Lecture 7 Recap

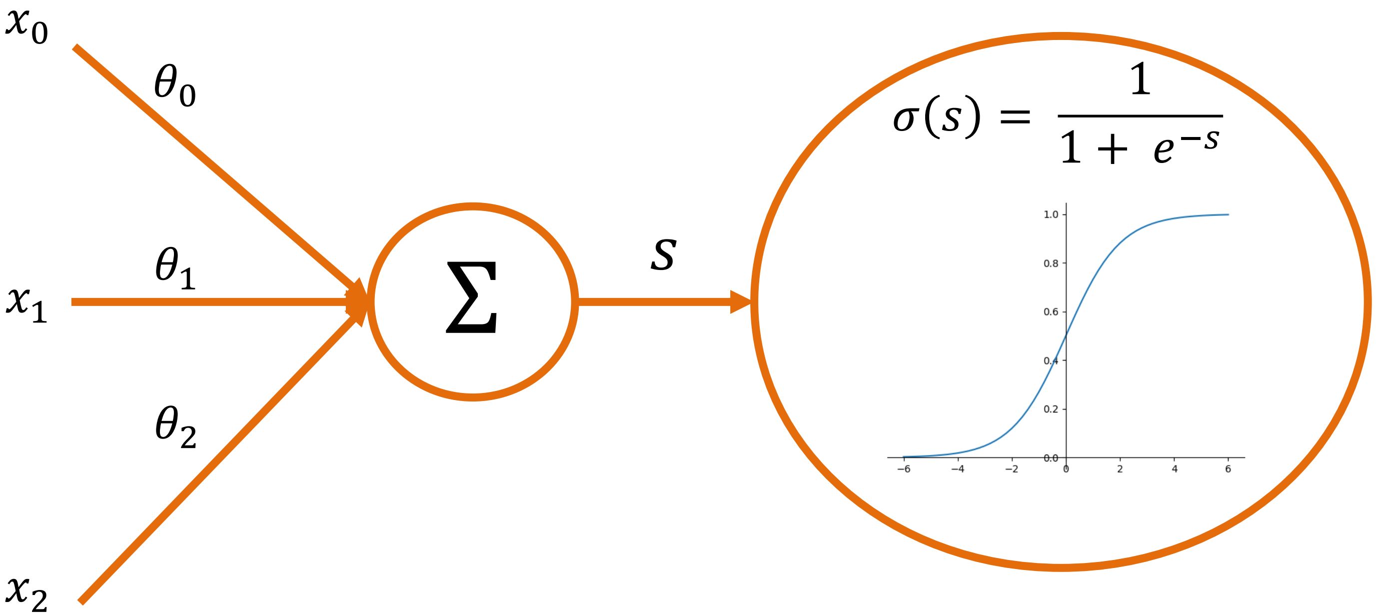
Naïve Losses: L2 vs L1

- L2 Loss:
  - $L^2 = \sum_{i=1}^{n}\left(y_i - f(x_i)\right)^2$
  - 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

$$\sigma(\boldsymbol{x}, \boldsymbol{\theta}) = \frac{1}{1 + e^{-\sum \theta_i x_i}}$$



1

0

Can be interpreted as a probability

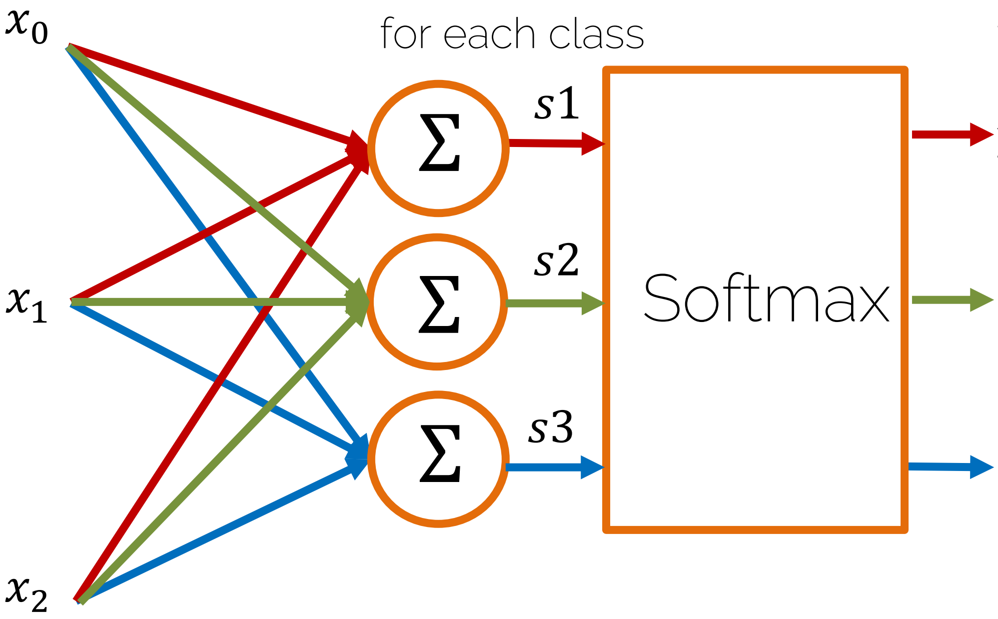
$p(y = 1 | x, \boldsymbol{\theta})$

Softmax Formulation

- What if we have multiple classes?

Scores for each class

Probabilities for each class



$$p(y = 1|\boldsymbol{x}, \boldsymbol{\theta}) = \frac{e^{s1}}{e^{s1} + e^{s2} + e^{s3}}$$

$$p(y = 2|\boldsymbol{x}, \boldsymbol{\theta}) = \frac{e^{s2}}{e^{s1} + e^{s2} + e^{s3}}$$

$$p(y = 3|\boldsymbol{x}, \boldsymbol{\theta}) = \frac{e^{s3}}{e^{s1} + e^{s2} + e^{s3}}$$

Example: Hinge vs Cross-Entropy

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

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

Given the following scores for $\boldsymbol{x}_i$:		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	$s = [5, -20, -20]$ $\boxed{y_i = 0}$	$\max(0, -20 - 5 + 1) +$ $\max(0, -20 - 5 + 1) = 0$	$-\ln\left(\frac{e^5}{e^5+e^{-20}+e^{-20}}\right)$ $= 2 * 10^{-11}$

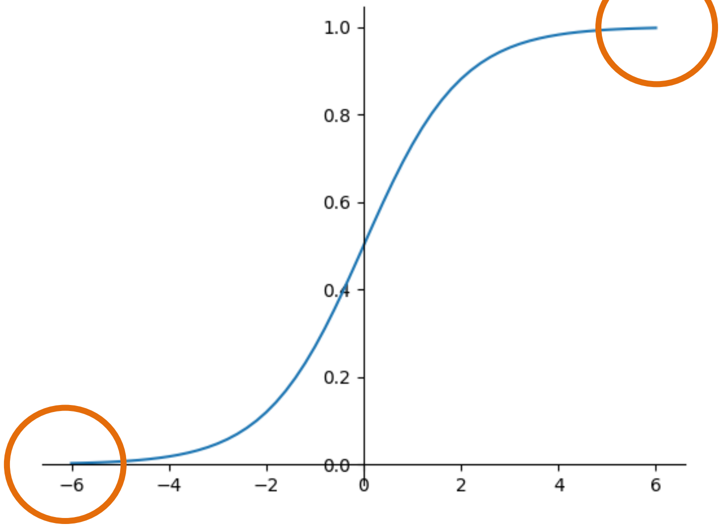
- Cross Entropy *always* wants to improve! (loss never 0)
- Hinge Loss saturates.

Sigmoid Activation

Forward

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

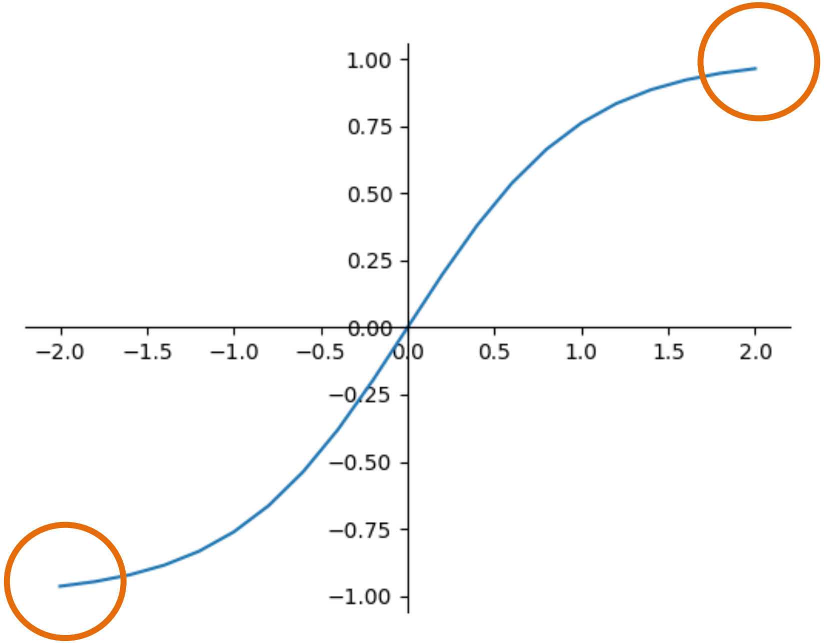
$$\frac{\partial L}{\partial w} = \frac{\partial s}{\partial w}\frac{\partial L}{\partial s}$$



