ФИНАНСОВЫЙ УНИВЕРСИТЕТ
ПРИ ПРАВИТЕЛЬСТВЕ РОССИЙСКОЙ ФЕДЕРАЦИИ

Современные нейросетевые технологии

Лекция 6. Процесс обучения сетей

Вопросы

1. Препроцессинг данных и инициализация весов
2. Поиск гиперпараметров
3. Продвинутая регуляризация
4. Transfer learning

github.com/balezz/modern_dl
Срок сдачи А6 – 08.10.2022 г.

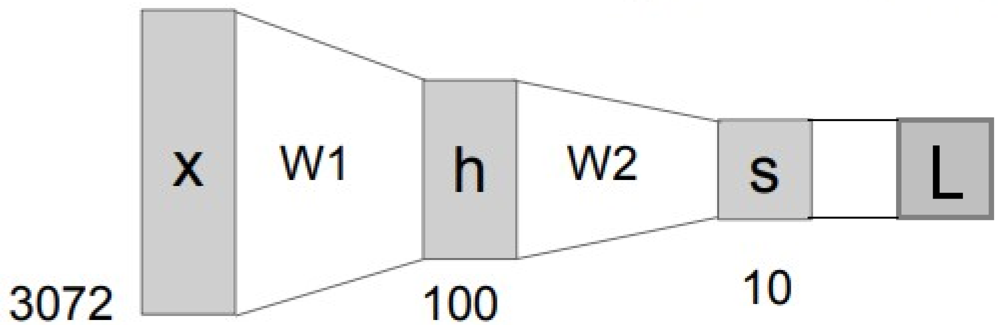
Источники:

- **dlcourse.ai**
- **cs231n.stanford.edu**
- **cs230.stanford.edu**

Neural Networks

Linear score function: $f = Wx$

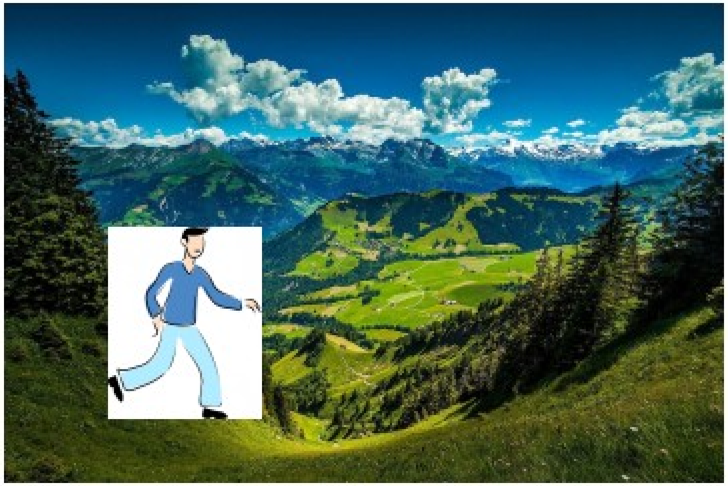
2-layer Neural Network $f = W_2 \max(0, W_1 x)$

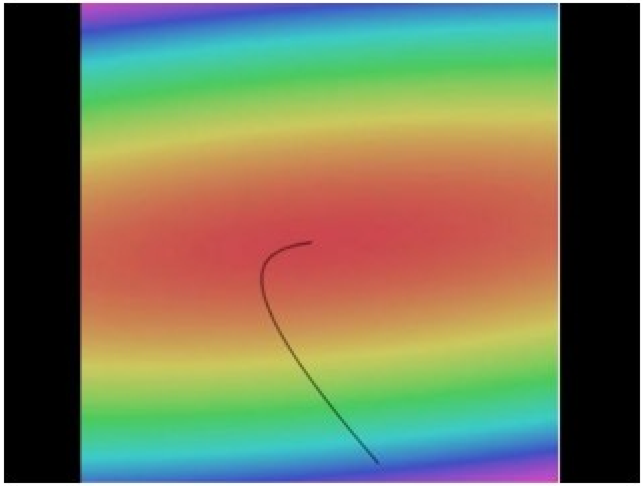


$$\frac{dL}{dx} = \frac{dh}{dx} \cdot \frac{ds}{dh} \cdot \frac{dL}{ds}$$

НА ПРОШЛОМ ЗАНЯТИИ

Learning network parameters through optimization





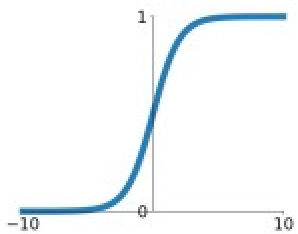
```
# Vanilla Gradient Descent

while True:
  weights_grad = evaluate_gradient(loss_fun, data, weights)
  weights += - step_size * weights_grad # perform parameter update
```

