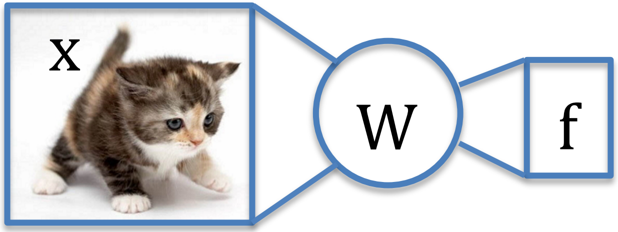
Lecture 7 Recap

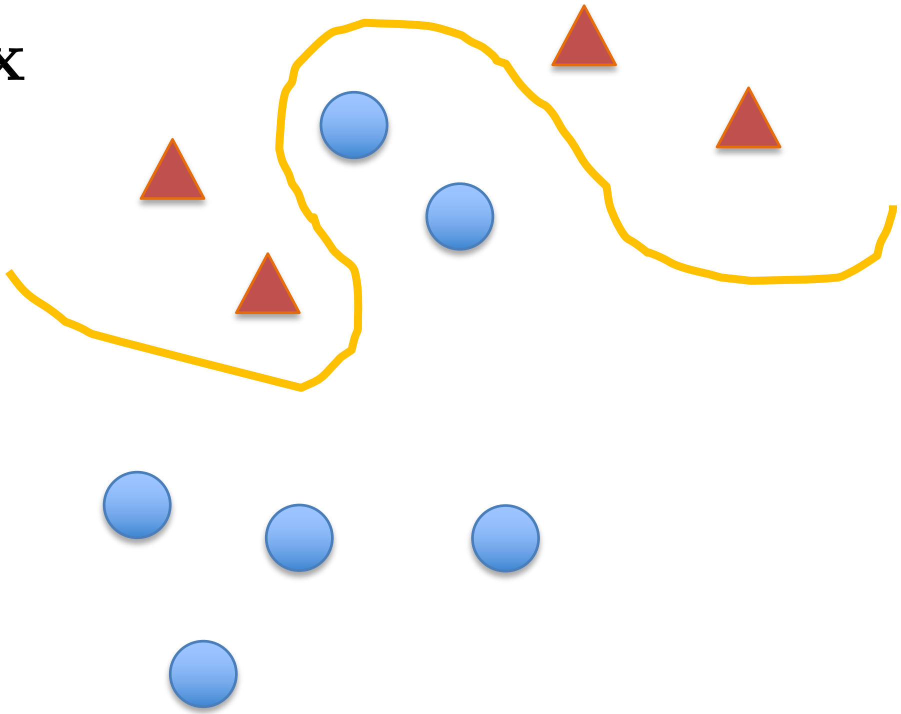
Beyond linear

1-layer network: $f = \mathbf{W}\mathbf{x}$

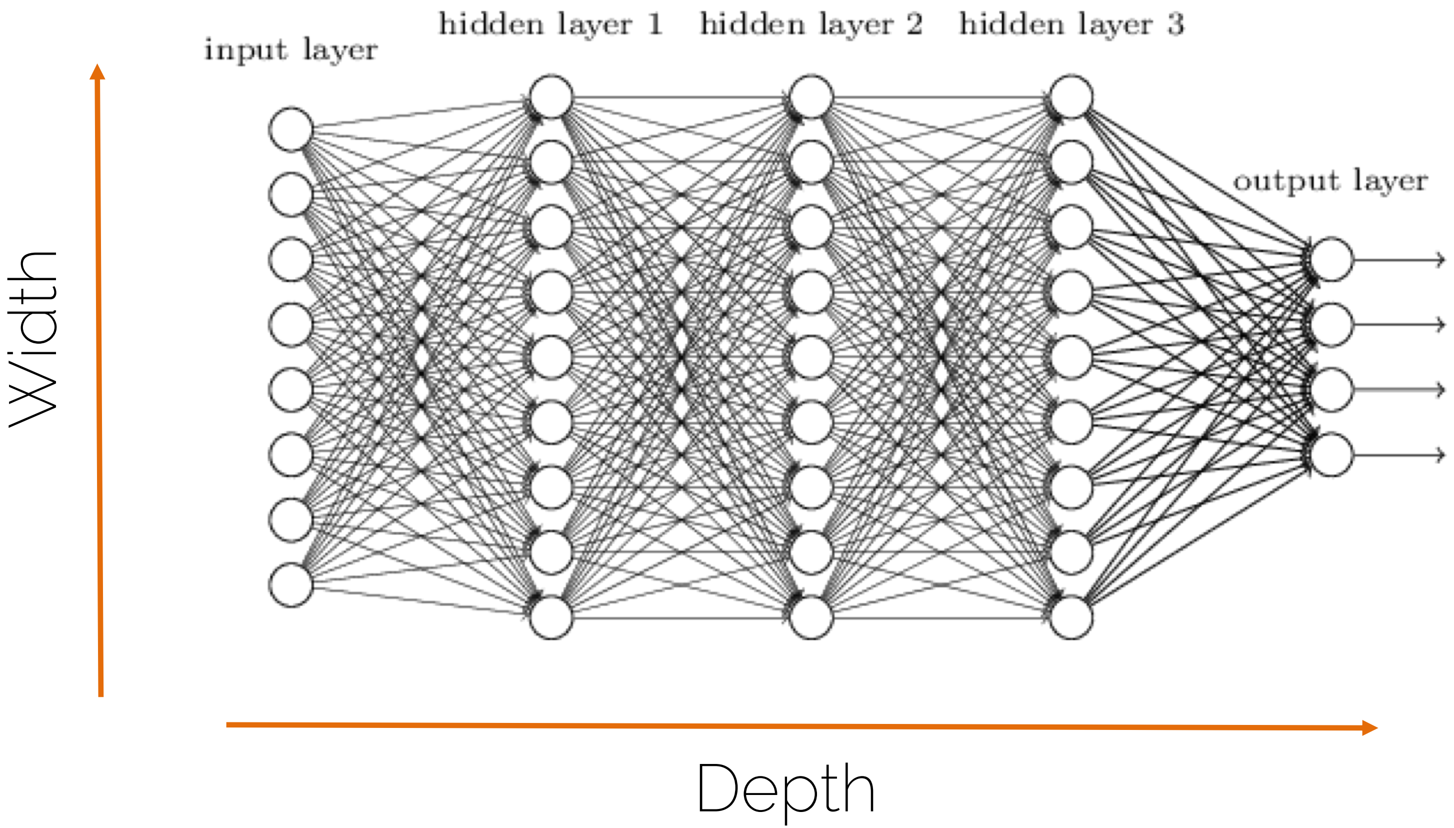


128×128

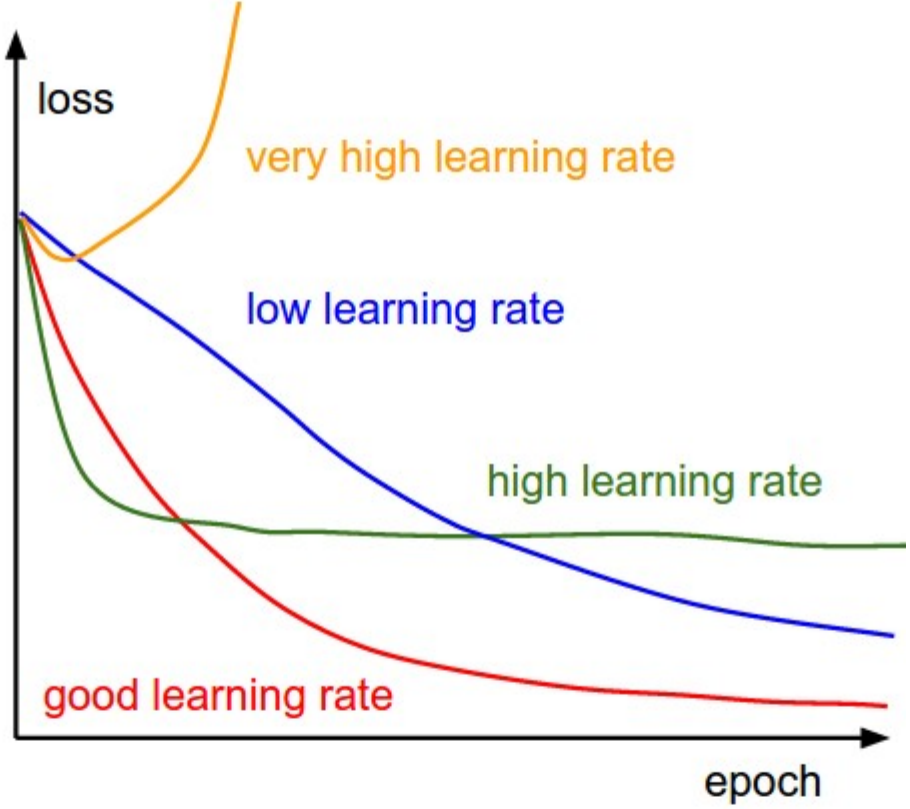
10



Neural Network



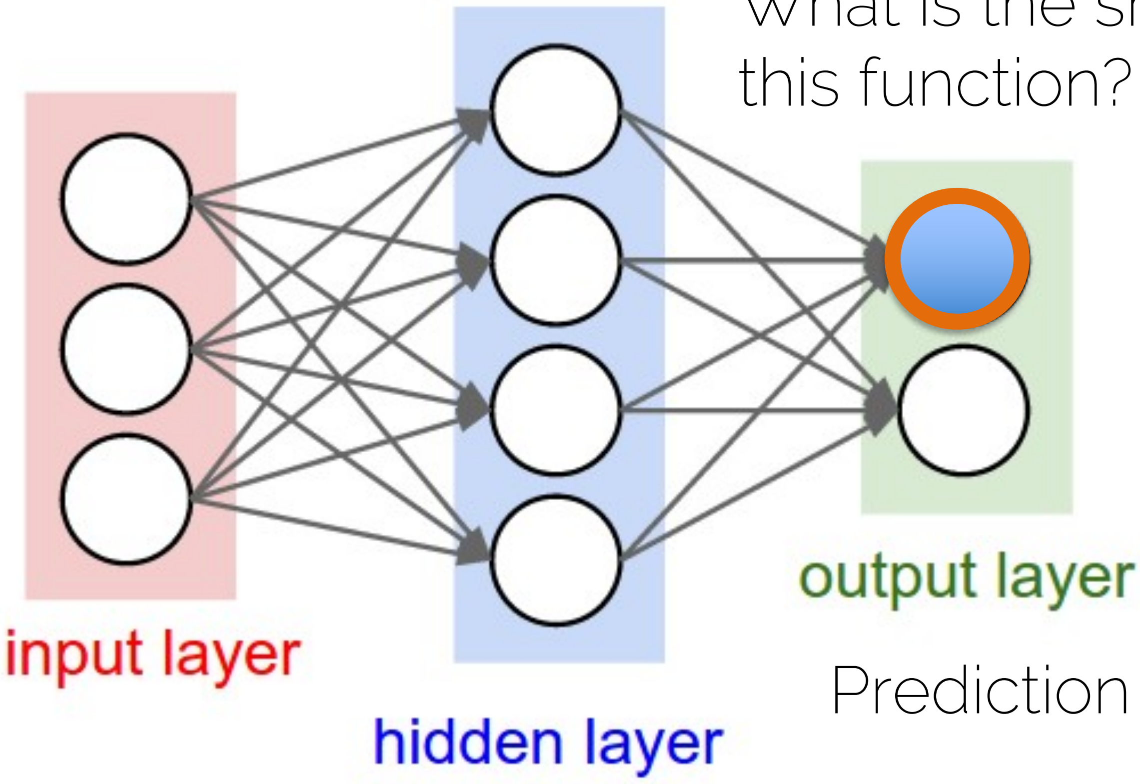
Optimization



Loss functions

Neural networks

What is the shape of this function?

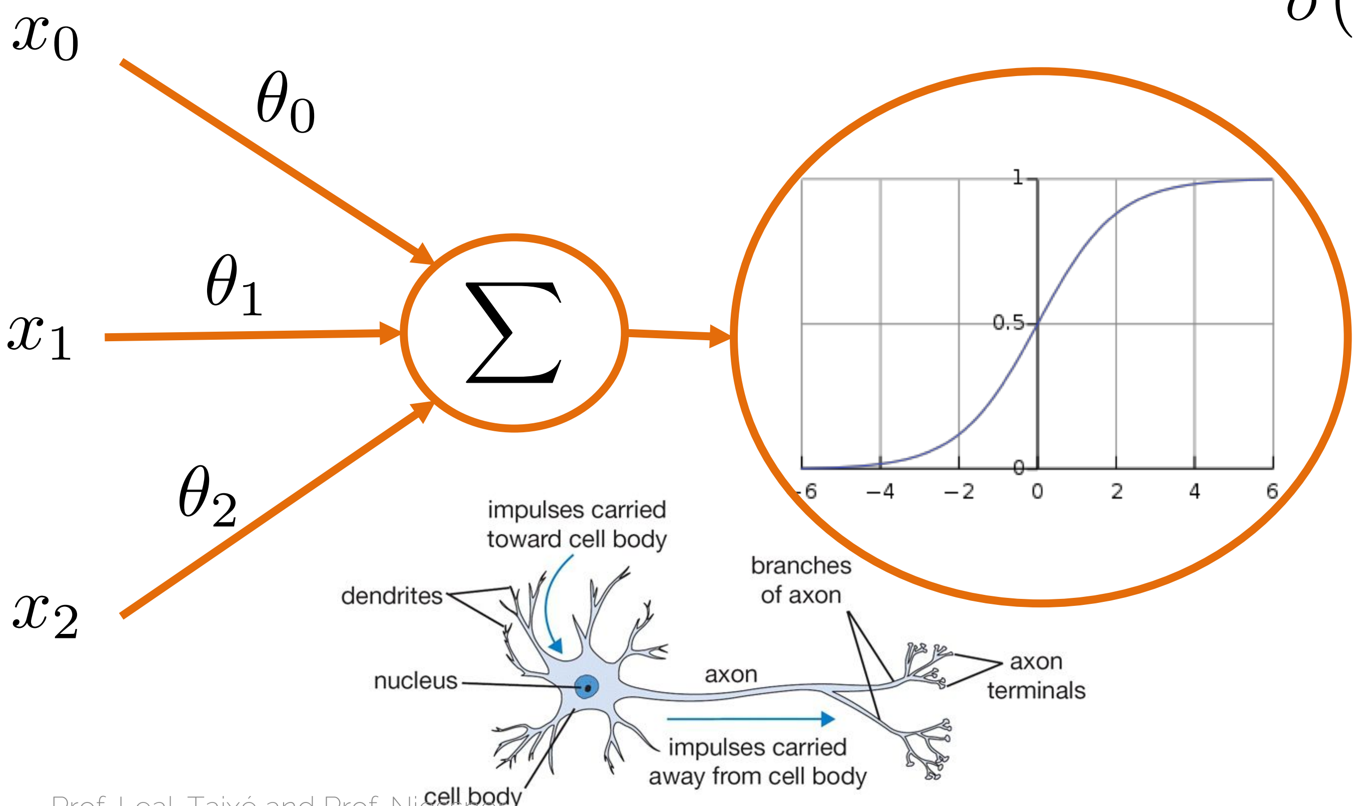


Loss (Softmax, Hinge)

Prediction

Sigmoid for binary predictions

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



1

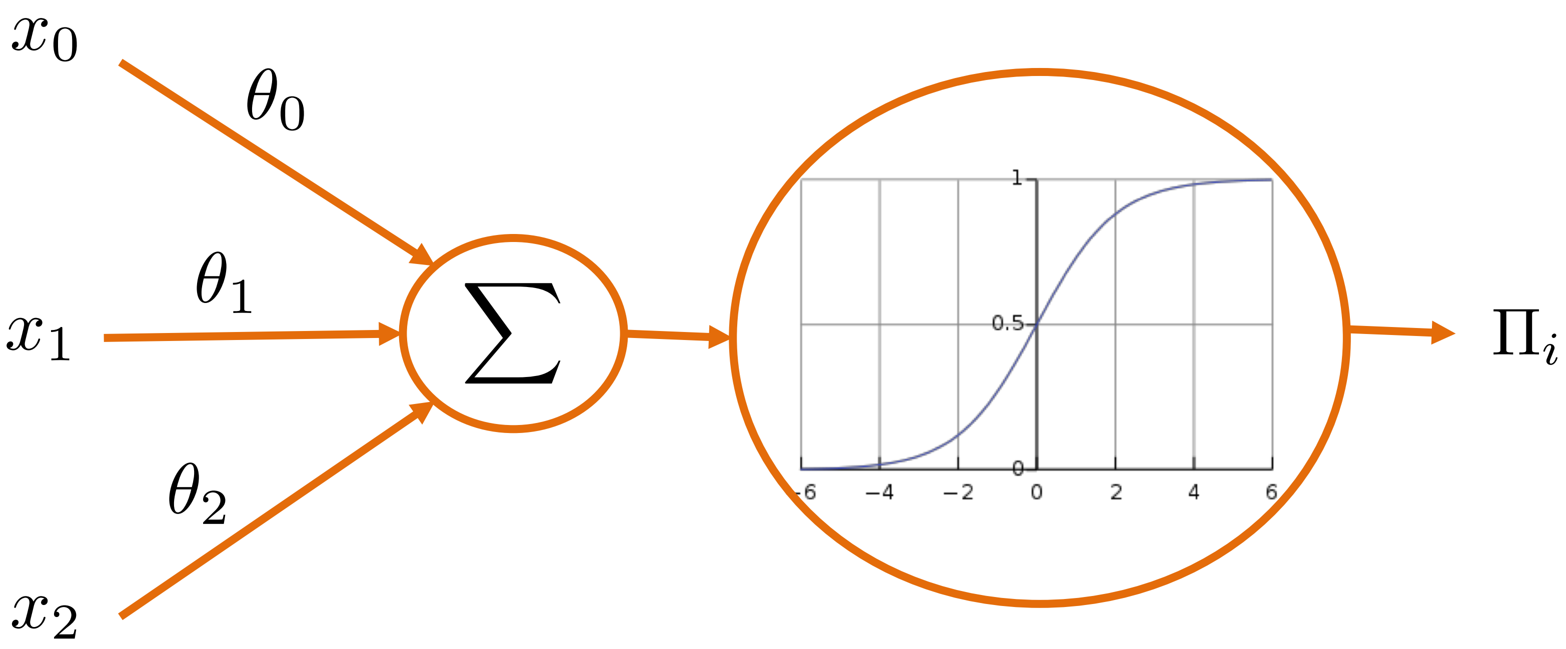
Can be interpreted as a probability

0

$$p(y_i = 1|\mathbf{x}_i, \boldsymbol{\theta})$$

Logitic regression

- Binary classification



Logistic regression

- Loss function

$$\mathcal{L}(\Pi_i, y_i) = y_i \log \Pi_i + (1 - y_i) \log(1 - \Pi_i)$$

One training sample

- Cost function

$$C(\boldsymbol{\theta}) = -\frac{1}{n} \sum_{i=1}^{n} y_i \log \Pi_i + (1 - y_i) \log(1 - \Pi_i)$$

