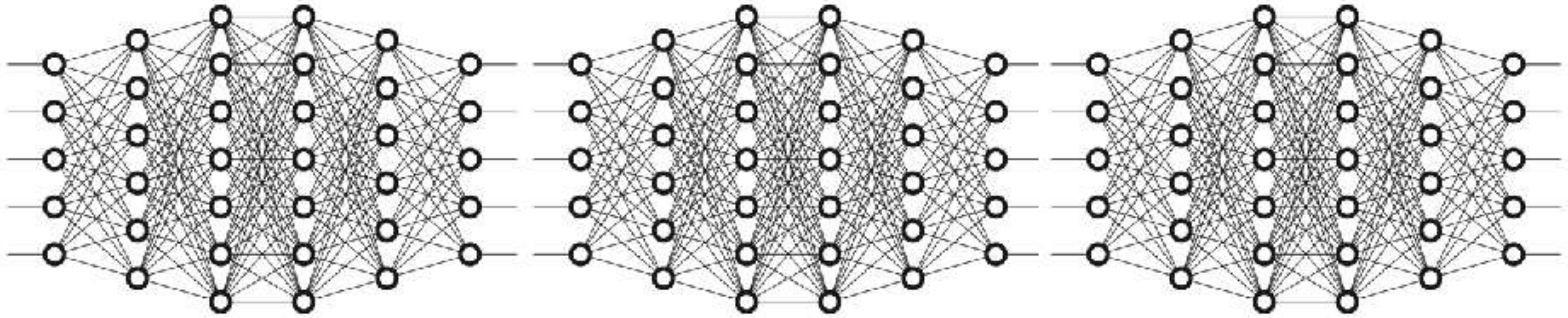
THE UNIVERSITY OF BRITISH COLUMBIA

CPSC 425: Computer Vision



Lecture 22: Neural Networks 3

Menu for Today

Topics:

— **Neural Networks** part 3

— Weight **Initialization**

— **Normalization**

— Preventing **Overfitting**

Readings:

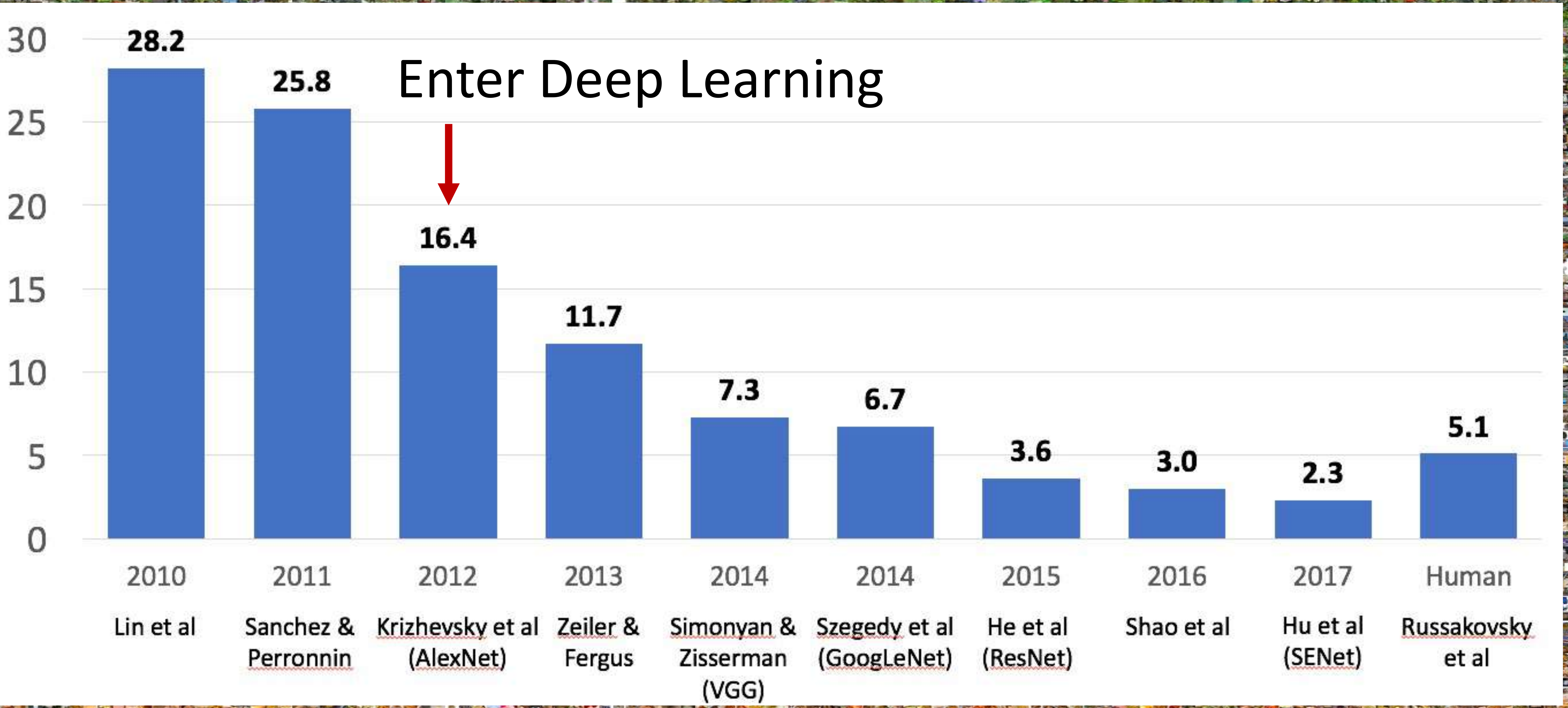
— **Today's** Lecture: Szeliski 5.1.3, 5.3-5.4, Justin Johnson Michigan EECS 498/598

Reminders:

— **Quiz 6** April 7th

— **Assignment 6**: due Apr 10th <— watch out!

IMAGENET Large Scale Visual Recognition Challenge



1959
Hubel & Wiesel

1963
Roberts

1970s
David Marr

1979
Gen. Cylinders

1986
Canny

AI Winter

1997
Norm. Cuts

1999
SIFT

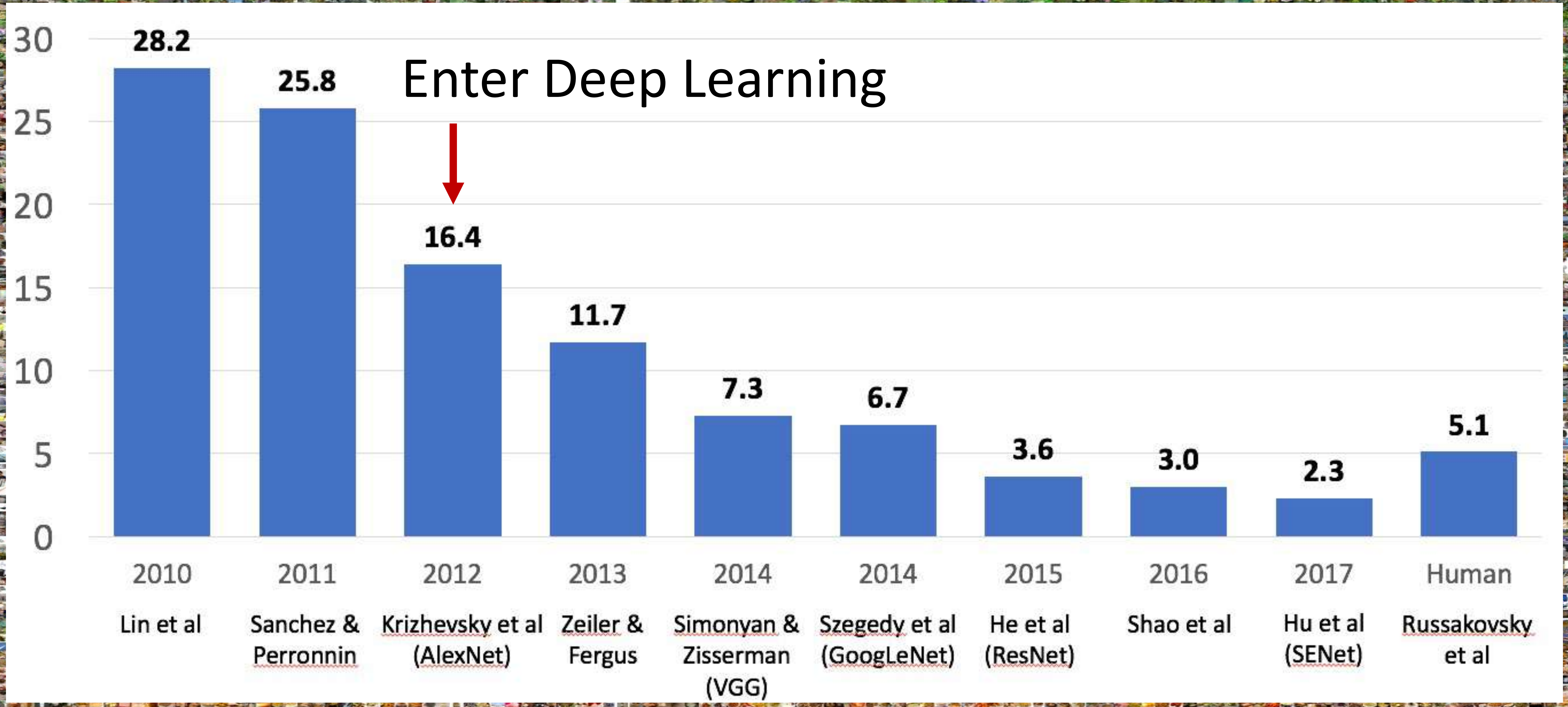
2001
V&J

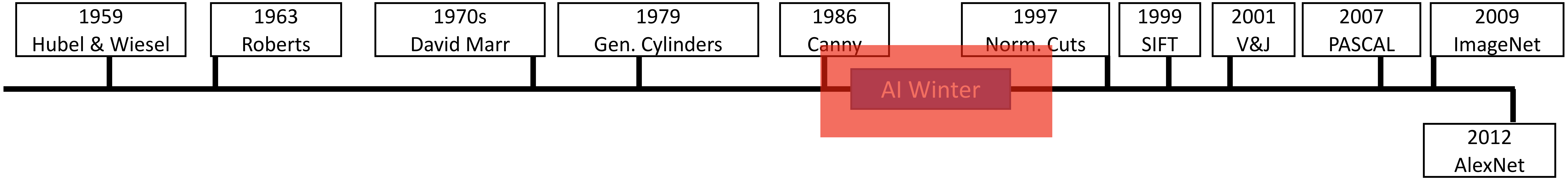
2007
PASCAL

2009
ImageNet

2012
AlexNet

IMAGENET Large Scale Visual Recognition Challenge





So why now?

Rise of large datasets

IMAGENET

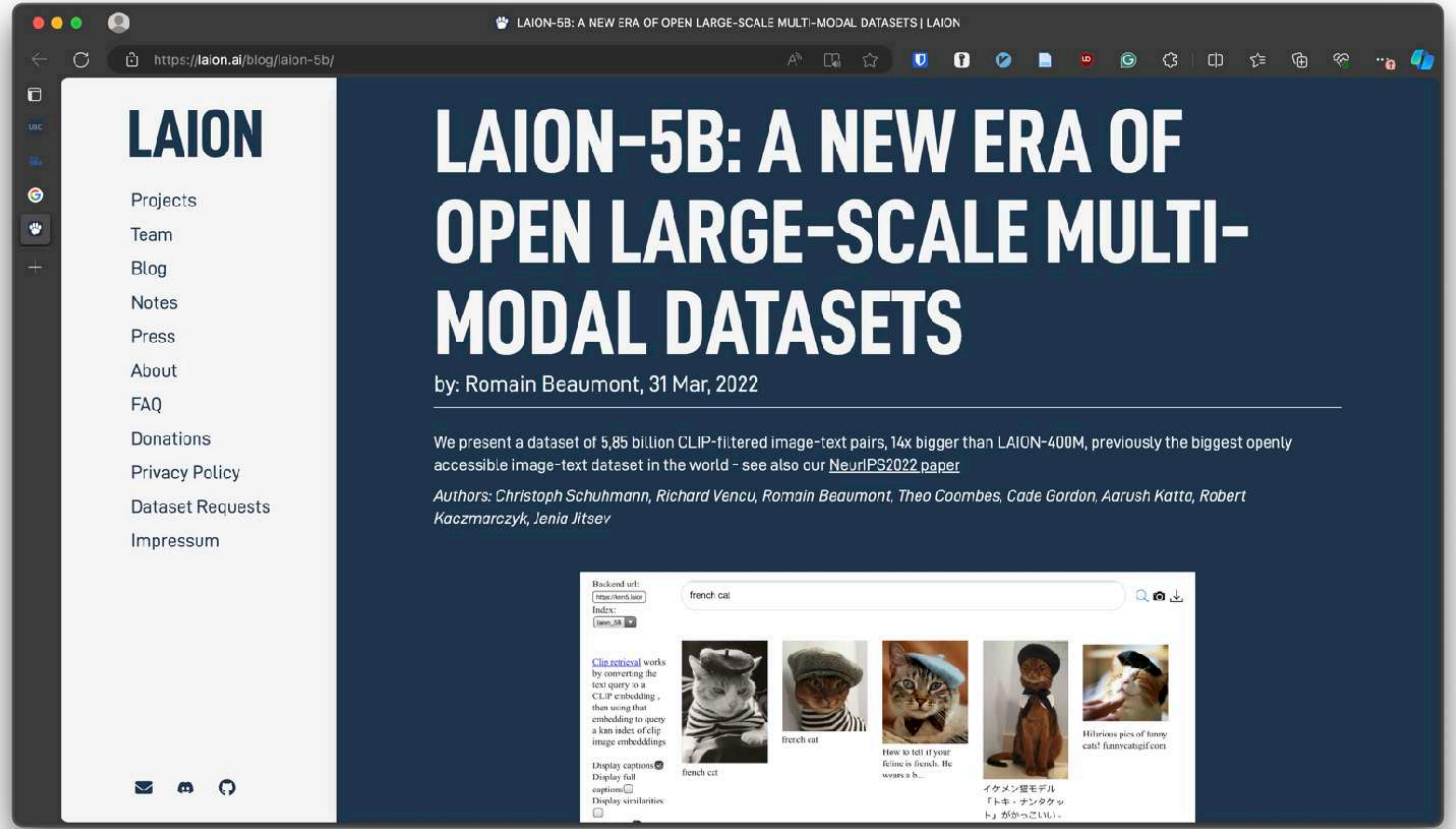
www.image-net.org

22K categories and **14M** images

- Animals
  - Bird
  - Fish
  - Mammal
  - Invertebrate
- Plants
  - Tree
  - Flower
- Food
- Materials
- Structures
- Artifact
  - Tools
  - Appliances
  - Structures
- Person
- Scenes
  - Indoor
  - Geological Formations
- Sport Activities

Deng, Dong, Socher, Li, Li, & Fei-Fei, 2009

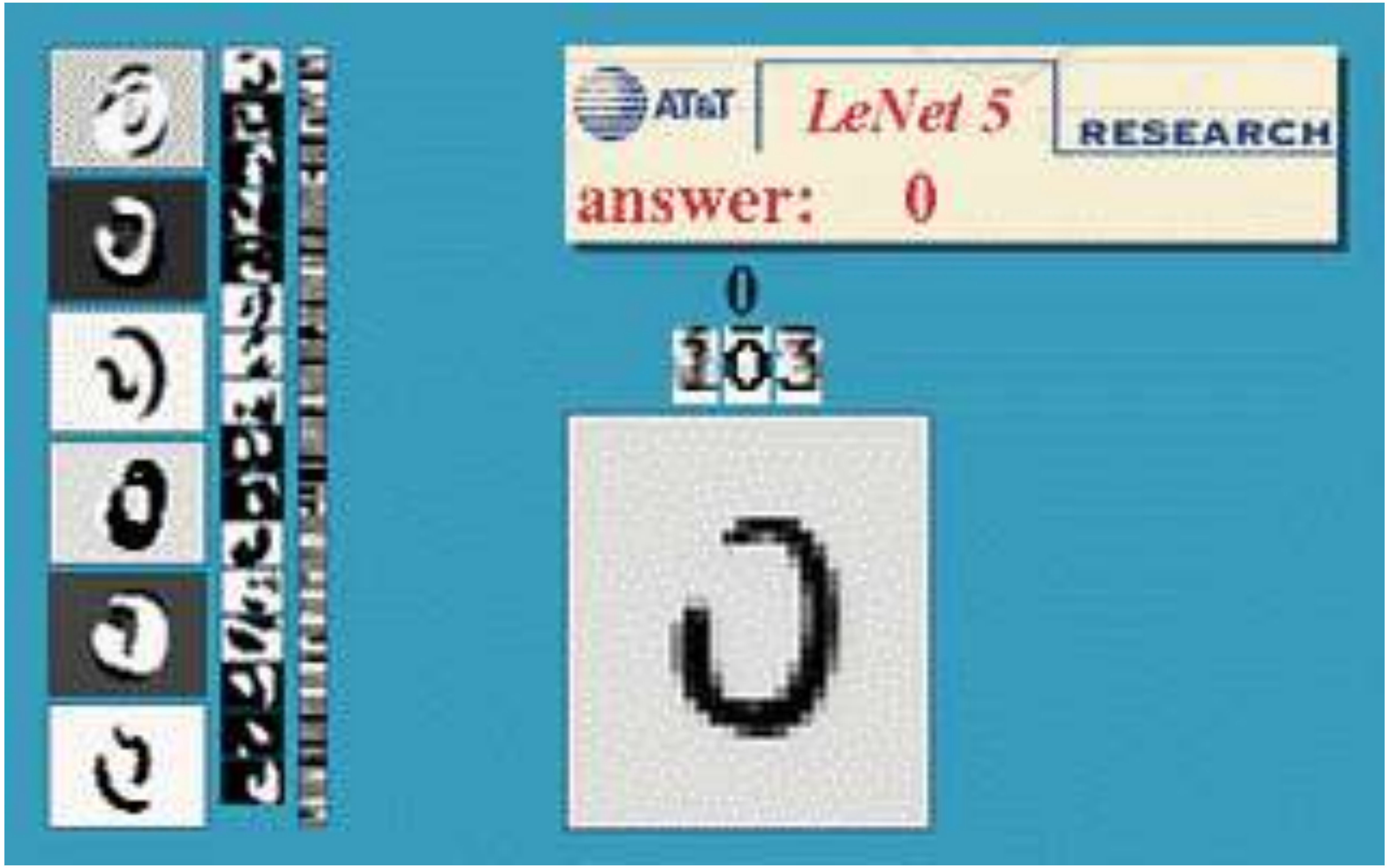
Rise of large datasets



https://laion.ai/blog/laion-5b/

Clever architectures

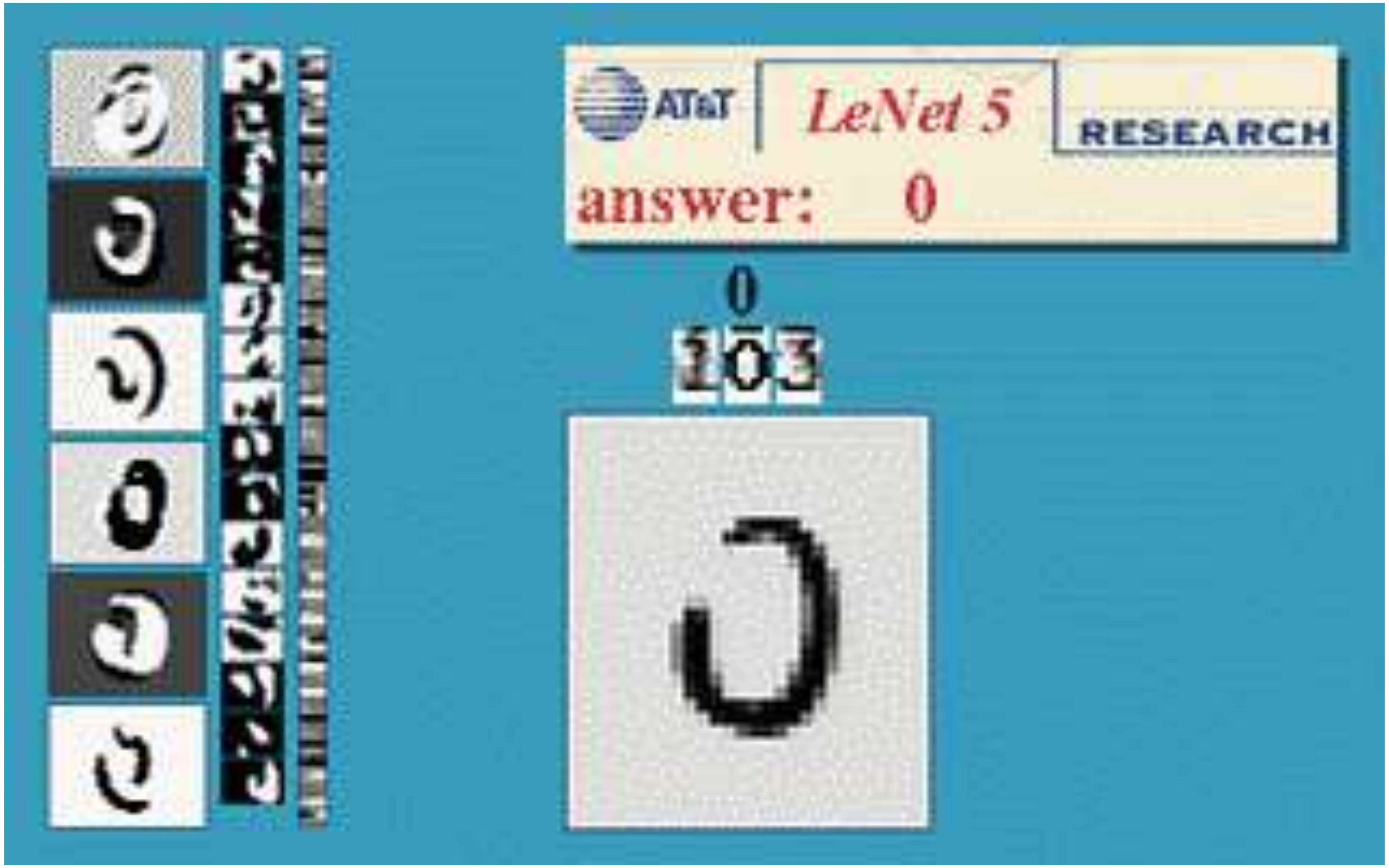
Convolutional neural networks



[Lecun, Bottou, Bengio, and Haffner, “Gradient-Based Learning Applied to Document Recognition”, 1998]

Clever architectures

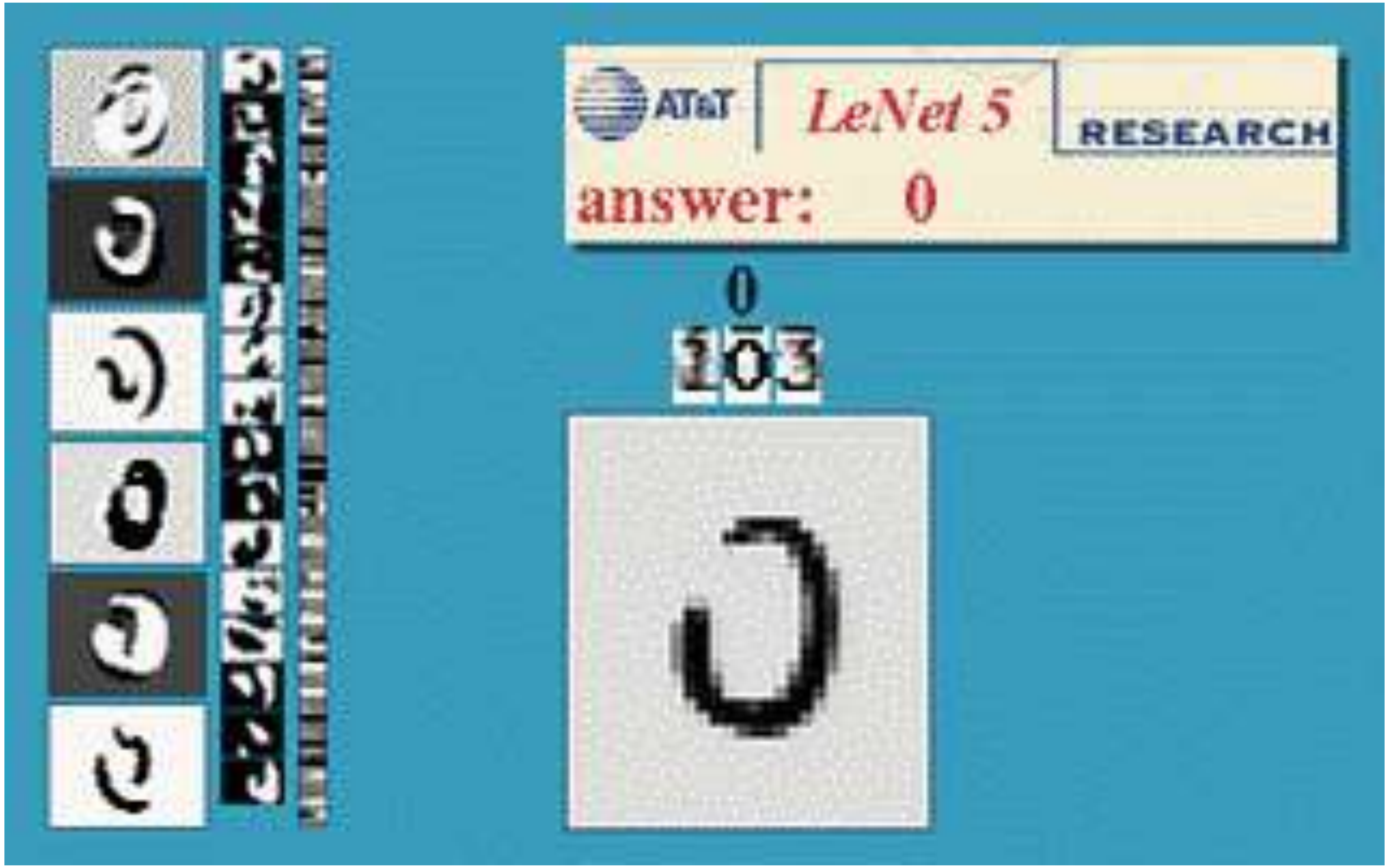
Convolutional neural networks

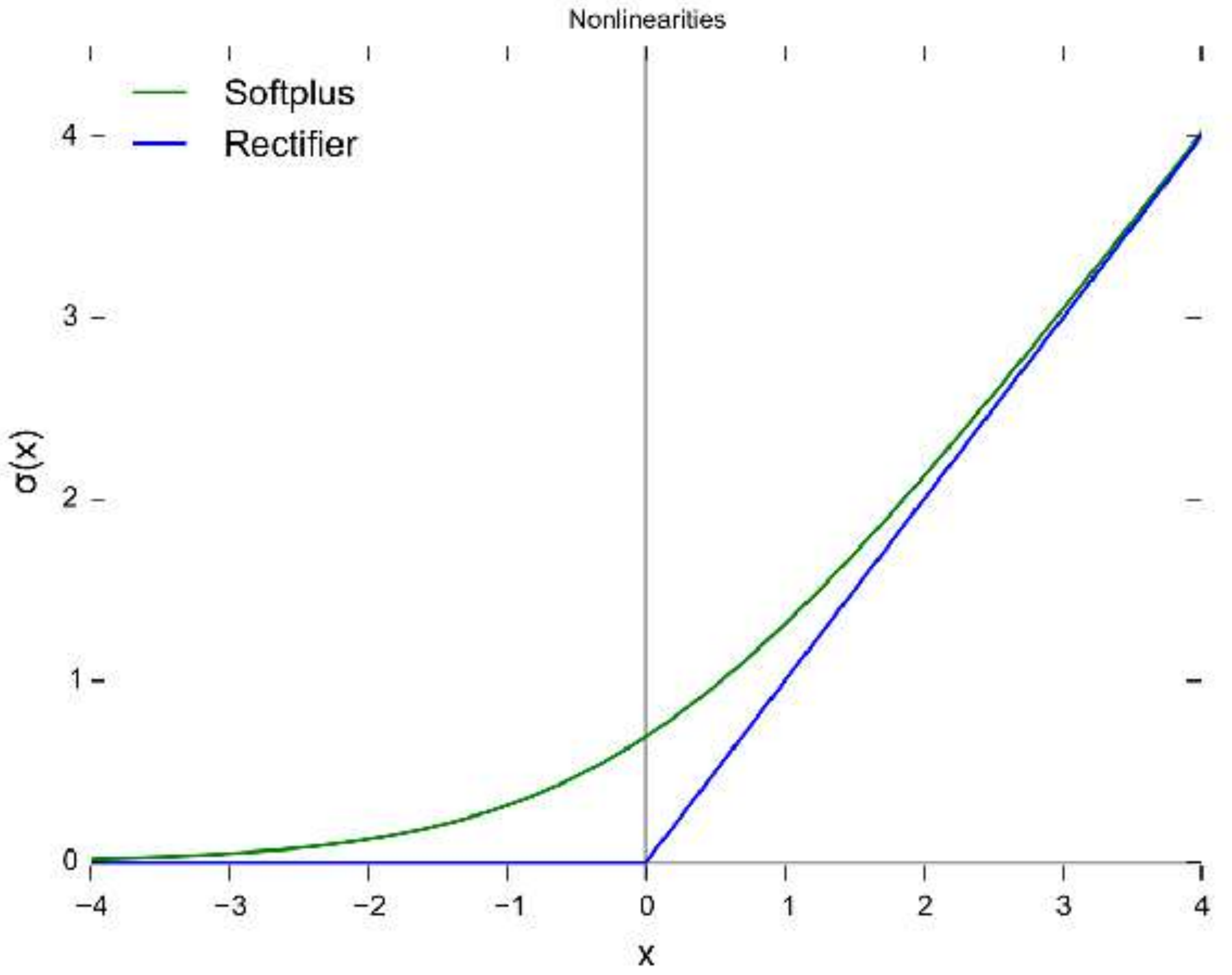


[Lecun, Bottou, Bengio, and Haffner, "Gradient-Based Learning Applied to Document Recognition", 1998]

Clever architectures

Convolutional neural networks



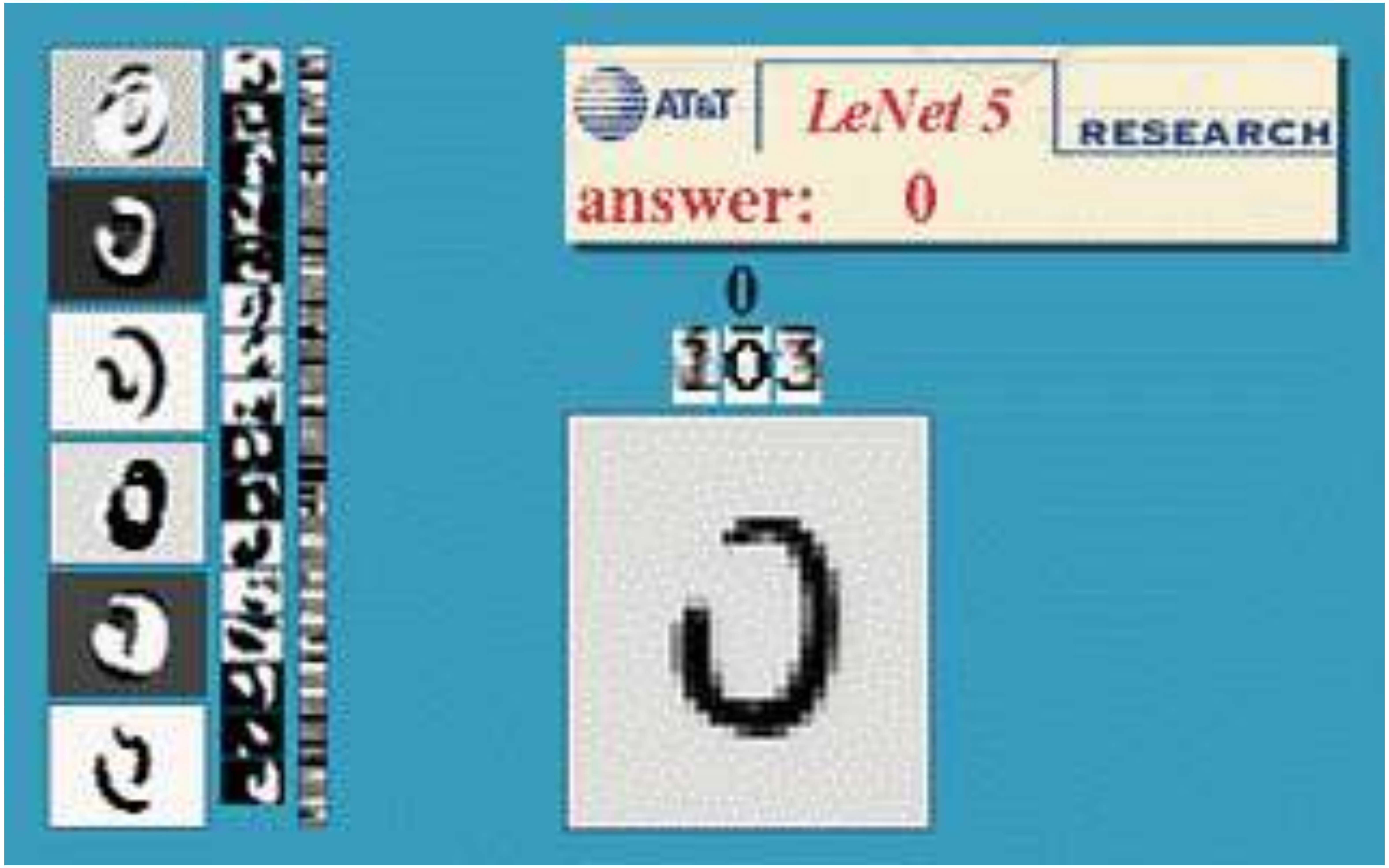


[Nair and Hinton, "Rectified Linear Units Improve Restricted Boltzmann Machines", 2010]