Activation Functions

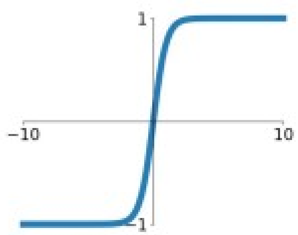
Sigmoid

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



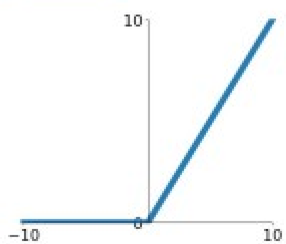
tanh

$\tanh(x)$



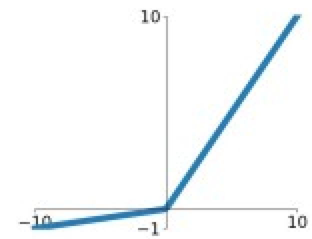
ReLU

$\max(0, x)$



Leaky ReLU

$\max(0.1x, x)$

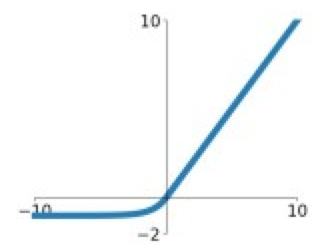


Maxout

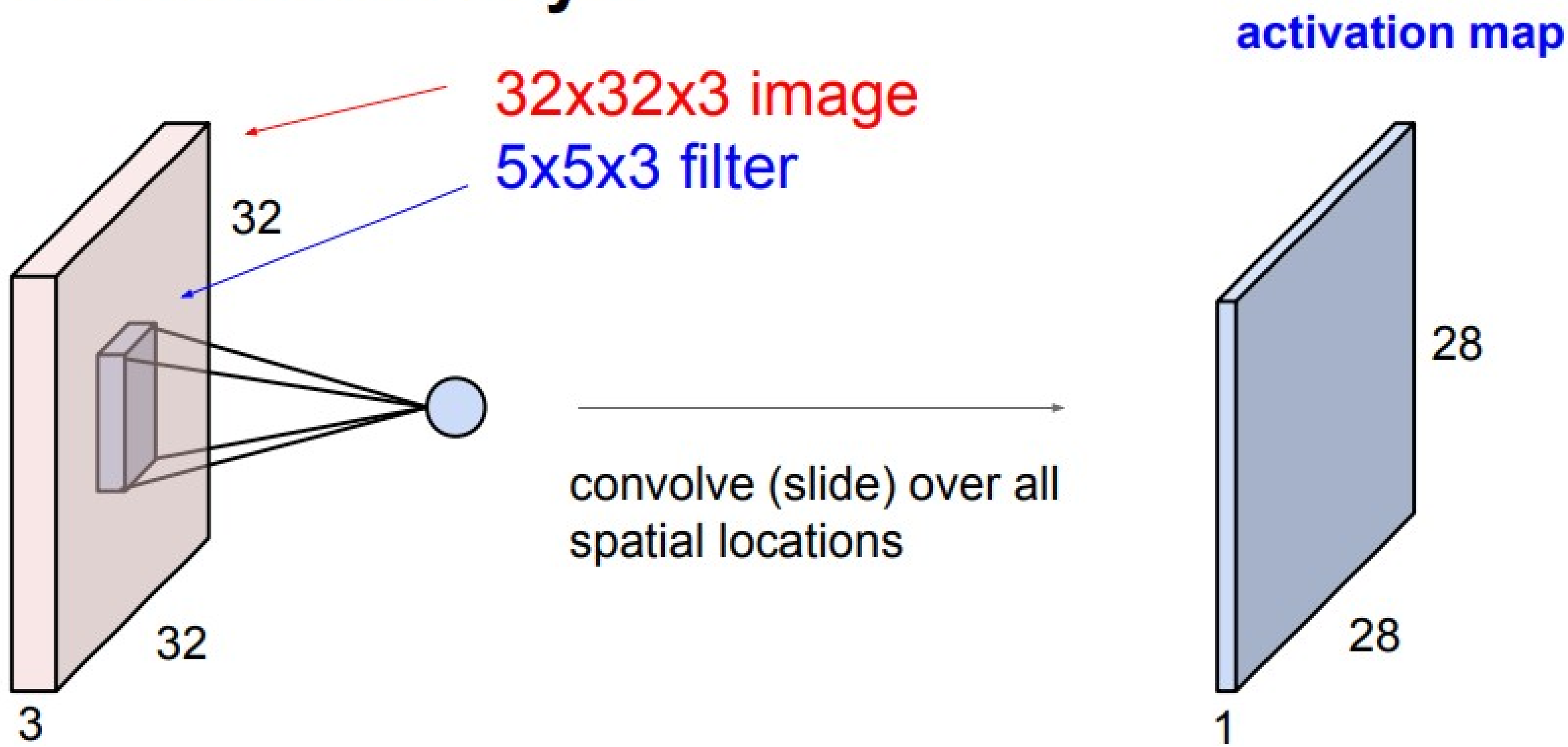
$\max(w_1^T x + b_1, w_2^T x + b_2)$

ELU

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



Convolutional Layer



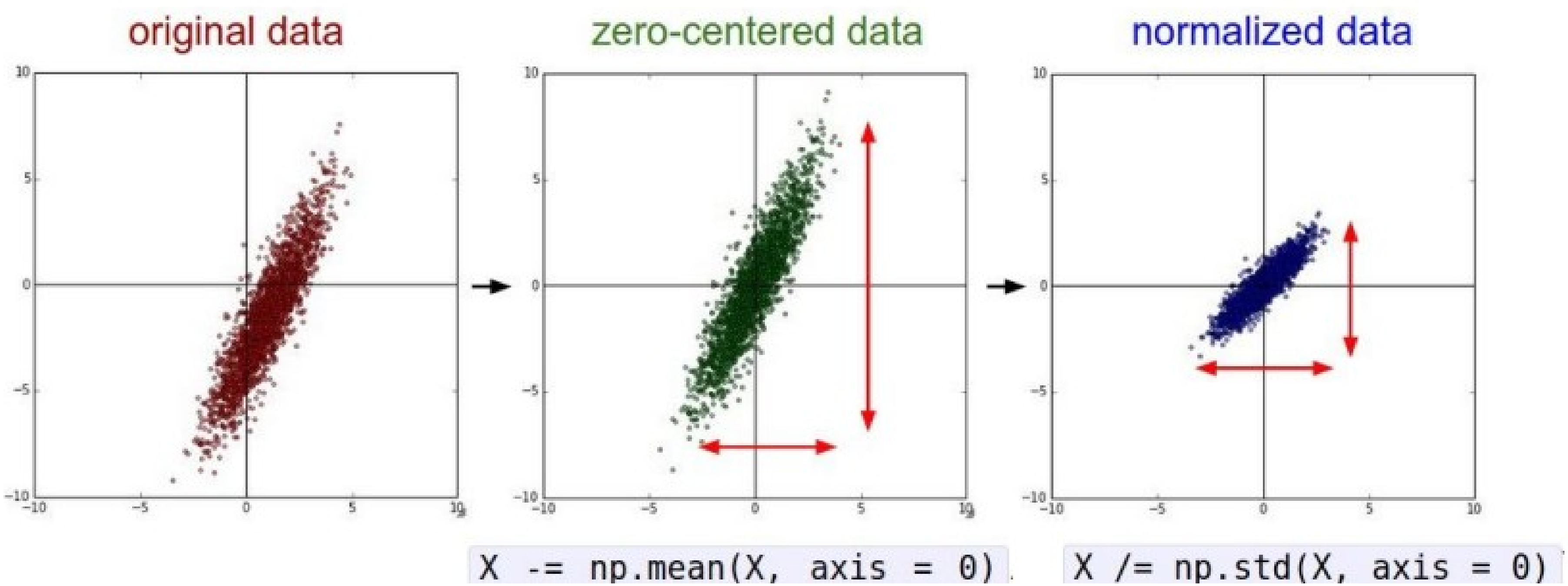
Перед процессом обучения

1. Определите препроцессинг данных.
2. Выберите простую архитектуру сети.
3. Проверьте loss на необученных весах. (e.g. 2.3 for 10 classes)
4. Проверьте, что сеть переобучается на малой выборке (20 samples per class).

```
Finished epoch 195 / 200: cost 0.002694, train: 1.000000, val 1.000000, lr 1.000000e-03
Finished epoch 196 / 200: cost 0.002674, train: 1.000000, val 1.000000, lr 1.000000e-03
Finished epoch 197 / 200: cost 0.002655, train: 1.000000, val 1.000000, lr 1.000000e-03
Finished epoch 198 / 200: cost 0.002635, train: 1.000000, val 1.000000, lr 1.000000e-03
Finished epoch 199 / 200: cost 0.002617, train: 1.000000, val 1.000000, lr 1.000000e-03
Finished epoch 200 / 200: cost 0.002597, train: 1.000000, val 1.000000, lr 1.000000e-03
finished optimization. best validation accuracy: 1.000000
```

