ВЫЧИСЛИТЕЛЬНЫЕ МЕТОДЫ · ЛЕКЦИЯ 7

Введение в машинное обучение

Персептрон, нейронные сети и обучение

Лю Шисян

Кафедра теплофизики (Э6)
МГТУ им. Н.Э. Баумана
Москва / 2026

Место этой лекции в курсе

Лекция 7 — Основы машинного обучения

В предыдущих лекциях мы задавали **алгоритм вычислений явно**. Теперь перейдём к моделям, параметры которых определяются **автоматически на основе данных**:

данные → **модель** → **ошибка** → **обучение**.

После лекции вы сможете:

- различать задачи **регрессии** и **классификации**;
- объяснять роль **модели**, **функции потерь** и **оптимизации**;
- понимать принцип работы **персептрона** и многослойной нейронной сети;
- объяснять, как данные используются для **обучения** и **проверки** модели.

План

TOC

1

Персептрон

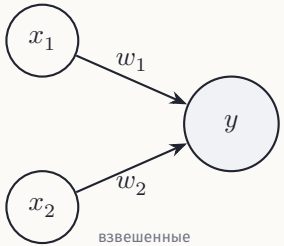
Что такое персептрон

Определение

Персептрон — простейшая модель искусственного нейрона.
Он получает входные сигналы

$$x_1, x_2, \dots, x_d,$$

умножает их на веса и формирует один выход y.

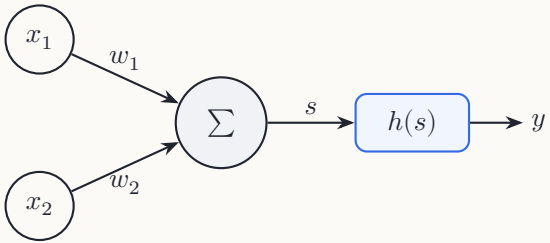


- каждому входу x_i соответствует вес w_i;
- вес определяет **вклад входа** в решение;
- в классическом персептроне

$$y \in \{0, 1\}.$$

Персептрон = взвешенная сумма + правило принятия решения.

Математическая модель персептрона



1. Взвешенная сумма

$$s = w_1 x_1 + w_2 x_2.$$

2. Пороговое решение

$$y = \begin{cases} 0, & s \le \theta, \\ 1, & s > \theta. \end{cases}$$

- x_i — входные сигналы;
- w_i — веса;
- θ — порог активации.

➠ Меняя только параметры w_i и θ, можно получать различные логические функции.

Код: perceptron.ipynb

Логический элемент AND (И)

Правило

AND выдаёт 1 **только тогда**, когда оба входа активны:

$$(x_1, x_2) = (1, 1) \quad \Longrightarrow \quad y = 1.$$

Выбираем параметры

$$(w_1, w_2, \theta) = (0.5,\ 0.5,\ 0.7),$$

поэтому

$$s = 0.5x_1 + 0.5x_2.$$

Только в последнем случае

$$s = 1.0 > \theta = 0.7.$$

Таблица истинности

x_1	x_2	y
0	0	0
1	0	0
0	1	0
1	1	**1**

Логический элемент NAND (НЕ–И)

Правило

NAND — отрицание AND:

$$(x_1, x_2) = (1, 1) \quad \Longrightarrow \quad y = 0,$$

а во всех остальных случаях $y = 1$.

Выбираем параметры

$$(w_1, w_2, \theta) = (-0.5,\ -0.5,\ -0.7),$$

поэтому

$$s = -0.5x_1 - 0.5x_2.$$

Только для двух активных входов

$$s = -1.0 \leq \theta = -0.7.$$

Таблица истинности

x_1	x_2	y
0	0	1
1	0	1
0	1	1
1	1	**0**

Логический элемент OR (ИЛИ)

Правило

OR выдаёт 1, если активен **хотя бы один** вход:

$$x_1 = 1 \quad \text{или} \quad x_2 = 1.$$

Ноль получается только при $(0, 0)$.

