

Основы глубинного обучения

Лекция 4

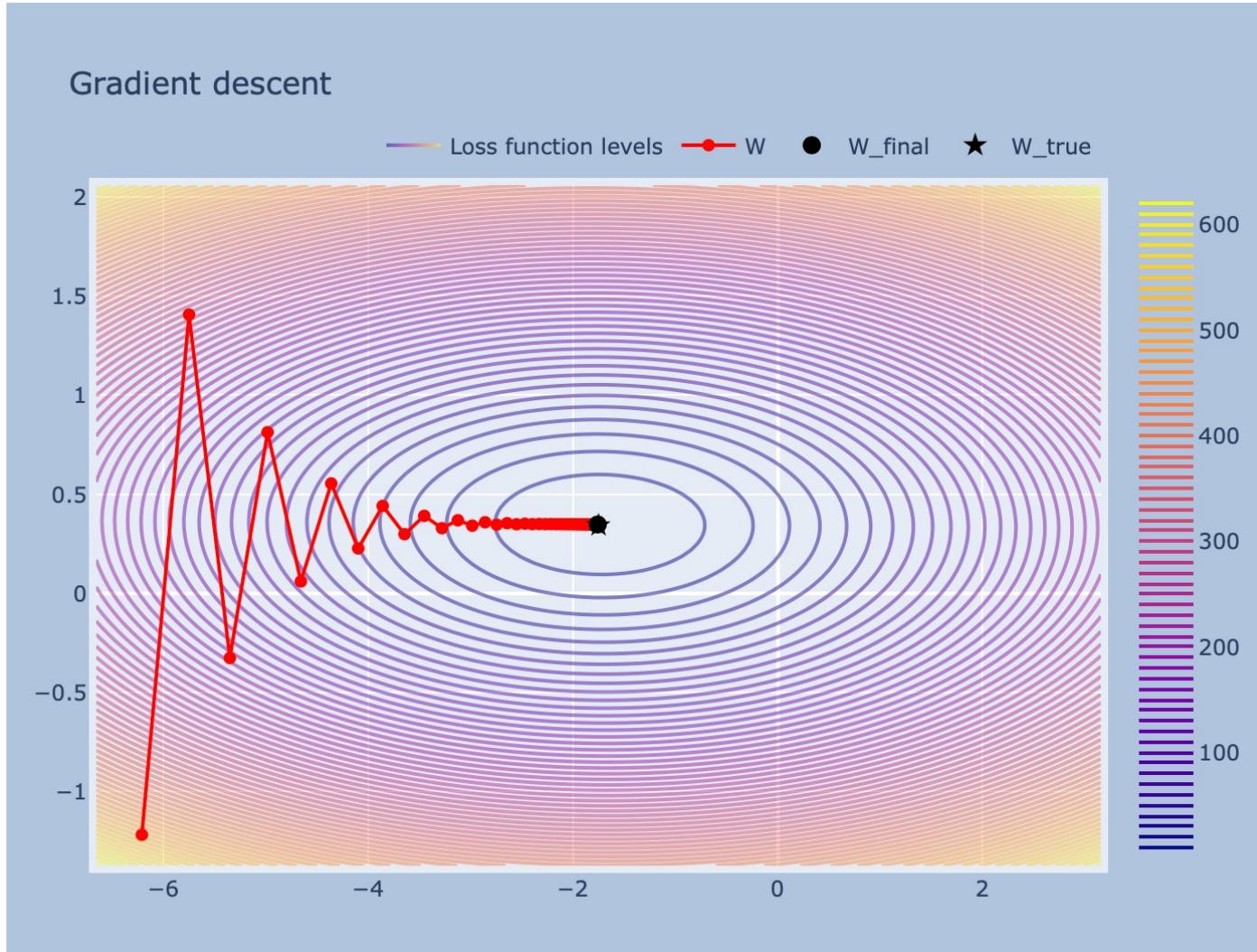
Оптимизация в глубинном обучении.
Свёрточные архитектуры.

Евгений Соколов
esokolov@hse.ru

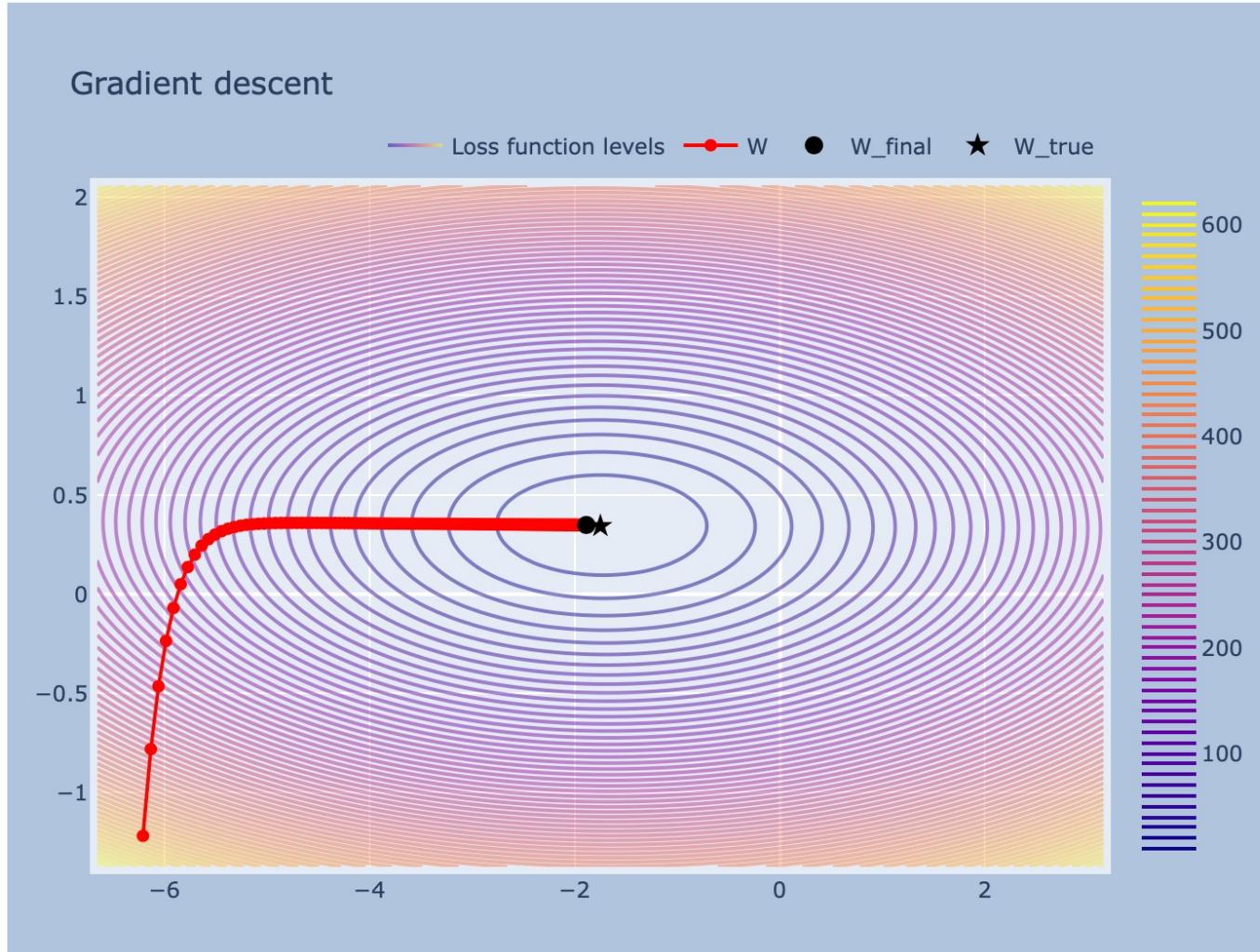
НИУ ВШЭ, 2025

Модификации градиентного спуска

Проблемы



Проблемы



Проблемы

- Если у функции «вытянуты» линии уровня, то градиентный спуск требует аккуратного выбора длины шага и будет долго сходиться

Momentum

$$h_t = \alpha h_{t-1} + \eta_t \nabla Q(w^{t-1})$$

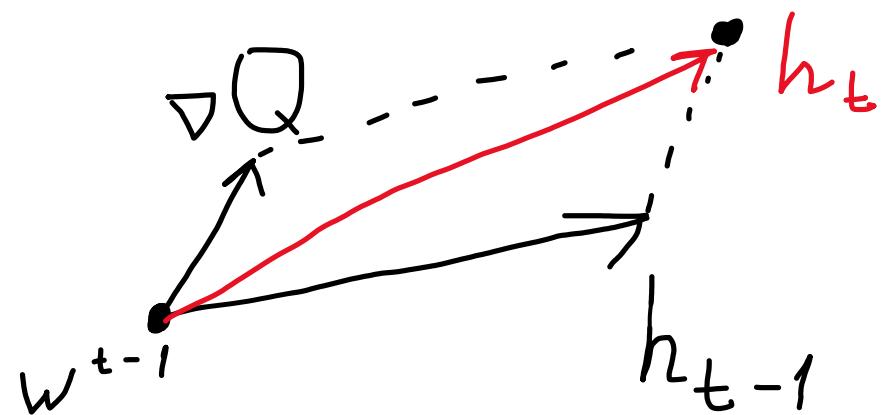
$$w^t = w^{t-1} - h_t$$

- h_t — «инерция», усреднённое направление движения
- α — параметр затухания
- Как будто шарик, который катится в сторону минимума, очень тяжёлый

