

«Книга написана ведущими специалистами в этой области и представляет собой единственное полное изложение предмета».

**— Илон Маск,
сооснователь компаний Tesla и SpaceX**



DMK
издательство

Я. Гудфеллоу, И. Бенджио, А. Курвилль

Глубокое обучение

УДК 004.85
ББК 32.971.3
Г93

Гудфеллоу Я., Бенджио И., Курвилль А.

Г93 Глубокое обучение / пер. с англ. А. А. Слинкина. – 2-е изд., испр. – М.: ДМК Пресс, 2018. – 652 с.: цв. ил.

ISBN 978-5-97060-618-6

Глубокое обучение — это вид машинного обучения, наделяющий компьютеры способностью учиться на опыте и понимать мир в терминах иерархии концепций. Книга содержит математические и концептуальные основы линейной алгебры, теории вероятностей и теории информации, численных расчетов и машинного обучения в том объеме, который необходим для понимания материала. Описываются приемы глубокого обучения, применяемые на практике, в том числе глубокие сети прямого распространения, регуляризация, алгоритмы оптимизации, сверточные сети, моделирование последовательностей и др. Рассматриваются такие приложения, как обработка естественных языков, распознавание речи, компьютерное зрение, онлайн-рекомендательные системы, биоинформатика и видеоигры.

Издание предназначено студентам вузов и аспирантам, а также опытным программистам, которые хотели бы применить глубокое обучение в составе своих продуктов или платформ.

УДК 004.85
ББК 32.971.3

Права на издание книги на русском языке предоставлены агентством Александра Корженевского.

Все права защищены. Любая часть этой книги не может быть воспроизведена в какой бы то ни было форме и какими бы то ни было средствами без письменного разрешения владельцев авторских прав.

Материал, изложенный в данной книге, многократно проверен. Но поскольку вероятность технических ошибок все равно существует, издательство не может гарантировать абсолютную точность и правильность приводимых сведений. В связи с этим издательство не несет ответственности за возможные ошибки, связанные с использованием книги.

ISBN 978-1-491-93799-0 (анг.)
ISBN 978-5-97060-618-6 (рус.)

© 2017 Massachusetts Institute of Technology
© Оформление, издание, перевод, ДМК Пресс, 2018

Содержание

Веб-сайт	14
Благодарности	15
Обозначения	18
 Глава 1. Введение.....	 21
1.1. На кого ориентирована эта книга	29
1.2. Исторические тенденции в машинном обучении	29
1.2.1. Нейронные сети: разные названия и переменчивая фортуна	30
1.2.2. Увеличение размера набора данных	36
1.2.3. Увеличение размера моделей	36
1.2.4. Повышение точности и сложности и расширение круга задач	40
 Часть I. Основы прикладной математики и машинного обучения	 43
 Глава 2. Линейная алгебра	 44
2.1. Скаляры, векторы, матрицы и тензоры	44
2.2. Умножение матриц и векторов	46
2.3. Единичная и обратная матрица	47
2.4. Линейная зависимость и линейная оболочка	48
2.5. Нормы	50
2.6. Специальные виды матриц и векторов	51
2.7. Спектральное разложение матрицы	52
2.8. Сингулярное разложение	54
2.9. Псевдообратная матрица Мура–Пенроуза	55
2.10. Оператор следа	56
2.11. Определитель	56
2.12. Пример: метод главных компонент	57
 Глава 3. Теория вероятностей и теория информации	 61
3.1. Зачем нужна вероятность?	61
3.2. Случайные величины	63
3.3. Распределения вероятности	63
3.3.1. Дискретные случайные величины и функции вероятности	64
3.3.2. Непрерывные случайные величины и функции плотности вероятности	64
3.4. Маргинальное распределение вероятности	65
3.5. Условная вероятность	65
3.6. Цепное правило	66
3.7. Независимость и условная независимость	66
3.8. Математическое ожидание, дисперсия и ковариация	66
3.9. Часто встречающиеся распределения вероятности	68
3.9.1. Распределение Бернулли	68

3.9.2. Категориальное распределение.....	68
3.9.3. Нормальное распределение.....	69
3.9.4. Экспоненциальное распределение и распределение Лапласа	70
3.9.5. Распределение Дирака и эмпирическое распределение.....	71
3.9.6. Смеси распределений	71
3.10. Полезные свойства употребительных функций	73
3.11. Правило Байеса	74
3.12. Технические детали непрерывных величин	75
3.13. Теория информации.....	76
3.14. Структурные вероятностные модели.....	78
Глава 4. Численные методы	82
4.1. Переполнение и потеря значимости.....	82
4.2. Плохая обусловленность	83
4.3. Оптимизация градиентным методом	84
4.3.1. Не только градиент: матрицы Якоби и Гессе.....	86
4.4. Оптимизация с ограничениями	92
4.5. Пример: линейный метод наименьших квадратов	94
Глава 5. Основы машинного обучения.....	96
5.1. Алгоритмы обучения.....	97
5.1.1. Задача Т	97
5.1.2. Мера качества Р.....	100
5.1.3. Опыт Е	101
5.1.4. Пример: линейная регрессия	103
5.2. Емкость, переобучение и недообучение	105
5.2.1. Теорема об отсутствии бесплатных завтраков.....	110
5.2.2. Регуляризация	112
5.3. Гиперпараметры и контрольные наборы.....	114
5.3.1. Перекрестная проверка.....	115
5.4. Оценки, смещение и дисперсия.....	115
5.4.1. Точечное оценивание	116
5.4.2. Смещение.....	117
5.4.3. Дисперсия и стандартная ошибка.....	119
5.4.4. Поиск компромисса между смещением и дисперсией для минимизации среднеквадратической ошибки.....	121
5.4.5. Состоятельность.....	122
5.5. Оценка максимального правдоподобия	122
5.5.1. Условное логарифмическое правдоподобие и среднеквадратическая ошибка.....	123
5.5.2. Свойства максимального правдоподобия	125
5.6. Байесовская статистика	125
5.6.1. Оценка апостериорного максимума (MAP).....	128
5.7. Алгоритмы обучения с учителем.....	129
5.7.1. Вероятностное обучение с учителем.....	129

5.7.2. Метод опорных векторов.....	130
5.7.3. Другие простые алгоритмы обучения с учителем	132
5.8. Алгоритмы обучения без учителя	134
5.8.1. Метод главных компонент.....	135
5.8.2. Кластеризация методом k средних	137
5.9. Стохастический градиентный спуск	138
5.10. Построение алгоритма машинного обучения	140
5.11. Проблемы, требующие глубокого обучения.....	141
5.11.1. Проклятие размерности	141
5.11.2. Регуляризация для достижения локального постоянства и гладкости.....	142
5.11.3. Обучение многообразий.....	145
Часть II. Глубокие сети: современные подходы.....	149
Глава 6. Глубокие сети прямого распространения	150
6.1. Пример: обучение XOR	152
6.2. Обучение градиентными методами	157
6.2.1. Функции стоимости.....	158
6.2.2. Выходные блоки	160
6.3. Скрытые блоки.....	169
6.3.1. Блоки линейной ректификации и их обобщения.....	170
6.3.2. Логистическая сигмоида и гиперболический тангенс	171
6.3.3. Другие скрытые блоки	172
6.4. Проектирование архитектуры	173
6.4.1. Свойства универсальной аппроксимации и глубина	174
6.4.2. Другие архитектурные подходы	177
6.5. Обратное распространение и другие алгоритмы дифференцирования.....	179
6.5.1. Графы вычислений	179
6.5.2. Правило дифференцирования сложной функции	181
6.5.3. Рекурсивное применение правила дифференцирования сложной функции для получения алгоритма обратного распространения.....	182
6.5.4. Вычисление обратного распространения в полносвязном МСП	185
6.5.5. Символьно-символьные производные	186
6.5.6. Общий алгоритм обратного распространения.....	188
6.5.7. Пример: применение обратного распространения к обучению МСП.....	191
6.5.8. Осложнения	192
6.5.9. Дифференцирование за пределами сообщества глубокого обучения	193
6.5.10. Производные высшего порядка	195
6.6. Исторические замечания	196
Глава 7. Регуляризация в глубоком обучении	199
7.1. Штрафы по норме параметров	200
7.1.1. Регуляризация параметров по норме L^2	201
7.1.2. L^1 -регуляризация.....	204
7.2. Штраф по норме как оптимизация с ограничениями	206

