Linear, BatchNorm, Dropout

Блуменау Марк, магистратура ИИ

Что мы хотим?

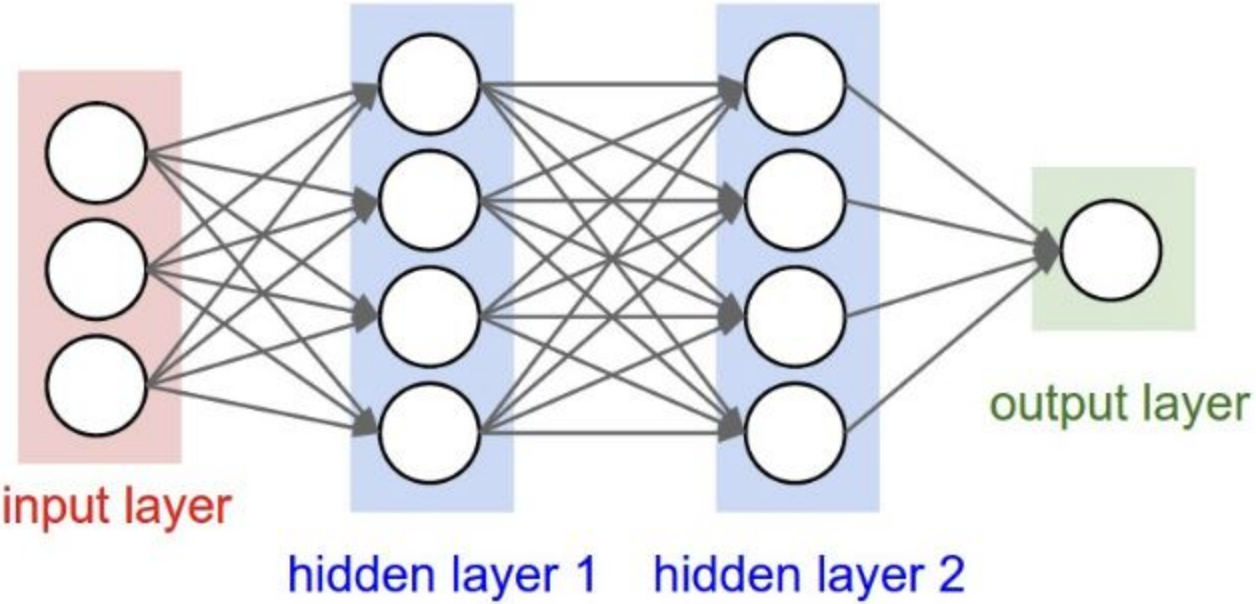
Имея какую-то функцию ошибок, например MSE:

$$L\left(y_i, a\left(x_i\right)\right) = \left(a\left(x_i, w\right) - y_i\right)^2$$

Наша цель:

$$\frac{1}{\ell}\sum_{i=1}^{\ell} L\left(y_i, a\left(x_i\right)\right) \to \min_{\mathbf{a}}$$

Полносвязный (=линейный=Linear PyTorch) слой



https://cs231n.github.io/convolutional-networks/

Что там на самом деле?

Каждый слой - набор линейных моделей

$$z_j = \sum_{i=1}^{n} w_{ji} x_i + b_j$$

Обычно слоев несколько. Сколько параметров в одном слое?

(Количество входов (=кол-во x) + 1) * Количество выходов (=кол-во z), которое хотим

То есть для n входов и m выходов: m(n+1)