Выбираем параметры

$$(w_1, w_2, \theta) = (0.5,\ 0.5,\ 0.4),$$

поэтому

$$s = 0.5x_1 + 0.5x_2.$$

Уже один активный вход даёт

$$s = 0.5 > \theta = 0.4.$$

Таблица истинности

x_1	x_2	y
0	0	0
1	0	**1**
0	1	**1**
1	1	**1**

От порога к смещению

До сих пор решение записывалось через порог:

$$s = \mathbf{w}^T \mathbf{x}, \qquad y = \begin{cases} 0, & s \le \theta, \\ 1, & s > \theta. \end{cases}$$

Удобнее перенести порог внутрь линейного выражения:

$$z = \mathbf{w}^T \mathbf{x} + b, \qquad b = -\theta$$

Тогда

$$y = \begin{cases} 0, & z \le 0, \\ 1, & z > 0. \end{cases}$$

Веса w

Определяют, **как входные признаки влияют** на результат:

$$w_i x_i.$$

Смещение b

Сдвигает границу решения и определяет, **где происходит переключение** между классами.

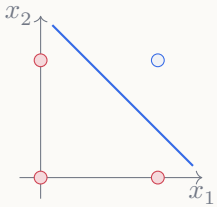
$z = \mathbf{w}^T \mathbf{x} + b$ — **стандартная форма линейного нейрона**

Геометрический смысл персептрона

Для двух входов условие

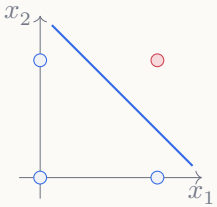
$$w_1 x_1 + w_2 x_2 + b = 0$$

задаёт **прямую**, разделяющую пространство входов на два класса.



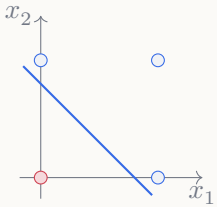
AND

$x_1 + x_2 = 1.4$



NAND

$x_1 + x_2 = 1.4$



OR

$x_1 + x_2 = 0.8$

Синие точки соответствуют $y = 1$, красные точки — $y = 0$.

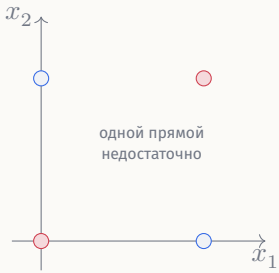
Один персептрон строит одну линейную границу решения.

XOR: задача, которую один персептрон не решает

Исключающее ИЛИ

XOR выдаёт 1, когда **ровно один** из входов равен 1.

x_1	x_2	y
0	0	0
1	0	**1**
0	1	**1**
1	1	0



- точки двух классов расположены по диагонали;
- **ни одна прямая** не может отделить синие точки от красных;
- следовательно, XOR — **линейно неразделимая** задача.

Чтобы решить XOR, нужно объединить несколько линейных границ.

Как построить XOR из простых элементов?

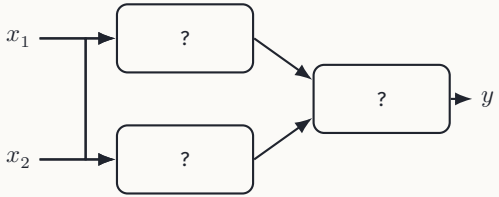
Мы уже умеем реализовывать с помощью персептрона три функции:

AND, NAND, OR.

Задача

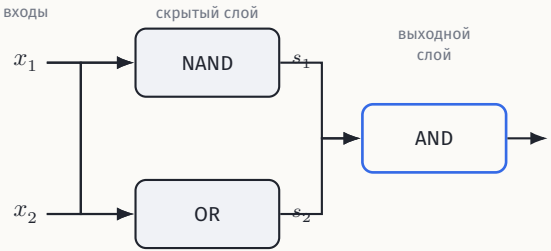
Можно ли **соединить несколько простых элементов** так, чтобы получить функцию XOR?

Выход должен быть равен 1, **только если один из двух входов равен** 1.



Подсказка: сначала сформируйте два промежуточных сигнала s_1 и s_2, затем объедините их.

