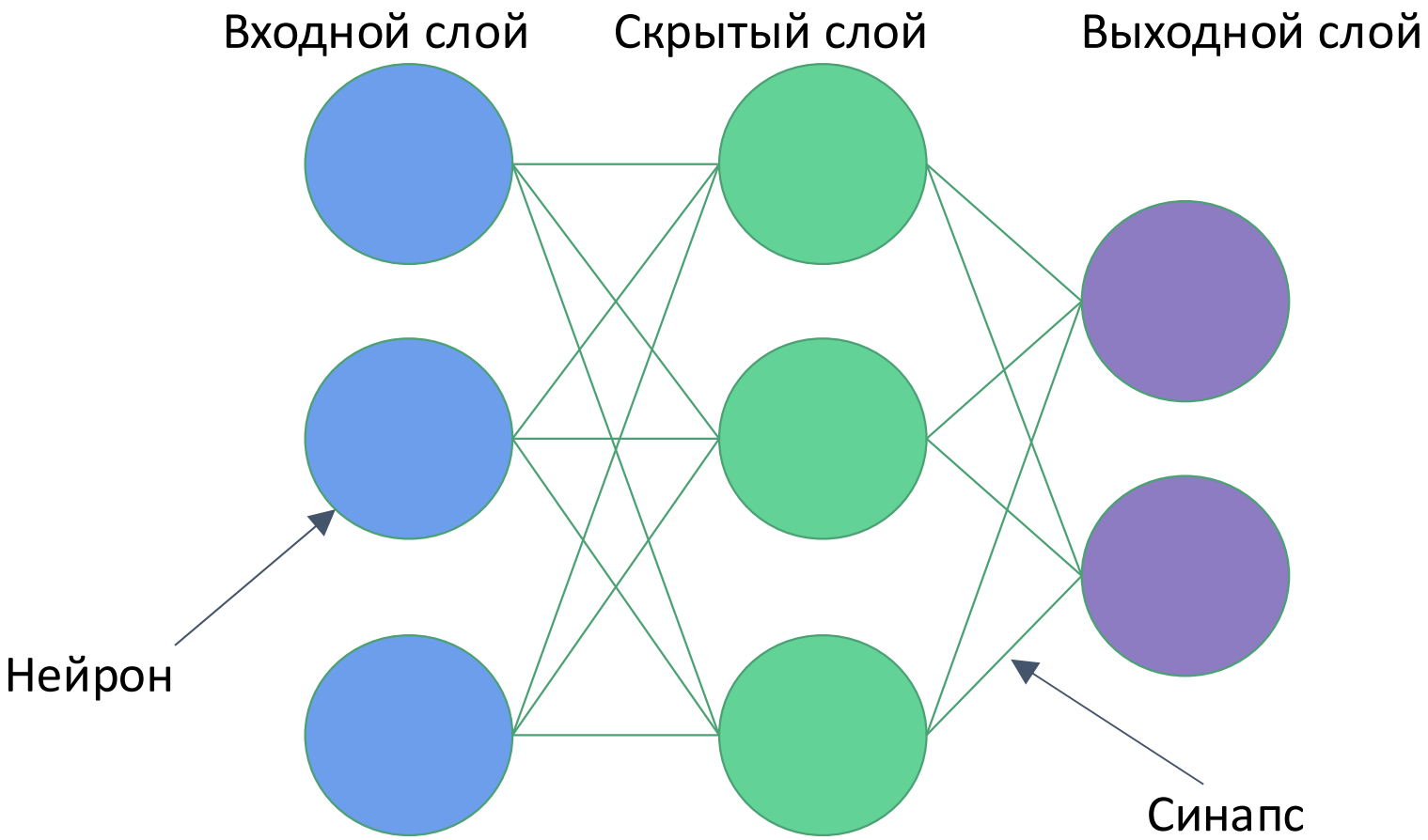
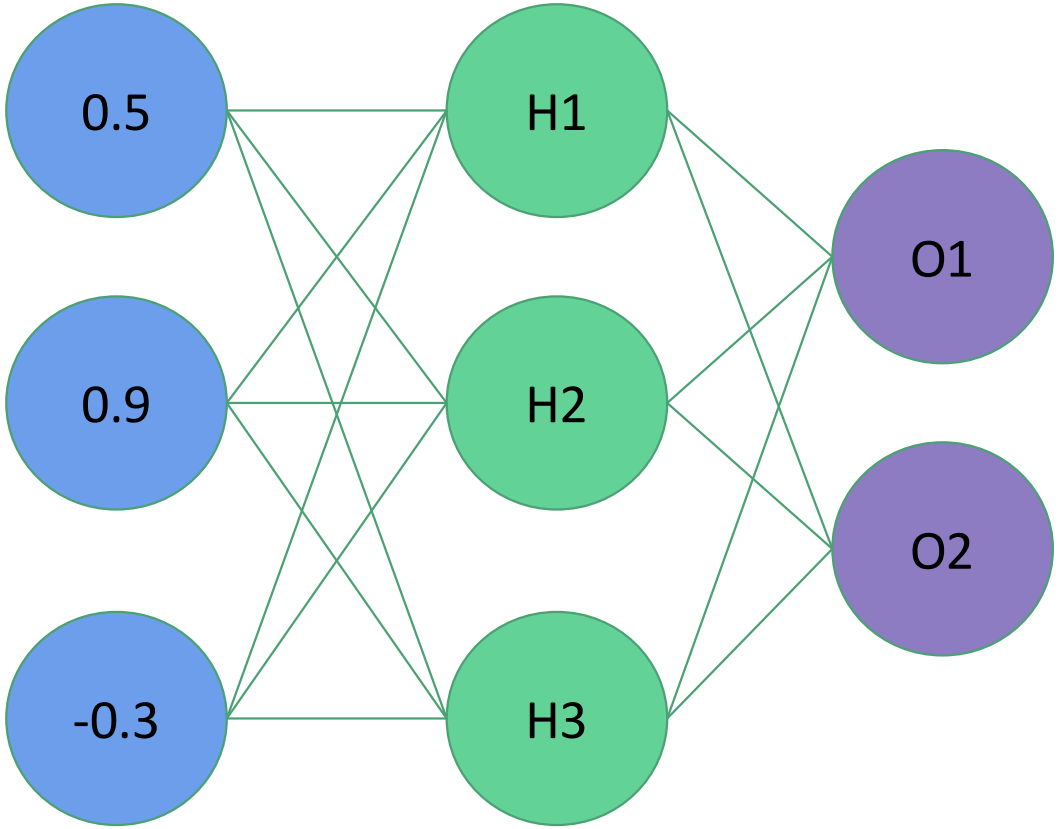
Архитектура нейронной сети



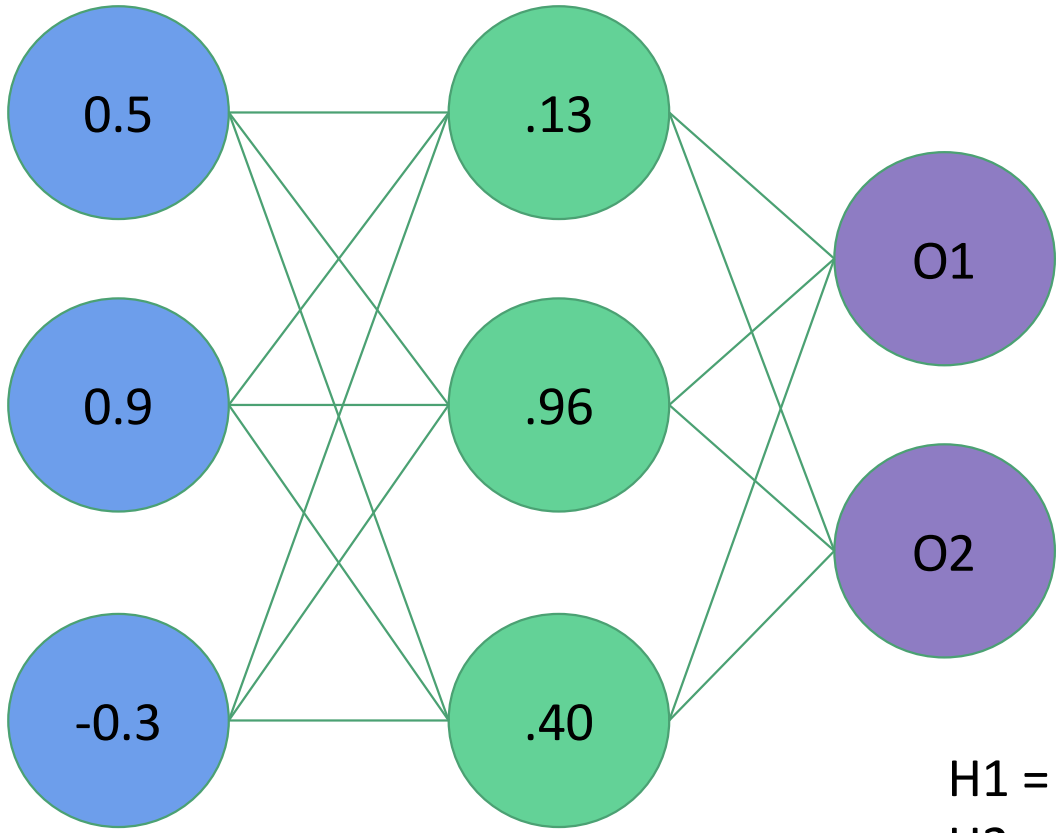
Вычисление прогноза (инференс)



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)

