CS 6501: Deep Learning for Computer Graphics

Training Neural Networks II

Connelly Barnes

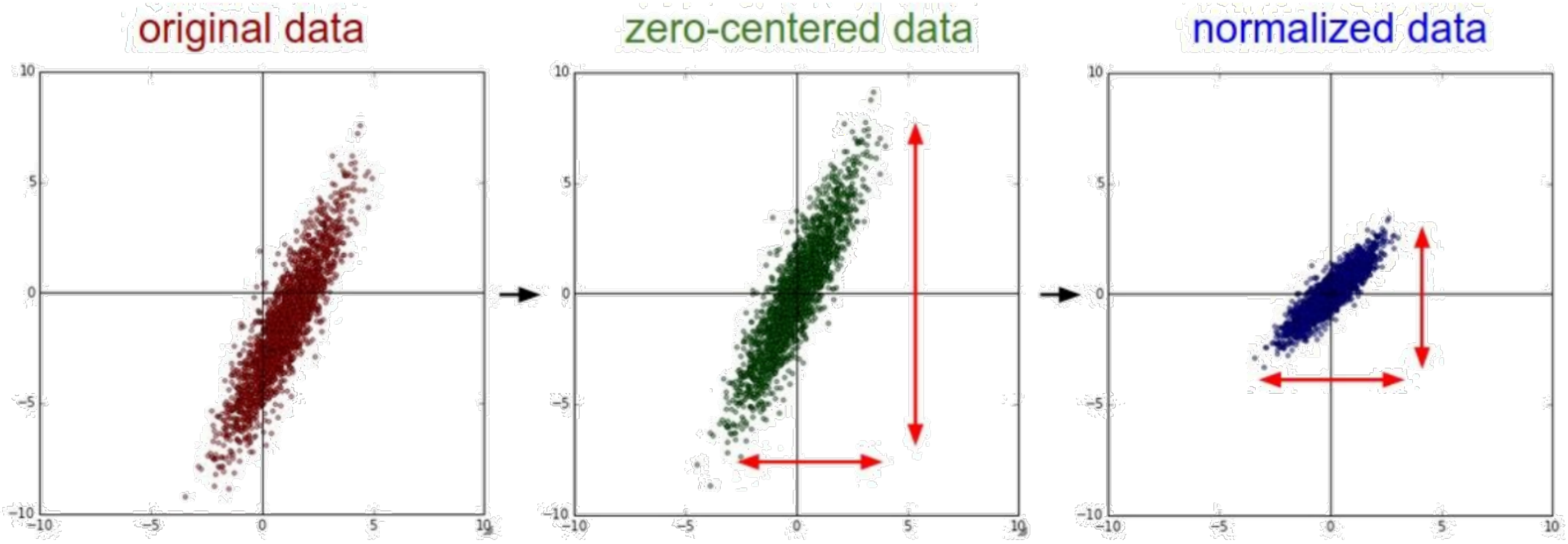
Overview

- **Preprocessing**
- Initialization
- Vanishing/exploding gradients problem
- Batch normalization
- Dropout

- Additional neuron types:
  - Softmax

Preprocessing

- Common: zero-center, can normalize variance.




```
X -= np.mean(X, axis = 0)
```