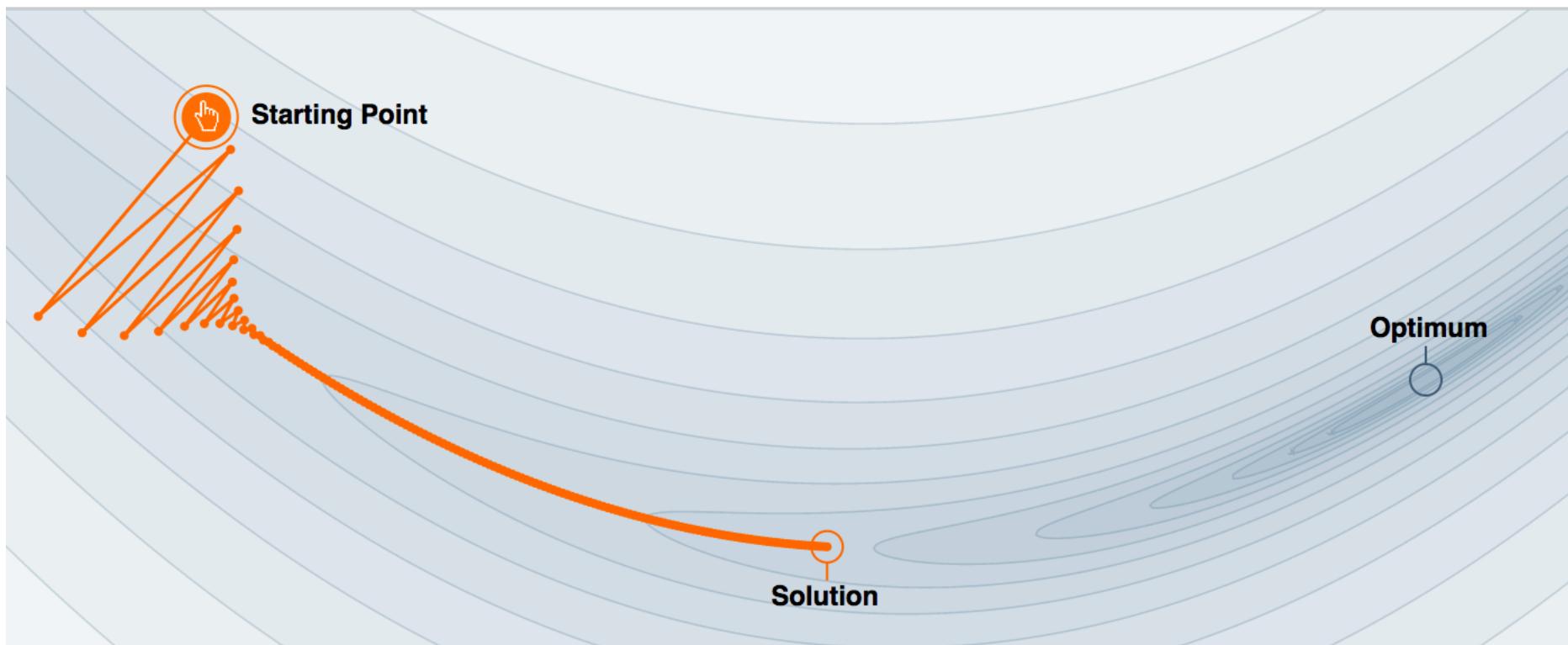
Momentum

$$h_t = \alpha h_{t-1} + \eta_t \nabla Q(w^{t-1})$$

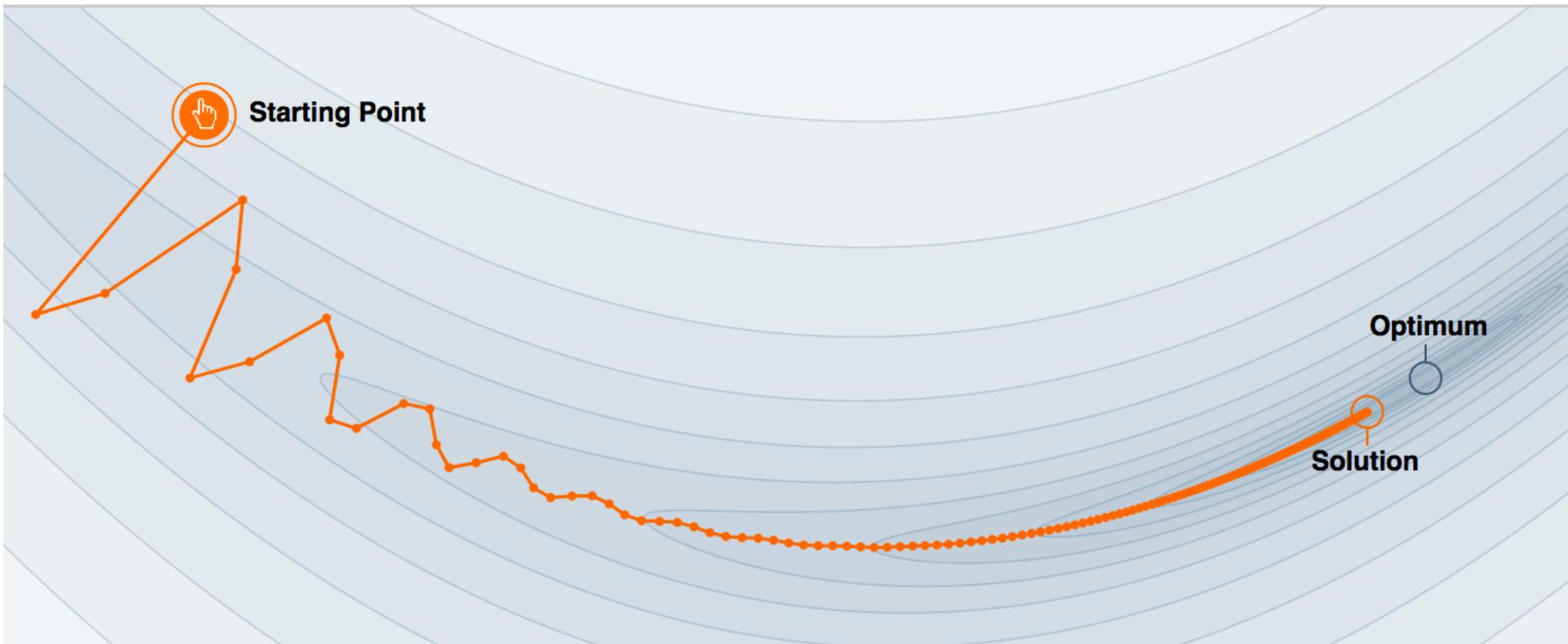
$$w^t = w^{t-1} - h_t$$



Без инерции



Синерцией

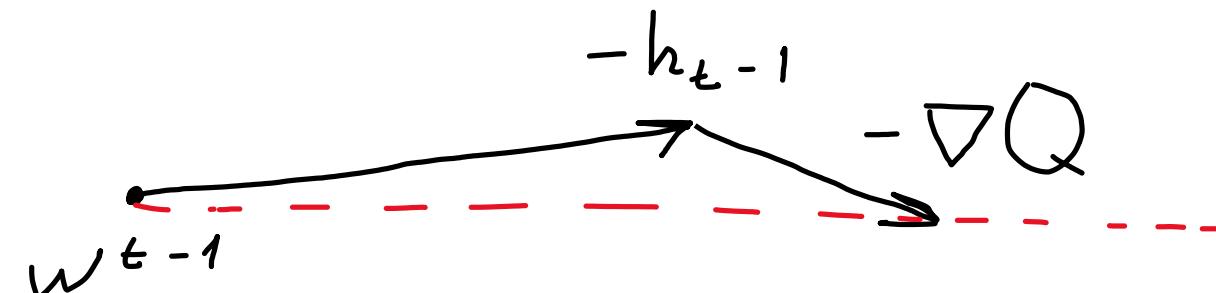
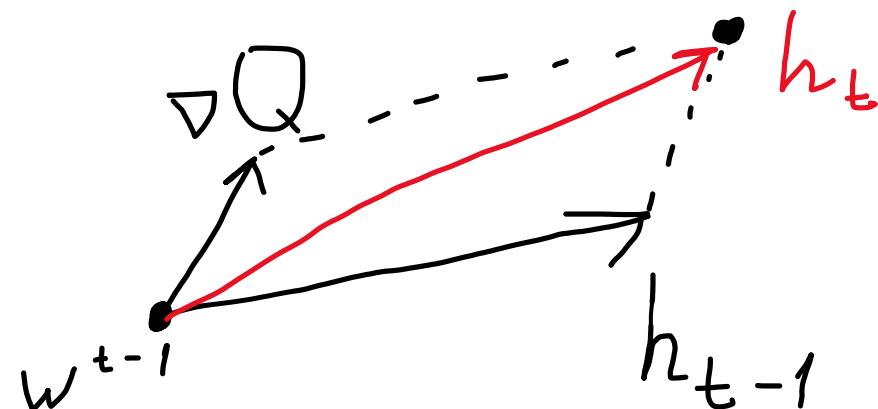


Nesterov Momentum

$$h_t = \alpha h_{t-1} + \eta_t \nabla Q(w^{t-1} - \alpha h_{t-1})$$

$$w^t = w^{t-1} - h_t$$

- $w^{t-1} - \alpha h_{t-1}$ — неплохая оценка того, куда мы попадём на следующем шаге



Проблема с разреженными данными

- Пример: модель над категориальными признаками
 - Используем one-hot кодирование

	1		0
	0		0
	1		0
	1		0
	1		0
	0		0
	0		1

- Делаем стохастический градиентный спуск
 - Через 7 шагов мы сделаем 4 обновления веса популярной категории и 1 обновление веса редкой категории

• тут уже медленные шаги

Проблема с разреженными данными

- По разным параметрам мы движемся с разной скоростью
- Будет здорово это учитывать — иначе мы обучим разные параметры с разным качеством

Проблема с разными масштабами

- Допустим, признаки имеют разный масштаб — от единиц до миллионов
- Тогда странно шагать по каждому параметру с одинаковой скоростью

AdaGrad

$$G_j^t = G_j^{t-1} + (\nabla Q(w^{t-1}))_j^2$$

$$w_j^t = w_j^{t-1} - \frac{\eta_t}{\sqrt{G_j^t + \epsilon}} (\nabla Q(w^{t-1}))_j$$

- По каждому параметру своя скорость
- η_t можно зафиксировать
- Длина шага может убывать слишком быстро и привести к ранней остановке

RMSProp

$$G_j^t = \alpha G_j^{t-1} + (1 - \alpha) (\nabla Q(w^{t-1}))_j^2$$

$$w_j^t = w_j^{t-1} - \frac{\eta_t}{\sqrt{G_j^t + \epsilon}} g_{tj}$$

- α можно взять около 0.9
- Скорость зависит только от недавних шагов

Adam

$$m_j^t = \frac{\beta_1 m_j^{t-1} + (1 - \beta_1) (\nabla Q(w^{t-1}))_j}{1 - \beta_1^t}$$

$$v_j^t = \frac{\beta_2 v_j^{t-1} + (1 - \beta_2) (\nabla Q(w^{t-1}))_j^2}{1 - \beta_2^t}$$

$$w_j^t = w_j^{t-1} - \frac{\eta_t}{\sqrt{v_j^t + \epsilon}} m_j^t$$

- Рекомендации: $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 10^{-8}$

Adam

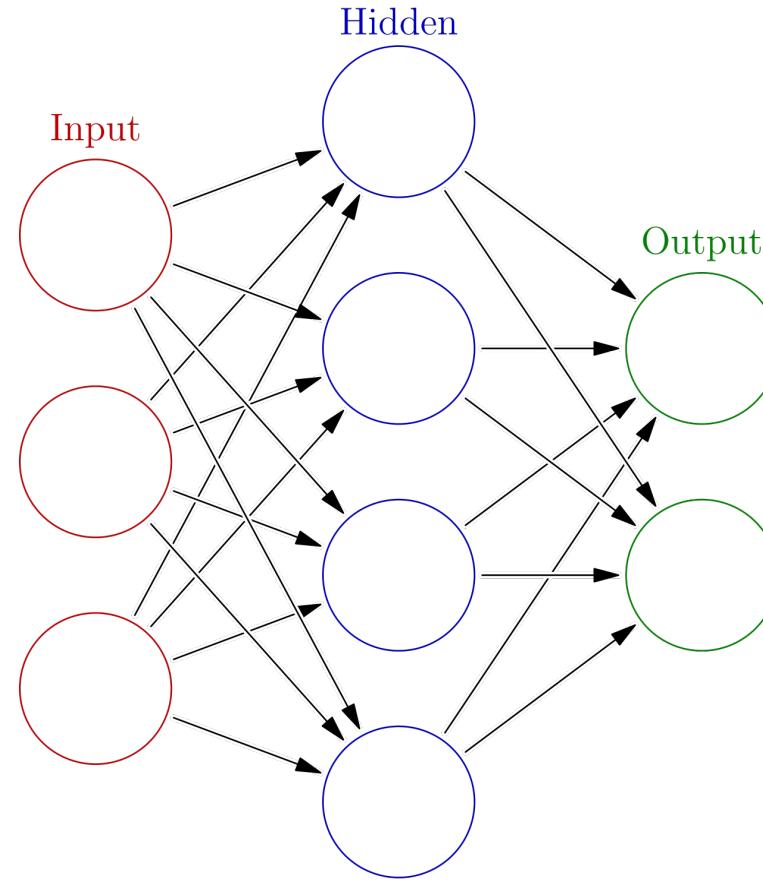
- Совмещает в себе идеи из метода инерции и RMSProp

Dropout

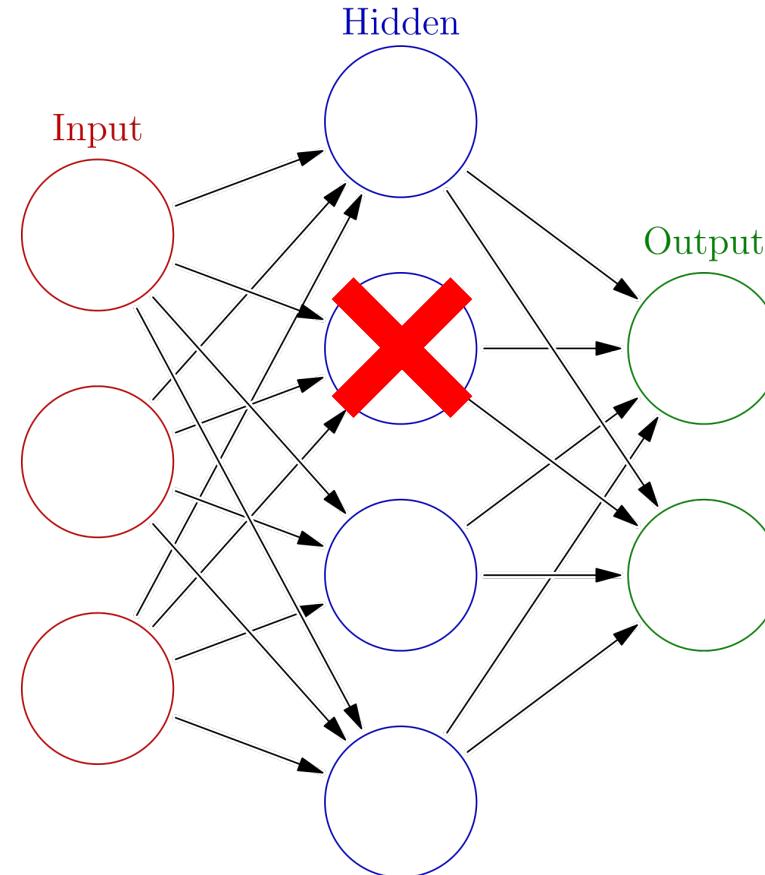
Борьба с переобучением

- Сокращение числа параметров (свёрточные слои помогают с этим)
- Регуляризация
- Можно как-то ещё мешать модели подгоняться под обучающую выборку