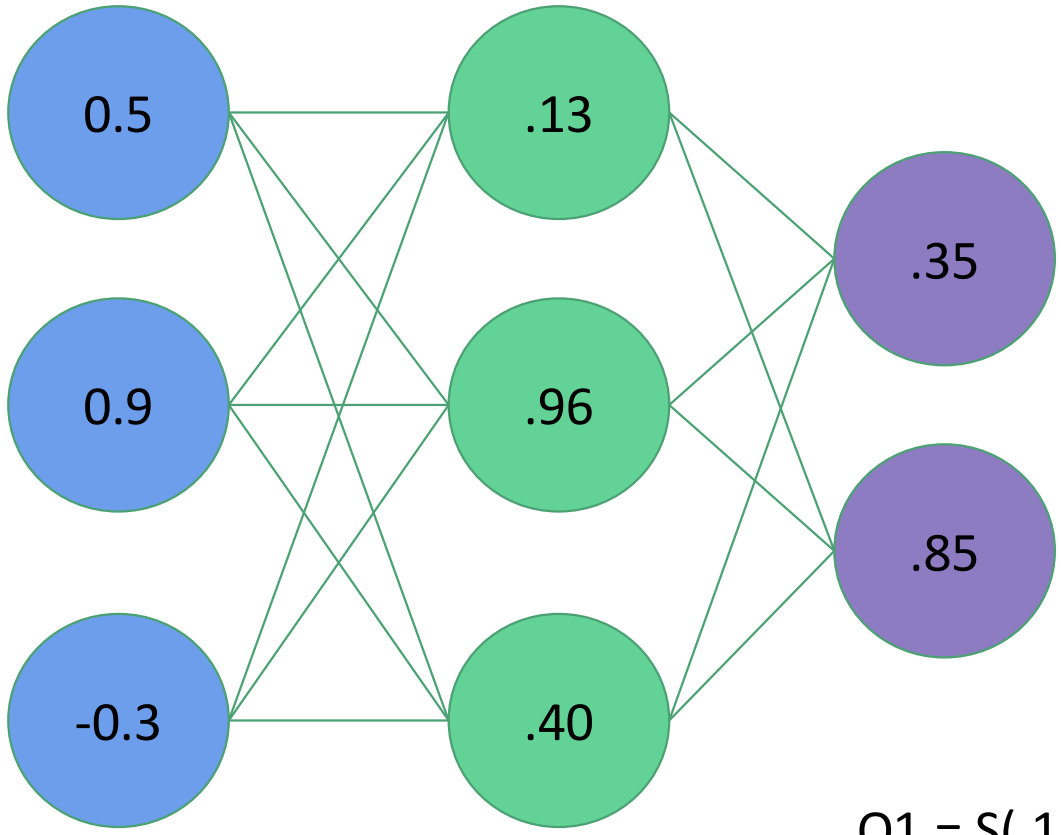


Веса H1 = (1.0, -2.0, 2.0)
Веса H2 = (2.0, 1.0, -4.0)
Веса H3 = (1.0, -1.0, 0.0)

Веса O1 = (-3.0, 1.0, -3.0)
Веса O2 = (0.0, 1.0, 2.0)

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)



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)