Препроцессинг данных

tf.keras.layers.Normalization()



Batch Normalization

[Ioffe and Szegedy, 2015]

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

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

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

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

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

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

Note: at test time BatchNorm layer functions differently:

The mean/std are not computed based on the batch. Instead, a single fixed empirical mean of activations during training is used.

(e.g. can be estimated during training with running averages)

- First idea: **Small random numbers**

(gaussian with zero mean and 1e-2 standard deviation)

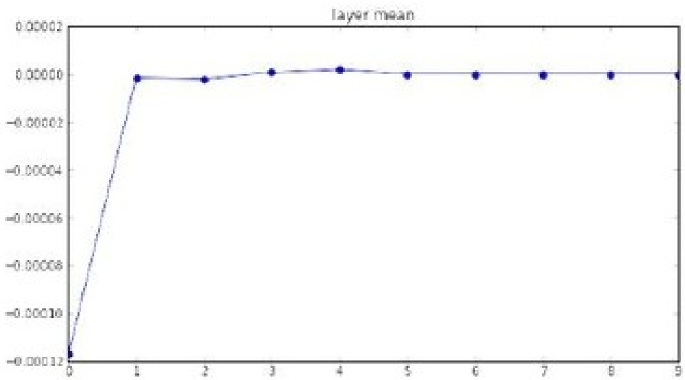
```
W = 0.01* np.random.randn(D,H)
```

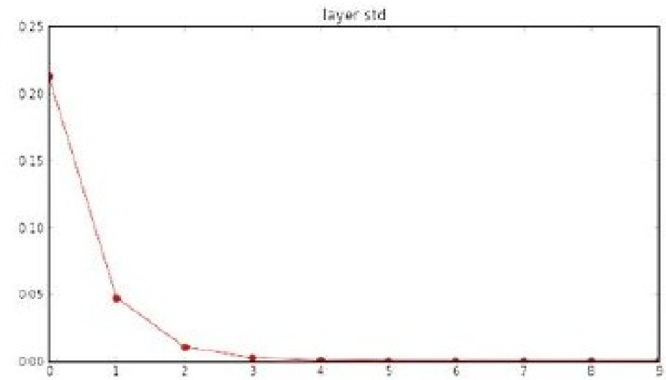
Works ~okay for small networks, but problems with deeper networks.

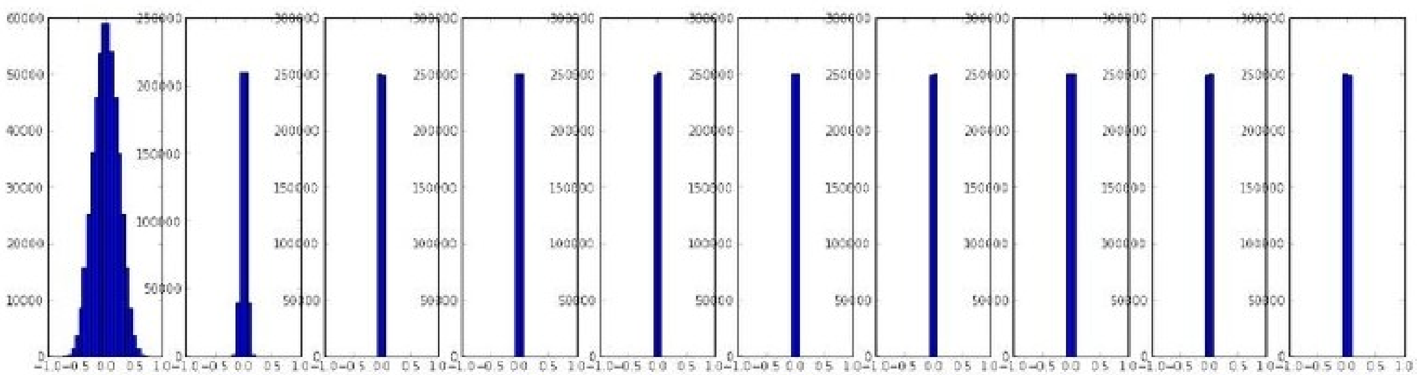
Инициализация весов

```
input layer had mean 0.000927 and std 0.998388
hidden layer 1 had mean -0.000117 and std 0.213081
hidden layer 2 had mean -0.000001 and std 0.047551
hidden layer 3 had mean -0.000002 and std 0.010630
hidden layer 4 had mean 0.000001 and std 0.002378
hidden layer 5 had mean 0.000002 and std 0.000532
hidden layer 6 had mean -0.000000 and std 0.000119
hidden layer 7 had mean 0.000000 and std 0.000026
hidden layer 8 had mean -0.000000 and std 0.000006
hidden layer 9 had mean 0.000000 and std 0.000001
hidden layer 10 had mean -0.000000 and std 0.000000
```

