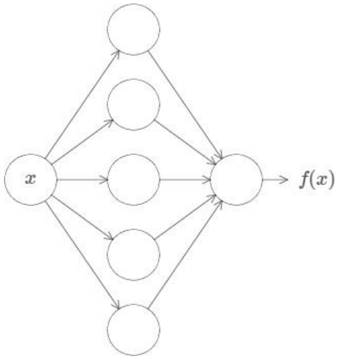
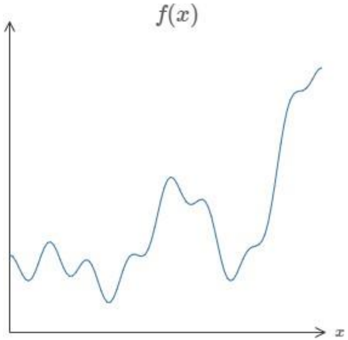
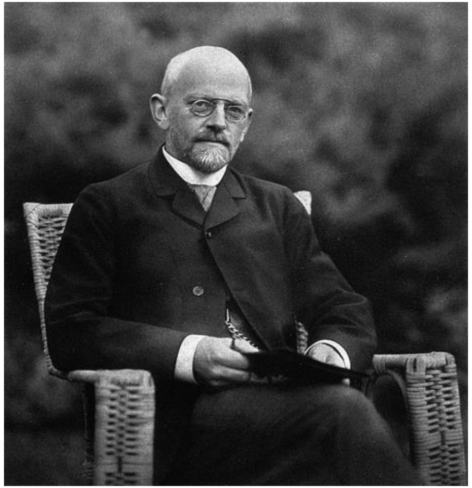
Лекция 8:
Обучение нейронных сетей,
Часть 2

AddOn: Интуиция теоремы Цыбенко:

Может ли нейронная сеть аппроксимировать произвольную функцию?







Дэвид Гильберт

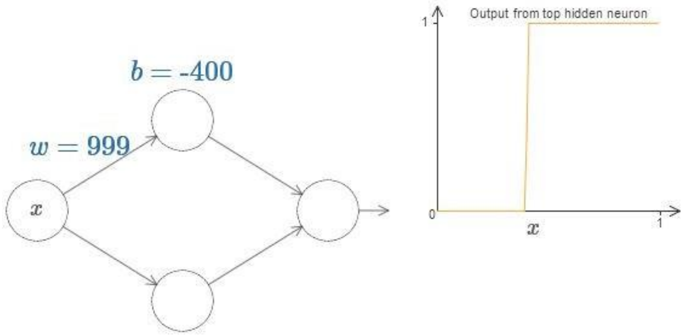
1900 - 13 проблема Гильберта - доказательство существования решений для всех уравнений 7-мой степени в виде алгебраических (непрерывных) функций.

1956 - Теорема Колмогорова-Арнольда о представлении.
Каждую многомерную непрерывную функцию можно записать в виде конечной композиции непрерывных функций одной переменной и бинарной операции сложения.

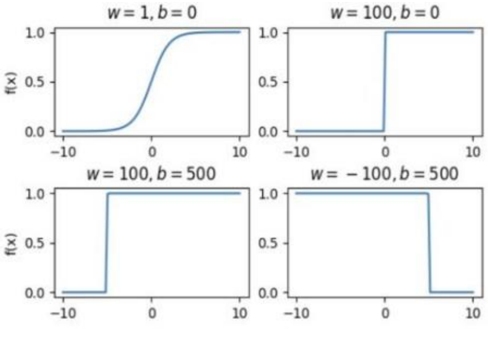
1989 - Универсальная теорема аппроксимации (Цыбенко). Любую функцию можно аппроксимировать сетью прямого распространения с одним скрытым слоем и функциями активации сигмоидального типа.

2021 - Проблема вновь актуальна для алгебраических функций!
Zinovy Reichstein, From Hilbert’s 13th problem to essential dimension and back, EMS magazine, 2021
https://habr.com/ru/post/544266/

AddOn: Интуиция теоремы Цыбенко:

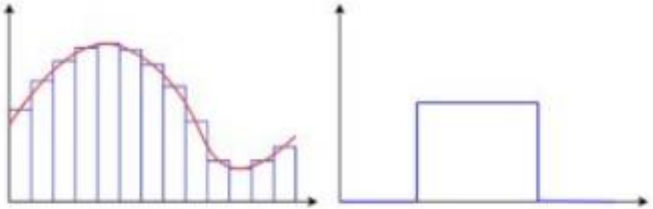


Сигмоидальный нейрон дает единичный скачок



The neuron output based on different values of w and b. The network input x is represented on the x axis.

Сигмоидальный нейрон дает прямоугольный импульс



The diagram on the left depicts continuous function approximation with a series of step functions, while the diagram on the right illustrates a single boxcar step function.

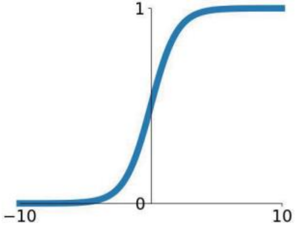
На основе прямоугольных импульсов можно "построить" аппроксимацию произвольной функции

Идея отсюда:
http://neuralnetworksanddeeplearning.com/chap4.html
Книга:
Advanced Deep Learning with Python. By Ivan Vasilev

Вспоминаем: Функции активации

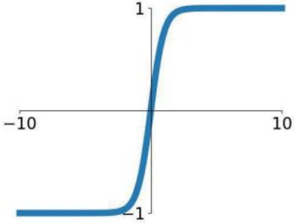
Sigmoid

$$\sigma(x) = \frac{1}{1+e^{-x}}$$



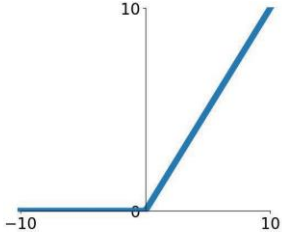
tanh

$$\tanh(x)$$



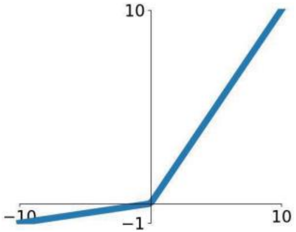
ReLU

$$\max(0, x)$$



Leaky ReLU

$$\max(0.1x, x)$$

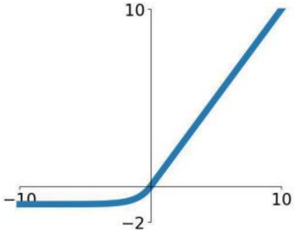


Maxout

$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

ELU

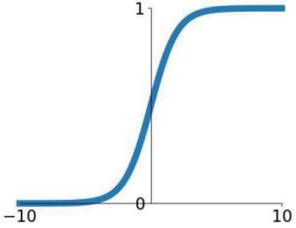
$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



Вспоминаем: Функции активации

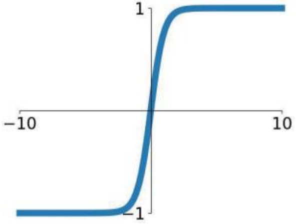
Sigmoid

$\sigma(x) = \frac{1}{1+e^{-x}}$



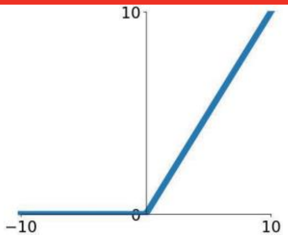
tanh

$\tanh(x)$



ReLU

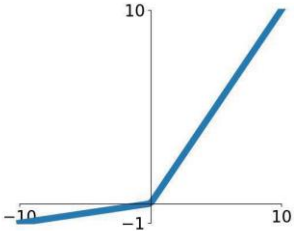
$\max(0, x)$



По умолчанию

Leaky ReLU

$\max(0.1x, x)$

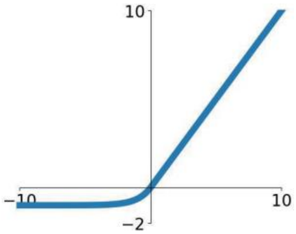


Maxout

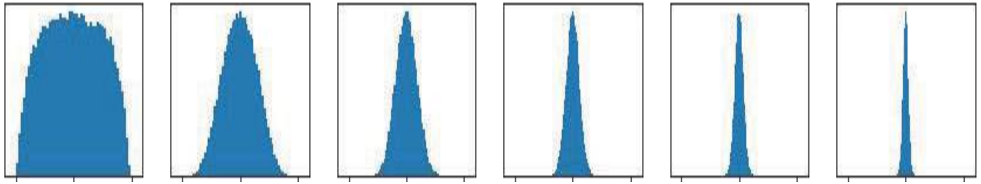
$\max(w_1^T x + b_1, w_2^T x + b_2)$

ELU

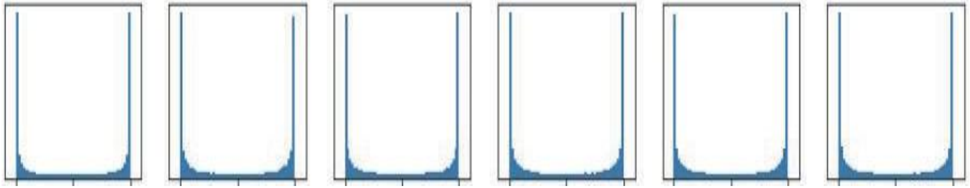
$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



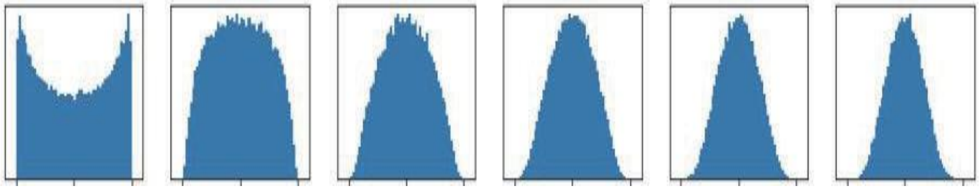
Вспоминаем: инициализация весов



Инициализация маленькая:
Активации нулевые, градиенты нулевые,
не учится =(

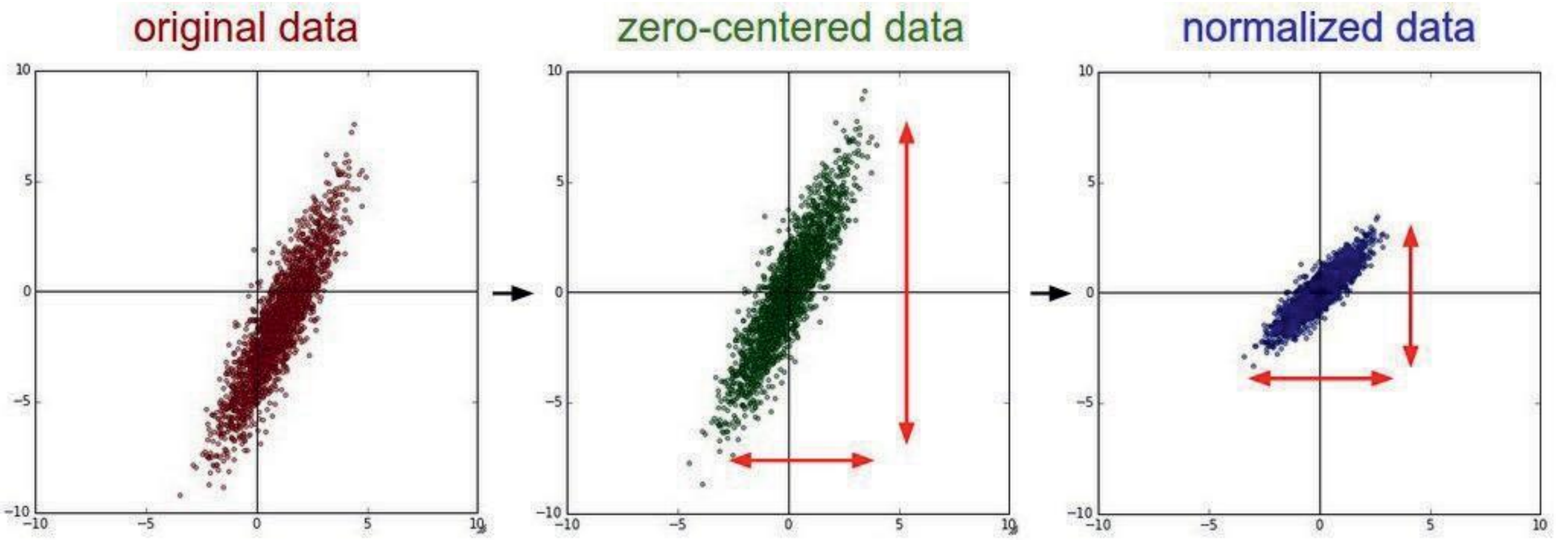


Инициализация большая:
Насыщение активаций (tanh),
Нулевые градиенты,
не учится =(



Хорошая инициализация:
Хорошие распределения активаций по всем слоям, Учится! =)

Вспоминаем: подготовка данных



Вспоминаем: Batch Normalization

[Ioffe and Szegedy, 2015]

Вход: $x : N \times D$

$$\mu_j = \frac{1}{N}\sum_{i=1}^{N} x_{i,j}$$

Поканальное среднее, размерности D

Обучаемые параметры масштаб и сдвиг:

$$\gamma, \beta : D$$

$$\sigma_j^2 = \frac{1}{N}\sum_{i=1}^{N}(x_{i,j} - \mu_j)^2$$

Поканальная дисперсия, размерности D

$$\hat{x}_{i,j} = \frac{x_{i,j} - \mu_j}{\sqrt{\sigma_j^2 + \varepsilon}}$$

Нормализованный x, размерности N x D

Подобра $\gamma = \sigma$, $\beta = \mu$ получим идентичное преобразование!

$$y_{i,j} = \gamma_j \hat{x}_{i,j} + \beta_j$$

Выход, размерности N x D

Сегодня:

- Минимизируем ошибку обучения:
  - *Крутые* оптимизаторы (Optimizers)
  - Политики Learning rate
- Минимизируем ошибку на тесте:
  - Регуляризация
  - Подбор гиперпараметров

Minimizing of the cost function J(θ) over the data

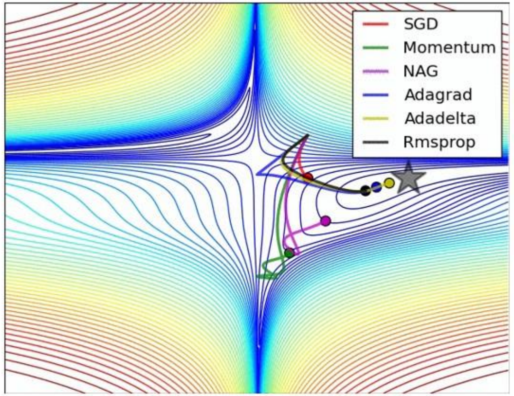
$\theta = \theta - \eta \cdot \nabla_\theta J(\theta)$. «Ванильный» градиентный спуск

$\theta = \theta - \eta \cdot \nabla_\theta J(\theta; x^{(i)}; y^{(i)})$. Стохастический ГС η(λ) – learning rate

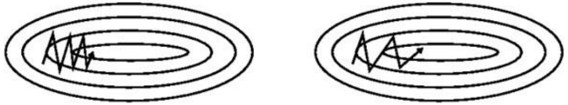
$\theta = \theta - \eta \cdot \nabla_\theta J(\theta; x^{(i:i+n)}; y^{(i:i+n)})$. Mini-batch SGD – пакетный СГС

$$v_t = \gamma v_{t-1} + \eta \nabla_\theta J(\theta)$$
$$\theta = \theta - v_t$$

Модификации SGD учитывают анизотропию фазового пространства – Adam etc.



Momentum γ:



Регуляризация наше все!

