



Московский государственный
технический университет
имени Н.Э. Баумана



Кафедра ИУ5
«Системы обработки информации
и управления»

Адаптированный курс для ГУИМЦ по глубокому
обучению

Введение в обучение с учителем

Канев Антон Игоревич
преподаватель кафедры ИУ5

aikanev@bmstu.ru

Канев Антон Игоревич



Окончил с отличием МГТУ
им. Н.Э. Баумана в 2016 г.

Аспирантуру МГТУ им. Н.Э.
Баумана в 2020 г.

С отличием окончил
университет Glyndwr
(Рексем, Великобритания)



Курс по глубокому
обучению

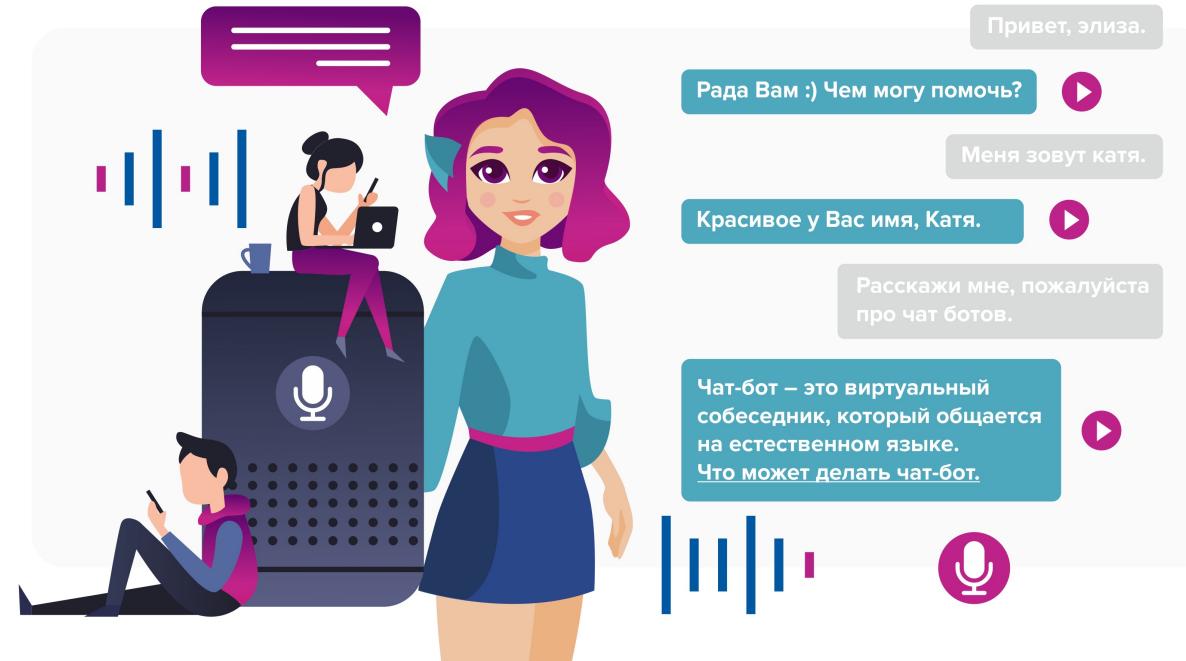
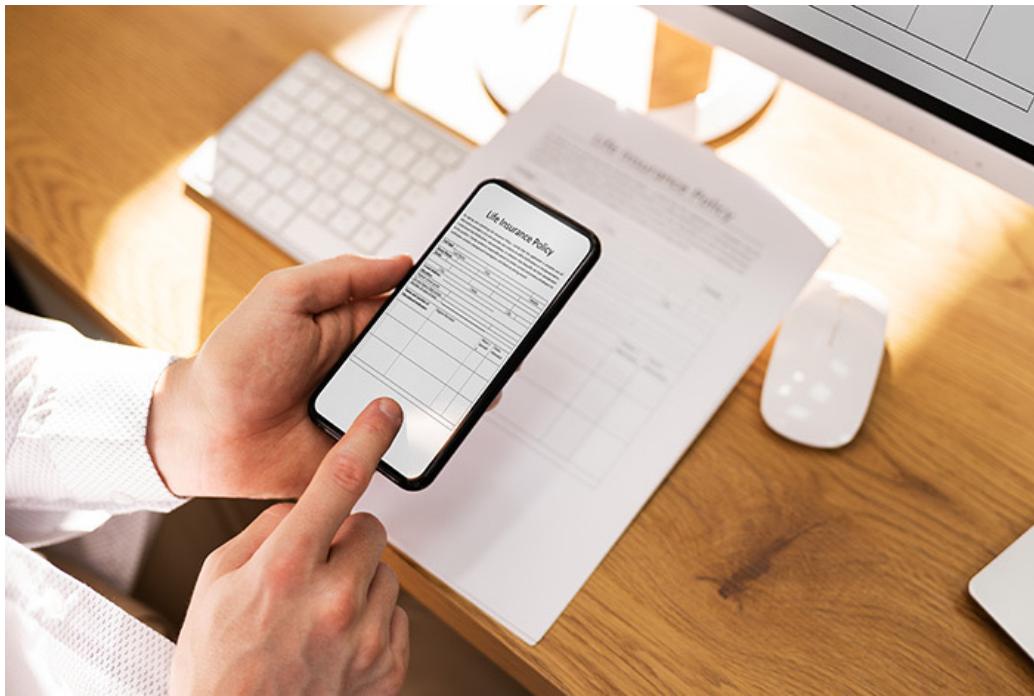
Курс по web разработке



