Градиентный спуск

и метод обратного распространения ошибки

Мария Макарова
Senior Data Analyst, Playrix

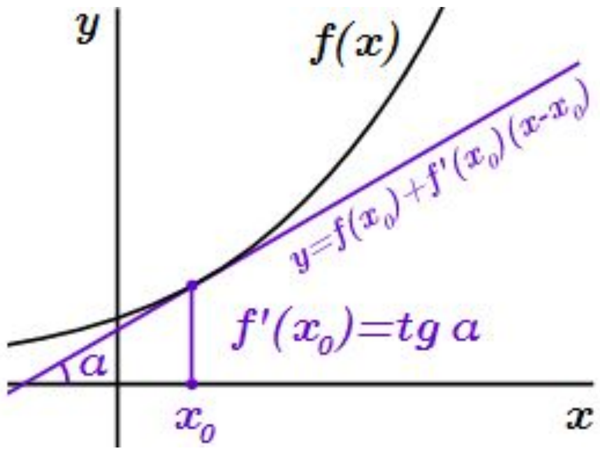
Производная функции одной переменной

- Скорость возрастания функции
- Приращение функции на единицу приращения ее аргумента
- Тангенс угла наклона касательной в заданной точке
- Отрицательная – если функция убывает

$$f(x) = e^x + 1$$

$$f'(x) = e^x$$

$$x_0 = 1, f'(x_0) = e$$



Производная функции нескольких переменных

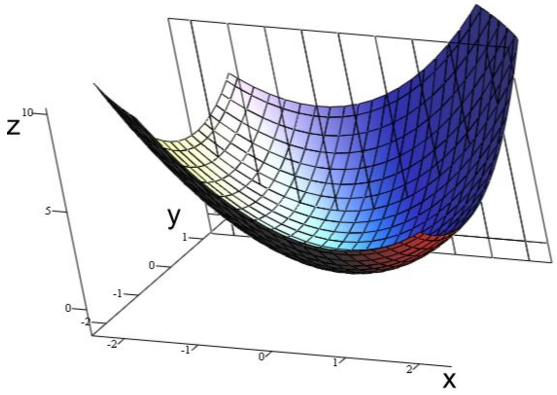
- Частная производная – скорость возрастания функции относительно заданной переменной

$$z = e^x + 2e^y + 1$$

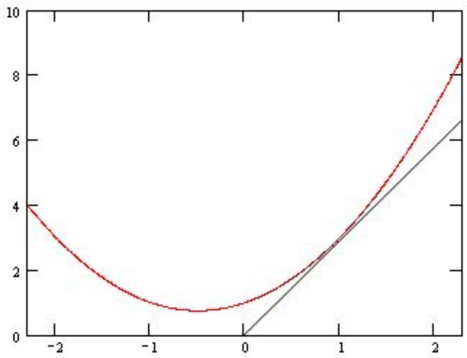
$$\frac{\partial z}{\partial x} = e^x, \frac{\partial z}{\partial y} = 2e^y$$

$$x_0 = 1, \frac{\partial z}{\partial x}|_{x=x_0} = e$$

Заданная функцией **z(x, y)** поверхность



Сечение при фиксированном значении **y**



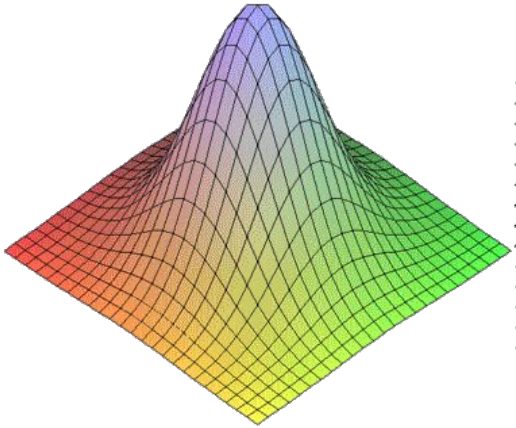
Градиент

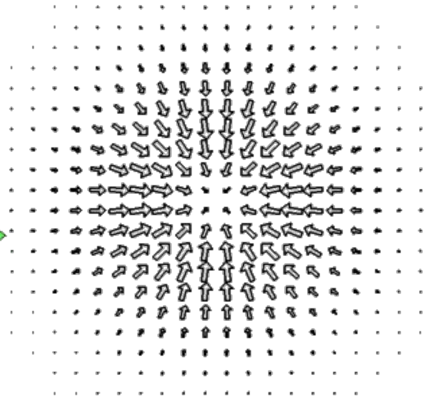
- Градиент – вектор направления наискорейшего роста функции
- Как изменить аргументы функции, чтобы ее значение выросло?

