Машинное обучение: Градиентный спуск

Лекция №2

Виды ML

Аналогия

Мы не программируем правила →

→ Мы показываем примеры

→ Ребенок учится сам

Не пишем *жесткие* инструкции, а даем компьютеру много данных (примеров) и он сам находит в них закономерности (корреляции) и алгоритмы для решения задачи.

Традиционное программирование

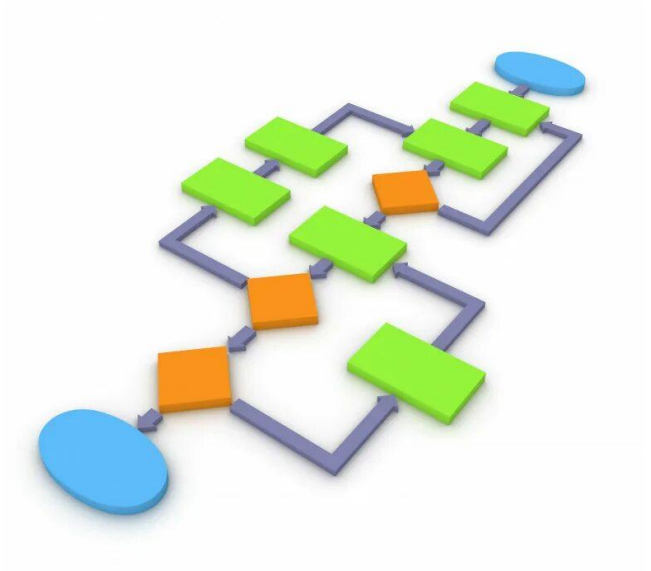
Входные данные + Алгоритм (набор правил) = **Результат**

Пример:

Входные данные: (7,8)

Алгоритм: Результат = x + y

Результат: 15



Машинное обучение

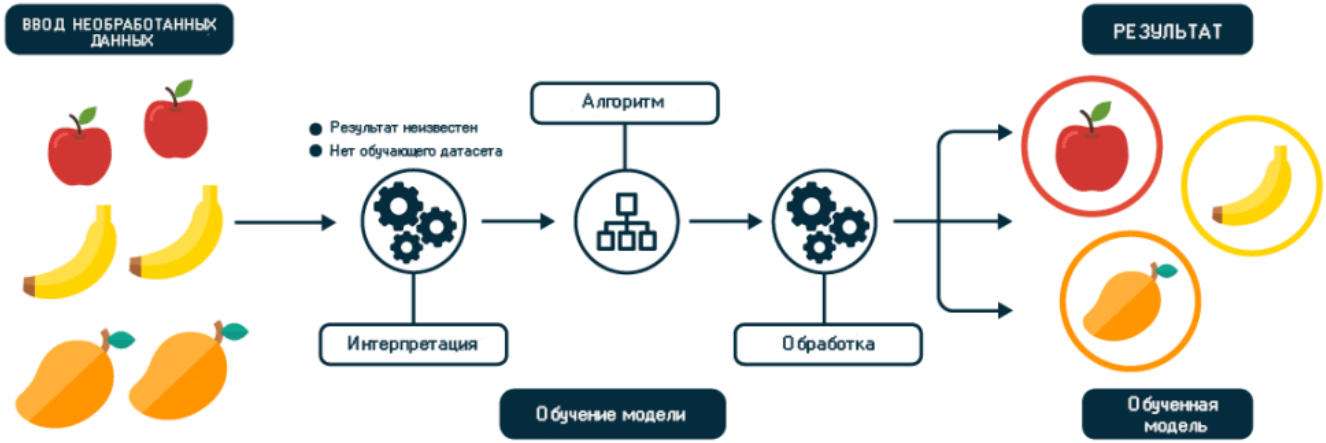
Входные данные + **Желаемый результат** = Модель (Алгоритм)

Пример:

Входные данные: Много пар чисел (2,3), (4,2), (1,2), …

Желаемый результат: 5, 6, 3 …

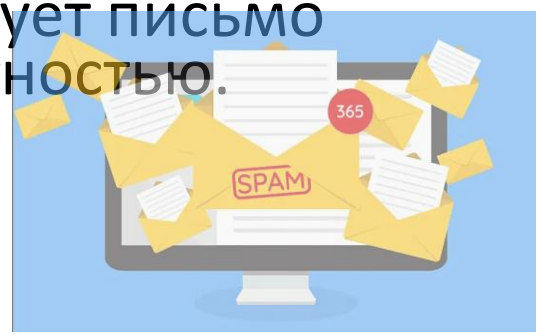
Получаемая модель: Сложение двух чисел.



Пример задачи классификации в ML

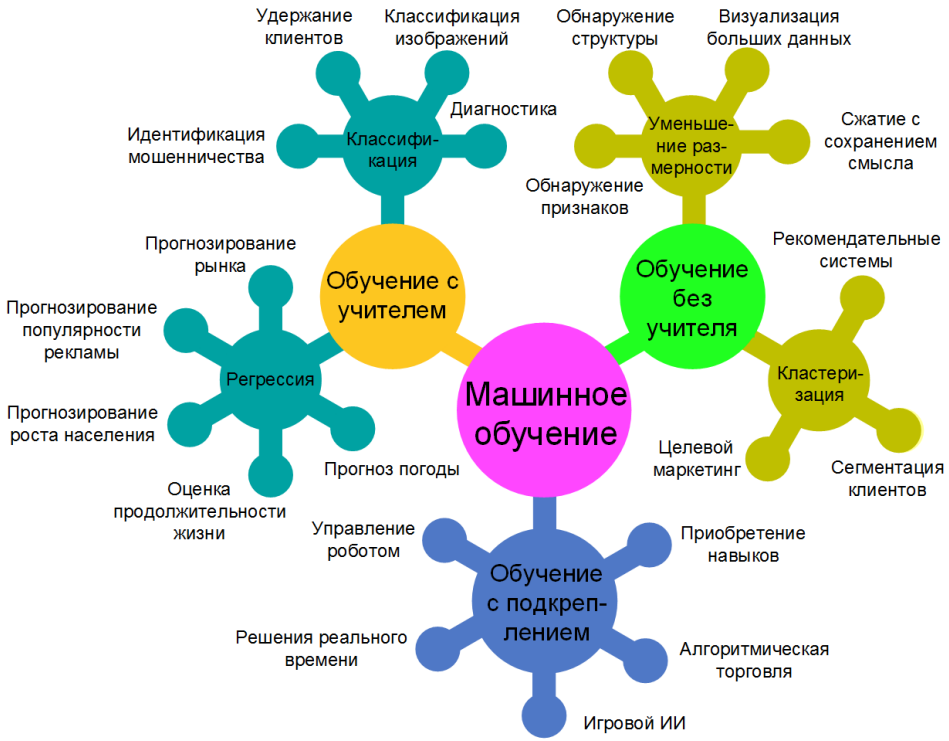
Фильтрация спама:

- Шаг 1. **Сбор и разметка данных**. Много писем, размеченных как «спам» и «не спам».
- Шаг 2. **Признаки.** Обучаем модель распознавать определенные слова в письме: «выиграл», «акция», «бесплатно», «ссылка» и т.д. – чаще встречаются в спаме. Такие слова как «встреча», «задача», «отчет», «проект».
- Шаг 3. **Обучение**. Компьютер анализирует тысячи писем и далее сам определяет какие письма являются спамом, а какие нет, обнаруживая новые зависимости, словосочетания, стиль и т.д.
- Шаг 4. **Прогноз**. Когда приходит спам, компьютер анализирует письмо и определяет спам оно или нет с определенной веростноятностью. Например, письмо «спам с вероятностью 88%».



Типы машинного обучения

- Обучение с учителем (Supervised Learning)
- Обучение без учителя (Unsupervised Learning)
- Обучение с подкреплением (Reinforcement Learning)

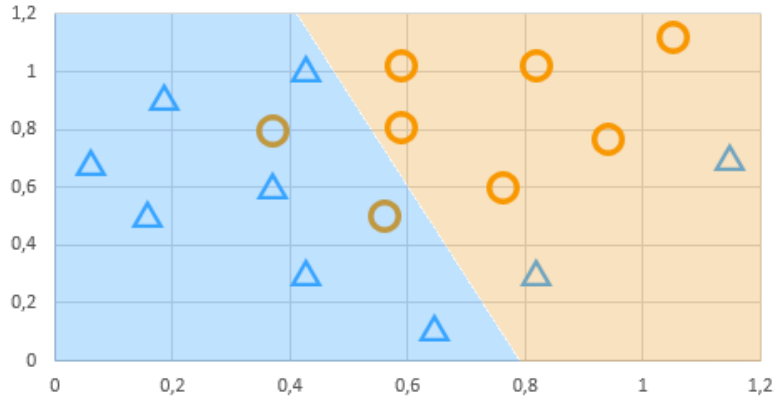


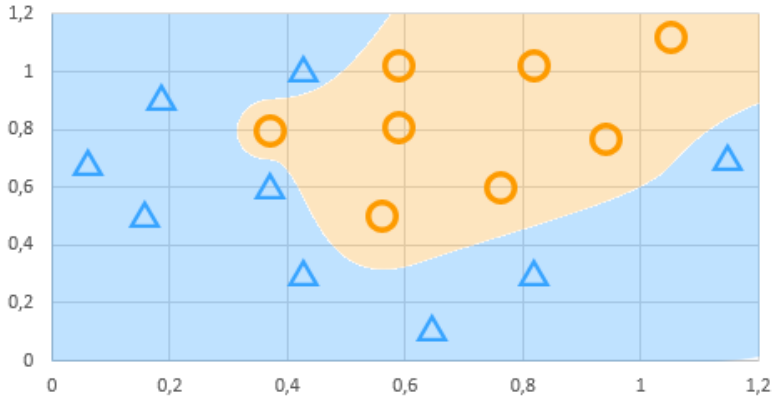
Обучение с учителем

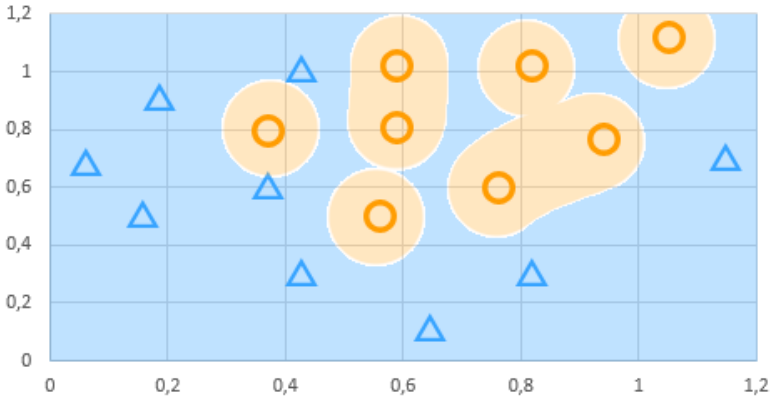
- Суть: У нас есть данные с «правильными ответами» (например, задача со «спамом»).
- Аналогия: Ученик и репетитор.
- Задачи:

- Классификация (спам/не спам, кошка/собака, цифры);

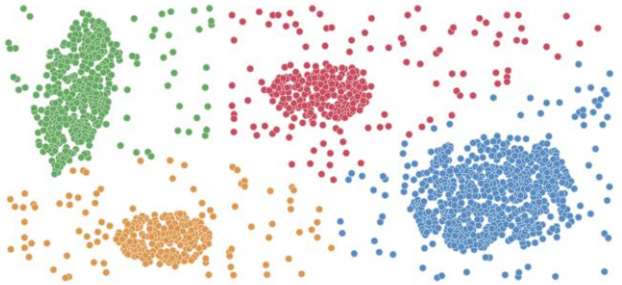
- Регрессия (предсказание погоды на основе данных за прошлые периоды, цены дома на основе площади, района и т.д.)



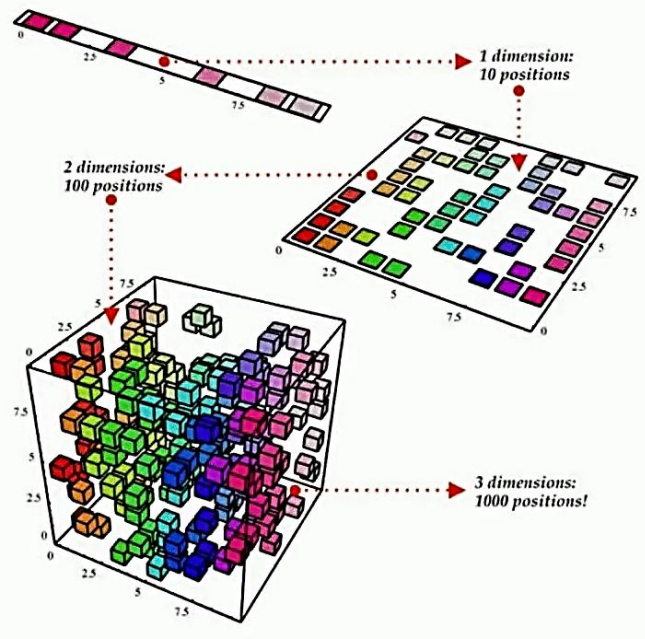




Обучение без учителя

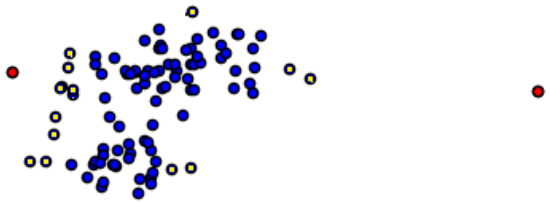


- Суть: **Неразмеченные данные без всяких пометок**. Алгоритм ищет скрытые закономерности, корреляции и структуры сам. Правильных ответов нет, на вход подаются только признаки.
- Ключевая метафора: **Исследователь в неизвестной стране**. У исследователя нет карты и гида. Он просто ходит, наблюдает за людьми, природой, зданиями и сам пытается найти какие-то группы, закономерности и структуры.
- Основные задачи:



- **Кластеризация**. Разбить данные на группы так, чтобы объекты внутри группы были максимально похожи, а объекты из разных групп максимально отличались (Группировка клиентов на основе их покупательского поведения, даже если мы не знаем какие типы клиентов существуют, Группировка документов по темам без информации какие темы существуют)

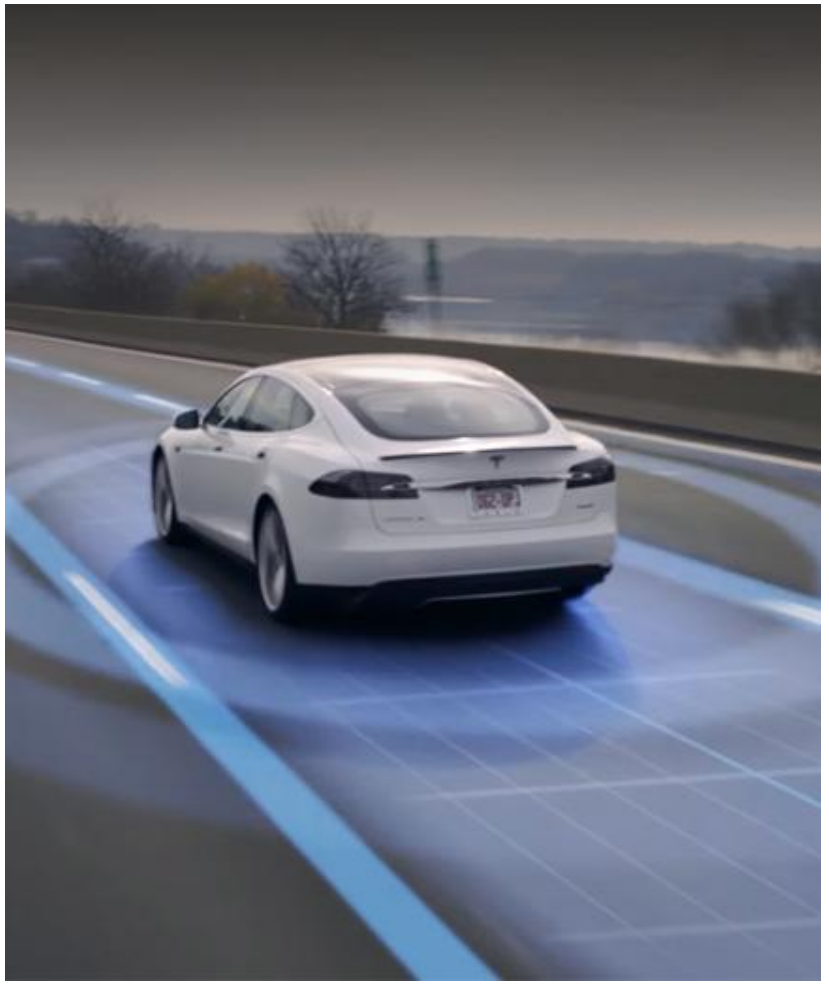
- **Понижение размерности**. Упростить данные, сократив количество признаков, но сохранив их суть. Например, нужно визуализировать сложные данные 2D|3D или ускорить работу алгоритмов, чтобы убрать информационный шум. (Пример. Есть данные о людях (рост, вес, размер обуви, объем талии, запястья и т.д. Многие из этих признаков взаимосвязаны. Алгоритм позволяет свести их к 1-2 главным фактором, условно, «размер тела»).



- **Обнаружение аномалий**. Найти непохожие, выбивающиеся из общего шаблона данные. Например, обнаружение мошеннических операций с кредитными картами, выявление сбоя на производственной линии по данным с датчиков.

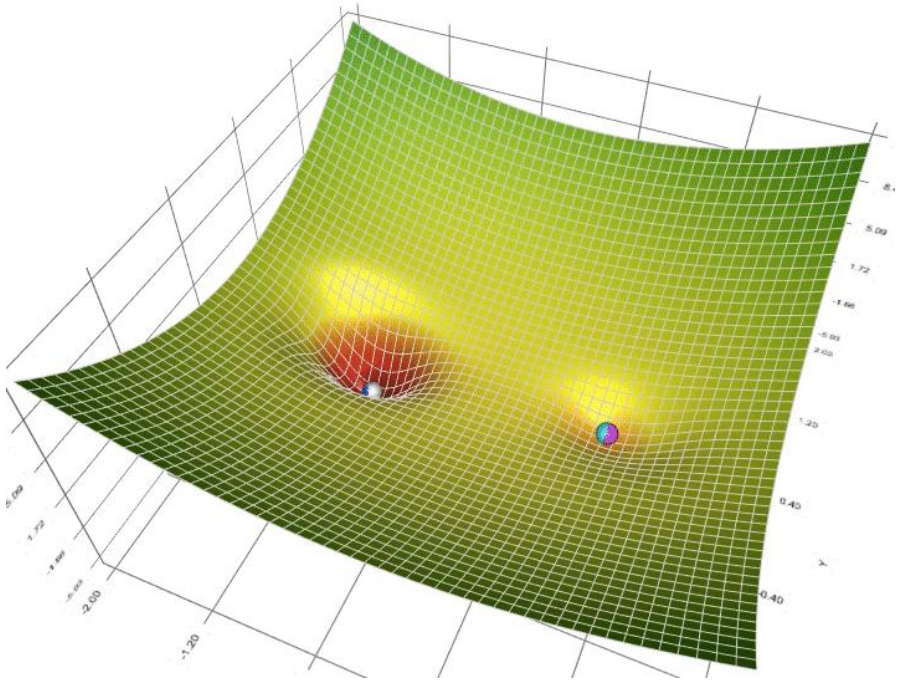
Обучение с подкреплением

- Суть: Система учится методом «проб и ошибок», получая «награду» за успешные действия. Нет готового набора данных.
- Аналогия: Дрессировка собаки, обучение алгоритма игры а шахматы и т.д. Агент находится в определенной среде. Он совершает действия и за правильные получает награду, а за ошибки штраф.
- Его цель – выработать стратегию, чтобы максимизировать совокупную награду.
- Пример. Беспилотные автомобили. Агент (беспилотник), среда (дорога), действие (ускорение, торможение, поворот руля и т.д.), награда (обратная связь от среды(+100 за объезд препятствия, -1000 за аварию)