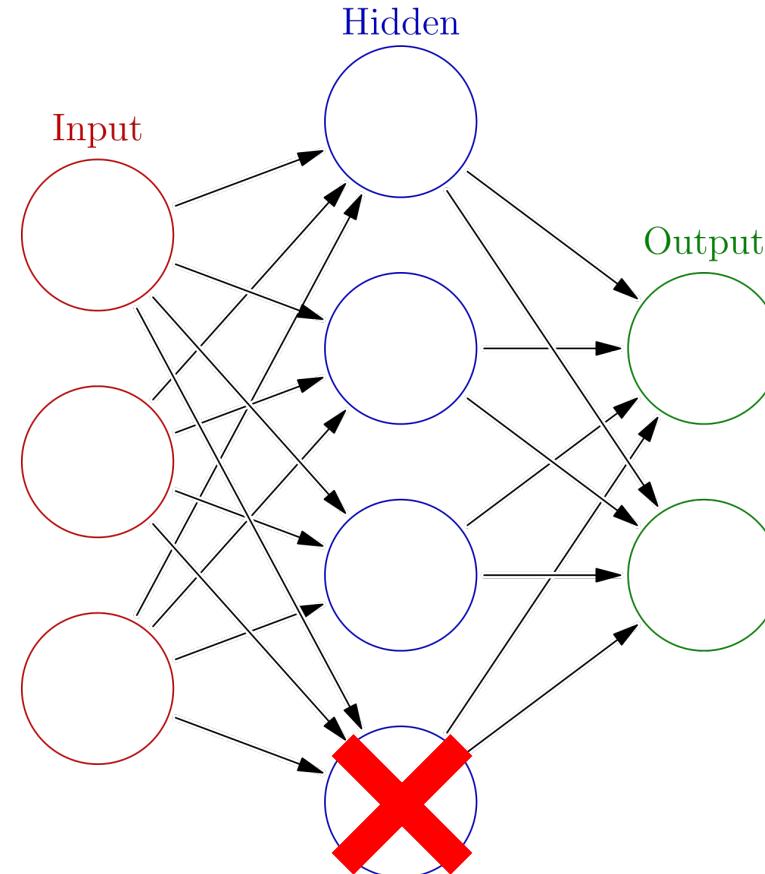
Dropout



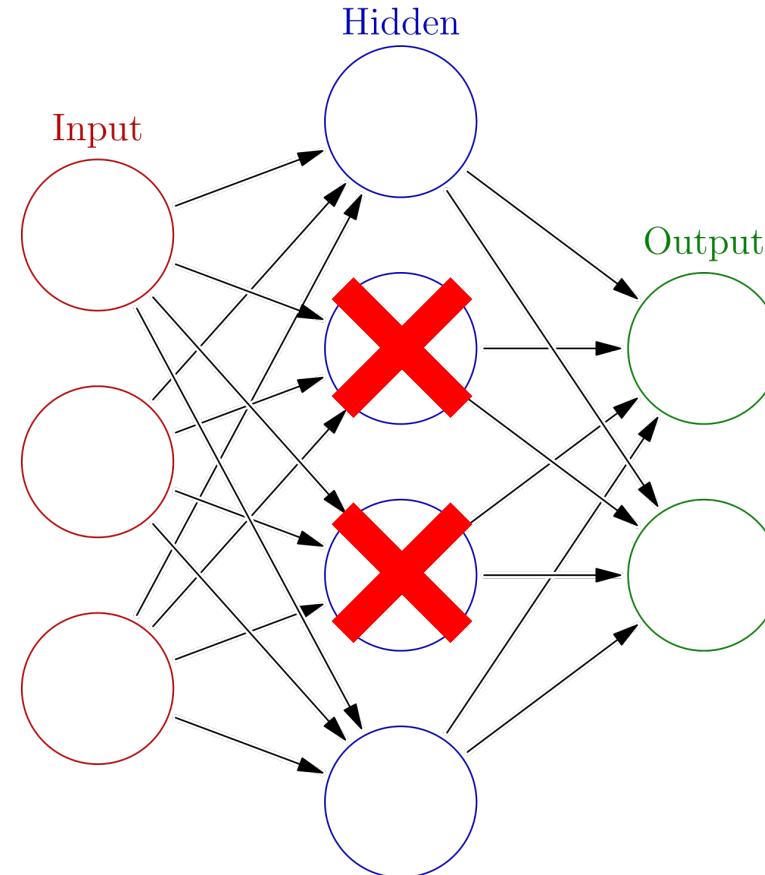
Dropout



Dropout



Dropout



Dropout

- Можно определить как слой $d(x)$
- Параметров нет, единственный гиперпараметр — p (вероятность удаления нейрона)
- На этапе обучения:

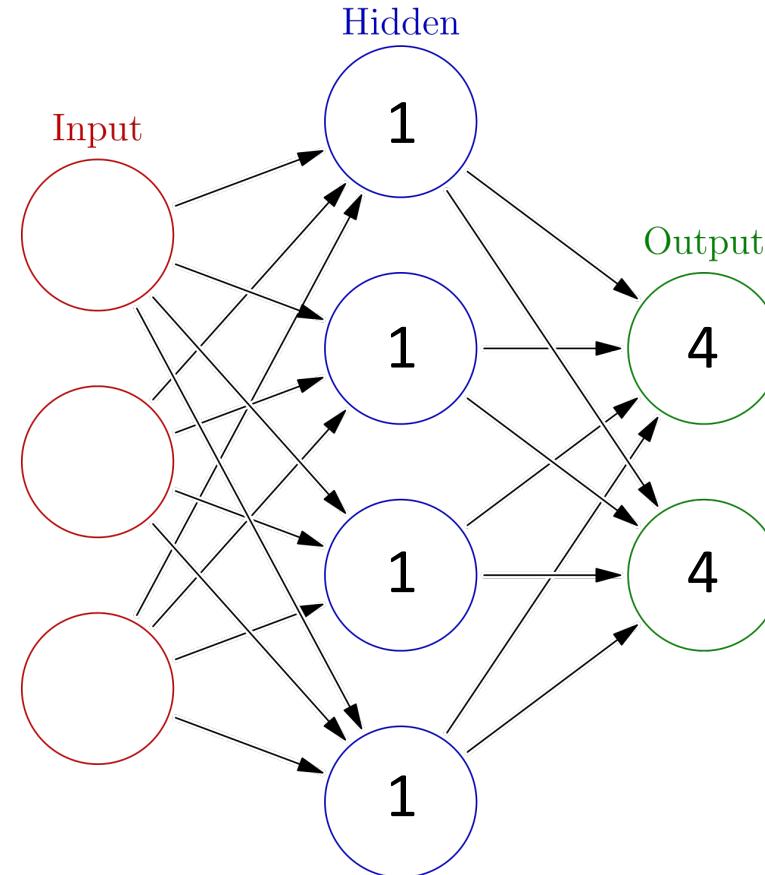
$$d(x) = \frac{1}{1-p} m \circ x$$

(m — вектор того же размера, что и x , элемент берутся из распределения $\text{Ber}(p)$)

- Деление на p — для сохранения суммарного масштаба выходов

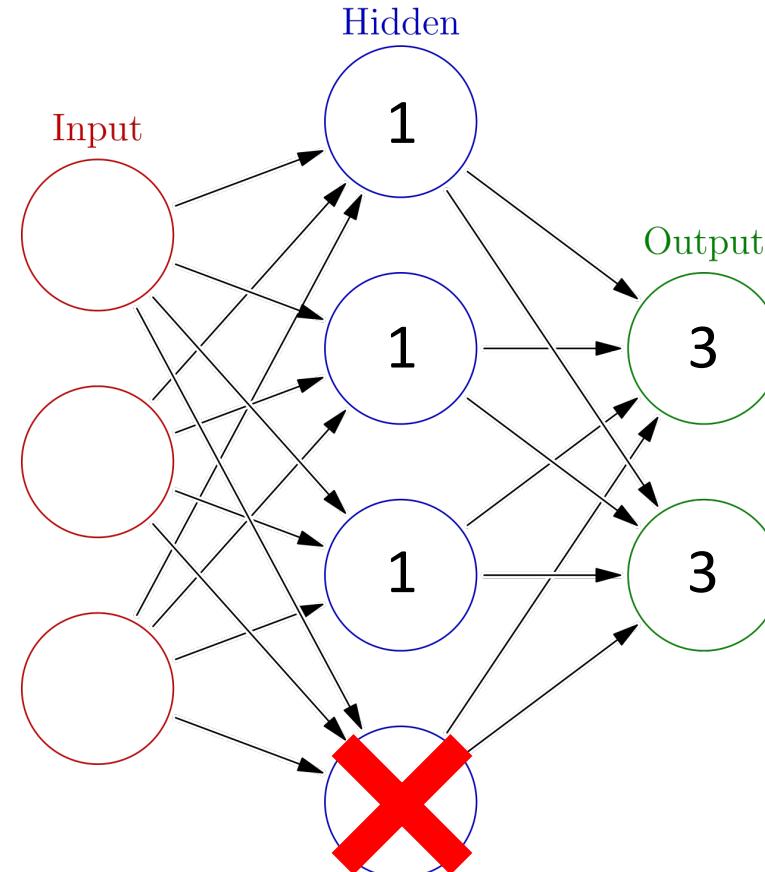
Dropout

Пусть все веса единичные



Dropout

Пусть все веса единичные



Надо компенсировать снижение
масштаба суммы выходов!

Dropout

- На этапе обучения:

$$d(x) = \frac{1}{1-p} m \circ x$$

- На этапе применения:

$$d(x) = x$$

В оригинальной статье нет нормировки на этапе обучения, но есть умножение на $(1 - p)$ на этапе применения

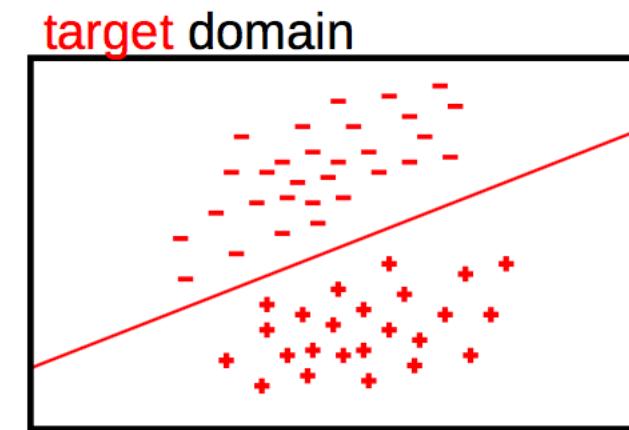
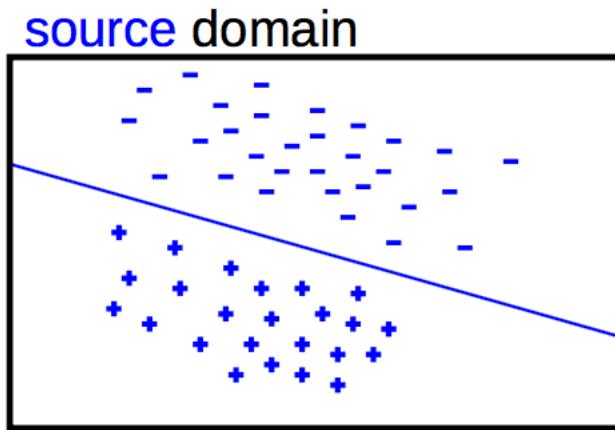
Вариант на слайде — inverted dropout (чуть меньше операций во время применения сети)

Dropout

- Интерпретация: мы обучаем все возможные архитектуры нейросетей, которые получаются из исходной выбрасыванием отдельных нейронов
- У всех этих архитектур общие веса
- На этапе применения (почти) усредняем прогнозы всех этих архитектур

Нормализации

Covariate shift



Covariate shift

- В классическом машинном обучении — изменение распределения данных
- Много методов решения

Domain adaptation

- Объекты по-разному распределены на обучении и на контроле
- Идея: взвешивать объекты при обучении

