академия больших данных

mail.ru group

Регуляризация в нейронных сетях. Аугментации.

Фёдор Киташов

Программист-исследователь в команде компьютерного зрения

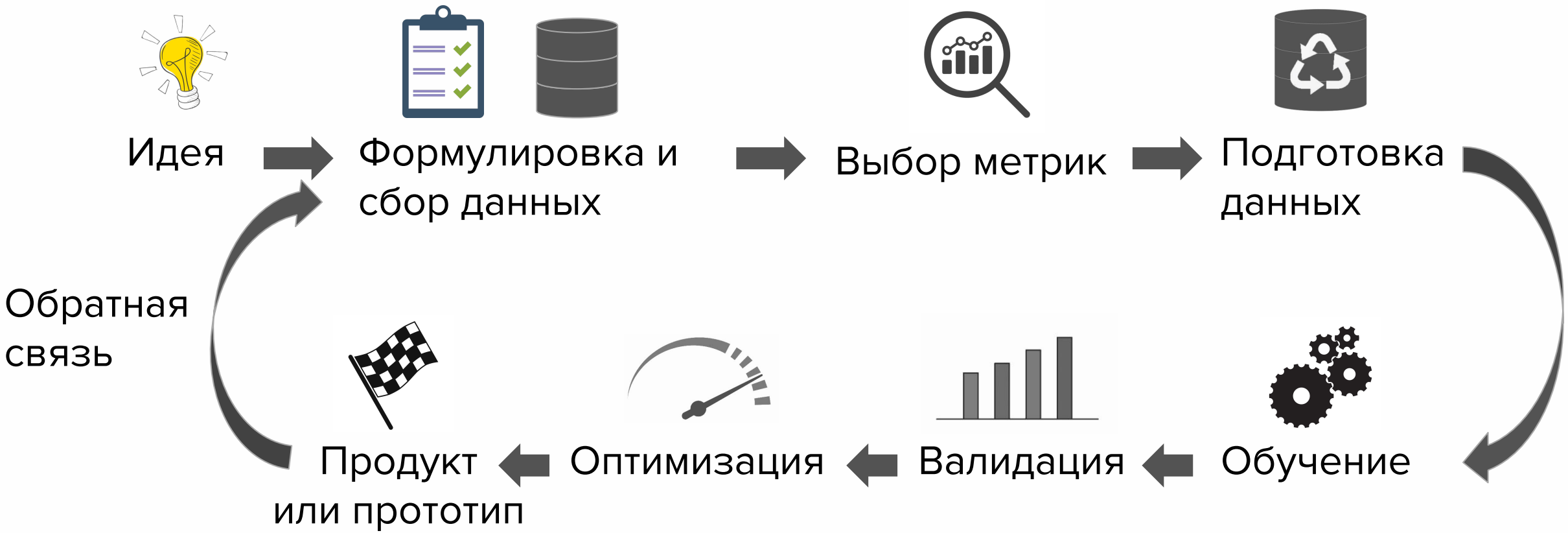
План лекции

- Recap
  - Пайплайн обучения
  - Архитектуры
- Регуляризация
  - L1, L2-регуляризации
  - Dropout
  - Batchnorm
  - Аугментации
- Практические аспекты обучения
  - Построение пайплайна обучения
  - Подбор гиперпараметров оптимизатора

Пайплайн машинного обучения

ML сложнее, чем кажется

На практике:



Регуляризация

Регуляризации

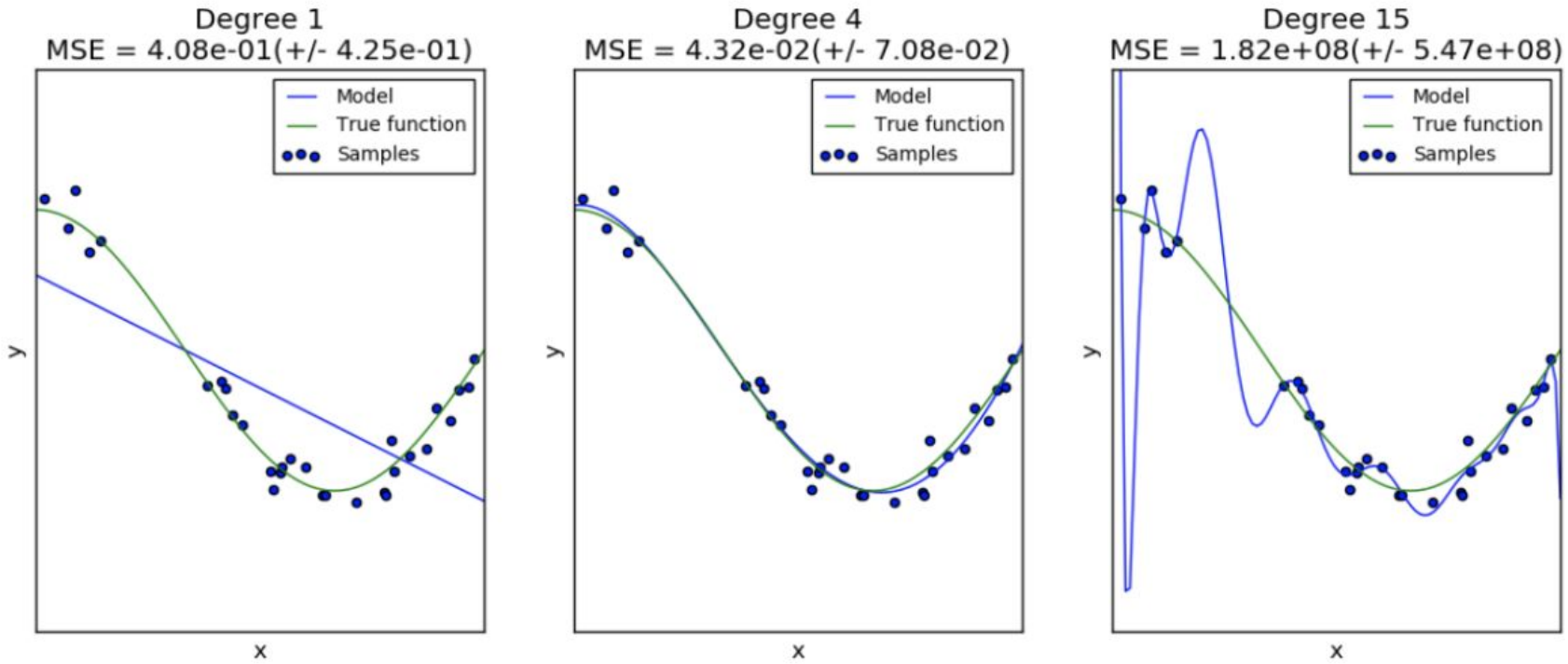
- Ограничение нормы весов
  - L1 и L2-регуляризаци
- Dropout
- Batch Normalization
- Аугментации

Повторение: регуляризация

Регуляризация - борьба с переобучением

Повторение: регуляризация

Регуляризация - борьба с переобучением



https://github.com/esokolov/ml-course-hse/blob/master/2019-fall/lecture-notes/lecture03-linregr.pdf

Повторение: регуляризация

Можно ограничить норму весов

$$L_\alpha(w) = L(w) + \alpha R(w)$$

$$R(w) = \|w\|_2 = \sum_{i=1}^{d} w_i^2,$$

$$R(w) = \|w\|_1 = \sum_{i=1}^{d} |w_i|.$$

https://github.com/esokolov/ml-course-hse/blob/master/2019-fall/lecture-notes/lecture03-linregr.pdf

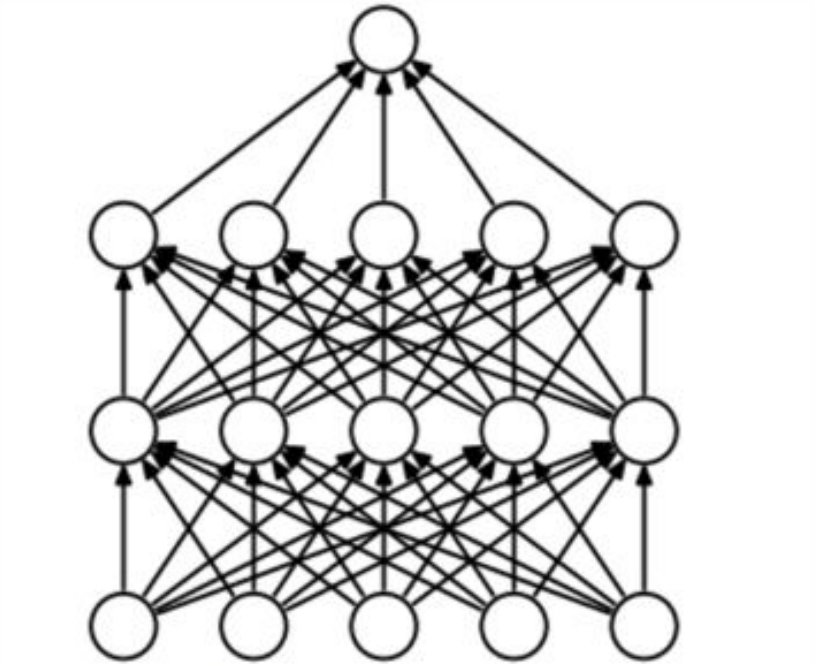
Dropout

Dropout - случайное обнуление весов слоя во время обучения

Dropout

Dropout - случайное обнуление

весов слоя во время обучения

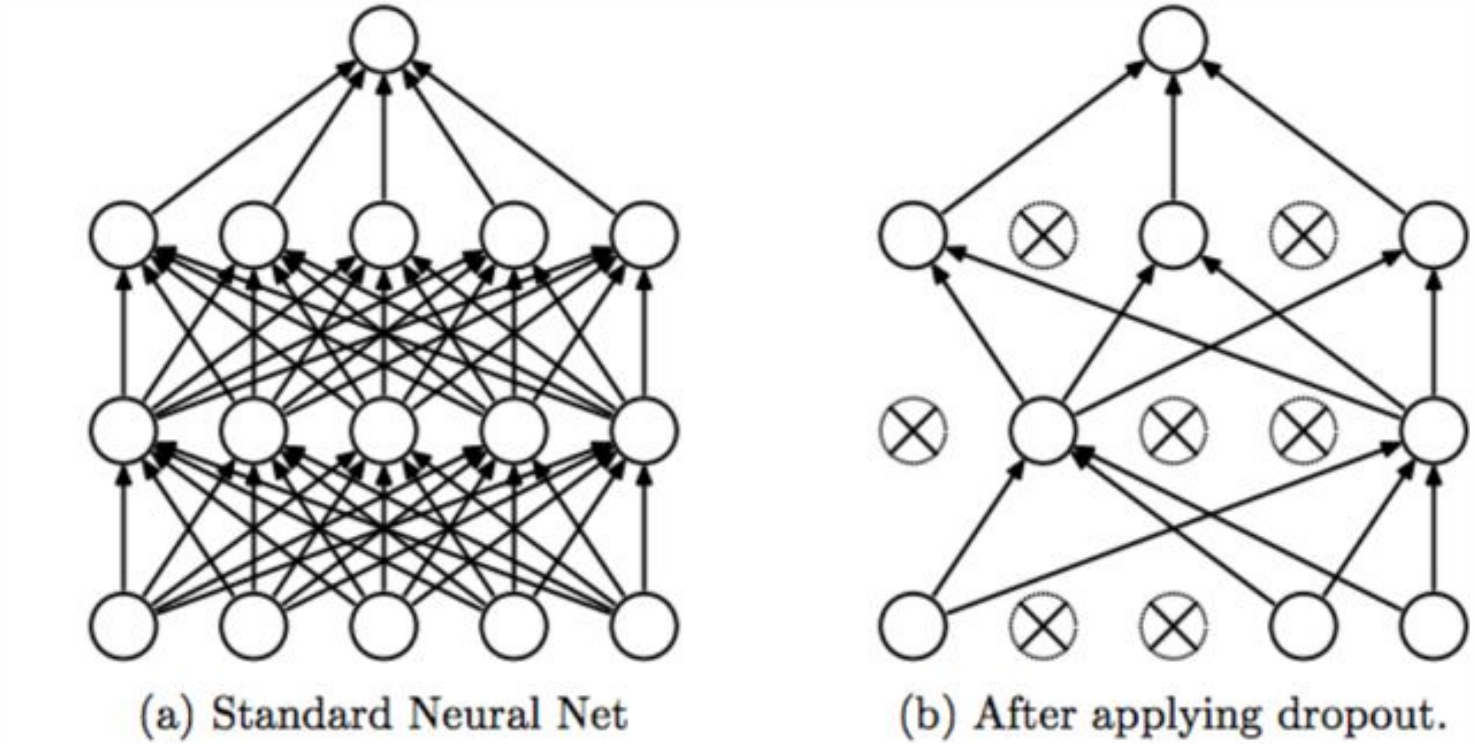


(a) Standard Neural Net

Dropout

Dropout - случайное обнуление

весов слоя во время обучения

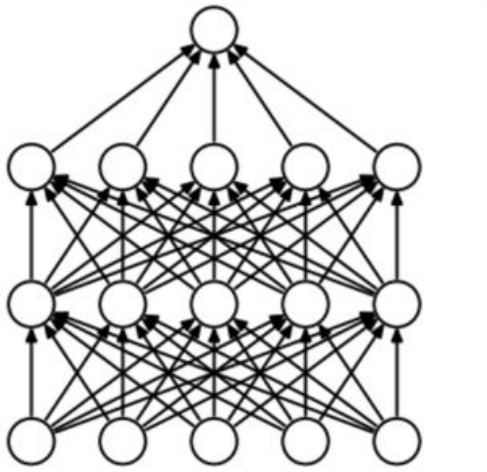


(a) Standard Neural Net

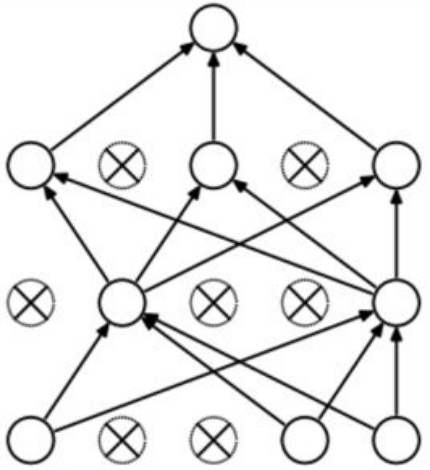
(b) After applying dropout.

Dropout: имплементация

- Создаем бинарную маску, где с вероятностью **p** встречается ноль



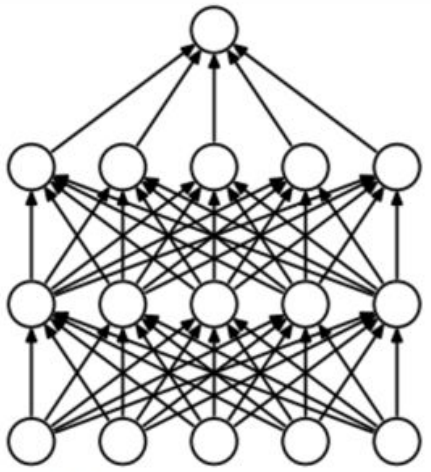
(a) Standard Neural Net



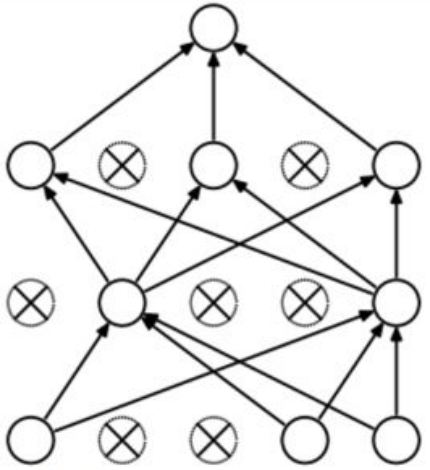
(b) After applying dropout.

Dropout: имплементация

- Создаем бинарную маску, где с вероятностью **p** встречается ноль
- Умножаем поэлементно входной тензор на маску



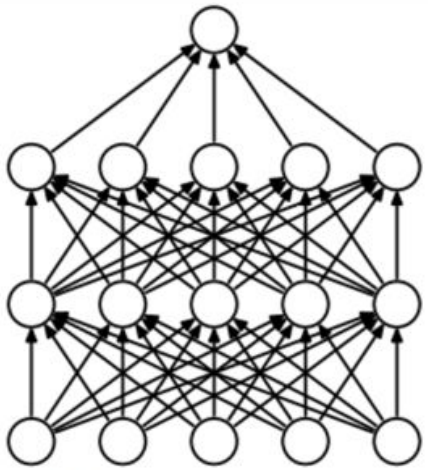
(a) Standard Neural Net



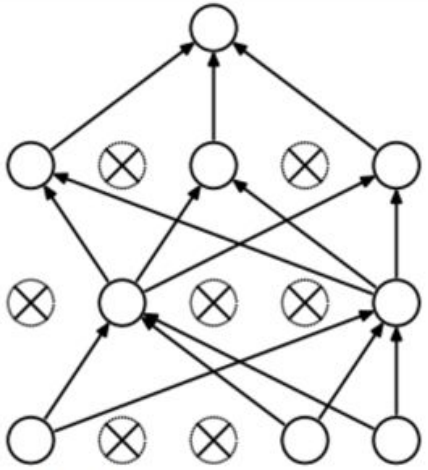
(b) After applying dropout.

