

# Глубокое обучение и вообще

Ульянкин Филипп

**Посиделка 3:** 50 оттенков градиентного спуска

# Agenda

- Стохастический градиентный спуск
- 50 оттенков градиентного спуска
- Эпохи, батчи, ранняя остановка

# Стохастический градиентный спуск



# Как обучать нейросеть?

- Нейросеть - сложная функция, зависящая от весов  $W$
- «Тренировка» — поиск оптимальных  $W$
- «Оптимальных» — минимизирующих какой-то функционал
- Какими бывают функционалы: MSE, MAE, logloss и многие другие
- Как оптимизировать: градиентный спуск

# Градиентный спуск



# Градиентный спуск (GD)

Проблема оптимизации:

$$L(w) = \frac{1}{n} \cdot \sum_{i=1}^n L(w, x_i, y_i) \rightarrow \min_w$$

# Градиентный спуск (GD)

Проблема оптимизации:

$$L(w) = \frac{1}{n} \cdot \sum_{i=1}^n L(w, x_i, y_i) \rightarrow \min_w$$

Градиент указывает направление максимального роста

$$\nabla L(w) = \left( \frac{\partial L(w)}{\partial w_0}, \frac{\partial L(w)}{\partial w_2}, \dots, \frac{\partial L(w)}{\partial w_k} \right)$$

# Градиентный спуск (GD)

Проблема оптимизации:

$$L(w) = \frac{1}{n} \cdot \sum_{i=1}^n L(w, x_i, y_i) \rightarrow \min_w$$

Градиент указывает направление максимального роста

$$\nabla L(w) = \left( \frac{\partial L(w)}{\partial w_0}, \frac{\partial L(w)}{\partial w_2}, \dots, \frac{\partial L(w)}{\partial w_k} \right)$$

Идём в противоположную сторону:

$$w_t = w_{t-1} - \eta \cdot \nabla L(w_{t-1})$$

скорость обучения

# Градиентный спуск (GD)

Проблема оптимизации:

$$L(w) = \sum_{i=1}^n L(w, x_i, y_i) \rightarrow \min_w$$

Инициализация  $w_0$

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

$$w_t = w_{t-1} - \eta_t \cdot g_t$$

**if**  $\|w_t - w_{t-1}\| < \varepsilon$  :

**break**

# Сходимость

- Останавливаем процесс, если

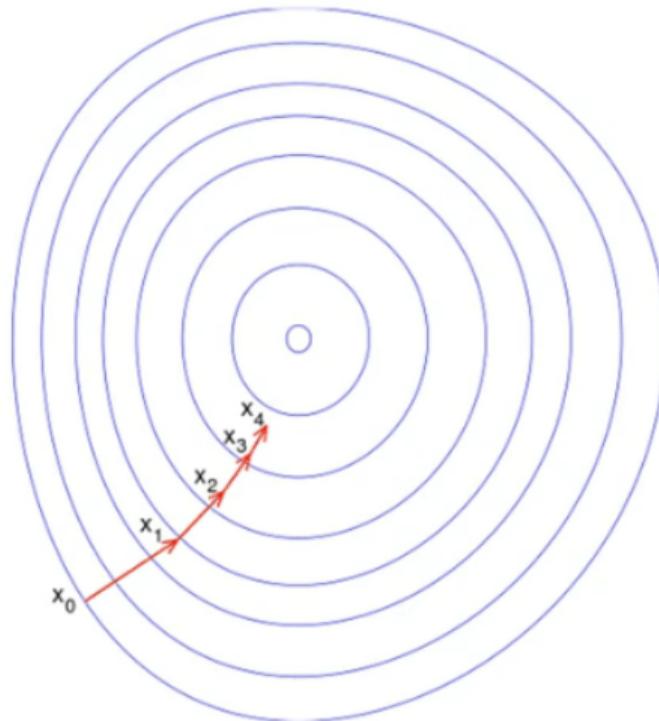
$$\|w_t - w_{t-1}\| < \varepsilon$$

- Другой вариант:

$$\|\nabla L(w_t)\| < \varepsilon$$

- Обычно в глубоком обучении: останавливаемся, когда ошибка на валидационной выборке перестаёт убывать

# Градиентный спуск



## Пример (линейная регрессия):

Проблема оптимизации:

$$L(w) = \frac{1}{n} \cdot \sum_{i=1}^n (y_i - x_i^T w)^2 \rightarrow \min_w$$

Градиент:

$$\nabla L(w) = -2 \cdot \frac{1}{n} \cdot \sum_{i=1}^n (y_i - x_i^T w) \cdot x_i$$

Идём в противоположную сторону:

$$w_t = w_{t-1} + 0.001 \cdot 2 \cdot \frac{1}{n} \cdot \sum_{i=1}^n (y_i - x_i^T w_{t-1}) \cdot x_i$$

# Пример (линейная регрессия):

Проблема оптимизации:

$$L(w) = \frac{1}{n} \cdot \sum_{i=1}^n (y_i - x_i^T w)^2 \rightarrow \min_w$$

Градиент:

$$\nabla L(w) = -2 \cdot \frac{1}{n} \cdot \sum_{i=1}^n (y_i - x_i^T w) \cdot x_i$$

Идём в противоположную сторону:

$$w_t = w_{t-1} + 0.001 \cdot -2 \cdot \frac{1}{n} \cdot \sum_{i=1}^n (y_i - x_i^T w_{t-1}) \cdot x_i$$

