



# Градиентные методы обучения

МАКСИМОВСКАЯ  
АНАСТАСИЯ

# Повторение

---

$$a(x) = w_0 + w_1 \cdot x_1 + \dots + w_d \cdot x_d = w_0 + \sum_{i=1}^d w_i \cdot x_i$$

# Кодировки

---

➤ Label Encoding

# Кодировки

---

- Label Encoding
- One-Hot Encoding

# Кодировки

---

- Label Encoding
- One-Hot Encoding
- Target Encoding

<b>id</b>	<b>job</b>	<b>job_mean</b>	<b>target</b>
<b>1</b>	Doctor	0,50	1
<b>2</b>	Doctor	0,50	0
<b>3</b>	Doctor	0,50	1
<b>4</b>	Doctor	0,50	0
<b>5</b>	Teacher	1	1
<b>6</b>	Teacher	1	1
<b>7</b>	Engineer	0,50	0
<b>8</b>	Engineer	0,50	1
<b>9</b>	Waiter	1	1
<b>10</b>	Driver	0	0

# Кодировки

---

- Label Encoding
- One-Hot Encoding
- Target Encoding (Рискуем переобучиться!)

<b>id</b>	<b>job</b>	<b>job_mean</b>	<b>target</b>
<b>1</b>	Doctor	0,50	1
<b>2</b>	Doctor	0,50	0
<b>3</b>	Doctor	0,50	1
<b>4</b>	Doctor	0,50	0
<b>5</b>	Teacher	1	1
<b>6</b>	Teacher	1	1
<b>7</b>	Engineer	0,50	0
<b>8</b>	Engineer	0,50	1
<b>9</b>	Waiter	1	1
<b>10</b>	Driver	0	0

# Кодировки

---

- Можно и не на таргет 😊

```
# функция возвращает значения нового признака
def code_mean(data, cat_feature, real_feature):
    return (data[cat_feature].map(data.groupby(cat_feature)[real_feature].mean()))

data['city_mean_income'] = code_mean(data, 'city', 'income')
data
```

	city	class	degree	income	city_mean_income
0	Moscow	A	1	10.2	8.5
1	London	B	1	11.6	10.2
2	London	A	2	8.8	10.2
3	Kiev	A	2	9.0	9.0
4	Moscow	B	3	6.6	8.5
5	Moscow	B	3	10.0	8.5
6	Kiev	A	1	9.0	9.0
7	Moscow	A	1	7.2	8.5

# Метрики для задачи регрессии

---

Mean Squared Error (среднеквадратичное отклонение):

$$\text{MSE}(a, X) = \frac{1}{\ell} \sum_{i=1}^{\ell} (a(x_i) - y_i)^2$$

# Метрики для задачи регрессии

---

Mean Squared Error (среднеквадратичное отклонение):

$$\text{MSE}(a, X) = \frac{1}{\ell} \sum_{i=1}^{\ell} (a(x_i) - y_i)^2$$

Значения будут получаться в квадратах! (например, предсказываем цену квартиры в рублях – MSE покажет отклонение в квадратах рублей).

# Метрики для задачи регрессии

---

Root Mean Squared Error (корень из среднеквадратичного отклонения):

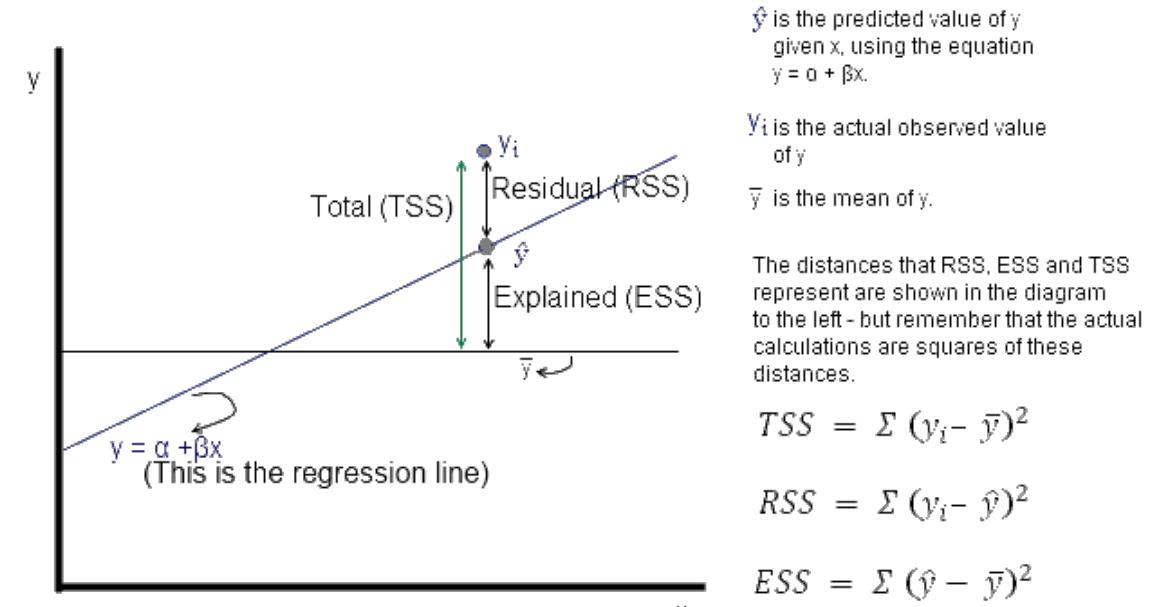
$$\text{RMSE}(a, X) = \sqrt{\frac{1}{\ell} \sum_{i=1}^{\ell} (a(x_i) - y_i)^2}$$

