ИСКУССТВЕННЫЕ НЕЙРОННЫЕ СЕТИ

Лекция 4

Функции потерь, градиентный спуск и обратное распространение ошибки

Обучение многослойного перцептрона:
математический вывод, численный расчёт и NumPy

Преподаватель: Аршын Алтыбай

Кафедра компьютерной инженерии,
Алматинский технологический университет, Алматы, Казахстан
Учебный год: 2026

> **Связь с лекцией 3.** Ранее мы вычисляли выход сети по заданным весам (forward propagation). Теперь ответим на главный вопрос: *как подобрать веса и смещения так, чтобы прогнозы совпадали с обучающими данными?* Для этого введём функцию потерь, градиентный метод оптимизации и алгоритм обратного распространения, являющийся систематическим применением цепного правила дифференцирования [7, 6].

1. Цели и результаты обучения

Цель: математически вывести и программно реализовать обучение многослойной нейронной сети с помощью градиентного спуска и обратного распространения ошибки.

После лекции студент должен уметь:

1. формулировать эмпирическую задачу минимизации и отделять функцию потерь от метрики качества;
2. вычислять производные MSE, бинарной и многоклассовой кросс-энтропии;
3. объяснять смысл градиента, скорости обучения и направления антиградиента;
4. различать batch, stochastic и mini-batch gradient descent;
5. выводить формулы backpropagation для одного и нескольких скрытых слоёв;

6. вручную просчитывать один шаг обновления весов небольшой сети;
7. реализовывать алгоритм на NumPy и проверять градиент конечными разностями;
8. распознавать проблемы затухающих и взрывающихся градиентов, плохого масштаба признаков и переобучения.

2. Постановка задачи обучения нейронной сети

Рассмотрим обучающую выборку $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^{N}$, где $x_i \in \mathbb{R}^d$, y_i — целевой ответ. Веса и смещения всех слоёв объединяются в параметр θ. Прогноз $\hat{y}_i = f_\theta(x_i)$ сравнивается с y_i функцией потерь $\ell(\hat{y}_i, y_i)$. Цель обучения обычно записывается как

$$J(\theta) = \frac{1}{N}\sum_{i=1}^{N} \ell(f_\theta(x_i), y_i) + \lambda\Omega(\theta), \qquad \theta^* \in \arg\min_\theta J(\theta). \tag{1}$$

Здесь Ω — регуляризатор, а $\lambda \geq 0$ задаёт его влияние. Результат обучения, вообще говоря, является *локальным* или приближённым решением задачи, поскольку J для глубокой сети обычно невыпукла [6].

> **Три объекта, которые нельзя смешивать** **Модель** f_θ переводит вход в прогноз. **Функция потерь** ℓ задаёт сигнал для обучения. **Метрика** (например accuracy или F1) сообщает, насколько модель полезна в прикладной задаче. Метрика может быть недифференцируемой и потому не использоваться напрямую как loss.

В supervised learning каждое наблюдение имеет целевую метку. При обучении без учителя целевой сигнал задаётся иначе (например, реконструкция входа автоэнкодером); общий принцип минимизации функции цели сохраняется. От выбора ℓ зависит, какие ошибки модель штрафует сильнее [3, 4].

3. Функции потерь для основных типов задач

3.1 Среднеквадратичная ошибка (MSE)

Для регрессии с одним выходом используется

$$\ell_{\text{MSE}}(\hat{y}, y) = \frac{1}{2}(\hat{y} - y)^2, \qquad \frac{\partial \ell}{\partial \hat{y}} = \hat{y} - y. \tag{2}$$

Фактор $1/2$ упрощает производную; некоторые библиотеки используют $(\hat{y} - y)^2$ без него. Для m выходов можно взять $\ell = \frac{1}{2m}\|\hat{y} - y\|_2^2$. Конкретная нормировка изменяет масштаб градиента, но не положение минимума при отсутствии иных слагаемых.

> **Ручной расчёт** При $y = 3$, $\hat{y} = 2$ получаем $\ell = \frac{1}{2}(2-3)^2 = 0.5$, $\partial \ell / \partial \hat{y} = -1$. Отрицательная производная означает: локальное увеличение прогноза уменьшит ошибку.

MSE особенно чувствительна к большим отклонениям: ошибка 4 даёт штраф, в 16 раз больший, чем ошибка 1. В задачах с выбросами рассматривают, например, MAE или Huber loss. MAE не имеет обычной производной в нуле, но допускает субградиентные методы.

