Авторский коллектив

Информатика

Учебник для 10–11 классов

Продвинутый уровень

От уравнений до больших языковых моделей

Москва • 2026

УДК 004.8
ББК 32.813

Искусственный интеллект. Учебник для 10–11 классов (продвинутый уровень). — Авторский коллектив. — 2026.

Учебник содержит изложение математических основ современного искусственного интеллекта, включая численные методы оптимизации, основы машинного обучения и глубоких нейронных сетей. Предназначен для школьников, изучающих информатику на профильном уровне, а также для подготовки к олимпиадам и проектной деятельности.

© Авторский коллектив, 2026

Оглавление

Предисловие

Учебник адресован школьникам 10–11 классов, изучающим информатику на *продвинутом уровне*, а также тем, кто готовится к олимпиадам, профильным экзаменам и проектным работам в области искусственного интеллекта. Цель книги — показать, что современный ИИ — это не магия и не «чёрный ящик», а аккуратная математика и инженерия, доступные старшекласснику, владеющему школьным курсом алгебры и началами анализа.

Изложение построено по принципу *от формулы к коду и обратно*. Каждый ключевой алгоритм сопровождается:

- исторической справкой и геометрической интуицией;
- математически строгим выводом и теоремой о свойствах;
- рабочей реализацией на языке Python;
- численным экспериментом с графиками и обсуждением границ применимости.

Параграфы, помеченные звёздочкой ($\star$), необязательны при первом прочтении: в них собран более продвинутый материал, рассчитанный на читателя, готового к доказательствам. Однако мы рекомендуем вернуться к ним позднее: именно эти разделы дают понимание, *почему* методы машинного обучения работают, а не просто *как* ими пользоваться.

Условные обозначения, используемые в книге:

$\mathbb{R}, \mathbb{N}, \mathbb{Z}$	множества вещественных, натуральных и целых чисел
$f'(x)$, $f''(x)$	первая и вторая производные функции f
$\|\cdot\|$	норма (длина) вектора
$\arg\min_x f(x)$	точка минимума функции f

Желаем интересного чтения!

Часть I

Введение в искусственный интеллект

Глава 1

Что такое искусственный интеллект

В этой главе мы дадим неформальное представление о том, какие задачи решает искусственный интеллект, и наметим математический «инструментарий», без которого современный ИИ не существовал бы.

1.1 Краткая история идеи

[Раздел-заготовка для авторского текста: от Алана Тьюринга и Дартмутской конференции 1956 года до глубокого обучения и больших языковых моделей. Здесь полезно вставить временну́ю шкалу с помощью tikz и набор гиперссылок на оригинальные статьи.]

1.2 Чему мы научимся в этой книге

В книге читателю встретятся три «слоя» материала:

1. **Математические основы**: линейная алгебра, элементы анализа, теория вероятностей и численные методы оптимизации.
2. **Алгоритмы машинного обучения**: от линейной регрессии до многослойных нейронных сетей.
3. **Современный ИИ**: трансформеры, большие языковые модели, генеративные методы.

В главе 2 мы подробно изучим один из старейших и до сих пор важнейших численных методов — *метод касательных Ньютона*, который используется и для решения уравнений, и для обучения современных нейросетей.

Часть II

Математические основы

Глава 2

Численные методы оптимизации

2.1 Метод касательных Ньютона и его приложения

2.1.1 От Ньютона до Канторовича

Задачи отыскания корня уравнения $f(x) = 0$ и поиска минимума функции $g(x)$ относятся к разным «жанрам» школьной математики, однако численно они решаются почти одним и тем же приёмом — итерационной схемой, придуманной Исааком Ньютоном.

> В работе *De analysi per aequationes numero terminorum infinitas* (1669) и затем в *Method of Fluxions* (рукопись 1671 г., издание 1736 г.) Ньютон предложил способ приближённого решения уравнений, который позже Джозеф Рафсон в 1690 г. переписал в современной рекуррентной форме. Поэтому в англоязычной литературе метод часто называют *Newton–Raphson*.
> Существенный вклад в строгое исследование метода внёс советский математик **Леонид Витальевич Канторович** (1912–1986), лауреат Нобелевской премии по экономике 1975 г. (за теорию оптимального распределения ресурсов). В 1948 г. он опубликовал теорему, носящую теперь его имя: она впервые дала универсальные *достаточные условия сходимости* метода Ньютона, причём не только для уравнений с одной переменной, но и для уравнений в бесконечномерных банаховых пространствах.

Геометрическая идея. Пусть требуется найти корень уравнения $f(x) = 0$, и нам известна точка x_k, близкая к корню. Заменим график $y = f(x)$ его *касательной* в точке $(x_k, f(x_k))$. Касательная — это прямая, и её пересечение с осью x найти просто. Точку этого пересечения и объявим следующим приближением x_{k+1}. Затем повторим. Так получается метод касательных (рис. 2.1).

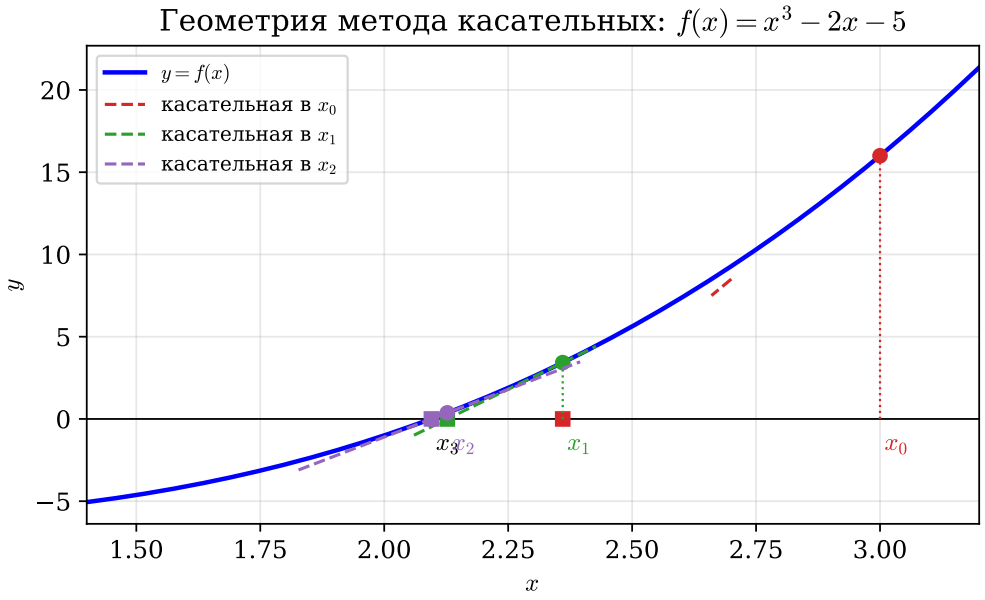


Рис. 2.1. Три итерации метода Ньютона для уравнения $x^3 - 2x - 5 = 0$. Каждая касательная (пунктир) пересекает ось абсцисс в точке, которая берётся в качестве следующего приближения (квадрат). После трёх шагов корень найден с точностью лучше 10^{-6}.

Уравнение касательной к графику $y = f(x)$ в точке x_k имеет вид $y = f(x_k) + f'(x_k)\,(x - x_k)$.

Полагая $y = 0$ и выражая x, получаем

$$\boxed{x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}} \qquad k = 0, 1, 2, \ldots \tag{2.1}$$

Это и есть *итерационная формула Ньютона*. Точка x_0 выбирается исследователем; от её выбора может зависеть, сойдётся ли метод вообще.

2.1.2 Метод Ньютона для задач оптимизации

Пусть теперь нужно найти минимум функции g. В любой точке минимума $x^\star$ выполнено $g'(x^\star) = 0$. Значит, минимизация дифференцируемой функции — это решение уравнения

$$g'(x) = 0.$$

Применим к нему формулу (2.1) с $f = g'$ (и тогда $f' = g''$):

$$\boxed{x_{k+1} = x_k - \frac{g'(x_k)}{g''(x_k)}} \tag{2.2}$$

Эту же формулу можно получить иначе — и это ценная для всей теории оптимизации точка зрения. Разложим g в окрестности x_k в **ряд Тейлора**:

$$g(x) \;=\; g(x_k) \;+\; g'(x_k)(x - x_k) \;+\; \tfrac{1}{2} g''(x_k)(x - x_k)^2 \;+\; o\big((x - x_k)^2\big). \tag{2.3}$$

Откуда берётся такое разложение? Идея проста: рядом с точкой x_k функция g ведёт себя «почти как многочлен». Линейное приближение $g(x_k) + g'(x_k)(x - x_k)$ — это уже знакомая нам касательная к графику. Если мы хотим добиться большей точности, надо учесть и *кривизну* графика — а она как раз кодируется второй производной g''. Так возникает квадратичный многочлен в (2.3).

Отбросим бесконечно малое слагаемое и обозначим оставшийся квадратичный многочлен через

$$q_k(x) \;=\; g(x_k) + g'(x_k)(x - x_k) + \tfrac{1}{2} g''(x_k)(x - x_k)^2.$$

Минимизировать q_k умеет любой школьник: это квадратное уравнение, решаемое в одну строку. Условие $q_k'(x) = 0$ даёт $g'(x_k) + g''(x_k)(x - x_k) = 0$, откуда $x = x_k - g'(x_k)/g''(x_k)$, что в точности совпадает с формулой (2.2).

> Метод Ньютона для оптимизации — классический пример принципа *разделяй и властвуй*: вместо того чтобы решать сложную задачу минимизации произвольной функции g целиком, мы заменяем её на каждом шаге простой подзадачей — минимизацией параболы q_k. Парабола подбирается так, чтобы локально (в окрестности x_k) имитировать поведение g, и её минимум вычисляется явно.

2.1.3 Пример 1. Извлечение квадратного корня

Пусть $a > 0$. Вычислим $\sqrt{a}$, применяя метод Ньютона к уравнению

$$f(x) = x^2 - a = 0.$$

Тогда $f'(x) = 2x$, и формула (2.1) принимает вид

$$\boxed{x_{k+1} = \frac{1}{2}\left(x_k + \frac{a}{x_k}\right)} \tag{2.4}$$

Это — знаменитый **метод Герона**, или вавилонский метод квадратного корня.

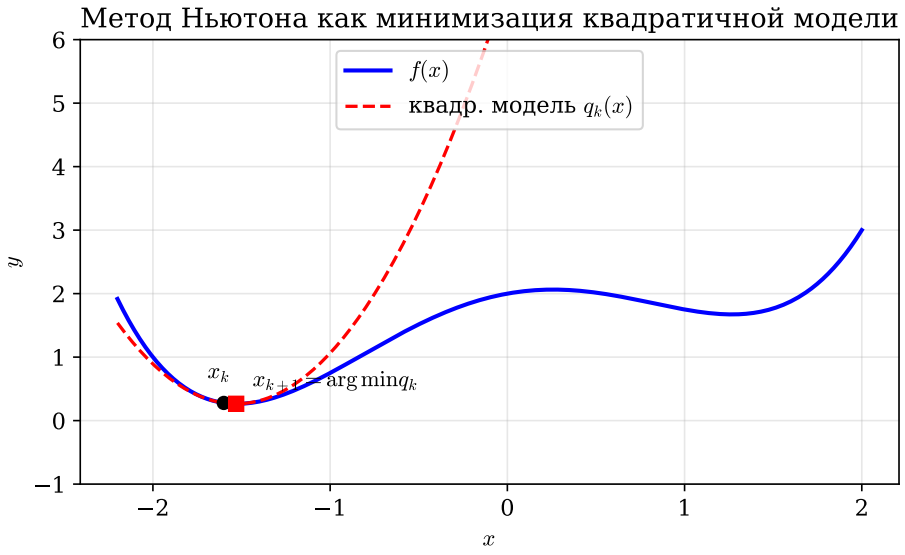


Рис. 2.2. Метод Ньютона как минимизация локальной квадратичной модели q_k (пунктир): следующая точка x_{k+1} — это вершина параболы, касающейся графика g в точке x_k.

Формулу (2.4) в явном виде описал *Герон Александрийский* (I век н. э.) в трактате «Метрика». Однако эту итерацию умели применять задолго до него: на вавилонской глиняной табличке YBC 7289 (примерно XVIII–XVI вв. до н. э.) сохранилось вычисление $\sqrt{2}$ с точностью до шестого десятичного знака — именно с помощью среднего арифметического и среднего гармонического, что эквивалентно (2.4). Получается, что одна из итерационных схем «пережила» три с половиной тысячи лет и по сей день используется в библиотеках вычисления корня.

Сходимость через сжимающее отображение

Запишем (2.4) как $x_{k+1} = T(x_k)$, где $T(x) = \frac{1}{2}(x + a/x)$. Точка $\sqrt{a}$ — *неподвижная* для T: $T(\sqrt{a}) = \sqrt{a}$.

Теорема 2.1. (Сходимость метода Герона)

Для любого $x_0 > 0$ последовательность (2.4) монотонно убывает (начиная с $k = 1$) и сходится к $\sqrt{a}$. Более того, для ошибки $e_k = x_k - \sqrt{a}$ выполнено

$$e_{k+1} = \frac{e_k^2}{2x_k}, \qquad \text{и при } k \geq 1\colon \quad 0 \leq e_{k+1} \leq \frac{e_k^2}{2\sqrt{a}}. \tag{2.5}$$

Доказательство. **Шаг 1 (нижняя оценка).** По неравенству о средних (AM–GM) для $x_0 > 0$: $x_1 = \frac{1}{2}(x_0 + a/x_0) \geq \sqrt{x_0 \cdot a/x_0} = \sqrt{a}$. Значит, начиная с x_1, все члены лежат в $[\sqrt{a}, \infty)$.

Шаг 2 (рекуррентность для ошибки).

$$e_{k+1} = x_{k+1} - \sqrt{a} = \frac{x_k + a/x_k}{2} - \sqrt{a} = \frac{x_k^2 - 2\sqrt{a}\,x_k + a}{2x_k} = \frac{(x_k - \sqrt{a})^2}{2x_k} = \frac{e_k^2}{2x_k}.$$

При $x_k \geq \sqrt{a}$ имеем $\frac{1}{2x_k} \leq \frac{1}{2\sqrt{a}}$, откуда вторая часть (2.5).

Шаг 3 (сжатие). Производная $T'(x) = \frac{1}{2}(1 - a/x^2)$. На луче $x \geq \sqrt{a}$ выполнено $T'(x) \in [0, \frac{1}{2})$. Значит, T — *сжимающее отображение* (с константой $\frac{1}{2}$) на $[\sqrt{a}, \infty)$. По теореме Банаха о неподвижной точке последовательность $x_k = T^k(x_1)$ сходится к единственной неподвижной точке $\sqrt{a}$. $\square$

Из (2.5) видно, что после первой итерации ошибка *квадрируется*: каждые два десятичных знака удваиваются за один шаг. Поэтому при разумном x_0 хватает 4–5 итераций, чтобы получить машинную точность.

Пример 2.2. Расстояние до горизонта

Стоящий человек ростом $h = 1{,}70$ м смотрит на море. До какой точки далеко он видит? Из теоремы Пифагора (рис. 2.3) для прямой, касательной к поверхности Земли:

$$d^2 + R^2 = (R+h)^2, \qquad d = \sqrt{2Rh + h^2} \;\approx\; \sqrt{2Rh}.$$

При $R = 6371$ км, $h = 1{,}7 \cdot 10^{-3}$ км:

$$2Rh \;\approx\; 2 \cdot 6371 \cdot 0{,}0017 \;=\; 21{,}66 \text{ (км}^2\text{)}.$$

Применим к этому числу метод Герона со стартом $x_0 = 5$:

k	x_k
0	5,000 000
1	$\frac{1}{2}(5 + 21{,}66/5) = 4{,}666\,000$
2	4,654 463
3	4,654 460

Получаем $d \approx 4{,}65$ км — расстояние, которое запоминается любому, кто хоть раз стоял у моря.

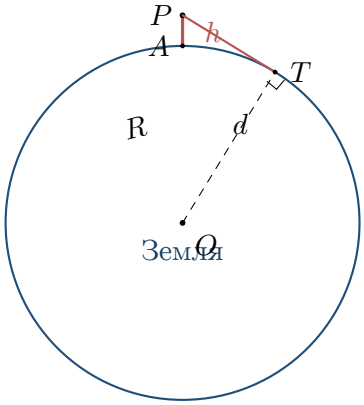


Рис. 2.3. Расстояние до горизонта $d = \sqrt{(R+h)^2 - R^2} \approx \sqrt{2Rh}$.

2.1.4 Пример 2. Вычисление обратного числа

Что, если на нашем процессоре нет аппаратного деления, и нужно вычислить $1/a$ при $a > 0$ (только сложениями и умножениями)? Применим метод Ньютона к уравнению

$$f(x) = \frac{1}{x} - a = 0.$$

Тогда $f'(x) = -1/x^2$, и подстановка в (2.1) даёт

$$x_{k+1} = x_k - \frac{1/x_k - a}{-1/x_k^2} = x_k + x_k^2\bigl(1/x_k - a\bigr) = x_k\bigl(2 - a x_k\bigr).$$

То есть

$$\boxed{x_{k+1} = x_k\,(2 - a x_k)} \tag{2.6}$$

Эта схема замечательна тем, что в ней нет ни одного деления — только сложение и умножение. Поэтому именно она исторически использовалась в первых ЭВМ для реализации операции деления на аппаратном уровне.

Анализ сходимости

Введём «относительную невязку» $u_k = 1 - a x_k$. Тогда

$$u_{k+1} = 1 - a x_{k+1} = 1 - a x_k(2 - a x_k) = 1 - 2a x_k + (a x_k)^2 = (1 - a x_k)^2 = u_k^2.$$

Таким образом

$$u_k = u_0^{2^k} \quad \Longrightarrow \quad u_k \to 0 \;\Leftrightarrow\; |u_0| < 1. \tag{2.7}$$

Условие $|1 - ax_0| < 1$ эквивалентно $0 < x_0 < 2/a$. *Радиусом сходимости* метода является интервал $\big(0,\, 2/a\big)$, и он **конечен**: вне этого интервала ($x_0 > 2/a$ или $x_0 < 0$) итерации расходятся. Это принципиальное отличие от метода Герона, который сходится для любого $x_0 > 0$.

> Если в (2.6) заменить число x_k на квадратную матрицу X_k, а число a — на матрицу A, получим знаменитую матричную итерацию **Ньютона–Шульца**: $X_{k+1} = X_k(2I - AX_k)$. Она вычисляет A^{-1} без обращения матриц на каждом шаге — лишь умножениями.
> Аналогичные полиномиальные итерации применяются для приближённой *ортогонализации* матриц. Именно так устроен оптимизатор **Muon**, появившийся в 2024–2025 гг. и используемый при обучении современных больших языковых моделей: в нём матрица градиента на каждом шаге быстро ортогонализуется несколькими итерациями Ньютона–Шульца — это намного дешевле, чем точное сингулярное разложение.

2.1.5 Реализация на Python и численный эксперимент

Запишем универсальную функцию, реализующую метод Ньютона по формуле (2.1), и применим её к двум разобранным примерам. Комментарии в коде даны по-английски (как принято в большинстве библиотек): `f`, `df` — сама функция и её производная; `x0` — начальное приближение; `n_iter` — максимальное число итераций; `tol` — точность остановки; функция возвращает массив `xs` с историей итераций.

```python
import numpy as np
import matplotlib.pyplot as plt


def newton(f, df, x0, n_iter=20, tol=1e-15):
    """Newton's tangent method.

    Parameters
    ----------
    f, df  : callables f(x) and f'(x).
    x0     : initial guess.
    n_iter : maximum number of iterations.
    tol    : stopping tolerance on |f(x)|.

    Returns
    -------
    Array of iterates, including x0.
    """
    xs = [x0]
    x = x0
    for _ in range(n_iter):
        fx = f(x)
        if abs(fx) < tol:
            break
        x = x - fx / df(x)
        xs.append(x)
    return np.array(xs)


# ---- Example 1. Square root of a ---------------------------------
a = 2.0
xs_sqrt = newton(lambda x: x**2 - a,
                 lambda x: 2 * x,
                 x0=50.0)          # very poor initial guess
```

```
err_sqrt = np.abs(xs_sqrt - np.sqrt(a))

# ---- Example 2. Reciprocal 1/a ----------------------------------
a = 3.0
xs_inv  = newton(lambda x: 1.0 / x - a,
                 lambda x: -1.0 / x**2,
                 x0=0.20)          # inside the convergence radius (0, 2/a)
err_inv = np.abs(xs_inv - 1.0 / a)

# ---- Convergence plot in log scale ------------------------------
plt.semilogy(err_sqrt, 'o-', label=r'sqrt: $|x_k-\sqrt{2}|$')
plt.semilogy(err_inv,  's-', label=r'1/a:  $|x_k-1/3|$')
plt.xlabel('iteration $k$'); plt.ylabel('error')
plt.legend(); plt.grid(True); plt.show()
```

Листинг 2.1. Универсальный метод касательных Ньютона и его применение.

Что показывает эксперимент. На рис. 2.4 приведены кривые сходимости метода Герона для $\sqrt{2}$ при разных стартах $x_0 \in \{0{,}1;\ 1;\ 5;\ 50\}$. Все четыре кривые сходятся к нулю, причём поведение типичное для квадратичной сходимости: после нескольких «согласующих» шагов число верных знаков удваивается за итерацию. Это иллюстрирует **глобальную сходимость** метода Герона: годится *любое* начальное приближение из $(0, \infty)$.

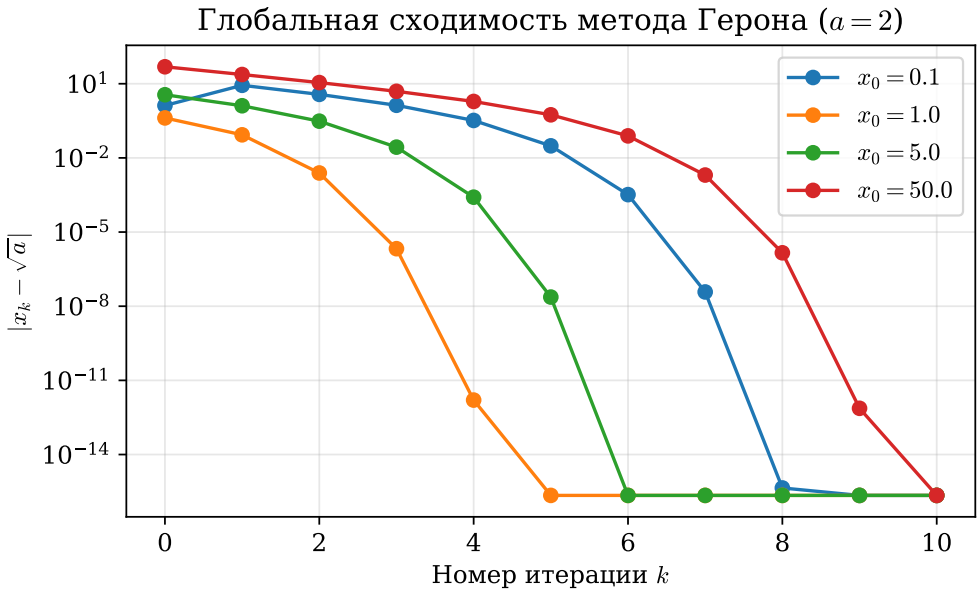


Рис. 2.4. Глобальная сходимость метода Герона для $\sqrt{2}$ из четырёх очень разных стартовых точек. Шкала ошибки логарифмическая.

Совсем другое поведение демонстрирует метод Ньютона–Шульца для $1/a$ при $a = 3$ (рис. 2.5): радиус сходимости равен интервалу $(0,\ 2/3) \approx (0,\ 0{,}667)$. Стартуя *внутри* этого интервала, метод сходится квадратично (нижние три кривые); на самой границе ($x_0 = 0{,}66$) — сходится, но «впритык»; а уже при $x_0 = 0{,}70$ ошибка взрывообразно растёт: $|u_k| = |u_0|^{2^k} \to \infty$ в полном согласии с (2.7). Это **локальная сходимость**.

2.1.6 Достаточные условия сверхлинейной сходимости*

Сравнение двух примеров наводит на вопрос: *когда именно* метод Ньютона сходится так быстро, как мы наблюдаем? Ответ даёт следующая теорема, восходящая к Канторовичу.

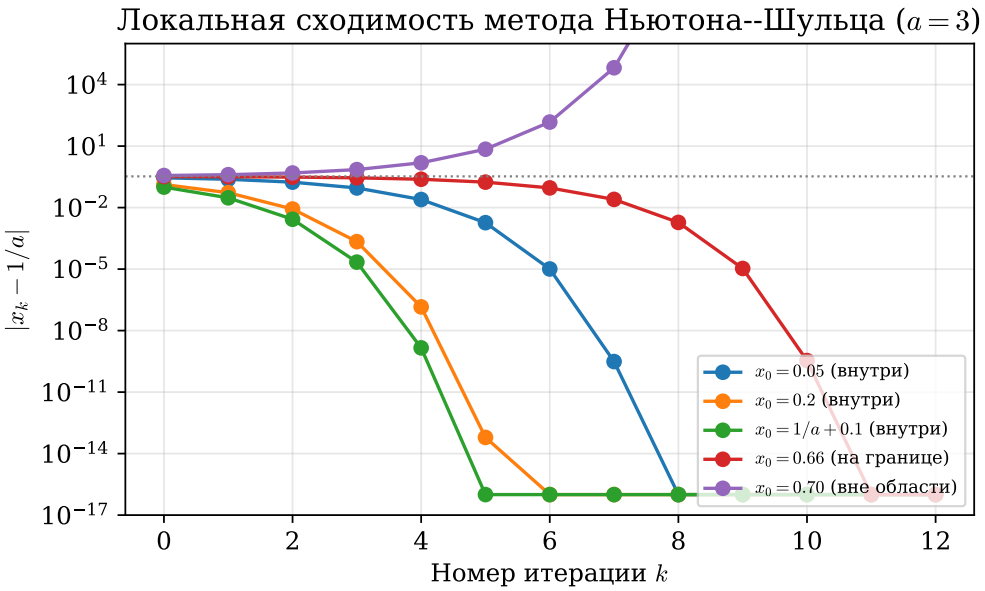


Рис. 2.5. Локальная сходимость метода Ньютона–Шульца для 1/3. Радиус сходимости — интервал $(0,\ 2/3)$; вне его метод расходится.

Определение 2.3. Сверхлинейная и квадратичная сходимости

Говорят, что последовательность $\{x_k\}$ сходится к $x^\star$ со **сверхлинейной скоростью**, если

$$\lim_{k\to\infty} \frac{|x_{k+1} - x^\star|}{|x_k - x^\star|} = 0.$$

Если же выполнено более сильное условие $|x_{k+1} - x^\star| \le M|x_k - x^\star|^2$ для некоторой константы $M > 0$, то сходимость называют **квадратичной**.

Определение 2.4. Сильно выпуклая функция

Функция $g\colon \mathbb{R} \to \mathbb{R}$ называется **сильно выпуклой** с константой $\mu > 0$, если $g \in C^2$ и $g''(x) \ge \mu$ для всех x. Сильная выпуклость гарантирует существование и единственность точки минимума и, что важно для нас, отделённость второй производной от нуля.

Теорема 2.5. (Сверхлинейная сходимость метода Ньютона)

Пусть выполнено хотя бы одно из условий:

(а) (*уравнение*) $f \in C^2$ в окрестности точки $x^\star$, $f(x^\star) = 0$ и $f'(x^\star) \ne 0$;

(б) (*сильно выпуклая оптимизация*) $g \in C^3$, g — сильно выпуклая на интервале, содержащем точку минимума $x^\star$, и применяется итерация (2.2).

Тогда существуют $\delta > 0$ и $M > 0$ такие, что для всякого $x_0 \in (x^\star - \delta,\ x^\star + \delta)$ итерации Ньютона определены и

$$|x_{k+1} - x^\star| \ \le\ M\,|x_k - x^\star|^2. \tag{2.8}$$

В частности, сходимость квадратичная (а значит, сверхлинейная).

Доказательство. Достаточно доказать случай (а): задача (б) сводится к нему, если положить $f = g'$, тогда условие $f'(x^\star) \ne 0$ эквивалентно $g''(x^\star) \ge \mu > 0$.

Разложим f по формуле Тейлора с остаточным членом в форме Лагранжа в окрестно-

сти x_k и подставим $x = x^\star$:

$$0 \;=\; f(x^\star) \;=\; f(x_k) + f'(x_k)(x^\star - x_k) + \tfrac{f''(\xi_k)}{2}(x^\star - x_k)^2,$$

где ξ_k — некоторая точка между x_k и $x^\star$. Разделим обе части на $f'(x_k)$ (по непрерывности эта величина не равна нулю в малой окрестности $x^\star$):

$$0 \;=\; \frac{f(x_k)}{f'(x_k)} + (x^\star - x_k) + \frac{f''(\xi_k)}{2f'(x_k)}(x^\star - x_k)^2.$$

Из определения $x_{k+1} = x_k - f(x_k)/f'(x_k)$ получаем $\frac{f(x_k)}{f'(x_k)} = x_k - x_{k+1}$. Подставляя:

$$0 = (x_k - x_{k+1}) + (x^\star - x_k) + \frac{f''(\xi_k)}{2f'(x_k)}(x^\star - x_k)^2,$$

откуда после очевидных переобозначений

$$x_{k+1} - x^\star \;=\; \frac{f''(\xi_k)}{2f'(x_k)}\,(x_k - x^\star)^2. \tag{2.9}$$

Это и есть искомая *рекуррентность для расстояния*.

Теперь подберём константы. По непрерывности f' и f'' существует $\delta_1 > 0$ и числа L, $m > 0$ такие, что $|f''(\xi)| \le L$ и $|f'(x)| \ge m$ для всех $\xi, x \in [x^\star - \delta_1,\, x^\star + \delta_1]$. Положим $M = L/(2m)$ и $\delta = \min\bigl(\delta_1,\, \frac{1}{2M}\bigr)$. Тогда из (2.9) при $|x_k - x^\star| < \delta$:

$$|x_{k+1} - x^\star| \le M|x_k - x^\star|^2 \le M\delta \cdot |x_k - x^\star| \le \tfrac{1}{2}|x_k - x^\star| < \delta.$$

Значит, последовательность не покидает δ-окрестность и оценка (2.8) выполнена для всех k. $\square$

Где условия выполняются для наших примеров? **Метод Герона.** Здесь $f(x) = x^2 - a$, $f'(x) = 2x$, $f''(x) = 2$. В точке $x^\star = \sqrt{a} > 0$ выполнено $f'(x^\star) = 2\sqrt{a} \neq 0$. *Условия теоремы выполняются на всём луче* $(0, \infty)$, *не содержащем ноль*, и константа в (2.8) принимает вид $M = \frac{1}{2\sqrt{a}}$ — в полном согласии с явным выражением (2.5), полученным элементарно.

Метод Ньютона–Шульца. Здесь $f(x) = 1/x - a$, $f'(x) = -1/x^2$, $f''(x) = 2/x^3$. В точке $x^\star = 1/a$ имеем $f'(x^\star) = -a^2 \neq 0$, и теорема применима в малой окрестности этой точки. Однако, в отличие от метода Герона, оценка снизу $|f'(x)| \ge m > 0$ *ломается* при $x \to \infty$: производная стремится к нулю. Именно поэтому радиус сходимости в этом примере конечен и равен $1/a$ слева и справа от $x^\star$, что в сумме даёт интервал $(0,\, 2/a)$, найденный нами ранее аналитически.

Таким образом, два простейших примера полностью укладываются в доказанную теорему: метод Ньютона сходится сверхлинейно *там, где* производная регулярна и отделена от нуля. Глобальную же сходимость, как у Герона, приходится доказывать дополнительно — например, ссылаясь на структуру функции (выпуклость) и теорему Банаха о сжимающем отображении, как мы и сделали в разделе 2.1.3.

Задачи для самостоятельной работы

1. Применяя метод Ньютона к уравнению $f(x) = x^n - a = 0$, получите итерационную формулу для $\sqrt[n]{a}$. Покажите, что для $n = 3$ она имеет вид $x_{k+1} = \frac{2x_k + a/x_k^2}{3}$.
2. Докажите, что для метода Герона из любого $x_0 > 0$ погрешность e_1 всегда положительна, после чего последовательность монотонно убывает. (Указание: используйте AM–GM.)

3. Для метода Ньютона–Шульца при $a = 2$ найдите наибольшее x_0, при котором итерации ещё сходятся. Сравните численно скорость сходимости при $x_0 = 0{,}1$ и $x_0 = 0{,}9$.
4. * Сформулируйте и докажите аналог теоремы 2.5 для метода Ньютона минимизации многомерной сильно выпуклой функции $g\colon \mathbb{R}^n \to \mathbb{R}$. Какую роль играет матрица Гессе $\nabla^2 g$?
5. * Прочитайте о теореме Канторовича в учебнике «Kantorovich theorem» и сформулируйте её в одну фразу: какие именно три величины должны быть «в нужном соотношении» для гарантии сходимости?

Глава 3

Вычислительная линейная алгебра

Если метод Ньютона из главы 2 был одним маленьким уравнением, упрямо повторяющимся, то линейная алгебра — это язык, на котором говорят сразу о тысячах уравнений и миллионах чисел. Этой главой мы покажем, что многие фокусы современного искусственного интеллекта — от того, как нейросеть «узнаёт» лицо человека, до того, как поисковик ранжирует страницы Интернета, — держатся всего на нескольких разложениях матриц. Мы посмотрим четыре истории: лица, котиков, Интернет и устойчивость нейросетей.

3.1 Зачем линейной алгебре отдельная глава

Вы уже встречались с матрицами в школьном курсе — как с прямоугольными таблицами чисел. В современном ИИ матрица — это **универсальный носитель данных**.

- Чёрно-белая фотография 1024×1024 — это матрица размера 1024×1024, где каждое число — яркость пикселя от 0 (чёрный) до 255 (белый). Цветная — три такие матрицы.
- Текст из N слов словаря — это вектор частот длины N (модель «мешка слов»), и набор из M документов — матрица $M \times N$.
- Социальная сеть из N пользователей — матрица $N \times N$, где элемент (i, j) равен 1, если i подписан на j.
- Веса одного слоя нейросети — матрица W, через которую входной сигнал умножается, чтобы получить выходной.

С такими матрицами надо уметь *что-то делать*: сжимать, сравнивать, упорядочивать, обновлять. И почти всегда оказывается, что нужная операция выражается через одно и то же фундаментальное разложение матрицы — **сингулярное**. Мы изучим его и применим к четырём задачам:

1. **Узнать лицо** по фотографии — *eigenfaces* (§3.5).
2. **Сжать пространство признаков** — *eigencats* и теорема Эккарта–Янга (§3.4, §3.6).
3. **Упорядочить весь Интернет** — *PageRank* (§3.7).
4. **Понять, когда нейросеть устойчива** — *оценки Липшицевости* (§3.8).

Прежде чем перейти к сюжетам, разберёмся с одной *очень* земной проблемой: с тем, как компьютер хранит числа.

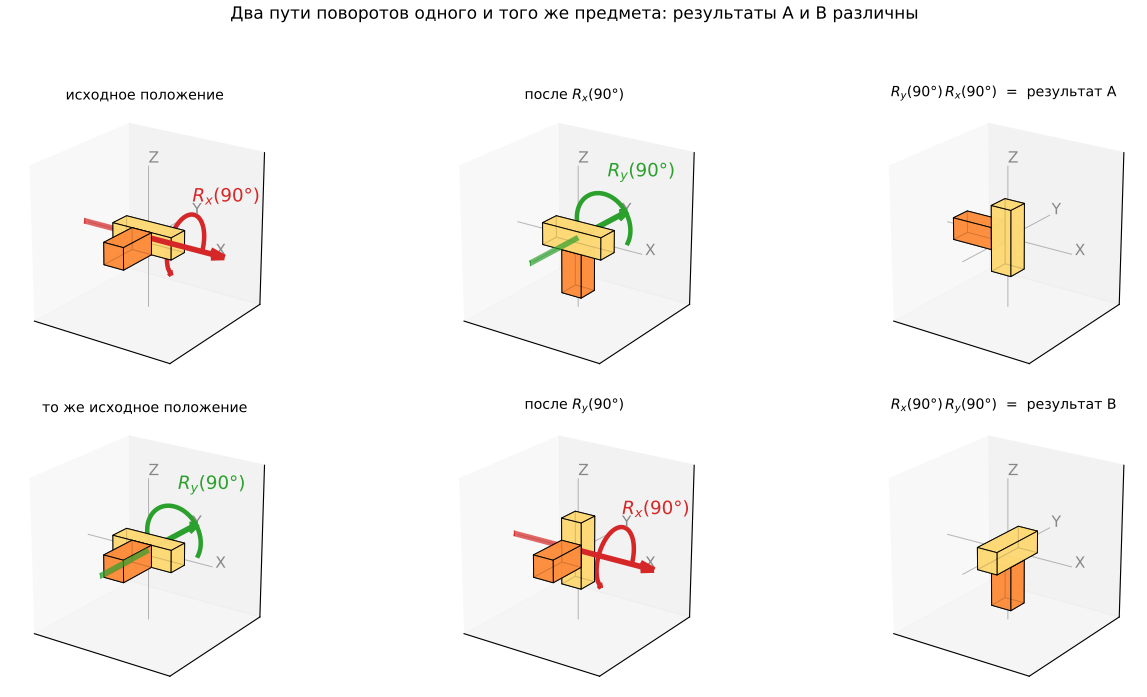
$AB \neq BA$, или почему линейная алгебра не похожа на школьную

В школе вы привыкли, что от перестановки множителей произведение не меняется: $2 \cdot 3 = 3 \cdot 2$. Для квадратных матриц это *в общем случае не так* — они не коммутируют. Самая наглядная демонстрация — повороты трёхмерного предмета. Возьмите смартфон, поверните его сначала на 90° вокруг горизонтальной оси X (вершина «нырнула» вперёд), потом на 90° вокруг вертикальной оси Y (предмет повернулся вправо). Теперь повторите эксперимент, но в обратном порядке. Финальные положения смартфона будут *разными*. Каждый поворот задаётся ортогональной матрицей 3×3, и эти матрицы

не коммутируют:

$$R_y(90^\circ)\,R_x(90^\circ) \;\neq\; R_x(90^\circ)\,R_y(90^\circ).$$

Геймеры одной из лучших игр 2023 года *The Legend of Zelda: Tears of the Kingdom* ощущают этот эффект каждый раз, когда крутят объект в инвентаре. Подробный разбор сюжета: t.me/fminxyz/15.



На двумерной плоскости повороты, кстати, коммутируют: $R(\alpha)R(\beta) = R(\beta)R(\alpha) = R(\alpha+\beta)$. Так что эффект чисто «трёхмерный»: чем больше степеней свободы, тем менее послушна алгебра.

3.2 Когда арифметика лжёт: числа с плавающей точкой

Откройте Python и наберите:

```
>>> 0.1 + 0.2
0.30000000000000004
>>> 0.1 + 0.2 == 0.3
False
```

Это не баг Python. То же самое выдаст любой современный язык — C, C++, Java, JavaScript. И этому есть фундаментальная причина: **двоичная система счисления** не умеет точно представить десятичную дробь 0,1.

3.2.1 Двоичные дроби и почему $0{,}1$ — это «$0{,}333\ldots$»

В десятичной системе $1/3 = 0{,}3333\ldots$ — бесконечная дробь. То же самое в двоичной системе случается с 0,1:

$$0{,}1_{10} \;=\; 0{,}0001\,1001\,1001\,1001\ldots_2$$

Дробь периодическая. Любой реальный компьютер хранит её, оборвав после конечного числа разрядов (обычно 52 двоичных знака после запятой — стандарт *IEEE 754, double precision*). Поэтому число, которое программист записывает как `0.1`, на самом деле равно

$$0{,}1 \;\approx\; 0{,}100\,000\,000\,000\,000\,005\,551\,115\ldots$$

Когда мы складываем такое «искажённое 0,1» с искажённым 0,2, ошибки усиливаются — и получается тот самый «лишний хвостик» $4 \cdot 10^{-17}$, который мы видели в листинге.

Машинной точностью называют наименьшее положительное число ε, такое что в компьютерной арифметике $1 + \varepsilon \neq 1$. Для стандарта `double` оно равно $\varepsilon_{\text{маш}} = 2^{-52} \approx 2{,}22 \cdot 10^{-16}$.
Это означает: какое бы хорошее число x вы ни записали, оно известно компьютеру лишь с относительной точностью $\sim \varepsilon_{\text{маш}}$.

3.2.2 Когда это становится опасным: катастрофическое сокращение

Маленькая ошибка в 17-м знаке — мелочь? Не всегда. Рассмотрим безобидное на вид вычисление:

$$g(x) = \frac{1 - \cos x}{x^2} \qquad \text{при } x = 10^{-8}.$$

По формуле Тейлора $1 - \cos x \approx x^2/2$, так что $g(10^{-8}) \approx 0{,}5$. Что выдаст компьютер?

```
>>> from math import cos
>>> x = 1e-8
>>> (1 - cos(x)) / x**2
0.0           # вместо 0.5!
```

Что произошло? Число $\cos(10^{-8}) \approx 1 - 5 \cdot 10^{-17}$ так близко к единице, что компьютер просто хранит его как *ровно единицу* (поправка $5 \cdot 10^{-17}$ меньше $\varepsilon_{\text{маш}}$). Разность $1 - \cos x$ обнулилась, и весь ответ обнулился вместе с ней.

Это явление называется **катастрофическим сокращением** (*catastrophic cancellation*): когда из двух близких чисел вычитают и теряют все значащие цифры разом.

Пример 3.1. Спасение через переписывание формулы

Воспользуемся тождеством $1 - \cos x = 2\sin^2(x/2)$:

$$g(x) = \frac{2\sin^2(x/2)}{x^2} = \frac{1}{2}\left(\frac{\sin(x/2)}{x/2}\right)^2.$$

Внутри теперь нет вычитания близких чисел, и Python выдаёт `0.4999999999999983` — погрешность лишь в последнем знаке.

Важно

Главный практический вывод. Если ваш численный алгоритм выдаёт странный ответ, первая гипотеза, которую следует проверить, — это не «бага в библиотеке», а *катастрофическое сокращение* в вашей формуле. Часто его можно убрать, переписав формулу алгебраически эквивалентным, но численно устойчивым способом.

Стандарт *IEEE 754* был принят в 1985 г., в значительной мере благодаря Уильяму Кэхэну (*William Kahan*, премия Тьюринга 1989 г.). До этого каждый производитель процессоров делал плавающую арифметику по-своему, и одна и та же программа на разных машинах могла давать заметно разные ответы. Кэхэн также является автором знаменитого **алгоритма компенсированного суммирования**, который позволяет складывать миллионы чисел с почти двойной точностью.

3.3 Сингулярное разложение SVD: главный фокус линейной алгебры

3.3.1 Геометрическая идея

Возьмём любую матрицу A размера $m \times n$ и применим её к единичному кругу в $\mathbb{R}^n$ (то есть к множеству всех векторов x с $\|x\| = 1$, где $\|\cdot\|$ — обычная евклидова длина). Что получится в $\mathbb{R}^m$?

Теорема 3.2. (Сингулярное разложение (SVD))

Для любой вещественной матрицы A размера $m \times n$ существует разложение

$$\boxed{A \;=\; U\,\Sigma\,V^{\top}} \tag{3.1}$$

где

- U — *ортогональная* матрица $m \times m$ (поворот в $\mathbb{R}^m$);
- V — ортогональная матрица $n \times n$ (поворот в $\mathbb{R}^n$);
- Σ — «диагональная» матрица $m \times n$ с неотрицательными числами $\sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_r > 0$ на диагонали (и нулями ниже), где $r = \operatorname{rank} A$.

Числа σ_i называются **сингулярными числами** матрицы A.

Содержательно (3.1) говорит: *любая* линейная операция — это три простых действия подряд:

$$\underbrace{V^{\top}}_{\text{поворот в исходном пространстве}} \;\to\; \underbrace{\Sigma}_{\text{растяжение по координатным осям}} \;\to\; \underbrace{U}_{\text{поворот в новом пространстве}}\,.$$

Геометрически это значит: образ единичной сферы под действием A — это **эллипсоид**, у которого полуоси равны $\sigma_1, \sigma_2, \ldots$ (см. рис. 3.1).

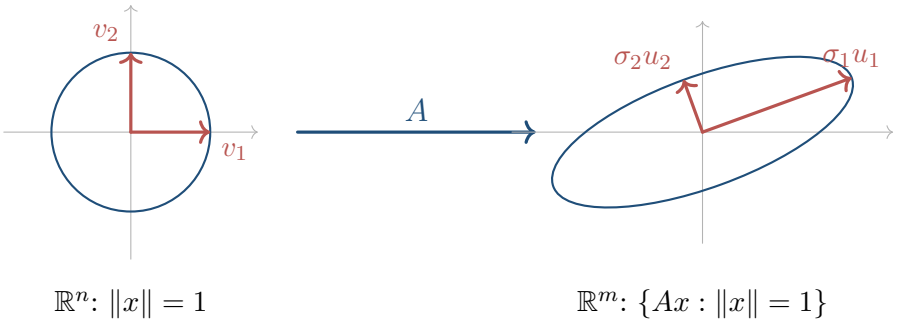


Рис. 3.1. SVD геометрически. Матрица A переводит единичный круг в эллипс, полуоси которого равны сингулярным числам σ_1, σ_2, а направления полуосей — столбцы матрицы U (левые сингулярные векторы). Векторы v_1, v_2 (столбцы V, правые сингулярные векторы) — это те направления в исходном пространстве, которые A переводит в полуоси эллипса.

3.3.2 Связь с собственными значениями

Сингулярные числа — это, по сути, переодетые собственные значения. А именно:

$$A^{\top}A \;=\; (U\Sigma V^{\top})^{\top}(U\Sigma V^{\top}) \;=\; V\,\Sigma^{\top}\Sigma\,V^{\top}. \tag{3.2}$$

Матрица $\Sigma^{\top}\Sigma$ — диагональная, на её диагонали стоят $\sigma_1^2, \sigma_2^2, \ldots$. Значит, σ_i^2 — это *собственные значения* симметричной матрицы $A^{\top}A$, а столбцы V — её собственные векторы.

Эта формула важна нам по двум причинам: (1) она обосновывает существование SVD (через известную спектральную теорему для симметричных матриц), (2) она будет рабочим инструментом в § 3.5, когда мы будем строить базис «собственных лиц» (eigenfaces).

Сингулярное разложение независимо открыли итальянский математик *Эудженио Бельтрами* (1873 г.) и француз *Камилл Жордан* (1874 г.). В современном виде, как разложение прямоугольной матрицы, его сформулировал *Карл Эккарт* с *Гейлом Янгом* в 1936 г. Численно надёжный алгоритм вычисления SVD появился лишь в 1965 г. (Голуб и Кахан) — и стал одним из ключевых алгоритмов XX века. Сегодня он реализован в любой математической библиотеке: `numpy.linalg.svd`, `scipy.linalg.svd`, `torch.svd`.

3.4 Лучшее малоранговое приближение: теорема Эккарта–Янга

Самое замечательное свойство SVD состоит в том, что оно позволяет *ужимать* матрицу, выбрасывая всё «несущественное».

3.4.1 Разложение по слагаемым ранга один

Перепишем (3.1) по-другому. Обозначим столбцы матрицы U через $u_1, \dots, u_m$ (это *левые сингулярные векторы*), а столбцы V — через $v_1, \dots, v_n$ (*правые сингулярные векторы*). Тогда A есть сумма r слагаемых, каждое из которых имеет ранг один:

$$A \;=\; \sum_{i=1}^{r} \sigma_i \, u_i v_i^{\top}. \tag{3.3}$$

Каждое слагаемое $u_i v_i^{\top}$ — это «вертикальный вектор умножить на горизонтальный», и оно само по себе является матрицей ранга один (рис. 3.2).

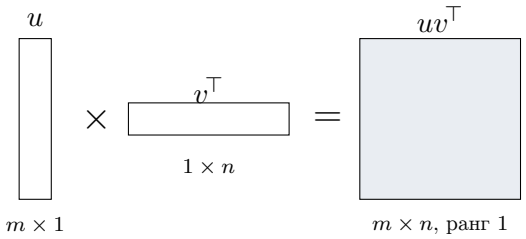


Рис. 3.2. Матрица ранга один: каждый столбец результата пропорционален u, каждая строка — пропорциональна $v^{\top}$. Чтобы её хранить, достаточно $m + n$ чисел вместо mn.

Если оборвать сумму (3.3) на k-м слагаемом ($k < r$), получим приближение:

$$A_k \;=\; \sum_{i=1}^{k} \sigma_i \, u_i v_i^{\top}. \tag{3.4}$$

Это матрица того же размера $m \times n$, но ранга всего лишь k.

3.4.2 Теорема Эккарта–Янга: A_k — наилучшее приближение

Теорема 3.3. (Эккарт–Янг–Мирский (1936/1960))

Среди всех матриц B размера $m \times n$ ранга не более k матрица A_k из (3.4) даёт *наименьшую* ошибку в смысле спектральной нормы:

$$\min_{\operatorname{rank} B \le k} \|A - B\|_2 \;=\; \|A - A_k\|_2 \;=\; \sigma_{k+1}. \tag{3.5}$$

Здесь $\|M\|_2 = \max_{\|x\|=1} \|Mx\| = \sigma_1(M)$ — **спектральная норма** матрицы M (равна её наибольшему сингулярному числу). Утверждение остаётся в силе и для *фробениусовой* нормы $\|M\|_F = \sqrt{\sum_{i,j} m_{ij}^2}$:

$$\|A - A_k\|_F \;=\; \sqrt{\sigma_{k+1}^2 + \sigma_{k+2}^2 + \ldots + \sigma_r^2}.$$

Смысл теоремы: *если хочется сжать матрицу до ранга k, лучшее, что вы можете сделать, — оставить первые k компонент SVD*. Ошибка контролируется *отброшенным* сингулярным числом σ_{k+1}.

3.4.3 Пример: сжатие изображения

Возьмём фотографию 1024×768 в градациях серого. Хранение «в лоб» — это $1024 \cdot 768 = 786\,432$ чисел. Применим к матрице фото SVD и оставим только k старших компонент: придётся хранить

$$k \cdot (1024 + 768) + k \;=\; k \cdot 1793$$

чисел — по одному столбцу u_i, одной строке $v_i^\top$ и одному числу σ_i на каждое слагаемое. Уже при $k = 50$ это $\sim 90\,000$ чисел — в *восемь* раз меньше исходного, а визуально потеря почти незаметна (рис. 3.3).

```python
import numpy as np
import matplotlib.pyplot as plt
from skimage import data, color

# Load and convert to grayscale
img = color.rgb2gray(data.astronaut())        # shape (512, 512)
U, S, Vt = np.linalg.svd(img, full_matrices=False)

# Reconstruct using only top-k singular components
ranks = [1, 5, 20, 50, 200]
fig, axes = plt.subplots(1, len(ranks)+1, figsize=(14, 3))
axes[0].imshow(img, cmap='gray'); axes[0].set_title('original')
for ax, k in zip(axes[1:], ranks):
    A_k = U[:, :k] @ np.diag(S[:k]) @ Vt[:k, :]
    ax.imshow(A_k, cmap='gray')
    ax.set_title(f'rank {k}')
for ax in axes: ax.axis('off')
plt.tight_layout(); plt.show()
```

А если картинка цветная? Цветное изображение $H \times W \times 3$ можно сжать тремя способами: (1) поканально, считая SVD отдельно для R, G, B; (2) перевести в $YCbCr$ и сжимать только канал яркости Y; (3) "развернуть" изображение в матрицу $H \times 3W$ и применить одну SVD. В простейшем поканальном варианте для ранга r нужно хранить $3r(H + W + 1)$ чисел вместо $3HW$:

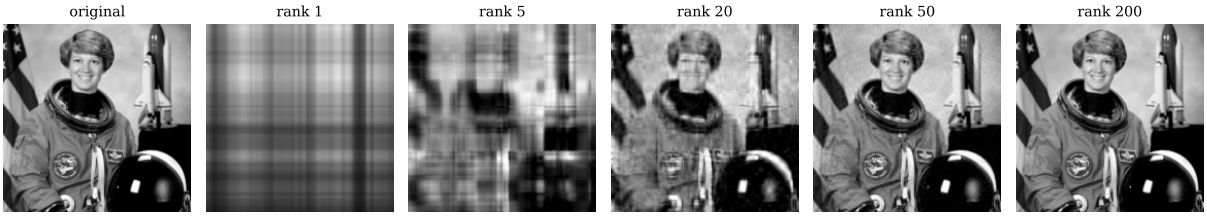


Рис. 3.3. Малоранговое приближение портрета астронавта. Уже при ранге 50 изображение визуально неотличимо от оригинала; при ранге 200 восстановление идеально. По теореме Эккарта–Янга это *лучшее* приближение фиксированного ранга.

