

Глубокое обучение и вообще

Ульянкин Филипп

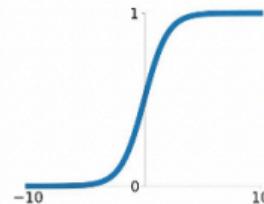
6 апреля 2022 г.

Посиделка 7: оптимизация и эвристики

В предыдущих сериях: активация

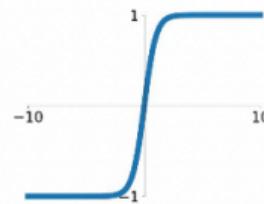
Sigmoid

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



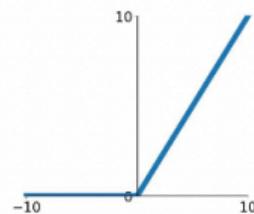
tanh

$$\tanh(x)$$



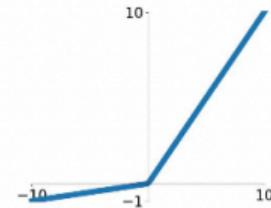
ReLU

$$\max(0, x)$$



Leaky ReLU

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

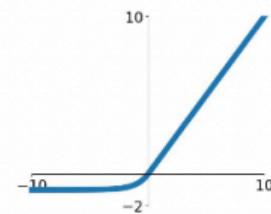


Maxout

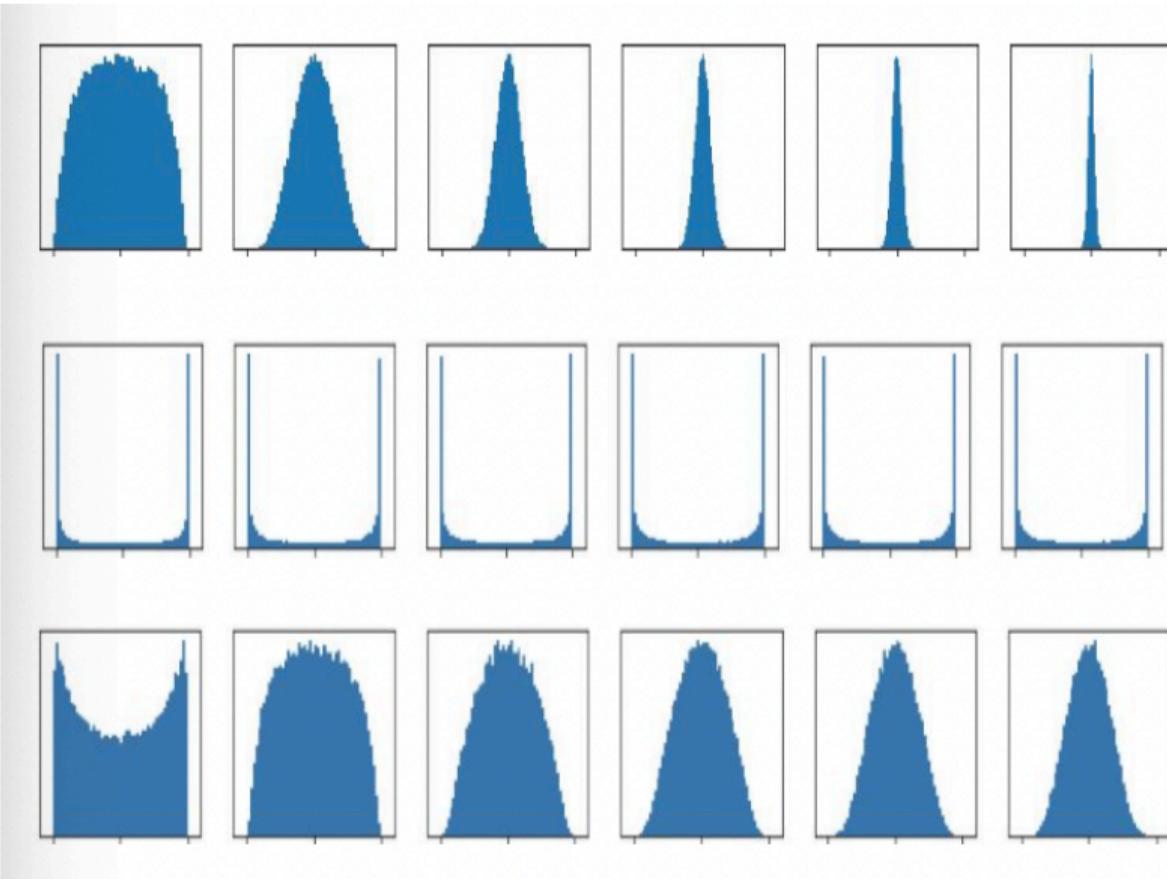
$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

ELU

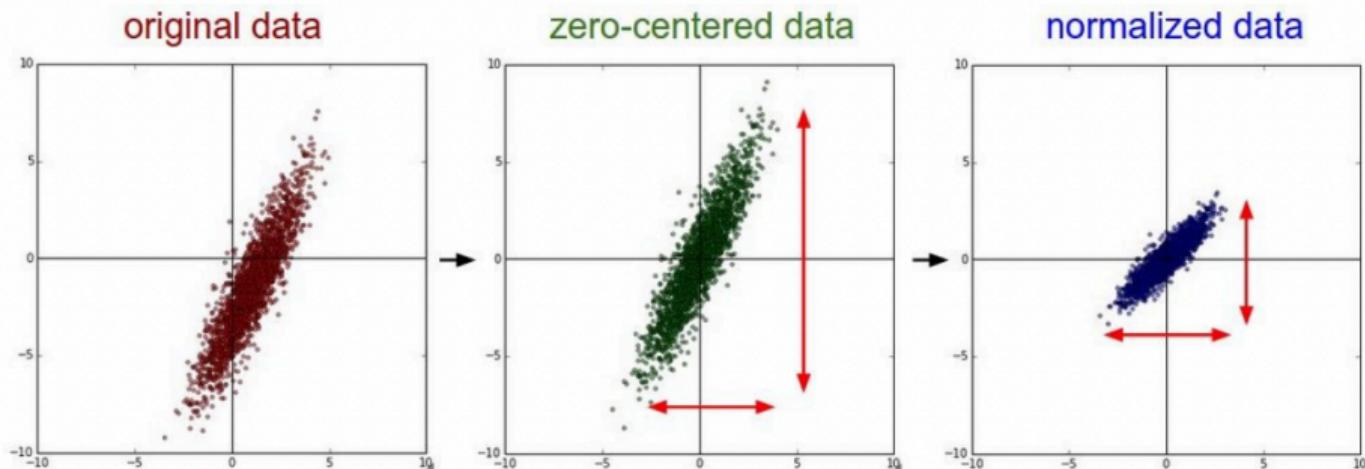
$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