XOR с помощью нескольких персептронов



$$
\begin{aligned}
s_1 &= \mathsf{NAND}(x_1, x_2),\\
s_2 &= \mathsf{OR}(x_1, x_2),\\
y &= \mathsf{AND}(s_1, s_2).
\end{aligned}
$$

Проверка

x_1	x_2	s_1	s_2	y
0	0	1	0	0
1	0	1	1	**1**
0	1	1	1	**1**
1	1	0	1	0

Теперь используются **несколько связанных персептронов**.
NAND и OR формируют промежуточные сигналы s_1 и s_2.
Они образуют **скрытый слой**, а AND формирует выход y.

От ручного подбора к обучению

До сих пор

Для простых логических функций мы сами выбирали параметры: $\mathbf{w}, b$.

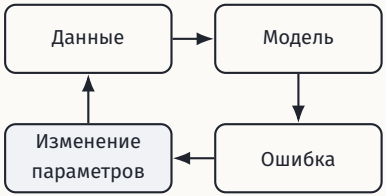
Но для реальной задачи:

- входных данных может быть очень много;
- параметров модели могут быть тысячи или миллионы;
- подходящие значения заранее неизвестны.

Идея машинного обучения

Параметры модели можно не задавать вручную.

Компьютер **автоматически изменяет w** и b, используя данные и информацию об ошибке.



Обучение — это автоматический подбор параметров модели на основе данных.

2

Нейронные сети

Переход к нейронным сетям

Что мы уже знаем?

Один персептрон вычисляет

$$z = \mathbf{w}^T \mathbf{x} + b$$

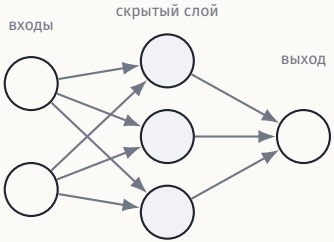
и принимает решение на основе значения z.

- один персептрон задаёт одну линейную границу решения;
- несколько персептронов можно объединять;
- параметры **w** и b можно обучать по данным.

Следующий шаг

Объединим нейроны в **слои**:

вход $\longrightarrow$ скрытые слои $\longrightarrow$ выход.



Нейронная сеть — это система связанных нейронов, организованных в слои.

Нейрон как вычислительный блок

Нейрон выполняет два последовательных шага.

1. Линейная комбинация

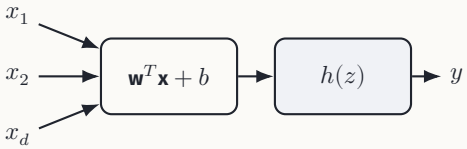
$$z = w_1 x_1 + w_2 x_2 + \cdots + w_d x_d + b$$

или короче

$$z = \mathbf{w}^T \mathbf{x} + b.$$

2. Функция активации

$$y = h(z)$$



Главная идея

Веса и смещение формируют значение z, а функция $h(z)$ преобразует его в выход нейрона.

Нейрон = линейное преобразование + функция активации.

Зачем нужна функция активации?

Для классического персептрона мы использовали **ступенчатую функцию**:

$$h(z) = \begin{cases} 0, & z \leq 0, \\ 1, & z > 0. \end{cases}$$

Она сразу превращает число z в решение:

$$z \quad \longrightarrow \quad h(z) \quad \longrightarrow \quad 0 \text{ или } 1.$$

Без активации

Нейрон выполнял бы только

$$z = \mathbf{w}^T \mathbf{x} + b,$$

то есть линейное преобразование.

С активацией

Функция $h(z)$ позволяет добавить **нелинейность** и строить более сложные зависимости.

Функция активации определяет, как нейрон преобразует полученный сигнал.

Почему нужна нелинейность?

Представим два последовательных слоя без функций активации:

$$\mathbf{z}_1 = W_1\mathbf{x} + \mathbf{b}_1,$$
$$\mathbf{z}_2 = W_2\mathbf{z}_1 + \mathbf{b}_2.$$

Подставим первое выражение во второе:

$$\mathbf{z}_2 = W_2\left(W_1\mathbf{x} + \mathbf{b}_1\right) + \mathbf{b}_2 = W\mathbf{x} + \mathbf{b}$$

— снова одно линейное преобразование.