Dropout: имплементация

- Создаем бинарную маску, где с вероятностью **p** встречается ноль
- Умножаем поэлементно входной тензор на маску
- Делим результат на **1 - p**

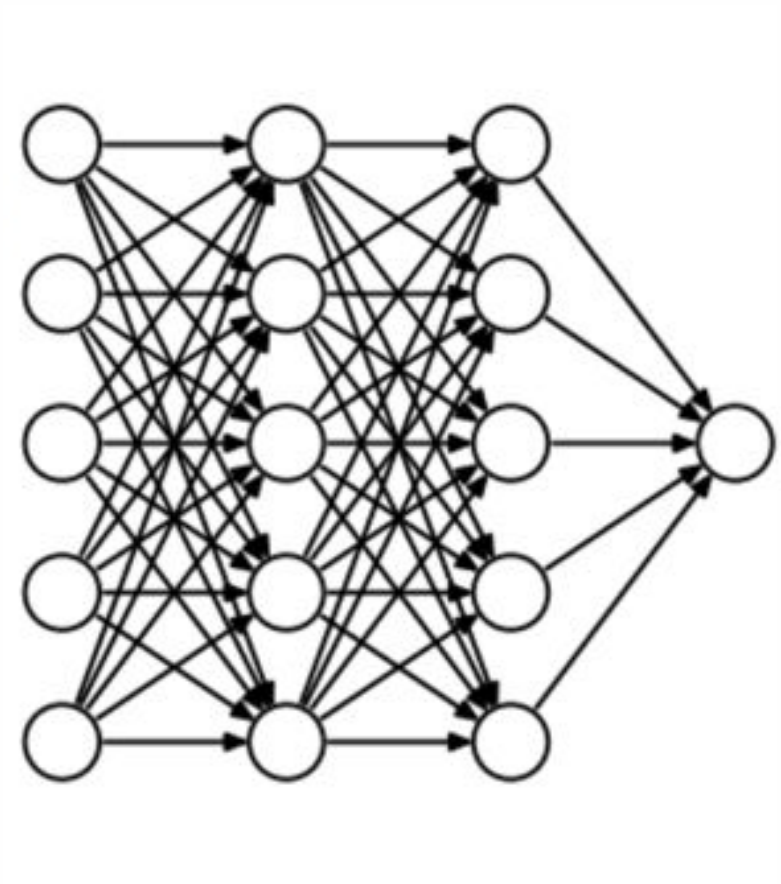


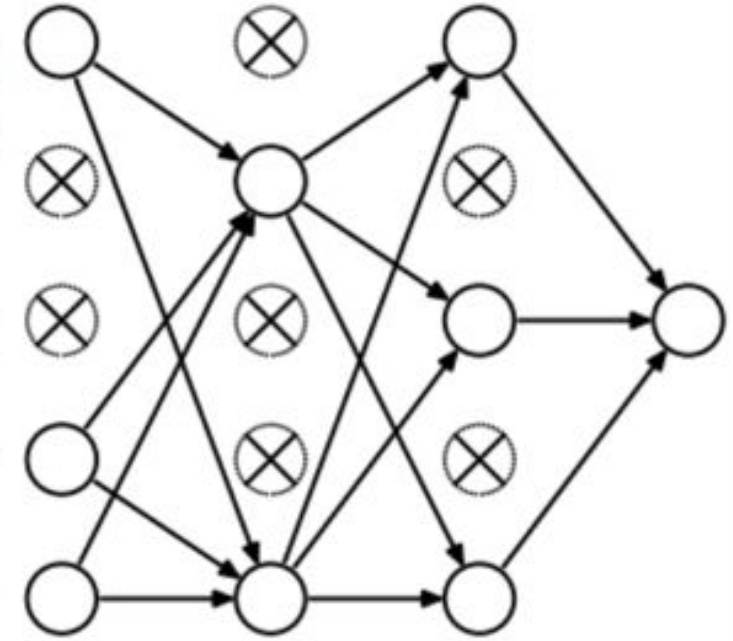
(a) Standard Neural Net



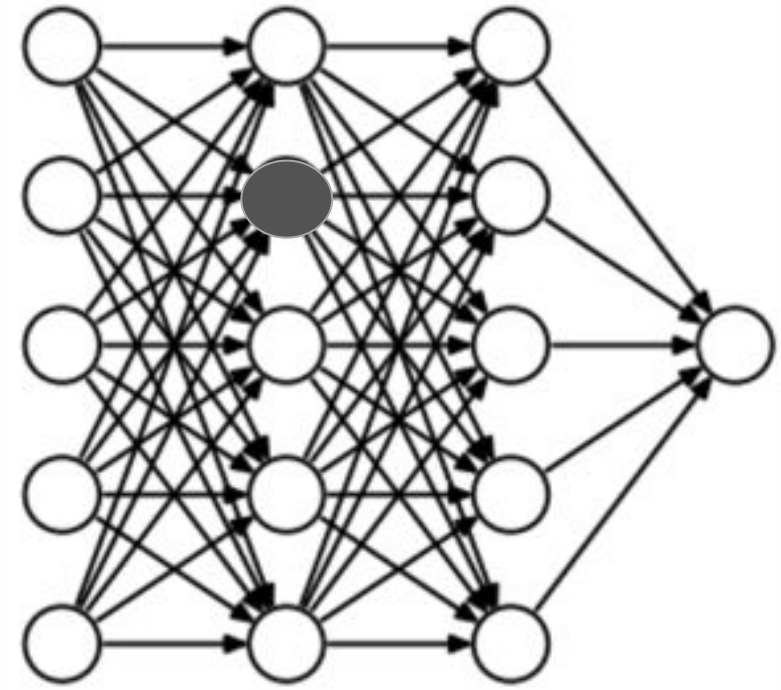
(b) After applying dropout.

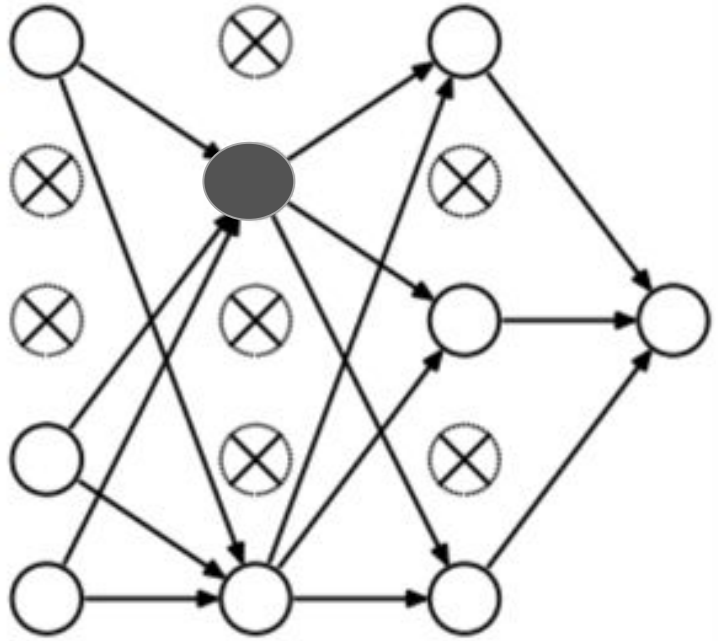
Зачем делить на **1 - p**





Зачем делить на **1 - p**





Чтобы сохранить среднее значение распределения из выхода нейрона

Зачем делить на **1 - p**

Чтобы сохранить среднее значение распределения из выхода нейрона.

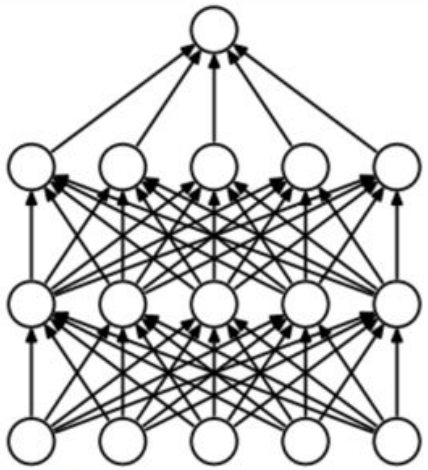
p процентов весов занулены. **1 - p** процентов весов не занулены.

То есть мат.ожидание выхода слоя стало в (1 - p) раз меньше.

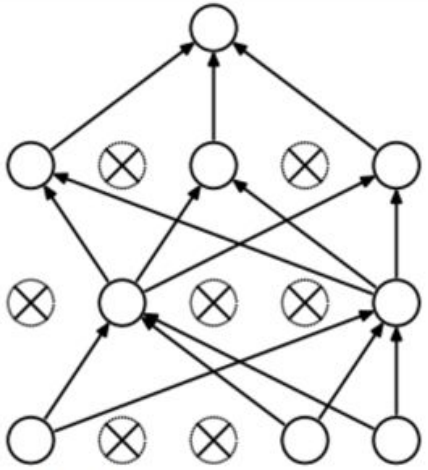
Чтобы мат.ожидание выхода слоя было одинаково вне зависимости от p, делим выход на **(1 - p)**

Dropout: имплементация

- Создаем бинарную маску, где с вероятностью **p** встречается ноль
- Умножаем поэлементно входной тензор на маску
- Делим результат на **1 - p**



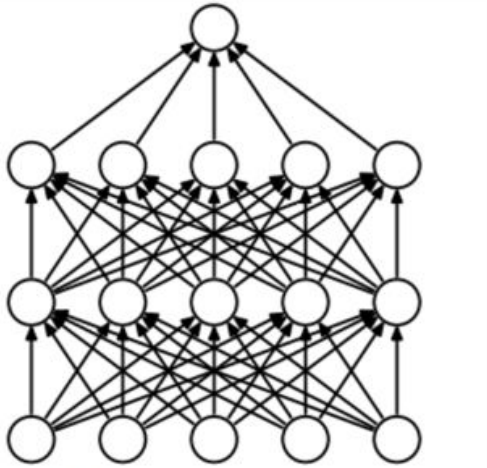
(a) Standard Neural Net



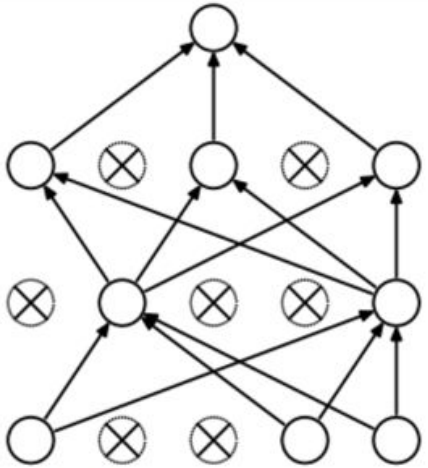
(b) After applying dropout.

Dropout: тест

- Что делает дропаут во время теста?



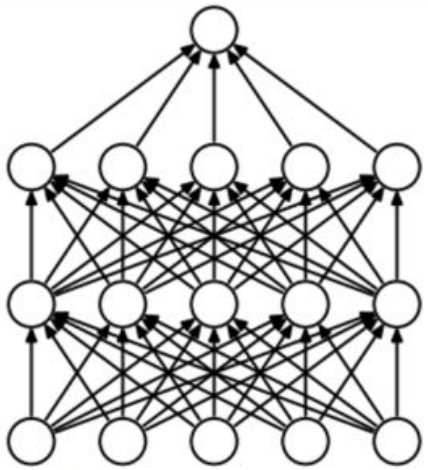
(a) Standard Neural Net



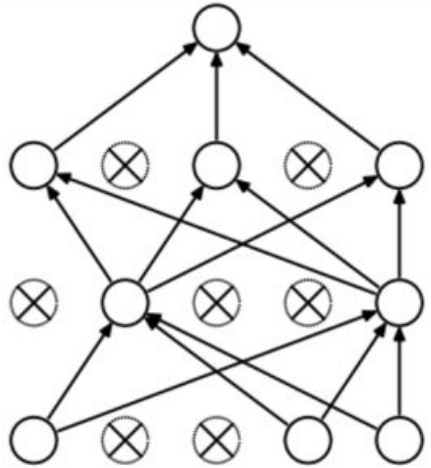
(b) After applying dropout.

Dropout: тест

- Что делает дропаут во время теста?



(a) Standard Neural Net



(b) After applying dropout.

Dropout: реализация

```
def dropout(input_tensor, p):
    """
    input_tensor: входной тензор
    p: вероятность обнуления
    """
    mask = np.random.binomial(1, 1 - p, input_tensor.shape)
```

Dropout: реализация

```
def dropout(input_tensor, p):
    """
    input_tensor: входной тензор
    p: вероятность обнуления
    """
    mask = np.random.binomial(1, 1 - p, input_tensor.shape)
    output = input_tensor * mask
```

```
Docstring:
binomial(n, p, size=None)

Draw samples from a binomial distribution.

Samples are drawn from a binomial distribution with specified
parameters, n trials and p probability of success where
n an integer >= 0 and p is in the interval [0,1]. (n may be
input as a float, but it is truncated to an integer in use)

Parameters
```

Dropout: реализация

```
def dropout(input_tensor, p):
    """
    input_tensor: входной тензор
    p: вероятность обнуления
    """
    mask = np.random.rand(*input_tensor.shape) >= p
```

Dropout: реализация

```
def dropout(input_tensor, p, train=True):
    """
    input_tensor: входной тензор
    p: вероятность обнуления
    """
    output = input_tensor
    if train:
        mask = np.random.binomial(1, 1 - p, input_tensor.shape)
```

Dropout: реализация

```
def dropout(input_tensor, p, train=True):
    """
    input_tensor: входной тензор
    p: вероятность обнуления
    """
    output = input_tensor
    if train:
        mask = np.random.binomial(1, 1 - p, input_tensor.shape)
        mask = np.divide(mask, 1 - p)
```

Dropout: реализация

```
def dropout(input_tensor, p, train=True):
    """
    input_tensor: входной тензор
    p: вероятность обнуления
    """
    output = input_tensor
    if train:
        mask = np.random.binomial(1, 1 - p, input_tensor.shape)
        mask = np.divide(mask, 1 - p)
        output = input_tensor * mask
    return output
```

Dropout: backprop

```
def backward(input_tensor, mask, gradient_output, train=True):
```

Повторение: Якобиан

$$\boldsymbol{f} : \mathbb{R}^n \to \mathbb{R}^m$$

$$\boldsymbol{f}(\boldsymbol{x}) = [f_1(x_1, \ldots, x_n), f_2(x_1, \ldots, x_n), \ldots, f_m(x_1, \ldots, x_n)]$$

$$\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{x}} = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial x_1} & \cdots & \frac{\partial f_m}{\partial x_n} \end{bmatrix}$$

http://web.stanford.edu/class/cs224n/readings/gradient-notes.pdf

Повторение: Якобиан

$$\boldsymbol{f}(x) = [f_1(x), f_2(x)]$$

$$\boldsymbol{g}(\boldsymbol{y}) = [g_1(y_1, y_2), g_2(y_1, y_2)]$$

$$\boldsymbol{g}(x) = [g_1(f_1(x), f_2(x)), g_2(f_1(x), f_2(x))]$$

http://web.stanford.edu/class/cs224n/readings/gradient-notes.pdf

Повторение: Якобиан

$$\boldsymbol{f}(x) = [f_1(x), f_2(x)]$$

$$\boldsymbol{g}(\boldsymbol{y}) = [g_1(y_1, y_2), g_2(y_1, y_2)]$$

$$\boldsymbol{g}(x) = [g_1(f_1(x), f_2(x)), g_2(f_1(x), f_2(x))]$$

$$\frac{\partial \boldsymbol{g}}{\partial x} =$$

http://web.stanford.edu/class/cs224n/readings/gradient-notes.pdf

Повторение: Якобиан

$$\boldsymbol{f}(x) = [f_1(x), f_2(x)]$$

$$\boldsymbol{g}(\boldsymbol{y}) = [g_1(y_1, y_2), g_2(y_1, y_2)]$$

$$\boldsymbol{g}(x) = [g_1(f_1(x), f_2(x)), g_2(f_1(x), f_2(x))]$$

$$\frac{\partial \boldsymbol{g}}{\partial x} = \begin{bmatrix} \frac{\partial}{\partial x} g_1(f_1(x), f_2(x)) \\ \frac{\partial}{\partial x} g_2(f_1(x), f_2(x)) \end{bmatrix}$$

http://web.stanford.edu/class/cs224n/readings/gradient-notes.pdf

Повторение: Якобиан

$$\boldsymbol{f}(x) = [f_1(x), f_2(x)]$$

$$\boldsymbol{g}(\boldsymbol{y}) = [g_1(y_1, y_2), g_2(y_1, y_2)]$$

$$\boldsymbol{g}(x) = [g_1(f_1(x), f_2(x)), g_2(f_1(x), f_2(x))]$$

$$\frac{\partial \boldsymbol{g}}{\partial x} = \begin{bmatrix} \frac{\partial}{\partial x} g_1(f_1(x), f_2(x)) \\ \frac{\partial}{\partial x} g_2(f_1(x), f_2(x)) \end{bmatrix} = \begin{bmatrix} \frac{\partial g_1}{\partial f_1}\frac{\partial f_1}{\partial x} + \frac{\partial g_1}{\partial f_2}\frac{\partial f_2}{\partial x} \\ \frac{\partial g_2}{\partial f_1}\frac{\partial f_1}{\partial x} + \frac{\partial g_2}{\partial f_2}\frac{\partial f_2}{\partial x} \end{bmatrix}$$

