# Градиентный спуск

Паточенко Евгений

НИУ ВШЭ

# План занятия

- Задача оптимизации
- Градиентный спуск
- Модификации градиентного спуска

# Задача оптимизации

*Напоминание*

На первом занятии мы уже упоминали термин «оптимизировать»:

# Задача оптимизации

*Напоминание*

На первом занятии мы уже упоминали термин «оптимизировать»:

**Обучение** — это по сути процесс подбора весов или параметров модели так, чтобы *оптимизировать* (в большинстве случаев — минимизировать) функцию потерь

# Задача оптимизации

*Определение*

**Оптимизация** — это задача нахождения экстремума (минимума или максимума) целевой функции в некоторой области конечномерного векторного пространства, ограниченной набором линейных и/или нелинейных равенств и/или неравенств

# Задача оптимизации

*Постановка задачи*

Среди элементов $x$, образующих множества $\mathrm{X}$ , найти такой элемент $x^*$, который доставляет минимальное (максимальное) значение $f(x^*)$ заданной функции $f(x)$ .

# Задача оптимизации

*Постановка задачи*

Среди элементов $x$, образующих множества $\mathrm{X}$ , найти такой элемент $x^*$, который доставляет минимальное (максимальное) значение $f(x^*)$ заданной функции $f(x)$ .

В простейшем случае линейной регрессии:

Элементы $x$, образующие множества $\mathrm{X}$ — веса модели

Функция $f(x)$ — среднеквадратичная ошибка

# Задача оптимизации

*Постановка задачи*

Среди элементов $x$, образующих множества $\mathrm{X}$ , найти такой элемент $x^*$, который доставляет минимальное (максимальное) значение $f(x^*)$ заданной функции $f(x)$

В простейшем случае линейной регрессии:

Элементы $x$, образующие множества $\mathrm{X}$ — веса модели

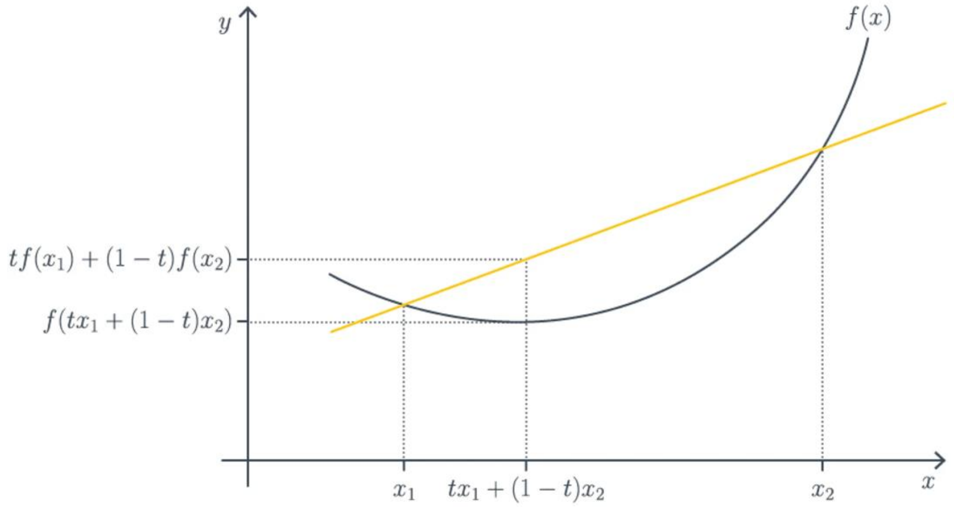
Функция $f(x)$ — среднеквадратичная ошибка

Почему оптимизация среднеквадратичной ошибки — простейший случай?