7.3. Регуляризация и недоопределенные задачи.....	208
7.4. Пополнение набора данных.....	208
7.5. Робастность относительно шума.....	210
7.5.1. Привнесение шума в выходные метки.....	211
7.6. Обучение с частичным привлечением учителя.....	211
7.7. Многозадачное обучение.....	212
7.8. Ранняя остановка.....	213
7.9. Связывание и разделение параметров.....	219
7.9.1. Сверточные нейронные сети.....	220
7.10. Разреженные представления.....	220
7.11. Баггинг и другие ансамблевые методы.....	222
7.12. Прореживание.....	224
7.13. Состязательное обучение.....	232
7.14. Тангенциальное расстояние, алгоритм распространения по касательной и классификатор по касательной к многообразию.....	233
Глава 8. Оптимизация в обучении глубоких моделей.....	237
8.1. Чем обучение отличается от чистой оптимизации.....	237
8.1.1. Минимизация эмпирического риска.....	238
8.1.2. Суррогатные функции потерь и ранняя остановка.....	239
8.1.3. Пакетные и мини-пакетные алгоритмы.....	239
8.2. Проблемы оптимизации нейронных сетей.....	243
8.2.1. Плохая обусловленность.....	243
8.2.2. Локальные минимумы.....	245
8.2.3. Плато, седловые точки и другие плоские участки.....	246
8.2.4. Утесы и резко растущие градиенты.....	248
8.2.5. Долгосрочные зависимости.....	249
8.2.6. Неточные градиенты.....	250
8.2.7. Плохое соответствие между локальной и глобальной структурами.....	250
8.2.8. Теоретические пределы оптимизации.....	252
8.3. Основные алгоритмы.....	253
8.3.1. Стохастический градиентный спуск.....	253
8.3.2. Импульсный метод.....	255
8.3.3. Метод Нестерова.....	258
8.4. Стратегии инициализации параметров.....	258
8.5. Алгоритмы с адаптивной скоростью обучения.....	263
8.5.1. AdaGrad.....	264
8.5.2. RMSProp.....	264
8.5.3. Adam.....	265
8.5.4. Выбор правильного алгоритма оптимизации.....	266
8.6. Приближенные методы второго порядка.....	267
8.6.1. Метод Ньютона.....	267
8.6.2. Метод сопряженных градиентов.....	268
8.6.3. Алгоритм BFGS.....	271
8.7. Стратегии оптимизации и метаалгоритмы.....	272

8.7.1. Пакетная нормировка.....	272
8.7.2. Покоординатный спуск.....	275
8.7.3. Усреднение Поляка.....	276
8.7.4. Предобучение с учителем.....	276
8.7.5. Проектирование моделей с учетом простоты оптимизации.....	279
8.7.6. Методы продолжения и обучение по плану.....	279
Глава 9. Сверточные сети.....	282
9.1. Операция свертки.....	282
9.2. Мотивация.....	284
9.3. Пулинг.....	290
9.4. Свертка и пулинг как бесконечно сильное априорное распределение.....	293
9.5. Варианты базовой функции свертки.....	295
9.6. Структурированный выход.....	304
9.7. Типы данных.....	305
9.8. Эффективные алгоритмы свертки.....	306
9.9. Случайные признаки и признаки, обученные без учителя.....	307
9.10. Нейробиологические основания сверточных сетей.....	308
9.11. Сверточные сети и история глубокого обучения.....	314
Глава 10. Моделирование последовательностей: рекуррентные и рекурсивные сети.....	316
10.1. Развертка графа вычислений.....	317
10.2. Рекуррентные нейронные сети.....	320
10.2.1. Форсирование учителя и сети с рекурсией на выходе.....	323
10.2.2. Вычисление градиента в рекуррентной нейронной сети.....	325
10.2.3. Рекуррентные сети как ориентированные графические модели.....	327
10.2.4. Моделирование контекстно-обусловленных последовательностей с помощью РНС.....	330
10.3. Двухнаправленные РНС.....	332
10.4. Архитектуры кодировщик-декодер или последовательность в последовательность.....	333
10.5. Глубокие рекуррентные сети.....	336
10.6. Рекурсивные нейронные сети.....	337
10.7. Проблема долгосрочных зависимостей.....	339
10.8. Нейронные эхо-сети.....	341
10.9. Блоки с утечками и другие стратегии нескольких временных масштабов.....	343
10.9.1. Добавление прямых связей сквозь время.....	343
10.9.2. Блоки с утечкой и спектр разных временных масштабов.....	343
10.9.3. Удаление связей.....	344
10.10. Долгая краткосрочная память и другие вентильные РНС.....	344
10.10.1. Долгая краткосрочная память.....	345
10.10.2. Другие вентильные РНС.....	347
10.11. Оптимизация в контексте долгосрочных зависимостей.....	348
10.11.1. Отсечение градиентов.....	348

10.11.2. Регуляризация с целью подталкивания информационного потока	350
10.12. Явная память	351
Глава 11. Практическая методология	355
11.1. Показатели качества	356
11.2. Выбор базовой модели по умолчанию	358
11.3. Надо ли собирать дополнительные данные?	359
11.4. Выбор гиперпараметров	360
11.4.1. Ручная настройка гиперпараметров	360
11.4.2. Алгоритмы автоматической оптимизации гиперпараметров	363
11.4.3. Поиск на сетке	364
11.4.4. Случайный поиск	365
11.4.5. Оптимизация гиперпараметров на основе модели	366
11.5. Стратегии отладки	367
11.6. Пример: распознавание нескольких цифр	370
Глава 12. Приложения	373
12.1. Крупномасштабное глубокое обучение	373
12.1.1. Реализации на быстрых CPU	373
12.1.2. Реализации на GPU	374
12.1.3. Крупномасштабные распределенные реализации	376
12.1.4. Сжатие модели	376
12.1.5. Динамическая структура	377
12.1.6. Специализированные аппаратные реализации глубоких сетей	379
12.2. Компьютерное зрение	380
12.2.1. Предобработка	381
12.3. Распознавание речи	385
12.4. Обработка естественных языков	388
12.4.1. n -граммы	388
12.4.2. Нейронные языковые модели	390
12.4.3. Многомерные выходы	391
12.4.4. Комбинирование нейронных языковых моделей с n -граммами	397
12.4.5. Нейронный машинный перевод	397
12.4.6. Историческая справка	401
12.5. Другие приложения	402
12.5.1. Рекомендательные системы	402
12.5.2. Представление знаний, рассуждения и ответы на вопросы	405
Часть III. Исследования по глубокому обучению	409
Глава 13. Линейные факторные модели	411
13.1. Вероятностный РСА и факторный анализ	412
13.2. Анализ независимых компонент (ICA)	413
13.3. Анализ медленных признаков	415
13.4. Разреженное кодирование	417
13.5. Интерпретация РСА в терминах многообразий	419

Глава 14. Автокодировщики	422
14.1. Понижающие автокодировщики	423
14.2. Регуляризированные автокодировщики	423
14.2.1. Разреженные автокодировщики	424
14.2.2. Шумоподавляющие автокодировщики	426
14.2.3. Регуляризация посредством штрафования производных	427
14.3. Репрезентативная способность, размер слоя и глубина	427
14.4. Стохастические кодировщики и декодеры	428
14.5. Шумоподавляющие автокодировщики	429
14.5.1. Сопоставление рейтингов	430
14.6. Обучение многообразий с помощью автокодировщиков	433
14.7. Сжимающие автокодировщики	436
14.8. Предсказательная разреженная декомпозиция	440
14.9. Применения автокодировщиков	441
Глава 15. Обучение представлений	443
15.1. Жадное послойное предобучение без учителя	444
15.1.1. Когда и почему работает предобучение без учителя?	446
15.2. Перенос обучения и адаптация домена	451
15.3. Разделение каузальных факторов с частичным привлечением учителя	454
15.4. Распределенное представление	459
15.5. Экспоненциальный выигрыш от глубины	465
15.6. Ключ к выявлению истинных причин	466
Глава 16. Структурные вероятностные модели в глубоком обучении	469
16.1. Проблема бесструктурного моделирования	470
16.2. Применение графов для описания структуры модели	473
16.2.1. Ориентированные модели	473
16.2.2. Неориентированные модели	475
16.2.3. Статистическая сумма	477
16.2.4. Энергетические модели	478
16.2.5. Разделенность и d-разделенность	480
16.2.6. Преобразование между ориентированными и неориентированными графами	481
16.2.7. Факторные графы	486
16.3. Выборка из графических моделей	487
16.4. Преимущества структурного моделирования	488
16.5. Обучение и зависимости	489
16.6. Вывод и приближенный вывод	490
16.7. Подход глубокого обучения к структурным вероятностным моделям	491
16.7.1. Пример: ограниченная машина Больцмана	492
Глава 17. Методы Монте-Карло	495
17.1. Выборка и методы Монте-Карло	495
17.1.1. Зачем нужна выборка?	495

