

# Lecture 7 Recap

# Regression Losses: L2 vs L1

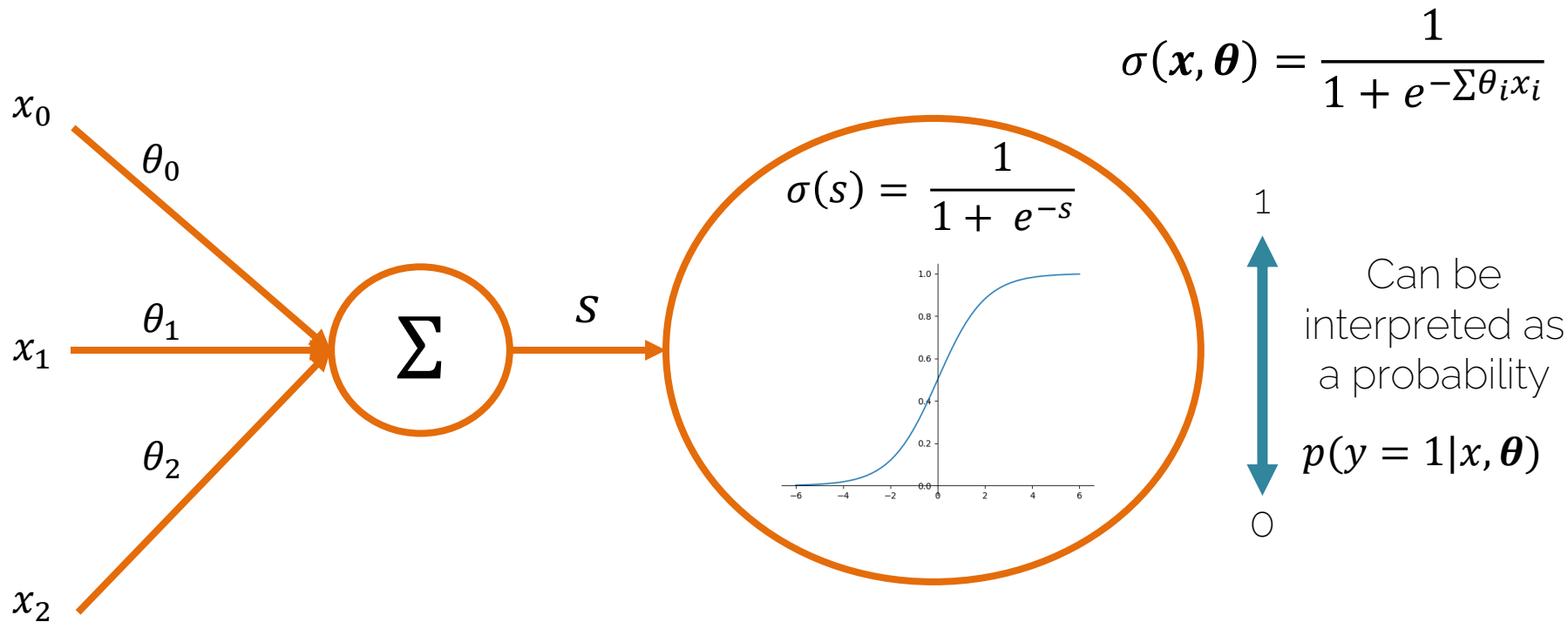
- L2 Loss:

- $L^2 = \sum_{i=1}^n (y_i - f(x_i))^2$
- Sum of squared differences (SSD)
- Prone to outliers
- Compute-efficient (optimization)
- Optimum is the mean

- L1 Loss:

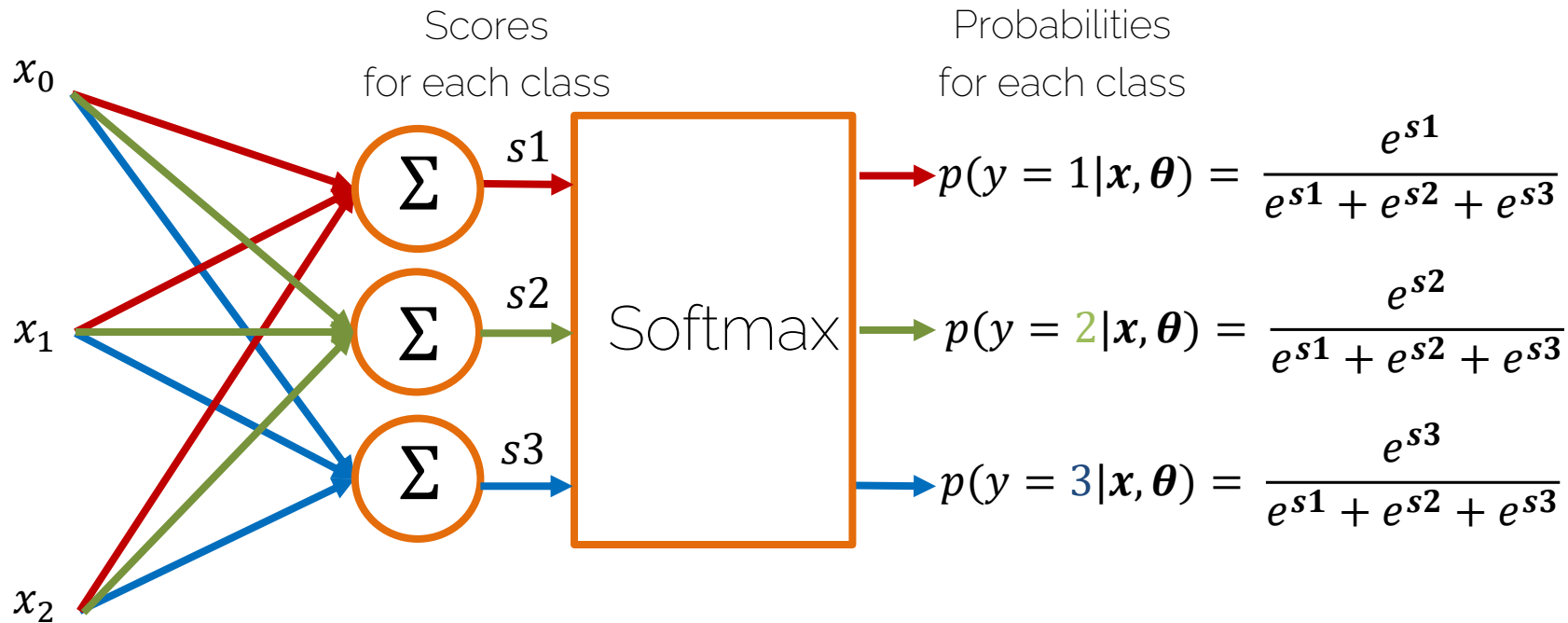
- $L^1 = \sum_{i=1}^n |y_i - f(x_i)|$
- Sum of absolute differences
- Robust
- Costly to compute
- Optimum is the median

# Binary Classification: Sigmoid



# Softmax Formulation

- What if we have multiple classes?



# Example: Hinge vs Cross-Entropy

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

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

Given the following scores for  $\mathbf{x}_i$ :

Hinge loss:

Cross Entropy loss:

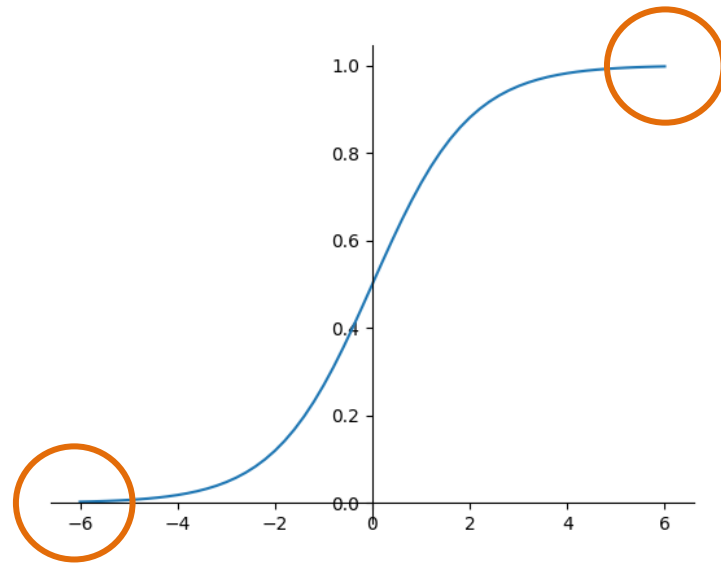
|         |  |   |   |
|---------|--|---|---|
| Model 1 | $s = [5, -3, 2]$                                       | $\max(0, -3 - 5 + 1) + \max(0, 2 - 5 + 1) = 0$    | $-\ln\left(\frac{e^5}{e^5 + e^{-3} + e^2}\right) = 0.05$              |
| Model 2 | $s = [5, 10, 10]$                                      | $\max(0, 10 - 5 + 1) + \max(0, 10 - 5 + 1) = 12$  | $-\ln\left(\frac{e^5}{e^5 + e^{10} + e^{10}}\right) = 5.70$           |
| Model 3 | $s = [5, -20, -20]$<br><div><math>y_i = 0</math></div> | $\max(0, -20 - 5 + 1) + \max(0, -20 - 5 + 1) = 0$ | $-\ln\left(\frac{e^5}{e^5 + e^{-20} + e^{-20}}\right) = 2 * 10^{-11}$ |

- Cross Entropy \*always\* wants to improve! (loss never 0)
- Hinge Loss saturates.

