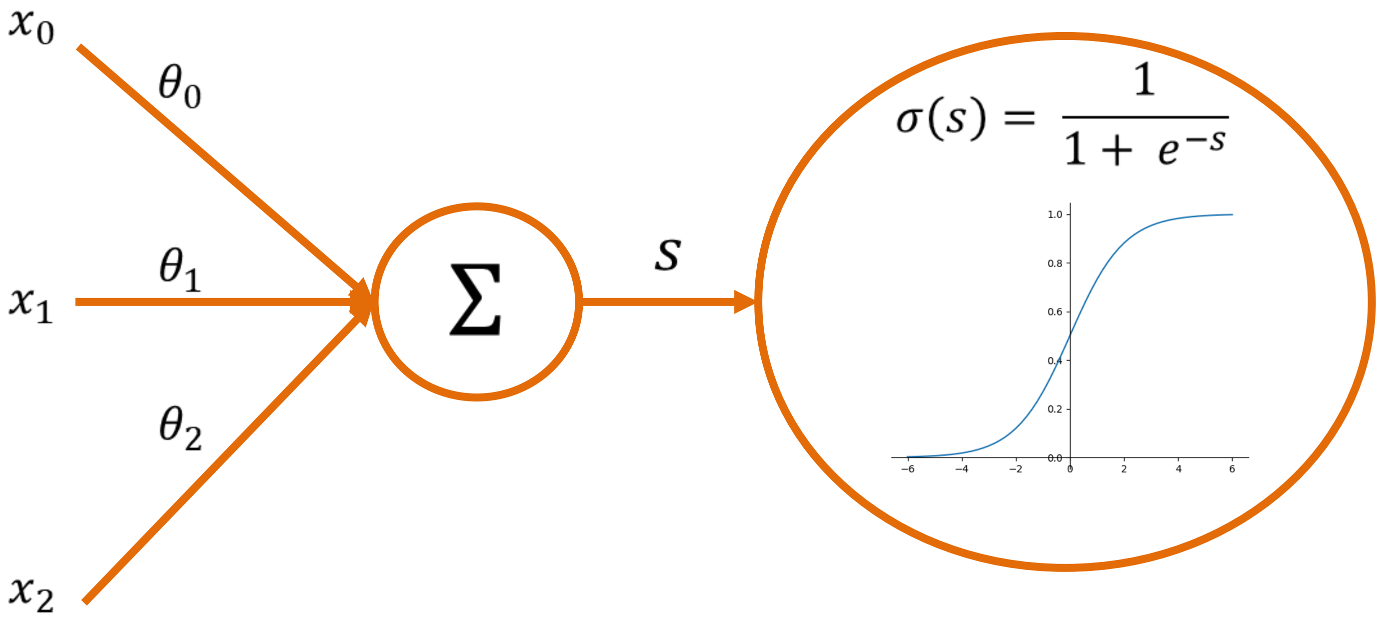
Data Augmentation and Advanced Regularization

Lecture 7 Recap

Binary Classification: Sigmoid

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



1

Can be interpreted as a probability

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

0

Multiclass Classification: Softmax

- Softmax

Exp

$$p(y_i|\boldsymbol{x_i}, \Theta) = \frac{e^{s_{y_i}}}{\sum_{k=1}^{C} e^{s_k}} = \frac{e^{\boldsymbol{x}_i \boldsymbol{\theta}_{y_i}}}{\sum_{k=1}^{C} e^{\boldsymbol{x}_i \boldsymbol{\theta}_k}}$$

Probability of the true class

normalize

training pairs $[\boldsymbol{x}_i; y_i]$,
$\boldsymbol{x}_i \in \mathbb{R}^D, y_i \in \{1,2 \ldots C\}$
y_i: label (true class)

Parameters:
$\boldsymbol{\Theta} = [\boldsymbol{\theta}_1, \boldsymbol{\theta}_2, \ldots, \boldsymbol{\theta}_C]$
$\boldsymbol{C}$: number of classes
$\boldsymbol{s}$: score of the class

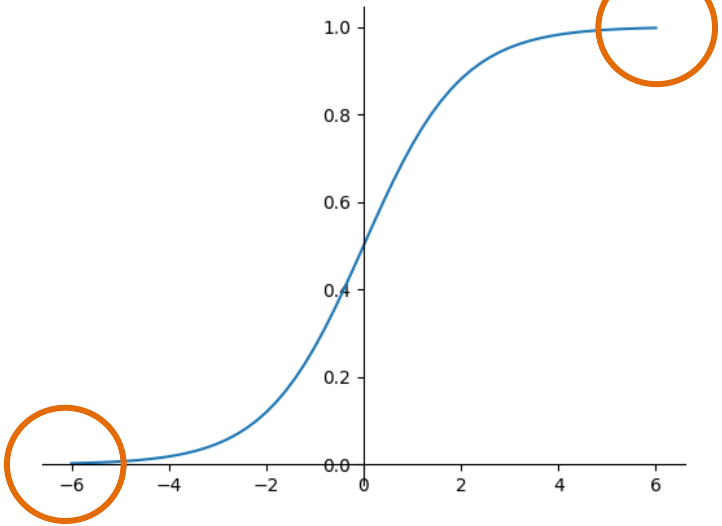
1. Exponential operation: make sure probability>0
2. Normalization: make sure probabilities sum up to 1.

Sigmoid Activation

Forward

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

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



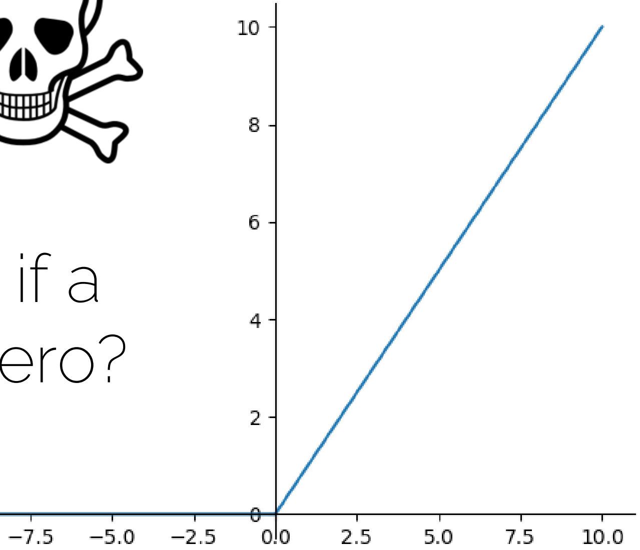
✘ Saturated neurons kill the gradient flow

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

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

Xavier/Kaiming Initialization

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

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

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

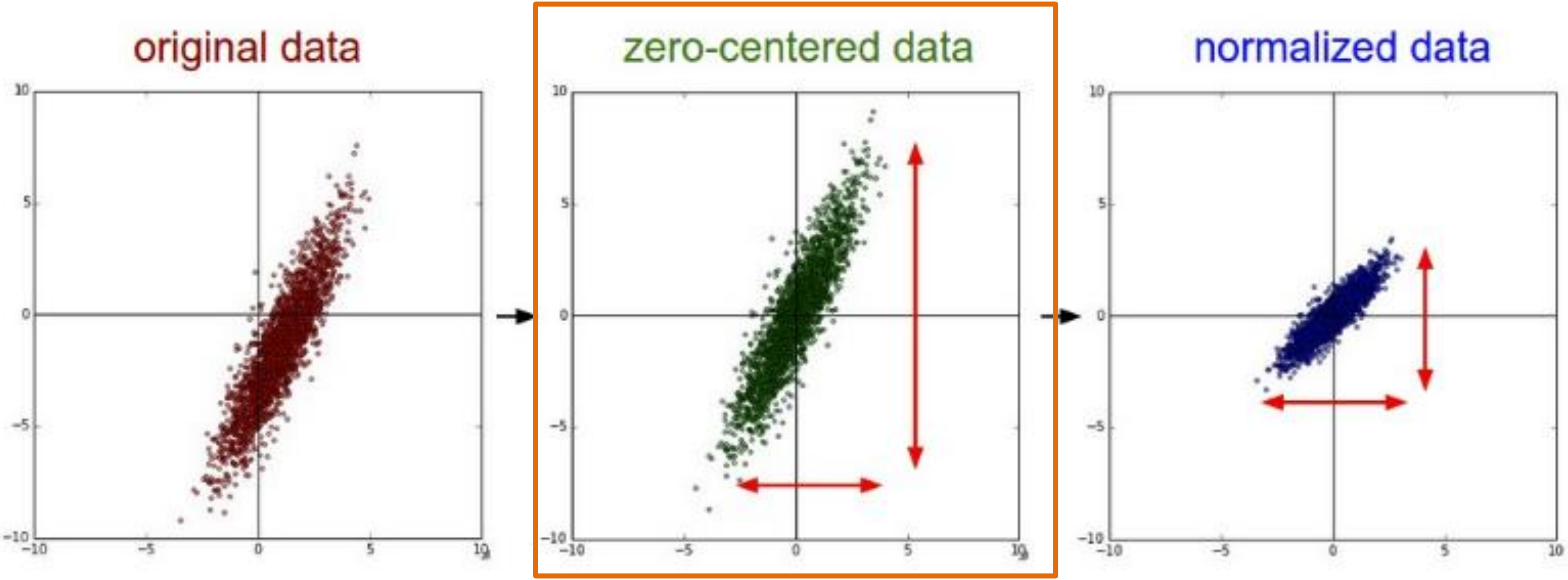
ReLU Kills half of the activations
-> adjust var by a factor of 2

