3.3 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.*

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 канони-

ческому 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. 3.13).

Примеры рукописных цифр (датасет UCI Optical Digits, 8×8)

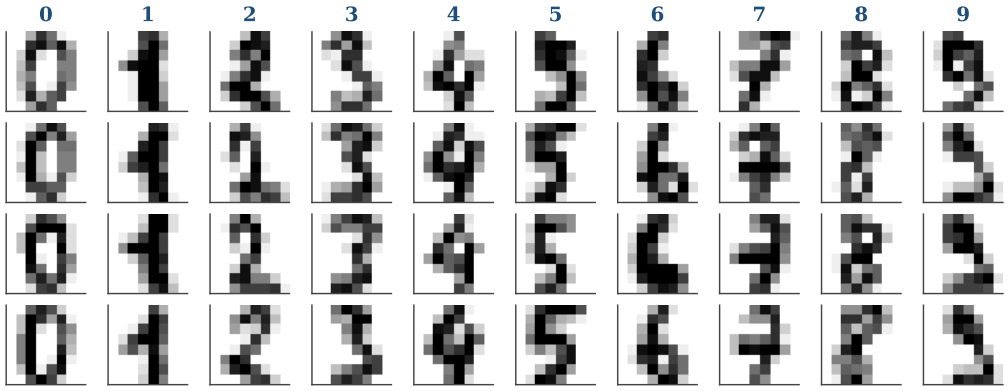

Рисунок 1. Рис. 3.13. Примеры изображений из датасета UCI Optical Digits: 10 классов цифр от 0 до 9, каждое изображение — $8 \times 8 = 64$ пикселя в оттенках серого. Внутри каждого класса написания заметно различаются — это и есть «трудность задачи»: модель должна обобщать, а не запоминать.

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

$$\mathbf{x} = (x_1, x_2, \ldots, x_{64}) \in \mathbb{R}^{64}. \tag{1}$$

Так каждая цифра превращается в точку 64-мерного пространства. Для полноразмерного MNIST — 784-мерного (рис. 3.14).

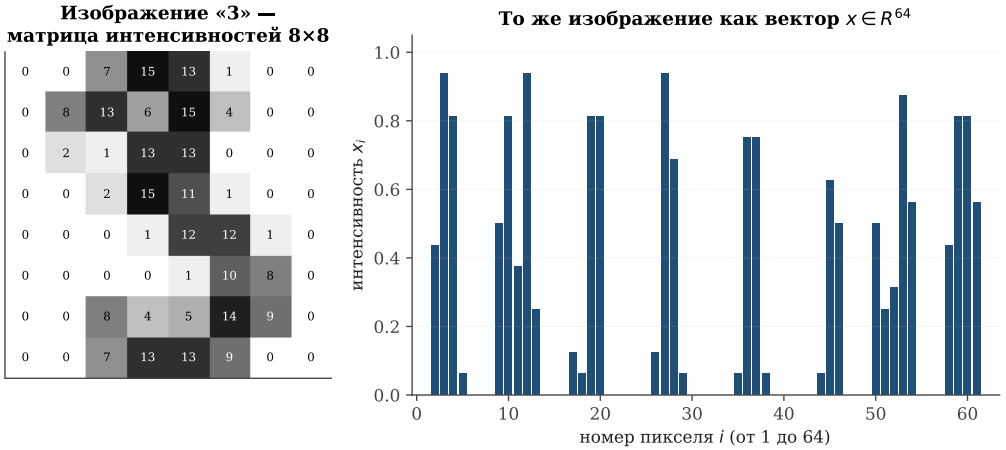

Рисунок 2. Рис. 3.14. Одно и то же изображение цифры в двух представлениях. Слева — матрица интенсивностей 8×8. Справа — та же цифра, развёрнутая в вектор $\mathbf{x} \in \mathbb{R}^{64}$ (по столбцам). Модели машинного обучения работают именно со вторым представлением — вектором чисел.

