Data science

Лекция 7. Нейронные сети

2025/2026 учебный год

Доцент кафедры МО&МО, Махно В.В.

ИНСТИТУТ
МАТЕМАТИКИ
МЕХАНИКИ
КОМПЬЮТЕРНЫХ
НАУК
имени И.И. Воровича

Нейронные сети

Нейронная сеть — это **модель**, которая умеет **извлекать сложные зависимости** из данных.

Она вдохновлена устройством мозга и состоит из **искусственных нейронов**, соединённых между собой и обучающихся на примерах.

Нейросети умеют:

- распознавать изображения
- переводить текст
- генерировать музыку
- водить машины

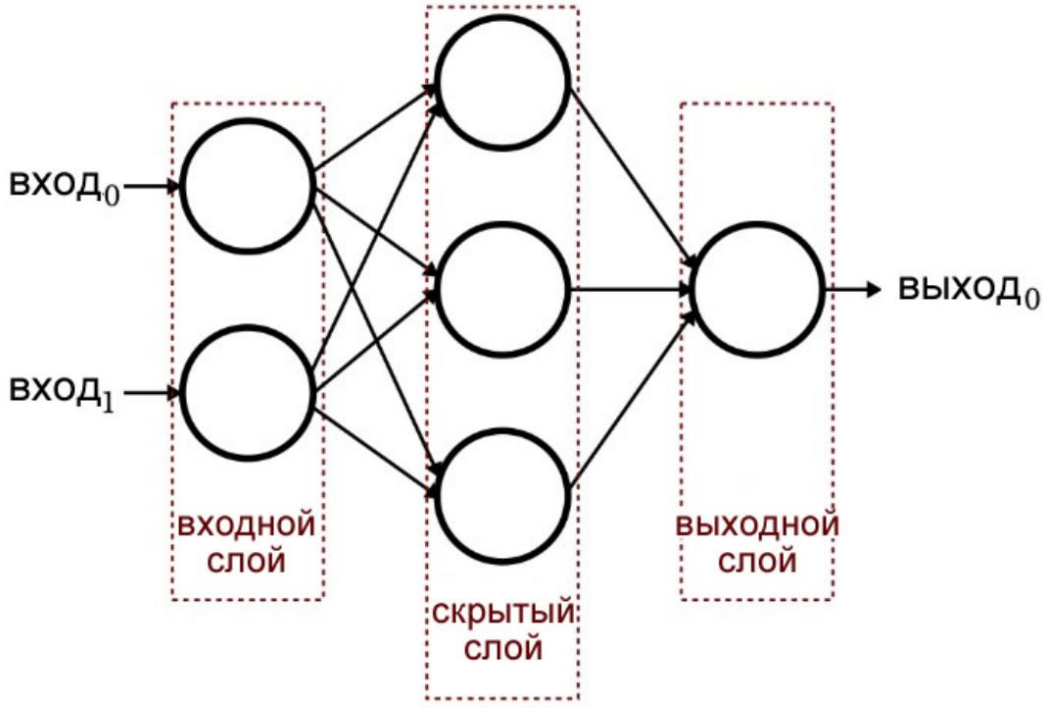
Как устроена нейросеть

Нейросеть — это **слои нейронов**, через которые проходят данные.

Входные данные → Скрытые слои → Выход

Каждый **нейрон**:

- получает на вход числа (признаки),
- умножает их на веса,
- применяет функцию активации,
- передаёт результат дальше.



Компоненты нейросети

Элемент	Что делает
Нейрон	Базовая вычислительная единица
Вес (weight)	Показывает "важность" признака
Функция активации	Добавляет нелинейность (ReLU, Sigmoid и т.д.)
Слои	Состоит из множества нейронов
Обратное распространение (backprop)	Механизм обучения сети

Обучение нейросети

1) Данные проходят через сеть → получаем предсказание
2) Считается **ошибка (loss)**
3) Ошибка **распространяется назад** по сети
4) Весы корректируются с помощью **градиентного спуска**
5) Повторяем много раз → сеть "обучается"

Зачем вообще нужны нейросети?