Рис. 3.4. Цветную фотографию сжимаем поканально: SVD ранга r для каждого из R, G, B и склейка. Под каждым кадром — доля памяти от оригинала. Варианты для цвета: t.me/fminxyz/16.

> В 2021 г. Эдвард Ху с соавторами заметили: когда большую языковую модель дообучают на новую задачу, *обновление* матриц весов ΔW оказывается почти малоранговой матрицей. Поэтому вместо обучения «полной» ΔW можно сразу искать её в виде $\Delta W = UV^\top$ с маленьким k, где U, V — тонкие прямоугольники.
> Это метод **LoRA** (*Low-Rank Adaptation*). Для модели с 175 миллиардами параметров такой приём сокращает объём дообучаемых весов в тысячи раз — и именно благодаря ему сегодня можно «дофайнтюнить» большую модель на бытовой видеокарте.
> Теоретический фундамент LoRA — та самая теорема Эккарта–Янга 3.3, которой почти 90 лет.

3.5 Eigenfaces: как алгоритм узнаёт лицо

3.5.1 Постановка задачи

В 1991 г. Мэттью Тёрк и Алекс Пентланд из MIT поставили простой вопрос: *можно ли обучить компьютер узнавать лицо, не прибегая ни к каким специальным «лицевым» признакам — носу, глазам, контурам, — а просто работая с матрицей пикселей?* Их ответ — метод **собственных лиц** (*eigenfaces*) — стал классикой и открыл целую область computer vision.

Идея: каждая фотография лица — это точка в очень многомерном пространстве. Например, лицо 64×64 — это вектор длины $64 \cdot 64 = 4\,096$ в пространстве $\mathbb{R}^{4096}$. Двух разных людей снимали много раз — получаем облако точек в этом пространстве.

Главное наблюдение: *это облако очень узкое*. Хотя пространство 4 096-мерное, реальные лица занимают в нём подпространство размерности всего лишь ~ 50. Найти это подпространство — и есть задача.

Рис. 3.5. Фрагмент датасета *Olivetti Faces*: 400 фотографий 64×64 пикселя, 40 человек по 10 снимков каждого. Каждая фотография — точка в $\mathbb{R}^{4096}$.

3.5.2 Шаг 1. Среднее лицо и центрирование

Пусть у нас N фотографий $x_1, \ldots, x_N \in \mathbb{R}^d$ (где $d = 4096$ для 64×64). Вычислим **среднее лицо**:

$$\bar{x} \;=\; \frac{1}{N} \sum_{i=1}^{N} x_i.$$

И вычтем его из каждого изображения: $\tilde{x}_i = x_i - \bar{x}$. *Среднее лицо* (рис. 3.6) — это размытая маска, в которой видно «лицо вообще»: усреднённые глаза, нос, рот, овал. Дальше мы будем работать только с отклонениями от этого среднего.

Рис. 3.6. Среднее лицо по 400 снимкам датасета Olivetti.

3.5.3 Шаг 2. Главные направления изменчивости

Соберём центрированные изображения в строки матрицы $X \in \mathbb{R}^{N \times d}$ (одна строка = одно лицо). Применим к ней SVD:

$$X \;=\; U \, \Sigma \, V^{\top}.$$

Каждая *строка* $v_i^\top$ матрицы $V^\top$ — это вектор длины $d = 4096$, который можно нарисовать как картинку 64×64. Эти картинки и называются **собственными лицами** — eigenfaces (рис. 3.7). Содержательно:

- v_1 — направление наибольшей изменчивости. Чаще всего это «освещение слева/справа».
- v_2 — направление второй по величине изменчивости (часто «улыбка / не улыбка»).
- $v_3, \ldots$ — постепенно более тонкие признаки: очки, форма щёк, наклон головы, индивидуальные особенности.

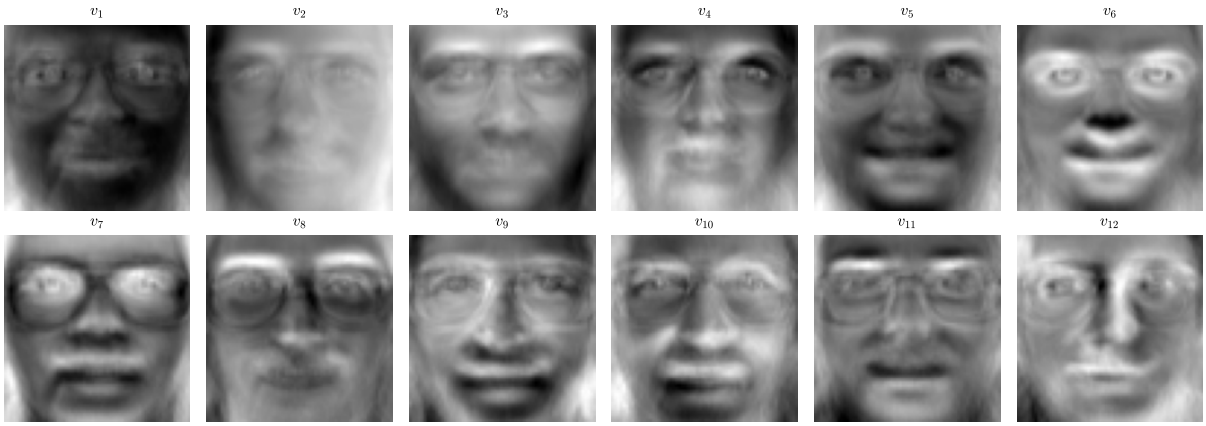


Рис. 3.7. Первые 12 eigenfaces датасета Olivetti, отрисованные как картинки 64×64. Эти странные «полу-лица» — координатные оси в пространстве, где живут все настоящие лица.

3.5.4 Шаг 3. Лицо как вектор из 50 чисел

Зафиксируем k (обычно $k = 30 \div 100$) и спроектируем каждое лицо x на эти k собственных лиц:

$$\alpha_i \;=\; v_i^\top (x - \bar{x}), \qquad i = 1, \ldots, k. \tag{3.6}$$

Получили вектор коэффициентов $\alpha = (\alpha_1, \ldots, \alpha_k) \in \mathbb{R}^k$. Это и есть «лицо человека» с точки зрения алгоритма — *всего* 50 *чисел* вместо 4096.

3.5.5 Шаг 4. Распознавание и реконструкция

Распознавание. Чтобы узнать, кто на новой фотографии y, вычислим её коэффициенты β по (3.6) и найдём в базе ближайшего соседа: $\hat{j} = \arg\min_j \left\| \beta - \alpha^{(j)} \right\|$.

Реконструкция. Зная α, можно восстановить лицо:

$$x \;\approx\; \bar{x} + \sum_{i=1}^{k} \alpha_i \, v_i.$$

Рис. 3.8 показывает, как качество реконструкции растёт с k.

3.5.6 Реализация на Python

```python
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import fetch_olivetti_faces

```

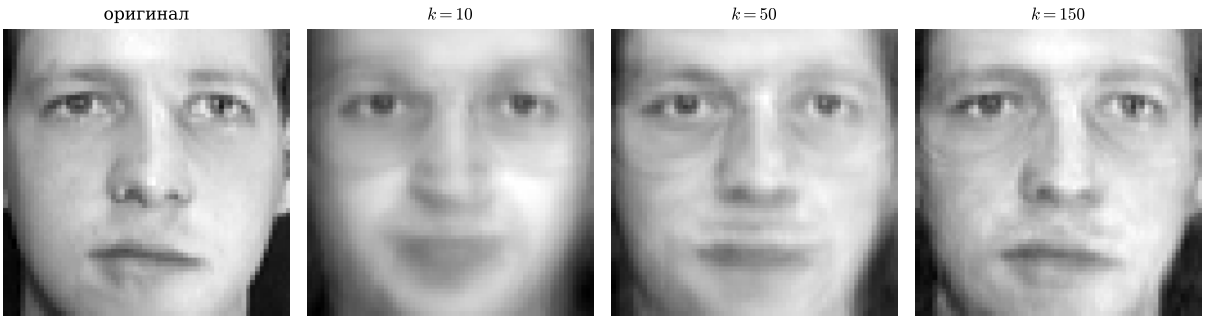


Рис. 3.8. Реконструкция одной из фотографий датасета через первые k eigenfaces. При $k = 10$ узнаётся «общий тип» лица; при $k = 50$ — уже конкретный человек; при $k = 150$ — почти оригинал.

```python
# 1. Load: 400 faces, 64x64 each
faces = fetch_olivetti_faces().images        # (400, 64, 64)
X = faces.reshape(400, -1)                   # (400, 4096)

# 2. Center: subtract the mean face
x_bar = X.mean(axis=0)
X_centered = X - x_bar

# 3. SVD => eigenfaces are rows of Vt
U, S, Vt = np.linalg.svd(X_centered, full_matrices=False)

# 4. Project all faces onto top-k eigenfaces
k = 50
alpha = X_centered @ Vt[:k].T                # (400, 50)

# 5. Recognise: nearest neighbour in alpha-space
def recognise(y, k=50):
    y_centered = y.flatten() - x_bar
    beta = Vt[:k] @ y_centered
    distances = np.linalg.norm(alpha - beta, axis=1)
    return distances.argmin()

# 6. Reconstruct a face from k coefficients
def reconstruct(idx, k=50):
    return x_bar + Vt[:k].T @ alpha[idx, :k]
```

Это интересно

Метод eigenfaces — родоначальник всех современных систем распознавания лиц, включая ту, что разблокирует ваш смартфон. Современные системы заменяют SVD на свёрточную нейросеть (см. §3.3.7), но базовая идея — *сжать лицо в короткий вектор, в котором близкие лица оказываются рядом* — осталась той же. Эти короткие векторы сегодня называют **эмбеддингами**.

3.6 Eigencats: тот же приём для котов*

Метод eigenfaces работает на любых однородных объектах, не только на лицах. Возьмём *Cats Faces Dataset*: 29 843 фотографии котов 64×64 пикселя. Векторизуем — и получим матрицу $A \in \mathbb{R}^{29\,843 \times 4096}$. Сингулярные векторы матрицы $A^\top A$ называют **eigencats**.

Как и в случае с лицами (§3.5), сами строки матрицы $V^\top$ образуют *базис* пространства котов. Каждая такая строка — собственный вектор размерности 4096, который можно нарисовать как картинку 64×64. Эти картинки и называют **eigencats**.

Рис. 3.9. Малоранговое восстановление четырёх котов на разных рангах r (анимация: t.me/fminxyz/12).

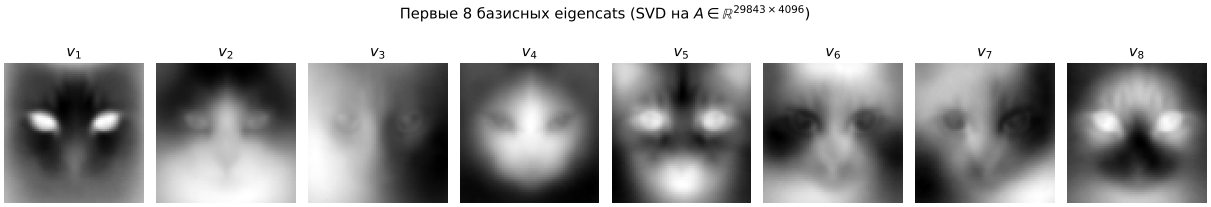


Рис. 3.10. Первые восемь eigencats — строки $V^\top$, свёрнутые в квадраты 64×64. v_1 ловит освещение и общую форму морды, v_2 — контраст верх/низ (наклон), v_3 — лево/право, дальше — всё более тонкие особенности.

В сжатом виде каждый кот превращается в ~ 50 чисел: коэффициенты $\alpha_i = v_i^\top(x - \bar{x})$. Этого хватает, чтобы потом восстановить узнаваемое изображение и отличать одного кота от другого.

> **Пространство «реальных» объектов всегда узкое.** Будь это лица людей, морды котов, рукописные цифры (датасет MNIST) или фотографии природы — реальные данные занимают в исходном многомерном пространстве *маленькое* подпространство. Распознавание = найти это подпространство и работать в нём.
> **Эта идея — фундамент современных автокодировщиков и эмбеддингов** (см. § 3.4.3–3.4.6). SVD — её самая ранняя и самая понятная реализация.

3.7 PageRank: как Google нашёл порядок в Интернете

3.7.1 Задача и история

К 1996 г. поисковики типа *AltaVista* и *Yahoo!* ранжировали страницы по тому, сколько раз искомое слово встречается в тексте. Это легко обманывалось: достаточно было набить страницу популярными словами белым цветом по белому фону, и она попадала в топ выдачи.

Лоуренс Пейдж и Сергей Брин, тогда аспиранты Стэнфорда, предложили другой принцип ранжирования: *страница важна, если на неё ссылаются другие важные страницы.*

3.7.2 Граф ссылок как матрица

Обозначим N — число всех страниц Интернета (на 1998 г. — около $2{,}4 \cdot 10^8$, сегодня — $\sim 10^{11}$). Построим матрицу $P \in \mathbb{R}^{N \times N}$:

$$P_{ij} = \begin{cases} \dfrac{1}{c_j}, & \text{если со страницы } j \text{ есть ссылка на } i, \\ 0, & \text{иначе,} \end{cases}$$

где c_j — *полное число исходящих ссылок* со страницы j. Содержательно: P_{ij} — это вероятность того, что пользователь, сидящий на странице j и случайно ткнувший на одну из её ссылок, окажется на странице i. Поэтому P называется матрицей переходов **случайного блуждания** по графу Интернета.

3.7.3 Стационарное распределение = собственный вектор

Пусть $r \in \mathbb{R}^N$ — вектор «важностей»: r_i — степень важности страницы i. Принцип Пейджа–Брина формализуется системой:

$$r = Pr. \tag{3.7}$$

То есть важность каждой страницы равна сумме важностей тех страниц, которые на неё ссылаются, делённой на число их исходящих ссылок.

Уравнение (3.7) говорит: r — это *собственный вектор* матрицы P с собственным значением 1. И в этом — вся суть PageRank.

Теорема 3.4. (Перрон–Фробениус для PageRank)

Если все элементы матрицы P положительны (или хотя бы граф ссылок сильно связен и непериодичен), то собственное значение 1 у P существует, оно *простое* и наибольшее по модулю, а соответствующий собственный вектор r единственен и имеет положительные компоненты. Этот r называется **стационарным распределением** случайного блуждания.

(Эта теорема — Оскар Перрон, 1907 г., и Георг Фробениус, 1912 г. — исторически появилась задолго до Интернета, в контексте моделей популяций.)

3.7.4 Демпфирующий множитель и формула Брина–Пейджа

В реальной матрице Интернета у некоторых страниц нет исходящих ссылок (*dangling pages*), а в других группах — циклы. Чтобы гарантировать применимость теоремы Перрона–Фробениуса, Пейдж и Брин ввели **демпфирующий множитель** $\beta \in (0, 1)$ (обычно $\beta = 0{,}85$):

$$\boxed{r = \beta P r + \frac{1-\beta}{N}\mathbf{1}.} \tag{3.8}$$

Здесь $\mathbf{1} = (1, 1, \ldots, 1)^\top$. Содержательно: с вероятностью $\beta \approx 0{,}85$ случайный сёрфер кликает по ссылке, а с вероятностью $1 - \beta \approx 0{,}15$ — телепортируется на случайную страницу Интернета.

3.7.5 Как находить r: степенной метод

Прямо решать систему $(I - \beta P)r = \frac{1-\beta}{N}\mathbf{1}$ с $N \sim 10^{11}$ невозможно. Но и не надо: вектор r можно получить простой итерацией.

Алгоритм

Степенной метод для PageRank.

1. Инициализация: $r^{(0)} \leftarrow \frac{1}{N}\mathbf{1}$.
2. Итерация: $r^{(t+1)} \leftarrow \beta\, P\, r^{(t)} + \dfrac{1-\beta}{N}\,\mathbf{1}$.
3. Остановка: $\left\| r^{(t+1)} - r^{(t)} \right\| < \varepsilon$.

Сходимость гарантируется теоремой Перрона–Фробениуса: $\left\| r^{(t)} - r \right\| \le C \cdot \beta^{t}$, то есть погрешность убывает геометрически со знаменателем $\beta = 0{,}85$. На практике хватает $50 \div 100$ итераций.

3.7.6 Игрушечный пример: 6 страниц

Возьмём маленький «Интернет» из шести страниц (рис. 3.11) и посчитаем для него PageRank.

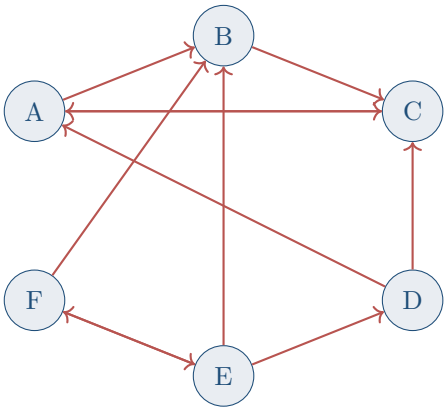


Рис. 3.11. Игрушечный «Интернет» из 6 страниц со ссылками. Видно, например, что на страницу C ссылаются три другие, а на F — только одна; будет ли C лидером по PageRank?

```python
import numpy as np

# Adjacency list -> column-stochastic transition matrix P
links = {'A': ['B','C'], 'B': ['C'], 'C': ['A'],
         'D': ['C','A'], 'E': ['D','B','F'], 'F': ['E','B']}
pages = list(links); n = len(pages); idx = {p: i for i, p in enumerate(pages)}

P = np.zeros((n, n))
for j, p in enumerate(pages):
    out = links[p]
    for q in out:
        P[idx[q], j] = 1.0 / len(out)

# PageRank via power iteration
beta, eps = 0.85, 1e-10
r = np.ones(n) / n
for t in range(200):
    r_new = beta * P @ r + (1 - beta) / n
    if np.linalg.norm(r_new - r, 1) < eps: break
    r = r_new

for p, ri in sorted(zip(pages, r), key=lambda t: -t[1]):
    print(f'{p}: {ri:.4f}')
```

Выдача программы:

$$\text{C: } 0{,}295, \quad \text{A: } 0{,}252, \quad \text{B: } 0{,}175, \quad \text{E: } 0{,}098, \quad \text{F: } 0{,}092, \quad \text{D: } 0{,}087.$$

Лидер — `C`, хотя на неё формально ссылаются столько же страниц, сколько на `A`: PageRank учёл, что одна из ссылок на `C` приходит с уже-важной страницы `B`.

> Алгоритм PageRank был опубликован в 1998 г. в студенческой работе *Sergey Brin, Larry Page*, «*The PageRank Citation Ranking: Bringing Order to the Web*». В том же году Брин и Пейдж основали Google. К 2004 г. компания вышла на IPO, а к 2010-м стала одной из крупнейших в мире. Сам Стэнфорд получил $\sim$ 336 миллионов долларов от лицензирования патента — который изначально просто описывал итерационный метод для собственного вектора.
> Идея использовать собственный вектор графа для измерения «важности» позднее была обобщена и применена в наукометрии (*Eigenfactor* для ранжирования научных журналов), в рекомендательных системах, а также в современных нейросетях для графов (*Graph Neural Networks*).

3.8 Когда нейросеть устойчива: оценки Липшицевости*

3.8.1 Сюжет: когда панда становится гиббоном

В 2014 г. Иэн Гудфеллоу и соавторы опубликовали ставшую знаменитой картинку (рис. 3.12): обычная фотография панды, которую современная нейросеть уверенно опознаёт как «panda» с вероятностью 57,7%. К каждому пикселю прибавлена крошечная, визуально незаметная человеку поправка. На обновлённой картинке сеть теперь уверенно (с вероятностью 99,3%) считает, что видит *гиббона*.

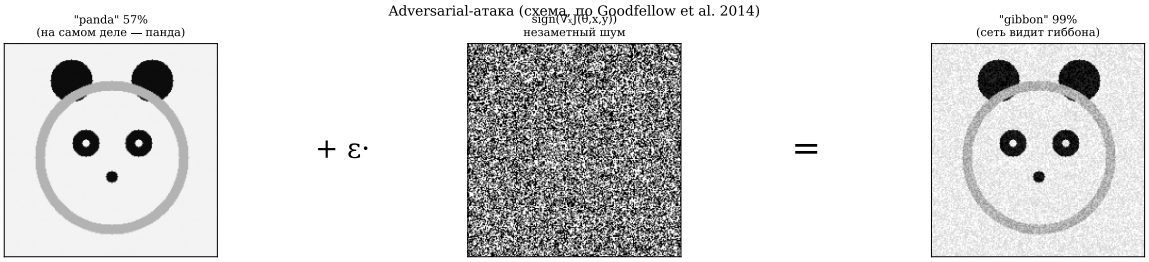


Рис. 3.12. Adversarial-атака на нейросеть (Goodfellow et al., 2014). Слева — оригинал, классифицирован как «panda» с уверенностью 57,7%. В центре — почти невидимое возмущение, масштабированное для наглядности. Справа — атакованный пример, классифицируется как «gibbon» с уверенностью 99,3%.

Этот эффект называется **adversarial-атакой** и долгое время считался экзотикой. Сейчас понятно: причина в простом числе, которое сопровождает любую функцию — её *константе Липшица*.

3.8.2 Константа Липшица: формальное определение

Определение 3.5. Константа Липшица функции

Функция $f\colon \mathbb{R}^n \to \mathbb{R}^m$ называется **Липшицевой** с константой $L > 0$, если для любых $x, y \in \mathbb{R}^n$ выполнено

$$\|f(x) - f(y)\| \;\le\; L\,\|x - y\|\,. \tag{3.9}$$

Наименьшее L, для которого (3.9) выполнено, называется *константой Липшица* функ-

ции f и обозначается $\mathrm{Lip}(f)$.

Содержательно $\mathrm{Lip}(f)$ показывает, во сколько раз может *усилиться* входное возмущение на выходе функции. Если $\mathrm{Lip}(f) = L$, то малое возмущение δ на входе может вызвать возмущение до $L \cdot \|\delta\|$ на выходе.

Пример. Для линейного отображения $f(x) = Wx$ имеем $f(x) - f(y) = W(x - y)$, откуда $\|f(x) - f(y)\| \le \|W\|_2 \cdot \|x - y\|$, и константа Липшица — это в точности спектральная норма матрицы W:

$$\mathrm{Lip}(Wx) \;=\; \|W\|_2 \;=\; \sigma_1(W). \tag{3.10}$$

Здесь $\sigma_1(W)$ — наибольшее сингулярное число матрицы W. Вот *здесь* в нашу историю входит SVD из § 3.3.

3.8.3 Композиция: Липшицева константа цепочки

Теорема 3.6. (Композиция Липшицевых функций)

Если f и g — Липшицевы с константами L_f и L_g, то их композиция $f \circ g$ — тоже Липшицева, причём

$$\mathrm{Lip}(f \circ g) \;\le\; \mathrm{Lip}(f) \cdot \mathrm{Lip}(g). \tag{3.11}$$

Доказательство. $\|f(g(x)) - f(g(y))\| \le L_f \, \|g(x) - g(y)\| \le L_f L_g \, \|x - y\|$. □ □

3.8.4 Применение к нейросети

Возьмём типичную нейросеть глубины K:

$$f(x) \;=\; \sigma\big(W_K \, \sigma(W_{K-1} \, \sigma(\cdots \sigma(W_1 x)))\big),$$

где W_i — матрицы весов, σ — поэлементная активация (ReLU, сигмоида, и т.п.). Для большинства активаций $\mathrm{Lip}(\sigma) \le 1$ (для ReLU и сигмоиды это равно 1). По теореме 3.6:

$$\boxed{\mathrm{Lip}(f) \;\le\; \prod_{i=1}^{K} \|W_i\|_2} \tag{3.12}$$

Это и есть тот «лишний рычаг», который объясняет уязвимость нейросетей. Если $\|W_i\|_2 \approx 2$ для каждого слоя из $K = 20$ слоёв, то возмущение на входе *потенциально* усиливается в $2^{20} \approx 10^6$ раз. Невидимая поправка δ с $\|\delta\| = 10^{-3}$ может вызвать изменение выхода до 10^3 — этого хватит, чтобы перебросить классификатор из одного класса в другой.

3.8.5 Как защищаться: спектральная нормализация

Естественная стратегия: после каждого шага обучения нормализовать матрицу W_i так, чтобы $\|W_i\|_2 = 1$. Это и есть метод **Spectral Normalization** (Миято и соавт., 2018), сделавший обучение генеративных нейросетей (GAN) намного стабильнее.

Технически: на каждом шаге считаем $\sigma_1(W)$ (с помощью одной итерации степенного метода — почти бесплатно!) и заменяем $W \mapsto W / \sigma_1(W)$.

```
def spectral_normalize(W, n_iter=1):
    """One step of power iteration to estimate sigma_1(W) and rescale."""
    u = np.random.randn(W.shape[0])
    for _ in range(n_iter):
        v = W.T @ u; v /= np.linalg.norm(v)
        u = W @ v;   u /= np.linalg.norm(u)
    sigma_1 = u @ W @ v
    return W / sigma_1
```

Это интересно

Связь с устойчивостью в физике. Похожая ситуация хорошо известна в теории динамических систем: если константа Липшица отображения > 1, малое возмущение начального условия экспоненциально нарастает (классический эффект бабочки в погодных моделях). Spectral Normalization можно рассматривать как искусственное гашение этого эффекта.

3.9 QR-алгоритм: как находят *все* собственные числа*

3.9.1 Зачем нужен ещё один метод

В §3.7 мы использовали *степенной метод*, чтобы найти *один* собственный вектор — доминантный, соответствующий наибольшему по модулю собственному числу. А что, если нужны *все* собственные числа сразу — например, чтобы найти главные компоненты k-ранговой аппроксимации, посчитать спектральную норму или определить устойчивость линейной системы $\dot{x} = Ax$ (она устойчива, когда все собственные числа имеют отрицательную действительную часть)?

Здесь и нужен **QR-алгоритм** — одна из жемчужин численных методов. Описан в 1961 году Джоном Фрэнсисом и независимо Верой Кубланской, по сей день остаётся *рабочей лошадкой* библиотек LAPACK и NumPy — именно его вы вызываете каждый раз, когда пишете `numpy.linalg.eig`.

3.9.2 Сам алгоритм: две строчки

Идея до неприличия проста. Возьмём QR-разложение матрицы A: $A = QR$, где Q ортогональная (поворот), R верхнетреугольная. Теперь переставим сомножители:

$$A_{k+1} \;=\; R_k Q_k, \quad \text{где } Q_k R_k = \mathrm{qr}(A_k).$$

И повторим. Всё.

QR-алгоритм

Вход: квадратная матрица $A_0 \in \mathbb{R}^{n\times n}$, число итераций N.
Повторять $k = 0, 1, \ldots, N-1$:

1. Вычислить QR-разложение: $Q_k, R_k \leftarrow \mathrm{qr}(A_k)$.
2. Обновить: $A_{k+1} \leftarrow R_k Q_k$.

Возврат: A_N — (почти) верхнетреугольная; на её диагонали стоят собственные числа исходной A.

3.9.3 Что происходит и почему это работает

Все матрицы A_k *подобны* исходной: $A_{k+1} = Q_k^\top A_k Q_k$. А подобные матрицы имеют одинаковые собственные числа (это базовый факт линейной алгебры). При этом с каждой итерацией поддиагональные элементы A_k уменьшаются в геометрической прогрессии, и в пределе A_k становится верхнетреугольной — значит, на её диагонали как раз и стоят собственные числа.

Теорема 3.7. (Сходимость QR-алгоритма)

Пусть собственные числа A действительны и различны по модулю: $|\lambda_1| > |\lambda_2| > \ldots > |\lambda_n|$. Тогда последовательность A_k, полученная QR-итерациями, сходится к верхне-

треугольной матрице, на диагонали которой стоят $\lambda_1, \lambda_2, \ldots, \lambda_n$ (в порядке убывания модуля), причём поддиагональные элементы убывают как $|\lambda_{i+1}/\lambda_i|^k$.

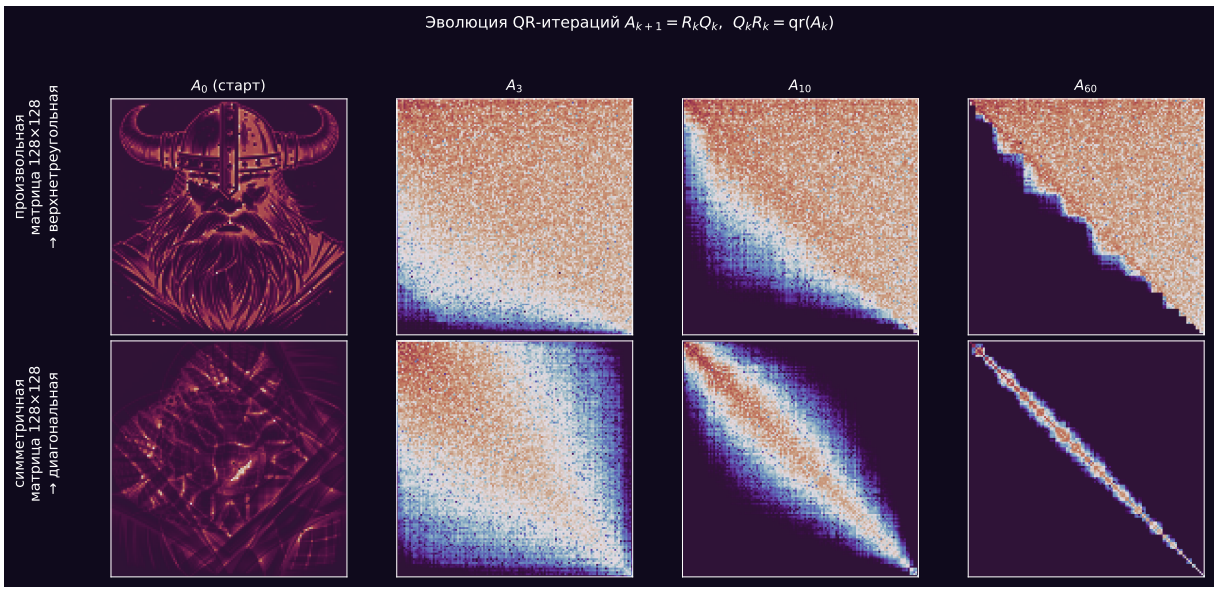


Рис. 3.13. Сходимость QR-алгоритма на 5×5-матрицах. Верхний ряд: произвольная (несимметричная) матрица сходится к верхнетреугольной с собственными числами на диагонали $(4,\ -3,\ 2.5,\ -1.5,\ 1)$. Нижний ряд: симметричная матрица сходится к *диагональной* — это полезное свойство симметрии. Источник: t.me/fminxyz/30.

3.9.4 Симметричный случай: сходимся к диагональной

Если исходная A симметрична, $A^\top = A$, то все промежуточные A_k тоже симметричны, и сходимость идёт не просто к верхнетреугольной, а сразу к *диагональной* матрице. На рис. 3.13 (нижний ряд) видно, что внедиагональные элементы зануляются с обеих сторон одновременно. Это особенно ценно: получив диагональ, мы сразу имеем и собственные числа, и собственные векторы (произведение всех Q_k сходится к матрице собственных векторов).

```python
import numpy as np

def qr_eigvals(A, n_iters=200):
    A = A.astype(float).copy()
    for _ in range(n_iters):
        Q, R = np.linalg.qr(A)
        A = R @ Q
    return np.diag(A)            # собственные числа

# Проверка на случайной симметричной матрице:
np.random.seed(0)
M = np.random.randn(5, 5)
S = (M + M.T) / 2
print(sorted(qr_eigvals(S),                reverse=True))
print(sorted(np.linalg.eigvalsh(S).tolist(), reverse=True))
```

В практических реализациях используют *двухступенчатый* вариант: сначала A приводят к **форме Хессенберга** (почти треугольной — только одна поддиагональ ненулевая) ортогональными преобразованиями Хаусхолдера, а потом запускают QR-итерации со сдвигами (*Wilkinson shift*). Это даёт сходимость за $\mathcal{O}(n)$ итераций вместо $\mathcal{O}(n \log n)$ «наивного» QR, и общая сложность получается $\mathcal{O}(n^3)$ — тот же порядок, что и у SVD.

3.9.5 Связь с SVD

QR и SVD — родственники. Сингулярные числа матрицы A совпадают с квадратными корнями собственных чисел матрицы $A^\top A$. Поэтому один из стандартных способов вычислить SVD — сначала построить $A^\top A$ (она симметрична), применить QR-алгоритм и получить $\sigma_i = \sqrt{\lambda_i}$. На практике, однако, так *не делают*: построение $A^\top A$ возводит в квадрат число обусловленности и теряет точность. Современные алгоритмы SVD (например, *Голуба–Кахана*) применяют QR-итерации напрямую к би-диагональной форме A, обходя $A^\top A$.

Подведение итогов главы

- Матрица — это *универсальный носитель данных* для ИИ: картинки, тексты, графы, веса нейросети.
- **Числа с плавающей точкой** (§3.2) — вычисления на компьютере всегда приближённые с относительной ошибкой $\sim \varepsilon_{\text{маш}} \approx 10^{-16}$. Самая опасная ошибка — *катастрофическое сокращение* при вычитании близких чисел; её часто можно устранить, переписав формулу алгебраически эквивалентным способом.
- **Сингулярное разложение** $A = U\Sigma V^\top$ (§3.3) представляет любую матрицу как «поворот → растяжение по осям → поворот». Сингулярные числа σ_i — это коэффициенты растяжения.
- **Теорема Эккарта–Янга** (§3.4): $A_k = \sum_{i \le k} \sigma_i u_i v_i^\top$ — лучшее приближение ранга k матрицы A. На этой теореме стоит сжатие изображений, метод eigenfaces и современный LoRA для дообучения больших языковых моделей.
- **Eigenfaces / eigencats** (§3.5–3.6): реальные данные (лица, котики, цифры) живут в очень узком подпространстве своего номинального многомерного пространства. Найти это подпространство = применить SVD к выборке.
- **PageRank** (§3.7): «важность» страницы Интернета = компонента собственного вектора матрицы переходов случайного блуждания. Алгоритм находит его простой итерацией; теорема Перрона–Фробениуса гарантирует сходимость.
- **Константа Липшица** нейросети (§3.8) растёт мультипликативно от слоя к слою; именно поэтому глубокие сети уязвимы к adversarial-атакам. Спектральная нормализация ограничивает $\|W_i\|_2 = 1$ и стабилизирует обучение.

Что почитать дальше

- *Е. Е. Тыртышников.* Методы численного анализа. М.: Академия, 2007. — классический российский учебник по вычислительной линейной алгебре университетского уровня.

- *G. Strang.* Linear Algebra and Learning from Data. Wellesley-Cambridge Press, 2019. — свежая книга от автора знаменитого MIT-курса 18.06, написанная с прицелом на data science.
- *L. N. Trefethen, D. Bau.* Numerical Linear Algebra. SIAM, 1997. — классика, с очень понятным изложением SVD и итерационных методов.
- *Видеокурс «Numerical Linear Algebra»*, И. В. Оселедец, Сколтех (github.com/MerkulovDaniil/nla360). Лекции, семинары, ноутбуки — из которых, в частности, и взяты eigencats для этой главы.
- *S. Brin, L. Page.* The Anatomy of a Large-Scale Hypertextual Web Search Engine. WWW Conference, 1998. — оригинальная статья про PageRank, читается легко.
- *I. Goodfellow et al.* Explaining and Harnessing Adversarial Examples. ICLR, 2015. — статья про панду-гиббона, из которой взят рис. 3.12.

Большой итоговый проект

Постройте свою рекомендательную систему для фильмов.

Возьмите датасет MovieLens-100K (открытый, 100 000 оценок, ~ 1000 пользователей $\times$ ~ 1700 фильмов). Постройте матрицу «пользователь–фильм» R, где $R_{ij} \in \{1, \dots, 5\}$ — оценка, поставленная пользователем i фильму j, а 0 — «не смотрел».

1. Постройте малоранговое приближение $R \approx U\Sigma V^\top$ с $k = 20$ компонентами через SVD.
2. Объясните содержательно, что такое строки U (профили пользователей) и столбцы V (профили фильмов) в этом приближении. Сколько чисел теперь хранит модель вместо $1000 \cdot 1700$?
3. Для нескольких реальных пользователей выпишите топ-5 фильмов, которые рекомендация модели предлагает посмотреть. Совпадают ли они с любимыми жанрами пользователя?
4. Найдите два «противоположных» фильма с наибольшим расстоянием между их столбцами V и подумайте, можно ли увидеть содержательный смысл в этой разделяющей оси (боевик vs. мелодрама? старое vs. новое?).
5. ($\star$) Замените SVD на алгоритм PageRank, построив граф «фильмы, которые часто оценивают одинаково». Получится ли тот же топ?

Задачи для самостоятельной работы

1. Сложите в Python 10 000 000 копий числа 0,1. Сколько должно получиться? Сколько получается на самом деле? Почему?
2. Вычислите $f(x) = \sqrt{x+1} - \sqrt{x}$ при $x = 10^{16}$ напрямую и через формулу $f(x) = \frac{1}{\sqrt{x+1}+\sqrt{x}}$. Сравните ответы. Какой устойчив, почему?
3. Найдите вручную SVD матрицы $A = \left(\begin{smallmatrix} 3 & 0 \\ 0 & 4 \end{smallmatrix}\right)$ и $A = \left(\begin{smallmatrix} 0 & 1 \\ -1 & 0 \end{smallmatrix}\right)$. Что в обоих случаях геометрически делают U, Σ, V?
4. Возьмите свою фотографию 300×300 в градациях серого. Сделайте сжатие через SVD с рангами $k = 5, 20, 50, 100$. При каком k вы перестаёте узнавать самого себя?
5. Для игрушечного графа из рис. 3.11 прокрутите 5 итераций степенного метода *вручную*, начав с $r^{(0)} = (\frac{1}{6}, \dots, \frac{1}{6})^\top$. Совпали ли ваши значения с программными после 5 шагов? а после 20?
6. * Покажите, что для функции $\sigma(x) = \max(0, x)$ (ReLU) константа Липшица равна

1. Тот же вопрос для сигмоиды $\sigma(x) = 1/(1 + e^{-x})$: чему равна $\mathrm{Lip}(\sigma)$ и в какой точке производная максимальна?

7. * Найдите спектральную норму матрицы $W = \left(\begin{smallmatrix} 1 & 2 \\ 2 & 1 \end{smallmatrix}\right)$ двумя способами: (а) через определение $\|W\|_2 = \max_{\|x\|=1} \|Wx\|$ (с использованием собственных значений $W^\top W$); (б) одной итерацией степенного метода (с любым стартом). Совпали ли ответы?

Часть III

Анализ данных и искусственный интеллект

Глава 4

Введение в анализ данных и искусственный интеллект

В предыдущих главах мы учились находить корни уравнений и минимумы функций. На вид это два сюжета из вычислительной математики — однако именно они окажутся главными «инструментами в кармане», когда мы будем извлекать смысл из данных.

Эта глава вводит *методологию анализа данных*. Мы пройдём путь от простейшей задачи — определить долю голосов за кандидата по результатам небольшого опроса — до моделей, на которых работают современные большие языковые модели и системы распознавания изображений. Один и тот же приём — **принцип максимума правдоподобия** — окажется работающим и в первом, и в последнем сюжетах. Сменяются только модели: монета — нормальный шум — линейная регрессия — нейронная сеть.

Современный анализ данных уверенно стоит на трёх китах:

- **вероятностном** — он превращает расплывчатое «извлечь информацию из данных» в строгую задачу: найти параметр модели, при котором наблюдаемые данные наиболее правдоподобны;
- **оптимизационном** — он указывает, как именно найти этот параметр: чаще всего это сводится к минимизации некоторого функционала, и тут пригождаются методы из главы 2;
- **линейно-алгебраическом** — он даёт язык, на котором записаны и сами данные (огромные таблицы — матрицы), и преобразования над ними (умножение матриц, малоранговые приближения, разложения).

В этой главе мы поочерёдно увидим все три кита в действии.

4.1 Простейшие примеры задач анализа данных. Принцип максимума правдоподобия

4.1.1 Опрос на втором туре выборов

На втором туре президентских выборов борьба идёт между двумя кандидатами, — назовём их условно A и B. До дня голосования социологическая служба проводит опрос: выбирает случайным образом n избирателей и спрашивает каждого, за кого тот собирается голосовать. Допустим, k человек из n ответили «за A».

Какую долю голосов в итоге наберёт кандидат A во всей стране?

Вероятностная модель. Предположим, что:

- истинная (нам неизвестная!) доля сторонников A среди всех избирателей равна $p \in (0, 1)$;
- опрашиваемые отбираются *независимо* друг от друга;
- каждый отвечает честно — то есть с вероятностью p говорит «за A», с вероятностью $1 - p$ говорит «за B».

Тогда ответ i-го опрошенного — это случайная величина X_i, принимающая значение 1 (за A) с вероятностью p и значение 0 (за B) с вероятностью $1-p$. Такие величины называют *бернуллиевскими*, а описанную модель — *схемой испытаний Бернулли*.

Определение 4.1. Схема испытаний Бернулли

Эксперимент, в котором n раз независимо повторяется испытание с двумя исходами («успех» с вероятностью p и «неудача» с вероятностью $1-p$), называется **схемой испытаний Бернулли**. Число успехов $S_n = X_1 + \cdots + X_n$ имеет *биномиальное распределение*:

$$\Pr(S_n = k) = \binom{n}{k} p^k (1-p)^{n-k}, \qquad k = 0, 1, \ldots, n. \tag{4.1}$$

Наблюдаемые данные — это вектор ответов $X = (X_1, \ldots, X_n)$ или, эквивалентно, число «единиц» в нём: $S_n = X_1 + \cdots + X_n = k$. Параметр p нам неизвестен — это и есть **то, что мы хотим оценить** по данным.

Идея метода максимума правдоподобия. Запишем вероятность того, что данные оказались именно такими, как мы их наблюдаем, — как функцию неизвестного параметра p:

$$L(p) \;=\; \Pr(X_1 = x_1, \ldots, X_n = x_n \mid p) \;=\; \prod_{i=1}^{n} p^{x_i}(1-p)^{1-x_i} \;=\; p^k (1-p)^{n-k}, \tag{4.2}$$

где $k = \sum_i x_i$ — наблюдаемое число «единиц». Функция $L(p)$ называется **функцией правдоподобия**. Метод максимума правдоподобия (*maximum likelihood estimation*, сокращённо **ОМП**) состоит в выборе того $\hat{p}$, при котором $L(p)$ достигает максимума:

$$\hat{p} \;=\; \arg\max_{p \in (0,1)} L(p).$$

Идея, на которой держится метод, очень житейская: *из всех возможных значений параметра выбираем то, при котором наблюдаемые данные были бы наиболее правдоподобными.* Если выпало 600 единиц из 1000, было бы странно объявить $p = 0{,}1$ — ведь при таком p событие $S_n = 600$ почти невозможно. Здравый смысл подсказывает: правдоподобнее всего гипотеза $p \approx 0{,}6$. Метод максимума правдоподобия и есть формализация этого здравого смысла.

Вычисление оценки. Удобнее искать максимум *логарифма* правдоподобия — это сводит произведение к сумме, не меняя точку максимума (логарифм монотонен):

$$\ln L(p) = k \,\ln p + (n-k)\,\ln(1-p). \tag{4.3}$$

Дифференцируем по p и приравниваем нулю (принцип Ферма):

$$\frac{d}{dp}\ln L(p) \;=\; \frac{k}{p} - \frac{n-k}{1-p} \;=\; 0 \quad\Longleftrightarrow\quad k(1-p) = (n-k)p \quad\Longleftrightarrow\quad p = \frac{k}{n}.$$

Чтобы убедиться, что это именно максимум, посмотрим на вторую производную: $\frac{d^2}{dp^2}\ln L(p) = -k/p^2 - (n-k)/(1-p)^2 < 0$ для всех $p \in (0,1)$. Значит, $\ln L$ — строго вогнутая функция, и найденная точка — её единственный максимум.

Теорема 4.2. (ОМП в схеме Бернулли)

Оценкой максимума правдоподобия параметра p в схеме испытаний Бернулли по выборке $X_1, \dots, X_n$ является **выборочное среднее**:

$$\hat{p} = \frac{S_n}{n} = \overline{X} = \frac{X_1 + \dots + X_n}{n}. \tag{4.4}$$

Это очень житейский ответ: «доля сторонников A оценивается долей ответивших “за A”». Тем интереснее, что этот же ответ выводится из общего, абсолютно формального принципа — ОМП. На рис. 4.1 видно, что чем больше выборка, тем «острее» становится максимум функции правдоподобия — то есть тем точнее данные «локализуют» неизвестный параметр.

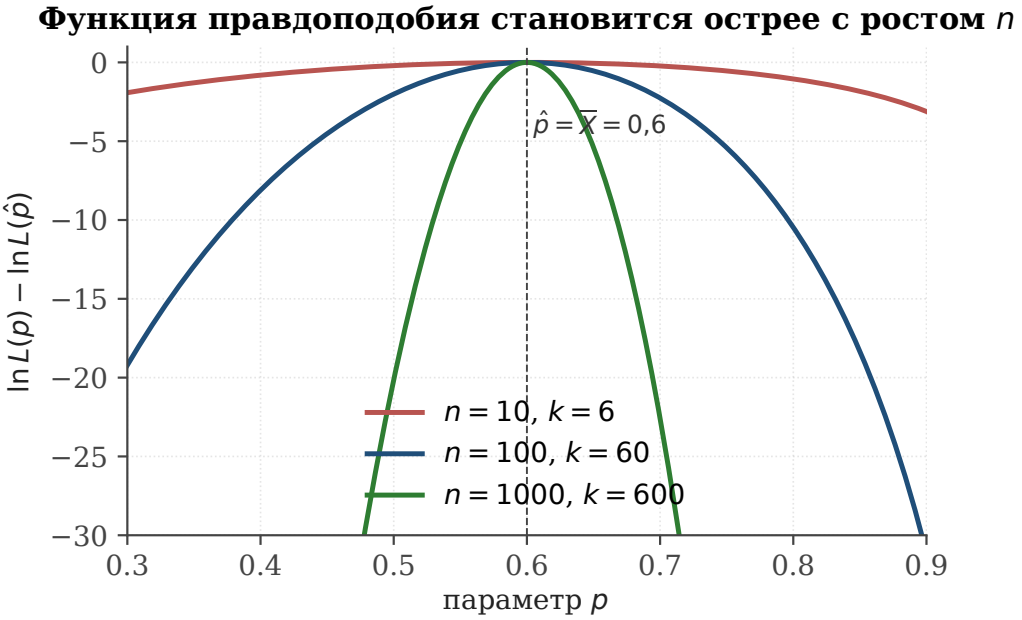


Рис. 4.1. Логарифм функции правдоподобия $\ln L(p) - \ln L(\hat{p})$ для трёх выборок одинаковой пропорции $k/n = 0{,}6$, но разного размера ($n = 10, 100, 1000$). С ростом n функция концентрируется всё плотнее вокруг истинного значения; ширина характерной зоны падает как $1/\sqrt{n}$.

4.1.2 Почему ОМП хорош: теоретические результаты Фишера и Ле Кама

Естественный вопрос: *является ли* $\hat{p} = \overline{X}$ *лучшей возможной оценкой* — или, может, существует ещё более точный способ извлечь p из данных?

Ответ на этот вопрос получили в первой половине XX века Р. Фишер и затем Л. Ле Кам. Изложим главные результаты на популярном уровне, без полных доказательств (они выходят за рамки школьной программы).

Рональд Эймлер Фишер (1890–1962) в работе 1922 г. «On the mathematical foundations of theoretical statistics» ввёл понятие функции правдоподобия и предложил метод её максимизации как универсальный способ оценивания. Он же ввёл термины «оценка», «параметр», «статистика» и доказал, что метод максимума правдоподобия обладает *асимптотической эффективностью* — его оценки имеют наименьшую возможную (по неравенству информации) дисперсию при больших размерах выборки.
Люсьен Ле Кам (1924–2000) в серии работ 1950-х—1970-х гг. построил общую тео-

рию *асимптотически нормальных* статистических экспериментов и доказал, что оценка максимума правдоподобия является, в очень широком смысле, *оптимальной*: для гладких параметрических семейств никакая другая разумная оценка не может быть точнее.
Аналог теоремы об асимптотической эффективности в более общем бесконечномерном случае независимо изучал советский статистик **Илья Александрович Ибрагимов** (р. 1932) с соавторами (Ибрагимов, Хасьминский, ЛОМИ, 1970-е).

Главную мысль можно сформулировать так:

При $n \to \infty$ оценка максимума правдоподобия $\hat{p}_n$ для гладкого параметрического семейства распределений:

1. *состоятельна*: $\hat{p}_n \to p$ при $n \to \infty$ (в смысле сходимости по вероятности);
2. *асимптотически нормальна*: распределение нормированной разности $\sqrt{n}\,(\hat{p}_n - p)$ при $n \to \infty$ стремится к нормальному распределению с нулевым средним и определённой дисперсией;
3. *асимптотически эффективна*: эта предельная дисперсия — **наименьшая** среди всех «разумных» (несмещённых, регулярных) оценок. Соответственно, доверительный интервал, построенный по ОМП, оказывается *кратчайшим* среди всех таких интервалов, то есть наиболее точно локализует неизвестный параметр.

Иначе говоря, в очень широком классе задач ОМП — это *асимптотически наилучшее* из всего, на что можно надеяться при росте n. Этот результат лежит в основе огромной части современной математической статистики и машинного обучения.

4.1.3 Доверительные интервалы: Чебышёв и центральная предельная теорема

Получив точечную оценку $\hat{p} = k/n$, мы хотим понимать, насколько ей можно доверять. Социологи говорят: «46 % за кандидата A с погрешностью ± 3 %». Откуда берётся это ± 3 %?

Подход 1: неравенство Чебышёва

Поскольку $S_n = X_1 + \cdots + X_n$ — сумма независимых одинаково распределённых случайных величин с математическим ожиданием p и дисперсией $p(1-p)$, для $\hat{p}_n = S_n/n$:

$$\mathbb{E}[\hat{p}_n] = p, \qquad \mathbb{D}[\hat{p}_n] = \frac{p(1-p)}{n}.$$

Применим неравенство Чебышёва: для любого $\varepsilon > 0$

$$\Pr\bigl(|\hat{p}_n - p| \geq \varepsilon\bigr) \;\leq\; \frac{\mathbb{D}[\hat{p}_n]}{\varepsilon^2} \;=\; \frac{p(1-p)}{n\,\varepsilon^2} \;\leq\; \frac{1}{4n\,\varepsilon^2}, \tag{4.5}$$

где мы воспользовались тем, что $p(1-p) \leq 1/4$ для всех $p \in [0,1]$.

Если потребовать, чтобы вероятность ошибки не превышала α (например, $\alpha = 0{,}05$), достаточно взять

$$\varepsilon \;=\; \frac{1}{2\sqrt{\alpha n}}.$$

Тогда с вероятностью $1 - \alpha$

$$\hat{p}_n - \frac{1}{2\sqrt{\alpha n}} \;\leq\; p \;\leq\; \hat{p}_n + \frac{1}{2\sqrt{\alpha n}}. \tag{4.6}$$

Подход 2: центральная предельная теорема

Неравенство Чебышёва справедливо при минимальных предположениях и потому *очень грубое*. Центральная предельная теорема (ЦПТ) — а её мы примем сейчас без доказательства — утверждает, что распределение нормированной суммы независимых одинаково распределённых случайных величин с конечной дисперсией стремится к нормальному.

Теорема 4.3. (Центральная предельная теорема (ЦПТ))

Пусть $X_1, X_2, \dots$ — независимые одинаково распределённые случайные величины с математическим ожиданием μ и конечной дисперсией $\sigma^2 > 0$, $S_n = X_1 + \cdots + X_n$. Тогда при $n \to \infty$

$$\Pr\left(\frac{S_n - n\mu}{\sigma\sqrt{n}} \le x\right) \longrightarrow \Phi(x) \stackrel{\text{def}}{=} \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{x} e^{-t^2/2}\, dt$$

(функция стандартного нормального распределения).