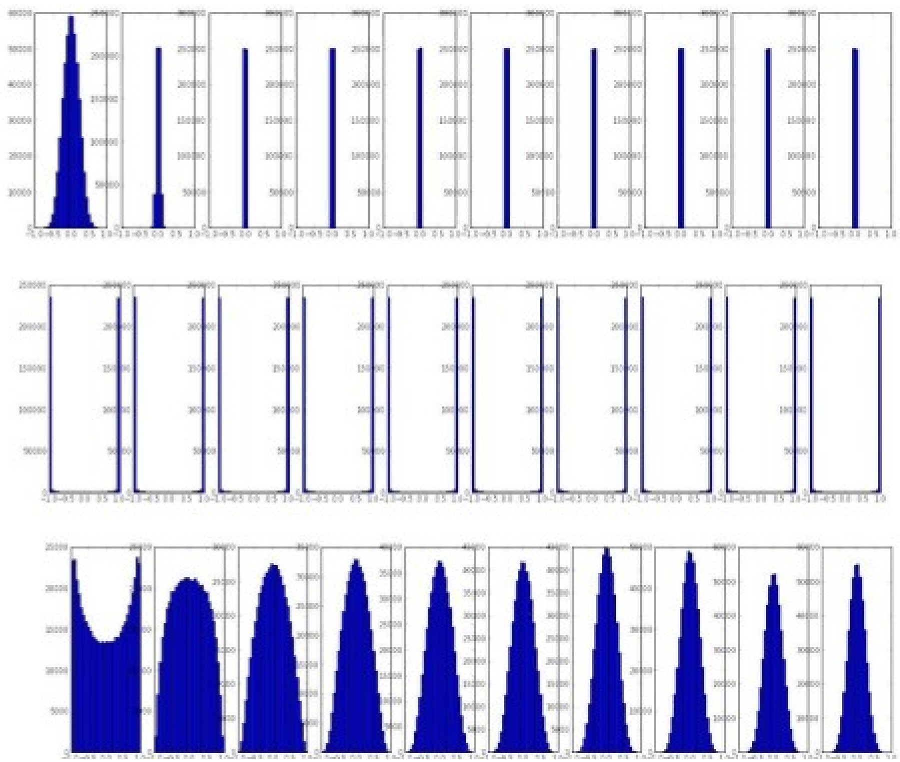
Forward pass:
 layers outputs becomes to zeros
Backward pass:
 gradients are zeros too







Инициализация весов



Initialization too small:
Activations go to zero, gradients also zero,
No learning

Initialization too big:
Activations saturate (for tanh),
Gradients zero, no learning

Initialization just right:
Nice distribution of activations at all layers,
Learning proceeds nicely

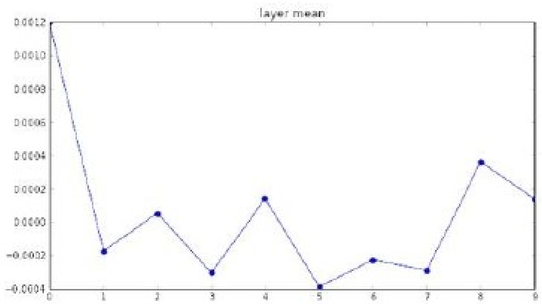
Инициализация весов

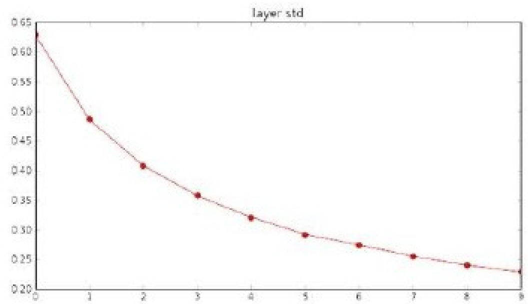
This works with tanh activations

```
input layer had mean 0.001800 and std 1.001311
hidden layer 1 had mean 0.001198 and std 0.627953
hidden layer 2 had mean -0.000175 and std 0.486051
hidden layer 3 had mean 0.000055 and std 0.407723
hidden layer 4 had mean -0.000306 and std 0.357108
hidden layer 5 had mean 0.000142 and std 0.320917
hidden layer 6 had mean -0.000389 and std 0.292116
hidden layer 7 had mean -0.000228 and std 0.273387
hidden layer 8 had mean -0.000291 and std 0.254935
hidden layer 9 had mean 0.000361 and std 0.239266
hidden layer 10 had mean 0.000139 and std 0.228008
```

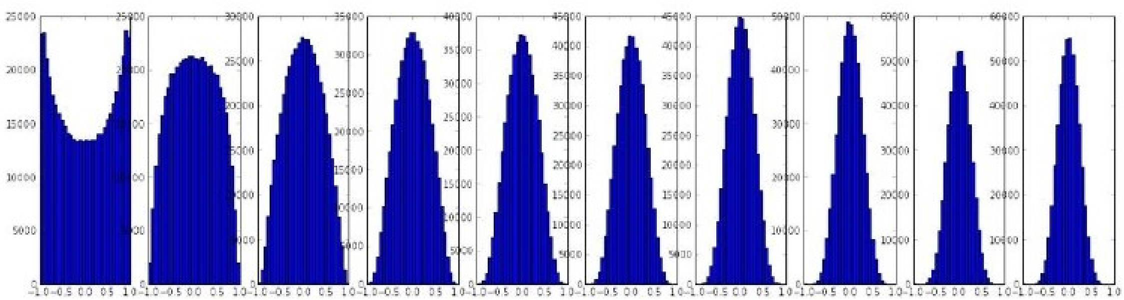
```
W = np.random.randn(fan_in, fan_out) / np.sqrt(fan_in) # layer initialization
```

"Xavier initialization"
[Glorot et al., 2010]





Reasonable initialization.
(Mathematical derivation assumes linear activations)