# Sigmoid Activation

Forward

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



**X** Saturated neurons kill the gradient flow

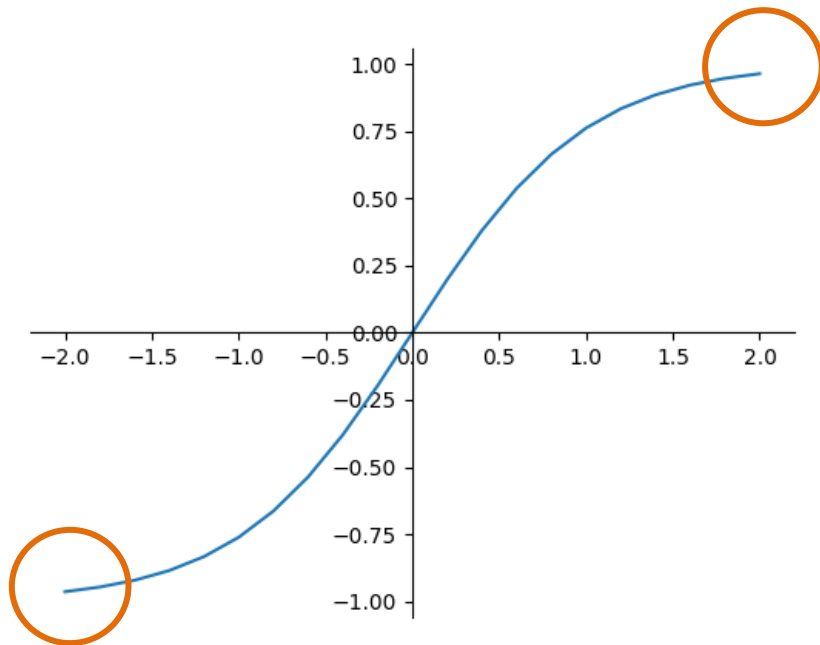
$$\cancel{\frac{\partial L}{\partial w}} = \frac{\partial s}{\partial w} \frac{\partial L}{\partial s}$$

$$\cancel{\frac{\partial L}{\partial s}} = \frac{\partial \sigma}{\partial s} \frac{\partial L}{\partial \sigma}$$

$$\cancel{\frac{\partial \sigma}{\partial s}}$$

$$\frac{\partial L}{\partial \sigma}$$

# TanH Activation



Still saturates



Zero-centered

[LeCun et al. 1991] Improving Generalization Performance in Character Recognition

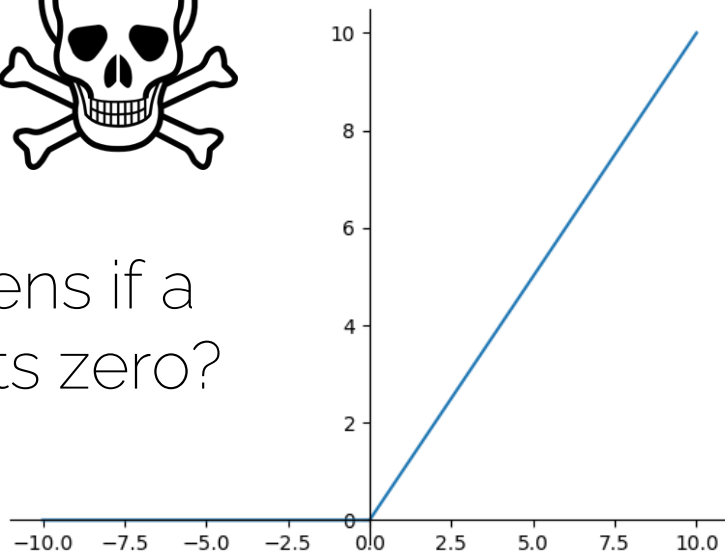
# Rectified Linear Units (ReLU)



Dead ReLU



What happens if a ReLU outputs zero?



Large and consistent gradients



Fast convergence



Does not saturate

[Krizhevsky et al. NeurIPS 2012] ImageNet Classification with Deep Convolutional Neural Networks



# Quick Guide

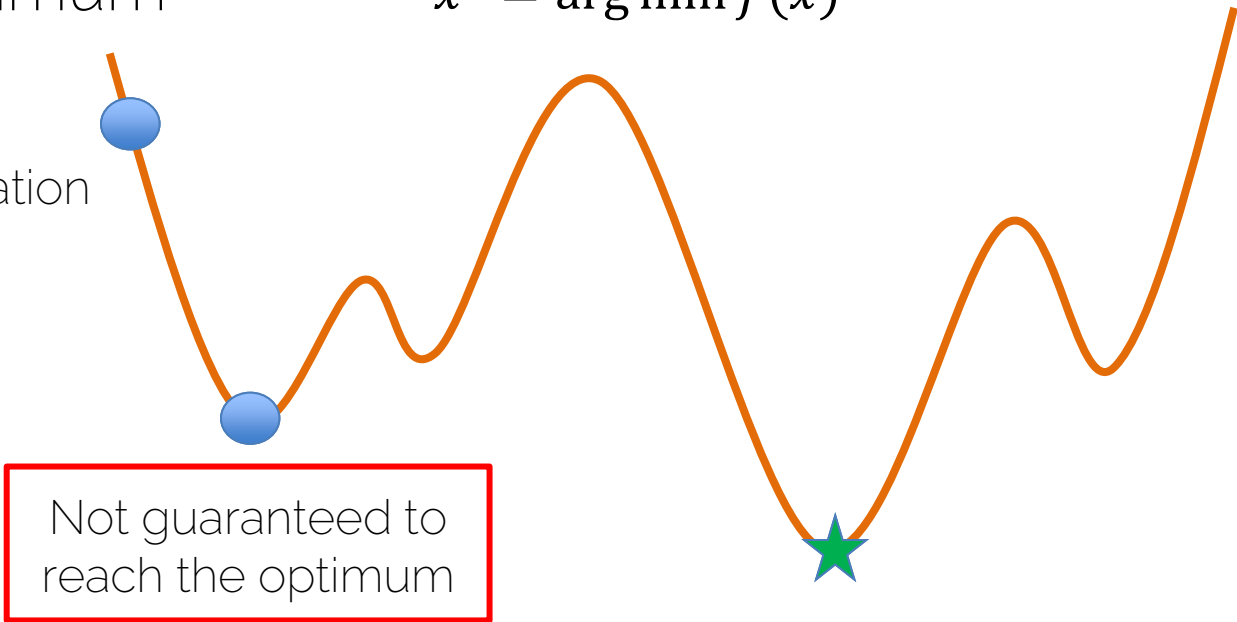
- Sigmoid is not really used.
- ReLU is the standard choice.
- Second choice are the variants of ReLU or Maxout.
- Recurrent nets will require TanH or similar.

# Initialization is Extremely Important!

- Optimum

$$x^* = \arg \min f(x)$$

Initialization



# Xavier/Kaiming Initialization

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

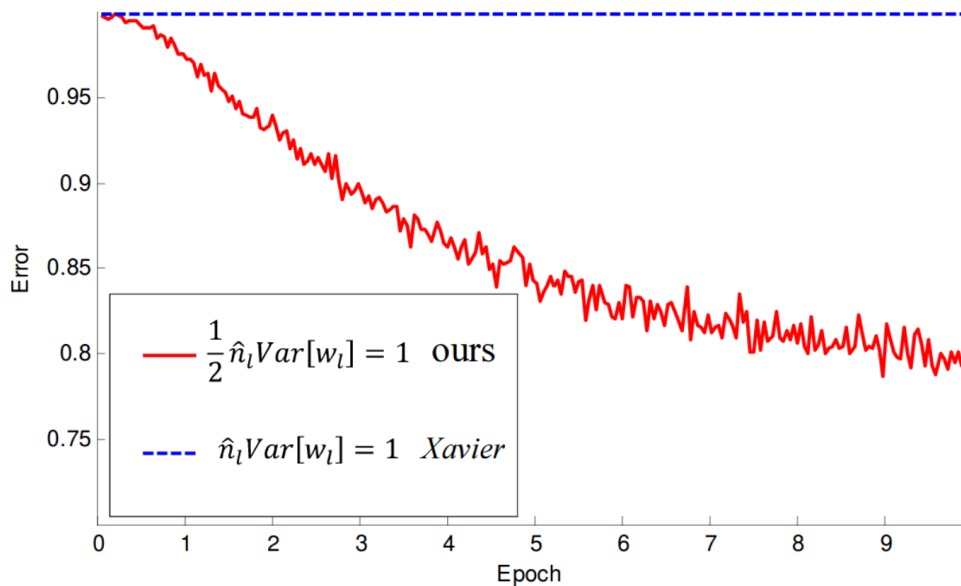
$$\underbrace{(n\text{Var}(w)\text{Var}(x))}_{= 1}$$

