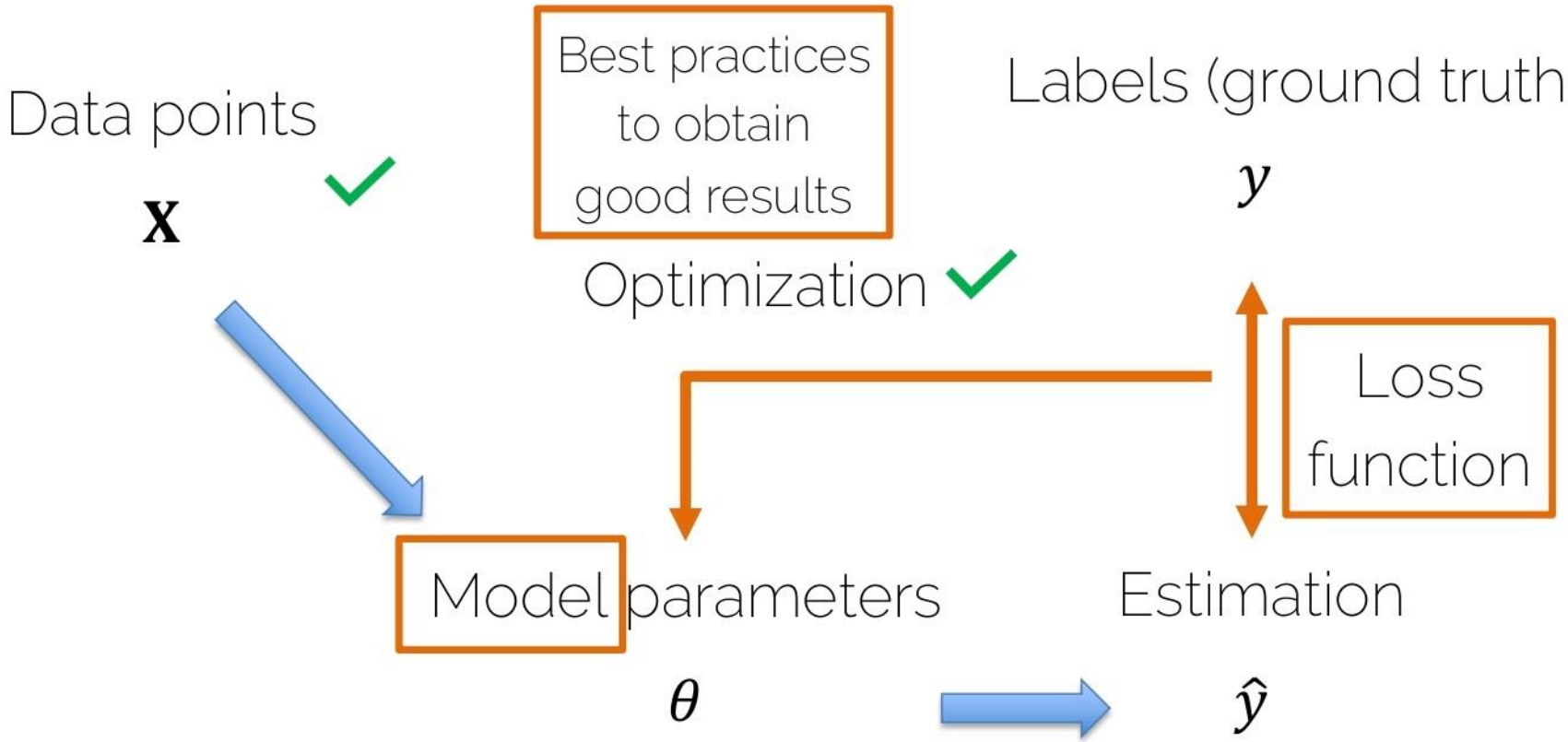
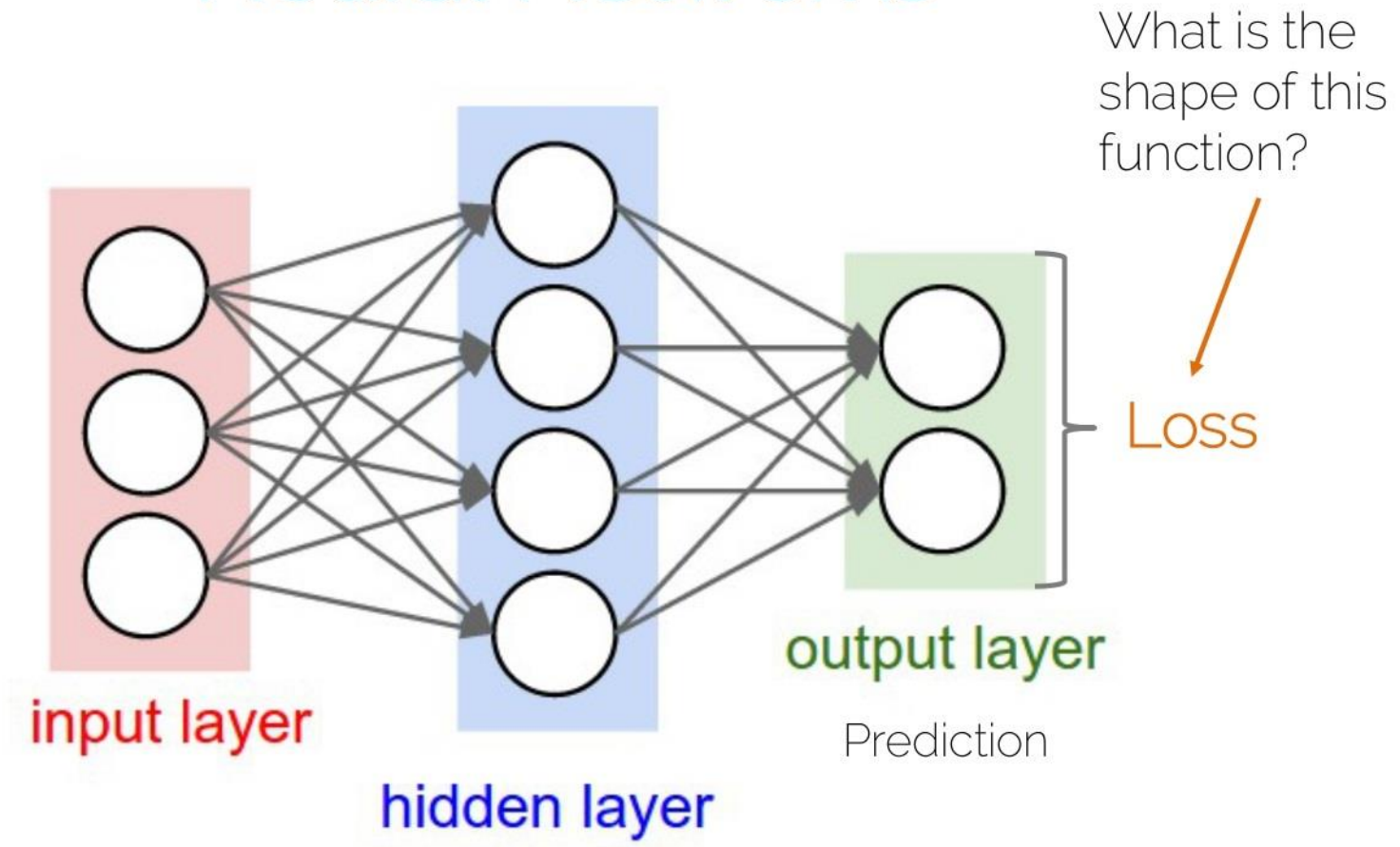
Lecture 8

What we have seen so far



Output and Loss Functions

Neural Networks



Naïve Losses

- L2 Loss: $L^2 = \sum_{i=1}^{n} \left(y_i - f(\boldsymbol{x}_i)\right)^2$
- L1 Loss: $L^1 = \sum_{i=1}^{n} |y_i - f(\boldsymbol{x}_i)|$

training pairs $[\boldsymbol{x}_i; y_i]$ (input and labels)

12	24	42	23
34	32	5	2
12	31	12	31
31	64	5	13

$f(\boldsymbol{x}_i)$

15	20	40	25
34	32	5	2
12	31	12	31
31	64	5	13

y_i

$$L^2(x, y) = 9 + 16 + 4 + 4 + 0 + \cdots + 0 = 33$$

$$L^1(x, y) = 3 + 4 + 2 + 2 + 0 + \cdots + 0 = 11$$

Naïve Losses: L2 vs L1

- L2 Loss:

$$L^2 = \sum_{i=1}^{n} \left(y_i - f(\boldsymbol{x}_i)\right)^2$$

  - Sum of squared differences (SSD)
  - Prone to outliers
  - Compute-efficient optimization
  - Optimum is the mean

- L1 Loss:

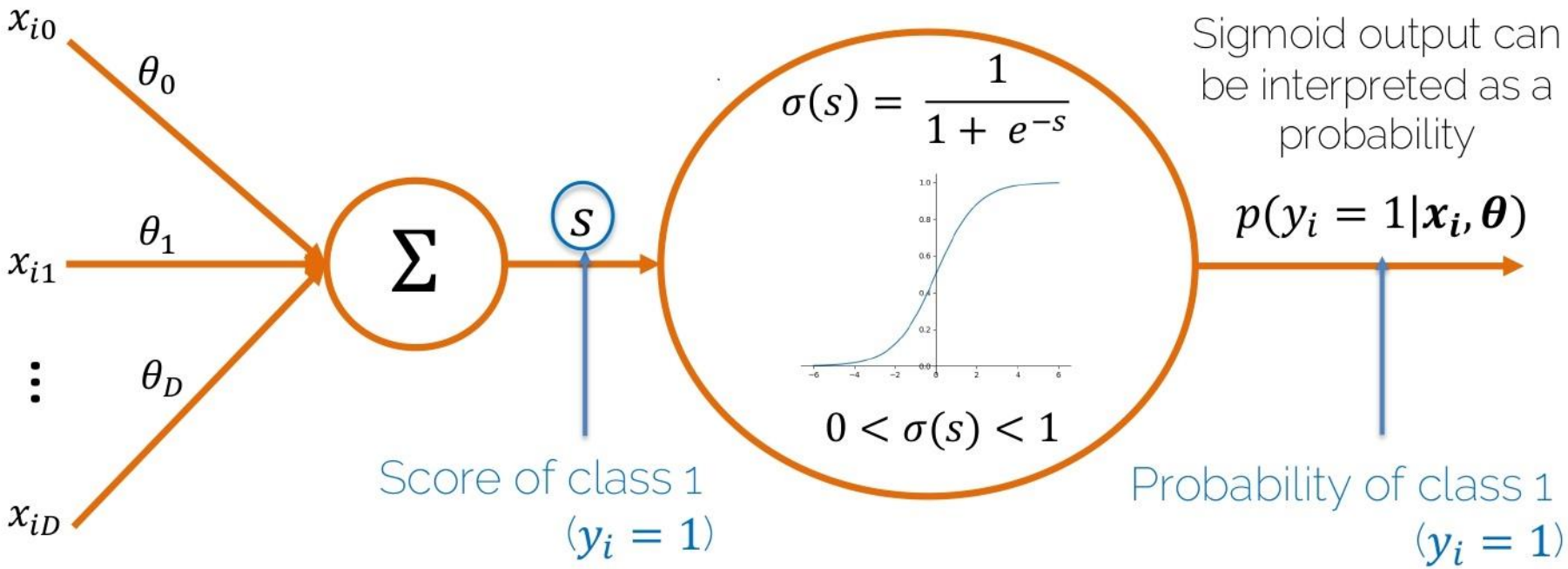
$$L^1 = \sum_{i=1}^{n} |y_i - f(\boldsymbol{x}_i)|$$

  - Sum of absolute differences
  - Robust (cost of outliers is linear)
  - Costly to optimize
  - Optimum is the median

Binary Classification: Sigmoid

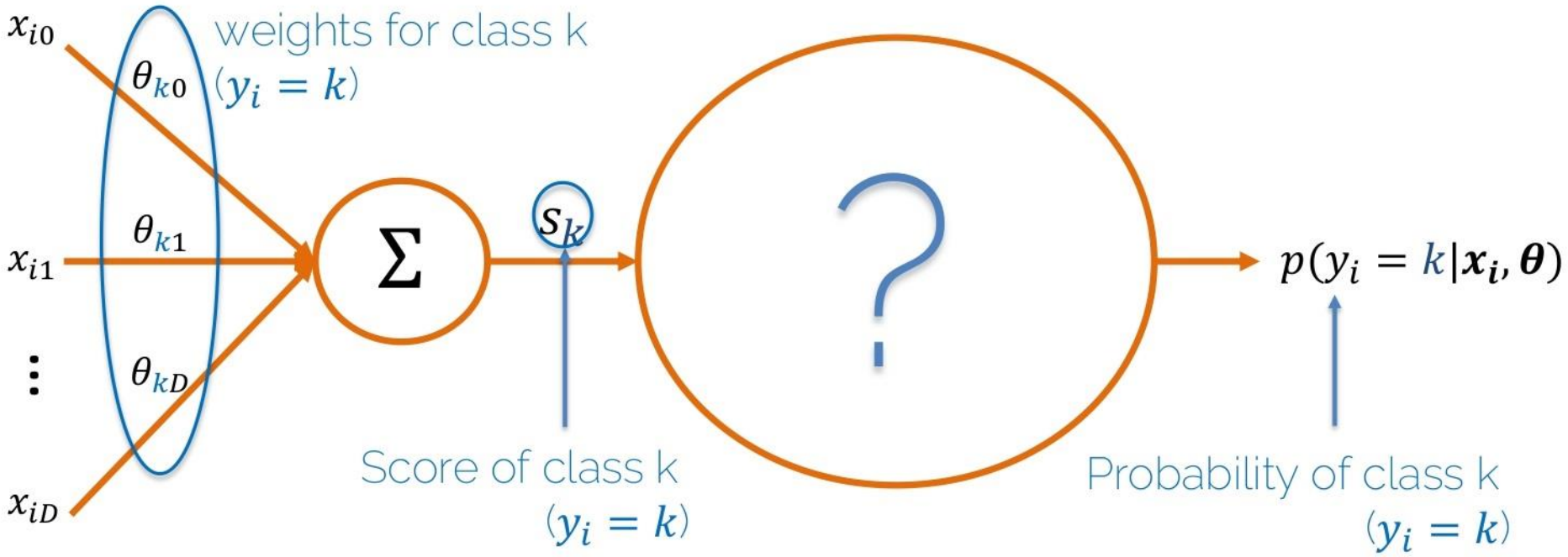
training pairs $[\boldsymbol{x}_i; y_i]$.
$\boldsymbol{x}_i \in \mathbb{R}^D, y_i \in \{1,0\}$ (2 classes)

$$p(y_i = 1|\boldsymbol{x}_i, \boldsymbol{\theta}) = \sigma(s) = \frac{1}{1 + e^{-\sum_{d=0}^{D} \theta_d x_{id}}}$$

