

Глубокое обучение

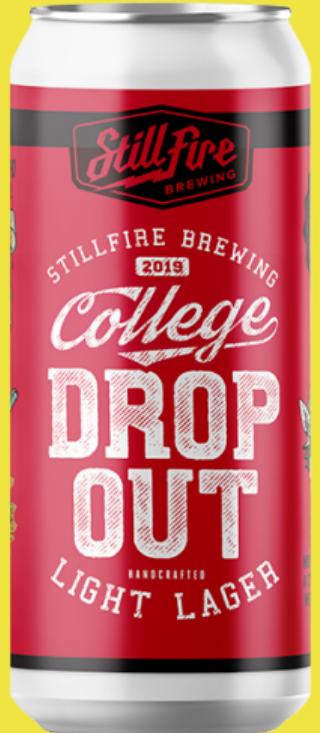
Дмитрий Никулин

12 мая 2021 г.

Неделя 7: Свёрточные нейросети (часть 2)

Agenda

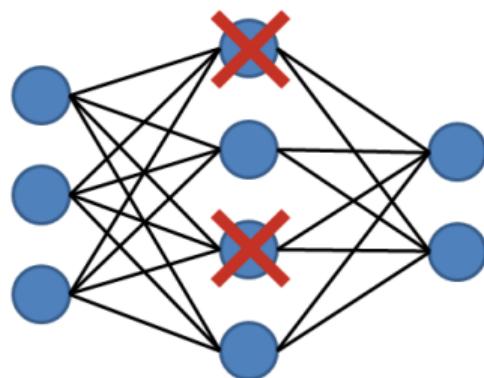
- Что ещё можно делать с архитектурами нейросетей
 - Dropout
 - Batch normalization
 - Skip connections
- Архитектуры
 - LeNet
 - AlexNet
 - VGG
 - GoogLeNet / Inception
 - ResNet
- Обучаем свою свёрточную нейронку





Dropout

- С вероятностью p зануляем активацию, иначе не трогаем
- Делает нейроны более устойчивыми к случайным возмущениям
- Борьба с ко-адаптацией (нейрон должен работать, даже если его соседи выдают мусор)



Dropout в формулах

- Forward pass:

h — активации предыдущего слоя

Dropout в формулах

- Forward pass:

h — активации предыдущего слоя

$$o = D \cdot h, \quad D = (D_1, \dots, D_k) \sim i.i.d. \text{Bernoulli}(p)$$

Dropout в формулах

- Forward pass:

h — активации предыдущего слоя

$$o = D \cdot h, \quad D = (D_1, \dots, D_k) \sim i.i.d. \text{Bernoulli}(p)$$

$$o_i = D_i \cdot h_i = \begin{cases} h_i, & 1 - p \\ 0, & p \end{cases}$$

Dropout — это просто домножение на случайный тензор из нулей и единиц.

Dropout в формулах

- Forward pass:

h — активации предыдущего слоя

$$o = D \cdot h, \quad D = (D_1, \dots, D_k) \sim i.i.d. \text{Bernoulli}(p)$$

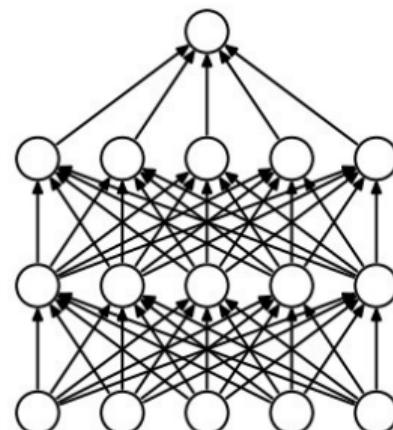
- Backward pass:

$$\frac{\partial L}{\partial h} = \frac{\partial L}{\partial o} \cdot D$$

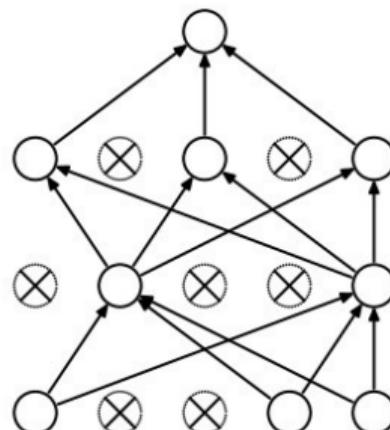
$$\left[\frac{\partial L}{\partial h} \right]_i = \left[\frac{\partial L}{\partial o} \right]_i \cdot D_i$$

Dropout

- При обучении мы домножаем часть выходов на D_i , тем самым мы изменяем только часть параметров и нейроны учатся более независимо
⇒ регуляризация
- Dropout в некотором смысле эквивалентен усреднению результатов обучения 2^n сетей



(a) Standard Neural Net



(b) After applying dropout.

Dropout

- При обучении мы домножаем часть выходов на D_i , тем самым мы изменяем только часть параметров и нейроны учатся более независимо
⇒ регуляризация
- Dropout в некотором смысле эквивалентен усреднению результатов обучения 2^n сетей
- Что делать на стадии тестирования?

Dropout

- При обучении мы домножаем часть выходов на D_i , тем самым мы изменяем только часть параметров и нейроны учатся более независимо
⇒ регуляризация
- Dropout в некотором смысле эквивалентен усреднению результатов обучения 2^n сетей
- Нам надо сымитировать работу такого ансамбля: можно отключать по очереди все возможные комбинации нейронов, получить 2^n прогнозов и усреднить их

Dropout

- При обучении мы домножаем часть выходов на D_i , тем самым мы изменяем только часть параметров и нейроны учатся более независимо
⇒ регуляризация
- Dropout в некотором смысле эквивалентен усреднению результатов обучения 2^n сетей
- Нам надо сымитировать работу такого ансамбля: можно отключать по очереди все возможные комбинации нейронов, получить 2^n прогнозов и усреднить их
- Но лучше просто брать по дропауту математическое ожидание

$$o = (1 - p) \cdot h$$

Обратный Dropout

- На тесте ищем математическое ожидание:

$$o = (1 - p) \cdot h$$

Обратный Dropout

- На тесте ищем математическое ожидание:

$$o = (1 - p) \cdot h$$

- Это не очень удобно.

Обратный Dropout

- На тесте ищем математическое ожидание:

$$o = (1 - p) \cdot h$$

- Это не очень удобно.
- Давайте лучше будем домножать на $\frac{1}{1-p}$ на этапе обучения:

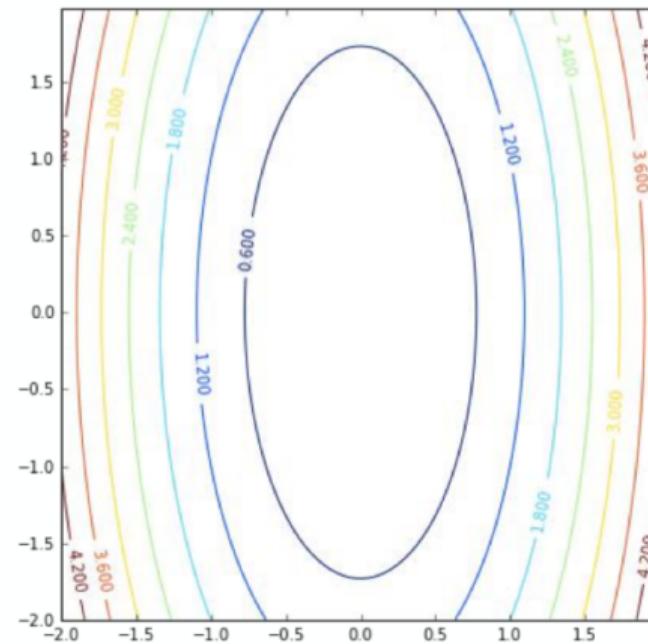
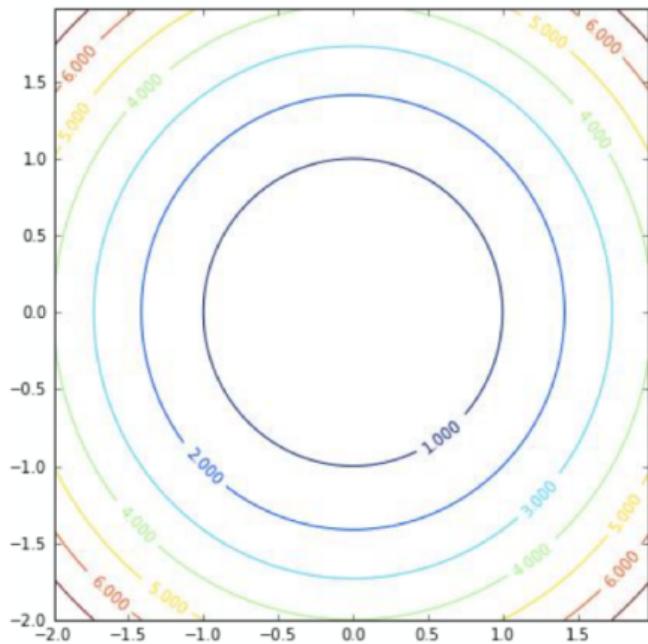
train:
$$o = \frac{1}{1-p} \cdot D \cdot h$$

