

Лекция 1

Обучение с учителем

Разработка нейросетевых систем

Канев Антон Игоревич

Материалы лекции



This NVIDIA DLI Certificate has been awarded to

Anton Kanev

for the successful completion of
Fundamentals of Deep Learning

Will Ramey



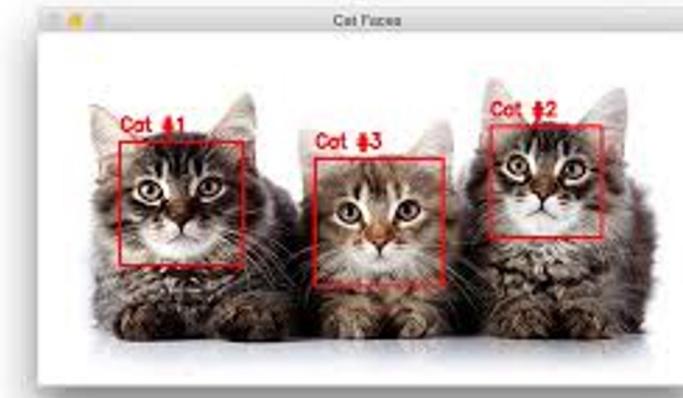
В лекции использованы материалы DLI Teaching Kit Deep Learning лицензированные компанией NVIDIA и New York University под лицензией [Creative Commons Attribution-NonCommercial 4.0 International License](#).

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

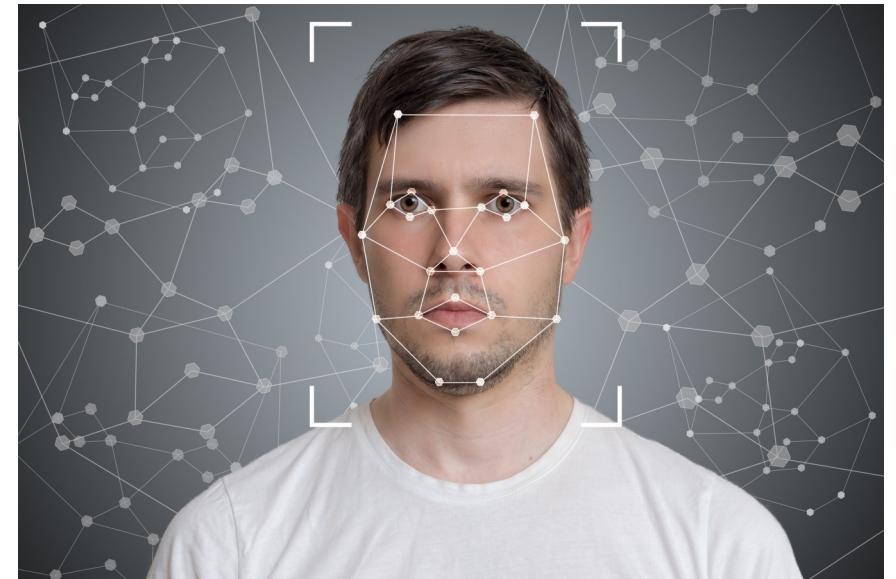
First Name

Last Name



Распознавание лиц/угроз на изображении

- Различные методы распознавания лиц

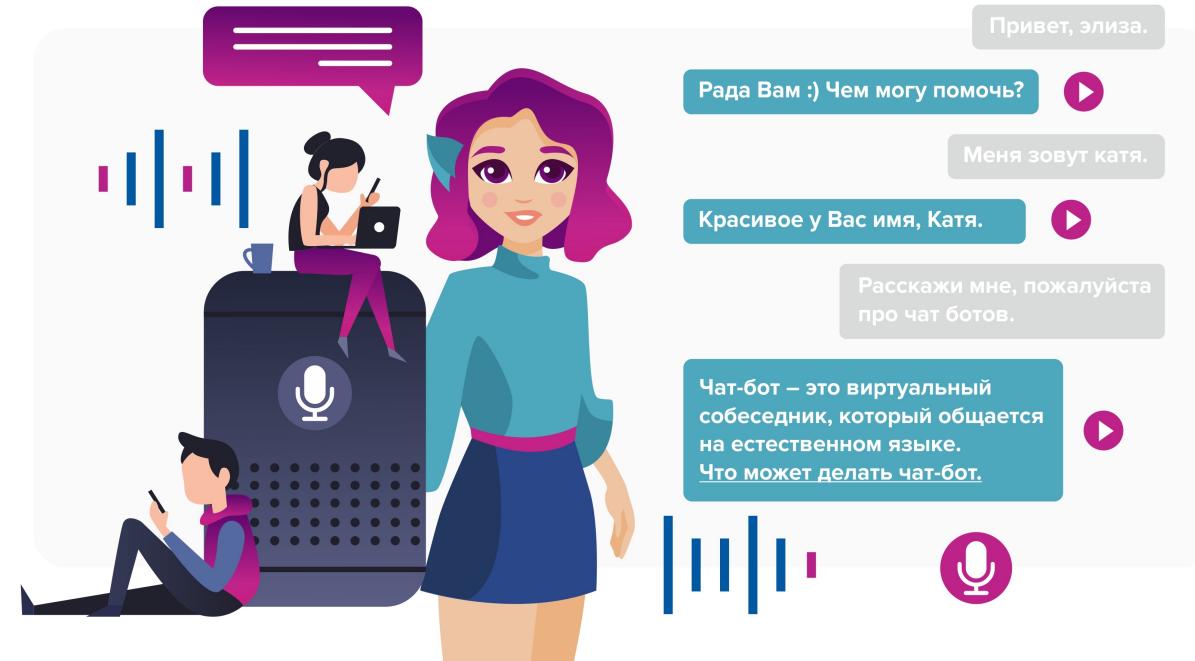
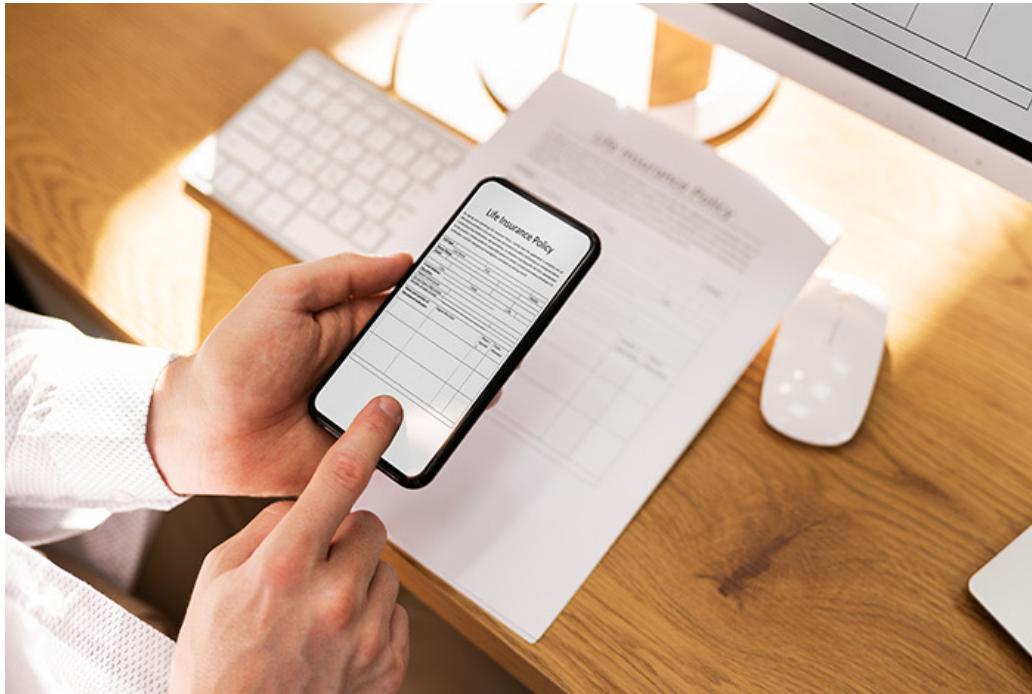


Распознавание и локализация yolo:

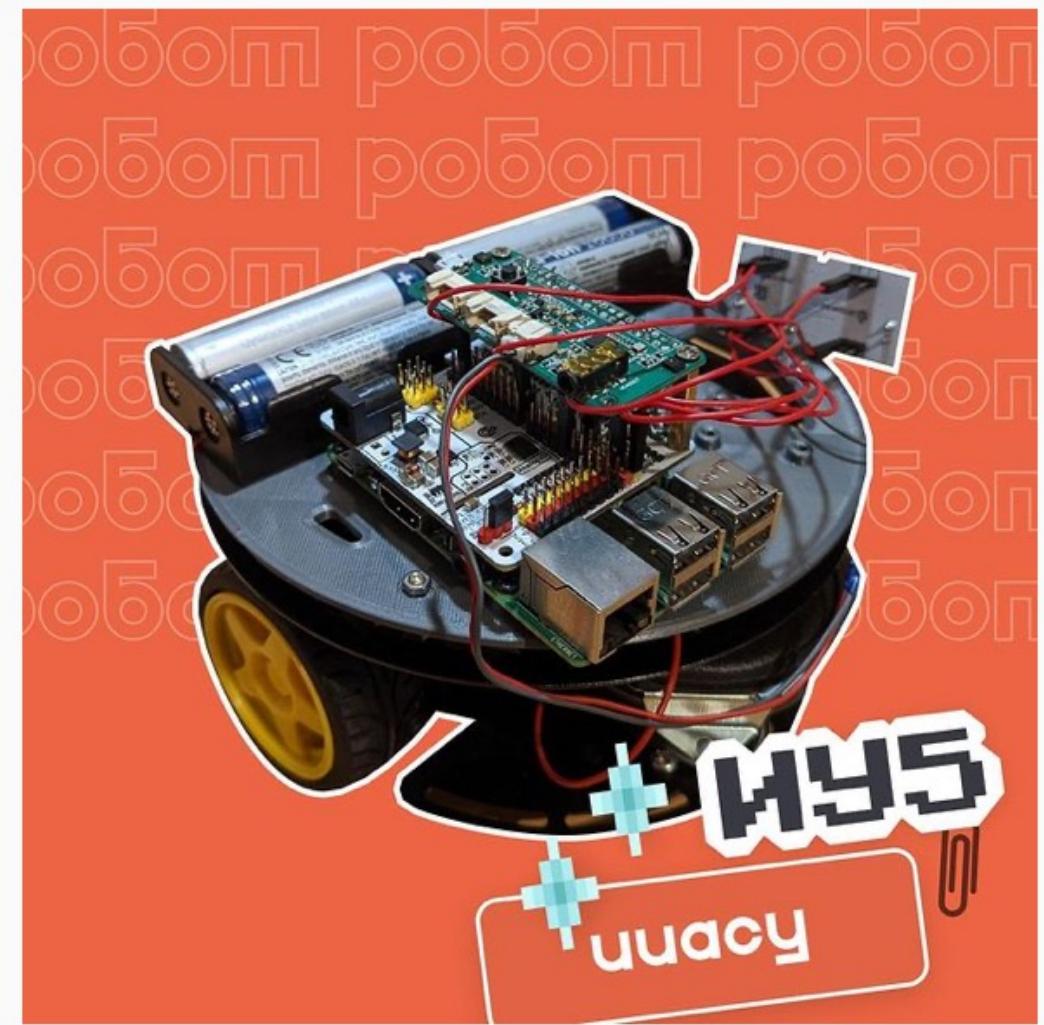
- Оружия
- Масок и тд

Обработка речи и текста

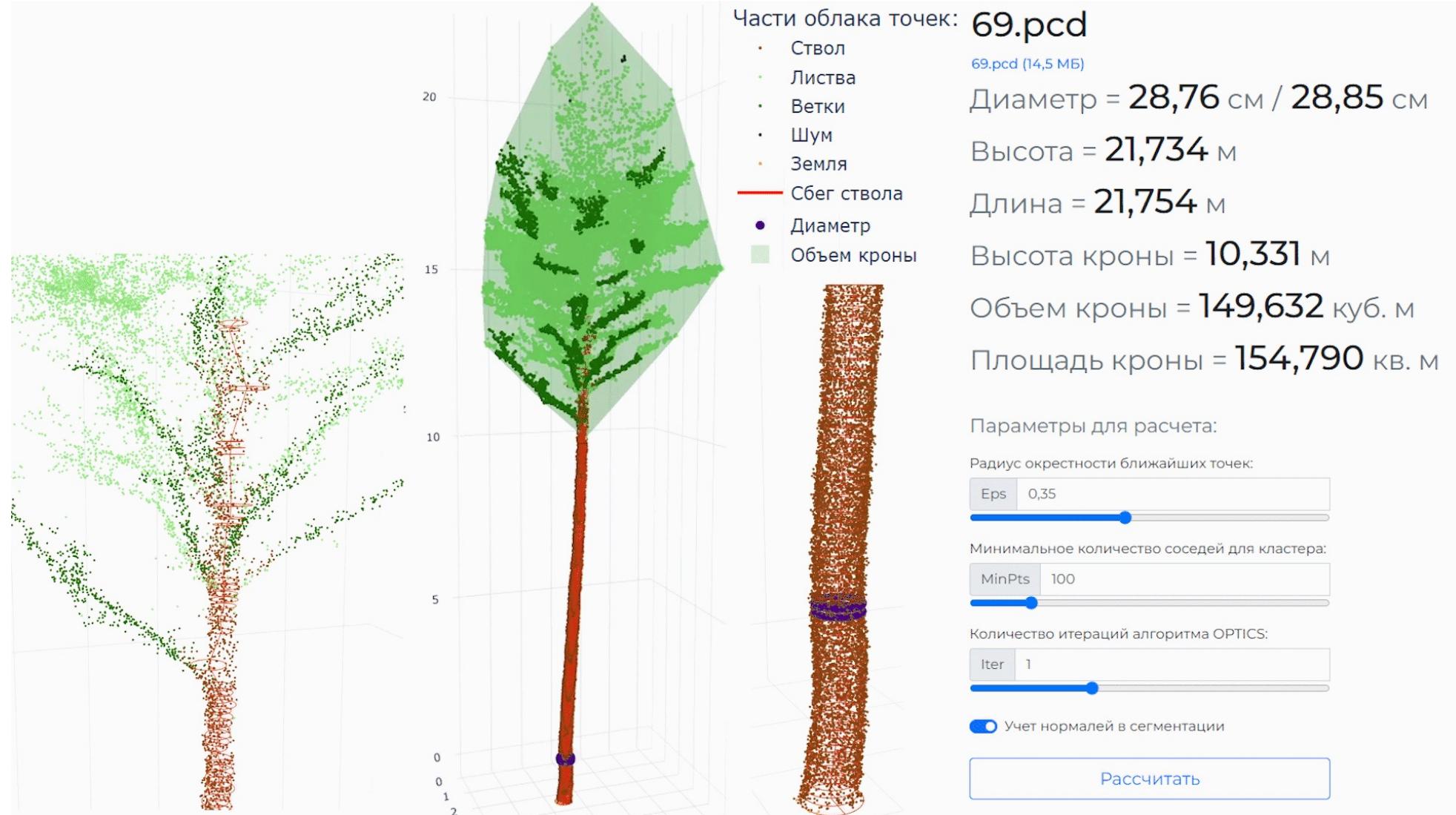
- Автоматическое распознавание речи
- Распознавание сканов - OCR