- Weight decay
- Dropout
- Pruning – контрастирование
- Batch-norm

2. Weight penalty terms

L2 weight decay

$$E = \frac{1}{2}\sum_j \left(t_j - y_j\right)^2 + \frac{\lambda}{2}\sum_{i,j} w_{ji}^2$$

$$\Delta w_{ji} = \varepsilon \delta_j x_i - \varepsilon \lambda w_{ji}$$

L1 weight decay

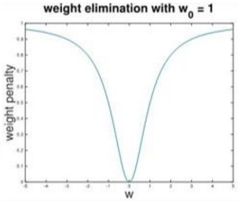
$$E = \frac{1}{2}\sum_j \left(t_j - y_j\right)^2 + \frac{\lambda}{2}\sum_{i,j} \left|w_{ji}\right|$$

$$\Delta w_{ji} = \varepsilon \delta_j x_i - \varepsilon \lambda \operatorname{sign}(w_{ji})$$

weight elimination

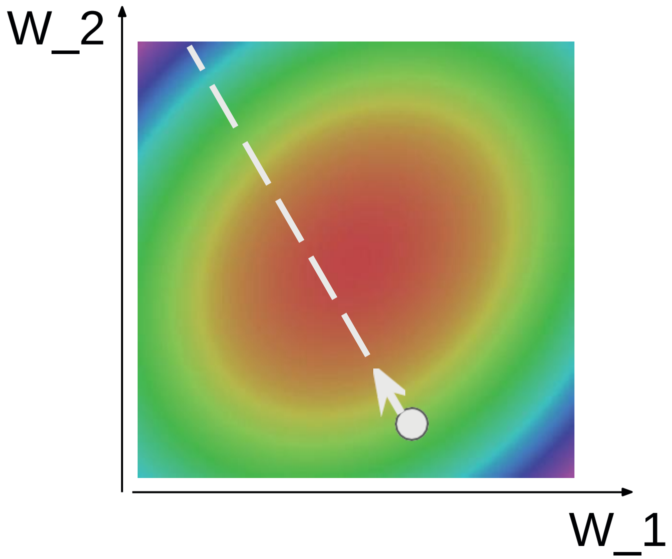
$$E = \frac{1}{2}\sum_j \left(t_j - y_j\right)^2 + \frac{\lambda}{2}\sum_{i,j} \frac{w_{ji}^2 / w_0^2}{1 + w_{ji}^2 / w_0^2}$$

See Reed (1993) for survey of ‘pruning’



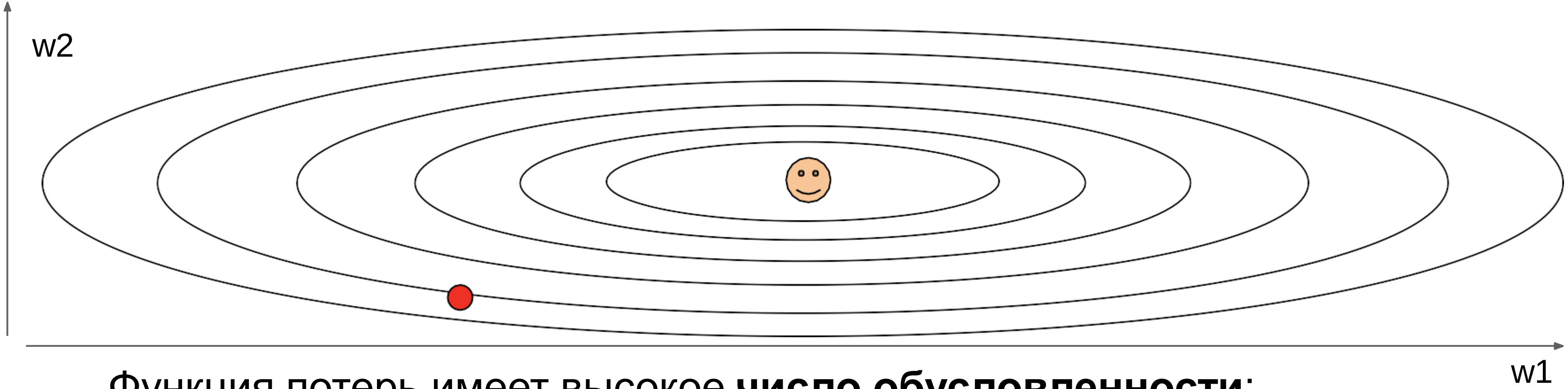
Оптимизация

```
# Vanilla Gradient Descent

while True:
  weights_grad = evaluate_gradient(loss_fun, data, weights)
  weights += - step_size * weights_grad # perform parameter update
```



Оптимизация: проблемы SGD

Что если ошибка спадает быстро по одному направлению и медленно по другому?
Что будет с градиентным спуском?



Функция потерь имеет высокое **число обусловленности**: отношение максимального собственного числа Гессиана к минимальному большое

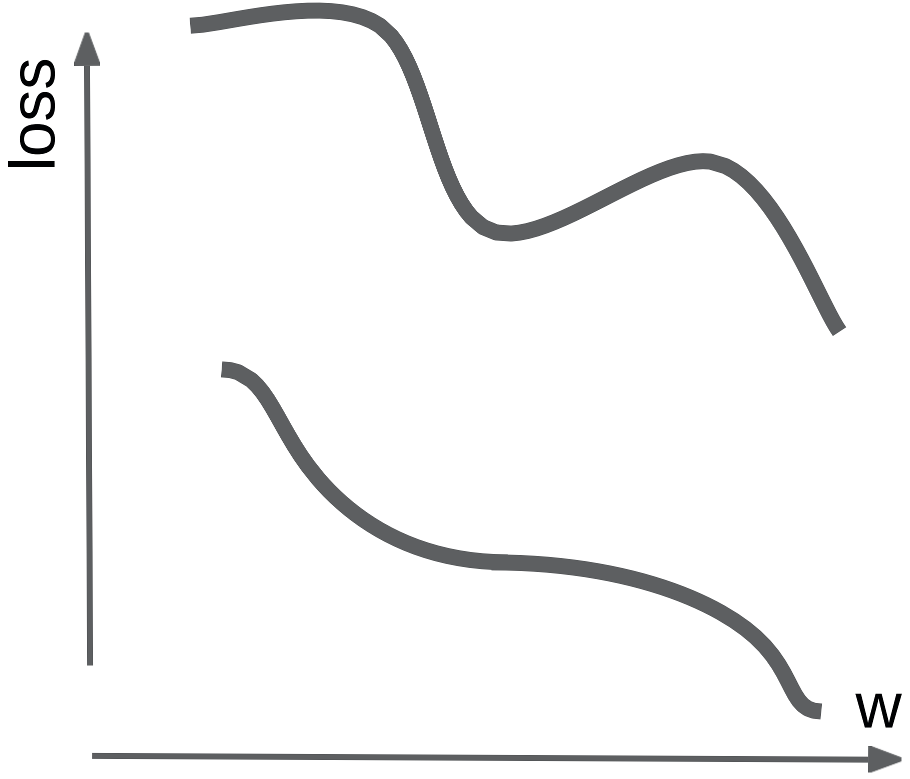
Оптимизация: проблемы SGD

Что если ошибка спадает быстро по одному направлению и медленно по другому?
Что будет с градиентным спуском?
Низкая скорость по одному направлению, разброс по другому направлению

Функция потерь имеет высокое **число обусловленности**: отношение максимального собственного числа Гессиана к минимальному большое

Оптимизация: проблемы SGD

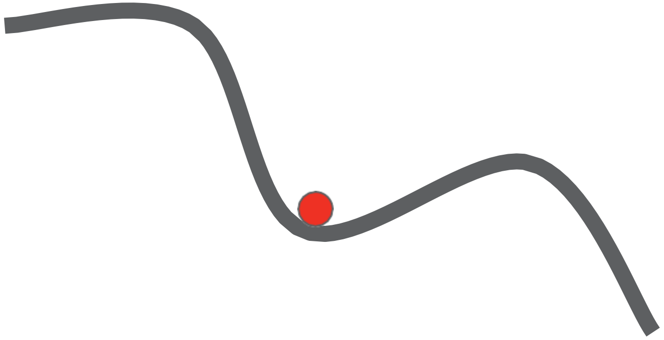
Что если попадем **локальный минимум** или **седловая точка**?

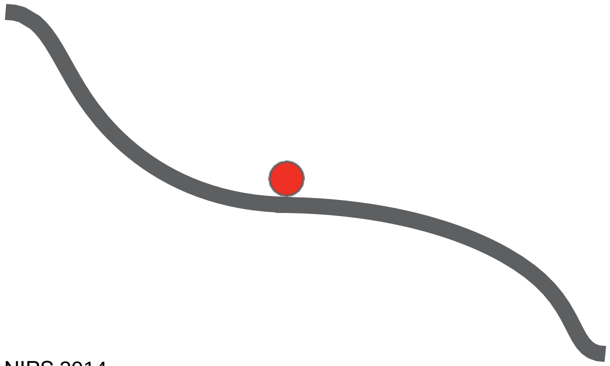


Оптимизация: проблемы SGD

Что если попадем **локальный минимум** или **седловая точка**?

Для высокой размерности седловые точки более вероятны





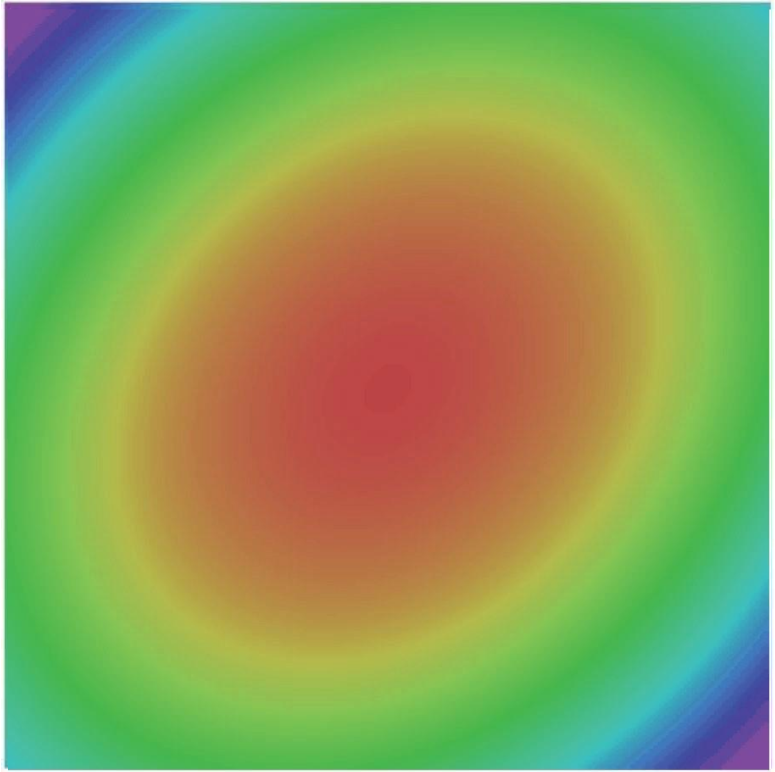
Dauphin et al, “Identifying and attacking the saddle point problem in high-dimensional non-convex optimization”, NIPS 2014

Оптимизация: проблемы SGD

Градиенты по минибатчам зашумлены!

$$L(W) = \frac{1}{N} \sum_{i=1}^{N} L_i(x_i, y_i, W)$$

$$\nabla_W L(W) = \frac{1}{N} \sum_{i=1}^{N} \nabla_W L_i(x_i, y_i, W)$$



SGD + Momentum

SGD

$$x_{t+1} = x_t - \alpha \nabla f(x_t)$$

```
while True:
  dx = compute_gradient(x)
  x -= learning_rate * dx
```

SGD+Momentum

$$v_{t+1} = \rho v_t + \nabla f(x_t)$$
$$x_{t+1} = x_t - \alpha v_{t+1}$$

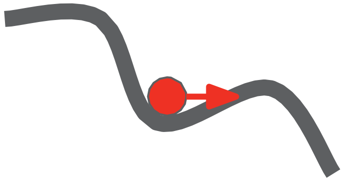
```
vx = 0
while True:
  dx = compute_gradient(x)
  vx = rho * vx + dx
  x -= learning_rate * vx
```

- Оценим “скорость” как скользящее среднее градиентов
- Rho задает “гибкость”; обычно, rho=0.9 или 0.99

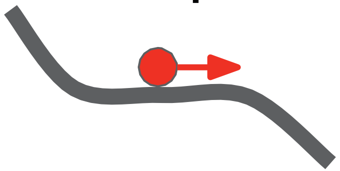
Sutskever et al, “On the importance of initialization and momentum in deep learning”, ICML 2013

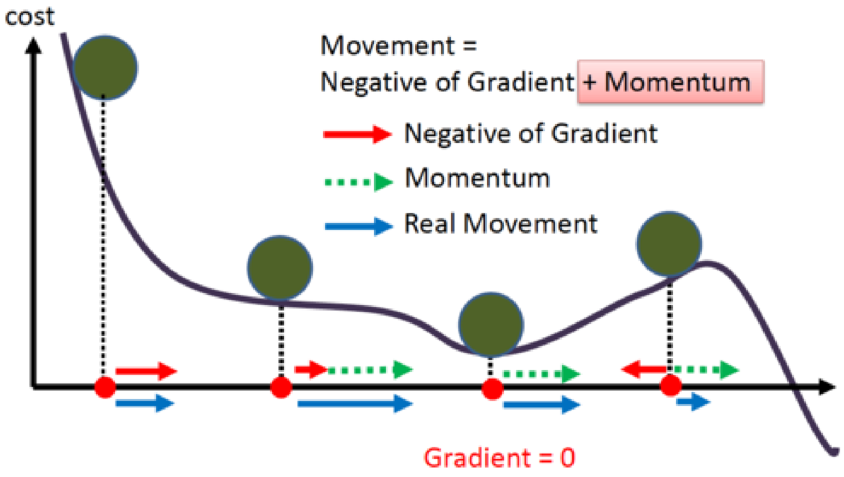
SGD + Momentum

Local Minima



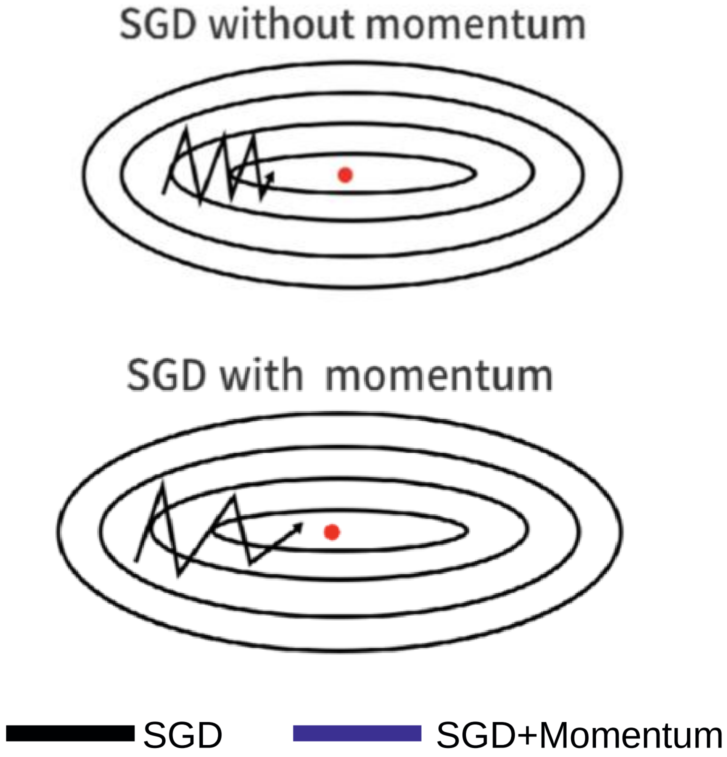
Saddle points





https://medium.com/analytics-vidhya/momentum-rmsprop-and-adam-optimizer-5769721b4b19

Gradient Noise



SGD + Momentum

SGD+Momentum

$$v_{t+1} = \rho v_t - \alpha \nabla f(x_t)$$
$$x_{t+1} = x_t + v_{t+1}$$

```
vx = 0
while True:
  dx = compute_gradient(x)
  vx = rho * vx - learning_rate * dx
  x += vx
```

SGD+Momentum

$$v_{t+1} = \rho v_t + \nabla f(x_t)$$
$$x_{t+1} = x_t - \alpha v_{t+1}$$

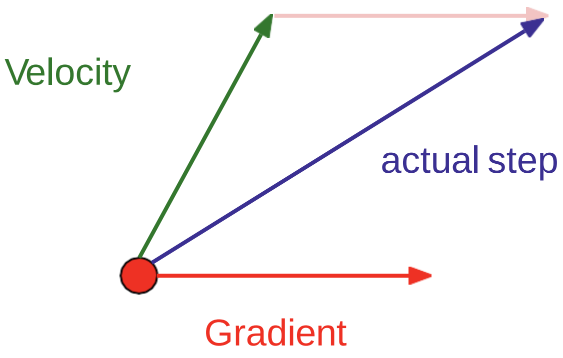
```
vx = 0
while True:
  dx = compute_gradient(x)
  vx = rho * vx + dx
  x -= learning_rate * vx
```

SGD+Momentum имеет несколько формулировок, но они приводят к одинаковой последовательности x

Sutskever et al, “On the importance of initialization and momentum in deep learning”, ICML 2013

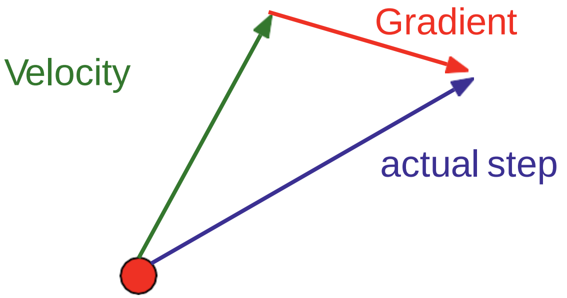
Nesterov Momentum

Momentum:



Комбинируем градиент в текущей точке со скоростью чтобы получить обновление весов

Nesterov Momentum



“Заглянем вперед” в точку, куда нас приведет «скорость»; оценим градиент там и скомбинируем новый градиент со скоростью, чтобы получить обновление весов