test:
$$o = h$$

Батч-нормализация

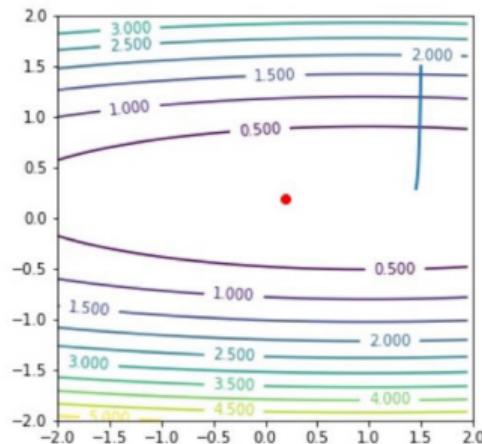
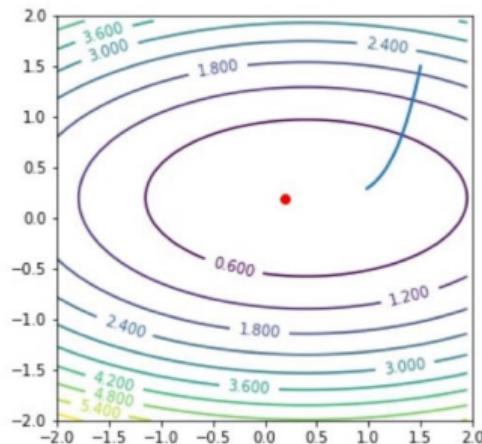
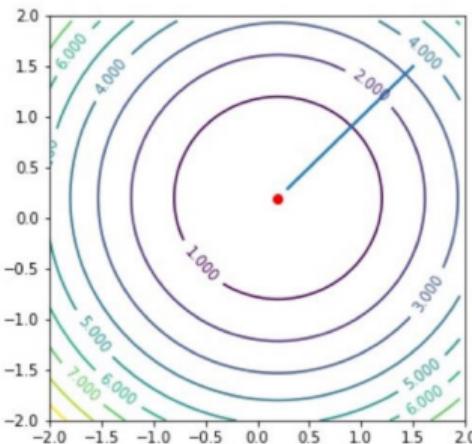


Стандартизация



Какая из ситуаций лучше для SGD?

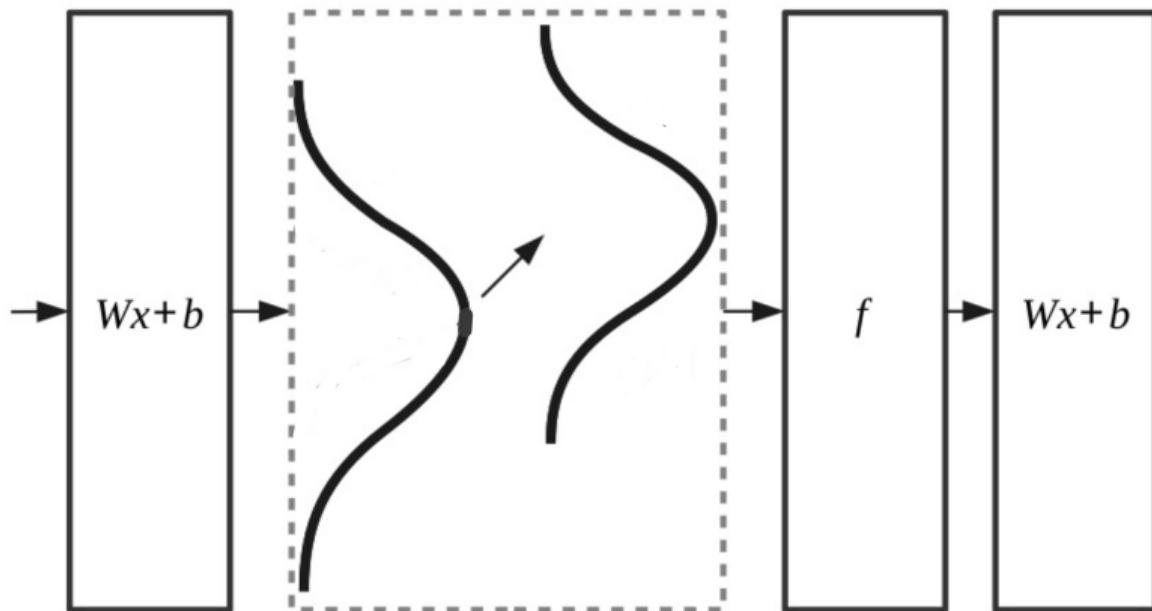
Стандартизация



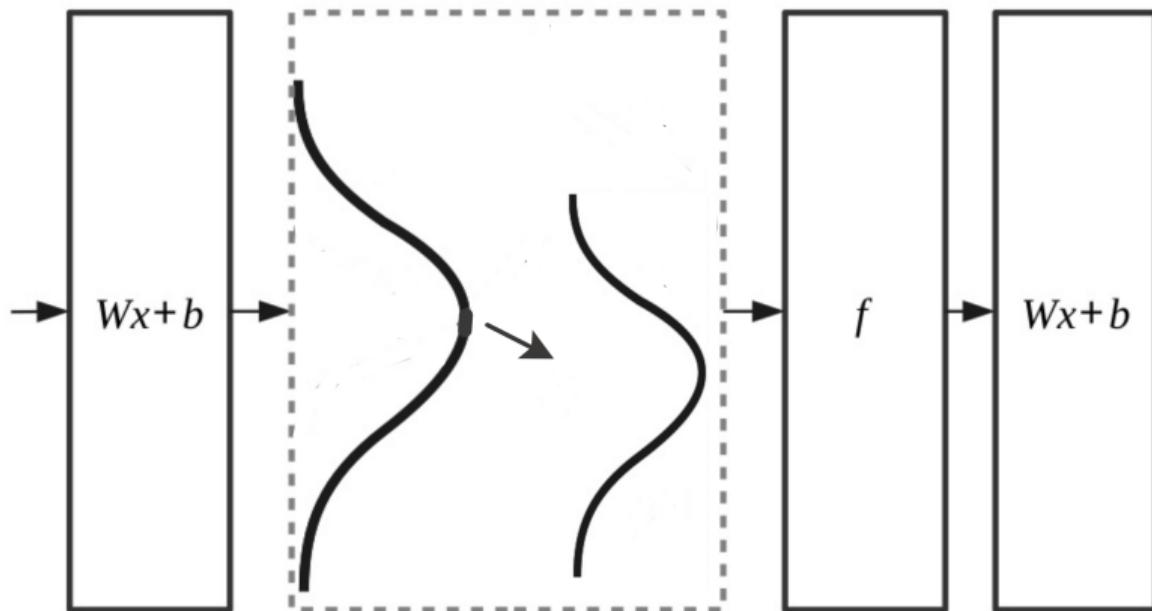
Градиентные методы быстрее сходятся, если градиенты в одном масштабе

[https://github.com/hse-aml/intro-to-dl/blob/master/week2/v2/
ill-conditioned-demo.ipynb](https://github.com/hse-aml/intro-to-dl/blob/master/week2/v2/ill-conditioned-demo.ipynb)

А что внутри нейросетки?