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

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

Вход

0.5
0.9
-0.3

$S($ * $) = S($ $) =$

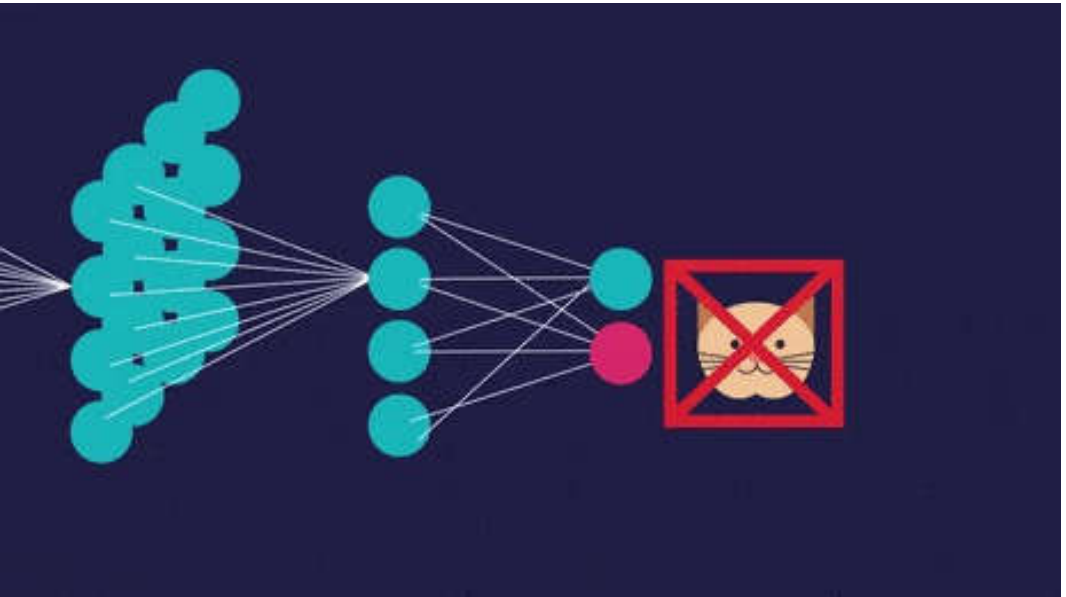
-1.9	3.1	-0.4

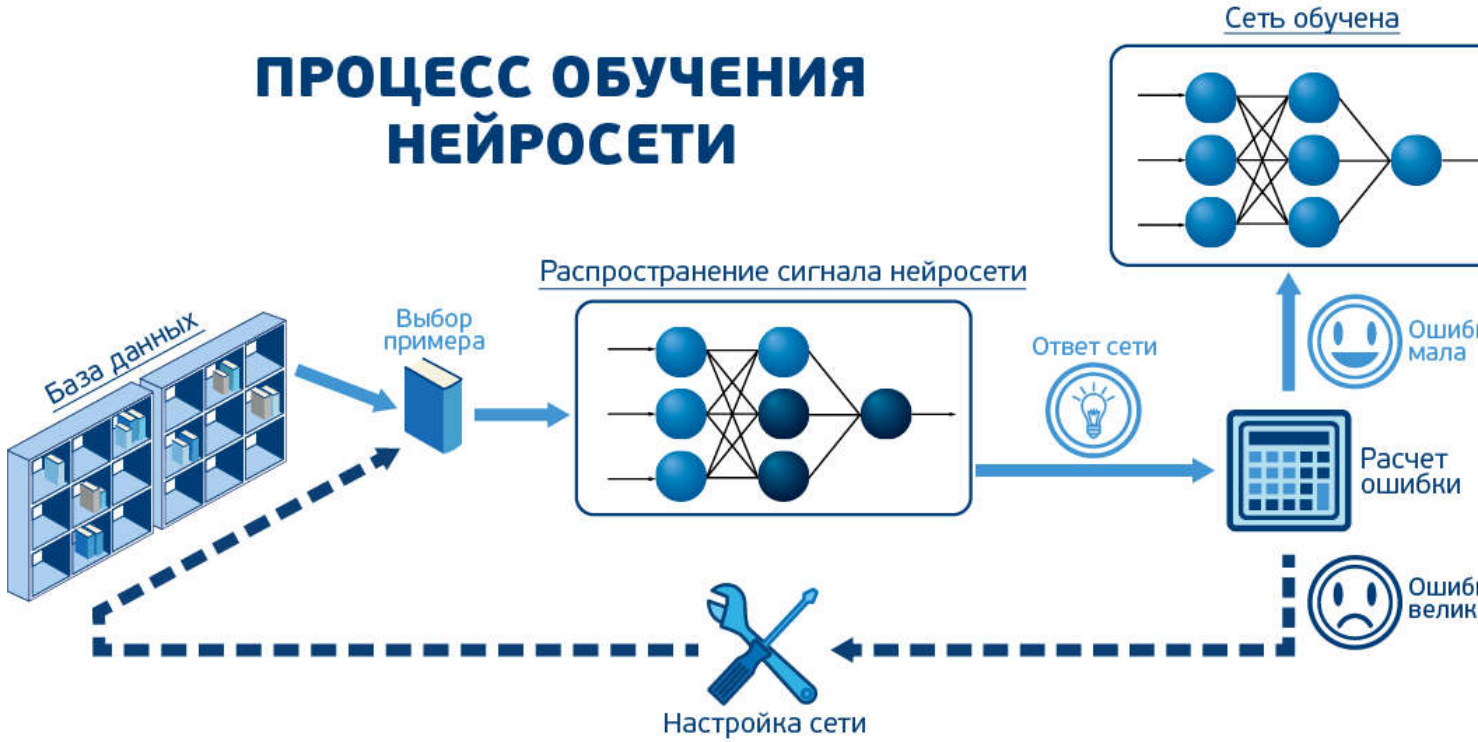
Значения скрытого слоя

.13	.96	0.4

Обучение нейросети

- Обучение нейросети
- Обучение с учителем





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

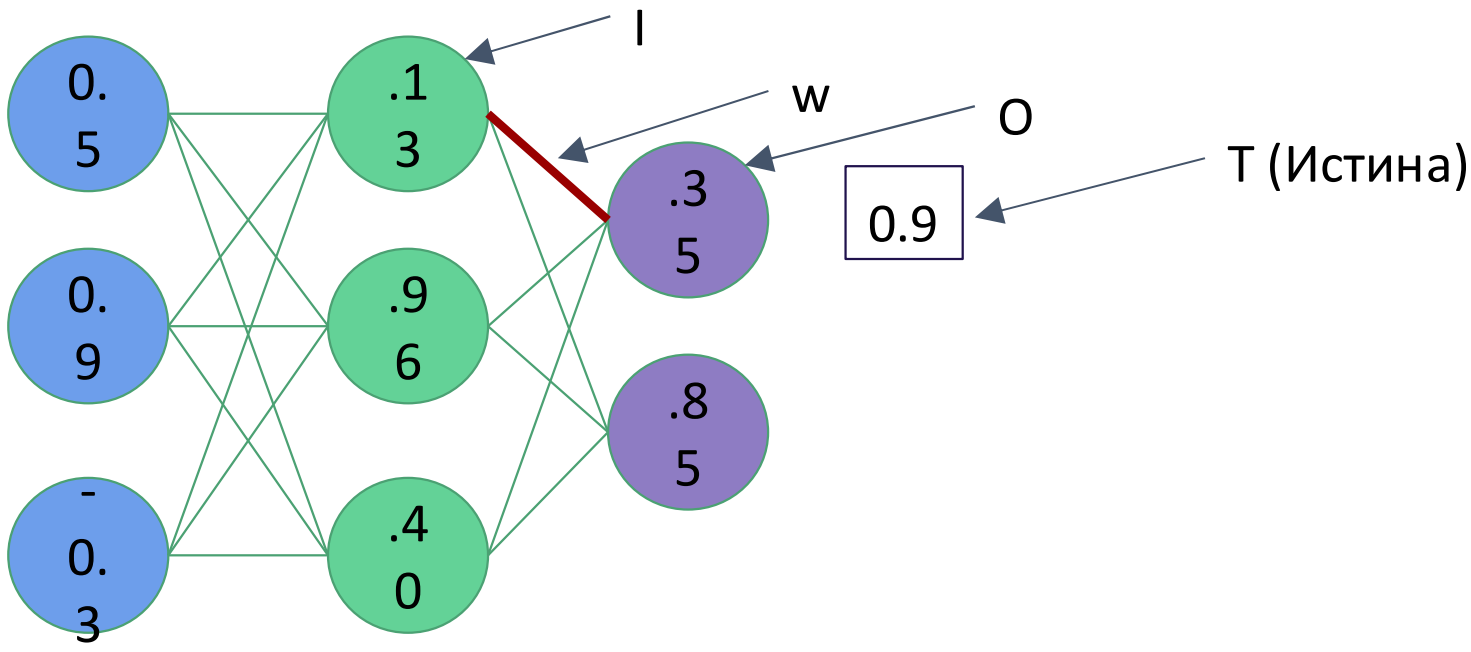
Обучение

```
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 двуэлементный 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()
```

Обратное распространение

- 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

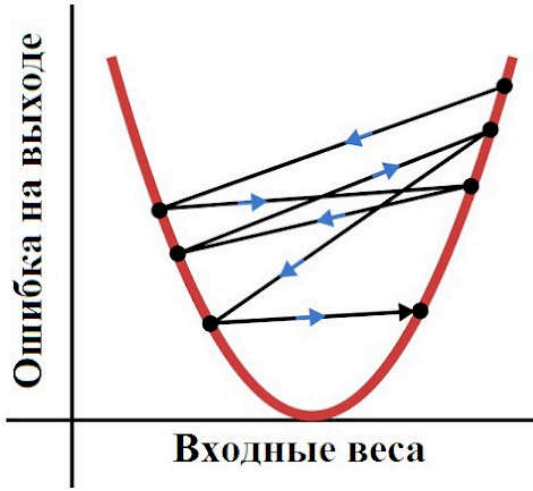


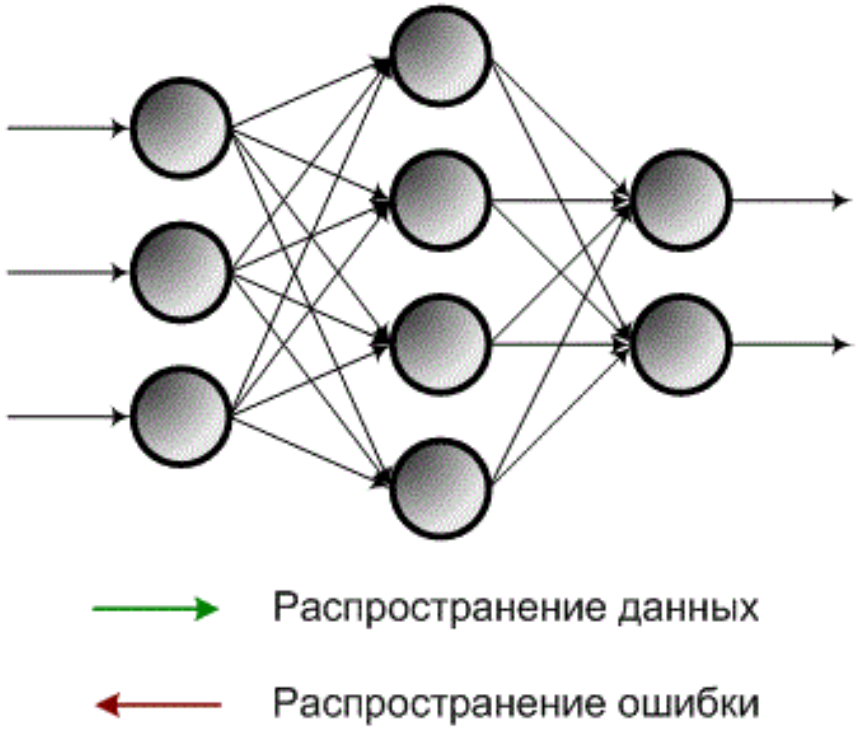
$$\frac{\partial E}{\partial w} = I \cdot (O - T) \cdot O \cdot (1 - O)$$

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

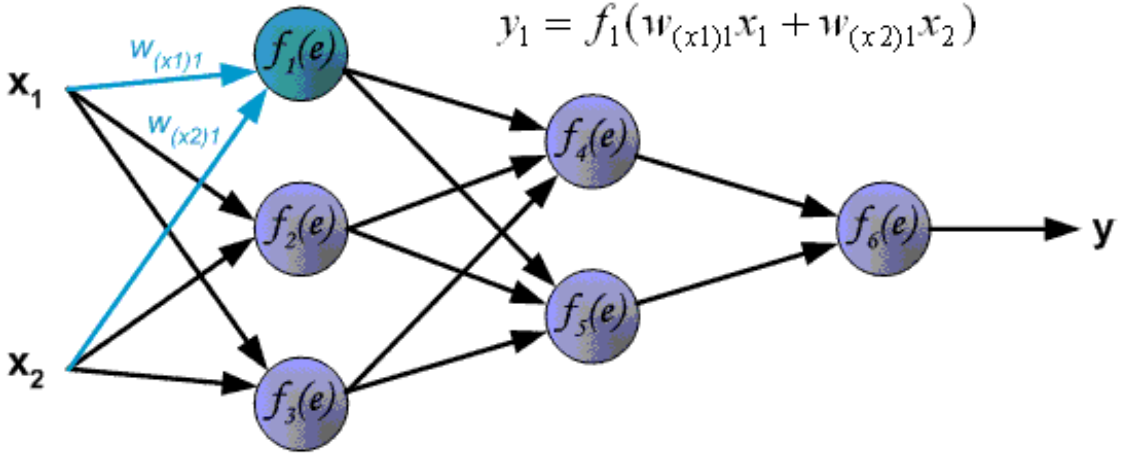
Обратное распространение ошибки

- Мы не знаем на самом деле красивый график функции потерь
- Поэтому считаем градиент и обновляем веса в одной точке
- Из ошибки функции потерь считаем градиент и обновляем веса