Дорого постоянно считать такие суммы по всей выборке

# Стохастический градиентный спуск (SGD)

Проблема оптимизации:

$$L(w) = \sum_{i=1}^n L(w, x_i, y_i) \rightarrow \min_w$$

Инициализация  $w_0$

рандомно выбрали  $i$

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

$$w_t = w_{t-1} - \eta_t \cdot g_t$$

**if**  $\|w_t - w_{t-1}\| < \varepsilon$  :

**break**

# Пример (линейная регрессия):

Проблема оптимизации:

$$L(w) = \frac{1}{n} \cdot \sum_{i=1}^n (y_i - x_i^T w)^2 \rightarrow \min_w$$

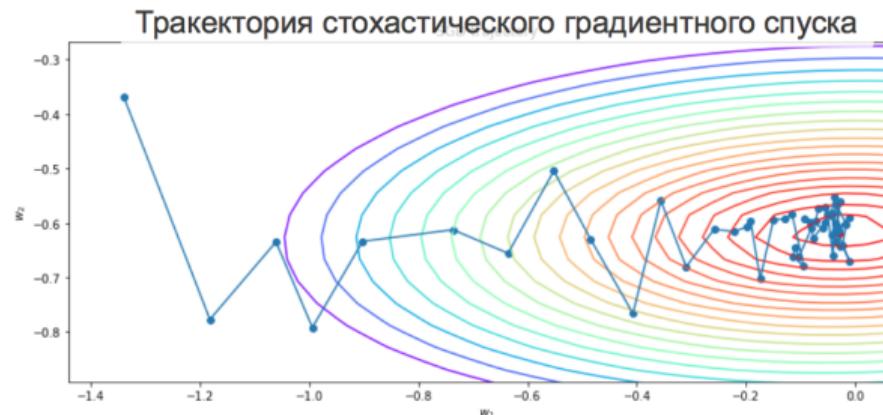
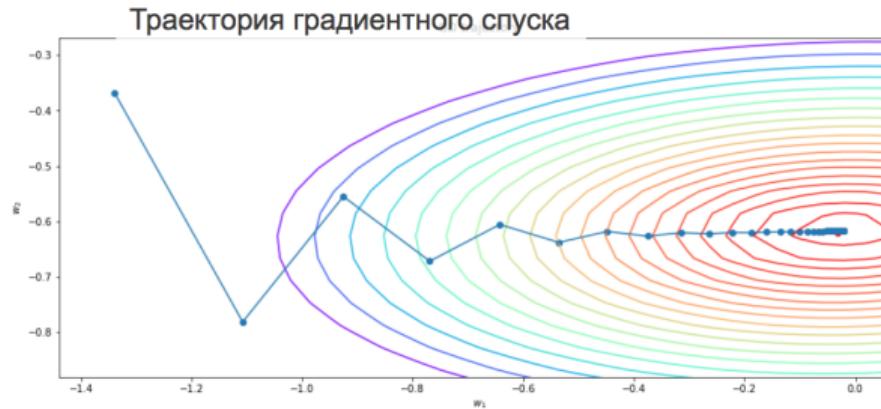
Градиент:

$$\nabla L(w) = -2 \cdot (y_i - x_i^T w) \cdot x_i$$

Идём в противоположную сторону:

$$w_t = w_{t-1} + 0.001 \cdot 2 \cdot (y_i - x_i^T w_{t-1}) \cdot x_i$$

# Скорость обучения в SGD



# Стохастический градиентный спуск (SGD)

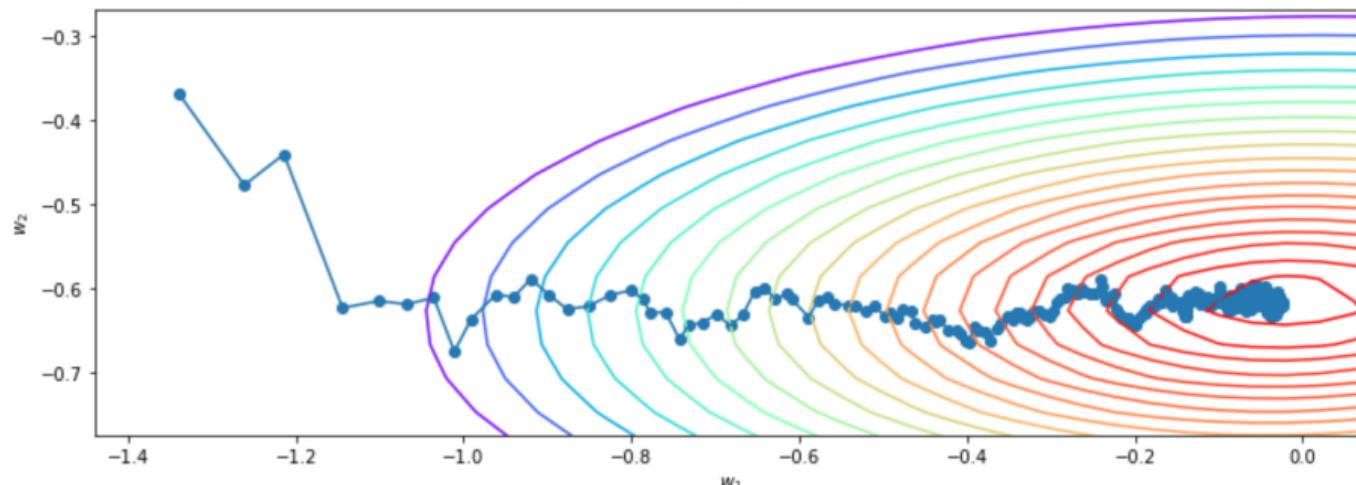
- Можно доказать, что оценка по одному объекту несмешённая, то есть в среднем мы идём в правильную сторону
- Даже в точке оптимума оценка по одному объекту вряд ли будет нулевой, поэтому важно, чтобы длина шага стремилась к нулю
- Длина шага должна зависить от номера итерации следующим образом:

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

- Сходимость к глобальному минимуму гарантируется только для выпуклых функций

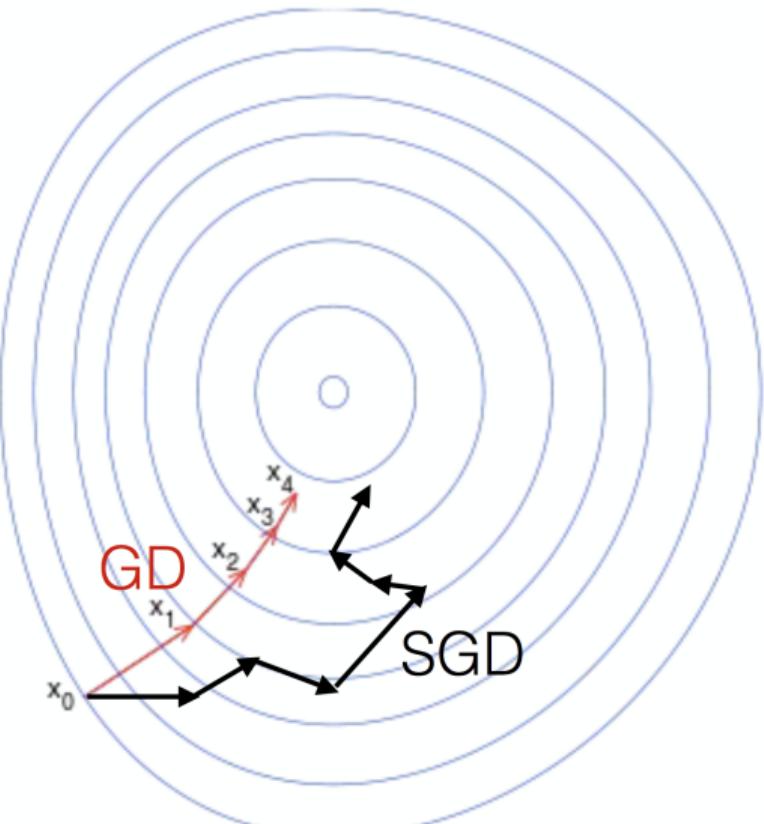
# Скорость обучения в SGD

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



Мы инженеры и используем то, что работает

# GD vs SGD



- И для GD и для SGD нет гарантий глобального минимума, сходимости
- SGD быстрее, на каждой итерации используется только одно наблюдение
- Для SGD спуск очень зашумлён
- GD:  $O(n)$ , SGD:  $O(1)$
- Шум в оценке градиента помогает выпрыгивать из локальных оптимумов

# Mini-bath SGD

Проблема оптимизации:

$$L(w) = \sum_{i=1}^n L(w, x_i, y_i) \rightarrow \min_w$$

Инициализация  $w_0$

рандомно выбрали  $m < n$  объектов

$$g_t = \frac{1}{m} \sum_{i=1}^m \nabla L(w, x_i, y_i)$$

$$w_t = w_{t-1} - \eta_t \cdot g_t$$

**if**  $\|w_t - w_{t-1}\| < \varepsilon$  :

**break**

## Mini-batch SGD

- Размер батча обычно десятки или сотни наблюдений
- Имеет смысл брать степень двойки
- Возможно, делает оценку градиента более стабильной
- За счёт векторизации также эффективен, как шаг по одному объекту

## Mini-batch SGD (2018)

The collected experimental results for the CIFAR-10, CIFAR-100 and ImageNet datasets show that increasing the mini-batch size progressively reduces the range of learning rates that provide stable convergence and acceptable test performance. On the other hand, small mini-batch sizes provide more up-to-date gradient calculations, which yields more stable and reliable training. The best performance has been consistently obtained for mini-batch sizes between  $m = 2$  and  $m = 32$ , which contrasts with recent work advocating the use of mini-batch sizes in the thousands.

<https://arxiv.org/abs/1804.07612>

# Mini-batch SGD (2018)



**Yann LeCun**  
@ylecun

...

Training with large minibatches is bad for your health.  
More importantly, it's bad for your test error.  
Friends dont let friends use minibatches larger than 32.  
[arxiv.org/abs/1804.07612](https://arxiv.org/abs/1804.07612)

[Перевести твит](#)

12:00 AM · 27 апр. 2018 г. · Facebook

---

**476** ретвитов

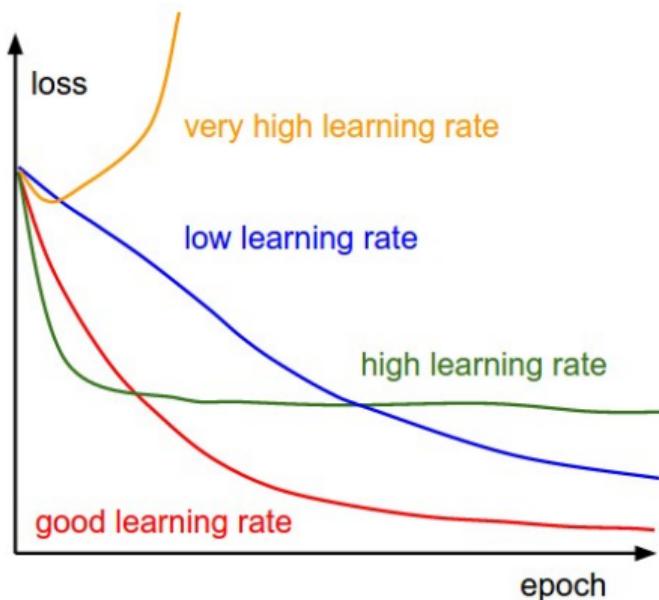
**38** твитов с цитатами

**1 346** отметок «Нравится»

<https://arxiv.org/abs/1804.07612>

# Скорость обучения

- Скорость обучения  $\eta$  надо подбирать аккуратно, если она будет большой, мы можем скакать вокруг минимума, если маленькой - вечно ползти к нему

