучения без учителя (т.е. нейронную архитектуру), использующий RNN для обнаружения таких мутаций.

6. Как бы вы изменили архитектуру из предыдущего вопроса, если бы вам была предоставлена тренировочная база данных, в которой позиции мутаций в каждой последовательности помечены, но в тестовой базе данных оставлены непомеченными?
7. Предложите рекомендации относительно предварительного обучения входного и выходного слоев в подходе для машинного перевода с применением обучения “последовательность в последовательность”.
8. Рассмотрим социальную сеть с большим объемом сообщений, которыми обмениваются пары “отправитель — получатель”. Но нас интересуют лишь сообщения, содержащие идентификаторы ключевых слов, так называемые *хештеги*. Создайте модель, работающую в режиме реального времени, используя RNN, способную рекомендовать хештеги для каждого пользователя, а также для его потенциальных подписчиков, которые могут быть заинтересованы в сообщениях, имеющих отношение к данному хештегу. Исходите из предположения, что имеющихся в вашем распоряжении вычислительных ресурсов достаточно для реализации инкрементной тренировки RNN.
9. Изменяются ли обученные веса после масштабирования тренировочного набора данных с использованием некоторого коэффициента масштабирования в случае пакетной или послонной нормализации? Каков был бы ваш ответ, если бы масштабированию был подвергнуто лишь небольшое подмножество точек тренировочных данных? Повлияет ли на обученные веса в случае любого метода нормализации изменение центрирования набора данных?
10. Рассмотрим большую базу данных, содержащую пары предложений на различных языках. Несмотря на достаточную полноту представления каждого языка, некоторые пары могут не быть представлены в базе данных. Покажите, как можно использовать эти тренировочные данные для 1) создания одного и того же универсального кода для конкретного предложения, независимо от языка, и 2) обеспечения возможности перевода даже между парами языков, не представленными в словаре.

Глава 8

Сверточные нейронные сети

Душа не может мыслить без образов.

Аристотель

8.1. Введение

Сверточные нейронные сети (convolutional neural networks, CNN) предназначены для работы со входами, имеющими сетчатую структуру и характеризующимися сильными пространственными зависимостями в локальных областях сетки. Наиболее показательным примером данных, имеющих такую структуру, является двухмерное изображение. Этот тип данных также демонстрирует пространственные зависимости, поскольку смежные пространственные участки изображения часто имеют одинаковые цветовые значения отдельных пикселей. Дополнительное измерение передает различные цвета, что создает трехмерный входной объем. Поэтому в сверточной нейронной сети между признаками существует взаимосвязь, определяемая их пространственным разнесением. Другие формы последовательных данных, такие как текст, временные ряды и последовательности, также могут считаться частными случаями данных, структурированных в виде сетки, с различными типами взаимосвязи между соседними элементами. В большинстве случаев сверточные нейронные сети ориентированы на обработку изображений, но могут также применяться для обработки любых типов временных, пространственных и пространственно-временных данных.

Важной особенностью изображений является то, что они в определенной степени обладают свойством пространственной инвариантности, чего нельзя сказать о многих других типах данных с сетчатой структурой. Например, банан интерпретируется одинаково, независимо от того, находится ли он в верхней или в нижней части изображения. Сверточные нейронные сети стремятся создавать похожие значения признаков из локальных областей, имеющих аналогичные образы. Одним из преимуществ изображений является то, что влияние

отдельных входов на представления признаков часто удается описать на интуитивном уровне. Поэтому в данной главе мы в основном будем работать именно с изображениями, но также кратко обсудим применение сверточных нейронных сетей для обработки данных других типов.

Определяющей характеристикой сверточных нейронных сетей является операция, которую называют *сверткой* (convolution). Это операция поэлементного умножения структурированного в виде сетки набора весов и аналогично структурированных входов, извлеченных из различных локальных пространственных участков входного объема. Операции такого типа полезны в случае данных с высокой степенью пространственной или иной локализации, таких как изображения. Поэтому сверточные сети определяются как сети, использующие операцию свертки по крайней мере в одном слое, хотя в большинстве сверточных сетей эта операция применяется в нескольких слоях.

8.1.1. Исторический обзор и биологические предпосылки

Сверточные нейронные сети стали одним из первых успешных проектов глубокого обучения задолго до того, как последние достижения в методиках обучения позволили улучшить производительность архитектур другого типа. Фактически именно яркие успехи некоторых архитектур CNN на соревнованиях по распознаванию образов, проводившихся после 2011 года, привлекли повышенное внимание к глубокому обучению. В промежуток времени между 2011 и 2015 годами долго державшиеся рекорды, продемонстрированные, например, победителями соревнований ImageNet с коэффициентом ошибок классификации топ-5 более 25% [581], были побиты с результатами, обеспечившими ошибку менее 4%. Сверточные нейронные сети хорошо подходят для процесса *иерархического конструирования признаков* (hierarchical feature engineering) с глубиной. Отражением этого является тот факт, что самые глубокие сети происходят от сверточных нейронных сетей. Кроме того, эти сети также могут служить ярким примером того, как нейронные сети, идеи которой были подсказаны биологией, иногда могут демонстрировать потрясающие результаты. Наилучшие из имеющихся на сегодняшний день сверточных нейронных сетей достигли или даже превысили уровень человеческих возможностей, чего не могли предвидеть большинство специалистов в области машинного зрения всего лишь пару десятилетий тому назад.

Первоначально идеи, лежащие в основе сверточных нейронных сетей, были подсказаны экспериментами Хубеля и Визеля, изучавшими зрительную кору кошек [212]. Зрительная кора содержит небольшие участки с нейронами, чувствительными к определенным областям зрительного поля. Другими словами, при возбуждении определенных областей зрительного поля возбуждаются и

соответствующие нейроны зрительной коры. Кроме того, возбуждение нейронов также зависит от формы и ориентации объектов в визуальном поле. Например, черта, проведенная в вертикальном направлении, вызывает возбуждение одних нейронов, тогда как черта, проведенная в горизонтальном направлении, — других. Нейроны соединяются с использованием слоистой архитектуры, и это открытие привело к гипотезе о том, что млекопитающие используют различные слои для конструирования частей изображений на различных уровнях абстракции. С точки зрения машинного обучения этот принцип аналогичен принципу иерархического выделения признаков. Как будет показано далее, сверточные нейронные сети достигают аналогичных целей, кодируя примитивные формы в ранних слоях, а более сложные формы — в поздних слоях.

Исходя из этих идей, почерпнутых из биологии, была разработана первая нейронная модель: неокогнитрон [127]. Однако между этой моделью и современной сверточной сетью существуют различия. Наиболее заметным из них является то, что в неокогнитроне не использовалось разделение весов. На основании этой архитектуры была разработана одна из первых полносверточных архитектур: LeNet-5 [279]. Данная сеть применялась банками для идентификации рукописных цифр на чеках. С тех пор сверточные нейронные сети значительно эволюционировали. В основном это проявилось в использовании большего количества слоев и устойчивых функций активации наподобие ReLU. Кроме того, разработанные к настоящему времени многочисленные изощренные методики тренировки наряду с увеличением вычислительной мощности доступного оборудования позволяют добиваться заметных успехов в обучении при работе с глубокими сетями и большими наборами данных.

Фактором, сыгравшим важную роль в продвижении сверточных нейронных сетей, послужили ежегодные соревнования ImageNet [582] (полное название — *ImageNet Large Scale Visual Recognition Challenge [ILSVRC]*). На соревнованиях ILSVRC используется набор данных ImageNet [581], который обсуждался в разделе 1.8.2. Начиная с 2012 года сверточные сети становятся неизменными победителями этих соревнований. В действительности превосходство CNN в отношении распознавания образов в наши дни стало настолько общепризнанным фактом, что почти все позиции в последних редакциях этого соревнования были заняты сверточными нейронными сетями. Одним из самых первых методов, которому удалось с большим отрывом опередить соперников на соревнованиях ImageNet в 2012 году, была сеть *AlexNet* [255]. Кроме того, повышение точности за последние несколько лет было настолько разительным, что полностью изменило ландшафт исследований в этой области. Несмотря на то что заметный прирост производительности произошел в 2012–2015 гг., архитектурные различия между недавними победителями соревнований и некоторыми ранними сверточными нейронными сетями довольно незначительны,

по крайней мере, на концептуальном уровне. Тем не менее небольшие детали, по-видимому, играют существенную роль при работе почти со всеми типами нейронных сетей.

8.1.2. Общие замечания относительно сверточных нейронных сетей

Секрет успеха любой нейронной архитектуры кроется в адаптации структуры сети к семантическому смыслу данных конкретной предметной области. Сверточные нейронные сети в значительной степени базируются на этом принципе, поскольку используют разреженные соединения с высокой степенью разделения параметров с учетом специфики области. Другими словами, не все состояния в каком-либо слое безоговорочно связываются с состояниями в предыдущем слое. Вместо этого значение признака в отдельном слое связывается лишь с локальной пространственной областью в предыдущем слое с помощью согласованного набора параметров по всему изображению. Такой тип архитектуры можно рассматривать как зависящую от предметной области регуляризацию, подсказанную биологическими экспериментами Хубеля и Визеля. Вообще говоря, успешность сверточных нейронных сетей послужила важным уроком для исследователей в других областях. Тщательно спроектированная архитектура, в которой отношения и зависимости между элементами данных используются для снижения количества параметров, дает ключ к результатам высокой точности.

Значительный уровень регуляризации, определяемой спецификой области, также доступен и в рекуррентных нейронных сетях, разделяющих параметры между различными периодами времени. Это разделение основано на том предположении, что временные зависимости остаются инвариантными во времени. Рекуррентные нейронные сети базируются на интуитивном понимании временных отношений, тогда как сверточные сети — на понимании пространственных отношений. В последнем случае интуитивное понимание пришло в результате исследования организации биологических нейронов в зрительной коре кошек. Этот выдающийся успех служит значительным стимулом к использованию результатов нейронаук для проектирования нейронных сетей продуманными способами. Даже признавая тот факт, что искусственные нейронные сети являются лишь жалким подобием биологического мозга с его чрезвычайно сложным строением, не стоит недооценивать важность обретения интуитивного понимания соответствующих процессов, которое может быть получено путем изучения базовых принципов нейронаук [176].

Структура главы

В следующем разделе будут изложены основы сверточных нейронных сетей с рассмотрением выполняемых в них операций, а также способы их организации. Процесс обучения сверточных сетей обсуждается в разделе 8.3. Примеры некоторых типичных сверточных нейронных сетей, завоевавших призы на недавних соревнованиях, приведены в разделе 8.4. Сверточный автокодирующий обсуждается в разделе 8.5. Ряд применений сверточных сетей рассмотрен в разделе 8.6. Резюме главы приведено в разделе 8.7.

8.2. Базовая структура сверточной сети

В CNN состояния в каждом слое выстраиваются в соответствии с сетчатой пространственной структурой. Эти пространственные отношения наследуются от одного слоя к другому, поскольку значение каждого признака базируется на небольшой пространственной области предыдущего слоя. Очень важно поддерживать эти пространственные отношения между ячейками сетки, поскольку от них критически зависят операция свертки и преобразование к следующему слою. Каждый слой в CNN представляет собой трехмерную сетчатую структуру, характеризующуюся *высотой*, *шириной* и *глубиной*. Понятие глубины слоя в сверточной сети не следует путать с понятием глубины самой сети. Термин “глубина” (когда он употребляется в контексте одиночного слоя) относится к количеству *каналов* в каждом слое, например к количеству первичных цветowych каналов (синий, зеленый и красный) во входном изображении или количеству карт признаков в скрытых слоях. Применение термина “глубина” одновременно в отношении количества карт признаков в каждом слое и количества слоев является примером неудачно перегруженной терминологии, которая используется в литературе по сверточным сетям, но мы будем тщательно следить за употреблением этого термина, чтобы его смысл был понятен из контекста.

Функции сверточной нейронной сети во многом подобны функциям традиционной сети прямого распространения, за исключением того, что операции в ее слоях пространственно организованы с использованием разреженных (и тщательно спроектированных) соединений между слоями. Как правило, в CNN представлены три слоя: *свертки*, *пулинга* и *ReLU*. Активация с помощью ReLU ничем не отличается от аналогичной активации в традиционной нейронной сети. Последний набор слоев часто является полносвязным и отображается специфическим для приложения способом на набор выходных узлов. В дальнейшем мы опишем каждый из различных типов операций и слоев, а также типичных способов чередования этих слоев в сверточной нейронной сети.

Для чего нужна глубина в каждом слое сверточной сети? Чтобы разобраться в этом, исследуем, как организован вход сверточной сети. Входные данные

CNN организуются в виде 2-мерной сетки, и значения отдельных точек сетки называются *пикселями*. Каждый пиксель соответствует пространственному расположению в пределах изображения. Однако для кодирования точного цвета пикселя нужен многомерный массив значений в каждой точке сетки. В цветовой схеме RGB мы имеем дело с интенсивностью трех основных цветов: красного, зеленого, синего. Поэтому для изображения с пространственными размерами 32×32 пикселя и глубиной 3 (в соответствии с числом цветовых каналов RGB) общее количество пикселей в изображении составит $32 \times 32 \times 3$. Этот размер изображения довольно распространен и, в частности, встречается в наборе данных CIFAR-10, который широко применяется в целях бенчмаркинга [583]. Пример такой организации приведен на рис. 8.1, а. Представление входного слоя в виде 3-мерной структуры вполне естественно, поскольку два измерения отвечают за пространственные отношения, а третье — за независимые свойства, относящиеся к этим каналам. Например, интенсивности основных цветов — независимые свойства первого слоя. В скрытых слоях эти независимые свойства соответствуют различным типам фигур, извлеченных из локальных областей изображения. Для целей нашего обсуждения предположим, что вход в q -м слое имеет размеры $L_q \times B_q \times d_q$, где L_q — высота (или длина), B_q — ширина (или размах), d_q — глубина. Почти во всех приложениях для работы с изображениями значения L_q и B_q совпадают. Однако для общности изложения мы будем различать понятия высоты и ширины.

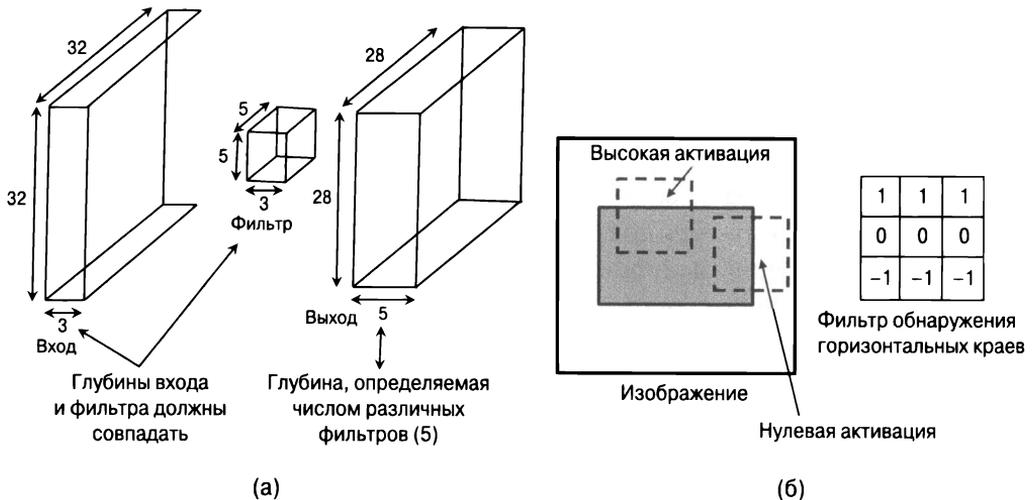


Рис. 8.1. Свертка входного слоя размера $32 \times 32 \times 3$ с фильтром размера $5 \times 5 \times 3$ создаст выходной слой с пространственными размерами 28×28 . Глубина результирующего выхода зависит от количества различных фильтров, а не от размеров входного слоя или фильтра (а). В процессе скольжения фильтра по изображению делается попытка обнаружить конкретный признак в различных окнах изображения (б)

Для первого (входного) слоя эти значения определяются природой входных данных и их предварительной обработки. В приведенном выше примере соответствующими значениями являются $L_1 = 32$, $B_1 = 32$ и $d_1 = 3$. Последующие слои имеют ту же 3-мерную организацию, за исключением того, что каждое из d_q 2-мерных значений сетки для конкретного входа уже не может рассматриваться в качестве сетки необработанных пикселей. Кроме того, значение d_q для скрытого слоя намного превышает значение 3, поскольку количество независимых свойств любой заданной области, имеющих отношение к классификации, может быть довольно значительным. При $q > 1$ такие сетки значений называются *картами признаков* (feature maps) или *картами активации* (activation maps). Эти значения аналогичны значениям в скрытых слоях сети прямого пространства.

В CNN параметры организуются в наборы 3-мерных структурных блоков, известных как *фильтры* (filters) или *ядра* (kernels). В пространственном отношении фильтр, как правило, представляет собой квадрат, размеры которого в типичных случаях намного меньше размеров слоя, в котором этот фильтр применяется. С другой стороны, *глубина фильтра всегда совпадает с глубиной его слоя*. Пусть фильтр в q -м слое имеет размеры $F_q \times F_q \times d_q$. Пример фильтра с размерами $F_1 = 5$ и $d_1 = 3$ приведен на рис. 8.1, а. Для параметра F_q общепринято устанавливать небольшие нечетные значения. Обычно в качестве таковых используются значения 3 и 5, хотя существуют некоторые интересные случаи, в которых вполне возможно взять $F_q = 1$.

Операция свертки помещает фильтр в каждую возможную позицию на изображении (или скрытом слое) так, чтобы он полностью перекрывался с изображением, и вычисляет скалярное произведение $F_q \times F_q \times d_q$ параметров фильтра и параметров соответствующей сетки входного объема (того же размера $F_q \times F_q \times d_q$). При вычислении этого скалярного произведения записи в соответствующей 3-мерной области входного объема и фильтра трактуются как векторы порядка $F_q \times F_q \times d_q$, элементы которых упорядочены в соответствии с их позициями в объемной сетке. Каково количество возможных позиций, в которые может быть помещен фильтр? Это важный вопрос, поскольку каждая такая позиция определяет пространственный “пиксель” (или, точнее, признак) в следующем слое. Иными словами, число возможных расположений фильтра на изображении определяет пространственную высоту и ширину следующего скрытого слоя. Относительные пространственные позиции признаков в следующем слое определяются относительными пространственными позициями левых верхних углов соответствующих пространственных сеток в предыдущем слое. При выполнении операции свертки в q -м слое фильтр можно расставить в $L_{q+1} = (L_q - F_q + 1)$ позициях по высоте и в $B_{q+1} = (B_q - F_q + 1)$ позициях по ширине изображения (так, чтобы часть фильтра не выходила за границы

изображения). Следовательно, полное количество возможных скалярных произведений равно $L_{q+1} \times B_{q+1}$, что определяет размер следующего скрытого слоя. Таким образом, мы получаем для L_2 и B_2 из предыдущего примера следующие значения:

$$\begin{aligned} L_2 &= 32 - 5 + 1 = 28, \\ B_2 &= 32 - 5 + 1 = 28. \end{aligned}$$

Следующий скрытый слой размером 28×28 представлен на рис. 8.1, а. Однако этот скрытый слой имеет глубину $d_2 = 5$. Откуда происходит такая глубина? Она получена в результате применения 5 различных фильтров с их собственными независимыми наборами параметров. Каждый из этих 5 наборов пространственно расположенных признаков, полученных из выходов одного фильтра, называется *картой признаков* (feature map). Очевидно, что увеличение количества карт признаков является результатом применения большего количества фильтров (т.е. большего количества параметров), равного $F_q^2 \cdot d_q \cdot d_{q+1}$ для q -го слоя. *Количество фильтров, используемых в каждом слое, управляет мощностью модели, поскольку от него напрямую зависит количество параметров.* Кроме того, увеличение количества фильтров в конкретном слое увеличивает количество карт признаков (т.е. глубину) следующего слоя. Различные слои могут иметь различное количество карт признаков, в зависимости от количества фильтров, которые мы использовали для операции свертки в предыдущем слое. К примеру, типичный входной слой имеет три цветовых канала, но каждый из последующих слоев может иметь глубину (т.е. количество карт признаков) более 500. Идея заключается в том, что каждый фильтр пытается идентифицировать отдельный тип пространственного образа в небольшой пространственной области изображения, поэтому для захвата как можно более широкого разнообразия возможных форм, комбинируемых для создания окончательного изображения, требуется большее количество фильтров (в отличие от случая входного слоя, в котором достаточно использовать RGB-каналы). Обычно в поздних слоях используется меньшее количество пространственных параметров, но их глубина больше за счет большего количества карт признаков. Например, фильтр, приведенный на рис. 8.1, б, представляет детектор горизонтальных краев на черно-белых изображениях с одним каналом. Как показано на рис. 8.1, б, результирующий признак будет иметь высокую активацию в каждой позиции, где видны горизонтальные линии. Идеальная вертикальная линия будет давать нулевую активацию, тогда как наклонная — промежуточную. Поэтому при скольжении фильтра по всему изображению уже будут обнаружены ключевые контуры изображения в одной карте признаков выходного объема. Для создания выходного объема с более чем одной картой признаков применяется несколько фильтров. Другой фильтр может создавать карту пространственных признаков активаций вертикальных линий.

Теперь мы готовы перейти к формальному определению операции свертки. Представим параметры p -го фильтра в q -м слое в виде 3-мерного тензора $W^{(p, q)} = [w_{ijk}^{(p, q)}]$. Индексы i, j, k задают позиции по высоте, ширине и глубине фильтра. Карты признаков в q -м слое представим в виде 3-мерного тензора $H^{(q)} = [h_{ijk}^{(p, q)}]$. Если $q = 1$, то частный случай, соответствующий обозначению $H^{(1)}$, представляет входной слой (который не скрыт). Тогда операция свертки из q -го слоя в $(q + 1)$ -й слой определяется следующим образом:

$$h_{ijp}^{(q+1)} = \sum_{r=1}^{F_q} \sum_{s=1}^{F_q} \sum_{k=1}^{d_q} w_{rsk}^{(p, q)} h_{i+r-1, j+s-1, k}^{(q)} \quad \begin{aligned} \forall i \in \{1 \dots L_q - F_q + 1\} \\ \forall j \in \{1 \dots B_q - F_q + 1\} \\ \forall p \in \{1 \dots d_q\} \end{aligned}$$

Несмотря на кажущуюся сложность нотации в приведенном выше выражении, лежащая в его основе базовая операция свертки — это просто скалярное произведение по всему объему фильтра, повторенное по всем действительным пространственным позициям (i, j) и фильтрам (проиндексированным с помощью p). Операцию свертки полезно представлять на интуитивном уровне как помещение фильтра в каждую из 28×28 возможных позиций в первом слое (см. рис. 8.1, *a*) с последующим вычислением скалярного произведения вектора, состоящего из $5 \times 5 \times 3 = 75$ значений фильтра, и соответствующих 75 значений, содержащихся в $H^{(1)}$. Несмотря на то что размер входного слоя на рис. 8.1, *a*, равен 32×32 , существует всего $(32 - 5 + 1) \times (32 - 5 + 1)$ возможных способов расположения фильтра размером 5×5 в пределах входного объема размером 32×32 .

Операция свертки заставляет вспомнить об экспериментах Хубеля и Визеля, которые использовали активации в небольших областях визуального поля для активации определенных нейронов. В случае сверточных нейронных сетей визуальное поле определяется фильтром, последовательно применяемым ко всему изображению для обнаружения форм в каждом пространственном расположении. Кроме того, фильтры в ранних слоях обнаруживают более примитивные формы, тогда как фильтры в поздних слоях создают более сложные композиции этих примитивных форм. Это и не удивительно, поскольку большинство глубоких нейронных сетей неплохо проявило себя при иерархическом конструировании признаков.

Одним из свойств свертки является ее *эквивариантность по отношению к переносам* (equivariance to translation). Иными словами, если мы сместим пиксельные значения в любом направлении на одну единицу, а затем применим операцию свертки, то значения соответствующих признаков сместятся вместе со входными значениями. Это обусловлено тем, что параметры фильтра разделяются всей сверткой. Причиной разделения параметров всей сверткой является то, что любая отдельно взятая форма, находящаяся в любой части

изображения, должна обрабатываться одинаковым образом, независимо от ее конкретного пространственного расположения.

Перейдем к более подробному рассмотрению операции свертки. На рис. 8.2 приведен пример входного слоя и фильтра, глубина которого для простоты принята равной 1 (что имеет место в случае изображений в градациях серого с одним цветовым каналом). Обратите внимание на то, что глубина слоя должна в точности совпадать с глубиной его фильтра/ядра, а вклады скалярного произведения по всем картам признаков в соответствующей области сетки конкретного слоя должны суммироваться (в общем случае) для создания одного значения признака в следующем слое. На рис. 8.2 представлены два конкретных примера выполнения операции свертки над слоем с размерами $7 \times 7 \times 1$ и отображенным под ним фильтром размером $3 \times 3 \times 1$. Кроме того, в правой части рис. 8.2 приведена полная карта признаков следующего слоя. На рисунке также представлены выходные значения двух примеров операции свертки, равные 16 и 26 соответственно. Эти значения получены в результате выполнения следующих операций умножения и агрегирования:

$$5 \times 1 + 8 \times 1 + 1 \times 1 + 1 \times 2 = 16,$$

$$4 \times 1 + 4 \times 1 + 4 \times 1 + 7 \times 2 = 26.$$

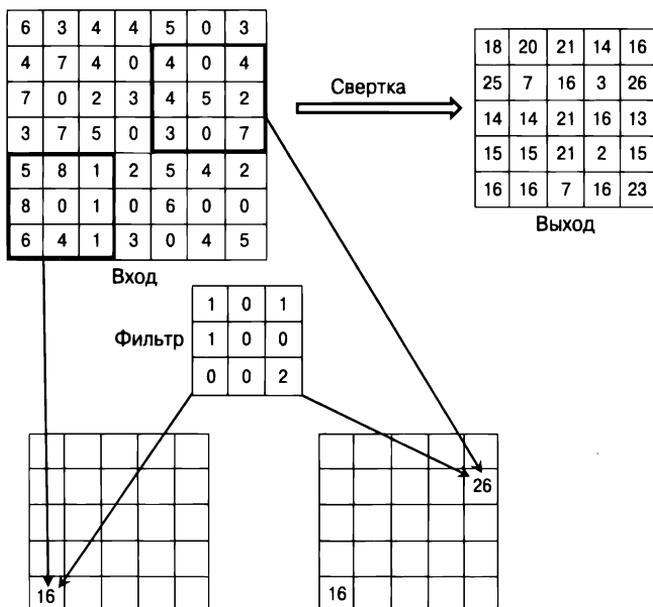


Рис. 8.2. Пример выполнения операции свертки над входом с размерами $7 \times 7 \times 1$ и фильтром с размерами $3 \times 3 \times 1$ и шагом 1. Глубина 1 для фильтра/входа была выбрана из соображений простоты. В случае глубины, превышающей 1, вклады каждой из входных карт признаков суммируются для создания одного значения карты признаков. Один фильтр всегда будет создавать одну карту признаков, независимо от его глубины

В приведенных выше выражениях агрегации умножение на нули опущено. В том случае, если глубина слоя и соответствующего ему фильтра превышает 1, приведенные выше операции выполняются для каждой пространственной карты, а затем агрегируются по всей глубине фильтра.

Свертка в q -м слое увеличивает *рецептивное поле* признака из q -го слоя в $(q + 1)$ -й слой. Иными словами, каждый признак в следующем слое захватывает большую пространственную область входного слоя. Например, в случае последовательного применения фильтра размером 3×3 в трех слоях активации в первом, втором и третьем скрытых слоях захватываются пиксельные области *исходного входного изображения* размером 3×3 , 5×5 и 7×7 соответственно. Как будет показано далее, другие типы операций еще более увеличивают рецептивные поля, поскольку они уменьшают пространственный объем слоев. Это естественное следствие того факта, что признаки в более поздних слоях захватывают сложные характеристики изображения по большим пространственным областям, а затем комбинируют более простые признаки из более ранних слоев.

При выполнении операции свертки из q -го слоя в $(q + 1)$ -й слой глубина d_{q+1} вычисляемого слоя зависит от количества фильтров в q -м слое и не зависит от глубины q -го слоя или любого из других его измерений. Иными словами, глубина d_{q+1} в $(q + 1)$ -м слое всегда равна количеству фильтров в q -м слое. Например, глубина второго слоя на рис. 8.1, *a*, равна 5, поскольку для выполнения преобразования в первом слое используются в общей сложности пять фильтров. Однако, для того чтобы выполнить свертки во втором слое (для создания третьего слоя), теперь, в соответствии с новой глубиной этого слоя, необходимо использовать фильтры глубиной 5, даже если при выполнении сверток в первом слое (с целью создания второго слоя) использовались фильтры глубиной 3.

8.2.1. Дополнение

Обратите внимание на то, что операция свертки снижает размер $(q + 1)$ -го слоя по сравнению с размером q -го слоя. Такой тип снижения размерности данных не всегда желателен, поскольку он приводит к потере определенной части информации, касающейся областей вдоль границ изображения (или границ карты признаков в случае скрытых слоев). Эту проблему можно разрешить, используя так называемое *дополнение* (padding). При этом вдоль всех границ карты признаков добавляются $(F_q - 1)/2$ “пикселей”, которые являются реальными значениями признаков в случае дополнения скрытых слоев. Каждое из этих дополненных значений признаков устанавливается в нуль, независимо от того, идет ли речь о дополнении входного или скрытых слоев. В результате пространственные высота и ширина входного объема увеличатся каждая на $(F_q - 1)$, что в точности совпадает с их уменьшением (в выходном объеме) после выполнения свертки. Дополненные части не вносят вклада в скалярное произведение,

поскольку их значения установлены в нуль. В определенном смысле можно сказать, что дополнение разрешает выполнение операции свертки с использованием фильтра, частично заходящего за границы слоя, а затем вычисляет скалярное произведение только по той части слоя, в которой определены значения. Этот тип дополнения называют *половинным дополнением* (half-padding), поскольку в тех случаях, когда фильтр помещается в свои крайние пространственные позиции вдоль ребер, (почти) половина фильтра заходит за границы со всех сторон пространственного входа.

Если дополнение не используется, то в программах оно задается аргументом “valid”. С точки зрения проведения экспериментов дополнение “valid” обычно работает не очень хорошо. Использование половинного дополнения гарантирует, что определенная часть критической информации на границах слоя будет представлена независимым образом. В случае дополнения “valid” вклады пикселей на границах слоя будут представлены недостаточно полно по сравнению с центральными пикселями в следующем скрытом слое, что нежелательно. Кроме того, эта неполнота представления будет только накапливаться по мере прохождения слоев. Поэтому обычно дополнение используется во всех слоях, а не только в первом, в котором пространственные расположения соответствуют входным значениям. Рассмотрим ситуацию, когда размеры слоя составляют $32 \times 32 \times 3$, а размеры фильтра — $5 \times 5 \times 3$. Таким образом, изображение окружается со всех сторон нулями в количестве $(5 - 1)/2 = 2$. В результате первоначальные пространственные размеры 32×32 сначала увеличиваются до 36×36 вследствие использования дополнения, а затем вновь уменьшаются до размеров 32×32 после выполнения свертки. Пример применения дополнения в одиночной карте признаков приведен на рис. 8.3, где изображение (или карта признаков) дополняется со всех сторон двумя нулями. Это имитирует ситуацию, которую мы только что обсудили, за исключением того, что пространственные размеры изображения в данном случае сделаны намного меньшими, чем 32×32 , с целью экономии места.

Другой полезной формой дополнения является *полное дополнение* (full-padding). В этом случае мы позволяем фильтру (почти) полностью выступать за границы входного изображения со всех сторон. Иными словами, части фильтра размером $F_q - 1$ разрешено выступать с любой стороны входного изображения с перекрыванием лишь одного пространственного признака. Например, ядро входного изображения может перекрываться в одном пикселе в самом углу. Поэтому входное изображение дополняется $(F_q - 1)$ нулями с каждой стороны. Иначе говоря, каждый пространственный размер входного изображения увеличивается на $2(F_q - 1)$. Поэтому, если входные размеры исходного изображения равны L_q и B_q , то дополненные пространственные размеры входного объема составят $L_q + 2(F_q - 1)$ и $B_q + 2(F_q - 1)$. После выполнения свертки размеры карты

признаков в слое $(q + 1)$ составят $L_q + F_q - 1$ и $B_q + F_q - 1$ соответственно. Несмотря на то что свертка обычно уменьшает пространственные размеры, полное дополнение увеличивает их. Интересно отметить, что полное дополнение увеличивает каждый размер на ту же величину $(F_q - 1)$, на которую отсутствие дополнения уменьшает их. Это соотношение не является простым совпадением, поскольку “обратная” операция свертки может быть реализована путем применения другой свертки к полностью дополненному выходу (исходной свертки) с использованием определенного подходящим образом ядра того же размера. Этот тип “обратной” свертки часто встречается в обратном распространении ошибки и алгоритмах автокодировщика для сверточных нейронных сетей. Полностью дополненные входы полезны тем, что они увеличивают пространственные размеры, что требуется в нескольких типах сверточных автокодировщиков.

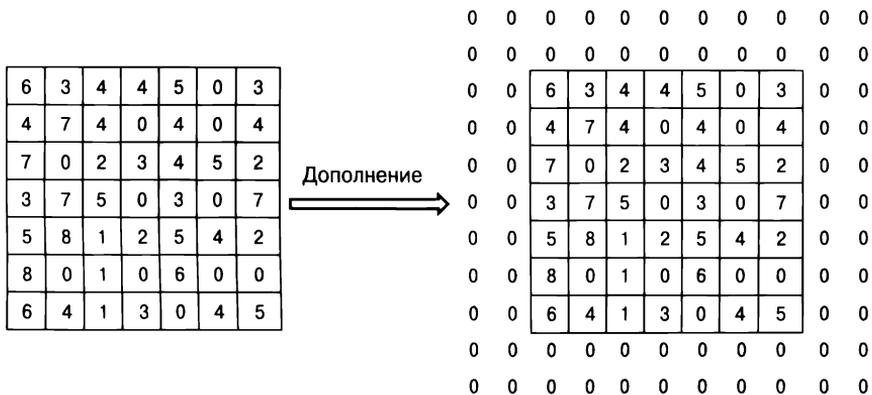


Рис. 8.3. Пример дополнения; таким же способом дополняется каждая из d_q карт активации по всей глубине q -го слоя

Другой полезной формой дополнения является *полное дополнение* (full padding). В этом случае мы позволяем фильтру (почти) полностью выступать за границы входного изображения со всех сторон. Иными словами, части фильтра размером $F_q - 1$ разрешено выступать с любой стороны входного изображения с перекрыванием лишь одного пространственного признака. Например, ядро входного изображения может перекрываться в одном пикселе в самом углу. Поэтому входное изображение дополняется $(F_q - 1)$ нулями с каждой стороны. Иначе говоря, каждый пространственный размер входного изображения увеличивается на $2(F_q - 1)$. Поэтому, если входные размеры исходного изображения равны L_q и B_q , то дополненные пространственные размеры входного объема составят $L_q + 2(F_q - 1)$ и $B_q + 2(F_q - 1)$. После выполнения свертки размеры карты признаков в слое $(q + 1)$ составят $L_q + F_q - 1$ и $B_q + F_q - 1$ соответственно. Несмотря на то что свертка обычно уменьшает пространственные размеры,

полное дополнение увеличивает их. Интересно отметить, что полное дополнение увеличивает каждый размер на ту же величину ($F_q - 1$), на которую отсутствие дополнения уменьшает их. Это соотношение не является простым совпадением, поскольку “обратная” операция свертки может быть реализована путем применения другой свертки к полностью дополненному выходу (исходной свертки) с использованием определенного подходящим образом ядра того же размера. Этот тип “обратной” свертки часто встречается в обратном распространении ошибки и алгоритмах автокодировщика для сверточных нейронных сетей. Полностью дополненные входы полезны тем, что они увеличивают пространственные размеры, что требуется в нескольких типах сверточных автокодировщиков.

8.2.2. Шаговая свертка

Существуют другие решения, в которых свертка позволяет уменьшить пространственную размерность изображения (или скрытого слоя). Описанный выше подход выполняет свертку в каждой позиции пространственного расположения карты признаков. Однако выполнение свертки в каждой пространственной позиции слоя не является обязательным. Уровень гранулярности свертки можно уменьшить, используя понятие *страйда* (stride), или *шага свертки*. Приведенное выше описание соответствует случаю, когда шаг равен 1. Если в q -м слое используется шаг S_q , то свертка выполняется в позициях 1, S_{q+1} , $2S_{q+1}$ и т.д. вдоль обоих пространственных измерений слоя. После выполнения такой свертки¹ выход имеет высоту, равную $(L_q - F_q)/S_{q+1}$, и ширину $(B_q - F_q)/S_{q+1}$. Таким образом, использование шаговой свертки будет приводить к уменьшению пространственных размеров слоя вдоль каждого измерения с коэффициентом, приблизительно равным S_q , и уменьшению площади с коэффициентом S_q^2 , хотя фактическое значение коэффициента может меняться в зависимости от краевых эффектов. Чаще всего используют шаг, равный 1, хотя иногда встречается и шаг, равный 2. В обычных условиях шаги величиной более 2 встречаются очень редко. Несмотря на то что в архитектуре сети-победителя соревнований ILSVRC, проводившихся в 2012 году [255], во входном слое был использован шаг, равный 4, в сети-победителе соревнований следующего года размер шага был уменьшен до 2 с целью улучшения точности [556]. Более крупные шаги могут быть полезными в условиях нехватки памяти или же в случаях, когда необходимо ослабить эффекты переобучения, если пространственное разрешение имеет неоправданно высокое значение. Следствием

¹ Здесь предполагается, что $(L_q - F_q)$ делится на S_q без остатка, что требуется для точного вписывания фильтра свертки в оригинальное изображение. В противном случае необходимо прибегать к специальным ухищрениям, чтобы учесть краевые эффекты. В общем случае такое решение нежелательно.

использования шаговой свертки является быстрое увеличение рецептивного поля архитектуры в скрытом слое с одновременным уменьшением размерности слоя в целом. Увеличение рецептивного поля полезно для выделения сложных признаков в крупных пространственных областях изображения. Как будет показано далее, процесс иерархического конструирования признаков сверточной нейронной сети позволяет выделять более сложные формы в поздних слоях. Для увеличения рецептивного поля ранее традиционно использовалась другая операция, известная как *подвыборка* (*субдискретизация*), или *пулинг по максимальному значению* (*max-pooling*). В последние годы вместо пулинга по максимуму, который мы рассмотрим позже, все чаще используют выборки с увеличенным шагом [184, 466].

8.2.3. Типичные параметры

В большинстве случаев шаг свертки устанавливается равным 1. Даже в случае применения шаговой свертки используются небольшие шаги величиной 2. Кроме того, принято устанавливать $L_q = B_q$. Иными словами, желательно работать с изображениями квадратной формы. В тех случаях, когда входные изображения не являются квадратными, прибегают к предварительной обработке, придающей изображениям квадратную форму. Например, можно извлечь квадратную область изображения для создания тренировочных данных. Количество фильтров в каждом слое часто устанавливают равным степени 2, поскольку это нередко приводит к более эффективной обработке. При таком подходе глубина слоев также становится равной степеням 2. Типичными значениями пространственных размеров фильтра (F_q) являются 3 или 5. В общем случае небольшие размеры фильтра часто способствуют получению наилучших результатов, хотя при использовании фильтров слишком малого размера возникают определенные трудности практического характера. Обычно использование фильтров небольшого размера приводит к увеличению глубины сети (при том же количестве параметров) и соответствующему повышению ее мощности. В действительности одна из сетей-призеров соревнований ILSVRC, сеть VGG [454], была первым экспериментом, в котором пространственный размер фильтра F_q для всех слоев был равен всего лишь 3, и, как оказалось, такой подход работает очень неплохо по сравнению с более крупными фильтрами.

Использование смещения

В CNN, как и в любой другой нейронной сети, в операции прямого распространения можно вводить смещение. С каждым уникальным фильтром в слое ассоциируется его собственное смещение. Таким образом, p -й фильтр в q -м слое имеет смещение $b^{(p, q)}$. При выполнении любой свертки в q -м слое с помощью p -го фильтра к скалярному произведению прибавляется значение $b^{(p, q)}$.

Использование смещения просто увеличивает количество параметров каждого фильтра на 1, что лишь незначительно увеличивает накладные расходы. Подобно другим параметрам, обучение смещения происходит на стадии обратного распространения ошибки. Смещение можно интерпретировать как вес соединения, вход которого всегда равен +1. Этот специальный вход используется во всех свертках независимо от их пространственного расположения. Таким образом, мы можем полагать, что на входе имеется специальный пиксель, значение которого всегда установлено в 1. Поэтому количество входных признаков в q -м слое равно $1 + L_q \times B_q \times d_q$. Это стандартный прием конструирования признаков, который используется для обработки смещения во всех формах машинного обучения.

8.2.4. Слой ReLU

Операция свертки перемежается с операциями пулинга и ReLU. ReLU-активация не слишком отличается от того, как она применяется в традиционных нейронных сетях. В результате применения функции активации ReLU к каждому из $L_q \times B_q \times d_q$ значений в слое создаются $L_q \times B_q \times d_q$ ограниченных порогом значений. Затем эти значения передаются следующему слою. Таким образом, применение ReLU не изменяет размерности слоя, поскольку эта операция представляет собой простую трансляцию активационных значений по принципу “один в один”. В традиционных нейронных сетях функция активации комбинируется с преобразованием, осуществляемым с помощью матрицы весов, для создания активаций следующего слоя. Точно так же в CNN за ReLU-активацией следует операция свертки (которая представляет собой грубый эквивалент линейного преобразования в традиционных сетях), и на иллюстративных изображениях архитектур сверточной нейронной сети слой ReLU часто не показывают в явном виде.

Следует отметить, что использование функции активации ReLU — сравнительно недавний результат эволюции дизайна нейронных сетей. В первые годы применялись функции активации с насыщением наподобие сигмоиды и гиперболического тангенса. Однако было показано [255], что использование ReLU обладает огромными преимуществами по сравнению с этими функциями активации в терминах скорости и точности. Увеличение скорости вычислений также связано с точностью, поскольку это позволяет использовать более глубокие модели и тренировать их в течение более длительного времени. В последние годы функции активации ReLU вытеснили другие активационные функции в архитектуре сверточных нейронных сетей до такой степени, что в этой главе мы будем использовать ReLU в качестве функции активации по умолчанию (если не оговорено иное).

8.2.5. Пулинг

Пулинг — совершенно другая операция. Пулинг работает на небольших областях сетки размером $P_q \times P_q$ в каждом слое и создает другой слой с той же глубиной (в отличие от фильтров). Для каждой из квадратных областей размером $P_q \times P_q$ в каждой из d_q карт признаков возвращается максимальное из значений. Этот подход носит название *пулинг по максимальному значению* (max-pooling). Если используется шаг, равный 1, то в результате будет создан новый слой размером $(L_q - P_q + 1) \times (B_q - P_q + 1) \times d_q$. Однако в случае пулинга чаще встречается шаг $S_q > 1$. В подобных случаях длина нового слоя будет составлять $(L_q - P_q)/S_{q+1}$, а ширина — $(B_q - P_q)/S_{q+1}$. Как следствие, пулинг резко уменьшает пространственные размеры каждой карты активации.

В отличие от операций свертки пулинг выполняется на уровне каждой карты активации. В то время как операция свертки использует одновременно все d_q карт признаков в сочетании с фильтром для создания значения одиночного признака. Поэтому в результате применения пулинга количество карт признаков не меняется. Иными словами, глубина слоя, создаваемого с использованием операции пулинга, остается той же, что и слоя, в котором выполнялся пулинг. Примеры пулинга с шагами 1 и 2 приведены на рис. 8.4. В данном случае используется пулинг по областям размером 3×3 . Типичный размер P_q области, по которой выполняется пулинг, составляет 2×2 . В этом случае при шаге 2 вообще не будет никакого перекрывания между различными областями пулинга, и такая настройка параметров довольно распространена. Однако иногда встречаются рекомендации, в соответствии с которыми по крайней мере некоторое перекрывание между областями, где применяется пулинг, является желательным, поскольку это делает данный подход менее подверженным переобучению.

Возможны и другие типы пулинга (такие, как пулинг по среднему значению), но они редко применяются. В самой ранней сверточной сети под названием LeNet-5 использовался вариант пулинга по среднему значению, на который ссылались² как на *субдискретизацию* (subsampling). Если говорить в целом, то пулинг по максимуму пользуется большей популярностью, чем пулинг по среднему. Слои пулинга по максимуму перемежаются со сверточными/ReLU-слоями, хотя в глубоких архитектурах они, как правило, встречаются не так часто. Это объясняется тем, что пулинг резко снижает пространственные размеры карты признаков, так что для снижения размера пространственной карты до небольшой постоянной величины потребуется не так уж много операций пулинга.

² В последние годы под субдискретизацией также понимают другие операции, снижающие размерность. Поэтому между классическим и современным толкованием этого термина существуют определенные различия.

В большинстве случаев быстрое снижение пространственных размеров слоев (и соответствующее увеличение рецептивных полей признаков) обеспечивается операциями пулинга. При шаге, большем 1, операции свертки лишь незначительно увеличивают рецептивное поле. В последние годы было выдвинуто предположение, что пулинг не всегда необходим. Можно спроектировать сеть, использующую только сверточные и ReLU-операции, и получить расширение рецептивного поля, используя более крупные шаги в операциях свертки [184, 466]. Поэтому в последние годы наметилась тенденция вообще избавиться от слоев пулинга по максимальному значению. Однако на момент выхода книги эта тенденция еще не установилась и не получила достаточно подтверждений. В пользу пулинга по максимальному значению можно выдвинуть по крайней мере несколько аргументов. Он вводит нелинейность и трансляционную инвариантность в большей степени, чем шаговые свертки. И хотя нелинейность можно обеспечить с помощью функций активации ReLU, важно то, что эффекты пулинга по максимальному значению не могут быть в точности воспроизведены с помощью шаговой свертки. Как бы то ни было, эти две операции не являются полностью взаимозаменяемыми.

8.2.6. Полносвязные слои

Каждый признак в последнем пространственном слое связан с каждым скрытым состоянием в первом полносвязном слое. Этот слой функционирует точно так же, как и в традиционной сети прямого распространения. В большинстве случаев для увеличения окончательной мощности вычислений можно использовать дополнительные полносвязные слои. Связи между этими слоями структурируются в точности так, как в традиционной сети прямого распространения. Поскольку полносвязные слои являются плотно связанными, большинство параметров относится к полносвязным слоям. Например, если каждый из двух полносвязных слоев содержит 4096 скрытых элементов, то соединения между ними имеют более 16 миллионов весов. Аналогичным образом соединения между последним пространственным слоем и первым полносвязным слоем будут характеризоваться большим количеством параметров. Несмотря на то что сверточные слои имеют большое количество активаций (для хранения которых требуется дополнительная память), полносвязные слои часто имеют большое количество соединений (и параметров). Причиной того, что активации выдвигают значительные дополнительные требования к доступным объемам памяти, является то, что количество активаций умножается на размер мини-пакета при отслеживании переменных во время прохождения прямой и обратной фаз процесса обратного распространения ошибки. Выбирая конкретный вариант реализации нейронной сети в условиях существования специфических ограничений на доступные ресурсы, полезно не упускать из виду необходимость

достижения определенных компромиссов (например, между объемом данных и объемом доступной памяти). Следует отметить, что природа полносвязного слоя может быть чувствительной к специфике конкретного приложения. Например, полносвязные слои в задачах классификации и в задачах сегментации будут иметь немного разную природу. Наше обсуждение относится к наиболее распространенному случаю классификации.

Выходной слой сверточной нейронной сети проектируется в соответствии со спецификой приложения. В дальнейшем мы рассмотрим в качестве примера задачу классификации. В этом случае выходной слой соединен с каждым нейроном предпоследнего слоя, и с каждой такой связью ассоциирован определенный вес. В зависимости от природы приложения (например, классификация или регрессия) можно использовать логистическую, линейную или Softmax-активацию.

Одна из возможных альтернатив использованию полносвязных слоев — применить пулинг по среднему значению ко всей пространственной области последнего набора карт активации для создания одного значения. Поэтому количество признаков, созданных в последнем пространственном слое, будет в точности равно количеству фильтров. В этом сценарии, если последняя карта активации имеет размер $7 \times 7 \times 256$, будет создано 256 признаков. Каждый признак будет представлять собой результат агрегирования 49 значений. Подход такого типа позволяет значительно уменьшить количество параметров полносвязных слоев и обеспечивает определенные преимущества в плане обобщаемости. Этот подход применялся в сети GoogLeNet [485]. В некоторых приложениях, таких как сегментация изображений, с каждым пикселем связывается метка класса, а полносвязные слои вообще не используются. Полносверточные слои со свертками 1×1 использовались для создания выходной пространственной карты.

8.2.7. Чередование слоев

Как правило, слои свертки, пулинга и ReLU чередуются в нейронной сети для усиления ее выразительной способности. Слои ReLU часто следуют за сверточными слоями подобно тому, как нелинейная функция активации обычно следует за линейной операцией скалярного произведения в традиционных нейронных сетях. Поэтому в типичных случаях слои свертки и ReLU идут в паре. На некоторых графических иллюстрациях нейронных архитектур, таких как AlexNet [255], слои ReLU не показаны в явном виде, поскольку предполагается, что они всегда присоединяются к линейным сверточным слоям. Вслед за двумя-тремя наборами комбинаций “свертка — ReLU” может включаться слой пулинга по максимальному значению. В качестве примера можно привести следующие возможные варианты этой базовой схемы:

CRCRP
CRCRCRP

Здесь буквой С обозначен сверточный слой, буквой R — слой ReLU, буквой P — слой пулинга по максимальному значению. Весь образец (включая слой пулинга по максимальному значению) может быть повторен несколько раз для создания глубокой нейронной сети. Например, если повторить первый из приведенных выше образцов три раза и присоединить полносвязный слой (обозначенный буквой F), то мы получим следующую нейронную сеть:

CRCRPCRCRPCRCRPF

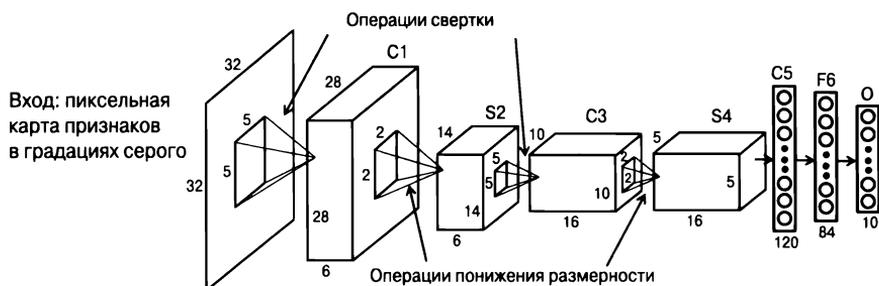
Это описание не является полным, поскольку в дополнение к нему необходимо также указать количество/размер/дополнение фильтров/слоев пулинга. Слой пулинга — ключевой шаг, приводящий к уменьшению пространственных размеров карт активации, поскольку он использует шаги, величина которых превышает 1. Пространственные размеры также можно уменьшать, используя шаговую свертку вместо пулинга по максимальному значению. Результирующие сети часто получаются довольно глубокими, и не столь уж редко можно встретить сверточные сети с более чем 15 слоями. В недавних архитектурах также используется опускание связей между слоями, что становится особенно важным по мере увеличения глубины сети (раздел 8.4.5).

LeNet-5

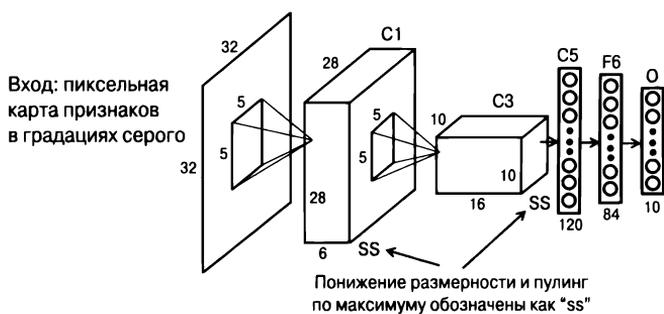
Ранние сети были довольно мелкими. В качестве примера одной из них можно привести сеть *LeNet-5* [279]. Входные данные в ней предоставляются в градациях серого, и существует только один цветовой канал. Предполагается, что входом является ASCII-представление символа. Для целей нашего обсуждения будем предполагать, что всего существует десять типов символов (и поэтому имеется 10 выходов), хотя данный подход можно использовать для любого количества классов.

Сеть *LeNet-5* содержала два сверточных слоя, два слоя пулинга и три завершающих полносвязных слоя. Однако поздние слои содержали несколько карт признаков, поскольку в каждом слое использовалось несколько фильтров. Архитектура этой сети представлена на рис. 8.5. В оригинальной работе первый полносвязный слой также назывался сверточным (обозначен как C5), поскольку существовала возможность его обобщения на пространственные признаки для более крупных входных карт. Однако в конкретной реализации сети *LeNet-5* слой C5 в действительности использовался в качестве полносвязного слоя, поскольку пространственные размеры фильтра совпадали с пространственными размерами входа. Именно по этой причине мы считаем слой C5 полносвязным в данном случае. Обратите внимание на то, что на рис. 8.5 представлены две

версии сети LeNet-5. На верхней диаграмме (рис. 8.5, а) в явном виде отображены слои субдискретизации, и именно так данная архитектура была представлена в оригинальной работе. Однако в случае диаграмм более глубоких архитектур, таких как *AlexNet* [255], слои субдискретизации или пулинга по максимальному значению обычно не показывают в явном виде, чтобы освободить место для большего количества слоев. В таком компактном виде архитектура LeNet-5 представлена на рис. 8.5, б. Слои функции активации также не представлены на рисунке в явном виде. В оригинальной работе, в которой была описана сеть LeNet-5, сигмоидная функция активации применяется сразу же вслед за операцией субдискретизации, хотя такой порядок их следования относительно редко встречается в последних архитектурах. В большинстве современных архитектур субдискретизация заменяется пулингом по максимальному значению, а слои пулинга по максимуму встречаются реже, чем сверточные слои. Кроме того, обычно активация выполняется сразу же после каждой свертки (а не после каждой операции пулинга по максимуму).



а) Детальное представление архитектуры



б) Компактное представление архитектуры

Рис. 8.5. LeNet-5 — одна из ранних сверточных нейронных сетей

Количество слоев в этой архитектуре часто подсчитывают в терминах количества слоев со взвешенными пространственными фильтрами и количества полносвязных слоев. Иными словами, слои субдискретизации/пулинга по

максимальному значению и слою функции активации отдельно не подсчитываются. В LeNet-5 субдискретизация осуществлялась с использованием пространственных областей размером 2×2 и шага 2. Кроме того, в отличие от пулинга по максимальному значению, значения усреднялись, масштабировались с обучаемым весом, после чего добавлялось смещение. Современные архитектуры обходятся без линейных операций масштабирования и смещения. Компактное представление архитектуры, приведенное на рис. 8.5, б, может несколько смущать новичков, поскольку в нем отсутствуют такие детали, как размер фильтров пулинга по максимальному значению/субдискретизации. В действительности для представления таких архитектурных деталей не существует какого-то единого способа, и разные авторы делают это по-разному. В данной главе вы познакомитесь с несколькими такими способами при рассмотрении типовых примеров.

По современным стандартам это предельно мелкая сеть, однако базовые принципы с тех пор не изменились. Основное отличие состоит в том, что ReLU-активация на тот момент еще не была известна, и в ранних архитектурах часто применялась сигмоидная активация. Кроме того, использование пулинга по среднему значению — чрезвычайно редкое явление в наши дни по сравнению с пулингом по максимальному значению. В последние годы наметился отход от использования как пулинга по максимальному значению, так и субдискретизации, и предпочтение отдается шаговым сверткам. В LeNet-5 также использовались десять элементов на основе радиальных базисных функций (RBF) в последнем слое (см. главу 5), где прототип каждого элемента сравнивался с его входным вектором и выходом служил квадрат евклидова расстояния между ними. Это то же самое, что использование отрицательного логарифмического правдоподобия гауссовского распределения, представленного RBF-элементами. Векторы параметров RBF-элементов выбирались вручную и соответствовали стилизованным растровым изображениям размером 7×12 соответствующих классов символов, которые уплощались до 84-мерного (7×12) представления. Обратите внимание на то, что размер предпоследнего слоя в точности равен 84, чтобы обеспечить возможность вычисления евклидова расстояния между вектором, соответствующим данному слою, и вектором параметров RBF-элемента. Десять выходов последнего слоя предоставляют оценки соответствующих классов, и в качестве предсказания берется наименьшая из оценок. Такой тип использования RBF-элементов является анахронизмом в современных сверточных сетях, и обычно стараются работать с Softmax-элементами и функциями потерь в виде логарифмического правдоподобия на выходах мультиномиальных меток. Сеть LeNet-5 интенсивно применялась для распознавания символов, а также многими банками для считывания чеков.

8.2.8. Нормализация локального отклика

В [255] был представлен трюк под названием *нормализация локального отклика* (local response normalization), который всегда применяется вслед за слоем ReLU и способствует улучшению обобщающей способности сети. Базовая идея такого подхода к нормализации была почерпнута из биологии, и ее суть состоит в создании связности между различными фильтрами. Сначала мы опишем формулу нормализации, использующую все фильтры, а затем покажем, как фактически выполнять вычисления с использованием лишь некоторого подмножества фильтров. Рассмотрим ситуацию, в которой слой содержит N фильтров, а значениями активации этих N фильтров в конкретной пространственной позиции (x, y) являются $a_1 \dots a_N$. Тогда каждое значение a_i преобразуется в нормализованное значение b_i с помощью следующей формулы:

$$b_i = \frac{a_i}{(k + \alpha \sum_j \alpha_j^2)^\beta}. \quad (8.1)$$

В [255] для базовых параметров были приняты значения $k = 2$, $\alpha = 10^{-4}$ и $\beta = 0,75$. Однако на практике нет нужды выполнять нормализацию по всем N фильтрам. Вместо этого фильтры заранее упорядочиваются произвольным образом для определения отношений “соседства” между фильтрами. После этого выполняется нормализация по каждому набору из n “соседних” фильтров для некоторого значения параметра n . В [255] для n использовалось значение 5. Таким образом, мы получаем следующую формулу:

$$b_i = \frac{a_i}{(k + \alpha \sum_{j=i-\lfloor n/2 \rfloor}^{i+\lfloor n/2 \rfloor} \alpha_j^2)^\beta}. \quad (8.2)$$

В ней любое значение $i - n/2$, меньшее нуля, устанавливается равным нулю, а любое значение $i + n/2$, превышающее N , устанавливается равным N . В настоящее время этот тип нормализации считается устаревшим, и его обсуждение дано исключительно для ознакомления.

8.2.9. Иерархическое конструирование признаков

Будет весьма поучительно исследовать активации фильтров, созданных реальными изображениями в различных слоях. В разделе 8.5 мы обсудим конкретный способ, которым могут быть визуализированы признаки, извлеченные из различных слоев. А пока что дадим субъективную интерпретацию. Активации признаков в ранних слоях представляют собой низкоуровневые признаки наподобие краев, тогда как признаки в поздних слоях сводят эти низкоуровневые признаки вместе. Например, признаки среднего уровня могут объединить

линии в шестиугольник, тогда как высокоуровневый признак может объединить шестиугольники среднего уровня для создания сотовой структуры. Понять, почему низкоуровневый фильтр может обнаруживать границы, не составляет труда. Рассмотрим ситуацию, в которой цвет изображения изменяется на границе. В результате разница между значениями соседних пикселей будет ненулевой лишь при переходе через эту границу. Тогда для получения желаемого результата нужно лишь подобрать подходящие веса в низкоуровневом фильтре. Обратите внимание на то, что для обнаружения горизонтальных линий потребуется другой фильтр. Это возвращает нас к экспериментам Хубеля и Визеля по возбуждению зрительной коры кошек линиями с различной ориентацией. Примеры фильтров, обнаруживающих горизонтальные и вертикальные линии, приведены на рис. 8.6. Фильтр следующего слоя работает со скрытыми признаками, поэтому его результаты труднее поддаются интерпретации. Тем не менее фильтр следующего слоя способен обнаруживать прямоугольники, сочетая горизонтальные и вертикальные линии.

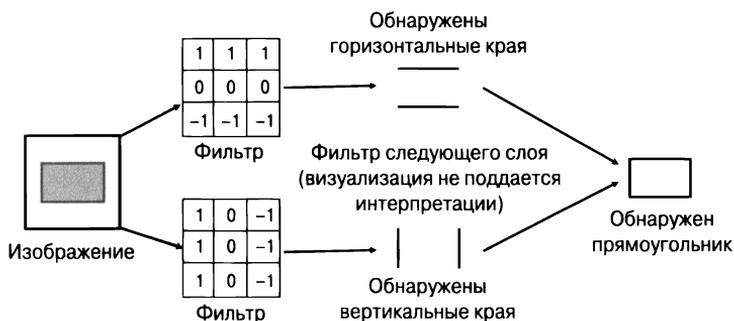


Рис. 8.6. Фильтры обнаруживают края и комбинируют их для создания прямоугольников

В одном из последующих разделов мы предоставим визуализацию того, как небольшие части реального изображения активируют различные скрытые признаки, во многом подобно биологической модели Хубеля и Визеля, в которой различные формы активируют различные нейроны. Мощь сверточных нейронных сетей зиждется на способности объединять примитивные формы в более сложные слой за слоем. Отметим, что первый сверточный слой не в состоянии обучиться любому признаку, размеры которого превышают $F_1 \times F_1$ пикселей, где типичное значение F_1 — небольшое число, например 3 или 5. Однако следующий сверточный уже способен объединить многие из этих фрагментов для создания признака из более крупной области изображения. Примитивные признаки, изученные в ранних слоях, объединяются логичным в семантическом отношении способом для обучения интерпретируемых визуальных признаков все увеличивающейся сложности. На выбор обучаемых признаков влияет то, как

обратное распространение ошибки адаптирует признаки к нуждам используемой функции потерь. Например, если приложение тренируется для классификации изображений как автомобилей, то этот подход может обеспечить обучение дугам для объединения их в окружности, а затем — объединению окружностей с другими формами для создания автомобильного колеса. Все это становится возможным благодаря иерархическим признакам глубокой сети.

Недавние соревнования *ImageNet* продемонстрировали, что значительная часть возможностей по распознаванию образов кроется в увеличении глубины сети. Отсутствие достаточного количества слоев в конечном счете препятствует обучению сети иерархическим регулярным образованиям в изображении, объединение которых позволило бы создавать семантически значимые компоненты. Также важно отметить, что природа обучаемых признаков будет чувствительна к специфике конкретных данных. Например, признаки, обучаемые для распознавания грузовиков, будут отличаться от тех, которые обучаются для распознавания моркови. В то же время некоторые наборы данных (наподобие *ImageNet*) достаточно разнообразны для того, чтобы обучаемые на них признаки имели универсальную значимость для многих приложений.

8.3. Тренировка сверточной сети

Процесс тренировки сверточной нейронной сети задействует алгоритм обратного распространения ошибки. Существуют три основных типа слоев: сверточный, ReLU и пулинга по максимальному значению. Мы предоставим отдельные описания алгоритма обратного распространения ошибки через каждый из этих слоев. Алгоритм обратного распространения через слой ReLU относительно прост, поскольку он не отличается от алгоритма, применяемого в традиционных нейронных сетях. В случае пулинга по максимальному значению и в отсутствие перекрытия между пулами, т.е. областями подвыборки, необходимо лишь определить, какой элемент имеет максимальное значение в пуле (с произвольно разрываемыми или пропорционально разделяемыми связями). Частная производная функции потерь по подвыборочному состоянию распространяется обратно к элементу с максимальным значением. Всем остальным ячейкам сетки назначается нулевое значение. Заметим, что обратное распространение через слой максимизации также описано в табл. 3.1. В тех случаях, когда пулы перекрываются, обозначим через $P_1 \dots P_r$ пулы, в которые включается элемент h , с соответствующими активациями $h_1 \dots h_r$ в следующем слое. Если h — максимальное значение в пуле P_i (и поэтому $h_i = h$), то градиент функции потерь по h_i распространяется обратно в h (с произвольно разрываемыми или пропорционально разделяемыми связями). Вклады различных перекрывающихся пулов (от $h_1 \dots h_r$ в следующем слое) суммируются

для вычисления градиента по элементу h . Поэтому обратное распространение через максимизацию и операции ReLU не очень отличаются от таковых в традиционных нейронных сетях.

8.3.1. Обратное распространение ошибки через свертки

Обратное распространение через свертки также не очень отличается от обратного распространения через линейные преобразования (например, матричные умножения) в сетях прямого распространения. Эта точка зрения станет особенно понятной, когда мы представим свертки в виде разновидности операций матричного умножения. Точно так же, как обратное распространение в сетях прямого распространения от слоя $(i + 1)$ к слою i достигается умножением производных ошибки по слою $(i + 1)$ на транспонированную матрицу прямого распространения между слоями i и $(i + 1)$ (см. табл. 3.1), обратное распространение в сверточных сетях также можно рассматривать как некую форму транспонированной свертки.

Сначала опишем простой поэлементный подход к обратному распространению ошибки. Предположим, что градиенты функции потерь ячеек в слое $(i + 1)$ уже вычислены. Производная функции потерь по ячейке в слое $(i + 1)$ определяется как частная производная функции потерь по скрытой переменной в этой ячейке. Операции свертки умножают активации в слое i на элементы фильтра для создания элементов в следующем слое. Поэтому ячейка в слое $(i + 1)$ получает агрегированные вклады от 3-мерного объема элементов в предыдущем слое фильтра размером $F_i \times F_i \times d_i$. В то же время ячейка c в слое i вносит вклад в несколько элементов (обозначенных как набор S_c) в слое $(i + 1)$, хотя количество элементов, в которые она вносит вклад, зависит от глубины следующего слоя и шага. Идентификация этого “прямого набора” — ключ к обратному распространению ошибки. Здесь важно то, что ячейка c вносит аддитивный вклад в каждый из элементов набора S_c после умножения активации ячейки c на элемент фильтра. Поэтому алгоритм обратного распространения должен просто умножить производную функции потерь каждого из элементов набора S_c по соответствующему элементу фильтра и агрегировать ее в обратном направлении. Ниже приведен псевдокод, который может быть использован для обратного распространения существующих производных в слое $(i + 1)$ к ячейке c в слое i .

- Идентифицировать все ячейки S_c в слое $(i + 1)$, в которые вносит вклад ячейка c в слое i .
- Для каждой ячейки $r \in S_c$: пусть δ_r — производная функции потерь по отношению к ячейке r (после обратного распространения).

- Для каждой ячейки $r \in S_c$: пусть w_r — вес элемента фильтра, используемого для свертки из ячеек c в ячейку r .

$$\delta_c = \sum_{r \in S_c} \delta_r \cdot w_r.$$

После вычисления градиентов функции потерь значения умножаются на значения скрытых элементов $(i - 1)$ -го слоя для получения градиентов по весам соединений, связывающих $(i - 1)$ -й и i -й слои. Иными словами, скрытое значение на одном конце связи умножается на градиент функции потерь на другом конце для получения частной производной по весу. Однако в этих вычислениях предполагается, что веса различны, тогда как веса в фильтре разделяются на всей пространственной протяженности слоя. Для учета этого факта необходимо суммировать частные производные по всем копиям разделяемых весов. Иными словами, сначала мы вычисляем частные производные по каждой копии разделяемого веса, действуя так, словно в каждой позиции используются различные фильтры, а затем суммируем частные производные функции потерь по всем копиям разделяемого веса.

Заметьте, что в описанном выше подходе используется простое линейное аккумулирование градиентов, как в традиционном обратном распространении ошибки. Однако при этом не помешает уделить внимание отслеживанию ячеек, влияющих на поведение других ячеек в следующем слое. Алгоритм обратного распространения можно реализовать с помощью операций тензорного умножения, которые далее могут быть упрощены до умножения матриц, получаемых из этих тензоров. Эта точка зрения обсуждается в следующих двух разделах, поскольку она способствует более глубокому пониманию того, насколько многочисленны те аспекты сетей прямого распространения, которые могут быть обобщены на сверточные нейронные сети.

8.3.2. Обратное распространение ошибки как свертка с инвертированным/транспонированным фильтром

В сверточных нейронных сетях операция обратного распространения ошибки выполняется путем умножения вектора градиентов в слое $(q + 1)$ на транспонированную матрицу весов связей между слоями q и $(q + 1)$ для получения вектора градиентов в слое q (см. табл. 3.1). В сверточных нейронных сетях производные, распространенные в обратном направлении, связываются также с пространственными позициями в слоях. Существует ли сверточная аналогия, которую мы могли бы применить к пространственной части распространенных в обратном направлении производных в некотором слое для получения аналогичных производных в предыдущем слое? Оказывается, это действительно возможно.

Рассмотрим случай, когда активации в слое q свертываются с фильтром для создания активаций в слое $(q + 1)$. Для простоты предположим, что глубина d_q входного слоя и глубина d_{q+1} выходного слоя равны 1; кроме того, используем свертки с шагом 1. В этом случае фильтр свертки инвертируется как по горизонтали, так и по вертикали для обратного распространения ошибки. Пример такого инвертированного фильтра приведен на рис. 8.7. Интуитивной причиной такого обращения служит тот факт, что фильтр “движется” по пространственной области входного объема для выполнения скалярного произведения, в то время как распространяемые в обратном направлении производные вычисляются по входному объему, перемещение которого относительно фильтра противоположно перемещению фильтра в процессе свертки. Заметим, что левая верхняя ячейка фильтра может даже не вносить вклад в верхнюю левую ячейку выходного объема (вследствие использования дополнения), но она почти всегда будет вносить вклад в левую нижнюю ячейку выходного объема. Это согласуется с инверсией фильтра. Обратное распространение набора производных $(q + 1)$ -го слоя свертывается с этим инвертированным фильтром для получения распространенного в обратном направлении набора производных q -го слоя. Каким образом дополнения сверток в прямом и обратном направлениях соотносятся между собой? Для шага 1 сумма дополнений в процессе прямого и обратного распространения равна $F_q - 1$, где F_q — длина стороны фильтра для q -го слоя.

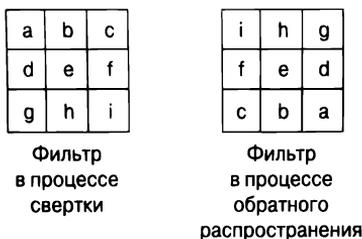


Рис. 8.7. Обращение ядра для обратного распространения ошибки

Рассмотрим случай, когда значения глубины d_q и d_{q+1} могут быть произвольными, а не равными только 1. В этом случае возникает необходимость в дополнительном транспонировании тензора. Весом (i, j, k) -й позиции p -го фильтра в q -м слое является $\mathcal{W} = [w_{ijk}^{(p, q)}]$. Учтите, что индексы i и j относятся к пространственным позициям, тогда как индекс k — к позиции глубины для данного веса. Обозначим 5-мерный тензор, соответствующий фильтрам обратного распространения от слоя $q + 1$ к слою q , как $\mathcal{U} = [u_{ijk}^{(p, q+1)}]$. Тогда элементы этого тензора определяются следующим образом:

$$u_{rsp}^{(k, q+1)} = w_{ijk}^{(p, q)}, \quad (8.3)$$

где $r = F_q - i + 1$ и $s = F_q - j + 1$. Обратите внимание на обмен индексами p идентификатора фильтра и глубины k в пределах фильтра между \mathcal{W} и \mathcal{U} в уравнении 8.3. Это и есть тензорное транспонирование.

Чтобы понять смысл описанного выше транспонирования, рассмотрим ситуацию, в которой мы применяем 20 фильтров к 3-канальному RGB-объему для создания выходного объема глубиной 20. В процессе обратного распространения ошибки потребуется брать *градиентный объем глубиной 20* и преобразовывать его в *градиентный объем глубиной 3*. Поэтому мы должны создать для обратного распространения 3 фильтра, предназначенных для обработки красного, зеленого и синего цветов. Мы извлекаем 20 пространственных срезов из этих 20 фильтров, которые применяются к красному цвету, инвертируем их, используя подход, представленный на рис. 8.7, а затем создаем фильтр глубиной 20 для обратного распространения градиентов по красному срезу. Аналогичные подходы применяются в отношении зеленого и синего цветов. Этим операциям соответствуют транспонирование и инверсия в уравнении 8.3.

8.3.3. Свертка и обратное распространение ошибки как матричное умножение

Целесообразно рассмотреть свертку как операцию матричного умножения, поскольку это облегчает определение различных родственные понятий, таких как *транспонированная свертка* (transposed convolution), *деконволюция* (deconvolution) и *дробная свертка* (fractional convolution). Эти понятия важны не только для понимания обратного распространения, но и для разработки средств, необходимых для сверточных автокодировщиков. В традиционных сетях прямого распространения матрицы, используемые для преобразования скрытых состояний во время прямой фазы, транспонируются во время обратной фазы (см. табл. 3.1) для обратного распространения частных производных по слоям. Точно так же матрицы, используемые в кодировщиках, часто транспонируются в декодировщиках, работающих в автокодировщиках в традиционных условиях. Несмотря на то что пространственная структура сверточной нейронной сети маскирует природу базового матричного умножения, ее можно “уплосщить” для выполнения умножения, а затем вернуть ей первоначальную форму пространственной структуры, используя известные пространственные позиции элементов уплосщенной матрицы. Такой немного косвенный подход способствует пониманию того факта, что на самом базовом уровне операция свертки аналогична матричному умножению в сетях прямого распространения. Кроме того, на практике реализация операций свертки часто осуществляется с использованием матричного умножения.

Для простоты сначала рассмотрим случай, когда q -й слой и соответствующий фильтр, используемый для свертки, имеют единичную глубину. Кроме того, предположим, что мы используем шаг, равный 1, с дополнением нулями. Тогда размерность входа равна $L_q \times B_q \times 1$, а выхода — $(L_q - F_q + 1) \times (B_q - F_q + 1) \times 1$. Обычно пространственные размеры соответствуют квадрату (т.е. $L_q = B_q$), и в этих условиях вход имеет пространственные размеры $A_I = L_q \times L_q$, а выход — $A_O = (L_q - F_q + 1) \times (L_q - F_q + 1)$. Здесь A_I и A_O — пространственные области матриц входа и выхода соответственно. Мы можем представить вход, сплющив область A_I в A_I -мерный вектор-столбец, в котором строки пространственной области конкатенируются сверху донизу. Обозначим этот вектор как \bar{f} . Соответствующий пример, в котором используется фильтр размером 2×2 и вход размером 3×3 , приведен на рис. 8.8. Поэтому выход имеет размеры 2×2 , и мы получаем: $A_I = 3 \times 3 = 9$ и $A_O = 2 \times 2 = 4$. На рис. 8.8 приведен также 9-мерный вектор-столбец для входа размером 3×3 . Вместо фильтра определяется разреженная матрица C , играющая ключевую роль в представлении свертки как матричного умножения. Мы определяем матрицу размером $A_O \times A_I$, в которой каждая строка соответствует свертке в одной из позиций области A_O . Эти строки ассоциируются с позицией левого верхнего угла области свертки во входной матрице, из которой они были извлечены. Значение каждого элемента в строке соответствует одной из позиций A_I во входной матрице, но это значение равно нулю, если входная позиция не вовлечена в свертку для строки. В противном случае значения устанавливаются равными соответствующим значениям фильтра, которые используются для умножения. Упорядочение элементов в строке основывается на том же принципе чувствительного к пространственным позициям упорядочения, которое применялось для сплющивания входной матрицы в A_I -мерный вектор. Поскольку размер фильтра обычно намного меньше размера входа, большинство элементов матрицы C являются нулевыми. Поэтому каждый элемент фильтра повторяется A_O раз в C , так как он используется для A_O операций умножения.

Пример матрицы C размером 4×9 приведен на рис. 8.8. Последующее умножение C на \bar{f} производит A_O -мерный вектор. Соответствующий 4-мерный вектор тоже показан на рис. 8.8. Поскольку каждая из A_O строк матрицы C ассоциирована с пространственным расположением, эти расположения наследуются произведением $C\bar{f}$ и используются для переформирования $C\bar{f}$ в пространственную матрицу. Переформирование 4-мерного вектора в матрицу размером 2×2 также показано на рис. 8.8.

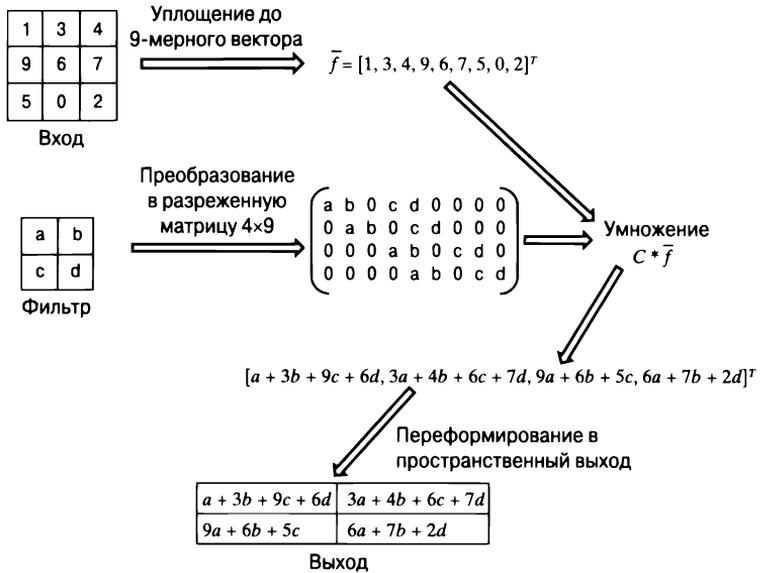


Рис. 8.8. Свертка как матричное умножение

Данное рассмотрение относится к упрощенному случаю, когда глубина равна 1. В тех случаях, когда глубина превышает 1, тот же подход применяется к каждому из 2-мерных срезов, и результаты суммируются. Иными словами, мы агрегируем величину $\sum_p C_p \vec{f}_p$ по различным индексам срезов p , а затем переформируем результаты в 2-мерную матрицу. Этот подход сводится к тензорному умножению, которое является непосредственным обобщением матричного умножения. Именно тензорное умножение лежит в основе фактической реализации свертки на практике. В общем случае мы будем иметь множество фильтров, соответствующих множеству выходных карт. Тогда k -й фильтр будет образовываться в разреженную матрицу $C_{p,k}$, а k -й картой признака выходного объема будет $\sum_p C_{p,k} \vec{f}_p$.

Подход, ориентированный на использование матриц, оказывается весьма полезным для обратного распространения ошибки, поскольку это дает возможность распространять градиенты, применяя тот же подход в обратном направлении, за исключением того, что для умножения на уплощенную векторную версию 2-мерного среза выходного градиента используется транспонированная матрица C^T . Уплотнение градиента по пространственной карте может быть выполнено аналогичным способом, поскольку уплощенный вектор \vec{f} создается на этапе прямого распространения. Рассмотрим простой случай, когда как входной, так и выходной объем имеют глубину 1. Если \vec{g} — уплощенный вектор градиента функции потерь по выходной пространственной карте, то

уплощенный градиент по входной пространственной карте получается в виде произведения $C^T \bar{g}$. Этот подход согласуется с тем, который применяется в сетях прямого распространения, когда при обратном распространении используется транспонированная матрица прямого распространения. Приведенный выше результат относится к простому случаю, когда как входной, так и выходной объем имеют глубину 1. А как обстоят дела в общем случае? Пусть глубина выходного объема $d > 1$; тогда обозначим градиенты по выходным картам $g_1 \dots g_d$. Соответствующий градиент по признакам в p -м пространственном срезе входного объема задается суммой $\sum_{k=1}^d C_{p,k}^T \bar{g}_k$. Здесь матрица $C_{p,k}$ получена путем преобразования p -го пространственного среза k -го фильтра в разреженную матрицу в соответствии с приведенным выше обсуждением. Данный подход является следствием уравнения 8.3. Такой тип транспонированной свертки также может быть использован для выполнения операции деконволюции в сверточных автокодировщиках, о чем речь пойдет в раздел 8.5.

8.3.4. Аугментация данных

Распространенным приемом, используемым для снижения эффектов переобучения в сверточных нейронных сетях, является *аугментация данных*. В этом случае новые тренировочные примеры генерируются за счет преобразования исходных примеров. Эта идея кратко обсуждалась в главе 4, хотя в одних областях она работает лучше, чем в других. Обработка изображений является одной из тех областей, для которых аугментация данных хорошо приспособлена. Это объясняется тем, что многие преобразования, такие как трансляция, вращение, извлечение фрагментов и отражение, в основном не изменяют свойства объекта на изображении. В то же время они увеличивают обобщающую способность сети, если тренировка осуществляется с использованием аугментированного набора данных. Например, если применять для тренировки зеркальные изображения и отраженные версии всех бананов, представленных на них, то модель будет лучше распознавать бананы в различных ориентациях.