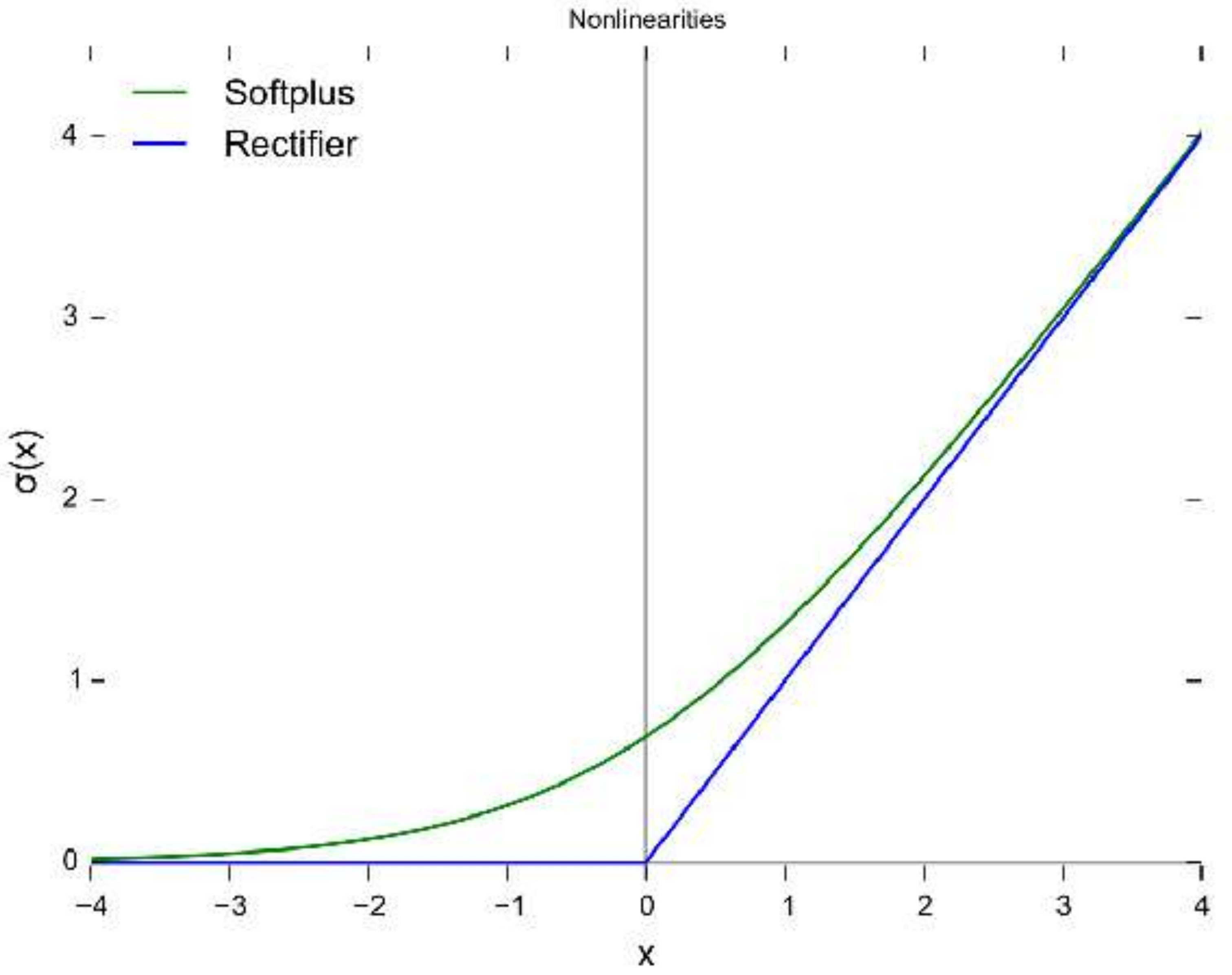
[Lecun, Bottou, Bengio, and Haffner, "Gradient-Based Learning Applied to Document Recognition", 1998]

Clever architectures

Convolutional neural networks



[Lecun, Bottou, Bengio, and Haffner, "Gradient-Based Learning Applied to Document Recognition", 1998]



[Nair and Hinton, "Rectified Linear Units Improve Restricted Boltzmann Machines", 2010]

Clever architectures

Transformers, Vaswani et al., 2017

Attention is all you need

Authors	Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, Illia Polosukhin
Publication date	2017
Journal	Advances in neural information processing systems
Volume	30
Description	The dominant sequence transduction models are based on complex recurrent orconvolutional neural networks in an encoder and decoder configuration. The best performing such models also connect the encoder and decoder through an attentionm echanisms. We propose a novel, simple network architecture based solely onan attention mechanism, dispensing with recurrence and convolutions entirely. Experiments on two machine translation tasks show these models to be superiorin quality while being more parallelizable and requiring significantly less timeto train. Our single model with 165 million parameters, achieves 27.5 BLEU onEnglish-to-German translation, improving over the existing best ensemble result by over 1 BLEU. On English-to-French translation, we outperform the previoussingle state-of-the-art with model by 0.7 BLEU, achieving a BLEU score of 41.1.
Total citations	Cited by 87925

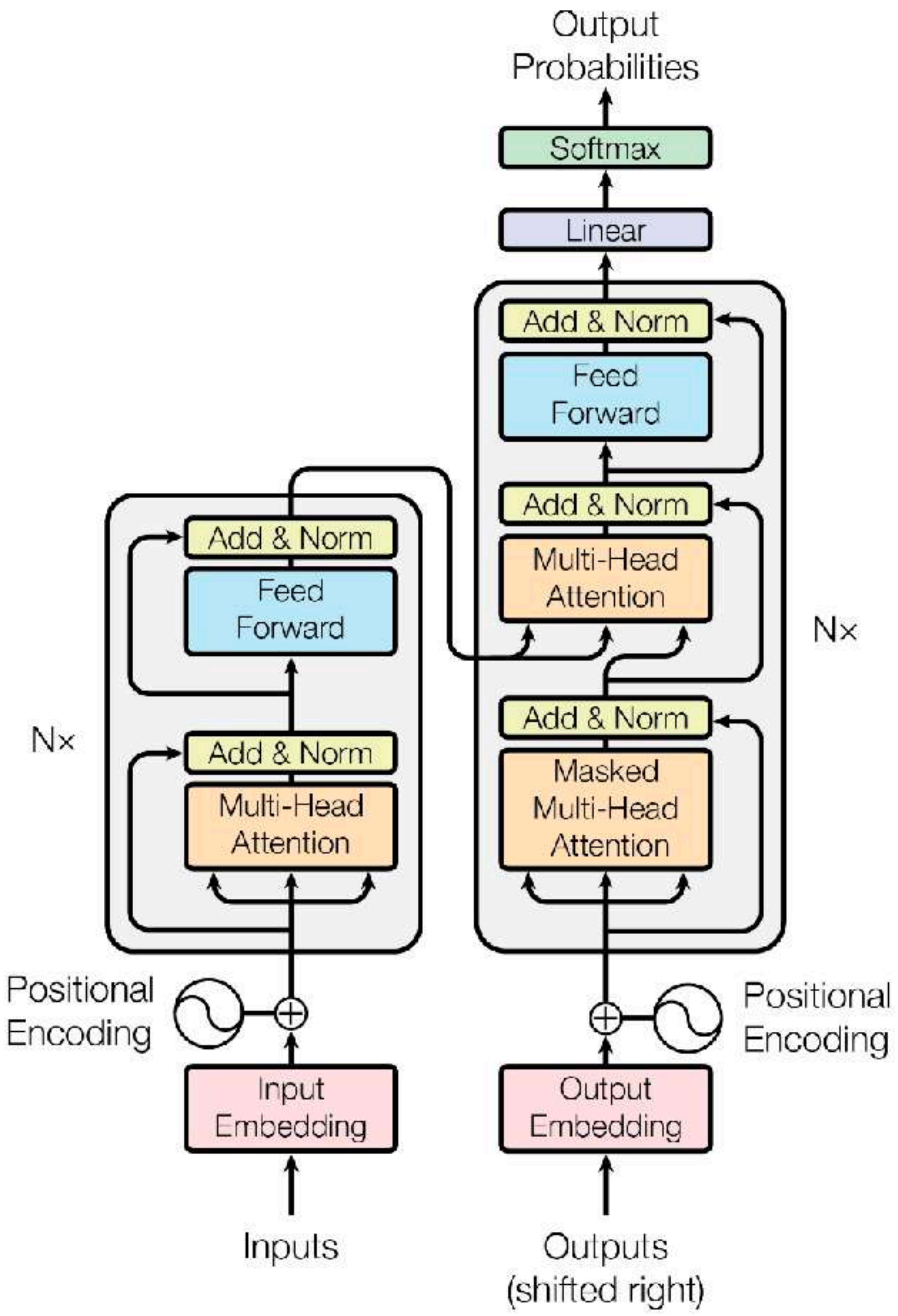
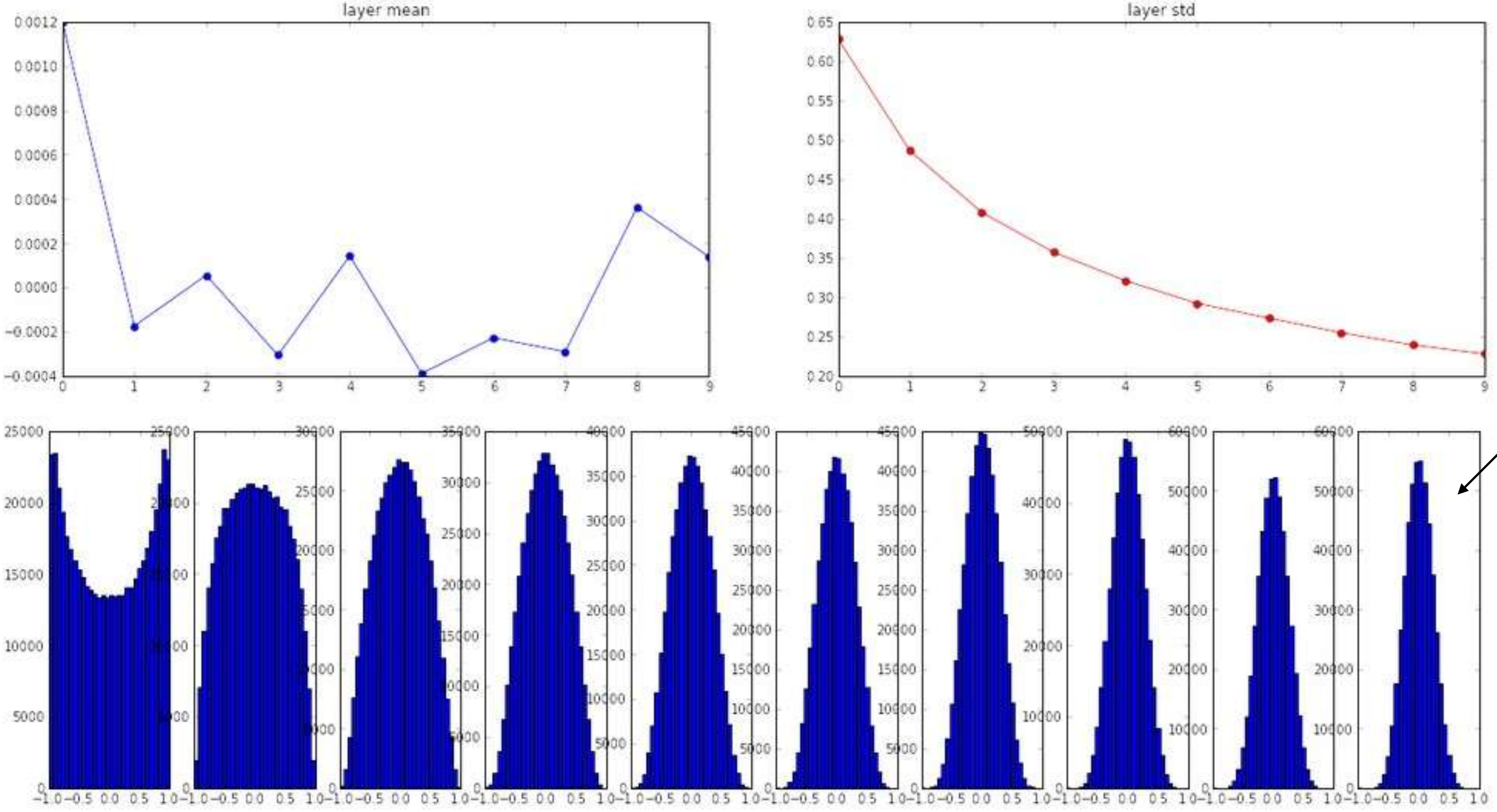


Figure 1: The Transformer - model architecture.

Proper initialization schemes

Much more important than you think

“Xavier initialization”
[Glorot et al., 2010]

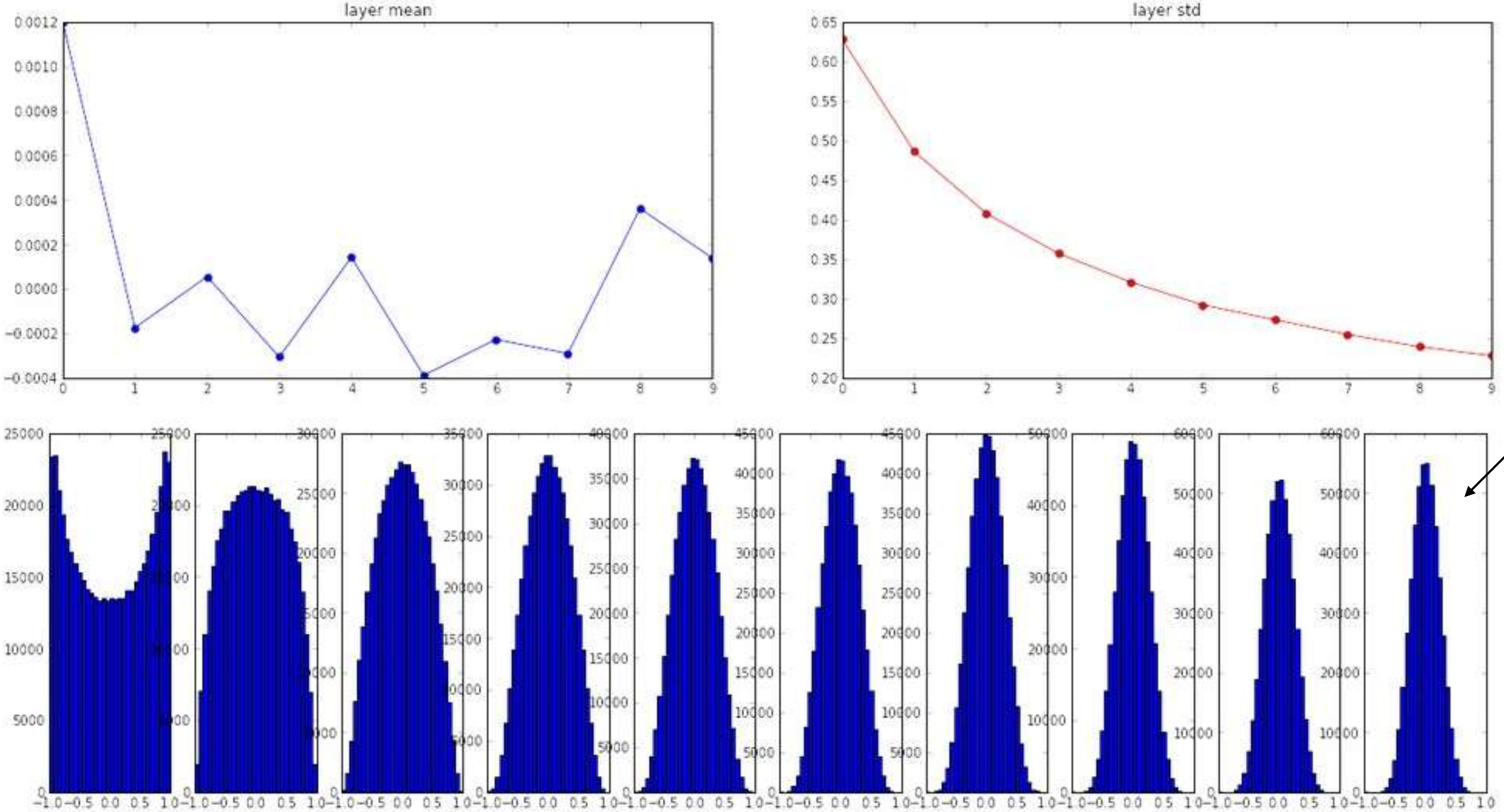


Neuron activations are well spread

Proper initialization schemes

Much more important than you think

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



Neuron activations are well spread

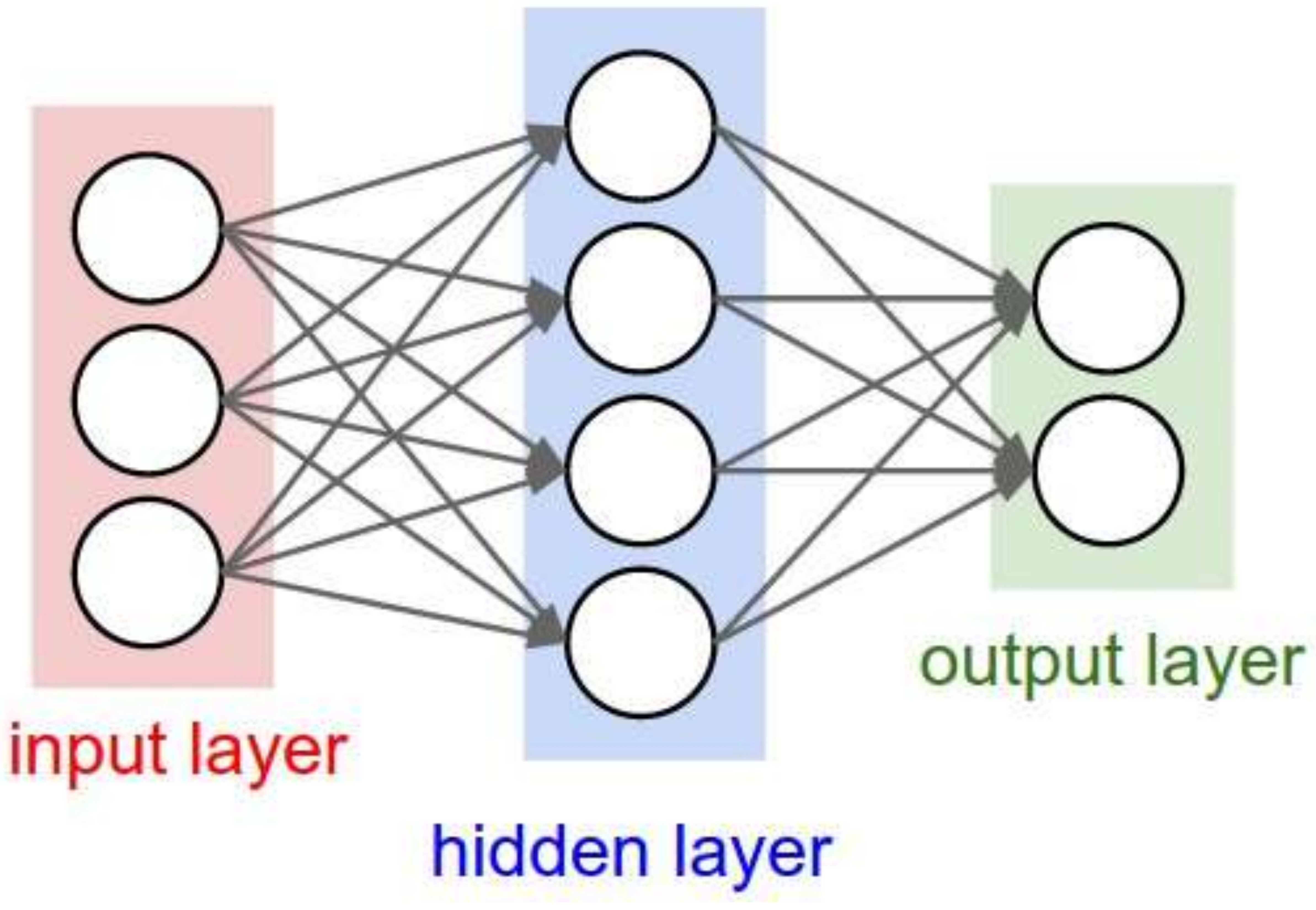
Based on slides for [Stanford cs231n](#) by Li, Jonson, and Young. Modified and reused with permission

Weight initialization

Weight initialization

A look into ways things can go wrong

Q: what happens when W=0 init is used?



Based on slides for [Stanford cs231n] by Li, Jonson, and Young. Modified and reused with permission

Weight initialization

A look into ways things can go wrong

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.

THE UNIVERSITY OF BRITISH COLUMBIA

We

A loo

github.com/eriklindernoren/PyTorch-GAN/blob/master/implementations/lsgan/lsgan.py

```python
parser.add_argument("--img_size", type=int, default=32, help="size of each image dimension")
parser.add_argument("--channels", type=int, default=1, help="number of image channels")
parser.add_argument("--sample_interval", type=int, default=1000, help="number of image channels")
opt = parser.parse_args()
print(opt)

cuda = True if torch.cuda.is_available() else False


def weights_init_normal(m):
    classname = m.__class__.__name__
    if classname.find("Conv") != -1:
        torch.nn.init.normal_(m.weight.data, 0.0, 0.02)
    elif classname.find("BatchNorm") != -1:
        torch.nn.init.normal_(m.weight.data, 1.0, 0.02)
        torch.nn.init.constant_(m.bias.data, 0.0)


class Generator(nn.Module):
    def __init__(self):
        super(Generator, self).__init__()

        self.init_size = opt.img_size // 4
        self.l1 = nn.Sequential(nn.Linear(opt.latent_dim, 128 * self.init_size ** 2))
```

Weight initialization

A look into ways things can go wrong

Let's look at some activation statistics

E.g. 10-layer net with 500 neurons on each layer, using tanh non-linearities, and initializing as described in last slide.

```
# assume some unit gaussian 10-D input data
D = np.random.randn(1000, 500)
hidden_layer_sizes = [500]*10
nonlinearities = ['tanh']*len(hidden_layer_sizes)
```

```
act = {'relu':lambda x:np.maximum(0,x), 'tanh':lambda x:np.tanh(x)}
Hs = {}
for i in xrange(len(hidden_layer_sizes)):
    X = D if i == 0 else Hs[i-1] # input at this layer
    fan_in = X.shape[1]
    fan_out = hidden_layer_sizes[i]
    W = np.random.randn(fan_in, fan_out) * 0.01 # layer initialization

    H = np.dot(X, W) # matrix multiply
    H = act[nonlinearities[i]](H) # nonlinearity
    Hs[i] = H # cache result on this layer
```

Init with small random numbers

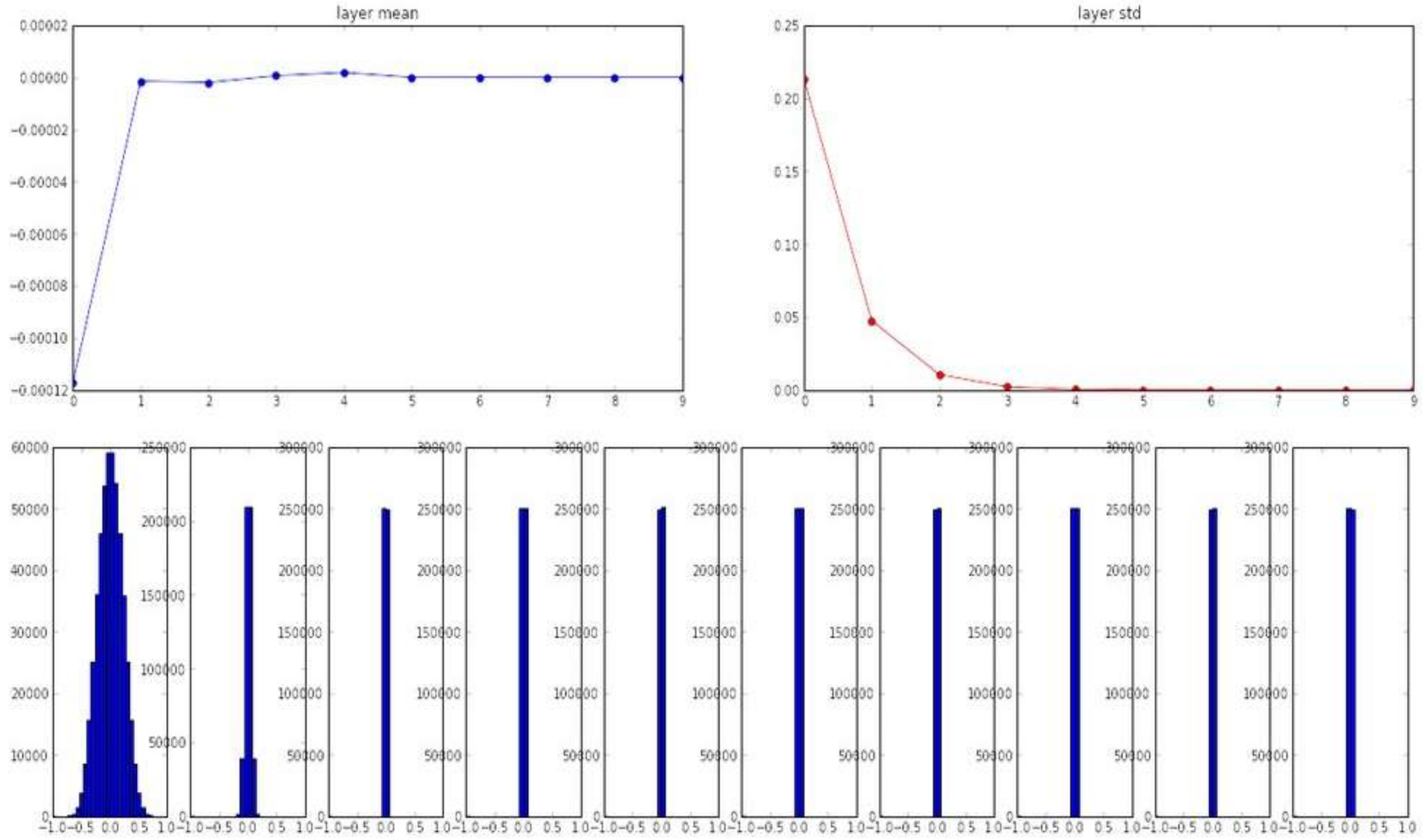
```
# look at distributions at each layer
print 'input layer had mean %f and std %f' % (np.mean(D), np.std(D))
layer_means = [np.mean(H) for i,H in Hs.iteritems()]
layer_stds = [np.std(H) for i,H in Hs.iteritems()]
for i,H in Hs.iteritems():
    print 'hidden layer %d had mean %f and std %f' % (i+1, layer_means[i], layer_stds[i])

# plot the means and standard deviations
plt.figure()
plt.subplot(121)
plt.plot(Hs.keys(), layer_means, 'ob-')
plt.title('layer mean')
plt.subplot(122)
plt.plot(Hs.keys(), layer_stds, 'or-')
plt.title('layer std')

# plot the raw distributions
plt.figure()
for i,H in Hs.iteritems():
    plt.subplot(1,len(Hs),i+1)
    plt.hist(H.ravel(), 30, range=(-1,1))
```

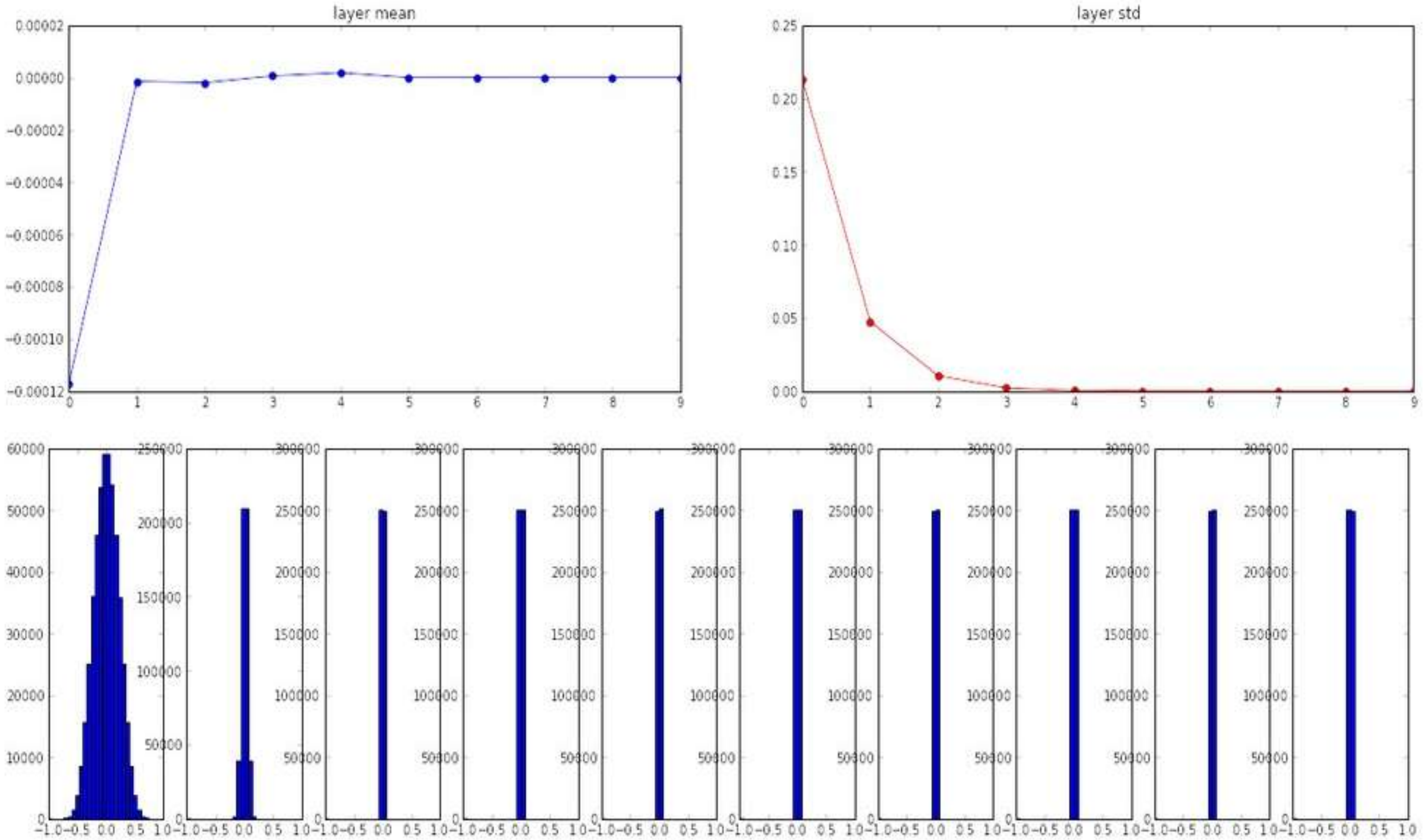
Weight initialization

A look into ways things can go wrong



Weight initialization

A look into ways things can go wrong



All activations become zero!

Q: think about the backward pass. What do the gradients look like?

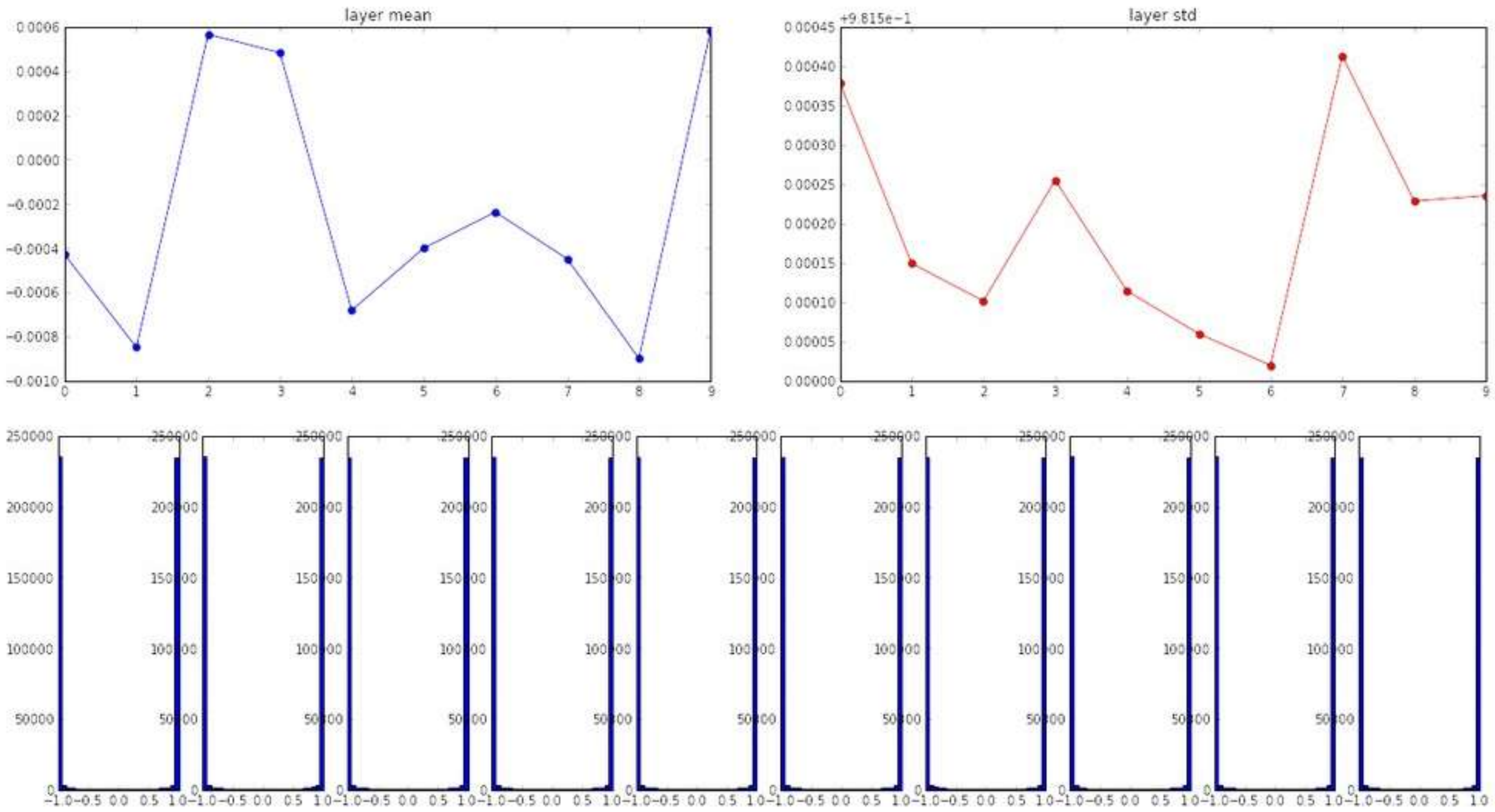
Hint: think about backward pass for a **W*X** gate.