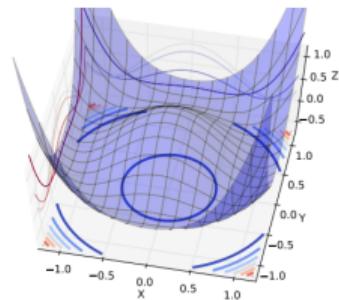
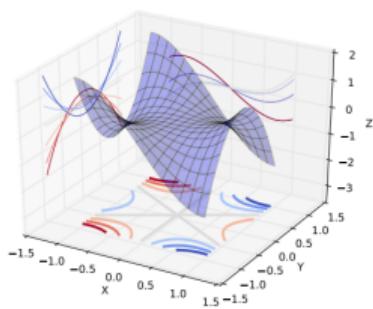
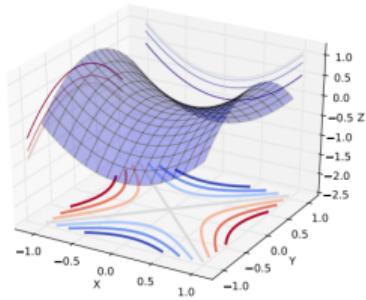
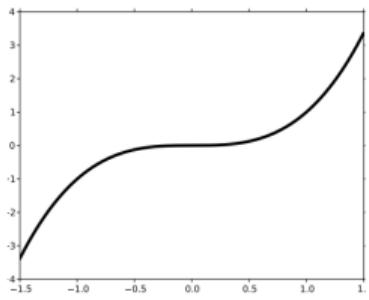


# Боб чилит в локальном минимуме



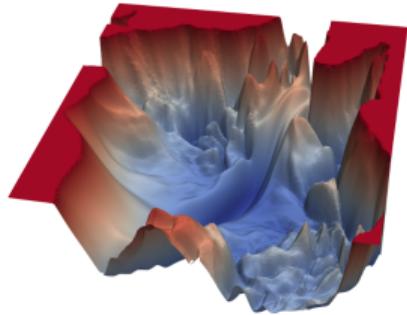
<https://hackernoon.com/life-is-gradient-descent-880c60ac1be8>

# Седловые точки

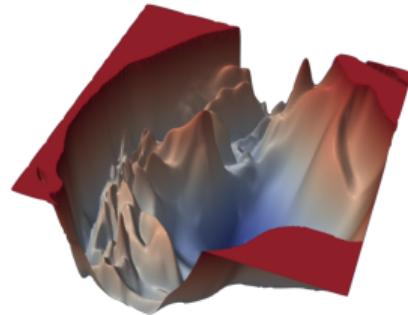


# Визуализация потерь

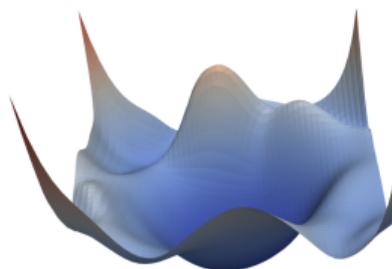
VGG-56



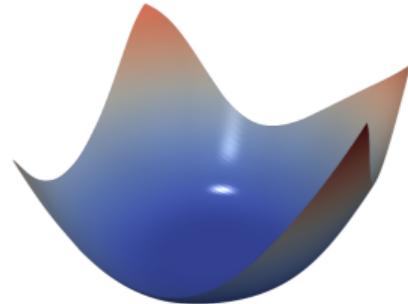
VGG-110



Renset-56



Densenet-121



<https://arxiv.org/pdf/1712.09913.pdf>

<https://github.com/tomgoldstein/loss-landscape>

# 50 оттенков градиентного спуска



# Mini-bath SGD

Проблема оптимизации:

$$L(w) = \sum_{i=1}^n L(w, x_i, y_i) \rightarrow \min_w$$

Инициализация  $w_0$

рандомно выбрали  $m < n$  индексов

$$g_t = \frac{1}{m} \sum_{i=1}^m \nabla L(w, x_i, y_i)$$

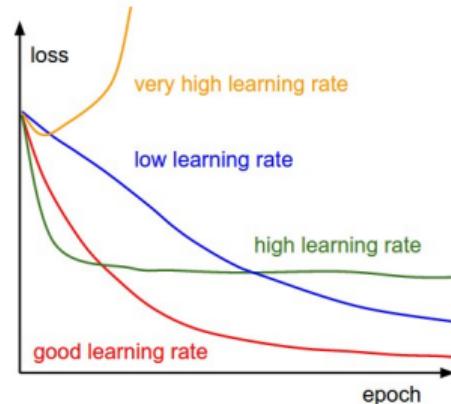
$$w_t = w_{t-1} - \eta_t \cdot g_t$$

**if**  $\|w_t - w_{t-1}\| < \varepsilon$  :

**break**

# Вызовы

- Скорость обучения  $\eta$  надо подбирать аккуратно, если она будет большой, мы можем скакать вокруг минимума, если маленькой - вечно ползти к нему.



- К обновлению всех параметров применяется одна и та же скорость обучения. Возможно, что какие-то параметры приходят в оптимальную точку быстрее, и их не надо обновлять.

# Momentum SGD

Мы считали на каждом шаге градиент по формуле

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

После шага мы забывали его, **давайте запоминать направление:**

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

$$w_t = w_{t-1} - h_t$$

- Движение поддерживается в том же направлении
- Нет резких изменений направления движения
- Обычно  $\alpha = 0.9$ .