В предыдущих сериях: инициализация



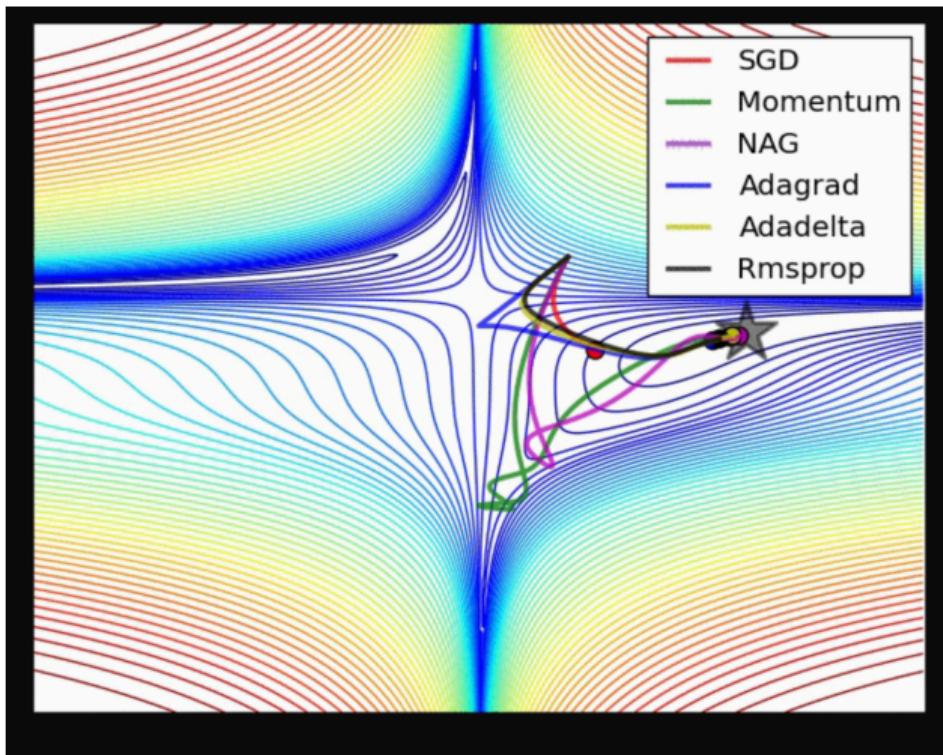
В предыдущих сериях: предобработка



В предыдущих сериях: регуляризация



В предыдущих сериях: 50 оттенков градиентного спуска



Agenda

- Ландшафт функции потерь
- Нормализация по батчам
- Организация DL-экспериментов

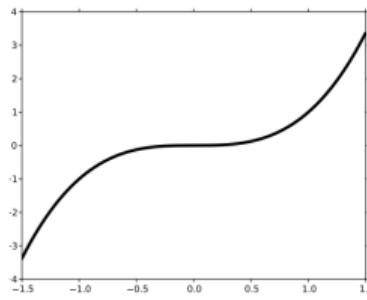
Ландшафт функции потерь

Боб чилит в локальном минимуме

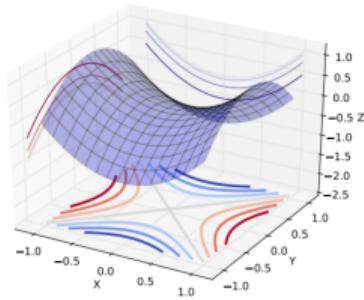


<https://hackernoon.com/life-is-gradient-descent-880c60ac1be8>

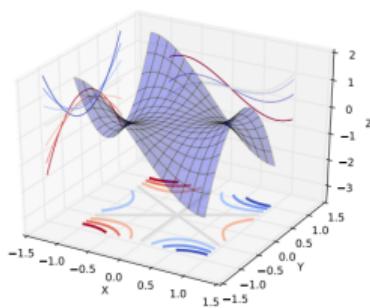
Седловые точки



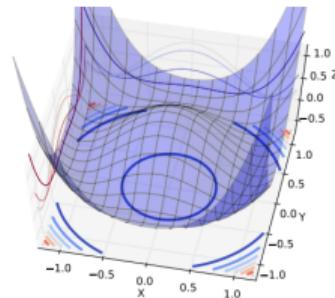
(a)



(b)



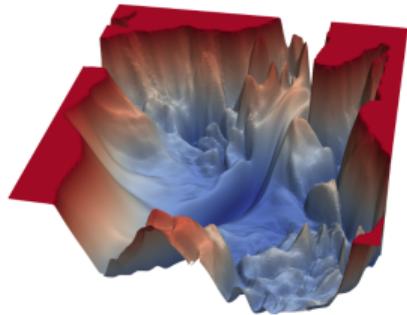
(c)



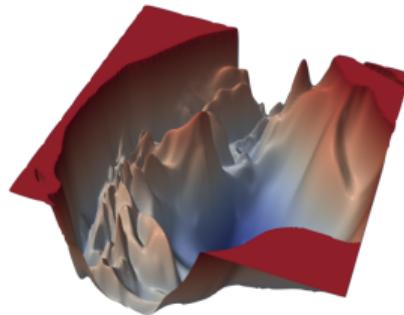
(d)

