

# Глубокое обучение и вообще

Ульянкин Филипп

23 января 2021 г.

**Посиделка 5:** эвристики для обучения сеток

# Agenda

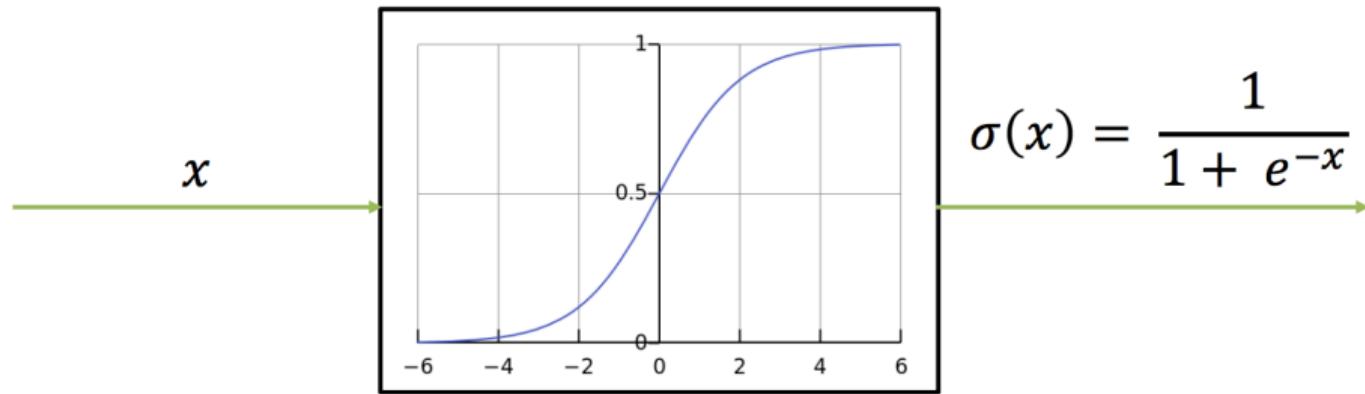
- Какими бывают функции активации
- Инициализация весов в нейросетках
- Нормализация по батчам
- Dropout
- Другие эвристики, используемые при обучении нейронных сетей

# Какими бывают функции активации

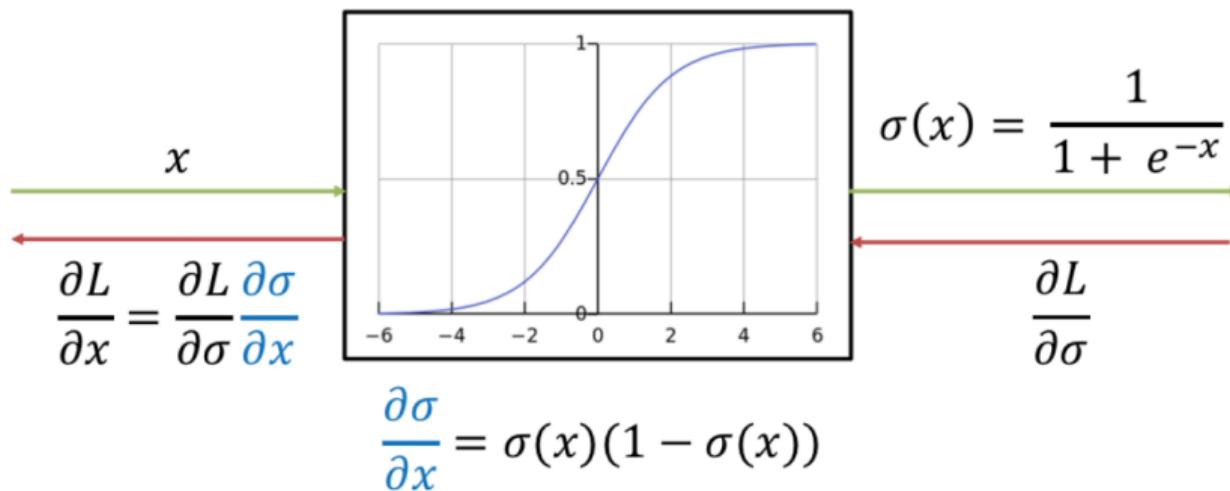


$$y = ?$$

# Sigmoid activation



# Sigmoid activation



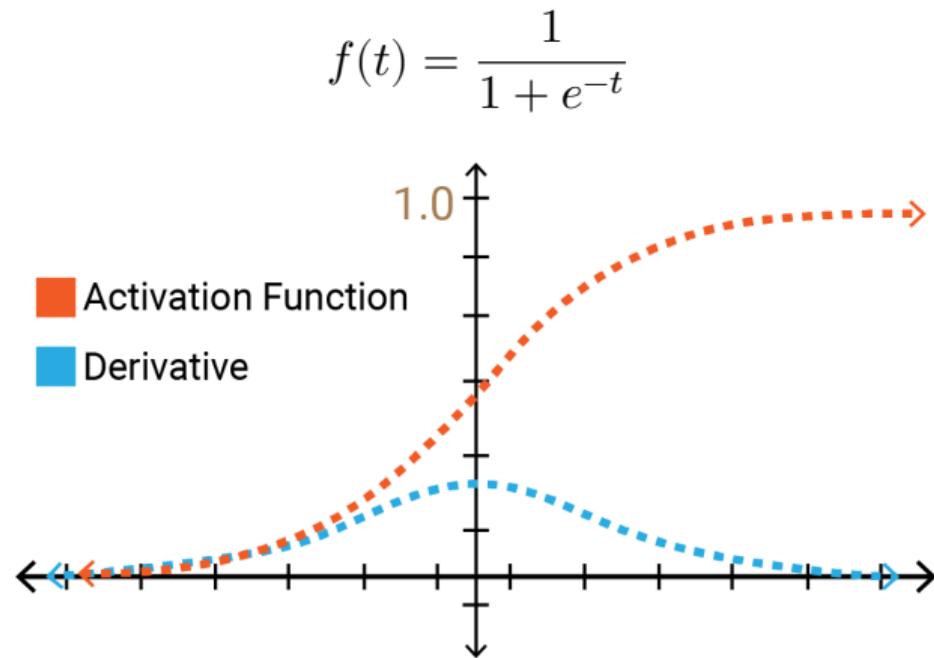
# Паралич сети

- В случае сигмоиды  $\sigma'(x) = \sigma(x) \cdot (1 - \sigma(x))$
- Сигмода принимает значения на отрезке  $[0; 1]$ , значит максимальное значение её производной это  $\frac{1}{4}$
- Если сеть очень глубокая, происходит **затухание градиента**
- Градиент затухает экспоненциально  $\Rightarrow$  сходимость замедляется, более ранние веса обновляются дольше, более глубокие веса быстрее  $\Rightarrow$  значение градиента становится ещё меньше  $\Rightarrow$  наступает **паралич сети**
- В сетях с небольшим числом слоёв этот эффект незаметен

# Затухание градиента (vanishing gradient problem)

- Изменение параметров в ходе обучения происходит на величину, которая никак не влияет на изменение выхода из сетки
- Проблема связана с очень маленькими градиентами при обновлении весов:

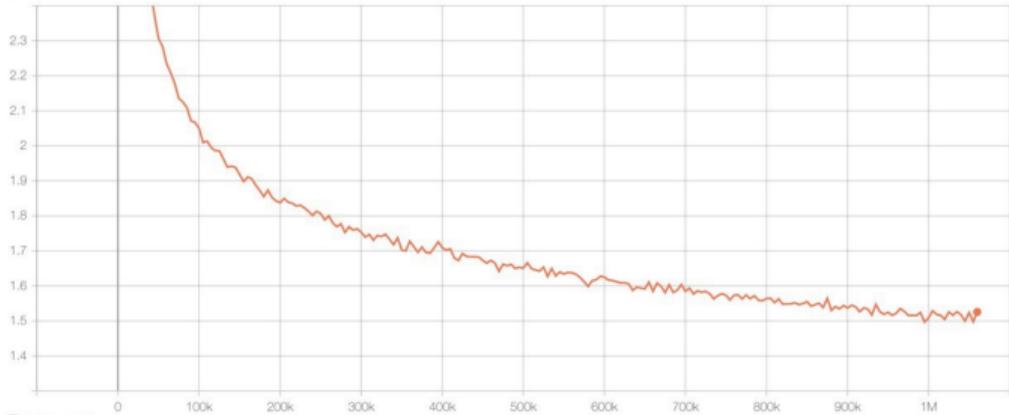
$$w_t = w_{t-1} - \gamma \cdot \nabla L(w_{t-1})$$



# Затухание градиента (vanishing gradient problem)

- Изменение параметров в ходе обучения происходит на величину, которая никак не влияет на изменение выхода из сетки
- Проблема связана с очень маленькими градиентами при обновлении весов:

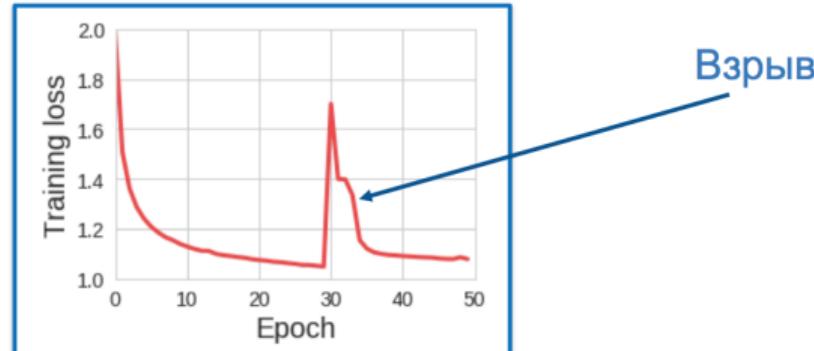
$$w_t = w_{t-1} - \gamma \cdot \nabla L(w_{t-1})$$



Толи обучение сошлося, толи веса просто  
больше не обновляются ...

# Взрыв градиента (exploding gradient problem)

- Изменение параметров в ходе обучения происходит на очень большую величину и обучение деградирует
- Проблема связана с очень большими градиентами при обновлении весов:

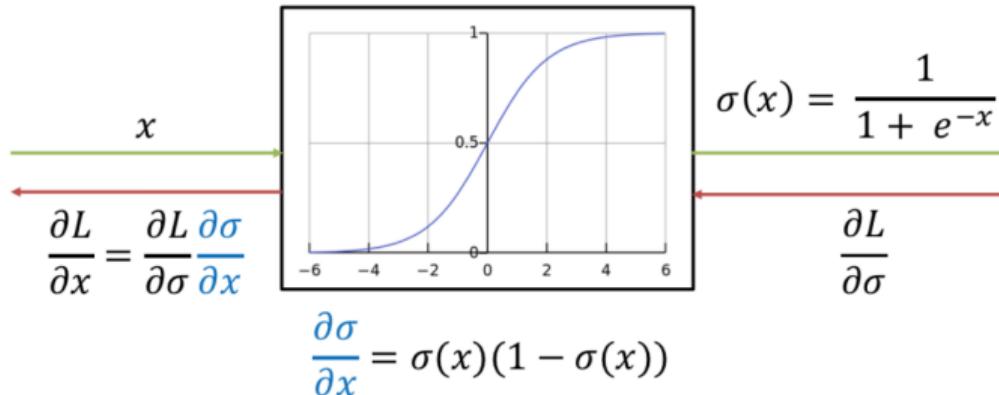


$$w_t = w_{t-1} - \gamma \cdot \nabla L(w_{t-1})$$

# Центрирование

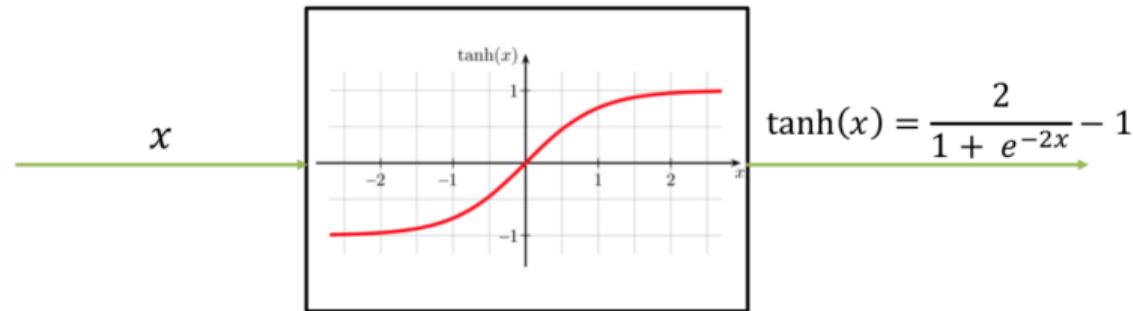
- Сигмоида не центрирована относительно нуля
- Выход слоя мы обычно находим как  $o_i = \sigma(h_i)$ , он всегда положительный, значит градиент по весам, идущим на вход в текущий нейрон тоже положительные  $\Rightarrow$  они обновляются в одинаковом направлении
- Сходимость идёт медленнее