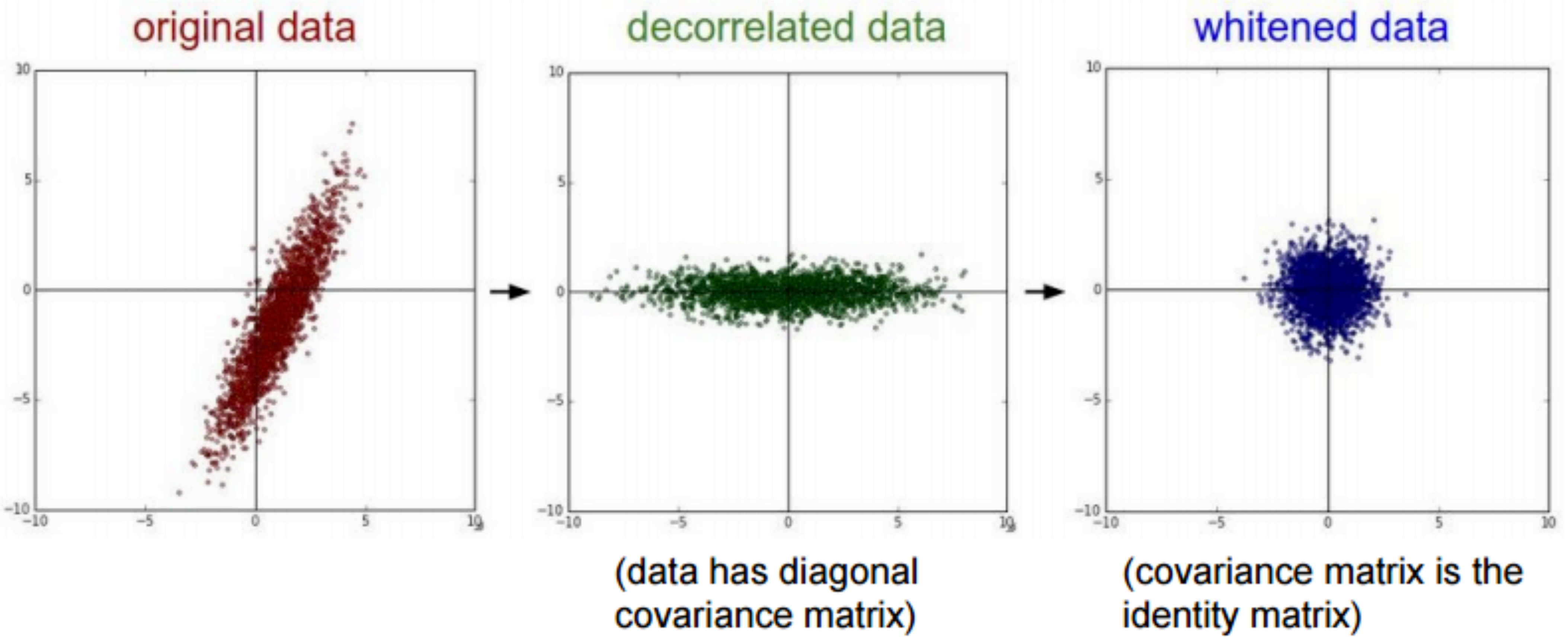
```
X /= np.std(X, axis = 0)
```

(Assume X [NxD] is data matrix, each example in a row)

Slide from [Stanford CS231n]

Preprocessing

- Can also decorrelate the data by using PCA, or whiten data



Slide from Stanford CS231n

Preprocessing for Images

- Center the data only
- Compute a mean image (examples of mean faces)
    - Either grayscale or compute separate mean for channels (RGB)
- Subtract the mean from your dataset

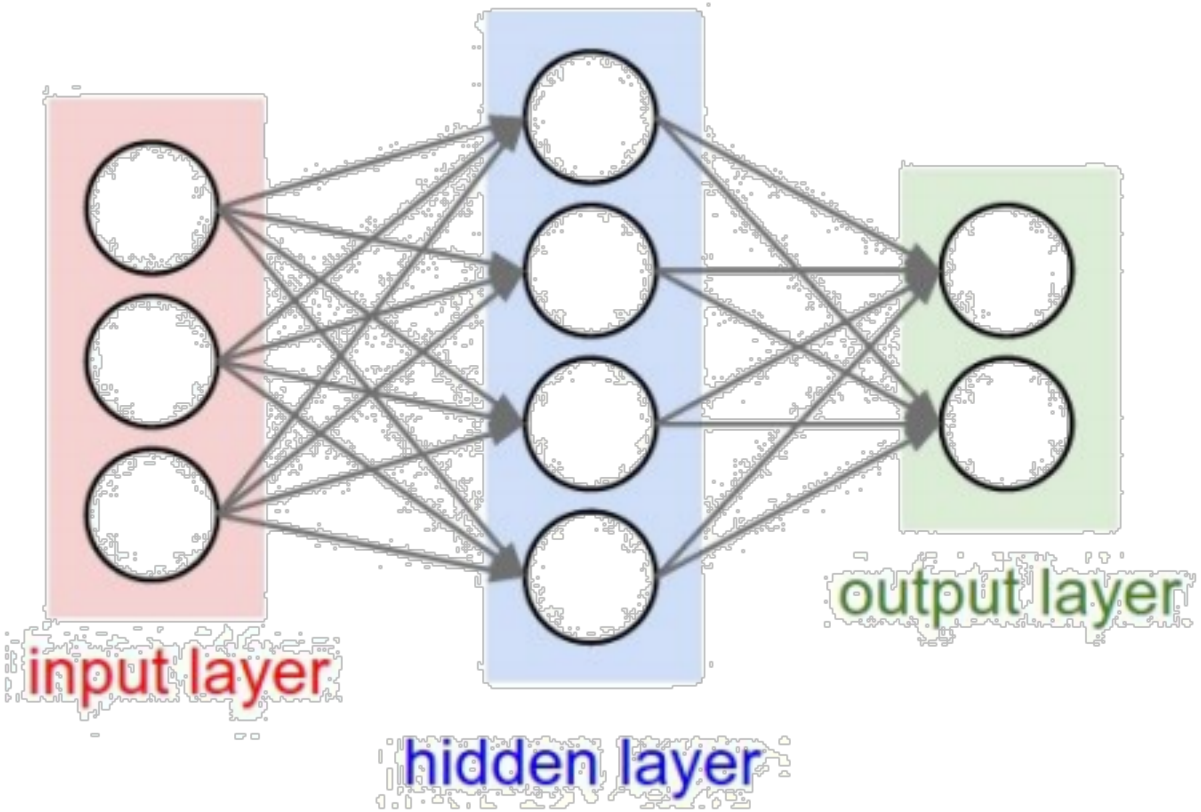
Overview

- Preprocessing
- **Initialization**
- Vanishing/exploding gradients problem
- Batch normalization
- Dropout

- Additional neuron types:
  - Softmax

Initialization

- Need to start gradient descent at an initial guess
- What happens if we initialize all weights to zero?