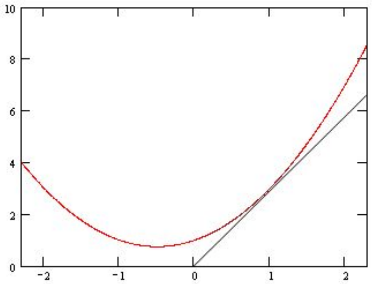
$$f = f(x_1, \ldots, x_n)$$

$$\nabla f = \left(\frac{\partial f}{\partial x_1}, \ldots, \frac{\partial f}{\partial x_n}\right)$$

- **Анти**градиент – вектор направления наискорейшего **убывания** функции

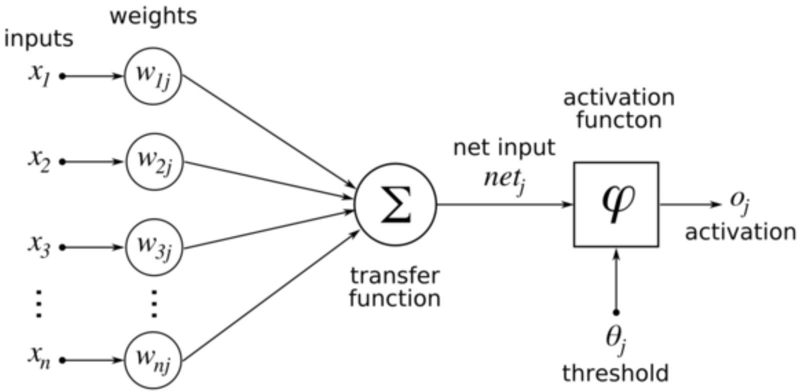






Зачем всё это нужно?

- Нейронная сеть – сложная дифференцируемая функция многих переменных (весов)
- Обучение нейросети – подбор оптимальных весов
- Оптимальных = таких, чтобы минимизировать какой-нибудь функционал
- Пример функционала: MSE, MAE, LogLoss и т. д.
- Приходим к задаче оптимизации (*аналитически в общем случае не решается => **используем численные методы***)
- Как оптимизировать – **градиентный спуск**



Градиентный спуск (GD - Gradient Descent)

- Проблема оптимизации

$$L(\omega) = \frac{1}{n}\sum_{i=0}^{n} L(\omega, x_i, y_i) \rightarrow \min_{\omega}$$

- Градиент указывает направление максимального роста функции

$$\nabla L(\omega) = \left(\frac{\partial L}{\partial \omega_1}, \ldots, \frac{\partial L}{\partial \omega_n}\right)$$

- Двигаемся в противоположную сторону итеративно с некоторым шагом

$$\omega_t = \omega_{t-1} - \eta \nabla L(\omega_{t-1})$$

скорость обучения (learning rate)

... пока не выполнится некоторое условие останова

Градиентный спуск (GD - Gradient Descent)

- Задача оптимизации

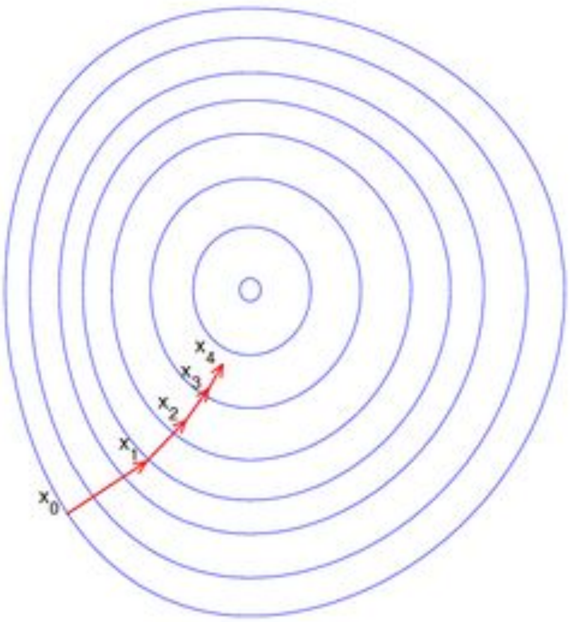
$$L(\omega) = \frac{1}{n}\sum_{i=0}^{n} L(\omega, x_i, y_i) \to \min_{\omega}$$

- Инициализируем веса (например, нулями)
- Считаем градиент по всей выборке

$$g_t = \frac{1}{n}\sum_{i=0}^{n} \nabla L(\omega_{t-1}, x_i, y_i)$$

- Обновляем веса

$$\omega_t = \omega_{t-1} - \eta_t \cdot g_t$$



Критерии останова

- Веса меняются очень слабо

$$||\omega_t - \omega_{t-1}|| < \epsilon$$

- Вектор градиента меняется очень слабо

$$||\nabla L(\omega_t)|| < \epsilon$$

- В глубинном обучении используется другой вариант

Останавливаемся, когда ошибка на валидационной выборке начинает расти

Пример (линейная регрессия)