Что меняет функция активации?

Если между слоями использовать нелинейную функцию

$$\mathbf{h}_1 = h\left(W_1\mathbf{x} + \mathbf{b}_1\right),$$

то несколько слоёв уже нельзя свести к одному линейному преобразованию.

Глубина сети становится полезной благодаря нелинейным функциям активации.

Примеры функций активации

Step

$$h(x) = \begin{cases} 0, & x \leq 0, \\ 1, & x > 0. \end{cases}$$

Резкое переключение между двумя состояниями.

Использовалась в классическом персептроне.

Sigmoid

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

Плавно отображает вход в интервал

$$0 < \sigma(x) < 1.$$

Часто используется на выходе при бинарной классификации.

ReLU

$$\text{ReLU}(x) = \max(0, x)$$

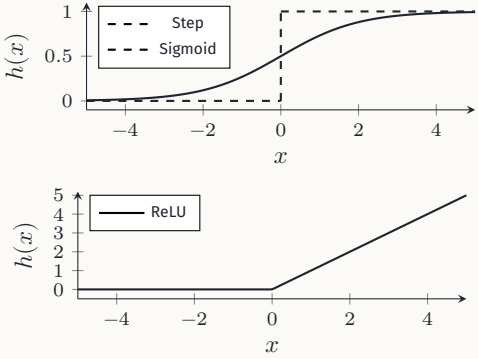
$$\text{ReLU}(x) = \begin{cases} 0, & x \leq 0, \\ x, & x > 0. \end{cases}$$

Широко используется в скрытых слоях нейронных сетей.

Разные функции активации по-разному преобразуют сигнал, но их общая задача — добавить нелинейность.

Step, Sigmoid и ReLU в Python

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

def step(x):
    return (x > 0).astype(int)

def sigmoid(x):
    return 1 / (1 + np.exp(-x))

def relu(x):
    return np.maximum(0, x)

x = np.linspace(-5, 5, 200)

plt.plot(x, step(x), label="Step")
plt.plot(x, sigmoid(x), label="Sigmoid"
    )
plt.plot(x, relu(x), label="ReLU")
```



От одного нейрона к слою

Если в слое несколько нейронов, у каждого есть свои веса и смещение.

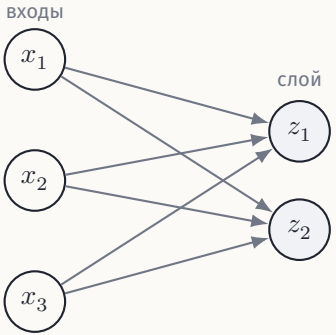
$$z_1 = \mathbf{w}_1^T \mathbf{x} + b_1,$$
$$z_2 = \mathbf{w}_2^T \mathbf{x} + b_2.$$

Обе формулы удобно объединить:

$$\mathbf{z} = W\mathbf{x} + \mathbf{b}$$

где

$$W = \begin{bmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \end{bmatrix}.$$



Матрица W объединяет веса всех нейронов слоя.

Матричное умножение в NumPy

Пусть

$$A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}, \qquad B = \begin{bmatrix} 5 & 6 \\ 7 & 8 \end{bmatrix}.$$

Тогда

$$AB = \begin{bmatrix} 19 & 22 \\ 43 & 50 \end{bmatrix}.$$

В NumPy оператор

$$A@B$$

означает матричное умножение.