# Метрики для задачи регрессии

R-squared (коэффициент детерминации):

$$R^2(a, X) = 1 - \frac{\sum_{i=1}^{\ell} (a(x_i) - y_i)^2}{\sum_{i=1}^{\ell} (y_i - \bar{y})^2}$$

- Измеряет долю дисперсии, объясненную моделью.
- Чем ближе к 1, тем лучше модель объясняет данные
- Чем ближе к 0, тем ближе модель к константному предсказанию

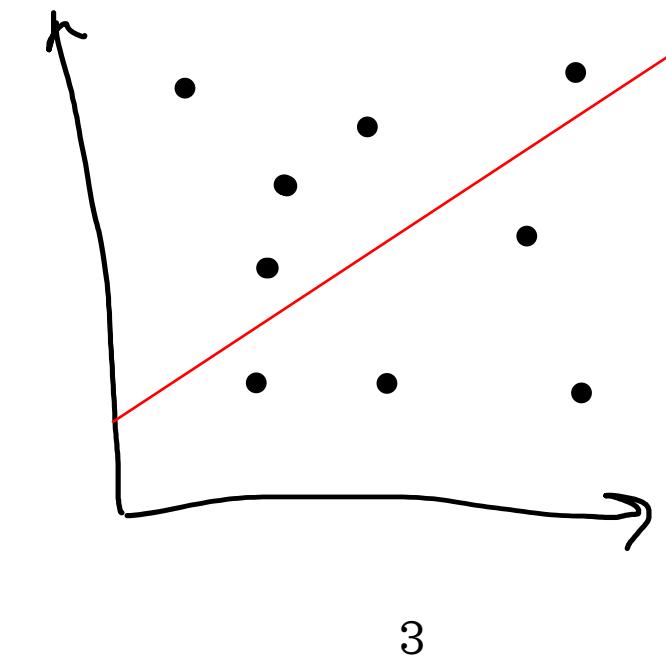
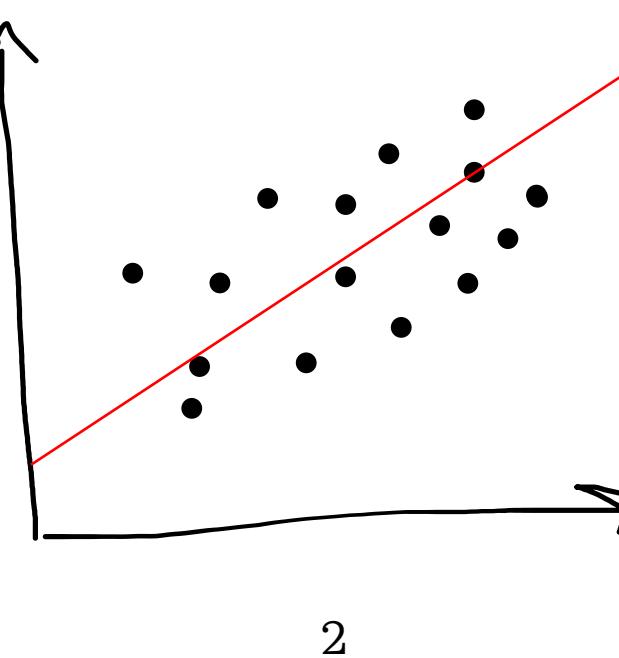
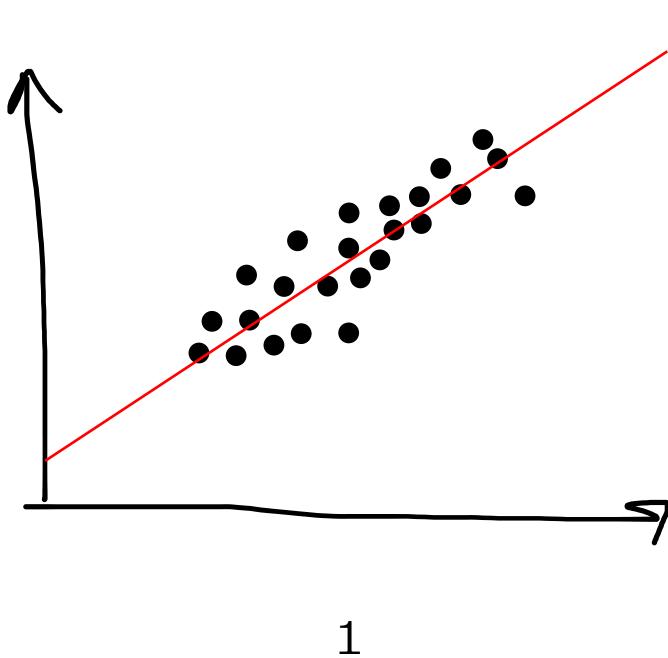