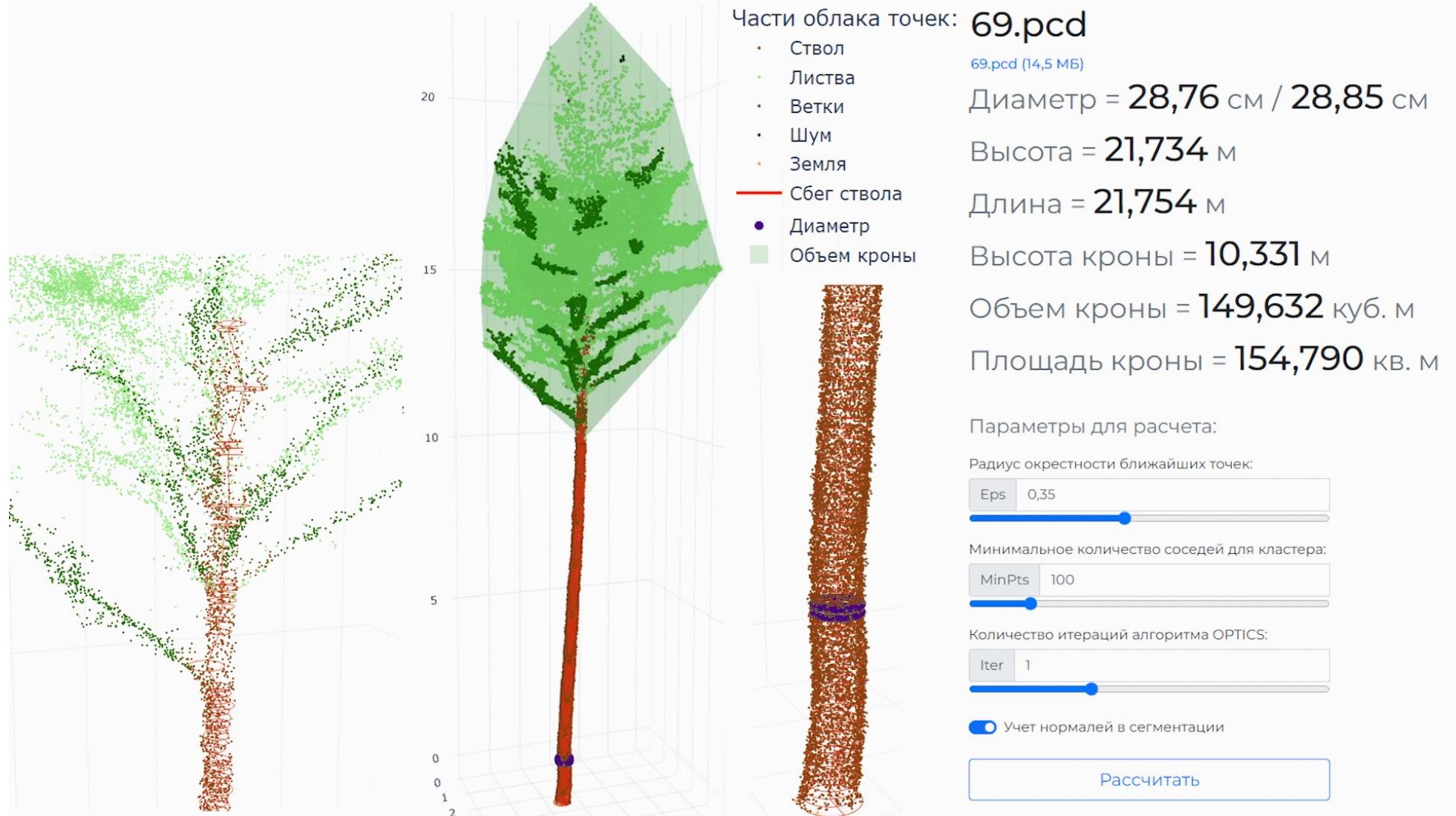
NVIDIA DLI Certificate

Обработка речи и текста

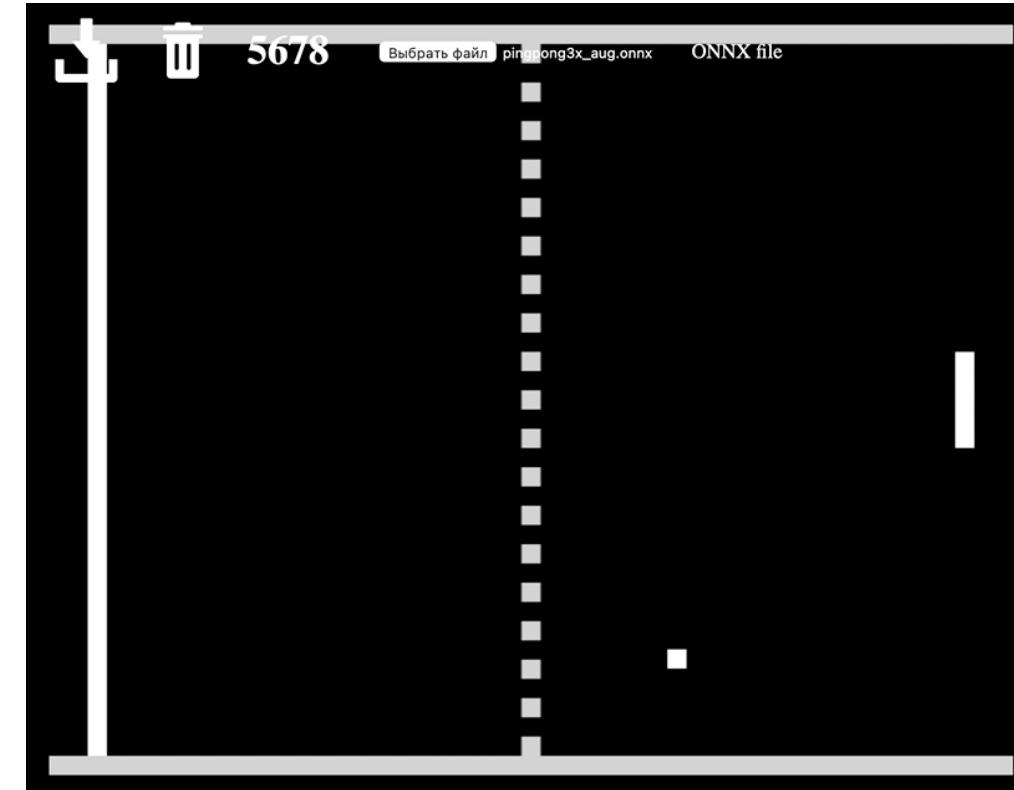
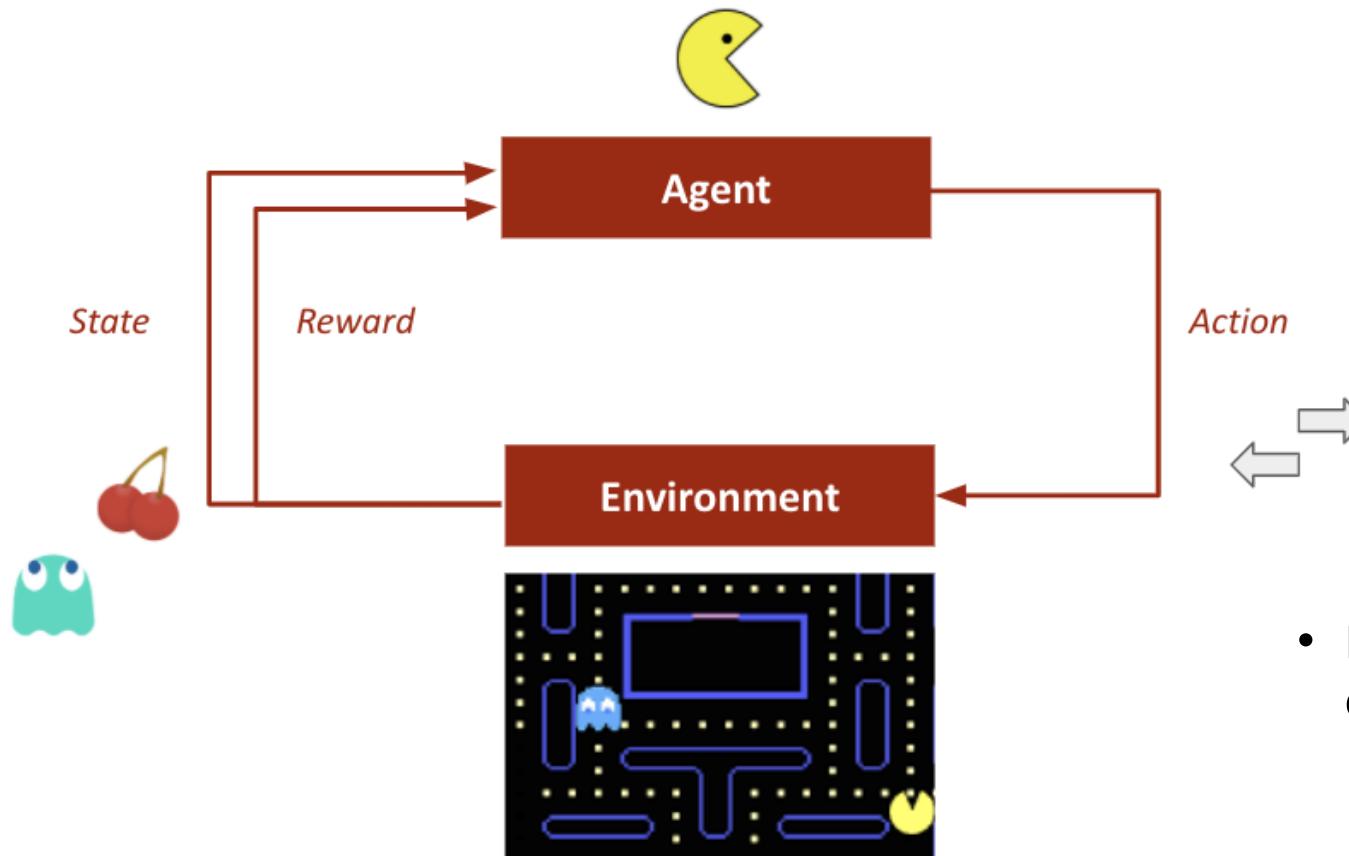
- Автоматическое распознавание речи
- Распознавание сканов - OCR



Lidar



Компьютерные игры



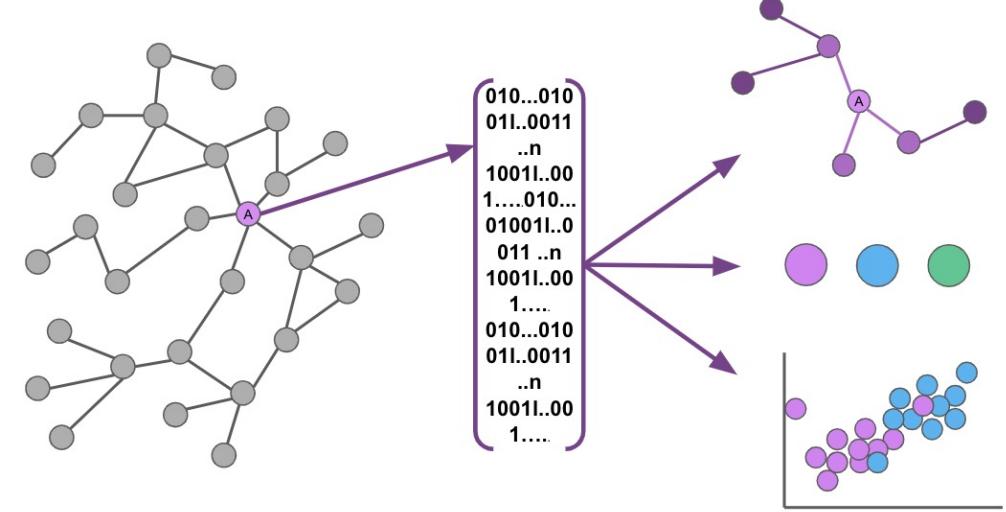
- Пример интеллектуального агента ping-pong обученного на кафедре