Minimization

$$\sigma(\mathbf{x}_i \boldsymbol{\theta})$$

Softmax regression

- Cost function for the binary case

$$C(\boldsymbol{\theta}) = -\frac{1}{n}\sum_{i=1}^{n} y_i \log \Pi_i + (1 - y_i)\log(1 - \Pi_i)$$

- Extension to multiple classes

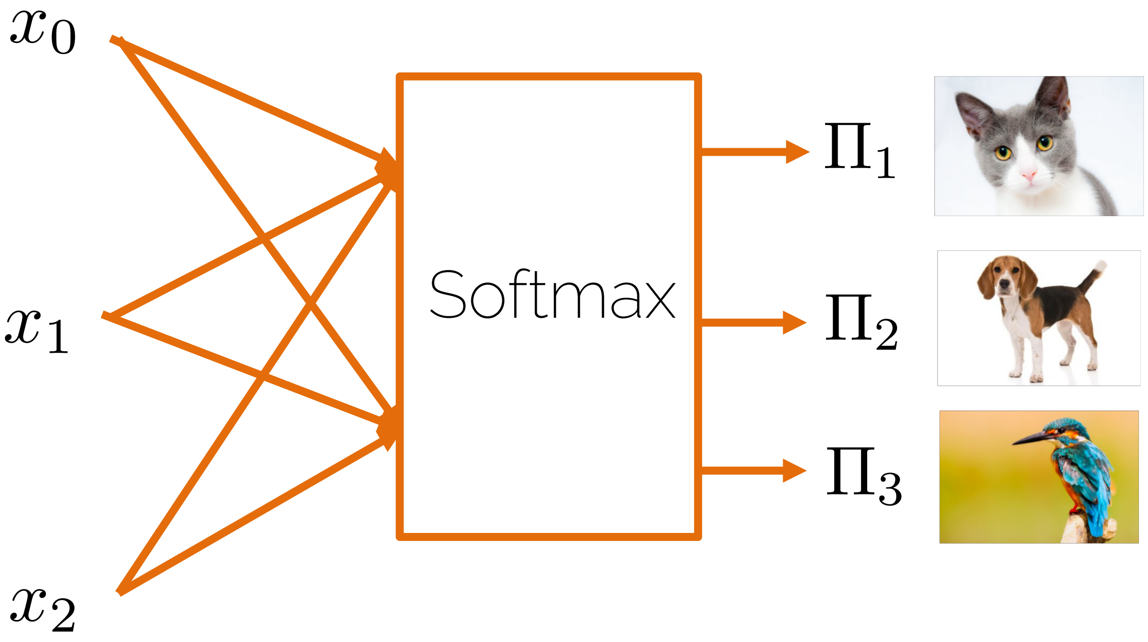
$$C(\boldsymbol{\theta}) = -\frac{1}{n}\sum_{i=1}^{n}\sum_{c=1}^{M} y_{i,c} \log p_{i,c}$$

Probability given by our sigmoid function

Binary indicator whether c is the label for image i

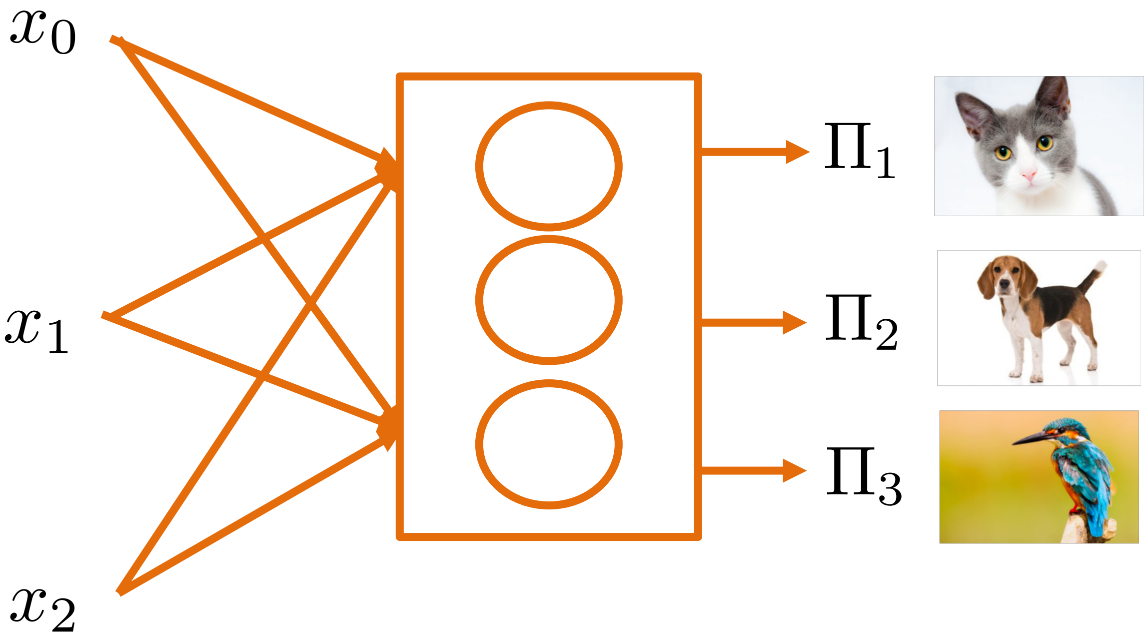
Softmax formulation

- What if we have multiple classes?



Softmax formulation

- Three neurons in the output layer for three classes



Softmax formulation

- What if we have multiple classes?

$$C(\boldsymbol{\theta}) = -\frac{1}{n}\sum_{i=1}^{n}\sum_{c=1}^{M} y_{i,c}\log p_{i,c}$$

- You can no longer assign $p_{i,c}$ to. Π_i as in the binary case, because all outputs need to sum to 1

$$\sum_{c} \Pi_{i,c}$$

Softmax formulation

Π_1  $p(cat|\mathbf{X}_i) = \frac{e^{s_{cat}}}{\sum_c e^{s_c}}$

Score for class cat given by all the layers of the network

Π_2  $p(dog|\mathbf{X}_i)$

Normalization

Π_3  $p(bird|\mathbf{X}_i)$

- Softmax takes M inputs (Scores) and outputs M probabilities (M is the number of classes)

Loss functions

- Softmax loss function

$$L_i = -\log\left(\frac{e^{s_{y_i}}}{\sum_k e^{s_k}}\right)$$

Evaluate the ground truth score for the image

Comes from Maximum Likelihood Estimate

- Hinge Loss (derived from the Multiclass SVM loss)

$$L_i = \sum_{k \neq y_i} \max(0, s_k - s_{y_i} + 1)$$

Loss functions

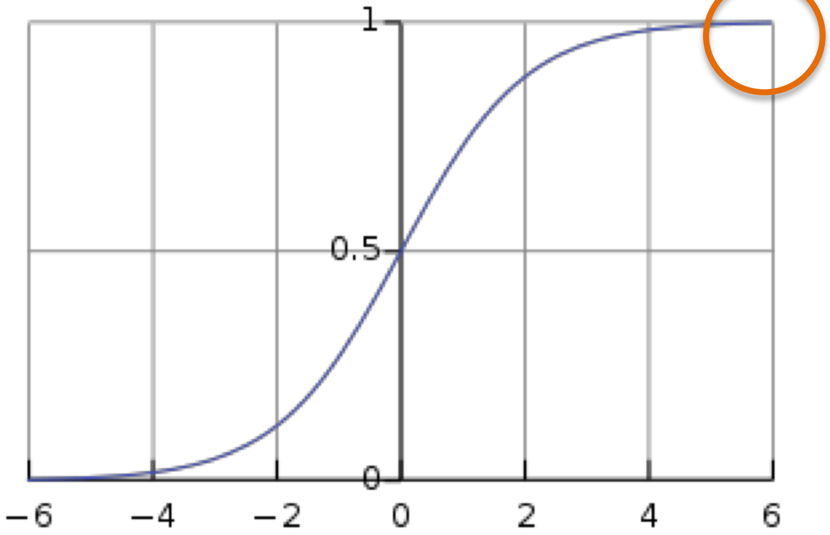
- Softmax loss function
  - Optimizes until the loss is zero
- Hinge Loss (derived from the Multiclass SVM loss)
  - Saturates whenever it has learned a class “well enough”

Activation functions

Sigmoid

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

Forward

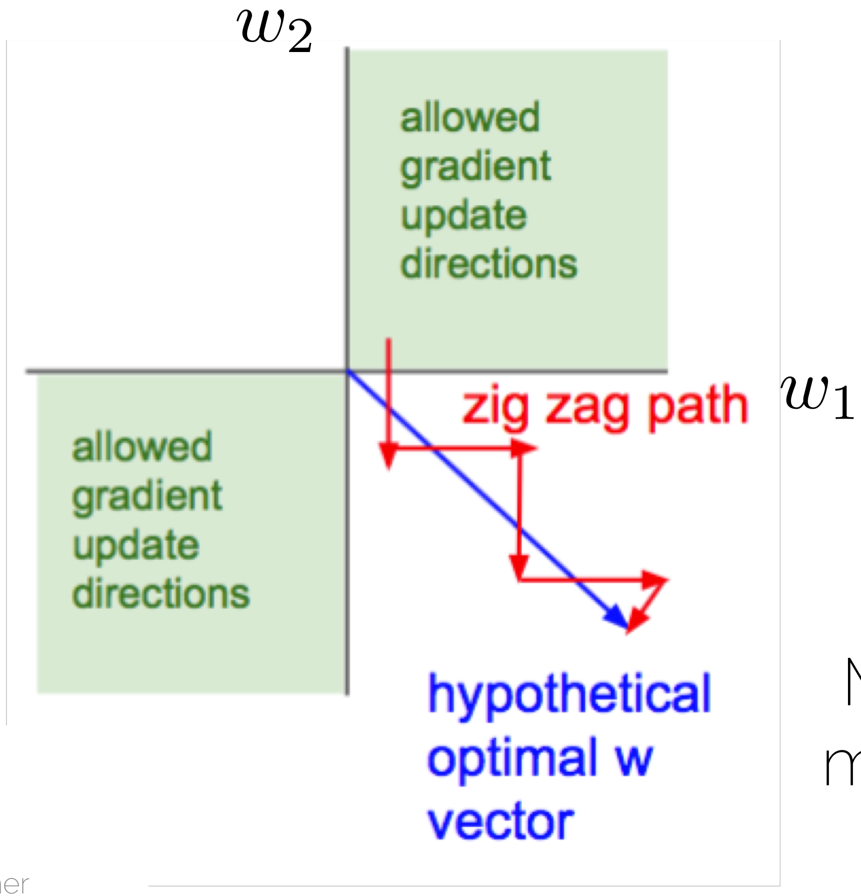


$x = 6$

Saturated neurons kill the gradient flow

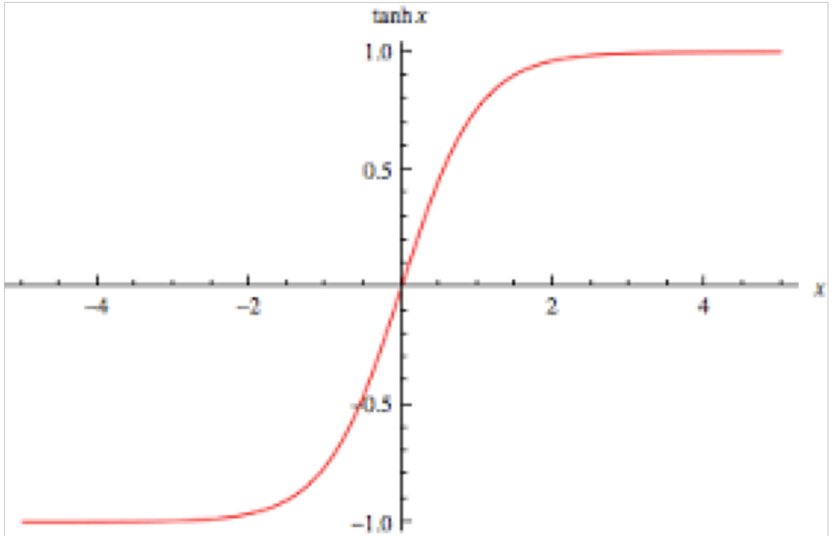
$$\frac{\partial L}{\partial x} = \frac{\partial \sigma}{\partial x}\frac{\partial L}{\partial \sigma} \qquad \frac{\partial \sigma}{\partial x} \qquad \frac{\partial L}{\partial \sigma}$$

Problem of positive output



More on zero-mean data later

tanh



✗ Still saturates

✓ Zero-centered

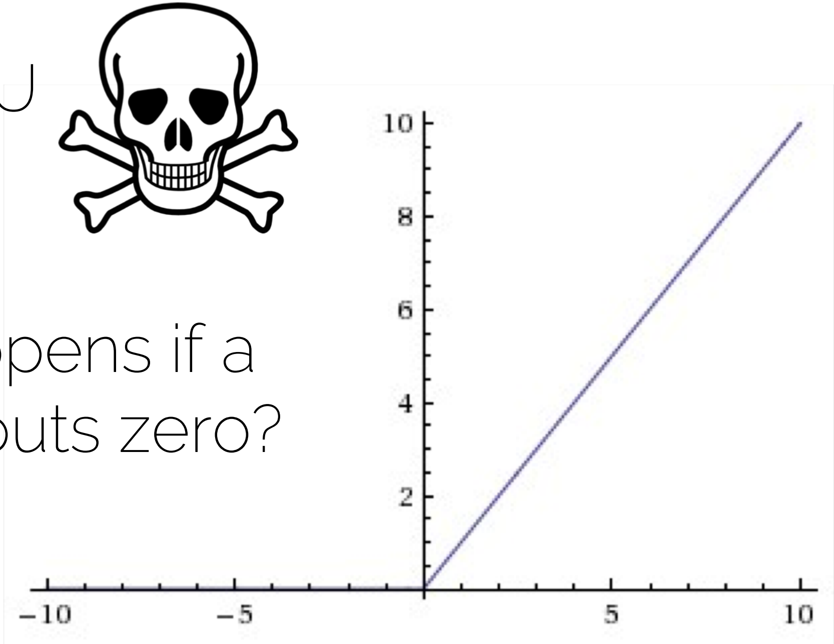
✗ Still saturates

LeCun 1991

Rectified Linear Units (ReLU)

✗ Dead ReLU

What happens if a ReLU outputs zero?