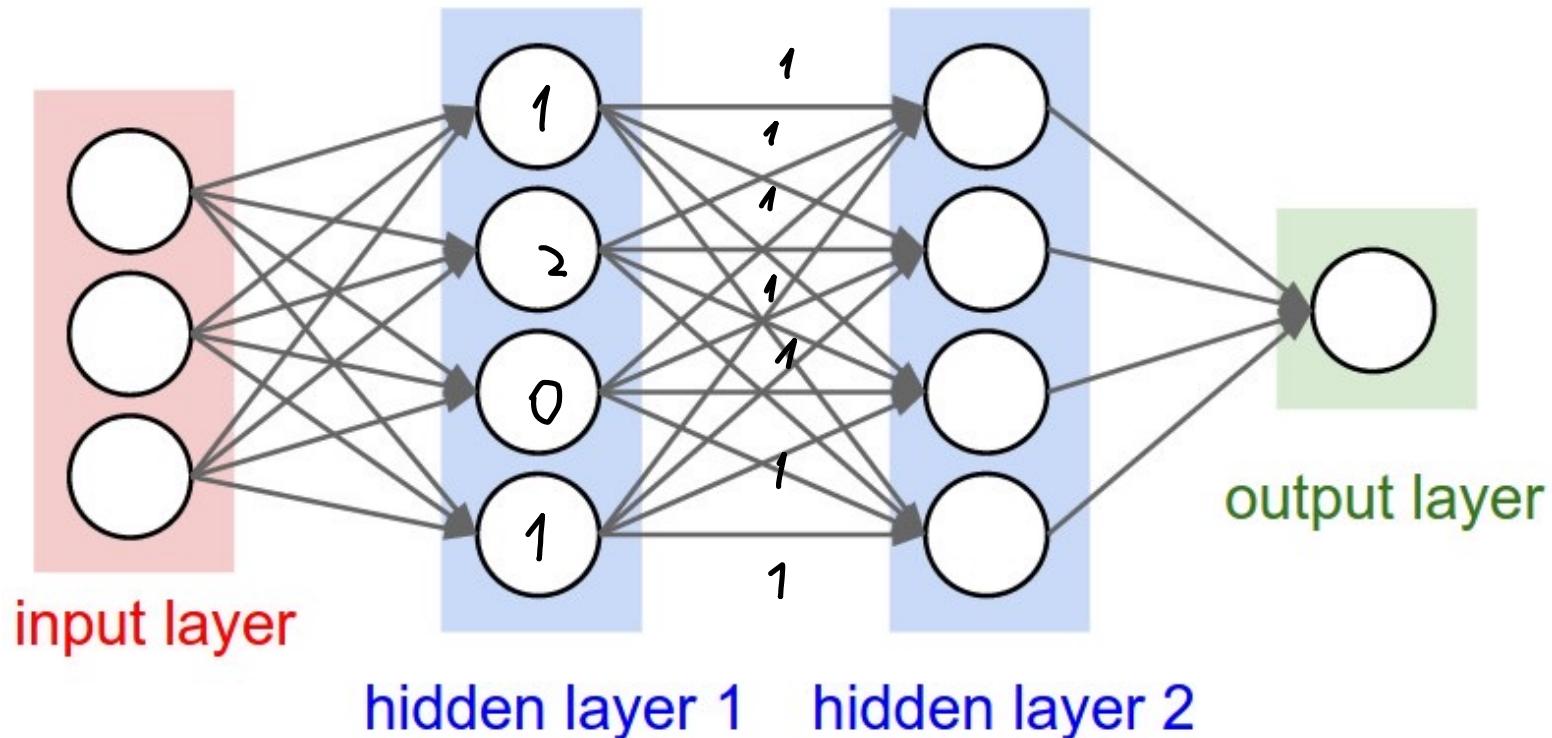
$$\sum_{i=1}^{\ell} s_i (a(x_i) - y_i)^2 \rightarrow \min$$

- Большие веса будем ставить объектам, которые похожи на объекты из тестовой выборки

Internal covariate shift

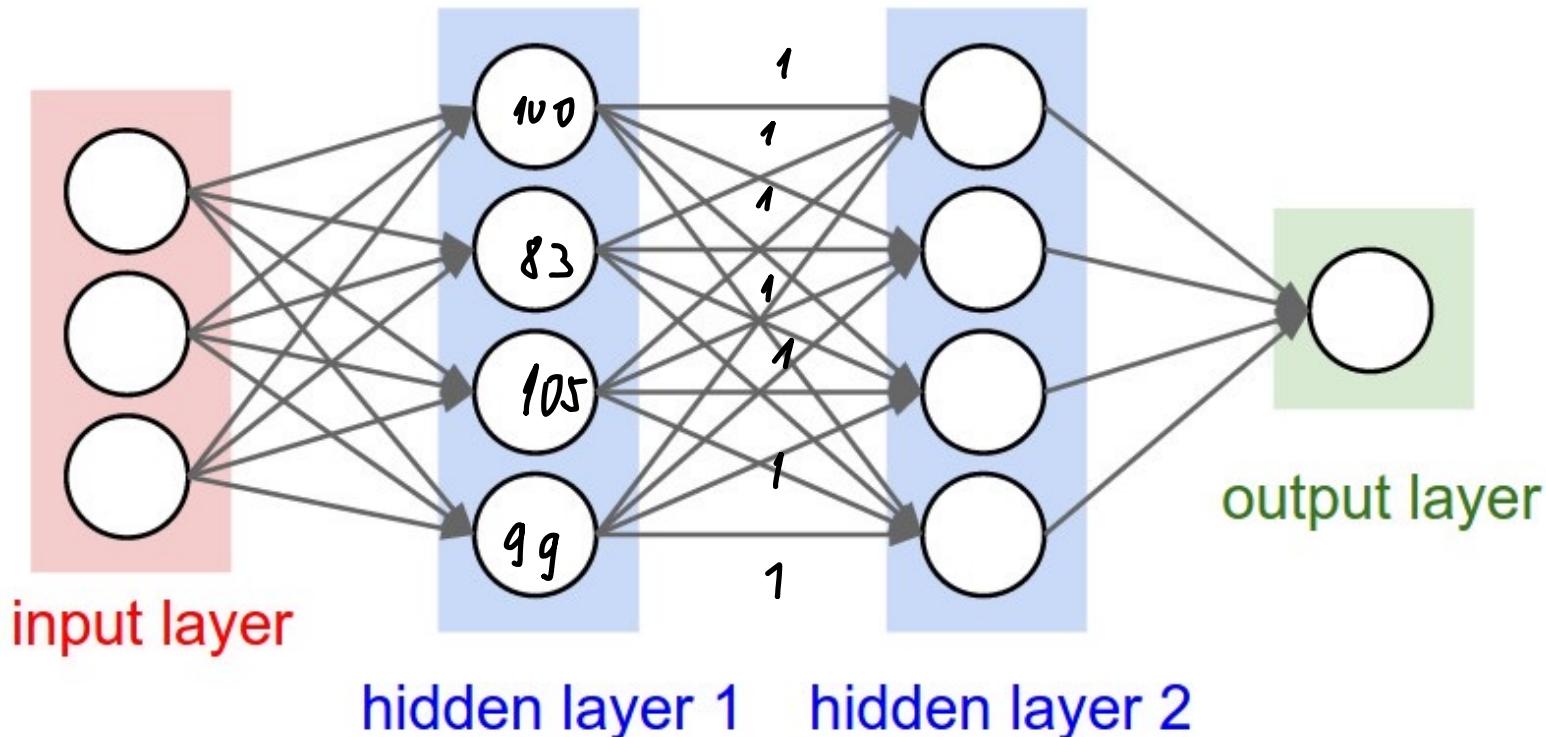
- В нейронной сети каждый слой обучается на выходах предыдущих слоёв
- Если слой в начале сильно меняется, то все следующие слои надо переделывать

Internal covariate shift



Internal covariate shift

Допустим, веса первого слоя сильно поменялись после градиентного шага



Internal covariate shift

- Идея: преобразовывать выходы слоёв так, чтобы они гарантированно имели фиксированное распределение

Batch Normalization

- Реализуется как отдельный слой
- Вычисляется для текущего батча
- Оценим среднее и дисперсию каждой компоненты входного вектора:

$$\mu_B = \frac{1}{n} \sum_{j=1}^n x_{B,j}$$

покоординатно

$$\sigma_B^2 = \frac{1}{n} \sum_{j=1}^n (x_{B,j} - \mu_B)^2$$

$x_{B,j}$ — j -й объект в батче B

Batch Normalization

- Отмасштабируем все выходы:

$$\tilde{x}_{B,j} = \frac{x_{B,j} - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}}$$

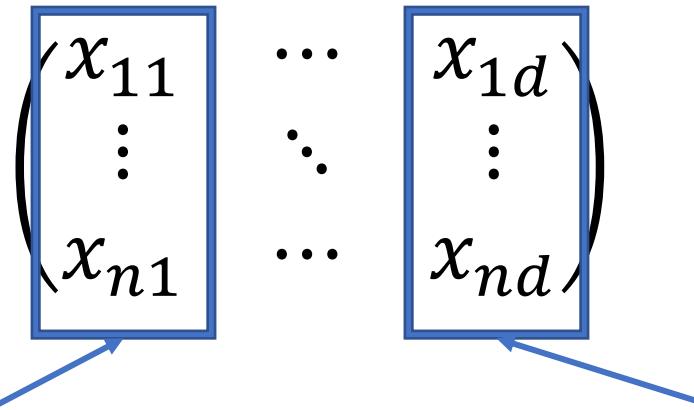
- Зададим нужные нам среднее и дисперсию:

$$z_{B,j} = \gamma \circ \tilde{x}_{B,j} + \beta$$



обучаемые параметры (векторы, размерность
равна размерности входных векторов)

Batch Normalization



Приводим среднее и
дисперсию к β_1 и γ_1

Приводим среднее и
дисперсию к β_d и γ_d

- n — размер батча
- d — размерность входного вектора

Batch Normalization

Важно: после BatchNorm среднее и дисперсия каждого выхода зависят только от параметров нормализации, но не от параметров прошлых слоёв!

Batch Normalization

Во время применения нейронной сети:

- Те же самые формулы, но вместо μ_B и σ_B^2 используем их средние значения по всем батчам

Batch Normalization

- Обычно вставляется между полносвязным/свёрточным слоём и нелинейностью
- Позволяет увеличить длину шага в градиентном спуске
- Не факт, что действительно устраняет covariance shift

В чём польза от BatchNorm?

How Does Batch Normalization Help Optimization?

Shibani Santurkar*
MIT
shibani@mit.edu

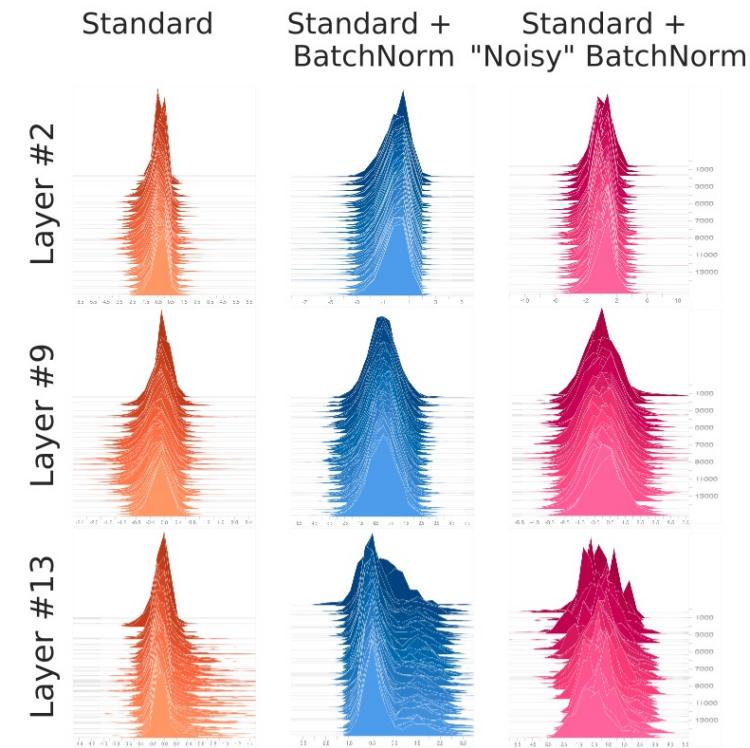
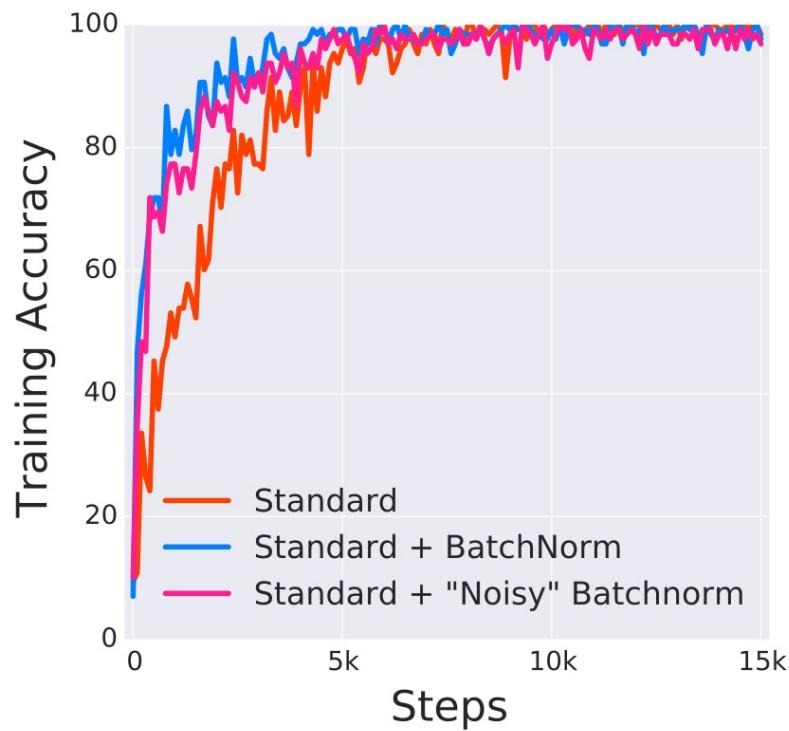
Dimitris Tsipras*
MIT
tsipras@mit.edu

