Глубинное обучение

Лекция 3

Обучение нейронных сетей

Михаил Гущин

mhushchyn@hse.ru

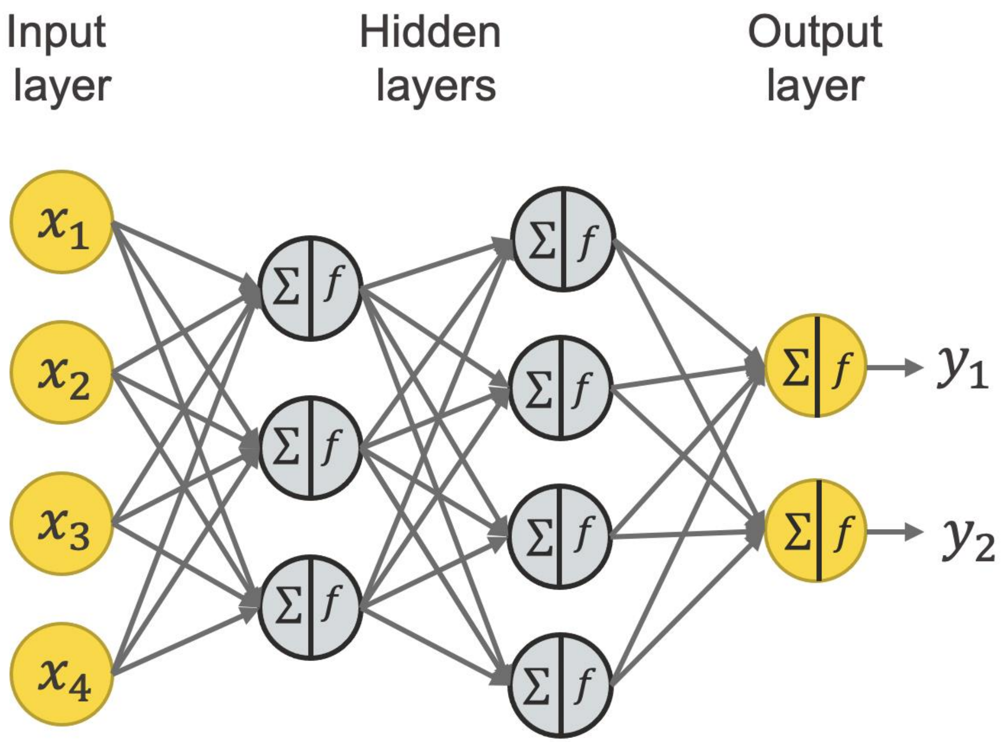
НИУ ВШЭ, 2025

НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ

Полносвязные нейронные сети
(повтор)

Нейронная сеть

- Пусть дан набор наблюдений $\{x_i, y_i\}_{i=1}^{n}$, где $x_i \in R^d$, $y_i \in R^q$
- Построим нейронную сеть



Нейронная сеть в матричной форме

- Матрица наблюдений $X \in R^{n \times d}$ из n объектов, каждый из которых имеет d входных признаков:

$$X = \begin{pmatrix} x_{11} & x_{12} & \cdots & x_{1d} \\ x_{21} & x_{22} & \cdots & x_{2d} \\ \vdots & \vdots & \vdots & \vdots \\ x_{n1} & x_{n2} & \cdots & x_{nd} \end{pmatrix}$$

- Число строк – число наблюдений (объектов)
- Число столбцов – число входных признаков

Нейронная сеть в матричной форме

$$H^{(1)} = f_1\left(XW^{(1)} + b^{(1)}\right)$$
$$H^{(2)} = f_2(H^{(1)}W^{(2)} + b^{(2)})$$
$$O = f_3(H^{(2)}W^{(3)} + b^{(3)})$$

Размеры матриц:

- $X \in R^{n\times d}$, $W^{(1)} \in R^{d\times h}$, $b^{(1)} \in R^{1\times h}$, h - число нейронов в первом слое
- $H^{(1)} \in R^{n\times h}$, $W^{(2)} \in R^{h\times m}$, $b^{(2)} \in R^{1\times m}$, m - число нейронов во втором слое
- $H^{(2)} \in R^{n\times m}$, $W^{(3)} \in R^{m\times q}$, $b^{(3)} \in R^{1\times q}$, q - число выходов сети
- $O \in R^{n\times q}$ - матрица прогнозов сети

Полносвязный слой

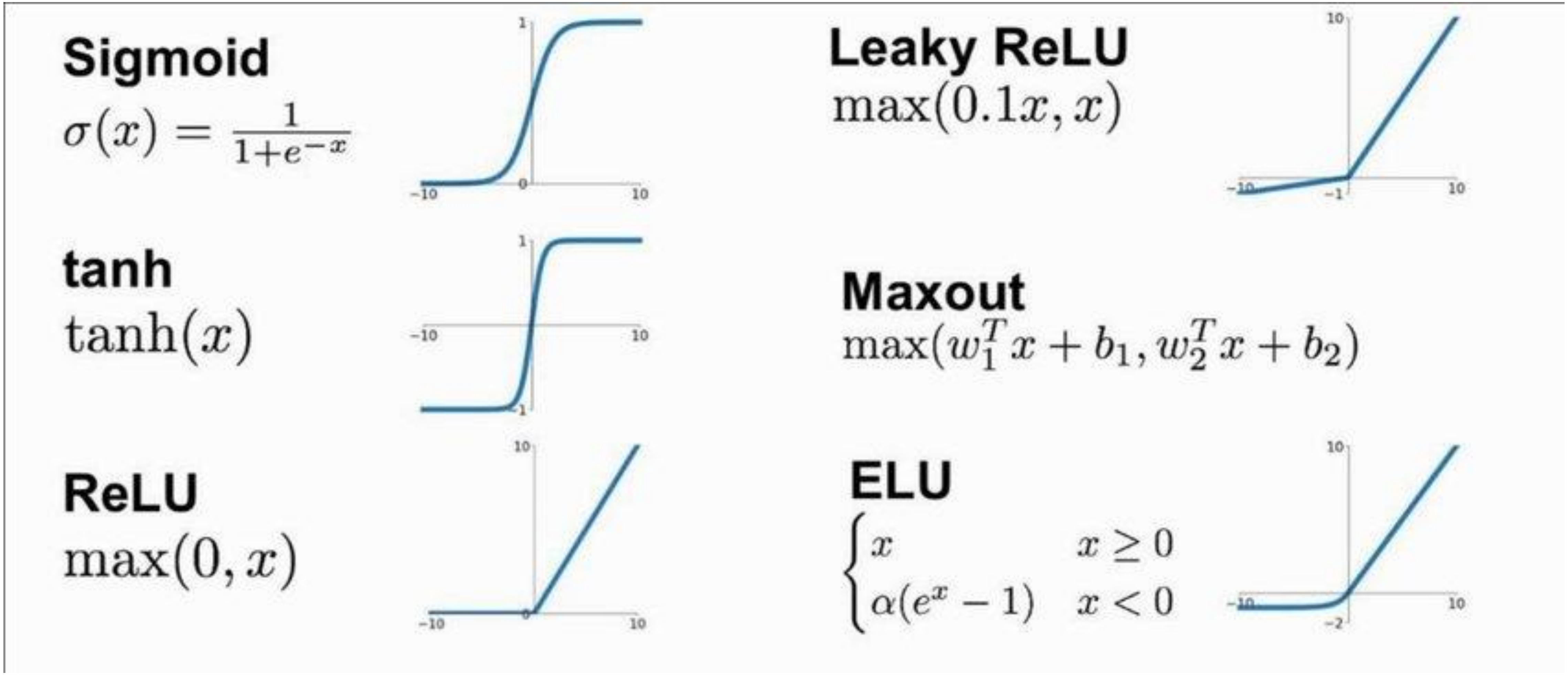
$$H^{(1)} = f_1\left(XW^{(1)} + b^{(1)}\right)$$

$$X = \begin{pmatrix} \boxed{x_{11} & x_{12} & \cdots & x_{1d}} \\ x_{21} & x_{22} & \cdots & x_{2d} \\ \vdots & \vdots & \vdots & \vdots \\ x_{n1} & x_{n2} & \cdots & x_{nd} \end{pmatrix}, \quad W^{(1)} = \begin{pmatrix} w_{11} & w_{12} & \cdots & w_{1h} \\ w_{21} & w_{22} & \cdots & w_{2h} \\ \vdots & \vdots & \vdots & \vdots \\ w_{d1} & w_{d2} & \cdots & w_{dh} \end{pmatrix}, \quad \mathrm{H}^{(1)} = \begin{pmatrix} h_{11} & \boxed{h_{12}} & \cdots & h_{1h} \\ h_{21} & h_{22} & \cdots & h_{2h} \\ \vdots & \vdots & \vdots & \vdots \\ h_{n1} & h_{n2} & \cdots & h_{nh} \end{pmatrix}$$

Размеры матриц:

- $X \in R^{n \times d}$, $W^{(1)} \in R^{d \times h}$, $b^{(1)} \in R^{1 \times h}$, h - число нейронов в первом слое
- $H^{(1)} \in R^{n \times h}$ - выход слоя
- $f_1(z)$ - **функция активации**

Функции активации f



Градиентный спуск

- Нейронные сети обучаем с помощью градиентного спуска

$$w = w - \alpha \frac{\partial L}{\partial w}$$

- Как посчитать градиент функции потерь по нужному весу сети?
- Про это поговорим на следующей лекции

Градиентный спуск

- Как посчитать градиент для нейронной сети?

Прямое распространение (forward propagation)

Прогноз нейронной сети

Рассмотрим нейронную сеть с одним скрытым слоем (b опустим для простоты):

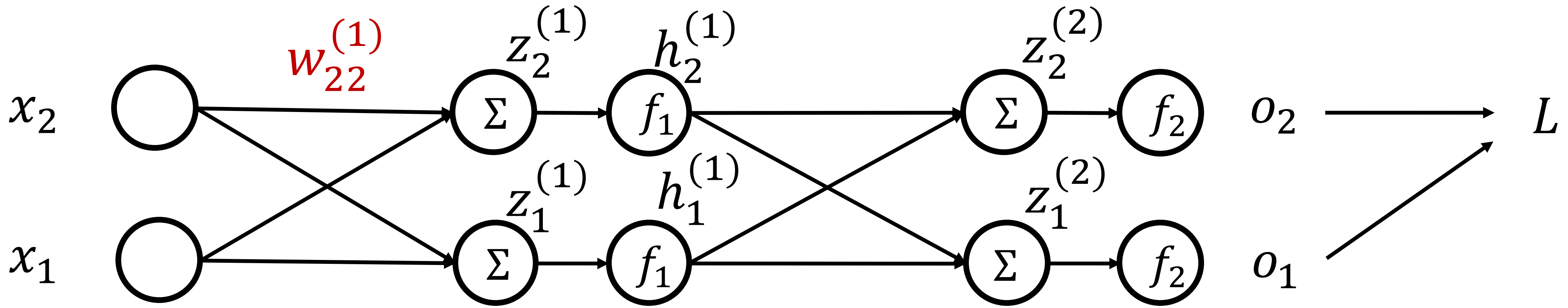
$$z^{(1)} = W^{(1)}x$$
$$h^{(1)} = f_1\left(z^{(1)}\right)$$
$$z^{(2)} = W^{(2)}h^{(1)}$$
$$o = f_2(z^{(2)})$$

Размеры матриц:

- $x \in R^{d\times1}$, $W^{(1)} \in R^{k\times d}$, k - число нейронов в первом (скрытом) слое
- $W^{(2)} \in R^{m\times k}$, m - число нейронов во втором (выходном) слое
- $\mathrm{o} \in R^{m\times1}$ - вектор прогнозов сети

Прямое распространение

$$o = f_2\left(W^{(2)} f_1\left(W^{(1)} x\right)\right)$$



Обратное распространение (backward propagation)