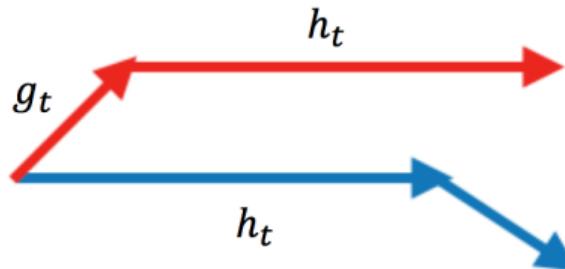
Крутой интерактив для моментума: <https://distill.pub/2017/momentum/>

# Momentum SGD

- Бежим с горки и всё больше ускоряемся в том направлении, в котором были направлены сразу несколько предыдущих градиентов, но при этом движемся медленно там, где градиент постоянно меняется
- Хотелось бы не просто бежать с горы, но и хотя бы на полшага смотреть себе под ноги, чтобы внезапно не споткнуться  $\Rightarrow$  **давайте смотреть на градиент в будущей точке**
- Согласно методу моментов  $\alpha \cdot h_{t-1}$  точно будет использоваться при шаге, давайте искать  $\nabla L(w_{t-1} - \alpha \cdot h_{t-1})$ .

# Nesterov Momentum SGD

- Мы теперь сначала прыгаем в том же направлении, в каком шли до этого, потом корректируем его (голубая траектория).



$$h_t = \alpha \cdot h_{t-1} + \eta \cdot \nabla L(w_{t-1} - \alpha \cdot h_{t-1})$$

$$w_t = w_{t-1} - h_t$$

# Разная скорость обучения

- Может сложиться, что некоторые веса уже близки к своим локальным минимумам, по этим координатам надо двигаться медленнее, а по другим быстрее  $\Rightarrow$  **адаптивные методы градиентного спуска**
- Шаг изменения должен быть меньше у тех параметров, которые в большей степени варьируются в данных, и больше у тех, которые менее изменчивы

# AdaGrad

$$G_t^j = G_{t-1}^j + g_{tj}^2$$
$$w_t^j = w_{t-1}^j - \frac{\eta}{\sqrt{G_t^j + \varepsilon}} \cdot g_t^j$$

- $g_t^j$  — градиент по  $j$ -ому параметру
- своя скорость обучения для каждого параметра
- обычно  $\eta = 0.01$ , т.к. параметр не очень важен
- $G_t^j$  всегда увеличивается, из-за этого обучение может рано останавливаться  $\Rightarrow$  RMSprop

# RMSprop

$$G_t^j = \alpha \cdot G_{t-1}^j + (1 - \alpha) \cdot g_{tj}^2$$
$$w_t^j = w_{t-1}^j - \frac{\eta_t}{\sqrt{G_t^j + \varepsilon}} \cdot g_t^j$$

- Обычно  $\alpha = 0.9$
- Скорость обучения адаптируется к последнему сделанному шагу, бесконтрольного роста  $G_t^j$  больше не происходит
- RMSprop нигде не был опубликован, Хинтон просто привёл его в своей лекции, сказав, что это норм тема

# Adam (Adaptive Moment Estimation)

- Комбинируем Momentum и индивидуальные скорости обучения

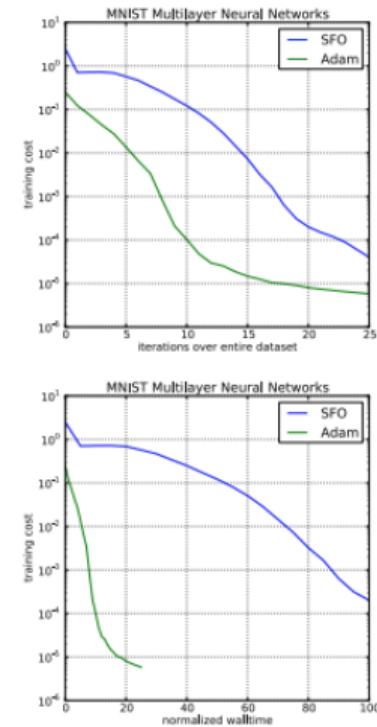
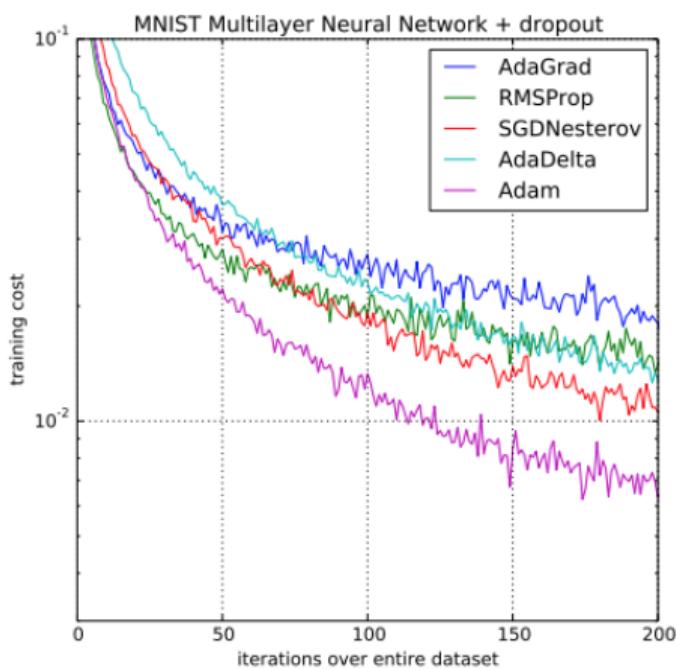
$$h_t^j = \frac{\beta_1 \cdot h_{t-1}^j + (1 - \beta_1) \cdot g_{tj}}{1 - \beta_1^t}$$

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

$$w_t^j = w_{t-1}^j - \frac{\eta_t}{\sqrt{G_t^j + \varepsilon}} \cdot h_t^j$$

- Фактически  $h_t$  и  $G_t$  это оценки первого и второго моментов для стохастического градиента

# Сравнение на MNIST



<https://arxiv.org/pdf/1412.6980.pdf>

# Резюме по методам градиентного спуска

- Momentum SGD сохраняет направление шага и позволяет добиваться более быстрой сходимости
- Адаптивные методы позволяют находить индивидуальную скорость обучения для каждого параметра
- Adam комбинирует в себе оба подхода
- Давайте посмотрим [визуализацию 1](#) и [визуализацию 2](#)
- В будущих лекциях мы продолжим разговор про то, как можно улучшить методы оптимизации нейронных сетей.