Weight initialization

A look into ways things can go wrong

```
W = np.random.randn(fan_in, fan_out) * 1.0 # layer initialization
```

*1.0 instead of *0.01



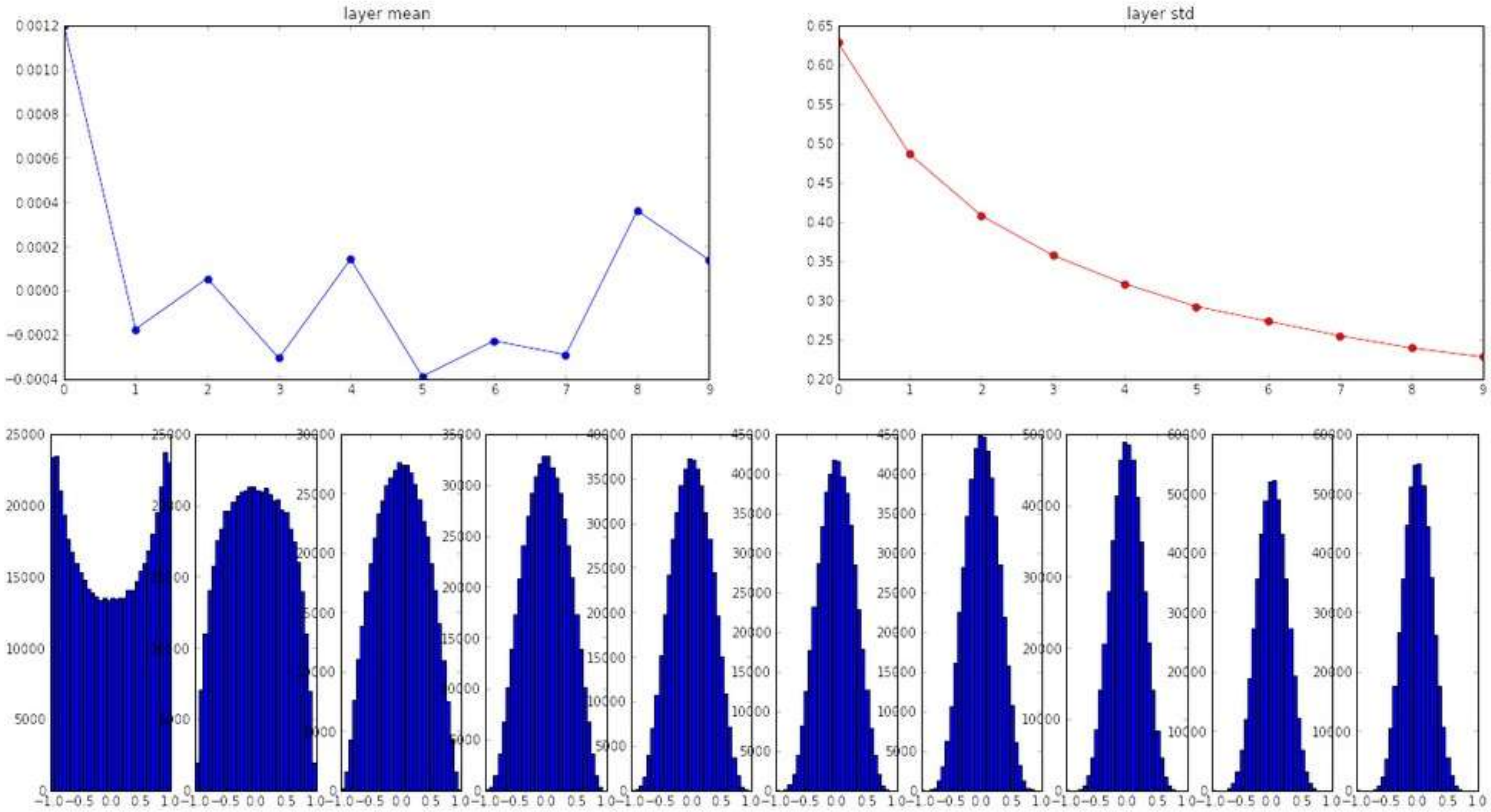
Almost all neurons completely saturated, either -1 and 1. Gradients will be all zero.

Weight initialization

Glorot initialization [Glorot et al., 2010]

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

Some number according to *"fan in"*



Statistically motivated

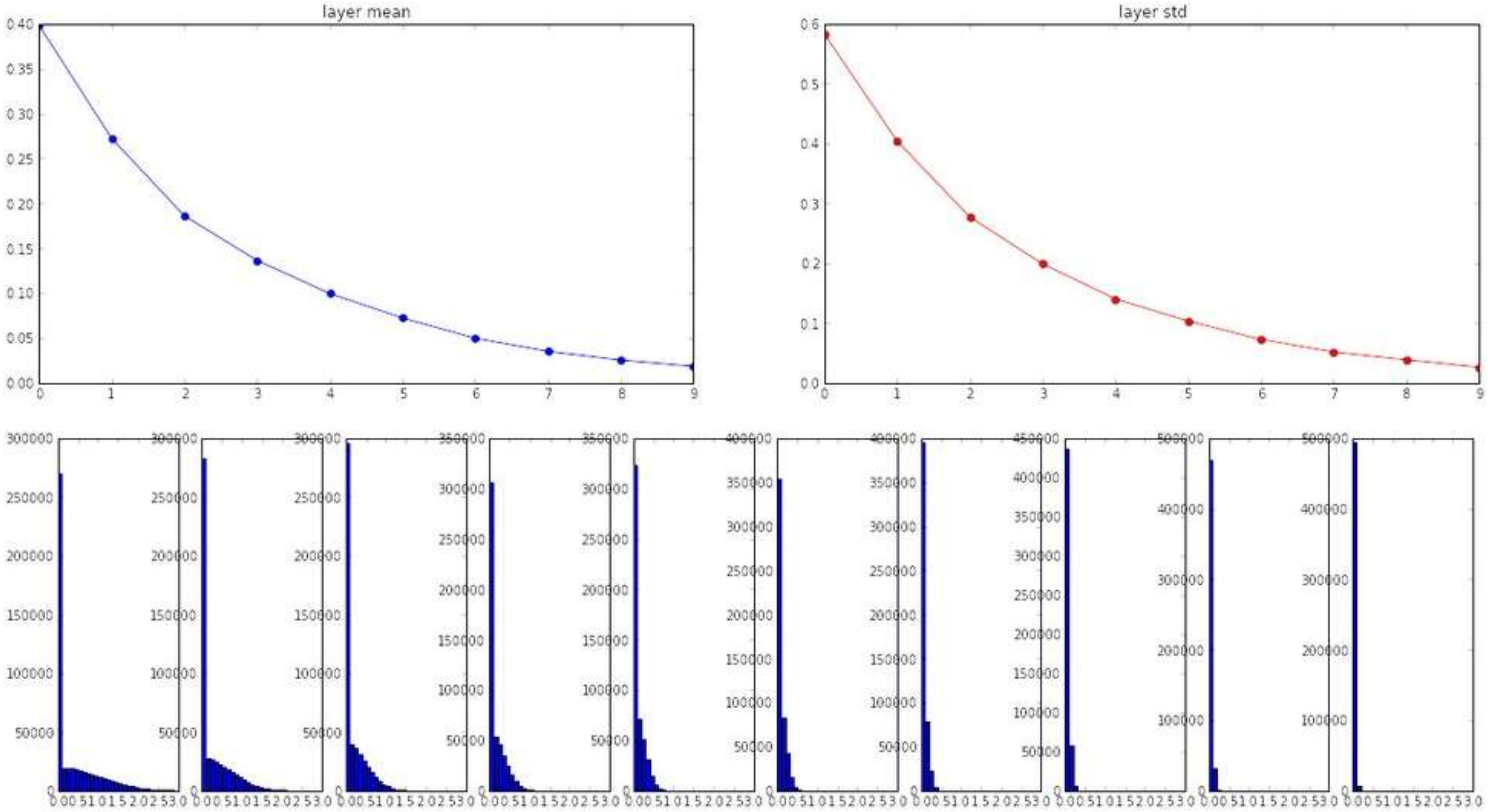
Good for tanh

Weight initialization

Glorot initialization [Glorot et al., 2010]

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

Some number according to “*fan in*”



Statistically motivated

Good for tanh

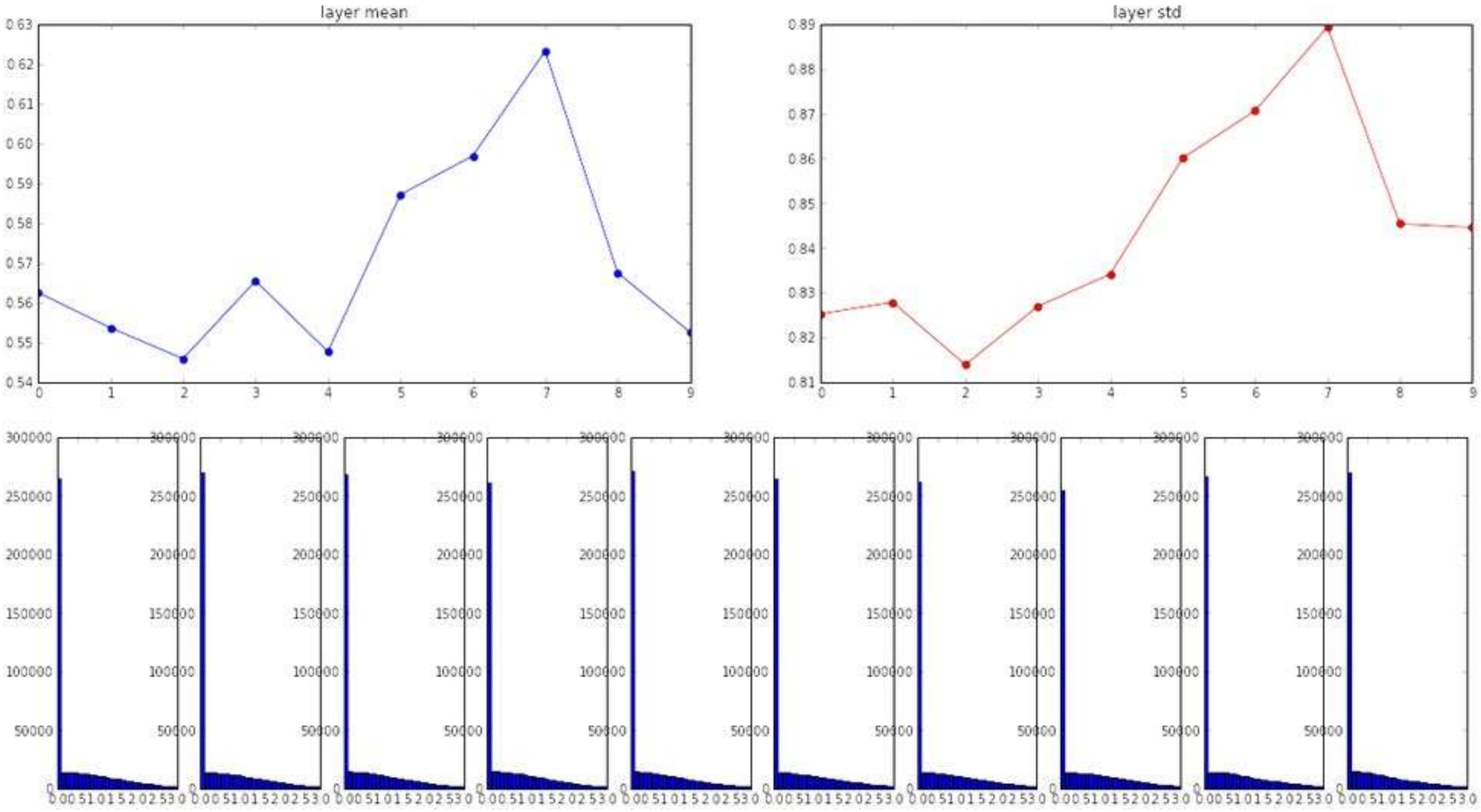
Not so good for ReLU

Weight initialization

He initialization [He et al., 2015]

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

magic number 2



Statistically motivated

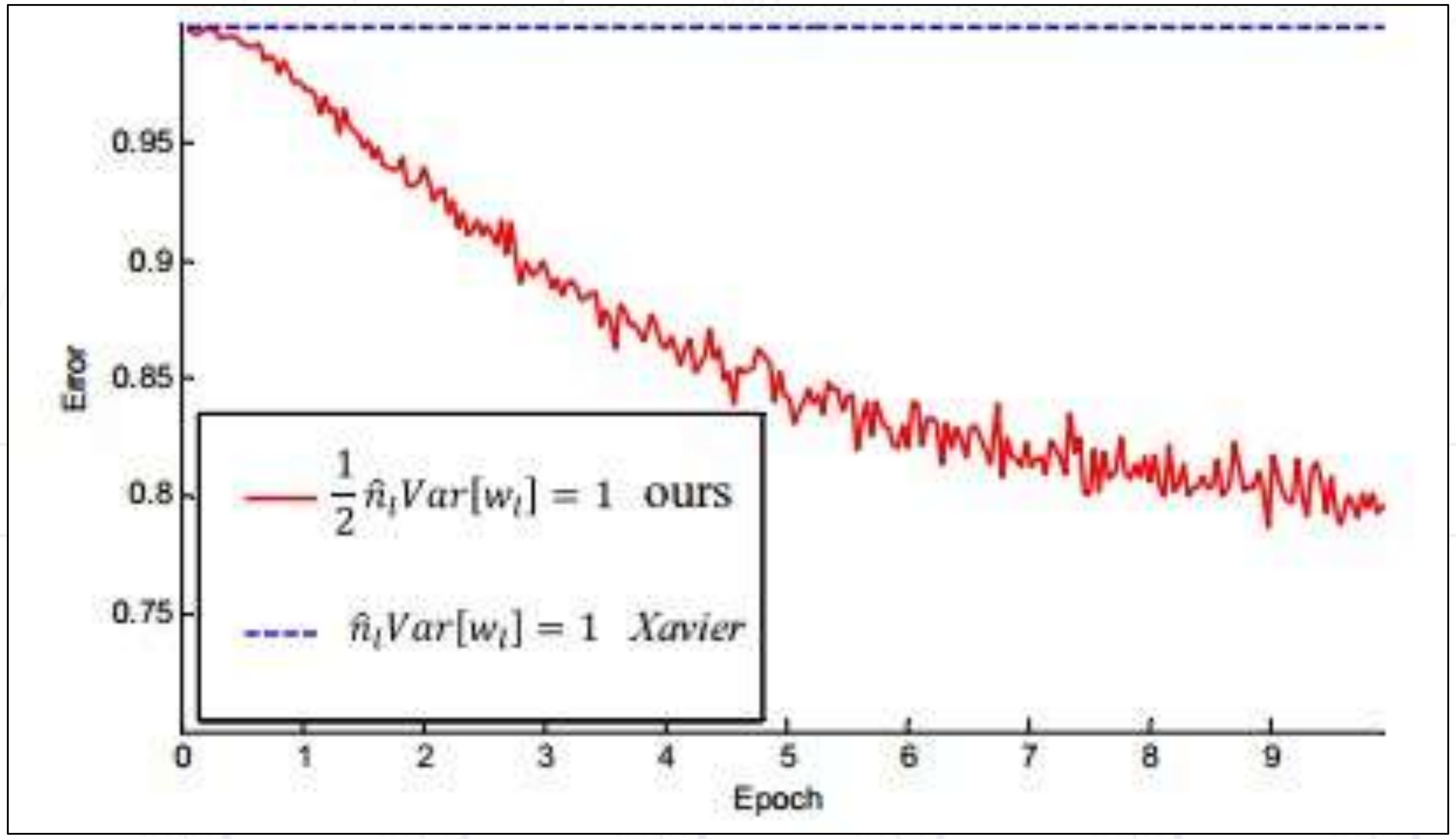
Good for ReLU

Weight initialization

He initialization [He et al., 2015]

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

magic number 2

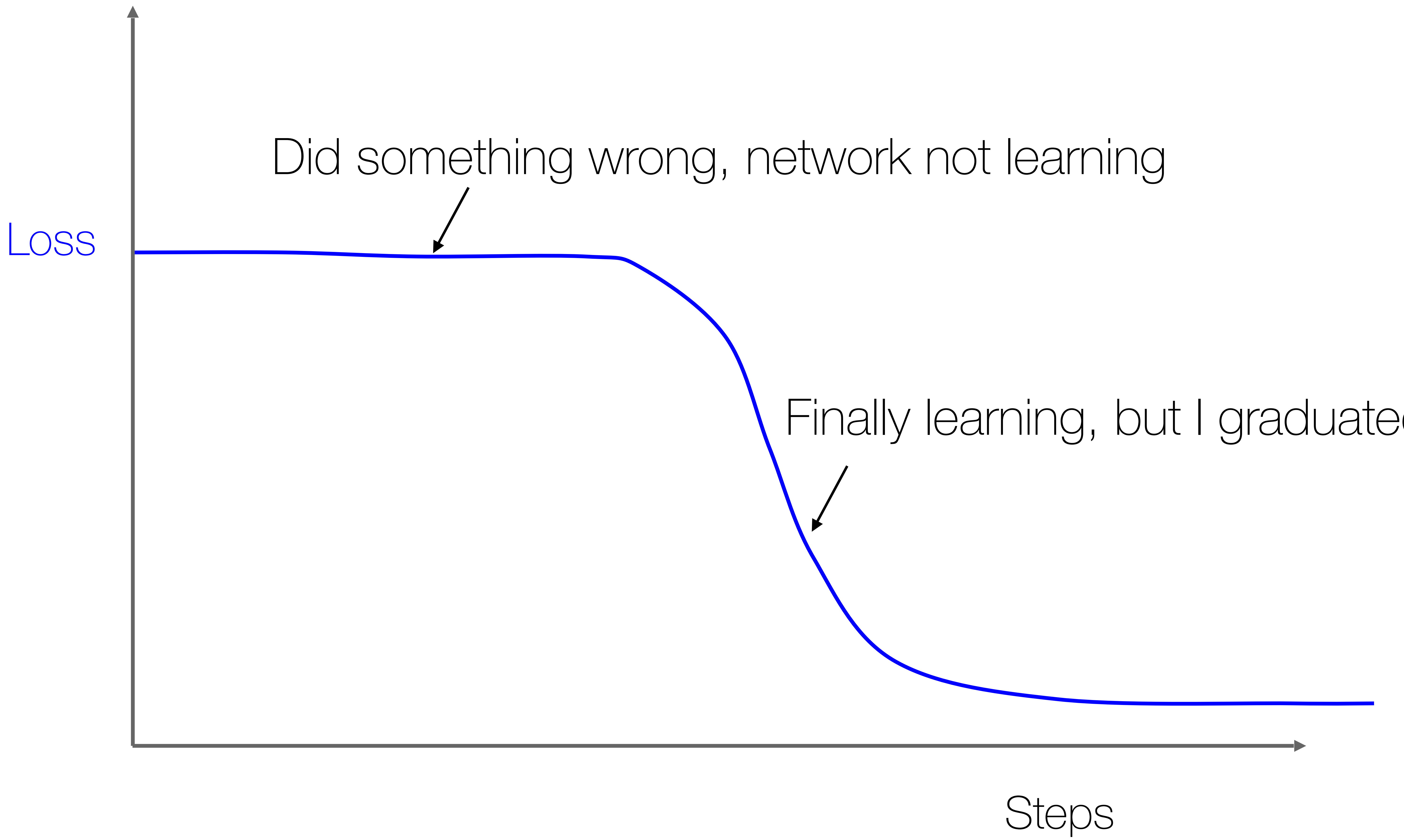


Statistically motivated

Good for ReLU

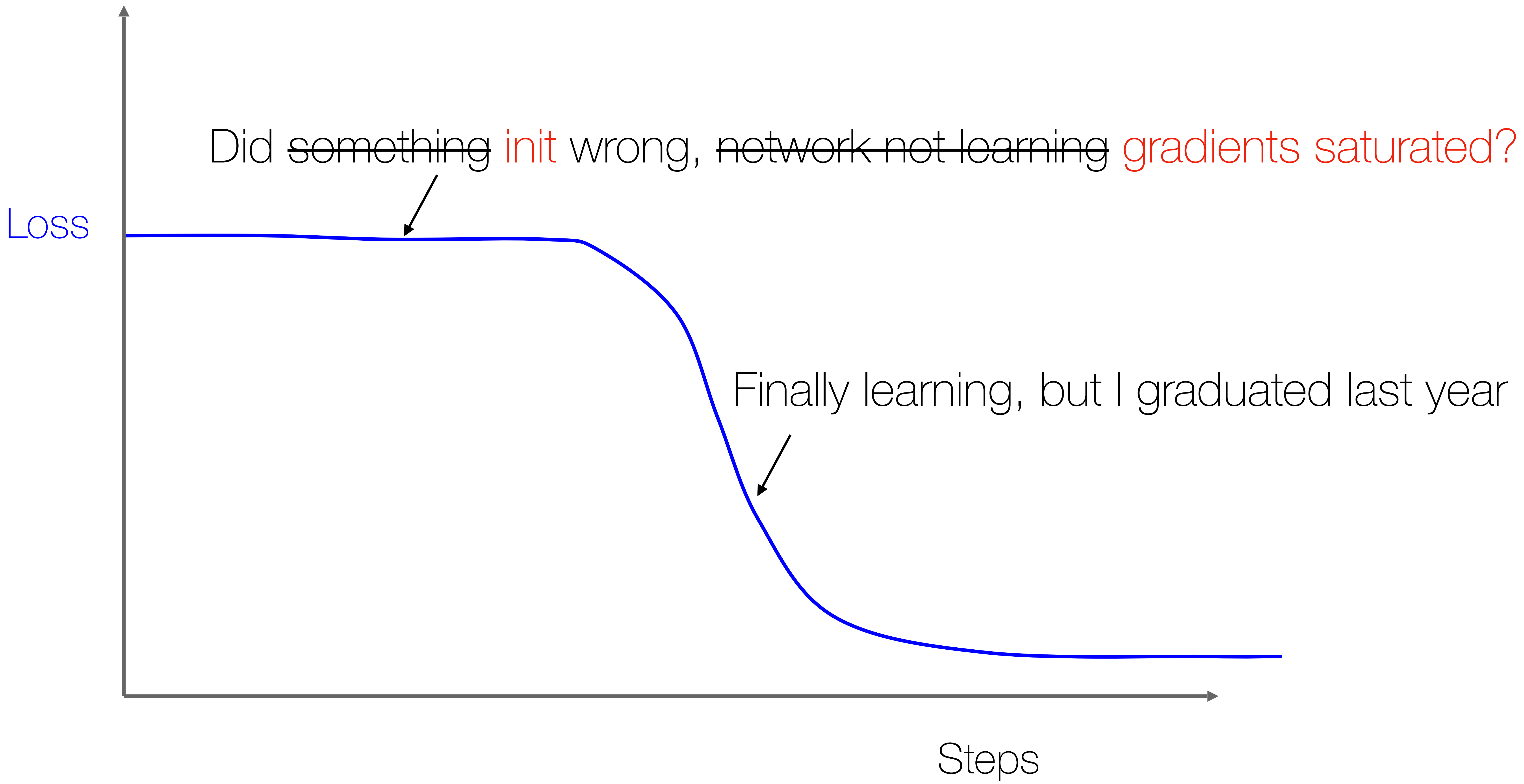
Recall: But it is never that easy

A typical sad loss curve



Recall: But it is never that easy

A typical sad loss curve

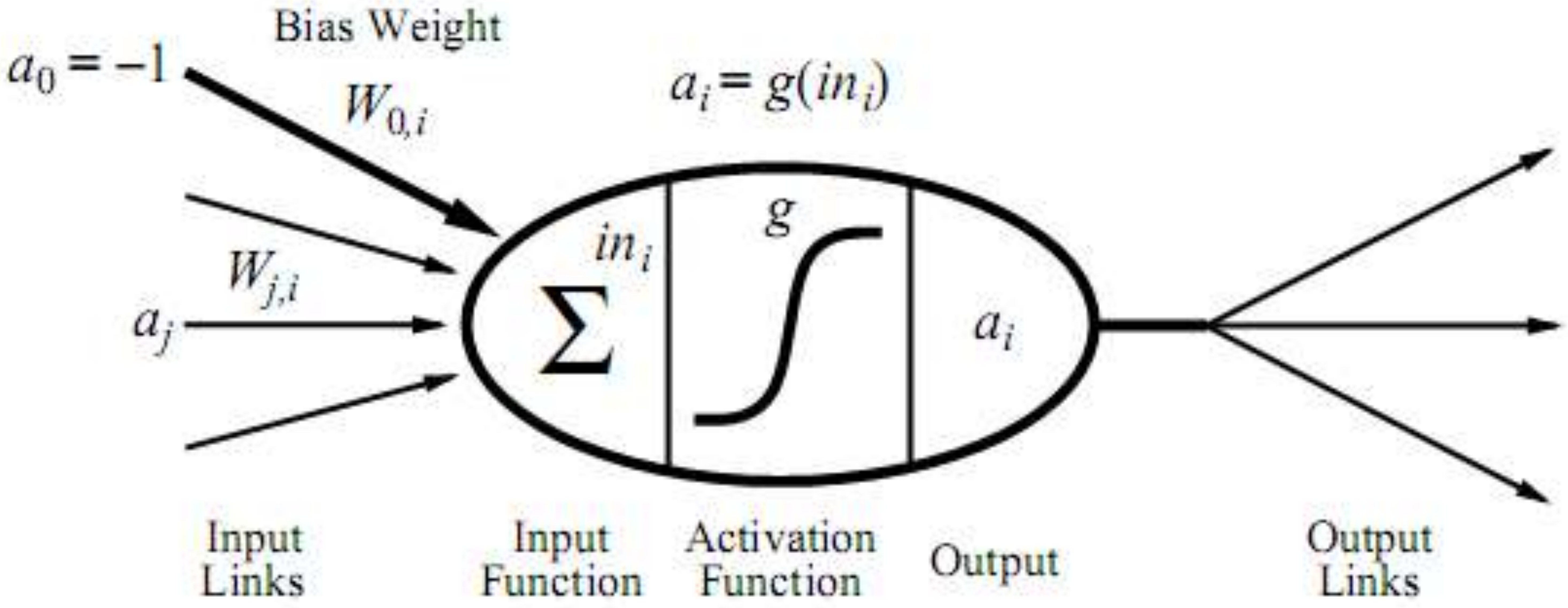


Normalization

Batch normalization [Ioffe and Szegedy, 2015]

Recall…

$$a_i \leftarrow g(in_i) = g\left(\Sigma_j W_{j,i} a_j\right)$$

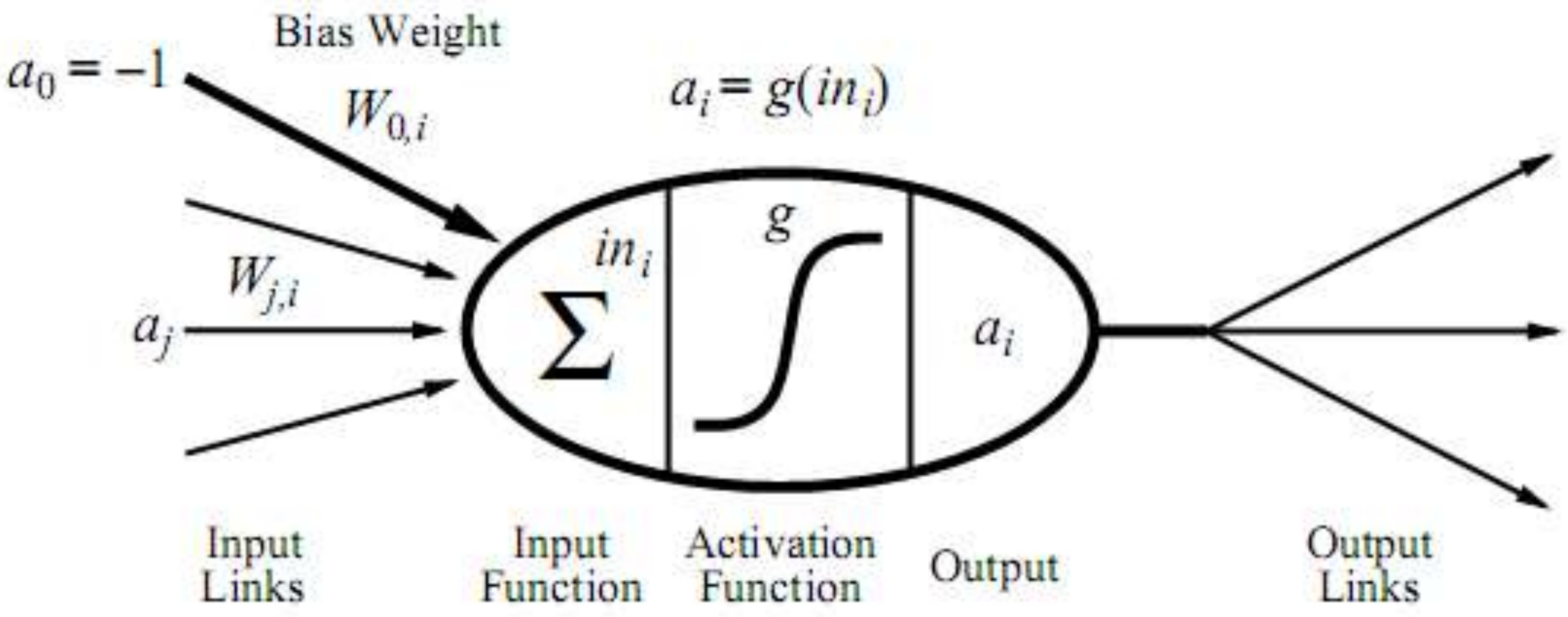


Batch normalization [Ioffe and Szegedy, 2015]

Recall…

Linear operations should cancel out

$$a_i \leftarrow g(in_i) = g\left(\Sigma_j W_{j,i} a_j\right)$$



Batch normalization [Ioffe and Szegedy, 2015]

Forcing a zero-mean and unit standard deviation

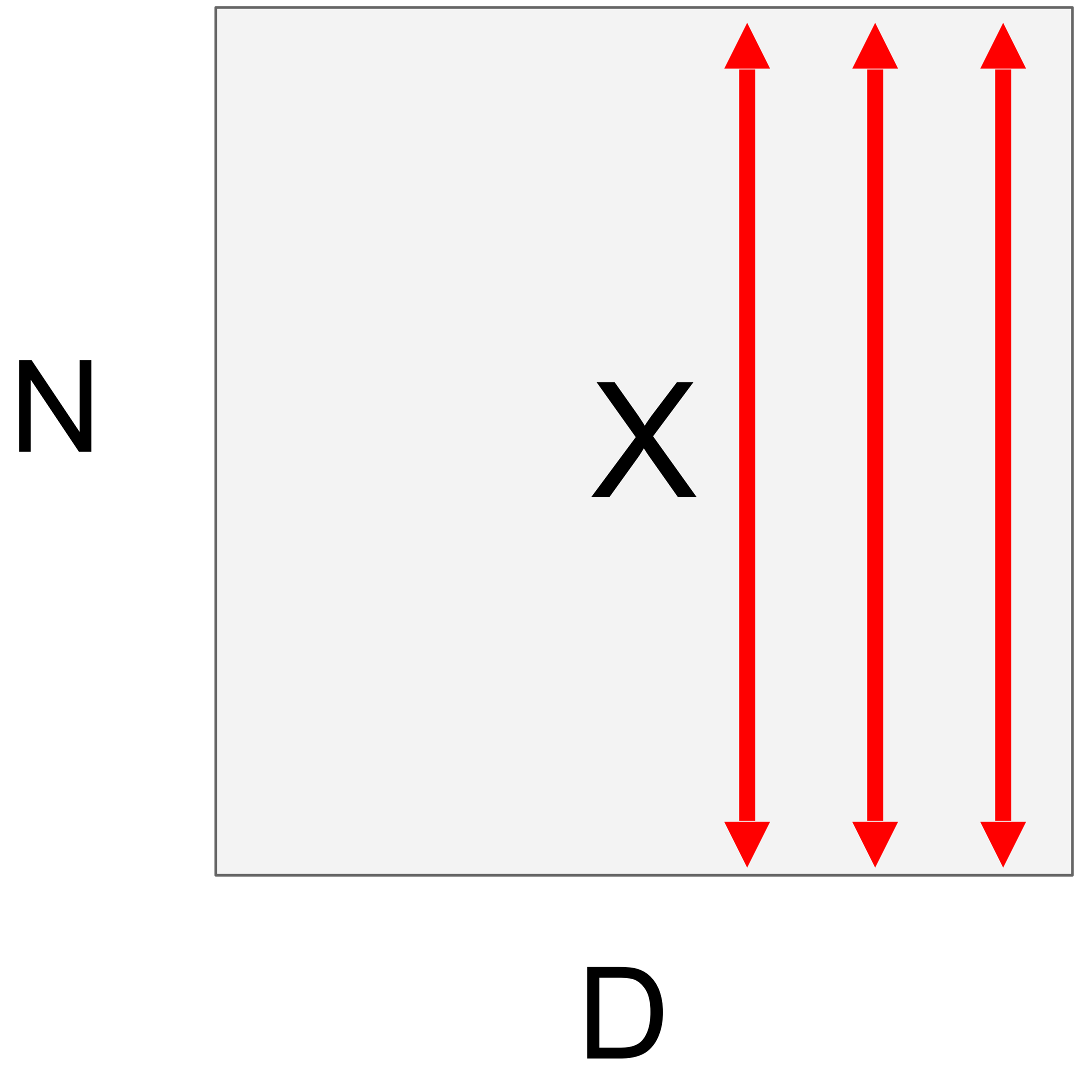
consider a batch of activations at some layer. To make each dimension unit gaussian, apply:

$$\widehat{x}^{(k)} = \frac{x^{(k)} - \mathrm{E}[x^{(k)}]}{\sqrt{\mathrm{Var}[x^{(k)}]}}$$

this is a linear differentiable function...