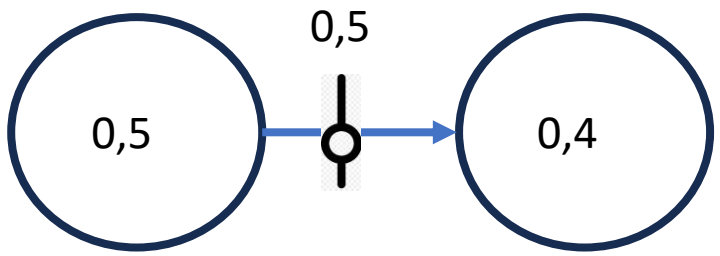


Градиентный спуск

Сравнение и обучение



Предсказание, сравнение, обучение



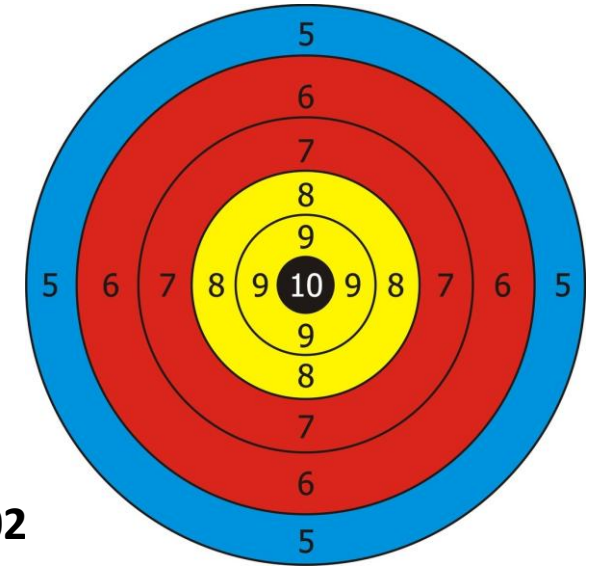
Вычисление ошибки среднеквадратичным способом

```
weight = 0.5
input = 0.5
goal_pred = 0.8

pred = input*weight                 // pred = 0.25

error = (pred – goal_pred)**2       // error = 0.302
print(error)
```

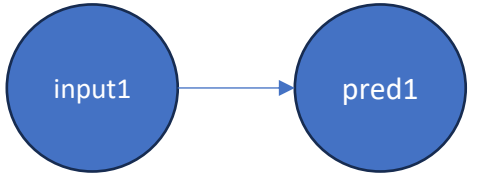
Истинное значение, которое нам известно



Оценка качества прогноза – одна из самых сложных задач МО.
Ошибка – мера промаха.

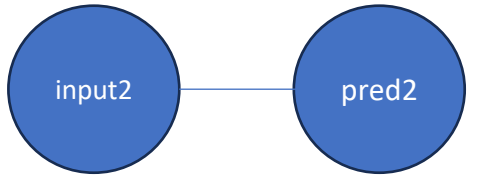
Среднеквадратичная ошибка:

- Делает чистую ошибку положительной (отрицательная не имеет смысла)
- Увеличивает важность большой ошибки
- Уменьшает важность маленькой ошибки



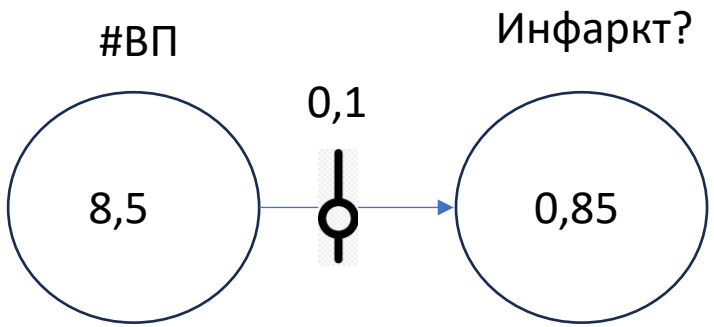
error1 = **1000**

Средняя ошибка = 0



error2 = **-1000**

Простейшая форма нейронного обучения: Метод холодно/горячо



1. Чистая сеть

```
Def neural_network(input, weight):
    prediction = input*weight
    return prediction
```

2. ПРОГНОЗИРОВАНИЕ: получение прогноза и вычисление ошибки

```
pred = neural_network(input, weight)
error = (pred – true)**2
```

3. СРАВНЕНИЕ: Получение прогноза с увеличенным значением веса и вычислением ошибки

```
up = neural_network(input, weight + step)
error_up = (up – true)**2
```

4. СРАВНЕНИЕ: Получение прогноза с уменьшенным значением веса и вычислением ошибки

```
down = neural_network(input, weight - step)
error_down = (down – true)**2
```

5. СРАВНЕНИЕ и ОБУЧЕНИЕ: сравнение ошибок и выбор нового значения веса

```
If (error_down < error_up)
        weight = weight – step
If (error_down > error_up)
        weight = weight + step
```

+ *Очень прост*
- *Неэффективен – для ОДНОГО изменения веса узла требуется вычислить прогноз НЕСКОЛЬКО раз*
- *Часто нельзя добиться идеальной точности прогноза – мы знаем направление поиска, но не знаем на какую величину делать шаг*

Предсказание, сравнение, обучение

ОБУЧЕНИЕ – процесс определения *причин* ошибок, т.е. анализ вклада каждого веса в общую ошибку.

1. **ЧИСТАЯ СЕТЬ**

```
def neural_network(input, weight):
    prediction = input*weight
    return prediction
```

2. **ПРОГНОЗИРОВАНИЕ**: получение прогноза и вычисление ошибки

```
pred = neural_network(input, weight)
error = (pred – true)**2
```

3. **СРАВНЕНИЕ**: вычисление разности на выходе между прогнозом и истиной

```
delta = pred – true
```

4. **ОБУЧЕНИЕ**: вычисление разности весов и их корректировка

```
weight_delta = input*delta
weight = weight – weight_delta*alpha
```

ВЫЧИСЛЕНИЕ НАПРАВЛЕНИЯ И ВЕЛИЧИНЫ ОШИБКИ:
МЕТОД ГРАДИЕНТНОГО СПУСКА

weight_delta – показывает как должен измениться вес, умножая *чистую ошибку delta* на *вход input*. Вычисляется сразу и направление, и величина изменения веса weight для уменьшения ошибки error (производится масштабирование, обращение знака и остановка)

Остановка – умножение на нулевой вес.
Обращение знака (при отрицательном значении input смещение веса вверх заставит прогноз сместиться вниз. Это гарантирует смещение веса в правильном направлении даже при отрицательном input.
Масштабирование – если input имеет большое значение, то по идее и вес нужно изменить на большую величину.

Обучение уменьшает ошибку

Уменьшить ошибку можно изменением веса!!!

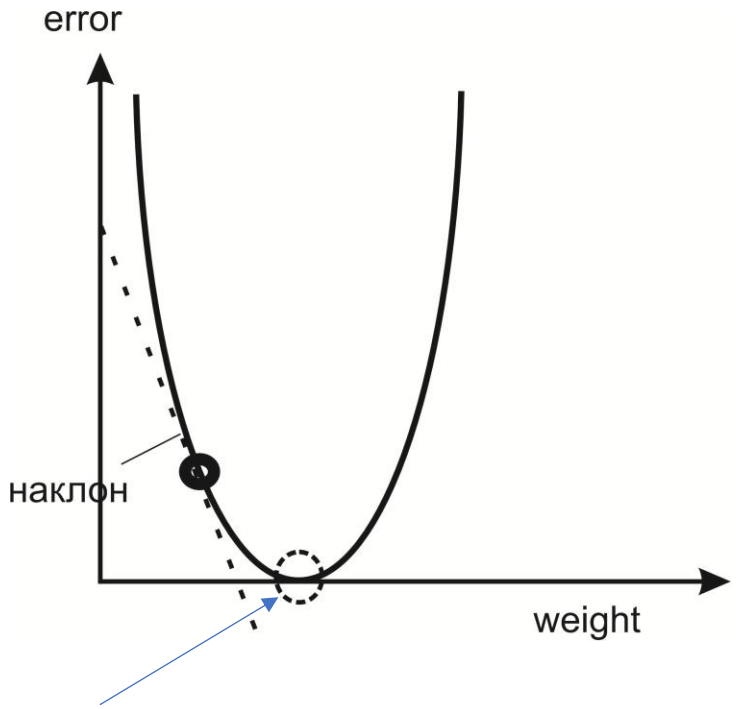
С помощью градиентного спуска можно корректировать каждый вес в правильном направлении и на правильную величину, чтобы в итоге уменьшить ошибку до нуля. ЭТО СУТЬ МАШИННОГО ОБУЧЕНИЯ!

error = (pred – true)**2
error = (**input***weight – **true**)**2

Для любых input и true существует точное отношение между error и weight. Оно определяется комбинацией формул прогнозирования и вычисления ошибки

Какой график функции будет описывать зависимость ошибки от веса?