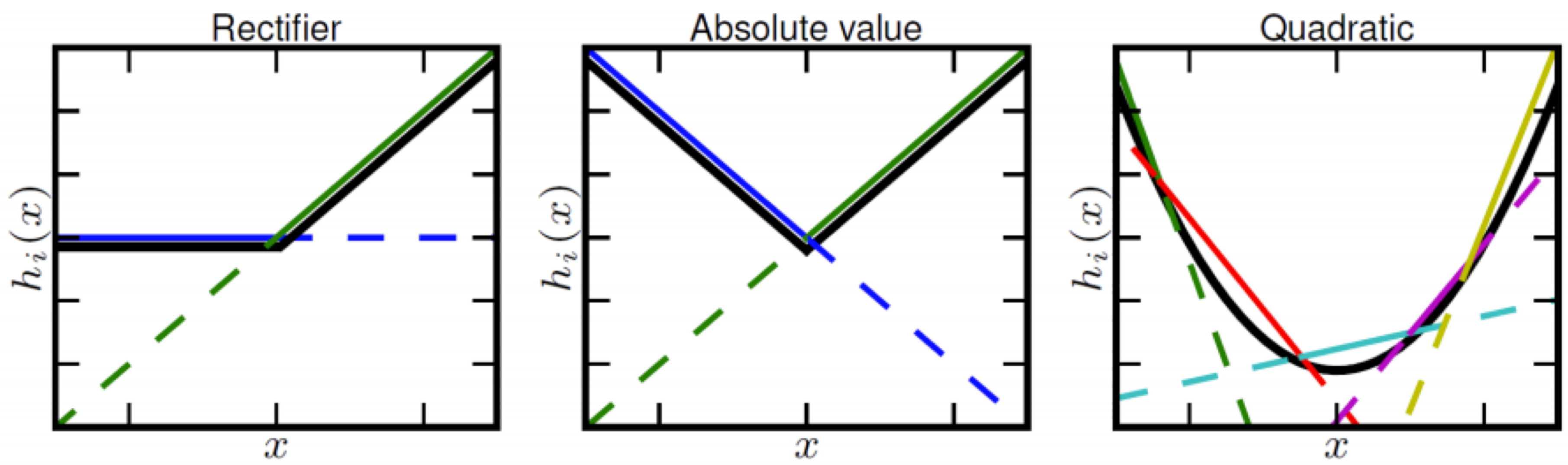


Large and consistent gradients ✓

✓ Fast convergence

✓ Does not saturate

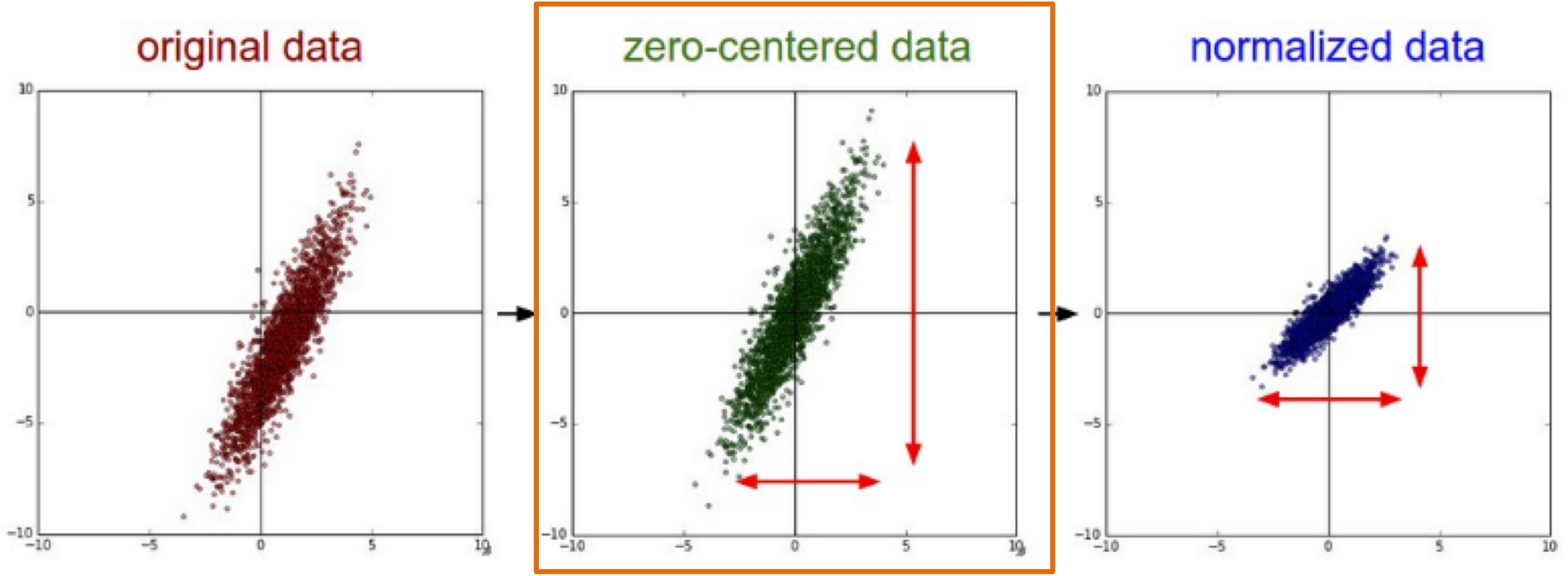
Maxout units



✓ Generalization of ReLUs ✓ Linear regimes ✓ Does not die ✓ Does not saturate

✗ Increase of the number of parameters

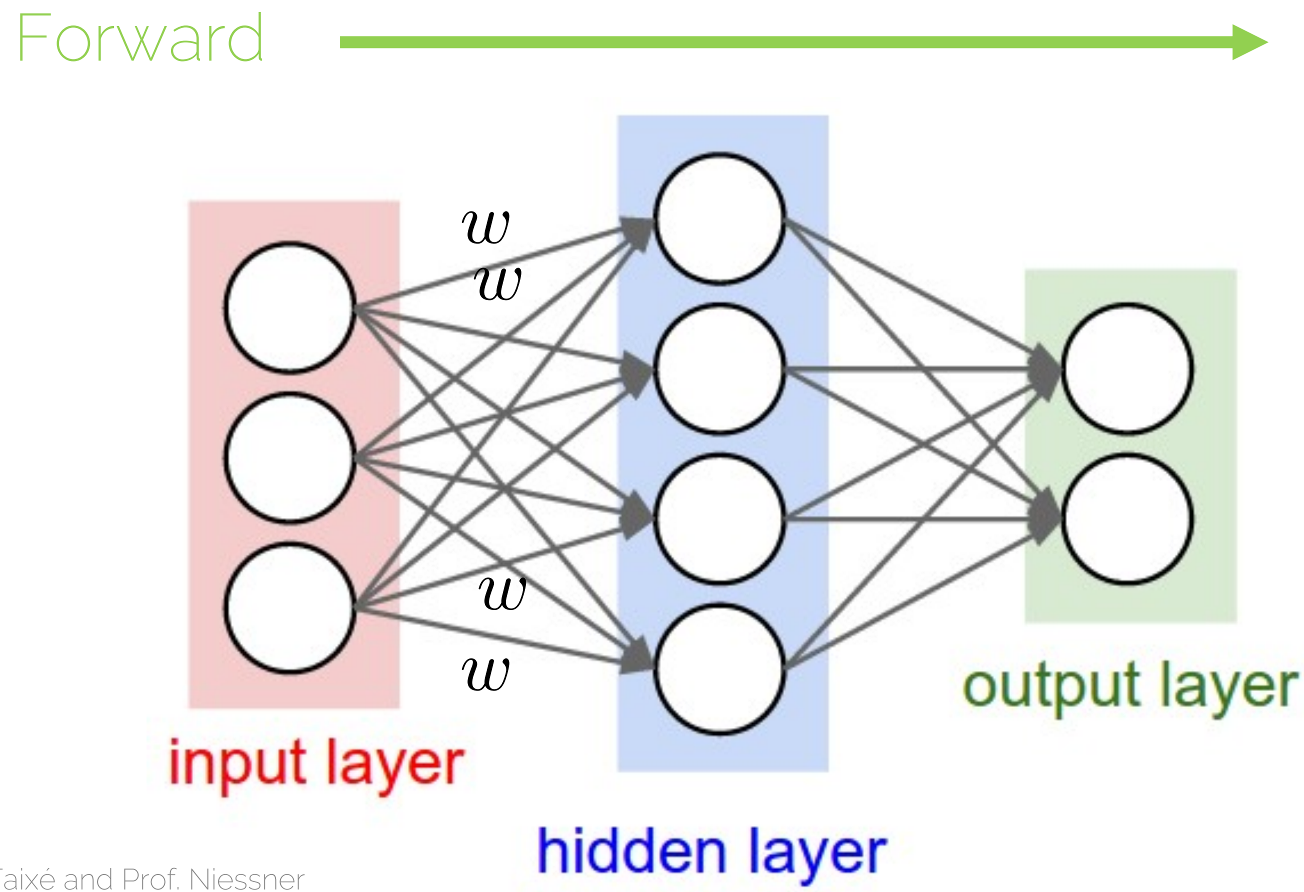
Data pre-processing



For images subtract the mean image (AlexNet) or per-channel mean (VGG-Net)

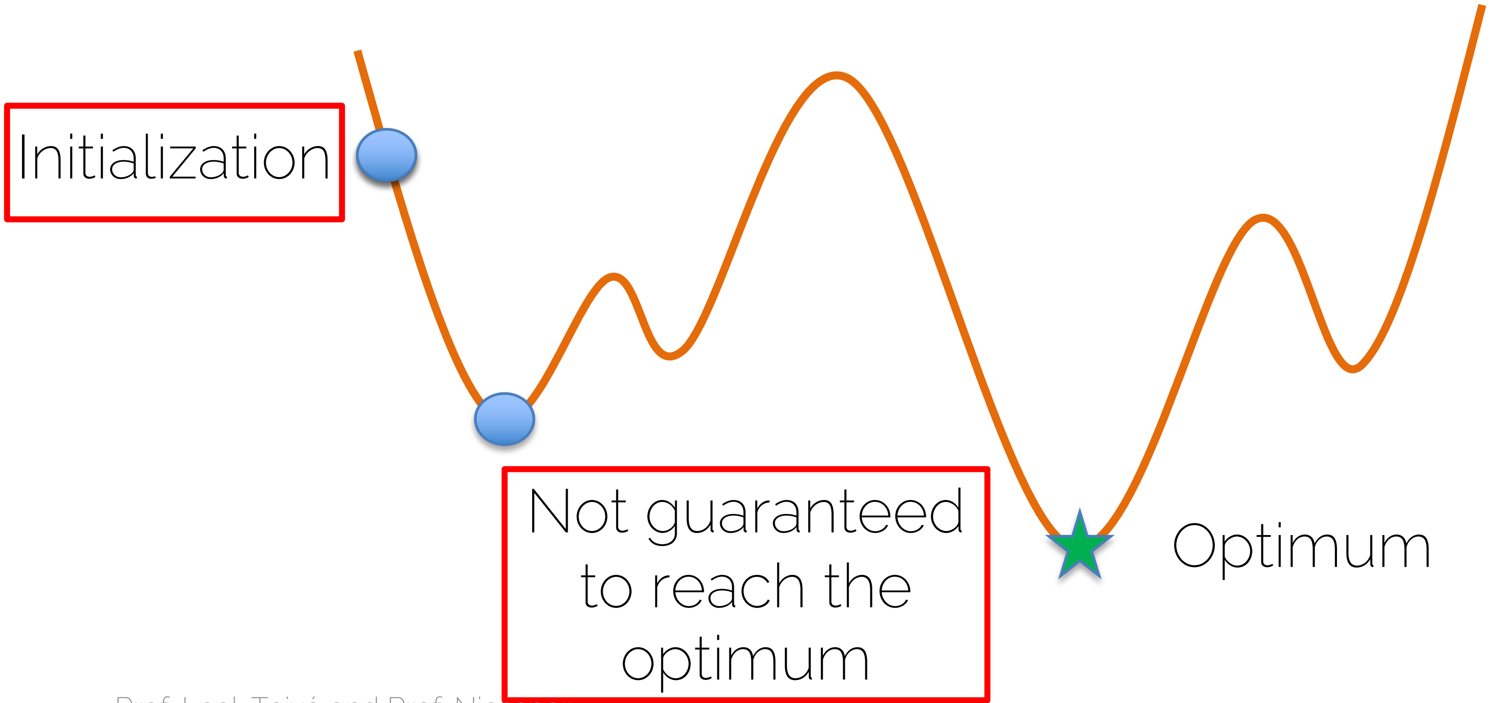
Weight initialization

How do I start?



Initialization is extremely important

$$\mathbf{x}^* = \arg\min f(\mathbf{x})$$

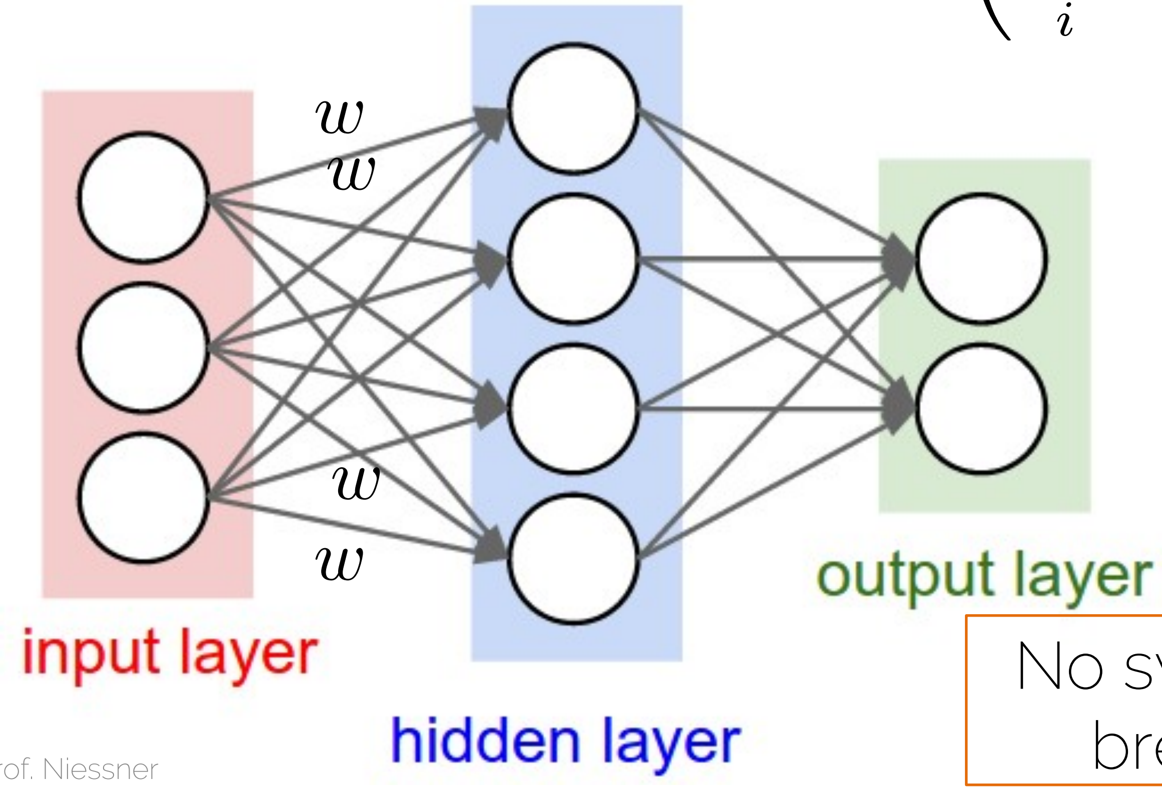


How do I start?

Forward $\longrightarrow$ $f\left(\sum_i w_i x_i + b\right)$

$w = 0$

What happens to the gradients?



No symmetry breaking

All weights to zero

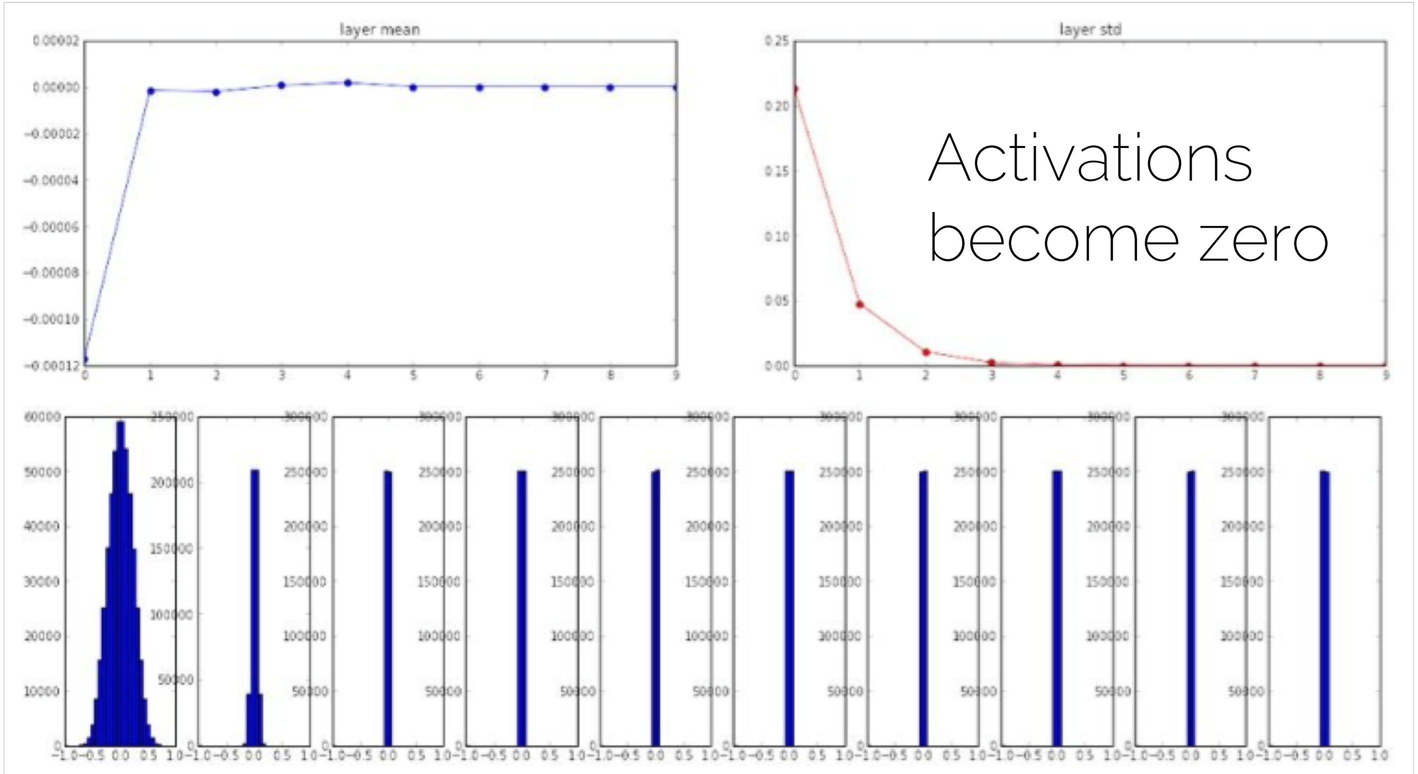
- Elaborate: the hidden units are all going to compute the same function, gradients are going to be the same

Small random numbers

- Gaussian with zero mean and standard deviation 0.01

- Let us see what happens:
  - Network with 10 layers with 500 neurons each
  - Tanh as activation functions
  - Input unit Gaussian data