# Sigmoid activation



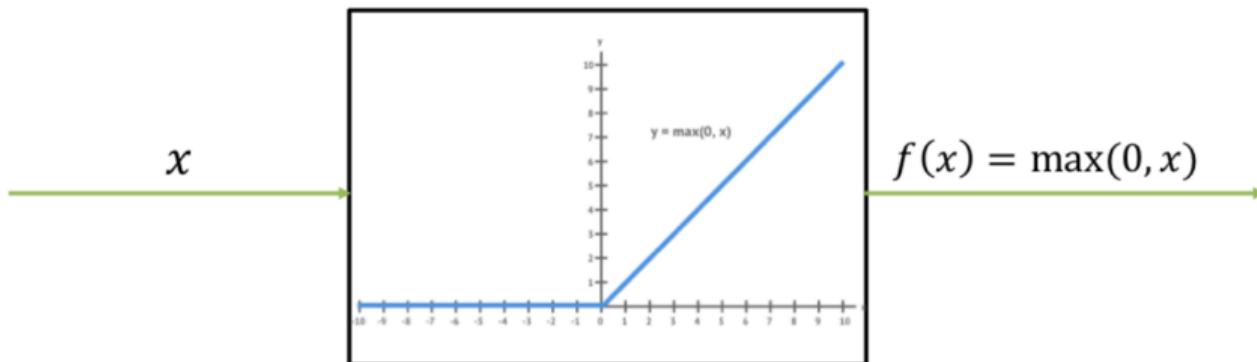
- Способствует затуханию градиента
- Не центрирована относительно нуля
- Вычислять  $e^x$  дорого

# Tanh activation



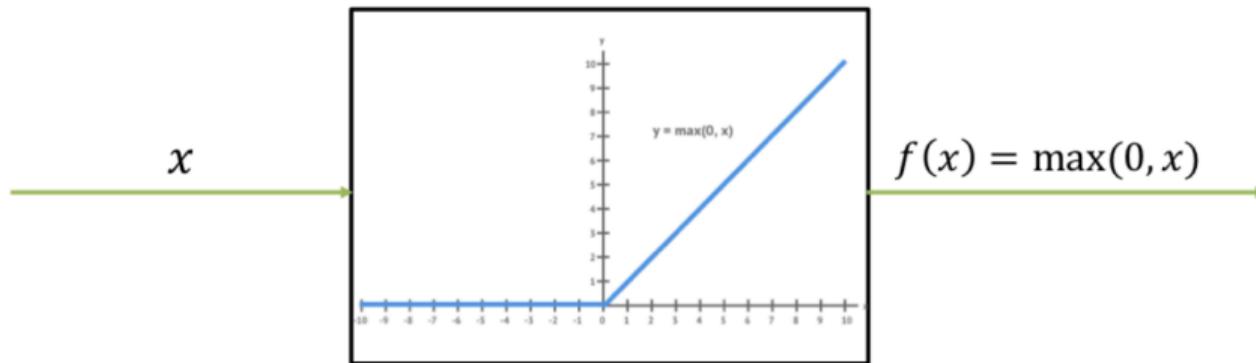
- Центрирован относительно нуля
- Всё ещё похож на сигмоиду
- $f'(x) = 1 - f(x)^2 \Rightarrow$  затухание градиента

# Rectifier Linear Unit (ReLU) activation



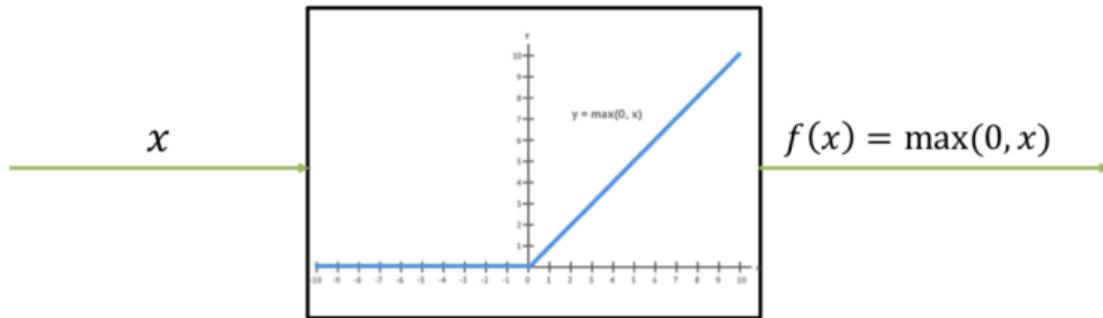
- Быстро вычисляется
- Градиент не затухает
- Сходимость сеток ускоряется

# ReLU activation



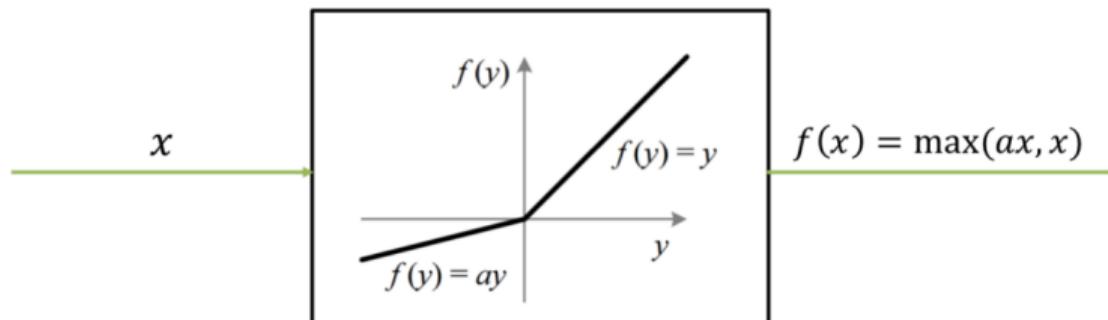
- Сетка может умереть, если активация занулится на всех нейронах
- Не центрирован относительно нуля

# Зануление ReLU



- $f(x) = \max(0, w_0 + w_1 \cdot h_1 + \dots + w_k \cdot h_k)$
- Если  $w_0$  инициализировано большим отрицательным числом, нейрон сразу умирает  $\Rightarrow$  надо аккуратно инициализировать веса

# Leaky ReLU activation

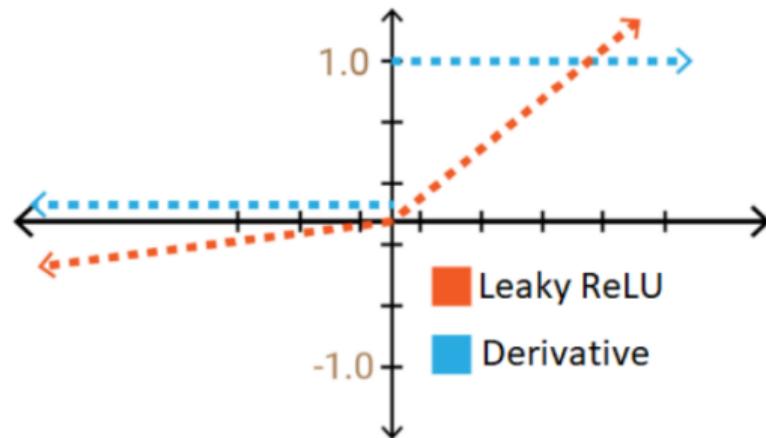


- Как ReLU, но не умирает, всё ещё легко считается
- Производная может быть любого знака
- Важно, чтобы  $a \neq 1$ , иначе линейность

# Leaky ReLU activation

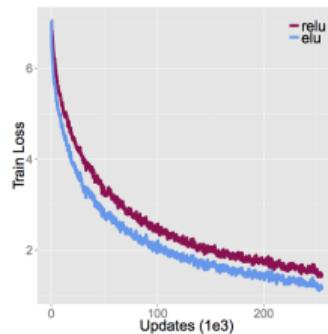
- При использовании ReLU градиенты не взрываются и не затухают, они при проходе через этот слой всегда равны константе

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

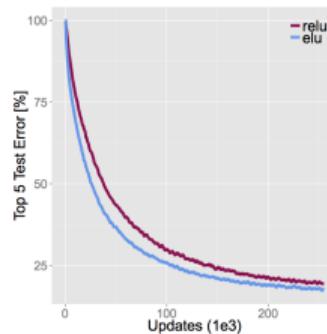


# Exponential linear unit (ELU) activation

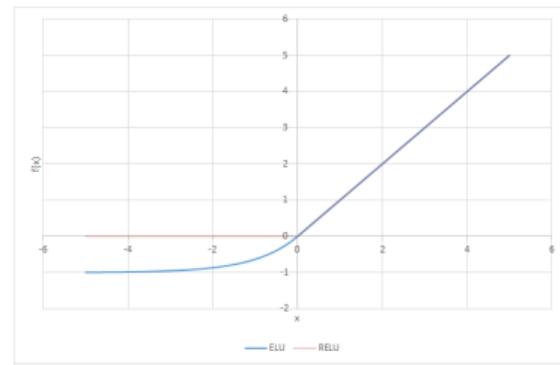
- Функция "примерно центрирована" относительно нуля
- Это позволяет избегать "смещения" в положительном направлении и ускоряет обучение



(a) Training loss



(b) Top-5 test error



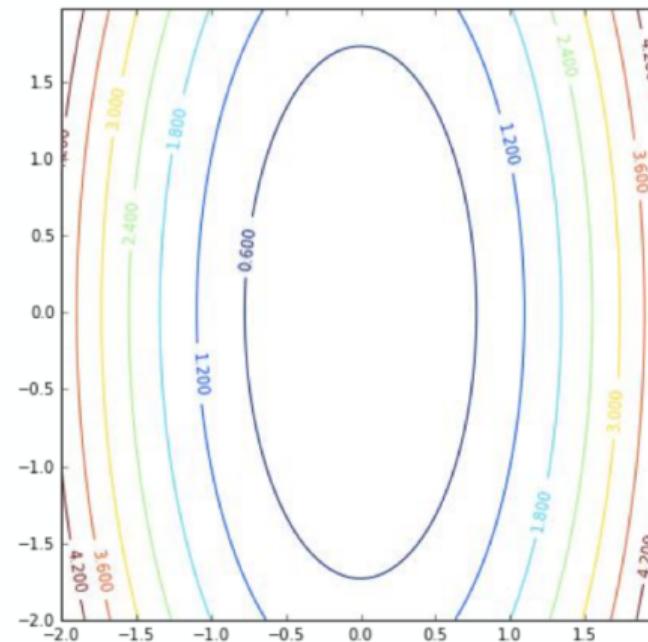
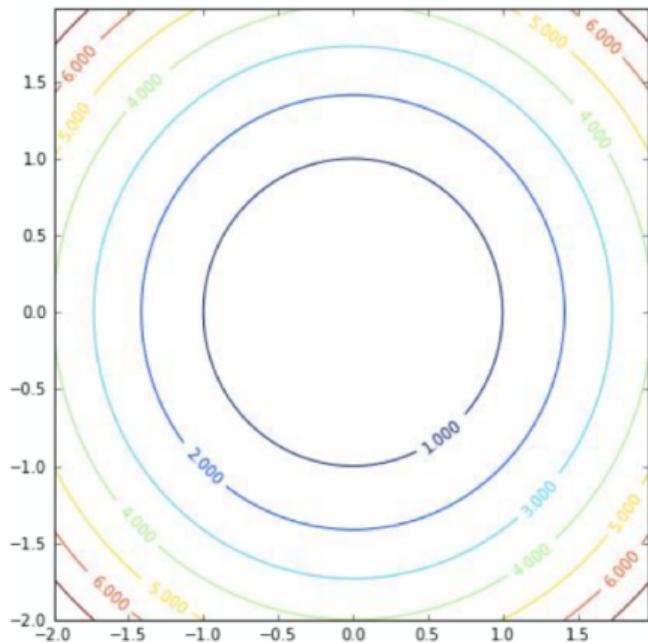
$$f(x) = \begin{cases} x, & x \geq 0 \\ \alpha \cdot (e^x - 1), & x < 0 \end{cases}$$

<https://arxiv.org/pdf/1511.07289.pdf>

# Батч-нормализация

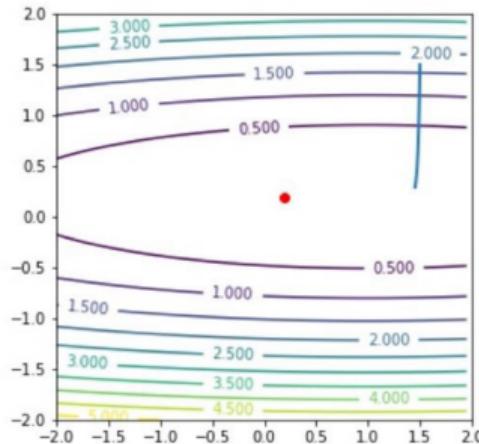
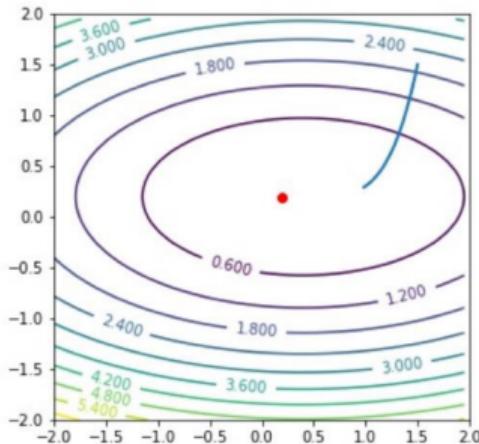
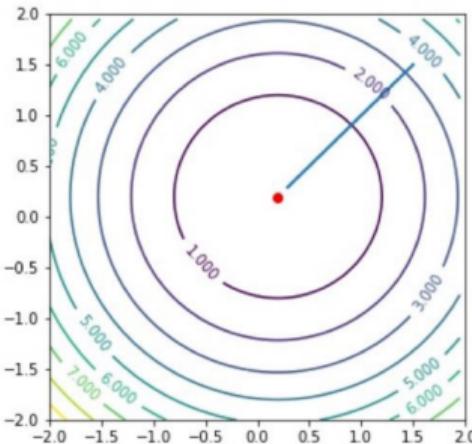


# Стандартизация



Какая из ситуаций лучше для SGD?