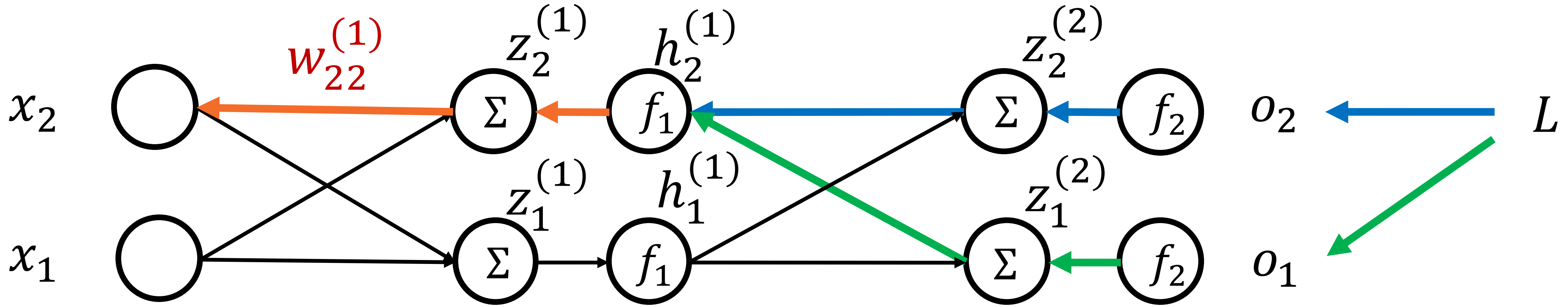
Производная сложной функции

- Вспомним формулу производной сложной функции
- Пусть даны функции $y = f(x)$, $z = g(y)$
- Тогда

$$\frac{\partial z}{\partial x} = \frac{\partial z}{\partial y}\frac{\partial y}{\partial x}$$

Обратное распространение (ошибки)

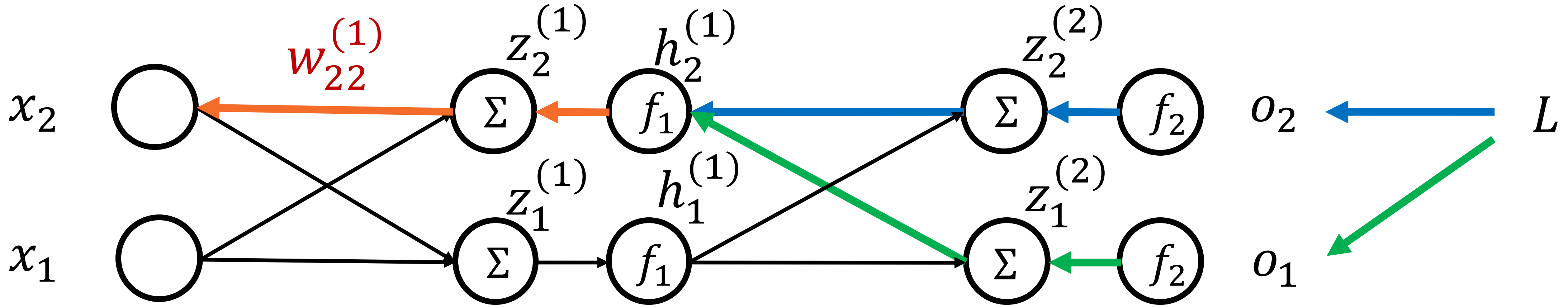
$$o = f_2\left(W^{(2)} f_1\left(W^{(1)} x\right)\right)$$



$$\frac{\partial L}{\partial w_{22}^{(1)}} = ?$$

Обратное распространение (ошибки)

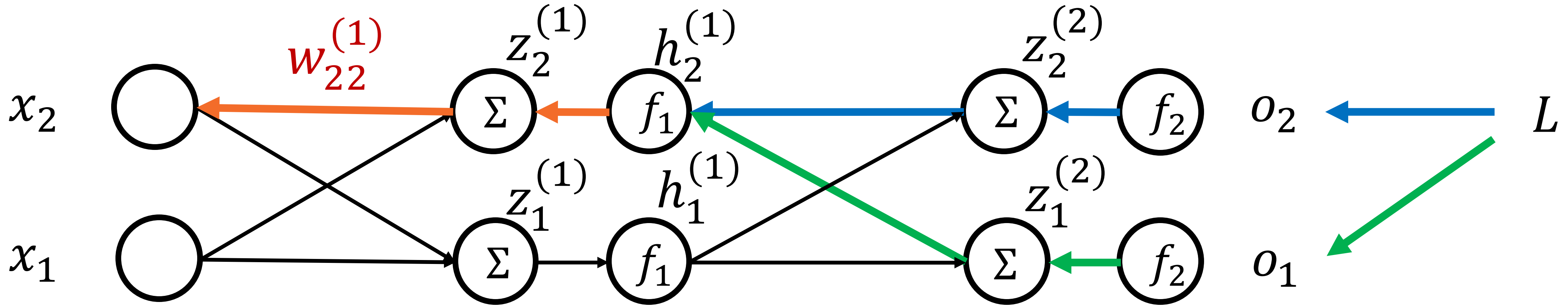
$$o = f_2\left(W^{(2)} f_1\left(W^{(1)} x\right)\right)$$



$$\frac{\partial L}{\partial w_{22}^{(1)}} = \frac{\partial L}{\partial o_1}\frac{\partial o_1}{\partial z_1^{(2)}}\frac{\partial z_1^{(2)}}{\partial h_2^{(1)}}\frac{\partial h_2^{(1)}}{\partial z_2^{(1)}}\frac{z_2^{(1)}}{\partial w_{22}^{(1)}} + \frac{\partial L}{\partial o_2}\frac{\partial o_2}{\partial z_2^{(2)}}\frac{\partial z_2^{(2)}}{\partial h_2^{(1)}}\frac{\partial h_2^{(1)}}{\partial z_2^{(1)}}\frac{z_2^{(1)}}{\partial w_{22}^{(1)}}$$

Обратное распространение (ошибки)

$$o = f_2\left(W^{(2)} f_1\left(W^{(1)} x\right)\right)$$



$$\frac{\partial L}{\partial w_{22}^{(1)}} = \frac{\partial L}{\partial o_1} \frac{\partial o_1}{\partial z_1^{(2)}} \frac{\partial z_1^{(2)}}{\partial h_2^{(1)}} \frac{\partial h_2^{(1)}}{\partial z_2^{(1)}} \frac{z_2^{(1)}}{\partial w_{22}^{(1)}} + \frac{\partial L}{\partial o_2} \frac{\partial o_2}{\partial z_2^{(2)}} \frac{\partial z_2^{(2)}}{\partial h_2^{(1)}} \frac{\partial h_2^{(1)}}{\partial z_2^{(1)}} \frac{z_2^{(1)}}{\partial w_{22}^{(1)}}$$

$$w_{22}^{(1)} = w_{22}^{(1)} - \alpha \frac{\partial L_i}{\partial w_{22}^{(1)}}$$

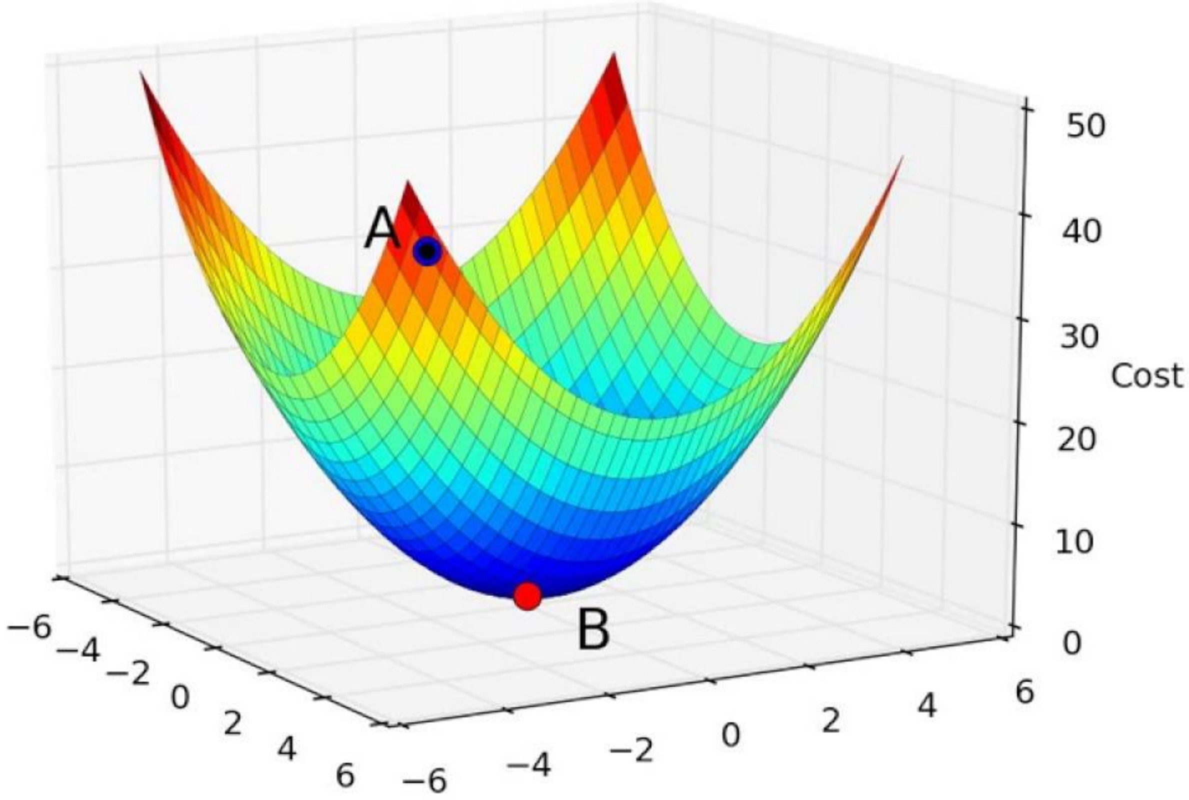
Градиентный спуск

Задача

- Есть функция $L(w)$
- Хотим найти ее минимум:

$$L \to \min_w$$

- Мы умеем считать ее производную $\frac{\partial L}{\partial w}$
- Но не умеем (или не хотим) решать уравнение $\frac{\partial L}{\partial w} = 0$



Градиент функции

- Градиент функции (∇L) – вектор первых частный производных функции:

$$\nabla L(w_0, w_1, \ldots, w_d) = \left(\frac{\partial L}{\partial w_0}, \frac{\partial L}{\partial w_1}, \ldots, \frac{\partial L}{\partial w_d}\right)$$

- В векторной форме мы будем писать:

$$\nabla L(w) = \frac{\partial L(w)}{\partial w}$$

Градиентный спуск

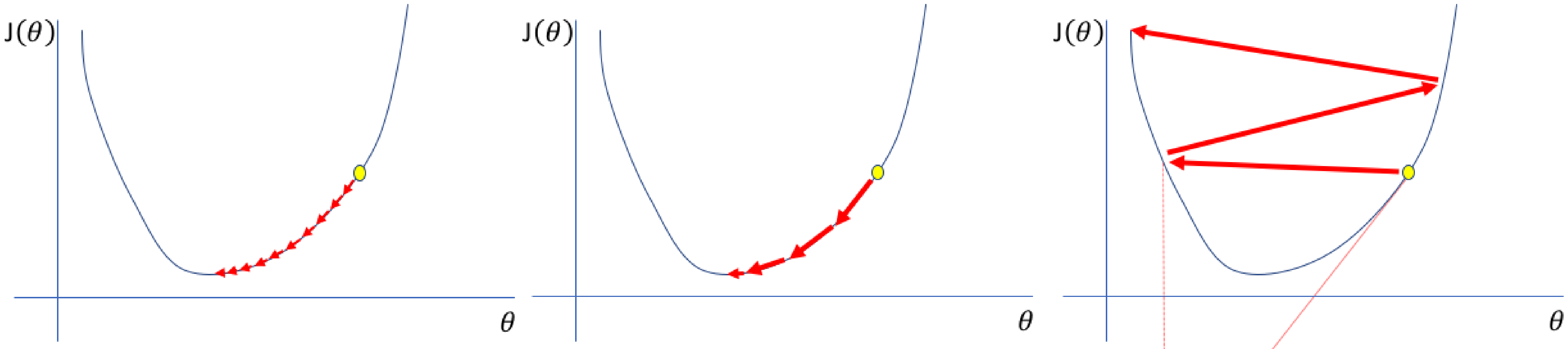
- Есть функция $L(w)$, минимум которой хотим найти
- Пусть $w^{(0)}$ - начальный вектор параметров. Например, $w^{(0)} = 0$
- Тогда **градиентный спуск** состоит в повторении:

$$w^{(k+1)} = w^{(k)} - \boldsymbol{\eta} \nabla L(w^{(k)})$$

  – $\boldsymbol{\eta}$ – длина шага градиентного спуска (**learning rate**) (мы сами его задаем)
  – k – номер итерации
  – $\nabla L(w^{(k)})$ – градиент функции потерь на итерации k