# Эпохи, батчи, ранняя остановка



# Нейросети и кросс-валидация

- Кросс-валидацию для нейронных сетей обычно не делают
- Сетка учится долго, дробить выборку на части и обучать несколько экземпляров очень дорого
- Перебор гиперпараметров по решётке обычно не делают, так как это тоже дорого
- При экспериментах делают одно какое-то изменение за раз и запускают обучение

# Эпохи и батчи

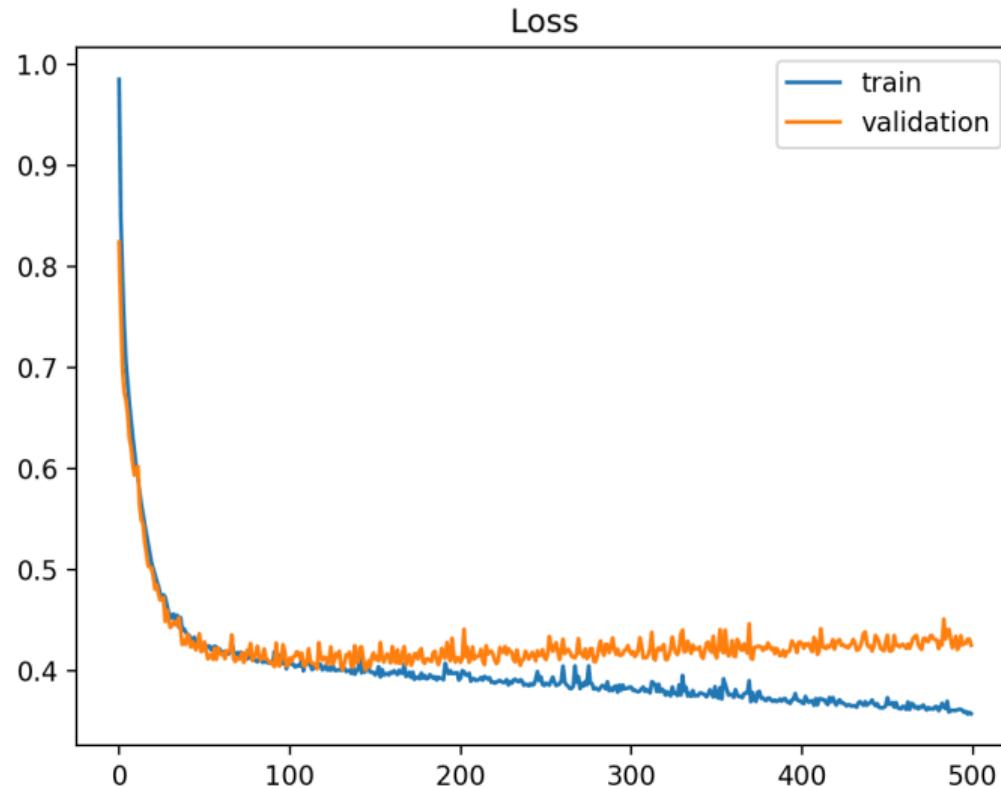
```
Epoch 1/10
100/100 [=====] - 57s 566ms/step - loss: 0.3263 - acc: 0.9115 - val_loss: 1.0140 - val_acc: 0.6790
Epoch 2/10
100/100 [=====] - 57s 569ms/step - loss: 0.2455 - acc: 0.9280 - val_loss: 0.9344 - val_acc: 0.7670
Epoch 3/10
100/100 [=====] - 56s 562ms/step - loss: 0.2805 - acc: 0.9255 - val_loss: 0.4332 - val_acc: 0.8910
Epoch 4/10
100/100 [=====] - 56s 564ms/step - loss: 0.2556 - acc: 0.9280 - val_loss: 0.2783 - val_acc: 0.9330
Epoch 5/10
100/100 [=====] - 56s 564ms/step - loss: 0.2200 - acc: 0.9335 - val_loss: 0.1693 - val_acc: 0.9530
Epoch 6/10
100/100 [=====] - 56s 564ms/step - loss: 0.2522 - acc: 0.9335 - val_loss: 0.1427 - val_acc: 0.9640
Epoch 7/10
100/100 [=====] - 56s 563ms/step - loss: 0.2287 - acc: 0.9400 - val_loss: 0.1284 - val_acc: 0.9640
Epoch 8/10
100/100 [=====] - 56s 564ms/step - loss: 0.2013 - acc: 0.9400 - val_loss: 0.1183 - val_acc: 0.9700
Epoch 9/10
100/100 [=====] - 56s 562ms/step - loss: 0.2253 - acc: 0.9370 - val_loss: 0.1147 - val_acc: 0.9700
Epoch 10/10
100/100 [=====] - 56s 563ms/step - loss: 0.2056 - acc: 0.9440 - val_loss: 0.1418 - val_acc: 0.9640
```

- Эпоха — один проход по данным
- Батч — маленький кусочек данных
- Перед каждой эпохой данные надо перемешивать