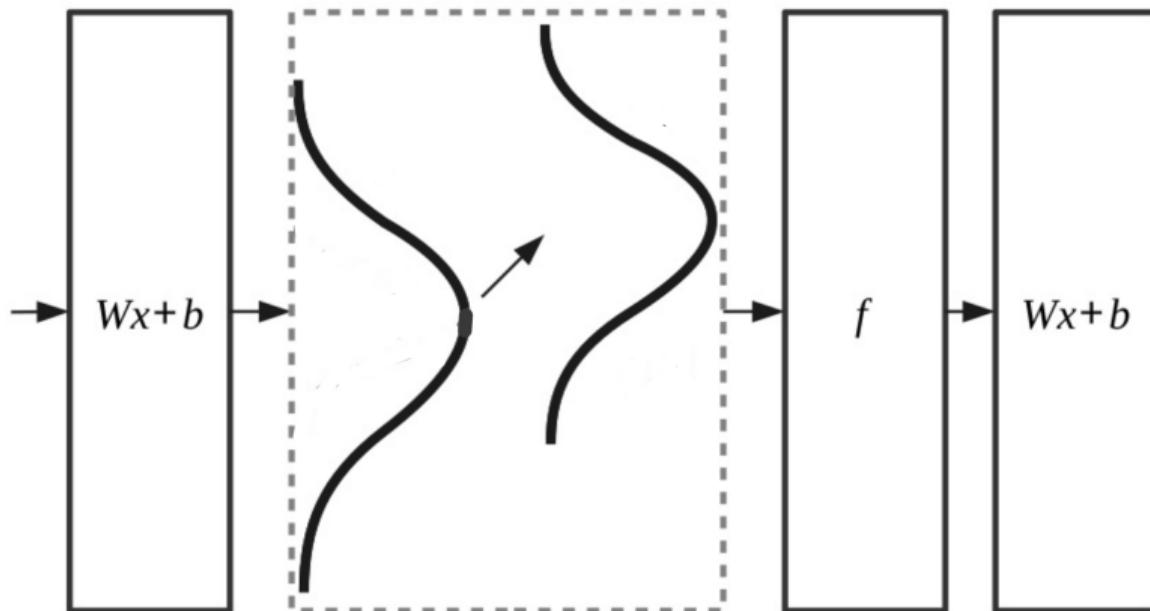
# Стандартизация



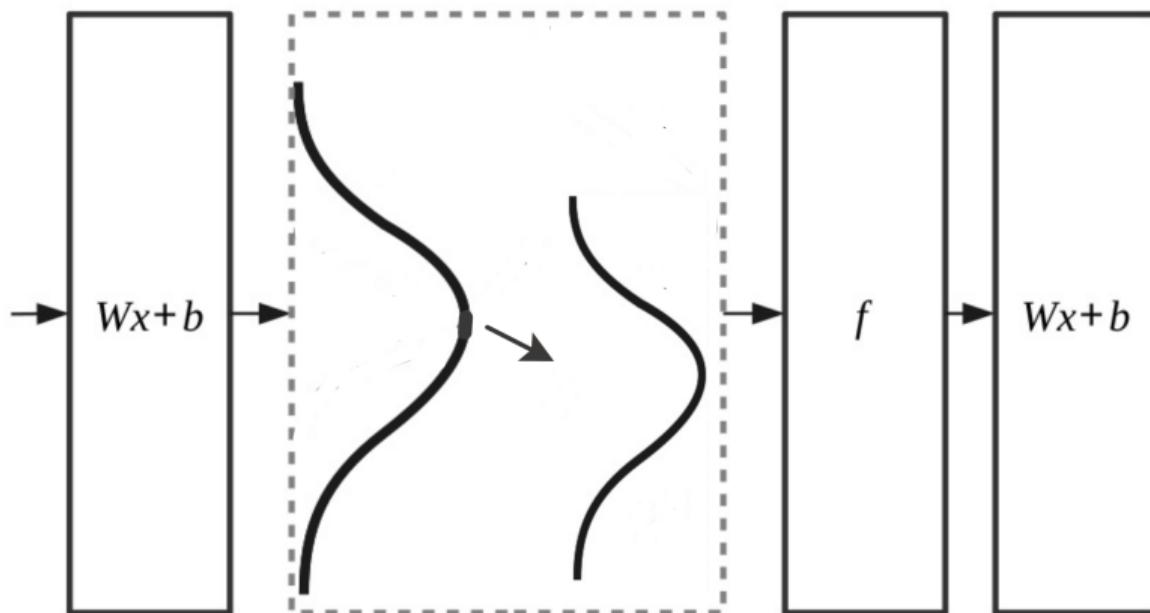
Градиентные методы быстрее сходятся, если градиенты в одном масштабе

<https://github.com/hse-aml/intro-to-dl/blob/master/week2/v2/ill-conditioned-demo.ipynb>

# А что внутри нейросетки?



# А что внутри нейросетки?



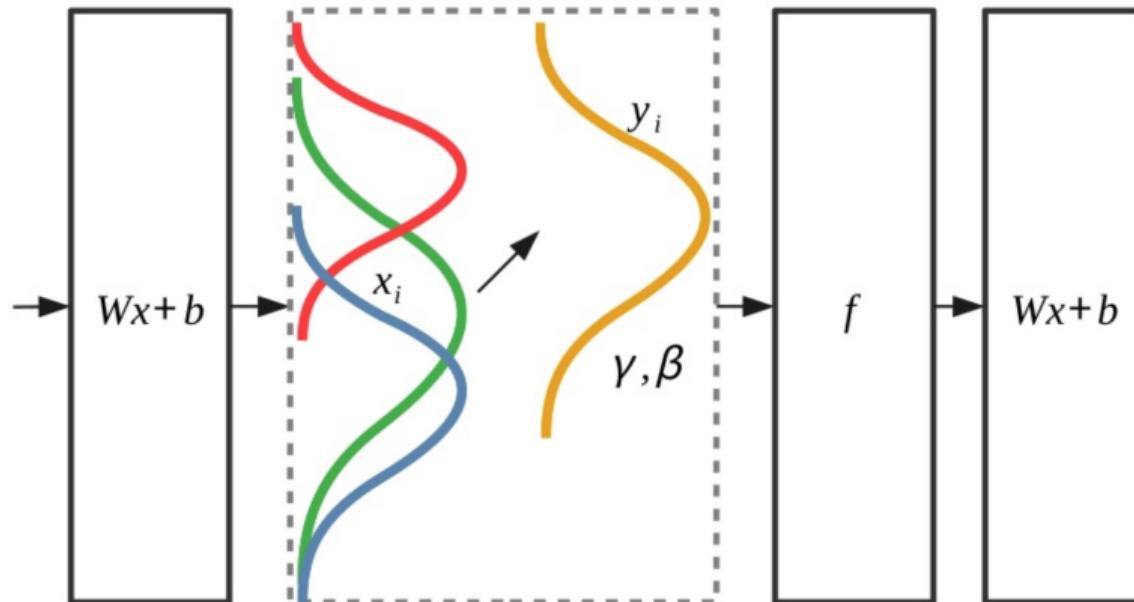
# Проблема

- 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  $\beta$  и  $\gamma$  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:  $\gamma, \beta$

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

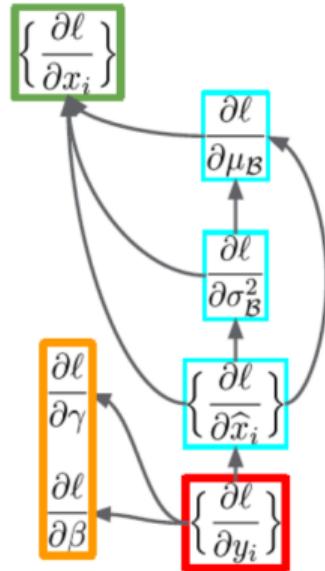
$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m} \sum_{i=1}^m x_i \quad // \text{mini-batch mean}$$

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

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

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

# 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)

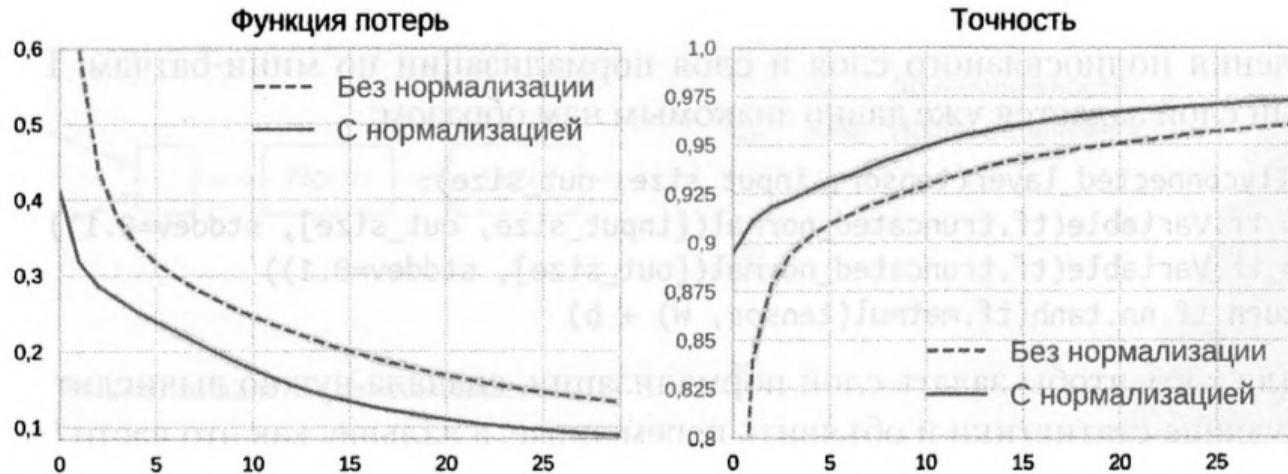
- Как считать  $\mu_h$  и  $\sigma_h$ ?
- Оценить по текущему батчу!