Выбор шага

Too low | **Just right** | **Too high**

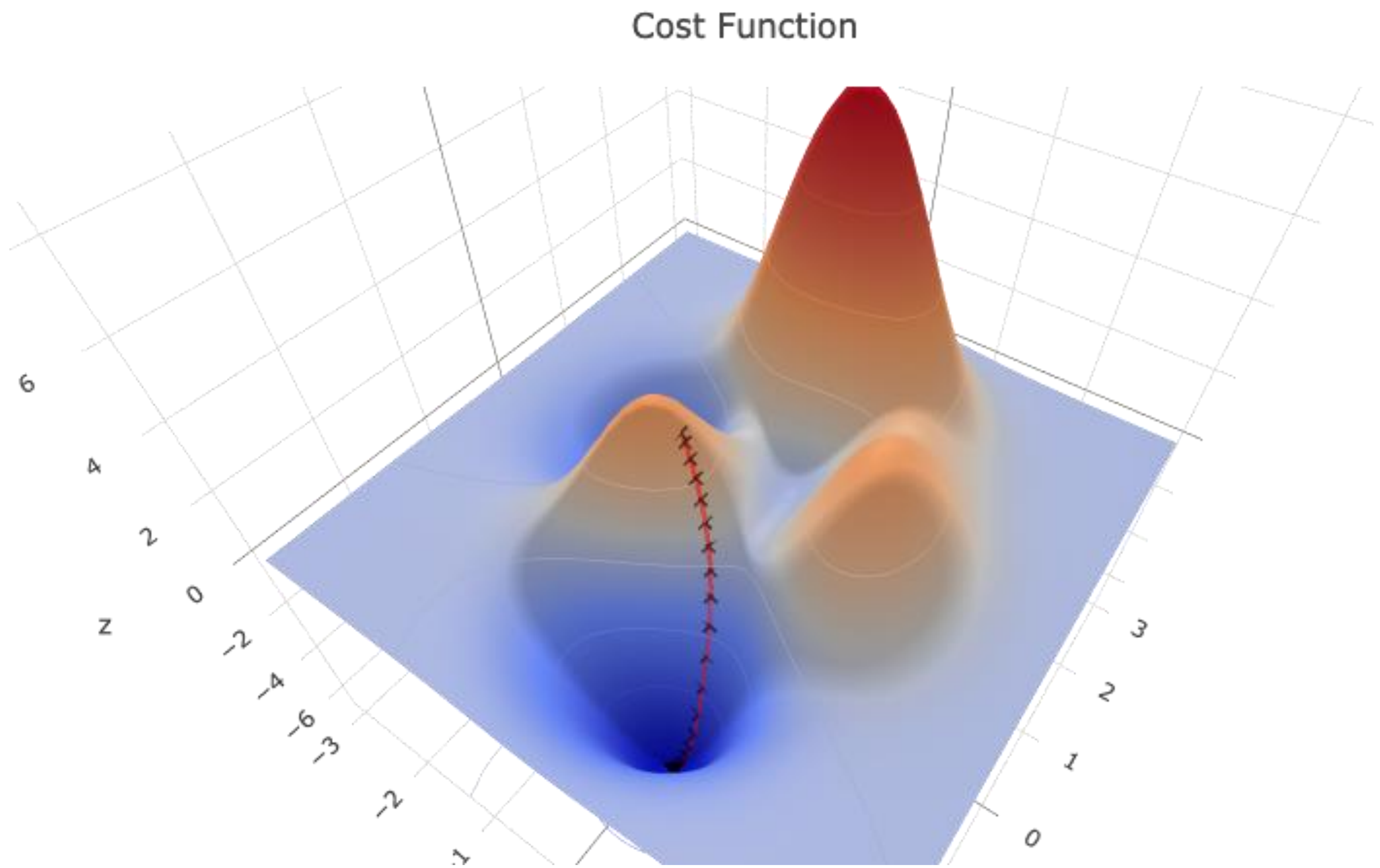


A small learning rate requires many updates before reaching the minimum point

The optimal learning rate swiftly reaches the minimum point

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

Демо

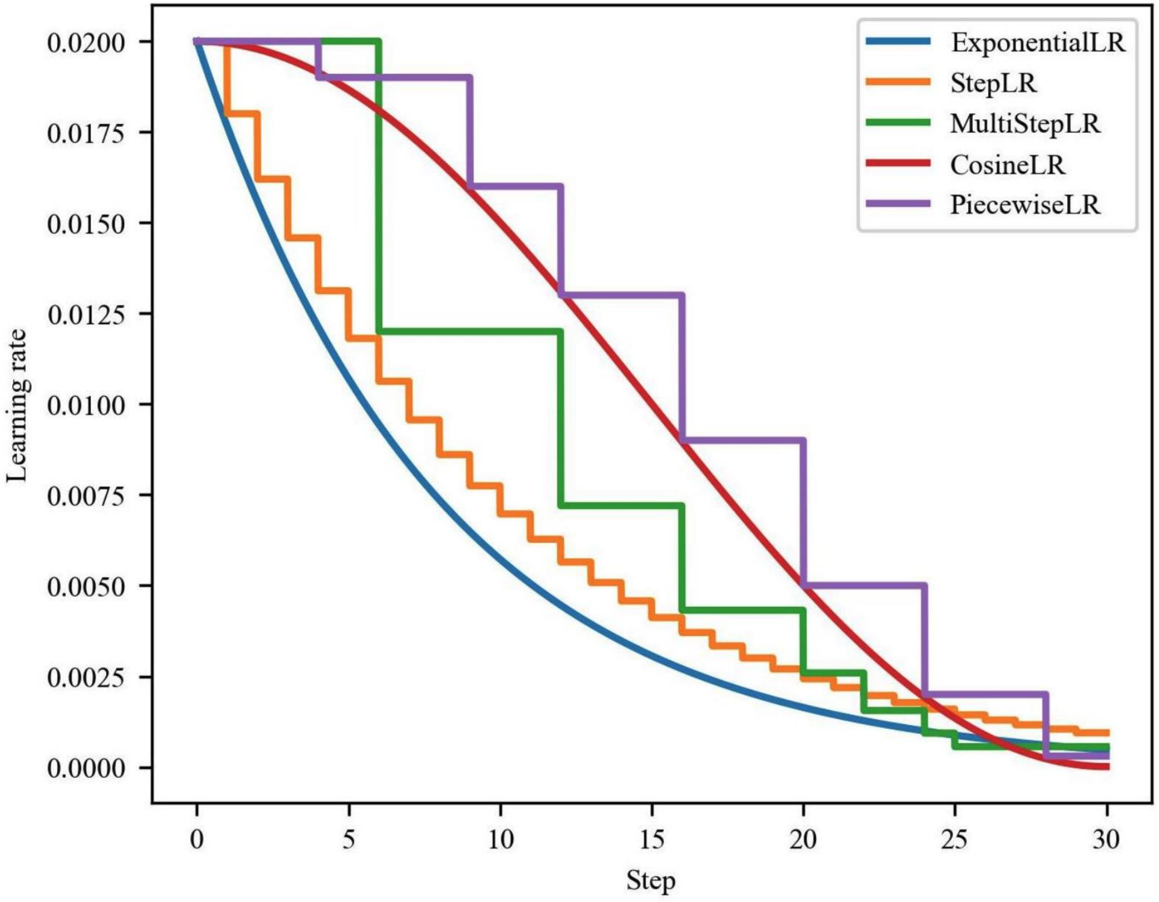


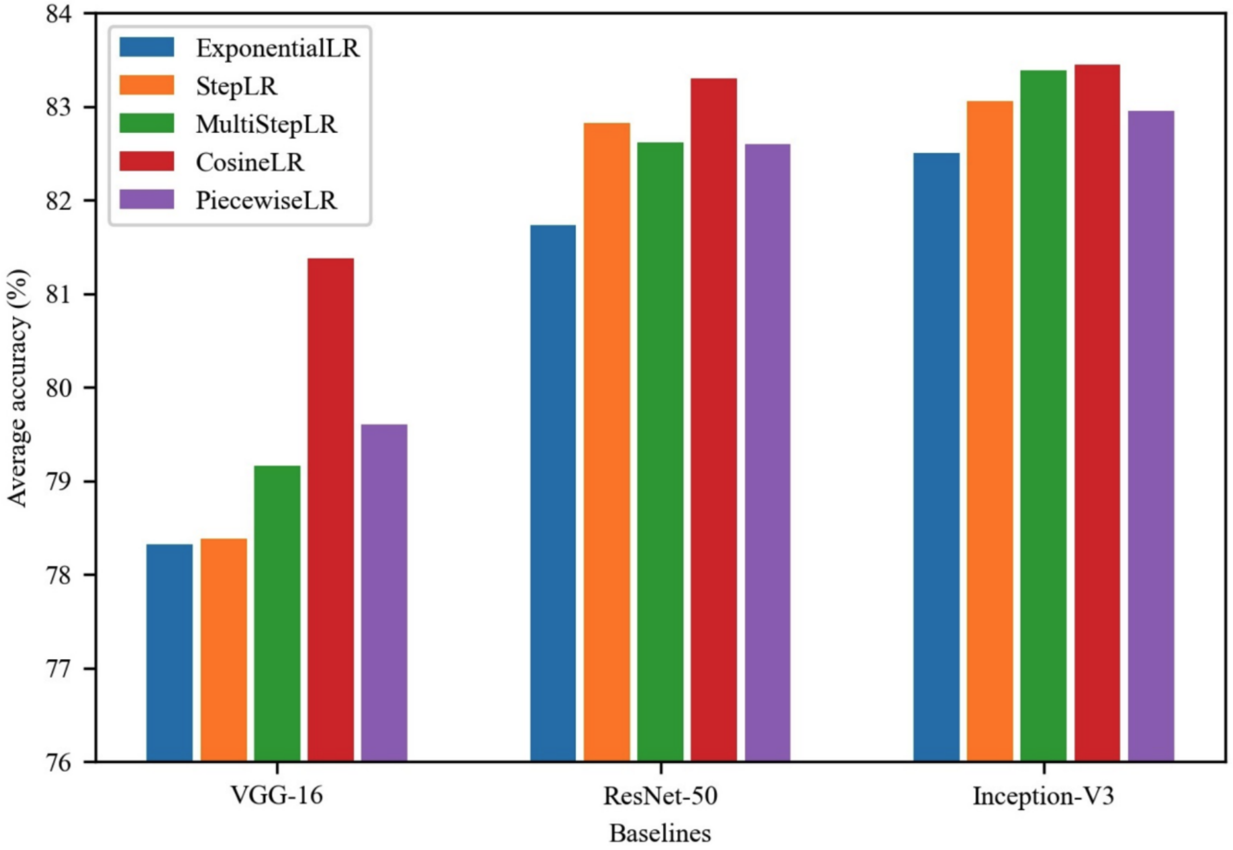
https://blog.skz.dev/gradient-descent

Выбор шага (Learning Rate Scheduler)