Saturated neurons kill the gradient flow

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

TanH Activation



✗ Still saturates

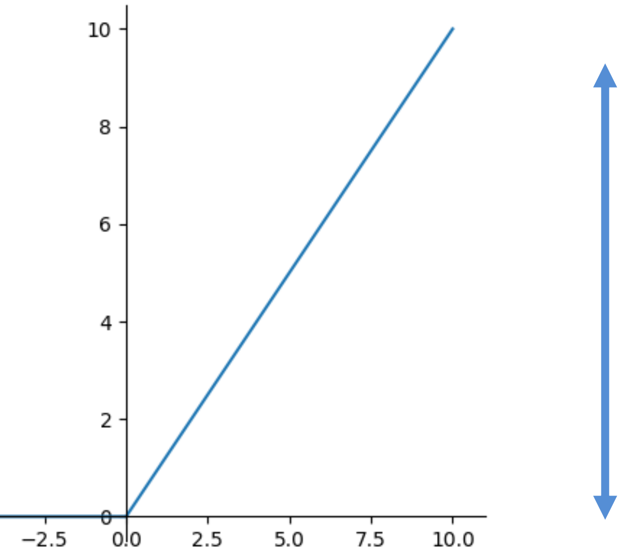
✓ Zero-centered

[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

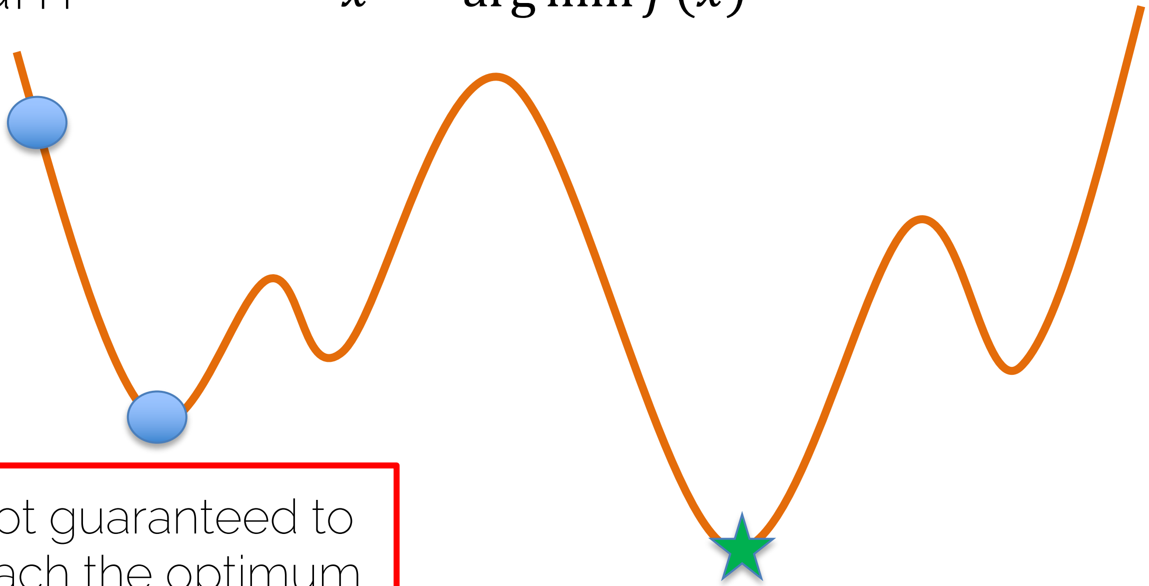
[Krizhevsky et al. NeurIPS 2012] ImageNet Classification with Deep Convolutional Neural Networks

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

Not guaranteed to reach the optimum

Xavier Initialization

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

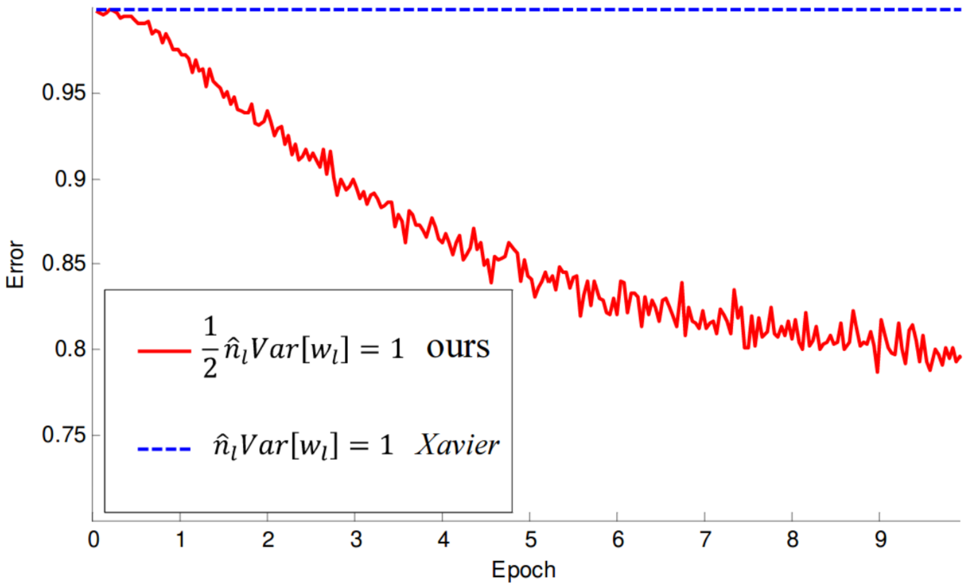
$$(\underbrace{nVar(w)}_{=1}Var(x))$$

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

ReLU Kills Half of the Data

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

It makes a huge difference!