http://web.stanford.edu/class/cs224n/readings/gradient-notes.pdf

Повторение: Якобиан

$$\frac{\partial \boldsymbol{g}}{\partial x} = \begin{bmatrix} \frac{\partial}{\partial x} g_1(f_1(x), f_2(x)) \\ \frac{\partial}{\partial x} g_2(f_1(x), f_2(x)) \end{bmatrix} = \begin{bmatrix} \frac{\partial g_1}{\partial f_1}\frac{\partial f_1}{\partial x} + \frac{\partial g_1}{\partial f_2}\frac{\partial f_2}{\partial x} \\ \frac{\partial g_2}{\partial f_1}\frac{\partial f_1}{\partial x} + \frac{\partial g_2}{\partial f_2}\frac{\partial f_2}{\partial x} \end{bmatrix}$$

http://web.stanford.edu/class/cs224n/readings/gradient-notes.pdf

Повторение: Якобиан

$$\frac{\partial \boldsymbol{g}}{\partial x} = \begin{bmatrix} \frac{\partial}{\partial x} g_1(f_1(x), f_2(x)) \\ \frac{\partial}{\partial x} g_2(f_1(x), f_2(x)) \end{bmatrix} = \begin{bmatrix} \frac{\partial g_1}{\partial f_1}\frac{\partial f_1}{\partial x} + \frac{\partial g_1}{\partial f_2}\frac{\partial f_2}{\partial x} \\ \frac{\partial g_2}{\partial f_1}\frac{\partial f_1}{\partial x} + \frac{\partial g_2}{\partial f_2}\frac{\partial f_2}{\partial x} \end{bmatrix}$$

$$\frac{\partial \boldsymbol{g}}{\partial x} = \frac{\partial \boldsymbol{g}}{\partial \boldsymbol{f}}\frac{\partial \boldsymbol{f}}{\partial x} = \begin{bmatrix} \frac{\partial g_1}{\partial f_1} & \frac{\partial g_1}{\partial f_2} \\ \frac{\partial g_2}{\partial f_1} & \frac{\partial g_2}{\partial f_2} \end{bmatrix} \begin{bmatrix} \frac{\partial f_1}{\partial x} \\ \frac{\partial f_2}{\partial x} \end{bmatrix}$$

http://web.stanford.edu/class/cs224n/readings/gradient-notes.pdf

Повторение: backprop

$$z = f(x) \qquad \frac{\partial z}{\partial x}?$$

http://web.stanford.edu/class/cs224n/readings/gradient-notes.pdf

Повторение: backprop

$$\boldsymbol{z} = f(\boldsymbol{x}) \quad \frac{\partial \boldsymbol{z}}{\partial \boldsymbol{x}}?$$

$$z_i = f(x_i)$$

$$\left(\frac{\partial \boldsymbol{z}}{\partial \boldsymbol{x}}\right)_{ij}$$

http://web.stanford.edu/class/cs224n/readings/gradient-notes.pdf

Повторение: backprop

$$\boldsymbol{z} = f(\boldsymbol{x}) \quad \frac{\partial \boldsymbol{z}}{\partial \boldsymbol{x}}?$$

$$z_i = f(x_i)$$

$$\left(\frac{\partial \boldsymbol{z}}{\partial \boldsymbol{x}}\right)_{ij} = \frac{\partial z_i}{\partial x_j}$$

http://web.stanford.edu/class/cs224n/readings/gradient-notes.pdf

Повторение: backprop

$$\boldsymbol{z} = f(\boldsymbol{x}) \quad \frac{\partial \boldsymbol{z}}{\partial \boldsymbol{x}}? \qquad z_i = f(x_i)$$

$$(\frac{\partial \boldsymbol{z}}{\partial \boldsymbol{x}})_{ij} = \frac{\partial z_i}{\partial x_j} = \frac{\partial}{\partial x_j} f(x_i)$$

http://web.stanford.edu/class/cs224n/readings/gradient-notes.pdf

Повторение: backprop

$$\boldsymbol{z} = f(\boldsymbol{x}) \quad \frac{\partial \boldsymbol{z}}{\partial \boldsymbol{x}}? \qquad z_i = f(x_i)$$

$$(\frac{\partial \boldsymbol{z}}{\partial \boldsymbol{x}})_{ij} = \frac{\partial z_i}{\partial x_j} = \frac{\partial}{\partial x_j} f(x_i) = \begin{cases} f'(x_i) & \text{if } i = j \\ 0 & \text{if otherwise} \end{cases}$$

http://web.stanford.edu/class/cs224n/readings/gradient-notes.pdf

Повторение: backprop

$$\boldsymbol{z} = f(\boldsymbol{x}) \quad \frac{\partial \boldsymbol{z}}{\partial \boldsymbol{x}}? \qquad z_i = f(x_i)$$

$$(\frac{\partial \boldsymbol{z}}{\partial \boldsymbol{x}})_{ij} = \frac{\partial z_i}{\partial x_j} = \frac{\partial}{\partial x_j} f(x_i) = \begin{cases} f'(x_i) & \text{if } i = j \\ 0 & \text{if otherwise} \end{cases}$$

$$\boxed{\frac{\partial \boldsymbol{z}}{\partial \boldsymbol{x}} = \text{diag}(f'(\boldsymbol{x}))}$$

http://web.stanford.edu/class/cs224n/readings/gradient-notes.pdf

Повторение: backprop

- Матричное умножение на диагональную матрицу = поэлементное умножение строки на соответствующее значение на диагонали

```
a = np.ones((2, 2))

b = np.array([[5, 0],
              [0, 3]])

c = np.array([[5, 5],
              [3, 3]])
```

```
np.matmul(a, b)
```

```
array([[5., 3.],
       [5., 3.]])
```

```
a * c
```

```
array([[5., 5.],
       [3., 3.]])
```

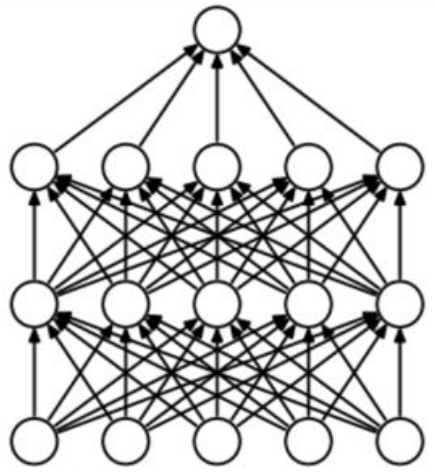
Dropout: backward и forward

```
def backward(input_tensor, mask, gradient_output, train=True):
    gradient_input = gradient_output
    if train:
        gradient_input = gradient_output * mask
    return gradient_input
```

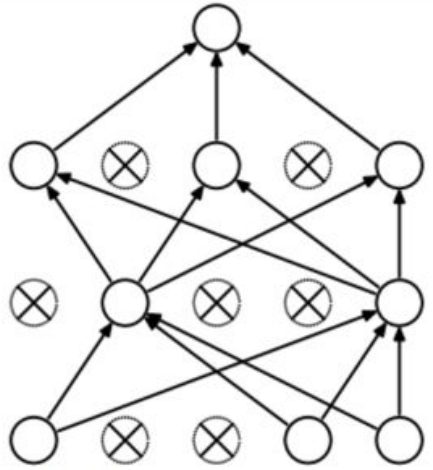
```
def dropout(input_tensor, p, train=True):
    """
    input_tensor: входной тензор
    p: вероятность обнуления
    """
    output = input_tensor
    if train:
        mask = np.random.binomial(1, 1 - p, input_tensor.shape)
        mask = np.divide(mask, 1 - p)
        output = input_tensor * mask
    return output
```

Интуиция за дропаутом

- Хотим опираться более-менее одинаково на все входные признаки, а не какие-то из них
- После применения дропаута получается меньшая сеть



(a) Standard Neural Net



(b) After applying dropout.

Реализация в Pytorch

Мы не просто занулили часть выхододов,
мы еще и разделили результат на 1 - p

В данном случае:
p = 0.8 (вероятность зануления)
1 - p = 0.2
1 / 0.2 = **5**
Мы видим, что именно число 5 получается
в итоговом тензоре

```
import torch
```

```
d = torch.nn.Dropout(p=0.8)
```

```
arr = torch.ones((4, 4))
```

```
arr
```

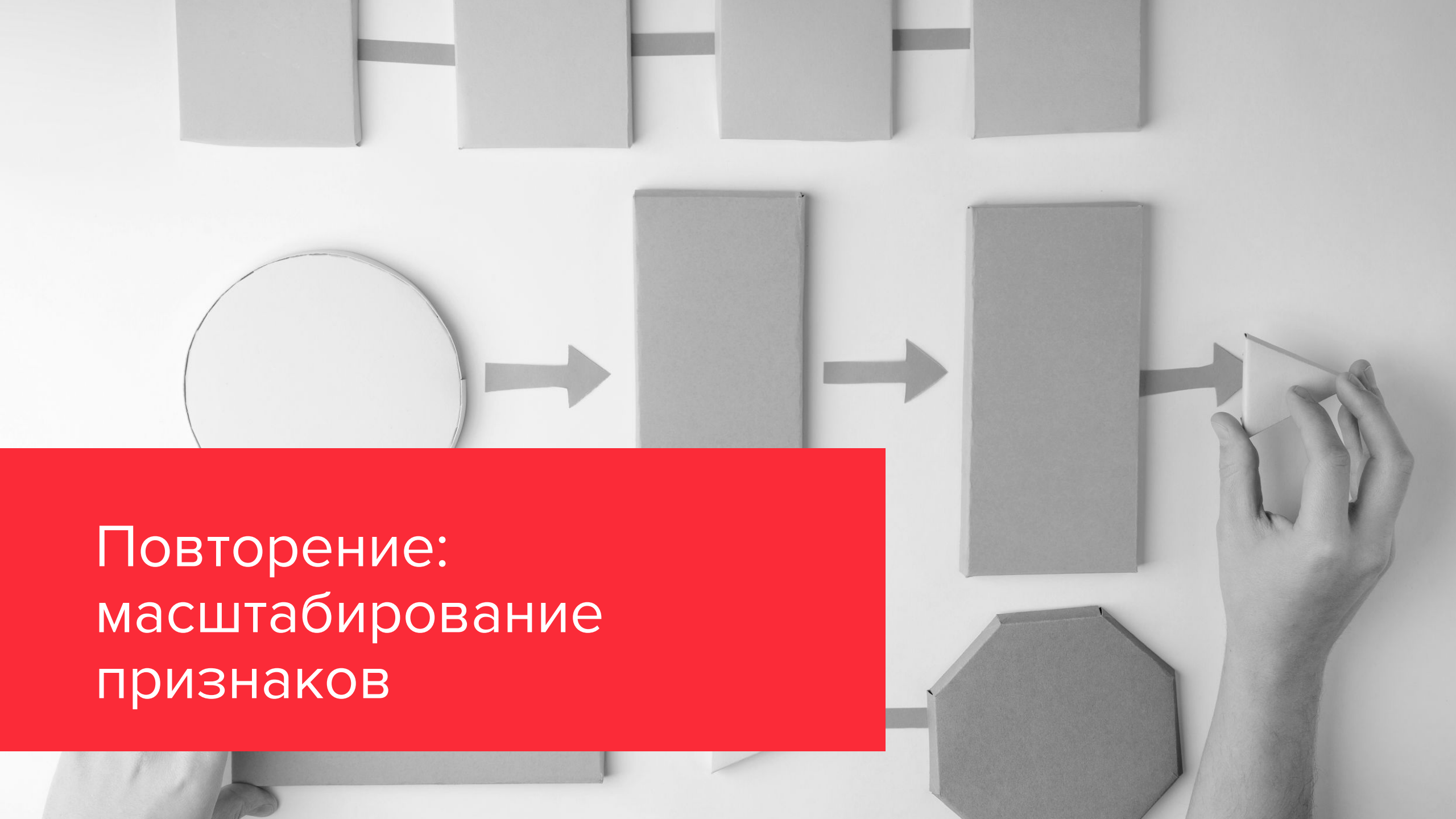
```
tensor([[1., 1., 1., 1.],
        [1., 1., 1., 1.],
        [1., 1., 1., 1.],
        [1., 1., 1., 1.]])
```

```
d(arr)
```

```
tensor([[0., 0., 0., 0.],
        [0., 0., 0., 5.],
        [0., 0., 0., 0.],
        [0., 0., 0., 0.]])
```

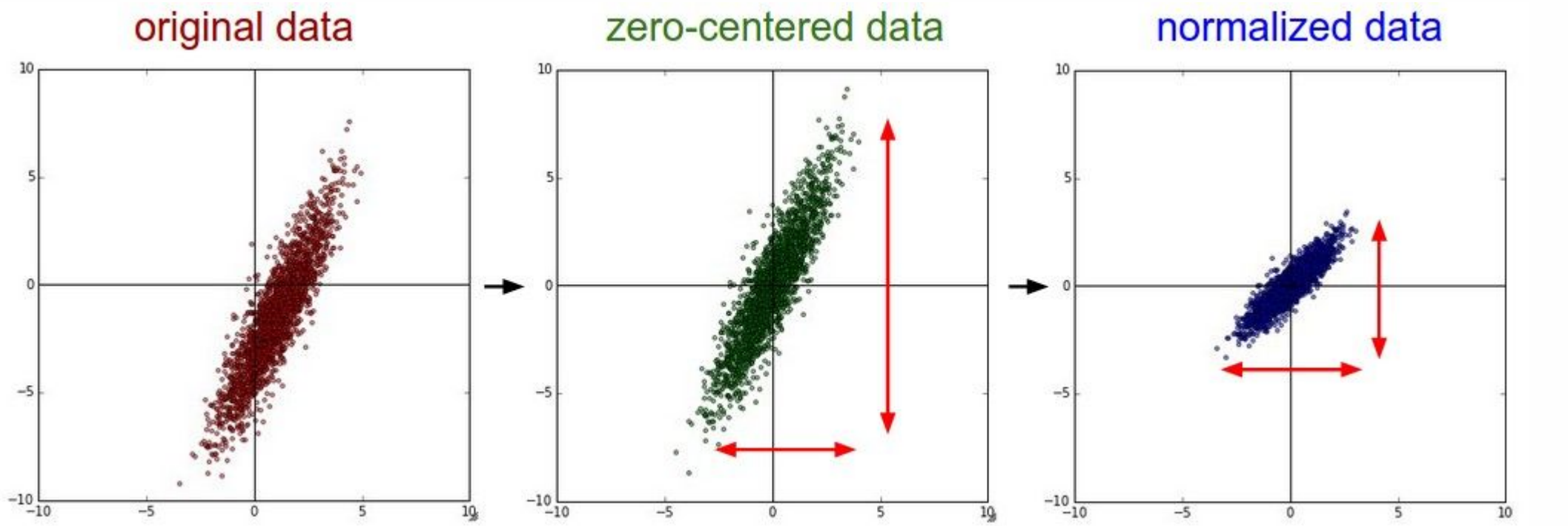
BatchNorm

- Идея и интуиция
- Работа во время обучения
- Работа во время теста
- Детали реализации