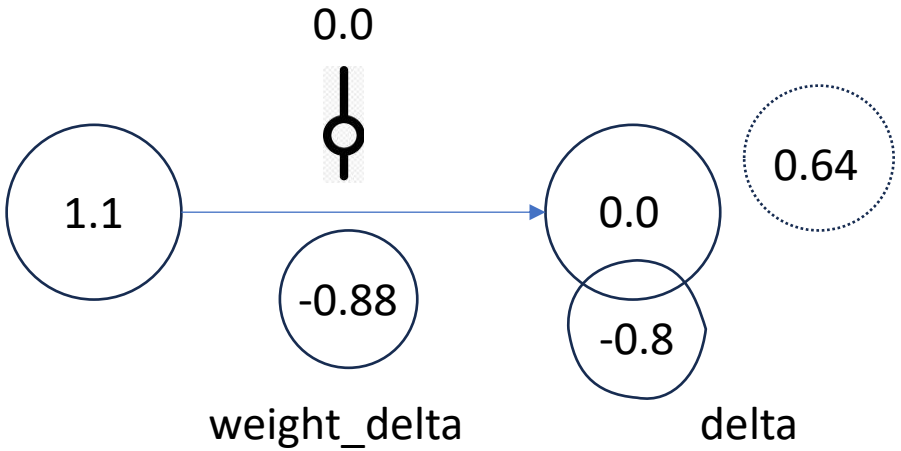
Суть обучения сводится к автоматическому изменению функции прогнозирования, чтобы она научилась давать точные прогнозы (ошибка равнялась нулю).

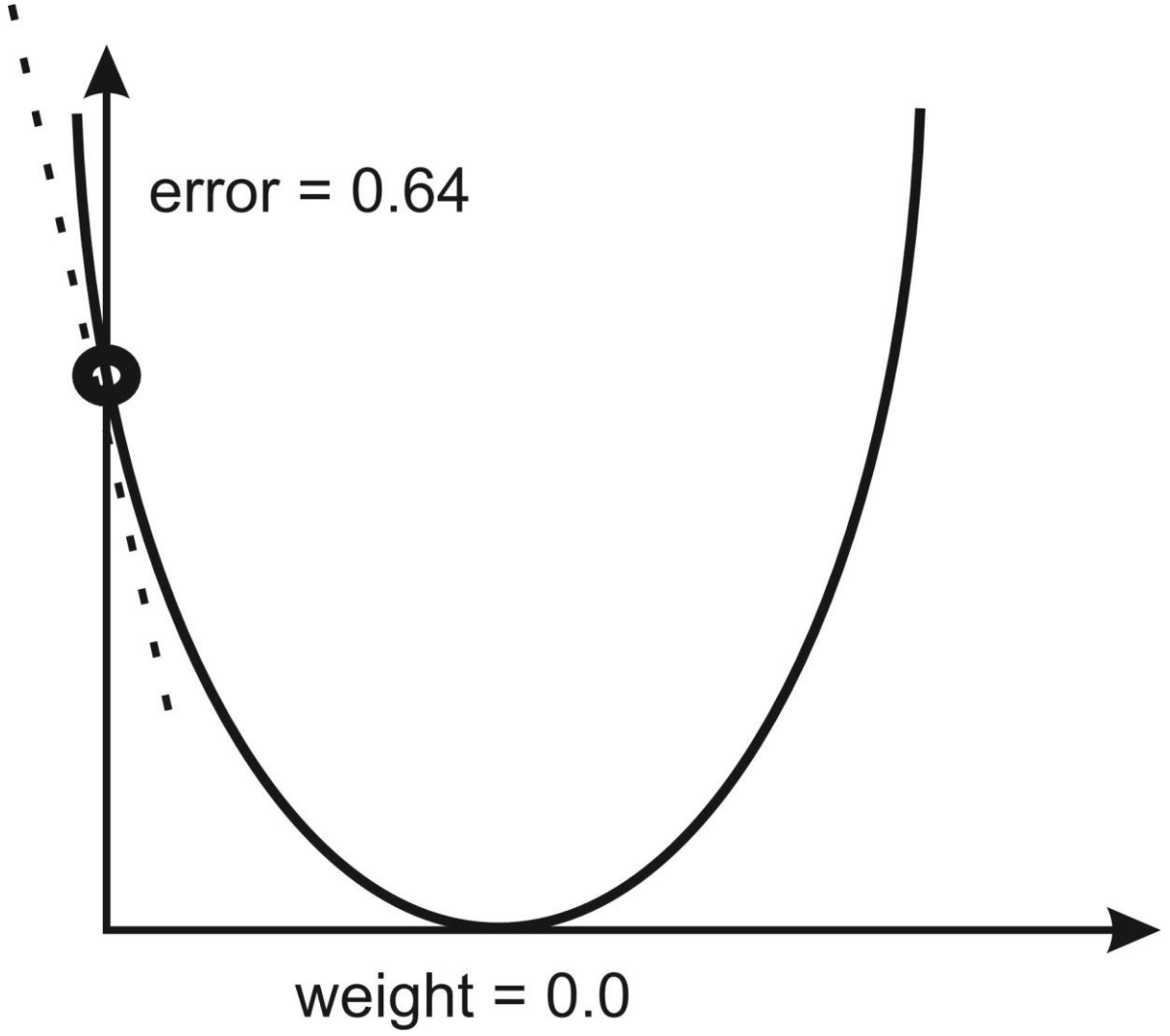


Наша конечная цель!!!

Пример цикла обучения

Шаг 1. Увеличиваем вес на большую величину



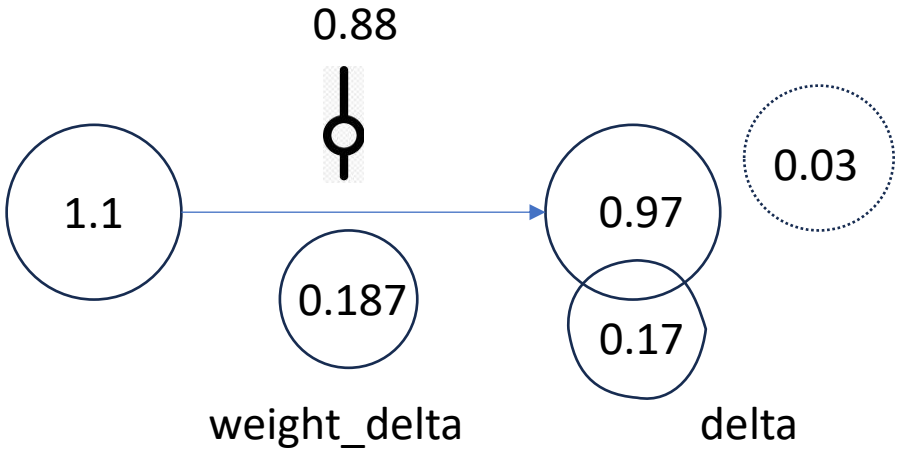


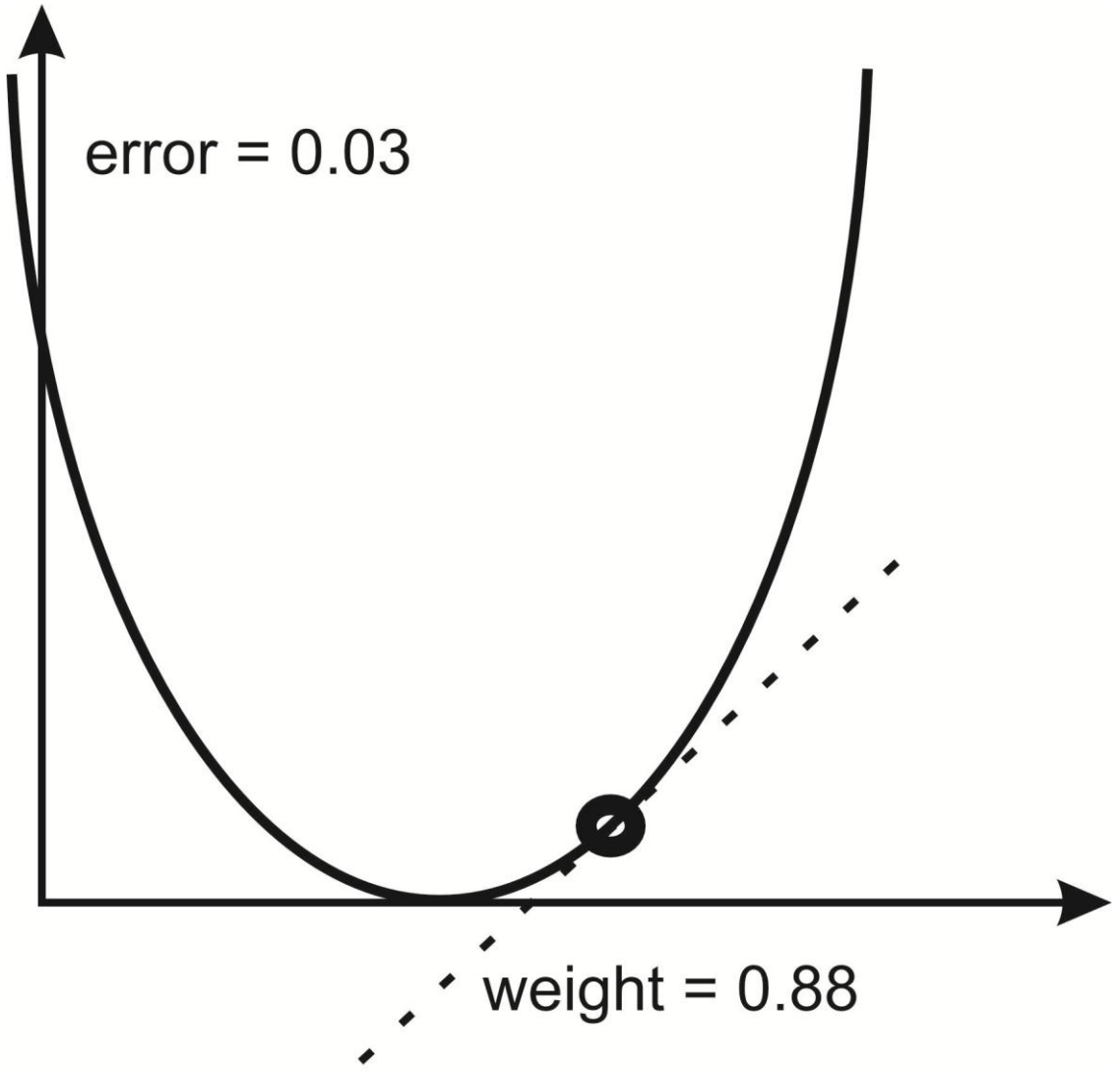
weight, true, input = (0.0, 0.8, 1.1)

pred = input*weight = 1.1*0.0 = 0
error = (pred – true)**2 = (0 – 0.8)**2 = 0.64
delta = pred – true = 0 - 0.8 = -0.8
weight_delta = delta*input = -0.8*1.1 = -0.88
weight = weight – weight_delta = 0 – (-0.88) = 0.88