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

- Коэффициенты  $\beta$  и  $\gamma$  оцениваются в ходе обратного распространения ошибки,  $\mu$  и  $\sigma$  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

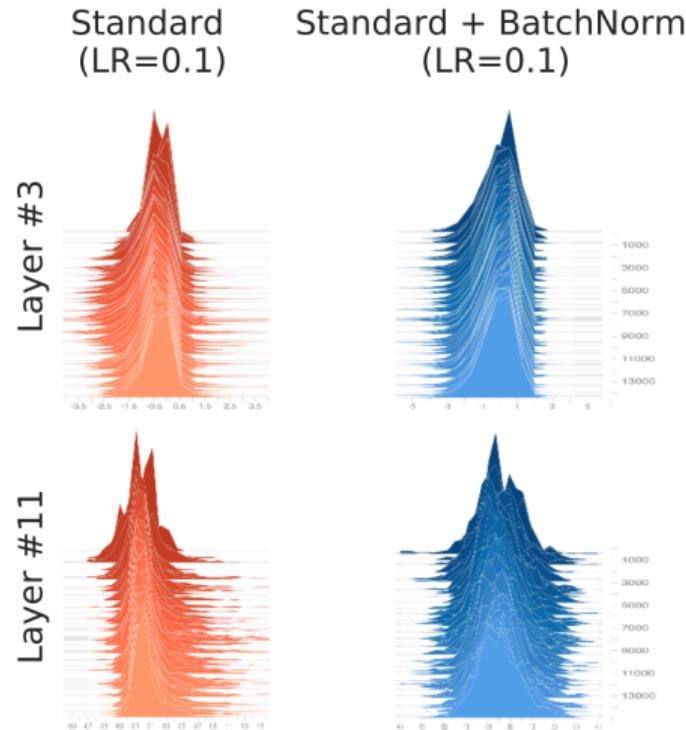
Andrew Ilyas\*  
MIT  
ailyas@mit.edu

Aleksander Mądry  
MIT  
madry@mit.edu

### Abstract

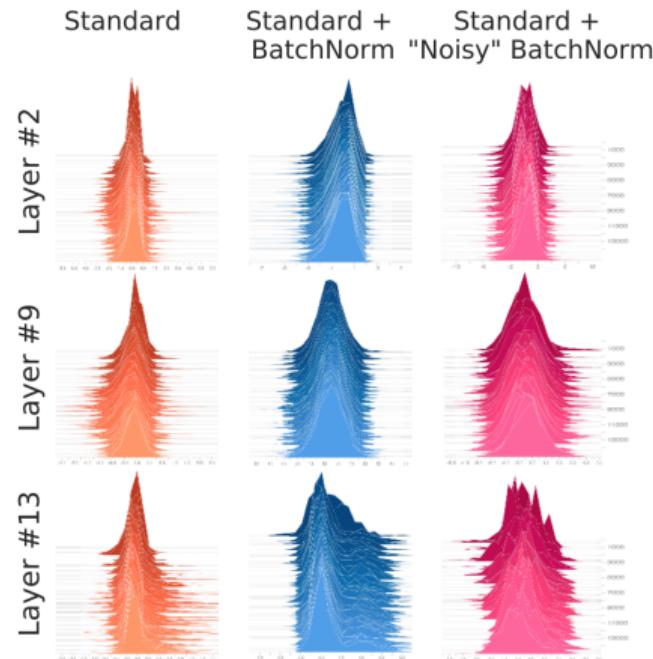
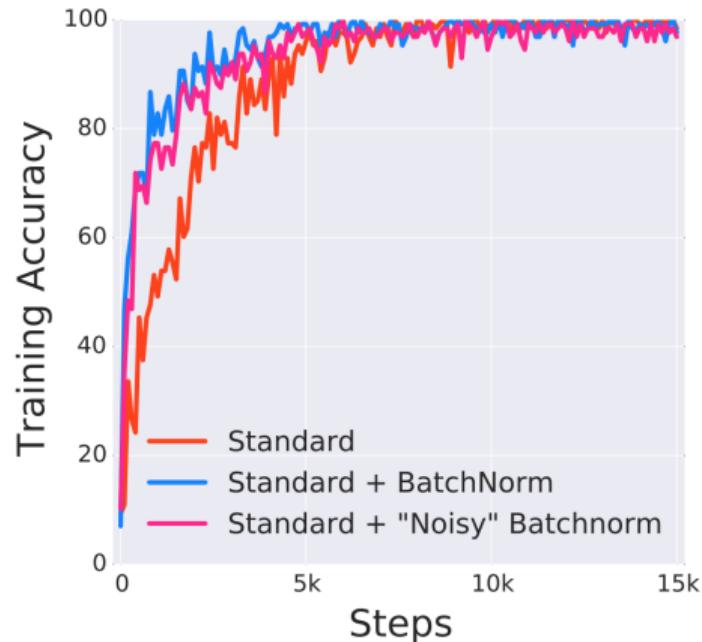
Batch Normalization (BatchNorm) is a widely adopted technique that enables faster and more stable training of deep neural networks (DNNs). Despite its pervasiveness, the exact reasons for BatchNorm's effectiveness are still poorly understood. The popular belief is that this effectiveness stems from controlling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

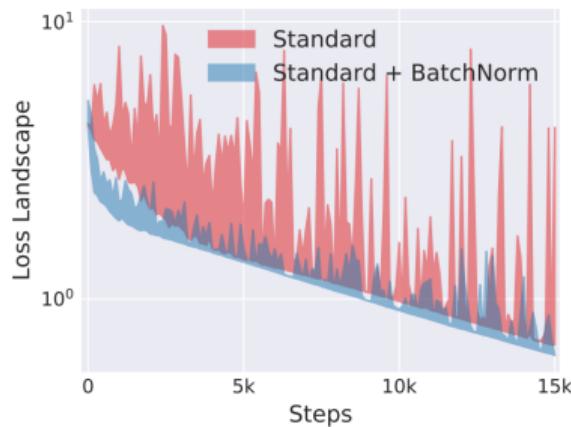
# Почему это помогает при обучении



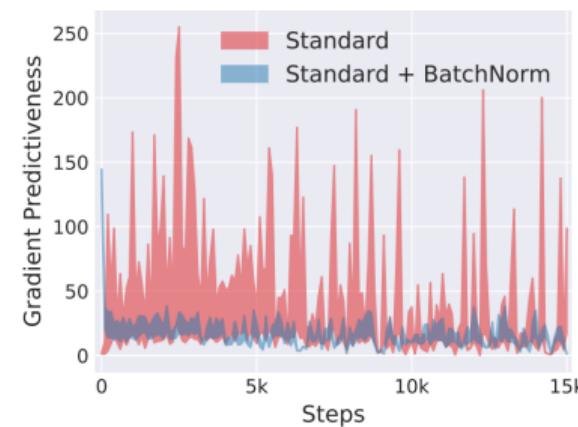
<https://arxiv.org/abs/1805.11604>

# Почему это помогает при обучении

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



(a) loss landscape

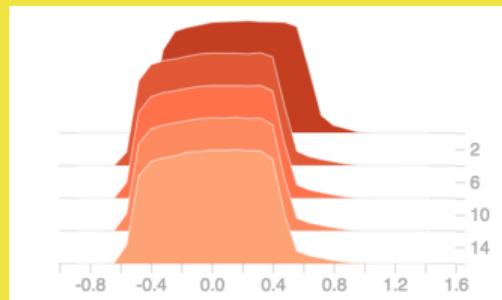


(b) gradient predictiveness

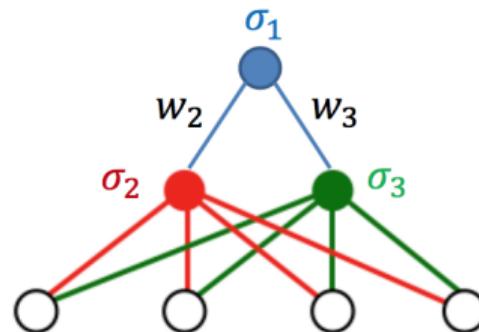
## Трюки

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

# Инициализация весов



# Инициализация весов

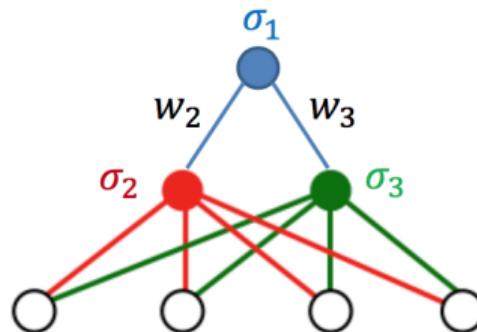


$$\frac{\partial L}{\partial w_2} = \frac{\partial L}{\partial \sigma_1} \sigma_1(1 - \sigma_1) \color{red}{\sigma_2}$$

$$\frac{\partial L}{\partial w_3} = \frac{\partial L}{\partial \sigma_1} \sigma_1(1 - \sigma_1) \color{green}{\sigma_3}$$

- Что будет, если инициализировать веса нулями?

# Инициализация весов

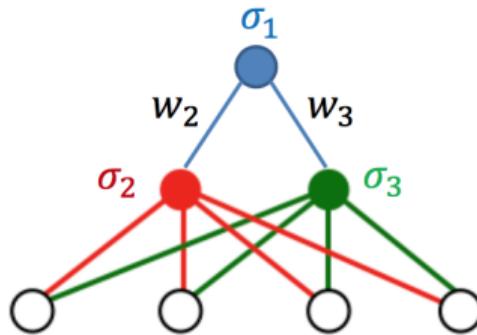


$$\frac{\partial L}{\partial w_2} = \frac{\partial L}{\partial \sigma_1} \sigma_1 (1 - \sigma_1) \color{red}{\sigma_2}$$

$$\frac{\partial L}{\partial w_3} = \frac{\partial L}{\partial \sigma_1} \sigma_1 (1 - \sigma_1) \color{green}{\sigma_3}$$

- Что будет, если инициализировать веса нулями?
- $\sigma_2$  и  $\sigma_3$  будут обновляться одинаково

# Инициализация весов



$$\frac{\partial L}{\partial w_2} = \frac{\partial L}{\partial \sigma_1} \sigma_1 (1 - \sigma_1) \sigma_2$$

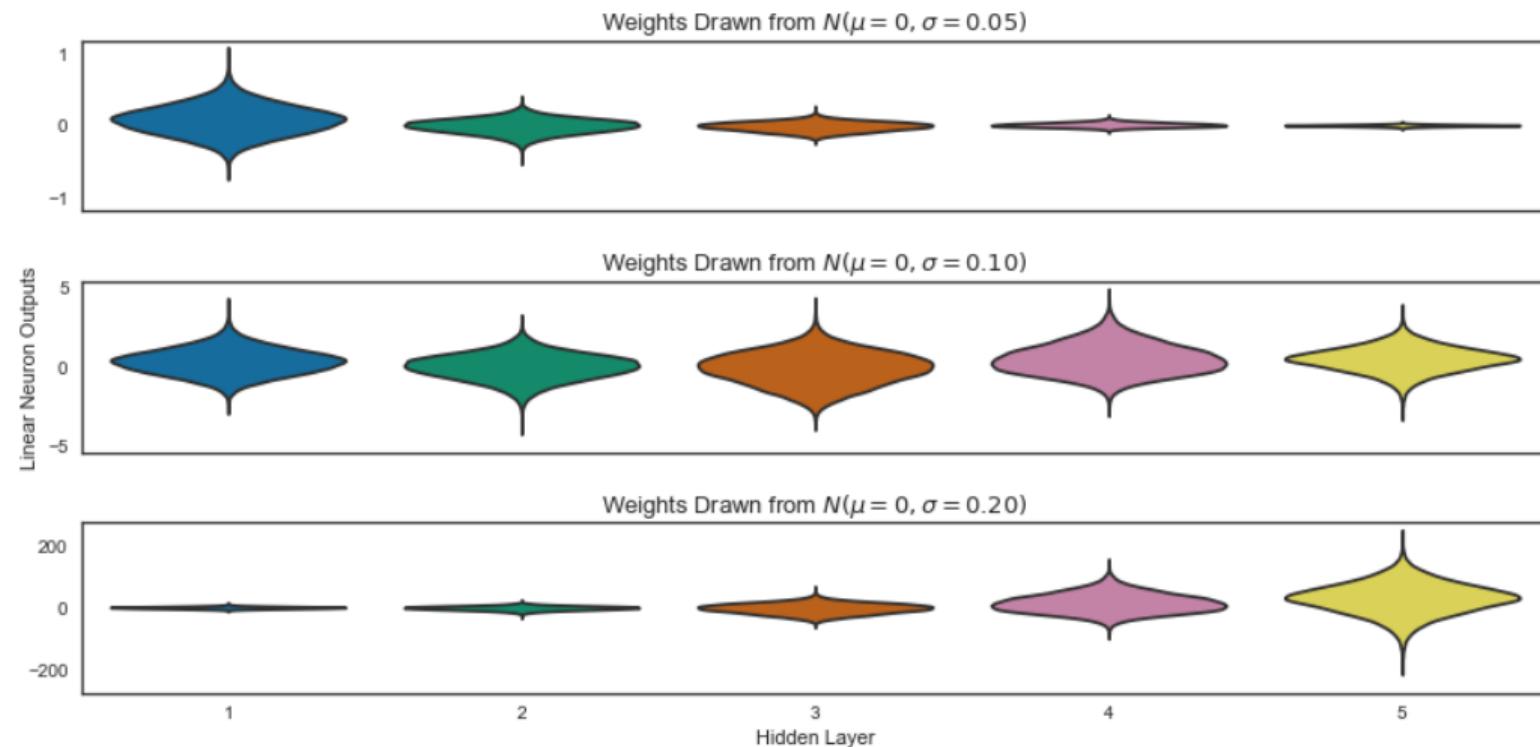
$$\frac{\partial L}{\partial w_3} = \frac{\partial L}{\partial \sigma_1} \sigma_1 (1 - \sigma_1) \sigma_3$$