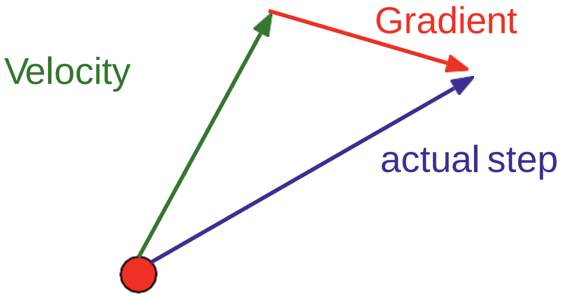
Nesterov, “A method of solving a convex programming problem with convergence rate O(1/k^2)”, 1983
Nesterov, “Introductory lectures on convex optimization: a basic course”, 2004
Sutskever et al, “On the importance of initialization and momentum in deep learning”, ICML 2013

Nesterov Momentum

$$v_{t+1} = \rho v_t - \alpha \nabla f(\boxed{x_t + \rho v_t})$$

$$x_{t+1} = x_t + v_{t+1}$$

Но нам нужно в терминах $x_t, \nabla f(x_t)$



Заменим переменную: $\tilde{x}_t = x_t + \rho v_t$

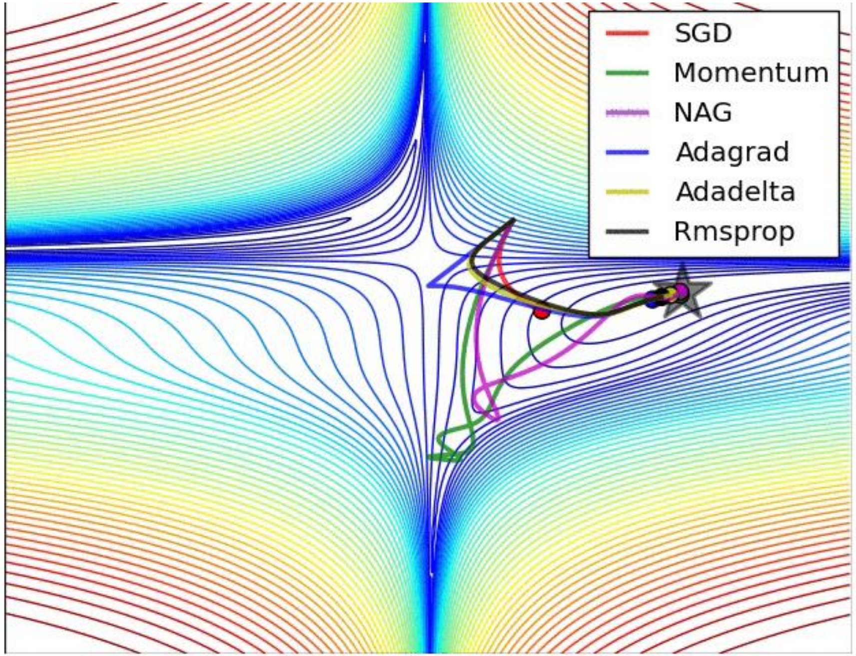
$$v_{t+1} = \rho v_t - \alpha \nabla f(\tilde{x}_t)$$

$$\begin{aligned} \tilde{x}_{t+1} &= \tilde{x}_t - \rho v_t + (1+\rho) v_{t+1} \\ &= \tilde{x}_t + v_{t+1} + \rho(v_{t+1} - v_t) \end{aligned}$$

“Заглянем вперед” в точку, куда нас приведет «скорость»; оценим градиент там и скомбинируем новый градиент со скоростью, чтобы получить обновление весов

https://cs231n.github.io/neural-networks-3/

Nesterov Momentum



https://wandb.ai/lavanyashukla/visualize-models/reports/Gradient-Descent-vs-Adagrad-vs-Momentum-in-TensorFlow--VmlldzoxOTg2MjM

AdaGrad – нормируем градиенты

```
grad_squared = 0
while True:
  dx = compute_gradient(x)
  grad_squared += dx * dx
  x -= learning_rate * dx / (np.sqrt(grad_squared) + 1e-7)
```

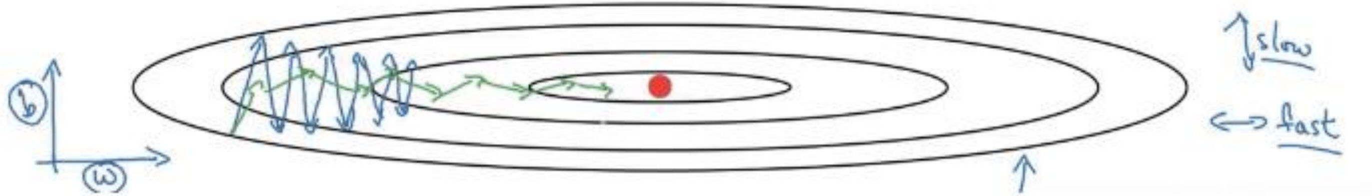
Добавим поэлементную нормировку на накопленную длину градиента по каждой координате

Получим “learning rate для каждого параметра” или “адаптивный learning rate”

Duchi et al, “Adaptive subgradient methods for online learning and stochastic optimization”, JMLR 2011

AdaGrad

```
grad_squared = 0
while True:
  dx = compute_gradient(x)
  grad_squared += dx * dx
  x -= learning_rate * dx / (np.sqrt(grad_squared) + 1e-7)
```

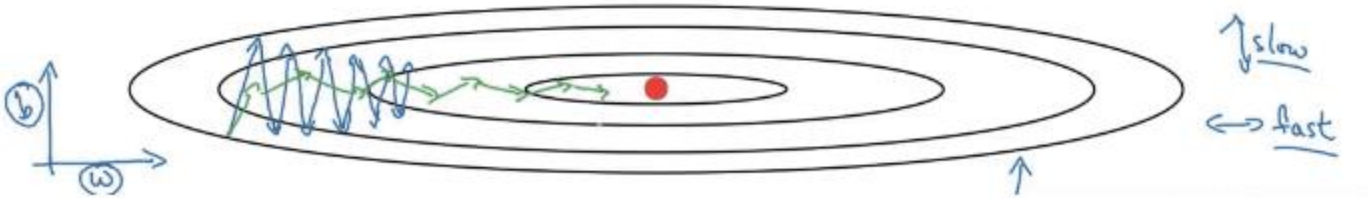


https://medium.com/analytics-vidhya/momentum-rmsprop-and-adam-optimizer-5769721b4b19

Q: Что нам это даст?

AdaGrad

```
grad_squared = 0
while True:
  dx = compute_gradient(x)
  grad_squared += dx * dx
  x -= learning_rate * dx / (np.sqrt(grad_squared) + 1e-7)
```



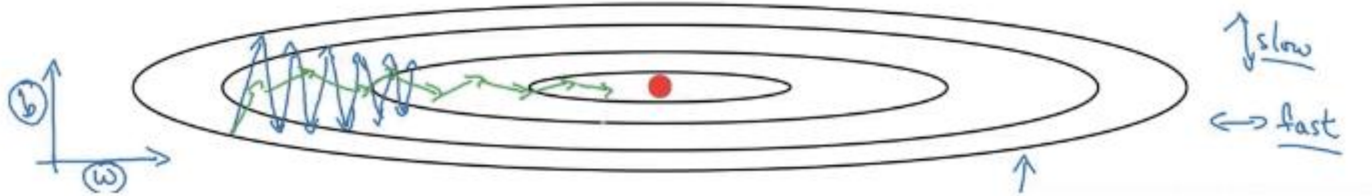
https://medium.com/analytics-vidhya/momentum-rmsprop-and-adam-optimizer-5769721b4b19

Q: Что нам это даст?

Спуск по “крутым” направлениям замедлится;
по “плоским” ускорится

AdaGrad

```
grad_squared = 0
while True:
  dx = compute_gradient(x)
  grad_squared += dx * dx
  x -= learning_rate * dx / (np.sqrt(grad_squared) + 1e-7)
```

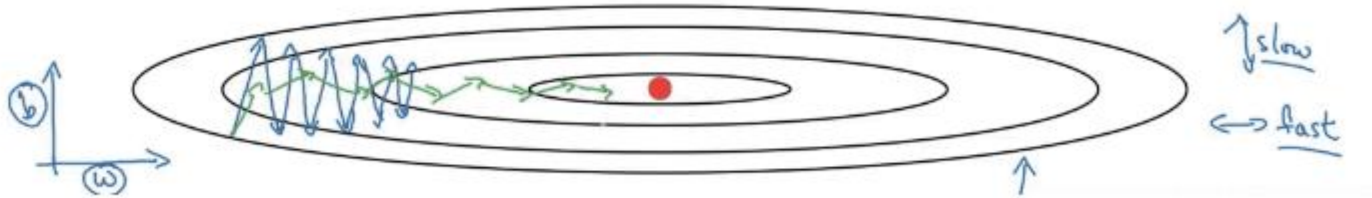


https://medium.com/analytics-vidhya/momentum-rmsprop-and-adam-optimizer-5769721b4b19

Q2: Что будет с размером шага с течением времени?

AdaGrad

```
grad_squared = 0
while True:
  dx = compute_gradient(x)
  grad_squared += dx * dx
  x -= learning_rate * dx / (np.sqrt(grad_squared) + 1e-7)
```



https://medium.com/analytics-vidhya/momentum-rmsprop-and-adam-optimizer-5769721b4b19

Q2: Что будет с размером шага с течением времени?

Затухнет до нуля….

RMSProp: “AdaGrad с утечкой”

AdaGrad

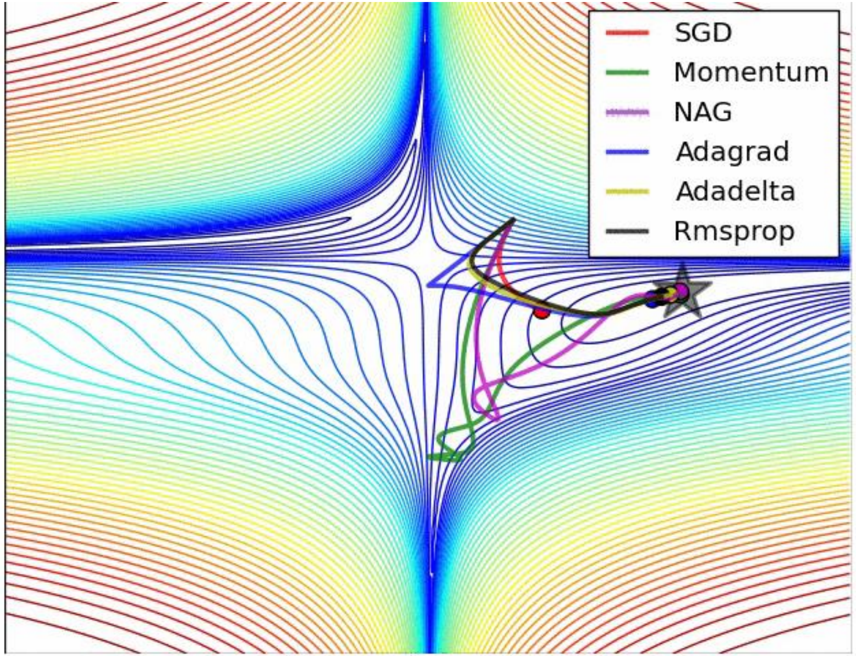
```
grad_squared = 0
while True:
  dx = compute_gradient(x)
  grad_squared += dx * dx
  x -= learning_rate * dx / (np.sqrt(grad_squared) + 1e-7)
```

RMSProp

```
grad_squared = 0
while True:
  dx = compute_gradient(x)
  grad_squared = decay_rate * grad_squared + (1 - decay_rate) * dx * dx
  x -= learning_rate * dx / (np.sqrt(grad_squared) + 1e-7)
```

Tieleman and Hinton, 2012

RMSProp



https://wandb.ai/lavanyashukla/visualize-models/reports/Gradient-Descent-vs-Adagrad-vs-Momentum-in-TensorFlow--VmlldzoxOTg2MjM

Adam (упрощенно)

```
first_moment = 0
second_moment = 0
while True:
  dx = compute_gradient(x)
  first_moment = beta1 * first_moment  + (1 - beta1) * dx
  second_moment = beta2 * second_moment + (1 - beta2) * dx * dx
  x -= learning_rate * first_moment / (np.sqrt(second_moment) + 1e-7))
```

Momentum

AdaGrad / RMSProp

Adam = RMSProp + momentum

Но с оценками моментов в начале будут проблемы…

Kingma and Ba, “Adam: A method for stochastic optimization”, ICLR 2015

Adam (полная версия)

```
first_moment = 0
second_moment = 0
for t in range(1, num_iterations):
  dx = compute_gradient(x)
  first_moment = beta1 * first_moment  + (1 - beta1) * dx
  second_moment = beta2 * second_moment + (1 - beta2) * dx * dx
  first_unbias = first_moment / (1 - beta1 ** t)
  second_unbias = second_moment / (1 - beta2 ** t)
  x -= learning_rate * first_unbias / (np.sqrt(second_unbias) + 1e-7))
```

Momentum

Bias correction

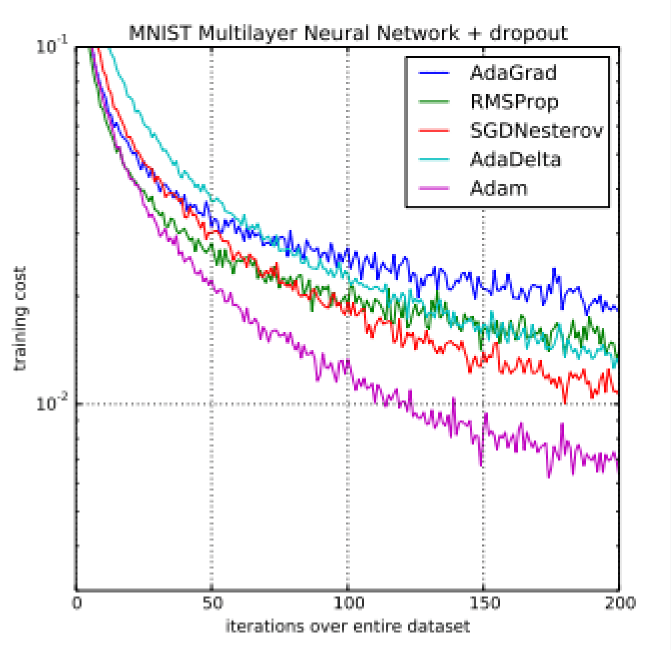
AdaGrad / RMSProp

Коррекция смещения с учетом того, что оценки first и second moment вначале будут нулевые

Kingma and Ba, “Adam: A method for stochastic optimization”, ICLR 2015

Adam с beta1 = 0.9, и beta2 = 0.999, и learning_rate = 1e-3 or 5e-4 выбор по умолчанию!

Adam

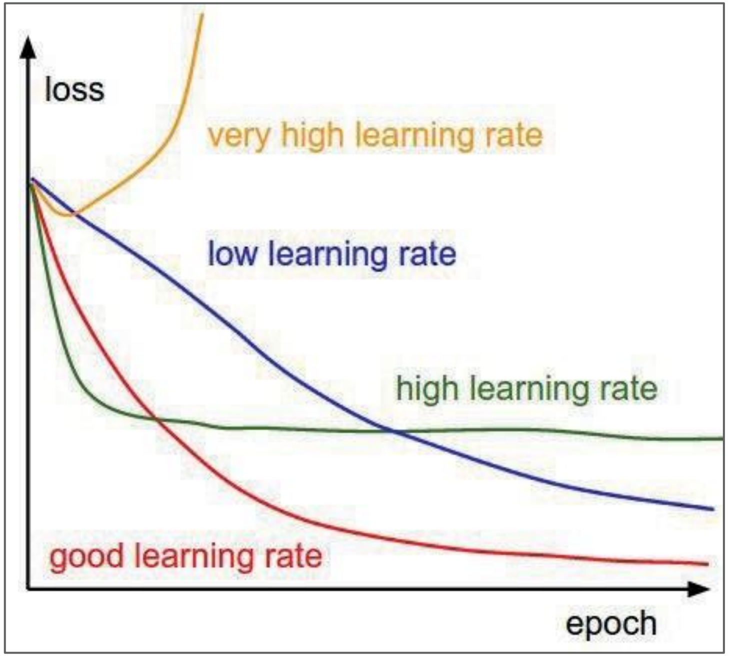


Comparison of Adam to Other Optimization Algorithms Training a Multilayer Perceptron

Kingma and Ba, “Adam: A method for stochastic optimization”, ICLR 2015

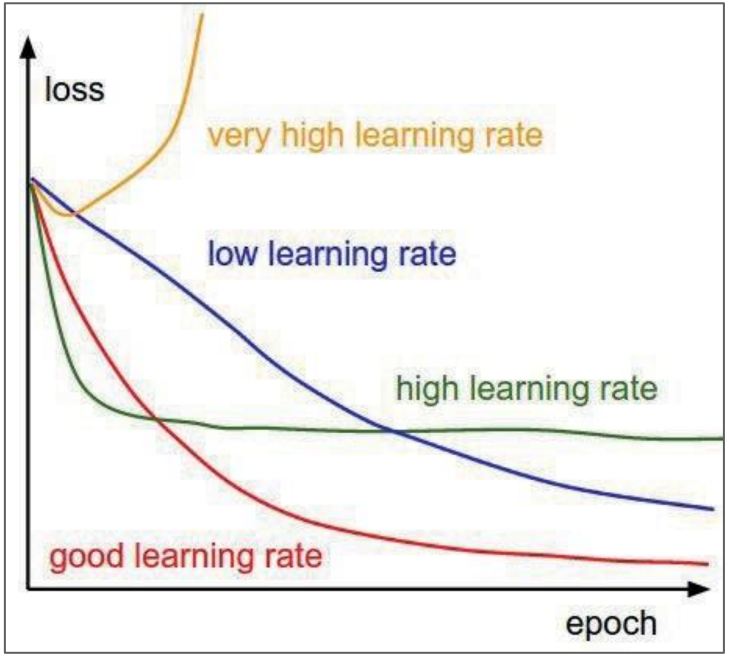
Настройка скорости обучения
(learning rate)

SGD, SGD+Momentum, Adagrad, RMSProp, Adam имеют гиперпараметр **learning rate**



Q: Какой из этих learning rates лучший?