Пример цикла обучения

Шаг 2. Перелет. Делаем шаг в обратную сторону

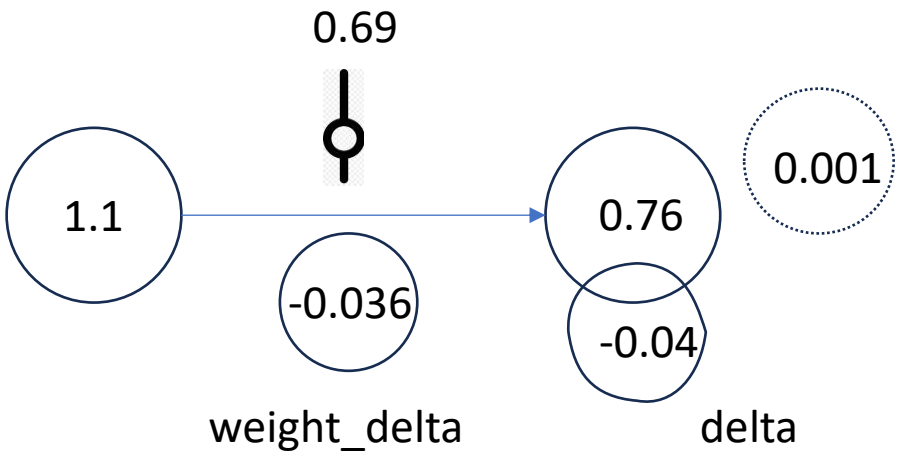


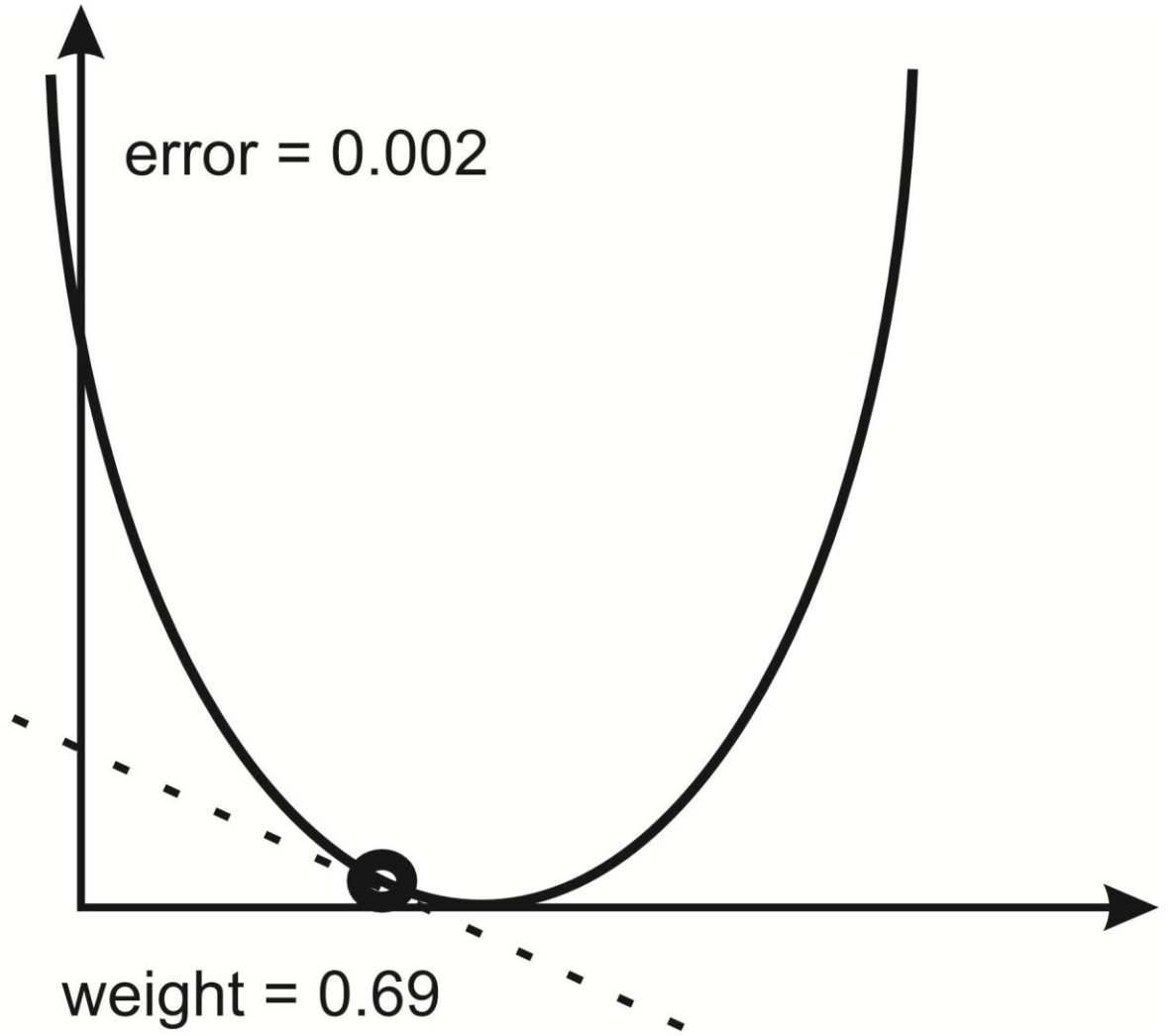


pred = input*weight = 1.1*0.88 = 0.968
error = (pred – true)**2 = (0.968 – 0.8)**2 = 0.028
delta = pred – true = 0.97 - 0.8 = 0.17
weight_delta = delta*input = 0.17*1.1 = 0.187
weight = weight – weight_delta = 0.88 – 0.187 = 0.693