Areas of investigation



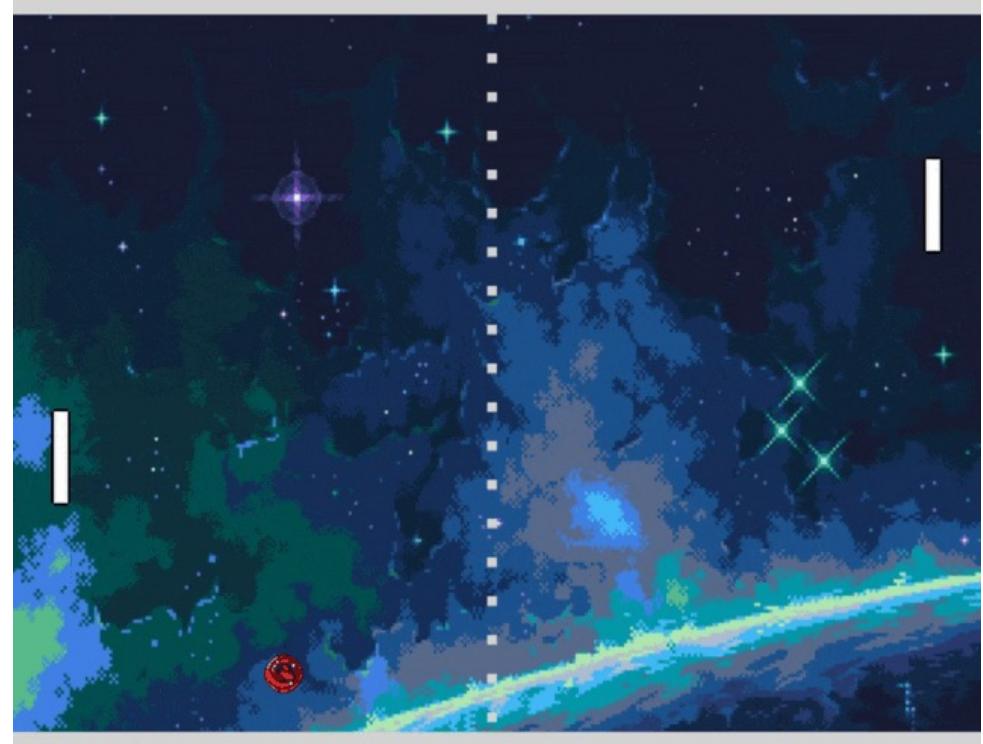
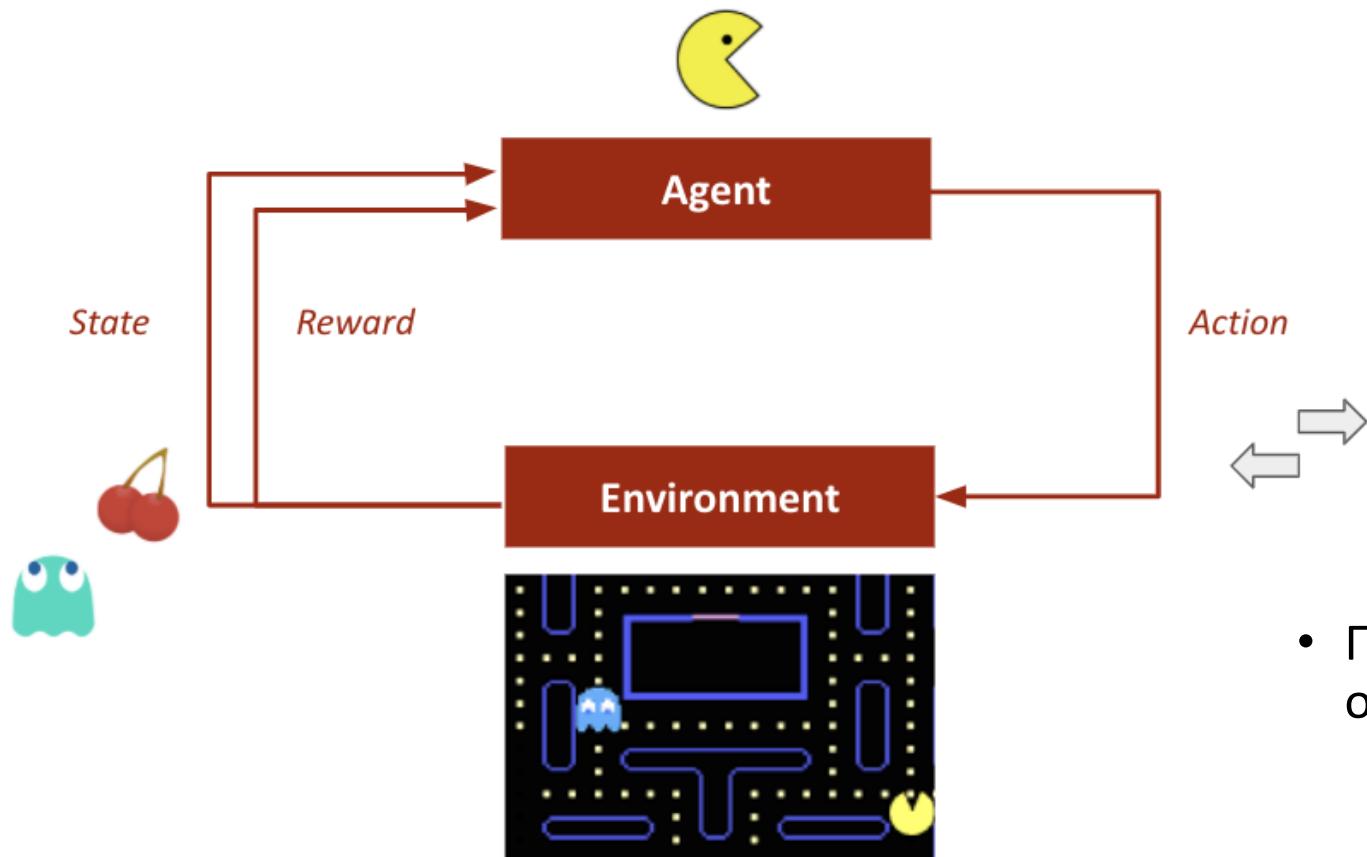
Lidar



*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

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

*Методы обучения с учителем

- Линейная регрессия
- Дерево решений
- Машина опорных векторов (SVM)
- К-ближайших соседей
- Нейронные сети

Фреймворки обучения с учителем

 PyTorch



TensorFlow

 Keras

Признаки в машинном обучении

- Признаки – это наблюдения, которые используются для формирования прогнозов
 - Для классификации изображений каждый пиксель является признаком
 - Для распознавания голоса, частота и громкость звуковых примеров являются признаками
 - Для беспилотных автомобилей данные с камер, радаров и GPS являются признаками
- Извлечение подходящих признаков важно для построения модели
 - Время суток - это неподходящий признак при классификации изображений
 - Время суток - это подходящий признак при классификации электронных писем, т.к. SPAM часто приходит по ночам
- Общие типы признаков в робототехнике
 - Пиксели (RGB данные)
 - Глубина (сонар, лазерные дальномеры)
 - Движение (значения с микросхем)
 - Ориентация или ускорение (Гироскоп, Акселерометр, Компас)

*Формирование набора данных. Примеры, входные данные и label

- Размер набора данных и распределение по классам
- Части набора данных: входные данные и метки; test и train



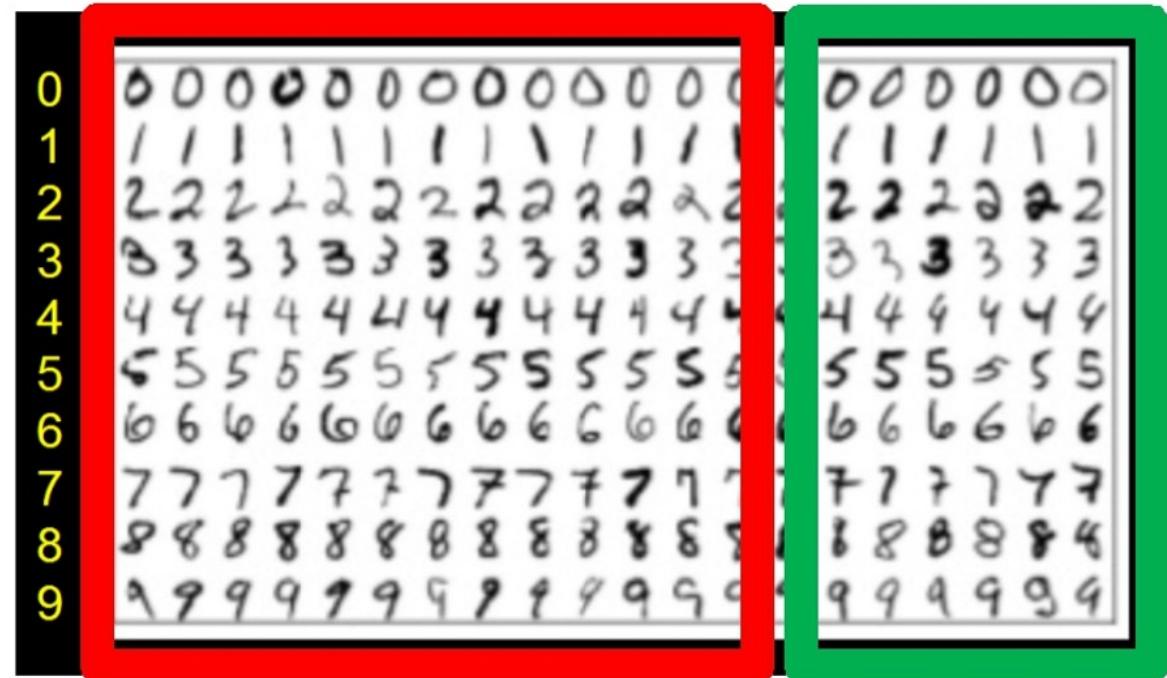
*Тестовая и обучающая выборки

- 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

Тестовая выборка



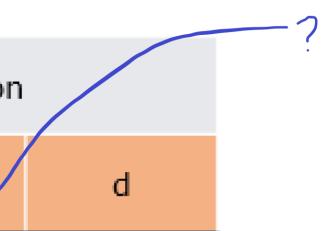
*Пример: Определение кошек

Прогноз:						
Картина:						
	Истинно Положительное (TP)	Истинно Отрицательное (TN)	Ложно Отрицательное (FN)	Ложно Положительное (FP)		

*Матрица ошибок

- 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
- Confusion Matrix для нескольких классов

		PREDICTED classification			
		a	b	c	d
ACTUAL classification	a	TN	FP	TN	TN
	b	FN	TP	FN	FN
	c	TN	FP	TN	TN
	d	TN	FP	TN	TN



*Метрики для классификации

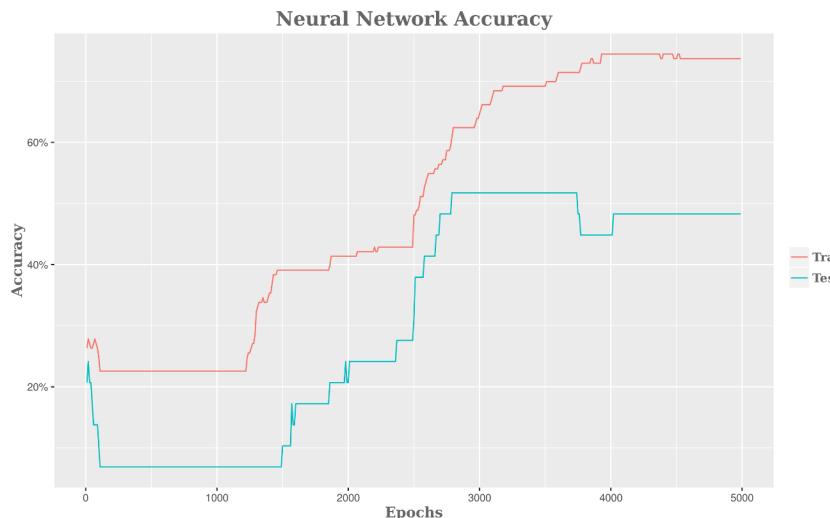
- Точность (Precision)
 - Процент положительных меток которые правильно определены
 - $Precision = (\# \text{ true positives}) / (\# \text{ true positives} + \# \text{ false positives})$
- Полнота (Recall)
 - Процент положительных примеров которые были правильно определены
 - $Recall = (\# \text{ true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

Остальные метрики

https://en.m.wikipedia.org/wiki/Sensitivity_and_specificity

Обучающие и тестовые данные

- Обучающие данные
 - Данные, на которых проводится обучение модели
- Тестовые данные
 - Данные, на которых проводится измерение точности модели
- Переобучение (overfitting)
 - Модель которая хорошо работает на обучающих данных и плохо на тестовых данных



Смещение и разброс (Bias and variance)

- Смещение: ожидаемая разница между прогнозами модели и истиной
- Разброс: Как сильно отличается ответ модели в обучающих наборах данных
- Возможные сценарии модели
 - Большое смещение: модель делает неправильный прогноз на обучающих примерах
 - Большой разброс: Модель не обобщает на новых наборах данных
 - Маленькое смещение: Модель делает правильный прогноз на обучающих данных
 - Маленький разброс: Модель обобщает на новых наборах данных

Результаты из лабораторной

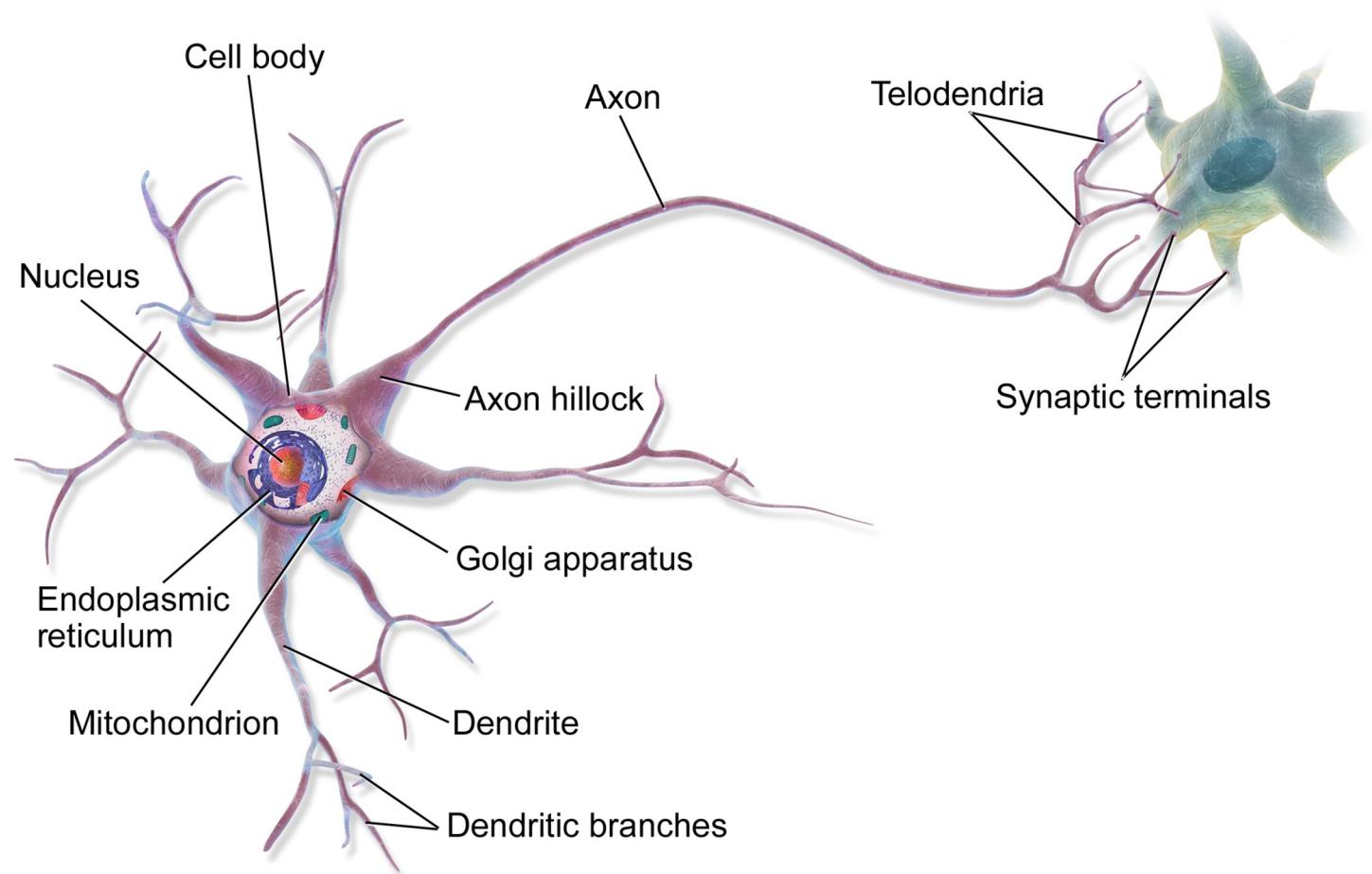
train

	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

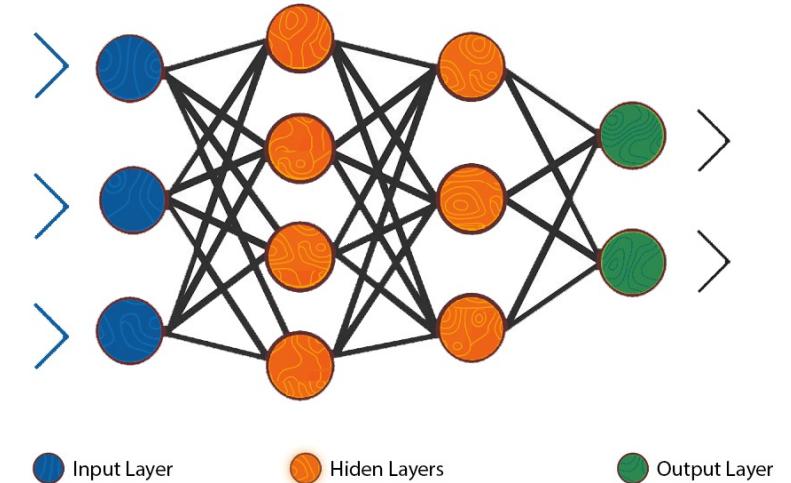
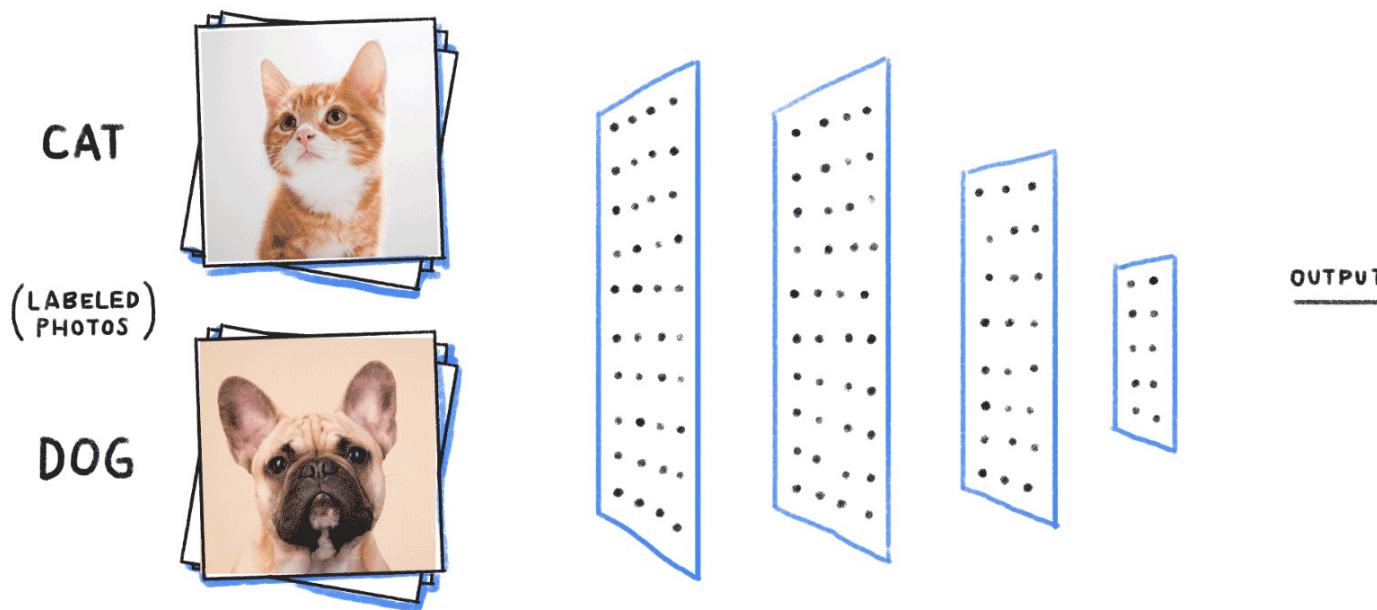
	precision	recall	f1-score	support
0	0.8120	0.9500	0.8756	100
55	0.7396	0.7100	0.7245	100
58	0.7471	0.6500	0.6952	100
accuracy			0.7700	300
macro avg	0.7662	0.7700	0.7651	300
weighted avg	0.7662	0.7700	0.7651	300