Многие из подобных форм аугментации данных требуют весьма незначительных вычислений, и поэтому аугментированные изображения не должны генерироваться заранее. Вместо этого их можно создавать во время обучения при обработке изображения. Например, в случае обработки изображения банана его искаженную версию можно получить во время тренировки. Аналогичным образом тот же банан можно представить с другим соотношением интенсивностей цветов в различных изображениях, поэтому может быть полезным создание представлений того же изображения с использованием различных интенсивностей цветов. Во многих случаях пригодится создание тренировочного набора данных на основе фрагментов изображения. В возрождении интереса к

глубокому обучению важную роль сыграла сеть *AlexNet*, в свое время ставшая победителем соревнований ILSVRC. Эта сеть обучалась путем извлечения из изображений фрагментов размером $224 \times 224 \times 3$, что определило размеры входа сети. Аналогичная методология извлечения фрагментов применялась и в сетях, участвующих в соревнованиях ILSVRC в последующие годы.

Несмотря на то что большинство методов аугментации данных продемонстрировало довольно высокую эффективность, некоторые виды преобразований используют анализ главных компонент (PCA), который может значительно увеличивать накладные расходы. Метод PCA применяется для изменения интенсивности цветов в изображении. В тех случаях, когда вычислительные расходы становятся слишком высокими, целесообразно заранее извлекать и сохранять изображения. Базовая идея заключается в использовании для каждого значения пикселя ковариационной матрицы размером 3×3 и вычислении главных компонент. Затем к каждой главной компоненте добавляется гауссовский шум с нулевым средним и дисперсией 0,01. Этот шум фиксируется по всем пикселям отдельного изображения. Данный подход основан на том факте, что идентичность объекта инвариантна по отношению к изменениям интенсивности цветов и освещения. Сообщалось [255], что аугментация набора данных снижает коэффициент ошибок на 1%.

Очень важно не применять аугментацию вслепую, безотносительно к конкретному набору данных или приложению. Например, применение поворотов и отражений к набору данных MNIST [281] рукописных цифр — неудачная идея, поскольку все цифры в нем представлены в одной и той же ориентации. Кроме того, зеркальное изображение рукописной цифры не является действительной цифрой, а повернутое '6' выглядит как '9'. Принимая решение о том, аугментация какого типа будет разумной, очень важно учитывать естественное распределение изображений в полном наборе данных, а также влияние конкретного типа аугментации набора данных на метки классов.

8.4. Примеры типичных сверточных архитектур

В этом разделе будут приведены примеры типичных сверточных архитектур. В качестве таковых выбраны сети, добившиеся успеха на соревнованиях ILSVRC в последние годы. Польза такого рассмотрения состоит в том, что оно позволит понять, какие факторы дизайна нейронной сети играют определяющую роль в обеспечении эффективной работы таких сетей. Несмотря на то что за последние годы в отношении архитектуры нейронных сетей был предложен ряд новых идей (в частности, ReLU-активация), остается лишь поражаться тому, насколько современные модели близки к базовой архитектуре сети *LeNet-5*. К числу основных изменений, отличающих современные архитектуры от *LeNet-5*, следует отнести резкое увеличение глубины нейронных сетей,

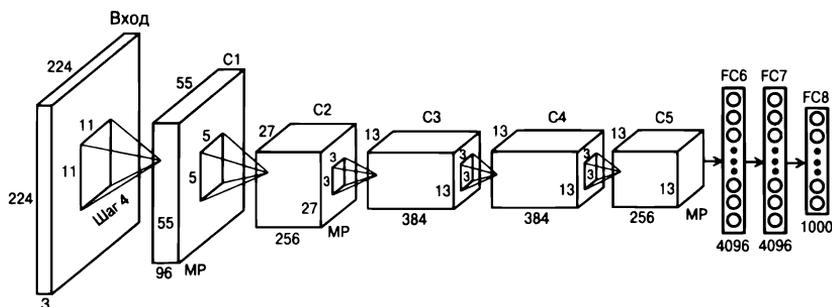
использование ReLU-активации, а также повышение эффективности обучения за счет возросших возможностей современного оборудования и усовершенствования алгоритмов оптимизации. Современные архитектуры значительно глубже и используют целый ряд вычислительных, архитектурных и аппаратных ухищрений, позволяющих тренировать сети на больших объемах данных. В этом отношении не стоит недооценивать значительный прогресс, достигнутый в разработке аппаратных средств. Современные платформы на основе GPU работают в 10 000 раз быстрее систем (примерно такой же стоимости), доступных в то время, когда была предложена сеть LeNet-5. Но даже на этих современных платформах для обучения сверточной нейронной сети, которая обеспечивала бы точность, делающую ее конкурентоспособной на соревнованиях ILSVRC, длительность тренировки может исчисляться неделями. Усовершенствования, касающиеся оборудования, доступности данных и алгоритмов, в определенной степени взаимосвязаны. Трудно испытывать новые алгоритмические трюки, если для того, чтобы проводить эксперименты со сложными/глубокими сетями за разумное время, не хватает данных или вычислительной мощности. Поэтому произошедшая в последнее время революция в области разработки глубоких сверточных сетей была бы невозможной без использования тех объемов данных и мощных вычислительных ресурсов, которые стали доступными в наши дни.

В следующих разделах представлен обзор хорошо известных моделей, которые часто применяются для проектирования обучающих алгоритмов, ориентированных на классификацию изображений. Некоторые из этих моделей доступны в формах, уже прошедших предварительное обучение на наборах данных ImageNet, так что результирующие признаки могут быть использованы в приложениях, не ограничивающихся только классификацией. Такой подход является разновидностью переносимого обучения, которое мы обсудим позже.

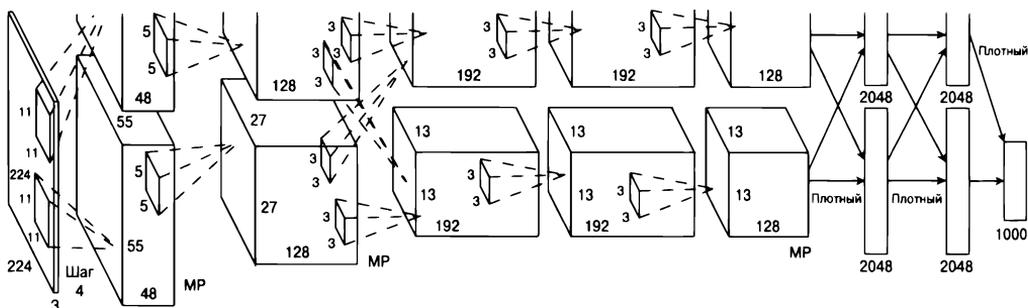
8.4.1. Сеть AlexNet

Сеть *AlexNet* стала победителем соревнований ILSVRC в 2012 году. Архитектура AlexNet представлена на рис. 8.9, *а*. Но оригинальная архитектура предполагала два параллельных процесса обработки, которые здесь не показаны. Эти два процесса выполнялись двумя GPU, работающими совместно для создания тренировочной модели с повышенной скоростью и разделением памяти. Первоначально сеть тренировалась с использованием GTX 580 с 3 Гбайт памяти, чего было недостаточно для проведения промежуточных вычислений. Поэтому сеть была распределена между двумя GPU. Оригинальная архитектура представлена на рис. 8.9, *б*, на котором сеть распределена между двумя GPU. Мы также привели вариант этой архитектуры без изменений, связанных с использованием GPU, чтобы сделать более очевидным сравнение с другими архитектурами сверточных нейронных сетей, которые обсуждаются в этой главе.

Следует отметить, что оба GPU взаимосвязаны лишь в подмножестве слоев, представленных на рис. 8.9, б, что приводит к некоторым различиям между рис. 8.9, а и б, в терминах фактически построенной модели. В частности, архитектура с GPU-разделением имеет меньше весов, поскольку не все слои связаны между собой. Исключение некоторых связей между ними уменьшает время обмена данными между процессорами и тем самым способствует повышению эффективности вычислений.



а) С GPU-разделением



б) Без GPU-разделения

Рис. 8.9. Архитектура AlexNet. За каждым слоем свертки следуют активации ReLU, которые не показаны. Слои пулинга по максимальному значению обозначены как MP; они следуют лишь за некоторым подмножеством комбинированных слоев “свертка — ReLU”. Архитектурная диаграмма (б) заимствована из [A. Krizhevsky, I. Sutskever, and G. Hinton. Imagenet classification with deep convolutional neural networks. NIPS Conference, p. 1097–1105. 2012.] ©2012 A. Krizhevsky, I. Sutskever, and G. Hinton

Работа сети AlexNet начинается с обработки изображений размером $224 \times 224 \times 3$ с использованием 96 фильтров размером $11 \times 11 \times 3$ в первом слое. Используется шаг 4. Это приводит к размеру первого слоя, равному $5 \times 55 \times 96$. Вслед за первым слоем используется слой пулинга по максимальному значению, обозначенный как ‘MP’ на рис. 8.9, а. Архитектура, приведенная на рис. 8.9, а, является упрощенной версией архитектуры, приведенной на рис. 8.9, б, на

которой в явном виде показаны два параллельных процесса обработки. Например, указанная на рис. 8.9, б, глубина первого сверточного слоя равна всего лишь 48, поскольку 96 карт признаков разделяются между GPU в целях распараллеливания вычислений. С другой стороны, на рис. 8.9, а, не предполагается использование GPU, поэтому указана ширина, равная 96. После каждого сверточного слоя применяется активационная функция ReLU, за которой следуют нормализация отклика и пулинг по максимальному значению. И хотя пулинг по максимальному значению показан на рисунке, в архитектуре он не выделен в виде отдельного блока. Кроме того, на схеме не показаны в явном виде слои ReLU и нормализации отклика. Компактные представления такого типа часто встречаются при описании нейронных архитектур.

Второй сверточный слой использует выход первого сверточного слоя, подвергнутый нормализации и пулингу, и фильтрует его с помощью 256 фильтров размером $5 \times 5 \times 96$. В третьем, четвертом и пятом сверточных слоях нормализация и пулинг не используются. Фильтры в этих сверточных слоях имеют размеры $3 \times 3 \times 256$ (384 фильтра), $3 \times 3 \times 384$ (384) и $3 \times 3 \times 384$ (256 фильтров) соответственно. Во всех слоях пулинга по максимальному значению задействуются фильтры размером 3×3 с шагом 2. Поэтому между пулами существует перекрытие. Полносвязные слои содержат 4096 нейронов. Последний набор из 4096 активаций можно рассматривать как 4096-мерное представление изображения. В последнем слое сети AlexNet в целях классификации используются 1000 выходов с Softmax-активацией. Следует отметить, что последний слой с 4096 активациями (обозначен как FC7 на рис. 8.9, а) часто применяется для создания 4096-мерного представления изображения для приложений, ориентированных не только на классификацию объектов. Эти признаки можно извлечь для любого не входящего в тренировочный набор изображения, пропустив его через обученную нейронную сеть. Такие признаки часто хорошо обобщаются на другие наборы и другие задачи и носят название *FC7-признаков*. Фактически использование признаков, извлеченных из предпоследнего слоя FC7, обрело популярность после появления сети AlexNet, хотя сам подход был известен задолго до этого. Как следствие, подобные признаки, извлеченные из предпоследнего слоя любой сверточной нейронной сети, часто называют признаками FC7, независимо от имеющихся в данной сети слоев. Необходимо отметить, что количество карт признаков в промежуточных слоях намного превышает начальную глубину объема во входном слое (которая равна всего лишь 3 в соответствии с количеством RGB-цветов), хотя их пространственные размеры меньше. Это объясняется тем, что начальная глубина содержит лишь цветовые RGB-компоненты, тогда как последующие слои захватывают в карты признаков семантические признаки различных типов.

Многие проектные решения, примененные в этой архитектуре, стали стандартом для более поздних архитектур. Конкретным примером может служить использование в архитектуре активации ReLU (вместо сигмоиды или гиперболического тангенса). В наши дни выбор функции активации в большинстве сверточных нейронных сетей фокусируется почти исключительно на ReLU, хотя до появления сети AlexNet ситуация была иная. Некоторые трюки обучения уже были известны в то время, но использование их в AlexNet принесло им широкую популярность. Примером этого может служить аугментация данных, оказавшаяся весьма полезной в отношении улучшения точности. Кроме того, сеть AlexNet продемонстрировала важность специализированного оборудования наподобие GPU для обучения на больших наборах данных. С целью улучшения обобщающей способности сети был использован метод исключения (дропаут) с L_2 -регуляризацией. В наши дни метод исключения применяется практически во всех типах архитектур, поскольку в большинстве случаев он служит дополнительным средством повышения эффективности сети. В последующих архитектурах от использования нормализации локального отклика в конечном счете отказались.

Рассмотрим вкратце варианты выбора параметров, применяемых в сети AlexNet. Заинтересованный читатель найдет полный код и файлы параметров сети AlexNet в [584]. L_2 -регуляризация использовалась с параметром 5×10^{-4} . Метод *исключений* был реализован на основе семплирования элементов с вероятностью 0,5. Для тренировки сети AlexNet применялся импульсный (мини-пакетный) стохастический градиентный спуск с параметром, равным 0,8. Размер пакета составлял 128. Скорость обучения была принята равной 0,01, хотя по мере достижения сходимости она пару раз уменьшалась. Даже с GPU для обучения сети AlexNet потребовалась примерно неделя.

Итоговый коэффициент ошибок топ-5, который определялся как доля случаев, когда корректное изображение не входило в топ-5 изображений, составил примерно 15,4%. Можете сравнить это значение³ с результатами предыдущих победителей соревнований, которым удалось снизить коэффициент ошибки всего лишь до 25%. Аналогичную величину имеет и разрыв между вторыми лучшими результатами. Использование одной сверточной сети обеспечило получение коэффициента ошибок топ-5, равного 18,2%, тогда как использование ансамбля из семи моделей позволило снизить коэффициент до рекордного уровня 15,4%. Отметим, что трюк с ансамблем обеспечивает систематическое улучшение результатов на 2-3%. К тому же, поскольку большинство ансамблевых

³ Говорить о коэффициенте ошибок топ-5 имеет смысл применительно к изображениям, в которых одно изображение может содержать объекты нескольких классов. На протяжении главы мы будем употреблять термин “коэффициент ошибок”, имея в виду ошибки топ-5.

методов на удивление хорошо приспособлено к параллелизму, при наличии достаточных аппаратных ресурсов реализация таких методов не вызывает особых трудностей. Ввиду столь большого отрыва от соперников на соревнованиях ILSVRC сеть AlexNet относится к разряду фундаментальных достижений в области компьютерного зрения. Ее успех возродил интерес к глубокому обучению вообще и к сверточным нейронным сетям в частности.

8.4.2. Сеть ZFNet

Победителем соревнований ILSVRC в 2013 году стал вариант сети ZFNet [556]. Ее архитектура в значительной мере базируется на сети AlexNet, хотя для дальнейшего повышения точности в нее был внесен ряд изменений. Большинство из них связано с различиями в выборе гиперпараметров, поэтому на фундаментальном уровне сеть ZFNet не слишком отличается от сети AlexNet. Одно из изменений заключалось в том, что начальные фильтры размером $11 \times 11 \times 3$ были заменены фильтрами размером $7 \times 7 \times 3$. Вместо шага 4 был использован шаг 2. Во втором слое использовались фильтры размером 5×5 с шагом, также равным 2. Как и сеть AlexNet, сеть ZFNet содержит три слоя пулинга по максимальному значению с теми же размерами фильтров пулинга. В то же время первая пара слоев пулинга по максимальному значению выполнялась после первой и второй сверток (а не второй и третьей). В результате пространственные размеры третьего слоя стали равными 13×13 , а не 27×27 , хотя все остальные пространственные размеры не изменились по сравнению с сетью AlexNet. Размеры различных слоев сетей AlexNet и ZFNet приведены в табл. 8.1.

Таблица 8.1. Сравнение сетей AlexNet и ZFNet

	AlexNet	ZFNet
Объем:	$224 \times 224 \times 3$	$224 \times 224 \times 3$
Операции:	Свертка 11×11 (шаг 4)	Свертка 7×7 (шаг 2), МР
Объем:	$55 \times 55 \times 96$	$55 \times 55 \times 96$
Операции:	Свертка 5×5 , МР	Свертка 5×5 (шаг 2), МР
Объем:	$27 \times 27 \times 256$	$13 \times 13 \times 256$
Операции:	Свертка 3×3 , МР	Свертка 3×3
Объем:	$13 \times 13 \times 384$	$13 \times 13 \times 512$
Операции:	Свертка 3×3	Свертка 3×3
Объем:	$13 \times 13 \times 384$	$13 \times 13 \times 1024$
Операции:	Свертка 3×3	Свертка 3×3
Объем:	$13 \times 13 \times 256$	$13 \times 13 \times 512$
Операции:	МР, полносвязный	МР, полносвязный

Окончание табл. 8.1

	AlexNet	ZFNet
FC6:	4096	4096
Операции:	Полносвязный	Полносвязный
FC7:	4096	4096
Операции:	Полносвязный	Полносвязный
FC8:	1000	1000
Операции:	Softmax	Softmax

В третьем, четвертом и пятом сверточных слоях сети ZFNet используется большее количество слоев, чем в сети AlexNet. Количество фильтров в этих слоях было изменено с (384, 384, 256) до (512, 1024, 512). В результате пространственные размеры большинства слоев в сетях AlexNet и ZFNet одинаковы, хотя глубины в последних трех сверточных слоях, имеющих аналогичные пространственные размеры, отличаются. С точки зрения общей перспективы в сети ZFNet были использованы те же принципы, что и в сети AlexNet, и основной выигрыш был получен за счет изменения архитектурных параметров AlexNet. Эта архитектура позволила снизить коэффициент ошибок до уровня 14,8% вместо прежних 15,4%, а дальнейшее увеличение ширины/глубины снизило ошибку до 11,1%. Поскольку большинство различий между сетями AlexNet и ZFNet относилось к категории незначительных проектных изменений, это подчеркивает тот факт, что при работе с алгоритмами глубокого обучения небольшие детали играют очень важную роль. Таким образом, для достижения наилучшей производительности иногда важно проводить интенсивные эксперименты над нейронными архитектурами. Архитектура ZFNet была расширена и углублена, и на соревнованиях ILSVRC, проводившихся в 2013 году, сеть была представлена как *Clarifai* — по названию компании⁴, учрежденной первым из авторов работы [556]. Различие⁵ между сетями Clarifai и ZFNet касалось изменения ширины/глубины сети, хотя точные детали различий неизвестны. Эта сеть стала победителем соревнований ILSVRC в 2013 году. Более подробная информация об этой сети вместе с графической иллюстрацией ее архитектуры приведена в [556].

8.4.3. Сеть VGG

Появление сети *VGG* [454] еще более усилило наметившийся среди разработчиков тренд к увеличению глубины сетей. Тестировались различные конфигурации сетей размером от 11 до 19 слоев, хотя наилучшие результаты про-

⁴ <http://www.clarifai.com>.

⁵ Личное сообщение Мэттью Зайлера.

демонстрировали сети, содержащие не менее 16 слоев. Сеть VGG была одной из лучших на соревнованиях ISLVRС в 2014, но не победителем. Им стала сеть *GoogLeNet* с коэффициентом ошибки 6,7% по сравнению с 7,3% сети VGG. Тем не менее сеть VGG сыграла важную роль, так как продемонстрировала действенность нескольких важных принципов проектирования, которые в конечном счете стали стандартом в последующих архитектурах.

Существенной инновацией VGG является то, что в ней были уменьшены размеры фильтров, но увеличена глубина. Важно понимать, что снижение размера фильтра вынуждает увеличивать глубину. Это объясняется тем, что если сеть не является глубокой, то небольшой фильтр в состоянии захватить лишь небольшую область изображения. Например, одиночный признак, являющийся результатом трех последовательных сверток размером 3×3 , захватит область входа размером 7×7 . Отметим, что применение одного фильтра размером 7×7 непосредственно к входным данным также захватит визуальные свойства области входа размером 7×7 . В первом случае мы используем $3 \times 3 \times 3 = 27$ параметров, во втором — $7 \times 7 \times 1 = 49$ параметров. Поэтому в случае использования трех последовательных сверток количество параметров оказывается меньшим. В то же время три последовательные свертки часто способны захватить больше интересных сложных признаков, чем одиночная свертка, и результирующие активации, полученные с помощью единственной свертки, будут представлять собой примитивные признаки в виде отрезков линий. Поэтому сеть 7×7 будет неспособна захватывать сложные формы в небольших областях.

В общем случае увеличение глубины приводит к большей нелинейности и более сильной регуляризации. Глубокая сеть будет характеризоваться большей нелинейностью в силу наличия большего количества слоев ReLU, а также большей регуляризацией, поскольку увеличенная глубина навязывает структуру слоев через повторение композиции сверток. Как обсуждалось выше, архитектуры с большей глубиной и уменьшенным размером фильтра требуют меньшего количества параметров. Частично это обусловлено тем, что количество параметров в каждом слое зависит от квадрата размера фильтра, тогда как общее количество параметров линейно зависит от глубины. Поэтому, вместо того чтобы “тратить” параметры на увеличение глубины, можно существенно уменьшить их количество, используя фильтры небольшого размера. Кроме того, увеличенная глубина позволяет использовать большее количество нелинейных активаций, что увеличивает дифференцируемую способность модели. Поэтому в VGG всегда используются фильтры с пространственными размерами 3×3 и размерами пулинга 2×2 . Свертка осуществлялась с шагом 1 и дополнением 1. Пулинг выполнялся с шагом 2. Использование фильтра размером 3×3 с дополнением 1 сохраняет пространственные размеры выходного объема, хотя пулинг всегда сжимает пространственные размеры. Поэтому

пулинг (в отличие от двух предыдущих архитектур) выполнялся на неперекрывающихся пространственных областях и всегда снижал пространственные размеры (как высоту, так и ширину) в два раза. Еще одним интересным проектным решением VGG было то, что количество фильтров часто увеличивалось вдвое после каждой операции пулинга по максимальному значению. Идея состояла в том, чтобы увеличивать глубину вдвое всякий раз, когда пространственные размеры уменьшались в два раза. Такой выбор параметров позволяет немного сбалансировать интенсивность вычислений в различных слоях, и это решение было унаследовано некоторыми более поздними архитектурами, такими как *ResNet*.

Одной из проблем, связанных с использованием глубоких конфигураций, было то, что увеличение глубины приводило к большей чувствительности к инициализации, что, как известно, вызывает нестабильность. Эта проблема была решена за счет использования предварительного обучения, при котором сначала тренировалась более мелкая архитектура, после чего добавлялись дополнительные слои. В то же время предварительное обучение выполнялось не послойно. Вместо этого сначала обучалось 11-слойное подмножество архитектуры, а затем предобученные слои использовались для инициализации подмножества слоев более глубокой архитектуры. На соревнованиях ISLVRС сеть VGG продемонстрировала коэффициент ошибки, равный всего лишь 7,3%, что было одним из наилучших результатов, хотя он не принес VGG победу. Различные конфигурации VGG приведены в табл. 8.2. Из них архитектурой-победителем стала архитектура, указанная в столбце D. Обратите внимание на то, что после каждой операции пулинга по максимальному значению количество фильтров увеличивается в два раза. Поэтому пулинг приводит к уменьшению пространственной высоты и ширины в два раза, что компенсируется увеличением глубины в два раза. Выполнение сверток с фильтрами размером 3×3 и дополнением 1 не изменяет пространственные размеры. Следовательно, размеры вдоль каждого из пространственных измерений (т.е. высота и ширина) в областях между различными операциями пулинга по максимальному значению, указанными в столбце D табл. 8.2, составляют 224, 112, 56, 28 и 14 соответственно. Последняя операция пулинга по максимальному значению выполняется непосредственно перед созданием полносвязного слоя, который дополнительно снижает пространственный размер до 7. Поэтому первый полносвязный слой характеризуется плотными соединениями между 4096 нейронами и имеет объем $7 \times 7 \times 512$. Как будет показано далее, большинство параметров нейронной сети скрыто в этих соединениях.

Таблица 8.2. Конфигурации, используемые в сети VGG

Название	A	A-LRN	B	C	D	E
Кол-во слоев	11	11	13	16	16	19
	C3D64	C3D64	C3D64	C3D64	C3D64	C3D64
		LRN	C3D64	C3D64	C3D64	C3D64
	M	M	M	M	M	M
	C3D128	C3D128	C3D128	C3D128	C3D128	C3D128
			C3D128	C3D128	C3D128	C3D128
	M	M	M	M	M	M
	C3D256	C3D256	C3D256	C3D256	C3D256	C3D256
	C3D256	C3D256	C3D256	C3D256	C3D256	C3D256
				C1D256	C3D256	C3D256
						C3D256
	M	M	M	M	M	M
	C3D512	C3D512	C3D512	C3D512	C3D512	C3D512
	C3D512	C3D512	C3D512	C3D512	C3D512	C3D512
				C1D512	C3D512	C3D512
						C3D512
	M	M	M	M	M	M
	C3D512	C3D512	C3D512	C3D512	C3D512	C3D512
	C3D512	C3D512	C3D512	C3D512	C3D512	C3D512
				C1D512	C3D512	C3D512
						C3D512
	M	M	M	M	M	M
	FC4096	FC4096	FC4096	FC4096	FC4096	FC4096
	FC4096	FC4096	FC4096	FC4096	FC4096	FC4096
	FC1000	FC1000	FC1000	FC1000	FC1000	FC1000
	S	S	S	S	S	S

Примечание: C3D64 обозначает сверточные слои, в которых свертки выполняются с использованием 64 фильтров с пространственными размерами 3×3 (иногда 1×1). Глубина фильтра согласуется с соответствующим слоем. Дополнение каждого фильтра выбирается из тех соображений, чтобы сохранялись пространственные размеры слоя. За всеми свертками следует слой ReLU. Другие обозначения: M — слой пулинга по максимальному значению; LRN — нормализация локального отклика; S — слой Softmax; FC4096 — полносвязный слой с 4096 элементами. Во всех слоях, кроме последнего набора слоев, количество фильтров всегда увеличивается после каждой операции пулинга по максимальному значению. Поэтому уменьшение пространственных размеров часто сопровождается увеличением глубины.

В [236] были продемонстрированы интересные результаты относительно того, на какие именно части сети приходится большинство параметров и память активаций. В частности, значительная часть *памяти* для хранения активаций и градиентов в фазах прямого и обратного распространения требуется в ранней части сверточной нейронной сети, имеющей наибольшие пространственные размеры. Этот момент очень важен, поскольку память, необходимая мини-пакету, масштабируется его размером. Например, в [236] было показано, что для каждого изображения требуется примерно 93 Мбайт. Поэтому для мини-пакета размером 128 необходимый общий объем памяти составил бы примерно 12 Гбайт. Несмотря на то что ранние слои, имеющие наибольшие пространственные размеры, требуют наибольшего объема памяти, количество параметров в них невелико из-за разреженности соединений и разделения весов. Фактически наибольшее количество параметров требуется для полносвязных слоев в оконечной части сети. Для соединения последнего пространственного слоя с размерами $7 \times 7 \times 512$ (см. столбец D в табл. 8.2) с 4096 нейронами потребовалось $7 \times 7 \times 512 \times 4096 = 102\,760\,448$ параметров. Общее количество параметров во всех слоях составило примерно 138 000 000. Поэтому *на один слой соединений приходится примерно 75% параметров*. Кроме того, большинство оставшихся параметров приходится на последние два полносвязных слоя. В итоге на плотные соединения приходится около 90% всех параметров нейронной сети. Это очень важный вывод, поскольку сеть *GoogLeNet* использует ряд инноваций для снижения количества параметров в последних слоях.

Примечательно то, что некоторые архитектуры допускают свертки 1×1 . Несмотря на то что свертки 1×1 не комбинируют активации смежных пространственных признаков, они комбинируют значения признаков различных каналов, если глубина объема превышает 1. Использование сверток 1×1 также является одним из способов внедрения дополнительной нелинейности в архитектуру, не требующей внесения фундаментальных изменений на пространственном уровне. Эта дополнительная нелинейность включается посредством присоединения ReLU-активаций к каждому слою (более подробную информацию см. в [454]).

8.4.4. Сеть GoogLeNet

В сети *GoogLeNet*, также известной как *Inception-v1*, была реализована новая концепция: архитектура *Inception*. Архитектура этого типа представляет собой *сеть в сети*. Ее начальная часть во многом напоминает традиционную сверточную сеть и называется *основой* (stem). Ключевой частью сети является промежуточный слой, который назвали *модулем Inception* (рис. 8.10, а). Базовая идея заключается в том, что ключевая информация в изображениях доступна на различных уровнях детализации. Использование крупного фильтра позволяет захватывать информацию в крупной области, содержащей ограниченную

вариацию. Если же мы используем фильтр небольшого размера, то можем захватывать детальную информацию в небольшой области. Несмотря на то что одним из решений могло бы стать последовательное соединение многих небольших фильтров, это привело бы к напрасному раздутию количества параметров и глубины, когда достаточно было бы использовать более широкие образы в более крупных областях. Проблема в том, что нам не известно заранее, какой уровень детализации требуется для той или иной области изображения. Почему бы не наделить нейронную сеть гибкостью, позволяющей ей моделировать изображение на различных уровнях гранулярности? Эта цель достигается с помощью модуля Inception, который выполняет свертку, применяя параллельно три различных фильтра размером 1×1 , 3×3 и 5×5 . Использование исключительно последовательной цепочки фильтров с одними и теми же размерами неэффективно при работе с объектами различного масштаба в различных изображениях. Поскольку все фильтры в слое Inception являются обучаемыми, и нейронная сеть может принимать решение относительно того, какой из них должен влиять на выход в наибольшей степени. Выбирая фильтры различного размера вдоль различных путей, можно представить различные области на различном уровне гранулярности. Сеть GoogLeNet состоит из девяти модулей Inception, расположенных последовательно. Это позволяет выбирать множество альтернативных путей прохождения через эту архитектуру, так что результирующие признаки будут представлять совершенно разные пространственные области. Например, прохождение через четыре фильтра 3×3 , за которыми следуют лишь фильтры 1×1 , обеспечит захват относительно небольшой пространственной области. С другой стороны, прохождение через множество фильтров 5×5 приведет к намного большим пространственным размерам. Иными словами, различия в масштабе форм, захваченных различными скрытыми признаками, будут усиливаться в поздних слоях. В последние годы в сочетании с архитектурой Inception применялась пакетная нормализация, что упрощает⁶ структуру сети по сравнению с ее первоначальной формой.

Следует заметить, что из-за использования большого количества сверток различного размера модуль Inception немного снижает эффективность вычислений. В связи с этим на рис. 8.10, б, представлена более эффективная реализация, в которой для первоначального снижения глубины карты признаков используются свертки 1×1 . Это объясняется тем, что количество сверточных фильтров 1×1 в умеренное количество раз меньше глубины входного объема. Например, глубину 256 входного объема сначала можно уменьшить до 64, используя 64 различных фильтра 1×1 . Эти дополнительные свертки 1×1

⁶ Оригинальная архитектура включала также вспомогательные классификаторы, которые в последние годы игнорируются.

называют операциями “бутылочного горлышка” (bottleneck operations), т.е. *ограничительными операциями* модуля Inception. Изначальное уменьшение глубины карты признаков (с помощью сверток 1×1) обеспечивает эффективность вычислений с более крупными свертками в силу снижения глубины слоев после применения ограничительных сверток. Свертки 1×1 можно рассматривать как разновидность контролируемого снижения размерности до применения пространственных фильтров большего размера. Такое снижение размерности называют контролируемым, поскольку параметры критических фильтров обучаются в процессе обратного распространения ошибки. Кроме того, ограничительные свертки способствуют уменьшению глубины после слоя пулинга. Трюк с ограничительными слоями используется и в ряде других архитектур для улучшения эффективности и выходной глубины.

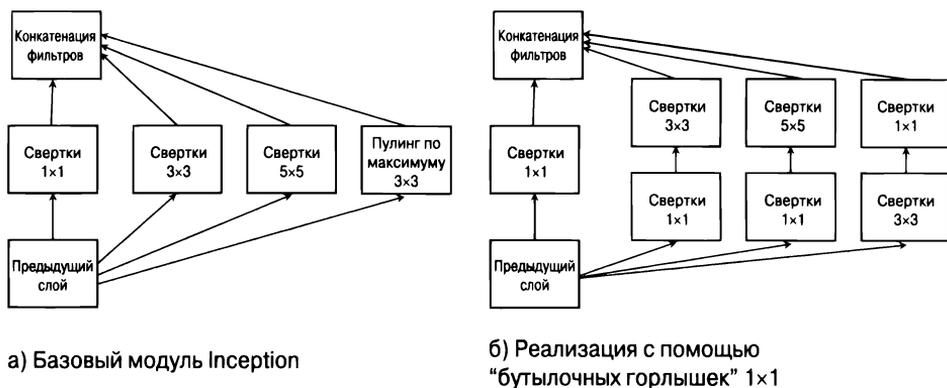


Рис. 8.10. Модуль Inception сети GoogLeNet

Выходной слой GoogLeNet также иллюстрирует некоторые интересные принципы проектирования сетей. Обычно вблизи выхода используют полносвязные слои. Однако в GoogLeNet по всей пространственной области последнего набора активационных карт применяется пулинг по среднему значению для создания одного значения. Поэтому количество признаков, созданных в последнем слое, будет в точности равно количеству фильтров. Важно то, что большинство параметров относится к соединениям между последним сверточным и первым полносвязным слоями. Подобный тип детализации свойств соединений не требуется для приложений, в которых должны предсказываться только метки классов. В связи с этим применяется подход на основе пулинга по среднему значению. Однако такое представление приводит к полной потере всей пространственной информации, о чем нельзя забывать в случае некоторых типов приложений. Важным свойством сети GoogLeNet была ее предельная компактность в отношении количества параметров, которое на порядок меньше по сравнению с сетью VGG. В основном это обусловлено

использованием пулинга по среднему значению, который в конечном счете стал стандартом во многих более поздних архитектурах. С другой стороны, общая архитектура GoogLeNet характеризуется более высокими накладными вычислительными расходами.

Своей гибкостью сеть GoogLeNet обязана 22-слойной архитектуре Inception, в которой объекты различного масштаба обрабатываются фильтрами соответствующих размеров. Эта гибкость мультигранулярной декомпозиции, ставшая возможной благодаря модулям Inception, была одним из определяющих факторов производительности сети. Кроме того, замена полносвязного слоя пулингом по среднему значению позволила значительно уменьшить количество параметров. Данная архитектура принесла GoogLeNet победу на соревнованиях ILSVRC в 2014 году, тогда как VGG немного уступила ей, заняв второе место. Но, несмотря на то что GoogLeNet обошла VGG, преимуществом последней является простота, которая порой высоко ценится на практике. Обе архитектуры продемонстрировали важные принципы проектирования сверточных нейронных сетей. С тех пор архитектура Inception оказалась в центре внимания исследователей [486, 487], и были предложены многочисленные изменения, улучшающие ее производительность. В результате сочетания разработанной в последние годы версии этой архитектуры, известной под названием *Inception-v4* [487], с некоторыми идеями, почерпнутыми из сети *ResNet* (см. следующий раздел), была создана 75-слойная архитектура, обеспечившая коэффициент ошибки на уровне всего лишь 3,08%.

8.4.5. Сеть ResNet

В сети *ResNet* [184] было задействовано 152 слоя, что почти на порядок превышало количество слоев, используемых до этого другими архитектурами. Данная архитектура завоевала первый приз на соревнованиях ILSVRC в 2015 году с коэффициентом ошибок топ-5 3,6%, тем самым став первым классификатором, которому удалось подняться до уровня человеческих возможностей. Такая точность была достигнута за счет использования ансамбля сетей *ResNet*, но даже одна модель обеспечила точность 4,5%. Без внедрения некоторых важных инноваций обучить такую архитектуру, насчитывающую 152 слоя, было бы просто невозможно.

Основной проблемой при обучении столь глубоких сетей является то, что поток градиента между слоями тормозится большим количеством операций в глубоких слоях, которые могут увеличивать или уменьшать размеры градиентов. Как обсуждалось в главе 3, увеличение глубины порождает проблемы затухающих и взрывных градиентов. Однако в [184] было выдвинуто предположение о том, что не этот фактор является основным источником проблем, особенно если используется пакетная нормализация. Основной проблемой является трудность

достижения сходимости процесса обучения в течение разумных промежутков времени. С подобными проблемами сходимости приходится часто сталкиваться в сетях со сложными поверхностями функции потерь. И хотя некоторые глубокие сети демонстрируют большой разрыв между ошибкой обучения и ошибкой тестирования, во многих глубоких сетях ошибка оказывается большой как на тренировочных, так и на тестовых данных. Это является следствием недостаточной скорости процесса оптимизации.

Несмотря на то что иерархическое конструирование признаков является Святым Граалем обучения с помощью нейронных сетей, его послойные реализации вынуждают применять один и тот же уровень абстракции по отношению ко всем элементам изображения. Одним элементам можно обучиться, используя мелкие сети, тогда как другие требуют использования мелкогранулярных соединений. Рассмотрим в качестве примера изображение слона, стоящего на квадратной платформе. Для конструирования характерных признаков слона может потребоваться большое количество слоев, тогда как для конструирования признаков квадратной платформы — совсем небольшое количество. Сходимость будет неоправданно замедленной, если применять очень глубокую сеть с фиксированной глубиной для всех путей, по которым осуществляется обучение признакам, для многих из которых было бы достаточно использовать мелкие архитектуры. Почему бы не позволить нейронной сети самой решать, сколько слоев нужно задействовать для обучения каждому признаку?

Сеть ResNet использует *замыкающие соединения*, или *соединения с пропуском слоев* (skip connections), чтобы обеспечить возможность копирования информации между слоями, и вводит *итеративный подход* к конструированию признаков (вместо иерархического). В сетях на основе *долгой краткосрочной памяти* (long short-term memory network — LSTM) и *управляемых рекуррентных блоков* (gated recurrent units) аналогичные принципы используются в отношении последовательных данных, разрешая копирование частей состояний из одного слоя в следующий с помощью регулируемых *вентилей*. В случае ResNet предполагается, что несуществующие “вентили” всегда полностью открыты. Многие сети прямого распространения содержат лишь соединения между слоями i и $(i + 1)$, тогда как ResNet содержит соединения между слоями i и $(i + r)$ для $r > 1$. Пример такого замыкающего соединения, которое образует базовый блок ResNet, приведен на рис. 8.11, *a*, для случая $r = 2$. Соединение копирует вход слоя i и прибавляет его к выходу слоя $(i + r)$. Данный подход обеспечивает эффективное перетекание градиента, поскольку теперь алгоритм обратного распространения ошибки обеспечивает беспрепятственное распространение градиентов, используя замыкающие соединения. Подобный базовый блок называют *остаточным модулем* (residual module), и сеть в целом создается путем объединения базовых модулей. В большинстве слоев используются фильтры с

соответствующим дополнением⁷ и шагом 1, поэтому пространственные размеры и глубина входа не изменяются от слоя к слою. В подобных случаях простое сложение входов слоев i и $(i + r)$ осуществляется очень легко. Однако в некоторых слоях для понижения каждого пространственного размера в два раза используются шаговые свертки. В то же время глубина увеличивается в два раза за счет применения большего количества фильтров. В таких случаях поверх замыкающих соединений нельзя использовать тождественную функцию. В связи с этим поверх пропускающего соединения должна применяться линейная проекционная матрица, согласующая размерности. Проекционная матрица определяет набор сверточных операций 1×1 с шагом 2 для уменьшения пространственных размеров вдвое. Параметры проекционной матрицы должны обучаться в процессе обратного распространения ошибки.

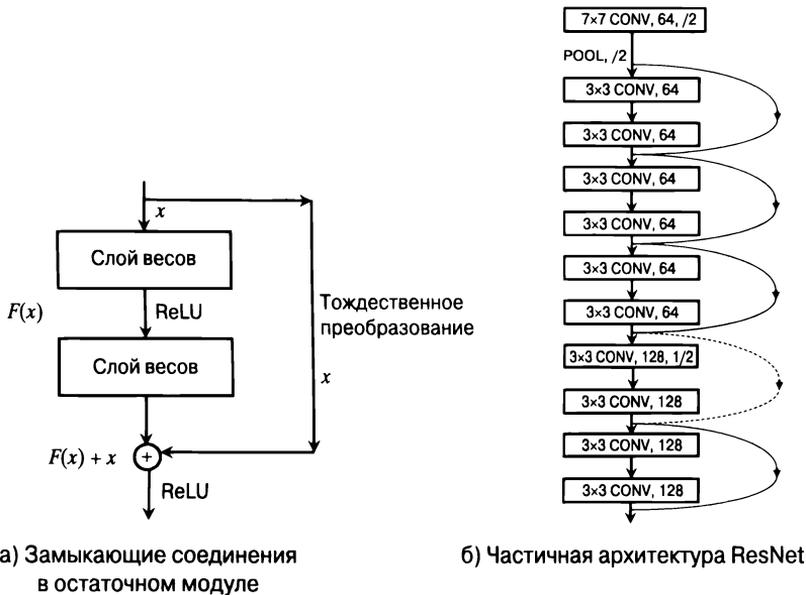


Рис. 8.11. Остаточный модуль и первые несколько слоев сети ResNet

В соответствии с оригинальной идеей ResNet суммироваться должны лишь входы слоев i и $(i + r)$. Например, при $r = 2$ соединения со всеми нечетными слоями пропускаются. Более поздние усовершенствованные варианты наподобие DenseNet продемонстрировали улучшенную производительность за счет добавления соединений между всеми парами слоев. Базовый блок, представленный на рис. 8.11, а, многократно повторяется в ResNet, что позволяет многократно использовать замыкающие соединения для распространения входа к

⁷ В типичных случаях используется фильтр 3×3 при шаге/дополнении, равном 1. Эта тенденция наметилась после появления VGG и была адаптирована в ResNet.

выходу напрямую, без выполнения каких-либо вычислений, связанных с прямым распространением. В качестве примера на рис. 8.11, б, приведены первые несколько слоев этой архитектуры. В основу данного примера положены первые несколько слоев 34-слойной архитектуры. Замыкающие соединения, составляющие большинство соединений, обозначены на рис. 8.11, б, сплошными линиями, которые соответствуют применению тождественной функции при неизменном объеме фильтра. Однако в некоторых слоях используется шаг 2, что приводит к изменению пространственных размеров и глубины. В этих слоях должна применяться проекционная матрица, чему на рисунке соответствует замыкающее соединение, обозначенное пунктирной линией. В оригинальной работе [184] тестировались четыре различные архитектуры, содержащие 34, 50, 101 и 152 слоя соответственно. Наивысшую производительность продемонстрировала 152-слойная архитектура, но даже 34-слойная архитектура обеспечила результаты, которые превосходили наилучшие результаты соревнований ILSVRC предыдущего года.

Использование замыкающих соединений обеспечивает пути для беспрепятственного прохождения потока градиента, что имеет важные последствия, определяющие поведение алгоритма обратного распространения ошибки. Замыкающие соединения играют роль магистралей для распространения потока градиента, создавая ситуацию, в которой существует множество путей переменной длины, ведущих от входа к выходу. В подобных случаях кратчайшие пути обеспечивают наиболее быстрое обучение, тогда как длинные пути можно рассматривать как остаточные вклады. Благодаря этому алгоритм обучения обретает гибкость в выборе подходящего уровня нелинейности для конкретного входа. Входы, которые можно классифицировать с использованием небольшой доли нелинейности, будут миновать многие соединения. Другие входы с более сложной структурой могут проходить через большее количество соединений для извлечения соответствующих признаков. Поэтому данный подход называют *остаточным обучением* (residual learning), при котором обучение вдоль длинных путей является своеобразной тонкой настройкой более легкого обучения, осуществляемого вдоль коротких путей. Иными словами, такой подход хорошо подходит для ситуаций, в которых различные аспекты изображения характеризуются различными уровнями сложности. В [184] показано, что интенсивность остаточных откликов от более глубоких слоев относительно невелика, что подтверждает справедливость интуитивных соображений относительно того, что фиксированная глубина создает препятствия для нормального обучения. В подобных случаях сходимость часто не является проблемой, поскольку более короткие пути способствуют увеличению доли обучения за счет беспрепятственного распространения потоков градиента. В [505] был сделан интересный вывод о том, что сеть ResNet ведет себя подобно ансамблю мелких сетей,

поскольку этот тип архитектуры активизирует множество альтернативных путей короткой протяженности. Лишь небольшая доля обучения осуществляется вдоль более глубоких путей и только в тех случаях, когда это действительно необходимо. Фактически в [505] представлена иллюстративная картина развернутой архитектуры ResNet, на которой различные пути показаны в явном виде как параллельные процессы. Такое развернутое представление обеспечивает ясное понимание причин того, почему в реализации ResNet просматривается сходство с принципами организации сетей в виде ансамблей. Следствием такой точки зрения является тот факт, что исключение некоторых слоев из обученной сети ResNet на стадии предсказания не сопровождается деградацией точности в той мере, в какой это происходит в других сетях, таких как VGG.

Более подробно об этом можно прочитать в [549], посвященной *широким остаточным сетям* (wide residual networks). В этой работе выдвинуто предположение о том, что увеличение глубины остаточной сети не всегда дает положительный эффект, поскольку предельно глубокие пути могут вообще не использоваться. Замыкающие соединения образуют альтернативные пути и эффективно увеличивают ширину сети. В [549] высказано предположение о том, что можно получить лучшие результаты, ограничивая до некоторой степени общее количество слоев (скажем, 50 вместо 150) и используя увеличенное количество фильтров в каждом слое. Отметим, что по стандартам, предшествующим сети ResNet, глубина 50 все еще довольно велика, но в то же время мала по сравнению с глубиной, использованной в недавних экспериментах с остаточными сетями. Такой подход создает благоприятные условия для распараллеливания операций.

Вариации архитектур с пропуском слоев

С тех пор как была предложена архитектура ResNet, было высказано несколько рекомендаций относительно дальнейшего улучшения ее производительности. Например, в независимо предложенных *магистральных сетях* (highway networks) [161] было введено понятие *управляемых (вентильных) замыкающих соединений* (gated skip connections), и такие сети можно рассматривать как более общий вариант архитектуры. В магистральных сетях вместо тождественных отображений используются *вентили*, хотя закрытый клапан не пропускает значительный объем информации. В подобных случаях вентильные сети ведут себя иначе, нежели остаточные сети. В то же время остаточные сети можно рассматривать как частный случай вентильных сетей, в которых клапаны всегда полностью открыты. Магистральные сети тесно связаны как с сетями LSTM, так и с сетями ResNet, хотя, по-видимому, последние демонстрируют более высокую производительность при распознавании изображений, поскольку они сфокусированы на предоставлении множества путей для

распространения градиента. В оригинальной архитектуре ResNet используется простой блок слоев между замыкающими соединениями. С другой стороны, архитектура *ResNext*, базирующаяся на этом принципе, вносит определенные вариации за счет использования модулей Insertion между замыкающими соединениями [537].

Вместо замыкающих соединений можно выполнять сверточные преобразования между каждой парой слоев [211]. Таким образом, вместо выполнения L преобразований в сети прямого распространения с L слоями можно использовать $L(L - 1)/2$ преобразований. Иными словами, l -й слой использует конкатенацию всех карт признаков предыдущих $(l - 1)$ слоев. Эта архитектура получила название *DenseNet*. Она решает задачи, аналогичные задачам замыкающих соединений, разрешая каждому слою обучаться на наиболее выгодном уровне абстракции.

Интересным вариантом, который, по-видимому, работает неплохо, является использование *стохастической глубины* [210]. В этом варианте некоторые блоки между замыкающими соединениями отбрасываются случайным образом в процессе тренировки, но на стадии тестирования используется целиком вся сеть. Данный подход напоминает метод исключения (дропаут), в котором за счет исключения узлов уменьшают ширину сети, а не глубину. Однако метод исключения несколько отличается от послойного отбрасывания узлов, поскольку последний в большей степени сфокусирован на улучшении условий распространения потока, а не на предотвращении коадаптации признаков.

8.4.6. Эффекты глубины

Значительные успехи в повышении производительности, продемонстрированные за последние годы на соревнованиях ILSVRC, стали возможными главным образом благодаря возросшей вычислительной мощности оборудования, увеличению объемов доступных данных и архитектурным изменениям, которые сделали возможным эффективное обучение нейронных сетей увеличенной глубины. Эти три аспекта взаимосвязаны, поскольку выполнение экспериментов с улучшенными архитектурами возможно лишь при наличии доступа к достаточным объемам данных и высокопроизводительному оборудованию. Это также является одной из причин того, почему тонкая настройка и использование всевозможных ухищрений в относительно старых архитектурах (наподобие рекуррентных нейронных сетей) с присущими им проблемами не могли быть реализованы до недавнего времени.

В табл. 8.3 приведены данные о количестве слоев и уровнях ошибки, обеспечиваемых различными сетями. Быстрое повышение точности за короткий период времени с 2012 по 2015 год весьма показательно и довольно необычно для большинства приложений машинного обучения, относящихся к области

распознавания образов. Немаловажен и тот факт, что между увеличением глубины сети и повышением точности существует тесная корреляция. Поэтому в последние годы ускорились исследования, которые направлены на разработку алгоритмических модификаций, поддерживающих увеличенную глубину нейронных архитектур. Следует подчеркнуть, что сверточные нейронные сети относятся к числу наиболее глубоких из всех классов нейронных сетей. Интересно отметить, что традиционные сети прямого распространения, применяемые в других областях, таких как классификация, в большинстве случаев не обязаны быть особенно глубокими. В действительности устоявшийся термин “глубокое обучение” во многом обязан своим происхождением впечатляющим успехам сверточных нейронных сетей и конкретным улучшениям, наблюдавшимся после увеличения глубины сети.

Таблица 8.3. Количество слоев, использовавшихся в сетях-призерах соревнований ILSVRC

Название	Год	Количество слоев	Коэффициент ошибки топ-5
–	До 2012 года	≤ 5	$> 25\%$
<i>AlexNet</i>	2012	8	15,4%
<i>ZfNet/Clarifai</i>	2013	8 / > 8	14,8% / 11,1%
<i>VGG</i>	2014	19	7,3%
<i>GoogLeNet</i>	2015	22	6,7%
<i>ResNet</i>	2016	152	3,6%

8.4.7. Предварительно обученные модели

Одной из трудностей, с которыми приходится сталкиваться аналитику, работающему с изображениями, является то, что для определенного приложения могут даже отсутствовать необходимые тренировочные данные. Рассмотрим случай, когда имеется набор данных, который необходимо использовать для извлечения изображений. В задачах подобного рода метки недоступны, но важно, чтобы признаки были семантически согласованными. В других случаях может потребоваться выполнить классификацию набора данных с определенными метками, доступ к которому может быть ограничен и который отличается от крупных ресурсов наподобие *ImageNet*. Эти условия создают проблемы, поскольку для обучения нейронных сетей с нуля необходимо располагать большим объемом тренировочных данных.

Однако в случае изображений важно то, что признаки, выделенные из какого-то одного набора, могут успешно использоваться со многими другими источниками данных. Например, изображения кошки из разных источников не

будут слишком отличаться, если они были созданы с использованием одного и того же количества пикселей и цветовых каналов. В подобных случаях полезны источники типовых данных, представляющие широкий спектр изображений. Например, набор данных *ImageNet* [581] содержит более миллиона изображений, относящихся к более чем 1000 категориям изображений, которые встречаются в повседневной жизни. Выбранные 1000 категорий и большое количество всевозможных изображений образуют достаточно представительный и исчерпывающий набор данных, который можно применять для выделения признаков изображений в различных целях. Так, признаки, извлеченные из набора данных *ImageNet*, можно использовать для представления совершенно другого набора изображений, пропуская его через предварительно обученную сверточную нейронную сеть (наподобие *AlexNet*) и извлекая многомерные признаки из полносвязных слоев. Это новое представление может быть использовано для совершенно другого приложения, связанного, например, с кластеризацией или извлечением объектов. Подход такого типа настолько распространен, что *сверточные нейронные сети редко обучают с нуля*. Признаки, извлеченные из предпоследнего слоя, часто называют FC7-признаками; это название унаследовано от сети *AlexNet*. Разумеется, количество слоев в произвольной сверточной сети и сети *AlexNet* может не совпадать, однако название “FC7” закрепилось.

Подход этого типа, основанный на извлечении уже готовых признаков [390], можно рассматривать как разновидность *переносимого обучения* (transfer learning), поскольку мы используем извлечение признаков из публичного ресурса наподобие *ImageNet* для решения задач в условиях дефицита данных для тренировки сети. Такой подход стал стандартной практикой для многих задач распознавания образов, и многие библиотеки, такие как *Caffe*, обеспечивают легкий доступ к этим признакам [585, 586]. В действительности библиотека *Caffe* предоставляет целый набор таких предобученных моделей, которые можно загрузить и использовать [586]. В случае доступности дополнительных тренировочных данных их можно применять лишь для тонкой настройки более глубоких слоев (т.е. слоев, близких к выходному слою). Веса более ранних слоев (близких к входному слою) фиксируются. Причиной, по которой тренируются лишь глубокие слои, в то время как ранние слои остаются фиксированными, является то, что ранние слои захватывают лишь примитивные признаки наподобие краев, тогда как глубокие слои захватывают более сложные признаки. Примитивные признаки не изменяются в значительной мере в зависимости от специфики конкретного приложения, в то время как глубокие признаки могут быть чувствительны к ней. Например, для представления изображений любого типа требуются прямолинейные отрезки различной ориентации (захваченные в ранних слоях), но признаки, соответствующие колесу или грузовику, будут присущи только данным, содержащим изображения грузовиков. Иными словами,

ранние слои стремятся захватить легко обобщаемые признаки (подходящие для различных наборов данных компьютерного зрения), тогда как поздние слои — специфические признаки. Переносимость признаков, извлеченных из сверточных нейронных сетей, на различные наборы данных и задачи описана в [361].

8.5. Визуализация и обучение без учителя

Интересным свойством сверточных нейронных сетей является то, что признаки, которым они обучаются, поддаются хорошо понятной интерпретации. Однако фактическая интерпретация этих признаков требует приложения определенных усилий. Первое, что приходит на ум, — это просто визуализировать двумерные (пространственные) компоненты фильтров. Несмотря на то что такой тип визуализации может предоставить интересные образцы визуализации примитивных краев и линий, изученных в первом слое нейронной сети, он приносит мало пользы поздним слоям. В первом слое визуализация этих фильтров возможна, поскольку они работают непосредственно с входным изображением, и фильтры часто выглядят как элементарные части изображения (такие, как ребра). Однако визуализировать фильтры в поздних слоях не легко, так как они оперируют с входными объемами, которые уже были перемешаны операциями свертки. Чтобы получить форму, допускающую интерпретацию, необходимо найти способ, позволяющий отобразить результат операций, выполненных на всем пути, начиная от входа. Поэтому цель визуализации часто заключается в том, чтобы идентифицировать и выделить те части входного изображения, которым соответствует тот или иной скрытый признак. Например, один скрытый признак может быть чувствительным к изменениям в той части изображения, которая соответствует колесу грузовика, тогда как другой скрытый признак может быть чувствительным к капоту. Это достигается вполне естественным образом за счет вычисления чувствительности (т.е. градиента) скрытого признака по отношению к каждому пикселю входного изображения. Как будет показано далее, эти типы визуализации тесно связаны с обратным распространением ошибки, обучением без учителя и транспонированными сверточными операциями (используемыми для декодирующих компонент автокодировщиков). В данной главе мы обсудим эти тесно связанные темы под единым углом зрения.

Существуют две основные ситуации, требующие кодирования и декодирования изображений. Первая из них соответствует случаям, когда сжатые карты признаков обучаются путем использования любой из моделей обучения без учителя, которые обсуждались в предыдущих разделах. После проведения тренировки по методу обучения с учителем можно попытаться реконструировать те части изображения, которые в наибольшей степени активируют данный признак. Кроме того, идентифицируются те части изображения, которые, вероятнее

всего, активируют конкретный признак или класс признаков. Как вы увидите далее, эта цель может быть достигнута за счет применения различных типов обратного распространения и приемов оптимизации. Вторая ситуация соответствует исключительно обучению без учителя, в котором сверточная сеть (кодировщик) соединяется с деконволюционной сетью (декодировщиком). Далее будет показано, что последняя также представляет собой разновидность транспонированной свертки, которая аналогична обратному распространению ошибки. Однако в этом случае веса кодировщика и декодировщика обучаются совместно для минимизации ошибки реконструкции. Очевидно, что более простым случаем является первая ситуация, поскольку кодировщик тренируется по методике обучения с учителем, и нам остается лишь обучить сеть воздействию различных частей входного поля на различные скрытые признаки. В условиях второй ситуации тренировка и обучение весов сети должны выполняться с нуля.

8.5.1. Визуализация признаков обученной сети

Рассмотрим нейронную сеть, которая уже была обучена с использованием крупного набора данных наподобие *ImageNet*. Наша цель — визуализация и анализ влияния различных частей входного изображения (т.е. входного поля) на различные признаки скрытых слоев и выходного слоя (например, 1000 Softmax-выходов в сети *AlexNet*). Мы хотели бы знать, как действовать в следующих двух случаях.

1. Задана активация признака в любом месте нейронной сети для *конкретного* входного изображения, и нужно визуализировать те части входа, которым соответствует этот признак. Отметим, что данным признаком может быть один из скрытых признаков, относящихся к пространственным слоям, полносвязным скрытым слоям (например, FC7) или даже одному из Softmax-выходов. В последнем случае мы получаем определенную информацию о возможности отнесения конкретного входного изображения к тому или иному классу. Например, если входное изображение активирует метку “банан”, то мы надеемся увидеть части конкретного входного изображения, с наибольшей вероятностью представляющего банан.
2. Задан определенный признак в любом месте сети, и нужно найти фантазийное изображение, которое в наибольшей степени активирует именно этот признак. Как и в предыдущем случае, таким признаком может быть один из скрытых признаков или даже один из признаков, относящихся к Softmax-выходам. Например, мы хотим знать, какой тип фантазийного изображения с наибольшей вероятностью будет классифицирован как “банан” в имеющейся тренированной сети.

В обоих этих случаях самым простым способом визуализации воздействия конкретных признаков является использование градиентных методов. Вторая из указанных выше задач довольно трудная, и зачастую получить удовлетворительную визуализацию, не выполнив при этом тщательно продуманную регуляризацию, не удастся.

Градиентная визуализация активированных признаков

Алгоритм обратного распространения ошибки, который применяется для тренировки нейронных сетей, может быть использован также для визуализации на основе градиента. Следует отметить, что вычисление градиента на основе обратного распространения ошибки представляет собой разновидность транспонированной свертки. В традиционных автокодировщиках транспонированные матрицы весов (из тех, которые используются в слое кодировщика) часто задействуются в декодировщике. Поэтому связи между обратным распространением и реконструированием признаков являются глубокими и применимы ко всем типам нейронных сетей. Основным отличием от традиционного применения обратного распространения является то, что нашей конечной целью является определение чувствительности скрытых/выходных признаков к различным пикселям входного изображения, а не к весам. Однако даже при традиционном обратном распространении чувствительность выходов к различным слоям вычисляется в качестве промежуточного шага, поэтому один и тот же подход можно почти в точности использовать в обоих случаях.

Если чувствительность выхода o вычисляется по отношению к входным пикселям, то результат визуализации этой чувствительности по соответствующим пикселям называют *картами салиентности* (saliency map), или *картами выделенности* [456]. Например, выходом o может быть Softmax-вероятность (или ненормализованная оценка до применения Softmax) класса “банан”. Тогда для каждого пикселя x_i изображения мы хотели бы определить значение $\frac{\partial o}{\partial x_i}$. Это значение можно вычислить непосредственно путем применения алгоритма обратного распространения ошибки на всем пути до входного слоя⁸. Softmax-вероятность метки “банан” будет относительно нечувствительной к небольшим изменениям в тех частях изображения, которые не имеют отношения к

⁸ Как правило, в качестве промежуточного шага для вычисления градиентов по входящим весам некоторого скрытого слоя используется обратное распространение ошибки, доходящее только до данного слоя. Поэтому в процессе традиционной тренировки обратное распространение до входного слоя в действительности никогда не применяется. В то же время обратное распространение до входного слоя ничем не отличается от обратного распространения до скрытых слоев.

распознаванию бананов. Поэтому значения $\frac{\partial o}{\partial x_i}$ будут близки к нулю для подобных областей, тогда как части изображения, которые определяют банан, будут иметь большую величину. Например, в случае сети *AlexNet* весь объем размером $224 \times 224 \times 3$, определенный величинами $\frac{\partial o}{\partial x_i}$ *распространенных в обратном направлении градиентов*, будет иметь части с большими значениями, соответствующими банану на изображении. Чтобы визуализировать этот объем, сначала преобразуем его в градации серого, беря максимальную *абсолютную величину градиента* по трем RGB-каналам для создания карты $224 \times 224 \times 1$, содержащей только неотрицательные значения. Яркие части этой визуализации в серых тонах будут информировать нас о том, какие части входного изображения имеют отношение к банану. Примеры визуализации в градациях серого тех частей изображения, которые возбуждают соответствующие классы, приведены на рис. 8.12. Например, яркая часть изображения на рис. 8.12, *a*, активирует изображение животного. Как обсуждалось в разделе 2.4, подход такого типа также может быть использован для интерпретации и отбора признаков в традиционных нейронных сетях (а не только в сочетании со сверточными методами).



(a)



(б)

Рис. 8.12. Примеры частей характерных изображений, активируемых определенными метками классов. Эти изображения приведены в работе Симоняна, Ведалди и Зиссермана [456] и воспроизводятся здесь с разрешения авторов (©2014 Simonyan, Vedaldi, and Zisserman)

Этот общий подход также применяется для визуализации активаций специфических скрытых признаков. Рассмотрим значение h скрытой переменной для конкретного входного изображения. Как эта переменная откликается на вход-

ное изображение при ее текущем уровне активации? Идея заключается в том, что если мы немного увеличим или уменьшим цветовую интенсивность некоторых пикселей, то значение h испытает большее воздействие, чем если бы мы увеличили или уменьшили интенсивность других пикселей. На переменную h прежде всего будет воздействовать небольшая прямоугольная область изображения (т.е. рецептивное поле), размеры которой будут очень малы, если переменная h имеет заметное значение в ранних слоях, но намного большее в более поздних слоях. Например, рецептивное поле h может иметь размеры, равные всего лишь 3×3 , если оно выбирается из первого скрытого слоя в случае сети VGG. Примеры вырезов, соответствующих конкретным изображениям, в которых определенный нейрон в скрытом слое активируется в значительной степени, приведены на рис. 8.13, *справа*. Обратите внимание на то, что каждый ряд содержит довольно похожие изображения. Это не простое совпадение, поскольку ряд соответствует определенному скрытому признаку, и вариации в этом ряду обусловлены различными выборами изображения. Выбор изображений для ряда также не является случайным, поскольку мы выбираем изображения, которые в максимальной степени активизируют данный признак. Поэтому все изображения будут содержать те же визуальные характеристики, которые вызывают активацию этого скрытого признака. Часть визуализации, выполненная в серых тонах, соответствует чувствительности признака к специфическим по отношению к пикселям значениям в соответствующем вырезе.

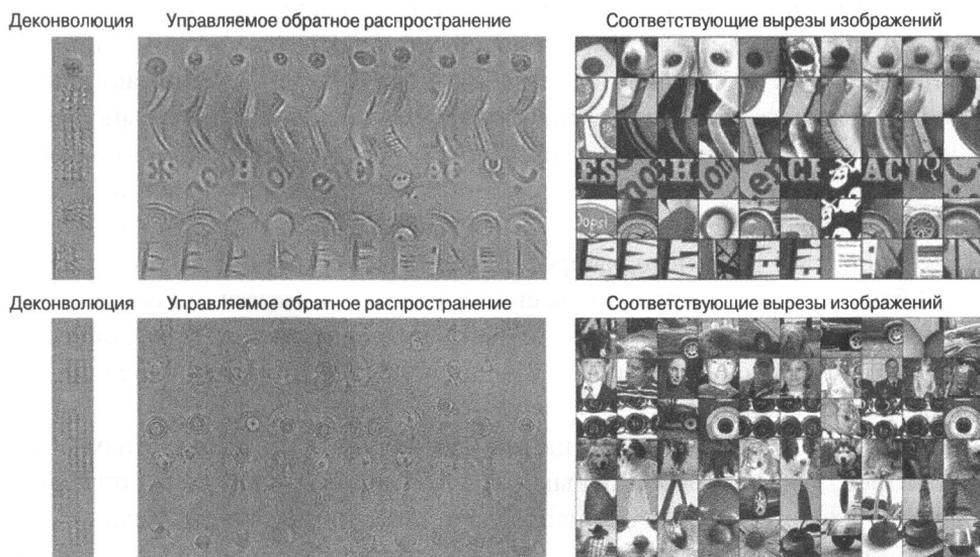


Рис. 8.13. Примеры визуализации активаций в различных слоях, которые приведены в работе Шпрингенберга и др. [466] и воспроизводятся здесь с разрешения авторов (© 2015 Springenberg, Dosovitskiy, Brox, Riedmiller)

При высоком уровне активации h некоторые из пикселей в этом рецептивном поле будут более чувствительны к h , чем к другим переменным. Изолируя пиксели, к которым скрытая переменная h проявляет наибольшую чувствительность, и визуализируя соответствующие прямоугольные области, можно получить представление о том, какая часть входной карты оказывает наибольшее воздействие на конкретный скрытый признак. Поэтому мы хотим вычислить $\frac{\partial h}{\partial x_i}$ для пикселей x_i , а затем визуализировать пиксели с большими значениями этого градиента. Однако вместо обратного распространения ошибки иногда применяют *обратное развертывание* (deconvnet) [556] и *контролируемое обратное распространение* (guided backpropagation) [466]. Идея обратного развертывания также используется в сверточных автокодировщиках. Основное отличие выражается в способе обратного распространения ReLU-нелинейности. Как обсуждалось при рассмотрении табл. 3.1, частная производная элемента ReLU копируется в обратном направлении в процессе обратного распространения ошибки, если вход ReLU является положительным; в противном случае она устанавливается равной нулю. Однако при обратном развертывании частная производная элемента ReLU копируется в обратном направлении, если она сама значительно больше нуля. Это все равно что использовать ReLU в отношении градиента, распространяемого в обратном направлении. Иными словами, мы заменяем $\bar{g}_i = \bar{g}_{i+1} \odot I(\bar{z}_i > 0)$ в табл. 3.1 на $\bar{g}_i = \bar{g}_{i+1} \odot I(\bar{g}_i > 0)$. Здесь \bar{z}_i представляет активации в прямом направлении, а \bar{g}_i — распространяемый в обратном направлении градиент по i -му слою, содержащему только ReLU-элементы. Функция $I(\cdot)$ — это поэлементная индикаторная функция, принимающая значение 1 для каждого элемента векторного аргумента, для которого выполняется заданное условие. В контролируемом обратном распространении мы сочетаем условия, применяемые в традиционном обратном распространении, и ReLU посредством условия $\bar{g}_i = \bar{g}_{i+1} \odot I(\bar{z}_i > 0) \odot I(\bar{g}_{i+1} > 0)$. Графическая иллюстрация трех вариаций обратного распространения ошибки приведена на рис. 8.14. В [466] было выдвинуто предположение о том, что контролируемое обратное распространение обеспечивает лучшую визуализацию, чем обратное развертывание, которое, в свою очередь, работает лучше, чем традиционное обратное распространение ошибки.

Один из способов интерпретации различий между традиционным обратным распространением и обратным развертыванием основан на толковании обратного распространения градиентов как операции декодировщика с транспонированной сверткой по отношению к кодировщику [456]. Однако в этом декодировщике мы вновь используем функцию ReLU, а не преобразование на основе градиента, подразумеваемое в ReLU. В конце концов, декодировщики любой формы используют те же функции активации, что и кодировщик. Другим

отличительным свойством подхода к визуализации, предложенного в [466], является полный отказ от использования слоев пулинга в сверточной нейронной сети с заменой их слоями шаговой свертки. В [466] были идентифицированы нейроны с высокой степенью активации в специфических слоях, соответствующих специфическим входным изображениям, и предоставлены визуализации прямоугольных областей изображений, соответствующих рецептивным полям этих скрытых нейронов. Как ранее уже отмечалось, на рис. 8.13, *справа*, содержатся области входа, соответствующие специфическим нейронам в скрытых слоях. Слева на этом же рисунке представлены специфические характеристики каждого изображения, которые возбуждают конкретный нейрон. Визуализация, представленная слева, получена с помощью контролируемого обратного распространения. Отметим, что верхний набор изображений соответствует шестому слою, в то время как нижний набор — девятому слою сверточной сети. В результате изображения в нижнем наборе соответствуют более крупным областям входного изображения, содержащим более сложные формы.

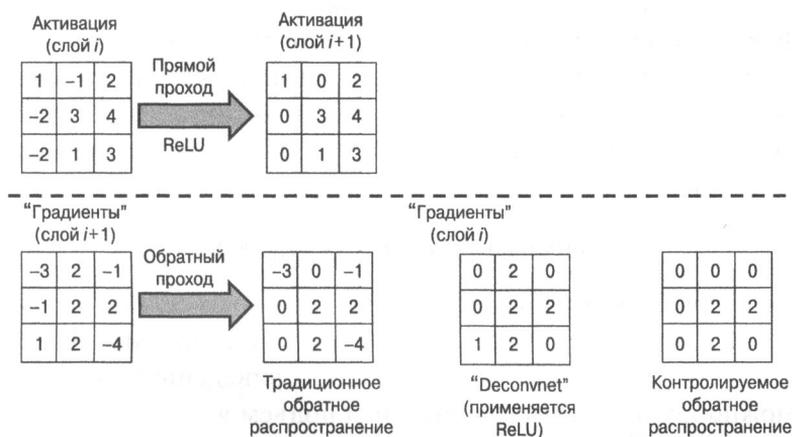


Рис. 8.14. Различные вариации обратного распространения ReLU для визуализации

Еще один показательный набор визуализаций из [556] приведен на рис. 8.15. Основным отличием является то, что в [556] также используются слои пулинга по максимальному значению, а вместо контролируемого обратного распространения применяется деконволюция — операция, обратная свертке. В качестве специфических скрытых переменных выбирались топ-9 наибольших активаций в каждой карте признаков. Во всех этих случаях вместе с соответствующей квадратной областью изображения представлена соответствующая визуализация. Очевидно, что скрытые признаки в ранних слоях соответствуют примитивным линиям, которые постепенно усложняются в поздних слоях. Это одна из причин того, что сверточные нейронные сети рассматриваются в качестве методов, которые создают иерархические признаки. Как правило, признаки в ранних слоях

являются в большей степени типовыми и могут использоваться на широком разнообразии наборов данных. Признаки в более поздних слоях обычно оказываются более специфическими в отношении отдельных наборов данных. Эта важная особенность играет важную роль в приложениях переносимого обучения, в которых широко используются предварительно обученные сети, и лишь более поздние слои подвергаются тонкой настройке способом, специфическим для набора данных и приложения.

Синтезированные изображения, возбуждающие определенные признаки

Приведенные выше примеры сообщают о тех частях конкретного изображения, которые в наибольшей степени воздействуют на конкретный нейрон. Можно задать и более общий вопрос: фрагменты изображений какого типа будут в максимальной степени активировать конкретный нейрон? Чтобы упростить обсуждение, рассмотрим случай, в котором нейроном является выходное значение o конкретного класса (т.е. ненормализованный выход до применения функции Softmax). Например, значением o может быть ненормализованная оценка класса “банан”. Заметьте, что аналогичный подход может быть применен к промежуточным нейронам, а не к оценкам классов. Мы хотим обучить входное изображение \bar{x} , которое максимизирует выход o , одновременно применяя регуляризацию к \bar{x} :

$$\text{максимизация } J(\bar{x}) = (o - \lambda \|\bar{x}\|^2).$$

Здесь λ — параметр регуляризации, выбор которого имеет большое значение для извлечения семантически интерпретируемых изображений. Для обучения входного изображения \bar{x} , максимизирующего приведенную выше целевую функцию, можно использовать градиентный подъем в сочетании с обратным распространением ошибки. Поэтому мы начинаем с нулевого изображения \bar{x} и обновляем \bar{x} , используя градиентный подъем в сочетании с обратным распространением по отношению к целевой функции. Иными словами, используется следующее правило обновления:

$$\bar{x} \leftarrow \bar{x} + \alpha \nabla_{\bar{x}} J(\bar{x}), \quad (8.4)$$

где α — скорость обучения. В данном случае очень важно то, что обратное распространение ошибки применяется для обновления пикселей изображения необычным способом с сохранением (уже обученных) фиксированных значений весов. Примеры синтезированных изображений для трех классов приведены на рис. 8.16. Другие передовые методы генерации более реалистичных изображений на основе меток классов обсуждаются в [358].

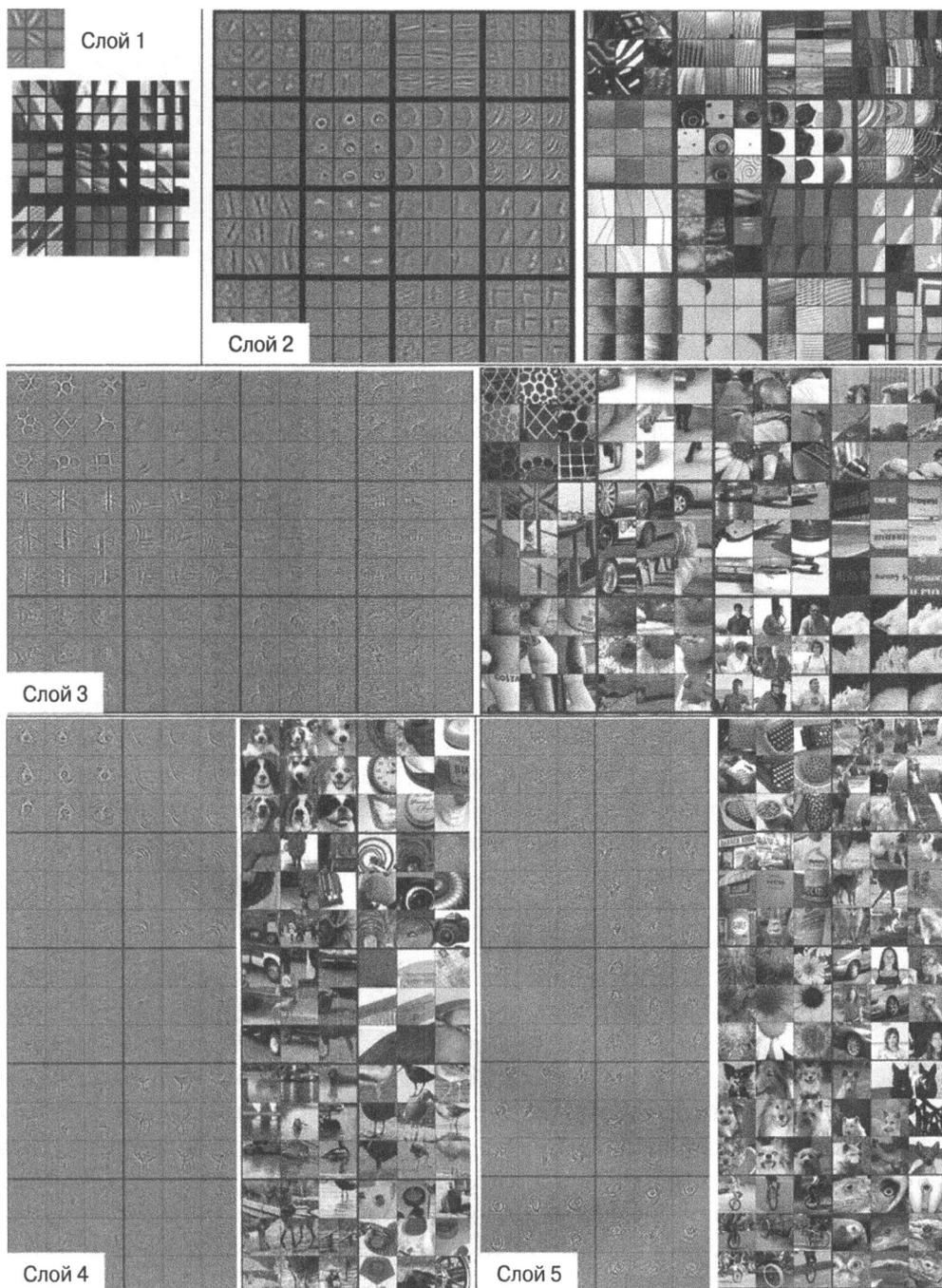


Рис. 8.15. Примеры визуализации активаций в различных слоях, приведенные в работе Зайлера и Фергюса [556] и воспроизведенные здесь с разрешения авторов (© Springer International Publishing Switzerland, 2014)

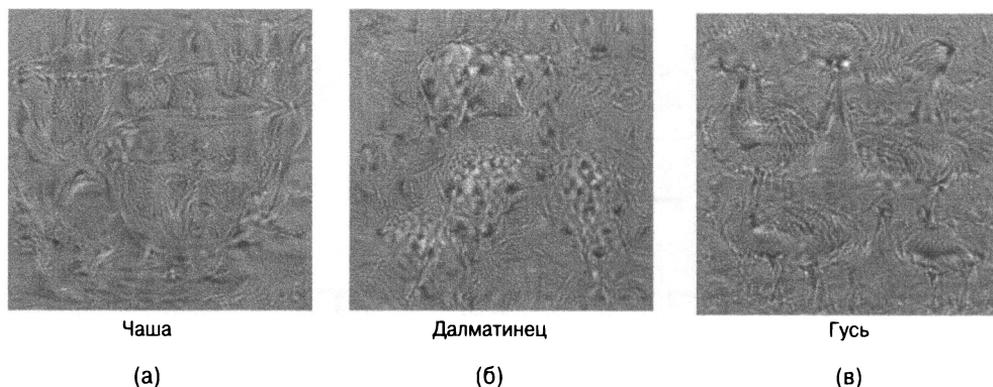


Рис. 8.16. Примеры синтезированных изображений для меток определенных классов. Эти изображения приведены в работе Симоняна, Ведалди и Зиссермана [456] и воспроизводятся здесь с разрешения авторов (©2014 Simonyan, Vedaldi, and Zisserman)

8.5.2. Сверточные автокодировщики

Использование автокодировщика в традиционных нейронных сетях обсуждалось в главах 2 и 4. Вспомните, что автокодировщик реконструирует точки данных после их прохождения через фазу сжатия. В некоторых случаях данные не сжимаются, несмотря на разреженность представлений. Та часть сети, которая предшествует наиболее сжатому слою архитектуры, называется кодировщиком, а часть, следующая за кодировщиком, — декодировщиком. Мы повторяем графическое представление архитектуры “кодировщик — декодировщик”, соответствующее традиционному случаю, на рис. 8.17, а. Сверточный автокодировщик работает по аналогичному принципу, реконструируя изображения после прохождения ими фазы сжатия. Основное различие между традиционными и сверточными автокодировщиками заключается в том, что последние сфокусированы на использовании пространственных отношений между точками для извлечения признаков, которые допускают визуальную интерпретацию. Операции пространственной свертки в промежуточных слоях достигают в точности тех же целей. Иллюстрация сверточного автокодировщика приведена на рис. 8.17, б, в сравнении с традиционным автокодировщиком, представленным на рис. 8.17, а. Обратите внимание на трехмерную пространственную форму кодировщика и декодировщика во втором случае. Однако можно предложить несколько вариаций этой базовой архитектуры. Например, коды в промежутке могут быть либо пространственными, либо уплощенными с помощью полносвязных слоев, в зависимости от текущей задачи. Полносвязные слои нужны для создания многомерного кода, который можно использовать с произвольными приложениями (не заботясь о пространственных ограничениях, налагаемых на признаки). В последующем мы упростим обсуждение, предполагая, что сжатый код в средней части является пространственным по своей природе.