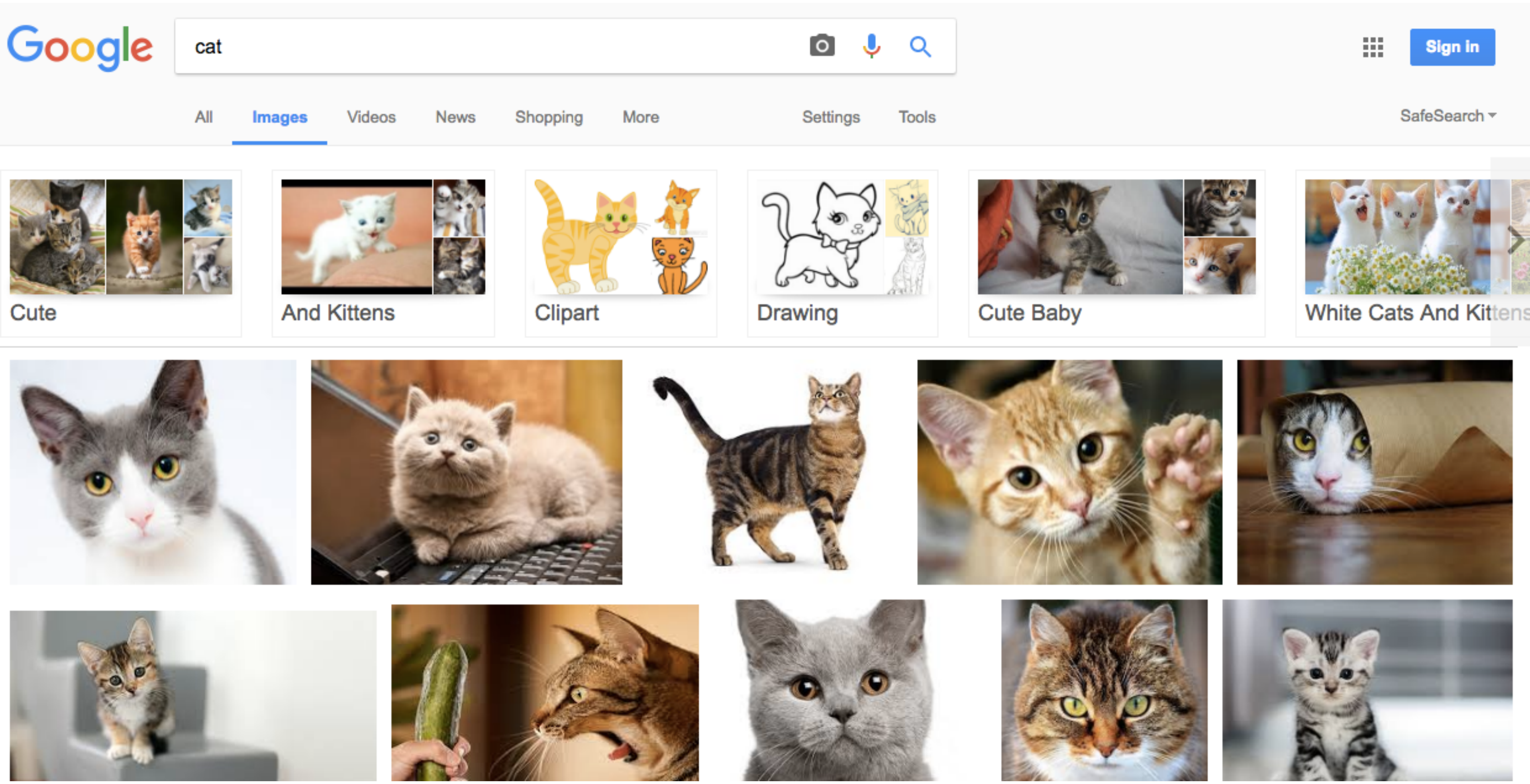
[He et al., ICCV'15] He Initialization

Lecture 8

Data Augmentation

Data Augmentation

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



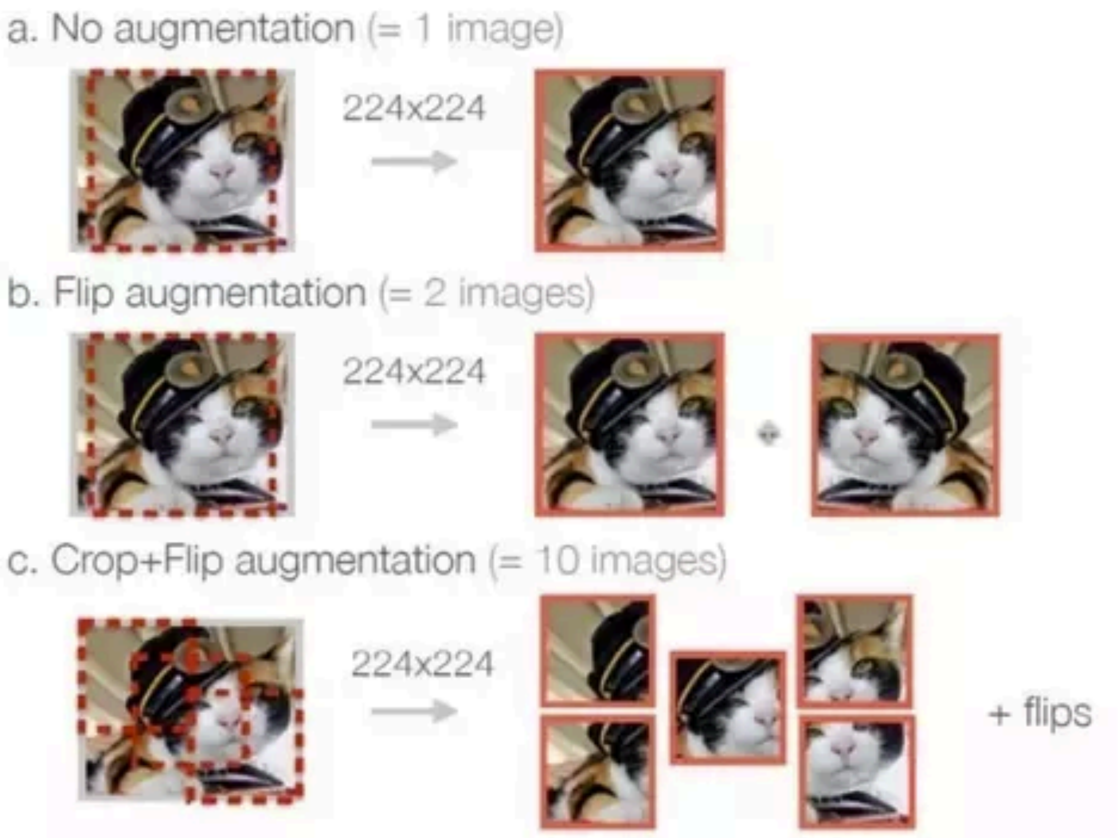
Pose Appearance Illumination

Data Augmentation

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

- Helping the classifier: synthesize data simulating plausible transformations

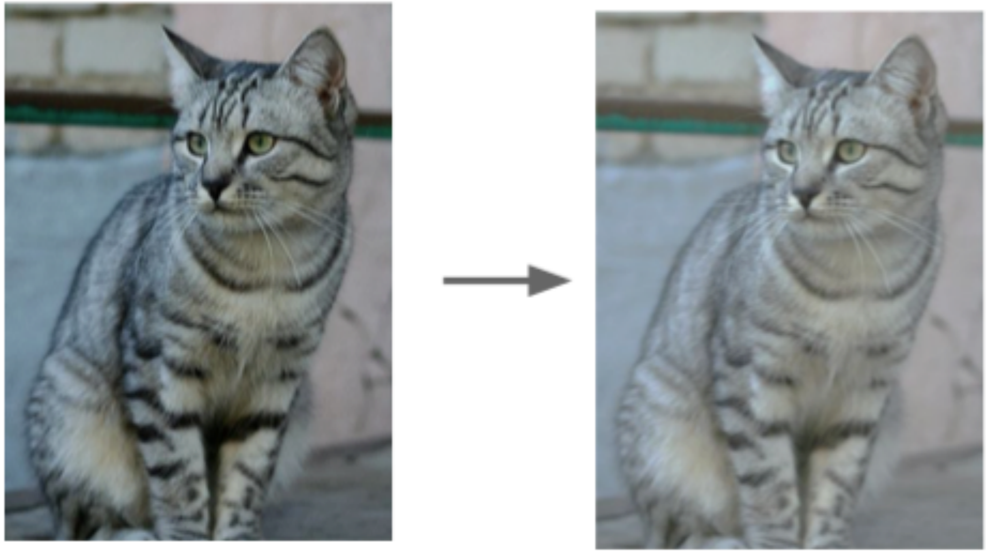
Data Augmentation



[Krizhevsky et al., NIPS'12] ImageNet

Data Augmentation: Brightness

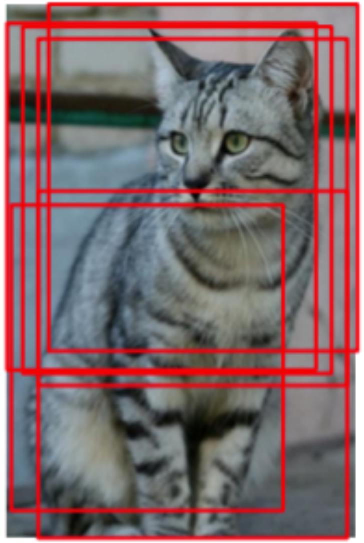
- Random brightness and contrast changes



[Krizhevsky et al., NIPS'12] ImageNet

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 + 1 center) $\times$ 2 flips

[Krizhevsky et al., NIPS'12] ImageNet

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

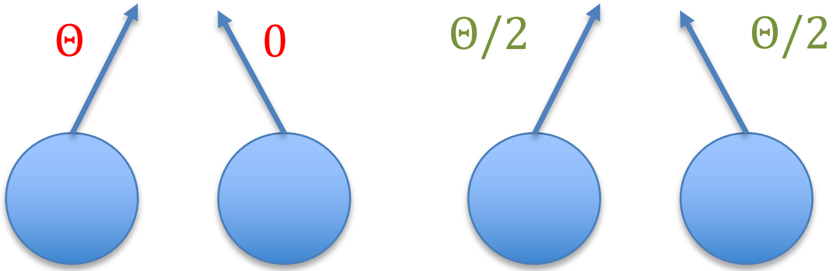
Weight Decay

- L2 regularization

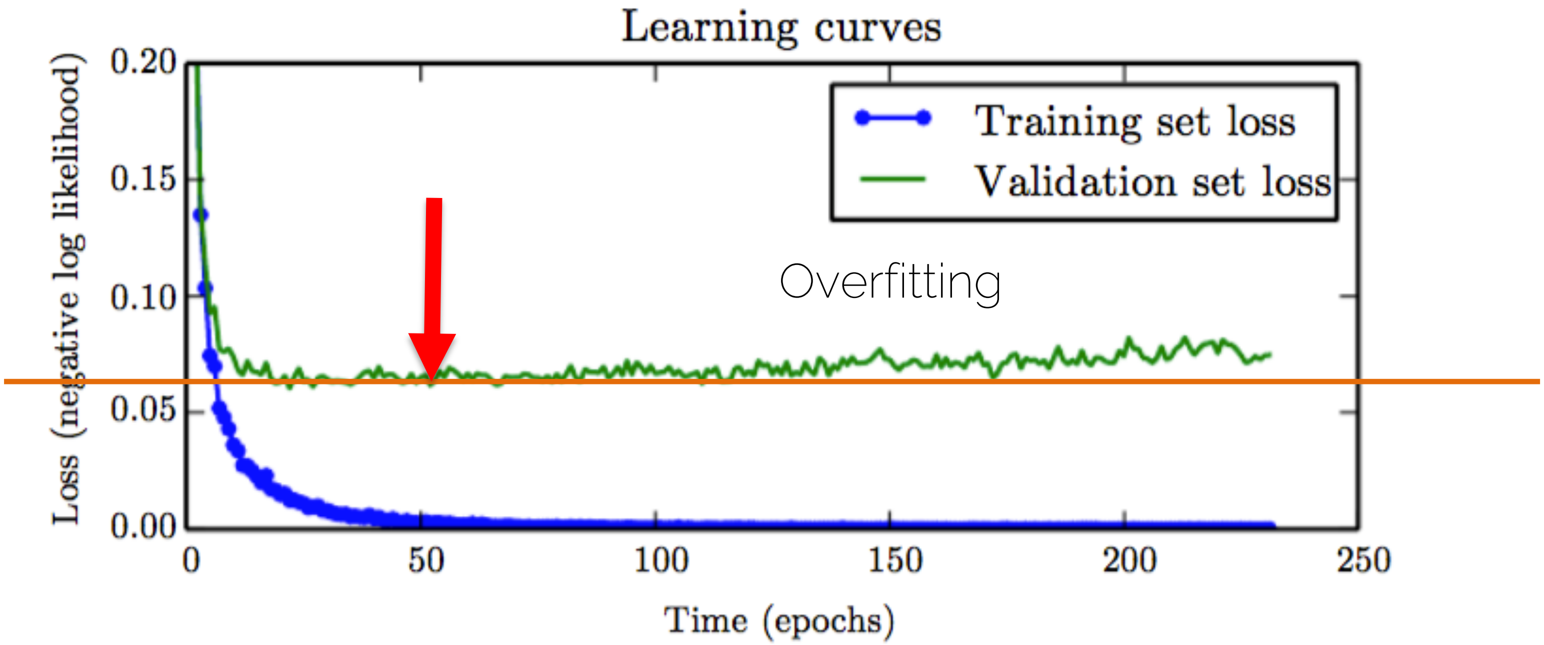
$$\Theta_{k+1} = \Theta_k - \epsilon \nabla_\Theta(\Theta_k, x, y) - \lambda \theta_k$$