Отлично, давайте возьмем два линейных слоя

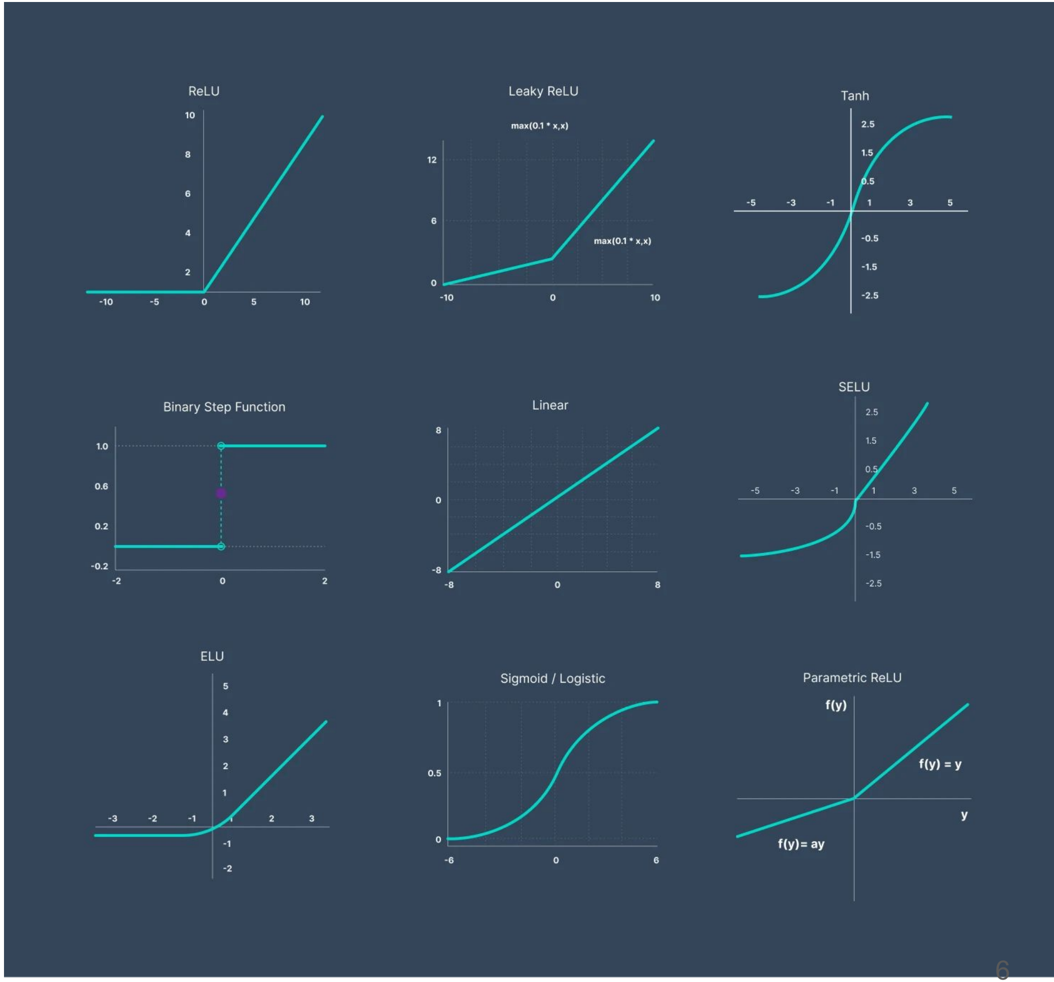
$$s_k = \sum_{j=1}^{m} v_{kj} z_j + c_k = \sum_{j=1}^{m} v_{kj} \sum_{i=1}^{n} w_{ji} x_i + \sum_{j=1}^{m} v_{kj} b_j + c_k =$$

$$= \sum_{j=1}^{m} \left(\sum_{i=1}^{n} v_{kj} w_{ji} x_i + v_{kj} b_j + \frac{1}{m} c_k \right)$$

