

# Глубинное обучение

## Лекция 2: Автоматическое дифференцирование

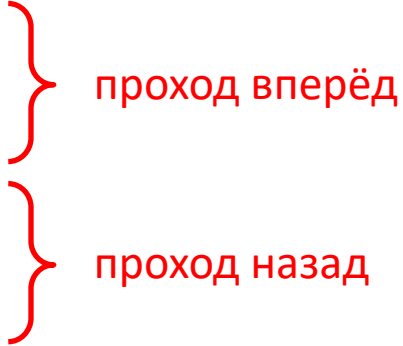
Лектор: Антон Осокин

ФКН ВШЭ, 2020



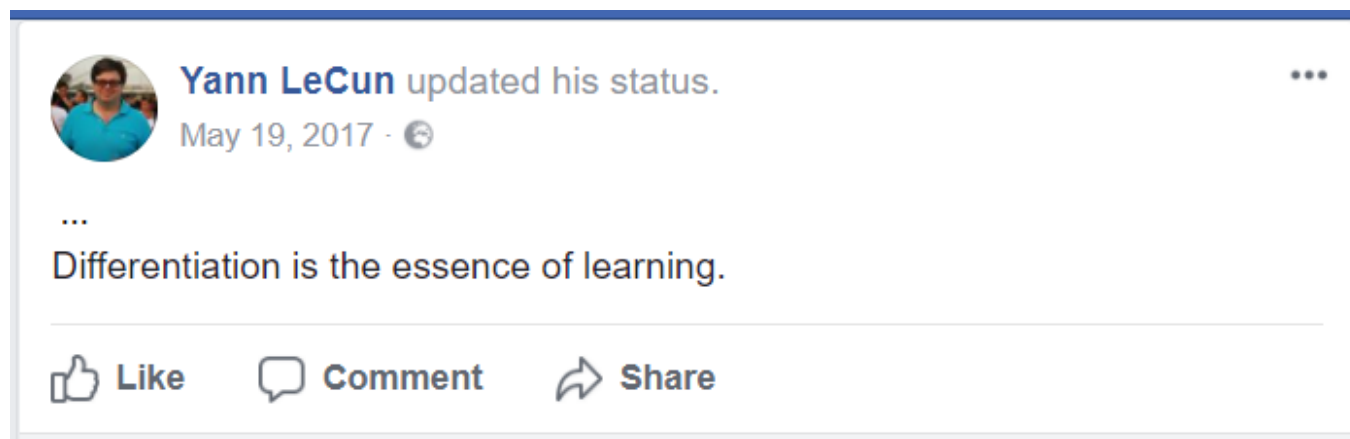
НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ  
УНИВЕРСИТЕТ

# Основной шаг обучения нейросетей

- Даны объект  $x$ , ответ  $y$ , значения параметров  $\theta$
  - Итерация обучения:
    - Выходы нейросети  $f(x, \theta)$
    - Функция потерь  $\ell(f(x, \theta), y)$
    - Градиент потерь по выходам нейросети
    - Градиент потерь по параметрам
    - Шаг стохастической оптимизации
- 
- проход вперёд
- проход назад