Learning rate   Gradient   Gradient of L2-regularization

- Penalizes large weights
- Improves generalization

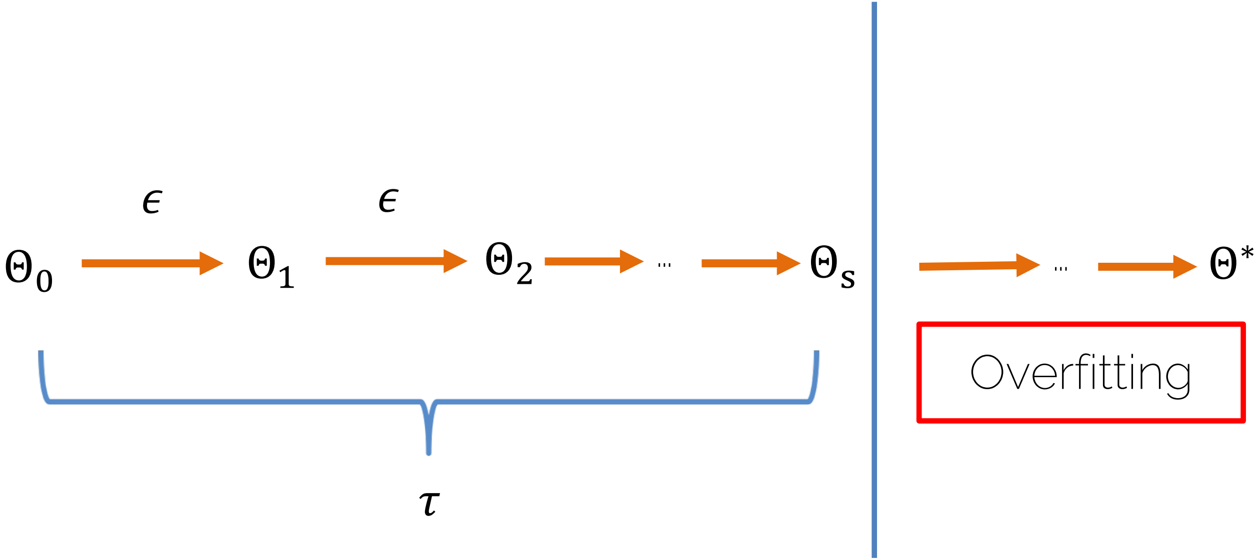


Early Stopping



Early Stopping

- Easy form of regularization



Bagging and Ensemble Methods

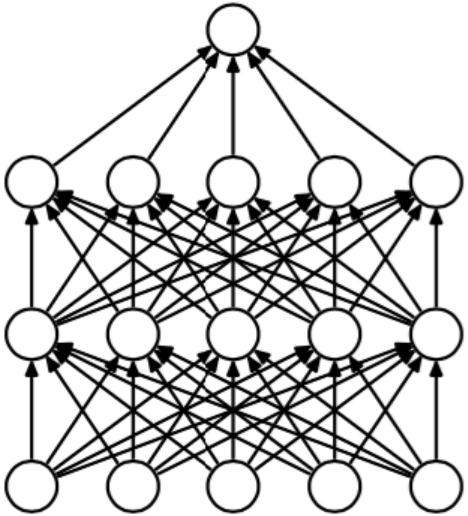
- Train multiple models and average their results

- E.g., use a different algorithm for optimization or change the objective function / loss function.

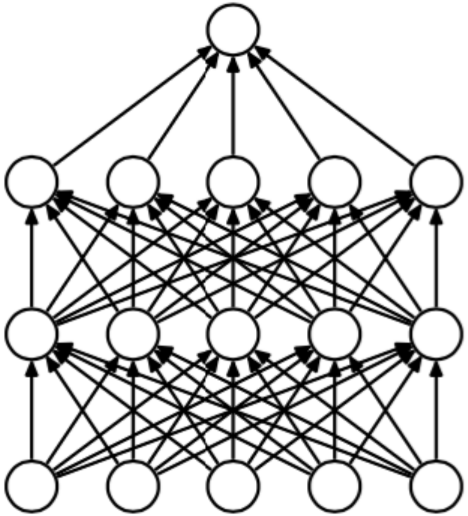
- If errors are uncorrelated, the expected combined error will decrease linearly with the ensemble size

Bagging and Ensemble Methods

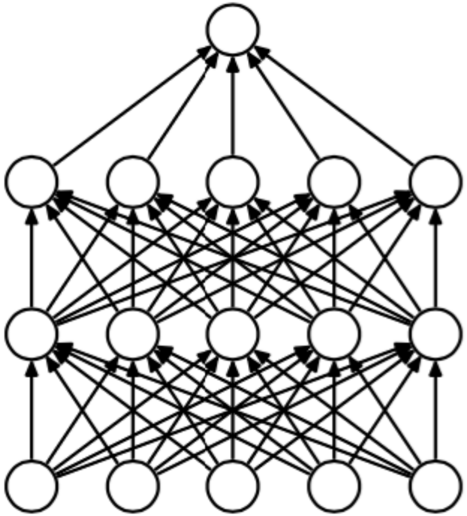
- Bagging: uses k different datasets



Training Set 1



Training Set 2



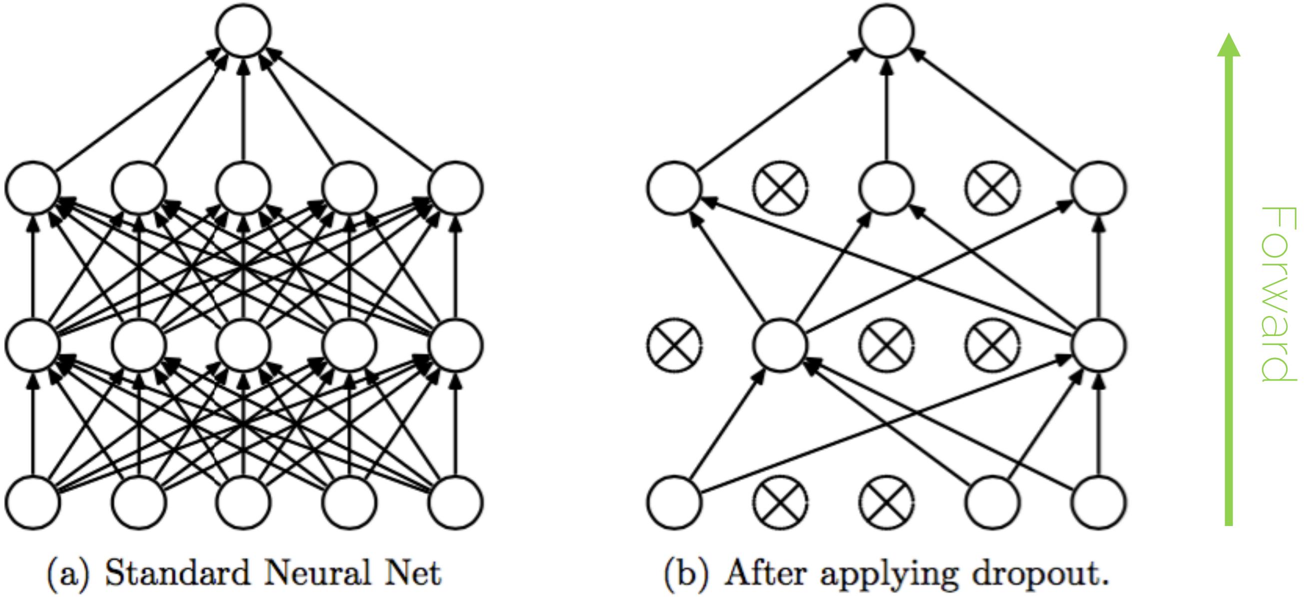
Training Set 3

Image Source: [Srivastava et al., JMLR'14] Dropout

Dropout

Dropout

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

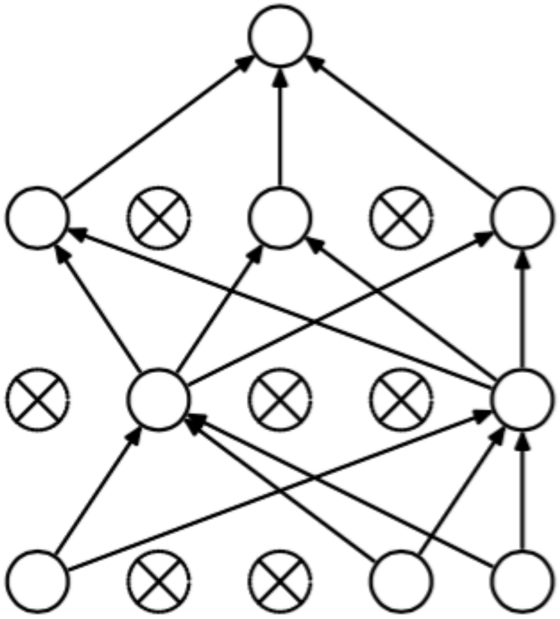


[Srivastava et al., JMLR'14] Dropout

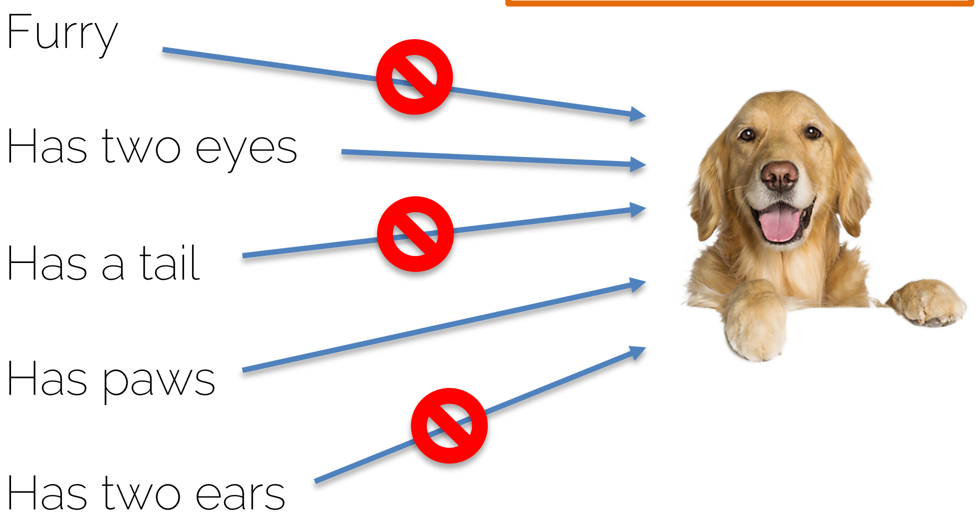
Dropout: Intuition

- Using half the network = half capacity

Redundant representations



(b) After applying dropout.