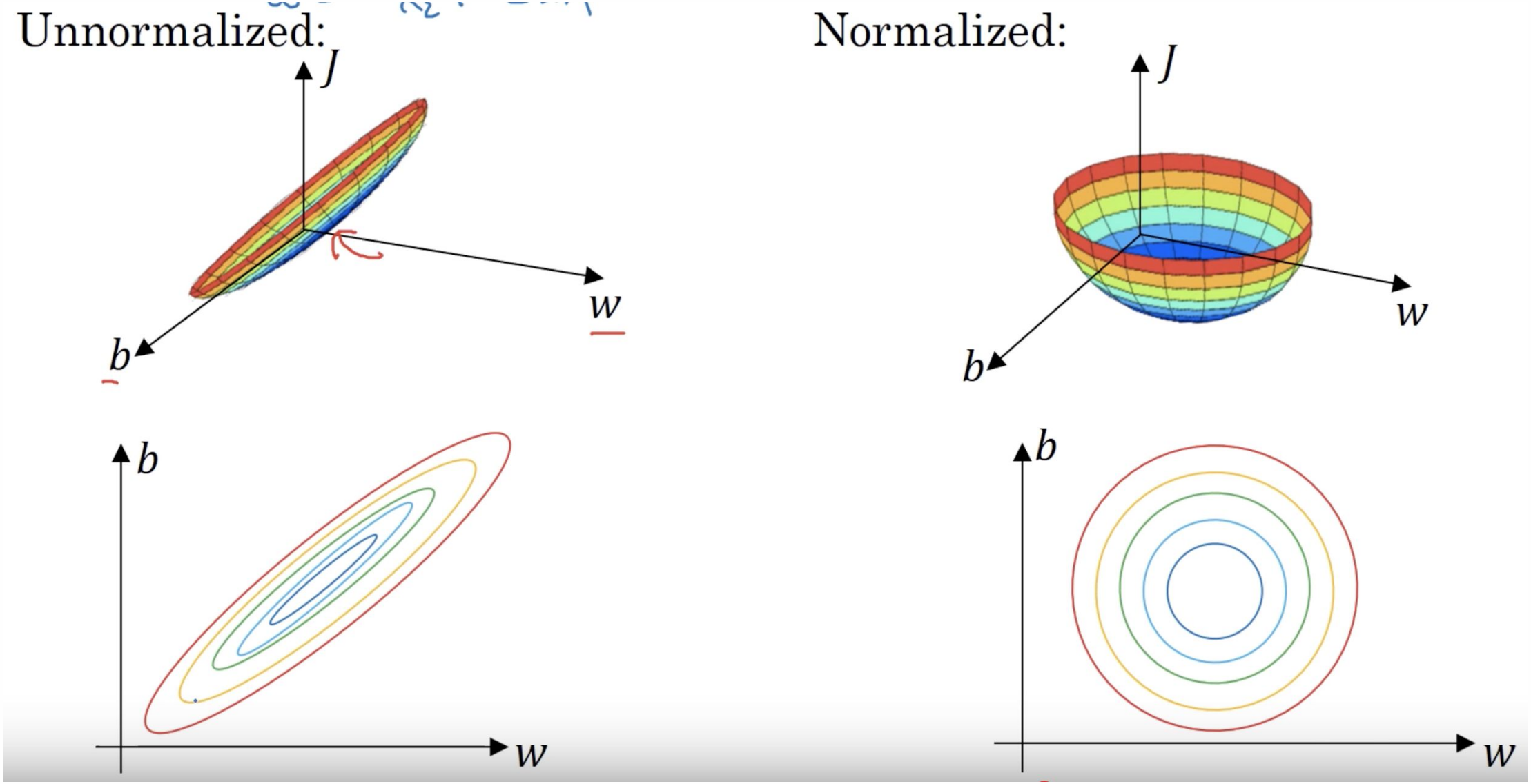
Повторение: масштабирование признаков

Повторение: масштабирование признаков



http://cs231n.github.io/neural-networks-2/

Повторение: масштабирование признаков

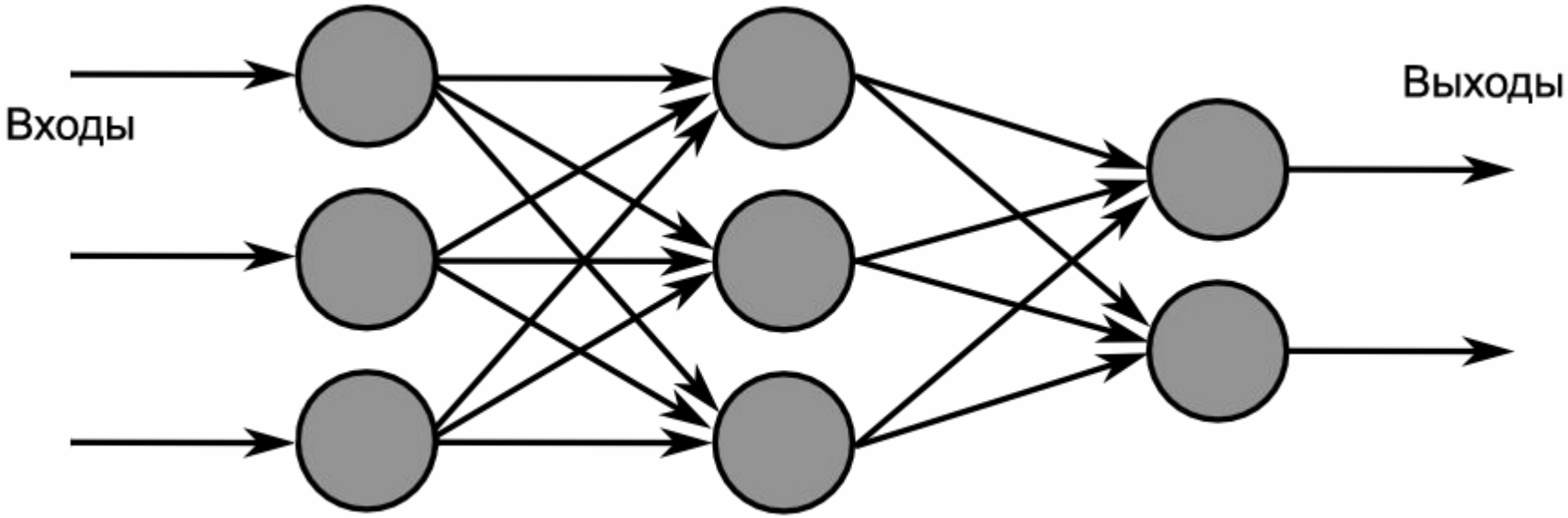


https://www.youtube.com/watch?v=FDCfw-YgWTE

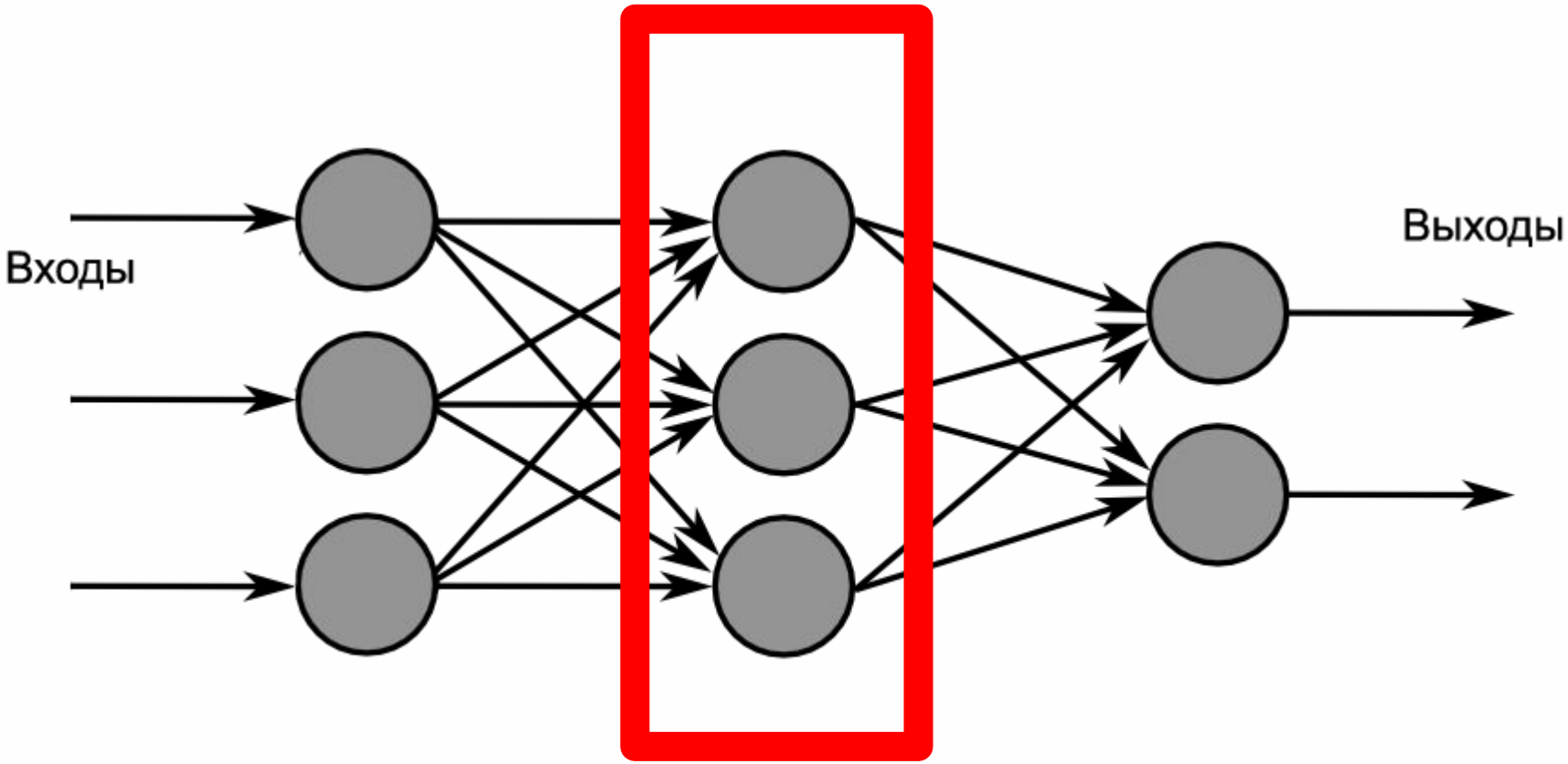
Повторение: масштабирование признаков

https://www.youtube.com/watch?v=FDCfw-YgWTE

BatchNorm: идея



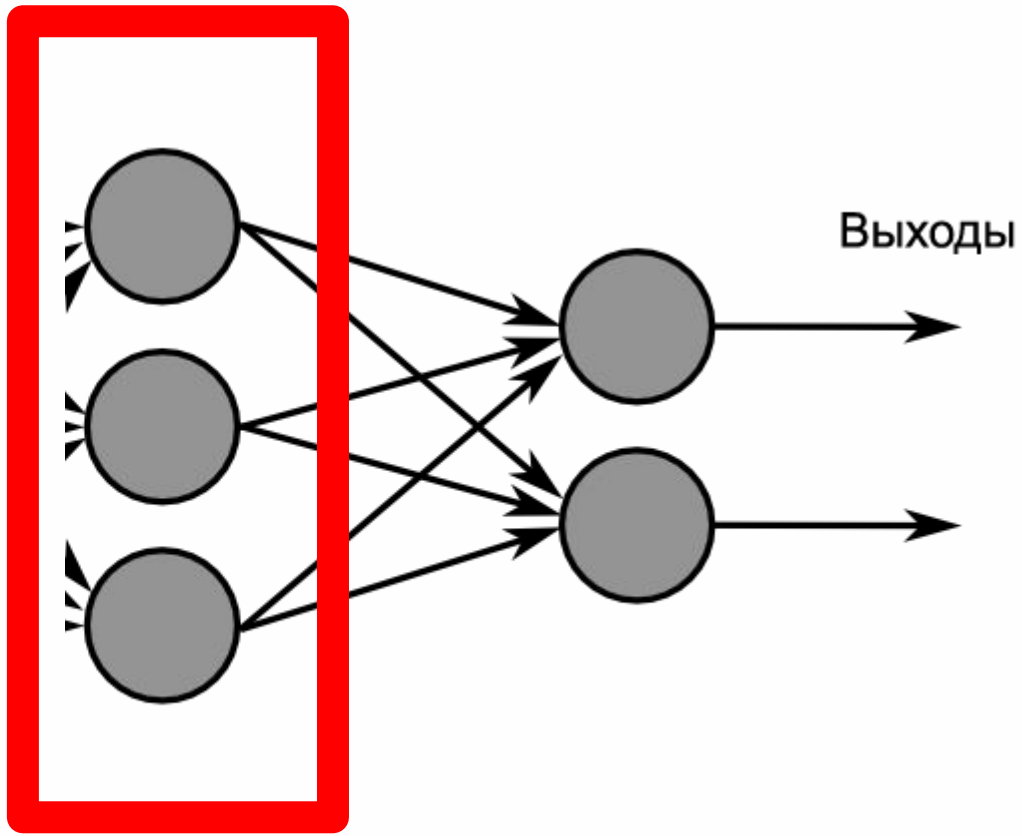
BatchNorm: идея



BatchNorm: идея

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i$$

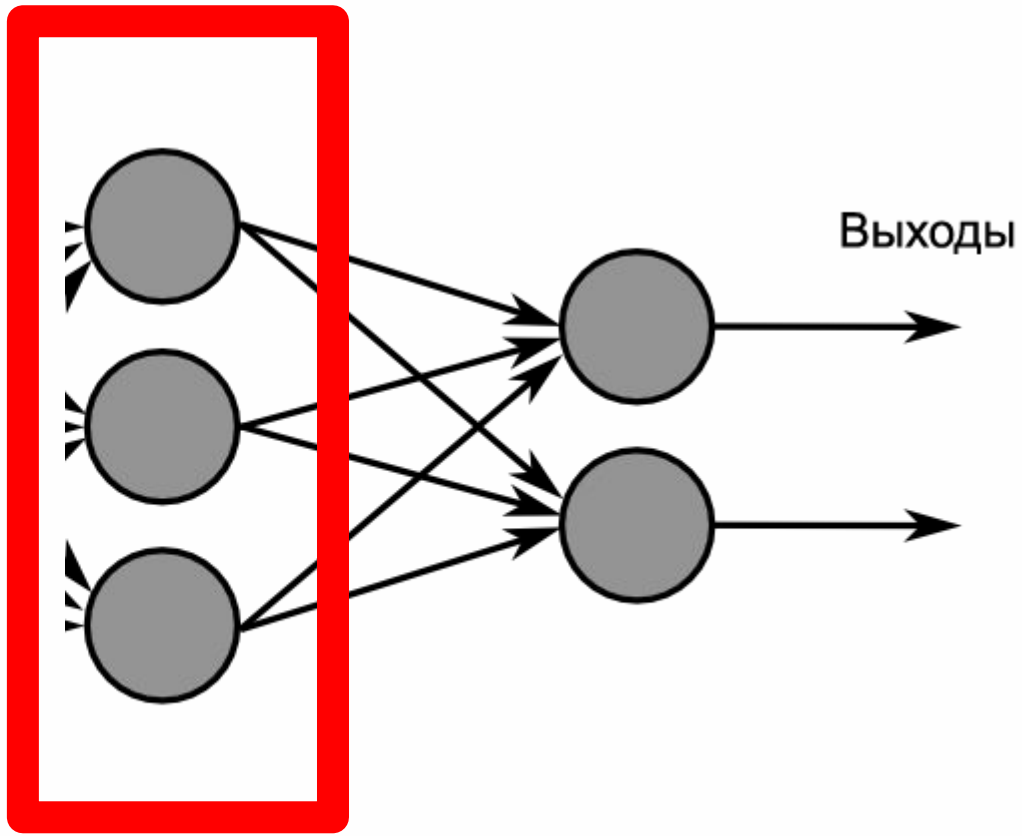
// mini-batch mean



BatchNorm: идея

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

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

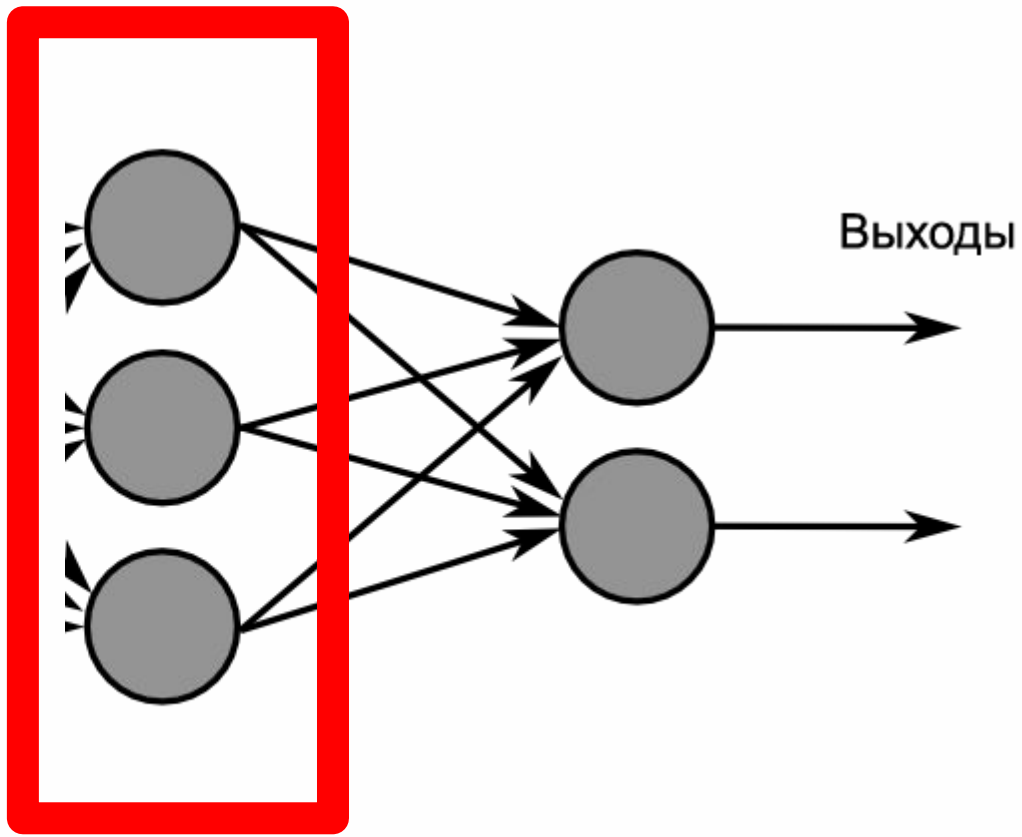


BatchNorm: идея

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

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

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}} \qquad \text{// normalize}$$