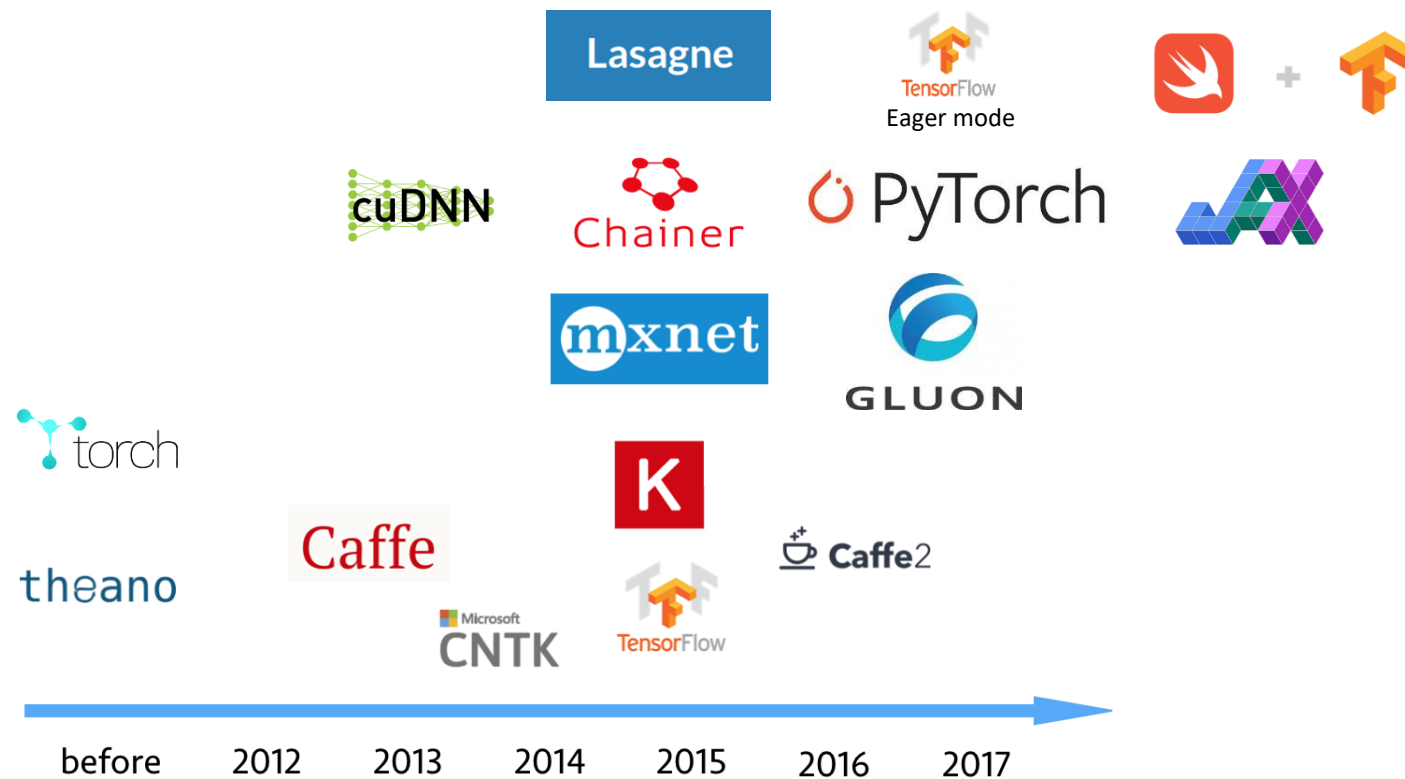
# Основной шаг обучения нейросетей

- Даны объект  $x$ , ответ  $y$ , значения параметров  $\theta$
  - Итерация обучения:
    - Выходы нейросети  $f(x, \theta)$
    - Функция потерь  $\ell(f(x, \theta), y)$
    - Градиент потерь по выходам нейросети
    - Градиент потерь по параметрам
    - Шаг стохастической оптимизации
- } проход вперёд
- } проход назад



# Deep learning frameworks

1. Есть все необходимые функции (CPU, GPU)
2. Организация единой системы
  - Собирает полные производные из производных слоев
  - Автоматическое дифференцирование