Andrew Ilyas*
MIT
ailyas@mit.edu

Aleksander Mądry
MIT
madry@mit.edu

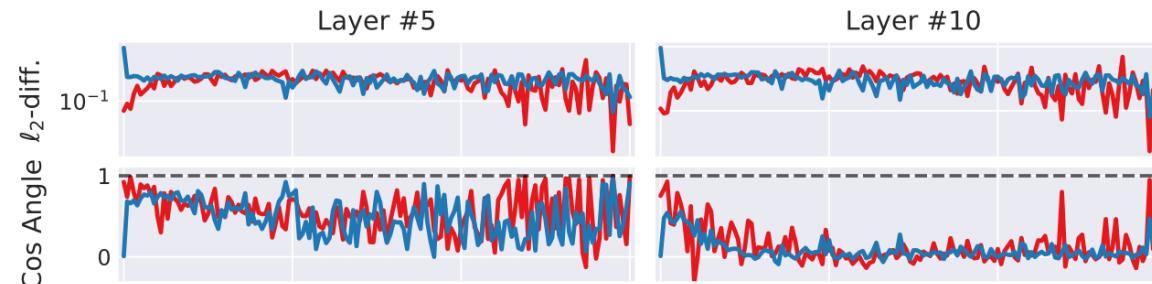
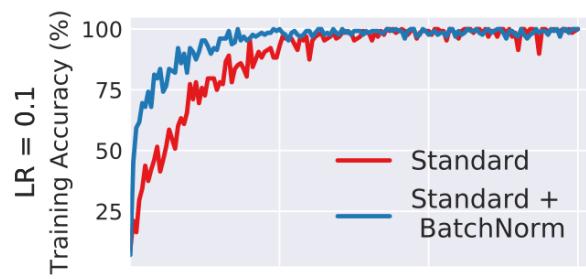
В чём польза от BatchNorm?

- Добавим шум после нормализации — хуже не становится!

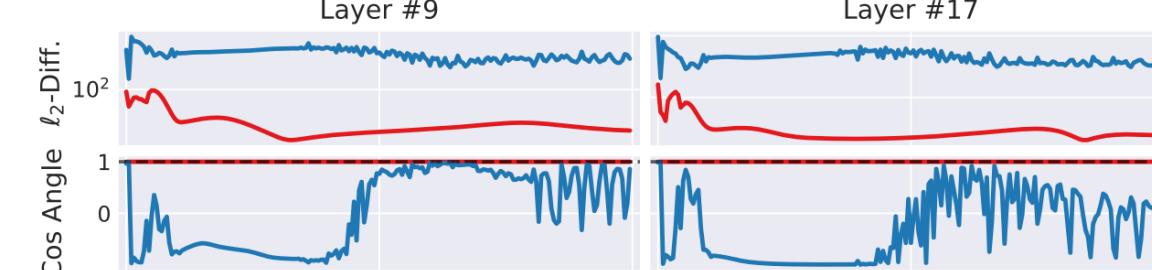
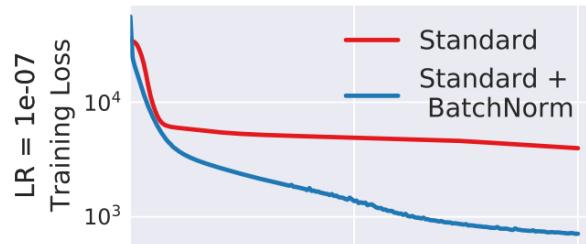


В чём польза от BatchNorm?

- Как связаны градиенты до и после обновления на предыдущих слоях?



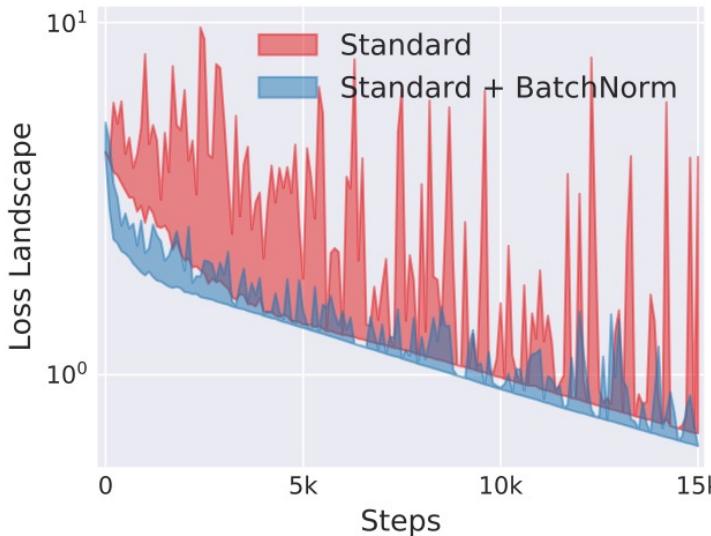
(a) VGG



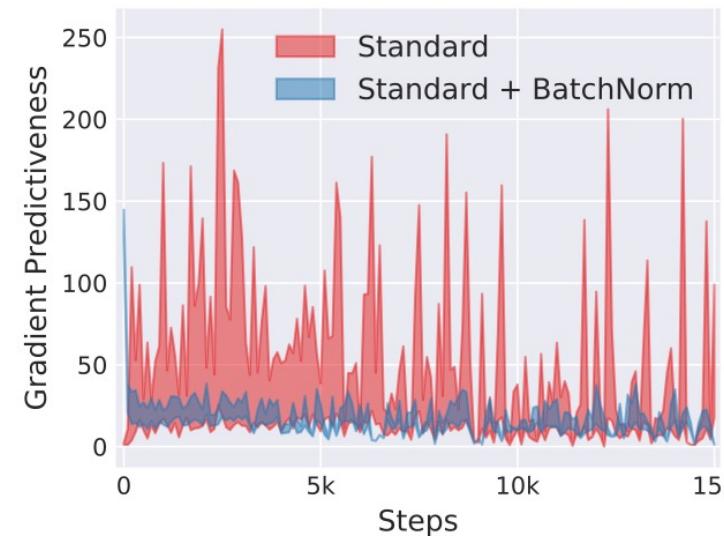
(b) DLN

В чём польза от BatchNorm?

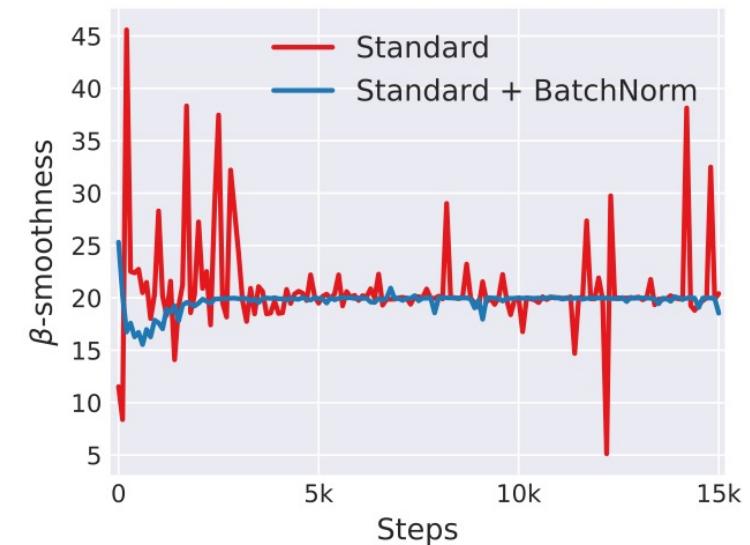
- Функционал ошибки становится более «гладким»!



(a) loss landscape

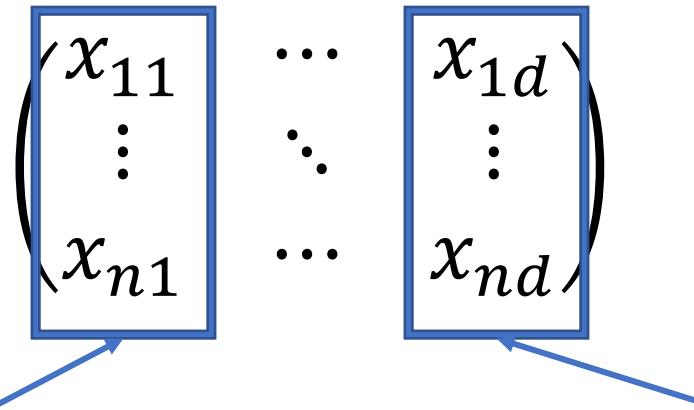


(b) gradient predictiveness



(c) “effective” β -smoothness

Batch Normalization

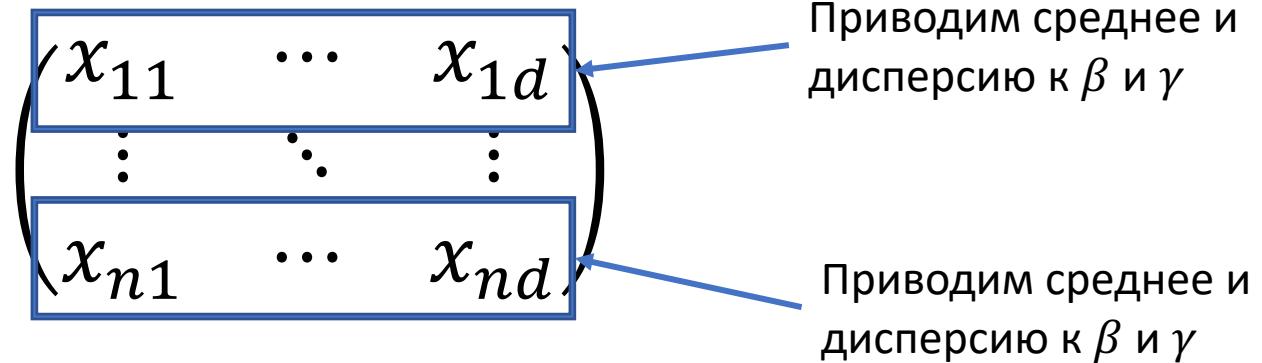


Приводим среднее и
дисперсию к β_1 и γ_1

Приводим среднее и
дисперсию к β_d и γ_d

- n — размер батча
- d — размерность входного вектора

Layer Normalization



- n — размер батча
- d — размерность входного вектора

Layer Normalization

- Нормализуем распределение «признаков» одного объекта

$$\mu_i = \frac{1}{d} \sum_{j=1}^d x_{ij}$$

$$\sigma_i^2 = \frac{1}{d} \sum_{j=1}^d (x_{ij} - \mu)^2$$

x_{ij} — j -й признак i -го объекта

Layer Normalization

- Отмасштабируем все выходы:

$$\tilde{x}_{ij} = \frac{x_{ij} - \mu_i}{\sqrt{\sigma_i^2 + \epsilon}}$$

- Зададим нужные нам среднее и дисперсию:

$$z_{ij} = \gamma \circ \tilde{x}_{ij} + \beta$$

↑ ↑
обучаемые параметры (скаляры)

Инициализации

Инициализация весов

- Не должно быть симметрий (плохо инициализировать всё одним числом)
- Хороший вариант:

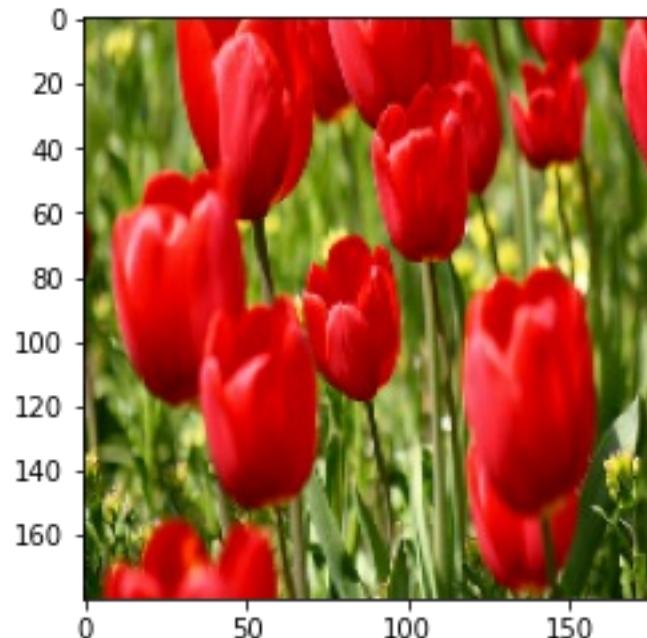
$$w_j \sim \frac{2}{\sqrt{n}} \mathcal{N}(0, 1)$$

n — число входов

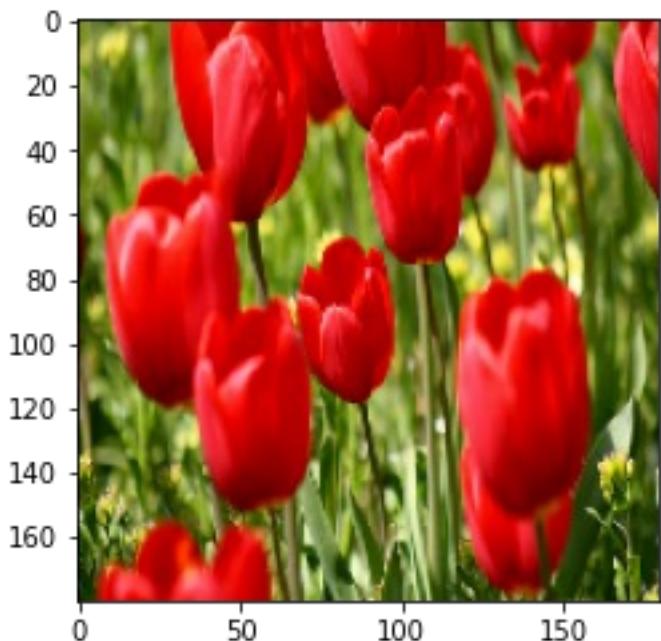
- Пытаемся сделать так, чтобы масштаб всех выходов был примерно одинаковым