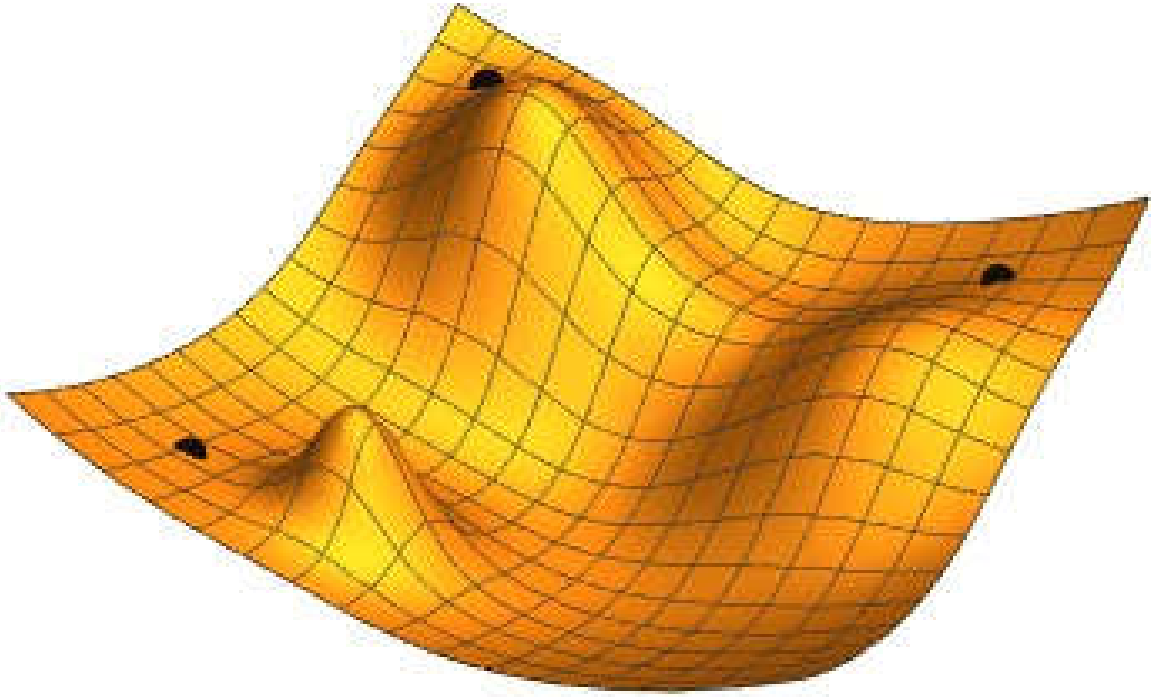


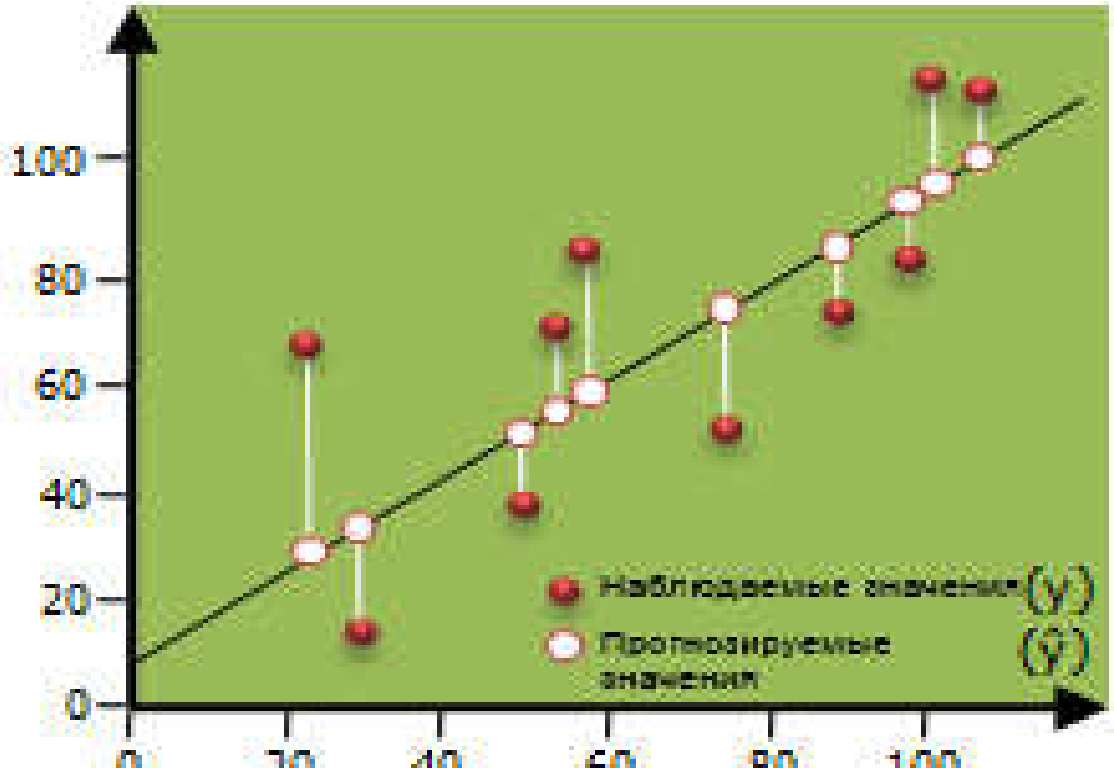
FP



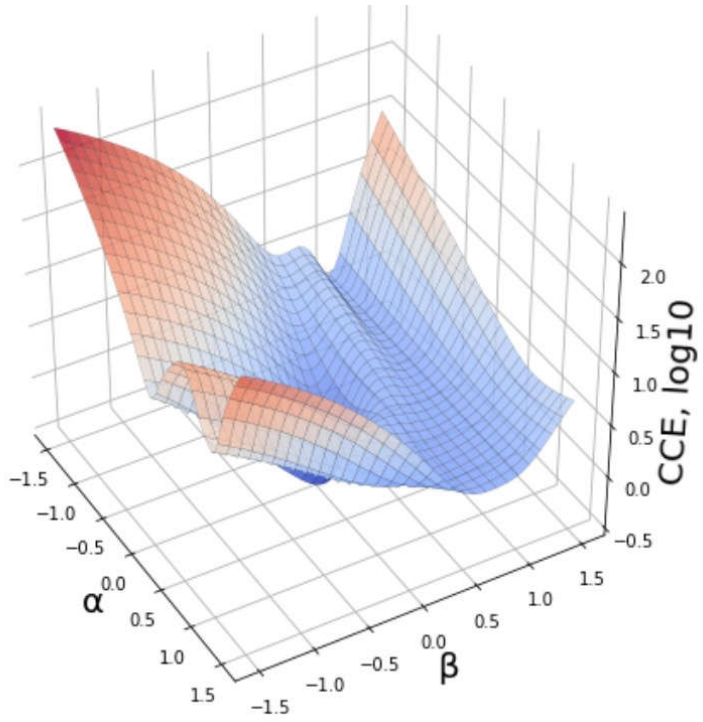
Функция потерь (loss)