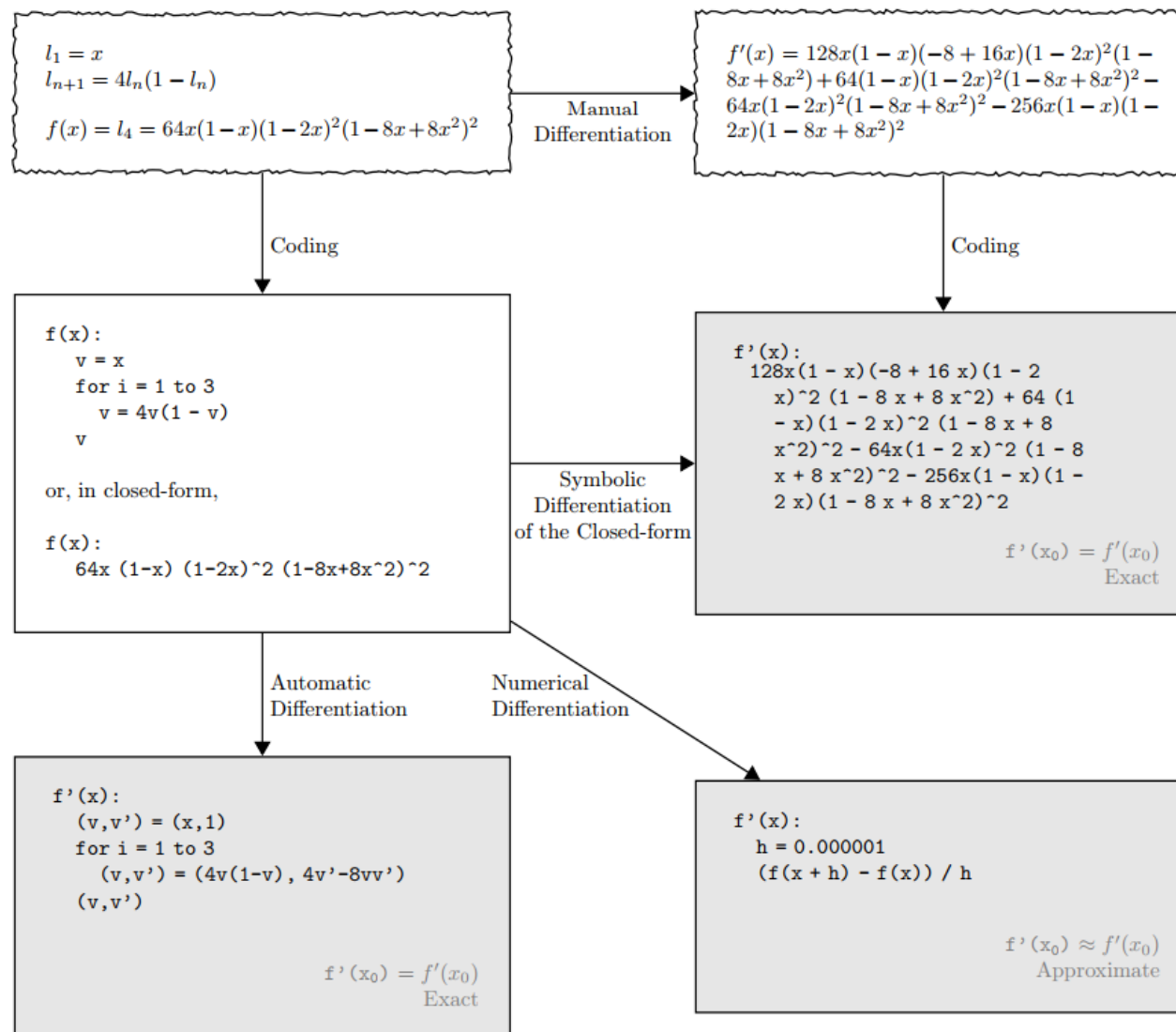
# Производные на компьютере

[Baydin et al., 2017]

- Аналитическая формула
- Численный градиент
  - Конечные разности
- Символьное дифференцирование
  - Производная вычисляется алгебраически
  - Сначала формула, затем вычисления
- Алгоритмическое дифференцирование
  - Производная вычисляется как композиция элементарных производных во время выполнения программы

# Производные на компьютере

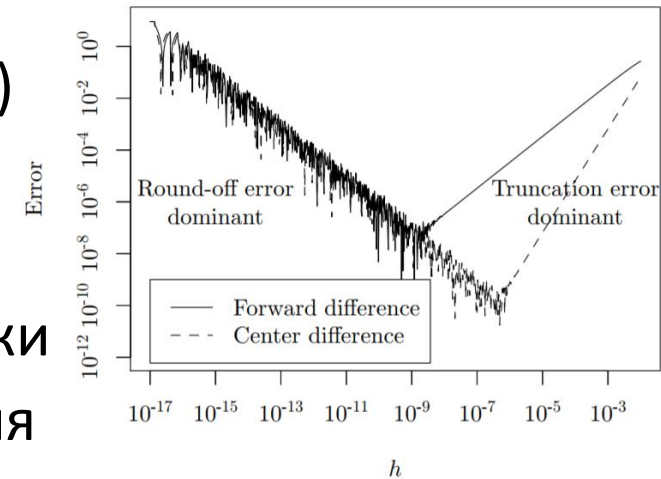
[Baydin et al., 2017]



# Производные на компьютере

[Baydin et al., 2017]

- Аналитическая формула
  - Возможны очень эффективные реализации
  - Требуется выводить формулы, не всегда есть удобная запись
- Численный градиент (конечные разности)
  - Универсальный метод
  - Численные нестабильности ( $A + \varepsilon$ ,  $A - A$ )
  - Медленно
- Символьное дифференцирование
  - Производная вычисляется алгебраически
  - Сначала вся формула, потом вычисления
- Алгоритмическое дифференцирование
  - Выполнение начинается сразу
  - Сохраняются промежуточные результаты



# Алгоритмическое дифференцирование

- Прямой метод (forward mode)
- Обратный метод (reverse mode, backprop)



# Алгоритмическое дифференцирование: прямой метод (forward mode)

- Производные вычисляются вместе с самой функцией
  - Функция  $y = f(x)$ ,  $x \in \mathbb{R}^n$ ,  $y \in \mathbb{R}^m$ , граф из  $v_i$
  - Производные  $v'_i = \frac{dv_i}{dx_1}$

$$v_3 := v_1 v_2 \quad \Rightarrow \quad v'_3 := v'_1 v_2 + v_1 v'_2$$

$$v_3 := \ln(v_2) \quad \Rightarrow \quad v'_3 := \frac{1}{v_2} v'_2$$

- Якобиан вычисляется по столбцам

$$J = \begin{pmatrix} \frac{dy_1}{dx_1} & \cdots & \frac{dy_1}{dx_n} \\ \vdots & \ddots & \vdots \\ \frac{dy_m}{dx_1} & \cdots & \frac{dy_m}{dx_n} \end{pmatrix}$$