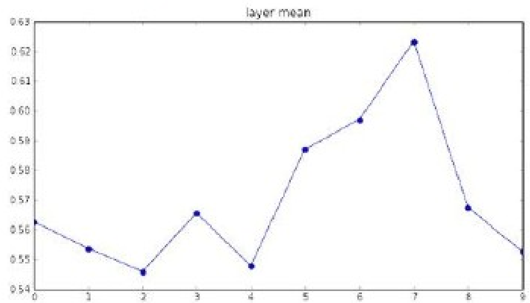
Инициализация весов

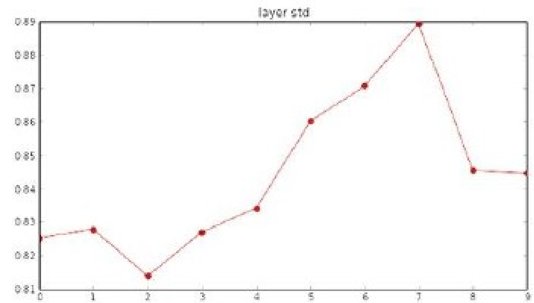
This works with ReLU activations

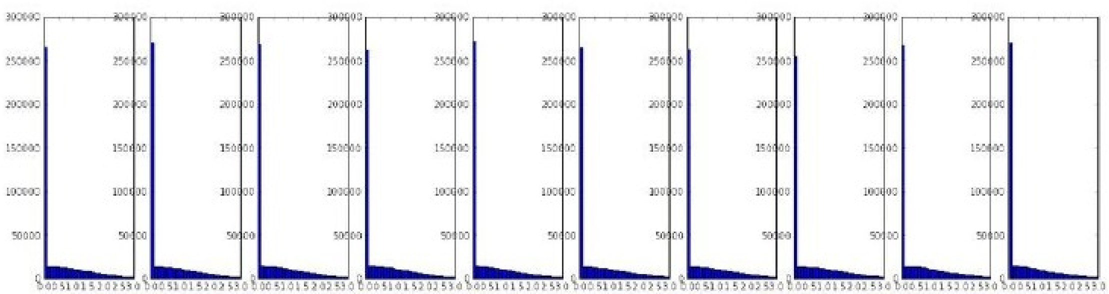
```
input layer had mean 0.000501 and std 0.999444
hidden layer 1 had mean 0.562488 and std 0.825232
hidden layer 2 had mean 0.553614 and std 0.827835
hidden layer 3 had mean 0.545867 and std 0.813855
hidden layer 4 had mean 0.565396 and std 0.826902
hidden layer 5 had mean 0.547678 and std 0.834092
hidden layer 6 had mean 0.587103 and std 0.860035
hidden layer 7 had mean 0.596867 and std 0.870610
hidden layer 8 had mean 0.623214 and std 0.889348
hidden layer 9 had mean 0.567498 and std 0.845357
hidden layer 10 had mean 0.552531 and std 0.844523
```

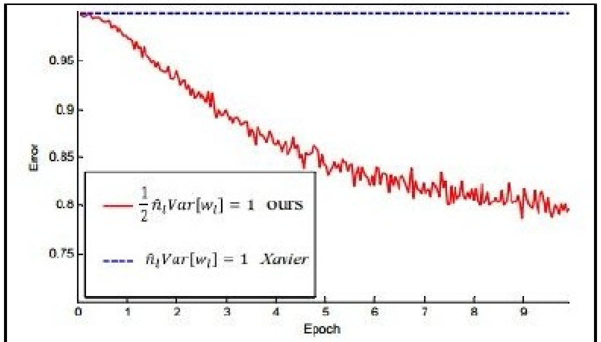
```
W = np.random.randn(fan_in, fan_out) / np.sqrt(fan_in/2) # layer initialization
```

He et al., 2015
(note additional /2)