Источник: [URL](#)

# Метрики для задачи регрессии

---

R-squared (коэффициент детерминации)



# Метрики для задачи регрессии

---

Mean Absolute Error (среднее абсолютное отклонение):

$$\text{MAE}(a, X) = \frac{1}{\ell} \sum_{i=1}^{\ell} |a(x_i) - y_i|$$

# Метрики для задачи регрессии

---

Mean Absolute Error (среднее абсолютное отклонение):

$$\text{MAE}(a, X) = \frac{1}{\ell} \sum_{i=1}^{\ell} |a(x_i) - y_i|$$

- Менее чувствителен к выбросам, чем MSE
- Не дифференцируем

# Метрики для задачи регрессии

---

Mean Squared Logarithmic Error, MSLE (средняя логарифмическая ошибка):

$$L(y, a) = (\log(a + 1) - \log(y + 1))^2$$

# Метрики для задачи регрессии

---

Mean Squared Logarithmic Error, MSLE (средняя логарифмическая ошибка):

$$L(y, a) = (\log(a + 1) - \log(y + 1))^2$$

1. Подходит для задач с неотрицательной целевой переменной
2. Штрафуем за отклонения в порядке величин
3. Штрафует заниженные прогнозы сильнее, чем завышенные.

# Метрики для задачи регрессии

---

Mean Absolute Percentage Error, MAPE (средняя абсолютная процентная ошибка):

$$\text{MAPE}(a, X) = \frac{1}{l} \sum_{i=1}^l \frac{|y_i - a(x_i)|}{|y_i|}$$

- Ограничена (принимает значения от 0 до 1)
- Интерпретируема

# Метрики для задачи регрессии

---

Mean Absolute Percentage Error, MAPE (средняя абсолютная процентная ошибка):

$$\text{MAPE}(a, X) = \frac{1}{l} \sum_{i=1}^l \frac{|y_i - a(x_i)|}{|y_i|}$$

Недопрогноз:

$$y = 100 \quad \hat{y} = 80$$

Перепрогноз:

$$y = 60 \quad \hat{y} = 80$$

MAPE :

# Метрики для задачи регрессии

---

Symmetric Mean Absolute Percentage Error, SMAPE (симметричная средняя абсолютная процентная ошибка):

$$\text{SMAPE}(a, X) = \frac{1}{l} \sum_{i=1}^l \frac{|y_i - a(x_i)|}{(|y_i| + |a(x_i)|)/2}$$

Симметричная модификация MAPE.

# Метрики для задачи регрессии

---

Symmetric Mean Absolute Percentage Error, SMAPE (симметричная средняя абсолютная процентная ошибка):

$$\text{SMAPE}(a, X) = \frac{1}{l} \sum_{i=1}^l \frac{|y_i - a(x_i)|}{(|y_i| + |a(x_i)|)/2}$$

Недопрогноз:

$$y = 100 \quad \hat{y} = 80$$

Перепрогноз:

$$y = 60 \quad \hat{y} = 80$$

SMAPE:

# Регуляризация

---

- Эмпирическое наблюдение: если модель переобучилась, почти наверняка у нее будут очень большие веса ( $10^6, 10^7, \dots$ )
- **Идея:** давайте штрафовать за большие веса!