- Легко вычислять произведения  $Jr$

# Алгоритмическое дифференцирование: обратный метод (reverse mode, backprop)

- Сначала функция, потом производные

- Функция  $y = f(x)$ ,  $x \in \mathbb{R}^n$ ,  $y \in \mathbb{R}^m$ , граф из  $v_i$

- Производные  $\bar{v}_i = \frac{dy_1}{dv_i}$

$$v_3 := v_1 v_2 \quad \Rightarrow \quad \bar{v}_1 := \frac{dy_1}{dv_1} = \frac{dy_1}{dv_3} \frac{dv_3}{dv_1} = \bar{v}_3 v_2 \quad \Rightarrow \quad \bar{v}_1 := \bar{v}_3 v_2, \quad \bar{v}_2 := v_1 \bar{v}_3$$

$$v_3 := \ln(v_2) \quad \Rightarrow \quad \bar{v}_2 := \bar{v}_3 \frac{1}{v_2}$$

- Якобиан вычисляется по строкам

$$J = \begin{pmatrix} \frac{dy_1}{dx_1} & \cdots & \frac{dy_1}{dx_n} \\ \vdots & \ddots & \vdots \\ \frac{dy_m}{dx_1} & \cdots & \frac{dy_m}{dx_n} \end{pmatrix}$$

Если  $m = 1$ , то Якобиан = градиент<sup>T</sup>

- Легко вычислять произведения  $rJ$

# Прямой или обратный метод?

- Производная из элементарных производных
- Функция  $y = f_4(f_3(f_2(f_1(x))))$ ,  $x \in \mathbb{R}^n$ ,  $y \in \mathbb{R}^m$
- Полная производная

$$\frac{dy}{dx} = \frac{df_1(x)}{dx} \frac{\partial f_2(f_1)}{\partial f_1} \frac{\partial f_3(f_2)}{\partial f_2} \frac{\partial f_4(f_3)}{\partial f_3}$$

- Прямой метод

$$\frac{dy}{dx} = \frac{df_1(x)}{dx} \frac{\partial f_2(f_1)}{\partial f_1} \frac{\partial f_3(f_2)}{\partial f_2} \frac{\partial f_4(f_3)}{\partial f_3}$$



- Обратный метод

$$\frac{dy}{dx} = \frac{df_1(x)}{dx} \frac{\partial f_2(f_1)}{\partial f_1} \frac{\partial f_3(f_2)}{\partial f_2} \frac{\partial f_4(f_3)}{\partial f_3}$$



# Прямой или обратный метод?

- Производная из элементарных производных
- Функция  $y = f_4(f_3(f_2(f_1(x))))$ ,  $x \in \mathbb{R}^n$ ,  $y \in \mathbb{R}^m$
- Прямой метод эффективен, когда  $n \ll m$
- Обратный метод эффективен когда  $n \gg m$
- В МО,  $x$  – признаки и параметры,  $y$  – функция потерь
  - Обычно используется обратный метод
  - В сложных случаях можно использовать комбинации
- Оптимальный порядок умножения матриц можно найти с помощью динамического программирования

# Как вычислить Гессиан на вектор?

- Функция  $y = f(x_1, x_2, \dots, x_n), x \in \mathbb{R}^n, y \in \mathbb{R}$
- Гессиан – матрица вторых производных

$$H = \begin{pmatrix} \frac{df}{dx_1 dx_1} & \cdots & \frac{df}{dx_1 dx_n} \\ & \ddots & \vdots \\ \frac{df}{dx_n dx_1} & \cdots & \frac{df}{dx_n dx_n} \end{pmatrix}$$

- Гессиан – очень большая матрица
- Стохастические методы второго порядка требуют вычисления  $Hv$  (пример – KFAC [Martens&Grosse, 2015])

Метод 1:

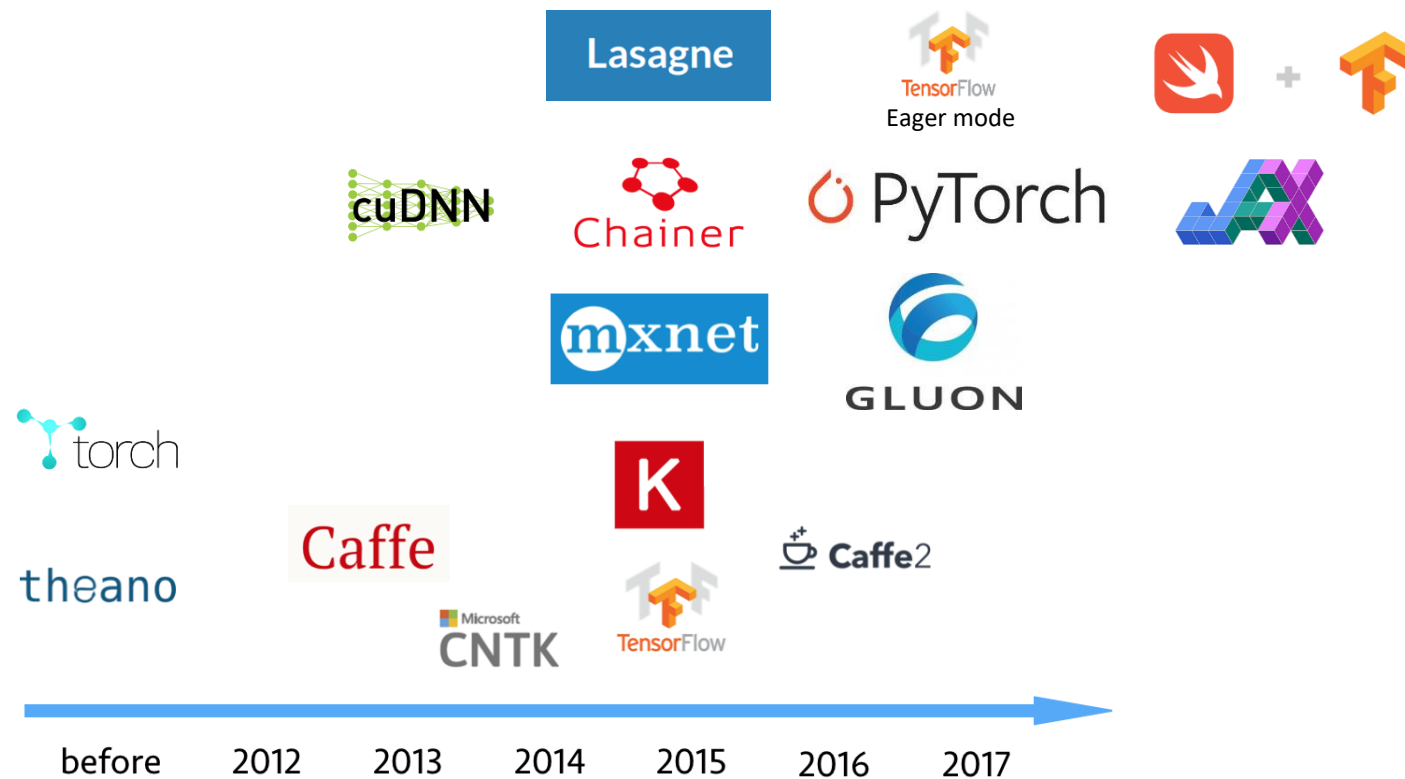
1.  $g(x) = v^T \nabla f(x)$
2.  $Hv = \nabla[g(x)]$

Метод 2:



1.  $g(x) = \nabla f(x)$
2.  $Hv = J_g v$

# Deep learning frameworks

1. Есть все необходимые функции (CPU, GPU)
2. Организация единой системы
  - Собирает полные производные из производных слоев
  - Автоматическое дифференцирование



# Библиотеки для глубинного обучения

1. Низкоуровневые операции: BLAS, LAPACK,  NumPy  cuDNN



2. Линейная алгебра с дифференцированием

– Отдельное построение вычислительного графа

theano

Microsoft  
CNTK

 TensorFlow™



– Построение графа совместно с выполнением



Caffe

  
Chainer

 PyTorch

 TensorFlow 2.0

  
GLUON

3. Высокоуровневые библиотеки

Lasagne

 K Keras



# Рекомендации (сентябрь 2020)

- Production



- Data science



- ML research



Что дальше?



JAX: Autograd и XLA для numpy  
<https://github.com/google/jax>



S4TF: Autograd в языке! Python - медленный  
<https://github.com/tensorflow/swift>



# Зачем знать про backprop?



Andrej Karpathy

[Follow](#)

Director of AI at Tesla. Previously Research Scientist at OpenAI and PhD student at Stanford. I like to train deep neural nets on large datasets.

Dec 19, 2016 · 7 min read

## Yes you should understand backprop

“Backprop – leaky abstraction!”

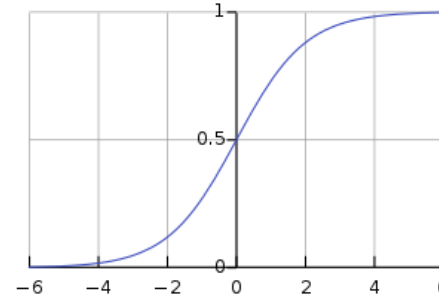
- Почему сеть не обучается?
- Почему сеть обучается медленно?