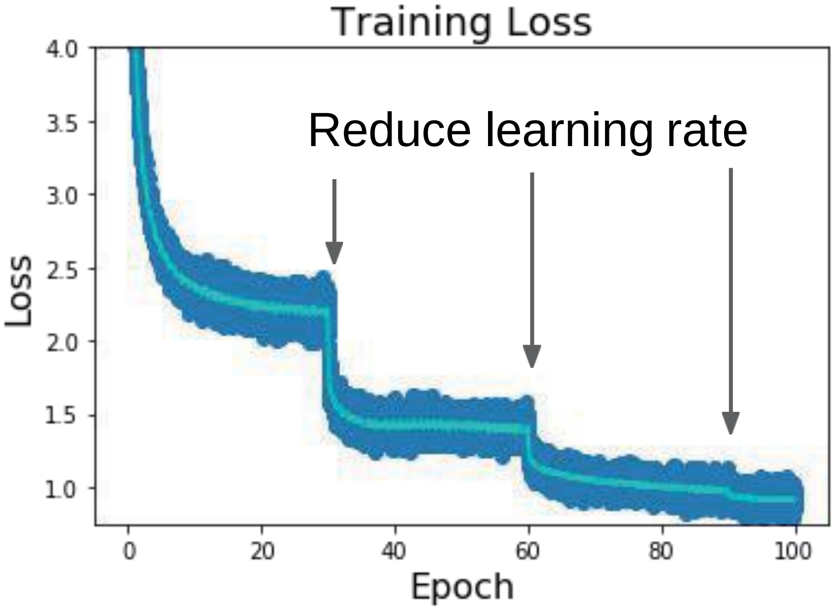
SGD, SGD+Momentum, Adagrad, RMSProp, Adam имеют гиперпараметр **learning rate**



Q: Какой из этих learning rates лучший?

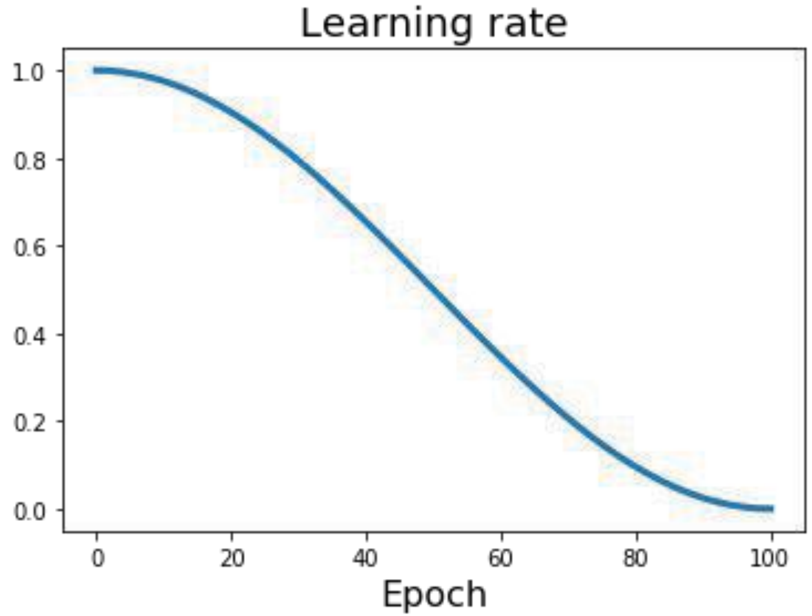
A: Начнем с большого, закончим маленьким!

Затухание скорости обучения



По шагам (step): Уменьшаем скорость обучения в нескольких фиксированных точках. Например, для ResNet, умножаем LR на 0.1 после эпох 30, 60 и 90.

Затухание скорости обучения



Step: Уменьшаем скорость обучения в нескольких фиксированных точках. Например, для ResNet, умножаем LR на 0.1 после эпох 30, 60 и 90.

Cosine: $$\alpha_t = \frac{1}{2}\alpha_0 \left(1 + \cos(t\pi/T)\right)$$

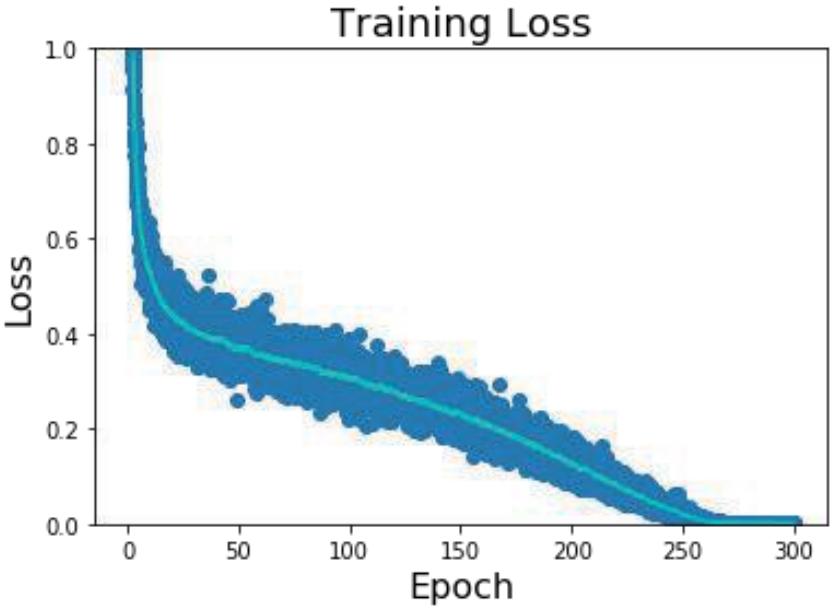
α_0 : Начальный learning rate

α_t : Learning rate в эпоху t

T : число эпох

Loshchilov and Hutter, “SGDR: Stochastic Gradient Descent with Warm Restarts”, ICLR 2017
Radford et al, “Improving Language Understanding by Generative Pre-Training”, 2018
Feichtenhofer et al, “SlowFast Networks for Video Recognition”, arXiv 2018
Child at al, “Generating Long Sequences with Sparse Transformers”, arXiv 2019

Затухание скорости обучения



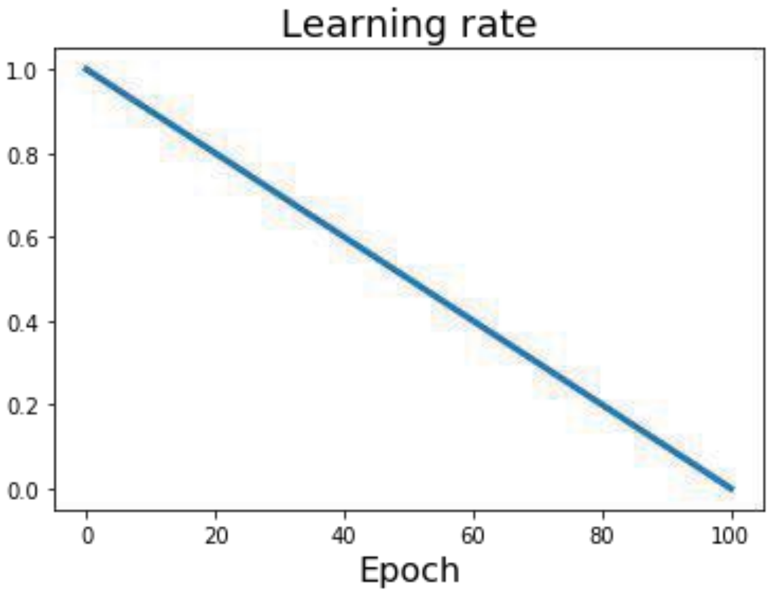
Step: Уменьшаем скорость обучения в нескольких фиксированных точках. Например, для ResNet, умножаем LR на 0.1 после эпох 30, 60 и 90.

Cosine: $$\alpha_t = \frac{1}{2}\alpha_0\left(1 + \cos(t\pi/T)\right)$$

α_0 : Начальный learning rate
α_t : Learning rate в эпоху t
T : число эпох

Loshchilov and Hutter, “SGDR: Stochastic Gradient Descent with Warm Restarts”, ICLR 2017
Radford et al, “Improving Language Understanding by Generative Pre-Training”, 2018
Feichtenhofer et al, “SlowFast Networks for Video Recognition”, arXiv 2018
Child at al, “Generating Long Sequences with Sparse Transformers”, arXiv 2019

Затухание скорости обучения



Step: Уменьшаем скорость обучения в нескольких фиксированных точках. Например, для ResNet, умножаем LR на 0.1 после эпох 30, 60 и 90.

Cosine: $$\alpha_t = \frac{1}{2}\alpha_0\left(1 + \cos(t\pi/T)\right)$$

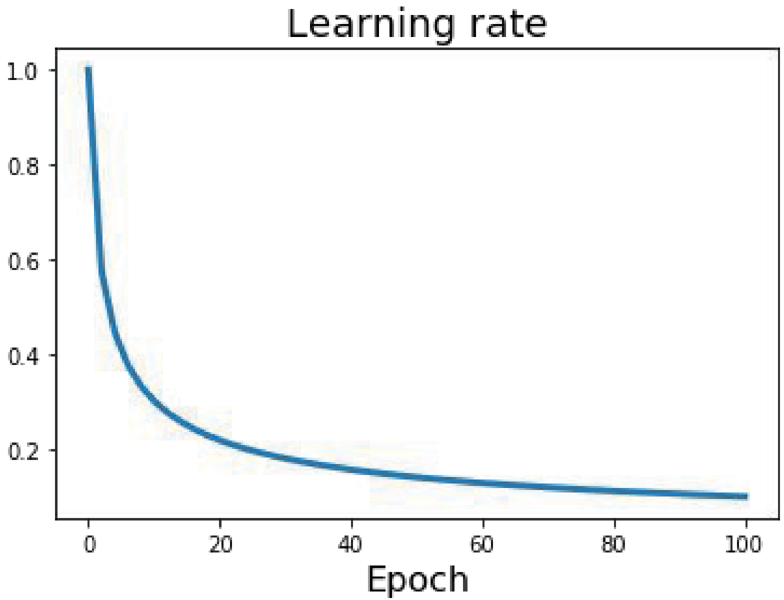
Linear: $$\alpha_t = \alpha_0(1 - t/T)$$

α_0 : Начальный learning rate
α_t : Learning rate в эпоху t
T : число эпох

Devlin et al, “BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding”, 2018

Свежие работы!

Затухание скорости обучения



Step: Уменьшаем скорость обучения в нескольких фиксированных точках. Например, для ResNet, умножаем LR на 0.1 после эпох 30, 60 и 90.

Cosine: $$\alpha_t = \frac{1}{2}\alpha_0\left(1 + \cos(t\pi/T)\right)$$

Linear: $$\alpha_t = \alpha_0(1 - t/T)$$

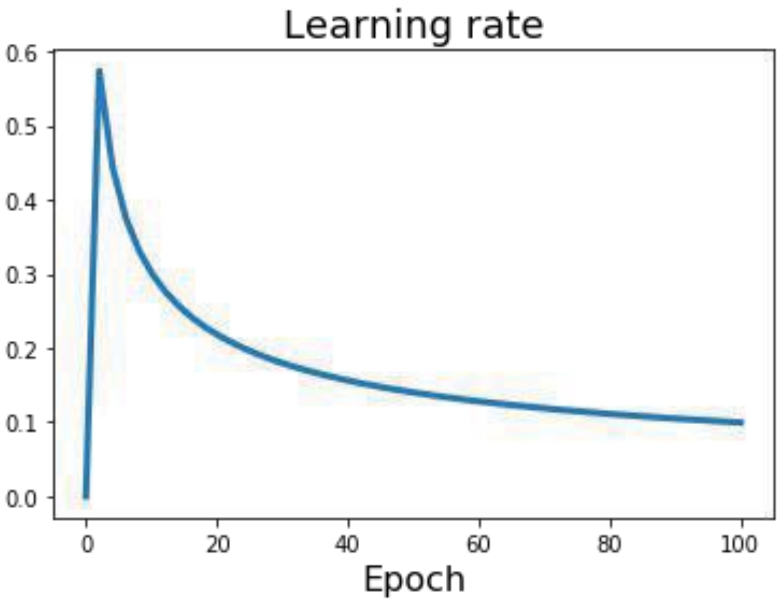
Inverse sqrt: $$\alpha_t = \alpha_0/\sqrt{t}$$

α_0 : Начальный learning rate
α_t : Learning rate в эпоху t
T : число эпох

Vaswani et al, “Attention is all you need”, NIPS 2017

Свежие работы!

Затухание скорости обучения: Линейный прогрев (Linear Warmup)



Высокий начальный LR может привести к взрывному росту ошибки. Линейный рост начиная с 0 за первые ~5000 итераций может предотвратить это.

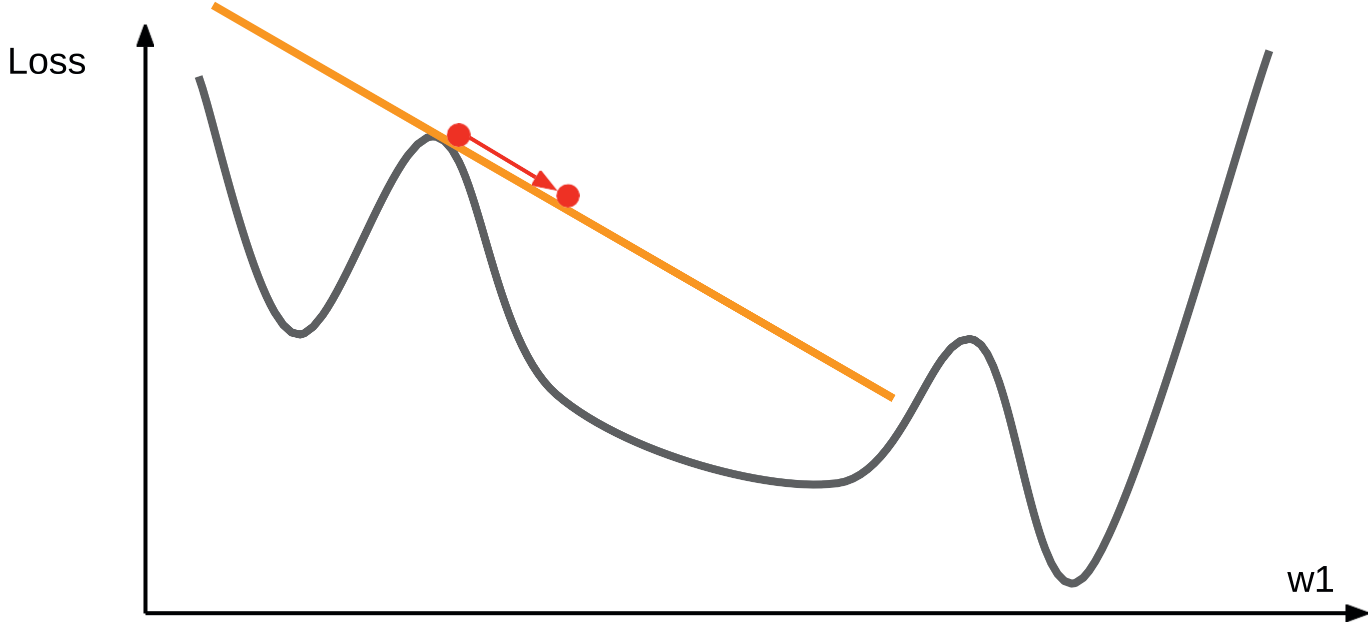
Эмпирическое правило: увеличивая размер батча в N раз, уменьшайте LR в N раз.

Goyal et al, “Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour”, arXiv 2017

Очень полезный подход на практике!