# Задача оптимизации

*Оптимизация MSE*

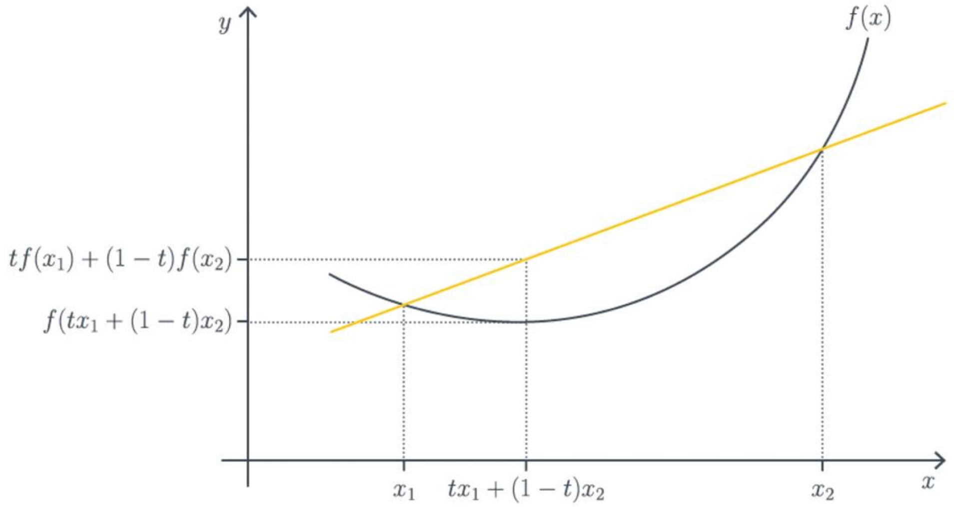
Функция среднеквадратичной ошибки — *выпуклая*



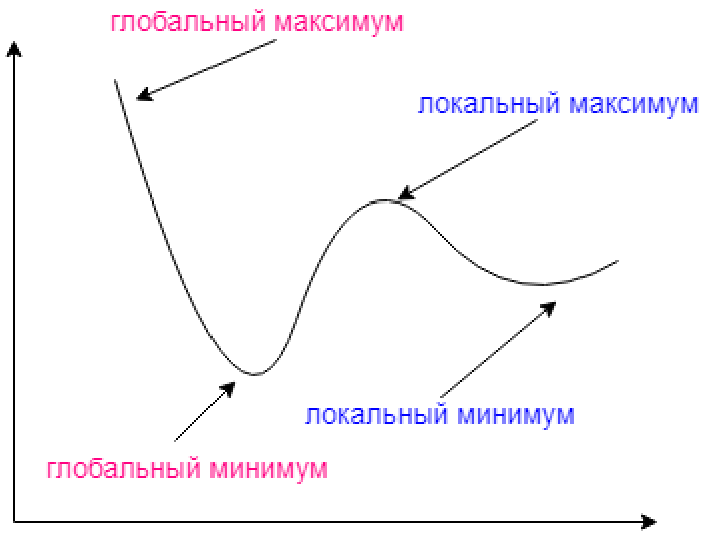
# Задача оптимизации

*Оптимизация MSE*

Функция среднеквадратичной ошибки — *выпуклая*




Для нас это означает, что у такой функции *локальный и глобальный минимумы совпадают* (в отличие от случая на картинке ниже)



# Задача оптимизации

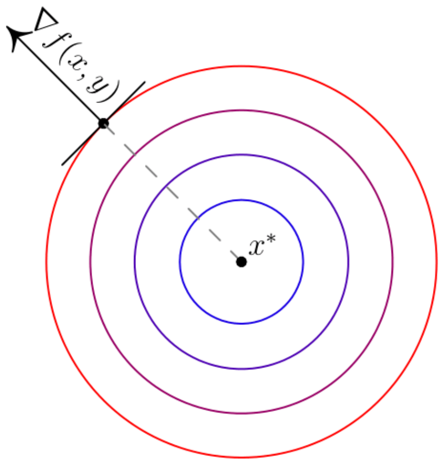
*Поиск минимума*

**Градиент** – это вектор, в направлении которого функция быстрее всего растет

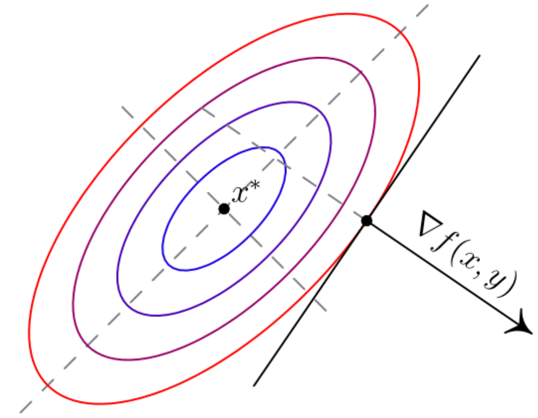
**Антиградиент** (вектор, противоположный градиенту) — вектор, в направлении которого функция быстрее всего убывает, при этом в общем случае он *не указывает точно на точку минимума функции*