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

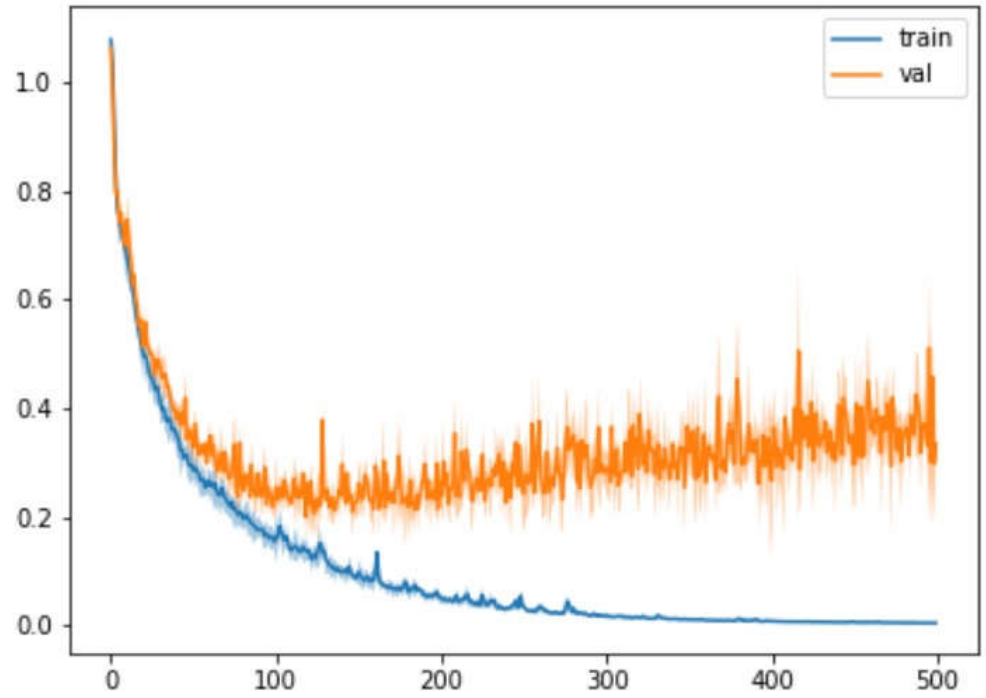




Функции потерь



Примеры из лабораторной



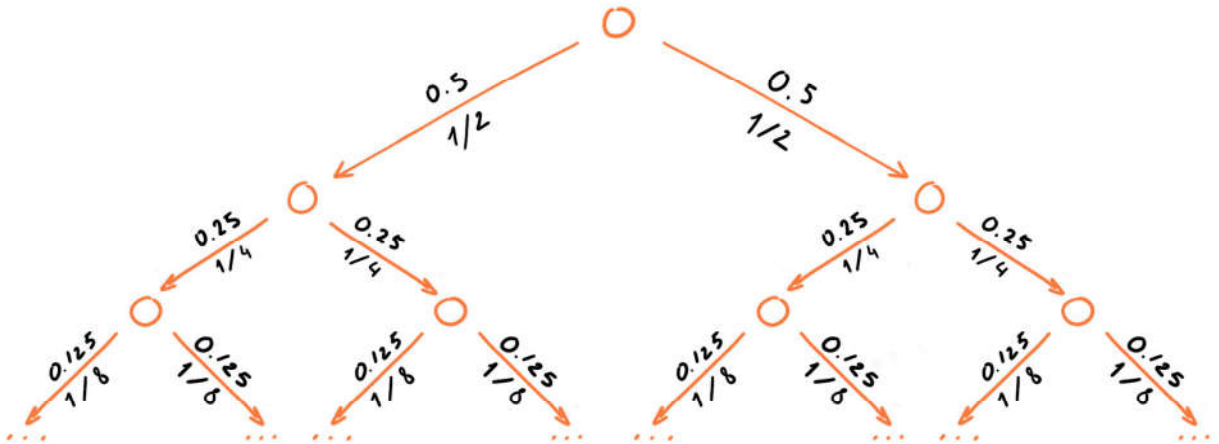

```
criterion = nn.CrossEntropyLoss()
```

```
# 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

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



$2^0 = 1 \quad \log_2(1) = 0$

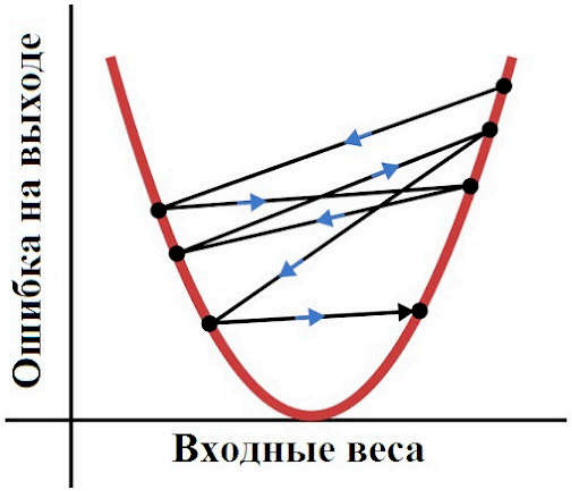
$2^1 = 2 \quad \log_2(2) = 1$

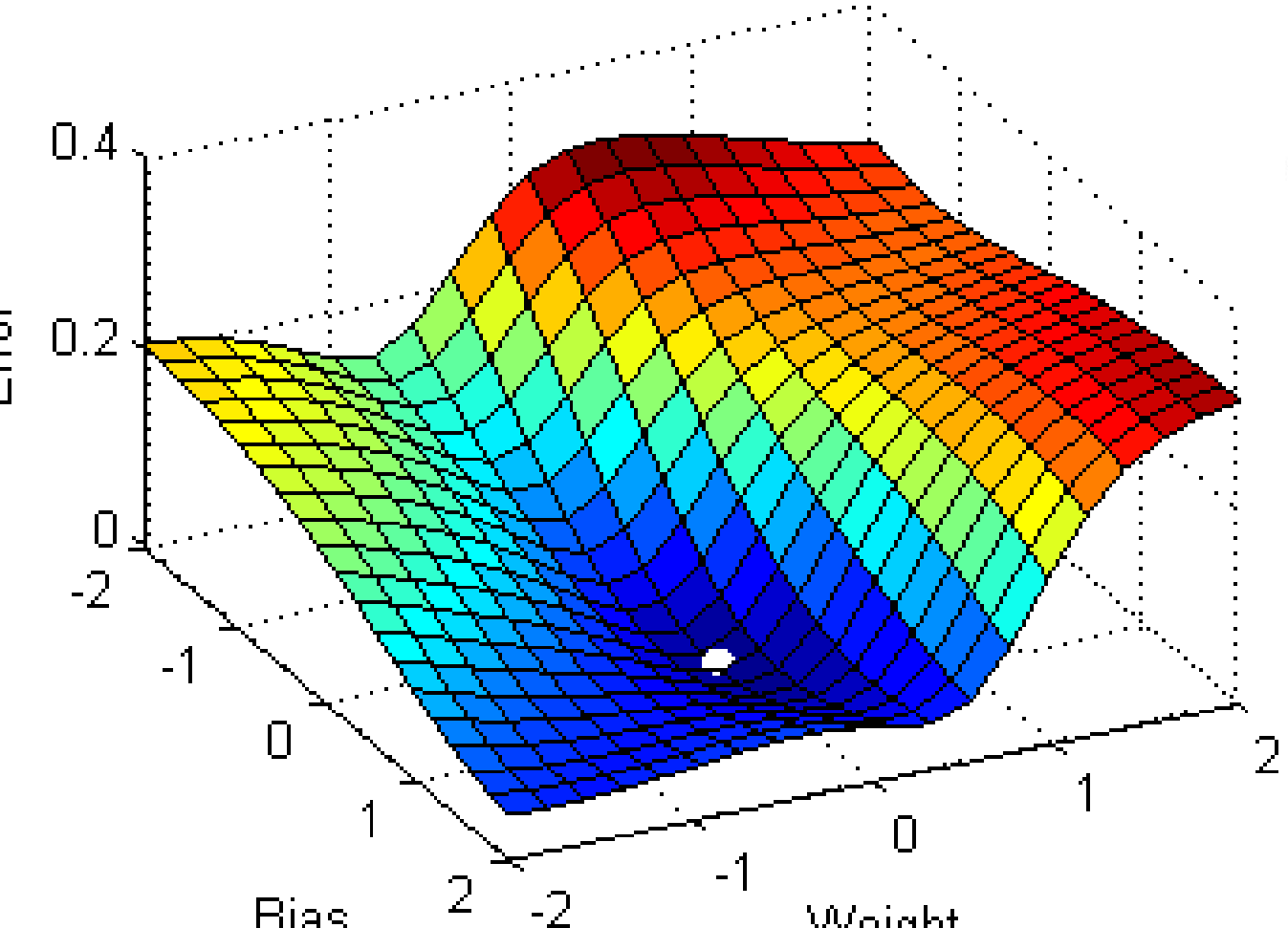
$2^2 = 4 \quad \log_2(4) = 2$

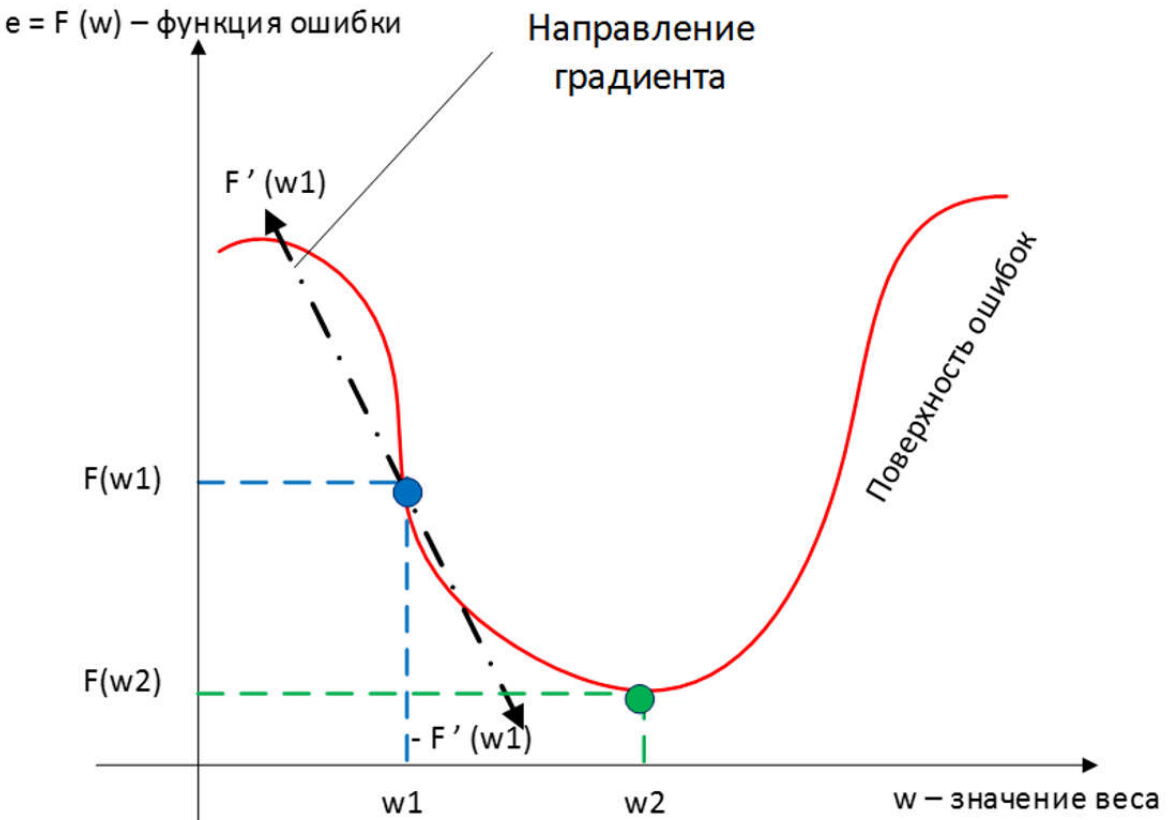
$2^3 = 8 \quad \log_2(8) = 3$

Градиентный спуск

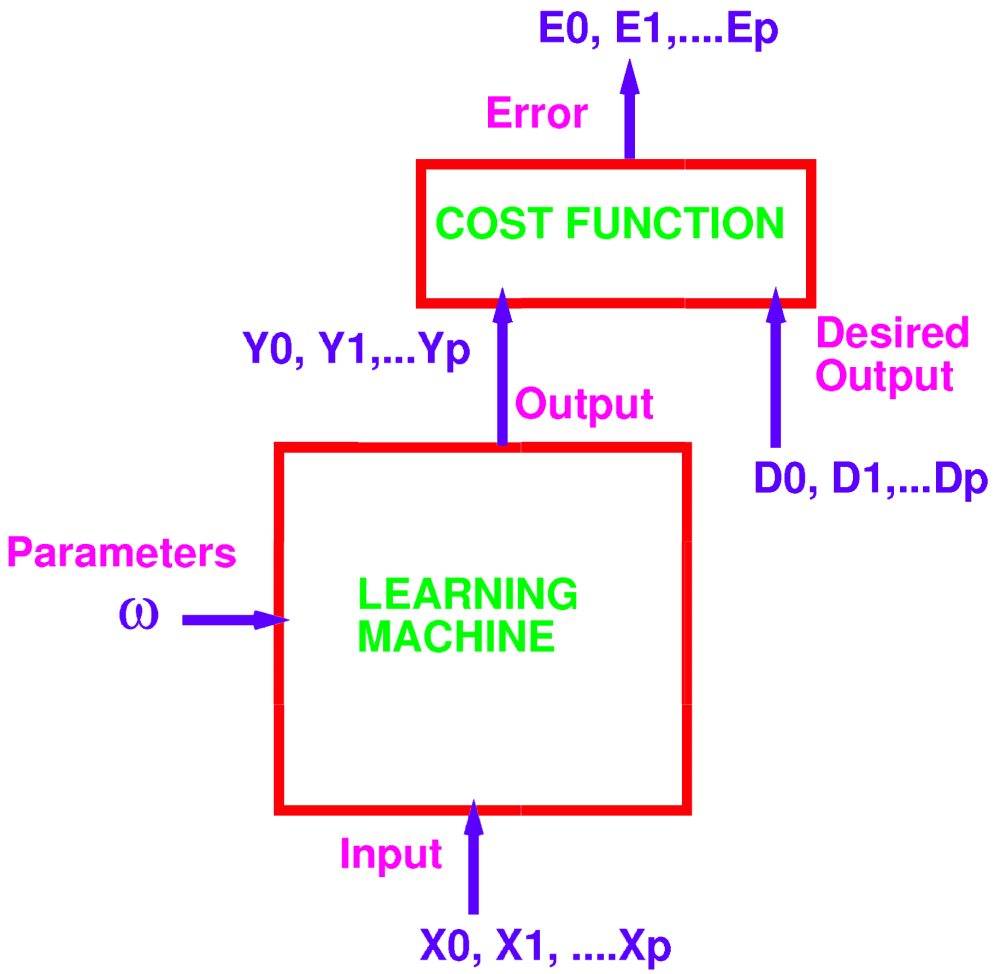
- Обучение по шагам с вычислением градиента







Основные понятия, терминология и обозначения

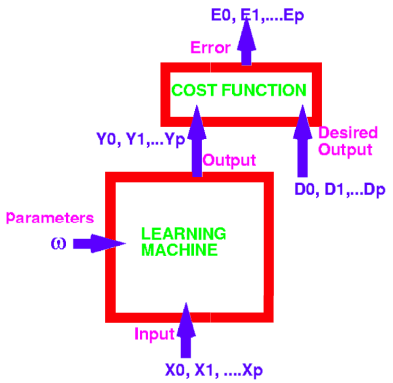


Average Error: $E = \frac{1}{p} \Sigma E\kappa$

Обучение градиентным спуском

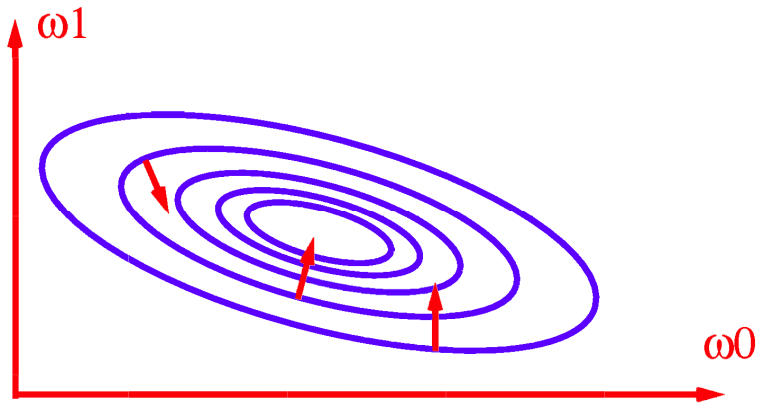
Average Error:

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