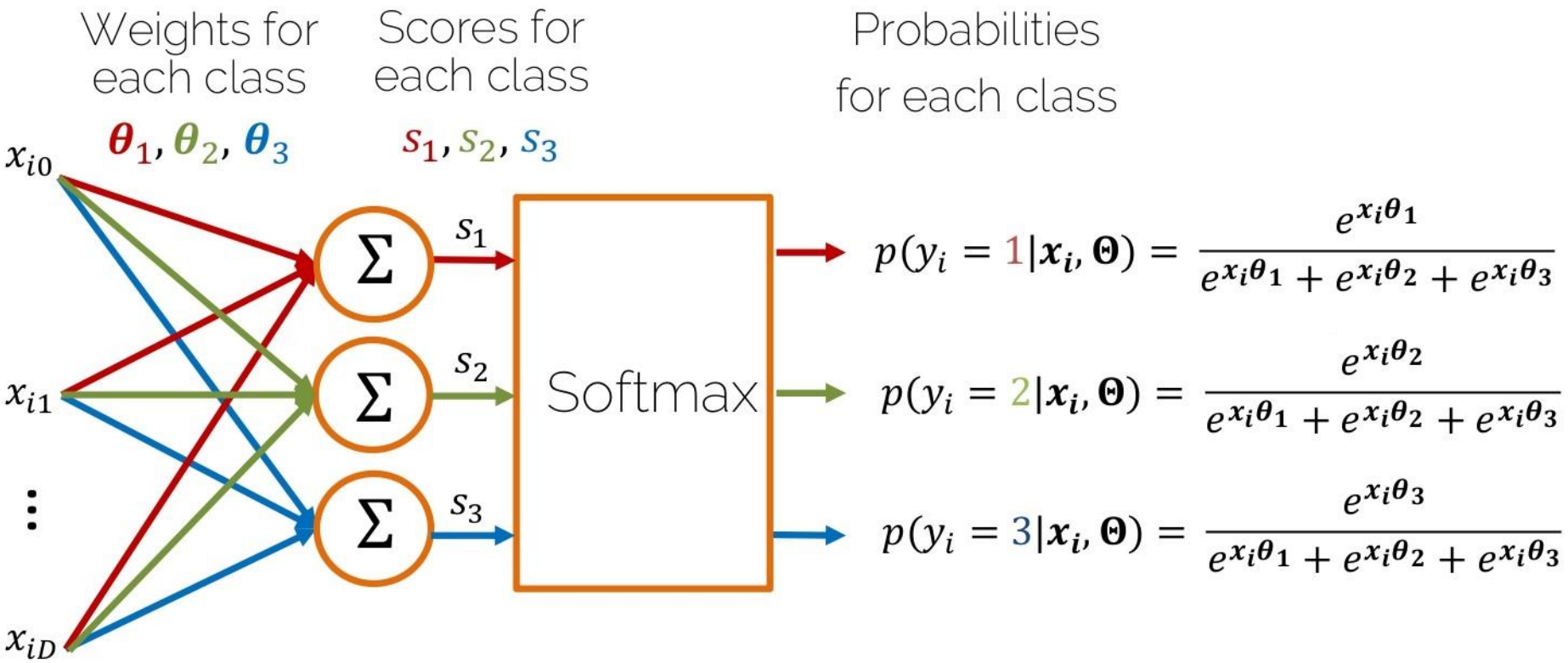


Multiclass Classification: Softmax

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



Multiclass Classification: Softmax



Multiclass Classification: Softmax

- Softmax

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

Exp

normalize

Probability of the true class

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

Parameters:
$\boldsymbol{\Theta} = [\boldsymbol{\theta}_1, \boldsymbol{\theta}_2, \dots, \boldsymbol{\theta}_C]$
$\boldsymbol{C}$: 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.

Multiclass Classification: Softmax

- Numerical Stability

$$p(y_i|\boldsymbol{x_i}, \Theta) = \frac{e^{s_{y_i}}}{\sum_{k=1}^{C} e^{s_k}} = \frac{e^{s_{y_i} - s_{max}}}{\sum_{k=1}^{C} e^{s_k - s_{max}}}$$

Try to prove it by yourself ☺

- Cross-Entropy Loss (Maximum Likelihood Estimate)

$$L_i = -\log(p(y_i|\boldsymbol{x_i}, \Theta)) = -\log(\frac{e^{s_{y_i}}}{\sum_k e^{s_k}})$$

Example: Cross-Entropy Loss

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

Score function $s = f(x_i, \Theta)$

e.g., $f(x_i, \Theta) = [x_{i0}, x_{i2}, \ldots, x_{id}] \cdot [\theta_1, \theta_2, \ldots, \theta_C]$

Suppose: 3 training examples and 3 classes



scores				
	cat	3.2	1.3	2.2
	chair	5.1	4.9	2.5
	car	-1.7	2.0	-3.1

Given a function with weights Θ, training pairs $[x_i; y_i]$ (input and labels)

$\theta_k = \begin{bmatrix} b_k \\ w_k \end{bmatrix}$ parameters for each class with C classes

Example: Cross-Entropy Loss

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

Score function $s = f(x_i, \Theta)$

e.g., $f(x_i, \Theta) = [x_{i0}, x_{i2}, \ldots, x_{id}] \cdot [\theta_1, \theta_2, \ldots, \theta_C]$

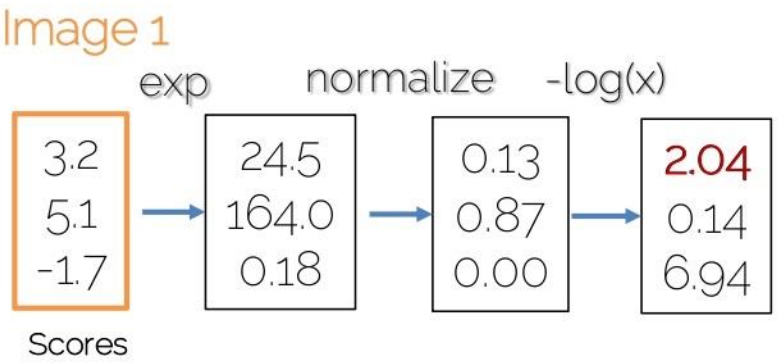
Suppose: 3 training examples and 3 classes



scores				
	cat	3.2	1.3	2.2
	chair	5.1	4.9	2.5
	car	-1.7	2.0	-3.1
	Loss	2.04		

Given a function with weights Θ, training pairs $[x_i; y_i]$ (input and labels)

$\theta_k = \begin{bmatrix} b_k \\ w_k \end{bmatrix}$ parameters for each class with C classes



Example: Cross-Entropy Loss

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

Score function $s = f(x_i, \Theta)$

e.g., $f(x_i, \Theta) = [x_{i0}, x_{i2}, \dots, x_{id}] \cdot [\theta_1, \theta_2, \dots, \theta_C]$

Suppose: 3 training examples and 3 classes



scores				
	cat	3.2	1.3	2.2
	chair	5.1	4.9	2.5
	car	-1.7	2.0	-3.1
Loss		2.04	0.079	6.156

Given a function with weights Θ, training pairs $[x_i; y_i]$ (input and labels)

$\theta_k = \begin{bmatrix} b_k \\ w_k \end{bmatrix}$ parameters for each class with C classes

$$L = \frac{1}{N}\sum_{i=1}^{N} L_i = \frac{L_1 + L_2 + L_3}{3}$$

$$= \frac{2.04 + 0.079 + 6.156}{3} =$$

$$= \mathbf{2.76}$$

Hinge Loss (SVM Loss)

- Score Function $\boldsymbol{s = f(x_i, \Theta)}$
  - e.g., $\boldsymbol{f(x_i, \Theta) = [x_{i0}, x_{i2}, \ldots, x_{id}] \cdot [\theta_1, \theta_2, \ldots, \theta_C]}$

- Hinge Loss (Multiclass SVM Loss)

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

Example: Hinge Loss (SVM Loss)

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

Score function $s = f(x_i, \Theta)$

e.g., $f(x_i, \Theta) = [x_{i0}, x_{i2}, \ldots, x_{id}] \cdot [\theta_1, \theta_2, \ldots, \theta_C]$

Suppose: 3 training examples and 3 classes



scores				
	cat	3.2	1.3	2.2
	chair	5.1	4.9	2.5
	car	-1.7	2.0	-3.1
	Loss			

Given a function with weights Θ, training pairs $[x_i; y_i]$ (input and labels)

$\theta_k = \begin{bmatrix} b_k \\ w_k \end{bmatrix}$ parameters for each class with C classes

Example: Hinge Loss (SVM Loss)

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

Score function $s = f(x_i, \Theta)$

e.g., $f(x_i, \Theta) = [x_{i0}, x_{i2}, \ldots, x_{id}] \cdot [\theta_1, \theta_2, \ldots, \theta_C]$

Suppose: 3 training examples and 3 classes

Given a function with weights Θ, training pairs $[x_i; y_i]$ (input and labels)

$\theta_k = \begin{bmatrix} b_k \\ w_k \end{bmatrix}$ parameters for each class with C classes



scores	cat	3.2	1.3	2.2
	chair	5.1	4.9	2.5
	car	-1.7	2.0	-3.1
	Loss	2.9		

$$L_1 = \max(0, 5.1 - 3.2 + 1) + \max(0, -1.7 - 3.2 + 1)$$
$$= \max(0, 2.9) + \max(0, -3.9)$$
$$= 2.9 + 0$$
$$= \mathbf{2.9}$$

Example: Hinge Loss (SVM Loss)

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

Score function $s = f(x_i, \Theta)$

e.g., $f(x_i, \Theta) = [x_{i0}, x_{i2}, \ldots, x_{id}] \cdot [\theta_1, \theta_2, \ldots, \theta_C]$

Suppose: 3 training examples and 3 classes

Given a function with weights Θ, training pairs $[x_i; y_i]$ (input and labels)

$\theta_k = \begin{bmatrix} b_k \\ w_k \end{bmatrix}$ parameters for each class with C classes



scores				
	cat	3.2	1.3	2.2
	chair	5.1	4.9	2.5
	car	-1.7	2.0	-3.1
	Loss	2.9	0	

$$L_2 = \max(0, 1.3 - 4.9 + 1) + \max(0, 2.0 - 4.9 + 1)$$
$$= \max(0, -2.6) + \max(0, -1.9)$$
$$= 0 + 0 = \mathbf{0}$$

Example: Hinge Loss (SVM Loss)

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

Score function $\boldsymbol{s} = \boldsymbol{f}(\boldsymbol{x_i}, \boldsymbol{\Theta})$

e.g., $\boldsymbol{f}(\boldsymbol{x_i}, \boldsymbol{\Theta}) = [x_{i0}, x_{i2}, \ldots, x_{id}] \cdot [\boldsymbol{\theta_1}, \boldsymbol{\theta_2}, \ldots, \boldsymbol{\theta_C}]$

Suppose: 3 training examples and 3 classes

Given a function with weights $\boldsymbol{\Theta}$, training pairs $[\boldsymbol{x_i}; y_i]$ (input and labels)

$\boldsymbol{\theta_k} = \begin{bmatrix} b_k \\ \boldsymbol{w_k} \end{bmatrix}$ parameters for each class with $\boldsymbol{C}$ classes



scores				
	cat	3.2	1.3	2.2
	chair	5.1	4.9	2.5
	car	-1.7	2.0	-3.1
	Loss	2.9	0	12.9

$$L_3 = \max(0, 2.2 - (-3.1) + 1) + \max(0, 2.5 - (-3.1) + 1)$$
$$= \max(0, 6.3) + \max(0, 6.6)$$
$$= 6.3 + 6.6$$
$$= \mathbf{12.9}$$

Example: Hinge Loss (SVM Loss)

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

Score function $\boldsymbol{s = f(x_i, \Theta)}$

e.g., $\boldsymbol{f(x_i, \Theta)} = [x_{i0}, x_{i2}, \ldots, x_{id}] \cdot [\boldsymbol{\theta_1}, \boldsymbol{\theta_2}, \ldots, \boldsymbol{\theta_C}]$

Suppose: 3 training examples and 3 classes

Given a function with weights $\boldsymbol{\Theta}$, training pairs $[\boldsymbol{x_i}; y_i]$ (input and labels)

$\boldsymbol{\theta_k} = \begin{bmatrix} b_k \\ \boldsymbol{w_k} \end{bmatrix}$ parameters for each class with $\boldsymbol{C}$ classes



scores	cat	3.2	1.3	2.2
	chair	5.1	4.9	2.5
	car	-1.7	2.0	-3.1
Loss		2.9	0	12.9

$$L = \frac{1}{N}\sum_{i=1}^{N} L_i = \frac{L_1 + L_2 + L_3}{3}$$

$$= \frac{2.9 + 0 + 12.9}{3}$$

$$= \mathbf{5.3}$$

Multiclass Classification: Hinge vs Cross-Entropy

- Hinge Loss: $L_i = \sum_{k \neq y_i} \max(0, s_k - s_{y_i} + 1)$
- Cross Entropy Loss: $L_i = -\log(\frac{e^{s_{y_i}}}{\sum_k e^{s_k}})$

Example: Hinge vs Cross-Entropy

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

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

For image $\boldsymbol{x_i}$ (assume $\boldsymbol{y_i = 0}$):

	Scores	Hinge loss:	Cross Entropy loss:
Model 1	$s = [5, -3, 2]$		
Model 2	$s = [5, 10, 10]$		
Model 3	$s = [5, -20, -20]$		

Example: Hinge vs Cross-Entropy

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

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

For image $\boldsymbol{x_i}$ (assume $\boldsymbol{y_i = 0}$):

	Scores	Hinge loss:	Cross Entropy loss:
Model 1	$s = [5, -3, 2]$	$\max(0, -3 - 5 + 1) + \max(0, 2 - 5 + 1) = 0$	
Model 2	$s = [5, 10, 10]$	$\max(0, 10 - 5 + 1) + \max(0, 10 - 5 + 1) = 12$	
Model 3	$s = [5, -20, -20]$	$\max(0, -20 - 5 + 1) + \max(0, -20 - 5 + 1) = 0$	

Apparently Model 3 is better, but losses show no difference between Model 1&3, since they all have same loss=0.