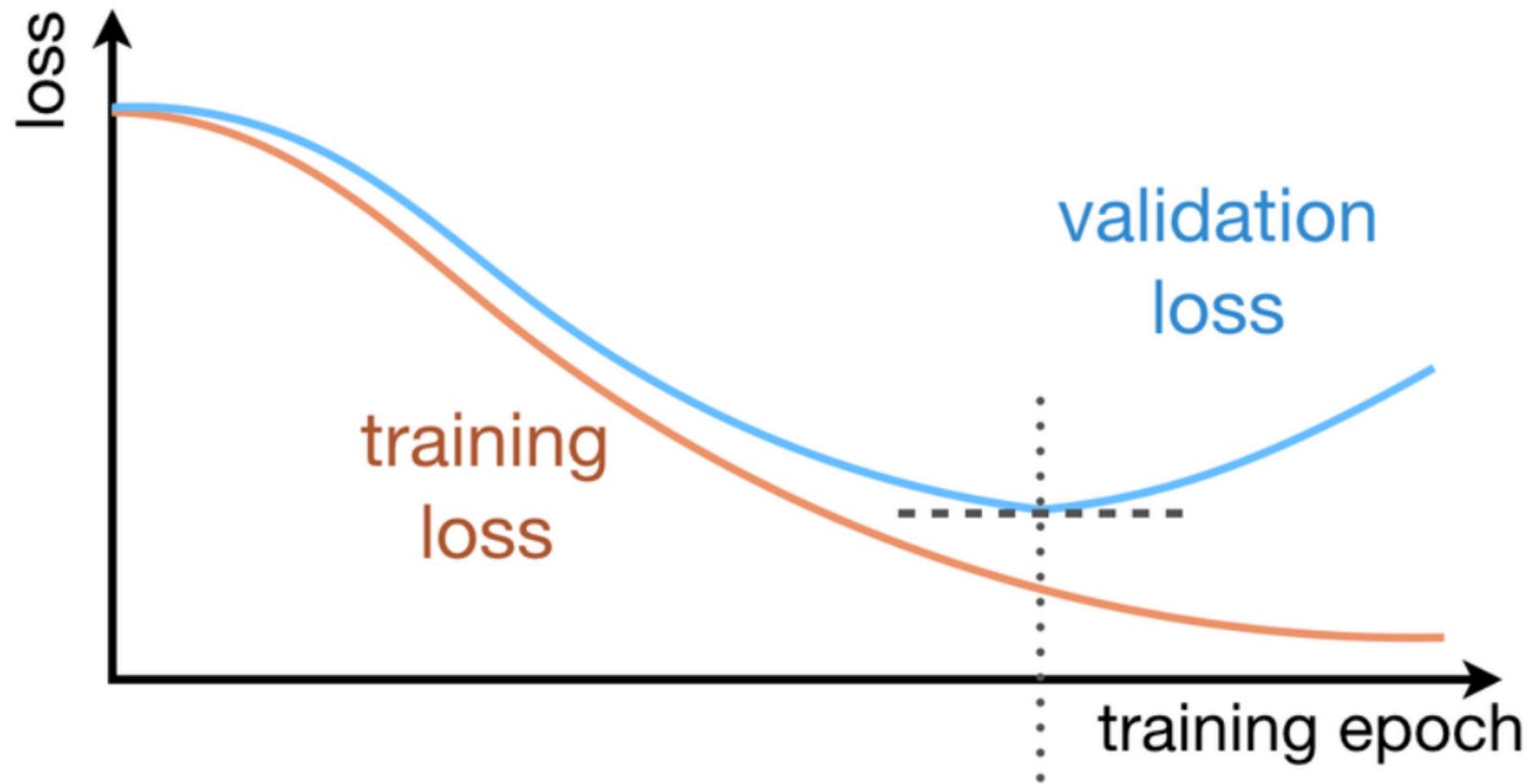
# Как отслеживать обучение? Графики!

- Значение функции потерь в зависимости от итерации (проверка идёт ли оптимизация)
- Обучающая выборка не покажет переобучение  $\Rightarrow$  следим за ошибкой на валидационной выборке
- **ВАЖНО:** использовать валидацию, а не тест
- Число эпох для обучения - гиперпараметр, на валидации можно понять, когда надо остановить обучение
- На тестовой выборке оцениваем окончательное качество модели

# Переобучение



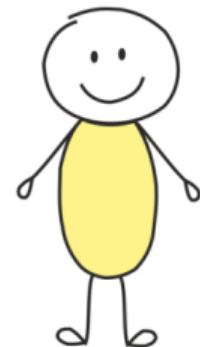
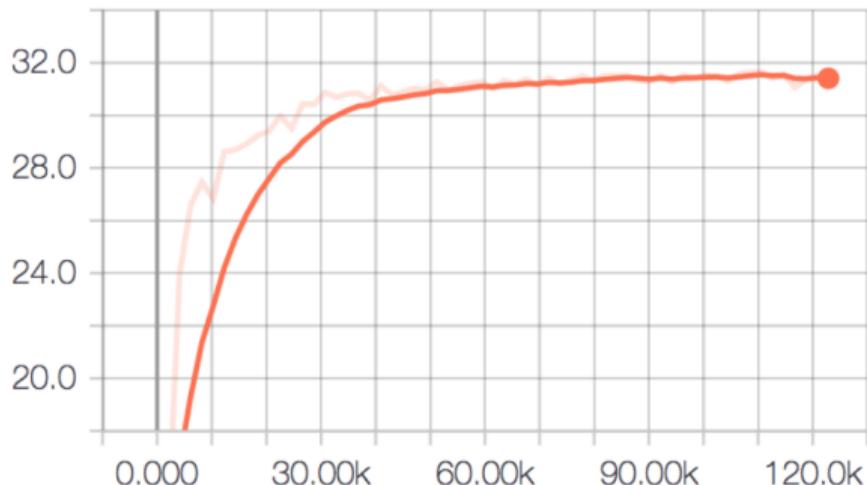
## Ранняя остановка