$$Q_\alpha(w) = Q(w) + \alpha R(w)$$

$\mathcal{L} \nearrow, 0$

# Регуляризация

---

- $\alpha$  – гиперпараметр
- Параметры настраиваются по обучающей выборке (например, веса в линейной регрессии)
- Гиперпараметры вводятся для улучшения качества модели на новых данных, их нельзя подбирать по обучающей выборке (почему?)
- Подбираем по отложенной выборке или кросс-валидации

# Регуляризация

---

Наиболее распространеными являются L2 и L1-регуляризаторы:

$$R(w) = \|w\|_2 = \sum_{i=1}^d w_i^2, \quad -\sqcup\sqcup$$

$$R(w) = \|w\|_1 = \sum_{i=1}^d |w_i|. \quad - \sqcup |$$

# Регуляризация

---

- Коэффициент  $\alpha$  называется параметром регуляризации и контролирует баланс между подгонкой под обучающую выборку и штрафом за излишнюю сложность. Значение параметра подбирается перебором под каждую задачу.
- Свободный коэффициент  $w_0$  нет смысла регуляризовывать — если мы будем штрафовать за его величину, то получится, что мы учитываем представления о близости целевой переменной к нулю и отсутствии необходимости в учёте её смещения.
- Особенно об этом следует помнить, если в выборке есть константный признак и коэффициент  $w_0$  обучается наряду с остальными весами. В этом случае исключаем константный признак из регуляризатора.

# Регуляризация

---

Квадратичный (или L2) регуляризатор достаточно прост в использовании в отличие от L1-регуляризатора, у которого нет производной в нуле. При этом L1-регуляризатор имеет интересную особенность: его использование приводит к занулению части весов.

А отбор признаков может быть полезен в ряде случаев:

- Некоторые признаки могут быть нерелевантны для задачи
- Чем меньше признаков, тем быстрее сможем обучать модель и делать предсказания

# Обучение линейной регрессии

---

Для MSE:

$$\vec{w} = (X^T X)^{-1} X^T \vec{y}$$

Проблемы:

- Обращение матрицы – трудоемкая операция (кубическая сложность от числа признаков)
- Для другого функционала (не MSE) аналитического решения может и не быть

# Градиентное обучение моделей

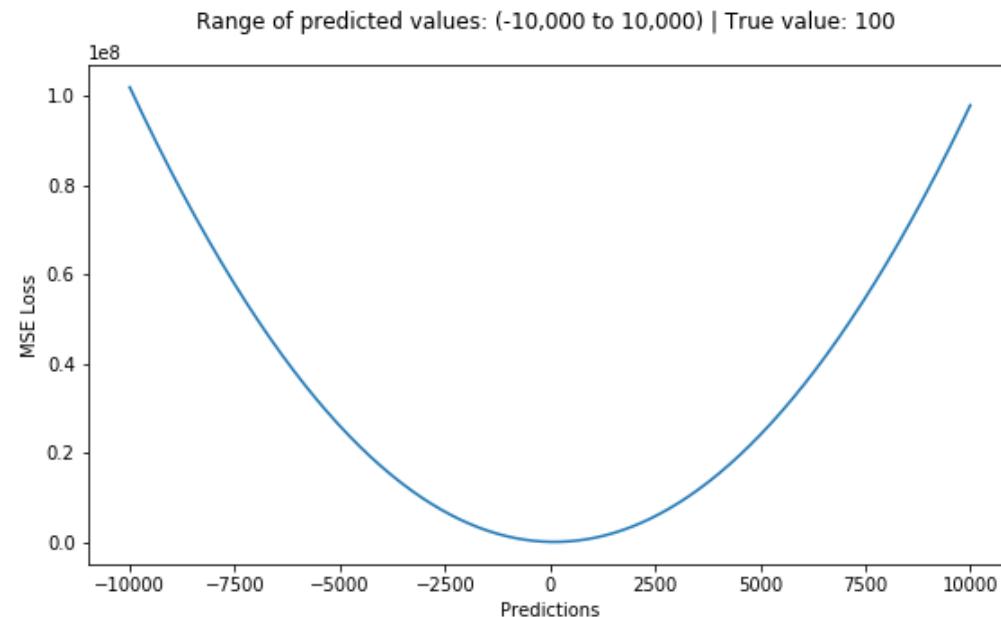
---

- **Градиент** – это вектор, в направлении которого функция быстрее всего растёт.
- Вектор градиента функции потерь обозначают  $\text{grad } Q$  или  $\nabla Q$
- **Антиградиент** (вектор, противоположный градиенту) – вектор, в направлении которого функция быстрее всего убывает.