# Проблемы в функциях активаций

[Karpathy, 2016]

- Сигмоида (или tanh)

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

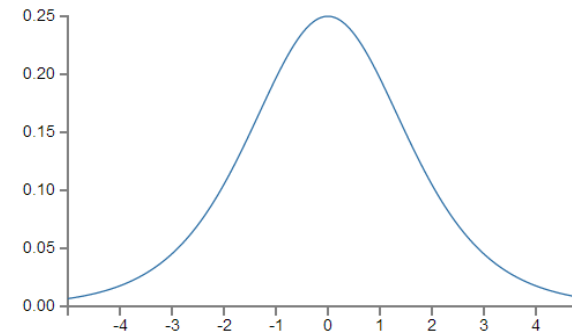


- Градиент  $\frac{\partial \sigma(x)}{\partial x} = \sigma(x)(1 - \sigma(x))$

- Градиент  $\approx 0$  при  $|x| \geq 6$
- Максимум градиента = 0.25

- При каждом умножении

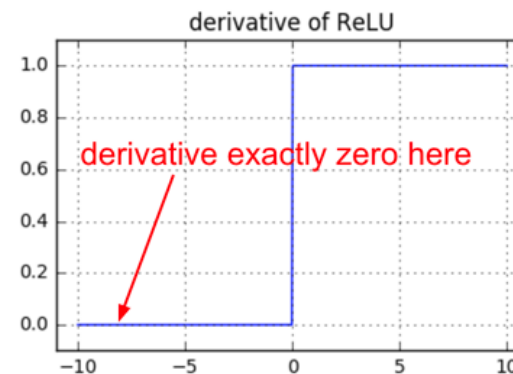
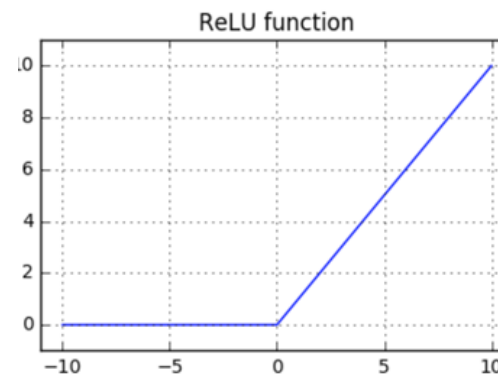
макс. градиента уменьшается



- $\text{ReLU}(x) = \begin{cases} 0, & x \leq 0, \\ x, & x > 0 \end{cases}$

$$\frac{\partial \text{ReLU}(x)}{\partial x} = \begin{cases} 0, & x \leq 0, \\ 1, & x > 0 \end{cases}$$

- Мёртвые узлы!



# Инициализация сетей

- ‘glorot’ (‘xavier’) и ‘kaiming’
- Инициализация линейного слоя:

[Glorot&Bengio, 2010]

[He et al., 2015]

$$y = w^T h(x)$$

- Идея: выбрать дисперсию весов так, чтобы активации были распределены стандартно-нормально
- Пусть  $w$  – i.i.d.,  $x$  – i.i.d.,  $x$  и  $w$  – независимы,