Example: Hinge vs Cross-Entropy

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

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

For image $\boldsymbol{x_i}$ (assume $\boldsymbol{y_i = 0}$):

	Scores	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	$s = [5, -20, -20]$	$\max(0, -20 - 5 + 1) + \max(0, -20 - 5 + 1) = 0$	$-\ln\left(\frac{e^5}{e^5 + e^{-20} + e^{-20}}\right) = 2 * 10^{-11}$

Model 3 has a clearly smaller loss now.

Example: Hinge vs Cross-Entropy

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

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

For image $\boldsymbol{x_i}$ (assume $\boldsymbol{y_i = 0}$):

	Scores	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	$s = [5, -20, -20]$	$\max(0, -20 - 5 + 1) + \max(0, -20 - 5 + 1) = 0$	$-\ln\left(\frac{e^5}{e^5 + e^{-20} + e^{-20}}\right) = 2 * 10^{-11}$

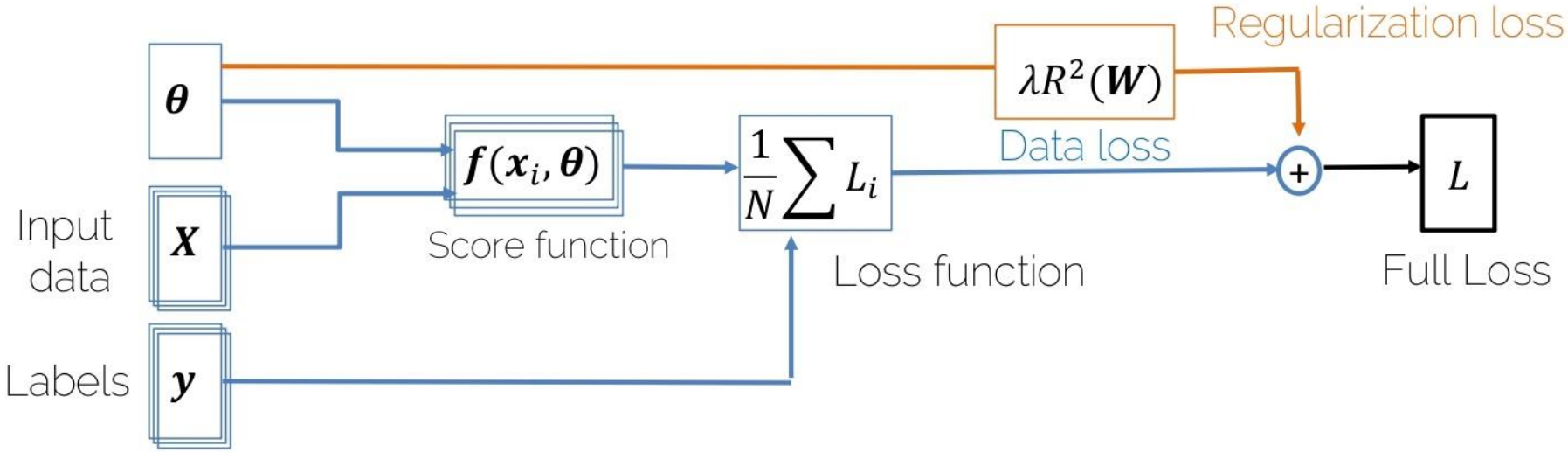
- Cross Entropy *always* wants to improve! (loss never 0)
- Hinge Loss saturates.

Loss in Compute Graph

- How do we combine loss functions with weight regularization?

- How to optimize parameters of our networks according to multiple losses?

Loss in Compute Graph



Want to find optimal $\boldsymbol{\theta}$. (weights are unknowns of optimization problem)

- Compute gradient w.r.t. $\boldsymbol{\theta}$.
- Gradient $\nabla_{\boldsymbol{\theta}} L$ is computed via backpropagation

Loss in Compute Graph

Given a function with weights $\boldsymbol{\theta}$,
Training pairs $[\boldsymbol{x_i}; y_i]$ (input and labels)

- Score function $\boldsymbol{s = f(x_i, \theta)}$
- Data Loss - Cross Entropy $L_i = -\log(\frac{e^{s_{y_i}}}{\sum_k e^{s_k}})$
  - SVM $L_i = \sum_{k \neq y_i} \max(0, s_k - s_{y_i} + 1)$
- Regularization Loss: e.g., $L2$-Reg: $R^2(\boldsymbol{W}) = \sum \boldsymbol{w_i^2}$
- Full Loss $L = \frac{1}{N} \sum_{i=1}^{N} L_i + \lambda R^2(\boldsymbol{W})$
- Full Loss = Data Loss + Reg Loss

Example: Regularization & SVM Loss

Multiclass SVM loss $L_i = \sum_{k \neq y_i} \max(0, \boldsymbol{f}(\boldsymbol{x}_i; \boldsymbol{\theta})_k - \boldsymbol{f}(\boldsymbol{x}_i; \boldsymbol{\theta})_{y_i} + 1)$

Full loss $L = \frac{1}{N} \sum_{i=1}^{N} \sum_{k \neq y_i} \max(0, \boldsymbol{f}(\boldsymbol{x}_i; \boldsymbol{\theta})_k - \boldsymbol{f}(\boldsymbol{x}_i; \boldsymbol{\theta})_{y_i} + 1) + \lambda R(\boldsymbol{W})$

$L1$-Reg: $R^1(\boldsymbol{W}) = \sum_{i=1}^{D} |\boldsymbol{w}_i|$

$L2$-Reg: $R^2(\boldsymbol{W}) = \sum_{i=1}^{D} \boldsymbol{w}_i^2$

Example:

$\boldsymbol{x} = [1,1,1,1]^T$

$\boldsymbol{w}_1 = [1,0,0,0]^T$

$\boldsymbol{w}_2 = [0.25, 0.25, 0.25, 0.25]^T$

$\boldsymbol{x}^T \boldsymbol{w}_1 = \boldsymbol{x}^T \boldsymbol{w}_2 = 1$

$R^2(\boldsymbol{w}_1) = 1$

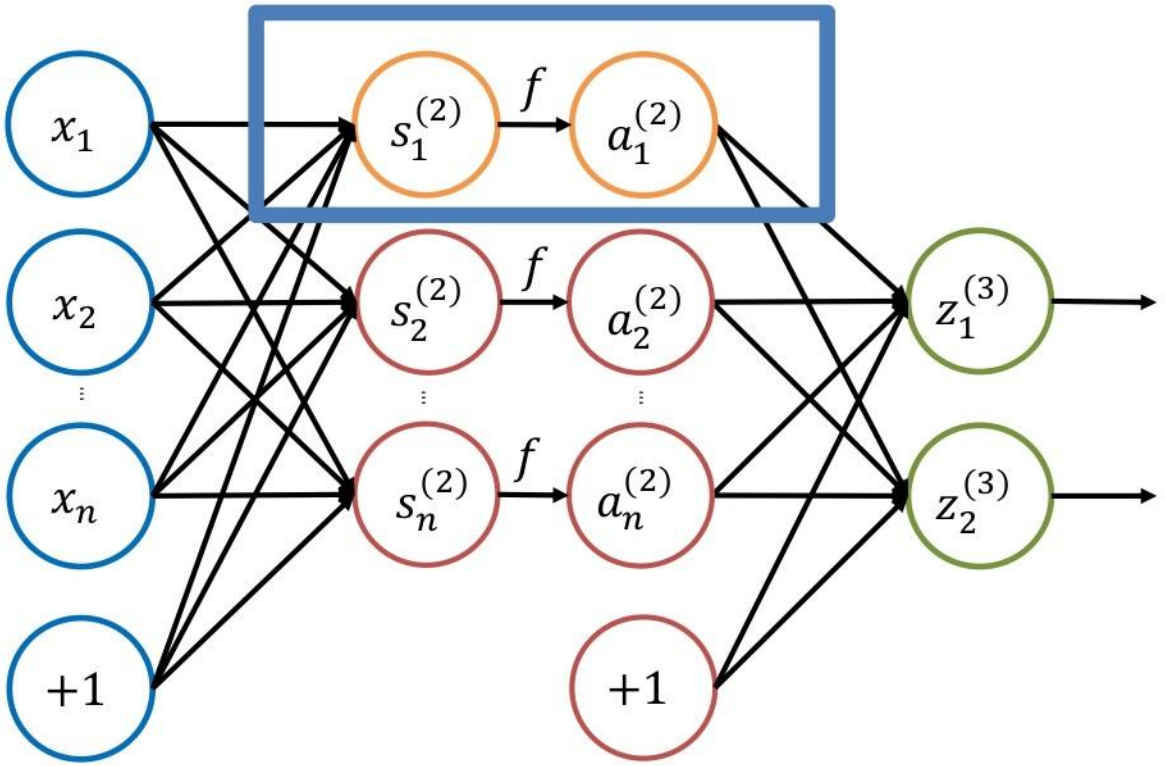
$R^2(\boldsymbol{w}_2) = 0.25^2 + 0.25^2 + 0.25^2 + 0.25^2$