# Градиентное обучение моделей

---

- Хотим: найти такие веса  $w$ , на которых достигается минимум функции ошибки
- График MSE, грубо говоря, является параболой



# Градиентный спуск

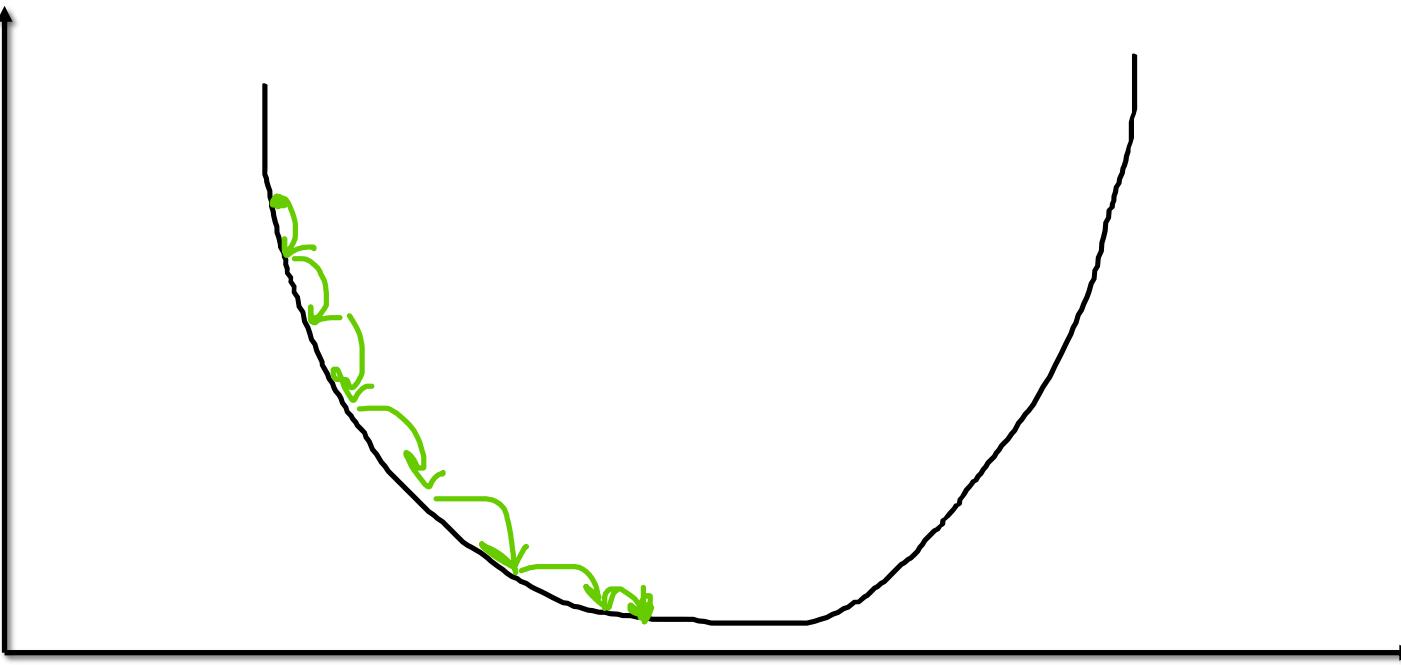
---

- Хотим: найти такие веса  $w$ , на которых достигается минимум функции ошибки
- График MSE, грубо говоря, является параболой

**Идея:** давайте на каждом шаге (на каждой итерации метода) двигаться в сторону антиградиента функции потерь! (=в направлении уменьшения ошибки)

# Градиентный спуск

---



# Градиентный спуск

---

1. Инициализируем веса  $w_0^{(0)}, w_1^{(0)}, w_2^{(0)}, \dots, w_n^{(0)}$
2. На каждом следующем шаге обновляем веса, сдвигаясь в направлении антиградиента функции потерь  $Q$ :