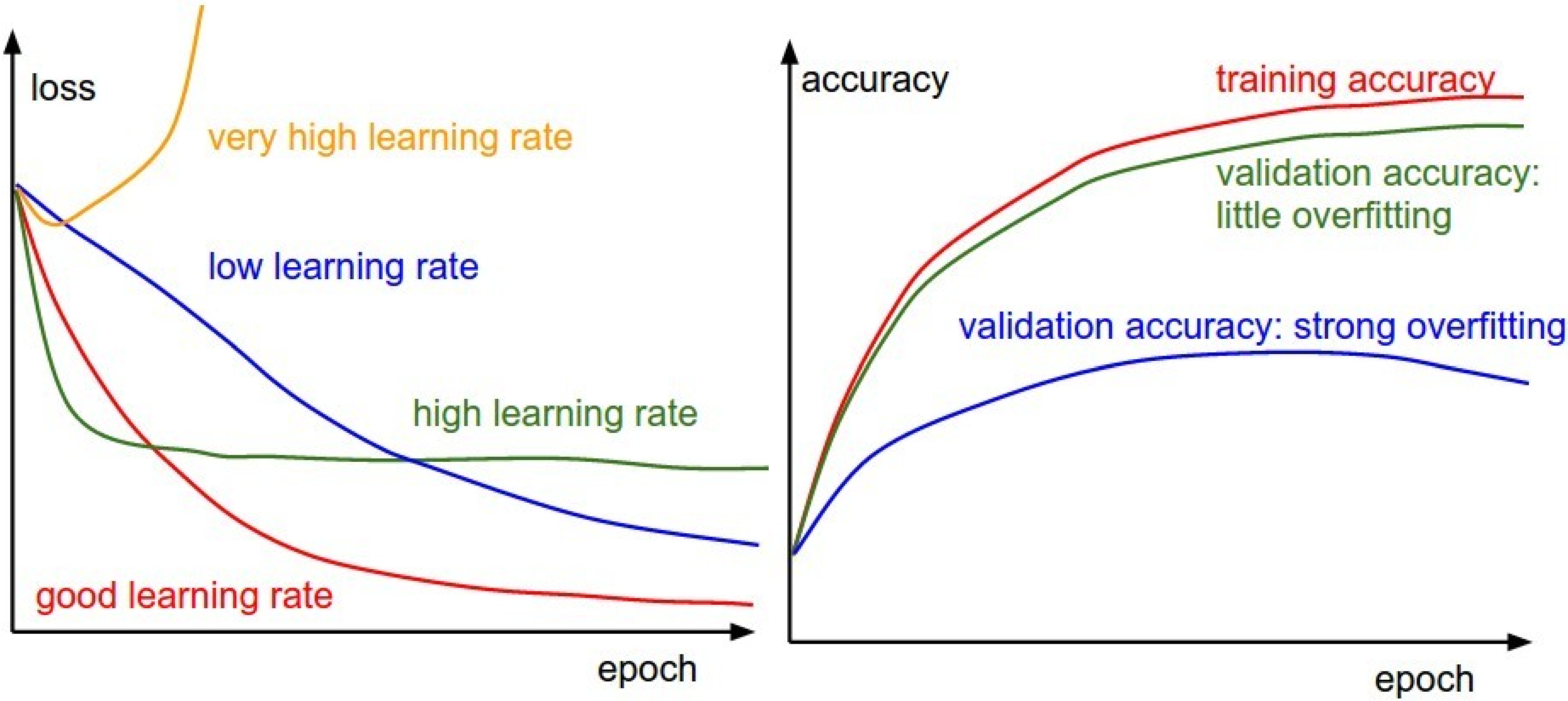








ПРОЦЕСС ОБУЧЕНИЯ



Поиск оптимальных гиперпараметров

Random Search vs. Grid Search

Random Search for Hyper-Parameter Optimization
Bergstra and Bengio, 2012

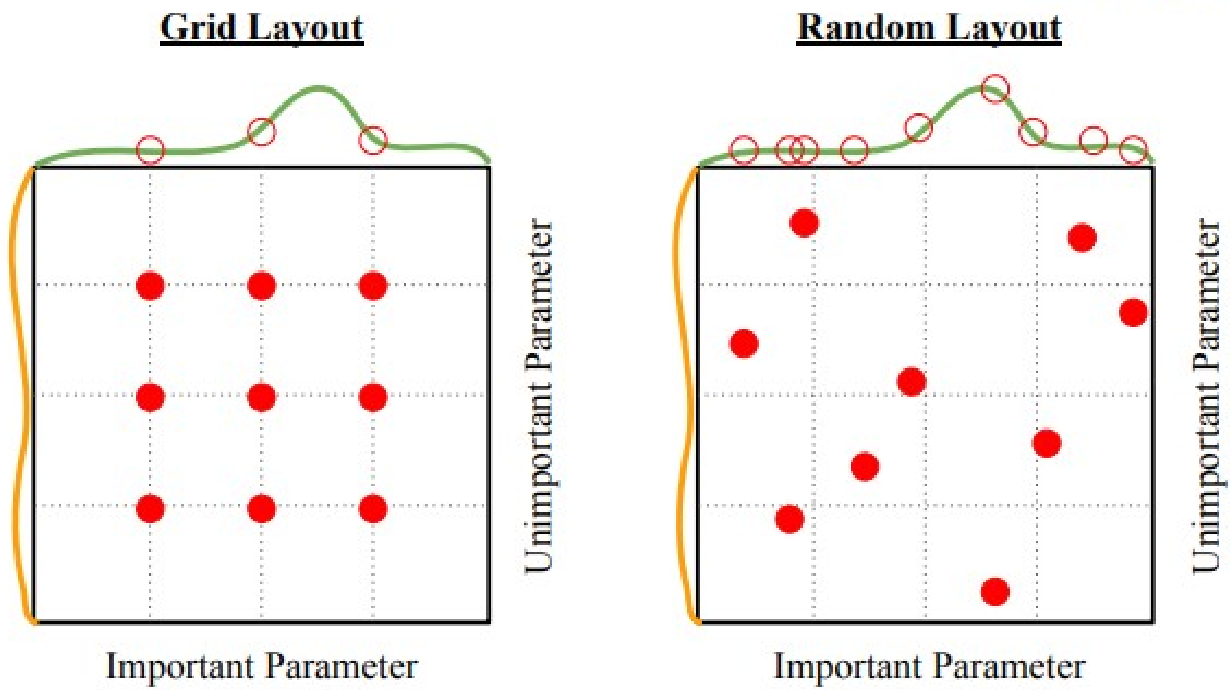
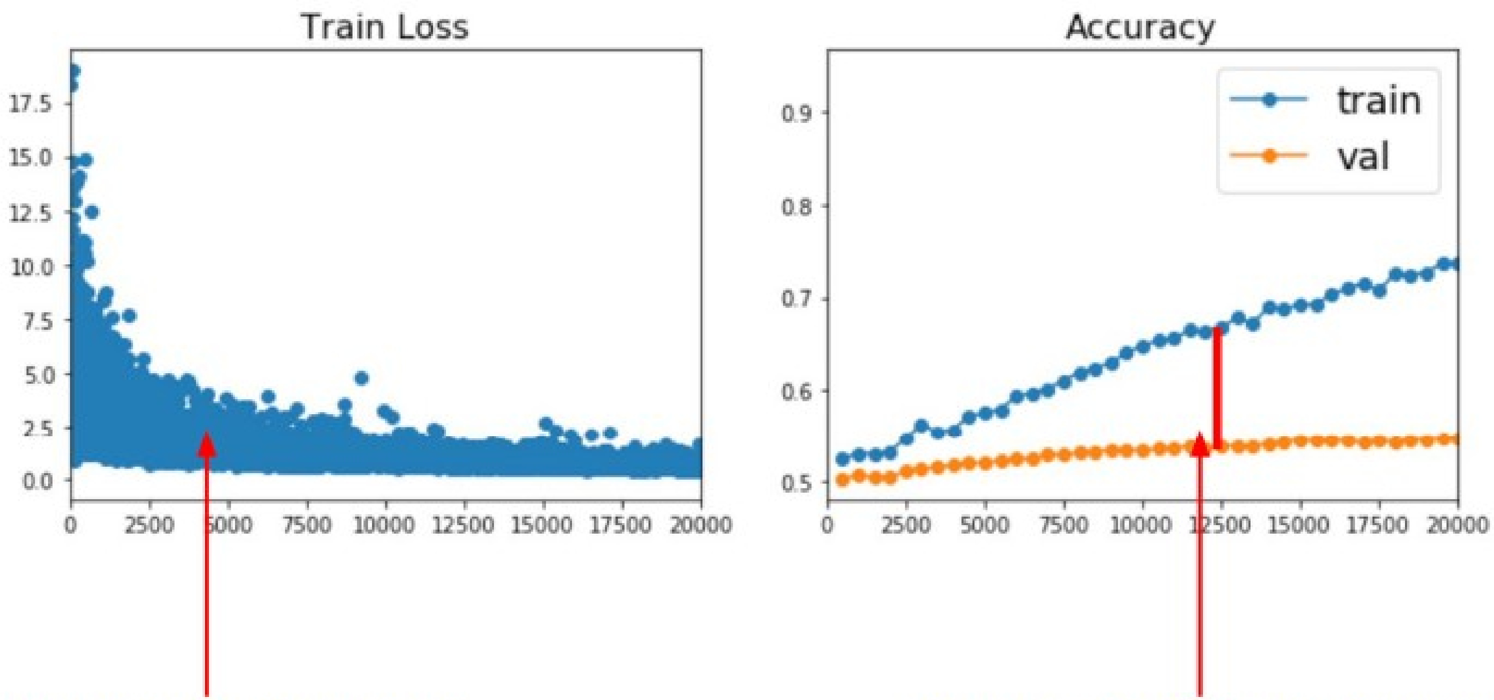