$= 0.25$

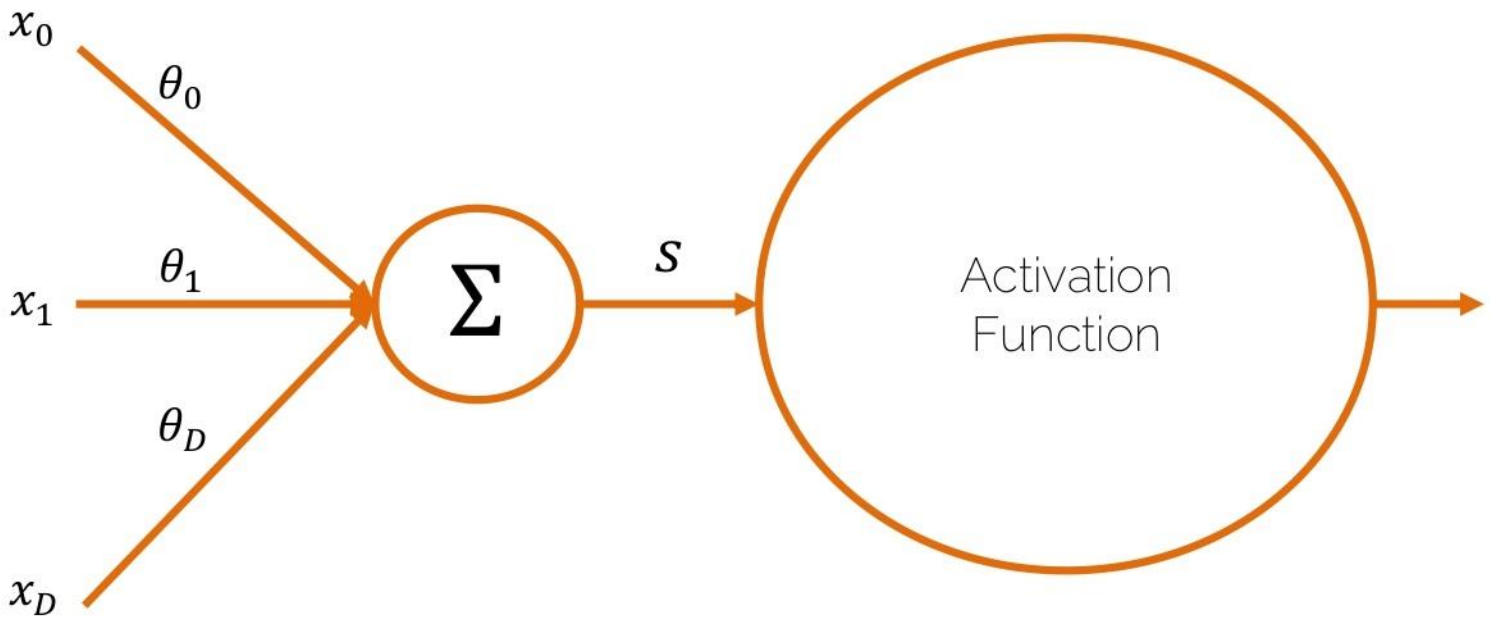
$R^2(\boldsymbol{W}) = 1 + 0.25 = 1.25$

Activation Functions

Neural Networks

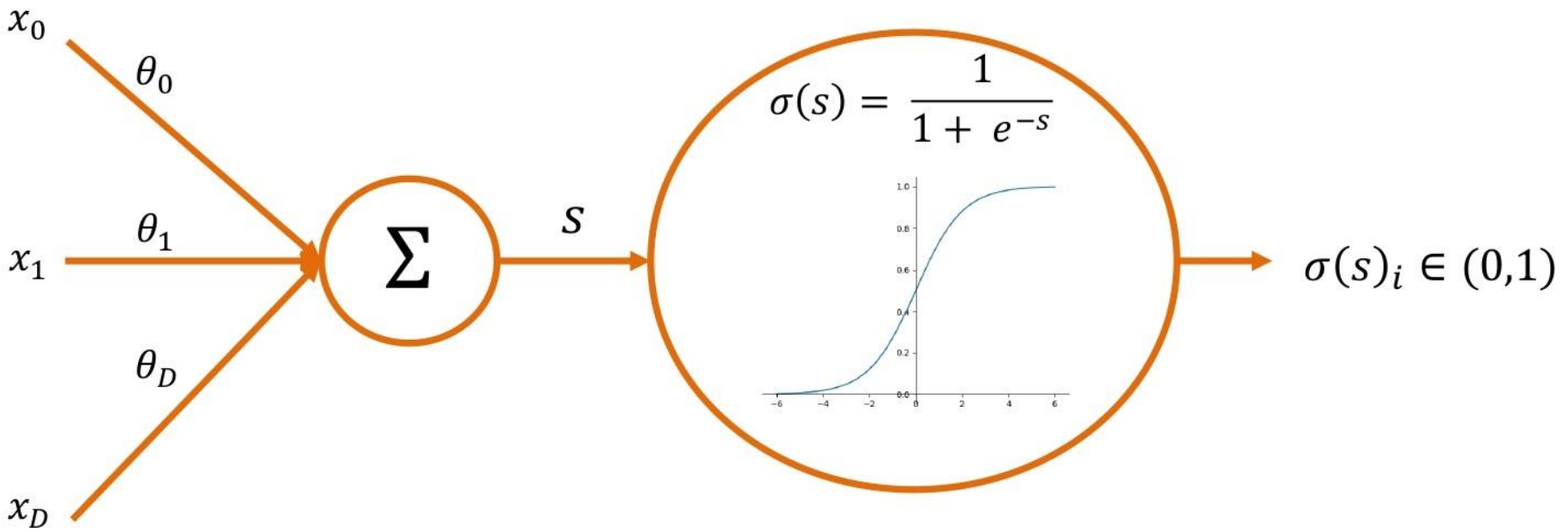


Activation Functions or Hidden Units



Sigmoid Activation

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



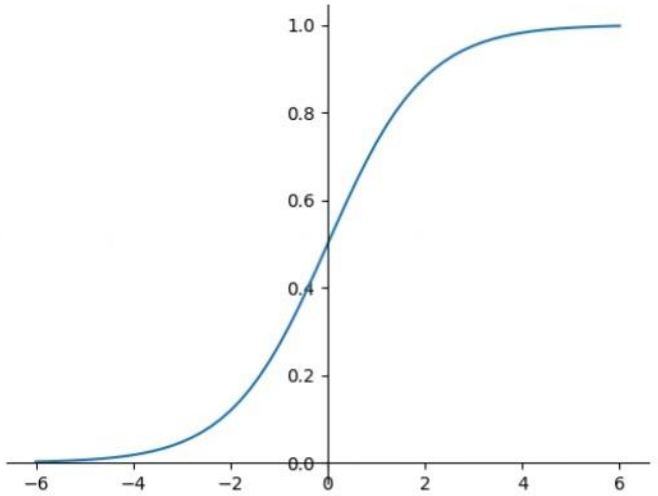
Sigmoid Activation

Forward

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

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

$\boldsymbol{x}^T$?



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

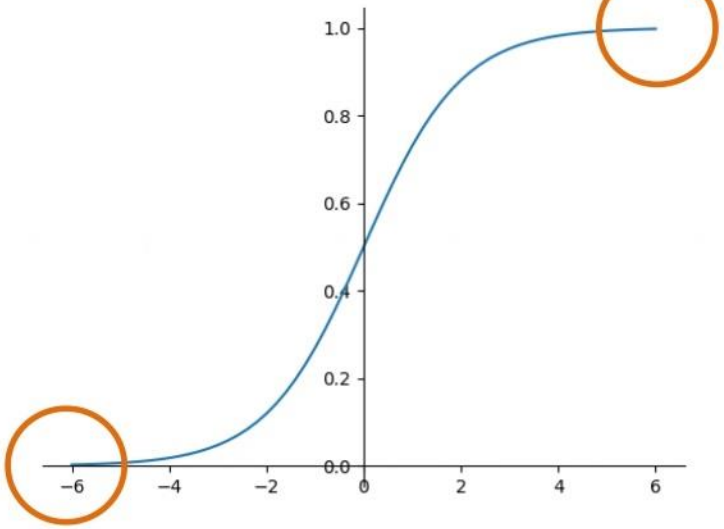
$$\frac{\partial \sigma}{\partial s}$$

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

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

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

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

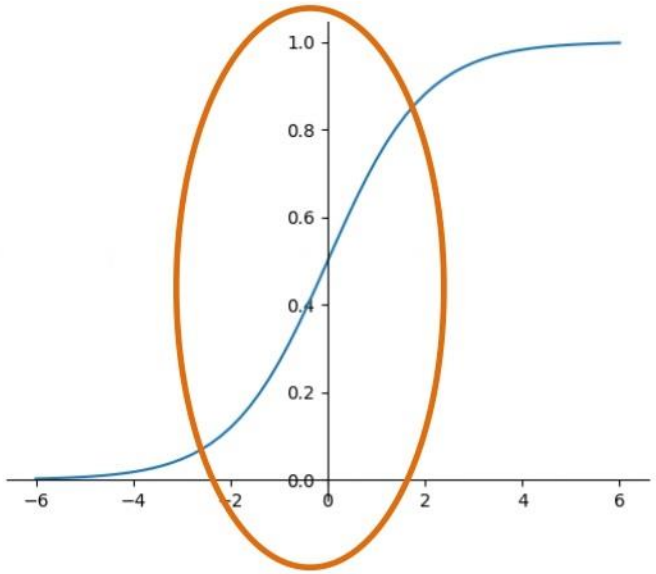
$$\frac{\partial L}{\partial \sigma}$$

Sigmoid Activation

Forward

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

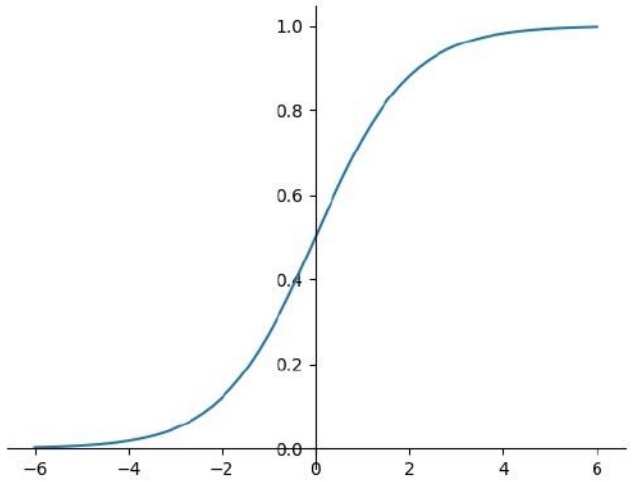
$$\frac{\partial L}{\partial w} = \frac{\partial s}{\partial w}\frac{\partial L}{\partial s}$$



Active region for gradient descent

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

Sigmoid Activation



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

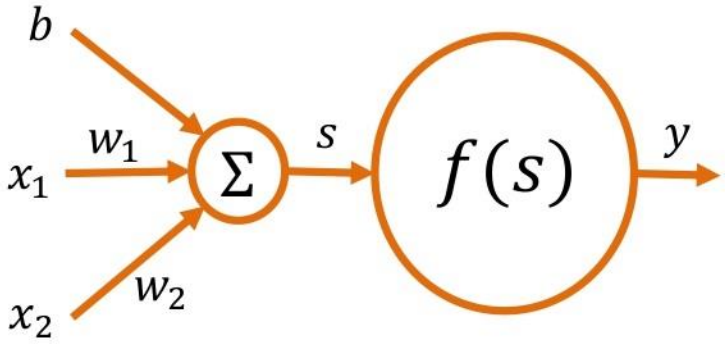
Output is always positive!

- Sigmoid output provides positive input for the next layer

What is the disadvantage of this?

Sigmoid Output not Zero-centered

- We want to compute the gradient w.r.t. the weights



Assume we have all positive data:

$\boldsymbol{x} = (x_1, x_2)^T > 0$

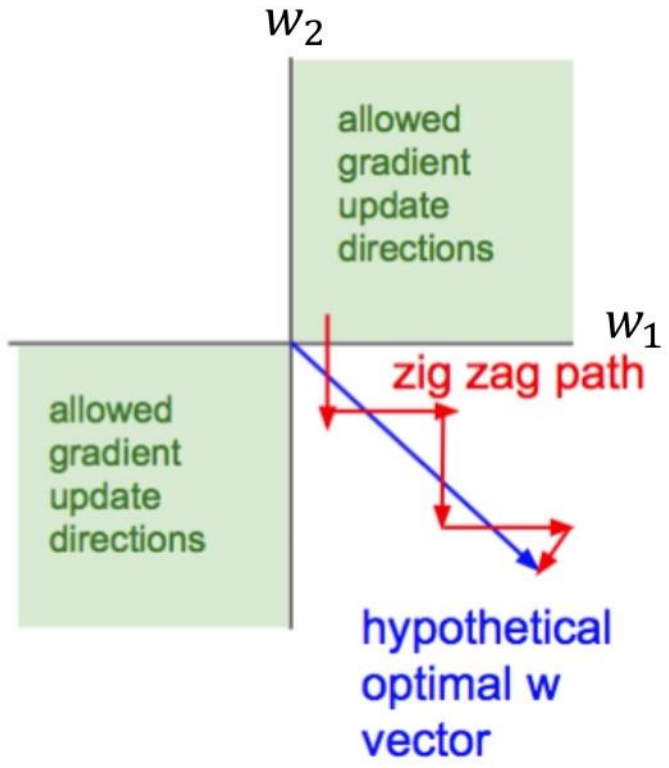
either positive or negative

$$\frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial y} \cdot \frac{\partial y}{\partial s} \cdot \frac{\partial s}{\partial w_1} \qquad x_1 > 0$$

$$\frac{\partial L}{\partial w_2} = \frac{\partial L}{\partial y} \cdot \frac{\partial y}{\partial s} \cdot \frac{\partial s}{\partial w_2} \qquad x_2 > 0$$

It is going to be either positive or negative for all weights' update.☹

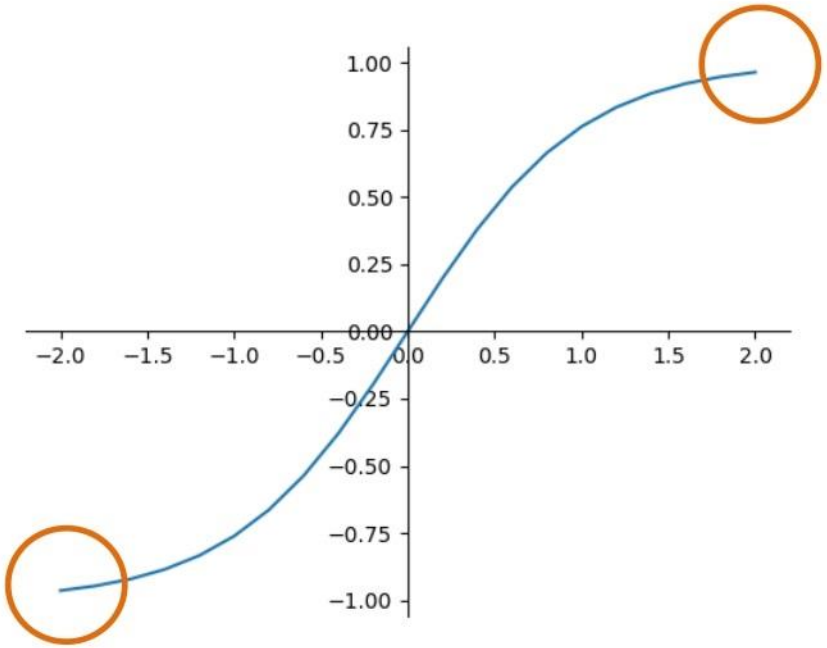
Sigmoid Output not Zero-centered



w_1, w_2 can only be increased or decreased at the same time, which is not good for update.

That is also why you need zero-centered data.

TanH Activation



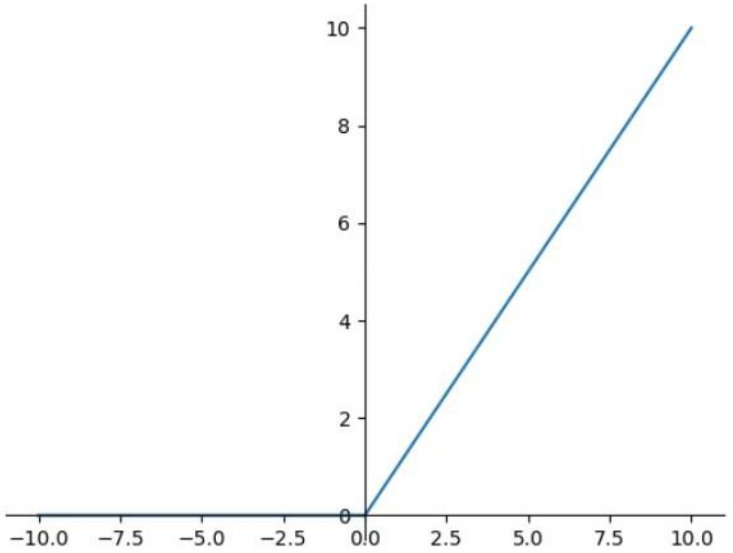
✗ Still saturates

✓ Zero-centered

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

Rectified Linear Units (ReLU)

$$\sigma(x) = \max(0, x)$$



Large and consistent gradients ✓

✓ Fast convergence

✓ Does not saturate

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

Rectified Linear Units (ReLU)

✗ Dead ReLU

What happens if a ReLU outputs zero?