17.1.2. Основы выборки методом Монте-Карло	495
17.2. Выборка по значимости	497
17.3. Методы Монте-Карло по схеме марковской цепи	499
17.4. Выборка по Гиббсу.....	502
17.5. Проблема перемешивания разделенных мод	503
17.5.1. Применение темперирования для перемешивания мод	506
17.5.2. Глубина может помочь перемешиванию	506

Глава 18. Преодоление трудностей, связанных

со статической суммой.....	508
18.1. Градиент логарифмического правдоподобия	508
18.2. Стохастическая максимизация правдоподобия и сопоставительное расхождение	510
18.3. Псевдоправдоподобие	517
18.4. Сопоставление рейтингов и сопоставление отношений	519
18.5. Шумоподавляющее сопоставление рейтингов.....	521
18.6. Шумосопоставительное оценивание	521
18.7. Оценивание статистической суммы.....	524
18.7.1. Выборка по значимости с отжигом	525
18.7.2. Мостиковая выборка	528

Глава 19. Приближенный вывод.....

19.1. Вывод как оптимизация.....	530
19.2. ЕМ-алгоритм	532
19.3. МАР-вывод и разреженное кодирование	533
19.4. Вариационный вывод и обучение	535
19.4.1. Дискретные латентные переменные	536
19.4.2. Вариационное исчисление.....	541
19.4.3. Непрерывные латентные переменные.....	544
19.4.4. Взаимодействия между обучением и выводом	545
19.5. Обученный приближенный вывод	546
19.5.1. Бодрствование-сон.....	546
19.5.2. Другие формы обученного вывода	547

Глава 20. Глубокие порождающие модели.....

20.1. Машины Больцмана.....	548
20.2. Ограниченные машины Больцмана.....	550
20.2.1. Условные распределения	550
20.2.2. Обучение ограниченных машин Больцмана.....	552
20.3. Глубокие сети доверия	553
20.4. Глубокие машины Больцмана	555
20.4.1. Интересные свойства.....	557
20.4.2. Вывод среднего поля в ГМБ	558
20.4.3. Обучение параметров ГМБ.....	560
20.4.4. Послойное предобучение	560

20.4.5. Совместное обучение глубоких машин Больцмана.....	563
20.5. Машины Больцмана для вещественных данных	566
20.5.1. ОМБ Гаусса–Бернулли	567
20.5.2. Неориентированные модели условной ковариации.....	568
20.6. Сверточные машины Больцмана.....	572
20.7. Машины Больцмана для структурных и последовательных выходов.....	573
20.8. Другие машины Больцмана.....	574
20.9. Обратное распространение через случайные операции	575
20.9.1. Обратное распространение через дискретные стохастические операции.....	577
20.10. Ориентированные порождающие сети	579
20.10.1. Сигмоидные сети доверия.....	580
20.10.2. Дифференцируемые генераторные сети	581
20.10.3. Вариационные автокодировщики	583
20.10.4. Порождающие состязательные сети	586
20.10.5. Порождающие сети с сопоставлением моментов.....	589
20.10.6. Сверточные порождающие сети.....	590
20.10.7. Авторегрессивные сети.....	591
20.10.8. Линейные авторегрессивные сети.....	591
20.10.9. Нейронные авторегрессивные сети.....	592
20.10.10. NADE.....	593
20.11. Выборка из автокодировщиков	595
20.11.1. Марковская цепь, ассоциированная с произвольным шумоподавляющим автокодировщиком	596
20.11.2. Фиксация и условная выборка.....	596
20.11.3. Возвратная процедура обучения.....	597
20.12. Порождающие стохастические сети.....	598
20.12.1. Дискриминантные GSN	599
20.13. Другие схемы порождения	599
20.14. Оценивание порождающих моделей	600
20.15. Заключение	603
Список литературы	604
Предметный указатель	646

Линейная алгебра

Линейная алгебра – это раздел математики, который широко используется в науке и технике. Но поскольку линейная алгебра имеет дело с непрерывными, а не дискретными величинами, специалисты по информатике редко с ней сталкиваются. Однако для понимания многих алгоритмов машинного обучения, особенно глубокого, уверенное владение аппаратом линейной алгебры необходимо. Поэтому мы предположим введение в глубокое обучение концентрированное изложение основ линейной алгебры.

Если вы уже знакомы с линейной алгеброй, то можете пропустить эту главу. Если вам раньше доводилось встречаться с этим аппаратом, но нужен подробный справочник, чтобы освежить в памяти формулы, то рекомендуем книгу «The Matrix Cookbook» (Petersen and Pedersen, 2006). Если вы совсем ничего не знаете о линейной алгебре, то из этой главы вы почерпнете достаточно сведений для чтения книги, но мы настоятельно советуем обратиться к еще какому-нибудь источнику, посвященному исключительно линейной алгебре, например книге Шилова, Shilov (1977). В этой главе не рассматриваются многие вопросы линейной алгебры, не существенные для понимания глубокого обучения.

2.1. Скаляры, векторы, матрицы и тензоры

В линейной алгебре изучаются математические объекты нескольких видов.

- **Скаляры.** Скаляр – это просто число, в отличие от большинства других изучаемых в линейной алгебре объектов, представляющих собой массивы чисел. Мы будем обозначать скаляры курсивным шрифтом и, как правило, строчными буквами. Вводя в рассмотрение скаляр, мы будем указывать, что он означает, например: «Пусть $s \in \mathbb{R}$ обозначает угловой коэффициент прямой» (вещественное число) или «Обозначим $n \in \mathbb{N}$ число блоков» (целое натуральное число).
- **Векторы.** Вектор – это массив чисел. Элементы вектора расположены в определенном порядке. Отдельный элемент определяется своим индексом. Обычно векторы обозначаются строчными буквами курсивным полужирным шрифтом, например \mathbf{x} . Для обозначения элемента вектора указывается имя вектора курсивным шрифтом с подстрочным индексом. Первый элемент вектора \mathbf{x} обозначается x_1 , второй – x_2 и т. д. Необходимо также указать тип хранящихся в векторе чисел. Если каждый элемент вектора принадлежит \mathbb{R} , а всего элементов n , то вектор принадлежит декартову произведению n экземпляров \mathbb{R} , оно обозначается \mathbb{R}^n . Если требуется явно перечислить элементы вектора, то они записываются в столбик в квадратных скобках:

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}. \quad (2.1)$$

Можно считать, что векторам соответствуют точки в пространстве, а каждый элемент – координата вдоль одной из осей.

Иногда необходимо обозначить подмножество элементов вектора. В таком случае мы будем указывать в качестве подстрочного индекса множество индексов интересующих элементов. Например, чтобы адресовать элементы x_1 , x_3 и x_6 , мы определим множество $S = \{1, 3, 6\}$ и будем писать \mathbf{x}_S . Для обозначения дополнения ко множеству будем использовать знак $-$. Например, \mathbf{x}_{-1} обозначает вектор, содержащий все элементы \mathbf{x} , кроме x_1 , а \mathbf{x}_{-S} – вектор, содержащий все элементы \mathbf{x} , кроме x_1 , x_3 и x_6 .