Применим ЦПТ к нашей задаче: $\mu = p$, $\sigma^2 = p(1-p)$, и при больших n

$$\frac{\hat{p}_n - p}{\sqrt{p(1-p)/n}} \approx \mathcal{N}(0,1).$$

Если $z_{1-\alpha/2}$ — квантиль уровня $1-\alpha/2$ стандартного нормального распределения, то с вероятностью $1-\alpha$

$$|\hat{p}_n - p| \le z_{1-\alpha/2} \cdot \sqrt{p(1-p)/n} \le \frac{z_{1-\alpha/2}}{2\sqrt{n}}.$$

Получаем **доверительный интервал по ЦПТ**:

$$\hat{p}_n - \frac{z_{1-\alpha/2}}{2\sqrt{n}} \le p \le \hat{p}_n + \frac{z_{1-\alpha/2}}{2\sqrt{n}}. \tag{4.7}$$

При $\alpha = 0{,}05$ имеем $z_{0{,}975} \approx 1{,}96$.

Сравнение двух подходов

Сравним полуширины интервалов (4.6) и (4.7) при $\alpha = 0{,}05$:

$$\varepsilon_{\text{Чеб.}} = \frac{1}{2\sqrt{0{,}05\, n}} = \frac{2{,}236}{2\sqrt{n}} = \frac{1{,}118}{\sqrt{n}}, \qquad \varepsilon_{\text{ЦПТ}} = \frac{1{,}96}{2\sqrt{n}} = \frac{0{,}98}{\sqrt{n}}.$$

Отношение $\varepsilon_{\text{Чеб.}}/\varepsilon_{\text{ЦПТ}} \approx 2{,}28$ — интервал Чебышёва шире более чем вдвое (рис. 4.2). Этим проиллюстрирована важная мысль: *неравенство Чебышёва — это универсальная, но очень консервативная оценка*; знание формы распределения (в нашем случае — асимптотической нормальности) позволяет её существенно улучшить.

Пример 4.4. Числовой пример

Опрошено $n = 1000$ избирателей; за кандидата A высказалось $k = 460$. Точечная оценка — $\hat{p} = 0{,}46$. Полуширина 95 % ДИ:

$$\varepsilon_{\text{Чеб.}} \approx \frac{1{,}118}{\sqrt{1000}} \approx 0{,}0354, \qquad \varepsilon_{\text{ЦПТ}} \approx \frac{0{,}98}{\sqrt{1000}} \approx 0{,}0310.$$

По ЦПТ: с уверенностью 95 % истинная доля сторонников A лежит в интервале $[0{,}429,\ 0{,}491]$, то есть точно меньше половины. *Можно с большой уверенностью утверждать, что кандидат A проиграет* — даже несмотря на то, что точечная оценка 0,46 не сильно ниже 0,5.

Из неравенства Чебышёва мы бы получили интервал $[0{,}425,\ 0{,}495]$ — его правый конец заметно ближе к 0,5, вывод об исходе выборов был бы менее уверенным.

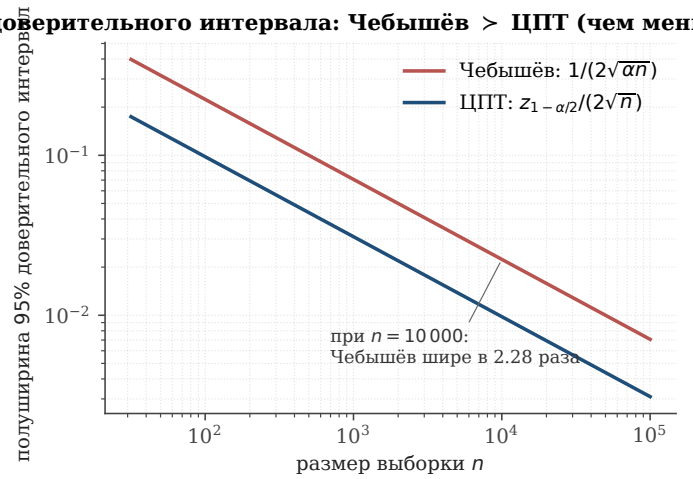


Рис. 4.2. Полуширина 95 % доверительного интервала как функция размера выборки n. Обе кривые убывают как $1/\sqrt{n}$, но ЦПТ-интервал примерно вдвое уже интервала Чебышёва: ЦПТ учитывает форму распределения, тогда как Чебышёв использует лишь дисперсию.

4.1.4 Оценка площади множества: метод Монте-Карло

То, что мы сделали в задаче с выборами, имеет красивое геометрическое обобщение. Заметим: доля «успехов» k/n есть не что иное, как **доля точек выборки, попавших в фиксированное множество**. В нашем случае множество — это «множество всех избирателей, голосующих за A»; но точно так же мы можем оценивать, например, *площадь* криволинейной фигуры.

Пример: оценка числа π

Впишем в единичный квадрат $Q = [0,1]^2$ круг D радиуса $1/2$ с центром в точке $(1/2,\ 1/2)$. Площадь квадрата равна 1; площадь круга равна

$$\mathrm{S}(D) = \pi \cdot (1/2)^2 = \pi/4.$$

Бросим в квадрат N *случайных* точек, равномерно распределённых по Q (для каждой точки независимо генерируем координаты $x, y \sim U(0,1)$). Пусть K — число точек, попавших в круг D. Тогда для каждой брошенной точки вероятность оказаться в D равна

$$p = \Pr\big((x,y) \in D\big) \;=\; \frac{\mathrm{S}(D)}{\mathrm{S}(Q)} \;=\; \pi/4.$$

Мы оказались в *той же* схеме испытаний Бернулли. По теореме 4.2 оценкой p является доля K/N, а оценкой числа π:

$$\boxed{\hat{\pi}_N \;=\; 4 \cdot \frac{K}{N}.} \tag{4.8}$$

Этот приём называется **методом Монте-Карло**.

> Идея использовать случайные точки для приближённого вычисления определённых интегралов восходит к работам Энрико Ферми (конец 1930-х), а в современном виде была развита Станиславом Уламом, Джоном фон Нейманом и Николасом Метрополисом в Лос-Аламосе в 1946–1949 гг. при расчётах прохождения нейтронов сквозь вещество. Название «Монте-Карло» дал Метрополис — в честь известного казино, по аналогии: расчёты с розыгрышем случайных чисел напомнили коллегам игру в рулетку.

Стоит сразу отделить главное от второстепенного. На примере вычисления π метод Монте-Карло выглядит *медленным*: ошибка убывает лишь как $1/\sqrt{N}$, и любая школьная формула численного интегрирования (трапеций, Симпсона) даёт куда быстрее куда более точный ответ. Так зачем же о нём говорить?
Сила метода — не в скорости, а в *универсальности*. Обычные квадратурные формулы плохо переносятся на функции *многих* переменных: чтобы посчитать интеграл по d-мерному кубу с точностью ε, узлов сетки требуется $\sim (1/\varepsilon)^d$ — то самое *«проклятие размерности»*, экспоненциальный рост с числом переменных. Монте-Карло же сохраняет свою скорость $1/\sqrt{N}$ *независимо* от размерности: для интеграла в 100 переменных он работает примерно так же хорошо (или плохо), как для одной.
Именно поэтому Монте-Карло сегодня применяется там, где размерность велика: в физике частиц (расчёты столкновений), квантовой химии (многоэлектронные интегралы), финансовой математике (опционы на много активов), байесовской статистике, а также при обучении нейронных сетей — например, при оценке градиента стохастическим спуском.

Численный эксперимент. На рис. 4.3 показан результат одного запуска при $N = 2000$: красные точки попали *вне* круга, синие — внутри. По формуле (4.8), $\hat{\pi}_{2000} \approx 3{,}154$ — ошибка около $0{,}4\,\%$.

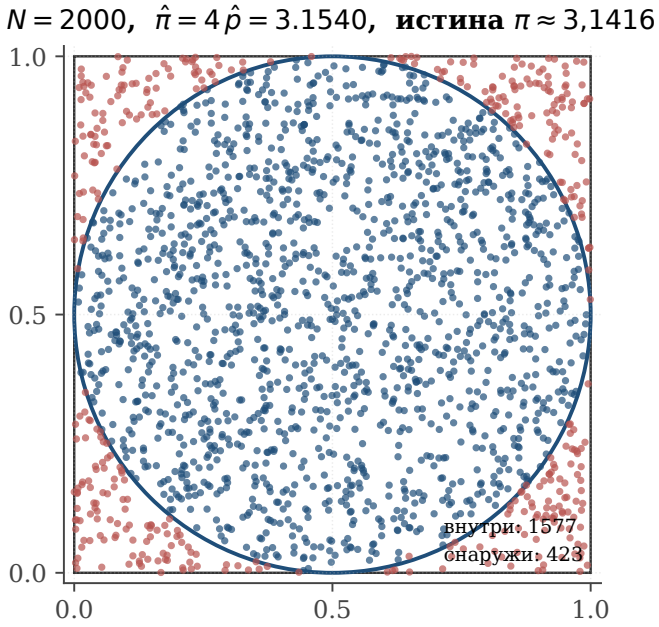


Рис. 4.3. $N = 2000$ случайных точек, равномерно распределённых в единичном квадрате. Точки внутри круга диаметра 1 закрашены синим, снаружи — красным. Доля синих, умноженная на 4, даёт оценку $\pi \approx 3{,}154$.

Графика одной реализации, конечно, мало: при другом значении начального состояния генератора случайных чисел получится немного другой результат. Поэтому естественно изучать *сходимость* — как ведёт себя оценка $\hat{\pi}_N$ при росте N. По формулам выше,

$$\mathbb{E}[\hat{\pi}_N] = \pi, \qquad \mathbb{D}[\hat{\pi}_N] = 16 \cdot \frac{p(1-p)}{N} = \frac{4\pi(4-\pi)}{N} \approx \frac{10{,}79}{N}.$$

Стандартное отклонение оценки убывает как $1/\sqrt{N}$. На рис. 4.4 это видно отчётливо: точка пересекает горизонтальную прямую $y = \pi$ туда-сюда, а $95\,\%$ «коридор» сужается как $1/\sqrt{N}$.

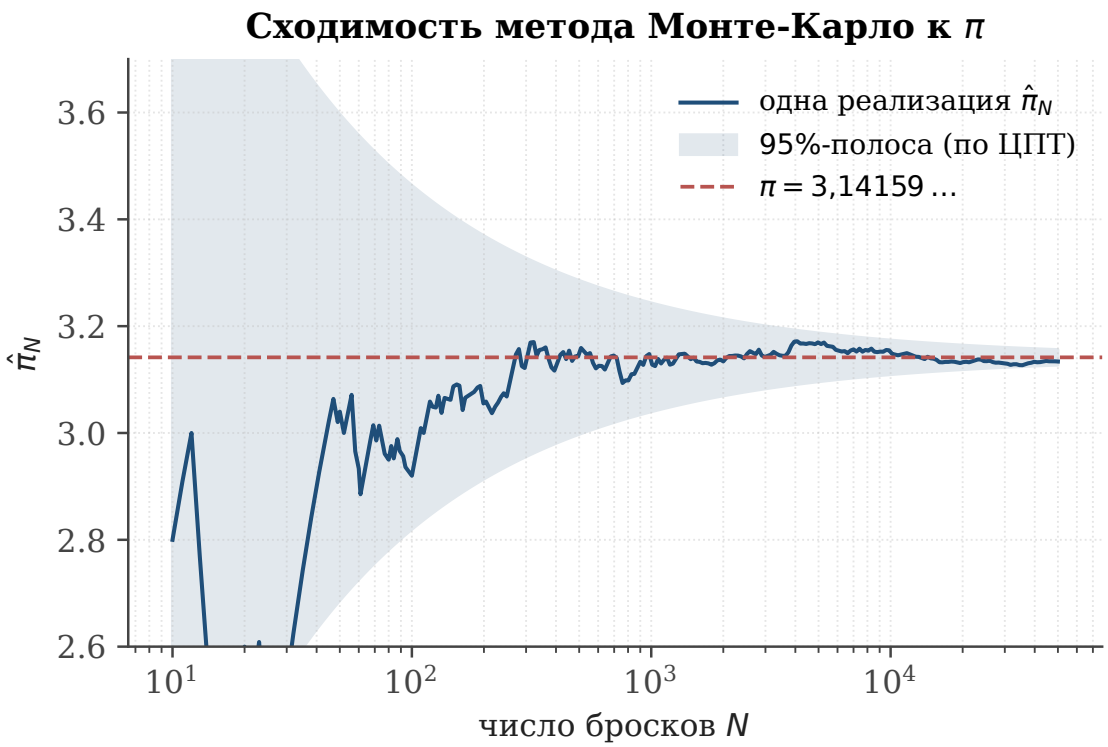


Рис. 4.4. Сходимость метода Монте-Карло для оценки π. Сплошная синяя линия — одна реализация процесса $\hat{\pi}_N$ для N от 10 до $5{\cdot}10^4$; красная пунктирная — точное значение π; заштрихованная полоса — 95 % полоса, предсказываемая ЦПТ. К $N = 50\,000$ ошибка не превосходит $\approx 0{,}015$.

Сколько точек нужно для трёх знаков после запятой? Чтобы получить $\hat{\pi}_N$ с точностью $\varepsilon = 10^{-3}$ с уверенностью 95 %, нужно по (4.7), $z = 1{,}96$: $1{,}96 \cdot \sqrt{10{,}79/N} \le 10^{-3}$, откуда $N \ge (1{,}96)^2 \cdot 10{,}79 \cdot 10^6 \approx 4{,}14 \cdot 10^7$. *Сорок миллионов* точек ради трёх знаков — это и впрямь не самый эффективный способ вычислять π.

Из всех способов, доступных читателю, не выходя за рамки школьной программы, очень быстро сходится формула Мэчина (1706):

$$\frac{\pi}{4} \;=\; 4\arctan\frac{1}{5} - \arctan\frac{1}{239}, \qquad \arctan x \;=\; x - \frac{x^3}{3} + \frac{x^5}{5} - \dots$$

Уже первые ~ 10 членов рядов для $\arctan(1/5)$ и $\arctan(1/239)$ дают *больше десяти* верных знаков числа π. Сравним: метод Монте-Карло потребовал бы для тех же 10 знаков порядка 10^{20} точек — астрономическое число.
Формула Мэчина и её обобщения (формулы Эйлера, Гаусса, Шторма) были основным инструментом ручных вычислений π с XVIII по середину XX века. К началу 1970-х были найдены формулы, дающие десятки знаков на итерацию (Брент, Саламин); современные миллиарды знаков π считаются именно такими методами и алгоритмом БПФ — но это уже совсем другая история.

Итак, метод Монте-Карло — это *плохой* способ считать π, но *отличный* способ считать многомерные объёмы и интегралы. Принципиально важно: он опирается на тот же приём, что и опрос избирателей — выборочное среднее с границами, заданными ЦПТ.

4.1.5 Вторая задача: оценка скалярного параметра при гауссовом шуме

Перейдём ко второй классической задаче, в которой работает тот же принцип максимума правдоподобия. Теперь данные — не «нули и единицы», а вещественные числа с погрешностями.

В 1906 г. английский антрополог **Фрэнсис Гальтон** (1822–1911) посетил сельскохозяйственную выставку в Плимуте, где проходил традиционный конкурс: посетители угадывали вес выставленного быка после того, как его освежевали и взвесили. Гальтон собрал *787* заполненных бланков и обнаружил поразительный факт: *среднее* (а точнее, медиана) догадок участников совпало с истинным весом быка с точностью до фунта (1198 lb).
Этот эпизод — классический пример «мудрости толпы»: несмотря на то что каждый отдельный участник ошибался на десятки фунтов в обе стороны, их *усреднение* оказалось неожиданно точным. С точки зрения математической статистики, перед нами модель «истинное значение плюс шум», и среднее — это и есть оценка максимума правдоподобия при предположении о *гауссовом* шуме. Эту связь мы сейчас установим формально.

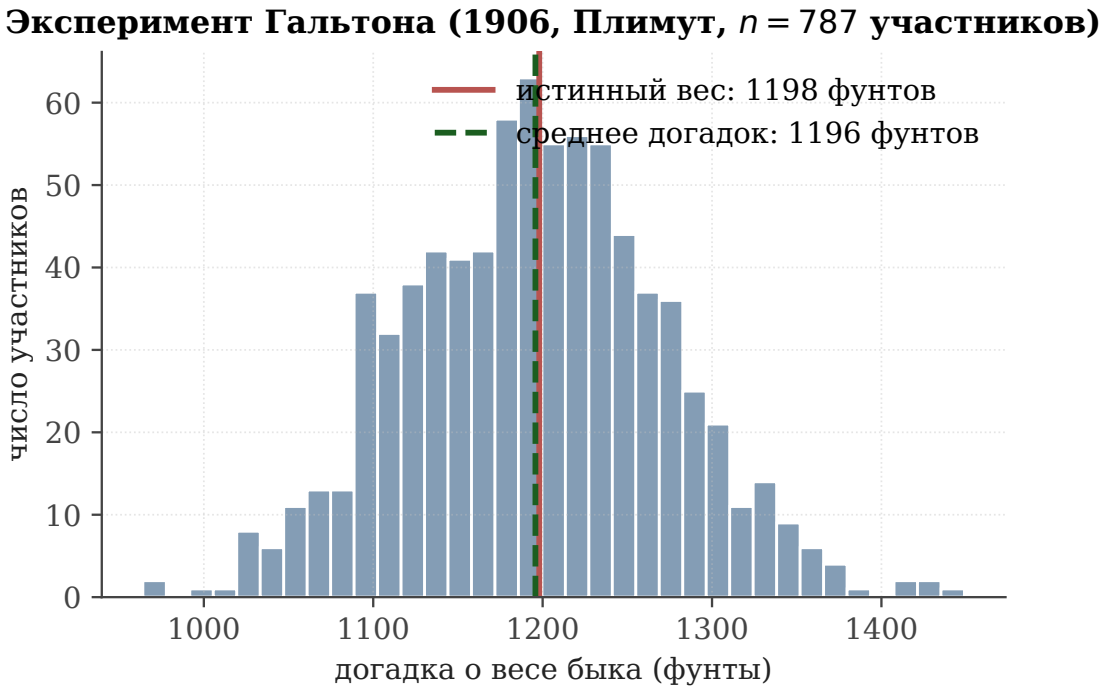


Рис. 4.5. Гистограмма догадок $n = 787$ участников плимутской ярмарки (стилизованная реконструкция распределения, которое описал Гальтон в журнале *Nature*, 1907). Истинный вес быка — 1198 фунтов; среднее догадок — 1196 фунтов. Хотя отдельные оценки ошибаются на сотню фунтов, их усреднение исключительно близко к истине.

Модель. Пусть мы хотим оценить неизвестную величину μ (вес быка; масса звезды; ускорение свободного падения — неважно). Мы располагаем n независимыми измерениями

$$X_i \;=\; \mu + \xi_i, \qquad i = 1, \dots, n, \tag{4.9}$$

где ξ_i — *ошибки измерения*. О них естественно предположить, что:

- они *независимы* (разные измерения не влияют друг на друга),

- *симметричны* относительно нуля (нет систематического сдвига вверх или вниз) и
- имеют *одинаковое* распределение.

Какое именно? Самым «канонически» правильным является **нормальное (гауссовское) распределение** (рис. 4.6).

Определение 4.5. Нормальное (гауссовское) распределение

Случайная величина ξ имеет **нормальное распределение** с параметрами $\mu \in \mathbb{R}$ и $\sigma^2 > 0$, если её плотность вероятности задаётся формулой

$$\varphi_{\mu,\sigma^2}(x) \;=\; \frac{1}{\sigma\sqrt{2\pi}}\,\exp\!\left(-\frac{(x-\mu)^2}{2\sigma^2}\right), \qquad x \in \mathbb{R}. \tag{4.10}$$

Обозначение: $\xi \sim \mathcal{N}(\mu, \sigma^2)$. Параметр μ — математическое ожидание (среднее значение), σ^2 — дисперсия; σ называют *стандартным отклонением*.

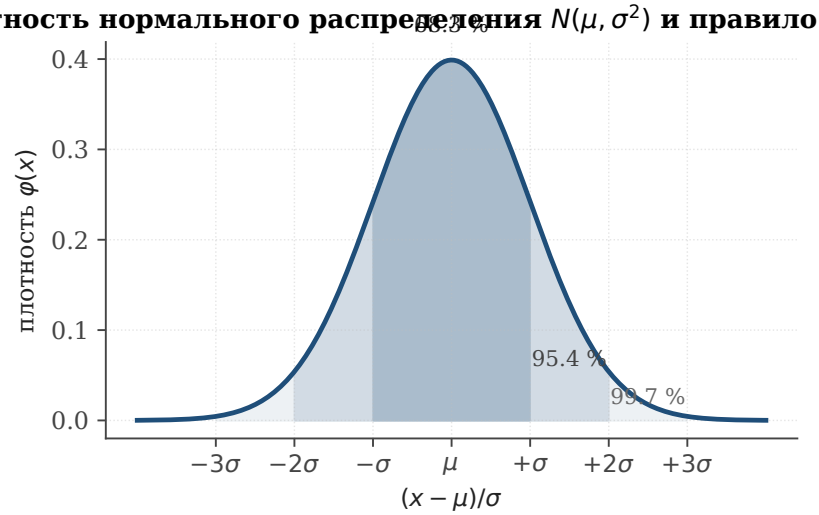


Рис. 4.6. Плотность нормального распределения $\mathcal{N}(\mu, \sigma^2)$. Закрашены 1σ-, 2σ- и 3σ-окрестности среднего: они содержат соответственно 68,3 %, 95,4 % и 99,7 % всей массы распределения. Это знаменитое *правило трёх сигм*.

Принцип максимума правдоподобия для гауссова шума. Согласно модели (4.9), $X_i \sim \mathcal{N}(\mu, \sigma^2)$ независимы. Совместная плотность выборки:

$$L(\mu) = \prod_{i=1}^{n} \varphi_{\mu,\sigma^2}(x_i) = \frac{1}{(\sigma\sqrt{2\pi})^n}\exp\left(-\frac{1}{2\sigma^2}\sum_{i=1}^{n}(x_i-\mu)^2\right), \tag{4.11}$$

$$\ln L(\mu) = -\frac{n}{2}\ln(2\pi\sigma^2) \;-\; \frac{1}{2\sigma^2}\sum_{i=1}^{n}(x_i-\mu)^2. \tag{4.12}$$

Параметр σ^2 можно считать на этом этапе известным или несущественным (мы оцениваем только μ). Тогда из всех слагаемых от μ зависит лишь сумма квадратов:

$$\mu \mapsto S(\mu) \;\stackrel{\text{def}}{=}\; \sum_{i=1}^{n}(x_i-\mu)^2. \tag{4.13}$$

Максимизация правдоподобия эквивалентна минимизации суммы квадратов отклонений. Это краеугольный камень всего, что последует дальше, поэтому отметим его в рамке:

$$\boxed{\text{ОМП при гауссовом шуме} \iff \text{метод наименьших квадратов (МНК)}} \tag{4.14}$$

Минимизация суммы квадратов. Найдём $\hat{\mu} = \arg\min_\mu S(\mu)$. Дифференцируем (4.13) по μ и приравниваем нулю:

$$\frac{dS}{d\mu} = -2\sum_{i=1}^{n}(x_i - \mu) = 0 \quad \Longleftrightarrow \quad \sum_{i=1}^{n} x_i = n\mu.$$

Откуда

$$\boxed{\hat{\mu} = \frac{x_1 + \cdots + x_n}{n} = \overline{X}.} \tag{4.15}$$

Это снова выборочное среднее — теперь уже в задаче с непрерывными данными. Вторая производная $S''(\mu) = 2n > 0$, поэтому это действительно минимум, причём единственный (рис. 4.7).

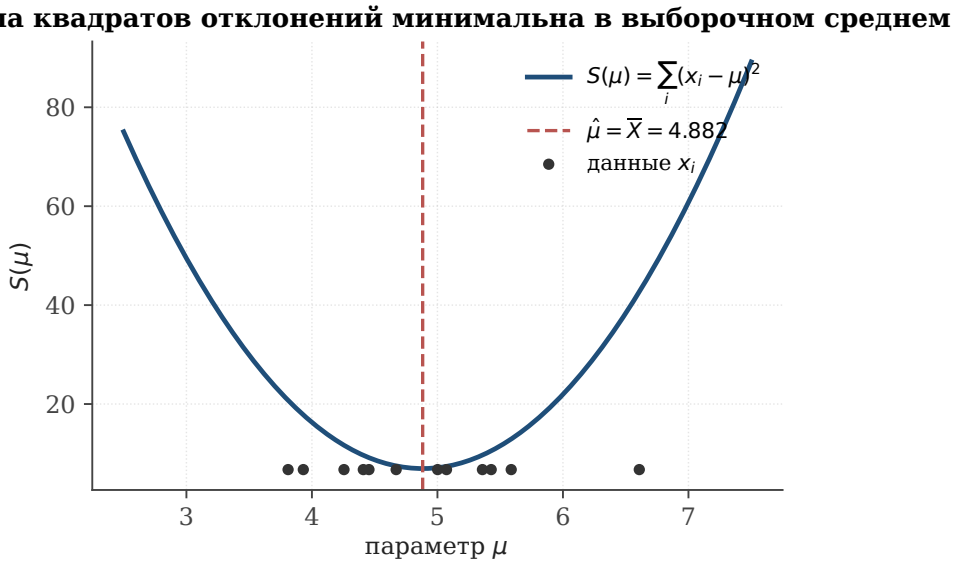


Рис. 4.7. Функция $S(\mu) = \sum(x_i - \mu)^2$ для модельной выборки $n = 12$ точек. Минимум достигается в точке $\hat{\mu} = \overline{X}$ (пунктир). Чёрные точки на нижней оси — сами данные x_i; видно, как $\overline{X}$ «балансирует» их.

4.1.6 Теорема Гаусса: симметричный шум должен быть нормальным

В рассуждении выше мы *приняли* нормальность шума как гипотезу. Поразительный факт — доказанный самим К. Ф. Гауссом, в чью честь и названо это распределение, — состоит в том, что эту гипотезу можно было бы и не принимать: она *вытекает* из требования, чтобы оценкой максимума правдоподобия было именно выборочное среднее.

Теорема 4.6. (Характеризация Гаусса (1809))

Пусть случайные величины $\xi_1, \ldots, \xi_n$ независимы, одинаково распределены и имеют плотность f, симметричную относительно нуля ($f(-x) = f(x)$), дважды дифференцируемую, не обращающуюся в ноль. Если для любого $n \geq 2$ и любой реализации $X_i = \mu + \xi_i$ оценкой максимума правдоподобия параметра μ является выборочное среднее $\overline{X}$, то f есть плотность нормального распределения с нулевым средним.

Идея доказательства. Логарифмическая производная плотности $\rho(x) = -f'(x)/f(x)$ играет роль «силы», возвращающей оценку к точке. Условие максимума ОМП

$$\sum_{i=1}^{n} \rho\big(X_i - \hat{\mu}\big) \;=\; 0$$

должно совпадать с уравнением для среднего $\sum_{i=1}^{n}(X_i - \hat{\mu}) = 0$ при *любых* наблюдениях. Отсюда вытекает, что $\rho(x) = cx$ для некоторой константы $c > 0$, то есть $f'(x)/f(x) = -cx$. Интегрирование даёт $\ln f(x) = -cx^2/2 + \text{const}$, то есть $f(x)$ — плотность нормального распределения с нулевым средним и дисперсией $1/c$. Полный вывод и обсуждение этого свойства Гаусса (*характеризации нормальности через оптимальность среднего*) можно найти в стандартных университетских курсах математической статистики. □

Теорема Гаусса показывает: предположение о нормальности шума и использование среднего арифметического в качестве оценки — это *одно и то же*, две стороны одной медали. Если в практической задаче мы по каким-то соображениям усредняем измерения — мы неявно соглашаемся, что наши ошибки имеют гауссов характер.

Пример 4.7. Применение к эксперименту Гальтона

В стилизованной реконструкции (рис. 4.5) среднее по $n = 787$ ответам составило $\overline{X} \approx 1196$ фунтов при истинном весе $\mu = 1198$ фунтов. Это не означает, что отдельный наблюдатель угадывает вес быка точно: разброс отдельных оценок составляет около $\sigma \approx 75$ фунтов. Однако стандартная погрешность *среднего*

$$\sigma_{\overline{X}} = \frac{\sigma}{\sqrt{n}} \;\approx\; \frac{75}{\sqrt{787}} \approx 2{,}7$$

фунта, и расхождение в 2 фунта вполне укладывается в одну стандартную погрешность. «Мудрость толпы» — следствие закона больших чисел, оформленного как ОМП для гауссовой модели (4.9).

4.1.7 Резюме параграфа

Что мы увидели. Две очень разные задачи — оценка доли голосов в опросе и оценка истинного значения по зашумлённым измерениям — решаются по *одной и той же схеме*:

1. Выписываем *вероятностную модель* порождения данных (Бернулли — в первой задаче; гауссов шум — во второй).
2. Записываем *функцию правдоподобия* $L(\theta)$ — вероятность наблюдать имеющиеся данные при параметре θ.
3. Находим точку максимума $\hat{\theta} = \arg\max L(\theta)$ дифференцированием по параметру.
4. Строим вокруг $\hat{\theta}$ *доверительный интервал* — с помощью неравенства Чебышёва (грубо, но универсально) или центральной предельной теоремы (точно, но при больших n).

В обеих задачах оценкой максимума правдоподобия оказалось одно и то же выражение — *выборочное среднее*. Это не случайно: за этим стоит теорема Гаусса (теорема 4.6), связывающая среднее арифметическое с гауссовым шумом, а в более общем плане — работы Фишера и Ле Кама об асимптотической оптимальности ОМП.

Что дальше. Возникающая задача минимизации суммы квадратов перестаёт быть тривиальной, как только величина μ перестаёт быть скалярной и начинает зависеть от других переменных — скажем, мы хотим найти, *как* время падения связано с высотой (Галилей), *как* период обращения планеты связан с радиусом её орбиты (Кеплер) или, в общем случае, *какова* наилучшая линейная зависимость между y и вектором признаков $x \in \mathbb{R}^d$. Этот сюжет — *линейная регрессия* — будет в следующем параграфе. А затем мы шаг за шагом, не теряя нити, дойдём до глубоких нейронных сетей.

Задачи для самостоятельной работы

1. В социологическом опросе $n = 400$, за кандидата A ответили $k = 220$. Постройте $90\,\%$ доверительный интервал для доли p (а) по неравенству Чебышёва; (б) по ЦПТ. Можно ли утверждать, что A победит с вероятностью $\geq 90\,\%$?
2. Игральную кость подбросили $n = 600$ раз. Выпало $k = 120$ шестёрок. Найдите оценку максимума правдоподобия вероятности p выпадения шестёрки и постройте $95\,\%$ ДИ. Согласуется ли результат с гипотезой «кость честная»?
3. Запрограммируйте метод Монте-Карло для оценки числа π. Постройте график зависимости абсолютной ошибки $|\hat{\pi}_N - \pi|$ от N в двойной логарифмической шкале и убедитесь, что наклон близок к $-1/2$.
4. Покажите, что для случайных величин $X_1, \dots, X_n$ с плотностью Лапласа $f(x;\mu) = \frac{1}{2}\exp(-|x-\mu|)$ оценкой максимума правдоподобия параметра μ является *выборочная медиана*, а не среднее. (Указание: суммируйте $-\ln f$ и минимизируйте.) Чем это объясняет известную «робастность» медианы по отношению к выбросам?
5. * Докажите теорему 4.6 полностью. Указание: подставьте $X_2 = \dots = X_n = 0$, тогда из условия $\sum \rho(X_i - \hat{\mu}) = 0$ при $\hat{\mu} = X_1/n$ выведите функциональное уравнение для ρ.
6. * Метод Монте-Карло для интеграла. Пусть требуется оценить $I = \int_0^1 g(x)\,dx$, где g — известная функция. Покажите, что оценкой максимума правдоподобия в подходящей вероятностной модели является $\hat{I}_N = \frac{1}{N}\sum_{i=1}^{N} g(U_i)$, $U_i \sim U(0,1)$. Применяя ЦПТ, оцените сверху число точек, при котором ошибка приближённого вычисления $\int_0^1 e^{-x^2}dx$ не превзойдёт 10^{-3} с уверенностью $95\,\%$.

4.2 Линейная регрессия и метод наименьших квадратов

В предыдущем параграфе мы оценивали *скалярную* величину μ: вес быка, долю голосов, площадь множества. На практике куда чаще встречается другая постановка: мы хотим выяснить, *как* одна величина зависит от другой. Период обращения планеты от радиуса её орбиты. Время падения от высоты. Цена квартиры от её площади. Доход компании от числа сотрудников.

Простейшая модель такой зависимости — *линейная*. И именно её мы и будем сейчас изучать.

4.2.1 Постановка задачи

Дано: n пар чисел $(x_1, y_1), \dots, (x_n, y_n)$ — результат измерений. Будем считать, что они связаны (приближённо!) линейной зависимостью

$$y_i \;=\; a\,x_i + b + \xi_i, \qquad i = 1, \dots, n, \tag{4.16}$$

где

- a, b — неизвестные коэффициенты, которые предстоит определить;
- ξ_i — случайные ошибки измерений (или просто «отклонения от точной линейной зависимости», вызванные тем, что в природе идеально линейных зависимостей не бывает).

Задача **линейной регрессии** — найти такие $\hat{a}$, $\hat{b}$, чтобы прямая $y = \hat{a}x + \hat{b}$ как можно лучше описывала наши данные.

4.2.2 Метод наименьших квадратов как ОМП при гауссовом шуме

Какой смысл вкладывать в слова «как можно лучше»? Эту неопределённость снимает принцип максимума правдоподобия из параграфа 4.1. Предположим, что ошибки ξ_i независимы и имеют нормальное распределение $\mathcal{N}(0, \sigma^2)$. Тогда из модели (4.16) следует, что и каждое y_i нормально распределено — с математическим ожиданием $ax_i + b$ и дисперсией σ^2. Совместная плотность выборки:

$$L(a, b) \;=\; \prod_{i=1}^{n} \frac{1}{\sigma\sqrt{2\pi}} \exp\!\left(-\frac{(y_i - ax_i - b)^2}{2\sigma^2}\right).$$

Логарифмируем и оставляем только слагаемые, зависящие от a, b:

$$\ln L(a, b) \;=\; C \;-\; \frac{1}{2\sigma^2} \sum_{i=1}^{n} (y_i - ax_i - b)^2, \tag{4.17}$$

C — константа, нам не интересная. Максимизация $\ln L$ по (a, b) эквивалентна минимизации функционала

$$\boxed{S(a, b) \;=\; \sum_{i=1}^{n} \big(y_i - ax_i - b\big)^2 \longrightarrow \min_{a,b}.} \tag{4.18}$$

Это и есть знаменитый **метод наименьших квадратов** (МНК, англ. *Ordinary Least Squares*, OLS).

> По теореме Гаусса (теорема 4.6), сумма квадратов — это *единственный* выбор функционала ошибки, при котором ОМП параметра $\mu = ax + b$ соответствует гауссовскому шуму. Замените квадраты на модули $|y_i - ax_i - b|$ — и вы будете неявно предполагать лапласовское распределение ошибок. Замените на четвёртую степень — и неявно предположите ещё более «острое» распределение. Гауссов шум *привилегирован* тем, что он естественно возникает в природе, когда ошибка — результат сложения большого числа независимых малых факторов.

4.2.3 Почему шум часто оказывается нормальным

Прежде чем считать формулы, разберёмся с предположением $\xi_i \sim \mathcal{N}(0, \sigma^2)$. Откуда такая уверенность? Дело в двух свойствах нормального распределения, которые делают его «магнитом» в теории вероятностей.

Свойство 1: устойчивость относительно суммирования. Если $\eta_1 \sim \mathcal{N}(\mu_1, \sigma_1^2)$ и $\eta_2 \sim \mathcal{N}(\mu_2, \sigma_2^2)$ независимы, то их сумма снова нормальна: $\eta_1 + \eta_2 \sim \mathcal{N}(\mu_1 + \mu_2,\, \sigma_1^2 + \sigma_2^2)$. Доказательство (через характеристические функции или прямой подсчёт свёртки плотностей) выходит за рамки школьной программы, но факт стоит запомнить: нормальное распределение — это «алгебраически самосогласованный» класс. Большинство других распределений после суммирования меняет форму, а нормальное — нет.

Свойство 2: ЦПТ. Гораздо более глубокий факт — центральная предельная теорема (теорема 4.3): сумма большого числа *независимых* случайных величин (любой природы, лишь бы с конечной дисперсией) распределена *приблизительно* нормально. В жизни ошибка измерения почти никогда не имеет одной-единственной природы: на неё влияют тепловые колебания прибора, вибрации, нечёткость зрения наблюдателя, погрешность шкалы, изменения освещённости и многое другое. Каждый из этих факторов даёт маленькую случайную поправку, а их сумма — по ЦПТ — получается почти нормальной (рис. 4.8).

Рис. 4.8. Иллюстрация ЦПТ. Слева: гистограмма единичной равномерной случайной величины ($n = 1$, форма — прямоугольник). В центре: сумма $n = 4$ независимых равномерных СВ (после нормировки). Справа: $n = 16$. Красная кривая — плотность стандартного нормального распределения. Уже при $n = 16$ совпадение почти идеальное.

Поэтому, когда мы пишем «шум измерения — нормальный», за этим стоят *два* обоснования: ЦПТ говорит, что природа благосклонна к гауссовскому шуму; а теорема Гаусса (теорема 4.6) говорит, что, выбрав МНК как метод оценки, мы автоматически предполагаем именно такой шум.

4.2.4 Решение задачи МНК. Принцип Ферма

Найдём минимум функции $S(a, b)$ из (4.18). Это функция от двух переменных. По принципу Ферма (известному из главы 2), в точке минимума обе частные производные обращаются в ноль:

$$\frac{\partial S}{\partial a} = 0, \qquad \frac{\partial S}{\partial b} = 0. \tag{4.19}$$

Вычислим частные производные явно:

$$\frac{\partial S}{\partial a} = \sum_{i=1}^{n} \frac{\partial}{\partial a}(y_i - a x_i - b)^2 = \sum_{i=1}^{n} 2(y_i - a x_i - b)\cdot(-x_i) = -2\sum_{i=1}^{n} x_i(y_i - a x_i - b),$$

$$\frac{\partial S}{\partial b} = \sum_{i=1}^{n} 2(y_i - a x_i - b)\cdot(-1) = -2\sum_{i=1}^{n}(y_i - a x_i - b).$$

Приравнивание обеих производных нулю даёт систему *нормальных уравнений*:

$$a\sum_{i=1}^{n} x_i^2 + b\sum_{i=1}^{n} x_i = \sum_{i=1}^{n} x_i y_i, \tag{4.20}$$

$$a\sum_{i=1}^{n} x_i + b\,n = \sum_{i=1}^{n} y_i. \tag{4.21}$$

Обозначим выборочные средние $\bar{x} = \frac{1}{n}\sum x_i$, $\bar{y} = \frac{1}{n}\sum y_i$. Делим (4.21) на n: $a\bar{x} + b = \bar{y}$, откуда

$$\boxed{\hat{b} = \bar{y} - \hat{a}\,\bar{x}.} \tag{4.22}$$

Это означает, что прямая $y = \hat{a}x + \hat{b}$ *всегда проходит через «центр масс» данных* $(\bar{x}, \bar{y})$.

Подставляем $\hat{b}$ в (4.20) и после простых преобразований (умножаем (4.21) на $\bar{x}$ и вычитаем из (4.20)) получаем

$$\boxed{\hat{a} = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^{n}(x_i - \bar{x})^2}.} \tag{4.23}$$

Числитель — так называемая *ковариация* выборок x и y (точнее, n раз ковариация), знаменатель — n раз выборочная дисперсия x. Формулы (4.22), (4.23) — основной рабочий инструмент простой линейной регрессии.

> МНК минимизирует сумму квадратов *вертикальных* расстояний от точек данных до прямой — не перпендикулярных и не горизонтальных, а именно *вертикальных* (рис. 4.9). Это связано с тем, что в модели (4.16) мы считаем, что значение x_i — *точное* (точка на оси абсцисс), а вся неопределённость сосредоточена в y_i. Если, напротив, ошибки есть и в x, и в y, на смену МНК приходит так называемый *общий МНК* (Total Least Squares), минимизирующий перпендикулярные расстояния.

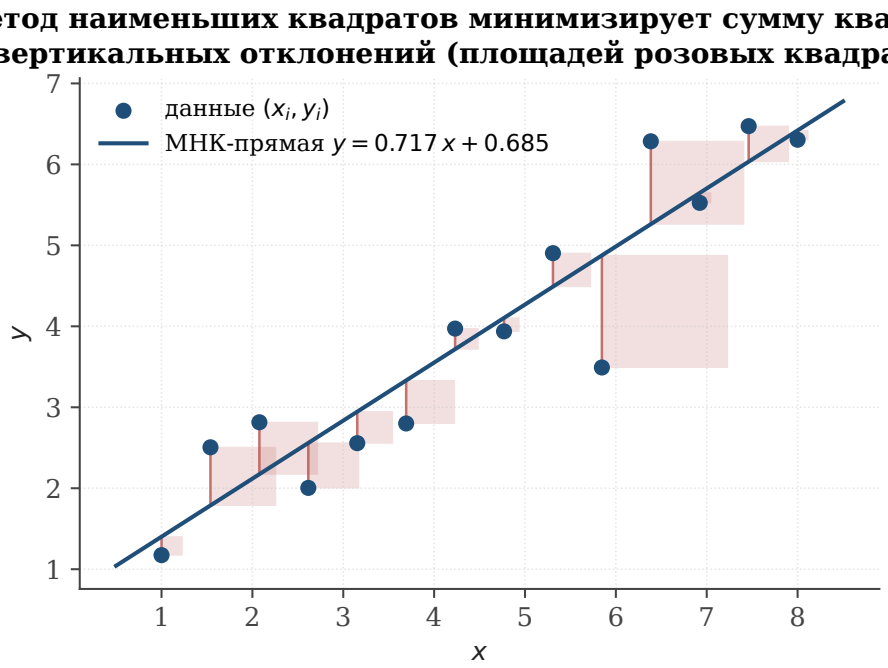


Рис. 4.9. Геометрия МНК. Регрессионная прямая (синяя) проводится так, чтобы сумма площадей розовых квадратов была наименьшей. Сторона каждого квадрата — это *вертикальный* остаток $|y_i - \hat{a}x_i - \hat{b}|$, а площадь — его квадрат.

Пример 4.8. Прямой счёт по формуле

Возьмём шесть точек: $(1, 2{,}1)$, $(2, 3{,}9)$, $(3, 5{,}8)$, $(4, 8{,}3)$, $(5, 10{,}0)$, $(6, 11{,}9)$. Считаем по (4.22), (4.23):

$$\bar{x} = 3{,}5, \quad \bar{y} = 7{,}0, \quad \sum (x_i - \bar{x})^2 = 17{,}5, \quad \sum (x_i - \bar{x})(y_i - \bar{y}) = 34{,}65.$$

Откуда $\hat{a} = 34{,}65/17{,}5 \approx 1{,}980$ и $\hat{b} = 7{,}0 - 1{,}98 \cdot 3{,}5 \approx 0{,}07$. Регрессионная прямая: $\hat{y} \approx 1{,}98\,x + 0{,}07$ — то есть данные «выложены вдоль» зависимости $y \approx 2x$.

4.2.5 Пример 1. III закон Кеплера по данным Тихо Браге

В качестве первого «настоящего» применения МНК разберём задачу, с которой началась современная физика. К XVI веку астрономы уже умели определять для каждой планеты:

- *период обращения* T (за сколько лет планета возвращается на ту же точку звёздного неба), и
- *средний радиус орбиты* a — большую полуось эллипса.

Тихо Браге (1546–1601), один из величайших астрономов-наблюдателей доньютоновской эпохи, посвятил всю жизнь точнейшим наблюдениям движения планет с помощью огромных угломерных инструментов на острове Вен (Дания). После его смерти эти таблицы достались его помощнику — молодому немцу **Иоганну Кеплеру** (1571–1630), который потратил два десятилетия на их интерпретацию.

> Тихо Браге — величайший астроном-наблюдатель домикроскопной эпохи. Точность его измерений угловых положений планет составляла $\sim 1'$ (угловая минута) — предел для невооружённого глаза. К концу жизни он накопил каталог из почти тысячи звёзд и подробнейшие записи о движении пяти видимых планет за 21 год.
> Кеплер унаследовал эти таблицы и в 1605 г. открыл *первый* закон движения планет: каждая планета движется по эллипсу, в одном из фокусов которого находится Солнце. В 1609 г. он добавил *второй*: радиус-вектор планеты заметает равные площади за равные времена. А в 1619 г. (в трактате *Harmonices Mundi*, «Гармонии мира») он опубликовал *третий* закон: квадраты периодов обращения относятся как кубы больших полуосей. Сам Кеплер выписал этот закон в форме $T^2 \sim a^3$, не зная ни логарифмов в общеупотребительном виде, ни тем более понятия регрессии. Но именно на «эпохе Браге–Кеплера» данные впервые позволили установить количественный закон одного астрономического параметра как функции другого.

Постановка как задача регрессии. Подозреваем степенную зависимость $T = C \cdot a^{\beta}$. Логарифмируя обе части, получаем линейную зависимость в логарифмических осях:

$$\lg T \;=\; \beta \,\lg a \;+\; \lg C. \tag{4.24}$$

Обозначим $x_i = \lg a_i$, $y_i = \lg T_i$. Это уже стандартная задача линейной регрессии — ищем $\hat{\beta}$ и $\widehat{\lg C}$ по формулам (4.23), (4.22).

Данные. Шесть планет, известных Кеплеру (значения близки к тем, что использовал он сам, но приведены в современных единицах):

Планета	a, а. е. [1]	T, лет	$\lg a$	$\lg T$
Меркурий	0,3871	0,2408	−0,4122	−0,6183
Венера	0,7233	0,6152	−0,1407	−0,2110
Земля	1,0000	1,0000	0	0
Марс	1,5237	1,8809	0,1829	0,2744
Юпитер	5,2034	11,862	0,7163	1,0742
Сатурн	9,5371	29,457	0,9794	1,4692

[1] Астрономическая единица (а. е.) равна среднему расстоянию от Земли до Солнца, $\approx 149{,}6$ млн км.

Расчёт МНК. Программа из десяти строк (или подсчёт «вручную» по формулам (4.23), (4.22)) даёт:

$$\hat{\beta} \;=\; 1{,}4999628\ldots, \qquad \widehat{\lg C} \;=\; -0{,}00003\ldots, \qquad R^2 \;=\; 0{,}99999996. \tag{4.25}$$

Коэффициент $\hat{\beta}$ совпадает с $3/2$ с точностью до четвёртого знака после запятой. Это и есть III закон Кеплера:

$$\boxed{T^2 \;=\; C \cdot a^3, \qquad C \approx 1 \text{ (в единицах год}/\text{а. е.}^{3/2}\text{)}.}$$

Это соотношение и графическое спрямление данных Кеплера в двойной логарифмической шкале показаны на рис. 4.10.

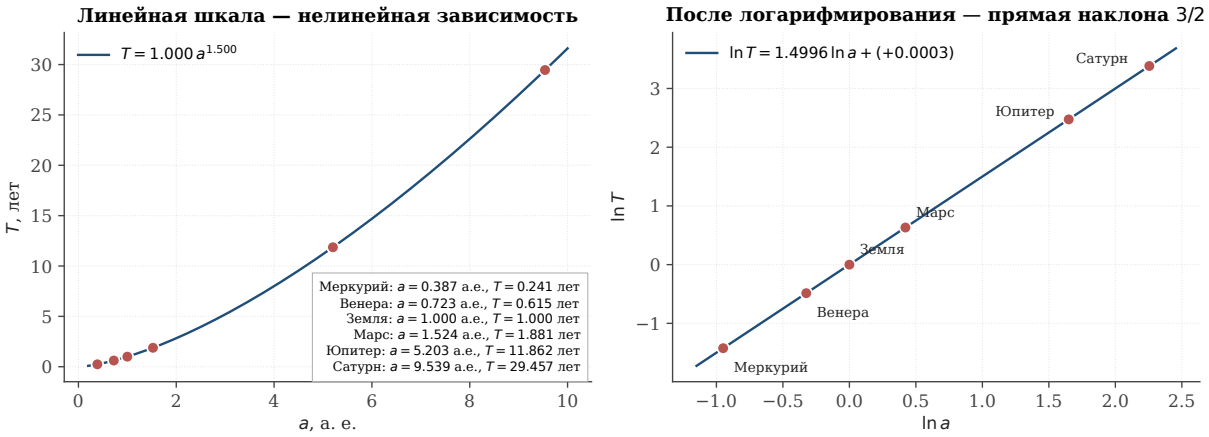


Рис. 4.10. Слева: данные о периоде обращения и радиусе орбиты для шести известных Кеплеру планет — степенная зависимость. Справа: те же данные в двойной логарифмической шкале — идеальная прямая. Линия МНК (синяя) имеет наклон $\hat{\beta} = 1{,}5000$ с коэффициентом детерминации $R^2 = 0{,}999\,999\,96$ — одно из самых аккуратных эмпирических соотношений в истории науки.

> Кеплер не имел в распоряжении ни МНК (его открыли через 100 лет: Лежандр — в 1805 г., Гаусс — в 1809 г.), ни графиков логарифмов в современном виде. Он подбирал зависимость *руками*, сравнивая отношения. После многомесячных проб он заметил, что $T^2_{\text{Юпитер}}/T^2_{\text{Земля}} \approx 140{,}7$ и $a^3_{\text{Юпитер}}/a^3_{\text{Земля}} \approx 140{,}9$ — почти равенство. Сегодня та же зависимость устанавливается за десяток строк кода — но идею Кеплера это не уменьшает. Напротив, наглядно показывает, насколько мощным инструментом стал МНК для последующих поколений физиков.

4.2.6 Пример 2. Ускорение свободного падения по данным Галилея

Галилео Галилей (1564–1642) стоял у истоков физики как *экспериментальной* науки. Его трактат «Беседы и математические доказательства, касающиеся двух новых отраслей науки» (1638) изложил закон свободного падения: пройденный путь пропорционален *квадрату* времени падения,

$$h \;=\; \frac{g}{2}\,t^2, \tag{4.26}$$

где g — ускорение свободного падения. Историки до сих пор спорят, на самом ли деле Галилей сбрасывал предметы с Пизанской башни (свидетельства его ученика Винченцо Вивиани, написанные через десятки лет, ставятся под сомнение). Но даже как «мысленный эксперимент» этот сюжет великолепно подходит, чтобы продемонстрировать МНК на ещё одной задаче (рис. 4.11).

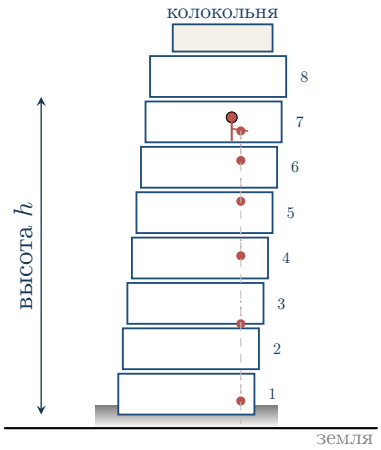


Рис. 4.11. Эксперимент Галилея на Пизанской башне (художественная реконструкция). Из-за небольшого наклона башни (около 4° к вертикали в современном состоянии) экспериментатору удобно сбрасывать предметы *вертикально вниз*, без начальной скорости: они летят почти вдоль внешней стены, не задевая её. Время падения t измеряется для разных уровней — разных значений высоты h.

Регрессионная модель. Записывать t через h напрямую нельзя (получится корень):

$$t = \sqrt{\frac{2h}{g}}.$$

Поэтому регрессия выписывается для *обратной* зависимости — высота от квадрата времени:

$$h = \frac{g}{2}\,t^2 + \xi. \tag{4.27}$$

Введя $X = t^2$, $Y = h$ и $A = g/2$, получаем линейную модель $Y = AX + \xi$ — частный случай (4.16) с $b = 0$. (Свободный член можно оставить и формально — он покажет систематическую погрешность типа «реакции экспериментатора при пуске секундомера»; см. ниже.)

«Данные». Воспроизведём эксперимент в духе *Discorsi*. Высоты ярусов Пизанской башни (приблизительно): 7, 12, 19, 25,5, 32, 38, 44, 50 м над землёй. Для каждой высоты «измеряем» время падения — настоящий Галилей использовал водяные часы (взвешивал воду, вытекшую за время падения!) с типичной погрешностью около $\sigma \sim 0{,}05$ секунды на одно измерение. Получим набор пар (h_i, t_i):

i	h_i, м	t_i, с (измерено)	t_i^2, с^2
1	7,0	1,253	1,569
2	12,0	1,570	2,465
3	19,0	1,897	3,597
4	25,5	2,299	5,287
5	32,0	2,497	6,235
6	38,0	2,793	7,799
7	44,0	3,092	9,562
8	50,0	3,196	10,213

Расчёт МНК. Применяем формулы (4.23), (4.22) к парам (t_i^2, h_i): $\hat{A} \approx 4{,}757$, $\hat{b} \approx 0{,}653$. Отсюда

$$\hat{g} = 2\hat{A} \approx 9{,}51 \text{ м/с}^2,$$

при «истинном» значении $g = 9{,}81$ м/с^2. Относительная погрешность $\sim 3\,\%$ — абсолютно реалистичный результат для экспериментатора XVII века с водяными часами. Коэффициент детерминации $R^2 \approx 0{,}9932$ — то есть модель объясняет более $99\,\%$ дисперсии в данных (рис. 4.12).