А что внутри нейросетки?



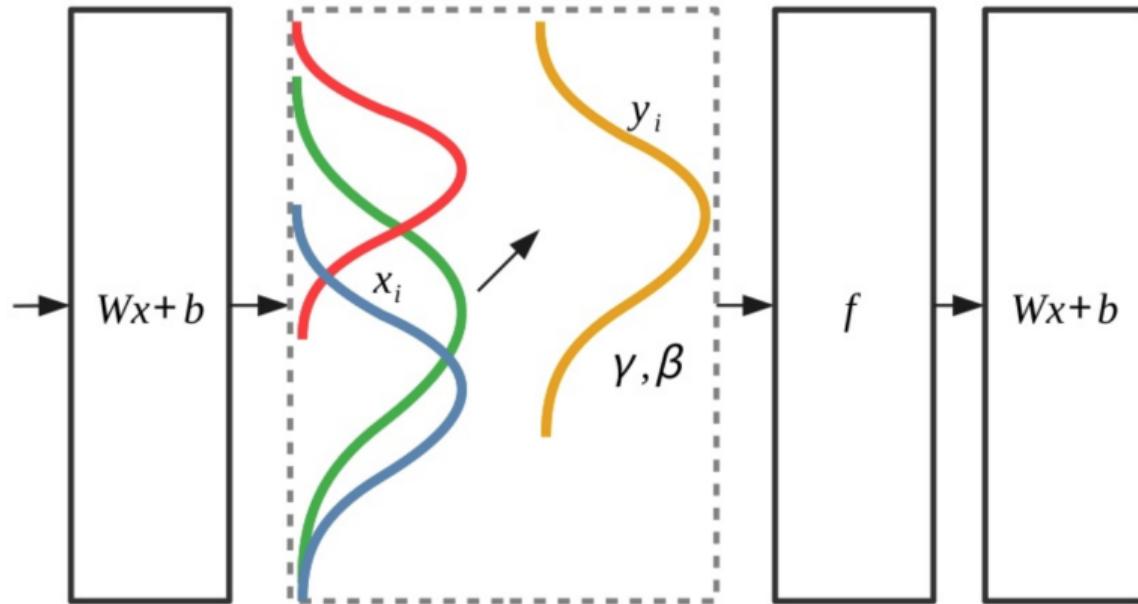
Проблема

- Давайте вместо X на входе использовать $\frac{X - \mu_X}{\sigma_X}$
- Даже если мы стандартизовали вход X , внутри сетки может произойти несчастье и скрытый слой окажется нестандартизован
- Скрытые представления $h = f(XW)$ могут менять своё распределение в процессе обучения, это усложняет его

Проблема

- Давайте вместо X на входе использовать $\frac{X - \mu_X}{\sigma_X}$
- Даже если мы стандартизовали вход X , внутри сетки может произойти несчастье и скрытый слой окажется нестандартизован
- Скрытые представления $h = f(XW)$ могут менять своё распределение в процессе обучения, это усложняет его
- Давайте на каждом слое вместо h использовать $\hat{h} = \frac{h - \mu_h}{\sigma_h}$
- На выход будем выдавать $\beta \cdot \hat{h} + \gamma$, для того, чтобы у нас было больше свободы, параметры β и γ тоже учим

Batch norm (2015)



Forward pass

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1\dots m}\}$;

Parameters to be learned: γ, β

Output: $\{y_i = \text{BN}_{\gamma, \beta}(x_i)\}$

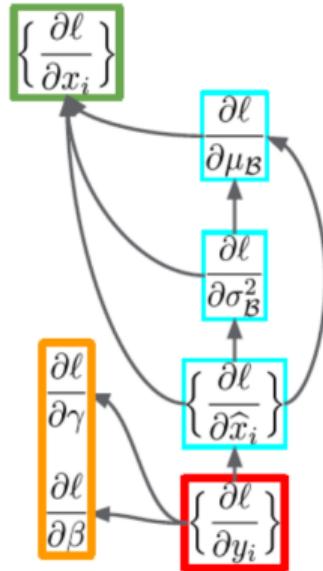
$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m} \sum_{i=1}^m x_i \quad // \text{mini-batch mean}$$

$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m} \sum_{i=1}^m (x_i - \mu_{\mathcal{B}})^2 \quad // \text{mini-batch variance}$$

$$\hat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \quad // \text{normalize}$$

$$y_i \leftarrow \gamma \hat{x}_i + \beta \equiv \text{BN}_{\gamma, \beta}(x_i) \quad // \text{scale and shift}$$

Backward pass



$$\frac{\partial \ell}{\partial \hat{x}_i} = \frac{\partial \ell}{\partial y_i} \cdot \gamma$$

$$\frac{\partial \ell}{\partial \sigma_B^2} = \sum_{i=1}^m \frac{\partial \ell}{\partial \hat{x}_i} \cdot (x_i - \mu_B) \cdot \frac{-1}{2} (\sigma_B^2 + \epsilon)^{-3/2}$$

$$\frac{\partial \ell}{\partial \mu_B} = \left(\sum_{i=1}^m \frac{\partial \ell}{\partial \hat{x}_i} \cdot \frac{-1}{\sqrt{\sigma_B^2 + \epsilon}} \right) + \frac{\partial \ell}{\partial \sigma_B^2} \cdot \frac{\sum_{i=1}^m -2(x_i - \mu_B)}{m-1}$$

$$\frac{\partial \ell}{\partial x_i} = \frac{\partial \ell}{\partial \hat{x}_i} \cdot \frac{1}{\sqrt{\sigma_B^2 + \epsilon}} + \frac{\partial \ell}{\partial \sigma_B^2} \cdot \frac{2(x_i - \mu_B)}{m-1} + \frac{\partial \ell}{\partial \mu_B} \cdot \frac{1}{m}$$

$$\frac{\partial \ell}{\partial \gamma} = \sum_{i=1}^m \frac{\partial \ell}{\partial y_i} \cdot \hat{x}_i$$

$$\frac{\partial \ell}{\partial \beta} = \sum_{i=1}^m \frac{\partial \ell}{\partial y_i}$$

Batch norm (2015)

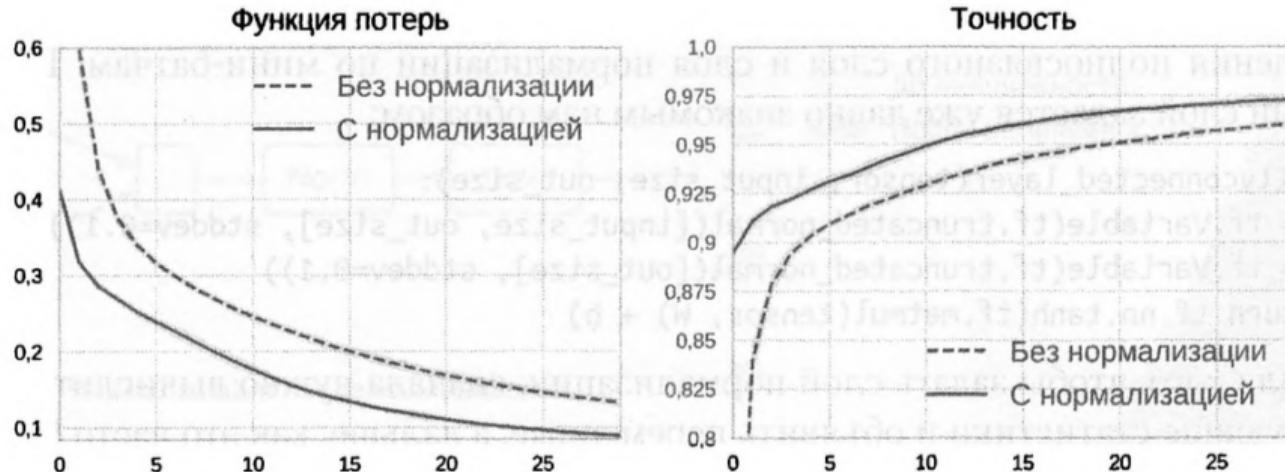
- Как считать μ_h и σ_h ?
- Оценить по текущему батчу!

$$\mu_h = \alpha \cdot \bar{x}_{batch} + (1 - \alpha) \cdot \mu_h$$
$$\sigma_h = \alpha \cdot \hat{s}_{batch} + (1 - \alpha) \cdot \sigma_h$$

- Коэффициенты β и γ оцениваются в ходе обратного распространения ошибки, μ и σ не обучаются
- Обучение довольно сильно ускоряется, сходимость улучшается

<https://arxiv.org/pdf/1502.03167.pdf>

Эксперимент с MNIST



Источник: Николенко, страница 160

Почему это помогает при обучении

How Does Batch Normalization Help Optimization?