<https://github.com/iu5git/ai-bot-games-in-js>

- Обучение с подкреплением или по данным игры пользователя

Вычисления в графах

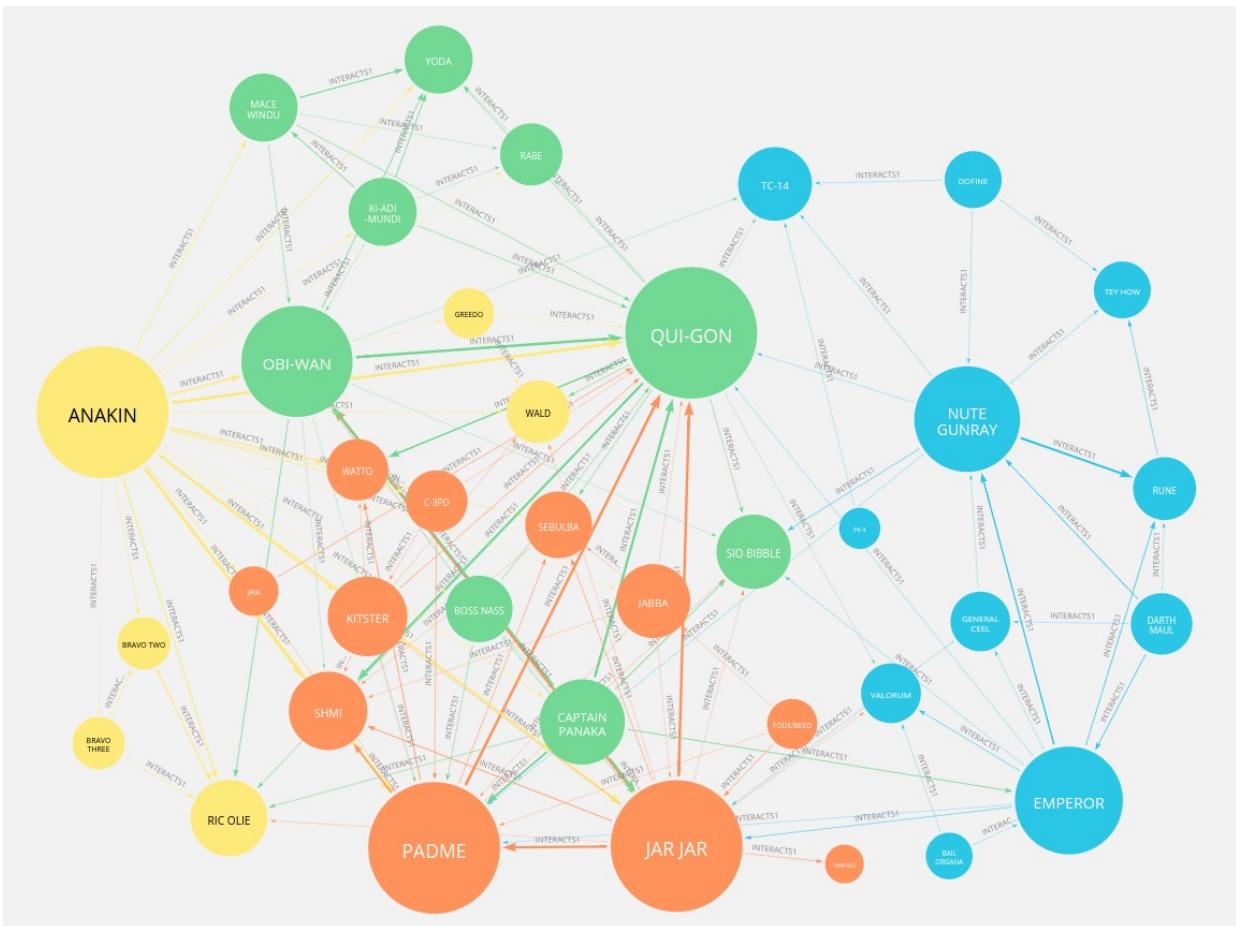
- Графовая БД Neo4j
- GraphML



Neighborhood
Prediction

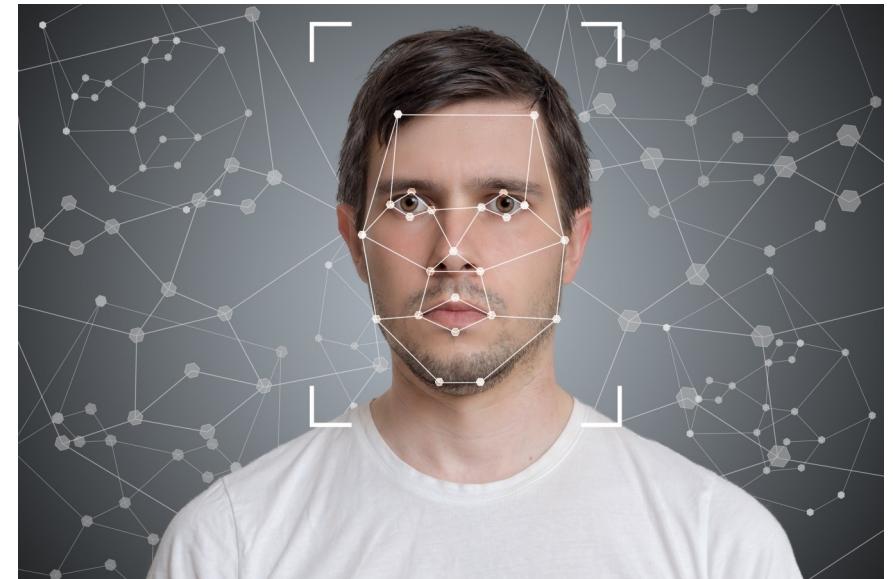
Classification

Visualization



Распознавание лиц/угроз на изображении

- Различные методы распознавания лиц

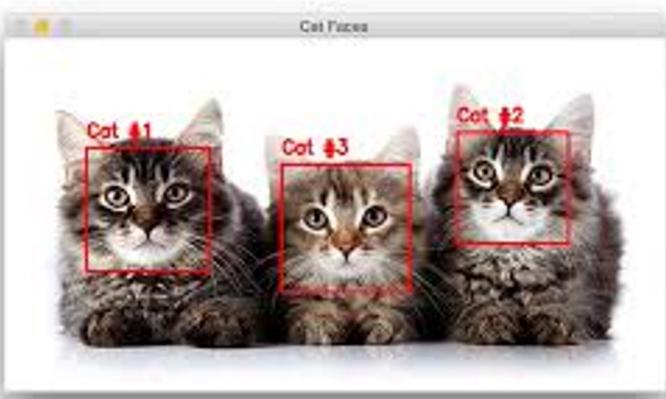


Распознавание и локализация yolo:

- Оружия
- Масок и тд

Машинное обучение

- Машинное обучение – это возможность обучать компьютер без детального программирования
- Для обучения компьютера используются наборы примеров для решения задач, которые сложно представить в программном коде



- Распознавание рукописных символов
 - Преобразование рукописных букв в цифровые буквы
- Перевод языка
 - Перевод произнесенного или написанного текста на другой язык (пример - Google Translate)
- Распознавание голоса
 - Преобразование голоса в текст (пример - Siri, Cortana, и Alexa)
- Классификация изображений
 - Разметка фотографии в соответствии с категорией (пример - Google Photos)
- Беспилотное вождение
 - Обучение машин вождению

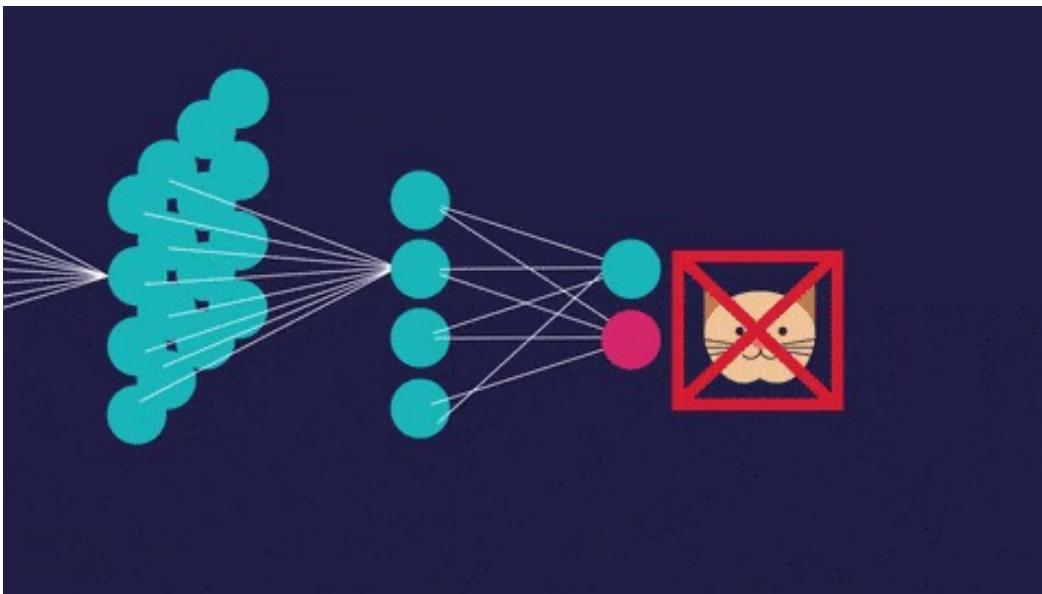
First Name	L	O	R	I								
Last Name	W	A	L	T	E	R	S					

Типы машинного обучения

- Обучение с учителем
 - Обучающие данные размечены
 - Задачей является правильно определить класс объекта
- Обучение с подкреплением
 - Обучающие данные не размечены
 - Система получает обратную связь от своих действий
 - Задачей является выбор правильных действий
- Обучение без учителя
 - Обучающие данные не размечены
 - Задачей является правильно определить категорию
- Алгоритмы обучения с учителем
 - Линейная регрессия
 - Дерево решений
 - Машина опорных векторов (SVM)
 - К-ближайших соседей
 - Нейронные сети

Обучение нейросети

ПРОЦЕСС ОБУЧЕНИЯ НЕЙРОСЕТИ



- Процедуры для обучения нейронных сетей
 - Произвести инференс на обучающем наборе
 - Вычислить ошибку между спрогнозированными значениями и истинными значениями на обучающем наборе
 - Определите вклад каждого нейрона в ошибку
 - Изменить веса нейронной сети для минимизации ошибки

Фреймворки обучения с учителем

 PyTorch



TensorFlow

 Keras

Метрики для классификации

Прогноз:



Картина:



**Истинно
Положи-
тельное
(TP)**

**Истинно
Отрицат-
ельное
(TN)**

**Ложно
Отрицат-
ельное
(FP)**

**Ложно
Положи-
тельное
(FN)**

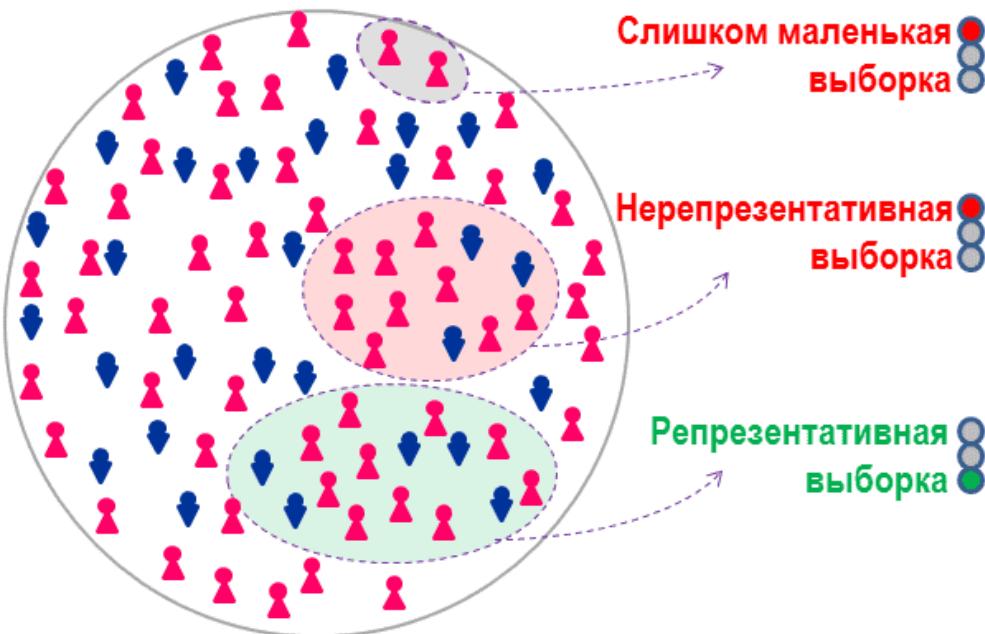
- Истинно положительные (True Positive, TP): Правильно определенная как соответствующая
- Истинно отрицательные (True Negative, TN): Правильно определенная как не соответствующая
- Ложно положительные (False Positive, FP): Неправильно определенная как соответствующая
- Ложно отрицательные (False Negative, FN): Неправильно определенная как не соответствующая

Precision, Recall, и Accuracy

- Точность (Precision)
 - Процент положительных меток которые правильно определены
 - $\text{Precision} = (\# \text{ true positives}) / (\# \text{ true positives} + \# \text{ false positives})$
- Полнота (Recall)
 - Процент положительных примеров которые были правильно определены
 - $\text{Recall} = (\# \text{ true positives}) / (\# \text{ true positives} + \# \text{ false negatives})$
- Accuracy
 - Процент положительных меток
 - $\text{Accuracy} = (\# \text{ true positives} + \# \text{ true negatives}) / (\# \text{ of samples})$

Тестовая и обучающая выборки

Генеральная совокупность включает ♂ - 1/3 и ♀ - 2/3



- Входные данные и метки
- Тестовая и обучающая части набора данных

```
with open('cifar-100-python/train', 'rb') as f:  
    data_train = pickle.load(f, encoding='latin1')  
with open('cifar-100-python/test', 'rb') as f:  
    data_test = pickle.load(f, encoding='latin1')
```