- Функционал потерь

$$L(\omega) = \frac{1}{n}\sum_{i=0}^{n}(y_i - x_i^T\omega)^2 \to \min_{\omega}$$

- Считаем градиент по всей выборке

$$\nabla L(\omega) = -\frac{2}{n}\sum_{i=0}^{n}(y_i - x_i^T\omega)x_i$$

- Делаем шаг по направлению антиградиента

$$\omega_t = \omega_{t-1} + \frac{2\eta}{n}\sum_{i=0}^{n}(y_i - x_i^T\omega)x_i$$

Считать по всей выборке на каждом шаге **дорого**

- На каждом шаге вычисления за O(n)

Стохастический градиентный спуск (SGD)

- Задача оптимизации

$$L(\omega) = \frac{1}{n}\sum_{i=0}^{n} L(\omega, x_i, y_i) \to \min_{\omega}$$

- Инициализируем веса (например, нулями)
- Случайным образом выбрали из выборки один объект с индексом *i*
- Считаем градиент по одному объекту

$$g_t = \nabla L(\omega_{t-1}, x_i, y_i)$$

- Обновляем веса

$$\omega_t = \omega_{t-1} - \eta_t \cdot g_t$$

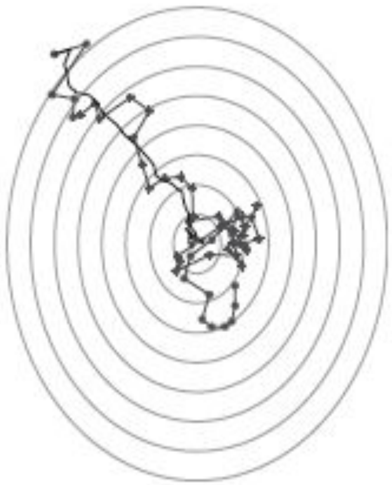
- На каждом шаге вычисления за **O(1)**

Стохастический градиентный спуск (SGD)

- Даже в точке оптимума градиент по одному объекту не будет нулевым, и мы уйдем в стохастическое блуждание
- Поэтому скорость обучения должна зависеть от номера шага и со временем уменьшаться
- Есть исследования, где показано, что скорость обучения должна удовлетворять следующим условиям

$$\sum_{t=0}^{\infty} \eta_t = \infty \qquad \sum_{t=0}^{\infty} \eta_t^2 < \infty$$

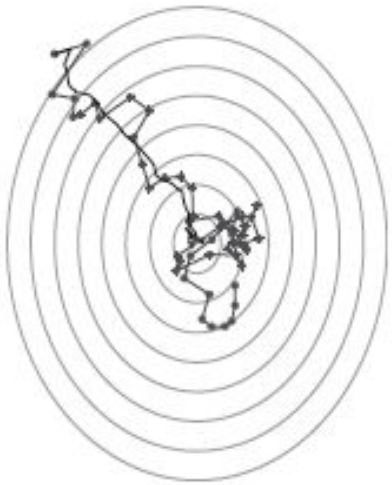
?



Стохастический градиентный спуск (SGD)

- Даже в точке оптимума градиент по одному объекту не будет нулевым, и мы уйдем в стохастическое блуждание
- Поэтому скорость обучения должна зависеть от номера шага и со временем уменьшаться
- Есть исследования, где показано, что скорость обучения должна удовлетворять следующим условиям

$$\sum_{t=0}^{\infty} \eta_t = \infty \qquad \sum_{t=0}^{\infty} \eta_t^2 < \infty \qquad \rightarrow \eta_t = \frac{1}{t}$$

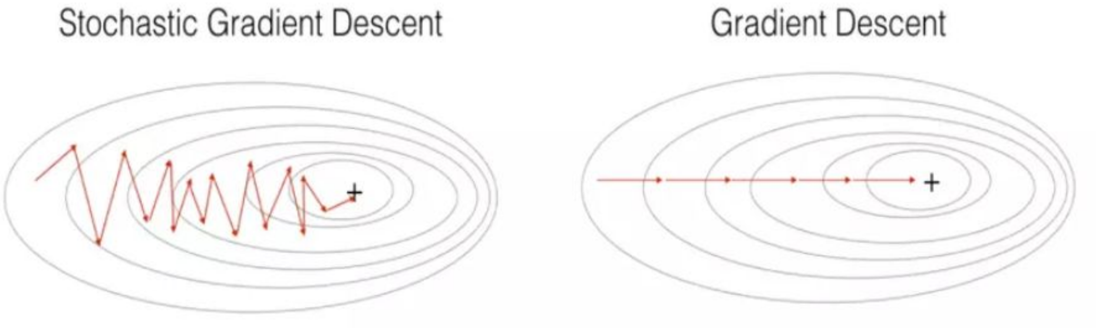


- На практике все равно делают по-другому

$$\eta_t = \frac{0.1}{t^{0.3}}$$

GD vs SGD

- Для GD и SGD нет гарантии того, что мы сойдемся к глобальному минимуму (можем упасть и остаться в локальном минимуме)
- SGD работает быстрее: O(1) < O(n)
- Движение по SGD очень зашумленное, но это помогает реже застревать в локальных минимумах