Large and consistent gradients ✓

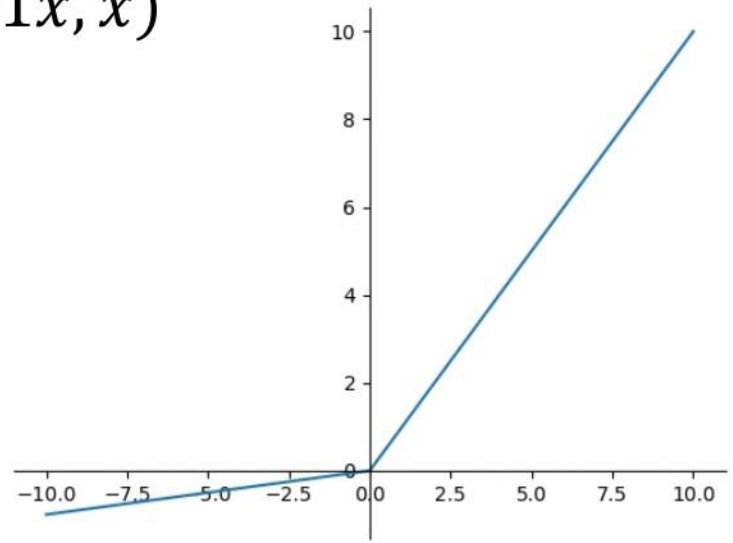
✓ Fast convergence

✓ Does not saturate

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

Leaky ReLU

$$\sigma(x) = \max(0.01x, x)$$



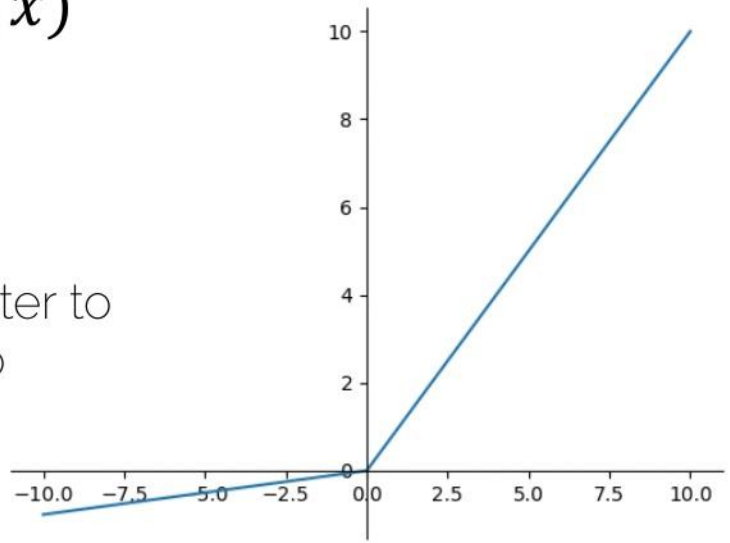
✓ Does not die

[Mass et al., ICML 2013] Rectifier Nonlinearities Improve Neural Network Acoustic Models

Parametric ReLU

$$\sigma(x) = \max(\alpha x, x)$$

One more parameter to backprop into

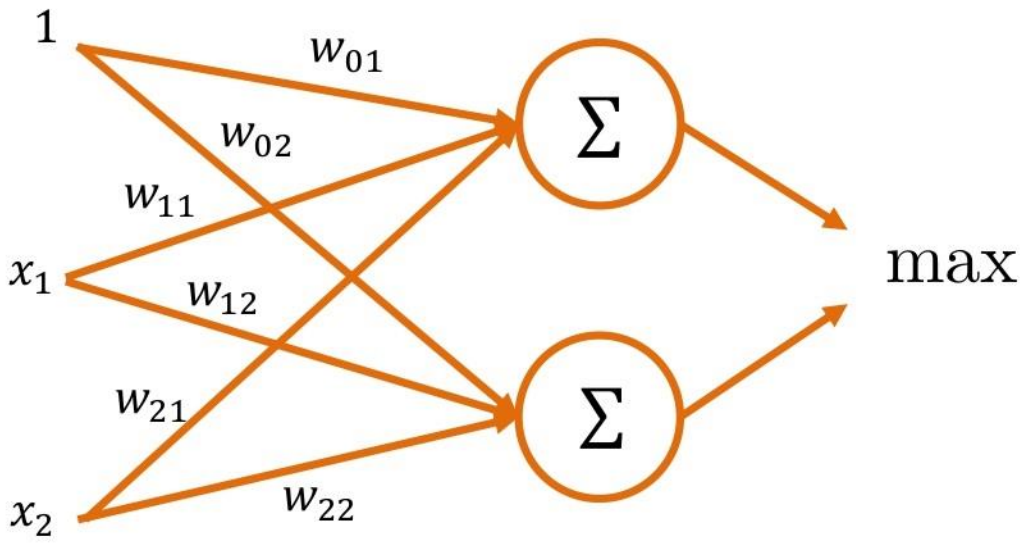


✓ Does not die

[He et al. ICCV 2015] Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification

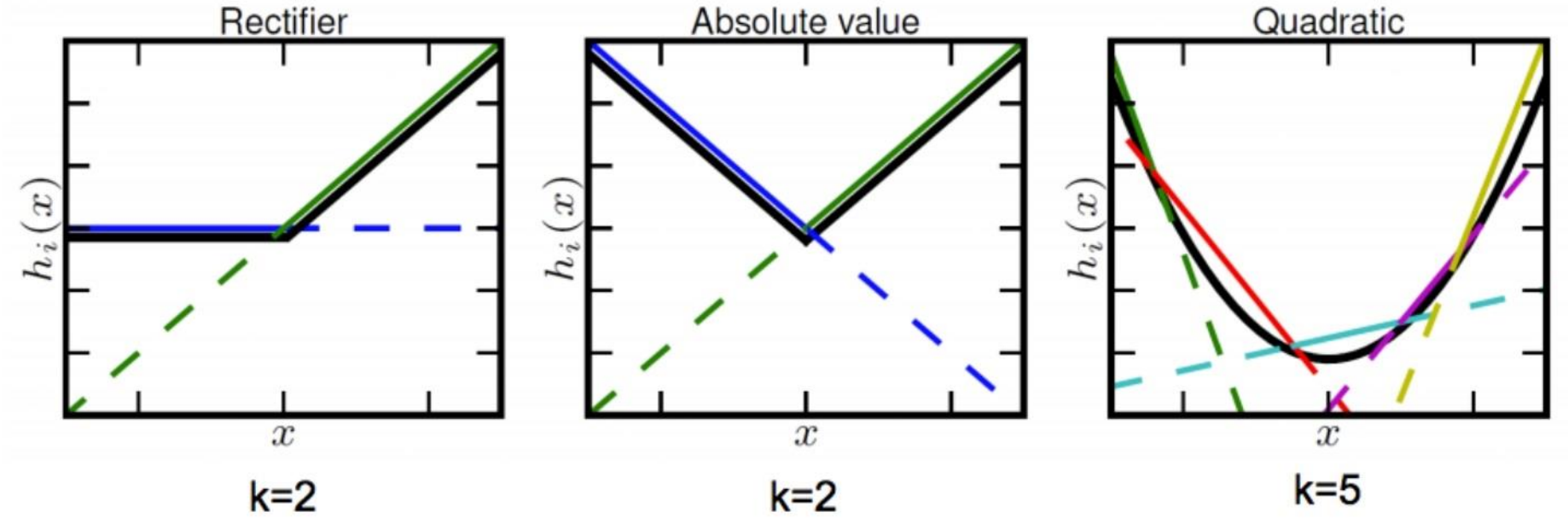
Maxout Units

$$Maxout = \max(w_1^T x + b_1, w_2^T x + b_2)$$



[Goodfellow et al. ICML 2013] Maxout Networks

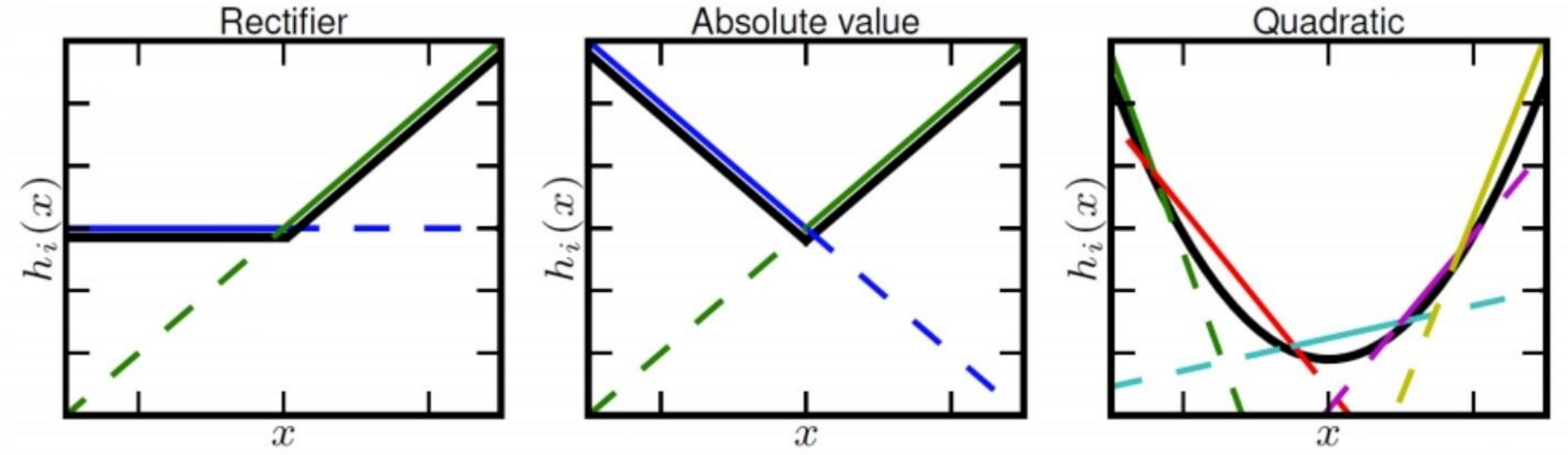
Maxout Units



Piecewise linear approximation of
a convex function with N pieces

[Goodfellow et al. ICML 2013] Maxout Networ

Maxout Units



✓ Generalization of ReLUs

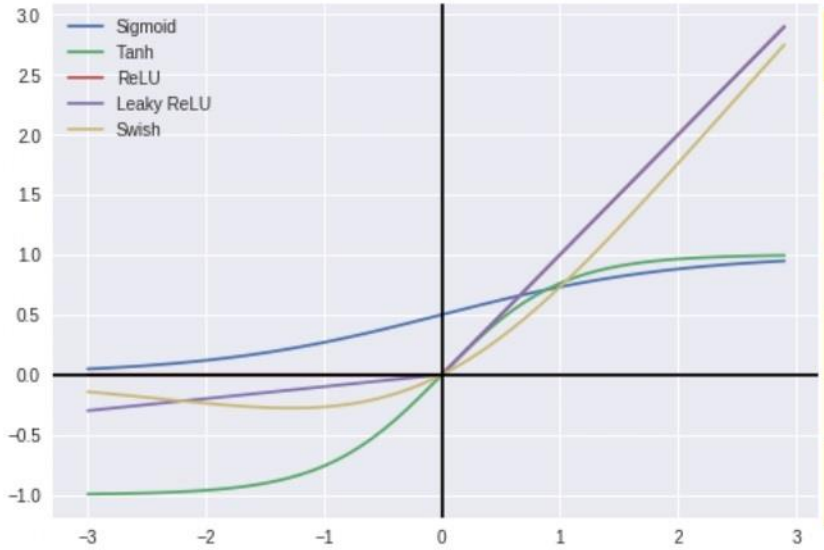
✓ Linear regimes

✓ Does not die

✓ Does not saturate

✗ Increases of the number of parameters

In a Nutshell



ACTIVATION FUNCTION	EQUATION	RANGE
Linear Function	$f(x) = x$	$(-\infty, \infty)$
Step Function	$f(x) = \begin{cases} 0 \; for \; x < 0 \\ 1 \; for \; x \geq 0 \end{cases}$	$\{0, 1\}$
Sigmoid Function	$f(x) = \sigma(x) = \frac{1}{1 + e^{-x}}$	$(0, 1)$
Hyperbolic Tanjant Function	$f(x) = \tanh(x) = \frac{(e^{x} - e^{-x})}{(e^{x} + e^{-x})}$	$(-1, 1)$
ReLU	$f(x) = \begin{cases} 0 \; for \; x < 0 \\ x \; for \; x \geq 0 \end{cases}$	$[0, \infty)$
Leaky ReLU	$f(x) = \begin{cases} 0.01 \; for \; x < 0 \\ x \; for \; x \geq 0 \end{cases}$	$(-\infty, \infty)$
Swish Function	$f(x) = 2x\sigma(\beta x) = \begin{cases} \beta = 0 \; for \; f(x) = x \\ \beta \to \infty \; for \; f(x) = 2\max(0, x) \end{cases}$	$(-\infty, \infty)$

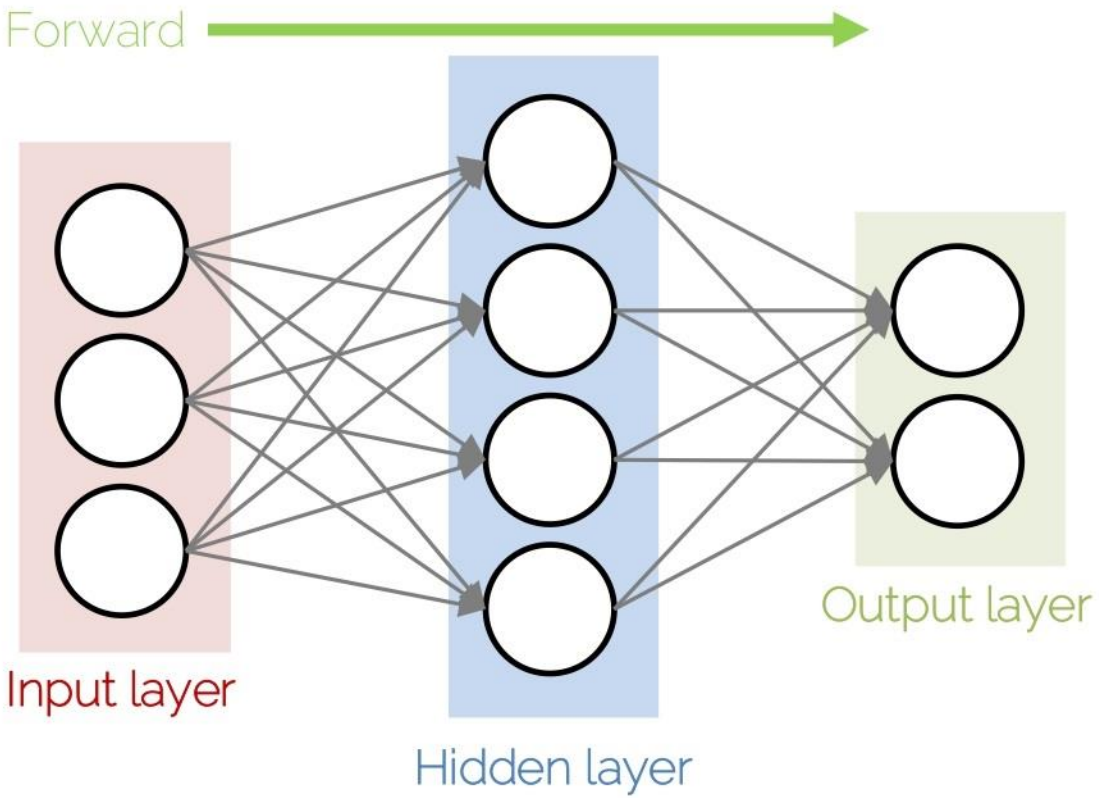
Source : https://towardsdatascience.com/comparison-of-activation-functions-for-deep-neural-networks-706ac4284c8a

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.