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

$$\{(\mathbf{x}_i, y_i)\}_{i=1}^{N}, \qquad \mathbf{x}_i \in \mathbb{R}^{64}, \quad y_i \in \{0, \ldots, 9\}. \tag{2}$$

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

Чем классификация отличается от регрессии

В задаче регрессии (3.2) 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:

$$\mathbf{p}(\mathbf{x}) = \big(\, p_0(\mathbf{x}), p_1(\mathbf{x}), \ldots, p_9(\mathbf{x}) \,\big), \qquad p_c \geq 0, \quad \sum_{c=0}^{9} p_c = 1. \tag{3}$$

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 $\mathbf{x}_i$:

$$\Pr(y_i = c \,|\, \mathbf{x}_i, \boldsymbol{\theta}) = p_c(\mathbf{x}_i; \boldsymbol{\theta}). \tag{4}$$

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

$$L(\boldsymbol{\theta}) = \prod_{i=1}^{N} p_{y_i}(\mathbf{x}_i; \boldsymbol{\theta}). \tag{5}$$

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

$$\mathcal{L}(\boldsymbol{\theta}) \stackrel{\text{def}}{=} -\frac{1}{N} \sum_{i=1}^{N} \ln p_{y_i}(\mathbf{x}_i; \boldsymbol{\theta}) \to \min_{\boldsymbol{\theta}}. \tag{6}$$

Этот функционал называется **кросс-энтропией**[^1] (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.

> **i** Почему «кросс-энтропия»?
>
> В теории информации Клод Шеннон (1948) ввёл понятие *энтропии* распределения $\mathbf{q}$:
>
> $$H(\mathbf{q}) = -\sum_{c} q_c \ln q_c, \tag{7}$$
>
> характеризующее «непредсказуемость» исхода. *Кросс-энтропия* двух распределений $\mathbf{q}$ (истинное) и $\mathbf{p}$ (предсказанное) определяется как
>
> $$H(\mathbf{q}, \mathbf{p}) = -\sum_{c} q_c \ln p_c. \tag{8}$$
>
> 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 (3.29) оказывается средней по выборке кросс-энтропией.

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*).

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

Определение 3.10. 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

$$(A\mathbf{z})_i = \sum_{j=1}^{n} a_{ij}\, z_j, \qquad i = 1, \dots, m. \tag{9}$$

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

Пример 3.11. Простой расчёт

$$A = \begin{pmatrix} 1 & 2 & -1 \\ 3 & 0 & 4 \end{pmatrix}, \qquad \mathbf{z} = \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix}. \tag{10}$$

Тогда

$$A\mathbf{z} = \begin{pmatrix} 1\cdot 1 + 2\cdot 2 + (-1)\cdot 3 \\ 3\cdot 1 + 0\cdot 2 + 4\cdot 3 \end{pmatrix} = \begin{pmatrix} 2 \\ 15 \end{pmatrix}. \tag{11}$$

Теперь можем определить полносвязную сеть (рис. 3.15).

Историческая справка

Идея «искусственного нейрона» — линей- я от входов, пропущенная через ть — восходит к работам **Уорре- лока и Уолтера Питтса** (1943) и *персептрону* Фрэнка Розенблат- озенблатт построил физическую rk I Perceptron на лампах и фото- пресса 1958 г. восторженно писа- шина скоро будет ходить, разго- осознавать своё существование». 69 г. Марвин Минский и Сеймур книге *Perceptrons* строго доказа- н-единственный слой не может даже простейшую функцию — *е ИЛИ* (XOR). Это охладило инте- сетям почти на 15 лет.

е пришло в 1986 г., когда Дэвид Джеффри Хинтон и Рональд едставили универсальный алго- ния *многослойных* сетей — ме- ого распространения ошибки ation). С этого момента и начина- современных нейронных сетей. , что общая математическая фор- что вычислительная сложность адиента сравнима со сложностью исления функции, в общем слу- строго установлена советскими ми: **Юрий Геннадьевич Ким, Юрий Евгеньевич Нестеров, Александр Скоков, Виктор Черкасский** (работы начала 1980-х). Об этом подробнее ниже, в разделе 3.22.