- Хочется уничтожить симметрию
- Обычно инициализируют маленькими рандомными числами из какого-то распределения (нормальное, равномерное)

## Инициализация весов

- Наши признаки  $X$  пришли к нам из какого-то распределения
- Выход слоя  $f(XW)$  будет принадлежать другому распределению
- Если инициализировать веса неправильно, дисперсия распределения может от слоя к слою затухать (сигнал будет теряться) либо наоборот, возрастать (сигнал будет рассеиваться)
- Эмпирически было выяснено, что это может портить сходимость для глубоких сеток
- Хочется контролировать дисперсию

# Симметричный случай



# Инициализация весов

- Посмотрим на выход нейрона перед активацией:

$$h_i = \sum_{i=1}^{n_{in}} w_i x_i$$

- Дисперсия  $h_i$  выражается через дисперсии  $x$  и  $w$
- Будем считать, что веса  $w_1, \dots, w_k \sim iid$ , наблюдения  $x_1, \dots, x_n \sim iid$ , а ещё  $x_i$  и  $w_i$  независимы между собой

## Инициализация весов (симметричный случай)

$$\begin{aligned}Var(h_i) &= Var\left(\sum_{i=1}^{n_{in}} w_i x_i\right) = \sum_{i=1}^{n_{in}} Var(w_i x_i) = \\&= \sum_{i=1}^{n_{in}} [E(x_i)]^2 \cdot Var(w_i) + [E(w_i)]^2 \cdot Var(x_i) + Var(x_i) \cdot Var(w_i) =\end{aligned}$$

[https://en.wikipedia.org/wiki/Variance#Product\\_of\\_independent\\_variables](https://en.wikipedia.org/wiki/Variance#Product_of_independent_variables)

## Инициализация весов (симметричный случай)

$$\begin{aligned}Var(h_i) &= Var\left(\sum_{i=1}^{n_{in}} w_i x_i\right) = \sum_{i=1}^{n_{in}} Var(w_i x_i) = \\&= \sum_{i=1}^{n_{in}} [E(x_i)]^2 \cdot Var(w_i) + [E(w_i)]^2 \cdot Var(x_i) + Var(x_i) \cdot Var(w_i) =\end{aligned}$$

- Если функция активации симметричная, тогда  $E(x_i) = 0$ . Будем инициализировать веса с нулевым средним, тогда  $E(w_i) = 0$ .

## Инициализация весов (симметричный случай)

$$\begin{aligned}Var(h_i) &= Var\left(\sum_{i=1}^{n_{in}} w_i x_i\right) = \sum_{i=1}^{n_{in}} Var(w_i x_i) = \\&= \sum_{i=1}^{n_{in}} [E(x_i)]^2 \cdot Var(w_i) + [E(w_i)]^2 \cdot Var(x_i) + Var(x_i) \cdot Var(w_i) = \\&= \sum_{i=1}^{n_{in}} Var(x_i) \cdot Var(w_i)\end{aligned}$$

- Предполагаем, что  $x_i$  инициализируются из одного распределения,  $w_i$  сами инициализируем из одного распределения.

## Инициализация весов (симметричный случай)

$$\begin{aligned}Var(h_i) &= Var\left(\sum_{i=1}^{n_{in}} w_i x_i\right) = \sum_{i=1}^{n_{in}} Var(w_i x_i) = \\&= \sum_{i=1}^{n_{in}} [E(x_i)]^2 \cdot Var(w_i) + [E(w_i)]^2 \cdot Var(x_i) + Var(x_i) \cdot Var(w_i) = \\&= \sum_{i=1}^{n_{in}} Var(x_i) \cdot Var(w_i) = Var(x) \cdot [n_{in} \cdot Var(w)]\end{aligned}$$

- Хотим, чтобы зелёная штука была равна единице, тогда у потока будет всегда постоянная дисперсия, совпадающая с  $Var(x)$ .

## Инициализация весов (симметричный случай)

$$\begin{aligned}Var(h_i) &= Var\left(\sum_{i=1}^{n_{in}} w_i x_i\right) = \sum_{i=1}^{n_{in}} Var(w_i x_i) = \\&= \sum_{i=1}^{n_{in}} [E(x_i)]^2 \cdot Var(w_i) + [E(w_i)]^2 \cdot Var(x_i) + Var(x_i) \cdot Var(w_i) = \\&= \sum_{i=1}^{n_{in}} Var(x_i) \cdot Var(w_i) = Var(x) \cdot \underbrace{[n_{in} \cdot Var(w)]}_{=1}\end{aligned}$$

# Плохая инициализация весов

Пущай

$$w_i \sim U \left[ -\frac{1}{\sqrt{n_{in}}}; \frac{1}{\sqrt{n_{in}}} \right],$$

тогда

$$Var(w_i) = \frac{1}{12} \cdot \left( \frac{1}{\sqrt{n_{in}}} + \frac{1}{\sqrt{n_{in}}} \right)^2 = \frac{1}{3n_{in}} \Rightarrow Var(h_i) = \frac{1}{3}$$

Получаем затухание!

# Немного лучше

Пущай

$$w_i \sim U \left[ -\frac{\sqrt{3}}{\sqrt{n_{in}}}, \frac{\sqrt{3}}{\sqrt{n_{in}}} \right],$$

тогда

$$\text{Var}(w_i) = \frac{1}{12} \cdot \left( \frac{\sqrt{3}}{\sqrt{n_{in}}} + \frac{\sqrt{3}}{\sqrt{n_{in}}} \right)^2 = \frac{1}{n_{in}} \Rightarrow \text{Var}(h_i) = 1$$

# Немного лучше

Пущай

$$w_i \sim U \left[ -\frac{\sqrt{3}}{\sqrt{n_{in}}}, \frac{\sqrt{3}}{\sqrt{n_{in}}} \right],$$

тогда

$$\text{Var}(w_i) = \frac{1}{12} \cdot \left( \frac{\sqrt{3}}{\sqrt{n_{in}}} + \frac{\sqrt{3}}{\sqrt{n_{in}}} \right)^2 = \frac{1}{n_{in}} \Rightarrow \text{Var}(h_i) = 1$$

При forward pass на вход идёт  $n_{in}$  наблюдений, при backward pass на вход идёт  $n_{out}$  градиентов  $\Rightarrow$  канал с дисперсией может быть непостоянным, если число весов от слоя к слою сильно колеблется

# Инициализация Ксавье (Глорота)

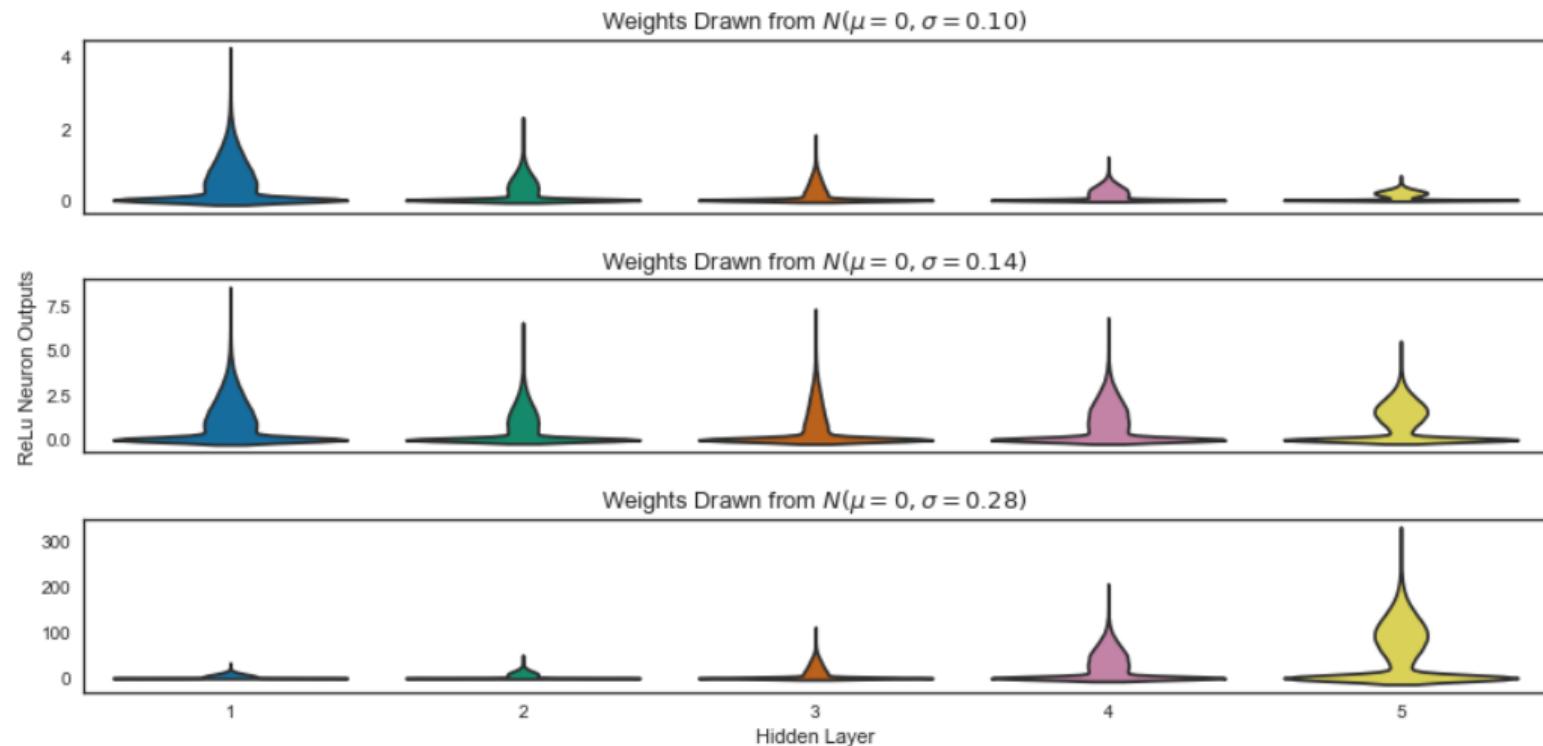
Для неодинаковых размеров слоёв невозможно удовлетворить обоим условиям, поэтому обычно усредняют и пытаются инициализировать веса с дисперсией  $\frac{2}{n_{in} + n_{out}}$ :

$$w_i \sim U \left[ -\frac{\sqrt{6}}{\sqrt{n_{out} + n_{in}}}; \frac{\sqrt{6}}{\sqrt{n_{out} + n_{in}}} \right],$$

Такая инициализация называется **инициализацией Ксавье (или Глорота)**

<http://proceedings.mlr.press/v9/glorot10a/glorot10a.pdf>

# Несимметричный случай



# Инициализация Хе

$$\begin{aligned}Var(h_i) &= Var\left(\sum_{i=1}^{n_{in}} w_i x_i\right) = \sum_{i=1}^{n_{in}} Var(w_i x_i) = \\&= \sum_{i=1}^{n_{in}} [E(x_i)]^2 \cdot Var(w_i) + [E(w_i)]^2 \cdot Var(x_i) + Var(x_i) \cdot Var(w_i)\end{aligned}$$

- Когда нет симметрии, можно занулить только второе слагаемое

# Инициализация Хе

$$\begin{aligned}Var(h_i) &= Var\left(\sum_{i=1}^{n_{in}} w_i x_i\right) = \sum_{i=1}^{n_{in}} Var(w_i x_i) = \\&= \sum_{i=1}^{n_{in}} [E(x_i)]^2 \cdot Var(w_i) + [E(w_i)]^2 \cdot Var(x_i) + Var(x_i) \cdot Var(w_i) = \\&= \sum_{i=1}^{n_{in}} [E(x_i)]^2 \cdot Var(w_i) + Var(x_i) \cdot Var(w_i) = \sum_{i=1}^{n_{in}} Var(w_i) \cdot E(x_i^2)\end{aligned}$$

- Когда нет симметрии, можно занулить только второе слагаемое

# Инициализация Хе

$$\begin{aligned}Var(h_i) &= Var\left(\sum_{i=1}^{n_{in}} w_i x_i\right) = \sum_{i=1}^{n_{in}} Var(w_i x_i) = \\&= \sum_{i=1}^{n_{in}} [E(x_i)]^2 \cdot Var(w_i) + [E(w_i)]^2 \cdot Var(x_i) + Var(x_i) \cdot Var(w_i) = \\&= \sum_{i=1}^{n_{in}} [E(x_i)]^2 \cdot Var(w_i) + Var(x_i) \cdot Var(w_i) = \sum_{i=1}^{n_{in}} Var(w_i) \cdot E(x_i^2) = \\&= E(x^2) \cdot [n_{in} \cdot Var(w)]\end{aligned}$$