Аугментация

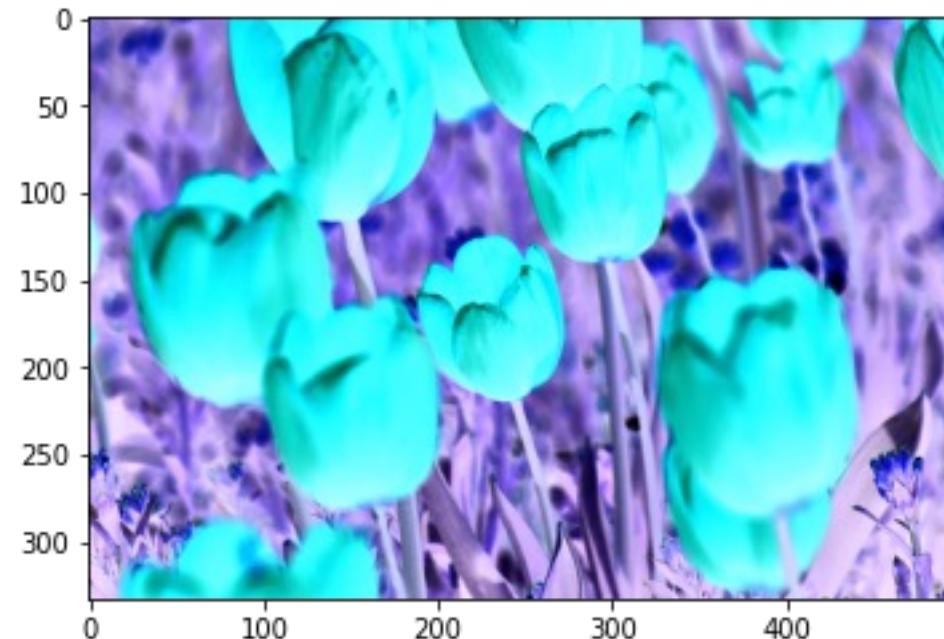
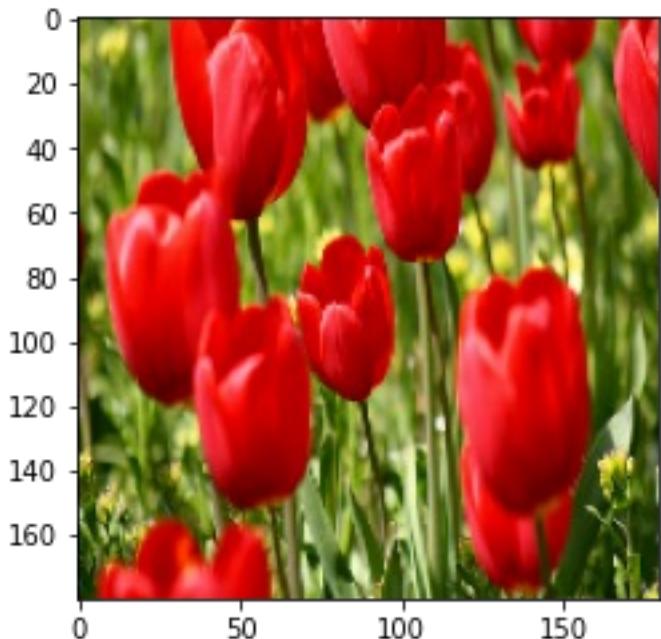
Аугментация



Аугментация



Аугментация



https://www.tensorflow.org/tutorials/images/data_augmentation



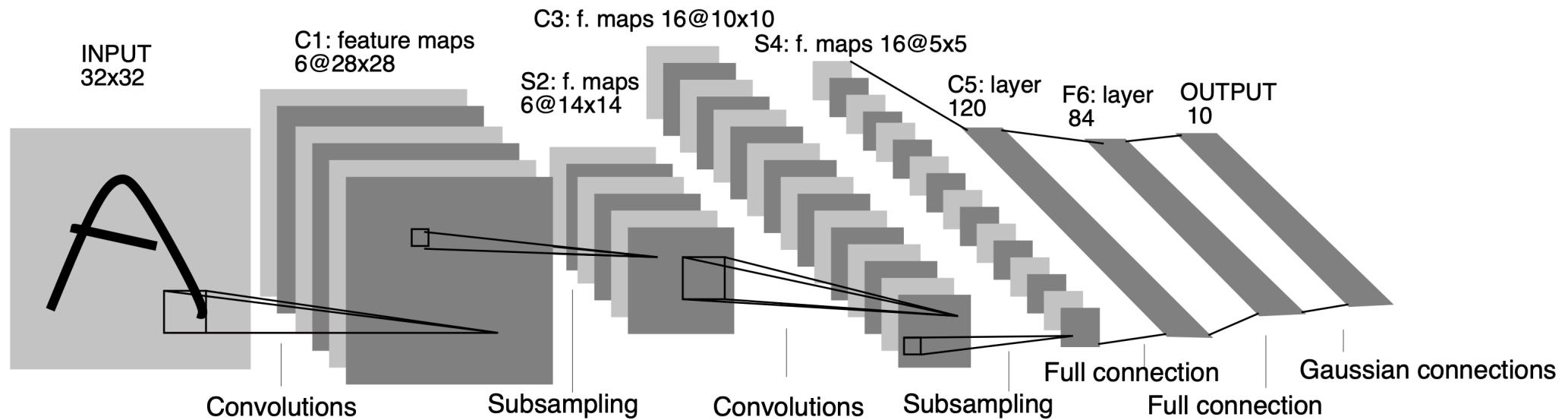
Аугментация

- Много разных вариантов
- «Бесплатное» расширение обучающей выборки
- В некотором смысле регуляризация модели

- Обычно аугментации случайно применяют к картинкам из текущего батча
- На этапе применения можно сделать несколько аугментаций картинки, применить сеть к каждой, усреднить предсказания

Архитектуры свёрточных сетей

LeNet (1998)



LeNet (1998)

- Для данных MNIST
- Идея end-to-end обучения
- Использовали аугментацию
- Около 60.000 параметров
- Доля ошибок на тесте 0.8%

ImageNet



- ImageNet Large Scale Visual Recognition Challenge (ILSVRC)
- Около 1.000.000 изображений
- 1000 классов
- Обычно качество измерялось на основе лучшей гипотезы модели

AlexNet (2012)

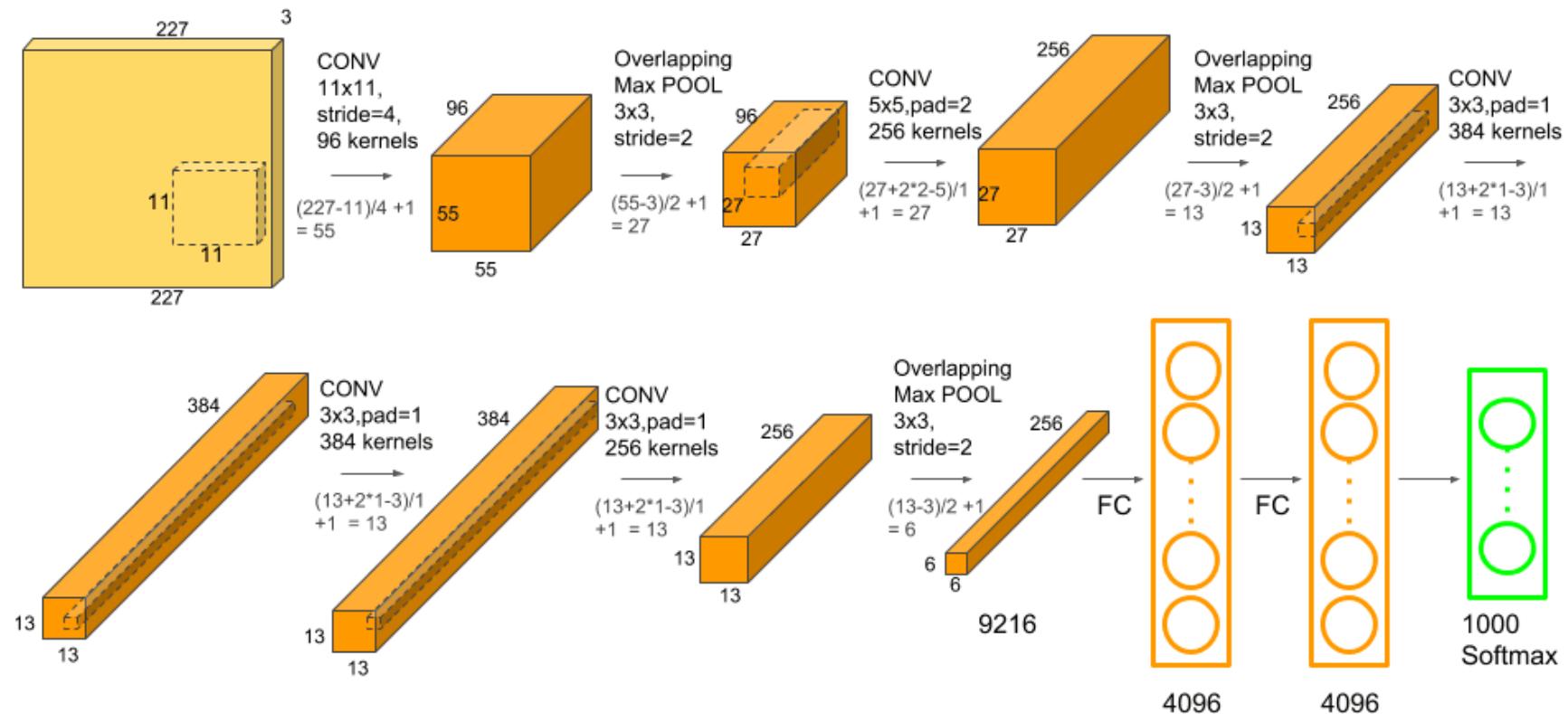
ImageNet Classification with Deep Convolutional Neural Networks

Alex Krizhevsky
University of Toronto
`kriz@cs.utoronto.ca`

Ilya Sutskever
University of Toronto
`ilya@cs.utoronto.ca`

Geoffrey E. Hinton
University of Toronto
`hinton@cs.utoronto.ca`

AlexNet (2012)



AlexNet (2012)

- Используют ReLU, аугментацию, dropout
- Градиентный спуск с инерцией (momentum)
- Обучение на двух GPU (5-6 суток)
- Около 60 миллионов параметров
- Ошибка около 17%

VGG (2014)

VERY DEEP CONVOLUTIONAL NETWORKS FOR LARGE-SCALE IMAGE RECOGNITION

Karen Simonyan* & **Andrew Zisserman⁺**

Visual Geometry Group, Department of Engineering Science, University of Oxford
`{karen,az}@robots.ox.ac.uk`

VGG (2014)

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 LRN	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 conv1-256	conv3-256 conv3-256 conv3-256	conv3-256 conv3-256 conv3-256
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 conv1-512	conv3-512 conv3-512 conv3-512	conv3-512 conv3-512 conv3-512
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 conv1-512	conv3-512 conv3-512 conv3-512	conv3-512 conv3-512 conv3-512
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

VGG (2014)

Table 2: Number of parameters (in millions).

Network	A,A-LRN	B	C	D	E
Number of parameters	133	133	134	138	144

VGG (2014)

- Только маленькие свёртки
 - Меньше параметров
 - Больше нелинейностей (т.к. больше свёрточных слоёв)
 - Больше параметров в свёрточных слоях
- Градиентный спуск с инерцией
- Dropout для двух первых полносвязных слоёв
- Хитрая инициализация (сначала обучается вариант A со случайными начальными весами, потом им инициализируются более глубокие сети)

VGG (2014)

Table 3: ConvNet performance at a single test scale.