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».

i Определение 3.12. Полносвязная нейронная сеть

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

$$ $$

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:

$$\operatorname{softmax}(\mathbf{z})_c = \frac{e^{z_c}}{\sum_{c'} e^{z_{c'}}}. \tag{12}$$

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

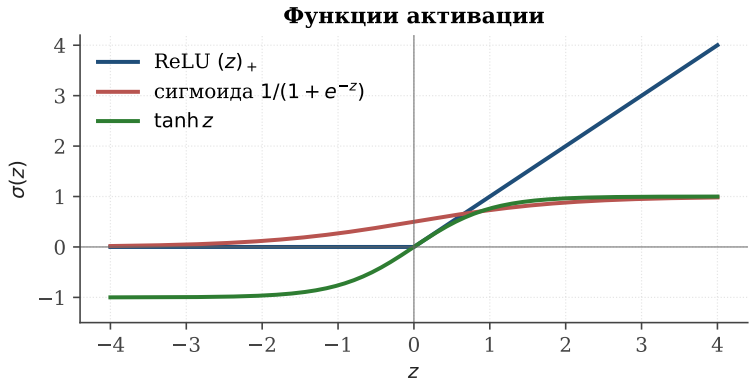

Рисунок 3. Рис. 3.15. Три распространённые функции активации. Сигмоида и гиперболический тангенс «насыщаются» при больших $|z|$ (производная стремится к нулю), что исторически затрудняло обучение глубоких сетей. ReLU $\sigma(z) = \max(0, z)$ — предложенная Хинтоном с соавторами в 2010 г. существенно 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):

$$\#\boldsymbol{\theta} = \underbrace{64 \cdot 32 + 32}_{\text{слой 1}} + \underbrace{32 \cdot 10 + 10}_{\text{слой 2}} = 2080 + 330 = 2410. \tag{13}$$

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.

Обучение MLP и кривые потерь

Минимизация функционала (3.29) 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 3.21; 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

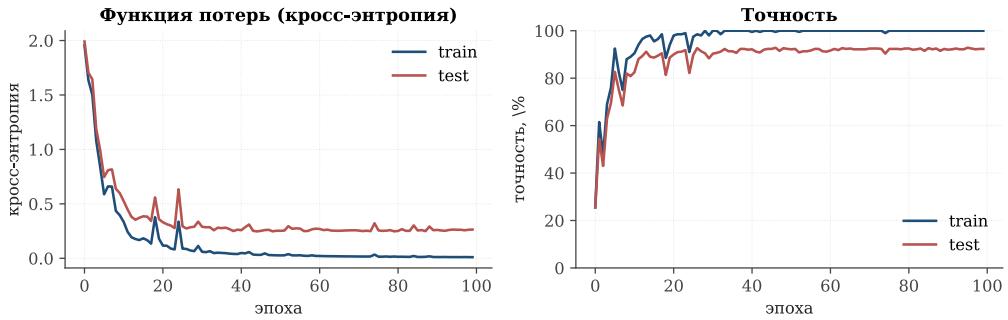

Рисунок 4. Рис. 3.16. Обучение MLP $64 \to 32 \to 10$ на 200 примерах из UCI Digits. Слева: кросс-энтропия по эпохам — на обучающей выборке (синяя) и на тестовой (красная). Справа: точность классификации, в %. Видно характерное поведение: после быстрого начального снижения потерь кривая «выходит на полку», 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.

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**.

i Определение 3.13. 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.

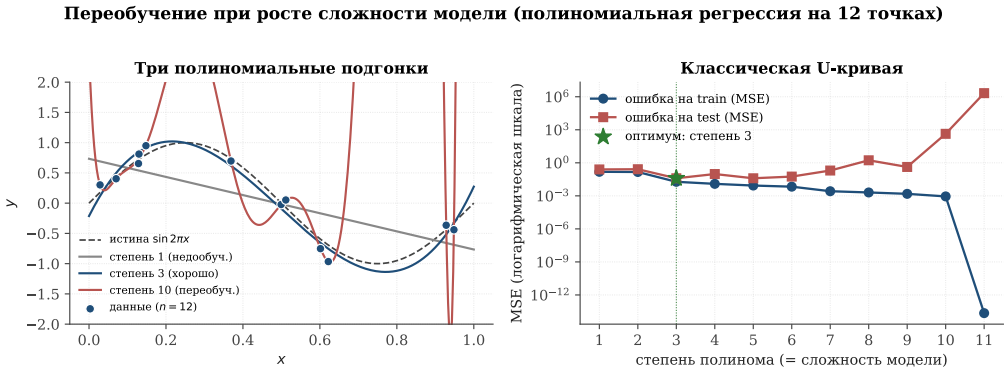

Рисунок 5. Рис. 3.17. Классическая U-образная кривая. *Слева*: три полиномиальные подгонки той же выборки из 12 точек. Степень 1 (прямая, серая) — *недообучение* (high bias): модель слишком простая. Степень 3 (синяя) — хороший компромисс, почти совпадает с истинной кривой $\sin 2\pi x$. Степень 10 (красная) 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.