# Tensorboard of a healthy man

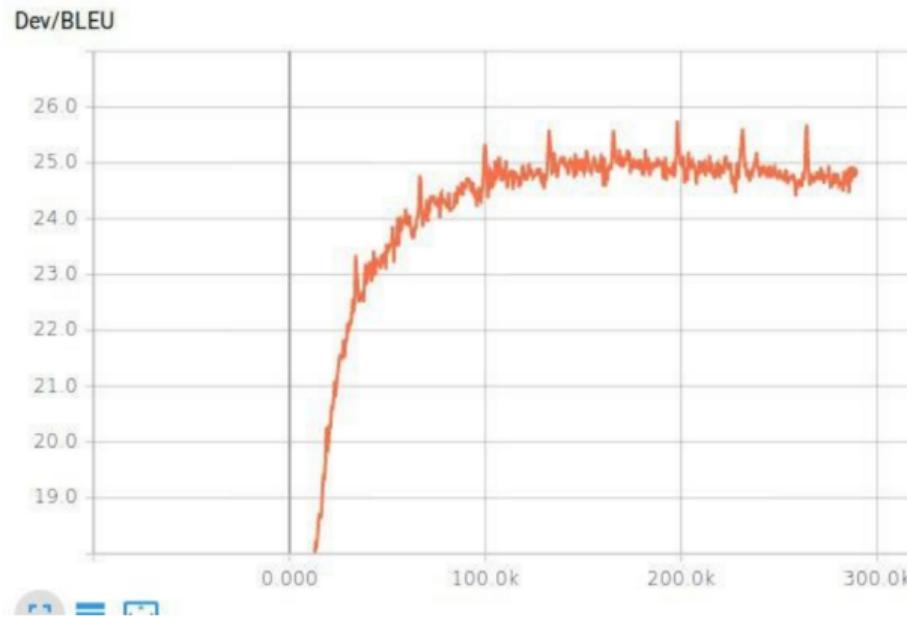
---

Dev/BLEU



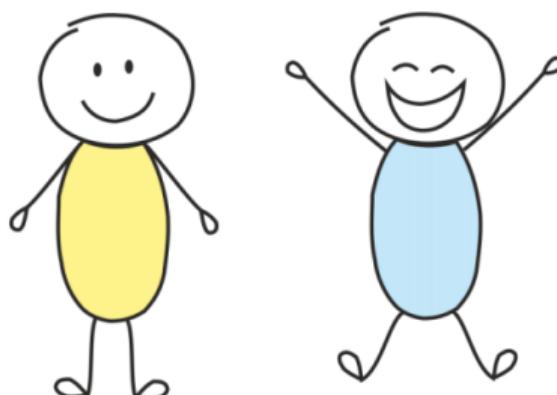
[https://github.com/yandexdataschool/nlp\\_course/tree/2019/week01\\_embeddings](https://github.com/yandexdataschool/nlp_course/tree/2019/week01_embeddings)  
[https://lena-voita.github.io/nlp\\_course.html](https://lena-voita.github.io/nlp_course.html)

# Tensorboard of a man who doesn't shuffle his data



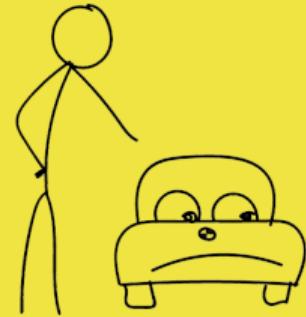
[https://github.com/yandexdataschool/nlp\\_course/tree/2019/week01\\_embeddings](https://github.com/yandexdataschool/nlp_course/tree/2019/week01_embeddings)  
[https://lena-voita.github.io/nlp\\_course.html](https://lena-voita.github.io/nlp_course.html)

# Shuffle your data!



[https://github.com/yandexdataschool/nlp\\_course/tree/2019/week01\\_embeddings](https://github.com/yandexdataschool/nlp_course/tree/2019/week01_embeddings)  
[https://lena-voita.github.io/nlp\\_course.html](https://lena-voita.github.io/nlp_course.html)

# Как обучить нейросеть?



Ты необучаем!

# Нейросеть — сложная функция

- Прямое распространение ошибки (forward propagation):

$$X \Rightarrow X \cdot W_1 \Rightarrow f(X \cdot W_1) \Rightarrow f(X \cdot W_1) \cdot W_2 \Rightarrow \dots \Rightarrow \hat{y}$$

- Считаем потери:

$$Loss = \frac{1}{2}(y - \hat{y})^2$$

- Все слои обычно дифференцируемы, поэтому можно посчитать производные по всем параметрам
- Для обучения можно использовать градиентный спуск

# Как обучить нейросеть?

$$L(W_1, W_2) = \frac{1}{2} \cdot (y - f(X \cdot W_1) \cdot W_2)^2$$

Секрет успеха в умении брать производную

# Как обучить нейросеть?

$$L(W_1, W_2) = \frac{1}{2} \cdot (y - f(X \cdot W_1) \cdot W_2)^2$$

Секрет успеха в умении брать производную

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

# Как обучить нейросеть?

$$L(W_1, W_2) = \frac{1}{2} \cdot (y - f(X \cdot W_1) \cdot W_2)^2$$

Секрет успеха в умении брать производную

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

$$\frac{\partial L}{\partial W_2} = -(y - f(X \cdot W_1) \cdot W_2) \cdot f(X \cdot W_1)$$

$$\frac{\partial L}{\partial W_1} = -(y - f(X \cdot W_1) \cdot W_2) \cdot W_2 \cdot f'(X \cdot W_1) \cdot X$$

# Как обучить нейросеть?

$$L(W_1, W_2) = \frac{1}{2} \cdot (y - f(X \cdot W_1) \cdot W_2)^2$$

Секрет успеха в умении брать производную

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

$$\frac{\partial L}{\partial W_2} = -(y - f(X \cdot W_1) \cdot W_2) \cdot f(X \cdot W_1)$$

$$\frac{\partial L}{\partial W_1} = -(y - f(X \cdot W_1) \cdot W_2) \cdot W_2 \cdot f'(X \cdot W_1) \cdot X$$

Дважды ищем одно и то же  $\Rightarrow$  оптимизация поиска производных даст нам алгоритм обратного распространения ошибки (back-propagation)

# Back-propagation