Кажется работает как один линейный слой

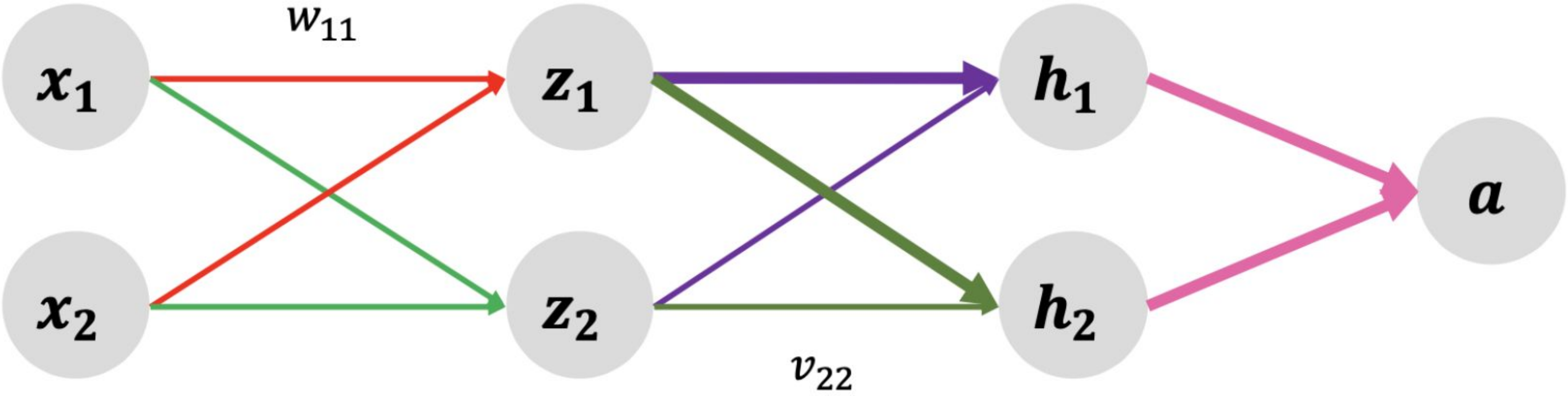
А как это исправить?

Нелинейные функции!
Желательные свойства:
1) Непрерывная дифференцируемость
2) Монотонность
3) Ограниченность
4) Близка к линейной около
начала координат



https://www.v7labs.com/blog/neural-networks-activation-functions

Обратное распространение ошибки (Backpropagation)



$$\frac{\partial}{\partial w_j}\left(a\left(x_i, w\right) - y_i\right)^2 = 2\left(a\left(x_i, w\right) - y_i\right)\frac{\partial}{\partial w_j} a\left(x_i, w\right)$$

$$\frac{\partial a}{\partial w_{11}} = \frac{\partial a}{\partial h_1}\frac{\partial h_1}{\partial z_1}\frac{\partial z_1}{\partial w_{11}} + \frac{\partial a}{\partial h_2}\frac{\partial h_2}{\partial z_1}\frac{\partial z_1}{\partial w_{11}}$$

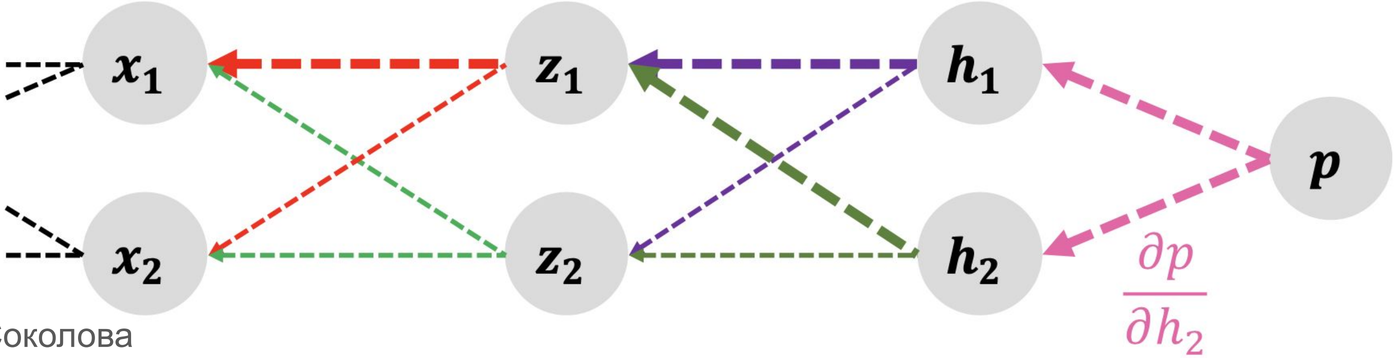
3: $\frac{\partial p}{\partial h_1}$ $\quad$ $\frac{\partial p}{\partial h_2}$

2: $\frac{\partial p}{\partial z_1} = \frac{\partial p}{\partial h_1}\frac{\partial h_1}{\partial z_1} + \frac{\partial p}{\partial h_2}\frac{\partial h_2}{\partial z_1}$ $\quad$ $\frac{\partial p}{\partial z_2} = \frac{\partial p}{\partial h_1}\frac{\partial h_1}{\partial z_2} + \frac{\partial p}{\partial h_2}\frac{\partial h_2}{\partial z_2}$