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

Lecture 8

Data Augmentation

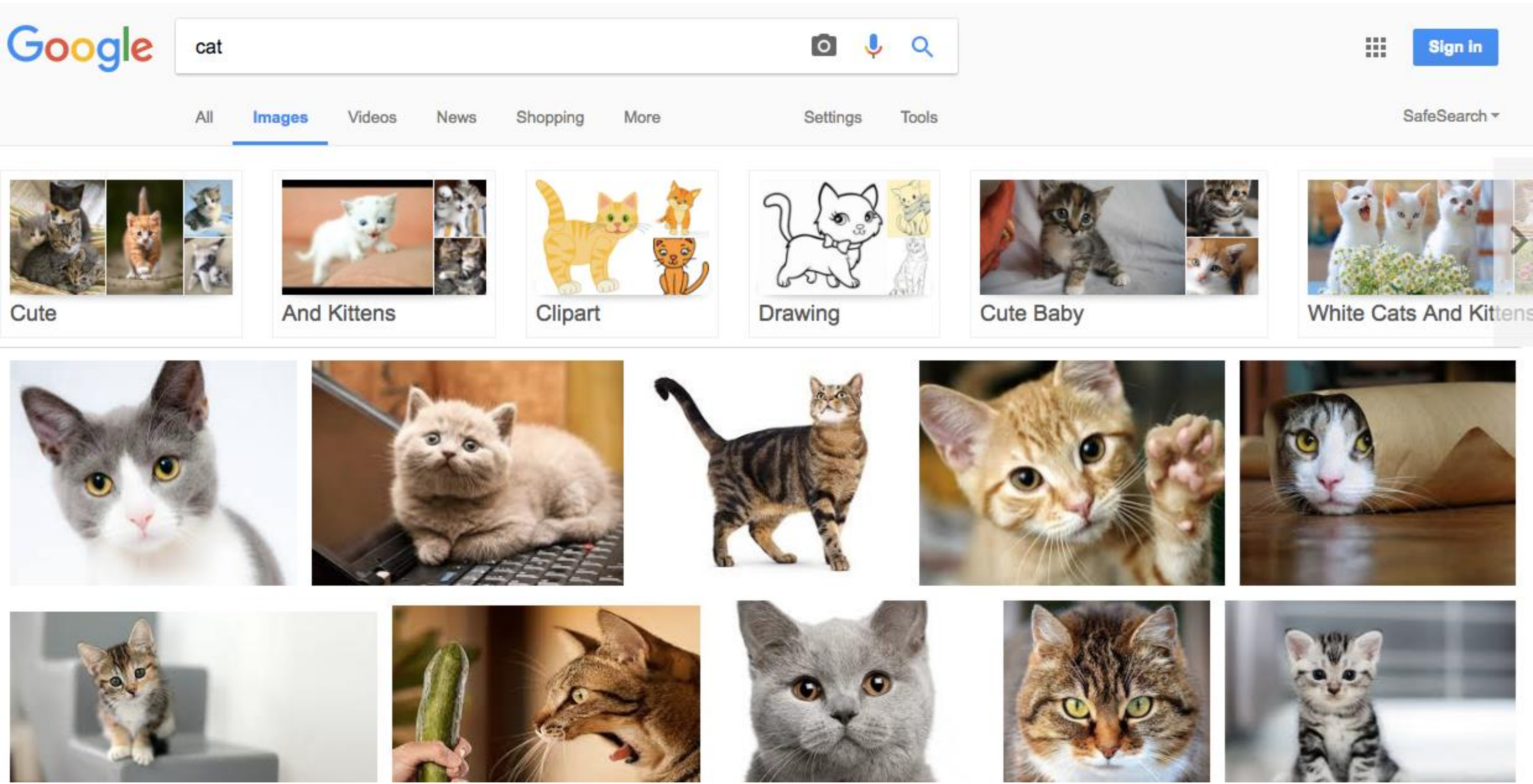
Data Pre-Processing



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

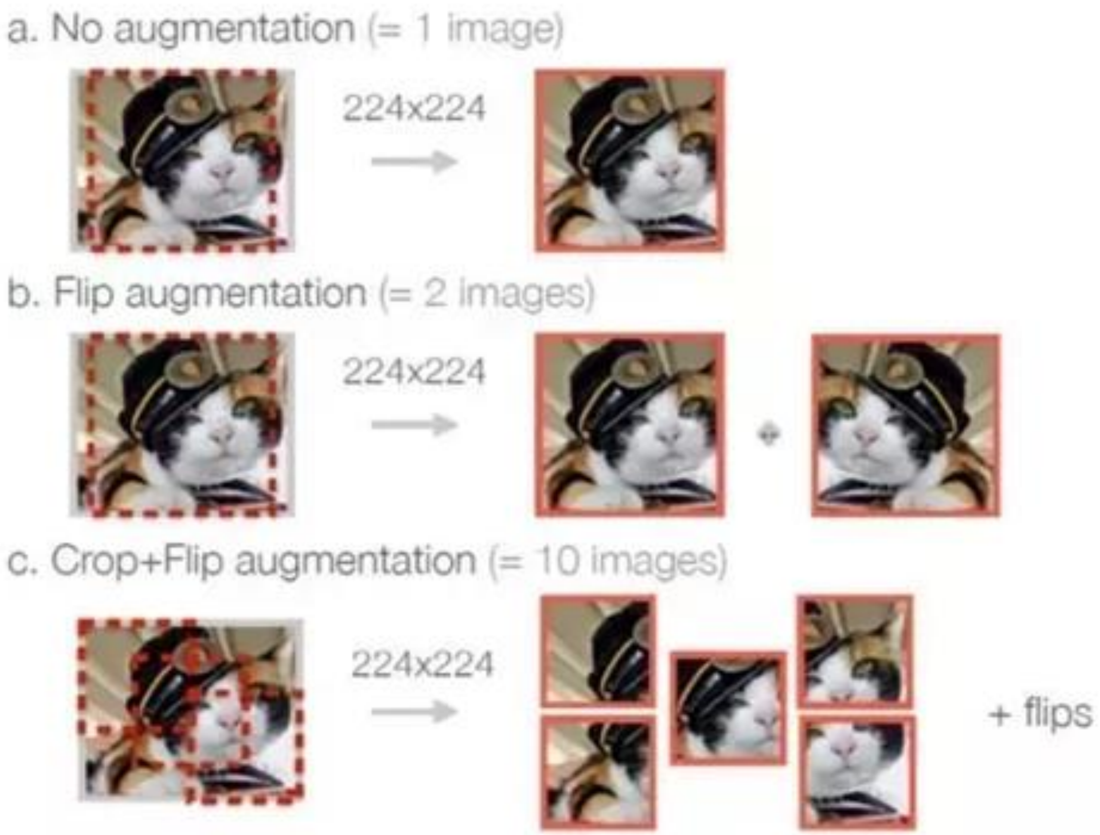


Pose Appearance Illumination

Data Augmentation

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

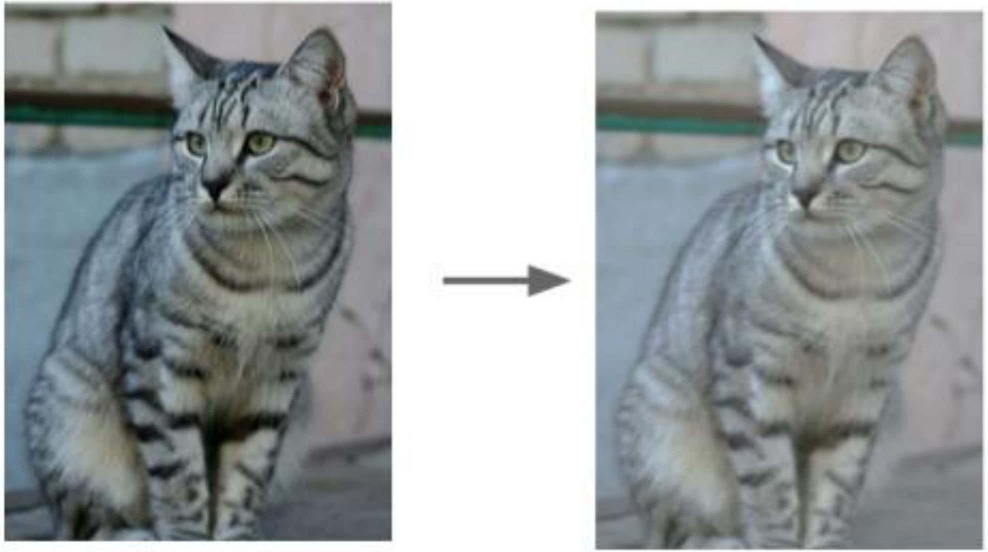
Data Augmentation



[Krizhevsky et al., NIPS'12] ImageNet

Data Augmentation: Brightness

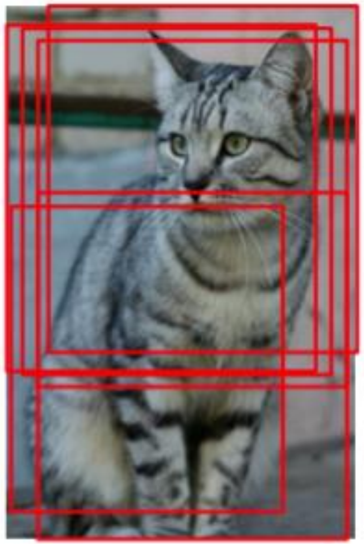
- Random brightness and contrast changes



[Krizhevsky et al., NIPS'12] ImageNet

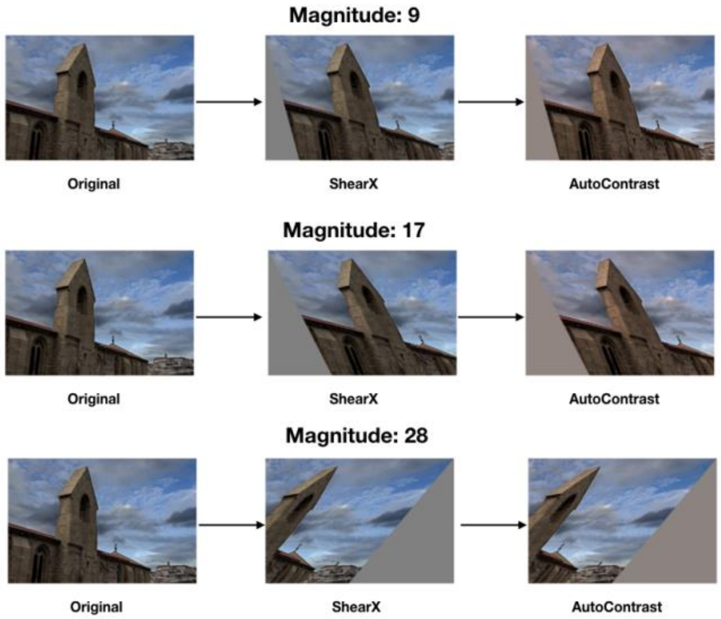
Data Augmentation: Random Crops

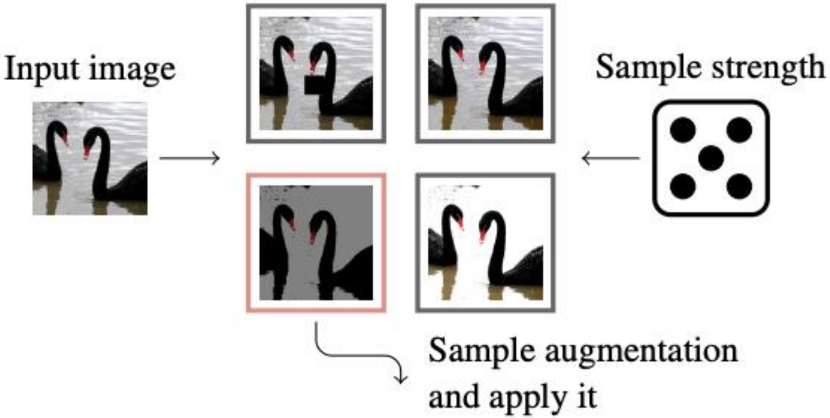
- Training: random crops