Биологическая ассоциация



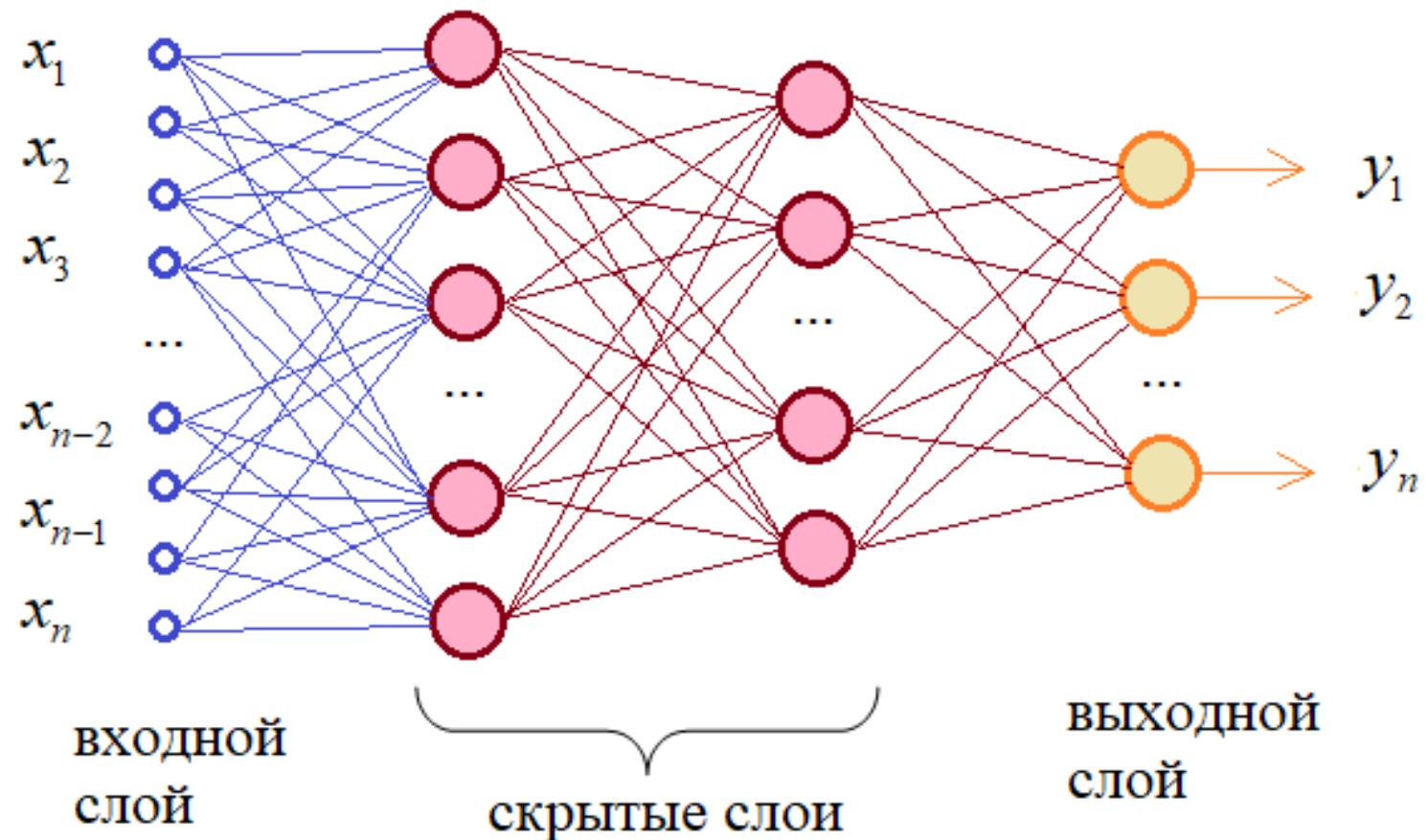
Нейронная сеть

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



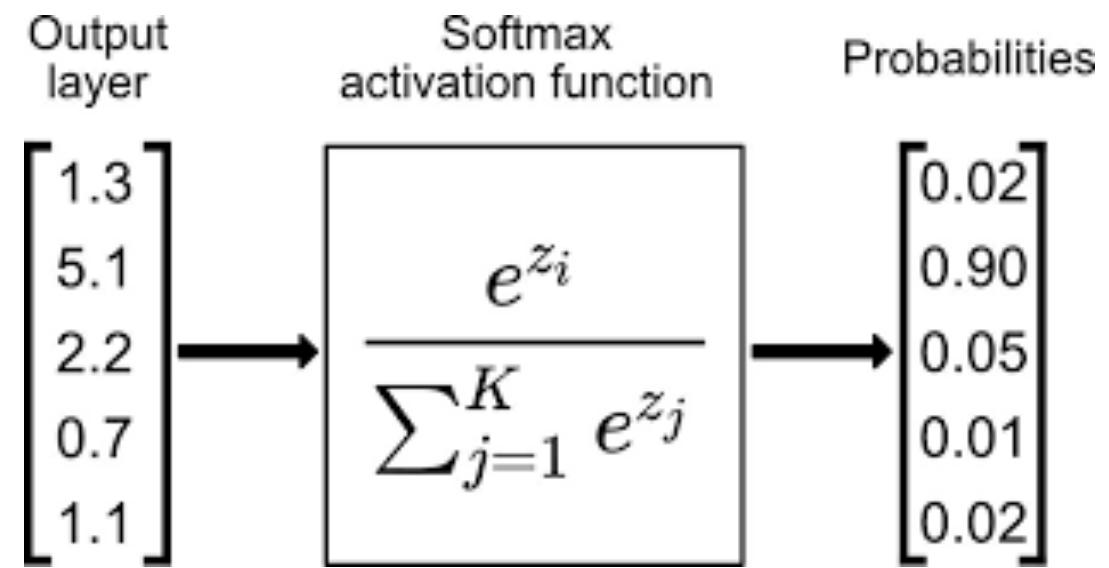
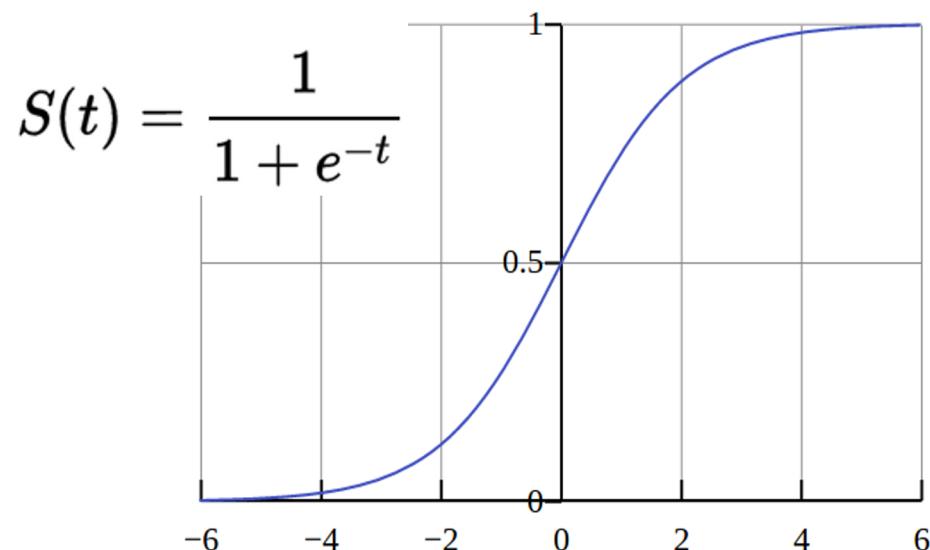
*Полносвязная нейросеть

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



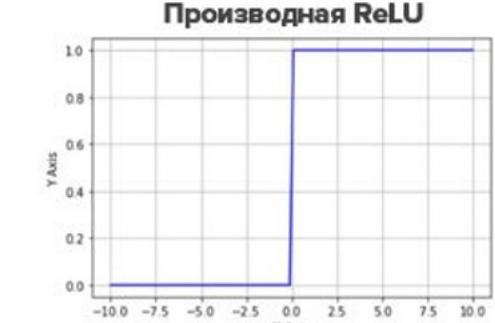
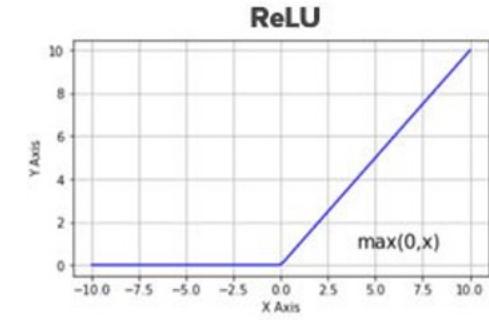
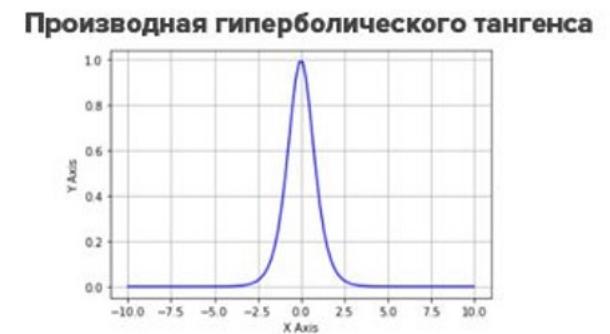
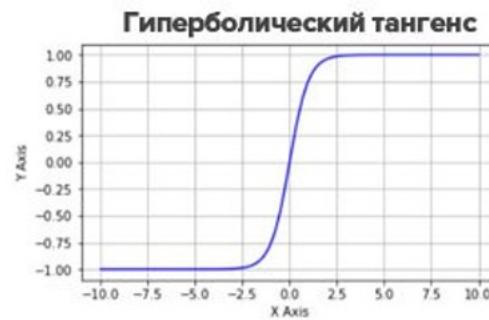
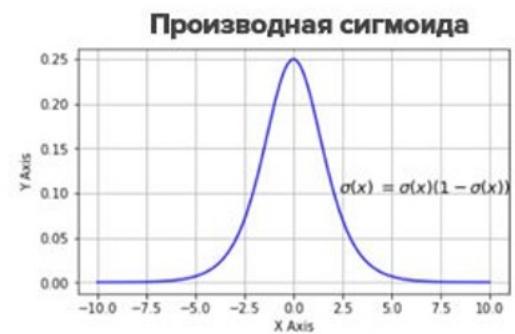
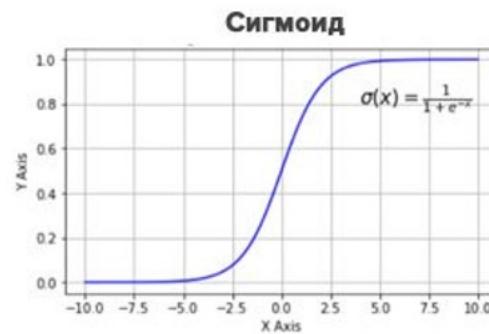
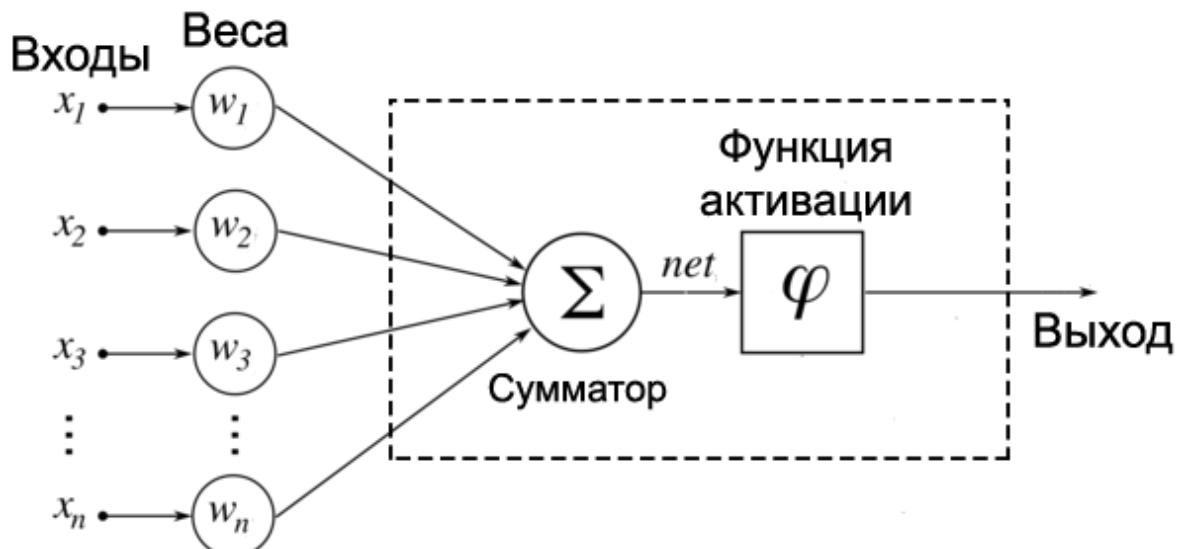
*Активационные функции

- Активационные функции применяются ко всем входным значениям в каждом нейроне
 - Сигмоидальная функция является распространенной активационной функцией
 - Softmax – это вариант сигмоиды для нескольких классов