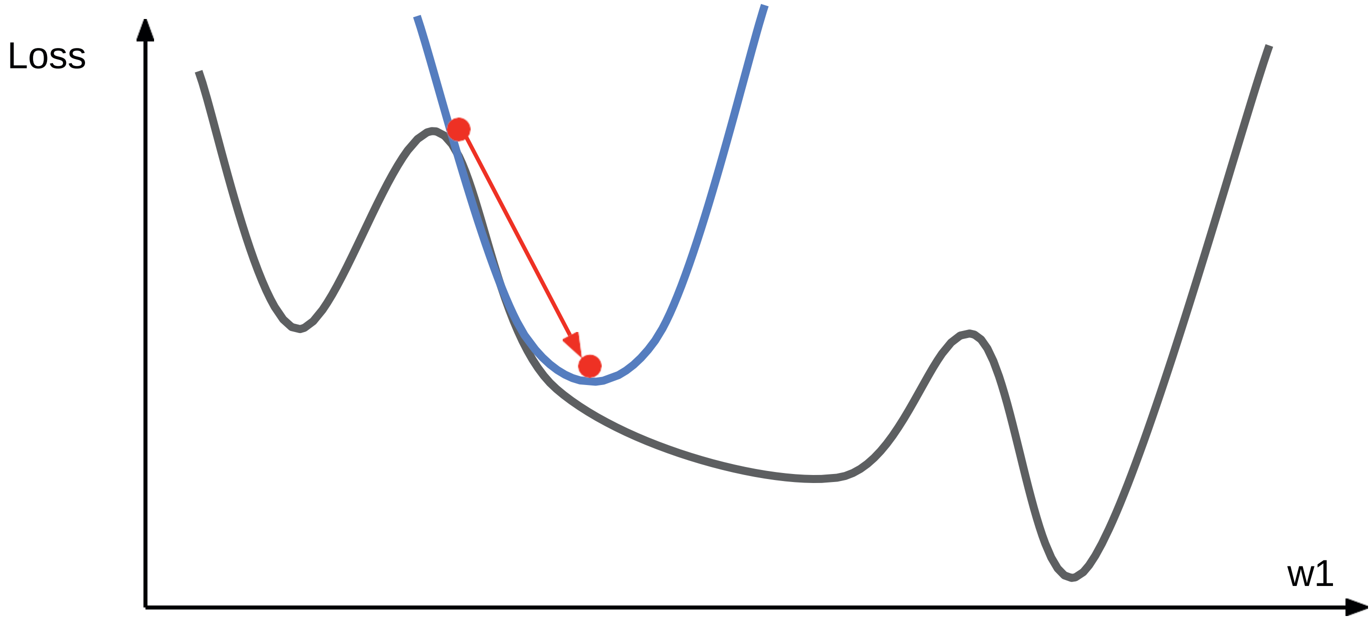
Оптимизация первого порядка и ее улучшение

(1) Локальная линейная аппроксимация на основе градиента
(2) Шаг в направлении минимизации



Оптимизация второго порядка

(1) На основе **Гессиана** строим **квадратичную** аппроксимацию
(2) Шагаем в направлении минимума этой аппроксимации



Оптимизация второго порядка

Ряд Тейлора второго порядка:

$$J(\boldsymbol{\theta}) \approx J(\boldsymbol{\theta}_0) + (\boldsymbol{\theta} - \boldsymbol{\theta}_0)^\top \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}_0) + \frac{1}{2}(\boldsymbol{\theta} - \boldsymbol{\theta}_0)^\top \boldsymbol{H}(\boldsymbol{\theta} - \boldsymbol{\theta}_0)$$

Решаем обратную задачу для метода Ньютона:

$$\boldsymbol{\theta}^* = \boldsymbol{\theta}_0 - \boldsymbol{H}^{-1} \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}_0)$$

Q: Можем ли использовать для глубокого обучения?

Оптимизация второго порядка

Ряд Тейлора второго порядка:

$$J(\boldsymbol{\theta}) \approx J(\boldsymbol{\theta}_0) + (\boldsymbol{\theta} - \boldsymbol{\theta}_0)^\top \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}_0) + \frac{1}{2}(\boldsymbol{\theta} - \boldsymbol{\theta}_0)^\top \boldsymbol{H}(\boldsymbol{\theta} - \boldsymbol{\theta}_0)$$

Решаем обратную задачу для метода Ньютона:

$$\boldsymbol{\theta}^* = \boldsymbol{\theta}_0 - \boldsymbol{H}^{-1} \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}_0)$$

В Гессиане N^2 элементов
Обращение - O(N^3)
N = (Десятки или сотни) миллионов

Q: Можем ли использовать для глубокого обучения?

Оптимизация второго порядка

$$\boldsymbol{\theta}^* = \boldsymbol{\theta}_0 - \boldsymbol{H}^{-1} \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}_0)$$

- Квазиньютоновские методы (**BFGS** самый популярный): *вместо обращения Гессиана (O(n^3)), аппроксимируют обратный Гессиан шагами со сложностью O(n^2).*

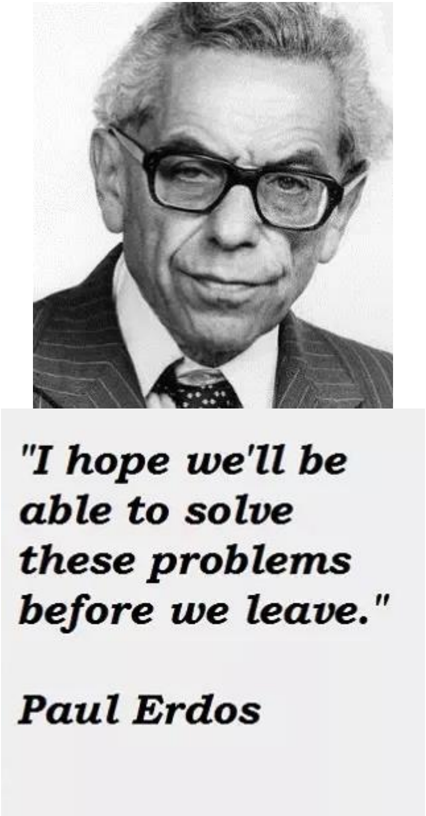
- **L-BFGS** (Limited memory BFGS):
  Не хранит Гессиан целиком.
  <u>Аппроксимация Гессиана полезна при дистиляции или контрастировании сети</u>

L-BFGS

- **хорошо работает с полным набором данных, в детерменированном смысле (без SGD!)**

  т.е. если у вас есть детерминированная f(x) то L-BFGS хорошее решение

- **Плохо работает в парадигме минибатчей**. Дает плохую сходимость. Адаптация методов второго порядка к стохастической парадигме - направление активных исследований.



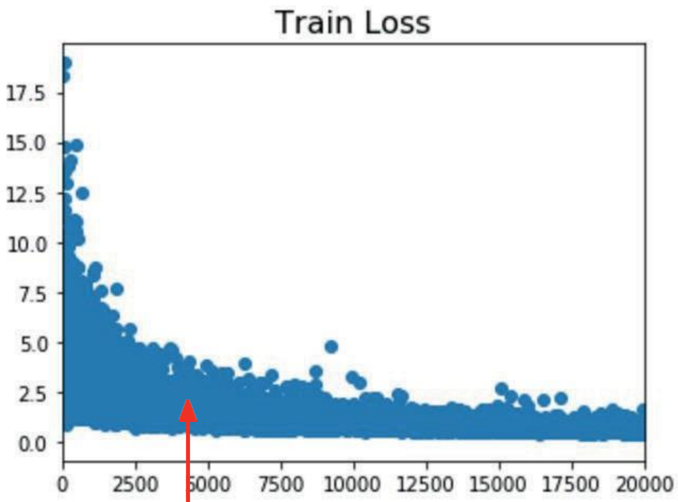
Le et al, "On optimization methods for deep learning, ICML 2011"
Ba et al, "Distributed second-order optimization using Kronecker-factored approximations", ICLR 2017

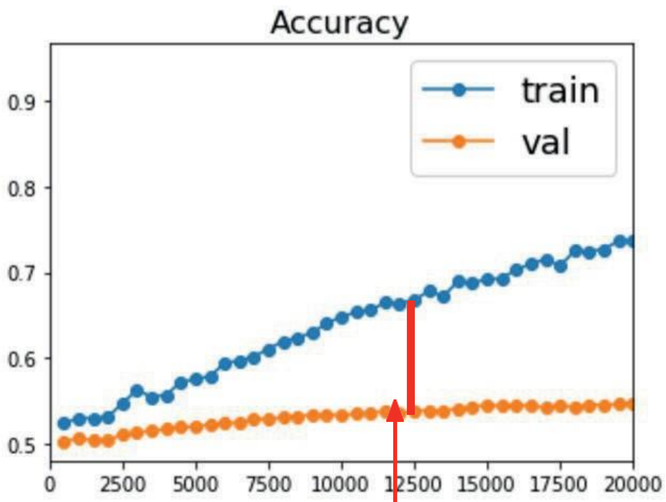
На практие:

- **Adam** хороший выбор для большинства случаев; часто работает хорошо с константным LR
- **SGD+Momentum** может дать лучший результат чем Adam, но нужно уменьшать LR
  - Косинусное уменьшение LR хороший выбор.
  - Warm Start – тоже очень хорош!
- На методы второго порядка можно посмотреть, например на **L-BFGS** (но надо убрать весь шум!)

Уменьшение ошибки на тесте

По ту сторону loss

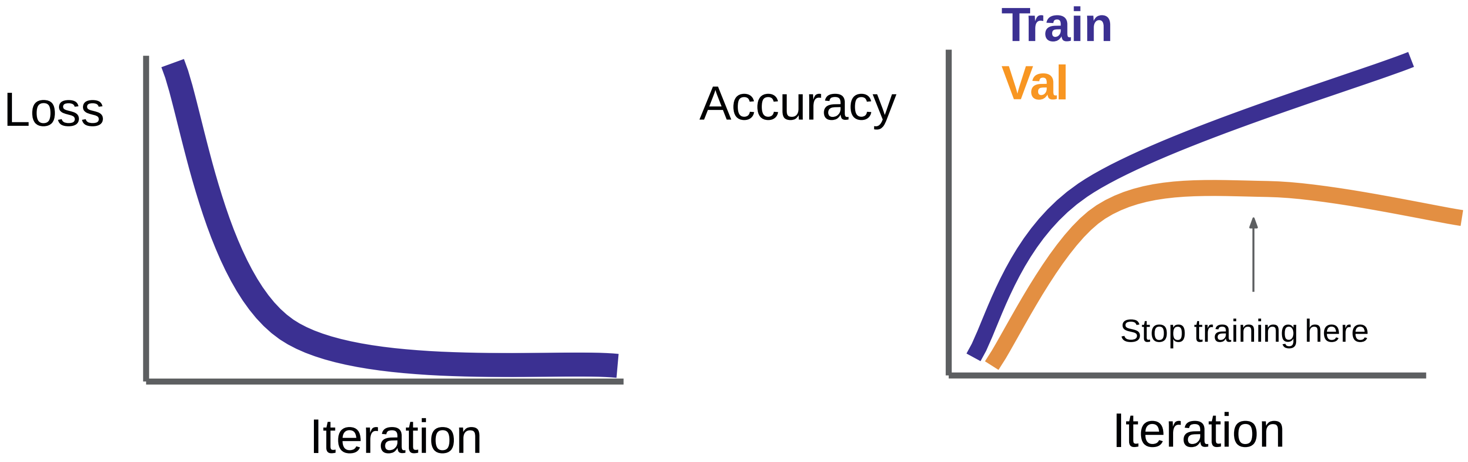




Улучшая архитектуру и оптимизаторы мы уменьшаем ошибку времени обучения (loss)

Но реально нам важна ошибка предсказания на новых данных – как сократить отрыв train и val?

Early Stopping: делайте это!



Останавливайте обучение когда точность на валидационном наборе уменьшается
Или обучайтесь до конца, но сохраняйте модели каждой эпохи, чтобы выбрать лучшую по валидационному набору

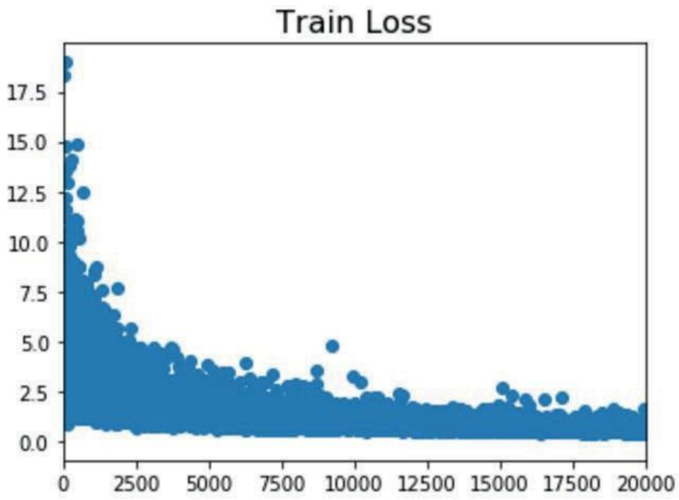
Ансамбли моделей/ Model ensembles

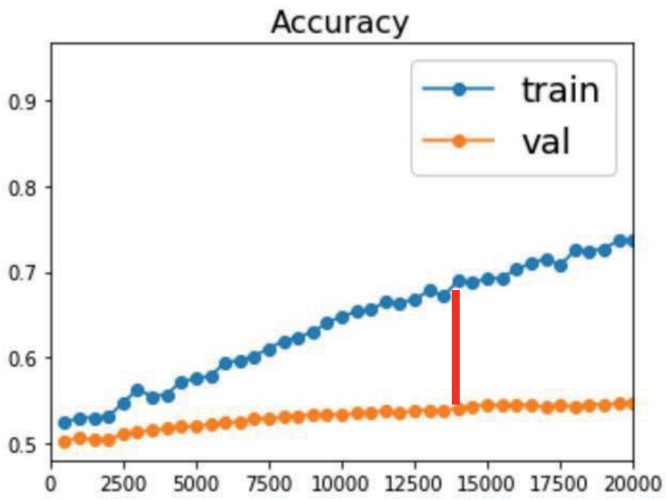
1. Обучаем независимые модели
2. На тесте усредняем их прогнозы

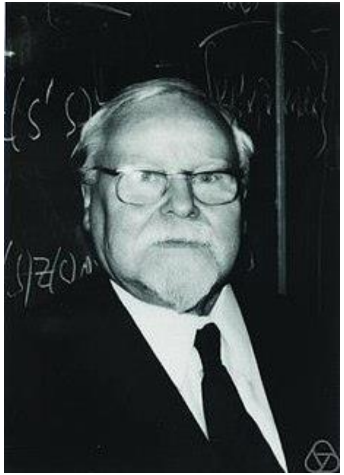
   (Берем среднее прогнозных распределений, берем argmax)

Получаем прирост в 2%

Как увеличить точность конкретной модели?







А.Н. Тихонов
Основатель
ВМК МГУ

Регуляризация!