  - Pick a random L in [256,480]
  - Resize training image, short side L
  - Randomly sample crops of 224x224
- Testing: fixed set of crops
  - Resize image at N scales
  - 10 fixed crops of 224x224: (4 corners + 1 center) × 2 flips



[Krizhevsky et al., NIPS'12] ImageNet

Data Augmentation: Advanced





Algorithm 1 TrivialAugment Procedure

procedure TA(x: image)
  Sample an augmentation a from $\mathcal{A}$
  Sample a strength m from $\{0, \dots, 30\}$
  Return $a(x, m)$
end procedure

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

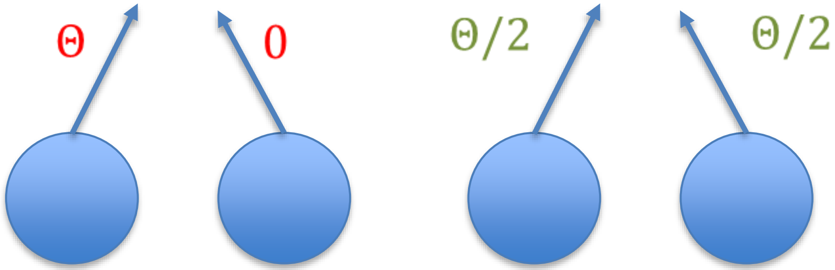
L2 regularization, also (wrongly) called weight decay

- L2 regularization

$$\Theta_{k+1} = \Theta_k - \epsilon \nabla_\Theta f(\Theta_k) - \lambda \Theta_k$$