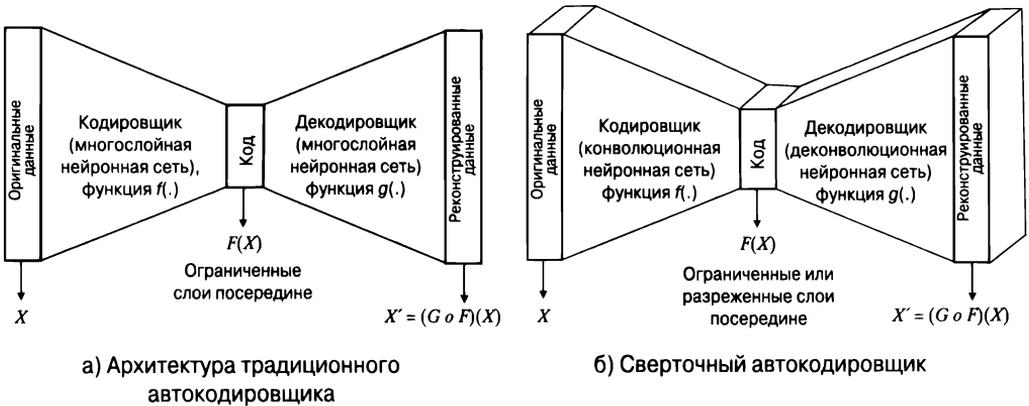


Рис. 8.17. Традиционный и сверточный автокодировщики

Точно так же, как часть, соответствующая декодировщику и отвечающая за сжатие, использует операцию свертки, часть, соответствующая операции распаковки, использует операцию *деконволюции*. Аналогичным образом пулингу (понижающей дискретизации) сопутствует обратная операция: *повышающая дискретизация*, или *анпулинг* (unpooling). Деконволюцию также называют *транспонированной сверткой* (transposed convolution). Интересно отметить, что операция транспонированной свертки — это та же операция, которая используется для обратного распространения ошибки. Возможно, термин “деконволюция” немного вводит в заблуждение, поскольку каждая такая операция в действительности является сверткой с фильтром, который получается путем транспонирования и инвертирования тензорного представления фильтра оригинальной свертки (см. рис. 8.7 и уравнение 8.3). Мы уже видим, что свертка и обратное распространение ошибки базируются на аналогичных принципах. Основное различие связано со способом обращения с функцией ReLU, который делает деконволюцию более похожей на обратное развертывание или контролируемое обратное распространение. В действительности декодер в сверточном автокодировщике выполняет операции, аналогичные операциям фазы обратного распространения визуализации на основе градиента. Некоторые архитектуры обходятся без операций понижения и повышения дискретизации и работают только с операциями свертки (вместе с функциями активации). Показательным примером является строение *полносверточных сетей* [449, 466].

Тот факт, что операция деконволюции в действительности мало чем отличается от операции свертки, не является неожиданным. Даже в традиционных сетях прямого распространения часть сети, соответствующая декодировщику, выполняет операции матричного умножения того же типа, что и часть сети, соответствующая кодировщику, за исключением того, что применяются транспонированные матрицы весов. Аналогия между традиционными и сверточными

автокодировщиками подытожена в табл. 8.4. Следует отметить, что с точки зрения выполнения матричных операций умножения распространение в прямом направлении и распространение в обратном направлении в традиционных и сверточных сетях соотносятся между собой одинаковым образом. То же самое справедливо и в отношении природы связи между кодировщиками и автокодировщиками.

Таблица 8.4. Связь между обратным распространением ошибки и декодировщиками

Линейная операция	Традиционные нейронные сети	Сверточные нейронные сети
Прямое распространение	Матричное умножение	Свертка
Обратное распространение	Транспонированное матричное умножение	Транспонированная свертка
Слой декодировщика	Транспонированное матричное умножение (идентично обратному распространению)	Транспонированная свертка (идентично обратному распространению)

В данном случае мы имеем дело с тремя операциями, соответствующими свертке, пулингу по максимальному значению и введению нелинейности с помощью ReLU. Задача заключается в выполнении в слое декодировщика операций, обратных тем, которые были выполнены в слое кодировщика. Простых способов инвертирования некоторых из этих операций (таких, как пулинг по максимуму и ReLU) не существует. Тем не менее удачный выбор проектных решений обеспечивает получение отличных реконструированных изображений. Сначала мы рассмотрим случай автокодировщика с одним слоем, в котором выполняются операции свертки, ReLU и пулинга по максимальному значению, а затем обсудим способы обобщения этой модели на несколько слоев.

Несмотря на то что обычно в декодировщике требуется обратить операции, выполненные в кодировщике, ReLU не является обращаемой функцией, поскольку для нулевого значения имеется множество возможных вариантов инверсии. По этой причине функция ReLU заменяется в слое декодировщика другой функцией ReLU (хотя также возможны альтернативные варианты решения). Архитектуру этого простого автокодировщика можно представить в следующем виде:

Свертка \Rightarrow ReLU \Rightarrow Пулинг по максимуму \Rightarrow Код \Rightarrow Анпулинг \Rightarrow ReLU \Rightarrow Деконволюция

Кодировщик

Декодировщик

Обратите внимание на симметрию в расположении слоев, выполняющих прямую и обратную операции в кодировщике и декодировщике соответственно. В то же время существуют многочисленные вариации этой базовой схемы. Например, ReLU можно поместить после слоя деконволюции. Кроме того, в некоторых вариациях [310] рекомендуется использовать кодировщики большей глубины, чем декодировщики, при сохранении симметрии архитектуры. В то же время в случае приведенной выше стековой вариации симметричной архитектуры существует возможность обучить один только кодировщик с классифицирующим выходным слоем (с использованием набора данных наподобие ImageNet), а затем применить симметричный ему декодировщик (с транспонированными/инвертированными фильтрами) для выполнения “развертывающей” визуализации [556]. Несмотря на то что этот подход всегда можно использовать для инициализации автокодировщика, мы обсудим его улучшенную версию, в которой кодировщик и декодировщик тренируются совместно по методике обучения без учителя.

В данном случае мы будем рассматривать каждый слой как сверточный, а ReLU — как отдельный слой, поэтому в общей сложности мы имеем семь слоев, включая вход. Это простейшая архитектура, поскольку в каждом ее кодировщике и декодировщике используется только по одному сверточному слою. В более общих архитектурах эти слои образуют стек для создания более мощных архитектур. Тем не менее полезно проиллюстрировать связь между такими базовыми операциями, как анпулинг и деконволюция, с соответствующими им операциями, выполняемыми в компоненте кодировщика (такими, как пулинг и свертка). Еще одним упрощением является то, что код содержится в пространственном слое, а посередине могут быть вставлены полносвязные слои. Несмотря на то что в этом примере (и на рис. 8.17, б) используется пространственный код, наличие полносвязных слоев посередине более полезно для практических применений. С другой стороны, расположенные посередине пространственные слои можно задействовать для визуализации промежуточных данных.

Рассмотрим ситуацию, когда в первом слое кодировщика используются d_2 квадратных фильтра $F_1 \times F_1 \times d_1$. Также предположим, что первый слой представляет собой (пространственный) объем с размерами $L_1 \times L_1 \times d_1$. Пусть (i, j, k) -й элемент p -го фильтра в первом слое имеет вес $w(p, 1)_{ijk}$. Эти обозначения согласуются с теми, которые использовались в разделе 8.2, где была определена операция свертки. В сверточном слое принято использовать дополнение вполне определенного уровня, такое, чтобы карты признаков во втором слое также имели размер L_1 . Таким уровнем дополнения является $F_1 - 1$, и его называют *половинным дополнением*. Однако в сверточном слое можно вообще не использовать дополнение, если в соответствующем деконволюционном слое применяется полное дополнение. В общем случае сумма дополнений между сверточным

и соответствующим ему деконволюционному слою должна быть равна $F_1 - 1$ для сохранения пространственного размера слоя в паре “свертка — деконволюция”.

Здесь важно понимать, что, хотя каждый элемент $W^{(p, 1)} = [w_{ijk}^{(p, 1)}]$ является трехмерным тензором, этот тензор можно сделать четырехмерным, включив в него индекс p . Операция деконволюции использует транспонированную версию тензора, что напоминает подход, применяемый в обратном распространении (см. раздел 8.3.3). Операция деконволюции выполняется с шестого по седьмой слой (учитывая промежуточные слои ReLU/пулинг/анпулинг). Поэтому мы будем определять (деконволюционный) тензор $U(s, 6) = [u_{ijk}^{(s, 6)}]$ в его связи с $W^{(p, 1)}$. Слой 5 содержит d_2 карт признаков, которые были унаследованы от операции свертки в первом слое (и не были изменены операциями пулинга/анпулинга/ReLU). Эти d_2 карты признаков должны быть отображены на d_1 слоев, где d_1 равно 3 для RGB-каналов цвета. Поэтому количество фильтров в деконволюционном слое равно глубине фильтров в сверточном слое, и наоборот. Это изменение формы можно рассматривать как результат транспонирования и пространственной инверсии четырехмерного тензора, созданного фильтрами. Кроме того, элементы этих двух четырехмерных тензоров связаны между собой следующим соотношением:

$$u_{ijk}^{(s, 6)} = w_{rms}^{(k, 1)} \quad \forall s \in \{1 \dots d_1\}, \quad \forall k \in \{1 \dots d_2\}. \quad (8.5)$$

Здесь $r = n - i + 1$ и $m = n - j + 1$, где первый слой имеет пространственные размеры $n \times n$. Обратите внимание на перестановку индексов s и k в вышеприведенном соотношении. Данное соотношение идентично уравнению 8.3. При этом ни взаимная привязка весов в кодировщике и декодировщике, ни даже симметрия архитектуры между этими двумя компонентами не являются необходимыми [310].

Фильтры $U^{(s, 6)}$ в шестом слое используются, как и любая другая свертка, для реконструкции цветowych RGB-каналов изображений из активаций в слое 6. Поэтому операция деконволюции в действительности является операцией свертки, за исключением того, что она выполняется с транспонированным и пространственно инвертированным фильтром. Как обсуждалось в разделе 8.3.2, этот тип операции деконволюции используется и при обратном распространении. Обе операции, свертка и деконволюция, также могут выполняться с помощью матричного умножения, о чем уже говорилось в этом разделе.

Однако операции пулинга приводят к необратимой потере части информации, поэтому их точное обращение невозможно. Это происходит из-за того, что в процессе пулинга утрачиваются все значения, кроме максимального. Операция пулинга по максимальному значению реализуется с помощью так называемых *коммутаторов*. В процессе выполнения пулинга точные позиции максимальных значений сохраняются в коммутаторе. В качестве примера рассмотрим

обычную ситуацию, когда выполняется пулинг 2×2 с шагом 2. В этом случае пулинг уменьшает оба пространственных размера в два раза и выбирает максимальное из $2 \times 2 = 4$ значений в каждой из (неперекрывающихся) областей пулинга. Точные координаты максимальных значений сохраняются в коммутаторе. При выполнении анпулинга размеры увеличиваются в два раза, и (максимальные) значения из предыдущего слоя, хранящиеся в коммутаторе, копируются в соответствующие позиции. Остальные значения устанавливаются в нуль. Поэтому в случае неперекрывающихся областей пулинга размером 2×2 мы получим после выполнения анпулинга слой, в котором ровно 75% элементов равны нулю.

Как и в традиционных автокодировщиках, функция потерь определяется ошибкой реконструкции по всем $L_1 \times L_1 \times d_1$ пикселям. Поэтому, если $h_{ijk}^{(1)}$ представляет значения пикселей в первом (входном) слое, а $h_{ijk}^{(7)}$ — значения пикселей в седьмом (выходном) слое, то функция потерь реконструкции E определяется следующей формулой:

$$E = \sum_{i=1}^{L_1} \sum_{j=1}^{L_1} \sum_{k=1}^{d_1} (h_{ijk}^{(1)} - h_{ijk}^{(7)})^2. \quad (8.6)$$

Применяются и другие функции ошибки (такие, как L_1 -потеря и отрицательная логарифмическая функция правдоподобия).

Совместно с автокодировщиком можно использовать традиционное обратное распространение ошибки. Обратное распространение через слой деконволюции и ReLU ничем не отличается от обратного распространения через слой свертки. В случае пулинга по максимальному значению градиент перетекает через коммутаторы без изменений. Поскольку параметры кодировщика и декодировщика взаимосвязаны, в процессе градиентного спуска необходимо суммировать градиенты соответствующих параметров обоих слоев. Также интересно отметить, что алгоритм обратного распространения ошибки через слой деконволюции использует операции, почти идентичные тем, которые применяются при прямом распространении через сверточный слой. Это объясняется тем, что как обратное распространение, так и деконволюция требуют последовательного транспонирования четырехмерного тензора, используемого для преобразования.

Базовый автокодировщик можно легко расширить на случай использования нескольких слоев свертки, пулинга и ReLU. Трудности, возникающие при работе с многослойными автокодировщиками, обсуждаются в [554]. В этой работе также предложено несколько трюков, позволяющих улучшить производительность. Существует несколько вариантов архитектурных решений, применяемых для улучшения производительности. В частности, важно отметить, что для уменьшения пространственных размеров кодировщика часто используют шаговые свертки (вместо пулинга по максимуму), что должно быть сбалансировано

в декодировщике с помощью *дробных шаговых сверток*. Рассмотрим ситуацию, в которой для снижения пространственных размеров в кодировщике используется шаг S с некоторым дополнением. В декодировщике можно увеличить пространственные размеры в соответствующее количество раз с помощью следующего трюка. В процессе выполнения свертки мы растягиваем входной объем, помещая $S - 1$ строк нулей⁹ между каждой парой строк и $S - 1$ столбцов нулей между каждой парой столбцов, прежде чем применять фильтр. В результате входной объем уже растягивается в каждом пространственном измерении с коэффициентом, равным примерно S . Помимо этого, перед выполнением свертки с транспонированным фильтром вдоль границ может быть применено дополнение. Такой подход обеспечивает дробный шаг и расширяет выходной размер в декодировщике. Альтернативным подходом к растяжению входного объема свертки является вставка интерполированных значений (вместо нулей) между исходными записями входного объема. Интерполяция выполняется с использованием выпуклой комбинации ближайших четырех значений, а в качестве коэффициента пропорциональности интерполяции применяется убывающая функция расстояний до каждого из этих значений [449]. Иногда подход, основанный на растяжении входов, сочетается с растягиванием фильтров, а также вставкой нулей в фильтр [449]. Растягивание фильтра приводит к подходу, получившему название *расширенная свертка* (dilated convolution), хотя в случае дробных шаговых сверток он не является универсальным. Подробное обсуждение арифметики сверток (включая дробные шаговые свертки) содержится в [109]. По сравнению с традиционным автокодировщиком реализация сверточного автокодировщика выглядит немного сложнее, причем возможны многочисленные вариации, ориентированные на достижение лучшей производительности. Для получения более подробной информации по этому вопросу обратитесь к библиографической справке.

Методы обучения без учителя также имеют применения, целью которых является улучшение обучения с учителем. Из них наиболее очевидным является *предварительное обучение* (pretraining), которое обсуждалось в разделе 4.7. Методология предварительного обучения, применяемая в сверточных нейронных сетях, принципиально ничем не отличается от той, которая используется в традиционных нейронных сетях. Предварительное обучение также может выполняться путем извлечения весов из обученной *сверточной сети глубокого доверия* (deep-belief convolutional network) [285]. Это аналогично подходу, применяемому в традиционных нейронных сетях, где одними из первых моделей, используемых для предварительного обучения, были каскадированные машины Больцмана.

⁹ Пример доступен по адресу http://deeplearning.net/software/theano/tutorial/conv_arithmetic.html.

8.6. Применение сверточных сетей

Сверточные нейронные сети (CNN) находят ряд применений в области обнаружения и локализации объектов, а также обработки видео и текстов. Базовый принцип заключается в использовании сверточных сетей для предоставления конструируемых признаков, поверх которых могут строиться многомерные приложения. В этом отношении практически ни один другой класс нейронных сетей не может сравниться с CNN. В последние годы даже были предложены конкурентоспособные методы, предназначенные для обучения по принципу “последовательность в последовательность”, которое традиционно было сферой применения преимущественно рекуррентных сетей.

8.6.1. Извлечение изображений на основе содержимого

При извлечении изображений на основе содержимого каждое изображение прежде всего преобразуется в набор многомерных признаков с помощью предварительно обученных классификаторов наподобие *AlexNet*. Обычно предварительное обучение осуществляется заранее с использованием крупных наборов данных наподобие *ImageNet*. Существует огромное количество доступных для выбора предварительно обученных классификаторов [586]. Признаки, извлеченные из полносвязных слоев классификатора, могут быть использованы для создания многомерных представлений изображений. Многомерные представления можно применять совместно с любой системой извлечения многомерных объектов, обеспечивая получение результатов высокого качества. Использование нейронных кодов для извлечения изображений обсуждается в [16]. Причиной работоспособности такого подхода является то, что признаки, извлеченные из *AlexNet*, имеют семантическое значение для различных типов форм, представленных в данных. В результате при работе с такими признаками качество извлечения обычно оказывается очень высоким.

8.6.2. Локализация объектов

Предположим, у нас имеется изображение с фиксированным набором объектов и мы хотим идентифицировать прямоугольные области, в пределах которых располагаются эти объекты. Основная идея заключается в том, чтобы выделить на изображении прямоугольными рамками каждый из объектов, количество которых *фиксировано*. Для простоты рассмотрим случай, когда на изображении имеется только один объект. Обычно задача локализации сопутствует задаче классификации, когда сначала необходимо классифицировать объект, а затем ограничить его прямоугольной рамкой. Пример классификации и локализации изображения приведен на рис. 8.18, где идентифицирован класс “рыба”, а участок изображения с этим объектом выделен ограничивающим прямоугольником.

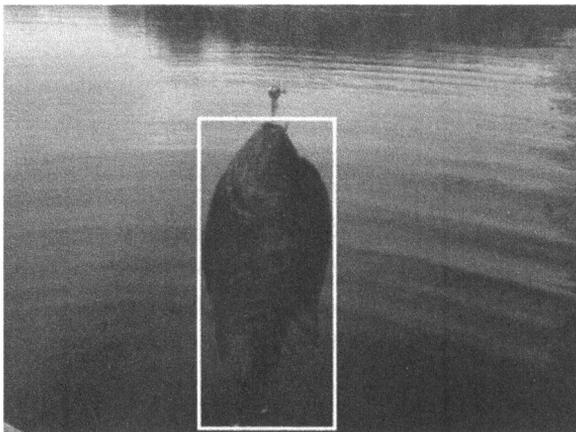


Рис. 8.18. Пример классификации/локализации изображения, в котором идентифицирован класс “рыба”, а соответствующий участок ограничен прямоугольной рамкой. Данное изображение приведено исключительно в иллюстративных целях

Ограничивающий прямоугольник можно однозначно определить с помощью четырех чисел. Обычно это делается путем задания координат левого верхнего угла и длин сторон прямоугольника. Этих четырех чисел будет вполне достаточно для идентификации прямоугольника. Следовательно, задача идентификации сводится к регрессионной задаче с несколькими целевыми значениями. Важно понимать, что модели, подлежащие обучению для задач классификации и регрессии, почти совпадают; небольшие различия относятся лишь к двум последним полностью связанным слоям. Это обусловлено тем, что семантическая природа признаков, извлекаемых из сверточной сети, часто хорошо обобщается на широкий класс задач. Поэтому мы можем использовать следующий подход.

1. Прежде всего, тренируем классификатор на основе нейронной сети наподобие *AlexNet* или используем предварительно обученную версию этого классификатора. Во время первой фазы достаточно тренировать классификатор на парах “изображение — класс”. Для этих целей можно воспользоваться одной из готовых версий классификатора, уже предобученной на наборе данных ImageNet.
2. Последние два полностью связанных слоя и слой Softmax удаляются. Этот удаленный набор слоев называется *классификационной головной частью сети* (classification head). Вместо них к оставшейся части сети подключается новый набор двух полностью связанных слоев и слой линейной регрессии. Лишь эти слои подлежат дальнейшему обучению на тренировочных данных, содержащих изображения вместе с их ограничительными прямоугольниками. Этот новый набор слоев называется *регрессионной головной*

частью сети (regression head). Имейте в виду, что веса в сверточных слоях не изменяются и остаются фиксированными. Классификационная и регрессионная заголовочные части сети представлены на рис. 8.19. Поскольку головные части классификации и регрессии никак не связаны между собой, их можно обучать независимо. На сверточные слои возлагаются функции создания визуальных признаков как для классификации, так и для регрессии.

3. Также возможна тонкая настройка сверточных слоев с целью сделать их чувствительными как к классификации, так и к регрессии (поскольку первоначально они были обучены лишь для решения задач классификации). В этом случае к сверточным слоям подключаются обе головные части, как классификационная, так и регрессионная, и сети предоставляются тренировочные данные, содержащие информацию об изображениях, их классах и ограничивающих прямоугольниках. Обратное распространение ошибки используется для тонкой настройки всех слоев. Пример такой полной архитектуры приведен на рис. 8.19.
4. Работа всей сети (вместе с подключенными головными частями классификации и регрессии) проверяется на тестовых изображениях. Выходы классификационной головной части предоставляют вероятности классов, тогда как выходы регрессионной головной части предоставляют ограничивающие прямоугольники.

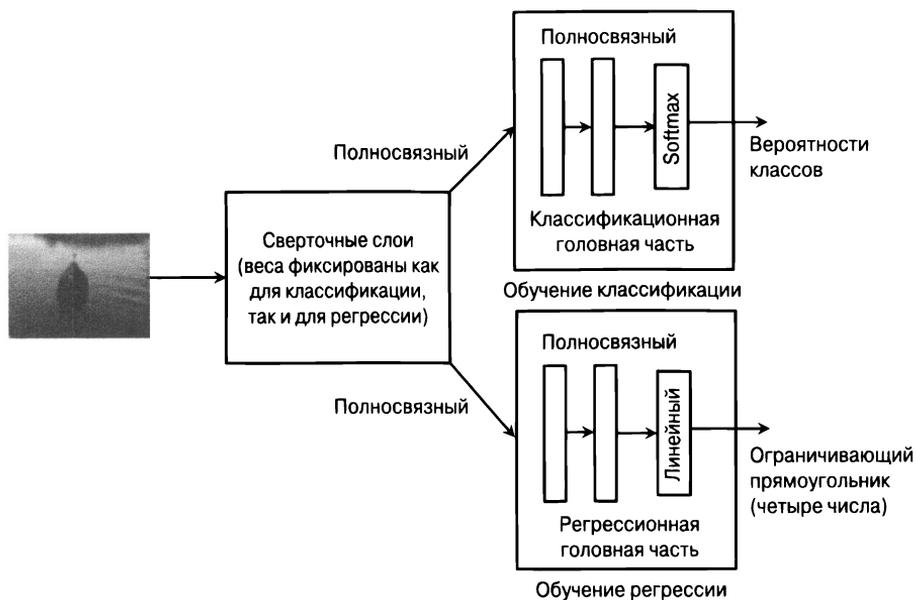


Рис. 8.19. Общая структура классификации и локализации

Для получения результатов высочайшего качества можно применять подход на основе скользящих окон. Основная идея такого подхода заключается в выполнении локализации во многих местах изображения с помощью скользящего окна с последующим объединением результатов различных прогонов. Примером такого подхода является метод *Overfeat* [441]. Ссылки на другие методы локализации приведены в разделе библиографической справки.

8.6.3. Обнаружение объектов

Обнаружение объектов весьма напоминает локализацию объектов, за исключением того, что на изображении встречается переменное количество объектов других классов. В этом случае мы хотим идентифицировать все объекты, встречающиеся на изображении, вместе с их классами. Пример обнаружения объектов представлен на рис. 8.20, который включает четыре объекта, соответствующие классам “рыба”, “девушка”, “ведро” и “сидение”. Ограничивающие прямоугольники этих классов также показаны на рисунке. Обнаружение объектов в целом — более трудная задача, чем локализация, из-за переменного количества выходов. Мы даже не знаем заранее, сколько объектов находится на изображении. В этом случае нельзя использовать архитектуру из предыдущего раздела, поскольку непонятно, какое количество классификационных или регрессионных головных частей может быть присоединено к сверточным слоям.

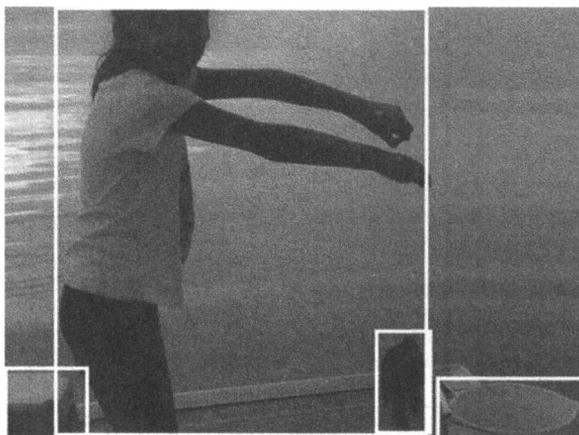


Рис. 8.20. Пример обнаружения объектов. В данном случае идентифицируются четыре объекта вместе с их ограничительными прямоугольниками. Этими четырьмя объектами являются “рыба”, “девушка”, “ведро” и “сидение”. Данное изображение приведено исключительно в иллюстративных целях

Простейший подход к решению проблемы заключается в использовании скользящего окна. В этом подходе испытываются все возможные ограничива-

ющие прямоугольники, к которым применяется локализация объектов для обнаружения одного объекта. В результате можно обнаружить различные объекты в различных ограничивающих прямоугольниках или же один и тот же объект в перекрывающихся прямоугольниках. После этого элементы, обнаруженные в различных ограничивающих прямоугольниках, могут объединяться для получения окончательного результата. К сожалению, это довольно дорогостоящий подход. В случае изображения размером $L \times L$ количество возможных прямоугольников равно L^4 . Учтите также, что во время тестирования для каждой из этих L^4 возможностей и для каждого изображения необходимо выполнять классификацию и регрессию. Это становится проблемой, поскольку обычно рассчитывают на то, что затраты времени на тестирование будут достаточно умеренными для того, чтобы можно было получать результаты за разумное время.

Для преодоления проблем подобного рода были разработаны *методы рекомендуемых областей* (region proposal methods). Базовая идея такого метода заключается в том, что он может служить в качестве универсального детектора объектов, который объединяет области с похожими пикселями с целью создания более крупных областей. Поэтому данные методы применяются для создания набора рекомендуемых ограничивающих прямоугольников, к каждому из которых затем применяются методы классификации/локализации. Учтите, что некоторые области-кандидаты могут не содержать корректных объектов, а некоторые могут содержать перекрывающиеся объекты. Полученные области используются для объединения и идентификации всех объектов на изображении. Этот универсальный подход применяется в различных алгоритмах, таких как *MCG* [172], *EdgeBoxes* [568] и *SelectiveSearch* [501].

8.6.4. Распознавание естественного языка и обучение последовательностей

Несмотря на то что в машинном обучении текстовых последовательностей предпочтение отдается рекуррентным нейронным сетям, в последние годы в этой области все большей популярностью пользуются сверточные нейронные сети. На первый взгляд может показаться, что сверточные нейронные сети не вписываются естественным образом в задачи, связанные с обработкой текста для извлечения информации. Во-первых, формы изображений интерпретируются одинаково, где бы они ни находились. Это не совсем то, что необходимо при работе с текстом, где от позиции слова в предложении зависит очень многое. Во-вторых, такие вопросы, как перенос позиции и сдвиг элементов, не могут трактоваться аналогичным образом при работе с текстовыми данными. Обычно соседние пиксели изображения очень сходны, тогда как соседние слова в тексте почти никогда не обладают таким свойством. Несмотря на все эти различия,

системы, основанные на сверточных сетях, в последние годы продемонстрировали улучшенную производительность.

Точно так же, как изображение представляется в виде двухмерного объекта с дополнительным измерением, определяемым количеством цветовых каналов, текстовая последовательность представляется в виде одномерного объекта с глубиной, определяемой размерностью его представления. Размерность представления текстовой последовательности равна размеру словаря в случае прямого кодирования. Поэтому вместо трехмерных объектов с пространственной протяженностью и глубиной (цветовые каналы/карты признаков) фильтры для текстовых данных являются двухмерными объектами, характеризующимися длиной окна (последовательности), используемого для скольжения вдоль последовательности, и глубиной, определяемой размером словаря. Кроме того, количество фильтров в данном слое определяет количество карт признаков в следующем слое (как и при работе с изображениями). В случае изображений операции свертки выполняются во всех двухмерных расположениях, тогда как в случае текстовых данных свертки выполняются во всех одномерных точках последовательности с одним и тем же фильтром. Одна из трудностей такого подхода заключается в том, что использование прямого кодирования увеличивает количество каналов, тем самым увеличивая количество параметров фильтров в первом слое. Размер словаря типичного корпуса часто может выражаться числами порядка 10^6 . Поэтому вместо прямого кодирования индивидуальных слов используются различные типы предварительно обученных распределенных представлений, или вложений, слов, такие как *word2vec* или *GLoVe* [371] (см. главу 2). Данные типы кодировки слов семантически богаче, и размерность представлений может быть уменьшена до нескольких тысяч (вместо сотен тысяч). При таком подходе количество параметров в первом слое может быть уменьшено на порядок, а дополнительным плюсом является получение семантически богатого представления. Все остальные операции (наподобие пулинга по максимальному значению или свертки) аналогичны операциям, выполняемым в отношении изображений.

8.6.5. Классификация видео

Видеопотоки можно рассматривать как обобщение графических данных, которое вводит временную компоненту для последовательности изображений. Таким образом, данные этого типа можно отнести к категории *пространственно-временных данных*, что требует от нас обобщения двухмерных пространственных сверток на трехмерные пространственно-временные свертки. Каждый кадр видеофайла можно рассматривать как изображение, поэтому мы имеем дело с последовательностью изображений, развернутой во времени. Рассмотрим ситуацию, когда каждое изображение имеет размеры $224 \times 224 \times 3$ и

всего получено 10 кадров. В таком случае размер видеосегмента составляет $224 \times 224 \times 10 \times 3$. Вместо выполнения пространственных сверток с двухмерным пространственным фильтром (с дополнительным измерением глубины, захватывающим три цветовых канала) мы выполняем пространственно-временные свертки с трехмерным пространственно-временным фильтром (также имеющим измерение глубины, захватывающим цветовые каналы). Интересно отметить, что природа фильтра зависит от специфики конкретного набора данных. Набор чисто последовательных данных (например, текст) требует использования одномерных сверток с окнами, набор изображений — двухмерных сверток, а набор видеоданных — трехмерных сверток. Ссылки на несколько статей, в которых для классификации видео используются трехмерные свертки, приведены в разделе библиографической справки.

Любопытно, что дополнительный вклад, обеспечиваемый использованием трехмерных сверток, оказывается довольно ограниченным по сравнению с тем, чего можно добиться усреднением классификаций индивидуальных кадров, полученных с помощью классификаторов изображений. Частично эта проблема обусловлена тем, что движение добавляет лишь ограниченный объем информации для целей классификации по сравнению с тем, который предоставляется индивидуальными изображениями. Кроме того, трудно подобрать достаточно большой набор видеоданных. Даже если набор данных включает миллион видеоклипов, зачастую даже этого будет недостаточно, поскольку объем данных, необходимых для трехмерных сверток, намного больше объема, требуемого для двухмерных сверток. Наконец, трехмерные сверточные нейронные сети хорошо подходят для сравнительно коротких видеосегментов (например, длительностью полсекунды), но не для длинных видеофайлов.

Если речь идет о длительных видео, то в этом случае целесообразно объединять сверточные нейронные сети с рекуррентными нейронными сетями (или сетями LSTM). Например, можно использовать двухмерные свертки по отношению к индивидуальным кадрам, а рекуррентные сети — для переноса состояний из одного кадра в следующий. Также можно применять трехмерные сверточные нейронные сети по отношению к коротким видеосегментам, а затем связывать их с рекуррентными блоками. Такой подход помогает идентифицировать действия на протяжении длительных временных промежутков. Ссылки на методы, в которых объединяются сверточные и рекуррентные нейронные сети, приведены в разделе библиографической справки.

8.7. Резюме

В этой главе было описано использование сверточных нейронных сетей (CNN) с акцентом на обработке изображений. Идея этих сетей была почерпнута из биологии, и примеры их успешного применения одними из первых

продемонстрировали мощь нейронных сетей. Основное внимание в главе было уделено задаче классификации, хотя у сверточных сетей имеются и другие сферы применения, такие как обучение признакам без учителя, обнаружение и локализация объектов. Сверточные нейронные сети обучаются иерархическим признакам в различных слоях, причем обучение примитивным формам осуществляется в ранних слоях, а более сложным формам — в поздних. Методы обратного распространения ошибки, применяемые в CNN, тесно связаны с задачами деконволюции (операция, обратная к свертке) и визуализации. В последнее время сверточные нейронные сети используются также в области обработки текста, где они продемонстрировали производительность, позволяющую им конкурировать с рекуррентными нейронными сетями.

8.8. Библиографическая справка

Идея сверточных нейронных сетей была подсказана результатами экспериментов Хубеля и Визеля со зрительной корой кошек [212]. Впоследствии на основе обобщения этих идей была разработана первая сверточная сеть: *LeNet-5* [279]. Одно из первых обсуждений наилучших подходов и принципов построения сверточных нейронных сетей содержится в [452]. Отличный обзор по CNN приведен в [236]. Практическое руководство по арифметике сверток доступно в [109]. Краткое обсуждение прикладных аспектов CNN содержится в [283].

Самым первым набором данных, который нашел широкое применение для тренировки сверточных нейронных сетей, была база данных MNIST, содержащая рукописные цифры [281]. Впоследствии стали популярными крупные наборы данных наподобие *ImageNet* [581]. На протяжении последних пяти лет источником наилучших алгоритмов стали конкурсы *ImageNet (ILSVRC)* [582]. В частности, в качестве примеров нейронных сетей, отлично проявивших себя на различных соревнованиях, можно привести сети *AlexNet* [255], *ZFNet* [556], *VGG* [454], *GoogLeNet* [485] и *ResNet* [184]. Сеть *ResNet*, тесно связанная с магистральными сетями [505], предоставляет итеративную точку зрения на конструирование признаков. Предшественником *GoogLeNet* была архитектура *Network-in-Network (NiN)* [297], продемонстрировавшая ряд полезных принципов проектирования модуля *inception* (таких, как использование операций “бутылочного горлышка”). Некоторые объяснения успешной работы *ResNet* приведены в [185, 505]. Использование модулей *inception* в промежутках между замыкающими соединениями предложено в [537]. Использование стохастической глубины в сочетании с остаточными нейронными сетями обсуждается в [210]. Широкие остаточные сети были предложены в [549]. В родственной архитектуре под названием *FractalNet* [268] используются как короткие, так и длинные пути по сети, но замыкающие соединения не применяются.

Тренировка осуществляется путем исключения подпутей, но на стадии предсказаний используется полная сеть.

Готовые методы извлечения признаков с помощью предварительно обученных моделей обсуждаются в [223,390, 585]. В тех случаях, когда природа приложения существенно отличается от природы данных ImageNet, может иметь смысл извлекать признаки только из нижних слоев предварительно обученных моделей. Это обусловлено тем, что нижние слои часто кодируют более типовые/примитивные признаки наподобие краевых линий и элементарных форм, которые будут работать во многих ситуациях. Подход на основе нормализации локального отклика тесно связан с контрастивной (сопоставительной) нормализацией [221].

В [466] предлагается замена слоя пулинга по максимальному значению сверточным слоем с увеличенным шагом. Отказ от использования слоя пулинга дает ряд преимуществ при создании автокодировщика, поскольку это позволяет использовать в декодировщике сверточный слой с дробным шагом [384]. Дробные шаги помещают нули в строки и столбцы входного объема, если желательно увеличить пространственные размеры результата операции свертки. Иногда применяется также *расширенная свертка* [544], предполагающая помещение нулей в строки/столбцы фильтра (а не входного объема). Связи между деконволюционными сетями и визуализацией на основе градиента описаны в [456, 466]. Простые методы инвертирования признаков, созданных сверточной нейронной сетью, обсуждаются в [104]. В [308] описаны способы оптимальной реконструкции изображения на основе заданного представления признаков. Ранние примеры использования сверточных автокодировщиков приведены в работах [318, 554, 555]. Идеи относительно обучения с учителем также могут быть почерпнуты из ограниченных машин Больцмана. Одна из первых таких идей, в которой используются сети глубокого доверия, представлена в [285]. Использование различных типов деконволюции, визуализации и реконструкции обсуждается в [130, 554–556]. О результатах крупномасштабного исследования в рамках извлечения признаков из изображений по методике обучения без учителя сообщалось в [270].

Существуют способы извлечения признаков по методике обучения без учителя, которые, по всей видимости, работают довольно неплохо. В [76] небольшие фрагменты изображений кластеризуются с помощью алгоритма k -средних с целью генерирования признаков. Центроиды кластеров могут быть использованы для извлечения признаков. Еще одной возможностью является применение случайных весов в качестве фильтров для извлечения признаков [85, 221, 425]. Некоторые рекомендации по этому поводу предложены в [425]. Там же показано, что сочетание свертки и пулинга приобретает свойства частотной избирательности и трансляционной инвариантности даже в условиях использования случайных весов.

Конструирование признаков для извлечения изображений обсуждается в [16]. В последние годы были предложены многочисленные методы, предназначенные для локализации изображений. В этом отношении особенно показательна система *Overfeat* [441], ставшая победителем соревнований *ImageNet* в 2013 году. Для получения результатов высокого качества в этой архитектуре использовался подход на основе скользящего окна. Различные вариации сетей *AlexNet*, *VGG* и *ResNet* также отлично проявили себя на соревнованиях *ImageNet*. Некоторые из ранних методов обнаружения объектов были предложены в [87, 117]. Решение, предложенное в последней из указанных работ, также называют *моделью деформируемых компонент* [117]. И хотя ни нейронные сети, ни глубокое обучение в них не применяются, между этими моделями и CNN могут быть проведены определенные параллели [163]. С наступлением эры глубокого обучения были предложены многочисленные методы, такие как *MCG* [172], *EdgeBoxes* [568] и *SelectiveSearch* [501]. Основным их недостатком является то, что они работают медленно. Недавно был предложен алгоритм *Yolo* [391], который обеспечивает быстрое обнаружение объектов. Однако определенный выигрыш в скорости вычислений достигается лишь за счет ухудшения точности. Тем не менее общая эффективность данного метода все еще остается довольно высокой. Использование сверточных нейронных сетей для сегментации изображений обсуждается в [180]. Методы синтеза текстур и переноса стилей с помощью CNN предложены в [131, 132, 226]. Последние годы ознаменовались огромными достижениями в области распознавания лиц с помощью нейронных сетей. В ранних работах [269, 407] было продемонстрировано, каким образом сверточные сети могут быть использованы для распознавания лиц. Глубокие варианты таких сетей описаны в [367, 474, 475].

Применение CNN для распознавания естественного языка обсуждается в [78, 79, 102, 227, 240, 517]. Для создания начальных данных с богатым набором признаков часто привлекаются методы *word2vec* или *GloVe* [325, 371]. Для классификации текста часто используют сочетание рекуррентных и сверточных сетей [260]. Использование сверточных сетей для классификации текста на символическом уровне рассматривается в [561]. Методы захвата изображений с помощью комбинаций сверточных и рекуррентных сетей описаны в [225, 509]. Использование сверточных нейронных сетей для обработки данных, структурированных на основе графов, обсуждается в [92, 188, 243]. Применение CNN для обработки временных рядов и распознавания речи описано в [276].

С точки зрения сверточных сетей видеоданные могут рассматриваться как пространственно-временное обобщение изображений [488]. Использование трехмерных сверточных нейронных сетей для крупномасштабной классификации обсуждается в [17, 222, 234, 500], тогда как в [17, 222] предложены ранние методы, позволяющие применять трехмерные CNN в целях классификации

видео. Для всех нейронных сетей, ориентированных на классификацию изображений, существуют естественные трехмерные аналоги. Например, обобщение сети VGG на видеоданные с помощью CNN описано в [500]. Как это ни удивительно, результаты, получаемые с помощью трехмерных сверточных нейронных сетей, лишь ненамного лучше результатов однокадровых методов, выполняющих классификацию отдельных кадров видео. Важно отметить, что индивидуальные кадры уже содержат много информации, которую можно использовать для классификации, и добавление движения часто мало что дает в этом плане, если только характеристики движения не имеют существенного значения для различения классов. Другая трудность заключается в том, что наборы видеоданных часто ограничены в размерах по сравнению с теми, которые действительно требуются для создания крупномасштабных систем. Несмотря на то что в [234] собран относительно крупномасштабный набор, созданный на основе миллионов видеороликов YouTube, даже его, по всей видимости, нельзя считать достаточно полным в контексте обработки видеоданных. В конце концов, обработка видео требует применения трехмерных сверток, которые гораздо сложнее двухмерных, используемых для обработки изображений. Вследствие этого зачастую имеет смысл прибегать к сочетанию созданных вручную признаков со сверточными нейронными сетями [514]. Еще одной полезной идеей, которая в последние годы находит свое применение, является понятие оптического потока [53]. Трехмерные сверточные нейронные сети могут быть полезными для классификации видео на протяжении коротких временных промежутков. Еще одна идея общего характера относительно классификации видео заключается в комбинировании сверточных нейронных сетей с рекуррентными нейронными сетями [17, 100, 356, 455]. В [17] был предложен один из первых методов сочетания RNN и CNN. RNN целесообразно использовать в тех случаях, когда требуется классификация, охватывающая длительные временные промежутки. Недавно был предложен метод [21], основанный на однородном сочетании сверточных и нейронных сетей. Его основная идея заключается в том, чтобы сделать каждый нейрон сверточной сети рекуррентным. Этот подход можно рассматривать как непосредственное расширение сверточных нейронных сетей.

8.8.1. Программные ресурсы и наборы данных

Инструменты для глубокого обучения с помощью сверточных нейронных сетей предлагаются во многих пакетах, включая *Caffe* [571], *Torch* [572], *Theano* [573] и *TensorFlow* [574]. Доступны расширения Caffe для Python и MATLAB. Извлечение признаков из Caffe обсуждается в [585]. “Зоопарк” предварительно обученных моделей доступен по адресу, указанному в [586]. Библиотека Theano базируется на Python и предоставляет в качестве интерфейсов такие

высокоуровневые пакеты, как *Keras* [575] и *Lasagne* [576]. Описание реализации CNN с открытым исходным кодом в MATLAB, известной как *MatConvNet*, содержится в [584].

Двумя наиболее популярными наборами данных для тестирования CNN являются *MNIST* и *ImageNet*. Оба этих набора данных подробно описаны в главе 1. Набор данных MNIST довольно удобен в работе благодаря тому, что содержащиеся в нем изображения центрированы и нормализованы. В результате изображения MNIST поддаются точной классификации даже с помощью традиционных методов машинного обучения, поэтому использование сверточных нейронных сетей для работы с ним не является обязательным. С другой стороны, в наборе ImageNet содержатся изображения, представленные в различных перспективах, и для работы с ними необходимо использовать CNN. Тем не менее из-за наличия в наборе ImageNet 1000 категорий изображений и многочисленности самих изображений он не является удачным кандидатом для использования в целях тестирования по причине трудоемкости вычислений. Размеры набора данных CIFAR-10 [583] более умеренны. Этот набор содержит всего лишь 60 000 примеров, в том числе 6000 цветных изображений, разбитых на десять категорий. Каждое изображение в этом наборе имеет размеры $32 \times 32 \times 3$. Следует отметить, что набор CIFAR-10 представляет собой лишь подмножество набора небольших изображений [642], изначально насчитывающего 80 миллионов изображений. Набор данных CIFAR-10 часто применяют для оперативного тестирования перед тем, как приступить к более масштабной тренировке с помощью ImageNet. Набор данных CIFAR-100 — это тот же набор данных CIFAR-10, но разбитый на 100 классов, каждый из которых содержит 600 примеров. Эти 100 классов сгруппированы в 10 суперклассов.

8.9. Упражнения

1. Пусть задан одномерный временной ряд значений 2, 1, 3, 4, 7. Выполните свертку, используя одномерный фильтр 1, 0, 1 и дополнение нулями.
2. Какова будет длина выхода для одномерного временного ряда длиной L в случае использования фильтра размером F ? Какими должны быть размеры дополнения, чтобы удержать размер выхода постоянным?
3. Объем активации имеет размеры $13 \times 13 \times 64$, а фильтр — размеры $3 \times 3 \times 64$. Можно ли выполнить свертку с шагами 2, 3, 4 и 5? Обоснуйте свой ответ в каждом из этих случаев.
4. Рассчитайте размеры пространственных сверточных слоев для каждого из столбцов табл. 8.2. В каждом из этих случаев мы начинаем с объема входного изображения, равного $224 \times 224 \times 3$.

5. Рассчитайте количество параметров в каждом из пространственных слоев для столбца D в табл. 8.2.
6. Загрузите реализацию архитектуры *AlexNet* из любой доступной библиотеки нейронных сетей по своему выбору. Обучите сеть на подмножествах данных переменного размера из набора ImageNet и нарисуйте график зависимости коэффициента ошибок топ-5 от размера данных.
7. Вычислите свертку входного объема, представленного в левой верхней части рис. 8.2, с помощью детектора горизонтальных краев, показанного на рис. 8.1, б. Используйте шаг 1 без дополнения.
8. Выполните пулинг 4×4 с шагом 1 по отношению к входному объему, представленному в левой верхней части рис. 8.4.
9. Опишите различные типы предварительного обучения, которые могут применяться в целях аннотирования изображений (см. раздел 7.7.1).
10. Предположим, вы располагаете большим набором данных, содержащим пользовательские рейтинговые оценки различных изображений. Покажите, каким образом можно объединить сверточную нейронную сеть с идеями коллаборативной фильтрации, которые обсуждались в главе 2, для создания гибридной коллаборативно-рекомендательной системы, ориентированной на содержимое.

Глава 9

Глубокое обучение с подкреплением

Вознаграждением за страдания является полученный опыт.

Гарри Трумэн

9.1. Введение

В реальной жизни люди учатся не на заранее подготовленных коллекциях тренировочных (учебных) примеров. Обучение человека представляет собой непрерывный процесс приобретения опыта, в ходе которого он постоянно принимает определенные решения, а проистекающие из них выгоды и потери, обусловленные ответной реакцией *окружения*, служат подсказками, облегчающими принятие решений в будущем. Иными словами, разумные существа учатся, действуя *методом проб и ошибок под управлением вознаграждения*. К тому же интеллект и инстинкты человека в значительной мере определяются генетическим кодом, совершенствовавшимся на протяжении миллионов лет в ходе другого управляемого средой процесса, который называется *эволюцией*. Поэтому почти все, что связано с биологическим интеллектом, в той или иной форме является результатом взаимодействия живого существа с его окружением методом проб и ошибок. В своей чрезвычайно интересной книге по искусственному интеллекту [453] Герберт Саймон выдвинул следующую *гипотезу о поведении муравьев*, которая, вероятно, применима и к людям.

“В том, что касается принципов своего поведения, муравей весьма прост. Кажущаяся сложность его поведения во времени в основном отражает сложность внешней среды, в которой он существует”.

Людей тоже можно считать простой системой, поскольку они постоянны, эгоистичны и действуют в интересах личной выгоды. Этот простой факт можно рассматривать в качестве первоосновы всех свойств биологического интеллекта. Так как задачей искусственного интеллекта является имитация

биологического интеллекта, то при поиске способов упрощения чрезвычайно сложных алгоритмов природного обучения вполне естественно черпать вдохновение в успешности биологического рационализма.

Процесс проб и ошибок, в ходе которого система учится взаимодействовать со сложным окружением таким образом, чтобы получить вознаграждение за свои действия, на языке машинного обучения называется *обучением с подкреплением* (reinforcement learning). Движущей силой этого процесса является потребность максимизировать ожидаемое вознаграждение за отведенное время. Через обучение с подкреплением может пролегать путь к созданию подлинно интеллектуальных *агентов*, таких как игровые движки, беспилотные автомобили и даже интеллектуальные роботы, способные взаимодействовать со своим окружением. Проще говоря, обучение с подкреплением открывает путь к созданию искусственного интеллекта в любой форме. Этот путь пока еще не пройден, однако за последние годы в данном направлении были сделаны огромные шаги, которые привели к получению впечатляющих результатов.

1. Системы глубокого обучения обучались видеоиграм, используя в качестве обратной связи лишь пиксели видеоконсоли. Классическим примером является консоль Atari 2600, поддерживающая множество игр. Входом для системы глубокого обучения со стороны платформы Atari служат экранные пиксели текущего состояния игры. Алгоритм обучения с подкреплением предсказывает действия на основании состояния экрана видеоигры и вводит их на консоли Atari. Первоначально компьютерный алгоритм совершает множество ошибок, что отражается на виртуальном вознаграждении, начисляемом консолю. Но по мере того как обучаемая система приобретает опыт, обучаясь на своих ошибках, она принимает все лучшие решения. Точно так же обучаются видеоиграм и люди. Один из последних алгоритмов для платформы Atari продемонстрировал превосходство над человеком в большом количестве видеоигр [165, 335, 336, 432]. Видеоигры — отличный испытательный полигон для алгоритмов обучения с подкреплением, поскольку их можно рассматривать как крайне упрощенные представления вариантов, между которыми приходится делать выбор в различных ситуациях, требующих принятия решений. Проще говоря, видеоигры — это подобие реального мира в миниатюре.
2. Компания DeepMind обучила свой алгоритм *AlphaGo* [445] игре го, используя систему вознаграждений за результаты ходов в играх как между компьютером и человеком, так и между компьютерами. Игра го отличается сложностью и в значительной мере требует проявления человеческой интуиции, а большое дерево возможностей (по сравнению с другими играми, такими как шахматы) делает ее невероятно трудным кандидатом для разработки соответствующего игрового алгоритма. Программа

AlphaGo не только добилась убедительных побед над всеми лучшими профессиональными игроками в го, против которых она играла [602, 603], но и внесла инновации в свойственный человеку стиль игры, используя необычные стратегии, позволившие победить этих игроков. Упомянутые инновации явились результатом приобретения опыта, накопленного программой *AlphaGo* в процессе игры с самой собой под управлением вознаграждения. Недавно этот подход был обобщен на шахматы, что позволило одержать убедительную победу над одним из лучших обычных движков [447].

3. В последние годы глубокое обучение с подкреплением проникло в беспилотные автомобили, где оно применяется для принятия решений на основе обратной связи с различными датчиками, установленными на машине. Несмотря на то что для этих целей обычно используют обучение с учителем (или *имитационное обучение*), вариант обучения с подкреплением всегда признавался в качестве вполне жизнеспособного решения [604]. Теперь в процессе управления автомобилем эти системы совершают меньше ошибок, чем свойственно людям.
4. Поиск путей создания самообучающихся роботов — задача обучения с подкреплением [286, 296, 432]. Например, обеспечить перемещение робота в условиях, требующих быстрого принятия решений и сноровки, необычайно трудно. Обучение робота ходьбе можно проводить в рамках задачи обучения с подкреплением, если не демонстрировать роботу, что собой представляет ходьба. В парадигме обучения с подкреплением мы лишь стимулируем робота к тому, чтобы он переместился из точки А в точку В, максимально эффективно используя свои суставы и двигатели [432]. Действуя методом проб и ошибок и получая соответствующее вознаграждение, робот учится перекатываться, ползти и в конечном счете ходить.

Обучение с подкреплением подходит для задач, *которые не требуют сложных вычислений, но которые трудно определить*. Например, можно легко вычислить действия игрока в конце сложной игры наподобие шахмат, но трудно специфицировать его действия в каждой возможной ситуации. Как и в случае биологических организмов, обучение с подкреплением позволяет *упростить обучение сложному поведению*, определив лишь функцию вознаграждения и позволив алгоритму обучаться сложному поведению, стремясь максимизировать вознаграждение. Сложность поведения автоматически наследуется от сложности окружения. В этом и состоит суть гипотезы Герберта Саймона о поведении муравьев [453], которая была процитирована в начале главы. Системы обучения с подкреплением по своей сути являются *сквозными системами*, в которых сложная задача не разбивается на меньшие компоненты, а рассматривается сквозь призму простого вознаграждения.

Простейшим примером условий, подходящих для применения обучения с подкреплением, может служить задача о *многоруком бандите*, в которой игрок пытается максимизировать свой выигрыш, выбирая один из многих игровых автоматов. Игрок рассчитывает на то, что (ожидаемое) вознаграждение не может быть одинаковым для всех игровых автоматов, и поэтому имеет смысл использовать игровой автомат с наибольшим ожидаемым выигрышем. Поскольку выигрыши, ожидаемые от разных автоматов, не известны заранее, игрок должен *исследовать* различные игровые автоматы, поочередно играя на них и используя получаемые при этом знания для максимизации выигрыша. Несмотря на то что изучение свойств конкретного игрового автомата может предоставить определенные знания относительно выигрыша, который он способен обеспечить, с этим также связан риск напрасной потери денег в процессе игры. Алгоритмы многорукого бандита дают нам тщательно продуманные стратегии игры, позволяющие достичь компромисса между исследованиями и эксплуатацией получаемых знаний. Однако в этой упрощенной постановке задачи каждое решение относительно выбора игрового автомата принимается в тех же условиях, в которых принималось предыдущее решение. В случае видеоигр и беспилотных автомобилей с сенсорными входами (предоставляющими, например, данные об игровом экране или условиях дорожного движения), которые определяют *состояние* системы, это не совсем так. Системы глубокого обучения отлично справляются с преобразованием данных от сенсорных входов в действия, *зависящие от состояния*, обертывая процесс обучения в общую схему “исследование/эксплуатация”.

Структура главы

В следующем разделе вводится концепция многоруких бандитов, которая служит одним из простейших вариантов постановки задач без запоминания состояний в обучении с подкреплением. Понятие состояний вводится в разделе 9.3. Q-метод обучения вводится в разделе 9.4. Градиентные методы оптимизации стратегии обсуждаются в разделе 9.5. Использование стратегий на основе поиска по дереву методом Монте-Карло описано в разделе 9.6. Ряд типичных примеров приведен в разделе 9.7. Вопросы безопасности, связанные с методами глубокого обучения с подкреплением, рассматриваются в разделе 9.8. Резюме главы приведено в разделе 9.9.

9.2. Алгоритмы без запоминания состояний: многорукие бандиты

Вернемся к задаче игрока, который строит свою стратегию игры со многими игровыми автоматами на основании предыдущего опыта. Исходя из полученных

ранее результатов, он полагает, что от одного из автоматов можно ожидать большее вознаграждение, чем от остальных, и пытается применить этот опыт, одновременно продолжая исследовать другие возможные варианты. Случайный выбор автоматов приводит к дополнительным затратам, но способствует расширению опыта. Попытка сыграть с каждым автоматом небольшое количество раз для того, чтобы выбрать тот из них, который приносит наибольшее вознаграждение, может приводить к решениям, не являющимся оптимальными в долгосрочной перспективе. Каким же образом можно достигнуть компромисса между применением и накоплением опыта? Следует учесть, что вознаграждение, получаемое в результате совершения определенного действия при каждой попытке, подчиняется тому же распределению, что и для всех предыдущих попыток, и поэтому понятия “состояние” для такой системы не существует. Подобный подход представляет собой упрощенный вариант традиционного обучения с подкреплением, в котором понятие состояния играет очень важную роль. В компьютерных играх перемещение курсора в определенном направлении приносит вознаграждение, которое в значительной степени зависит от текущего состояния видеоигры.

Существует ряд стратегий, которые игрок может задействовать для достижения компромисса между исследованием пространства поиска и применением уже имеющихся данных. Далее мы кратко опишем некоторые из наиболее распространенных стратегий, которые применяются в системах, работающих по принципу “многорукого бандита”. Ознакомление с ними будет весьма поучительным, поскольку это даст вам первоначальное представление о базовых идеях и архитектуре обучения с подкреплением. В действительности некоторые из рассмотренных ниже алгоритмов, не запоминающих состояние, используются в качестве подпрограмм в общих реализациях обучения с подкреплением. Именно по этой причине важно изучить упрощенную постановку задачи.

9.2.1. Наивный алгоритм

В этом подходе игрок совершает фиксированное количество попыток с каждым игровым автоматом во время фазы исследования. В дальнейшем, во время фазы эксплуатации, постоянно используется автомат, который обеспечил наибольший выигрыш. Несмотря на то что такой подход поначалу может показаться разумным, у него имеется ряд недостатков. Главная проблема заключается в трудности определения того, какое количество попыток может считаться достаточным для того, чтобы с уверенностью предсказать, какой именно игровой автомат будет приносить наибольший выигрыш. Процесс оценки величины вознаграждения может занять длительное время, особенно в тех случаях, когда события выигрыша происходят реже, чем события проигрыша. Многократные пробные попытки приведут к тому, что значительные усилия будут затрачены

на неоптимальные стратегии. К тому же, если в конечном счете будет выбрана неверная стратегия, игрок постоянно будет задействовать игровой автомат, который не обеспечивает получение максимального выигрыша. Поэтому подход, основанный на использовании фиксированной стратегии, не является наилучшим с практической точки зрения.

9.2.2. Жадный алгоритм

Эпсилон-жадный (ϵ -жадный) алгоритм ориентирован на выбор наилучшей стратегии сразу же, как только это становится возможным, без ненужных пробных попыток. Основная идея заключается в выборе случайного игрового автомата для доли попыток, равной ϵ . Эти разведочные попытки тоже выбираются случайным образом (с вероятностью ϵ) из общего количества возможных попыток и чередуются с отобранными для использования оптимальными попытками. В оставшейся доле попыток ($1 - \epsilon$) задействуется игровой автомат, который к этому времени дает наибольший средний выигрыш. Важное преимущество такого подхода заключается в том, что он гарантированно исключает возможность постоянного использования неверной стратегии. Кроме того, поскольку стадия эксплуатации наступает уже на раннем этапе, часто возникают ситуации, когда наилучшая стратегия применяется большую часть времени.

Значение ϵ — параметр алгоритма. Например, на практике можно установить $\epsilon = 0,1$, хотя наилучшее значение ϵ будет зависеть от конкретного приложения. Зачастую трудно заранее сказать, какое значение ϵ следует выбрать в конкретных условиях. Тем не менее, чтобы использование описанного подхода принесло максимальную пользу, значение ϵ должно быть относительно небольшим. Однако при малых значениях ϵ идентификация подходящего игрового автомата может занять длительное время. Обычно применяют метод *отжига* (annealing), когда сначала используют большие значения параметра ϵ , а затем постепенно уменьшают их с течением времени.

9.2.3. Методы верхней границы

Несмотря на то что в динамических условиях ϵ -жадная стратегия лучше наивной, она все еще остается достаточно неэффективной в отношении изучения новых игровых автоматов. В стратегиях ограничения сверху игрок не использует знания о среднем вознаграждении, получаемом от игрового автомата. Вместо этого игрок занимает оптимистическую позицию относительно ценности игровых автоматов, которые еще не были достаточно испытаны, и поэтому выбирает игровой автомат с наилучшей *статистической верхней границей* вознаграждения. Следовательно, верхнюю границу U_i оценки вознаграждения от игрового автомата i можно рассматривать как сумму ожидаемых вознаграждений Q_i и доверительных интервалов длиной C_i :

$$U_i = Q_i + C_i \quad (9.1)$$

Значение C_i является своего рода бонусом за повышенную неопределенность умонастроения игрока относительно данного игрового автомата. Значение C_i пропорционально стандартному отклонению от *среднего* вознаграждения сделанных до сих пор попыток. В соответствии с центральной предельной теоремой это стандартное отклонение обратно пропорционально квадратному корню из количества испытаний, которому подвергался игровой автомат i (в предположении о независимости выборок с идентичным распределением). Можно оценить среднее значение μ_i и стандартное отклонение i -го игрового автомата, а затем задать C_i равным $K \cdot \sigma_i / \sqrt{n_i}$, где n_i — количество испытаний, которым подвергался i -й игровой автомат. Здесь K определяет уровень доверительного интервала. Следовательно, редко тестируемые автоматы обычно будут получать большие верхние оценки (в силу большей ширины доверительных интервалов C_i) и поэтому будут чаще испытываться.

В отличие от ε -жадного алгоритма, испытания уже не делятся на категории исследования и эксплуатации. Процесс отбора игровых автоматов с наибольшей верхней границей отдачи характеризуется дуальным эффектом кодирования как исследовательских, так и эксплуатационных аспектов в каждом испытании. Достижением компромисса между исследованием и эксплуатацией можно управлять, задавая определенный уровень статистического доверия. Выбор $K = 3$ приводит к 99,99% доверительному интервалу для верхней границы в предположении гауссовского распределения вероятности. В общем случае увеличение K будет предоставлять большие бонусы C_i за неопределенность, тем самым увеличивая долю исследований в игровых попытках по сравнению с алгоритмом, в котором используются меньшие значения K .

9.3. Базовая постановка задачи обучения с подкреплением

Рассмотренные в предыдущем разделе алгоритмы многорукого бандита не запоминают состояния. Иными словами, в каждый момент времени решения принимаются в условиях одного и того же окружения, и действия, совершенные в прошлом, влияют лишь на знания агента (но не на само окружение). В типичных условиях обучения с подкреплением, таких как видеоигры или беспилотные автомобили, к которым применимо понятие *состояние*, это не так.

Как правило, в обучении с подкреплением вознаграждение связывается с каждым отдельным действием. В процессе видеоигры вы получаете вознаграждение не только за то, что сделали конкретный ход. Вознаграждение за ход зависит от результатов всех предыдущих ходов, внедренных в *состояние* окружения. В случае видеоигры или беспилотного автомобиля требуется другой

способ начисления вознаграждения в определенном состоянии системы. Например, в случае беспилотного автомобиля вознаграждение за резкий поворот в обычных условиях будет отличаться от вознаграждения за то же самое действие, совершенное при угрозе столкновения. Иными словами, нам нужен способ количественной оценки каждого действия, специфический для конкретного состояния системы.

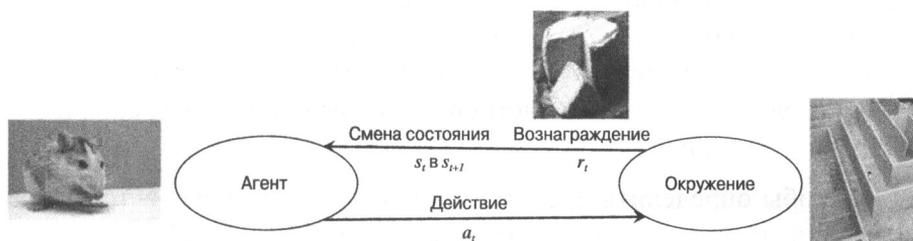
В обучении с подкреплением есть *агент*, который взаимодействует с окружением с помощью *действий*. Например, в видеоигре агент — это игрок, а действие — это движение джойстика в определенном направлении. Окружение — это вся сцена игры как таковой. Действия изменяют окружение и переводят его в новое состояние. В видеоигре состояние охватывает все переменные, описывающие текущую позицию игрока в конкретный момент времени. Окружение вознаграждает агента в зависимости от того, насколько хорошо решается задача обучения. Например, вознаграждением в игре является начисление очков агенту. Иногда вознаграждение может быть непосредственно связано не с отдельным действием, а с комбинацией действий, предпринятых какое-то время тому назад. Например, игрок мог дальновидно поместить курсор в особенно удачной точке еще несколько ходов назад, так что последующие ходы не будут оказывать существенного влияния на величину вознаграждения. Кроме того, вознаграждение за действие (например, манипулирование рычагом игрового автомата) само по себе может не быть определяющим в данном состоянии. *Одной из главных задач обучения с подкреплением является идентификация ценности действий в различных состояниях, независимо от временных и стохастических характеристик вознаграждения.*

Процесс обучения помогает агенту выбирать действия на основании их внутренней ценности в различных состояниях. Этот общий принцип применим ко всем формам обучения с подкреплением в биологических системах. Достаточно вспомнить эксперименты с обучением мышей прохождению лабиринтов для получения вознаграждения. Вознаграждение, заработанное мышью, зависит от всей последовательности действий, а не только от самого последнего из них. Как только заработано вознаграждение, синаптические веса в мозге мыши перестраиваются в соответствии с тем, каким образом должны использоваться сенсорные входы для принятия будущих решений относительно передвижения в лабиринте. Это в точности тот подход, который применяется в глубоком обучении с подкреплением, когда нейронная сеть используется для прогнозирования действий на основании сенсорных входов (например, пикселей видеоигры). Отношения между агентом и окружением показаны на рис. 9.1.

Весь набор состояний и действий вместе с правилами перехода из одного состояния в другое называют *марковским процессом принятия решений*. Основным свойством марковского процесса принятия решений является то, что в

любой конкретный момент времени состояние содержит всю информацию, необходимую окружению для выполнения перехода в другое состояние и назначения вознаграждения на основании действий агента. Конечные марковские процессы принятия решений (как, например, в игре крестики-нолики) завершаются за конечное число шагов, в связи с чем такие процессы называют *эпизодическими*. Отдельный эпизод марковского процесса представляет собой конечную последовательность действий, состояний и вознаграждений. Примером может служить последовательность длиной $(n + 1)$:

$$s_0 a_0 r_0 s_1 a_1 r_1 \dots s_t a_t r_t \dots s_n a_n r_n.$$



1. Агент (мышь) предпринимает действие a_t (левый поворот в лабиринте) из состояния (позиции) s_t
2. Окружение назначает мыши вознаграждение r_t (сыр/нет сыра)
3. Агент переходит в состояние s_{t+1}
4. Нейроны мыши обновляют синаптические связи на основании того, было ли действие вознаграждено сыром

Итог: со временем агент обучается совершению действий, приносящих вознаграждение

Рис. 9.1. Общая схема обучения с подкреплением

Обратите внимание на то, что s_t — это состояние до выполнения действия a_t , а выполнение действия a_t приводит к вознаграждению r_t и переходу в состояние s_{t+1} . Это соглашение о временных метках используется на протяжении всей главы (а также в некоторых других источниках), хотя в книге Саттона и Барто [483] вознаграждение r_{t+1} соответствует действию a_t в состоянии s_t (что несколько изменяет индексы во всех результатах). В бесконечных марковских процессах (например, в случае непрерывно функционирующих роботов) отсутствуют эпизоды конечной длины, и поэтому их называют *неэпизодическими*.

Примеры

Несмотря на то что состояние системы ссылается на полное описание окружения, на практике часто прибегают к различным приближениям. Например, в видеоигре Atari состояние системы может определяться окнами снимков игры конечной длины. Соответствующие примеры приведены ниже.

1. *Игра крестики-нолики, шахматы или го*. Состояние — это позиция на доске в любой момент времени, а действия соответствуют ходам, выполненным агентом. Вознаграждением являются оценки +1, 0 или -1

(соответствующие выигрышу, ничьей и проигрышу), получаемые в конце игры. Однако вознаграждение часто не следует сразу же за совершением стратегически значимых действий.

2. *Движения робота.* Состояние соответствует текущей конфигурации сочленений робота и его позиции. Действия соответствуют моментам сил, действующих в сочленениях робота. Вознаграждение в каждый момент времени является функцией, зависящей от того, сохраняет ли робот устойчивое вертикальное положение, а также от величины перемещения из точки А в точку В.
3. *Беспилотные автомобили.* Состояния соответствуют данным, которые поступают от датчиков, установленных на автомобиле, а действия соответствуют манипуляциям рулем, педалями газа и тормоза. Вознаграждение определяется вручную конструируемой функцией движения и безопасности автомобиля.

Обычно, чтобы определить представления состояний и соответствующих вознаграждений, требуется приложить определенные усилия. Но, после того как выбор сделан, схема организации обучения с подкреплением сводится к сквозным системам.

9.3.1. Трудности обучения с подкреплением

Обучение с подкреплением представляет собой более трудную задачу по сравнению с традиционными формами обучения с учителем в силу следующих причин.

1. При получении вознаграждения (например, в результате выигрыша шахматной партии) точные вклады в него каждого из действий остаются неизвестными. Основной проблемой обучения с подкреплением является *способ определения величины вознаграждения*. К тому же вознаграждение может иметь вероятностную природу (и зависеть, например, от того, рычаг какого игрового автомата был задействован), в связи с чем может поддаваться лишь приближенной оценке на основании данных.
2. Система обучения с подкреплением может иметь очень большое количество состояний (таких, например, как всевозможные позиции фигур на шахматной доске), поэтому она должна быть способна принимать решения относительно состояний, с которыми ранее не сталкивалась. Задача обобщения модели является основной функцией глубокого обучения.
3. Конкретный выбор действия влияет на сбор данных, учитываемых при определении будущих действий. Как и в ситуации с многоруким бандитом, необходимо стремиться к достижению естественного компромисса между

разведочными действиями и эксплуатацией полученных знаний. Если действия предпринимаются лишь для исследования возможного вознаграждения, которое присуждается определенным действиям, то игрок при этом будет нести дополнительные расходы. С другой стороны, придерживаясь уже изученных действий, можно получать неоптимальные решения.

4. Обучение с подкреплением объединяет понятия коллекции данных и обучения. Реалистическая имитация больших физических систем, таких как роботы и беспилотные автомобили, ограничена необходимостью физического выполнения этих задач и фиксации реакции окружения на совершаемые агентом действия в условиях, когда сбои могут приводить к ситуациям, представляющим реальную опасность для окружения. Во многих случаях ранние этапы обучения характеризуются небольшим количеством успехов и большим количеством неудач. *Невозможность сбора достаточного объема данных в реальных условиях, не ограниченных имитируемыми или игровыми средами, является, вероятно, наибольшей трудностью, с которой приходится сталкиваться в обучении с подкреплением.*

В следующем разделе вы познакомитесь с простым алгоритмом обучения с подкреплением и ролью методов глубокого обучения.

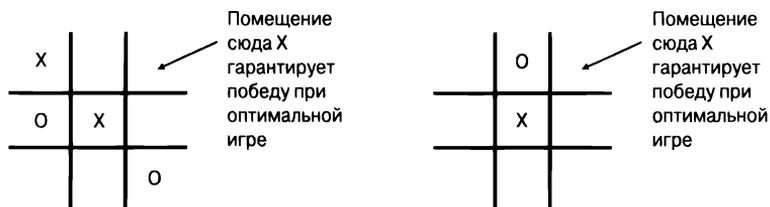
9.3.2. Простой алгоритм обучения с подкреплением для игры крестики-нолики

Рассмотренный в предыдущем разделе ϵ -жадный алгоритм без запоминания состояний можно обобщить на игру в крестики-нолики. В этом случае каждая позиция на доске является состоянием, а действия соответствуют вставке символа 'X' или 'O' в разрешенную позицию. Количество возможных позиций на доске размером 3×3 ограничено сверху значением $3^9 = 19\,683$, что соответствует различным вариантам расположения одного из трех символов ('X', 'O' и пробел) в каждой из 9 позиций. Теперь вместо оценки ценности, или значимости, каждого действия (не имеющего состояния), как в случае многорукого бандита, мы можем оценивать ценность каждой пары "состояние — действие" (s, a) на основании предыстории действия a в состоянии s в игре против фиксированного противника. Быстрым выигрышам отдается предпочтение с коэффициентом $\gamma < 1$, поэтому *ненормализованная* ценность действия a в состоянии s получает приращение γ^{r-1} в случае выигрышей и $-\gamma^{r-1}$ в случае проигрышей после r ходов (включая текущий). Ничья оценивается нулевым значением. Коэффициент γ также учитывает тот факт, что в реальных задачах значимость действия убывает со временем. В этом случае таблица значимости действий обновляется лишь после выполнения всех ходов в игре (хотя методы, которые будут рассмотрены далее, допускают *оперативное* обновление после каждого хода).

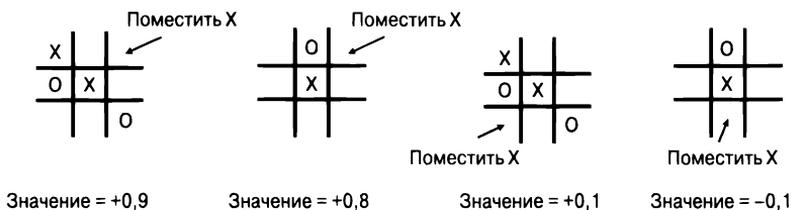
Нормализованные значения ценности действий, содержащиеся в таблице, получаются путем деления ненормализованных значений на количество выполненных обновлений пары “состояние — действие” (которая поддерживается отдельно). Эта таблица начинается с небольших случайных значений, а в качестве действия a в состоянии s с вероятностью $1 - \epsilon$ выбирается (с использованием процедуры жадного поиска) действие с наибольшим нормализованным значением ценности или же случайное действие в остальных случаях. Все сделанные ходы оцениваются по завершении каждой игры. Со временем будут обучены все пары “состояние — действие”, а результирующие ходы будут адаптированы к фиксированному противнику. Кроме того, для оптимальной генерации таблиц можно использовать игру алгоритма с самим собой. В случае этого варианта обновление табличных значений $\{-\gamma^r, 0, \gamma^r\}$ выполняется в зависимости от выигрыша/ничьей/проигрыша с точки зрения игрока, за которого делаются ходы. При выводе суждений выбирается ход, имеющий наибольшее нормализованное значение ценности с точки зрения этого игрока.

9.3.3. Роль глубокого обучения и алгоритм “соломенного пугала”

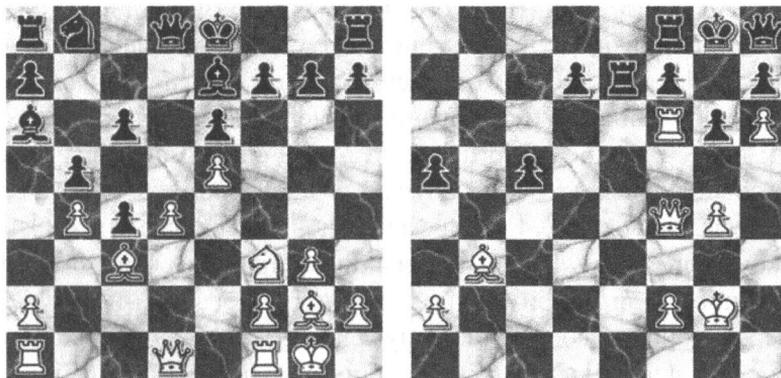
В вышеупомянутом алгоритме для игры в крестики-нолики не используются ни нейронные сети, ни глубокое обучение, и то же самое касается многих других традиционных алгоритмов обучения с подкреплением [483]. Важнейшей целью ϵ -жадного алгоритма является обучение долгосрочной ценности, свойственной каждой паре “состояние — действие”, поскольку вознаграждение будет получено спустя длительное время после выполнения значимых действий, обусловивших этот результат. Целью процесса обучения является идентификация тех действий, которые, будучи выполненными в определенном состоянии, оказались действительно благоприятными в долгосрочной перспективе. Например, всего один удачный ход, создающий ловушку для противника в игре крестики-нолики, может гарантировать выигрыш. Примеры двух таких сценариев приведены на рис. 9.2, а, (хотя ловушка справа менее очевидна). Поэтому необходимо приписывать определенную ценность не только последнему ходу, который привел к выигрышу, но и *стратегически* удачным ходам, указанным в таблице пар “состояние — действие”. Метод проб и ошибок, основанный на ϵ -жадном алгоритме, о котором говорилось в разделе 9.3.2, действительно будет присваивать высокие значения хитрым ловушкам. Примеры типичных значений из этой таблицы приведены на рис. 9.2, б. Обратите внимание на то, что менее очевидной ловушке из тех, которые приведены на рис. 9.2, а, присвоено чуть меньшее значение, поскольку ходы, гарантирующие выигрыш через длительные игровые отрезки, оцениваются с коэффициентом γ , в связи с чем ϵ -жадному методу проб и ошибок будет труднее обнаружить победу после установки ловушки.



а) Два примера позиций из игры крестики-нолики, гарантированно приводящих к победе



б) Четыре записи из таблицы пар “состояние — действие” в игре крестики-нолики. Метод проб и ошибок обеспечивает обучение тому, что ходы, гарантированно ведущие к победе, имеют высокие значения



в) Позиции из двух различных игр между программами Alpha Zero (белые) и Stockfish (черные) [447]. На рисунке слева белые жертвуют пешку, чтобы заблокировать белого слона черных их собственными пешками. Эта стратегия в конечном счете привела к победе белых спустя большое количество ходов, намного превышающее горизонт обычной шахматной программы наподобие Stockfish. На рисунке справа белые жертвовали фигуры, чтобы загнать черных в позицию, в которой любой их ход лишь ухудшает положение. Постепенное усиление позиционного преимущества — отличительный признак лучших шахматистов-людей, но не шахматных программ наподобие Stockfish, в которых создаваемые вручную оценки иногда не способны улавливать тонкие различия в позициях. Нейронная сеть обучения с подкреплением, которая использует состояние доски в качестве входа, оценивает позиции интегральным способом без каких-либо априорных предположений. Данные, генерируемые методом проб и ошибок, являются единственным источником опыта для обучения очень сложной функции оценки, которая косвенным образом закодирована в параметрах нейронной сети. Поэтому обученная сеть обобщает полученный опыт на новые позиции. Это аналогично тому, как человек учится лучше оценивать позиции на доске, изучая предыдущие игры

Рис. 9.2. Обширные пространства состояний, как в случае (в), требуют использования глубокого обучения

Основной проблемой такого подхода является то, что во многих задачах обучения с подкреплением количество параметров оказывается слишком большим для того, чтобы их можно было представить в табличном виде. Например, количество возможных позиций в шахматной игре настолько велико, что набор всех позиций, известных человечеству, составляет микроскопическую долю всех допустимых позиций. Описанный в разделе 9.3.2 алгоритм в действительности представляет собой улучшенную разновидность механического обучения, т.е. *заучивания*, при котором имитация методом Монте-Карло используется для уточнения и запоминания долгосрочных значений *уже встречавшихся* состояний. Обучение значению ловушки в игре крестики-нолики происходит лишь потому, что в результате имитации игры методом Монте-Карло *в точности такая же позиция* множество раз приводила к победе. В большинстве сложных случаев, таких как шахматы, знания, полученные на основе предыдущего опыта, должны обобщаться на состояния, которые прежде не предъявлялись системе. Любая форма обучения (в том числе и обучение с подкреплением) приносит максимальную пользу, если она используется для обобщения известного опыта на неизвестные ситуации. В подобных случаях разновидности обучения с подкреплением, основанные на применении таблиц, оказываются совершенно непригодными. Модели глубокого обучения играют роль *функций-аппроксиматоров*. Вместо обучения и табулирования значений всех ходов во всех позициях (методом проб и ошибок с соответствующим способом назначения вознаграждения) значение каждого хода обучается как функция входного состояния на модели, обученной с использованием исходов для предыдущих позиций. Без такого подхода применимость обучения с подкреплением была бы ограничена лишь такими простейшими ситуациями, как игра “крестики-нолики”.

Например, алгоритм “соломенного пугала” (кстати, не очень хороший) мог бы задействовать для шахмат тот же ϵ -жадный алгоритм, который был описан в разделе 9.3.2, но при этом вычислять значения действий, используя состояние доски в качестве входа для сверточной нейронной сети. В этом случае выходом является позиция на доске. Эпсилон-жадный алгоритм имитируется до завершения с выходными значениями, а эталонное значение каждого хода при такой имитации выбирается из набора $\{\gamma^{r-1}, 0, -\gamma^{r-1}\}$, в зависимости от выигрыша/ничьей/проигрыша и количества ходов r до завершения игры (включая текущий ход). Теперь вместо обновления табличных пар “состояние — действие” обновляются параметры нейронной сети, что достигается обработкой каждого хода как тренировочной точки. Позиция на доске является входом, а выход нейронной сети сравнивается с эталонным значением из набора $\{\gamma^{r-1}, 0, -\gamma^{r-1}\}$ для обновления параметров. При выводе суждений можно использовать ход с наилучшей выходной оценкой (с учетом результатов некоторого минимаксного опережающего просмотра).

Несмотря на то что вышеупомянутый подход чересчур наивен, на его основе была разработана сложная система *Alpha Zero* с поиском по дереву методом Монте-Карло, которая недавно была обучена [447] игре в шахматы. Два примера позиций [447] из различных игр, сыгранных во время матча между *Alpha Zero* и обычной шахматной программой *Stockfish-8.0*, приведены на рис. 9.2, в. В шахматной позиции, представленной слева, система обучения с подкреплением делает *стратегически* дальновидный ход, блокирующий слона противника за счет жертвы пешки, чему большинство компьютерных алгоритмов оценки ходов на основе вручную составленной таблицы не отдало бы предпочтения. В позиции, изображенной справа, *Alpha Zero* жертвует две пешки и разменивает фигуры для того, чтобы постепенно довести черных до цугцванга, когда все их фигуры парализованы. Несмотря на то что *Alpha Zero* (вероятнее всего) никогда не сталкивалась с данными конкретными позициями в процессе тренировки, ее система глубокого обучения способна извлекать соответствующие признаки и шаблоны на основе предыдущего опыта, накопленного в процессе обучения другим позициям. В данном конкретном случае нейронная сеть, по всей видимости, сумела оценить предпочтительность пространственных шаблонов, представляющих тонкие позиционные факторы, по сравнению с материальными факторами (во многом подобно нейронной сети человека).

В реальных задачах состояния часто описывают, используя сенсорные входы. Система глубокого обучения использует входное представление состояния для обучения значениям (полезности) тех или иных действий (например, выполнения хода в игре) вместо таблицы пар “состояние — действие”. Даже если входное представление состояния (например, пиксели) довольно примитивно, нейронные сети мастерски выжмут из него нужную скрытую информацию. Это напоминает то, как биологическая нейронная сеть человека обрабатывает данные, поступающие от органов чувств, для определения состояния окружающего мира и принятия решений относительно дальнейших действий. Мы не располагаем таблицей с парами “состояние — действие” для каждой возможной жизненной ситуации. Парадигма глубокого обучения преобразует таблицу пар “состояние — действие”, имеющую угрожающие размеры, в параметризованную модель, сопоставляющую пары “состояние — действие” со значениями, которые легко поддаются обучению с помощью обратного распространения ошибки.