- Константа: $\eta = const$
- Factor Scheduler: $\eta_{t+1} = \eta_t * \alpha,\ \alpha \in (0, 1)$
- Exponential Scheduler: $\eta_{t+1} = \eta_0 * \alpha^t,\ \alpha \in (0, 1)$
- Polynomial Scheduler: $\eta_t = \frac{\eta_0}{(1+\alpha t)^\beta},\ \beta \in (0, 1), \alpha > 0$
- Cosine Scheduler: $\eta_t = \eta_T + \frac{\eta_0 - \eta_T}{2}(1 + \cos(\pi t / T)),\ \mathrm{t} \in (0, T)$
- Можно задавать свои

Пример:





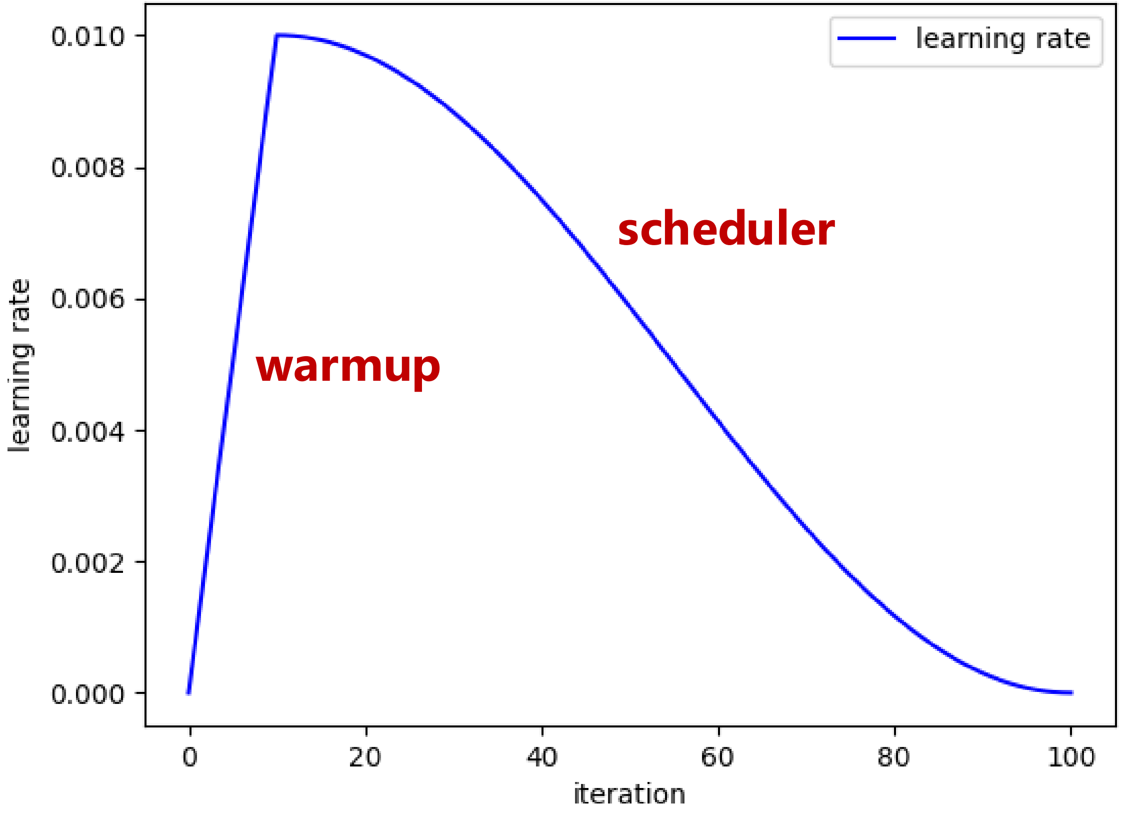
Источник: https://www.mdpi.com/1424-8220/22/21/8089

Прогрев обучения (warmup)

- Начальные значения параметров (весов сети) инициализируются случайно, что может приводить к нестабильным решениям

- Можно выбрать малый шаг обучения, но спуск будет медленным

- **Warmup** – это когда мы увеличиваем шаг обучения с 0 до базового значения в течение первых эпох обучения



Критерии остановки

- Близость градиента к нулю: $\nabla L \approx 0$

- Малое изменение вектора весов: $|\mathrm{w}^{(\mathrm{k}+1)} - \mathrm{w}^{(\mathrm{k})}| \approx 0$

- Когда значение функции потерь L перестало уменьшаться

Стохастический градиентный спуск

Нейронная сеть в матричной форме

- Матрица наблюдений $X \in R^{n \times d}$ из n объектов, каждый из которых имеет d входных признаков:

$$X = \begin{pmatrix} x_{11} & x_{12} & \cdots & x_{1d} \\ x_{21} & x_{22} & \cdots & x_{2d} \\ \vdots & \vdots & \vdots & \vdots \\ x_{n1} & x_{n2} & \cdots & x_{nd} \end{pmatrix}$$

- Число строк – число наблюдений (объектов)
- Число столбцов – число входных признаков

Стохастический градиентный спуск

- Полный градиентный спуск (gradient descent):

$$L(w) = \frac{1}{n}\sum_{i=1}^{n} L(o_i, y_i)$$

- **Пакетный** градиентный спуск (batch gradient descent) ($1 < b < n$):

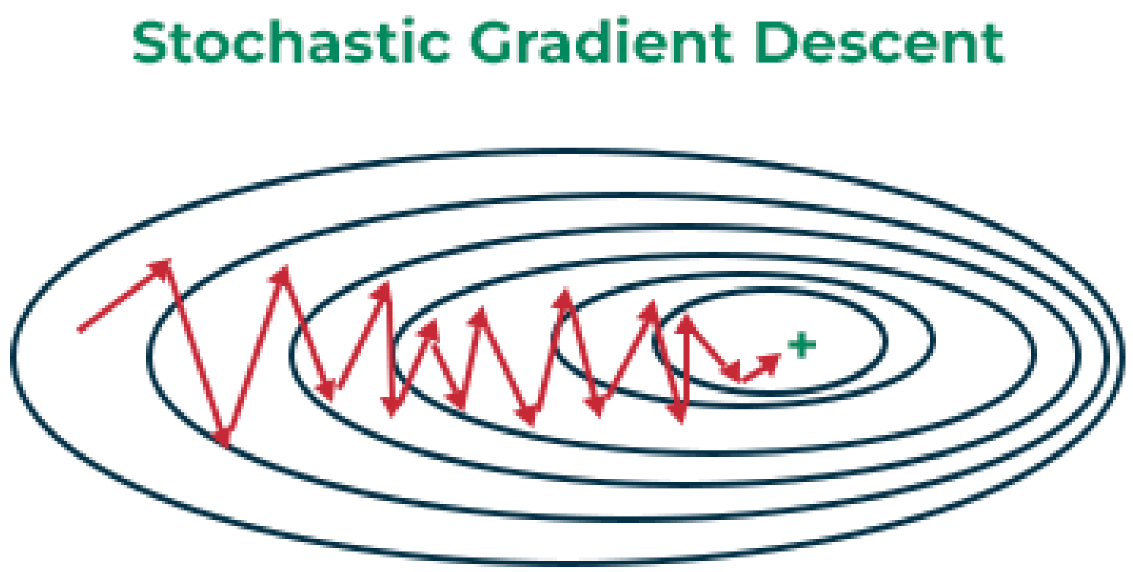
$$L(w) = \frac{1}{b}\sum_{i=1}^{b} L(o_i, y_i)$$

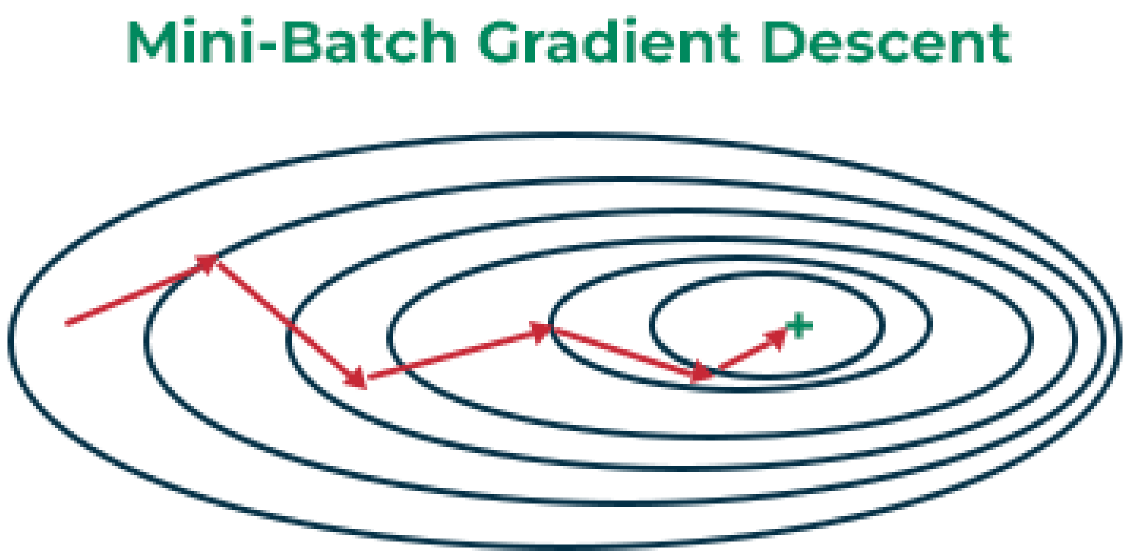
- **Стохастический** градиентный спуск (stochastic gradient descent):

$$L(w) = L(o_i, y_i),$$

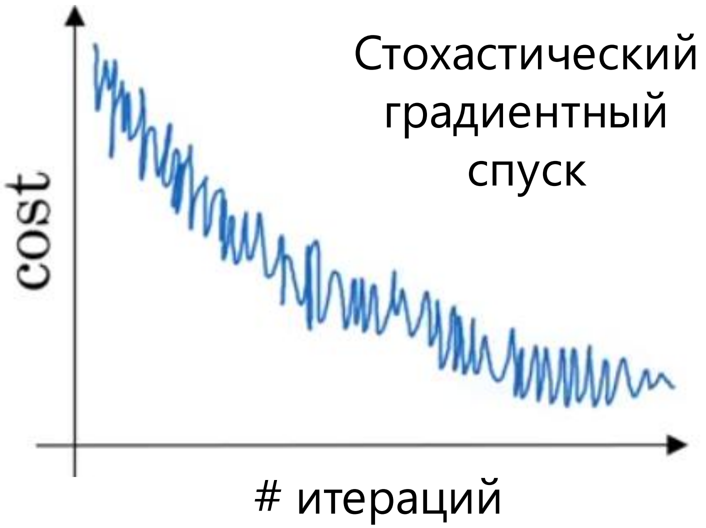
$$w^{(k+1)} = w^{(k)} - \eta \nabla L(w^{(k)})$$

Стохастический градиентный спуск





Стохастический градиентный спуск



- Стохастический ГС требует меньше вычислительный операций
- В полном ГС обучение стабильнее
- Полный ГС требует меньше итераций, но больше вычислительных операций