$$\mathbb{E}w_i = 0$$

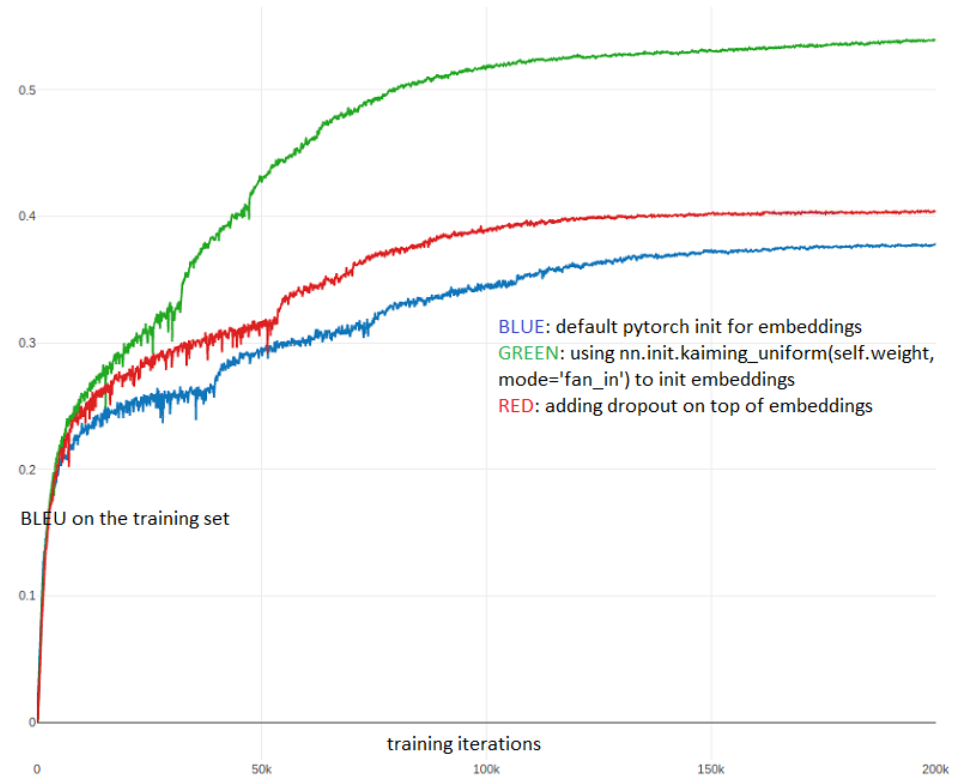
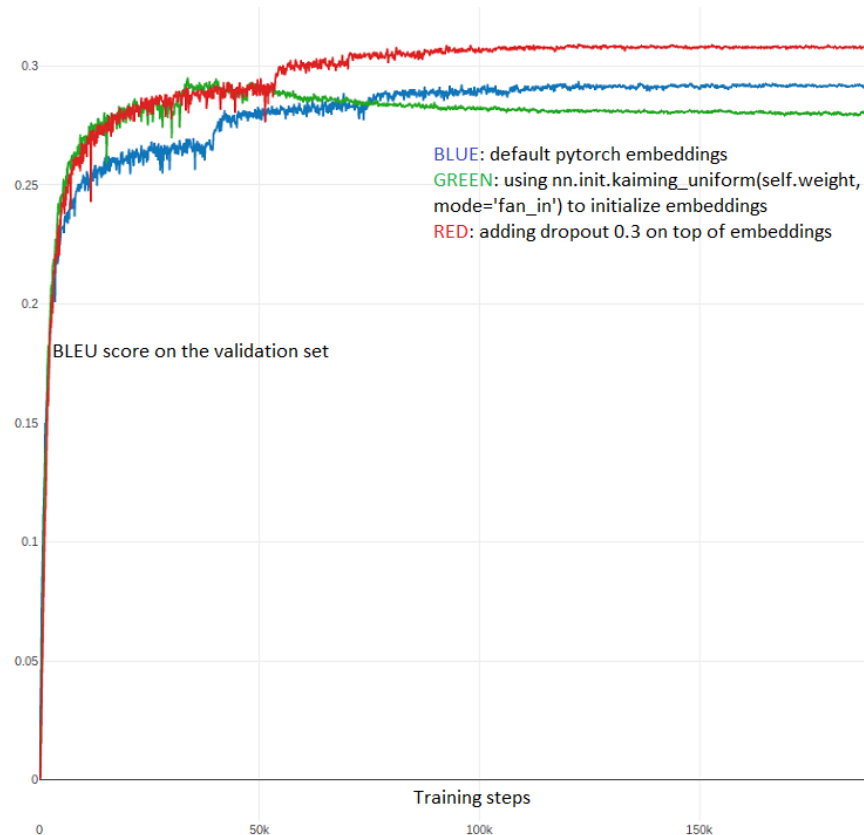
$$\text{Var}[y] = d\text{Var}[w_i h(x_i)] = d\text{Var}[w_i] \mathbb{E}[h(x_i)^2]$$

- Выберем 
$$\text{Var}[w_i] := \frac{1}{d} \frac{\text{Var}[x_i]}{\mathbb{E}[h(x_i)^2]}$$

- Для ReLu  $\text{Var}[w_i] := \frac{2}{d}$  , или  $w_i \sim \mathcal{N}(0, \sigma^2 := \frac{2}{d})$

# Инициализация сетей

- Пример из личной практики  
seq2seq с вниманием для машинного перевода  
БАГ: медленное обучение и потерянные 3 BLEU