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

# ReLU Kills Half of the Data

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

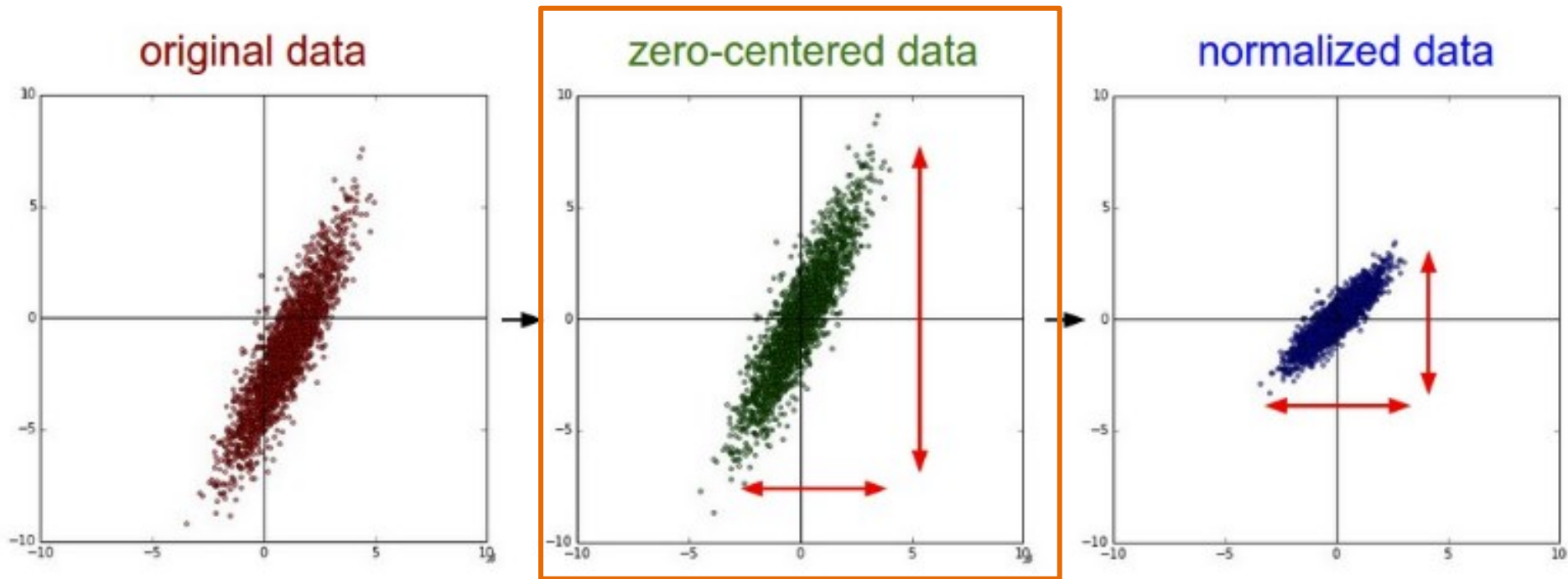
It makes a huge difference!



# Lecture 8

# Data Augmentation

# Data Pre-Processing



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

# Data Augmentation

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



All Images Videos News Shopping More Settings Tools

SafeSearch ▼



Cute



And Kittens



Clipart



Drawing



Cute Baby



White Cats And Kittens



Pose

Appearance

Illumination

# Data Augmentation

- A classifier has to be invariant to a wide variety of transformations
- Helping the classifier: synthesize data simulating plausible transformations

# Data Augmentation

a. No augmentation (= 1 image)



b. Flip augmentation (= 2 images)

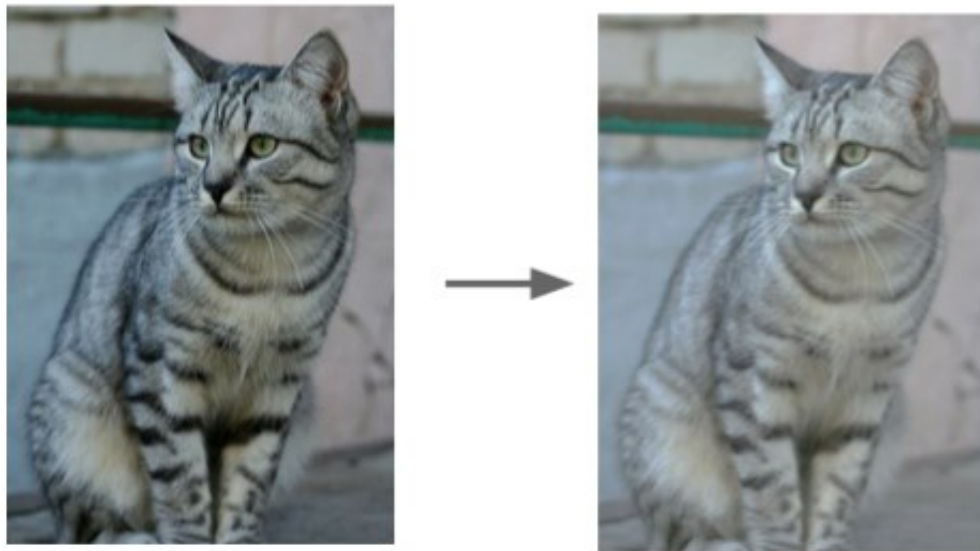


c. Crop+Flip augmentation (= 10 images)



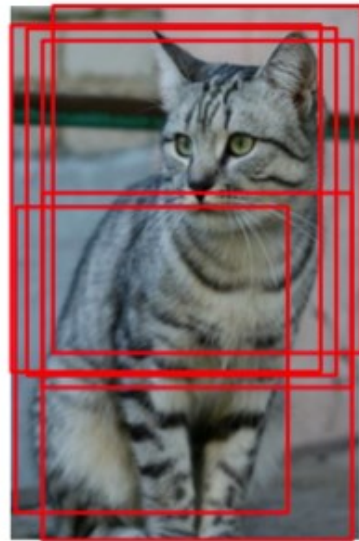
# Data Augmentation: Brightness

- Random brightness and contrast changes

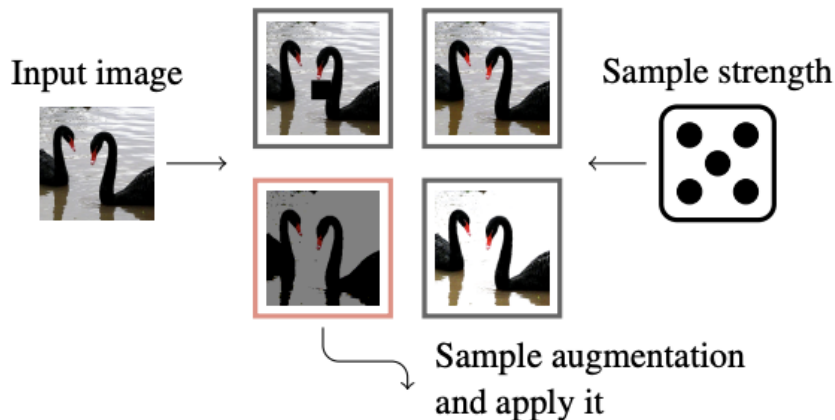
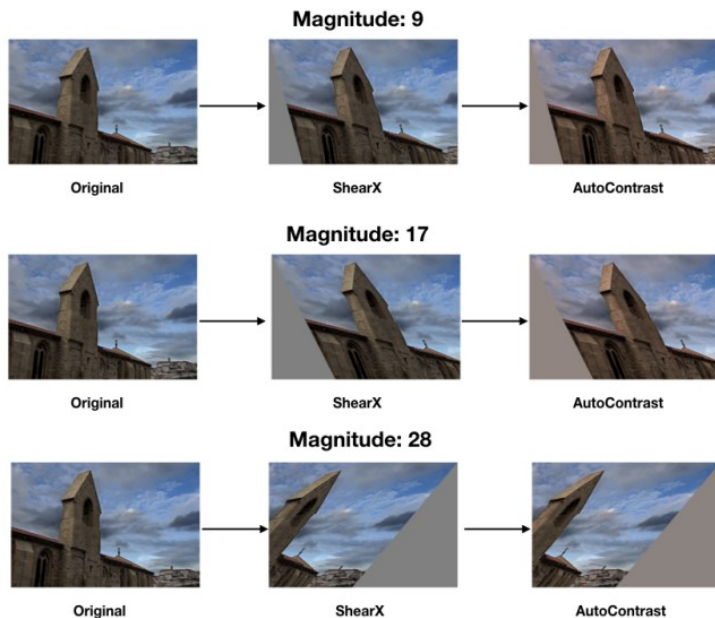


# Data Augmentation: Random Crops

- Training: random crops
  - Pick a random  $L$  in  $[256, 480]$
  - Resize training image, short side  $L$
  - Randomly sample crops of  $224 \times 224$
- Testing: fixed set of crops
  - Resize image at  $N$  scales
  - 10 fixed crops of  $224 \times 224$ : 4 corners + 1 center )  $\times 2$  flips



# Data Augmentation: Advanced



Cubuk et al., RandAugment, CVPRW 2020

Muller et al., Trivial Augment, ICCV 2021

# Data Augmentation

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

# Advanced Regularization



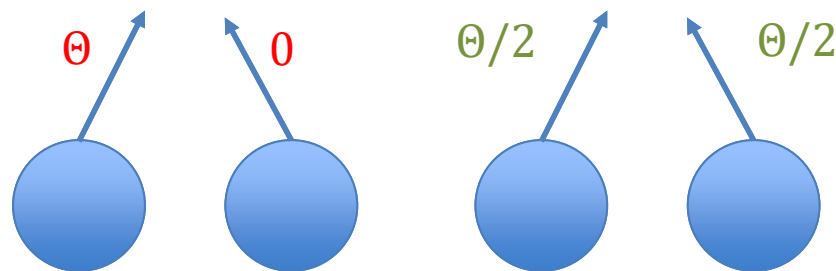
# L2 regularization, also (wrongly) called weight decay

- L2 regularization

$$\Theta_{k+1} = \Theta_k - \epsilon \nabla_{\Theta}(\Theta_k, x, y) - \lambda \theta_k$$

Learning rate      Gradient      Gradient of L2-regularization

- Penalizes large weights
- Improves generalization



# L2 regularization, also (wrongly) called weight decay

- Weight decay regularization

$$\Theta_{k+1} = (1 - \lambda)\Theta_k - \alpha \nabla f_k(\Theta_k)$$

Learning rate of weight  
decay

A blue arrow points from the text 'Learning rate of weight decay' to the parameter  $\lambda$  in the equation above.