Small random numbers



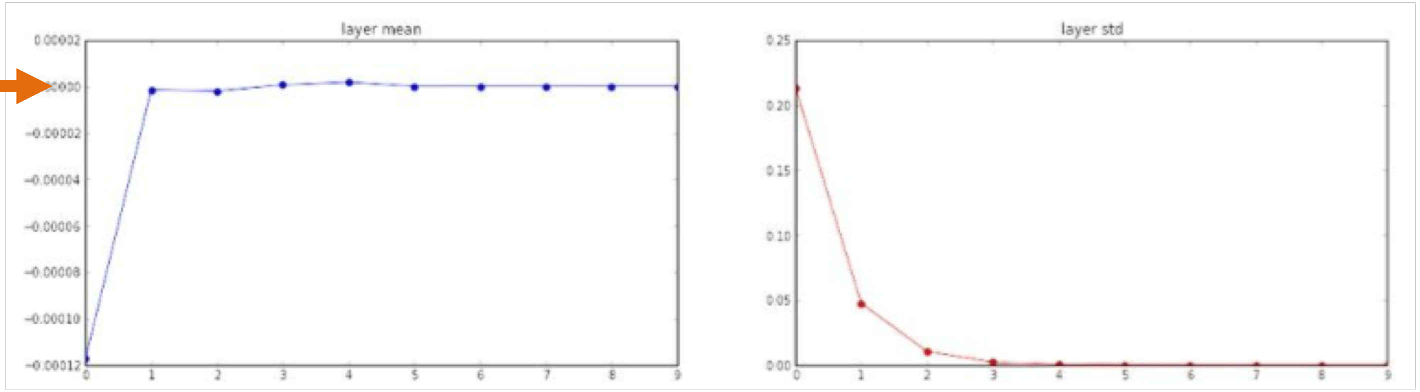
Activations become zero

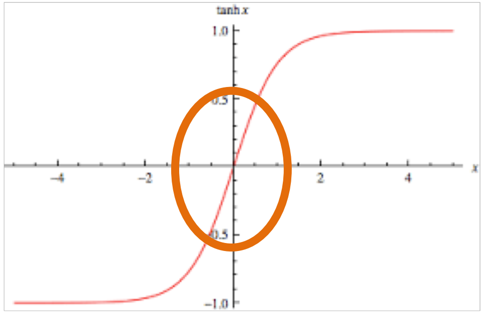
Input

Last layer

Forward

Small random numbers



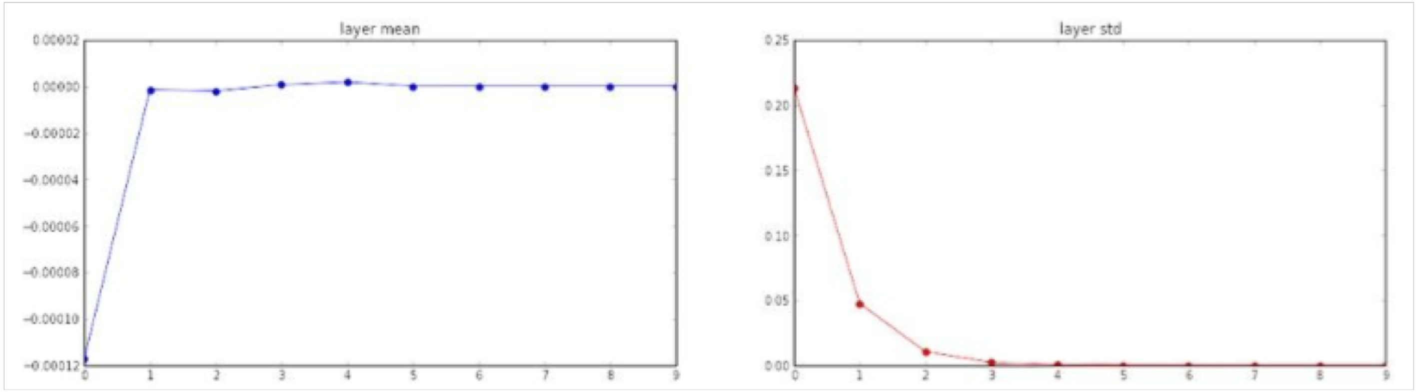


small

$$f\left(\sum_i w_i x_i + b\right)$$

Forward

Small random numbers



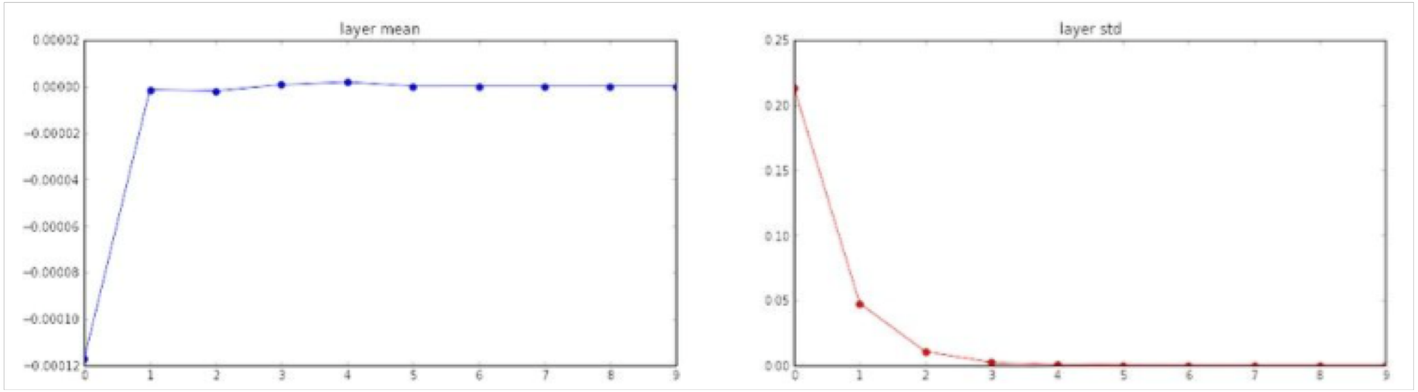
1. Activation function gradient is ok

$$f\left(\sum_i \overbrace{w_i x_i + b}\right)$$

2. Compute the gradients wrt the weights

Backward ←

Small random numbers



1. Activation function gradient is ok

$$f\left(\overbrace{\sum_i w_i x_i}+b\right)$$

2. Compute the gradients wrt the weights

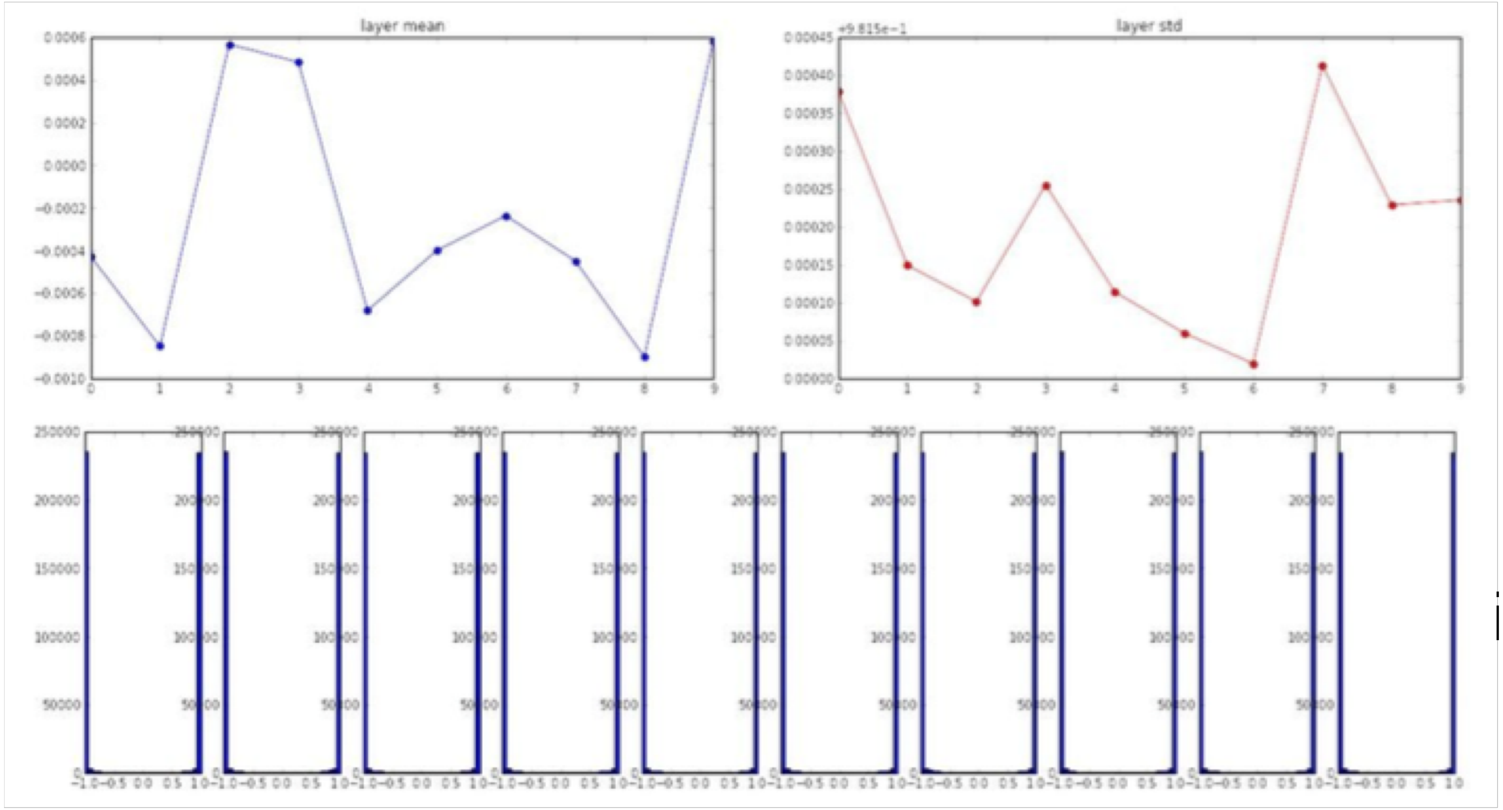
Gradients vanish

Big random numbers

- Gaussian with zero mean and standard deviation 1

- Let us see what happens:
  - Network with 10 layers with 500 neurons each
  - Tanh as activation functions
  - Input unit Gaussian data

Big random numbers



Everything is saturated

How to solve this?

- Working on the initialization

- Working on the output generated by each layer

Xavier initialization

- Gaussian with zero mean, but what standard deviation?

$$\mathrm{Var}(s) = \mathrm{Var}(\sum_{i}^{n} w_i x_i) = \sum_{i}^{n} \mathrm{Var}(w_i x_i)$$

Glorot 2010

Xavier initialization

- Gaussian with zero mean, but what standard deviation?

$$\mathrm{Var}(s) = \mathrm{Var}\left(\sum_i^n w_i x_i\right) = \sum_i^n \mathrm{Var}(w_i x_i)$$

Independent

$$= \sum_i^n [E(w_i)]^2 \mathrm{Var}(x_i) + E[(x_i)]^2 \mathrm{Var}(w_i) + \mathrm{Var}(x_i)\mathrm{Var}(w_i)$$

Zero mean

Xavier initialization

- Gaussian with zero mean, but what standard deviation?

$$\mathrm{Var}(s) = \mathrm{Var}(\sum_i^n w_i x_i) = \sum_i^n \mathrm{Var}(w_i x_i)$$

$$= \sum_i^n [E(w_i)]^2 \mathrm{Var}(x_i) + E[(x_i)]^2 \mathrm{Var}(w_i) + \mathrm{Var}(x_i)\mathrm{Var}(w_i)$$

$$= \sum_i^n \mathrm{Var}(x_i)\mathrm{Var}(w_i) = (n\mathrm{Var}(w))\,\mathrm{Var}(x)$$

Identically distributed

Xavier initialization

- Gaussian with zero mean, but what standard deviation?

$$\mathrm{Var}(s) = \mathrm{Var}\left(\sum_i^n w_i x_i\right) = \sum_i^n \mathrm{Var}(w_i x_i)$$

$$= \sum_i^n [E(w_i)]^2 \mathrm{Var}(x_i) + E[(x_i)]^2 \mathrm{Var}(w_i) + \mathrm{Var}(x_i)\mathrm{Var}(w_i)$$

$$= \sum_i^n \mathrm{Var}(x_i)\mathrm{Var}(w_i) = (n\mathrm{Var}(w))\,\mathrm{Var}(x)$$

Variance gets multiplied by the number of inputs

Xavier initialization

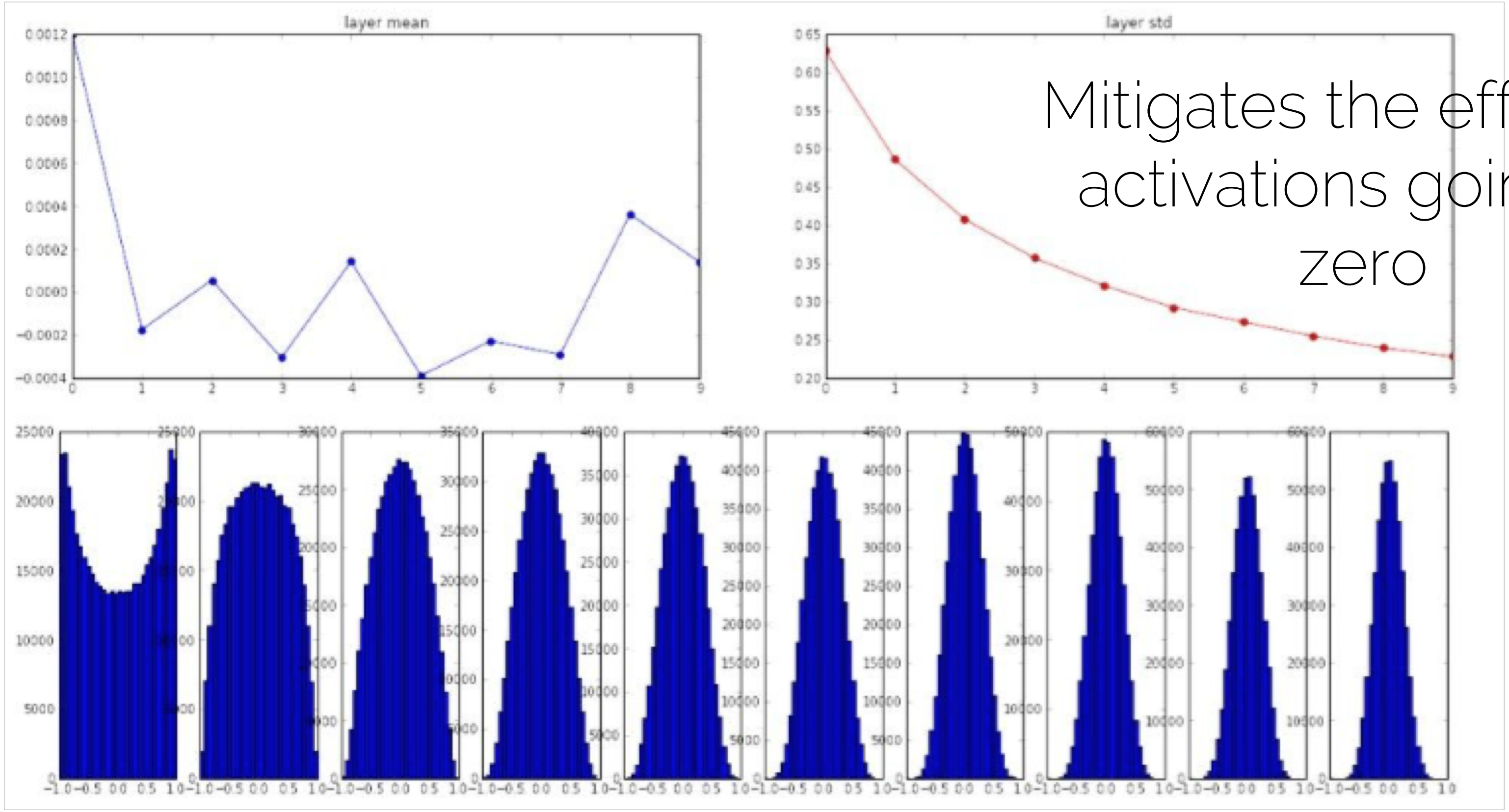
- How to ensure the variance of the output is the same as the input?