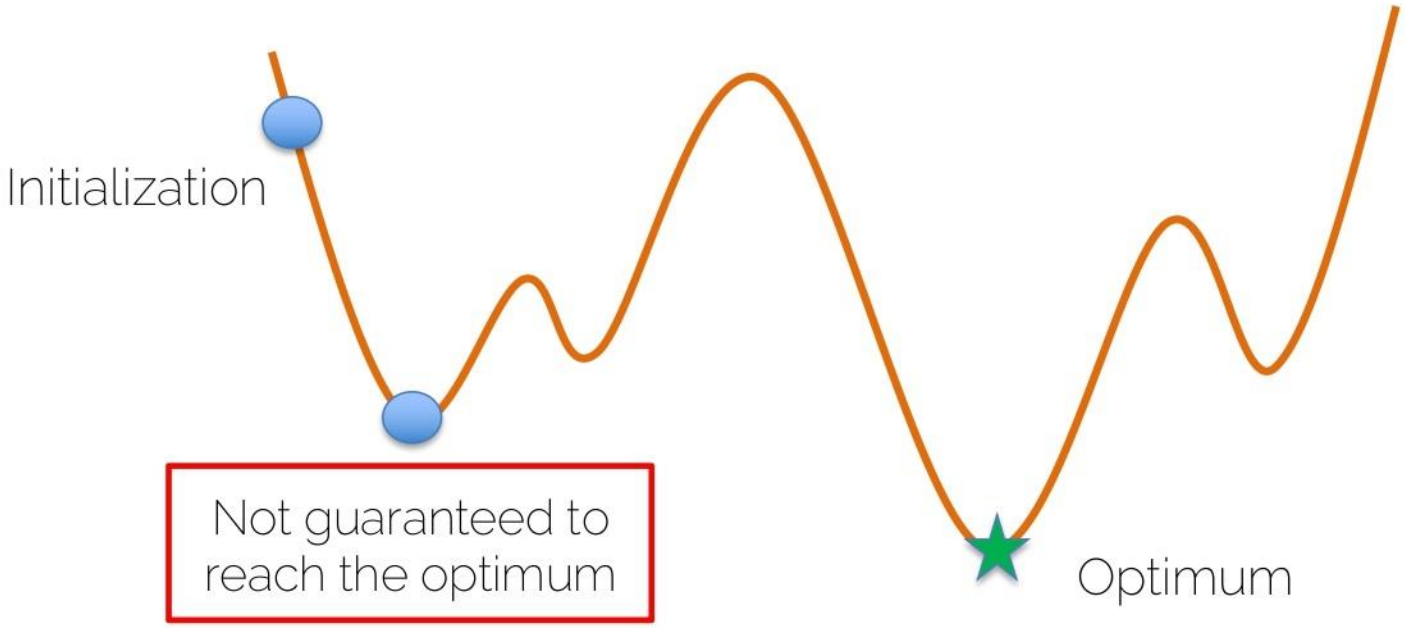
Weight Initialization

How do I start?



Initialization is Extremely Important

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



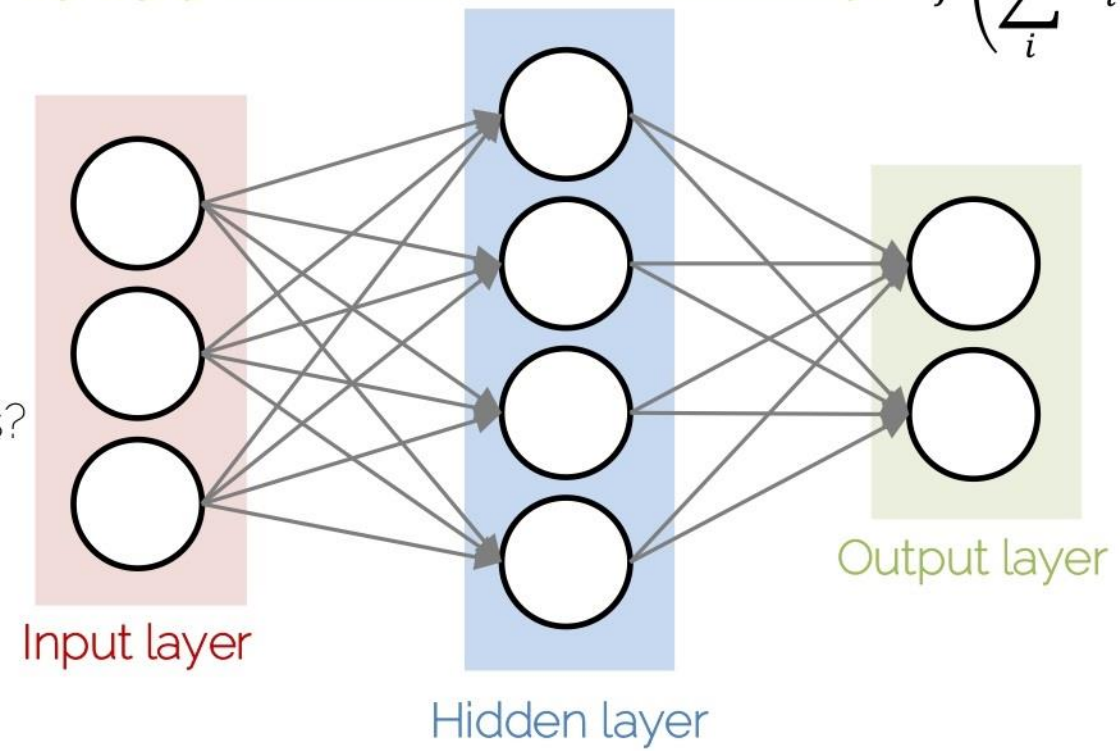
How do I start?

Forward

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

$w = 0$

What happens to the gradients?



All Weights Zero

- What happens to the gradients?

- The hidden units are all going to compute the same function, gradients are going to be the same
  - No symmetry breaking

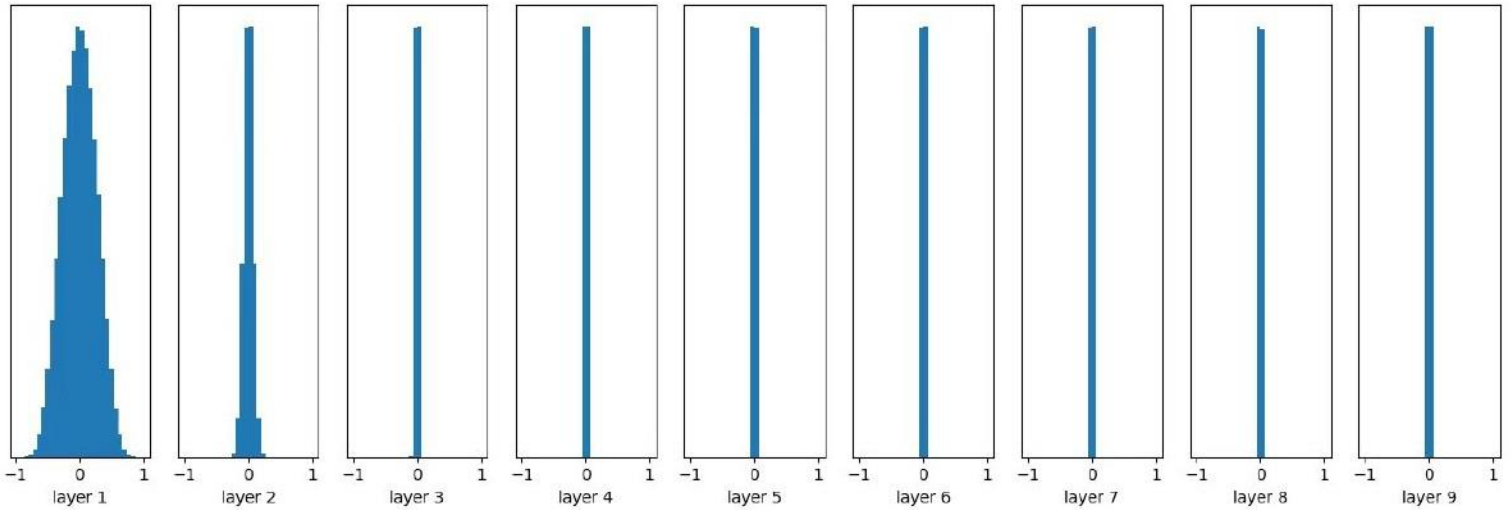
Small Random Numbers

- Gaussian with zero mean and standard deviation 0.01

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

Small Random Numbers

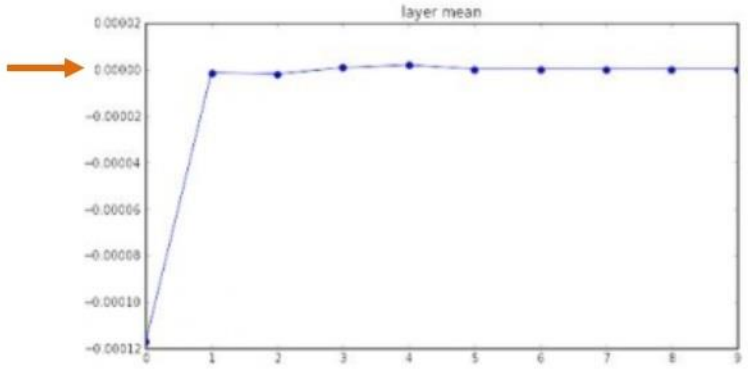
tanh as activation functions

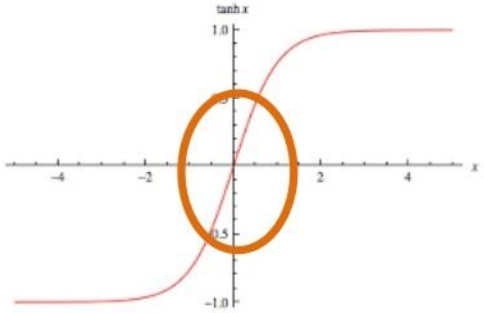


Output become to zero

Forward

Small Random Numbers





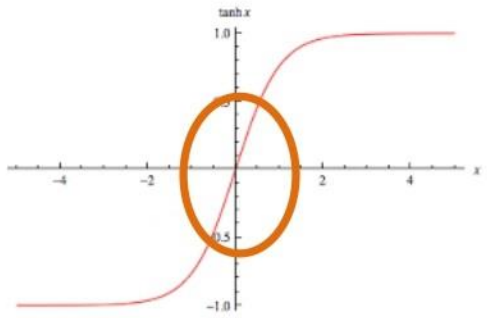
Small w_i^l cause small output for layer l:

$$f_l\left(\sum_i w_i^l x_i^l + b^l\right) \approx \mathbf{0}$$

Forward

Small Random Numbers

Even activation function's gradient is ok, we still have vanishing gradient problem.



Small outputs of layer l (input of layer $l+1$) cause small gradient w.r.t to the weights of layer $l+1$:

$$f_{l+1}\left(\sum_i w_i^{l+1} x_i^{l+1} + b^{l+1}\right)$$

$$\frac{\partial L}{\partial w_i^{l+1}} = \frac{\partial L}{\partial f_{l+1}} \cdot \frac{\partial f_{l+1}}{\partial w_i^{l+1}} = \frac{\partial L}{\partial f_{l+1}} \cdot x_i^{l+1} \approx 0$$

Vanishing gradient, caused by small output

Backward

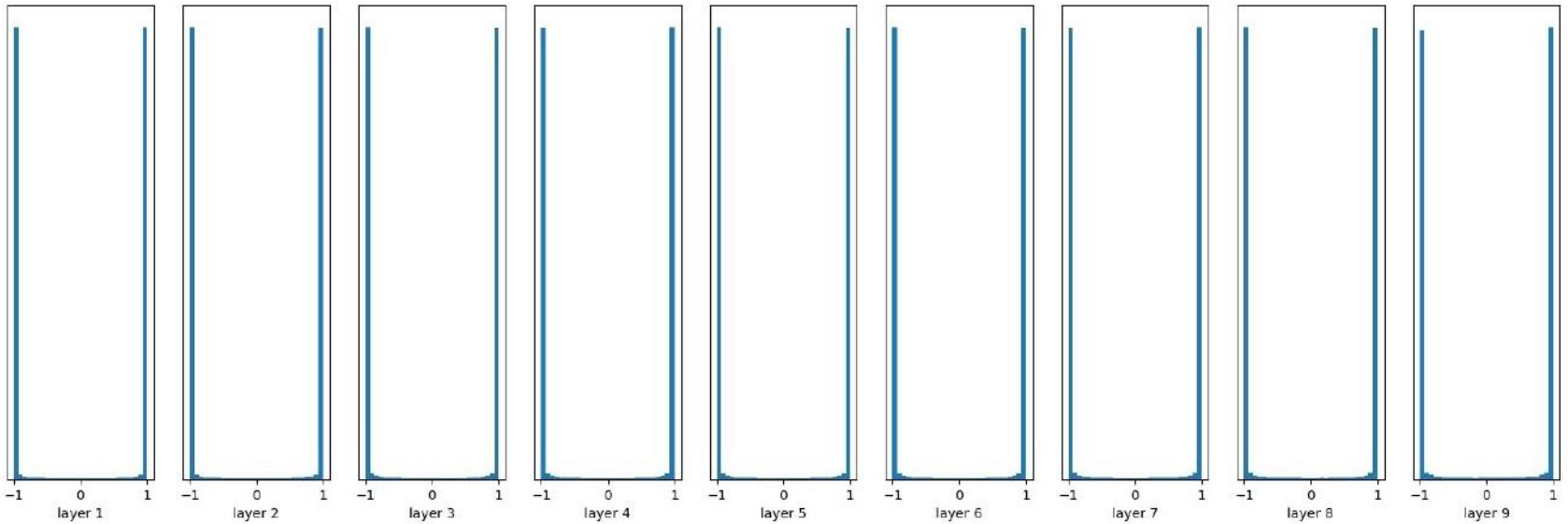
Big Random Numbers

- 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

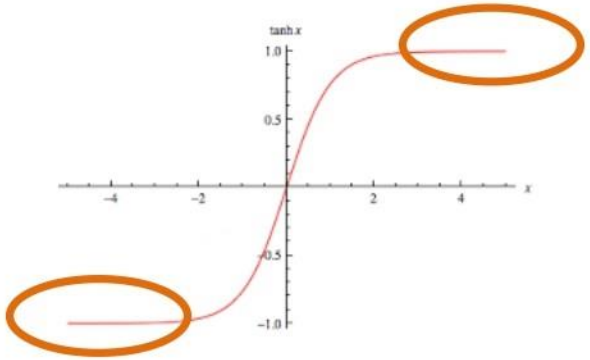
tanh as activation functions



Output saturated to -1 and 1

Big Random Numbers

Output saturated to -1 and 1.
Gradient of the activation function becomes close to 0.