9.4. Бутстрэппинг для обучения функции оценки

Простое обобщение ϵ -жадного алгоритма на игру крестики-нолики (см. раздел 9.3.2) — довольно наивный подход, который не работает в случае *неэпизодических* задач. В такой эпизодической игре, как крестики-нолики, для характеристики полного и окончательного вознаграждения можно использовать последовательность, состоящую самое большее из девяти ходов. В условиях

неэпизодических задач, как в случае роботов, марковский процесс принятия решений может не быть конечным или же может иметь большую длину. В этих условиях создание образцов эталонного вознаграждения путем семплирования по методу Монте-Карло затрудняется, в связи с чем может оказаться более желательным использование оперативного обновления параметров. Это достигается с помощью методологии *бутстрэппинга*.

Примечание 9.4.1 (бутстрэппинг). *Рассмотрим марковский процесс принятия решений, в котором мы предсказываем значения (например, долгосрочные вознаграждения) для каждого момента времени (временного шага). Если мы можем частично имитировать будущее для улучшения предсказания в текущий момент времени, то потребность в значениях, относящихся к каждому моменту времени, отпадает. Это улучшенное предсказание можно применить в качестве эталонного значения для модели, не располагающей знаниями о будущем.*

Например, в программе Сэмюэля для игры в шашки [421] использовалось различие в оценке текущей позиции и минимаксной оценке, полученной путем просмотра нескольких ходов наперед с помощью той же функции (ошибка предсказания), для обновления функции оценки. Идея заключается в том, что минимаксная оценка, полученная путем опережающего просмотра, более предпочтительна по сравнению с той, которая получена без такого просмотра, и поэтому она может служить в качестве “эталонного значения” для вычисления ошибки.

Рассмотрим марковский процесс принятия решений со следующей последовательностью состояний, действий и вознаграждений:

$$s_0 a_0 r_0 s_1 a_1 r_1 \dots s_t a_t r_t \dots$$

Например, в видеоигре каждое состояние s_t может представлять историческое окно пикселей [335] с представлением признака \bar{X}_t . Чтобы учесть (возможные) отложенные вознаграждения действий, вычисляется кумулятивное вознаграждение R_t в момент времени t , которое выражается в виде дисконтированной суммы немедленных вознаграждений во все будущие моменты времени:

$$R_t = r_t + \gamma \cdot r_{t+1} + \gamma^2 \cdot r_{t+2} + \gamma^3 \cdot r_{t+3} \dots = \sum_{i=0}^{\infty} \gamma^i r_{t+i}. \quad (9.2)$$

Коэффициент дисконтирования $\gamma \in (0, 1)$ позволяет управлять *миопичностью* опережающего просмотра при определении размера вознаграждения. Значения коэффициента γ меньше 1, поскольку будущие вознаграждения имеют меньшую ценность, чем получаемые немедленно. Выбор $\gamma = 0$ приведет к тому, что полное вознаграждение R_t ограничится вознаграждением в текущий момент времени r_t , без каких-либо дополнительных вкладов. Поэтому такой выбор

сделает невозможным обучение долгосрочным ловушкам в игре крестики-нолики. Значения γ , слишком близкие к 1, приведут к нестабильности моделирования в случае очень длинных марковских процессов принятия решений.

Мерой долгосрочного значения, присущего выполнению действия a_t в состоянии s_t , служит Q -функция, или Q -значение, пары “состояние — действие” (s_t, a_t) , обозначаемая как $Q(s_t, a_t)$. Функция $Q(s_t, a_t)$ представляет наилучшее возможное вознаграждение, получаемое в конце игры за выполнение действия a_t в состоянии s_t . Другими словами, $Q(s_t, a_t)$ равна $\max \{E[R_{t+1} | a_t]\}$. Поэтому, если A — набор всех возможных действий, то в качестве действия в момент времени t выбирается действие a_t^* , максимизирующее $Q(s_t, a_t)$. Таким образом, имеем следующее уравнение:

$$a_t^* = \operatorname{argmax}_{a_t \in A} Q(s_t, a_t). \quad (9.3)$$

Предсказанное действие в такой форме — неплохой выбор для следующего хода, хотя нередко оно комбинируется с исследовательской компонентой (например, получаемой с помощью ε -жадного алгоритма) для улучшения долгосрочных тренировочных исходов.

9.4.1. Модели глубокого обучения как аппроксиматоры функций

Чтобы упростить последующее обсуждение, рассмотрим случай платформы Atari [335], когда состояние s_t предоставляется фиксированным окном, включающим несколько последних снимков экрана. Обозначим через \bar{X}_t признаковое представление состояния s_t . Нейронная сеть использует \bar{X}_t в качестве входа, а выходом для каждого возможного допустимого действия из полного набора действий A является $Q(s_t, a)$.

Предположим, нейронная сеть параметризована вектором весов \bar{W} и имеет $|A|$ выходов, которые содержат Q -значения, соответствующие различным действиям из набора A . Иными словами, для каждого действия $a \in A$ нейронная сеть способна вычислить функцию $F(\bar{X}_t, \bar{W}, a)$, выступающую в качестве обучаемой оценки $Q(s_t, a)$:

$$F(\bar{X}_t, \bar{W}, a) = \hat{Q}(s_t, a). \quad (9.4)$$

Обратите внимание на символ циркумфлекса над обозначением Q -функции, указывающий на то, что данное значение предсказывается с использованием обучаемых параметров \bar{W} . Обучение \bar{W} — ключ к применению модели для принятия решений о том, какое именно действие следует использовать в данный момент времени. Рассмотрим в качестве примера видеоигру, в которой возможными ходами являются перемещения вверх, вниз, влево и вправо. В этом случае нейронная сеть будет иметь четыре выхода (рис. 9.3). В частном случае игр для

Atari 2600 вход содержит $m = 4$ пространственные пиксельные карты в градациях серого, представляющие окно последних m ходов [335, 336]. Для преобразования пикселей в Q -значения используется сверточная нейронная сеть, которая называется Q -сетью. Специфику данной архитектуры мы обсудим чуть позже.



Рис. 9.3. Q -сеть для видеоигры на платформе Atari

Алгоритм Q -обучения

Весы \bar{W} нейронной сети нуждаются в обучении посредством тренировки. И здесь мы сталкиваемся с одной интересной проблемой. Мы можем обучить вектор весов, только если имеются наблюдаемые значения Q -функции. Располагая наблюдаемыми значениями Q -функции, мы можем легко задать функцию потерь в виде $Q(s_t, a_t) - \hat{Q}(s_t, a)$, чтобы выполнять обучение после каждого действия. Проблема в том, что Q -функция представляет максимальное дисконтированное (приведенное) вознаграждение по всем будущим комбинациям действий, наблюдать которые в текущий момент времени невозможно.

Существует один интересный трюк, которым можно воспользоваться для задания функции потерь нейронной сети. В соответствии с примечанием 9.4.1, если благодаря использованию частичных знаний о будущих состояниях нам известны улучшенные оценки Q -значений, то в действительности мы не нуждаемся в наблюдаемых значениях для того, чтобы определить функцию потерь. В таком случае мы можем использовать улучшенную оценку для создания суррогатного “наблюдаемого” значения. Это “наблюдаемое” значение определяется уравнением Беллмана [26], которое является математическим выражением принципов динамического программирования и которому удовлетворяет Q -функция, а частичным знанием является вознаграждение для каждого действия, наблюдаемое в текущий момент времени. Согласно уравнению Беллмана мы задаем эталонное значение, выполняя просмотр на один шаг вперед и предсказывая значение в момент времени s_{t+1} :

$$Q(s_t, a_t) = r_t + \gamma \max_a \hat{Q}(s_{t+1}, a). \quad (9.5)$$

Справедливость этого соотношения следует из того факта, что Q -функция предназначена для максимизации дисконтированной будущей выгоды. По сути, мы просматриваем все действия на шаг вперед для того, чтобы создать улучшенную оценку $Q(s_t, a_t)$. В случае эпизодических последовательностей важно установить $\hat{Q}(s_{t+1}, a)$ в 0, если выполнение действия завершает процесс. Это же

соотношение можно также выразить в терминах предсказаний нашей нейронной сети:

$$F(\bar{X}_t, \bar{W}, a_t) = r_t + \gamma \max_a F(\bar{X}_{t+1}, \bar{W}, a). \quad (9.6)$$

Отметим, что, прежде чем мы сможем вычислить “наблюдаемое” значение в момент времени t в правой части приведенного выше уравнения, необходимо дождаться наблюдения состояния \bar{X}_{t+1} и вознаграждения r_t путем выполнения действия a_t . Это обеспечивает естественный способ выражения функции потерь L_t нейронной сети в момент времени t посредством сравнения (суррогатного) наблюдаемого значения с предсказанным значением в момент времени t :

$$L_t = \left\{ \underbrace{\left[r_t + \gamma \max_a F(\bar{X}_{t+1}, \bar{W}, a) \right]}_{\text{Обрабатывать как постоянное эталонное значение}} - F(\bar{X}_t, \bar{W}, a_t) \right\}^2. \quad (9.7)$$

Таким образом, теперь мы можем обновить вектор весов \bar{W} , применяя к этой функции потерь алгоритм обратного распространения ошибки. Очень важно подчеркнуть, что целевые значения, оцениваемые с помощью входов в момент времени $(t + 1)$, обрабатываются алгоритмом обратного распространения как постоянные эталонные значения. Поэтому производная функции потерь будет трактовать эти оценочные значения как константы, даже если они получены от параметризованной нейронной сети со входом \bar{X}_{t+1} . Обработка $F(\bar{X}_{t+1}, \bar{W}, a)$ не как константы приведет к плохим результатам. Это объясняется тем, что мы рассматриваем предсказание в момент времени $(t + 1)$ как улучшенную оценку эталонного значения (на основании принципа бутстреппинга). В результате алгоритм обратного распространения будет выполнять вычисления по следующей формуле:

$$\bar{W} \leftarrow \bar{W} + \alpha \left\{ \underbrace{\left[r_t + \gamma \max_a F(\bar{X}_{t+1}, \bar{W}, a) \right]}_{\substack{\text{Обрабатывать как постоянное} \\ \text{эталонное значение}}} - F(\bar{X}_t, \bar{W}, a_t) \right\} \frac{\partial F(X_t, \bar{W}, a_t)}{\partial \bar{W}}. \quad (9.8)$$

В матричных обозначениях частная производная функции $F(\cdot)$ по вектору \bar{W} является, в сущности, градиентом $\nabla_{\bar{W}} F$. В начале процесса Q-значения, оцениваемые нейронной сетью, являются случайными, поскольку вектор весов \bar{W} иницируется случайными значениями. Однако со временем эта оценка становится более точной, поскольку веса постоянно обновляются для максимизации вознаграждений.

Поэтому в любой заданный момент времени t , в котором наблюдаются действие a_t и вознаграждение r_t , для обновления весов \bar{W} применяется следующий тренировочный процесс.

1. Выполнить проход по сети в прямом направлении со входом \bar{X}_{t+1} для вычисления $\hat{Q}_{t+1} = \max_a F(\bar{X}_{t+1}, \bar{W}, a)$. Если процесс прекращается после выполнения действия a_t , то это значение равно нулю. *Специальная обработка завершающего состояния очень важна.* В соответствии с уравнением Беллмана Q-значение в предыдущий момент времени должно быть равным $r_t + \gamma \hat{Q}_{t+1}$ для наблюдаемого действия a_t в момент времени t . Поэтому вместо использования наблюдаемых значений мы создаем *суррогат* целевого значения в момент времени t и считаем, что этот суррогат и есть то наблюдаемое значение, которое нам предоставляется.
2. Выполнить проход по сети в прямом направлении со входом \bar{X}_t для $F(\bar{X}_t, \bar{W}, a_t)$.
3. Задать функцию потерь $L_t = (r_t + \gamma Q_{t+1} - F(\bar{X}_t, \bar{W}, a_t))^2$ и выполнить обратное распространение ошибки в сети со входом \bar{X}_t . Обратите внимание на то, что эти потери связаны с выходным узлом сети, соответствующим действию a_t , а потери для всех остальных действий равны нулю.
4. Теперь к этой функции потерь можно применить обратное распространение ошибки для обновления вектора весов \bar{W} . Даже если член $r_t + \gamma Q_{t+1}$ в функции потерь также получается в виде предсказания для входа нейронной сети, равного \bar{X}_{t+1} , он обрабатывается как (постоянное) наблюдаемое значение в процессе вычисления градиента алгоритмом обратного распространения.

Тренировка и предсказание осуществляются одновременно, поскольку значения действий используются для обновления весов и выбора следующего действия. Здесь возникает соблазн выбрать в качестве соответствующего предсказания действие с наибольшим Q-значением. Однако такой подход может привести к неадекватному исследованию пространства поиска. Поэтому при выборе следующего действия мы сочетаем оптимальность предсказания с какой-либо стратегией, например ϵ -жадной. Действие с наибольшим предсказанным вознаграждением выбирается с вероятностью $1 - \epsilon$, иначе выбирается случайное действие. Значение ϵ сначала может устанавливаться большим, а затем уменьшаться с течением времени. Поэтому *целевое прогнозное значение* для нейронной сети вычисляется с использованием в уравнении Беллмана наилучшего из возможных действий (которое в конечном счете может отличаться от наблюдаемого действия a_{t+1} , определенного на основе ϵ -жадной стратегии). По этой причине Q-обучение называют *алгоритмом, не зависящим от стратегии* (off-policy algorithm), в котором целевые предсказанные значения для обновления нейронной сети вычисляются с использованием действий, которые могут отличаться от фактически наблюдаемых действий в будущем.

Существует ряд модификаций этого базового подхода, цель которых — повысить стабильность обучения. Многие из них представлены в контексте видеоигр на платформе Atari [335]. Прежде всего, предоставление тренировочных примеров *точно в той же* последовательности, в какой они встречаются в игре, может приводить к локальным минимумам в силу большого сходства между тренировочными примерами. Как следствие, используется пул в виде истории действий/вознаграждений фиксированной длины, который можно рассматривать как предыдущие истории опыта. Из этого пула семплируются образцы предыдущего опыта для выполнения мини-пакетного градиентного спуска. Вообще говоря, одно и то же действие может семплироваться не один раз, что повышает эффективность использования обучающих данных. Пул обновляется с течением времени по мере того, как старые действия исключаются из него, а новые — добавляются. Поэтому тренировка осуществляется с учетом временной предыстории, но в приближенном, а не строгом смысле. Такой подход называют *воспроизведением опыта* (experience replay), поскольку образцы предыдущего опыта многократно воспроизводятся в несколько иной последовательности, чем оригинальные действия.

Другое видоизменение заключается в том, что сеть, используемая для оценки целевых Q-значений с помощью уравнений Беллмана (описанный выше шаг 1), отделена от сети, используемой для прогнозирования Q-значений (шаг 2). Первая из них обновляется медленнее для повышения стабильности. Наконец, одной из проблем этих систем является разреженность вознаграждений, особенно на начальной стадии обучения, когда ходы выбираются случайным образом. В подобных случаях можно использовать ряд приемов, таких как *приоритетное воспроизведение опыта* (prioritized experience replay) [428]. Базовая идея заключается в повышении эффективности использования тренировочных данных, собранных в процессе обучения с подкреплением, путем присваивания более высокого приоритета действиям, которые могут внести больший вклад в обучение.

9.4.2. Пример: нейронная сеть для игр на платформе Atari

Размеры экрана для сверточной нейронной сети [335, 336] были установлены равными 84×84 пикселя, что также определяет пространственные размеры первого слоя сверточной сети. Использовался вход в градациях серого, поэтому для каждого экрана требовалась лишь одна карта пространственных признаков, хотя во входном слое требовалась глубина 4 для представления четырех предыдущих пиксельных окон. Применялись три сверточных слоя с фильтрами 8×8 , 4×4 и 3×3 соответственно. В первом сверточном слое использовались 32 фильтра, в двух других — по 64 фильтра, причем свертка осуществлялась с

использованием шагов 4, 2 и 1 соответственно. За сверточными слоями следовали два полносвязных слоя. Количество нейронов в предпоследнем слое составляло 512, а в последнем было равно количеству выходов (возможных действий). Количество выходных слоев менялось от 4 до 18 в зависимости от игры. Общая архитектура сверточной сети представлена на рис. 9.4.

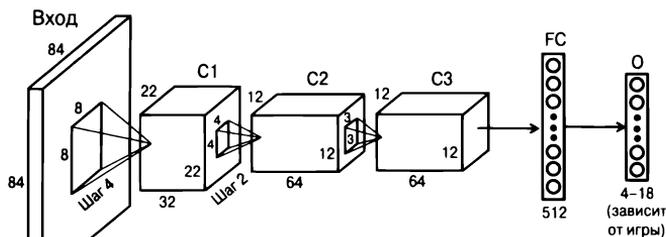


Рис. 9.4. Сверточная нейронная сеть для игр на платформе Atari

Во всех скрытых слоях применялась ReLU-активация, а в выходном — линейная активация для предсказания вещественных Q-значений. Пулинг не был задействован, а использование шаговой свертки обеспечило пространственное сжатие. Платформа Atari поддерживает множество игр, и для всех применялась одна и та же общая архитектура, что позволило продемонстрировать ее общность. Производительность в различных играх варьировалась в определенных пределах, но во многих случаях сеть работала на уровне человеческих возможностей. С наибольшими трудностями алгоритм столкнулся в тех играх, которые требовали использования долгосрочных стратегий. Тем не менее надежное функционирование сравнительно однородной сети во многих играх послужило вдохновляющим примером.

9.4.3. Методы, привязанные и не привязанные к стратегии: SARSA

Q-обучение относится к классу методов, получившему название *обучение на основе временных разностей* (temporal difference learning), или *TD-обучение*. В Q-обучении действия выбираются в соответствии с ϵ -жадной стратегией. Однако параметры нейронной сети обновляются на основании наилучшего из возможных действий на каждом шаге с помощью уравнения Беллмана. Наилучшее возможное действие на каждом шаге — это не совсем то же самое, что действие, определяемое ϵ -жадной политикой и используемое для имитации. Поэтому Q-обучение — это *метод обучения с подкреплением, не привязанный к стратегии*. Выбор другой политики для выполнения действий из числа тех, которые применяются для выполнения обновлений, не ухудшит способности сети находить оптимальные решения, что и является целью обновлений.

В действительности использование для исследовательских действий преимущественно рандомизированной политики позволяет избегать локальных оптимумов.

В методах, привязанных к стратегии, действия согласуются с обновлениями, поэтому обновления могут рассматриваться как политика *оценки*, а не как *оптимизация*. Чтобы понять суть, мы опишем обновления для алгоритма SARSA (State-Action-Reward-State-Action — состояние–действие–вознаграждение–состояние–действие), в котором оптимальное вознаграждение на следующем шаге не используется для вычисления обновлений. Вместо этого следующий шаг обновляется с помощью той же ε -жадной стратегии, с помощью которой получают действия a_{t+1} для вычисления целевых значений. Функция потерь для следующего шага определяется следующим образом:

$$L_t = \{r_t + \gamma F(\bar{X}_{t+1}, \bar{W}, a_{t+1}) - F(\bar{X}_t, \bar{W}, a_t)\}^2. \quad (9.9)$$

Функция $F(\cdot, \cdot, \cdot)$ определяется точно так же, как и на предыдущем разделе. Вектор весов обновляется на основании этой функции потерь, после чего выполняется действие a_{t+1} :

$$\bar{W} \leftarrow \bar{W} + \alpha \left\{ \underbrace{\left[r_t + \gamma F(\bar{X}_{t+1}, \bar{W}, a_{t+1}) \right]}_{\substack{\text{Обрабатывать как постоянное} \\ \text{эталонное значение}}} - F(\bar{X}_t, \bar{W}, a_t) \right\} \frac{\partial F(\bar{X}_t, \bar{W}, a_t)}{\partial \bar{W}}. \quad (9.10)$$

Будет поучительно сравнить это обновление с теми, которые применялись в Q-обучении в соответствии с уравнением 9.8. В Q-обучении для обновления параметров используют наилучшее возможное действие в каждом состоянии, даже если фактически применяется ε -жадная стратегия (поощряющая исследовательские действия). В SARSA мы используем действие, которое было фактически выбрано ε -жадным методом для выполнения обновления. Именно поэтому такой подход и называется *методом, привязанным к стратегии*. Методы, не привязанные к стратегии, такие как Q-обучение, способны отделять исследования от эксплуатации накопленного опыта, тогда как в методах, не привязанных к стратегии, это не делается. Заметьте, что если мы установим для ε в ε -жадной стратегии нулевое значение (т.е. применим чисто жадную стратегию), то Q-обучение и SARSA будет соответствовать одному и тому же алгоритму. Однако такой подход будут не очень хорошо работать, поскольку он исключает исследовательские действия. Подход SARSA полезен в тех случаях, когда обучение не может быть выполнено отдельно от прогнозирования. С другой стороны, Q-обучение полезно в ситуациях, когда можно независимо обучить модель, а затем использовать обученную политику в сочетании с чисто жадным методом с $\varepsilon = 0$ (и без необходимости дальнейшего обновления модели). Использование

ϵ -жадного метода на этапе вывода в Q-обучении было бы опасно, поскольку эта политика никогда не вознаграждает исследовательскую компоненту, а значит, не учится тому, как поддерживать безопасность исследовательских действий. Например, робот, обученный по методике Q-обучения, будет выбирать кратчайший путь из точки А в точку В, даже если этот путь проходит по краю ущелья, тогда как робот, тренированный по SARSA, не допустит такой ошибки.

Обучение без аппроксиматоров функций

В тех случаях, когда пространство состояний очень мало, обучение Q-значений может быть выполнено без аппроксиматоров функций. Например, в такой небольшой игре, как крестики-нолики, можно выполнить явное обучение $Q(s_t, a_t)$, участь на игре против сильного противника методом проб и ошибок. В этом случае уравнения Беллмана (см. уравнение 9.5) используются на каждом ходе для обновления массива, содержащего явные значения $Q(s_t, a_t)$. Непосредственное использование уравнения 9.5 — чересчур агрессивная стратегия. Обычно применяют более мягкие обновления со скоростью обучения $\alpha < 1$:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t)(1 - \alpha) + \alpha(r_t + \gamma \max_a Q(s_{t+1}, a)). \quad (9.11)$$

Использование значения $\alpha = 1$ приведет к уравнению 9.5. Непрерывное обновление массива приведет к таблице, содержащей корректное стратегическое значение каждого хода. Чтобы понять, какой смысл вкладывается в термин *стратегическое значение*, обратитесь, например, к рис. 9.2, а. Примеры четырех записей из такой таблицы приведены на рис. 9.2, б.

Алгоритм SARSA также можно применять без аппроксиматоров функций, используя действие a_{t+1} на основе ϵ -жадной стратегии. Мы используем верхний индекс p в $Q^p(\cdot, \cdot)$ для указания того, что это оператор оценки стратегии p (которая в данном случае является ϵ -жадной):

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t)(1 - \alpha) + \alpha(r_t + \gamma \max_a Q(s_{t+1}, a_t)). \quad (9.12)$$

Этот подход представляет собой более сложную альтернативу ϵ -жадному методу, который обсуждался в разделе 9.3.2. Заметим, что если действие a_t в состоянии s_t приводит к завершению (эпизодического) процесса, то $Q^p(s_t, a_t)$ устанавливается в r_t .

9.4.4. Моделирование состояний, а не пар “состояние — действие”

Незначительной вариацией темы предыдущих разделов является обучение значения отдельного состояния (а не пары “состояние — действие”). Все методы, которые мы обсуждали ранее, можно реализовать, поддерживая значения состояний, а не пары “состояние — действие”. Например, SARSA можно реа-

лизовать, оценивая значения всех состояний, являющихся результатом каждого возможного действия, и выбирая подходящее состояние на основании заранее установленной стратегии, например ϵ -жадной. В действительности самые ранние методы обучения на основе временных разностей (*TD-обучение*) поддерживали значения состояний, а не пары “состояние — действие”. С точки зрения эффективности для принятия решений на основе значений удобнее выводить значения всех действий за один раз (а не вычислять повторно каждое будущее состояние). Работать со значениями состояний, а не с парами “состояние — действие”, целесообразно только в тех случаях, когда не удастся выразить удобным способом стратегию в терминах пар “состояние — действие”. Например, мы можем просмотреть наперед многообещающие ходы в шахматах и получить некоторое усредненное значение для бутстрэппинга. В подобных случаях желательно вычислять состояния, а не пары “состояние — действие”. Поэтому в данном разделе мы обсудим вариацию метода временных разностей, в котором вычисляются непосредственно состояния.

Обозначим через $V(s_t)$ значение состояния s_t . Предположим, что у нас имеется параметризованная нейронная сеть, которая использует наблюдаемые атрибуты \bar{X}_t (например, пиксели последних четырех экранов игры на платформе Atari) состояния s_t для оценки $V(s_t)$. Пример такой нейронной сети приведен на рис. 9.5. Тогда, если функцией, вычисляемой нейронной сетью, является $G(\bar{X}_t, \bar{W})$ с вектором параметров \bar{W} , выполняется следующее соотношение:

$$G(\bar{X}_t, \bar{W}) = \hat{V}(s_t). \quad (9.13)$$

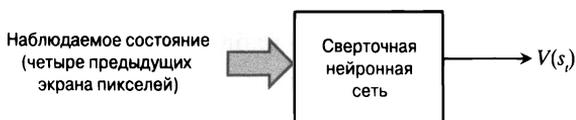


Рис. 9.5. Оценка значения состояния при обучении методом временных разностей

Стратегия, которой необходимо следовать, принимая решения относительно действий, может использовать некоторую произвольную оценку просмотренных наперед состояний для принятия таких решений. Пока что предположим, что мы располагаем некоей разумной эвристической стратегией выбора действий, которая каким-то образом использует значения просмотренных наперед состояний. Например, если мы вычислим каждое будущее состояние, являющееся результатом выполнения определенного действия, и выберем одно из них на основании заранее принятой стратегии (например, ϵ -жадной), то обсуждаемый ниже подход будет совпадать с методом SARSA.

Пусть действие a_t выполняется с вознаграждением r_t , а значением результирующего состояния s_{t+1} является $V(s_{t+1})$. Поэтому компенсационную эталонную

оценку для $V(s_t)$ можно получить с помощью следующего опережающего просмотра:

$$V(s_t) = r_t + \gamma V(s_{t+1}). \quad (9.14)$$

Эта оценка также может быть выражена через параметры нейронной сети:

$$G(\bar{X}_t, \bar{W}) = r_t + \gamma G(\bar{X}_{t+1}, \bar{W}). \quad (9.15)$$

Во время фазы тренировки необходимо сместить веса таким образом, чтобы вытолкнуть $G(\bar{X}_t, \bar{W})$ в направлении улучшенного эталонного значения $r_t + \gamma G(\bar{X}_{t+1}, \bar{W})$. Как и в случае Q-обучения, мы работаем в предположении бутстрэппинга так, словно значение $r_t + \gamma G(\bar{X}_{t+1}, \bar{W})$ является предоставленным нам наблюдаемым значением. Следовательно, мы хотим минимизировать *TD-ошибку*, определяемую таким соотношением:

$$\delta_t = \underbrace{r_t + \gamma G(\bar{X}_{t+1}, \bar{W})}_{\text{“Наблюдаемое” значение}} - G(\bar{X}_t, \bar{W}). \quad (9.16)$$

Поэтому функция потерь определяется следующим образом:

$$L_t = \delta_t^2 = \left\{ \underbrace{r_t + \gamma G(\bar{X}_{t+1}, \bar{W})}_{\text{“Наблюдаемое” значение}} - G(\bar{X}_t, \bar{W}) \right\}^2. \quad (9.17)$$

Как и в случае Q-обучения, чтобы вычислить $r_t + \gamma G(\bar{X}_{t+1}, \bar{W})$, необходимо сначала вычислить наблюдаемое значение состояния в момент времени t , используя вход \bar{X}_{t+1} нейронной сети. Поэтому необходимо дождаться наблюдения действия a_t , когда станут доступными наблюдаемые признаки \bar{X}_{t+1} состояния s_{t+1} . Затем, когда вход \bar{X}_t будет использован для предсказания значения состояния s_t , это наблюдаемое значение состояния s_t , определяемое величиной $r_t + \gamma G(\bar{X}_{t+1}, \bar{W})$, станет (постоянным) целевым значением для обновления весов нейронной сети. Таким образом, веса нейронной сети необходимо сместить на основании градиента следующей функции потерь:

$$\begin{aligned} \bar{W} &\leftarrow \bar{W} - \alpha \frac{\partial L_t}{\partial \bar{W}} = \\ &= \bar{W} + \alpha \left\{ \underbrace{[r_t + \gamma G(\bar{X}_{t+1}, \bar{W})]}_{\text{“Наблюдаемое” значение}} - G(\bar{X}_t, \bar{W}) \right\} \frac{\partial G(\bar{X}_t, \bar{W})}{\partial \bar{W}} = \\ &= \bar{W} + \alpha \delta_t (\nabla G(\bar{X}_t, \bar{W})). \end{aligned}$$

Этот алгоритм является частным случаем алгоритма $TD(\lambda)$ с λ , равным нулю. В данном частном случае нейронная сеть обновляется только за счет создания

компенсационного эталонного значения для текущего момента времени на основании вычислений для следующего момента времени. Определение эталонных значений таким способом является *миотическим приближением* (myopic approximation). Например, в случае шахмат система обучения с подкреплением могла непреднамеренно допустить ошибку много ходов назад, вследствие чего на каком-то этапе результаты компенсационных предсказаний могут неожиданно продемонстрировать высокий уровень ошибок, чего прежде не замечалось. Ошибки в компенсационных предсказаниях указывают на то, что нами получена новая информация относительно всех прошлых состояний \bar{X}_k , которую мы можем использовать для изменения предсказания. Одной из возможностей является выполнение бутстрэппинга путем просмотра на много шагов вперед (см. упражнение 7). Другим возможным решением является использование $TD(\lambda)$ -обучения, которое исследует континуум между эталонным значением, полученным методом Монте-Карло, и одношаговой аппроксимацией с плавным затуханием. Величина поправок к старым предсказаниям постепенно уменьшается со скоростью $\lambda < 1$. Можно показать [482], что в этом случае формула для обновления приобретает следующий вид:

$$\bar{W} \leftarrow \bar{W} + \alpha \delta_t \underbrace{\sum_{k=0}^t (\lambda \gamma)^{t-k} (\nabla G(\bar{X}_k, \bar{W}))}_{\text{После предсказания } \bar{X}_k}. \quad (9.18)$$

При $\lambda = 1$ этот подход эквивалентен методу, в котором для вычисления эталонных значений применяется метод Монте-Карло (т.е. развертывание эпизодического процесса до самого конца) [482]. Это объясняется тем, что в данном случае мы всегда используем новую информацию об ошибках для того, чтобы полностью скорректировать наши прошлые ошибки без дисконтирования при $\lambda = 1$, тем самым создавая несмещенную оценку. Обратите внимание на то, что λ применяется только для дисконтирования шагов, тогда как γ используется также при вычислении TD-ошибки δ_t в соответствии с уравнением 9.16. Параметр λ *специфичен в отношении алгоритма*, в то время как параметр γ *специфичен в отношении окружения*. Использование $\lambda = 1$ или семплирования методом Монте-Карло приводит к уменьшению смещения и увеличению дисперсии. В качестве примера рассмотрим игру в шахматы, в которой агенты Алиса и Боб совершают каждый по три ошибки в ходе одной игры, но в конечном счете выигрывает Алиса. Одиночное развертывание этой игры методом Монте-Карло не сможет различить влияние каждой конкретной ошибки и будет назначать дисконтированное вознаграждение за финальный исход игры каждой позиции на доске. С другой стороны, n -шаговый метод временных разностей (т.е. n -кратная оценка позиций на доске) позволяет фиксировать ошибку для каждой позиции на доске, в которой агент допустил ошибку и это было обнаружено в

результате опережающего n -шагового просмотра. Метод Монте-Карло сможет различать различные типы ошибок только при наличии достаточного объема данных (т.е. большего количества игр). В то же время выбор очень малых значений λ приведет к возникновению трудностей (к большому смещению) в начале обучения, поскольку ошибки с долгосрочными последствиями не будут обнаружены. Подобные проблемы с началом обучения хорошо документированы [22, 496].

Обучение методом временных разностей использовалось в знаменитой программе Сэмюэля для игры в шашки [421] и побудило Тезауро к разработке программы TD-Gammon для игры в нарды [492]. Значения состояний оценивались с помощью нейронной сети, а ее параметры обновлялись бутстрэппингом по методу временных разностей по последовательным ходам. Окончательный вывод выполнялся посредством минимаксного вычисления улучшенной функции оценки на мелкой глубине порядка 2 или 3. Программе TD-Gammon удалось победить нескольких опытных игроков. Она также продемонстрировала ряд необычных игровых стратегий, которые в конечном счете были переняты игроками высшего уровня.

9.5. Градиентный спуск по стратегиям

Методы, основанные на оценке значений, такие как Q-обучение, пытаются предсказать значение действия с помощью нейронной сети в сочетании с типовой стратегией (например, ϵ -жадной). С другой стороны, методы, основанные на *градиентном спуске по стратегиям* (policy gradient), оценивают *вероятность* каждого действия на каждом шаге для максимизации общего вознаграждения. Поэтому вместо использования оценки значений в качестве промежуточного шага для выбора действий параметризуется сама стратегия.

Нейронную сеть, применяемую для оценки стратегий, называют *сетью стратегий* (policy network). В этой сети входом является текущее состояние системы, а выходом — набор вероятностей, связанных с различными действиями в видеоигре (например, перемещение вверх, вниз, вправо или влево). Как и в случае Q-сети, входом может быть наблюдаемое представление состояния агента. Так, в видеоигре на платформе Atari наблюдаемым состоянием могут быть последние четыре пиксельных экрана. Пример сети стратегий, соответствующий условиям видеоигр Atari, приведен на рис. 9.6. Поучительно сравнить эту сеть стратегий с Q-сетью, представленной на рис. 9.3. При заданном выходе вероятностей для различных действий мы бросаем “неправильную” кость, грани которой связаны с этими вероятностями, и выбираем одно из действий. Поэтому для каждого действия a , наблюдаемого представления состояния \bar{X}_t и текущего параметра \bar{W} нейронная сеть способна вычислить функцию $P(\bar{X}_t, \bar{W}, a)$, пред-

ставляющую вероятность того, что должно быть выполнено действие a . Сэмплируется одно из действий, и наблюдается вознаграждение за это действие. Если данная политика неудачна, то действие с большей вероятностью будет ошибочным, и, соответственно, будет низким и вознаграждение. На основании размера вознаграждения, полученного за выполнение действия, обновляется вектор весов \bar{W} для следующей итерации. Обновление вектора весов базируется на понятии градиента стратегии по вектору весов \bar{W} . Одной из трудностей оценки градиента стратегии является то, что вознаграждение за действие часто не наблюдается сразу же, но тесно интегрируется в последовательность будущих вознаграждений. Нередко приходится прибегать к *развертыванию стратегии методом Монте-Карло* (Monte Carlo policy roll-out), когда нейронная сеть используется для того, чтобы следовать определенной стратегии для оценки дисконтированных вознаграждений в пределах дальних горизонтов.

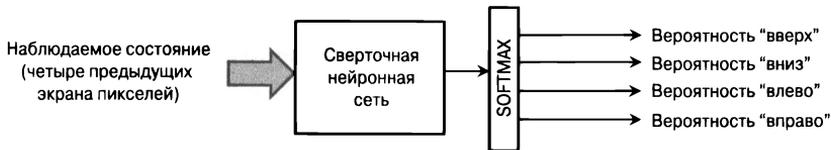


Рис. 9.6. Сеть стратегий, соответствующая условиям видеоигр на платформе Atari. Поучительно сравнить эту конфигурацию с Q-сетью, представленной на рис. 9.3

Мы хотим обновить вектор весов нейронной сети вдоль направления роста вознаграждения. Как и в случае Q-обучения, ожидаемые дисконтированные вознаграждения для заданного горизонта H вычисляются следующим образом:

$$J = E[r_0 + \gamma \cdot r_1 + \gamma^2 \cdot r_2 + \dots + \gamma^H \cdot r_H] = \sum_{i=0}^H E[\gamma^i r_i]. \quad (9.19)$$

Поэтому задача заключается в обновлении вектора весов по следующей формуле:

$$\bar{W} \leftarrow \bar{W} - \alpha \nabla J. \quad (9.20)$$

Основной проблемой при оценке градиента ∇J является то, что нейронная сеть выводит лишь вероятности. Наблюдаемые вознаграждения являются лишь выборками из этих выводов, полученными методом Монте-Карло, тогда как мы хотим вычислить градиенты ожидаемых вознаграждений (см. уравнение 9.19). К числу распространенных градиентных методов моделирования стратегий относятся *метод конечных разностей* (finite difference method), *метод относительного правдоподобия* (likelihood ratio method) и *естественный градиентный спуск по стратегиям* (natural policy gradients). Ниже мы обсудим лишь первые два из этих методов.

9.5.1. Метод конечных разностей

Метод конечных разностей (finite differences method) обходит проблему стохастичности с помощью эмпирических имитаций, представляющих оценки градиента. Для оценки градиентов вознаграждения методы конечных разностей применяют возмущения весов. Суть идеи заключается в использовании различных возмущений весов нейронной сети и исследовании ожидаемого изменения вознаграждения DJ . Учтите, что оценка изменения размера вознаграждения потребует от вас применения стратегии возмущений к горизонту ходов H . Такую последовательность ходов H называют *разверткой* (roll-out). Например, для оценки изменения величины вознаграждения в случае игры Atari мы должны воспроизвести игру вдоль траектории ходов H для каждого из s различных наборов возмущенных весов. В отсутствие достаточно сильного противника для игры тренировку можно проводить, играя против версии противника, базирующейся на параметрах, которым модель обучилась несколько итераций назад.

В общем случае значение H может быть достаточно большим для того, чтобы мы могли достигнуть конца игры, и поэтому используемым счетом будет окончательный счет игры. В такой игре, как го, счет становится известным только в конце игры и равен $+1$ в случае победы и -1 в случае проигрыша. В подобных случаях важно выбирать горизонт H достаточно большим, чтобы игру можно было довести до конца. В результате мы получим s различных векторов изменения весов $\Delta\bar{W}_1 \dots \Delta\bar{W}_s$ вместе с соответствующими изменениями $\Delta J_1 \dots \Delta J_s$ в общем вознаграждении. Каждая из этих пар в грубом приближении удовлетворяет следующему соотношению:

$$(\Delta\bar{W}_r)\Delta J^T \approx \Delta J_r, \quad \forall r \in \{1 \dots s\}. \quad (9.21)$$

Мы можем создать s -мерный вектор-столбец $\bar{y} = [\Delta J_1 \dots \Delta J_s]^T$, представляющий изменения целевой функции, и матрицу D размером $N \times s$, расположив строки $\Delta\bar{W}_r$ одна поверх другой, где N — количество параметров нейронной сети. В итоге мы приходим к следующему соотношению:

$$D[\Delta J]^T \approx \bar{y}. \quad (9.22)$$

Тогда градиент стратегии можно получить, выполнив линейную регрессию изменения целевой функции в зависимости от изменения векторов весов. Используя формулу линейной регрессии (см. раздел 2.2.2.2), получаем

$$\Delta J^T = (D^T D)^{-1} D^T \bar{y}. \quad (9.23)$$

Этот градиент применяется для обновлений в уравнении 9.20. Чтобы оценить градиенты, потребуется выполнить стратегию для последовательности H ходов по каждому из s примеров. Иногда этот процесс может быть очень медленным.

9.5.2. Методы относительного правдоподобия

Методы относительного правдоподобия (likelihood-ratio) были предложены Уильямсом [533] в контексте алгоритма REINFORCE. Предположим, что мы следуем стратегии с вектором вероятностей \bar{p} и хотим максимизировать функцию $E[Q^p(s, a)]$, представляющую долгосрочное ожидаемое значение для состояния s и каждого из семплированных действий a . Рассмотрим случай, когда вероятность действия a равна $p(a)$ (выход нейронной сети). Мы хотим найти градиент $E[Q^p(s, a)]$ по вектору весов \bar{W} нейронной сети для стохастического градиентного подъема. Способ нахождения градиента ожидаемого значения для семплированных событий далеко не очевиден. Однако трюк с логарифмом вероятности позволяет получить выражение для градиента в аддитивной форме по парам “состояние — действие”:

$$\Delta E[Q^p(s, a)] = E[Q^p(s, a) \nabla \log(p(a))]. \quad (9.24)$$

Докажем справедливость этого результата в терминах частной производной по единственному весу w нейронной сети в предположении, что a — дискретная переменная:

$$\begin{aligned} \frac{\partial E[Q^p(s, a)]}{\partial w} &= \frac{\partial \left[\sum_a Q^p(s, a) p(a) \right]}{\partial w} = \sum_a Q^p(s, a) \frac{\partial p(a)}{\partial w} = \sum_a Q^p(s, a) \left[\frac{1}{p(a)} \frac{\partial p(a)}{\partial w} \right] p(a) = \\ &= \sum_a Q^p(s, a) \left[\frac{\partial \log(p(a))}{\partial w} \right] p(a) = E \left[Q^p(s, a) \frac{\partial \log(p(a))}{\partial w} \right]. \end{aligned}$$

Можно показать, что вышеприведенный результат остается справедливым и в случае непрерывной переменной a (см. упражнение 1). Непрерывные переменные часто встречаются в робототехнике (например, расстояние, на которое должна переместиться рука).

Этот прием можно легко использовать для оценки параметров нейронной сети. С каждым действием, семплируемым в процессе имитации, ассоциировано долгосрочное вознаграждение $Q^p(s, a)$, которое получается имитацией методом Монте-Карло. На основании приведенного выше соотношения градиент ожидаемой выгоды получается умножением градиента логарифмической вероятности $\log(p(a))$ этого действия (вычисляемой с помощью нейронной сети, приведенной на рис. 9.6, посредством обратного распространения ошибки) на долгосрочное вознаграждение $Q^p(s, a)$ (получаемое имитацией методом Монте-Карло).

Рассмотрим шахматную игру, заканчивающуюся выигрышем/проигрышем/ничьей. В этом случае долгосрочное вознаграждение каждого хода может принимать одно из значений $\{\gamma^{r-1}, 0, -\gamma^{r-1}\}$, где γ — коэффициент дисконтиро-

вания (фактор затухания), а r — количество ходов, оставшихся до завершения игры. Значение вознаграждения зависит от конечного исхода игры и количества оставшихся ходов (в силу дисконтирования вознаграждения). Пусть игра содержит самое большое H ходов. Благодаря использованию множества разверток мы получим большое количество тренировочных примеров для различных входных состояний и соответствующих выходов нейронной сети. Например, если мы запустим имитацию для 100 разверток, то получим самое большое $100 \times H$ различных примеров. Каждый из них будет иметь долгосрочное вознаграждение, определяемое значением из набора $\{+\gamma^{r-1}, 0, -\gamma^{r-1}\}$. Для каждого из этих примеров вознаграждение играет роль веса при обновлении логарифмической вероятности семплированного действия в процессе градиентного подъема:

$$\bar{W} \leftarrow \bar{W} + Q^P(s, a) \nabla \log(p(a)), \quad (9.25)$$

где $p(a)$ — вероятность семплированного действия на выходе нейронной сети. Градиенты вычисляются с использованием обратного распространения ошибки, и эти обновления аналогичны тем, которые определяются уравнением 9.20. Этот процесс семплирования и обновления выполняется до наступления сходимости.

Градиент логарифмической вероятности эталонного класса часто применяется с целью обновления Softmax-классификаторов с кросс-энтропийной функцией потерь для увеличения вероятности корректного класса (что похоже на обновление, используемое нами в данном случае). Различие заключается в том, что мы взвешиваем обновление с помощью Q -значений, поскольку заинтересованы в более интенсивном выталкивании параметров в направлении действий, приносящих как можно более высокое вознаграждение. Также можно использовать мини-пакетный градиентный подъем по действиям в семплированных развертках. В качестве меры, помогающей избежать локальных минимумов, появление которых обусловлено корреляцией между находящимися в тесной взаимосвязи последовательными примерами, можно использовать случайное семплирование из различных разверток.

Уменьшение дисперсии с помощью базовых линий. Несмотря на то что в качестве величины, подлежащей оптимизации, мы применяли долгосрочное вознаграждение $Q^P(s, a)$, более распространен подход, в котором вознаграждение отсчитывается от уровня *базовой линии*, что дает свои преимущества (например, облегчает дифференцированную оценку вкладов действий в ожидаемое значение). В идеальном случае базовая линия должна быть специфичной по отношению к состояниям, но она также может быть константой. В оригинальной работе по REINFORCE использовалась постоянная базовая линия (которая в типичных случаях представляет некую меру долгосрочного вознаграж-

дения, усредненного по всем состояниям). Даже такая простая мера позволяет ускорить процесс обучения, поскольку это снижает вероятность действий, приводящих к результатам ниже среднего, и повышает вероятность более результативных действий (а не просто увеличивает вероятность и тех и других в разной степени). Выбор постоянной базовой линии не влияет на смещение процедуры, но уменьшает дисперсию. Вариантом базовой линии, специфической к состояниям, является значение $V^p(s)$ состояния s непосредственно перед семплированием действия a . Такой выбор приводит к преимуществу ($Q^p(s, a) - V^p(s)$), идентичному ошибке метода временных разностей. Интуиция подсказывает, что такой выбор имеет смысл, поскольку ошибка метода временных разностей содержит дополнительную информацию о дифференцированном вознаграждении помимо той, которая нам известна до выполнения этого действия. Обсуждение базовых линий содержится в [374, 433].

Рассмотрим пример агента в игре Atari, в которой в процессе развертки семплируется ход ВВЕРХ, и выходная вероятность хода ВВЕРХ составила 0,2. Предположим, что уровень (постоянной) базовой линии принят равным 0,17, а долгосрочное вознаграждение данного действия равно +1, поскольку игра завершается победой (следовательно, вознаграждение не дисконтируется). Поэтому каждое действие в данной развертке оценивается значением 0,83 (после вычитания уровня базовой линии). Тогда выигрыш, связанный со всеми остальными действиями (выходными узлами нейронной сети), отличными от хода ВВЕРХ, в данный момент времени будет равен нулю, а выигрыш, связанный с выходным узлом, соответствующим ходу ВВЕРХ, составит $0,83 \cdot \log(0,2)$. Эту величину можно использовать в процессе обратного распространения ошибки для обновления параметров нейронной сети.

Корректировке с использованием базовой линии, специфической в отношении состояний, нетрудно дать интуитивное объяснение. Рассмотрим пример игры в шахматы между агентами Алисой и Бобом. Если использовать в качестве базовой линии нулевой уровень, то за каждый ход будет начисляться лишь вознаграждение, соответствующее окончательному результату, и различие между удачными и неудачными ходами не будет очевидным. Иначе говоря, для дифференциации позиций нам придется имитировать намного больше игр. С другой стороны, если использовать в качестве базовой линии значение состояния (до выполнения действия), то о выгодности данного действия можно будет судить по (более точной) ошибке метода временных разностей. В этом случае ходы с большим специфическим по отношению к состоянию влиянием будут считаться более выгодными (в контексте одной игры). В результате для обучения потребуется меньше времени.

9.5.3. Сочетание обучения с учителем с градиентными методами моделирования стратегий

Обучение с учителем целесообразно применять для инициализации сети стратегий перед тем, как приступить к обучению с подкреплением. Если, скажем, речь идет о шахматах, то можно предварительно подготовить примеры заведомо сильных (экспертных) ходов, сделанных профессионалами. В этом случае мы просто выполняем градиентный подъем, используя ту же сеть стратегии, если не считать того, что для вычисления градиента в соответствии с уравнением 9.24 каждому экспертному ходу начисляется фиксированное значение 1. Эта задача становится идентичной задаче Softmax-классификации, в которой целью сети стратегии является предсказание хода, совпадающего с экспертным. Тренировочные данные можно разнообразить примерами неудачных ходов с некоторым отрицательным вознаграждением, получаемым из компьютерных оценок. Такой подход следует рассматривать как обучение с учителем, а не как обучение с подкреплением, поскольку мы всего лишь задействуем готовые данные, а не генерируем/имитируем данные, применяемые для обучения (как в обучении с подкреплением). Эту общую идею можно распространить на любую задачу обучения с подкреплением, в которой доступны готовые примеры действий и связанных с ними вознаграждений. В подобных случаях, учитывая трудности получения высококачественных данных на ранних этапах процесса обучения, стало чрезвычайно популярным применение обучения с учителем для инициализации параметров. Вопросам чередования обучения с учителем и обучения с подкреплением с целью повышения качества данных посвящено большое количество работ [286].

9.5.4. Методы “актор — критик”

В методах, которые мы до сих пор обсуждали, доминировали либо *критики*, либо *акторы*, причем это происходило следующим образом.

1. Q-обучение и TD(λ)-методы работают, используя понятие *функции значения*, или *функции оценки* (value function), которая подлежит оптимизации. Функция значения — это критик, и стратегия (например, ϵ -жадная) актора формируется под его непосредственным влиянием. Поэтому актор подчинен критику, и подобные методы рассматриваются как *методы критика*.
2. В методах градиентного спуска по стратегиям функция значения вообще не используется, и они обучаются непосредственно на вероятностях действий стратегии. Соответствующие значения часто оцениваются путем семплирования методом Монте-Карло. Поэтому подобные методы рассматриваются как *методы актора*.

Методы градиентного спуска по стратегиям не нуждаются в вычислении выгоды промежуточных действий, и эта оценка до сих пор выполнялась посредством имитации методом Монте-Карло, но данный процесс отличается высокой сложностью, и его нельзя использовать в онлайн-режиме.

Но оказывается, что обучение выгоде промежуточных действий становится возможным, если применять методы на основе функции оценки. Как и в предыдущем разделе, обозначим через $Q^p(s_t, a)$ значение (ценность) действия a , где p — стратегия, используемая сетью стратегии. Следовательно, теперь мы имеем две сети: сеть стратегии и Q-сеть. Сеть стратегии обучается вероятностям действий, а Q-сеть — значениям $Q^p(s_t, a)$ различных действий, чтобы предоставить сети стратегии оценку выгоды. Таким образом, сеть стратегии использует $Q^p(s, a)$ (с коррекцией на базовую линию) для взвешивания обновлений градиентного подъема. Q-сеть обновляется с использованием *обновлений, привязанных к стратегии* (on-policy update), как в методе SARSA, где стратегия контролируется сетью стратегии (а не ϵ -жадным алгоритмом). Однако, в отличие от Q-обучения, Q-сеть не принимает непосредственно решений относительно действий, поскольку решения о стратегии находятся вне сферы ее контроля (выходят за рамки ее роли критика). Поэтому сеть стратегий является актором, а сеть значений — критиком. Чтобы отличить сеть стратегий от Q-сети, обозначим вектор параметров сети стратегий через $\bar{\Theta}$, а вектор параметров Q-сети — через \bar{W} .

Обозначим через s_t состояние в момент времени t , а наблюдаемые входные признаки состояния — через \bar{X}_t . В связи с этим ниже мы будем использовать s_t и \bar{X}_t взаимозаменяемым образом. Рассмотрим ситуацию в момент времени t , в котором действие a_t наблюдается после состояния s_t с вознаграждением r_t . Тогда в отношении состояния в момент времени $(t + 1)$ применяется следующая последовательность действий.

1. Семплировать действие a_{t+1} , используя текущее состояние параметров в сети стратегий. Обратите внимание на то, что текущим состоянием является s_{t+1} , поскольку действие a_t уже наблюдалось.
2. Пусть $F(\bar{X}_t, \bar{W}, a_t) = \hat{Q}^p(s_t, a_t)$ представляет оценку значения $Q^p(s_t, a_t)$ Q-сетью с использованием наблюдаемого представления \bar{X}_t состояний и параметров \bar{W} . Оценить $Q^p(s_t, a_t)$ и $Q^p(s_{t+1}, a_{t+1})$, используя Q-сеть. Вычислить TD-ошибку δ_t с помощью следующей формулы:

$$\begin{aligned} \delta_t &= r_t + \gamma \hat{Q}^p(s_{t+1}, a_{t+1}) - \hat{Q}^p(s_t, a_t) = \\ &= r_t + \gamma F(\bar{X}_{t+1}, \bar{W}, a_{t+1}) - F(\bar{X}_t, \bar{W}, a_t). \end{aligned}$$

3. [Обновление параметров сети стратегий]. Пусть $P(\bar{X}_t, \bar{\Theta}, a_t)$ — вероятность действия a_t , предсказанная сетью стратегий. Обновить параметры сети стратегий:

$$\bar{\Theta} \leftarrow \bar{\Theta} + \alpha \hat{Q}^P(s_t, a_t) \nabla_{\bar{\Theta}} \log(P(\bar{X}_t, \bar{\Theta}, a_t)),$$

где α — скорость обучения для сети стратегий, а значение $\hat{Q}^P(s_t, a_t) = F(\bar{X}_t, \bar{W}, a_t)$ получается из Q-сети.

4. [Обновление параметров Q-сети]. Обновить параметры Q-сети:

$$\bar{W} \leftarrow \bar{W} + \beta \delta_t \nabla_w F(\bar{X}_t, \bar{W}, a_t),$$

где β — скорость обучения для Q-сети. Необходимо подчеркнуть, что скорость обучения для Q-сети обычно выше скорости обучения для сети стратегий.

Затем для наблюдения состояния s_{t+2} выполняется действие a_{t+1} и инкрементируется значение t . Следующая итерация этого подхода выполняется (путем повторения вышеописанных шагов) для этого инкрементированного значения t . Итерационный процесс продолжается до достижения сходимости. Значение $\hat{Q}^P(s_t, a_t)$ совпадает со значением $\hat{V}^P(s_{t+1})$.

Если мы используем $\hat{V}^P(s_t)$ в качестве базовой линии, то выгода $\hat{A}^P(s_t, a_t)$ определяется следующим образом:

$$\hat{A}^P(s_t, a_t) = \hat{Q}^P(s_t, a_t) - \hat{V}^P(s_t).$$

Это приводит к следующему изменению формулы обновления:

$$\bar{\Theta} \leftarrow \bar{\Theta} + \alpha \hat{A}^P(s_t, a_t) \nabla_{\bar{\Theta}} \log(P(\bar{X}_t, \bar{\Theta}, a_t)).$$

Обратите внимание на замену величины $\hat{Q}(s_t, a_t)$ в описании оригинального алгоритма величиной $\hat{A}(s_t, a_t)$. Одной из возможностей оценки значения $\hat{V}^P(s_t)$ является поддержание набора параметров, представляющих сеть значений (которая отличается от Q-сети). Для обновления параметров сети значений можно использовать TD-алгоритм. Однако оказывается, что для этого достаточно одной сети значений. Такое возможно потому, что вместо $\hat{Q}(s_t, a_t)$ мы можем использовать выражение $r_t + \gamma \hat{V}^P(s_{t+1})$. Это приводит к функции выгоды (advantage function), которая представляет собой то же самое, что и TD-ошибка:

$$\hat{A}^P(s_t, a_t) = r_t + \gamma \hat{V}^P(s_{t+1}) - \hat{V}^P(s_t).$$

Иными словами, нам нужна одна сеть значений (см. рис. 9.5), которая играет роль критика. Приведенный выше подход также можно обобщить для использования TD(λ)-алгоритма при любом значении λ .

9.5.5. Непрерывное пространство действий

Все методы, которые мы до сих пор обсуждали, связаны с дискретными пространствами состояний. Например, в видеоигре возможен дискретный набор вариантов выбора, таких как перемещение курсора вверх, вниз, влево или вправо. Однако в задачах робототехники возможны непрерывные пространства состояний, в которых, например, мы хотим, чтобы рука робота переместилась на определенное расстояние. Одной из возможностей является дискретизация действия в набор мелко градуированных интервалов и использование средней точки интервала в качестве представительного значения. После этого задачу можно трактовать как задачу с дискретным выбором. Однако такой подход нельзя считать вполне удовлетворительным. Во-первых, упорядоченность различных вариантов выбора будет утеряна из-за обработки упорядоченных в силу самой своей природы (числовых) значений как категориальных значений. Во-вторых, это приводит к взрывному расширению пространства возможных действий, особенно если такое пространство многомерно (например, отдельные измерения для расстояний, на которые перемещаются рука и нога робота). Результатом такого подхода может быть переобучение, а для обучения может потребоваться значительно больший объем данных.

Распространенный подход заключается в том, чтобы позволить нейронной сети выводить параметры непрерывного распределения (например, среднее значение и стандартное отклонение гауссовского распределения), а затем использовать семплирование из параметров этого распределения для вычисления значения действия на следующем шаге. Поэтому нейронная сеть будет выводить среднее значение μ и стандартное отклонение σ для расстояния, на которое перемещается рука робота, а фактическое действие a будет семплироваться из гауссиана $\mathcal{N}(\mu, \sigma)$ с этими параметрами:

$$a \sim \mathcal{N}(\mu, \sigma). \quad (9.26)$$

В данном случае действие a представляет расстояние, на которое перемещается рука робота. Обучение значениям μ и σ можно проводить с помощью обратного распространения ошибки. В некоторых вариациях данного подхода σ заранее фиксируется в качестве гиперпараметра, так что обучаться необходимо лишь среднему значению μ . В данном случае также применим прием с относительным правдоподобием, за исключением того, что вместо дискретной вероятности действия a мы используем логарифм плотности в точке a .

9.5.6. Преимущества и недостатки градиентного спуска по стратегиям

Методы градиентного спуска по стратегиям представляют наиболее естественный вариант выбора для таких приложений, как робототехника, с непрерывными пространствами состояний и действий. В случае многомерных непрерывных пространств действий количество возможных комбинаций действий может быть очень большим. Поскольку методы Q-обучения требуют вычисления максимального Q-значения, определенного на всей совокупности действий, то с вычислительной точки зрения такой подход может оказаться нереализуемым на практике. К тому же методы градиентного спуска по стратегиям отличаются стабильностью и хорошей сходимостью, однако этому сопутствует риск попадания в локальные минимумы. По сравнению с ними методы Q-обучения менее стабильны в отношении сходимости и иногда могут приводить к осцилляциям в окрестности отдельных решений, но при этом обеспечивают лучшие возможности для достижения глобальных оптимумов.

Методы градиентного спуска по стратегиям обладают тем дополнительным преимуществом, что они обеспечивают обучение стохастическим стратегиям, улучшающим производительность в условиях (таких, как игры с угадыванием), когда детерминистические стратегии оказываются неоптимальными из-за того, что ими может воспользоваться противник. Q-обучение предоставляет детерминированные стратегии, а это означает, что в случае подобных задач градиентный спуск по стратегиям, который обеспечивает семплирование действий из вероятностного распределения их возможной совокупности, является более предпочтительным.

9.6. Поиск по дереву методом Монте-Карло

Поиск по дереву методом Монте-Карло — это способ улучшения прогностической силы обученных политик и значений на стадии вывода за счет их сочетания с опережающим просмотром. Данное улучшение также предоставляет базис для бутстрэппинга на основе опережающего просмотра, как в обучении методом временных разностей. Кроме того, его используют в качестве вероятностной альтернативы детерминированным минимаксным деревьям, которые применяются в обычных игровых программах (хотя они не ограничены только играми). Каждый узел дерева соответствует состоянию, а каждая ветвь — возможному действию. По мере того как в процессе поиска встречаются новые состояния, это дерево постепенно растет. Целью поиска по дереву является выбор ветви, наилучшей для рекомендации прогнозируемого действия агента. С каждой ветвью связывается значение, определенное на основании предыдущих исходов в дереве поиска, начинающемся с этой ветви, и “бонуса” верхней

границы, который уменьшается с расширением диапазона разведочного поиска. Это значение служит для установления приоритета ветвей в процессе разведочных действий. Полезность ветви корректируется после каждого разведочного действия, поэтому при выполнении последующих разведочных действий предпочтение отдается ветвям с положительным исходом.

Ниже в качестве иллюстративного примера описан поиск по дереву методом Монте-Карло, который применяется в программе AlphaGo. Предположим, что вероятность $P(s, a)$ каждого действия (хода) a в состоянии (позиция на доске) s можно оценить с помощью сети стратегий. В то же время для каждого хода мы имеем величину $Q(s, a)$, которая служит мерой качества хода a в состоянии s . Например, значение $Q(s, a)$ увеличивается с ростом числа побед, следующих за действием a из состояния s в процессе имитации. Система AlphaGo применяет более сложный алгоритм, включающий дополнительные нейтральные оценки позиции на доске после нескольких ходов (см. уравнение 9.7.1). Затем на каждой итерации вычисляется верхняя граница $u(s, a)$ качества хода a в состоянии s , которая определяется следующим выражением:

$$u(s, a) = Q(s, a) + K \cdot \frac{P(s, a) \sqrt{\sum_b N(s, b)}}{N(s, a) + 1}, \quad (9.27)$$

где $N(s, a)$ — количество событий, заключающихся в совершении действия a в состоянии s в процессе выполнения поиска по дереву методом Монте-Карло. Иными словами, верхняя граница получается сложением качества $Q(s, a)$ и “бонуса”, зависящего от вероятности $P(s, a)$ и количества раз, когда совершалось действие, соответствующее данной ветви. Идея масштабирования величины $P(s, a)$ в зависимости от количества посещений соответствующего узла дерева заключается в уменьшении вклада часто посещаемых ветвей и поощрении разведочных действий. Подход Монте-Карло основан на стратегии отбора ветви с наибольшей верхней границей, как в методах “многоруких бандитов” (см. раздел 9.2.3). Здесь второй член в правой части уравнения 9.27 играет роль доверительного интервала для вычисления верхней границы. По мере выполнения все большего и большего количества ходов, соответствующих данной ветви, ее разведочный “бонус” уменьшается в связи с уменьшением ее доверительного интервала. Гиперпараметр K управляет размером “бонуса”.

В любом заданном состоянии выполняется действие a , которому соответствует наибольшее значение $u(s, a)$. Этот подход применяется рекурсивно до того момента, когда выполнение оптимального действия уже не приведет к существующему узлу. Тогда это новое состояние s' добавляется в качестве узла листа (концевого узла), величины $N(s', a)$ и $Q(s', a)$ которого инициализируются нулевыми значениями. Обратите внимание на то, что процесс имитации,

выполняемый вплоть до конечного узла, полностью детерминирован и не включает рандомизацию, поскольку величины $P(s, a)$ и $Q(s, a)$ вычисляются детерминированным образом. Имитация методом Монте-Карло применяется лишь для оценки значения вновь добавляемого конечного узла s' . В частности, развертки методом Монте-Карло в сети стратегий (например, получаемые с использованием $P(s, a)$ для семплирования действий) возвращают либо +1, либо -1, в зависимости от того, является ли конечный результат выигрышем или проигрышем соответственно. В разделе 9.7.1 мы обсудим альтернативные варианты вычислений для конечных узлов, в которых также используются сети значений. После вычисления значения листового узла соответствующим образом обновляются значения $Q(s'', a'')$ и $N(s'', a'')$ для всех ребер (s'', a'') на пути от текущего состояния s к листу s' . Значение $Q(s'', a'')$ поддерживается как среднее значение результатов вычислений для всех конечных узлов, достигаемых из данной ветви в процессе поиска по дереву методом Монте-Карло. После многократного проведения поиска из узла s выбирается наиболее часто посещаемое ребро, которое принимается в качестве желаемого действия.

9.6.1. Использование в бутстрэппинге

Поиск по дереву методом Монте-Карло традиционно использовался в процессе вывода, а не в процессе обучения. Однако, поскольку этот метод обеспечивает улучшенную оценку величины $Q(s, a)$ значения пары “состояние — действие” (в результате опережающего просмотра), его можно применять также в целях бутстрэппинга (см. примечание 9.4.1). Поиск по дереву методом Монте-Карло представляет собой отличную альтернативу n -шаговым методам временных разностей. Подобные методы, привязанные к стратегии, имеют ту особенность, что они исследуют одиночную n -шаговую последовательность с использованием ϵ -жадной стратегии, что, как правило, делает их (с увеличением глубины, но не ширины разведочного поиска) слишком слабыми. Одним из способов их усиления является исследование всех возможных n -последовательностей и выбор наиболее оптимальной из них в сочетании с методикой, не привязанной к стратегии (например, основанной на обобщении 1-шагового подхода Беллмана). Фактически именно такой подход (позже получивший название *TD-Leaf* [22]) был применен в программе Сэмюэля для игры в шашки [421], в которой наилучший из вариантов использовался в минимаксном дереве для бутстрэппинга. Необходимость исследования всех возможных n -последовательностей усложняет процесс. Поиск по дереву методом Монте-Карло может предоставить надежную альтернативу бутстрэппингу, поскольку он позволяет исследовать множество ветвей, исходящих из узла, для генерации усредненных целевых значений. Например, эталонные значения, полученные на основе опережающего просмотра, могут использовать усредненные характеристики для всех разведочных действий, начинающихся в заданном узле.

В программе *AlphaGo Zero* [447] осуществляется бутстрэппинг политик, а не значений состояний, что встречается крайне редко. В ней используются относительные вероятности посещения ветвей каждого узла в качестве апостериорных вероятностей действий в данном состоянии. Эти апостериорные вероятности улучшаются на основе вероятностных выходов сети стратегий благодаря тому факту, что решения относительно посещения узлов принимаются с учетом знаний относительно будущих узлов (т.е. на основе результатов вычислений для более глубоких узлов дерева Монте-Карло). Поэтому апостериорные вероятности применяются в качестве эталонных значений по отношению к вероятностям сети стратегий и служат для обновления параметров весов (см. раздел 9.7.1.1).

9.7. Типовые примеры

Ниже рассмотрены типовые примеры, взятые из реальной практики, которые демонстрируют применение обучения с подкреплением в различных задачах. Мы покажем примеры использования обучения с подкреплением в игре го, робототехнике, диалоговых системах, беспилотных автомобилях, а также в обучении гиперпараметров нейронных сетей.

9.7.1. AlphaGo: достижение чемпионского уровня в игре го

Го — настольная игра, в которой участвуют два человека. Сложность настольной игры для двоих в значительной мере зависит от размеров доски и количества допустимых шагов в каждой позиции. Простейший пример такой игры — крестики-нолики с размером доски 3×3 , и большинство людей в состоянии найти ее оптимальное решение без помощи компьютера. Шахматы, доска которых имеет размер 8×8 , — значительно более сложная игра, хотя в настоящее время существуют интеллектуальные вариации применения метода грубой силы для *селективного* исследования минимаксного дерева ходов вплоть до некоторой глубины, способные превзойти сегодняшних чемпионов. Игра го занимает предельное положение в отношении сложности, поскольку размеры ее доски составляют 19×19 .

В го играют два игрока: один — белыми, другой — черными *камнями*, которые хранятся в чашах рядом с доской. Иллюстративный пример доски го приведен на рис. 9.7. Игра начинается с пустой доски, на которую игроки выставляют камни. Первыми ходят черные, в чаше которых первоначально находится 181 камень, тогда как белые начинают, имея 180 камней. Общее количество связей равно общему количеству камней в чашах двух игроков. Делая ход, игрок помещает камень своего цвета (достав его из чаши) в определенную позицию, после чего не имеет права перемещать его. Камень противника можно захватить,

окружив его. Целью игры для каждого игрока является установление контроля над большей частью доски, чем та, которую контролирует его соперник, окружая ее своими камнями.

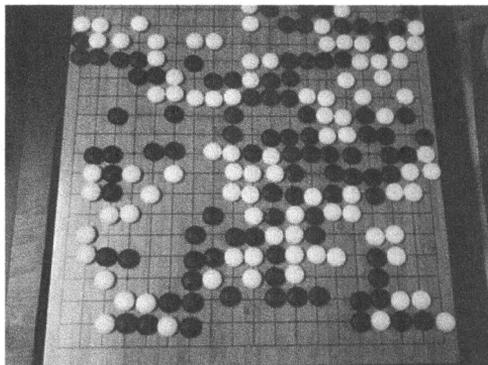


Рис. 9.7. Пример доски го с камнями

Если в шахматах в отдельной позиции можно сделать около 35 возможных ходов (коэффициент ветвления дерева), то среднее количество возможных ходов в отдельной позиции в игре го составляет 250, т.е. на порядок большую величину. Кроме того, среднее количество последовательных ходов (глубина дерева) в игре го составляет около 150, что примерно в два раза больше, чем в шахматах. Все эти аспекты делают го намного более трудным кандидатом с точки зрения автоматизации игры. Типичная стратегия программ для игры в шахматы заключается в конструировании минимаксного дерева всех сочетаний ходов, которые могут сделать игроки, вплоть до определенной глубины, и последующем вычислении конечных позиций на доске на основе эвристик, специфических для шахмат (таких, как количество оставшихся фигур и тактические угрозы). Неоптимальные части этого дерева отсекаются эвристическим способом. Такой подход — просто улучшенная версия стратегии грубой силы, в которой исследуются все позиции вплоть до определенной глубины. Даже в случае анализа умеренной глубины (просчет 20 ходов для каждого игрока) количество узлов в минимаксном дереве го превысит количество атомов в наблюдаемой части Вселенной. Поскольку в подобных условиях очень важно интуитивно чувствовать пространство, люди всегда справляются с игрой лучше, чем стратегии игры в го на основе метода грубой силы. Использование обучения с подкреплением в го приближает этот подход к тому способу, который пытается использовать человек. Мы редко стремимся исследовать все возможные комбинации ходов; вместо этого мы визуально изучаем различные конфигурации фигур на шахматной доске, чтобы получить представление о выигрышных позициях, и пытаемся совершать ходы в тех направлениях, которые сулят получение преимущества.

Автоматическое обучение признакам пространственных конфигураций, позволяющим предсказывать наилучшие ходы, осуществляется с помощью сверточной нейронной сети. Состояние системы кодируется в определенной точке позиции на доске, хотя представление доски в AlphaGo включает ряд дополнительных признаков, касающихся статуса связей или количества ходов, выполненных с того момента, когда был сделан ход камнем. Для представления знаний о состоянии в полном объеме требуется несколько таких пространственных карт. Например, одна карта признаков представляет состояние каждой связи, количество кругов, сыгранных с того момента, когда был сделан ход камнем, и т.п. Целочисленные карты признаков кодируются с использованием нескольких плоскостей в представлении прямого кодирования. В целом игровая доска может быть представлена с использованием 48 бинарных плоскостей размером 19×19 пикселей.

Для обучения удачным стратегиям выполнения ходов в различных позициях с помощью сети обучения стратегий алгоритм AlphaGo учитывает свой опыт побед и проигрышей, накопленный в повторных играх (используя как ходы, сделанные профессиональными игроками, так и ходы, сделанные в играх с самим собой). Кроме того, оценка каждой позиции на доске го осуществляется с помощью сети значений. Впоследствии для окончательного вывода применяется поиск по дереву методом Монте-Карло. Поэтому AlphaGo представляет собой многозвенную модель, компоненты которой обсуждаются в следующих разделах.

Сети стратегий

Сеть стратегий принимает в качестве входа вышеупомянутое визуальное представление доски и выводит вероятность выполнения действия a в состоянии s . Эта вероятность обозначается как $p(s, a)$. Обратите внимание на то, что в игре го действиям соответствуют вероятности помещения камня в каждую допустимую позицию на доске. Поэтому в выходном слое используется Softmax-активация. Для обучения двух отдельных сетей стратегий использовались различные подходы. Две указанные сети имели идентичную структуру, содержащую сверточные слои с нелинейностями ReLU. Каждая сеть содержала 13 слоев. В большинстве сверточных слоев, кроме первого и последнего, свертка выполнялась с использованием фильтров 3×3 . В первом и последнем слоях применялись соответственно фильтры 5×5 и 1×1 . Для поддержания размеров сверточных слоев использовалось дополнение нулями, а общее количество фильтров составляло 192. Была задействована нелинейность ReLU; пулинг по максимальному значению не применялся.

Эти сети последовательно обучались следующим образом.

- *Обучение с учителем.* В качестве тренировочных данных использовались выбираемые случайным образом примеры профессиональных игр. Вход служил состоянием сети, а выходом было действие, совершаемое

профессиональным игроком. Оценка (польза) такого хода всегда принималась равной +1, поскольку цель тренировки заключалась в том, чтобы научить сеть имитировать ходы профессионалов (так называемое *имитационное обучение*). Функцией выгоды в этой сети в процессе обратного распространения ошибки служила логарифмическая вероятность выбранного действия. Эта сеть была названа *сетью SL-стратегий* (от *Supervised Learning* — обучение с учителем). Следует отметить, что подобные контролируемые формы имитационного обучения довольно часто применяются в обучении с подкреплением во избежание проблем “холодного запуска”. Однако, как было показано в [446], лучшим вариантом является отказ от использования такой формы обучения.

- *Обучение с подкреплением.* В этом случае для тренировки сети применялось обучение с подкреплением. Одной из проблем является то, что в игре го участвуют два игрока, поэтому для генерирования ходов сеть играла сама с собой. В соответствии с этим текущая сеть всегда играла со случайно выбранной сетью из числа тех, которые встречались несколько итераций тому назад, чтобы обучение с подкреплением могло иметь пул рандомизированных противников. Игра доигрывалась до самого конца, и с каждым ходом ассоциировалась оценка выгоды +1 или -1, в зависимости от того, закончилась игра победой или поражением. Затем эти данные использовались для обучения сети стратегий. Эта сеть была названа *сетью RL-стратегий* (от *Reinforcement Learning* — обучение с подкреплением).

Следует отметить, что уже эти сети представляли собой довольно грозных игроков в го по сравнению с лучшими из имеющихся на то время программ, но они были дополнительно усилены за счет объединения с поиском по дереву Монте-Карло.

Сеть значений

Эта сеть также была сверточной нейронной сетью, которая использует состояние сети в качестве входа, а предсказанную оценку в диапазоне значений $[-1, +1]$ — в качестве выхода, где значение +1 указывает на идеальную вероятность, равную 1. Выходом является предсказанная оценка для следующего игрока, независимо от того, играет он белыми или черными, поэтому вход кодирует также “цвет” камней в терминах “игрок” или “противник”, а не “черные” или “белые”. Архитектура сети значений очень напоминала архитектуру сети стратегий, если не считать некоторых отличий входа и выхода. Вход содержал дополнительный признак, указывающий на то, играет ли следующий игрок белыми или черными камнями. Оценка вычислялась с использованием одного

нелинейного элемента в виде гиперболического тангенса в конце сети, поэтому ее значение лежит в диапазоне $[-1, +1]$. Ранние сверточные слои сети значительны — те же, что и в сети стратегий, хотя в слое 12 добавлен дополнительный сверточный слой. За последним слоем, содержащим 256 элементов и ReLU-активацию, следует завершающий сверточный слой. Одним из возможных вариантов тренировки сети является использование позиций из набора данных [606] игр го. Однако более предпочтительным оказалось генерирование данных в процессе игры сети с самой собой с использованием сетей SL-и RL-стратегий до завершения игры, чтобы генерировались окончательные исходы. Для тренировки сверточной нейронной сети использовались пары “состояние — исход”. Поскольку позиции в пределах одной игры коррелированы, их последовательное использование в процессе тренировки приводит к переобучению. Во избежание переобучения, обусловленного тесной корреляцией между тренировочными примерами, было очень важно семплировать позиции из разных игр. Поэтому каждый тренировочный пример брался из другой игры, сыгранной сетью с самой собой.

Поиск по дереву методом Монте-Карло

Для разведочного анализа применялся упрощенный вариант уравнения 9.27, в котором для гиперпараметра K в каждом узле s устанавливается значение $1 / \sqrt{\sum_b N(s, b)}$. В разделе 9.6 была описана версия поиска по дереву методом Монте-Карло, в которой для оценки листовых узлов использовалась лишь сеть RL-стратегий. В случае AlphaGo комбинируются два подхода. Во-первых, путем доигрывания позиции из листового узла до конца методом Монте-Карло создавалась оценка e_1 . И хотя для доигрывания позиций можно было применять сеть стратегий, алгоритм AlphaGo обучал упрощенный Softmax-классификатор, используя базу данных профессиональных игр и некоторые созданные вручную признаки, что позволило ускорить процессы доигрывания. Во-вторых, сеть значений создавала независимую оценку e_2 для листовых узлов. Окончательная оценка e получалась в виде линейной комбинации этих двух оценок: $e = \beta e_1 + (1 - \beta) e_2$. Значение $\beta = 0,5$ обеспечило наилучшую производительность, хотя к близкой производительности приводило использование только сети значений. Наиболее часто посещаемая ветвь в дереве поиска Монте-Карло выбиралась в качестве предсказанного хода.

9.7.1.1. AlphaGo Zero: улучшения, не требующие профессиональных знаний

Усовершенствованный вариант AlphaGo, получивший название *AlphaGo Zero* [446], устранил необходимость в использовании ходов из профессиональных

партий (т.е. SL-сети). В AlphaGo Zero вместо отдельных сетей стратегий и значений одна сеть выводит как стратегию (т.е. вероятности действий) $p(s, a)$, так и оценку $v(s)$ позиции. Кросс-энтропийные потери выходных вероятностей стратегии и квадратичные потери выходных оценок суммировались для получения единой функции потерь. Если в оригинальной версии AlphaGo дерево поиска Монте-Карло применялось лишь для создания вывода обученных сетей, то *версии с нулевым знанием* используют счетчики посещений ветвей в дереве поиска Монте-Карло также для тренировки сети. Счетчики посещений каждого узла дерева поиска можно рассматривать в качестве оператора, воздействующего на $p(s, a)$ и *улучшающего* стратегию за счет опережающего разведочного поиска. Тем самым создается основа для самозагрузки начальных эталонных значений (см. примечание 9.4.1) для обучения нейронной сети. Если при обучении методом временных разностей процедура бутстрэппинга создает начальные значения состояний, то в этом подходе бутстрэппинг инициализирует счетчики посещений для стратегий обучения. Предсказанная поиском по дереву Монте-Карло вероятность действия a в состоянии доски s подчиняется соотношению $\pi(s, a) \propto N(s, a) / \tau$, где τ — параметр температуры. Значение $N(s, a)$ вычисляется с использованием поиска Монте-Карло, аналогичного тому, который применяется в AlphaGo, при этом *априорные* вероятности $p(s, a)$, выводимые нейронной сетью, используются для вычисления уравнения 9.27. Значение $Q(s, a)$ в уравнении 9.27 устанавливается равным среднему значению выхода $v(s')$ нейронной сети для вновь создаваемых листовых узлов, достигаемых из состояния s .

AlphaGo Zero обновляет нейронную сеть бутстрэппингом $\pi(s, a)$ как эталонного значения, тогда как эталонные значения состояний генерируются имитациями Монте-Карло. В каждом состоянии s вероятности $\pi(s, a)$, значения $Q(s, a)$ и счетчики посещений $N(s, a)$ обновляются путем (многократного) выполнения процедуры поиска по дереву Монте-Карло, начиная с состояния s . Нейронная сеть из предыдущей итерации используется для выбора ветвей в соответствии с уравнением 9.27, пока не будет достигнуто конечное состояние или состояние, которого в дереве не существует. Для каждого несуществующего состояния в дерево добавляется новый лист, Q -значения и счетчики посещаемости которого устанавливаются равными нулю. Q -значения и счетчики посещаемости всех ребер вдоль пути от s к листовому узлу обновляются на основе вычисления листа нейронной сетью или по правилам игры для конечных состояний. После выполнения многократного поиска, начинающегося с узла s , *апостериорная* вероятность $\pi(s, a)$ используется для семплирования действия в процессе игры сети с самой собой и достижения следующего узла s' . Вся процедура повторяется в узле s' для рекурсивного получения следующей позиции s'' . Игра рекурсивно продолжается до завершения, и в качестве эталонного значения $z(s)$

состояний s , однородно семплируемых вдоль пути игры, возвращается конечное значение из набора $\{-1, +1\}$. Отметим, что значение $z(s)$ определяется с точки зрения игрока в состоянии s . Эталонные значения вероятностей уже доступны в $\pi(s, a)$ для различных значений a . Поэтому можно создать тренировочный пример для нейронной сети, содержащей входное представление состояния s , получить бутстрэппингом эталонные вероятности $\pi(s, a)$ и эталонное значение $z(s)$ метода Монте-Карло. Этот тренировочный пример используется для обновления параметров нейронной сети. Если выходными вероятностями и значениями для нейронной сети являются $p(s, a)$ и $v(s)$ соответственно, то функция потерь для нейронной сети с вектором весов \bar{W} определяется следующим образом:

$$L = [v(s) - z(s)]^2 - \sum_a \pi(s, a) \log[p(s, a)] + \lambda \|\bar{W}\|^2, \quad (9.28)$$

где $\lambda > 0$ — параметр регуляризации.