Slide from [Stanford CS231n]

Initialization

- Idea: random numbers (e.g. normal distribution)

$$w_{ij} = \mathcal{N}(\mu, \sigma)$$

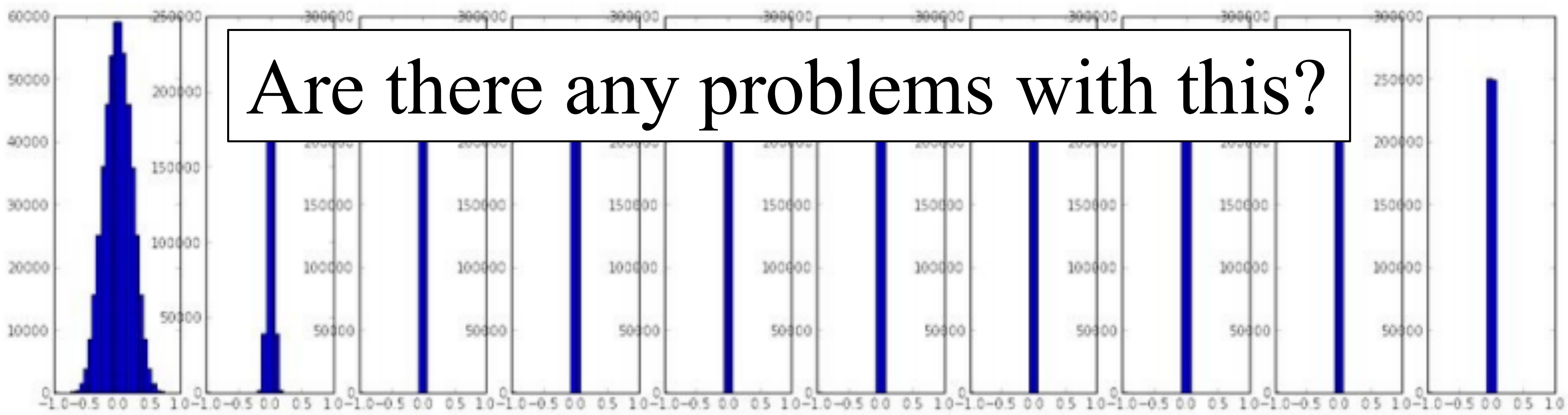
$$\mu = 0, \quad \sigma = \text{const}$$

- OK for shallow networks, but what about deep networks?

Initialization, σ = 0.01

- Simulation: multilayer perceptron, 10 fully-connected hidden layers
- Tanh() activation function

Hidden layer activation function statistics:



Are there any problems with this?

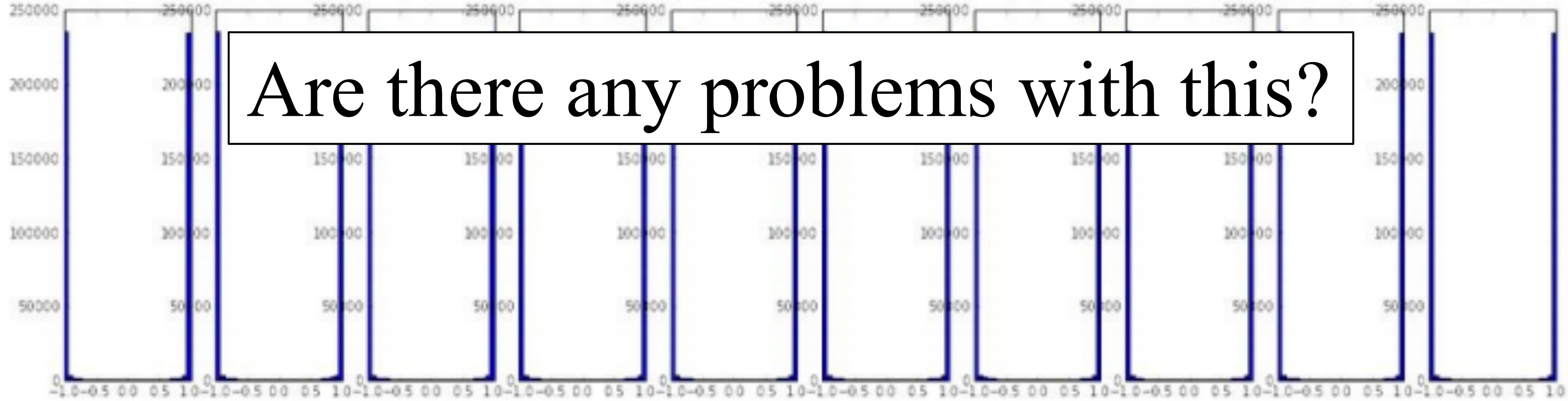
Hidden Layer 1 Hidden Layer 10

Slide from Stanford CS231n

Initialization, σ = 1

- Simulation: multilayer perceptron, 10 fully-connected hidden layers
- Tanh() activation function

Hidden layer activation function statistics:



Are there any problems with this?