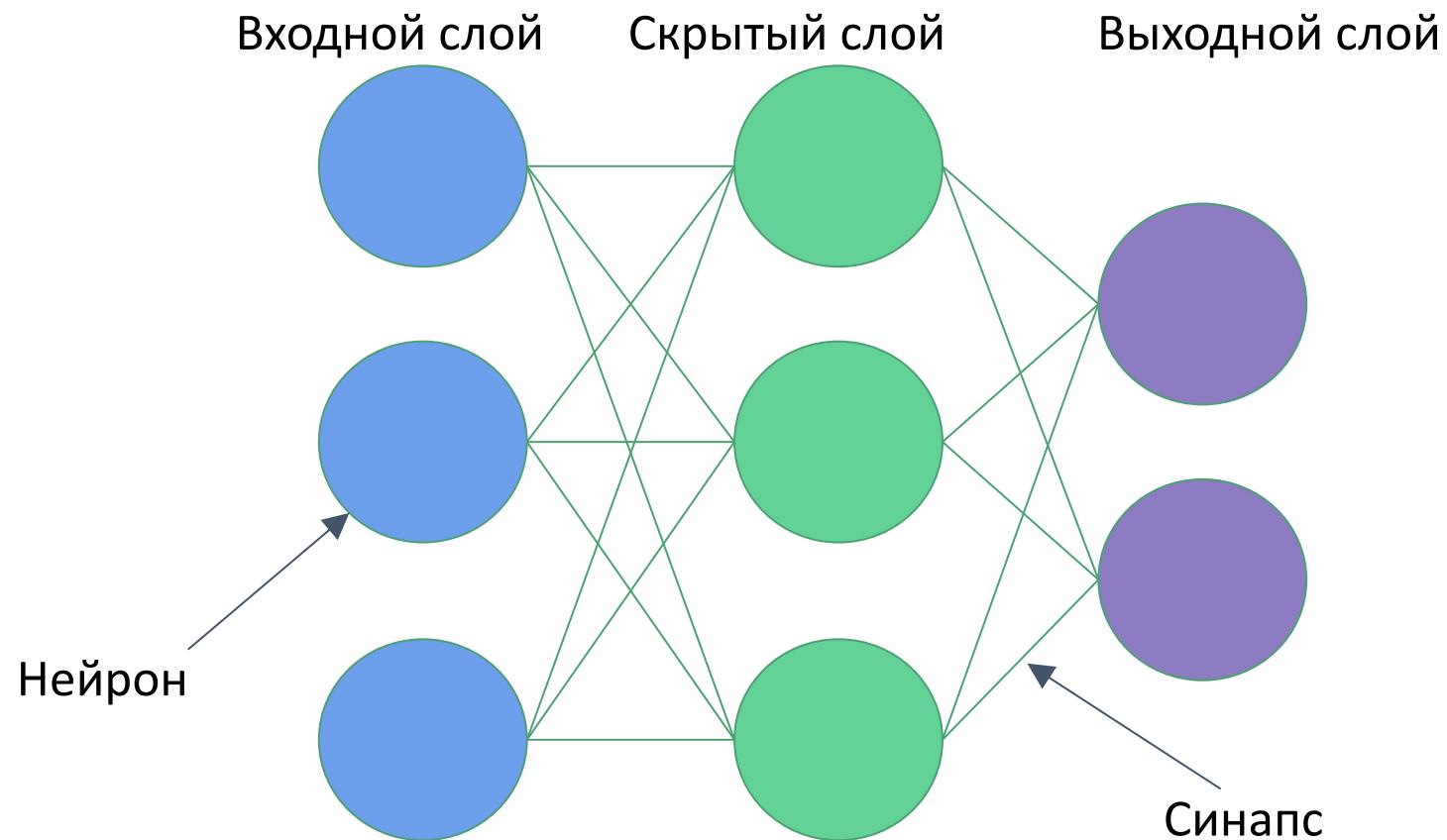


Активационная функция

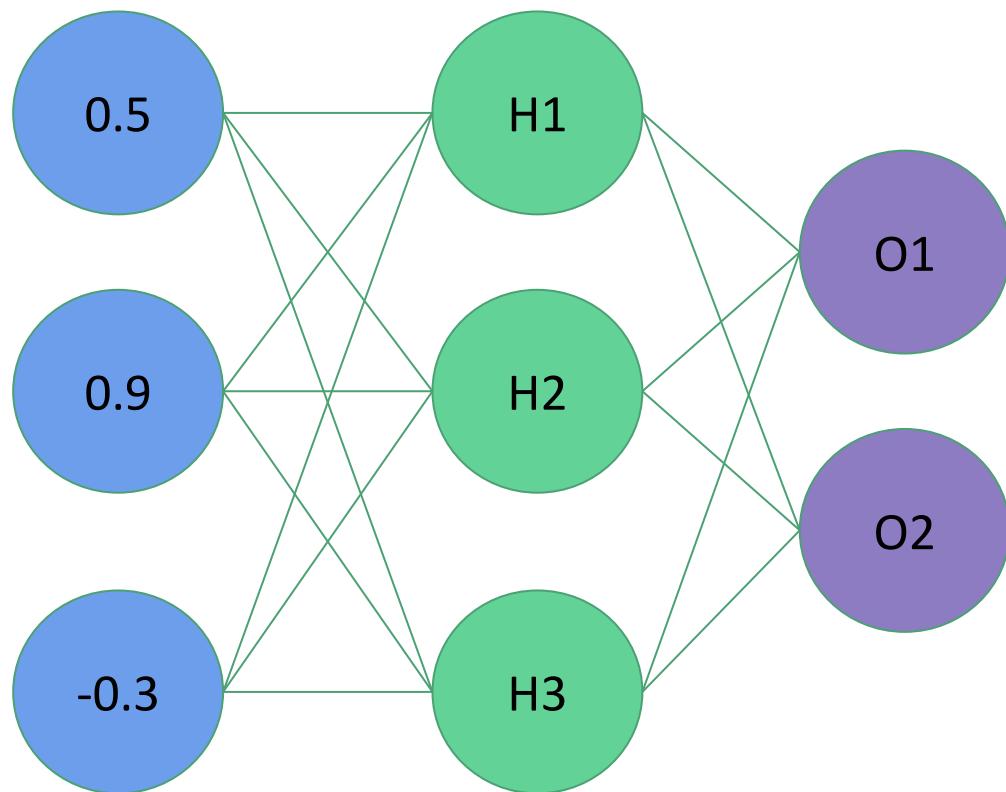
- Помимо сумматора есть активационная функция
- Они разные для разных задач



Архитектура нейронной сети



Вычисление прогноза (инференс)



Веса H1 = (1.0, -2.0, 2.0)

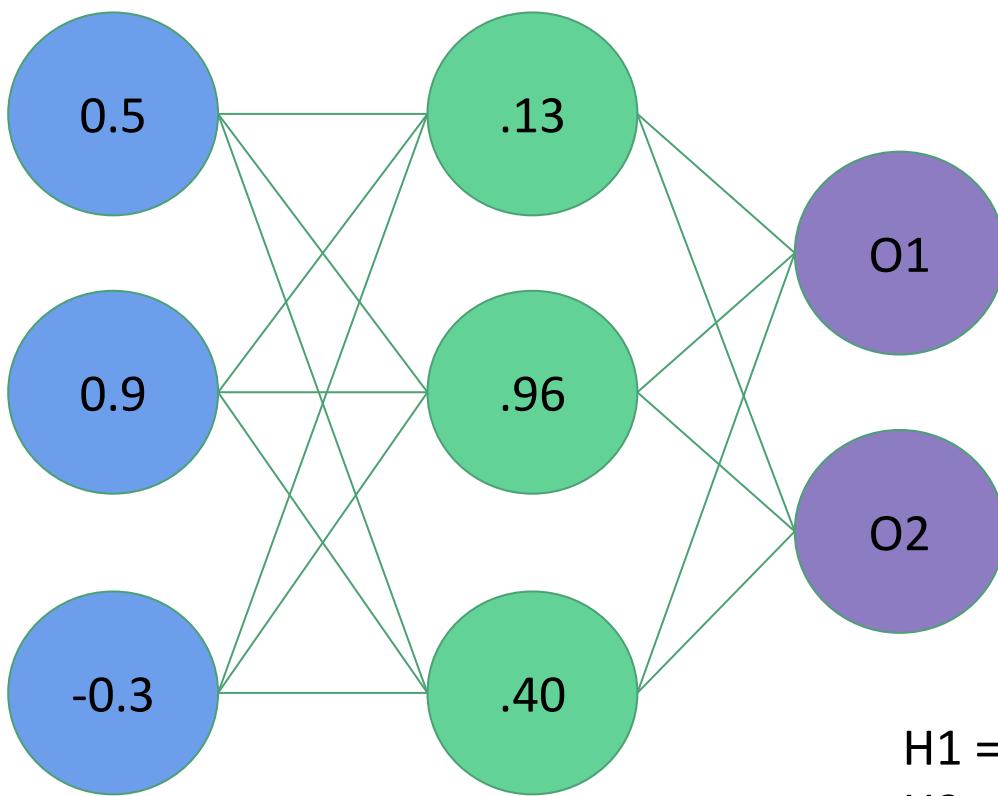
Веса H2 = (2.0, 1.0, -4.0)

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)



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

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

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

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

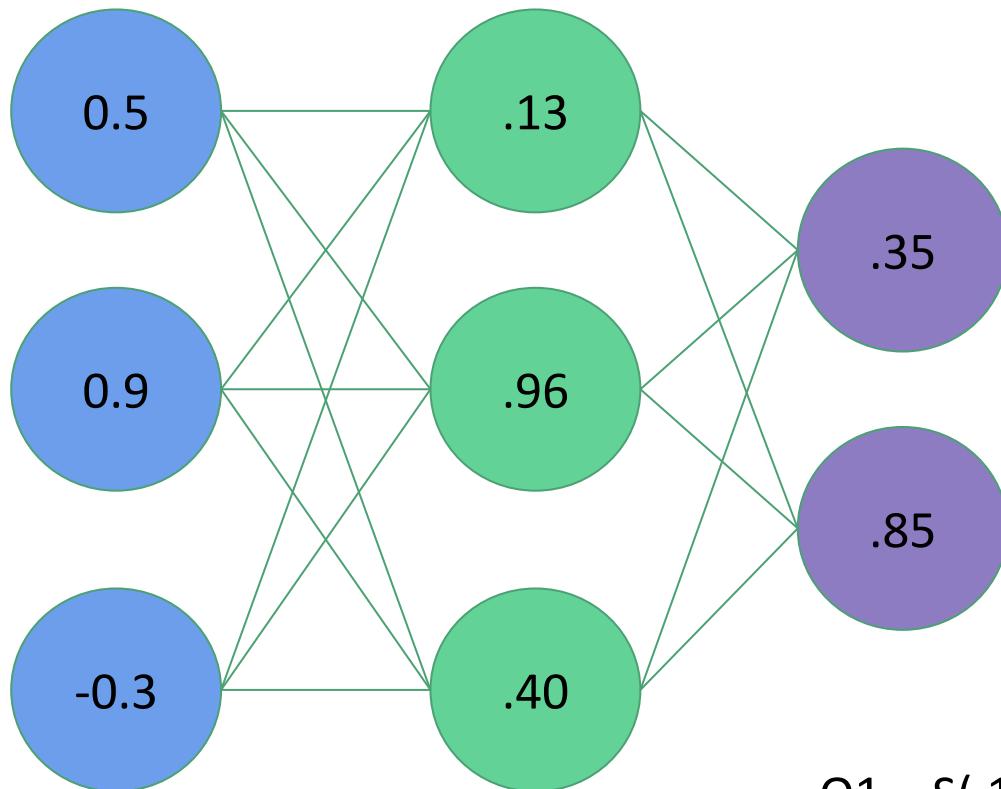
$$\text{Веса } O2 = (0.0, 1.0, 2.0)$$

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

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

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

*Вычисление прогноза (инференс)



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

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

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

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

$$\text{Веса } O2 = (0.0, 1.0, 2.0)$$

$$O1 = S(.13 * -3.0 + .96 * 1.0 + .40 * -3.0) = S(-.63) = .35$$

$$O2 = S(.13 * 0.0 + .96 * 1.0 + .40 * 2.0) = S(1.76) = .85$$

*Матричное представление

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 скрытого слоя Вход

$S($

1.0	-2.0	2.0
2.0	1.0	-4.0
1.0	-1.0	0.0

 *

0.5
0.9
-0.3

) = $S($

-1.9	3.1	-0.4
------	-----	------

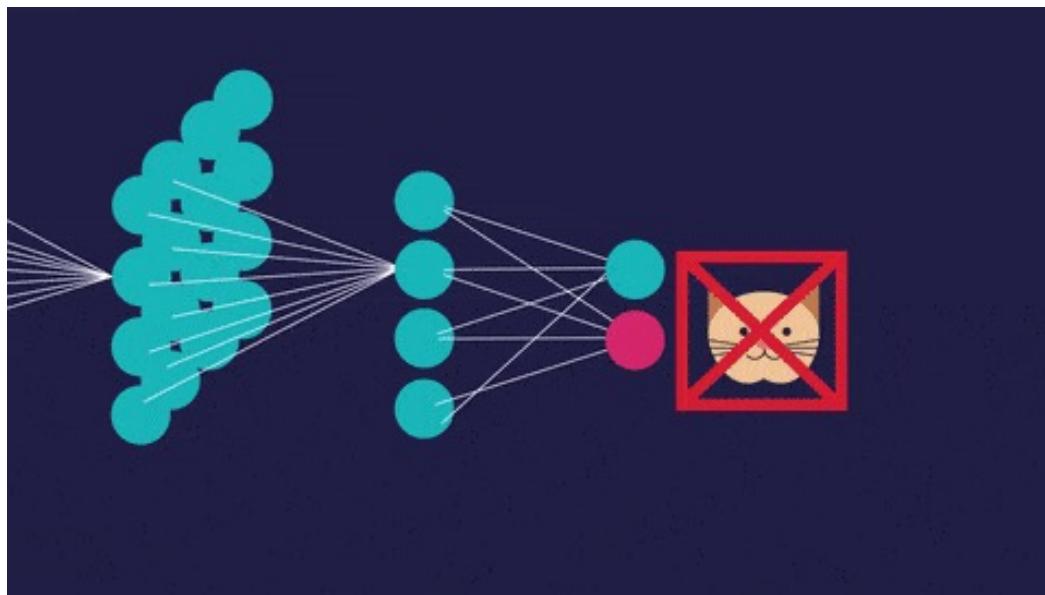
) =

.13	.96	0.4
-----	-----	-----

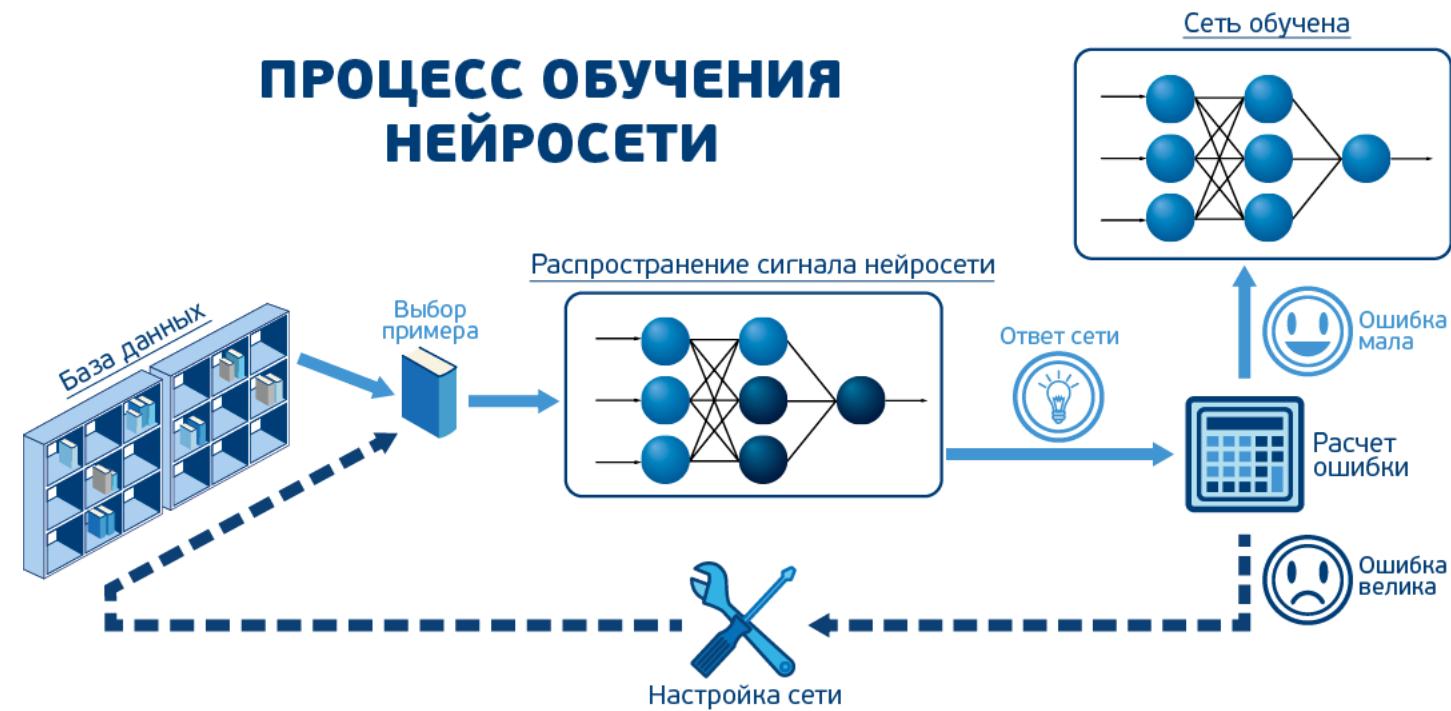
 Значения скрытого слоя

*Обучение нейросети

- Обучение нейросети
- Обучение с учителем



ПРОЦЕСС ОБУЧЕНИЯ НЕЙРОСЕТИ



*Обучение нейронных сетей

- 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
- Вклад в размер ошибки вычисляется с помощью обратного распространения (Backpropagation)
- Минимизация ошибки достигается благодаря градиентному спуску (Gradient Descent)