Нейросети — это **универсальные приближатели функций**.
Они **сами учат признаки**, в отличие от классических моделей, которым мы их явно задаём.

Пример

from tensorflow.keras.models import Sequential //Импортируется класс Sequential для создания модели нейронной сети

from tensorflow.keras.layers import Dense //Импортируется класс Dense для создания полносвязных (fully connected) слоев

```
model = Sequential([
    Dense(8, activation='relu', input_shape=(4,)),
    Dense(1, activation='sigmoid')
])
```

//Создается модель типа Sequential, где слои добавляются последовательно
//Первый слой: 8 нейронов, функция активации ReLU, входная размерность - 4 признака
//Второй слой: 1 нейрон, функция активации sigmoid (для бинарной классификации)

model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy']) //Настраивается процесс обучения: оптимизатор adam, функция потерь для бинарной классификации
Добавляется метрика accuracy для отслеживания точности во время обучения

model.fit(X_train, y_train, epochs=10, batch_size=32) //Запускается обучение на тренировочных данных X_train (признаки) и y_train (метки)

Обучение проходит 10 эпох (полных проходов по данным)
Размер батча - 32 образца за одну итерацию обновления весов
Это типичная архитектура для задачи бинарной классификации с небольшим количеством признаков на входе.

Недостатки:

Недостаток	Что это значит
Сложность	Трудно интерпретировать результат
Объём	Требуют много данных
Вычисления	Нужны GPU и много ресурсов
Риск переобучения	Без регуляризации могут "запомнить", а не обобщить

Backpropagation

Backpropagation (обратное распространение ошибки) — это алгоритм, который позволяет нейросети учиться, корректируя свои веса после каждой ошибки.

Нейросеть делает предсказание → сравнивает с правильным ответом → считает ошибку → и распространяет эту ошибку назад, чтобы обновить веса сети и сделать предсказание лучше в следующий раз.

Прямой проход (Forward Pass)

Данные проходят через сеть:
вход → скрытые слои → выход
Получаем предсказание $\hat{y}$.

Вычисление ошибки (Loss)

Сравниваем $\hat{y}$ с правильным y:
Например, используя **MSE** или **binary cross-entropy**.

$$\text{Loss} = \frac{1}{2}(y - \hat{y})^2$$

Обратный проход (Backpropagation)

Считаем, **как каждый вес влияет на ошибку**, используя **правила дифференцирования (цепное правило)**.

Как нужно изменить вес w, чтобы ошибка уменьшилась?

$$\frac{\partial \text{Loss}}{\partial w}$$

Обновление весов (Gradient Descent)

$w := w - \eta \cdot \frac{\partial \text{Loss}}{\partial w}$ **ный спуск**:

где:

w — вес

η (learning rate) — насколько сильно менять вес

Где используются нейросети

Нейросети — это фундамент современного искусственного интеллекта. Они лежат в основе технологий, которые мы используем каждый день: от голосовых помощников и автопилотов до генерации картин и общения с ChatGPT.

Нейросети — это не только научные статьи и роботы в лаборатории. Они **уже создают тренды в соцсетях**, управляют контентом, влияют на юмор, музыку, визуал и мемы.