Batch normalization [Ioffe and Szegedy, 2015]

Forcing a zero-mean and unit standard deviation



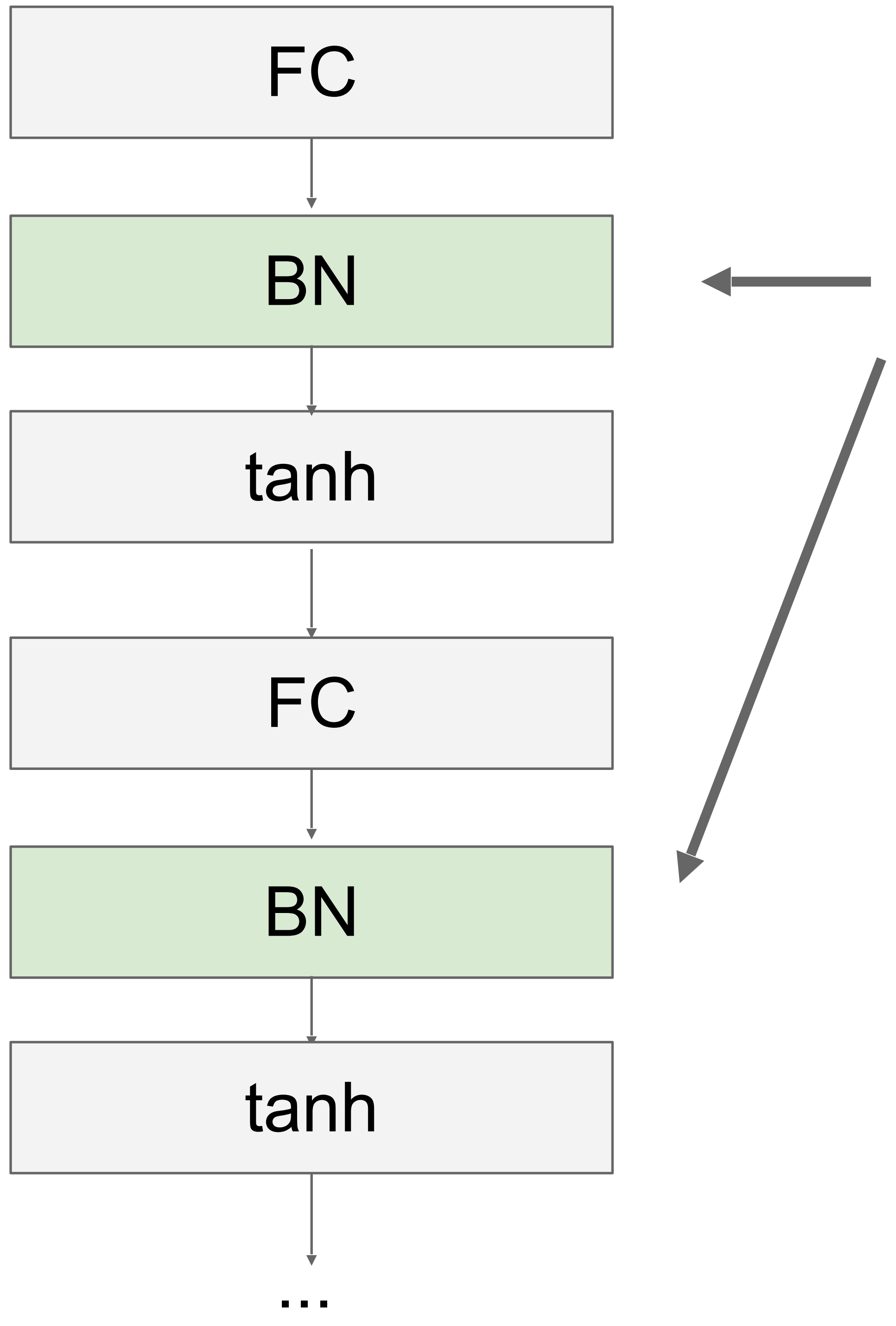
1. Compute the empirical mean and variance independently for each dimension.

2. Normalize

$$\widehat{x}^{(k)} = \frac{x^{(k)} - \mathrm{E}[x^{(k)}]}{\sqrt{\mathrm{Var}[x^{(k)}]}}$$

Batch normalization [Ioffe and Szegedy, 2015]

Forcing a zero-mean and unit standard deviation



Usually inserted after Fully Connected or Convolutional layers, and before nonlinearity.

$$\widehat{x}^{(k)} = \frac{x^{(k)} - \mathrm{E}[x^{(k)}]}{\sqrt{\mathrm{Var}[x^{(k)}]}}$$

Batch normalization [Ioffe and Szegedy, 2015]

Introducing learnable scale / shift

$$\widehat{x}^{(k)} = \frac{x^{(k)} - \mathrm{E}[x^{(k)}]}{\sqrt{\mathrm{Var}[x^{(k)}]}}$$

And then allow the network to squash the range if it wants to:

$$y^{(k)} = \gamma^{(k)}\widehat{x}^{(k)} + \beta^{(k)}$$

Note, the network can learn:

$$\gamma^{(k)} = \sqrt{\mathrm{Var}[x^{(k)}]}$$

$$\beta^{(k)} = \mathrm{E}[x^{(k)}]$$

to recover the identity mapping.

Batch normalization [Ioffe and Szegedy, 2015]

Introducing learnable scale / shift

IMPORTANT: At test time, we don't have these — use training time stats

$$\widehat{x}^{(k)} = \frac{x^{(k)} - \mathrm{E}[x^{(k)}]}{\sqrt{\mathrm{Var}[x^{(k)}]}}$$

And then allow the network to squash the range if it wants to:

$$y^{(k)} = \gamma^{(k)}\widehat{x}^{(k)} + \beta^{(k)}$$

Note, the network can learn:

$$\gamma^{(k)} = \sqrt{\mathrm{Var}[x^{(k)}]}$$

$$\beta^{(k)} = \mathrm{E}[x^{(k)}]$$

to recover the identity mapping.

Skipped in class
(outside of scope)

Other normalization techniques

Batch Normalization

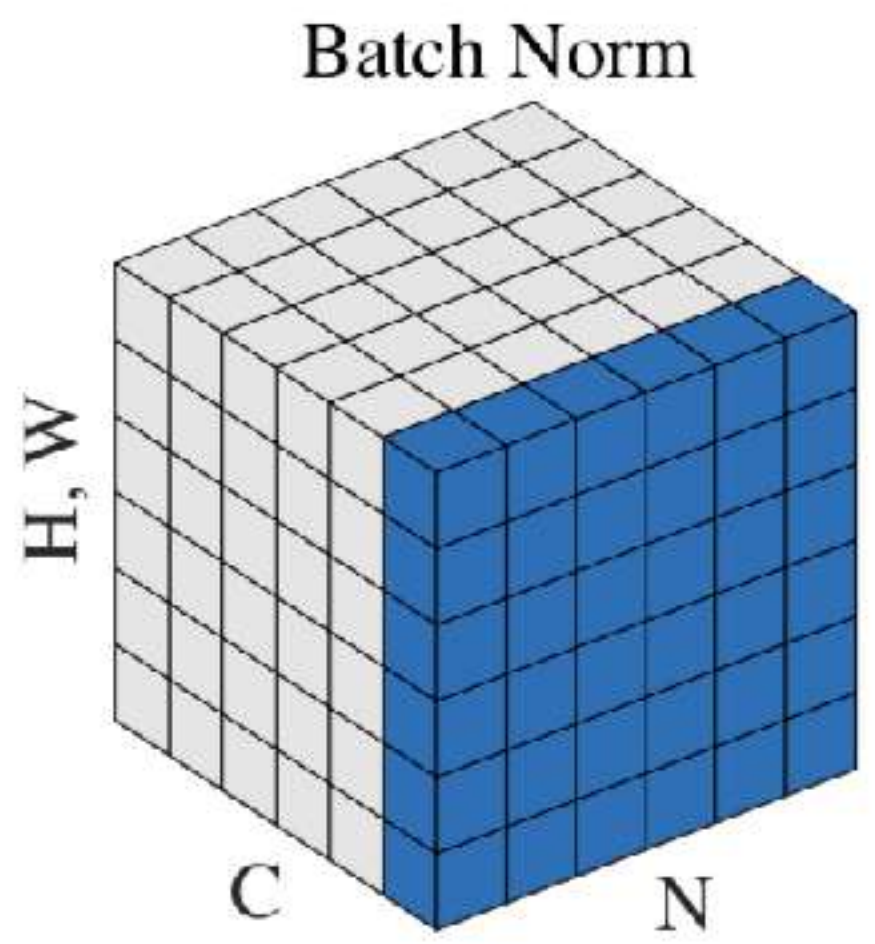


Image from Wu and He 2018. Reproduced for educational purposes.

Skipped in class (outside of scope)

Other normalization techniques

Batch Normalization

Batch Normalization for **fully-connected** networks

$$\mathbf{x}: N \times D$$

Normalize

$$\mu, \sigma: 1 \times D$$
$$\gamma, \beta: 1 \times D$$
$$y = \gamma(x-\mu)/\sigma+\beta$$

Batch Normalization for **convolutional** networks (Spatial Batchnorm, BatchNorm2D)

$$\mathbf{x}: N \times C \times H \times W$$

Normalize

$$\mu, \sigma: 1 \times C \times 1 \times 1$$
$$\gamma, \beta: 1 \times C \times 1 \times 1$$
$$y = \gamma(x-\mu)/\sigma+\beta$$

Skipped in class (outside of scope)

Other normalization techniques

Batch Normalization

Batch Normalization for **fully-connected** networks

$$\mathbf{x}: \boxed{N} \times D$$

Normalize

$$\boldsymbol{\mu}, \boldsymbol{\sigma}: \boxed{1} \times D$$

$$\gamma, \beta: 1 \times D$$

$$y = \gamma(x-\mu)/\sigma+\beta$$

Batch Normalization for **convolutional** networks (Spatial Batchnorm, BatchNorm2D)

$$\mathbf{x}: \boxed{N} \times C \times H \times W$$

Normalize

$$\boldsymbol{\mu}, \boldsymbol{\sigma}: \boxed{1} \times C \times 1 \times 1$$

$$\gamma, \beta: 1 \times C \times 1 \times 1$$

$$y = \gamma(x-\mu)/\sigma+\beta$$

This is why train/test needs to be different

Skipped in class (outside of scope)

Other normalization techniques

Batch Normalization

Batch Normalization for **fully-connected** networks

$$\mathbf{x}: \boxed{N} \times D$$

Normalize

$$\boldsymbol{\mu}, \boldsymbol{\sigma}: \boxed{1} \times D$$

$$\gamma, \beta: 1 \times D$$

$$y = \gamma(x-\mu)/\sigma+\beta$$

Batch Normalization for **convolutional** networks (Spatial Batchnorm, BatchNorm2D)

$$\mathbf{x}: \boxed{N} \times C \times H \times W$$

Normalize

$$\boldsymbol{\mu}, \boldsymbol{\sigma}: \boxed{1} \times C \times 1 \times 1$$

$$\gamma, \beta: 1 \times C \times 1 \times 1$$

$$y = \gamma(x-\mu)/\sigma+\beta$$

Always watch out when implementing!!!

This is why train/test needs to be different

Skipped in class (outside of scope)

Other normalization techniques

Batch Normalization

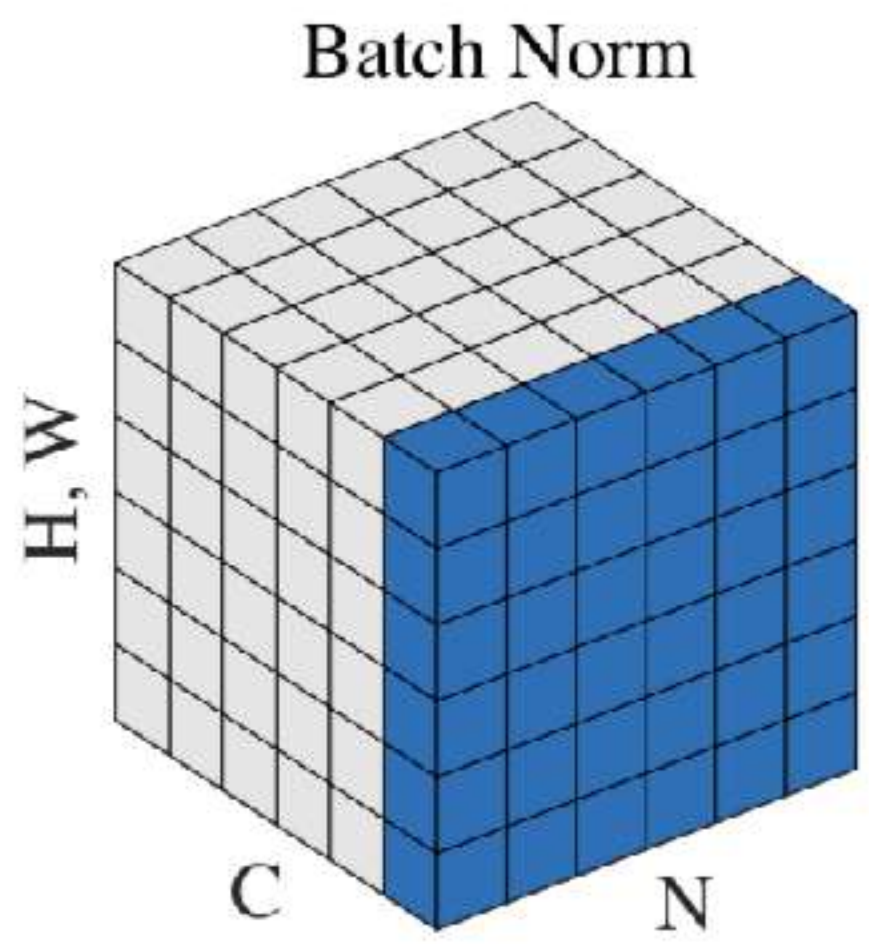


Image from Wu and He 2018. Reproduced for educational purposes.

Skipped in class
(outside of scope)

Other normalization techniques

Layer Normalization

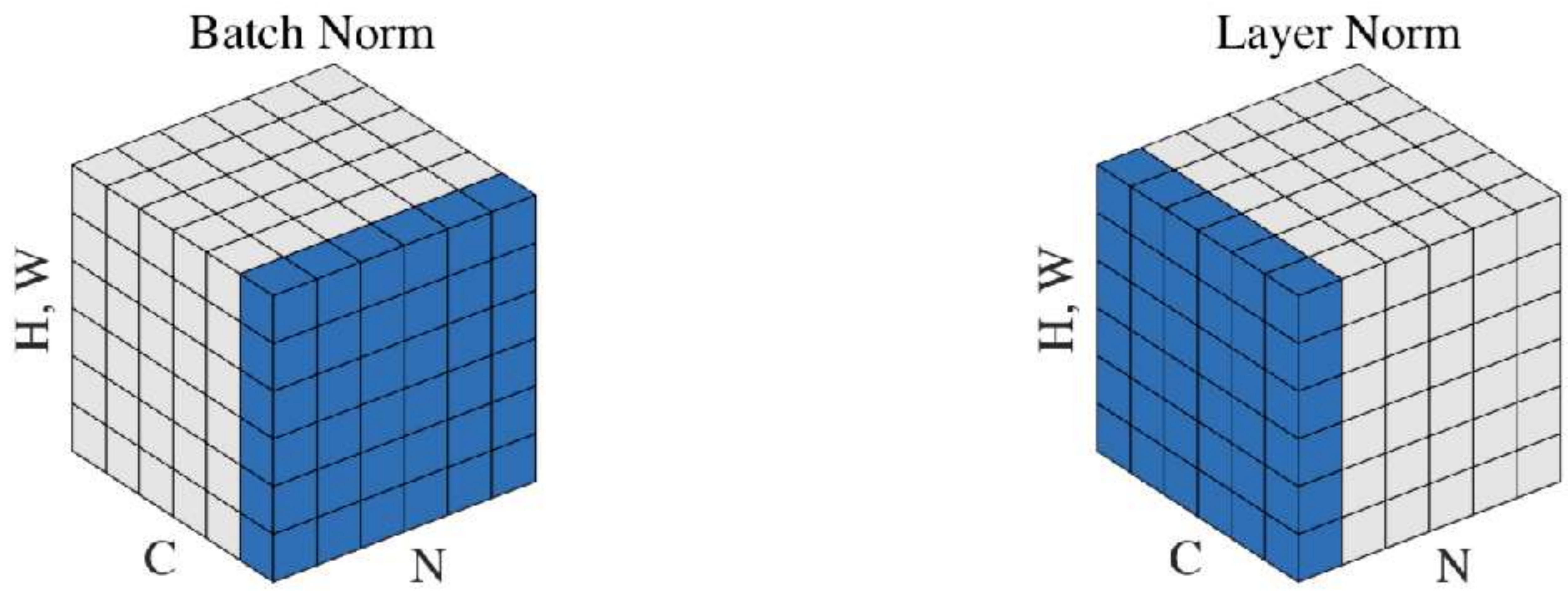


Image from Wu and He 2018. Reproduced for educational purposes.

Skipped in class
(outside of scope)

Other normalization techniques

Layer Normalization

Batch Normalization for
fully-connected networks

$\mathbf{x}: N \times D$

Normalize

$\boldsymbol{\mu}, \boldsymbol{\sigma}: 1 \times D$

$\gamma, \beta: 1 \times D$

$y = \gamma(x-\mu)/\sigma+\beta$

Layer Normalization for
fully-connected networks
Same behavior at train and test!
Can be used in recurrent networks

$\mathbf{x}: N \times D$

Normalize

$\boldsymbol{\mu}, \boldsymbol{\sigma}: N \times 1$

$\gamma, \beta: 1 \times D$

$y = \gamma(x-\mu)/\sigma+\beta$

Image from Wu and He 2018. Reproduced for educational purposes.

Skipped in class (outside of scope)

Other normalization techniques

Instance Normalization

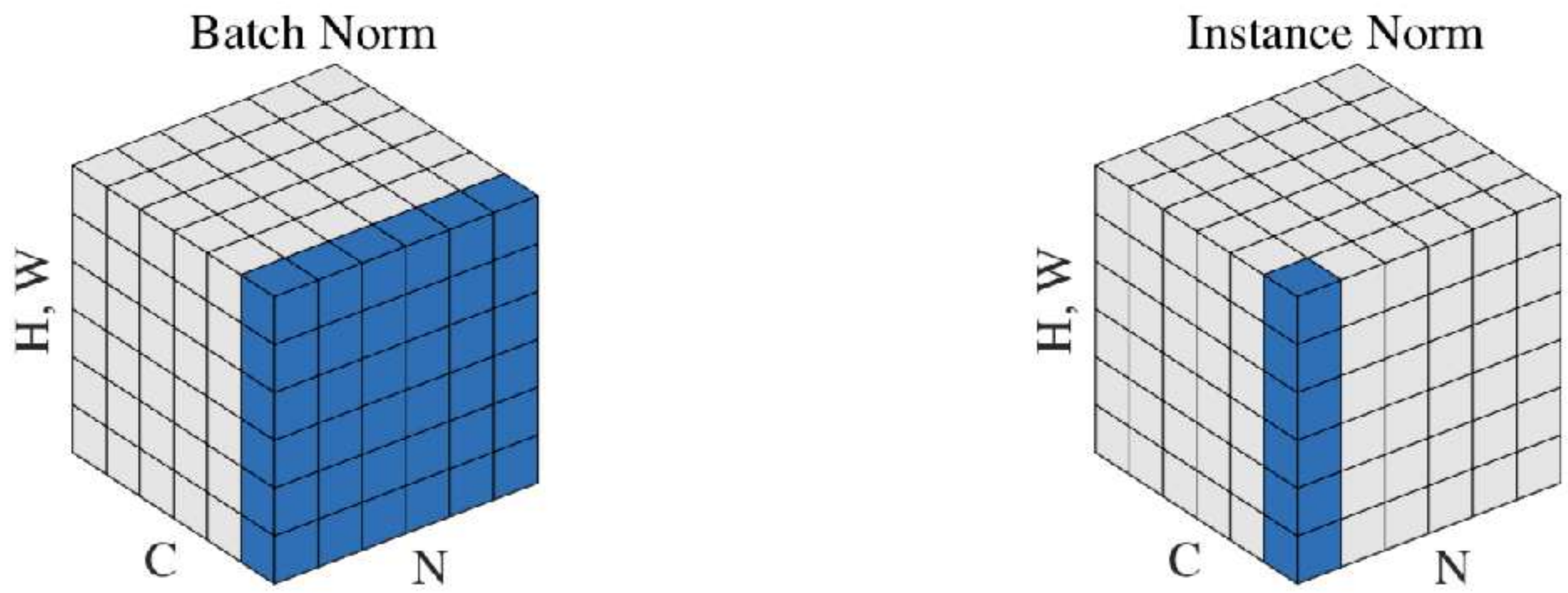


Image from Wu and He 2018. Reproduced for educational purposes.

Skipped in class (outside of scope)

Other normalization techniques

Instance Normalization

Batch Normalization for convolutional networks

$$\mathbf{x}: \; N \times C \times H \times W$$

Normalize

$$\mu, \sigma: \; 1 \times C \times 1 \times 1$$

$$\gamma, \beta: \; 1 \times C \times 1 \times 1$$

$$y = \gamma(x - \mu)/\sigma + \beta$$

Instance Normalization for convolutional networks
Same behavior at train / test!

$$\mathbf{x}: \; N \times C \times H \times W$$

Normalize

$$\mu, \sigma: \; N \times C \times 1 \times 1$$

$$\gamma, \beta: \; 1 \times C \times 1 \times 1$$

$$y = \gamma(x - \mu)/\sigma + \beta$$

Ulyanov et al, Improved Texture Networks: Maximizing Quality and Diversity in Feed-forward Stylization and Texture Synthesis, CVPR 2017

THE UNIVERSITY OF BRITISH COLUMBIA

Skipped in class (outside of scope)

Other normalization techniques

Group Normalization

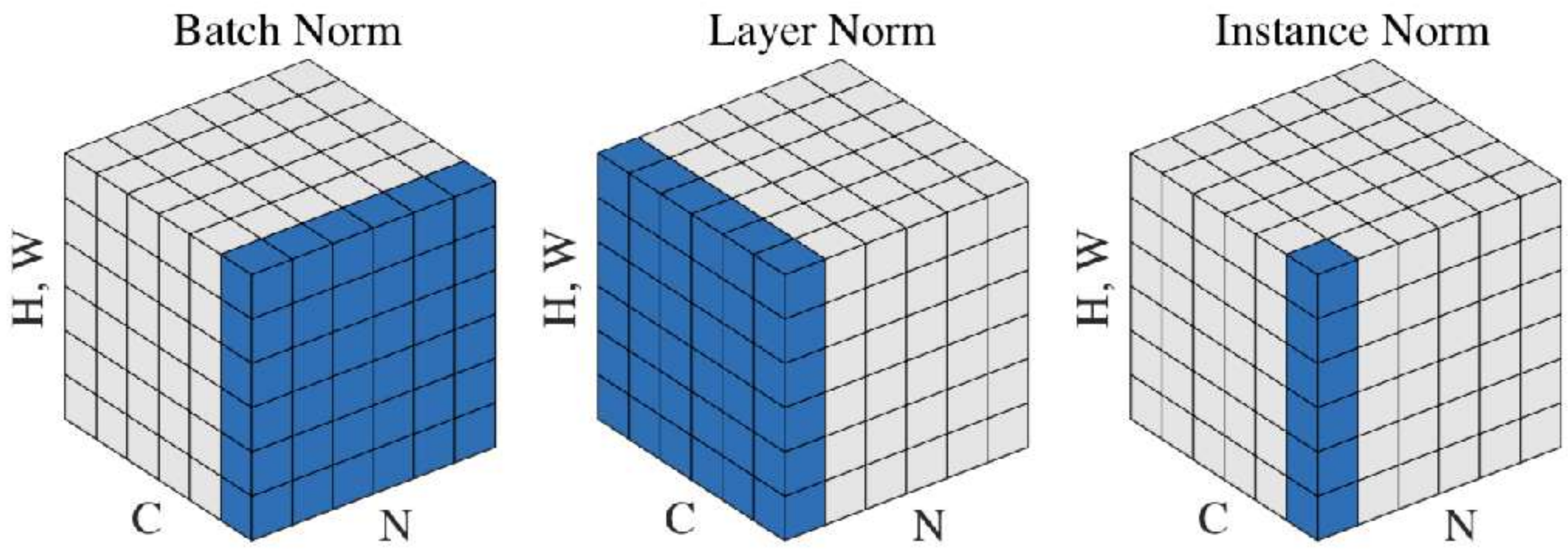


Image from Wu and He 2018. Reproduced for educational purposes.

Skipped in class (outside of scope)

Other normalization techniques

Group Normalization

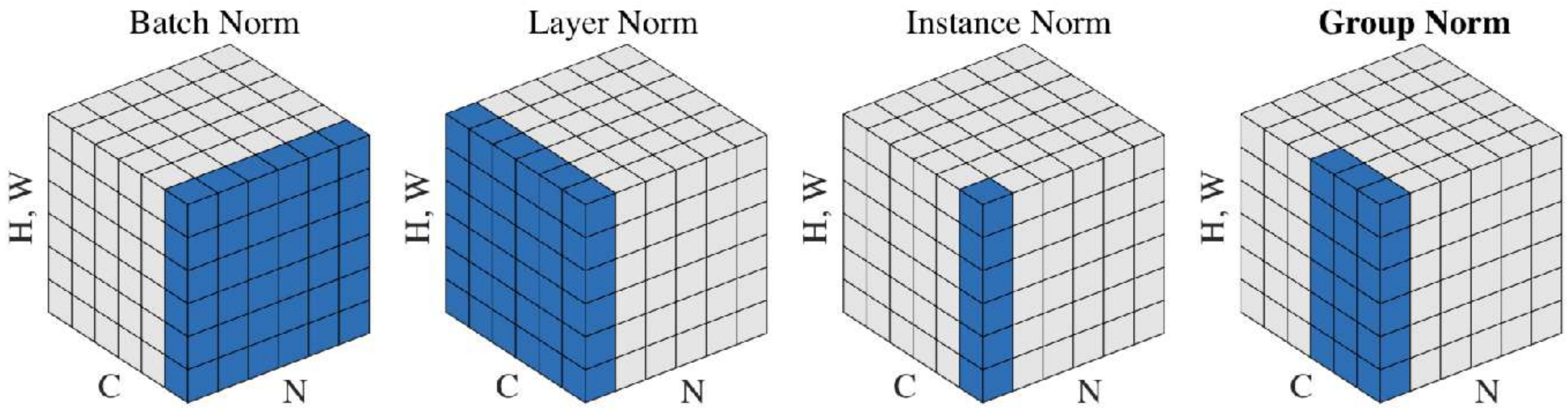
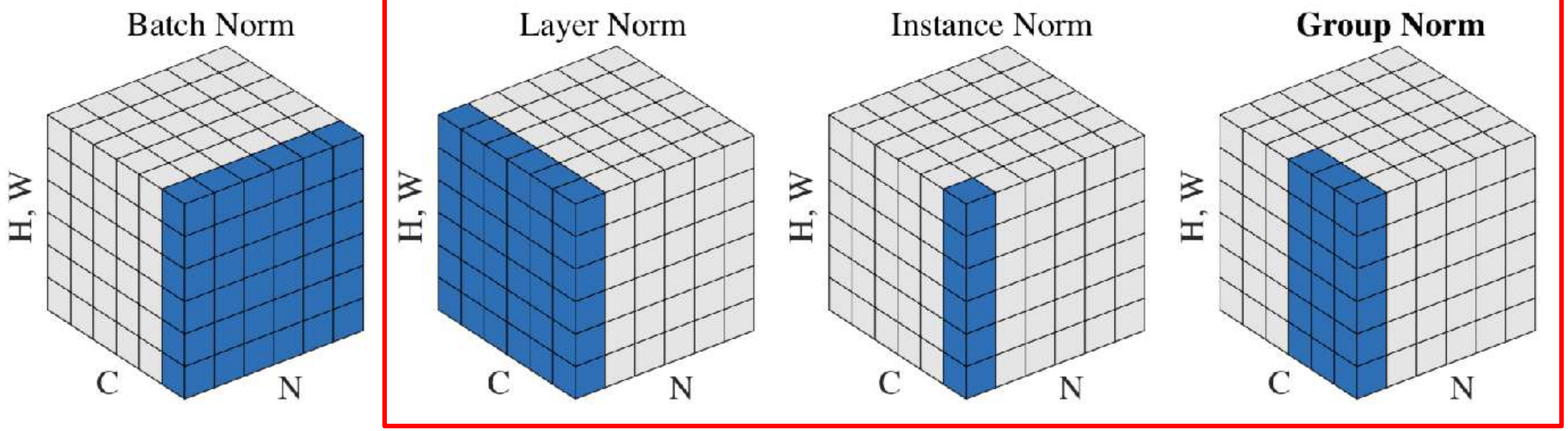


Image from Wu and He 2018. Reproduced for educational purposes.

Skipped in class
(outside of scope)

Other normalization techniques

Group Normalization



No train/test-time differences.

Much preferred in my opinion.

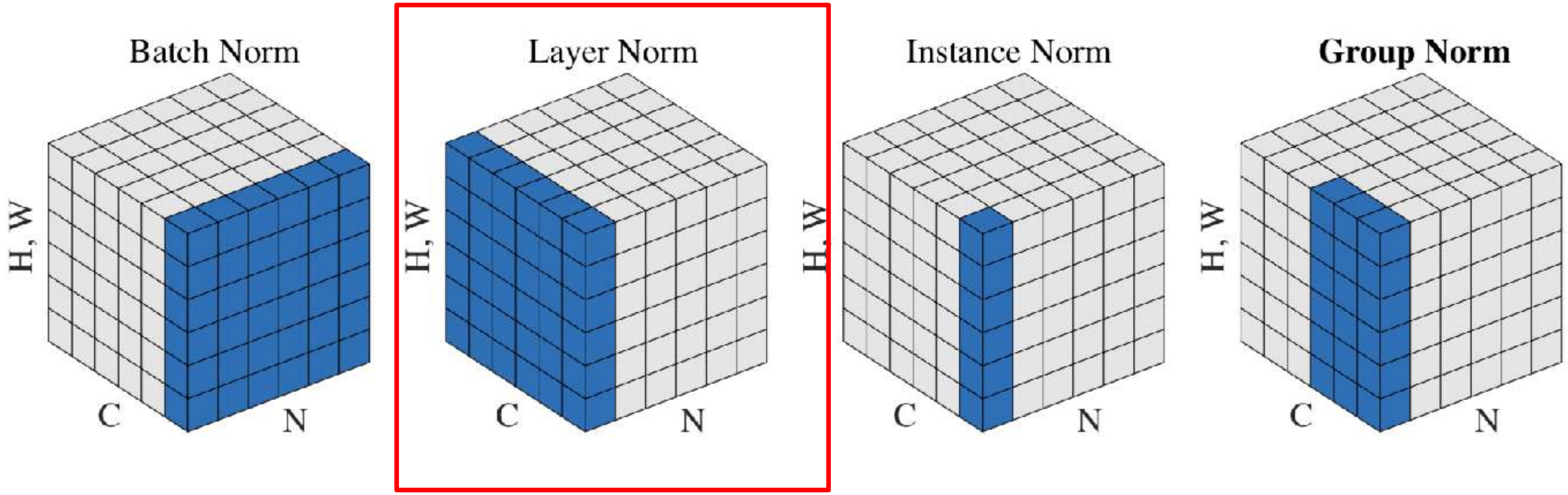
Image from Wu and He 2018. Reproduced for educational purposes.

THE UNIVERSITY OF BRITISH COLUMBIA

Skipped in class (outside of scope)

Other normalization techniques

Group Normalization



Can be implemented using PyTorch's Group norm.

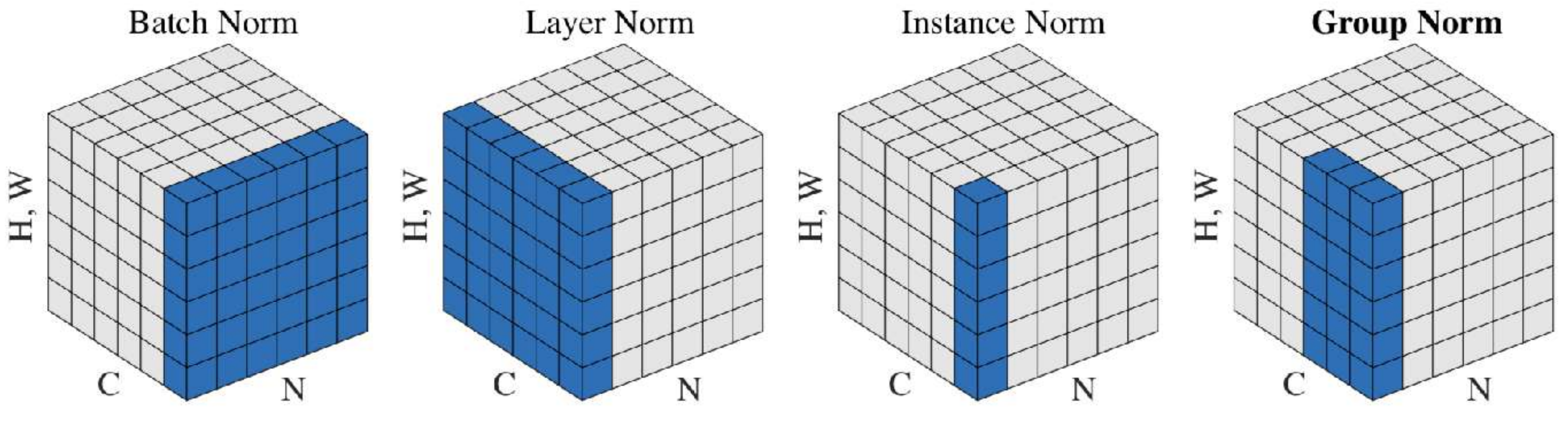
Image from Wu and He 2018. Reproduced for educational purposes.