Регуляризация: дополнительное слагаемое к ошибке

$$L = \frac{1}{N}\sum_{i=1}^{N}\sum_{j\neq y_i}\max(0, f(x_i;W)_j - f(x_i;W)_{y_i} + 1) + \boxed{\lambda R(W)}$$

Обычно:

L2 регуляризация $R(W) = \sum_k \sum_l W_{k,l}^2$ (Weight decay) Затухание весов

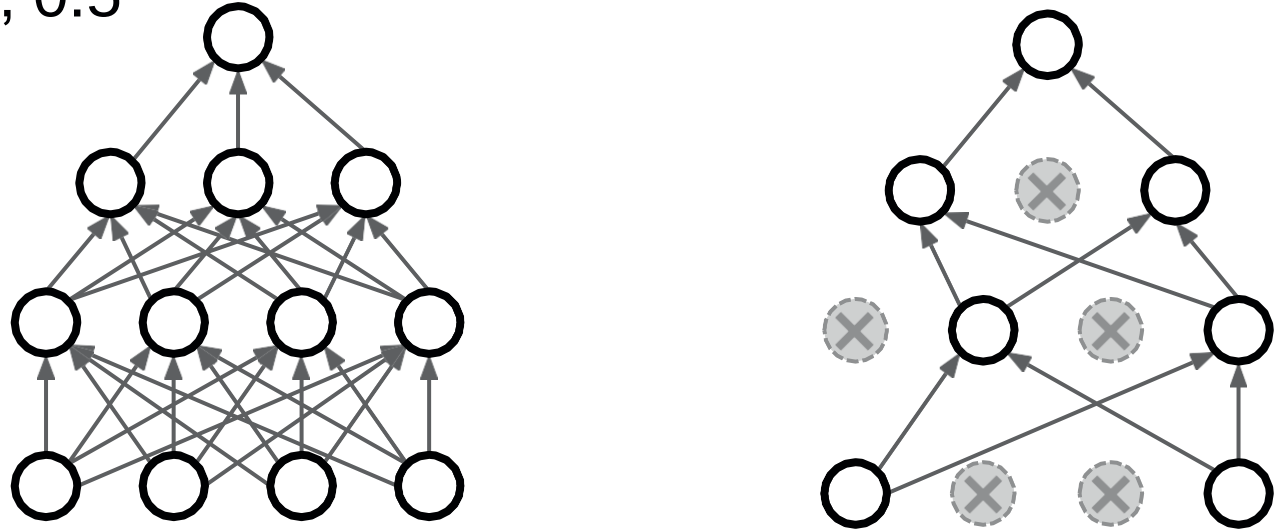
L1 регуляризация $R(W) = \sum_k \sum_l |W_{k,l}|$

Elastic net (L1 + L2) $R(W) = \sum_k \sum_l \beta W_{k,l}^2 + |W_{k,l}|$

Регуляризация: Dropout

В каждом прямом проходе зануляем часть нейронов

Вероятность зануления - гиперпараметр; обычно, 0.5

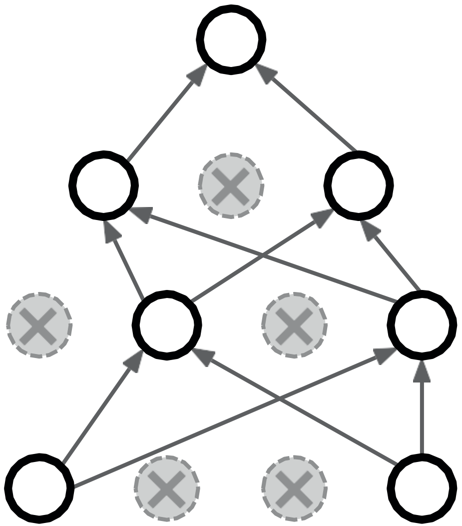


Srivastava et al, “Dropout: A simple way to prevent neural networks from overfitting”, JMLR 2014

Регуляризация: Dropout

Пример dropout в трехслойной сети

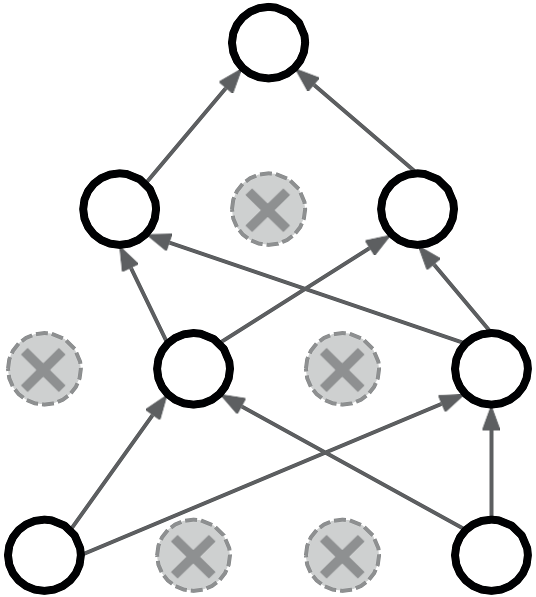
```
p = 0.5 # probability of keeping a unit active. higher = less dropout

def train_step(X):
  """ X contains the data """

  # forward pass for example 3-layer neural network
  H1 = np.maximum(0, np.dot(W1, X) + b1)
  U1 = np.random.rand(*H1.shape) < p # first dropout mask
  H1 *= U1 # drop!
  H2 = np.maximum(0, np.dot(W2, H1) + b2)
  U2 = np.random.rand(*H2.shape) < p # second dropout mask
  H2 *= U2 # drop!
  out = np.dot(W3, H2) + b3

  # backward pass: compute gradients... (not shown)
  # perform parameter update... (not shown)
```

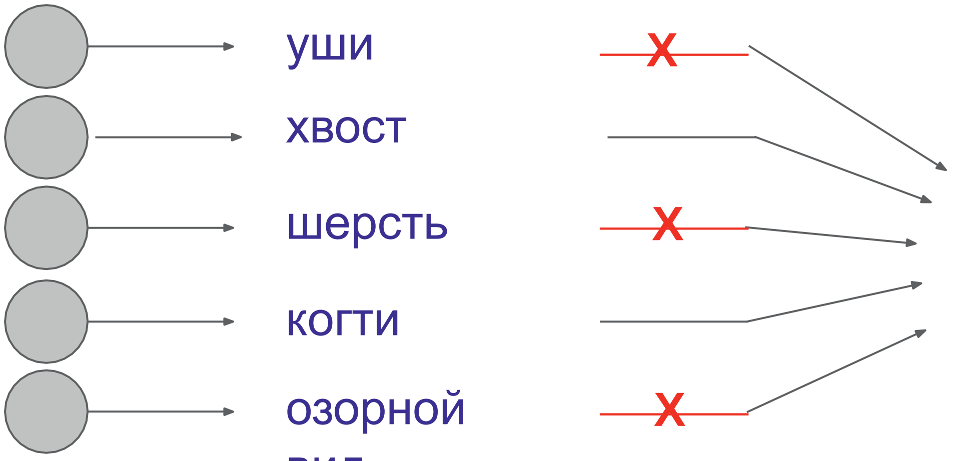


Регуляризация: Dropout

Почему это работает?

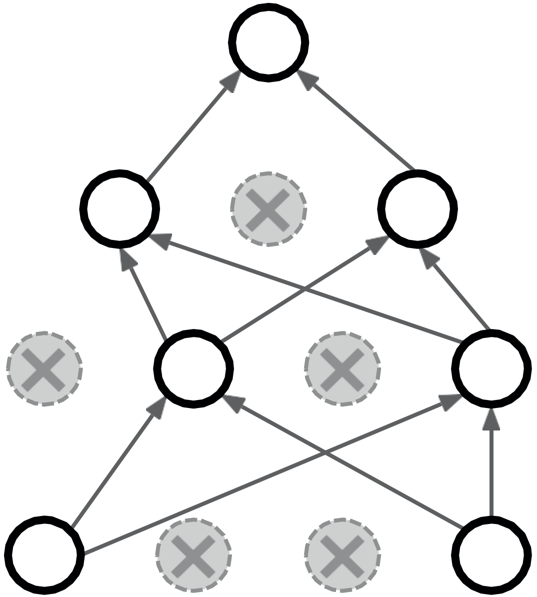
Заставляет сеть формировать устойчивое представление.
Предотвращает ко-адаптацию признаков:





Регуляризация: Dropout

Почему это работает?



Альтернативное обоснование:

Dropout реализует **ансамбль** моделей с общими параметрами.

Каждая бинарная маска dropout – одна модель

FC слой с 4096 весами дает
$2^{4096} \sim 10^{1233}$ возможных масок!
Во Вселенной всего ~ 10^{82} атомов...

Dropout: Тест/инференс

Dropout делает выход случайным!

Output (label)

Input (image)

Random mask

$$\boxed{y} = f_W(\boxed{x}, \boxed{z})$$

Давайте “усредним” случайность при тесте

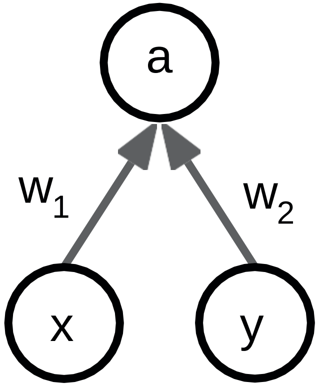
$$y = f(x) = E_z\big[f(x, z)\big] = \int p(z) f(x, z) dz$$

Но это сложный интеграл …

Dropout: тест/инференс

Аппроксимируем интеграл

$$y = f(x) = E_z\left[f(x,z)\right] = \int p(z)f(x,z)dz$$



Рассмотрим один нейрон.

Во время теста: $E\left[a\right] = w_1x + w_2y$

Во время обучения:

$$\begin{aligned}E\left[a\right] =& \frac{1}{4}(w_1x + w_2y) + \frac{1}{4}(w_1x + 0y) \\ &+ \frac{1}{4}(0x + 0y) + \frac{1}{4}(0x + w_2y) \\ =& \frac{1}{2}(w_1x + w_2y)\end{aligned}$$

Во время теста, просто **умножим** на вероятность dropout!

Dropout: тест/инференс

```
def predict(X):
  # ensembled forward pass
  H1 = np.maximum(0, np.dot(W1, X) + b1) * p # NOTE: scale the activations
  H2 = np.maximum(0, np.dot(W2, H1) + b2) * p # NOTE: scale the activations
  out = np.dot(W3, H2) + b3
```

Во время теста/инференса все нейроны активны
=> мы должны взвесить все активации нейронов так, чтобы:

<u>выход во время теста</u> = <u>ожидаемый выход во время обучения</u>

Dropout: Итоги

```python
""" Vanilla Dropout: Not recommended implementation (see notes below) """

p = 0.5 # probability of keeping a unit active. higher = less dropout

def train_step(X):
  """ X contains the data """

  # forward pass for example 3-layer neural network
  H1 = np.maximum(0, np.dot(W1, X) + b1)
  U1 = np.random.rand(*H1.shape) < p # first dropout mask
  H1 *= U1 # drop!
  H2 = np.maximum(0, np.dot(W2, H1) + b2)
  U2 = np.random.rand(*H2.shape) < p # second dropout mask
  H2 *= U2 # drop!
  out = np.dot(W3, H2) + b3

  # backward pass: compute gradients... (not shown)
  # perform parameter update... (not shown)

def predict(X):
  # ensembled forward pass
  H1 = np.maximum(0, np.dot(W1, X) + b1) * p # NOTE: scale the activations
  H2 = np.maximum(0, np.dot(W2, H1) + b2) * p # NOTE: scale the activations
  out = np.dot(W3, H2) + b3
```

Во время обучения: занулим с вероятностью *p*

Во время теста: взвесим с вероятностью *p*

Более удобно: “Inverted dropout”

```
p = 0.5 # probability of keeping a unit active. higher = less dropout

def train_step(X):
  # forward pass for example 3-layer neural network
  H1 = np.maximum(0, np.dot(W1, X) + b1)
  U1 = (np.random.rand(*H1.shape) < p) / p # first dropout mask. Notice /p!
  H1 *= U1 # drop!
  H2 = np.maximum(0, np.dot(W2, H1) + b2)
  U2 = (np.random.rand(*H2.shape) < p) / p # second dropout mask. Notice /p!
  H2 *= U2 # drop!
  out = np.dot(W3, H2) + b3

  # backward pass: compute gradients... (not shown)
  # perform parameter update... (not shown)

def predict(X):
  # ensembled forward pass
  H1 = np.maximum(0, np.dot(W1, X) + b1) # no scaling necessary
  H2 = np.maximum(0, np.dot(W2, H1) + b2)
  out = np.dot(W3, H2) + b3
```

Во время теста ничего не меняем!

Регуляризация: Общий подход

Training: Добавим случайностей различной природы

$$y = f_W(x, z)$$

Testing: Усредним случайности (иногда аппроксимируем)

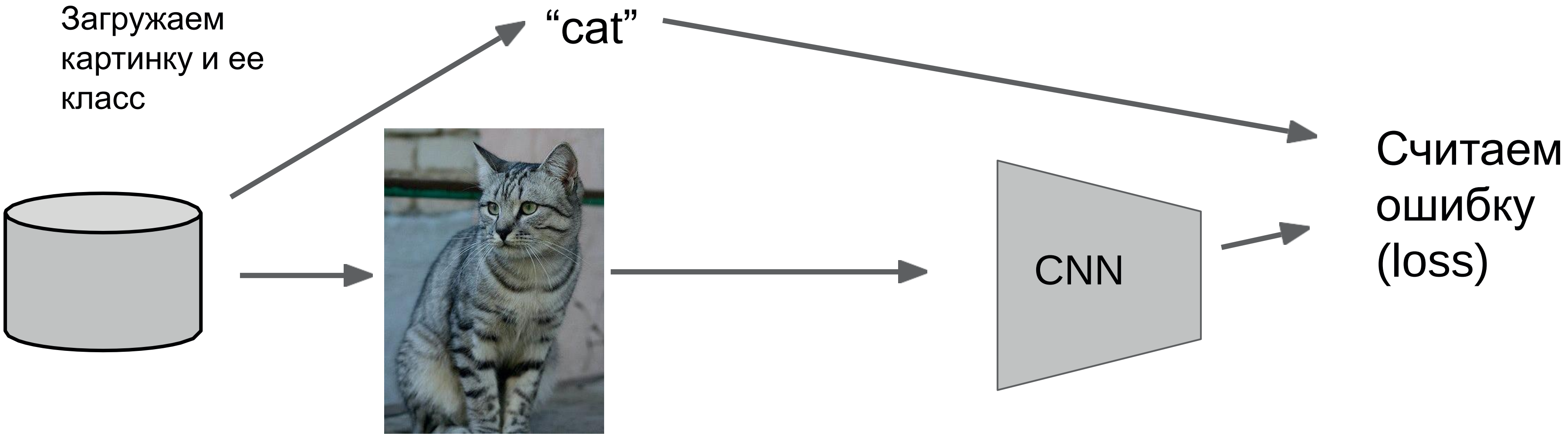
$$y = f(x) = E_z\big[f(x, z)\big] = \int p(z) f(x, z) dz$$

Например: Batch Normalization

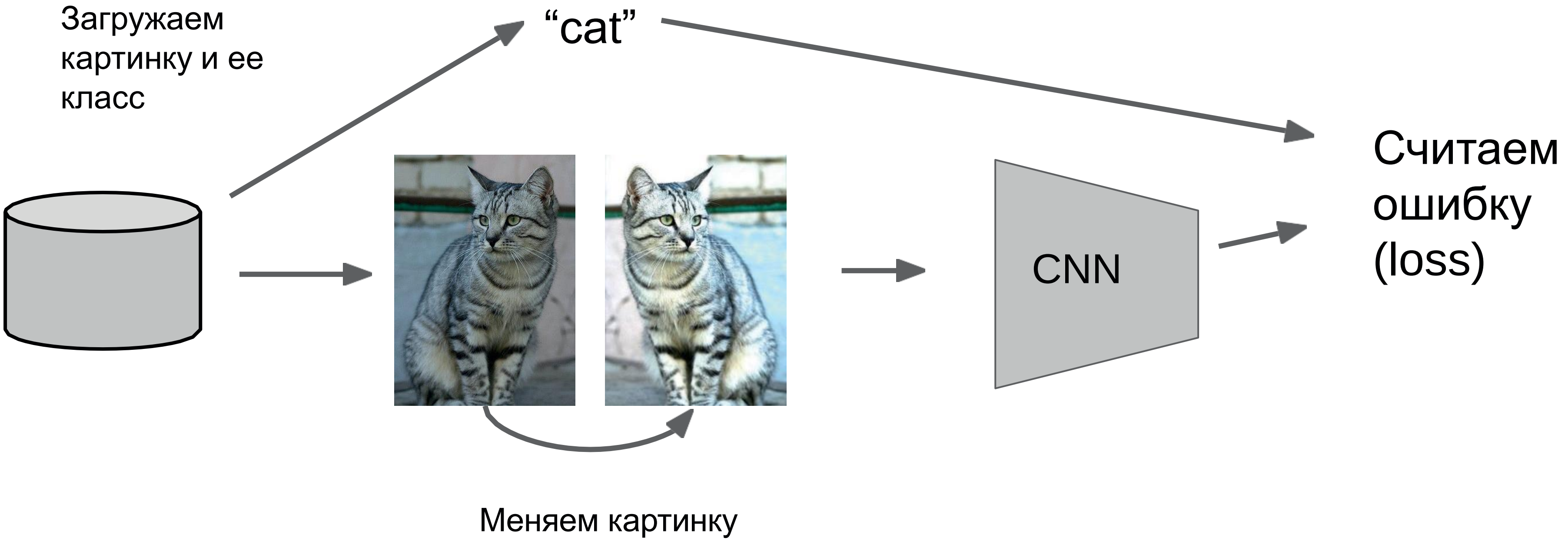
Training: Нормализуем с учетом статистик минибатчей

Testing: нормализуем с использованием фиксированных статистик

Регуляризация: Аугментация данных



Регуляризация: Аугментация данных



Аугментация данных
Горизонтальные отражения

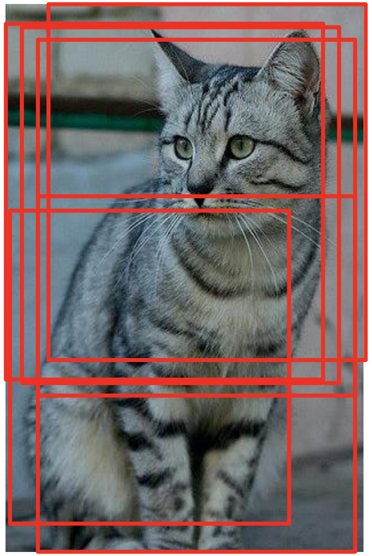
Аугментация данных

Случайное изменение размера и масштабирование

Training: набираем случайную обрезку / масштабы

Например, для ResNet:

1. Выбираем рандомно L из диапазона [256, 480]
2. Меняем размер изображения из обучающей выборки, L по короткой стороне
3. Выбираем случайный фрагмент 224 x 224

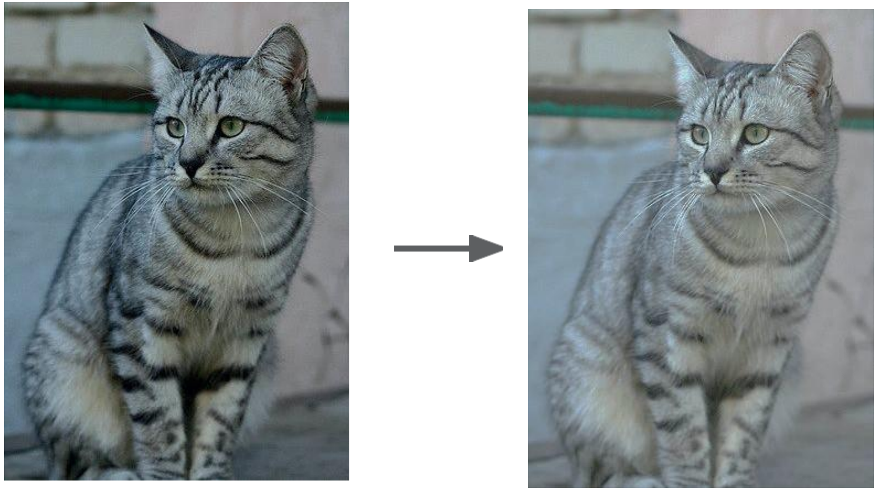


Training: Узнаем чуть позже!