Нейросеть	Что делает	Где используется
GPT (OpenAI)	Понимает и генерирует текст	ChatGPT, Copilot, Bing
BERT (Google)	Понимает смысл запроса	Google Поиск, NLP
DALL·E / MidJourney	Генерирует изображения по описанию	Арт, реклама, креатив
YOLO / Detectron2	Распознаёт объекты на фото	Безопасность, автопилоты
Stable Diffusion	Генерация изображений и стилей	Креатив, маркетинг
DeepSpeech / Whisper	Распознаёт голос	Перевод, диктовка
StyleGAN	Генерирует реалистичные лица	DeepFake, анимация

Виды нейросетей и решаемые задачи

Тип нейросети	Что делает / Где применяется	Примеры задач
MLP (полносвязная сеть)	Классическая нейросеть для простых табличных данных	Классификация, регрессия, предсказание спроса
CNN (сверточная сеть)	Обрабатывает изображения и видео	Распознавание лиц, объектов, диагностика по снимкам
RNN (рекуррентная)	Обрабатывает **последовательности**	Текст, временные ряды, прогноз погоды
LSTM / GRU	Улучшенные RNN, умеют "помнить" долгосрочные зависимости	Перевод текста, чат-боты, генерация текста
Transformer	Параллельно обрабатывает последовательности	ChatGPT, BERT, перевод, поиск, ответы на вопросы
GAN (генеративные сети)	Создают новые данные, похожие на обучающие	Генерация лиц, DeepFake, создание картин
AutoEncoder	Сжимают данные и восстанавливают обратно	Сжатие, поиск аномалий, удаление шума
VAE (вариационные автоэнкодеры)	Генеративная версия автоэнкодеров	Генерация изображений, данных
Siamese Network	Сравнивает объекты и учит "похожесть"	Сопоставление лиц, рекомендация похожих товаров
Capsule Network	Учитывает иерархию и положение объектов	Распознавание сложных образов на изображениях

CNN (Convolutional Neural Network)

Обрабатывает **изображения**:

- Учится распознавать **паттерны на изображении** (углы, линии, формы)

Используется в: медицина, видеоаналитика, фильтрация контента

Примеры:

- Распознавание номеров на фото
- Диагностика рака по снимку
- Автоматическая разметка дорог в автопилотах

RNN / LSTM / GRU

Работают с **последовательными данными** — где важен порядок:

- Текст
- Временные ряды
- Музыка, речь

Примеры:

- Перевод текста
- Генерация заголовков
- Прогноз курса валют

Transformers

Новейший стандарт: работает лучше RNN и обучается быстрее.
Главные модели: **BERT**, **GPT**, **T5**, **XLNet**.

Примеры:

- ChatGPT
- Поиск по смыслу
- Генерация кода / текста
- Q&A системы

GAN (Generative Adversarial Network)

Две нейросети соревнуются:
одна **генерирует**,
другая **оценивает**.
Модель учится создавать **новые данные, неотличимые от настоящих**.

Примеры:

- Генерация лиц (thispersondoesnotexist.com)
- DeepFake
- Стиль "Моне" на обычной фотографии
- Реконструкция фото из шума

AutoEncoder

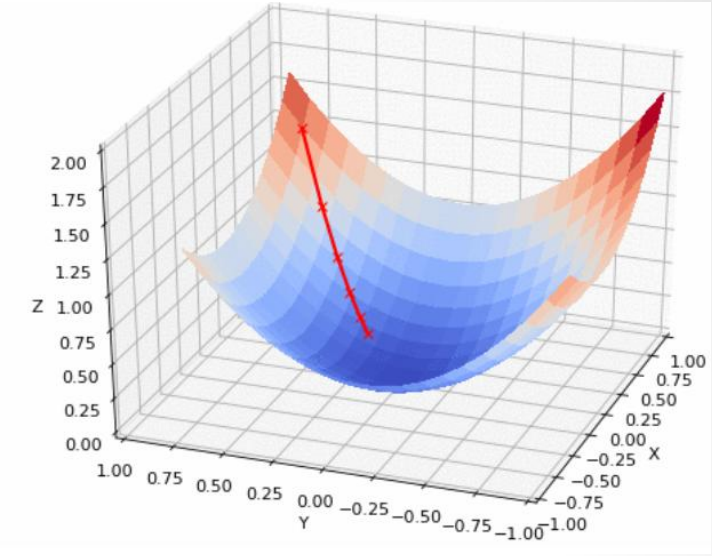
Сжимает данные → восстанавливает обратно
Используется, когда нужно:

- Удалить шум
- Найти выбросы
- Сделать компактное представление данных

Примеры:

- Удаление шума на фото
- Детектирование аномалий в финансах
- Предобработка изображений

Градиентный спуск



Градиентный спуск (Gradient Descent) — это **итеративный алгоритм оптимизации**, с помощью которого можно **минимизировать функцию потерь**. Это фундамент почти всех алгоритмов обучения: от простой линейной регрессии до глубоких нейросетей.

В ML мы почти всегда хотим **найти такие параметры модели**, которые минимизируют ошибку. Градиентный спуск — способ найти эти параметры.

Спуск с холма

Представьте, что Вы стоите на вершине горы (график ошибки) в тумане и хотите добраться в самую низкую точку — минимум функции потерь.

- Вы не видите карту, но можете **ощущать наклон**.
- Вы делаете **шаг в сторону уменьшения наклона** — вниз.
- И так **постепенно приближаетесь к минимуму**.

Это и есть градиентный спуск. На каждом шаге мы идём в сторону, где **ошибка убывает быстрее всего**.

Шаг по градиенту

Пусть есть функция потерь:

L(w)=ошибка модели от параметров w

Цель — найти такое w, при котором L(w) минимальна.

Обновление параметров с помощью градиентного спуска:

$$w_{\text{new}} = w_{\text{old}} - \eta \cdot \frac{\partial L}{\partial w}$$

Где:

- w_{old} – текущее значение параметра
- w_{new} – новое значение после шага
- η– скорость обучения (learning rate): **насколько большой шаг мы делаем**
- $\frac{\partial L}{\partial w}$ – **градиент функции потерь по параметру** `w` (направление наибольшего роста функции потерь)

Идём **в противоположную сторону от градиента**, потому что хотим **уменьшать ошибку**, а не увеличивать её.

Градиентный спуск в линейной регрессии

🎯 Цель

Обучить модель линейной регрессии:

$$\hat{y} = w \cdot x + b$$

Найти такие значения `w` и `b`, при которых **ошибка между предсказаниями и настоящими значениями минимальна**.

🧮 Шаг 1: Целевая функция (ошибка)

Мы используем **среднеквадратичную ошибку (MSE)** как функцию потерь:

$$L(w, b) = \frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 = \frac{1}{n} \sum_{i=1}^{n} (y_i - (w \cdot x_i + b))^2$$

🧠 Шаг 2: Градиенты — как понять, в какую сторону менять `w` и `b`

Чтобы минимизировать функцию потерь, считаем её производные (градиенты):

$$\frac{\partial L}{\partial w} = -\frac{2}{n} \sum x_i \cdot (y_i - \hat{y}_i)$$

$$\frac{\partial L}{\partial b} = -\frac{2}{n} \sum (y_i - \hat{y}_i)$$

Эти градиенты говорят нам: **насколько сильно и в какую сторону нужно изменить параметры `w` и `b`**.