1:

$$\frac{\partial p}{\partial x_1} = \frac{\partial p}{\partial h_1}\frac{\partial h_1}{\partial z_1}\frac{\partial z_1}{\partial x_1} + \frac{\partial p}{\partial h_2}\frac{\partial h_2}{\partial z_1}\frac{\partial z_1}{\partial x_1} + \frac{\partial p}{\partial h_1}\frac{\partial h_1}{\partial z_2}\frac{\partial z_2}{\partial x_1} + \frac{\partial p}{\partial h_2}\frac{\partial h_2}{\partial z_2}\frac{\partial z_2}{\partial x_1}$$

$$\frac{\partial p}{\partial x_2} = \frac{\partial p}{\partial h_1}\frac{\partial h_1}{\partial z_1}\frac{\partial z_1}{\partial x_2} + \frac{\partial p}{\partial h_2}\frac{\partial h_2}{\partial z_1}\frac{\partial z_1}{\partial x_2} + \frac{\partial p}{\partial h_1}\frac{\partial h_1}{\partial z_2}\frac{\partial z_2}{\partial x_2} + \frac{\partial p}{\partial h_2}\frac{\partial h_2}{\partial z_2}\frac{\partial z_2}{\partial x_2}$$



Из лекций Е. Соколова

Теперь узнаем, почему PyTorch

Keras

Меньше кода

Напоминает конструктор

Готовый метод fit

Статический граф вычислений*

Вычисления могут быть оптимизированы заранее

PyTorch

Больше гибкости

Тонкая настройка каждого шага обучения

Динамический граф вычислений

Можно модифицировать модель во время исполнения

Теорема Цыбенко (Универсальная теорема аппроксимации)

Нестрогое изложение:

Имея некую непрерывную функцию f(x), мы можем построить такую двухслойную нейронную сеть, что будем приближать f(x) с любой заранее заданной точностью.

=> полносвязные слои очень мощный инструмент

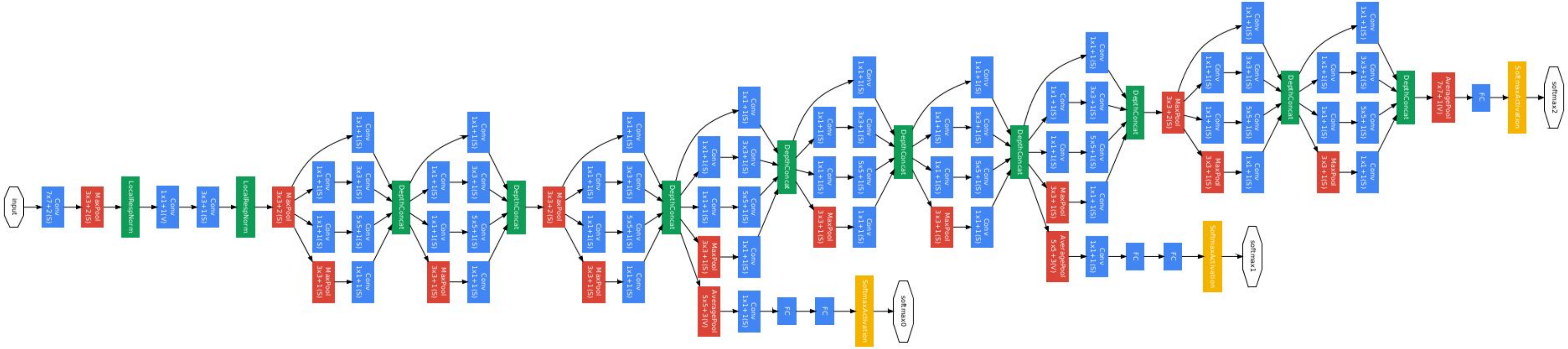
What is the catch?

“Ширину” такой сети придется очень сильно увеличивать

https://web.archive.org/web/20151010204407/http://deeplearning.cs.cmu.edu/pdfs/Cybenko.pdf

Как избежать взрывного роста количества параметров?