Визуализация потерь

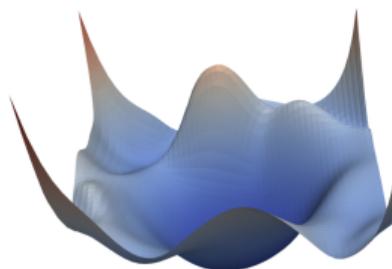
VGG-56



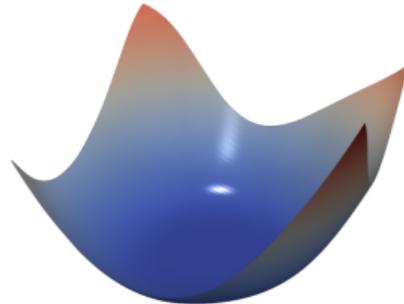
VGG-110



Renset-56



Densenet-121



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

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

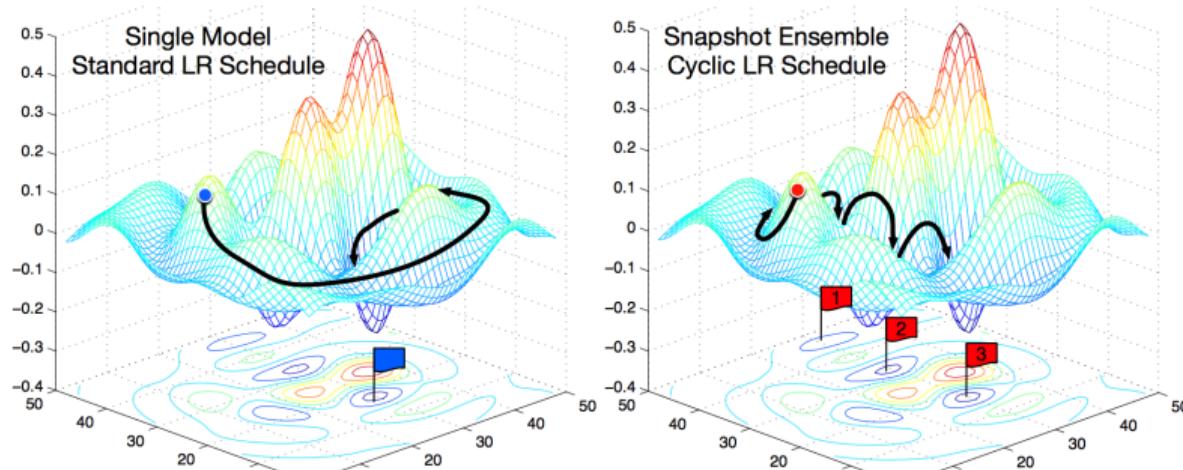
Визуализация потерь

Подробнее про это!

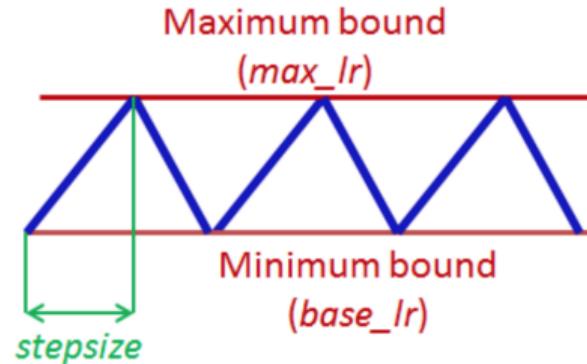
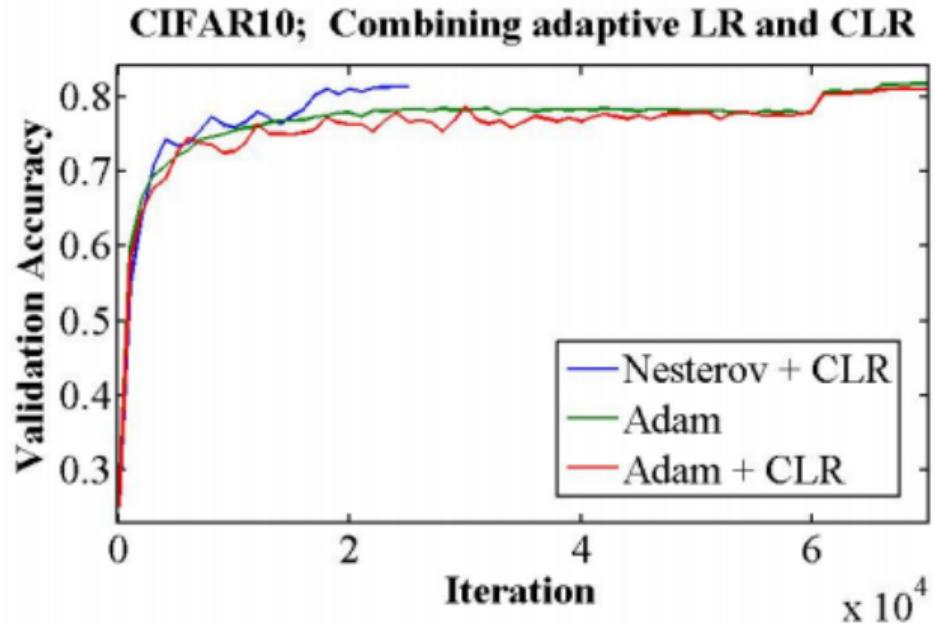
Обновить слайды про циклическую скорость, если честно это хуйня
полная, добавить нормальные слайды про параболы да берты

Циклическая скорость обучения (CLR)

- Хочется, чтобы был шанс вылезти из локального минимума, а также шанс сползти с седла \Rightarrow давайте менять глобальную скорость обучения циклически



Циклическая скорость обучения (CLR)



Нестеров с CLR отработал быстрее и лучше Adam

Нет одного правильного алгоритма на все случаи!

Всегда надо экспериментировать

<https://arxiv.org/pdf/1506.01186.pdf>

<https://openreview.net/pdf?id=BJYwwY9II>

LR range test aka LRFinder

- Эвристика для подбора learning rate
- Увеличиваем learning rate после каждого батча, остановка перед взрывом значений loss-функции
- Есть готовые реализации (в том числе в pytorch и tensorflow)
- Аналогично можно искать оптимальное значение для momentum