3.2 Бинарная кросс-энтропия (binary cross-entropy)

Для $y \in \{0, 1\}$ и прогнозируемой вероятности $p \in (0, 1)$:

$$\ell_{\text{BCE}}(p, y) = -y \log p - (1 - y) \log(1 - p). \tag{3}$$

При $p = \sigma(z) = 1/(1 + e^{-z})$ производные имеют вид

$$\frac{\partial \ell}{\partial p} = -\frac{y}{p} + \frac{1 - y}{1 - p}, \qquad \frac{\partial p}{\partial z} = p(1 - p).$$

Следовательно,

$$\boxed{\frac{\partial \ell_{\text{BCE}}}{\partial z} = p - y.} \tag{4}$$

Это точное сокращение справедливо для согласованной пары *sigmoid + BCE*. В реальных библиотеках предпочтительны устойчивые вычисления непосредственно по логитам (например BCEWithLogitsLoss), исключающие явные $\log(0)$ и ненужное численное округление [5, 4].

3.3 Softmax и многоклассовая кросс-энтропия

Если $z \in \mathbb{R}^K$ — логиты классов, то

$$p_k = \mathrm{softmax}(z)_k = \frac{e^{z_k}}{\sum_{j=1}^{K} e^{z_j}}, \qquad \ell = -\sum_{k=1}^{K} y_k \log p_k.$$

Для one-hot меток y производная по логитам равна

$$\boxed{\nabla_z \ell = p - y.} \tag{5}$$

Для устойчивости вычисляют $\mathrm{softmax}(z) = \mathrm{softmax}(z - c)$ при $c = \max_j z_j$; это не меняет вероятности. Формула (5) лежит в основе обучения многоклассовых классификаторов [6, 3].

Задача	Выход модели	Типичная потеря
Регрессия	Вещественное число, часто без активации	MSE, MAE, Huber
Бинарная классификация	Один логит или вероятность	Binary cross-entropy
Многоклассовая классификация	K логитов, softmax-вероятности	Categorical cross-entropy

4. Градиент как направление наибольшего возрастания

Для дифференцируемой функции $J : \mathbb{R}^P \to \mathbb{R}$ градиент

$$\nabla J(\theta) = \left(\frac{\partial J}{\partial \theta_1}, \ldots, \frac{\partial J}{\partial \theta_P} \right)^\top$$

задаёт направление максимального локального возрастания. По разложению Тейлора

$$J(\theta + \Delta\theta) = J(\theta) + \nabla J(\theta)^\top \Delta\theta + o(\|\Delta\theta\|).$$

Если $\Delta\theta = -\eta \nabla J(\theta)$, $\eta > 0$, то линейная часть изменения равна $-\eta \|\nabla J(\theta)\|^2 \leq 0$. Отсюда метод градиентного спуска:

$$\boxed{\theta^{(t+1)} = \theta^{(t)} - \eta \nabla J(\theta^{(t)}).} \tag{6}$$

Гарантия уменьшения J для *конечного* шага требует ограничений на гладкость и достаточно малой η; слишком большой шаг может увеличить потери.

4.1 Точный пример квадратичной функции

Пусть $J(w) = \frac{1}{2}(w - 3)^2$, тогда $J'(w) = w - 3$ и

$$w_{t+1} = w_t - \eta(w_t - 3) = (1 - \eta) w_t + 3\eta.$$

При $w_0 = 0$, $\eta = 0.2$:

$$w_1 = 0.6, \qquad w_2 = 1.08, \qquad w_3 = 1.464.$$

Ошибка удовлетворяет $w_t - 3 = (1 - \eta)^t (w_0 - 3)$; для этого примера сходимость есть при $0 < \eta < 2$. Это *не* универсальная граница для нейронных сетей.

4.2 Batch, SGD и mini-batch

Полный градиент использует все N объектов:

$$g_{\text{batch}} = \frac{1}{N}\sum_{i=1}^{N}\nabla_\theta \ell_i.$$

Стохастический градиент использует один случайный объект, mini-batch — подвыборку B:

$$g_B = \frac{1}{|B|}\sum_{i\in B}\nabla_\theta \ell_i, \qquad \theta \leftarrow \theta - \eta g_B. \tag{7}$$

Mini-batch обычно сочетает эффективные матричные операции и умеренную дисперсию оценки градиента. Если выборка случайна по установленной схеме, g_B может быть несмещённой оценкой градиента эмпирической потери, но это не означает обязательного убывания loss на каждом mini-batch [3, 6].