Learning rate Gradient Gradient of L2-regularization

- Penalizes large weights
- Improves generalization



L2 regularization, also (wrongly) called weight decay

- Weight decay regularization

$$\Theta_{k+1} = (1 - \lambda)\Theta_k - \alpha \nabla_\Theta f(\Theta_k)$$

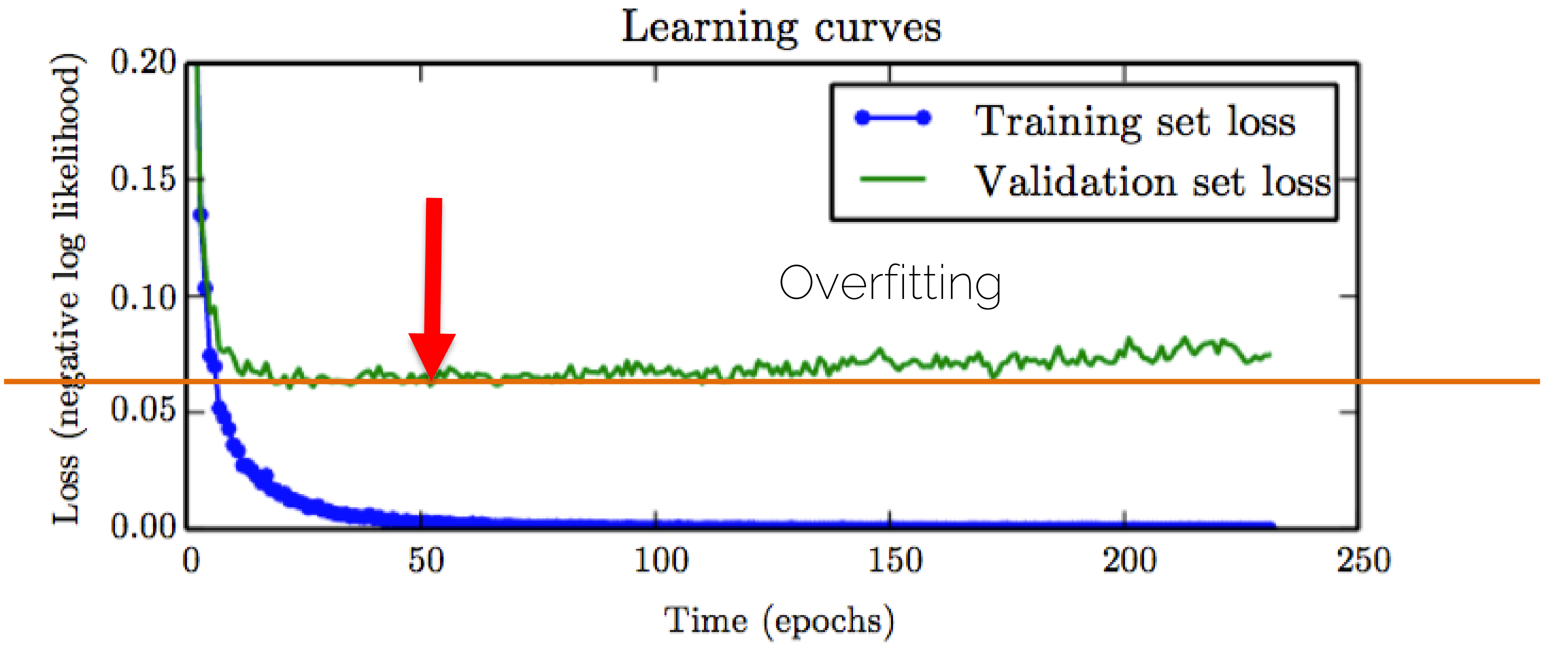
Learning rate of weight decay

Learning rate of the optimizer

- 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

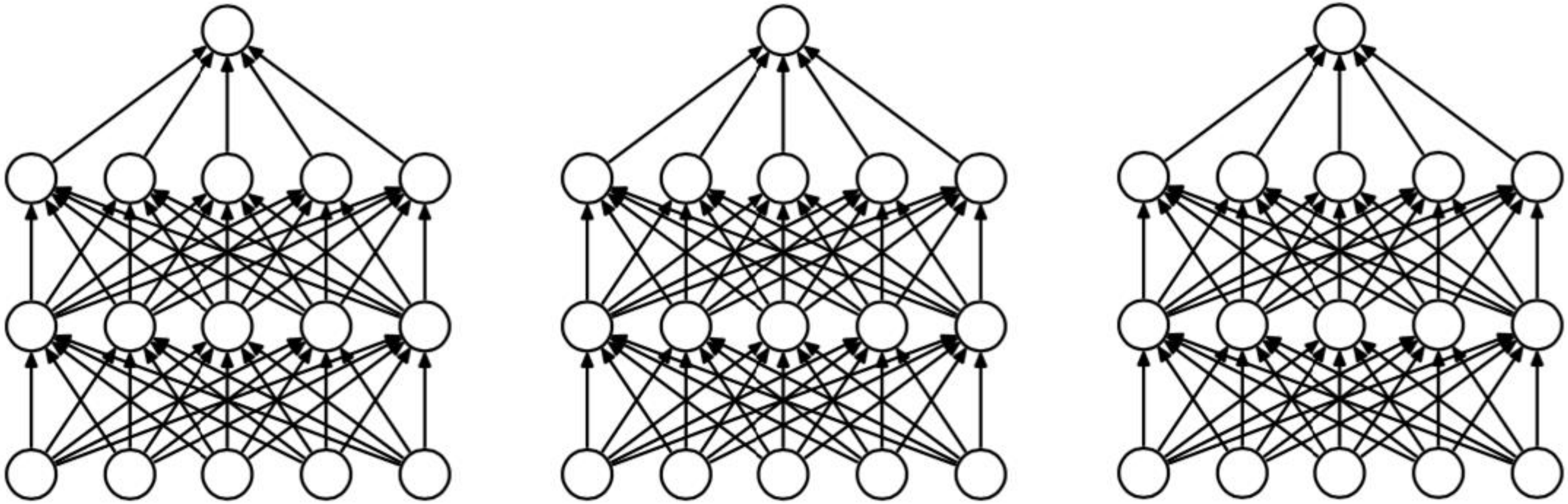
- Train multiple models and average their results

- 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)



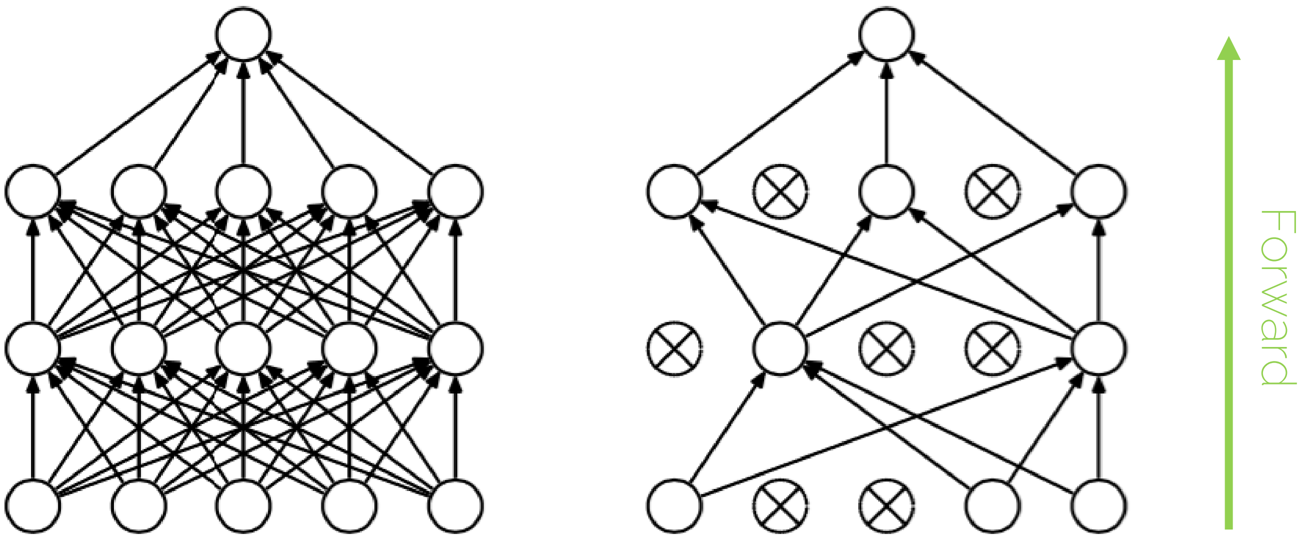
Training Set 1 Training Set 2 Training Set 3

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

Dropout

Dropout

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

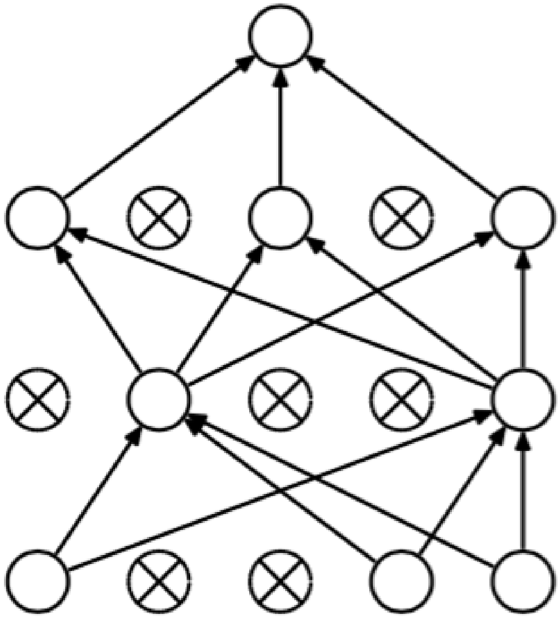


(a) Standard Neural Net (b) After applying dropout.

[Srivastava et al., JMLR'14] 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

[Srivastava et al., JMLR'14] Dropout

Dropout: Intuition

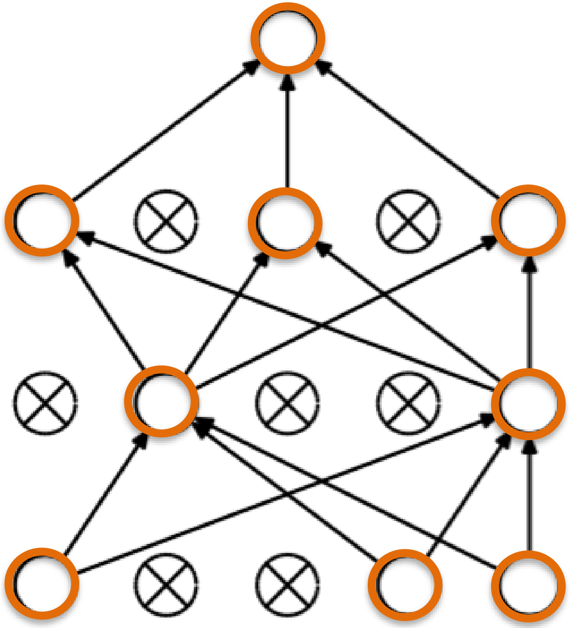
- Using half the network = half capacity
  - Redundant representations
  - Base your scores on more features