THE UNIVERSITY OF BRITISH COLUMBIA

Skipped in class (outside of scope)

Other normalization techniques

Group Normalization



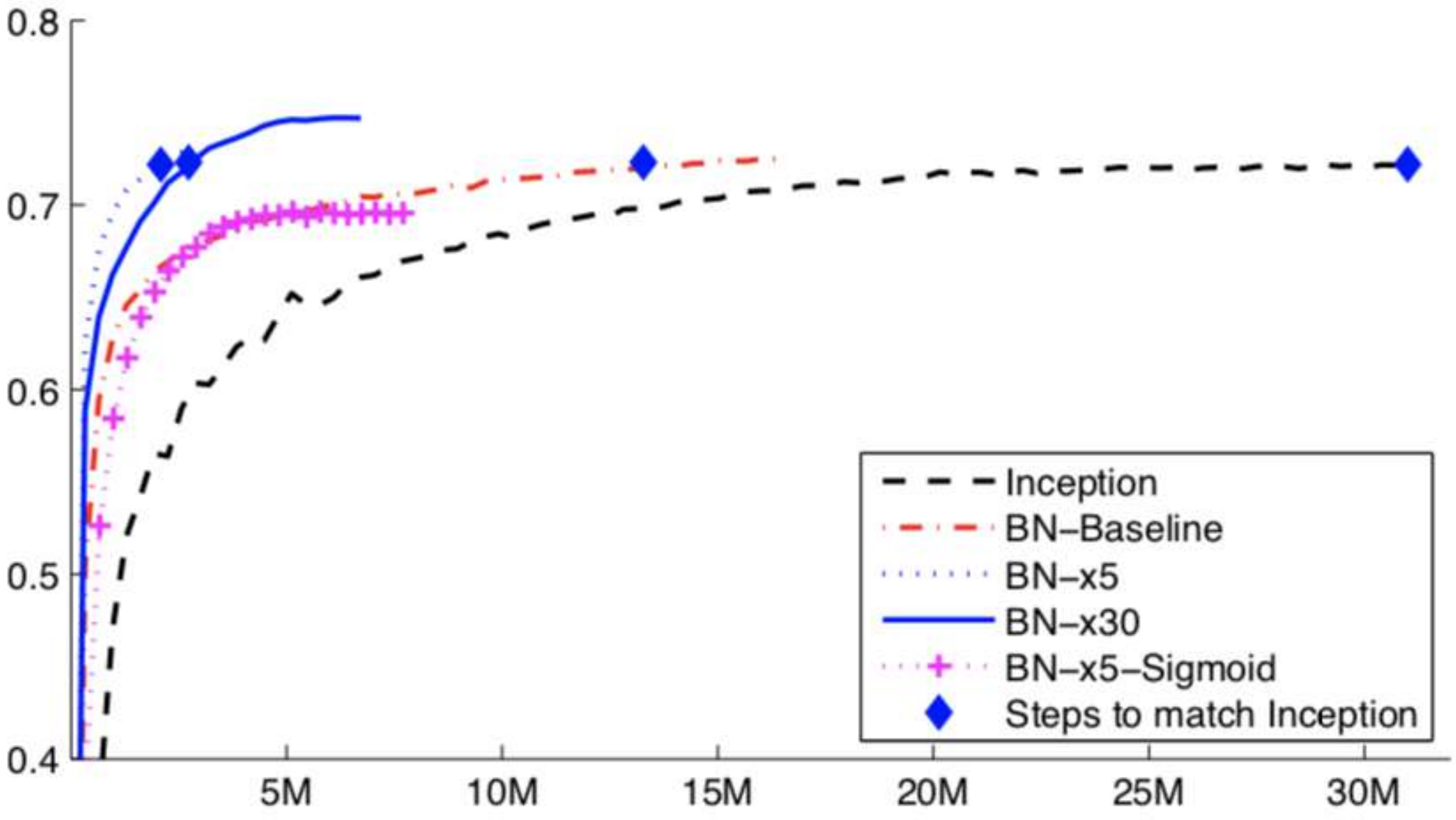
Choice of normalization should be data dependent

Image from Wu and He 2018. Reproduced for educational purposes.

By the way… with normalization something else also happens

[Ioffe and Szegedy, 2015]

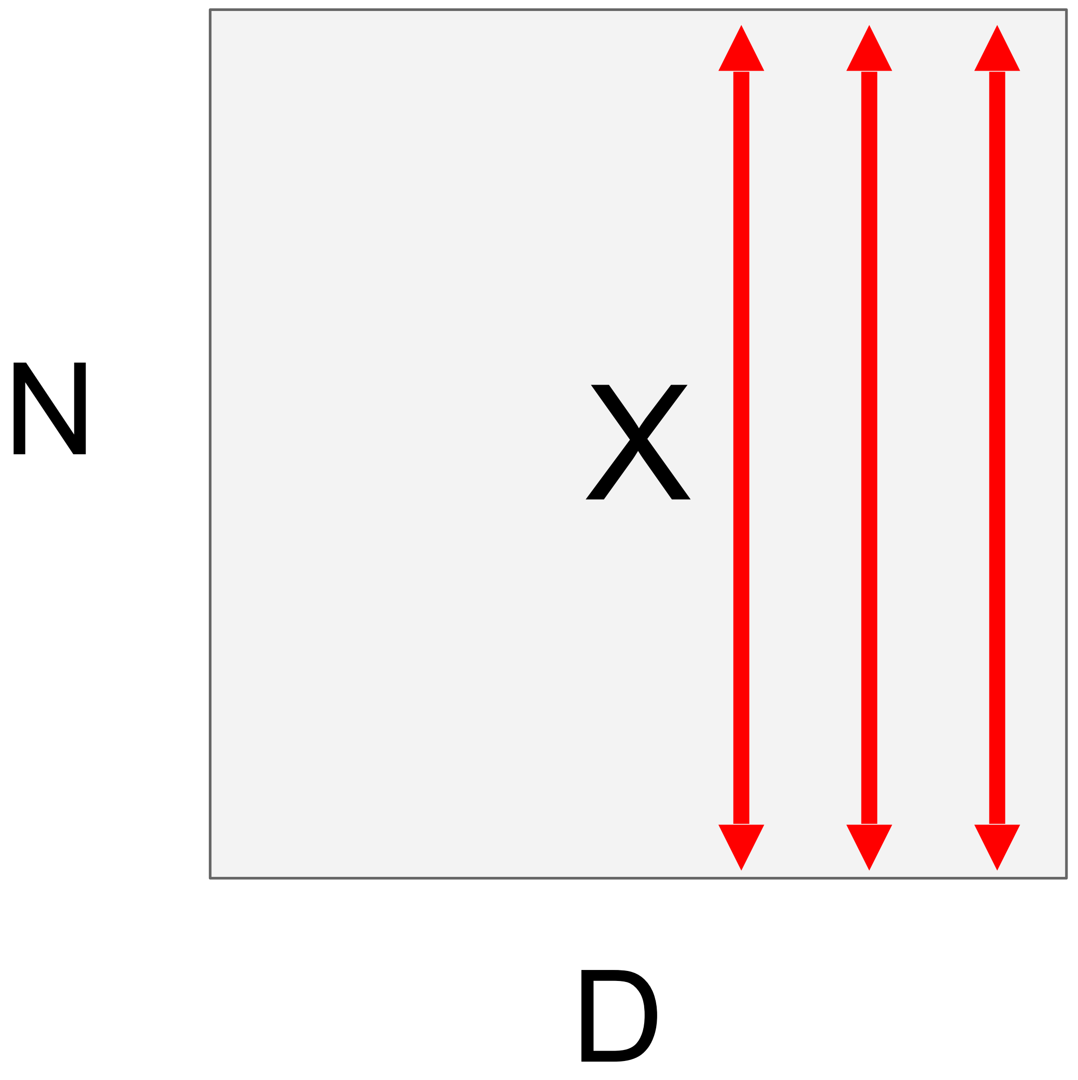
Batch normalization



[Ioffe and Szegedy, 2015]

Batch normalization

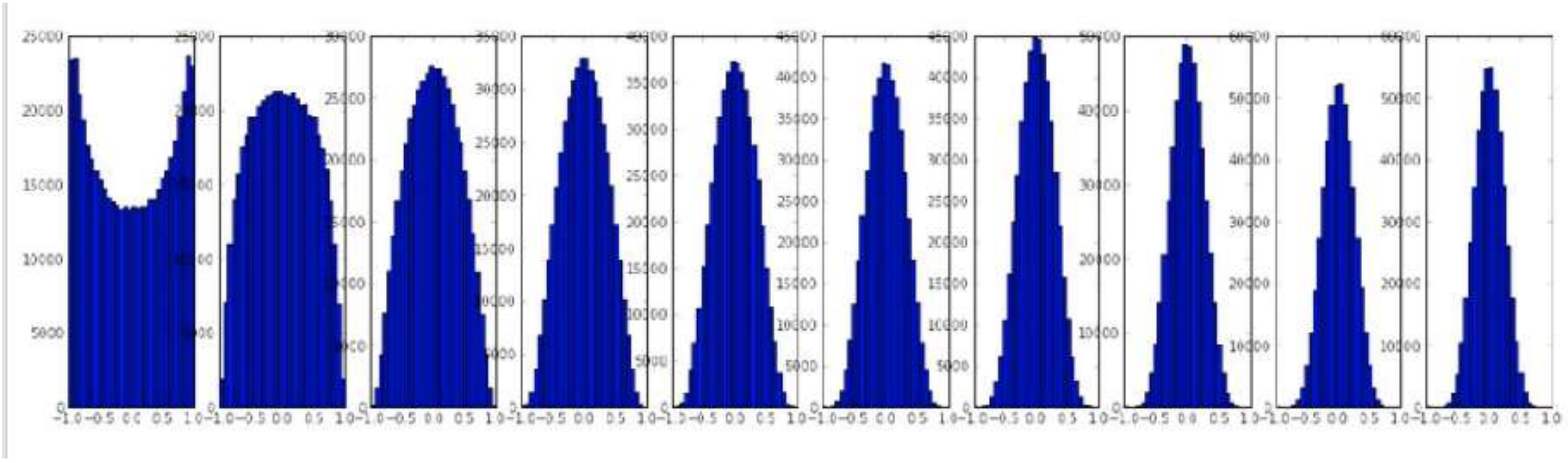
Recall...



1. compute the empirical mean and variance independently for each dimension.

2. Normalize

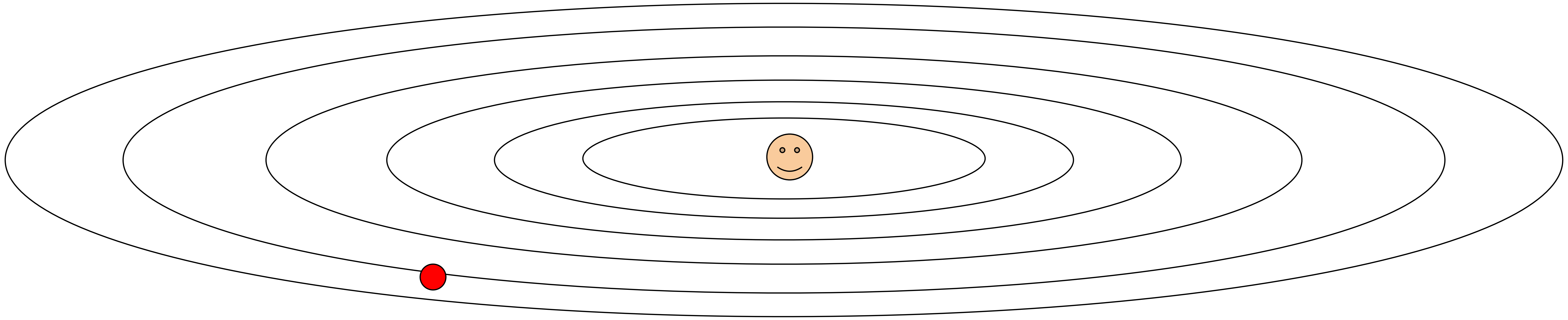
$$\widehat{x}^{(k)} = \frac{x^{(k)} - \mathrm{E}[x^{(k)}]}{\sqrt{\mathrm{Var}[x^{(k)}]}}$$



[Ioffe and Szegedy, 2015]

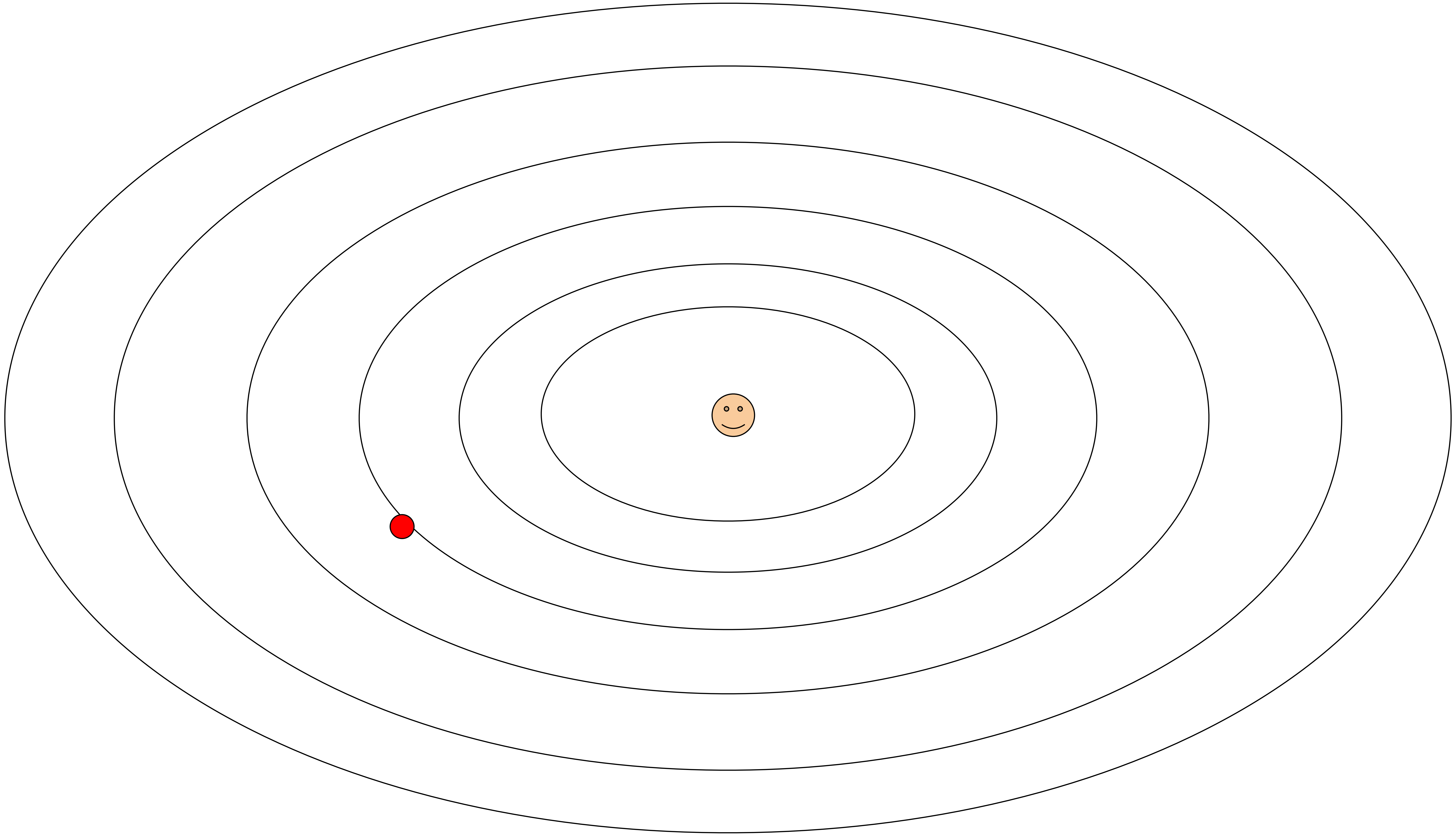
Batch normalization

This imbalance between dimensions is the problem



[Ioffe and Szegedy, 2015]

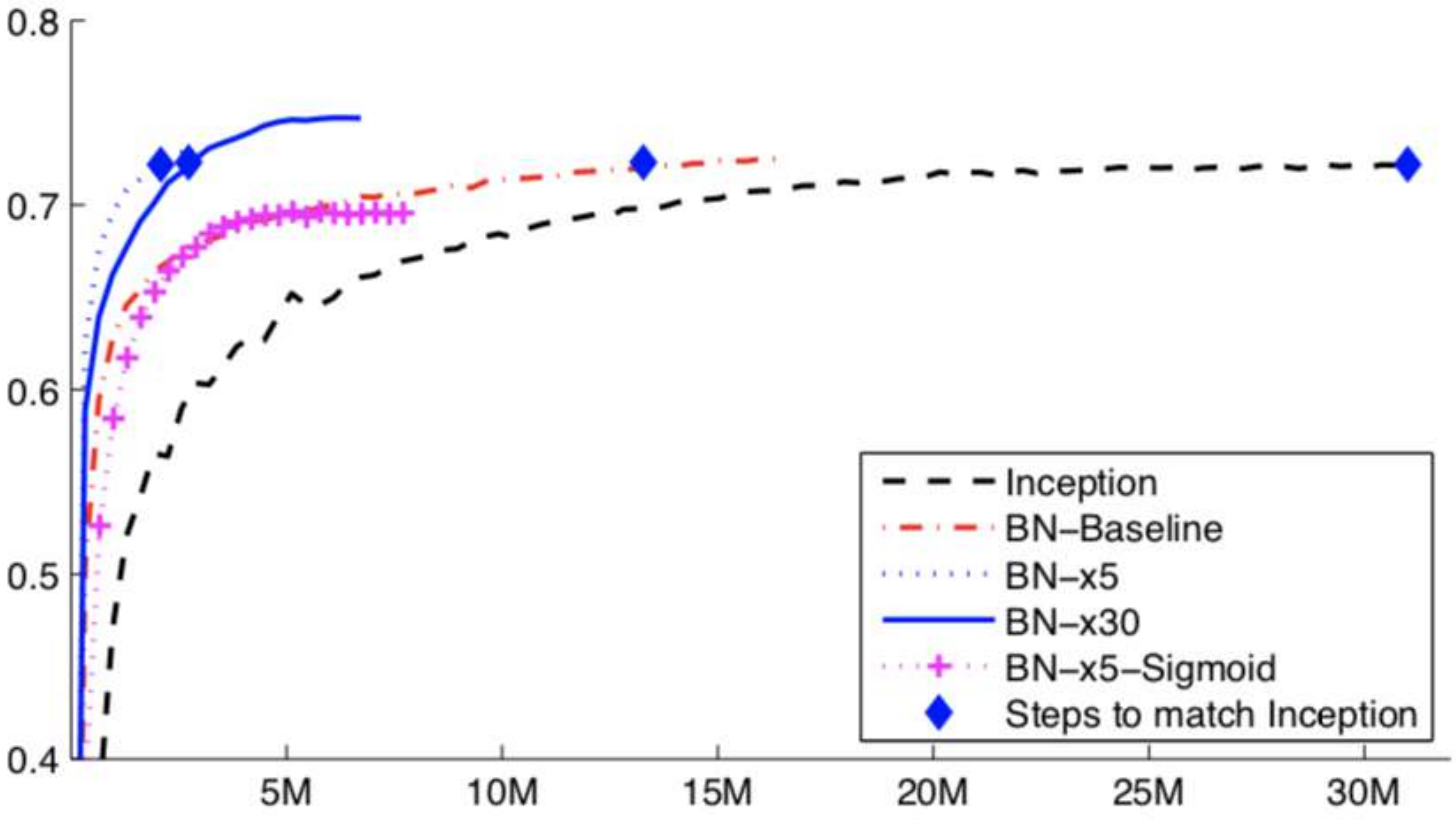
Batch normalization



Let's artificially make it like this!

[Ioffe and Szegedy, 2015]

Batch normalization

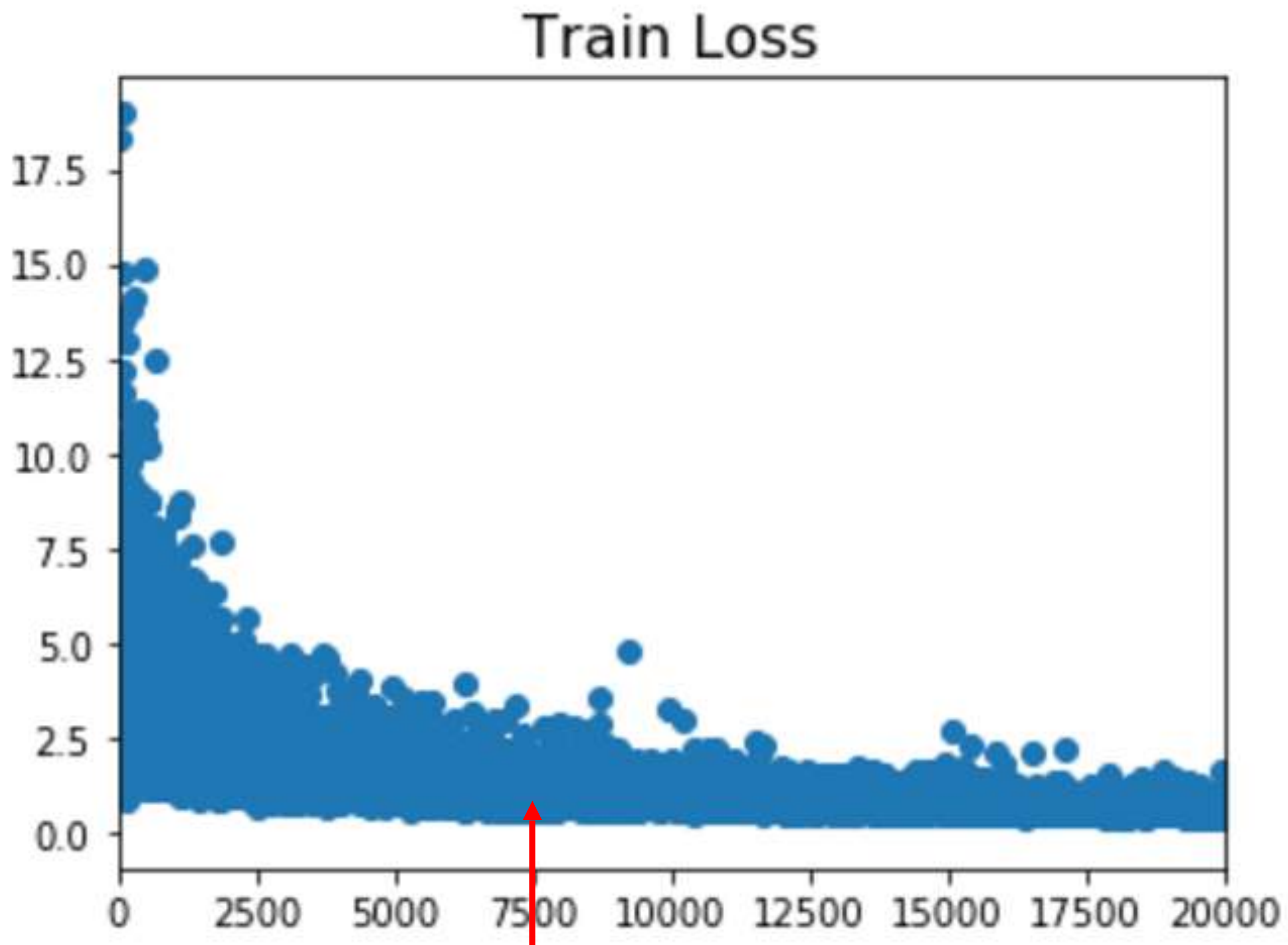


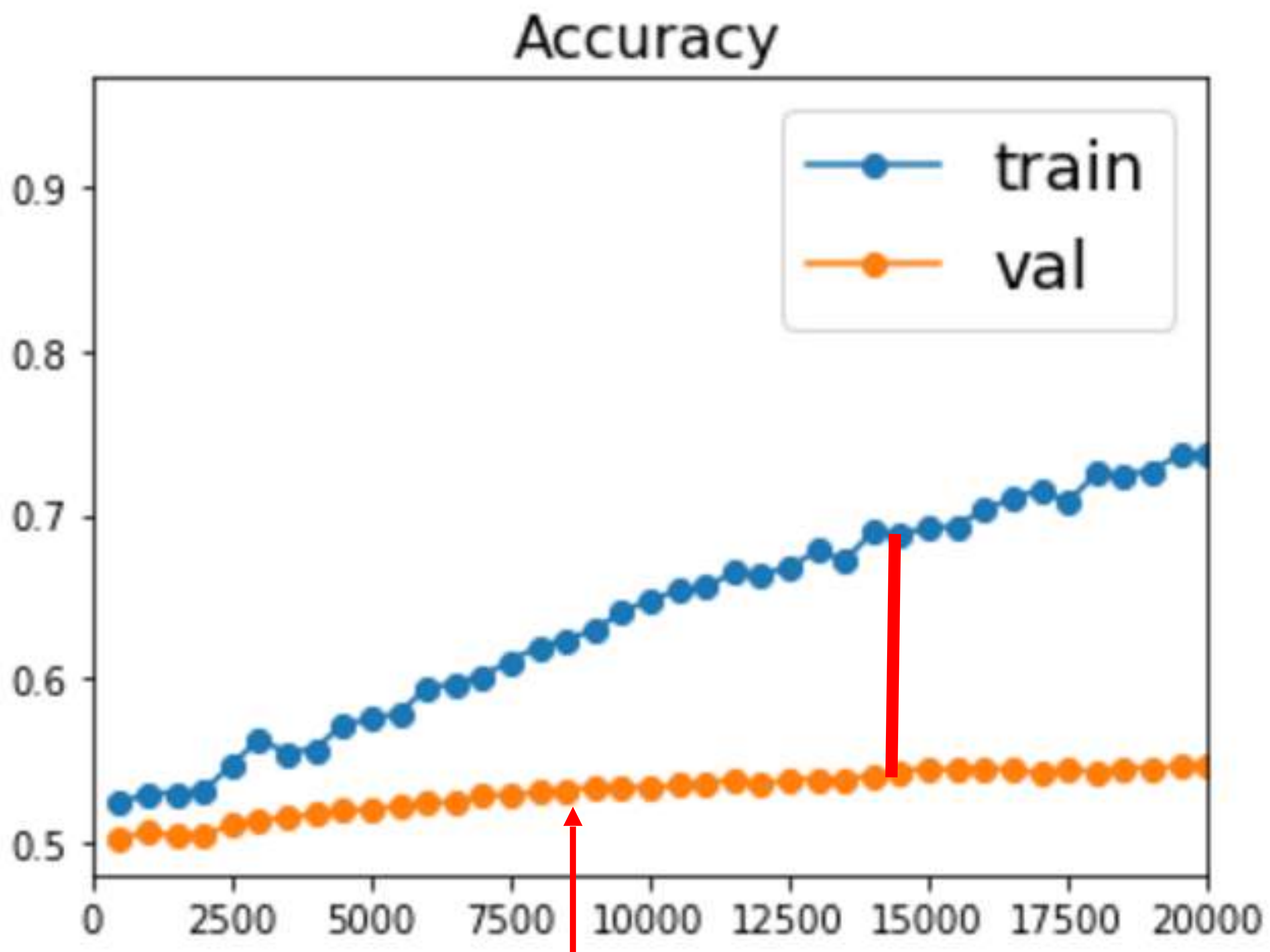
Based on slides for [Stanford cs231n]() by Li, Jonson, and Young. Modified and reused with permission

Preventing overfitting

Beyond training loss

Recall the other problem



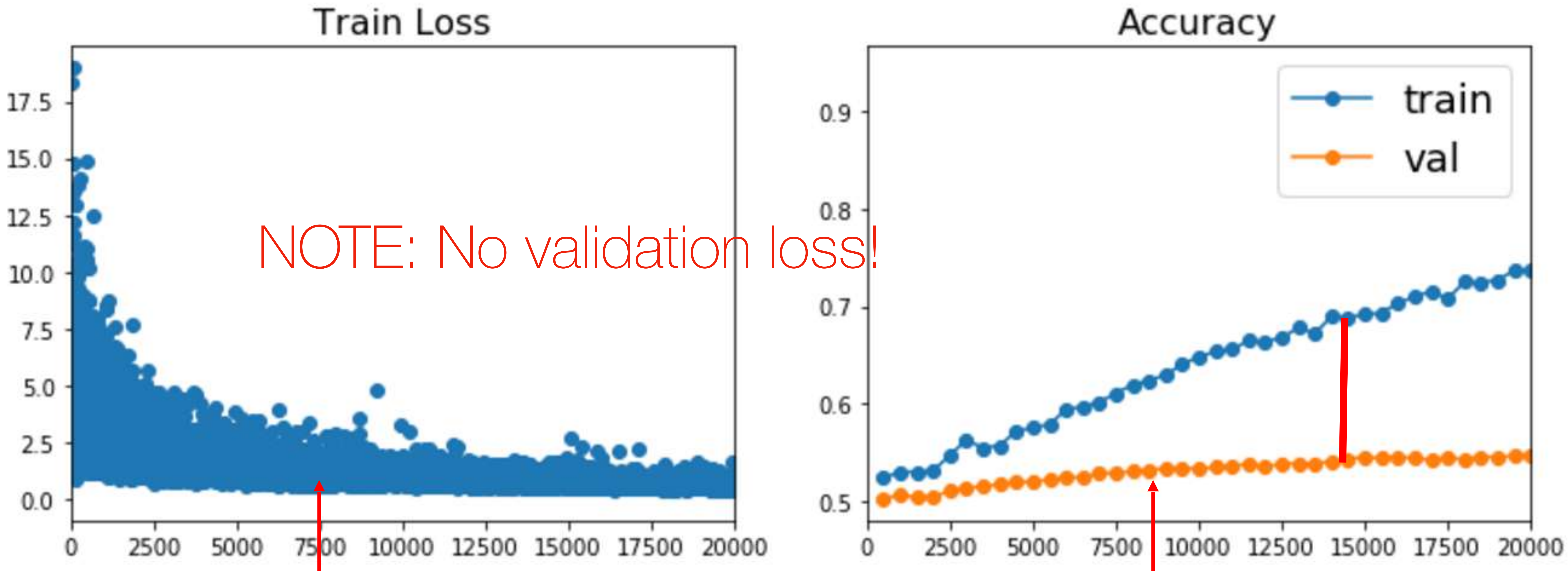


Better optimization algorithms help reduce training loss

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

Beyond training loss

Recall the other problem



NOTE: No validation loss!

Better optimization algorithms help reduce training loss

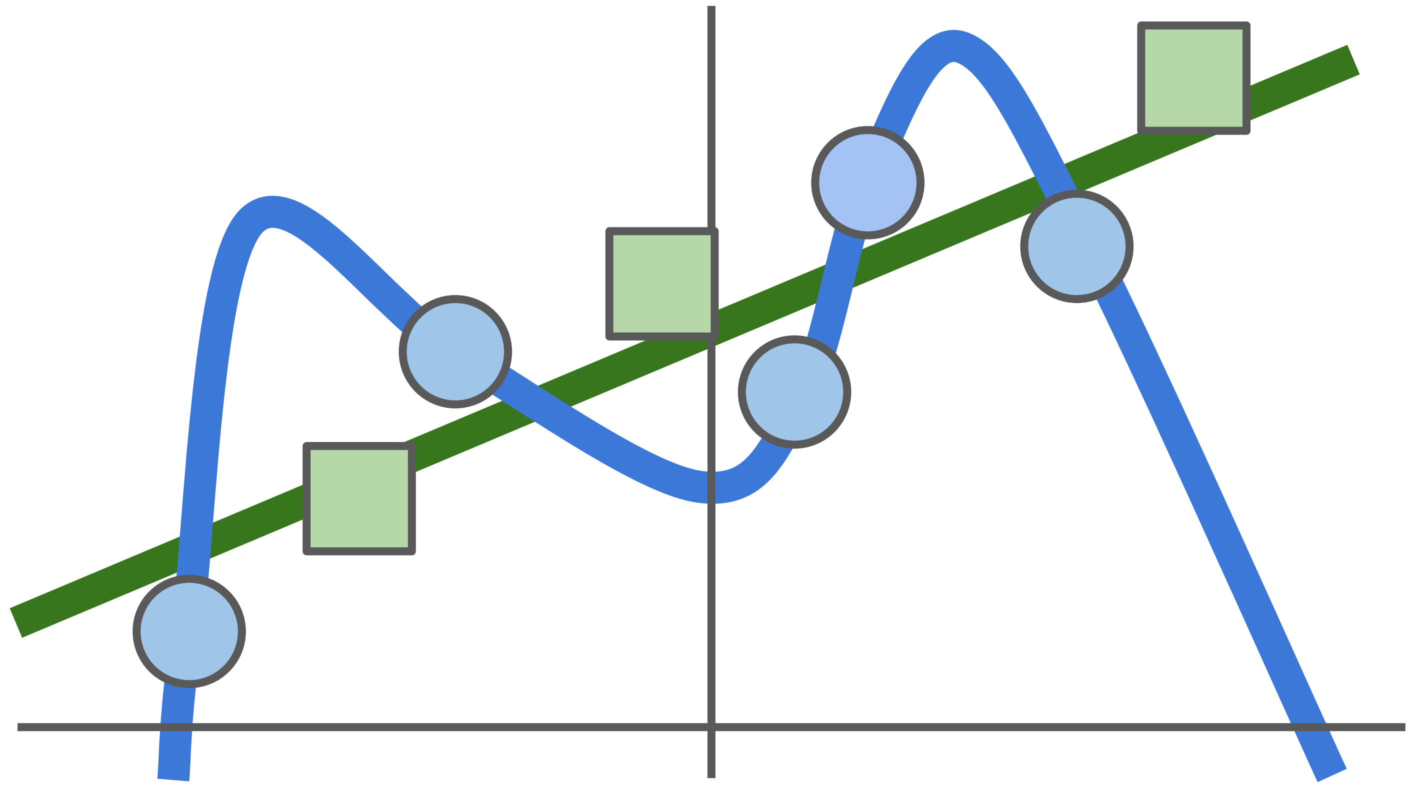
But we really care about error on new data - how to reduce the gap?