[Srivastava et al., JMLR'14] Dropout

Dropout: Intuition

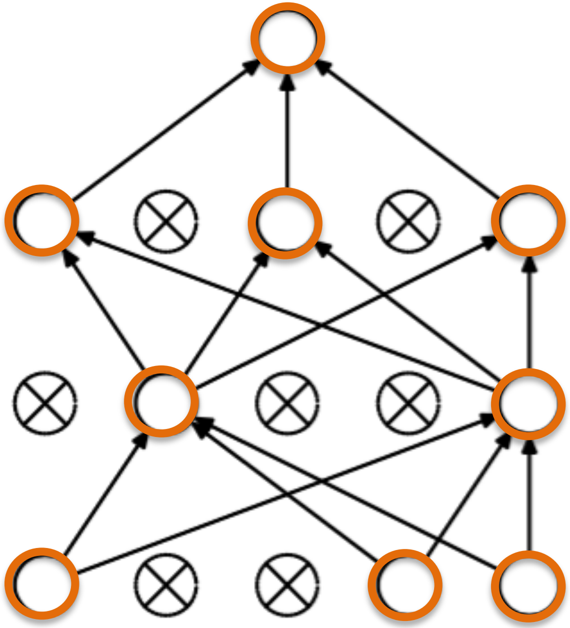
- Using half the network = half capacity
  - Redundant representations
  - Base your scores on more features

- Consider it as a model ensemble

[Srivastava et al., JMLR'14] Dropout

Dropout: Intuition

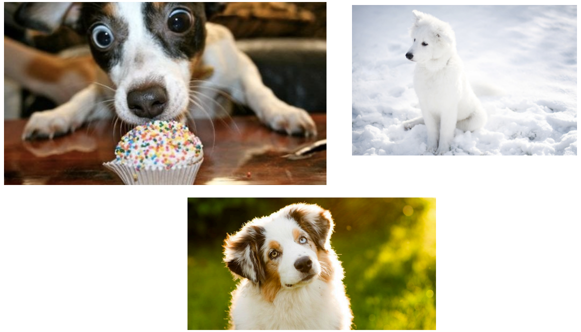
- Two models in one



(b) After applying dropout.

Model 1

Model 2



[Srivastava et al., JMLR'14] Dropout

Dropout: Intuition

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

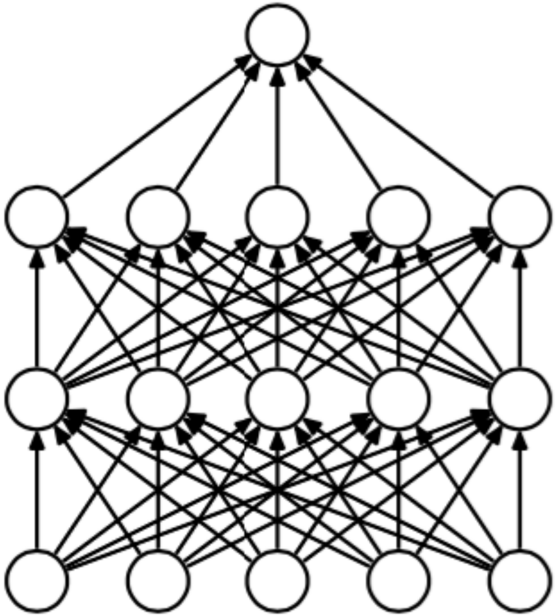
- Consider it as two models in one
  - Training a large ensemble of models, each on different set of data (mini-batch) and with SHARED parameters

Reducing co-adaptation between neurons

[Srivastava et al., JMLR'14] Dropout

Dropout: Test Time

- All neurons are "turned on" – no dropout



Conditions at train and test time are not the same

[Srivastava et al., JMLR'14] Dropout

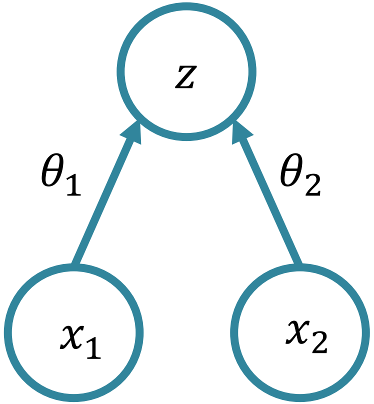
Dropout: Test Time

- Test:

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

Dropout probability
$p = 0.5$

- Train:



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

Weight scaling inference rule

[Srivastava et al., JMLR'14] Dropout

Dropout: Verdict

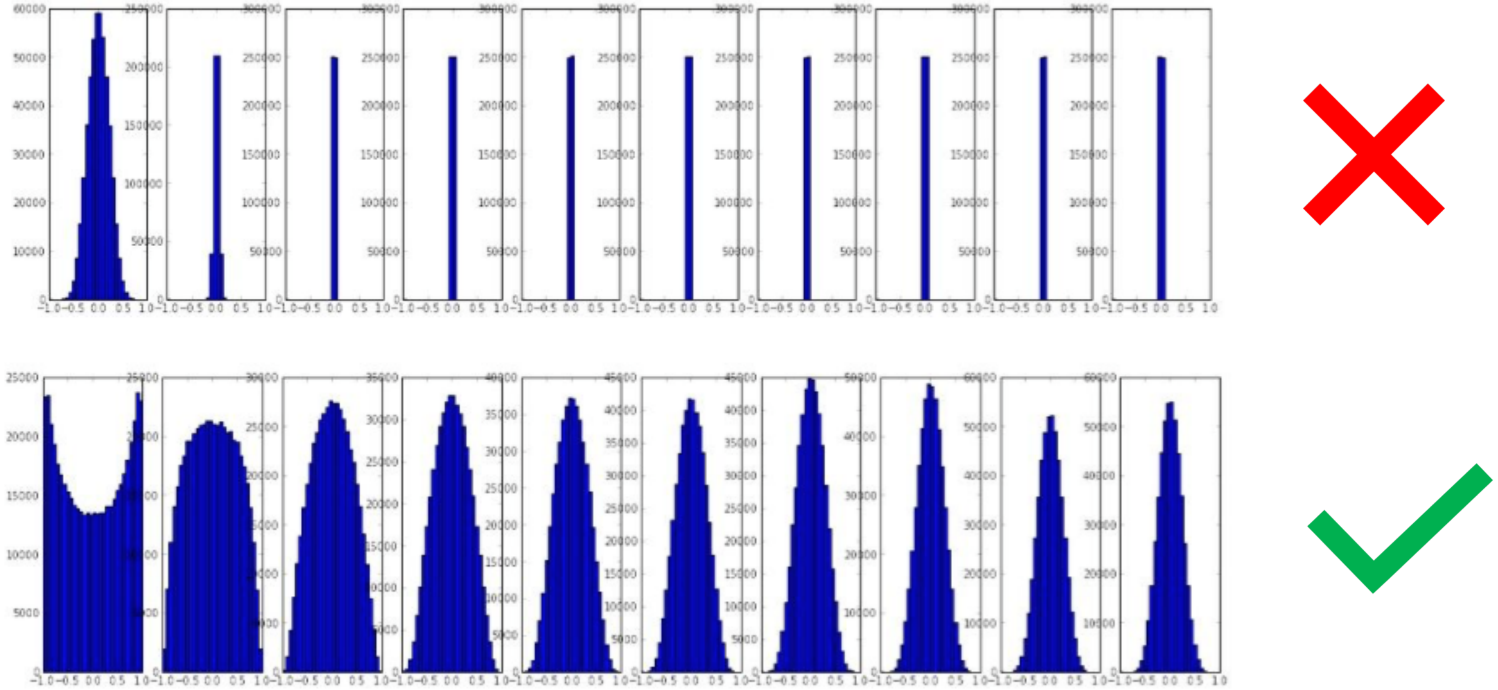
- Efficient bagging method with parameter sharing
- Try it!
- Dropout reduces the effective capacity of a model → larger models, more training time