- **Матрицы.** Матрица – это двумерный массив чисел, в котором каждый элемент идентифицируется двумя индексами, а не одним. Обычно матрицы обозначаются прописными буквами полужирным курсивным шрифтом, например \mathbf{A} . Если матрица вещественных чисел \mathbf{A} имеет высоту m и ширину n , то говорят, что $\mathbf{A} \in \mathbb{R}^{m \times n}$. Элементы матрицы обычно обозначаются светлым курсивным шрифтом, а индексы перечисляются через запятую. Например, $A_{1,1}$ – элемент \mathbf{A} в левом верхнем углу, а $A_{m,n}$ – в правом нижнем. Для адресации всех элементов в одной строке мы указываем номер этой строки i , а вместо номера столбца – двоеточие, например $A_{i,:}$ обозначает i -ю строку \mathbf{A} . Аналогично $A_{:,i}$ обозначает i -й столбец \mathbf{A} . Если требуется явно поименовать элементы матрицы, то мы записываем их в виде прямоугольной таблицы, заключенной в квадратные скобки:

$$\begin{bmatrix} A_{1,1} & A_{1,2} \\ A_{2,1} & A_{2,2} \end{bmatrix}. \quad (2.2)$$

Иногда требуется индексировать матричные выражения, не обозначенные одной буквой. Тогда нижние индексы ставятся после всего выражения. Например, $f(\mathbf{A})_{i,j}$ обозначает элемент в позиции (i, j) матрицы, полученной применением функции f к матрице \mathbf{A} .

- **Тензоры.** Бывает, что нужен массив размерности, большей 2. В общем случае массив чисел в узлах равномерной сетки с произвольным числом осей называется тензором. Тензор с именем « \mathbf{A} » будет обозначаться таким шрифтом: \mathbf{A} , а его элемент в позиции (i, j, k) – $A_{i,j,k}$.

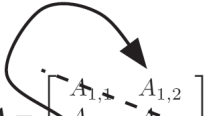
Важной операцией над матрицами является **транспонирование**. Транспонированной называется матрица, отраженная относительно главной диагонали, идущей из левого верхнего угла направо и вниз. На рис. 2.1 показано графическое изображение этой операции. Результат транспонирования матрицы \mathbf{A} обозначается \mathbf{A}^\top и формально определяется следующим образом:

$$(\mathbf{A}^\top)_{i,j} = A_{j,i}. \quad (2.3)$$

Векторы можно считать матрицами с одним столбцом. Тогда результатом транспонирования вектора будет матрица с одной строкой. Иногда мы будем записывать

вектор прямо в основном тексте в виде матрицы-строки, тогда транспонирование преобразует его в стандартный вектор-столбец, например: $\mathbf{x} = [x_1, x_2, x_3]^\top$.

Скаляр можно рассматривать как матрицу из одного элемента. Это значит, что результатом транспонирования скаляра является он сам: $a = a^\top$.



$$A = \begin{bmatrix} A_{1,1} & A_{1,2} \\ A_{2,1} & A_{2,2} \\ A_{3,1} & A_{3,2} \end{bmatrix} \Rightarrow A^\top = \begin{bmatrix} A_{1,1} & A_{2,1} & A_{3,1} \\ A_{1,2} & A_{2,2} & A_{3,2} \end{bmatrix}$$

Рис. 2.1 ❖ Транспонирование матрицы можно рассматривать как отражение относительно главной диагонали

Матрицы одинакового размера можно складывать поэлементно: $\mathbf{C} = \mathbf{A} + \mathbf{B}$ означает, что $C_{i,j} = A_{i,j} + B_{i,j}$.

Определены также операции сложения матрицы со скаляром и умножения матрицы на скаляр. Для этого нужно произвести соответствующую операцию для каждого элемента матрицы: $\mathbf{D} = a \cdot \mathbf{B} + c$ означает, что $D_{i,j} = a \cdot B_{i,j} + c$.

В контексте глубокого обучения применяется также не совсем традиционная нотация. Мы определяем операцию сложения матрицы и вектора, дающую матрицу: $\mathbf{C} = \mathbf{A} + \mathbf{b}$ означает, что $C_{i,j} = A_{i,j} + b_j$. Иными словами, вектор \mathbf{b} прибавляется к каждой строке матрицы. Такая сокращенная запись позволяет обойтись без определения матрицы, в которой каждая строка содержит вектор \mathbf{b} . Это неявное копирование \mathbf{b} сразу в несколько мест называется **укладыванием** (broadcasting).

2.2. Умножение матриц и векторов

Одна из самых важных операций над матрицами – перемножение двух матриц. Произведением матриц \mathbf{A} и \mathbf{B} также является матрица, \mathbf{C} . Она определена, только если число столбцов \mathbf{A} равно числу строк \mathbf{B} . Если \mathbf{A} имеет размер $m \times n$, а \mathbf{B} – размер $n \times p$, то \mathbf{C} будет иметь размер $m \times p$. Для обозначения произведения имена матриц записываются подряд, например:

$$\mathbf{C} = \mathbf{AB}. \quad (2.4)$$

Произведение определяется следующим образом:

$$C_{i,j} = \sum_k A_{i,k} B_{k,j}. \quad (2.5)$$

Отметим, что обычное произведение двух матриц получается не в результате **поэлементного перемножения**. Но такая операция тоже существует – она называется **произведением Адамара** и обозначается $\mathbf{A} \odot \mathbf{B}$.

Скалярным произведением двух векторов \mathbf{x} и \mathbf{y} одинаковой размерности называется произведение матриц $\mathbf{x}^\top \mathbf{y}$. Таким образом, элемент $C_{i,j}$ матрицы $\mathbf{C} = \mathbf{AB}$ является скалярным произведением i -й строки \mathbf{A} и j -го столбца \mathbf{B} .

Операции над матрицами обладают рядом полезных свойств, упрощающих их математический анализ. Например, операция умножения дистрибутивна относительно сложения:

$$\mathbf{A}(\mathbf{B} + \mathbf{C}) = \mathbf{AB} + \mathbf{AC}. \quad (2.6)$$

Она также ассоциативна:

$$\mathbf{A}(\mathbf{BC}) = (\mathbf{AB})\mathbf{C}. \quad (2.7)$$

В отличие от умножения скаляров, умножение матриц не коммутативно (равенство $\mathbf{AB} = \mathbf{BA}$ справедливо не всегда). Но скалярное произведение векторов коммутативно:

$$\mathbf{x}^\top \mathbf{y} = \mathbf{y}^\top \mathbf{x}. \quad (2.8)$$

Операции транспонирования и умножения связаны простой формулой:

$$(\mathbf{AB})^\top = \mathbf{B}^\top \mathbf{A}^\top. \quad (2.9)$$

Отсюда следует доказательство тождества (2.8), поскольку скалярное произведение – это скаляр и, значит, совпадает с результатом транспонирования:

$$\mathbf{x}^\top \mathbf{y} = (\mathbf{x}^\top \mathbf{y})^\top = \mathbf{y}^\top \mathbf{x}. \quad (2.10)$$

Поскольку эта книга – не учебник по линейной алгебре, мы не станем приводить полный список полезных свойств матричного произведения, а просто сообщим, что их еще очень много.

Теперь мы знаем о принятых в линейной алгебре обозначениях достаточно, для того чтобы записать систему линейных уравнений:

$$\mathbf{Ax} = \mathbf{b}, \quad (2.11)$$

где $\mathbf{A} \in \mathbb{R}^{m \times n}$ – известная матрица, $\mathbf{b} \in \mathbb{R}^m$ – известный вектор, а $\mathbf{x} \in \mathbb{R}^n$ – неизвестный вектор, элементы которого, x_i , мы хотим найти. Каждая строка \mathbf{A} и соответственный элемент \mathbf{b} дают одно ограничение. Уравнение (2.11) можно переписать в виде:

$$\mathbf{A}_{1,\cdot} \mathbf{x} = b_1 \quad (2.12)$$

$$\mathbf{A}_{2,\cdot} \mathbf{x} = b_2 \quad (2.13)$$

$$\dots \quad (2.14)$$

$$\mathbf{A}_{m,\cdot} \mathbf{x} = b_m \quad (2.15)$$

или даже в еще более явном виде:

$$\mathbf{A}_{1,1}x_1 + \mathbf{A}_{1,2}x_2 + \dots + \mathbf{A}_{1,n}x_n = b_1 \quad (2.16)$$

$$\mathbf{A}_{2,1}x_1 + \mathbf{A}_{2,2}x_2 + \dots + \mathbf{A}_{2,n}x_n = b_2 \quad (2.17)$$

$$\dots \quad (2.18)$$

$$\mathbf{A}_{m,1}x_1 + \mathbf{A}_{m,2}x_2 + \dots + \mathbf{A}_{m,n}x_n = b_m \quad (2.19)$$

Нотация умножения матрицы на вектор позволяет записывать такие уравнения более компактно.