<https://sgugger.github.io/how-do-you-find-a-good-learning-rate.html>

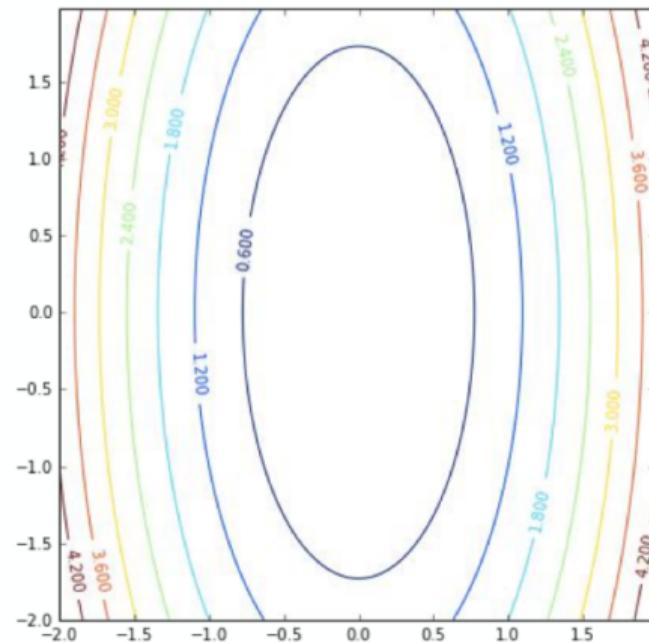
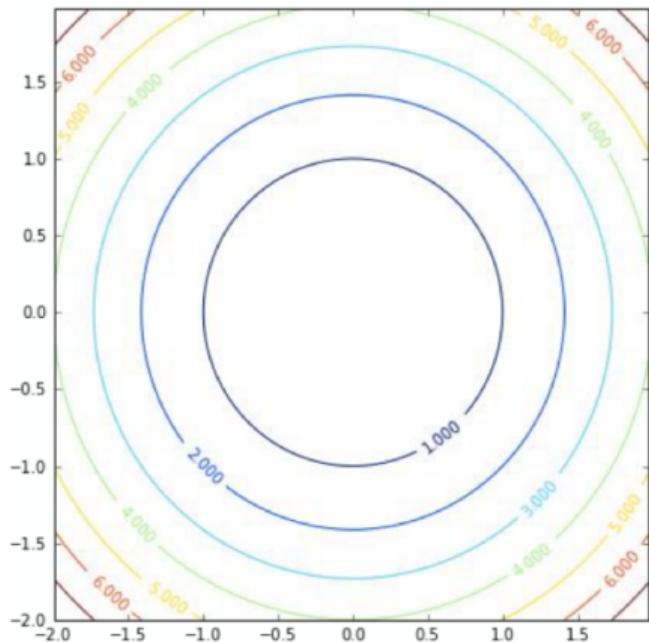
<https://arxiv.org/abs/1506.01186>

<https://arxiv.org/abs/1803.09820>

Батч-нормализация

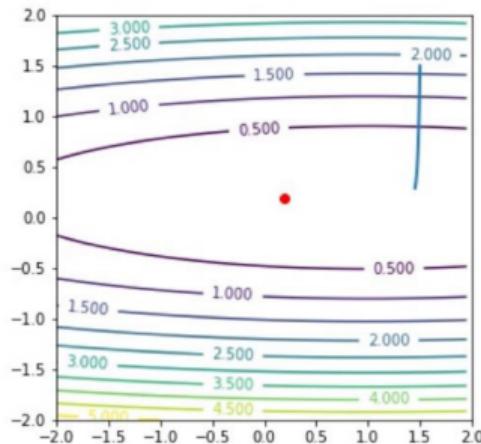
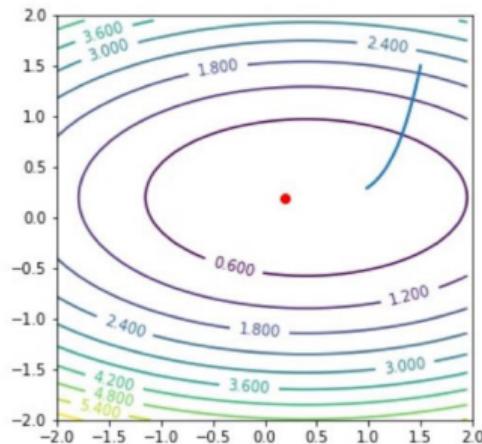
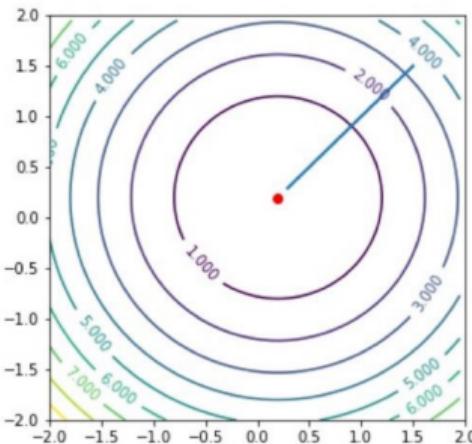


Стандартизация



Какая из ситуаций лучше для градиентного спуска?

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>

Стандартизация

Будем подавать на вход в нейросетку стандартизованные данные:

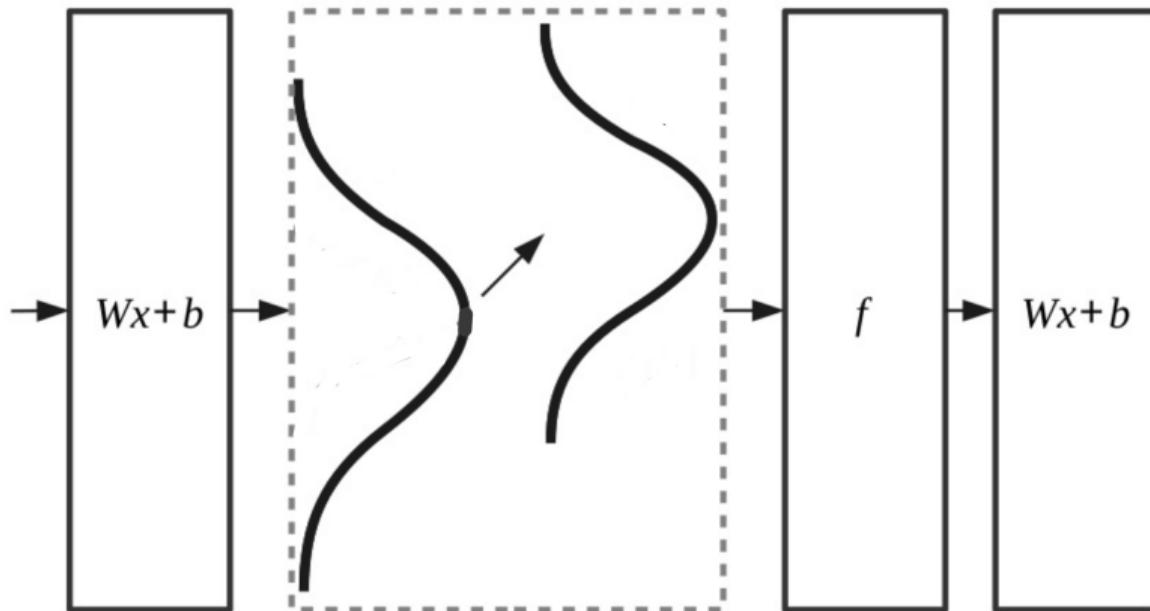
$$x_{ij}^* = \frac{x_{ij} - \hat{\mu}_j}{\sqrt{\hat{\sigma}_j^2 + \varepsilon}}$$