```python
import numpy as np

A = np.array([
    [1, 2],
    [3, 4]
])

B = np.array([
    [5, 6],
    [7, 8]
])

C = A @ B

print(C)

# [[19 22]
#  [43 50]]
```

Вычисление слоя нейронной сети

Пусть вход имеет два признака:

$$\mathbf{x} = \begin{bmatrix} 1 \\ 2 \end{bmatrix}.$$

Слой содержит три нейрона:

$$W = \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix}.$$

Тогда

$$\mathbf{z} = W\mathbf{x} = \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix} \begin{bmatrix} 1 \\ 2 \end{bmatrix} = \begin{bmatrix} 5 \\ 11 \\ 17 \end{bmatrix}.$$

```python
import numpy as np

x = np.array([1, 2])
W = np.array([
    [1, 2],
    [3, 4],
    [5, 6]
])
z = W @ x

print(z)   # [ 5 11 17 ]
```

С учётом смещения:

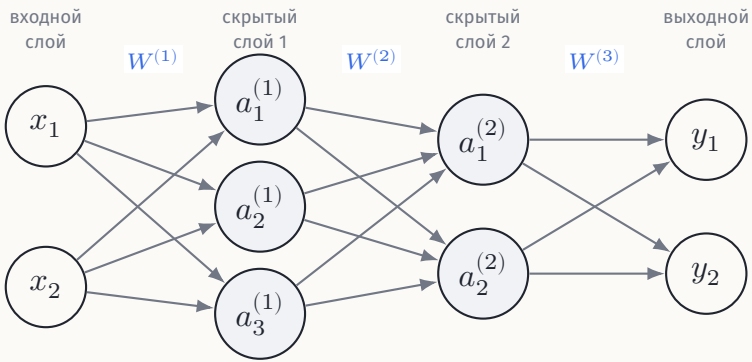
$$\mathbf{z} = W\mathbf{x} + \mathbf{b}$$

а после функции активации:

3

Многослойные нейронные сети

Структура многослойной нейронной сети



$a^{(0)} = \mathbf{x}$

$$\mathbf{z}^{(1)} = W^{(1)}\mathbf{a}^{(0)} + \mathbf{b}^{(1)} \quad \mathbf{a}^{(1)} = h^{(1)}(\mathbf{z}^{(1)})$$

$$\mathbf{z}^{(2)} = W^{(2)}\mathbf{a}^{(1)} + \mathbf{b}^{(2)} \quad \mathbf{a}^{(2)} = h^{(2)}(\mathbf{z}^{(2)})$$

$$\mathbf{z}^{(3)} = W^{(3)}\mathbf{a}^{(2)} + \mathbf{b}^{(3)} \quad \hat{\mathbf{y}} = \mathbf{a}^{(3)} = h^{(3)}(\mathbf{z}^{(3)})$$

Кружки обозначают выходы нейронов $a^{(l)}$, а переход между слоями задаётся матрицей весов $W^{(l)}$.

Прямое распространение через многослойную сеть

Один слой

Для первого скрытого слоя:

$$\mathbf{z}^{(1)} = W^{(1)}\mathbf{a}^{(0)} + \mathbf{b}^{(1)}$$

где $\mathbf{a}^{(0)} = \mathbf{x}$.
Например, для трёх нейронов:

$$\mathbf{z}^{(1)} = \begin{bmatrix} z_1^{(1)} \\ z_2^{(1)} \\ z_3^{(1)} \end{bmatrix}.$$

После этого применяется функция активации:

$$\mathbf{a}^{(1)} = h^{(1)}\left(\mathbf{z}^{(1)}\right)$$

В общем случае

Для любого слоя l:

$$\mathbf{z}^{(l)} = W^{(l)}\mathbf{a}^{(l-1)} + \mathbf{b}^{(l)}$$

$$\mathbf{a}^{(l)} = h^{(l)}\left(\mathbf{z}^{(l)}\right)$$

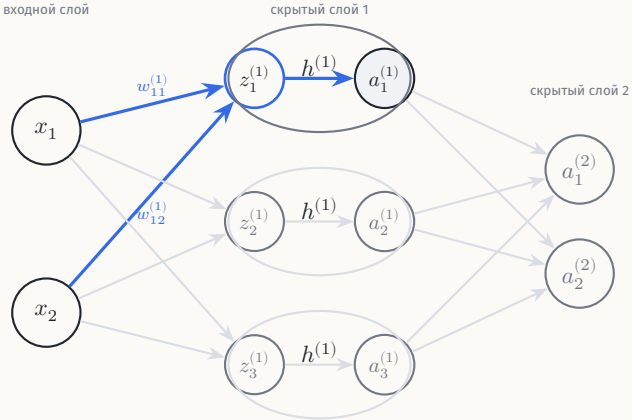
Последовательность вычислений

$$\mathbf{x} \to \mathbf{z}^{(1)} \to \mathbf{a}^{(1)} \to \mathbf{z}^{(2)} \to \mathbf{a}^{(2)} \to \cdots \to \hat{\mathbf{y}}$$

Каждый следующий слой получает **выход предыдущего слоя**.

Это и называется

Что происходит внутри скрытого слоя?



Каждый нейрон скрытого слоя сначала z, а затем преобразует его функцией активации в a.

Функция активации выходного слоя

Выбор функции активации в последнем слое зависит от типа задачи.