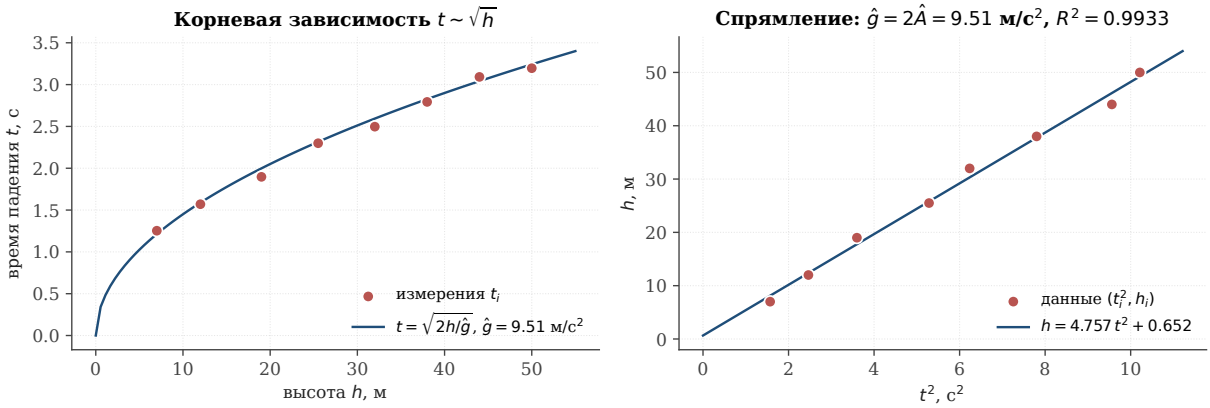


Рис. 4.12. Слева: измеренные времена падения от высоты — зависимость $t \sim \sqrt{h}$. Справа: те же данные после «спрямления» по оси t^2. Линия МНК (синяя) почти проходит через начало координат, её угловой коэффициент $\hat{A} = 4{,}757$ даёт $\hat{g} = 2\hat{A} \approx 9{,}51$ м/с^2. Маленький свободный член $\hat{b} = 0{,}65$ м соответствует реакции экспериментатора — задержке между моментом разжатия пальцев и началом отсчёта по водяным часам.

> И в задаче Кеплера, и в задаче Галилея исходная зависимость — не линейная ($T = Ca^{3/2}$, $t = \sqrt{2h/g}$). Прежде чем применить МНК, мы её *линеаризуем*: берём логарифм у Кеплера и квадрат t у Галилея. Это очень распространённый трюк: подбор показателя степени, экспоненциальный рост, степенные законы — все они сводятся к линейной регрессии заменой переменных.
> Однако с этим связано важное наблюдение. После замены переменных *меняется* и распределение шума: гауссов шум на исходных y после взятия логарифма уже не гауссов на $\lg y$. Поэтому формально «линеаризованная» оценка слегка отличается от ОМП в исходной модели — но в большинстве практических задач отличие незначительно, и линеаризацией пользуются как удобным и универсальным приёмом.

4.2.7 Множественная линейная регрессия (краткий обзор)

В реальных задачах одной входной переменной обычно мало. Цена квартиры зависит *одновременно* от её площади, расстояния до метро, этажа, года постройки. Время на маршруте — от длины пути, числа светофоров, загруженности, времени суток.

Постановка. Пусть n — число наблюдений, d — число входных признаков. Для i-го наблюдения известны: вектор признаков $\mathbf{x}_i = (x_{i,1}, \ldots, x_{i,d}) \in \mathbb{R}^d$ и значение отклика $y_i \in \mathbb{R}$. Запишем общую **множественную линейную модель**:

$$y_i \;=\; w_1 x_{i,1} + w_2 x_{i,2} + \cdots + w_d x_{i,d} + w_0 + \xi_i, \qquad i = 1, \ldots, n. \tag{4.28}$$

Здесь $w_1, \ldots, w_d$ — коэффициенты при признаках, w_0 — свободный член, $\xi_i \sim \mathcal{N}(0, \sigma^2)$ — независимый гауссов шум.

Матричная запись. Соберём **матрицу плана** $X \in \mathbb{R}^{n \times (d+1)}$, у которой i-я строка — это $(x_{i,1}, x_{i,2}, \ldots, x_{i,d}, 1)$ (последняя единица позволяет учесть свободный член w_0 как обычный

коэффициент). Вектор параметров $\mathbf{w} = (w_1, \dots, w_d, w_0)^\top \in \mathbb{R}^{d+1}$, вектор откликов $\mathbf{y} = (y_1, \dots, y_n)^\top$. Тогда модель компактно записывается одной строкой:

$$\mathbf{y} \;=\; X\,\mathbf{w} + \boldsymbol{\xi}.$$

Метод наименьших квадратов. Сумма квадратов ошибок принимает вид

$$S(\mathbf{w}) \;=\; \sum_{i=1}^{n} \bigl(y_i - \mathbf{x}_i^\top \mathbf{w}\bigr)^2 \;=\; \|\mathbf{y} - X\mathbf{w}\|^2,$$

где для краткости $\mathbf{x}_i^\top \mathbf{w} = x_{i,1}w_1 + \dots + x_{i,d}w_d + w_0$ (т. е. стандартное скалярное произведение). Приравнивание градиента S нулю даёт систему *нормальных уравнений*:

$$\boxed{(X^\top X)\,\hat{\mathbf{w}} \;=\; X^\top \mathbf{y}} \qquad \text{(нормальные уравнения)}. \tag{4.29}$$

Если матрица $X^\top X$ размера $(d+1)\times(d+1)$ *невырождена*, отсюда $\hat{\mathbf{w}} = (X^\top X)^{-1} X^\top \mathbf{y}$. Эта формула — наиболее известная в линейной алгебре статистической оценки.

Условие применимости. Невырожденность $X^\top X$ равносильна тому, что столбцы X *линейно независимы*: ни один признак не является линейной комбинацией других, и число наблюдений не меньше числа признаков ($n \geq d+1$). Если это не так — решение не единственно, и приходится либо отбирать признаки, либо использовать *регуляризацию* (см. раздел 4.3.5).

Пример 4.9. Цена квартиры по двум признакам

Пусть для пяти квартир известны площадь $x_{i,1}$ (м^2), расстояние до метро $x_{i,2}$ (минуты пешком) и цена y_i (млн руб.):

i	$x_{i,1}$	$x_{i,2}$	y_i
1	35	5	9,5
2	45	12	10,0
3	60	8	13,5
4	75	3	17,0
5	50	20	9,8

Подставим $X \in \mathbb{R}^{5\times 3}$ (последний столбец из единиц) в формулу (4.29) и численно решим систему. Получим $\hat{\mathbf{w}} \approx (0{,}183,\ -0{,}142,\ 2{,}74)$, то есть

$$\hat{y} \;=\; 0{,}183\,x_1 \;-\; 0{,}142\,x_2 \;+\; 2{,}74.$$

Содержательно: каждый дополнительный квадратный метр прибавляет к цене $\approx$ 183 тыс. руб., каждая дополнительная минута до метро *уменьшает* цену на $\approx$142 тыс. руб., а свободный член 2,74 млн — это «базовая стоимость», когда оба признака равны нулю.

Главная идея. С ростом размерности *тип формул не меняется*. МНК — универсальный инструмент, одинаково работающий и для одного-единственного признака (Кеплер, Галилей), и для тысячи признаков (модели ценообразования, прогнозирование спроса, физика частиц). Мы вернёмся к этим формулам в параграфе 4.4 при обсуждении малоранговых аппроксимаций матриц.

4.2.8 Резюме параграфа

Метод наименьших квадратов — первый *реальный* инструмент анализа данных, который мы получили. Его эквивалентность принципу максимума правдоподобия в гауссовой модели делает его универсальным.

На примере Кеплера МНК даёт III закон с точностью до семи знаков: наклон $\hat{\beta} = 1{,}49996 \approx 3/2$. На примере Галилея — ускорение свободного падения $\hat{g} \approx 9{,}51$ м/с^2 при истинном 9,81.

Главные идеи, которые мы должны вынести из этого параграфа и которые будут проявляться дальше в более сложных моделях:

1. **Принцип максимума правдоподобия**, применённый к модели «параметр плюс гауссов шум», даёт *минимизацию суммы квадратов* как универсальный функционал ошибки.
2. Принцип Ферма (приравнивание частных производных к нулю) приводит к *системе линейных уравнений*, которая называется системой нормальных уравнений.
3. Зависимость, не являющаяся линейной, часто *спрямляется* заменой переменных (логарифмирование, возведение в квадрат) и после этого тоже становится задачей МНК.

В следующем параграфе мы шагнём от линейных моделей к существенно нелинейным — нейронным сетям. Принцип максимума правдоподобия и тут останется методологической отправной точкой, но функционал станет сложнее, а вместо аналитического решения нормальных уравнений мы будем использовать численные методы — градиентный спуск и его варианты.

Задачи для самостоятельной работы

1. Для точек $(1,1{,}2)$, $(2,2{,}3)$, $(3,3{,}5)$, $(4,4{,}3)$, $(5,5{,}1)$ найдите $\hat{a}, \hat{b}$ методом наименьших квадратов. Постройте график «данные + регрессионная прямая».
2. Выведите формулу (4.23) в явном виде, проделав все преобразования между (4.20) и (4.23).
3. Кеплер. По данным таблицы выше посчитайте *вручную* (или с помощью калькулятора) $\bar{x} = \overline{\lg a}$, $\bar{y} = \overline{\lg T}$, ковариацию и дисперсию, и убедитесь, что МНК даёт $\hat{\beta} \approx 1{,}5$.
4. Галилей. Зачем в (4.27) мы добавили свободный член ξ? Какое физическое явление он моделирует?
5. Постройте регрессию веса человека (y) от его роста (x) на выборке из 10 ваших знакомых (можно вымышленных). Сравните ваш результат с известной эмпирической формулой Брока $y \approx x - 100$ (см. массой в кг, рост в см).
6. * Покажите, что МНК-оценка $\hat{a}$ из (4.23) *несмещённая*: $\mathbb{E}[\hat{a}] = a$, если на самом деле $y_i = ax_i + b + \xi_i$ с независимыми ошибками с нулевым математическим ожиданием. Найдите $\mathbb{D}[\hat{a}]$.
7. * Метод наименьших квадратов с весами. Пусть нам известно, что точность i-го измерения характеризуется собственной дисперсией σ_i^2 (разные приборы — разная погрешность). Покажите, что ОМП-оценка минимизирует *взвешенную* сумму квадратов $\sum_i \sigma_i^{-2}(y_i - ax_i - b)^2$. Найдите для этой задачи аналог формул (4.23), (4.22).

4.3 Задача классификации и нейронные сети

До сих пор мы решали задачи, в которых ответ модели — вещественное число. В задаче выборов это была доля голосов; у Гальтона — масса быка; у Кеплера — период обращения. Такие задачи называют задачами **регрессии**.

Но огромный класс практических задач устроен иначе. Алгоритму показывают фотографию — он должен сказать «кошка» или «собака». Письмо приходит в почтовый ящик — система решает: «спам» или «не спам». Изображение с томографа поступает в больницу — модель должна определить, есть ли на нём патология. Это задачи **классификации**: каждому входному объекту нужно сопоставить *дискретную метку* из конечного списка возможных классов.

В этом параграфе мы:

1. познакомимся с классической задачей классификации — распознаванием рукописных цифр;
2. покажем, как из принципа максимума правдоподобия выводится функционал *кросс-энтропии* — стандартная мера качества классификатора;
3. построим простейшие модели — полносвязную сеть и свёрточную сеть — и сравним их;
4. обнаружим важнейшее явление — *переобучение* (обнаружат его и сами вычислительные эксперименты);
5. обсудим теоретическое обоснование того, что нейронные сети могут приближать любую разумную зависимость — теоремы Колмогорова–Арнольда и Цыбенко;
6. изучим основной численный метод обучения — *градиентный спуск* — и его эффективную реализацию для нейронных сетей — *обратное распространение ошибки*.

4.3.1 Задача классификации MNIST и UCI Digits

Самый знаменитый «учебный» датасет для классификации — это **MNIST** (Modified National Institute of Standards and Technology database): 70 000 чёрно-белых изображений рукописных цифр размера 28×28 пикселей. Его собрал в 1998 г. Янн ЛеКун, который позже получил премию Тьюринга за работы по глубокому обучению. Все обсуждения и итоговые выводы этой главы относятся именно к MNIST как к каноническому ориентиру — именно его читатель встретит во всех профессиональных курсах и публикациях по машинному обучению.

Почему в численных экспериментах книги — не MNIST. Чтобы все приводимые ниже эксперименты можно было *полностью воспроизвести на школьном ноутбуке за несколько секунд*, мы используем **UCI Optical Digits** — меньший близкий по структуре датасет: те же 10 классов цифр (0–9), но изображения 8×8 (всего 1 797 примеров вместо 70 000). На полном MNIST обучение глубокой сети занимает минуты или десятки минут даже на современном железе и плохо ложится в формат «прочитал страницу учебника — запустил код — получил картинку». Качественные эффекты — переобучение, U-образная кривая ошибки, преимущество свёрточных сетей перед полносвязными — проявляются и на «миниатюре» точно так же; в подписях к рисункам сравнения с полноразмерным MNIST и ImageNet будут приведены отдельно (рис. 4.13).

Математическая постановка. Изображение размера 8×8 — это таблица из 64 чисел (значений яркости от 0 до 16). «Развернём» эту таблицу в один длинный вектор:

$$\mathbf{x} = (x_1, x_2, \ldots, x_{64}) \in \mathbb{R}^{64}.$$

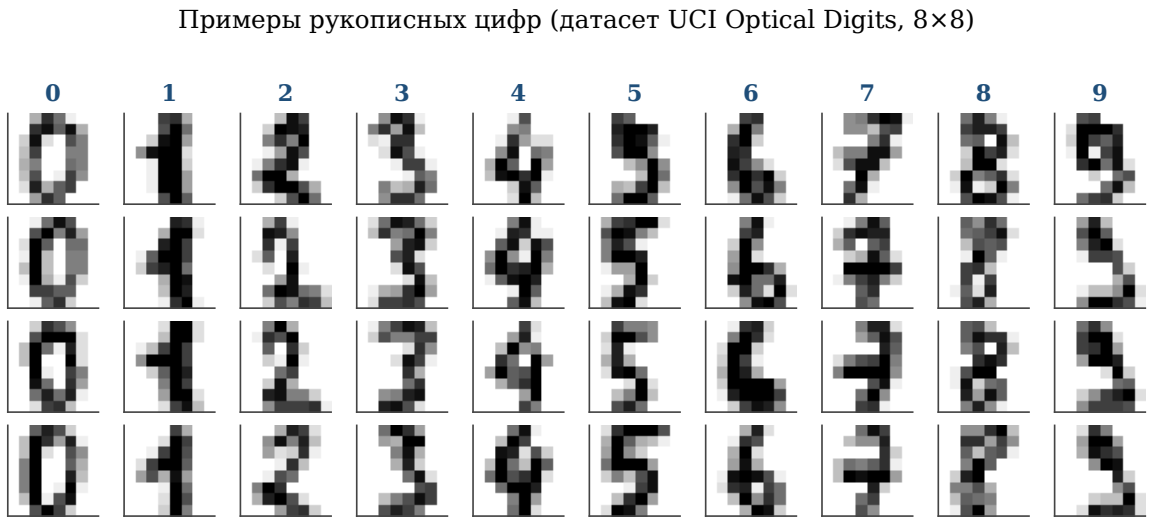


Рис. 4.13. Примеры изображений из датасета UCI Optical Digits: 10 классов цифр от 0 до 9, каждое изображение — $8 \times 8 = 64$ пикселя в оттенках серого. Внутри каждого класса написания заметно различаются — это и есть «трудность задачи»: модель должна обобщать, а не запоминать.

Так каждая цифра превращается в точку 64-мерного пространства. Для полноразмерного MNIST — 784-мерного (рис. 4.14).

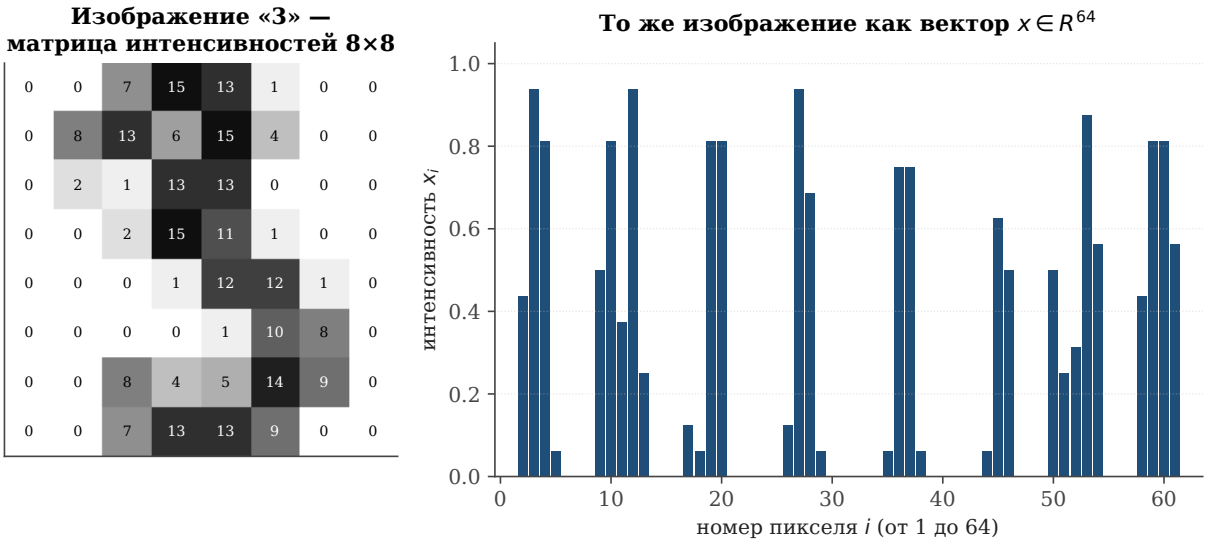


Рис. 4.14. Одно и то же изображение цифры в двух представлениях. Слева — матрица интенсивностей 8×8. Справа — та же цифра, развёрнутая в вектор $\mathbf{x} \in \mathbb{R}^{64}$ (по столбцам). Модели машинного обучения работают именно со вторым представлением — вектором чисел.

Метка — одно из 10 значений: $y \in \{0, 1, \ldots, 9\}$. Датасет — это набор пар

$$\{(\mathbf{x}_i,\, y_i)\}_{i=1}^{N}, \qquad \mathbf{x}_i \in \mathbb{R}^{64}, \quad y_i \in \{0, \ldots, 9\}.$$

Задача классификации — построить *функцию-классификатор* $\hat{y} = \hat{y}(\mathbf{x})$, которая по новому, ранее не виденному вектору $\mathbf{x}$ предсказывает правильную метку.

4.3.2 Чем классификация отличается от регрессии

В задаче регрессии (раздел 4.2) функция, которую мы искали, отображала $\mathbb{R}^d \to \mathbb{R}$ — то есть выдавала одно вещественное число. В задаче классификации мы хотим, чтобы алгоритм выдавал *категориальный* ответ. Принципиальные отличия — два.

Отличие 1: ответ не число, а распределение вероятностей. Хорошая модель должна выдавать не «эта цифра — семёрка», а более информативно: «семёрка с вероятностью 92 %, единица — 5 %, четвёрка — 2 %, остальное — меньше процента». Соответственно выход модели — это *вектор* из 10 чисел, неотрицательных и суммирующихся в 1:

$$\mathbf{p}(\mathbf{x}) \;=\; \big(p_0(\mathbf{x}),\, p_1(\mathbf{x}),\, \ldots,\, p_9(\mathbf{x})\big), \qquad p_c \geq 0, \quad \sum_{c=0}^{9} p_c = 1.$$

Здесь $p_c(\mathbf{x})$ — предсказанная моделью вероятность того, что $\mathbf{x}$ изображает цифру c. Окончательное предсказание метки — $\hat{y}(\mathbf{x}) = \arg\max_c p_c(\mathbf{x})$.

Отличие 2: функционал ошибки — не сумма квадратов, а кросс-энтропия. В регрессии мы минимизировали $\sum_i (y_i - \hat{y}_i)^2$ — это была ОМП при гауссовом шуме. Сейчас «гауссова шума» нет: правильная метка y_i — это *дискретная* категория. Что вместо суммы квадратов? Принцип максимума правдоподобия — даст ответ.

Вывод кросс-энтропии из ОМП

Модель порождения данных. Считаем, что наблюдённые пары $(\mathbf{x}_i, y_i)$ порождены так: для каждого i метка y_i — случайная величина со значениями в $\{0, 1, \ldots, 9\}$ и распределением, которое зависит от $\mathbf{x}_i$:

$$\Pr(y_i = c \,|\, \mathbf{x}_i, \boldsymbol{\theta}) \;=\; p_c(\mathbf{x}_i; \boldsymbol{\theta}).$$

Здесь $\boldsymbol{\theta}$ — параметры модели (веса и смещения нейронной сети), которые предстоит определить. Совместное правдоподобие выборки размера N — это произведение вероятностей:

$$L(\boldsymbol{\theta}) \;=\; \prod_{i=1}^{N} p_{y_i}(\mathbf{x}_i; \boldsymbol{\theta}).$$

Берём отрицательный логарифм (минимизировать удобнее, чем максимизировать) и нормируем на размер выборки:

$$\boxed{\mathcal{L}(\boldsymbol{\theta}) \;\stackrel{\text{def}}{=}\; -\frac{1}{N} \sum_{i=1}^{N} \ln p_{y_i}(\mathbf{x}_i; \boldsymbol{\theta}) \;\longrightarrow\; \min_{\boldsymbol{\theta}}.} \tag{4.30}$$

Этот функционал называется **кросс-энтропией**[2] (англ. *cross-entropy*, *categorical log-loss*). Его минимизация по $\boldsymbol{\theta}$ — это в точности максимизация правдоподобия. Мы снова применили один и тот же универсальный принцип — и получили совершенно другой функционал, естественный для своей задачи.

> В теории информации Клод Шеннон (1948) ввёл понятие *энтропии* распределения $\mathbf{q}$:
>
> $$H(\mathbf{q}) \;=\; -\sum_c q_c \ln q_c,$$
>
> характеризующее «непредсказуемость» исхода. *Кросс-энтропия* двух распределений $\mathbf{q}$ (истинное) и $\mathbf{p}$ (предсказанное) определяется как
>
> $$H(\mathbf{q}, \mathbf{p}) \;=\; -\sum_c q_c \ln p_c.$$

[2] Также встречается название *logloss* или *отрицательное лог-правдоподобие*.

В нашей задаче $\mathbf{q}$ — это «one-hot» вектор истины ($q_c = 1$ только при $c = y_i$, иначе 0), а $\mathbf{p}$ — предсказание модели. Тогда $H(\mathbf{q}_i, \mathbf{p}_i) = -\ln p_{y_i}$, и формула (4.30) оказывается средней по выборке кросс-энтропией.

4.3.3 Полносвязная нейронная сеть

Нам осталось определить семейство функций $p_c(\mathbf{x}; \boldsymbol{\theta})$, по которому будет проводиться минимизация. Самое известное — *полносвязная нейронная сеть*, она же *многослойный перцептрон* (**MLP**, *multilayer perceptron*).

Идея «искусственного нейрона» — линейная функция от входов, пропущенная через нелинейность — восходит к работам **Уоррена Маккаллока и Уолтера Питтса** (1943) и обучаемому *перцептрону* Фрэнка Розенблатта (1958). Розенблатт построил физическую машину Mark I Perceptron на лампах и фоторезисторах; пресса 1958 г. восторженно писала, что «машина скоро будет ходить, разговаривать и осознавать своё существование».
Однако в 1969 г. Марвин Минский и Сеймур Пейперт в книге *Perceptrons* строго доказали, что один-единственный слой не может реализовать даже простейшую функцию — *исключающее ИЛИ* (XOR). Это охладило интерес к нейросетям почти на 15 лет.
Возрождение пришло в 1986 г., когда Дэвид Румельхарт, Джеффри Хинтон и Рональд Уильямс представили универсальный алгоритм обучения *многослойных* сетей — метод **обратного распространения ошибки** (backpropagation). С этого момента и начинается эпоха современных нейронных сетей. Подчеркнём, что общая математическая формула того, что вычислительная сложность подсчёта градиента сравнима со сложностью самого вычисления функции, в общем случае была строго установлена советскими математиками: **Юрий Геннадьевич Ким, Юрий Евгеньевич Нестеров, Александр Скоков, Виктор Черкасский** (работы начала 1980-х). Об этом подробнее ниже, в разделе 4.3.9.
В 2018 г. премию Тьюринга — «нобелевскую» премию по информатике — получили Янн ЛеКун, Джеффри Хинтон и Йошуа Бенжио — «за концептуальные и инженерные прорывы, благодаря которым глубокие нейронные сети стали важнейшим компонентом современной информатики».

Строгое определение. Нам понадобится понятие **матрично-векторного умножения**.

Определение 4.10. Умножение матрицы на вектор

Пусть $A = (a_{ij})$ — матрица размера $m \times n$ (то есть m строк и n столбцов), и $\mathbf{z} = (z_1, \dots, z_n)^\top$ — вектор из $\mathbb{R}^n$. Произведение $A\mathbf{z}$ — это вектор из $\mathbb{R}^m$, i-я координата которого равна

$$(A\mathbf{z})_i = \sum_{j=1}^{n} a_{ij}\, z_j, \qquad i = 1, \dots, m. \tag{4.31}$$

Иначе говоря, мы скалярно умножаем i-ю строку матрицы на вектор и получаем i-ю координату результата.

Пример 4.11. Простой расчёт

$$A = \begin{pmatrix} 1 & 2 & -1 \\ 3 & 0 & 4 \end{pmatrix}, \qquad \mathbf{z} = \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix}.$$

Тогда

$$A\mathbf{z} = \begin{pmatrix} 1\cdot 1 + 2\cdot 2 + (-1)\cdot 3 \\ 3\cdot 1 + 0\cdot 2 + 4\cdot 3 \end{pmatrix} = \begin{pmatrix} 2 \\ 15 \end{pmatrix}.$$

Теперь можем определить полносвязную сеть (рис. 4.15).

Определение 4.12. Полносвязная нейронная сеть

Полносвязная нейронная сеть (MLP) глубины L — это функция $\mathbf{x} \mapsto \mathbf{p}(\mathbf{x})$, заданная как композиция L *слоёв*:

$$\mathbf{z}^{(\ell)} = W^{(\ell)}\mathbf{a}^{(\ell-1)} + \mathbf{b}^{(\ell)}, \qquad \mathbf{a}^{(\ell)} = \sigma\big(\mathbf{z}^{(\ell)}\big), \qquad \ell = 1, \dots, L-1, \tag{4.32}$$

$$\mathbf{z}^{(L)} = W^{(L)}\mathbf{a}^{(L-1)} + \mathbf{b}^{(L)}, \qquad \mathbf{p}(\mathbf{x}) = \operatorname{softmax}\big(\mathbf{z}^{(L)}\big). \tag{4.33}$$

Здесь $\mathbf{a}^{(0)} = \mathbf{x}$ (вход), $W^{(\ell)} \in \mathbb{R}^{d_\ell \times d_{\ell-1}}$ — матрица весов ℓ-го слоя, $\mathbf{b}^{(\ell)} \in \mathbb{R}^{d_\ell}$ — вектор смещений (bias). Функция σ применяется покомпонентно — это *функция активации* (см. ниже). Числа $d_1, \dots, d_{L-1}$ — ширины скрытых слоёв; d_L — число классов (у нас — 10). Функция **softmax** превращает вектор произвольных вещественных чисел в распределение вероятностей:

$$\operatorname{softmax}(\mathbf{z})_c = \frac{e^{z_c}}{\sum_{c'} e^{z_{c'}}}. \tag{4.34}$$

Совокупность всех матриц $W^{(\ell)}$ и векторов $\mathbf{b}^{(\ell)}$ объявляется *обучаемыми параметрами* $\boldsymbol{\theta}$ модели.

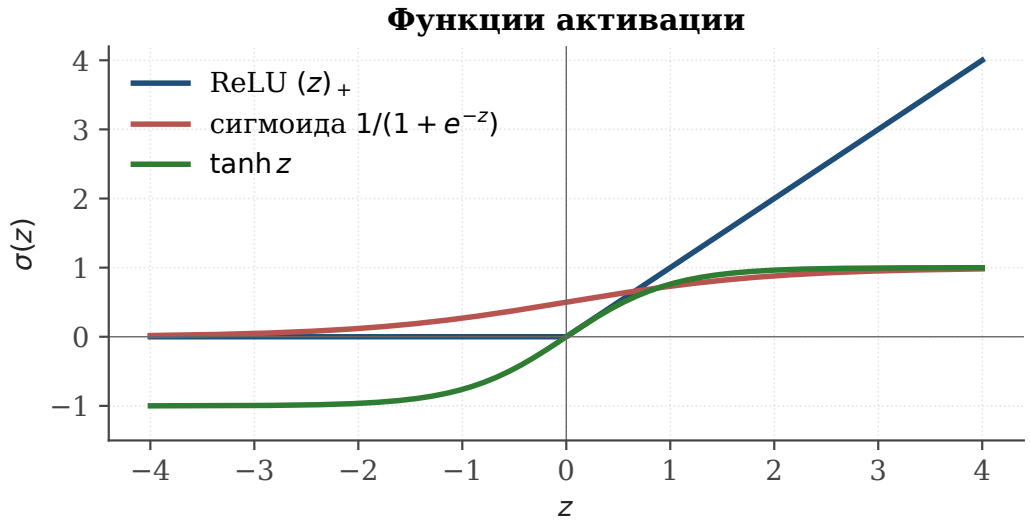


Рис. 4.15. Три распространённые функции активации. Сигмоида и гиперболический тангенс «насыщаются» при больших $|z|$ (производная стремится к нулю), что исторически затрудняло обучение глубоких сетей. ReLU $\sigma(z) = \max(0, z)$ — предложенная Хинтоном с соавторами в 2010 г. существенно ускорила обучение и сегодня является «рабочей лошадкой» во всех больших сетях.

Содержательная интерпретация. Без функций активации σ композиция линейных преобразований оставалась бы линейной (произведение матриц — снова матрица), и вся сеть свелась бы к простой линейной модели. *Именно нелинейность делает сеть выразительной* — позволяет ей описывать сложные, непрямолинейные зависимости.

Подсчёт числа параметров. Для сети $64 \to 32 \to 10$ (то есть один скрытый слой шириной 32):

$$\#\boldsymbol{\theta} = \underbrace{64 \cdot 32 + 32}_{\text{слой 1}} + \underbrace{32 \cdot 10 + 10}_{\text{слой 2}} = 2080 + 330 = 2410.$$

Сравните: датасет UCI Digits содержит около 1800 изображений, то есть параметров уже *больше*, чем обучающих примеров. Эта диспропорция — типичная для современных нейросетей — ставит важный вопрос о *переобучении*, к которому мы скоро вернёмся.

4.3.4 Обучение MLP и кривые потерь

Минимизация функционала (4.30) в нейронных сетях — это большая задача оптимизации в пространстве $\boldsymbol{\theta}$ размерности от тысяч до миллиардов. Аналитического решения нет (как было в МНК), поэтому применяют *численные методы* — прежде всего градиентный спуск. Подробно к нему мы вернёмся в разделе 4.3.8; пока примем как данность и посмотрим на поведение функции потерь и точности в процессе обучения.

Обучение MLP 64–32–10 на 200 примерах: cross-entropy и accuracy

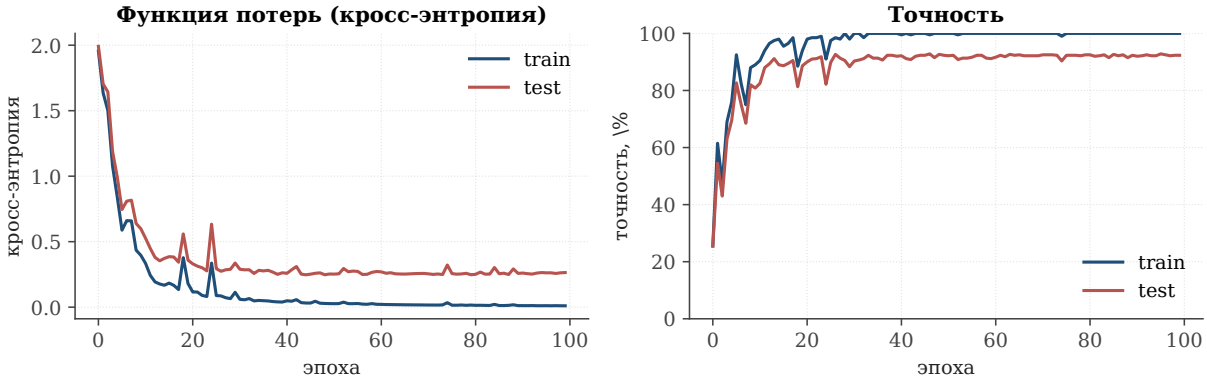


Рис. 4.16. Обучение MLP $64 \to 32 \to 10$ на 200 примерах из UCI Digits. Слева: кросс-энтропия по эпохам — на обучающей выборке (синяя) и на тестовой (красная). Справа: точность классификации, в %. Видно характерное поведение: после быстрого начального снижения потерь кривая «выходит на полку», и при этом разрыв между train и test небольшой — модель не переобучается.

В этом эксперименте модель достигает $\approx 92\,\%$ точности на тесте — вполне приличный результат для столь простой архитектуры.

4.3.5 Переобучение и U-образная кривая

Что произойдёт, если взять модель «помощнее»? Интуитивно: чем больше параметров, тем больше выразительная сила, тем лучше модель должна работать. Это *верно* до определённого предела — и *катастрофически неверно* после него. Это и называется **переобучением**.

Определение 4.13. Переобучение

Переобучение (англ. *overfitting*) — ситуация, когда модель отлично запоминает обучающую выборку (ошибка на train близка к нулю), но плохо работает на ранее не виденных данных (ошибка на test велика). Модель «выучила шум», а не закономерность.

Простейший пример — полиномиальная регрессия. Возьмём 12 точек, порождённых функцией $y = \sin 2\pi x$ плюс небольшой гауссов шум, и будем приближать их полиномами растущей степени.

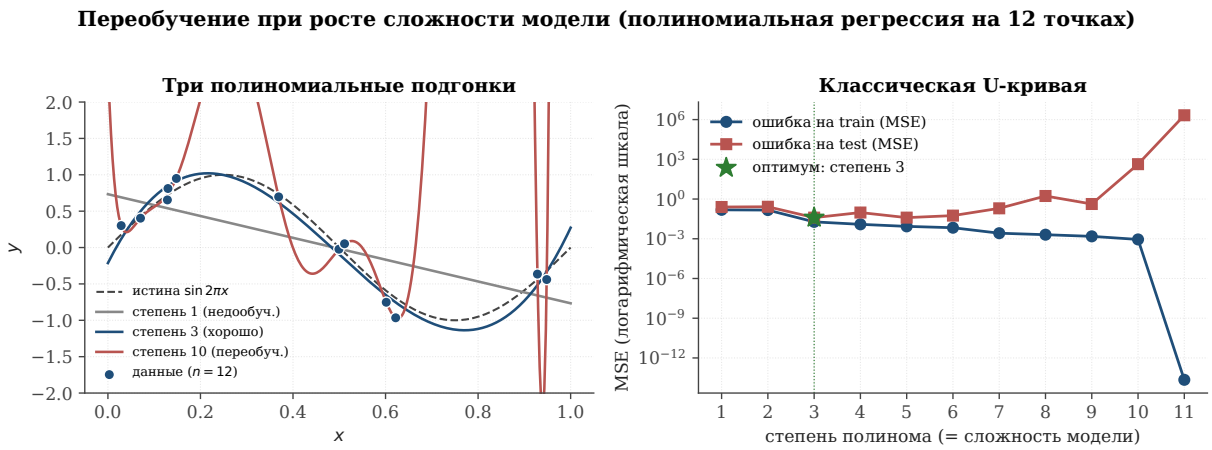


Рис. 4.17. Классическая U-образная кривая. *Слева*: три полиномиальные подгонки той же выборки из 12 точек. Степень 1 (прямая, серая) — *недообучение* (high bias): модель слишком простая. Степень 3 (синяя) — хороший компромисс, почти совпадает с истинной кривой $\sin 2\pi x$. Степень 10 (красная) проходит точно через все 12 точек, но между ними — безумно осциллирует: это *переобучение* (high variance). *Справа*: зависимость MSE от степени полинома. *Train-MSE* монотонно убывает (вплоть до нуля при степени 11 — полином идёт точно через 12 точек). А вот *test-MSE* ведёт себя по-U-образному: сначала падает (исчезает недообучение), а потом резко взлетает (начинается переобучение). Минимум — при степени 3.

Три выборки: train, validation, test. Стандартный приём борьбы с переобучением — разбить имеющиеся данные на *три* непересекающиеся части.

- **Обучающая (train)** — на ней непосредственно подбирают параметры $\boldsymbol{\theta}$ (веса сети).
- **Валидационная (validation)** — на ней подбирают *гиперпараметры*: глубину сети, ширину скрытого слоя, скорость обучения, степень полинома и т. п. Это *не* обучение в строгом смысле — мы лишь сравниваем разные варианты модели.
- **Контрольная (test)** — её модель видит *только один раз*, в самом конце, и только для того, чтобы честно оценить качество финальной модели. Если test-выборка используется неоднократно для подбора, она перестаёт быть «честной» — происходит так называемая *утечка* информации.

Типичные пропорции: 60–80 % на обучение, по 10–20 % на остальные две.

4.3.6 Теоремы об универсальной аппроксимации

Возникает естественный вопрос: *что в принципе может выучить нейронная сеть*? Может ли она приблизить произвольную непрерывную функцию — или есть классы зависимостей, недоступные ей в принципе?

Ответ дают два глубоких математических результата, открытые с разрывом в почти 30 лет.

Теорема 4.14. (Теорема Колмогорова–Арнольда (1957))
Всякая непрерывная функция $f\colon [0,1]^n \to \mathbb{R}$ от n переменных представима в виде

суперпозиции непрерывных функций *одной* переменной и операции сложения:

$$f(x_1, \ldots, x_n) \;=\; \sum_{q=0}^{2n} \Phi_q \left(\sum_{p=1}^{n} \psi_{q,p}(x_p) \right), \tag{4.35}$$

где Φ_q, $\psi_{q,p}$ — некоторые непрерывные функции одной переменной.

Эта теорема была доказана А. Н. Колмогоровым в 1957 г. и усилена В. И. Арнольдом — в виде, эквивалентном решению 13-й проблемы Гильберта (доказательство того, что любая непрерывная функция трёх переменных представима суперпозицией функций двух переменных). С точки зрения нейронных сетей утверждение (4.35) говорит: *двух слоёв и нескольких десятков аккуратно выбранных одномерных функций уже хватает для представления любой непрерывной функции от n переменных.* Это, конечно, теоретическое утверждение: функции $\Phi_q, \psi_{q,p}$ из доказательства Колмогорова чрезвычайно сложно устроены и непрактичны для прямого обучения. Но идейно теорема заложила фундамент.

Современная формулировка — с обычными функциями активации, такими как сигмоида или ReLU — была получена Дж. Цыбенко.

Теорема 4.15. (Теорема Цыбенко (1989))

Пусть $\sigma\colon \mathbb{R} \to \mathbb{R}$ — непрерывная, ограниченная, монотонно возрастающая функция (например, сигмоида). Тогда для любой непрерывной функции f на компакте $K \subset \mathbb{R}^n$ и любого $\varepsilon > 0$ существуют число N, числа $a_j, b_j, w_{j,i}$, $j = 1, \ldots, N$, $i = 1, \ldots, n$, такие что функция

$$g(\mathbf{x}) \;=\; \sum_{j=1}^{N} a_j \, \sigma\left(\sum_{i=1}^{n} w_{j,i}\, x_i + b_j \right) \tag{4.36}$$

приближает f равномерно на K с точностью ε: $\sup_{\mathbf{x}\in K} |g(\mathbf{x}) - f(\mathbf{x})| < \varepsilon$.

Иначе говоря, *одного скрытого слоя достаточной ширины* достаточно, чтобы приблизить любую непрерывную функцию с заданной точностью. Это **теорема об универсальной аппроксимации** (рис. 4.18).

Теорема Цыбенко требует *очень широких* сетей: чтобы достичь точности ε, ширина N может расти экспоненциально с размерностью n или с $1/\varepsilon$. На практике куда выгоднее *глубокие* сети — те, что используют несколько слоёв средней ширины. Это утверждение было математически уточнено в работах 2010-х годов: для некоторых классов функций глубокая сеть требует *экспоненциально меньше* параметров, чем эквивалентная по точности неглубокая.

4.3.7 Свёрточные нейронные сети

Полносвязная сеть «не знает» о том, что её вход — это изображение. Она одинаково обработала бы цифру и любую случайную перестановку её 64 пикселей. Это и теоретически, и практически — неоптимально: *в изображении соседние пиксели связаны* (вместе образуют чёрточки, дуги, кружочки — из которых и состоит цифра), а далёкие — почти независимы.

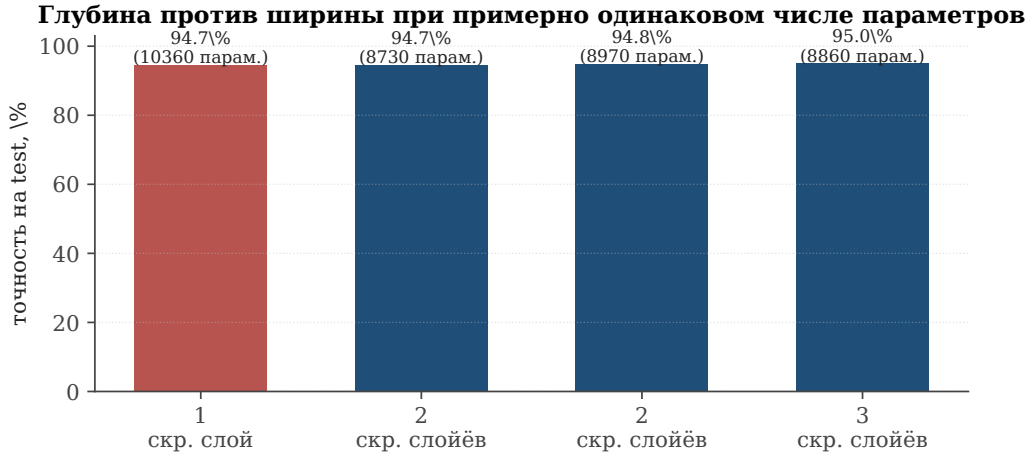


Рис. 4.18. Эмпирическое сравнение на UCI Digits: четыре MLP с примерно *одинаковым* числом параметров (~ 9000), но разной *глубиной* — 1, 2, 3 и 4 скрытых слоя. При фиксированном бюджете параметров глубокие модели работают чуть лучше неглубоких. Разница невелика (UCI Digits — простая задача), но на сложных задачах она составляет уже десятки процентов: на классификации ImageNet переход от 8-слойной AlexNet (2012) к 152-слойной ResNet (2015) снизил ошибку с 16 % до 3 %.

Идея использовать локальные «свёрточные» фильтры пришла из биологии. В 1962 г. нейрофизиологи Дэвид Хьюбел и Торстен Визель показали, что зрительная кора кошки устроена иерархически: на первом уровне — нейроны, реагирующие на *простые признаки* (края, отрезки прямых под определённым углом); на следующих — на их комбинации (углы, дуги, контуры); и так до целых форм. За это открытие они получили Нобелевскую премию по физиологии в 1981 г.

В 1980 г. японский исследователь Кунихико Фукусима реализовал эту идею в виде *неокогнитрона* — иерархической сети для распознавания символов. А в 1989 г. Янн ЛеКун — тогда ещё аспирант у Хинтона — добавил к неокогнитрону обучение методом обратного распространения и предложил универсальную архитектуру **свёрточной нейронной сети** (*Convolutional Neural Network*, CNN). Его сеть LeNet-5 (1998) распознавала рукописные цифры с точностью 99 % и применялась для чтения почтовых индексов и банковских чеков.

В 2012 г. работа Алекса Крижевского, Ильи Суцкевера и Джеффри Хинтона *ImageNet Classification with Deep Convolutional Neural Networks* (сеть AlexNet) продемонстрировала, что свёрточные сети, обученные на графических процессорах (GPU), могут с разгромным счётом превзойти все предыдущие подходы в задаче классификации изображений. С этого момента и началась «революция глубокого обучения».

Что такое свёртка. Рассмотрим изображение X размера $H \times W$ и небольшой *фильтр* K размера $k \times k$ (обычно $k = 3$ или 5). **Свёрткой** изображения с фильтром называется новое изображение Y, в каждой точке которого стоит «скалярное произведение» фильтра и маленького окошка изображения:

$$Y_{r,c} \;=\; \sum_{u=1}^{k} \sum_{v=1}^{k} K_{u,v}\, X_{r+u-1,\,c+v-1}. \tag{4.37}$$

Фильтр «скользит» по всему изображению; в итоге получается новая карта признаков, чувствительная к тому, что «видит» фильтр (см. рис. 4.19 — четыре фильтра, обученные на UCI Digits, очевидно выделяют локальные «уголки» и «полосы»).

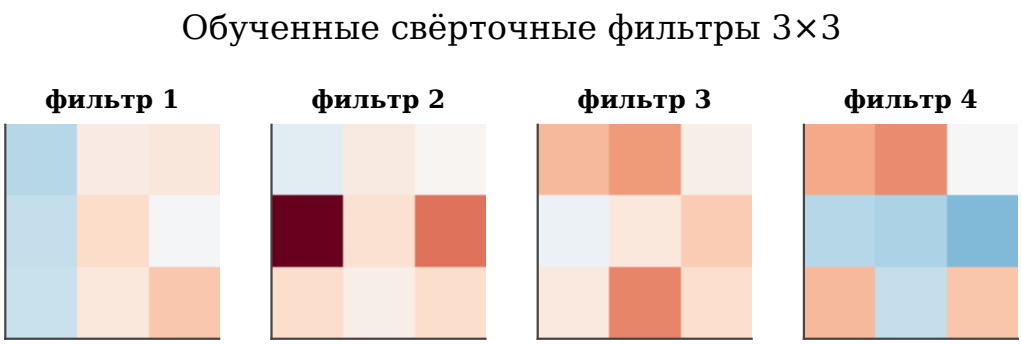


Рис. 4.19. Четыре обученных свёрточных фильтра 3×3 из простой CNN, обученной на UCI Digits. Тёмно-красные клетки — большие положительные веса, синие — большие отрицательные. Каждый фильтр «заточен» под свой локальный признак (наклонная линия, перепад яркости в определённом направлении и т. д.).

Определение 4.16. Свёрточная нейронная сеть (CNN)

Свёрточная нейронная сеть — это нейронная сеть, в которой некоторые слои представляют собой не полносвязные матрично-векторные умножения, а *свёртки* вида (4.37) с обучаемыми фильтрами K. Обычно после свёртки применяется функция активации и операция *пулинга* (выборки максимума по окрестности — для уменьшения разрешения). После нескольких свёрточных слоёв полученная «карта признаков» разворачивается в вектор и передаётся в обычный полносвязный слой с softmax.

Сравнение MLP и CNN на UCI Digits. Чтобы оценить выигрыш от свёрточной структуры, обучим две сети с примерно одинаковым числом параметров: обычную MLP и CNN с одним свёрточным слоем. Результат показан на рис. 4.20.

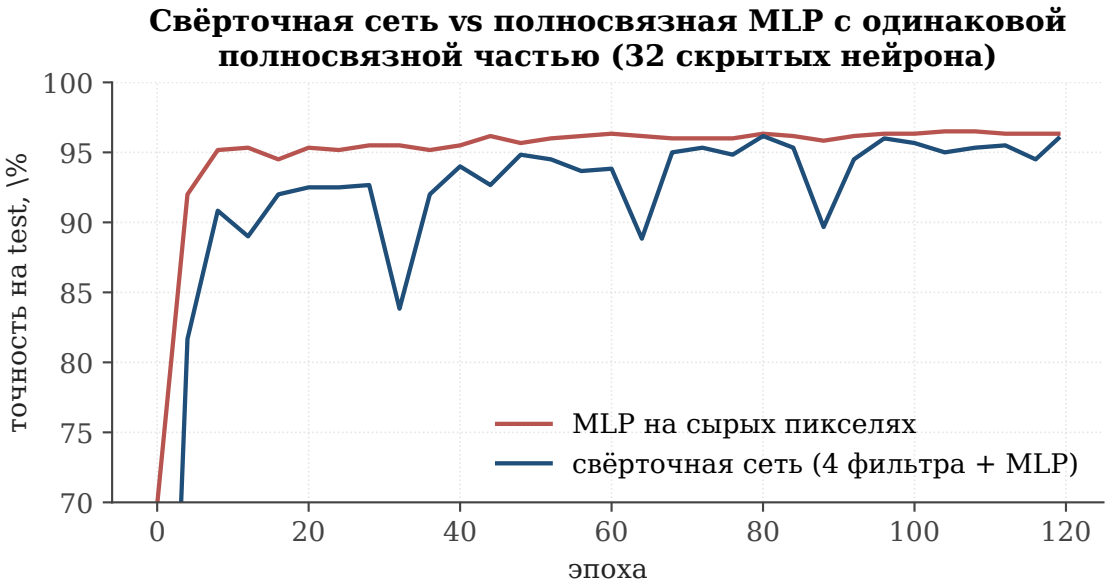


Рис. 4.20. Сравнение точности на тесте для полносвязной MLP и простой CNN с теми же финальными скрытыми слоями. На крошечных изображениях 8×8 из UCI Digits разница невелика — задача слишком проста, локальная структура почти не помогает. На полноразмерном MNIST 28×28 CNN превосходит MLP на 1–2 процентных пункта; на цветных изображениях ImageNet (224×224) — уже на 10–15 процентных пунктов.

4.3.8 Градиент и градиентный спуск

Перейдём к ключевому вопросу: *как же подобрать параметры $\boldsymbol{\theta}$, минимизирующие функционал* (4.30)? Аналитическое решение, как было в МНК (раздел 4.2.4), здесь невозможно: функция $\mathcal{L}(\boldsymbol{\theta})$ очень сложная, многомерная, нелинейная. Поэтому применяют итерационные численные методы, и важнейший из них — **градиентный спуск**.

Производная по направлению и градиент

Пусть $F\colon \mathbb{R}^d \to \mathbb{R}$ — дифференцируемая функция (в нашем случае $F = \mathcal{L}$, а d — число параметров сети). В точке $\boldsymbol{\theta}$ зададим какое-нибудь направление $\mathbf{v} \in \mathbb{R}^d$ единичной длины. **Производной функции F в точке $\boldsymbol{\theta}$ по направлению $\mathbf{v}$** называется

$$\partial_{\mathbf{v}} F(\boldsymbol{\theta}) \;=\; \lim_{t \to 0} \frac{F(\boldsymbol{\theta} + t\mathbf{v}) - F(\boldsymbol{\theta})}{t}.$$

Чтобы вычислить её, рассмотрим функцию одной переменной $\varphi(t) = F(\boldsymbol{\theta} + t\mathbf{v})$ и применим правило дифференцирования сложной функции:

$$\partial_{\mathbf{v}} F \;=\; \varphi'(0) \;=\; \sum_{j=1}^{d} \frac{\partial F}{\partial \theta_j} \cdot v_j \;=\; \langle \nabla F,\, \mathbf{v} \rangle, \tag{4.38}$$

где $\nabla F = \bigl(\partial F/\partial \theta_1, \dots, \partial F/\partial \theta_d\bigr)$ — **градиент** функции F в точке $\boldsymbol{\theta}$, а $\langle \cdot, \cdot \rangle$ — скалярное произведение в $\mathbb{R}^d$.

Теорема 4.17. (Антиградиент — направление наискорейшего спуска)

Среди всех единичных векторов $\mathbf{v} \in \mathbb{R}^d$ производная $\partial_{\mathbf{v}} F(\boldsymbol{\theta})$ принимает *минимальное* значение при

$$\mathbf{v}^{\star} \;=\; -\frac{\nabla F(\boldsymbol{\theta})}{\|\nabla F(\boldsymbol{\theta})\|}. \tag{4.39}$$

При этом $\partial_{\mathbf{v}^\star} F = -\,\|\nabla F(\boldsymbol{\theta})\|$.