Обучающая
выборка

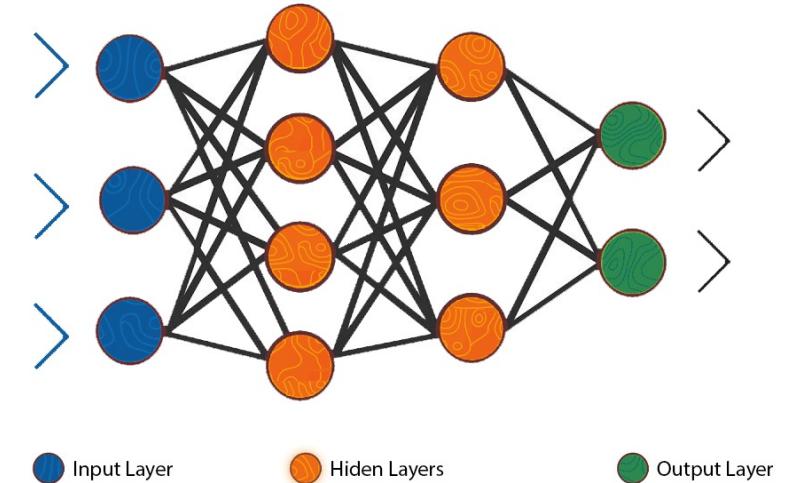
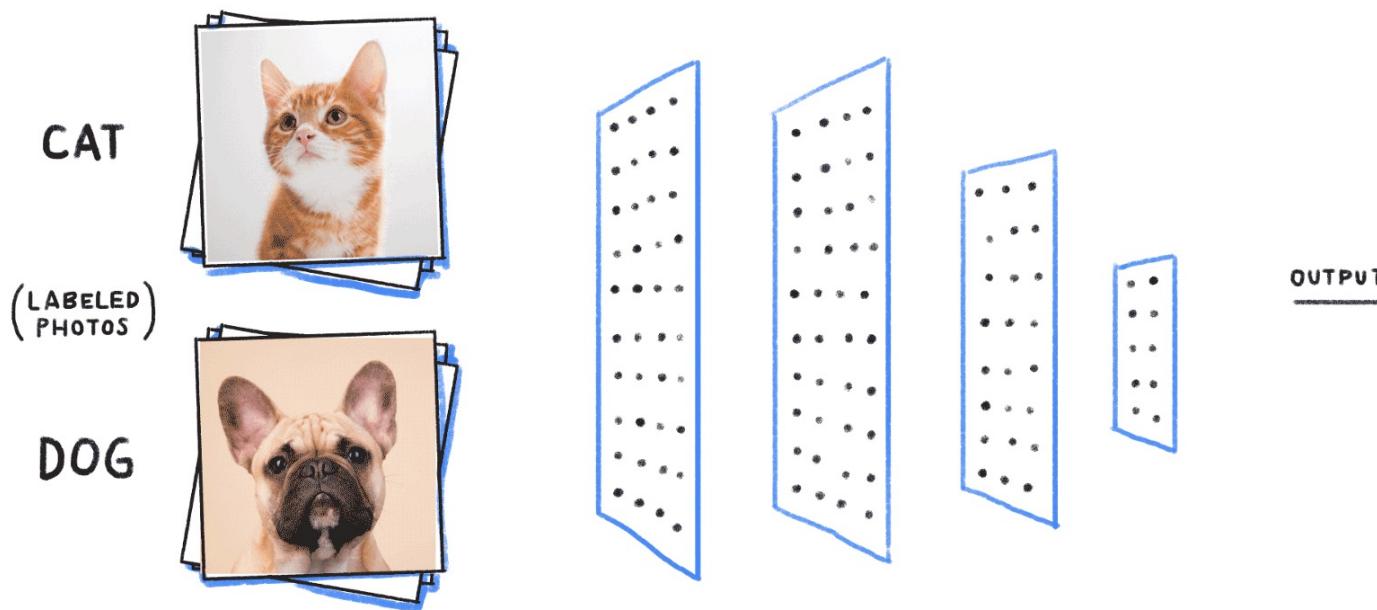
0	0 0 0 0 0 0 0 0 0 0
1	1 1 1 1 1 1 1 1 1 1
2	2 2 2 2 2 2 2 2 2 2
3	3 3 3 3 3 3 3 3 3 3
4	4 4 4 4 4 4 4 4 4 4
5	5 5 5 5 5 5 5 5 5 5
6	6 6 6 6 6 6 6 6 6 6
7	7 7 7 7 7 7 7 7 7 7
8	8 8 8 8 8 8 8 8 8 8
9	9 9 9 9 9 9 9 9 9 9

Тестовая
выборка

0	0 0 0 0 0 0
1	1 1 1 1 1 1
2	2 2 2 2 2 2
3	3 3 3 3 3 3
4	4 4 4 4 4 4
5	5 5 5 5 5 5
6	6 6 6 6 6 6
7	7 7 7 7 7 7
8	8 8 8 8 8 8
9	9 9 9 9 9 9

Нейронная сеть

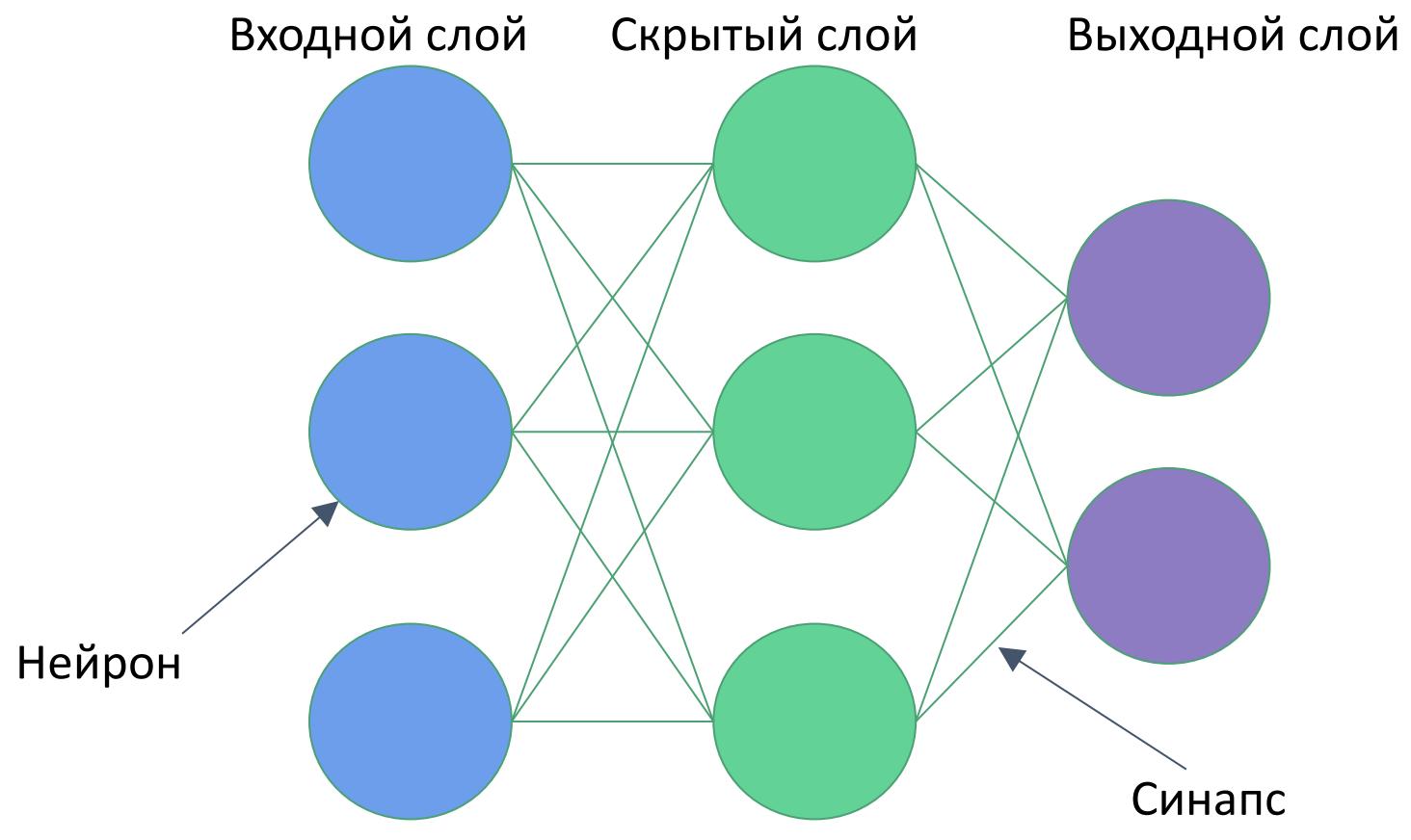
- Сеть состоит из слоев нейронов
- Каждый нейрон – сумматор
- Обучение – вычисление весов w нейрона



Архитектура нейронной сети

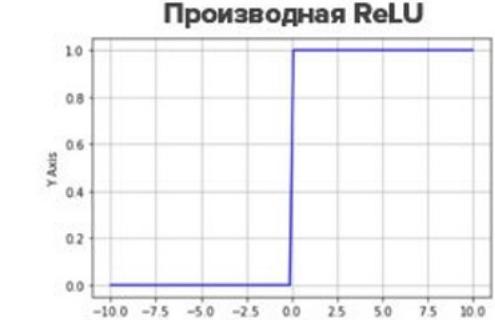
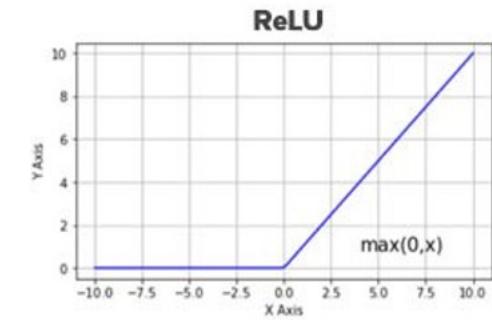
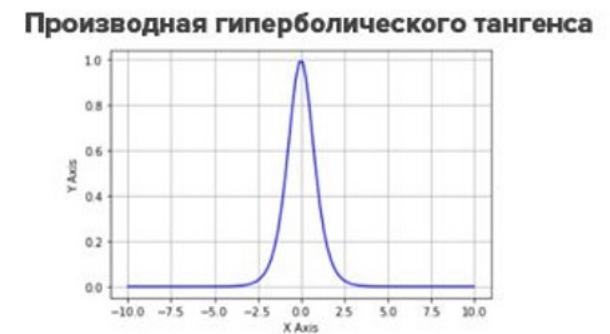
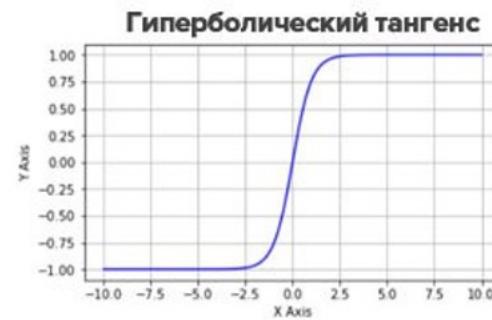
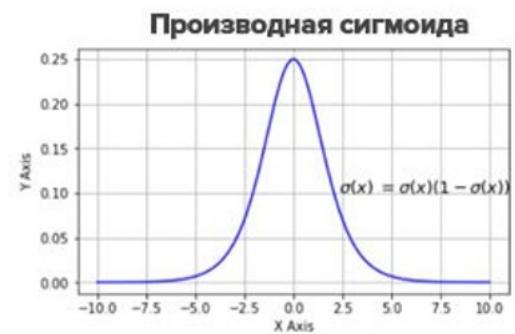
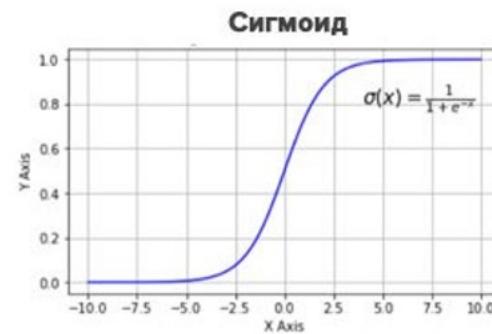
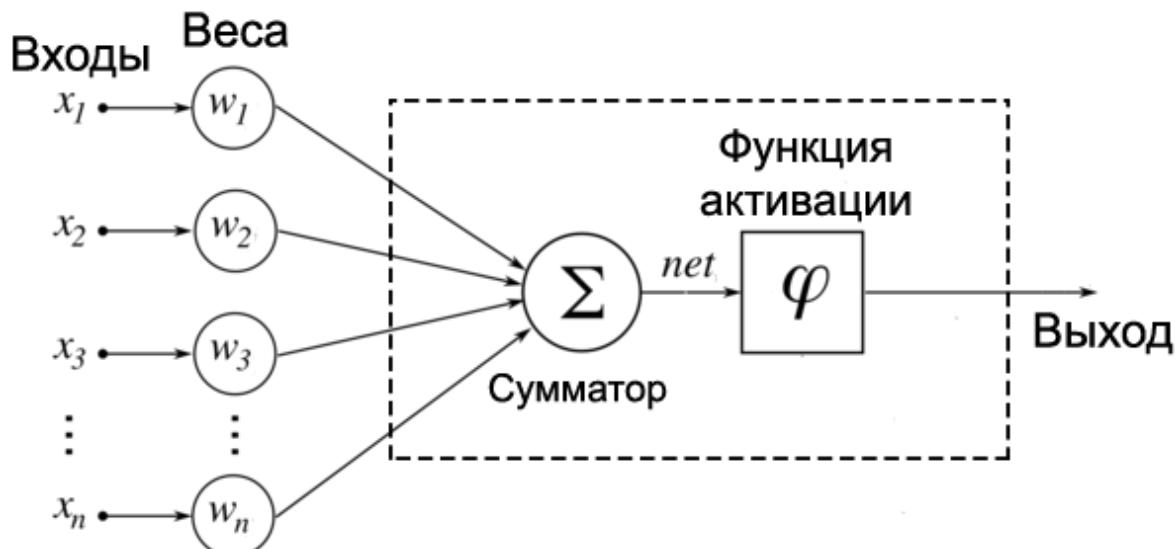
- Хорошая классификация при малом объеме входных данных
- Очень много связей при большом объеме данных

```
self.seq = nn.Sequential(  
    nn.Linear(32*32*3, hidden_size),  
    nn.ReLU(),  
    # активационная функция  
    nn.Linear(hidden_size, classes), )
```