- Consider it as a model ensemble

 [Srivastava et al., JMLR'14] Dropout

Dropout: Intuition

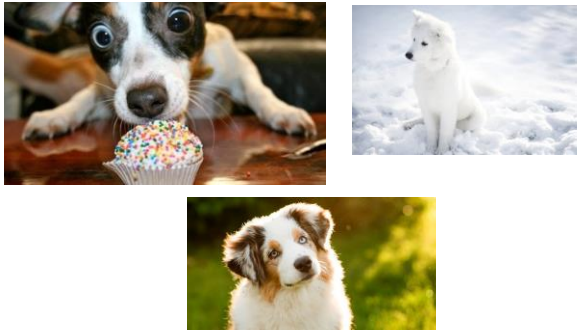
- Two models in one



(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

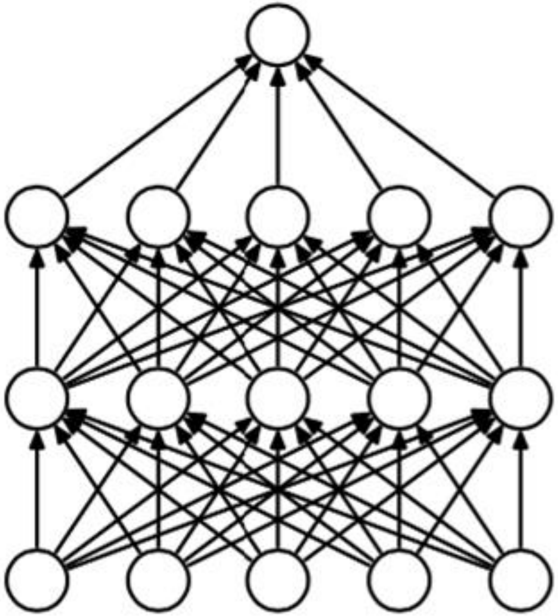
- Consider it as two models in one
  - Training a large ensemble of models, each on different set of data (mini-batch) and with SHARED parameters

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

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

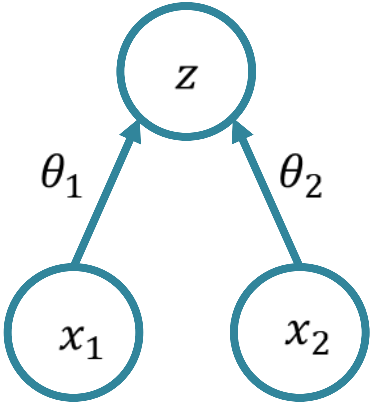
 [Srivastava et al., JMLR'14] Dropout

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

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

[Srivastava et al., JMLR'14] Dropout

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).

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

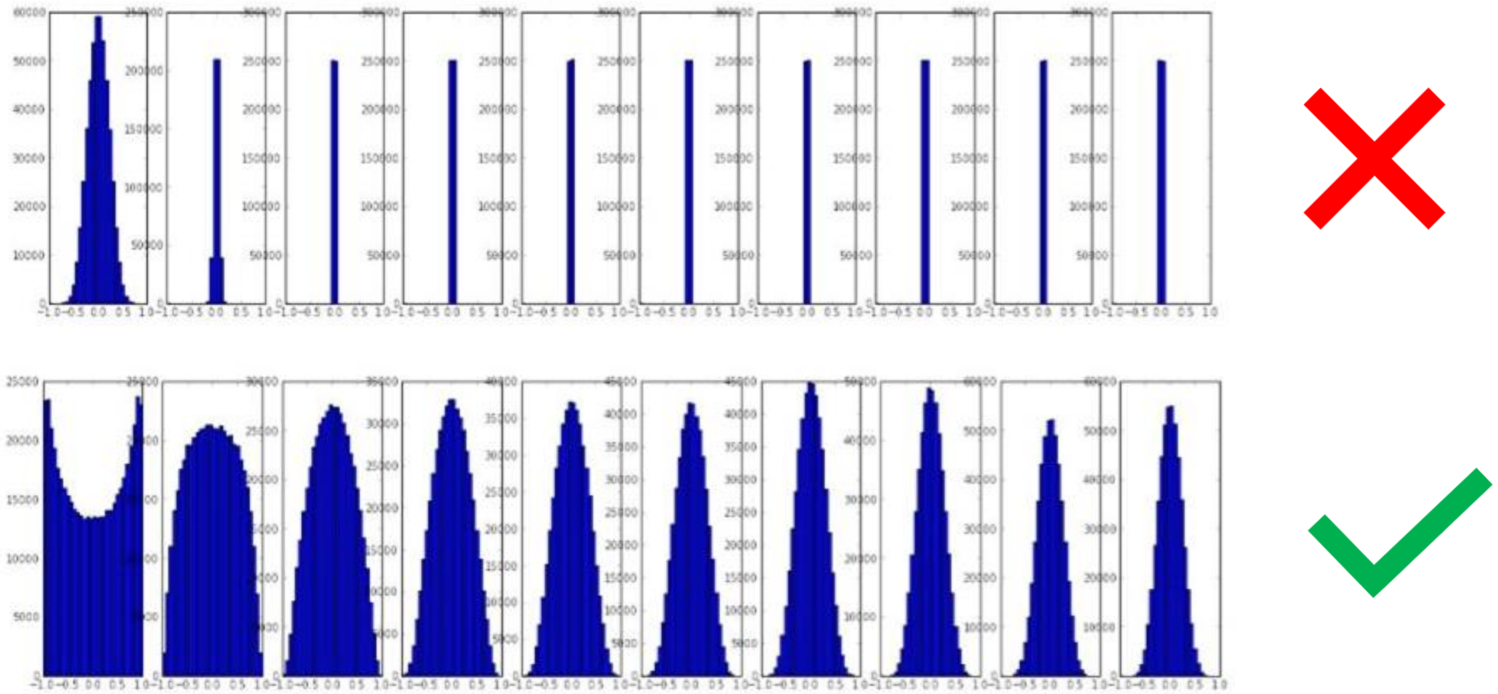
Batch Normalization: Reducing Internal Covariate Shift

Batch Normalization: Reducing Internal Covariate Shift

What is internal covariate shift, by the way?

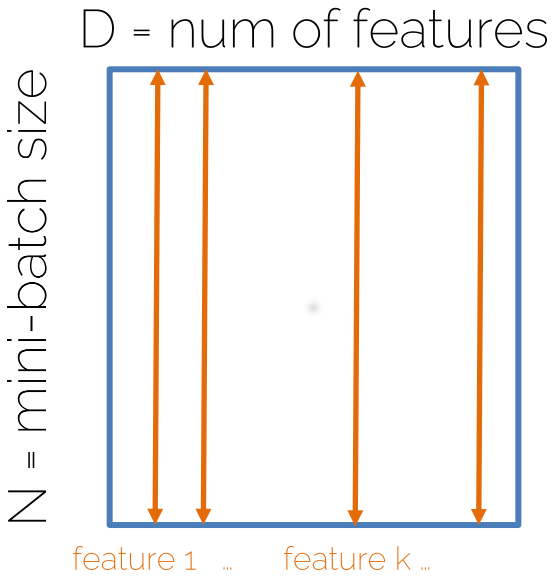
Our Goal

- 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



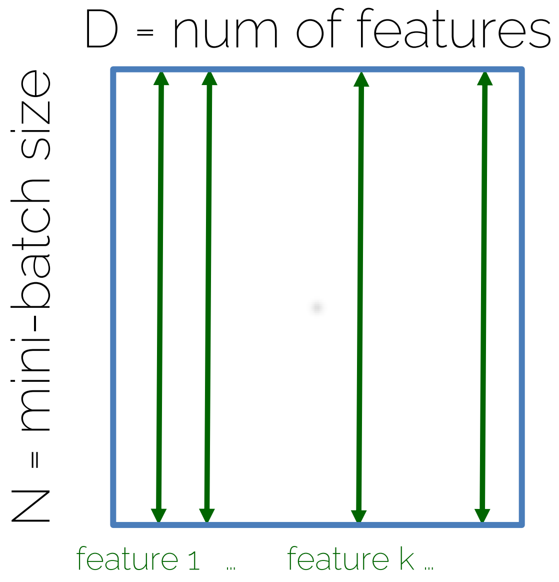
Mean of your mini-batch examples over feature k

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

[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

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

Unit Gaussian

[Ioffe and Szegedy, PMLR'15] Batch Normalization

Batch Normalization

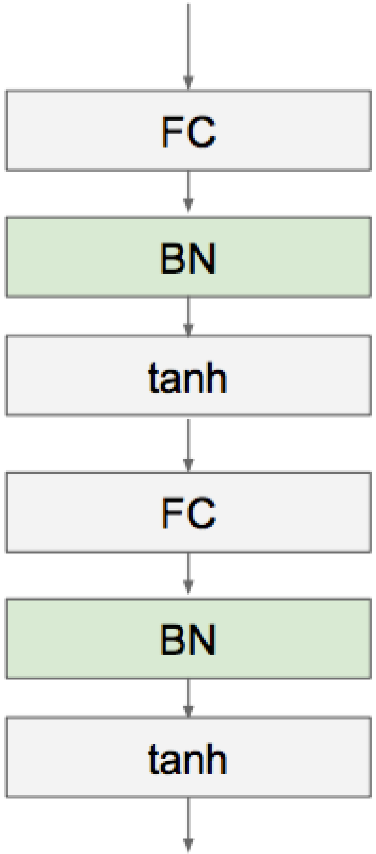
- In each dimension of the features, you have a unit gaussian (in our example)