Вектор градиента обозначают

$grad$ или $\nabla$




$f(x,y) = x^2 + y^2$




$f(x,y) = \frac{3}{2}(x-y)^2 + \frac{1}{3}(x+y)^2$

# Задача оптимизации

*Градиент*

Если $f$ — функция $n$ переменных $x_1,\ldots, x_n$, ее градиентом называется $n$-мерный вектор $\left(\frac{\partial f}{\partial x_1}, \ldots, \frac{\partial f}{\partial x_n}\right)$, компоненты которого равны частным производным $f$ по всем ее аргументам

Для одномерного пространства градиентом функции $f(x)$ будет $\frac{\partial f}{\partial x}$

# Градиентный спуск

*Определение*

**Градиентный спуск** — метод оптимизации первого порядка для нахождения локального экстремума (минимума или максимума) функции с помощью движения вдоль градиента

Методы оптимизации:

- прямые — требуют только вычислений целевой функции в точках приближений
- первого порядка — требуют вычисления первых частных производных функции
- второго порядка — требуют вычисления вторых частных производных, то есть гессиана целевой функции

# Градиентный спуск

*Алгоритм*

- Выбираем $x_0$ — начальную точку градиентного спуска.
- Каждую следующую точку выбираем по формуле:

$$x_{k+1} = x_k - \alpha \nabla f(x_k),$$

где $\alpha$ — это размер шага (learning rate)

- Проверяем критерий останова. Если не выполнен, повторяем предыдущий шаг. Выполнен — останавливаемся

# Градиентный спуск

*Алгоритм*

При обучении модели мы подбираем веса. Соответственно, формула

$$x_{k+1} = x_k - \alpha \nabla f(x_k),$$

Может быть записана так:

$$\beta_{k+1} = \beta_k - \alpha \nabla f(\beta_k)$$

То есть на каждом шаге для нахождения новой точки мы *обновляем веса*

# Градиентный спуск

*Инициализация весов*

# Градиентный спуск

*Инициализация весов*

- $\beta_k = 0, k = 1, \ldots, n$

# Градиентный спуск

*Инициализация весов*

- $\beta_k = 0, k = 1, \ldots, n$
- Небольшие случайные значения: $\beta_k \coloneqq \mathrm{random}(-\varepsilon, \varepsilon)$ из некоторого распределения

# Градиентный спуск

*Инициализация весов*

- $\beta_k = 0, k = 1, \dots, n$
- Небольшие случайные значения: $\beta_k \coloneqq \mathrm{random}(-\varepsilon, \varepsilon)$ из некоторого распределения
- Обучение по небольшой случайной подвыборке объектов

# Градиентный спуск

*Инициализация весов*

- $\beta_k = 0, k = 1, \ldots, n$
- Небольшие случайные значения: $\beta_k \coloneqq \mathrm{random}(-\varepsilon, \varepsilon)$ из некоторого распределения
- Обучение по небольшой случайной подвыборке объектов
- Мультистарт — многократный запуск из разных случайных начальных приближений и выбор лучшего решения

# Градиентный спуск

*Выбор градиентного шага (скорость обучения, learning rate)*

$\alpha$ — это гиперпараметр, который мы подбираем и который влияет на качество и скорость обучения. При неудачном выборе метод может либо слишком медленно сходиться, либо пропускать минимум