Наращивать глубину, типичная нейронная сеть:

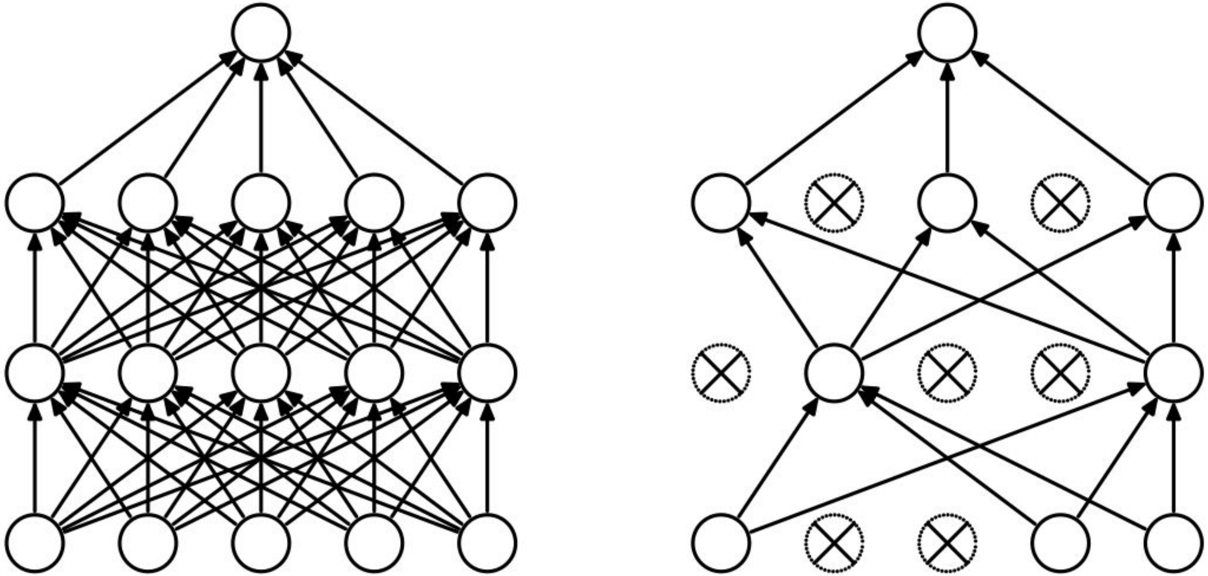


https://medium.com/swlh/understanding-inception-simplifying-the-network-architecture-54cd31d38949 https://jmlr.org/papers/volume15/srivastava14a/srivastava14a.pdf

Как избежать переобучения?

1) Weight_decay, см. прошлую лекцию

2)



(a) Standard Neural Net (b) After applying dropout.

Принцип работы (PyTorch)