Hidden Layer 1 Hidden Layer 10

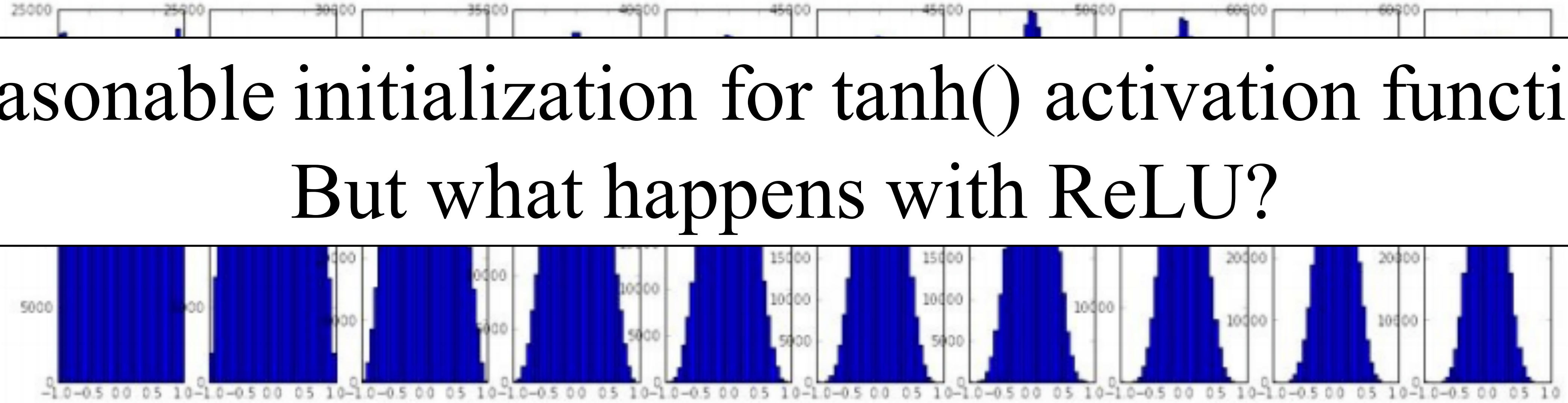
Slide from [Stanford CS231n]

Xavier Initialization

$$\sigma = \frac{1}{\sqrt{n_{\text{in}}}}$$

n_{in}: Number of neurons feeding into given neuron
(actually, Xavier used a uniform distribution)

Hidden layer activation function statistics:



Reasonable initialization for tanh() activation function.
But what happens with ReLU?

Hidden Layer 1 Hidden Layer 10

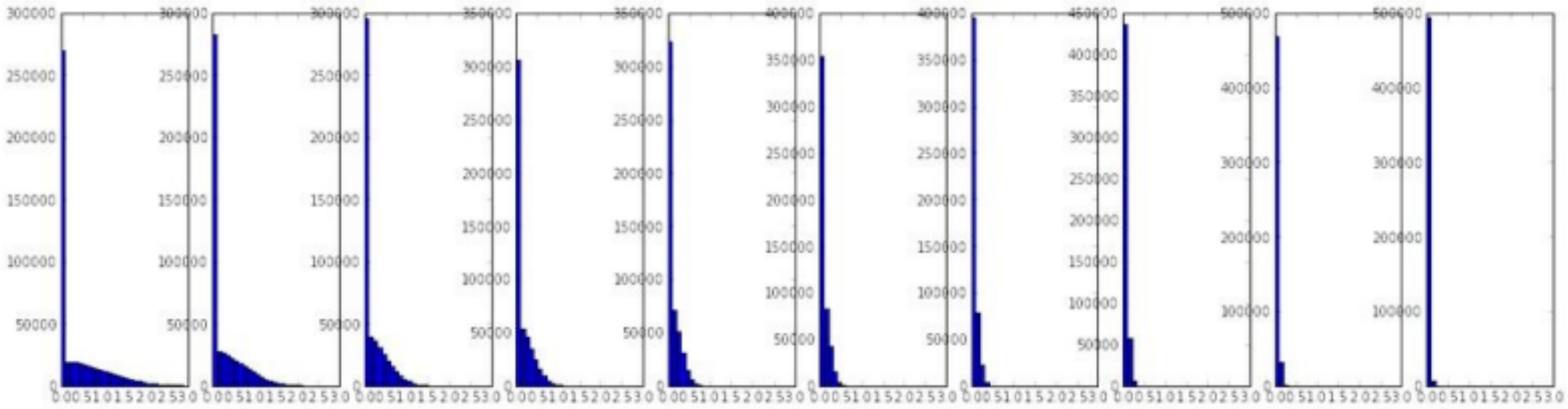
Slide from Stanford CS231n

Xavier Initialization, ReLU

$$\sigma = \frac{1}{\sqrt{n_{\text{in}}}}$$

n_{in}: Number of neurons feeding into given neuron

Hidden layer activation function statistics:



Hidden Layer 1

Hidden Layer 10

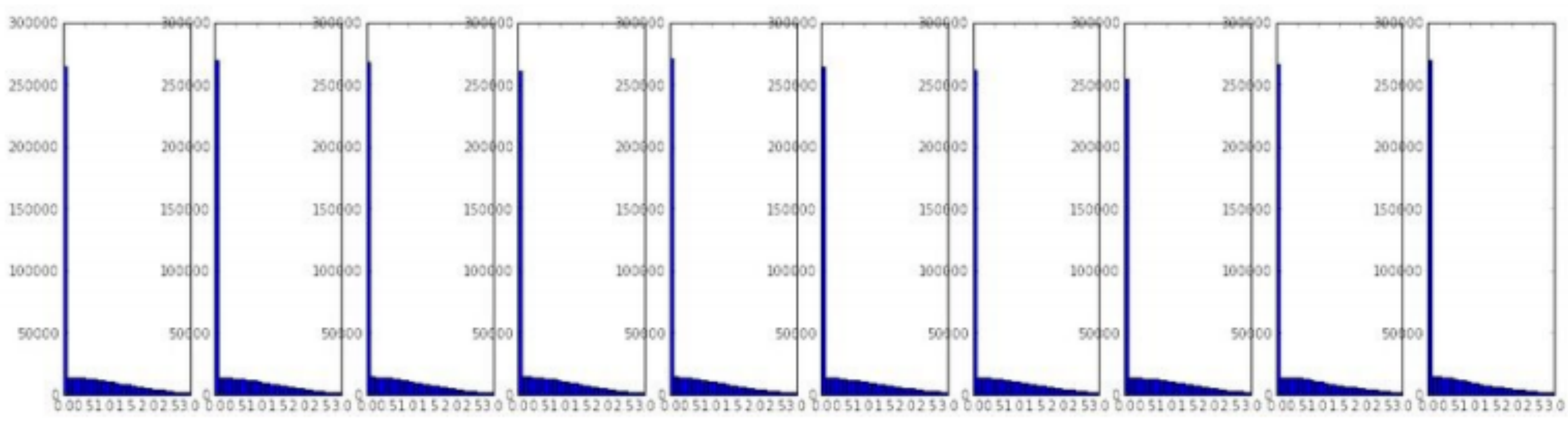
Slide from Stanford CS231n

He et al. 2015 Initialization, ReLU

$$\sigma = \frac{\sqrt{2}}{\sqrt{n_{\text{in}}}}$$

n_{in}: Number of neurons feeding into given neuron

Hidden layer activation function statistics:



Hidden Layer 1

Hidden Layer 10

Slide from Stanford CS231n

Other Ways to Initialize?

- Start with an existing pre-trained neural network's weights, **fine tune** its weights by re-running gradient descent
  - This is really transfer learning, since it also transfers knowledge from the previously trained network
- Previously, people used unsupervised pre-training with autoencoders
  - But we have better initializations now

Overview

- Preprocessing
- Initialization
- **Vanishing/exploding gradients problem**
- Batch normalization
- Dropout

- Additional neuron types:
    - Softmax

Vanishing/exploding gradient problem

- Recall from the backpropagation algorithm (last class slides):

$$\frac{\partial E}{\partial w_{ij}} = \delta_j o_i$$

$$\delta_j = \frac{\partial E}{\partial o_j}\frac{\partial o_j}{\partial \text{net}_j} = \varphi'(o_j)\begin{cases} (o_j - t_j) & \text{if } j \text{ is an output neuron} \\ \sum_{l \in L} \delta_l w_{jl} & \text{if } j \text{ is an interior neuron} \end{cases}$$

- Take $||\boldsymbol{\delta}||$ over all neurons in a layer.
- We can call this a "learning speed."

Vanishing/exploding gradient problem

- **Vanishing gradients problem**: neurons in earlier layers learn more slowly than in latter layers.