$$\hat{\mu}_j = \frac{1}{n} \sum_{i=1}^n x_{ij}$$

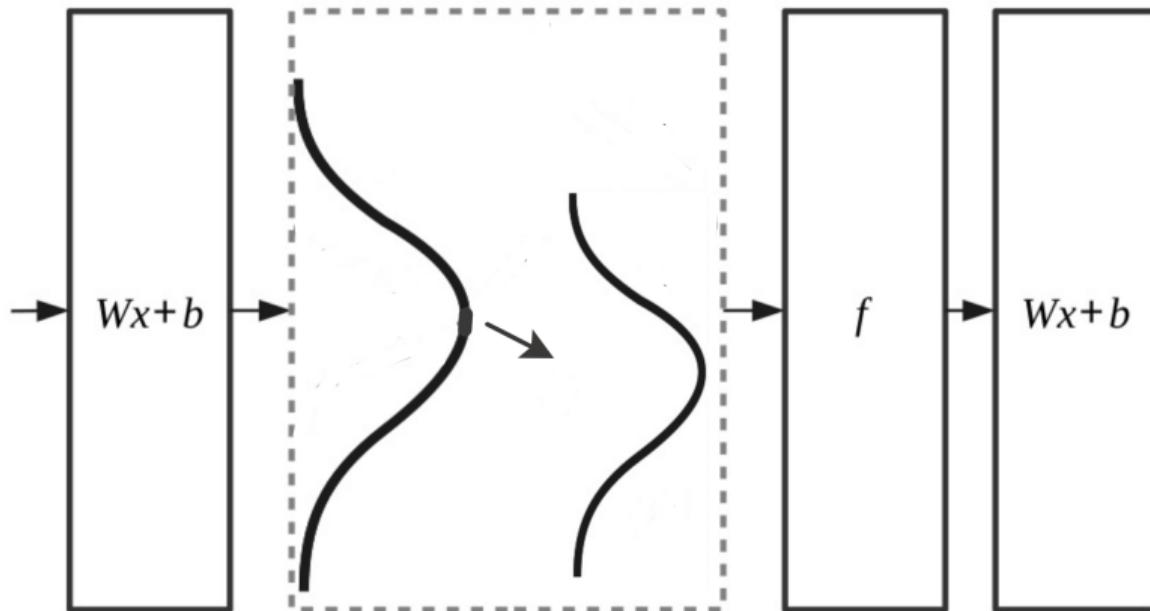
$$\hat{\sigma}_j^2 = \frac{1}{n} \sum_{i=1}^n (x_{ij} - \hat{\mu}_j)^2$$

Это помогает градиентному спуску лучше работать

А что внутри нейросетки?



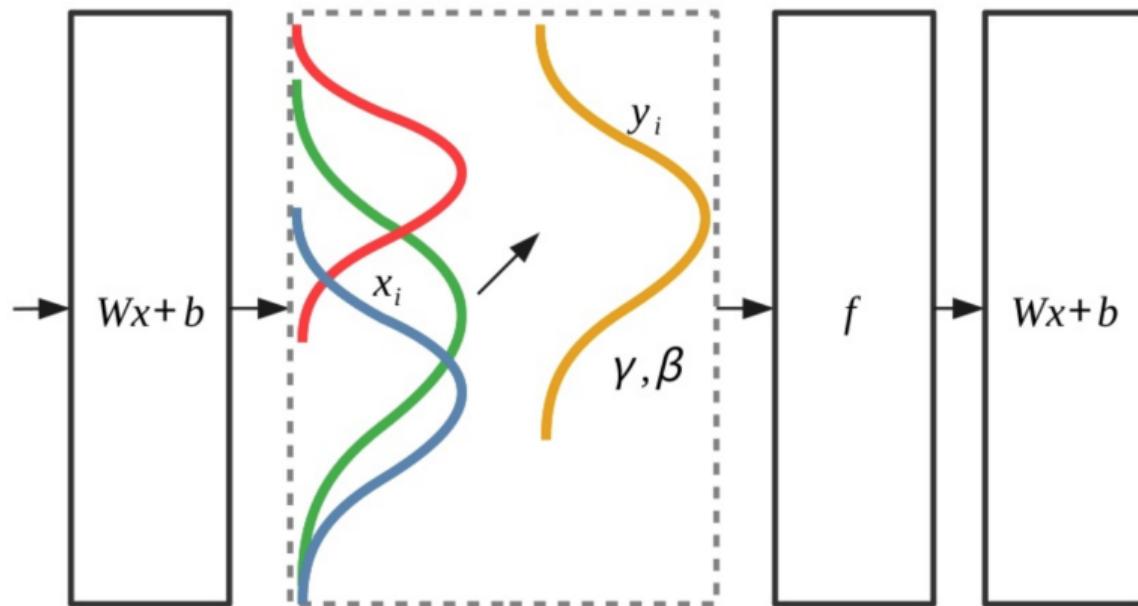
А что внутри нейросетки?



Проблема

- 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 $\hat{h} = \frac{h - \hat{\mu}_h}{\hat{\sigma}_h}$
- На выход будем выдавать $\beta \cdot \hat{h} + \gamma$, для того, чтобы у нас было больше свободы, параметры β и γ учим как параметры полносвязного слоя

Batch Normalization (2015)



Forward pass

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_1 \dots m\}$;

Parameters to be learned: γ, β

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

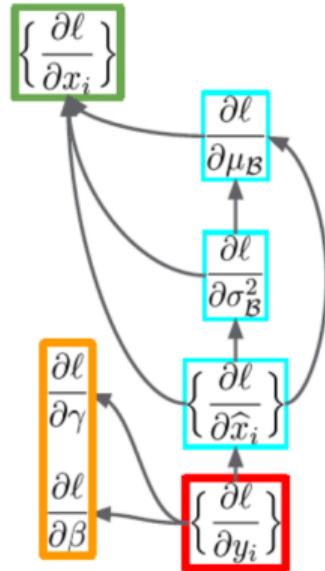
$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m} \sum_{i=1}^m x_i \quad // \text{mini-batch mean}$$