A typical approach to overfitting

Regularization

$$L(W) = \underbrace{\frac{1}{N}\sum_{i=1}^{N} L_i(f(x_i, W), y_i)}_{\text{Data loss}} + \underbrace{\lambda R(W)}_{\text{Regularization}}$$

Data loss: Model predictions should match training data



Regularization: Model should be “simple”, so it works on test data

$$\|\mathbf{W}\|_2^2$$

Occam’s Razor:
“Among competing hypotheses, the simplest is the best”
William of Ockham, 1285 - 1347

Common regularizers

$$L = \frac{1}{N}\sum_{i=1}^{N}\sum_{j\neq y_i}\max(0, f(x_i;W)_j - f(x_i;W)_{y_i} + 1) + \boxed{\lambda R(W)}$$

In common use:

L2 regularization $R(W) = \sum_k \sum_l W_{k,l}^2$ (Weight decay)

L1 regularization $R(W) = \sum_k \sum_l |W_{k,l}|$

Elastic net (L1 + L2) $R(W) = \sum_k \sum_l \beta W_{k,l}^2 + |W_{k,l}|$

Common regularizers

$$L = \frac{1}{N}\sum_{i=1}^{N}\sum_{j\neq y_i}\max(0, f(x_i;W)_j - f(x_i;W)_{y_i} + 1) + \boxed{\lambda R(W)}$$

In common use:

L2 regularization $R(W) = \boxed{\sum_k \sum_l W_{k,l}^2 \quad \text{(Weight decay)}}$

L1 regularization $R(W) = \sum_k \sum_l |W_{k,l}|$

Elastic net (L1 + L2) $R(W) = \sum_k \sum_l \beta W_{k,l}^2 + |W_{k,l}|$

Common regularizers

My personal warning against L2

$$L = \frac{1}{N}\sum_{i=1}^{N}\sum_{j\neq y_i}\max(0, f(x_i;W)_j - f(x_i;W)_{y_i} + 1) + \boxed{\lambda R(W)}$$

In common use:

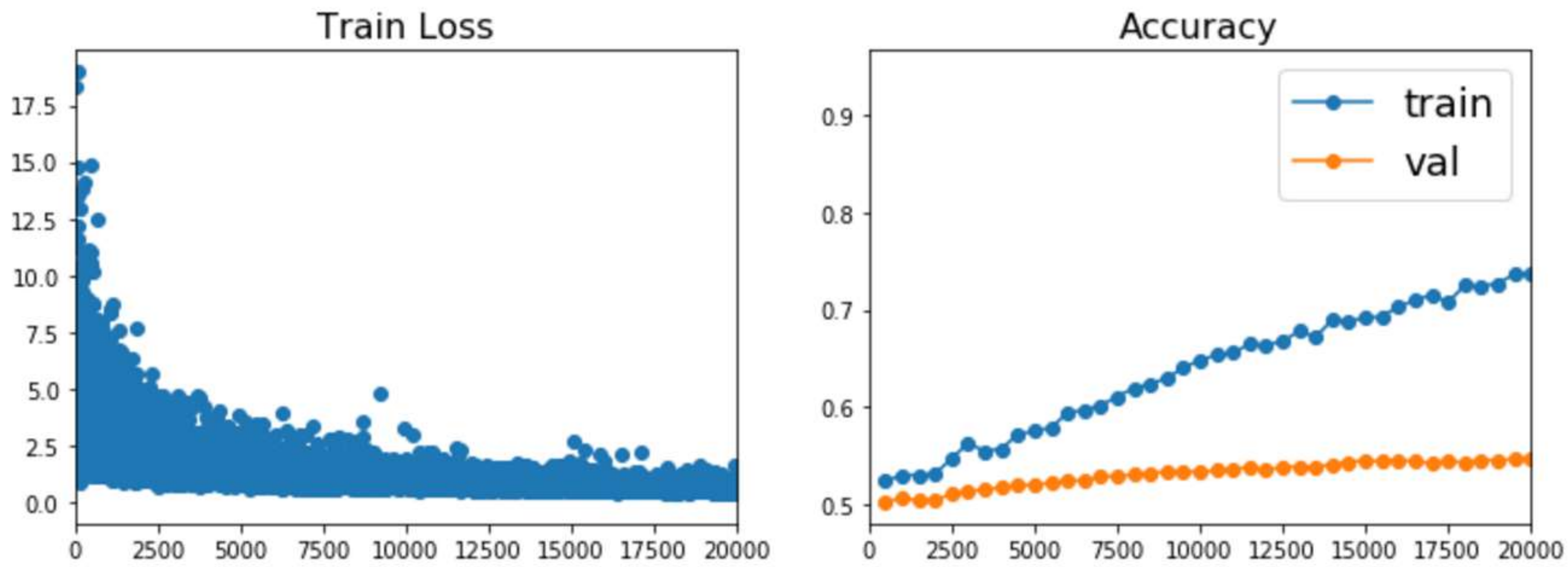
L2 regularization $R(W) = \boxed{\sum_k \sum_l W_{k,l}^2 \quad \text{(Weight decay)}}$

L1 regularization $R(W) = \sum_k \sum_l |W_{k,l}|$

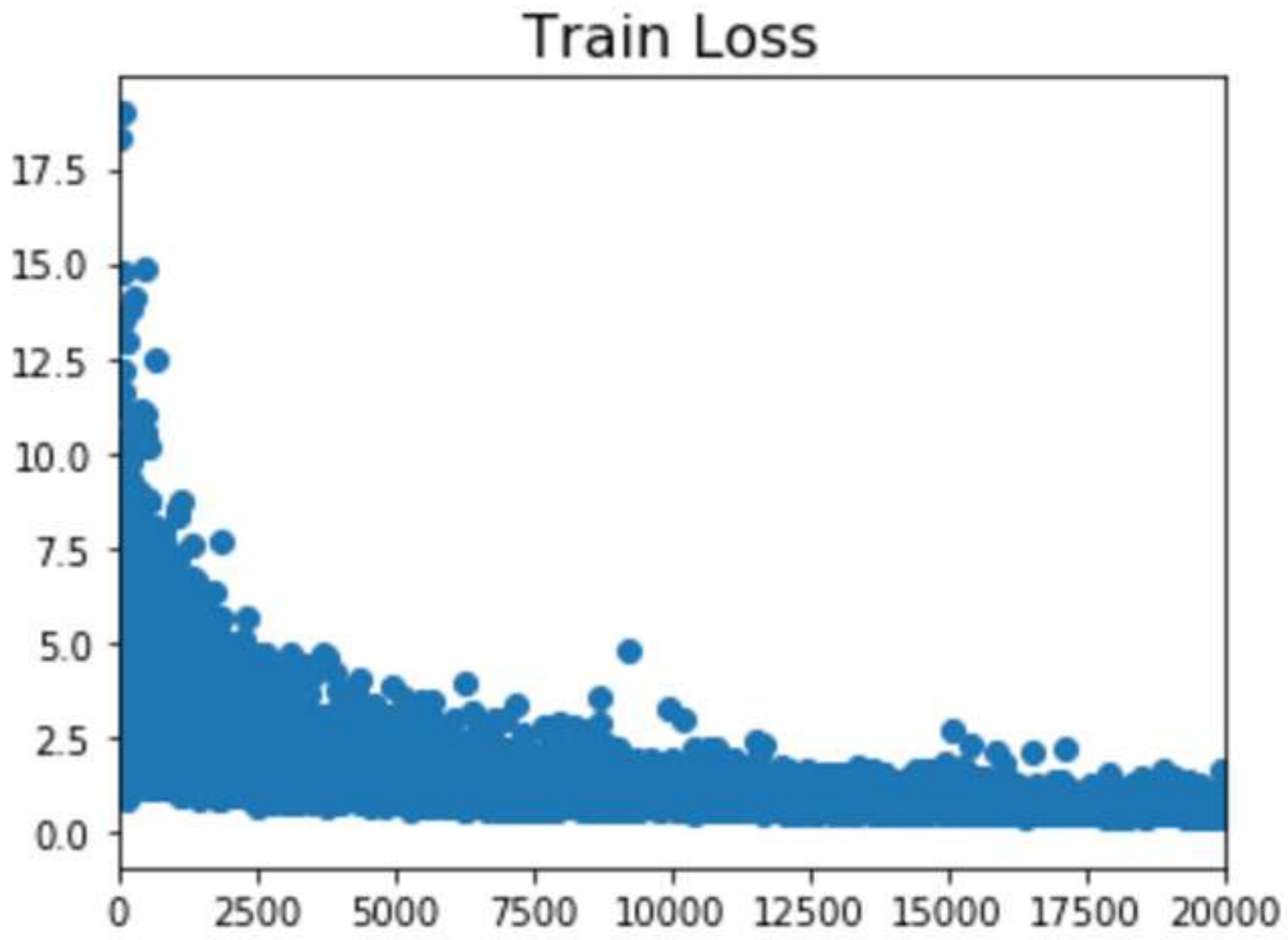
Ela

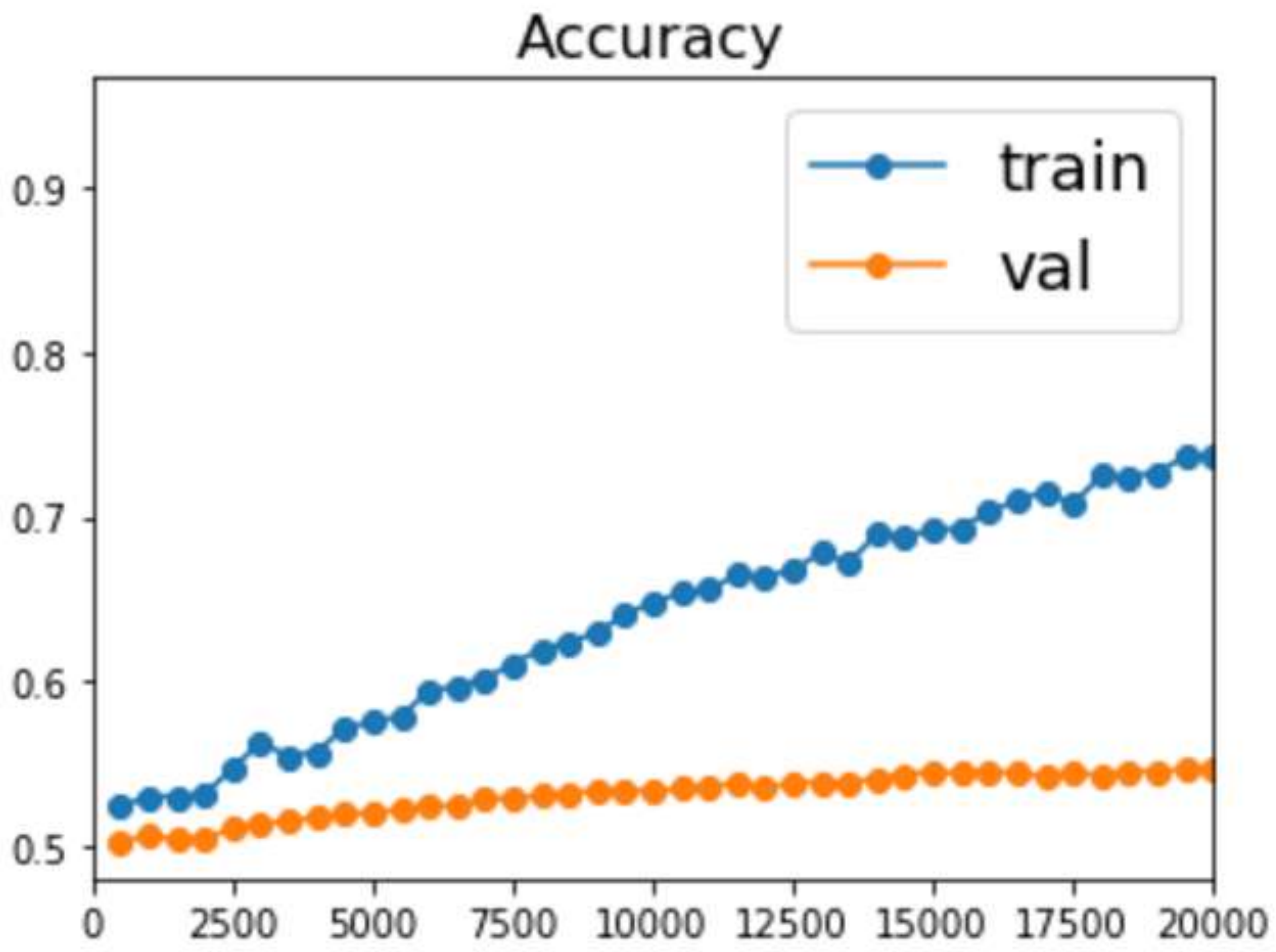
Laarhoven, 2017, "However, we show that L2 regularization has **no regularizing effect when combined with normalization**. Instead, regularization has an influence on the scale of weights, and thereby on the effective learning rate."

Why does this happen in the first place?



Why does this happen in the first place?



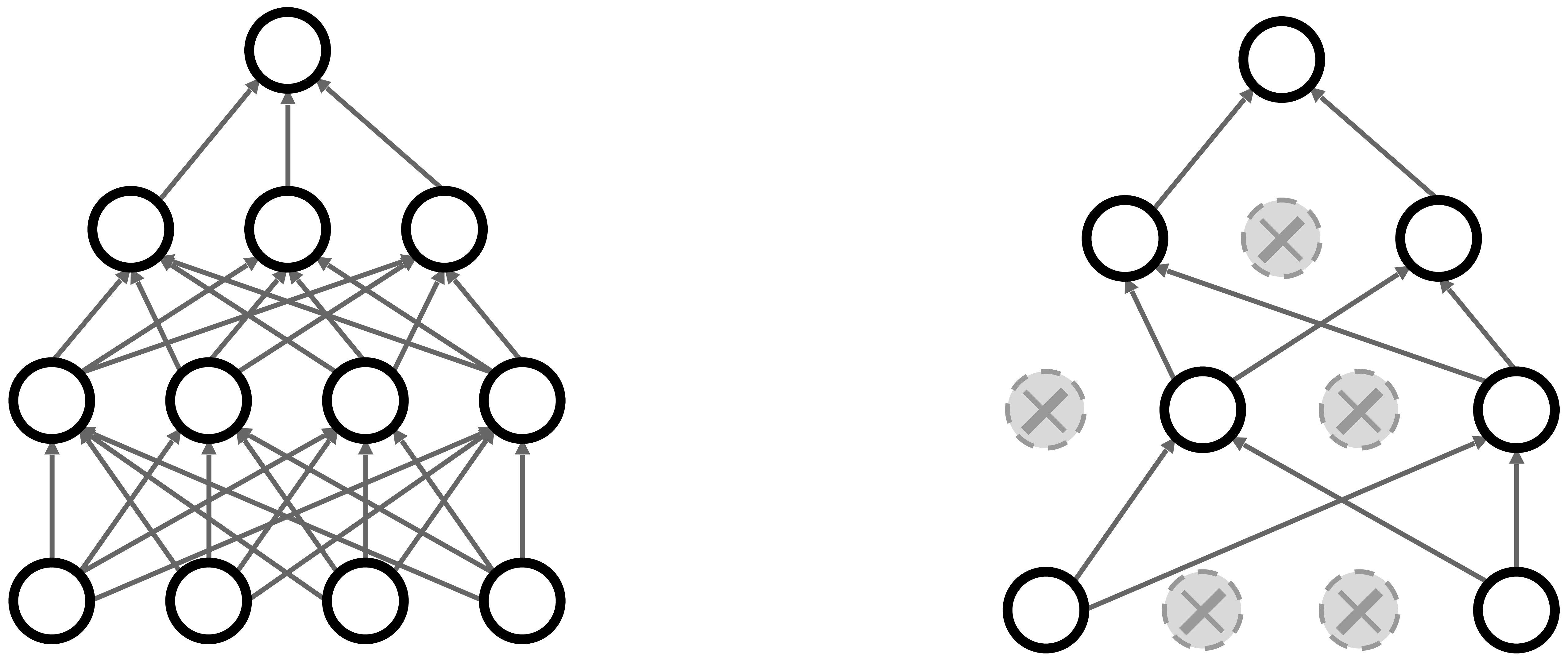


Can we somehow encode uncertainty in data?

Regularization: Dropout

Making it impossible to trust the data 100%

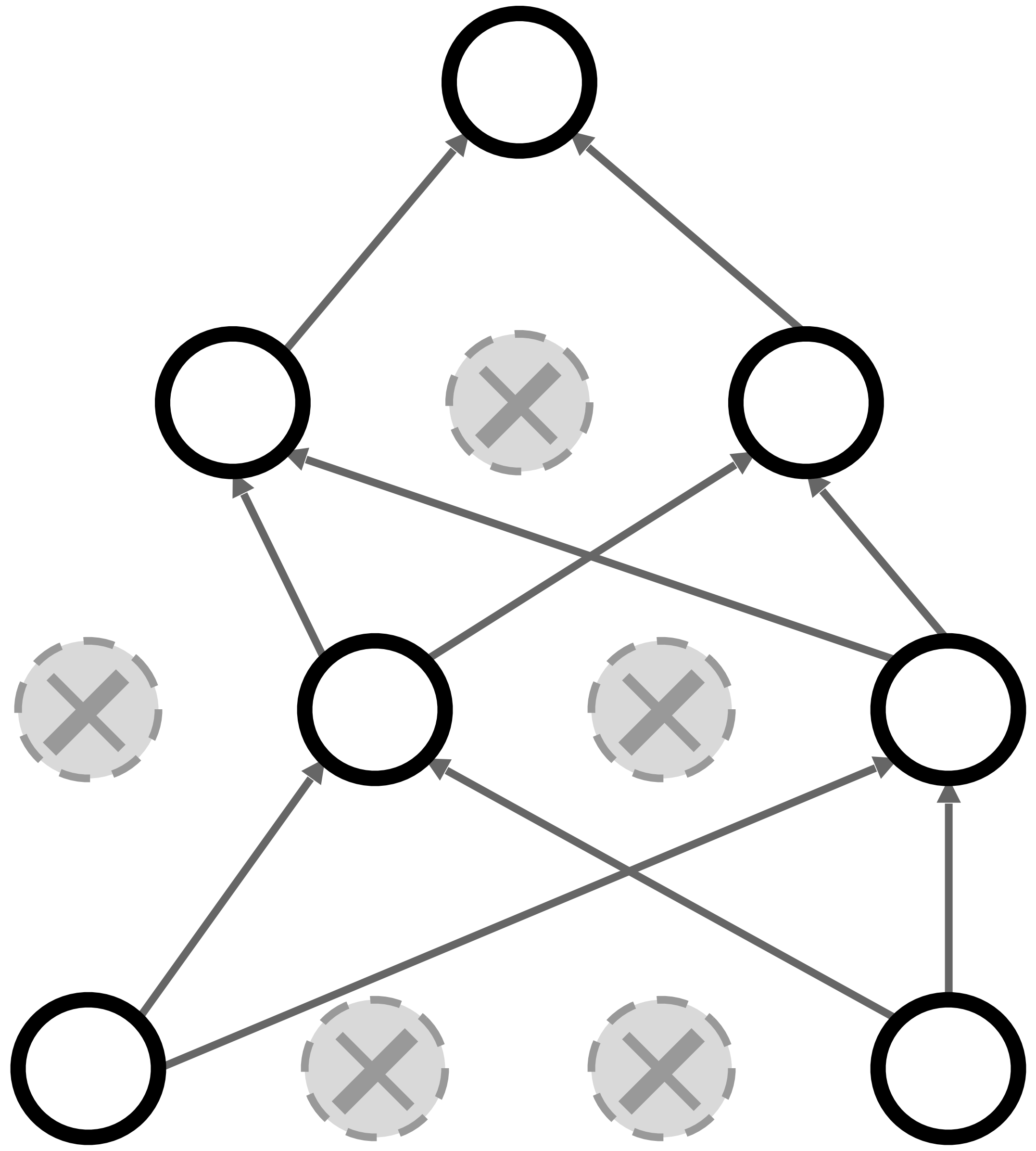
In each forward pass, randomly set some neurons to zero
Probability of dropping is a hyperparameter; 0.5 is common



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

Regularization: Dropout

Making it impossible to trust the data 100%



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

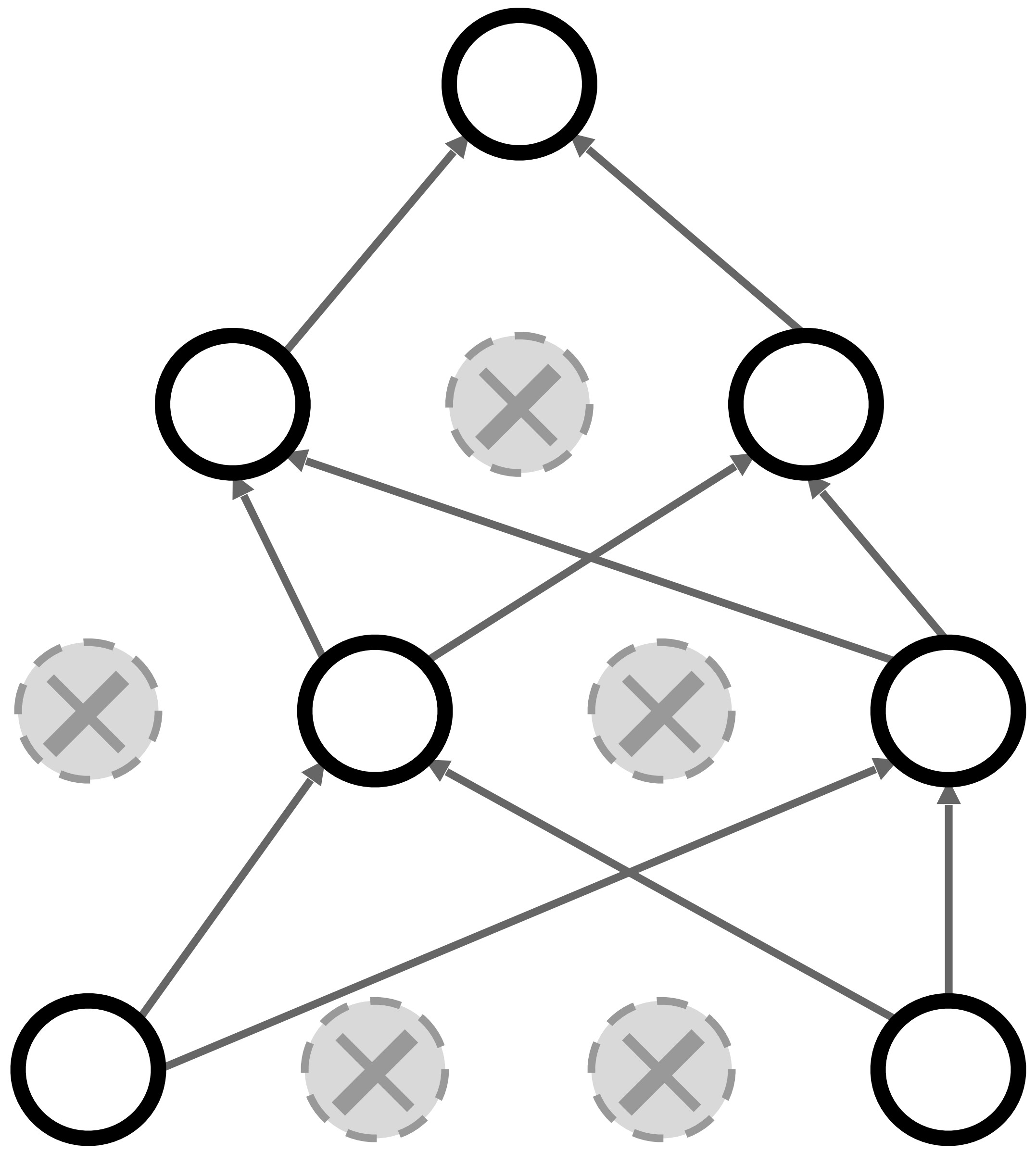
Srivastava et al, "Dropout: A simple way to prevent neural networks from overfitting", JMLR 2014

Based on slides for Stanford cs231n by Li, Jonson, and Young. Modified and reused with permission

Skipped in class (outside of scope)

Regularization: Dropout

Making it impossible to trust the data 100%



Another interpretation:

Dropout is training a large **ensemble** of models (that share parameters).

Each binary mask is one model

An FC layer with 4096 units has $2^{4096} \sim 10^{1233}$ possible masks!
Only $\sim 10^{82}$ atoms in the universe...

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

Skipped in class (outside of scope)

Regularization: Dropout at **test** time

Again the train / test gap

Output (label)

Input (image)

Random mask

Dropout makes our output random! $\boxed{y} = f_W(\boxed{x}, \boxed{z})$

Want to "average out" the randomness at test-time

$$y = f(x) = E_z\left[f(x, z)\right] = \int p(z) f(x, z) dz$$

But this integral seems hard …

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

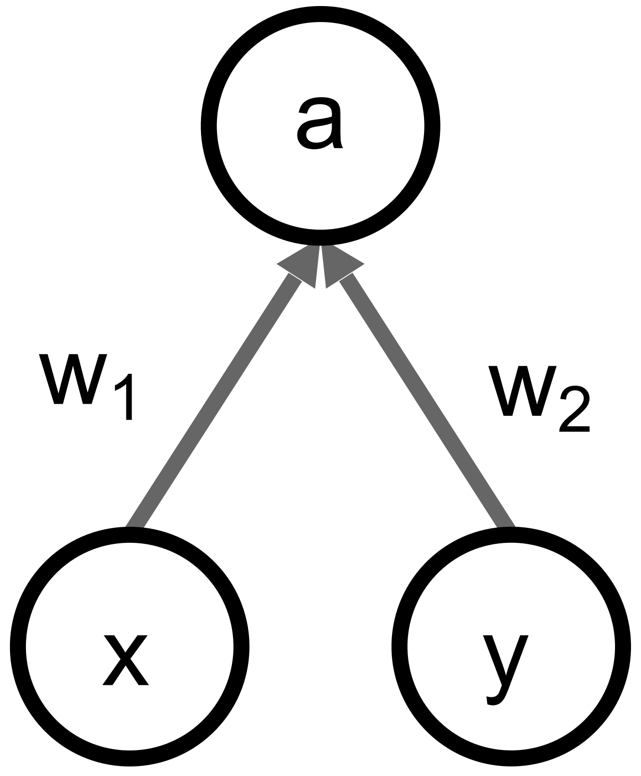
Skipped in class (outside of scope)

Regularization: Dropout at **test** time

An approximate solution

Want to approximate the integral $$y = f(x) = E_z\left[f(x,z)\right] = \int p(z) f(x,z) dz$$

Consider a single neuron.



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

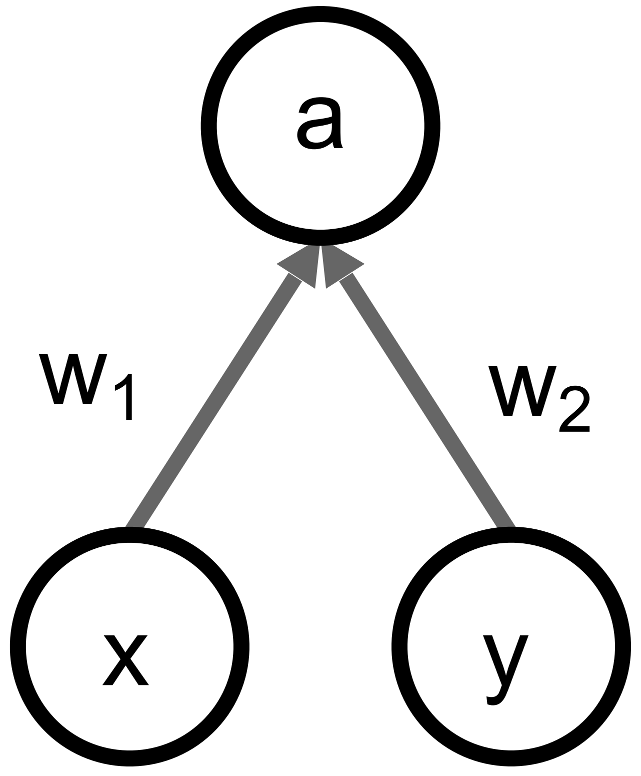
Skipped in class (outside of scope)

Regularization: Dropout at **test** time

An approximate solution

Want to approximate the integral $y = f(x) = E_z\left[f(x,z)\right] = \int p(z) f(x,z) dz$

Consider a single neuron.



At test time we have: $E\left[a\right] = w_1 x + w_2 y$

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

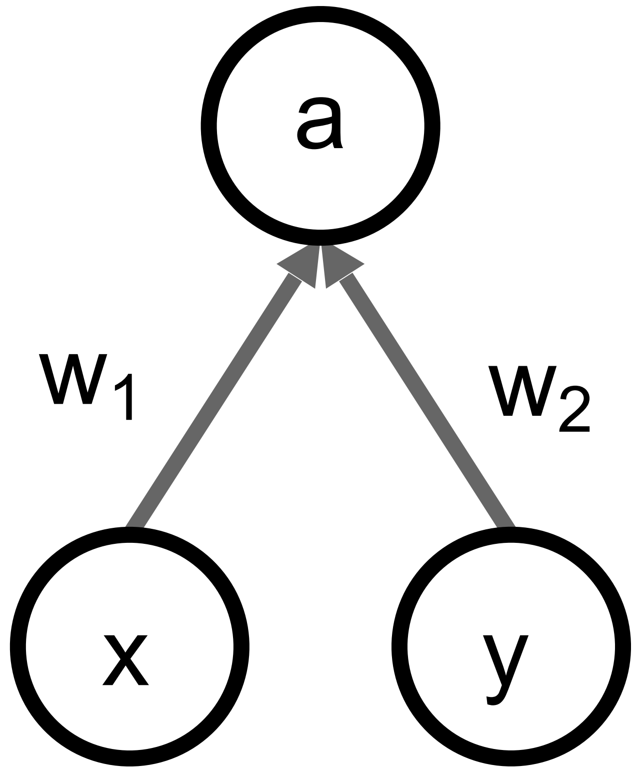
Skipped in class (outside of scope)

Regularization: Dropout at **test** time

An approximate solution

Want to approximate the integral $$y = f(x) = E_z\left[f(x,z)\right] = \int p(z) f(x,z) dz$$

Consider a single neuron.



At test time we have: $E\left[a\right] = w_1 x + w_2 y$

During training we have:
$$\begin{aligned} E\left[a\right] =& \frac{1}{4}(w_1 x + w_2 y) + \frac{1}{4}(w_1 x + 0y) \\ &+ \frac{1}{4}(0x + 0y) + \frac{1}{4}(0x + w_2 y) \\ =& \frac{1}{2}(w_1 x + w_2 y) \end{aligned}$$

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

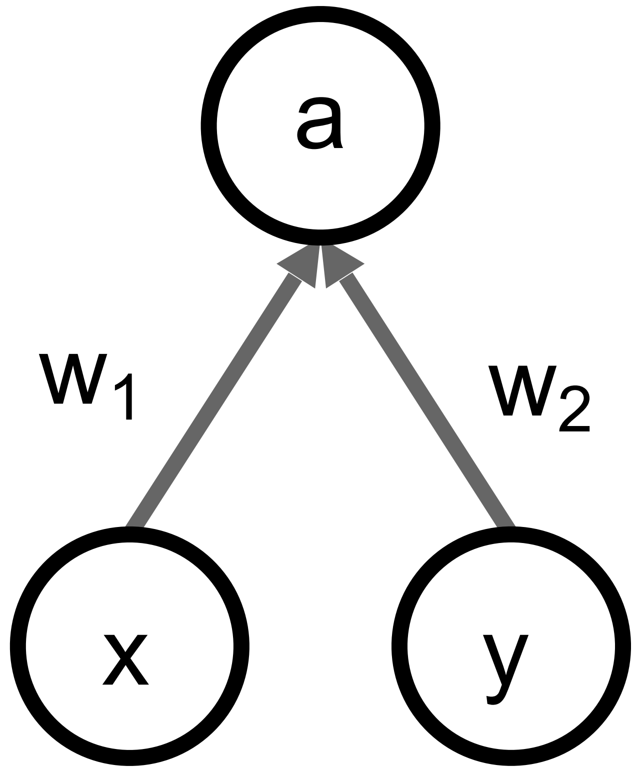
Skipped in class (outside of scope)

Regularization: Dropout at **test** time

An approximate solution

Want to approximate the integral $$y = f(x) = E_z\big[f(x, z)\big] = \int p(z) f(x, z) dz$$

Consider a single neuron.



At test time we have: $E\big[a\big] = w_1 x + w_2 y$

During training we have:

$$\begin{aligned} E\big[a\big] &= \frac{1}{4}(w_1 x + w_2 y) + \frac{1}{4}(w_1 x + 0y) \\ &\quad + \frac{1}{4}(0x + 0y) + \frac{1}{4}(0x + w_2 y) \\ &= \frac{1}{2}(w_1 x + w_2 y) \end{aligned}$$

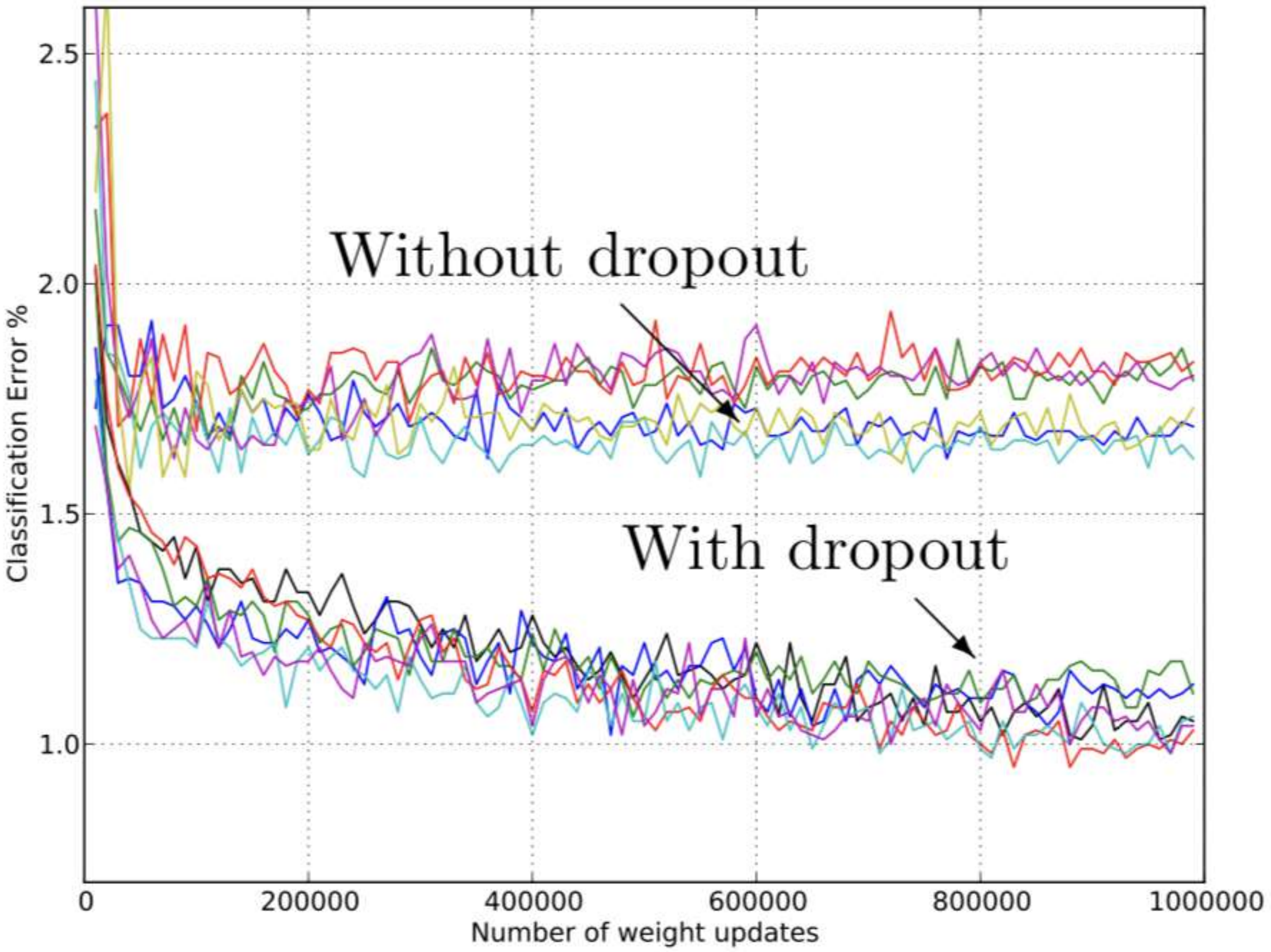
At test time, **multiply** by dropout probability

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

Skipped in class (outside of scope)

Regularization: Dropout

How good is it?