# Инициализация Хе

$$\begin{aligned}Var(h_i) &= E(x_i^2) \cdot [n_{in} \cdot Var(w)] \\x_i &= max(0; h_{i-1})\end{aligned}$$

# Инициализация Хе

$$\begin{aligned}Var(h_i) &= E(x_i^2) \cdot [n_{in} \cdot Var(w)] \\x_i &= \max(0; h_{i-1})\end{aligned}$$

Пусть  $w_{i-1}$  распределены симметрично относительно нуля, тогда  $h_{i-1}$  тоже будут симметрично распределены, тогда:

# Инициализация Хе

$$\begin{aligned}Var(h_i) &= E(x_i^2) \cdot [n_{in} \cdot Var(w)] \\x_i &= \max(0; h_{i-1})\end{aligned}$$

Пусть  $w_{i-1}$  распределены симметрично относительно нуля, тогда  $h_{i-1}$  тоже будут симметрично распределены, тогда:

$$E(x_i^2) = \frac{1}{2} \cdot Var(h_{i-1})$$

## Инициализация Хе

$$\begin{aligned}Var(h_i) &= E(x_i^2) \cdot [n_{in} \cdot Var(w)] \\x_i &= \max(0; h_{i-1})\end{aligned}$$

Пусть  $w_{i-1}$  распределены симметрично относительно нуля, тогда  $h_{i-1}$  тоже будут симметрично распределены, тогда:

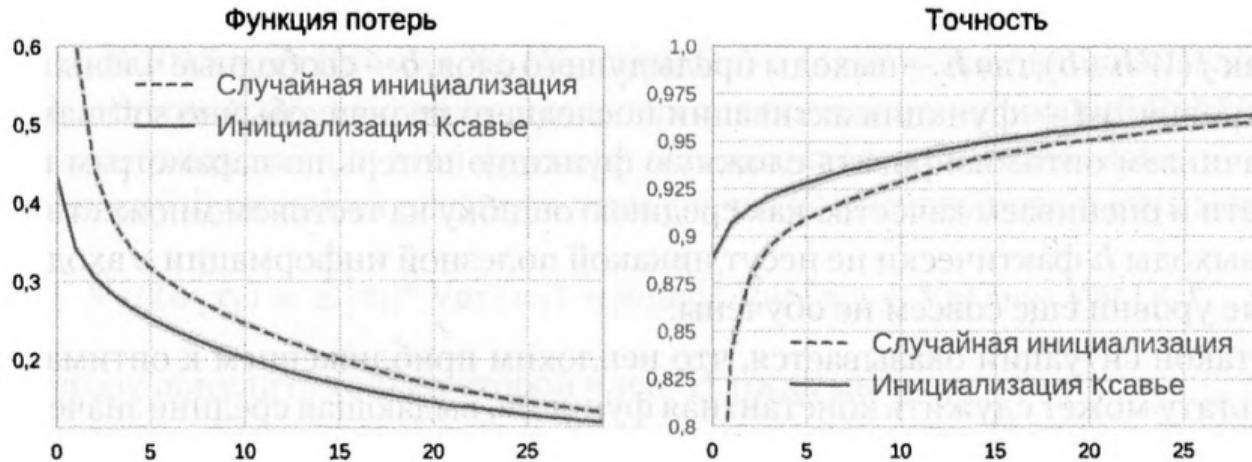
$$E(x_i^2) = \frac{1}{2} \cdot Var(h_{i-1})$$

$$Var(h_i) = \frac{1}{2} \cdot Var(h_{i-1}) \cdot [n_{in} \cdot Var(w)] \Rightarrow Var(w_i) = \frac{2}{n_{in}}$$

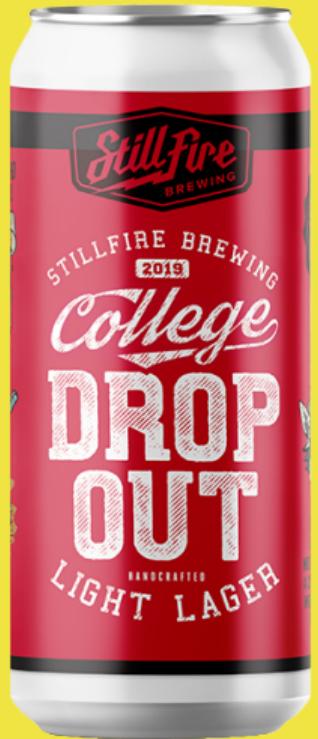
## Краткие итоги

- Для симметричных функций с нулевым средним используйте инициализацию Ксавье (Глорота) `init="glorot_uniform"`
- Для ReLU и им подобным инициализацию Хе `init="he_uniform"` или `init="he_normal"`
- Эти две инициализации корректируют параметры распределений в зависимости от входа и выхода слоя так, чтобы поддерживать дисперсию равной единице

# Эксперимент с MNIST



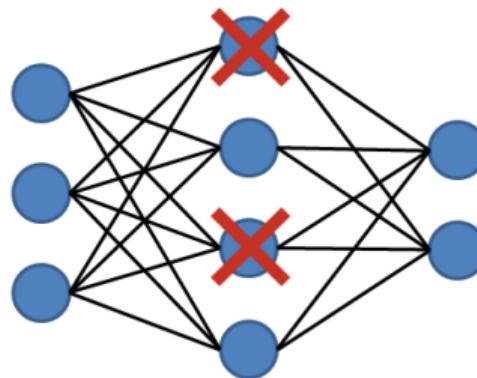
Источник: Николенко, страница 149





# Dropout

- Оставляем нейрон с вероятностью  $p$ , иначе отключаем (заменяем на 0)
- Делает нейроны более устойчивыми к случайным возмущениям
- Борьба с ко-адаптацией, не все соседи похожи, не все дети похожи на родителей



# Dropout в формулах

- forward pass:

$$o = f(X \cdot W + b)$$

# Dropout в формулах

- forward pass:

$$o = f(X \cdot W + b)$$

$$o = D \cdot f(X \cdot W + b), \quad D = (D_1, \dots, D_k) \sim iidBern(p)$$

# Dropout в формулах

- forward pass:

$$o = f(X \cdot W + b)$$

$$o = D \cdot f(X \cdot W + b), \quad D = (D_1, \dots, D_k) \sim iidBern(p)$$

$$o_i = D_i \cdot f(wx_i^T + b) = \begin{cases} f(wx_i^T + b), & 1 - p \\ 0, & p \end{cases}$$

Дропаут – это просто небольшая модификация функции активации

# Dropout в формулах

- forward pass:

$$o = f(X \cdot W + b)$$

$$o = D \cdot f(X \cdot W + b), \quad D = (D_1, \dots, D_k) \sim iidBern(p)$$

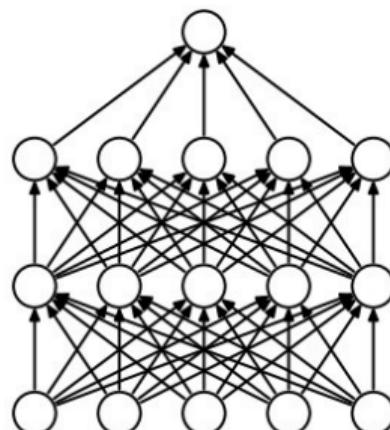
- backward pass:

$$d = f'(h) \cdot W \cdot d$$

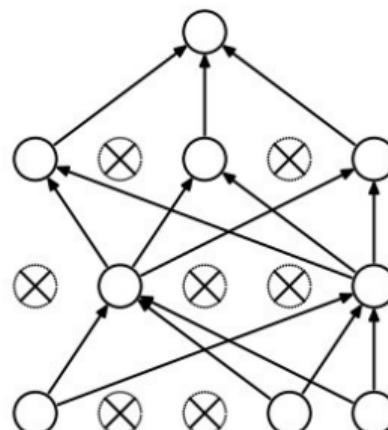
$$d = D \cdot f'(h) \cdot W \cdot d$$

# Dropout

- При обучении мы домножаем часть выходов на  $D_i$ , тем самым мы изменяем только часть параметров и нейроны учатся более независимо  $\Rightarrow$  регуляризация
- Dropout эквивалентен обучению  $2^n$  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  $\Rightarrow$  регуляризация
- Dropout эквивалентен обучению  $2^n$  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  $\Rightarrow$  регуляризация
- Dropout эквивалентен обучению  $2^n$  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  $\Rightarrow$  регуляризация
- Dropout эквивалентен обучению  $2^n$  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

$$o = p \cdot f(X \cdot W + b)$$

# Обратный Dropout

- На тесте ищем математическое ожидание:

$$o = p \cdot f(X \cdot W + b)$$

# Обратный Dropout

- На тесте ищем математическое ожидание:

$$o = p \cdot f(X \cdot W + b)$$

- Это неудобно! Надо переписывать функцию для прогнозов!

# Обратный Dropout

- На тесте ищем математическое ожидание:

$$o = p \cdot f(X \cdot W + b)$$

- Это неудобно! Надо переписывать функцию для прогнозов!
- Давайте лучше будем домножать на  $\frac{1}{p}$  на этапе обучения:

**train:**  $o = \frac{1}{p} \cdot D \cdot f(X \cdot W + b)$

**test:**  $o = f(X \cdot W + b)$

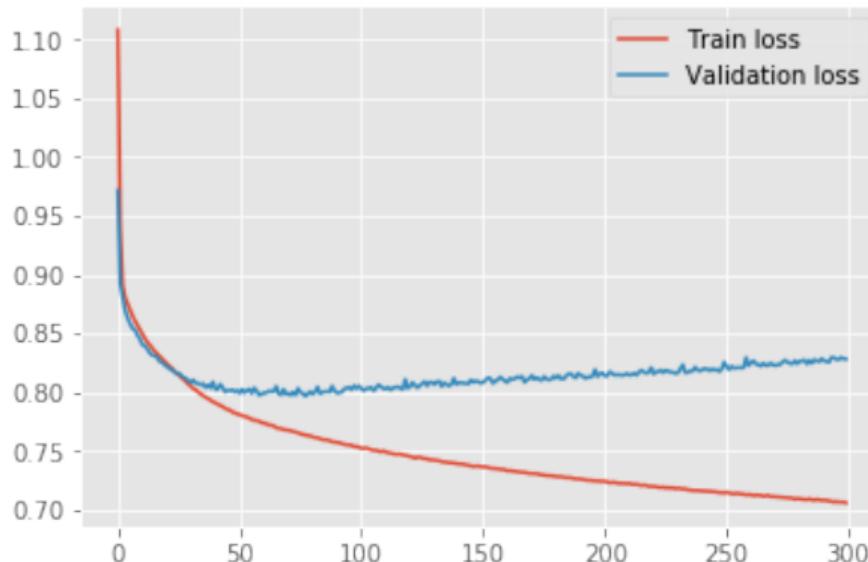


# Другая регуляризация

## Уже обсуждали

- $l_1$  и  $l_2$  регуляризация
- Ранняя остановка
- Различные новые градиентные спуски, ускоряющие процедуру сходимости

# Early stopping



- Будем останавливать обучение, когда качество на валидации начинает падать

# Регуляризация

- $L_2$ : приплюсовываем к функции потерь  $\lambda \cdot \sum w_i^2$
- $L_1$ : приплюсовываем к функции потерь  $\lambda \cdot \sum |w_i|$
- Можно регуляризовать не всю сетку, а отдельный нейрон или слой
- Не даёт нейрону сфокусироваться на слишком выделяющемся входе

# Регуляризация

- В keras можно добавить для каждого слоя на три вида связей:
- **kernel\_regularizer** - на матрицу весов слоя;
- **bias\_regularizer** - на вектор свободных членов;
- **kernel\_regularizer** - на вектор выходов.
- **Делается примерно так:**

```
model.add(Dense(256, input_dim = 32,  
               kernel_regularizer = regularizers.l1(0.001),  
               bias_regularizer = regularizers.l2(0.1),  
               activity_regularizer = regularizers.l2(0.01)))
```