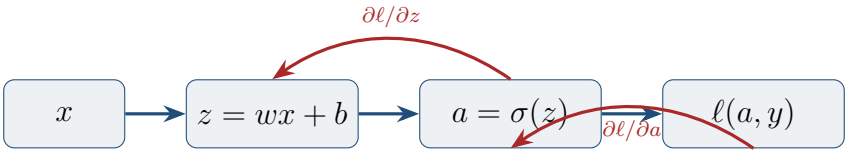
5. Цепное правило — фундамент backpropagation

Алгоритм backpropagation *не является отдельным методом оптимизации*. Его задача — эффективно найти производные $\partial J/\partial W$ и $\partial J/\partial b$. Метод оптимизации (например, SGD или Adam) затем использует найденные производные для изменения параметров.

Для композиции $u = u(v)$, $v = v(w)$ выполняется

$$\frac{\partial J}{\partial w} = \frac{\partial J}{\partial u}\frac{\partial u}{\partial v}\frac{\partial v}{\partial w}.$$

Для многих параметров вместо независимого вычисления всех цепочек используют переиспользование промежуточных производных — принцип обратного режима автоматического дифференцирования [7, 6].



6. Обратное распространение для одного нейрона

Пусть $z = wx + b$, $a = \sigma(z)$, $\ell = \frac{1}{2}(a - y)^2$. Из цепного правила

$$\frac{\partial \ell}{\partial a} = a - y, \qquad \frac{\partial a}{\partial z} = a(1 - a), \tag{8}$$

$$\frac{\partial z}{\partial w} = x, \qquad \frac{\partial z}{\partial b} = 1. \tag{9}$$

Поэтому

$$\frac{\partial \ell}{\partial w} = (a - y)a(1 - a)x, \qquad \frac{\partial \ell}{\partial b} = (a - y)a(1 - a). \tag{10}$$

Удобно ввести $\delta = \partial \ell / \partial z$. Тогда $\partial \ell / \partial w = \delta x$, $\partial \ell / \partial b = \delta$. Для sigmoid+BCE вместо $\delta = (a - y)a(1 - a)$ получаем упрощение $\delta = a - y$.

> **Почему сокращение важно** Дополнительный множитель $a(1 - a)$ в MSE+sigmoid становится маленьким при насыщении sigmoid. У пары BCE+sigmoid формула производной по логиту проще; однако даже при этом общая проблема затухания градиентов в глубоких сетях не исчезает.

7. Backpropagation для сети с одним скрытым слоем

Рассмотрим полносвязную сеть с входом $x \in \mathbb{R}^d$, скрытым слоем ширины h и выходом ширины k:

$$z^{(1)} = W^{(1)}x + b^{(1)}, \quad a^{(1)} = \phi(z^{(1)}), \quad z^{(2)} = W^{(2)}a^{(1)} + b^{(2)}, \quad \hat{y} = \psi(z^{(2)}). \tag{11}$$

Размерности: $W^{(1)} \in \mathbb{R}^{h \times d}$, $W^{(2)} \in \mathbb{R}^{k \times h}$. Пусть $J = \ell(\hat{y}, y)$. Введём *ошибку слоя* $\delta^{(\ell)} = \nabla_{z^{(\ell)}} J$.

7.1 Шаг 1. Ошибка выходного слоя

Сначала вычисляем

$$\delta^{(2)} = \nabla_{\hat{y}} \ell \odot \psi'(z^{(2)}). \tag{12}$$

Если выходная функция softmax сочетается с cross-entropy, используется $\delta^{(2)} = \hat{y} - y$. Для линейного выхода и MSE с нормировкой $1/2$ также имеем $\delta^{(2)} = \hat{y} - y$.

7.2 Шаг 2. Ошибка скрытого слоя

Каждая скрытая активация влияет на все выходные нейроны, поэтому требуется транспонированная матрица весов:

$$\boxed{\delta^{(1)} = \left((W^{(2)})^\top \delta^{(2)}\right) \odot \phi'(z^{(1)}).} \tag{13}$$

Символ $\odot$ обозначает поэлементное умножение. Важно использовать *старые* веса $W^{(2)}$ при вычислении всех градиентов одного backward pass; обновление начинается после завершения их расчёта.