Shibani Santurkar*
MIT
shibani@mit.edu

Dimitris Tsipras*
MIT
tsipras@mit.edu

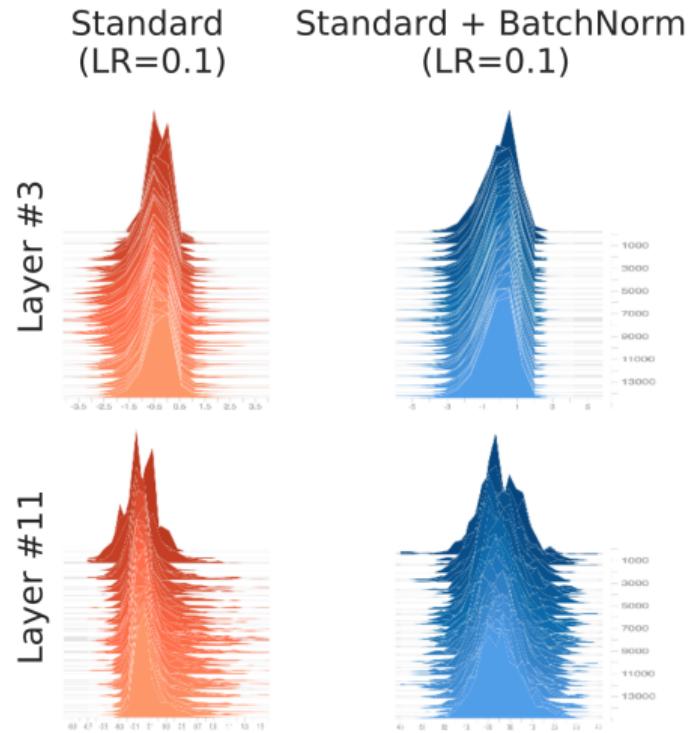
Andrew Ilyas*
MIT
ailyas@mit.edu

Aleksander Mądry
MIT
madry@mit.edu

Abstract

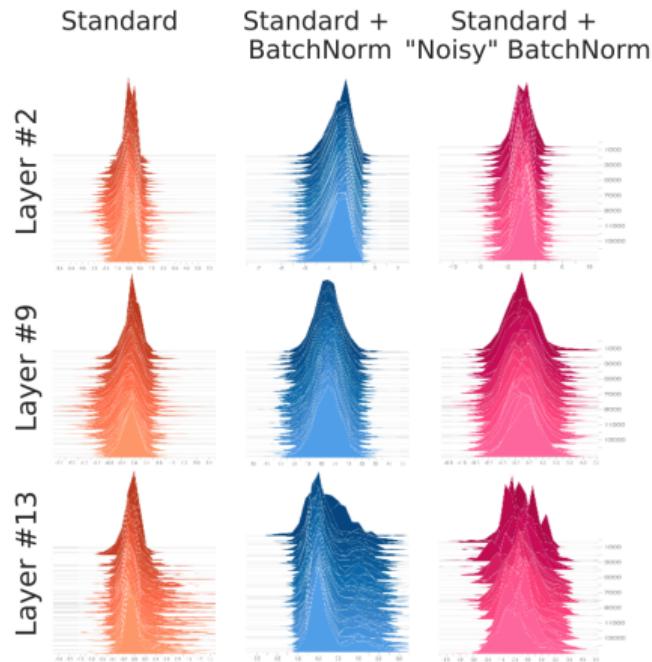
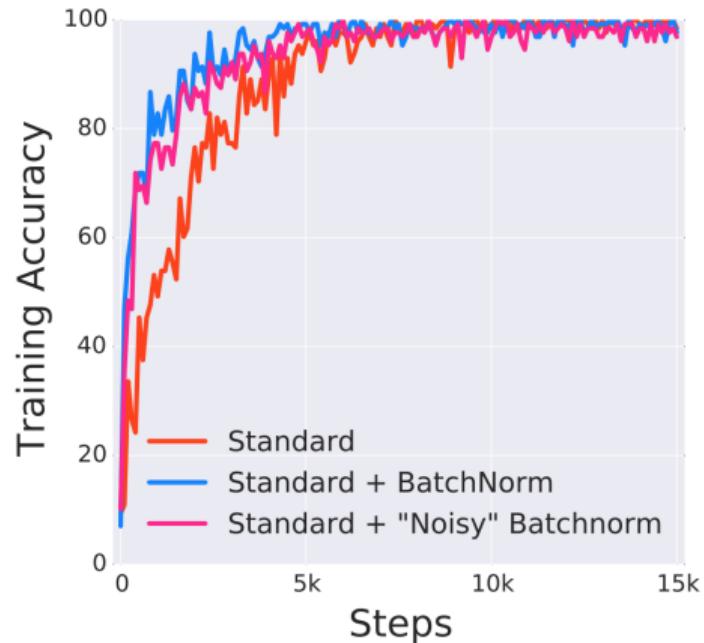
Batch Normalization (BatchNorm) is a widely adopted technique that enables faster and more stable training of deep neural networks (DNNs). Despite its pervasiveness, the exact reasons for BatchNorm's effectiveness are still poorly understood. The popular belief is that this effectiveness stems from controlling the change of the layers' input distributions during training to reduce the so-called "internal covariate shift". In this work, we demonstrate that such distributional stability of layer inputs has little to do with the success of BatchNorm. Instead, we uncover a more fundamental impact of BatchNorm on the training process: it makes the optimization landscape significantly smoother. This smoothness induces a more predictive and stable behavior of the gradients, allowing for faster training.

Почему это помогает при обучении



<https://arxiv.org/abs/1805.11604>

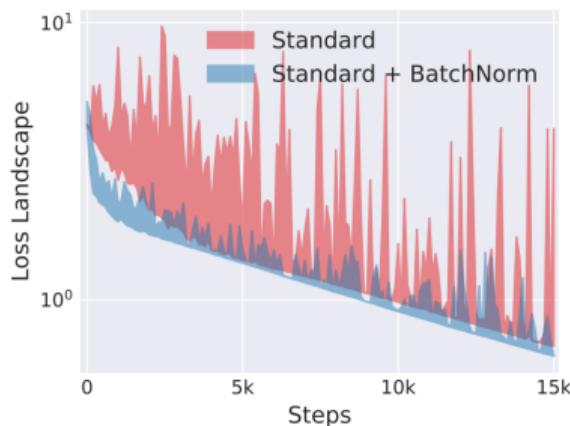
Почему это помогает при обучении



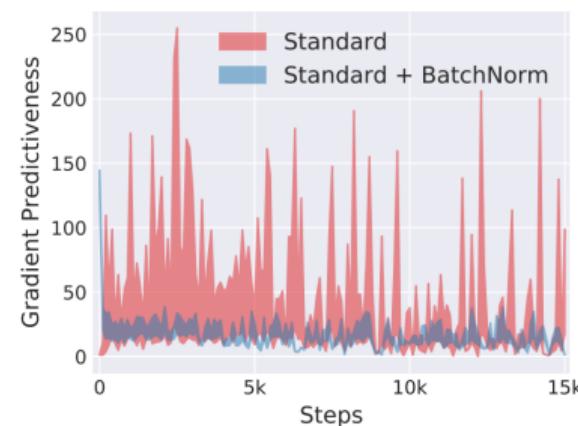
<https://arxiv.org/abs/1805.11604>

Почему это помогает при обучении

Батчнорм сглаживает ландшафт функции потерь и из-за этого градиентный спуск идёт более гладко.