Mini-batch SGD

- Задача оптимизации

$$L(\omega) = \frac{1}{n}\sum_{i=0}^{n} L(\omega, x_i, y_i) \rightarrow \min_{\omega}$$

- Инициализируем веса (например, нулями)
- Случайным образом выбрали из выборки **несколько (m) объектов:** $m < n$
- Считаем градиент по этим объектам

$$g_t = \frac{1}{m}\sum_{i=0}^{m} \nabla L(\omega_{t-1}, x_i, y_i)$$

- Обновляем веса

$$\omega_t = \omega_{t-1} - \eta_t \cdot g_t$$

- На каждом шаге вычисления за **O(m)**

Mini-batch SGD

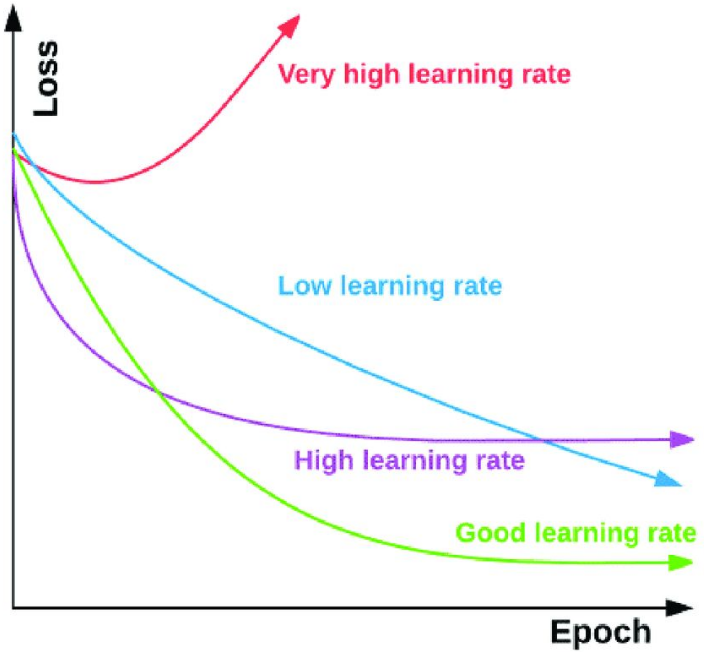
- Быстрее, чем классический GD
- Стабильнее, чем SGD
- За счет векторизации так же эффективен, как шаг по одному объекту

Как выбрать размер батча?

- Обычно это десятки или сотни наблюдений
- Есть исследования, где показано, что в некоторых задачах CV эффективно брать от 2 до 32
- *Не обязательно брать степень двойки – это миф*
- Если используем CPU, сильно много не ставим
- Если используем GPU, ставим так, чтобы использовать GPU на 100%

Скорость обучения

- **Маленькая скорость** – очень медленно идем к оптимуму
- **Большая скорость** – доходим до окрестности оптимума и начинаем в ней блуждать
- **Слишком большая скорость** – вообще не доходим до окрестности оптимума, поскольку перескакиваем через него за счет больших шагов

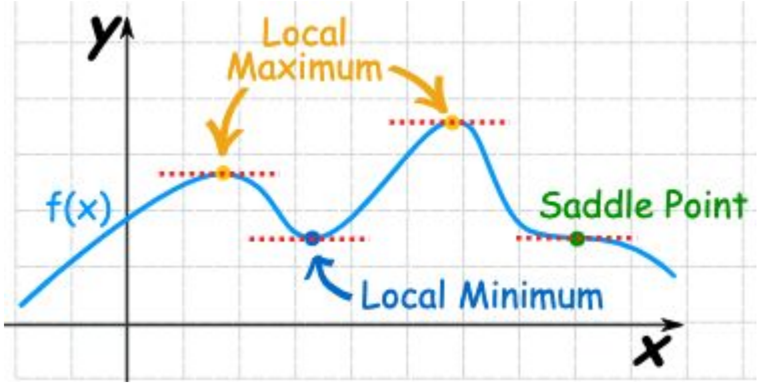


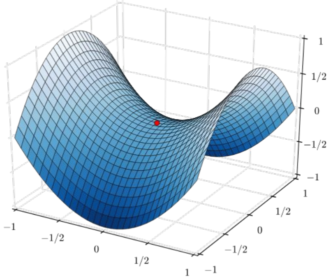
Какие могут возникнуть проблемы

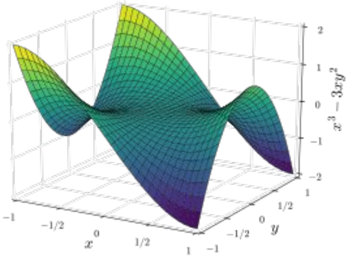
- Можем остаться в локальном минимуме
- Можем попасть на седловую точку и там остаться

Что делать?