7.3 Шаг 3. Градиенты по параметрам

По правилу дифференцирования аффинного преобразования:

$$\boxed{\nabla_{W^{(2)}} J = \delta^{(2)}(a^{(1)})^\top, \quad \nabla_{b^{(2)}} J = \delta^{(2)}, \quad \nabla_{W^{(1)}} J = \delta^{(1)} x^\top, \quad \nabla_{b^{(1)}} J = \delta^{(1)}.} \tag{14}$$

Здесь $\delta^{(2)}(a^{(1)})^\top$ — внешнее произведение $k \times 1$ и $1 \times h$, дающее матрицу $k \times h$. Аналогично $\delta^{(1)} x^\top$ имеет размер $h \times d$.

> **Запоминание формул.** Для каждого слоя: «ошибка» δ умножается на *вход предыдущего слоя*; градиент веса есть внешнее произведение, градиент смещения равен δ. Для перехода к предыдущему слою ошибка умножается на транспонированную матрицу следующего слоя и производную активации.

8. Общая рекурсия для произвольного числа слоёв

Пусть $a^{(0)} = x$, а для $\ell = 1, \dots, L$ выполнено

$$z^{(\ell)} = W^{(\ell)} a^{(\ell-1)} + b^{(\ell)}, \qquad a^{(\ell)} = \phi_\ell(z^{(\ell)}).$$

После прямого прохода вычисляем ошибку выходного слоя $\delta^{(L)}$ и затем в обратном порядке

$$\boxed{\delta^{(\ell)} = \left((W^{(\ell+1)})^\top \delta^{(\ell+1)}\right) \odot \phi'_\ell(z^{(\ell)}), \quad \ell = L-1, \dots, 1.} \tag{15}$$

Для каждого слоя

$$\boxed{\nabla_{W^{(\ell)}} J = \delta^{(\ell)} (a^{(\ell-1)})^\top, \qquad \nabla_{b^{(\ell)}} J = \delta^{(\ell)}.} \tag{16}$$

При batch из m объектов градиенты суммируются и делятся на m (если функция потерь определяется средним). Регуляризационное слагаемое добавляет отдельную производную, например для $\Omega = \frac{1}{2} \sum_\ell \|W^{(\ell)}\|_F^2$ градиент дополняется на $\lambda W^{(\ell)}$.

9. Подробный численный пример: один полный шаг обучения

Рассмотрим сеть 2–2–1 с двумя входами, ReLU-активацией скрытого слоя и *линейным* выходом. Выберем

$$x = \begin{pmatrix} 1 \\ 2 \end{pmatrix}, \qquad y = 1, \quad W^{(1)} = \begin{pmatrix} 0.2 & 0.1 \\ -0.1 & 0.3 \end{pmatrix}, \quad b^{(1)} = \begin{pmatrix} 0 \\ 0.1 \end{pmatrix}, \quad W^{(2)} = \begin{pmatrix} 0.5 & -0.4 \end{pmatrix}, \quad b^{(2)} = 0.2.$$

Шаг A: прямое распространение.

$$z^{(1)} = W^{(1)}x + b^{(1)} = \begin{pmatrix} 0.4 \\ 0.6 \end{pmatrix}, \qquad a^{(1)} = \mathrm{ReLU}(z^{(1)}) = \begin{pmatrix} 0.4 \\ 0.6 \end{pmatrix}.$$

$$\hat{y} = 0.5 \cdot 0.4 - 0.4 \cdot 0.6 + 0.2 = 0.16.$$

При $\ell = \frac{1}{2}(\hat{y} - y)^2$ имеем $\ell = \frac{1}{2}(-0.84)^2 = 0.3528$.

Шаг B: ошибка выхода. Поскольку выход линейный,

$$\delta^{(2)} = \hat{y} - y = -0.84.$$

Отсюда

$$\nabla_{W^{(2)}}\ell = -0.84 \begin{pmatrix} 0.4 & 0.6 \end{pmatrix} = \begin{pmatrix} -0.336 & -0.504 \end{pmatrix}, \qquad \nabla_{b^{(2)}}\ell = -0.84.$$

Шаг C: ошибка скрытого слоя. Оба $z_j^{(1)} > 0$, поэтому $\mathrm{ReLU}'(z_j^{(1)}) = 1$:

$$\delta^{(1)} = \begin{pmatrix} 0.5 \\ -0.4 \end{pmatrix} (-0.84) \odot \begin{pmatrix} 1 \\ 1 \end{pmatrix} = \begin{pmatrix} -0.42 \\ 0.336 \end{pmatrix}.$$