(a) loss landscape



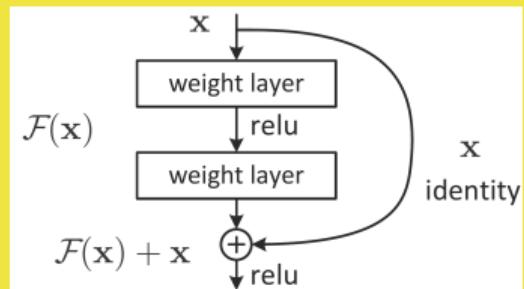
(b) gradient predictiveness

Трюки

- С батч-нормализацией нужно уменьшить силу Dropout и регуляризацию
- Батч-нормализация и Dropout могут конфликтовать
- Не забывайте перемешивать обучающую выборку перед каждой новой эпохой, чтобы батчи были разнообразными
- Существует довольно много техник нормализации помимо BatchNorm: InstanceNorm, LayerNorm, GroupNorm и другие
- Реализации можете посмотреть в
<https://github.com/dniki/dl-norms>

http://openaccess.thecvf.com/content_CVPR_2019/papers/Li_Understanding_the_Discrepancy_Between_Dropout_and_Batch_Normalization_by_Variance_CVPR_2019_paper.pdf

Skip connections и ResNet

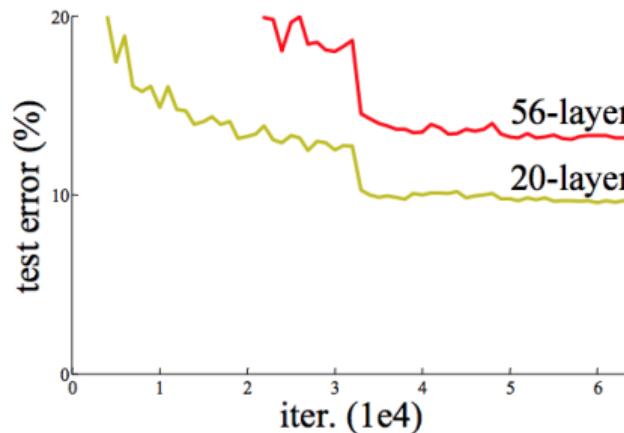
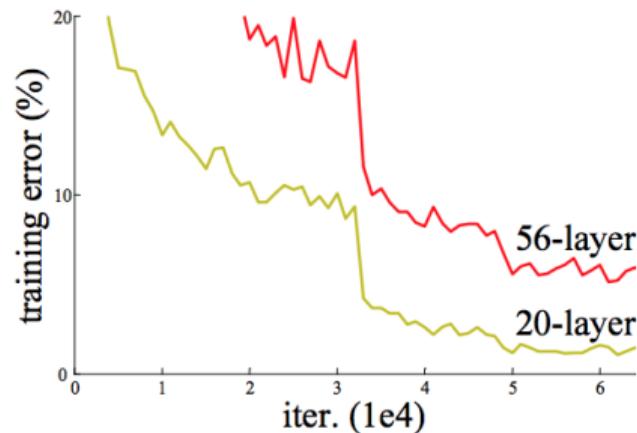


Очень глубокие сети



Идея ResNet

Чем глубже нейронная сеть, тем сложнее её обучать, возникает новая проблема, которую нельзя свести к переобучению или затуханию градиента. Её называют **деградация обучения** (training degradation)

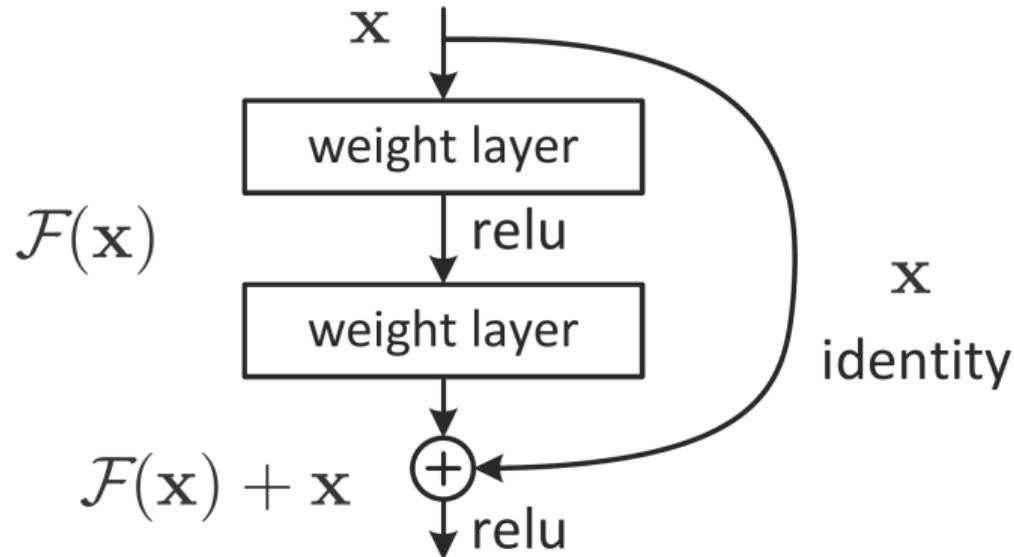


<https://arxiv.org/abs/1512.03385>

Идея ResNet

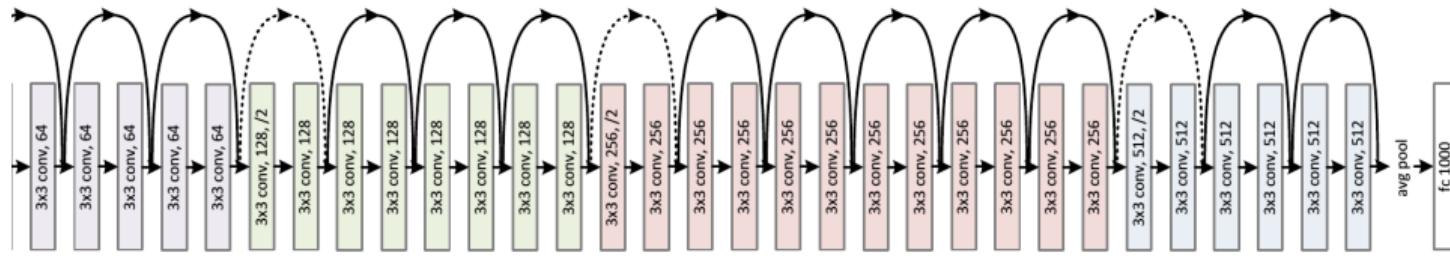
- Огромная свёрточная сеть из 20 слоёв и свёрточная сеть из 56 слоёв, большая сетка обучается хуже
- **Проблема:** слои инициализированы шумом, если какой-то один слой не натренирован, он убивает работу сети, через него не проходит полезный сигнал
- Чем больше слоёв, тем более ярко выражен этот эффект
- **Решение:** Будем посыпать вход на выход и давать слою возможность немного его подправить (residual слой)
- Идея чем-то похожа на бустинг, сеть сама решает когда заканчивать подправлять выходы (грубо говоря, сама выбирает глубину)

ResNet (2015)



<https://arxiv.org/abs/1512.03385>

ResNet (2015)



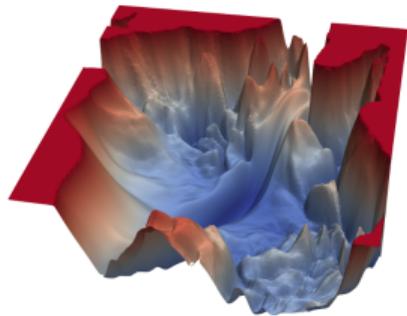
<https://arxiv.org/abs/1512.03385>

ResNet (2015)