2.3. Единичная и обратная матрица

В линейной алгебре определена операция обращения матриц, которая позволяет аналитически решить уравнение (2.11) для многих значений \mathbf{A} .

Чтобы описать операцию обращения, мы должны сначала определить **единичную матрицу**. Так называется матрица, которая при умножении на любой вектор оставля-

ет этот вектор без изменения. Единичную матрицу, сохраняющую n -мерные векторы, будем обозначать I_n . Формально говоря, $I_n \in \mathbb{R}^{n \times n}$ и

$$\forall \mathbf{x} \in \mathbb{R}^n, I_n \mathbf{x} = \mathbf{x}. \quad (2.20)$$

Единичная матрица устроена просто: все элементы главной диагонали равны единице, а все остальные – нулю. Пример приведен на рис. 2.2.

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Рис. 2.2 ❖ Пример единичной матрицы I_3

Матрица, обратная к A , обозначается A^{-1} и по определению удовлетворяет следующему соотношению:

$$A^{-1}A = I_n. \quad (2.21)$$

Теперь для решения уравнения (2.11) нужно проделать следующие действия:

$$A\mathbf{x} = \mathbf{b}; \quad (2.22)$$

$$A^{-1}A\mathbf{x} = A^{-1}\mathbf{b}; \quad (2.23)$$

$$I_n\mathbf{x} = A^{-1}\mathbf{b}; \quad (2.24)$$

$$\mathbf{x} = A^{-1}\mathbf{b}. \quad (2.25)$$

Разумеется, эта процедура предполагает, что матрицу A^{-1} можно найти. В следующем разделе мы обсудим условия существования обратной матрицы.

В случае, когда A^{-1} существует, для ее нахождения в замкнутой форме можно применить один из нескольких алгоритмов. Теоретически один раз найденную обратную матрицу можно использовать многократно для решения уравнения с разными значениями \mathbf{b} . Однако на практике A^{-1} редко используется в программах. Поскольку матрицу A^{-1} можно представить в компьютере лишь с ограниченной точностью, алгоритмы, в которых используется значение \mathbf{b} , обычно дают более точные оценки \mathbf{x} .

2.4. Линейная зависимость и линейная оболочка

Для существования A^{-1} уравнение (2.11) должно иметь единственное решение для любого значения \mathbf{b} . Бывает и так, что система уравнений не имеет ни одного решения или имеет бесконечно много решений для некоторых значений \mathbf{b} . Но никогда не может быть так, что система имеет конечное число решений, большее 1; если \mathbf{x} и \mathbf{y} – решения, то

$$\mathbf{z} = \alpha\mathbf{x} + (1 - \alpha)\mathbf{y} \quad (2.26)$$

тоже решение при любом вещественном α .

Чтобы проанализировать, сколько решений имеет уравнение, представим себе, что столбцы A определяют различные направления от **начала координат** (точки, соответствующей вектору, все элементы которого равны нулю), а затем подумаем, сколько

есть способов достичь точки \mathbf{b} . Тогда элемент x_i определяет, как далеко следует пройти в направлении столбца i :

$$\mathbf{Ax} = \sum_i x_i \mathbf{A}_{:,i}. \quad (2.27)$$

В общем случае такое выражение называется **линейной комбинацией**. Формально для получения линейной комбинации некоторого множества векторов $\{\mathbf{v}^{(1)}, \dots, \mathbf{v}^{(n)}\}$ нужно умножить каждый вектор $\mathbf{v}^{(i)}$ на скалярный коэффициент и сложить результаты:

$$\sum_i c_i \mathbf{v}^{(i)}. \quad (2.28)$$

Линейной оболочкой множества векторов называется множество всех векторов, представимых в виде их линейных комбинаций.

Таким образом, чтобы определить, имеет ли уравнение $\mathbf{Ax} = \mathbf{b}$ решение, нужно проверить, входит ли \mathbf{b} в линейную оболочку столбцов матрицы \mathbf{A} . Эта линейная оболочка называется **пространством столбцов**, или **областью значений** матрицы \mathbf{A} .

Итак, чтобы система $\mathbf{Ax} = \mathbf{b}$ имела решение для любого $\mathbf{b} \in \mathbb{R}^m$, необходимо, чтобы пространство столбцов \mathbf{A} совпадало с \mathbb{R}^m . Если в \mathbb{R}^m существует точка \mathbf{b} , не принадлежащая пространству столбцов, то для такого значения \mathbf{b} система не будет иметь решения. Из требования о совпадении пространства столбцов \mathbf{A} с \mathbb{R}^m сразу следует, что в \mathbf{A} должно быть по меньшей мере m столбцов, т. е. $n \geq m$. Иначе размерность пространства столбцов была бы меньше m . Рассмотрим, к примеру, матрицу 3×2 . Правая часть \mathbf{b} имеет размерность 3, а размерность \mathbf{x} равна всего 2, поэтому путем изменения \mathbf{x} мы в лучшем случае сможем заместить двумерную плоскость в \mathbb{R}^3 . Уравнение будет иметь решение тогда и только тогда, когда \mathbf{b} лежит в этой плоскости.

Условие $n \geq m$ необходимо, чтобы система имела решение для любой правой части, но недостаточно, поскольку некоторые столбцы могут быть избыточны. Рассмотрим матрицу 2×2 , в которой оба столбца одинаковы. Ее пространство столбцов такое же, как у матрицы 2×1 , содержащей только один экземпляр повторяющегося столбца. Иными словами, пространство столбцов – прямая, не покрывающая все \mathbb{R}^2 , несмотря на то что столбцов два.

Такой вид избыточности называется **линейной зависимостью**. Множество векторов называется **линейно независимым**, если ни один вектор из этого множества не является линейной комбинацией других. Если добавить во множество вектор, являющийся линейной комбинацией каких-то других, то линейная оболочка исходного множества не пополнится новыми точками. Следовательно, для того чтобы пространство столбцов матрицы совпадало со всем пространством \mathbb{R}^m , матрица должна содержать, по меньшей мере, один набор m линейно независимых столбцов. Это необходимое и достаточное условие существования у уравнения (2.11) решения для любого значения \mathbf{b} . Отметим, что в условии говорится о наличии в точности m линейно независимых столбцов в наборе, а не хотя бы m . Ни один набор m -мерных векторов не может содержать более m линейно независимых векторов, но матрица, имеющая более m столбцов, может содержать несколько таких наборов.

Для существования обратной матрицы необходимо еще, чтобы уравнение (2.11) имело не более одного решения для каждого значения \mathbf{b} . Это означает, что в матрице может быть не больше m столбцов, иначе существовало бы более одного способа параметризовать каждое решение.

Собирая все вместе, мы заключаем, что матрица должна быть квадратной, т. е. $m = n$, и что ее столбцы должны быть линейно независимы. Квадратная матрица с линейно зависимыми столбцами называется **вырожденной** (особенной, сингулярной).

Если A не квадратная или квадратная, но вырожденная, то решить уравнение все-таки можно, но метод обращения матрицы не подойдет.

До сих пор при обсуждении обратных матриц мы имели в виду умножение слева. Но можно также определить матрицу, обратную относительно умножения справа:

$$AA^{-1} = I. \quad (2.29)$$

Для квадратных матриц обе обратные матрицы совпадают.

2.5. Нормы

Иногда требуется получить длину вектора. В машинном обучении для измерения длины применяются функции, называемые **нормами**. Норма L^p определяется следующим образом:

$$\|\mathbf{x}\|_p = \left(\sum_i |x_i|^p \right)^{\frac{1}{p}}. \quad (2.30)$$

для $p \in \mathbb{R}, p \geq 1$.

Нормы, в т. ч. и норма L^p , – это функции, отображающие векторы на неотрицательные числа. Интуитивно норма вектора \mathbf{x} измеряет расстояние от точки \mathbf{x} до начала координат. Более строго, нормой называется любая функция f , обладающая следующими свойствами:

- $f(\mathbf{x}) = 0 \Rightarrow \mathbf{x} = \mathbf{0}$;
- $f(\mathbf{x} + \mathbf{y}) \leq f(\mathbf{x}) + f(\mathbf{y})$ (неравенство треугольника);
- $\forall \alpha \in \mathbb{R} f(\alpha \mathbf{x}) = |\alpha| f(\mathbf{x})$.