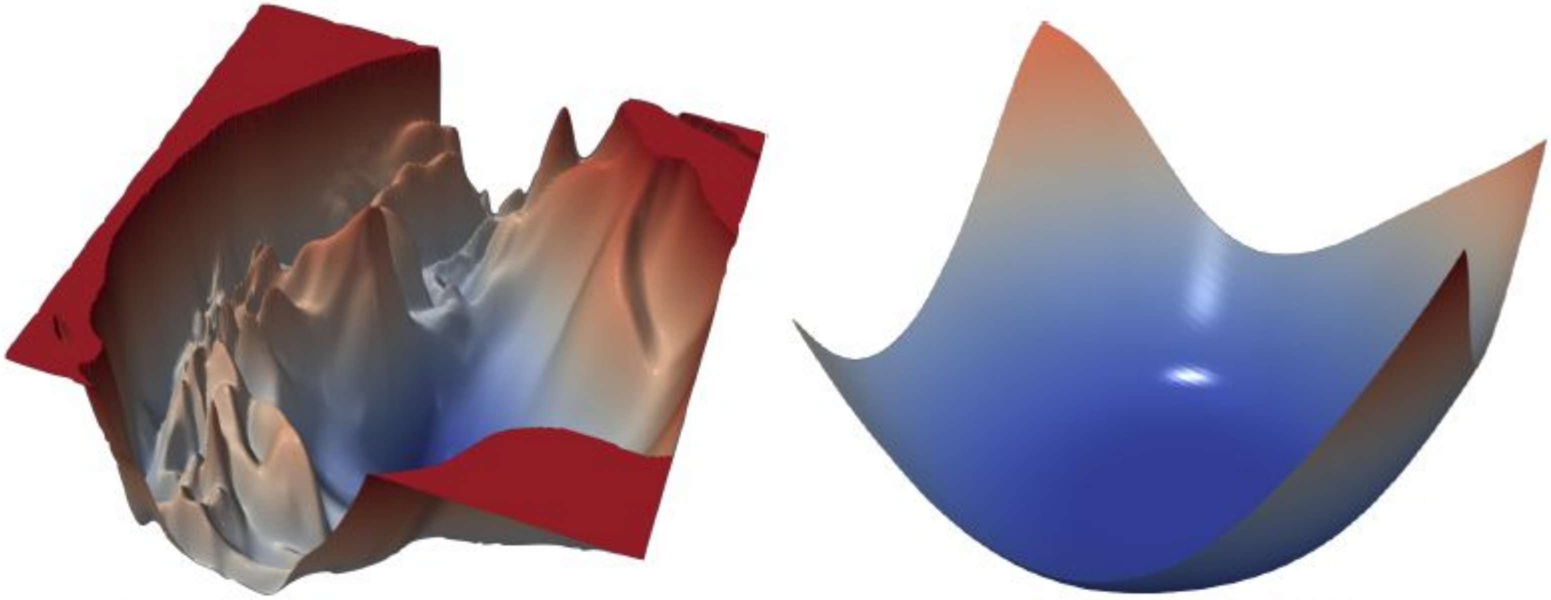
- Сам Mini-Batch подход частично может эти проблемы решить за счет случайности подвыборки
- Циклическое обучение – скорость градиентного спуска меняем циклически, например, по косинусу
- Мультистарт – параллельно запускаем обучение из нескольких стартовых точек







Ландшафт функции потерь (Loss landscape)



(a) ResNet-110, no skip connections (b) DenseNet, 121 layers

Статья: https://arxiv.org/pdf/1712.09913.pdf

Mini-batch SGD

Задача оптимизации

$$L(\omega) = \frac{1}{n} \sum_{i=0}^{n} L(\omega, x_i, y_i) \rightarrow \min_{\omega}$$

Градиент по мини-батчу

$$g_t = \frac{1}{m} \sum_{i=0}^{m} \nabla L(\omega_{t-1}, x_i, y_i)$$

Обновляем веса

$$\omega_t = \omega_{t-1} - \eta_t \cdot g_t$$

Используем информацию только о градиенте на текущем шаге **t**, а вдруг до этого мы шагнули лучше?

Momentum SGD

На текущем шаге **t** сам градиент считаем как обычно

$$g_t = \frac{1}{m}\sum_{i=0}^{m} \nabla L(\omega_{t-1}, x_i, y_i)$$

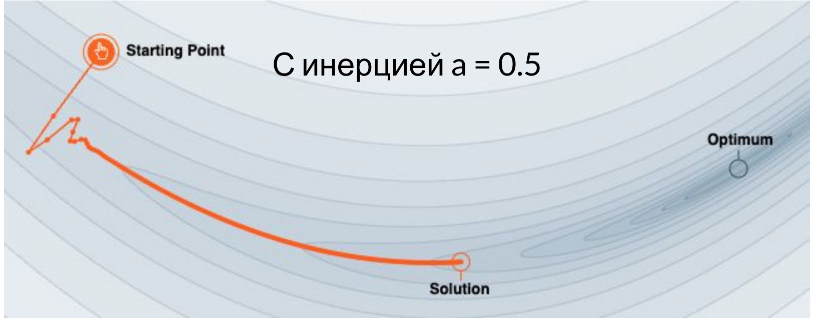
Раньше все градиенты с предыдущих шагов мы забывали, **теперь будем их запоминать**