- 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

[Ioffe and Szegedy, PMLR'15] Batch Normalization

BN: Train vs Test

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

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

- 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_{test}^2 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

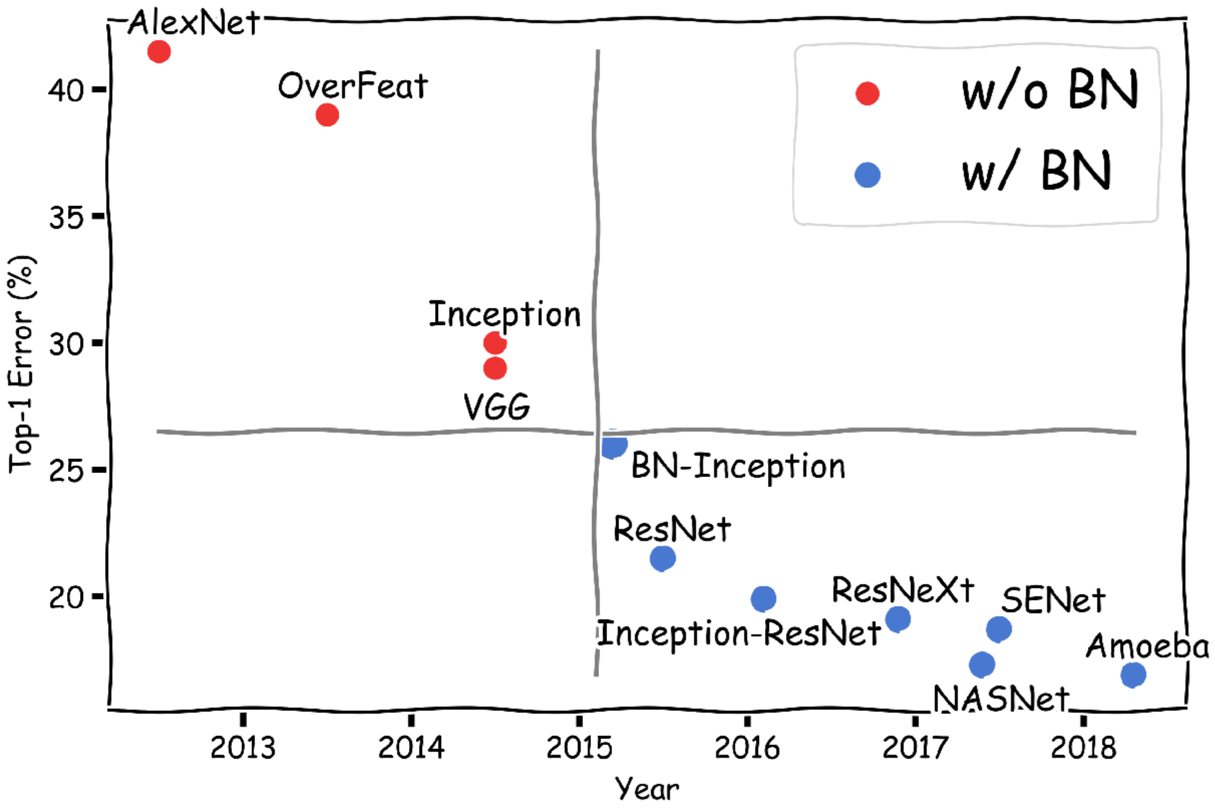
BN: What do you get?

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

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

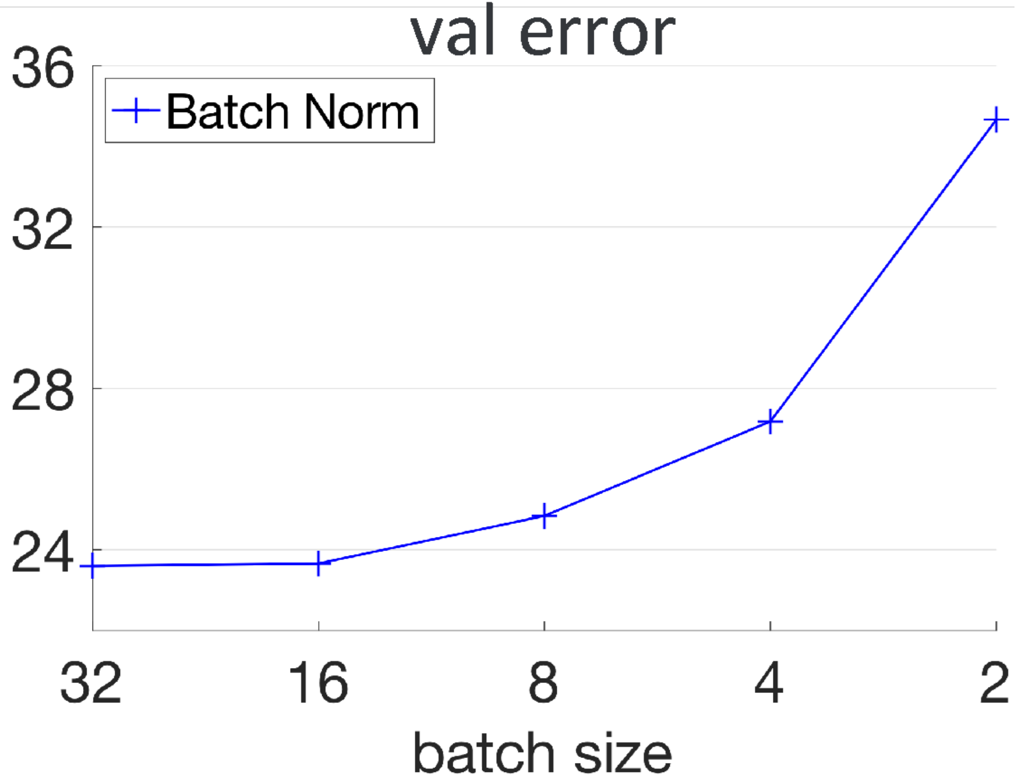
[Ioffe and Szegedy, PMLR'15] Batch Normalization

BN: A Milestone



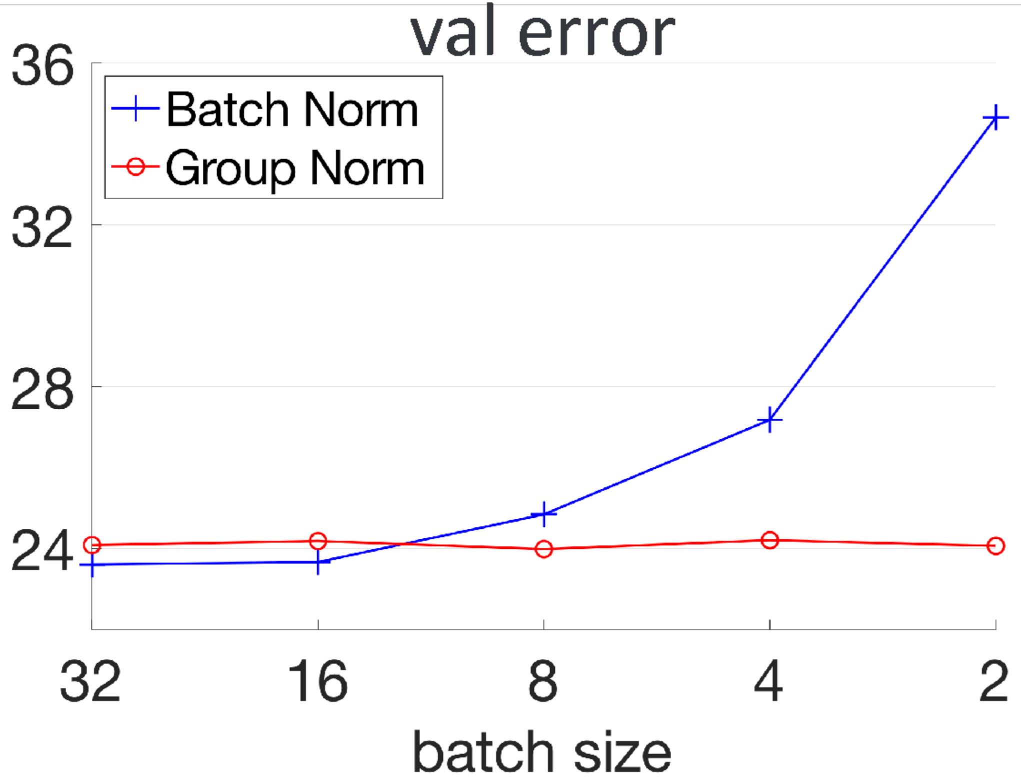
[Wu and He, ECCV'18] Group Normalization