Illustration of Bergstra et al., 2012 by Shayne Longpre, copyright CS231n 2017

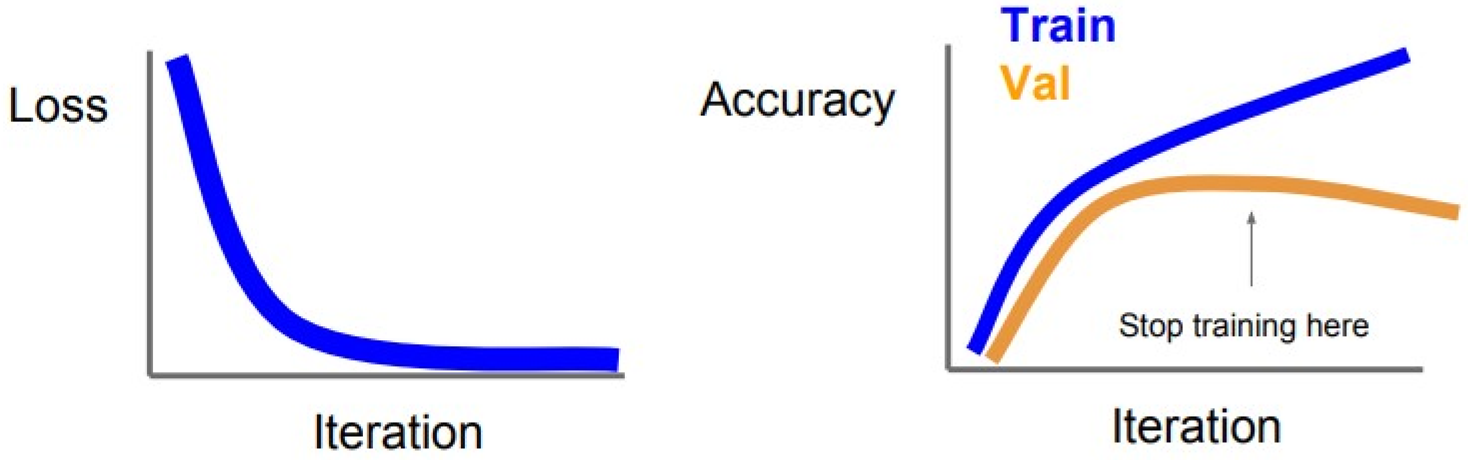
Beyond Training Error



Better optimization algorithms help reduce training loss

But we really care about error on new data - how to reduce the gap?

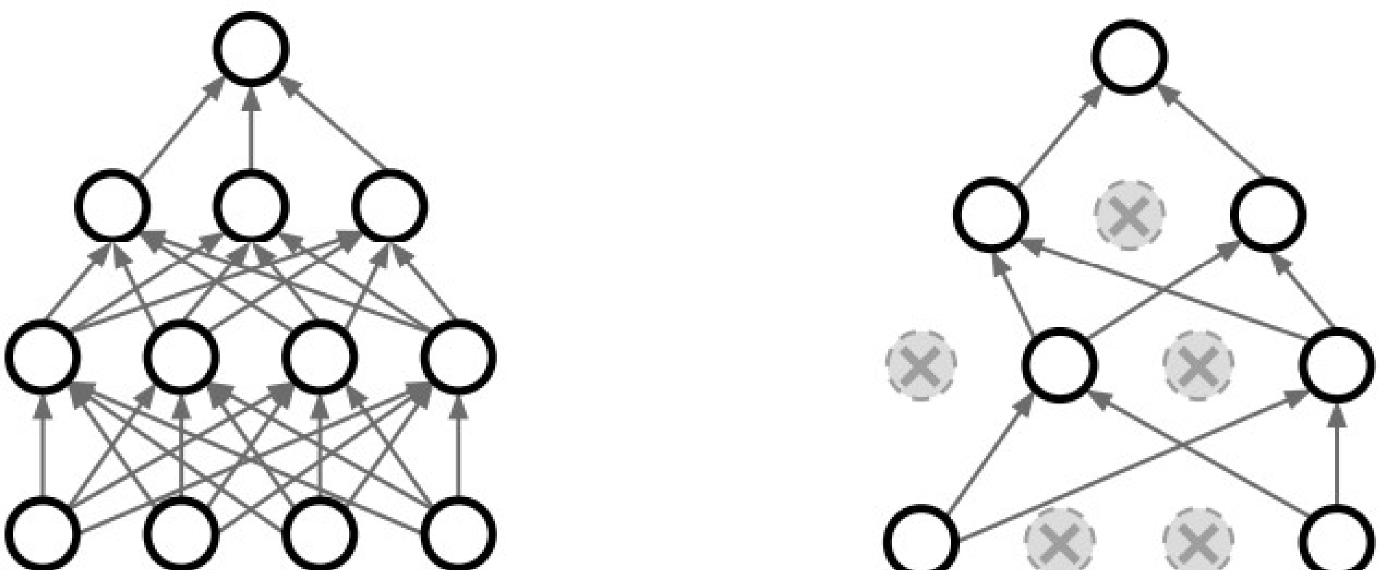
Early Stopping



Stop training the model when accuracy on the validation set decreases
Or train for a long time, but always keep track of the model snapshot that worked best on val

Regularization: Dropout

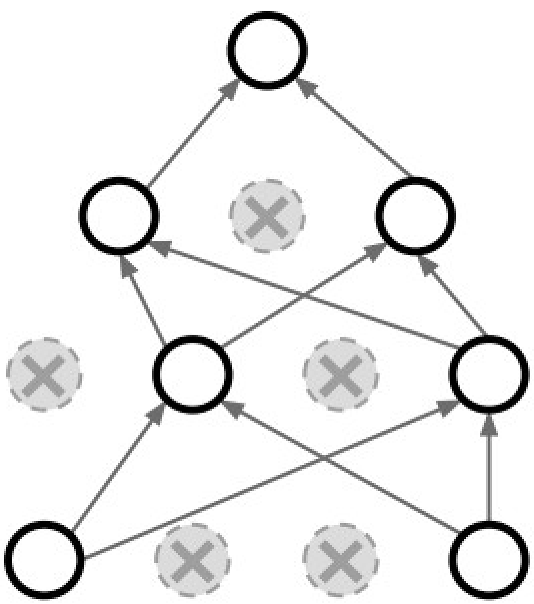
In each forward pass, randomly set some neurons to zero
Probability of dropping is a hyperparameter; 0.5 is common



Srivastava et al, "Dropout: A simple way to prevent neural networks from overfitting", JMLR 2014