ConvNet config. (Table 1)	smallest image side		top-1 val. error (%)	top-5 val. error (%)
	train (S)	test (Q)		
A	256	256	29.6	10.4
A-LRN	256	256	29.7	10.5
B	256	256	28.7	9.9
C	256	256	28.1	9.4
	384	384	28.1	9.3
	[256;512]	384	27.3	8.8
D	256	256	27.0	8.8
	384	384	26.8	8.7
	[256;512]	384	25.6	8.1
E	256	256	27.3	9.0
	384	384	26.9	8.7
	[256;512]	384	25.5	8.0

GoogLeNet (2014)

Going Deeper with Convolutions

Christian Szegedy¹, Wei Liu², Yangqing Jia¹, Pierre Sermanet¹, Scott Reed³,
Dragomir Anguelov¹, Dumitru Erhan¹, Vincent Vanhoucke¹, Andrew Rabinovich⁴

¹Google Inc. ²University of North Carolina, Chapel Hill

³University of Michigan, Ann Arbor ⁴Magic Leap Inc.

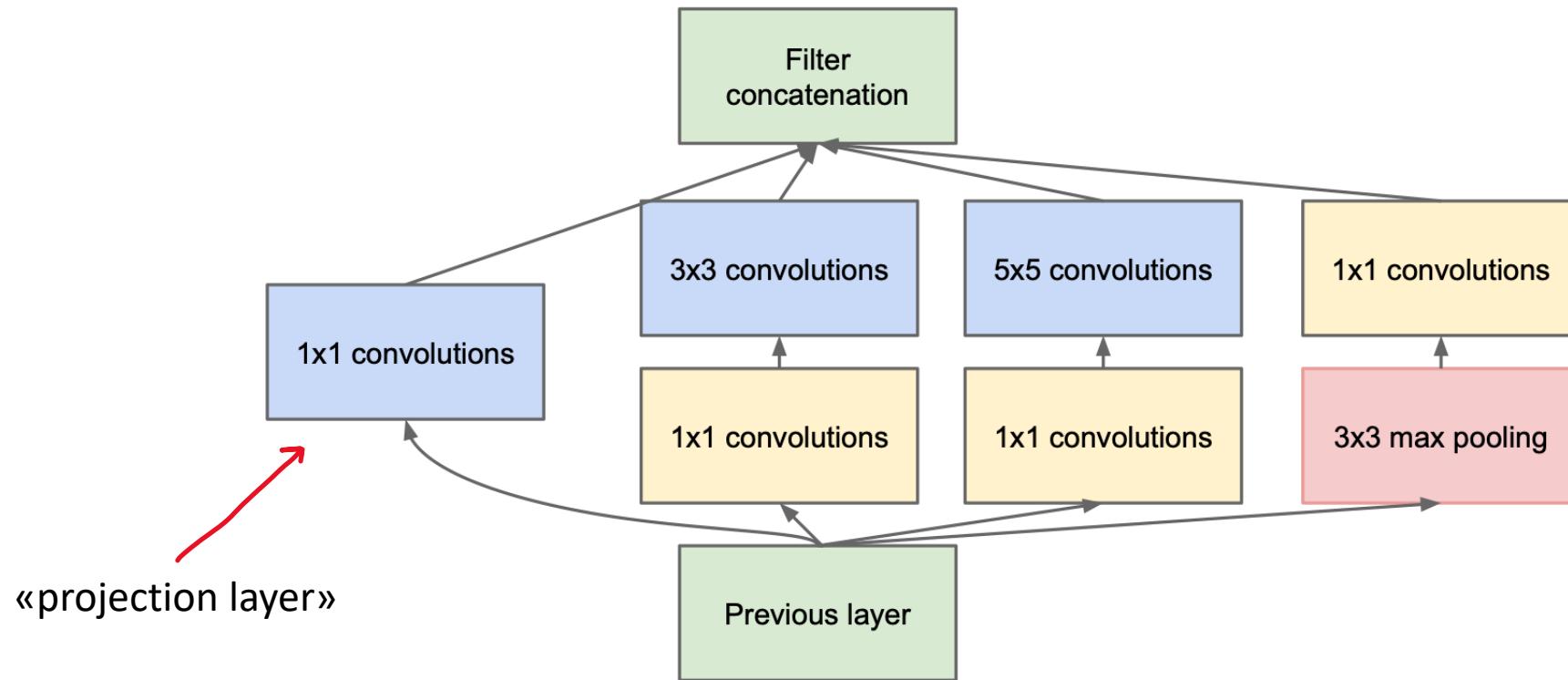
¹{szegedy,jiayq,sermanet,dragomir,dumitru,vanhoucke}@google.com

²wliu@cs.unc.edu, ³reedscott@umich.edu, ⁴arabinovich@magineleap.com



Figure 3: GoogLeNet network with all the bells and whistles.

GoogLeNet (2014)



(b) Inception module with dimensionality reduction

свёртки делаются с паддингом!

<http://arxiv.org/abs/1409.4842>

GoogLeNet (2014)

- Снижается число каналов перед «тяжёлыми» свёртками
- Несколько выходных слоёв для улучшения обучаемости
- Обучается градиентным спуском с инерцией
- Ошибка 6.67% на ImageNet

ResNet (2015)

Deep Residual Learning for Image Recognition

Kaiming He

Xiangyu Zhang

Shaoqing Ren

Jian Sun

Microsoft Research

{kahe, v-xiangz, v-shren, jiansun}@microsoft.com

ResNet (2015)

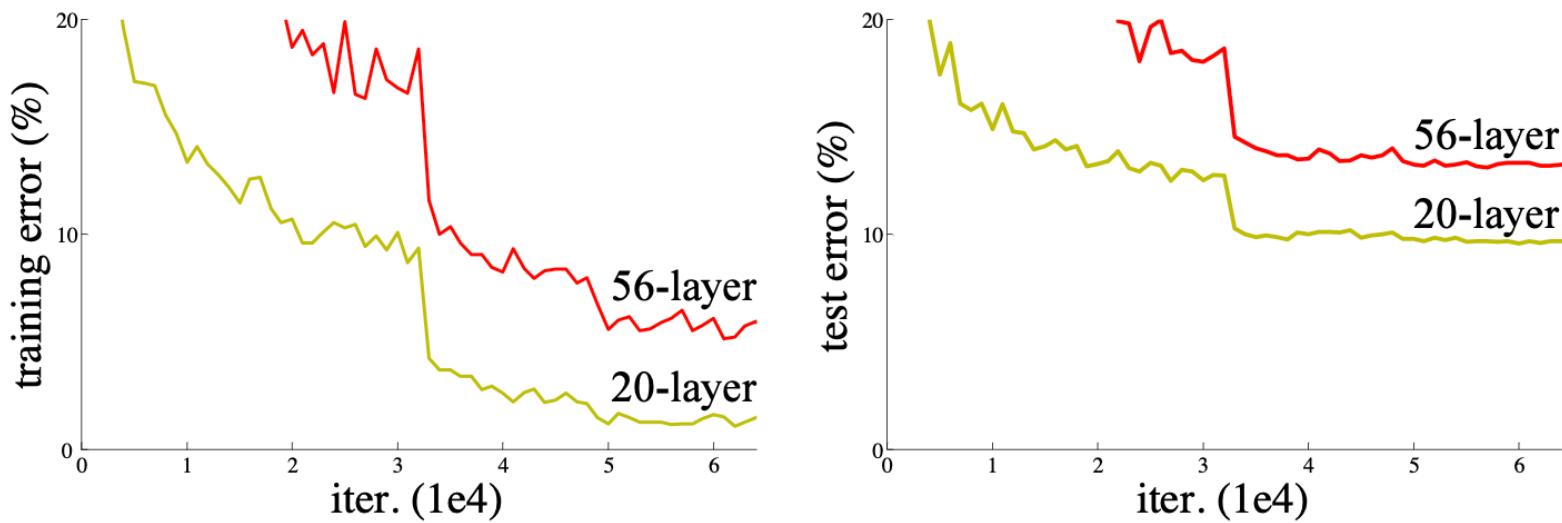
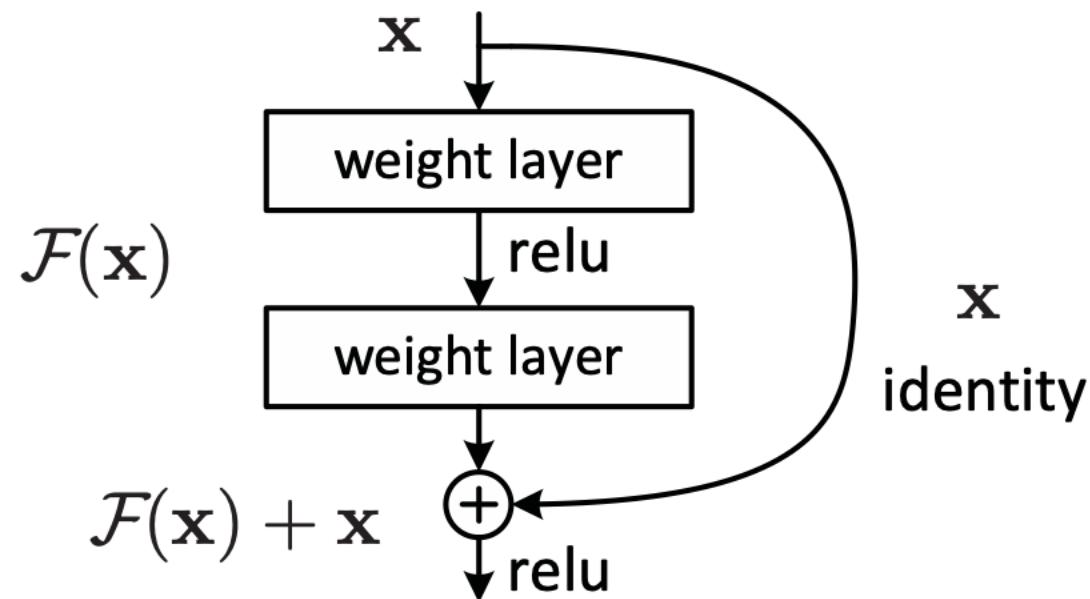


Figure 1. Training error (left) and test error (right) on CIFAR-10 with 20-layer and 56-layer “plain” networks. The deeper network has higher training error, and thus test error. Similar phenomena on ImageNet is presented in Fig. 4.

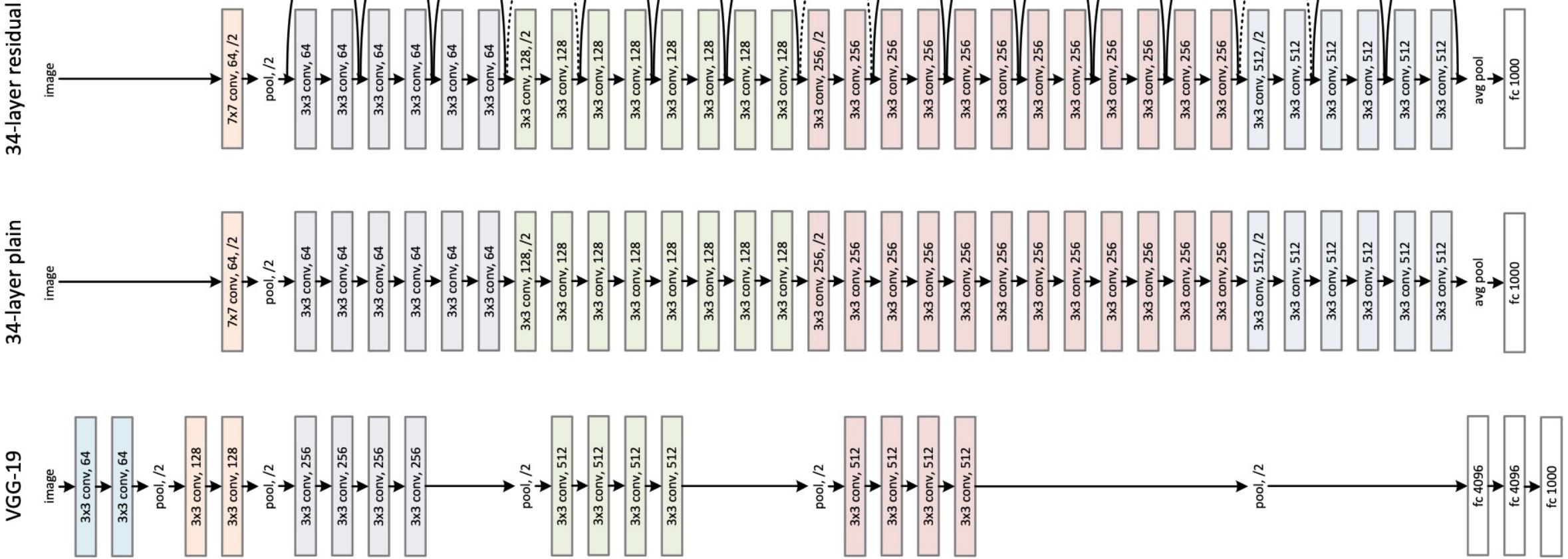
ResNet (2015)