*Метод градиентного спуска

- 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

*Цикл обучения из лабораторной

```
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

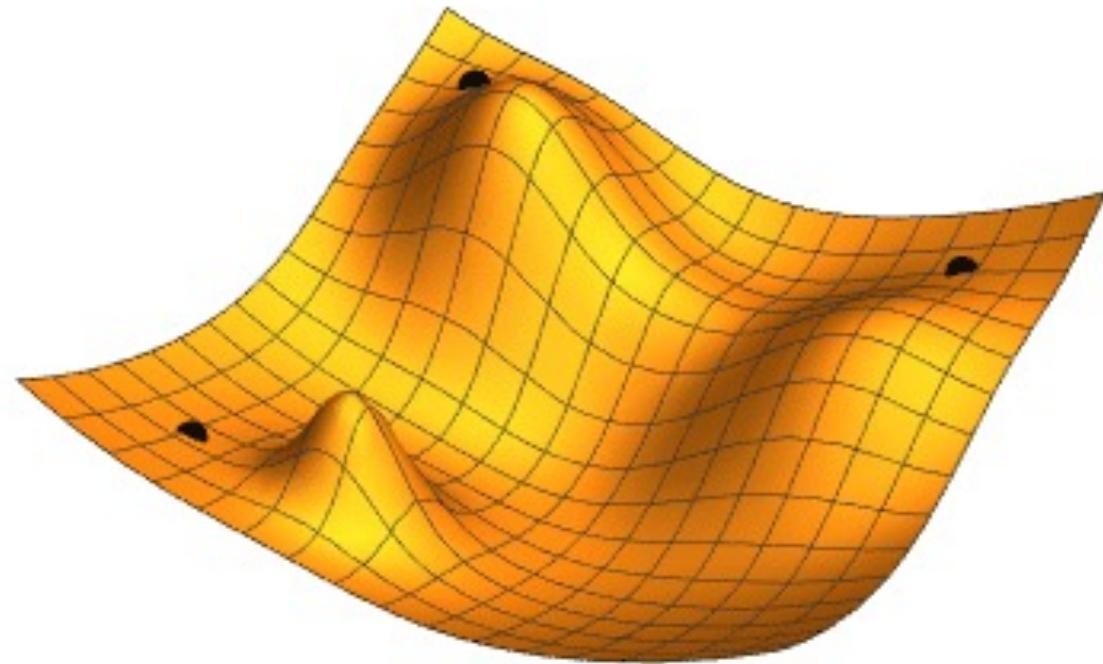
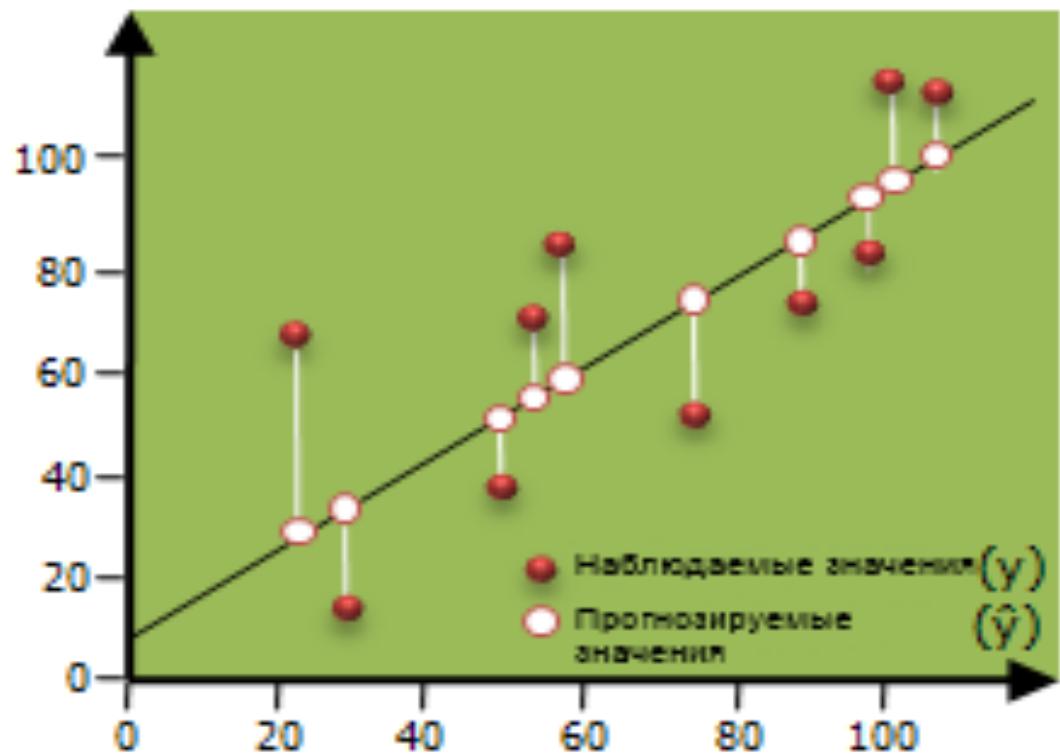
        # очищение прошлых градиентов с прошлой итерации
        optimizer.zero_grad()

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

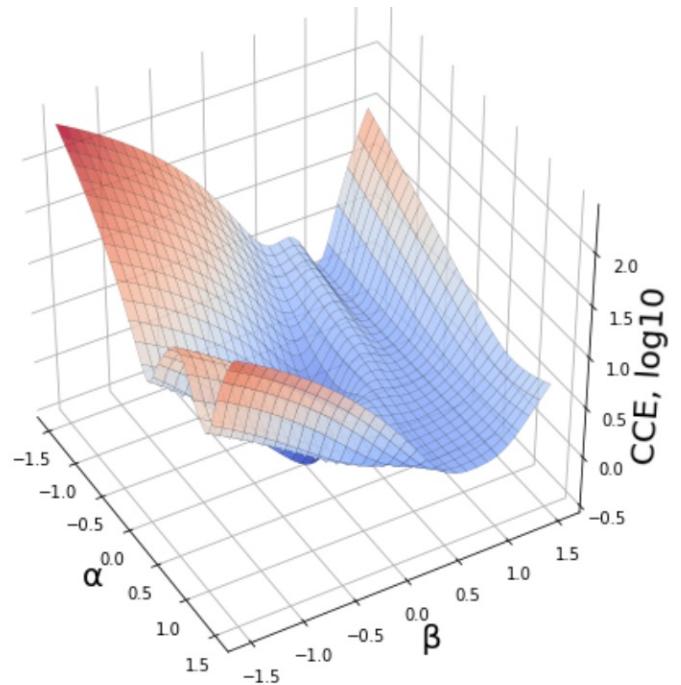
        #Для обновления параметров нейронной сети используется метод step, применённый к экземпляру
        optimizer.step()
```

ФУНКЦИЯ ПОТЕРЬ (loss)

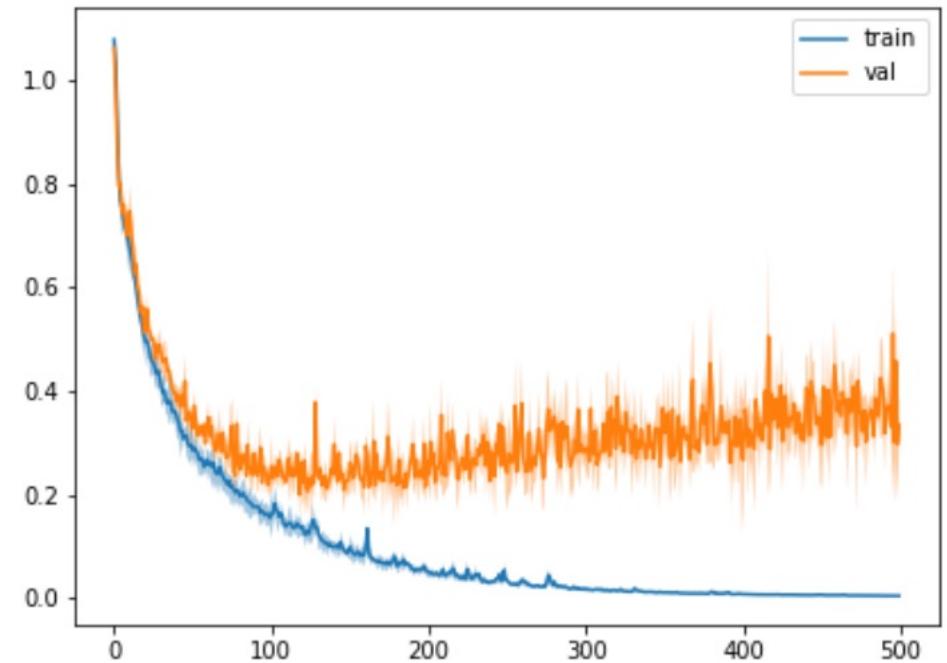
- Нам нужен минимум функции потерь
- Метод наименьших квадратов
- Кроссэнтропия



ФУНКЦИИ ПОТЕРЬ



Примеры из лабораторной

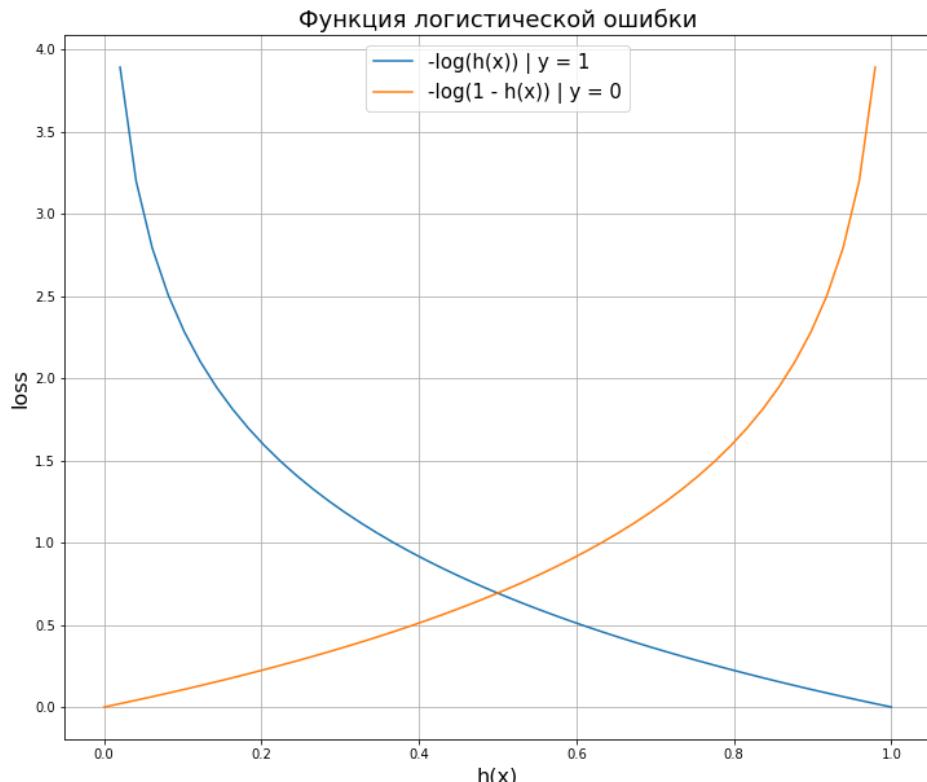


```
criterion = nn.CrossEntropyLoss()
```

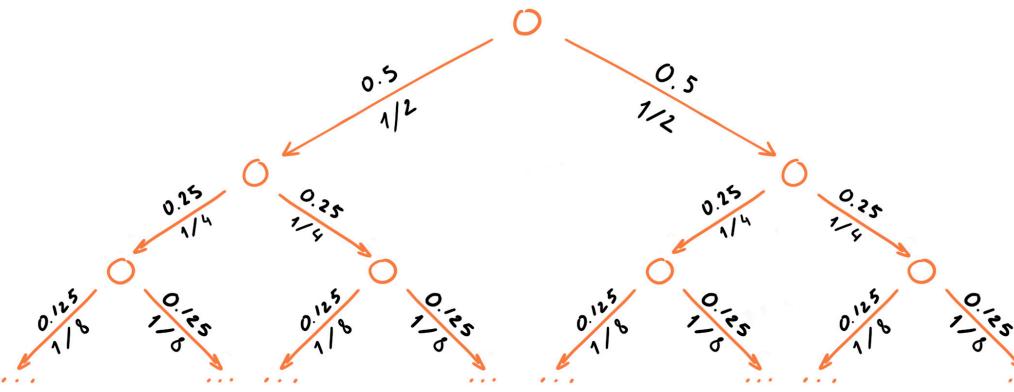
```
# Logloss
def loss(y_true, y_pred):
    return -1*(y_true*torch.log(y_pred)+(1-y_true)*torch.log(1-y_pred)).sum()
```

*Перекрестная энтропия

- В задачах классификации в качестве функции потерь используется перекрестная энтропия – cross entropy
- Случай бинарной кросс-энтропии



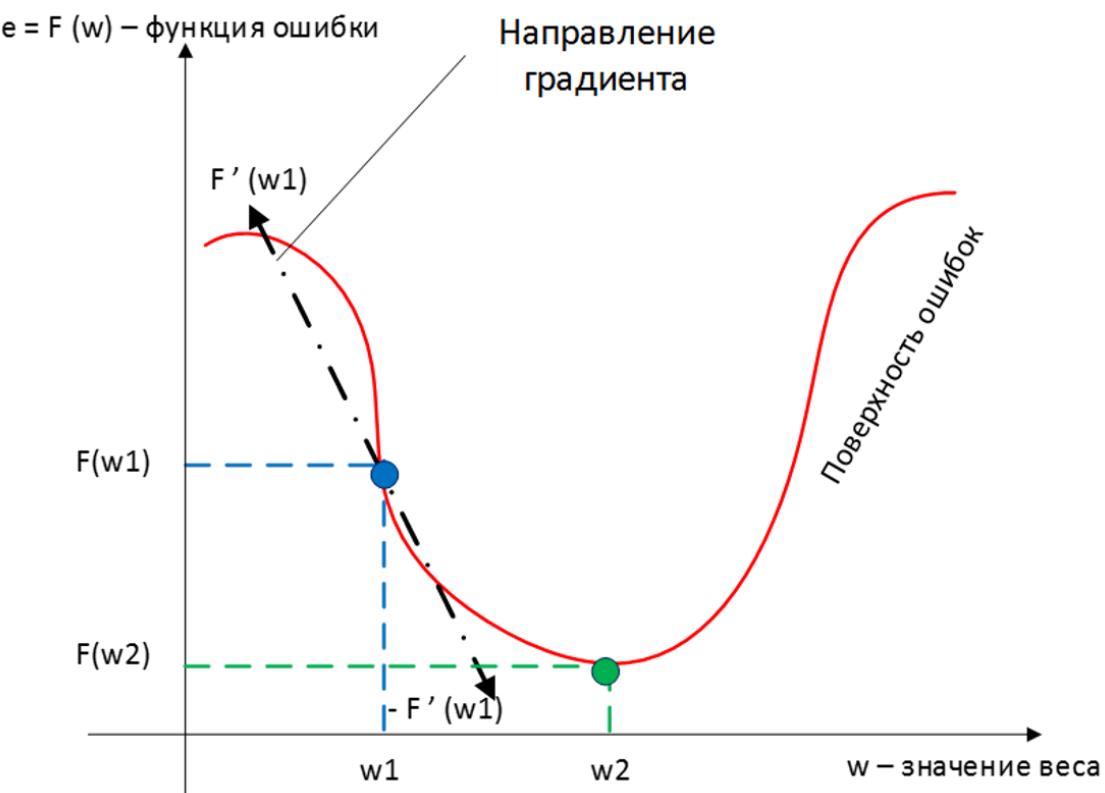
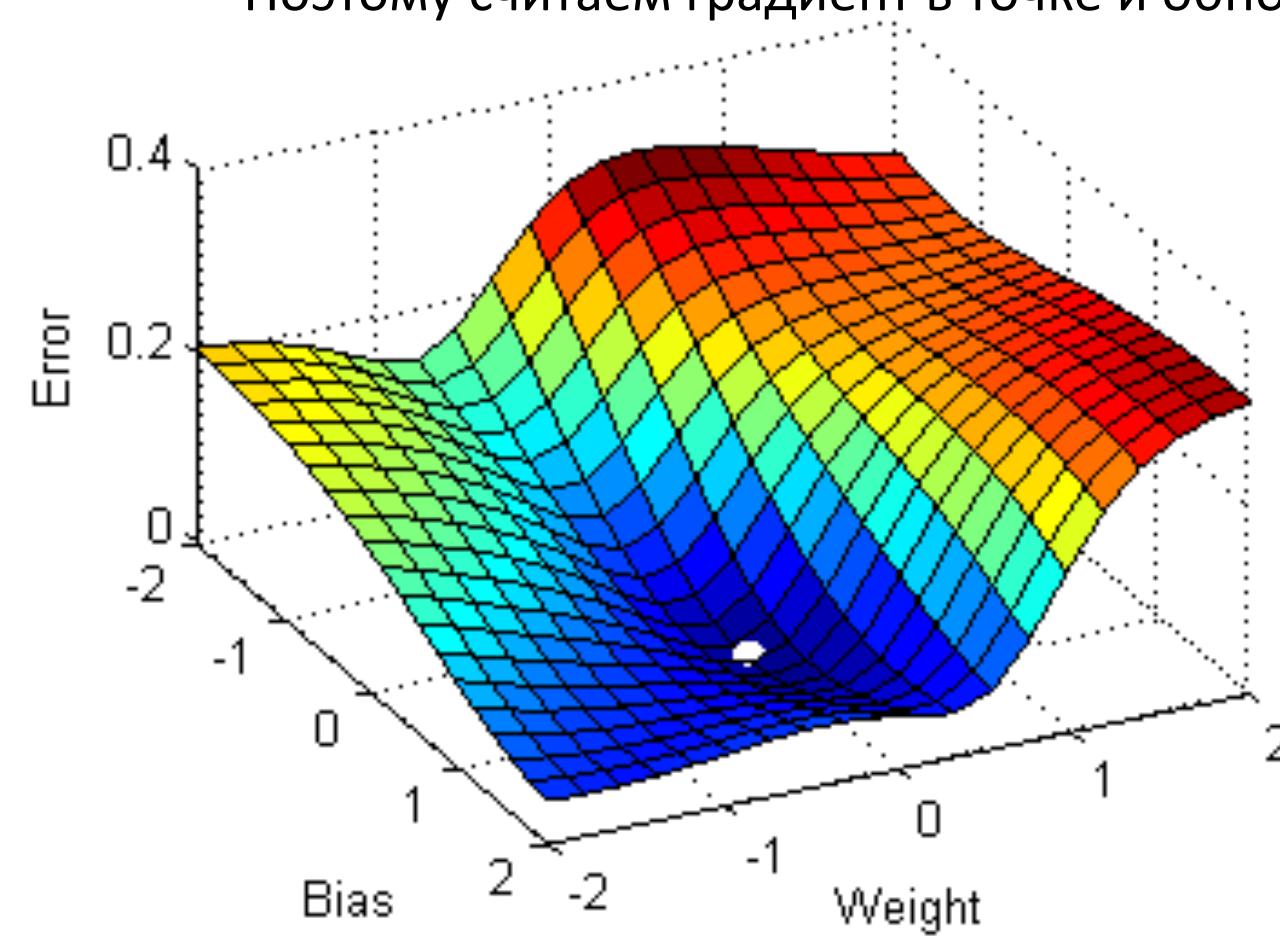
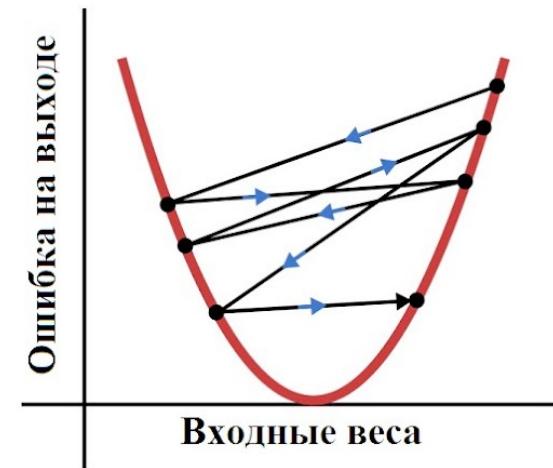
$$H_p(q) = -\frac{1}{N} \sum_{i=1}^N y_i \cdot \log(p(y_i)) + (1 - y_i) \cdot \log(1 - p(y_i))$$