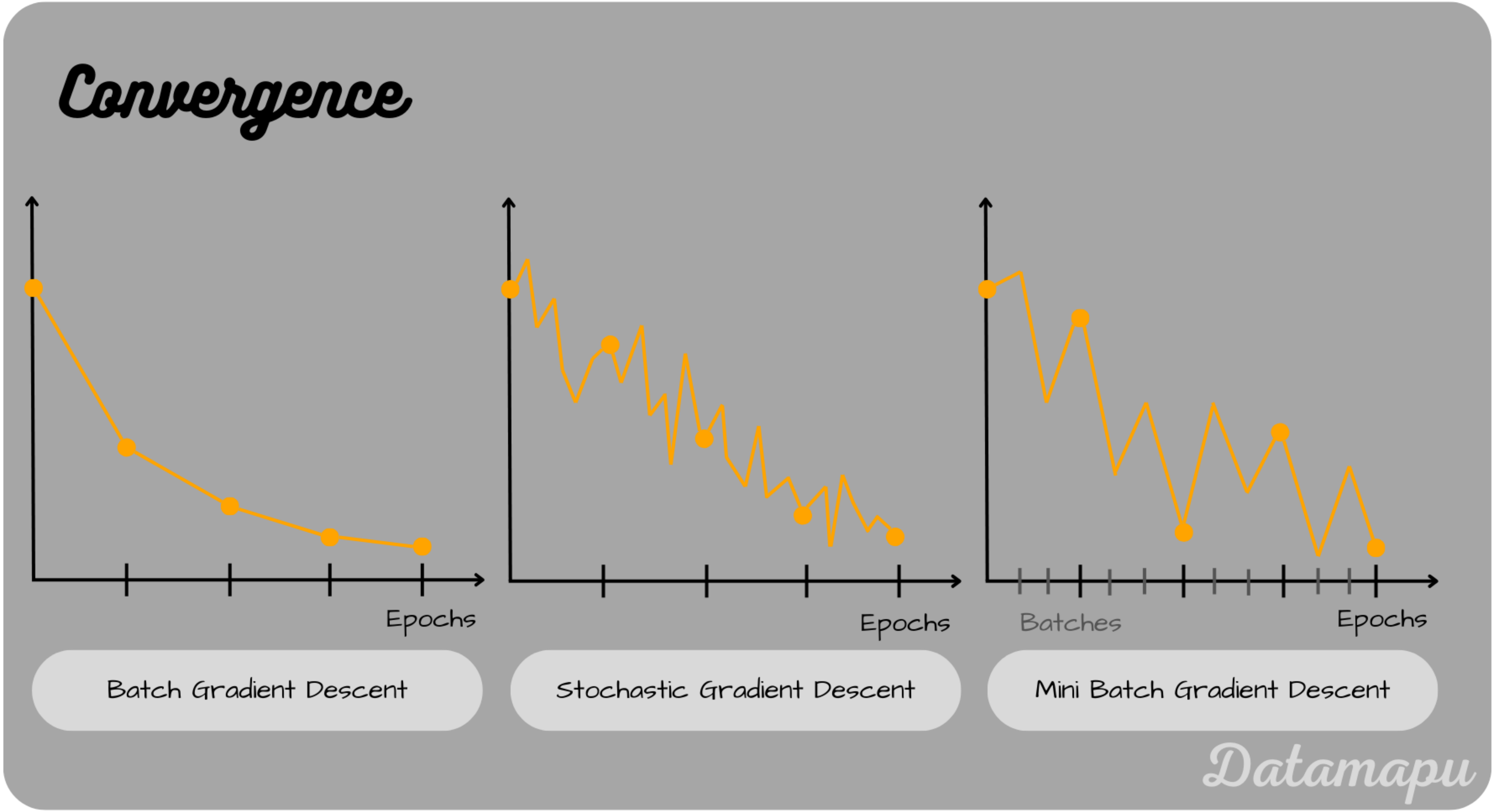
Эпохи и итерации

- **Итерация** – один шаг градиентного спуска (любого)
- **Эпоха** – это когда все объекты выборки прошли обучение

Пример: пусть обучающая выборка содержит 100 объектов. Тогда:

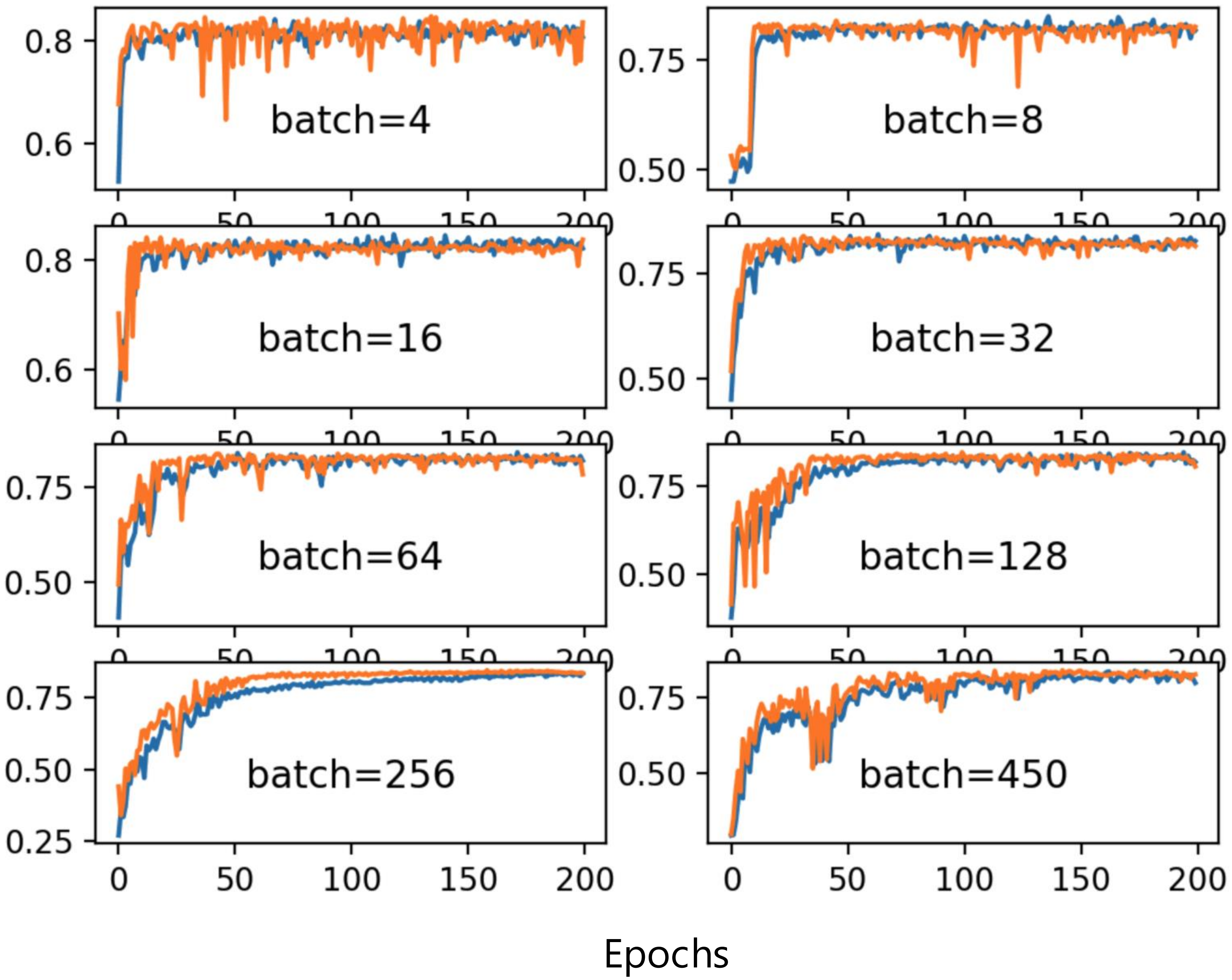
- Градиентный спуск (ГС): 1 эпоха = 1 итерация
- Пакетный ГС с batch_size=10: 1 эпоха = 10 итераций
- Стохастический ГС: 1 эпоха = 100 итераций

Стохастический градиентный спуск



Пример:

Источник: https://machinelearningmastery.com/how-to-control-the-speed-and-stability-of-training-neural-networks-with-gradient-descent-batch-size/



Нормализация данных

- Модель линейной регрессии:

$$\hat{y}_i = w_0 + w_1 x_{i1} + w_2 x_{i2}$$

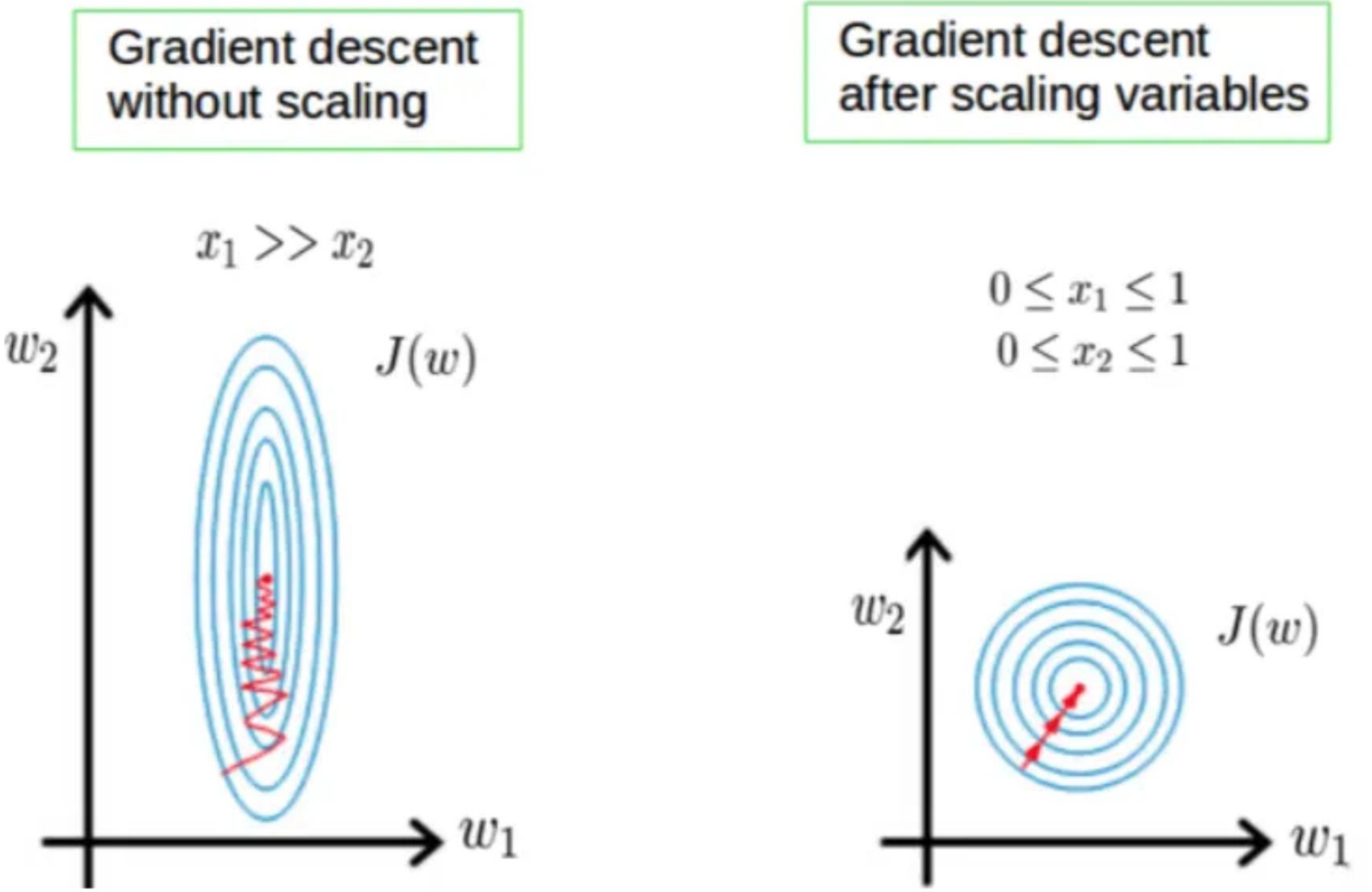
- Функция потерь MSE:

$$L(w) = \frac{1}{n}\sum_{i=1}^{n}(\hat{y}_i - y_i)^2 \to \min_w$$

- Пусть признаки разные по масштабу:
  - например $|x_1| \approx 1000$, $|x_2| \approx 1$

- Тогда малые изменения dw_2 приводят к малым изменениям $L(w)$
- Малые изменения dw_1 приводят к большим изменениям $L(w)$

Нормализация данных



- Нормализация данных стабилизирует градиентный спуск
- Обучение происходит быстрее