Норма L^2 называется **евклидовой нормой**, это просто евклидово расстояние от начала координат до точки, определяемой вектором \mathbf{x} . Норма L^2 применяется в машинном обучении так часто, что ее обозначают просто $\|\mathbf{x}\|$, опуская надстрочный индекс 2. Кроме того, длину вектора принято измерять как квадрат нормы L^2 , который можно записать в виде $\mathbf{x}^\top \mathbf{x}$.

Квадрат нормы L^2 удобнее самой нормы с точки зрения математических выкладок и вычислений. Например, частная производная квадрата нормы L^2 по каждому элементу \mathbf{x} зависит только от этого элемента, тогда как производные самой нормы L^2 зависят от всего вектора в целом. Во многих контекстах квадрат нормы L^2 нежелателен, потому что он слишком медленно растет вблизи начала координат. В некоторых приложениях машинного обучения важно различать элементы, в точности равные нулю и мало отличающиеся от нуля. В таких случаях мы прибегаем к функции, которая всюду растет с одинаковой скоростью, но сохраняет математическую простоту: норме L^1 . Норму L^1 можно записать в виде:

$$\|\mathbf{x}\|_1 = \sum_i |x_i|. \quad (2.31)$$

Эта норма часто используется в машинном обучении, когда важно различать нулевые и ненулевые элементы. Всякий раз как элемент вектора \mathbf{x} отдалается от 0 на ε , норма L^1 увеличивается на ε .

Иногда длина вектора измеряется количеством его ненулевых элементов. Некоторые авторы называют такую функцию «нормой L^0 », но такая терминология некорректна. Количество ненулевых элементов вектора нормой не является, поскольку при умножении вектора на α это количество не меняется. Зачастую вместо подсчета ненулевых элементов используют норму L^1 .

В машинном обучении также часто возникает норма L^∞ , которую еще называют **максимальной нормой**. Она определяется как максимальное абсолютное значение элементов вектора:

$$\|\mathbf{x}\|_\infty = \max_i |x_i|. \quad (2.32)$$

Иногда возникает необходимость оценить размер матрицы. В контексте глубокого обучения для этого обычно применяют **норму Фробениуса**:

$$\|A\|_F = \sqrt{\sum_{i,j} A_{i,j}^2}, \quad (2.33)$$

аналогичную норме L^2 вектора.

Скалярное произведение двух векторов можно выразить через нормы:

$$\mathbf{x}^\top \mathbf{y} = \|\mathbf{x}\|_2 \|\mathbf{y}\|_2 \cos \theta, \quad (2.34)$$

где θ – угол между \mathbf{x} и \mathbf{y} .

2.6. Специальные виды матриц и векторов

Некоторые частные случаи матриц и векторов особенно полезны.

В **диагональной матрице** все элементы, кроме находящихся на главной диагонали, равны нулю. Точнее, матрица \mathbf{D} называется диагональной, если $D_{i,j} = 0$ для всех $i \neq j$. Примером диагональной матрицы служит единичная матрица, в которой все элементы на главной диагонали равны 1. Квадратная диагональная матрица, диагональ которой описывается вектором \mathbf{v} , обозначается $\text{diag}(\mathbf{v})$. Диагональные матрицы представляют особый интерес, потому что произведение с такой матрицей вычисляется очень эффективно. Чтобы вычислить $\text{diag}(\mathbf{v})\mathbf{x}$, нужно умножить каждый элемент x_i на v_i . Иначе говоря, $\text{diag}(\mathbf{v})\mathbf{x} = \mathbf{v} \odot \mathbf{x}$. Обратить квадратную диагональную матрицу тоже просто. Обратная к ней матрица существует, только если все диагональные элементы отличны от нуля, и тогда $\text{diag}(\mathbf{v})^{-1} = \text{diag}([1/v_1, \dots, 1/v_n]^\top)$. Во многих случаях алгоритм машинного обучения можно сформулировать в терминах произвольных матриц, но получить менее накладный (но и менее общий) алгоритм, ограничившись только диагональными матрицами.

Диагональная матрица может быть не только квадратной, но и прямоугольной. У неквадратной диагональной матрицы нет обратной, но умножение на нее все равно обходится дешево. Произведение диагональной матрицы \mathbf{D} на вектор \mathbf{x} сводится к умножению каждого элемента \mathbf{x} на некоторое число и дописыванию нулей в конец получившегося вектора, если высота \mathbf{D} больше ее ширины, или отбрасыванию последних элементов вектора в противном случае.

Матрица называется **симметричной**, если совпадает с транспонированной:

$$\mathbf{A} = \mathbf{A}^\top. \quad (2.35)$$

Симметричные матрицы часто образуются, когда их элементы порождаются некоторой функцией двух аргументов, результат которой не зависит от порядка аргументов. Например, если $A_{i,j}$ – расстояние от точки i до точки j , то $A_{i,j} = A_{j,i}$, поскольку функция расстояния симметрична.

Единичным вектором называется вектор с нормой, равной 1:

$$\|\mathbf{x}\|_2 = 1. \quad (2.36)$$

Говорят, что векторы \mathbf{x} и \mathbf{y} ортогональны, если $\mathbf{x}^\top \mathbf{y} = 0$. Если нормы обоих векторов не равны 0, то это значит, что угол между векторами составляет 90 градусов. В пространстве \mathbb{R}^n можно найти не более n взаимно ортогональных ненулевых векторов. Ортогональные векторы, норма которых равна 1, называются **ортонормированными**.

Ортогональной называется квадратная матрица, строки и столбцы которой образуют ортонормированные системы векторов:

$$\mathbf{A}^\top \mathbf{A} = \mathbf{A} \mathbf{A}^\top = \mathbf{I}. \quad (2.37)$$

Отсюда следует, что

$$\mathbf{A}^{-1} = \mathbf{A}^\top. \quad (2.38)$$

Ортогональные матрицы интересны из-за простоты вычисления обратной матрицы. Обратите внимание на определение ортогональной матрицы: ее строки должны образовывать не просто ортогональную, а ортонормированную систему векторов. Не существует специального термина для матриц, строки и столбцы которых образуют ортогональную, но не ортонормированную систему векторов.

2.7. Спектральное разложение матрицы

Многие математические объекты удастся лучше понять, если разложить их на составные части или найти какие-то универсальные свойства, не зависящие от способа представления.

Например, целые числа можно разложить на простые множители. Способ представления числа 12 зависит от того, в какой системе счисления оно записано, например двоичной или десятичной, но в любом случае $12 = 2 \times 2 \times 3$. Из этого представления можно вывести ряд полезных свойств, например что 12 не делится на 5 или что любое число, делящееся на 12, делится также на 3.

Таким образом, истинная природа целого числа раскрывается в результате разложения его на простые множители. И точно так же мы можем разложить матрицу и тем самым получить о ее функциональных свойствах некоторую информацию, не очевидную из представления матрицы в виде массива элементов.

Одно из самых популярных разложений матрицы называется **спектральным разложением**, или разложением на множество собственных векторов и собственных значений.

Собственным вектором квадратной матрицы \mathbf{A} называется ненулевой вектор \mathbf{v} – такой, что умножение \mathbf{A} на \mathbf{v} изменяет лишь масштаб \mathbf{v} :

$$\mathbf{A} \mathbf{v} = \lambda \mathbf{v}. \quad (2.39)$$

Скаляр λ называется **собственным значением**, соответствующим этому собственному вектору. (Можно также искать левые собственные векторы, для которых $\mathbf{v}^T \mathbf{A} = \lambda \mathbf{v}^T$, но обычно нас интересуют только правые собственные векторы.)

Если \mathbf{v} – собственный вектор \mathbf{A} , то собственным будет и вектор $s\mathbf{v}$ для любого $s \in \mathbb{R}$, $s \neq 0$. Более того, вектору $s\mathbf{v}$ соответствует то же собственное значение, что и \mathbf{v} . Поэтому мы обычно ищем только единичные собственные векторы.