Дополнительные улучшения были предложены в виде программы AlphaZero [447], способной играть в несколько игр, таких как го, сеги и шахматы. AlphaZero легко победила наилучшие шахматные программы, *Stockfish*, а также наилучшую программу для игры в сеги (*Elmo*). Победа в шахматах была особенно неожиданной для большинства лучших игроков, поскольку всегда предполагалось, что системе обучения с подкреплением требуется передать слишком большой объем сведений, чтобы она могла выиграть у системы, использующей вычисления с признаками, подготовленными вручную.

Замечания по поводу производительности

Программа AlphaGo продемонстрировала экстраординарное мастерство в состязаниях с самыми разными компьютерными и “живыми” игроками. В играх с компьютером она выиграла 494 игры из 495 [445]. Даже в тех случаях, когда игру AlphaGo затрудняли, предоставляя противнику фору в виде четырех свободных камней, она выиграла 77, 86 и 99% игр против компьютерных программ *Crazy Stone*, *Zen* и *Pachi* соответственно. Она также нанесла поражение известным профессиональным игрокам, в том числе чемпионам Европы и мира, и другим игрокам с высоким рейтингом.

Интересно, однако, то, каким образом достигались эти победы. В нескольких из этих игр AlphaGo делала необычные блистательные ходы, смысл которых прояснялся лишь в результате послематчевого анализа партий [607,608]. Были случаи, когда ходы, сделанные программой AlphaGo, на первый взгляд противоречили здравому смыслу, но в конечном счете оказывалось, что это новинки, которым AlphaGo научилась, играя сама с собой. После таких матчей некоторые лучшие игроки в го кардинально пересматривали свои взгляды на стратегию игры в целом.

Аналогичный уровень мастерства программа AlphaZero продемонстрировала и в шахматах, в которых она часто жертвовала материальным преимуществом для улучшения своей позиции и стеснения соперника. Такой тип поведения свойствен людям и отличается от поведения традиционных шахматных программ (которые уже играют лучше людей). В отличие от вычислений с признаками, конструируемыми вручную, в программу не были заранее вложены какие-либо понятия о материальной ценности фигур или об условиях, в которых позиция короля в центре доски может считаться безопасной. Кроме того, в процессе игры самой с собой программа открыла для себя многие из хорошо известных дебютных начал, и создавалось впечатление, что у нее есть собственное мнение относительно того, что “лучше”, а что “хуже”. Иными словами, она обладала возможностью самостоятельно открывать знания. Ключевое отличие обучения с подкреплением от обучения с учителем состоит в том, что *оно способно вносить инновации, выходящие за пределы известных знаний, посредством обучения методом проб и ошибок под управлением вознаграждения*. Такое поведение открывает многообещающие перспективы в других приложениях.

9.7.2. Самообучающиеся роботы

Самообучающиеся роботы представляют важное направление исследований в области искусственного интеллекта, цель которого — обучение роботов выполнению различных задач, таких как локомоция, механический ремонт или поиск объектов. В качестве примера рассмотрим случай, когда требуется сконструировать робота с возможностями *физической* локомоции (имеются в виду особенности его конструкции и набор доступных движений), но такого, который должен обучиться точному выбору движений, чтобы переместиться из точки А в точку В, все время находясь в устойчивом равновесии. Люди, будучи двуногими существами, способны ходить, сохраняя равновесие естественным образом, и мы даже не думаем о том, как этого добиться, но для двуногого робота это очень непростая задача, и любое неверное движение может легко привести к его опрокидыванию. Трудности такой задачи еще более усугубляются, если робот действует на неизвестной местности и на его пути могут встречаться препятствия.

Задачи описанного типа естественным образом вписываются в обучение с подкреплением, поскольку можно легко судить о том, правильно ли ходит робот, но очень тяжело специфицировать точные правила, которые определяли бы, что должен делать робот в каждой возможной ситуации. В подходе на основе обучения с подкреплением, управляемого вознаграждением, робот получает (виртуальное) поощрение всякий раз, когда достигает прогресса в перемещении из точки А в точку В. Во всем остальном роботу предоставляется полная свобода действий, и это не предварительное обучение с использованием знаний о

конкретном выборе действий, которое помогло бы роботу сохранять баланс и ходить. Иными словами, он не пичкается всевозможными знаниями о том, что собой представляет ходьба (за исключением того, что он будет получать вознаграждение за использование доступных действий, обеспечивающих продвижение из точки А в точку В). Это классический пример обучения с подкреплением, поскольку теперь робот нуждается в обучении конкретной последовательности действий, за выполнение которой он получит соответствующее вознаграждение. Несмотря на то что в данном случае мы используем в качестве частного примера локомоцию, этот общий принцип применим к любому типу обучения роботов. В качестве примера можно привести такие задачи, как захват роботом объекта или откручивание бутылочного колпачка. Оба этих случая кратко обсуждаются ниже.

9.7.2.1. Глубокое обучение навыкам локомоции

В данном случае локомоции обучались виртуальные роботы [433], имитируемые с помощью физического движка *MuJoCo* [609], название которого является сокращением от *Multi-Joint Dynamics with Contact* (многозвенная динамика с возможностью контакта). Этот физический движок облегчает проведение исследований и разработку в областях робототехники, биомеханики, графики и анимации, где существует потребность в быстрой и точной имитации процессов без фактического конструирования реальных роботов. Для этих целей применялись как роботы-гуманоиды, так и четвероногие роботы. Пример двуногой модели показан на рис. 9.8. Преимуществами имитации этого типа являются невысокая себестоимость виртуальной имитации и избежание естественных рисков, связанных с вопросами безопасности и материальными расходами, которые могут возникнуть ввиду физического повреждения реальных физических объектов в процессе проведения экспериментов, что вполне возможно из-за высокой вероятности ошибочных/неосторожных действий. С другой стороны, физическая модель позволяет получить более реалистичные результаты. В целом имитацию нередко можно применять в целях мелкомасштабного тестирования, прежде чем приступать к созданию физической модели.

Модель гуманоида имела состояние, характеризующееся 33 измерениями, и 10 степеней свободы, в то время как состояние четырехногой модели характеризовалось 29 измерениями, и она имела 8 степеней свободы. Модели вознаграждались за передвижение в заданном направлении, однако эпизоды преждевременно прерывались, если центр масс робота опускался ниже определенной точки. Действия робота управлялись механизмами передачи крутящего момента. Роботу был доступен ряд признаков, поступающих от датчиков положений препятствий, позиций сочленений, углов и т.п. Эти признаки поступали в нейронную сеть. Использовались две нейронные сети: одна оценивала значения

(ценность) действий, а вторая — стратегии. В связи с этим был применен метод градиентного спуска по стратегиям, в котором сеть значений была задействована для оценки полезности действий. Такой подход представляет собой реализацию метода “актер — критик”.

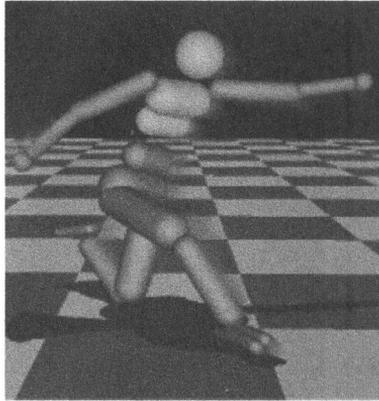


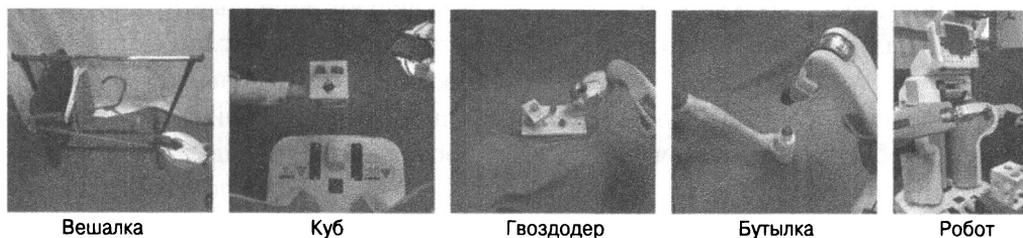
Рис. 9.8. Пример виртуального робота-гуманоида (оригинальное изображение см. в [609])

Применялась нейронная сеть прямого распространения с тремя скрытыми слоями, содержащими соответственно 100, 50 и 25 элементов с нелинейностью в виде гиперболического тангенса. Используемый в [433] подход требует оценки как функции стратегии, так и функции значения, и в обоих случаях для скрытых слоев применялась одна и та же архитектура. В то же время для сети оценки значений требуется только один выход, тогда как для сети оценки стратегий количество выходов определяется количеством действий. Поэтому основные различия в этих архитектурах относились к выходному слою и применяемой функции потерь. Обобщенная оценка (GAE) использовалась в сочетании с оптимизацией стратегии в пределах доверительной области (TRPO). Ссылки на подробное описание этих методов вы найдете в библиографической справке. В результате тренировки нейронной сети с помощью обучения с подкреплением в течение 1000 итераций робот приобрел внешне безукоризненную походку. Видеоролик, демонстрирующий окончательные результаты обучения робота хождению, доступен по ссылке [610]. Впоследствии компания Google DeepMind опубликовала еще более впечатляющие результаты, демонстрирующие способность робота избегать препятствий и справляться с другими трудностями [187].

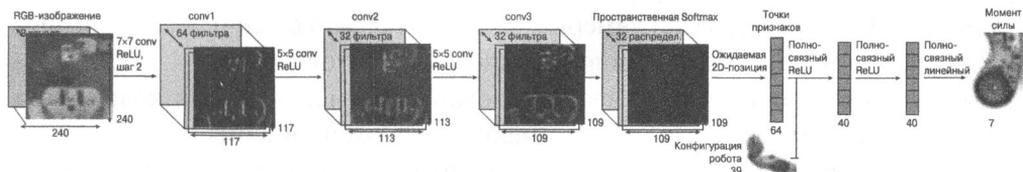
9.7.2.2. Глубокое обучение навыкам визуальной моторики

Другой интересный пример использования обучения с подкреплением описан в [286], где робот обучался нескольким задачам, требующим тесной координации его зрительных и моторных функций. В частности, робот научился

вешать одежные плечики на вешалку, вставлять фигуру определенной формы в сортировочный куб, подводить гвоздодер под шляпку гвоздя и закручивать крышку на бутылке. Примеры выполнения этих задач приведены на рис. 9.9, а, вместе с изображением робота. Робот выполнял действия с помощью приводного механизма, имеющего 7 подвижных соединений, управляемых командами. Для оптимального выполнения задачи каждое действие требовало подачи последовательности соответствующих команд. В данном случае обучалась физическая модель робота. Для определения местоположения объектов и манипулирования ими робот использовал изображение, поступающее от камеры. Эта камера была своего рода глазами робота, а работа сверточной нейронной сети была основана на принципах функционирования зрительной коры (установленных экспериментами Хубеля и Визеля). Несмотря на кажущиеся значительные отличия этой задачи от видеоигр Atari, между ними существует значительное сходство в том, каким образом кадры изображения можно преобразовывать в стратегию действий. Например, в случае видеоигр Atari сверточная сеть также имеет дело с необработанными пикселями. Однако в рассматриваемом нами случае есть дополнительные входы, соответствующие позициям робота и объекта. Эти задачи требуют обучения элементам визуального восприятия, координации и контактной динамики, причем это обучение должно осуществляться автоматически.



а) Обучение робота видеомоторным навыкам



б) Архитектура сверточной нейронной сети

Рис. 9.9. Глубокое обучение навыкам видеомоторики; рисунки взяты из [286] (© 2016 Sergey Levine, Chelsea Finn, Trevor Darrell and Pieter Abbeel)

Для преобразования кадров изображения вполне естественно использовать сверточную нейронную сеть. Как и в случае игр Atari, слои сверточной сети

должны быть обучены пространственным признакам, приносящим соответствующее вознаграждение за выполнение действий, полезных в рамках данной задачи. Эта сверточная нейронная сеть имеет 7 слоев и 92 000 параметров. Первые три слоя были сверточными, четвертый — пространственным слоем с активацией Softmax, а пятый выполнял фиксированное преобразование пространственных карт признаков в компактный набор из двух координат. Идея заключалась в применении функции Softmax ко всей карте пространственных признаков. Это дает вероятность каждой позиции в карте признаков. Ожидаемая позиция, полученная с использованием такого распределения вероятностей, предоставляет двухмерные координаты точки, которая называется *точкой признака* (feature point). Отметим, что каждая пространственная карта признаков в сверточном слое создает одну точку признака, которую можно рассматривать как результат применения своего рода операции *soft argmax* к указанному распределению пространственной вероятности. Пятый слой довольно заметно отличался от того, с чем мы обычно встречаемся в сверточных нейронных сетях, и предназначался для создания точного представления визуальной сцены, пригодного для организации обратной связи. Точки пространственных признаков конкатенируются с конфигурацией робота, что является дополнительным входом, встречающимся только после сверточных слоев. Этот набор конкатенированных признаков поступает в два полносвязных слоя. Каждый из них содержит 40 полулинейных элементов, за которыми следуют линейные соединения с элементами привода. Обратите внимание на то, что в первый слой сверточной нейронной сети поступали лишь наблюдения, соответствующие камере, тогда как наблюдения, соответствующие роботу, подавались в первый полносвязный слой. Это обусловлено тем, что состояния робота мало что дадут сверточным слоям, так что имеет смысл конкатенировать входы, связанные с состоянием, после обработки визуальных входов сверточными слоями. В целом сеть содержала 92 000 параметров, из которых 86 000 относились к сверточным слоям. Архитектура сверточной нейронной сети приведена на рис. 9.9, б. Наблюдения состоят из RGB-изображений камеры, показаний датчиков сочленений, скоростей и положений конечных исполнительных элементов.

Полные состояния робота характеризовались измерениями в количестве от 14 до 32, такими как углы в сочленениях, положения конечных исполнительных элементов, позиции объектов и их скорости. Набор этих величин образует практическую реализацию понятия состояния. Как и во всех методах, основанных на стратегиях, выходы соответствуют различным действиям (приводным механизмам). Интересным аспектом подхода, который обсуждается в [286], является то, что он преобразует задачу обучения с подкреплением в задачу обучения с учителем. Использовался метод *управляемого поиска стратегий* (guided policy search), который в этой главе не обсуждался. Заинтересованных

читателей отсылаем к [286], где также приведен видеоролик, демонстрирующий работу робота (обученного с использованием этой системы).

9.7.3. Создание разговорных систем: глубокое обучение чат-ботов

Чат-боты также называют *разговорными* или *диалоговыми* системами. Конечной целью чат-ботов является создание агента, способного свободно общаться с человеком на различные темы. Пока что мы еще очень далеки от достижения этой цели. Однако в создании чат-ботов для определенных задач (таких, как переговоры или помощник по совершению покупок) достигнут значительный прогресс. В качестве примера можно привести относительно универсальную систему Apple Siri, представляющую собой персонального цифрового помощника. Siri можно считать универсальной системой, поскольку она допускает общение по широкому ряду вопросов. Каждый, кто пользуется системой Siri, согласится с тем, что в некоторых случаях помощник не в состоянии дать удовлетворительный ответ на трудный вопрос, тогда как в других случаях очень бойко отвечает на вопросы общего характера, ответы на которые заранее запрограммированы. Конечно же, в этом нет ничего удивительного, поскольку система задумывалась как относительно универсальная, и мы пока что не приблизились к созданию разговорной системы, возможности которой были бы сравнимы с возможностями человека. В отличие от этого узкоспециальные системы предназначены для выполнения вполне конкретных заданий и поэтому легче поддаются надежному обучению.

Ниже мы опишем систему, созданную компанией *Facebook* для сквозного обучения торговым переговорам [290]. Это узкоспециализированная система, поскольку она предназначена специально для ведения переговоров. В качестве тестовой модели использовалась следующая задача. Двум агентам предоставляется коллекция товаров разного типа (например, две книги, одна шляпа, три воздушных шарика). Агентам поручается разделить эти товары между собой, договариваясь о том, как они будут распределены. Важный момент состоит в том, что ценность каждого типа товара различна для каждого из двух агентов, но ни один из них не знает, какую ценность данный товар представляет для другого агента. Подобные ситуации часто встречаются в процессе реальных переговоров, когда пользователи пытаются достигнуть взаимовыгодного исхода сделки, торгуясь за товары, представляющие для них ценность.

Предполагается, что стоимость товаров ни при каких условиях не может быть отрицательной и генерируется случайным образом с использованием определенных ограничений. Во-первых, общая ценность всех товаров равна 10. Во-вторых, каждый товар имеет ненулевую ценность по крайней мере для одного пользователя, чтобы не было смысла вообще игнорировать данный товар.

Наконец, некоторые товары имеют ненулевую ценность для обоих пользователей. Эти ограничения исключают возможность того, что оба пользователя наберут максимальное количество очков, равное 10, что гарантирует соревновательный характер процесса торговли. После 10 кругов переговоров агентам предоставляется возможность завершить переговоры, не достигнув сделки, что приносит каждому из пользователей ноль очков. Использовались три типа товаров: книги, шляпы и воздушные шарик, причем общее количество товаров в коллекции колебалось в пределах от 5 до 7 единиц. Тот факт, что ценность товаров была разной для обоих пользователей (и при этом ни одному из них не было известно, какую ценность тот или иной товар представляет для другого пользователя), играет очень важную роль. Если бы оба пользователя были талантливыми переговорщиками, то они смогли бы добиться для себя товаров общей стоимостью более 10. Тем не менее лучший переговорщик сможет овладеть товарами большей стоимости, оптимально торгуясь за товары, представляющие ценность для обоих.

Функцией вознаграждения при такой постановке задачи обучения с подкреплением является окончательная стоимость товаров, которой добился пользователь. На предварительном этапе можно использовать обучение с учителем с целью максимизации правдоподобности высказываний. Непосредственное применение рекуррентных сетей для максимизации правдоподобности высказываний приводила к тому, что агенты становились слишком уступчивыми. Поэтому был предпринят подход, в котором обучение с подкреплением сочеталось с обучением с учителем. Включение этапа обучения с учителем в обучение с подкреплением способствует тому, чтобы модели не отклонялись от стиля речи, свойственного людям. Была введена форма планирования, которую называют *развертыванием* диалогов. В этом подходе применяется рекуррентная архитектура “кодировщик — декодировщик”, в которой декодировщик максимизирует функцию вознаграждения, а не вероятность высказываний. В основе такой архитектуры “кодировщик — декодировщик” лежит обучение “последовательность в последовательность”, о чем шла речь в разделе 7.7.2.

В целях обучения с учителем использовались диалоги, собранные на интернет-площадке *Amazon Mechanical Turk*. Всего было собрано 5808 диалогов в 2236 уникальных сценариях, где сценарий определяется назначением определенного набора стоимостей товаров. Из этих примеров было отобрано 252 сценария, соответствующих 526 диалогам. Каждый сценарий приводил к двум тренировочным примерам, соответствующим точкам зрения каждого из агентов. Конкретным тренировочным примером мог быть сценарий, в котором распределяемыми между агентами товарами были 3 книги, 2 шляпы и 1 воздушный шарик. Эти данные являются частью входа для каждого из агентов. Вторым входом могла быть стоимость (ценность) каждого товара для агентов: 1) для

агента А — книга:1, шляпа:3, воздушный шарик:1; и 2) для агента В — книга:2, шляпа:1, воздушный шарик:2. Обратите внимание на то, что в этих условиях агент А, соблюдая скрытность своих намерений в ходе переговоров, должен попытаться получить как можно больше шляп, тогда как агент В должен сфокусироваться на книгах и воздушных шариках. Приведенный ниже пример диалога, содержащегося в тренировочных данных, взят из [290].

Агент А: Я хочу книги и шляпы, ты получаешь воздушный шарик.

Агент В: Дай мне в придачу книгу, и мы договорились.

Агент А: Согласен.

Агент В: <выбирает>.

Конечным выходом для агента А являются 2 книги и 2 шляпы, тогда как конечным выходом для агента В является 1 книга и 1 воздушный шарик. Таким образом, каждый агент имеет собственный набор входов и выходов, и диалоги также рассматриваются с позиций каждого из них в отношении того, какая часть диалога им принимается, а какая — высказывается. Поэтому каждый сценарий генерирует два тренировочных примера, а для генерации записей и окончательного выхода каждого агента используется одна и та же разделяемая рекуррентная сеть. Диалог x — это список токенов (лексем) $x_0 \dots x_T$, содержащих предложения каждого агента, которые перемежаются символами, обозначающими, было ли принято данное предложение агентом их партнера. Специальный токен в конце указывает на то, что один из агентов обозначил достижение соглашения.

Процедура обучения с учителем использует четыре различных вентиляльных рекуррентных блока (GRU). Первый GRU_g кодирует входные цели, второй GRU_q генерирует условия диалога, а блоки GRU_o и $GRU_{\bar{o}}$ — это вентиляльные рекуррентные блоки для выходов в прямом и обратном направлениях соответственно. По сути, выход создается двунаправленным GRU. Все эти блоки подключаются сквозным образом. В модели обучения с учителем параметры обучаются с использованием входов, диалогов и выходов, доступных в тренировочных данных. Функция потерь обучения с учителем представляет собой взвешенную сумму функции потерь диалога и функции потерь выходного предсказания выбора товара.

Однако в случае обучения с подкреплением используется развертывание диалогов. Обратите внимание на то, что группа GRU в модели обучения с учителем, по сути, предоставляет вероятностные выходы. Поэтому ту же модель можно адаптировать для обучения с подкреплением, просто изменив функцию потерь. Иными словами, комбинацию GRU можно рассматривать как некий тип сети обучения стратегий. Эту сеть стратегий можно использовать для генерации развертывания методом Монте-Карло различных диалогов и их окончательных вознаграждений. Каждое из семплируемых действий становится

частью тренировочных данных, и это действие ассоциируется с соответствующим окончательным вознаграждением такого развертывания. То есть в данном подходе агент обучается лучшей стратегией, торгуясь сам с собой. Указанное окончательное вознаграждение используется для обновления параметров сети стратегий. Эти вознаграждения вычисляются на основе стоимости товаров, согласованных по завершении диалога. Описанный подход можно рассматривать как пример реализации алгоритма REINFORCE [533]. Одной из проблем самообучения является тенденция агентов к обучению собственному языку, который отличается от обычной человеческой речи, если обе стороны используют обучение с подкреплением. Поэтому один из агентов ограничивается моделью, основанной на обучении с учителем.

Что касается выработки окончательного предсказания, то одна из возможностей заключается в семплировании из вероятностного выхода GRU. Однако при работе с рекуррентными сетями такой подход часто оказывается неоптимальным. В связи с этим применяется двухстадийный подход. Прежде всего, с помощью семплирования создаются высказывания-кандидаты. Затем вычисляются ожидаемые вознаграждения, соответствующие каждому высказыванию-кандидату, и выбирается тот из них, для которого вознаграждение имеет максимальное значение. Для вычисления ожидаемого вознаграждения выход масштабировался в соответствии с вероятностью диалога, поскольку отбор диалога с низкой вероятностью любим из агентов весьма маловероятен.

В [290] был сделан ряд важных наблюдений относительно производительности такого подхода. Во-первых, методы обучения с учителем часто приводят к тому, что агент легко сдается, тогда как методы обучения с подкреплением заставляют агентов быть более настойчивыми в попытках добиться для себя лучшей сделки. Во-вторых, методы обучения с подкреплением часто вырабатывают тактику ведения переговоров, свойственную людям. В отдельных случаях, чтобы добиться лучших для себя условий относительно другого товара, агенту удавалось проявлять поддельную заинтересованность в товаре, который на самом деле представлял для него не очень большую ценность.

9.7.4. Беспилотные автомобили

Как и в случае задачи локомоции робота, автомобиль вознаграждается за продвижение из точки А в точку В без аварий и других нежелательных дорожных происшествий. Автомобиль оборудуется видео- и аудиоаппаратурой, датчиками сближения с посторонними предметами и датчиками движения для записи наблюдений. Задачей системы обучения с подкреплением является безопасное передвижение автомобиля из точки А в точку В независимо от дорожных условий.

Управление автомобилем — задача, в которой трудно определить правила, регламентирующие действия в любой возможной ситуации. С другой стороны,

судить о том, что управление автомобилем осуществляется правильно, относительно легко. Именно такая постановка задачи хорошо подходит для обучения с подкреплением. И хотя полностью автономный автомобиль должен иметь широкий ряд компонент, соответствующих входам и датчикам различного типа, мы ограничимся упрощенными условиями, когда используется единственная камера [46, 47]. Изучение этой системы будет полезно, поскольку позволит продемонстрировать, что даже с одной фронтальной камерой в сочетании с обучением с подкреплением можно добиться очень многого. Интересно отметить, что идея была подсказана опубликованной в 1989 году работой, содержащей описание системы *Autonomous Land Vehicle in a Neural Network (ALVINN)* [381], и основным отличием от указанной работы, выполненной 25 годами ранее, было использование более мощных наборов данных и вычислительных ресурсов. Таким образом, можно увидеть, насколько важную роль играют данные и вычислительные ресурсы в построении современных систем обучения с подкреплением.

Сбор тренировочных данных осуществлялся посредством вождения автомобиля по разным дорогам при различных условиях. Преимущественно это были дороги в центральной части штата Нью-Джерси, однако также совершались поездки по магистралям штатов Иллинойс, Мичиган, Пенсильвания и Нью-Йорк. И хотя основным источником данных служила фронтальная камера, установленная на месте водителя, во время фазы тренировки использовались две дополнительные камеры, которые были установлены в двух других позициях в передней части автомобиля и предназначались для сбора повернутых и смещенных изображений. Эти вспомогательные камеры не применялись для принятия окончательных решений, однако были полезны для сбора дополнительных данных. Размещение дополнительных камер гарантировало, что с их помощью получались повернутые и сдвинутые изображения, поэтому их можно было использовать для того, чтобы обучить сеть распознавать случаи, когда безопасность автомобиля подвергалась риску. Вообще говоря, эти камеры использовались в целях аугментации данных. Нейронная сеть обучалась минимизации расхождения между выходной командой управления сети и командой управления водителем-человека. Заметьте, что это делает данный подход более близким к обучению с учителем, чем к обучению с подкреплением. Методы обучения такого типа также называют *имитационным обучением* (imitation learning) [427]. Имитационное обучение часто задействуют в качестве первого шага для облегчения “холодного запуска”, свойственного системам обучения с подкреплением.

Сценарии, включающие имитационное обучение, часто напоминают сценарии, включающие обучение с подкреплением. В таком сценарии можно относительно легко использовать обучение с подкреплением, предоставляя вознаграждение в тех случаях, когда автомобиль движется без вмешательства человека.

С другой стороны, если автомобиль не движется в нужном направлении или требуется вмешательство человека, назначается штраф. Однако это не совсем соответствует тому, как должна обучаться беспилотная система [46, 47]. Одной из проблем, свойственных задачам наподобие беспилотных автомобилей, является то, что в процессе тренировки всегда необходимо учесть вопросы, связанные с безопасностью. И хотя для большинства доступных беспилотных автомобилей подробные описания предоставлены лишь в ограниченном объеме, все же можно сделать вывод о том, что в этих условиях обучению с учителем отдавалось большее предпочтение по сравнению с обучением с подкреплением. Тем не менее в рамках более широкой архитектуры нейронной сети, которая была бы полезной в данном случае, различия между использованием обучения с учителем и обучения с подкреплением являются незначительными. Общее обсуждение обучения с подкреплением в контексте беспилотных автомобилей содержится в [612].

Архитектура сверточной нейронной сети представлена на рис. 9.10. Она включает 9 слоев, в том числе слой нормализации, 5 сверточных слоев и 3 полносвязных слоя. В первом сверточном слое используется фильтр 5×5 с шагом 2. В каждом из следующих двух слоев используется бесшаговая свертка с фильтром 3×3 . За этими сверточными слоями следуют три полносвязных слоя. Конечным выходом являлось управляющее значение, соответствующее обратной величине радиуса поворота. Сеть имела 27 миллионов связей и 250 000 параметров. Подробное описание того, каким образом глубокая нейронная сеть управляет автомобилем, приведено в [47].

Обученный автомобиль испытывался как в режиме имитации, так и в условиях реального дорожного движения. В дорожных испытаниях всегда принимал участие человек, который в случае необходимости мог взять управление автомобилем на себя. Исходя из этого, в качестве меры вычислялась процентная доля времени, в течение которого требовалось вмешательство человека. Было установлено, что автомобиль был способен двигаться автономно в течение 98% времени. Видеоролик, демонстрирующий этот тип автономного вождения автомобиля, доступен по ссылке [611]. Благодаря визуализации карт активации обученной сверточной нейронной сети (что выполнялось с использованием методологии, описанной в главе 8) был сделан ряд интересных наблюдений. В частности, было замечено значительное смещение признаков в направлении тех аспектов обучения, которые имели существенное значение для управления автомобилем. В случае дорог без покрытия карты активации признаков обнаружили изменения профиля дороги. С другой стороны, если автомобиль находился в лесу, то карты активации признаков переполнялись шумом. В случае сверточной нейронной сети, обученной на изображениях ImageNet, такого не происходило, поскольку карты активации признаков в типичных случаях

содержали полезные характеристики деревьев, листьев и т.п. Отличия этих двух случаев обусловлены тем, что при беспилотном вождении автомобиля сверточная сеть тренируется для определенной цели и обучается выделению признаков, имеющих значение для управления транспортным средством, к которому специфические характеристики деревьев в лесу не имеют никакого отношения.

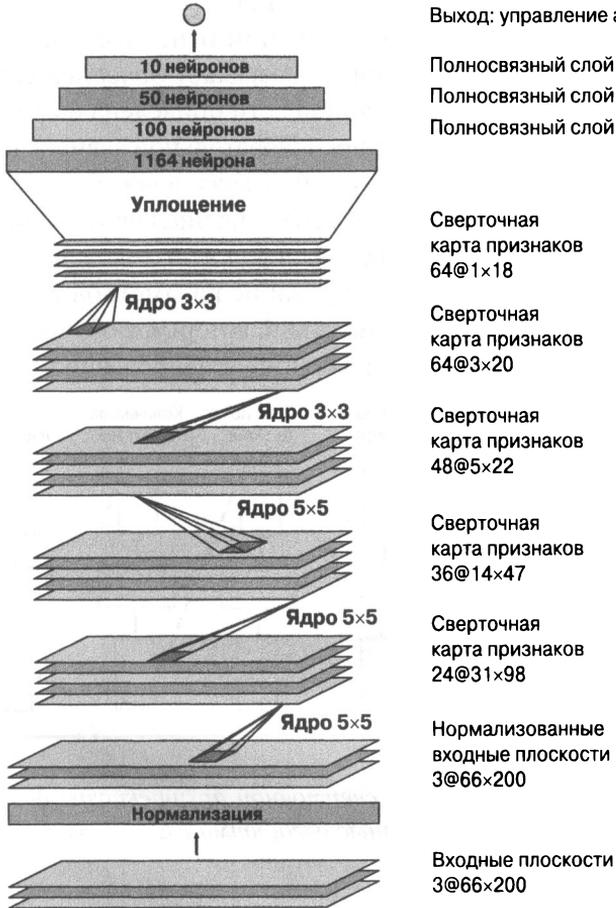


Рис. 9.10. Архитектура нейронной сети системы управления в беспилотном автомобиле, которая обсуждалась в [46] (публикуется с разрешения компании NVIDIA)

9.7.5. Предложение нейронных архитектур с помощью обучения с подкреплением

Интересным применением обучения с подкреплением является обучение архитектуры нейронной сети для выполнения специфических задач. В качестве примера обсудим случай, когда мы хотим определить структуру сверточной

нейронной сети, предназначенной для классификации такого набора данных, как CIFAR-10 [583]. Очевидно, что структура нейронной сети зависит от количества гиперпараметров, таких как количество фильтров, высота фильтра, ширина фильтра, шаг по высоте и шаг по ширине. Эти параметры взаимосвязаны, и параметры поздних слоев зависят от параметров ранних.

Метод обучения с подкреплением использует рекуррентную сеть в качестве *контроллера*, управляющего принятием решений относительно параметров сверточной нейронной сети, которую также называют *дочерней сетью* (child network) [569]. Общая архитектура нейронной сети приведена на рис. 9.11. Выбор рекуррентной сети обуславливается последовательной зависимостью между различными архитектурными параметрами. Классификатор Softmax служит для предсказания каждого выхода в виде токена, а не числового значения. Затем этот токен используется в качестве входа для следующего слоя, что обозначено на рис. 9.11 штриховыми линиями. Генерирование параметров в виде токенов приводит к дискретным пространствам действий, которые в обучении с подкреплением обычно встречаются чаще по сравнению с непрерывными.

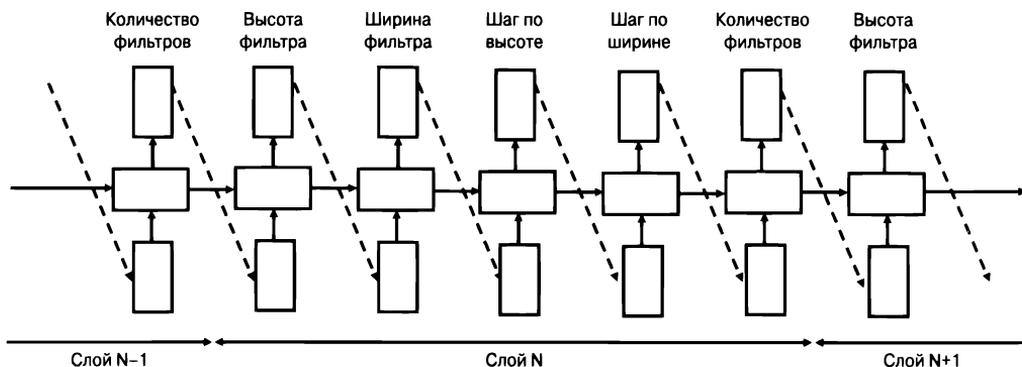


Рис. 9.11. Сеть-контроллер для обучения сверточной архитектуры дочерней сети [569]. Сеть-контроллер обучается с помощью алгоритма REINFORCE

Для генерирования сигнала вознаграждения дочерняя сеть запускалась на валидационном наборе изображений CIFAR-10. Отметим, что для тестирования точности дочерней сети она должна тестироваться на наборе данных CIFAR-10. Поэтому данный процесс требует выполнения полной процедуры тренировки дочерней сети, что обходится довольно дорого. Для тренировки параметров сети-контроллера используется сигнал вознаграждения в сочетании с алгоритмом REINFORCE. Поэтому в данном случае сеть-контроллер в действительности представляет собой *сеть обучения стратегий* (policy network), которая генерирует последовательность взаимосвязанных параметров.

Ключевую роль играет количество слоев дочерней сети (которое также определяет количество слоев рекуррентной сети). Значение этого параметра не

остаётся постоянным и следует определенному графику изменения в процессе тренировки. На ранних итерациях используется меньшее количество слоев, поэтому обучаемая архитектура сверточной нейронной сети является мелкой. По мере обучения сети количество слоев постепенно увеличивается с течением времени. Метод градиентного спуска по стратегиям не слишком отличается от того, который обсуждался в данной главе, за исключением того, что рекуррентная сеть тренируется с использованием сигнала вознаграждения, а не сети прямого распространения. В [569] также обсуждаются различные типы оптимизации, такие как эффективные реализации параллелизма и использование передовых архитектур наподобие архитектуры с *короткозамкнутыми соединениями* (skip connections).

9.8. Практические вопросы безопасности

Упрощение проектирования крайне сложных алгоритмов обучения с помощью обучения с подкреплением иногда может иметь неожиданные последствия. Системы обучения с подкреплением обладают гораздо более широким набором степеней свободы по сравнению с другими системами обучения, и это может порождать риски, связанные с обеспечением безопасности. В то время как биологическая жадность является важным фактором человеческого интеллекта, она также служит источником многих нежелательных аспектов человеческого поведения. Простота, являющаяся величайшим достижением обучения, управляемого поощрением, одновременно является опаснейшей ловушкой для биологических систем. Поэтому имитация таких систем сталкивается с аналогичными ловушками с точки зрения искусственного интеллекта. Например, плохо продуманная система вознаграждений может привести к непредвиденным последствиям, обусловленным исследовательской природой процесса, посредством которого система обучается выполнению действий. Системы обучения с подкреплением часто способны обучаться неизвестным “уловкам” и “хитростям”, скрытым в плохо спроектированных видеоиграх, которые мы должны рассматривать как предупреждения о том, что может случиться в далеко не совершенном реальном мире. Робот, если ему удастся обмануть своими действиями человека или функцию оценки, может научиться тому, что для получения вознаграждения ему достаточно лишь сделать вид, что он закручивает колпачок на бутылке. Иными словами, проектирование функции оценки иногда может оказаться вовсе не таким простым делом, как кажется на первый взгляд.

Кроме того, система может попытаться получить вознаграждение “неэтичным способом”. Например, робот-уборщик может попытаться заработать вознаграждение, сначала создав беспорядок, а затем устранив его [10]. Еще более мрачные сценарии можно представить для роботов-медсестер. Что интересно, подобные типы поведения иногда проявляют и люди. Такое нежелательное

сходство является непосредственным результатом упрощения процесса обучения машин за счет использования простых принципов, базирующихся на жажде поощрения, свойственной живым организмам. Стремление к простоте результатов приводит к передаче больших полномочий машине, что может иметь непредвиденные последствия. В некоторых случаях даже при проектировании функции вознаграждения возникают проблемы этического характера. Например, если становится очевидным, что избежать ДТП не удастся, то кого должен спасать беспилотный автомобиль: своего водителя или двух пешеходов? Большинство людей, действуя рефлекторно в силу биологических инстинктов, в подобных ситуациях попытаются спасти себя, но захотим ли мы ориентировать на это обучающую систему? В то же время убедить водителя-оператора в том, что он должен доверять транспортному средству, несмотря на то что его личная безопасность не является высшим приоритетом для обучающей системы, очень трудно. Кроме того, системы обучения с подкреплением чувствительны к тому, как люди-операторы взаимодействуют с ними и манипулируют назначением вознаграждений. Были случаи, когда чат-бота научили отпускать замечания обидного или расистского характера.

Обучающим системам приходится очень нелегко при обобщении их опыта на незнакомые ситуации. Эта проблема носит название *сдвиг распределения* (distributional shift). Так, беспилотный автомобиль, который обучался в одной стране, может быть бесполезным в другой стране. Точно так же в случае обучения с подкреплением разведочные действия иногда могут становиться опасными. Представьте робота, пытающегося припаять провод в электронном устройстве, напичканном хрупкими электронными компонентами. В этих условиях разведочные действия чреваты неприятностями. Все это указывает на то, что невозможно создавать системы искусственного интеллекта (ИИ), не уделяя должного внимания вопросам безопасности. Некоторые организации, такие как *OpenAI* [613], стали лидерами в вопросах обеспечения безопасности ИИ. Проблемы подобного рода обсуждаются также в [10] в рамках более широкого спектра возможных решений. По всей видимости, во многих случаях обеспечение безопасности решений будет требовать определенного участия человека в процессе обучения систем [424].

9.9. Резюме

Эта глава посвящена обучению с подкреплением, в котором агенты обучаются оптимальным действиям, взаимодействуя с окружением и получая определенное вознаграждение (или наказание) за свои действия. Существует несколько классов методов обучения с подкреплением, из которых наиболее популярны метод Q-обучения и методы, основанные на обучении стратегиям игры. В последние годы методы, обучающиеся стратегиям, приобретают все большую

популярность. Многие из них реализуются в виде сквозных систем, интегрирующих нейронные сети глубокого обучения для получения сенсорных входных данных и обучения стратегиям, оптимизирующим вознаграждение. Алгоритмы обучения с подкреплением применяются во многих задачах, таких как видеоигры, робототехника и беспилотные автомобили. Способность этих алгоритмов обучаться на основе проведения экспериментов приводит к инновационным решениям, которые невозможно реализовать с помощью других форм обучения. Вместе с тем алгоритмы обучения с подкреплением порождают уникальные проблемы безопасности, обусловленные чрезмерным упрощением процесса обучения вследствие использования функций оценки вознаграждения.

9.10. Библиографическая справка

Великолепный обзор методов обучения с подкреплением содержится в книге Саттона и Барто [483]. Ряд других обзоров доступен по ссылке [293]. Лекции Дейвида Сильвера по обучению с подкреплением доступны на YouTube [619]. Метод временных разностей был предложен Сэмюэлем в контексте программы для игры в шашки [421] и формализован Саттоном [482]. Метод Q-обучения был предложен Уоткинсом [519], а доказательство его сходимости было приведено в [520]. Алгоритм SARSA был введен в [412]. Ранние методы использования нейронных сетей в обучении с подкреплением были предложены в [296, 349, 492–494]. Нейронная сеть TD-Gammon, лежащая в основе программы для игры в нарды, была предложена в [492].

Система, в которой для создания алгоритма глубокого Q-обучения, работающего с пикселями, использовалась сверточная нейронная сеть, была предложена в пионерских работах [335, 336]. В [335] было высказано предположение о том, что представленный подход можно улучшить, воспользовавшись другими хорошо известными идеями, такими как *приоритетная очистка* (prioritized sweeping) [343]. Асинхронные методы, использующие несколько агентов для выполнения обучения, обсуждаются в [337]. Применение нескольких асинхронных потоков выполнения позволяет избежать проблем корреляции внутри потока, что улучшает сходимость к высококачественным решениям. Асинхронный подход такого типа часто используют вместо методики *воспроизведения опыта* (experience replay). В той же работе была предложена *n*-шаговая методика, в которой для предсказания Q-значений используется просмотр на *n* шагов вперед (вместо 1 шага).

Одним из недостатков Q-обучения является то, что при определенных обстоятельствах оно переоценивает значения действий. Улучшенный вариант Q-обучения, известный как *двойное Q-обучение* (double Q-learning), был предложен в [174]. В оригинальной форме Q-обучения одни и те же значения используются как для выбора, так и для оценки действий. В случае двойного Q-обучения эти

значения разделяются, поэтому для выбора и оценки действий применяются разные значения. Как правило, такое изменение снижает чувствительность данного подхода к проблеме переоценки. Использование приоритетного воспроизведения опыта для улучшения производительности алгоритмов обучения с подкреплением в условиях разреженных данных обсуждается в [428]. Такой подход значительно улучшает производительность системы на играх Atari.

В последние годы популярность градиентного спуска по стратегиям превысила популярность методов Q-обучения. Интересное упрощенное описание этого подхода применительно к игре *Pong* на платформе Atari приведено в [605]. Ранние подходы, основанные на использовании методов конечных разностей в градиентном спуске по стратегиям, обсуждаются в [142, 355]. Методы относительного правдоподобия для градиентного спуска по стратегиям были впервые применены в алгоритме REINFORCE [533]. Ряд аналитических результатов, касающихся этих алгоритмов, представлен в [484]. Градиентный спуск по стратегиям был использован в целях обучения в программе для игры го [445], хотя в общем подходе сочетается целый ряд различных элементов. Естественный градиентный спуск по стратегиям был предложен в [230]. Было продемонстрировано, что один из подобных методов [432] хорошо работает при обучении движениям роботов. Использование *обобщенной оценки выгоды* (generalized advantage estimation — GAE) с непрерывным вознаграждением обсуждается в [433]. Описанный в [432, 433] подход основан на оптимизации с помощью естественного градиентного спуска по стратегиям и называется *оптимизацией стратегии в пределах доверительной области* (trust region policy optimization — TRPO). Основная идея заключается в том, что в обучении с подкреплением плохие шаги должны наказываться более строго (чем в обучении с учителем), поскольку они ухудшают качество собираемых данных. Поэтому для метода TRPO более предпочтительны методы второго порядка с сопряженными градиентами (см. главу 3), в которых обновления не выходят за пределы доверительных областей. Также доступны обзоры, посвященные специфическим типам методов обучения с подкреплением, таким как алгоритм “актор — критик” [162].

Поиск по дереву методом Монте-Карло был предложен в [246]. Впоследствии он был применен в игре го [135, 346, 445, 446]. Обзор этих методов можно прочитать в [52]. Более поздние версии программы AlphaGo обходились без компонент обучения с учителем, адаптированных к шахматам и сего, и работали лучше в отсутствие начальных знаний [446, 447]. В подходе AlphaGo сочетается несколько идей, включая использование сетей обучения стратегий, дерево поиска методом Монте-Карло и сверточные нейронные сети. Применение сверточных нейронных сетей для игры в го исследовалось в [73, 307, 481]. Многие из этих методов задействуют обучение с учителем, чтобы имитировать

стиль игры профессионалов го. Также были исследованы некоторые методы TD-обучения для игры в шахматы, такие как *NeuroChess* [496], *KnightCap* [22] и *Giraffe* [259], однако они оказались менее успешными, чем традиционные движки. Сочетание сверточных нейронных сетей и обучения с подкреплением для пространственных игр — это новый (и успешный) рецепт, отличающий алгоритм *Alpha Zero* от этих методов. Ряд методов, предназначенных для тренировки самообучающихся роботов, представлен в [286, 432, 433]. Обзор методов глубокого обучения с подкреплением для генерации диалогов приведен в [291]. Разговорные модели, использующие лишь обучение с учителем с помощью рекуррентных сетей, обсуждаются в [440, 508]. Торговый чат-бот, упоминавшийся в этой главе, описан в [290]. Описание беспилотных автомобилей основано на [46, 47]. Курс по беспилотным автомобилям, который читается в Массачусетском технологическом институте, доступен по ссылке [612]. Обучение с подкреплением также применяется для генерирования структурированных запросов на естественном языке [563] и для обучения нейронных архитектур в различных задачах [19, 569].

Обучение с подкреплением можно использовать для улучшения моделей глубокого обучения. Это достигается с помощью механизма *внимания* [338, 540], в котором обучение с подкреплением позволяет сфокусировать внимание на избранных фрагментах данных. Идея заключается в том, что значительная часть данных зачастую несущественна для обучения, и обучение тому, как фокусировать внимание на избранных фрагментах, может значительно улучшить результаты. Отбор релевантных фрагментов осуществляется посредством обучения с подкреплением. Механизмы внимания обсуждаются в разделе 10.2. В этом смысле обучение с подкреплением является одной из тем машинного обучения, которая связана с глубоким обучением гораздо теснее, чем кажется на первый взгляд.

9.10.1. Программные ресурсы и испытательные системы

Несмотря на значительный прогресс в проектировании алгоритмов обучения с подкреплением, достигнутый в последние годы, ассортимент коммерческих программ, использующих эти методы, все еще довольно ограничен. Тем не менее доступен ряд испытательных программных платформ, которые можно применять для тестирования различных алгоритмов. Возможно, наилучшим источником высококачественных базовых средств обучения с подкреплением является библиотека *OpenAI* [623]. Реализации алгоритмов обучения с подкреплением также доступны в библиотеках *TensorFlow* [624] и *Keras* [625].

Большинство фреймворков для тестирования и разработки алгоритмов обучения с подкреплением основываются на специфических типах сценариев

обучения. Некоторые фреймворки облегчены и могут служить для быстрого тестирования. Например, библиотека *ELF* [498], созданная компанией Facebook, предназначена для стратегических игр в режиме реального времени и представляет собой облегченный фреймворк глубокого обучения с открытым исходным кодом. Библиотека *OpenAI Gym* [620] предоставляет среды разработки алгоритмов обучения с подкреплением для игр на платформе Atari и имитированных роботов. Библиотека *OpenAI Universe* [621] может использоваться для адаптации программ обучения с подкреплением к средам Gym. Например, в эту среду добавлены имитации беспилотных автомобилей. Обучающая среда *Arcade*, предназначенная для разработки агентов в контексте игр Atari, описана в [25]. Имитатор *MuJoCo* [609] (от англ. *Multi-Joint dynamics with Contact* — многозвенная динамика с контактом) — это физический движок, предназначенный для имитации роботов. *ParlAI* [622] — фреймворк с открытым исходным кодом, запущенный компанией Facebook. Он предназначен для исследования диалоговых систем и реализован на языке Python. Компания Vaidu создала платформу с открытым исходным кодом для собственного проекта беспилотных автомобилей, который называется *Apollo* [626].

9.11. Упражнения

1. Справедливость уравнения 9.24, на котором основан трюк, используемый в методе относительного правдоподобия, была доказана в этой главе для случая дискретных действий a . Обобщите этот результат на случай действий, характеризующихся непрерывными значениями.
2. На протяжении всей главы для реализации градиентного спуска по стратегиям использовалась нейронная сеть, обучающая стратегию игры (сеть обучения стратегии). Опишите важность выбора архитектуры этой сети в условиях различных задач.
3. Имеются два игровых автомата, каждый из которых оборудован 100 световыми индикаторами. Распределение вероятных размеров вознаграждения, которое можно получить от каждого автомата, является неизвестной (и, возможно, зависящей от автомата) функцией текущей конфигурации включенных индикаторов. В процессе игры конфигурация включенных индикаторов изменяется закономерным образом, однако эта закономерность неизвестна. Объясните, почему эта задача более трудная по сравнению с задачей многорукого бандита. Предложите решение на основе глубокого обучения, которое обеспечило бы при каждой попытке оптимальный выбор автомата, максимизирующего среднее вознаграждение в пересчете на одну попытку в установившемся состоянии.

4. Рассмотрим популярную игру “камень, ножницы, бумага”. Люди часто пытаются предугадать следующий ход, исходя из предыстории ходов. Какой подход вы использовали бы для обучения этой игре: метод Q-обучения или метод на основе обучения стратегии игры? Почему? Рассмотрим ситуацию, когда игрок-человек выбирает один из трех ходов с вероятностью, которая является неизвестной функцией предыстории ходов, включающей по 10 предыдущих ходов каждого игрока. Предложите метод глубокого обучения, предназначенный для игры с таким противником. Обеспечит ли хорошо продуманный метод глубокого обучения достижение преимущества над игроком-человеком? Какой стратегии должен придерживаться игрок-человек, чтобы обеспечить вероятностный паритет с противником в виде модели глубокого обучения?
5. Рассмотрим игру крестики-нолики, по завершении которой вознаграждением является одно из значений $\{-1, 0, +1\}$. Допустим, вы обучили значения всех состояний (в предположении, что каждая из сторон играет оптимальным образом). Объясните, почему состояния, которые соответствуют позициям, не являющимся конечными, будут иметь ненулевые значения. Что вы можете сказать о связи оценок ценности промежуточных шагов с вознаграждением, получаемым в конце игры?
6. Напишите реализацию Q-обучения, которая обучает значение каждой пары “состояние — действие” для игры крестики-нолики, многократно играя против человека. Никакие аппроксиматоры функций не используются, поэтому вся таблица пар “состояние — действие” обучается с помощью уравнения 9.5. Исходите из предположения, что каждое из Q-значений в этой таблице может быть инициализировано нулевым значением.
7. Двухшаговая TD-ошибка определяется следующим образом:

$$\delta_t^{(2)} = r_t + \gamma r_{t+1} + \gamma^2 V(s_{t+2}) - V(s_t).$$

- А. Предложите алгоритм TD-обучения для 2-шагового случая.
- Б. Предложите привязанный к стратегии n -шаговый алгоритм наподобие SARSA. Покажите, что в этом случае обновление будет представлять собой усеченный вариант уравнения 9.16, в котором $\lambda = 1$. Что вы можете сказать относительно случая, когда $n = \infty$?
- В. Предложите не привязанный к стратегии n -шаговый алгоритм обучения наподобие Q-обучения и опишите его преимущества и недостатки по сравнению с алгоритмом Б.

Глава 10

Дополнительные вопросы глубокого обучения

Почему бы нам вместо того, чтобы пытаться создать программу, имитирующую ум взрослого человека, не попытаться создать программу, которая имитировала бы ум ребенка? Ведь если ум ребенка получает соответствующее воспитание, то он становится умом взрослого человека.

Алан Тьюринг “Вычислительные машины и разум”

10.1. Введение

В этой главе мы рассмотрим ряд дополнительных вопросов глубокого обучения, которые либо не вписываются естественным образом в тематику предыдущих глав, либо их уровень сложности требует отдельного описания. В частности, мы обсудим следующие темы.

1. *Модели внимания.* Люди не используют активно сразу всю информацию, поступающую им от окружения в каждый момент времени. Вместо этого они фокусируются на определенных фрагментах данных, имеющих отношение к текущей задаче. В биологии это называется *вниманием*. Аналогичный принцип может быть использован и в приложениях искусственного интеллекта. Модели с вниманием применяют обучение с подкреплением (или другие методы) для фокусирования на небольших порциях данных, имеющих отношение к задаче, решаемой в данный момент. В последнее время такие методы привлекаются для улучшения производительности.
2. *Модели с селективным доступом к внутренней памяти.* Тесно связаны с моделями внимания, хотя между ними имеется и различие, которое заключается в том, что модели внимания фокусируются главным образом на

специфических частях сохраненных данных. Полезной аналогией может служить то, как человек пользуется своей памятью при решении конкретных задач. Люди обладают огромным репозиторием данных, хранящихся в клетках головного мозга. Однако в любой момент времени осуществляется доступ лишь к небольшой части этих данных, имеющих отношение к текущей задаче. Точно так же современные компьютеры располагают значительными объемами памяти, но доступ компьютерных программ к этой памяти осуществляется управляемым селективным способом посредством переменных, реализующих механизмы косвенной адресации. Все нейронные сети обладают памятью в виде скрытых состояний. Однако эта память настолько тесно связана с вычислениями, что отделить доступ к данным от вычислений очень трудно. За счет повышения селективности операций чтения и записи во внутреннюю память нейронной сети и явного введения механизмов адресации можно добиться того, чтобы выполняемые сетью вычисления более тесно отражали стиль программирования, свойственный людям. По сравнению с более традиционными нейронными сетями такие сети часто обладают большей обобщающей способностью при выполнении предсказаний для данных, не входящих в тренировочный набор. Селективное обращение к памяти можно рассматривать как одну из форм применения механизма внимания к *внутренней* памяти нейронной сети. Результирующую архитектуру называют *сетью с памятью* (memory network) или *нейронной машиной Тьюринга* (neural Turing machine).

3. *Генеративно-состязательные сети*. Предназначены для создания моделей, порождающих данные на основе предоставленных примеров. Сети этого типа могут создавать реалистично выглядящие образцы на основе данных, используя две конкурирующие сети. Одна из сетей (генератор) формирует синтетические образцы, тогда как другая (дискриминатор) классифицирует смешанный набор оригинальных примеров и сгенерированных образцов, относя их либо к реальному, либо к синтетическому классу. В результате такой состязательной игры генератор постепенно улучшается до тех пор, пока дискриминатор уже не сможет отличать поддельные образцы от реальных. Кроме того, учет специфики конкретного контекста (например, аннотирование изображений) позволяет управлять созданием желаемых образцов определенного типа.

Моделям внимания часто приходится принимать нелегкие решения относительно того, за какими частями данных необходимо следить. Эту проблему можно считать аналогичной проблеме выбора, с которой приходится сталкиваться в алгоритме обучения с подкреплением. Некоторые из методов, применяемых для создания моделей на основе внимания, в значительной мере привязаны к обучению с подкреплением, в то время как для других методов это не

так. Поэтому, прежде чем читать данную главу, настоятельно рекомендуется изучить материал, изложенный в главе 9.

С нейронными машинами Тьюринга тесно связан класс архитектур, получивший название *сети с памятью*. В последнее время эти сети хорошо зарекомендовали себя при создании систем “вопрос — ответ”, хотя результаты все еще оставляют желать лучшего. Структуру нейронной машины Тьюринга можно считать шлюзом ко многим возможностям искусственного интеллекта, которые пока что не реализованы в полной мере. Как это вообще свойственно истории развития нейронных сетей, значительную роль в обеспечении реального прогресса в этом направлении сыграет увеличение объемов доступных данных и вычислительных мощностей.

Большая часть книги посвящена обсуждению сетей прямого распространения, функционирование которых основано на изменении весов под воздействием ошибок. *Состязательное обучение* (competitive learning) — это совершенно другой способ обучения, в котором нейроны соревнуются за право реагировать на поднабор входных данных. Веса модифицируются на основании того, кто именно стал победителем. Такой подход представляет собой разновидность обучения по правилам Хебба (см. главу 6), и его полезно применять в таких задачах обучения без учителя, как кластеризация, снижение размерности и сжатие данных. Эта парадигма также будет обсуждаться в данной главе.

Структура главы

В следующем разделе обсуждаются механизмы внимания, применяемые в глубоком обучении. Некоторые из этих методов тесно связаны с моделями глубокого обучения. Дополнение нейронных сетей внешней памятью рассматривается в разделе 10.3. Генеративно-состязательные сети описаны в разделе 10.4, а методы соревновательного обучения обсуждаются в разделе 10.5. Об ограничениях нейронных сетей речь пойдет в разделе 10.6. Резюме главы приведено в разделе 10.7.

10.2. Механизмы внимания

Для решения конкретных задач люди редко используют весь объем входных данных, поступающих от органов чувств. В качестве примера рассмотрим задачу нахождения адреса по номеру дома (находящегося на известной улице). Здесь важной составляющей задачи является идентификация номера, указанного на табличке или почтовом ящике дома. В процессе этого на сетчатку глаза поступает изображение более широкой сцены, однако человек редко фокусирует внимание на изображении в целом. Сетчатка содержит небольшую область, так называемое *желтое пятно*, или *макулу*, с центральной *ямкой сетчатки*, которая обладает чрезвычайно высоким разрешением по сравнению с

остальной частью глаза. Эта область характеризуется высокой концентрацией чувствительных к цвету *колбочек*, в то время как большая часть нецентральных участков характеризуется сравнительно низким разрешением и преобладанием нечувствительных к цвету *палочек*. Различные области глаза представлены на рис. 10.1. При чтении номера дома центральная ямка фиксируется на нем, и его изображение попадает на ту часть сетчатки, которая соответствует макуле (особенно центральной ямке). Несмотря на то что вы отдаете себе отчет о других объектах, находящихся вне центрального поля зрения, использовать эти изображения для выполнения каких-либо детализированных задач практически невозможно. Например, очень трудно читать буквы, проецируемые на периферийные участки сетчатки. Области ямки соответствует лишь малая часть всей поверхности сетчатки, и ее диаметр составляет всего лишь 1,5 мм. По сути, глаз передает в высоком разрешении менее 0,5% поверхности изображения, попадающего на всю сетчатку. Такой подход выгоден с биологической точки зрения, поскольку в высоком разрешении передается лишь тщательно отобранная часть изображения, благодаря чему снижается объем внутренней обработки, которая требуется для *решения текущей задачи*. Несмотря на то что строение глаза позволяет легко объяснить селективное фокусирование внимания на визуальных входах, подобного рода селективность не ограничивается лишь визуальными аспектами. Острота восприятия большинства других органов чувств человека, таких как слух или обоняние, тоже часто зависит от текущей ситуации. В соответствии с этим мы начнем обсуждение понятия внимания в контексте компьютерного зрения, а затем рассмотрим его применительно к другим областям, таким как обработка текста.

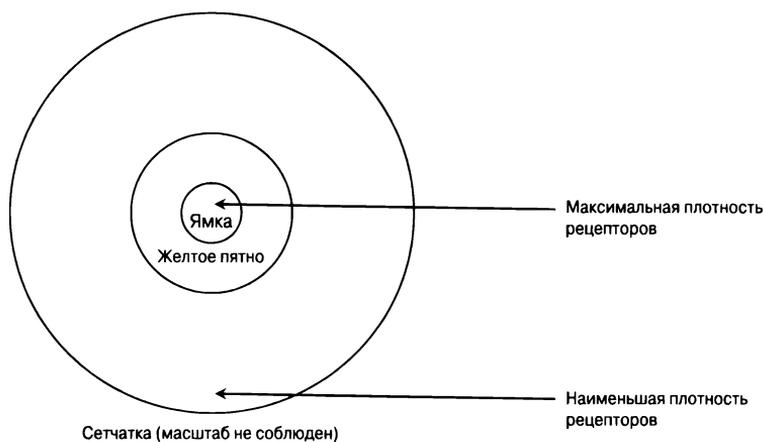


Рис. 10.1. Разрешающая способность различных участков глаза. Большая часть того, на чем фиксируется внимание глаза, воспринимается макулой

Интересный случай применения описанного подхода предоставляют изображения, захватываемые системой *Google Street View* (Просмотр улиц), которая была создана компанией Google для получения панорамных видов улиц в городах многих стран. Для такого типа изображений требуется связывание домов с их номерами. Несмотря на принципиальную возможность записи номеров домов в процессе получения снимков, эта информация должна быть отделена от самого изображения. Существует ли систематический способ идентификации номеров, соответствующих адресу, в случае наличия большого изображения фронтальной части дома? Ключевую роль в этом играет возможность фокусирования на небольшой части изображения для обнаружения искомого объекта. Основной трудностью является отсутствие заранее известного способа идентификации соответствующей части изображения с необходимой информацией. Следовательно, в данном случае требуется итеративный поиск определенных частей изображений с использованием информации, полученной на предыдущих итерациях. В подобных ситуациях полезно брать пример с биологических организмов, которые быстро находят визуальные подсказки среди того, на чем фокусируется их взгляд, чтобы определить, где следует продолжить поиск. Например, если первоначально наш взгляд случайно останавливается на дверной ручке, то из своего опыта мы знаем (об этом нам сообщают обученные нейроны), что, для того чтобы увидеть номер, необходимо перевести взгляд вверх влево или вправо. Итеративный процесс такого типа во многом напоминает методы обучения с подкреплением, обсуждавшиеся в предыдущей главе, в которых для обучения тому, что именно необходимо сделать (например, обнаружить номер дома), чтобы получить вознаграждение, итеративно применяются подсказки, полученные на предыдущих шагах. Как будет показано далее, многие приложения, обладающие механизмом внимания, сочетаются с обучением с подкреплением.

Понятие внимания также хорошо подходит для обработки естественного языка, когда искомая информация скрыта в длинных текстовых сегментах. Эта задача часто возникает в приложениях наподобие машинного перевода и системах “вопрос — ответ”, в которых рекуррентная нейронная сеть должна кодировать все предложение в виде вектора фиксированной длины (см. раздел 7.7.2). В результате нейронная сеть часто не может сфокусироваться на соответствующих частях исходного предложения для его перевода на целевой язык. В подобных случаях целесообразно выравнивать целевое предложение по соответствующим частям исходного предложения в процессе перевода. И здесь механизмы внимания могут оказаться очень полезными при выделении соответствующих частей исходного предложения в процессе создания определенной части целевого предложения. Примечательно то, что механизмы внимания не всегда должны помещаться в каркас обучения с подкреплением. В действительности

большинство механизмов внимания в моделях обработки естественного языка не используют обучение с подкреплением, в то же время концентрируясь на определении весов конкретных частей входных данных.

10.2.1. Рекуррентные модели визуального внимания

В работе по рекуррентным моделям визуального внимания [338] для концентрации на важных частях изображения применяется обучение с подкреплением. Идея заключается в том, чтобы использовать (относительно простую) нейронную сеть, в которой высокое разрешение поддерживается только для определенных областей изображения, центрированных на определенном участке. Этот участок может изменяться с течением времени по мере обучения все большему числу признаков релевантных частей изображения, подлежащих исследованию. Выбор определенного участка в конкретный момент времени называется *взглядом* (glimpse). Рекуррентная нейронная сеть служит в качестве контроллера для идентификации точного расположения в каждый момент времени (на каждом временном шаге); этот выбор базируется на обратной связи со взглядом на предыдущем временном шаге. В [338] показано, что использование простой нейронной сети, так называемой *сети обработки взгляда* (glimpse network), для обработки изображения с помощью обучения с подкреплением способно превосходить традиционные сверточные сети в задачах классификации.

Мы рассмотрим динамические условия, в которых изображение может быть частично наблюдаемым, а наблюдаемые части могут меняться от одного временного шага t к другому. Такая постановка задачи является довольно общей, хотя, безусловно, мы можем использовать ее в более конкретных условиях, когда изображение \bar{X}_t остается фиксированным во времени. Общую архитектуру можно представить на модульном уровне, рассматривая отдельные части нейронной сети как “черные ящики”. Описание этих модульных частей приводится ниже.

1. *Датчик взгляда.* Для заданного изображения с представлением \bar{X}_t датчик взгляда создает изображение, подобное тому, которое формируется на сетчатке глаза. Предполагается, что датчик взгляда не имеет полного доступа к изображению (ввиду ограничения полосы пропускания) и ограничен наблюдением лишь небольшой части изображения в высоком разрешении, центрированной в точке l_{t-1} . Это аналогично тому, как глаз воспринимает изображение в реальной жизни. Разрешение определенных участков изображения снижается с увеличением расстояния от точки l_{t-1} . Обозначим это редуцированное представление изображения через $\rho(\bar{X}_t, l_{t-1})$. Датчик взгляда, показанный в левом верхнем углу на рис. 10.2, является частью более крупной сети — *сети обработки взгляда*. Эта сеть обсуждается ниже.

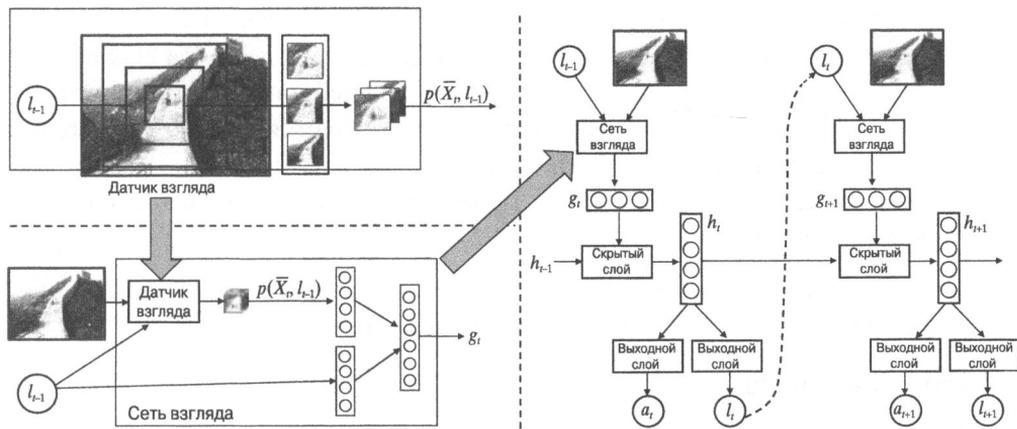


Рис. 10.2. Рекуррентная архитектура, позволяющая задействовать механизм визуального внимания

2. **Сеть обработки взгляда** (или просто *сеть взгляда*). Содержит датчик взгляда и кодирует в скрытых пространствах как положение l_{t-1} , так представление $\rho(\bar{X}_t, l_{t-1})$ взгляда, объединяя их впоследствии в единое скрытое представление с использованием линейного слоя. Результирующий выход g_t является входом, соответствующим t -й временной метке скрытого слоя рекуррентной нейронной сети. Сеть взгляда представлена в правом нижнем углу на рис. 10.2.
3. **Рекуррентная нейронная сеть**. Это основная сеть, создающая на каждом временном шаге выходы, управляемые действиями (для получения вознаграждений). Рекуррентная нейронная сеть включает сеть обработки взгляда, а значит, и датчик взгляда (поскольку он является частью сети обработки взгляда). Обозначим выходное действие сети в момент времени t через a_t , и с каждым таким действием связывается вознаграждение. В простейшем случае вознаграждением может быть метка класса объекта или число из примера с Google Street View. Выводится также положение участка l_t изображения, на котором должен быть сфокусирован взгляд в следующий момент времени. Выход $\pi(a_t)$ реализуется в виде вероятности действия a_t . Эта вероятность определяется посредством функции *Softmax*, как это обычно делается в сетях обучения стратегий (см. рис. 9.6). Тренировка рекуррентной сети осуществляется с использованием целевой функции фреймворка REINFORCE для максимизации ожидаемого вознаграждения с течением времени. Размер поощрения за каждое действие получается путем умножения $\log(\pi(a_t))$ на выгоду от данного действия (см. раздел 9.5.2). Поэтому подход в целом представляет собой метод обучения с подкреплением, в котором выходы, соответствующие положению фокуса

внимания и выполняемому действию, обучаются одновременно. Следует отметить, что предыстория действий рекуррентной сети кодируется в скрытых состояниях h_t . Общая архитектура нейронной сети приведена на рис. 10.2, *справа*. Сеть обработки взгляда является частью этой архитектуры, поскольку рекуррентная сеть задействует взгляд на изображение (или текущее состояние сцены) для выполнения вычислений на каждом временном шаге.

Следует отметить, что в этих контекстах использование архитектуры рекуррентной нейронной сети полезно, но не является необходимым.

Обучение с подкреплением

Рассматриваемый подход встраивается в рамки обучения с подкреплением, что позволяет использовать его вместо распознавания или классификации изображений в любой визуальной задаче, требующей обучения с подкреплением (такой, например, как выбор действий робота для достижения определенной цели). Тем не менее частным случаем данного подхода является обучение с учителем.

Действия a_t соответствуют выбору метки класса с помощью Softmax-предсказания. Вознаграждение r_t на t -м временном шаге равно 1 в случае корректной классификации после выполнения t шагов в процессе данного развертывания или 0 в противном случае. Общее вознаграждение R_t на t -м временном шаге определяется суммой всех дисконтированных вознаграждений по будущим временным шагам. Однако это действие может меняться в зависимости от конкретного приложения. Например, если речь идет о задаче аннотирования изображений, то действию может соответствовать выбор следующего слова подписи.

В рассматриваемых условиях тренировка выполняется аналогично тому, как это делается в подходе, который обсуждался в разделе 9.5.2. Градиент ожидаемого вознаграждения в момент времени t определяется следующим выражением:

$$\nabla E[R_t] = R_t \nabla \log(\pi(a_t)). \quad (10.1)$$

Обратное распространение ошибки в нейронной сети выполняется с использованием этого градиента и *разверток стратегий* (policy rollouts). На практике приходится задействовать ряд разверток, каждая из которых содержит ряд действий. В таком случае для получения окончательного направления спуска необходимо просуммировать градиенты по всем действиям (или мини-пакетам действий). Как это общепринято в методах градиентного спуска по стратегиям, для уменьшения дисперсии из вознаграждений вычитается базовая линия. Поскольку выходом является метка класса на каждом временном шаге, с увеличением количества взглядов точность будет увеличиваться. Чтобы описанный подход работал хорошо, достаточно использовать шесть–восемь взглядов.

10.2.1.1. Применение в задачах аннотирования изображений

В этом разделе описано применение подхода на основе механизма визуального внимания (который обсуждался в предыдущем разделе) в задаче *аннотирования изображений* (image captioning). Эта задача обсуждалась в разделе 7.7.1. В данном подходе входом рекуррентной нейронной сети для первой временной отметки является одиночное представление \bar{v} всего изображения. Если входом служит представление признака всего изображения, то оно определяет вход лишь на *первом временном шаге*, когда генерирование подписи только начинается. Однако в случае применения механизма внимания мы хотим сфокусироваться на той части изображения, которая соответствует генерируемому слову. Поэтому имеет смысл предоставлять различные входы, ориентированные на использование механизма внимания, на различных шагах. Рассмотрим, например, следующую подпись:

“Bird flying during sunset” (Птица, летящая на закате)

Когда генерируется слово “flying”, внимание должно концентрироваться на крыльях птицы, представленной на изображении, а когда генерируется слово “sunset” — на заходящем солнце. В данном случае рекуррентная нейронная сеть должна получать на каждом временном шаге представление изображения, в котором внимание сфокусировано на определенном участке. Кроме того, как обсуждалось в предыдущем разделе, значения, соответствующие этим расположениям, также генерируются рекуррентной сетью на предыдущем временном шаге.

Данный подход может быть реализован с помощью архитектуры, приведенной на рис. 10.2, посредством предсказания на каждом временном шаге одного слова подписи (в качестве действия) вместе с расположением на изображении, на котором должно быть сфокусировано внимание на следующем временном шаге. Эта идея адаптирована в [540], но с определенными модификациями, обусловленными повышенной сложностью задачи. Во-первых, сеть обработки взгляда использует более сложную сверточную архитектуру для создания карты признаков размера 14×14 (рис. 10.3). Вместо того чтобы получать измененную версию изображения на каждом временном шаге с помощью датчика взгляда, в [540] весь процесс начинается с L различных предварительно обработанных вариантов изображения, которые центрируются на различных участках изображения, поэтому механизм внимания ограничивается выбором одного из этих расположений. Затем вместо получения расположения l_t на $(t - 1)$ -м временном шаге получается вектор вероятностей \bar{a}_t порядка L , указывающий на релевантность каждого из L расположений по отношению к тем представлениям, которые были предварительно обработаны сверточной нейронной сетью. В моделях *жесткого внимания* (hard attention) с помощью вектора вероятностей \bar{a}_t сем-

плируется одно из L расположений, а предварительно обработанное представление этого расположения подается в качестве входа для скрытого состояния h_t рекуррентной сети на следующем временном шаге. Иными словами, сеть обработки взгляда в задаче классификации заменяется этим механизмом семплирования. В моделях *мягкого внимания* (soft attention) представительные модели всех L расположений усредняются посредством использования вектора вероятностей $\bar{\alpha}_t$ в качестве весов. Это усредненное представление подается в качестве входа для скрытого состояния на временном шаге t . В моделях мягкого внимания для тренировки используется непосредственно обратное распространение, тогда как в моделях жесткого внимания — алгоритм REINFORCE (см. раздел 9.5.2). Подробно оба этих метода описаны в [540].

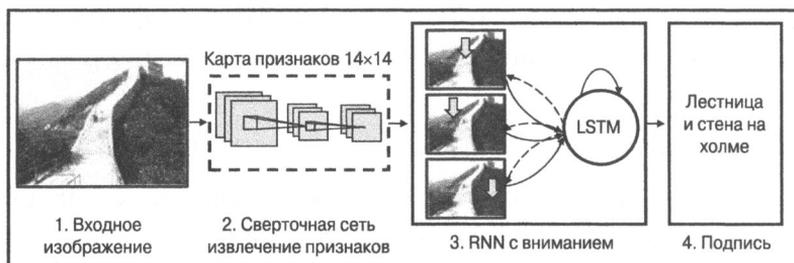


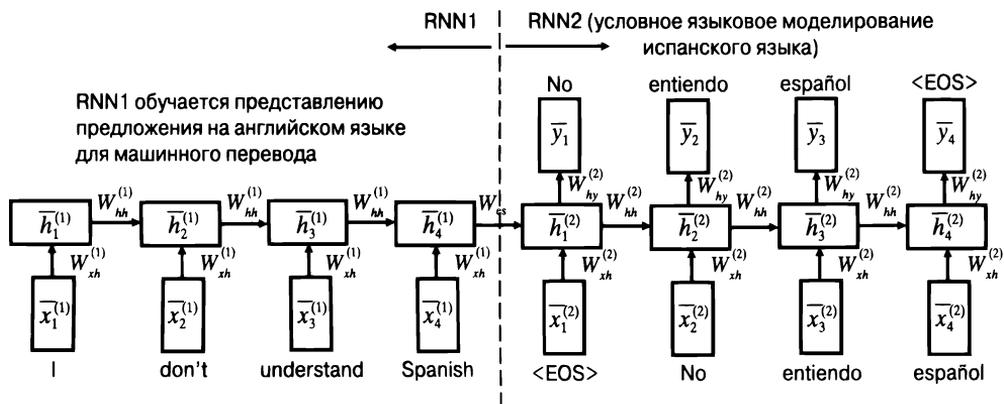
Рис. 10.3. Использование визуального внимания при аннотировании изображений

10.2.2. Механизмы внимания для машинного перевода

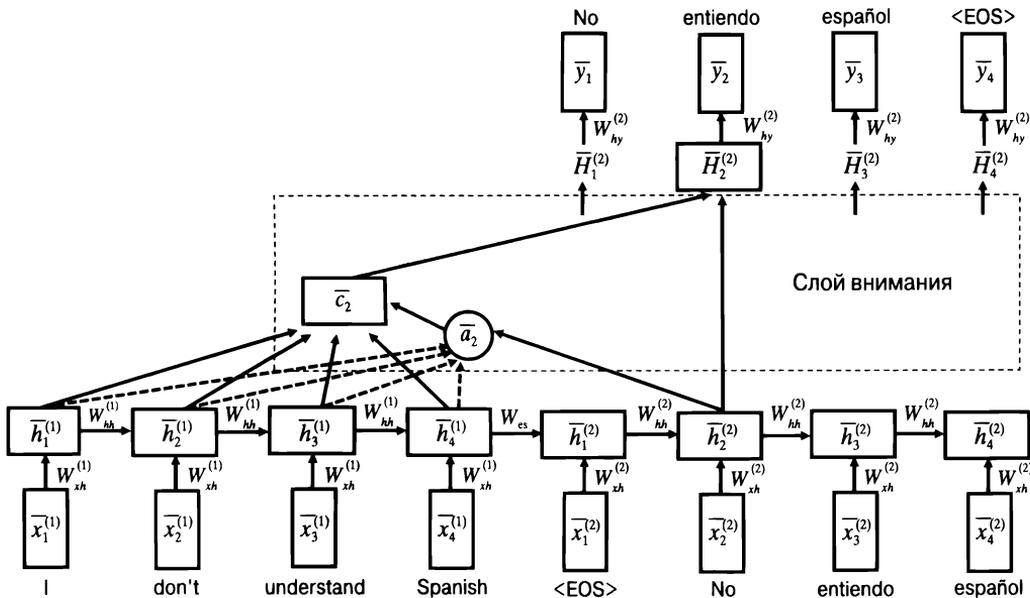
Как обсуждалось в разделе 7.7.2, рекуррентные нейронные сети (и, в частности, их реализации в виде долгой краткосрочной памяти — LSTM) часто применяются для машинного перевода. Ниже мы используем обобщенные обозначения, соответствующие любому типу рекуррентной нейронной сети, хотя в этих условиях вариант LSTM почти всегда наиболее предпочтительный. Ради простоты изложения (а также для упрощения рисунков со схемами нейронных архитектур) ограничимся рассмотрением однослойной сети. На практике используются несколько слоев, однако обобщение нашего рассмотрения на случай многослойных сетей не составит большого труда. Для встраивания механизма внимания в нейронную сеть машинного перевода существует несколько способов. Остановимся на методе, предложенном Луонгом и др. [302], который представляет собой улучшение оригинального механизма, предложенного Бадауна и др. [18].

Начнем с архитектуры, рассмотренной в разделе 7.7.2. Для того чтобы упростить обсуждение, эта архитектура заново воспроизведена на рис. 10.4, а. Обратите внимание на наличие двух рекуррентных нейронных сетей, одна из которых кодирует исходное предложение в представление фиксированной длины, тогда как вторая декодирует это представление в целевое предложение. Таким

образом, в данном случае мы имеем дело непосредственно с обучением по методу “последовательность в последовательность”, который используется для машинного перевода с помощью нейронных сетей. Скрытые состояния исходной и целевой сетей обозначим через $h_t^{(1)}$ и $h_t^{(2)}$, где $h_t^{(1)}$ соответствует скрытому состоянию t -го слова в исходном предложении, а $h_t^{(2)}$ — скрытому состоянию t -го слова в целевом предложении. Эти обозначения заимствованы из раздела 7.7.2.



а) Машинный перевод без использования механизма внимания



б) Машинный перевод с использованием механизма внимания

Рис. 10.4. Нейронная архитектура (а) совпадает с той, которая была приведена на рис. 7.10. В нейронной архитектуре (б) добавлен дополнительный слой внимания

В методах, основанных на внимании, скрытые состояния $h_i^{(2)}$ преобразуются в улучшенные состояния $H_i^{(2)}$ посредством дополнительной обработки в *слое внимания* (attention layer). Задача последнего состоит в том, чтобы встроить контекст из исходных скрытых состояний в целевые скрытые состояния для создания нового, улучшенного набора целевых скрытых состояний.