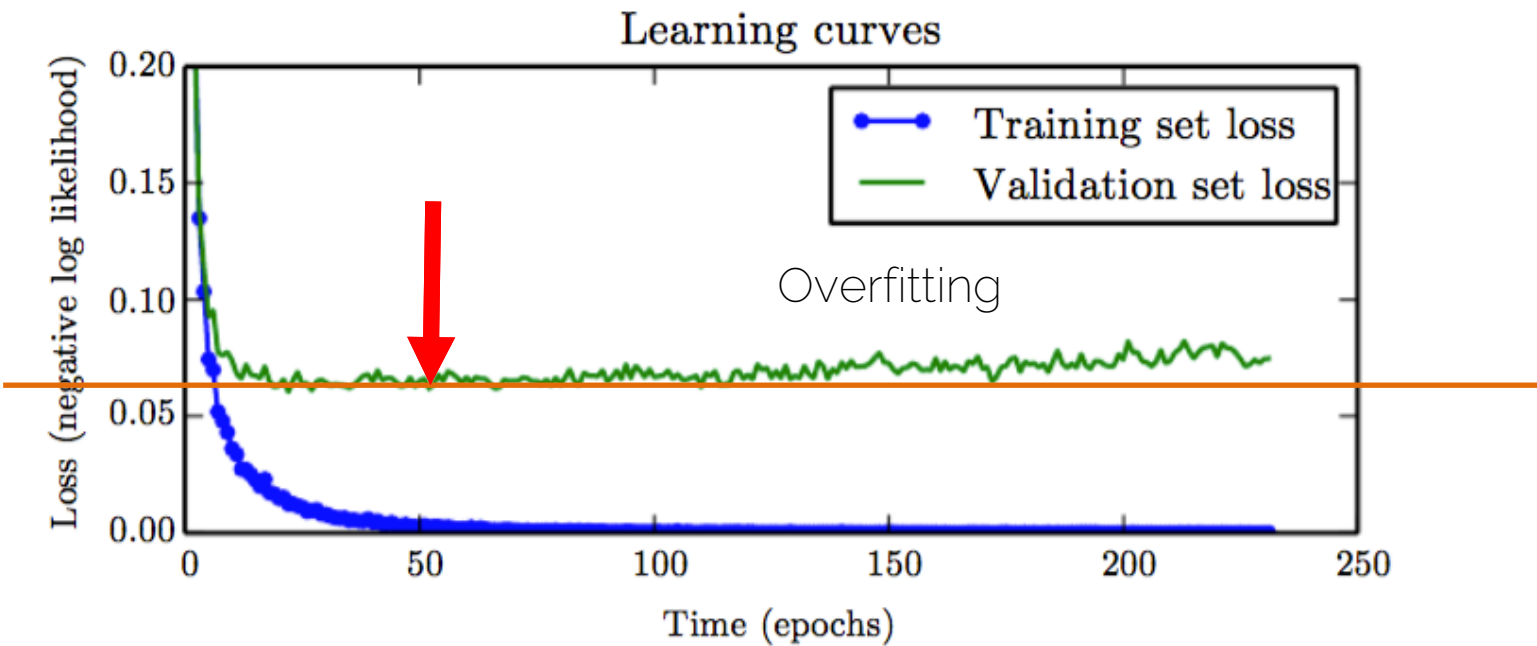
Learning rate of the  
optimizer

A blue arrow points from the text 'Learning rate of the optimizer' to the parameter  $\alpha$  in the equation above.

- Equivalent to L2 regularization in GD, but not in Adam.

Loshchilov and Hutter, Decoupled Weight Decay  
Regularization, ICLR 2019

# Early Stopping

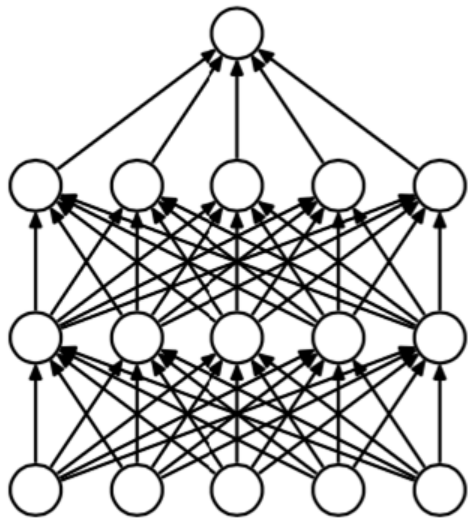


# Bagging and Ensemble Methods

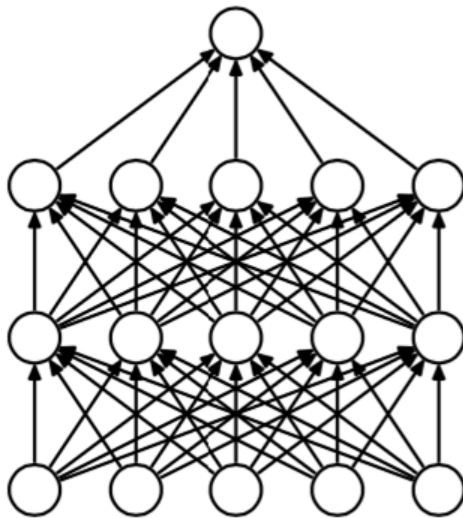
- Train multiple models and average their results
- E.g., use a different algorithm for optimization or change the objective function / loss function.
- If errors are uncorrelated, the expected combined error will decrease linearly with the ensemble size

# Bagging and Ensemble Methods

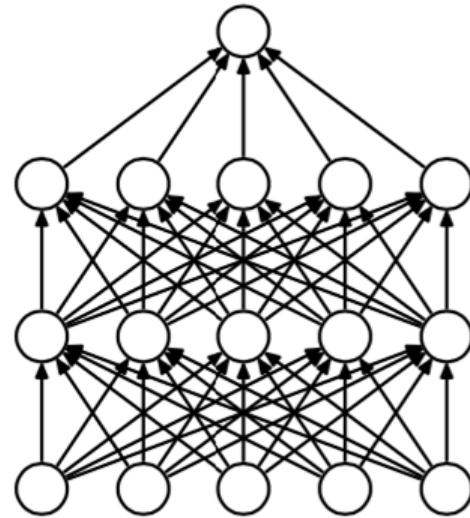
- Bagging: uses  $k$  different datasets (or SGD/init noise)



Training Set 1



Training Set 2

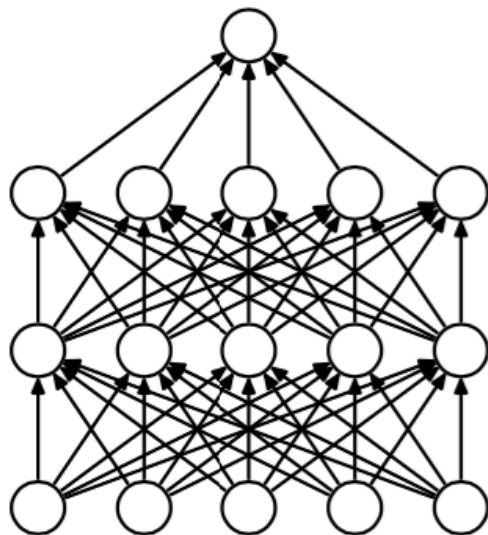


Training Set 3

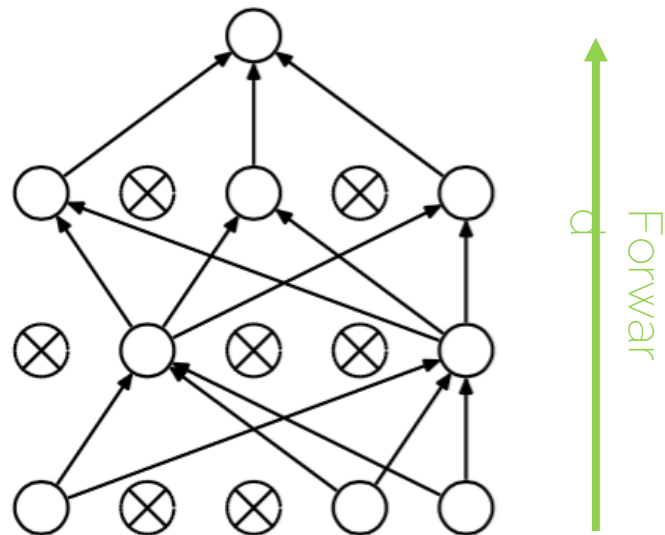
# Dropout

# Dropout

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



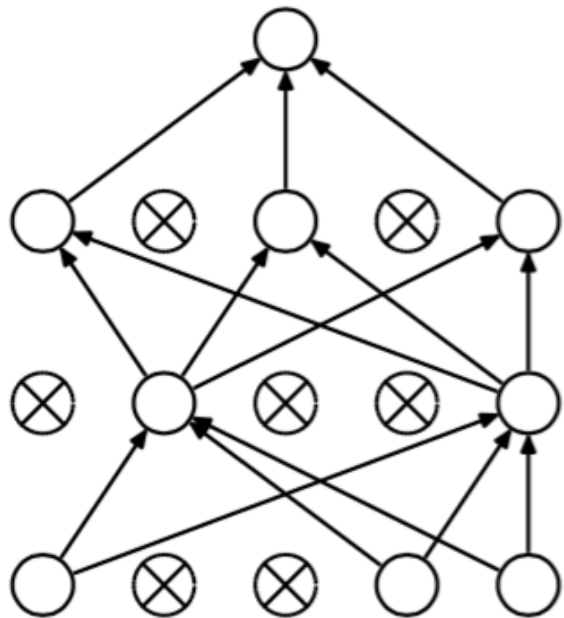
(a) Standard Neural Net



(b) After applying dropout.