Получаем

$$\nabla_{W^{(1)}}\ell = \begin{pmatrix} -0.42 \\ 0.336 \end{pmatrix} \begin{pmatrix} 1 & 2 \end{pmatrix} = \begin{pmatrix} -0.42 & -0.84 \\ 0.336 & 0.672 \end{pmatrix}, \qquad \nabla_{b^{(1)}}\ell = \begin{pmatrix} -0.42 \\ 0.336 \end{pmatrix}.$$

Шаг D: изменение параметров. При $\eta = 0.1$ новые веса равны

$$W^{(2)}_{\text{new}} = \begin{pmatrix} 0.5336 & -0.3496 \end{pmatrix}, \qquad b^{(2)}_{\text{new}} = 0.284,$$

$$W^{(1)}_{\text{new}} = \begin{pmatrix} 0.242 & 0.184 \\ -0.1336 & 0.2328 \end{pmatrix}, \qquad b^{(1)}_{\text{new}} = \begin{pmatrix} 0.042 \\ 0.0664 \end{pmatrix}.$$

Новый forward pass даёт $z^{(1)}_{\text{new}} = (0.652, 0.3984)^\top$, $\hat{y}_{\text{new}} = 0.49262656$ и

$$\ell_{\text{new}} = \frac{1}{2}(0.49262656 - 1)^2 \approx 0.128714.$$

Потеря уменьшилась с 0.3528 до ≈ 0.1288. Это полезная иллюстрация, но для произвольного шага и произвольной сети убывание потерь не гарантировано.

10. Реализация численного примера на NumPy

Пример 1. Один шаг backpropagation для сети 2-2-1

```python
import numpy as np

x = np.array([1., 2.])
y = 1.0
W1 = np.array([[.2, .1], [-.1, .3]])
b1 = np.array([0., .1])
W2 = np.array([[.5, -.4]])
b2 = np.array([.2])
eta = 0.1

def forward(W1, b1, W2, b2):
    z1 = W1 @ x + b1
    a1 = np.maximum(z1, 0.)
    yh = (W2 @ a1 + b2).item()
    return z1, a1, yh

z1, a1, yh = forward(W1, b1, W2, b2)
loss_before = .5 * (yh-y)**2

d2 = yh - y
# ReLU derivative is well-defined here: z1 > 0
d1 = (W2.T[:, 0] * d2) * (z1 > 0)
dW2 = (d2 * a1)[None, :]
db2 = np.array([d2])
dW1 = np.outer(d1, x)
db1 = d1

W1 = W1 - eta * dW1
b1 = b1 - eta * db1
W2 = W2 - eta * dW2
b2 = b2 - eta * db2
_, _, yh_new = forward(W1, b1, W2, b2)
loss_after = .5 * (yh_new-y)**2
print('prediction:', round(yh, 6), '->', round(yh_new, 6))
print('loss:', round(loss_before, 6), '->', round(loss_after, 6))
```

Ожидаемый вывод

```
prediction: 0.16 -> 0.492627
loss: 0.3528 -> 0.128714
```

11. Векторизация: обучение mini-batch

Для пакета $X \in \mathbb{R}^{m\times d}$, где объекты расположены *по строкам*, удобно писать

$$Z^{(1)} = X(W^{(1)})^\top + \mathbf{1}_m (b^{(1)})^\top, \quad A^{(1)} = \phi(Z^{(1)}),$$
$$Z^{(2)} = A^{(1)}(W^{(2)})^\top + \mathbf{1}_m (b^{(2)})^\top.$$

Если $D^{(2)}$ — матрица производных потерь по логитам выходного слоя *по отдельным объектам*, то для усреднённой функции потерь

$$\nabla_{W^{(2)}} J = \frac{1}{m}(D^{(2)})^\top A^{(1)}, \qquad \nabla_{b^{(2)}} J = \frac{1}{m}\sum_{i=1}^{m} D^{(2)}_{i,:}. \tag{17}$$

Аналогичные формулы применяются ко всем слоям. Главное — *не делить на m дважды*: если библиотека уже сформировала среднюю loss и вычислила её градиент, дополнительный множитель $1/m$ ошибочен.

12. Минимальная обучаемая сеть XOR на чистом NumPy

Пример реализует двухслойный MLP с одним скрытым слоем, tanh и sigmoid+BCE. Все производные рассчитаны аналитически; библиотека автодифференцирования не применяется. Для устойчивости loss считается по логитам формулой $\log(1+e^z) - yz = \mathrm{logaddexp}(0, z) - yz$.