Пример цикла обучения

Шаг 3. Опять перелет. Делаем шаг обратно поменьше

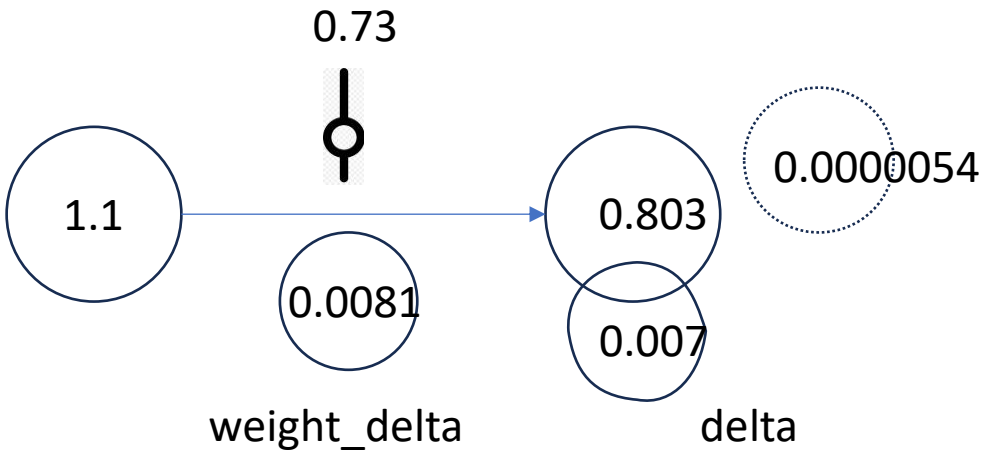


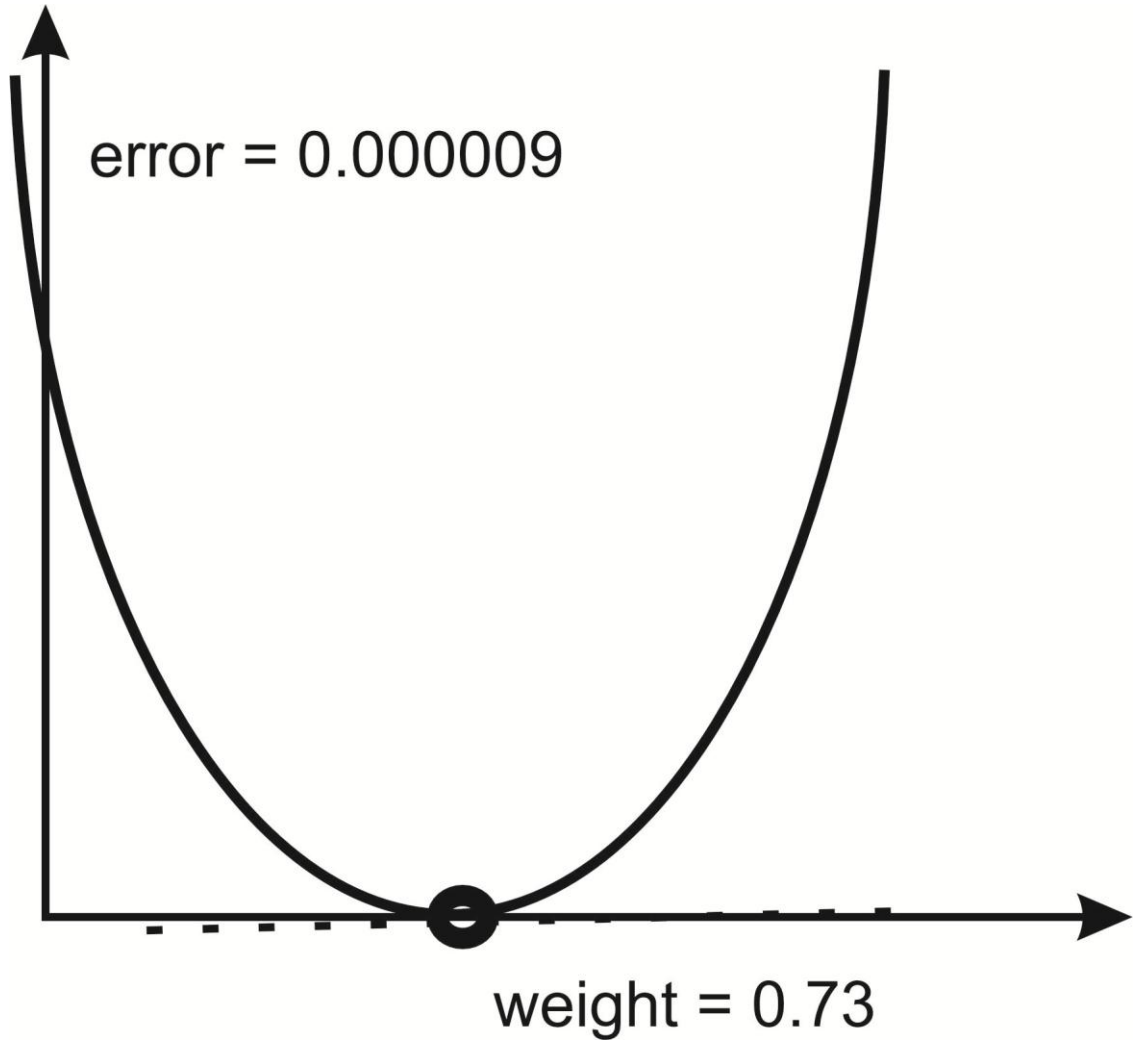


pred = input*weight = 1.1*0.69 = 0.759
error = (pred – true)**2 = (0.76 – 0.8)**2 = 0.0017
delta = pred – true = 0.76 - 0.8 = -0.04
weight_delta = delta*input = 0.04*1.1 = -0.044
weight = weight – weight_delta = 0.69 – (-0.044) = 0.734

Пример цикла обучения

Шаг 4. Почти попали в цель!





pred = input*weight = 1.1*0.73 = 0.803
error = (pred – true)**2 = (0.803 – 0.8)**2 = 0.000009
delta = pred – true = 0.803 - 0.8 = 0.03
weight_delta = delta*input = 0.03*1.1 = 0.033
weight = weight – weight_delta = 0.73 – 0.033 = 0.697

Какая связь между ошибкой и весом?

Связь (функция) – это как изменение одной переменной влияет на поведение другой.
Чувствительность к изменениям – насколько направление и величина изменение веса влияет на общую ошибку.
Например, y = 2*x.

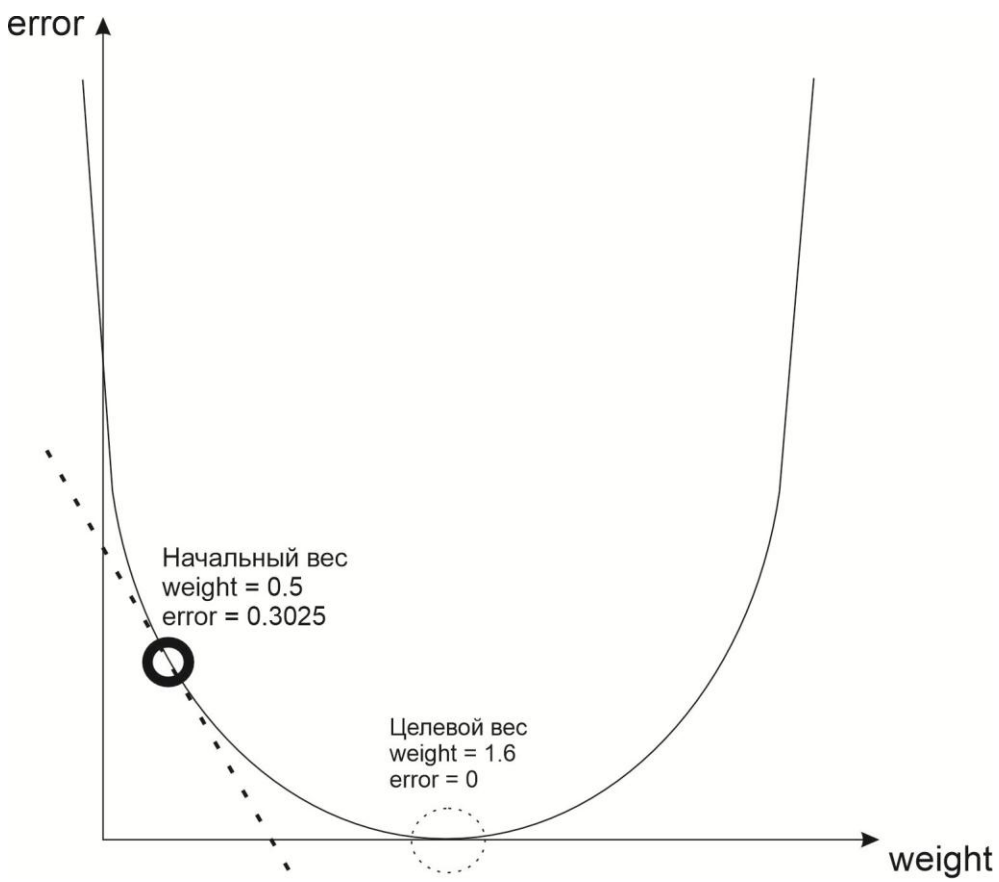
error = (input*weight – true)**2 – универсальная форма связи ошибки и веса.

Можем ли мы менять true? Чтобы уменьшить ошибку до нуля?
A input? A операции возведения в квадрат или арифметические операции?
Как нам подстроиться под закономерность данных?

Что такое производная?
Производная – функция, которая показывает как изменится одна переменная при изменении другой.
Между двумя переменными всегда есть производная!
Производная – это наклон прямой или кривой в данной точке.

error = (0.5*weight – 0.8)**2

Графическое представление связи между ошибкой и весом?



При удалении от целевого веса, тем круче наклон производной. Справа наклон положительный, а слева – отрицательный. Знак наклона дает направление, а крутизна наклона – величину изменения.

Если взять производную для любого значения веса, можно определить насколько далеко мы находимся от целевого веса и в каком направлении нам двигаться.

Для любой самой сложной функции ошибки можно вычислить отношение между любым весом и окончательной ошибкой сети. То есть производная показывает **какой вклад вносит в ошибку изменение веса.**

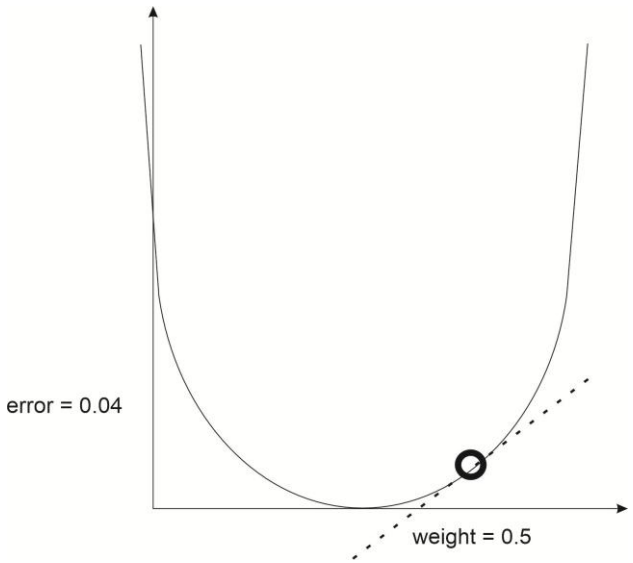
Наклон кривой графика указывает в направлении, противоположном направлению от самой нижней точку на кривой: если наклон отрицательный – мы должны увеличить вес.

В каком направлении нужно двигаться, чтобы уменьшить ошибку?

Этот метод называется – градиентный спуск.