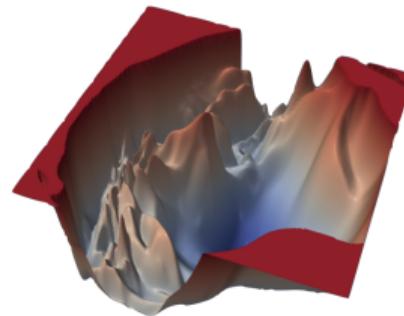
- **Идея:** более глубокие уровни должны улавливать разницу между новым и тем, что было раньше
- Ключевым элементом архитектуры является связь, которая пропускает несколько слоёв, передавая результат предыдущего слоя
- Такое изменение позволило полностью отказаться от таких техник регуляризации, как Dropout
- Градиенты не взрываются
- ResNet-архитектура ведёт себя как ансамбль неглубоких сетей:
<https://arxiv.org/abs/1605.06431>

Визуализация потерь

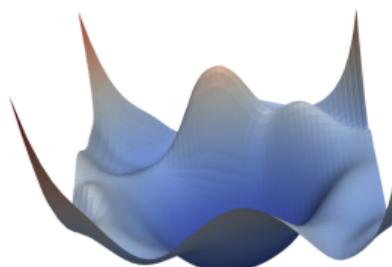
VGG-56



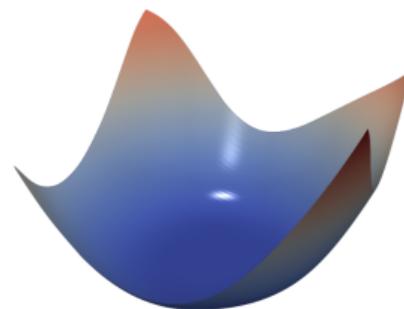
VGG-110



Renset-56



Densenet-121



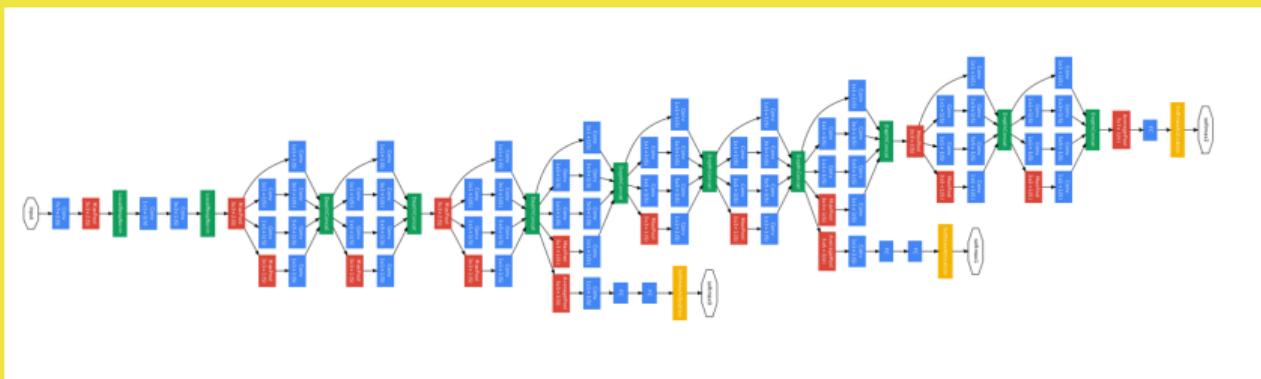
<https://arxiv.org/pdf/1712.09913.pdf>

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

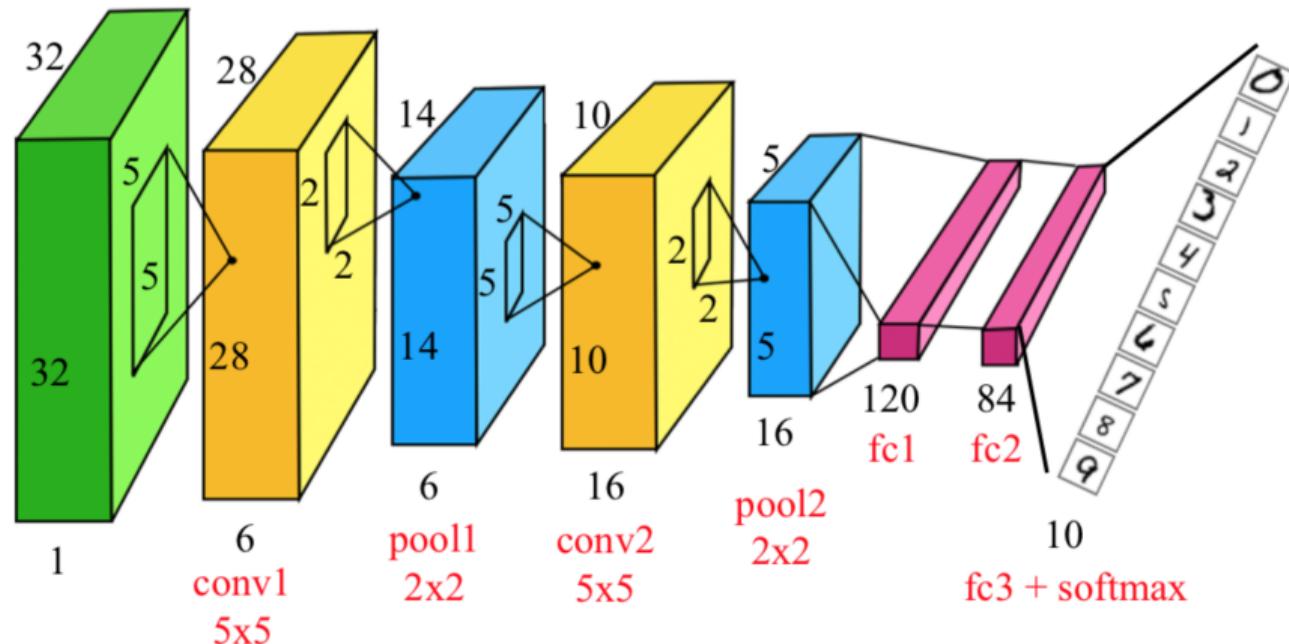
ResNet

- ResNet породил целый букет новых архитектур
- Сегодня эта идея активно используется в рекуррентных сетках и трансформерах