# Градиентный спуск

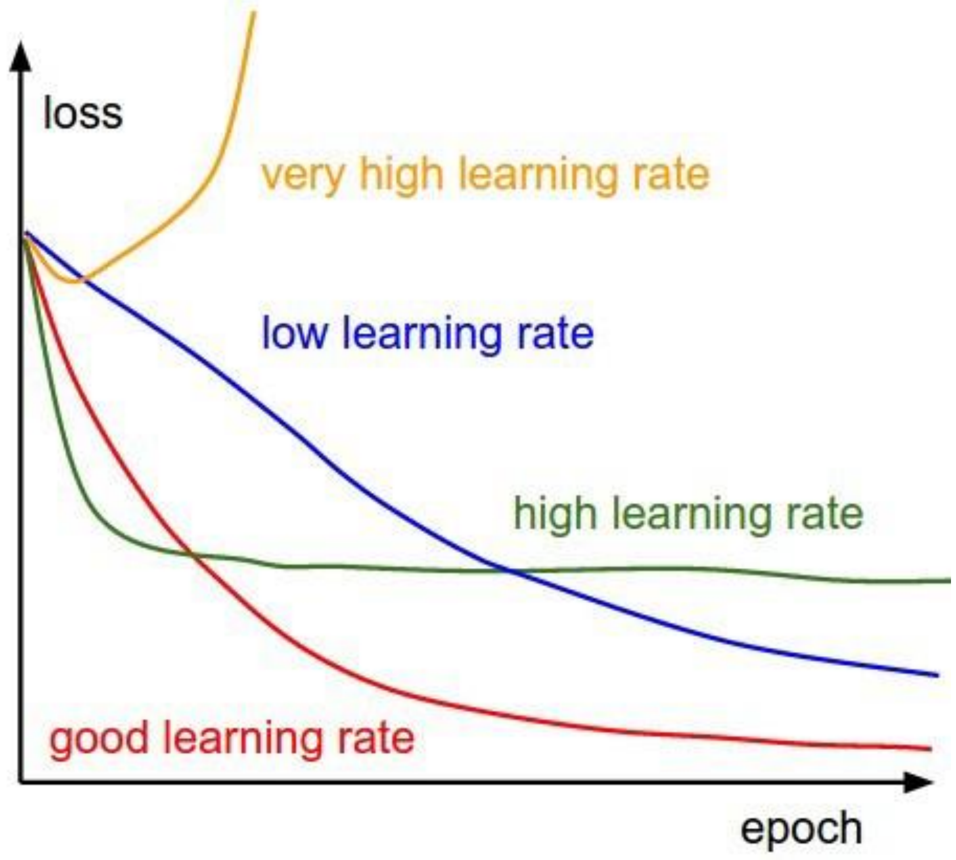
*Выбор градиентного шага (скорость обучения, learning rate)*

$\alpha$ — это гиперпараметр, который мы подбираем и который влияет на качество и скорость обучения. При неудачном выборе метод может либо слишком медленно сходиться, либо пропускать минимум

- Если хотим, чтобы получаемая точка минимума была как можно ближе к точному значению, выбираем как можно меньшую $\alpha$
- Если важнее скорость и хотим уменьшить количество шагов, выбираем как можно большую $\alpha$

# Градиентный спуск

*Выбор градиентного шага (скорость обучения, learning rate)*



# Градиентный спуск

*Критерии останова*

# Градиентный спуск

*Критерии останова*

- Остановка через заранее заданное число шагов

# Градиентный спуск

*Критерии останова*

- Остановка через заранее заданное число шагов
- Остановка на основе изменения функционала $\left|Q\left(\beta^{k+1}\right) - Q(\beta^{k})\right| < \varepsilon$

# Градиентный спуск

*Критерии останова*

- Остановка через заранее заданное число шагов
- Остановка на основе изменения функционала $\left|Q\left(\beta^{k+1}\right) - Q(\beta^k)\right| < \varepsilon$
- Остановка через изменение весов $\left\|\beta^{k+1} - \beta^k\right\| < \varepsilon$

# Градиентный спуск

*Недостаток градиентного спуска*

# Градиентный спуск

*Недостатки градиентного спуска*

- На каждом шаге мы вычисляем целую матрицу производных — по каждому весу от каждого объекта. Это затратно и по времени, и по памяти.