$$(n\mathrm{Var}(w))\,\mathrm{Var}(x)$$

$$1$$

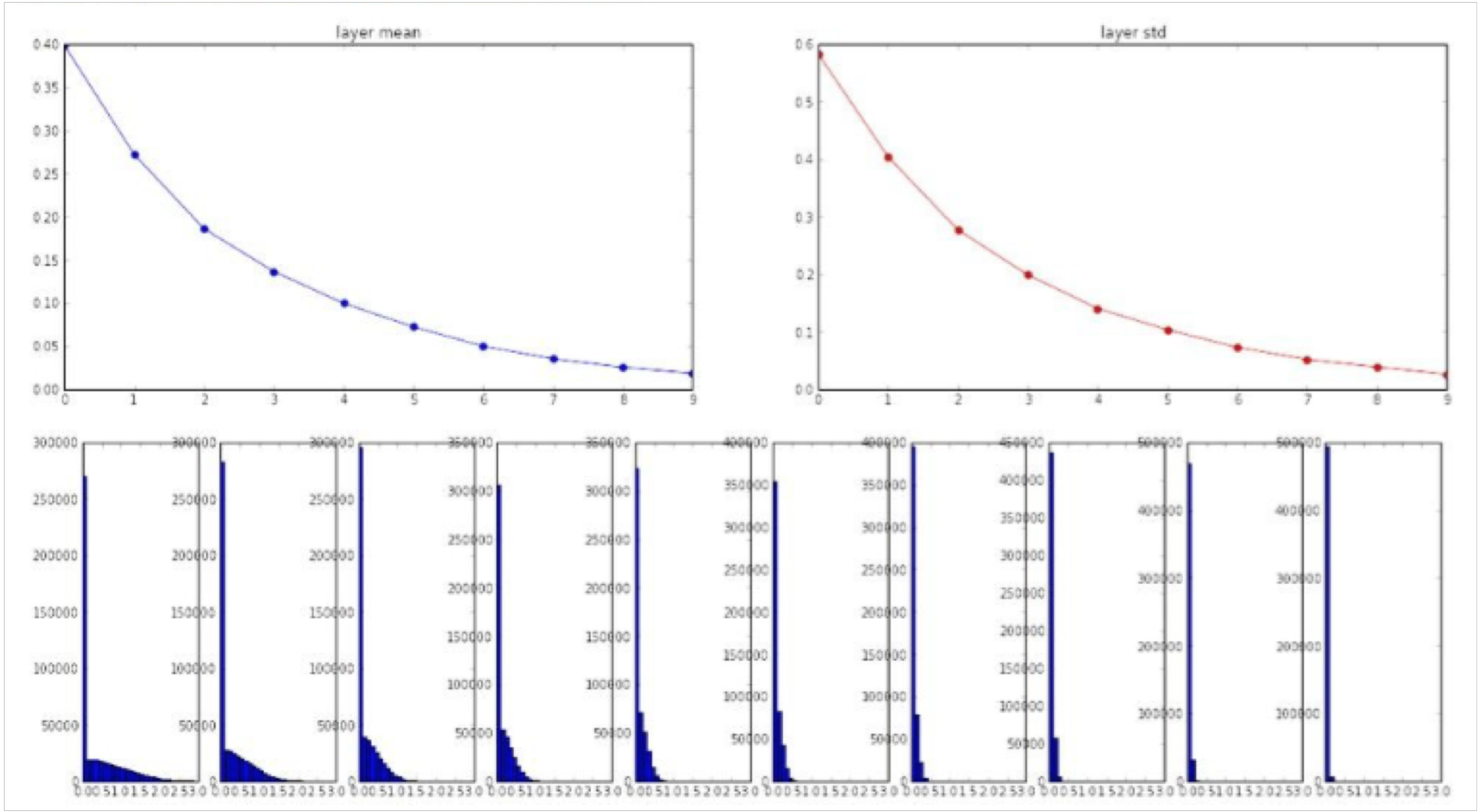
$$\boxed{Var(w) = \frac{1}{n}}$$

Xavier initialization



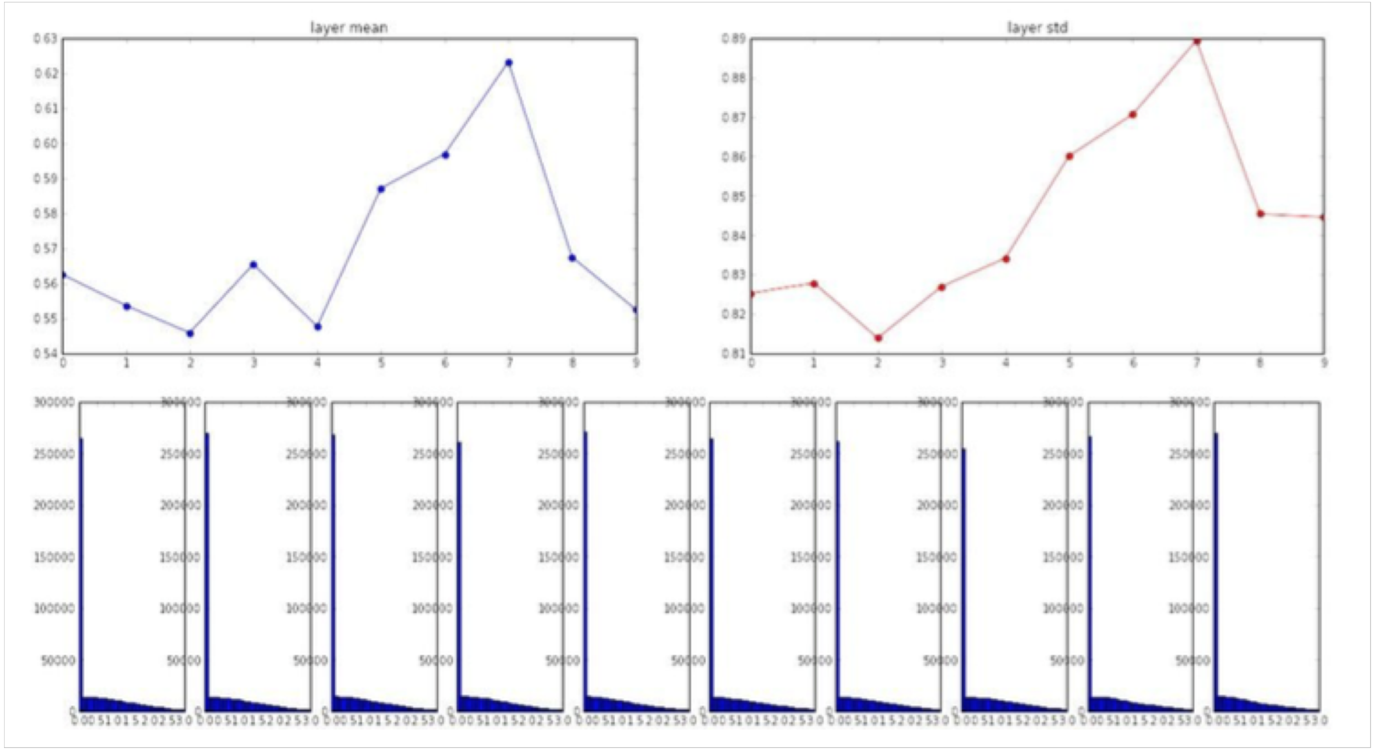
Mitigates the effect of activations going to zero

Xavier initialization with ReLU



ReLU kills half of the data

$$Var(w) = \frac{2}{n}$$

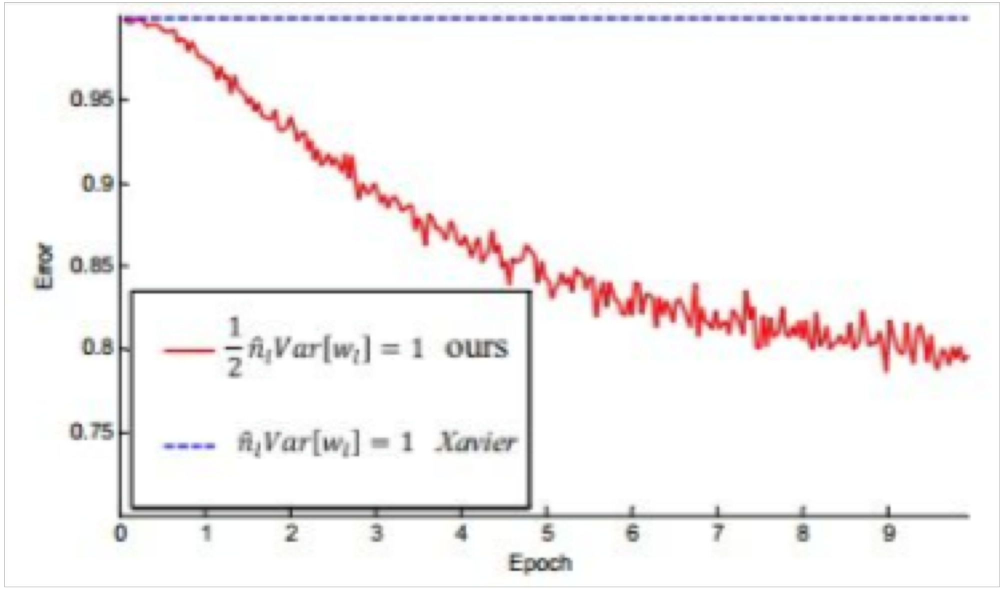


He 2015

ReLU kills half of the data

$$Var(w) = \frac{2}{n}$$

It makes a huge difference!



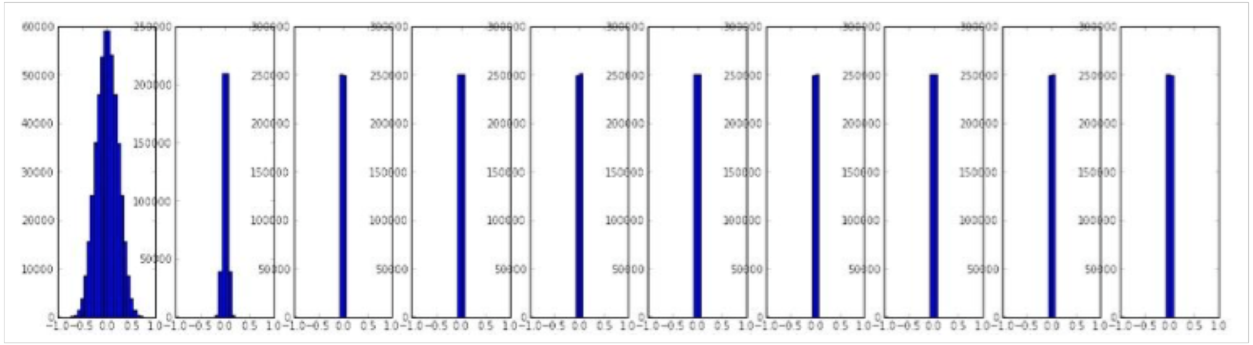
Tips and tricks

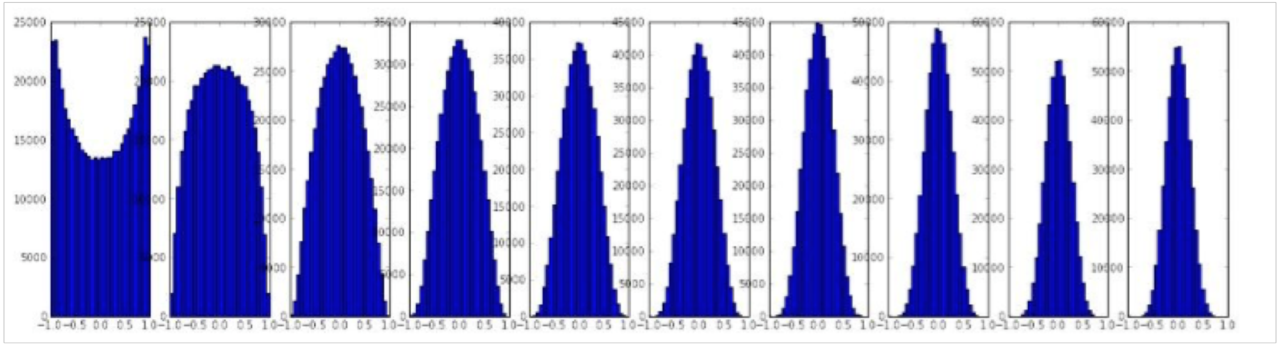
- Use ReLU and Xavier/2 initialization

Batch normalization

Our goal

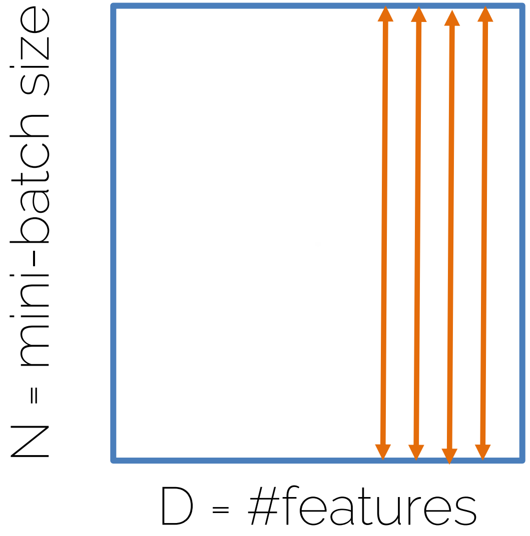
- All we want is that our activations do not die out





Batch normalization

- Wish: unit Gaussian activations (in our example)
- Solution: let's do it



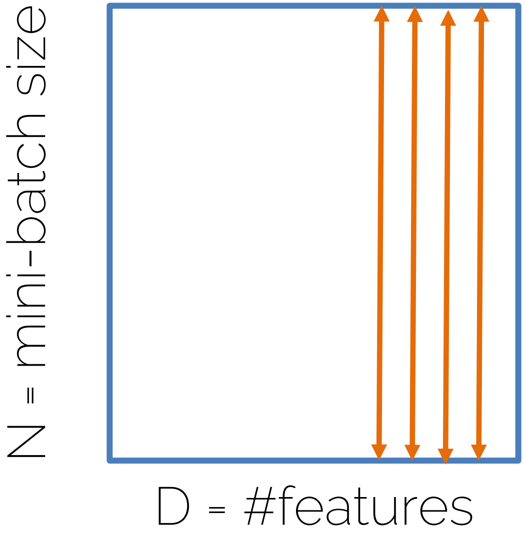
Mean of your mini-batch examples over feature k

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

Ioffe and Szegedy 2015

Batch normalization

- In each dimension of the features, you have a unit gaussian (in our example)



Mean of your mini-batch examples over feature k

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

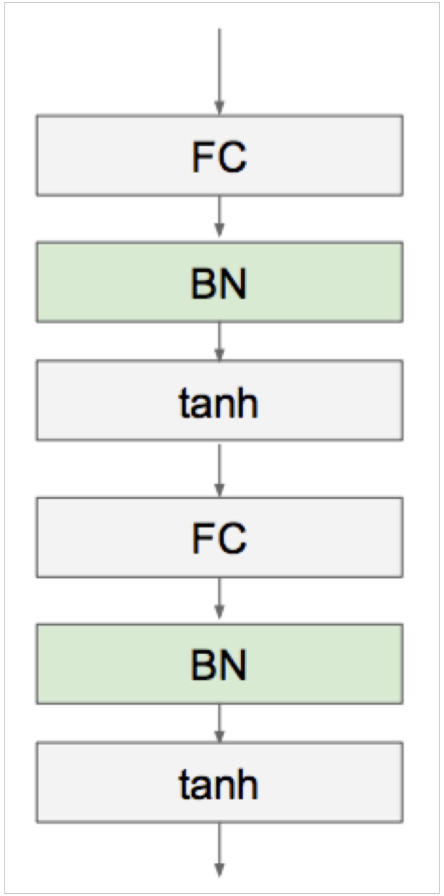
Batch normalization

- In each dimension of the features, you have a unit gaussian (in our example)
- For NN in general → BN normalizes the mean and variance of the inputs to your activation functions

BN layer

- A layer to be applied after Fully Connected (or Convolutional) layers and **before** non-linear activation functions

- Is it a good idea to have all unit Gaussians before tanh? This normalization might not be the best for the network!



Batch normalization

- 1. Normalize

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

Differentiable function so we can backprop through it….

- 2. Allow the network to change the range

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

These parameters will be optimized during backprop

Batch normalization

- 1. Normalize

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

- 2. Allow the network to change the range

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

backprop

The network *can* learn to undo the normalization

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

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

Batch normalization

- Is it ok to treat dimensions separately? Shown empirically that even if features are not decorrelated, convergence is still faster with this method

- You can set all biases of the layers before BN to zero, because they will be cancelled out by BN anyway

BN: train vs test time

- Train time: mean and variance is taken over the mini-batch

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

- Test-time: what happens if we can just process one image at a time?
  - No chance to compute a meaningful mean and variance

BN: train vs test time

Training	Testing
• Compute mean and variance from mini-batch 1 • Compute mean and variance from mini-batch 2 • Compute mean and variance from mini-batch 3	• Compute mean and variance by running an exponentially weighted averaged across training mini-batches

μ_{test} σ^2_{test}

BN: what do you get?

- Very deep nets are much easier to train → more stable gradients
- A much larger range of hyperparameters works similarly when using BN