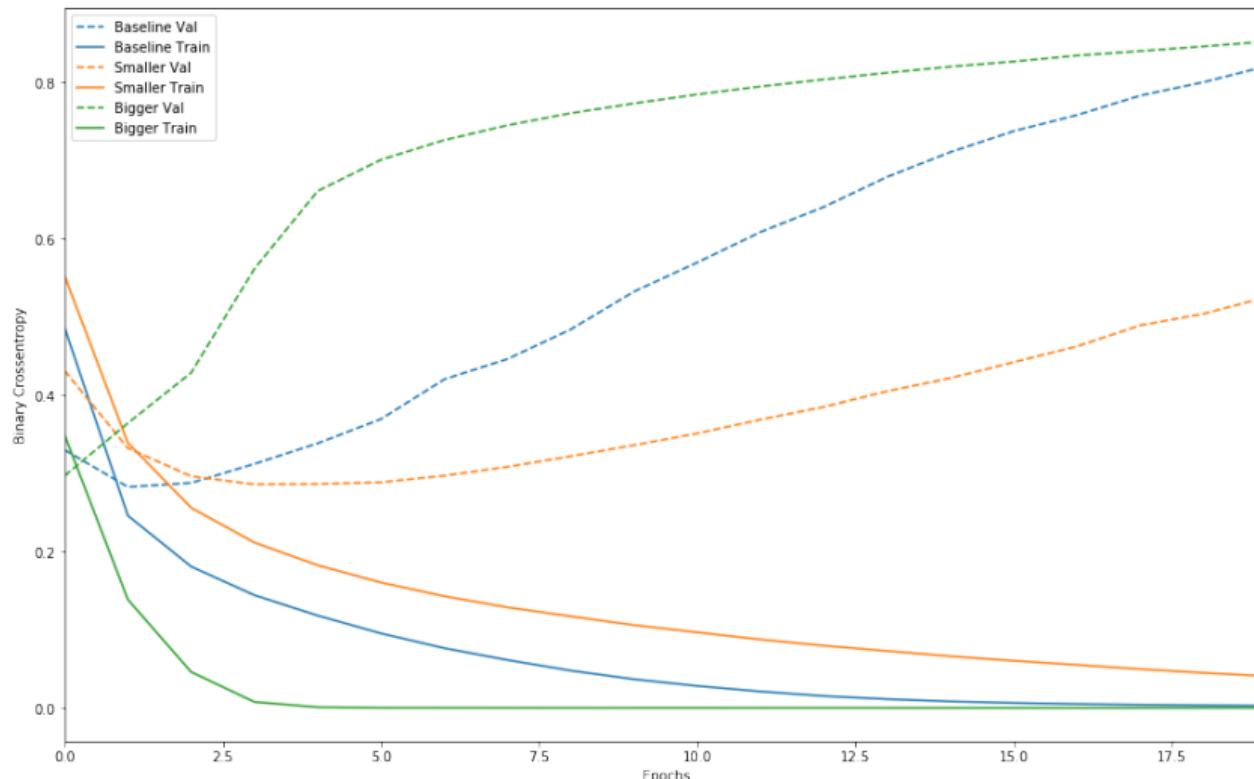
## Взаимосвязи

- На практике обычно используют Dropout. Действия всех этих регуляризаторов оказывается схожим:
- Например, в [1] написано:

«We show that the dropout regularizer is first-order equivalent to an L2 regularizer applied after scaling the features by an estimate of the inverse diagonal Fisher information matrix»
- У Гудфеллоу в Глубоком обучении на стр. 218 можно найти, что ранняя остановка для линейных моделей эквивалентна  $l_2$  регуляризации с MSE, обучаемой SGD.

[1] <https://arxiv.org/abs/1307.1493>

# Размеры сеток и переобучение



# Предобучение

- Обучаем каждый нейрон на рандомной подвыборке, каждый нейрон впитает какие-то отдельные её особенности, после скрепляем все нейроны вместе и продолжаем обучение на всей выборке
- **На будущее:** обучаем на корпусе картинок автокодировщик, encoder благодаря этому учится выделять наиболее важные фичи, которые позволяют эффективно сжимать изображения. После срезаем decoder и на его месте достраиваем слои для решения нашей задаче, запускаем обычное дообучение.

## Динамическое наращивание сети

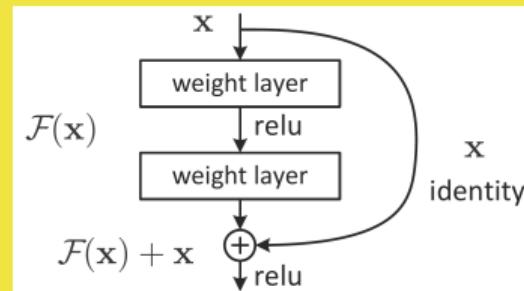
- Обучение сети при заведомо недостаточном числе нейронов  $H$
- После стабилизации функции потерь – добавление нового нейрона и его инициализация путём обучения
  - либо по случайной подвыборке
  - либо по объектам с наибольшими значениями потерь
  - либо по случайному подмножеству входов
  - либо из различных случайных начальных приближений
- Снова итерации BackProp

**Эмпирический опыт:** Общее время обучения обычно лишь в 1.5 – 2 раза больше, чем если бы в сети сразу было итоговое число нейронов. Полезная информация, накопленная сетью не теряется при добавлении нейронов.

# Прореживание сети

- Начать с большого количества нейронов и удалять незначимые по какому-нибудь критерию
- Пример: обнуляем вес, смотрим как сильно упала ошибка, сортируем все связи по этому критерию, удаляем  $N$  наименее значимых
- После прореживания снова запускаем backprop
- Если качество модели сильно упала, надо вернуть последние удалённые связи

# Skip-connection и ResNet

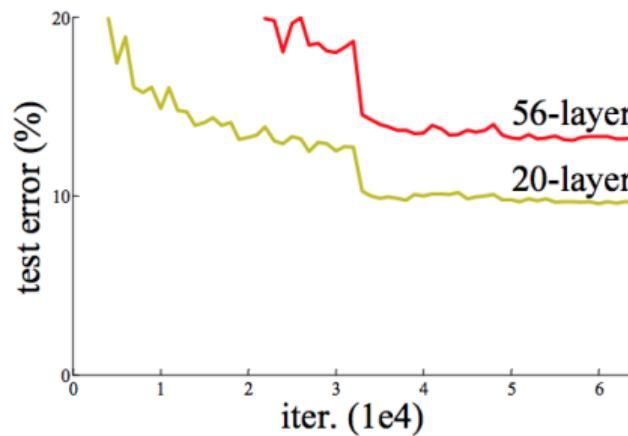
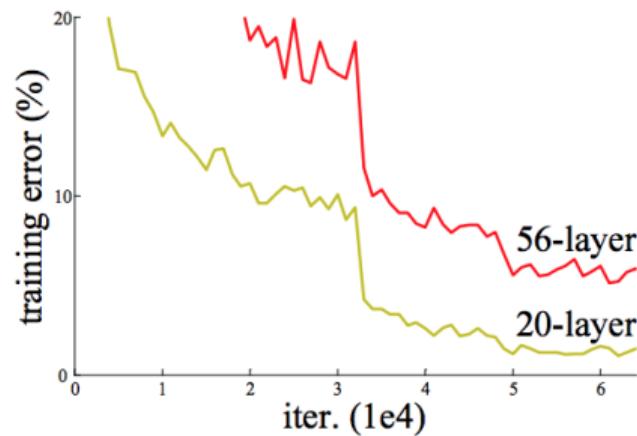


Очень глубокие сети



# Идея ResNet

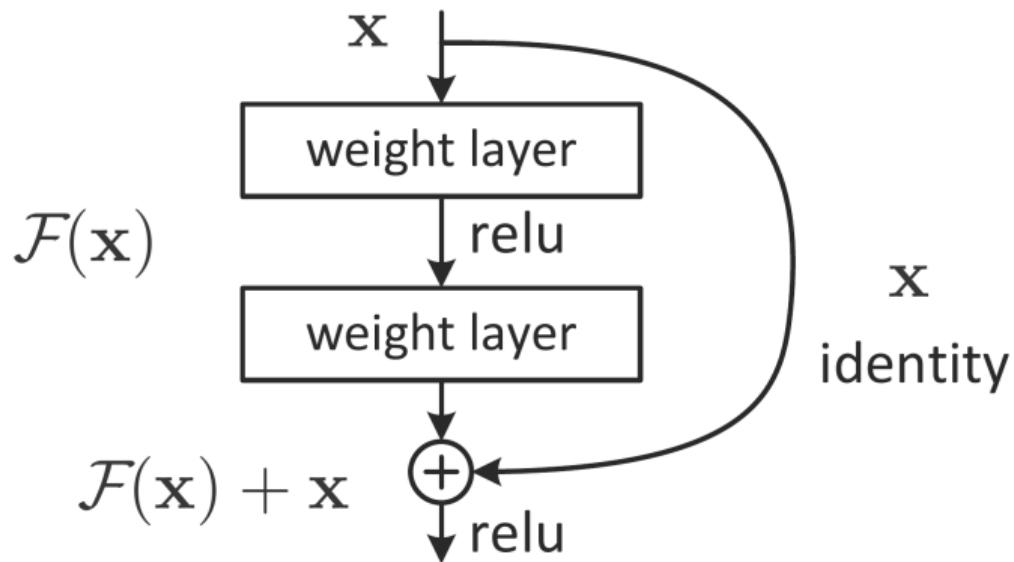
Чем глубже нейронная сеть, тем сложнее её обучать, возникает новая проблема, которую нельзя свести к переобучению или затуханию градиента. Её называют деградация обучения (training degradation)



## Идея ResNet

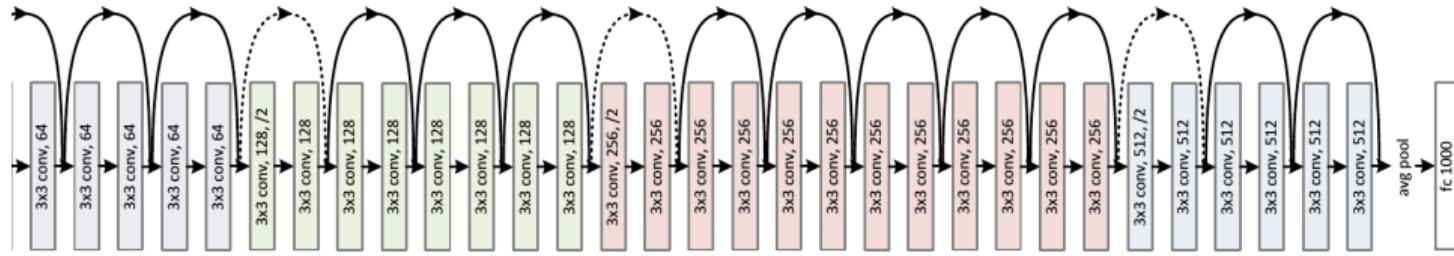
- Огромная свёрточная сеть из 20 слоёв и свёрточная сеть из 56 слоёв, большая сетка обучается хуже
- **Проблема:** слои инициализированы шумом, если какой-то один слой не натренирован, он убивает работу сети, через него не проходит полезный сигнал
- Чем больше слоёв, тем более ярко выражен этот эффект
- **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)



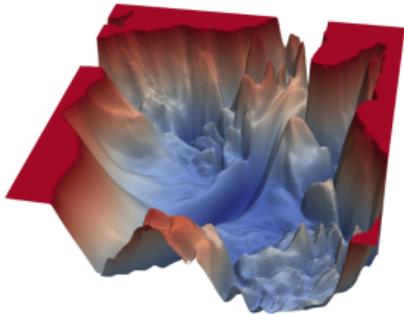
<https://arxiv.org/abs/1512.03385>

## ResNet (2015)

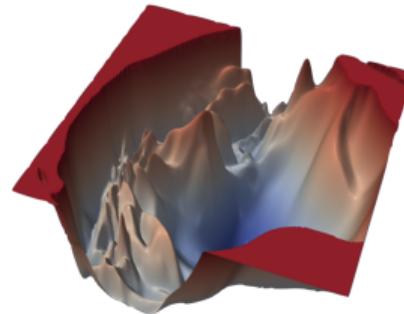
- **Идея:** более глубокие уровни должны улавливать разницу между новым и тем, что было раньше
- Ключевым элементом архитектуры является связь, которая пропускает несколько слоёв, передавая результат предыдущего слоя
- Такое изменение позволило полностью отказаться от таких техник регуляризации, как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