Архитектуры

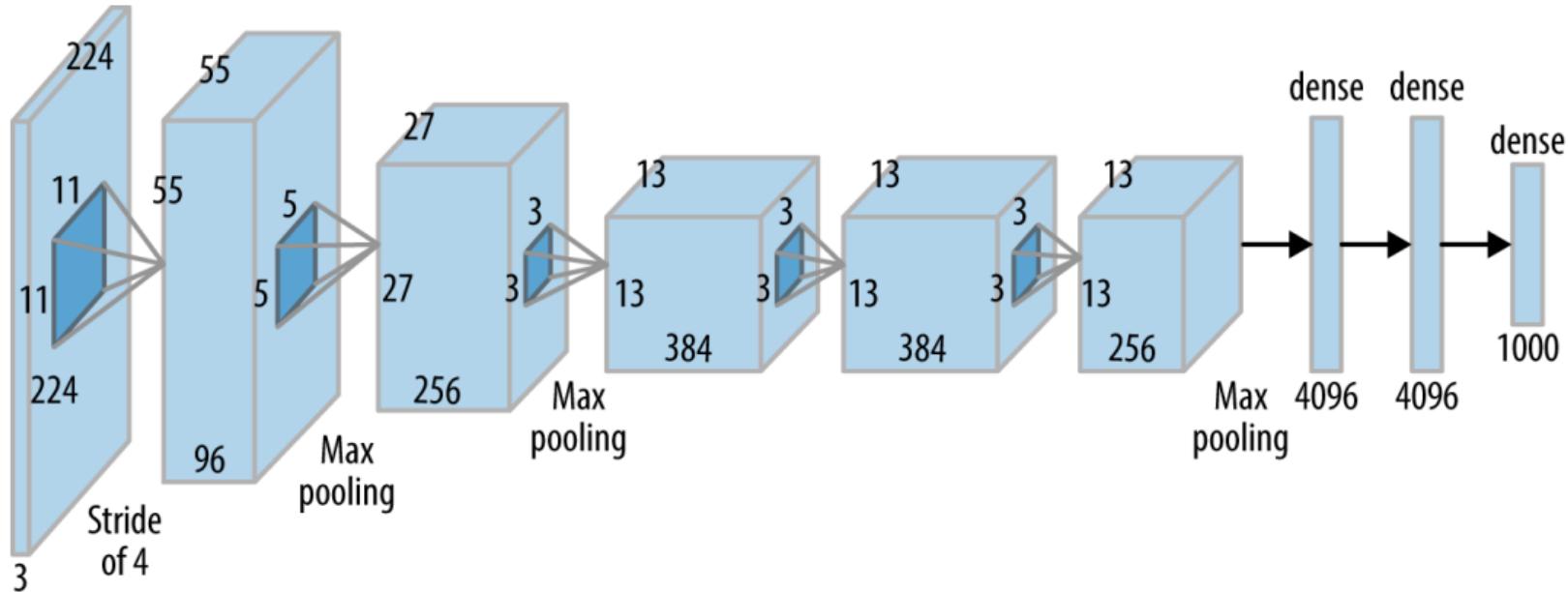


LeNet-5 (1998)



Статья: <http://yann.lecun.com/exdb/publis/pdf/lecun-98.pdf>
Пример кода: <https://github.com/maorshutman/lenet5-pytorch>

AlexNet (2012)



Статья: <https://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf>

Код: <https://github.com/pytorch/vision/blob/master/torchvision/models/alexnet.py>

Иллюстрация: <https://towardsdatascience.com/the-w3h-of-alexnet-vggnet-resnet-and-inception-7baaaecc96>

AlexNet (2012)

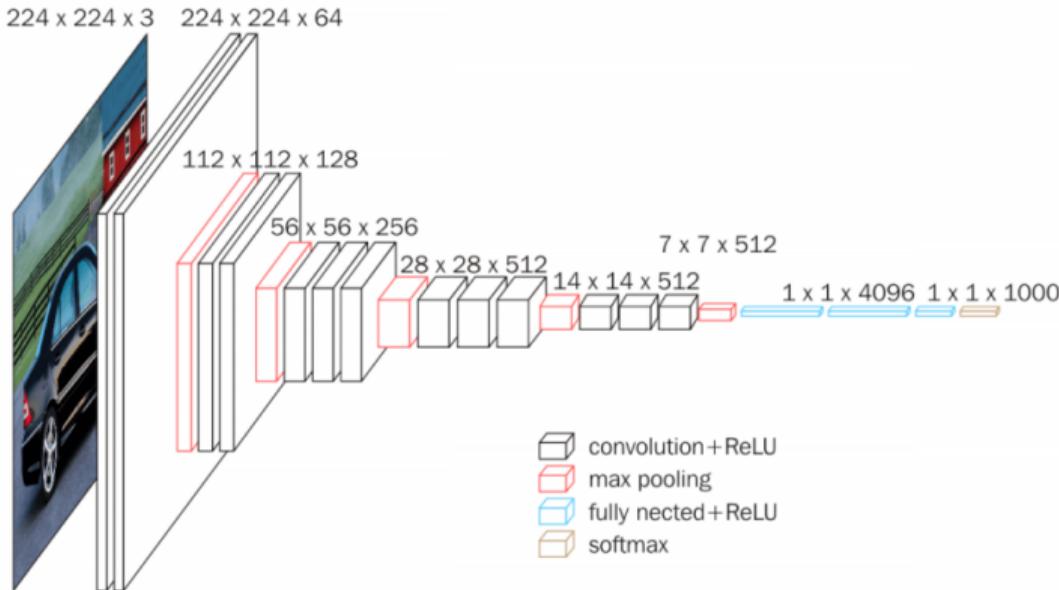
AlexNet Network - Structural Details													
Input			Output			Layer	Stride	Pad	Kernel size		in	out	# of Param
227	227	3	55	55	96	conv1	4	0	11	11	3	96	34944
55	55	96	27	27	96	maxpool1	2	0	3	3	96	96	0
27	27	96	27	27	256	conv2	1	2	5	5	96	256	614656
27	27	256	13	13	256	maxpool2	2	0	3	3	256	256	0
13	13	256	13	13	384	conv3	1	1	3	3	256	384	885120
13	13	384	13	13	384	conv4	1	1	3	3	384	384	1327488
13	13	384	13	13	256	conv5	1	1	3	3	384	256	884992
13	13	256	6	6	256	maxpool5	2	0	3	3	256	256	0
					fc6			1	1	9216	4096	37752832	
					fc7			1	1	4096	4096	16781312	
					fc8			1	1	4096	1000	4097000	
Total											62,378,344		

Статья: <https://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf>

Код: <https://github.com/pytorch/vision/blob/master/torchvision/models/alexnet.py>

Иллюстрация: <https://towardsdatascience.com/the-w3h-of-alexnet-vggnet-resnet-and-inception-7baaaecc96>

VGG (2014)



Статья: <https://arxiv.org/abs/1409.1556>

Код: <https://github.com/pytorch/vision/blob/master/torchvision/models/vgg.py>

Иллюстрация: <https://towardsdatascience.com/the-w3h-of-alexnet-vggnet-resnet-and-inception-7baaaecc96>

VGG (2014)

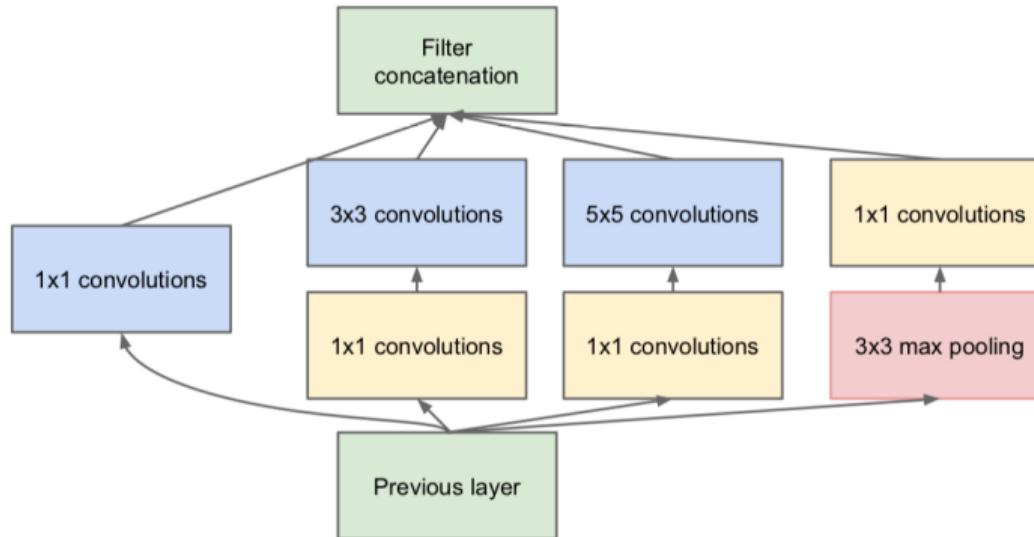
VGG16 - Structural Details													
#	Input Image			output		Layer	Stride	Kernel	in	out	Param		
1	224	224	3	224	224	64	conv3-64	1	3	3	3	64	1792
2	224	224	64	224	224	64	conv3064	1	3	3	64	64	36928
	224	224	64	112	112	64	maxpool	2	2	2	64	64	0
3	112	112	64	112	112	128	conv3-128	1	3	3	64	128	73856
4	112	112	128	112	112	128	conv3-128	1	3	3	128	128	147584
	112	112	128	56	56	128	maxpool	2	2	2	128	128	65664
5	56	56	128	56	56	256	conv3-256	1	3	3	128	256	295168
6	56	56	256	56	56	256	conv3-256	1	3	3	256	256	590080
7	56	56	256	56	56	256	conv3-256	1	3	3	256	256	590080
	56	56	256	28	28	256	maxpool	2	2	2	256	256	0
8	28	28	256	28	28	512	conv3-512	1	3	3	256	512	1180160
9	28	28	512	28	28	512	conv3-512	1	3	3	512	512	2359808
10	28	28	512	28	28	512	conv3-512	1	3	3	512	512	2359808
	28	28	512	14	14	512	maxpool	2	2	2	512	512	0
11	14	14	512	14	14	512	conv3-512	1	3	3	512	512	2359808
12	14	14	512	14	14	512	conv3-512	1	3	3	512	512	2359808
13	14	14	512	14	14	512	conv3-512	1	3	3	512	512	2359808
	14	14	512	7	7	512	maxpool	2	2	2	512	512	0
14	1	1	25088	1	1	4096	fc		1	1	25088	4096	102764544
15	1	1	4096	1	1	4096	fc		1	1	4096	4096	16781312
16	1	1	4096	1	1	1000	fc		1	1	4096	1000	4097000
Total										138,423,208			