# Dropout: Intuition

- Using half the network = half capacity



(b) After applying dropout.

Redundant  
representations

Furry



Has two  
eyes



Has a tail



Has paws

Has two ears



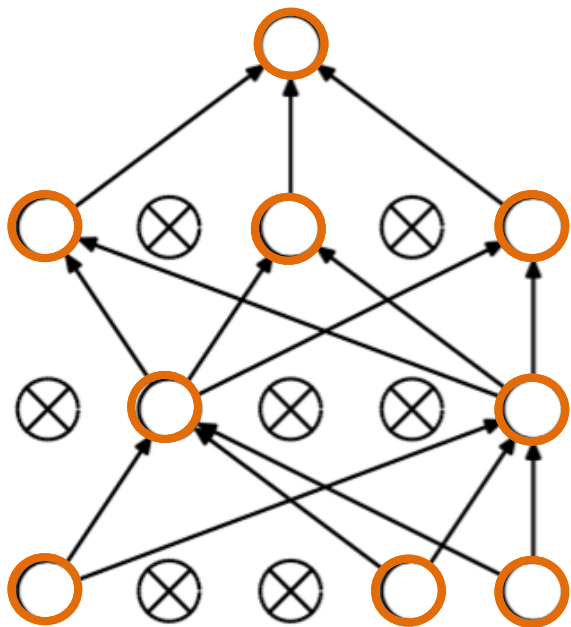


# Dropout: Intuition

- Using half the network = half capacity
  - Redundant representations
  - Base your scores on more features
- Consider it as a model ensemble

# Dropout: Intuition

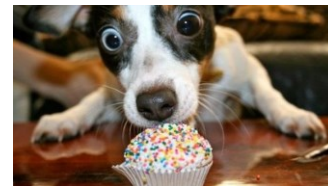
- Two models in one



(b) After applying dropout.

 Model 1

 Model 2



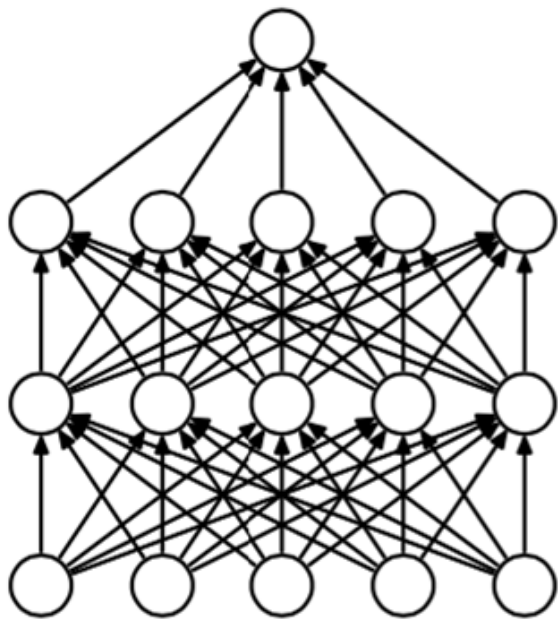
# Dropout: Intuition

- 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

PyTorch: `model.train()` and `model.eval()`



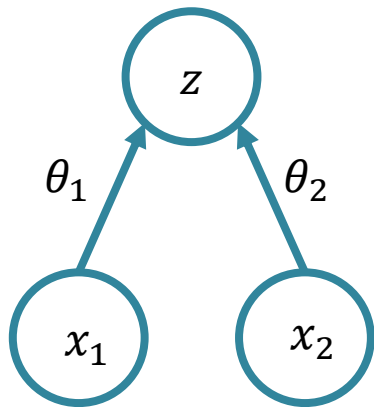
# Dropout: Test Time

- Test:

$$z = (\theta_1 x_1 + \theta_2 x_2) \cdot p$$

Dropout  
probability  
 $p = 0.5$

- Train:



Weight scaling  
inference rule

$$\begin{aligned} E[z] &= \frac{1}{4} (\theta_1 0 + \theta_2 0 \\ &\quad + \theta_1 x_1 + \theta_2 0 \\ &\quad + \theta_1 0 + \theta_2 x_2 \\ &\quad + \theta_1 x_1 + \theta_2 x_2) \\ &= \frac{1}{2} (\theta_1 x_1 + \theta_2 x_2) \end{aligned}$$

# Dropout: Before

- Efficient bagging method with parameter sharing
- Try it!
- Dropout reduces the effective capacity of a model, but needs more training time
- Efficient regularization method, can be used with L2

# Dropout: Nowadays

- Usually does not work well when combined with batch-norm.
- Training takes a bit longer, usually 1.5x
- But, can be used for uncertainty estimation.
- Monte Carlo dropout (Yarin Gal and Zoubin Ghahramani series of papers).

⇒ How  
confident  
is model

# Monte Carlo Dropout

- Neural networks are massively overconfident.
- We can use dropout to make the softmax probabilities more calibrated.
- Training: use dropout with a low  $p$  (0.1 or 0.2).
- Inference, run the same image multiple times (25-100), and average the results.

Gal et al., Bayesian Convolutional Neural Networks with Bernoulli Approximate Variational Inference, ICLRW 2015

Gal and Ghahramani, Dropout as a Bayesian approximation, ICML 2016

Gal et al., Deep Bayesian Active Learning with Image Data, ICML 2017

Gal, Uncertainty in Deep Learning, PhD thesis 2017



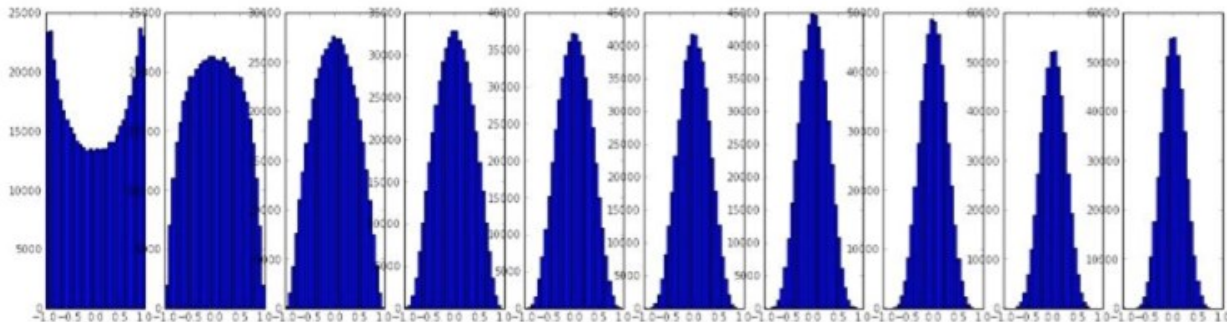
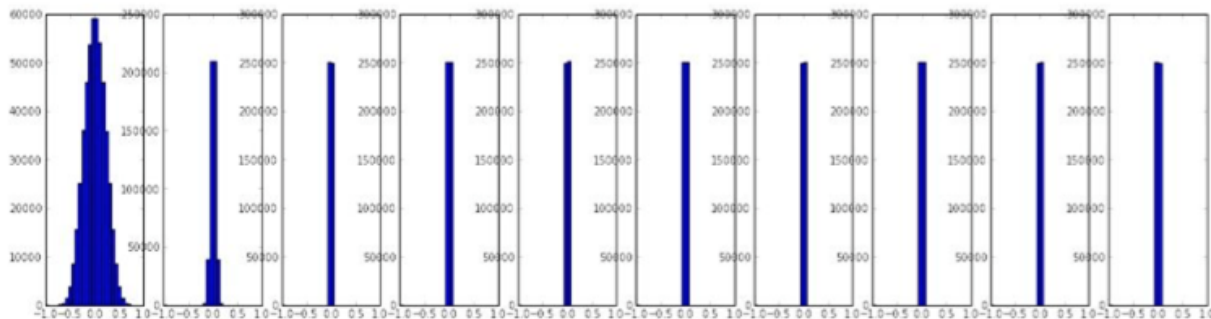
# Batch Normalization: Reducing Internal Covariate Shift

# Batch Normalization: Reducing Internal Covariate Shift

What is internal covariate shift, by the way?

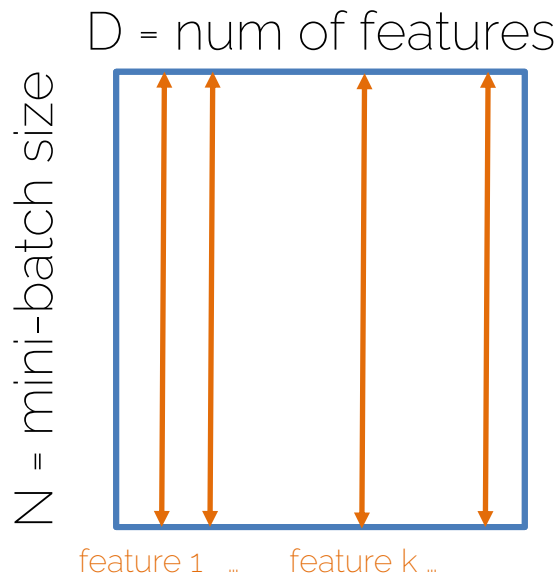
# Our Goal

- 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



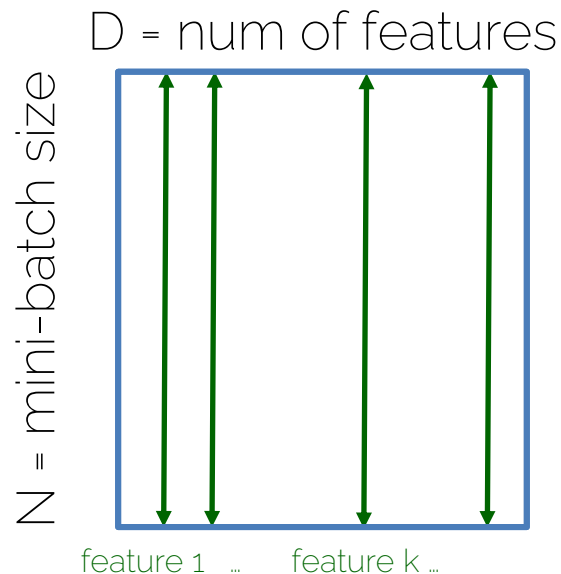
Mean of your mini-batch  
examples over feature k

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

An orange arrow points from the text 'Mean of your mini-batch examples over feature k' to the term  $E[\mathbf{x}^{(k)}]$  in the numerator of the equation.