[Srivastava et al., JMLR'14] Dropout

Batch Normalization

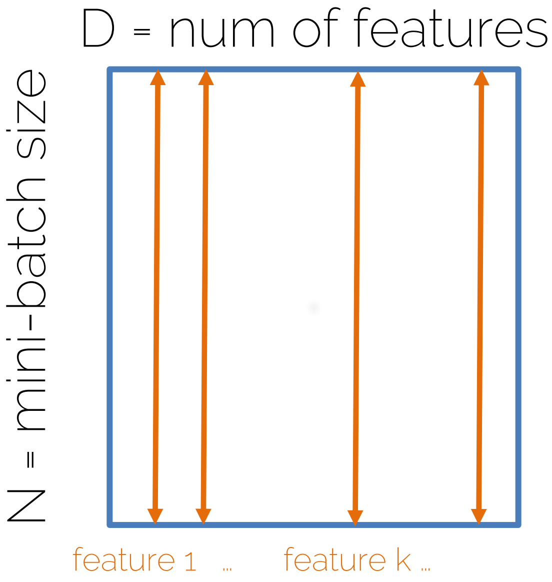
Our Goal

- 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



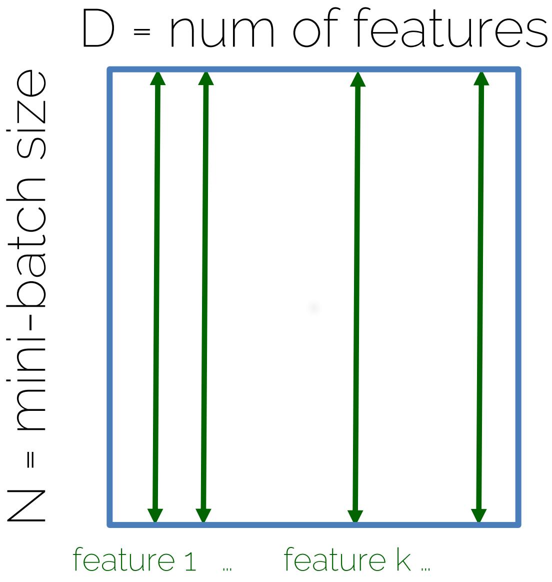
Mean of your mini-batch examples over feature k

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

[Ioffe and Szegedy, PMLR'15] Batch Normalization

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

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

Unit gaussian

[Ioffe and Szegedy, PMLR'15] Batch Normalization

Batch Normalization

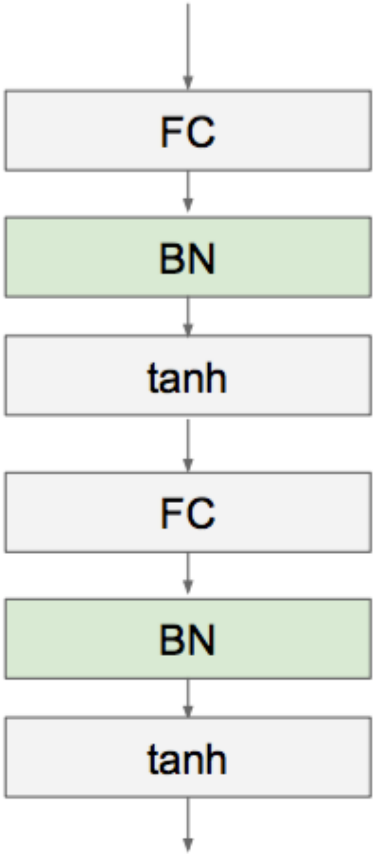
- In each dimension of the features, you have a unit gaussian (in our example)

- For NN in general → BN normalizes the mean and variance of the inputs to your activation functions

[Ioffe and Szegedy, PMLR'15] Batch Normalization

BN Layer

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



[Ioffe and Szegedy, PMLR'15] Batch Normalization

Batch Normalization

- 1. Normalize

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

Differentiable function so we can backprop through it….

- 2. Allow the network to change the range

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

These parameters will be optimized during backprop

[Ioffe and Szegedy, PMLR'15] Batch Normalization

Batch Normalization

- 1. Normalize

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

- 2. Allow the network to change the range

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

backprop

The network *can* learn to undo the normalization

$$\gamma^{(k)} = \sqrt{Var\left[\boldsymbol{x}^{(k)}\right]}$$

$$\beta^{(k)} = E\left[\boldsymbol{x}^{(k)}\right]$$

[Ioffe and Szegedy, PMLR'15] Batch Normalization

Batch Normalization

- Ok to treat dimensions separately?
  Shown empirically that even if features are not correlated, 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

[Ioffe and Szegedy, PMLR'15] Batch Normalization

BN: Train vs Test

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

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

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

[Ioffe and Szegedy, PMLR'15] Batch Normalization

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 σ^2_{test} and μ_{test} .

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

β_m: momentum (hyperparameter)

[Ioffe and Szegedy, PMLR'15] Batch Normalization

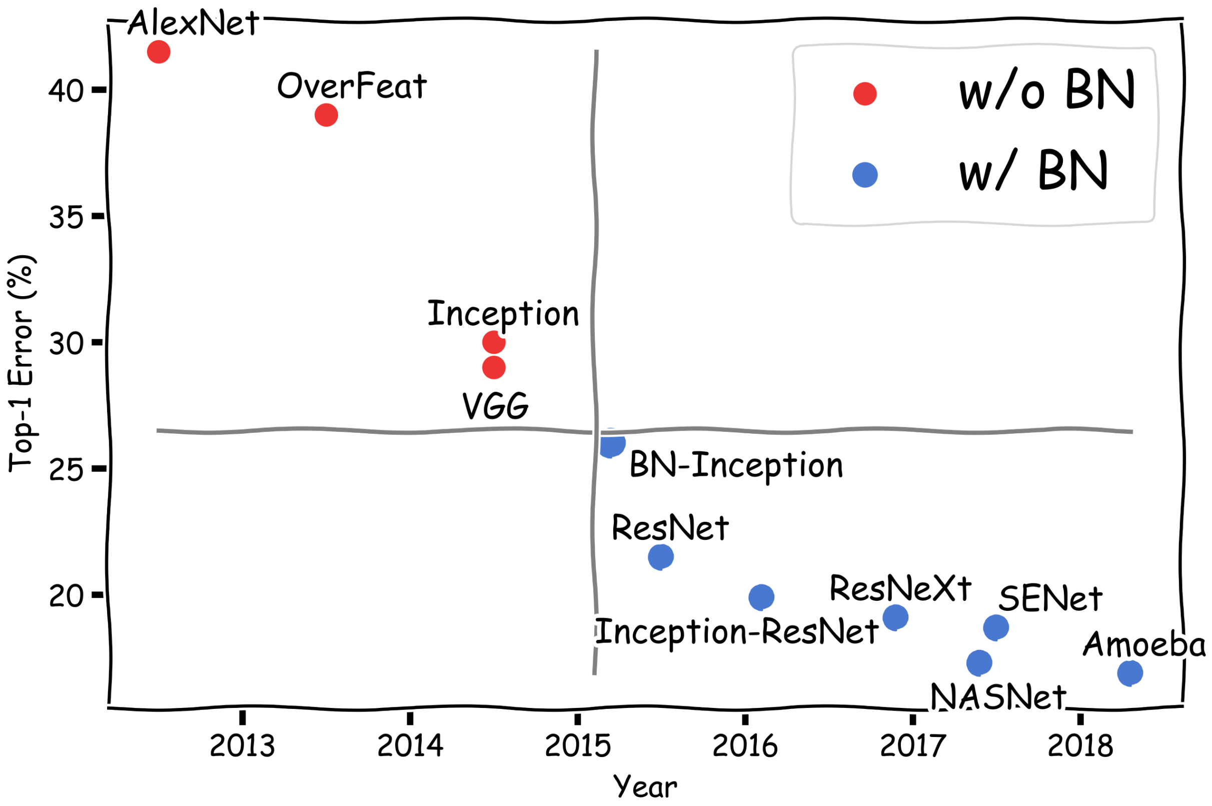
BN: What do you get?

- Very deep nets are much easier to train → more stable gradients

- A much larger range of hyperparameters works similarly when using BN

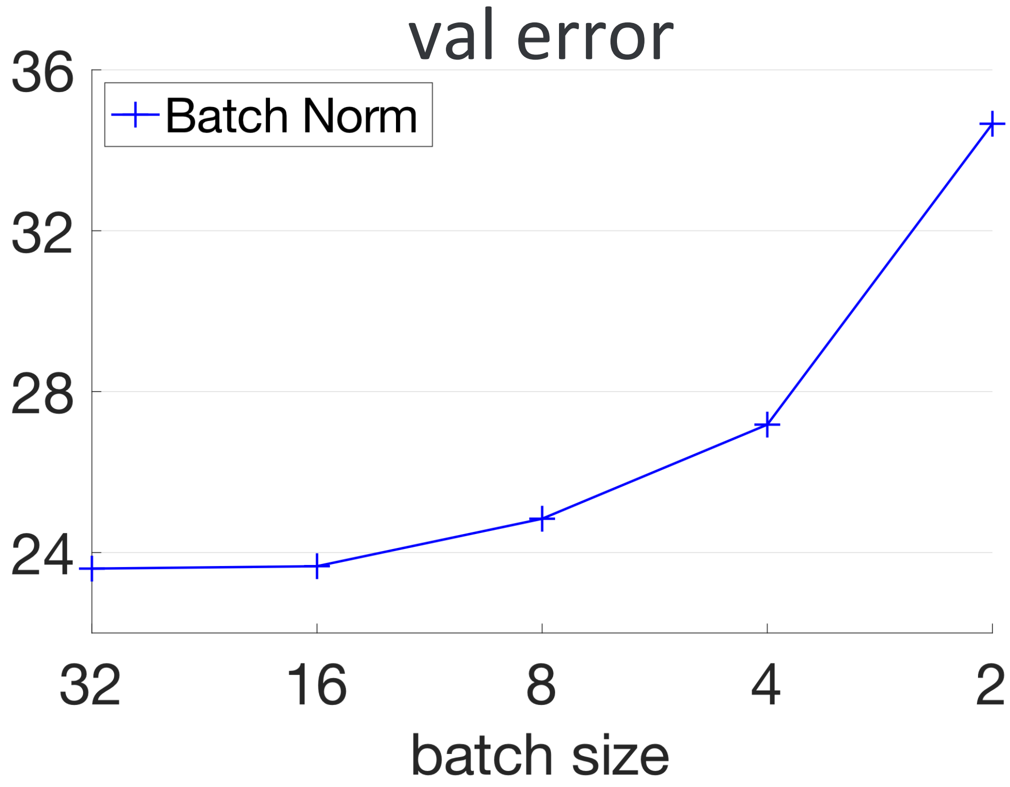
[Ioffe and Szegedy, PMLR'15] Batch Normalization