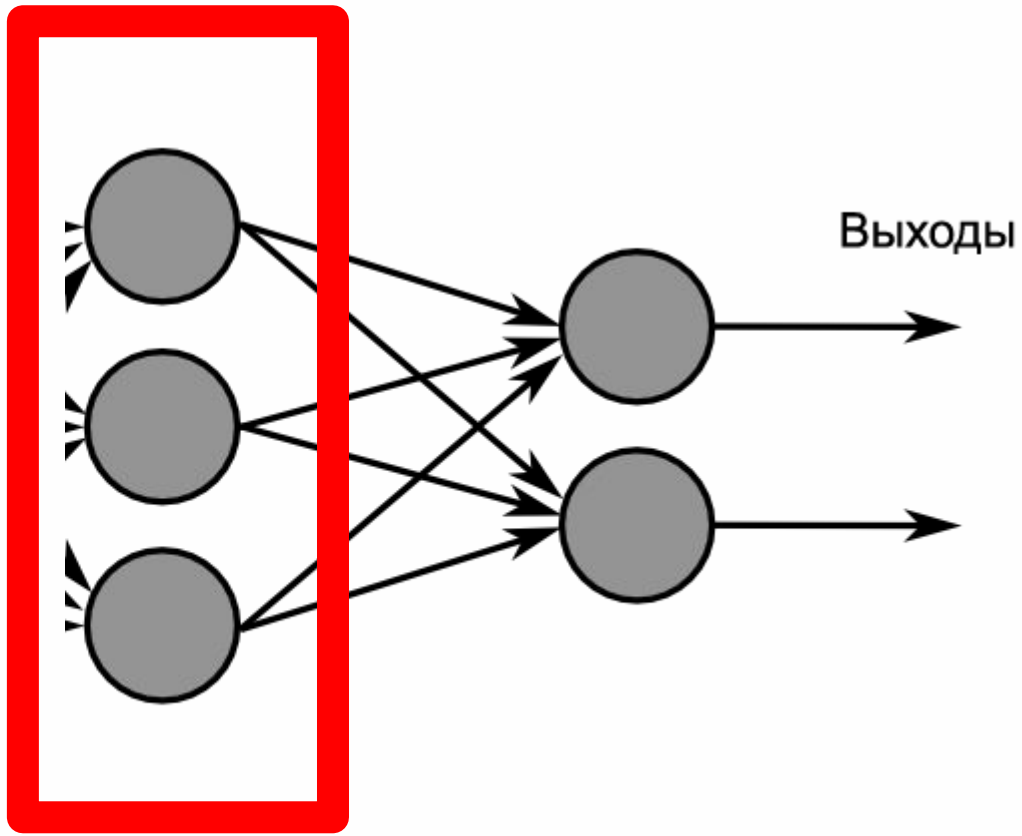
BatchNorm: идея

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

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

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}} \qquad \text{// normalize}$$

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



BatchNorm: идея

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β

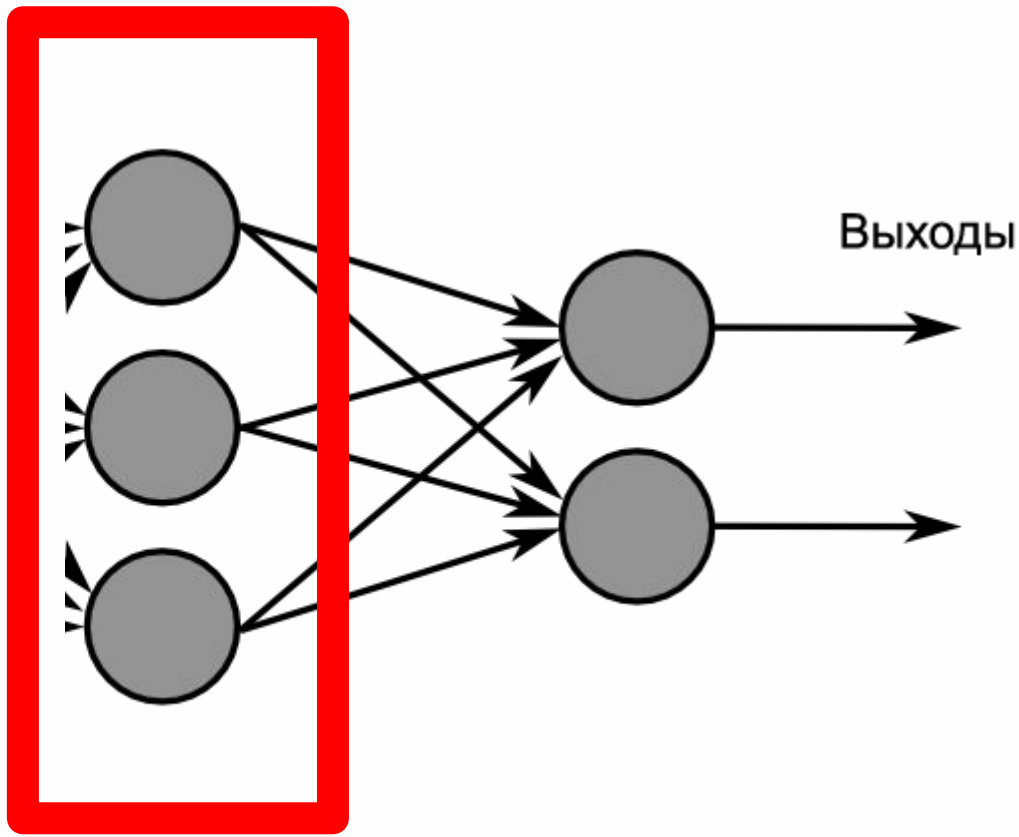
Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

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

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

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}} \qquad \text{// normalize}$$

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



Batchnorm: обучение

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β

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

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

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

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}} \qquad \text{// normalize}$$

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

Batchnorm: обучение

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β
Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

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

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

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}} \qquad \text{// normalize}$$

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

Чему равен **y**, если:

$$\gamma = \sqrt{\sigma^2 + \epsilon}$$

$$\beta = \mu$$

Batchnorm: обучение

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β
Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

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

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

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}} \qquad \text{// normalize}$$

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

Чему равен **y**, если:

$$\gamma = \sqrt{\sigma^2 + \epsilon}$$

$$\beta = \mu$$

$$y_i = x_i$$

Batchnorm: тест

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β
Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

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

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

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}} \qquad \text{// normalize}$$

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

Batchnorm: тест

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β

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

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

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

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}} \qquad \text{// normalize}$$

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

Batchnorm: тест

Проблема: во время теста непонятно откуда брать смещение и разброс

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β

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

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

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

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}} \qquad \text{// normalize}$$

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

Batchnorm: тест

Проблема: во время теста непонятно откуда брать смещение и разброс

Решение:

накапливать знания о смещении и разбросе во время обучения

$$\hat{\mu}_t = \alpha * \mu_t + (1 - \alpha) * \hat{\mu_{t-1}}$$

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β

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

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

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

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}} \qquad \text{// normalize}$$

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

Batchnorm: backprop

$$y_i = \gamma \hat{x}_i + \beta$$

$$\frac{\partial y_i}{\partial \gamma} =$$

Batchnorm: backprop

$$y_i = \gamma \hat{x}_i + \beta$$

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

Batchnorm: backprop

$$y_i = \gamma \hat{x}_i + \beta$$

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

Batchnorm: backprop

$$y_i = \gamma \hat{x}_i + \beta$$

$$\frac{\partial y_i}{\partial \gamma} = \hat{x}_i \qquad \frac{\partial y_i}{\partial \beta} =$$

Batchnorm: backprop

$$y_i = \gamma \hat{x}_i + \beta$$

$$\frac{\partial y_i}{\partial \gamma} = \hat{x}_i \qquad \frac{\partial y_i}{\partial \beta} = I$$

Batchnorm: backprop

$$y_i = \gamma \hat{x}_i + \beta$$

$$\frac{\partial y_i}{\partial \beta} = I$$

$$z = x$$

http://web.stanford.edu/class/cs224n/readings/gradient-notes.pdf

Batchnorm: backprop

$$y_i = \gamma \hat{x}_i + \beta$$

$$\frac{\partial y_i}{\partial \beta} = I$$

$$z = x$$

$$z_i = x_i$$

http://web.stanford.edu/class/cs224n/readings/gradient-notes.pdf

Batchnorm: backprop

$$y_i = \gamma \hat{x}_i + \beta$$

$$\frac{\partial y_i}{\partial \beta} = I$$

$$z = x$$

$$z_i = x_i$$

$$\left(\frac{\partial \boldsymbol{z}}{\partial \boldsymbol{x}}\right)_{ij} =$$

http://web.stanford.edu/class/cs224n/readings/gradient-notes.pdf

Batchnorm: backprop

$$y_i = \gamma \hat{x}_i + \beta$$

$$\frac{\partial y_i}{\partial \beta} = I$$

$$z = x$$

$$z_i = x_i$$

$$\left(\frac{\partial \boldsymbol{z}}{\partial \boldsymbol{x}}\right)_{ij} = \frac{\partial z_i}{\partial x_j} =$$

http://web.stanford.edu/class/cs224n/readings/gradient-notes.pdf

Batchnorm: backprop

$$y_i = \gamma \hat{x}_i + \beta$$

$$\frac{\partial y_i}{\partial \beta} = I$$

$$z = x$$

$$z_i = x_i$$

$$\left(\frac{\partial \boldsymbol{z}}{\partial \boldsymbol{x}}\right)_{ij} = \frac{\partial z_i}{\partial x_j} = \frac{\partial}{\partial x_j} x_i =$$

http://web.stanford.edu/class/cs224n/readings/gradient-notes.pdf

Batchnorm: backprop

$$y_i = \gamma \hat{x}_i + \beta$$

$$\frac{\partial y_i}{\partial \beta} = I$$

$$z = x$$

$$z_i = x_i$$

$$\left(\frac{\partial \boldsymbol{z}}{\partial \boldsymbol{x}}\right)_{ij} = \frac{\partial z_i}{\partial x_j} = \frac{\partial}{\partial x_j} x_i = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{if otherwise} \end{cases}$$

http://web.stanford.edu/class/cs224n/readings/gradient-notes.pdf

Batchnorm: backprop

$$y_i = \gamma \hat{x}_i + \beta$$

$$\frac{\partial y_i}{\partial \beta} = I$$

$$z = x$$

$$z_i = x_i$$

$$\left(\frac{\partial \boldsymbol{z}}{\partial \boldsymbol{x}}\right)_{ij} = \frac{\partial z_i}{\partial x_j} = \frac{\partial}{\partial x_j} x_i = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{if otherwise} \end{cases}$$

$$\boxed{\frac{\partial \boldsymbol{z}}{\partial \boldsymbol{x}} = \boldsymbol{I}}$$

http://web.stanford.edu/class/cs224n/readings/gradient-notes.pdf

Batchnorm: backprop

$$y_i = \gamma \hat{x}_i + \beta$$

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

http://web.stanford.edu/class/cs224n/readings/gradient-notes.pdf

Batchnorm: backprop

$$y_i = \gamma \hat{x}_i + \beta$$

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

$$\boldsymbol{z} = \boldsymbol{W}\boldsymbol{x}$$

http://web.stanford.edu/class/cs224n/readings/gradient-notes.pdf

Batchnorm: backprop

$$y_i = \gamma \hat{x}_i + \beta$$

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

$$\boldsymbol{z} = \boldsymbol{W}\boldsymbol{x}$$

$$z_i = \sum_{k=1}^{m} W_{ik} x_k$$

http://web.stanford.edu/class/cs224n/readings/gradient-notes.pdf

Batchnorm: backprop

$$y_i = \gamma \hat{x}_i + \beta$$

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

$$\boldsymbol{z} = \boldsymbol{W}\boldsymbol{x}$$

$$z_i = \sum_{k=1}^{m} W_{ik} x_k$$

http://web.stanford.edu/class/cs224n/readings/gradient-notes.pdf

Batchnorm: backprop

$$y_i = \gamma \hat{x}_i + \beta$$

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

$$\boldsymbol{z} = \boldsymbol{W}\boldsymbol{x} \qquad z_i = \sum_{k=1}^{m} W_{ik} x_k$$

$$(\frac{\partial \boldsymbol{z}}{\partial \boldsymbol{x}})_{ij} = \frac{\partial z_i}{\partial x_j} = \frac{\partial}{\partial x_j} \sum_{k=1}^{m} W_{ik} x_k = \sum_{k=1}^{m} W_{ik} \frac{\partial}{\partial x_j} x_k = W_{ij}$$

http://web.stanford.edu/class/cs224n/readings/gradient-notes.pdf

Batchnorm: backprop

$$y_i = \gamma \hat{x}_i + \beta$$

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

$$\boldsymbol{z} = \boldsymbol{W}\boldsymbol{x} \qquad z_i = \sum_{k=1}^{m} W_{ik} x_k$$

$$\left(\frac{\partial \boldsymbol{z}}{\partial \boldsymbol{x}}\right)_{ij} =$$

http://web.stanford.edu/class/cs224n/readings/gradient-notes.pdf

Batchnorm: backprop

$$y_i = \gamma \hat{x}_i + \beta$$

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

$$\boldsymbol{z} = \boldsymbol{W}\boldsymbol{x} \qquad z_i = \sum_{k=1}^{m} W_{ik} x_k$$

$$\left(\frac{\partial \boldsymbol{z}}{\partial \boldsymbol{x}}\right)_{ij} = \frac{\partial z_i}{\partial x_j} =$$

http://web.stanford.edu/class/cs224n/readings/gradient-notes.pdf

Batchnorm: backprop

$$y_i = \gamma \hat{x}_i + \beta$$

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

$$\boldsymbol{z} = \boldsymbol{W}\boldsymbol{x} \qquad z_i = \sum_{k=1}^{m} W_{ik} x_k$$

$$\left(\frac{\partial \boldsymbol{z}}{\partial \boldsymbol{x}}\right)_{ij} = \frac{\partial z_i}{\partial x_j} = \frac{\partial}{\partial x_j} \sum_{k=1}^{m} W_{ik} x_k =$$

http://web.stanford.edu/class/cs224n/readings/gradient-notes.pdf

Batchnorm: backprop

$$y_i = \gamma \hat{x}_i + \beta$$

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

$$\boldsymbol{z} = \boldsymbol{W}\boldsymbol{x} \qquad z_i = \sum_{k=1}^{m} W_{ik} x_k$$

$$(\frac{\partial \boldsymbol{z}}{\partial \boldsymbol{x}})_{ij} = \frac{\partial z_i}{\partial x_j} = \frac{\partial}{\partial x_j} \sum_{k=1}^{m} W_{ik} x_k = \sum_{k=1}^{m} W_{ik} \frac{\partial}{\partial x_j} x_k =$$

http://web.stanford.edu/class/cs224n/readings/gradient-notes.pdf

Batchnorm: backprop

$$y_i = \gamma \hat{x}_i + \beta$$

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

$$\boldsymbol{z} = \boldsymbol{W}\boldsymbol{x} \qquad z_i = \sum_{k=1}^{m} W_{ik} x_k$$

$$(\frac{\partial \boldsymbol{z}}{\partial \boldsymbol{x}})_{ij} = \frac{\partial z_i}{\partial x_j} = \frac{\partial}{\partial x_j} \sum_{k=1}^{m} W_{ik} x_k = \sum_{k=1}^{m} W_{ik} \frac{\partial}{\partial x_j} x_k = W_{ij}$$

http://web.stanford.edu/class/cs224n/readings/gradient-notes.pdf

Batchnorm: backprop

$$y_i = \gamma \hat{x}_i + \beta$$

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

$$\boldsymbol{z} = \boldsymbol{W}\boldsymbol{x} \qquad z_i = \sum_{k=1}^{m} W_{ik} x_k$$

$$\boxed{\frac{\partial \boldsymbol{z}}{\partial \boldsymbol{x}} = \boldsymbol{W}}$$

$$\left(\frac{\partial \boldsymbol{z}}{\partial \boldsymbol{x}}\right)_{ij} = \frac{\partial z_i}{\partial x_j} = \frac{\partial}{\partial x_j} \sum_{k=1}^{m} W_{ik} x_k = \sum_{k=1}^{m} W_{ik} \frac{\partial}{\partial x_j} x_k = W_{ij}$$

http://web.stanford.edu/class/cs224n/readings/gradient-notes.pdf

Batchnorm: backprop

$$y_i = \gamma \hat{x}_i + \beta$$

$$\frac{\partial y_i}{\partial \gamma} = \hat{x}_i \qquad \frac{\partial y_i}{\partial \beta} = I$$

http://web.stanford.edu/class/cs224n/readings/gradient-notes.pdf

Batchnorm: backprop

$$y_i = \gamma \hat{x}_i + \beta$$

$$\frac{\partial y_i}{\partial \gamma} = \hat{x}_i \qquad \frac{\partial y_i}{\partial \beta} = I$$

```
def backward(input_tensor, gradient_output):
        gradient_beta = np.sum(gradient_output, axis=0)
        gradient_gamma = np.sum(input_tensor * gradient_output, axis=0)
```