Регрессия

Прогнозируется вещественное число.

Обычно используется тождественная функция:

$$\boxed{\hat{y} = z}$$

Пример:

$$T = 325.4 \text{ К}.$$

Бинарная классификация

Два класса:

$$0 \quad \text{или} \quad 1.$$

Используется sigmoid:

$$\boxed{\hat{y} = \frac{1}{1 + e^{-z}}}$$

$$0 < \hat{y} < 1.$$

Многоклассовая классификация

Один из нескольких классов. Используется softmax:

$$\boxed{\hat{y}_k = \frac{e^{z_k}}{\sum_i e^{z_i}}}$$

причём

$$\sum_k \hat{y}_k = 1.$$

Структура скрытых слоёв может быть одинаковой, а выходной слой определяется задачей.

4

Обучение нейронных сетей

Что значит обучить нейронную сеть?

До сих пор мы вычисляли выход сети при уже известных параметрах:

$$\mathbf{x} \longrightarrow \text{нейронная сеть} \longrightarrow \hat{\mathbf{y}}$$

Но значения матриц $W^{(l)}$ и векторов $\mathbf{b}^{(l)}$ заранее неизвестны.

Цель обучения

Найти такие параметры сети, при которых предсказание $\hat{\mathbf{y}}$ как можно лучше соответствует правильному ответу $\mathbf{t}$.

$$(\mathbf{x}, \mathbf{t}) \to \hat{\mathbf{y}} \to L(\hat{\mathbf{y}}, \mathbf{t}) \to \text{изменение } W, \mathbf{b} \to \text{повторение}$$

- сеть делает предсказание;
- функция потерь измеряет ошибку;
- вычисляются градиенты;
- параметры изменяются так, чтобы уменьшить ошибку.

Обучение — это автоматический подбор параметров W и $\mathbf{b}$ на основе данных.

Функция потерь

Как понять, насколько хорошо сеть предсказывает правильный ответ?

Для этого используется **функция потерь** (loss function):

$$L(\hat{\mathbf{y}}, \mathbf{t})$$

Чем меньше L, тем ближе предсказание сети к правильному ответу.

Регрессия: MSE

Для прогноза численного значения:

$$L_{\text{MSE}} = \frac{1}{m} \sum_{i=1}^{m} (\hat{y}_i - t_i)^2$$

Большая ошибка между прогнозом и истинным значением даёт большую потерю.

Классификация: Cross-Entropy

Для многоклассовой классификации:

$$L_{\text{CE}} = -\sum_{k} t_k \ln \hat{y}_k$$

Если правильный класс задан one-hot вектором:

$$L_{\text{CE}} = -\ln p_{\text{true}}.$$

Функция потерь превращает качество предсказания в одно число, которое можно минимизировать.

Градиентный спуск и скорость обучения

Цель обучения:

$$\min_{W,\mathbf{b}} L$$

Градиент показывает направление **наиболее быстрого увеличения** функции потерь. Поэтому для уменьшения L параметры изменяют в противоположном направлении:

$$\theta \leftarrow \theta - \eta \nabla_\theta L$$

где θ обозначает все параметры сети.

Маленький η

Шаги очень маленькие:

$$\eta \ll 1.$$