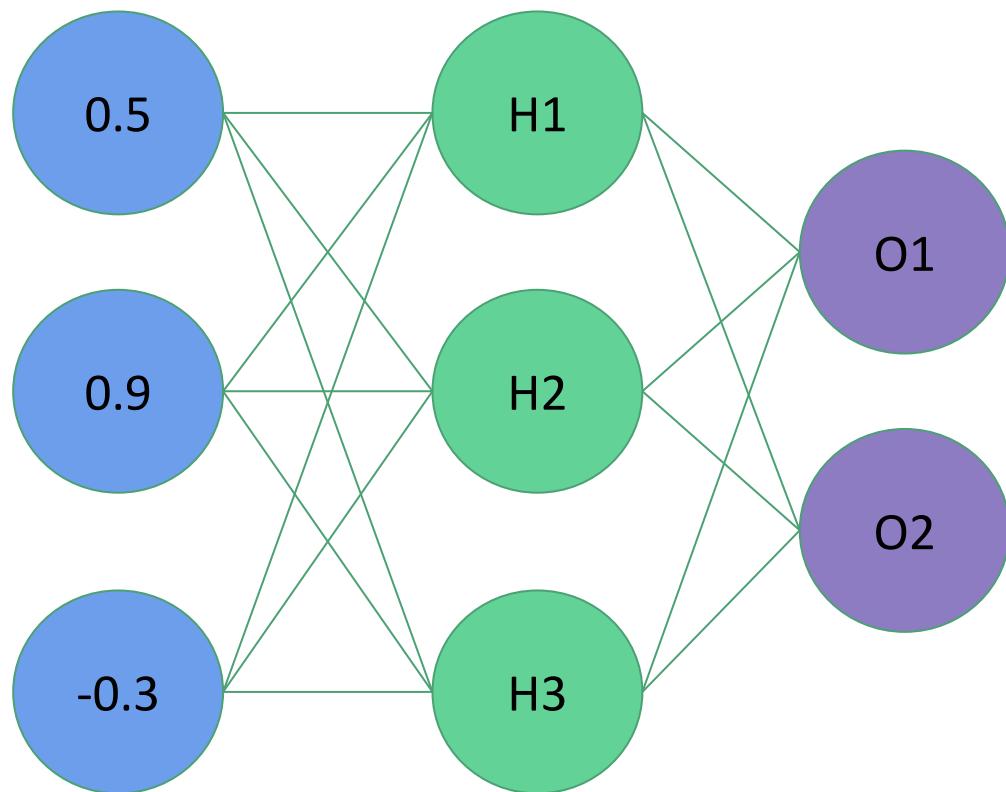


Устройство нейрона

- Помимо сумматора есть активационная функция
- Они разные для разных задач



Вычисление прогноза (инференс)



Веса H1 = (1.0, -2.0, 2.0)

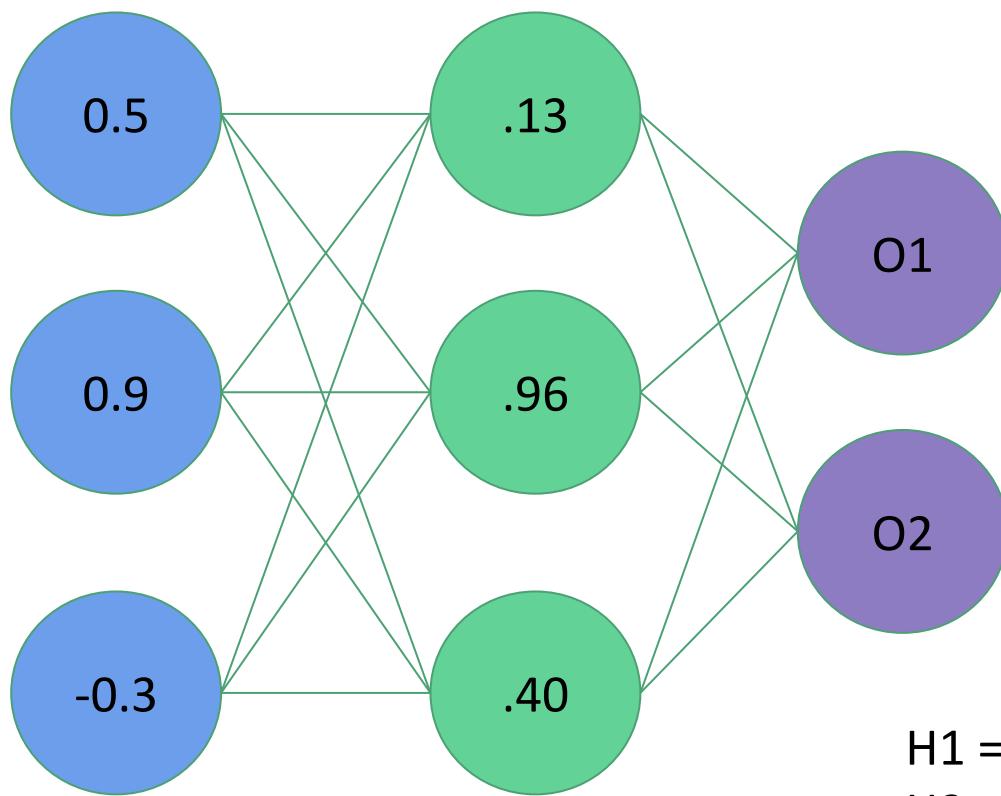
Веса H2 = (2.0, 1.0, -4.0)

Веса H3 = (1.0, -1.0, 0.0)

Веса O1 = (-3.0, 1.0, -3.0)

Веса O2 = (0.0, 1.0, 2.0)

Вычисление прогноза (инференс)



$$\text{Веса } H1 = (1.0, -2.0, 2.0)$$

$$\text{Веса } H2 = (2.0, 1.0, -4.0)$$

$$\text{Веса } H3 = (1.0, -1.0, 0.0)$$

$$\text{Веса } O1 = (-3.0, 1.0, -3.0)$$

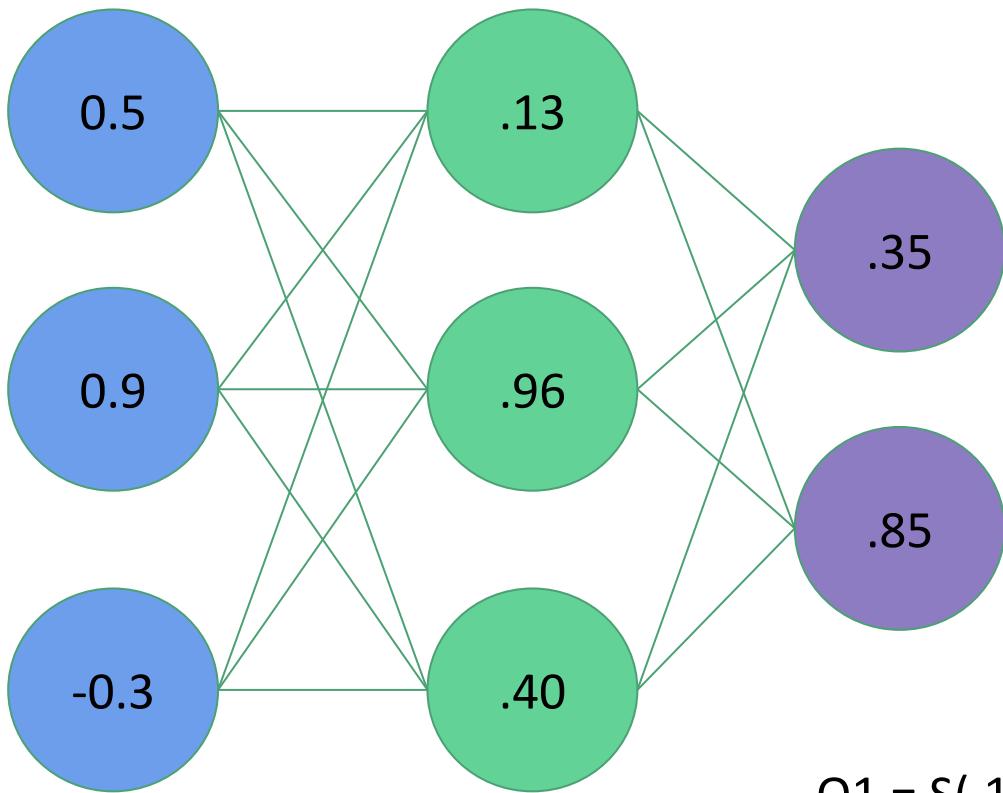
$$\text{Веса } O2 = (0.0, 1.0, 2.0)$$

$$H1 = S(0.5 * 1.0 + 0.9 * -2.0 + -0.3 * 2.0) = S(-1.9) = .13$$

$$H2 = S(0.5 * 2.0 + 0.9 * 1.0 + -0.3 * -4.0) = S(3.1) = .96$$

$$H3 = S(0.5 * 1.0 + 0.9 * -1.0 + -0.3 * 0.0) = S(-0.4) = .40$$

Вычисление прогноза (инференс)