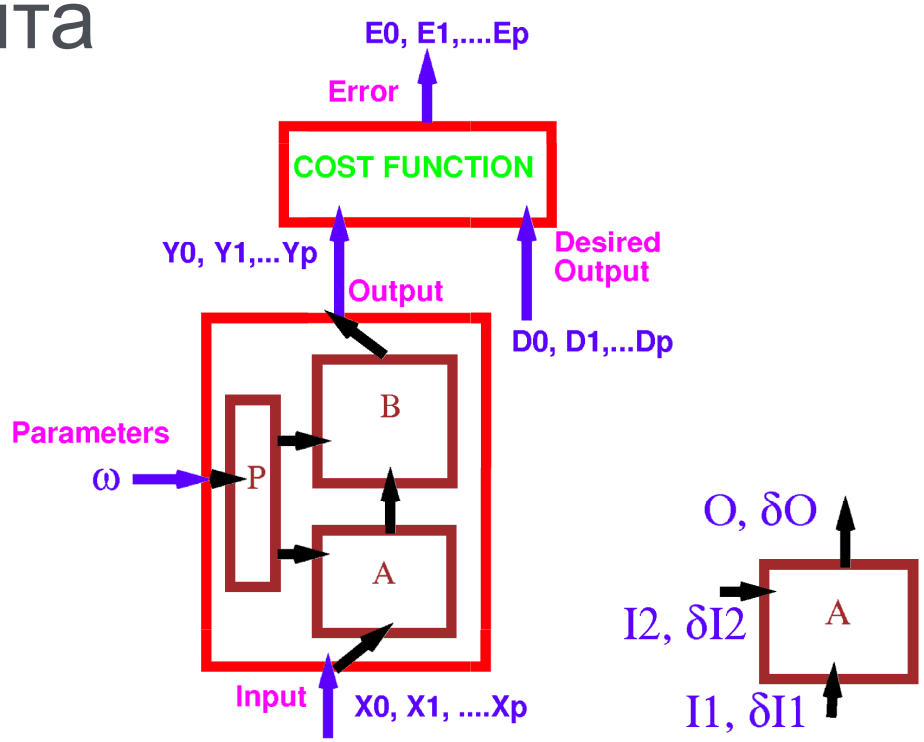


Gradient Descent:

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



Вычисление градиента в обратном распространении



– Обучаемая модель состоит из модулей (например слои)

– Каждый модуль делает две процедуры:

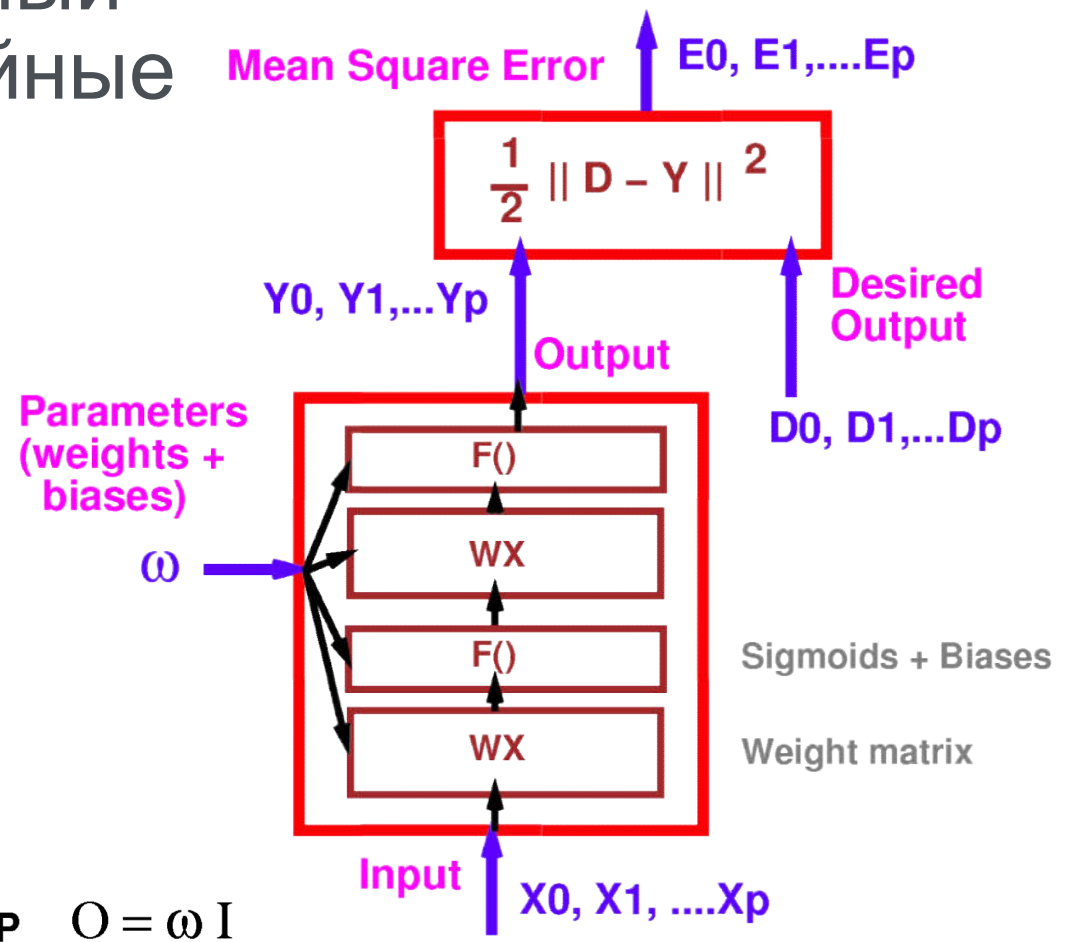
1. Вычисляет свои выходные значения из своих входных значений (пр. распр.)

$$O = A(I1, I2)$$

2. Вычисляет градиентные векторы входных значений по градиентным векторам выходных значений (обр. распр.)

$$\delta I1 = \frac{\partial A}{\partial I1}\,\delta O \qquad \delta I2 = \frac{\partial A}{\partial I2}\,\delta O$$

Интересный частный случай: многослойные сети



– Матрицы весов:

FPROP $O = \omega I$

BPROP $\delta I = \omega' \delta O ; \quad \delta\omega = \delta O' I$

– Сигмоиды + отступ(bias):

FPROP $O = f(I+B)$

BPROP $\delta I = f'(I+B)\ \delta O ; \ \delta B = \delta I$

Стохастичное обновление