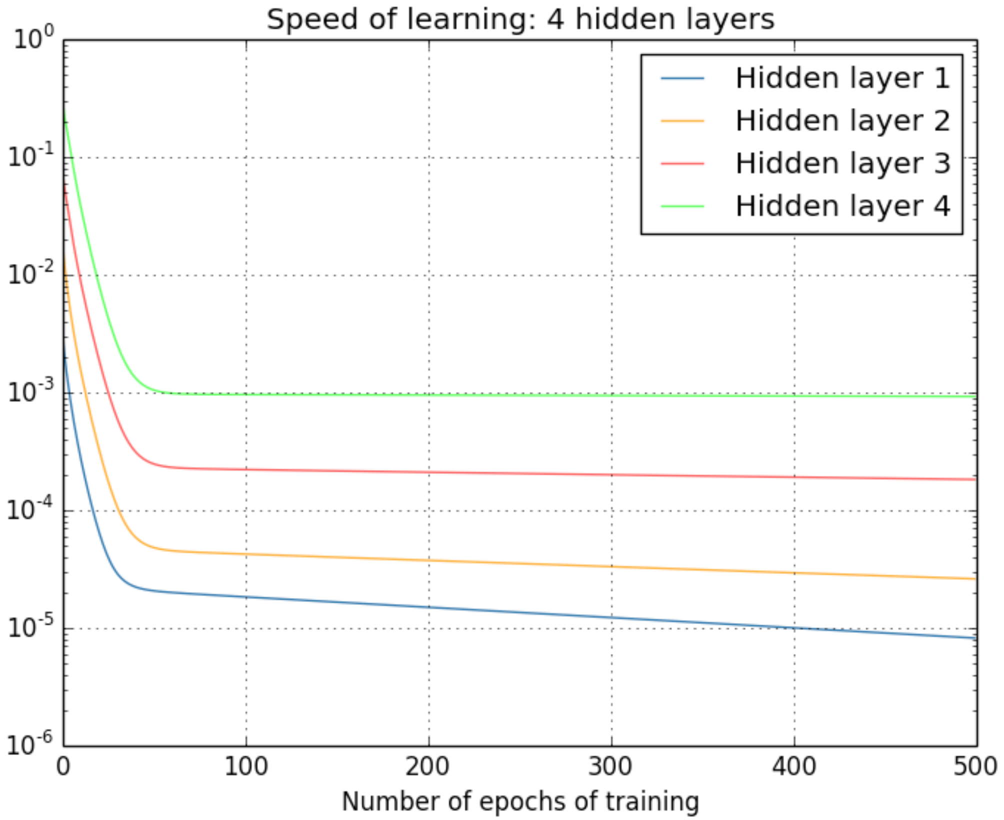


Image from Nielson 2015

Vanishing/exploding gradient problem

- **Vanishing gradients problem**: neurons in earlier layers learn more slowly than in latter layers.
- **Exploding gradients problem**: gradients are significantly larger in earlier layers than latter layers.

- How to avoid?
  - Use a good initialization
  - Do not use sigmoid for deep networks
  - Use momentum with carefully tuned schedules, e.g.:

$$\mu_t = \min(1 - 2^{-1-\log_2(\lfloor t/250 \rfloor + 1)}, \mu_{\max})$$

Image from Nielson 2015

Overview

- Preprocessing
- Initialization
- Vanishing/exploding gradients problem
- **Batch normalization**
- Dropout

- Additional neuron types:
  - Softmax

Batch normalization

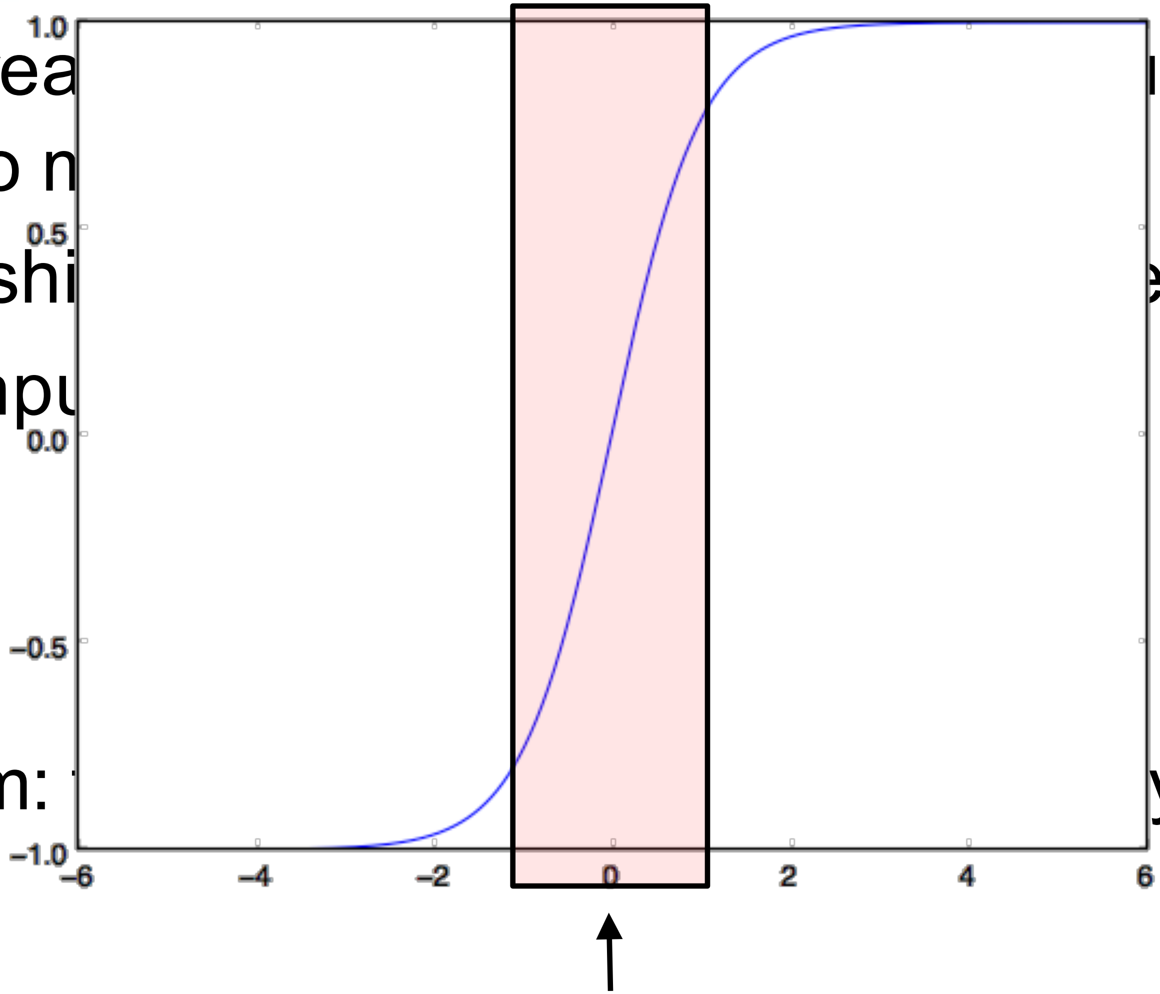
- It would be great if we could just **whiten** the inputs to all neurons in a layer: i.e. zero mean, variance of 1.
  - Avoid vanishing gradients problem, improve learning rates!
  - For each input *k* to the next layer:

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

- Slight problem: this reduces representation ability of network
  - Why?

Batch normalization

- It would be grea… …its to all neurons in a layer: i.e. zero n…
  - Avoid vanishi… …earning rates!
  - For each inpu…
- Slight problem: … …y of network
  - Why?



Get stuck in this part of the activation function

Batch normalization

- First whiten each input *k* independently, using statistics from the mini-batch:

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

- Then introduce parameters to scale and shift each input:

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

- These parameters are learned by the optimization.

Batch normalization

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

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

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

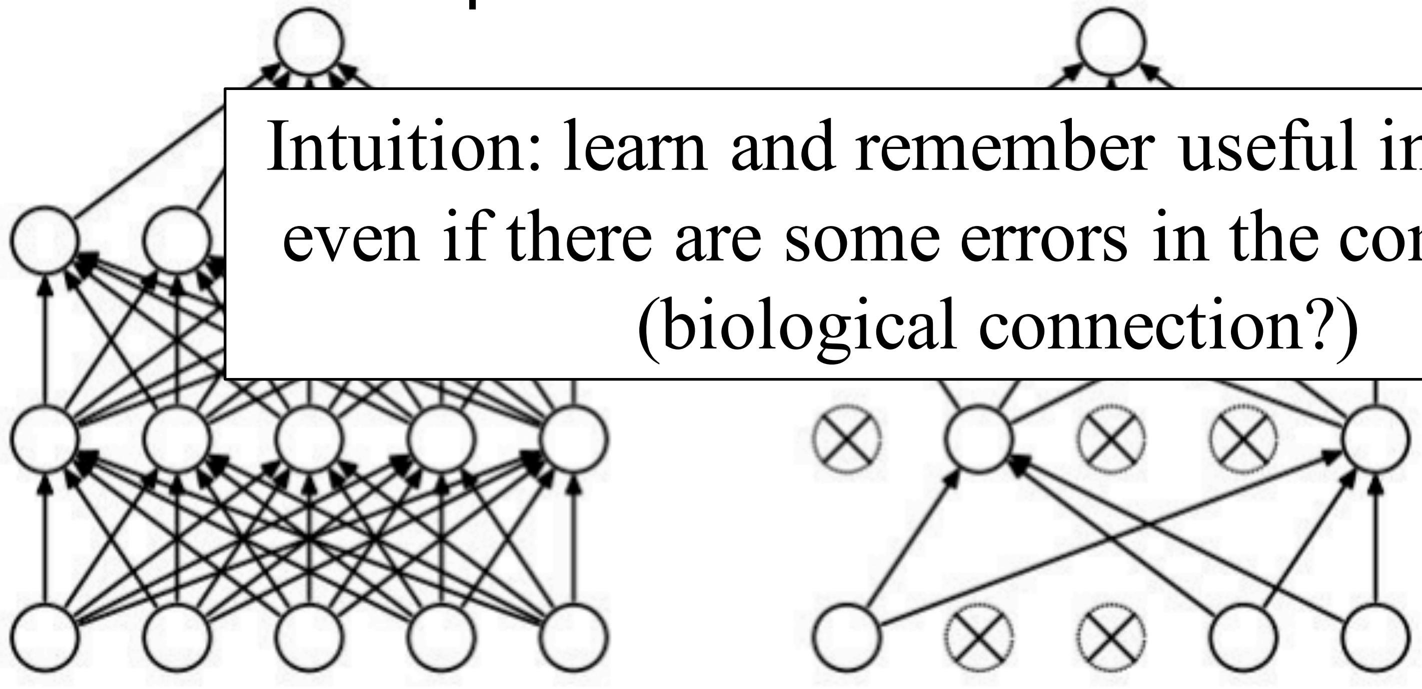
$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m} (x_i - \mu_{\mathcal{B}})^2$$ // mini-batch variance

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

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

Dropout: regularization

- Randomly zero outputs of p fraction of the neurons during training
- Can we learn representations that are robust to loss of neurons?



Intuition: learn and remember useful information even if there are some errors in the computation (biological connection?)