Модификации градиентного спуска

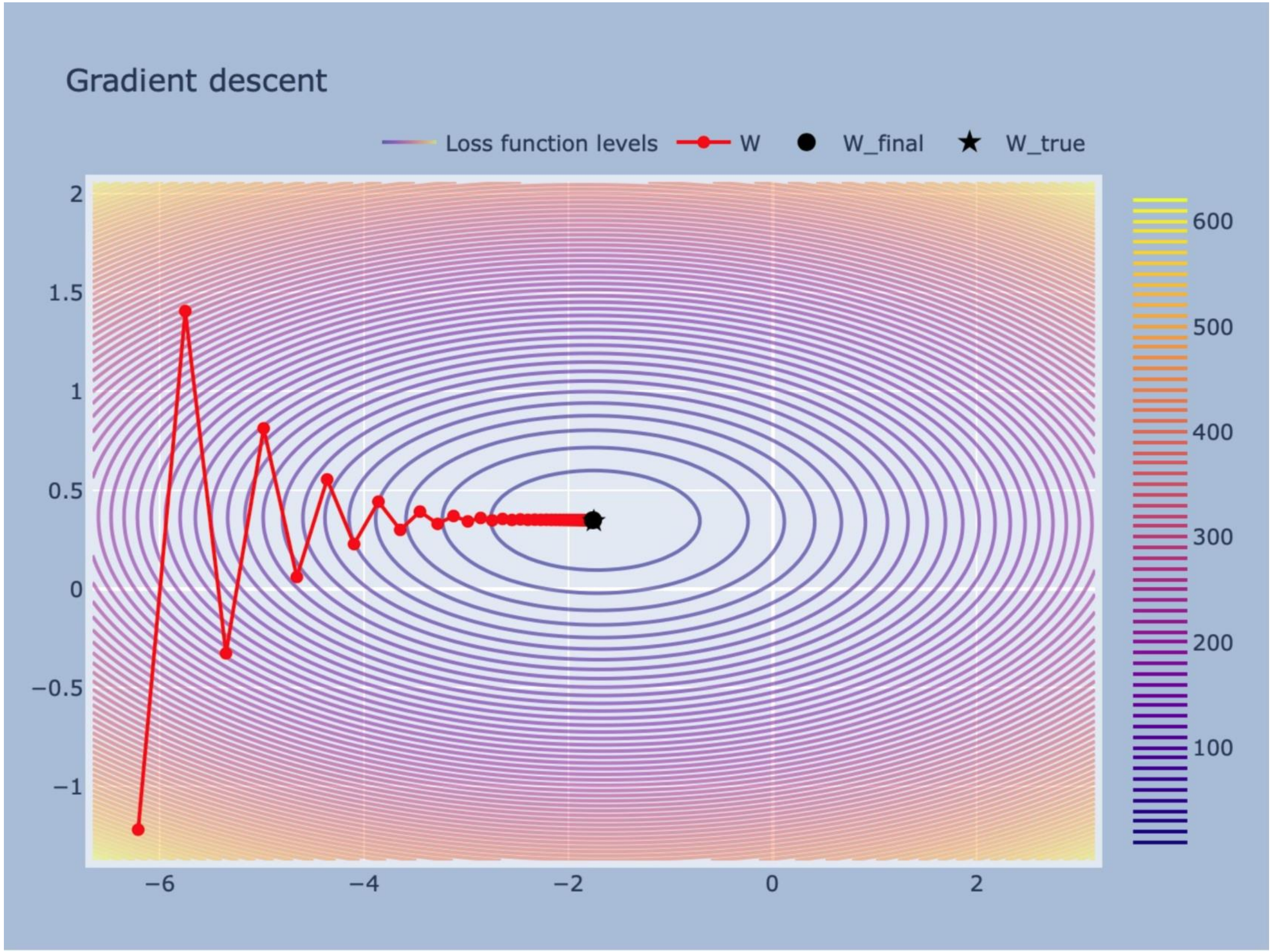
Проблемы градиентного спуска

- Рассмотрим функцию для оптимизации:

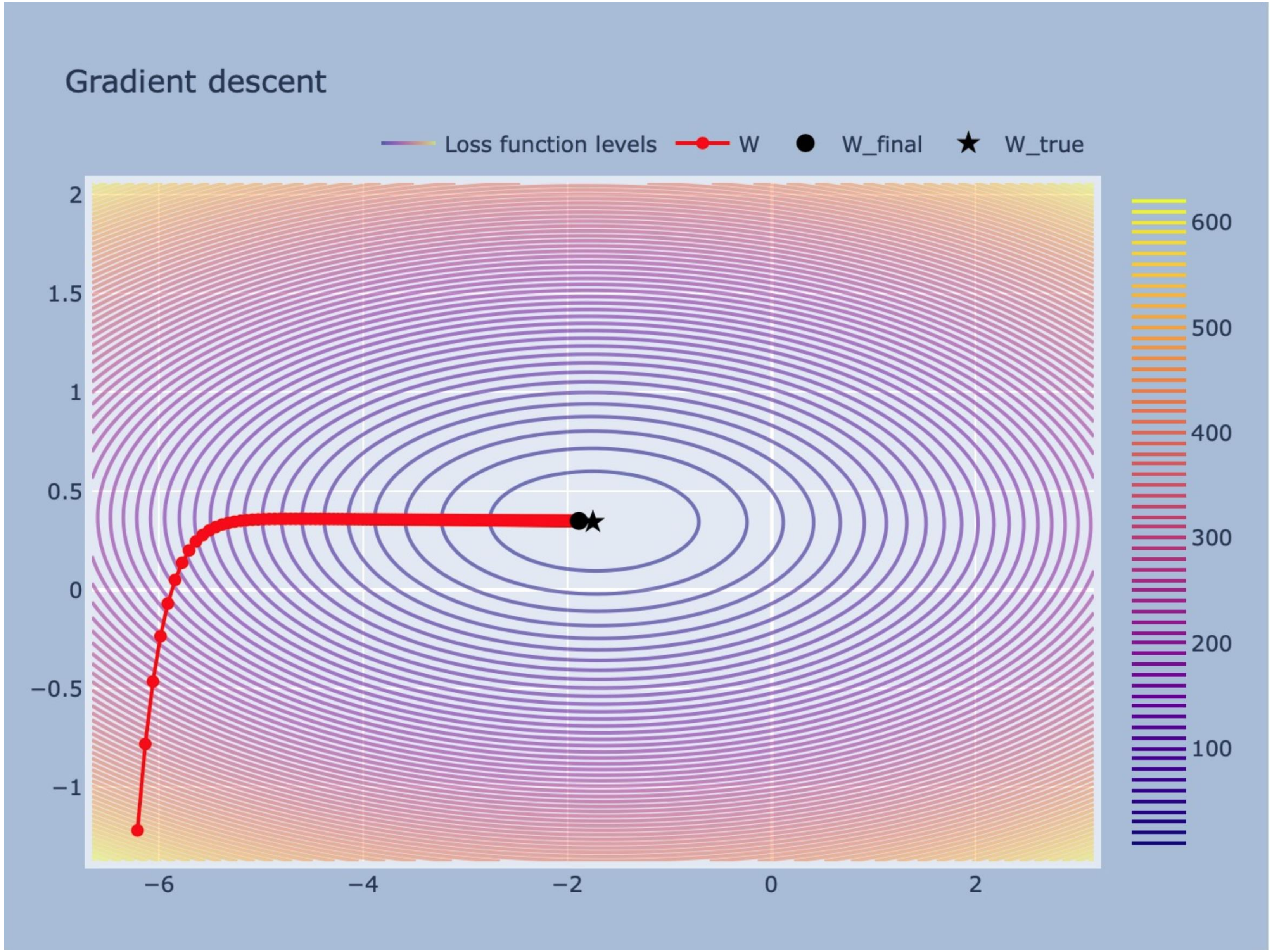
$$L(w) = 0.1w_1^2 + 2w_2^2$$

- Видим, что вклад w_2 сильно больше, чем для w_1
- Применим градиентный спуск

Проблемы градиентного спуска



Проблемы градиентного спуска



Проблемы градиентного спуска

- Если линии уровня функции вытянуты, то градиентный спуск требует аккуратного выбора длины шага
- Если шаг слишком большой, то градиентный спуск не сойдется
- Если шаг слишком маленький, то градиентный спуск будет сходиться слишком долго