> **i** Зачем три выборки, а не две?
>
> - **Обучающая (train)** — на ней непосредственно подбирают параметры θ (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.

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 3.14. Теорема Колмогорова–Арнольда (1957)

Всякая непрерывная функция $f : [0,1]^n \to \mathbb{R}$ от n переменных представима в виде суперпозиции непрерывных функций *одной* переменной и операции сложения:

$$f(x_1, \ldots, x_n) = \sum_{q=0}^{2n} \Phi_q \left(\sum_{p=1}^{n} \psi_{q,p}(x_p) \right), \tag{14}$$

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 (3.34) 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 3.15. Теорема Цыбенко (1989)

Пусть $\sigma : \mathbb{R} \to \mathbb{R}$ — 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

$$g(\mathbf{x}) = \sum_{j=1}^{N} a_j \sigma\left(\sum_{i=1}^{n} w_{j,i}\, x_i + b_j \right) \tag{15}$$

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. 3.18).

Глубина vs ширина

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.

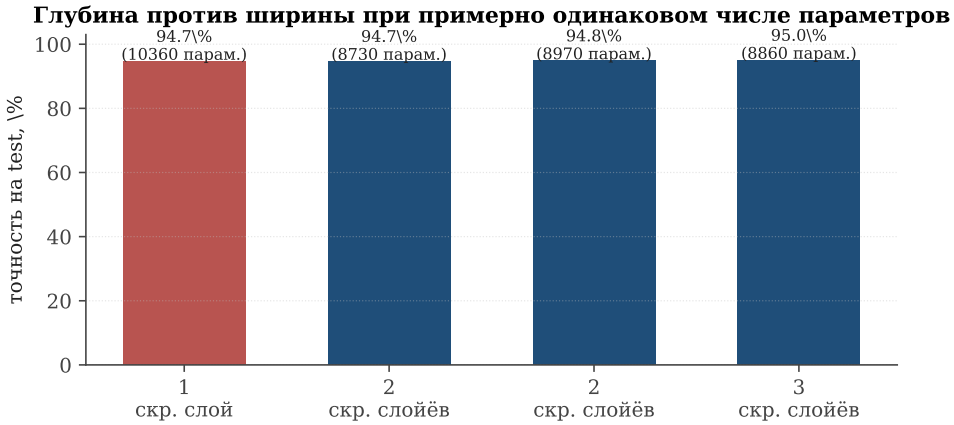

Рисунок 6. Рис. 3.18. Эмпирическое сравнение на UCI Digits: четыре MLP с примерно *одинаковым* числом параметров (~ 9000), но разной *глубиной* — 1, 2, 3 и 4 скрытых слоя. При фиксированном бюджете параметров глубокие модели работают чуть лучше неглубоких. Разница невелика (UCI Digits — простая задача), 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 %.

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.

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:

$$Y_{r,c} = \sum_{u=1}^{k} \sum_{v=1}^{k} K_{u,v}\, X_{r+u-1,\, c+v-1}. \tag{16}$$

Фильтр «скользит» по всему изображению; в итоге получается новая карта признаков, чувствительная к тому, что «видит» фильтр (см. рис. 3.19 — четыре фильтра, обученные на UCI Digits, очевидно выделяют локальные «уголки» и «полосы»).

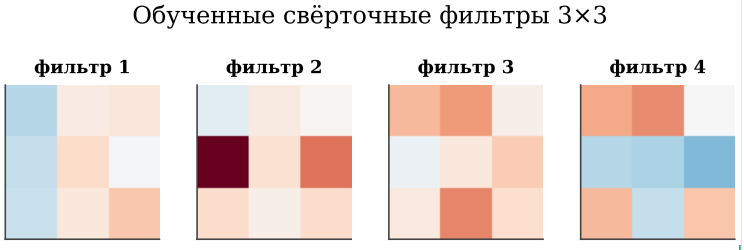

Рисунок 7. Рис. 3.19. Четыре обученных свёрточных фильтра 3×3 из простой CNN, обученной на UCI Digits. Тёмно-красные клетки — большие положительные веса, синие — большие отрицательные. Каждый фильтр «заточен» под свой локальный признак (наклонная линия, перепад яркости в определённом направлении и т. д.).