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

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

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

Backward pass



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

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

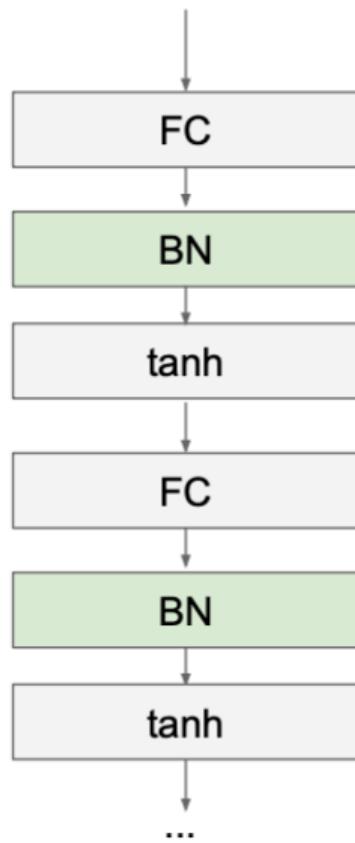
$$\frac{\partial \ell}{\partial \mu_B} = \left(\sum_{i=1}^m \frac{\partial \ell}{\partial \hat{x}_i} \cdot \frac{-1}{\sqrt{\sigma_B^2 + \epsilon}} \right) + \frac{\partial \ell}{\partial \sigma_B^2} \cdot \frac{\sum_{i=1}^m -2(x_i - \mu_B)}{m-1}$$

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

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

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

Batch Normalization (2015)



- Делает обучение глубоких сетей проще
- Обеспечивает более высокие темпы обучения и более быструю сходимость
- Сети становятся более устойчивыми к инициализации
- Действует как регуляризатор во время обучения
- Во время обучения и тестирования работает по-разному, это является источником большого числа багов и ошибок

Batch norm (2015)

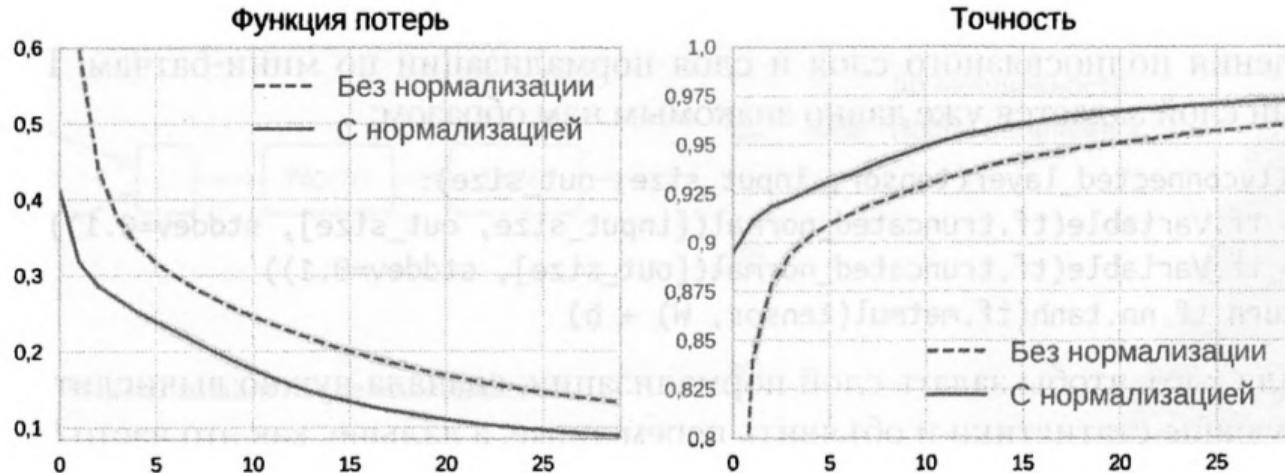
- Как считать μ_h и σ_h ?
- Оценить по текущему батчу!

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

- 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

Batch Normalization for convolutional networks

Batch Normalization for
fully-connected networks

$$\begin{aligned} \mathbf{x}: & N \times D \\ \text{Normalize} & \downarrow \\ \boldsymbol{\mu}, \sigma: & 1 \times D \\ \gamma, \beta: & 1 \times D \\ \mathbf{y} = \gamma(\mathbf{x} - \boldsymbol{\mu}) / \sigma + \beta & \end{aligned}$$

Batch Normalization for
convolutional networks
(Spatial Batchnorm, BatchNorm2D)

$$\begin{aligned} \mathbf{x}: & N \times C \times H \times W \\ \text{Normalize} & \downarrow \quad \downarrow \quad \downarrow \\ \boldsymbol{\mu}, \sigma: & 1 \times C \times 1 \times 1 \\ \gamma, \beta: & 1 \times C \times 1 \times 1 \\ \mathbf{y} = \gamma(\mathbf{x} - \boldsymbol{\mu}) / \sigma + \beta & \end{aligned}$$

Все скрины спёр у Стэнфорда, влом было набирать

Layer Normalization (2016)

Layer Normalization

Batch Normalization for
fully-connected networks

Layer Normalization for
fully-connected networks
Same behavior at train and test!
Can be used in recurrent networks

$\mathbf{x}: N \times D$

Normalize



$\mu, \sigma: 1 \times D$

$\gamma, \beta: 1 \times D$

$$\mathbf{y} = \gamma(\mathbf{x} - \mu) / \sigma + \beta$$

$\mathbf{x}: N \times D$

Normalize



$\mu, \sigma: N \times 1$

$\gamma, \beta: 1 \times D$

$$\mathbf{y} = \gamma(\mathbf{x} - \mu) / \sigma + \beta$$

Instance Normalization (2017)

Instance Normalization

Batch Normalization for convolutional networks

$$\boxed{\begin{array}{l} \mathbf{x}: N \times C \times H \times W \\ \text{Normalize} \quad \downarrow \quad \downarrow \quad \downarrow \\ \boldsymbol{\mu}, \boldsymbol{\sigma}: 1 \times C \times 1 \times 1 \end{array}}$$