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

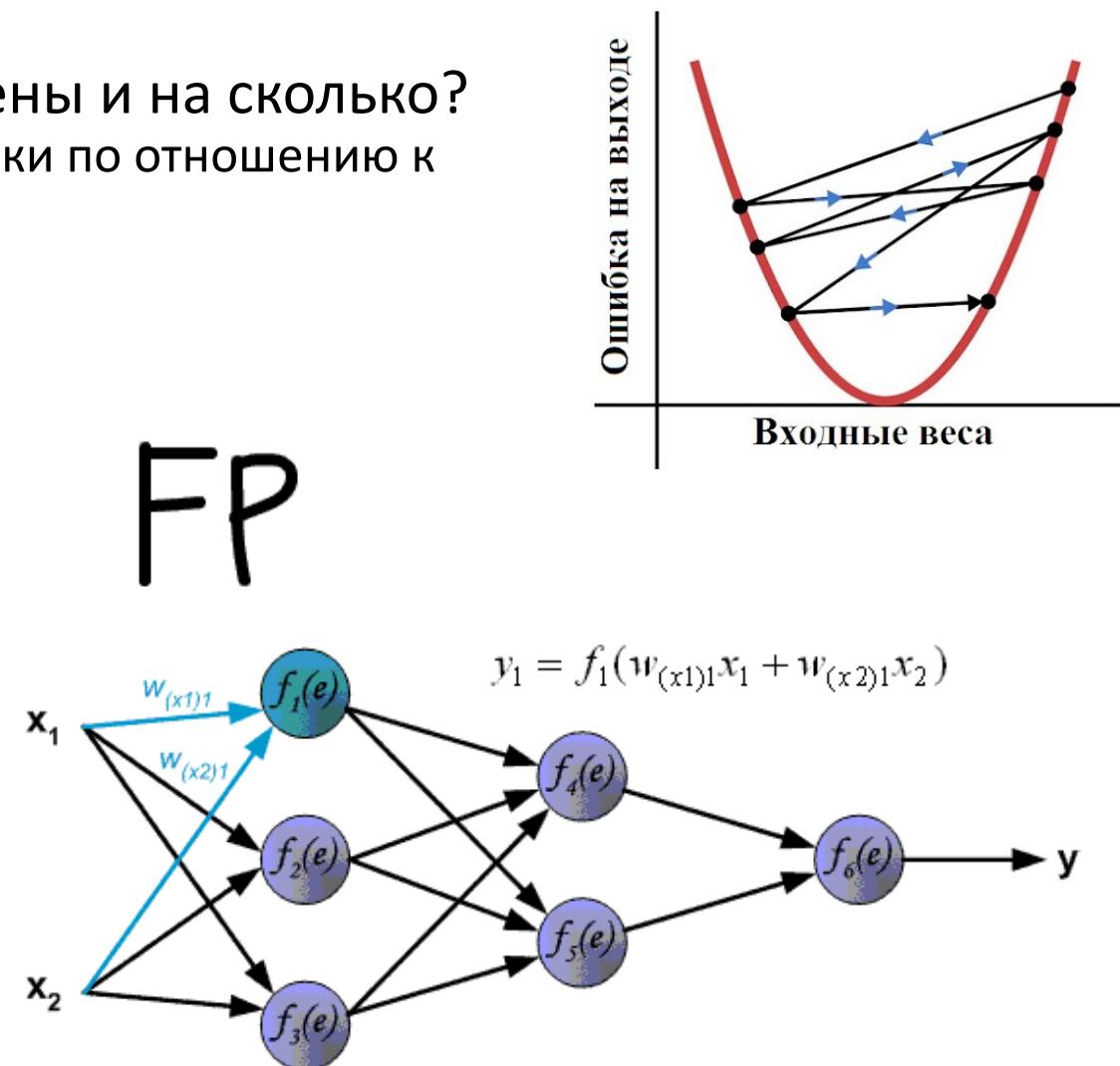
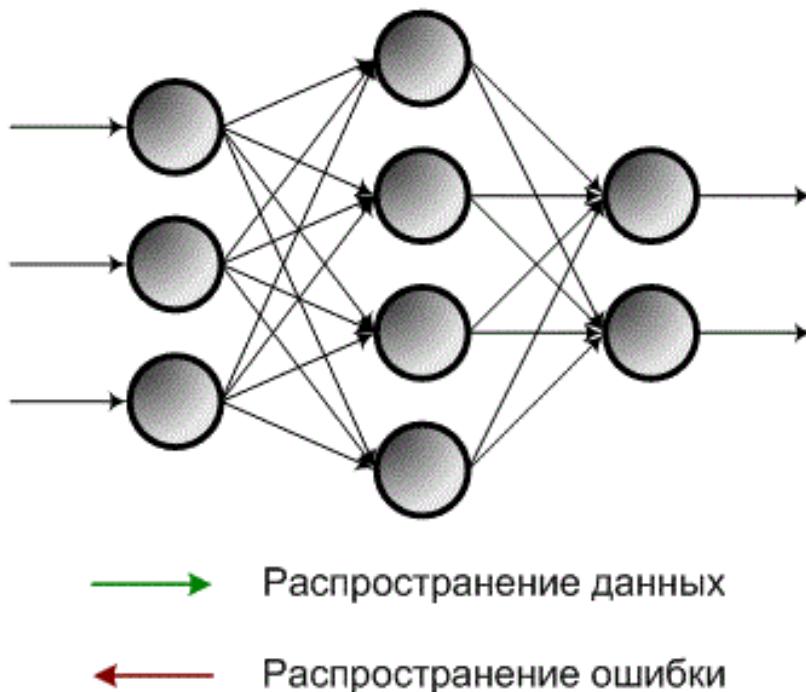
Градиентный спуск

- Мы не знаем на самом деле красивый график функции потерь
- Как будто спускаемся по оврагу с закрытыми глазами
- Поэтому считаем градиент в точке и обновляем веса

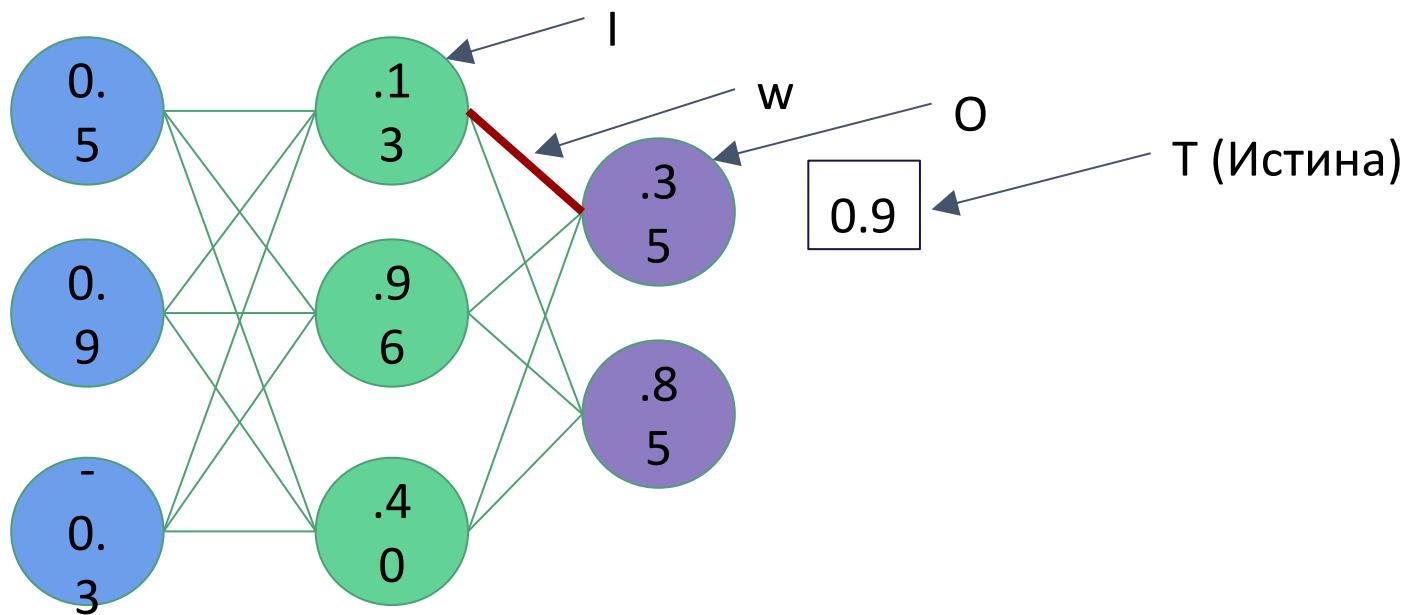


*Обратное распространение ошибки

- Вопрос: какие веса должны быть обновлены и на сколько?
 - Подсказка: используйте производную от ошибки по отношению к весу, чтобы найти объем «вины»



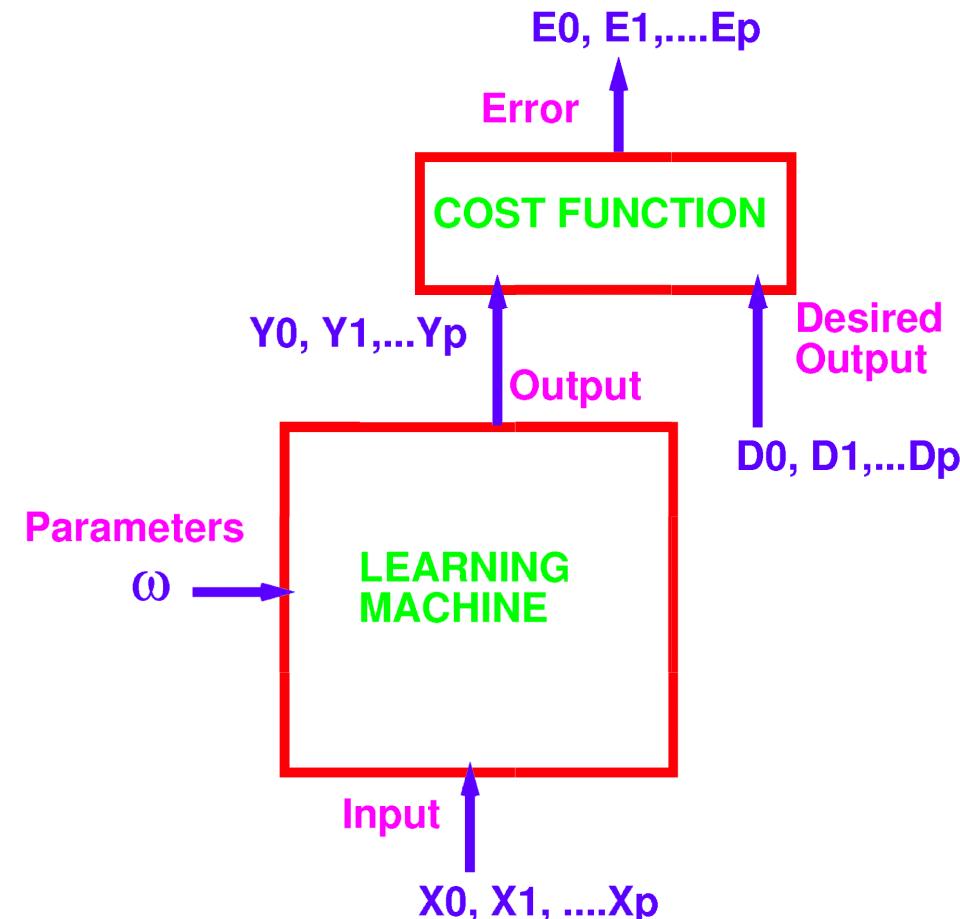
Пример обратного распространения



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

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

Основные понятия, терминология и обозначения

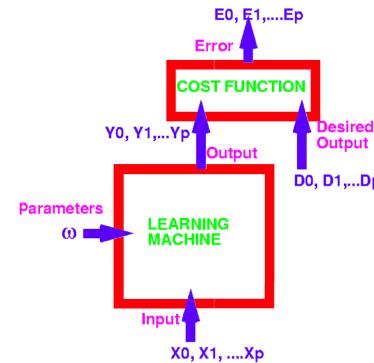


Average Error: $E = \frac{1}{p} \sum E_k$

Обучение градиентным спуском

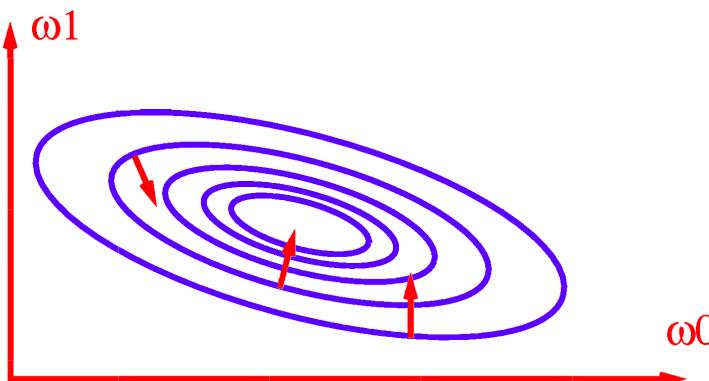
Average Error:

$$E(\omega) = \frac{1}{p} \sum E_k(\omega)$$

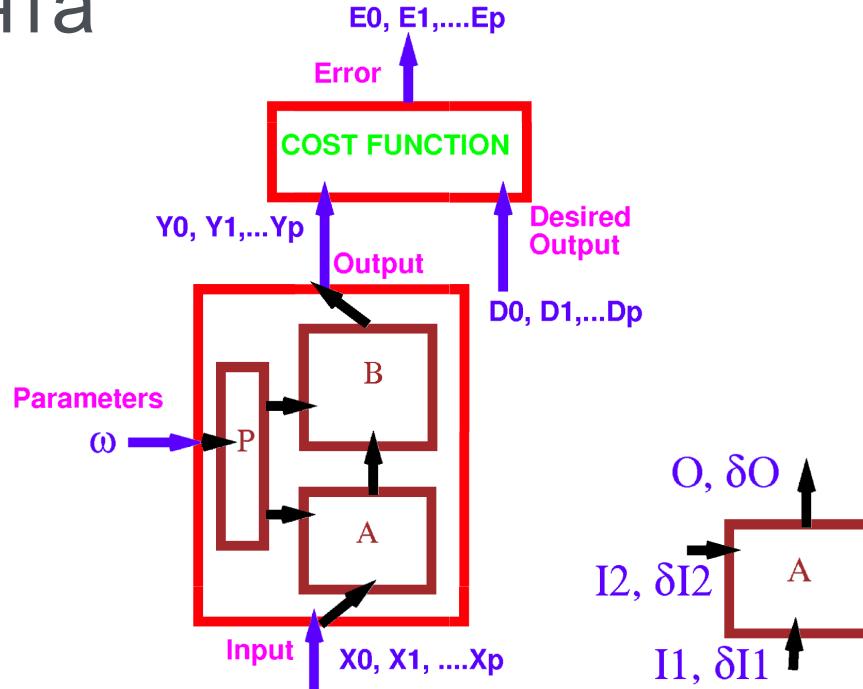


Gradient Descent:

$$\omega(\tau+1) = \omega(\tau) - \eta \frac{\partial E}{\partial \omega}$$

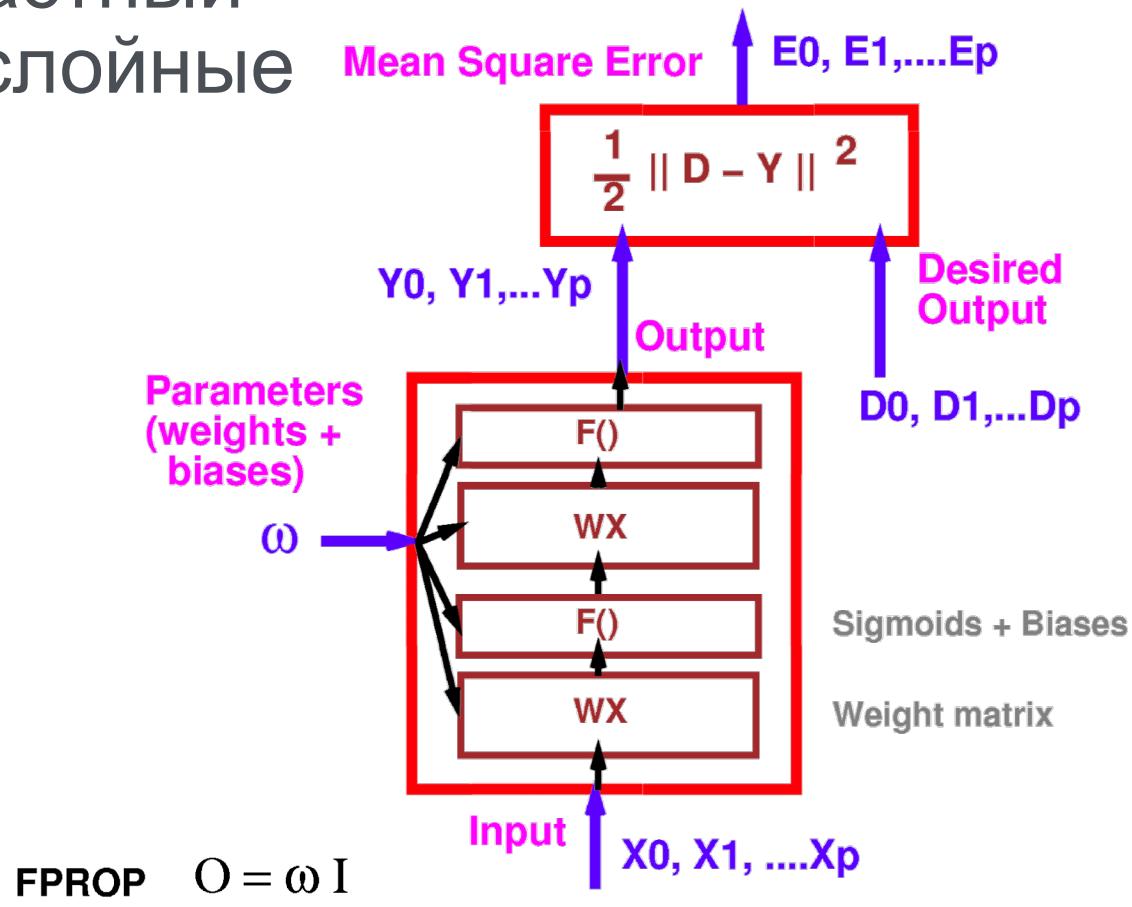


Вычисление градиента в обратном распространении



- Обучаемая модель состоит из модулей (например слои)
- Каждый модуль делает две процедуры:
 1. Вычисляет свои выходные значения из своих входных значений (пр. распр.)
$$O = A(I_1, I_2)$$
 2. Вычисляет градиентные векторы входных значений по градиентным векторам выходных значений (обр. распр.)
$$\delta I_1 = \frac{\partial A}{\partial I_1} \delta O \quad \delta I_2 = \frac{\partial A}{\partial I_2} \delta O$$

Интересный частный случай: многослойные сети



– Матрицы весов:

$$\text{FPROP} \quad O = f(I+B)$$

– Сигмоиды + отступ(bias):

$$\text{BPROP} \quad \delta I = f'(I+B) \delta O ; \quad \delta B = \delta I$$

Стохастичное обновление

- Стохастический градиент

$$\omega(\tau+1) = \omega(\tau) - \eta \frac{\partial E\tau}{\partial \omega}$$

```
Repeat {
    for all examples in training set {
        forward prop      // compute output
        backward prop     // compute gradients
        update parameters }}
```

- Параметры обновляются после показа каждого примера

- Полный градиент

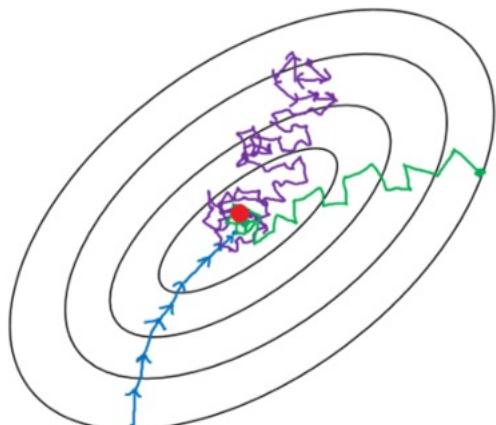
$$\omega(\rho+1) = \omega(\rho) - \eta \frac{\partial E}{\partial \omega}$$

```
Repeat {
    for all examples in training set {
        forward prop      // compute output
        backward prop     // compute gradients
        accumulate gradient }
    update parameters }
```

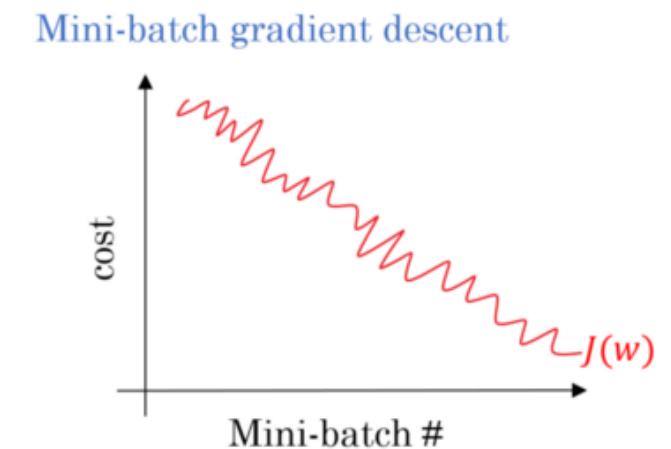
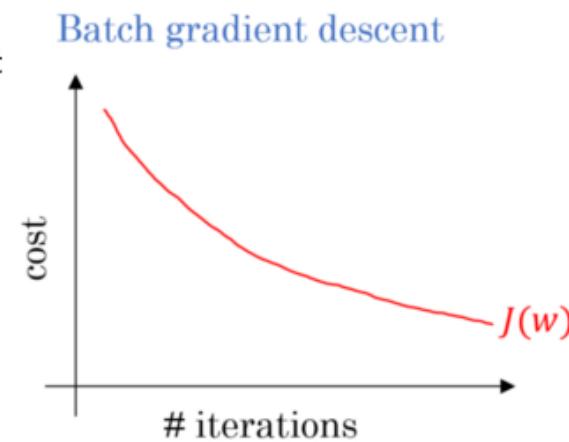
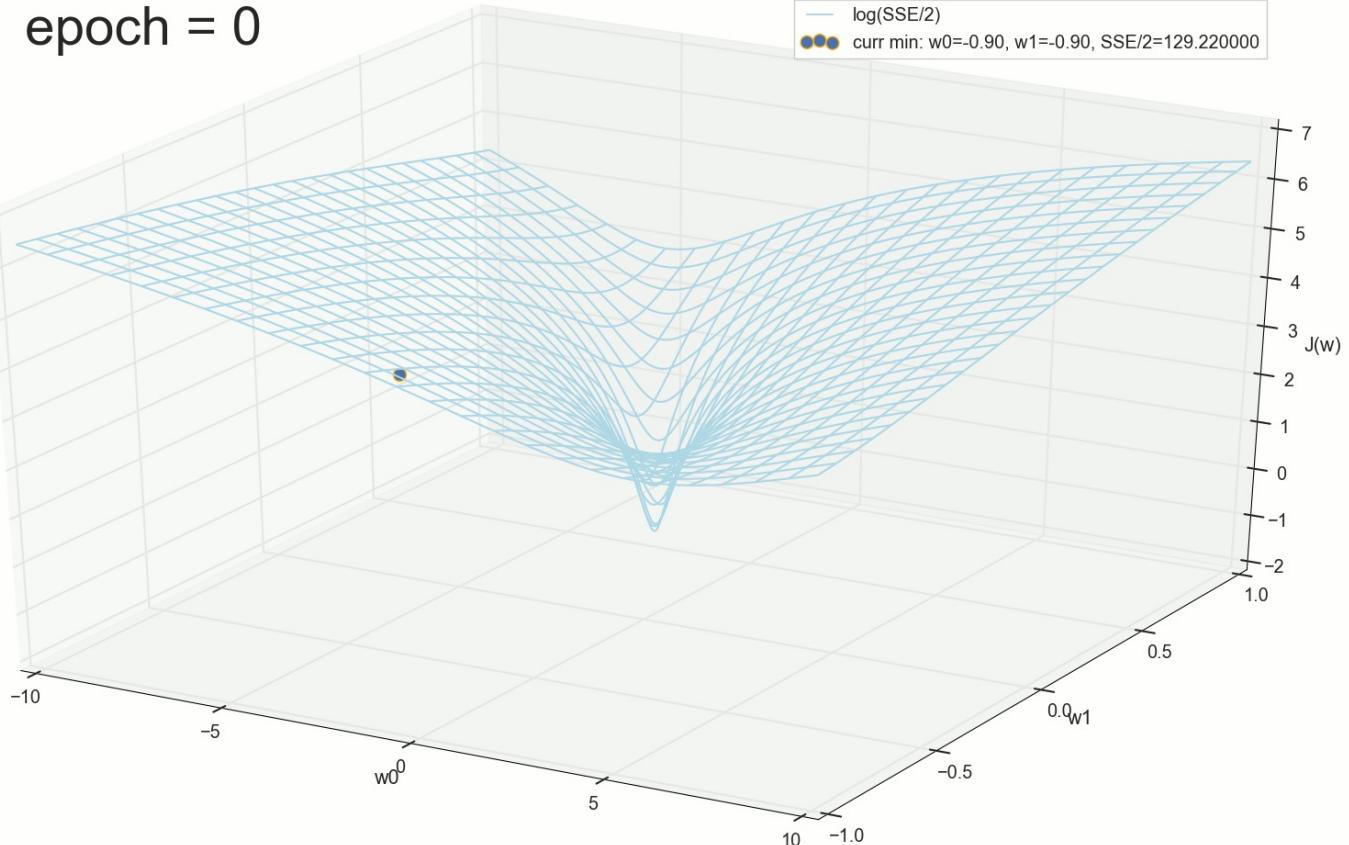
- Перед обновлением параметра из всего обучающего набора накапливаются градиенты

*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

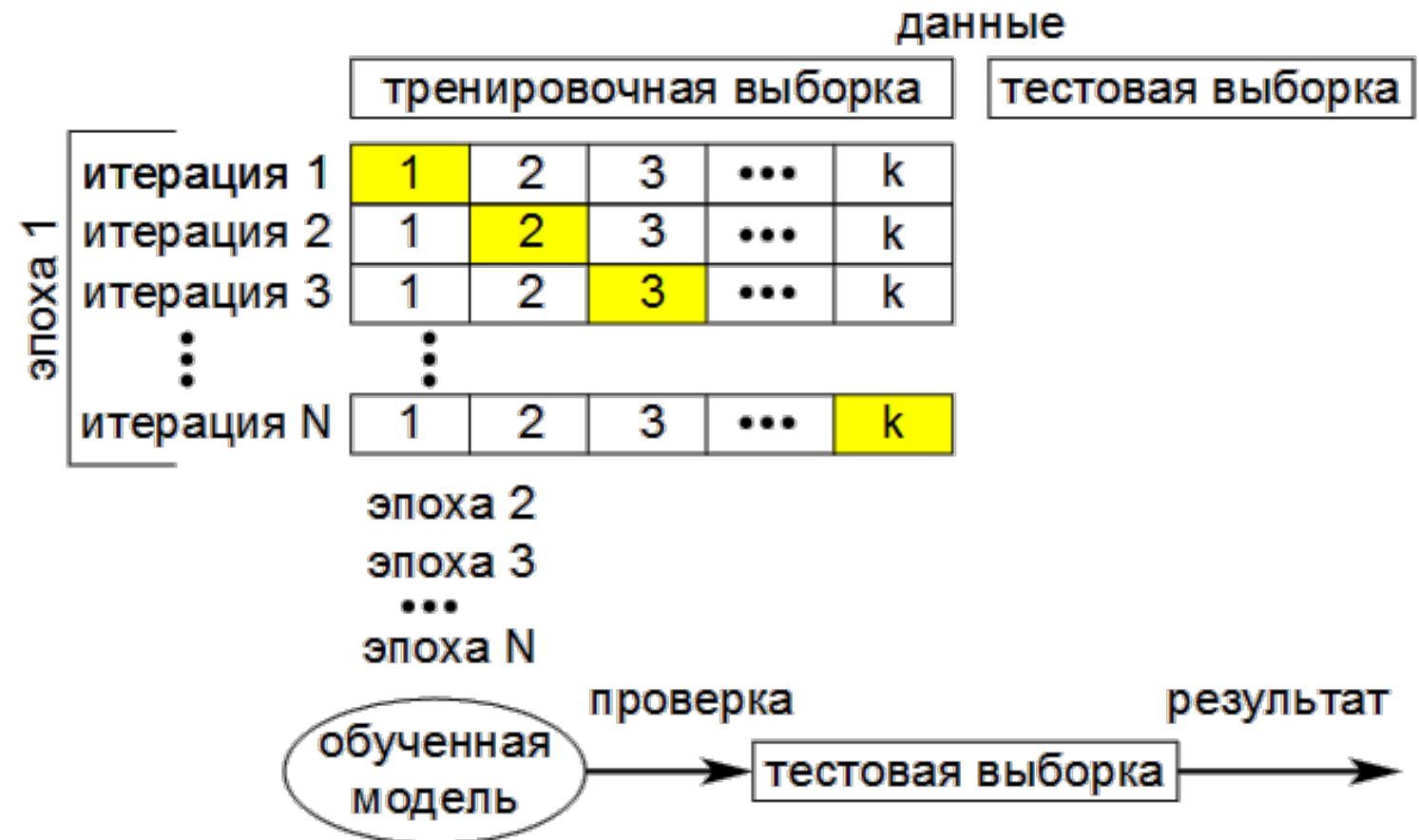


— Batch gradient descent
— Mini-batch gradient Descent
— Stochastic gradient descent



*Итерации и эпохи

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

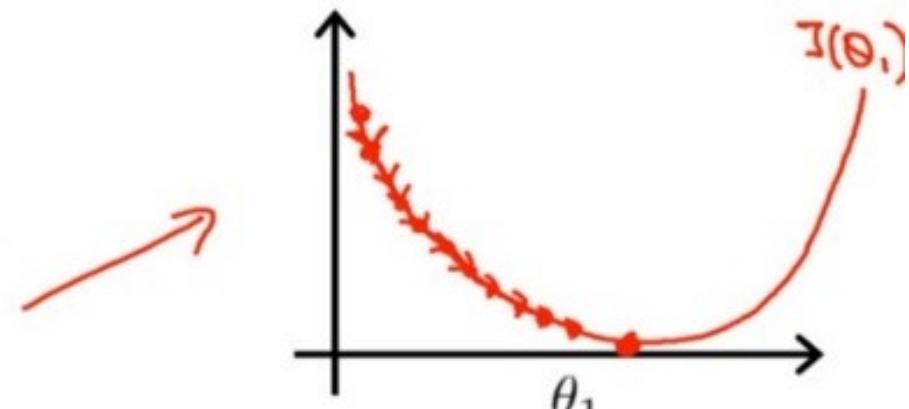


*Скорость обучения

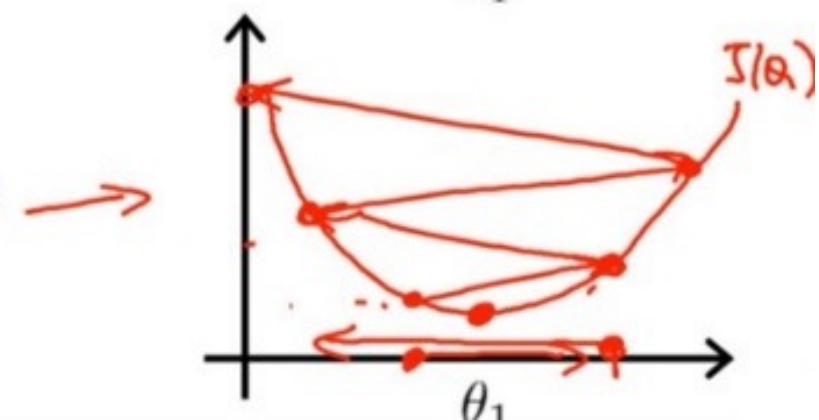
- Параметры (веса) сети меняются при обучении
- Гиперпараметры – нет. Управляем обучением

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

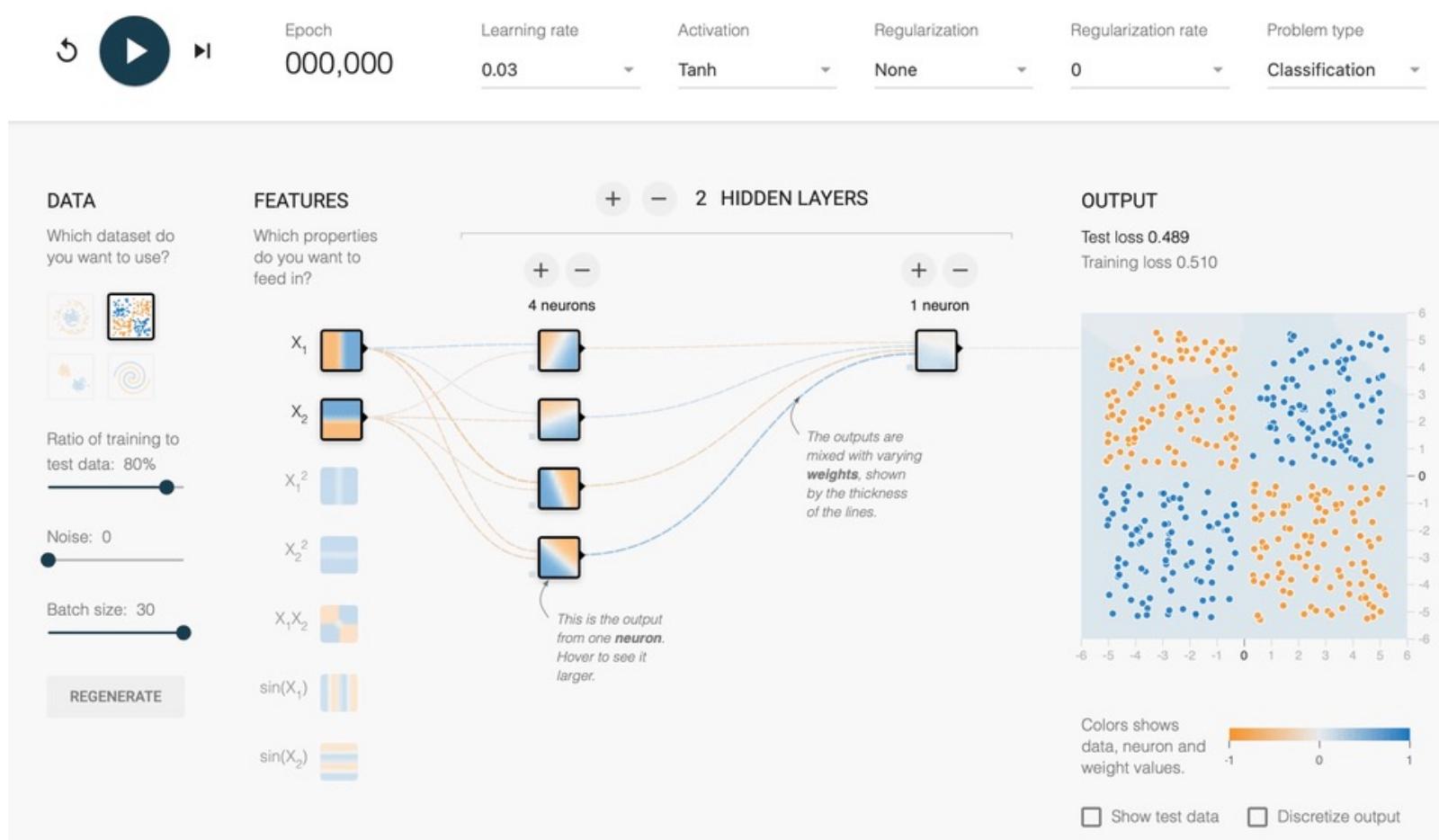
Если α слишком маленькая, то градиентный спуск может быть слишком медленным.