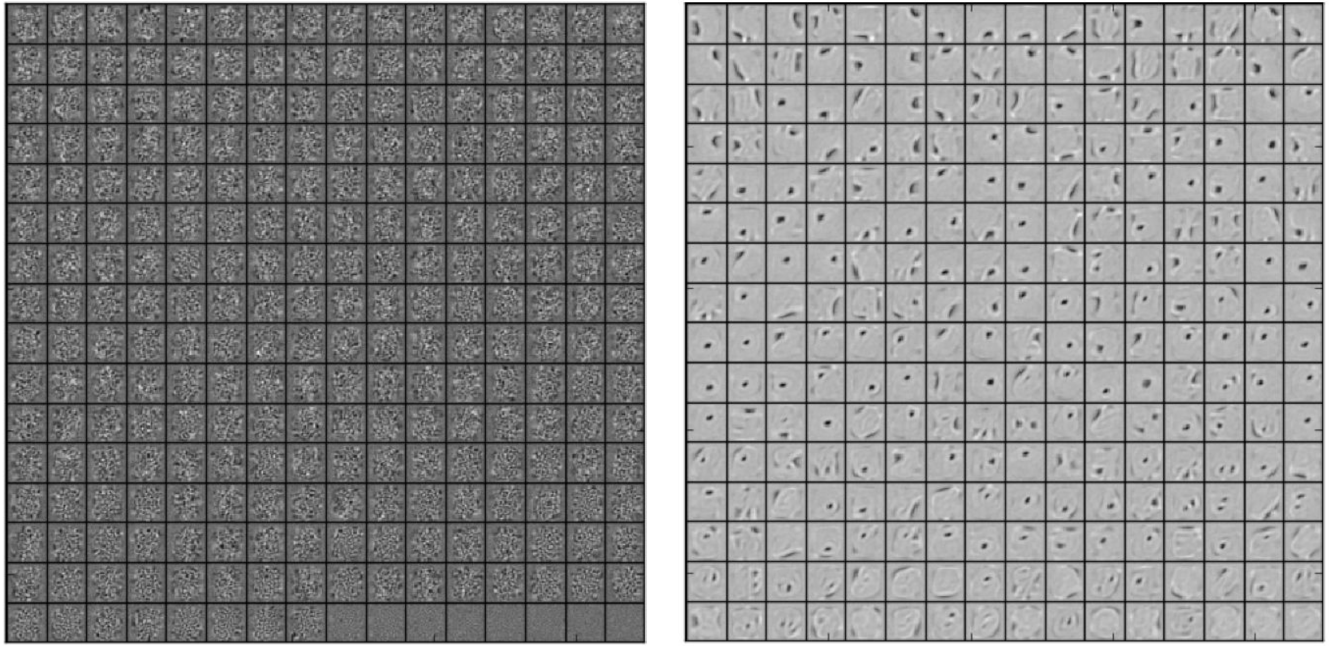
Train mode

1) Для каждого нейрона нулим его выход с вероятностью **p** (подброс монетки)
2) После проделанного домножаем все выходы нейронов на 1/(1-p)

Eval mode

1) Отдыхаем (просто домножаем все выходы на 1)