# Визуализация потерь

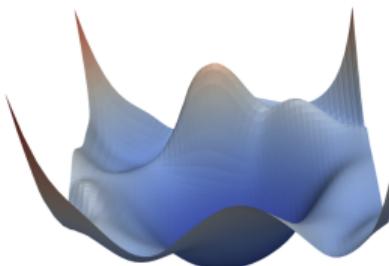
VGG-56



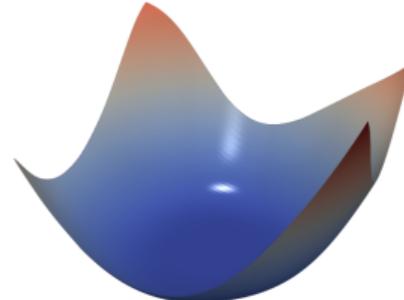
VGG-110



Renset-56



Densenet-121



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

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

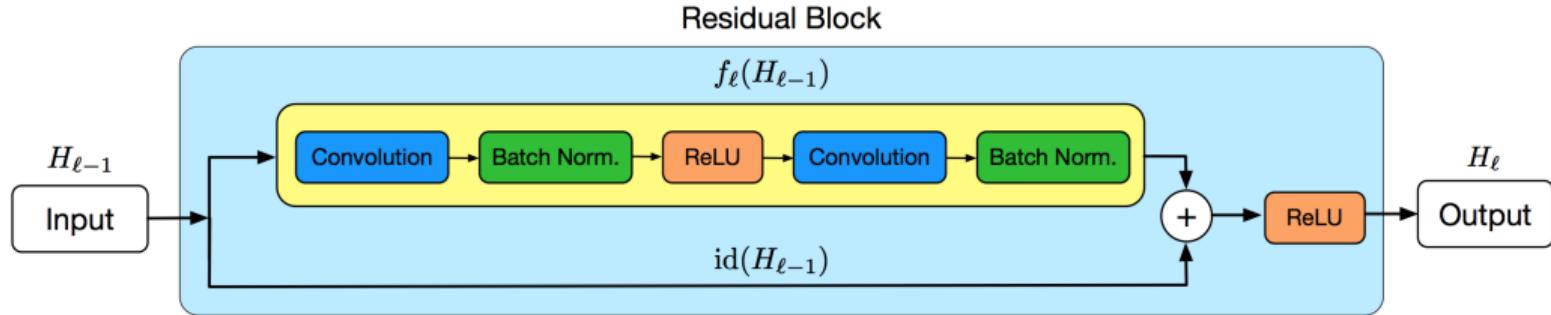
# ResNet

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

# Нейросети со стохастической глубиной



# Нейросети со стохастической глубиной

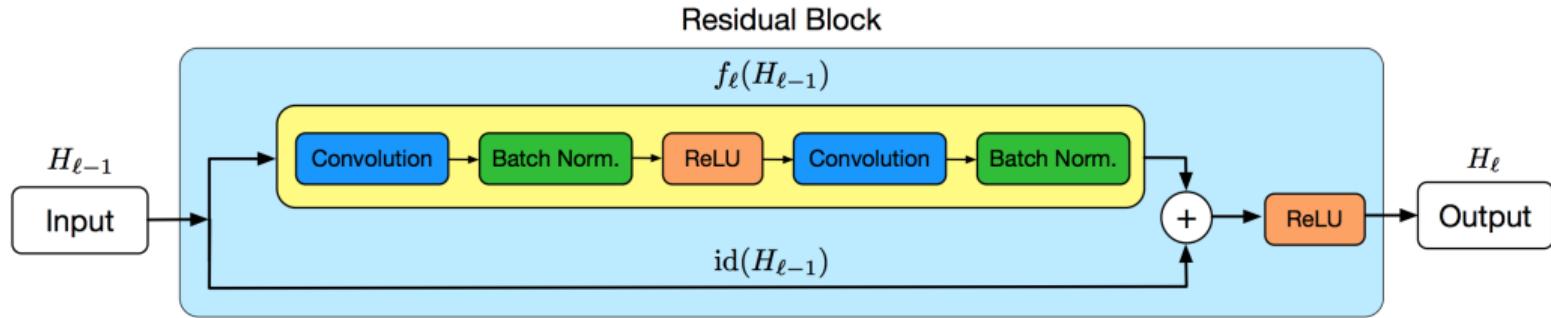


Обычный RESNET-блок:

$$H_l = \text{ReLU}(f_l(H_{l-1}) + H_{l-1})$$

<https://arxiv.org/abs/1603.09382>

# Нейросети со стохастической глубиной



Немного видоизменим его:

$$b_l \sim \text{Bern}(p_l)$$

$$H_l = \text{ReLU}(b_l \cdot f_l(H_{l-1}) + H_{l-1})$$

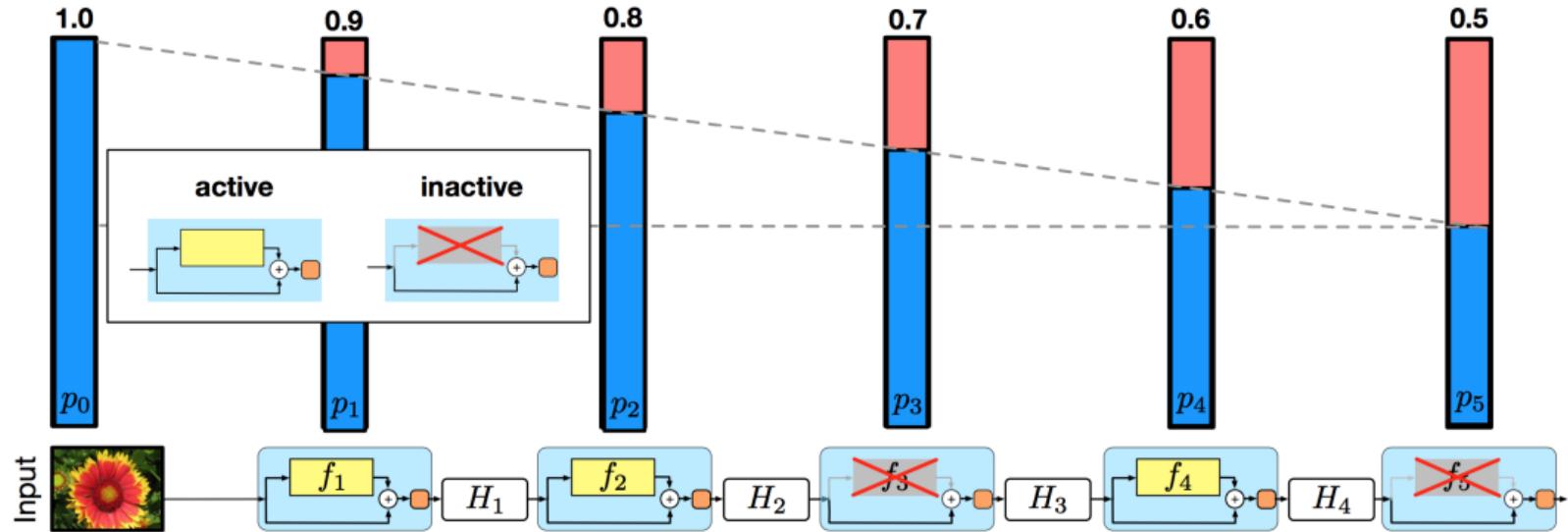
<https://arxiv.org/abs/1603.09382>

# Нейросети со стохастической глубиной

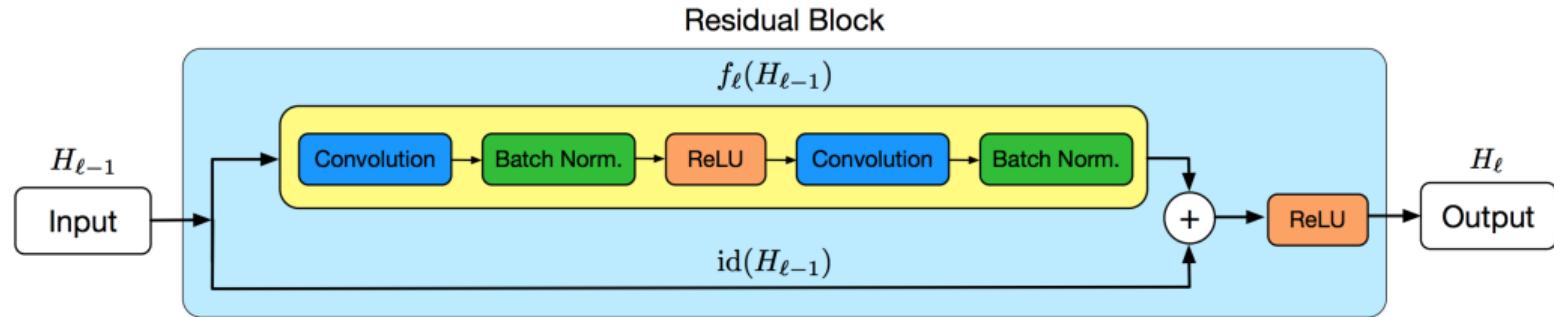
- Для слоя  $l$  мы задаём вероятность  $p_l$ , которая сохраняет этот слой в нейронной сетке
- Если  $b_l = 1$ , у нас остаётся обычный RESNET-блок, если  $b_l = 0$  остаётся только skip-connection
- Глубина сети зависит от того, как сгенерируется случайная величина
- Как подобрать  $p_l$ ?
- Первые слои важнее, так как они создают скрытые представления

# Нейросети со стохастической глубиной

$$p_l = 1 - \frac{l}{L} \cdot (1 - p_L), \quad p_L = \text{const} \in [0; 1]$$



# Этап предсказания

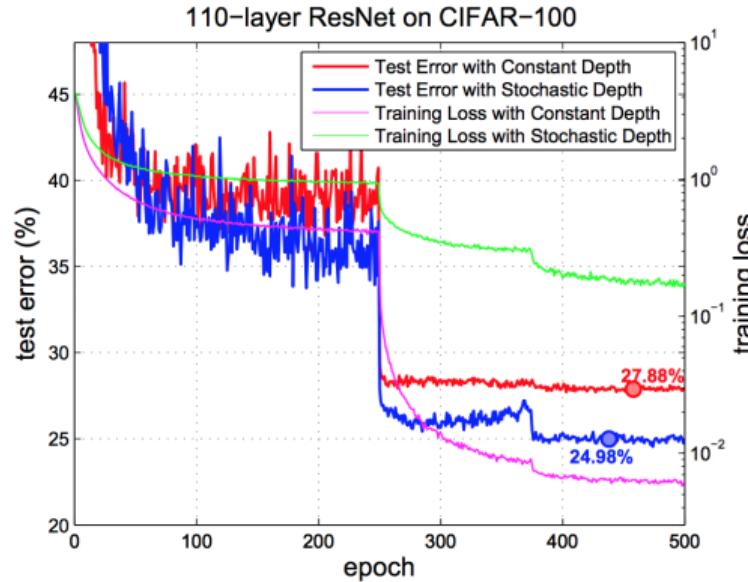
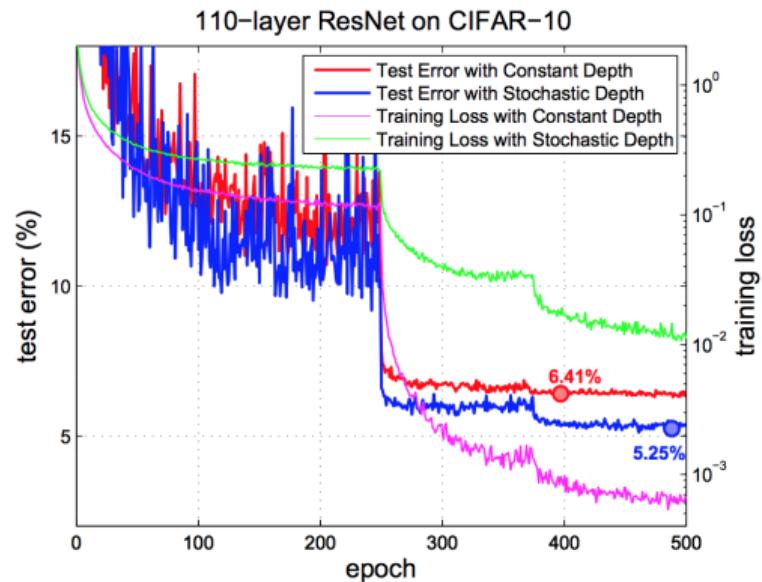


На этапе предсказания, пользуемся приёмом из Dropout.

$$H_l^{test} = \text{ReLU}(p_l \cdot f_l(H_{l-1}^{test}) + H_{l-1}^{test})$$

<https://arxiv.org/abs/1603.09382>

# Нейросети со стохастической глубиной



<https://arxiv.org/abs/1603.09382>

# Нейросети со стохастической глубиной

- Сетка показывает более плохие показатели на тренировочной выборке, но выигрывает на тестовой
- Обучение из-за отсутствия некоторых блоков идёт быстрее
- Параметр  $p_L$  отвечает за агрессивность выкидывания блоков, эксперименты показывают наилучшее качество сетки при  $p_L = 0.5$

<https://arxiv.org/abs/1603.09382>