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



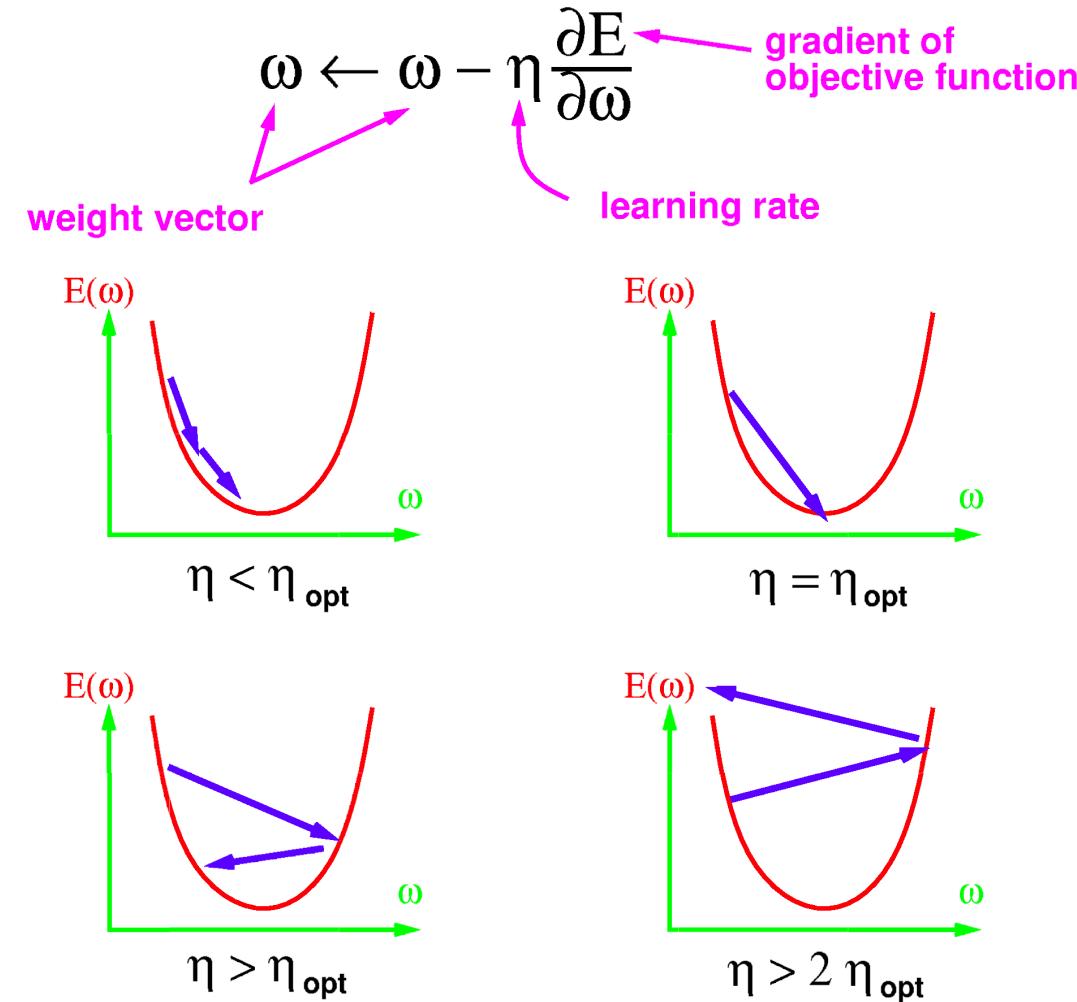
*Визуализация обучения



<https://playground.tensorflow.org>

Немного теории

Градиентный спуск в одном измерении



Оптимальный шаг обучения в 1D

Weight change:

$$\Delta\omega = \eta \frac{\partial E}{\partial \omega}$$

Assuming E is quadratic:

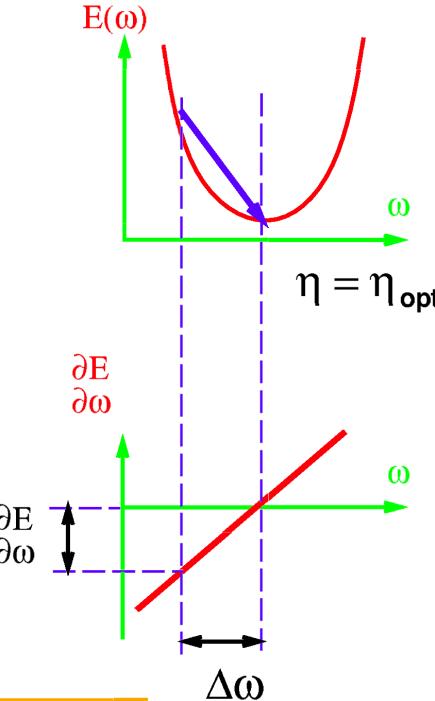
$$\frac{\partial^2 E}{\partial \omega^2} \Delta\omega = \frac{\partial E}{\partial \omega}$$

Optimal Learning Rate

$$\eta_{\text{opt}} = \left(\frac{\partial^2 E}{\partial \omega^2} \right)^{-1}$$

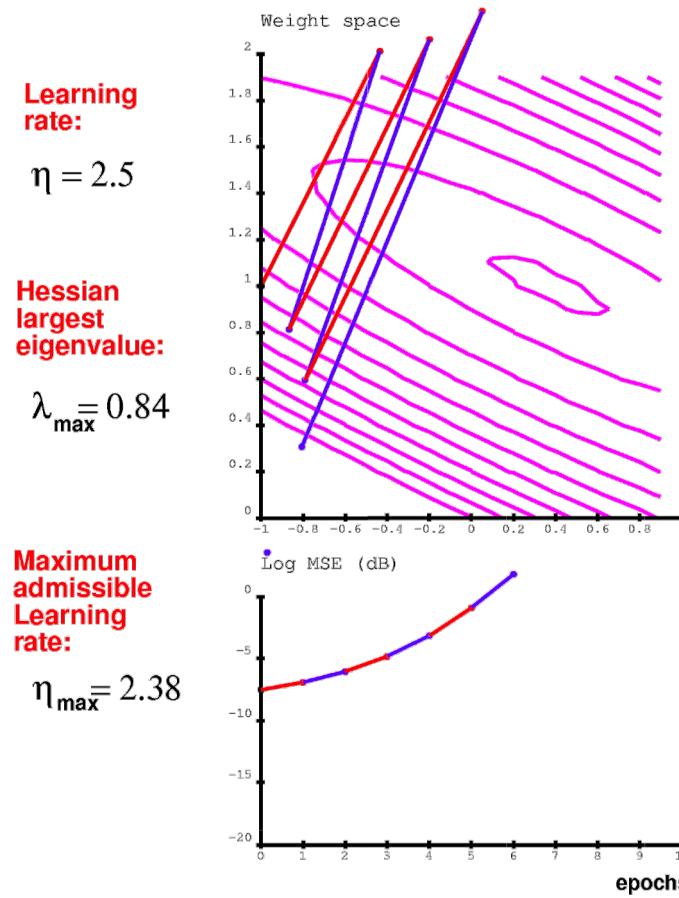
Maximum Learning Rate

$$\eta_{\text{max}} = 2 \eta_{\text{opt}}$$



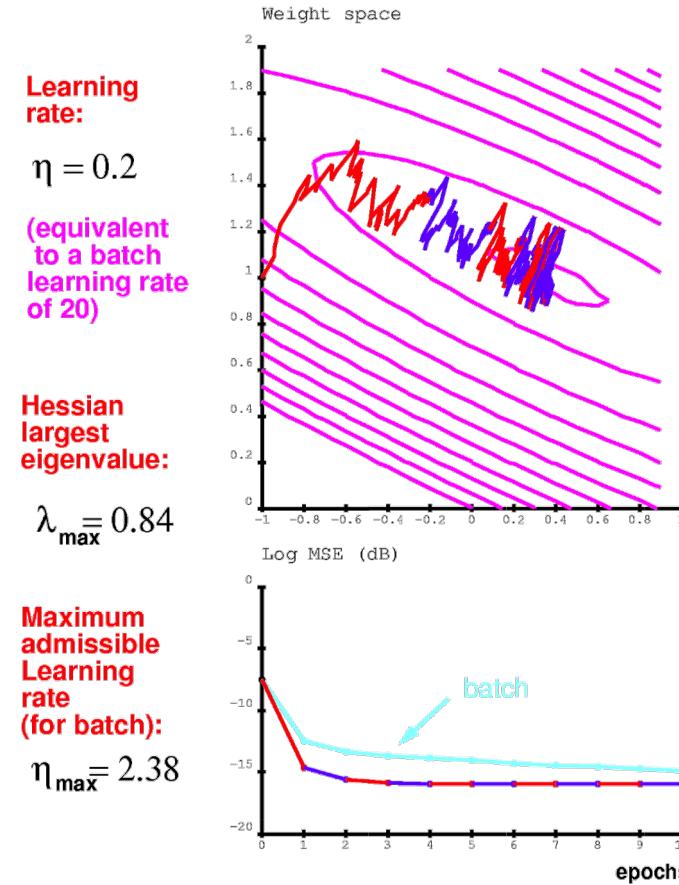
Пакетный градиентный спуск

- набор данных: набор-1 (100 примеров, 2 гауссова распределения) сеть: 1 линейный нейрон, 2 входа, 1 выход. 2 веса, 1 отступ.



Стохастический градиентный спуск

- набор данных: набор-1 (100 примеров, 2 гауссова распределения) сеть: 1 линейный нейрон, 2 входа, 1 выход. 2 веса, 1 отступ.



*Стохастическое vs пакетное обновление

- Стохастическое обновление обычно НАМНОГО быстрее, чем пакетное обновление. Особенно на больших избыточных наборах данных.
- Вот почему:
 - Представьте что у вас есть обучающий набор из 1000 примеров.
 - Этот обучающий набор состоит из 10 экземпляров по 100 примеров.
- Пакет (батч): вычисление для одного обновления будет в 10 раз больше необходимого
- Стохастическое: будет использоваться избыточность в учебном наборе для получения преимущества обучения. Одна эпоха на большом наборе будет похожа на 10 эпох на меньшем наборе.
- Пакетное обновление будет КАК МИНИМУМ в 10 раз медленнее стохастичного
- В реальной жизни повторения редко происходят, но очень часто обучающие примеры очень избыточны (много примеров похожи друг на друга), что имеет тот же эффект.
- На практике нередки разницы скорости (на порядки) между пакетным и стохастическим обновлением.
- Маленькие пакеты могут использоваться без штрафа, если примеры в минибатче не слишком похожи.

*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

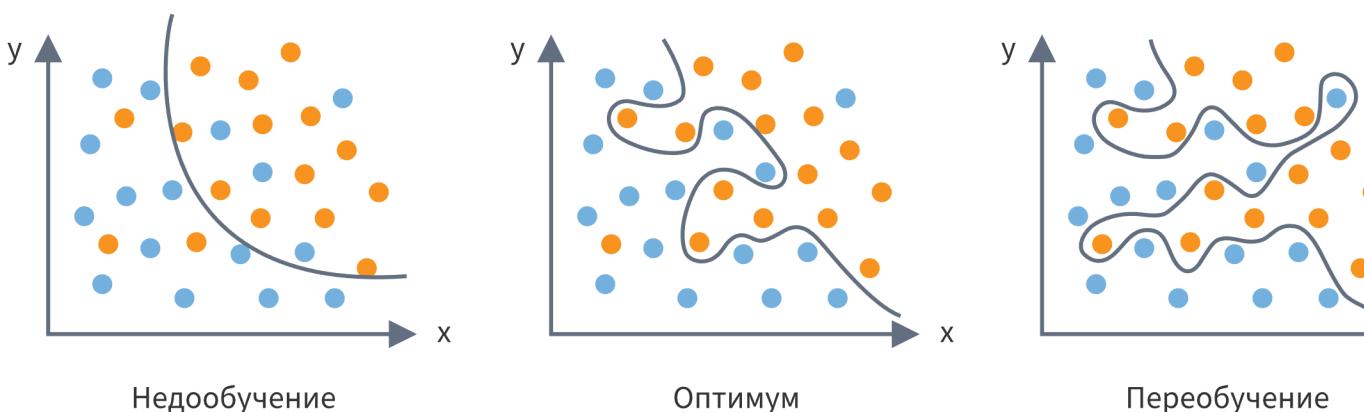
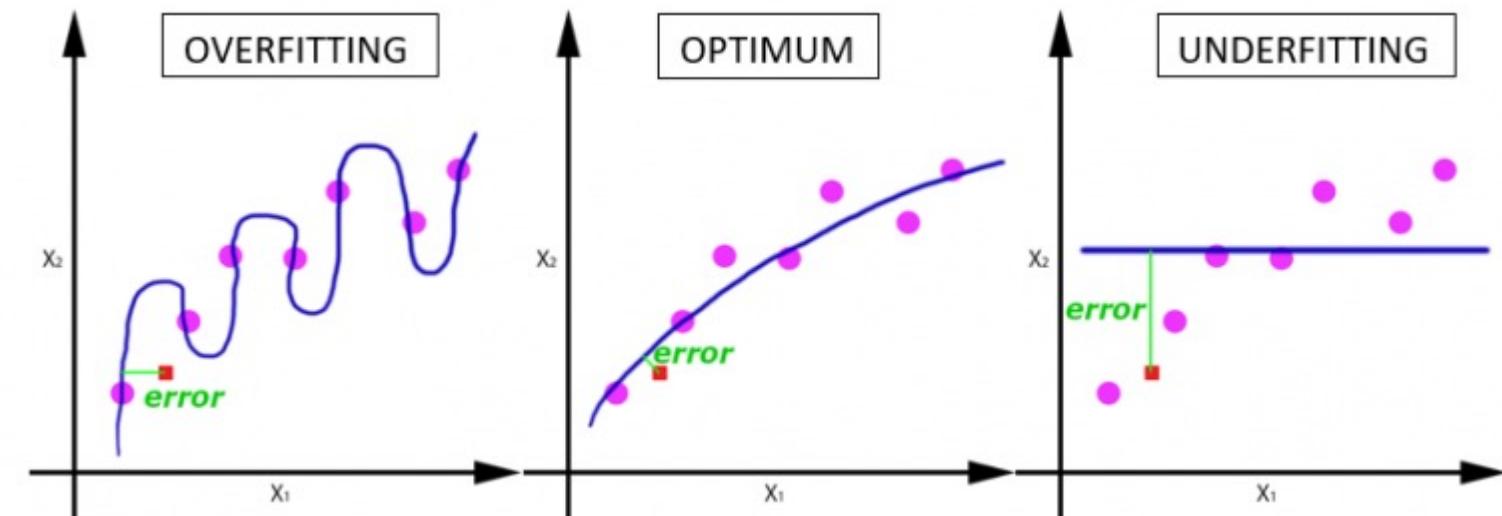
– Преимущества:

- 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).

*Точность, переобучение



- Переобучение при долгом обучении слишком сложной модели