# Градиентный спуск

*Недостатки градиентного спуска*

- На каждом шаге мы вычисляем целую матрицу производных — по каждому весу от каждого объекта. Это затратно и по времени, и по памяти.
- Может застрять в локальном минимуме

# Модификации градиентного спуска

*Стохастический градиентный спуск (SGD)*

На каждом шаге выбираем один случайный (отсюда — «стохастический» в названии метода) объект и двигаемся в сторону антиградиента по нему

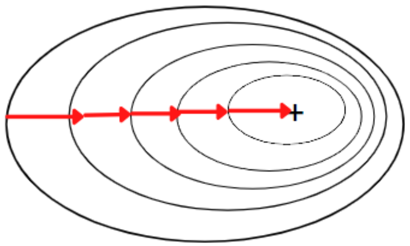
Такая модификация решает техническую проблему ресурсоемкости градиентного спуска, при этом вычисленный градиент, хоть и придет при правильно подобранном шаге обучения и начальной точке в точку минимума, может потребовать больше шагов, чтобы сойтись

# Модификации градиентного спуска

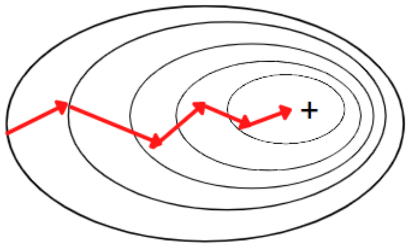
*Mini-batch SGD*

На каждом шаге выбираем подвыборку объектов (*batch size, например, 32, 64 и т. д.*) и двигаемся в сторону антиградиента функции потерь, вычисленной по батчу

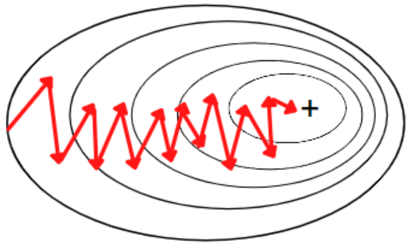
**Batch Gradient Descent**



**Mini-Batch Gradient Descent**



**Stochastic Gradient Descent**



# Модификации градиентного спуска

*Momentum*

Добавляем к градиентному шагу еще одно слагаемое так, чтобы использовать информацию о предыдущих градиентах:

$$x_{k+1} = x_k - \alpha \nabla f(x_k) + \eta_k (x_k - x_{k-1})$$

Такой подход, когда каждое следующее значение рекуррентно учитывает несколько предыдущих называют экспоненциальным сглаживанием.

Можно также  представить в виде двух параллельных процессов:

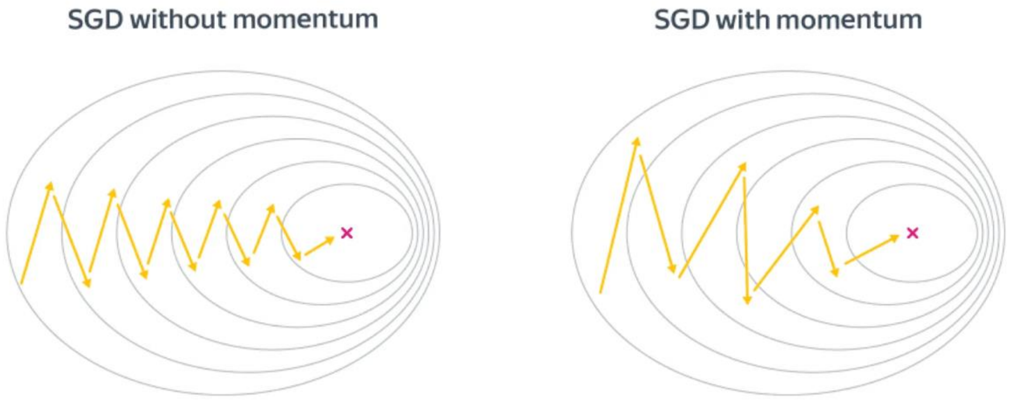
1. $v_{k+1} = \eta_k v_k - \alpha \nabla f(x_k)$
2. $x_{k+1} = x_k + v_{k+1}$

# Модификации градиентного спуска

*Momentum*

Преимущества:

- Удается сгладить направление каждого из шагов, что решает проблему частой смены направления шага (в том числе из-за небольшого батча) и как следствие решается проблема общей низкой скорости сходимости
- За счет инерции шага удается перешагивать некоторые локальные минимумы