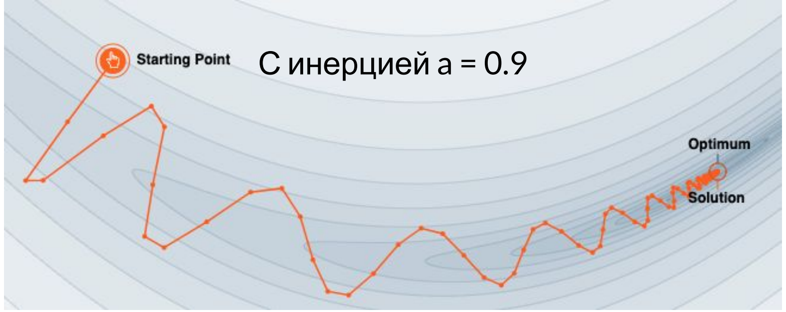
$$h_t = \alpha \cdot h_{t-1} + \eta \cdot g_t$$

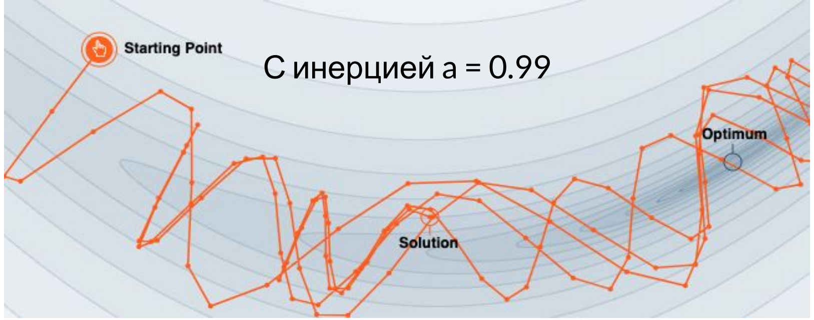
$$\omega_t = \omega_{t-1} - h_t$$

- Движение с инерцией
- Нет резких изменений направлений
- Обычно коэффициент инерции берут **a = 0.9**

Momentum SGD







Статья: https://distill.pub/2017/momentum/

Momentum SGD

На текущем шаге **t** сам градиент считаем как обычно

$$g_t = \frac{1}{m}\sum_{i=0}^{m} \nabla L(\omega_{t-1}, x_i, y_i)$$

Шаг с учетом инерции (память о предыдущих шагах – учимся на своих ошибках)

$$h_t = \boxed{\alpha \cdot h_{t-1}} + \eta \cdot g_t$$

$$\omega_t = \omega_{t-1} - h_t$$

На эту часть слагаемого мы точно уже шагнем. Давайте тогда **сначала шагнем**, а **затем посчитаем градиент**.

Nesterov Momentum SGD

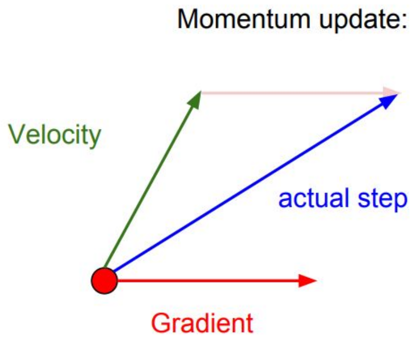
На текущем шаге **t** градиент считаем в точке с учетом предстоящего сдвига

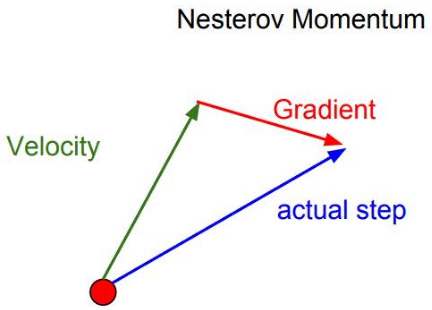
$$g_t = \frac{1}{m} \sum_{i=0}^{m} \nabla L(\omega_{t-1} - \alpha h_{t-1}, x_i, y_i)$$

Шаг с учетом инерции (память о предыдущих шагах – учимся на своих ошибках)

$$h_t = \alpha \cdot h_{t-1} + \eta \cdot g_t$$

$$\omega_t = \omega_{t-1} - h_t$$





Разная скорость обучения

- Во время обучения оптимизируем много параметров – некоторые могут раньше оказаться в окрестности оптимума, и тогда по этим координатам нужно шагать медленнее
- Почему бы не изменять параметры с разной скоростью?
  - в зависимости от номера шага
  - в зависимости от характера изменения значения самого параметра
- Шаг изменения должен быть меньше у параметров с большей изменчивостью и больше – у параметров с меньшей изменчивостью
- **Адаптивные методы градиентного спуска**

AdaGrad (Adaptive Gradient)