Выполнение обработки, основанной на внимании, требует нахождения исходного представления, близкого к обрабатываемому в данный момент целевому скрытому состоянию $h_i^{(2)}$. Это достигается за счет использования взвешенного среднего исходных векторов для создания вектора контекста \bar{c}_i :

$$\bar{c}_i = \frac{\sum_{j=1}^{T_s} \exp(\bar{h}_j^{(1)} \cdot \bar{h}_i^{(2)}) \bar{h}_j^{(1)}}{\sum_{j=1}^{T_s} \exp(\bar{h}_j^{(1)} \cdot \bar{h}_i^{(2)})}, \quad (10.2)$$

где T_s — длина исходного предложения. Из всех версий методов, которые обсуждаются в [18, 302], данный способ является самым простым. Однако существуют и другие альтернативные методы, часть из которых параметризована. Одним из способов наблюдения и взвешивания исходных векторов является использование понятия *переменной внимания* $a(t, s)$, указывающей на важность исходного слова s для целевого слова t :

$$a(t, s) = \frac{\exp(\bar{h}_s^{(1)} \cdot \bar{h}_t^{(2)})}{\sum_{j=1}^{T_s} \exp(\bar{h}_j^{(1)} \cdot \bar{h}_t^{(2)})}. \quad (10.3)$$

Назовем вектор $[a(t, 1), a(t, 2), \dots, a(t, T_s)]$, специфический для целевого слова t , вектором внимания \bar{a}_t . Этот вектор можно рассматривать как набор вероятностных весов, сумма которых равна 1, и его порядок зависит от длины T_s исходного предложения. Нетрудно видеть, что уравнение 10.2 создается в виде взвешенной по вниманию суммы исходных скрытых векторов, где вес внимания целевого слова t по отношению к исходному слову s равен $a(t, s)$. Иными словами, уравнение 10.2 можно переписать в следующем виде:

$$\bar{c}_i = \sum_{j=1}^{T_s} a(t, s) \bar{h}_j^{(1)}. \quad (10.4)$$

По сути, этот подход идентифицирует контекстное представление исходных скрытых состояний, наиболее релевантных по отношению к рассматриваемому в данный момент текущему целевому скрытому состоянию. Релевантность определяется посредством сходства скалярного произведения векторов исход-

ных и скрытых состояний и сводится к вектору внимания. Поэтому мы создаем новое скрытое состояние $H_i^{(2)}$, объединяющее в себе информацию о контексте и первоначальное скрытое состояние следующим образом:

$$\bar{H}_i^{(2)} = \tanh \left(W_c \begin{bmatrix} \bar{c}_i \\ \bar{h}_i^{(2)} \end{bmatrix} \right). \quad (10.5)$$

Для окончательного предсказания вместо первоначального скрытого представления $\bar{h}_i^{(2)}$ используется вновь созданное скрытое представление $\bar{H}_i^{(2)}$. Общая архитектура системы на основе механизма внимания приведена на рис. 10.4, б. Данная модель усилена по сравнению с той, которая приведена на рис. 10.4, а, за счет добавления механизма внимания. В [302] эта модель названа *моделью глобального внимания* (global attention model). Это модель мягкого внимания, поскольку вместо строгих заключений относительно того, какое из исходных слов является наиболее релевантным по отношению к целевому слову, они взвешиваются с использованием вероятностных весов. В оригинальной работе [302] обсуждается еще одна, *локальная* модель, в которой делаются однозначные заключения о релевантности целевых слов. Для ознакомления с подробным описанием этой модели обратитесь к [302].

Варианты улучшения

Базовую модель можно усовершенствовать, используя несколько вариантов ее улучшения. Во-первых, вектор внимания \bar{a}_i вычисляется через экспоненты от скалярных произведений векторов $h_i^{(1)}$ и $\bar{h}_s^{(2)}$ (см. уравнение 10.3). Эти скалярные произведения также называются *оценками сходства*. В действительности нет никаких причин для того, чтобы аналогичные позиции в исходном и целевом предложениях имели одинаковые скрытые состояния. Фактически исходная и целевая сети даже не обязаны использовать скрытые представления одной и той же размерности (хотя именно так часто и делается на практике). Тем не менее в [302] было показано, что оценки сходства на основе скалярного произведения хорошо работают в моделях глобального внимания и оказались наилучшим выбором по сравнению с параметризованными альтернативами. Не исключено, что удовлетворительная работа этого простого подхода является результатом того, что он регуляризирует данную модель. Альтернативные параметризованные варианты вычисления сходства работают лучше в локальных моделях (основанных на механизме жесткого внимания), которые нами подробно не обсуждаются.

В большинстве этих альтернативных моделей вычисления сходства управляются параметрами, что обеспечивает дополнительную гибкость в отношении связывания исходных и целевых позиций. Возможны следующие варианты оценки сходства:

$$\text{Оценка}(t, s) = \begin{cases} \bar{h}_s^{(1)} \cdot \bar{h}_t^{(2)} & \text{Скалярное произведение} \\ (\bar{h}_t^{(2)})^T W_a \bar{h}_s^{(1)} & \text{General: матрица параметров } W_a \\ \bar{v}_a^T \tanh \left(W_a \begin{bmatrix} \bar{h}_s^{(1)} \\ \bar{h}_t^{(2)} \end{bmatrix} \right) & \text{Concat: матрица параметров } W_a \text{ и вектор } \bar{v}_a \end{cases} \quad (10.6)$$

Первый из этих вариантов идентичен обсуждавшемуся в предыдущем разделе в связи с уравнением 10.3. Две остальные модели носят названия *General* (общая) и *Concat* (конкатенированная). Оба этих варианта параметризуются с помощью векторов весов, также указанных выше. Вычислив оценки сходства, можно вычислить значения внимания тем же способом, что и в случае оценок сходства на основе скалярного произведения:

$$a(t, s) = \frac{\exp(\text{Оценка}(t, s))}{\sum_{j=1}^{T_s} \exp(\text{Оценка}(t, j))}. \quad (10.7)$$

Эти значения внимания используются точно так же, как и в случае оценки сходства с помощью скалярного произведения. Матрица параметров W_a и вектор \bar{v}_a должны быть обучены в процессе тренировки. Модель *Concat* была предложена в более ранней работе [18], тогда как модель *General*, по всей видимости, хорошо работает в случае использования механизма жесткого внимания.

Между этой моделью [302] и моделью, предложенной ранее Бадану и др. [18], существуют определенные различия. Мы выбрали эту модель, поскольку она проще и непосредственно иллюстрирует базовые концепции. Кроме того, в соответствии с результатами, приведенными в [302], она, вероятно, обеспечивает лучшую производительность. Также существуют различия в выборе архитектуры нейронной сети. В работе Луонга и др. использовалась однонаправленная рекуррентная нейронная сеть, тогда как в работе Бадану и др. отдано предпочтение двунаправленной рекуррентной нейронной сети.

В отличие от задачи аннотирования изображений, рассмотренной в предыдущем разделе, в машинном переводе применяется модель мягкого внимания. По-видимому, условия жесткого внимания изначально ориентировались на обучение с подкреплением, тогда как условия мягкого внимания допускают дифференцирование и поэтому могут использоваться с обратным распространением ошибки. В [302] также предлагается механизм локального внимания, фокусирующийся на небольшом окне контекста. Такой подход имеет общие черты с механизмом жесткого внимания (например, в части фокусирования внимания на небольшой области изображения, что обсуждалось в предыдущем разделе). Но вместе с тем этот подход не совсем жесткий, поскольку он фокусируется на

небольшой части предложения, используя веса важности, генерируемые механизмом внимания. Такой подход способен реализовать локальный механизм, избегая трудностей тренировки в методе обучения с подкреплением.

10.3. Нейронные сети с внешней памятью

В последние годы было предложено несколько родственных архитектур, дополняющих нейронные сети с *постоянной памятью* (persistent memory), в которых понятие памяти отчетливо отделяется от вычислений и предоставляется возможность управлять способами селективного доступа к памяти и изменения ее ячеек. Можно считать, что сети LSTM обладают постоянной памятью, хотя память в них не отделена в явном виде от вычислений. Это обусловлено тем, что вычисления в нейронной сети тесно связаны со значениями скрытых состояний, которые играют роль хранилищ промежуточных результатов вычислений.

Нейронные машины Тьюринга — это нейронные сети с *внешней памятью*. Базовая нейронная сеть может обращаться к внешней памяти для выполнения операций чтения/записи и поэтому играет роль контроллера при выполнении вычислений. За исключением LSTM-сетей большинство нейронных сетей не использует понятие постоянной памяти на длинных временных промежутках. В действительности в традиционных нейронных сетях (включая LSTM-сети) понятия вычислений и памяти не разделяются отчетливым образом. Возможность манипулирования постоянной памятью, если она сочетается с отчетливым разделением вычислений и памяти, приводит к *программируемому компьютеру*, который может имитировать алгоритмы на основе примеров входа и выхода. Этот принцип привел к появлению ряда родственных архитектур, таких как *нейронные машины Тьюринга* [158], *дифференциальные нейронные компьютеры* [159] и *сети с памятью* [528].

Почему полезно учиться на примерах входа и выхода? Почти все универсальные формы ИИ базируются на предположении о возможности имитировать поведение биологических существ, когда мы имеем лишь примеры входов (например, сигналы датчиков) и выходов (например, действия) без ясного понимания того, что собой представляют алгоритмы/функции, которые фактически вычислялись для данного набора поведений. Чтобы понять, в чем состоит трудность обучения на примерах, начнем с задачи сортировки. И хотя алгоритмы сортировки хорошо известны и строго определены, мы рассмотрим вымышленную ситуацию, в которой отсутствует доступ к этим определениям и алгоритмам. Иными словами, алгоритм начинает работать в условиях, когда ему ничего не известно о том, что собой представляет сортировка. Имеются лишь примеры входных данных и результаты их сортировки.

10.3.1. Воображаемая видеоигра: обучение сортировке на примерах

Несмотря на то что сортировка набора чисел с помощью любого известного алгоритма (например, быстрой сортировки) не составляет труда, эта задача значительно усложнится, если нам не сообщат, что функцией данного алгоритма является сортировка чисел. Вместо этого нам предоставляют лишь пары перемешанных входов и отсортированных выходов, и мы должны автоматически обучиться последовательностям действий для любого заданного входа, чтобы выход отражал то, чему мы обучились на примерах. Поэтому задачей является обучение сортировке на примерах, используя конкретный набор predetermined действий. Это обобщенное представление машинного обучения, в котором входы и выходы могут иметь любой формат (например, это могут быть пиксели или звуки), и задача заключается в том, чтобы научиться выполнять преобразование входа в выход путем выполнения последовательности действий. Эти действия являются элементарными шагами, которые разрешено выполнять в данном алгоритме. Мы уже можем заметить, что такой подход, управляемый действиями, тесно связан с методологиями обучения с подкреплением, которые обсуждались в главе 9.

Рассмотрим простой случай, когда мы хотим сортировать только последовательности, состоящие из четырех чисел, и поэтому имеем на нашем “экране видеоигры” четыре позиции, содержащие текущее состояние исходной последовательности чисел. Экран этой воображаемой видеоигры показан на рис. 10.5, *a*. Существует шесть возможных действий, которые игрок может совершить, и каждое действие имеет вид функции $\text{SWAP}(i, j)$, которая меняет местами содержимое в расположениях i и j . Поскольку существуют четыре возможных значения каждой из позиций i и j , общее количество возможных действий равно $\binom{4}{2} = 6$. Целью видеоигры является сортировка чисел за счет использования как можно меньшего количества перестановок. Мы хотим сконструировать алгоритм путем разумного выбора перестановок. Кроме того, алгоритм машинного обучения не наделяется предварительным знанием того, что выходы должны представлять собой отсортированные последовательности, и ему предоставляются лишь примеры входов и выходов для создания модели, которая (в идеальном случае) обучается стратегии для преобразования входов в их отсортированные версии. Кроме того, игроку не предоставляются пары “вход — выход”, и он лишь поощряется вознаграждениями, когда совершает удачные перестановки, которые ведут к отсортированному порядку.

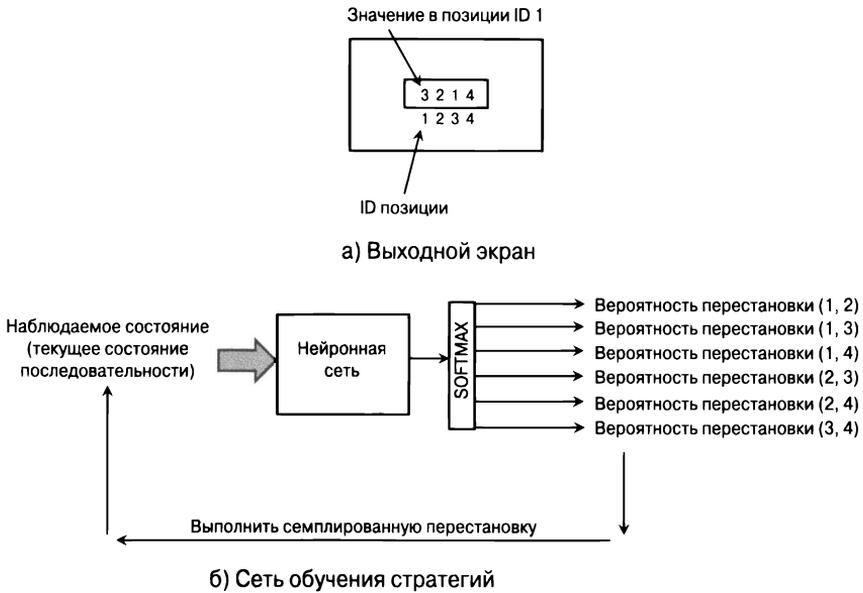


Рис. 10.5. Выходной экран и сеть стратегий для обучения воображаемой игре в сортировку

Эта ситуация почти идентична ситуации с видеоигрой Atari, которая обсуждалась в главе 9. Например, мы можем использовать сеть обучения стратегий, в которой текущая последовательность четырех чисел является входом нейронной сети, а выходом является вероятность каждого из шести возможных действий. Эта архитектура приведена на рис. 10.5, б. Полезно сравнить ее с сетью обучения стратегий, приведенной на рис. 9.6. Предпочтительность каждого действия можно моделировать рядом эвристических способов. Например, при наивном подходе можно развертывать стратегию для T ходов-перестановок и устанавливать вознаграждение равным $+1$, если к этому моменту удастся получить корректный выход, или -1 в противном случае. Уменьшение значения T ускоряет процесс, но ухудшает точность. Также можно определить улучшенную функцию выгоды, в соответствии с которой вознаграждение за последовательность ходов определяется в зависимости от того, насколько удастся приблизиться к известному выходу.

Рассмотрим ситуацию, в которой вероятность действия $a = \text{SWAP}(i, j)$ равна $\pi(a)$ (выводится функцией *Softmax* нейронной сети), а выгода равна $F(a)$. Тогда в методах градиентного спуска по стратегиям мы устанавливаем целевую функцию J_a , которая представляет собой ожидаемую выгоду от выполнения действия a . Как обсуждалось в разделе 9.5, градиент этой выгоды по параметрам сети обучения стратегий задается следующим выражением:

$$\nabla J_a = F(a) \cdot \nabla \log(\pi(a)). \quad (10.8)$$

Этот градиент добавляется сверх мини-пакета действий из предыдущих развертываний игры и служит для обновления весов нейронной сети. Интересно отметить, что обучение с подкреплением способствует реализации стратегии для алгоритма, который обучается на примерах.

10.3.1.1. Реализация перестановок посредством операций с памятью

Описанную выше видеоигру также можно реализовать с помощью нейронной сети, в которой допускается обращение к памяти для выполнения операций чтения/записи, и мы хотим, чтобы сортировка последовательности осуществлялась посредством как можно меньшего количества таких операций. Например, одним из вариантов решения задачи может быть сохранение состояния последовательности во внешней памяти с дополнительным пространством для хранения временных переменных, используемых при перестановках. Как будет показано ниже, можно легко реализовать перестановки, задействуя память для выполнения операций чтения/записи. Для копирования состояний из одного временного шага в другой можно применить рекуррентную нейронную сеть. Операция $SWAP(i, j)$ может быть реализована посредством чтения i и j из ячеек памяти с последующим их сохранением во временных регистрах. После этого регистр i может быть записан в ячейку памяти j , а регистр j — в ячейку i . Таким образом, для реализации перестановок можно использовать последовательности операций чтения/записи с памятью. Иными словами, мы можем реализовать стратегию сортировки путем обучения рекуррентной сети-“контроллера”, которая принимает решения о том, какие ячейки памяти следует использовать для чтения, а какие — для записи. В то же время, если мы создадим обобщенную архитектуру, основанную на обращениях к памяти, то контроллер сможет обучиться более эффективной стратегии, чем простая реализация перестановок. Важно понимать, что в данном случае весьма полезно иметь постоянную память в той или иной форме, обеспечивающую хранение текущего состояния сортируемой последовательности. Состояния нейронной сети, включая (просто) рекуррентную нейронную сеть, слишком кратковременны для того, чтобы в них можно было сохранять информацию такого типа.

Расширение доступа к большим объемам памяти увеличивает возможности архитектуры, но при этом усложняет ее. При небольших объемах доступной памяти сеть стратегий может обучаться лишь простым $O(n^2)$ -алгоритмам, использующим перестановки. С другой стороны, с увеличением объема доступной памяти сеть обучения стратегий сможет выполнять операции чтения/записи с памятью для синтеза более широкого круга операций, что обеспечит возможность обучения более быстрым алгоритмам сортировки. В конечном счете функция вознаграждения, которая поощряет стратегию, обеспечивающую

достижение корректного порядка сортировки за T шагов, будет отдавать предпочтение стратегиям, требующим выполнения меньшего количества ходов.

10.3.2. Нейронные машины Тьюринга

Давно распознанной слабостью нейронных сетей является то, что они неспособны отчетливо отделить внутренние переменные (т.е. скрытые состояния) от выполняемых в сети вычислений, что делает состояния кратковременными (в отличие от биологической или компьютерной памяти). Нейронная сеть, допускающая использование внешней памяти и выполнение операций чтения/записи над ее различными ячейками контролируемым образом, обладает очень большими возможностями и открывает путь к имитации обычных классов алгоритмов, которые могут быть реализованы на современных компьютерах. Архитектуру подобного типа называют *нейронной машиной Тьюринга* (neural Turing machine) или *дифференциальным нейронным компьютером* (differentiable neural computer). Эта архитектура называется *дифференциальной*, поскольку обучается имитации алгоритмов (образующих дискретную последовательность шагов) с использованием непрерывной оптимизации. Непрерывная оптимизация обладает тем преимуществом, что позволяет применять к переменным дифференциальное исчисление, а значит, допускает возможность использования механизма обратного распространения ошибки для обучения оптимизированным алгоритмическим шагам.

Следует отметить, что традиционные нейронные сети также располагают памятью в виде скрытых состояний. Частным примером этого могут служить LSTM-сети, в которых некоторые из скрытых состояний являются долговременными. Однако в нейронных машинах Тьюринга проводится четкое разделение между внешней памятью и скрытыми состояниями внутри нейронной сети. Скрытые состояния в пределах нейронной сети можно рассматривать как регистры CPU, которые служат для промежуточных вычислений, тогда как внешняя память используется для проведения вычислений с постоянно хранимыми переменными. Внешняя память наделяет нейронную машину Тьюринга возможностями выполнения вычислений способом, близким к тому, как программисты манипулируют данными на современных компьютерах. Благодаря этому свойству нейронные машины Тьюринга часто приобретают большую обобщающую способность, чем похожие модели наподобие LSTM. Кроме того, этот подход открывает путь к определению постоянно хранимых (персистентных) структур данных, надежно отделенных от нейронных вычислений. Невозможность отчетливого отделения переменных программы от вычислительных операций давно была признана одним из самых слабых мест традиционных нейронных сетей.

Общая архитектура нейронной машины Тьюринга приведена на рис. 10.6. Сердцевиной машины Тьюринга является контроллер, который реализуется с помощью одной из разновидностей рекуррентной нейронной сети (хотя возможны и другие альтернативы). Рекуррентная архитектура применяется для переноса состояния из одного временного шага в другой в процессе реализации нейронной машиной Тьюринга любого конкретного алгоритма или стратегии. Например, в нашей игре в сортировку текущее состояние последовательности чисел переносится из каждого очередного шага в следующий. На каждом временном шаге сеть получает входы от окружения и записывает выходы в окружение. Кроме того, она имеет внешнюю память, к которой может обращаться для выполнения операций чтения/записи с использованием считывающей и записывающей головок. Память структурируется в виде матрицы размера $N \times t$, где N — количество ячеек памяти, а t — длина каждой из них. Для t -мерного вектора в i -й строке памяти на t -м временном шаге используется обозначение $\bar{M}_t(i)$.

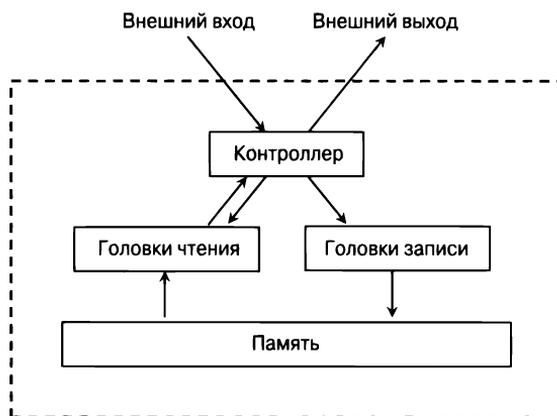


Рис. 10.6. Нейронная машина Тьюринга

Головки выводят специальный вес $w_t(i) \in (0, 1)$, ассоциируемый с каждым расположением i на временном шаге t и управляющий уровнем, до которого следует осуществлять чтение или запись в каждое выходное расположение. Иными словами, если головка чтения выводит вес 0,1, то все, что читается из i -го расположения в памяти, интерпретируется после масштабирования с коэффициентом 0,1, а затем суммируется со взвешенными результатами по всем различным значениям i . Вес головки записи определяется аналогичным образом, о чем будет говориться далее. Обратите внимание на индекс t в обозначении веса, указывающий на то, что на каждом временном шаге t формируется отдельный набор весов. В данном примере этот вес играет роль Softmax-вероятности перестановки в видеоигре с сортировкой, так что дискретное действие преобразуется в непрерывное дифференцируемое значение. Однако имеется одно

отличие, которое заключается в том, что нейронная машина Тьюринга не определяется стохастически, как сеть обучения стратегий, рассмотренная в предыдущем разделе. Другими словами, мы используем вес $w_t(i)$ не для стохастического семплирования памяти, а для определения того, в какой степени должно выполняться считывание или затирание содержимого данной ячейки. Иногда полезно рассматривать каждое обновление как ожидаемую величину той степени, в которой стохастическая стратегия будет считывать или обновлять данные. Ниже мы предоставим более формальное описание этого процесса.

Как только определены веса $w_t(i)$, можно прочитать m -мерный вектор, представляющий собой взвешенную комбинацию векторов, соответствующих различным расположениям в памяти:

$$r_t = \sum_{i=1}^N w_t(i) \bar{M}_t(i). \quad (10.9)$$

Веса $w_t(i)$ определяются таким образом, чтобы их сумма по всем N векторам памяти была равна 1 (подобно вероятностям):

$$\sum_{i=1}^N w_t(i) = 1. \quad (10.10)$$

Запись базируется на принципе внесения изменений путем первоначального затирания части памяти с последующим добавлением в нее новой информации. Поэтому на i -м временном шаге записывающая головка выдает вектор весов $w_t(i)$ вместе с векторами стирания e_t и добавления a_t порядка m . Обновление ячейки осуществляется комбинированием затирания и добавления информации. Сначала выполняется операция затирания:

$$\bar{M}'_t(i) \leftarrow \underbrace{\bar{M}_{t-1}(i) \odot (1 - w_t(i) \bar{e}_t(i))}_{\text{Частичное затирание}}, \quad (10.11)$$

где символом \odot обозначена операция поэлементного умножения по m измерениям i -й строки матрицы памяти. Каждый элемент вектора стирания e_t извлекается из интервала $(0, 1)$, и m -мерный вектор стирания обеспечивает тонкое управление выбором элементов из m -мерной строки, которая может быть затерта. Также допускается использование нескольких записывающих головок, причем порядок выполнения умножений с помощью различных головок не имеет значения, поскольку операция умножения обладает коммутативным и ассоциативным свойствами. Операция добавления выполняется следующим образом:

$$\bar{M}_t(i) = \underbrace{\bar{M}'_t(i) + w_t(i) \bar{a}_t(i)}_{\text{Частичное добавление}}. \quad (10.12)$$

Если имеется несколько записывающих головок, то порядок выполнения операций, связанных с добавлением новой информации, не важен. Однако для того, чтобы обеспечить согласованность результата, независимо от порядка выполнения операций добавления, перед ними должны быть выполнены все операции затирания.

Обратите внимание на крайне сдержанный характер внесения изменений в состояния ячеек, достигаемый за счет того, что сумма весов равна 1. Такое обновление можно рассматривать на интуитивном уровне как аналог стохастического выбора одной из N строк памяти (с вероятностью $w_i(i)$) и последующего семплирования индивидуальных элементов (с вероятностями e_i) для их обновления. Однако подобные обновления не являются дифференцируемыми (если только они не параметризованы с помощью приемов градиентного спуска по стратегиям, заимствованных из обучения с подкреплением). В данном случае мы ограничиваемся мягким обновлением, при котором все ячейки подвергаются лишь незначительным изменениям, что позволяет сохранить их дифференцируемость. Использование нескольких записывающих головок приводит к более резкому характеру обновлений. Можно также провести аналогию между этими весами и селективным обменом информацией между скрытыми состояниями и состояниями памяти в LSTM-сетях с помощью сигмоидных функций для управления объемами информации, читаемой и записываемой посредством обращения к долгосрочной памяти (см. главу 7).

Взвешивание как механизм адресации

Процедуру взвешивания можно рассматривать с точки зрения того, как работает механизм адресации. Например, возможен вариант, когда для чтения или записи информации в память выбирается i -я строка матрицы памяти, что соответствует жесткому механизму. Мягкий механизм адресации нейронной машины Тьюринга несколько отличается от этого тем, что информация читается и записывается во все ячейки, но величина вносимых поправок очень мала. До сих пор мы не обсуждали, как работает этот механизм адресации установки весов $w_i(i)$. Такая адресация может осуществляться либо по содержимому, либо по расположению.

В случае адресации по содержимому вектор \vec{v}_i порядка t , являющийся *вектором ключей*, используется для взвешивания местоположений на основании их сходства с вектором \vec{v}_i , определяемого с помощью скалярного произведения. Для регулирования значимости сходства, определяемого точечным произведением, при взвешивании применяется экспоненциальный механизм:

$$w_i^c(i) = \frac{\exp(\cos(\vec{v}_i, \vec{M}_i(i)))}{\sum_{j=1}^N \exp(\cos(\vec{v}_i, \vec{M}_i(j)))}. \quad (10.13)$$

Обратите внимание на то, что в обозначении $w_i^c(i)$ появился верхний индекс, указывающий на то, что в данном случае применяется механизм взвешивания, основанный исключительно на содержимом. Дополнительная гибкость достигается за счет использования параметра температуры в аргументах экспонент для настройки уровня жесткости адресации. Например, если имеется температурный параметр β_i , то веса могут вычисляться следующим образом:

$$w_i^c(i) = \frac{\exp(\beta_i \cos(\bar{v}_i, \bar{M}_i(i)))}{\sum_{j=1}^N \exp(\beta_i \cos(\bar{v}_i, \bar{M}_i(j)))}. \quad (10.14)$$

Увеличение β_i приближает этот подход к жесткой адресации, в то время как уменьшение β_i подобно мягкой адресации. Если желательно задействовать только адресацию по содержимому, то можно использовать веса $w_i(i) = w_i^c(i)$. Адресация, основанная исключительно на содержимом, подобна почти случайному доступу. Например, если содержимое расположения $\bar{M}_i(i)$ включает собственное расположение, то обращение к памяти по ключу напоминает мягкий произвольный доступ к памяти.

Суть второго метода адресации заключается в использовании последовательной адресации по отношению к расположению на предыдущем временном шаге. Такой подход называют адресацией на основе местоположения. В этом виде адресации в качестве стартовых точек используются значение веса содержимого $w_i^c(i)$ на текущей итерации и окончательные веса $w_{i-1}(i)$ на предыдущей итерации. В первую очередь *интерполяция* подмешивает частичный уровень произвольного доступа к расположению, к которому осуществлялось обращение на предыдущей итерации (посредством веса содержимого), после чего операция *сдвига* добавляет элемент последовательного доступа. Наконец, с помощью параметра — аналога температуры мягкость адресации уточняется. Весь процесс адресации на основе расположения основан на следующих операциях:

Веса содержимого(\bar{v}_i, β_i) \Rightarrow Интерполяция(g_i) \Rightarrow Сдвиг(\bar{s}_i) \Rightarrow Уточнение(γ_i).

Каждая из этих операций использует выходы контроллера в качестве входных параметров, указанных выше в соответствующих операциях. Поскольку создание весов содержимого $w_i^c(i)$ уже обсуждалось, опишем остальные три шага.

1. *Интерполяция*. В этом случае вектор из предыдущей итерации комбинируется с весами содержимого $w_i^c(i)$, созданными во время текущей итерации с использованием весов интерполяции $g_i \in (0, 1)[0, 1]$, которые выводятся контроллером. Таким образом, имеем

$$w_i^g(i) = g_i \cdot w_i^c(i) + (1 - g_i) \cdot w_{i-1}(i). \quad (10.15)$$

Заметьте, что если g , равно нулю, то содержимое вообще не используется.

2. *Сдвиг*. В этом случае выполняется поворотное смещение, при котором используется нормализованный вектор целочисленных смещений. Рассмотрим, например, ситуацию, когда $s_i[-1] = 0,2$, $s_i[0] = 0,5$, а $s_i[1] = 0,3$. Это означает, что веса должны смещаться на -1 с *весом пропускания*, равным $0,2$, и на 1 с *весом пропускания*, равным $0,3$. Поэтому определяем смещенный вектор $w_i^s(i)$ следующим образом:

$$w_i^s(i) = \sum_{j=-1}^1 w_i^g(i) \cdot s_i[j]. \quad (10.16)$$

Здесь индекс в $s_i[i - j]$ применяется в сочетании с функцией взятия модуля, чтобы он работал во всем целочисленном интервале от -1 до $+1$ (или в другом целочисленном интервале, на котором определены величины $s_i[i - j]$).

3. *Уточнение*. Процесс уточнения весов подразумевает взятие текущего набора весов и смещение их в сторону нулевых значений или 1 без изменения их упорядочения. Для этого используется параметр $\gamma_i \geq 1$, где большие значения γ_i создают более жесткие значения:

$$w_i(i) = \frac{[w_i^s(i)]^{\gamma_i}}{\sum_{j=-1}^1 [w_i^s(j)]^{\gamma_i}}. \quad (10.17)$$

Параметр γ_i играет ту же роль, что и параметр температуры β_i в случае уточнения весов на основании содержимого. Этот тип уточнения очень важен, поскольку механизм смещения приводит к определенному размытию весов.

Цель описанных действий заключается в следующем. Прежде всего, можно перейти к режиму адресации, основанному исключительно на содержимом, используя вес пропускания g_i , равный 1 . Это можно рассматривать в качестве своеобразного механизма произвольного доступа к памяти по вектору-ключу. Целью использования вектора весов $w_{i-1}(i)$ предыдущей итерации на стадии интерполяции является обеспечение возможности последовательного доступа из опорной точки предыдущего шага. Вектор смещения определяет, насколько мы хотим сместиться из опорной точки в соответствии с вектором интерполяции. Наконец, уточнение позволяет контролировать уровень мягкости адресации.

Архитектура контроллера

Важным проектным решением является выбор нейронной архитектуры контроллера. Естественным выбором будет использование рекуррентной ней-

ронной сети, в которой уже существует понятие временных состояний. Кроме того, LSTM предоставляет внутреннюю память в дополнение к внешней памяти нейронной машины Тьюринга. Состояния в нейронной сети подобны регистрам CPU, используемым для внутренних вычислений, однако они не являются персистентными (в отличие от внешней памяти). Следует подчеркнуть, что в случае привлечения внешней памяти использование рекуррентной сети не является абсолютно необходимым. Это обусловлено тем, что память позволяет фиксировать состояния. Выполнение операций чтения/записи с использованием одного и того же набора расположений ячеек памяти на протяжении последовательных временных шагов обеспечивает запоминание временных состояний подобно рекуррентной сети. Поэтому в контроллере можно применять также сеть прямого распространения, которая обеспечивает большую прозрачность по сравнению со скрытыми состояниями контроллера. Основным ограничением архитектуры прямого распространения является то обстоятельство, что количество операций, выполняемых на каждом временном шаге, ограничивается количеством считывающих и записывающих головок.

Сравнение с рекуррентными нейронными сетями и сетями LSTM

Как известно, все рекуррентные нейронные сети являются *полными по Тьюрингу* (Turing complete) [444], т.е. их можно использовать для имитации любого вычислительного алгоритма. Поэтому с теоретической точки зрения нейронные машины Тьюринга не добавляют ничего нового к внутренним возможностям любой рекуррентной нейронной сети (включая LSTM). Тем не менее, несмотря на тьюринг-полноту рекуррентных сетей, на практике они подвержены определенным ограничениям касательно их производительность и способности обобщаться на наборы данных, содержащие длинные последовательности. Например, если мы тренируем рекуррентную сеть на последовательностях определенного размера, а затем используем ее на тестовых данных с другим распределением размеров, то производительность ухудшится.

Управляемая форма доступа к внешней памяти в нейронной машине Тьюринга предоставляет ей определенные преимущества по сравнению с рекуррентной нейронной сетью, в которой значения, хранящиеся в кратковременных скрытых состояниях, тесно интегрируются с вычислениями. Несмотря на то что сети LSTM дополняются собственной внутренней памятью, обновление которой несколько затруднено, процессы вычислений и обращения к памяти все еще остаются не полностью разделенными (как в современных компьютерах). В действительности во всех нейронных сетях объем вычислений (т.е. количество активаций) и объем памяти (т.е. количество скрытых состояний) также тесно связаны при выполнении любой работы. Четкое разделение памяти

и вычислений позволяет управлять операциями с памятью более интерпретируемым способом, в известной степени напоминающим то, как программисты читают и записывают информацию с использованием внутренних структур. Например, в системах “вопрос — ответ” мы хотим иметь возможность читать фрагмент текста, а затем отвечать на связанные с ним вопросы, и для этого требуются улучшенные возможности контроля чтения и сохранения данных в памяти того или иного вида.

Результаты сравнительных экспериментов [158] говорят о том, что в случае длинных последовательностей входных данных нейронная машина Тьюринга работает лучше, чем LSTM. В одном из экспериментов LSTM и нейронной машине Тьюринга предоставлялись идентичные пары входных и выходных последовательностей. Задача заключалась в копировании входа на выход. В подобном случае нейронная машина Тьюринга обычно работала лучше, чем LSTM, особенно если входы были длинными. В отличие от не интерпретируемых LSTM, операции в сети с памятью были гораздо более интерпретируемыми, и алгоритм копирования, неявно обучаемый машиной Тьюринга, выполнял действия, напоминающие те, которые выполнял бы программист, решая такую же задачу. В конечном счете алгоритм мог обобщаться даже на более длинные последовательности, чем те, которые встречались во время тренировки с нейронной машиной Тьюринга (но не в такой степени, как в случае LSTM). В каком-то смысле интуитивный способ, которым нейронная машина Тьюринга обрабатывает обновления памяти при переходе от одного временного шага к другому, обеспечивает полезную регуляризацию. Например, если алгоритм копирования нейронной машины Тьюринга имитирует стиль кодирования человека-программиста при реализации алгоритма копирования, то он работает лучше с длинными последовательностями во время тестирования.

Кроме того, было экспериментально показано, что нейронная машина Тьюринга хорошо справляется с задачей *ассоциативного повторного обращения* (associative recall), при котором входом является последовательность элементов вместе со случайным элементом, выбранным из этой последовательности. Выходом является следующий элемент последовательности. Опять-таки, нейронная машина Тьюринга обучилась этой задаче лучше, чем LSTM. Помимо этого было реализовано также приложение сортировки [158]. Несмотря на относительную простоту всех этих приложений, данная работа отличается своим потенциалом применительно к использованию архитектур, более тщательно настроенных для выполнения сложных задач. Одним из улучшений стал дифференциальный нейронный компьютер [159], который использовался для сложных задач, требующих вывода умозаключений при обработке графов и естественного языка. Такие задачи трудно реализовать с помощью традиционной нейронной сети.

10.3.3. Дифференциальный нейронный компьютер: краткий обзор

Дифференциальный нейронный компьютер — это усиление нейронных машин Тьюринга за счет использования дополнительных структур, управляющих выделением памяти и отслеживанием временных последовательностей записей. Эти улучшения призваны устранить следующие два основных слабых места нейронных машин Тьюринга.

1. Несмотря на способность нейронной машины Тьюринга адресовать память как по содержимому, так и по расположению, не существует способа, позволяющего избежать того факта, что она осуществляет запись в перекрывающиеся блоки памяти, если использует механизмы сдвига для адресации смежных блоков расположений. В современных компьютерах эта проблема решается за счет подходящего распределения памяти во время выполнения. Дифференциальный нейронный компьютер встраивает механизмы распределения памяти в архитектуру.
2. Нейронная машина Тьюринга не отслеживает порядок, в котором осуществляется запись в память, хотя такая возможность может пригодиться во многих случаях, например при отслеживании последовательности инструкций.

Ниже приведено лишь краткое обсуждение реализации этих двух дополнительных механизмов. За более подробной информацией обратитесь к [159].

Механизм распределения памяти в дифференциальном нейронном компьютере основан на предположениях о том, что 1) расположения, в которые информация была записана, но пока что не прочитана, еще могут быть полезными, и что 2) чтение информации из расположения снижает его полезность. Механизм распределения памяти отслеживает величину, которая называется *показателем использования расположения*. Этот показатель автоматически увеличивается при каждой записи в данное расположение и, возможно, уменьшается при чтении. Прежде чем осуществлять запись в память, контроллер выдает набор свободных вентилях каждой головки чтения, которые определяют, следует ли освободить расположения, из которых недавно осуществлялось чтение. Далее эта информация используется для обновления вектора использования из предыдущего временного шага. В [159] обсуждается ряд алгоритмов, которые могут быть задействованы для идентификации расположений, предназначенных для записи, на основании показателей использования.

Вторая проблема, которую призван разрешить дифференциальный нейронный компьютер, касается отслеживания очередности расположений в памяти, в которые выполняется запись. Здесь важно понимать, что операции записи в

расположения памяти являются мягкими, что делает невозможным строгое упорядочение. Вместо этого между всеми парами расположений существует мягкое упорядочение. Поэтому поддерживается матрица временных связей размера $N \times N$ с элементами $L_t[i, j]$. Значение $L_t[i, j]$ всегда находится в пределах интервала $(0, 1)$ и определяет степень заполнения записями строки i матрицы памяти размером $N \times t$ вплоть до расположения, непосредственно следующего за строкой j . В целях обновления матрицы временных связей для расположений в строках памяти определяются приоритеты весов. В частности, величина $p_t(i)$ определяет степень, до которой в последний раз было заполнено записями расположение i на t -м временном шаге. Эти соотношения старшинства используются для обновления матрицы временных связей на каждом временном шаге. Несмотря на то что теоретически для матрицы временных связей требуется пространство $O(N^2)$, она очень разрежена и поэтому может храниться в пространстве $O(N \cdot \log(N))$. Подробно о том, каким образом поддерживается матрица временных связей, см. в [159].

Следует отметить, что многие из идей нейронных машин Тьюринга, сетей с памятью и механизмов внимания тесно взаимосвязаны. Первые две идеи были независимо предложены примерно в одно и то же время. В начальных статьях на эту тему описывались результаты тестирования на различных задачах. Например, нейронная машина Тьюринга тестировалась на таких простых задачах, как копирование или сортировка, тогда как сети с памятью тестировались на системах “вопрос — ответ”. Однако впоследствии, когда на системах “вопрос — ответ” были протестированы дифференциальные нейронные компьютеры, эти различия были размыты. Образно говоря, все эти приложения пока что переживают “младенческий возраст”, и еще многое предстоит сделать, прежде чем они достигнут уровня коммерческого использования.

10.4. Генеративно-сопоставительные сети

Прежде чем перейти к рассмотрению *генеративно-сопоставительных сетей* (generative adversarial network — GAN), обсудим понятия *генеративной* и *дискриминационной моделей*, поскольку они используются для создания таких сетей.

1. *Дискриминационные модели.* Непосредственно оценивают условную вероятность $P(y | \bar{X})$ метки y при заданных значениях признаков \bar{X} . Примером дискриминационной модели может служить логистическая регрессия.
2. *Генеративные модели.* Оценивают совместную вероятность $P(\bar{X}, y)$, которая является порождающей вероятностью примера данных. Совместную вероятность можно использовать для оценки условной вероятности y при заданном \bar{X} с помощью правила Байеса:

$$(y | \bar{X}) = \frac{P(\bar{X}, y)}{P(\bar{X})} = \frac{P(\bar{X}, y)}{\sum_z P(\bar{X}, z)}. \quad (10.18)$$

Примером генеративной модели может служить наивный байесовский классификатор.

Дискриминационные модели могут применяться только в условиях обучения с учителем, в то время как генеративные модели используются как в обучении с учителем, так и в обучении без учителя. Например, в случае многоклассовой постановки задачи можно создать генеративную модель только для одного из классов, определив для него подходящее априорное распределение, а затем генерировать примеры этого класса путем их семплирования из полученного распределения. Аналогичным образом можно генерировать каждую точку во всем наборе данных из конкретного распределения, используя вероятностную модель с конкретным априорным распределением. Такой подход применяется в вариационных автокодировщиках (см. раздел 4.10.4) для семплирования точек из гауссовского распределения (рассматриваемого в качестве априорного) с последующим использованием этих выборок в качестве входа декодировщика для генерирования образцов, похожих на данные.

Генеративно-сопоставительные сети работают одновременно с двумя моделями нейронных сетей. Одна из них — генеративная модель, создающая синтетические примеры объектов, похожих на реальный репозиторий примеров. При этом ставится задача сделать искусственно создаваемые объекты настолько реалистичными, чтобы обучаемый наблюдатель не смог определить, принадлежит ли конкретный объект к первоначальному набору данных или же был сгенерирован синтетически. Например, если имеется репозиторий изображений автомобилей, то генеративная сеть будет создавать синтетические примеры изображений автомобилей, используя генеративную модель. В результате мы будем иметь набор как реальных, так и поддельных изображений автомобилей. Вторая сеть — дискриминационная — обучается на наборе данных с метками, указывающими на то, является ли данное изображение реальным или поддельным. Дискриминационная модель принимает на входе либо реальные примеры из базы данных, либо синтетические объекты, созданные с помощью генеративной сети, и пытается различить, какие объекты реальные, а какие — поддельные. В определенном смысле генеративную сеть можно рассматривать в качестве “фальшивомонетчика”, пытающегося подделывать деньги, а дискриминационную сеть — в качестве “полицейского”, который пытается уличить фальшивомонетчика в подделке банкнот. Таким образом, эти две сети состязаются между собой, и тренировка улучшает сети до тех пор, пока между ними не установится равновесие. Как будет показано далее, такой состязательный подход к тренировке сводится к формулированию задачи минимакса.

Если дискриминационная сеть способна правильно распознать синтетический объект как поддельный, то генеративная сеть использует этот факт для изменения своих весов, чтобы затруднить классификацию синтезируемых ею образцов дискриминационной сетью. После изменения весов сети-генератора она генерирует новые образцы, и весь процесс повторяется. Со временем генеративная сеть учится все лучше и лучше производить подделки. В конечном счете сеть-дискриминатор не сможет отличать реальные объекты от тех, которые генерируются синтетически. В действительности можно привести формальное доказательство того, что равновесию Нэша в этой минимаксной игре соответствует такая установка параметров (генератора), при которой распределение точек, созданное генератором, совпадает с распределением образцов данных. Для хорошей работы такого подхода важно, чтобы дискриминатор представлял собой модель с высокой емкостью и имел доступ к большому объему данных.

Генерируемые объекты часто используются при создании больших объемов синтетических данных для алгоритмов машинного обучения и могут быть полезными при аугментации данных. Кроме того, добавление контекста позволяет задействовать этот подход для генерирования объектов с различными свойствами. В частности, входом может быть текстовая подпись, например “пятнистый кот с ошейником”, а выходом — фантазийное изображение, соответствующее этому описанию [331, 392]. Генерируемые объекты иногда используются в художественных целях. Недавно подобные методы нашли применение в приложениях, преобразующих изображения в изображения. При таком преобразовании недостающие характеристики изображения оформляются реалистическим способом. Прежде чем перейти к обсуждению приложений, сначала подробно обсудим процесс обучения генеративно-сопоставительной сети.

10.4.1. Тренировка генеративно-сопоставительной сети

Процесс тренировки генеративно-сопоставительной сети осуществляется путем чередования обновления параметров генератора и дискриминатора. Как генератор, так и дискриминатор являются нейронными сетями. Дискриминатор — это нейронная сеть с d -мерными входами и одиночным выходом в интервале $(0, 1)$, представляющим вероятность того, является ли d -мерный входной пример реальным. Значение 1 указывает на то, что пример реальный, а нулевое значение — на то, что пример синтетический. Обозначим выход дискриминатора для входа \bar{X} через $D(\bar{X})$.

Генератор принимает в качестве входа зашумленные образцы из p -мерного распределения вероятности и использует их для генерирования d -мерных примеров данных. Генератор можно считать аналогичным декодировщику, входящему в состав вариационного автокодировщика (см. раздел 4.10.4), в котором входным распределением является p -мерная точка, извлеченная из гауссовского

распределения (являющегося *априорным* распределением), а выходом декодирующего — d -мерная точка данных с аналогичным распределением в качестве реальных примеров. Однако в данном случае процесс тренировки очень отличается от такового в вариационном автокодировщике. Для обучения генератора созданию других образов, похожих на распределение входных данных, вместо ошибки реконструкции используется ошибка дискриминатора.

Задача дискриминатора — корректная классификация реальных примеров с присвоением им метки 1 и синтетически сгенерированных примеров с присвоением им метки 0. С другой стороны, задачей генератора является генерирование примеров, способных обмануть дискриминатор (т.е. чтобы он присвоил им метку 1). Обозначим m реальных примеров, случайно семплированных из набора реальных данных, через R_m , а m синтетических примеров, сгенерированных с использованием генератора, — через S_m . Отметим, что синтетические примеры генерируются путем первоначального создания набора N_m p -мерных зашумленных образцов $\{\bar{Z}_1 \dots \bar{Z}_m\}$ и последующего применения генератора к этим образцам в качестве входа для создания образцов данных $S_m = \{G(\bar{Z}_1) \dots G(\bar{Z}_m)\}$. Поэтому для целевой функции дискриминатора J_D формулируется следующая задача максимизации:

$$\text{Максимизировать } J_D = \underbrace{\sum_{\bar{X} \in R_m} \log[D(\bar{X})]}_{m \text{ образцов реальных примеров}} + \underbrace{\sum_{\bar{X} \in S_m} \log[1 - D(\bar{X})]}_{m \text{ образцов синтетических примеров}}.$$

Нетрудно убедиться в том, что целевая функция достигает максимального значения, когда реальные примеры корректно классифицируются с меткой 1, а синтетические примеры корректно помечаются меткой 0.

Далее мы определяем целевую функцию генератора, задача которого — обмануть дискриминатор. В случае генератора мы не нуждаемся в реальных примерах, поскольку он сам генерирует образцы. Генератор создает m синтетических образцов S_m , и его задача — добиться того, чтобы дискриминатор распознал эти образцы как подлинные. Поэтому целевая функция генератора J_G минимизирует вероятность того, что эти образцы будут помечены как синтетические, что приводит нас к следующей задаче минимизации:

$$\begin{aligned} \text{Минимизировать } J_D &= \underbrace{\sum_{\bar{X} \in S_m} \log[1 - D(\bar{X})]}_{m \text{ образцов синтетических примеров}} = \\ &= \sum_{\bar{Z} \in N_m} \log[1 - D(G(\bar{Z}))]. \end{aligned}$$

Целевая функция достигает минимального значения, когда синтетические примеры некорректно помечаются меткой 1. Минимизируя целевую функцию, мы, по сути, пытаемся обучить параметры генератора тому, чтобы ввести

дискриминатор в заблуждение и заставить его классифицировать синтетические примеры как образцы из реального набора данных. Альтернативным вариантом целевой функции генератора является максимизация $\log[D(\bar{X})]$ для каждого $\bar{X} \in S_m$, а не минимизация $\log[1 - D(\bar{X})]$, и иногда эта альтернативная целевая функция работает лучше на ранних итерациях оптимизации.

Поэтому общая задача оптимизации формулируется в виде минимаксной игры с целевой функцией J_D . Следует заметить, что максимизация J_G путем варьирования параметров генератора G — это то же самое, что максимизация J_D , поскольку $J_D - J_G$ не включает никаких параметров генератора G . Таким образом, общая задача оптимизации (по параметрам как генератора, так и дискриминатора) формулируется в следующем виде:

$$\text{Минимизация}_G \text{ Максимизация}_D J_D. \quad (10.19)$$

Результатом такой оптимизации является *седловая точка*. Примеры того, как выглядят седловые точки на фоне топологии функции потерь, были приведены¹ на рис. 3.17.

Для обучения параметров дискриминатора используется *стохастический градиентный подъем*, а для обучения параметров генератора — *стохастический градиентный спуск*. Шаги градиентного обновления параметров генератора и дискриминатора чередуются. Однако на практике на каждый шаг обновления генератора приходится k шагов обновления дискриминатора. Поэтому шаги градиентного обновления можно описать следующим образом.

1. **(Повторить k раз).** Создается мини-пакет размером $2m$ с равным количеством реальных и синтетических примеров. Синтетические примеры создаются путем подачи на вход генератора зашумленных образцов, семплируемых из априорного распределения, тогда как реальные примеры выбираются из базового набора данных. К параметрам дискриминатора применяется метод стохастического градиентного подъема, максимизирующий вероятность того, что дискриминатор корректно классифицирует как реальные, так и синтетические примеры. На каждом шаге обновления это достигается за счет выполнения в сети дискриминатора обратного распространения ошибки по отношению к мини-пакету, содержащему $2m$ реальных/синтетических примеров.
2. **(Выполнить однократно).** К выходу генератора подключается дискриминатор (рис. 10.7). Генератору предоставляются m зашумленных входов для создания m синтетических примеров (образующих текущий

¹ Приведенные в главе 3 примеры относятся к различным контекстам. Тем не менее, если допустить, что функция потерь на рис. 3.17, б, представляет J_D , то аннотированные седловые точки на этом рисунке становятся визуальными информативными.

мини-пакет). Для минимизации вероятности того, что дискриминатор корректно классифицирует синтетические примеры, выполняется стохастический градиентный спуск по параметрам генератора. Минимизация величины $\log[1 - D(\bar{X})]$ в функции потерь явным образом поощряет предсказание поддельных примеров как реальных.

Несмотря на то что к генератору подключается дискриминатор, градиентные обновления (в процессе обратного распространения ошибки) выполняются лишь по отношению к параметрам сети генератора. Механизм обратного распространения ошибки автоматически вычисляет градиенты для сетей как генератора, так и дискриминатора этой объединенной конфигурации, но обновляются лишь параметры сети генератора.



Рис. 10.7. Объединение конфигураций генератора и дискриминатора для обновления параметров генератора методом градиентного спуска

Значение k обычно невелико (менее 5), хотя допускается использование значения $k = 1$. Этот итеративный процесс повторяется до сходимости, пока не будет достигнуто равновесие Нэша. В этой точке дискриминатор уже не сможет отличить синтетические примеры от реальных.

Существует ряд факторов, на которые следует обращать внимание при выполнении тренировки. Если генератор тренируется слишком долго без обновления дискриминатора, то, во-первых, это может привести к ситуации, когда генератор будет многократно создавать очень похожие образцы. Иными словами, между примерами, производимыми генератором, будет существовать лишь небольшой разброс. Именно это и есть причиной одновременной тренировки поочередно генератора и дискриминатора. Во-вторых, генератор будет создавать плохие примеры на ранних итерациях, и поэтому величина $D(\bar{X})$ будет близка к нулю. В результате функция потерь также будет близка к нулю, а ее градиент будет довольно ограниченным. Такой тип насыщения приведет к замедлению процесса тренировки параметров генератора. В подобных случаях на ранних

стадиях тренировки параметров генератора имеет смысл использовать не минимизацию $\log[1 - D(\bar{X})]$, а максимизацию $D(\bar{X})$. Несмотря на эвристическую мотивацию этого подхода и невозможность записать для него минимаксную формулировку наподобие определения 10.19, он хорошо зарекомендовал себя на практике (особенно на ранних стадиях тренировки, когда дискриминатор отвергает все образцы).

10.4.2. Сравнение с вариационным автокодировщиком

Вариационный автокодировщик и генеративно-сопоставительная сеть были независимо разработаны примерно в одно и то же время. Между ними можно провести много интересных параллелей и различий. Данный раздел посвящен сравнению этих двух моделей.

В отличие от вариационного автокодировщика, в процессе обучения генеративно-сопоставительной сети участвует только декодировщик (т.е. генератор), но не кодировщик. Поэтому генеративно-сопоставительная сеть не предназначена для реконструкции конкретных входных образцов, как это делают вариационные кодировщики. Однако обе модели могут генерировать изображения, такие как базовые данные, поскольку скрытое состояние имеет известную структуру (обычно гауссиан), из которой могут семплироваться точки. В общем случае генеративно-сопоставительная сеть создает образцы более высокого качества (т.е. менее размытые изображения), чем вариационный автокодировщик. Это обусловлено тем, что сопоставительный подход предназначен для того, чтобы создавать реалистические изображения, тогда как вариационный автокодировщик фактически ухудшает качество создаваемых объектов. Кроме того, если выход для конкретного изображения в вариационном автокодировщике создается с использованием ошибки реконструкции, то это вынуждает модель прибегать к усреднению по различным приемлемым выходам, которые часто лишь незначительно сдвинуты относительно друг друга, что является непосредственной причиной размытия. С другой стороны, метод, изначально предназначенный для создания объектов, способных обмануть дискриминатор, будет создавать одиночный объект, между различными частями которого существует гармония (и поэтому такой объект выглядит более реалистично).

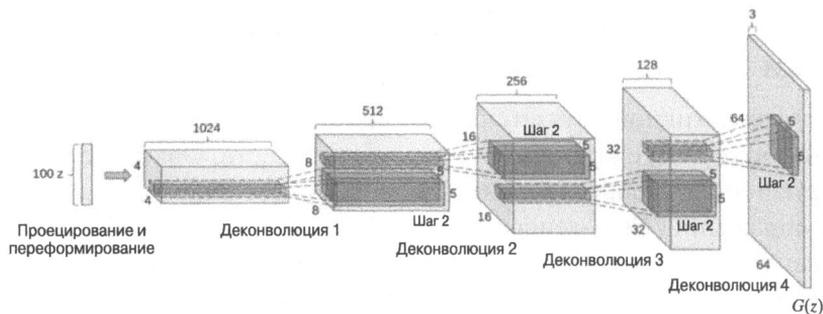
С точки зрения методологии между вариационным автокодировщиком и генеративно-сопоставительной сетью существуют заметные отличия. Применяемый в вариационном автокодировщике подход, основанный на повторной параметризации, полезен для сетей стохастической природы. Существует принципиальная возможность использования такого подхода в условиях нейронных сетей другого типа с порождающим скрытым слоем. В последние годы некоторые из идей вариационного автокодировщика были объединены с идеями генеративно-сопоставительных сетей.

10.4.3. Использование GAN для генерирования данных изображений

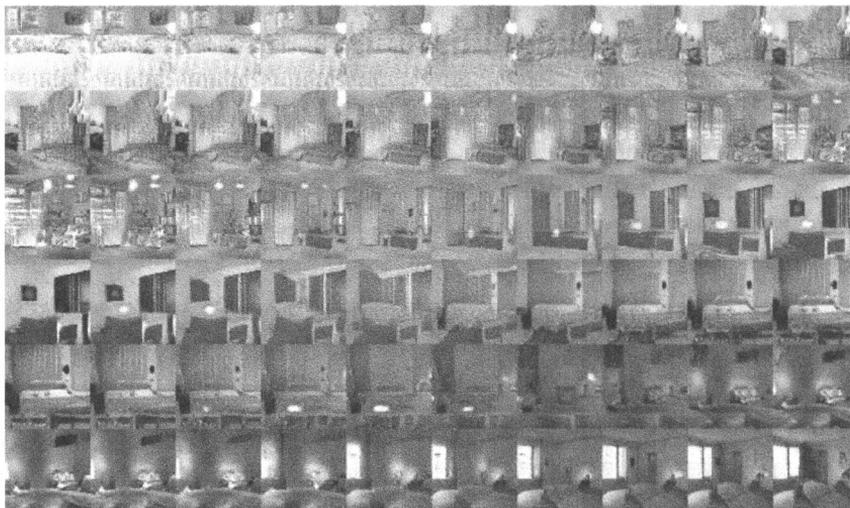
Обычно генеративно-состязательные сети используют для генерирования объектов изображений с различными типами контекста. Фактически в настоящее время наиболее распространенной сферой применения GAN является обработка изображений. Сеть-генератор, предназначенную для работы в этих условиях, называют *сетью обращения свертки*, или *деконволюционной сетью* (deconvolutional network). Наиболее популярные способы создания деконволюционных сетей для GAN обсуждаются в [384]. Поэтому соответствующие генеративно-состязательные сети также называют *сетями DCGAN*. Необходимо отметить, что в последние годы вместо термина “деконволюция” обычно используют термин “транспонированная свертка”, поскольку первый из них может вводить в заблуждение.

В [384] начальной точкой для декодера служит 100-мерный гауссовский шум, который затем преобразуется в 1024 карты признаков размером 4×4 . Это достигается за счет перемножения полносвязной матрицы и 100-мерного входа, а результат представляется в виде тензора. Впоследствии глубина каждого слоя уменьшается с коэффициентом 2 при одновременном увеличении длины и ширины с коэффициентом 2. Например, второй слой содержит 512 карт признаков, третий слой — 256 и т.д.

В то же время увеличение длины и ширины в процессе свертки кажется странным, поскольку даже свертка с шагом 1 уменьшает размеры пространственной карты (если только не используется дополнение нулями). Тогда каким же образом удастся задействовать свертки для увеличения ширины и длины в два раза? Это достигается за счет *дробной шаговой свертки* (fractionally strided convolution) или *транспонированной свертки* (transposed convolution) с дробным значением 0,5. Эти типы транспонированных сверток были описаны в разделе 8.5.2. Случай дробных шагов не слишком отличается от случая единичных шагов, и его можно концептуально рассматривать как свертку, выполняемую после растяжения пространственного объема либо за счет вставки нулей между строками/столбцами, либо за счет вставки интерполированных значений. Поскольку входной объем уже растянут с использованием некоего множителя, то применение свертки с шагом 1 к такому входу эквивалентно применению дробных шагов к первоначальному объему. Альтернативой подходу, основанному на дробной шаговой свертке, является использование пулинга или анпулинга (повышающей дискретизации) для манипулирования пространственными картами. В случае дробной шаговой свертки необходимость в пулинге или анпулинге отпадает. Общая архитектура генератора в DCGAN приведена на рис. 10.8. Подробное обсуждение сверточной арифметики, применяемой в дробной шаговой свертке, содержится в [109].



а) Сверточная архитектура DCGAN



б) В каждом ряду представлены сглаженные переходы изображений, обусловленные изменением входного шума



в) Арифметические операции над входным шумом обладают семантической значимостью

Рис. 10.8. Сверточная архитектура DCGAN и генерируемые изображения. Эти рисунки приведены в [384] и используются с разрешения авторов

Сгенерированные изображения чувствительны к зашумленным образцам. На рис. 10.8, б, приведены примеры изображений, сгенерированных с использованием различных образцов шума. Интересный пример представлен в шестом ряду, где комната без окон постепенно превращается в комнату с большим окном [384]. Такие же плавные переходы наблюдаются и в случае вариационного автокодировщика. К зашумленным образцам также применима векторная арифметика, допускающая семантическую интерпретацию. Например, можно вычесть зашумленный образец женщины с нейтральным выражением лица из изображения улыбающейся женщины и добавить зашумленный образец улыбающегося мужчины. Этот зашумленный образец передается в качестве входа генератору для получения образца изображения улыбающегося мужчины. Данный пример [384] представлен на рис. 10.8, в.

В дискриминаторе также применяется архитектура сверточной нейронной сети, за исключением того, что вместо ReLU используется ReLU с уткой. Последний сверточный слой дискриминатора уплощается и подается на одиночный сигмоидный выход. Полносвязные слои не используются ни в генераторе, ни в дискриминаторе. Как это общепринято в нейронных сетях, применяется ReLU-активация. Для устранения возможных проблем исчезающих и взрывных градиентов задействуется пакетная нормализация [214].

10.4.4. Условные генеративно-сопоставительные сети

В *условных генеративно-сопоставительных сетях* (conditional adversarial generative network — CGAN) как генератор, так и дискриминатор подстраиваются (кондиционируются) под дополнительный входной объект, который может быть меткой, подписью или даже другим объектом того же типа. В таком случае входу в типичных случаях соответствуют *ассоциированные пары целевых объектов и контекстов*. Как правило, контексты связаны с целевыми объектами специфическим для данной предметной области способом, которому обучается модель. Например, такой контекст, как “улыбающаяся девушка”, может представлять изображение улыбающейся девушки. Здесь важно подчеркнуть, что существует множество возможных изображений, которые могут быть созданы в CGAN для представления улыбающейся девушки, и выбор конкретного изображения зависит от значения зашумленного входа. Поэтому CGAN может создать большой набор целевых объектов, в зависимости от креативности сети. В общем случае, если контекст сложнее целевого выхода, то набор сократится, и может даже случиться так, что генератор выведет фиксированные объекты независимо от передаваемого генератору зашумленного входа. Поэтому, как правило, контекстуальные входы более просты по сравнению с моделируемыми объектами. Например, обычно контекстом является подпись, а объектом — изображение, а не наоборот. Тем не менее технически возможны обе ситуации.

Примеры различных типов отмеченного выше подстраивания под вход в CGAN приведены на рис. 10.9. Контекст предоставляет дополнительный вход, необходимый для такого подстраивания. Вообще говоря, контекстом может быть объект любого типа, а сгенерированным выходом — объект любого другого типа. Большой интерес представляют те случаи CGAN, когда контекст (например, подпись) отличается меньшей сложностью по сравнению с генерируемым выходом (например, изображением). В подобных случаях CGAN демонстрируют определенный уровень креативности в заполнении отсутствующих деталей. Эти детали могут меняться в зависимости от зашумленного входа генератора. Некоторые примеры пар “объект — контекст” описаны ниже.

1. Каждый объект может ассоциироваться с меткой. Метка предоставляет условия подстройки для генерирования изображений. Например, в случае набора данных MNIST (см. главу 1) кондиционирующим фактором является значение метки в интервале от 0 до 9, и ожидается, что при таком кондиционировании генератор будет создавать изображение, помеченное предоставленной цифрой. Точно так же в случае набора изображений фактором кондиционирования может быть метка наподобие “морковь”, а выходом — изображение морковки. В экспериментах, описанных в оригинальной работе по условным состязательным сетям [331], генерировалось 784-мерное представление цифры на основе метки в интервале от 0 до 9. Базовые примеры цифр взяты из набора данных MNIST (см. раздел 1.8.1).
2. Целевой объект и контекст могут быть объектами одного и того же типа, хотя контекст может не быть столь богат деталями, как целевой объект. Например, контекстом может служить нарисованный художником эскиз дамской сумочки, а целевым объектом — фактическая фотография той же сумочки со всеми деталями. В качестве другого примера можно привести фоторобот преступника (являющийся контекстом), тогда как целевым объектом (выходом генератора) может быть экстраполяция фактической фотографии человека. Задача заключается в том, чтобы использовать заданный эскиз для генерирования различных реалистических образцов со всеми деталями. Этот пример приведен на рис. 10.9, *вверху*. В случае объектов контекста, имеющих сложные представления, например в виде изображений или текста, может потребоваться их преобразование в многомерное представление с помощью кодировщика, чтобы их можно было совместить с многомерным гауссовским шумом. Таким кодировщиком может быть сверточная нейронная сеть в случае графического контекста или рекуррентная нейронная сеть в случае текстового контекста.
3. Каждый объект может быть ассоциирован с текстовым описанием (пример — изображение и подпись к нему), которое предоставляет контекст.

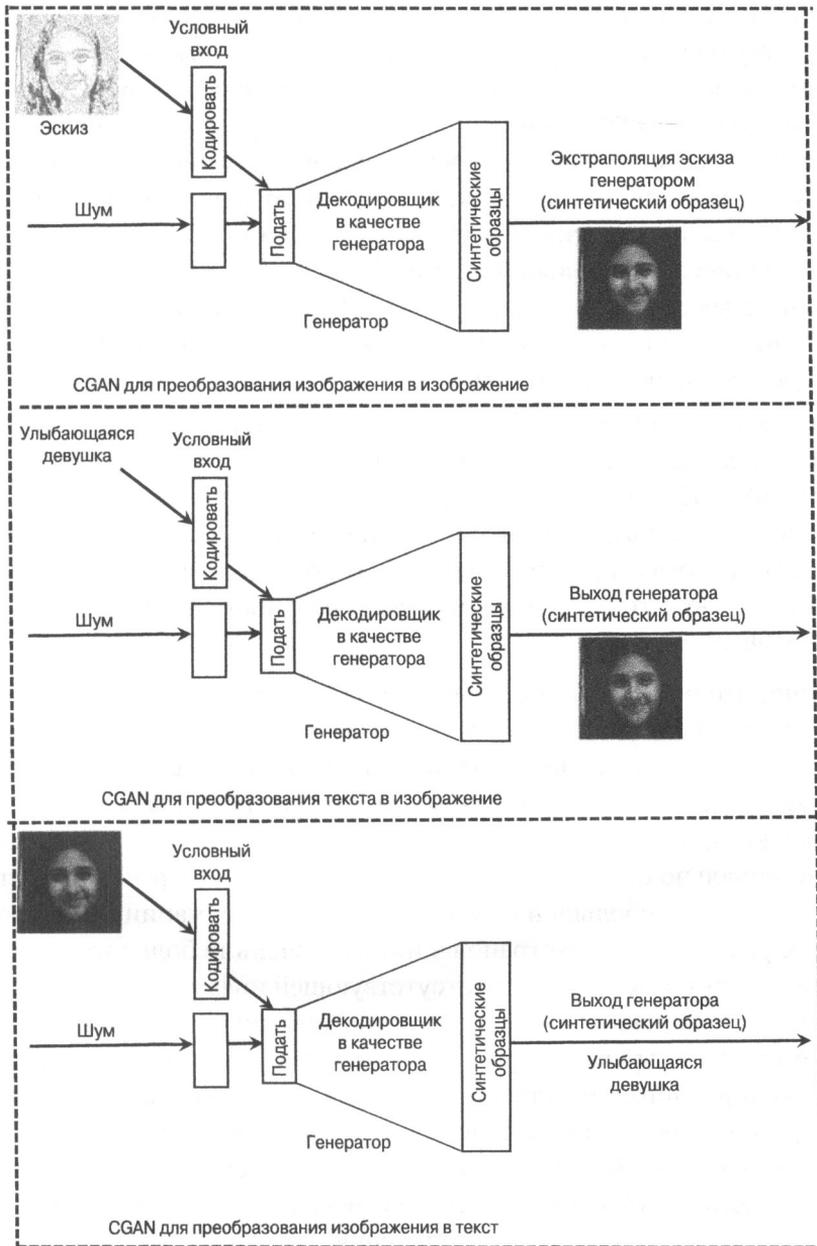


Рис. 10.9. Различные типы условных генераторов для состязательной сети. Эти примеры носят чисто иллюстративный характер и не отражают фактический выход CGAN

Подпись обеспечивает подстройку под данный объект. Идея заключается в том, что генератор, которому предоставлен контекст наподобие “синяя

птица с острыми когтями”, должен создать фантазийное изображение, соответствующее данному описанию. Иллюстративный пример, сгенерированный с использованием контекста “улыбающаяся девушка”, приведен на рис. 10.9. Обратите внимание на то, что допускается использовать изображение в качестве контекста и генерировать подпись к нему с помощью GAN, как показано на рис. 10.9, *внизу*. Однако чаще генерируют сложные объекты (например, изображения) на основе простых контекстов (например, подписей), а не наоборот. Это объясняется тем, что для генерирования простых объектов (таких, как метки или подписи) на основе сложных объектов (например, изображений) существует целый ряд более точных методов обучения с учителем.

4. Базовым объектом может быть черно-белая фотография или видео (например, старый фильм), а выходом — цветная версия данного объекта. По сути, GAN обучается на примерах таких пар способом наиболее реалистичного окрашивания черно-белых сцен. Например, сеть будет использовать цвета деревьев на предоставленных тренировочных фотографиях для соответствующего окрашивания сгенерированного объекта без изменения его базовых контуров.

Очевидно, что во всех этих случаях генеративно-состязательные сети способны очень хорошо справляться с *восполнением отсутствующей информации*. CGAN является специальным случаем, в котором контекст вообще не предоставлен в какой-либо форме, и поэтому она вынуждена создавать изображение, не опираясь ни на какую информацию. С прикладной точки зрения условный случай потенциально более интересен, поскольку часто встречаются ситуации, когда имеется лишь небольшой объем частичной информации, что вынуждает прибегать к реалистичной экстраполяции. При очень небольших объемах доступного контекста методы анализа отсутствующей информации не будут работать, поскольку для реконструкции такой информации требуется значительно больше контекста. С другой стороны, от генеративно-состязательных (в отличие от автокодировщиков и методов матричной факторизации) нельзя ожидать надежной реконструкции данных, но они обеспечивают реалистичную экстраполяцию, при которой объект гармонично дополняется отсутствующими деталями. Как следствие, GAN использует эту свободу для генерирования образцов высокого качества, а не просто для получения размытой оценки усредненного представления реконструированного объекта. И хотя такой процесс может не обеспечить идеального отражения заданного контекста, всегда существует возможность сгенерировать несколько образцов для исследования различных типов экстраполяции одного и того же контекста. Например, при наличии фоторобота преступника можно сгенерировать несколько фотографий, отличающихся

детальями, которые отсутствуют в эскизе. В этом смысле генеративно-состязательные сети проявляют определенный уровень артистизма/креативности, чем не могут похвастаться традиционные методы реконструкции данных. Такого типа креативность существенна при работе с небольшими объемами начального контекста, и поэтому модель должна иметь достаточную степень свободы выбора для дополнения объекта отсутствующими деталями разумным способом.

Существует целый ряд задач машинного обучения (включая классификацию), которые можно рассматривать с точки зрения восполнения отсутствующих данных. С технической точки зрения для этих задач можно использовать также CGAN. Однако условные GAN более полезны для специфических типов отсутствующих данных, когда отсутствующая часть достаточно велика для того, чтобы модель могла ее реконструировать. И хотя CGAN можно применять даже для классификации или аннотирования изображений, совершенно очевидно, что это не наилучший способ использования² модели генератора, ориентированного на генеративную креативность. Если объект преобразования сложнее выходного объекта, то CGAN может сгенерировать фиксированный выход независимо от входного шума.

В случае генератора входы соответствуют точке, сгенерированной из зашумленного распределения и преобразуемого объекта, которые объединяются для создания единого скрытого кода. Этот вход передается генератору (декодеру), который создает преобразованный образец для данных. Для дискриминатора входом является образец из базы данных и его контекст. Базовый объект и его условный вход сначала сливаются в единое скрытое представление, после чего дискриминатор классифицирует данное представление как реальное или сгенерированное. Общая архитектура обучения генератора проиллюстрирована на рис. 10.10. Поучительно сравнить эту архитектуру с архитектурой безусловной GAN, приведенной на рис. 10.7. Основным отличием является наличие дополнительного условного входа во втором случае. Функции потерь и общее устройство скрытых слоев в обоих случаях очень близки. Поэтому переход от безусловной GAN к условной требует лишь незначительного изменения общей архитектуры. При этом обратное распространение ошибки не испытывает значительных изменений, за исключением того, что в той части нейронной сети, которая связана с кондиционированием входов, существуют дополнительные веса, которым может потребоваться обновление.

² Оказывается, что путем видоизменения *дискриминатора* для вывода классов (включая поддельные) можно получить классификацию на уровне современных методов, основанных на полуавтоматическом (с частичным привлечением учителя) обучении с очень небольшим количеством меток [420]. В то же время использование *генератора* для вывода меток — не слишком удачная идея.

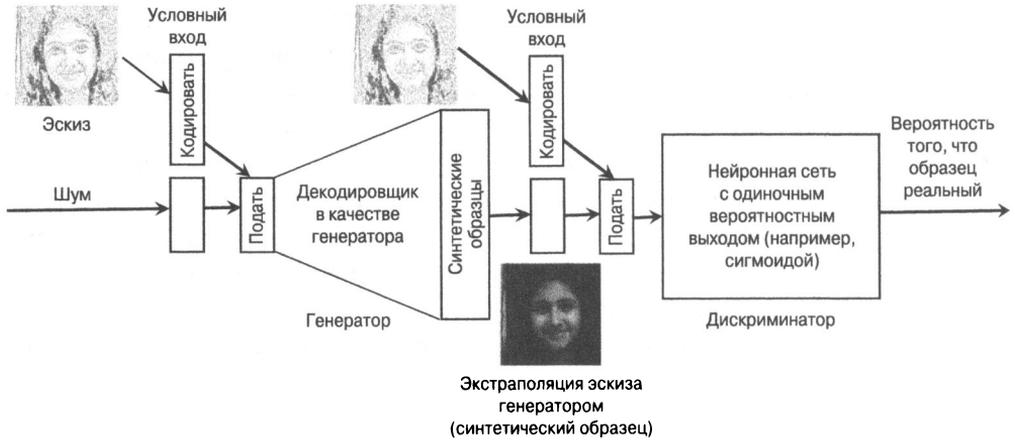


Рис. 10.10. Условная генеративно-сопоставительная сеть для подключенного дискриминатора. Поучительно сравнить эту архитектуру с архитектурой безусловной генеративно-сопоставительной сети, приведенной на рис. 10.7

Важно отметить, что использование GAN с различными типами данных может требовать внесения в сеть изменений для выполнения кодирования и декодирования способом, учитывающим специфику данных. Ранее мы уже приводили примеры, касающиеся обработки изображений и текста, однако описание алгоритма большей частью фокусировалось на простых многомерных данных (а не на изображениях и текстовых данных). Даже если в качестве контекста использовалась метка, требовалось ее кодирование в многомерное представление (например, на основе прямого кодирования). Поэтому на рис. 10.9 и 10.10 обозначены компоненты, предназначенные для кодирования контекста. В самой ранней работе по условным GAN [331] в качестве кодировщика контекста изображения использовалась предварительно обученная сверточная сеть *AlexNet* [255] (без последнего слоя, предсказывающего метку). Сеть *AlexNet* предварительно обучалась на базе данных *ImageNet*. В [331] даже используется мультимодальный вариант, в котором изображение подается на вход вместе с текстовыми аннотациями. Выходом является другой набор текстовых тегов, дополнительно описывающих изображение. В качестве кодировщика для текстовых аннотаций применялась предварительно обученная (скип-грамм) модель *word2vec*. Следует отметить, что в процессе обновления весов генератора (посредством обратного распространения ошибки с выходом за пределы генератора в кодировщике) возможна даже тонкая настройка весов предварительно обученных сетей кодировщика. Это особенно полезно, когда природа набора данных для генерирования объекта в GAN заметно отличается от природы наборов данных, на которых предварительно обучались кодировщики. Однако в оригинальной работе [331] предварительно обученная конфигурация оставалась фиксированной, и тем не менее это не помешало генерировать разумные результаты высокого качества.

Несмотря на то что в приведенном выше конкретном примере для кодирования текста применяется модель *word2vec*, существуют и другие возможные опции. Одна из них — использование рекуррентной нейронной сети, если входом является целое предложение, а не просто слово. Для обработки слов также можно использовать рекуррентную нейронную сеть, работающую на уровне символов. Во всех случаях можно начинать с кодировщика, предварительно обученного соответствующим образом, последующая тонкая настройка которого осуществляется в процессе тренировки CGAN.

10.5. Соревновательное обучение

Большинство методов обучения, которые обсуждались в этой главе, базируется на обновлении весов нейронной сети с целью коррекции ошибок. *Соревновательное обучение* (competitive learning) — это совершенно другая парадигма, в задачи которой не входит коррекция ошибок. Вместо этого нейроны соревнуются за право реагировать на подмножество похожих входных данных и корректируют свои веса, приближаясь к одной или нескольким точкам входных данных. Поэтому процесс обучения также очень отличается от алгоритма обратного распространения ошибки, применяемого в нейронных сетях.

В общих чертах суть тренировки заключается в следующем. Активация выходного нейрона увеличивается с увеличением сходства между вектором весов нейрона и входа. Предполагается, что вектор весов нейрона имеет ту же размерность, что и вход. Обычный подход предполагает использование евклидова расстояния между входом и вектором весов для вычисления активации. Чем меньше это расстояние, тем больше активация. Выходной элемент, имеющий наивысшую активацию для заданного входа, объявляется победителем и перемещается ближе ко входу.

В соответствии со стратегией “победитель получает все” обновляется лишь нейрон-победитель (т.е. нейрон с наибольшей активацией), тогда как остальные нейроны остаются неизменными. В других вариантах парадигмы соревновательного обучения допускается участие других нейронов в обновлении на основе предварительно определенных отношений соседства. Кроме того, доступны также механизмы, которые разрешают одним нейронам подавлять другие. Эти механизмы представляют собой разновидности регуляризации, которые могут быть использованы для обучения представлениям со специфическим типом предопределенной структуры, что может быть полезным в задачах наподобие двухмерной визуализации. Сначала мы обсудим простую версию алгоритма соревновательного обучения, в которой используется подход “победитель получает все”.

Обозначим через \bar{X} входной вектор в d измерениях, а через \bar{W}_i — вектор весов, связанный с i -м нейроном, с тем же количеством измерений. Предположим,

что всего используется m нейронов, где m обычно намного меньше размера набора данных n . В рассматриваемом алгоритме многократно используются описанные ниже действия, заключающиеся в повторном семплировании \bar{X} из входных данных и выполнении соответствующих вычислений.

5. Для каждого i вычисляется евклидово расстояние $\|\bar{W}_i - \bar{X}\|$. Если p -й нейрон имеет наименьшее значение евклидова расстояния, то он объявляется победителем. Обратите внимание на то, что $\|\bar{W}_i - \bar{X}\|$ обрабатывается как значение активации i -го нейрона.
6. Далее p -й нейрон обновляется с помощью следующего правила:

$$\bar{W}_p \leftarrow \bar{W}_p + \alpha (\bar{X} - \bar{W}_p), \quad (10.20)$$

где $\alpha > 0$ — скорость обучения. Обычно значение α намного меньше 1. В некоторых случаях скорость обучения уменьшается по мере продвижения алгоритма.

Суть идеи соревновательного обучения заключается в том, чтобы рассматривать векторы весов как прототипы (наподобие центроидов в методе кластеризации k -средних), а затем перемещать (выигравший) прототип на небольшое расстояние в направлении обучающего примера. Значение α регулирует долю расстояния между точкой и вектором весов, на которое происходит перемещение \bar{W}_p . Заметьте, что метод кластеризации k -средних достигает аналогичных целей, хотя и другим способом. В конце концов, если центроиду назначается точка, то он смещается на небольшое расстояние в направлении обучающего примера в конце итерации. Соревновательное обучение допускает ряд естественных вариаций этого подхода, которые могут использоваться в задачах обучения без учителя, таких как кластеризация и снижение размерности.

10.5.1. Векторная квантизация

Векторная квантизация — простейшая задача соревновательного обучения. В базовую парадигму соревновательного обучения вносятся определенные изменения, обусловленные введением понятия *чувствительности*. С каждым узлом ассоциируется чувствительность $s_i \geq 0$. Использование параметра чувствительности помогает обеспечивать баланс точек между различными кластерами. Базовые шаги векторной квантизации аналогичны шагам, применяемым в алгоритме соревновательного обучения, за исключением различий, обусловленных необходимостью обновления и использования s_i в вычислениях. Значения s_i инициализируются нулями в каждой точке. На каждой итерации значение s_i увеличивается на величину $\gamma > 0$ для узлов, не являющихся победителем, и 0 для узла-победителя. Кроме того, критерием выбора победителя служит наименьшее значение $\|\bar{W}_i - \bar{X}\| - s_i$. Такой подход способен балансировать кластеры,

даже если плотность точек в различных областях меняется в широких пределах. Он гарантирует, что точки в плотных областях будут располагаться близко к одному из векторов весов, в то время как точки в разреженных областях аппроксимируются очень плохо. Подобное поведение характерно для таких приложений, как снижение размерности и сжатие данных. Значение γ регулирует эффект чувствительности. Установка γ в 0 возвращает нас к чисто соревновательному обучению, которое обсуждалось выше.

Наиболее распространенной областью применения векторной квантизации является сжатие данных. В процессе сжатия каждая точка представляется ближайшим к ней вектором весов \bar{W}_i , где i пробегает значения от 1 до m . Заметим, что значение m намного меньше количества точек n в наборе данных. Первым шагом является создание книги кодов, содержащей векторы $\bar{W}_1 \dots \bar{W}_m$, что требует пространства размером $m \cdot d$ для набора данных размерности d . Каждая точка сохраняется в виде значения индекса в интервале от 1 до m , в зависимости от ее ближайшего вектора весов. Однако для хранения каждой точки данных требуется лишь $\log_2(m)$ битов. Поэтому общий требуемый размер пространства данных составляет $m \cdot d + \log_2(m)$, что обычно намного меньше размера исходного пространства, равного $n \cdot d$. Например, для набора данных, содержащего 10 млрд точек в 100 измерениях, требуется пространство размером порядка 4 Тбайт, если для каждого измерения требуется 4 бита. С другой стороны, при квантизации с $m = 106$ размер пространства, требуемого для кодовой книги, составляет менее 0,5 Гбайт, а для каждой точки требуется 20 бит. Таким образом, размер пространства, требуемого для точек (без кодовой книги), составляет менее 3 Гбайт, а общий размер требуемого пространства (включая кодовую книгу) — менее 3,5 Гбайт. Этот тип сжатия — с потерями, а ошибка аппроксимации точки \bar{X} равна $\|\bar{X} - \bar{W}_i\|$. Точки в плотных областях аппроксимируются очень хорошо, тогда как выбросы в разреженных областях — плохо.