Batchnorm: backprop

$$\frac{\partial y_i}{\partial x_i} = ?$$

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β

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

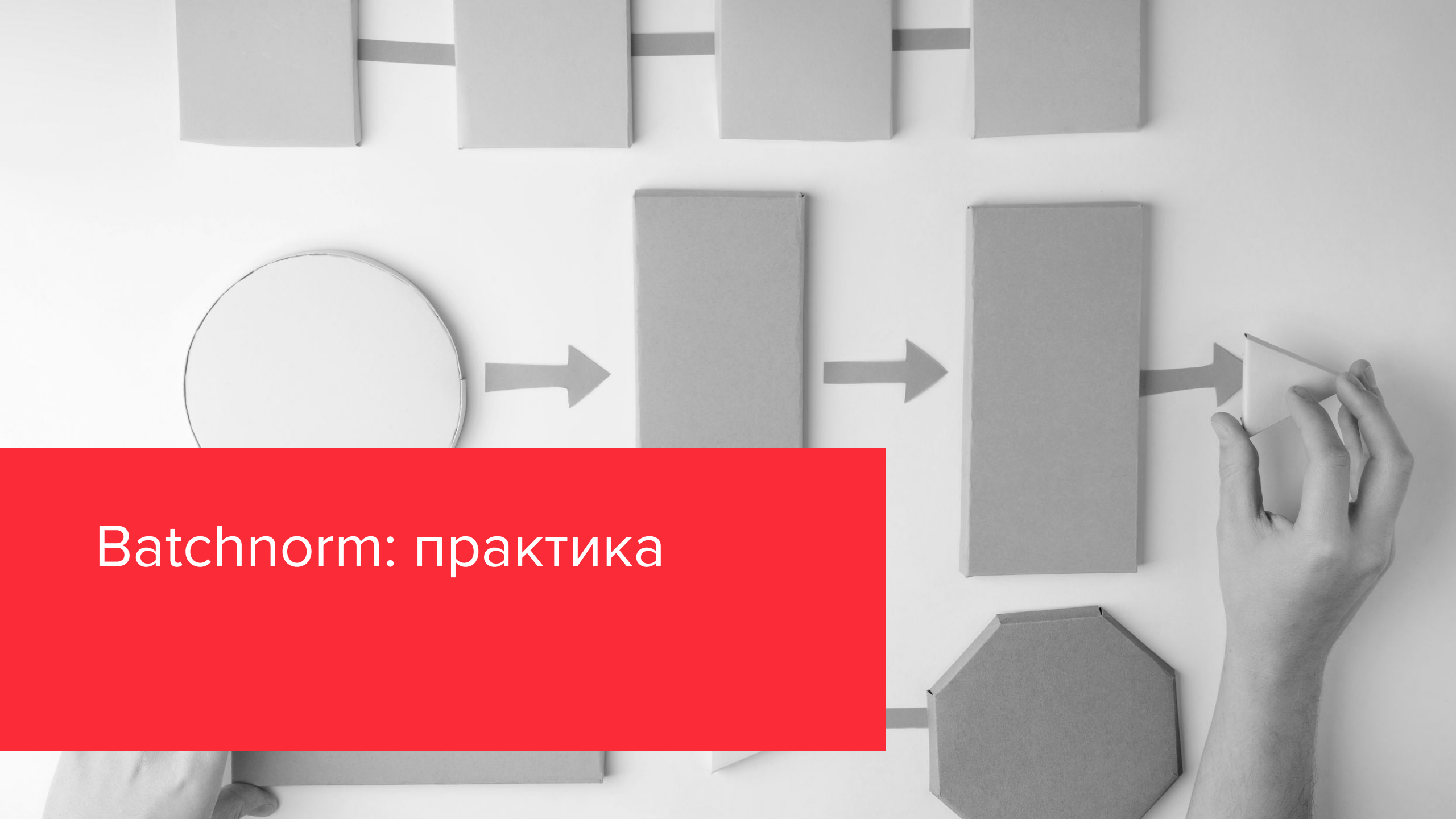
$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

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

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}} \qquad \text{// normalize}$$

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

https://kratzert.github.io/2016/02/12/understanding-the-gradient-flow-through-the-batch-normalization-layer.html



Batchnorm: практика

SyncBatchNorm

SyncBatchNorm

CLASS `torch.nn.SyncBatchNorm(num_features, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True, process_group=None)` [SOURCE]

Applies Batch Normalization over a N-Dimensional input (a mini-batch of [N-2]D inputs with additional channel dimension) as described in the paper Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift .

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

The mean and standard-deviation are calculated per-dimension over all mini-batches of the same process groups. γ and β are learnable parameter vectors of size C (where C is the input size). By default, the elements of γ are sampled from $\mathcal{U}(0, 1)$ and the elements of β are set to 0.

Also by default, during training this layer keeps running estimates of its computed mean and variance, which are then used for normalization during evaluation. The running estimates are kept with a default `momentum` of 0.1.

If `track_running_stats` is set to `False`, this layer then does not keep running estimates, and batch statistics are instead used during evaluation time as well.

https://pytorch.org/docs/master/nn.html#torch.nn.SyncBatchNorm

Повторение: Resnet

Повторение: Resnet

Наблюдение: наращивание глубины может ухудшить качество работы сети

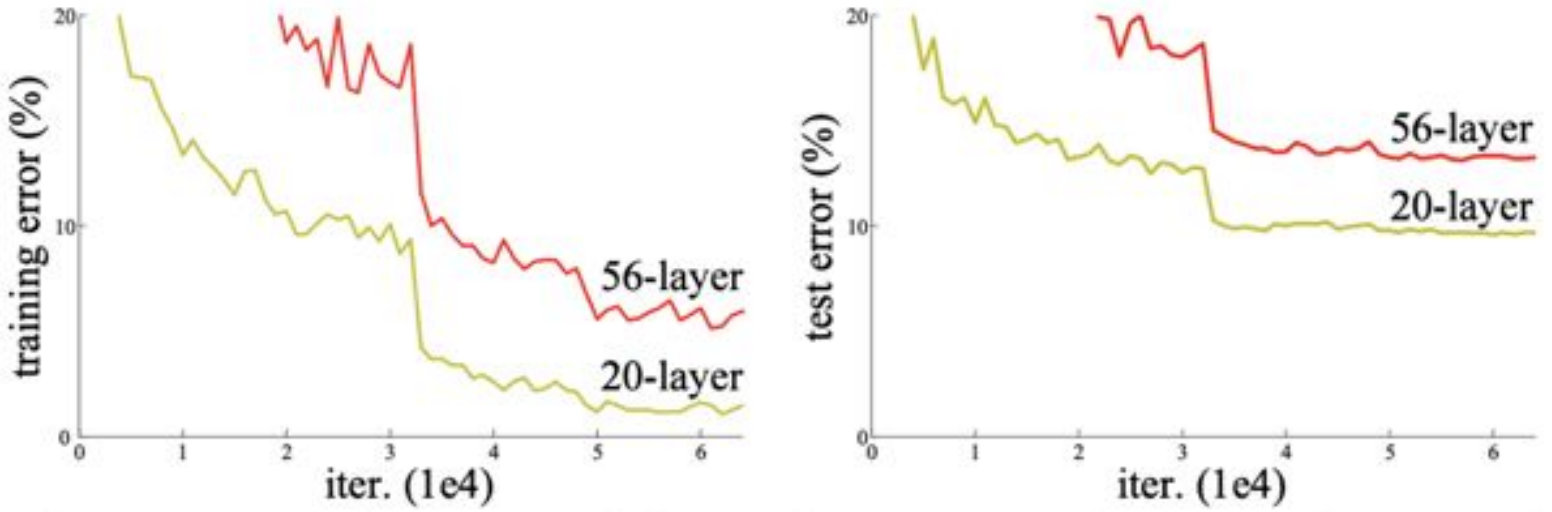


Figure 1. Training error (left) and test error (right) on CIFAR-10 with 20-layer and 56-layer "plain" networks. The deeper network has higher training error, and thus test error. Similar phenomena on ImageNet is presented in Fig. 4.

- https://arxiv.org/abs/1512.03385

Повторение: Resnet

- Идея: позволить сети “пропускать” слои, если они не способствуют улучшению

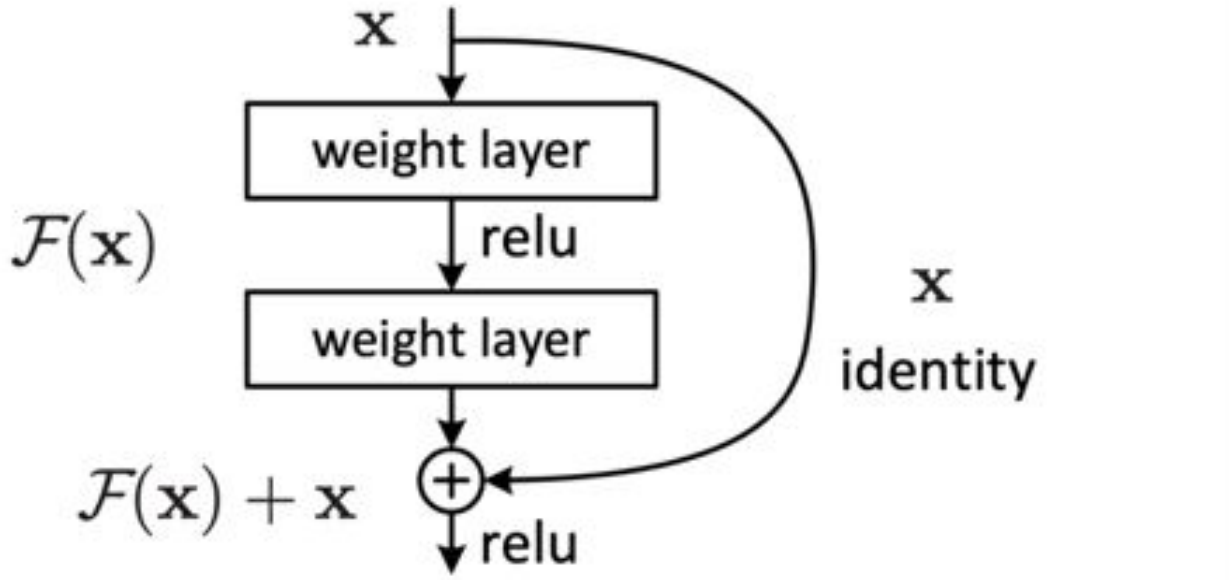
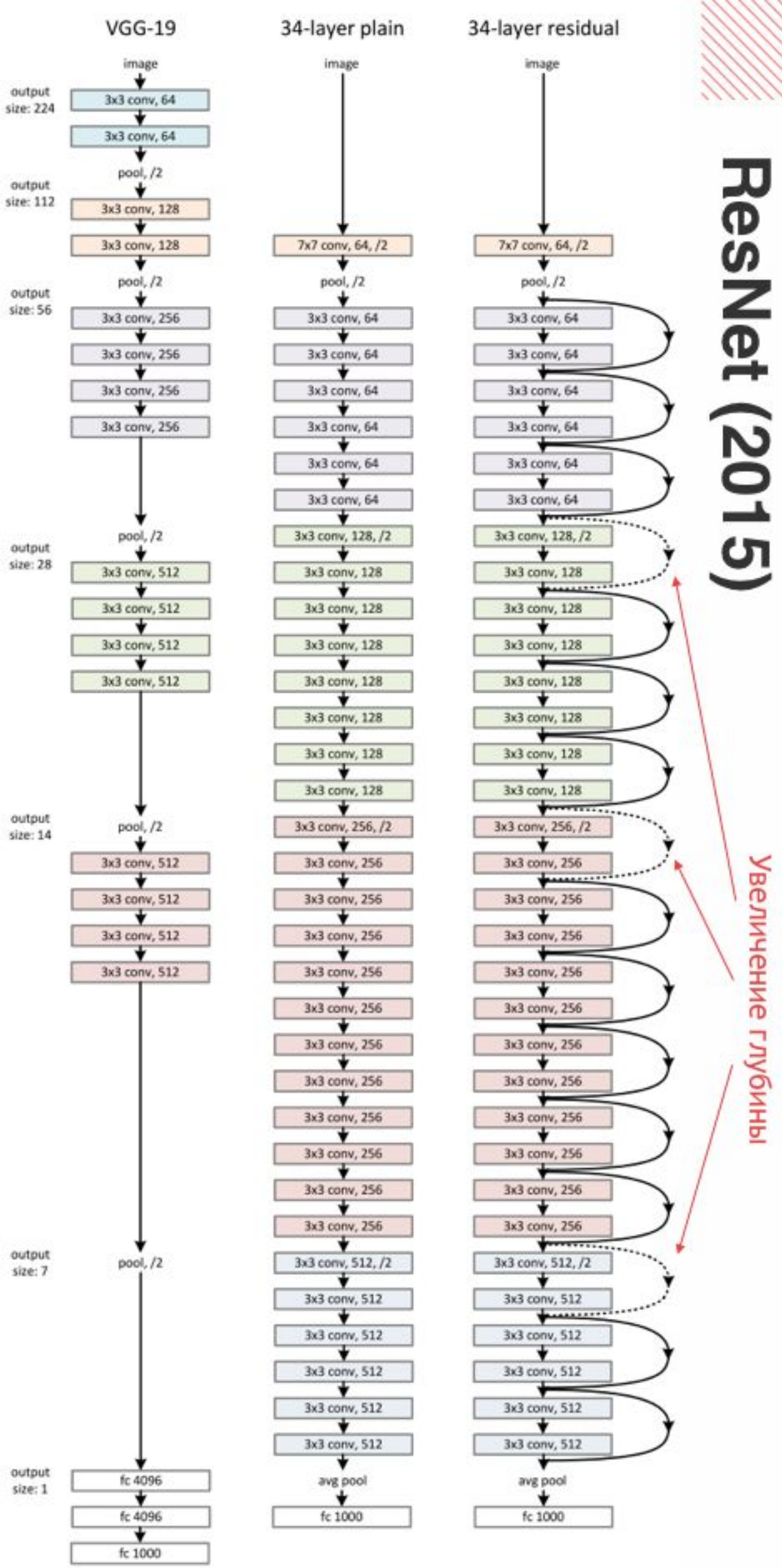


Figure 2. Residual learning: a building block.

- https://arxiv.org/abs/1512.03385


VGG-19
34-layer plain
34-layer residual
image
output size: 224
3x3 conv, 64
3x3 conv, 64
pool, /2
output size: 112
3x3 conv, 128
3x3 conv, 128
pool, /2
output size: 56
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
pool, /2
output size: 28
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
pool, /2
output size: 14
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
pool, /2
output size: 7
fc 4096
fc 4096
fc 1000
output size: 1
7x7 conv, 64, /2
pool, /2
3x3 conv, 64
3x3 conv, 128, /2
3x3 conv, 128
3x3 conv, 256, /2
3x3 conv, 256
3x3 conv, 512, /2
3x3 conv, 512
avg pool
fc 1000
ResNet (2015)
Увеличение глубины

Повторение: Resnet

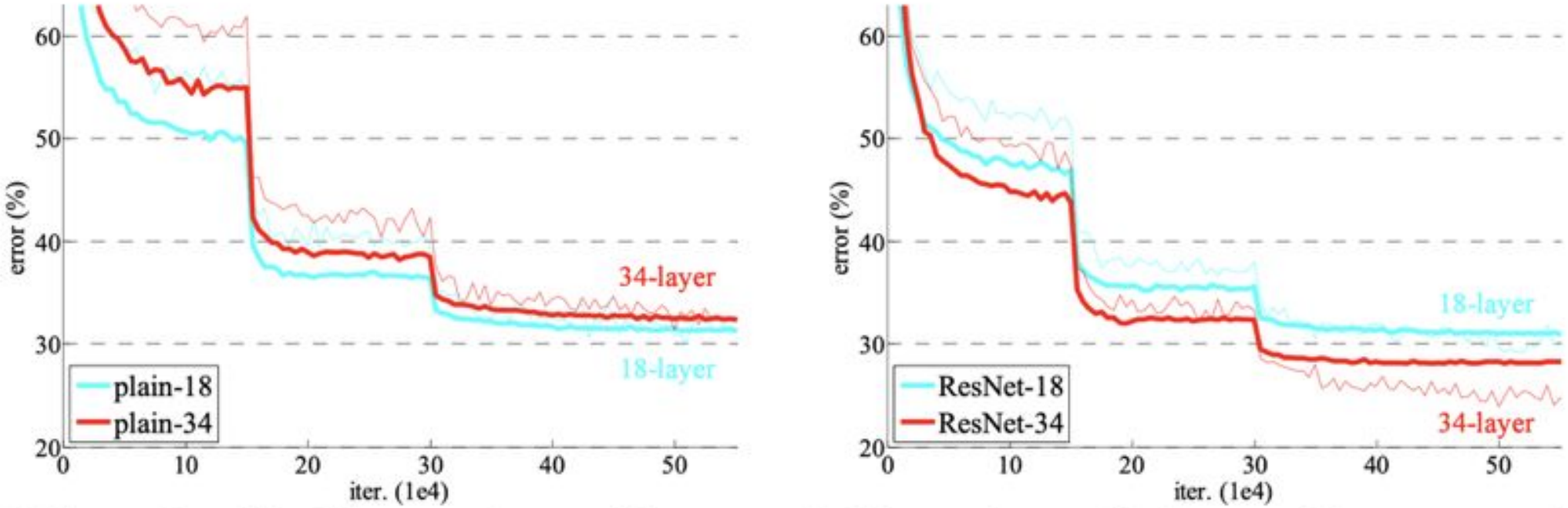


Figure 4. Training on **ImageNet**. Thin curves denote training error, and bold curves denote validation error of the center crops. Left: plain networks of 18 and 34 layers. Right: ResNets of 18 and 34 layers. In this plot, the residual networks have no extra parameter compared to their plain counterparts.