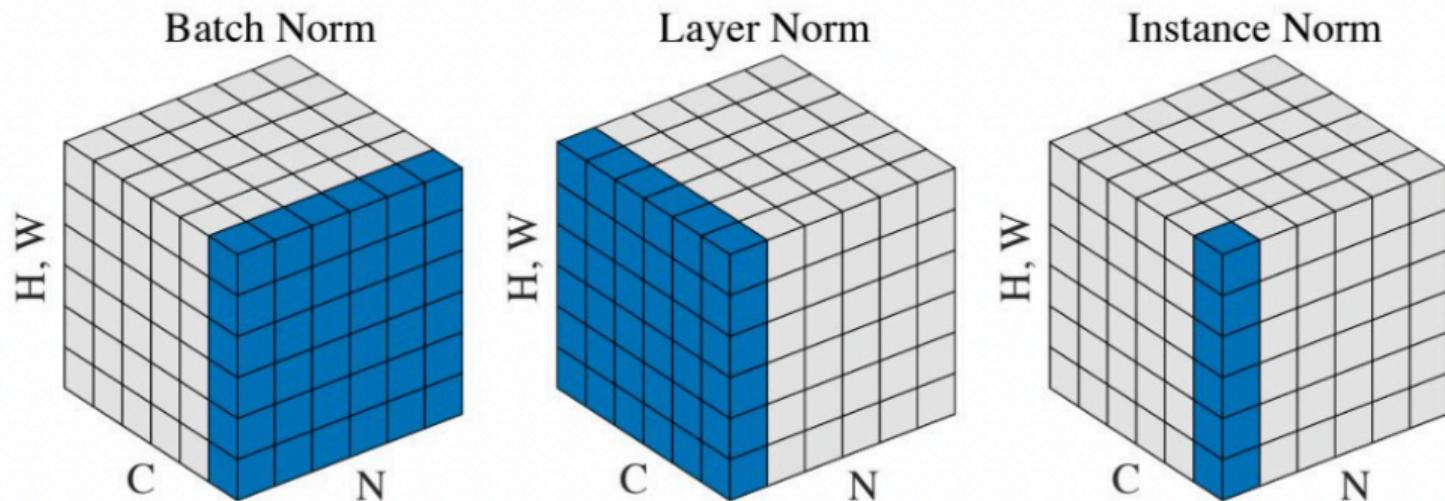
$$\begin{aligned} \mathbf{y}, \beta &: 1 \times C \times 1 \times 1 \\ \mathbf{y} &= \gamma(\mathbf{x} - \boldsymbol{\mu}) / \boldsymbol{\sigma} + \beta \end{aligned}$$

Instance Normalization for convolutional networks
Same behavior at train / test!

$$\boxed{\begin{array}{l} \mathbf{x}: N \times C \times H \times W \\ \text{Normalize} \quad \downarrow \quad \downarrow \\ \boldsymbol{\mu}, \boldsymbol{\sigma}: N \times C \times 1 \times 1 \end{array}}$$

$$\begin{aligned} \mathbf{y}, \beta &: 1 \times C \times 1 \times 1 \\ \mathbf{y} &= \gamma(\mathbf{x} - \boldsymbol{\mu}) / \boldsymbol{\sigma} + \beta \end{aligned}$$

Сравнение слоёв для нормализации



Почему это помогает при обучении

How Does Batch Normalization Help Optimization?

Shibani Santurkar*
MIT
shibani@mit.edu

Dimitris Tsipras*
MIT
tsipras@mit.edu

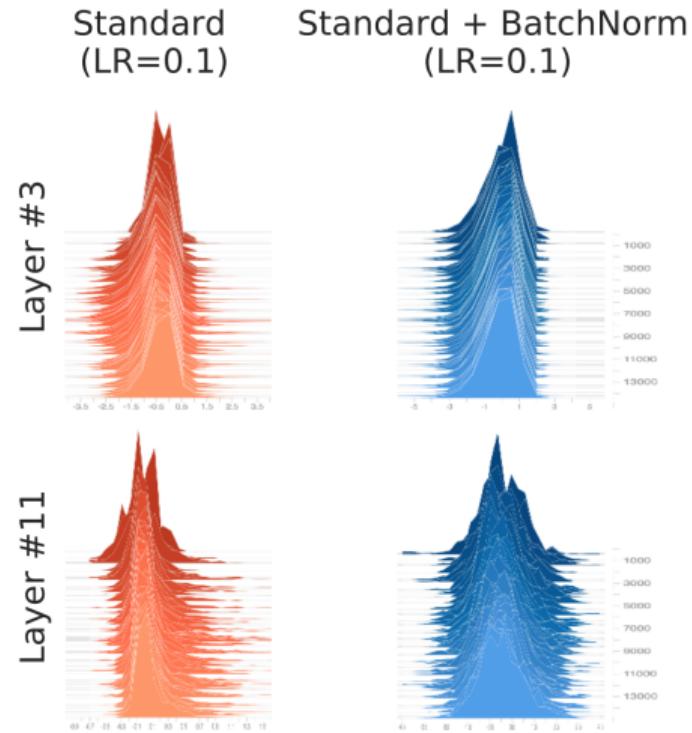
Andrew Ilyas*
MIT
ailyas@mit.edu

Aleksander Mądry
MIT
madry@mit.edu

Abstract

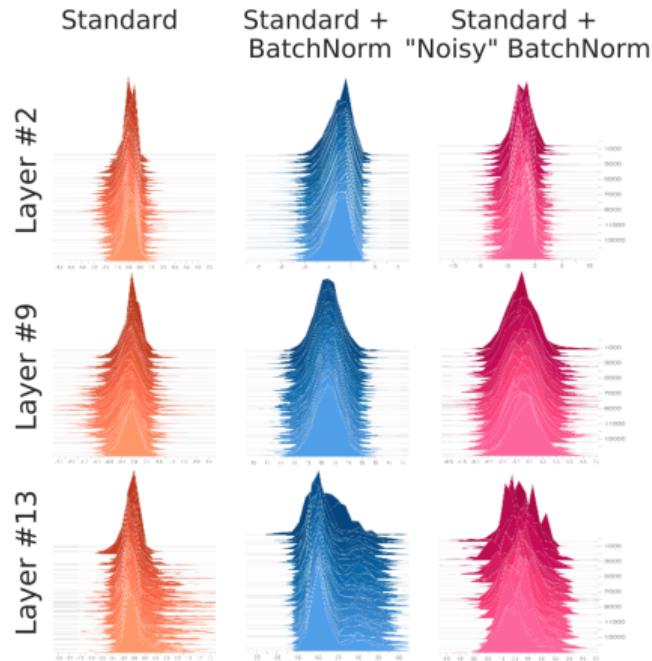
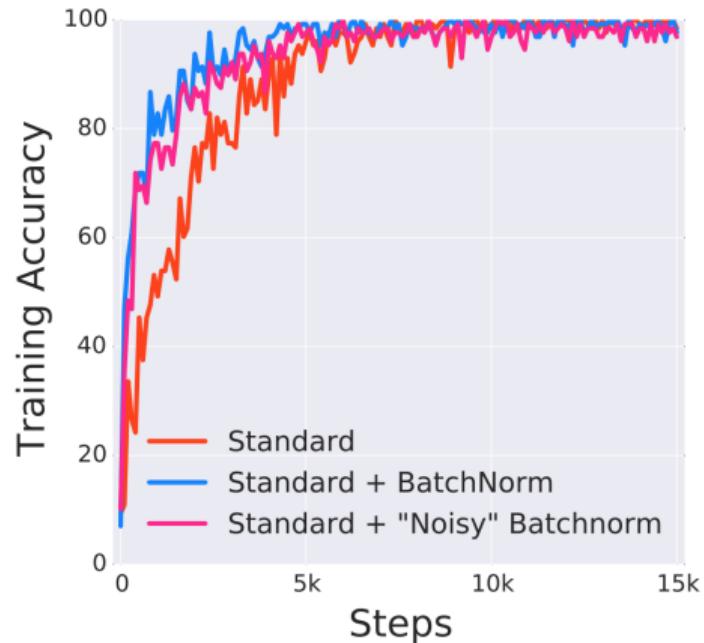
Batch Normalization (BatchNorm) is a widely adopted technique that enables faster and more stable training of deep neural networks (DNNs). Despite its pervasiveness, the exact reasons for BatchNorm's effectiveness are still poorly understood. The popular belief is that this effectiveness stems from controlling 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>