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

Skipped in class (outside of scope)

Regularization: A common pattern

Training: Add some kind of randomness

$$y = f_W(x, z)$$

Testing: Average out randomness (sometimes approximate)

$$y = f(x) = E_z\big[f(x, z)\big] = \int p(z) f(x, z) dz$$

Skipped in class (outside of scope)

Regularization: A common pattern

Training: Add some kind of randomness

$$y = f_W(x, z)$$

Testing: Average out randomness (sometimes approximate)

$$y = f(x) = E_z\big[f(x, z)\big] = \int p(z)f(x, z)dz$$

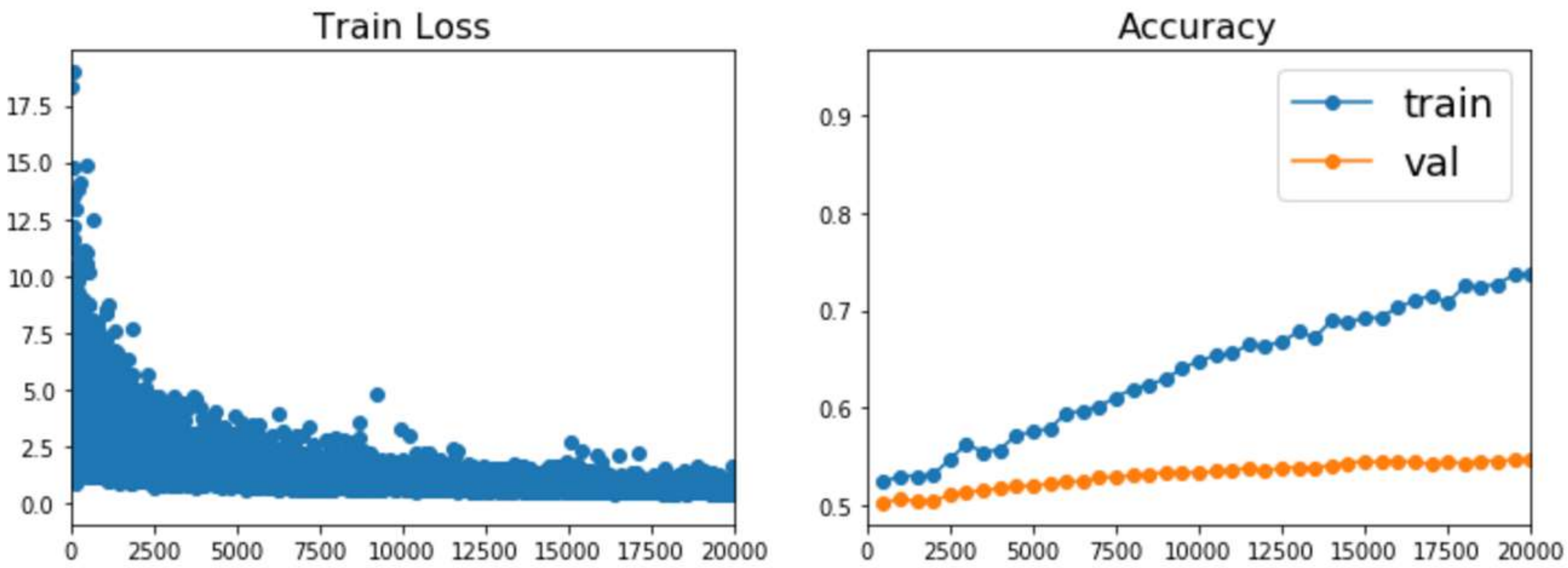
Example: Batch Normalization

Training: Normalize using stats from random minibatches

Testing: Use fixed stats to normalize

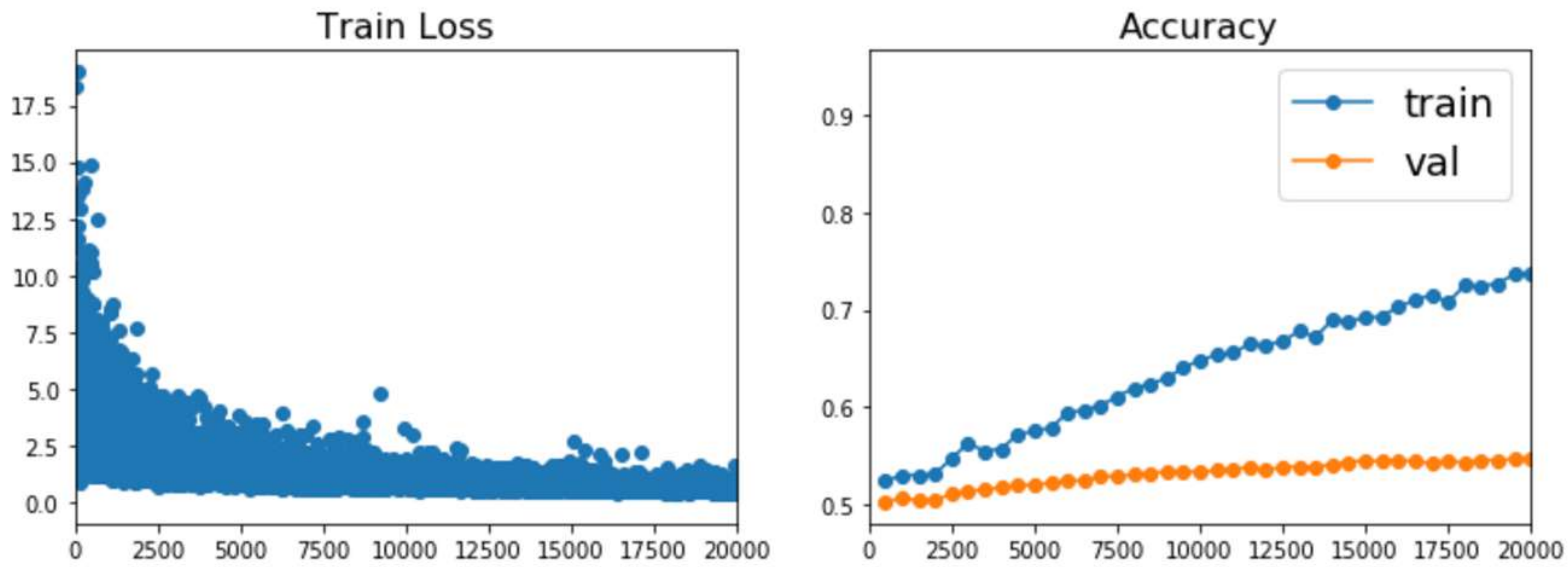
Skipped in class
(outside of scope)

Why does this happen in the first place?



Skipped in class (outside of scope)

Why does this happen in the first place?

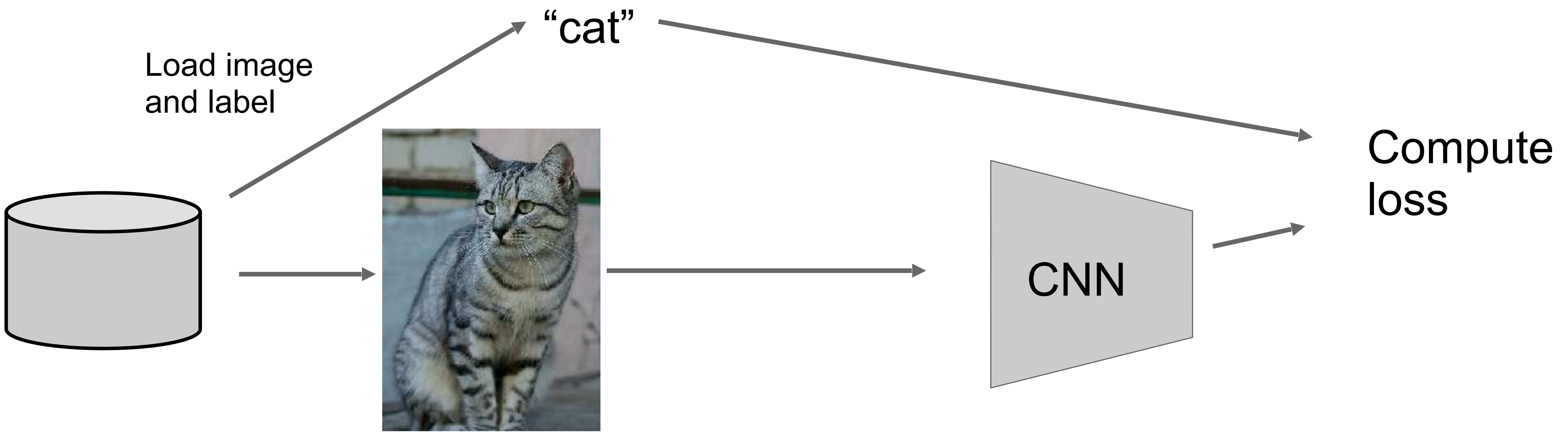


How can we have more data?

THE UNIVERSITY OF BRITISH COLUMBIA

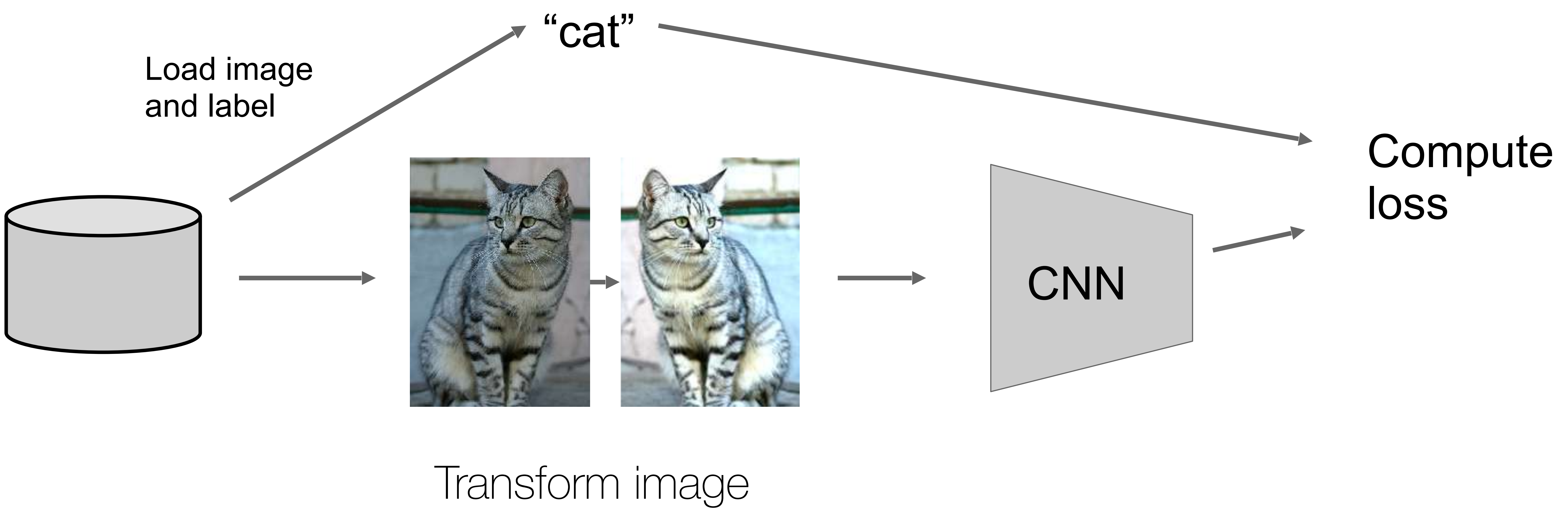
Skipped in class (outside of scope)

Regularization: Data augmentation



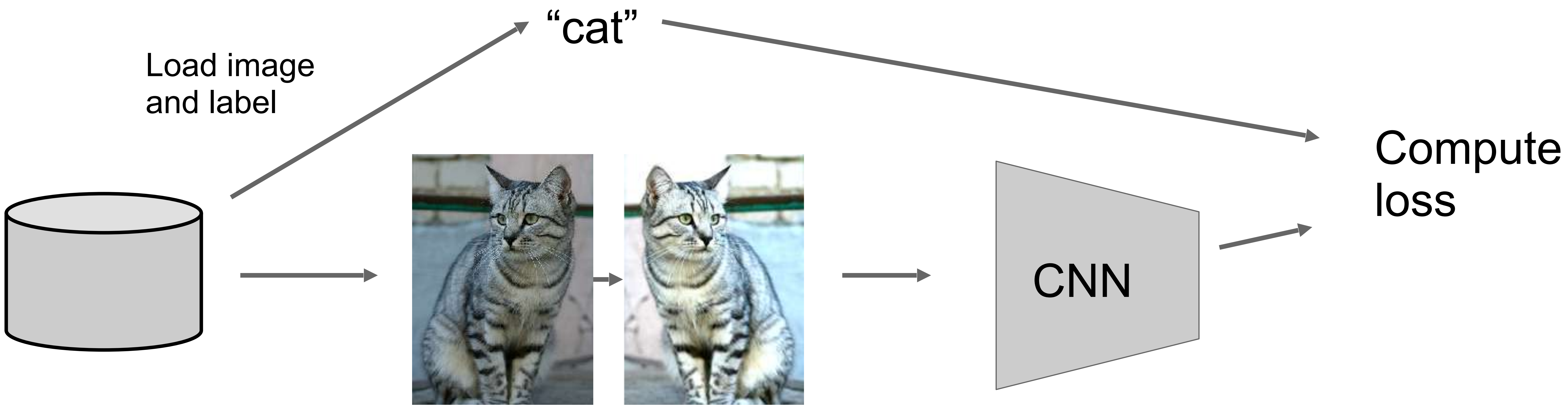
Skipped in class (outside of scope)

Regularization: Data augmentation



Skipped in class (outside of scope)

Regularization: Data augmentation

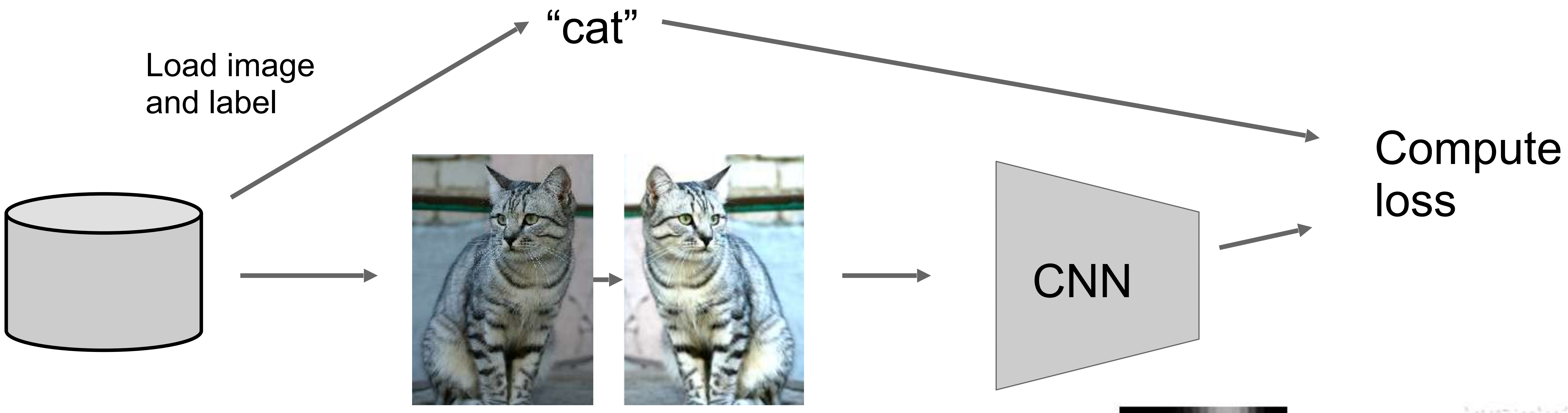


Transform image

- Horizontal / vertical flips
- Color / brightness
- Rotations / scaling
- Elastic transformation

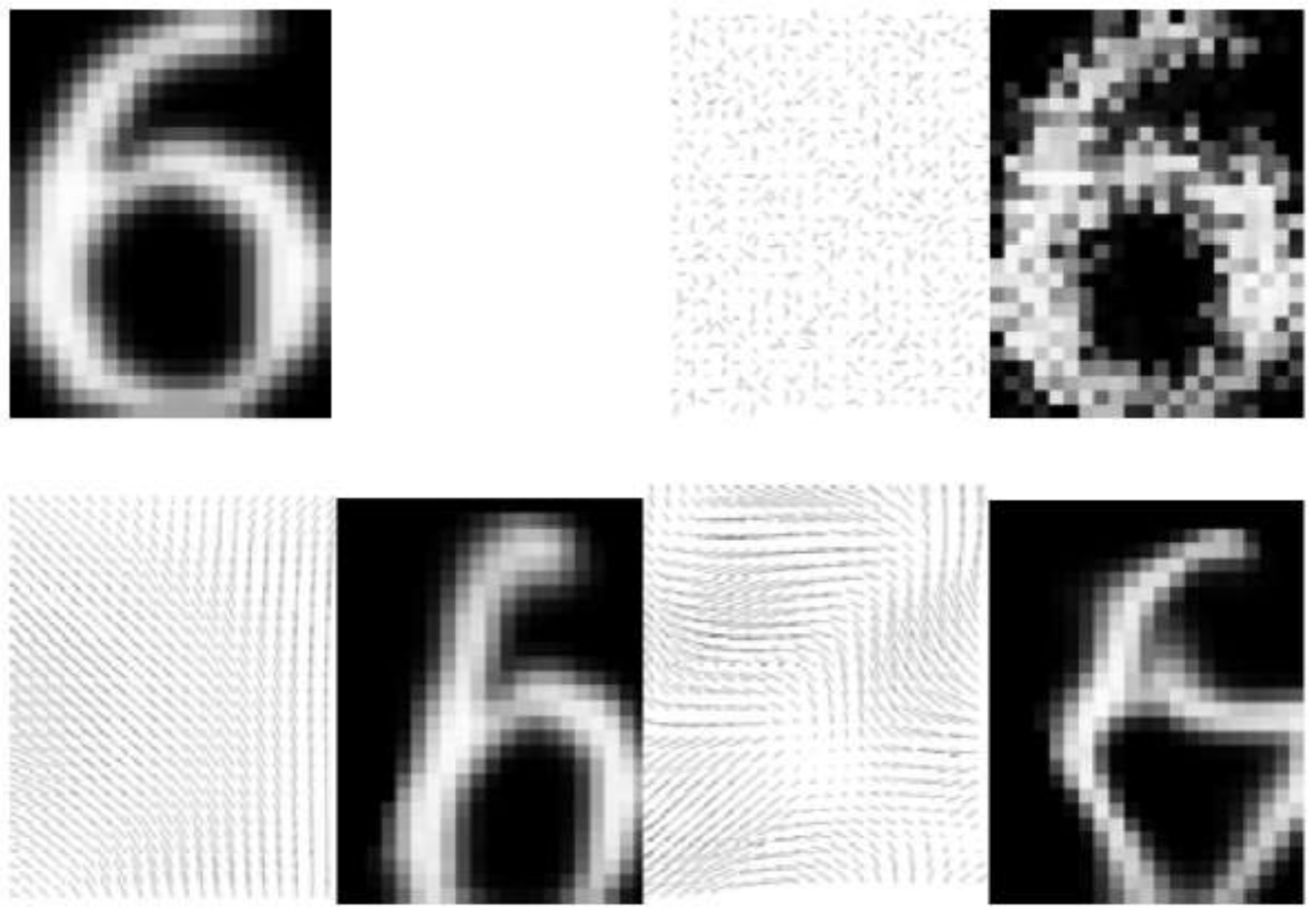
Regularization: Data augmentation

Skipped in class (outside of scope)



Transform image

- Horizontal / vertical flips
- Color / brightness
- Rotations / scaling
- Elastic transformation



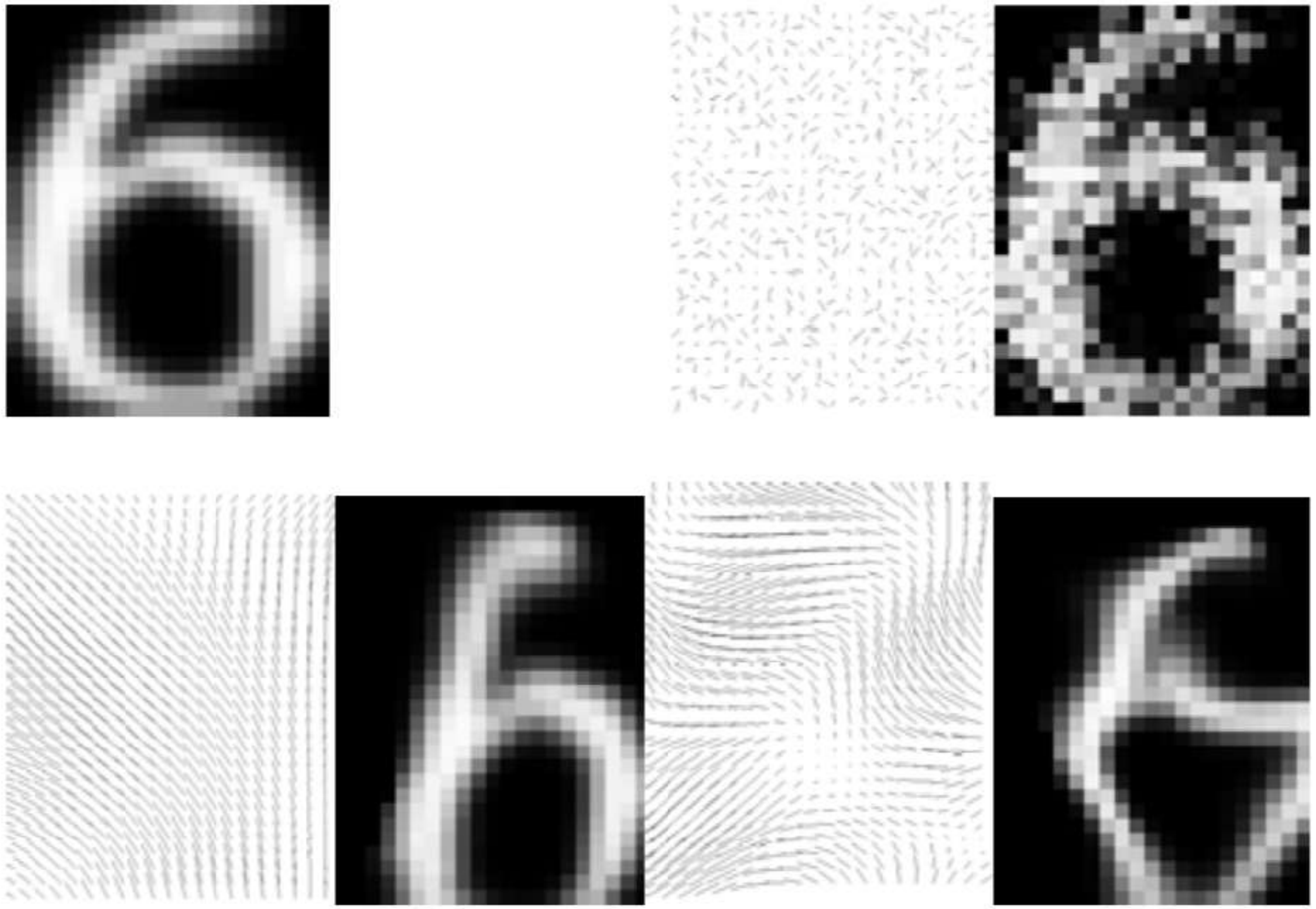
Simard, Steinkraus and Platt, "Best Practices for Convolutional Neural Networks applied to Visual Document Analysis", ICDAR, 2003

Based on slides for Stanford cs231n by Li, Jonson, and Young. Modified and reused with permission

Skipped in class (outside of scope)

Regularization: Data augmentation

Elastic deformations



Figures copyright IEEE, 2003. Reproduced for educational purposes.

1. Create random displacement field with uniform distribution
2. Smooth the displacement field with a Gaussian

Simard, Steinkraus and Platt, "Best Practices for Convolutional Neural Networks applied to Visual Document Analysis", ICDAR, 2003

Skipped in class (outside of scope)

Regularization: Data augmentation

Elastic deformations

Algorithm	Distortion	Error	Ref.
2 layer MLP (MSE)	affine	1.6%	[3]
SVM	affine	1.4%	[9]
Tangent dist.	affine+thick	1.1%	[3]
Lenet5 (MSE)	affine	0.8%	[3]
Boost. Lenet4 MSE	affine	0.7%	[3]
Virtual SVM	affine	0.6%	[9]
2 layer MLP (CE)	none	1.6%	this paper
2 layer MLP (CE)	affine	1.1%	this paper
2 layer MLP (MSE)	elastic	0.9%	this paper
2 layer MLP (CE)	elastic	0.7%	this paper
Simple conv (CE)	affine	0.6%	this paper
Simple conv (CE)	elastic	**0.4%**	this paper

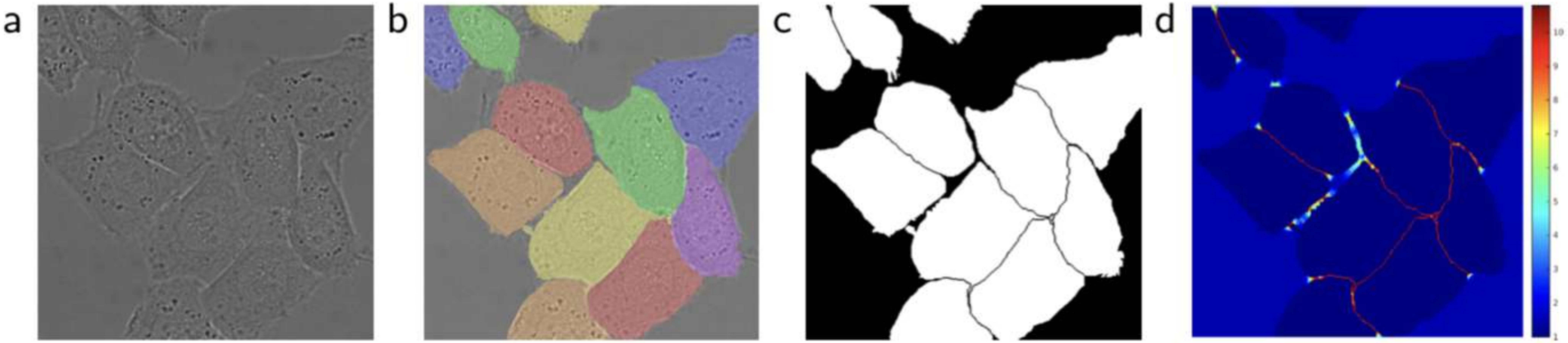
Table 1. Comparison between various algorithms.

Simard, Steinkraus and Platt, "Best Practices for Convolutional Neural Networks applied to Visual Document Analysis", ICDAR, 2003

Skipped in class (outside of scope)

Regularization: Data augmentation

Elastic deformations



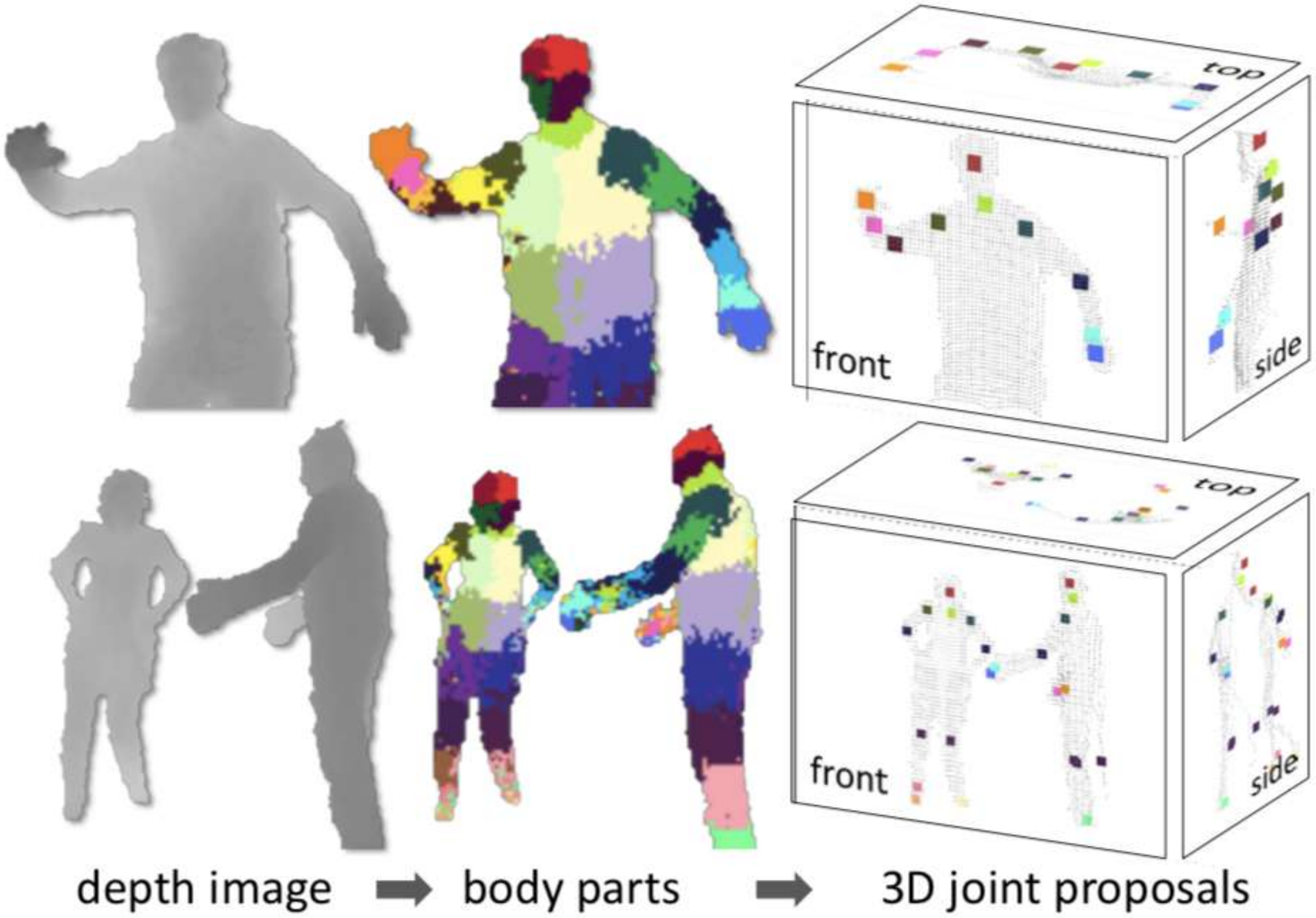
Name	PhC-U373	DIC-HeLa
IMCB-SG (2014)	0.2669	0.2935
KTH-SE (2014)	0.7953	0.4607
HOUS-US (2014)	0.5323	-
second-best 2015	0.83	0.46
u-net (2015)	**0.9203**	**0.7756**

Ronneberger et. al,, "U-Net: Convolutional Networks for Biomedical Image Segmentation", 2015

Skipped in class
(outside of scope)

Regularization: Data augmentation

Synthetic data



Shotton et. al,, "Real-Time Human Pose Recognition in Parts from Single Depth Images", 2011

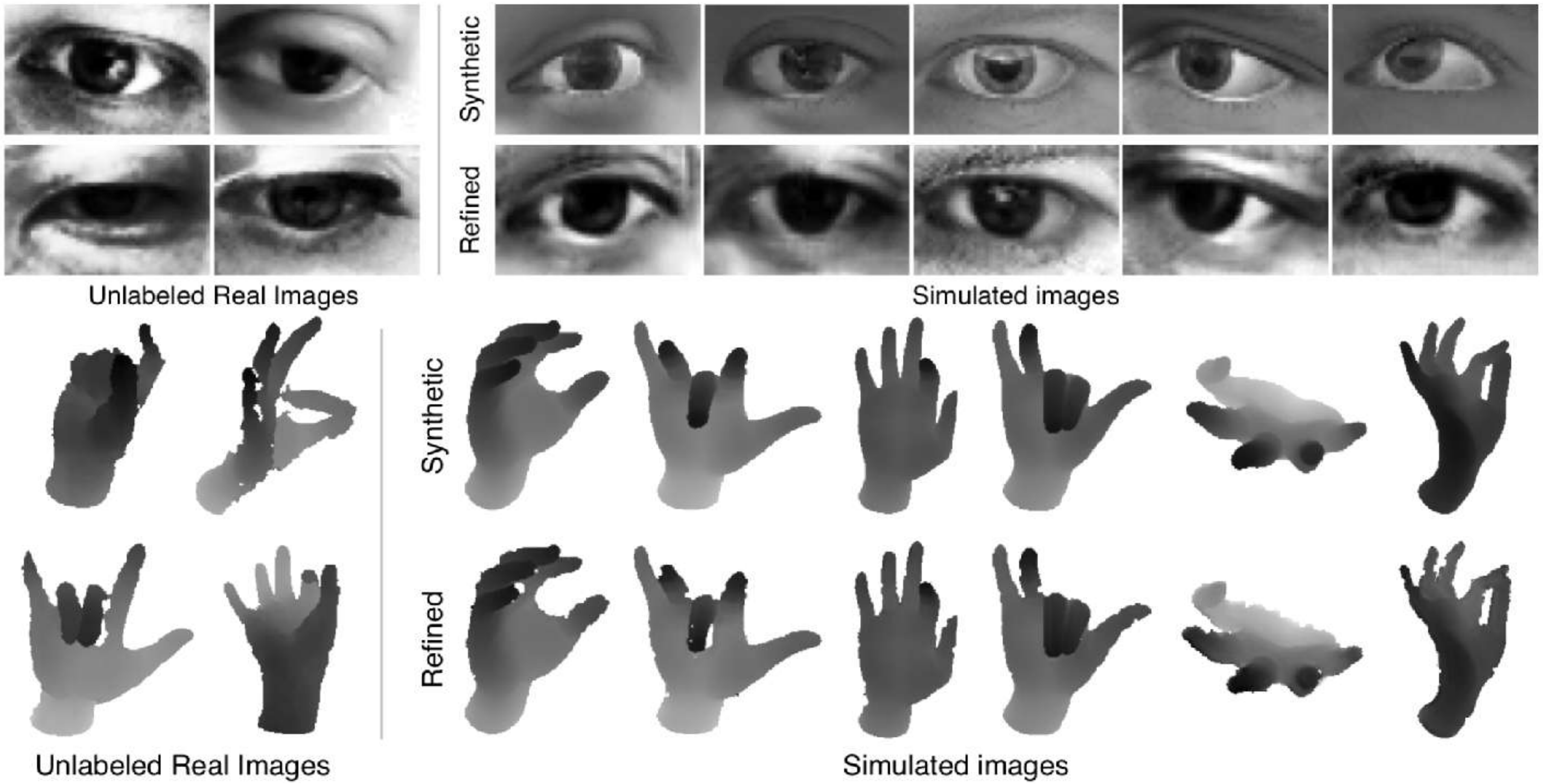
Based on slides for Stanford cs231n by Li, Jonson, and Young. Modified and reused with permission