10.5.2. Самоорганизующиеся карты Кохонена

Самоорганизующаяся карта Кохонена (Kohonen self-organizing map) — это разновидность парадигмы соревновательного обучения, в которой нейроны подчинены одномерной структуре, подобной строке, или двумерной структуре, подобной решетке. Ради большей общности обсуждения рассмотрим случай двумерной решеточной структуры. Как будет показано далее, такой тип решеточной структуры обеспечивает отображение всех точек на двумерное пространство с целью визуализации. Пример двумерной решеточной структуры из 25 нейронов, расположенных в виде прямоугольной сетки размером 5×5 , приведен на рис. 10.11, а. Гексагональная решетка с тем же количеством нейронов приведена на рис. 10.11, б. Форма решетки влияет на форму двумерных областей, на которые будут отображаться кластеры. Случай одномерной структуры, подобной

строке, описывается аналогичным образом. Идея использования решеточной структуры состоит в том, чтобы значения \bar{W}_i для нейронов в смежных узлах решетки были близкими. Здесь важно ввести отдельные обозначения, позволяющие отличать расстояние $\|\bar{W}_i - \bar{W}_j\|$ от расстояния в решетке. Расстояние между парой соседних нейронов в решетке в точности равно единице. Например, расстояние между нейронами i и j в решеточной структуре на рис. 10.11, а, равно 1, а расстояние между нейронами i и k равно $\sqrt{2^2 + 3^2}$. Векторное расстояние в исходном пространстве (например, $\|\bar{X} - \bar{W}_i\|$ или $\|\bar{W}_i - \bar{W}_j\|$) обозначим как $Dist(\bar{W}_i, \bar{W}_j)$. С другой стороны, расстояние между нейронами i и j вдоль решеточной структуры обозначим как $LDist(i, j)$. Заметьте, что значение $LDist(i, j)$ зависит только от индексов (i, j) и не зависит от значений векторов \bar{W}_i и \bar{W}_j .

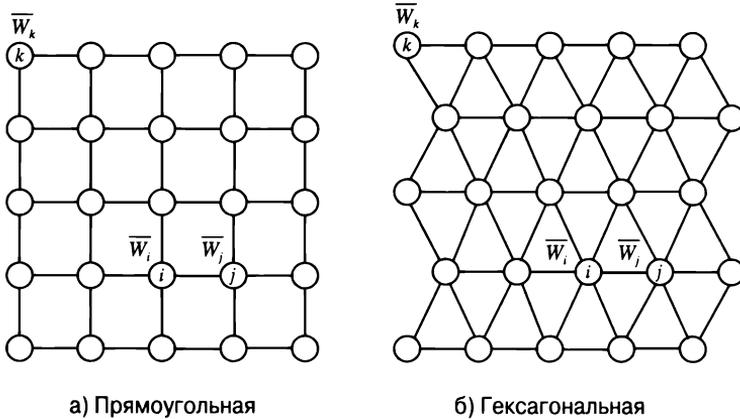


Рис. 10.11. Пример решеточной структуры размером 5×5 для самоорганизующейся карты. Поскольку нейроны i и j расположены близко друг к другу в решетке, процесс обучения будет смещать значения \bar{W}_i и \bar{W}_j в сторону их сближения. Прямоугольная решетка приведет к прямоугольным кластеризованным областям в результирующем двухмерном представлении, а гексагональная — к гексагональным

Процесс обучения в самоорганизующейся карте регулируется так, чтобы близость нейронов i и j (на основании решеточного расстояния) приводила также к сближению их векторов весов. Иными словами, *решеточная структура самоорганизующихся карт действует как регуляризатор в процессе обучения*. Как будет показано далее, наложение двухмерной структуры такого типа на обучаемые веса помогает визуализировать исходные точки данных с помощью двухмерного вложения.

Общий алгоритм обучения самоорганизующейся карты выполняется аналогично алгоритму соревновательного обучения путем семплирования \bar{X} из тренировочных данных и нахождения нейрона-победителя на основании оценок

евклидова расстояния. Веса нейрона-победителя обновляются аналогично тому, как это делается в простом алгоритме соревновательного обучения. Основным отличием является то, что ослабленная версия этого обновления применяется также к решеточным соседям нейрона-победителя. В действительности в мягкой версии данного метода это обновление можно применять ко всем нейронам, привязывая уровень его ослабления к расстоянию данного нейрона от нейрона-победителя в решетке. *Функция ослабления* (damping function), значения которой всегда лежат в пределах $[0, 1]$, обычно определяется в виде гауссовского ядра:

$$Damp(i, j) = \exp\left(-\frac{LDist(i, j)^2}{2\sigma^2}\right), \quad (10.21)$$

где σ — ширина гауссовского ядра. Используя предельно малые значения σ , мы возвращаемся к обучению по принципу “победитель получает все”, тогда как использование больших значений приводит к усилению регуляризации, при которой соседние узлы решетки получают близкие веса. В случае небольших значений σ функция ослабления будет равна 1 только для нейрона-победителя и нулю для других нейронов. Поэтому значение σ является одним из параметров, доступных для пользовательской настройки. Для управления регуляризацией и ослаблением доступны и другие функции ядра. Например, вместо гладкой гауссовской функции ослабления можно использовать пороговое ступенчатое ядро, которое имеет значение 1, если $LDist(i, j) < \sigma$, и нуль в противном случае.

Алгоритм обучения повторно семплирует \bar{X} из тренировочных данных и вычисляет расстояния \bar{X} до каждого веса \bar{W}_i . Вычисляется индекс p нейрона-победителя. Вместо того чтобы применять обновление только к \bar{W}_p (как в стратегии “победитель получает все”), к каждому \bar{W}_i применяется следующее правило обновления:

$$\bar{W}_i \leftarrow \bar{W}_i + \alpha \cdot Damp(i, p) \cdot \|\bar{X} - \bar{W}_i\| \quad \forall i, \quad (10.22)$$

где $\alpha > 0$ — скорость обучения. Обычно скорости обучения α предоставляют возможность уменьшаться со временем. Итерации продолжаются до тех пор, пока не будет достигнута сходимость. Веса, соответствующие близким узлам решетки, будут получать близкие обновления, и поэтому с течением времени их значения будут сближаться. *Таким образом, процесс тренировки вынуждает кластеры соседних узлов решетки иметь близкие точки, что удобно для визуализации.*

Использование обученной карты для 2D-вложений

Самоорганизующуюся карту можно применять для создания двухмерного вложения точек. Для сетки размера $k \times k$ все двухмерные решеточные координаты будут располагаться в квадрате, находящемся в положительном квадранте

с вершинами $(0, 0)$, $(0, k - 1)$, $(k - 1, 0)$ и $(k - 1, k - 1)$. Каждая точка сетки в решетке является вершиной с целочисленными координатами. Простейшим двумерным вложением является представление каждой точки \bar{X} ближайшей к ней точкой сетки (нейроном-победителем). Однако такой подход приведет к перекрыванию представлений точек. Кроме того, можно построить двумерное представление данных, в котором каждая координата является одним из $k \times k$ значений $\{0 \dots k - 1\} \times \{0 \dots k - 1\}$. Именно по этой причине самоорганизующуюся карту иногда называют *дискретизированным* методом снижения размерности. Для устранения неоднозначности, возникающей из-за перекрывания точек, можно привлекать различные эвристики. В случае применения к многомерным данным документов визуальная проверка указывает на то, что документы, относящиеся к определенной теме, часто отображаются на определенные локальные области. К тому же документы, относящиеся к родственным темам (например, политике и выборам), обычно отображаются на смежные области. Иллюстративные примеры того, как располагаются самоорганизующиеся документы, относящиеся к четырем темам, в случае использования прямоугольной и гексагональной решеток, приведены на рис. 10.12, а и б, соответственно. Области раскрашены различными цветами в зависимости от того, какая тема является преобладающей в документах, принадлежащих к данной области.

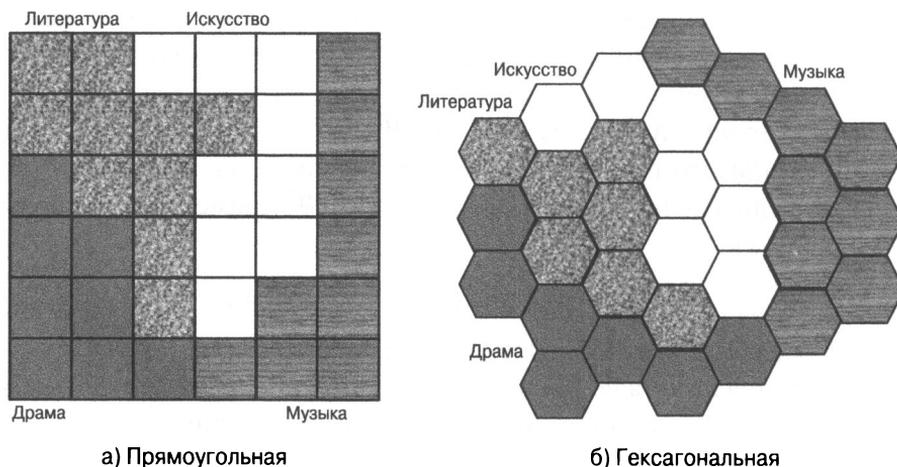


Рис. 10.12. Примеры двумерной визуализации документов, относящихся к четырем темам

Для структуры самоорганизующихся карт находят веские обоснования в нейробиологии. В головном мозге млекопитающих входные сигналы, поступающие от различных органов чувств (например, от органов осязания), отображаются на ряд складчатых клеточных слоев, так называемых *экраных структур* [129]. Если какие-либо близко расположенные части тела подвергаются

воздействию (например, тактильному), то это приводит к возбуждению групп клеток, которые также являются соседними. Поэтому близость (сенсорных) входных воздействий транслируется в близость активизируемых нейронов, как в случае самоорганизующихся карт. Аналогично идее сверточных нейронных весов, возникшей на основе результатов, полученных в нейробиологии, подобные сведения всегда используются для создания тех или иных форм регуляризации.

Несмотря на то что с началом современной эры глубокого обучения сети Кохонена используются не так часто, как прежде, в них скрыты значительные возможности в отношении обучения без учителя. Кроме того, базовая идея соревновательности может быть внедрена даже в многослойных сетях прямого распространения. Многие принципы соревновательности часто комбинируются с более традиционными сетями прямого распространения. Например, как разреженные автокодировщики, так и автокодировщики, действующие по принципу “победитель получает все” (см. раздел 2.5.5.1), базируются на принципах соревновательного обучения. Точно так же понятие нормализации локального отклика (см. раздел 8.2.8) базируется на идеях соревнования между нейронами. Даже идеи фокусирования внимания на подмножестве активаций, которые обсуждались в этой главе, задействуют принципы соревновательности. Поэтому, несмотря на снижение популярности самоорганизующихся карт в последние годы, можно ожидать, что базовые принципы соревновательности все еще будут внедряться в традиционные сети прямого распространения.

10.6. Ограничения нейронных сетей

Глубокое обучение значительно прогрессировало в последние годы, а в некоторых задачах, таких как классификация изображений, даже продемонстрировало свое превосходство над возможностями человека. Такие же поразительные результаты в некоторых играх, требующих последовательного планирования действий, продемонстрировало и обучение с подкреплением. Поэтому весьма соблазнительными кажутся заявления о том, что в конечном счете превосходство искусственного интеллекта над человеком может приобрести всеобщий характер. Однако существует ряд фундаментальных технических барьеров, которые необходимо преодолеть, прежде чем мы сможем создавать машины, способные обучаться и думать как люди [261]. В частности, для получения результатов высокого качества, превосходящих возможности человека, нейронным сетям необходимо предоставлять большие объемы обучающих данных. Кроме того, расходы энергии, необходимой нейронным сетям для решения различных задач, намного превышают потребление энергии человеком, выполняющим аналогичные задачи. Эти факторы налагают серьезные ограничения на способность нейронных сетей демонстрировать превосходство над человеком

по определенным параметрам. Ниже мы обсудим ключевые проблемы, а заодно и некоторые направления последних исследований.

10.6.1. Амбициозная задача: разовое обучение

Несмотря на то что в последние годы глубокое обучение привлекает к себе все большее внимание, что обусловлено его успехами при решении крупномасштабных задач (по сравнению с умеренной производительностью, продемонстрированной в первые годы на небольших объемах данных), в имеющихся на сегодняшний день технологиях глубокого обучения существуют узкие места. В задачах наподобие классификации изображений, при решении которых глубокому обучению удалось превзойти возможности человека, это достигалось *неэффективным с точки зрения формирования выборок способом*. Например, база данных *ImageNet* содержит свыше миллиона изображений, и для того, чтобы сеть могла правильно классифицировать какой-либо класс, ей часто требуется предоставлять тысячи образцов. Человеку, чтобы он научился распознавать, что такое грузовик, не требуется изучать тысячи изображений. Во многих случаях достаточно показать ребенку грузовик всего один раз, и он сможет распознать другой грузовик, даже если тот отличается моделью, формой и цветом. Это свидетельствует о том, что возможности человека в отношении обобщения полученных знаний применительно к новым условиям гораздо выше возможностей искусственных нейронных сетей. Подход, основанный на способности к обучению всего на одном или нескольких примерах, получил название *разовое обучение* (*one-shot learning*).

Способность людей к обобщению, основанному всего лишь на небольшом количестве примеров, не должна удивлять, поскольку сеть нейронов в головном мозге человека относительно разрежена и тщательно спроектирована самой природой. Эта архитектура эволюционировала на протяжении многих миллионов лет и передавалась от поколения к поколению. В некотором смысле можно считать, что в структуре нейронных соединений человека уже закодированы знания, накопленные за миллионы лет благодаря эволюционному опыту. Кроме того, люди пополняют знания в течение всей жизни, решая самые разнообразные задачи, что ускоряет процесс обучения решению любой новой задачи. Впоследствии обучение выполнению специфических задач (таких, как распознавание грузовика) сводится к тонкой настройке кода, либо врожденного, либо приобретенного человеком на протяжении жизни. Иными словами, люди — мастера переносимого обучения как в пределах поколения, так и между поколениями.

Важнейшей областью будущих исследований является разработка обобщенных форм переносимого обучения, обеспечивающих повторное использование знаний, приобретенных в процессе обучения выполнению конкретной задачи,

для решения других задач. Преимущества переносимого обучения в какой-то степени уже были продемонстрированы в глубоком обучении. Как обсуждалось в главе 8, сверточные нейронные сети наподобие *AlexNet* [255] часто подвергаются предварительному обучению на таких больших репозиториях изображений, как *ImageNet*. Впоследствии, когда потребуется применить нейронную сеть к новому набору данных, веса могут быть подвергнуты дополнительной тонкой настройке. Часто при такой настройке можно ограничиться намного меньшим количеством примеров, поскольку большинство основных признаков, обученных в предыдущих слоях, остается неизменным в новом наборе данных. Во многих случаях обученные признаки удается обобщить на ряд задач, удаляя последние слои и заменяя их слоями, учитывающими специфику новой задачи. Этот общий принцип используется и при интеллектуальном анализе текста. Например, многие модели обучения текстовых признаков, такие как *word2vec*, повторно применяются во многих задачах интеллектуального анализа текста, даже если они были обучены на другом корпусе документов. В общем случае перенос знаний может осуществляться посредством извлеченных признаков, параметров модели и другой контекстной информации.

Существует еще одна форма переносимого обучения, которая базируется на понятии так называемого *межзадачного обучения* (*learning across tasks*). Основная идея заключается в повторном использовании результатов той работы, которая уже была выполнена в процессе тренировки, полной или частичной, в рамках одной задачи, для улучшения способности сети обучаться решению других задач. Этот принцип называется *обучение обучению* (*learning-to-learn*). Тран и Платт [497] определили обучение обучению следующим образом. Если заданы семейство задач, опыт обучения каждой задаче и семейство мер производительности (по одной для каждой задачи), то говорят, что алгоритм является *алгоритмом обучения обучению*, если его производительность при выполнении каждой задачи улучшается как с накоплением опыта, так и с увеличением количества задач. Основная проблема обучения обучению коренится в том факте, что все задачи в чем-то отличаются друг от друга, что затрудняет перенос опыта между ними. Поэтому обучение в пределах одной задачи осуществляется быстро, тогда как межзадачное обучение осуществляется под управлением знаний, приобретаемых более плавным способом, позволяющим уловить изменение структуры задачи при переходе от одной целевой области к другой [416]. Иными словами, в этом случае мы имеем дело с двухзвенной организацией обучения задачам. Данный подход также называют *мета-обучением* (*meta-learning*), хотя этот термин несколько перегружен и используется еще в нескольких концепциях машинного обучения. Способность обучаться обучению — уникальное свойство биологических существ, благодаря которому живые организмы функционируют лучше даже при решении задач, слабо связанных с теми задачами, на которых

приобретался опыт. На простейшем уровне даже предварительная тренировка сетей может служить примером обучения обучению, поскольку веса сети, тренировавшейся на определенных наборе данных и задаче, могут использоваться в другой постановке задачи, чтобы обучение в новых условиях происходило быстрее. Например, в сверточной нейронной сети признаки во многих ранних слоях представляют собой примитивные формы (например, отрезки) и сохраняют свою полезность независимо от рода задачи и набора данных, к которым применяются. С другой стороны, последний слой может быть весьма специфическим для конкретной задачи. Однако на тренировку одного слоя уйдет гораздо меньше времени, чем на тренировку всей сети.

В ранней работе по разовому обучению [116] для переноса изученных знаний из одной категории задач в другую использовались байесовские модели. Некоторые успехи в мета-обучении были продемонстрированы с помощью структурированных архитектур, в которых использовались понятия внимания, рекурсии и памяти. В частности, хорошие результаты были получены при обучении на задачах различных категорий с помощью нейронных машин Тьюринга [416]. Способность сетей с памятью к обучению на ограниченных объемах данных была известна давно. Например, даже сеть с внутренней памятью наподобие LSTM продемонстрировала впечатляющие результаты при обучении не предоставлявшимся ей ранее квадратичным функциям на небольшом количестве примеров. В этом отношении нейронная машина Тьюринга — еще более эффективная архитектура, и в [416] показано, как ее можно использовать в целях мета-обучения. Нейронные машины Тьюринга также применялись для создания согласующих сетей, используемых при разовом обучении [507]. Несмотря на то что указанные сети отражают определенный прогресс в области разового обучения, возможности этих методов все еще остаются примитивными по сравнению с возможностями человека. В силу этого данная тема остается областью, открытой для будущих исследований.

10.6.2. Амбициозная задача: энергетически эффективное обучение

С эффективностью семплирования тесно связана энергетическая эффективность. Системы глубокого обучения, работающие на высокопроизводительном оборудовании, неэффективны с точки зрения потребления энергии, необходимой им для выполнения своих функций. Например, для параллельного выполнения интенсивных вычислений с помощью нескольких GPU может потребоваться мощность свыше одного киловатта. В то же время для функционирования человеческого мозга едва ли потребуется мощность, превышающая 20 Вт, что намного меньше мощности обычной лампы накаливания. Другой важный момент заключается в том, что человеческий мозг редко выполняет

точные вычисления и обычно лишь делает оценки. Для многих задач обучения этого оказывается вполне достаточно, а иногда данный фактор даже приводит к повышению обобщающей способности. Это дает основания предположить, что архитектуры, делающие акцент на обобщении, а не на точности, могут демонстрировать неплохую энергоэффективность.

Недавно был разработан ряд алгоритмов, обеспечивающих повышенную энергоэффективность вычислений за счет определенной потери точности. Некоторые из этих методов демонстрируют улучшенную обобщающую способность, обусловленную эффектами шума вычислений с пониженной точностью. В [83] предлагаются методы использования бинарных весов, обеспечивающие эффективное выполнение вычислений. Влияние использования различных представительных кодов на энергоэффективность исследовано в [289]. Известно, что некоторые типы нейронных сетей, содержащие *спайковые*, или *импульсные*, нейроны, отличаются большей энергоэффективностью [60]. Понятие импульсных нейронов непосредственно основано на биологической модели мозга млекопитающих. Базовая идея заключается в том, что нейроны возбуждаются не на каждом цикле распространения, а только тогда, когда мембранный потенциал достигает определенного значения. *Мембранный потенциал* — это внутренняя характеристика нейрона, связанная с его электрическим зарядом.

Энергоэффективность часто достигается в сетях небольшого размера с удаленными избыточными соединениями. Кроме того, удаление избыточных соединений способствует регуляризации. В [169] предлагается обучать веса и соединения нейронной сети одновременно с удалением избыточных соединений. В частности, удалять можно веса, значения которых близки к нулю. Как обсуждалось в главе 4, обучение сети, приводящее к нулевым весам, может достигаться за счет L_1 -регуляризации. В то же время в [169] показано, что L_2 -регуляризация обеспечивает более высокую точность. Поэтому в [169] используется L_2 -регуляризация наряду с удалением весов, значения которых находятся ниже определенного порога. Удаление весов осуществляется итеративным способом, при котором веса повторно обучаются после их удаления, после чего связи с низкими значениями весов вновь удаляются. На каждой итерации обученные веса из предыдущей фазы используются для следующей фазы. Тем самым плотная сеть может быть разрежена до сети с намного меньшим количеством соединений. Кроме того, удаляются также мертвые нейроны с нулевыми входными и выходными соединениями. О возможных дальнейших улучшениях сообщалось в [168], где данный подход был объединен с кодированием Хаффмана и квантизацией для сжатия. Целью квантизации является уменьшение количества битов, представляющих каждое соединение. Применение такого подхода в сети *AlexNet* [255] позволило уменьшить объем требуемой памяти в 35 раз, сократив его с 240 до 6,9 Мбайт без потери точности. Как следствие, модель удалось

разместить во внутрипроцессорной кеш-памяти SRAM вместо внешней памяти DRAM. Преимуществами такого подхода являются скорость, энергоэффективность и возможность выполнения мобильных вычислений на встроженных устройствах. В частности, для достижения этих целей в [168] было использовано аппаратное ускорение, что стало возможным благодаря размещению модели в SRAM-кеше.

Еще одним направлением является разработка оборудования, специально предназначенного для нейронных сетей. Следует отметить, что в человеческом организме не существует различий между “программами” и “оборудованием”. В случае компьютеров разделение этих понятий оправдано, но одновременно является источником неэффективности, не свойственной мозгу человека. Проще говоря, модели вычислений, идеи которых почерпнуты из известных нам экспериментальных фактов о функционировании человеческого мозга, характеризуются тесной интеграцией программ и оборудования. В последние годы значительный прогресс был достигнут в области *нейроморфных вычислений* (neuromorphic computing) [114]. Это понятие базируется на новой архитектуре микрочипов, содержащих импульсные нейроны, синапсы низкой точности и масштабируемую коммуникационную сеть. Более подробное описание архитектуры сверточной нейронной сети (основанной на нейроморфных вычислениях), которая обеспечивает распознавание изображений на уровне, отвечающем требованиям современности, приведено в [114].

10.7. Резюме

Эта глава была посвящена рассмотрению ряда тем глубокого обучения, характеризующихся повышенной сложностью. Мы начали с обсуждения механизмов внимания, которые используются как для обработки изображений, так и текстовых данных. Во всех случаях привлечение механизмов внимания приводило к улучшению обобщающей способности нейронной сети. Механизмы внимания также могут быть задействованы для дополнения сетей внешней памятью. С теоретической точки зрения сеть с памятью обладает в терминах тьюринг-полноты теми же свойствами, что и рекуррентная нейронная сеть. Однако обычно она позволяет выполнять вычисления способом, легче поддающимся интерпретации, и поэтому хорошо обобщается на тестовые наборы данных, отличающиеся в каком-либо отношении от обучающих данных. Например, это обеспечивает более высокую точность классификации при работе с последовательностями, длина которых превышает длину последовательностей, содержащихся в тренировочном наборе данных. Простейшим примером сети с памятью является нейронная машина Тьюринга, последующим обобщением которой стал дифференциальный нейронный компьютер.

Генеративно-состязательные сети — относительно недавняя архитектура, использующая процесс состязательного взаимодействия генеративной (порождающей) и дискриминационной сетей для генерирования синтетических образцов, похожих на реальные примеры из базы данных. Такие сети можно применять в качестве генеративных моделей, создающих входные примеры для тестирования алгоритмов машинного обучения. Кроме того, налагая определенные условия на генеративный процесс, можно создавать образцы с различными типами контекста. Эти идеи находят применение в различных задачах, включая преобразование текста в изображение и преобразование изображения в изображение.

Также были кратко рассмотрены многочисленные дополнительные темы, обсуждаемые в последние годы, такие как одноразовое обучение и энергоэффективное обучение. Эти направления представляют области, в которых технология нейронных сетей еще значительно отстает от возможностей человека. Несмотря на существенный прогресс, достигнутый за последнее время, эти области все еще оставляют широкий простор для будущих исследований.

10.8. Библиографическая справка

Ранние методы использования механизмов внимания для обучения нейронных сетей были предложены в [59, 266]. Рекуррентные модели визуального внимания, которые обсуждались в этой главе, базируются на [338]. Распознавание нескольких объектов на изображении с помощью визуального внимания обсуждается в [15]. Две наиболее известные модели нейронного машинного перевода с использованием механизма внимания рассмотрены в [18, 302]. Идеи механизмов внимания также были расширены на аннотирование изображений. Например, в [540] представлены методы аннотирования изображений, базирующиеся как на мягкой, так и на жесткой модели внимания. Использование моделей внимания для резюмирования текста описано в [413]. Применение механизмов внимания для фокусирования на специфических частях изображения позволяет визуализировать работу систем “вопрос — ответ” [395, 539, 542]. Полезным дополнением механизма внимания является использование пространственных сетей-трансформеров, обеспечивающих селективное кадрирование или фокусирование на отдельных частях изображения. Использование моделей внимания для визуализации систем “вопрос — ответ” описано в [299].

Нейронные машины Тьюринга [158] и сети с памятью [473, 528] были предложены примерно в одно и то же время. Впоследствии нейронная машина Тьюринга была обобщена до уровня дифференциального нейронного компьютера за счет использования улучшенных механизмов распределения памяти и механизмов отслеживания последовательности записей. Нейронная машина Тьюринга

и дифференциальный нейронный компьютер применялись в ряде задач, таких как копирование, ассоциативное повторное обращение, сортировка, опрос графовых баз данных и языковые запросы. С другой стороны, основными областями применения сетей с памятью были понимание естественного языка и системы “вопрос — ответ” [473, 528]. Однако обе эти архитектуры довольно схожи. Основным различием является то, что модель, описанная в [473], фокусируется на механизмах адресации по содержимому, а не по расположению, что уменьшает необходимость в увеличении четкости изображений. Более целенаправленное исследование систем “вопрос — ответ” предоставлено в [257]. В [393] введено понятие нейронного программного интерпретатора, представляющего собой рекуррентно-композиционную сеть, которая обучается представлению и выполнению программ. Большой интерес представляет версия машины Тьюринга, спроектированная с применением обучения с подкреплением [550, 551], которую можно использовать для обучения широкому классу сложных задач. В [551] продемонстрировано обучение простых алгоритмов на примерах. Параллелизация этих методов с помощью GPU обсуждается в [229].

Генеративно-сопоставительные сети (GAN) были предложены в [149], а великолепное руководство по этой теме содержится в [145]. В одном из ранних методов аналогичная архитектура предлагалась для генерирования изображений стульев с помощью сверточных сетей [103]. Улучшенные алгоритмы обучения рассматриваются в [420]. Основные трудности, возникающие в процессе тренировки сопоставительных сетей, связаны с *нестабильностью* и *насыщением*. Теоретическое рассмотрение некоторых из этих проблем наряду с методами их устранения содержится в [11, 12]. GAN на основе понятия энергии предложены в [562], где утверждается, что сети этого типа характеризуются лучшей стабильностью. Идеи сопоставительности также были обобщены на архитектуры автокодировщиков [311]. Генеративно-сопоставительные сети часто применяются для генерации реалистичных изображений с различными свойствами [95, 384]. В таких случаях в качестве генератора используется деконволюционная сеть, в связи с чем результирующие сети называют сетями DCGAN. Идея условных генеративных сетей и их применение для генерирования объектов с контекстом обсуждается в [331, 392]. Недавно такой подход был использован для преобразования изображений в изображения [215, 370, 518]. Несмотря на то что генеративно-сопоставительные сети часто задействуются для обработки изображений, в последнее время область их применения была расширена на последовательности [546]. Использование CGAN для предсказания следующего видеокadra описано в [319].

Ссылки на самые ранние работы по соревновательному обучению можно найти в [410, 411]. В работе Гершо и Грея [136] содержится отличный обзор методов векторной квантизации, которые представляют собой альтернативу методике разреженного кодирования [75]. Самоорганизующаяся карта Кохонена

была введена в [248], а более подробное обсуждение этой концепции тем же автором содержится в [249, 250]. Различные варианты базовой архитектуры, такие как *нейронный газ*, используются для инкрементного обучения [126, 317].

Методы обучения обучению обсуждаются в [497]. В самых ранних методах этой категории применяются байесовские модели [116]. Более поздние модели фокусировались на нейронных машинах Тьюринга различного типа [416, 507]. Методы нулевого обучения предложены в [364, 403, 462]. Для долгосрочного обучения могут быть использованы эволюционные методы [543]. Для повышения энергоэффективности глубокого обучения были предложены многочисленные методы, такие как использование бинарных весов [83, 389], специальные микрочипы [114] и механизмы сжатия [213, 168, 169]. Также были разработаны специализированные методы для сверточных нейронных сетей [68].

10.8.1. Программные ресурсы

Рекуррентная модель для механизма визуального внимания доступна по ссылке [627]. Основанный на пакете MATLAB код механизма внимания для нейронного машинного перевода, который рассматривался в данной главе, доступен по ссылке [628]. Реализации нейронной машины Тьюринга средствами *TensorFlow* доступны по ссылкам [629, 630]. Обе реализации связаны между собой, поскольку подход, применяемый в [630], адаптирует некоторые элементы подхода [629]. В оригинальной реализации используется контроллер LSTM. Реализации в *Keras*, *Lasagne* и *Torch* доступны по ссылкам [631–633]. Некоторые реализации сетей с памятью из *Facebook* доступны по ссылке [634]. Реализация сетей с памятью в *TensorFlow* доступна по ссылке [635]. Реализации сетей с динамической памятью в *Theano* и *Lasagne* доступны по ссылке [636].

Реализация DCGAN в *TensorFlow* доступна по ссылке [637]. Некоторые варианты GAN (и другие, которые обсуждались в данной главе) доступны по ссылке [638]. Реализация GAN в *Keras* доступна по ссылке [639]. Реализации различных типов GAN, включая Wasserstein GAN и вариационный автокодировщик, доступны в [640]. Эти реализации выполняются в средах *PyTorch* и *TensorFlow*. Реализация в *TensorFlow* сети GAN для преобразования текста в изображение доступна по ссылке [641], причем эта реализация построена поверх вышеупомянутой DCGAN, реализованной в [637].

10.9. Упражнения

1. В чем состоят основные отличия подходов, используемых в моделях с мягким и жестким вниманием?
2. Покажите, как бы вы использовали модели внимания для улучшения приложения классификации на основе токенов, которое обсуждалось в главе 7?

3. Какое отношение имеет алгоритм k -средних к соревновательному обучению?
4. Реализуйте самоорганизующуюся карту Кохонена с помощью 1) прямоугольной и 2) гексагональной решеток.
5. Пусть задана игра для двух участников наподобие GAN с целевой функцией $f(x, y)$, и мы хотим вычислить $\min_x \max_y f(x, y)$. Как соотносятся между собой величины $\min_x \max_y f(x, y)$ и $\min_y \max_x f(x, y)$ и когда они равны?
6. Пусть задана функция $f(x, y) = \sin(x + y)$. Мы хотим минимизировать $f(x, y)$ по x и максимизировать по y . Реализуйте для GAN чередование процессов градиентного спуска и подъема, которые обсуждались в данной книге, с целью оптимизации этой функции. Будете ли вы всегда получать одно и то же решение, если начинать с разных стартовых точек?

Библиография

1. D. Ackley, G. Hinton, and T. Sejnowski. A learning algorithm for Boltzmann machines. *Cognitive Science*, 9(1), pp. 147–169, 1985.
2. C. Aggarwal. Data classification: Algorithms and applications, *CRC Press*, 2014.
3. C. Aggarwal. Data mining: The textbook. *Springer*, 2015.
4. C. Aggarwal. Recommender systems: The textbook. *Springer*, 2016.
5. C. Aggarwal. Outlier analysis. *Springer*, 2017.
6. C. Aggarwal. Machine learning for text. *Springer*, 2018.
7. R. Ahuja, T. Magnanti, and J. Orlin. Network flows: Theory, algorithms, and applications. *Prentice Hall*, 1993.
8. E. Aljalbout, V. Golkov, Y. Siddiqui, and D. Cremers. Clustering with deep learning: Taxonomy and new methods. *arXiv:1801.07648*, 2018.
<https://arxiv.org/abs/1801.07648>
9. R. Al-Rfou, B. Perozzi, and S. Skiena. Polyglot: Distributed word representations for multilingual nlp. *arXiv:1307.1662*, 2013.
<https://arxiv.org/abs/1307.1662>
10. D. Amodei et al. Concrete problems in AI safety. *arXiv:1606.06565*, 2016.
<https://arxiv.org/abs/1606.06565>
11. M. Arjovsky and L. Bottou. Towards principled methods for training generative adversarial networks. *arXiv:1701.04862*, 2017.
<https://arxiv.org/abs/1701.04862>
12. M. Arjovsky, S. Chintala, and L. Bottou. Wasserstein gan. *arXiv:1701.07875*, 2017.
<https://arxiv.org/abs/1701.07875>
13. J. Ba and R. Caruana. Do deep nets really need to be deep? *NIPS Conference*, pp. 2654–2662, 2014.
14. J. Ba, J. Kiros, and G. Hinton. Layer normalization. *arXiv:1607.06450*, 2016.
<https://arxiv.org/abs/1607.06450>

15. J. Ba, V. Mnih, and K. Kavukcuoglu. Multiple object recognition with visual attention. *arXiv: 1412.7755*, 2014.
<https://arxiv.org/abs/1412.7755>
16. A. Babenko, A. Slesarev, A. Chigorin, and V. Lempitsky. Neural codes for image retrieval. *arXiv:1404.1777*, 2014.
<https://arxiv.org/abs/1404.1777>
17. M. Baccouche, F. Mamalet, C. Wolf, C. Garcia, and A. Baskurt. Sequential deep learning for human action recognition. *International Workshop on Human Behavior Understanding*, pp. 29–39, 2011.
18. D. Bahdanau, K. Cho, and Y. Bengio. Neural machine translation by jointly learning to align and translate. *ICLR*, 2015. Also *arXiv:1409.0473*, 2014.
<https://arxiv.org/abs/1409.0473>
19. B. Baker, O. Gupta, N. Naik, and R. Raskar. Designing neural network architectures using reinforcement learning. *arXiv:1611.02167*, 2016.
<https://arxiv.org/abs/1611.02167>
20. P. Baldi, S. Brunak, P. Frasconi, G. Soda, and G. Pollastri. Exploiting the past and the future in protein secondary structure prediction. *Bioinformatics*, 15(11), pp. 937–946, 1999.
21. N. Ballas, L. Yao, C. Pal, and A. Courville. Delving deeper into convolutional networks for learning video representations. *arXiv:1511.06432*, 2015.
<https://arxiv.org/abs/1511.06432>
22. J. Baxter, A. Tridgell, and L. Weaver. Knightcap: a chess program that learns by combining td (λ) with game-tree search. *arXiv cs/9901002*, 1999.
23. M. Bazarara, H. Sherali, and C. Shetty. Nonlinear programming: theory and algorithms. *John Wiley and Sons*, 2013.
24. S. Becker, and Y. LeCun. Improving the convergence of back-propagation learning with second order methods. *Proceedings of the 1988 connectionist models summer school*, pp. 29–37, 1988.
25. M. Bellemare, Y. Naddaf, J. Veness, and M. Bowling. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47, pp. 253–279, 2013.
26. R. E. Bellman. *Dynamic Programming*. Princeton University Press, 1957.
27. Y. Bengio. Learning deep architectures for AI. *Foundations and Trends in Machine Learning*, 2(1), pp. 1–127, 2009.
28. Y. Bengio, A. Courville, and P. Vincent. Representation learning: A review and new perspectives. *IEEE TPAMI*, 35(8), pp. 1798–1828, 2013.

29. Y. Bengio and O. Delalleau. Justifying and generalizing contrastive divergence. *Neural Computation*, 21(6), pp. 1601–1621, 2009.
30. Y. Bengio and O. Delalleau. On the expressive power of deep architectures. *Algorithmic Learning Theory*, pp. 18–36, 2011.
31. Y. Bengio, P. Lamblin, D. Popovici, and H. Larochelle. Greedy layer-wise training of deep networks. *NIPS Conference*, 19, 153, 2007.
32. Y. Bengio, N. Le Roux, P. Vincent, O. Delalleau, and P. Marcotte. Convex neural networks. *NIPS Conference*, pp. 123–130, 2005.
33. Y. Bengio, J. Louradour, R. Collobert, and J. Weston. Curriculum learning. *ICML Conference*, 2009.
34. Y. Bengio, L. Yao, G. Alain, and P. Vincent. Generalized denoising auto-encoders as generative models. *NIPS Conference*, pp. 899–907, 2013.
35. J. Bergstra et al. Theano: A CPU and GPU math compiler in Python. *Python in Science Conference*, 2010.
36. J. Bergstra, R. Bardenet, Y. Bengio, and B. Kegl. Algorithms for hyper-parameter optimization. *NIPS Conference*, pp. 2546–2554, 2011.
37. J. Bergstra and Y. Bengio. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13, pp. 281–305, 2012.
38. J. Bergstra, D. Yamins, and D. Cox. Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures. *ICML Conference*, pp. 115–123, 2013.
39. D. Bertsekas. Nonlinear programming. *Athena Scientific*, 1999.
40. C. M. Bishop. Pattern recognition and machine learning. *Springer*, 2007.
41. C. M. Bishop. Neural networks for pattern recognition. *Oxford University Press*, 1995.
42. C. M. Bishop. Bayesian Techniques. Chapter 10 in “Neural Networks for Pattern Recognition,” pp. 385–439, 1995.
43. C. M Bishop. Improving the generalization properties of radial basis function neural networks. *Neural Computation*, 3(4), pp. 579–588, 1991.
44. C. M. Bishop. Training with noise is equivalent to Tikhonov regularization. *Neural computation*, 7(1), pp. 108–116, 1995.
45. C. M. Bishop, M. Svensen, and C. K. Williams. GTM: A principled alternative to the self-organizing map. *NIPS Conference*, pp. 354–360, 1997.
46. M. Bojarski et al. End to end learning for self-driving cars. *arXiv:1604.07316*, 2016.

<https://arxiv.org/abs/1604.07316>

47. M. Bojarski et al. Explaining How a Deep Neural Network Trained with End-to-End Learning Steers a Car. *arXiv:1704.07911*, 2017.
<https://arxiv.org/abs/1704.07911>
48. H. Bourslard and Y. Kamp. Auto-association by multilayer perceptrons and singular value decomposition. *Biological Cybernetics*, 59(4), pp. 291–294, 1988.
49. L. Breiman. Random forests. *Journal Machine Learning archive*, 45(1), pp. 5–32, 2001.
50. L. Breiman. Bagging predictors. *Machine Learning*, 24(2), pp. 123–140, 1996.
51. D. Broomhead and D. Lowe. Multivariable functional interpolation and adaptive networks. *Complex Systems*, 2, pp. 321–355, 1988.
52. C. Browne et al. A survey of monte carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in Games*, 4(1), pp. 1–43, 2012.
53. T. Brox and J. Malik. Large displacement optical flow: descriptor matching in variational motion estimation. *IEEE TPAMI*, 33(3), pp. 500–513, 2011.
54. A. Bryson. A gradient method for optimizing multi-stage allocation processes. *Harvard University Symposium on Digital Computers and their Applications*, 1961.
55. C. Bucilu, R. Caruana, and A. Niculescu-Mizil. Model compression. *ACM KDD Conference*, pp. 535–541, 2006.
56. P. Buhlmann and B. Yu. Analyzing bagging. *Annals of Statistics*, pp. 927–961, 2002.
57. M. Buhlmann. Radial Basis Functions: Theory and implementations. *Cambridge University Press*, 2003.
58. Y. Burda, R. Grosse, and R. Salakhutdinov. Importance weighted autoencoders. *arXiv:1509.00519*, 2015.
<https://arxiv.org/abs/1509.00519>
59. N. Butko and J. Movellan. I-POMDP: An infomax model of eye movement. *IEEE International Conference on Development and Learning*, pp. 139–144, 2008.
60. Y. Cao, Y. Chen, and D. Khosla. Spiking deep convolutional neural networks for energy efficient object recognition. *International Journal of Computer Vision*, 113(1), 54–66, 2015.
61. M. Carreira-Perpinan and G. Hinton. On Contrastive Divergence Learning. *AISTATS*, 10, pp. 33–40, 2005.
62. S. Chang, W. Han, J. Tang, G. Qi, C. Aggarwal, and T. Huang. Heterogeneous network embedding via deep architectures. *ACM KDD Conference*, pp. 119–128, 2015.

63. N. Chawla, K. Bowyer, L. Hall, and W. Kegelmeyer. SMOTE: synthetic minority oversampling technique. *Journal of Artificial Intelligence Research*, 16, pp. 321–357, 2002.
64. J. Chen, S. Sathe, C. Aggarwal, and D. Turaga. Outlier detection with autoencoder ensembles. *SIAM Conference on Data Mining*, 2017.
65. S. Chen, C. Cowan, and P. Grant. Orthogonal least-squares learning algorithm for radial basis function networks. *IEEE Transactions on Neural Networks*, 2(2), pp. 302–309, 1991.
66. W. Chen, J. Wilson, S. Tyree, K. Weinberger, and Y. Chen. Compressing neural networks with the hashing trick. *ICML Conference*, pp. 2285–2294, 2015.
67. Y. Chen and M. Zaki. KATE: K-Competitive Autoencoder for Text. *ACM KDD Conference*, 2017.
68. Y. Chen, T. Krishna, J. Emer, and V. Sze. Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks. *IEEE Journal of Solid-State Circuits*, 52(1), pp. 127–138, 2017.
69. K. Cho, B. Merriënboer, C. Gulcehre, F. Bougares, H. Schwenk, and Y. Bengio. Learning phrase representations using RNN encoder-decoder for statistical machine translation. *EMNLP*, 2014.
<https://arxiv.org/pdf/1406.1078.pdf>
70. J. Chorowski, D. Bahdanau, D. Serdyuk, K. Cho, and Y. Bengio. Attention-based models for speech recognition. *NIPS Conference*, pp. 577–585, 2015.
71. J. Chung, C. Gulcehre, K. Cho, and Y. Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv:1412.3555*, 2014.
<https://arxiv.org/abs/1412.3555>
72. D. Ciresan, U. Meier, L. Gambardella, and J. Schmidhuber. Deep, big, simple neural nets for handwritten digit recognition. *Neural Computation*, 22(12), pp. 3207–3220, 2010.
73. C. Clark and A. Storkey. Training deep convolutional neural networks to play go. *ICML Conference*, pp. 1766–1774, 2015.
74. A. Coates, B. Huval, T. Wang, D. Wu, A. Ng, and B. Catanzaro. Deep learning with COTS HPC systems. *ICML Conference*, pp. 1337–1345, 2013.
75. A. Coates and A. Ng. The importance of encoding versus training with sparse coding and vector quantization. *ICML Conference*, pp. 921–928, 2011.
76. A. Coates and A. Ng. Learning feature representations with k-means. *Neural networks: Tricks of the Trade*, Springer, pp. 561–580, 2012.
77. A. Coates, A. Ng, and H. Lee. An analysis of single-layer networks in unsupervised feature learning. *AAAI Conference*, pp. 215–223, 2011.

78. R. Collobert, J. Weston, L. Bottou, M. Karlen, K. Kavukcuoglu, and P. Kuksa. Natural language processing (almost) from scratch. *Journal of Machine Learning Research*, 12, pp. 2493–2537, 2011.
79. R. Collobert and J. Weston. A unified architecture for natural language processing: Deep neural networks with multitask learning. *ICML Conference*, pp. 160–167, 2008.
80. J. Connor, R. Martin, and L. Atlas. Recurrent neural networks and robust time series prediction. *IEEE Transactions on Neural Networks*, 5(2), pp. 240–254, 1994.
81. T. Cooijmans, N. Ballas, C. Laurent, C. Gulcehre, and A. Courville. Recurrent batch normalization. *arXiv:1603.09025*, 2016.
<https://arxiv.org/abs/1603.09025>
82. C. Cortes and V. Vapnik. Support-vector networks. *Machine Learning*, 20(3), pp. 273–297, 1995.
83. M. Courbariaux, Y. Bengio, and J.-P. David. BinaryConnect: Training deep neural networks with binary weights during propagations. *arXiv:1511.00363*, 2015.
<https://arxiv.org/pdf/1511.00363.pdf>
84. T. Cover. Geometrical and statistical properties of systems of linear inequalities with applications to pattern recognition. *IEEE Transactions on Electronic Computers*, pp. 326–334, 1965.
85. D. Cox and N. Pinto. Beyond simple features: A large-scale feature search approach to unconstrained face recognition. *IEEE International Conference on Automatic Face and Gesture Recognition and Workshops*, pp. 8–15, 2011.
86. G. Dahl, R. Adams, and H. Larochelle. Training restricted Boltzmann machines on word observations. *arXiv:1202.5695*, 2012.
<https://arxiv.org/abs/1202.5695>
87. N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. *Computer Vision and Pattern Recognition*, pp. 886–893, 2005.
88. Y. Dauphin, R. Pascanu, C. Gulcehre, K. Cho, S. Ganguli, and Y. Bengio. Identifying and attacking the saddle point problem in high-dimensional non-convex optimization. *NIPS Conference*, pp. 2933–2941, 2014.
89. N. de Freitas. Machine Learning, University of Oxford (Course Video), 2013.
<https://www.youtube.com/watch?v=w20twL5Tlow&list=PLE6Wd9FREdyJ51bF18Uu-GjecnVw66F6>
90. N. de Freitas. Deep Learning, University of Oxford (Course Video), 2015.
<https://www.youtube.com/watch?v=PlhFWT7vAEw&list=PLjK8ddCbDMphIMSXn-1IjyYpHU3DaUYw>
91. J. Dean et al. Large scale distributed deep networks. *NIPS Conference*, 2012.

92. M. Defferrard, X. Bresson, and P. Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. *NIPS Conference*, pp. 3844–3852, 2016.
93. O. Delalleau and Y. Bengio. Shallow vs. deep sum-product networks. *NIPS Conference*, pp. 666–674, 2011.
94. M. Denil, B. Shakibi, L. Dinh, M. A. Ranzato, and N. de Freitas. Predicting parameters in deep learning. *NIPS Conference*, pp. 2148–2156, 2013.
95. E. Denton, S. Chintala, and R. Fergus. Deep Generative Image Models using a Laplacian Pyramid of Adversarial Networks. *NIPS Conference*, pp. 1466–1494, 2015.
96. G. Desjardins, K. Simonyan, and R. Pascanu. Natural neural networks. *NIPS Conference*, pp. 2071–2079, 2015.
97. F. Despaigne and D. Massart. Neural networks in multivariate calibration. *Analyst*, 123(11), pp. 157R–178R, 1998.
98. T. Dettmers. 8-bit approximations for parallelism in deep learning. *arXiv:1511.04561*, 2015.
<https://arxiv.org/abs/1511.04561>
99. C. Ding, T. Li, and W. Peng. On the equivalence between non-negative matrix factorization and probabilistic latent semantic indexing. *Computational Statistics and Data Analysis*, 52(8), pp. 3913–3927, 2008.
100. J. Donahue, L. Anne Hendricks, S. Guadarrama, M. Rohrbach, S. Venugopalan, K. Saenko, and T. Darrell. Long-term recurrent convolutional networks for visual recognition and description. *IEEE conference on computer vision and pattern recognition*, pp. 2625–2634, 2015.
101. G. Dorffner. Neural networks for time series processing. *Neural Network World*, 1996.
102. C. Dos Santos and M. Gatti. Deep Convolutional Neural Networks for Sentiment Analysis of Short Texts. *COLING*, pp. 69–78, 2014.
103. A. Dosovitskiy, J. Tobias Springenberg, and T. Brox. Learning to generate chairs with convolutional neural networks. *CVPR Conference*, pp. 1538–1546, 2015.
104. A. Dosovitskiy and T. Brox. Inverting visual representations with convolutional networks. *CVPR Conference*, pp. 4829–4837, 2016.
105. K. Doya. Bifurcations of recurrent neural networks in gradient descent learning. *IEEE Transactions on Neural Networks*, 1, pp. 75–80, 1993.
106. C. Doersch. Tutorial on variational autoencoders. *arXiv:1606.05908*, 2016.
<https://arxiv.org/abs/1606.05908>

107. H. Drucker and Y. LeCun. Improving generalization performance using double backpropagation. *IEEE Transactions on Neural Networks*, 3(6), pp. 991–997, 1992.
108. J. Duchi, E. Hazan, and Y. Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12, pp. 2121–2159, 2011.
109. V. Dumoulin and F. Visin. A guide to convolution arithmetic for deep learning. *arXiv:1603.07285*, 2016.
<https://arxiv.org/abs/1603.07285>
110. A. Elkahky, Y. Song, and X. He. A multi-view deep learning approach for cross domain user modeling in recommendation systems. *WWW Conference*, pp. 278–288, 2015.
111. J. Elman. Finding structure in time. *Cognitive Science*, 14(2), pp. 179–211, 1990.
112. J. Elman. Learning and development in neural networks: The importance of starting small. *Cognition*, 48, pp. 781–799, 1993.
113. D. Erhan, Y. Bengio, A. Courville, P. Manzagol, P. Vincent, and S. Bengio. Why does unsupervised pre-training help deep learning? *Journal of Machine Learning Research*, 11, pp. 625–660, 2010.
114. S. Essar et al. Convolutional neural networks for fast, energy-efficient neuro-morphic computing. *Proceedings of the National Academy of Science of the United States of America*, 113(41), pp. 11441–11446, 2016.
115. A. Fader, L. Zettlemoyer, and O. Etzioni. Paraphrase-Driven Learning for Open Question Answering. *ACL*, pp. 1608–1618, 2013.
116. L. Fei-Fei, R. Fergus, and P. Perona. One-shot learning of object categories. *IEEE TPAMI*, 28(4), pp. 594–611, 2006.
117. P. Felzenszwalb, R. Girshick, D. McAllester, and D. Ramanan. Object detection with discriminatively trained part-based models. *IEEE TPAMI*, 32(9), pp. 1627–1645, 2010.
118. A. Fader, L. Zettlemoyer, and O. Etzioni. Open question answering over curated and extracted knowledge bases. *ACM KDD Conference*, 2014.
119. A. Fischer and C. Igel. An introduction to restricted Boltzmann machines. *Progress in Pattern Recognition, Image Analysis, Computer Vision, and Applications*, pp. 14–36, 2012.
120. R. Fisher. The use of multiple measurements in taxonomic problems. *Annals of Eugenics*, 7:pp. 179–188, 1936.

121. P. Frasconi, M. Gori, and A. Sperduti. A general framework for adaptive processing of data structures. *IEEE Transactions on Neural Networks*, 9(5), pp. 768–786, 1998.
122. Y. Freund and R. Schapire. A decision-theoretic generalization of online learning and application to boosting. *Computational Learning Theory*, pp. 23–37, 1995.
123. Y. Freund and R. Schapire. Large margin classification using the perceptron algorithm. *Machine Learning*, 37(3), pp. 277–296, 1999.
124. Y. Freund and D. Haussler. Unsupervised learning of distributions on binary vectors using two layer networks. *Technical report*, Santa Cruz, CA, USA, 1994
125. B. Fritzke. Fast learning with incremental RBF networks. *Neural Processing Letters*, 1(1), pp. 2–5, 1994.
126. B. Fritzke. A growing neural gas network learns topologies. *NIPS Conference*, pp. 625–632, 1995.
127. K. Fukushima. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernetics*, 36(4), pp. 193–202, 1980.
128. S. Gallant. Perceptron-based learning algorithms. *IEEE Transactions on Neural Networks*, 1(2), pp. 179–191, 1990.
129. S. Gallant. Neural network learning and expert systems. *MIT Press*, 1993.
130. H. Gao, H. Yuan, Z. Wang, and S. Ji. Pixel Deconvolutional Networks. *arXiv:1705.06820*, 2017.
<https://arxiv.org/abs/1705.06820>
131. L. Gatys, A. S. Ecker, and M. Bethge. Texture synthesis using convolutional neural networks. *NIPS Conference*, pp. 262–270, 2015.
132. L. Gatys, A. Ecker, and M. Bethge. Image style transfer using convolutional neural networks. *IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2414–2423, 2015.
133. H. Gavin. The Levenberg-Marquardt method for nonlinear least squares curve-fitting problems, 2011.
<http://people.duke.edu/~hpgavin/ce281/lm.pdf>
134. P. Gehler, A. Holub, and M. Welling. The Rate Adapting Poisson (RAP) model for information retrieval and object recognition. *ICML Conference*, 2006.
135. S. Gelly et al. The grand challenge of computer Go: Monte Carlo tree search and extensions. *Communications of the ACM*, 55, pp. 106–113, 2012.

136. A. Gersho and R. M. Gray. Vector quantization and signal compression. *Springer Science and Business Media*, 2012.
137. A. Ghodsi. STAT 946: Topics in Probability and Statistics: Deep Learning, *University of Waterloo*, Fall 2015.
<https://www.youtube.com/watch?v=fyAZszlPphs&list=PLehuLRPyt1Hyi78UOkMP-WCGRxGcA9NVOE>
138. W. Gilks, S. Richardson, and D. Spiegelhalter. Markov chain Monte Carlo in practice. *CRC Press*, 1995.
139. F. Girosi and T. Poggio. Networks and the best approximation property. *Biological Cybernetics*, 63(3), pp. 169–176, 1990.
140. X. Glorot and Y. Bengio. Understanding the difficulty of training deep feedforward neural networks. *AISTATS*, pp. 249–256, 2010.
141. X. Glorot, A. Bordes, and Y. Bengio. Deep Sparse Rectifier Neural Networks. *AISTATS*, 15(106), 2011.
142. P. Glynn. Likelihood ratio gradient estimation: an overview, *Proceedings of the 1987 Winter Simulation Conference*, pp. 366–375, 1987.
143. Y. Goldberg. A primer on neural network models for natural language processing. *Journal of Artificial Intelligence Research (JAIR)*, 57, pp. 345–420, 2016.
144. C. Goller and A. Küchler. Learning task-dependent distributed representations by backpropagation through structure. *Neural Networks*, 1, pp. 347–352, 1996.
145. I. Goodfellow. NIPS 2016 tutorial: Generative adversarial networks. *arXiv:1701.00160*, 2016.
<https://arxiv.org/abs/1701.00160>
146. I. Goodfellow, O. Vinyals, and A. Saxe. Qualitatively characterizing neural network optimization problems. *arXiv:1412.6544*, 2014. [Also appears in *International Conference in Learning Representations*, 2015]
<https://arxiv.org/abs/1412.6544>
147. I. Goodfellow, Y. Bengio, and A. Courville. Deep learning. *MIT Press*, 2016.
148. I. Goodfellow, D. Warde-Farley, M. Mirza, A. Courville, and Y. Bengio. Max-out networks. *arXiv:1302.4389*, 2013.
149. I. Goodfellow et al. Generative adversarial nets. *NIPS Conference*, 2014.
150. A. Graves, A. Mohamed, and G. Hinton. Speech recognition with deep recurrent neural networks. *Acoustics, Speech and Signal Processing (ICASSP)*, pp. 6645–6649, 2013.
151. A. Graves. Generating sequences with recurrent neural networks. *arXiv:1308.0850*, 2013.
<https://arxiv.org/abs/1308.0850>

152. A. Graves. Supervised sequence labelling with recurrent neural networks *Springer*, 2012.
<http://rd.springer.com/book/10.1007%2F978-3-642-24797-2>
153. A. Graves, S. Fernandez, F. Gomez, and J. Schmidhuber. Connectionist temporal classification: labelling unsegmented sequence data with recurrent neural networks. *ICML Conference*, pp. 369–376, 2006.
154. A. Graves, M. Liwicki, S. Fernandez, R. Bertolami, H. Bunke, and J. Schmidhuber. A novel connectionist system for unconstrained handwriting recognition. *IEEE TPAMI*, 31(5), pp. 855–868, 2009.
155. A. Graves and J. Schmidhuber. Framewise Phoneme Classification with Bidirectional LSTM and Other Neural Network Architectures. *Neural Networks*, 18(5–6), pp. 602–610, 2005.
156. A. Graves and J. Schmidhuber. Offline handwriting recognition with multidimensional recurrent neural networks. *NIPS Conference*, pp. 545–552, 2009.
157. A. Graves and N. Jaitly. Towards End-To-End Speech Recognition with Recurrent Neural Networks. *ICML Conference*, pp. 1764–1772, 2014.
158. A. Graves, G. Wayne, and I. Danihelka. Neural Turing machines. *arXiv:1410.5401*, 2014.
<https://arxiv.org/abs/1410.5401>
159. A. Graves et al. Hybrid computing using a neural network with dynamic external memory. *Nature*, 538.7626, pp. 471–476, 2016.
160. K. Greff, R. K. Srivastava, J. Koutnik, B. Steunebrink, and J. Schmidhuber. LSTM: A search space odyssey. *IEEE Transactions on Neural Networks and Learning Systems*, 2016.
<http://ieeexplore.ieee.org/abstract/document/7508408/>
161. K. Greff, R. K. Srivastava, and J. Schmidhuber. Highway and residual networks learn unrolled iterative estimation. *arXiv:1612.07771*, 2016.
<https://arxiv.org/abs/1612.07771>
162. I. Grondman, L. Busoniu, G. A. Lopes, and R. Babuska. A survey of actor-critic reinforcement learning: Standard and natural policy gradients. *IEEE Transactions on Systems, Man, and Cybernetics*, 42(6), pp. 1291–1307, 2012.
163. R. Girshick, F. Iandola, T. Darrell, and J. Malik. Deformable part models are convolutional neural networks. *IEEE Conference on Computer Vision and Pattern Recognition*, pp. 437–446, 2015.
164. A. Grover and J. Leskovec. node2vec: Scalable feature learning for networks. *ACM KDD Conference*, pp. 855–864, 2016.

165. X. Guo, S. Singh, H. Lee, R. Lewis, and X. Wang. Deep learning for real-time Atari game play using offline Monte-Carlo tree search planning. *Advances in NIPS Conference*, pp. 3338–3346, 2014.
166. M. Gutmann and A. Hyvarinen. Noise-contrastive estimation: A new estimation principle for unnormalized statistical models. *AISTATS*, 1(2), pp. 6, 2010.
167. R. Hahnloser and H. S. Seung. Permitted and forbidden sets in symmetric threshold-linear networks. *NIPS Conference*, pp. 217–223, 2001.
168. S. Han, X. Liu, H. Mao, J. Pu, A. Pedram, M. Horowitz, and W. Dally. EIE: Efficient Inference Engine for Compressed Neural Network. *ACM SIGARCH Computer Architecture News*, 44(3), pp. 243–254, 2016.
169. S. Han, J. Pool, J. Tran, and W. Dally. Learning both weights and connections for efficient neural networks. *NIPS Conference*, pp. 1135–1143, 2015.
170. L. K. Hansen and P. Salamon. Neural network ensembles. *IEEE TPAMI*, 12(10), pp. 993–1001, 1990.
171. M. Hardt, B. Recht, and Y. Singer. Train faster, generalize better: Stability of stochastic gradient descent. *ICML Conference*, pp. 1225–1234, 2006.
172. B. Hariharan, P. Arbelaez, R. Girshick, and J. Malik. Simultaneous detection and segmentation. *arXiv:1407.1808*, 2014.
<https://arxiv.org/abs/1407.1808>
173. E. Hartman, J. Keeler, and J. Kowalski. Layered neural networks with Gaussian hidden units as universal approximations. *Neural Computation*, 2(2), pp. 210–215, 1990.
174. H. van Hasselt, A. Guez, and D. Silver. Deep Reinforcement Learning with Double Q-Learning. *AAAI Conference*, 2016.
175. B. Hassibi and D. Stork. Second order derivatives for network pruning: Optimal brain surgeon. *NIPS Conference*, 1993.
176. D. Hassabis, D. Kumaran, C. Summerfield, and M. Botvinick. Neuroscience-inspired artificial intelligence. *Neuron*, 95(2), pp. 245–258, 2017.
177. T. Hastie, R. Tibshirani, and J. Friedman. The elements of statistical learning. *Springer*, 2009.
178. T. Hastie and R. Tibshirani. Generalized additive models. *CRC Press*, 1990.
179. T. Hastie, R. Tibshirani, and M. Wainwright. Statistical learning with sparsity: the lasso and generalizations. *CRC Press*, 2015.
180. M. Havaei et al. Brain tumor segmentation with deep neural networks. *Medical Image Analysis*, 35, pp. 18–31, 2017.

181. S. Hawkins, H. He, G. Williams, and R. Baxter. Outlier detection using replicator neural networks. *International Conference on Data Warehousing and Knowledge Discovery*, pp. 170–180, 2002.
182. S. Haykin. *Neural networks and learning machines*. Pearson, 2008.
183. K. He, X. Zhang, S. Ren, and J. Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. *IEEE International Conference on Computer Vision*, pp. 1026–1034, 2015.
184. K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. *IEEE Conference on Computer Vision and Pattern Recognition*, pp. 770–778, 2016.
185. K. He, X. Zhang, S. Ren, and J. Sun. Identity mappings in deep residual networks. *European Conference on Computer Vision*, pp. 630–645, 2016.
186. X. He, L. Liao, H. Zhang, L. Nie, X. Hu, and T. S. Chua. Neural collaborative filtering. *WWW Conference*, pp. 173–182, 2017.
187. N. Heess et al. Emergence of Locomotion Behaviours in Rich Environments. *arXiv:1707.02286*, 2017.
<https://arxiv.org/abs/1707.02286>
- Видео 1:**
https://www.youtube.com/watch?v=hx_bgoTF7bs
- Видео 2:**
<https://www.youtube.com/watch?v=gn4nRCC9TwQ&feature=youtu.be>
188. M. Henaff, J. Bruna, and Y. LeCun. Deep convolutional networks on graph-structured data. *arXiv:1506.05163*, 2015.
<https://arxiv.org/abs/1506.05163>
189. M. Hestenes and E. Stiefel. Methods of conjugate gradients for solving linear systems. *Journal of Research of the National Bureau of Standards*, 49(6), 1952.
190. G. Hinton. Connectionist learning procedures. *Artificial Intelligence*, 40(1–3), pp. 185–234, 1989.
191. G. Hinton. Training products of experts by minimizing contrastive divergence. *Neural Computation*, 14(8), pp. 1771–1800, 2002.
192. G. Hinton. To recognize shapes, first learn to generate images. *Progress in Brain Research*, 165, pp. 535–547, 2007.
193. G. Hinton. A practical guide to training restricted Boltzmann machines. *Momentum*, 9(1), 926, 2010.

194. G. Hinton. Neural networks for machine learning, *Coursera Video*, 2012.
195. G. Hinton, P. Dayan, B. Frey, and R. Neal. The wake–sleep algorithm for unsupervised neural networks. *Science*, 268(5214), pp. 1158–1162, 1995.
196. G. Hinton, S. Osindero, and Y. Teh. A fast learning algorithm for deep belief nets. *Neural Computation*, 18(7), pp. 1527–1554, 2006.
197. G. Hinton and T. Sejnowski. Learning and relearning in Boltzmann machines. *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, MIT Press, 1986.
198. G. Hinton and R. Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313, (5766), pp. 504–507, 2006.
199. G. Hinton and R. Salakhutdinov. Replicated softmax: an undirected topic model. *NIPS Conference*, pp. 1607–1614, 2009.
200. G. Hinton and R. Salakhutdinov. A better way to pretrain deep Boltzmann machines. *NIPS Conference*, pp. 2447–2455, 2012.
201. G. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv:1207.0580*, 2012.
<https://arxiv.org/abs/1207.0580>
202. G. Hinton, O. Vinyals, and J. Dean. Distilling the knowledge in a neural network. *NIPS Workshop*, 2014.
203. R. Hochberg. Matrix Multiplication with CUDA: A basic introduction to the CUDA programming model. *Unpublished manuscript*, 2012.
<http://www.shodor.org/media/content/petascale/materials/UPModules/matrixMultiplication/moduleDocument.pdf>
204. S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural Computation*, 9(8), pp. 1735–1785, 1997.
205. S. Hochreiter, Y. Bengio, P. Frasconi, and J. Schmidhuber. Gradient flow in recurrent nets: the difficulty of learning long-term dependencies, *A Field Guide to Dynamical Recurrent Neural Networks*, IEEE Press, 2001.
206. T. Hofmann. Probabilistic latent semantic indexing. *ACM SIGIR Conference*, pp. 50–57, 1999.
207. J. J. Hopfield. Neural networks and physical systems with emergent collective computational abilities. *National Academy of Sciences of the USA*, 79(8), pp. 2554–2558, 1982.
208. K. Hornik, M. Stinchcombe, and H. White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5), pp. 359–366, 1989.

209. Y. Hu, Y. Koren, and C. Volinsky. Collaborative filtering for implicit feedback datasets. *IEEE International Conference on Data Mining*, pp. 263–272, 2008.
210. G. Huang, Y. Sun, Z. Liu, D. Sedra, and K. Weinberger. Deep networks with stochastic depth. *European Conference on Computer Vision*, pp. 646–661, 2016.
211. G. Huang, Z. Liu, K. Weinberger, and L. van der Maaten. Densely connected convolutional networks. *arXiv:1608.06993*, 2016.
<https://arxiv.org/abs/1608.06993>
212. D. Hubel and T. Wiesel. Receptive fields of single neurones in the cat’s striate cortex. *The Journal of Physiology*, 124(3), pp. 574–591, 1959.
213. F. Iandola, S. Han, M. Moskewicz, K. Ashraf, W. Dally, and K. Keutzer. SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and < 0.5 MB model size. *arXiv:1602.07360*, 2016.
<https://arxiv.org/abs/1602.07360>
214. S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv:1502.03167*, 2015.
215. P. Isola, J. Zhu, T. Zhou, and A. Efros. Image-to-image translation with conditional adversarial networks. *arXiv:1611.07004*, 2016.
<https://arxiv.org/abs/1611.07004>
216. M. Iyyer, J. Boyd-Graber, L. Claudino, R. Socher, and H. Daume III. A Neural Network for Factoid Question Answering over Paragraphs. *EMNLP*, 2014.
217. R. Jacobs. Increased rates of convergence through learning rate adaptation. *Neural Networks*, 1(4), pp. 295–307, 1988.
218. M. Jaderberg, K. Simonyan, and A. Zisserman. Spatial transformer networks. *NIPS Conference*, pp. 2017–2025, 2015.
219. H. Jaeger. The “echo state” approach to analysing and training recurrent neural networks — with an erratum note. *German National Research Center for Information Technology GMD Technical Report*, 148(34), 13, 2001.
220. H. Jaeger and H. Haas. Harnessing nonlinearity: Predicting chaotic systems and saving energy in wireless communication. *Science*, 304, pp. 78–80, 2004.
221. K. Jarrett, K. Kavukcuoglu, M. Ranzato, and Y. LeCun. What is the best multi-stage architecture for object recognition? *International Conference on Computer Vision (ICCV)*, 2009.
222. S. Ji, W. Xu, M. Yang, and K. Yu. 3D convolutional neural networks for human action recognition. *IEEE TPAMI*, 35(1), pp. 221–231, 2013.
223. Y. Jia et al. Caffe: Convolutional architecture for fast feature embedding. *ACM International Conference on Multimedia*, 2014.

224. C. Johnson. Logistic matrix factorization for implicit feedback data. *NIPS Conference*, 2014.
225. J. Johnson, A. Karpathy, and L. Fei-Fei. Densecap: Fully convolutional localization networks for dense captioning. *IEEE Conference on Computer Vision and Pattern Recognition*, pp. 4565–4574, 2015.
226. J. Johnson, A. Alahi, and L. Fei-Fei. Perceptual losses for real-time style transfer and superresolution. *European Conference on Computer Vision*, pp. 694–711, 2015.
227. R. Johnson and T. Zhang. Effective use of word order for text categorization with convolutional neural networks. *arXiv:1412.1058*, 2014.
<https://arxiv.org/abs/1412.1058>
228. R. Jozefowicz, W. Zaremba, and I. Sutskever. An empirical exploration of recurrent network architectures. *ICML Conference*, pp. 2342–2350, 2015.
229. L. Kaiser and I. Sutskever. Neural GPUs learn algorithms. *arXiv:1511.08228*, 2015.
<https://arxiv.org/abs/1511.08228>
230. S. Kakade. A natural policy gradient. *NIPS Conference*, pp. 1057–1063, 2002.
231. N. Kalchbrenner and P. Blunsom. Recurrent continuous translation models. *EMNLP*, 3, 39, pp. 413, 2013.
232. H. Kandel, J. Schwartz, T. Jessell, S. Siegelbaum, and A. Hudspeth. Principles of neural science. *McGraw Hill*, 2012.
233. A. Karpathy, J. Johnson, and L. Fei-Fei. Visualizing and understanding recurrent networks. *arXiv:1506.02078*, 2015.
<https://arxiv.org/abs/1506.02078>
234. A. Karpathy, G. Toderici, S. Shetty, T. Leung, R. Sukthankar, and L. Fei-Fei. Large-scale video classification with convolutional neural networks. *IEEE Conference on Computer Vision and Pattern Recognition*, pp. 725–1732, 2014.
235. A. Karpathy. The unreasonable effectiveness of recurrent neural networks, *Blog post*, 2015.
<http://karpathy.github.io/2015/05/21/rnn-effectiveness/>
236. A. Karpathy, J. Johnson, and L. Fei-Fei. Stanford University Class CS321n: Convolutional neural networks for visual recognition, 2016.
<http://cs231n.github.io/>
237. H. J. Kelley. Gradient theory of optimal flight paths. *Ars Journal*, 30(10), pp. 947–954, 1960.
238. F. Khan, B. Mutlu, and X. Zhu. How do humans teach: On curriculum learning and teaching dimension. *NIPS Conference*, pp. 1449–1457, 2011.

239. T. Kietzmann, P. McClure, and N. Kriegeskorte. Deep Neural Networks In Computational Neuroscience. *bioRxiv*, 133504, 2017.
<https://www.biorxiv.org/content/early/2017/05/04/133504>
240. Y. Kim. Convolutional neural networks for sentence classification. *arXiv:1408.5882*, 2014.
241. D. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv:1412.6980*, 2014.
<https://arxiv.org/abs/1412.6980>
242. D. Kingma and M. Welling. Auto-encoding variational bayes. *arXiv:1312.6114*, 2013.
<https://arxiv.org/abs/1312.6114>
243. T. Kipf and M. Welling. Semi-supervised classification with graph convolutional networks. *arXiv:1609.02907*, 2016.
<https://arxiv.org/pdf/1609.02907.pdf>
244. S. Kirkpatrick, C. Gelatt, and M. Vecchi. Optimization by simulated annealing. *Science*, 220, pp. 671–680, 1983.
245. J. Kivinen and M. Warmuth. The perceptron algorithm vs. winnow: linear vs. logarithmic mistake bounds when few input variables are relevant. *Computational Learning Theory*, pp. 289–296, 1995.
246. L. Kocsis and C. Szepesvari. Bandit based monte-carlo planning. *ECML Conference*, pp. 282–293, 2006.
247. R. Kohavi and D. Wolpert. Bias plus variance decomposition for zero-one loss functions. *ICML Conference*, 1996.
248. T. Kohonen. The self-organizing map. *Neurocomputing*, 21(1), pp. 1–6, 1998.
249. T. Kohonen. Self-organization and associative memory. *Springer*, 2012.
250. T. Kohonen. Self-organizing maps, *Springer*, 2001.
251. D. Koller and N. Friedman. Probabilistic graphical models: principles and techniques. *MIT Press*, 2009.
252. E. Kong and T. Dietterich. Error-correcting output coding corrects bias and variance. *ICML Conference*, pp. 313–321, 1995.
253. Y. Koren. Factor in the neighbors: Scalable and accurate collaborative filtering. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 4(1), 1, 2010.
254. A. Krizhevsky. One weird trick for parallelizing convolutional neural networks. *arXiv:1404.5997*, 2014.
<https://arxiv.org/abs/1404.5997>
255. A. Krizhevsky, I. Sutskever, and G. Hinton. Imagenet classification with deep convolutional neural networks. *NIPS Conference*, pp. 1097–1105. 2012.

256. M. Kubat. Decision trees can initialize radial-basis function networks. *IEEE Transactions on Neural Networks*, 9(5), pp. 813–821, 1998.
257. A. Kumar et al. Ask me anything: Dynamic memory networks for natural language processing. *ICML Conference*, 2016.
258. Y. Koren. Collaborative filtering with temporal dynamics. *ACM KDD Conference*, pp. 447–455, 2009.
259. M. Lai. Giraffe: Using deep reinforcement learning to play chess. *arXiv:1509.01549*, 2015.
260. S. Lai, L. Xu, K. Liu, and J. Zhao. Recurrent Convolutional Neural Networks for Text Classification. *AAAI*, pp. 2267–2273, 2015.
261. B. Lake, T. Ullman, J. Tenenbaum, and S. Gershman. Building machines that learn and think like people. *Behavioral and Brain Sciences*, pp. 1–101, 2016.
262. H. Larochelle. Neural Networks (Course). *Universite de Sherbrooke*, 2013.
<https://www.youtube.com/watch?v=SGZ6BttHMPw&list=PL6Xpj9I5qXYEcOhn7-TqghAJ6NAPrNmUBH>
263. H. Larochelle and Y. Bengio. Classification using discriminative restricted Boltzmann machines. *ICML Conference*, pp. 536–543, 2008.
264. H. Larochelle, M. Mandel, R. Pascanu, and Y. Bengio. Learning algorithms for the classification restricted Boltzmann machine. *Journal of Machine Learning Research*, 13, pp. 643–669, 2012.
265. H. Larochelle and I. Murray. The neural autoregressive distribution estimator. *International Conference on Artificial Intelligence and Statistics*, pp. 29–37, 2011.
266. H. Larochelle and G. E. Hinton. Learning to combine foveal glimpses with a third-order Boltzmann machine. *NIPS Conference*, 2010.
267. H. Larochelle, D. Erhan, A. Courville, J. Bergstra, and Y. Bengio. An empirical evaluation of deep architectures on problems with many factors of variation. *ICML Conference*, pp. 473–480, 2007.
268. G. Larsson, M. Maire, and G. Shakhnarovich. Fractalnet: Ultra-deep neural networks without residuals. *arXiv:1605.07648*, 2016.
<https://arxiv.org/abs/1605.07648>
269. S. Lawrence, C. L. Giles, A. C. Tsoi, and A. D. Back. Face recognition: A convolutional neural-network approach. *IEEE Transactions on Neural Networks*, 8(1), pp. 98–113, 1997.
270. Q. Le et al. Building high-level features using large scale unsupervised learning. *ICASSP*, 2013.

271. Q. Le, N. Jaitly, and G. Hinton. A simple way to initialize recurrent networks of rectified linear units. *arXiv:1504.00941*, 2015.
<https://arxiv.org/abs/1504.00941>
272. Q. Le and T. Mikolov. Distributed representations of sentences and documents. *ICML Conference*, pp. 1188–196, 2014.
273. Q. Le, J. Ngiam, A. Coates, A. Lahiri, B. Prochnow, and A. Ng. On optimization methods for deep learning. *ICML Conference*, pp. 265–272, 2011.
274. Q. Le, W. Zou, S. Yeung, and A. Ng. Learning hierarchical spatio-temporal features for action recognition with independent subspace analysis. *CVPR Conference*, 2011.
275. Y. LeCun. Modeles connexionnistes de l'apprentissage. *Doctoral Dissertation*, Universite Paris, 1987.
276. Y. LeCun and Y. Bengio. Convolutional networks for images, speech, and time series. *The Handbook of Brain Theory and Neural Networks*, 3361(10), 1995.
277. Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. *Nature*, 521(7553), pp. 436–444, 2015.
278. Y. LeCun, L. Bottou, G. Orr, and K. Muller. Efficient backprop. in G. Orr and K. Muller (eds.) *Neural Networks: Tricks of the Trade*, Springer, 1998.
279. Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11), pp. 2278–2324, 1998.
280. Y. LeCun, S. Chopra, R. M. Hadsell, M. A. Ranzato, and F.-J. Huang. A tutorial on energybased learning. *Predicting Structured Data*, MIT Press, pp. 191–246, 2006.
281. Y. LeCun, C. Cortes, and C. Burges. The MNIST database of handwritten digits, 1998.
<http://yann.lecun.com/exdb/mnist/>
282. Y. LeCun, J. Denker, and S. Solla. Optimal brain damage. *NIPS Conference*, pp. 598–605, 1990.
283. Y. LeCun, K. Kavukcuoglu, and C. Farabet. Convolutional networks and applications in vision. *IEEE International Symposium on Circuits and Systems*, pp. 253–256, 2010.
284. H. Lee, C. Ekanadham, and A. Ng. Sparse deep belief net model for visual area V2. *NIPS Conference*, 2008.
285. H. Lee, R. Grosse, B. Ranganath, and A. Y. Ng. Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations. *ICML Conference*, pp. 609–616, 2009.

286. S. Levine, C. Finn, T. Darrell, and P. Abbeel. End-to-end training of deep visuomotor policies. *Journal of Machine Learning Research*, 17(39), pp. 1–40, 2016.
Видео:
<https://sites.google.com/site/visuomotorpolicy/>
287. O. Levy and Y. Goldberg. Neural word embedding as implicit matrix factorization. *NIPS Conference*, pp. 2177–2185, 2014.
288. O. Levy, Y. Goldberg, and I. Dagan. Improving distributional similarity with lessons learned from word embeddings. *Transactions of the Association for Computational Linguistics*, 3, pp. 211–225, 2015.
289. W. Levy and R. Baxter. Energy efficient neural codes. *Neural Computation*, 8(3), pp. 531–543, 1996.
290. M. Lewis, D. Yarats, Y. Dauphin, D. Parikh, and D. Batra. Deal or No Deal? End-to-End Learning for Negotiation Dialogues. *arXiv:1706.05125*, 2017.
<https://arxiv.org/abs/1706.05125>
291. J. Li, W. Monroe, A. Ritter, M. Galley, J. Gao, and D. Jurafsky. Deep reinforcement learning for dialogue generation. *arXiv:1606.01541*, 2016.
<https://arxiv.org/abs/1606.01541>
292. L. Li, W. Chu, J. Langford, and R. Schapire. A contextual-bandit approach to personalized news article recommendation. *WWW Conference*, pp. 661–670, 2010.
293. Y. Li. Deep reinforcement learning: An overview. *arXiv:1701.07274*, 2017.
<https://arxiv.org/abs/1701.07274>
294. Q. Liao, K. Kawaguchi, and T. Poggio. Streaming normalization: Towards simpler and more biologically-plausible normalizations for online and recurrent learning. *arXiv:1610.06160*, 2016.
<https://arxiv.org/abs/1610.06160>
295. D. Liben-Nowell, and J. Kleinberg. The link-prediction problem for social networks. *Journal of the American Society for Information Science and Technology*, 58(7), pp. 1019–1031, 2007.
296. L.-J. Lin. Reinforcement learning for robots using neural networks. *Technical Report*, DTIC Document, 1993.
297. M. Lin, Q. Chen, and S. Yan. Network in network. *arXiv:1312.4400*, 2013.
<https://arxiv.org/abs/1312.4400>
298. Z. Lipton, J. Berkowitz, and C. Elkan. A critical review of recurrent neural networks for sequence learning. *arXiv:1506.00019*, 2015.
<https://arxiv.org/abs/1506.00019>

299. J. Lu, J. Yang, D. Batra, and D. Parikh. Hierarchical question-image co-attention for visual question answering. *NIPS Conference*, pp. 289–297, 2016.
300. D. Luenberger and Y. Ye. Linear and nonlinear programming, *Addison-Wesley*, 1984.
301. M. Lukosevicius and H. Jaeger. Reservoir computing approaches to recurrent neural network training. *Computer Science Review*, 3(3), pp. 127–149, 2009.
302. M. Luong, H. Pham, and C. Manning. Effective approaches to attention-based neural machine translation. *arXiv:1508.04025*, 2015.
<https://arxiv.org/abs/1508.04025>
303. J. Ma, R. P. Sheridan, A. Liaw, G. E. Dahl, and V. Svetnik. Deep neural nets as a method for quantitative structure-activity relationships. *Journal of Chemical Information and Modeling*, 55(2), pp. 263–274, 2015.
304. W. Maass, T. Natschlager, and H. Markram. Real-time computing without stable states: A new framework for neural computation based on perturbations. *Neural Computation*, 14(11), pp. 2351–2560, 2002.
305. L. Maaten and G. E. Hinton. Visualizing data using t-SNE. *Journal of Machine Learning Research*, 9, pp. 2579–2605, 2008.
306. D. J. MacKay. A practical Bayesian framework for backpropagation networks. *Neural Computation*, 4(3), pp. 448–472, 1992.
307. C. Maddison, A. Huang, I. Sutskever, and D. Silver. Move evaluation in Go using deep convolutional neural networks. *International Conference on Learning Representations*, 2015.
308. A. Mahendran and A. Vedaldi. Understanding deep image representations by inverting them. *IEEE Conference on Computer Vision and Pattern Recognition*, pp. 5188–5196, 2015.
309. A. Makhzani and B. Frey. K-sparse autoencoders. *arXiv:1312.5663*, 2013.
<https://arxiv.org/abs/1312.5663>
310. A. Makhzani and B. Frey. Winner-take-all autoencoders. *NIPS Conference*, pp. 2791–2799, 2015.
311. A. Makhzani, J. Shlens, N. Jaitly, I. Goodfellow, and B. Frey. Adversarial autoencoders. *arXiv:1511.05644*, 2015.
<https://arxiv.org/abs/1511.05644>
312. C. Manning and R. Socher. CS224N: Natural language processing with deep learning. *Stanford University School of Engineering*, 2017.
https://www.youtube.com/watch?v=OQQ-W_63UgQ
313. J. Martens. Deep learning via Hessian-free optimization. *ICML Conference*, pp. 735–742, 2010.