Накапливаем для каждого параметра что-то наподобие дисперсии

$$G_{j,t} = G_{j,t-1} + g_{jt}^2$$

Обновляем веса со своей скоростью для каждого параметра

$$\omega_{j,t} = \omega_{j,t-1} - \frac{\eta}{\sqrt{G_{j,t} + \epsilon}} g_{jt}$$

обычно 0.01 для всех параметров

“как бы” стандартное отклонение

Итого: чем больше до этого шагали, тем медленнее будем шагать дальше

Проблема: **G** с каждым шагом увеличивается, значит, обучение может рано остановиться

RMSprop

Накапливаем для каждого параметра что-то наподобие дисперсии, **но с весом**

$$G_{j,t} = \alpha \cdot G_{j,t-1} + (1 - \alpha) \cdot g_{jt}^2$$

Обновляем веса со своей скоростью для каждого параметра

$$\omega_{j,t} = \omega_{j,t-1} - \frac{\eta}{\sqrt{G_{j,t} + \epsilon}} g_{jt}$$

обычно 0.01 для всех параметров

“как бы” стандартное отклонение

alpha обычно 0.9

Итого: скорость обучения адаптируется к последнему сделанному шагу, теперь нет бесконтрольного роста **G**

Adam (Adaptive Moment Estimation)

Комбинируем Momentum и адаптивную скорость обучения

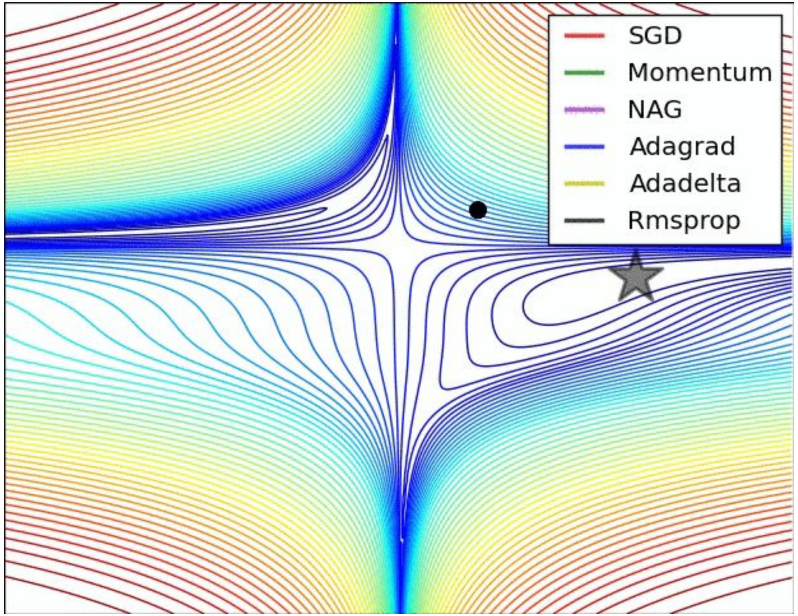
$$h_{j,t} = \frac{\beta_1 \cdot h_{j,t-1} + (1-\beta_1)\cdot g_{jt}}{1-\beta_1^t}$$

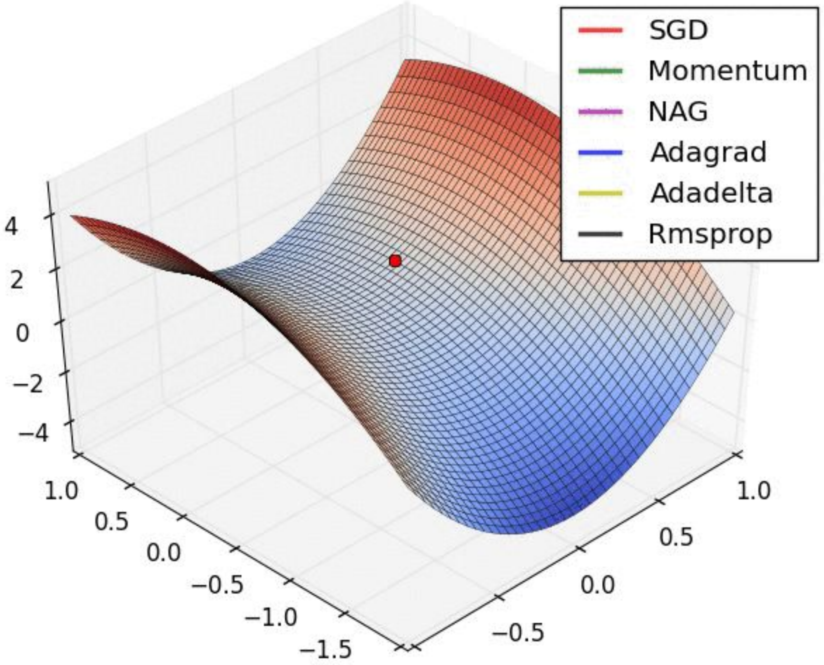
$$G_{j,t} = \frac{\beta_2 \cdot G_{j,t-1} + (1-\beta_2)\cdot g_{jt}^2}{1-\beta_2^t}$$

$$\omega_{j,t} = \omega_{j,t-1} - \frac{\eta_t}{\sqrt{G_{j,t}+\epsilon}} h_{jt}$$