Доказательство. По неравенству Коши–Буняковского $\langle \nabla F, \mathbf{v} \rangle \geq -\,\|\nabla F\| \cdot \|\mathbf{v}\| = -\,\|\nabla F\|$, с равенством в точности тогда, когда $\mathbf{v}$ *сонаправлен с* $-\nabla F$. □

Этот простой факт — сердце всей теории численной оптимизации. *Локально, для уменьшения функции F, надо двигаться в сторону антиградиента.* Именно так устроен **метод градиентного спуска**:

$$\boxed{\boldsymbol{\theta}_{k+1} \;=\; \boldsymbol{\theta}_k \;-\; \alpha\, \nabla F(\boldsymbol{\theta}_k)} \tag{4.40}$$

Здесь $\alpha > 0$ — *шаг обучения* (*learning rate*), гиперпараметр. Слишком маленький — сходимость медленная; слишком большой — итерации могут «перескакивать» через минимум и расходиться (рис. 4.21).

Стохастический градиентный спуск (SGD)

Когда обучающая выборка велика, вычислять полный градиент $\nabla \mathcal{L}(\boldsymbol{\theta}) = \frac{1}{N}\sum_{i=1}^{N} \nabla \ell_i(\boldsymbol{\theta})$ (где $\ell_i = -\ln p_{y_i}(\mathbf{x}_i; \boldsymbol{\theta})$ — ошибка на i-м примере) на каждом шаге — очень дорого. Поэтому используют **стохастический градиентный спуск** (SGD): на каждой итерации случайно выбирается *батч* (англ. *batch*, «пачка») — подмножество $B_k \subset \{1, 2, \dots, N\}$ из m примеров, — и шаг делается по *приближённому* градиенту, вычисленному только по этому батчу:

$$\boldsymbol{\theta}_{k+1} \;=\; \boldsymbol{\theta}_k \;-\; \alpha\, \widehat{\nabla \mathcal{L}}(\boldsymbol{\theta}_k), \qquad \widehat{\nabla \mathcal{L}} = \frac{1}{m} \sum_{i \in B_k} \nabla \ell_i. \tag{4.41}$$

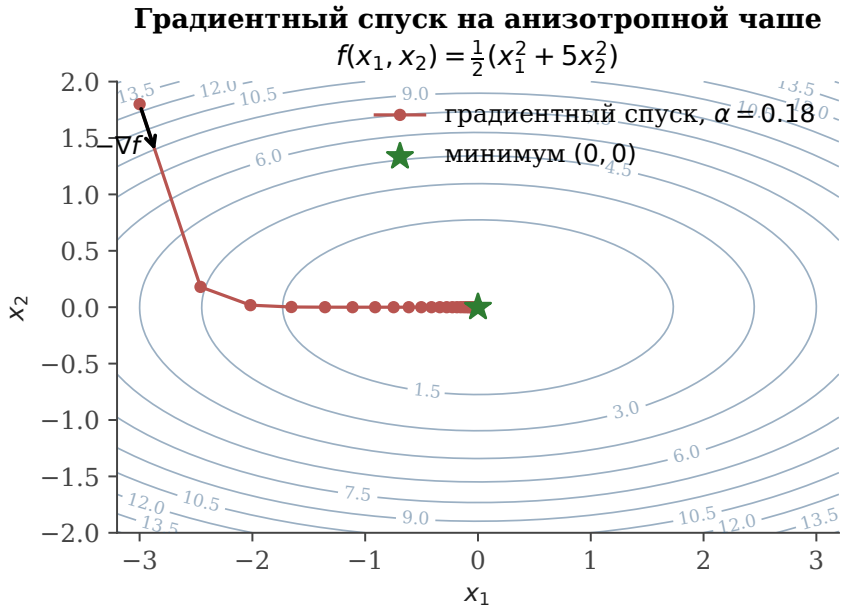


Рис. 4.21. Траектория градиентного спуска для квадратичной функции $F(x_1, x_2) = \frac{1}{2}(x_1^2 + 5x_2^2)$. Линии уровня — эллипсы (функция «вытянута» вдоль оси x_1). Метод стартует из точки $(-3, 2)$ и за 20 итераций приходит к минимуму в $(0, 0)$. Видно характерное «зигзагообразное» движение, типичное для плохо обусловленных задач.

Здесь B_k — случайное подмножество индексов («мини-батч» на шаге k), $|B_k| = m$, обычно $m \in \{32, 64, 128, 256\}$. Такой приближённый градиент — несмещённая оценка истинного градиента (в духе ОМП!), но вычисляется в N/m раз быстрее.

Дополнительный бонус: «шум» от случайного выбора батча помогает итерациям выбираться из неглубоких локальных минимумов и седловых точек. Поэтому SGD — де-факто стандартный оптимизатор глубоких сетей.

4.3.9 Метод обратного распространения ошибки

Чтобы реализовать градиентный спуск, надо уметь *вычислять градиент* функции потерь $\mathcal{L}$ по параметрам $\boldsymbol{\theta}$. В нейронной сети это нетривиально: параметры $\boldsymbol{\theta}$ запрятаны в матрицы $W^{(\ell)}$ нескольких слоёв, и каждая участвует в формуле потерь через цепочку преобразований

$$\mathbf{x} \to \mathbf{z}^{(1)} \to \mathbf{a}^{(1)} \to \mathbf{z}^{(2)} \to \ldots \to \mathbf{p} \to \mathcal{L}.$$

Прямой подсчёт частных производных «в лоб» — через определение производной как предела — потребовал бы для каждого параметра *отдельного* перевычисления всей сети. Для сети с миллионом параметров это означало бы миллион полных прогонов вместо одного — полностью неприемлемо.

Обратное распространение ошибки (*backpropagation*) — алгоритм, который вычисляет градиент *сразу по всем параметрам* с вычислительной стоимостью, сравнимой со стоимостью одного прогона функции. Идея алгоритма — системное применение цепного правила дифференцирования.

> Backpropagation — классический пример переоткрытия в науке. Идея многократно появлялась с 1960-х:
>
> - **Сеппо Линнайнмаа** (1970) — финский магистр, в диссертации описал общий алгоритм автоматического дифференцирования в обратном режиме.

- **Ким Юрий Геннадьевич, Нестеров Юрий Евгеньевич, Скоков Александр, Черкасский Виктор** — работы начала 1980-х в СССР: в общей форме строго установлен фундаментальный результат о том, что вычислительная сложность подсчёта градиента дифференцируемой функции, заданной программой, ограничена сверху константой, умноженной на сложность вычисления самой функции. Этот результат — математическая основа того, что современные «миллиарды параметров» вообще возможно обучать.
- **Поль Вербос** (1974, докторская диссертация Гарварда) и **Дэвид Румельхарт, Джеффри Хинтон, Рональд Уильямс** (1986) — именно последняя работа в журнале *Nature* популяризовала backpropagation в применении к обучению многослойных нейронных сетей.

Сегодня backpropagation реализован в каждой библиотеке глубокого обучения (PyTorch, TensorFlow, JAX) как *automatic differentiation*.

Идея алгоритма для MLP. Рассмотрим сеть из определения 4.12. Ради прозрачности выкладок ограничимся одним обучающим примером $(\mathbf{x}, y)$; функция потерь $\ell = -\ln p_y$.

Прямой проход. Сначала вычисляем все промежуточные значения $\mathbf{z}^{(\ell)}, \mathbf{a}^{(\ell)}$ ровно так, как они определены в формулах (4.32), (4.33). Это — один обычный прогон сети.

Обратный проход. Хотим вычислить градиенты потерь по всем параметрам. Для ℓ-го слоя обозначим

$$\boldsymbol{\delta}^{(\ell)} \stackrel{\text{def}}{=} \frac{\partial \ell}{\partial \mathbf{z}^{(\ell)}} \in \mathbb{R}^{d_\ell}. \tag{4.42}$$

Это вспомогательный вектор; его называют «*ошибкой ℓ-го слоя*». Алгоритм состоит из трёх формул, получаемых строгим применением цепного правила:

(B1) Ошибка на последнем слое. Для пары «softmax + кросс-энтропия» прямой подсчёт даёт красивую формулу:

$$\boldsymbol{\delta}^{(L)} = \mathbf{p} - \mathbf{e}_y, \tag{4.43}$$

где $\mathbf{e}_y$ — one-hot вектор истины (1 в позиции y, 0 в остальных). То есть ошибка — это просто разница между предсказанным распределением вероятностей и истинным.

(B2) Распространение ошибки назад по слоям. Связь между ошибками двух соседних слоёв даётся формулой

$$\boldsymbol{\delta}^{(\ell-1)} = \big(W^{(\ell)}\big)^{\top} \boldsymbol{\delta}^{(\ell)} \odot \sigma'\big(\mathbf{z}^{(\ell-1)}\big), \tag{4.44}$$

где $\odot$ — покомпонентное произведение векторов, а σ' применяется покомпонентно. Откуда она берётся? Распишем цепное правило для одного перехода между слоями: $\mathbf{z}^{(\ell)} = W^{(\ell)}\mathbf{a}^{(\ell-1)} + \mathbf{b}^{(\ell)}$ и $\mathbf{a}^{(\ell-1)} = \sigma(\mathbf{z}^{(\ell-1)})$. Тогда

$$\frac{\partial \ell}{\partial z_j^{(\ell-1)}} = \sum_i \frac{\partial \ell}{\partial z_i^{(\ell)}} \cdot \frac{\partial z_i^{(\ell)}}{\partial a_j^{(\ell-1)}} \cdot \frac{\partial a_j^{(\ell-1)}}{\partial z_j^{(\ell-1)}} = \sum_i \delta_i^{(\ell)} \cdot W_{i,j}^{(\ell)} \cdot \sigma'\big(z_j^{(\ell-1)}\big).$$

Собирая всё по j в один вектор, получаем именно (4.44). Это и есть «*распространение ошибки назад*»: ошибка более глубокого слоя ℓ «протаскивается» через транспонированную матрицу весов в более ранний слой $\ell - 1$.

(B3) Градиенты по параметрам. Зная $\boldsymbol{\delta}^{(\ell)}$, градиенты по параметрам ℓ-го слоя выписываются сразу:

$$\frac{\partial \ell}{\partial W^{(\ell)}} = \boldsymbol{\delta}^{(\ell)} \big(\mathbf{a}^{(\ell-1)}\big)^{\top}, \tag{4.45}$$

$$\frac{\partial \ell}{\partial \mathbf{b}^{(\ell)}} = \boldsymbol{\delta}^{(\ell)}. \tag{4.46}$$

(Первое — матрица того же размера, что $W^{(\ell)}$; второе — вектор.)

Стоимость алгоритма. Каждая из формул (4.44), (4.45) — это одно матрично-векторное умножение, по стоимости такое же, как и прямой проход для того же слоя. *Итого: полный обратный проход стоит не дороже одного прямого* — то есть мы получаем все градиенты по *всем* параметрам ценой одного дополнительного прохода. Это и есть тот фундаментальный результат, который в общей форме был установлен в работах Кима–Нестерова–Скокова–Черкасского начала 1980-х: для любой дифференцируемой функции, заданной программой,

$$\boxed{\text{стоимость подсчёта градиента} \;\le\; C \cdot (\text{стоимость подсчёта функции}),}$$

с константой $C \le 5$ в наихудшем случае. Без этого результата обучение современных сетей с миллиардами параметров было бы вычислительно невозможно.

> В практике глубокого обучения один из любимых рецептов отладки backpropagation — *численная проверка градиента*:
>
> $$\frac{\partial \ell}{\partial \theta_j} \;\approx\; \frac{\ell(\theta_j + h) - \ell(\theta_j - h)}{2h}, \qquad h \sim 10^{-5}.$$
>
> Это очень дорого (по одному вычислению функции на каждый параметр), поэтому делается только на десяти-двадцати случайных параметрах маленьких сетей, и только при отладке. Совпадение с аналитическим градиентом до 6-го знака означает, что backprop реализован правильно.

4.3.10 Резюме параграфа

Мы прошли длинный путь от формулировки задачи классификации до полноценного «рецепта» обучения нейронной сети. Ключевые идеи:

1. **Кросс-энтропия** — естественный функционал для задач классификации, выводимый из принципа максимума правдоподобия (как МНК выводился из ОМП для гауссова шума).
2. **Нейронная сеть** — параметризованное семейство функций в виде композиции линейных и нелинейных слоёв. *Теоремы Колмогорова–Арнольда и Цыбенко* гарантируют, что это семейство достаточно богато, чтобы приблизить любую непрерывную функцию.
3. **Переобучение** — ключевая опасность: с ростом сложности модели train-ошибка убывает монотонно, а test-ошибка ведёт себя по U-образной кривой. Для борьбы с этим выделяют *валидационную* и *контрольную* выборки.
4. **Свёрточные сети** учитывают пространственную структуру изображения и работают лучше полносвязных там, где локальность существенна.
5. **Градиентный спуск** и его стохастический вариант — универсальный численный метод. Двигаться надо в сторону *антиградиента*; обоснование — неравенство Коши–Буняковского.
6. **Backpropagation** — вычислительно эффективная реализация градиента для нейронных сетей; работает за время, сопоставимое с одним прямым проходом сети.

В следующем параграфе мы перейдём к задачам *обучения без учителя*, в которых правильные ответы y_i для обучающих объектов *не известны*. Удивительным образом и там удастся применить часть тех же идей.

Задачи для самостоятельной работы

1. Покажите формулу (4.43): вычислите производную кросс-энтропии $\ell = -\ln p_y = -\ln \frac{e^{z_y}}{\sum_c e^{z_c}}$ по z_c и убедитесь, что она равна $p_c - [c = y]$.
2. Сколько обучаемых параметров в сети с архитектурой $784 \to 256 \to 128 \to 10$ (стандартная MLP для полного MNIST)? А если добавить ещё один скрытый слой шириной 64?
3. Покажите, что произведение двух матриц A и B можно определить так, чтобы для любого вектора $\mathbf{v}$ выполнялось $(AB)\mathbf{v} = A(B\mathbf{v})$. Какое тогда условие на размеры A и B?
4. Реализуйте на Python градиентный спуск для функции $F(x, y) = \frac{1}{2}(x^2 + 10y^2)$. При каких значениях шага α метод сходится? При каком из них — быстрее всего?
5. Запрограммируйте простейшую полносвязную сеть с одним скрытым слоем на UCI Digits и обучите её. Постройте кривые потерь и точности — как на рис. 4.16.
6. ⋆ Возьмите 12 точек (x_i, y_i) из любой гладкой функции (например, $y = \sin x$ при $x \in [0, 2\pi]$) с добавленным гауссовым шумом. Постройте полиномы степеней $1, 2, \ldots, 11$ и нарисуйте U-кривую, как на рис. 4.17.
7. ⋆ Докажите формулу (4.44) полностью, начиная с цепного правила $\partial\ell/\partial z_j^{(\ell-1)} = \sum_k (\partial\ell/\partial z_k^{(\ell)}) \cdot (\partial z_k^{(\ell)}/\partial z_j^{(\ell-1)})$.

4.4 Задачи обучения без учителя

Во всех задачах предыдущих параграфов каждому обучающему объекту $\mathbf{x}_i$ сопутствовал *правильный ответ* y_i: голос за кандидата, период обращения планеты, рукописная цифра. Такая задача называется обучением **с учителем** (*supervised learning*): учителем выступает разметка, кем-то заранее проставленная.

На практике разметка часто отсутствует. У онлайн-кинотеатра — терабайты данных о просмотрах и оценках, но никто не объяснил алгоритму, что «Иван — любитель боевиков, а Маша — драм». У поисковой системы — миллиарды веб-страниц, но никто не пометил, какие из них «про физику», а какие «про поэзию». В таких ситуациях говорят об обучении **без учителя** (*unsupervised learning*). Цель — самостоятельно обнаружить структуру в данных, выделить «главные оси», на которых эти данные расположены, и научиться компактно представлять каждый объект.

Этот параграф устроен так. Мы разберём два центральных сюжета — *восстановление матриц по неполным данным* (matrix completion, она же ставшая знаменитой Netflix problem) и *автокодировщики*. Затем покажем, что оба сводятся к универсальной алгебраической конструкции — **сингулярному разложению матриц (SVD)**. В завершении обсудим эмбеддинги и контрастивное обучение, которые лежат в основе современных систем распознавания лиц, поиска и больших языковых моделей.

4.4.1 Задача восстановления матрицы. Netflix problem

В октябре 2006 г. американская компания **Netflix** — тогда ещё сервис проката DVD по почте — объявила публичный конкурс с призом в 1 миллион долларов: первый, кто построит алгоритм, угадывающий оценки пользователей хотя бы на 10 % точнее, чем собственный алгоритм Netflix, получит всю сумму.
Компания выложила открытый датасет: 480 189 пользователей, 17 770 фильмов и ≈

100 млн оценок (по 5-балльной шкале). Каждая оценка сопровождалась датой; нужно было предсказывать неизвестные оценки.
Конкурс длился почти три года. Победила команда BellKor's Pragmatic Chaos (объединение исследователей из AT&T, Yahoo, Big Chaos и Pragmatic Theory), и выиграла она 21 сентября 2009 г. с разницей в *20 минут* над командой Ensemble. Сердцевиной победившего метода была именно **малоранговая матричная факторизация**, о которой мы сейчас расскажем. Этот сюжет принципиально изменил индустрию рекомендательных систем — сегодня описанные подходы лежат в основе «вам может понравиться» в Кинопоиске, Окко, YouTube, Spotify, Amazon и сотнях других сервисов.

Формальная постановка. Пусть имеется m пользователей и n фильмов. Полная информация об оценках — это матрица $A \in \mathbb{R}^{m\times n}$:

$$A_{ij} \;=\; \text{оценка, которую пользователь } i \text{ поставил фильму } j.$$

Однако в реальности *большинство* элементов этой матрицы неизвестны — средний пользователь Netflix посмотрел и оценил лишь несколько десятков фильмов из почти 18 тысяч. Обозначим

$$\Omega \;\subset\; \{1,\dots,m\}\times\{1,\dots,n\}$$

множество *наблюдённых* пар (i,j). Задача восстановления матрицы по неполным данным: по известным A_{ij}, $(i,j)\in\Omega$, восстановить значения A_{ij} для $(i,j)\notin\Omega$ (рис. 4.22).

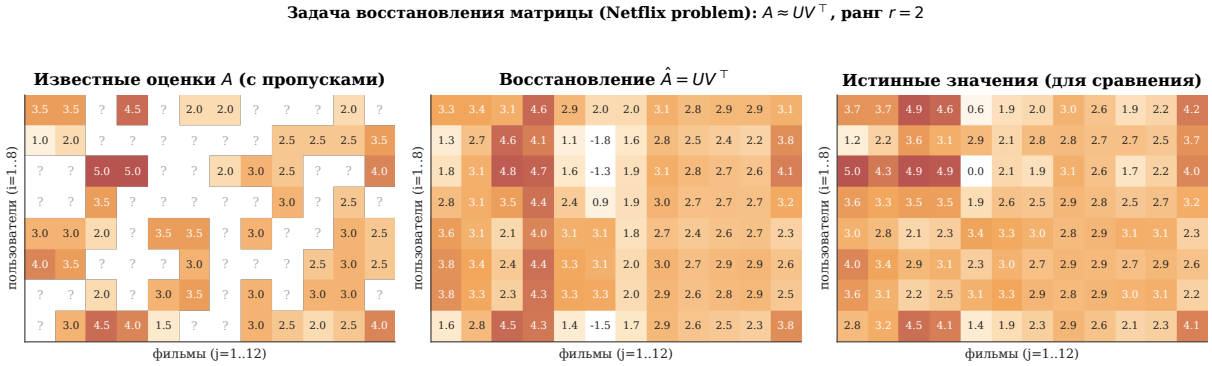


Рис. 4.22. Иллюстрация задачи Netflix на маленьком примере: 8 пользователей, 12 фильмов, ранг $r=2$. Слева — известные оценки (большинство клеток помечены '?'). В центре — восстановление $\hat{A}=UV^\top$ с r-факторизацией (*все* клетки теперь заполнены, в том числе неизвестные). Справа — истинные значения (для проверки). Метод не просто заполняет пропуски, но и обнаруживает *скрытые* (латентные) закономерности в данных.

Идея: малоранговая факторизация. Без дополнительных предположений задача безнадёжна: восстановить неизвестное число можно как угодно. Но если предположить, что есть *скрытые признаки* — например, «насколько фильм комедийный», «насколько драматичный», «сколько в нём действия» — и описать каждого пользователя i парой чисел, отражающей его пристрастия, а каждый фильм j — такой же парой чисел, то оценка естественным образом получается как «скалярное произведение пристрастий и свойств».

Формализуем. Пусть r — небольшое число (например, $r=2$, 5 или 20). Введём

- матрицу $U\in\mathbb{R}^{m\times r}$: i-я её строка $\mathbf{u}_i$ — вектор пристрастий пользователя i;
- матрицу $V\in\mathbb{R}^{n\times r}$: j-я её строка $\mathbf{v}_j$ — вектор свойств фильма j.

Тогда модель оценок:

$$\boxed{A_{ij} \;\approx\; \langle \mathbf{u}_i,\, \mathbf{v}_j \rangle \;=\; \sum_{k=1}^{r} u_{ik}\, v_{jk},} \qquad \text{или в матричной форме} \qquad A \;\approx\; U\, V^{\top}. \tag{4.47}$$

Сколько параметров в модели? В исходной матрице A — $m \cdot n$ чисел (например, $480\,189 \cdot 17\,770 \approx 8{,}5$ млрд). В факторизованной модели — $m \cdot r + n \cdot r = (m+n)r$ чисел (для $r = 20$, $m \approx 5 \cdot 10^5$, $n \approx 2 \cdot 10^4$ — около 10^7, то есть на *три порядка* меньше). Если истинная структура данных — действительно малоранговая, то этой существенно меньшей информации достаточно для восстановления.

> *Рангом* матрицы A называется наименьшее r, при котором A можно представить в виде $A = UV^{\top}$ с матрицами размера $m \times r$ и $n \times r$ соответственно. Эквивалентное определение: ранг r — это максимальное число линейно независимых столбцов (а также строк) матрицы. Чем ниже ранг, тем больше избыточности (повторяющихся закономерностей) в данных — и тем более «сжимаема» матрица.

Постановка как задачи оптимизации. Параметры $\mathbf{u}_i$, $\mathbf{v}_j$ подбираются так, чтобы модель наилучшим образом соответствовала наблюдённым данным:

$$\sum_{(i,j)\in\Omega} \bigl(A_{ij} - \langle \mathbf{u}_i,\, \mathbf{v}_j \rangle\bigr)^2 \;\longrightarrow\; \min_{U,V}. \tag{4.48}$$

Это снова метод наименьших квадратов — но теперь сумма квадратов *только по известным* элементам матрицы. Если интерпретировать данные как «истинные оценки плюс гауссов шум», функционал (4.48) получается из принципа максимума правдоподобия — ровно тем же выводом, что в раздел 4.2.2.

> Этот пример хорош тем, что наглядно показывает водораздел. *В задачах математической статистики* (как в раздел 4.1, раздел 4.2) распределение данных *с точностью до неизвестных параметров* известно, и функционал ошибки выводится из принципа максимума правдоподобия.
> *В задачах машинного обучения и ИИ* распределение чаще всего неизвестно; функционал ошибки выбирают исходя из здравого смысла. В нашем примере — квадратичная невязка $(A_{ij} - \langle \mathbf{u}_i, \mathbf{v}_j \rangle)^2$ — это естественный выбор, но он *не выводится* из строгой модели порождения данных. Можно, конечно, постулировать, что $A_{ij} = \langle \mathbf{u}_i, \mathbf{v}_j \rangle + \xi_{ij}$ с $\xi_{ij} \sim \mathcal{N}(0, \sigma^2)$, и тогда квадраты возникают из ОМП — но это уже не вывод из данных, а наше *дополнительное предположение* о структуре шума. Кроме того, в ИИ часто используются нелинейные модели — нейронные сети как универсальные аппроксиматоры.

Неотрицательность. В практической рекомендательной системе оценки лежат в шкале от 1 до 5, а пристрастия и свойства принципиально «качественные»: фильм скорее *в большей или меньшей степени* принадлежит жанру, чем «в антинаправленной». Поэтому естественно ограничиться *неотрицательной* малоранговой факторизацией:

$$A \;\approx\; UV^{\top}, \qquad U \geq 0,\; V \geq 0, \tag{4.49}$$

где неравенство понимается покомпонентно. Такая факторизация называется **неотрицательным матричным разложением** (non-negative matrix factorization, **NMF**); её свойства глубоко изучали Д. Ли и Х. Сын в работах 1999 г.

Пример 4.18. Маленький конкретный пример

Рассмотрим частично заполненную матрицу 4 пользователей и 5 фильмов (оценки от 1 до 5, '?' — неизвестно):

$$A = \begin{pmatrix} 5 & 4 & ? & 1 & ? \\ ? & 5 & 5 & ? & 2 \\ 1 & ? & 2 & 5 & 5 \\ ? & 1 & 1 & 4 & 5 \end{pmatrix}.$$

Видна закономерность: первые два пользователя любят первые три фильма; третий и четвёртый — последние два. Если попытаться найти факторизацию $A \approx UV^\top$ с рангом $r = 2$:

$$U = \begin{pmatrix} 1{,}9 & 0{,}1 \\ 1{,}9 & 0{,}2 \\ 0{,}1 & 1{,}7 \\ 0{,}0 & 1{,}7 \end{pmatrix}, \qquad V = \begin{pmatrix} 2{,}6 & 0{,}5 \\ 2{,}3 & 0{,}3 \\ 2{,}5 & 0{,}5 \\ 0{,}6 & 2{,}8 \\ 0{,}4 & 3{,}0 \end{pmatrix},$$

то $UV^\top$ почти точно воспроизводит наблюдённые элементы и «предсказывает» неизвестные:

$$UV^\top \approx \begin{pmatrix} 5{,}0 & 4{,}4 & \boxed{4{,}8} & 1{,}4 & \boxed{1{,}1} \\ \boxed{5{,}1} & 4{,}4 & 4{,}9 & \boxed{1{,}7} & 1{,}4 \\ \boxed{1{,}1} & \boxed{0{,}7} & 1{,}1 & 4{,}8 & 5{,}1 \\ 0{,}0 & 0{,}5 & 0{,}9 & 4{,}8 & 5{,}1 \end{pmatrix}.$$

Обведены восстановленные элементы. Видно: пользователи 1 и 2 в этой модели — любители «жанра 1» (первая координата пристрастий велика); пользователи 3 и 4 — любители «жанра 2». А столбцы V показывают, какому жанру какие фильмы соответствуют.

Алгоритм решения. Задача (4.48) *не* имеет аналитического решения и относится к нелинейной невыпуклой оптимизации. На практике её решают методом **переменных наименьших квадратов** (Alternating Least Squares, ALS): фиксируем V и оптимизируем U (это уже выпуклая задача и решается по формулам линейной регрессии), потом фиксируем U и оптимизируем V. Чередуем до сходимости. Для очень больших матриц используется стохастический градиентный спуск (раздел 4.3.8).

4.4.2 Три кита и роль малоранговой аппроксимации в ИИ

Разобранный пример хорошо демонстрирует обещание начала параграфа: современный анализ данных стоит на трёх китах.

Вероятностный кит. Хотя ни одного распределения мы явно не вводили, постановка (4.48) имеет естественную вероятностную интерпретацию — «истинные значения плюс гауссов шум». Эта вероятностная подкладка — то, что превращает расплывчатое «найти хорошее восстановление» в строгую задачу.

Оптимизационный кит. Полученная задача оптимизации (4.48) — невыпуклая нелинейная, она требует подходящих численных методов и (для матриц размером в миллиарды элементов) распределённых вычислений на сотнях GPU. Современная индустрия рекомендательных систем — это в первую очередь огромная оптимизационная машинерия.

Линейно-алгебраический кит. Сама постановка — «представить матрицу в виде произведения двух маленьких» — это вопрос линейной алгебры. И тут возникает важная *общая*

идея, далеко выходящая за рамки рекомендательных систем.

Идея малоранговой аппроксимации играет ключевую роль в современных больших языковых моделях. Когда нужно *дообучить* (fine-tune) готовую модель с миллиардами параметров под узкую задачу (юриспруденция, медицина, диалект), обновлять все веса — слишком дорого. Метод **LoRA** (*Low-Rank Adaptation*, Хью и др., 2021) предлагает: оставить исходные веса W_0 замороженными, а поверх каждой матрицы добавить *малоранговую поправку* $\Delta W = AB^\top$, где A, B — матрицы с очень небольшим рангом r (обычно $r = 8$ или 16).
Обновляются только параметры A, B — их в сотни раз меньше, чем параметров полной модели. При этом результат дообучения почти неотличим от полного. LoRA-адаптеры (один на каждую узкую задачу) весят считанные мегабайты при том, что базовая модель — сотни гигабайтов; их можно «навинчивать» и снимать на лету. Это и доказывает фундаментальную важность идеи малоранговой аппроксимации в современном машинном обучении.

4.4.3 Автокодировщики: общая идея

Второй классический сюжет обучения без учителя — **автокодировщик** (*autoencoder*). Его базовая идея проста и красива: научить нейронную сеть *воспроизводить свой собственный вход*, но сделать это, проходя через узкое «горло» из небольшого числа нейронов. Тогда сеть будет вынуждена сжать всю существенную информацию о входе в этом горле — получится содержательное представление, или **эмбеддинг** (рис. 4.23).

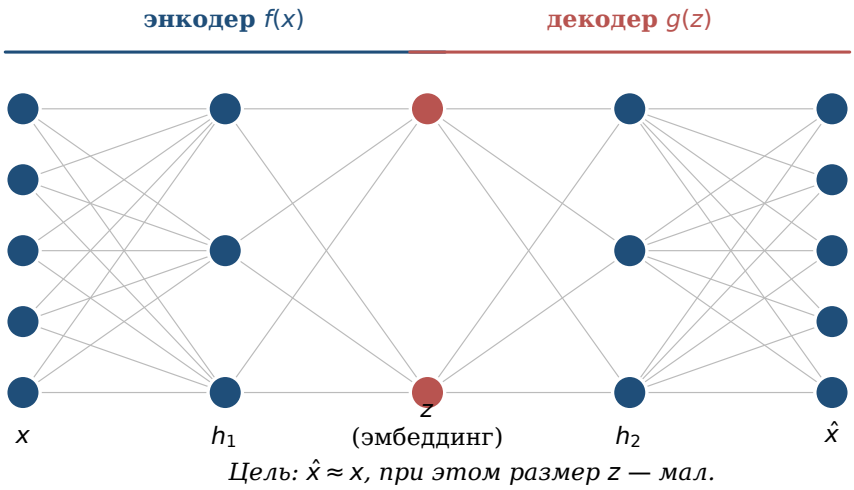


Рис. 4.23. Архитектура автокодировщика. Энкодер f преобразует вход $\mathbf{x}$ в компактный вектор $\mathbf{z}$ малой размерности (узкое горлышко). Декодер g пытается из этого $\mathbf{z}$ восстановить $\hat{\mathbf{x}} \approx \mathbf{x}$. Обучение минимизирует $\|\mathbf{x} - \hat{\mathbf{x}}\|^2$ на выборке. Поскольку $\mathbf{z}$ — маломерен, сеть вынуждена сохранять только самое существенное.

Определение 4.19. Автокодировщик

Автокодировщик — это пара функций $f\colon \mathbb{R}^n \to \mathbb{R}^r$ (энкодер) и $g\colon \mathbb{R}^r \to \mathbb{R}^n$ (декодер), $r < n$, обученные на выборке $\{\mathbf{x}_1, \ldots, \mathbf{x}_N\} \subset \mathbb{R}^n$ минимизацией функционала

$$\mathcal{L}(f, g) \;=\; \frac{1}{N} \sum_{i=1}^{N} \|\mathbf{x}_i - g(f(\mathbf{x}_i))\|^2 . \tag{4.50}$$

Вектор $\mathbf{z}_i = f(\mathbf{x}_i) \in \mathbb{R}^r$ называется **эмбеддингом** (*embedding*, скрытым представлением) объекта $\mathbf{x}_i$. Размерность r — *гиперпараметр* модели.

4.4.4 Линейный автокодировщик и связь с PCA

Самый простой случай — когда f и g *линейны*: $f(\mathbf{x}) = W_e\mathbf{x}$, $g(\mathbf{z}) = W_d\mathbf{z}$, где $W_e \in \mathbb{R}^{r\times n}$ и $W_d \in \mathbb{R}^{n\times r}$ — обучаемые матрицы (для простоты считаем, что данные предварительно центрированы, $\overline{\mathbf{x}} = \mathbf{0}$).

Постановка. Функционал (4.50) принимает вид

$$\mathcal{L}(W_d, W_e) \;=\; \frac{1}{N}\sum_{i=1}^{N} \|\mathbf{x}_i - W_d W_e \mathbf{x}_i\|^2 \;\longrightarrow\; \min_{W_d, W_e}. \tag{4.51}$$

Решение и его геометрия. Обозначим $X \in \mathbb{R}^{n\times N}$ — матрицу, столбцы которой — центрированные обучающие вектора $\mathbf{x}_1, \ldots, \mathbf{x}_N$. Введём *выборочную ковариационную матрицу* $C = \frac{1}{N}XX^\top \in \mathbb{R}^{n\times n}$. Эта матрица симметрична и неотрицательно определена.

Матрица $C \in \mathbb{R}^{n\times n}$ называется *симметричной*, если $C = C^\top$, и *неотрицательно определённой*, если $\langle \mathbf{v}, C\mathbf{v}\rangle \geq 0$ для любого $\mathbf{v} \in \mathbb{R}^n$. *Спектральная теорема* линейной алгебры утверждает: для любой симметричной матрицы C существует ортонормированный базис $\mathbf{e}_1, \ldots, \mathbf{e}_n$ из её собственных векторов, $C\mathbf{e}_k = \lambda_k \mathbf{e}_k$. Для неотрицательно определённой матрицы все $\lambda_k \geq 0$. Геометрически: оператор с матрицей C в этом базисе — просто покоординатное растяжение в λ_k раз вдоль k-й оси.

Упорядочим собственные значения C по убыванию: $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_n \geq 0$, с соответствующими ортонормированными собственными векторами $\mathbf{e}_1, \ldots, \mathbf{e}_n$.

Теорема 4.20. (Решение линейного автокодировщика)
Минимум функционала (4.51) (при условии, что f и g линейные, $r < n$) достигается, когда *подпространство*, порождённое строками W_e, совпадает с подпространством, натянутым на первые r собственных векторов $\mathbf{e}_1, \ldots, \mathbf{e}_r$ ковариационной матрицы C. При этом действие энкодера f есть *ортогональная проекция* на это подпространство.

(Доказательство опирается на спектральное разложение и теорему Эккарта–Янга–Мирского, к которой мы перейдём в раздел 4.4.8.)

Что это означает по содержанию. Линейный автокодировщик *ровно тот же* объект, что **метод главных компонент** (*Principal Component Analysis*, PCA), открытый К. Пирсоном в 1901 г. задолго до всех нейронных сетей.

Геометрически: энкодер — ортогональная проекция в подпространство максимальной дисперсии; декодер — обратное вложение из этого подпространства назад. Эмбеддинг точки — её координаты в этом подпространстве.

4.4.5 Нелинейные автокодировщики и их разновидности

Принципиальный шаг вперёд — разрешить f и g быть произвольными нейронными сетями. Тогда эмбеддинг $\mathbf{z}$ может «свернуть» в свои несколько координат нелинейные комбинации исходных признаков, что несоизмеримо мощнее линейной проекции.

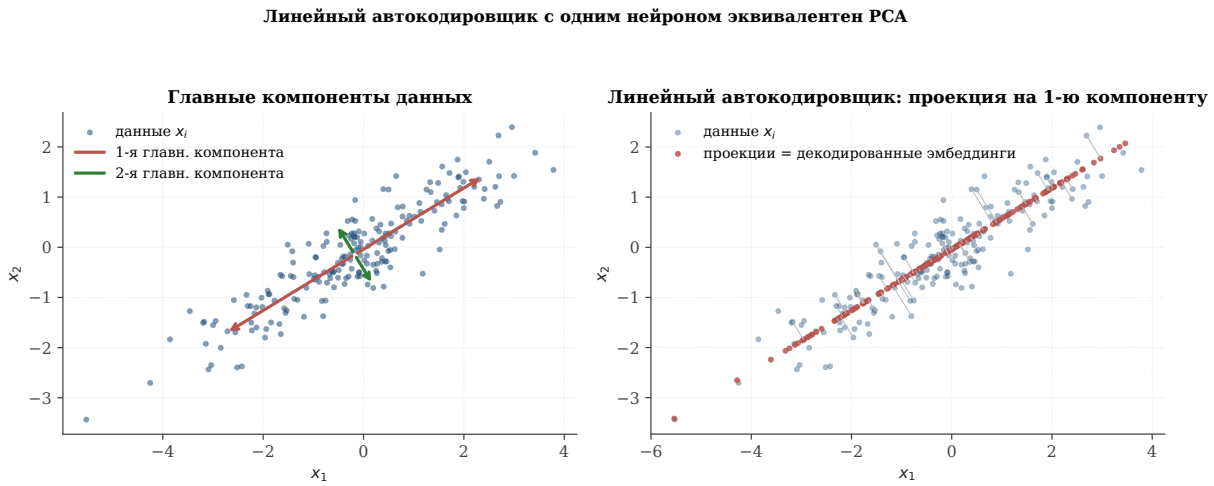


Рис. 4.24. Геометрический смысл линейного автокодировщика. *Слева*: 2-мерные данные (синие точки) лежат, в основном, вдоль одного направления; красная стрелка — первый собственный вектор ковариационной матрицы (1-я главная компонента); зелёная — второй (2-я главная компонента, существенно короче — дисперсия по этому направлению мала). *Справа*: линейный автокодировщик с размерностью эмбеддинга $r = 1$ восстанавливает каждую точку — её *ортогональной проекцией* на 1-ю главную компоненту (красные точки). Эта проекция и есть «лучшее одномерное представление» исходных двумерных данных в смысле наименьших квадратов.

Свёрточные автокодировщики. Если вход — изображение, естественно использовать свёрточные слои в энкодере (раздел 4.3.7) и обратные операции (deconvolution, или transposed convolution) в декодере. Так строятся *свёрточные автокодировщики*, лежащие в основе ранних систем сжатия изображений и шумоподавления.

Разреженный автокодировщик. Добавим к функционалу (4.50) штрафное слагаемое, требующее, чтобы эмбеддинг был *разрежён* (т. е. большинство его координат близки к нулю):

$$\mathcal{L}_{\text{sparse}} \;=\; \mathcal{L} + \lambda \sum_{i,k} |z_{ik}|, \qquad z_{ik} = f(\mathbf{x}_i)_k.$$

Это заставляет сеть «активировать» лишь несколько координат эмбеддинга для каждого входа — получается интерпретируемое представление, в котором за каждым активным нейроном «закрепляется» свой признак.

Шумоподавляющий автокодировщик. Берём $\mathbf{x}_i$, портим его шумом $\tilde{\mathbf{x}}_i = \mathbf{x}_i + \boldsymbol{\eta}$, а сеть учим восстанавливать *чистое* $\mathbf{x}_i$ по *шумному* $\tilde{\mathbf{x}}_i$:

$$\mathcal{L}_{\text{denoise}} \;=\; \frac{1}{N} \sum_{i=1}^{N} \bigl\| \mathbf{x}_i - g(f(\tilde{\mathbf{x}}_i)) \bigr\|^2 .$$

Это вариант, предложенный П. Венсаном и Й. Бенжио в 2008 г., — очень полезен на этапе предобучения глубоких сетей. Эта идея является концептуальным прародителем современных *диффузионных моделей* (Stable Diffusion, DALL-E — 2022 г. и позднее): они обучаются шаг за шагом удалять гауссов шум из изображения и таким образом порождают совершенно новые картинки из «случайной соли и перца».

Вариационный автокодировщик (VAE). Самая глубокая идейная модификация автокодировщика — **вариационный автокодировщик** (Кингма, Веллинг, 2013). Энкодер выдаёт не сам эмбеддинг, а параметры *распределения* над эмбеддингами:

$$f(\mathbf{x}) \;=\; \bigl(\boldsymbol{\mu}(\mathbf{x}),\, \boldsymbol{\sigma}(\mathbf{x})\bigr),$$

после чего сам вектор $\mathbf{z}$ выбирается случайно из нормального распределения $\mathcal{N}\big(\boldsymbol{\mu}(\mathbf{x}), \boldsymbol{\sigma}(\mathbf{x})^2\big)$ и подаётся в декодер. Обучаемая функция потерь складывается из двух слагаемых:

$$\mathcal{L} \;=\; \underbrace{\|\mathbf{x} - g(\mathbf{z})\|^2}_{\text{невязка восстановления}} \;+\; \beta\, \underbrace{\mathrm{KL}\big(\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\sigma}^2)\,\|\,\mathcal{N}(0,1)\big)}_{\text{регуляризатор}}.$$

Здесь KL — **расхождение Кульбака–Лейблера**, или «расстояние» между двумя вероятностными распределениями (определяется в курсе теории вероятностей). В нашем случае оно *штрафует отклонение* распределения эмбеддингов $\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\sigma}^2)$ от «эталонного» стандартного нормального $\mathcal{N}(0,1)$. Содержательно это означает: похожие на вид изображения должны давать близкие $\mathbf{z}$, а всё пространство $\mathbf{z}$ должно быть «равномерно заполненным».

Главное преимущество VAE — возможность *генерации* новых объектов: достаточно выбрать $\mathbf{z}$ случайно из $\mathcal{N}(0,1)$ и прогнать через декодер; благодаря регуляризатору почти любой такой $\mathbf{z}$ декодируется в осмысленное изображение.

4.4.6 Эмбеддинги в современных моделях

Идея эмбеддинга вышла далеко за рамки автокодировщиков. Сегодня *любая* глубокая модель ИИ построена вокруг представления своих входов в виде векторов в каком-то многомерном пространстве — их называют общим словом *эмбеддинги*.

Эмбеддинги в компьютерном зрении. Рассмотрим обученную свёрточную сеть для классификации изображений (например, ResNet-50 на ImageNet с 1000 классов). Архитектурно её последние два слоя устроены так:

$$\underbrace{\mathbf{x} \;\longrightarrow\; \ldots \;\longrightarrow\; \mathbf{z} \in \mathbb{R}^{2048}}_{\text{свёрточный «костяк»}} \;\xrightarrow{W \in \mathbb{R}^{1000\times 2048}}\; \mathbf{p} = \mathrm{softmax}(W\mathbf{z}) \in \mathbb{R}^{1000}.$$

Вектор $\mathbf{z}$ на предпоследнем слое — это и есть *эмбеддинг изображения*: сжатое представление в $\mathbb{R}^{2048}$, из которого последнее линейное преобразование W восстанавливает вероятности классов. Эмпирически оказалось, что $\mathbf{z}$ *переносим*: его можно использовать для совсем других задач — поиска похожих фотографий (сравнение по косинусной мере), классификации классов, которых не было в обучении (*transfer learning*), распознавания одного и того же лица на разных снимках. В этом и состоит главная практическая ценность эмбеддингов: одна обученная сеть даёт универсальное «представление изображения», пригодное для целого спектра приложений.

Эмбеддинги в больших языковых моделях. Каждое слово, а точнее — каждый *токен* (подпоследовательность символов), современная LLM кодирует вектором фиксированной длины (обычно 1024–4096 чисел). Эти эмбеддинги — обучаемые параметры модели; в процессе обучения семантически близкие слова приобретают близкие эмбеддинги (рис. 4.25).

> Современная большая языковая модель (**LLM**; примеры — GPT-серии от OpenAI, Claude от Anthropic, Llama от Meta, GigaChat от Сбера, YandexGPT от Яндекса) построена на архитектуре **трансформер** (Васвани и др., 2017). На входе модель получает последовательность токенов; каждый токен превращается в эмбеддинг.
> Дальше эти эмбеддинги многократно прогоняются через слои двух типов:
>
> - *механизм внимания* (*self-attention*): каждый токен «смотрит» на другие токены последовательности и обновляет свой эмбеддинг, учитывая контекст. Это и есть то, что делает модель «контекстно-зависимой»: то же слово в разных контекстах

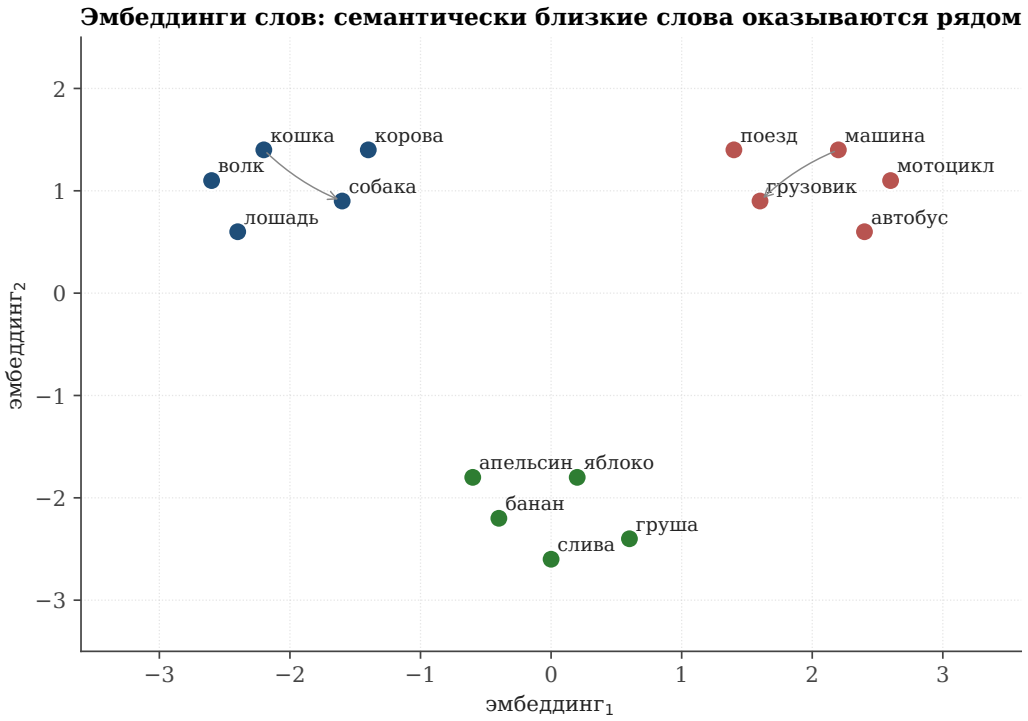


Рис. 4.25. Двумерная проекция эмбеддингов 15 слов на трёх семантических кластеров: животные, транспорт, фрукты. Эмбеддинги получены из обучения модели предсказывать слово по контексту (модель word2vec, 2013). Слова, близкие по смыслу, оказываются близкими в пространстве эмбеддингов — хотя никаких меток «это слово относится к классу ‘животные’» модели не давалось! Семантика возникает из статистики.

получает разный финальный эмбеддинг.

- *полносвязный блок* (FFN, *feedforward network*) — обычный полносвязный слой большой ширины. Именно в этих блоках «хранятся знания» модели — факты, выученные из обучающих текстов.

В конце последний эмбеддинг конвертируется в распределение вероятностей над всеми токенами словаря — и модель предсказывает следующий токен. Так, генерируя по одному токену за раз, LLM пишет тексты, отвечает на вопросы, сочиняет код — всё это, по сути, последовательное предсказание следующего токена.

4.4.7 Сиамские сети и контрастивное обучение

В автокодировщике мы заставляли сеть восстанавливать вход целиком. Часто это избыточно: для практических задач нам нужно не «восстановить картинку до пикселя», а узнать, *похожи ли две картинки между собой*. Этим занимается **контрастивное (или контрастное) обучение** (*contrastive learning*).

Сиамская сеть. **Сиамская сеть** (*Siamese network*, Бромли с соавторами, 1993) состоит из двух копий одной и той же нейронной сети f с общими весами. На вход подают *пару* объектов; каждый прогоняется через свою копию f, и в результате получаются два эмбеддинга $f(\mathbf{x})$ и $f(\mathbf{y})$. Их близость или удалённость характеризует «похожесть» объектов (рис. 4.26).

Контрастивный функционал. Пусть имеется набор обучающих пар $\{(\mathbf{x}_i, \mathbf{y}_i, t_i)\}$, где $t_i = 1$, если пара — «положительная» (похожие), $t_i = 0$, если «отрицательная» (непохожие).

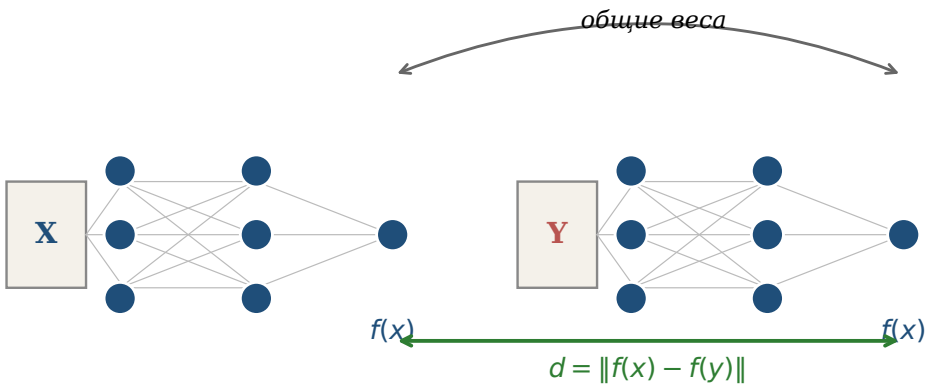


Рис. 4.26. Архитектура сиамской сети. Два экземпляра одной и той же нейронной сети f обрабатывают пару объектов $\mathbf{x}$ и $\mathbf{y}$. На выходе вычисляется расстояние $d = \|f(\mathbf{x}) - f(\mathbf{y})\|$. В процессе обучения параметры f настраиваются так, чтобы d было *малым* для «похожих» пар (например, двух фотографий одного и того же человека) и *большим* для «непохожих» (фотографии разных людей).

Простейший *контрастивный* функционал (Хадселл, Шопра, ЛеКун, 2006):

$$\mathcal{L} \;=\; \sum_{i} \Big[\, t_i \cdot d_i^2 \;+\; (1 - t_i) \cdot \big(\max(0,\, m - d_i)\big)^2 \,\Big], \tag{4.52}$$

где $d_i = \|f(\mathbf{x}_i) - f(\mathbf{y}_i)\|$, а $m > 0$ — *отступ* (*margin*), гиперпараметр. Смысл: для похожих пар ($t_i = 1$) минимизируется d_i^2 — расстояние «прижимается» к нулю; для непохожих ($t_i = 0$) штрафуется только то, что d_i *меньше* m — то есть от непохожих пар требуется минимальное расстояние m, а дальше уже всё равно.

Применения. Распознавание лиц. Сиамская сеть **FaceNet** (Google, 2015) обучается на миллионах фотографий: эмбеддинг каждого лица — вектор в $\mathbb{R}^{128}$. Положительные пары — два снимка одного человека (часто в разных условиях); отрицательные — разных. После обучения по тестовому снимку можно определить личность простым поиском ближайшего эмбеддинга в базе. Точность — более 99,6 % на стандартном бенчмарке LFW. Такая система — основа всех современных систем распознавания лиц.

Поиск изображений по тексту (и обратно). Модель **CLIP** (OpenAI, 2021) обучает *два* энкодера: один — для изображений (свёрточная или трансформерная сеть), другой — для текстов (трансформер). Положительные пары — картинка и её настоящая подпись; отрицательные — картинка и случайная подпись из батча. После обучения эмбеддинги изображения и эмбеддинги его осмысленного текстового описания оказываются близкими в общем многомерном пространстве. Это даёт колоссальные возможности: искать картинки по произвольным описаниям («рыжий кот на пианино»), классифицировать изображения в произвольные классы, не имея размеченной выборки (*zero-shot classification*), и многое другое.

Дубликаты в поиске. Тот же контрастивный подход используют поисковые системы (Яндекс, Google) для дедупликации страниц, рекомендательные системы — для поиска похожих товаров, антивирусы — для группировки похожих образцов вредоносного кода.

4.4.8 Сингулярное разложение матриц*

Этот раздел — *со звёздочкой*: материал более продвинутый, и при первом чтении его можно пропустить. Однако именно здесь раскрывается глубокая связь между двумя сюжетами параграфа — матричной факторизацией и линейным автокодировщиком — через одну и ту же алгебраическую конструкцию.