Batchnorm: применение в моделях

```
from torchvision.models import resnet18
```

```
model = resnet18()
```

```
model
```

```
ResNet(
  (conv1): Conv2d(3, 64, kernel_size=(7, 7), stride=(2, 2), padding=(3, 3), bias=False)
  (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (relu): ReLU(inplace)
  (maxpool): MaxPool2d(kernel_size=3, stride=2, padding=1, dilation=1, ceil_mode=False)
  (layer1): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace)
      (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
    (1): BasicBlock(
      (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace)
      (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
```

Batchnorm: применение в моделях

```
from torchvision.models import resnet18
```

```
model = resnet18()
```

```
model
```

```
ResNet(
  (conv1): Conv2d(3, 64, kernel_size=(7, 7), stride=(2, 2), padding=(3, 3), bias=False)
  (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (relu): ReLU(inplace)
  (maxpool): MaxPool2d(kernel_size=3, stride=2, padding=1, dilation=1, ceil_mode=False)
  (layer1): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace)
      (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
    (1): BasicBlock(
      (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace)
      (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
```

Batchnorm: применение в моделях

```
from torchvision.models import resnet18
```

```
model = resnet18()
```

```
model
```

```
ResNet(
  (conv1): Conv2d(3, 64, kernel_size=(7, 7), stride=(2, 2), padding=(3, 3), bias=False)
  (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (relu): ReLU(inplace)
  (maxpool): MaxPool2d(kernel_size=3, stride=2, padding=1, dilation=1, ceil_mode=False)
```

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}} \qquad \text{// normalize}$$

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

Batchnorm: применение в моделях

```
from torchvision.models import resnet18
```

```
model = resnet18()
```

```
model
```

```
ResNet(
  (conv1): Conv2d(3, 64, kernel_size=(7, 7), stride=(2, 2), padding=(3, 3), bias=False)
  (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (relu): ReLU(inplace)
  (maxpool): MaxPool2d(kernel_size=3, stride=2, padding=1, dilation=1, ceil_mode=False)
```

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}} \qquad \text{// normalize}$$

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

$$x_i = conv(x_i) + bias$$

Batchnorm: применение в моделях

```
from torchvision.models import resnet18
```

```
model = resnet18()
```

```
model
```

```
ResNet(
  (conv1): Conv2d(3, 64, kernel_size=(7, 7), stride=(2, 2), padding=(3, 3), bias=False)
  (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (relu): ReLU(inplace)
  (maxpool): MaxPool2d(kernel_size=3, stride=2, padding=1, dilation=1, ceil_mode=False)
```

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}} \qquad \text{// normalize}$$

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

$$x_i = conv(x_i) + \cancel{bias}$$

Batchnorm: какая размерность весов?

```
batchrnorm = torch.nn.BatchNorm2d(64)
```

```
batchrnorm.weight.shape
```

Batchnorm: какая размерность весов?

```
batchrnorm = torch.nn.BatchNorm2d(64)
```

```
batchrnorm.weight.shape
```

```
torch.Size([64])
```

```
batchrnorm.bias.shape
```

```
torch.Size([64])
```

Альтернативные нормализации

- Популярные решения:
  - Group Normalization
  - Weight Normalization
  - Layer Normalization
  - Instance Normalization
  - Batch Renormalization
  - Batch-Instance Normalization
  - Обзор: https://mlexplained.com/2018/11/30/an-overview-of-normalization-methods-in-deep-learning/

Аугментации

Аугментации

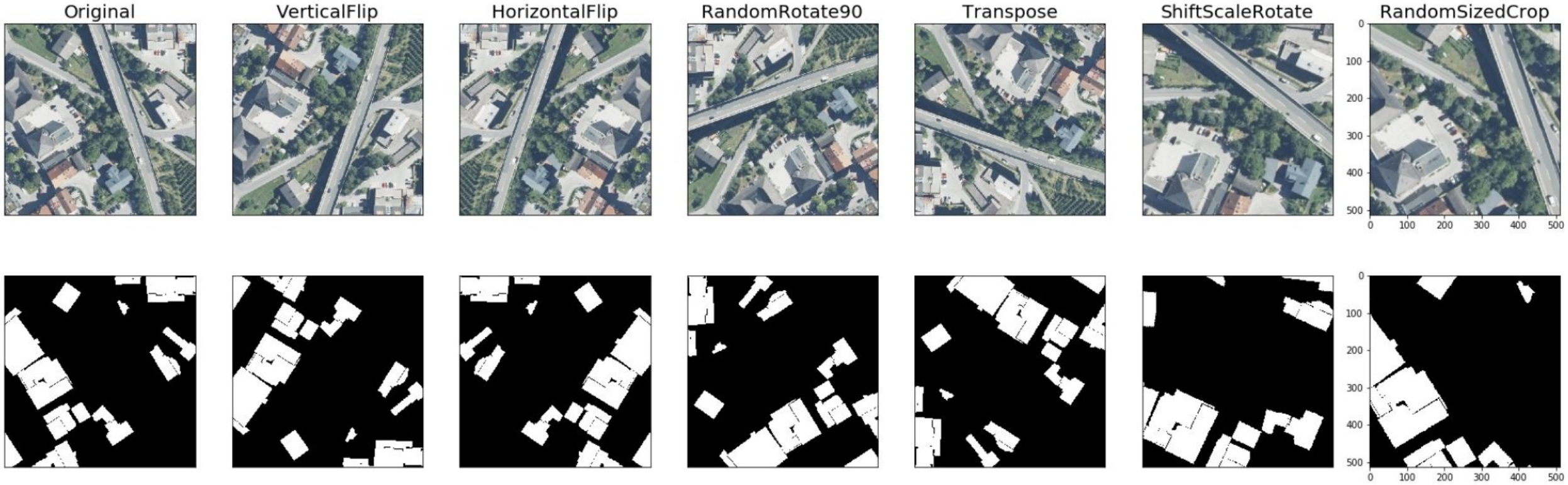
- С помощью аугментаций можно дешево увеличить объем обучающей выборки
- Популярные решения:
  - torchvision.transforms
  - albumentations
  - imgaug

Аугментации: примеры

https://github.com/albumentations-team/albumentations

https://github.com/aleju/imgaug

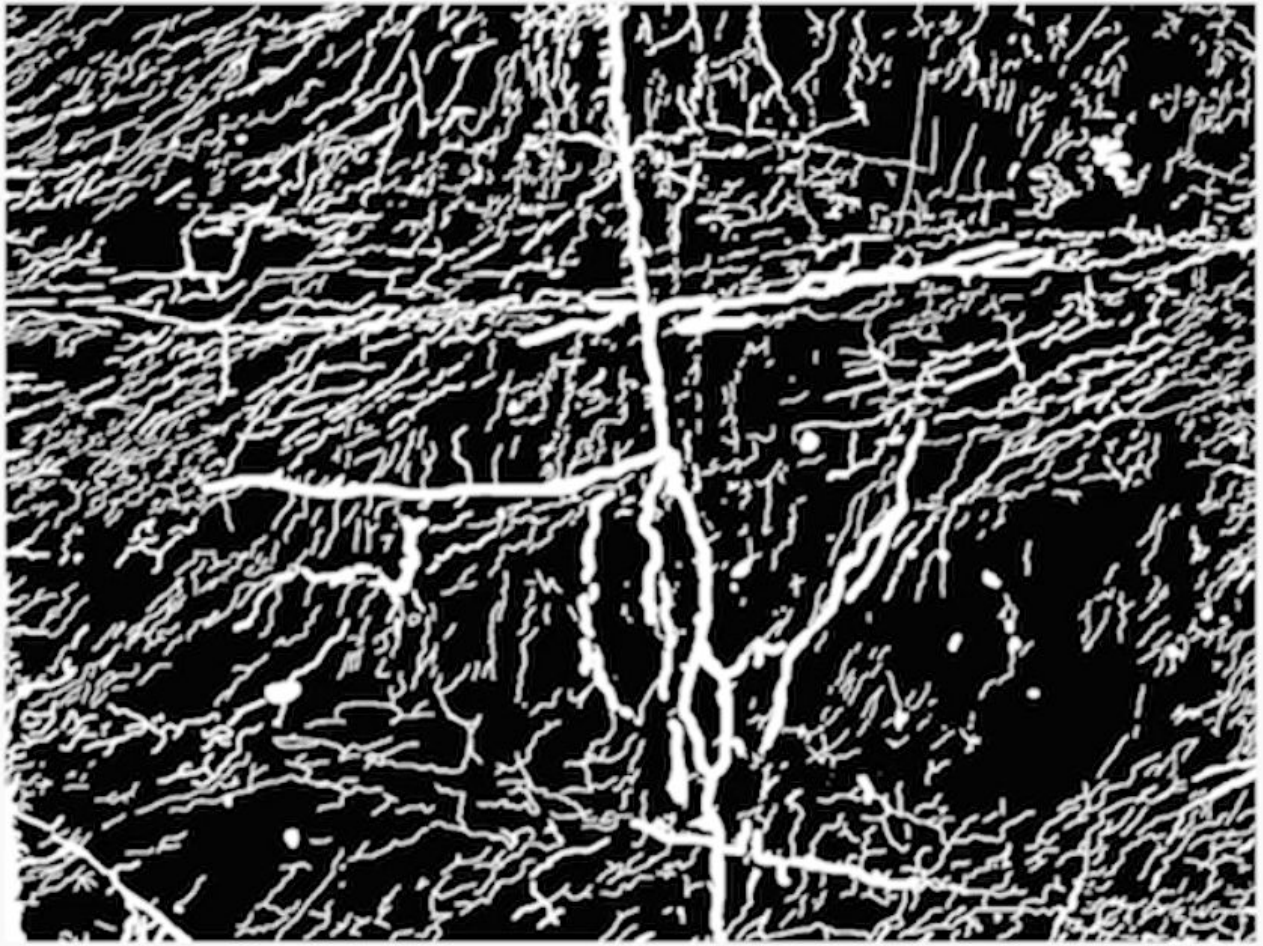
Аугментации: примеры



https://github.com/albumentations-team/albumentations

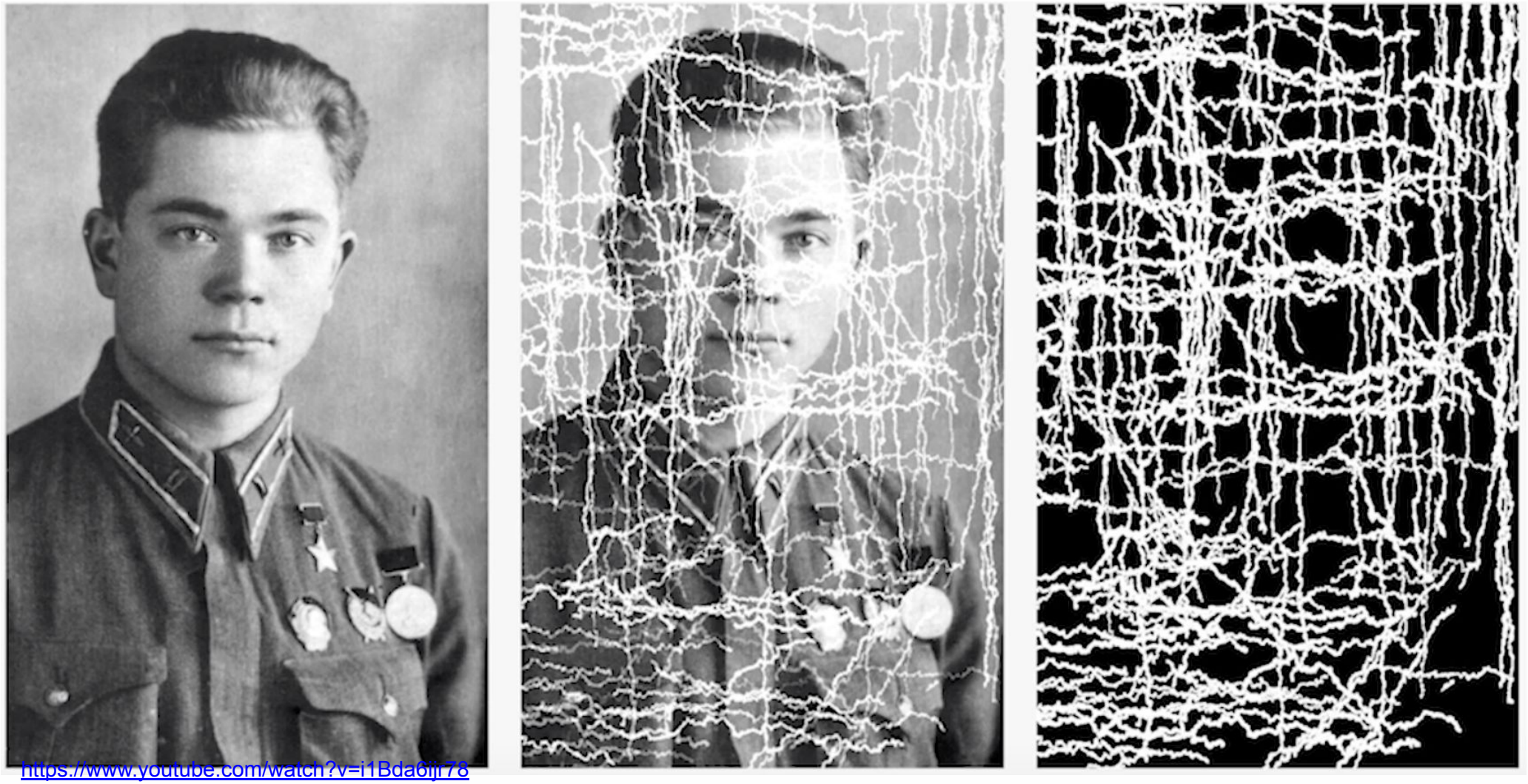
https://9may.mail.ru/restoration/

Аугментации: генераторы данных



https://www.youtube.com/watch?v=i1Bda6ijr78

Аугментации: генераторы данных



https://www.youtube.com/watch?v=i1Bda6ijr78

Аугментации: генераторы данных

https://www.youtube.com/watch?v=i1Bda6ijr78

Аугментации: генераторы данных

https://www.youtube.com/watch?v=i1Bda6ijr78

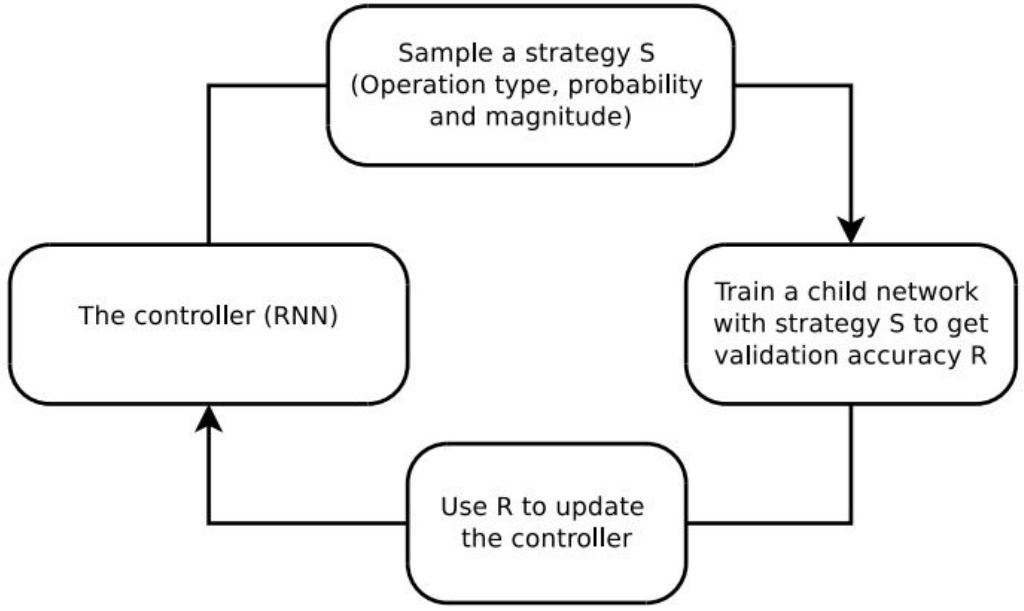
Аугментации: pytorch

```
tran = transforms.Compose([
    transforms.Resize((image_size, image_size)),
    transforms.RandomHorizontalFlip(),
    transforms.ToTensor(),
    normalize,
    cutout
])
```

Автоматические аугментации

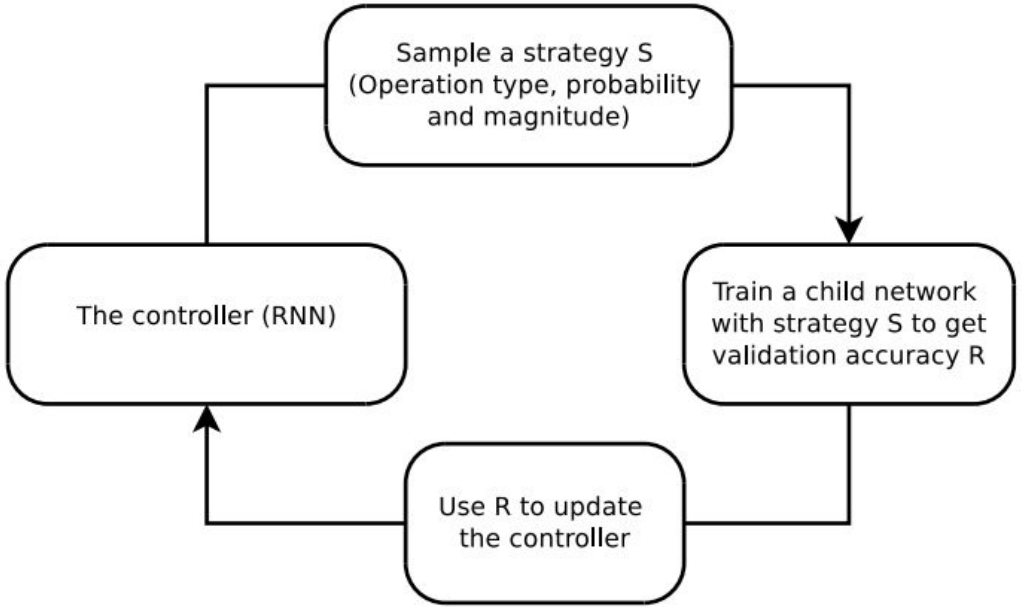
- Population based search
- AutoAugment
- RandAugment
- Ctaugment