BN: A Milestone



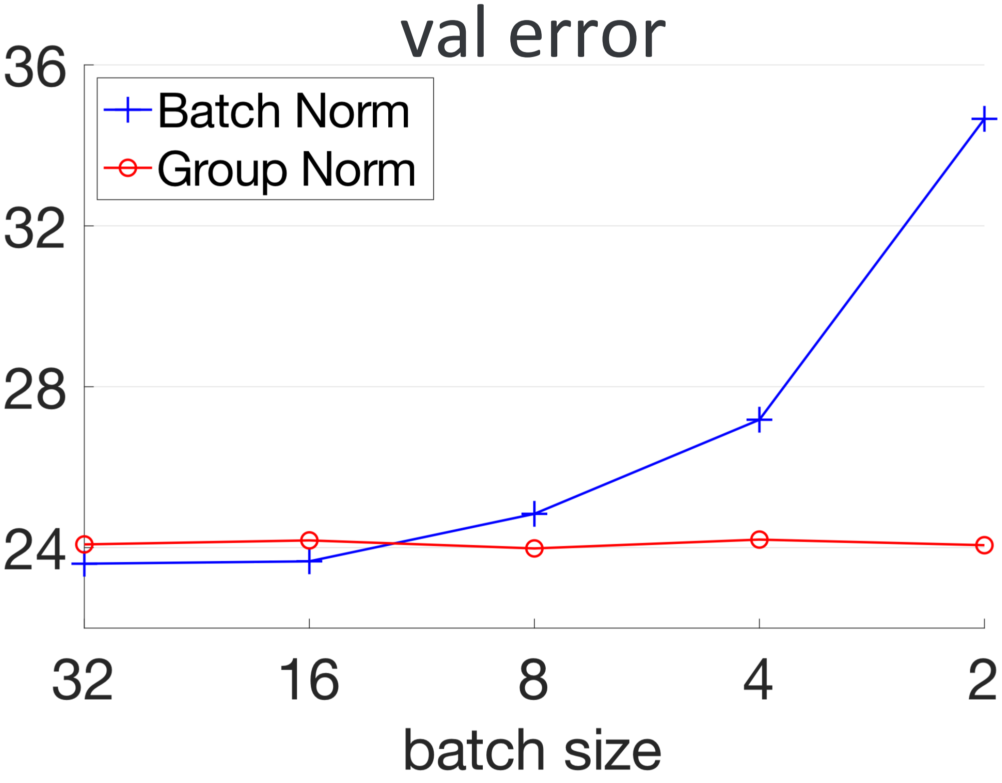
[Wu and He, ECCV'18] Group Normalization