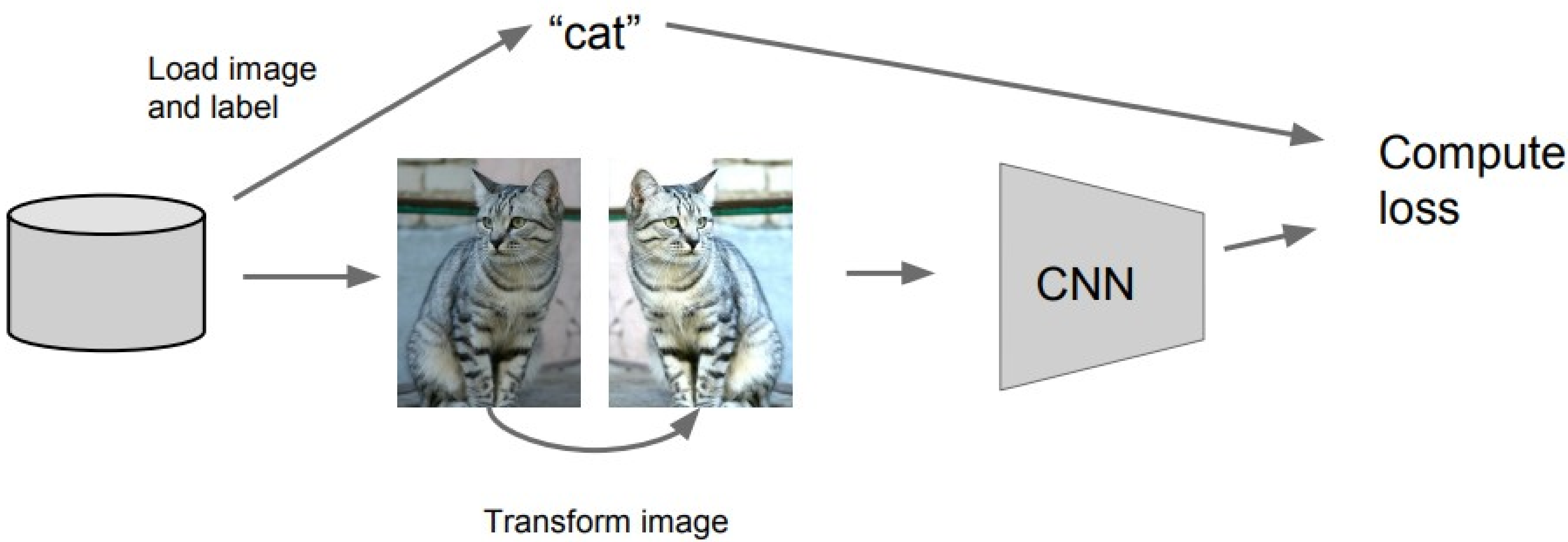
Regularization: Dropout

How can this possibly be a good idea?



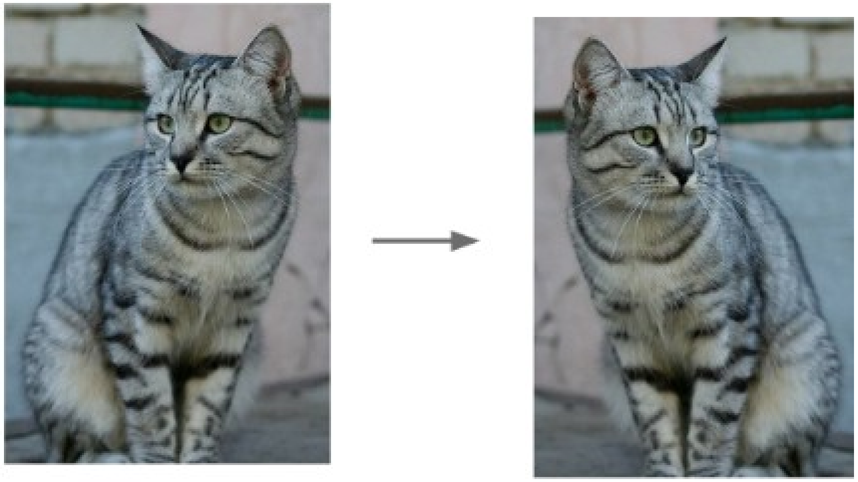
Forces the network to have a redundant representation;
Prevents co-adaptation of features

Regularization: Data Augmentation



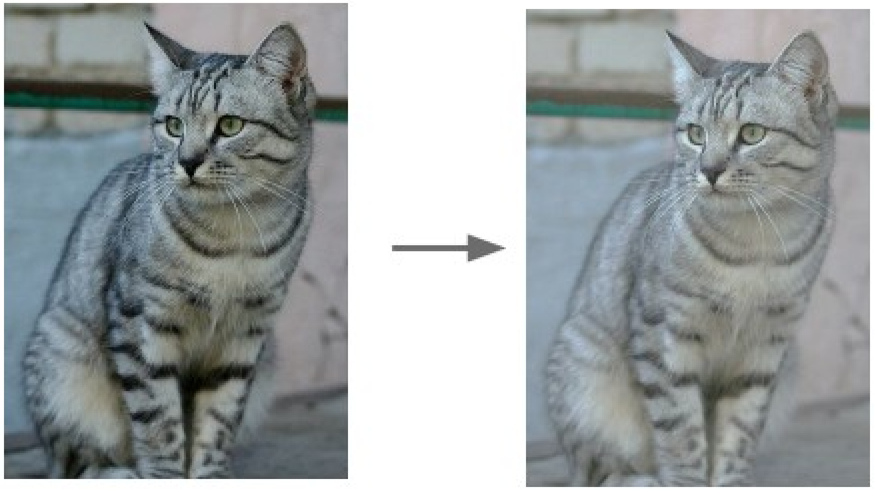
Data Augmentation

Horizontal Flip



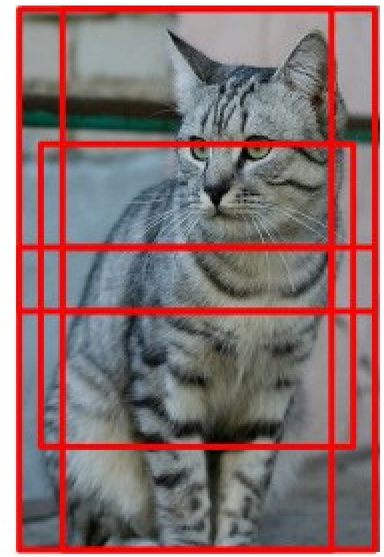
Color Jitter

Simple: Randomize
contrast and brightness



Data Augmentation

Random crops and scales



Training: sample random crops / scales

ResNet:

1. Pick random L in range [256, 480]
2. Resize training image, short side = L
3. Sample random 224 x 224 patch

Testing: average a fixed set of crops

ResNet:

1. Resize image at 5 scales: {224, 256, 384, 480, 640}
2. For each size, use 10 224 x 224 crops: 4 corners + center, + flips

Transfer learning