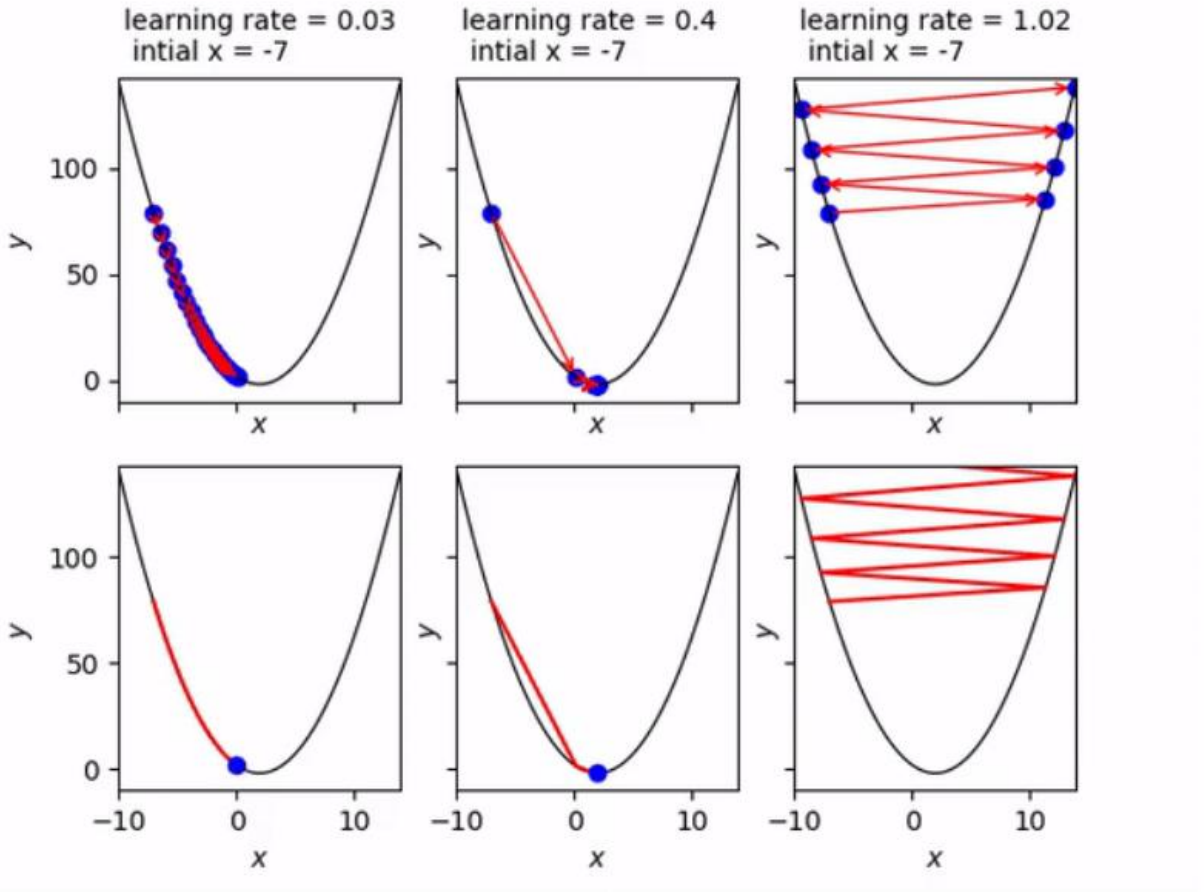
🔁 Шаг 3: Обновление параметров

$$w_{\text{new}} = w_{\text{old}} - \eta \cdot \frac{\partial L}{\partial w}$$

$$b_{\text{new}} = b_{\text{old}} - \eta \cdot \frac{\partial L}{\partial b}$$

Скорость обучения (learning rate)

- Слишком большой η : перескакиваем минимум, модель не обучается (или «взорвётся»).
- Слишком маленький η : обучение идёт очень медленно.
- Иногда η адаптируется во времени или по направлению (см. Adam ниже).



Градиентный спуск на python

```
import numpy as np

# Данные
X = np.array([1, 2, 3, 4, 5], dtype=float)
y = np.array([3, 5, 7, 9, 11], dtype=float)  # истинная зависимость: y = 2x + 1

# Начальные параметры
w = 0.0
b = 0.0
lr = 0.01  # скорость обучения

# Обучение
for i in range(1000):
    y_pred = w * X + b
    error = y - y_pred

    dw = -2 * np.mean(X * error)
    db = -2 * np.mean(error)

    w -= lr * dw
    b -= lr * db

# Результат
print(f"w = {w:.2f}, b = {b:.2f}")  # Должно быть близко к 2 и 1
```

Благодарю за внимание!