314. J. Martens and I. Sutskever. Learning recurrent neural networks with hessian-free optimization. *ICML Conference*, pp. 1033–1040, 2011.
315. J. Martens, I. Sutskever, and K. Swersky. Estimating the hessian by back-propagating curvature. *arXiv:1206.6464*, 2016.
<https://arxiv.org/abs/1206.6464>
316. J. Martens and R. Grosse. Optimizing Neural Networks with Kronecker-factored Approximate Curvature. *ICML Conference*, 2015.
317. T. Martinez, S. Berkovich, and K. Schulten. ‘Neural-gas’ network for vector quantization and its application to time-series prediction. *IEEE Transactions on Neural Network*, 4(4), pp. 558–569, 1993.
318. J. Masci, U. Meier, D. Ciresan, and J. Schmidhuber. Stacked convolutional auto-encoders for hierarchical feature extraction. *Artificial Neural Networks and Machine Learning*, pp. 52–59, 2011.
319. M. Mathieu, C. Couprie, and Y. LeCun. Deep multi-scale video prediction beyond mean square error. *arXiv:1511.054*, 2015.
<https://arxiv.org/abs/1511.05440>
320. P. McCullagh and J. Nelder. Generalized linear models *CRC Press*, 1989.
321. W. S. McCulloch and W. H. Pitts. A logical calculus of the ideas immanent in nervous activity. *The Bulletin of Mathematical Biophysics*, 5(4), pp. 115–133, 1943.
322. G. McLachlan. Discriminant analysis and statistical pattern recognition. *John Wiley & Sons*, 2004.
323. C. Micchelli. Interpolation of scattered data: distance matrices and conditionally positive definite functions. *Constructive Approximations*, 2, pp. 11–22, 1986.
324. T. Mikolov. Statistical language models based on neural networks. *Ph.D. thesis, Brno University of Technology*, 2012.
325. T. Mikolov, K. Chen, G. Corrado, and J. Dean. Efficient estimation of word representations in vector space. *arXiv:1301.3781*, 2013.
<https://arxiv.org/abs/1301.3781>
326. T. Mikolov, A. Joulin, S. Chopra, M. Mathieu, and M. Ranzato. Learning longer memory in recurrent neural networks. *arXiv:1412.7753*, 2014.
<https://arxiv.org/abs/1412.7753>
327. T. Mikolov, I. Sutskever, K. Chen, G. Corrado, and J. Dean. Distributed representations of words and phrases and their compositionality. *NIPS Conference*, pp. 3111–3119, 2013.
328. T. Mikolov, M. Karafiat, L. Burget, J. Cernocky, and S. Khudanpur. Recurrent neural network based language model. *Interspeech*, Vol 2, 2010.

329. G. Miller, R. Beckwith, C. Fellbaum, D. Gross, and K. J. Miller. Introduction to WordNet: An on-line lexical database. *International Journal of Lexicography*, 3(4), pp. 235–312, 1990.
<https://wordnet.princeton.edu/>
330. M. Minsky and S. Papert. Perceptrons. An Introduction to Computational Geometry, *MIT Press*, 1969.
331. M. Mirza and S. Osindero. Conditional generative adversarial nets. *arXiv:1411.1784*, 2014.
<https://arxiv.org/abs/1411.1784>
332. A. Mnih and G. Hinton. A scalable hierarchical distributed language model. *NIPS Conference*, pp. 1081–1088, 2009.
333. A. Mnih and K. Kavukcuoglu. Learning word embeddings efficiently with noise-contrastive estimation. *NIPS Conference*, pp. 2265–2273, 2013.
334. A. Mnih and Y. Teh. A fast and simple algorithm for training neural probabilistic language models. *arXiv:1206.6426*, 2012.
<https://arxiv.org/abs/1206.6426>
335. V. Mnih et al. Human-level control through deep reinforcement learning. *Nature*, 518 (7540), pp. 529–533, 2015.
336. V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller. Playing atari with deep reinforcement learning. *arXiv:1312.5602*, 2013.
<https://arxiv.org/abs/1312.5602>
337. V. Mnih et al. Asynchronous methods for deep reinforcement learning. *ICML Conference*, pp. 1928–1937, 2016.
338. V. Mnih, N. Heess, and A. Graves. Recurrent models of visual attention. *NIPS Conference*, pp. 2204–2212, 2014.
339. H. Mobahi and J. Fisher. A theoretical analysis of optimization by Gaussian continuation. *AAAI Conference*, 2015.
340. G. Montufar. Universal approximation depth and errors of narrow belief networks with discrete units. *Neural Computation*, 26(7), pp. 1386–1407, 2014.
341. G. Montufar and N. Ay. Refinements of universal approximation results for deep belief networks and restricted Boltzmann machines. *Neural Computation*, 23(5), pp. 1306–1319, 2011.
342. J. Moody and C. Darken. Fast learning in networks of locally-tuned processing units. *Neural Computation*, 1(2), pp. 281–294, 1989.
343. A. Moore and C. Atkeson. Prioritized sweeping: Reinforcement learning with less data and less time. *Machine Learning*, 13(1), pp. 103–130, 1993.

344. F. Morin and Y. Bengio. Hierarchical Probabilistic Neural Network Language Model. *AISTATS*, pp. 246–252, 2005.
345. R. Miotto, F. Wang, S. Wang, X. Jiang, and J. T. Dudley. Deep learning for healthcare: review, opportunities and challenges. *Briefings in Bioinformatics*, pp. 1–11, 2017.
346. M. Müller, M. Enzenberger, B. Arneson, and R. Segal. Fuego — an open-source framework for board games and Go engine based on Monte-Carlo tree search. *IEEE Transactions on Computational Intelligence and AI in Games*, 2, pp. 259–270, 2010.
347. M. Musavi, W. Ahmed, K. Chan, K. Faris, and D. Hummels. On the training of radial basis function classifiers. *Neural Networks*, 5(4), pp. 595–603, 1992.
348. V. Nair and G. Hinton. Rectified linear units improve restricted Boltzmann machines. *ICML Conference*, pp. 807–814, 2010.
349. K. S. Narendra and K. Parthasarathy. Identification and control of dynamical systems using neural networks. *IEEE Transactions on Neural Networks*, 1(1), pp. 4–27, 1990.
350. R. M. Neal. Connectionist learning of belief networks. *Artificial Intelligence*, 1992.
351. R. M. Neal. Probabilistic inference using Markov chain Monte Carlo methods. *Technical Report CRG-TR-93-1*, 1993.
352. R. M. Neal. Annealed importance sampling. *Statistics and Computing*, 11(2), pp. 125–139, 2001.
353. Y. Nesterov. A method of solving a convex programming problem with convergence rate $O(1/k^2)$. *Soviet Mathematics Doklady*, 27, pp. 372–376, 1983.
354. A. Ng. Sparse autoencoder. *CS294A Lecture notes*, 2011.
[https://nlp.stanford.edu/~socherr/sparseAutoencoder 2011new.pdf](https://nlp.stanford.edu/~socherr/sparseAutoencoder%2011new.pdf)
[https://web.stanford.edu/class/cs294a/sparseAutoencoder 2011new.pdf](https://web.stanford.edu/class/cs294a/sparseAutoencoder%2011new.pdf)
355. A. Ng and M. Jordan. PEGASUS: A policy search method for large MDPs and POMDPs. *Uncertainty in Artificial Intelligence*, pp. 406–415, 2000.
356. J. Y.-H. Ng, M. Hausknecht, S. Vijayanarasimhan, O. Vinyals, R. Monga, and G. Toderici. Beyond short snippets: Deep networks for video classification. *IEEE Conference on Computer Vision and Pattern Recognition*, pp. 4694–4702, 2015.
357. J. Ngiam, A. Khosla, M. Kim, J. Nam, H. Lee, and A. Ng. Multimodal deep learning. *ICML Conference*, pp. 689–696, 2011.

358. A. Nguyen, A. Dosovitskiy, J. Yosinski, T. Brox, and J. Clune. Synthesizing the preferred inputs for neurons in neural networks via deep generator networks. *NIPS Conference*, pp. 3387–3395, 2016.
359. J. Nocedal and S. Wright. Numerical optimization. *Springer*, 2006.
360. S. Nowlan and G. Hinton. Simplifying neural networks by soft weight-sharing. *Neural Computation*, 4(4), pp. 473–493, 1992.
361. M. Oquab, L. Bottou, I. Laptev, and J. Sivic. Learning and transferring mid-level image representations using convolutional neural networks. *IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1717–1724, 2014.
362. G. Orr and K.-R. Müller (editors). *Neural Networks: Tricks of the Trade*, Springer, 1998.
363. M. J. L. Orr. Introduction to radial basis function networks, *University of Edinburgh Technical Report, Centre of Cognitive Science*, 1996.
<ftp://ftp.cogsci.ed.ac.uk/pub/mjo/intro.ps.Z>
364. M. Palatucci, D. Pomerleau, G. Hinton, and T. Mitchell. Zero-shot learning with semantic output codes. *NIPS Conference*, pp. 1410–1418, 2009.
365. J. Park and I. Sandberg. Universal approximation using radial-basis-function networks. *Neural Computation*, 3(1), pp. 246–257, 1991.
366. J. Park and I. Sandberg. Approximation and radial-basis-function networks. *Neural Computation*, 5(2), pp. 305–316, 1993.
367. O. Parkhi, A. Vedaldi, and A. Zisserman. Deep Face Recognition. *BMVC*, 1(3), pp. 6, 2015.
368. R. Pascanu, T. Mikolov, and Y. Bengio. On the difficulty of training recurrent neural networks. *ICML Conference*, 28, pp. 1310–1318, 2013.
369. R. Pascanu, T. Mikolov, and Y. Bengio. Understanding the exploding gradient problem. *CoRR*, <abs/1211.5063>, 2012.
370. D. Pathak, P. Krahenbuhl, J. Donahue, T. Darrell, and A. A. Efros. Context encoders: Feature learning by inpainting. *CVPR Conference*, 2016.
371. J. Pennington, R. Socher, and C. Manning. Glove: Global Vectors for Word Representation. *EMNLP*, pp. 1532–1543, 2014.
372. B. Perozzi, R. Al-Rfou, and S. Skiena. Deepwalk: Online learning of social representations. *ACM KDD Conference*, pp. 701–710.
373. C. Peterson and J. Anderson. A mean field theory learning algorithm for neural networks. *Complex Systems*, 1(5), pp. 995–1019, 1987.
374. J. Peters and S. Schaal. Reinforcement learning of motor skills with policy gradients. *Neural Networks*, 21(4), pp. 682–697, 2008.

375. F. Pineda. Generalization of back-propagation to recurrent neural networks. *Physical Review Letters*, 59(19), 2229, 1987.
376. E. Polak. Computational methods in optimization: a unified approach. *Academic Press*, 1971.
377. L. Polanyi and A. Zaenen. Contextual valence shifters. *Computing Attitude and Affect in Text: Theory and Applications*, pp. 1–10, *Springer*, 2006.
378. G. Pollastri, D. Przybylski, B. Rost, and P. Baldi. Improving the prediction of protein secondary structure in three and eight classes using recurrent neural networks and profiles. *Proteins: Structure, Function, and Bioinformatics*, 47(2), pp. 228–235, 2002.
379. J. Pollack. Recursive distributed representations. *Artificial Intelligence*, 46(1), pp. 77–105, 1990.
380. B. Polyak and A. Juditsky. Acceleration of stochastic approximation by averaging. *SIAM Journal on Control and Optimization*, 30(4), pp. 838–855, 1992.
381. D. Pomerleau. ALVINN, an autonomous land vehicle in a neural network. *Technical Report*, Carnegie Mellon University, 1989.
382. B. Poole, J. Sohl-Dickstein, and S. Ganguli. Analyzing noise in autoencoders and deep networks. *arXiv:1406.1831*, 2014.
<https://arxiv.org/abs/1406.1831>
383. H. Poon and P. Domingos. Sum-product networks: A new deep architecture. *Computer Vision Workshops (ICCV Workshops)*, pp. 689–690, 2011.
384. A. Radford, L. Metz, and S. Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv:1511.06434*, 2015.
<https://arxiv.org/abs/1511.06434>
385. A. Rahimi and B. Recht. Random features for large-scale kernel machines. *NIPS Conference*, pp. 1177–1184, 2008.
386. M.' A. Ranzato, Y-L. Boureau, and Y. LeCun. Sparse feature learning for deep belief networks. *NIPS Conference*, pp. 1185–1192, 2008.
387. M.' A. Ranzato, F. J. Huang, Y-L. Boureau, and Y. LeCun. Unsupervised learning of invariant feature hierarchies with applications to object recognition. *Computer Vision and Pattern Recognition*, pp. 1–8, 2007.
388. A. Rasmus, M. Berglund, M. Honkala, H. Valpola, and T. Raiko. Semi-supervised learning with ladder networks. *NIPS Conference*, pp. 3546–3554, 2015.
389. M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi. Xnor-net: Imagenet classification using binary convolutional neural networks. *European Conference on Computer Vision*, pp. 525–542, 2016.

390. A. Razavian, H. Azizpour, J. Sullivan, and S. Carlsson. CNN features off-the-shelf: an astounding baseline for recognition. *IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pp. 806–813, 2014.
391. J. Redmon, S. Divvala, R. Girshick, and A. Farhadi. You only look once: Unified, real-time object detection. *IEEE Conference on Computer Vision and Pattern Recognition*, pp. 779–788, 2016.
392. S. Reed, Z. Akata, X. Yan, L. Logeswaran, B. Schiele, and H. Lee. Generative adversarial text to image synthesis. *ICML Conference*, pp. 1060–1069, 2016.
393. S. Reed and N. de Freitas. Neural programmer-interpreters. *arXiv:1511.06279*, 2015.
394. R. Rehurek and P. Sojka. Software framework for topic modelling with large corpora. *LREC 2010 Workshop on New Challenges for NLP Frameworks*, pp. 45–50, 2010.
<https://radimrehurek.com/gensim/index.html>
395. M. Ren, R. Kiros, and R. Zemel. Exploring models and data for image question answering. *NIPS Conference*, pp. 2953–2961, 2015.
396. S. Rendle. Factorization machines. *IEEE ICDM Conference*, pp. 995–100, 2010.
397. S. Rifai, P. Vincent, X. Muller, X. Glorot, and Y. Bengio. Contractive auto-encoders: Explicit invariance during feature extraction. *ICML Conference*, pp. 833–840, 2011.
398. S. Rifai, Y. Dauphin, P. Vincent, Y. Bengio, and X. Muller. The manifold tangent classifier. *NIPS Conference*, pp. 2294–2302, 2011.
399. D. Rezende, S. Mohamed, and D. Wierstra. Stochastic backpropagation and approximate inference in deep generative models. *arXiv:1401.4082*, 2014.
<https://arxiv.org/abs/1401.4082>
400. R. Rifkin. Everything old is new again: a fresh look at historical approaches in machine learning. *Ph. D. Thesis*, Massachusetts Institute of Technology, 2002.
401. R. Rifkin and A. Klautau. In defense of one-vs-all classification. *Journal of Machine Learning Research*, 5, pp. 101–141, 2004.
402. V. Romanuke. Parallel Computing Center (Khmelnitskiy, Ukraine) represents an ensemble of 5 convolutional neural networks which performs on MNIST at 0.21 percent error rate. Retrieved 24 November 2016.
403. B. Romera-Paredes and P. Torr. An embarrassingly simple approach to zero-shot learning. *ICML Conference*, pp. 2152–2161, 2015.
404. X. Rong. word2vec parameter learning explained. *arXiv:1411.2738*, 2014.
<https://arxiv.org/abs/1411.2738>

405. F. Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(6), 386, 1958.
406. D. Ruck, S. Rogers, and M. Kabrisky. Feature selection using a multilayer perceptron. *Journal of Neural Network Computing*, 2(2), pp. 40–88, 1990.
407. H. A. Rowley, S. Baluja, and T. Kanade. Neural network-based face detection. *IEEE TPAMI*, 20(1), pp. 23–38, 1998.
408. D. Rumelhart, G. Hinton, and R. Williams. Learning representations by back-propagating errors. *Nature*, 323 (6088), pp. 533–536, 1986.
409. D. Rumelhart, G. Hinton, and R. Williams. Learning internal representations by backpropagating errors. In *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, pp. 318–362, 1986.
410. D. Rumelhart, D. Zipser, and J. McClelland. *Parallel Distributed Processing*, MIT Press, pp. 151–193, 1986.
411. D. Rumelhart and D. Zipser. Feature discovery by competitive learning. *Cognitive science*, 9(1), pp. 75–112, 1985.
412. G. Rummery and M. Niranjan. Online Q-learning using connectionist systems (Vol. 37). *University of Cambridge, Department of Engineering*, 1994.
413. A. M. Rush, S. Chopra, and J. Weston. A Neural Attention Model for Abstractive Sentence Summarization. *arXiv:1509.00685*, 2015.
<https://arxiv.org/abs/1509.00685>
414. R. Salakhutdinov, A. Mnih, and G. Hinton. Restricted Boltzmann machines for collaborative filtering. *ICML Conference*, pp. 791–798, 2007.
415. R. Salakhutdinov and G. Hinton. Semantic Hashing. *SIGIR workshop on Information Retrieval and applications of Graphical Models*, 2007.
416. A. Santoro, S. Bartunov, M. Botvinick, D. Wierstra, and T. Lillicrap. One shot learning with memory-augmented neural networks. *arXiv: 1605:06065*, 2016.
<https://www.arxiv.org/pdf/1605.06065.pdf>
417. R. Salakhutdinov and G. Hinton. Deep Boltzmann machines. *Artificial Intelligence and Statistics*, pp. 448–455, 2009.
418. R. Salakhutdinov and H. Larochelle. Efficient Learning of Deep Boltzmann Machines. *AISTATS*, pp. 693–700, 2010.
419. T. Salimans and D. Kingma. Weight normalization: A simple reparameterization to accelerate training of deep neural networks. *NIPS Conference*, pp. 901–909, 2016.
420. T. Salimans, I. Goodfellow, W. Zaremba, V. Cheung, A. Radford, and X. Chen. Improved techniques for training gans. *NIPS Conference*, pp. 2234–2242, 2016.

421. A. Samuel. Some studies in machine learning using the game of checkers. *IBM Journal of Research and Development*, 3, pp. 210–229, 1959.
422. T. Sanger. Neural network learning control of robot manipulators using gradually increasing task difficulty. *IEEE Transactions on Robotics and Automation*, 10(3), 1994.
423. H. Sarimveis, A. Alexandridis, and G. Bafas. A fast training algorithm for RBF networks based on subtractive clustering. *Neurocomputing*, 51, pp. 501–505, 2003.
424. W. Saunders, G. Sastry, A. Stuhlmüller, and O. Evans. Trial without Error: Towards Safe Reinforcement Learning via Human Intervention. *arXiv:1707.05173*, 2017.
<https://arxiv.org/abs/1707.05173>
425. A. Saxe, P. Koh, Z. Chen, M. Bhand, B. Suresh, and A. Ng. On random weights and unsupervised feature learning. *ICML Conference*, pp. 1089–1096, 2011.
426. A. Saxe, J. McClelland, and S. Ganguli. Exact solutions to the nonlinear dynamics of learning in deep linear neural networks. *arXiv:1312.6120*, 2013.
427. S. Schaal. Is imitation learning the route to humanoid robots? *Trends in Cognitive Sciences*, 3(6), pp. 233–242, 1999.
428. T. Schaul, J. Quan, I. Antonoglou, and D. Silver. Prioritized experience replay. *arXiv:1511.05952*, 2015.
<https://arxiv.org/abs/1511.05952>
429. T. Schaul, S. Zhang, and Y. LeCun. No more pesky learning rates. *ICML Conference*, pp. 343–351, 2013.
430. B. Schölkopf, K. Sung, C. Burges, F. Girosi, P. Niyogi, T. Poggio, and V. Vapnik. Comparing support vector machines with Gaussian kernels to radial basis function classifiers. *IEEE Transactions on Signal Processing*, 45(11), pp. 2758–2765, 1997.
431. J. Schmidhuber. Deep learning in neural networks: An overview. *Neural Networks*, 61, pp. 85–117, 2015.
432. J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz. Trust region policy optimization. *ICML Conference*, 2015.
433. J. Schulman, P. Moritz, S. Levine, M. Jordan, and P. Abbeel. High-dimensional continuous control using generalized advantage estimation. *ICLR Conference*, 2016.
434. M. Schuster and K. Paliwal. Bidirectional recurrent neural networks. *IEEE Transactions on Signal Processing*, 45(11), pp. 2673–2681, 1997.

435. H. Schwenk and Y. Bengio. Boosting neural networks. *Neural Computation*, 12(8), pp. 1869–1887, 2000.
436. S. Sedhain, A. K. Menon, S. Sanner, and L. Xie. Autorec: Autoencoders meet collaborative filtering. *WWW Conference*, pp. 111–112, 2015.
437. T. J. Sejnowski. Higher-order Boltzmann machines. *AIP Conference Proceedings*, 15(1), pp. 298–403, 1986.
438. G. Seni and J. Elder. Ensemble methods in data mining: Improving accuracy through combining predictions. *Morgan and Claypool*, 2010.
439. I. Serban, A. Sordoni, R. Lowe, L. Charlin, J. Pineau, A. Courville, and Y. Bengio. A hierarchical latent variable encoder-decoder model for generating dialogues. *AAAI*, pp. 3295–3301, 2017.
440. I. Serban, A. Sordoni, Y. Bengio, A. Courville, and J. Pineau. Building end-to-end dialogue systems using generative hierarchical neural network models. *AAAI Conference*, pp. 3776–3784, 2016.
441. P. Sermanet, D. Eigen, X. Zhang, M. Mathieu, R. Fergus, and Y. LeCun. Overfeat: Integrated recognition, localization and detection using convolutional networks. *arXiv:1312.6229*, 2013.
<https://arxiv.org/abs/1312.6229>
442. A. Shashua. On the equivalence between the support vector machine for classification and sparsified Fisher’s linear discriminant. *Neural Processing Letters*, 9(2), pp. 129–139, 1999.
443. J. Shewchuk. An introduction to the conjugate gradient method without the agonizing pain. *Technical Report, CMU-CS-94-125*, Carnegie-Mellon University, 1994.
444. H. Siegelmann and E. Sontag. On the computational power of neural nets. *Journal of Computer and System Sciences*, 50(1), pp. 132–150, 1995.
445. D. Silver et al. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529.7587, pp. 484–489, 2016.
446. D. Silver et al. Mastering the game of go without human knowledge. *Nature*, 550.7676, pp. 354–359, 2017.
447. D. Silver et al. Mastering chess and shogi by self-play with a general reinforcement learning algorithm. *arXiv*, 2017.
<https://arxiv.org/abs/1712.01815>
448. S. Shalev-Shwartz, Y. Singer, N. Srebro, and A. Cotter. Pegasos: Primal estimated subgradient solver for SVM. *Mathematical Programming*, 127(1), pp. 3–30, 2011.
449. E. Shelhamer, J., Long, and T. Darrell. Fully convolutional networks for semantic segmentation. *IEEE TPAMI*, 39(4), pp. 640–651, 2017.

450. J. Sietsma and R. Dow. Creating artificial neural networks that generalize. *Neural Networks*, 4(1), pp. 67–79, 1991.
451. B. W. Silverman. *Density Estimation for Statistics and Data Analysis*. Chapman and Hall, 1986.
452. P. Simard, D. Steinkraus, and J. C. Platt. Best practices for convolutional neural networks applied to visual document analysis. *ICDAR*, pp. 958–962, 2003.
453. H. Simon. *The Sciences of the Artificial*. MIT Press, 1996.
454. K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv:1409.1556*, 2014.
<https://arxiv.org/abs/1409.1556>
455. K. Simonyan and A. Zisserman. Two-stream convolutional networks for action recognition in videos. *NIPS Conference*, pp. 568–584, 2014.
456. K. Simonyan, A. Vedaldi, and A. Zisserman. Deep inside convolutional networks: Visualising image classification models and saliency maps. *arXiv:1312.6034*, 2013.
457. P. Smolensky. Information processing in dynamical systems: Foundations of harmony theory. *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, Volume 1: Foundations. pp. 194–281, 1986.
458. J. Snoek, H. Larochelle, and R. Adams. Practical bayesian optimization of machine learning algorithms. *NIPS Conference*, pp. 2951–2959, 2013.
459. R. Socher, C. Lin, C. Manning, and A. Ng. Parsing natural scenes and natural language with recursive neural networks. *ICML Conference*, pp. 129–136, 2011.
460. R. Socher, J. Pennington, E. Huang, A. Ng, and C. Manning. Semi-supervised recursive autoencoders for predicting sentiment distributions. *Empirical Methods in Natural Language Processing (EMNLP)*, pp. 151–161, 2011.
461. R. Socher, A. Perelygin, J. Wu, J. Chuang, C. Manning, A. Ng, and C. Potts. Recursive deep models for semantic compositionality over a sentiment tree-bank. *Empirical Methods in Natural Language Processing (EMNLP)*, p. 1642, 2013.
462. Socher, Richard, Milind Ganjoo, Christopher D. Manning, and Andrew Ng. Zero-shot learning through cross-modal transfer. *NIPS Conference*, pp. 935–943, 2013.
463. K. Sohn, H. Lee, and X. Yan. Learning structured output representation using deep conditional generative models. *NIPS Conference*, 2015.
464. R. Solomonoff. A system for incremental learning based on algorithmic probability. *Sixth Israeli Conference on Artificial Intelligence, Computer Vision and Pattern Recognition*, pp. 515–527, 1994.

465. Y. Song, A. Elkahky, and X. He. Multi-rate deep learning for temporal recommendation. *ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 909–912, 2016.
466. J. Springenberg, A. Dosovitskiy, T. Brox, and M. Riedmiller. Striving for simplicity: The all convolutional net. *arXiv:1412.6806*, 2014.
<https://arxiv.org/abs/1412.6806>
467. N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1), pp. 1929–1958, 2014.
468. N. Srivastava and R. Salakhutdinov. Multimodal learning with deep Boltzmann machines. *NIPS Conference*, pp. 2222–2230, 2012.
469. N. Srivastava, R. Salakhutdinov, and G. Hinton. Modeling documents with deep Boltzmann machines. *Uncertainty in Artificial Intelligence*, 2013.
470. R. K. Srivastava, K. Greff, and J. Schmidhuber. Highway networks. *arXiv:1505.00387*, 2015.
<https://arxiv.org/abs/1505.00387>
471. A. Storkey. Increasing the capacity of a Hopfield network without sacrificing functionality. *Artificial Neural Networks*, pp. 451–456, 1997.
472. F. Strub and J. Mary. Collaborative filtering with stacked denoising autoencoders and sparse inputs. *NIPS Workshop on Machine Learning for eCommerce*, 2015.
473. S. Sukhbaatar, J. Weston, and R. Fergus. End-to-end memory networks. *NIPS Conference*, pp. 2440–2448, 2015.
474. Y. Sun, D. Liang, X. Wang, and X. Tang. Deepid3: Face recognition with very deep neural networks. *arXiv:1502.00873*, 2013.
<https://arxiv.org/abs/1502.00873>
475. Y. Sun, X. Wang, and X. Tang. Deep learning face representation from predicting 10,000 classes. *IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1891–1898, 2014.
476. M. Sundermeyer, R. Schluter, and H. Ney. LSTM neural networks for language modeling. *Interspeech*, 2010.
477. M. Sundermeyer, T. Alkhouli, J. Wuebker, and H. Ney. Translation modeling with bidirectional recurrent neural networks. *EMNLP*, pp. 14–25, 2014.
478. I. Sutskever, J. Martens, G. Dahl, and G. Hinton. On the importance of initialization and momentum in deep learning. *ICML Conference*, pp. 1139–1147, 2013.

479. I. Sutskever and T. Tieleman. On the convergence properties of contrastive divergence. *International Conference on Artificial Intelligence and Statistics*, pp. 789–795, 2010.
480. I. Sutskever, O. Vinyals, and Q. V. Le. Sequence to sequence learning with neural networks. *NIPS Conference*, pp. 3104–3112, 2014.
481. I. Sutskever and V. Nair. Mimicking Go experts with convolutional neural networks. *International Conference on Artificial Neural Networks*, pp. 101–110, 2008.
482. R. Sutton. Learning to Predict by the Method of Temporal Differences, *Machine Learning*, 3, pp. 9–44, 1988.
483. R. Sutton and A. Barto. Reinforcement Learning: An Introduction. *MIT Press*, 1998.
484. R. Sutton, D. McAllester, S. Singh, and Y. Mansour. Policy gradient methods for reinforcement learning with function approximation. *NIPS Conference*, pp. 1057–1063, 2000.
485. C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. *IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1–9, 2015.
486. C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna. Rethinking the inception architecture for computer vision. *IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2818–2826, 2016.
487. C. Szegedy, S. Ioffe, V. Vanhoucke, and A. Alemi. Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning. *AAAI Conference*, pp. 4278–4284, 2017.
488. G. Taylor, R. Fergus, Y. LeCun, and C. Bregler. Convolutional learning of spatio-temporal features. *European Conference on Computer Vision*, pp. 140–153, 2010.
489. G. Taylor, G. Hinton, and S. Roweis. Modeling human motion using binary latent variables. *NIPS Conference*, 2006.
490. C. Thornton, F. Hutter, H. H. Hoos, and K. Leyton-Brown. Auto-WEKA: Combined selection and hyperparameter optimization of classification algorithms. *ACM KDD Conference*, pp. 847–855, 2013.
491. T. Tieleman. Training restricted Boltzmann machines using approximations to the likelihood gradient. *ICML Conference*, pp. 1064–1071, 2008.
492. G. Tesauro. Practical issues in temporal difference learning. *Advances in NIPS Conference*, pp. 259–266, 1992.

493. G. Tesauro. Td-gammon: A self-teaching backgammon program. *Applications of Neural Networks*, Springer, pp. 267–285, 1992.
494. G. Tesauro. Temporal difference learning and TD-Gammon. *Communications of the ACM*, 38(3), pp. 58–68, 1995.
495. Y. Teh and G. Hinton. Rate-coded restricted Boltzmann machines for face recognition. *NIPS Conference*, 2001.
496. S. Thrun. Learning to play the game of chess *NIPS Conference*, pp. 1069–1076, 1995.
497. S. Thrun and L. Platt. Learning to learn. *Springer*, 2012.
498. Y. Tian, Q. Gong, W. Shang, Y. Wu, and L. Zitnick. ELF: An extensive, lightweight and flexible research platform for real-time strategy games. *arXiv:1707.01067*, 2017.
<https://arxiv.org/abs/1707.01067>
499. A. Tikhonov and V. Arsenin. Solution of ill-posed problems. *Winston and Sons*, 1977.
500. D. Tran et al. Learning spatiotemporal features with 3d convolutional networks. *IEEE International Conference on Computer Vision*, 2015.
501. R. Uijlings, A. van de Sande, T. Gevers, and M. Smeulders. Selective search for object recognition. *International Journal of Computer Vision*, 104(2), 2013.
502. H. Valpola. From neural PCA to deep unsupervised learning. *Advances in Independent Component Analysis and Learning Machines*, pp. 143–171, Elsevier, 2015.
503. A. Vedaldi and K. Lenc. Matconvnet: Convolutional neural networks for matlab. *ACM International Conference on Multimedia*, pp. 689–692, 2005.
<http://www.vlfeat.org/matconvnet/>
504. V. Veeriah, N. Zhuang, and G. Qi. Differential recurrent neural networks for action recognition. *IEEE International Conference on Computer Vision*, pp. 4041–4049, 2015.
505. A. Veit, M. Wilber, and S. Belongie. Residual networks behave like ensembles of relatively shallow networks. *NIPS Conference*, pp. 550–558, 2016.
506. P. Vincent, H. Larochelle, Y. Bengio, and P. Manzagol. Extracting and composing robust features with denoising autoencoders. *ICML Conference*, pp. 1096–1103, 2008.
507. O. Vinyals, C. Blundell, T. Lillicrap, and D. Wierstra. Matching networks for one-shot learning. *NIPS Conference*, pp. 3530–3638, 2016.
508. O. Vinyals and Q. Le. A Neural Conversational Model. *arXiv:1506.05869*, 2015.
<https://arxiv.org/abs/1506.05869>

509. O. Vinyals, A. Toshev, S. Bengio, and D. Erhan. Show and tell: A neural image caption generator. *CVPR Conference*, pp. 3156–3164, 2015.
510. J. Walker, C. Doersch, A. Gupta, and M. Hebert. An uncertain future: Forecasting from static images using variational autoencoders. *European Conference on Computer Vision*, pp. 835–851, 2016.
511. L. Wan, M. Zeiler, S. Zhang, Y. LeCun, and R. Fergus. Regularization of neural networks using dropconnect. *ICML Conference*, pp. 1058–1066, 2013.
512. D. Wang, P. Cui, and W. Zhu. Structural deep network embedding. *ACM KDD Conference*, pp. 1225–1234, 2016.
513. H. Wang, N. Wang, and D. Yeung. Collaborative deep learning for recommender systems. *ACM KDD Conference*, pp. 1235–1244, 2015.
514. L. Wang, Y. Qiao, and X. Tang. Action recognition with trajectory-pooled deep-convolutional descriptors. *IEEE Conference on Computer Vision and Pattern Recognition*, pp. 4305–4314, 2015.
515. S. Wang, C. Aggarwal, and H. Liu. Using a random forest to inspire a neural network and improving on it. *SIAM Conference on Data Mining*, 2017.
516. S. Wang, C. Aggarwal, and H. Liu. Randomized feature engineering as a fast and accurate alternative to kernel methods. *ACM KDD Conference*, 2017.
517. T. Wang, D. Wu, A. Coates, and A. Ng. End-to-end text recognition with convolutional neural networks. *International Conference on Pattern Recognition*, pp. 3304–3308, 2012.
518. X. Wang and A. Gupta. Generative image modeling using style and structure adversarial networks. *ECCV*, 2016.
519. C. J. H. Watkins. Learning from delayed rewards. *PhD Thesis*, King's College, Cambridge, 1989.
520. C. J. H. Watkins and P. Dayan. Q-learning. *Machine Learning*, 8(3–4), pp. 279–292, 1992.
521. K. Weinberger, B. Packer, and L. Saul. Nonlinear Dimensionality Reduction by Semidefinite Programming and Kernel Matrix Factorization. *AISTATS*, 2005.
522. M. Welling, M. Rosen-Zvi, and G. Hinton. Exponential family harmoniums with an application to information retrieval. *NIPS Conference*, pp. 1481–1488, 2005.
523. A. Wendemuth. Learning the unlearnable. *Journal of Physics A: Math. Gen.*, 28, pp. 5423–5436, 1995.
524. P. Werbos. Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences. *PhD thesis*, Harvard University, 1974.

525. P. Werbos. The roots of backpropagation: from ordered derivatives to neural networks and political forecasting (Vol. 1). *John Wiley and Sons*, 1994.
526. P. Werbos. Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, 78(10), pp. 1550–1560, 1990.
527. J. Weston, A. Bordes, S. Chopra, A. Rush, B. van Merriënboer, A. Joulin, and T. Mikolov. Towards ai-complete question answering: A set of pre-requisite toy tasks. *arXiv:1502.05698*, 2015.
<https://arxiv.org/abs/1502.05698>
528. J. Weston, S. Chopra, and A. Bordes. Memory networks. *ICLR*, 2015.
529. J. Weston and C. Watkins. Multi-class support vector machines. *Technical Report CSD-TR-98-04*, Department of Computer Science, Royal Holloway, University of London, May, 1998.
530. D. Wettschereck and T. Dietterich. Improving the performance of radial basis function networks by learning center locations. *NIPS Conference*, pp. 1133–1140, 1992.
531. B. Widrow and M. Hoff. Adaptive switching circuits. *IRE WESCON Convention Record*, 4(1), pp. 96–104, 1960.
532. S. Wieseler and H. Ney. A convergence analysis of log-linear training. *NIPS Conference*, pp. 657–665, 2011.
533. R. J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8(3–4), pp. 229–256, 1992.
534. C. Wu, A. Ahmed, A. Beutel, A. Smola, and H. Jing. Recurrent recommender networks. *ACM International Conference on Web Search and Data Mining*, pp. 495–503, 2017.
535. Y. Wu, C. DuBois, A. Zheng, and M. Ester. Collaborative denoising auto-encoders for top-n recommender systems. *Web Search and Data Mining*, pp. 153–162, 2016.
536. Z. Wu. Global continuation for distance geometry problems. *SIAM Journal of Optimization*, 7, pp. 814–836, 1997.
537. S. Xie, R. Girshick, P. Dollar, Z. Tu, and K. He. Aggregated residual transformations for deep neural networks. *arXiv:1611.05431*, 2016.
<https://arxiv.org/abs/1611.05431>
538. E. Xing, R. Yan, and A. Hauptmann. Mining associated text and images with dual-wing harmoniums. *Uncertainty in Artificial Intelligence*, 2005.
539. C. Xiong, S. Merity, and R. Socher. Dynamic memory networks for visual and textual question answering. *ICML Conference*, pp. 2397–2406, 2016.
540. K. Xu et al. Show, attend, and tell: Neural image caption generation with visual attention. *ICML Conference*, 2015.

541. O. Yadan, K. Adams, Y. Taigman, and M. Ranzato. Multi-gpu training of convnets. *arXiv:1312.5853*, 2013.
<https://arxiv.org/abs/1312.5853>
542. Z. Yang, X. He, J. Gao, L. Deng, and A. Smola. Stacked attention networks for image question answering. *IEEE Conference on Computer Vision and Pattern Recognition*, pp. 21–29, 2016.
543. X. Yao. Evolving artificial neural networks. *Proceedings of the IEEE*, 87(9), pp. 1423–1447, 1999.
544. F. Yu and V. Koltun. Multi-scale context aggregation by dilated convolutions. *arXiv:1511.07122*, 2015.
<https://arxiv.org/abs/1511.07122>
545. H. Yu and B. Wilamowski. Levenberg–Marquardt training. *Industrial Electronics Handbook*, 5(12), 1, 2011.
546. L. Yu, W. Zhang, J. Wang, and Y. Yu. SeqGAN: Sequence Generative Adversarial Nets with Policy Gradient. *AAAI Conference*, pp. 2852–2858, 2017.
547. W. Yu, W. Cheng, C. Aggarwal, K. Zhang, H. Chen, and WeiWang. NetWalk: A flexible deep embedding approach for anomaly detection in dynamic networks, *ACM KDD Conference*, 2018.
548. W. Yu, C. Zheng, W. Cheng, C. Aggarwal, D. Song, B. Zong, H. Chen, and W. Wang. Learning deep network representations with adversarially regularized autoencoders. *ACM KDD Conference*, 2018.
549. S. Zagoruyko and N. Komodakis. Wide residual networks. *arXiv:1605.07146*, 2016.
<https://arxiv.org/abs/1605.07146>
550. W. Zaremba and I. Sutskever. Reinforcement learning neural turing machines. *arXiv:1505.00521*, 2015.
551. W. Zaremba, T. Mikolov, A. Joulin, and R. Fergus. Learning simple algorithms from examples. *ICML Conference*, pp. 421–429, 2016.
552. W. Zaremba, I. Sutskever, and O. Vinyals. Recurrent neural network regularization. *arXiv:1409.2329*, 2014.
553. M. Zeiler. ADADELTA: an adaptive learning rate method. *arXiv:1212.5701*, 2012.
<https://arxiv.org/abs/1212.5701>
554. M. Zeiler, D. Krishnan, G. Taylor, and R. Fergus. Deconvolutional networks. *Computer Vision and Pattern Recognition (CVPR)*, pp. 2528–2535, 2010.

555. M. Zeiler, G. Taylor, and R. Fergus. Adaptive deconvolutional networks for mid and high level feature learning. *IEEE International Conference on Computer Vision (ICCV)*, pp. 2018–2025, 2011.
556. M. Zeiler and R. Fergus. Visualizing and understanding convolutional networks. *European Conference on Computer Vision, Springer*, pp. 818–833, 2013.
557. C. Zhang, S. Bengio, M. Hardt, B. Recht, and O. Vinyals. Understanding deep learning requires rethinking generalization. *arXiv:1611.03530*.
<https://arxiv.org/abs/1611.03530>
558. D. Zhang, Z.-H. Zhou, and S. Chen. Non-negative matrix factorization on kernels. *Trends in Artificial Intelligence*, pp. 404–412, 2006.
559. L. Zhang, C. Aggarwal, and G.-J. Qi. Stock Price Prediction via Discovering Multi-Frequency Trading Patterns. *ACM KDD Conference*, 2017.
560. S. Zhang, L. Yao, and A. Sun. Deep learning based recommender system: A survey and new perspectives. *arXiv:1707.07435*, 2017.
<https://arxiv.org/abs/1707.07435>
561. X. Zhang, J. Zhao, and Y. LeCun. Character-level convolutional networks for text classification. *NIPS Conference*, pp. 649–657, 2015.
562. J. Zhao, M. Mathieu, and Y. LeCun. Energy-based generative adversarial network. *arXiv:1609.03126*, 2016.
<https://arxiv.org/abs/1609.03126>
563. V. Zhong, C. Xiong, and R. Socher. Seq2SQL: Generating structured queries from natural language using reinforcement learning. *arXiv:1709.00103*, 2017.
<https://arxiv.org/abs/1709.00103>
564. C. Zhou and R. Paffenroth. Anomaly detection with robust deep autoencoders. *ACM KDD Conference*, pp. 665–674, 2017.
565. M. Zhou, Z. Ding, J. Tang, and D. Yin. Micro Behaviors: A new perspective in e-commerce recommender systems. *WSDM Conference*, 2018.
566. Z.-H. Zhou. Ensemble methods: Foundations and algorithms. *CRC Press*, 2012.
567. Z.-H. Zhou, J. Wu, and W. Tang. Ensembling neural networks: many could be better than all. *Artificial Intelligence*, 137(1–2), pp. 239–263, 2002.
568. C. Zitnick and P. Dollar. Edge Boxes: Locating object proposals from edges. *ECCV*, pp. 391–405, 2014.
569. B. Zoph and Q. V. Le. Neural architecture search with reinforcement learning. *arXiv:1611.01578*, 2016.
<https://arxiv.org/abs/1611.01578>

570. <https://deeplearning4j.org/>
571. <http://caffe.berkeleyvision.org/>
572. <http://torch.ch/>
573. <http://deeplearning.net/software/theano/>
574. <https://www.tensorflow.org/>
575. <https://keras.io/>
576. <https://lasagne.readthedocs.io/en/latest/>
577. http://www.netflixprize.com/community/topic_1537.html
578. <http://deeplearning.net/tutorial/lstm.html>
579. <https://arxiv.org/abs/1609.08144>
580. <https://github.com/karpathy/char-rnn>
581. <http://www.image-net.org/>
582. <http://www.image-net.org/challenges/LSVRC/>
583. <https://www.cs.toronto.edu/~kriz/cifar.html>
584. <http://code.google.com/p/cuda-convnet/>
585. http://caffe.berkeleyvision.org/gathered/examples/feature_extraction.html
586. <https://github.com/caffe2/caffe2/wiki/Model-Zoo>
587. <http://scikit-learn.org/>
588. <http://clic.cimec.unitn.it/composes/toolkit/>
589. <https://github.com/stanfordnlp/GloVe>
590. <https://deeplearning4j.org/>
591. <https://code.google.com/archive/p/word2vec/>
592. <https://www.tensorflow.org/tutorials/word2vec/>
593. <https://github.com/aditya-grover/node2vec>
594. <https://www.wikipedia.org/>
595. <https://github.com/caglar/autoencoders>
596. <https://github.com/y0ast>
597. <https://github.com/fastforwardlabs/vae-tf/tree/master>
598. <https://science.education.nih.gov/supplements/webversions/BrainAddiction/guide/lesson2-1.html>
599. <https://www.ibm.com/us-en/marketplace/deep-learning-platform>
600. <https://www.coursera.org/learn/neural-networks>

601. <https://archive.ics.uci.edu/ml/datasets.html>
602. <http://www.bbc.com/news/technology-35785875>
603. <https://deepmind.com/blog/exploring-mysteries-alphago/>
604. <http://selfdrivingcars.mit.edu/>
605. <http://karpathy.github.io/2016/05/31/rl/>
606. <https://github.com/hughperkins/kgsgo-dataset-preprocessor>
607. <https://www.wired.com/2016/03/two-moves-alphago-lee-sedol-redefined-future/>
608. <https://qz.com/639952/googles-ai-won-the-game-go-by-defying-millennia-of-basic-human-instinct/>
609. <http://www.mujioco.org/>
610. <https://sites.google.com/site/gaepapersupp/home>
611. <https://drive.google.com/file/d/0B9raQzOpizn1TkRI-a241ZnBEcjQ/view>
612. <https://www.youtube.com/watch?v=1L0TKZQcUtA&list=PLrAXtmErZgOeiKm4sgN0kn-GvNjby9efdf>
613. <https://openai.com/>
614. <http://jaberg.github.io/hyperopt/>
615. <http://www.cs.ubc.ca/labs/beta/Projects/SMAC/>
616. <https://github.com/JasperSnoek/spearmint>
617. <https://deeplearning4j.org/lstm>
618. <http://colah.github.io/posts/2015-08-Understanding-LSTMs/>
619. <https://www.youtube.com/watch?v=2pWv7GOvuf0>
620. <https://gym.openai.com>
621. <https://universe.openai.com>
622. <https://github.com/facebookresearch/ParlAI>
623. <https://github.com/openai/baselines>
624. <https://github.com/carpedm20/deep-rl-tensorflow>
625. <https://github.com/matthiasplappert/keras-rl>
626. <http://apollo.auto/>
627. <https://github.com/Element-Research/rnn/blob/master/examples/>

628. <https://github.com/lmthang/nmt.matlab>
629. <https://github.com/carpedm20/NTM-tensorflow>
630. <https://github.com/camigord/Neural-Turing-Machine>
631. <https://github.com/SigmaQuan/NTM-Keras>
632. <https://github.com/snipsco/ntm-lasagne>
633. <https://github.com/kaishengtai/torch-ntm>
634. <https://github.com/facebook/MemNN>
635. <https://github.com/carpedm20/MemN2N-tensorflow>
636. <https://github.com/YerevaNN/Dynamic-memory-networks-in-Theano>
637. <https://github.com/carpedm20/DCGAN-tensorflow>
638. <https://github.com/carpedm20>
639. <https://github.com/jacobgil/keras-dcgan>
640. <https://github.com/wiseodd/generative-models>
641. <https://github.com/paarthneekhara/text-to-image>
642. <http://horatio.cs.nyu.edu/mit/tiny/data/>
643. <https://developer.nvidia.com/cudnn>
644. <http://www.nvidia.com/object/machine-learning.html>
645. [https://developer.nvidia.com/deep-learning-frames-works](https://developer.nvidia.com/deep-learning-frameworks)

Предметный указатель

A

AdaDelta, 230
AdaGrad, 227
Adaline, 111
Adam, 231
AlexNet, 519
AlphaGo, 570; 609
AlphaGo Zero, 613
AlphaZero, 615

B

BFGS, 243
 с ограниченным использованием
 памяти, 244
BPTT, 81; 436

C

Caffe, 538
CBOW, 152; 153
CGAN, 673
CIFAR-10, 566
CNN, 485
CUDA, 258

D

DCGAN, 671
DeepWalk, 172
DenseNet, 533; 536

F

FC7, 521; 538

G

GAN, 89; 145; 336; 664
GoogLeNet, 525; 528
GPU, 254
GRU, 458

I

ILSVRC, 93
ImageNet, 92; 487; 537; 566
Inception, 528

L

L-BFGS, 244
LeNet-5, 82; 505; 518
LSTM, 453; 532

M

Maxout, 222
MCMC, 383
MIMIC, 263
MNIST, 90; 566
MSE, 282

N

node2vec, 172

P

PCA, 133; 141; 212
PLSA, 407

Q

QA-система, 466
Q-сеть, 586; 596
Q-функция, 585

- R**
- RBF, 77; 345
 - тренировка, 349
 - RBM, 78; 369; 386
 - ReLU, 41; 46; 500
 - с утечкой, 221
 - ResNet, 531
 - ResNext, 536
 - RMSProp, 228
 - RNN, 425; 428; 460
- S**
- SARSA, 591
 - SGNS, 162
 - SVD, 129; 151
 - SVM, 37
 - Уэстона — Уоткинса, 121
- T**
- TD-обучение, 590
 - TRPO, 632
 - t-SNE, 140
- V**
- VGG, 524
- W**
- word2vec, 151
 - WordNet, 92; 126
- Z**
- ZFNet, 523
- A**
- Автокодировщик, 43; 127
 - вариационный, 145; 670
 - обучение, 332
 - глубокий, 138
 - контрактивный, 145
 - неполный, 323
 - разреженный, 143; 321
 - сверточный, 548
 - сверхполный, 321
 - сжимающий, 324
 - шумоподавляющий, 145; 322
 - Авторегрессионная модель, 473
 - Адаптивный линейный нейрон, 111
 - Аксон, 23
 - Актор, 602
 - Алгоритм
 - Q-обучения, 586
 - Бройдена — Флетчера —
 - Гольдфарба — Шанно, 243
 - карманный, 37
 - Анализ главных компонент, 133; 212
 - Аннотирование изображений, 461; 645
 - Анпулинг, 549
 - Ансамблевый метод, 62; 296
 - Аугментация данных, 517
- Б**
- Базовая линия, 600
 - Беспилотный автомобиль, 624
 - Бустинг, 296; 339
 - Бутстрэппинг, 584; 608
 - Бэггинг, 276; 296; 298
- В**
- Валидационные данные, 209
 - Валидационный набор, 141; 275; 285; 306
 - Валидация модели, 291
 - Вариационный кодировщик, 328
 - Ведро моделей, 299
 - Вектор
 - ключей, 658
 - прототипов, 345
 - Векторная квантизация, 680
 - Векторное представление слов, 151
 - Вентиль, 532; 535
 - Вероятностный латентно-семантический анализ, 407
 - Взгляд, 642
 - Взрывной градиент, 63; 215
 - Визуализация признаков, 539
 - градиентная, 541

Визуальная моторика, 618
 Вложение, 151
 Внимание, 637; 639
 жесткое, 645
 мягкое, 646
 Внутренний ковариационный сдвиг, 249
 Возмущение данных, 293
 Воспроизведение опыта, 589
 Временной шаг, 425
 Время обжига, 383
 Выброс, 142
 Выпуклая задача оптимизации, 247
 Вычислительный граф, 25; 51; 182

Г

Гауссовские ядра, 68
 Гауссовский шум, 322
 Генеративная модель, 664
 Генеративно-состязательная сеть, 89;
 145; 336; 638; 664
 тренировка, 666
 условная, 673
 Гессиан, 235
 Гиперболический тангенс, 40
 спрямленный, 46
 Гиперпараметр, 208
 параллелизм, 258
 Гипотеза о поведении муравьев, 569
 Глубокая машина Больцмана, 417
 Глубокая сеть доверия, 418
 Глубокое обучение, 62; 585
 Градиентный спуск, 166; 222
 мини-пакетный стохастический, 203
 по стратегиям, 596
 ранняя остановка, 306
 стохастический, 33; 118
 Граф, 169

Д

Датчик взгляда, 642
 Деконволюционная сеть, 671
 Деконволюция, 514; 549

Дельта-правило, 111
 Дендрит, 23
 Дилемма смещения —
 дисперсии, 274; 278
 Динамическое
 программирование, 183; 187
 Дискриминационная модель, 664
 Дисперсия, 59; 279
 Дифференциальный нейронный компьютер, 655; 663
 Дифференцировочная способность, 525
 Долгая краткосрочная память, 65; 448;
 453; 532
 Дополнение, 495
 половинное, 551
 Дочерняя сеть, 628
 Дробная шаговая свертка, 671
 Дропаут, 276; 301

Е

Емкость модели, 373

Ж

Жадный алгоритм, 574
 Жесткое внимание, 645
 Жидкая машина, 450

З

Зазор, 38
 Замыкающее соединение, 532
 управляемое, 535
 Заполнение матриц, 403
 Затухающий градиент, 63; 215

И

Имитационное обучение, 612; 625
 Имитация отжига, 318
 Импульсный нейрон, 689
 Инициализация
 Глоро, 214
 Ксавье, 214

- И**
Исключение, 301
- К**
Карманный алгоритм, 37
Карта
 активации, 491
 выделенности (салиентности), 541
 признаков, 491; 492
Квазиньютоновское условие, 243
Квантизация, 261
Классификация, 403
 видео, 560
 метод наименьших квадратов, 110
 на уровне предложений, 469
Кoadаптация признаков, 303
Кодирование Хаффмана, 162
Коллаборативная фильтрация, 398
Коммутатор, 552
Конечные разности, 598
Коннекционная временная
 классификация, 479
Конструирование признаков, 84
 автоматическое, 395
 иерархическое, 486; 508
Контрактивный автокодировщик, 174
Контрастная дивергенция, 391
 персистентная, 419
Контрастная оценка шума, 162
Крапчатый шум, 322
Критерий перцептрона, 32; 36; 107; 358
 сглаженный, 109
Кросс-энтропийная потеря, 44
Кусочно-линейные потери, 37; 117
- Л**
Лестничная сеть, 340
Линейно разделимые данные, 34
Логистическая регрессия, 44; 113
 мультиномиальная, 43; 123
Локализация объектов, 555
Локомоция, 617
Лучевой поиск, 433
- М**
Максимальное правдоподобие, 113
Марковский процесс, 576
Марковское случайное поле, 370
Маскирующий шум, 322
Матричная факторизация
 логистическая, 150; 164
 мультиномиальная, 150; 168
Машина Больцмана, 380
 глубокая, 417
 на основе аппроксимации среднего
 поля, 419
 обучение весов, 383
 ограниченная, 369; 386
 каскадная, 413
Машинный перевод, 463; 646
Межзадачное обучение, 687
Мембранный потенциал, 689
Мертвый нейрон, 221
Мета-обучение, 687
Метод
 импульсов, 224
 Нестерова, 226
 квазиньютоновский, 243
 конечных разностей, 598
 крутейшего спуска, 222
 Монте-Карло, 606
 наименьших квадратов, 107
 линейный, 110
 ортогональный, 354
 опорных векторов, 37; 116
 ядерный, 363
 относительного правдоподобия, 599
 рекомендуемых областей, 559
 сопряженных градиентов, 238
 субградиента, 294
 Уидроу — Хоффа, 110
Мешок слов, 152
 непрерывный, 153
Мини-пакет, 165; 192
Миопическое приближение, 595
Многорукий бандит, 572
Множитель Лагранжа, 362

Модальность, 410
Моделирование языка, 79
Модель
 Бернулли, 152
 деформируемых компонент, 564
 Маккаллока — Питтса, 35
 мультиномиальная, 152
 обучения Уидроу — Хоффа, 77
Модель-учитель, 263
Мультимодальные данные, 146; 410
Мультиномиальная логистическая
 регрессия, 43
Мягкое внимание, 646

Н

Насыщение, 68
Недообучение, 289
Нейроморфные вычисления, 690
Нейрон, 23
 импульсный, 689
 мертвый, 221
 смещения, 31
Нейронная
 машина Тьюринга, 638; 655
 сеть, 24
 глубина, 71
 глубокая, 26
 многослойная, 46
 прямого распространения, 29; 46
 радиально-базисных функций, 77
 рекуррентная, 79; 425; 428; 460
 репликаторная, 129
 сверточная, 27; 82; 462; 485
 с внешней памятью, 651
 сумм и произведений, 75
 тренировка, 52
 языковая модель, 125
Неконтролируемое обучение, 309
Неокогнитрон, 27; 82; 487
Нормализация
 локального отклика, 508
 пакетная, 249
 признаков, 211
Норма Фробениуса, 130

О

Обнаружение объектов, 558
Обобщение модели, 25
Обобщенная оценка выгоды, 632
Обратное распространение
 ошибки, 39; 52; 182
 во времени, 81; 435
 усеченное, 437
 контролируемое, 544
 с постактивационными
 переменными, 189
 с преактивационными
 переменными, 193
 через свертки, 511
Обучение
 без учителя, 416
 на основе временных
 разностей, 590
 обучению, 687
 разовое, 686
 с подкреплением, 88; 570; 575; 644
 с продолжением, 317
 с учителем, 417
 энергетически эффективное, 688
Овраг, 232
Ограниченная машина Больцмана, 78;
 369; 386
 каскадная, 413
 обучение, 389
Оптимизация, 247
 без гессиана, 238
Ортогональный метод наименьших
 квадратов, 354
Ослабление веса, 60
Остаточная матрица, 130
Остаточное обучение, 534
Остаточный модуль, 532
Отбеливание, 212
Отбрасывание соединений, 300
Отложенный набор, 275; 287
Относительное правдоподобие, 599
Отрицательное семплирование, 149;
 152; 378

Отсечение градиентов, 234; 446
 Оценка сходства, 649
 Ошибка обобщения, 275

П

Пакетная нормализация, 249; 447
 Параболическая ограничивающая функция, 412
 Парадокс Сейре, 479
 Параллелизм, 258
 Переключатель контекстной валентности, 470
 Перекрестная проверка, 288
 Переносимое обучение, 86; 538
 Переобучение, 50; 57; 271
 Перцептрон, 29; 105
 критерий, 32
 мультиклассовый, 119
 оптимальной стабильности, 37
 Пиксель, 490
 Повышающая дискретизация, 549
 Подвыборка, 276; 298
 Поиск по дереву, 606
 Полносверточная сеть, 549
 Полносвязный слой, 503
 Полнота по Тьюрингу, 81; 427; 661
 Послойная нормализация, 254; 448
 Постактивационное значение, 39
 поэтапное обучение, 276; 317; 318
 Правило
 обучения Сторки, 377
 обучения Хебба, 376
 Преактивационное значение, 39
 Предварительное обучение, 64; 84; 276
 без учителя, 308
 контролируемое, 314
 Прогнозирование, 472
 Проекционная матрица, 533
 Прореживание, 161
 весов, 261
 Пространственно-временные данные, 560
 Пространство действий, 605

Псевдообратная матрица, 354
 Псевдообращение матрицы, 112; 132
 Пулинг, 499; 501
 по максимальному значению, 200

Р

Равновесие Нэша, 89
 Развертка, 598
 Разделение параметров, 319
 Разделяемая матричная факторизация, 410
 Разделяемые веса, 206
 Размытие, 318
 Разовое обучение, 686
 Раннее прекращение обучения, 61
 Ранняя остановка, 276; 306
 Распознавание
 естественного языка, 559
 речи, 479
 рукописного текста, 479
 Регрессия, 25
 логистическая, 113
 мультиномиальная, 123
 метод наименьших квадратов, 61;
 107
 мультиномиальная
 логистическая, 43
 ядерная, 362
 Регуляризация, 402
 L1, 293
 L2, 294
 на основе штрафов, 275; 289
 Тихонова, 60; 290
 Резервуарные вычисления, 481
 Рекомендательная система, 147; 474
 Рекуррентная сеть, 425; 428; 460
 двунаправленная, 439
 многослойная, 442
 обучение, 444
 Рекуррентный блок, 458
 Репараметризация, 329
 Ряд Тейлора, 235

- С**
- Салиентность, 541
 - Самообучающийся робот, 616
 - Самоорганизующаяся карта
 - Кохонена, 681
 - Свертка, 82; 486; 491
 - дробная, 514
 - дробная шаговая, 554
 - расширенная, 554
 - транспонированная, 514; 549
 - шаг, 498
 - Сверточная нейронная сеть, 82; 462; 485
 - структура, 489
 - тренировка, 510
 - Связывание весов, 133
 - Сдвиг распределения, 630
 - Седловая точка, 244; 668
 - Семантическое хеширование, 420
 - Семплирование
 - методом Монте-Карло марковских цепей, 383
 - по Гиббсу, 383
 - Сентимент-анализ, 469
 - Сеточный поиск, 209
 - Сеть
 - обработки взгляда, 642
 - обращения свертки, 671
 - радиально-базисных функций, 345
 - с памятью, 638
 - стратегий, 596; 611
 - Хопфилда, 78; 370
 - емкость, 377
 - обучение, 376
 - Сигмоида, 39; 40
 - производная, 45
 - Сигмоидная сеть доверия, 418
 - Сигнум, 40
 - Синапс, 23
 - Сингулярное разложение, 43; 129
 - усеченное, 130
 - Синсет, 92
 - Сквозное распознавание речи, 479
 - Скип-грамма, 152; 157
 - с отрицательным семплированием, 162
 - Скорость обучения, 33; 223; 227
 - Скрытый слой, 29; 46
 - Слой внимания, 648
 - Смещение, 31; 59; 279
 - Снижение размерности, 394
 - Собственный вектор, 212
 - Соединение с пропуском слоев, 532
 - Сопряженный градиент, 238
 - Соревновательное обучение, 679
 - Состязательное обучение, 639
 - Спайковый нейрон, 689
 - Спектральный радиус, 451
 - Среднеквадратическая ошибка, 282
 - Статистическая сумма, 381
 - Стохастическая глубина, 536
 - Страйд, 498
 - Субградиент, 294
 - Субдискретизация, 499; 501
- Т**
- Тематическое моделирование, 407
 - Теорема
 - Ковера, 346; 452
 - о представителе, 364
 - Тепловое равновесие, 383
 - Тестовый набор, 285
 - Точка признака, 620
 - Трансляционная инвариантность, 502
 - Транспонированная свертка, 549; 671
 - Тренировочные данные, 24; 285
- У**
- Управляемый рекуррентный блок, 458; 532
 - Уравнение Беллмана, 586
 - Условие взаимной сопряженности, 239
 - Усреднение Поляка, 246
 - Утес, 219

Ф

- Факторизация, 127
- Фильтр, 491
 - инвертированный, 512
- Функция
 - sign, 30; 40
 - производная, 45
 - Softmax, 42; 123; 125; 196; 435
 - tanh, 40
 - производная, 46
 - XOR, 71
 - активации, 30; 39
 - выгоды, 604
 - знаковая, 40
 - значения, 602
 - кусочно-линейная, 44
 - ослабления, 683
 - параболическая
 - ограничивающая, 412
 - потерь, 32; 39; 43; 201
 - двоичная, 36
 - кросс-энтропийная, 197
 - кусочно-линейная, 358
 - сглаженная суррогатная, 36
 - сжатия, 68; 74
 - энергии сети, 372

Ц

- Центрирование по среднему, 211
- Цепное правило, 183

Ч

- Частичное обучение, 316
- Чередование слоев, 504
- Чувствительность, 680

Ш

- Широкая остаточная сеть, 535
- Шум, 279; 322

Э

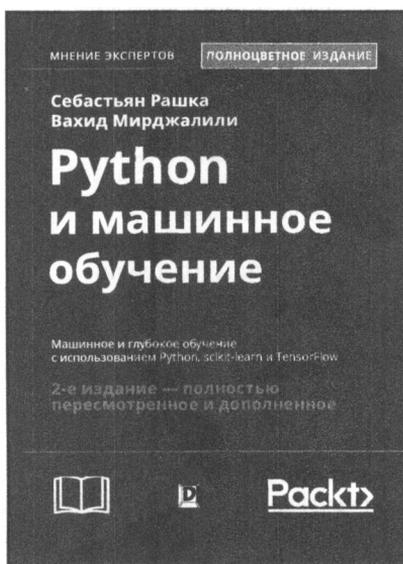
- Эквивариантность, 493
- Эпоха, 33
- Эхо-сеть, 447; 450

Я

- Ядерный метод, 362
- Ядро, 491
- Языковое моделирование, 430; 432
 - условное, 460
- Якобиан, 216

PYTHON И МАШИННОЕ ОБУЧЕНИЕ МАШИННОЕ И ГЛУБОКОЕ ОБУЧЕНИЕ С ИСПОЛЬЗОВАНИЕМ PYTHON, SCIKIT-LEARN И TENSORFLOW, 2-Е ИЗДАНИЕ

**Себастьян Рашка
и Вахид Мирджалили**



www.dialektika.com

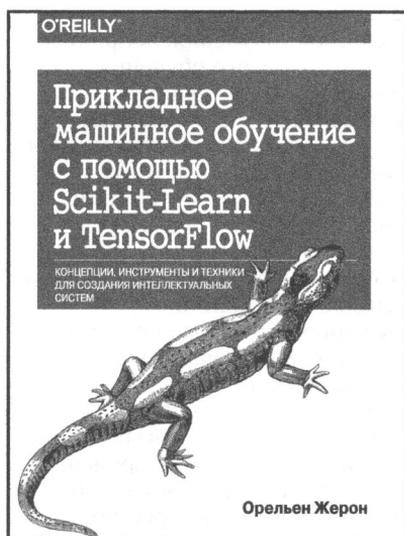
Машинное обучение поглощает мир программного обеспечения, и теперь глубокое обучение расширяет машинное обучение. Освойте и работайте с передовыми технологиями машинного обучения, нейронных сетей и глубокого обучения с помощью 2-го издания бестселлера Себастьяна Рашки. Будучи основательно обновленной с учетом самых последних библиотек Python с открытым кодом, эта книга предлагает практические знания и приемы, которые необходимы для создания и содействия машинному обучению, глубокому обучению и современному анализу данных. Если вы читали 1-е издание книги, то вам доставит удовольствие найти новый баланс классических идей и современных знаний в машинном обучении. Каждая глава была серьезно обновлена, и появились новые главы по ключевым технологиям. У вас будет возможность изучить и поработать с TensorFlow более вдумчиво, нежели ранее, а также получить важнейший охват библиотеки для нейронных сетей Keras наряду с самыми свежими обновлениями библиотеки scikit-learn.

ISBN 978-5-907114-52-4

в продаже

ПРИКЛАДНОЕ МАШИННОЕ ОБУЧЕНИЕ С ПОМОЩЬЮ SCIKIT-LEARN И TENSORFLOW

Орельен Жерон



www.dialektika.com

Благодаря серии недавних достижений глубокое обучение значительно усилило всю область машинного обучения. В наше время даже программисты, почти ничего не знающие об этой технологии, могут использовать простые и эффективные инструменты для реализации программ, которые способны обучаться на основе данных. В настоящем практическом руководстве показано, что и как следует делать. За счет применения конкретных примеров, минимума теории и двух фреймворков Python производственного уровня — Scikit-Learn и TensorFlow — автор книги Орельен Жерон поможет вам получить интуитивное представление о концепциях и инструментах, предназначенных для построения интеллектуальных систем. Вы узнаете о ряде приемов, начав с простой линейной регрессии и постепенно добравшись до глубоких нейронных сетей. Учитывая наличие в каждой главе упражнений, призванных закрепить то, чему вы научились, для начала работы нужен лишь опыт программирования.

ISBN 978-5-9500296-2-2

в продаже

ГЛУБОКОЕ ОБУЧЕНИЕ ГОТОВЫЕ РЕШЕНИЯ

Давид Осинга



www.williamspublishing.com

Благодаря готовым примерам, приведенным в книге, вы научитесь решать задачи, связанные с классификацией и генерированием текста, изображений и музыки. В каждой главе описывается несколько решений, объединяемых в единый проект, например приложение, реализующее тренировку музыкальной рекомендательной системы. Также имеется глава с описанием методик, которые в случае необходимости помогут выполнить отладку нейронной сети. Основные темы книги:

- использование векторных представлений слов для вычисления схожести текстов;
- построение рекомендательной системы фильмов на основе ссылок в Википедии;
- визуализация внутренних состояний нейронной сети;
- создание модели, рекомендующей эмодзи для фрагментов текста;
- повторное использование предварительно обученных сетей для создания службы обратного поиска изображений;
- генерирование пиктограмм с помощью генеративно-состязательных сетей (GAN), автокодировщиков и рекуррентных сетей (RNN);
- распознавание музыкальных жанров и индексирование коллекций песен.

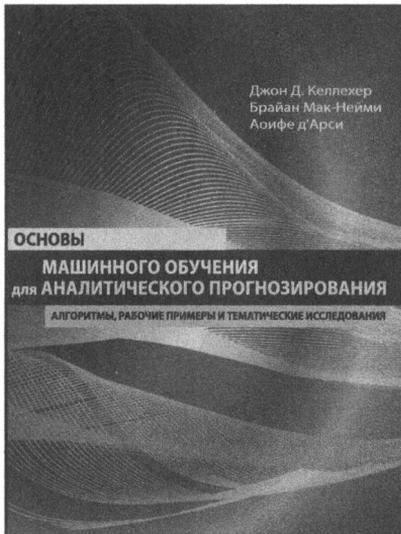
ISBN: 978-5-907144-50-7

в продаже

ОСНОВЫ МАШИННОГО ОБУЧЕНИЯ ДЛЯ АНАЛИТИЧЕСКОГО ПРОГНОЗИРОВАНИЯ

АЛГОРИТМЫ, РАБОЧИЕ ПРИМЕРЫ И ТЕМАТИЧЕСКИЕ
ИССЛЕДОВАНИЯ

**Джон Д. Келлехер
Брайан Мак-Нейми
Аоифе д'Арси**



www.williamspublishing.com

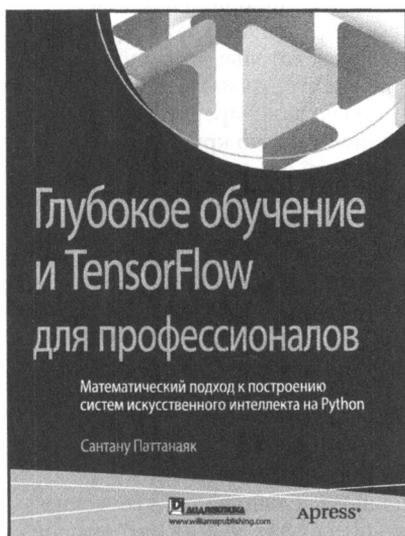
Книга представляет собой учебник по машинному обучению с акцентом на коммерческие приложения. Она предлагает подробное описание наиболее важных подходов к машинному обучению, используемых в интеллектуальном анализе данных, охватывающих как теоретические концепции, так и практические приложения. Формальный математический материал дополняется пояснительными примерами, а примеры исследований иллюстрируют применение этих моделей в более широком контексте бизнеса. В книге рассмотрены информационное обучение, обучение на основе сходства, вероятностное обучение и обучение на основе ошибок. Описанию каждого из этих подходов предшествует объяснение основополагающей концепции, за которой следуют математические модели и алгоритмы, иллюстрированные подробными рабочими примерами.

ISBN 978-5-6040044-9-4

в продаже

ГЛУБОКОЕ ОБУЧЕНИЕ И TENSORFLOW ДЛЯ ПРОФЕССИОНАЛОВ

Сантану Паттанаяк



www.williamspublishing.com

Данная книга представляет собой углубленное практическое руководство, которое позволит читателям освоить методы глубокого обучения на уровне, достаточном для развертывания готовых решений. Прочитав книгу, вы сможете быстро приступить к работе с библиотекой TensorFlow и заняться оптимизацией архитектур глубокого обучения. Весь программный код доступен в виде блокнотов iPython и сценариев, позволяющих с легкостью воспроизводить примеры и экспериментировать с ними.

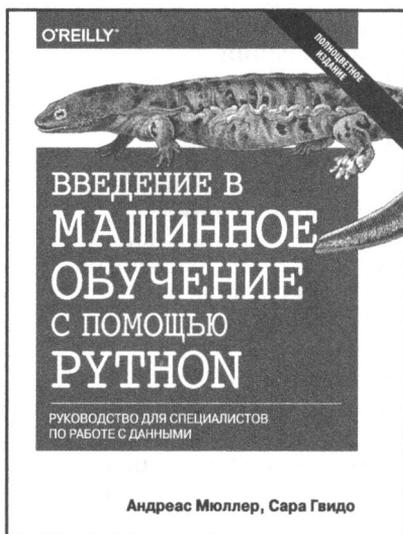
Благодаря этой книге вы:

- овладеете полным стеком технологий глубокого обучения с использованием TensorFlow и получите необходимую для этого математическую подготовку;
- научитесь развертывать сложные приложения глубокого обучения в производственной среде с помощью TensorFlow;
- сможете проводить исследования в области глубокого обучения и выполнять самостоятельные эксперименты в TensorFlow.

ISBN: 978-5-907144-25-5 в продаже

ВВЕДЕНИЕ В МАШИННОЕ ОБУЧЕНИЕ С ПОМОЩЬЮ PYTHON РУКОВОДСТВО ДЛЯ СПЕЦИАЛИСТОВ ПО РАБОТЕ С ДАННЫМИ

**Андреас Мюллер
Сара Гвидо**



www.williamspublishing.com

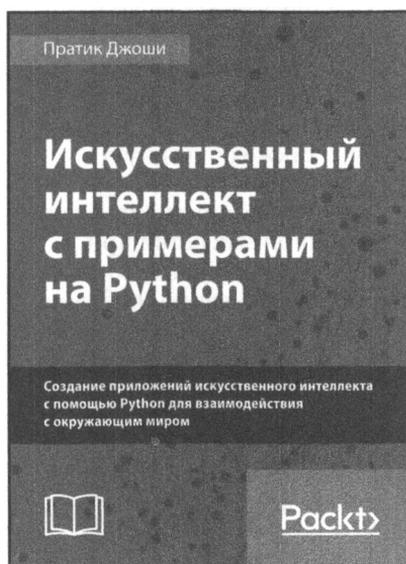
Эта полноцветная книга — отличный источник информации для каждого, кто собирается использовать машинное обучение на практике. Ныне машинное обучение стало неотъемлемой частью различных коммерческих и исследовательских проектов, и не следует думать, что эта область — прерогатива исключительно крупных компаний с мощными командами аналитиков. Эта книга научит вас практическим способам построения систем МО, даже если вы еще новичок в этой области. В ней подробно объясняются все этапы, необходимые для создания успешного проекта машинного обучения, с использованием языка Python и библиотек scikit-learn, NumPy и matplotlib. Авторы сосредоточили свое внимание исключительно на практических аспектах применения алгоритмов машинного обучения, оставив за рамками книги их математическое обоснование. Данная книга адресована специалистам, решающим реальные задачи, а поскольку область применения методов МО практически безгранична, прочитав эту книгу, вы сможете собственными силами построить действующую систему машинного обучения в любой научной или коммерческой сфере.

ISBN 978-5-9908910-8-1

в продаже

ИСКУССТВЕННЫЙ ИНТЕЛЛЕКТ С ПРИМЕРАМИ НА PYTHON

Пратик Джоши



www.dialektika.com

В этой книге исследуются различные сценарии применения искусственного интеллекта. Вначале рассматриваются общие концепции искусственного интеллекта, после чего обсуждаются более сложные темы, такие как предельно случайные леса, скрытые марковские модели, генетические алгоритмы, сверточные нейронные сети и др. Вы узнаете о том, как принимать обоснованные решения при выборе необходимых алгоритмов, а также о том, как реализовывать эти алгоритмы на языке Python для достижения наилучших результатов.

Основные темы книги:

- различные методы классификации и регрессии данных;
- создание интеллектуальных рекомендательных систем;
- логическое программирование и способы его применения;
- построение автоматизированных систем распознавания речи;
- основы эвристического поиска и генетического программирования;
- разработка игр с использованием искусственного интеллекта;
- обучение с подкреплением;
- алгоритмы глубокого обучения и создание приложений на их основе.

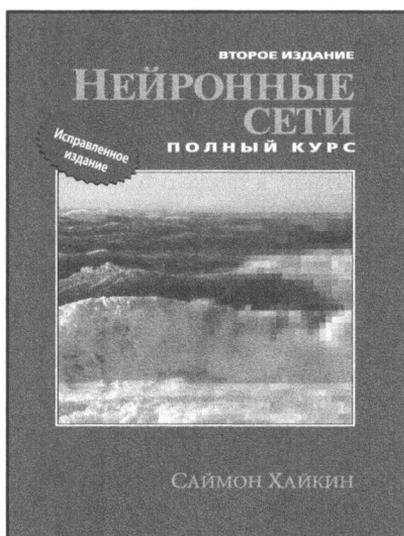
ISBN: 978-5-907114-41-8

в продаже

НЕЙРОННЫЕ СЕТИ: ПОЛНЫЙ КУРС

2-Е ИЗДАНИЕ

Саймон Хайкин



www.dialektika.com

В книге рассматриваются основные парадигмы искусственных нейронных сетей. Представленный материал содержит строгое математическое обоснование всех нейросетевых парадигм, иллюстрируется примерами, описанием компьютерных экспериментов, содержит множество практических задач, а также обширную библиографию.

В книге также анализируется роль нейронных сетей при решении задач распознавания образов, управления и обработки сигналов. Структура книги очень удобна для разработки курсов обучения нейронным сетям и интеллектуальным вычислениям.

Книга будет полезна для инженеров, специалистов в области компьютерных наук, физиков и специалистов в других областях, а также для всех тех, кто интересуется искусственными нейронными сетями.

ISBN 978-5-907144-22-4

в продаже

Чару Аггарвал

Нейронные сети и глубокое обучение

Учебный курс

В книге рассматриваются как классические, так и современные модели глубокого обучения. Главы книги можно разбить на три группы.

- **Основы нейронных сетей.** Суть многих традиционных моделей машинного обучения можно понять, рассматривая их как частные случаи нейронных сетей. В первых двух главах основной упор сделан на понимании взаимосвязи традиционного машинного обучения и нейронных сетей. Будет показано, что метод опорных векторов, линейная и логистическая регрессия, сингулярное разложение, факторизация матриц и рекомендательные системы являются именно такими частными случаями. Наряду с ними рассматриваются и такие сравнительно новые методы конструирования признаков, как word2vec.
- **Фундаментальные понятия нейронных сетей.** Главы 3 и 4 посвящены подробному обсуждению процессов тренировки и регуляризации нейронных сетей. В главах 5 и 6 рассмотрены сети радиально-базисных функций (RBF) и ограниченные машины Больцмана.
- **Дополнительные вопросы нейронных сетей.** В главах 7 и 8 обсуждаются рекуррентные и сверточные нейронные сети. Главы 9 и 10 посвящены более сложным темам, таким как глубокое обучение с подкреплением, нейронные машины Тьюринга, самоорганизующиеся карты Кохонена и генеративно-состязательные сети.

Книга предназначена для студентов старших курсов, исследователей и специалистов-практиков. Там, где это возможно, автор обращает особое внимание на прикладные аспекты использования каждого класса методов.



Чару Аггарвал — заслуженный научный сотрудник (DRSM) исследовательского центра IBM имени Томаса Дж. Уотсона в г. Йорктаун Хайтс, штат Нью-Йорк. Базовое образование в области компьютерных наук получил в Индийском технологическом институте в г. Канпур, который окончил в 1993 г., а в 1996 г. получил степень доктора философии в Массачусетском технологическом институте. Автор свыше 350 публикаций и более чем 80 зарегистрированных патентов. Выпустил 18 книг, в том числе учебники по анализу данных и рекомендательным системам. Ввиду коммерческой ценности его патентов трижды получал звание заслуженного изобретателя компании IBM. Удостоен ряда международных наград, включая *EDBT Test of Time Award* (2014) и *IEEE ICDM Research Contributions Award* (2015). Главный редактор журналов *ACM Transactions on Knowledge Discovery from Data* и *ACM SIGKDD Explorations*.

Категория: искусственный интеллект/нейронные сети

Уровень: для пользователей средней и высокой квалификации

 **ДАЛЕКТИКА**

www.dialektika.com

 **Springer**

ISBN: 978-5-907203-01-3



9 785907 203013