BN: Drawbacks



[Wu and He, ECCV'18] Group Normalization

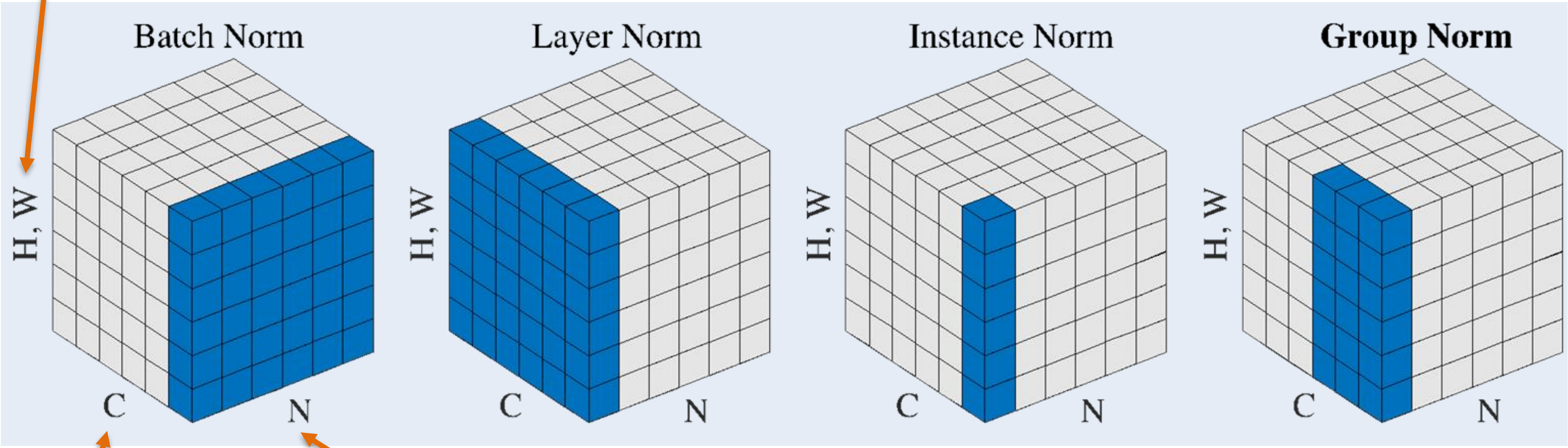
Other Normalizations



[Wu and He, ECCV'18] Group Normalization

Other Normalizations

Image size



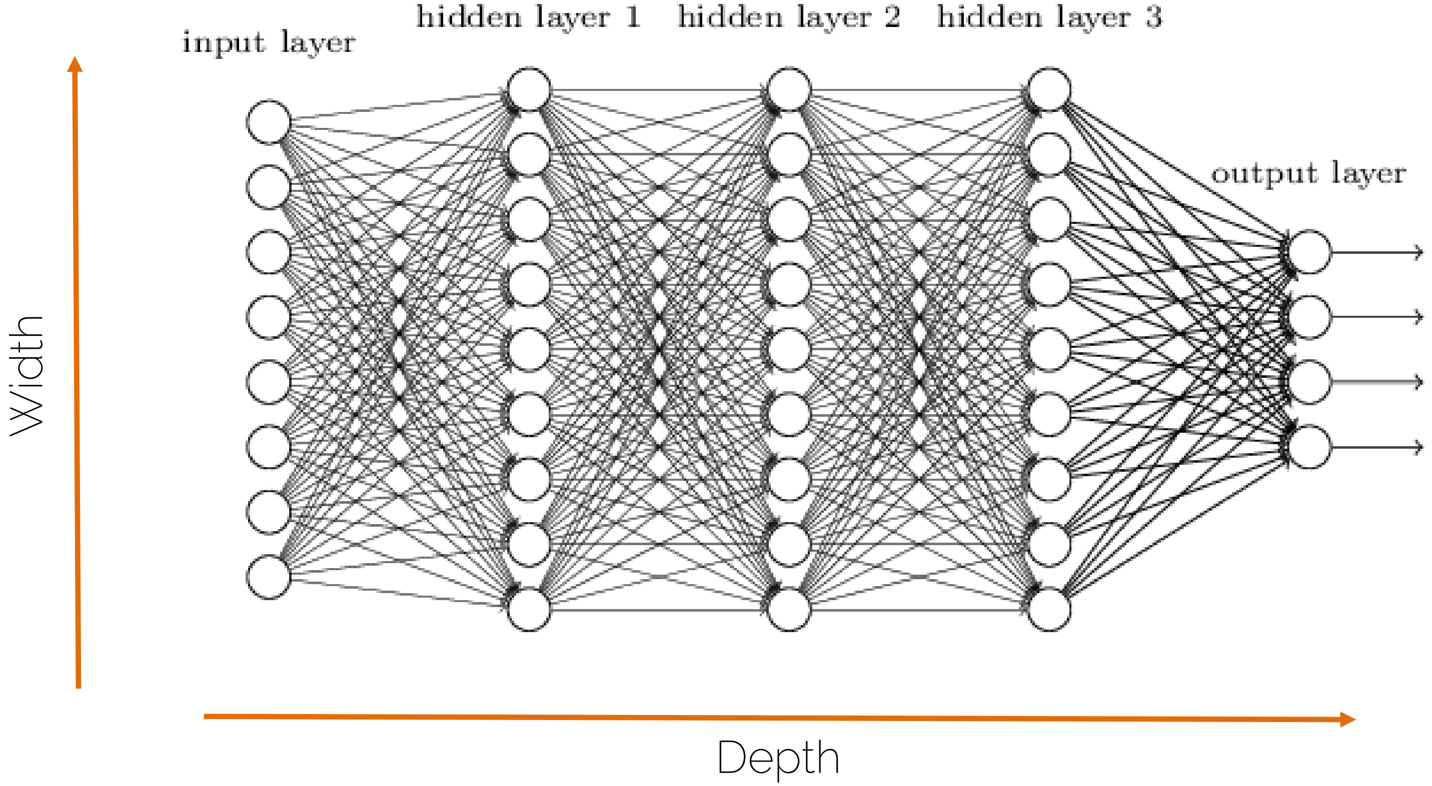
Number of elements in the batch

Number of channels

[Wu and He, ECCV'18] Group Normalization

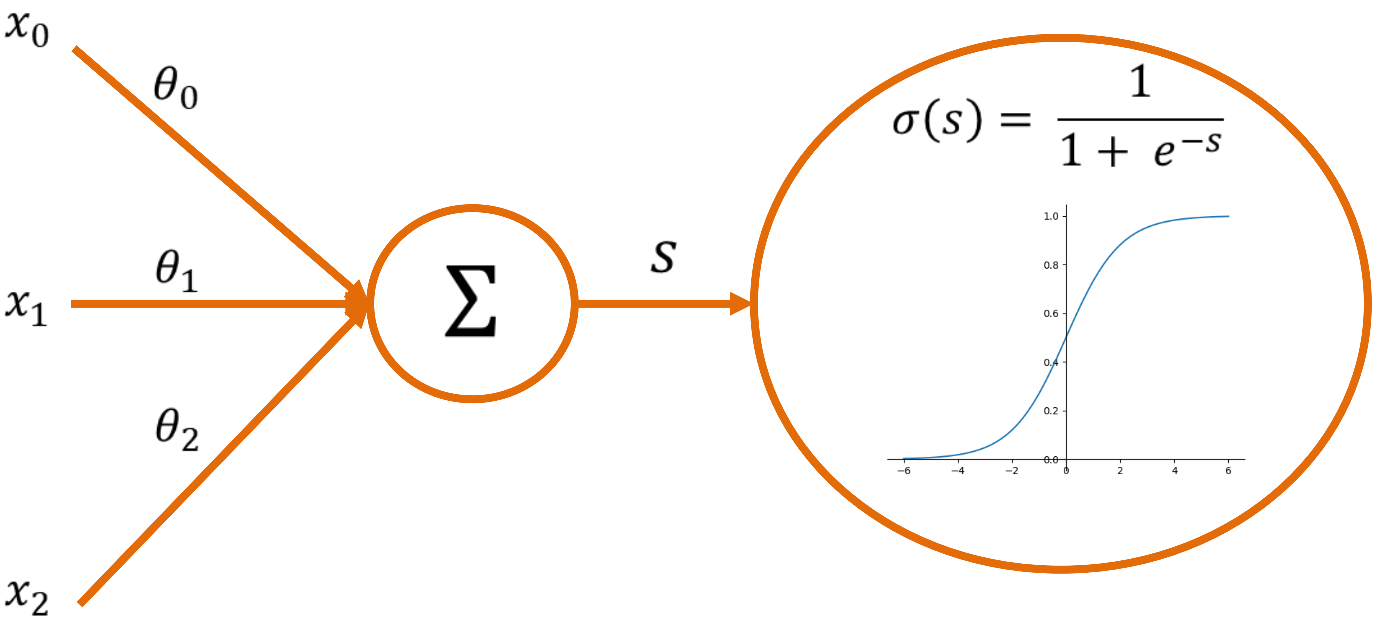
What We Know

What do we know so far?