# 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

Unit gaussian

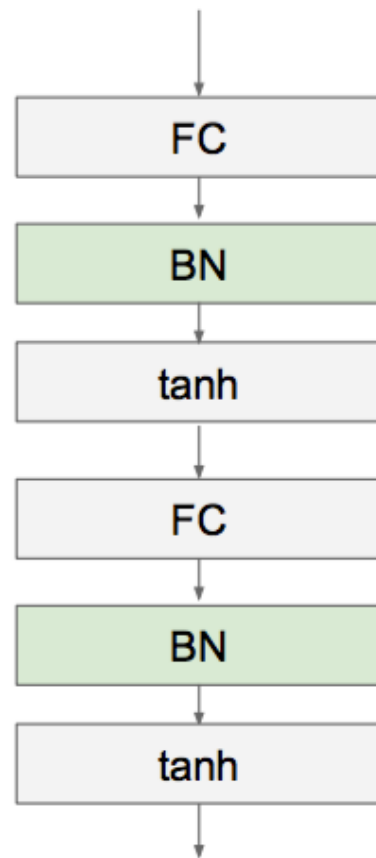
$$\hat{\mathbf{x}}^{(k)} = \frac{\mathbf{x}^{(k)} - E[\mathbf{x}^{(k)}]}{\sqrt{\text{Var}[\mathbf{x}^{(k)}]}}$$

# 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



# Batch Normalization

- 1. Normalize

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

← Differentiable function so we can backprop through it...

- 2. Allow the network to change the range

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

← These parameters will be optimized during backprop



# Batch Normalization

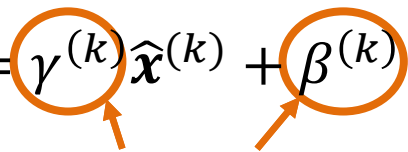
- 1. Normalize

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

- 2. Allow the network to change the range

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

backprop

The diagram shows the equation  $\mathbf{y}^{(k)} = \gamma^{(k)} \hat{\mathbf{x}}^{(k)} + \beta^{(k)}$ . The terms  $\gamma^{(k)}$  and  $\beta^{(k)}$  are circled in orange. Two orange arrows point from these circles down to the word "backprop" centered below the equation.

The network *can* learn to undo the normalization

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

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

# Batch Normalization

- Ok to treat dimensions separately?  
Shown empirically that even if features are not correlated, convergence is still faster with this method

# BN: Train vs Test

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

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

- 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

**Training:** Compute mean and variance from mini-batch 1,2,3 ...

**Testing:** Compute mean and variance by running an exponentially weighted averaged across training mini-batches. Use them as  $\sigma_{test}^2$  and  $\mu_{test}$  .

$$Var_{running} = \beta_m * Var_{running} + (1 - \beta_m) * Var_{minibatch}$$

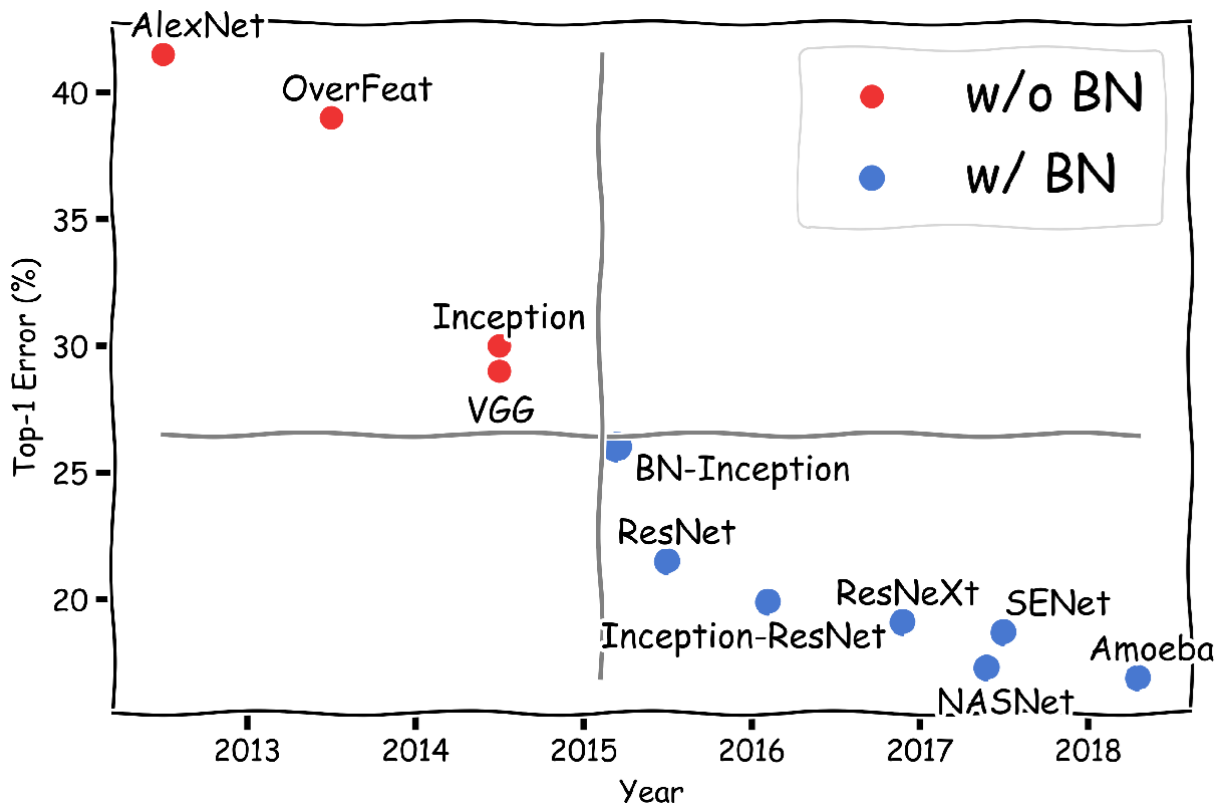
$$\mu_{running} = \beta_m * \mu_{running} + (1 - \beta_m) * \mu_{minibatch}$$

$\beta_m$ : momentum (hyperparameter)

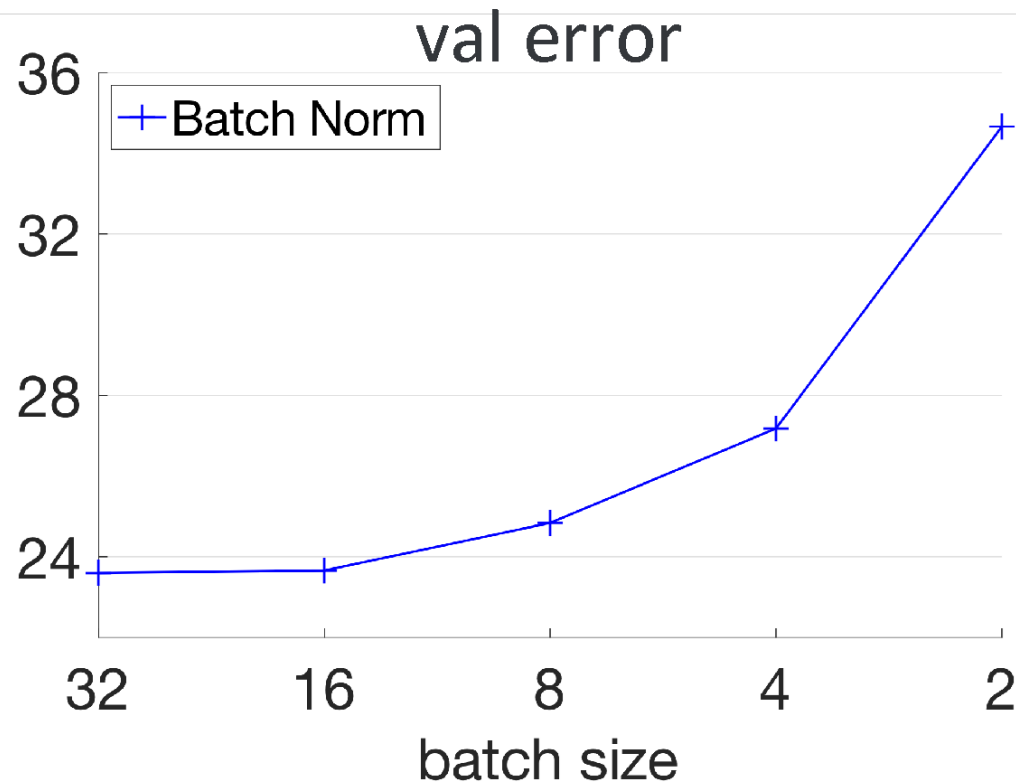
# BN: What do you get?