Обучение обычно устойчивое, но может идти слишком медленно.

Большой η

Параметры изменяются слишком сильно. Можно перескочить минимум, получить колебания или расходимость.

Learning rate η определяет размер шага градиентного спуска.

Зачем нужен backpropagation?

Сначала сеть вычисляет предсказание и значение функции потерь:

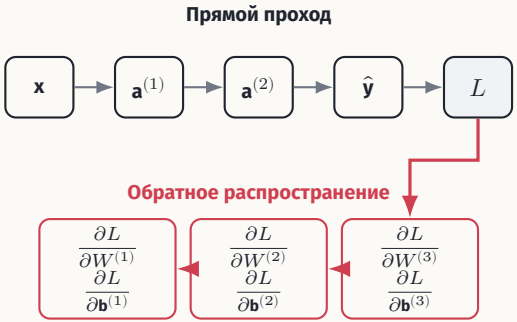
$$\mathbf{x} \to \hat{\mathbf{y}} \to L.$$

Но значение L только показывает, **насколько велика ошибка**.

Чтобы изменить параметры сети, необходимо узнать:

$$\frac{\partial L}{\partial W^{(l)}}, \qquad \frac{\partial L}{\partial \mathbf{b}^{(l)}}.$$

Эти градиенты вычисляются **от выходного слоя к входному** с помощью правила цепочки.



Forward: $\mathbf{x} \to L$ Backward: $L \to \nabla L$

Backpropagation на одном нейроне

Рассмотрим один нейрон:

$$z = wx + b, \qquad a = h(z), \qquad L = L(a).$$

Чтобы определить влияние веса w на L, используем **правило цепочки**:

$$\frac{\partial L}{\partial w} = \frac{\partial L}{\partial a}\frac{\partial a}{\partial z}\frac{\partial z}{\partial w}, \quad \frac{\partial L}{\partial b} = \frac{\partial L}{\partial a}\frac{\partial a}{\partial z}\frac{\partial z}{\partial b}$$

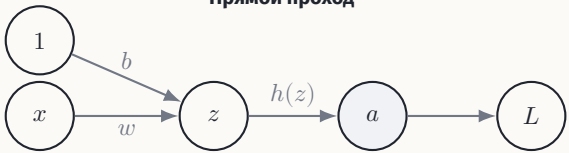
Так как

$$\frac{\partial a}{\partial z} = h'(z), \qquad \frac{\partial z}{\partial w} = x,$$

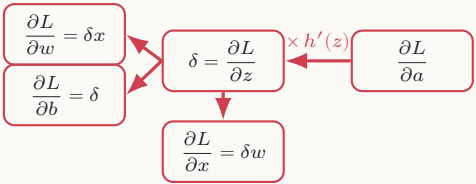
получаем

$$\frac{\partial L}{\partial w} = \frac{\partial L}{\partial a}\, h'(z)\, x, \quad \frac{\partial L}{\partial b} = \frac{\partial L}{\partial a}\, h'(z)$$

Прямой проход



Обратное распространение



5

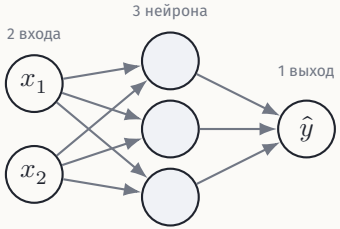
Практика и итоги

Практика: обучаем сеть решать XOR

Задача

Обучим нейронную сеть воспроизводить функцию XOR:

x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	0



Сеть сама подбирает веса и смещения и учится решать нелинейно разделимую задачу XOR.

Итоги лекции

От персептрона к сети

- один нейрон вычисляет

$$z = \mathbf{w}^T \mathbf{x} + b;$$

- функция активации формирует

$$a = h(z);$$

- несколько нейронов образуют слой;
- несколько слоёв образуют **многослойную нейронную сеть**.

Как сеть обучается

- forward pass вычисляет предсказание;
- функция потерь измеряет ошибку;
- backpropagation вычисляет градиенты;
- градиентный спуск обновляет W и **b**;
- процесс повторяется на mini-batch в течение нескольких эпох.

Нейронная сеть не получает готовые правила.
Она **настраивает свои параметры по данным**, последовательно уменьшая функцию потерь.