THE UNIVERSITY OF BRITISH COLUMBIA

Skipped in class (outside of scope)

Regularization: Data augmentation

Synthetic data + generative models



Shrivastava et. al,, "Learning from Simulated and Unsupervised Images through Adversarial Training", 2011

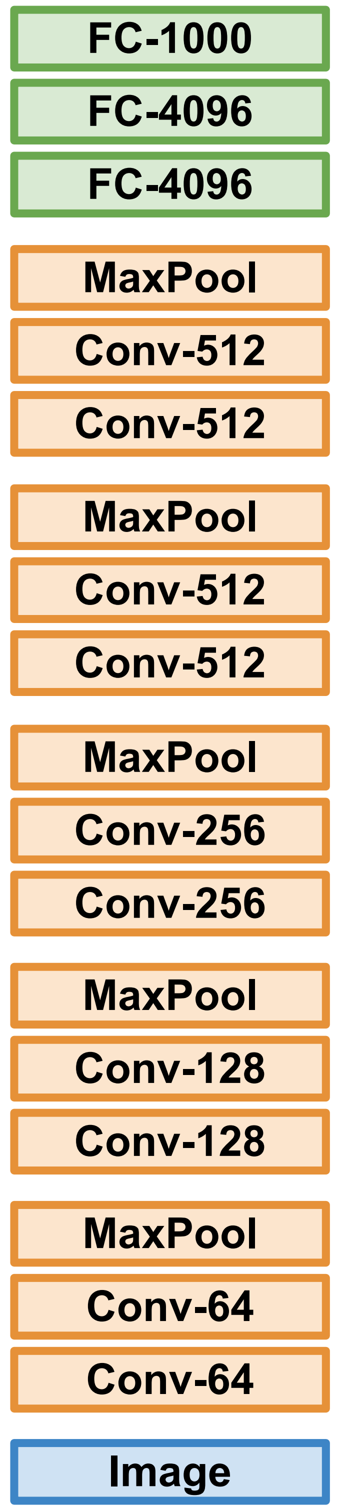
Based on slides for Stanford cs231n by Li, Jonson, and Young. Modified and reused with permission

Donahue et al, "DeCAF: A Deep Convolutional Activation Feature for Generic Visual Recognition", ICML 2014
Razavian et al, "CNN Features Off-the-Shelf: An Astounding Baseline for Recognition", CVPR Workshops 2014

Using pretrained networks

Skipped in class (outside of scope)

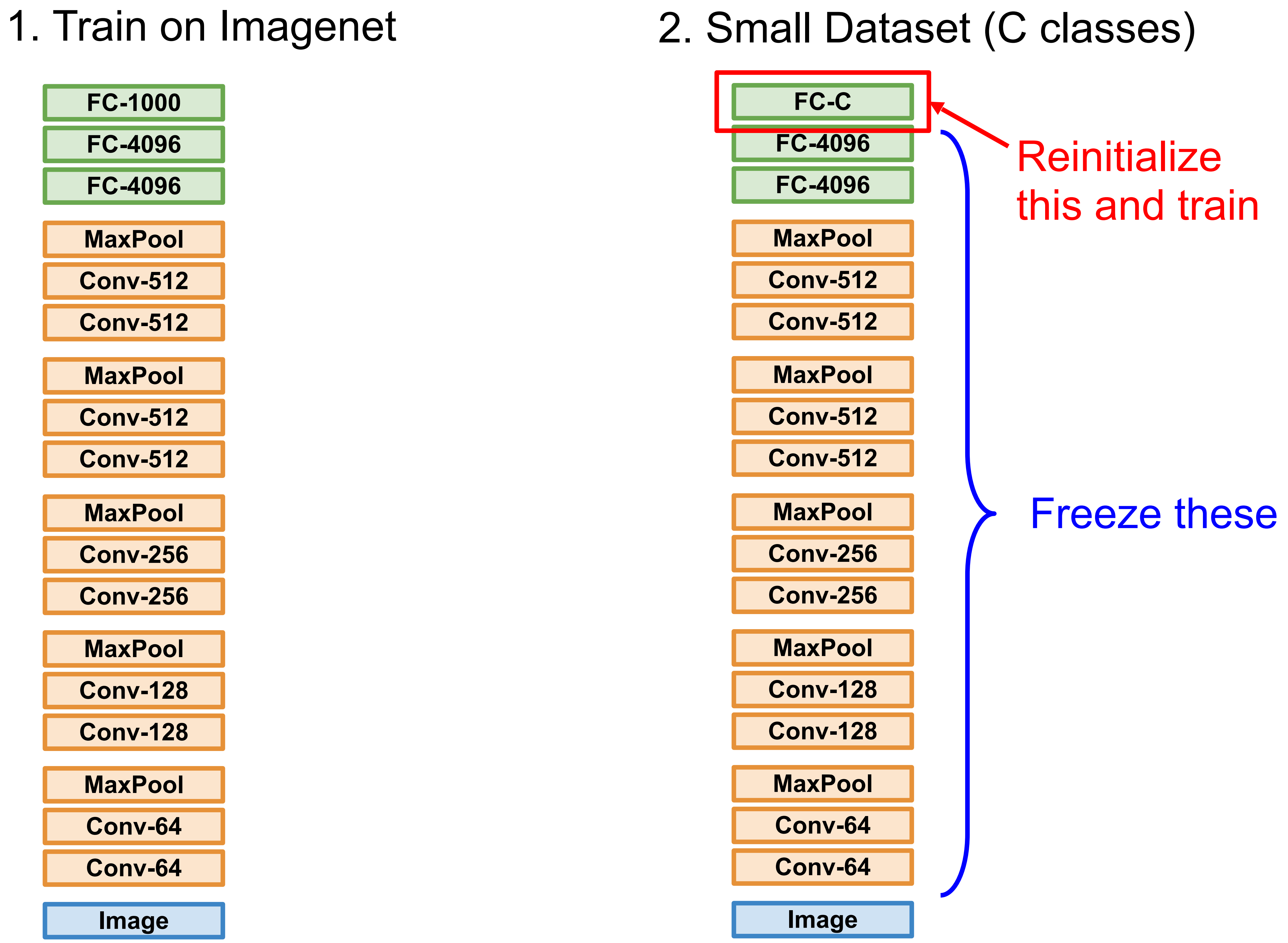
1. Train on Imagenet



Donahue et al, "DeCAF: A Deep Convolutional Activation Feature for Generic Visual Recognition", ICML 2014
Razavian et al, "CNN Features Off-the-Shelf: An Astounding Baseline for Recognition", CVPR Workshops 2014

Skipped in class (outside of scope)

Using pretrained networks

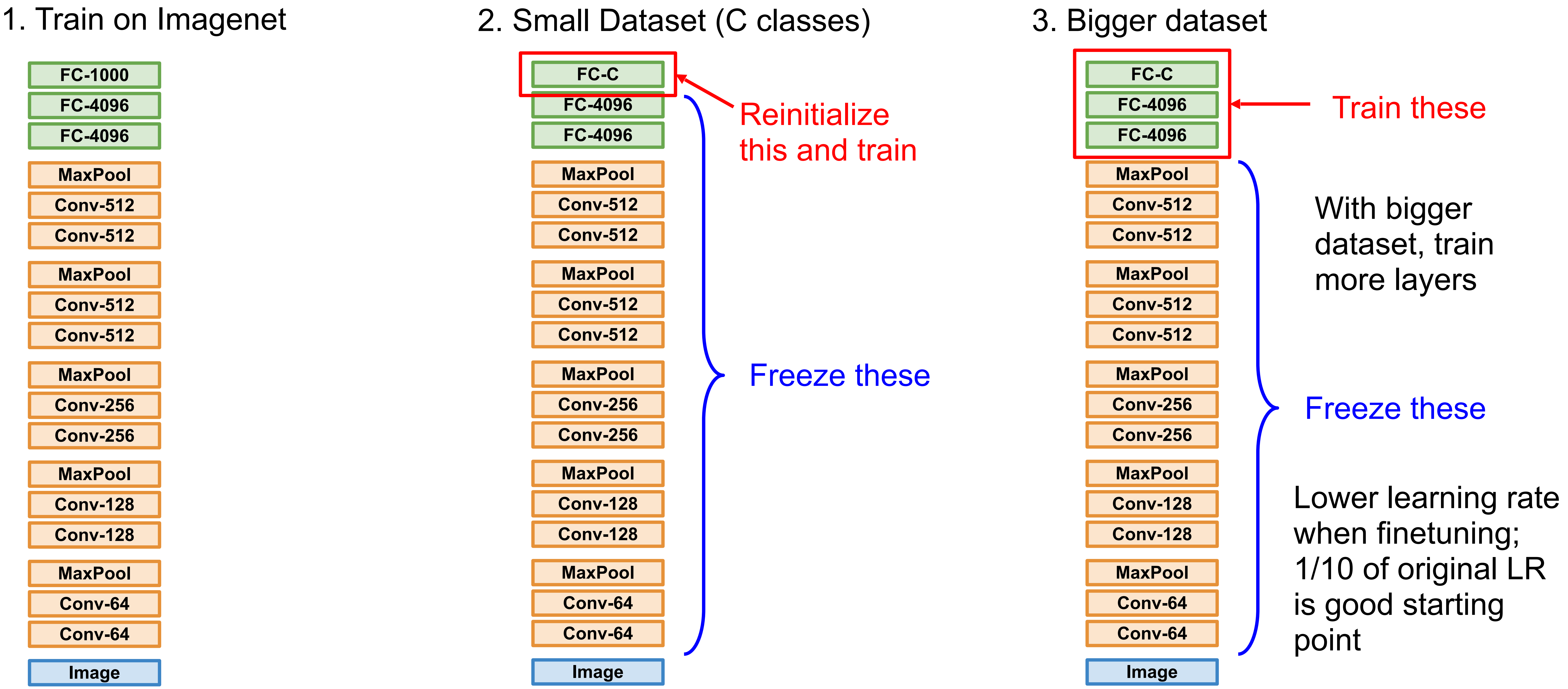


THE UNIVERSITY OF BRITISH COLUMBIA

Donahue et al, "DeCAF: A Deep Convolutional Activation Feature for Generic Visual Recognition", ICML 2014
Razavian et al, "CNN Features Off-the-Shelf: An Astounding Baseline for Recognition", CVPR Workshops 2014

Using pretrained networks

Skipped in class (outside of scope)



Based on slides for Stanford cs231n by Li, Jonson, and Young. Modified and reused with permission

THE UNIVERSITY OF BRITISH COLUMBIA

Large generative models

Skipped in class (outside of scope)

[Video from https://twitter.com/HaiperGenAI/status/1745845670844522760

Skipped in class (outside of scope)

Large generative models

[Video from https://twitter.com/HaiperGenAI/status/1745845670844522760

Fishing information within SD

Skipped in class (outside of scope)

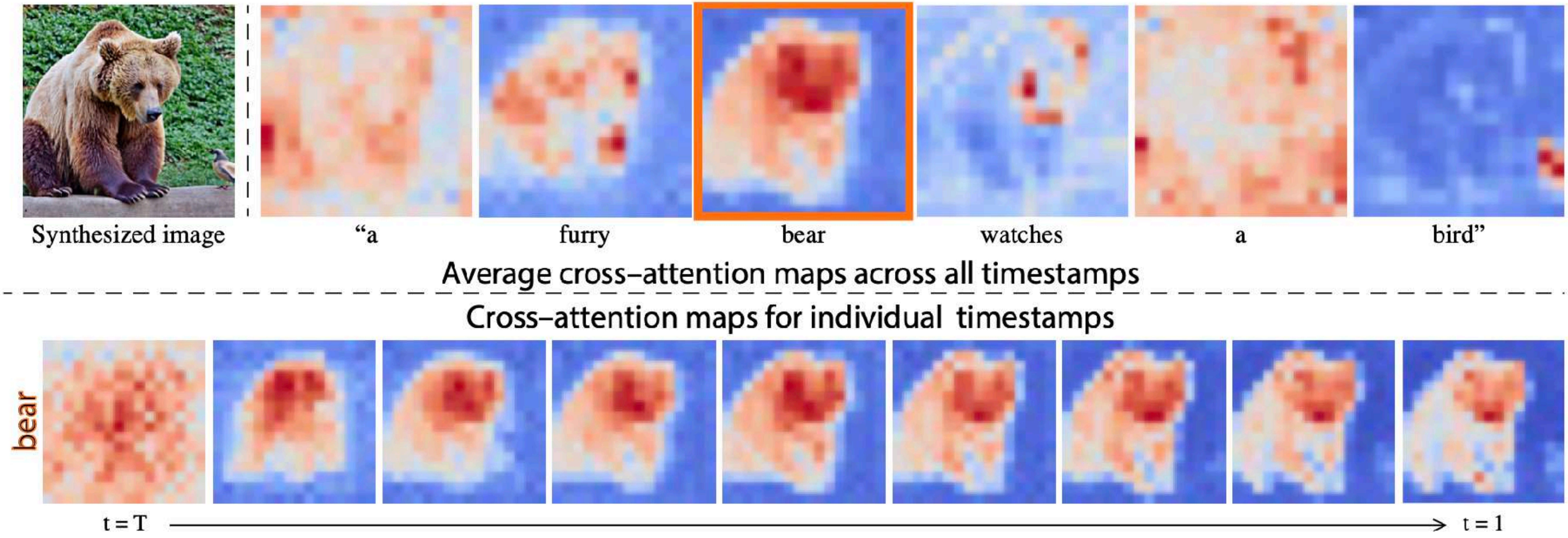
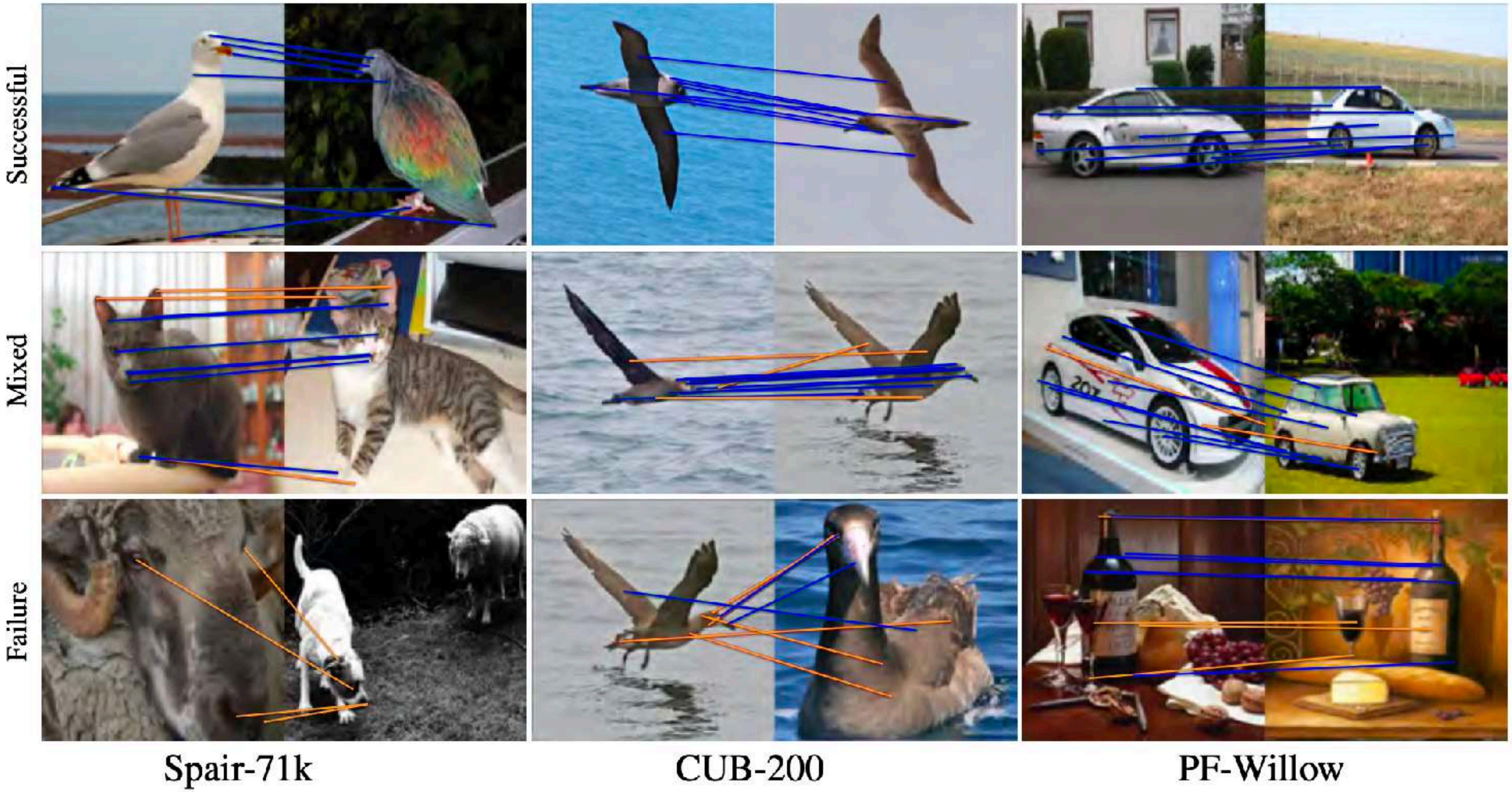


Image from [Hertz et al., ICLR, 2023]

THE UNIVERSITY OF BRITISH COLUMBIA

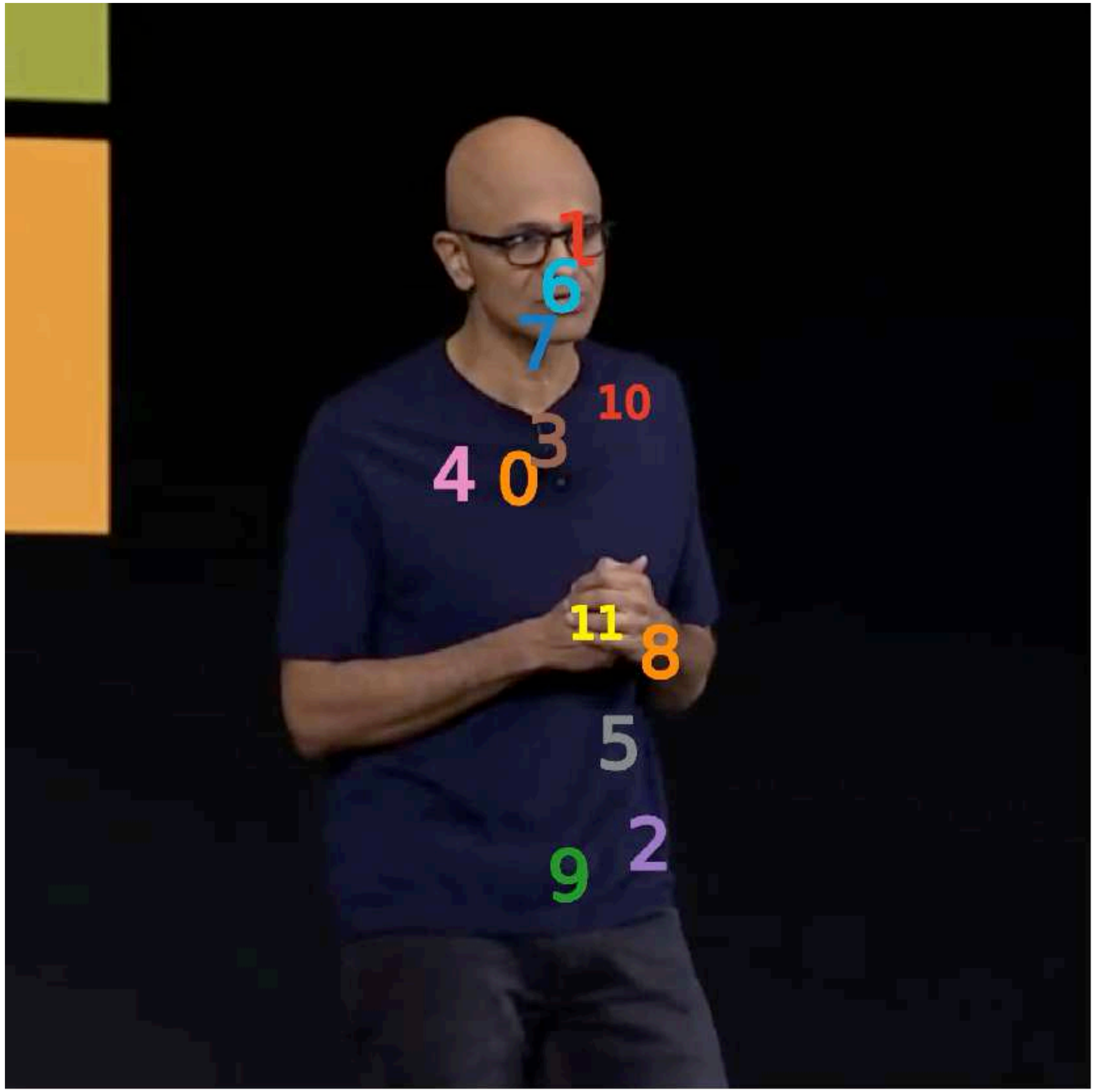
Correspondences from SD

Skipped in class (outside of scope)



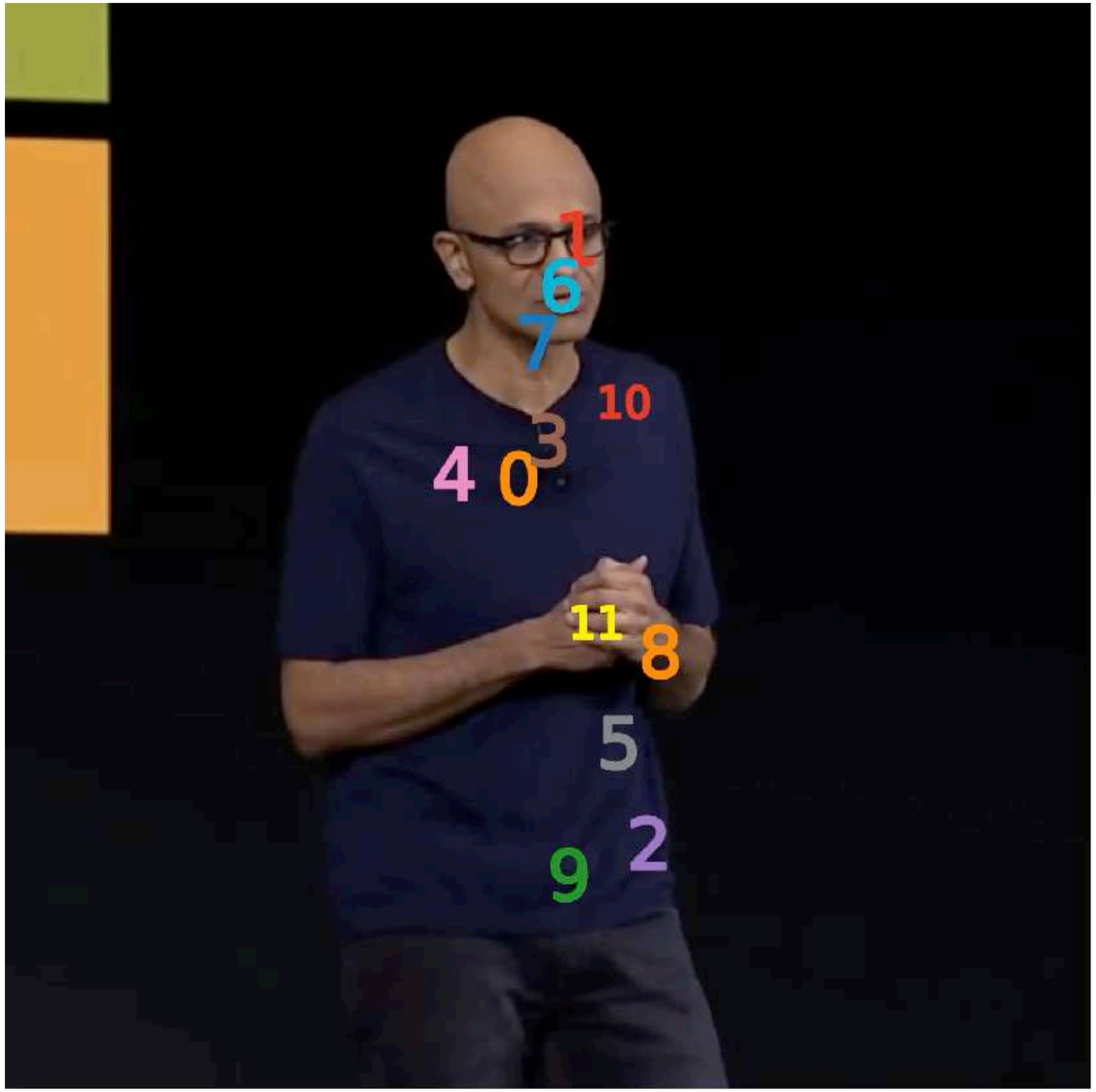
Keypoints from SD

Skipped in class (outside of scope)



Keypoints from SD

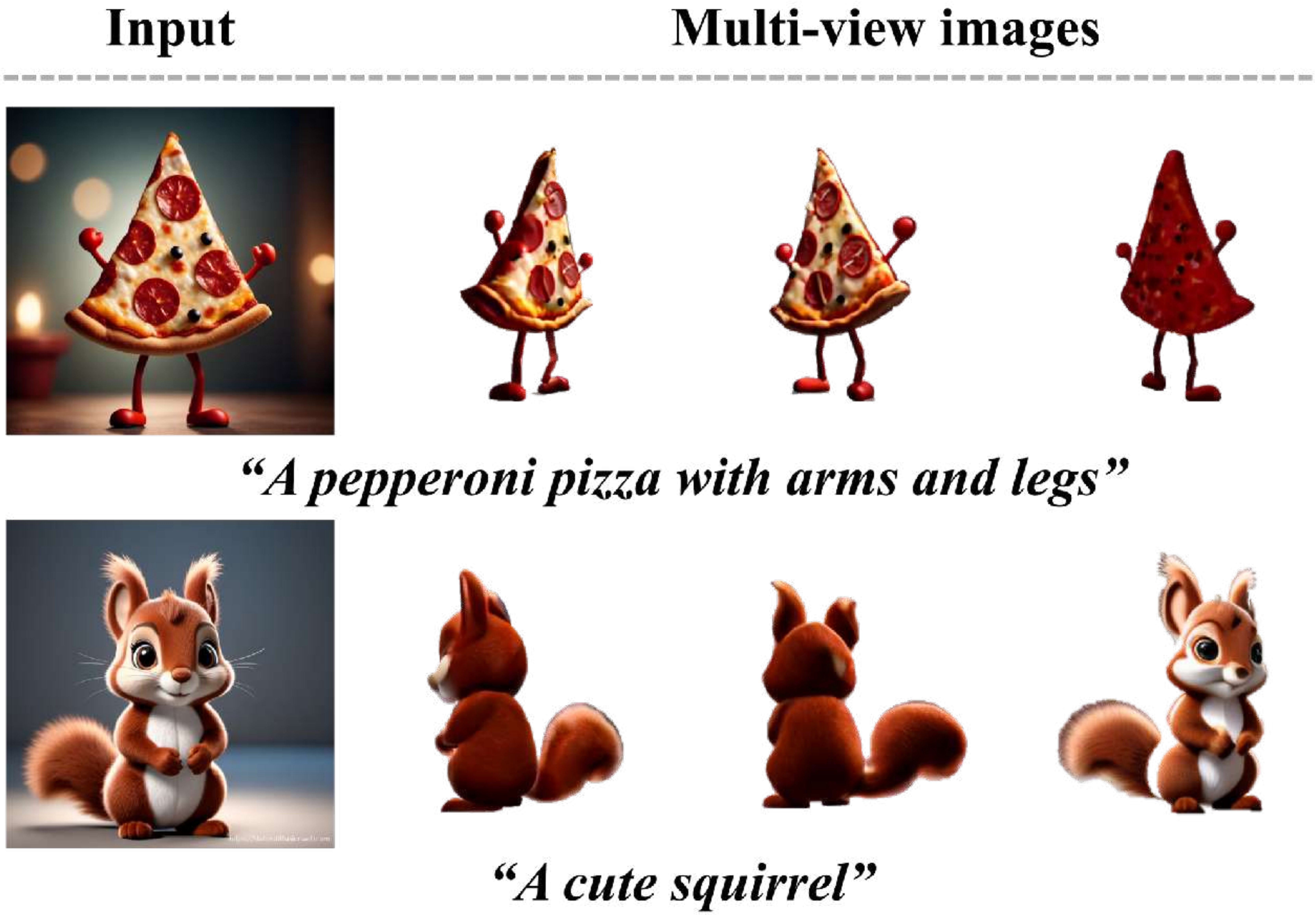
Skipped in class
(outside of scope)



THE UNIVERSITY OF BRITISH COLUMBIA

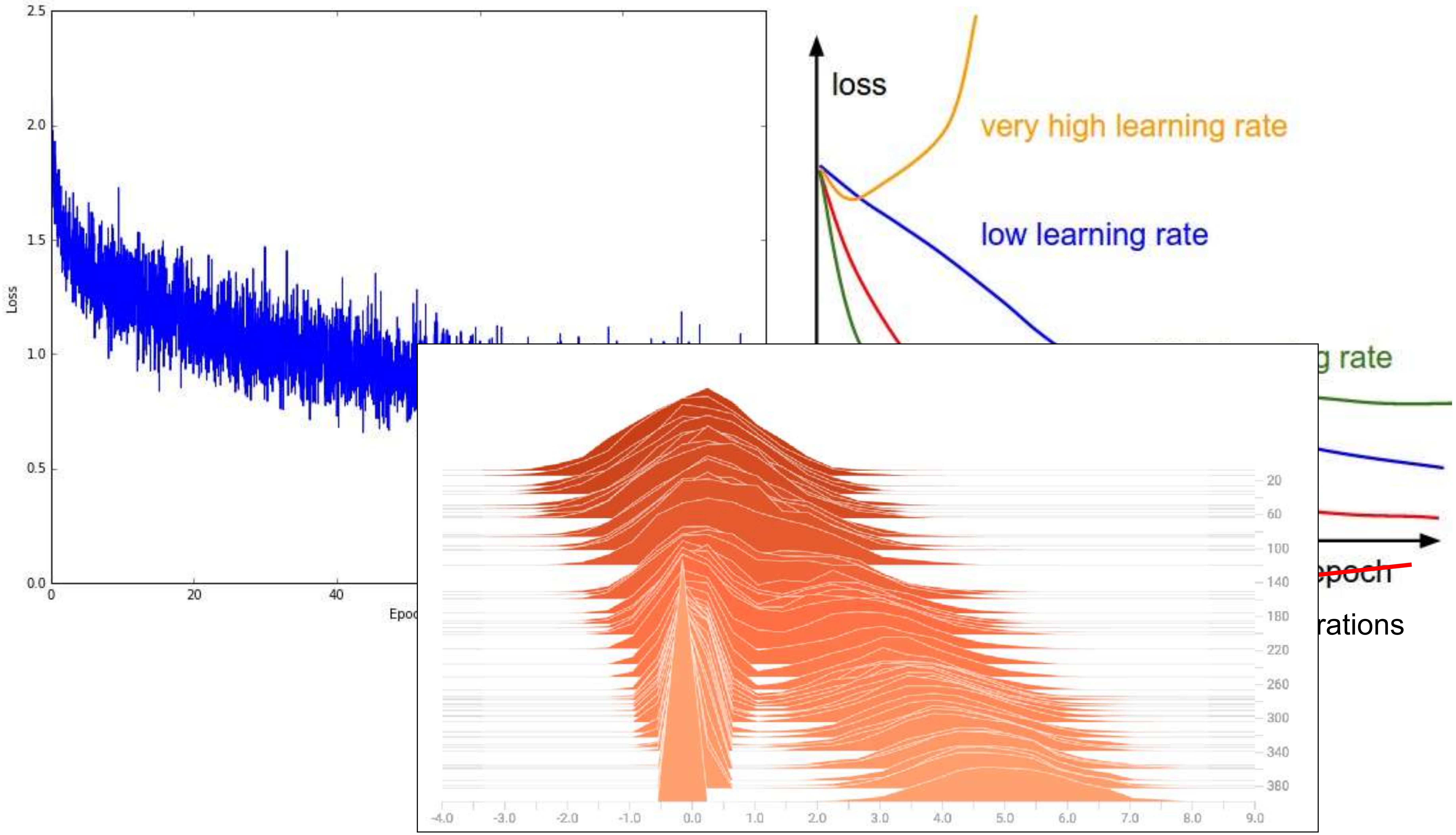
Text-to-3D from SD

Skipped in class (outside of scope)



Skipped in class

Visualize VISUALIZE **VISUALIZE** (outside of scope)



More on Neural Networks

Skipped in class (outside of scope)

Lots more to learn! A good place to start is

Justin Johnson, University of Michigan, EECS 498/598, e.g.,

https://web.eecs.umich.edu/~justincj/teaching/eecs498/WI2022/

Training Neural Nets: **Clever** Hans

Skipped in class
(outside of scope)

Training Neural Nets: **Clever** Hans

Skipped in class
(outside of scope)

Hans could get 89% of the math questions right

Training Neural Nets: **Clever** Hans

Skipped in class
(outside of scope)

Hans could get 89% of the math questions right