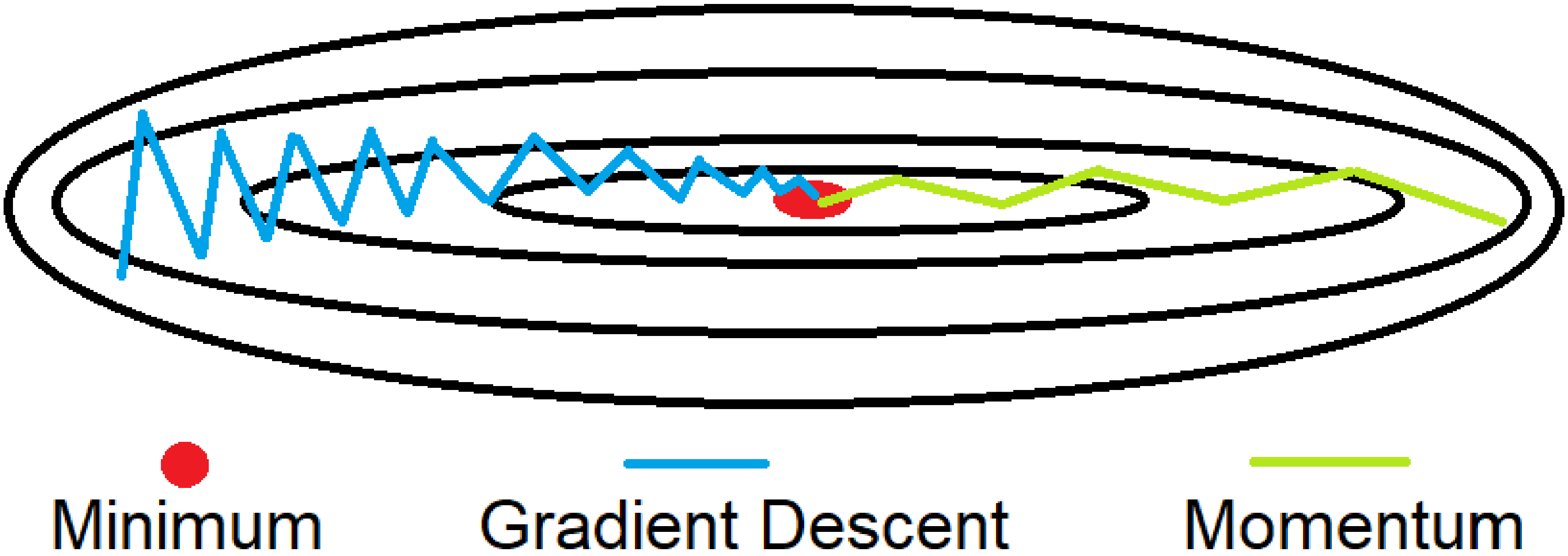
Momentum

$$h_t = \beta h_{t-1} + \nabla L(w^{(t-1)})$$

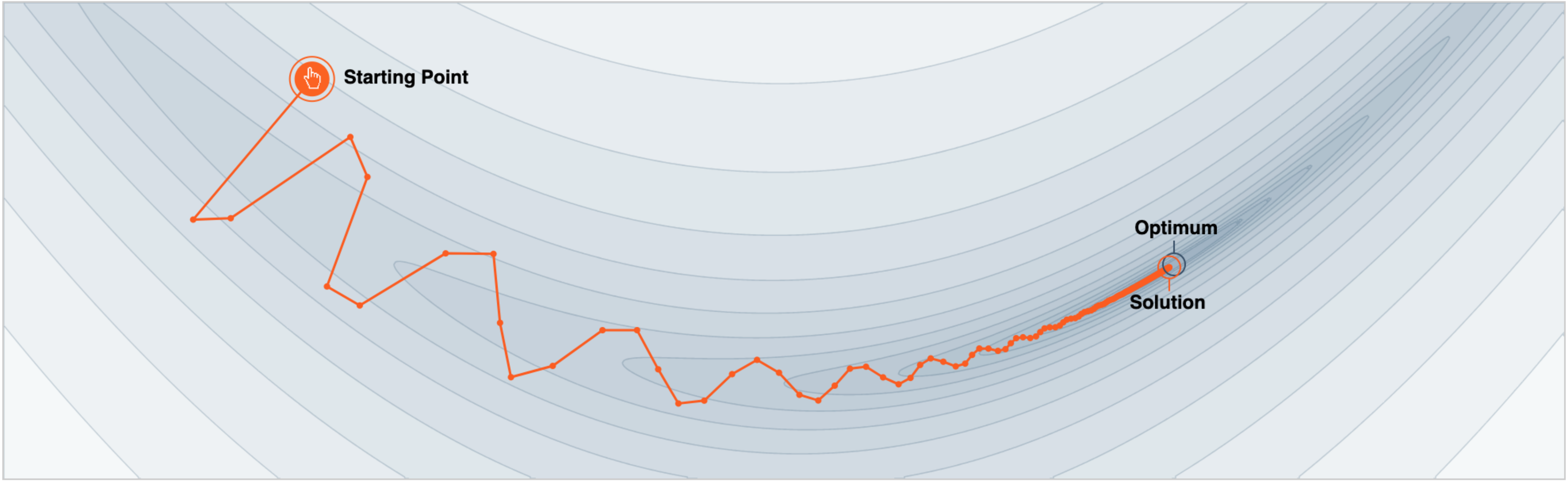
$$w^{(t)} = w^{(t-1)} - \eta h_t$$

- h_t – инерция, усредненное направление движения
- β – гиперпараметр затухания

Пример momentum



Пример momentum



Step-size α = 0.02

0 0.003 0.006

Momentum β = 0.83

0.00 0.500 0.990

We often think of Momentum as a means of dampening oscillations and speeding up the iterations, leading to faster convergence. But it has other interesting behavior. It allows a larger range of step-sizes to be used, and creates its own oscillations. What is going on?

https://distill.pub/2017/momentum

AdaGrad

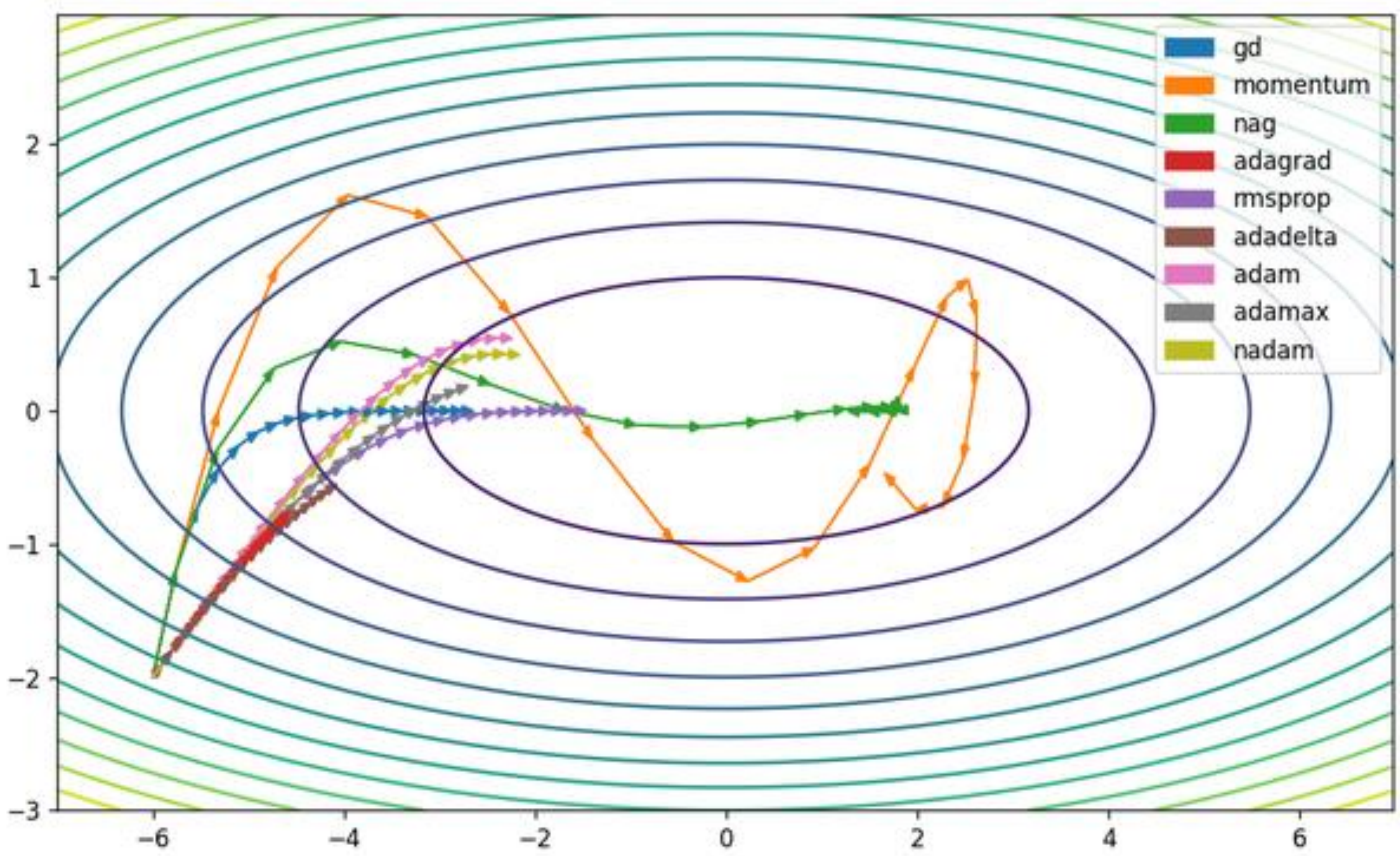
$$g_{t-1} = \nabla L\left(w^{(t-1)}\right)$$

$$s_t = s_{t-1} + (g_{t-1})^2$$

$$w^{(t)} = w^{(t-1)} - \frac{\eta}{\sqrt{s_t + \epsilon}} \odot g_{t-1}$$

- $\odot$ - поэлементное умножение
- По каждому параметру (весу сети) своя скорость спуска. Это полезно, когда у признаков разный масштаб (1, 100, 1000)

Пример



gd
momentum
nag
adagrad
rmsprop
adadelta
adam
adamax
nadam
2
1
0
−1
−2
−3
−6
−4
−2
0
2
4
6

RMSprop

$$g_{t-1} = \nabla L\left(w^{(t-1)}\right)$$

$$s_t = \gamma s_{t-1} + (1-\gamma)(g_{t-1})^2$$

$$w^{(t)} = w^{(t-1)} - \frac{\eta}{\sqrt{s_t + \epsilon}} \odot g_{t-1}$$

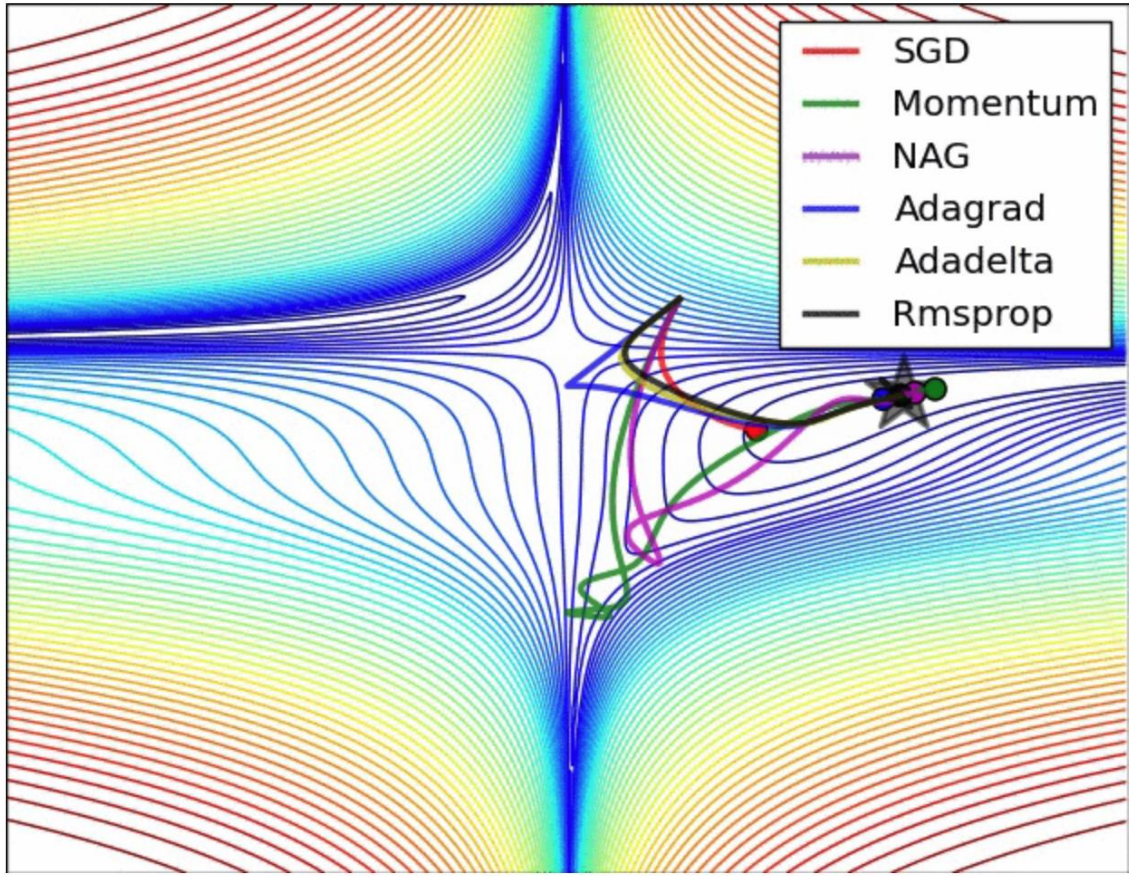
- γ=0.9
- Скорость спуска сильнее зависит от недавних шагов

Adam

$$g_{t-1} = \nabla L\left(w^{(t-1)}\right)$$
$$h_t = \beta_1 h_{t-1} + (1 - \beta_1) g_{t-1}$$
$$\hat{h}_t = \frac{h_t}{1 - \beta_1^t}$$
$$s_t = \beta_2 s_{t-1} + (1 - \beta_2)(g_{t-1})^2$$
$$\hat{s}_t = \frac{s_t}{1 - \beta_2^t}$$
$$w^{(t)} = w^{(t-1)} - \frac{\eta}{\sqrt{\hat{s}_t + \epsilon}} \odot \hat{h}_t$$