$$\text{Веса } H1 = (1.0, -2.0, 2.0)$$

$$\text{Веса } H2 = (2.0, 1.0, -4.0)$$

$$\text{Веса } H3 = (1.0, -1.0, 0.0)$$

$$\text{Веса } O1 = (-3.0, 1.0, -3.0)$$

$$\text{Веса } O2 = (0.0, 1.0, 2.0)$$

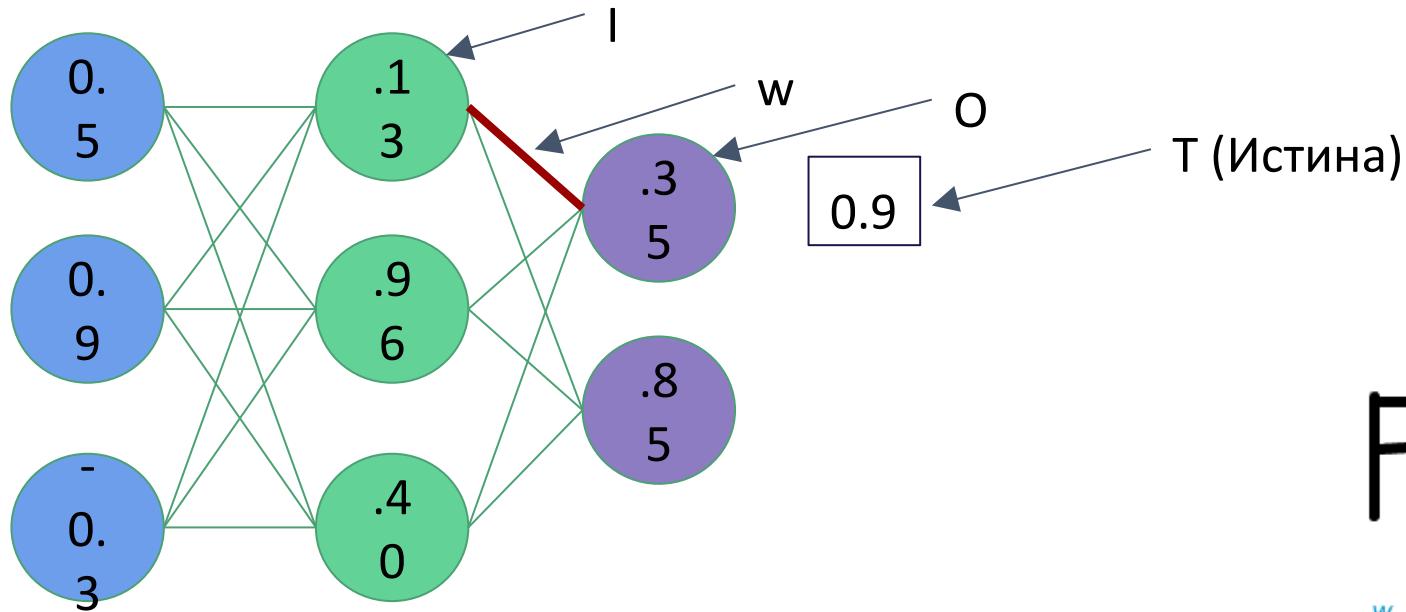
$$O1 = S(0.13 * -3.0 + 0.96 * 1.0 + 0.40 * -3.0) = S(-0.63) = 0.35$$

$$O2 = S(0.13 * 0.0 + 0.96 * 1.0 + 0.40 * 2.0) = S(1.76) = 0.85$$

Метод градиентного спуска

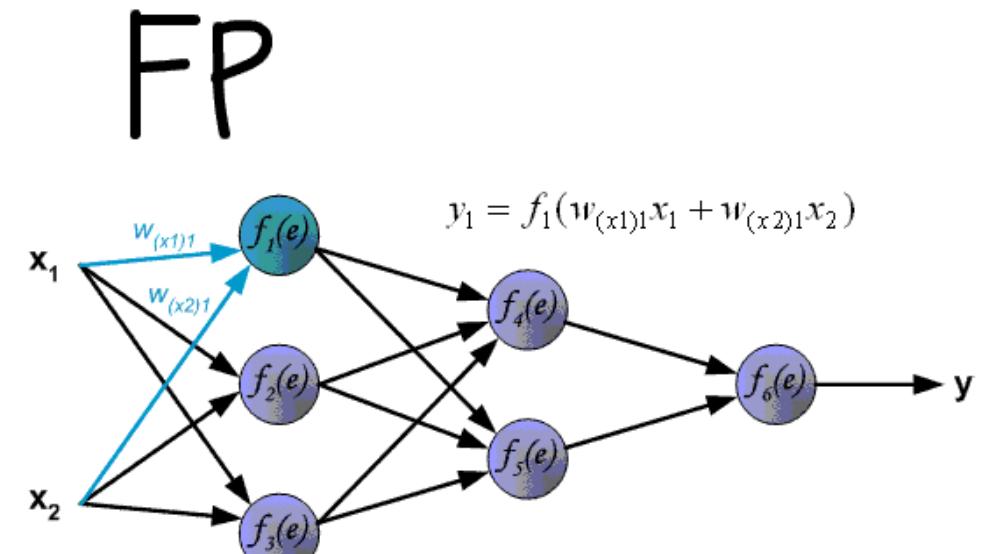
- Вклад в размер ошибки вычисляется с помощью обратного распространения (Backpropagation)
- Минимизация ошибки достигается благодаря градиентному спуску (Gradient Descent)
- Метод градиентного спуска минимизирует ошибку нейронной сети
 - На каждой итерации ошибка сети рассчитывается на основе обучающих данных
 - Затем модифицируются веса для уменьшения ошибки
- Метод градиентного спуска останавливается когда:
 - Ошибка достаточно мала
 - Максимальное количество итераций превышено
- Вопрос: какие веса должны быть обновлены и на сколько?
 - Подсказка: используйте производную от ошибки по отношению к весу, чтобы найти объем «вины»

Пример обратного распространения



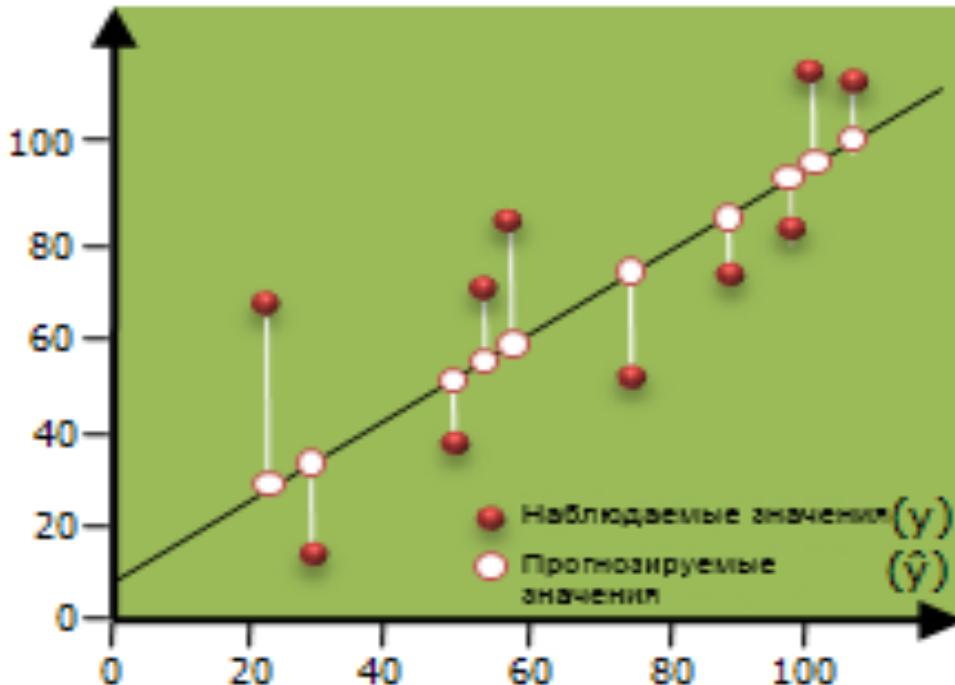
$$\frac{\partial E}{\partial w} = I \cdot (O - T) \cdot O \cdot (1 - O)$$

$$\frac{\partial E}{\partial w} = .13 \cdot (.35 - .9) \cdot .35 \cdot (1 - .35)$$

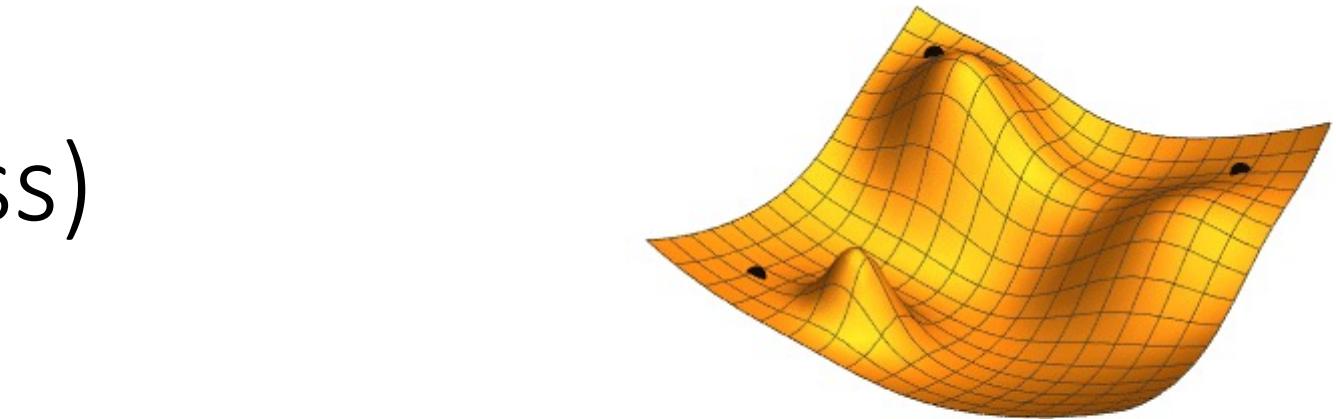


ФУНКЦИЯ ПОТЕРЬ (loss)

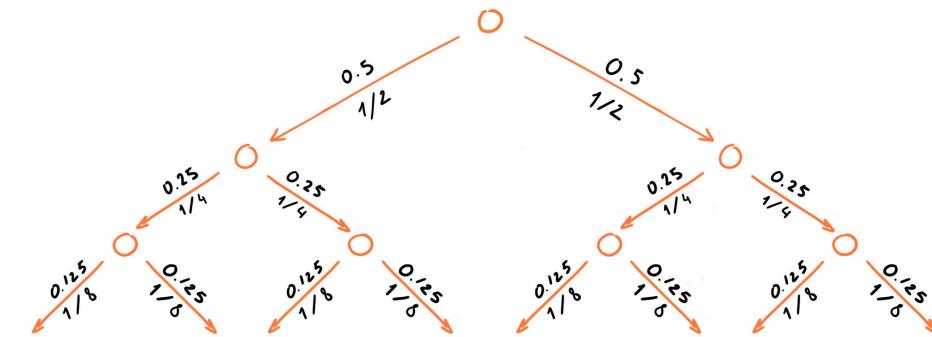
- Нам нужен минимум функции потерь
- Метод наименьших квадратов (снизу)
- Кроссэнтропия (справа)



$$H_p(q) = -\frac{1}{N} \sum_{i=1}^N y_i \cdot \log(p(y_i)) + (1 - y_i) \cdot \log(1 - p(y_i))$$