Матрицы как линейные преобразования. Каждая матрица $A \in \mathbb{R}^{m\times n}$ задаёт *линейное преобразование* $\mathbb{R}^n \to \mathbb{R}^m$, $\mathbf{x} \mapsto A\mathbf{x}$ (см. определение 4.10). Столбцы матрицы A — это образы базисных ортов $\mathbf{e}_1, \dots, \mathbf{e}_n$: j-й столбец равен $A\mathbf{e}_j$. Произведение матриц $C = AB$ соответствует *суперпозиции* преобразований: сначала действует B, потом A.

Простейшие линейные преобразования. Среди всех линейных преобразований $\mathbb{R}^n \to \mathbb{R}^n$ выделяются два простейших класса.

- **Повороты** — матрицы Q, сохраняющие длины: $\|Q\mathbf{x}\| = \|\mathbf{x}\|$. Они называются *ортогональными* и характеризуются равенством $Q^\top Q = I$.
- **Растяжения по осям** — диагональные матрицы Σ: $\Sigma\mathbf{e}_k = \sigma_k \mathbf{e}_k$, $\sigma_k \geq 0$. Они растягивают k-ю ось в σ_k раз.

Сингулярное разложение. Знаменитый факт линейной алгебры (Бельтрами, 1873; Йордан, 1874; Сильвестр, 1889 для квадратных матриц; Экарт и Янг, 1936 для прямоугольных) — *любое* линейное преобразование можно представить как комбинацию поворота, растяжения и снова поворота:

Теорема 4.21. (Сингулярное разложение (SVD))

Для любой матрицы $A \in \mathbb{R}^{m\times n}$ существует представление

$$\boxed{A \;=\; U\,\Sigma\,V^\top,} \tag{4.53}$$

где $U \in \mathbb{R}^{m\times m}$ и $V \in \mathbb{R}^{n\times n}$ — ортогональные матрицы ($U^\top U = I_m$, $V^\top V = I_n$), а $\Sigma \in \mathbb{R}^{m\times n}$ — «диагональная» (т. е. $\Sigma_{ij} = 0$ при $i \neq j$) с неотрицательными элементами на диагонали $\sigma_1 \geq \sigma_2 \geq \dots \geq \sigma_{\min(m,n)} \geq 0$. Эти числа называются **сингулярными значениями** матрицы A; столбцы U и V — **сингулярными векторами**.

Геометрически: преобразование с матрицей A можно представить последовательностью — сначала поворот в исходном пространстве (матрица $V^\top$), затем покоординатное растяжение (матрица Σ), затем поворот в пространстве-образе (матрица U) (рис. 4.27).

Полезный способ читать SVD: столбцы $\mathbf{v}_1, \dots, \mathbf{v}_n$ матрицы V — это *«входные направления»*, на которых матрица A действует чище всего (как простое растяжение, без поворота). Столбцы $\mathbf{u}_1, \dots, \mathbf{u}_m$ матрицы U — соответствующие *«выходные направления»*: $A\mathbf{v}_i = \sigma_i\,\mathbf{u}_i$. Числа σ_i показывают, во сколько раз растягивается каждое такое направление; по соглашению они упорядочены по убыванию.

Связь с симметричной задачей. Доказательство SVD сводится к спектральной теореме для симметричной матрицы. Заметим:

- $A^\top A = (U\Sigma V^\top)^\top (U\Sigma V^\top) = V\Sigma^\top U^\top U\Sigma V^\top = V\Sigma^\top\Sigma V^\top$ — это спектральное разложение симметричной неотрицательной матрицы $A^\top A$. Её собственные значения — σ_k^2; собственные векторы — столбцы V.
- Аналогично, $AA^\top = U\Sigma\Sigma^\top U^\top$. Собственные векторы $AA^\top$ — столбцы U.

Так одно общее утверждение для произвольных матриц редуцируется к симметричному случаю.

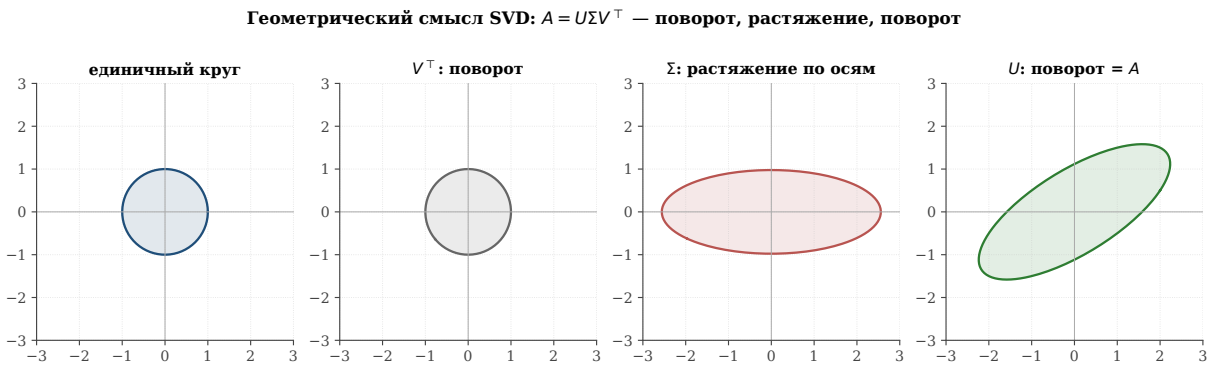


Рис. 4.27. Геометрическая иллюстрация SVD $A = U\Sigma V^\top$ на примере $A = \left(\begin{smallmatrix} 2 & 1 \\ 0{,}5 & 1{,}5 \end{smallmatrix}\right)$. Единичная окружность последовательно: 1) поворачивается матрицей $V^\top$ — столбцы $\mathbf{v}_1, \mathbf{v}_2$ матрицы V при этом становятся координатными осями; 2) растягивается матрицей Σ по координатным осям — появляется эллипс с полуосями σ_1, σ_2; 3) ещё раз поворачивается матрицей U — координатные оси переходят в столбцы $\mathbf{u}_1, \mathbf{u}_2$ матрицы U, и итог совпадает с прямым применением A к окружности.

Теорема о наилучшей малоранговой аппроксимации. SVD замечательно тем, что позволяет получить наилучшее приближение матрицы матрицей фиксированного ранга.

Теорема 4.22. (Эккарта–Янга–Мирского (1936))

Пусть $A \in \mathbb{R}^{m\times n}$ и $A = U\Sigma V^\top$ — её сингулярное разложение. Для любого $r \le \min(m,n)$ определим *обрезанное* разложение

$$A_r \;=\; \sum_{k=1}^{r} \sigma_k\, \mathbf{u}_k\, \mathbf{v}_k^\top \;=\; U_r\, \Sigma_r\, V_r^\top,$$

где U_r, V_r — первые r столбцов матриц U, V, а $\Sigma_r = \operatorname{diag}(\sigma_1, \dots, \sigma_r)$. Тогда матрица A_r имеет ранг r и реализует наилучшую среди всех матриц ранга $\le r$ аппроксимацию исходной матрицы:

$$\min_{\operatorname{rank} B \le r} \|A - B\|_F \;=\; \|A - A_r\|_F \;=\; \sqrt{\sigma_{r+1}^2 + \dots + \sigma_{\min(m,n)}^2}, \tag{4.54}$$

где $\|\cdot\|_F$ — норма Фробениуса: $\|M\|_F^2 = \sum_{i,j} M_{ij}^2$.

Иными словами, отбрасывая в разложении (4.53) все сингулярные значения, начиная с $(r+1)$-го, мы получаем оптимальную малоранговую аппроксимацию. Если первые r сингулярных значений велики, а последующие малы, то A_r — почти такая же хорошая, как A, но в $\frac{mn}{(m+n)r}$ раз более компактна по числу параметров (рис. 4.28).

Связь с автокодировщиком и матричной факторизацией. *Линейный автокодировщик из раздел 4.4.4* — это в точности SVD-факторизация центрированных данных. Если столбцы X — обучающая выборка, и $X = U\Sigma V^\top$ её SVD, то оптимальный энкодер — $f(\mathbf{x}) = U_r^\top \mathbf{x}$, оптимальный декодер — $g(\mathbf{z}) = U_r \mathbf{z}$, где U_r — первые r столбцов матрицы U. Сингулярные значения σ_k показывают, насколько важна k-я главная компонента (большое σ_k — много дисперсии в этом направлении).

Матричная факторизация $A \approx UV^\top$ из раздел 4.4.1 — это (без ограничения на неотрицательность) ровно $A \approx A_r$ из теоремы Эккарта–Янга–Мирского. Если бы все элементы A были известны, идеальной стратегией было бы взять обрезанное SVD-разложение ранга r. В реальной Netflix problem известны лишь $\sim 1\,\%$ элементов матрицы, и приходится решать задачу (4.48), которая является обобщением SVD на случай пропусков.

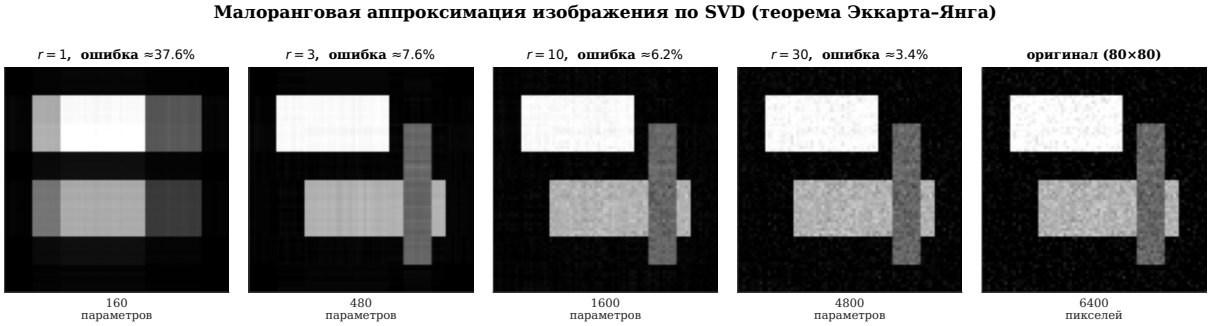


Рис. 4.28. Малоранговая аппроксимация изображения по SVD: восстановление картинки 80×80 при разных рангах. При $r = 1$ — крупная полоса (один «прототип»). При $r = 3$ виден общий силуэт. При $r = 10$ — почти неотличимо от оригинала, но в ~ 4 раза меньше параметров. При $r = 30$ совпадение полное, при том что число параметров на 38 % меньше, чем в оригинальном представлении.

Школа Е. Е. Тыртышникова и тензорные разложения. Идея малоранговой аппроксимации не ограничивается матрицами. **Тензором** называется многомерный массив чисел: матрица — двумерный тензор, а матрицы с тремя и более индексами — тензоры высших порядков. Тензоры возникают повсюду в современных приложениях: изображение видеоряда — 4-индексный тензор (время × высота × ширина × цвет); веса трансформера — тысячи матриц, организованных в одну сложную структуру; квантово-механические задачи — тензоры с десятками индексов.

Аналог малоранговой аппроксимации для тензоров — значительно более тонкая задача. Современный мощный инструмент — **разложение в виде тензорного поезда** (*tensor train decomposition*), предложенное в 2009 г. российским математиком, профессором РАН **Иваном Валерьевичем Оселедцем**, учеником академика **Евгения Евгеньевича Тыртышникова** (МГУ, ИВМ РАН). Тензорный поезд позволяет хранить тензор с d индексами при размере N по каждому индексу с помощью $\sim d\,N\,r^2$ параметров вместо N^d — то есть избавляет от знаменитого «проклятия размерности». Этот результат лежит в основе многих современных алгоритмов в квантовой химии, машинном обучении и решении многомерных дифференциальных уравнений.

За эти и связанные работы Е. Е. Тыртышников — один из мировых лидеров в области тензорных и матричных разложений — является лауреатом премий Сбербанка в области искусственного интеллекта.

4.4.9 Резюме параграфа

Обучение без учителя — это семейство задач, в которых модель должна самостоятельно обнаружить структуру в данных без подсказок-меток. В этом параграфе мы разобрали два центральных сюжета.

Восстановление матрицы (Netflix problem):

- Гипотеза малоранговой факторизации $A \approx UV^\top$ — из неё естественным образом получается рекомендательная система.
- Функционал ошибки $\sum_{(i,j)\in\Omega}(A_{ij} - \langle \mathbf{u}_i, \mathbf{v}_j\rangle)^2$ — здравый смысл, который при желании можно вывести из ОМП в предположении гауссова шума.
- Идея малоранговой коррекции легла в основу LoRA — стандарта дообучения больших языковых моделей.

Автокодировщики:

- Энкодер f сжимает вход в эмбеддинг $\mathbf{z}$ малой размерности; декодер g восстанавливает вход из этого эмбеддинга. Минимизируется $\|\mathbf{x} - g(f(\mathbf{x}))\|^2$.

- В линейном случае оптимальное решение — ортогональная проекция на подпространство максимальной дисперсии (PCA).
- Эмбеддинги — универсальное представление в современных моделях ИИ: компьютерное зрение, языковые модели, рекомендации.
- Контрастивное обучение через сиамские сети позволяет обучать качественные эмбеддинги, не требуя восстанавливать вход целиком — достаточно знать, что одни пары похожи, а другие нет.

Оба сюжета фундаментально связаны через **сингулярное разложение** $A = U\Sigma V^\top$ и теорему Эккарта–Янга–Мирского о наилучшей малоранговой аппроксимации. Это — мощный универсальный инструмент линейной алгебры, лежащий и за рекомендательными системами, и за PCA, и за современными тензорными разложениями (школа Тыртышникова–Оселедца).

Задачи для самостоятельной работы

1. Проверьте, что в примере 4.18 матрицы U и V действительно дают $UV^\top$, близкое к A во всех наблюдённых клетках.
2. Сколько параметров в модели $A \approx UV^\top$, если $m = 10^6$ пользователей, $n = 10^4$ фильмов, ранг $r = 20$? Во сколько раз это меньше числа элементов исходной матрицы?
3. Покажите, что для центрированной выборки матрица $C = \frac{1}{N} XX^\top$ действительно симметрична и неотрицательно определена.
4. Сделайте свой PCA: возьмите датасет $\mathbf{x}_i \in \mathbb{R}^2$ вашего сочинения (хотя бы 30 точек), посчитайте C, найдите собственные значения и векторы (можно с помощью `numpy.linalg.eig`). Постройте график как на рис. 4.24.
5. Возьмите изображение в градациях серого. Получите его SVD (`numpy.linalg.svd`) и постройте малоранговые аппроксимации с $r = 1, 5, 20, 50$. Постройте график зависимости ошибки в норме Фробениуса от r. Найдите минимальное r, при котором ошибка не превышает 5 % от $\|A\|_F$.
6. ⋆ Докажите формулу (4.54), опираясь на SVD и ортогональность U, V в норме Фробениуса.
7. ⋆ Сиамская сеть для рукописных цифр. Возьмите MNIST/UCI Digits. Постройте сиамскую сеть, обучающуюся на парах «одна и та же цифра / разные цифры». После обучения покажите, что эмбеддинги одной цифры из разных написаний оказываются близкими.

4.5 Как искусственный интеллект меняет мир

В этом завершающем параграфе мы поднимемся над частными алгоритмами и посмотрим на современный ИИ как на явление, — разберёмся, как последние десять лет изменили картину мира, какие силы движут революцию больших моделей, что ИИ умеет, а что — пока нет, и как с большой осторожностью можно говорить о ближайшем будущем (рис. 4.29).

4.5.1 AlphaGo и второй прорыв в обучении с подкреплением

С точки зрения общественного восприятия, ИИ стал «настоящим ИИ» 9 марта 2016 г., в первый день матча **AlphaGo** против Ли Седоля. Если поражение Гарри Каспарова шахматному компьютеру Deep Blue в 1997 г. многие готовы были списать на «грубый перебор»,

Рис. 4.29. Хронология важнейших вех в развитии ИИ. От первого перцептрона Розенблатта (1957) до Нобелевских премий по физике и химии 2024 г., присуждённых за вклад в ИИ, — 67 лет, причём революционные прорывы концентрируются на последних 15 годах.

то с игрой **го** такое объяснение не проходило: число позиций ($\sim 10^{170}$, что превосходит число атомов в наблюдаемой Вселенной) исключает любой полный перебор. Считалось, что компьютер достигнет человеческого уровня в го не раньше чем через 30 лет. AlphaGo, разработанная компанией DeepMind, обыграла Ли Седоля со счётом 4:1.

Архитектура AlphaGo. Под капотом находились две глубокие свёрточные нейронные сети:

- *сеть стратегии* (*policy network*): по позиции выдавала распределение вероятностей следующего хода;
- *сеть оценки* (*value network*): по позиции выдавала оценку шансов на победу.

Они комбинировались с *поиском Монте-Карло по дереву* (Monte Carlo Tree Search, MCTS) и обучались на партиях, сыгранных людьми, — своего рода имитационное обучение.

AlphaGo Zero: ИИ, который учился сам. Поразительным продолжением стала система **AlphaGo Zero** (октябрь 2017): она не использовала *никаких* человеческих партий, а училась только играя сама с собой — так называемое *обучение с подкреплением через self-play*. За 40 дней самообучения она достигла уровня, превосходящего AlphaGo 2016-го года, и в матче из 100 партий выиграла у предыдущей версии со счётом 100:0.

AlphaGo Zero показала: *ИИ может учиться, взаимодействуя сам с собой*, не нуждаясь ни в каких внешних данных — лишь имея формализованную задачу и среду, в которой можно играть. Это *принципиально* меняет масштаб возможного: данных не хватает — ИИ их сам производит. Этот принцип лежит в основе многих современных направлений: автоматического доказательства теорем, поиска новых материалов, проектирования лекарств. Развитие идеи привело к концепции *мультиагентного* ИИ — системы, где много обучаемых агентов взаимодействуют друг с другом, образуя экосистему, в которой коллективное знание растёт быстрее индивидуального.

4.5.2 Революция трансформеров и появление ChatGPT

Вторая, ещё более масштабная революция произошла в области обработки естественного языка — и она опирается на одну архитектурную идею.

В июне 2017 г. группа исследователей Google (Васвани, Шазир, Пармáр и др.) опубликовала статью *Attention Is All You Need*, в которой предложила архитектуру **трансформер** (*Transformer*). Идея была одновременно простой и революционной: отказаться от последовательной (рекуррентной) обработки текста и заменить её на *механизм внимания*, который позволяет каждому слову последовательности «смотреть» на любые другие слова одновременно. Это дало модели возможность учитывать сколь угодно далёкие зависимости в тексте и принципиально ускорило обучение за счёт параллельных вычислений на GPU.

Каждое слово (точнее — его эмбеддинг) превращается в три вектора: *ключ* (key), *значение* (value) и *запрос* (query). Затем для каждого слова считаются *веса внимания* — скалярные произведения его *запроса* с *ключами* всех остальных слов, пропущенные через softmax. Эти веса говорят, насколько каждое другое слово важно для текущего. Финальный эмбеддинг слова получается как взвешенное среднее *значений* (с этими весами).
Многократно повторяя эту операцию слоями, модель формирует всё более и более «смысловое» представление каждого слова в контексте.

От Transformer до ChatGPT.

- **BERT** (Google, 2018) — 110 млн параметров; первая big-scale демонстрация мощи трансформеров на задачах NLP.
- **GPT-3** (OpenAI, 2020) — 175 млрд параметров; впервые показала, что увеличение масштаба ведёт к новым, ранее невидимым способностям (*emergent abilities*) — перевод, программирование, элементарная арифметика «из коробки», без специального обучения этим задачам.
- **ChatGPT** (OpenAI, ноябрь 2022) — интерактивный чат-интерфейс к модели GPT-3.5, дообученной на диалогах с обратной связью (*RLHF*). За 5 дней — миллион пользователей; за 2 месяца — 100 миллионов (самый быстрорастущий сервис в истории интернета). С этого момента ИИ стал массовым потребительским явлением.

Отечественные модели. Россия — одна из немногих стран, имеющих собственную линейку больших языковых моделей промышленного уровня. **ruGPT-3** (Сбер, 2020) была одной из первых крупных русскоязычных LLM в мире — 13 млрд параметров. **GigaChat** (Сбер, начиная с 2023) — семейство мультимодальных диалоговых моделей. **YandexGPT** (Яндекс, 2023+) — модель, оптимизированная под русский язык и интегрированная в продукты Яндекса (Алиса, Поиск, Браузер).

4.5.3 Законы скейлинга: формулы

К 2020-му году стало понятно, что качество LLM — удивительно *предсказуемая* функция трёх параметров: размера модели N (числа обучаемых параметров), размера датасета D (числа токенов в обучающей выборке) и вычислительного бюджета C (числа операций FLOPS, затраченных на обучение).

Закон Каплана (2020). Группа Каплана из OpenAI обнаружила, что кросс-энтропия (метрика качества языковой модели) убывает по степенному закону:

$$L(N) \;\approx\; L_\infty + \frac{A}{N^\alpha}, \tag{4.55}$$

с $\alpha \approx 0{,}076$ (фиксируя D достаточно большим), где L_∞ — асимптотическая «непредсказуемая шумовая часть» естественного языка.

Закон Чинчилла (2022). Команда DeepMind (Хоффман и др., работа *Training Compute-Optimal Large Language Models*) уточнила картину: для оптимального обучения размер модели и размер датасета должны расти *вместе*, в определённой пропорции. Совместный закон:

$$\boxed{L(N, D) \;=\; L_\infty \;+\; \frac{A}{N^{0{,}34}} \;+\; \frac{B}{D^{0{,}28}},} \tag{4.56}$$

с константами $L_\infty \approx 1{,}69$, $A \approx 406$, $B \approx 411$ (в естественных нормировках).

Главный практический вывод — правило $20:1$. При фиксированном вычислительном бюджете $C \sim 6 \cdot N \cdot D$ (приближённая формула для трансформера) минимум функции $L(N, D)$ достигается, когда

$$D \;\approx\; 20 \cdot N.$$

Иными словами, на каждый параметр модели должно приходиться примерно *20 токенов* обучающего текста. Это правило — неожиданное и *критически важное*: до его обнаружения почти все большие модели (включая GPT-3) были *недообучены* — слишком велики при слишком маленькой обучающей выборке (рис. 4.30).

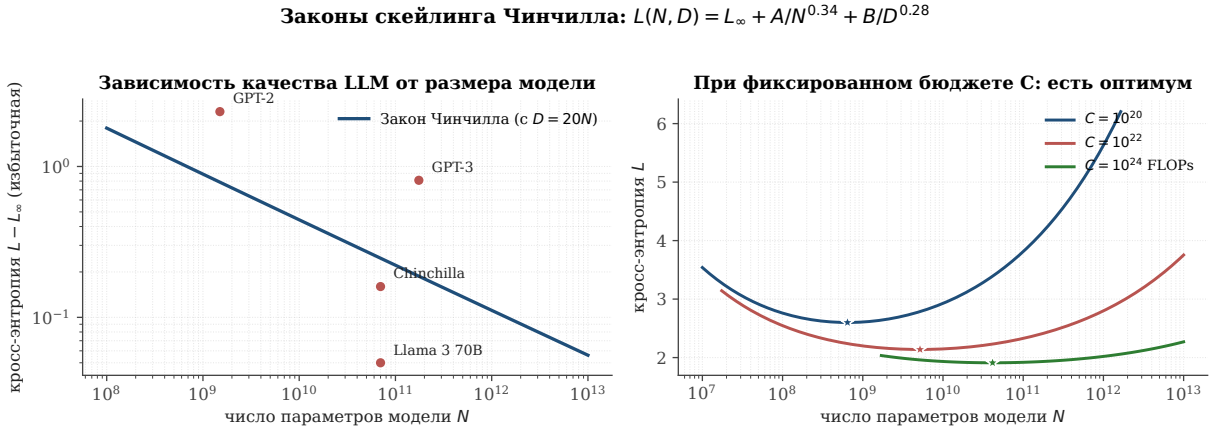


Рис. 4.30. Закон скейлинга Чинчилла в действии. *Слева*: при оптимальном соотношении $D = 20N$ кросс-энтропия убывает по степенному закону на много порядков масштаба. Точки — известные модели; их отклонения вверх от прямой — признак того, что модель *недообучена*. *Справа*: при фиксированном вычислительном бюджете C существует оптимальное соотношение между размером модели и размером датасета. Звёздочками отмечены минимумы — они смещаются вправо с ростом C.

Закон скейлинга объясняет, почему именно сейчас, в 2020-х, ИИ совершил качественный скачок. До 2017-го у нас не было архитектуры (трансформера), способной эффективно обучаться при таких масштабах. До 2010-х не было таких массивов данных (всемирная паутина накопила миллиарды документов). И не было дешёвых GPU, способных параллельно крутить миллиарды умножений матриц. *Качество* современных LLM определяется тремя факторами:

1. грамотными архитектурными идеями (внимание, residual connections, layer normalization, RLHF и пр.);
2. размером модели N;
3. объёмом и качеством обучающих данных D.

И всё это поверх классической оптимизации (раздел 4.3.8) и backpropagation (раздел 4.3.9).

4.5.4 AlphaFold и Нобелевская премия 2024 г.

Долгое время ИИ оставался занятием академическим. Прорывом, после которого никто уже не мог говорить о «модной игрушке», стала **AlphaFold** (DeepMind, 2018; AlphaFold 2 — 2020). Эта система решает задачу, считавшуюся одной из главных «50-летних» проблем биологии: *по последовательности аминокислот предсказать трёхмерную структуру белка.*

Почему это важно. Белки — работники клетки. От их трёхмерной формы зависит, как они работают: блокируют ли вирусную инфекцию, переносят ли кислород, разлагают ли пластик, вызывают ли болезнь Альцгеймера. Долгое время структуру белка определяли экспериментально — методами рентгеновской кристаллографии или криоэлектронной микроскопии. Расшифровка одного белка могла занимать годы, и стоила сотни тысяч долларов.

К концу 2010-х было известно $\sim 170\,000$ структур (база PDB), при том что в человеческом организме *одних только* белков несколько сотен тысяч.

Что сделала AlphaFold. В 2020 г. команда DeepMind во главе с Демисом Хассабисом и Джоном Джампером представила AlphaFold 2 на конкурсе *CASP14* (Critical Assessment of protein Structure Prediction) — международном двухгодичном соревновании по предсказанию структур. Результат был поразительным: средняя точность ≈ 87 по шкале GDT_TS, что сравнимо с экспериментальной точностью кристаллографии (см. рис. 4.31).

В 2021 г. DeepMind открыто опубликовала структуры ~ 200 миллионов белков — практически *все известные* белки в природе. К концу 2024 г. базой AlphaFold пользовались более 2 миллионов учёных из 190 стран.

Нобелевская премия 2024. 9 октября 2024 г. **Нобелевская премия по химии** была присуждена *Демису Хассабису* и *Джону Джамперу* (DeepMind) за создание AlphaFold, а также *Дэвиду Бейкеру* (Вашингтонский университет) за работы по компьютерному дизайну белков. День спустя — **Нобелевская премия по физике** была присуждена *Джону Хопфилду* и *Джеффри Хинтону* «за основополагающие открытия и изобретения, позволившие машинное обучение с помощью искусственных нейронных сетей». Впервые в истории Нобелевские премии по *двум* естественнонаучным номинациям в один и тот же год были связаны с ИИ.

Рис. 4.31. Прогресс точности предсказания структуры белков на конкурсе CASP. Слева — классические алгоритмы 2010-х (рост точности на 1–2 пункта за два года). С появлением AlphaFold 1 на CASP13 (2018) — скачок до 58. С появлением AlphaFold 2 на CASP14 (2020) — скачок до ≈ 87, что эквивалентно экспериментальной точности. Это и стало прорывом, за который в 2024 г. присуждена Нобелевская премия по химии.

Связь с трансформерами. В сердце AlphaFold лежит модификация архитектуры трансформера. Её основное отличие: вместо последовательности слов модель обрабатывает *«последовательность аминокислот»* (~ 20 типов «букв»); аналогом внимания между словами выступает внимание между парами аминокислот в свёрнутом белке. Это и есть глубокая идея: задачу структурной биологии удалось редуцировать к языковой архитектуре. Аналогичный подход — редукция новой задачи к языку — сегодня применяется к десяткам новых областей: химия, материаловедение, геномика, программирование, математика.

4.5.5 Шкалирование во время вывода и цепочки рассуждений

К 2023–2024 гг. стало понятно: одного только увеличения размера модели для следующих качественных скачков может не хватить. Возникла новая идея — **масштабирование на этапе вывода** (*inference-time scaling*).

Цепочки рассуждений (Chain-of-Thought). Классическая большая языковая модель сразу выдаёт ответ. Оказалось, что если попросить модель сначала *рассуждать пошагово вслух* — выписать промежуточные шаги, потом сделать вывод — качество ответов резко возрастает, особенно в задачах математики, программирования, формальной логики. Этот феномен описан в работе Wei et al. (2022) и получил название *цепочек рассуждений* (chain-of-thought, CoT).

Reasoning-модели (o1, o3 и далее). В сентябре 2024 г. OpenAI представила модель *o1* — **reasoning model**, специально обученную «думать дольше» перед ответом. На олимпиадных задачах по математике (AIME, IMO уровень), программированию (Codeforces) и физике она показала результаты, сравнимые с лучшими школьниками-олимпиадниками. В 2025-м — модель o3, более продвинутая. У других разработчиков появились свои аналоги — Claude Sonnet с extended thinking (Anthropic), Gemini Deep Think (Google DeepMind), DeepSeek-R1 (Китай) и многие другие.

Reasoning-модель тратит существенно больше вычислений в момент *ответа на запрос*, а не на этапе обучения. Технически это делается двумя способами: (а) обучить модель генерировать *длинные цепочки промежуточных рассуждений* (десятки или сотни тысяч токенов) перед финальным ответом; (б) явно искать в дереве возможных рассуждений (как MCTS в AlphaGo), оценивая разные ветки и выбирая лучшую.
Возникает новая ось скейлинга: *время на запрос*. Это принципиально — значит, дальнейший прогресс может идти не только за счёт увеличения моделей, но и за счёт более длительных рассуждений.

4.5.6 Воплощённый ИИ и физическая картина мира

При всех успехах языковых моделей у современного ИИ есть очевидные слабости. Главная — отсутствие у моделей того, что **Янн ЛеКун** (главный исследователь Meta AI и один из лауреатов премии Тьюринга 2018 г.) называет «*здравым смыслом физического мира*» (*common sense*). Даже самая большая LLM не знает простых вещей, известных любому двухлетнему ребёнку: если положить чашку на край стола и сдвинуть — она упадёт; если объект скрылся за дверью — он не перестал существовать; если предмет тяжёлый, его трудно поднять.

ЛеКун в работах 2022–2025 гг. предлагает альтернативную к LLM парадигму — *world models* (модели мира). Идея: ИИ должен учиться не на текстах (которые описывают мир), а на *видео и сенсорных данных* — так же, как учатся дети. Внутреннее представление мира получает структуру «причинно-следственного графа»: действие → изменение состояния. Такой ИИ называется *воплощённым* (*embodied AI*). Он, возможно, не будет поражать литературными способностями, но сможет *планировать*, *прогнозировать* и *взаимодействовать* с физическим миром — то, чего нынешним моделям заметно не хватает.
Реализация этой идеи — одна из «святых граалей» современного ИИ. Если она удастся, мы получим ИИ, способный быть оператором робота, а не просто собеседником.

4.5.7 Исчерпание данных и пределы скейлинга

Закон Чинчилла говорит: чтобы получить более качественную модель, нужно одновременно увеличивать N и D. Но публично доступный интернет конечен. По разным оценкам, к 2026–2028 гг. передовые модели будут обучаться на *всём качественном тексте*, написанном человечеством в открытом доступе. Что дальше?

Возможные пути:

- **Синтетические данные** — ИИ сам генерирует обучающую выборку. Уже сейчас часть обучающих данных моделей синтезируется другими моделями. Это путь к *self-improving AI* — системе, обучающейся самой на себе (по аналогии с AlphaGo Zero).
- **Мультимодальные данные** — видео и аудио содержат на порядки больше информации, чем текст. Видео в сети (YouTube и др.) хватит на десятилетия обучения.
- **Алгоритмические улучшения** — эффективные архитектуры могут учиться на меньших объёмах данных. Например, mixture of experts (MoE), retrieval-augmented generation (RAG), новые виды внимания, более эффективные оптимизаторы.
- **Вычислительные пределы.** Закон Мура замедляется; энергетика обучения крупнейших моделей сравнима с потреблением небольших городов. Это уже инженерное ограничение.

4.5.8 Прогнозы и сингулярность по Курцвейлю

Рэй Курцвейль — американский инженер, футуролог, лауреат Национальной медали технологий США — знаменит своими долгосрочными прогнозами в области ИИ. Его взгляды изложены в двух книгах с почти одинаковыми названиями — они отличаются всего одной буквой:

- *The Singularity Is Near* («сингулярность близка»), опубликована в 2005 году. В ней Курцвейль предсказал, что компьютер сравняется с человеком в задачах общего интеллекта к 2029 году, а *сингулярность* — момент кардинального ускорения технологического развития — наступит в 2045 году.
- *The Singularity Is Nearer* («сингулярность ещё ближе»), опубликована почти через двадцать лет, в 2024 году. В этой новой книге Курцвейль подтвердил оба прогноза 2005 года, ссылаясь на ChatGPT и GPT-4 как на «доказательство по дороге к 2029-му».

Большинство экспертов считают эти прогнозы оптимистичными, но смены парадигмы — быстрого и кардинального изменения возможностей ИИ — сегодня не отрицает никто.

> **Artificial General Intelligence** (AGI) — ИИ, превосходящий человека в *любой* интеллектуальной задаче. Сегодня все системы — *узкие*: AlphaGo гениально играет в го, но не знает, что такое кошка; GPT-5 пишет талантливые стихи, но не способен поработать поваром. AGI означал бы единую систему, которая может всё.
> Прогнозы насчёт AGI разнятся от 2027 (Сэм Альтман) до конца 2040-х (многие академические эксперты). Само понятие AGI часто критикуют: проблема в том, что граница между «узким» и «общим» интеллектом расплывчата, а отдельные интеллектуальные навыки появляются у современных моделей постепенно, а не одновременно.

4.5.9 Меняющийся ландшафт профессий

ИИ принципиально меняет очень многие профессии. Главный тренд — *не замена* человека ИИ, а *симбиоз*: человек делает то, что у него хорошо получается (постановка задач, оценка результата, этические решения, общение, креативные прорывы), а рутинная интеллектуальная работа автоматизируется. Этот принцип хорошо называется *«centaur»* — кентавр, союз человека и машины.

Несколько примеров. Программирование. Уже в 2025 г. значительная часть производственного кода в IT-компаниях пишется при участии или полностью с помощью моделей-кодеров (GitHub Copilot, Cursor, Claude Code, Codeium). Программист всё больше превращается в архитектора и ревьюера; рутинные функции, обвязки, тестовый код, документация — автоматизируются.

Наука. Уже сейчас существуют системы, способные *самостоятельно формулировать научную гипотезу, ставить эксперимент (в симуляции), писать черновик статьи.* Это *AI Scientist* — класс систем, появившийся в 2024–2025 гг. В материаловедении, химии, биологии счёт открытым новым материалам, структурам, реакциям с участием ИИ идёт на сотни тысяч в год.

Финансовый сектор. Уже более 70 % алгоритмической торговли ведётся ИИ-системами (включая системы на основе нейросетей). Анализ рисков, кредитный скоринг, обнаружение мошенничества, прогноз рынков — всё это давно автоматизировано.

Медицина. Системы помощи в диагностике: компьютерная томография анализируется ИИ; патология (распознавание раковых клеток на гистологических снимках); чтение ЭКГ. Уровень точности часто превосходит средневзвешенного врача.

Дизайн, киноиндустрия и музыка. Генеративные модели изображений (Midjourney, Stable Diffusion, DALL-E, FLUX) изменили рекламу, графический дизайн, начальные стадии разработки игр и кино. Музыка (Suno, Udio): системы, генерирующие профессионального уровня композиции по текстовому запросу. Эти технологии в основном работают вместе с человеком — как мощные кисти, а не как замена художника.

Юриспруденция. Анализ контрактов, поиск прецедентов, подготовка проектов документов — области, где ИИ за минуту делает работу, на которую тратился час или день.

Государственное управление и общественные сервисы. Цифровизация госуслуг (МФЦ, портал Госуслуг в России) — лишь начало. ИИ-ассистенты для граждан, помощь в обработке обращений, переводы официальных документов — всё это уже внедряется.

4.5.10 Учиться учиться: главный навык XXI века

Каков же универсальный совет тем, кто только начинает свой профессиональный путь? Парадоксально, ИИ-революция предъявляет к человеку *гуманистические* требования.

Что точно не заменится.

- *Постановка задач и целеполагание.* ИИ может найти оптимальный путь к цели, но цель должен ставить человек.
- *Глубокое понимание области.* ИИ многое знает поверхностно; специалист, который понимает, *почему* та или иная идея работает, остаётся незаменим как заказчик и ревьюер ИИ-инструментов.
- *Творчество и нетривиальные идеи.* Революционные научные прорывы, художественные открытия, организация людей — области, где смелые, рисковые, неожиданные решения остаются за человеком.
- *Этика и ответственность.* Решения, влияющие на жизни людей, должны принимать люди.

Что точно полезно.

- *Уметь быстро учиться.* В мире, где появляются новые ИИ-инструменты каждый месяц, выигрывает тот, кто умеет быстро осваивать новое и применять его в своей работе.
- *Уметь работать с ИИ-инструментами.* «Промпт-инженерия» (искусство задавать правильные вопросы моделям) и навык критической оценки результатов ИИ становятся базовыми.
- *Понимать математику.* Парадоксально, но именно математика — наиболее «незаменимый» навык в эпоху ИИ. Кто понимает, как устроены модели, кто умеет читать их внутреннее устройство, кто может улучшать алгоритмы — тому открыты самые интересные позиции в этой отрасли.

> ИИ не заменяет человека — он становится *инструментом усиления* человеческого интеллекта. Самая успешная команда XXI века — это *человек, мастерски использующий ИИ*, более продуктивный, чем команда того же размера без ИИ — но *в десять или сто раз менее* продуктивный, чем человек, который понимает, что и как делают эти системы. Книга, которую вы держите в руках, — небольшой шаг в эту сторону.

4.5.11 Что мы прошли в этой главе

Вернёмся к карте маршрута, нарисованного в начале главы. Мы вышли из точки «опрос на выборах» и пришли в точку «AGI и нобелевские лауреаты». На пути — один и тот же универсальный методологический принцип, проходящий красной нитью:

Принцип максимума правдоподобия: запиши вероятностную модель, выпиши функцию правдоподобия данных, максимизируй по параметрам — и получишь оптимальную оценку.

В простейших задачах ОМП даёт выборочное среднее (раздел 4.1). В линейной модели — метод наименьших квадратов (раздел 4.2). В классификации — кросс-энтропию (раздел 4.3). В рекомендательных системах и автокодировщиках — квадратичную невязку в матричных моделях (раздел 4.4). А в современных языковых моделях — то же самое: предсказание следующего токена — это ОМП для условного распределения $\Pr(\text{токен}_t \mid \text{токены}_{1..t-1})$.

Вокруг этого универсального принципа вращаются три кита анализа данных:

- **Вероятностный кит** даёт постановку задачи: что мы хотим оценить и каковы шумы.
- **Оптимизационный кит** даёт инструмент: градиентный спуск (раздел 4.3.8) с эффективной реализацией градиента через backpropagation (раздел 4.3.9).
- **Линейно-алгебраический кит** даёт язык: матрично-векторные умножения внутри нейронных сетей (теорема 4.10); малоранговые аппроксимации (SVD, теорема 4.21); тензорные разложения.

Этот аппарат, изложенный в одной главе на двух десятках страниц, — один и тот же — лежит и в основе академической математической статистики XX века, и в основе многомиллиардной индустрии ИИ XXI века. Между ними — лишь шкала: данных стало в миллиарды раз больше, моделей в миллиарды раз сложнее, вычислений в миллиарды раз быстрее. Но *идеи* — те же.

Заключение главы

Современный ИИ — не магия, а аккуратная математика, помноженная на инженерию, помноженную на огромные вычислительные ресурсы. Все ключевые идеи — от ОМП до backpropagation, от SVD до трансформеров — доступны старшекласснику. Чем глубже понимать эти идеи, тем меньше страха перед ИИ и тем больше возможностей применять его себе на пользу.

Мы живём в очень интересное время. Возможно, ваше поколение — первое поколение в истории, для которого партнёром в каждой интеллектуальной работе будет ИИ. Желаем вам быть в этом партнёрстве *старшим* партнёром — тем, кто понимает, кто ставит цели и кто принимает решения. А ИИ — помощник, инструмент усиления, как в своё время паровая машина, электричество, компьютер, интернет. Каждая из этих технологий перекраивала мир — но не отменяла главного, что делает нас людьми.

Задачи для самостоятельной работы

1. Посмотрите запись 2-й партии матча AlphaGo–Ли Седоль (Wikipedia: AlphaGo vs Lee Sedol). 37-й ход AlphaGo — знаменитый «нечеловеческий» ход. Прочитайте о нём и подумайте, почему он удивил даже мировых экспертов го.
2. Вычислите по закону Чинчилла (4.56) ожидаемую кросс-энтропию для модели на $N = 10^{11}$ параметров, обученной на $D = 2 \cdot 10^{12}$ токенах. Сравните с моделью того же размера, но обученной на $D = 10^{11}$ токенах (типичный случай 2020 г.).
3. Найдите в открытых источниках три современных применения AlphaFold — на-

пример, в разработке новых лекарств. Опишите кратко.

4. Поразмышляйте над выбором профессии. Какие задачи вашей планируемой будущей деятельности можно делегировать ИИ уже сегодня? Какие — останутся за вами навсегда? Сформулируйте ответ в виде короткого эссе.
5. ⋆ Прочитайте оригинальную статью *Attention Is All You Need* (2017, Васвани и др.; arXiv:1706.03762). Попробуйте понять, как именно устроен механизм внимания $\mathrm{softmax}(QK^\top/\sqrt{d})\,V$. Запишите это формулой, объяснив роль матриц Q, K, V.
6. ⋆ Запустите локально маленькую LLM (например, через Ollama: модели Llama, Qwen, GigaChat-Mini, YandexGPT-Lite). Сравните качество её ответов на ваши вопросы. Какие задачи она выполняет хорошо? Где ошибается? Замечаете ли вы предсказуемые сценарии её слабости?

Часть IV

Дискретная математика и криптография

Глава 5

Элементы теории чисел и шифрование

«Математика — царица наук, а теория чисел — царица математики», — говорил Карл Фридрих Гаусс почти двести лет назад. Долгое время теория чисел считалась самой «чистой» из математических дисциплин — занимающейся ради красоты и никак не связанной с практикой. Сегодня, в эпоху Интернета, банковских карт и мессенджеров, эта наука оказалась едва ли не самой востребованной: без неё не отправить безопасное сообщение, не оплатить покупку и не зайти на сайт по защищённому соединению.

В этой главе мы пройдём путь от античного алгоритма Евклида до современных криптографических протоколов, обеспечивающих безопасность в Интернете. Один и тот же небольшой набор фактов о *делимости и сравнениях по модулю* окажется работающим везде — от шифрования RSA до структуры данных для подсчёта различных элементов в больших потоках.

Читатель встретит три «слоя» материала:

- **Теоретико-числовой фундамент**: алгоритм Евклида, малая теорема Ферма, функция Эйлера, тесты простоты — классические результаты, без которых не существует современной криптографии.
- **Криптографические протоколы**: симметричное и асимметричное шифрование, RSA и Диффи–Хеллман, электронная цифровая подпись, схемы электронного голосования.
- **Вероятностные алгоритмы**: универсальное хеширование, счётчики HyperLogLog, измерение «социального расстояния» в графах больших сетей.

Многие из изученных конструкций ежесекундно работают внутри устройств читателя: каждое HTTPS-соединение, каждый клик в банковском приложении, каждый запрос в поисковике — результат применения математических фактов, которым посвящена эта глава.

5.1 Элементы теории чисел

5.1.1 Делимость, остатки и сравнения по модулю

Целые числа — самые знакомые объекты в математике, и кажется, что про них трудно сказать что-то новое. Однако именно из вопросов о *делимости* целых чисел вырос целый раздел математики — **теория чисел**, — и именно её результаты сейчас работают внутри каждого мессенджера и банковской карты.

Определение 5.1

Говорят, что целое число a **делится** на целое число $b \neq 0$ (или что b **делит** a), если существует такое целое число q, что $a = b \cdot q$. Пишут $b \mid a$.

Если a не делится на b, то всё равно можно записать

$$a = b \cdot q + r, \qquad 0 \leq r < |b|.$$

Это знакомое со школы **деление с остатком**. Число q называется *неполным частным*, а r — **остатком от деления** a на b. Остаток обозначают также $a \bmod b$ (читается «a по модулю b»).

Пример 5.2

$17 = 5 \cdot 3 + 2$, поэтому $17 \bmod 5 = 2$.
$-17 = 5 \cdot (-4) + 3$, поэтому $(-17) \bmod 5 = 3$ (остаток — неотрицательное число!).

Простые числа — «атомы» делимости.

Определение 5.3

Натуральное число $p > 1$ называется **простым**, если оно делится только на единицу и на само себя. Натуральное $n > 1$, не являющееся простым, называется **составным**.

Первые простые числа: $2, 3, 5, 7, 11, 13, 17, 19, 23, 29, \ldots$. Простых чисел бесконечно много — это доказал ещё Евклид (доказательство мы повторим в п. 5.1.2).

Основная теорема арифметики говорит, что любое натуральное число $n > 1$ *единственным* (с точностью до порядка) способом раскладывается в произведение простых:

$$n = p_1^{\alpha_1} \cdot p_2^{\alpha_2} \cdots p_k^{\alpha_k}.$$

Сравнения по модулю. В каждой задаче, где важна только «остаточная» информация (например, день недели в зависимости от номера дня в году), удобно вместо чисел рассматривать их остатки от деления на некоторое фиксированное число.

Определение 5.4

Пусть $m \geq 2$ — натуральное число (**модуль**). Целые числа a и b называются **сравнимыми по модулю** m, если их разность $a - b$ делится на m. Пишут

$$a \equiv b \pmod{m}.$$

Равносильное определение: $a \equiv b \pmod{m}$ тогда и только тогда, когда a и b дают одинаковые остатки при делении на m.

Пример 5.5

$23 \equiv 9 \pmod{7}$, потому что $23 - 9 = 14 = 2 \cdot 7$.
В обоих случаях остаток от деления на 7 равен 2.

Сравнения — очень удобный язык, потому что они *ведут себя как обычные равенства*: их можно складывать, вычитать, умножать и возводить в степень.

Теорема 5.6. (свойства сравнений)

Если $a \equiv b \pmod{m}$ и $c \equiv d \pmod{m}$, то:

1. $a + c \equiv b + d \pmod{m}$;
2. $a - c \equiv b - d \pmod{m}$;
3. $a \cdot c \equiv b \cdot d \pmod{m}$;
4. $a^n \equiv b^n \pmod{m}$ для любого натурального n.

Идея доказательства. По определению $a - b = m \cdot k_1$ и $c - d = m \cdot k_2$. Тогда $(a+c) - (b+d) = m(k_1 + k_2)$ делится на m. Для умножения: $ac - bd = ac - bc + bc - bd = c(a - b) + b(c - d)$, что тоже делится на m. Возведение в степень — следствие умножения. □

С *делением* ситуация сложнее. Нельзя «просто так» поделить обе части сравнения — например, $6 \equiv 0 \pmod{6}$, но «сократив на 2», получим $3 \equiv 0 \pmod{6}$, что неверно. Когда деление законно — мы разберём чуть позже, после знакомства с алгоритмом Евклида.

Это интересно

Сравнения встречаются повсюду в жизни. Часы — арифметика по модулю 12 (или 24): если сейчас 10 часов, то через 5 часов будет $10 + 5 = 15 \equiv 3 \pmod{12}$. День недели — арифметика по модулю 7. Контрольная цифра ИНН, номер банковской карты, штрих-код книги — все они вычисляются как сумма цифр (с разными весами) по некоторому модулю.

5.1.2 Алгоритм Евклида

Одна из главных операций в теории чисел — нахождение **наибольшего общего делителя** (НОД) двух чисел.

Определение 5.7

Наибольший общий делитель чисел a и b (не равных одновременно нулю) — это наибольшее натуральное число, на которое делятся одновременно и a, и b. Обозначается $\text{НОД}(a, b)$.
Если $\text{НОД}(a, b) = 1$, то числа a и b называются **взаимно простыми**.

Как найти $\text{НОД}(a, b)$? Можно, например, разложить оба числа на простые множители и взять общие. Но при больших числах разложение на множители — очень тяжёлая задача (об этом мы поговорим в конце параграфа). К счастью, ещё в III веке до н. э. Евклид предложил способ, который работает несравнимо быстрее.

Геометрическая идея. Представим себе прямоугольник со сторонами a и b, причём $a > b$. Будем «выкладывать» в него квадраты со стороной b, пока это возможно. Когда останется полоска со сторонами b и $r = a \bmod b$, повторим процедуру для неё, и так далее. Когда выкладывание закончится точным квадратом — сторона этого последнего квадрата и есть $\text{НОД}(a, b)$ (рис. 5.1).

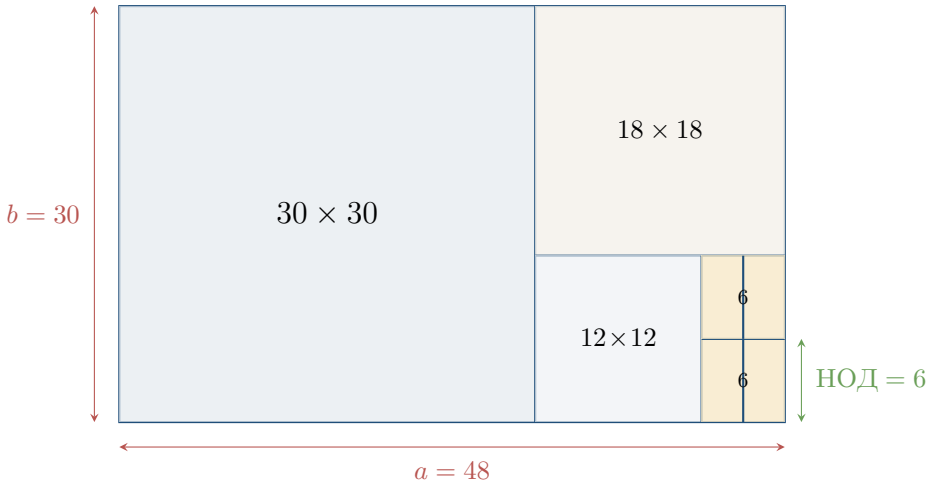


Рис. 5.1. Геометрическая иллюстрация алгоритма Евклида для пары (48, 30). Прямоугольник заполняется квадратами: 30×30, затем 18×18, затем 12×12 и, наконец, два квадрата 6×6. Сторона последнего квадрата равна $\text{НОД}(48, 30) = 6$.

Численно это соответствует последовательным делениям с остатком:

$$\begin{aligned} 48 &= 1 \cdot 30 + 18, \\ 30 &= 1 \cdot 18 + 12, \\ 18 &= 1 \cdot 12 + 6, \end{aligned}$$

$$12 = 2 \cdot 6 + 0 \implies \text{НОД}(48, 30) = 6.$$

Почему это работает? Ключевое наблюдение: если $a = bq + r$, то $\text{НОД}(a, b) = \text{НОД}(b, r)$. Действительно, любое число, делящее a и b, делит и $r = a - bq$; обратно, любое число, делящее b и r, делит и $a = bq + r$. Значит, у пар (a, b) и (b, r) один и тот же набор общих делителей — и, следовательно, один и тот же НОД. На каждом шаге второй аргумент уменьшается; раньше или позже он станет нулём, и тогда первое число — искомый НОД.

Алгоритм: нахождение НОД (Евклид)

Вход: натуральные числа a, b, $a \geq b > 0$.
Выход: $\text{НОД}(a, b)$.

1. Если $b = 0$, вернуть a.
2. Иначе вычислить $r = a \bmod b$ и повторить с парой (b, r).

На псевдокоде:

```
function gcd(a, b):
while b ≠ 0:
a, b := b, a mod b
return a
```

Скорость работы. Сколько шагов делает алгоритм? Можно доказать (теорема Ламе, 1844 г.): число делений в алгоритме Евклида для $\text{НОД}(a, b)$ не превосходит примерно $5 \log_{10} \min(a, b)$. Это значит, что для чисел в *сотни цифр* (а такие числа реально используются в криптографии) НОД находится за несколько сотен делений — доли секунды.