BN: a milestone

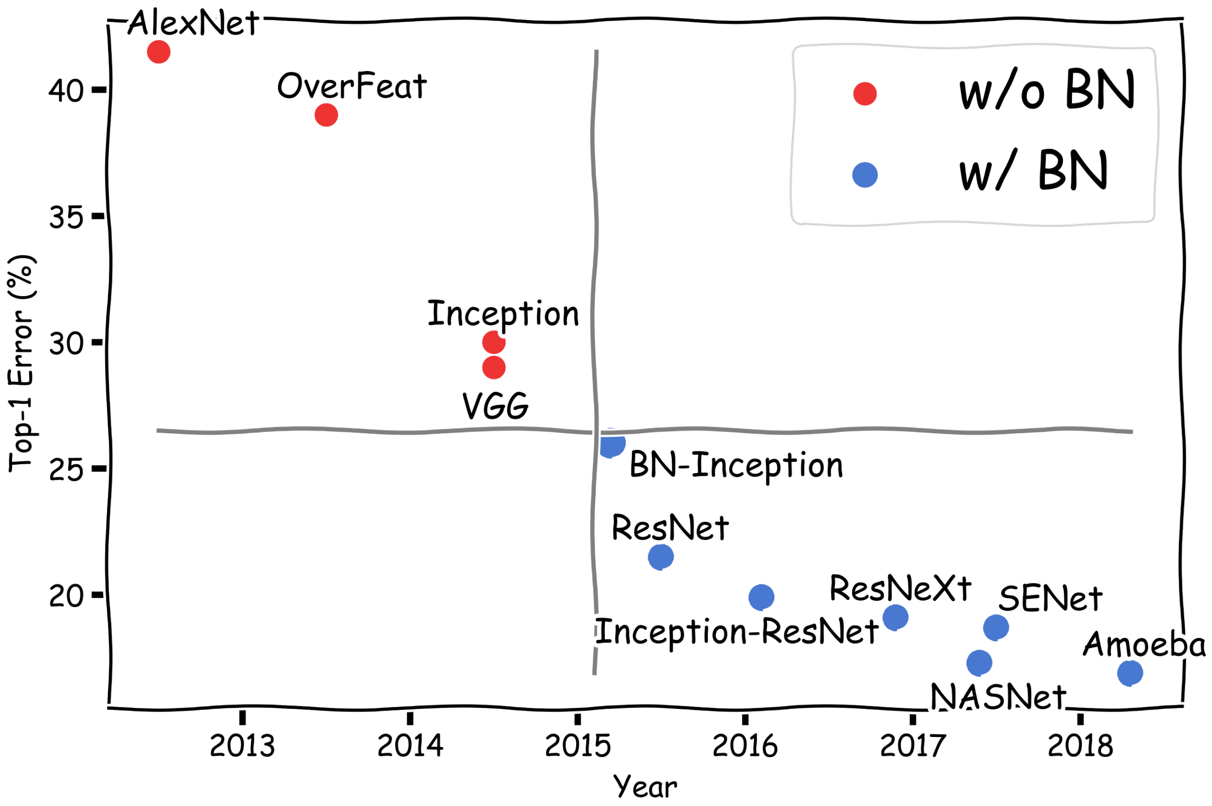


Image from Yuin Wu, Kaiming He

BN: drawbacks

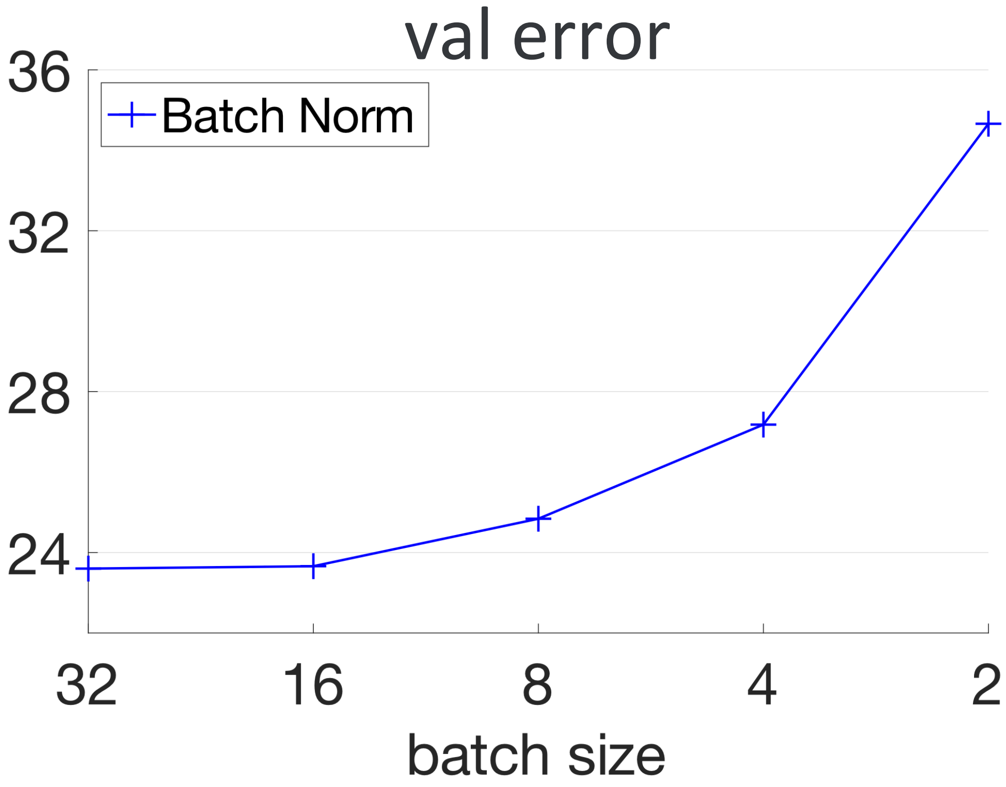


Image from Yuin Wu, Kaiming He

Other normalizations

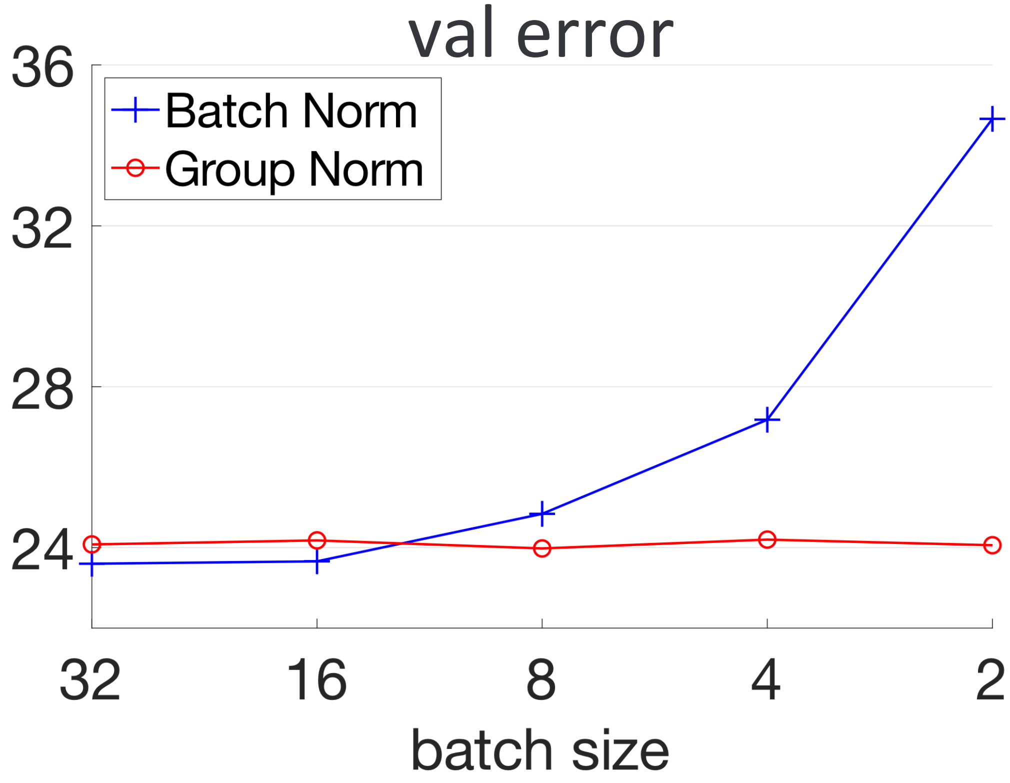
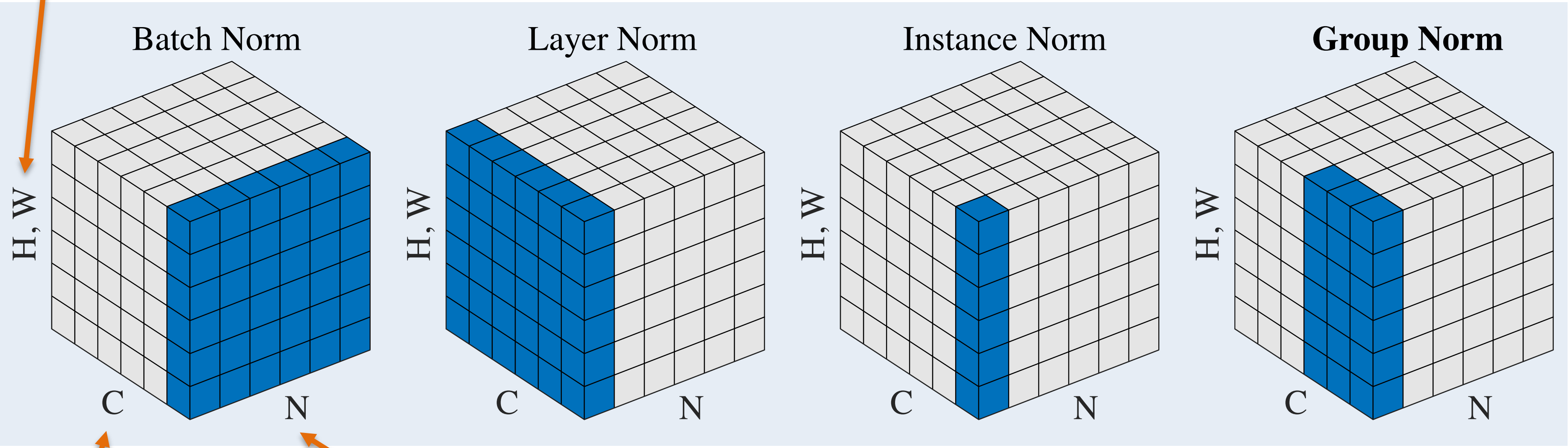


Image from Yuin Wu, Kaiming He

Other normalizations

Image size



Number of elements in the batch

Number of channels

Image from Yuin Wu, Kaiming He

Regularization

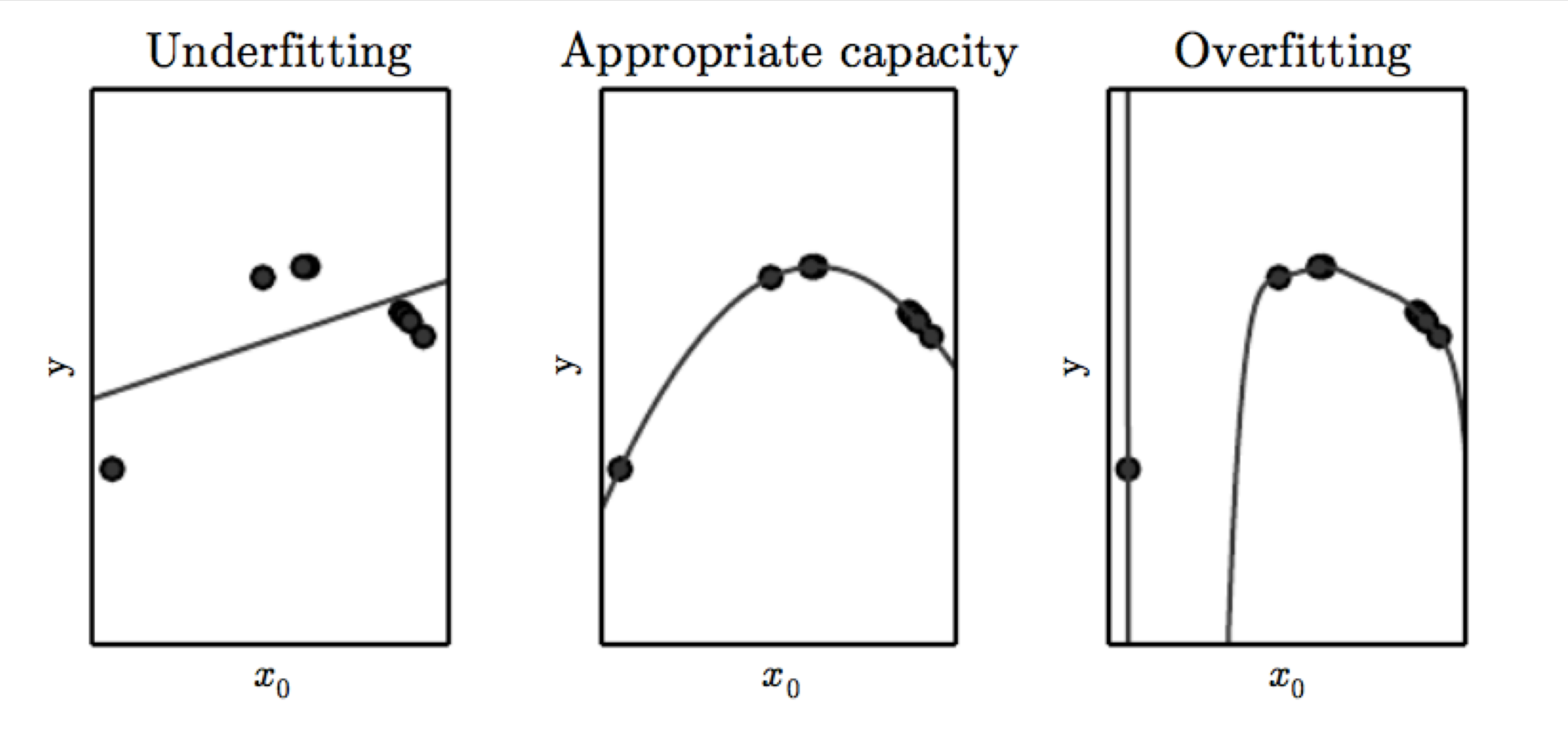
Regularization

- Any strategy that aims to

Lower validation error

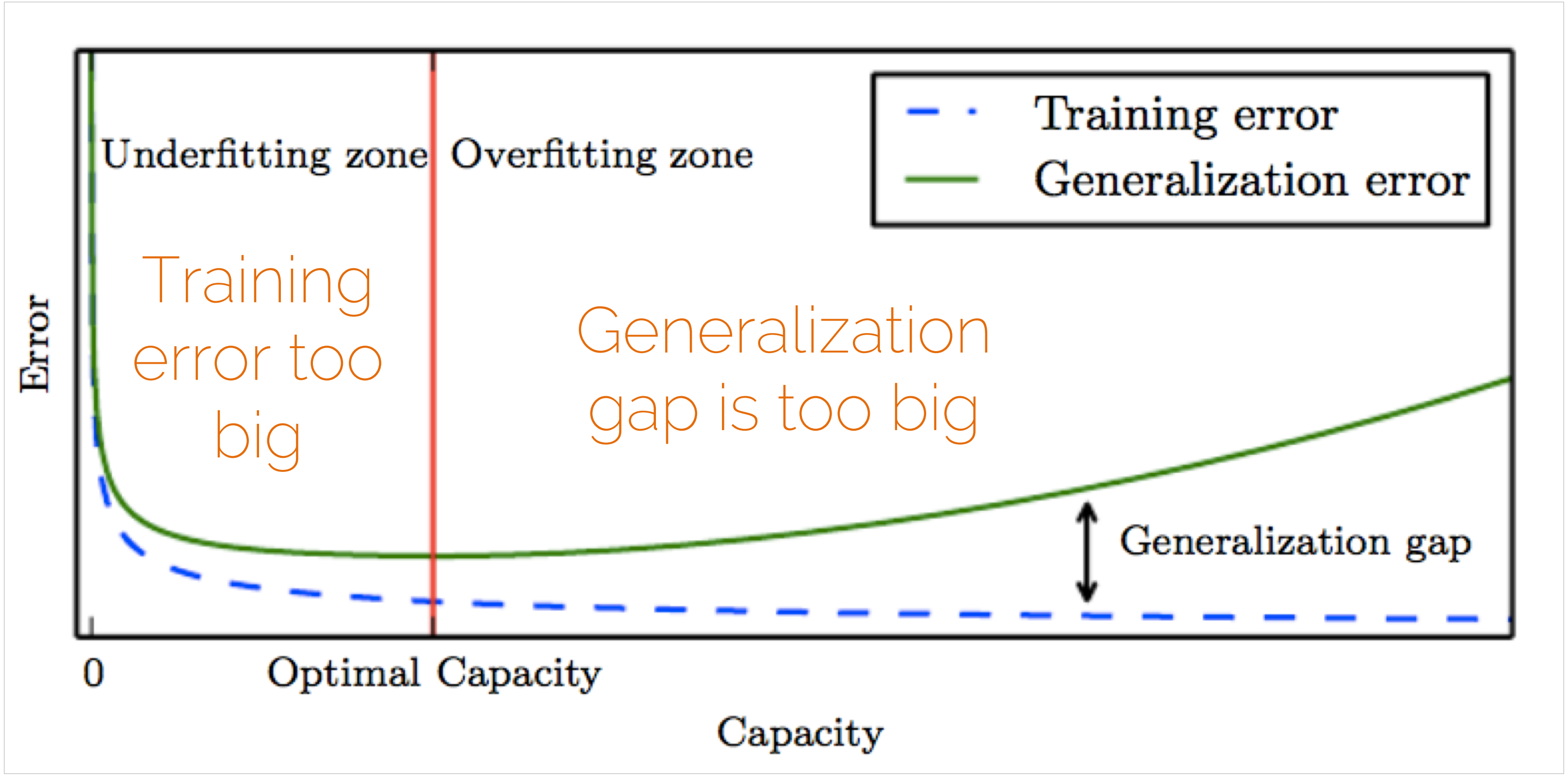
Increasing training error

Overfitting and underfitting



Credits: Deep Learning. Goodfellow et al.

Overfitting and underfitting



Credits: Deep Learning. Goodfellow et al.

Weight decay

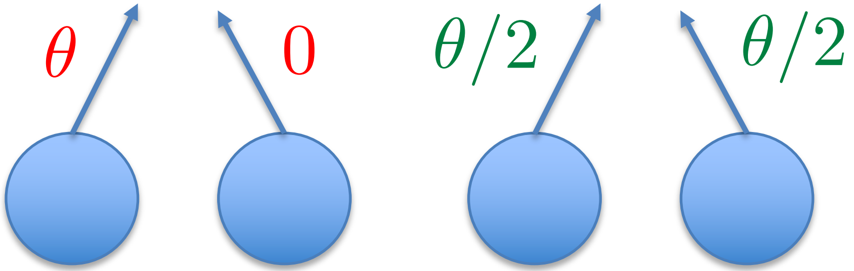
- L^2 regularization

$$\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k - \epsilon \nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}_k, \mathbf{x}^i, \mathbf{y}^i) - \lambda \boldsymbol{\theta}_k^T \boldsymbol{\theta}_k$$