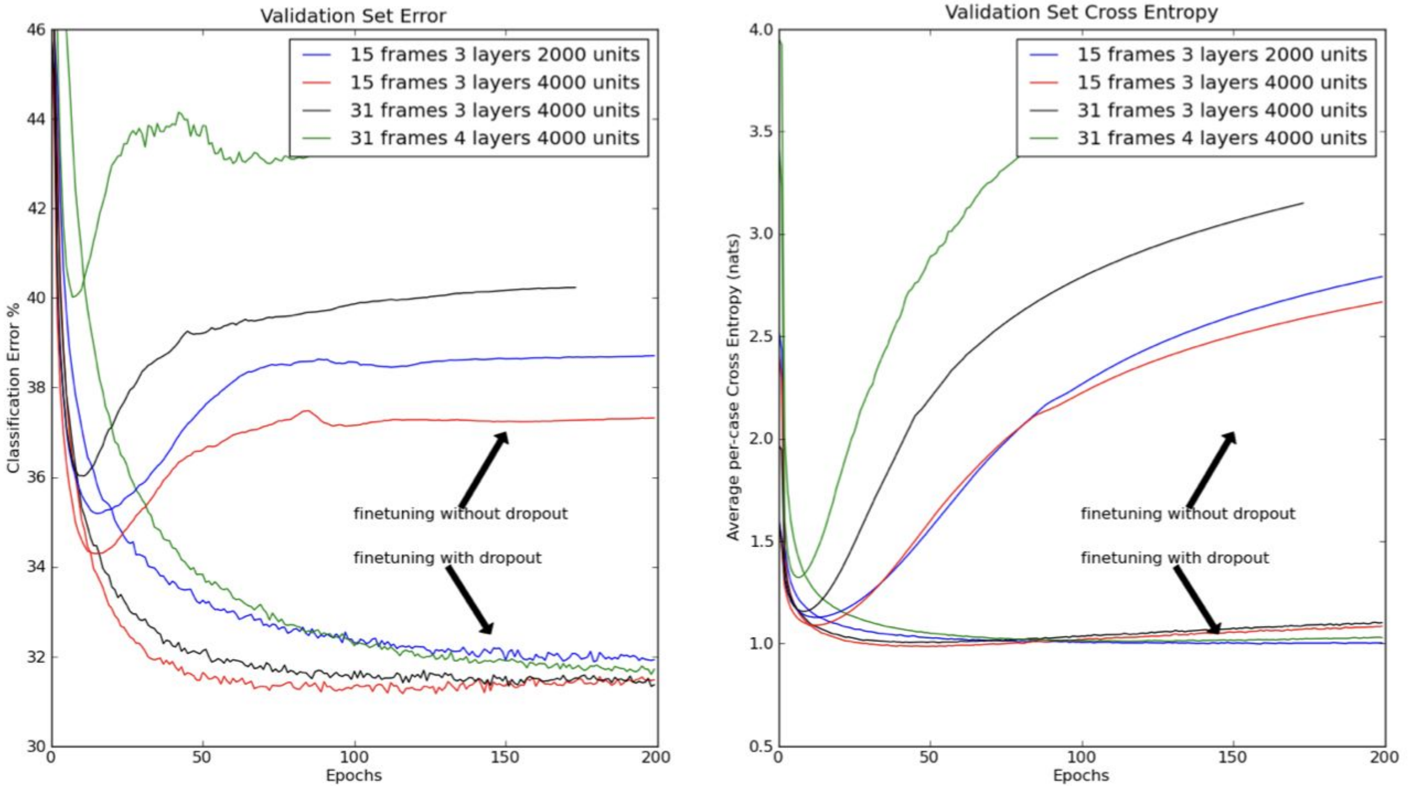
Польза невооруженным взглядом



(a) Without dropout (b) Dropout with $p = 0.5$.

https://jmlr.org/papers/volume15/srivastava14a/srivastava14a.pdf

При дообучении тоже влияет



https://arxiv.org/pdf/1207.0580

А как ещё стабилизировать обучение?

Batch Normalization

$$y = \frac{x - \mathrm{E}[x]}{\sqrt{\mathrm{Var}[x] + \epsilon}} * \gamma + \beta$$

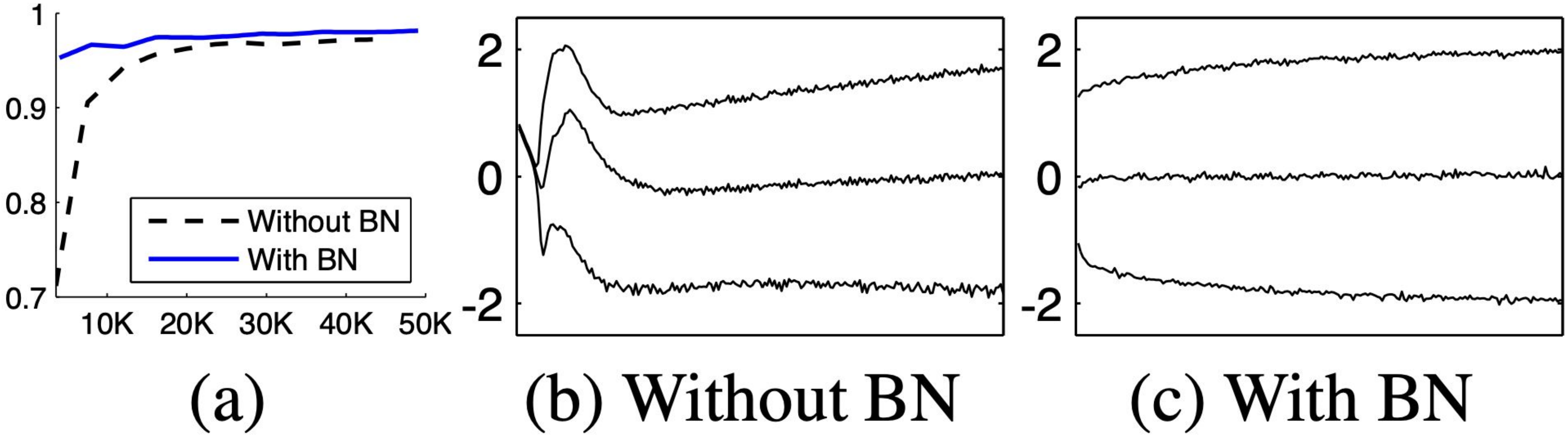
Train mode

По-честному считаем всё

Eval mode

Используем с обучения бегущие средние с momentum

Эффект от использования



https://arxiv.org/pdf/1502.03167

Функции потерь

MSE

$$l_n = (x_n - y_n)^2$$

CrossEntropy

$$l_n = -w_{y_n} \log \frac{\exp(x_{n,y_n})}{\sum_{c=1}^{C} \exp(x_{n,c})} \cdot 1\{y_n \neq \text{ignore\_index}\}$$