5.1.3 Расширенный алгоритм Евклида

Поразительный факт: Евклид нашёл не только НОД, но и способ *представить* его в виде «целочисленной линейной комбинации» исходных чисел.

Теорема 5.8. (соотношение Безу)

Для любых натуральных a, b существуют целые числа x, y (*не обязательно положительные*), такие что

$$a \cdot x + b \cdot y = \text{НОД}(a, b).$$

Эти x, y называются **коэффициентами Безу**. Найти их позволяет **расширенный алгоритм Евклида**: при делениях с остатком мы заодно «отслеживаем», как выражается каждый текущий остаток через исходные a и b.

Пример: те же 48 и 30. Прокрутим цепочку делений «снизу вверх», выражая 6 (наш НОД) через всё бо́льшие числа.

$$\begin{aligned} 6 &= 18 - 1 \cdot 12 && \text{(из третьего деления)} \\ &= 18 - 1 \cdot (30 - 1 \cdot 18) = 2 \cdot 18 - 1 \cdot 30 && \text{(подставили } 12 = 30 - 18\text{)} \\ &= 2 \cdot (48 - 1 \cdot 30) - 1 \cdot 30 = 2 \cdot 48 - 3 \cdot 30 && \text{(подставили } 18 = 48 - 30\text{)} \end{aligned}$$

Получили: $\boxed{2 \cdot 48 + (-3) \cdot 30 = 6}$. Коэффициенты Безу: $x = 2$, $y = -3$.

Пример 5.9

Проверим: $2 \cdot 48 - 3 \cdot 30 = 96 - 90 = 6 = \text{НОД}(48, 30)$. Совпадает.

Удобно вести расчёт в таблице:

шаг	r	q	x	y
0	48	—	1	0
1	30	—	0	1
2	18	1	1	-1
3	12	1	-1	2
4	6	1	2	-3
5	0	2	—	—

В каждой строке выполняются равенства $r = 48x + 30y$. На предпоследней строке (где $r = \text{НОД} = 6$) и считываются коэффициенты.

5.1.4 Решение линейных сравнений

Применим то, что получили, к решению уравнений в модулярной арифметике.

Обратный элемент по модулю. В обычной арифметике число, обратное к a, — это $1/a$: при умножении даёт единицу. По модулю m **обратным к** a называется такое целое a^{-1}, что

$$a \cdot a^{-1} \equiv 1 \pmod{m}.$$

Например, по модулю 7: $3 \cdot 5 = 15 = 2 \cdot 7 + 1 \equiv 1 \pmod 7$, значит, $3^{-1} \equiv 5 \pmod 7$.

Теорема 5.10

Обратный элемент к a по модулю m *существует* тогда и только тогда, когда $\text{НОД}(a, m) = 1$. При этом он находится с помощью расширенного алгоритма Евклида.

Идея. Если $\text{НОД}(a, m) = 1$, то по теореме 5.1.3 существуют целые x, y, такие что $ax + my = 1$. Это сравнение по модулю m принимает вид $ax \equiv 1 \pmod m$, то есть $x \equiv a^{-1} \pmod m$.

Обратно, если $a \cdot a^{-1} \equiv 1 \pmod m$, то $a \cdot a^{-1} - 1 = m \cdot k$ для некоторого k, откуда любой общий делитель a и m должен делить 1, — значит, $\text{НОД}(a, m) = 1$. □

Пример 5.11

Найдём $11^{-1} \pmod{26}$. Применяем расширенный алгоритм Евклида к $(26, 11)$:

$$\begin{aligned} 26 &= 2 \cdot 11 + 4, \\ 11 &= 2 \cdot 4 + 3, \\ 4 &= 1 \cdot 3 + 1, \\ 3 &= 3 \cdot 1 + 0. \end{aligned}$$

$\text{НОД} = 1$. Раскручиваем:

$$\begin{aligned} 1 &= 4 - 1 \cdot 3 = 4 - (11 - 2 \cdot 4) = 3 \cdot 4 - 11 \\ &= 3 \cdot (26 - 2 \cdot 11) - 11 = 3 \cdot 26 - 7 \cdot 11. \end{aligned}$$

Значит, $-7 \cdot 11 \equiv 1 \pmod{26}$, то есть $11^{-1} \equiv -7 \equiv 19 \pmod{26}$. Проверка: $11 \cdot 19 = 209 = 8 \cdot 26 + 1$.

Линейные сравнения. Уравнение вида

$$a \cdot x \equiv b \pmod m$$

называется **линейным сравнением**. Если $\text{НОД}(a, m) = 1$, его решение находится «домножением на обратный»: $x \equiv a^{-1} \cdot b \pmod m$.

5.1.5 Быстрое возведение в степень

В криптографии нужно уметь вычислять выражения вида $a^n \bmod m$ для огромных n (порядка 2^{1000}). Если в лоб умножать a на себя n раз — это никогда не закончится: даже за миллиард умножений в секунду на это ушло бы больше времени, чем существует Вселенная.

Спасает идея **«разделяй и властвуй»**, известная также как **бинарное возведение в степень**. Идея проста: разделим показатель пополам:

$$a^n = \begin{cases} \left(a^{n/2}\right)^2, & n \text{ чётно}, \\ a \cdot \left(a^{(n-1)/2}\right)^2, & n \text{ нечётно}. \end{cases}$$

Каждое такое деление пополам уменьшает показатель примерно вдвое, поэтому всего нужно около $\log_2 n$ шагов вместо n (рис. 5.2).

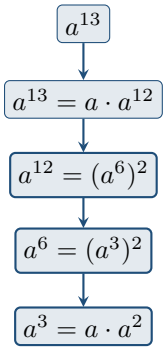


Рис. 5.2. Дерево бинарного возведения a^{13}: показатель уменьшается вдвое (с поправкой на чётность) на каждом шаге. Всего около $\log_2 13 \approx 4$ шагов.

Каждый «спуск» делит показатель примерно на 2, поэтому всего получается около $\log_2 n$ шагов. Для $n = 2^{1000}$ это *тысяча* умножений вместо 2^{1000}.

На псевдокоде:

```
function powmod(a, n, m):
result := 1
a := a mod m
while n > 0:
if n is odd:
result := (result * a) mod m
a := (a * a) mod m
n := n div 2
return result
```

Пример 5.12

Вычислим $7^{13} \bmod 23$. Запишем $13 = 1101_2$, то есть $13 = 8+4+1$. Значит, $7^{13} = 7^8 \cdot 7^4 \cdot 7^1$. Считаем последовательно по модулю 23:

$$7^1 = 7, \quad 7^2 = 49 \equiv 3, \quad 7^4 \equiv 3^2 = 9, \quad 7^8 \equiv 9^2 = 81 \equiv 81 - 3 \cdot 23 = 12.$$

Тогда $7^{13} \equiv 12 \cdot 9 \cdot 7 = 756 \equiv 756 - 32 \cdot 23 = 756 - 736 = 20 \pmod{23}$. Всего — около пяти умножений вместо тринадцати.

Та же идея в другом месте: числа Фибоначчи. Знакомая со школы последовательность Фибоначчи задаётся рекуррентно:

$$F_0 = 0, \quad F_1 = 1, \quad F_{n+1} = F_n + F_{n-1}.$$

На первый взгляд, чтобы вычислить F_n, нужно сделать n сложений — линейное время. Однако и здесь работает «разделяй и властвуй»! Запишем рекуррентное соотношение в матричной форме:

$$\begin{pmatrix} F_{n+1} \\ F_n \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} F_n \\ F_{n-1} \end{pmatrix}.$$

Обозначим матрицу как A. Применяя её n раз к вектору $(F_1, F_0)^\top = (1, 0)^\top$, получим:

$$\begin{pmatrix} F_{n+1} \\ F_n \end{pmatrix} = A^n \begin{pmatrix} 1 \\ 0 \end{pmatrix}.$$

Прямое доказательство — индукцией по n.

Важно

Тем самым задача сводится к возведению матрицы 2×2 в степень n. Но матрицы можно перемножать «делением пополам» точно так же, как числа! Сложность — $O(\log n)$ матричных умножений, то есть около $\log_2 n$ операций; вместо линейного числа шагов получается логарифмическое (рис. 5.3).

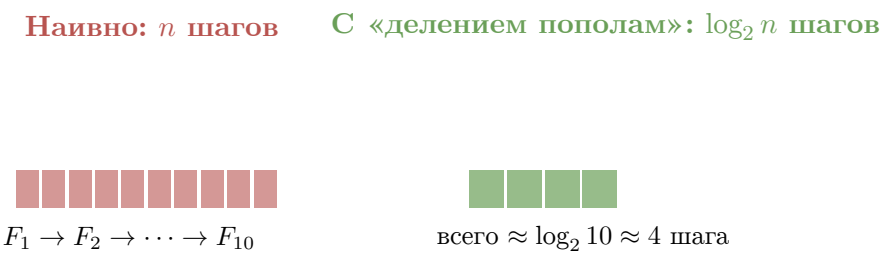


Рис. 5.3. Сравнение наивного и «разделяй и властвуй»-подхода к вычислению F_n.

Этот пример замечателен тем, что показывает: даже линейная на первый взгляд задача может «прятать» внутри себя структуру, ускоряющую решение экспоненциально.

5.1.6 Функция Эйлера и малая теорема Ферма

Чем дальше, тем интересней. Сейчас мы введём одну функцию, играющую ключевую роль в современной криптографии.

Определение 5.13

Функцией Эйлера $\varphi(n)$ называется количество натуральных чисел в отрезке $[1, n]$, взаимно простых с n.

Пример 5.14

$\varphi(1) = 1$, $\varphi(2) = 1$, $\varphi(6) = 2$ (а именно, числа 1 и 5), $\varphi(10) = 4$ (числа 1, 3, 7, 9).

Случай простого числа. Если p — простое число, то с p *не* взаимно прост лишь сам p среди чисел $1, 2, \ldots, p$. Значит,

$$\boxed{\varphi(p) = p - 1} \quad \text{для простого } p.$$

Случай произведения двух простых. Пусть $n = p \cdot q$, где p, q — различные простые. Какие числа из $[1, n]$ не взаимно просты с n? Только кратные p или кратные q. Кратных

p ровно q штук (это $p, 2p, \ldots, qp$); кратных q ровно p штук; единственное число, кратное обоим, — $pq = n$. По формуле включений-исключений:

$$\varphi(pq) = pq - p - q + 1 = (p-1)(q-1).$$

Это короткое наблюдение — сердце алгоритма RSA, который мы изучим в § 5.3.

Пример 5.15

$\varphi(15) = \varphi(3 \cdot 5) = 2 \cdot 4 = 8$. Действительно, взаимно простыми с 15 в отрезке $[1, 15]$ являются числа $\{1, 2, 4, 7, 8, 11, 13, 14\}$ — ровно 8 чисел.

В общем виде: для $n = p_1^{\alpha_1} \cdots p_k^{\alpha_k}$ выполняется $\varphi(n) = n \prod_{i=1}^{k} (1 - \frac{1}{p_i})$, но эта формула нам сегодня понадобится только в указанных двух частных случаях.

Малая теорема Ферма. Один из самых красивых результатов классической теории чисел.

Теорема 5.16. (малая теорема Ферма, 1640 г.)

Пусть p — простое число и a — целое, не делящееся на p. Тогда

$$a^{p-1} \equiv 1 \pmod{p}.$$

Равносильно: для любого целого a выполняется $a^p \equiv a \pmod{p}$.

Схема доказательства. Рассмотрим набор $\{a, 2a, 3a, \ldots, (p-1)a\}$ — $(p-1)$ ненулевых остатков, умноженных на a. Все они различны по модулю p: если $ia \equiv ja \pmod{p}$, то $a(i-j) \equiv 0 \pmod{p}$, а так как a не делится на p и $|i-j| < p$, получаем $i = j$.

Значит, набор $\{a, 2a, \ldots, (p-1)a\}$ — это в точности (с точностью до порядка) набор $\{1, 2, \ldots, p-1\}$ по модулю p. Перемножим всё:

$$a \cdot 2a \cdot 3a \cdots (p-1)a \;\equiv\; 1 \cdot 2 \cdot 3 \cdots (p-1) \pmod{p},$$

то есть $a^{p-1} \cdot (p-1)! \equiv (p-1)! \pmod{p}$. Поскольку $(p-1)!$ взаимно прост с p, его можно «сократить» — получим $a^{p-1} \equiv 1 \pmod{p}$. □

Обобщение — **теорема Эйлера**: $a^{\varphi(n)} \equiv 1 \pmod{n}$ для любого a, взаимно простого с n. Доказывается аналогично, только вместо чисел $1, \ldots, p-1$ берутся все числа из $[1, n]$, взаимно простые с n.

Это интересно

Пьер Ферма, по профессии юрист, посвящал математике вечера. В 1640 году он написал письмо своему другу с этим утверждением, заметив: «Я бы прислал доказательство, если бы не боялся, что оно слишком длинное». Первое известное полное доказательство опубликовал Леонард Эйлер — спустя сто лет.

5.1.7 Сколько вокруг простых чисел?

Прежде чем переходить к криптографии, ответим на важный практический вопрос: *часто ли встречаются простые числа?* Если простых чисел мало — ими нельзя будет полноценно пользоваться.

Теорема 5.17. (Чебышёв – Адамар – Валле-Пуссен, асимптотический закон распределения пр

Пусть $\pi(N)$ — количество простых чисел, не превосходящих N. Тогда

$$\pi(N) \sim \frac{N}{\ln N} \qquad \text{при } N \to \infty.$$

В пересчёте на «вероятность»: если случайно взять натуральное число вокруг N, оно окажется простым примерно с шансом $1/\ln N$. Например, среди 1000-значных чисел простыми оказывается примерно одно из $\ln 10^{1000} = 1000 \ln 10 \approx 2300$. Это совсем немало (рис. 5.4).

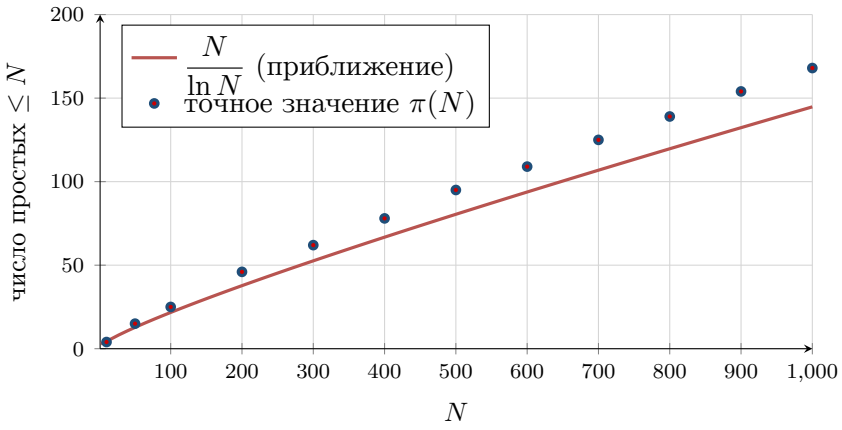


Рис. 5.4. Закон распределения простых чисел: количество простых чисел до N ведёт себя как $N/\ln N$.

Практический смысл: чтобы найти простое число длиной, скажем, 1024 бита для RSA-ключа, нужно перебрать в среднем порядка 700 случайных чисел и каждое проверить на простоту. И это нас приводит к следующему вопросу.

5.1.8 Проверка простоты и задача факторизации

Два «двойственных» вопроса:

Проверка простоты: дано число n — простое оно или составное?

Факторизация: дано составное n — разложить его на простые множители.

Может показаться, что эти задачи примерно одинаковой сложности. Но это не так: *они принципиально разной сложности*, и именно на этом контрасте основана вся современная криптография.

Тест Ферма. Если p простое, то по теореме 5.1.6 для любого a, не кратного p, выполняется $a^{p-1} \equiv 1 \pmod p$. Возьмём это как *пробный признак*: если для случайного a выполнено $a^{n-1} \not\equiv 1 \pmod n$, то n заведомо *не простое*. Если же $a^{n-1} \equiv 1 \pmod n$ — то n простое *с большой вероятностью* (но иногда бывают и составные числа, проходящие тест — так называемые **числа Кармайкла**).

Тест Миллера–Рабина. Усовершенствованный вариант теста Ферма. Он использует ту же идею, но дополнительно «отсеивает» числа Кармайкла. Принципиальные свойства:

- это **вероятностный** тест: при отрицательном ответе число точно составное; при положительном — простое с вероятностью ошибки не больше $1/4$ за один проход;
- делая k независимых проходов, мы уменьшаем вероятность ошибки до 4^{-k}. Уже при $k = 20$ это меньше, чем 10^{-12};

- работает за время $O(k \cdot \log^3 n)$ — то есть **полиномиально** от числа цифр.

Тест AKS (2002 г.). Манидра Агравал, Нираж Каял и Нитин Саксена — индийские математики — впервые предъявили *детерминированный* полиномиальный алгоритм проверки простоты. На практике он работает медленнее Миллера–Рабина, но важен как теоретическое достижение: показал, что класс задач, разрешимых за полиномиальное время, включает в себя проверку простоты.

Это интересно

Открытие AKS-алгоритма стало сенсацией: один из руководителей — Нитин Саксена — был на тот момент аспирантом первого года. Результат опубликован в 2004 году в одном из самых престижных математических журналов — «Annals of Mathematics» — и был назван «алгоритмом тысячелетия» в области теории чисел.

А что с факторизацией? Тут картина совсем другая. Лучший известный сегодня алгоритм — *общий метод решета числового поля* — работает за время, растущее как

$$\exp\bigl(c \cdot \sqrt[3]{(\ln n)(\ln \ln n)^2}\bigr),$$

что по сути **субэкспоненциально**, но *не полиномиально*. Для 1024-битных n это означает миллиарды лет работы суперкомпьютера.

Важно

Колоссальный разрыв в сложности: *проверить, простое ли число* — легко (полиномиально), а *разложить число на простые множители* — невероятно трудно (субэкспоненциально). Именно этот разрыв и эксплуатируется в RSA и других криптосистемах. Безопасность интернета прямо опирается на то, что у человечества пока нет быстрого алгоритма факторизации.

А если появится квантовый компьютер? В 1994 году Питер Шор предложил квантовый алгоритм, факторизующий числа за полиномиальное время. Реальных квантовых компьютеров достаточного масштаба пока нет (на 2025 год удавалось факторизовать лишь крошечные числа в качестве демонстрации). Но если такие компьютеры будут построены — современный RSA рухнет, и потребуются принципиально новые криптографические схемы (это направление называется *постквантовая криптография*).

Что мы получили

Подведём итог. На вопрос «зачем нам всё это в учебнике информатики» мы готовы ответить: только что мы построили *инструменты*, без которых не работает ни один защищённый сайт. А именно:

- **сравнения по модулю** — язык, на котором будем описывать шифрование;
- **расширенный алгоритм Евклида** — быстро ищет обратный элемент по модулю;
- **быстрое возведение в степень** — позволяет работать с гигантскими степенями;
- **малая теорема Ферма** и **функция Эйлера** — ключ к схеме RSA;
- **разрыв в сложности** между проверкой простоты и факторизацией — источник «безопасности».

В следующих параграфах мы соберём из этих кирпичиков работающие криптографические протоколы.

Задачи для самостоятельной работы

1. Найдите НОД$(231, 165)$ алгоритмом Евклида. Представьте НОД в виде $231x + 165y$.
2. Найдите $5^{-1} \pmod{37}$. Используйте расширенный алгоритм Евклида.
3. Вычислите $3^{100} \bmod 7$, пользуясь малой теоремой Ферма.
4. Сколько существует чисел в отрезке $[1, 100]$, взаимно простых с 100? (Подсказка: $100 = 2^2 \cdot 5^2$, но вы можете использовать формулу для $\varphi(p^2)$ или прямой подсчёт.)
5. Запишите быстрое возведение в степень в виде программы на знакомом вам языке программирования.
6. * Докажите, что для составного n всегда найдётся a из $[2, n-1]$, для которого $a^{n-1} \not\equiv 1 \pmod{n}$ — кроме чисел Кармайкла. Найдите наименьшее число Кармайкла.
7. * С помощью матричной формулы для чисел Фибоначчи вычислите F_{20}.

5.2 Шифрование: от тайнописи к математической задаче

5.2.1 Зачем людям шифры

Каждый из вас, заходя на любой сайт по адресу, начинающемуся с `https://`, доверяет ему *шифрование*. Браузер и сервер обмениваются миллионами сообщений, но никто посторонний прочесть их не может. В этом параграфе мы поймём, какие задачи решает шифрование, какие общие схемы существуют и почему именно теория чисел оказалась его математической основой.

Античность. Первые шифры были чисто механическими. В Древней Греции (V век до н. э.) воины Спарты использовали **скиталу** — цилиндр определённого диаметра, на который наматывалась лента с текстом. Без точно такой же палочки прочесть сообщение было нельзя. Юлий Цезарь, по свидетельству Светония, в переписке со своими полководцами сдвигал каждую букву алфавита на три позиции: `A` → `D`, `B` → `E`, и так далее. Этот **шифр Цезаря** — простейший *шифр замены* (рис. 5.5).

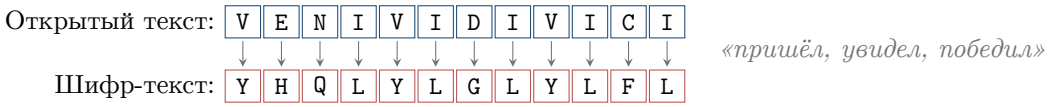


Рис. 5.5. Шифр Цезаря со сдвигом 3: каждая буква заменяется на букву через три позиции в алфавите.

Любой такой шифр легко взламывается: букв в алфавите немного, и можно просто перебрать все 25 возможных сдвигов. Более изобретательные шифры (например, **шифр Виженера** XVI века) использовали *переменный* сдвиг, заданный ключевым словом. Они продержались несколько столетий, пока в середине XIX века Чарлз Бэббидж и независимо Фридрих Касиски не предложили универсальный метод их взлома.

XX век: машины. Во время Второй мировой войны на сцену вышли электромеханические шифровальные машины — знаменитая немецкая **Энигма**, японская «Пурпурная» (Purple), советская «Фиалка». Их взлом стал отдельной героической страницей: в Великобритании Алан Тьюринг возглавлял проект *Bletchley Park*, где была сконструирована

машина «Бомба», взламывавшая Энигму. Считается, что этот успех сократил войну в Европе на 2–4 года.

Это интересно

Деятельность Алана Тьюринга и его коллег была настолько секретной, что заслуги команды Bletchley Park были рассекречены только в 1970-х годах — то есть через 25 лет после конца войны. Сам Тьюринг при жизни так и не узнал о признании. Тьюринг также считается одним из отцов теоретической информатики — именно его именем названа престижная Премия Тьюринга, аналог «Нобелевской премии в информатике».

5.2.2 Общая схема симметричного шифрования

Все шифры до 1970-х годов имели одну общую особенность: для шифрования и расшифрования использовался *один и тот же* ключ. Такие шифры называются **симметричными** (или *secret-key*) (рис. 5.6).

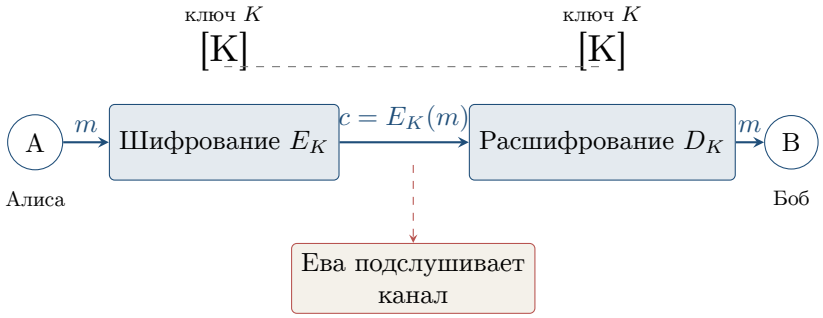


Рис. 5.6. Симметричное шифрование: один общий секретный ключ K позволяет и зашифровать, и расшифровать. Подслушивающая Ева не должна узнать K.

Симметричные шифры быстры и до сих пор широко применяются (например, стандарт *AES* — то, чем шифруется содержимое любого Telegram-сообщения после установки соединения). Но у них есть фундаментальная *проблема распределения ключей*: как Алисе и Бобу договориться о ключе, если они никогда раньше не встречались, а связь между ними прослушивается?

Важно

Парадокс симметричного шифрования. Чтобы безопасно обмениваться сообщениями, нужен общий ключ. Чтобы безопасно передать ключ — нужен другой общий ключ. И так до бесконечности.

В большом мире (миллиарды пользователей интернета) задача распределения ключей становится непреодолимой: каждой паре нужен свой ключ, всего $\binom{n}{2} \approx n^2/2$ ключей.

5.2.3 Революция 1976 года: открытый ключ

В 1976 году произошёл прорыв, перевернувший всю криптографию. Уитфилд Диффи и Мартин Хеллман предложили принципиально новую идею: **асимметричное шифрование**, или **криптография с открытым ключом**.

Каждый пользователь имеет *пару* ключей:

- **открытый ключ** (*public key*) — его можно (и нужно) сообщать всем: публиковать на сайте, отправлять незашифрованно;
- **закрытый ключ** (*private key*) — он хранится в строжайшем секрете у владельца.

Ключевое свойство: что зашифровано открытым ключом — расшифровать может только владелец соответствующего закрытого ключа (рис. 5.7).

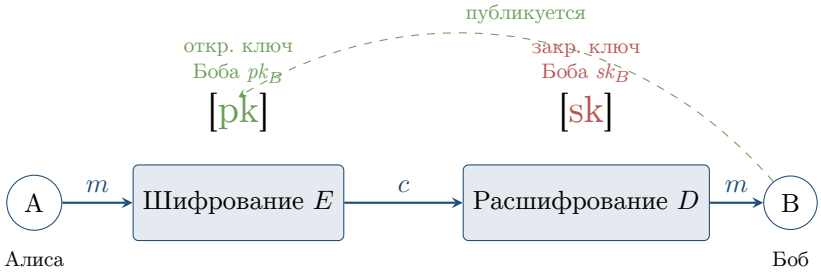


Рис. 5.7. Асимметричное шифрование: Алиса использует *открытый* ключ Боба, который тот заранее опубликовал. Расшифровать может только Боб — его *закрытым* ключом.

Заметьте: больше нет проблемы распределения ключей. Алиса не должна заранее ничего секретного передавать Бобу — она просто берёт его открытый ключ из открытого источника.

5.2.4 Односторонние функции и функции с секретом

В чём математическая суть асимметричной криптографии? В существовании **односторонних функций**.

Определение 5.18

Односторонняя функция (*one-way function*) — это функция $f\colon X \to Y$, обладающая двумя свойствами:

- *Прямое вычисление просто:* зная x, легко (полиномиально быстро) вычислить $f(x)$.
- *Обращение трудно:* зная $y = f(x)$, восстановить x без какой-либо дополнительной информации — задача *практически* нерешаемая (требует экспоненциального времени).

Это интересно

Существование «настоящих» односторонних функций математически не доказано (это связано с одной из главных нерешённых задач — проблемой $\mathbf{P} \neq \mathbf{NP}$). На практике мы используем функции, для которых обращение *считается* трудным, потому что человечество за десятилетия так и не научилось делать это быстро.

Хороший бытовой образ: *разбить вазу* — легко, *собрать обратно* — очень тяжело. Или: *перемешать колоду карт* — мгновенная операция, *восстановить исходный порядок* — немыслимо.

Главные примеры из теории чисел:

- **Умножение vs. факторизация:** перемножить $p \cdot q$ — мгновенно; разложить произведение на простые — трудно. Это база RSA.
- **Дискретный логарифм:** возвести g в степень x по модулю p — быстро; зная $g^x \bmod p$, восстановить x — очень трудно. Это база схемы Диффи–Хеллмана.

Функции с секретом. Просто односторонней функции для асимметричного шифрования мало: нужно, чтобы у *кого-то одного* (владельца) была возможность всё-таки обратить функцию — именно для расшифрования сообщений (рис. 5.8).

Определение 5.19

Функция с секретом (*trapdoor function*) — односторонняя функция f, для которой существует дополнительная секретная информация t (*лазейка*, *trapdoor*), знание которой делает обращение f полиномиально быстрым.

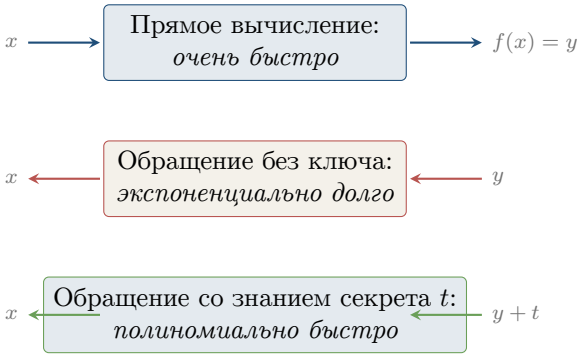


Рис. 5.8. Функция с секретом ведёт себя как односторонняя, но «открывается» при знании дополнительной информации t.

Можно представить себе сейф с замком: запереть сейф (положить документ внутрь) может каждый, но открыть — только владелец ключа.

Идея использования. Открытый ключ *задаёт* функцию с секретом f. Кто угодно может вычислить $c = f(m)$ — зашифровать сообщение. Для расшифрования нужно обратить f, что без секрета t — безнадёжно. У владельца секрета t это вычисление быстрое. Тем самым роль закрытого ключа — хранить лазейку t.

5.2.5 Где криптография работает прямо сейчас

Чтобы дальнейший материал не казался академическим упражнением, перечислим, в каких знакомых каждому ситуациях работают (часто прямо сейчас, пока вы это читаете) описанные в этой главе идеи.

- **Загрузка любого сайта** (`https`). Браузер договаривается с сервером о ключе через схему Диффи–Хеллмана, после чего весь трафик шифруется симметричным алгоритмом (*AES*).
- **Мессенджеры** (WhatsApp, Telegram, Signal). Используют так называемое *сквозное шифрование* (*end-to-end*): сервер мессенджера видит лишь шифр-текст. Согласование ключей — асимметричное.
- **Банковские карты и оплата.** Транзакция от карты проходит через цепочку обмена цифровыми подписями.
- **Госуслуги и налоговая.** Подача декларации сопровождается **электронной цифровой подписью** (ЭЦП), о которой подробно — в § 5.4.
- **Цифровой пропуск/QR-код** (от билета в кино до подтверждения вакцинации). Содержит подпись эмитента, проверяемую его открытым ключом.
- **Криптовалюты.** Биткойн и его «родственники» — огромная распределённая система с электронными подписями: кошелёк — это просто пара (открытый ключ, закрытый ключ).

Что должно быть у «правильной» криптосистемы. Когда мы строим конкретную схему, надо проверять, что она удовлетворяет следующим естественным требованиям:

1. *Корректность.* Если Алиса честно зашифровала открытым ключом Боба, Боб всегда расшифрует — ровно исходное сообщение.
2. *Стойкость.* Подслушивающий, видя только шифр-текст и открытый ключ, не может практически восстановить исходное сообщение.
3. *Эффективность.* Шифрование и расшифрование выполняются быстро (полиномиально).
4. *Защита ключа.* Закрытый ключ не должен «утекать» из открытого — даже зная открытый ключ, нельзя восстановить закрытый.

В следующих параграфах мы построим конкретные схемы — RSA и Диффи–Хеллман — и убедимся, что они этим требованиям удовлетворяют.

Задачи для самостоятельной работы

1. Зашифруйте шифром Цезаря со сдвигом 5 слово `INFORMATIKA`.
2. Почему симметричное шифрование плохо подходит для миллиарда пользователей? Подсчитайте, сколько ключей надо хранить.
3. Объясните своими словами, в чём состоит свойство «односторонности» функции.
4. * Знаменитая Энигма имела примерно 10^{20} возможных стартовых установок. Хватило бы перебора, если перебирать миллион вариантов в секунду?
5. Приведите три бытовые операции, которые ведут себя как «односторонние функции».

5.3 Криптосистемы RSA и Диффи–Хеллмана

В этом параграфе мы построим две настоящие, работающие сегодня схемы асимметричной криптографии. Обе появились почти в одно время — между 1976 и 1978 годами — и обе опираются на результаты предыдущего параграфа.

5.3.1 Криптосистема RSA

Идея была опубликована в 1978 году тремя сотрудниками Массачусетского технологического института — *R*ональдом Райвестом, *Ш*амиром (Ади Шамир) и *A*длеманом (Леонард Адлеман). Их фамилии и дали название системе — RSA.

Генерация ключей. Каждый пользователь, желающий получать зашифрованные сообщения, выполняет одноразовую процедуру.

Алгоритм: генерация ключей RSA

1. Выбрать два больших случайных *различных* простых числа p и q (на практике — по 1024 бита каждое).
2. Вычислить $n = p \cdot q$ и $\varphi(n) = (p-1)(q-1)$.
3. Выбрать целое e, взаимно простое с $\varphi(n)$ (часто берут $e = 65537$).

4. С помощью расширенного алгоритма Евклида найти d такое, что
$$e \cdot d \equiv 1 \pmod{\varphi(n)}.$$

5. **Открытый ключ:** пара (n, e) — публикуется.
Закрытый ключ: число d — хранится в секрете.
Лишние данные: числа $p, q, \varphi(n)$ безопасно уничтожаются.

Шифрование. Алиса хочет послать сообщение m Бобу. Сообщение представлено числом из отрезка $[0, n-1]$ (длинное сообщение разбивается на блоки). Алиса берёт открытый ключ Боба (n, e) и вычисляет:
$$c \;=\; m^e \bmod n.$$

Число c называется **шифр-текстом** и передаётся открыто.

Расшифрование. Боб, получив c, использует свой закрытый ключ d и вычисляет:
$$m' \;=\; c^d \bmod n.$$

Утверждается: $m' = m$ (рис. 5.9).

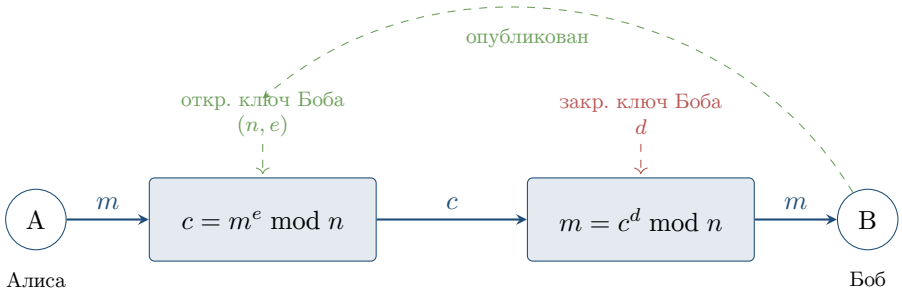


Рис. 5.9. Схема работы RSA: шифрование — возведение в степень e по модулю n, расшифрование — возведение в степень d. Обе операции эффективны благодаря быстрому возведению в степень.

5.3.2 Почему RSA работает: корректность

Теорема 5.20. (корректность RSA)
В обозначениях процедуры RSA, для любого $m \in [0, n-1]$ выполняется
$$(m^e)^d \equiv m \pmod{n}.$$

Доказательство. По построению $e \cdot d \equiv 1 \pmod{\varphi(n)}$, значит, найдётся целое k, такое что $e \cdot d = 1 + k \cdot \varphi(n)$. Подставим:
$$m^{ed} = m^{1+k\varphi(n)} = m \cdot \left(m^{\varphi(n)}\right)^k.$$

Случай 1. НОД$(m, n) = 1$. Тогда по теореме Эйлера $m^{\varphi(n)} \equiv 1 \pmod{n}$, и
$$m^{ed} \equiv m \cdot 1^k = m \pmod{n}.$$

Случай 2. НОД$(m, n) > 1$. Поскольку $n = pq$ и p, q простые, это значит, что m делится либо на p, либо на q (но не на оба — иначе $m \geq pq = n$, чего быть не может). Рассмотрим,

например, $p \mid m$. Тогда $m \equiv 0 \pmod p$, и обе части сравнения $m^{ed} \equiv m \pmod p$ обращаются в ноль — утверждение выполнено.

С другой стороны, НОД$(m, q) = 1$, поэтому по малой теореме Ферма $m^{q-1} \equiv 1 \pmod q$. А значит, $m^{(p-1)(q-1)} = m^{\varphi(n)} \equiv 1 \pmod q$, и

$$m^{ed} = m \cdot \left(m^{\varphi(n)}\right)^k \equiv m \cdot 1^k = m \pmod q.$$

Получили $m^{ed} \equiv m$ одновременно по модулям p и q. По китайской теореме об остатках (или просто потому, что разность $m^{ed} - m$ делится и на p, и на q, а значит, на их произведение n) сравнение $m^{ed} \equiv m \pmod n$ выполнено. □

Учебный пример (маленькие числа). Полноценный RSA использует тысячебитные числа, но для понимания работает и крошечный пример.

Пример 5.21. генерация и применение ключа RSA

Пусть $p = 11$, $q = 23$. Тогда:

- $n = 11 \cdot 23 = 253$, $\varphi(n) = 10 \cdot 22 = 220$.
- Выберем $e = 7$. Проверим взаимную простоту: НОД$(7, 220) = 1$.
- Найдём $d = 7^{-1} \bmod 220$. Расширенным алгоритмом Евклида:

  $$220 = 31 \cdot 7 + 3, \quad 7 = 2 \cdot 3 + 1.$$

  Раскручиваем: $1 = 7 - 2 \cdot 3 = 7 - 2(220 - 31 \cdot 7) = 63 \cdot 7 - 2 \cdot 220$. Значит, $d = 63$ (проверка: $7 \cdot 63 = 441 = 2 \cdot 220 + 1$).
- Открытый ключ: $(253, 7)$. Закрытый ключ: $d = 63$.

Шифрование. Пусть $m = 50$. Считаем $c = 50^7 \bmod 253$ быстрым возведением:

$$50^2 = 2500 \equiv 2500 - 9 \cdot 253 = 223, \qquad 50^4 \equiv 223^2 = 49\,729.$$

Делим $49\,729/253 \approx 196{,}5$, $196 \cdot 253 = 49\,588$, остаток 141. Значит, $50^4 \equiv 141$. Далее $50^7 = 50^4 \cdot 50^2 \cdot 50 \equiv 141 \cdot 223 \cdot 50 \bmod 253$. Считаем: $141 \cdot 223 = 31\,443 = 124 \cdot 253 + 71$, значит $\equiv 71$. Затем $71 \cdot 50 = 3550 = 14 \cdot 253 + 8$, значит $\equiv 8$. Итак, $c = 8$.
Расшифрование. Боб считает $c^d = 8^{63} \bmod 253$. Вместо длинных вычислений «вручную» можно использовать малую теорему Ферма по модулям 11 и 23 раздельно, а затем китайскую теорему об остатках — именно так и поступают на практике. Ответ: 50.

Это интересно

Удивительный исторический парадокс: ровно та же схема была независимо открыта в 1973 году британским математиком Клиффордом Коксом, работавшим в Британском управлении правительственной связи (GCHQ). Но работа была засекречена и рассекречена только в 1997 году. К этому моменту Райвест, Шамир и Адлеман уже стали всемирно известными, основали компанию RSA Security, а Шамир получил Премию Тьюринга. Типичный пример того, как секретность тормозит развитие науки.

5.3.3 Почему RSA безопасен

Важно

Стойкость RSA опирается на предположение: *задача факторизации n практически нерешаема при больших n.*

Действительно, если злоумышленник умеет факторизовать n, он восстанавливает p, q, затем $\varphi(n) = (p-1)(q-1)$ и наконец $d = e^{-1} \bmod \varphi(n)$ — получит закрытый ключ. Обратное (что взлом RSA эквивалентен факторизации) также считается верным, но строго не доказано: это одна из открытых проблем в криптографии.

Заметим, что *знать* n *и* e (открытый ключ) — ещё далеко не значит знать $\varphi(n)$. Знание $\varphi(n)$ и знание разложения $n = pq$ — одна и та же информация: ведь $p+q = n-\varphi(n)+1$, а $p-q = \sqrt{(p+q)^2 - 4n}$. То есть «нахождение $\varphi(n)$» и «факторизация n» — задачи одинаковой сложности.

Почему нужны именно *большие* простые числа? Современный рекомендуемый размер n — 2048 или даже 4096 бит. Чтобы взломать 2048-битный RSA, нужно факторизовать число длиной около 617 десятичных цифр; лучший известный алгоритм потратит на это больше времени, чем существует Вселенная.

Пример 5.22. границы практической факторизации

В 2020 году группе исследователей удалось факторизовать число RSA-250 (250 десятичных цифр $\approx$ 829 бит). Заняло это около 2700 ядро-лет вычислений — то есть один компьютер с 2700 ядрами работал бы целый год. Это подтверждает, что 1024-битный RSA скоро будет «на грани», а 2048-битный — надолго безопасен.

Проверим выполнение требований из § 5.2.

1. *Корректность* — доказана теоремой выше.
2. *Стойкость* — опирается на сложность факторизации.
3. *Эффективность* — шифрование/расшифрование сводятся к возведению в степень по модулю, что мы умеем делать полиномиально (см. § 5.1.5).
4. *Защита ключа* — знание e, n не даёт быстрого способа найти d без факторизации.

5.3.4 Протокол Диффи–Хеллмана

Уитфилд Диффи и Мартин Хеллман решали несколько другую (хотя и родственную) задачу: *договориться* о секретном ключе по открытому каналу, — не передавая по нему ничего секретного. Их работа 1976 года так и называется: «Новые направления в криптографии».

Идея на пальцах: краски. Прежде чем вводить формулы, разберём наглядную аналогию (рис. 5.10).

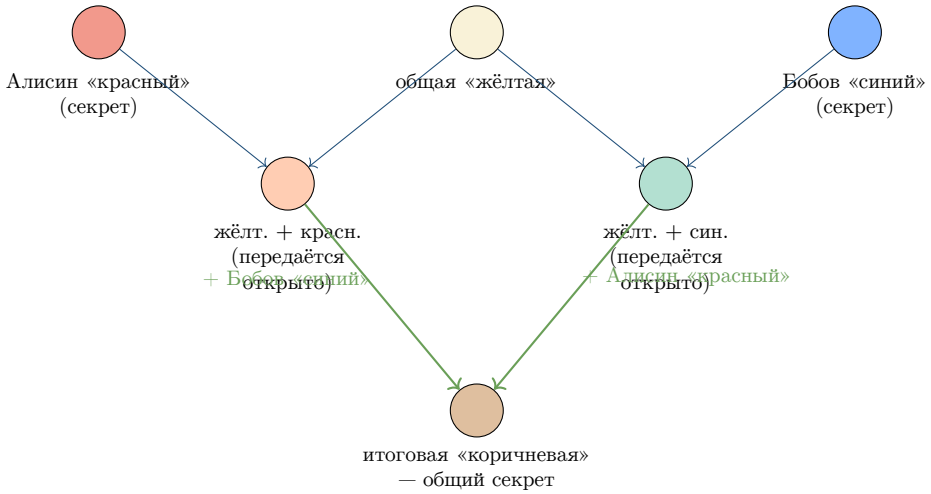


Рис. 5.10. Аналогия Диффи–Хеллмана со смешиванием красок. Смешать — легко, разделить смесь обратно на исходные цвета — невозможно. Алиса и Боб приходят к одному цвету разными путями.

Алиса и Боб условились о «жёлтой» базовой краске (известна всем). Каждый выбрал свою секретную краску (Алиса — красный, Боб — синий) и публично передал смесь «своя + жёлтая». Подслушивающий видит обе смеси, но не может разложить их обратно. А Алиса и Боб, добавив к чужой смеси свою секретную краску, приходят к одинаковому итогу: жёлт. + красн. + син.

Математическая реализация. Роль «смешивания» играет **дискретное возведение в степень** по модулю простого числа (рис. 5.11).

Алгоритм: протокол Диффи–Хеллмана

Подготовка (общедоступно): большое простое число p и целое g, $1 < g < p$.

1. Алиса выбирает случайный секрет $a \in [1, p-2]$ и посылает Бобу $A = g^a \bmod p$.
2. Боб выбирает случайный секрет $b \in [1, p-2]$ и посылает Алисе $B = g^b \bmod p$.
3. Алиса вычисляет общий ключ: $K = B^a \bmod p$.
4. Боб вычисляет общий ключ: $K = A^b \bmod p$.

Результат. Алиса и Боб получили один и тот же ключ

$$K = g^{ab} \bmod p,$$

который никогда не передавался по каналу связи.

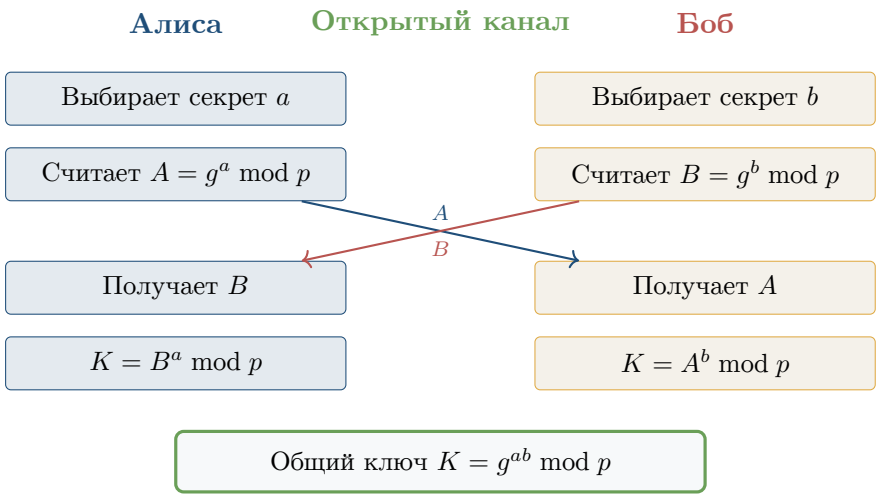


Рис. 5.11. Диаграмма обмена в протоколе Диффи–Хеллмана.

Учебный пример.

Пример 5.23. обмен ключами Диффи–Хеллмана

Пусть $p = 23$, $g = 5$.

- Алиса выбирает $a = 6$, отправляет $A = 5^6 \bmod 23$. Считая последовательно по таблице степеней пятёрки:

$$5^1 \equiv 5,\ 5^2 \equiv 2,\ 5^3 \equiv 10,\ 5^4 \equiv 4,\ 5^5 \equiv 20,\ 5^6 \equiv 8,$$

  получаем $A = 8$.

- Боб выбирает $b = 15$, отправляет $B = 5^{15} \bmod 23$. Продолжая таблицу:

$$5^7 \equiv 17,\ 5^8 \equiv 16,\ 5^9 \equiv 11,\ 5^{10} \equiv 9,\ 5^{11} \equiv 22,$$

$$5^{12} \equiv 18,\ 5^{13} \equiv 21,\ 5^{14} \equiv 13,\ 5^{15} \equiv 19,$$

  получаем $B = 19$.

- Алиса вычисляет общий ключ $K = B^a = 19^6 \bmod 23$. Замечая, что $19 \equiv -4 \pmod{23}$, имеем $(-4)^6 = 4^6 = 4096 = 178 \cdot 23 + 2$, значит, $K = 2$.

- Боб вычисляет $K = A^b = 8^{15} \bmod 23$. Но $8 = 5^6$, значит, $8^{15} = 5^{90}$. По малой теореме Ферма $5^{22} \equiv 1 \pmod{23}$, а $90 = 4 \cdot 22 + 2$, поэтому $5^{90} \equiv 5^2 = 2 \pmod{23}$. Получаем $K = 2$.

Согласование удалось: общий ключ $K = 2$. И это притом, что числа $a = 6$ и $b = 15$ никогда не передавались по каналу!

Безопасность Диффи–Хеллмана. Чтобы подслушивающий, видя g, p, A, B, мог найти общий ключ $K = g^{ab}$, ему нужно сначала восстановить хотя бы один из секретов a или b, — то есть решить уравнение

$$g^a \equiv A \pmod{p}$$

относительно a при известных g, A, p. Эта задача называется **задачей дискретного логарифма** (в группе $\mathbb{Z}_p^*$).

Важно

Для всех известных классических алгоритмов задача дискретного логарифма по модулю большого простого — субэкспоненциальная (примерно той же сложности, что и факторизация). Однако *по модулю эллиптической кривой* — значительно сложнее, и поэтому современные мессенджеры обычно используют вариант Диффи–Хеллмана не по модулю числа, а на эллиптических кривых (*ECDH*). Идея та же.

5.3.5 Атака «человек посередине»

И RSA, и Диффи–Хеллман в чистом виде уязвимы к одной атаке. Представим, что между Алисой и Бобом «вклинился» злоумышленник — Мэллори (*Man-in-the-Middle*) (рис. 5.12).

Рис. 5.12. Атака «человек посередине»: Мэллори создаёт две независимые сессии, в каждой выдавая себя за противоположную сторону.

Мэллори перехватывает A от Алисы, заменяет на свой A' и пересылает Бобу. С Бобом он устанавливает один ключ, с Алисой — другой. Затем расшифровывает каждое сообщение, читает (и при желании меняет), снова шифрует и передаёт дальше. Внешне Алиса и Боб ничего не замечают.

Защита. Нужна *аутентификация* — способ убедиться, что собеседник действительно тот, за кого себя выдаёт. В реальных системах это решается через **цифровые сертификаты** и **электронную цифровую подпись**. О них — следующий параграф.

Задачи для самостоятельной работы

1. В системе RSA дано $p = 7$, $q = 13$, $e = 5$. Найдите n, $\varphi(n)$ и d.
2. Используя ключ из задачи 1, зашифруйте сообщение $m = 9$.
3. В Диффи–Хеллмане положите $p = 17$, $g = 3$, $a = 5$, $b = 7$. Найдите A, B и общий ключ K.
4. Почему в RSA нельзя брать $e = 1$? А $e = \varphi(n) - 1$?
5. Что произойдёт, если в RSA Алиса случайно использует $m \geq n$?
6. * Объясните, как именно атака «человек посередине» работает в случае RSA (не Диффи–Хеллмана).
7. * Что вы посоветуете человеку, который хочет «шифровать всё своими силами без сертификатов»? Какие угрозы он не учитывает?

5.4 Применения RSA: аутентификация, цифровая подпись, электронное голосование

В предыдущем параграфе мы научились с помощью RSA *передавать секретное сообщение*. На первый взгляд кажется, что этого уже достаточно для всех задач сетевой безопасности. Однако в действительности шифрование решает лишь одну из проблем —

конфиденциальность. Помимо неё, в современных системах необходимо обеспечивать ещё как минимум три:

- **Аутентификацию** — уверенность, что вы общаетесь именно с тем, за кого себя выдаёт собеседник.
- **Целостность** — уверенность, что сообщение не было незаметно изменено по дороге.
- **Неотказуемость** — невозможность для отправителя впоследствии отказаться от своего сообщения («это не я отправлял»).

Удивительно, но все эти задачи можно решить с помощью *той же самой схемы RSA*, если применить её не «прямо», а наоборот — сначала закрытым ключом, а потом открытым. Об этом и пойдёт речь в этом параграфе.

5.4.1 Аутентификация: «докажи, что это ты»

Представим типичную ситуацию: Алиса заходит в свой личный кабинет на сайте банка. Банк хочет убедиться, что это действительно его клиентка, а не злоумышленник, подсмотревший пароль.

Самый простой подход — запросить пароль. Но у него есть очевидные недостатки: пароль можно подсмотреть, перехватить, угадать, заставить пользователя его выдать. Кроме того, чтобы проверить пароль, банк должен *хранить* его (или его хеш — см. §5.5) у себя на серверах. А значит, при взломе сервера утечёт сразу множество паролей.

Криптография позволяет построить намного более изящную схему, в которой *секрет вообще никогда не передаётся по сети*.

Важно: принцип «доказательства с нулевым разглашением»

Сторона A должна доказать стороне B, что обладает неким секретом, *не выдавая никакой информации о самом секрете*. Идея, кажущаяся парадоксальной, на практике вполне реализуема и лежит в основе многих криптографических протоколов.

Простейший пример — **протокол challenge-response** («запрос–ответ») на базе RSA. Пусть у Алисы заведена пара ключей: открытый (e, n) опубликован в системе банка, а закрытый d хранится только у неё (например, в специальном USB-токене или в защищённом хранилище смартфона).

Алгоритм: аутентификация через RSA

1. **Запрос (challenge).** Банк генерирует *случайное* число r $(0 < r < n)$, запоминает его и отсылает Алисе.
2. **Ответ (response).** Алиса вычисляет $s = r^d \bmod n$, используя свой закрытый ключ, и отправляет s обратно.
3. **Проверка.** Банк вычисляет $r' = s^e \bmod n$ и сравнивает с тем r, которое он отсылал. Если $r' = r$, проверка пройдена.