Пример 2. Обучение XOR с аналитическим backpropagation

```python
import numpy as np
rng = np.random.default_rng(7)
X = np.array([[0.,0.],[0.,1.],[1.,0.],[1.,1.]])
y = np.array([[0.],[1.],[1.],[0.]])
W1 = rng.normal(0, 0.8, size=(2, 4))
b1 = np.zeros((1, 4))
W2 = rng.normal(0, 0.8, size=(4, 1))
b2 = np.zeros((1, 1))
eta, m = 0.5, len(X)

for epoch in range(12000):
```

Пример 2 (продолжение)

```
    Z1 = X @ W1 + b1
    A1 = np.tanh(Z1)
    Z2 = A1 @ W2 + b2
    P = 1 / (1 + np.exp(-np.clip(Z2, -60, 60)))
    # d(loss)/d(Z2) for sigmoid + BCE (mean over rows)
    dZ2 = (P - y) / m
    dW2 = A1.T @ dZ2
    db2 = dZ2.sum(axis=0, keepdims=True)
    dZ1 = (dZ2 @ W2.T) * (1 - A1*A1)
    dW1 = X.T @ dZ1
    db1 = dZ1.sum(axis=0, keepdims=True)
    W1 -= eta * dW1
    b1 -= eta * db1
    W2 -= eta * dW2
    b2 -= eta * db2

A1 = np.tanh(X @ W1 + b1)
Z2 = A1 @ W2 + b2
P = 1/(1 + np.exp(-Z2))
print('predicted labels:', (P[:, 0] >= 0.5).astype(int))
print('probabilities:', np.round(P[:, 0], 3))
```

Проверка воспроизводимости. Результаты зависят от инициализации и числа эпох. Код задаёт фиксированный seed и выводит итоговые метки/вероятности; проверять следует прежде всего правильность классификации $[0, 1, 1, 0]$ и уменьшение loss, а не совпадение промежуточных весов с чужой реализацией.

13. Проверка аналитического градиента конечными разностями

Из-за сложной композиции матричных операций легко ошибиться со знаком, транспонированием или коэффициентом усреднения. Для скалярного параметра θ_j центральная конечная разность задаёт приближение

$$g_j^{\mathrm{FD}} = \frac{J(\theta + \varepsilon e_j) - J(\theta - \varepsilon e_j)}{2\varepsilon} = \frac{\partial J}{\partial \theta_j} + O(\varepsilon^2) \tag{18}$$

при достаточной гладкости. Обычно берут $\varepsilon \sim 10^{-5}$–10^{-6} для double precision, но слишком малое ε увеличивает ошибки округления. Для ReLU в точке $z = 0$ производная не определена; сравнение там может быть неинформативным.

Пример 3. Проверка одного элемента матрицы градиента

```python
import numpy as np
x = np.array([1., 2.]); y = 1.
W1 = np.array([[.2, .1],[-.1,.3]])
b1 = np.array([0.,.1]); W2 = np.array([[.5,-.4]])
b2 = np.array([.2])

def loss(W):
    hidden = np.maximum(W @ x + b1, 0.)
    pred = (W2 @ hidden + b2).item()
    return .5 * (pred-y)**2

h = np.maximum(W1 @ x + b1, 0.)
delta2 = (W2 @ h + b2).item()-y
delta1 = W2[0] * delta2
analytic = delta1[0]*x[0]
eps = 1e-5
plus = W1.copy(); minus = W1.copy()
plus[0, 0] += eps; minus[0, 0] -= eps
finite_difference = (loss(plus)-loss(minus))/(2*eps)
print(round(analytic, 6), round(finite_difference, 6))
```

Ожидаемый вывод

```
-0.42 -0.42
```

14. Почему обучение глубоких сетей бывает нестабильным

14.1 Затухающие и взрывающиеся градиенты

Для глубокой сети формула (15) последовательно умножает векторы на транспонированные матрицы весов и производные активаций. Нормы произведений могут стремиться к нулю или расти. У sigmoid имеем $0 < \sigma'(z) \leq 1/4$; в длинных цепочках это может существенно ослабить градиент. Большие нормы матриц и неудачный масштаб приводят к взрывному росту. Это одна из причин популярности ReLU-подобных активаций, продуманной инициализации, нормализации и gradient clipping [6, 4].

14.2 Влияние масштаба признаков

Пусть входные признаки измеряются в диапазонах $[0, 1]$ и $[0, 10^6]$. Тогда градиенты разных компонент весов могут существенно различаться по масштабу. Стандартизация по обучающей выборке

$$x'_j = (x_j - \mu_j^{\text{train}})/(s_j^{\text{train}} + \epsilon)$$

обычно облегчает оптимизацию. Параметры μ_j, s_j нельзя оценивать на тестовой выборке во избежание утечки данных.