What do we know so far?

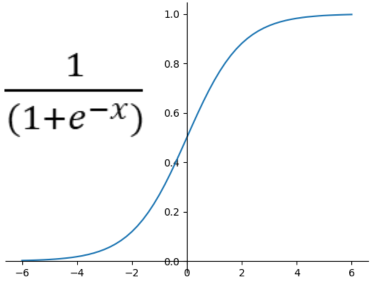
Concept of a 'Neuron'



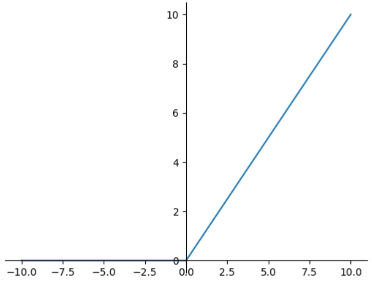
What do we know so far?

Activation Functions (non-linearities)

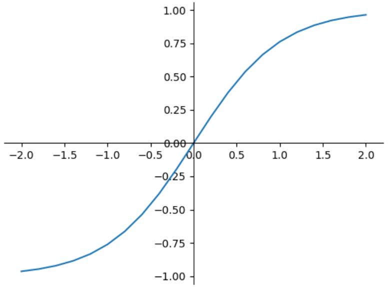
- Sigmoid: $\sigma(x) = \frac{1}{(1+e^{-x})}$



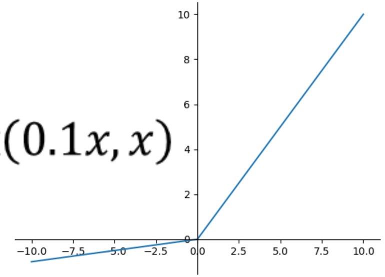
- ReLU: $\max(0, x)$



- TanH: $\tanh(x)$

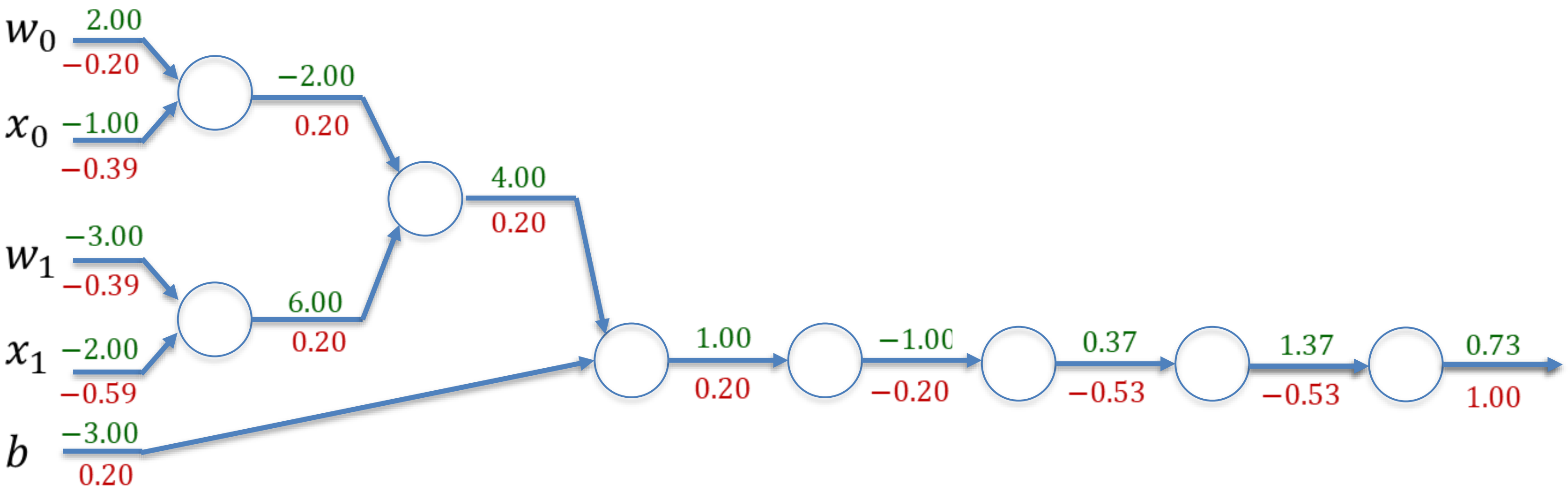


- Leaky ReLU: $\max(0.1x, x)$



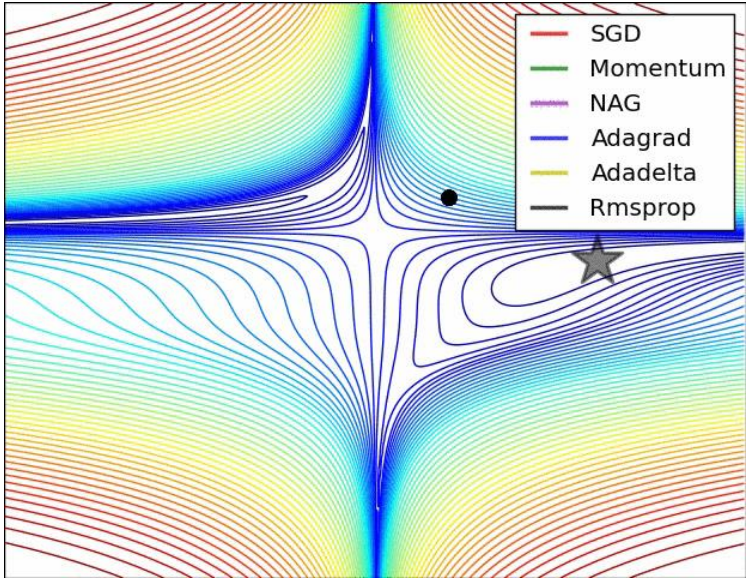
What do we know so far?

Backpropagation



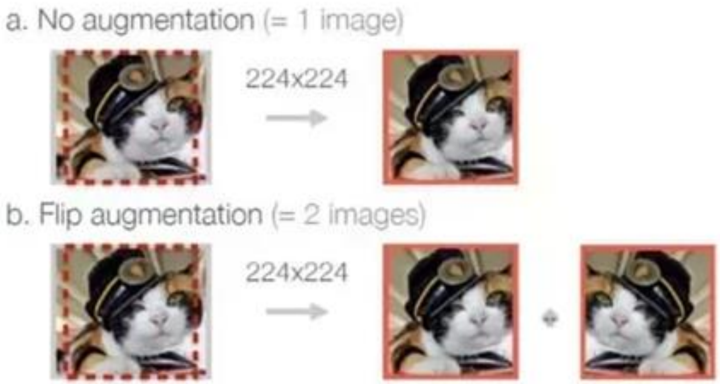
What do we know so far?

SGD Variations (Momentum, etc.)



What do we know so far?

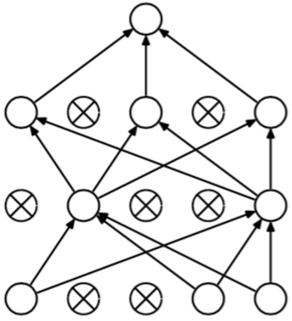
Data Augmentation



Batch-Norm

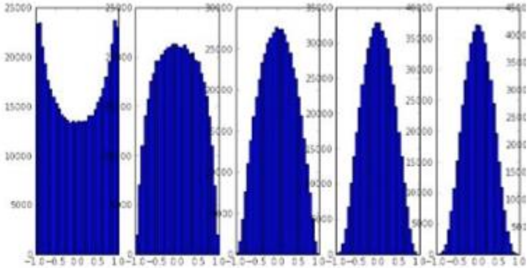
$$\hat{\boldsymbol{x}}^{(k)} = \frac{\boldsymbol{x}^{(k)} - E\left[\boldsymbol{x}^{(k)}\right]}{\sqrt{Var\left[\boldsymbol{x}^{(k)}\right]}}$$

Dropout



(b) After applying dropout.

Weight Initialization
(e.g., Kaiming)



Weight Regularization

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

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/