- Добавление слоёв в свёрточную сеть ухудшает качество даже на обучении
- Хотя возможностей для переобучения больше, сеть почему-то не может ими воспользоваться

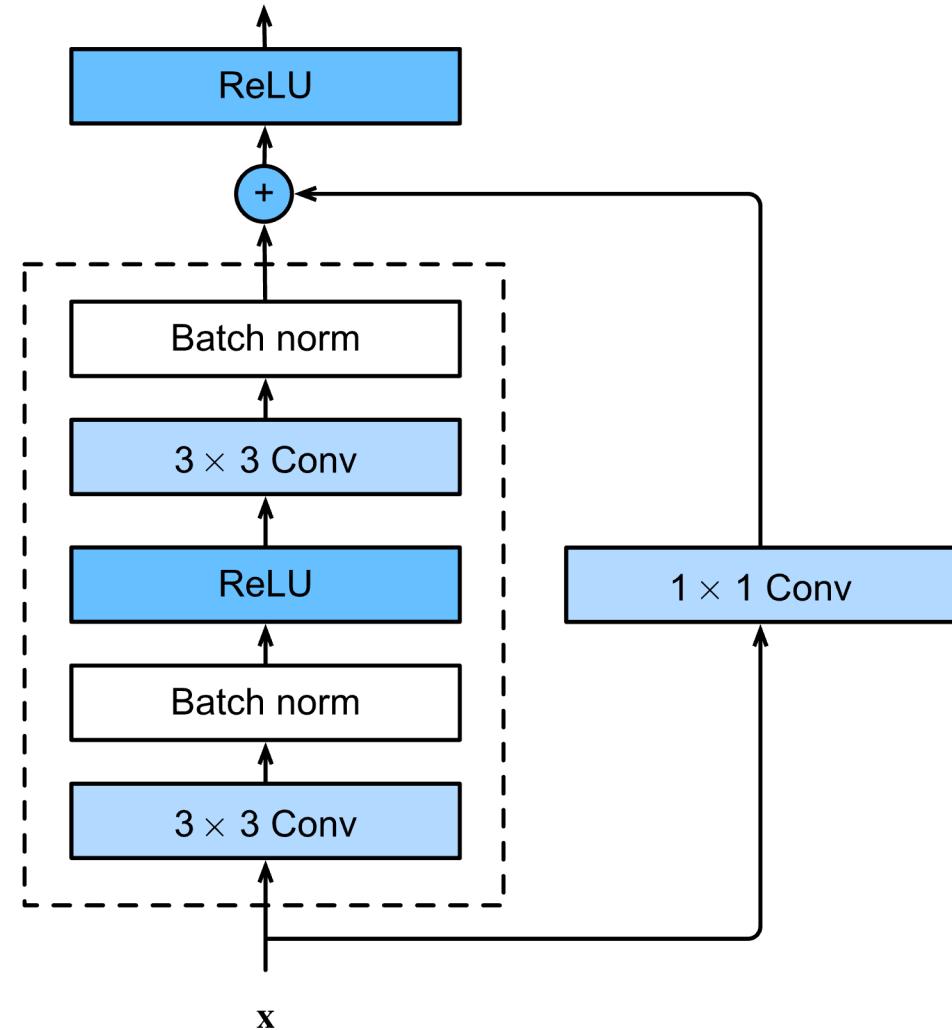
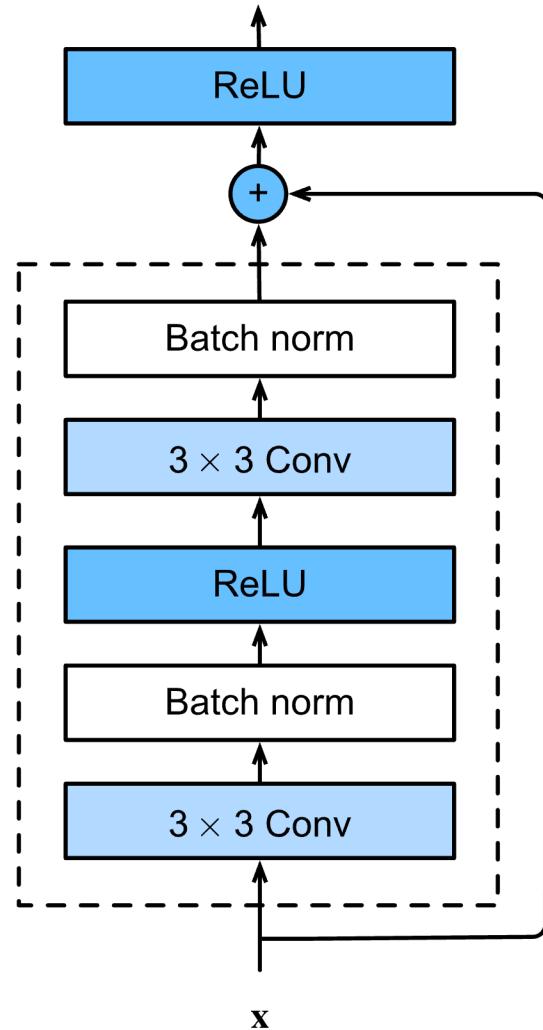
ResNet (2015)



ResNet (2015)



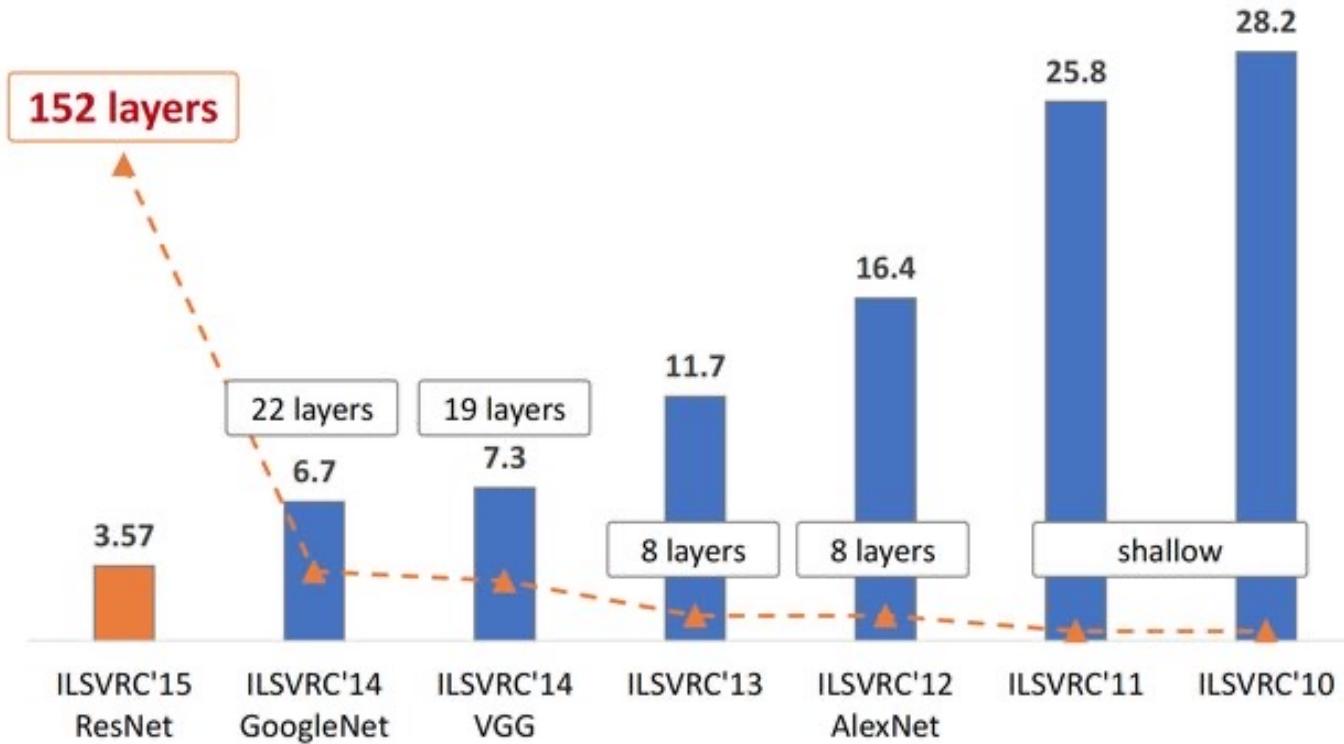
ResNet block



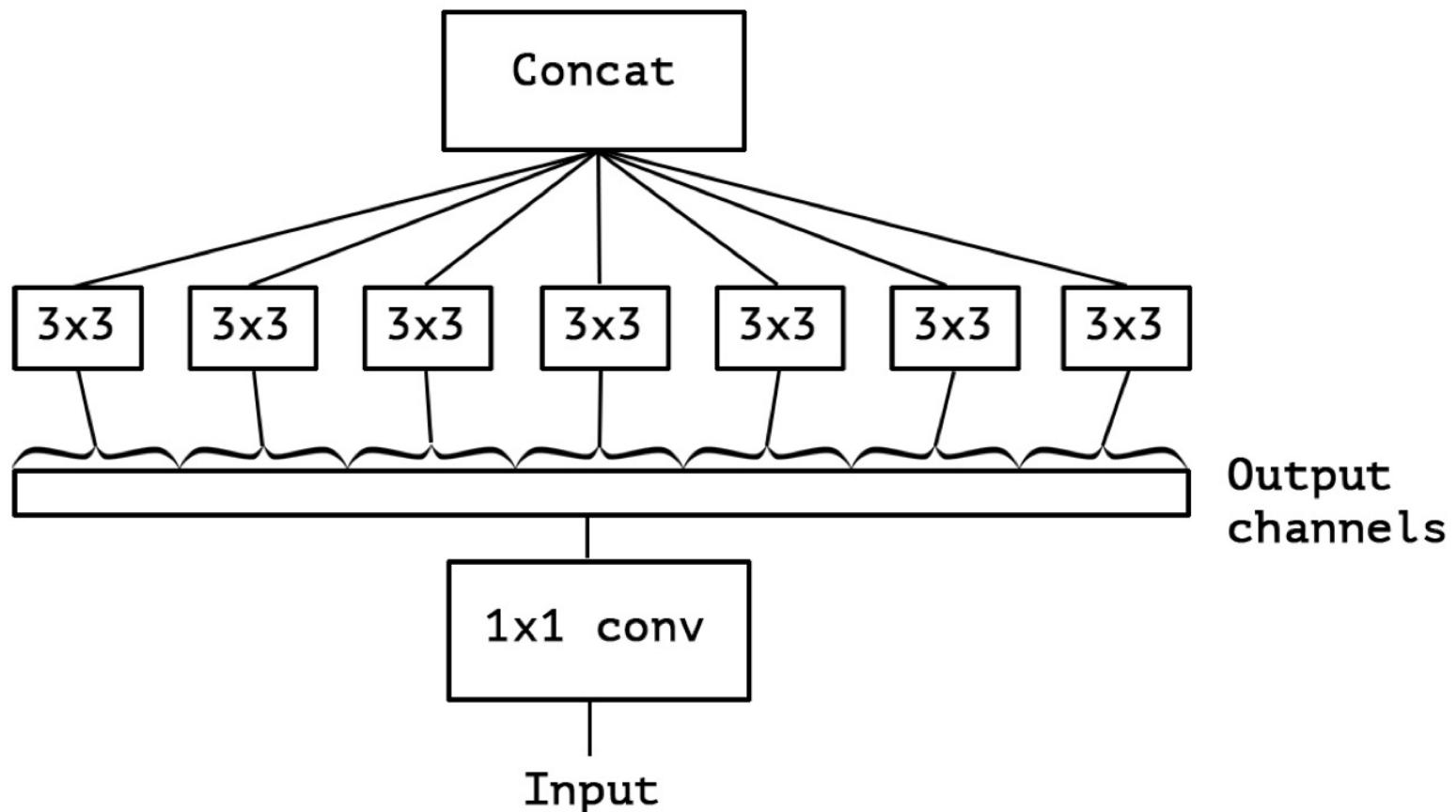
ResNet (2015)

- Даёт низкую ошибку на обучении даже с 1000 слоёв (но там плохо на тестовой выборке)
- Обучается градиентным спуском с инерцией со случайной инициализацией
- Global Average Pooling — позволяет работать с изображениями разного размера
- Ошибка 4.49% на ImageNet

Эволюция архитектур



Xception



Xception

- Разделяется роль свёрток: либо по каналам, либо по пространству
- Более эффективное использование параметров

Что ещё?

- Highway networks
- Inception-ResNet
- Squeeze and Excitation Network
- MobileNet
- EfficientNet
- ...