$$w_0^{(k)} = w_0^{(k-1)} - \nabla Q(w_0^{(k-1)}),$$

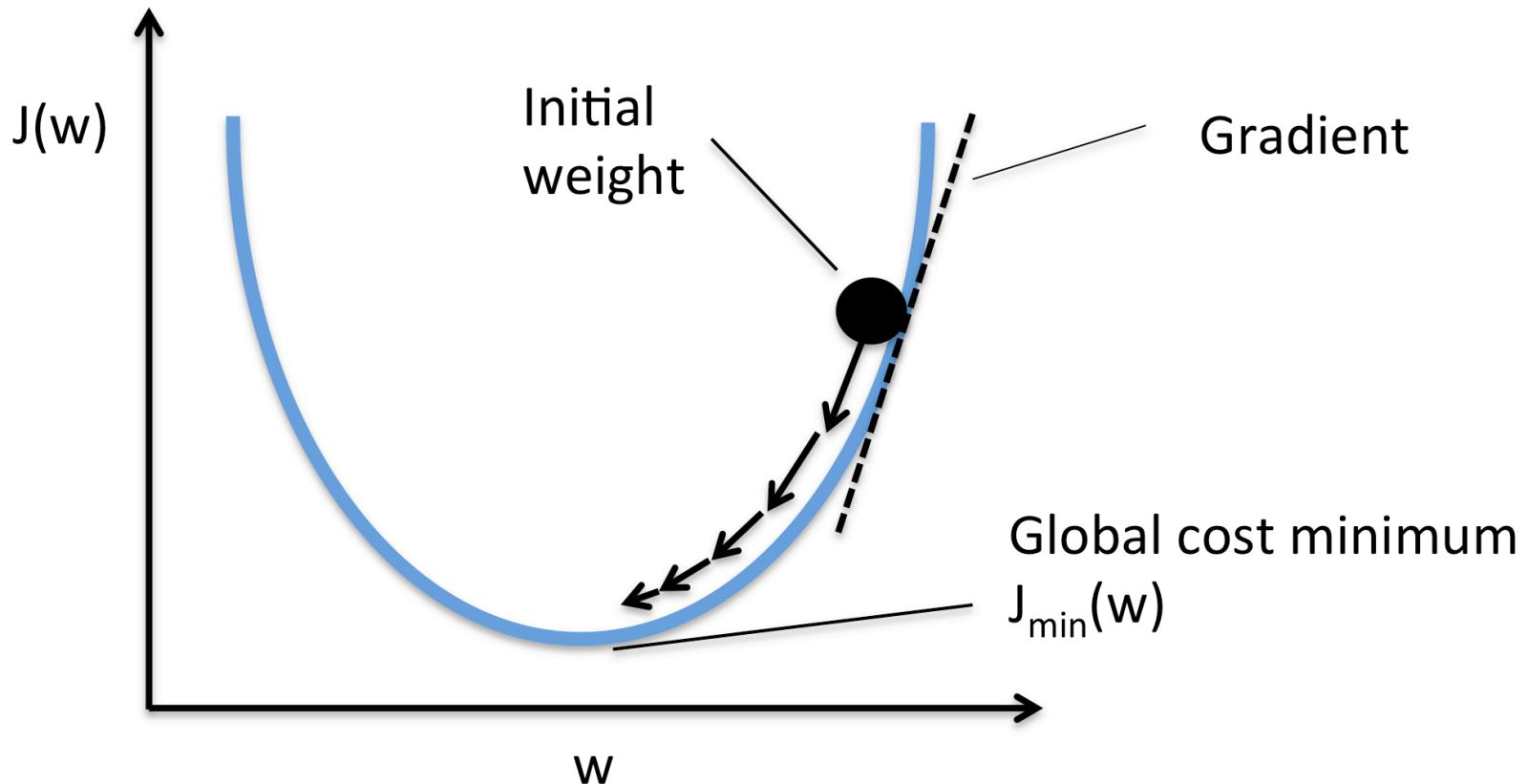
$$w_1^{(k)} = w_1^{(k-1)} - \nabla Q(w_1^{(k-1)}),$$

...

$$w_n^{(k)} = w_n^{(k-1)} - \nabla Q(w_n^{(k-1)}),$$

# Градиентный спуск

---



# Градиентный спуск

---

В векторном виде:

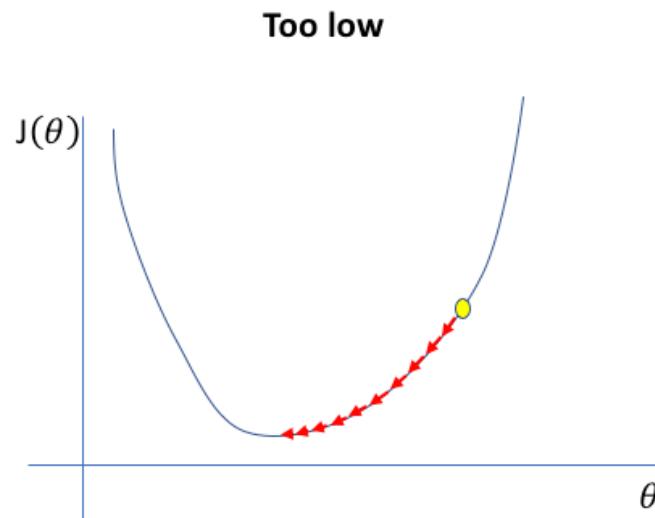
1. Инициализируем веса  $w^{(0)}$
2. На каждом следующем шаге обновляем веса по формуле:

$$\mathbf{w}^{(k)} = \mathbf{w}^{(k-1)} - \eta \nabla Q(\mathbf{w}^{(k-1)})$$

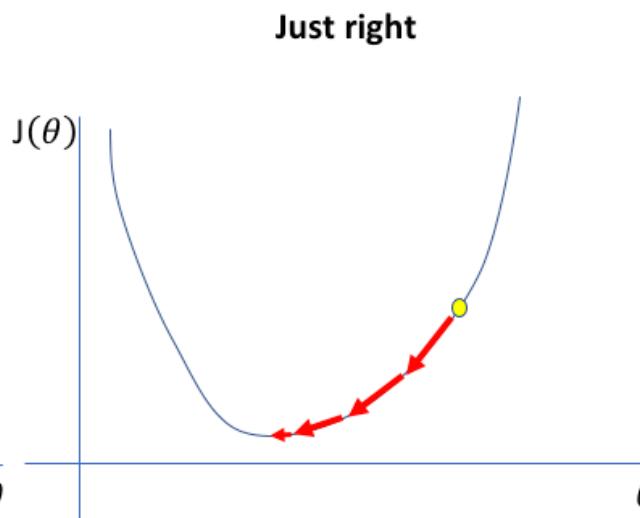
Выше мы добавили параметр learning rate (длина шага), который отвечает за скорость движения в сторону антиградиента.

# Градиентный спуск

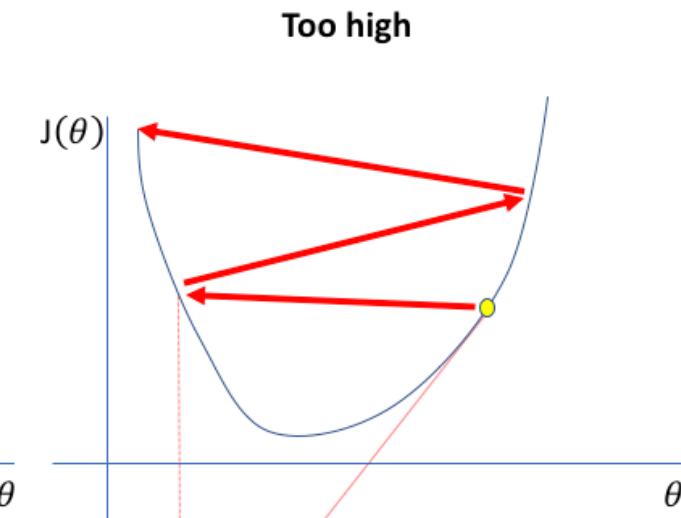
---



A small learning rate requires many updates before reaching the minimum point



The optimal learning rate swiftly reaches the minimum point



Too large of a learning rate causes drastic updates which lead to divergent behaviors

# Варианты инициализации весов

---

- Нулями
- Небольшими случайными значениями
- Обучение по маленькой случайной подвыборке объектов
- Мультистарт: многократный запуск из разных случайных начальных приближений и выбор лучшего решения

# Длина шага

---

- Константа:  $\eta_k = c$
- Монотонно убывающая по мере движения:  $\eta_k = \frac{1}{k}$
- Более сложные вариации. Ниже пример из пакета vowpal wabbit:

$$\eta_k = \lambda \left( \frac{s_0}{s_0 + k} \right)^p$$

# Когда останавливаться?

---

- При близости градиента к нулю

$$|\nabla Q(w^{(k-1)})| < \varepsilon$$

- Или при слишком малом изменении вектора весов на последней итерации

$$\Delta w = |w^{(k)} - w^{(k-1)}| < \varepsilon$$

Это называется **критериями останова**.