Определение 3.16. Свёрточная нейронная сеть (CNN)

Свёрточная нейронная сеть — это нейронная сеть, в которой некоторые слои представляют собой не полносвязные матрично-векторные умножения, а *свёртки* вида (3.36) 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.

Историческая справка

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».

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. 3.20.

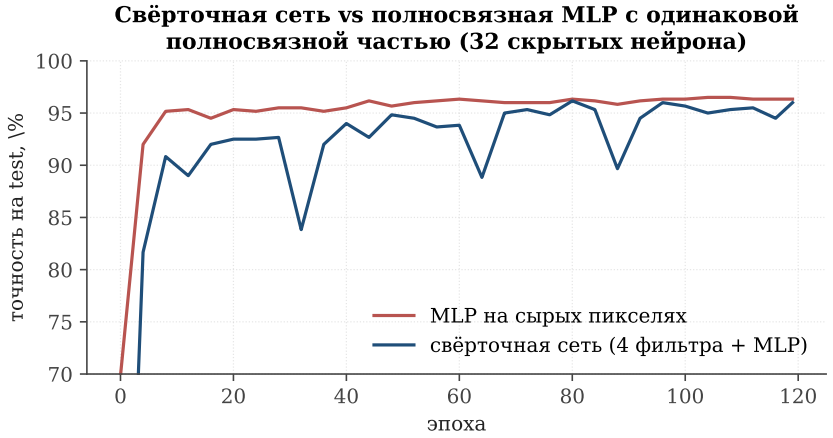

Рисунок 8. Рис. 3.20. Сравнение точности на тесте для полносвязной MLP и простой CNN с теми же финальными скрытыми слоями. На крошечных изображениях 8×8 из UCI Digits разница невелика — задача слишком проста, локальная структура почти не помогает. На полноразмерном MNIST 28×28 CNN превосходит MLP на 1–2 процентных пункта; на цветных изображениях ImageNet (224×224) — уже на 10–15 процентных пунктов.

Градиент и градиентный спуск

Перейдём к ключевому вопросу: *как же подобрать параметры $\boldsymbol{\theta}$, минимизирующие функционал (3.29)*? Аналитическое решение, как было в МНК (3.10), 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 $F : \mathbb{R}^d \to \mathbb{R}$ — 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 θ по направлению $\mathbf{v}$** называется

$$\partial_{\mathbf{v}} F(\boldsymbol{\theta}) = \lim_{t \to 0} \frac{F(\boldsymbol{\theta} + t\mathbf{v}) - F(\boldsymbol{\theta})}{t}. \tag{17}$$

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

$$\partial_{\mathbf{v}} F = \varphi'(0) = \sum_{j=1}^{d} \frac{\partial F}{\partial \theta_j} \cdot v_j = \langle \nabla F, \mathbf{v} \rangle, \tag{18}$$

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

⚠ Теорема 3.17. 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

$$\mathbf{v}^\star = -\frac{\nabla F(\boldsymbol{\theta})}{\| \nabla F(\boldsymbol{\theta}) \|}. \tag{19}$$

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**:

$$\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k - \alpha \, \nabla F(\boldsymbol{\theta}_k) \tag{20}$$

Здесь $\alpha > 0$ — *шаг обучения* (*learning rate*), гиперпараметр. Слишком маленький — сходимость медленная; слишком большой — итерации могут «перескакивать» через минимум и расходиться (рис. 3.21).