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

$$\frac{\partial L}{\partial w_i} = \frac{\partial L}{\partial f} \cdot \frac{\partial f}{\partial s} \cdot \frac{\partial s}{\partial w_i} \approx 0$$

Vanishing gradient, caused by saturated activation function.

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?

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

Notice: $\boldsymbol{n}$ is the number of input neurons for the layer of weights you want to initialized. This $\boldsymbol{n}$ is not the number $\boldsymbol{N}$ of input data $\boldsymbol{X} \in \boldsymbol{R}^{\boldsymbol{N}\times\boldsymbol{D}}$. For the first layer $\boldsymbol{n} = \boldsymbol{D}$.

Tipps:

$$E[X^2] = Var[X] + E[X]^2$$

If X, Y are independent:

$$Var[XY] = E[X^2Y^2] - E[XY]^2$$
$$E[XY] = E[X]E[Y]$$

[Glorot and Bengio, AISTATS'10] Xavier Initialization

Xavier Initialization

- Gaussian with zero mean, but what standard deviation?

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

Independent

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

Zero mean

Zero mean

[Glorot and Bengio, AISTATS'10] Xavier Initialization

Xavier Initialization

- Gaussian with zero mean, but what standard deviation?

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

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

$$= \sum_i^n Var(x_i)Var(w_i) = n(Var(w)Var(x))$$

Identically distributed (each random variable has the same distribution)

[Glorot and Bengio, AISTATS'10] Xavier Initialization

Xavier Initialization

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

Goal:

$$Var(s) = Var(x) \longrightarrow \underbrace{n \cdot Var(w)}_{=1} Var(x) = Var(x)$$

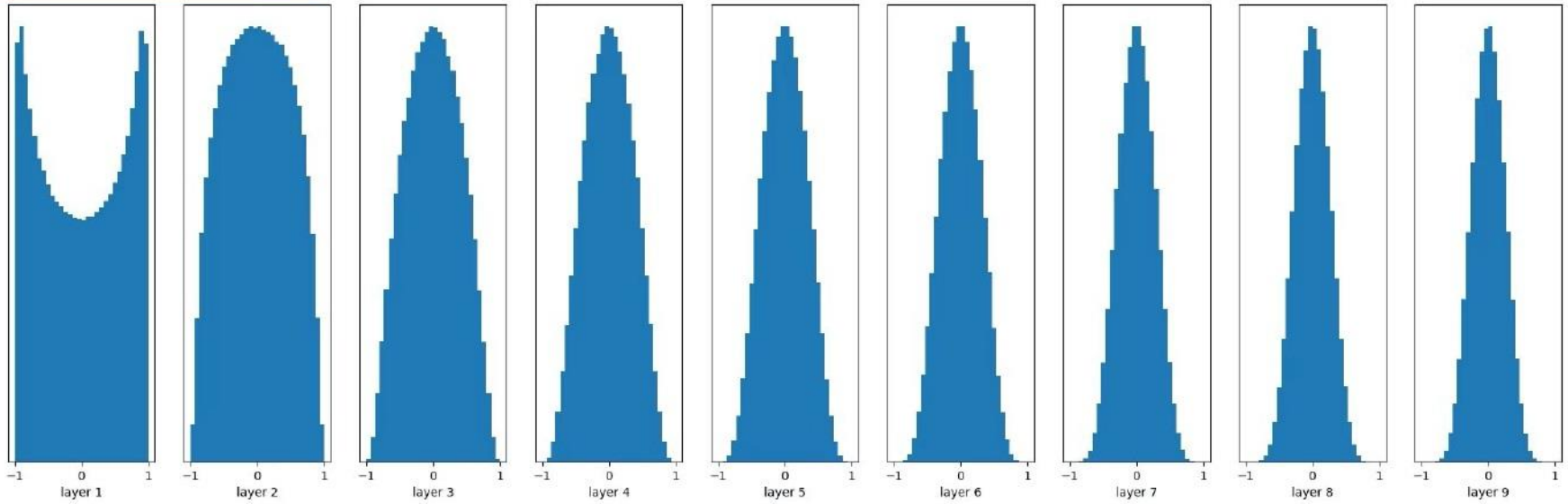
$$\longrightarrow Var(w) = \frac{1}{n}$$

n: number of input neurons

Xavier Initialization

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

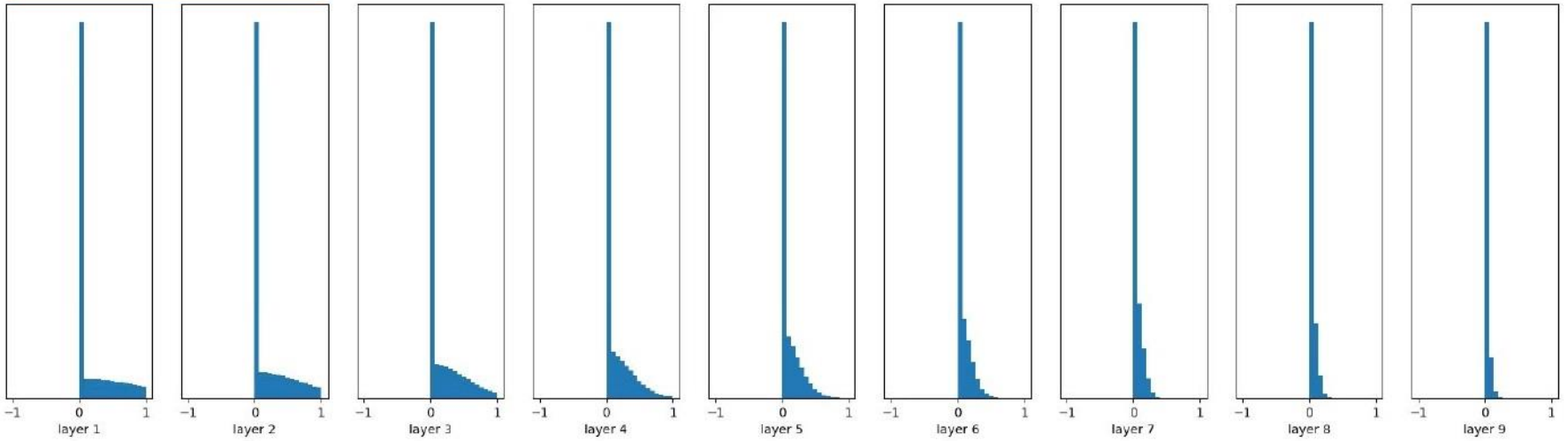
tanh as activation functions



Xavier Initialization with ReLU

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

tanh as activation functions



ReLU kills Half of the Data

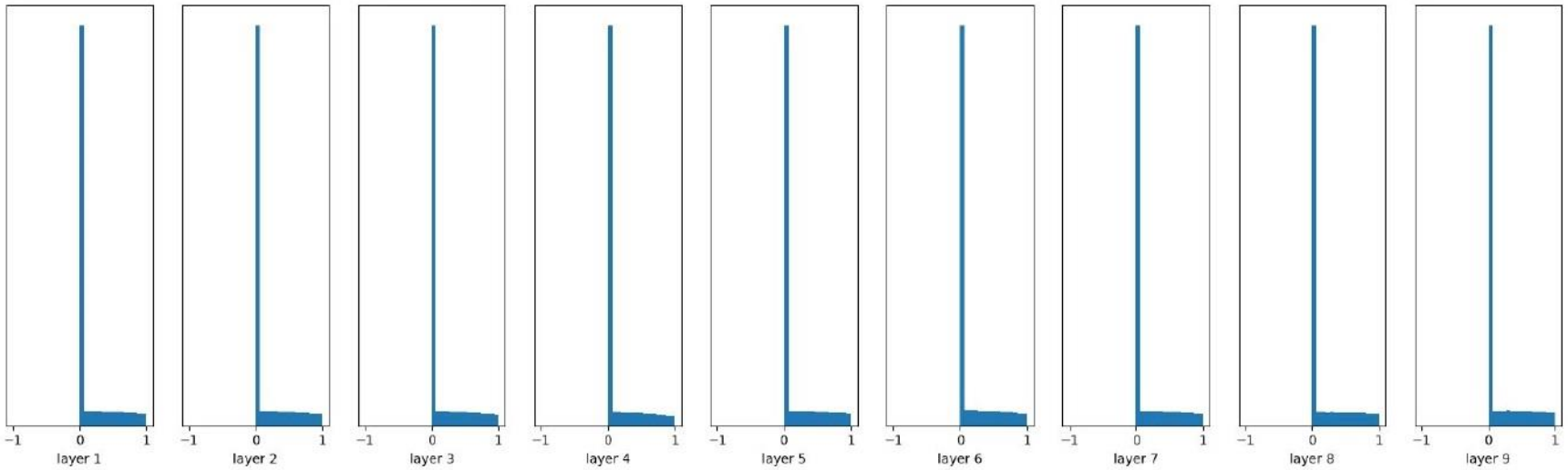
What's the solution?

When using ReLU, output close to zero again ☹

Xavier/2 Initialization with ReLU

$$Var(w) = \frac{1}{n/2} = \frac{2}{n}$$

tanh as activation functions



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

Summary

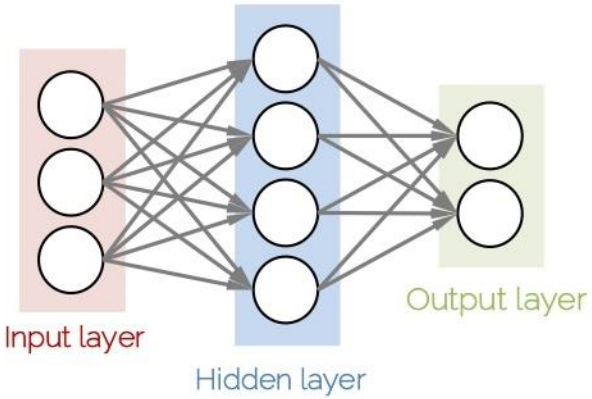


Image Classification	Output Layer	Loss function
Binary Classification	Sigmoid	Binary Cross entropy
Multiclass Classification	Softmax	Cross entropy

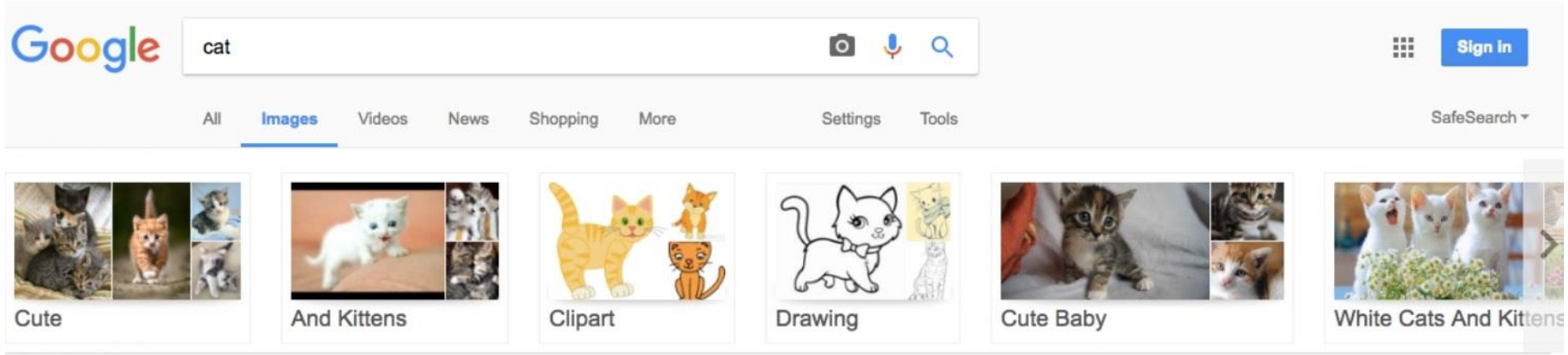
Other Losses:
SVM Loss (Hinge Loss), L1/L2-Loss