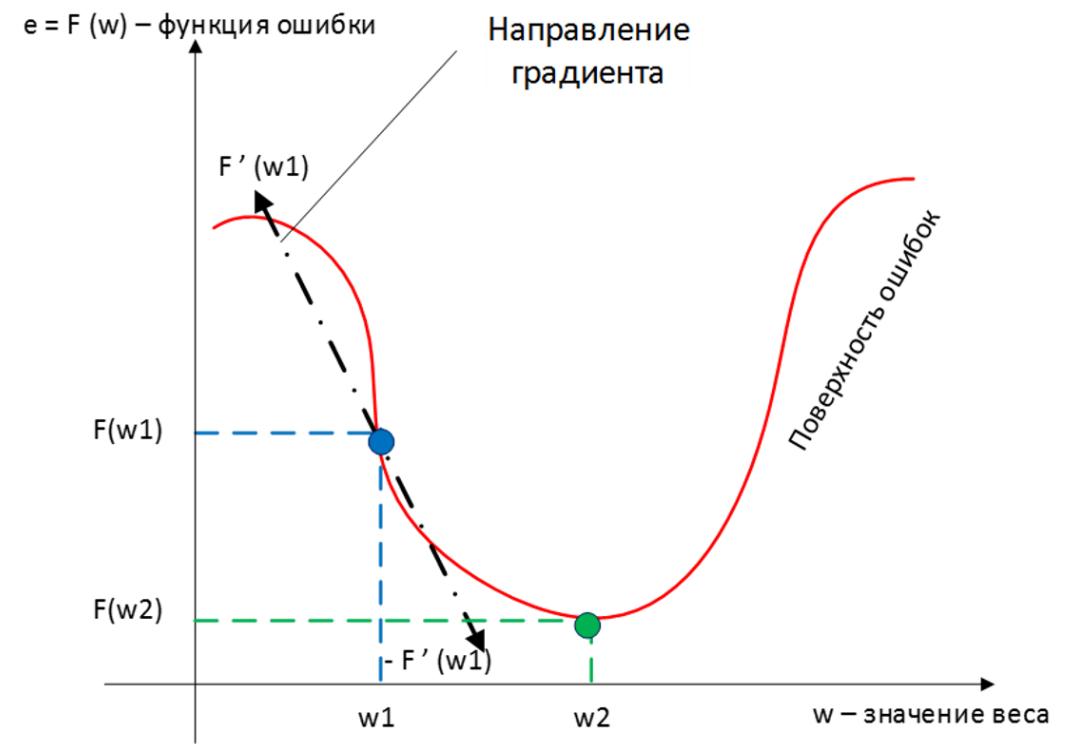
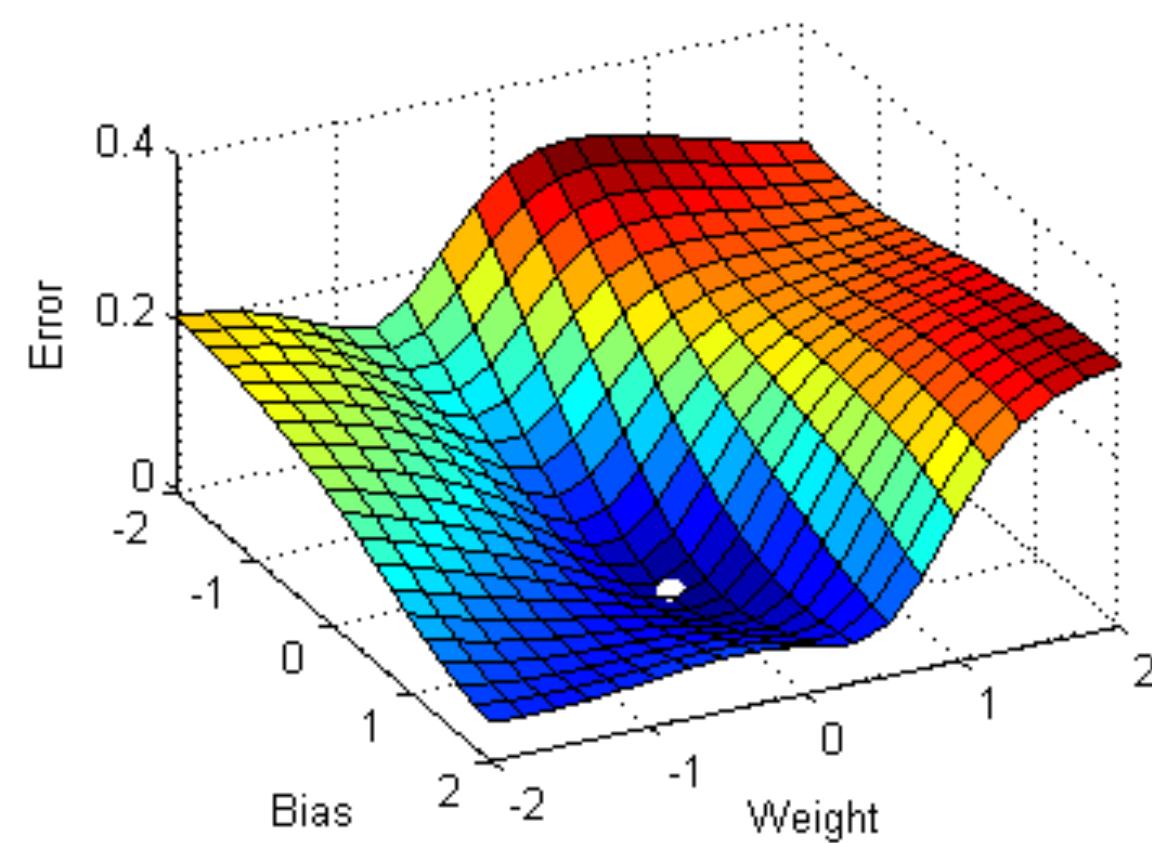
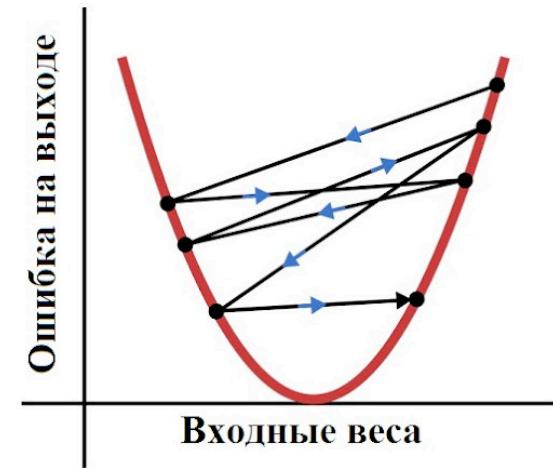
Функция потерь criterion = nn.CrossEntropyLoss()



$2^0 = 1$	$\log_2(1) = 0$
$2^1 = 2$	$\log_2(2) = 1$
$2^2 = 4$	$\log_2(4) = 2$
$2^3 = 8$	$\log_2(8) = 3$

Градиентный спуск

lr - шаг обучения. Данный параметр можно изменять.
optimizer = optim.SGD(model.parameters(), lr=0.005)



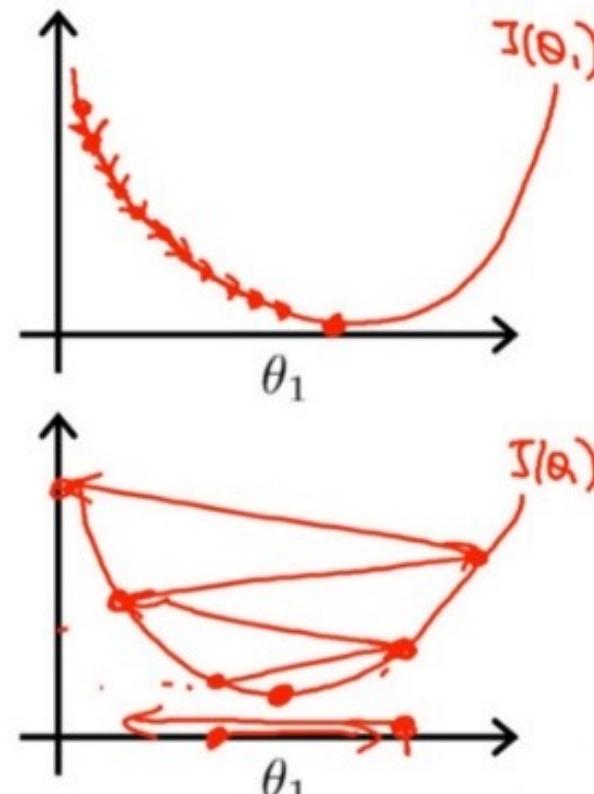
Скорость обучения

- Параметры (веса) сети меняются при обучении
- Гиперпараметры – нет. Управляем обучением

$$\theta_1 := \theta_1 - \alpha \frac{\partial}{\partial \theta_1} J(\theta_1)$$

Если α слишком маленькая, то градиентный спуск может быть слишком медленным.

Если α слишком большая, то градиентный спуск может не попасть в точку минимума. Кроме того, сходимость может быть не достигнута.