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

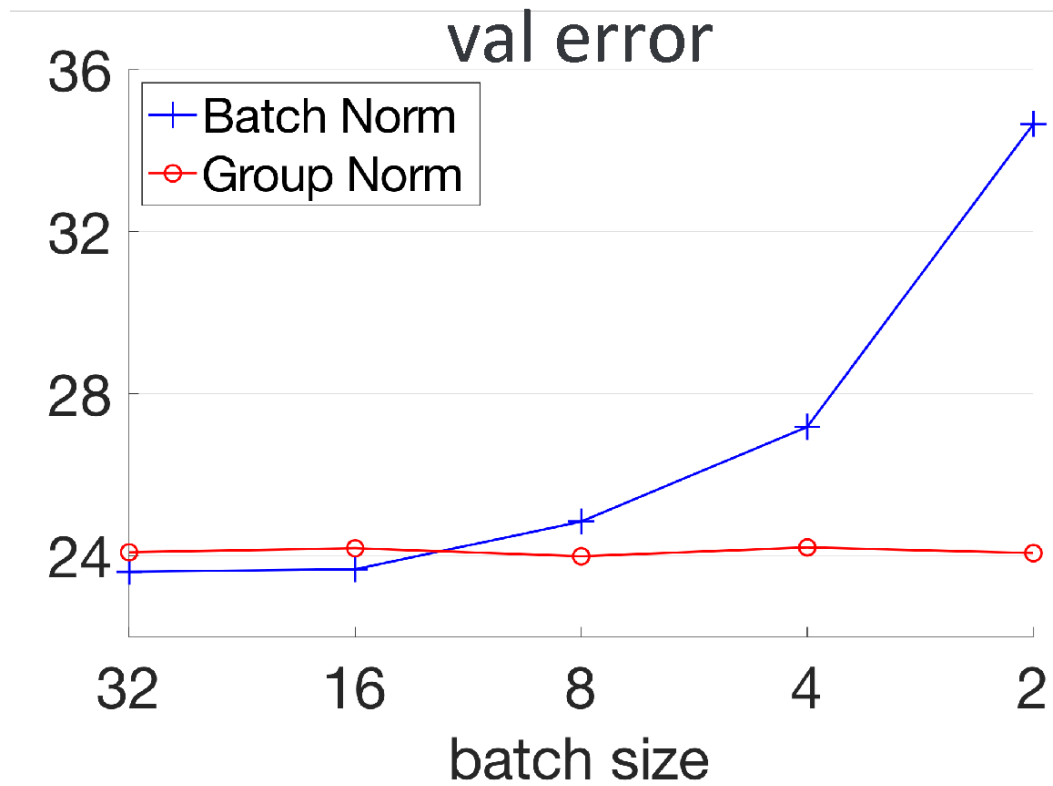
# BN: A Milestone



# BN: Drawbacks



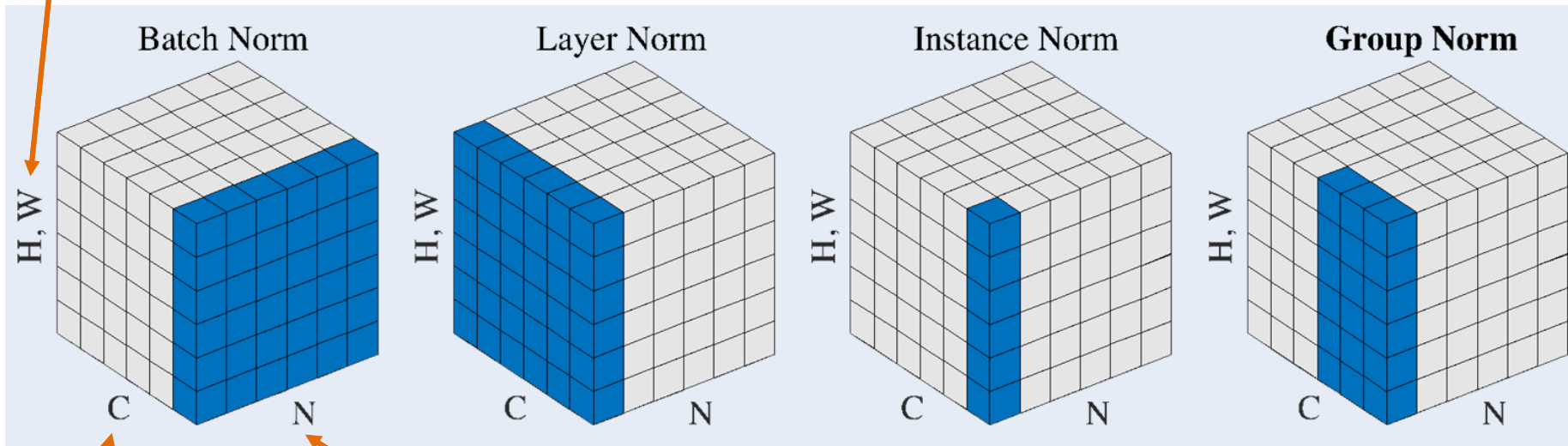
# Other Normalizations





# Other Normalizations

Image size

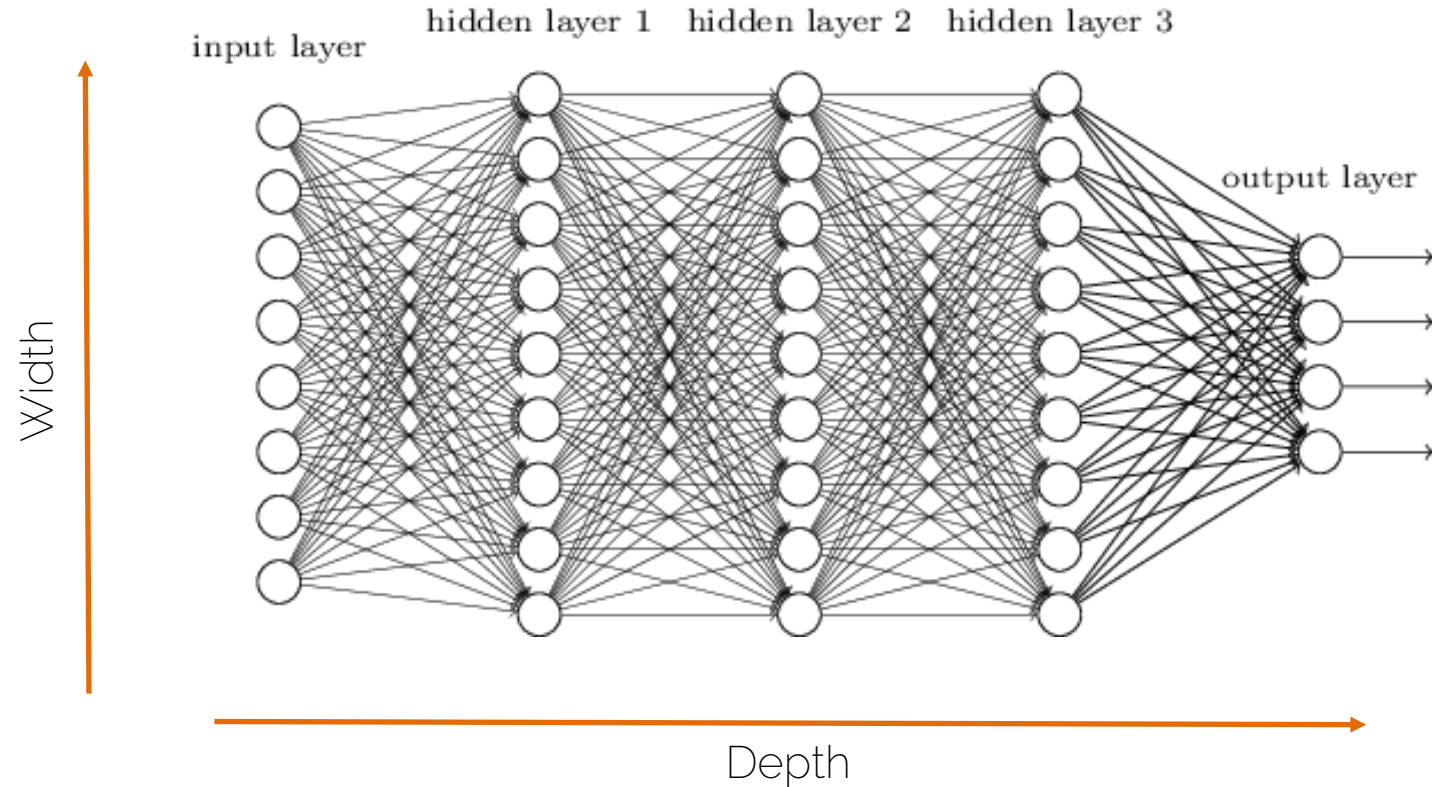


Number of elements in the batch

Number of channels

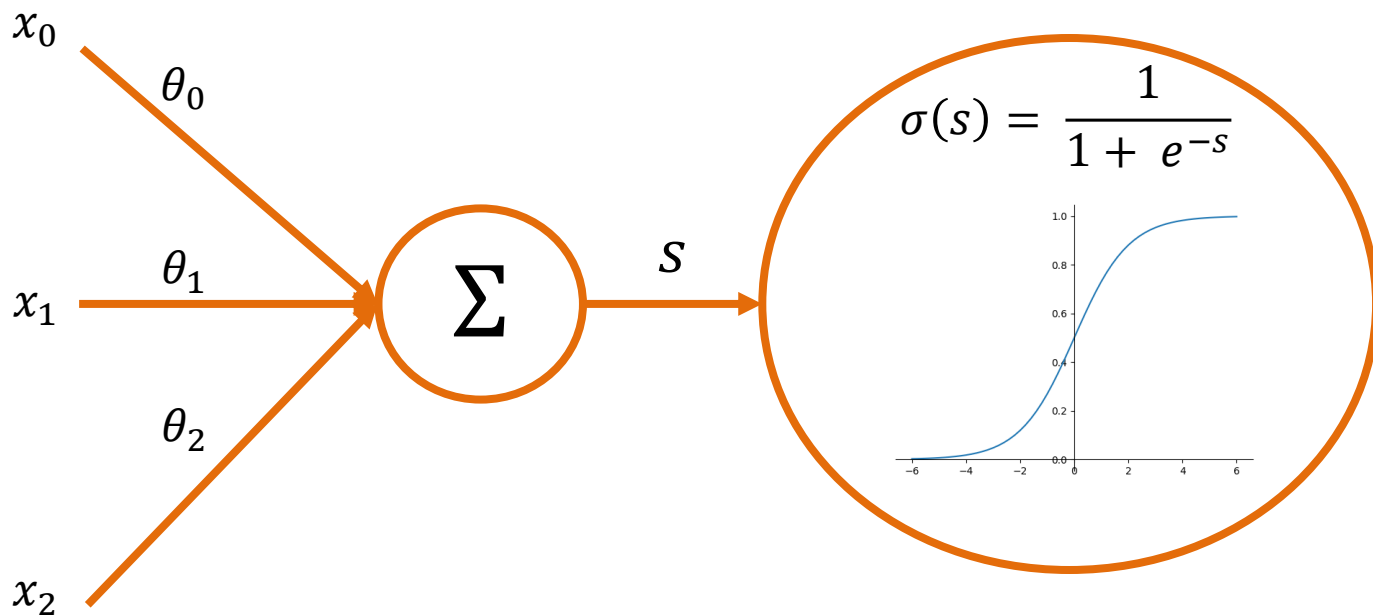
# What We Know

# What do we know so far?



# What do we know so far?

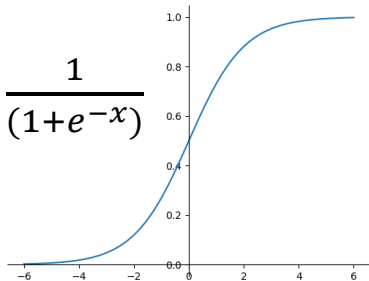
Concept of a 'Neuron'



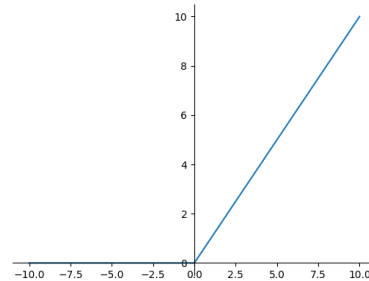
# What do we know so far?

## Activation Functions (non-linearities)

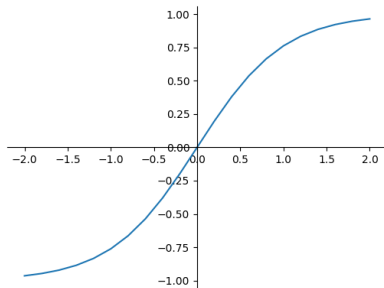
- Sigmoid:  $\sigma(x) = \frac{1}{(1+e^{-x})}$



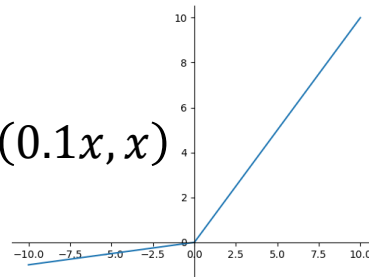
- ReLU:  $\max(0, x)$



- TanH:  $\tanh(x)$

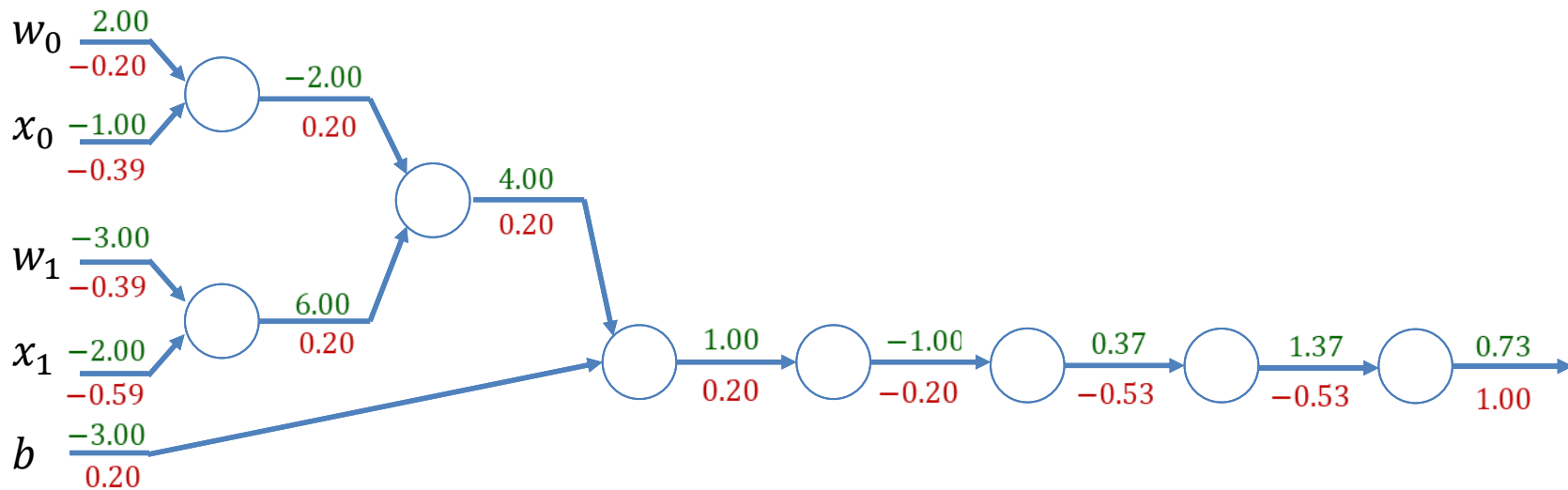


- Leaky ReLU:  $\max(0.1x, x)$



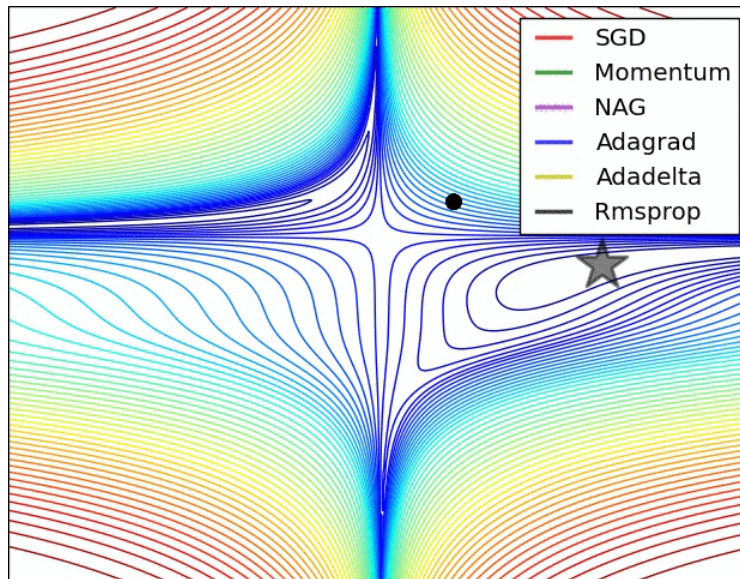
# What do we know so far?

## Backpropagation



# What do we know so far?

SGD Variations (Momentum, etc.)



# What do we know so far?

## Data Augmentation

a. No augmentation (= 1 image)