Аугментация данных

Рандомизация цвета

Простая: Рандомизируем яркость и контраст



Более сложно:

1. Применяем PCA к [R, G, B] цветам пикселов
2. Рандомизируем “цветовой сдвиг” по главным компонентам
3. Добавляем сдвиг ко всем цветам пикселов

(As seen in *[Krizhevsky et al. 2012],* ResNet, etc)

Аугментация данных

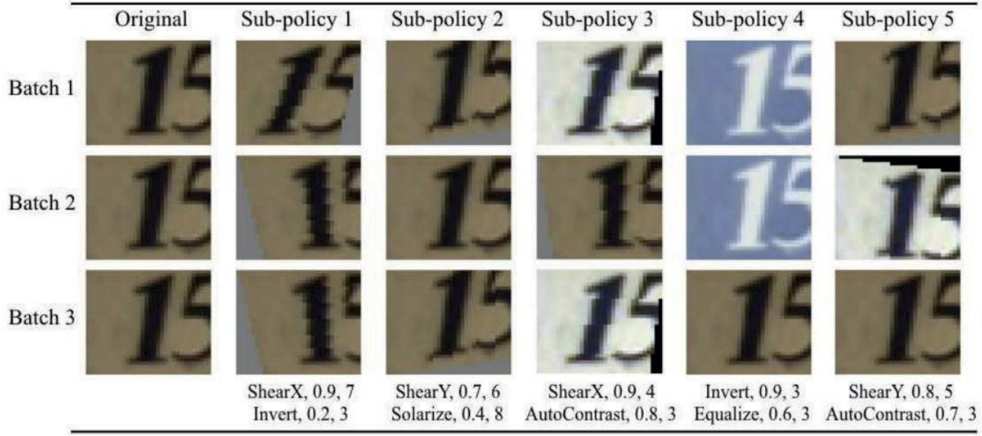
Проявим творчество!

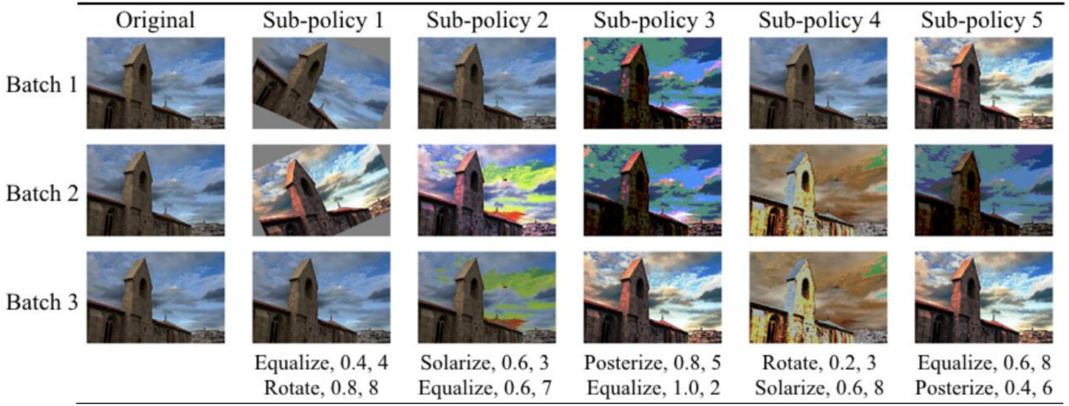
Рандомизируем комбинацию:

- сдвигов
- поворотов
- растяжений
- срезов,
- дисторсий оптики, ISO шумы…

Библиотека abluminations
https://github.com/albumentations-team/albumentations

Автоматическая аугментация данных





Cubuk et al., “AutoAugment: Learning Augmentation Strategies from Data”, CVPR 2019

Регуляризация: Маргинализация

Training: Добавим случайного шума и/или искажений

Testing: Маргинализируем по шуму/искажениям

Примеры:

Dropout

Batch Normalization Augmentation

Маргинализация

Пример маргинализации:
Для UK P(happiness|weather) = P(happiness, country=England | weather) + P(happiness, country=Scotland | weather) + P(happiness, country=Wales | weather), т.к. UK состоит из England, Scotland, Wales

Подробнее про маргинализацию, например:
https://towardsdatascience.com/probability-concepts-explained-marginalisation-2296846344fc

Название «частное распределение» используется в переводах под редакцией Колмогорова, «маргинальное распределение» — в более современной литературе путём заимствования из английского языка (англ. *marginal distribution*); название в английском языке в свою очередь является переводом с немецкого (нем. *Randverteilungen*) из публикации Колмогорова: A. Kolmogoroff, 1933

Википедия, Частное распределение

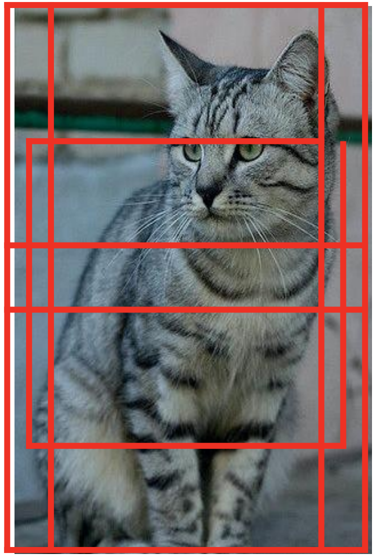
Маргинализация для Dropout:

$$E[a] = w_1 x + w_2 y$$

$$\begin{aligned} E[a] &= \frac{1}{4}(w_1 x + w_2 y) + \frac{1}{4}(w_1 x + 0y) \\ &+ \frac{1}{4}(0x + 0y) + \frac{1}{4}(0x + w_2 y) \\ &= \frac{1}{2}(w_1 x + w_2 y) \end{aligned}$$

Аугментация данных

Случайное изменение размера и масштабирование



Training: набираем случайную обрезку / масштабы

Например, для ResNet:

1. Выбираем рандомно L из диапазона [256, 480]
2. Меняем размер изображения из обучающей выборки, L по короткой стороне
3. Выбираем случайный фрагмент 224 x 224

Testing: усредняем по фиксированному набору вырезов (crop) – **Маргинализация**

ResNet:

1. Масштабируем на 5 размеров: {224, 256, 384, 480, 640}
2. Для каждого размера, берем 10 224 x 224 выреза: 4 по углам + центр, + отражения

Регуляризация: DropConnect

Training: Случайно зануляем веса связей между нейронами

Testing: Используем все связи

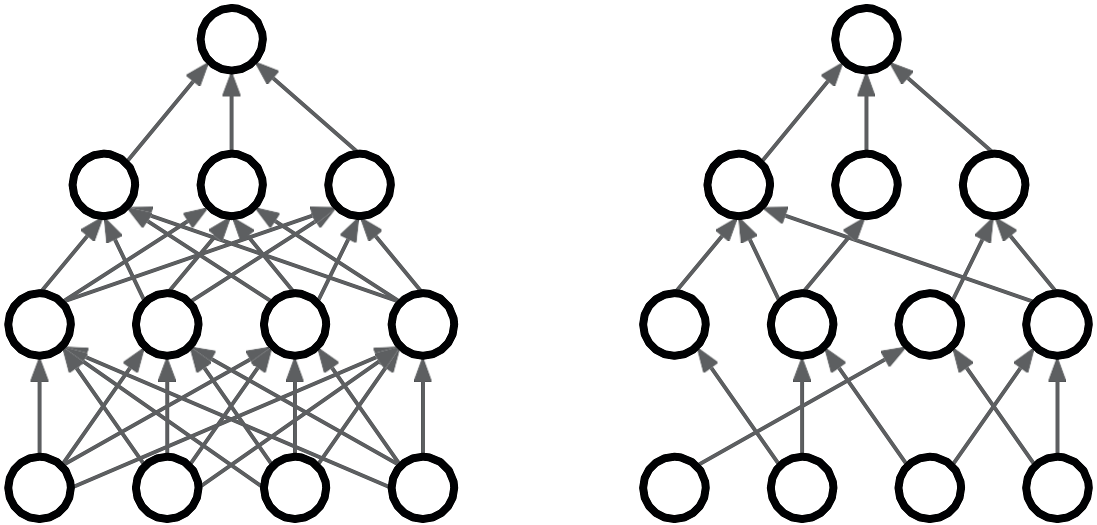
Examples:

Dropout

Batch Normalization

Data Augmentation

DropConnect

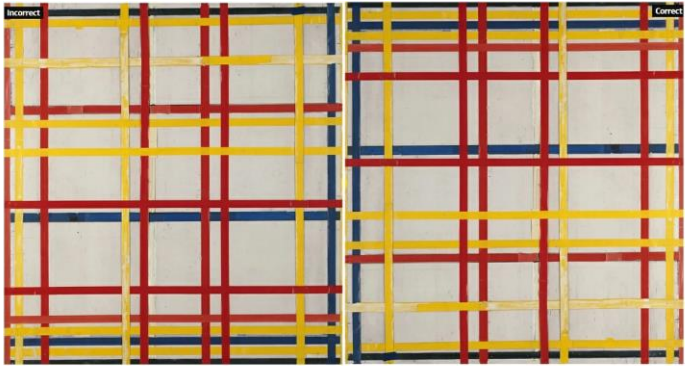


Wan et al, “Regularization of Neural Networks using DropConnect”, ICML 2013

Регуляризация: Fractional Pooling

Training: Используем случайные регионы пуллинга

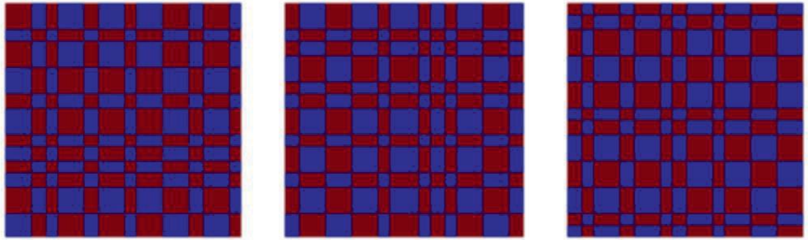
Testing: Усредняем по разным регионам

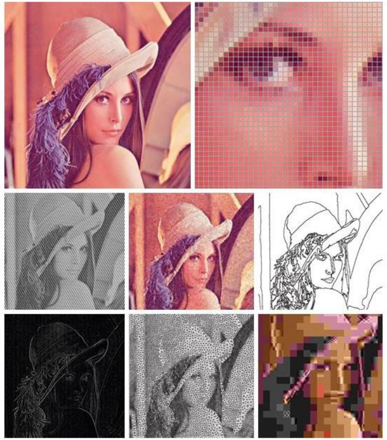


Перевернутый Мондриан

Examples:

Dropout
Batch Normalization
Data Augmentation
DropConnect
Fractional Max Pooling



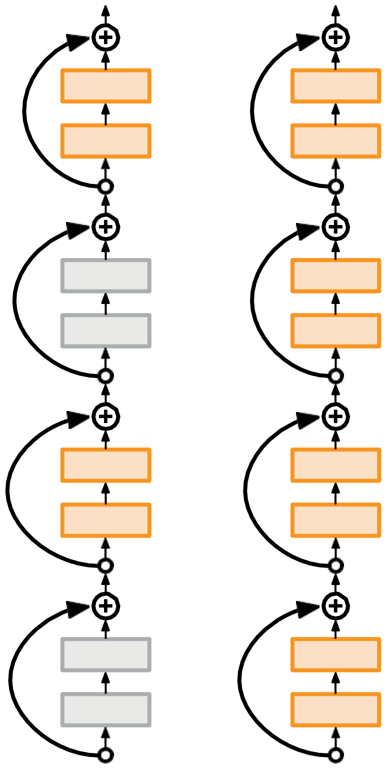

Lenna

Graham, “Fractional Max Pooling”, arXiv 2014

Регуляризация: Stochastic Depth

Training: Выкинем несколько слоев

Testing: Используем все слои



Examples:

Dropout

Batch Normalization

Data Augmentation

DropConnect

Fractional Max Pooling

Stochastic Depth

Huang et al, “Deep Networks with Stochastic Depth”, ECCV 2016

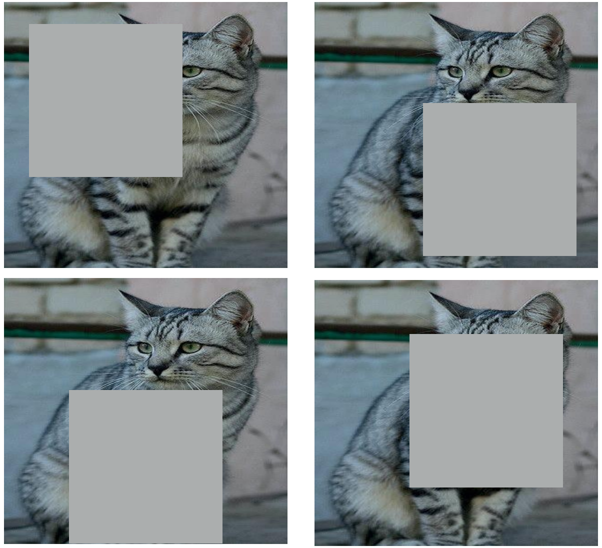
Regularization: Cutout

Training: Закроем часть картинки

Testing: Используем всю картинку

Examples:

Dropout
Batch Normalization
Data Augmentation
DropConnect
Fractional Max Pooling
Stochastic Depth
Cutout / Random Crop



Хорошо работает на чем-то маленьком вроде CIFAR, похуже на реальных данных вроде ImageNet

DeVries and Taylor, “Improved Regularization of Convolutional Neural Networks with Cutout”, arXiv 2017

Регуляризация: Mixup

Training: Учим на случайных наложениях картинок

Testing: Используем оригинальные картинки

Examples:

Dropout

Batch Normalization

Data Augmentation

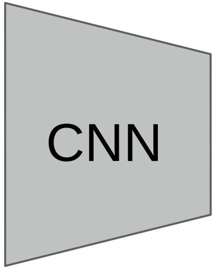
DropConnect

Fractional Max Pooling

Stochastic Depth

Cutout / Random Crop

Mixup



Target label:
cat: 0.4
dog: 0.6

Наложим две картинки с прозрачностью,
например, 40% cat, 60% dog

Zhang et al, “*mixup*: Beyond Empirical Risk Minimization”, ICLR 2018

Regularization - Практика

Training: Добавим случайные искажения

Testing: Маргинализируем по искажениям

Examples:

Dropout
Batch Normalization
Data Augmentation
DropConnect
Fractional Max Pooling
Stochastic Depth
Cutout / Random Crop
Mixup

- dropout хорош для больших полносвязных (FC) каскадов, BatchNorm - предпочтительнее
- BatchNorm и аугментация почти всегда отлично работает
- Попробуйте cutout и mixup особенно для маленьких датасетов

Подбор гиперпараметров

(без сотен GPU!)