– Стохастичный 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

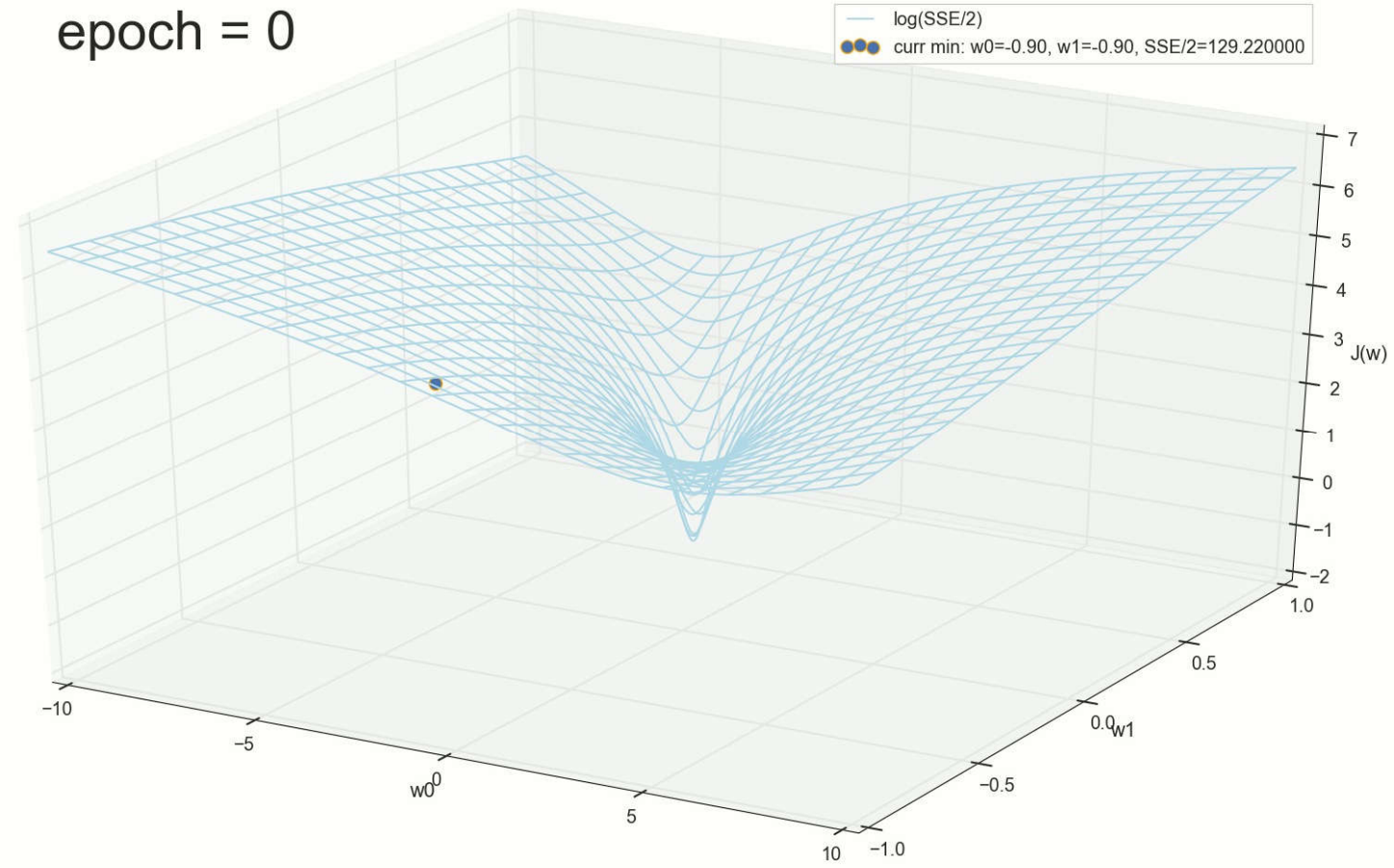
$$\omega(\rho+1) = \omega(\rho) - \eta \frac{\partial E}{\partial \omega}$$

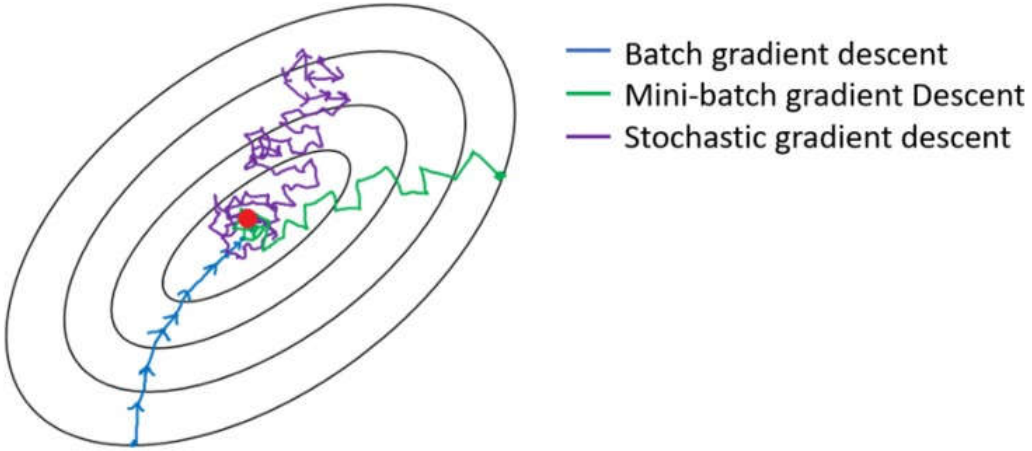
```
 Repeat {
  for all examples in training set {
     forward prop          // compute output
     backward prop         // compute gradients
     accumulate gradient }
  update parameters }
```

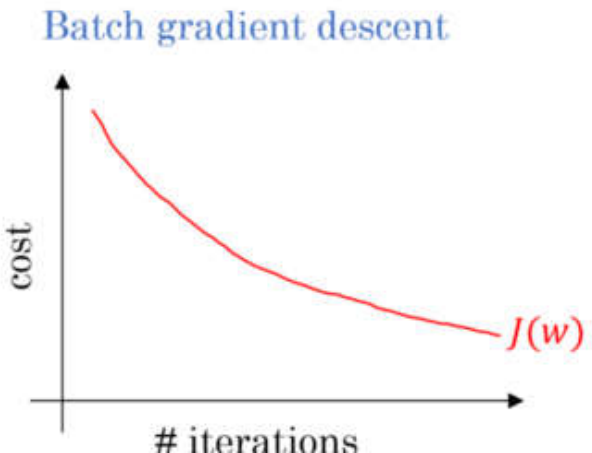
– Перед обновлением параметра из всего обучающего набора накапливаются градиенты

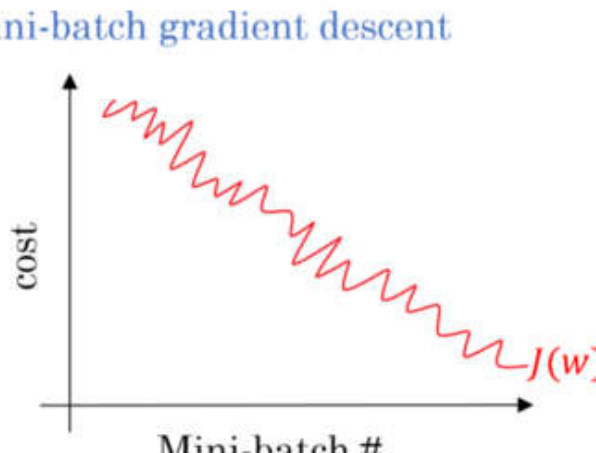
Батчи

- 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









Итерации и эпохи

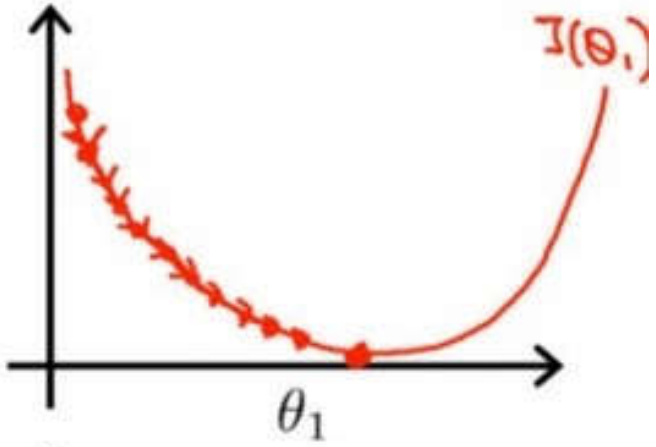
- Итерации – один шаг обучения
- Эпоха – обход всех экземпляров набора данных

Скорость обучения

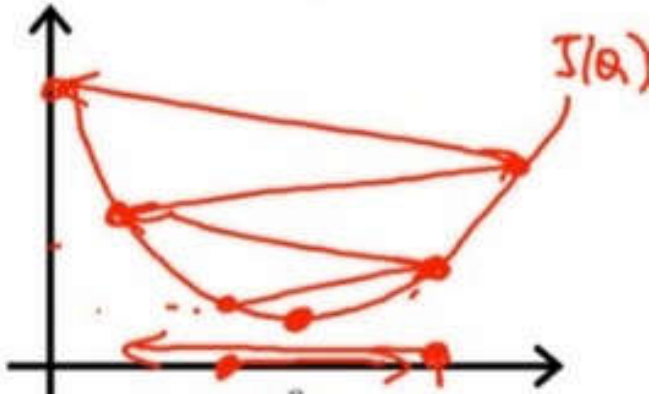
- Параметры (веса) сети меняются при обучении
- Гиперпараметры – нет. Управляем обучением