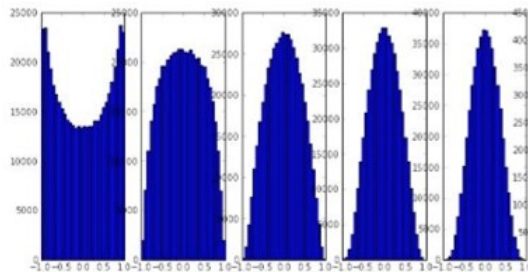
b. Flip augmentation (= 2 images)



## Batch-Norm

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

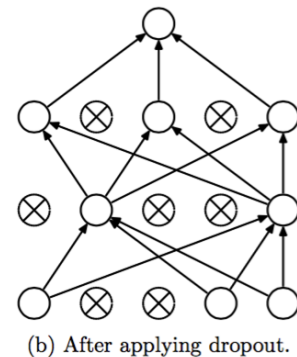
## Weight Initialization (e.g., Kaiming)



## Weight Regularization

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

## Dropout





# Why not simply more layers?

- Neural nets with at least one hidden layer are universal function approximators.
- But generalization is another issue.
- Why not just go deeper and get better?
  - No structure!!
  - It is just brute force!
  - Optimization becomes hard
  - Performance plateaus / drops!
- We need more! More means CNNs, RNNs and eventually Transformers.

See you next week!

# References

- Goodfellow et al. "Deep Learning" (2016),
  - Chapter 6: Deep Feedforward Networks
- Bishop "Pattern Recognition and Machine Learning" (2006),
  - Chapter 5.5: Regularization in Network Nets
- <http://cs231n.github.io/neural-networks-1/>
- <http://cs231n.github.io/neural-networks-2/>
- <http://cs231n.github.io/neural-networks-3/>