Подбор гиперпараметров

Step 1: Check initial loss - Смотрим, какой получается loss на старте обучения

Включаем/отключаем затухание весов (weight decay), проверяем корректность loss на старте
Например, log(C) для softmax с C классами

Подбор гиперпараметров

Step 1: Check initial loss
Step 2: Добьемся переобучения на маленьком датасете

Постараемся получить 100% точности на маленьком дадасете (~5-10 батчей); подбираем архитектуру, learning rate, инициализацию

Loss не убывает? LR слишком маленький, не угадали с инициализацией
Loss улетает в Inf или NaN? LR слишком большой, не угадали с инициализацией

Подбор гиперпараметров

Step 1: Check initial loss
Step 2: Overfit a small sample
Step 3: Подберем LR который дает убывание ошибки (loss)

Архитектуру берем с предыдущего шага, берем всю обучающую выборку, включаем weight decay с небольшим весом, подбираем learning rate который дает значительное падение loss за ~100 итераций

Хороший набор LR для начала: 1e-1, 1e-2, 1e-3, 1e-4

Подбор гиперпараметров

Step 1: Check initial loss
Step 2: Overfit a small sample
Step 3: Find LR that makes loss go down
Step 4: С грубой сеткой обучать ~1-5 эпох

Для нескольких значений LR и weight decay которые работали на шаге 3, обучить несколько моделей
Выбираем несколько значений LR и weight decay около того, что работало на Шаге 3, учим несколько моделей ~1-5 эпох.
Хорошие варианты weight decay: 1e-4, 1e-5, 0

Подбор гиперпараметров

Step 1: Check initial loss
Step 2: Overfit a small sample
Step 3: Find LR that makes loss go down
Step 4: Coarse grid, train for ~1-5 epochs
Step 5: Делаем мелкую сетку, учим дольше

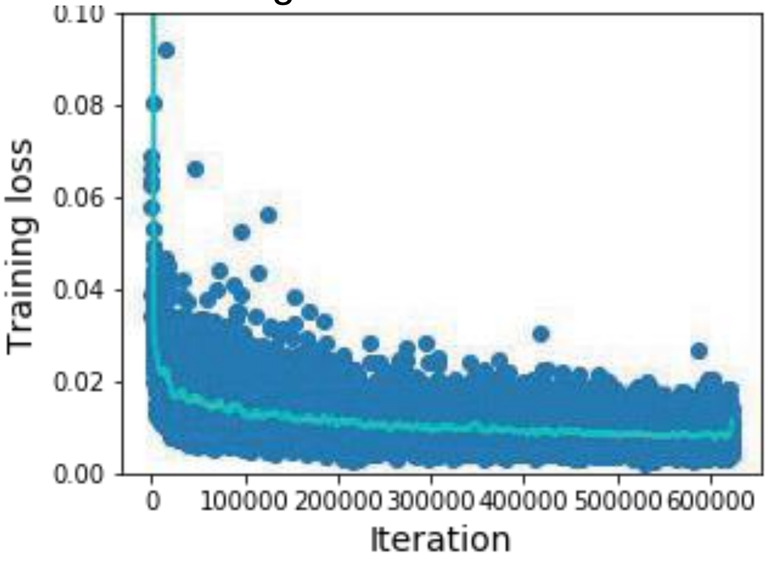
Берем лучшие модели с Шага 4, учим дольше (~10-20 эпох) без уменьшения LR

Подбор гиперпараметров

Step 1: Check initial loss
Step 2: Overfit a small sample
Step 3: Find LR that makes loss go down
Step 4: Coarse grid, train for ~1-5 epochs
Step 5: Refine grid, train longer
Step 6: Запускаем надолго, смотрим на графики loss

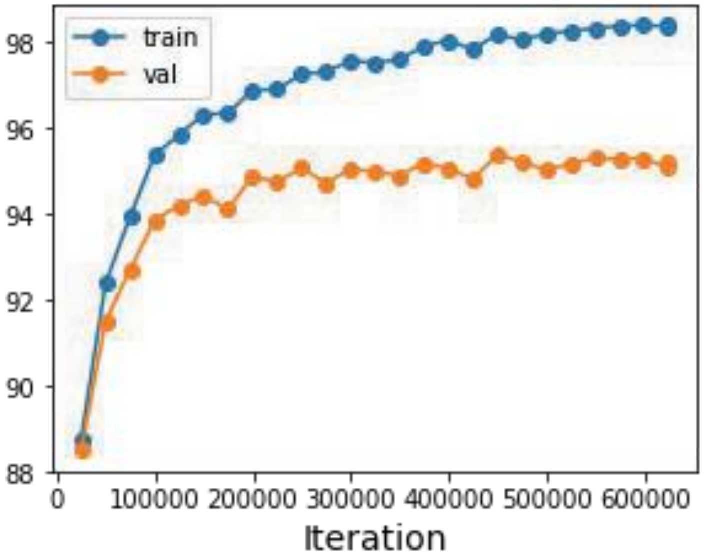
Графики ошибок и точности

Training Loss/Ошибка



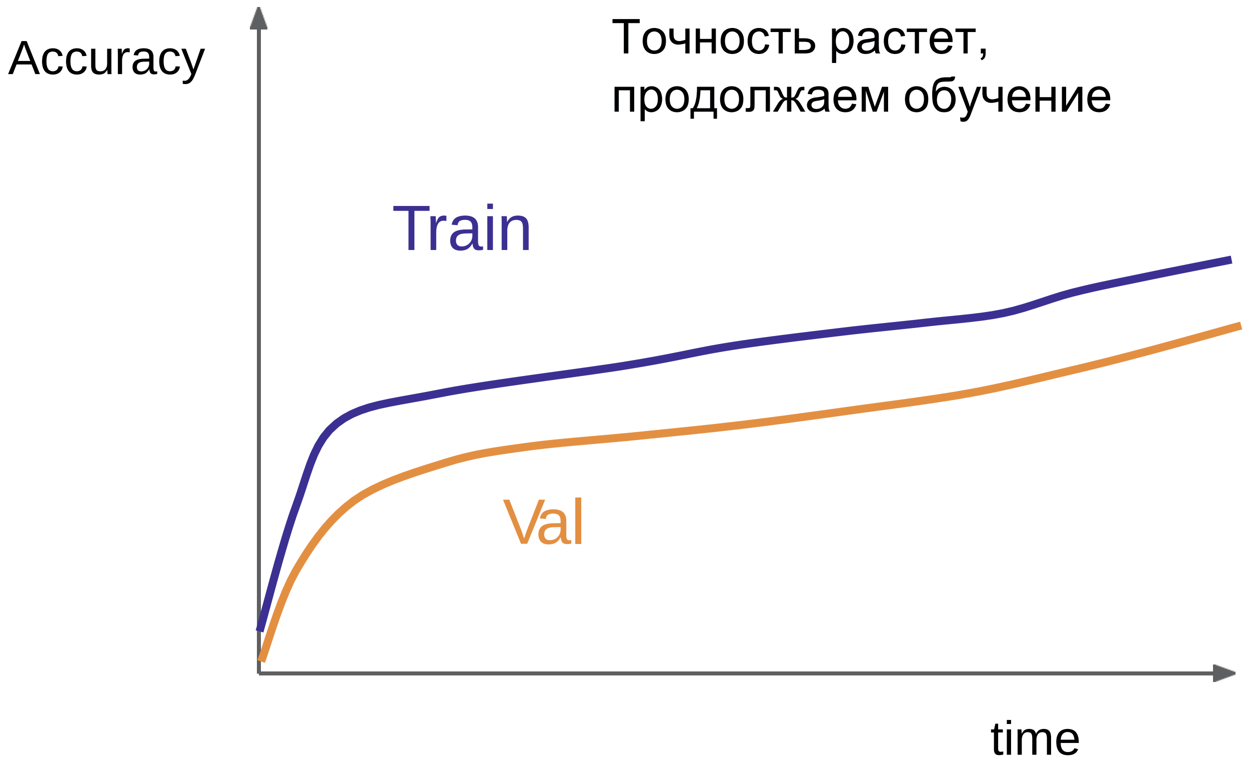
Ошибка обычно шумная, используйте скользящее среднее

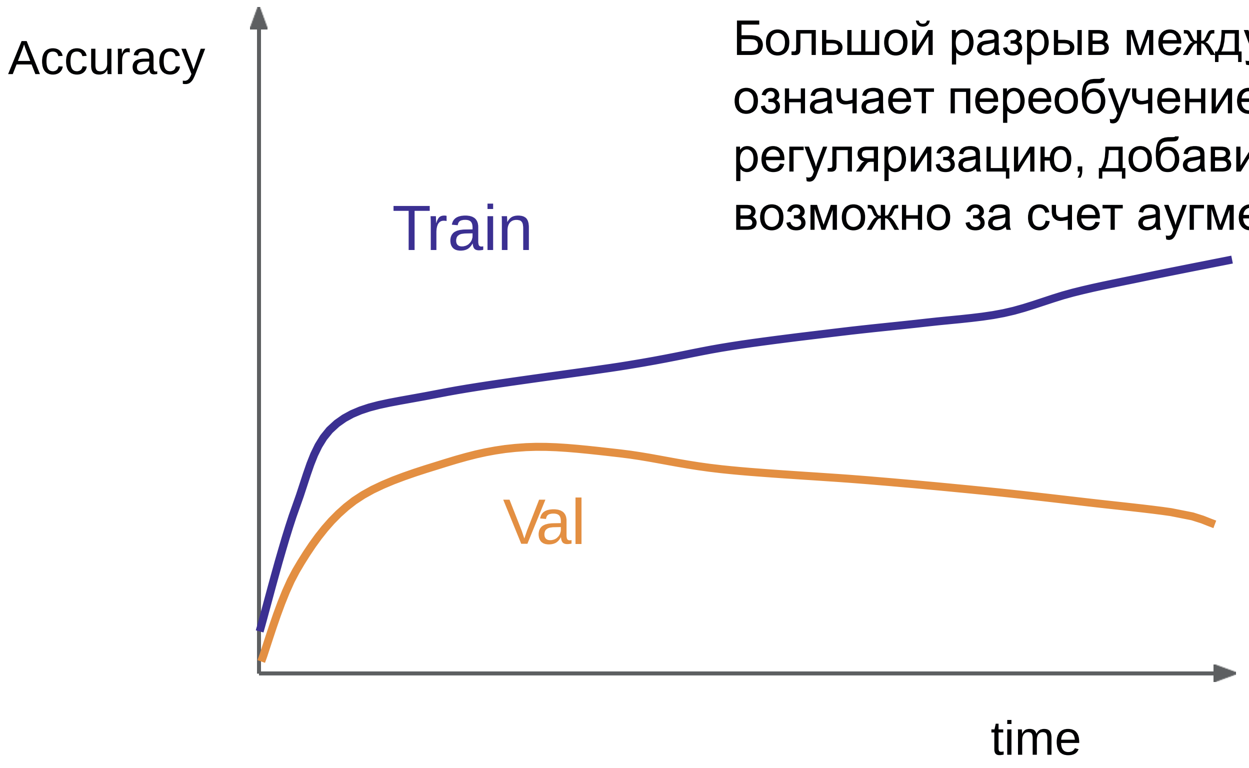
Train / Val Accuracy / Точность



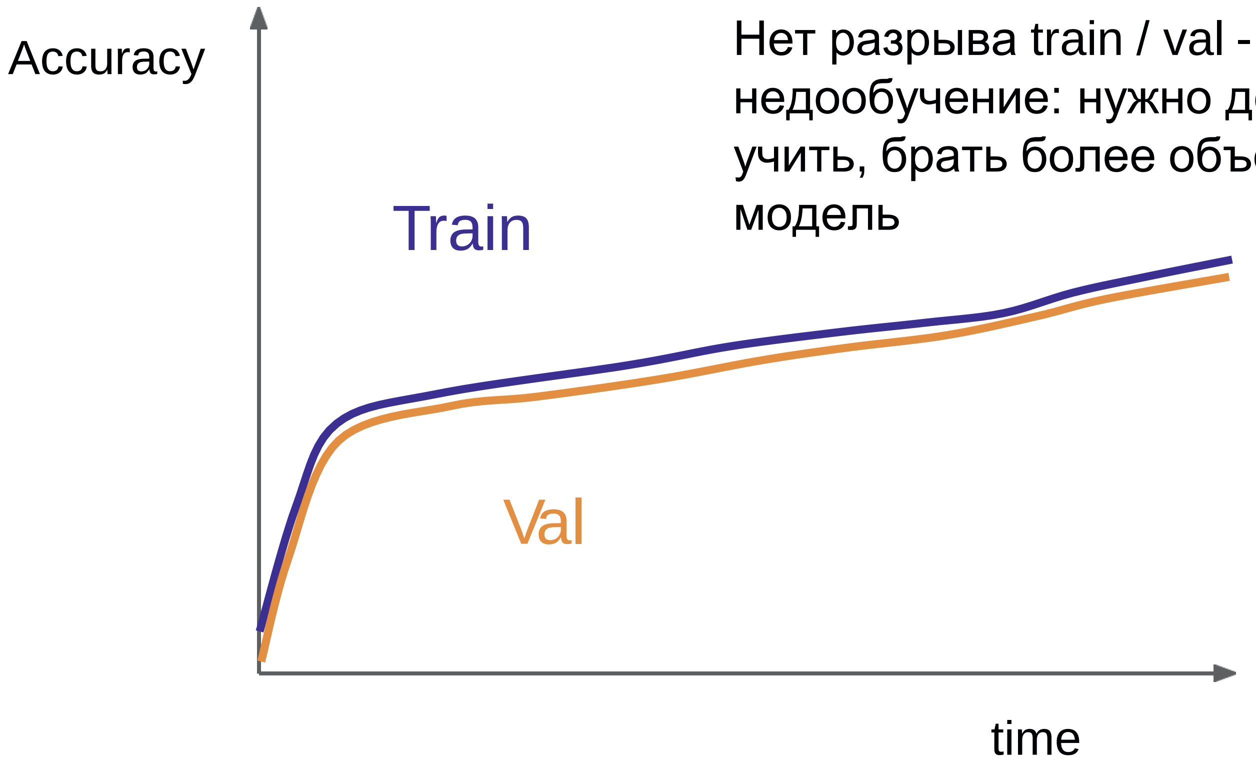
Точность всегда финально считается на валидационном наборе, **который не входит в обучение**

Точность растет,
продолжаем обучение





Большой разрыв между train / val означает переобучение! Нужно усилить регуляризацию, добавить данных, возможно за счет аугментации



Нет разрыва train / val - недообучение: нужно дольше учить, брать более объемную модель

Подбор гиперпараметров

Step 1: Check initial loss
Step 2: Overfit a small sample
Step 3: Find LR that makes loss go down
Step 4: Coarse grid, train for ~1-5 epochs
Step 5: Refine grid, train longer
Step 6: Look at loss curves
Step 7: Идем на шаг 5

Подбор гиперпараметров

Step 1: Начальная проверка loss
Step 2: Переобучим на маленьком примере (до ~100%)
Step 3: Подберем LR который приводит к уменьшению loss
Step 4: Грубый подбор параметров, учим ~1-5 эпох
Step 5: Точный подбор параметров, учим дольше
Step 6: Наблюдаем за ошибкой и точностью
Step 7: Идем на шаг 5

Какие гиперпараметры подбираем:

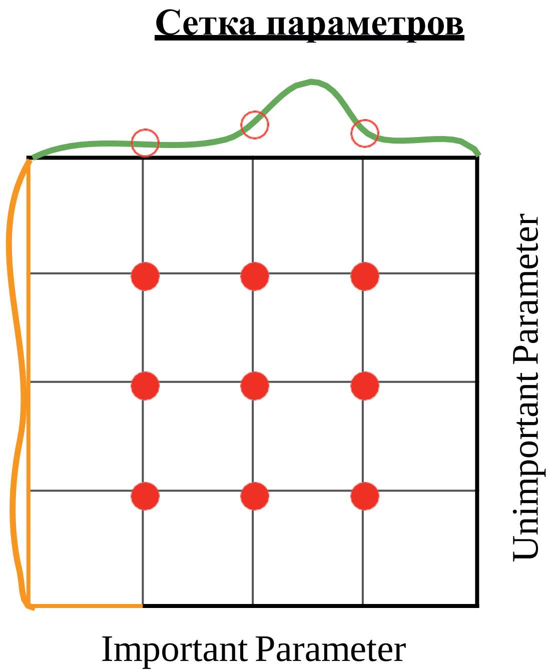
- Архитектура сети
- learning rate, weight decay, политику обновления
- регуляризация (L2/Dropout, их вес)

Командный центр tensorboard или – WandB

https://neptune.ai/blog/the-best-tensorboard-alternatives

Random Search vs. Grid Search

Random Search for Hyper-Parameter Optimization
Bergstra and Bengio, 2012



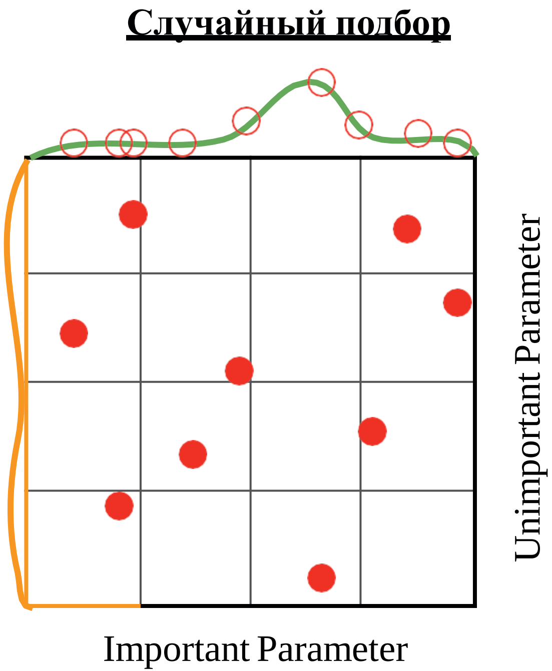


Illustration of Bergstra et al., 2012 by Shayne Longpre, copyright CS231n 2017

Итого

- Уменьшение ошибки обучения:
  - Оптимизаторы
  - Изменение learning rate
- Увеличение точности:
  - Регуляризация
  - Подбор гиперпарарметров

Далее: Архитектуры СНС