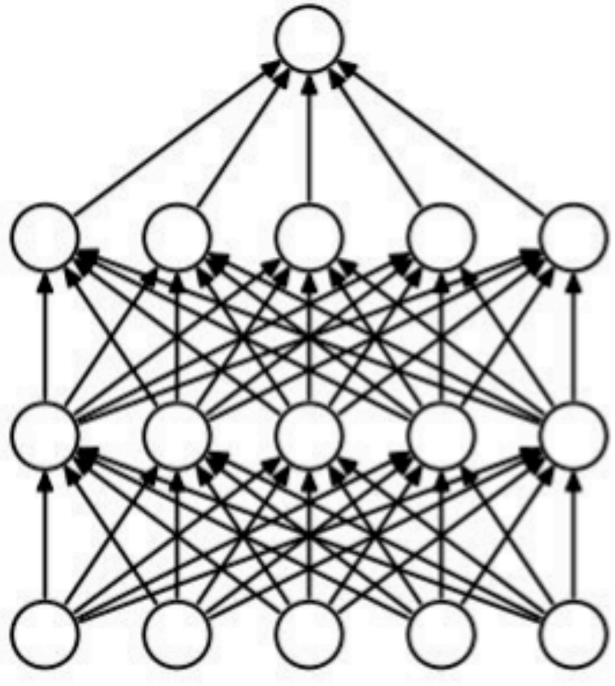
(a) Standard Neural Net (b) After applying dropout.

[Srivastava et al., 2014]

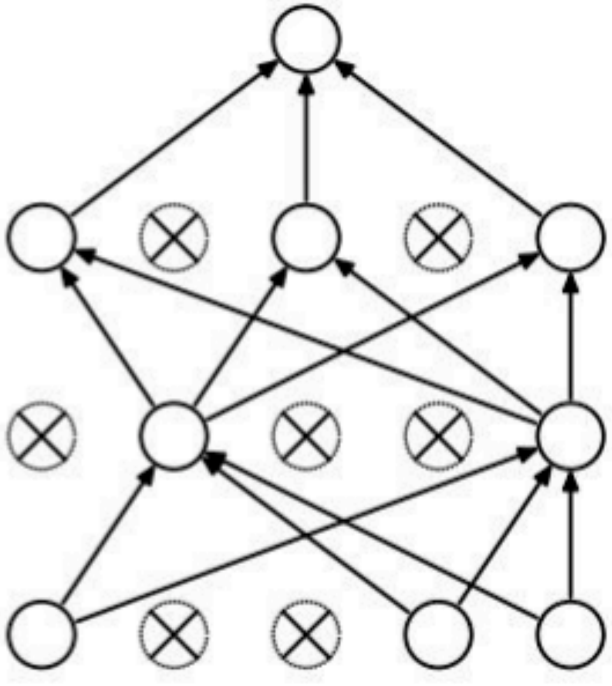
Slide from [Stanford CS231n]

Dropout

- Another interpretation: we are learning a large ensemble of models that share weights.



(a) Standard Neural Net



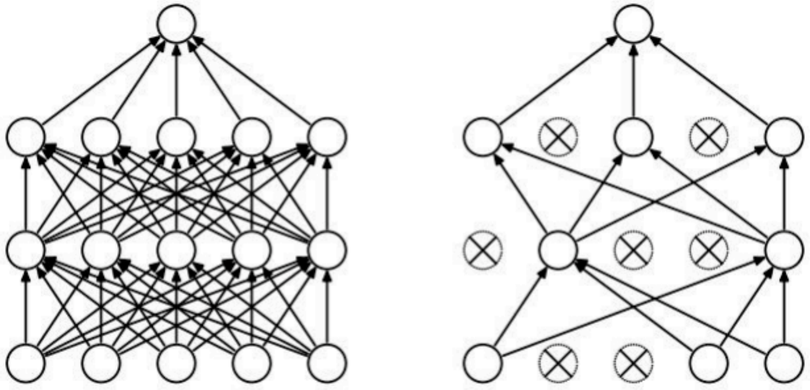
(b) After applying dropout.

[Srivastava et al., 2014]

Slide from Stanford CS231n

Dropout

- Another interpretation: we are learning a large ensemble of models that share weights.
- What can we do during testing to correct for the dropout process?
  - Multiply all neurons outputs by *p*.
  - Or equivalently (**inverse dropout**) simply divide all neurons outputs by *p* during training.



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

[Srivastava et al., 2014]

Slide from Stanford CS231n

Overview

- Preprocessing
- Initialization
- Vanishing/exploding gradients problem
- Batch normalization
- Dropout

- **Additional neuron types:**
  - Softmax

Softmax

- Often used in final output layer to convert neuron outputs into a class probability scores that sum to 1.
- For example, might want to convert the final network output to:
  - P(dog) = 0.2 (Probabilities in range [0, 1])
  - P(cat) = 0.8
  - (Sum of all probabilities is 1).

Softmax

- Softmax takes a vector **z** and outputs a vector of the same length.

$$\sigma(\mathbf{z})_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}} \quad \text{for } j = 1, \ldots, K.$$

$$\frac{\partial}{\partial q_k}\sigma(\mathbf{q}, i) = \cdots = \sigma(\mathbf{q}, i)(\delta_{ik} - \sigma(\mathbf{q}, k))$$