# Модификации градиентного спуска

*Accelerated Gradient Descent (Nesterov Momentum)*

Модифицируем предыдущий метод и рассчитываем градиент не в текущей точке, а как будто бы в точке, в которую мы бы пошли, следуя импульсу

1. $v_{k+1} = \eta_k v_k - \alpha \nabla f(x_k + \eta_k v_k)$
2. $x_{k+1} = x_k + v_{k+1}$

Позволяет значительно повысить устойчивость и скорость сходимости в некоторых случаях (в частности, для выпуклых функций)

# Модификации градиентного спуска

*Adagrad (Adaptive Gradient)*

Адаптация стохастического градиентного спуска, позволяющая динамически менять learning rate

Зафиксируем $\alpha$ и запишем формулу обновления:

$$G_{k+1} = G_k + \left(\nabla f(x_k)\right)^2$$

$$x_{k+1} = x_k - \frac{\alpha}{\sqrt{G_{k+1} + \varepsilon}} \nabla f(x_k)$$

# Модификации градиентного спуска

*Adagrad (Adaptive Gradient)*

Возведение в квадрат и деления векторов покомпонентные, то есть мы подбираем размер шага для каждой координаты по отдельности. ε на практике оставляется дефолтными $1e - 8$.

Если мы вышли на плато по какой-то координате и соответствующая компонента градиента начала затухать, нельзя уменьшать размер шага слишком сильно, чтобы не остаться на плато, при этом в целом уменьшать его нужно, так как плато может содержать оптимум

Если градиент долго слишком большой, может быть нужно увеличить размер шага

# Модификации градиентного спуска

*RMSProp (Root Mean Square Propagation)*

Модификация Adagrad, в которой мы не складываем нормы градиентов, а усредняем их в скользящем режиме:

$$G_{k+1} = \gamma G_k + (1 - \gamma)\left(\nabla f(x_k)\right)^2$$

$$x_{k+1} = x_k - \frac{\alpha}{\sqrt{G_{k+1} + \varepsilon}} \nabla f(x_k)$$

В этом методе так же учитывается история градиентов, но размер шага уменьшается не так быстро

# Модификации градиентного спуска

*Adam (ADAptive Momentum)*

Алгоритм, который считается решением по умолчанию и включает в себя:

— Экспоненциальное сглаживание для градиента (Momentum)

— Вычисление градиента в точке, определяемой инерцией (Nesterov Momentum)

— Нормировку градиента для адаптивного регулирования шага по разным его компонентам (Adagrad + RMSProp)

При использовании Adam очень важно правильно подобрать гиперпараметр $\alpha$. Чаще всего начинают со значения $3e-4$

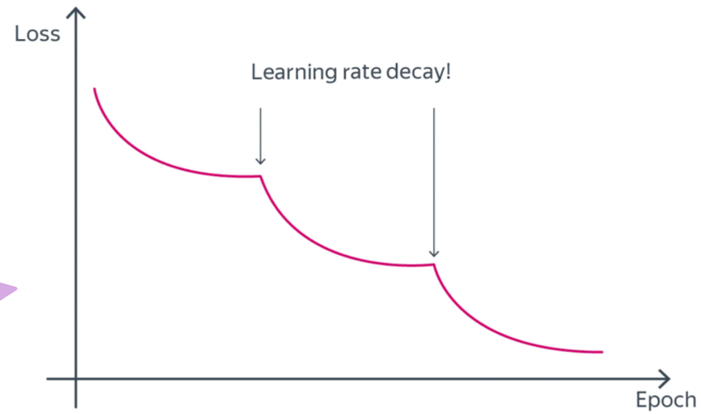
# Еще немного про шаг обучения

*Расписания*

Часто шаг обучения понижают итеративно. Его можно понижать:

— Каждые $n$ итераций (ссылка на документацию LRScheduler в Pytorch)

— При выходе функции потерь на плато




— Сделав сначала warmup (первые *warmup_steps* шагов происходит линейный рост lr, затем он начинает уменьшаться как $\frac{1}{\sqrt{t}}$, где t — число итераций