Пусть матрица \mathbf{A} имеет n линейно независимых собственных векторов $\{\mathbf{v}^{(1)}, \dots, \mathbf{v}^{(n)}\}$ с собственными значениями $\{\lambda_1, \dots, \lambda_n\}$. Образует из них матрицу \mathbf{V} , в которой каждый столбец – это собственный вектор: $\mathbf{V} = [\mathbf{v}^{(1)}, \dots, \mathbf{v}^{(n)}]$. А из собственных значений образуем вектор $\boldsymbol{\lambda} = [\lambda_1, \dots, \lambda_n]^T$. Тогда **спектральное разложение** матрицы \mathbf{A} описывается формулой:

$$\mathbf{A} = \mathbf{V} \text{diag}(\boldsymbol{\lambda}) \mathbf{V}^{-1}. \quad (2.40)$$

Мы видели, что конструирование матриц с заданными собственными значениями и собственными векторами позволяет растягивать пространство в нужных направлениях. Но часто бывает нужно разложить имеющуюся матрицу по ее собственным векторам и собственным значениям. Это помогает анализировать некоторые свойства матрицы точно так же, как разложение целого числа на простые множители помогает понять поведение этого числа.

Не у каждой матрицы есть спектральное разложение. Иногда спектральное разложение существует, но состоит из комплексных, а не вещественных чисел. К счастью, в этой книге нам обычно придется иметь дело только с матрицами специального вида, у которых имеется простое разложение. Точнее, у любой симметричной вещественной матрицы все собственные векторы и собственные значения вещественные.

$$\mathbf{A} = \mathbf{Q} \boldsymbol{\Lambda} \mathbf{Q}^T, \quad (2.41)$$

где \mathbf{Q} – ортогональная матрица, образованная собственными векторами \mathbf{A} , а $\boldsymbol{\Lambda}$ – диагональная матрица. Собственное значение $\boldsymbol{\Lambda}_{i,i}$ ассоциировано с собственным вектором в i -м столбце \mathbf{Q} , обозначаемым $\mathbf{Q}_{:,i}$. Поскольку \mathbf{Q} – ортогональная матрица, можно считать, что \mathbf{A} масштабирует пространство с коэффициентом λ_i в направлении $\mathbf{v}^{(i)}$. На рис. 2.3 показан пример.

Хотя для любой симметричной вещественной матрицы \mathbf{A} существует спектральное разложение, это разложение может быть не единственным. Если какие-то два или более собственных векторов имеют одинаковое собственное значение, то любые ортогональные векторы, принадлежащие их линейной оболочке, также будут собственными векторами \mathbf{A} с тем же самым собственным значением, поэтому матрицу \mathbf{Q} можно с тем же успехом образовать из этих векторов. По принятому соглашению элементы $\boldsymbol{\Lambda}$ обычно располагаются в порядке убывания. При таком соглашении спектральное разложение единственно, только если все собственные значения различны.

Спектральное разложение может многое рассказать о матрице. Матрица является вырожденной тогда и только тогда, когда у нее есть хотя бы одно нулевое собственное значение. Кроме того, спектральное разложение вещественной симметричной матрицы можно использовать для оптимизации квадратичных форм вида $f(\mathbf{x}) = \mathbf{x}^T \mathbf{A} \mathbf{x}$ при условии $\|\mathbf{x}\|_2 = 1$. Если \mathbf{x} – собственный вектор \mathbf{A} , то f равно его собственному значению. Максимальное (минимальное) значение f при заданном ограничении равно максимальному (минимальному) собственному значению.

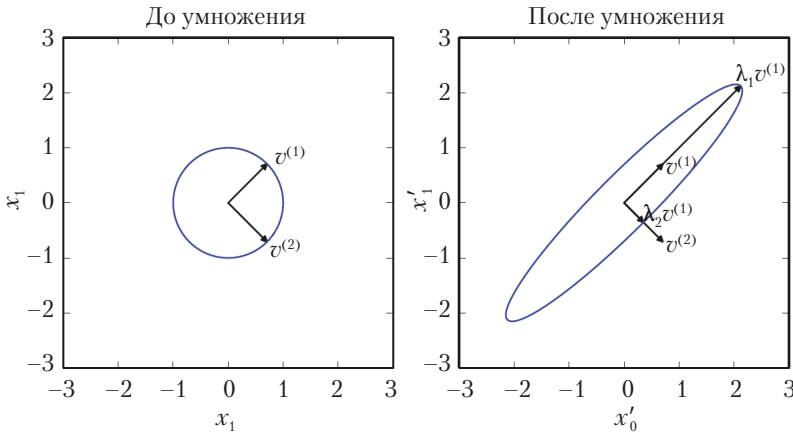


Рис. 2.3 ❖ Геометрический смысл собственных векторов и собственных значений. У матрицы \mathbf{A} два ортонормированных собственных вектора: $\mathbf{v}^{(1)}$ с собственным значением λ_1 и $\mathbf{v}^{(2)}$ с собственным значением λ_2 . (Слева) Множество всех единичных векторов $\mathbf{u} \in \mathbb{R}^2$ изображено в виде единичной окружности. (Справа) Изображено множество точек $\mathbf{A}\mathbf{u}$. Наблюдая, во что \mathbf{A} переводит единичную окружность, мы можем сделать вывод, что она масштабирует пространство в направлении $\mathbf{v}^{(i)}$ с коэффициентом λ_i

Матрица, все собственные значения которой положительны, называется **положительно определенной**. Если же все собственные значения положительны или равны нулю, то матрица называется **положительно полуопределенной**. Аналогично, если все собственные значения отрицательны, то матрица называется **отрицательно определенной**, а если отрицательны или равны нулю – то **отрицательно полуопределенной**. Положительно полуопределенные матрицы интересны тем, что для них выполняется неравенство $\mathbf{x}^\top \mathbf{A} \mathbf{x} \geq 0$ при любом \mathbf{x} . Положительная определенность дополнительно гарантирует, что $\mathbf{x}^\top \mathbf{A} \mathbf{x} = 0 \Rightarrow \mathbf{x} = 0$.

2.8. Сингулярное разложение

В разделе 2.7 мы видели, как разложить матрицу по собственным векторам и собственным значениям. **Сингулярное разложение** (singular value decomposition – SVD) – это другой способ разложения матрицы: по **сингулярным векторам** и **сингулярным значениям**. SVD несет ту же информацию, что спектральное разложение, но применимо в более общем случае. Сингулярное разложение есть у любой вещественной матрицы, чего не скажешь о спектральном разложении. Например, если матрица не квадратная, то спектральное разложение не определено, поэтому мы вынуждены использовать сингулярное разложение.

Напомним, что для получения спектрального разложения матрицы \mathbf{A} нужно найти матрицу \mathbf{V} собственных векторов и вектор $\boldsymbol{\lambda}$ собственных значений – такие, что:

$$\mathbf{A} = \mathbf{V} \text{diag}(\boldsymbol{\lambda}) \mathbf{V}^{-1}. \quad (2.42)$$

Сингулярное разложение аналогично, но теперь \mathbf{A} записывается в виде произведения трех матриц:

$$\mathbf{A} = \mathbf{UDV}^T. \quad (2.43)$$

Пусть \mathbf{A} – матрица $m \times n$. Тогда \mathbf{U} должна быть матрицей $m \times m$, \mathbf{D} – матрицей $m \times n$, а \mathbf{V} – матрицей $n \times n$.

Эти матрицы обладают определенными свойствами. Матрицы \mathbf{U} и \mathbf{V} ортогональные, а матрица \mathbf{D} диагональная (при этом необязательно квадратная).

Элементы на диагонали \mathbf{D} называются **сингулярными значениями** матрицы \mathbf{A} . Столбцы \mathbf{U} называются **левыми сингулярными векторами**, а столбцы \mathbf{V} – **правыми сингулярными векторами**.

Сингулярное разложение \mathbf{A} можно интерпретировать в терминах спектрального разложения функций \mathbf{A} . Левые сингулярные векторы \mathbf{A} являются собственными векторами \mathbf{AA}^T , а правые сингулярные векторы \mathbf{A} – собственными векторами $\mathbf{A}^T\mathbf{A}$. Ненулевые сингулярные значения \mathbf{A} равны квадратным корням из собственных значений $\mathbf{A}^T\mathbf{A}$ (и \mathbf{AA}^T).

Но, пожалуй, самое полезное свойство сингулярного разложения – использование его для обобщения операции обращения матриц на неквадратные матрицы, о чем мы и поговорим в следующем разделе.

2.9. Псевдообратная матрица Мура–Пенроуза

Операция обращения определена только для квадратных матриц. Допустим, что требуется найти левую обратную матрицу \mathbf{B} для матрицы \mathbf{A} , что позволило бы решить линейное уравнение

$$\mathbf{Ax} = \mathbf{y} \quad (2.44)$$

путем умножения обеих частей слева на \mathbf{B} :

$$\mathbf{x} = \mathbf{By} \quad (2.45)$$

Единственное отображение \mathbf{A} на \mathbf{B} возможно не всегда; это зависит от характера задачи.