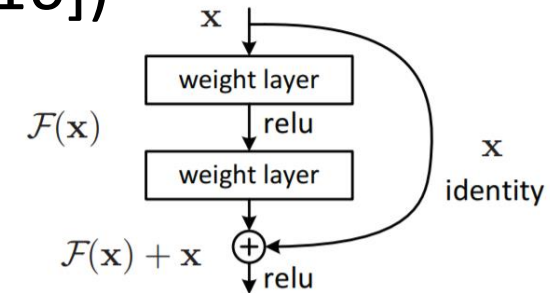


# Проблемы с градиентом – повод разрабатывать новые архитектуры!

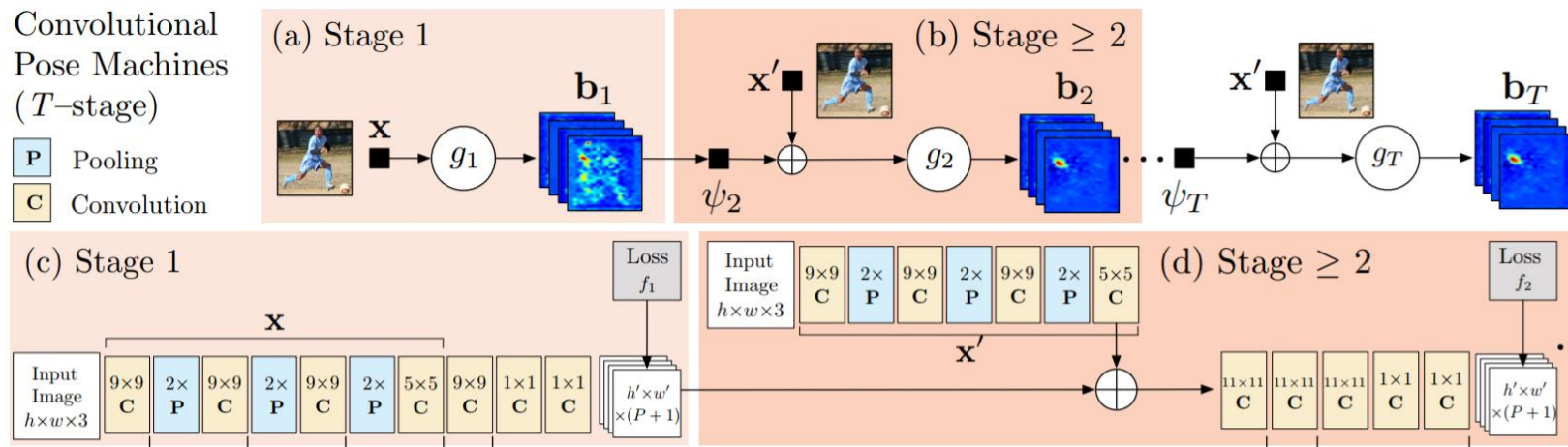
- Skip connections (ResNet, [He et al., 2016])

– Regular:  $y := f(x) \Rightarrow \frac{d\ell}{dx} := f'(x) \frac{d\ell}{dy}$

– Skip:  $y := f(x) + x \Rightarrow \frac{d\ell}{dx} := f'(x) \frac{d\ell}{dy} + \frac{d\ell}{dy}$



- Функции потерь на разной глубине (e.g. [Wei et al., 2016])

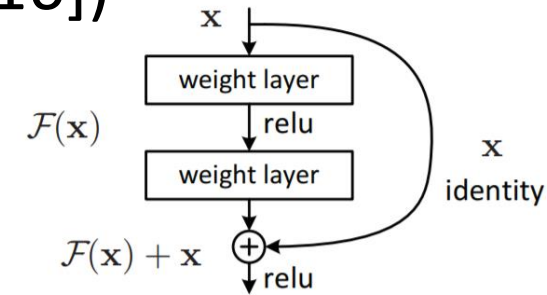


# Проблемы с градиентом – повод разрабатывать новые архитектуры!

- Skip connections (ResNet, [He et al., 2016])

- Regular:  $y := f(x) \Rightarrow \frac{d\ell}{dx} := f'(x) \frac{d\ell}{dy}$

- Skip:  $y := f(x) + x \Rightarrow \frac{d\ell}{dx} := f'(x) \frac{d\ell}{dy} + \frac{d\ell}{dy}$



- Функции потерь на разной глубине (e.g. [Wei et al., 2016])
- Специальные слои (LSTM/GRU) [Hochreiter& Schmidhuber, 1997]
  - RNN – итеративное применение слоя

$$h_t := W_h \sigma(h_{t-1}) + W_x x_t + b$$

- Матрица  $W_h$  возводится в степень  $t$
  - Градиент либо затухает (LSTM), либо взрывается (clipping)

# Заключение

- Backprop –автоматическое дифференцирование
  - По умолчанию используется обратный метод
  - В ряде случаев нужны и другие методы
- При обучении моделей надо думать о градиенте
  - Затухание и взрыв градиентов
  - Инициализация
  - Специальные архитектуры

# Заключение

- Backprop –автоматическое дифференцирование
  - По умолчанию используется обратный метод
  - В ряде случаев нужны и другие методы
- При обучении моделей надо думать о градиенте
  - Затухание и взрыв градиентов
  - Инициализация
  - Специальные архитектуры

Спасибо за внимание!