Learning rate

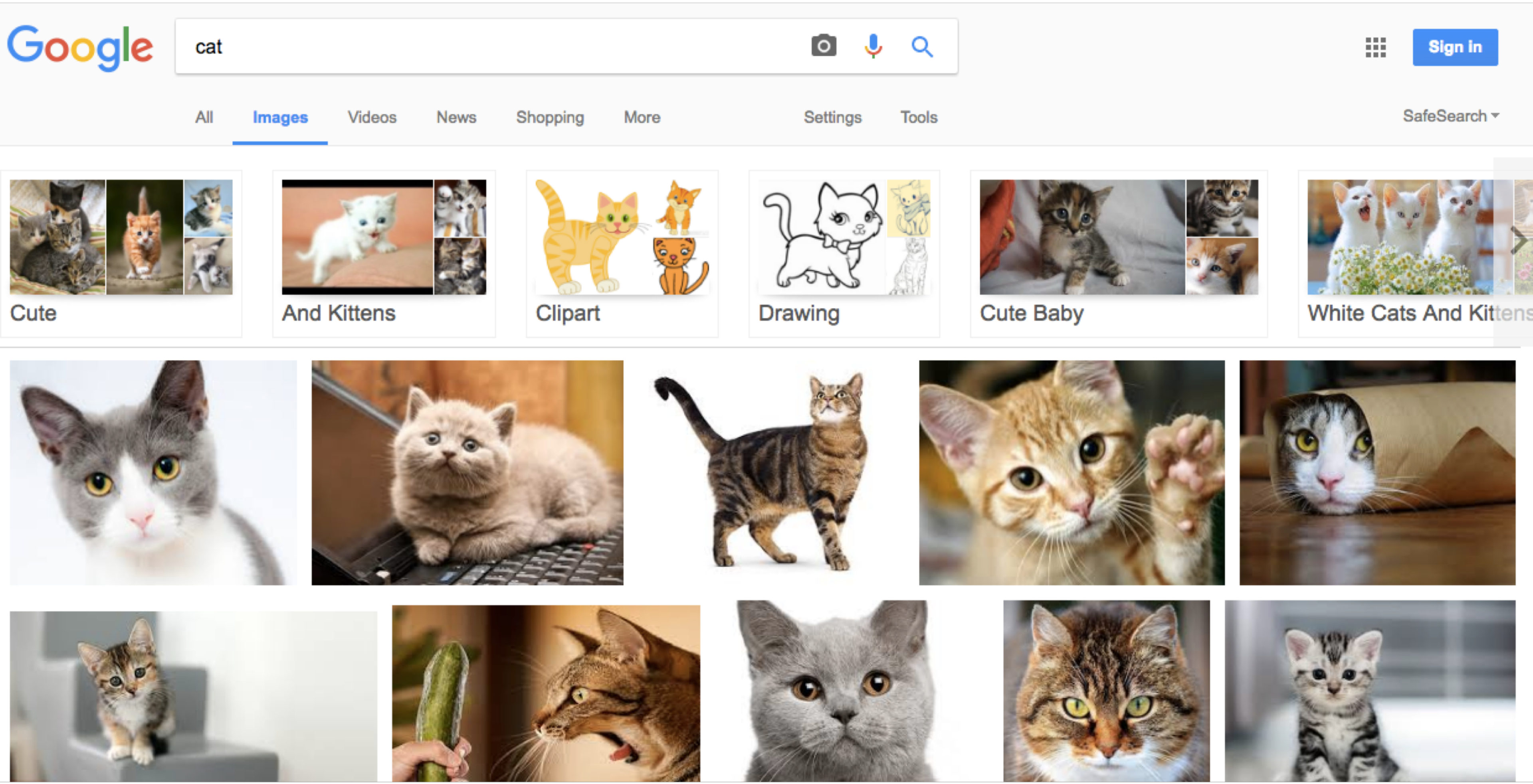
Gradient

- Penalizes large weights
- Improves generalization



Data augmentation

- A classifier has to be invariant to a wide variety of transformations



Pose Appearance Illumination

Data augmentation

- A classifier has to be invariant to a wide variety of transformations

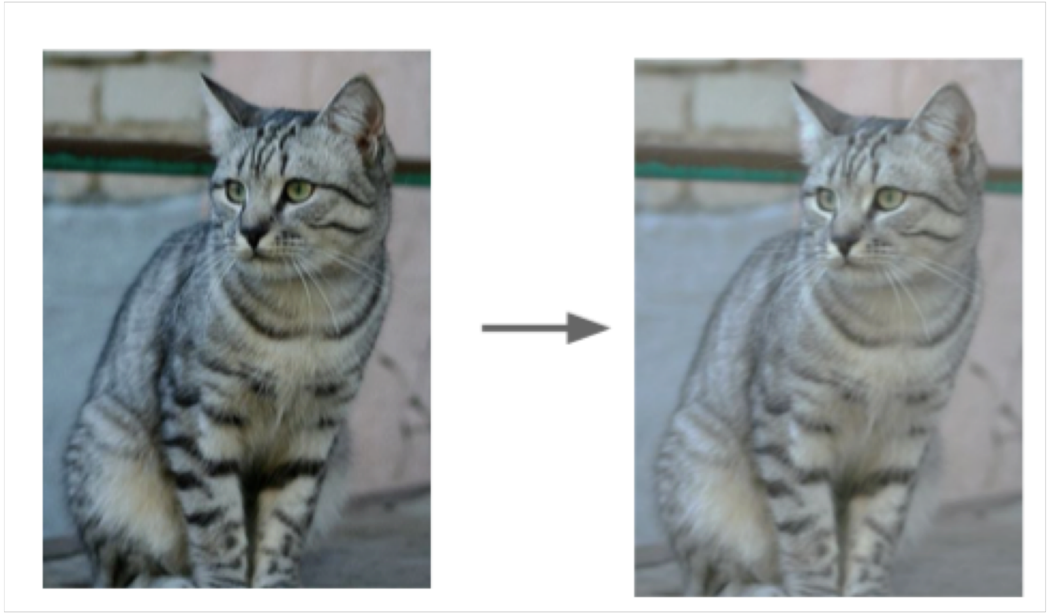
- Helping the classifier: generate fake data simulating plausible transformations

Data augmentation

Krizhevsky 2012

Data augmentation: random crops

- Random brightness and contrast changes

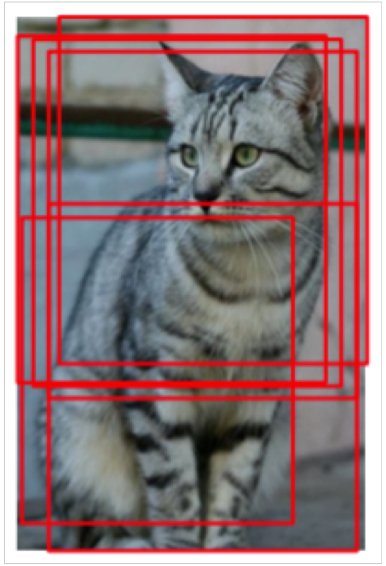


Krizhevsky 2012

Data augmentation: random crops

- Training: random crops
  - Pick a random L in [256,480]
  - Resize training image, short side L
  - Randomly sample crops of 224x224

- Testing: fixed set of crops
  - Resize image at N scales
  - 10 fixed crops of 224x224: 4 corners + center + flips

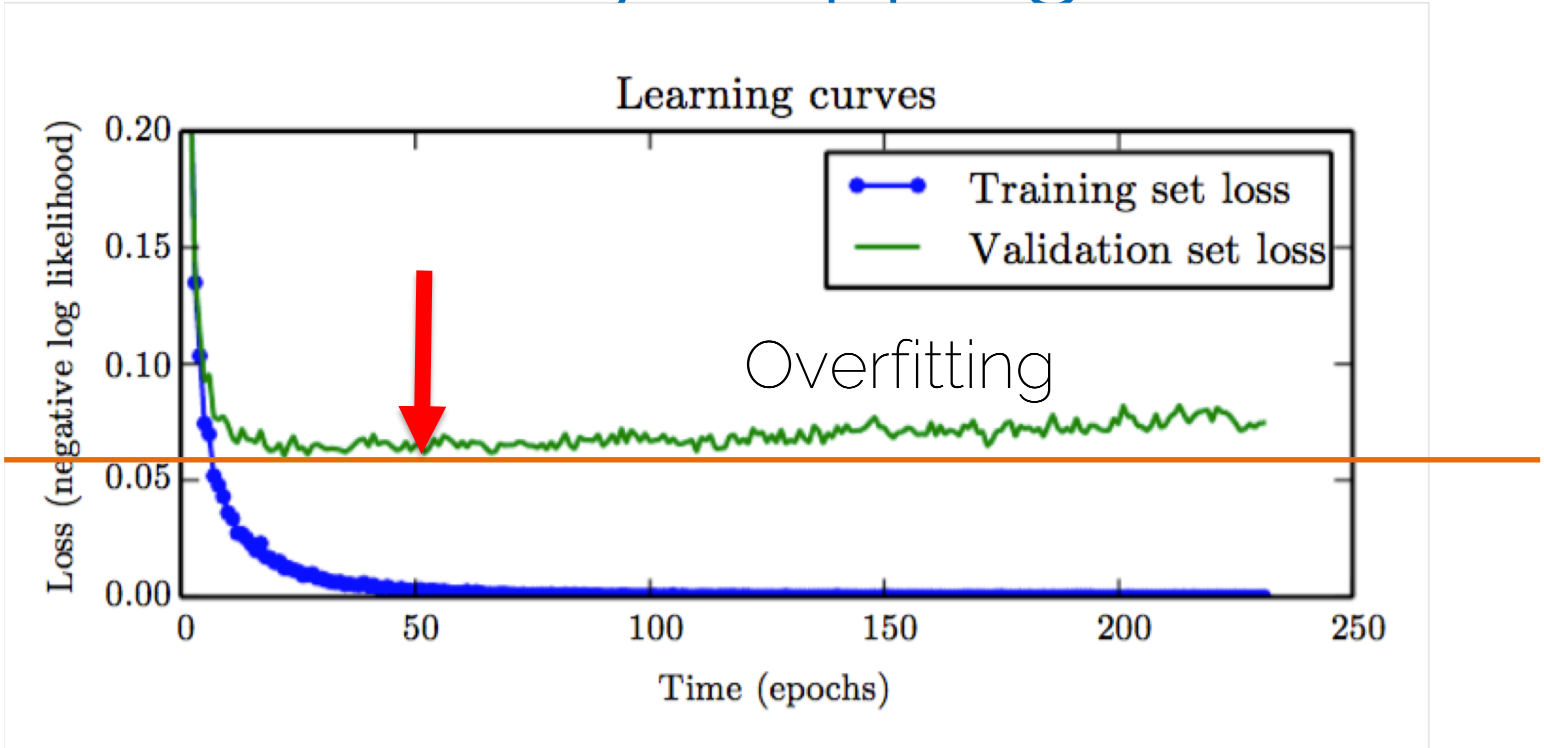


Krizhevsky 2012

Data augmentation

- When comparing two networks make sure to use the same data augmentation!
- Consider data augmentation a part of your network design

Early stopping



Training time is also a hyperparameter

Early stopping

- Easy form of regularization

$$\underbrace{\boldsymbol{\theta}_0 \xrightarrow{\epsilon} \boldsymbol{\theta}_1 \xrightarrow{\epsilon} \boldsymbol{\theta}_2 \qquad \boldsymbol{\theta}_s}_{\tau} \qquad\qquad \boldsymbol{\theta}^*$$

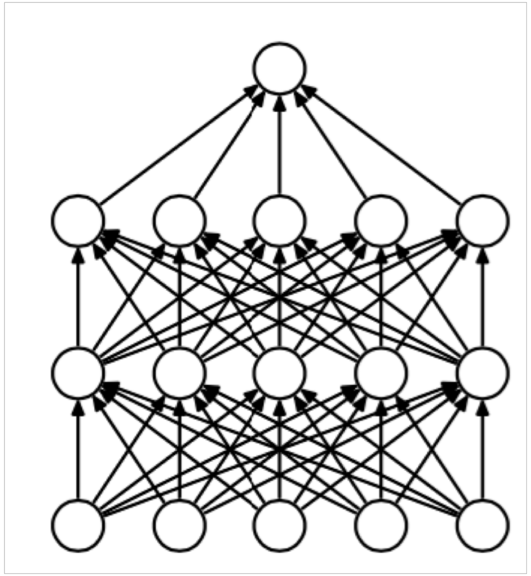
Overfitting

Bagging and ensemble methods

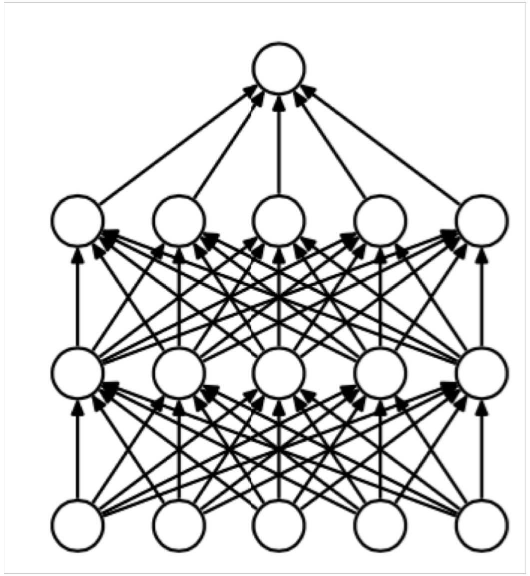
- Train three models and average their results
- Change a different algorithm for optimization or change the objective function
- If errors are uncorrelated, the expected combined error will decrease linearly with the ensemble size

Bagging and ensemble methods

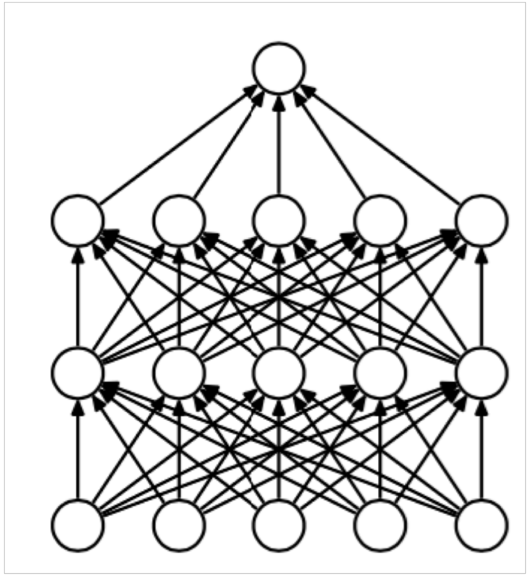
- Bagging: uses k different datasets



Training Set 1



Training Set 2

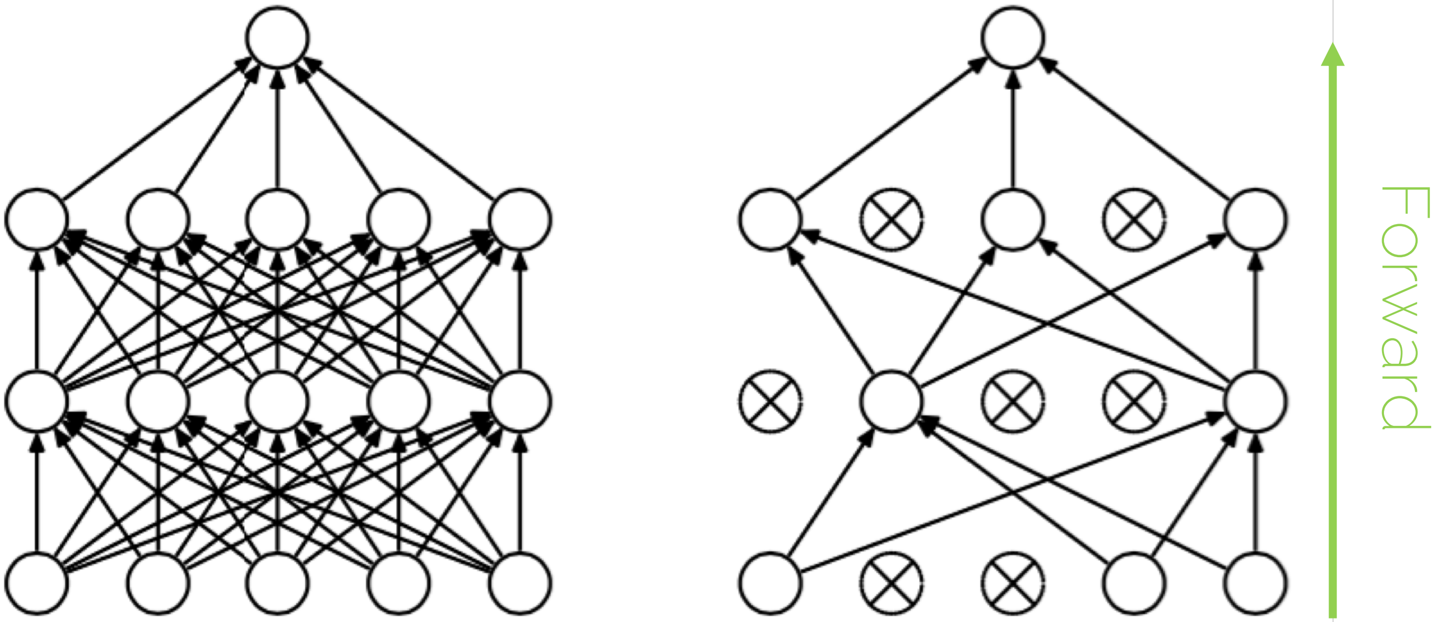


Training Set 3

Dropout

Dropout

- Disable a random set of neurons (typically 50%)



(a) Standard Neural Net

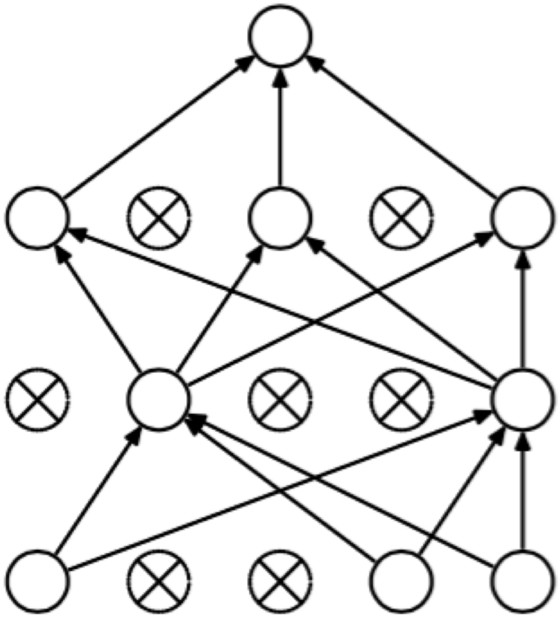
(b) After applying dropout.

Srivastava 2014

Dropout: intuition

- Using half the network = half capacity

Redundant representations



(b) After applying dropout.

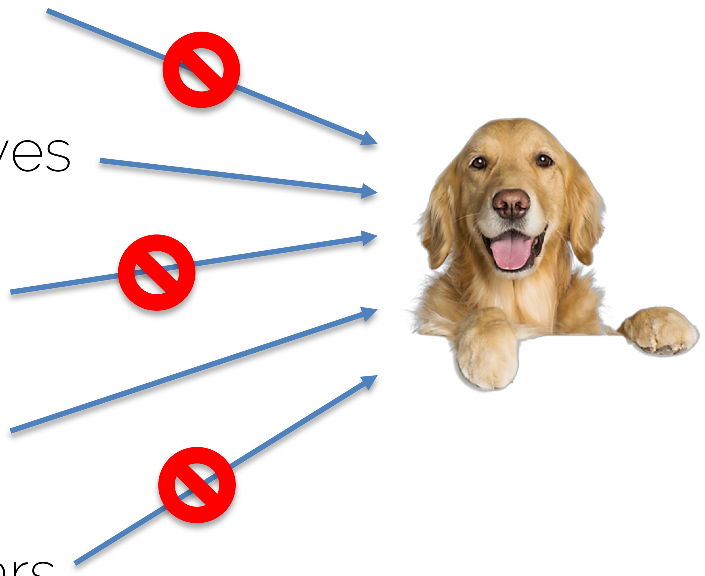
Furry

Has two eyes

Has a tail

Has paws

Has two ears



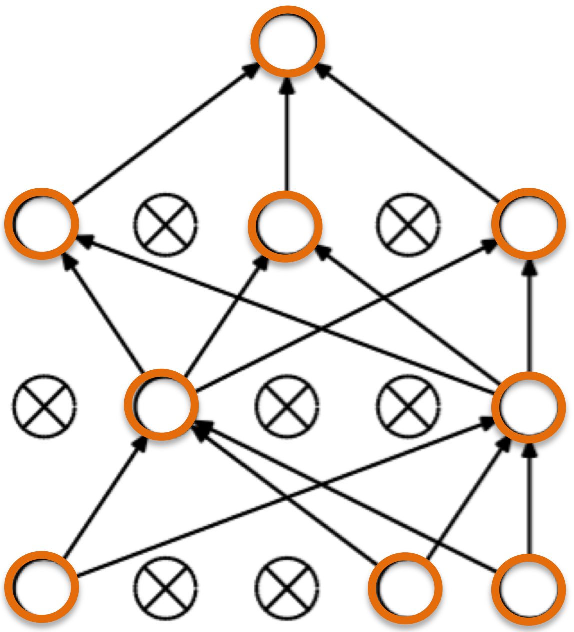
Dropout: intuition

- Using half the network = half capacity
  - Redundant representations
  - Base your scores on more features

- Consider it as model ensemble

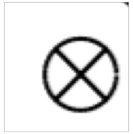
Dropout: intuition

- Two models in one



(b) After applying dropout.