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

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.



Немного теории

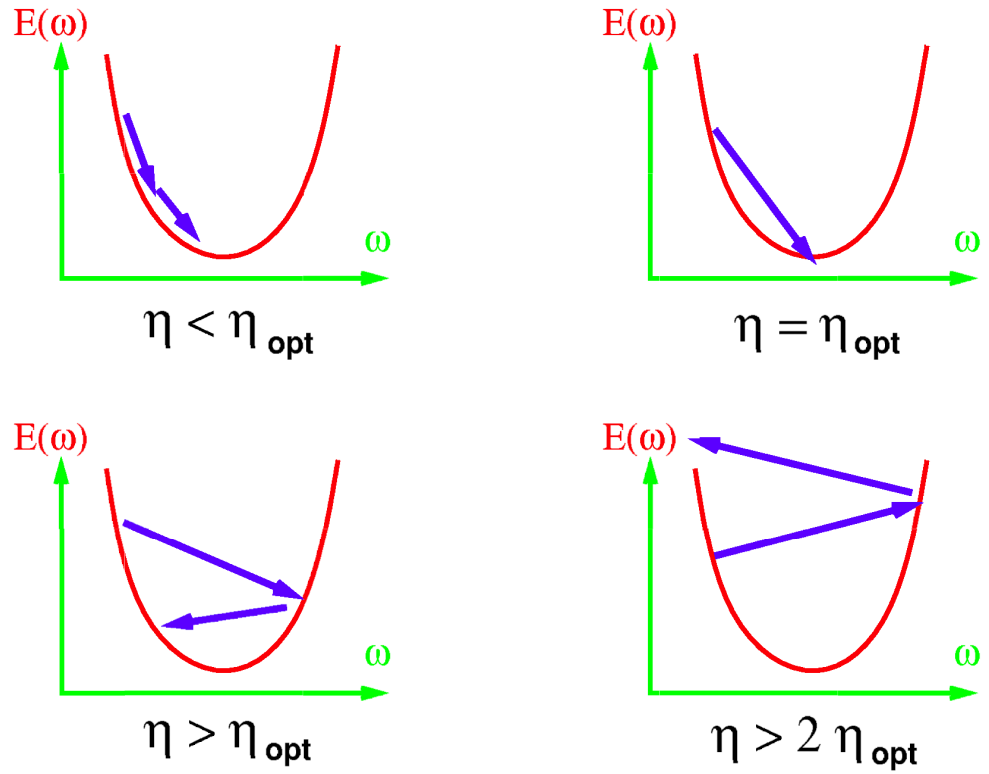
Градиентный спуск в одном измерении

$$\omega \leftarrow \omega - \eta \frac{\partial E}{\partial \omega}$$

gradient of objective function

weight vector

learning rate



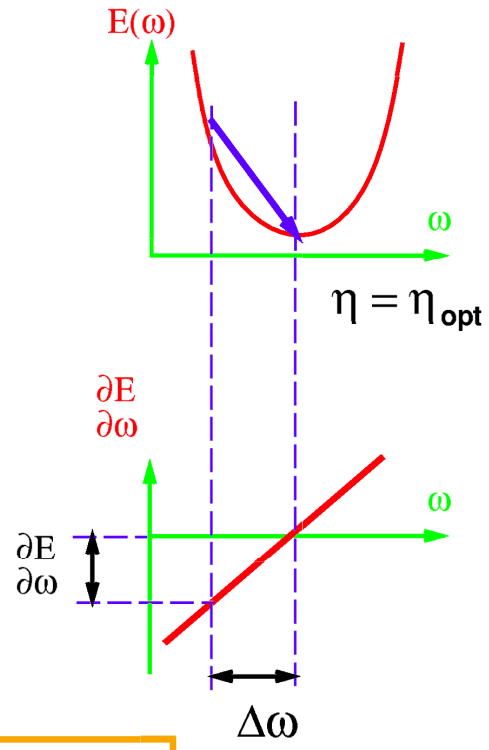
Оптимальный шаг обучения в 1D

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

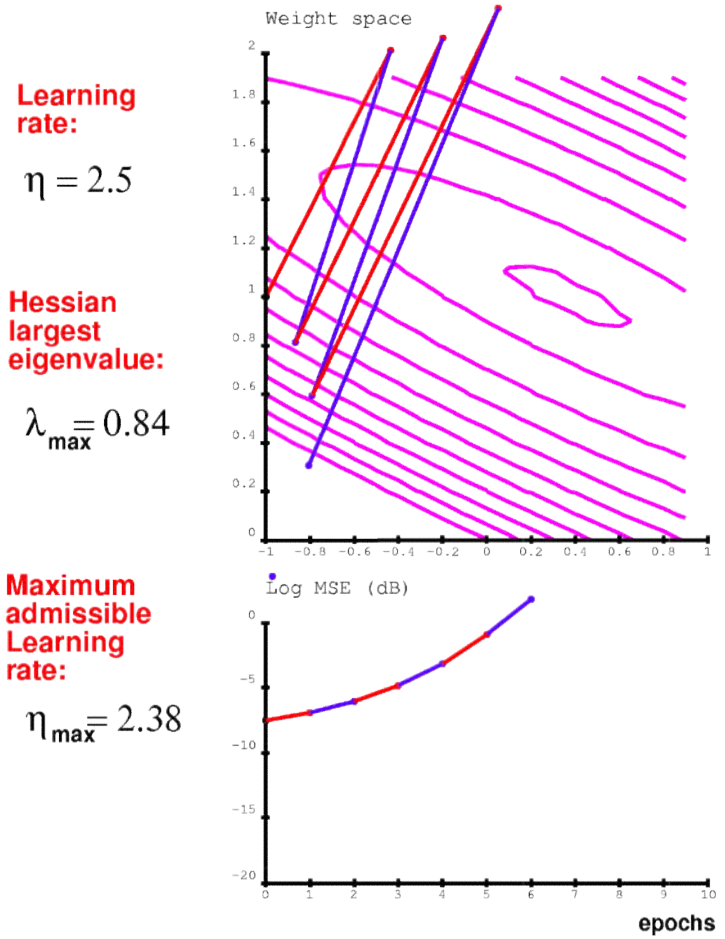
$$\eta_{opt} = \left(\frac{\partial^2 E}{\partial \omega^2} \right)^{-1}$$

Maximum Learning Rate

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

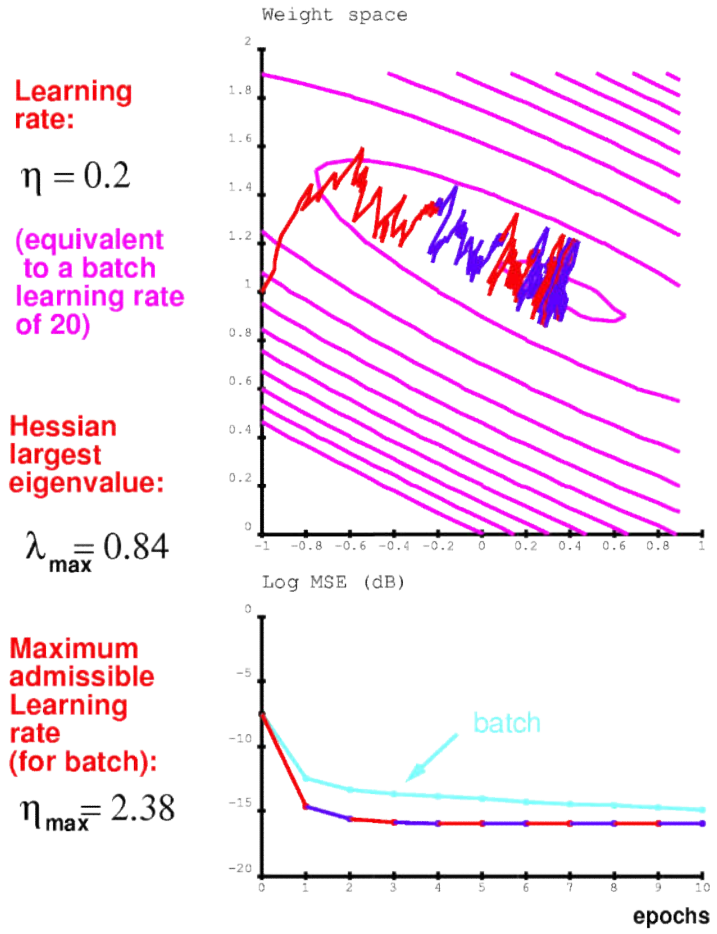
Пакетный градиентный спуск

– 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

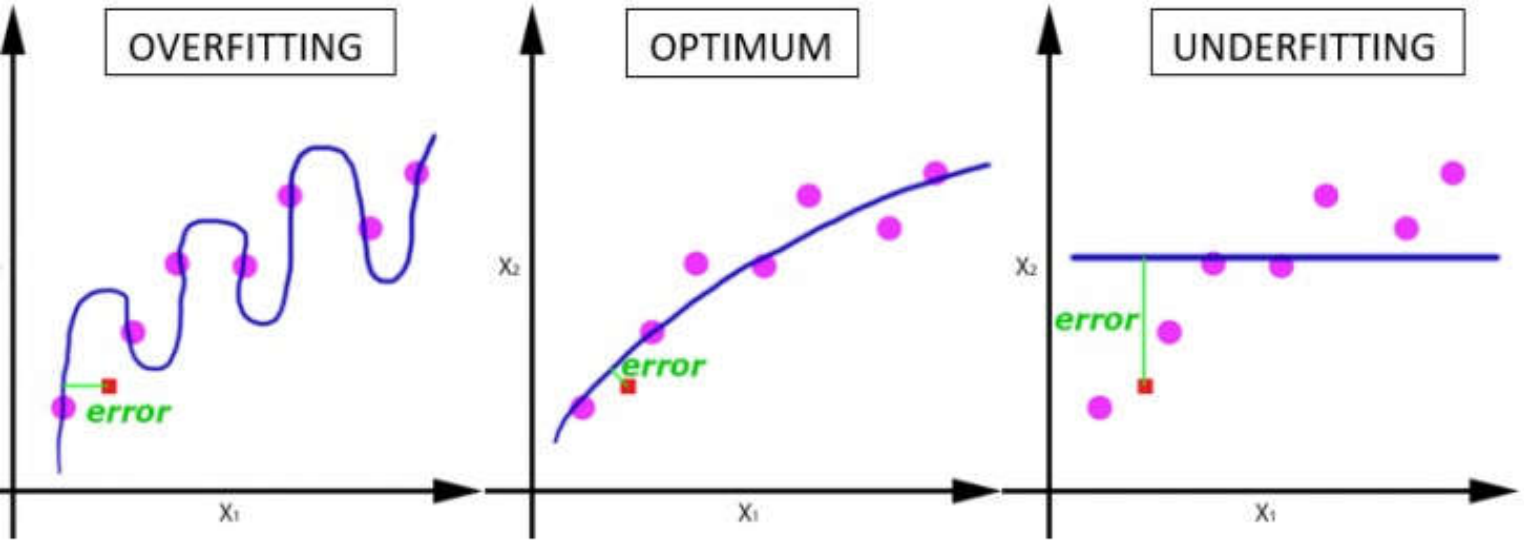
Стохастическое

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

Точность, переобучение



- Переобучение при долгом обучении слишком сложной модели