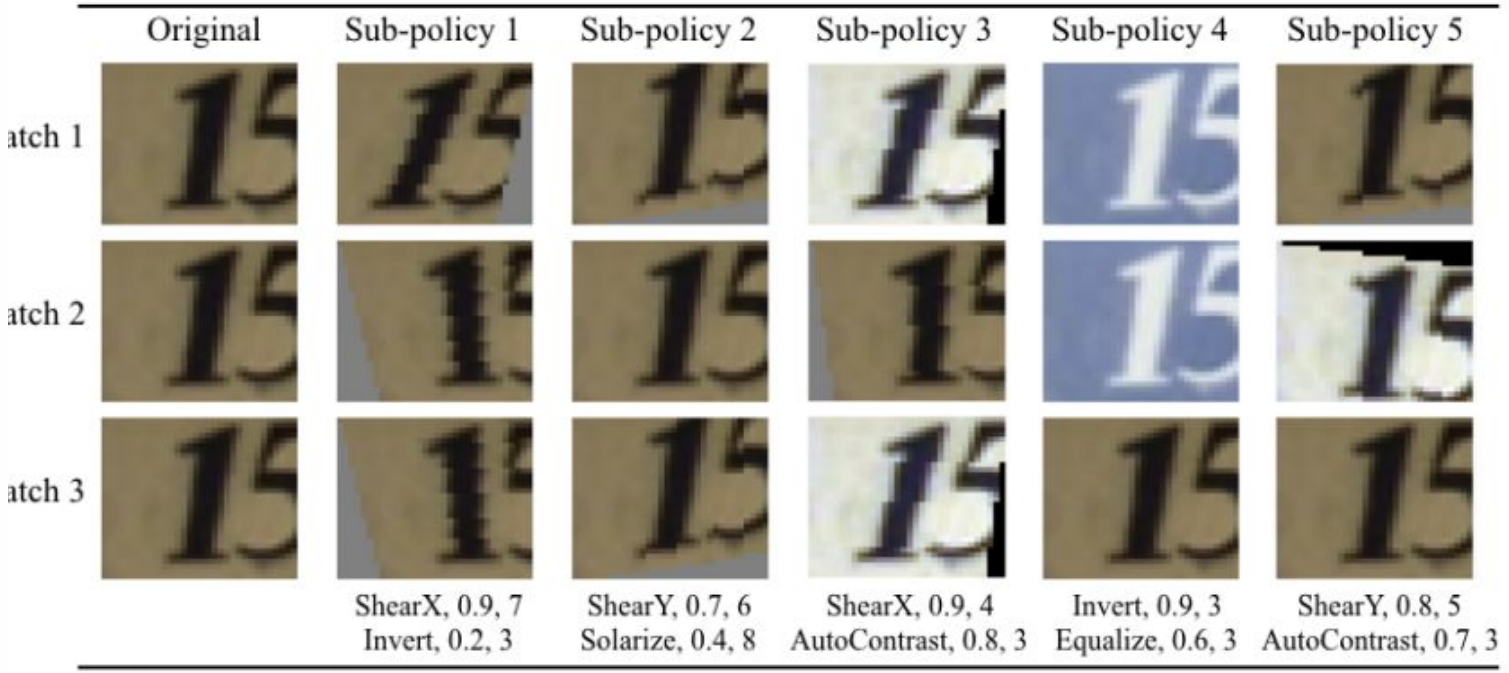
Autoaugment



https://arxiv.org/pdf/1805.09501.pdf

Autoaugment





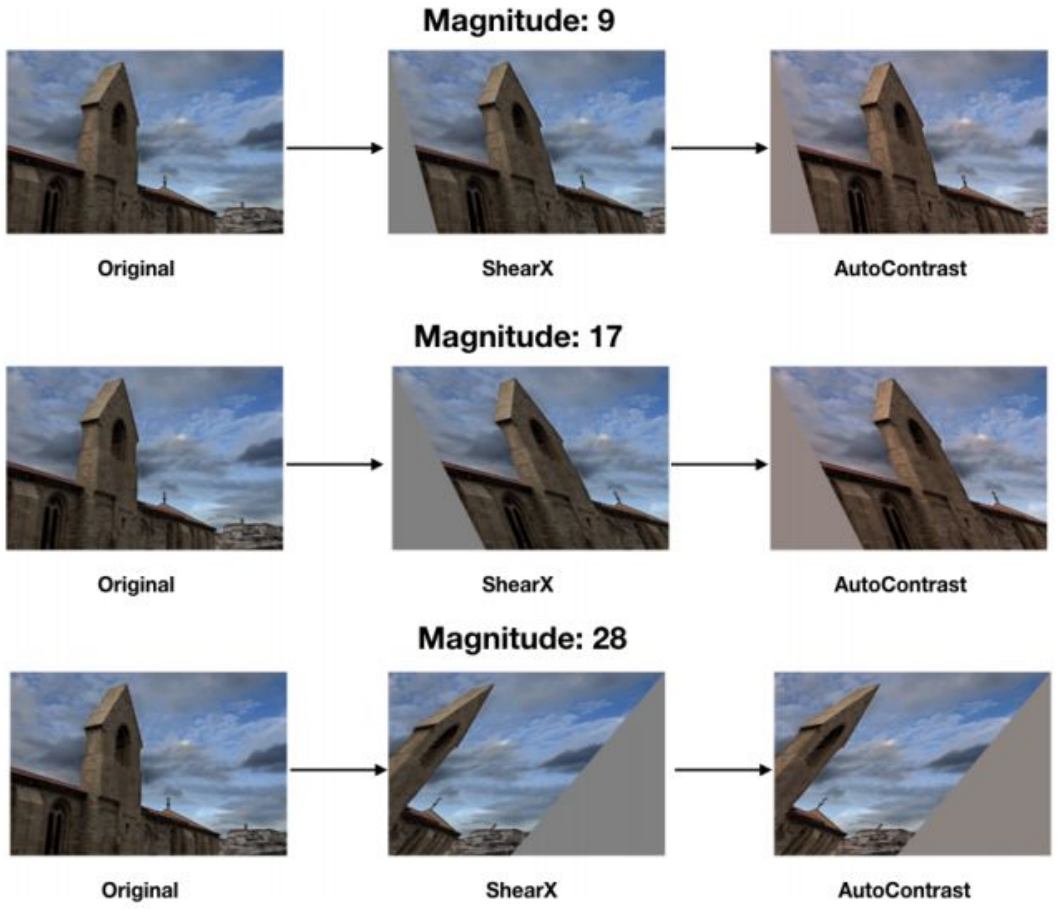
https://arxiv.org/pdf/1805.09501.pdf

Autoaugment

Model	Inception Pre-processing [59]	AutoAugment ours
ResNet-50	76.3 / 93.1	77.6 / 93.8
ResNet-200	78.5 / 94.2	80.0 / 95.0
AmoebaNet-B (6,190)	82.2 / 96.0	82.8 / 96.2
AmoebaNet-C (6,228)	83.1 / 96.1	**83.5 / 96.5**

https://arxiv.org/pdf/1805.09501.pdf

Randaugment



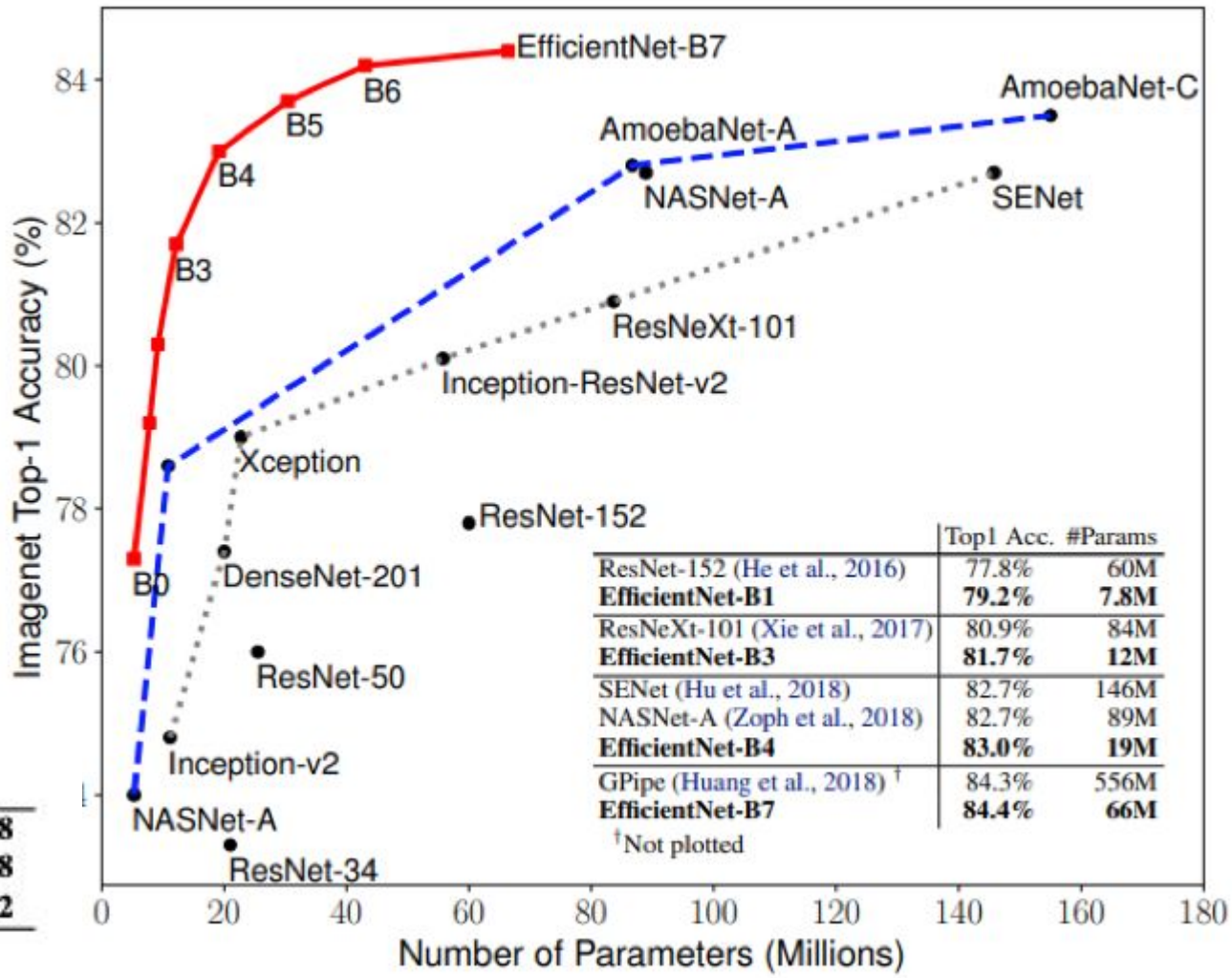

```
transforms = [
'Identity', 'AutoContrast', 'Equalize',
'Rotate', 'Solarize', 'Color', 'Posterize',
'Contrast', 'Brightness', 'Sharpness',
'ShearX', 'ShearY', 'TranslateX', 'TranslateY']

def randaugment(N, M):
"""Generate a set of distortions.

  Args:
    N: Number of augmentation transformations to
        apply sequentially.
    M: Magnitude for all the transformations.
"""

  sampled_ops = np.random.choice(transforms, N)
  return [(op, M) for op in sampled_ops]
```

Figure 2. Python code for RandAugment based on numpy.

Randaugment

	baseline	PBA	Fast AA	AA	RA
CIFAR-10					
Wide-ResNet-28-2	94.9	-	-	**95.9**	95.8
Wide-ResNet-28-10	96.1	**97.4**	97.3	**97.4**	97.3
Shake-Shake	97.1	**98.0**	**98.0**	**98.0**	**98.0**
PyramidNet	97.3	**98.5**	98.3	**98.5**	**98.5**
CIFAR-100					
Wide-ResNet-28-2	75.4	-	-	**78.5**	78.3
Wide-ResNet-28-10	81.2	**83.3**	82.7	82.9	**83.3**
SVHN (core set)					
Wide-ResNet-28-2	96.7	-	-	98.0	**98.3**
Wide-ResNet-28-10	96.9	-	-	98.1	**98.3**
SVHN					
Wide-ResNet-28-2	98.2	-	-	**98.7**	**98.7**
Wide-ResNet-28-10	98.5	98.9	98.8	98.9	**99.0**

	baseline	Fast AA	AA	RA
ResNet-50	76.3 / 93.1	**77.6** / 93.7	**77.6 / 93.8**	**77.6 / 93.8**
EfficientNet-B5	83.2 / 96.7	-	83.3 / 96.7	**83.9 / 96.8**
EfficientNet-B7	84.0 / 96.9	-	84.4 / 97.1	**85.0 / 97.2**



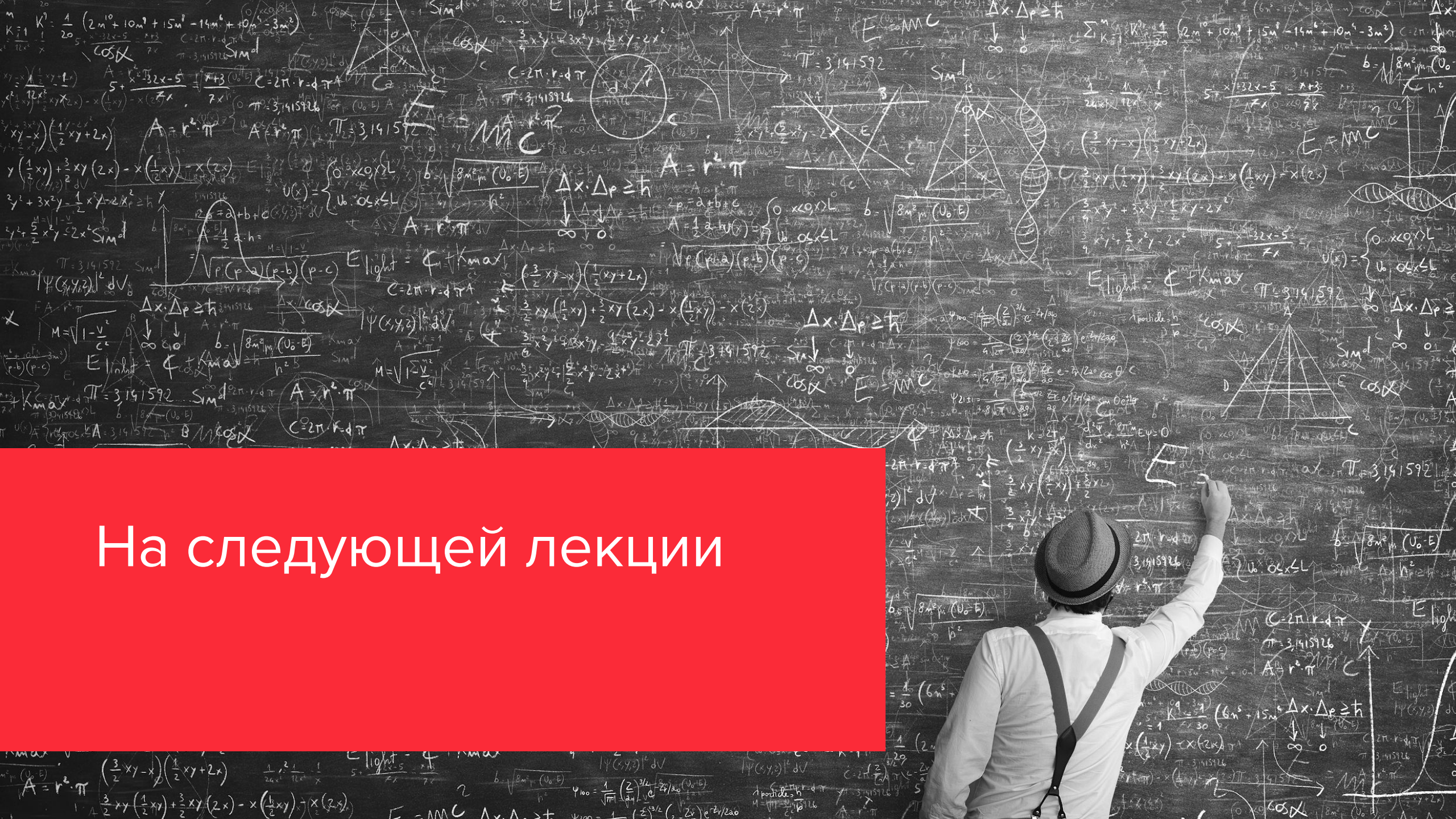
	Top1 Acc.	#Params
ResNet-152 (He et al., 2016)	77.8%	60M
EfficientNet-B1	**79.2%**	**7.8M**
ResNeXt-101 (Xie et al., 2017)	80.9%	84M
EfficientNet-B3	**81.7%**	**12M**
SENet (Hu et al., 2018)	82.7%	146M
NASNet-A (Zoph et al., 2018)	82.7%	89M
EfficientNet-B4	**83.0%**	**19M**
GPipe (Huang et al., 2018) †	84.3%	556M
EfficientNet-B7	**84.4%**	**66M**

†Not plotted

https://arxiv.org/pdf/1905.11946.pdf

Практические трюки

- Подбор learning rate для оптимизатора
- Кастомные функции потерь (arcface etc.)
- Хитрые инициализации весов и прогрев



На следующей лекции

Начало контеста

- Задача: сделать детектор ключевых точек лица

На следующей лекции

- Детекторы
  - RCNN
  - MTCNN
  - детекторы людей