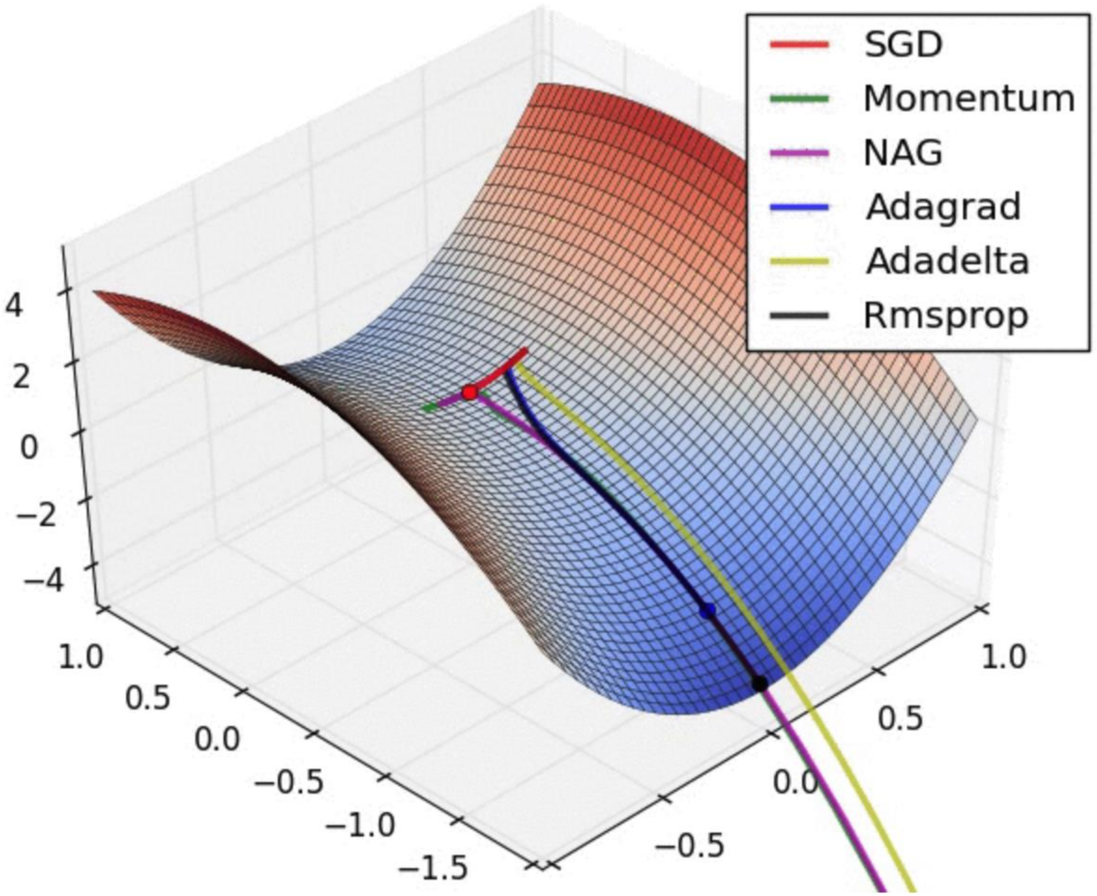
- $\beta_1 = 0.9$
- $\beta_2 = 0.999$

Пример



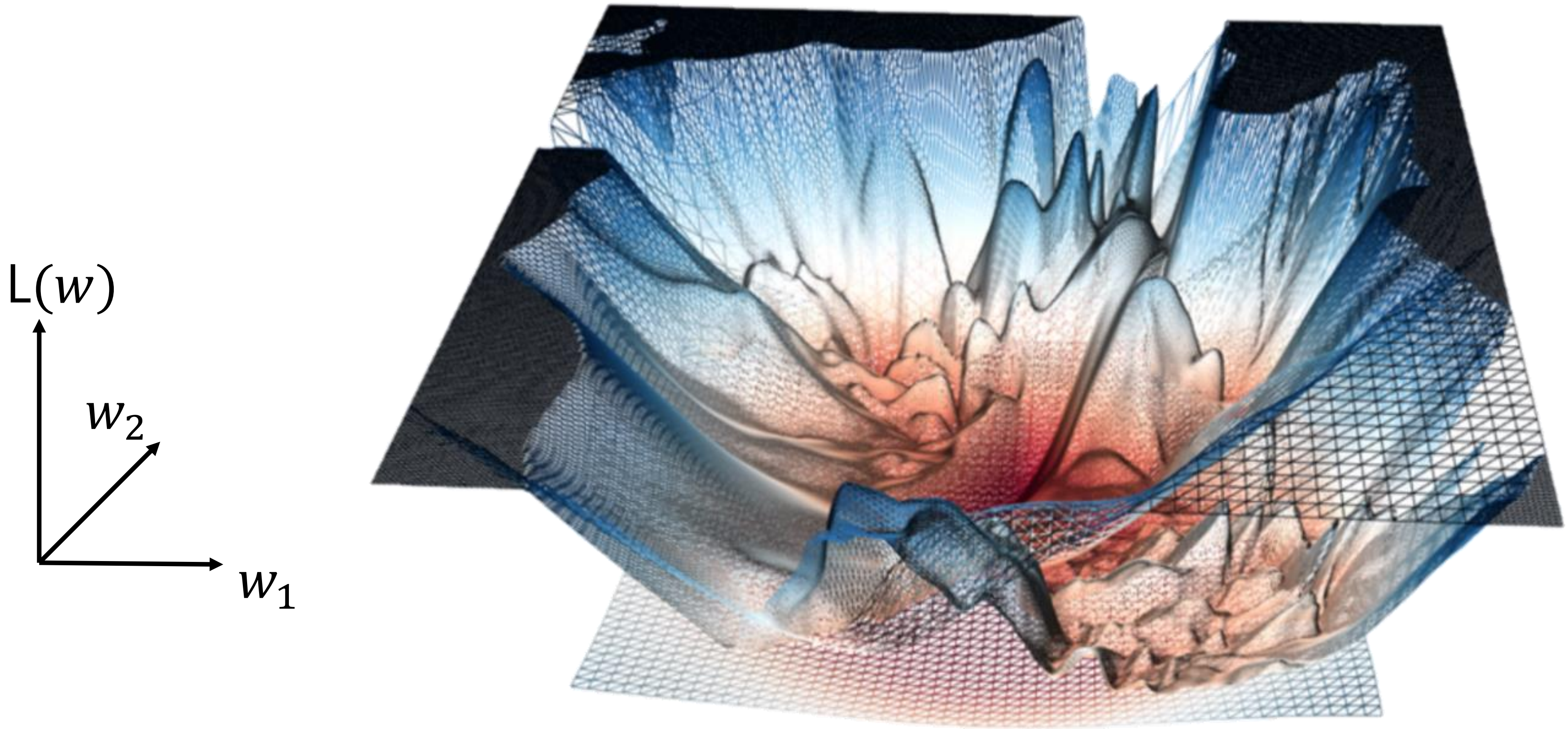
https://api.wandb.ai/files/lavanyashukla/images/projects/36993/d07b8a79.gif

Пример



https://api.wandb.ai/files/lavanyashukla/images/projects/36993/2a6f771c.gif

Примеры функций потерь нейронных сетей



- Demo: https://www.telesens.co/loss-landscape-viz/viewer.html
- Функции потерь как искусство: https://losslandscape.com/

Инициализация весов сети

Инициализация весов сети

- Как инициализировать значения весов сети?
  - Константа
  - Нормальное распределение
  - Равномерное распределение
  - Как-то еще
- Какой из способов лучше?
- На что влияет это выбор?

Произвольный нейрон

$$z = \sum_{i=1}^{n} w_i x_i$$

$$\hat{y} = ReLU(z)$$

где:

x_i — входные значения нейрона;

w_i — веса нейрона;

$ReLU()$ — функция активации;

$\hat{y}$ — выходное значение нейрона

Способы инициализации

- Константа

$$w_i = const$$

- Стандартное нормальное распределение

$$w_i \sim \mathcal{N}(\mu = 0, \sigma^2 = 1)$$

- Равномерное распределение

$$w_i \sim \mathcal{U}(-a, a)$$

Метод инициализации Завьера (Xavier)

$$z = \sum_{i=1}^{n} w_i x_i$$

$$\hat{y} = ReLU(z)$$

Инициализация:

$$w_i \sim \mathcal{N}\left(\mu = 0, \sigma^2 = \frac{1}{n}\right),$$

где n — количество весов нейрона

Метод инициализации Завьера (Xavier)

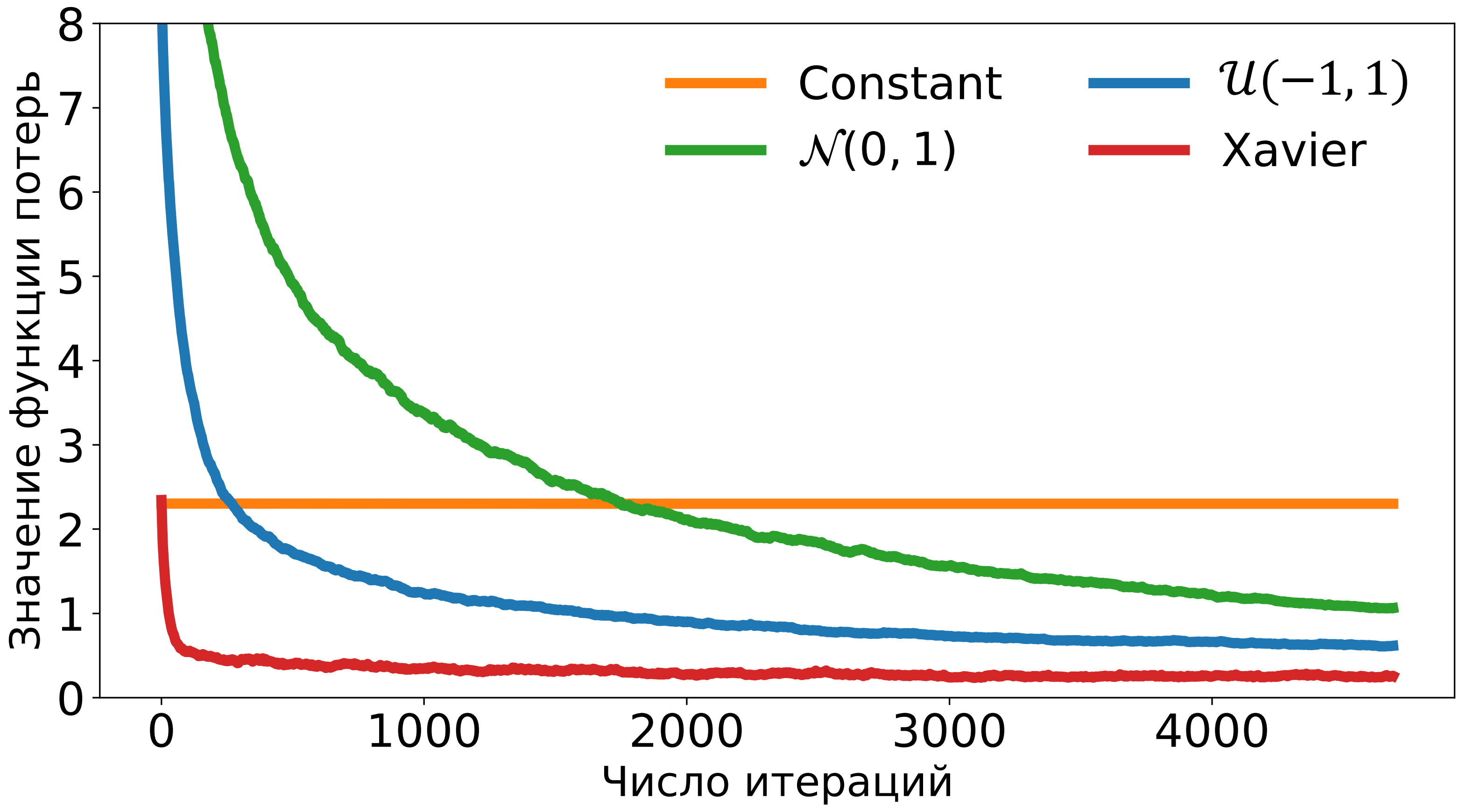
$$Var(z) = \boldsymbol{n} * \boldsymbol{Var(w_i)} * Var(x_i) = Var(x_i)$$

$$\mathbf{Var}(\hat{\boldsymbol{y}}) \approx \boldsymbol{Var(z)} = \boldsymbol{Var(x_i)}$$

- Дисперсия выхода нейрона = дисперсии входов нейрона
- Для любой функции активации
- Ускоряет обучение нейронной сети

Пример

- Результат обучения нейронной сети на данных MNIST по распознаванию изображений рукописных цифр



Инициализация константой

$$z = \sum_{i=1}^{n} w_i x_i$$

$$\hat{y} = ReLU(z)$$

Если $w_i = const$:

- Все нейроны в слое одинаковы
- Их значения и градиенты равны
- Сеть не обучается

Выводы

- Инициализация весов влияет на скорость обучения
- Метод Завьера сохраняет дисперсию выходов нейронов
- Значительное ускорение обучения сети

Поставьте свою оценку лекции