На самом деле, можно почти что угодно

$$\mathbf{e}_1 = \left(\partial_t \hat{\mathbf{v}} + (\hat{\mathbf{v}} \cdot \nabla)\hat{\mathbf{v}} + \nabla \hat{p} + e^{\lambda_1}\hat{\mathbf{v}} - e^{\lambda_2}\nabla^2 \hat{\mathbf{v}} - \hat{\mathbf{f}}\right) L/U^2,$$

$$e_2 = (\partial_x \hat{v}_x + \partial_y \hat{v}_y)\, L/U,$$

$$e_3 = \left(\partial_x \hat{f}_x + \partial_y \hat{f}_y\right) L^2/U^2.$$

$$\mathcal{L}_{\mathrm{data}} = \frac{1}{N_{\mathrm{data}} U^2} \sum_{n=1}^{N_{\mathrm{data}}} |\hat{\mathbf{v}}(\mathbf{r}_n^d, t_n^d) - \mathbf{v}^d(\mathbf{r}_n^d, t_n^d)|^2$$

$$\mathcal{L}_{\mathrm{eq}} = \frac{1}{N_{\mathrm{eq}}} \sum_{n=1}^{N_{\mathrm{eq}}} \left[\mathbf{e}_1^2(\mathbf{r}_n^e, t_n^e) + e_2^2(\mathbf{r}_n^e, t_n^e) + e_3^2(\mathbf{r}_n^e)\right]$$

http://jetpletters.ru/ps/2485/article_36470.shtml

Инициализация весов

Xavier Glorot

$$W \sim U\left[-\frac{\sqrt{6}}{\sqrt{n_j + n_{j+1}}}, \frac{\sqrt{6}}{\sqrt{n_j + n_{j+1}}}\right]$$

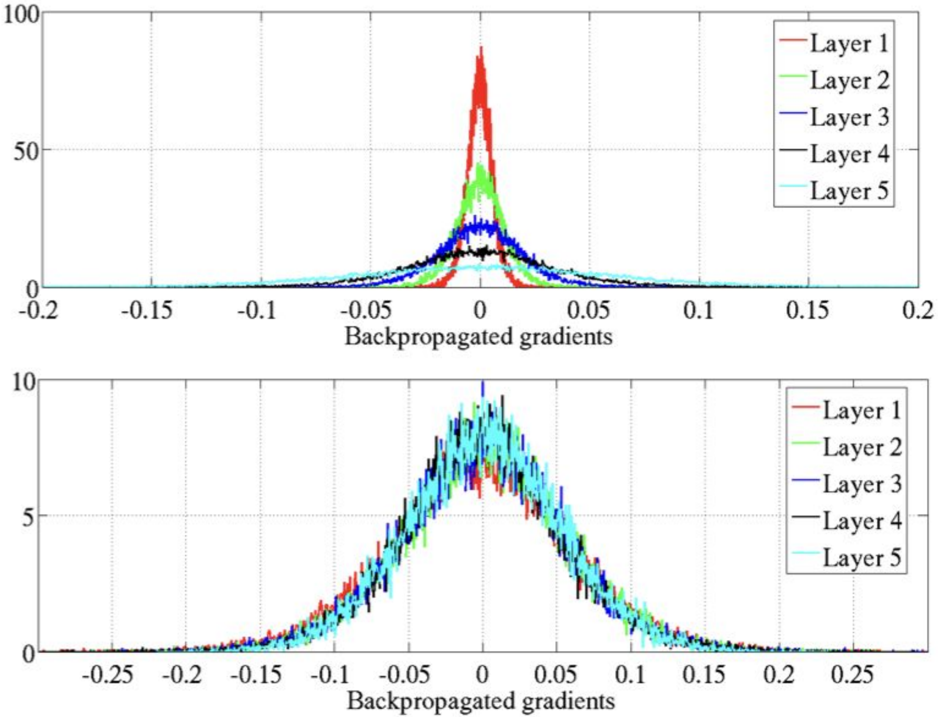
https://proceedings.mlr.press/v9/glorot10a/glorot10a.pdf

Kaiming He

$$W \sim \mathcal{N}\left(0, \frac{2}{n^l}\right)$$

https://paperswithcode.com/paper/delving-deep-into-rectifiers-surpassing-human

Почему она важна?



https://proceedings.mlr.press/v9/glorot10a/glorot10a.pdf