14.3 Инициализация весов и насыщение

Все нулевые веса в одном скрытом слое сохраняют симметрию нейронов: одинаковые нейроны получают одинаковые градиенты. Случайная инициализация нарушает симметрию. Для ReLU часто применяют He initialization, для tanh — Xavier/Glorot; величины зависят от числа входов/выходов слоя [3, 6].

14.4 Недообучение и переобучение

Underfitting: training loss остаётся большой, модель не описывает обучающие закономерности. *Overfitting*: training loss уменьшается, но качество на независимой validation выборке ухудшается. Методы контроля: early stopping, регуляризация, увеличение данных, настройка архитектуры. Validation участвует в выборе параметров; test должен оставаться для итоговой оценки.

15. Оптимизаторы и скорость обучения

15.1 SGD с импульсом

Одна из форм momentum:

$$v_t = \beta v_{t-1} + g_t, \qquad \theta_{t+1} = \theta_t - \eta v_t,$$

где g_t — gradient mini-batch и $\beta \in [0, 1)$ — коэффициент памяти. Различные библиотеки могут использовать эквивалентные схемы с иным расположением множителя η.

15.2 Adam: адаптивный шаг

Adam сочетает экспоненциальные оценки первого и второго моментов градиента:

$$m_t = \beta_1 m_{t-1} + (1-\beta_1) g_t, \qquad v_t = \beta_2 v_{t-1} + (1-\beta_2) g_t^{\odot 2},$$

$$\hat{m}_t = \frac{m_t}{1-\beta_1^t}, \quad \hat{v}_t = \frac{v_t}{1-\beta_2^t}, \quad \theta_{t+1} = \theta_t - \eta \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon}.$$

Adam часто удобен как стартовый оптимизатор, но не гарантирует лучшего итогового обобщения или оптимальности для любой задачи. Более подробно оптимизаторы рассматриваются в следующих темах курса [5, 3].

Алгоритм	Идея	Что настраивают
SGD	Шаг по градиенту текущего mini-batch	Learning rate, batch size
Momentum	Сглаженная накопленная направленность шага	Learning rate, momentum
Adam	Адаптация шага по оценкам моментов	Learning rate, β_1, β_2, ϵ

16. Практический алгоритм одной эпохи обучения

1. Перемешать индексы *обучающих* объектов и сформировать mini-batches.
2. Для каждого mini-batch выполнить прямое распространение через все слои.
3. Вычислить loss (и при необходимости регуляризационный штраф).
4. Найти ошибки δ в обратном порядке и градиенты всех весов и смещений.
5. Применить один шаг выбранного оптимизатора *после* расчёта соответствующих градиентов.
6. В конце эпохи вычислить отдельно training и validation loss/metrics.
7. Сохранить лучшие параметры по заранее определённому критерию (например по validation loss).

Псевдокод выражает общую структуру независимо от фреймворка. В PyTorch градиенты обычно вычисляются через `loss.backward()`, после чего вызывается `optimizer.step()`, но концептуально применяется тот же цепной закон [5].

17. Типичные ошибки студентов при back-propagation

1. **Неверный знак**: прибавляют $\eta \nabla J$ вместо вычитания.
2. **Перепутанные формы**: матрица $\delta a^\top$ подменяется $a\delta^\top$.
3. **Не та производная**: для ReLU используют $\sigma'(z)$ или забывают маску $z > 0$.
4. **Несогласованные output/loss**: sigmoid применяется второй раз перед BCEWithLogitsLoss.
5. **Двойное усреднение**: делят на размер batch после того, как loss уже возвращает среднее.
6. **Раннее обновление**: изменяют веса выходного слоя прежде, чем передали ошибку в скрытый.
7. **Обучение на test**: многократно подбирают гиперпараметры по тестовой выборке.
8. **Плохая проверка**: оценивают только точность, не строя learning curves и не выполняя gradient checking.

18. Итоги лекции

> **Главная идея** Обучение MLP состоит из трёх логически различных операций: **forward** вычисляет прогноз и loss; **backward** по цепному правилу находит производные по всем параметрам; **optimizer** изменяет параметры, стремясь уменьшить целевую функцию. Back-propagation является эффективным способом вычисления градиентов, а не отдельной функцией потерь и не отдельным оптимизатором.

Ключевые формулы: (6) — градиентный спуск, (15) — передача ошибки между слоями, (16) — градиенты весов и смещений. На практике корректность проверяют как численно (gradient checking), так и экспериментально (training/validation curves) [7, 1].