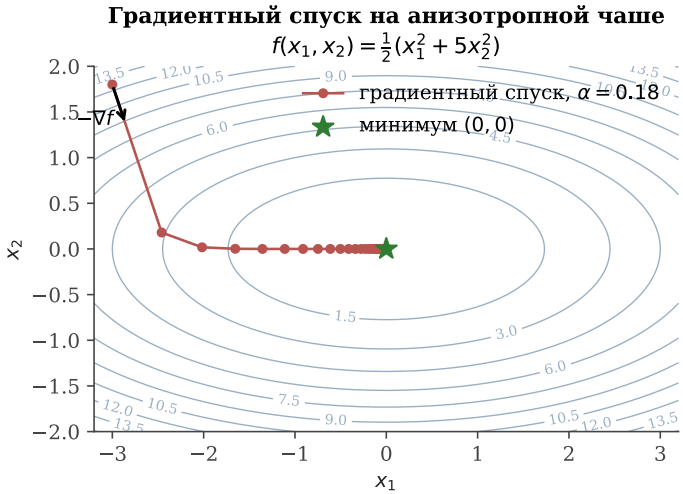

Рисунок 9. Рис. 3.21. Траектория градиентного спуска для квадратичной функции $F(x_1, x_2) = \frac{1}{2}(x_1^2 + 5x_2^2)$. Линии уровня — эллипсы (функция «вытянута» вдоль оси x_1). Метод стартует из точки $(-3, 2)$ и за 20 итераций приходит к минимуму в $(0, 0)$. Видно характерное «зигзагообразное» движение, типичное для плохо обусловленных задач.

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 $B_k \subset \{1, 2, \ldots, N\}$ из m примеров, — и шаг делается по *приближённому* градиенту, вычисленному только по этому батчу:

$$\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k - \alpha\,\widehat{\nabla\mathcal{L}}(\boldsymbol{\theta}_k), \qquad \widehat{\nabla\mathcal{L}} = \frac{1}{m}\sum_{i\in B_k}\nabla\ell_i. \tag{21}$$

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.

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

$$\mathbf{x} \to \mathbf{z}^{(1)} \to \mathbf{a}^{(1)} \to \mathbf{z}^{(2)} \to \ldots \to \mathbf{p} \to \mathcal{L}. \tag{22}$$

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.

Идея алгоритма для MLP.

Историческая справка

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**

Рассмотрим сеть из определения 3.12. 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 (3.31), (3.32). Это — один обычный прогон сети.

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

$$\boldsymbol{\delta}^{(\ell)} \stackrel{\text{def}}{=} \frac{\partial \ell}{\partial \mathbf{z}^{(\ell)}} \in \mathbb{R}^{d_\ell}. \tag{23}$$

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:

$$\boldsymbol{\delta}^{(L)} = \mathbf{p} - \mathbf{e}_y, \tag{24}$$

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

$$\boldsymbol{\delta}^{(\ell-1)} = \bigl(W^{(\ell)}\bigr)^{\top} \boldsymbol{\delta}^{(\ell)} \odot \sigma'\bigl(\mathbf{z}^{(\ell-1)}\bigr), \tag{25}$$

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

$$\frac{\partial \ell}{\partial z_j^{(\ell-1)}} = \sum_i \frac{\partial \ell}{\partial z_i^{(\ell)}} \cdot \frac{\partial z_i^{(\ell)}}{\partial a_j^{(\ell-1)}} \cdot \frac{\partial a_j^{(\ell-1)}}{\partial z_j^{(\ell-1)}} = \sum_i \delta_i^{(\ell)} \cdot W_{i,j}^{(\ell)} \cdot \sigma'\bigl(z_j^{(\ell-1)}\bigr). \tag{26}$$

Собирая всё по j в один вектор, получаем именно (3.43). 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:

$$\begin{aligned} \frac{\partial \ell}{\partial W^{(\ell)}} &= \boldsymbol{\delta}^{(\ell)} \left(\mathbf{a}^{(\ell-1)} \right)^{\top}, \\ \frac{\partial \ell}{\partial \mathbf{b}^{(\ell)}} &= \boldsymbol{\delta}^{(\ell)}. \end{aligned} \tag{27}$$

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

Стоимость алгоритма. Каждая из формул (3.43), (3.44) — 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,

$$\text{стоимость подсчёта градиента} \le C \cdot (\text{стоимость подсчёта функции}), \tag{28}$$

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

> **i** Что именно проверять, если backprop «сломан»
>
> В практике глубокого обучения один из любимых рецептов отладки backpropagation — *численная проверка градиента*:
>
> $$\frac{\partial \ell}{\partial \theta_j} \approx \frac{\ell(\theta_j + h) - \ell(\theta_j - h)}{2h}, \qquad h \sim 10^{-5}. \tag{29}$$
>
> 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.

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 Кол-*

могорова–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 (3.42): 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. 3.16.
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. 3.17.
7. ⋆ Докажите формулу (3.43) полностью, начиная с цепного правила $\partial\ell/\partial z_j^{(\ell-1)} = \sum_k (\partial\ell/\partial z_k^{(\ell)}) \cdot (\partial z_k^{(\ell)}/\partial z_j^{(\ell-1)})$.