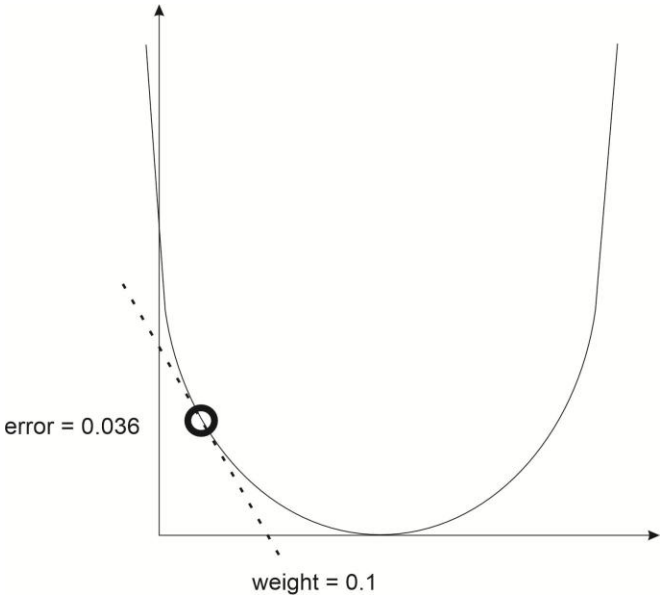
Избыточная коррекция веса

Рассмотрим input = 2, true = 0.8



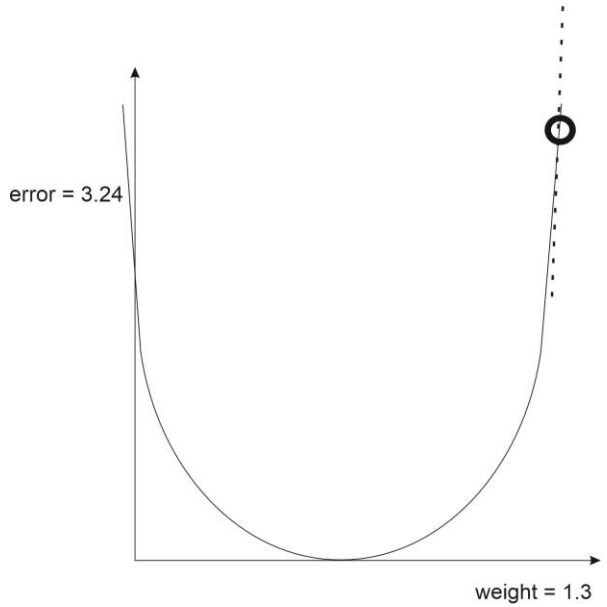
1. Слишком большое Значение веса

Чистая ошибка = 0.2



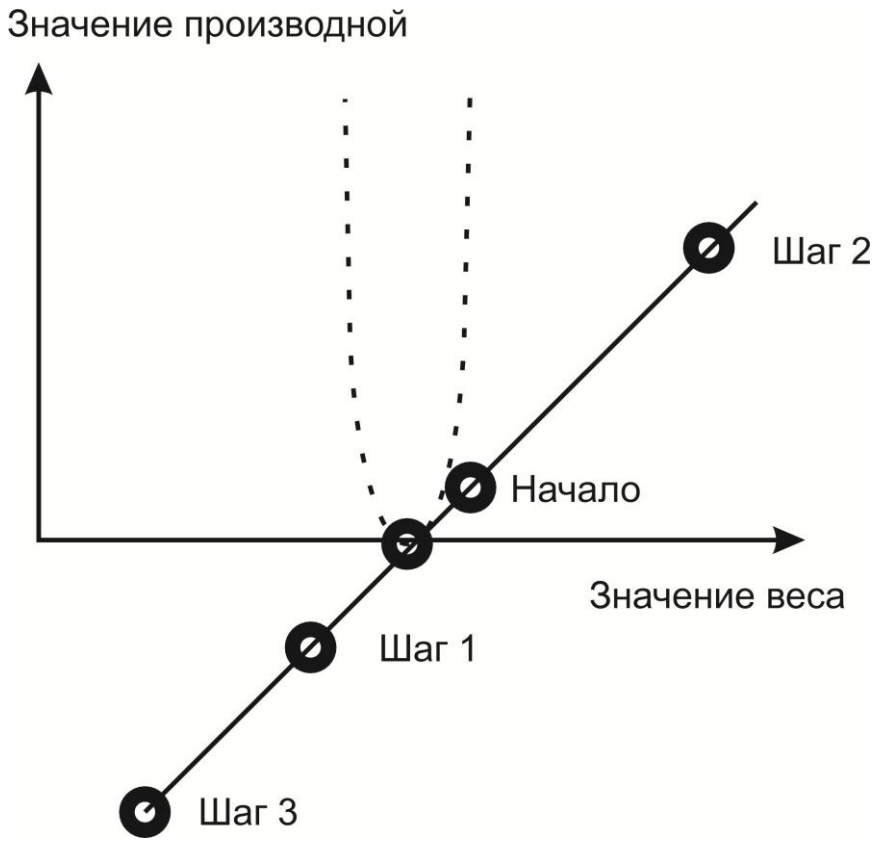
2. Перелет, делаем шаг назад

Чистая ошибка = - 0.6



3. Снова перелет, делаем шаг назад, но уже меньше
Чистая ошибка = 1.8

Избыточная коррекция веса: расхождение



Произошло взрывное расхождение веса!

weight = weight – input*(pred – true)

При очень больших входных значениях прогноз становится чувствительным к изменению веса.

В этом случае вес изменяется на большую величину при малом значении ошибки – происходит избыточная коррекция веса в сети.

Нормализация: альфа-коэффициент

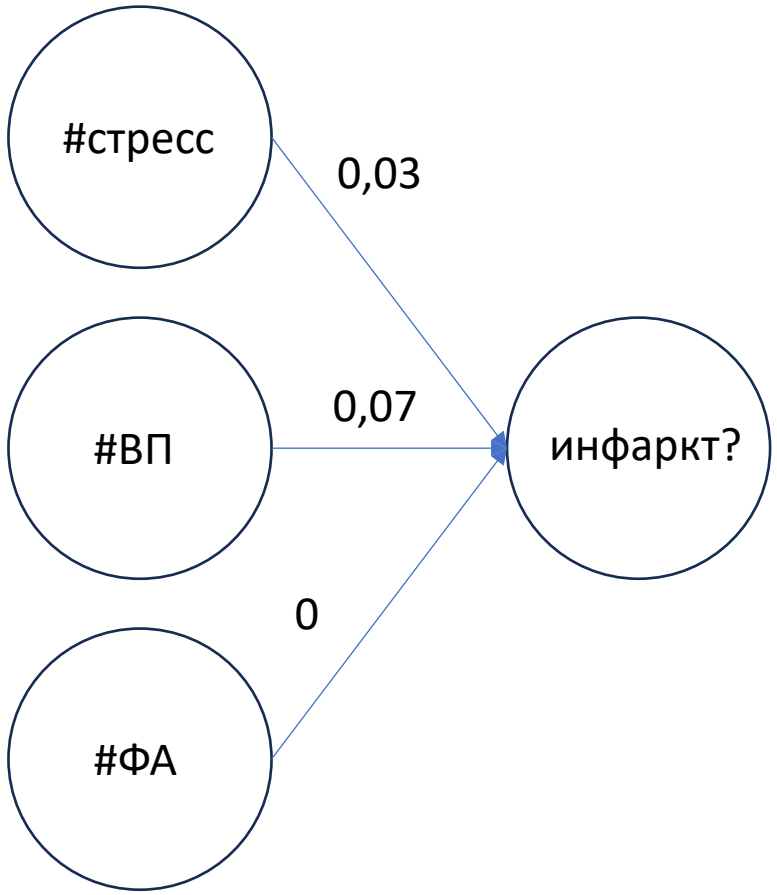
В результате избыточной коррекции веса величина производной в новой точке будет превосходить величину прежней производной (с другим знаком).

Для корректировки значения производной вносится случайная величина. Например, 10, 1, 0.1, 0.01, 0.001 и так далее)

Weight = weight – alpha*input*weight

Выбор значения коэффициента происходит интуитивно!

Обобщение градиентного спуска с несколькими входами (корректировка нескольких весов)



1. Чистая сеть с несколькими входами

```
def neural_network(input, weights):
        pred = w_sum(input, weights)
        return pred
```

2. Прогноз+сравнение

```
pred = neural_network(input, weights)
error = (pred – true)**2
delta = pred – true
```

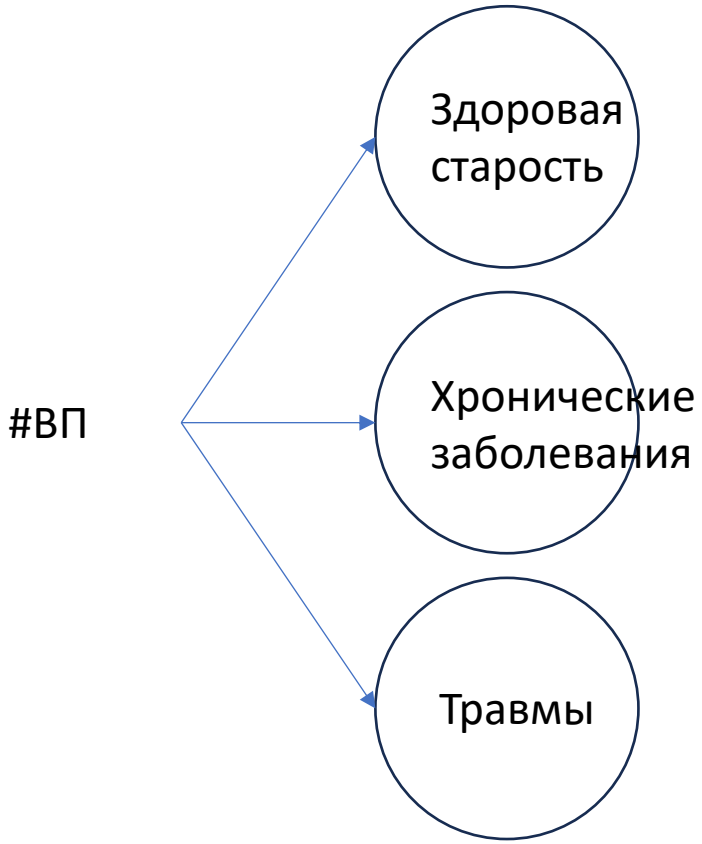
3. Обучение

```
weight_deltas = ele_mul(delta, input)
```

4. Обучение с корректировкой весов

```
alpha = 0.01
for i in range(len(weights)):
        weights[i] -= alpha*weights_deltas[i]
```

Градиентный спуск с несколькими выходами



1. Чистая сеть с несколькими выходами

```
def neural_network(input, weights):
        pred = w_sum(input, weights)
        return pred
```

2. Прогноз и вычисление ошибки

```
pred = neural_network(input, weights)
For i in range(len(true)):
        error[i] = (pred [i] – true [i] )**2
        delta [i] = pred [i] – true [i]
```