- Фактически, **h** и **G** – это оценки первого и второго моментов для стохастического градиента. Знаменатель в этих оценках нужен, чтобы оценка была несмещенной (это можно доказать).
- Если в **G** копить настоящую дисперсию, то получим AdaBelief (2020)
  https://github.com/juntang-zhuang/Adabelief-Optimizer
  https://juntang-zhuang.github.io/adabelief/

Модификации градиентного спуска





Статья: https://www.ruder.io/optimizing-gradient-descent/

Нейронная сеть

Нейронная сеть – это некоторая сложная функция нескольких переменных (forward propagation)

$$X \to X \cdot W_1 \to f(X \cdot W_1) \to f(X \cdot W_1) \cdot W_2 \to \ldots \to y$$

Функция потерь

$$Loss = \frac{1}{2}\left(y_{true} - y\right)^2$$

Все слои дифференцируемы (мы такими их делаем), поэтому можно посчитать производные по всем
г

$$\nabla L(W) = \left(\frac{\partial L}{\partial W_1}, \ldots, \frac{\partial L}{\partial W_n}\right)$$

Обучаем градиентным спуском

В чем подвох?

Нейронная сеть

Для примера упростим нашу нейронную сеть и перепишем функцию потерь

$$X \to X \cdot W_1 \to f(X \cdot W_1) \to f(X \cdot W_1) \cdot W_2 \to y$$

$$L(W_1, W_2) = \frac{1}{2}\left(y_{true} - f(X \cdot W_1) \cdot W_2\right)^2$$

Половина успеха заключается в том, чтобы правильно посчитать производную сложной функции :)

$$f(g(x))' = f'(g(x)) \cdot g'(x)$$

Считаем производные по каждому параметру

$$\frac{\partial L}{\partial W_2} = \boxed{-\left(y_{true} - f(X \cdot W_1) \cdot W_2\right)} \cdot f(X \cdot W_1)$$

$$\frac{\partial L}{\partial W_1} = \boxed{-\left(y_{true} - f(X \cdot W_1) \cdot W_2\right)} \cdot W_2 \cdot f'(X \cdot W_1) \cdot X$$

Обратное распространение ошибки (back propagation)

Считать на каждом шаге одно и то же **дорого** (особенно для больших архитектур)

$$X \to X \cdot W_1 \to f(X \cdot W_1) \to f(X \cdot W_1) \cdot W_2 \to y$$

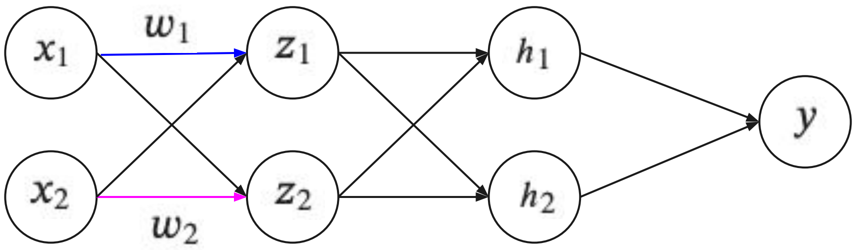
Давайте запоминать то, что уже посчитали!

$$\frac{\partial L}{\partial W_2} = \boxed{-\left(y_{true} - f(X \cdot W_1) \cdot W_2\right)} \cdot f(X \cdot W_1)$$

$$\frac{\partial L}{\partial W_1} = \boxed{-\left(y_{true} - f(X \cdot W_1) \cdot W_2\right)} \cdot W_2 \cdot f'(X \cdot W_1) \cdot X$$

Алгоритм обратного распространения ошибки заключается в том, чтобы значительно сократить вычисления при градиентном спуске за счет **запоминания общих множителей** для нескольких производных

Обратное распространение ошибки (back propagation)



$$\frac{\partial y}{\partial w_1} = \boxed{\frac{\partial y}{\partial h_1}} \frac{\partial h_1}{\partial z_1} \boxed{\frac{\partial z_1}{\partial w_1}} + \boxed{\frac{\partial y}{\partial h_2}} \frac{\partial h_2}{\partial z_1} \boxed{\frac{\partial z_1}{\partial w_1}}$$

$$\frac{\partial y}{\partial w_2} = \boxed{\frac{\partial y}{\partial h_1}} \frac{\partial h_1}{\partial z_2} \boxed{\frac{\partial z_2}{\partial w_2}} + \boxed{\frac{\partial y}{\partial h_2}} \frac{\partial h_2}{\partial z_2} \boxed{\frac{\partial z_2}{\partial w_2}}$$

Не считаем одно и то же несколько раз – **выигрываем в вычислениях**

Спасибо за внимание!