Почему это работает? В предыдущем параграфе мы доказали, что для любого r выполняется

$$(r^d)^e \equiv r \pmod{n}.$$

То есть только тот, кто знает d, может «правильно» преобразовать r так, чтобы после открытого преобразования $\cdot^e$ получилось снова r. Любой другой пользователь, не знающий d, может лишь *угадывать* ответ — а вероятность угадать одно нужное число из миллиарда миллиардов исчезающе мала.

Почему это *безопасно*?

- Алиса *не передаёт* свой закрытый ключ d — он остаётся у неё.
- Каждый раз r *случайное и новое*. Поэтому даже если злоумышленник перехватит весь сеанс и услышит пару (r, s), в следующий раз банк пришлёт другое r, и старый ответ s не подойдёт. Атака повторного воспроизведения (replay attack) исключена.
- Банк *не хранит секретов клиента*: у него лишь открытый ключ, утечка которого ничем не страшна.

Схему этого протокола в графическом виде даёт рис. 5.13.

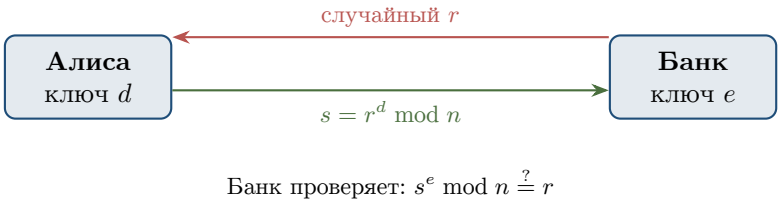


Рис. 5.13. Схема аутентификации по запросу–ответу.

Это интересно: «запомни всё, чтобы ничего не помнить»

В современных смартфонах закрытый ключ d хранится в специальном чипе — *защищённом анклаве* (Secure Enclave у Apple, Trusted Execution Environment на Android). Этот чип *физически не позволяет считать* закрытый ключ извне. На него можно лишь «передать» число и попросить выполнить с ним операцию $\cdot^d \bmod n$. Так устроены, например, биометрические разблокировки — отпечаток пальца «разрешает» чипу применить ключ, но не извлекает его.

5.4.2 Электронная цифровая подпись

Аутентификация решает задачу «докажи, что ты — это ты». Теперь рассмотрим родственную, но другую: «докажи, что *именно ты* отправил *именно это* сообщение». Эта задача стоит особенно остро при подписании документов: договоров, заявлений, налоговых деклараций.

Бумажная подпись от руки выполняет сразу две функции: *идентифицирует* автора и *связывает* его с конкретным документом. К сожалению, при простом копировании в электронном виде это свойство теряется: цифровое изображение подписи легко вырезать и приклеить к любому документу. Нужна принципиально иная конструкция.

Определение 5.24. электронная цифровая подпись

Электронная цифровая подпись (ЭЦП) — это короткий блок данных σ, привязываемый к документу m, обладающий двумя свойствами:

- только владелец секретного ключа может сгенерировать σ для произвольного m;
- любой желающий, зная открытый ключ, может проверить, что σ действительно соответствует m и подписан данным владельцем.

Идея конструкции на основе RSA удивительно проста и красива: достаточно поменять местами роли ключей в шифровании.

Алгоритм: цифровая подпись RSA

Алиса имеет открытый ключ (e, n) и закрытый d. Хочет подписать документ m $(0 < m < n)$.

1. **Подпись.** Алиса вычисляет
$$\sigma = m^d \bmod n$$
и публикует пару (m, σ).

2. **Проверка.** Любой проверяющий вычисляет $m' = \sigma^e \bmod n$ и сравнивает с m. Если $m' = m$, подпись подлинна.

Почему это подпись? По тому же самому соотношению $(m^d)^e \equiv m \pmod n$. Только владелец закрытого ключа d может построить такое σ, что после возведения в степень e получится исходное m. Если бы это умел кто-то другой, он, по сути, умел бы разлагать n на множители — а это, как мы обсуждали, считается практически невыполнимой задачей.

Привязка к документу. В отличие от рукописной подписи, ЭЦП *зависит от содержимого*: если изменить хоть один бит в m, изменится и σ. Невозможно «отделить» подпись от документа и приклеить её к другому: для другого документа потребуется заново вычислять $\cdot^d \bmod n$, что без d невыполнимо (рис. 5.14).

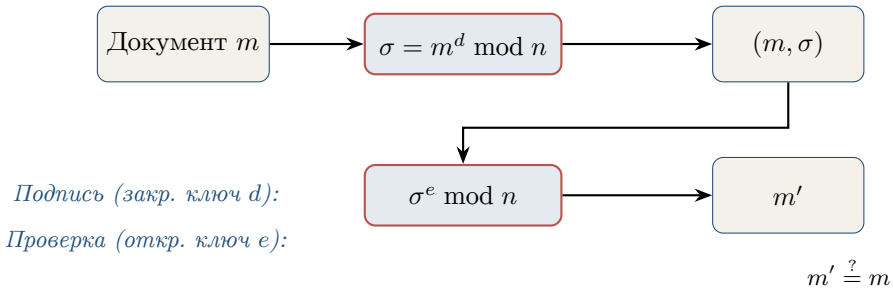


Рис. 5.14. Подписание и проверка электронной цифровой подписи на базе RSA.

Проблема большого документа. В реальности документ может быть огромным (например, несколько мегабайт). Возводить миллион байтов в степень d по модулю n слишком медленно. Поэтому в практических системах поступают иначе:

1. вычисляют **хеш** (короткий «отпечаток») документа: $h = H(m)$ — небольшое число, скажем, длиной в 256 бит;
2. подписывают именно хеш: $\sigma = h^d \bmod n$.

При проверке вычисляют $h' = H(m)$ заново и сравнивают $\sigma^e \bmod n$ с h'. О хеш-функциях и их математическом устройстве пойдёт речь в §5.5.

Это интересно: ЭЦП в России

В Российской Федерации электронная цифровая подпись юридически приравнена к собственноручной с 2002 года (закон № 1-ФЗ «Об электронной цифровой подписи», заменён в 2011 году на № 63-ФЗ «Об электронной подписи»). С помощью ЭЦП сегодня сдают налоговую отчётность, регистрируют недвижимость, участвуют в государственных закупках — всё это без бумажных носителей. Криптографические алгоритмы, используемые в российских ЭЦП, описаны в стандарте ГОСТ Р 34.10–2012; математически они близки к описанной RSA-схеме, но используют не модулярное возведение в степень, а операции на так называемых *эллиптических кривых*.

5.4.3 Электронное голосование

С распространением Интернета возникает естественный вопрос: можно ли перевести голосование — например, выборы или опросы — в цифровой формат? На первый взгляд, это просто: создать сайт, попросить каждого выбрать вариант, посчитать голоса. На деле возникает несколько противоречивых требований, и решить их одновременно — содержательная криптографическая задача.

Важно: требования к электронному голосованию

Хорошая система электронного голосования должна одновременно обеспечивать:

1. **Корректность**: каждый имеющий право проголосовать голосует не более одного раза.
2. **Анонимность**: никто, включая организаторов, не может сопоставить конкретного избирателя с его голосом.
3. **Проверяемость**: избиратель может убедиться, что его голос учтён, а наблюдатели — что общий подсчёт верен.
4. **Неподдельность**: никто не может вбросить «лишние» голоса.

Заметим противоречие между пунктами 1 и 2: чтобы убедиться, что Алиса голосует не дважды, надо как-то её опознать; но если её опознали, как сохранить анонимность? На бумажных выборах эта задача решается *физически* (список явки и тайная урна разделены), а в цифровом мире — математически. Ключевой инструмент — **слепая цифровая подпись**, предложенная Дэвидом Чаумом ещё в 1982 году.

Слепая подпись. Идея слепой подписи: можно подписать документ, *не зная его содержания*. Этот странный, казалось бы, инструмент имеет естественный аналог в офлайне.

Это интересно: аналогия с конвертом и копиркой

Представьте: вы кладёте лист бумаги в конверт, внутри которого вложена копирка. Затем просите начальника поставить подпись *на конверте*. Подпись через копирку переходит и на лист внутри. Раскрыв конверт, вы получаете подписанный документ, причём начальник никогда не видел его содержания.
Слепая RSA-подпись — это математическая реализация той же идеи: «копирка» — модулярное умножение на случайный множитель.

Формально схема устроена так. Пусть у центра голосования есть RSA-ключи (e, n) (открытый) и d (закрытый).

Алгоритм: слепая подпись RSA

Алиса хочет получить подпись центра под её бюллетенем m, не раскрывая центру содержимое.

1. Алиса выбирает случайное k, взаимно простое с n, и вычисляет *ослеплённое* сообщение
$$m' = m \cdot k^e \bmod n.$$
Отправляет m' центру.
2. Центр (после проверки, что Алиса действительно имеет право голоса) ставит подпись:
$$\sigma' = (m')^d \bmod n = m^d \cdot k^{ed} \bmod n = m^d \cdot k \bmod n.$$

Возвращает σ' Алисе.

3. Алиса *снимает ослепление*: вычисляет

$$\sigma = \sigma' \cdot k^{-1} \bmod n = m^d \bmod n.$$

Получена настоящая RSA-подпись центра под её бюллетенем m, хотя центр никогда не видел m.

Где здесь теория чисел. Заметим, что на шаге 3 Алиса делит на k, точнее — умножает на $k^{-1} \bmod n$. Существование *обратного* по модулю n обеспечивается тем, что $\gcd(k, n) = 1$, и его можно эффективно вычислить расширенным алгоритмом Евклида (§5.1). Кроме того, в выкладке использовалось $k^{ed} \equiv k \pmod{n}$ — следствие малой теоремы Ферма (точнее, её обобщения — теоремы Эйлера, гласящей, что $k^{\varphi(n)} \equiv 1 \pmod{n}$ для $\gcd(k, n) = 1$).

Сам протокол голосования. На базе слепой подписи строится следующая схема (в упрощённом виде; подробности — в книге Музыкантского и Фурина).

Алгоритм: электронное голосование

1. *Подготовка.* Алиса формирует бюллетень m — например, число, кодирующее её выбор, плюс случайный «уникальный идентификатор» r, чтобы исключить совпадение бюллетеней.
2. *Получение подписи.* Алиса ослепляет m, отправляет m' **Регистратору**. Регистратор проверяет по своему списку, что Алиса имеет право голосовать и ещё не получала подпись, и возвращает σ'. Алиса снимает ослепление и получает (m, σ).
3. *Анонимная отправка.* Через анонимный канал (например, через сеть посредников или с другого устройства) Алиса отправляет (m, σ) **Счётчику**.
4. *Подсчёт.* Счётчик проверяет подпись σ: если она верна и идентификатор r ранее не встречался — голос засчитан.

Почему это работает.

- *Корректность.* Регистратор контролирует, что одна и та же Алиса не получит подпись дважды.
- *Анонимность.* Регистратор знает, кто такая Алиса, но не видит содержимого m (оно ослеплено). Счётчик видит m и σ, но не знает, кому принадлежит подпись — лишь то, что она поставлена настоящим Регистратором.
- *Неподдельность.* Без знания закрытого ключа d невозможно сгенерировать σ к произвольному m — это потребовало бы взлома RSA.
- *Проверяемость.* Список $\{(m, \sigma)\}$ публикуется. Алиса может убедиться, что её (m, σ) в списке; любой наблюдатель может пересчитать сумму, не зная имён.

Общую идею протокола поясняет рис. 5.15.

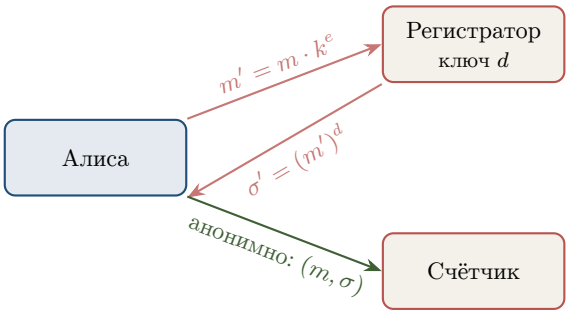


Рис. 5.15. Электронное голосование на базе слепой подписи. С Регистратором Алиса работает «лицом к лицу» (он знает её, но не видит бюллетень). К Счётчику бюллетень приходит анонимно, но с проверяемой подписью Регистратора.

Это интересно: голосование в реальности

Системы электронного голосования на основе слепой подписи (и более сложных схем) уже применялись на практике: например, в Эстонии часть выборов проходит через интернет с 2005 года, в России на муниципальных выборах в Москве с 2019 года использовалась система на базе криптографических конструкций (правда, на основе не только RSA, но и других схем — с гомоморфным шифрованием и блокчейном). Подробное обсуждение разных схем электронного голосования можно найти в книге Музыкантского и Фурина «Лекции по криптографии».

5.4.4 Итог параграфа

Мы увидели, что одна и та же конструкция RSA — модулярное возведение в степень — решает четыре совершенно разные задачи:

шифрование	аутентификация	цифровая подпись	слепая подпись

Различие лишь в том, *в каком порядке* применяются операции $\cdot^e$ и $\cdot^d$ и какие именно числа подставляются на вход. В этом одна из главных красот современной криптографии: глубокая математическая идея даёт целый каскад прикладных решений.

Задачи для самостоятельной работы

1. Объясните своими словами, в чём принципиальная разница между «шифрованием с открытым ключом» и «цифровой подписью», если в обеих схемах используются одни и те же ключи и одна и та же формула.

2. Почему в протоколе аутентификации банк каждый раз присылает *новое случайное* r, а не одно и то же?

3. В схеме ЭЦП Алиса публикует пару (m, σ). Что произойдёт, если она опубликует пару (m', σ) с изменённым $m' \neq m$? Сможет ли она тем самым обмануть проверяющего?

4. * В схеме слепой подписи покажите подробной выкладкой, что после снятия ослепления получается именно $m^d \bmod n$ (используйте $k^{ed} \equiv k \pmod n$).

5. * Предложите простую (возможно, наивную) атаку на схему голосования, если из протокола убрать пункт о случайном идентификаторе r в бюллетене.

6. * Подумайте, какие ещё свойства бумажного бюллетеня (например, возможность переголосовать, защита от принуждения) электронное голосование решает плохо. Каковы возможные подходы?

5.5 Хеширование: теория чисел встречается с рандомизацией

В предыдущих параграфах мы видели, как теория чисел даёт «жёсткие» гарантии безопасности: вычислить нужный ответ практически невозможно из-за вычислительной сложности задачи факторизации. Сейчас мы рассмотрим совсем другой тип использования теории чисел — **вероятностные алгоритмы**, в которых правильность работы достигается не «навсегда», а *в среднем*, с очень высокой вероятностью.

Главная мысль: добавив в алгоритм *случайный выбор* (на этапе настройки, ещё до запуска), можно гарантировать, что для *любых* входных данных алгоритм работает в среднем хорошо. Принципиальное отличие от наивного «алгоритм работает хорошо для случайных данных» в том, что входные данные могут быть какими угодно — даже подобраны злоумышленником, — а случайность спрятана *внутри* алгоритма.

Этот параграф следует изложению из классической книги С. Дасгупты, Х. Пападимитриу и У. Вазирани «Алгоритмы»[1], в котором роль теории чисел особенно прозрачна.

5.5.1 Задача быстрого поиска

Представьте сервис электронной почты, в котором зарегистрировано сто миллионов пользователей. Каждый пользователь идентифицируется адресом, скажем, длиной до 30 символов. При каждом входе нужно *мгновенно* (за время, не зависящее от размера базы) определить: есть ли такой пользователь, и если есть — найти его данные.

Технически: имеется большая «вселенная» возможных ключей U (всех допустимых адресов), и интересующее нас подмножество $S \subseteq U$ из n реальных пользователей. Мы хотим организовать структуру данных, позволяющую за $O(1)$ операций проверять, лежит ли $x \in S$, и получать данные, связанные с x.

Прямая адресация. Самое простое решение — завести массив, индексируемый *всеми* элементами U. Такой массив имеет длину $|U|$. Если $|U|$ велико (а оно *колоссально*: количество всевозможных 30-символьных строк — это 26^{30}, что больше 10^{42}), это решение нереализуемо: не хватит памяти всей планеты.

Идея хеширования. Вместо того, чтобы выделять ячейку под каждый возможный элемент, заведём массив размера $m \approx n$ (того же порядка, что число реальных пользователей), и зададим **хеш-функцию** $h : U \to \{0, 1, \ldots, m-1\}$, которая каждому ключу сопоставляет номер ячейки.

Определение 5.25. хеш-функция и хеш-таблица

Хеш-функция — это произвольная функция $h : U \to \{0, 1, \ldots, m-1\}$. По ней строится **хеш-таблица**: массив $T[0], T[1], \ldots, T[m-1]$. Ключ $x \in U$ хранится в ячейке $T[h(x)]$.

Если двум разным ключам $x \neq y$ присвоилось одно и то же значение $h(x) = h(y)$, мы говорим, что произошла **коллизия**.

Иллюстрация работы хеш-функции и появления коллизий приведена на рис. 5.16.

[1] Дасгупта С., Пападимитриу Х., Вазирани У. *Алгоритмы.* М.: МЦНМО, 2014. Глава о хешировании.

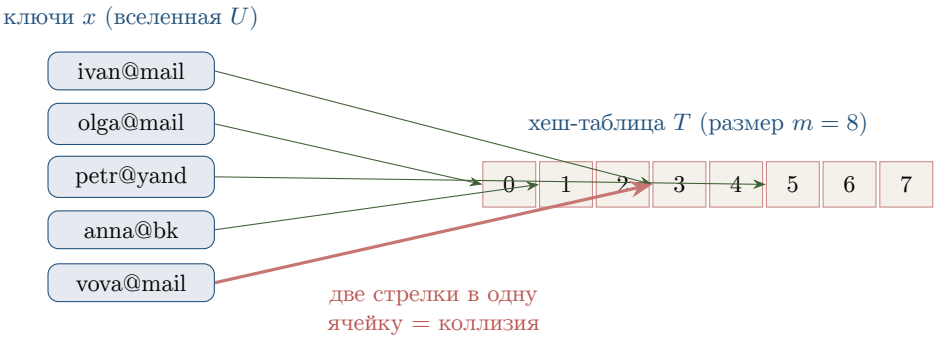


Рис. 5.16. Хеш-функция «утрамбовывает» большую вселенную ключей в маленькую таблицу. Иногда возникают коллизии.

Если коллизий нет (или их мало), мы успешно сжали огромную вселенную ключей в небольшую таблицу. Если же коллизий много, эффективность падает: на одну ячейку приходится много ключей, поиск замедляется.

5.5.2 Беда детерминированного хеширования

Зафиксируем какую-нибудь конкретную хеш-функцию — например, $h(x) = x \bmod m$ (в предположении, что ключи x представлены как целые числа). На большинстве входов работает прекрасно. Но злоумышленник, зная нашу функцию, может специально подобрать набор ключей $S = \{0, m, 2m, 3m, \dots\}$ — все они дают одинаковый остаток 0. Все они попадут в одну ячейку. Хеш-таблица деградирует до линейного списка, и каждый поиск занимает $O(n)$ вместо $O(1)$.

Этот сценарий не теоретический: реальные сетевые сервисы испытывали атаки именно такого рода — атаки «алгоритмическим усложнением», когда поток специально подобранных запросов приводил к замедлению сервера в тысячи раз.

Важно: принципиальное наблюдение

Для *любой* фиксированной хеш-функции h найдётся «плохой» набор ключей, на котором коллизий очень много. Никакая «гениальная» формула h не спасает.
Спасает другое: *не фиксировать h заранее*, а выбирать её случайно из специального семейства функций.

5.5.3 Универсальное семейство хеш-функций

Идея, восходящая к Картеру и Вегману (1977), состоит в следующем. Договоримся об *целом семействе* хеш-функций $\mathcal{H} = \{h_a\}$. При создании хеш-таблицы выберем из $\mathcal{H}$ *случайную* функцию h_a. Эта функция и будет использоваться. Поскольку выбор делается уже после того, как зафиксированы данные (или независимо от них), злоумышленник заранее не знает, какая h_a будет применена, и не может «подстроиться» под неё.

Чтобы это сработало, семейство $\mathcal{H}$ должно обладать специальным свойством.

Определение 5.26. универсальное семейство

Семейство хеш-функций $\mathcal{H} = \{h : U \to \{0, \dots, m-1\}\}$ называется **универсальным**, если для любых двух различных ключей $x, y \in U$ $(x \neq y)$ вероятность коллизии при случайном выборе $h \in \mathcal{H}$ не превосходит $1/m$:

$$\Pr_{h \in \mathcal{H}}[\, h(x) = h(y) \,] \leq \frac{1}{m}.$$

Поясним: если бы h присваивала каждому ключу совершенно случайный номер ячейки из m возможных, вероятность совпадения двух конкретных ключей в одной ячейке составила бы ровно $1/m$. Универсальное семейство просит лишь того же порядка — вероятность *не больше* $1/m$.

Теорема 5.27. (оценка коллизий)

Если хеш-функция выбрана из универсального семейства, то математическое ожидание числа коллизий заданного ключа x с прочими $n-1$ ключами таблицы не превосходит $(n-1)/m$. В частности, если $m \geq n$, среднее число коллизий не превосходит 1.

Доказательство. Введём индикаторы: $X_y = 1$, если $h(x) = h(y)$, иначе $X_y = 0$. Общее число коллизий ключа x есть $\sum_{y \in S, y \neq x} X_y$. По линейности математического ожидания:

$$\mathbb{E}\Big[\sum X_y\Big] = \sum \mathbb{E}[X_y] = \sum \Pr[h(x) = h(y)] \leq (n-1) \cdot \frac{1}{m}.$$

□

Таким образом, для $m \approx n$ *в среднем* на одну ячейку приходится не более одной–двух коллизий, и время поиска остаётся константным.

Где встречается случайность. Подчеркнём: математическое ожидание берётся *по случайному выбору* h, а не по случайному выбору ключей! Сами ключи могут быть «худшими из возможных» — утверждение всё равно справедливо. Это и есть сила рандомизации.

5.5.4 Конструкция универсального семейства через теорию чисел

Теперь самый интересный вопрос: *как же построить* такое семейство $\mathcal{H}$? И вот тут на сцену выходит теория чисел.

Шаг 1: представление ключа. Будем считать, что каждый ключ $x \in U$ — это *вектор* из k небольших чисел: $x = (x_1, x_2, \ldots, x_k)$. Например, если ключ — это IP-адрес вида 192.168.0.1, его естественно представить как четвёрку чисел $(192, 168, 0, 1)$. Если ключ — это строка, разобьём её на четвёрки или восьмёрки байтов.

Пусть каждое x_i лежит в диапазоне $0 \leq x_i < p$, где p — **простое число**, чуть большее размера наших значений.

Шаг 2: выбор размера таблицы. Положим $m = p$. (В этом разделе для простоты принимаем такое допущение; в общем случае схема легко обобщается.)

Шаг 3: семейство $\mathcal{H}$. Для каждого вектора $a = (a_1, \ldots, a_k)$, где $0 \leq a_i < p$, определим хеш-функцию:

$$h_a(x) = \big(a_1 x_1 + a_2 x_2 + \cdots + a_k x_k\big) \bmod p.$$

Семейство $\mathcal{H} = \{h_a : a \in \{0, \ldots, p-1\}^k\}$. Размер семейства — p^k.

Шаг 4: случайный выбор. При построении таблицы выбираем a случайно равновероятно из $\{0, \ldots, p-1\}^k$ (например, генератором случайных чисел).

Теорема 5.28. (Картер–Вегман)

Описанное выше семейство $\mathcal{H}$ *универсально*.

Доказательство. Возьмём два разных ключа $x = (x_1, \dots, x_k)$ и $y = (y_1, \dots, y_k)$, $x \neq y$. Поскольку $x \neq y$, найдётся хотя бы один индекс i, в котором $x_i \neq y_i$; без ограничения общности будем считать $i = 1$.

Коллизия $h_a(x) = h_a(y)$ означает:

$$a_1x_1 + a_2x_2 + \dots + a_kx_k \equiv a_1y_1 + a_2y_2 + \dots + a_ky_k \pmod p,$$

или, после перестановки,

$$a_1(x_1 - y_1) \equiv -\sum_{i=2}^{k} a_i(x_i - y_i) \pmod p. \qquad (*)$$

Обозначим правую часть как R. Заметим, что R зависит от $a_2, \dots, a_k$, но *не зависит* от a_1.

Здесь и появляется теория чисел. Поскольку p — *простое*, а $x_1 - y_1 \neq 0 \pmod p$ (по нашему предположению $x_1 \neq y_1$, и оба они в диапазоне $[0, p)$, значит их разность ненулевая по модулю p), у числа $(x_1 - y_1)$ *существует и единственен* обратный элемент $(x_1 - y_1)^{-1} \bmod p$. Это — ровно то свойство модулярной арифметики над простым модулем, которое мы доказали в §5.1 (расширенный алгоритм Евклида).

Следовательно, уравнение $(*)$ имеет ровно одно решение относительно a_1 при фиксированных $a_2, \dots, a_k$:

$$a_1 = R \cdot (x_1 - y_1)^{-1} \bmod p.$$

Среди p возможных значений a_1 только одно приводит к коллизии. Поэтому условная вероятность коллизии (при фиксированных $a_2, \dots, a_k$) равна $1/p$. Усреднив по $a_2, \dots, a_k$, получаем

$$\Pr_a[h_a(x) = h_a(y)] = \frac{1}{p} = \frac{1}{m}.$$

□

Важно: почему именно простое p

Ключевую роль играет то, что p — *простое*. Будь p составным, у числа $x_1 - y_1$ *могло бы не быть обратного* (если бы оно делилось на общий с p множитель), и тогда уравнение $a_1 \cdot (x_1 - y_1) \equiv R \pmod p$ либо не имело бы решения, либо имело несколько — и аккуратной оценки $1/p$ не получилось бы.

Так теорема единственности обратного элемента по простому модулю, открытая в античности, оказывается фундаментом современных структур данных, ежесекундно работающих во всех серверах мира.

5.5.5 Пример: маленькая хеш-таблица

Пусть простое $p = 7$ (значит, и $m = 7$), а ключи — двойки чисел из $\{0, 1, \dots, 6\}$, то есть $k = 2$. Возьмём пять «настоящих» ключей:

$$S = \{(1, 2),\ (3, 4),\ (5, 5),\ (2, 6),\ (4, 1)\}.$$

Случайно выбираем $a = (a_1, a_2)$. Например, пусть случайный генератор выдал $a = (3, 5)$. Тогда $h_a(x) = (3x_1 + 5x_2) \bmod 7$:

ключ x	$3x_1 + 5x_2$	$\bmod 7$
$(1, 2)$	$3 + 10 = 13$	6
$(3, 4)$	$9 + 20 = 29$	1
$(5, 5)$	$15 + 25 = 40$	5
$(2, 6)$	$6 + 30 = 36$	1
$(4, 1)$	$12 + 5 = 17$	3

Получаем коллизию: $(3,4)$ и $(2,6)$ попали в ячейку 1. Если бы случайный генератор выдал другое a, например $a=(2,1)$, картина была бы другой:

ключ x	$2x_1+x_2$	$\bmod 7$
$(1,2)$	4	4
$(3,4)$	10	3
$(5,5)$	15	1
$(2,6)$	10	3
$(4,1)$	9	2

Снова коллизия (правда, в другой ячейке: $(3,4)$ и $(2,6)$ дали 3). Удивительно: эти два ключа коллидируют при разных a. Дело в том, что $(3,4)-(2,6)=(1,-2)$, и многие a удовлетворяют $a_1-2a_2\equiv 0 \pmod 7$. Однако в среднем по всем a вероятность коллизии любых двух конкретных ключей — ровно $1/7$, как и обещает теорема.

5.5.6 Применения хеширования

Универсальное хеширование — одна из самых востребованных конструкций в современной информатике. Вот лишь несколько мест, где оно работает:

- **Поиск дубликатов и быстрая выборка.** Все базы данных, все таблицы соответствий в операционных системах, все словари в языках программирования (Python: `dict`, JavaScript: `Map`, Java: `HashMap`) построены на хешировании.
- **Проверка целостности файлов.** Контрольные суммы при скачивании файла (CRC, MD5, SHA) — по сути, хеш-функции. Если хеш скачанного файла не совпадает с заявленным, файл повреждён.
- **Цифровая подпись.** Как мы видели в §5.4.2, RSA-подпись на самом деле ставится не на весь документ, а на его хеш.
- **Антивирусы и системы обнаружения дубликатов.** База вирусных сигнатур — это хеши известных вредоносных файлов; чтобы проверить, не заражён ли файл, антивирус считает его хеш и ищет в базе.
- **Blockchain.** В криптовалютах хеш каждого блока включается в следующий блок, образуя цепочку, в которой невозможно незаметно подправить «прошлое».

Это интересно: криптографические хеш-функции

В этом параграфе мы говорили о «простых» хеш-функциях, минимизирующих коллизии. В криптографии используется родственное, но более сильное понятие — **криптографическая хеш-функция**. От неё требуется не только малое число коллизий, но и *вычислительная невозможность найти* два разных входа с одинаковым хешем (*collision resistance*), а также найти прообраз заданного хеша (*preimage resistance*). Примеры таких функций — SHA-256, BLAKE2, в России — «Стрибог» (ГОСТ Р 34.11–2012). Внутри они устроены сложнее (и не на основе модулярной арифметики), но используют тот же общий принцип: смешать входные биты так, чтобы малейшее изменение во входе приводило к радикальному изменению в выходе.

Задачи для самостоятельной работы

1. Для простого $p=5$, $k=1$ и $a=3$ найдите $h_a(x)=ax \bmod p$ для всех $x\in\{0,1,2,3,4\}$.

2. Пусть ключи — пары (x_1, x_2), $p = 11$, $a = (4, 7)$. Найдите коллидирует ли пара $(1, 3)$ с парой $(8, 1)$ при таком a.

3. Объясните, что произойдёт с оценкой $1/m$, если в семействе Картера–Вегмана взять p составным (например, $p = 6$).

4. Почему в формуле хеш-функции $h_a(x) = (a_1x_1 + \cdots + a_kx_k) \bmod p$ нет «свободного члена» a_0, как в линейной функции $ax + b$? Что изменится, если его добавить?

5. * Покажите, что если выбирать a всего лишь из множества $\{1\}$ (одно значение), то семейство не универсально.

6. * Покажите, что вероятность того, что у заданного ключа *не будет* коллизий ни с каким из $n - 1$ других ключей в таблице размера $m \geq n$, не меньше $1/2$ (используйте неравенство Маркова).

5.6 Счётчики с короткой памятью HyperLogLog и эксперимент Стенли Мильграма

В этом, заключительном параграфе главы мы рассмотрим ещё один сюжет на стыке теории чисел, теории вероятностей и информатики — задачу, в которой важно *подсчитывать число различных объектов* в огромном потоке данных, не имея возможности хранить их все. Это пригодится для задачи, на первый взгляд несвязанной с криптографией: измерения «социальных расстояний» в больших сетях, и проверки знаменитой гипотезы Стенли Мильграма о шести рукопожатиях.

Изложение в этом параграфе опирается на главу 7 книги Литвак и Райгородского «Кому нужна математика?»[2], которую мы рекомендуем для самостоятельного изучения.

5.6.1 Задача подсчёта различных элементов

Представьте, что вы работаете в большой интернет-компании и хотите узнать: сколько *уникальных* пользователей зашло сегодня на сайт? Не общее число обращений (его легко посчитать инкрементируя счётчик при каждом запросе), а именно *различных* людей.

Точное решение. Простое и очевидное: завести множество (например, хеш-таблицу из §5.5), и при каждом приходящем идентификаторе пользователя проверять, есть ли он там; если нет — добавить и увеличить счётчик. По окончании дня размер множества — ответ.

Проблема. Если число уникальных пользователей — сотни миллионов или миллиарды, такая таблица занимает гигабайты памяти. Для одной задачи на одном сервере — допустимо. А если таких задач тысячи, и каждая на каждом из тысяч серверов? Память становится узким местом.

Возникает естественный вопрос: можно ли подсчитать число различных элементов в потоке *приближённо*, но используя *гораздо меньше памяти*? Например, дать ответ с относительной погрешностью $\pm 2\%$, заняв всего несколько килобайт?

Удивительно, но ответ положительный. И построение такого алгоритма опирается на красивые идеи теории вероятностей и — что нам особенно интересно — использует хеш-функции, родственные тем, что мы изучали в §5.5.

[2] Литвак Н. В., Райгородский А. М. *Кому нужна математика? Содержательное введение в математику и computer science*. М.: Манн, Иванов и Фербер, 2017. Гл. 6, 7 и приложения.

5.6.2 Идея: «самый длинный хвост нулей»

Алгоритм, который мы сейчас опишем (он называется **HyperLogLog**, предложен Филиппом Флажоле и его коллегами в 2007 году), основан на одной красивой вероятностной идее.

Пусть у нас есть «хорошая» хеш-функция h, которая для каждого входного объекта (например, идентификатора пользователя) выдаёт случайную на вид последовательность из L битов. Слово «случайную» означает: для разных входов биты выглядят как независимые подбрасывания монеты[3].

Наблюдение. Если число $h(x)$ действительно ведёт себя случайно, то рассмотрим, сколько *нулей подряд* стоит в начале двоичной записи $h(x)$.

- Вероятность, что первый бит — ноль: $1/2$.
- Вероятность, что первые два бита — нули: $1/4$.
- Вероятность, что первые k битов — нули: $1/2^k$.

Наглядно эта закономерность показана на рис. 5.17.

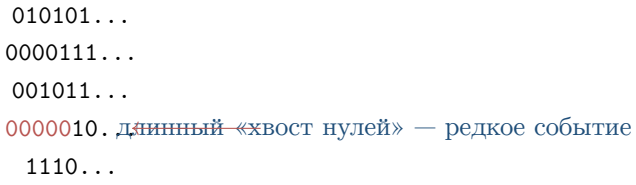


Рис. 5.17. Иллюстрация двоичных хешей. Чем длиннее «хвост нулей» в начале, тем реже такое событие.

Иначе говоря, если в потоке встречается элемент x, такой что $h(x)$ начинается, скажем, с 20 нулей подряд, это значит, что в потоке *скорее всего побывало порядка* $2^{20} \approx 10^6$ *различных элементов* — иначе крайне маловероятно увидеть такое редкое событие.

Базовый алгоритм («одиночный счётчик»).

Алгоритм: наивный счётчик различных элементов

1. Заведём одну переменную $M = 0$.
2. Для каждого приходящего элемента x потока:
   (a) вычислим $h(x)$;
   (b) посчитаем число $\rho(h(x))$ — длину начального блока нулей в $h(x)$, плюс единица;
   (c) обновим $M \leftarrow \max(M, \rho(h(x)))$.
3. На выходе оценим число различных элементов как $\widehat{n} = 2^M$.

Память. Чтобы хранить M от 0 до L (где L — длина хеша, скажем, 32 бита), достаточно *шести битов памяти*. Шесть битов — чтобы оценить число различных элементов в потоке любой длины!

[3] Конечно, h детерминированна, но её устройство таково, что заранее, не считая, угадать биты невозможно. На практике используются криптографические или близкие к ним хеш-функции, обладающие именно таким свойством «равномерности».

Проблема. Эта оценка очень шумная: одно «случайно длинное» обнуление сбивает её в большую сторону, а его отсутствие — в меньшую. Дисперсия чудовищная.

5.6.3 Идея «несколько счётчиков»

Чтобы сгладить случайные колебания, разделим поток на много подпотоков и усредним результаты. Это и есть алгоритм HyperLogLog.

Алгоритм: HyperLogLog (упрощённая схема)

1. Выберем число b — скажем, $b = 10$, — и заведём $2^b = 1024$ независимых счётчика $M_0, M_1, \ldots, M_{2^b-1}$. Каждый счётчик — 6 битов.
2. Для каждого приходящего элемента x:
   (a) вычислим $h(x)$ длины L;
   (b) первые b битов $h(x)$ интерпретируем как номер счётчика j, $0 \le j < 2^b$;
   (c) к остальным $L - b$ битам применяем правило: ρ = длина начального блока нулей плюс 1;
   (d) обновляем: $M_j \leftarrow \max(M_j, \rho)$.
3. На выходе оцениваем

$$\widehat{n} = \alpha_b \cdot 2^b \cdot \left(\frac{1}{2^b} \sum_{j=0}^{2^b-1} 2^{-M_j} \right)^{-1},$$

где α_b — известная константа корректировки (для $b = 10$ примерно 0.7213).

Почему так? Формула в третьем шаге — это *гармоническое среднее* величин 2^{M_j}, а не простое арифметическое. Гармоническое среднее устойчиво к выбросам (одно случайно большое M_j его существенно не искажает) — именно поэтому HyperLogLog так точен.

Теорема 5.29. (точность HyperLogLog)
Относительная погрешность оценки $\widehat{n}$ по сравнению с истинным значением n имеет порядок $1.04/\sqrt{2^b}$. При $b = 10$ это примерно 3.25%; при $b = 14$ — около 0.81%.

Память. При $b = 14$ имеется $2^{14} = 16\,384$ счётчиков по 6 битов — всего ≈ 12 КБ. И этим объёмом мы оцениваем число уникальных элементов в потоке длиной до $\approx 10^{18}$ с точностью около 1%! (рис. 5.18)

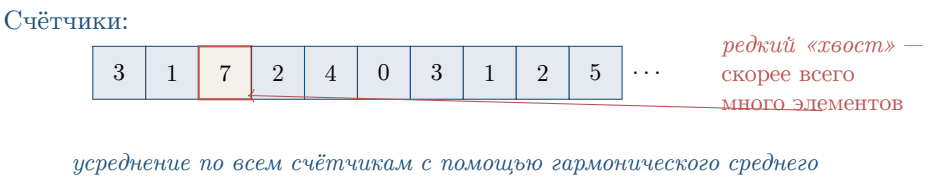


Рис. 5.18. Множество счётчиков HyperLogLog. Каждый запоминает максимальную длину «хвоста нулей» для своего подпотока.

Это интересно: где работает HyperLogLog

HyperLogLog встроен в Redis (популярную систему кэширования), в Google BigQuery (язык запросов к огромным базам данных), в систему аналитики ВКонтакте, в инфраструктуру PostgreSQL и многих других сервисов. Когда вы видите в личном кабинете рекламной системы оценку «охват: 3,5 млн уникальных пользователей», очень вероятно, что под капотом работает именно этот алгоритм.

5.6.4 Эксперимент Стенли Мильграма

Теперь, когда у нас в руках мощный инструмент для подсчёта уникальных объектов, обратимся к классической задаче из социологии, которую Литвак и Райгородский подробно разбирают в главе 7 своей книги.

Историческое введение. В 1960-х годах американский социальный психолог **Стенли Мильграм** (1933–1984) задался необычным вопросом: *насколько на самом деле «велик» мир*? Точнее: если выбрать наугад двух человек на разных концах страны, через сколько посредников можно их связать?

В 1967 году Мильграм провёл свой знаменитый эксперимент. Участникам в Канзасе и Небраске были даны письма с просьбой передать их некоему адресату в Бостоне, причём *нельзя* было отправлять напрямую — только через личного знакомого, который, как кажется участнику, может быть ближе к получателю; тот, в свою очередь, выбирал следующего знакомого, и так далее.

Удивительный результат. Большинство писем, дошедших до адресата, преодолевали путь *в среднем за 5–6 пересылок*. Так родилась знаменитая гипотеза «шести рукопожатий»: между любыми двумя людьми на Земле в среднем стоит лишь шесть посредников.

Это интересно: «шесть рукопожатий» в литературе

Идея «малого мира» проникла в массовую культуру: пьеса Джона Гуэра «Six Degrees of Separation» (1990), фильм с тем же названием, многочисленные «эстафеты знакомств». В науке термин **small-world network** стал важным понятием в теории графов и сетевой социологии.

Современная проверка. Эксперимент Мильграма страдал серьёзными методологическими дефектами: 1/ подавляющее большинство писем (около 80%) до адресата так и не дошли; 2/ выборка была очень ограниченной (всего около 300 писем); 3/ «знакомые» отбирались не случайно, а с предположением о близости к получателю.

В современную эпоху, когда подавляющая часть знакомств и контактов перешла в цифровой формат, задачу можно проверить иначе: *построить граф знакомств целиком* и измерить кратчайшие расстояния. Именно это в 2011 году проделали исследователи в рамках одной известной социальной сети — они проанализировали данные *всех* её пользователей (на тот момент — около 720 миллионов человек) и более 69 миллиардов «дружеских связей».

Что они обнаружили. Средняя длина кратчайшего пути между двумя случайно выбранными пользователями оказалась равной *4,74 рукопожатия*. Иначе говоря, в современных социальных сетях гипотеза Мильграма не просто подтверждается, а даже *улучшается*: между двумя пользователями стоит в среднем менее пяти промежуточных знакомых, а не шесть. Принято округлять это число до 4 — отсюда выражение «четыре рукопожатия» (рис. 5.19).

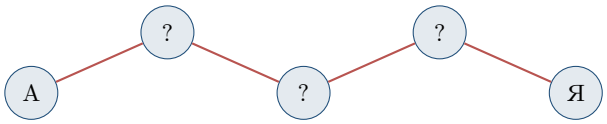


Рис. 5.19. «Четыре рукопожатия»: между Алисой и Яковом стоят всего трое промежуточных знакомых.

5.6.5 При чём здесь HyperLogLog?

Возникает вопрос: *как именно* вычислить среднюю длину кратчайшего пути в графе из 720 миллионов вершин и 69 миллиардов рёбер? Прямой подсчёт всех пар (а их $\binom{720\cdot 10^6}{2} \approx 2{,}6 \cdot 10^{17}$) непосильно даже для самых мощных кластеров.

Здесь и приходит на помощь HyperLogLog (а также его «родственники» — алгоритмы вероятностного подсчёта). Идея, идущая в реальном анализе, такова:

Алгоритм: оценка расстояний в графе через подсчёт «соседей»

1. Для каждой вершины v заведём HyperLogLog-счётчик C_v.
2. Вначале добавим в C_v только саму вершину v. То есть $|C_v| = 1$.
3. На шаге $t = 1, 2, \ldots, T$:
   - для каждой вершины v обновим $C_v \leftarrow C_v \cup \bigcup_{u:(u,v)\text{ – ребро}} C_u^{(t-1)}$,
   - то есть в C_v добавляются все вершины, до которых дошли соседи v на предыдущем шаге.
4. Величина $|C_v|$ после t шагов — это число вершин, до которых от v существует путь длины $\le t$.

Где экономия памяти. Если бы мы хранили C_v как точное множество, для одной вершины пришлось бы выделить до 720 миллионов битов памяти — 90 мегабайт. Умножим на 720 миллионов вершин — получим 65 петабайт. Невозможно.

А поскольку HyperLogLog-счётчик занимает 12 КБ на вершину, всё хранилище умещается в $720{\cdot}10^6 \cdot 12$ КБ $\approx 8{,}6$ ТБ — объём, который уже умещается в один большой серверный кластер.

Объединение HyperLogLog-счётчиков. У алгоритма есть ещё одно прекрасное свойство: **HLL-счётчики можно объединять**. Если C_1 оценивает мощность множества A, а C_2 — мощность B, то новый счётчик C, у которого $C[j] = \max(C_1[j], C_2[j])$, оценивает мощность $A \cup B$. Это позволяет распределять вычисления по тысячам машин и в конце «склеивать» результаты.

Важно: краткое резюме

Современное измерение «социального расстояния» в больших сетях — это композиция нескольких идей:

1. теория графов даёт постановку задачи (кратчайшие расстояния);
2. теория чисел и хеширование (§5.5) обеспечивают «равномерное» распределение элементов;
3. теория вероятностей и алгоритм HyperLogLog позволяют оценивать мощности

множеств за десятки тысяч раз меньшую память;

4. распределённые вычисления позволяют разнести работу по тысячам компьютеров.

5.6.6 От социологии — к стратегическим выводам

Эксперимент Мильграма и его цифровое продолжение — не просто забавный факт. Из него следуют важные вещи:

- *Информация распространяется быстро.* Новость, попавшая в социальную сеть, при равномерной передаче «через знакомых» теоретически охватывает всю сеть за 4–5 шагов. На практике это и наблюдается с вирусными мемами.
- *Эпидемии распространяются быстро.* В аналогичной модели заражения болезнь, передающаяся при контакте, охватывает большую долю населения за считанные «волны» контактов.
- *Цифровая приватность хрупка.* Если ваш друг знает всё о вас, его друг — через пять рукопожатий — статистически связан с миллионами людей. Любая утечка приватной информации распространяется по сети быстрее, чем кажется.
- *Поиск работы и идей.* Знаменитая работа социолога Марка Грановеттера 1973 года показала, что новые контакты, идеи, рабочие места часто приходят к нам не от близких друзей, а от «слабых связей» — знакомых второго и третьего уровня. Малый диаметр сети объясняет, почему такая стратегия работает.

Это интересно: модель «малого мира» Уоттса–Строгаца

Математическая модель социальной сети, объясняющая эффект малого мира, была построена Дунканом Уоттсом и Стивеном Строгацем в 1998 году. Они показали: если у регулярной решётки добавить совсем немного случайных «дальних» рёбер, диаметр графа резко падает с линейного до логарифмического. Социальная сеть устроена именно так: большинство ваших связей — локальные (одноклассники, коллеги), но достаточно нескольких «дальних» — например, родственника в другом городе или знакомого по интернету, — чтобы вся сеть оказалась «компактной».

Задачи для самостоятельной работы

1. Объясните, почему в наивном алгоритме одиночного счётчика оценка $\widehat{n} = 2^M$ имеет такую большую дисперсию.
2. Пусть хеш-функция выдаёт строки по 8 битов. Сколько (в среднем) различных элементов нужно подать на вход, чтобы хотя бы у одного оказалось 6 нулей в начале хеша?
3. Объясните, почему гармоническое среднее устойчивее к выбросам, чем арифметическое. Приведите пример набора чисел, в котором эти средние существенно различаются.
4. Оцените, сколько памяти займёт HyperLogLog при $b = 12$, и какова при этом ожидаемая относительная погрешность.
5. * Почему в алгоритме оценки расстояний в графе нельзя просто хранить $|C_v|$ как число (это же 4 байта)? Зачем хранить целый HLL-счётчик?

6. * Среднее число рукопожатий в графе оценено как 4,74. Что в этой формулировке играет роль случайной величины, а что — константы? Какой стандартной операции теории вероятностей соответствует «среднее число»?

Подведение итогов главы

Теория чисел. Простые числа, остатки от деления, сравнения по модулю — основы модулярной арифметики. Алгоритм Евклида и его расширенная версия позволяют не только находить НОД, но и решать модулярные линейные уравнения. Быстрое возведение в степень за $O(\log k)$ операций позволяет работать с гигантскими числами; та же идея даёт логарифмический алгоритм для чисел Фибоначчи через возведение матрицы 2×2 в степень. Малая теорема Ферма $a^{p-1} \equiv 1 \pmod p$ — ключевое утверждение, обеспечивающее корректность RSA. Простых чисел много (закон $\pi(N) \sim N/\ln N$), и проверка простоты быстра (тест Миллера–Рабина, AKS), но *факторизация* составного числа на множители — задача экспоненциальной сложности.
История криптографии. От шифра Цезаря — к симметричным шифрам с одним общим ключом — к проблеме доставки ключа. Идея **односторонней функции** (легко в одну сторону, очень сложно в обратную) и **функции с секретом** (трапдор) даёт возможность строить асимметричные шифры с открытым и закрытым ключами.
RSA и Диффи–Хеллман. RSA опирается на сложность факторизации $n = pq$: открытый ключ (n, e) позволяет шифровать, закрытый $d = e^{-1} \bmod \varphi(n)$ — расшифровывать. Корректность доказывается малой теоремой Ферма. Диффи–Хеллман — протокол выработки общего ключа по открытому каналу: его стойкость основана на сложности задачи дискретного логарифма. Обе схемы уязвимы к атаке «человек посередине» без аутентификации.
Применения RSA. Та же конструкция, применённая в разных порядках, решает четыре задачи: шифрование сообщения, аутентификация пользователя через challenge-response, электронная цифровая подпись (подпись закрытым ключом, проверка открытым) и слепая подпись (основа протоколов электронного голосования).
Хеширование. Универсальное семейство Картера–Вегмана $h_a(x) = (a_1x_1 + \cdots + a_kx_k) \bmod p$ (p — простое) гарантирует, что вероятность коллизии любых двух ключей при случайном выборе a не превосходит $1/p$. Ключевую роль играет существование и единственность обратного элемента по простому модулю — результат расширенного алгоритма Евклида.
Вероятностные счётчики. HyperLogLog оценивает число различных элементов в потоке с точностью $\sim 1\%$, используя всего килобайты памяти. В сочетании с теорией графов это позволяет современным сетям подтверждать гипотезу Стенли Мильграма о малом мире: средняя длина «социальной» цепочки в крупных сетях — около *четырёх рукопожатий*.

Что почитать дальше

- *Музыкантский А. И., Фурин В. В.* Лекции по криптографии. М.: МЦНМО, разные годы. — Доступное изложение криптографических протоколов, в том числе электронного голосования; рекомендуется к разделу 1.3 и главе 2.

- *Литвак Н. В., Райгородский А. М.* Кому нужна математика? Содержательное введение в математику и computer science. — Главы 6, 7 и приложения к ним; материал о

малых мирах, графах и вероятностных алгоритмах.

- *Дасгупта С., Пападимитриу Х., Вазирани У.* Алгоритмы. — Глава о хешировании, главы об алгоритмах теории чисел.
- *Кнут Д.* Искусство программирования, том 2 (Получисленные алгоритмы). — Классический фундаментальный труд по алгоритмам теории чисел.
- *Виноградов И. М.* Основы теории чисел. — Классический российский учебник теории чисел, доступный школьникам старших классов.

Большой итоговый проект

В качестве длинного исследовательского проекта на каникулы или на год предлагаем выбрать одно из следующих направлений:

- **«Своя» реализация RSA.** На любом удобном языке программирования (Python, Java, C++) реализуйте полный цикл: генерацию пары ключей (e, n) и d (с честным алгоритмом Миллера–Рабина для подбора простых), шифрование, расшифрование, цифровую подпись. Протестируйте на примерах из этой главы.
- **Атака на «слабый» RSA.** Возьмите малое n (50–100 битов) и попытайтесь его факторизовать. Сравните скорость наивного перебора и метода Полларда ρ. Подумайте, при каких размерах n ваш ноутбук «сдаётся».
- **Симуляция «малого мира».** Сгенерируйте случайный граф из 1000–10000 вершин по модели Уоттса–Строгаца. Вычислите средние длины кратчайших путей. Сравните с предсказанием теории.
- **Свой HyperLogLog.** Реализуйте упрощённый вариант HyperLogLog ($b = 8$ или 10). Сгенерируйте поток из миллиона случайных идентификаторов с заранее известным числом уникальных. Оцените точность алгоритма.
- **Стойкость хеш-функции.** Возьмите криптографическую хеш-функцию (например, SHA-1 уже взломан) и реализуйте поиск коллизий простым перебором. Сравните с теоретической оценкой «парадокса дней рождения».

Литература и веб-ресурсы

1. *Канторович Л. В., Акилов Г. П.* Функциональный анализ. — М.: Наука, 1984.
2. *Бахвалов Н. С., Жидков Н. П., Кобельков Г. М.* Численные методы. — М.: БИНОМ, 2011.
3. *Nocedal J., Wright S.* Numerical Optimization. — Springer, 2006.
4. *Виноградов И. М.* Основы теории чисел. — Любое издание. — М.: Наука.
5. *Музыкантский А. И., Фурин В. В.* Лекции по криптографии. — М.: МЦНМО.
6. *Литвак Н. В., Райгородский А. М.* Кому нужна математика? Содержательное введение в математику и computer science. — М.: Манн, Иванов и Фербер, 2017.
7. *Дасгупта С., Пападимитриу Х., Вазирани У.* Алгоритмы. — М.: МЦНМО, 2014.
8. *Кнут Д. Э.* Искусство программирования. Том 2. Получисленные алгоритмы. — М.: Вильямс, разные издания.
9. Diffie W., Hellman M. E. New Directions in Cryptography // IEEE Transactions on Information Theory, 1976.
10. Rivest R., Shamir A., Adleman L. A method for obtaining digital signatures and public-key cryptosystems // Communications of the ACM, 1978.
11. Flajolet P., Fusy É., Gandouet O., Meunier F. HyperLogLog: the analysis of a near-optimal cardinality estimation algorithm // Discrete Mathematics and Theoretical Computer Science, 2007.
12. Документация Python: https://docs.python.org/3/.
13. Открытый онлайн-курс «Методы оптимизации»: https://fmin.xyz.
14. Описание оптимизатора Muon: https://kellerjordan.github.io/posts/muon/.