19. Контрольные вопросы

1. Что минимизируется при supervised learning?
2. Чем функция потерь отличается от метрики качества?
3. Почему для регрессии часто используют MSE?
4. Какова производная $\frac{1}{2}(\hat{y} - y)^2$ по $\hat{y}$?
5. Запишите формулу бинарной кросс-энтропии.
6. Почему sigmoid вместе с BCE даёт производную $p - y$ по логиту?
7. Какая производная получается у softmax + cross-entropy?
8. Что означает градиент многомерной функции?
9. Как получается формула градиентного спуска из первого порядка Тейлора?
10. Почему слишком большой learning rate может быть опасен?
11. Чем batch gradient descent отличается от SGD?
12. Что означает mini-batch?
13. Почему backpropagation не называют оптимизатором?
14. Как применяется цепное правило к одному нейрону?
15. Что обозначает вектор $\delta^{(\ell)}$?
16. Почему при переходе к предыдущему слою используется транспонированная матрица?
17. Запишите градиент $\nabla_{W^{(\ell)}} J$.
18. Как определить размерность градиента весовой матрицы?
19. Что такое gradient checking и когда он может дать неверный сигнал?
20. Почему нельзя обновлять следующий слой до завершения backward pass?
21. В чём причины затухающих градиентов?
22. Что такое exploding gradients?
23. Как нормализация входных признаков помогает обучению?
24. Чем early stopping отличается от изменения архитектуры?
25. Какую задачу решают momentum и Adam?

20. Задания для самостоятельной работы

1. Вручную вычислите $\partial \ell / \partial w$ и $\partial \ell / \partial b$ для $z = wx + b$, $a = \sigma(z)$, $\ell = \frac{1}{2}(a - y)^2$ при $x = 2$, $w = 0.3$, $b = -0.1$, $y = 1$.
2. Для сети $3 \to 4 \to 2$ выпишите размерности всех W, b, z, a, δ и матриц градиентов; вычислите число обучаемых параметров.
3. Проверьте вручную все четыре градиента численного примера 2–2–1; повторите шаг с $\eta = 0.01$ и $\eta = 1$ и сравните потери.
4. Дополните NumPy-пример полноценной функцией `train_step` и автоматической проверкой градиентов всех элементов $W^{(1)}$.
5. Обучите сеть XOR для различных seed, ширин скрытого слоя и learning rate. Сохраните результаты не менее пяти экспериментов.
6. Сформулируйте, почему использование тестовой выборки при настройке числа эпох нарушает корректный протокол оценки.
7. Реализуйте график loss против эпох для собственного mini-batch обучения и объясните причины возможных колебаний.

21. Библиография и дальнейшее чтение

По вычислениям «с нуля» рекомендуются [1, 2, 7]; для матричного описания и теории аппроксимации — [6]; для практической реализации — [3, 4, 5].

References

[1] Trask A. W. *Grokking Deep Learning*. Manning Publications, 2019. Русское издание: «Грокаем глубокое обучение». https://www.manning.com/books/grokking-deep-learning.

[2] Rashid T. *Make Your Own Neural Network*. 2016. Русское издание: «Глубокое обучение. Лёгкая теория и практика на Python» (название может варьироваться по изданию). https://makeyourownneuralnetwork.blogspot.com/.

[3] Géron A. *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow*. 3rd ed., O'Reilly Media, 2022. Русское издание: «Прикладное машинное обучение с помощью Scikit-Learn, Keras и TensorFlow». https://www.oreilly.com/library/view/hands-on-machine-learning/9781098125967/.

[4] Chollet F. *Deep Learning with Python*. 2nd ed., Manning Publications, 2021. Русское издание: «Глубокое обучение на Python». https://www.manning.com/books/deep-learning-with-python-second-edition.

[5] Stevens E., Antiga L., Viehmann T. *Deep Learning with PyTorch*. Manning Publications, 2020. Русское издание: «Глубокое обучение с PyTorch». https://www.manning.com/books/deep-learning-with-pytorch.

[6] Goodfellow I., Bengio Y., Courville A. *Deep Learning*. MIT Press, 2016. Русское издание: «Глубокое обучение». https://www.deeplearningbook.org/.

[7] Nielsen M. A. *Neural Networks and Deep Learning*. 2015. Бесплатный онлайн-учебник: «Нейронные сети и глубокое обучение». http://neuralnetworksanddeeplearning.com/.