Статья: <https://arxiv.org/abs/1409.1556>

Код: <https://github.com/pytorch/vision/blob/master/torchvision/models/vgg.py>

Иллюстрация: <https://towardsdatascience.com/the-w3h-of-alexnet-vggnet-resnet-and-inception-7baaaecc96>

GoogLeNet (2014)

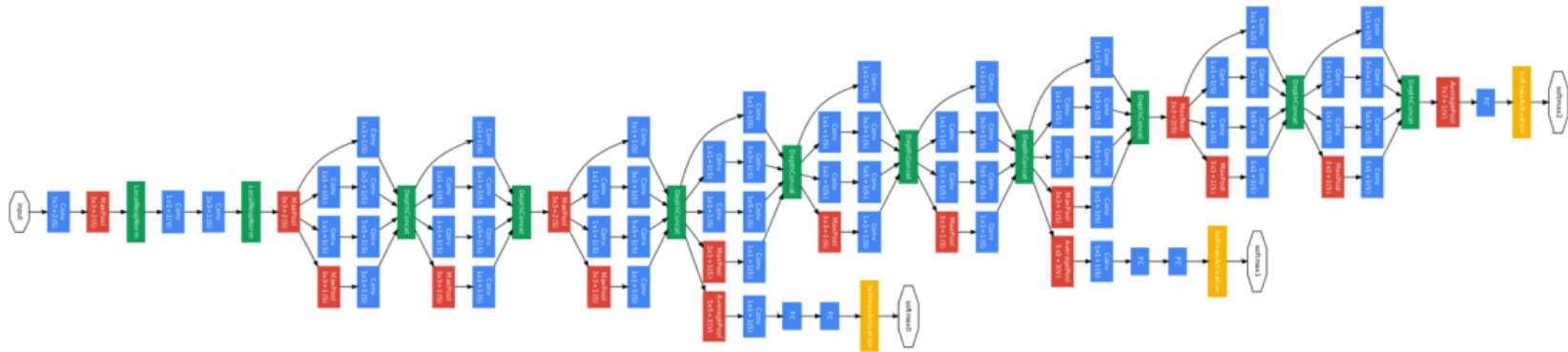


Статья: <https://research.google.com/pubs/archive/43022.pdf>

Код: <https://github.com/pytorch/vision/blob/master/torchvision/models/googlenet.py>

Иллюстрация: <https://towardsdatascience.com/the-w3h-of-alexnet-vggnet-resnet-and-inception-7baaaecc96>

GoogLeNet (2014)



Статья: <https://research.google.com/pubs/archive/43022.pdf>

Код: <https://github.com/pytorch/vision/blob/master/torchvision/models/googlenet.py>

Иллюстрация: из статьи

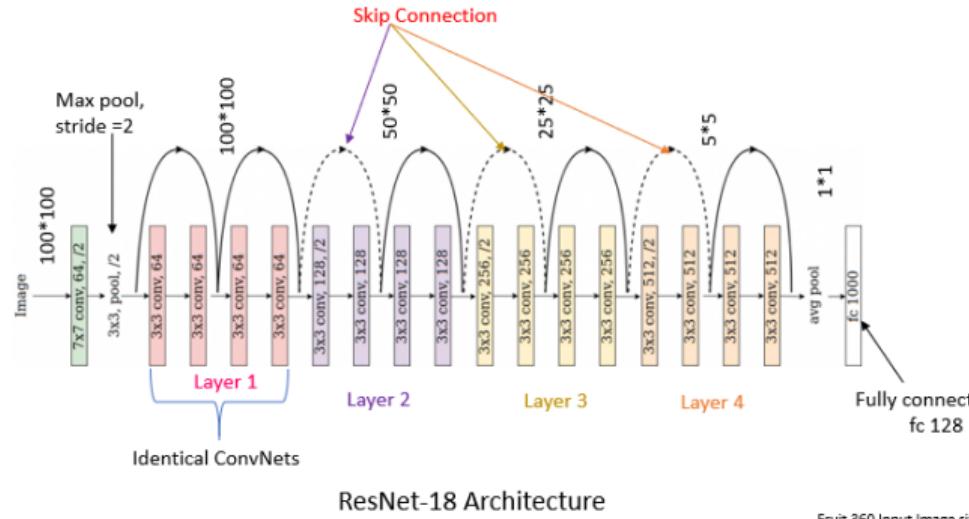
GoogLeNet (2014)

Статья: <https://research.google.com/pubs/archive/43022.pdf>

Код: <https://github.com/pytorch/vision/blob/master/torchvision/models/googlenet.py>

Иллюстрация: <https://towardsdatascience.com/the-w3h-of-alexnet-vggnet-resnet-and-inception-7baaaecc96>

ResNet (2015)



Статья: <https://arxiv.org/abs/1512.03385>

Код: <https://github.com/pytorch/vision/blob/master/torchvision/models/resnet.py>

Иллюстрация: <https://www.pluralsight.com/guides/introduction-to-resnet>

ResNet (2015)

ResNet18 - Structural Details														
#	Input Image			output			Layer	Stride	Pad	Kernel	in	out	Param	
1	227	227	3	112	112	64	conv1	2	1	7	7	3	64	9472
	112	112	64	56	56	64	maxpool	2	0.5	3	3	64	64	0
2	56	56	64	56	56	64	conv2-1	1	1	3	3	64	64	36928
3	56	56	64	56	56	64	conv2-2	1	1	3	3	64	64	36928
4	56	56	64	56	56	64	conv2-3	1	1	3	3	64	64	36928
5	56	56	64	56	56	64	conv2-4	1	1	3	3	64	64	36928
6	56	56	64	28	28	128	conv3-1	2	0.5	3	3	64	128	73856
7	28	28	128	28	28	128	conv3-2	1	1	3	3	128	128	147584
8	28	28	128	28	28	128	conv3-3	1	1	3	3	128	128	147584
9	28	28	128	28	28	128	conv3-4	1	1	3	3	128	128	147584
10	28	28	128	14	14	256	conv4-1	2	0.5	3	3	128	256	295168
11	14	14	256	14	14	256	conv4-2	1	1	3	3	256	256	590080
12	14	14	256	14	14	256	conv4-3	1	1	3	3	256	256	590080
13	14	14	256	14	14	256	conv4-4	1	1	3	3	256	256	590080
14	14	14	256	7	7	512	conv5-1	2	0.5	3	3	256	512	1180160
15	7	7	512	7	7	512	conv5-2	1	1	3	3	512	512	2359808
16	7	7	512	7	7	512	conv5-3	1	1	3	3	512	512	2359808
17	7	7	512	7	7	512	conv5-4	1	1	3	3	512	512	2359808
	7	7	512	1	1	512	avg pool	7	0	7	7	512	512	0
18	1	1	512	1	1	1000	fc					512	1000	513000
							Total							11,511,784

Статья: <https://arxiv.org/abs/1512.03385>

Код: <https://github.com/pytorch/vision/blob/master/torchvision/models/resnet.py>

Иллюстрация: <https://towardsdatascience.com/the-w3h-of-alexnet-vggnet-resnet-and-inception-7baaaeccccc96>

Пишем код