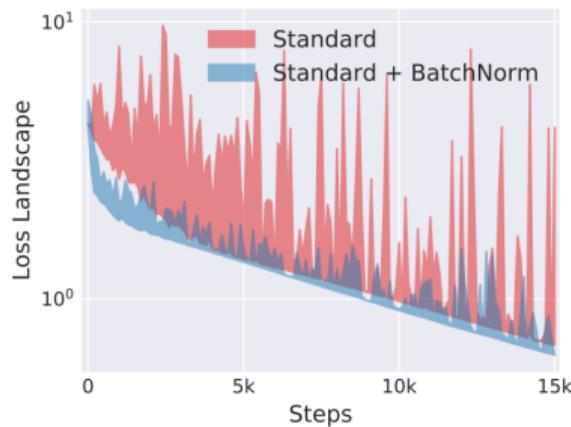
Почему это помогает при обучении



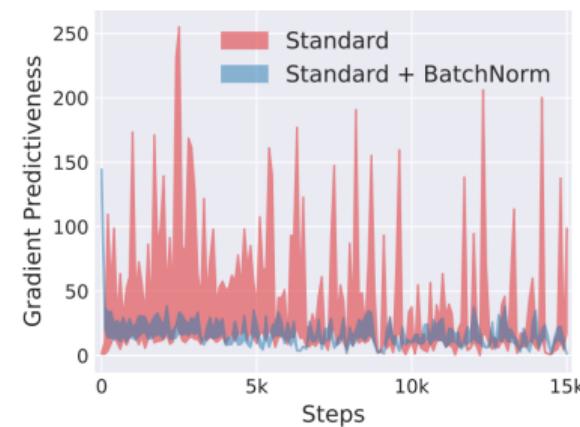
<https://arxiv.org/abs/1805.11604>

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: Layer Normalization, Weight Normalization, Batch Renormalization, Adaptive Instance Normalization, Group Normalization etc.

Организация DL-экспериментов

Проблемы при обучении нейросетей

- «Neural net training is a leaky abstraction» — Andrej Karpathy
- Знания архитектур, оптимизаторов порой недостаточно для получения хорошей модели
- Универсально наилучшего решения не бывает
- Важна точка начала экспериментов и инкрементальные улучшения

<http://karpathy.github.io/2019/04/25/recipe/>

Максим Рябинин: https://github.com/aozokin/dl_cshse_ami/blob/master/2021-fall/lectures/DL21-fall-lecture5-bestpractices.pdf

Перед началом

- Используйте проверенные временем стандарты
- Вместо своих моделей — архитектуры из популярных публикаций (ResNet в зрении, ELMo/Transformer в текстах)
- Adam со стандартным LR без расписания обойти нелегко
- Сложные функции потерь/аугментации лучше отложить
- Первые запуски на небольших датасетах, подвыборке или синтетике

Как искать ошибки

- Чтобы легче находить ошибки, снизьте число факторов влияния
- Баги могут быть как в определении и обучении модели, так и в проверке качества (даже в загрузке данных)
- В меньшем масштабе можно быстрее итерироваться и находить проблемы
- Пробуйте прогнать код на одном батче и оценить, насколько адекватные результаты вы получили: есть ли сходимость, есть ли переобучение на валидации, адекватно ли меняются метрики
- Визуализируйте всё, что можете: метрики, примеры работы модели
- DL-код — всё ещё код: полезно писать unit-тесты

Типичные ошибки: модели

- Использование ad-hoc архитектур, когда не надо
- Использование нестабильных/сложных функций потерь вместо кросс-энтропии в классификации
- Использование устаревших функций активации в глубоких сетях (сигмоид, тангенс)
- Плохая инициализация: нули/константы вместо Glorot/He

Типичные ошибки: данные

- Отсутствие аугментации/использование некорректной аугментации, разные аугментации при обучении и валидации
- Если используете предобученные модели, препроцессинг данных должен быть максимально похожим
- Считывать все данные сразу, используйте Dataset

Типичные ошибки: обучение

- Делайте чекпоинты, при них сохраняйте также параметры оптимизатора
- Функция потерь должна быть максимально близка к метрике, которую вы оптимизируете
- Если используете pytorch, не забывайте делать `zero_grad`

Организация экспериментов

- Тестируйте за один раз только одно изменение, чтобы понимать влияние каждого фактора по отдельности
- На ранних стадиях не обязательно учить до сходимости и использовать всю выборку целиком
- Ведите лог всех экспериментов
- Примеры инструментов для лога экспериментов:
<https://www.wandb.com/>
<https://www.comet.ml/>
<https://neptune.ai/>
<https://dvc.org/>
<https://mlflow.org/>

Как улучшать качество?

- Функция потерь должна быть максимально близка к метрике
- Начните с небольших экспериментов и масштабируйтесь, когда всё отлажено
- Работа с данными (количество, качество, предобработка) зачастую приносит гораздо больше эффекта, чем перебор архитектур и оптимизаторов
- Архитектуры влияют существенно, но учитывайте свои ресурсы
- Занимайтесь оптимизацией гиперпараметров в самую последнюю очередь
- Размер батча важен (ряд моделей иначе просто не учится)

Выводы

- Пользуйтесь проверенными техниками и опытом других людей
- Начните с небольших экспериментов
- Когда всё протестировано, можно масштабироваться
- Отслеживайте все доступные метрики
- Тестируйте одно изменение за раз
- Сохраняйте код/конфигурацию всех экспериментов и их результаты