BN: Drawbacks



[Wu and He, ECCV'18] Group Normalization

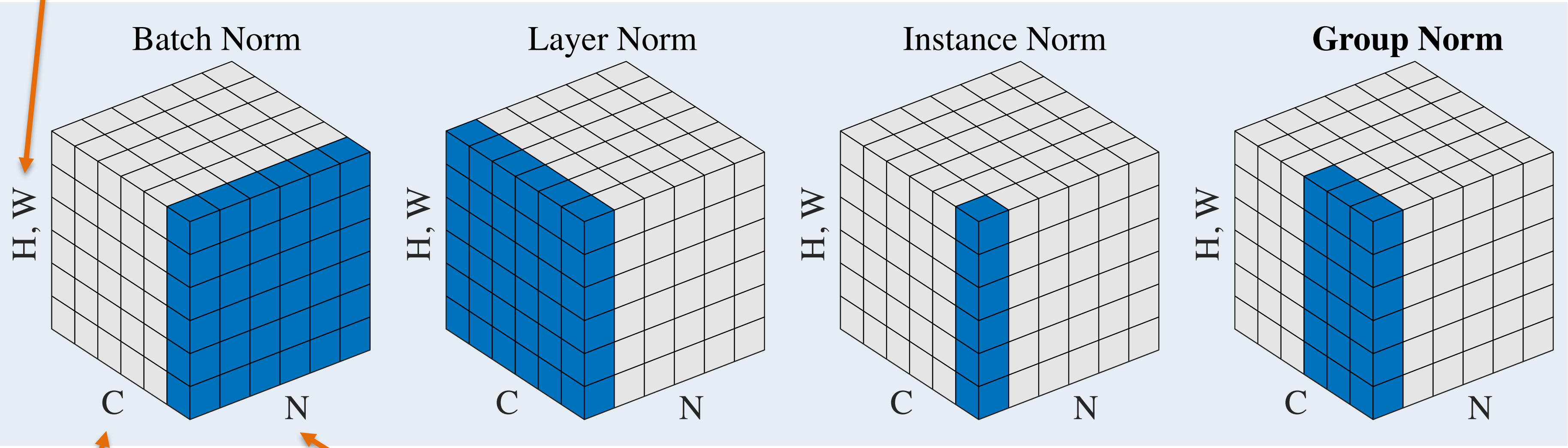
Other Normalizations



[Wu and He, ECCV'18] Group Normalization

Other Normalizations

Image size



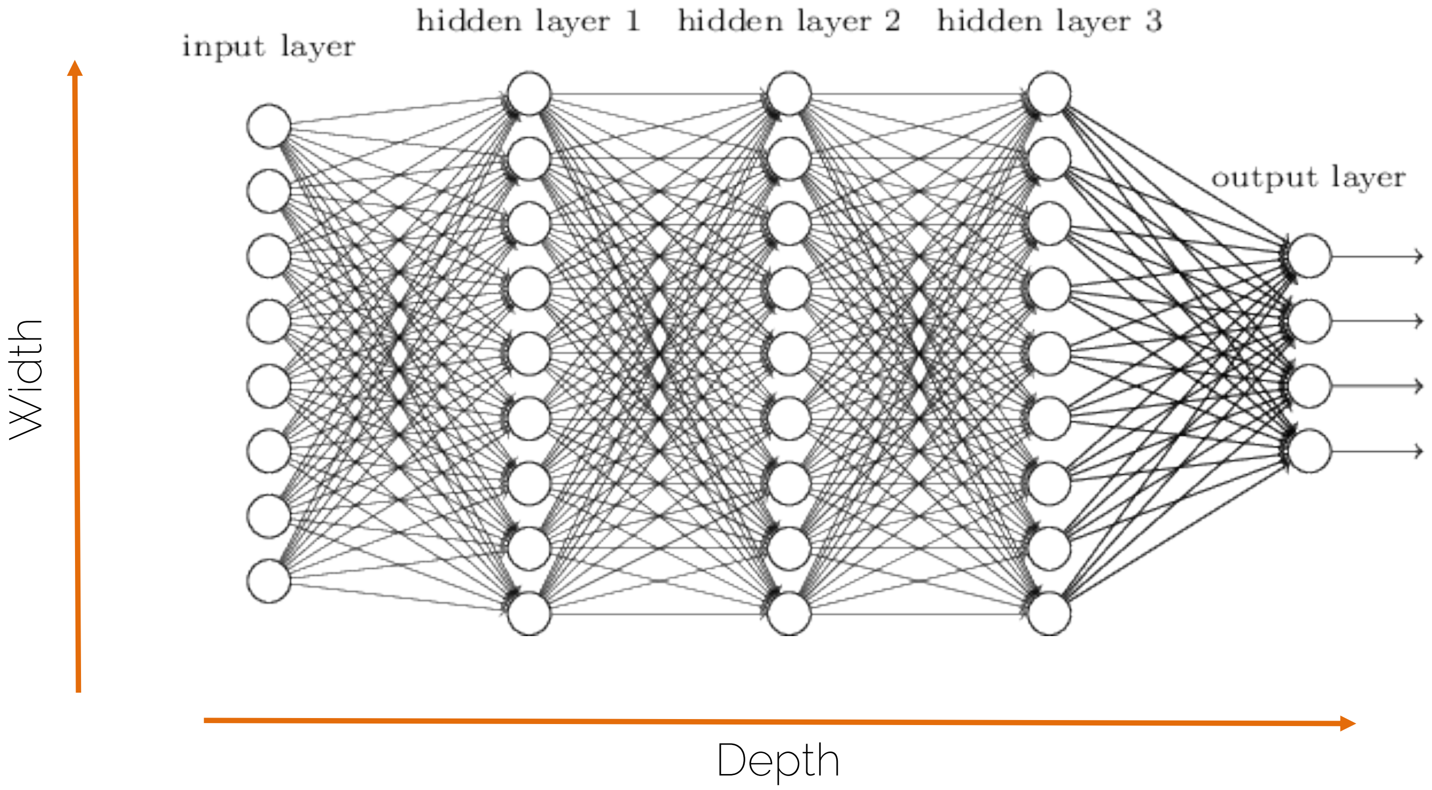
Number of elements in the batch

Number of channels

[Wu and He, ECCV'18] Group Normalization

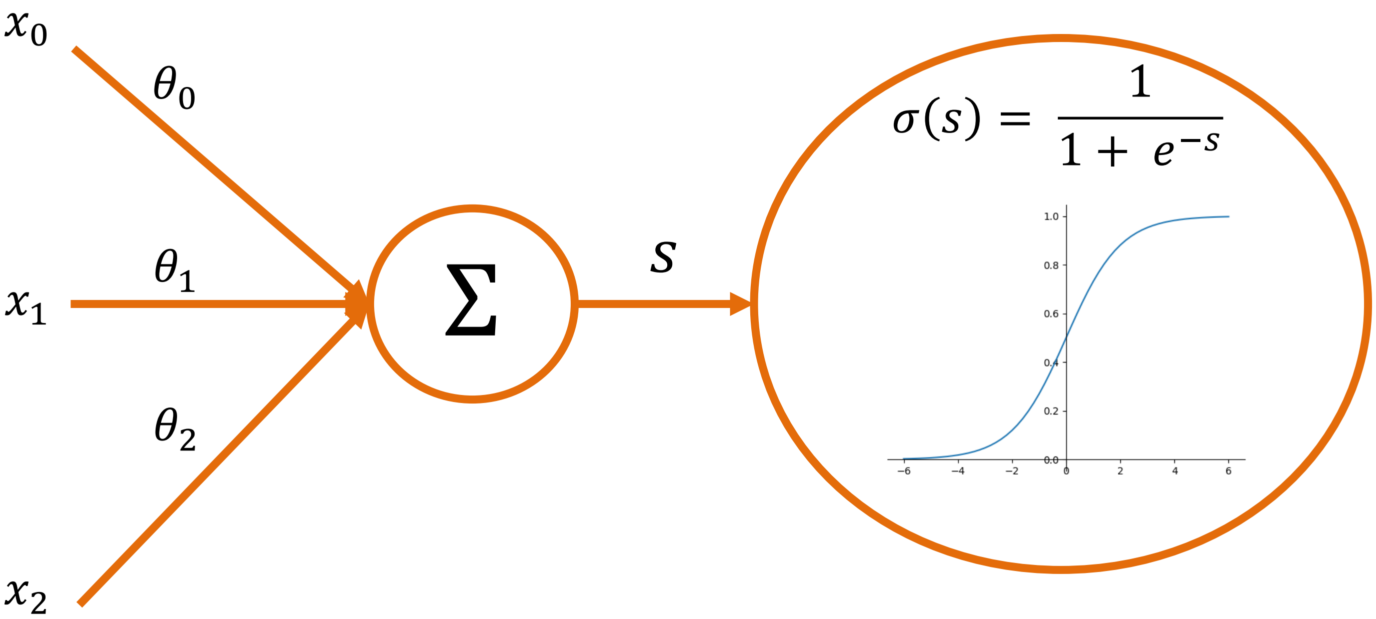
What We Know

What do we know so far?



What do we know so far?

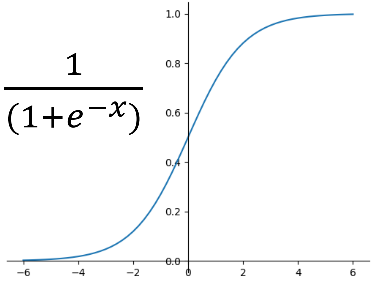
Concept of a 'Neuron'



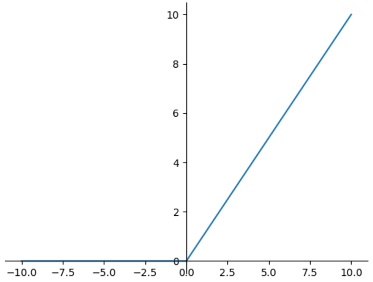
What do we know so far?

Activation Functions (non-linearities)

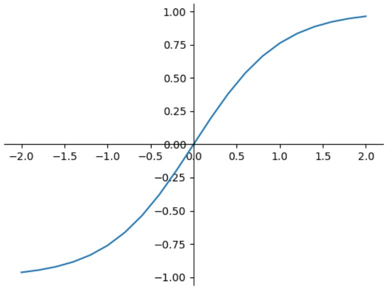
- Sigmoid: $\sigma(x) = \frac{1}{(1+e^{-x})}$



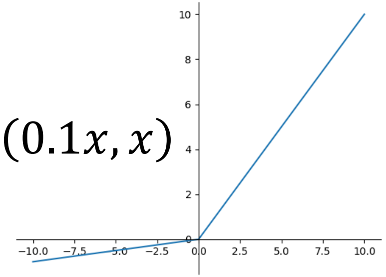
- ReLU: $\max(0, x)$



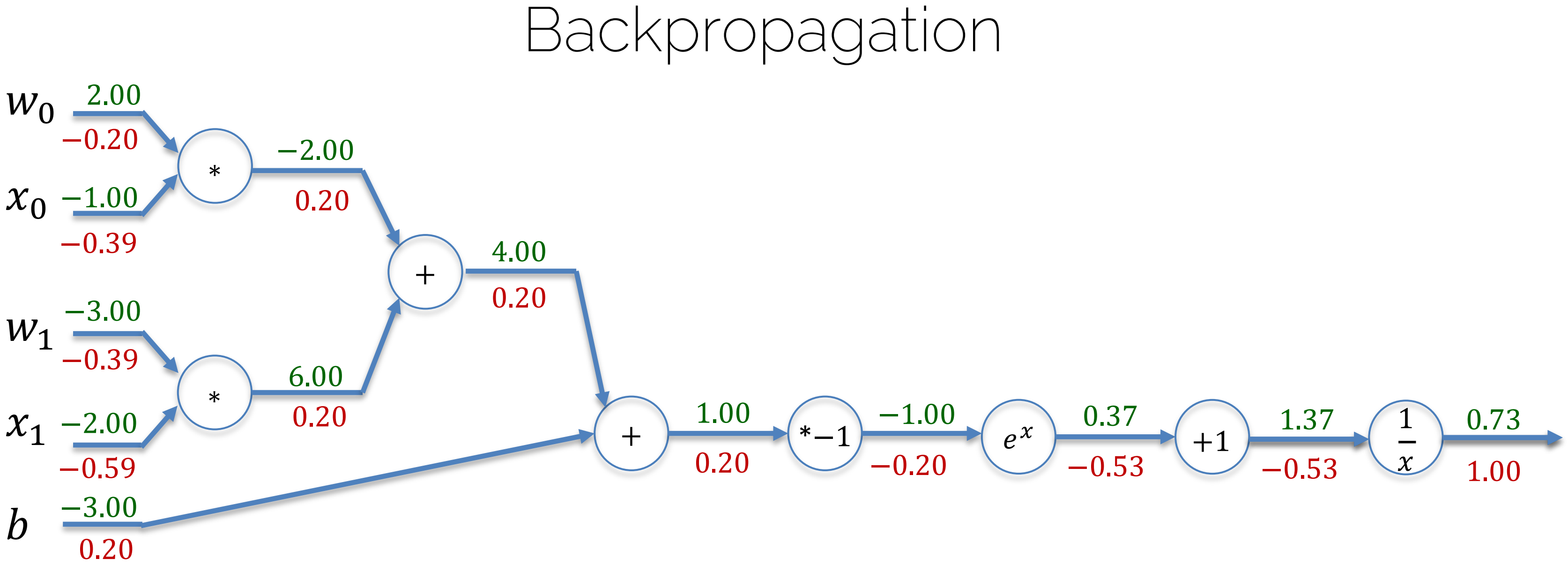
- TanH: $\tanh(x)$



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

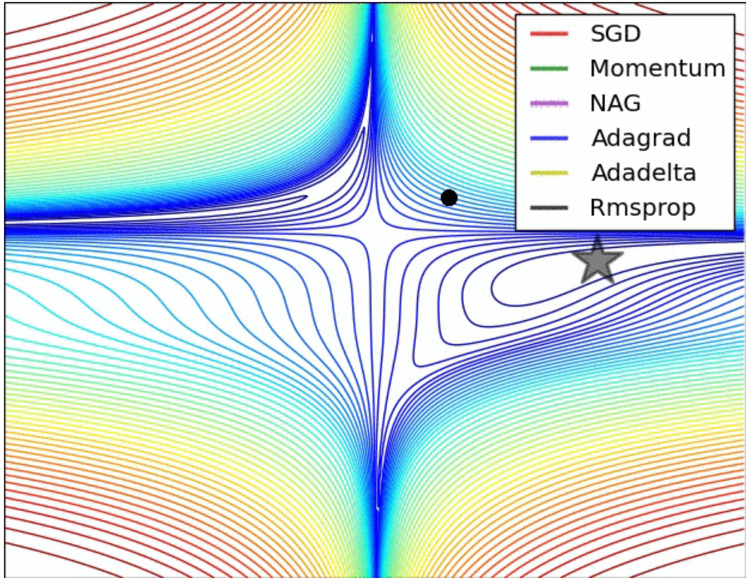


What do we know so far?



What do we know so far?

SGD Variations (Momentum, etc.)



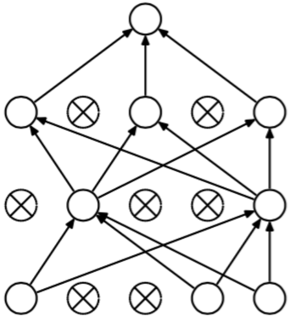
What do we know so far?

Data Augmentation

Batch-Norm

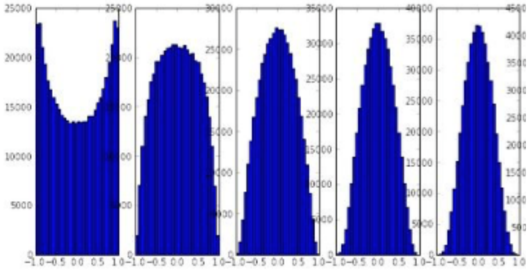
$$\widehat{\boldsymbol{x}}^{(k)} = \frac{\boldsymbol{x}^{(k)} - E\left[\boldsymbol{x}^{(k)}\right]}{\sqrt{Var\left[\boldsymbol{x}^{(k)}\right]}}$$

Dropout



(b) After applying dropout.

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



Weight Regularization

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

Why not simply more Layers?

- We cannot make networks arbitrarily complex

- Why not just go deeper and get better?
  - No structure!!
  - It is just brute force!
  - Optimization becomes hard
  - Performance plateaus / drops!

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/