Model 1



Model 2

Dropout: intuition

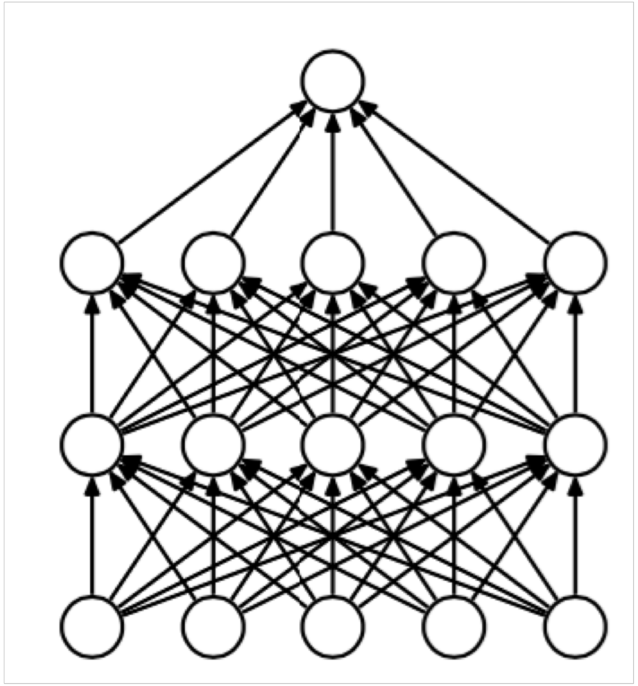
- Using half the network = half capacity
  - Redundant representations
  - Base your scores on more features

- Consider it as two models in one
  - Training a large ensemble of models, each on different set of data (mini-batch) and with SHARED parameters

Reducing co-adaptation between neurons

Dropout: test time

- All neurons are "turned on" – no dropout

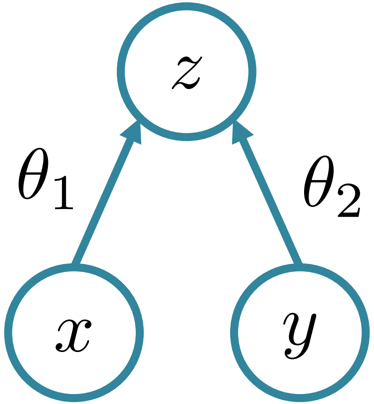


Conditions at train and test time are not the same

Dropout: test time

Dropout probability p=0.5

- Test: $z = \theta_1 x + \theta_2 y$



- Train: $$\mathrm{E}[z] = \frac{1}{4}(\theta_1 0 + \theta_2 0 + \theta_1 x + \theta_2 0 + \theta_1 0 + \theta_2 y + \theta_1 x + \theta_2 y) = \frac{1}{2}(\theta_1 x + \theta_2 y)$$

Weight scaling inference rule

Dropout: verdict

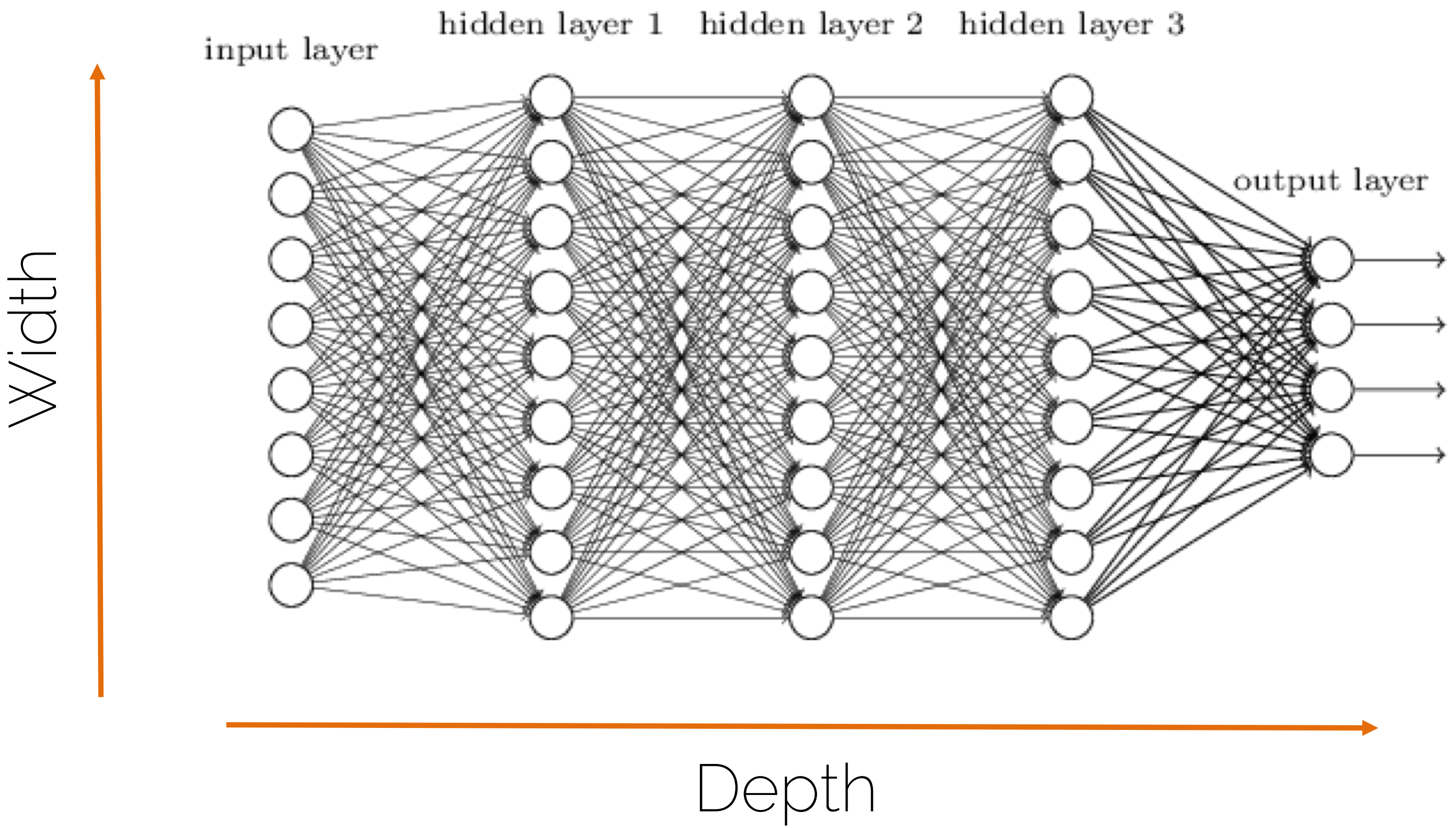
- Efficient bagging method with parameter sharing

- Use it!

- Dropout reduces the effective capacity of a model → larger models, more training time

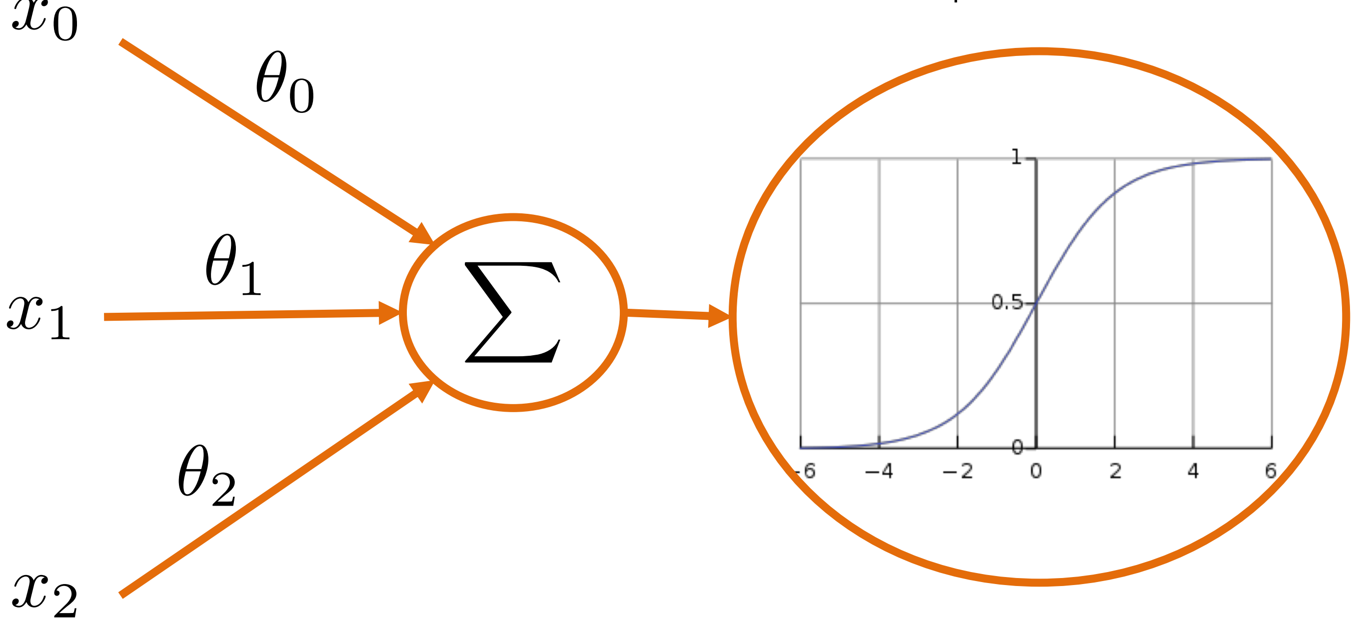
Recap

What do we know so far?



What do we know so far?

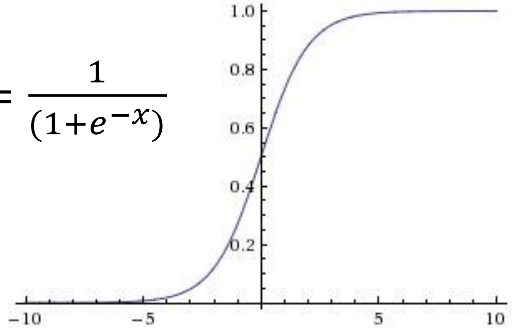
Concept of a 'Neuron'



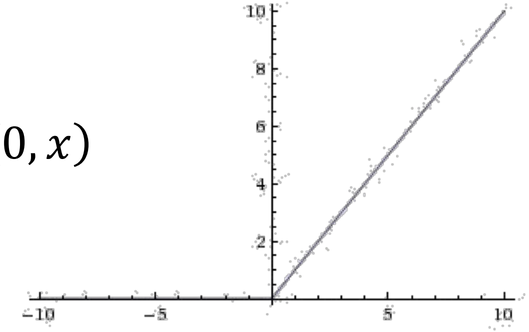
What do we know so far?

Activation Functions (non-linearities)

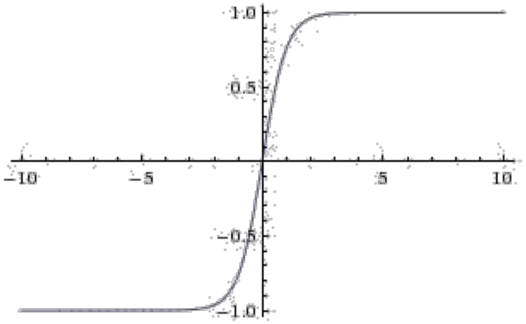
Sigmoid: $\sigma(x) = \frac{1}{(1+e^{-x})}$



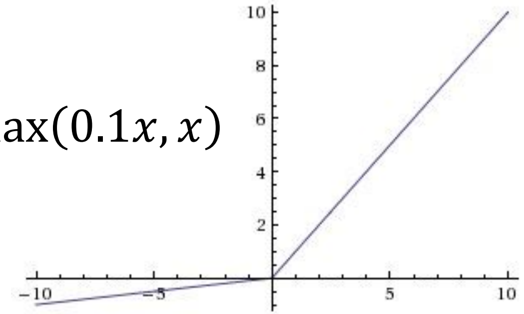
ReLU: $\mathbf{max}(0, x)$



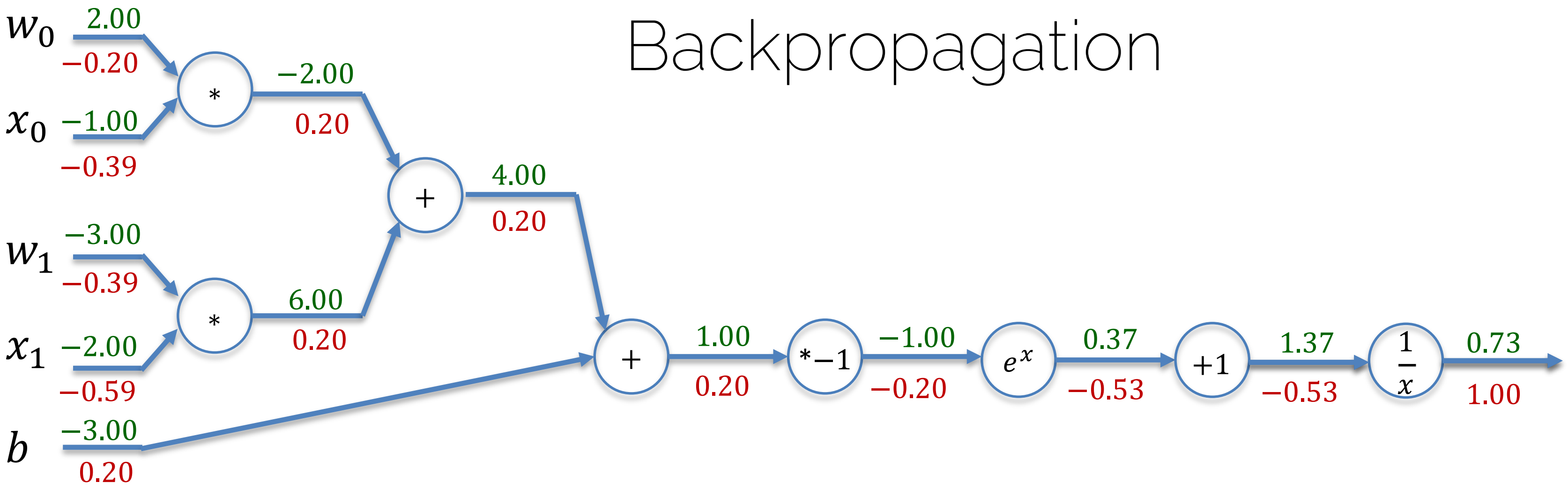
tanh: $\mathbf{tanh}(x)$



Leaky ReLU: $\mathbf{max}(0.1x, x)$

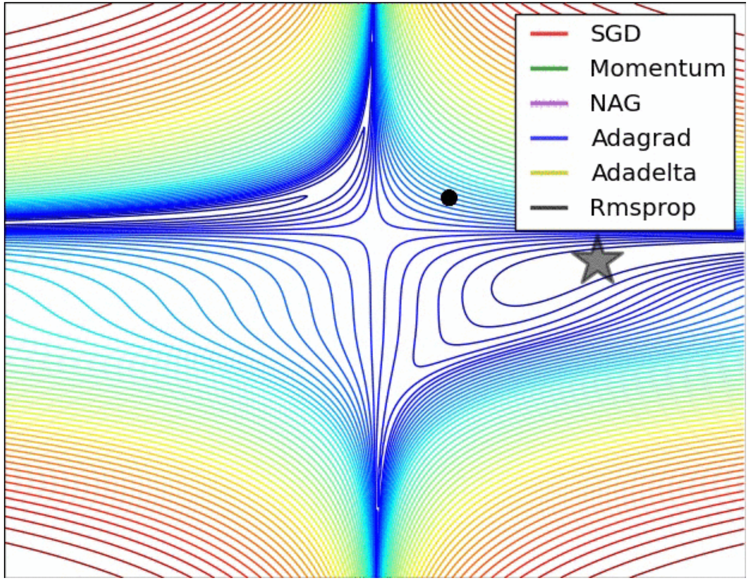


What do we know so far?



What do we know so far?

SGD Variations (Momentum, etc.)



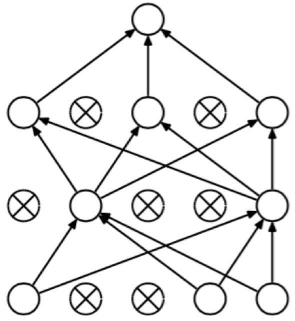
What do we know so far?

Data Augmentation

Batch-Norm

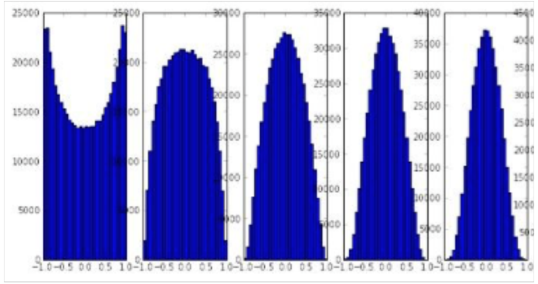
$$\hat{x}^{(k)} = \frac{x^{(k)} - \mathrm{E}[x^{(k)}]}{\sqrt{\mathrm{Var}[x^{(k)}]}}$$

Dropout



(b) After applying dropout.

Weight Initialization (e.g., Xavier/2)



Weight Regularization

e.g., L^2-reg: $R^2(W) = \sum_{i=1}^{N} w_i^2$

Why not only more Layers?

- We can not make networks arbitrarily complex
  - Why not just go deeper and get better?
  - No structure!!
  - It's just brute force!
  - Optimization becomes hard
  - Performance plateaus / drops!

Administrative Things

- Happy holidays!
- Tuesday December 18th: solution to Exercise 2, introduction to exercise 3 and introduction to PyTorch!
- Thursday January 10th: Starting with CNN