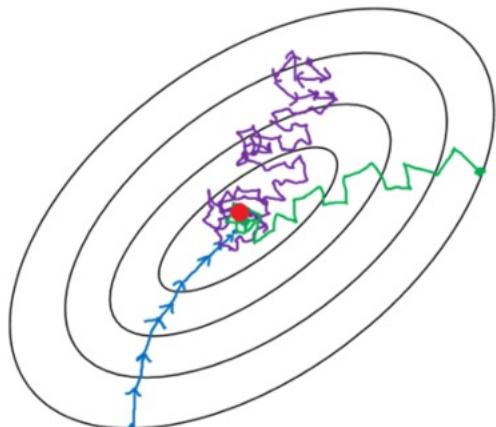
Итерации и эпохи

- Итерации – один шаг обучения
- Эпоха – обход всех экземпляров набора данных

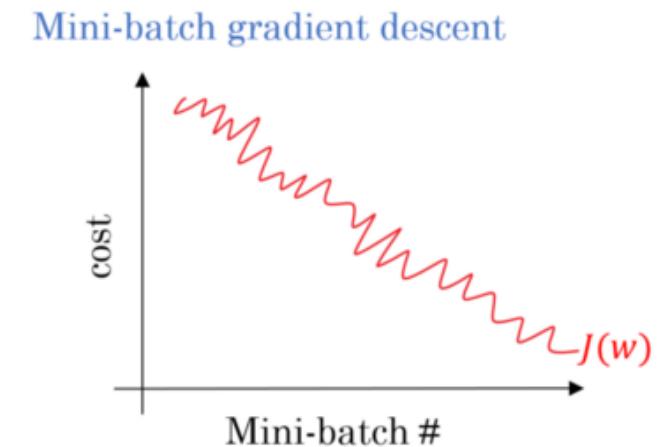
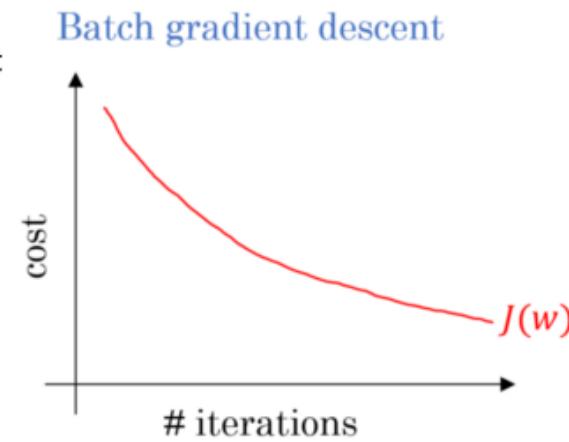
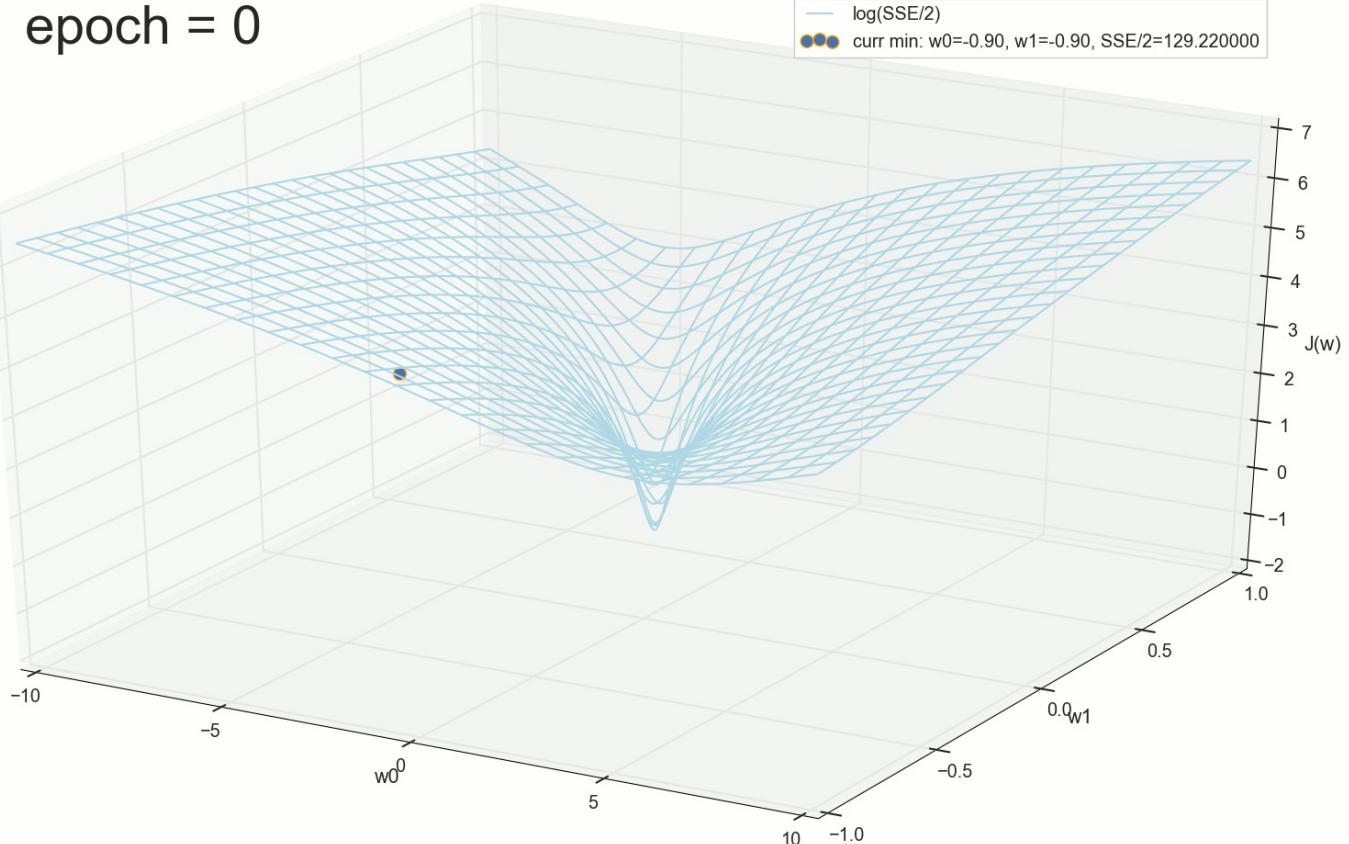


БАТЧИ

- Функцию потерь считаем по одному примеру, но потом их складываем в одну
- Обучать на всем датасете – долго. Каждый раз берем небольшую порцию данных
- Но обучение становится хаотичнее



— Batch gradient descent
— Mini-batch gradient Descent
— Stochastic gradient descent



Стохастическое vs пакетное обновление

Стохастическое

– Преимущества:

- Быстрая сходимость на больших избыточных данных
- Стохастическая траектория позволяет избежать локальных минимумов

– Недостатки:

- Продолжает «прыгать», если скорость обучения не уменьшается
- Теоретические условия сходимости не так понятны, как для пакетного обновления
- Доказательства сходимости вероятностны
- Большинство хороших способов ускорения или методов второго порядка не работают со стохастическим градиентом
- Сложнее распараллелить, чем пакетное обновление

Пакетное

– Преимущества:

- Гарантированное сходимость к локальному минимуму в простых условиях
- Много способов и методов второго порядка для ускорения
- Простые доказательства сходимости

– Недостатки:

- Болезненно медленный на больших задачах
- Несмотря на длинный список недостатков для стохастического обновления, это то, что большинство людей используют (что справедливо, по крайней мере, для больших задач).

Обучение

```
EPOCHS = 250
steps_per_epoch = len(dataloader['train'])
steps_per_epoch_val = len(dataloader['test'])
for epoch in range(EPOCHS): # проход по набору данных несколько раз
    running_loss = 0.0
    model.train()
    for i, batch in enumerate(dataloader['train'], 0):
        # получение одного минибатча; batch это двухэлементный список из [inputs, labels]
        inputs, labels = batch

        # очищение прошлых градиентов с прошлой итерации
        optimizer.zero_grad()

        # прямой + обратный проходы + оптимизация
        outputs = model(inputs)
        loss = criterion(outputs, labels)
        #loss = F.cross_entropy(outputs, labels)
        loss.backward()

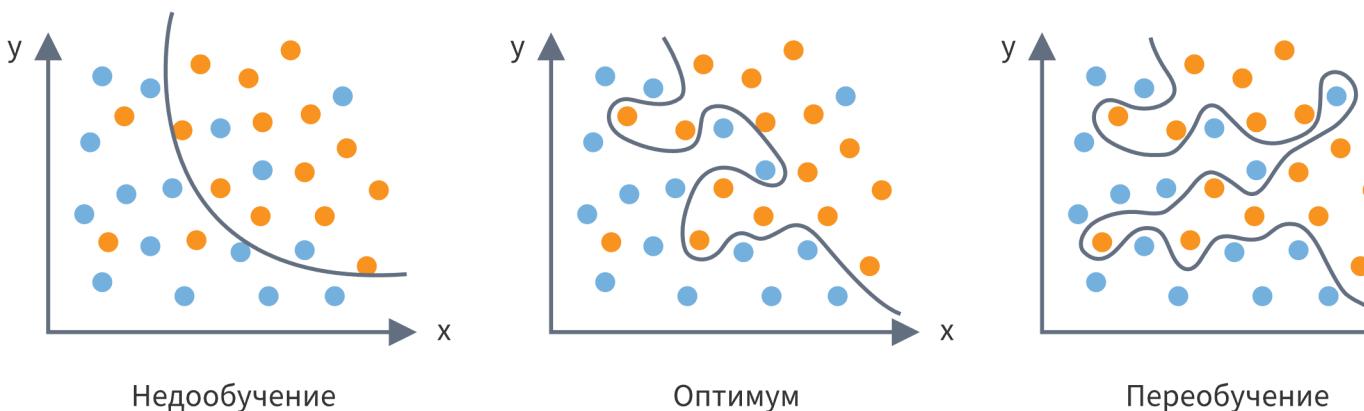
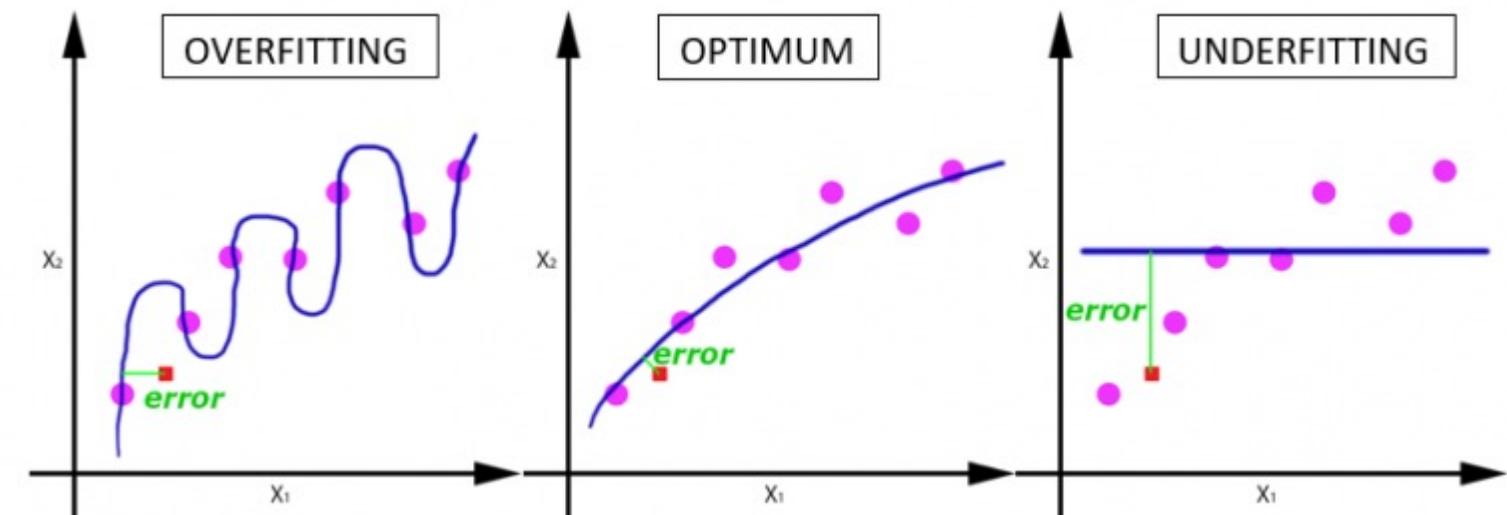
        #Для обновления параметров нейронной сети используется метод step, применённый к экземпляру
        optimizer.step()
```

Итоговая точность

train		precision	recall	f1-score	support
	0	0.9940	1.0000	0.9970	500
	55	1.0000	0.9900	0.9950	500
	58	0.9960	1.0000	0.9980	500
		accuracy		0.9967	1500
macro avg		0.9967	0.9967	0.9967	1500
weighted avg		0.9967	0.9967	0.9967	1500

test		precision	recall	f1-score	support
	0	0.8120	0.9500	0.8756	100
	55	0.7396	0.7100	0.7245	100
	58	0.7471	0.6500	0.6952	100
		accuracy		0.7700	300
macro avg		0.7662	0.7700	0.7651	300
weighted avg		0.7662	0.7700	0.7651	300

Точность, переобучение



- Переобучение при долгом обучении слишком сложной модели