# Может застревать в локальных минимумах

---



Bob chillin at a local optima

# Градиентный спуск

---

Плюсы:

- Достаточно универсальный метод
- Легко реализуем

# Градиентный спуск

---

Минусы:

- Застревает в локальных минимумах
- Шаги могут быть медленными
- Неэффективен для больших выборок
- Неприменим для недифференцируемых функций

**Идея:** приближенно оценивать градиент

# Градиентный спуск

---

- Функционал обычно представим в виде суммы функций:

$$Q(w) = \frac{1}{\ell} \sum_{i=1}^{\ell} q_i(w)$$

- Тогда при применении метода градиентного спуска на каждом шаге мы будем считать градиент всей суммы (полный градиент):

$$\nabla_w Q(w) = \frac{1}{\ell} \sum_{i=1}^{\ell} \nabla_w q_i(w)$$

# Стохастический градиентный спуск

---

- Оценить градиент суммы функций можно градиентом одного случайно взятого слагаемого:

$$\nabla_w Q(w) \approx \nabla_w q_{i_k}(w)$$

- Обновлять веса тогда будем так:

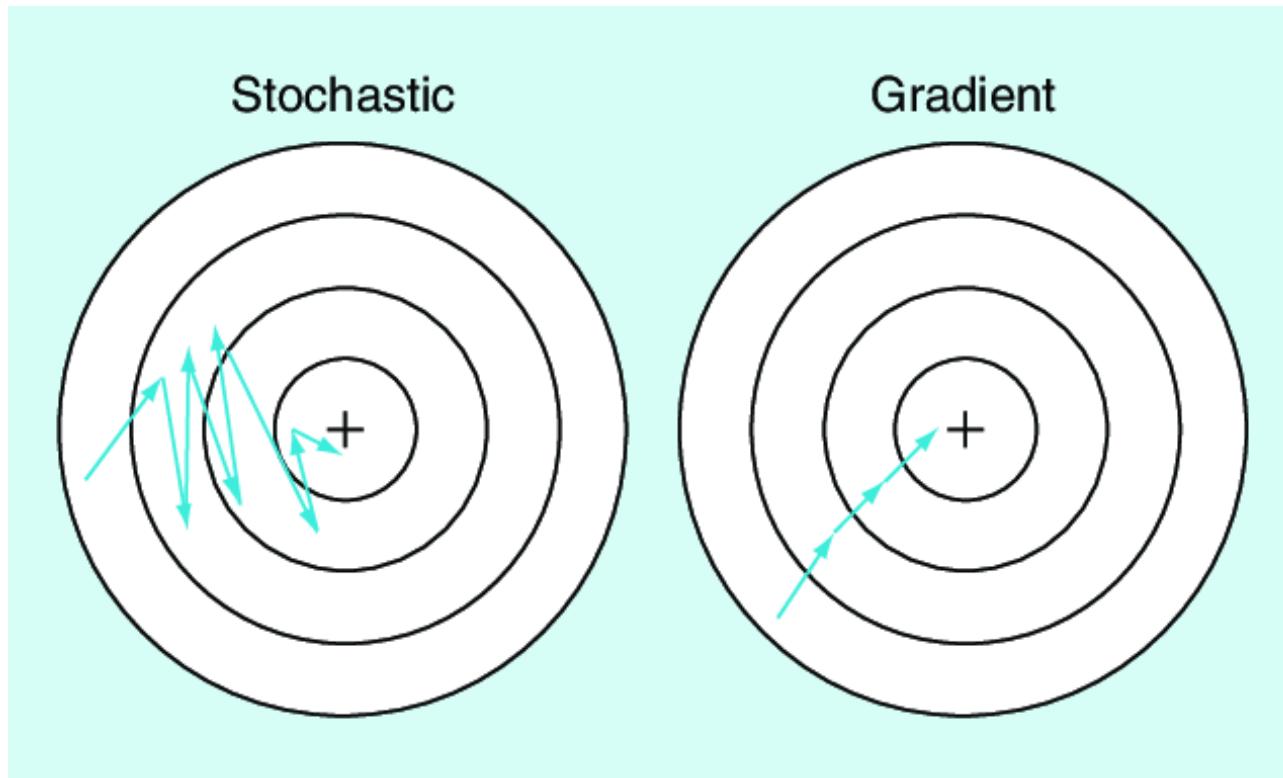
$$w^{(k)} = w^{(k-1)} - \eta_k \nabla q_{i_k}(w^{(k-1)})$$

- Еще один плюс: теперь на каждой итерации держим в памяти всего один объект из выборки

- Повысить точность метода можно взять чуть больше слагаемых – тогда получим mini-batch gradient descent

# Стохастический градиентный спуск

---



На картинке – линии уровня функции, которую оптимизируем.

**Линией уровня функции** называется линия (множество точек) на координатной плоскости, в которых функция принимает одинаковые значения.