Если высота \mathbf{A} больше ширины, то у такого уравнения может не оказаться решений. Если же ширина \mathbf{A} больше высоты, то решений может быть много.

Операция **псевдообращения Мура–Пенроуза** позволяет кое-что сделать и в этих случаях. Псевдообратная к \mathbf{A} матрица определяется следующим образом:

$$\mathbf{A}^+ = \lim_{\alpha \searrow 0} (\mathbf{A}^T\mathbf{A} + \alpha\mathbf{I})^{-1}\mathbf{A}^T. \quad (2.46)$$

Применяемые на практике алгоритмы вычисления псевдообратной матрицы основаны не на этом определении, а на формуле

$$\mathbf{A}^+ = \mathbf{VD}^+\mathbf{U}^T, \quad (2.47)$$

где \mathbf{U} , \mathbf{D} и \mathbf{V} составляют сингулярное разложение \mathbf{A} , а псевдообратная матрица \mathbf{D}^+ диагональной матрицы \mathbf{D} получается путем обращения всех ненулевых диагональных элементов с последующим транспонированием.

Если число столбцов \mathbf{A} больше числа строк, то решение линейного уравнения, найденное псевдообращением, – лишь одно из многих возможных. Точнее, будет найдено решение $\mathbf{x} = \mathbf{A}^+\mathbf{y}$ с минимальной евклидовой нормой $\|\mathbf{x}\|_2$.

Если число строк \mathbf{A} больше числа столбцов, то может не оказаться ни одного решения. В таком случае псевдообращение находит такой вектор \mathbf{x} , для которого \mathbf{Ax} максимально близко к \mathbf{y} в терминах евклидовой нормы $\|\mathbf{Ax} - \mathbf{y}\|_2$.

2.10. Оператор следа

Оператор следа вычисляет сумму всех диагональных элементов матрицы:

$$\text{Tr}(\mathbf{A}) = \sum_i \mathbf{A}_{i,i}. \quad (2.48)$$

Этот оператор полезен по многим причинам. Некоторые операции, которые трудно выразить, не прибегая к нотации суммирования, можно записать с помощью произведения матриц и оператора следа. Вот, например, как можно переписать определение нормы Фробениуса матрицы:

$$\|\mathbf{A}\|_F = \sqrt{\text{Tr}(\mathbf{AA}^\top)}. \quad (2.49)$$

Если в выражение входит оператор следа, то открываются возможности преобразовывать это выражение, пользуясь различными тождествами. Например, след инвариантен относительно транспонирования:

$$\text{Tr}(\mathbf{A}) = \text{Tr}(\mathbf{A}^\top). \quad (2.50)$$

След квадратной матрицы, представленной в виде произведения сомножителей, инвариантен также относительно перемещения последнего сомножителя в начало, если формы матриц допускают такую операцию:

$$\text{Tr}(\mathbf{ABC}) = \text{Tr}(\mathbf{CAB}) = \text{Tr}(\mathbf{BCA}), \quad (2.51)$$

или в более общем виде:

$$\text{Tr}\left(\prod_{i=1}^n \mathbf{F}^{(i)}\right) = \text{Tr}\left(\mathbf{F}^{(n)} \prod_{i=1}^{n-1} \mathbf{F}^{(i)}\right). \quad (2.52)$$

Эта инвариантность относительно циклической перестановки имеет место даже тогда, когда меняется форма произведения. Например, если $\mathbf{A} \in \mathbb{R}^{m \times n}$, $\mathbf{B} \in \mathbb{R}^{n \times m}$, то

$$\text{Tr}(\mathbf{AB}) = \text{Tr}(\mathbf{BA}), \quad (2.53)$$

несмотря на то что $\mathbf{AB} \in \mathbb{R}^{m \times m}$, а $\mathbf{BA} \in \mathbb{R}^{n \times n}$.

Полезно также помнить, что след скаляра равен ему самому: $a = \text{Tr}(a)$.

2.11. Определитель

Определитель квадратной матрицы, обозначаемый $\det(\mathbf{A})$, — это функция, сопоставляющая матрице вещественное число. Определитель равен произведению всех собственных значений матрицы. Абсолютную величину определителя можно рассматривать как меру сжатия или расширения пространства матрицей. Если определитель равен 0, то пространство полностью сворачивается хотя бы по одному измерению, т. е. теряется весь объем. Если определитель равен 1, то преобразование сохраняет объем.

2.12. Пример: метод главных компонент

Один из простых алгоритмов машинного обучения, метод главных компонент (principal components analysis – PCA), можно вывести из основ линейной алгебры.

Пусть имеется набор m точек $\{\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(m)}\}$ в пространстве \mathbb{R}^n , и мы хотим подвергнуть их сжатию с потерей информации, т. е. сохранить точки в меньшем объеме памяти, возможно, ценой некоторой потери точности. Но хотелось бы свести эти потери к минимуму.

Один из способов кодирования точек – представить их в пространстве меньшей размерности. Для каждой точки $\mathbf{x}^{(i)} \in \mathbb{R}^n$ мы ищем соответствующий ей кодовый вектор $\mathbf{c}^{(i)} \in \mathbb{R}^l$. Если l меньше n , то для хранения кодированных точек потребуется меньше памяти, чем для исходных. Мы хотим найти функцию кодирования $f(\mathbf{x}) = \mathbf{c}$ и функцию декодирования, реконструирующую исходную точку по кодированной, $\mathbf{x} \approx g(f(\mathbf{x}))$.

Алгоритм PCA определяется выбором функции декодирования. Чтобы упростить декодер, мы будем использовать умножение матриц для обратного перехода в \mathbb{R}^n . Пусть $g(\mathbf{c}) = \mathbf{D}\mathbf{c}$, где $\mathbf{D} \in \mathbb{R}^{n \times l}$ – матрица, определяющая декодирование.

Вычисление оптимального декодера может оказаться трудной задачей. Для ее упрощения в методе PCA на матрицу \mathbf{D} налагается ограничение: ее столбцы должны быть ортогональны (но это не означает, что \mathbf{D} – ортогональная матрица в определенном выше смысле, разве что $l = n$).

У поставленной задачи бесконечно много решений, поскольку мы можем увеличить масштаб $D_{:,p}$ одновременно пропорционально уменьшив c_i для всех точек. Для получения единственного решения потребуем еще, чтобы норма всех столбцов \mathbf{D} была равна 1.

Чтобы превратить смутную идею в алгоритм, нужно первым делом решить, как генерировать оптимальную кодовую точку \mathbf{c}^* для каждой исходной точки \mathbf{x} . Один из способов – минимизировать расстояние между \mathbf{x} и ее реконструкцией $g(\mathbf{c}^*)$. Для измерения этого расстояния применим какую-нибудь норму. В алгоритме главных компонент берется норма L^2 :

$$\mathbf{c}^* = \arg \min_{\mathbf{c}} \|\mathbf{x} - g(\mathbf{c})\|_2. \quad (2.54)$$

Мы можем использовать квадрат нормы L^2 , а не ее саму, потому что минимум достигается при одном и том же значении \mathbf{c} , т. к. норма L^2 неотрицательна, а операция возведения в квадрат – монотонно возрастающая для неотрицательных аргументов.

$$\mathbf{c}^* = \arg \min_{\mathbf{c}} \|\mathbf{x} - g(\mathbf{c})\|_2^2. \quad (2.55)$$

Минимизируемую функцию можно переписать в виде:

$$(\mathbf{x} - g(\mathbf{c}))^\top (\mathbf{x} - g(\mathbf{c})) \quad (2.56)$$

(по определению нормы L^2 из формулы (2.30))

$$= \mathbf{x}^\top \mathbf{x} - \mathbf{x}^\top g(\mathbf{c}) - g(\mathbf{c})^\top \mathbf{x} + g(\mathbf{c})^\top g(\mathbf{c}) \quad (2.57)$$

(в силу дистрибутивности)

$$= \mathbf{x}^\top \mathbf{x} - 2\mathbf{x}^\top g(\mathbf{c}) + g(\mathbf{c})^\top g(\mathbf{c}) \quad (2.58)$$

(поскольку результат транспонирования скаляра $g(\mathbf{c})^\top \mathbf{x}$ совпадает с ним самим).