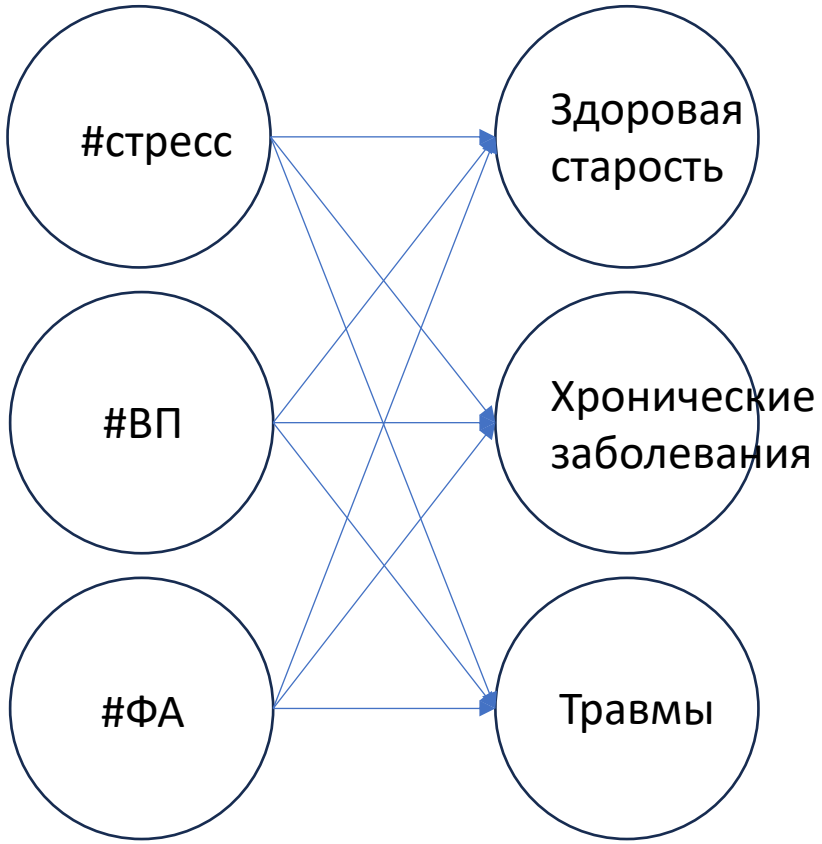
3. Сравнение: вычислинение каждого приращения weight_delta и коррекция каждого веса

```
weight_deltas = scalar_ele_mul(input, delta)
```

4. Обучение с корректировкой весов

```
alpha = 0.1
for i in range(len(weights)):
        weights[i] -= alpha*weights_deltas[i]
```

Градиентный спуск с несколькими входами и выходами



1. Чистая сеть с несколькими выходами

```
def neural_network(input, weights):
        pred = vect_mat_mul(input, weights)
        return pred
```

2. Прогноз и вычисление ошибки и разности

```
pred = neural_network(input, weights)
for i in range(len(true)):
        error[i] = (pred [i] – true [i] )**2
        delta [i] = pred [i] – true [i]
```

3. Сравнение: вычисление каждого приращения weight_delta и коррекция каждого веса

```
weight_deltas = outer_prod(input, delta)
```

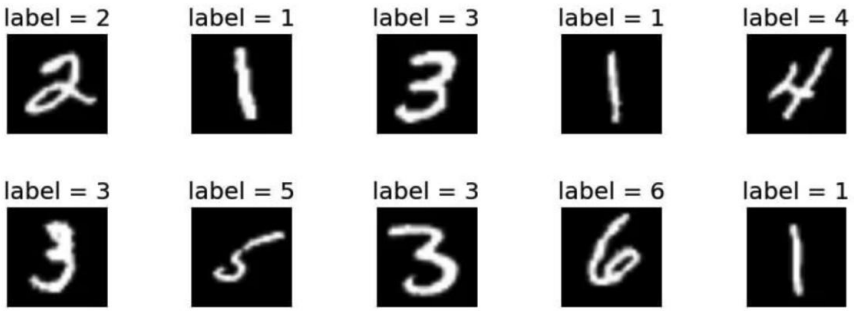
4. Обучение с корректировкой весов

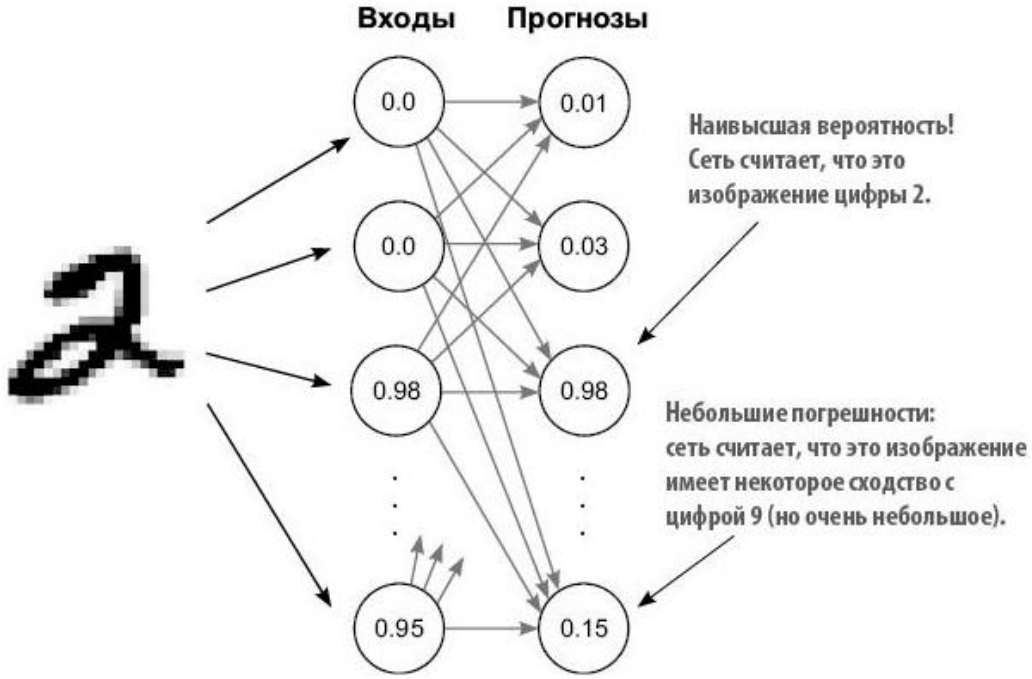
```
alpha = 0.1
for i in range(len(weights)):
        weights[i] -= alpha*weights_deltas[i]
```

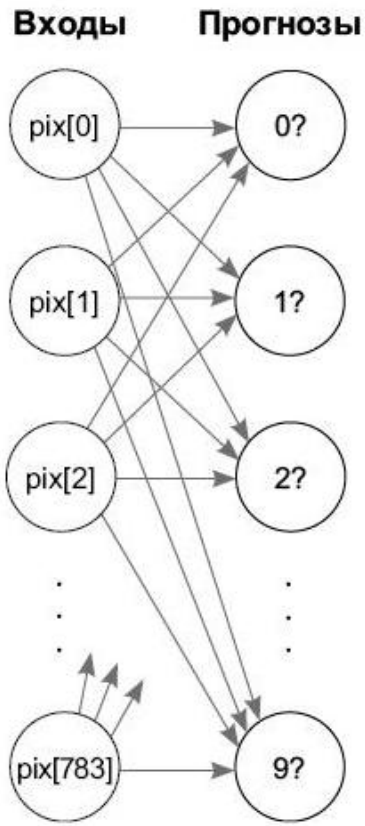
Набор для обучения нейронной сети MNIST

Modified National Institute of Standards and Technology

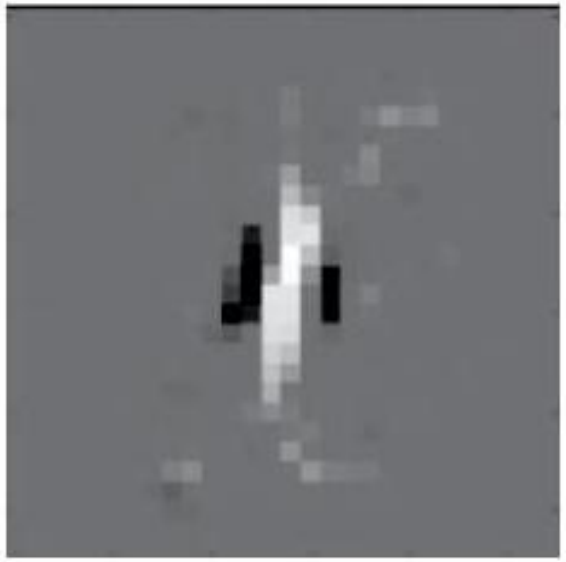
Изображения чисел от 0 до 9 (28x28 пикселей, 784 пикселя)

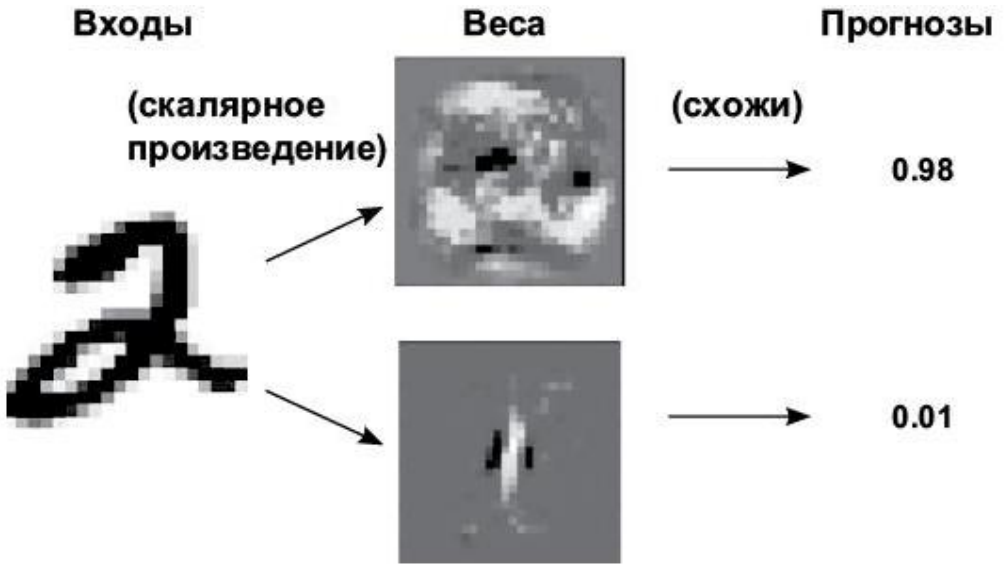






Визуализация значений весов и скалярных произведений сумм весов





a = [0, 1, 0, 1] c = [0, 1, 1, 0] b = [1, 0, 1, 0]

b = [1, 0, 1, 0] d = [.5, 0,.5, 0] c = [0, 1, 1, 0]

[0, 0, 0, 0] -> 0

Градиентный спуск – универсальный алгоритм ML