Initialization of optimization
- How to set weights at beginning

Data Augmentation

Data Augmentation

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



Pose Appearance Illumination

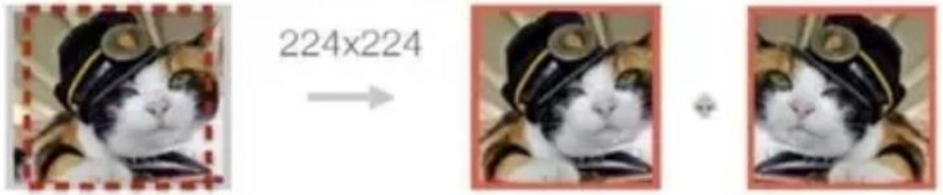
Data Augmentation

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

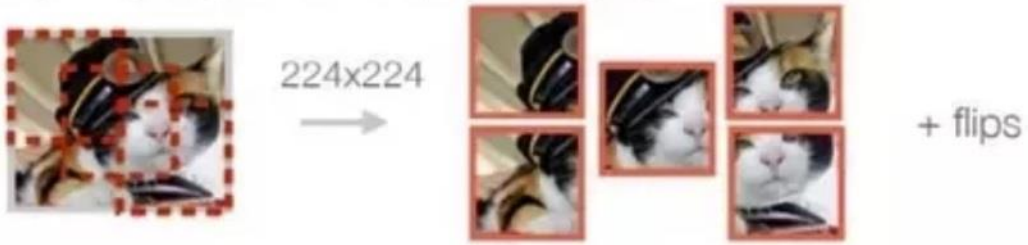
Data Augmentation

a. No augmentation (= 1 image)

b. Flip augmentation (= 2 images)



c. Crop+Flip augmentation (= 10 images)



[Krizhevsky et al., NIPS'12] ImageNet

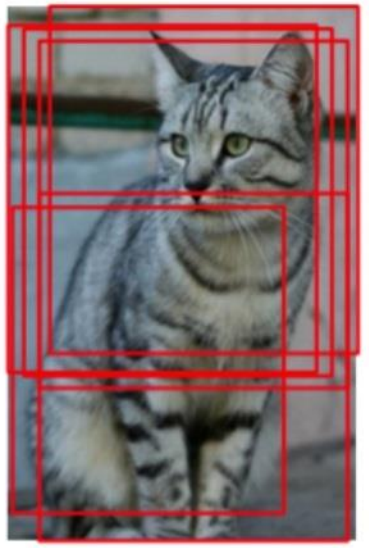
Data Augmentation: Brightness

- 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

- When comparing two networks make sure to use the same data augmentation!

- Consider data augmentation a part of your network design

Advanced Regularization

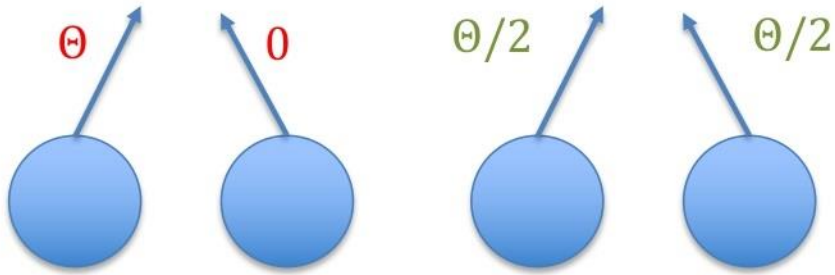
Weight Decay

- L2 regularization

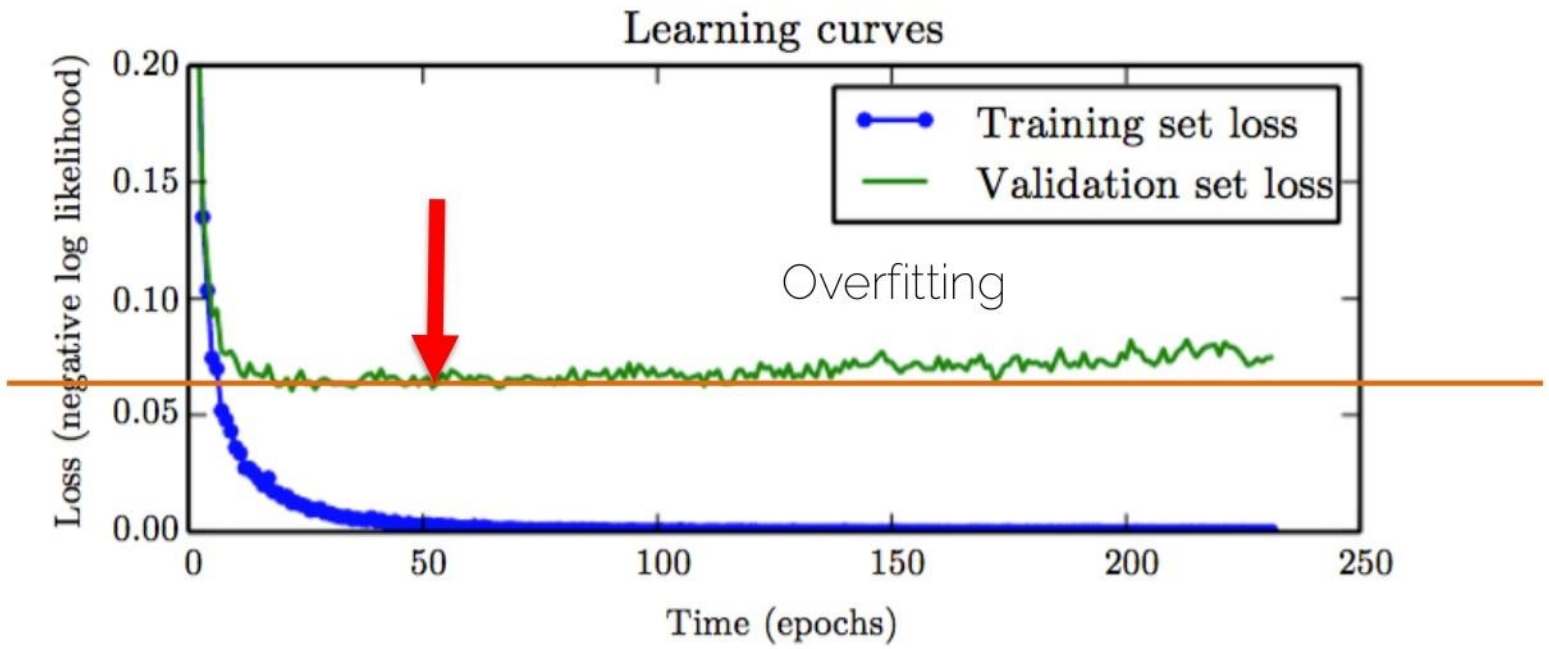
$$\Theta_{k+1} = \Theta_k - \epsilon \nabla_\Theta (\Theta_k, x, y) - \lambda \theta_k$$

Learning rate — Gradient — Gradient of L2-regularization

- Penalizes large weights
- Improves generalization

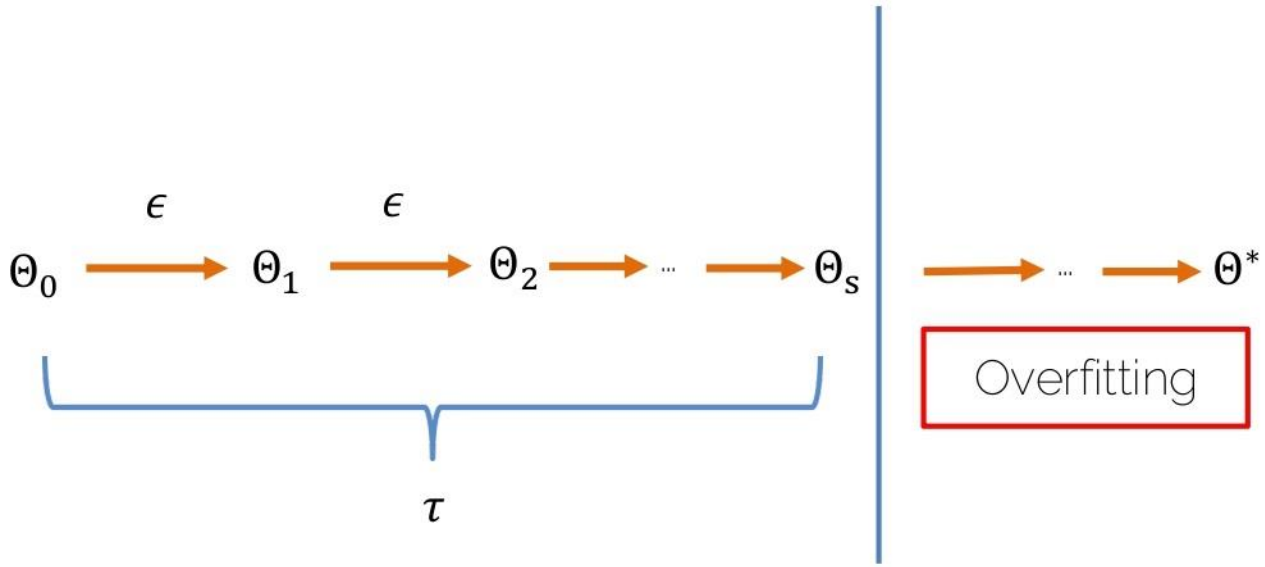


Early Stopping



Early Stopping

- Easy form of regularization

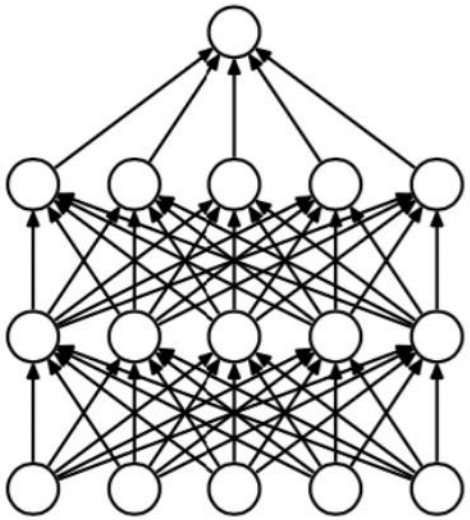


Bagging and Ensemble Methods

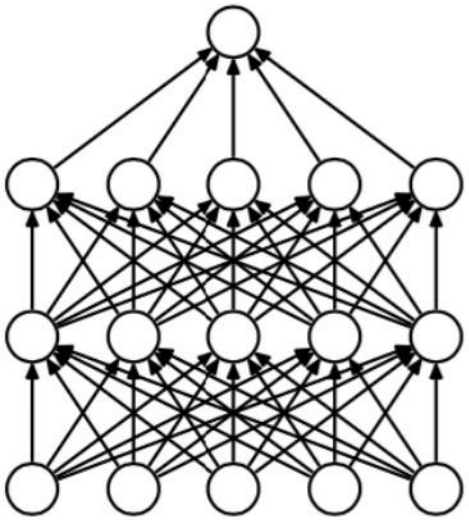
- Train multiple models and average their results
- E.g., use a different algorithm for optimization or change the objective function / loss function.
- If errors are uncorrelated, the expected combined error will decrease linearly with the ensemble size

Bagging and Ensemble Methods

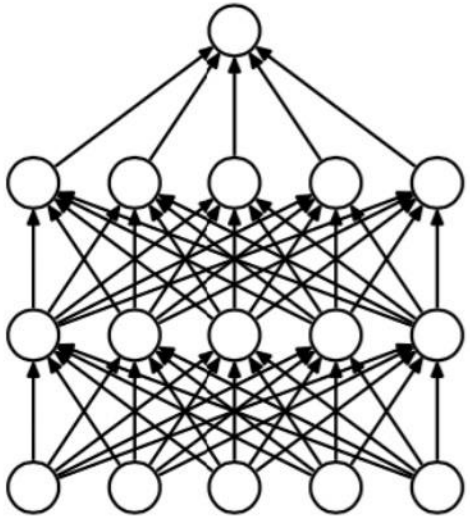
- Bagging: uses k different datasets



Training Set 1



Training Set 2



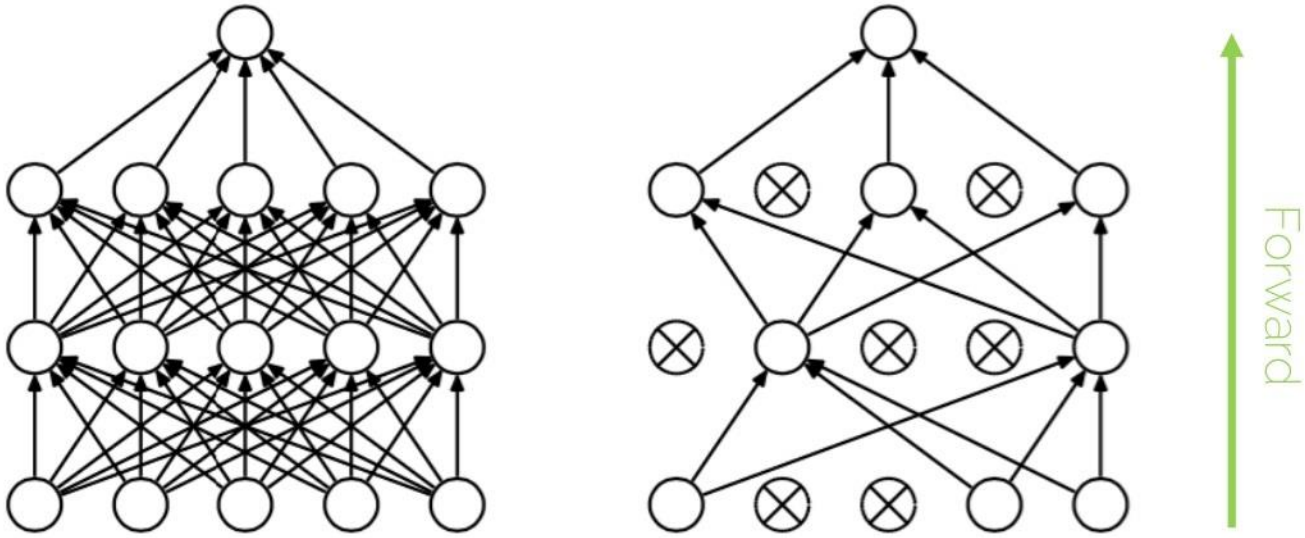
Training Set 3

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

Dropout

Dropout

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

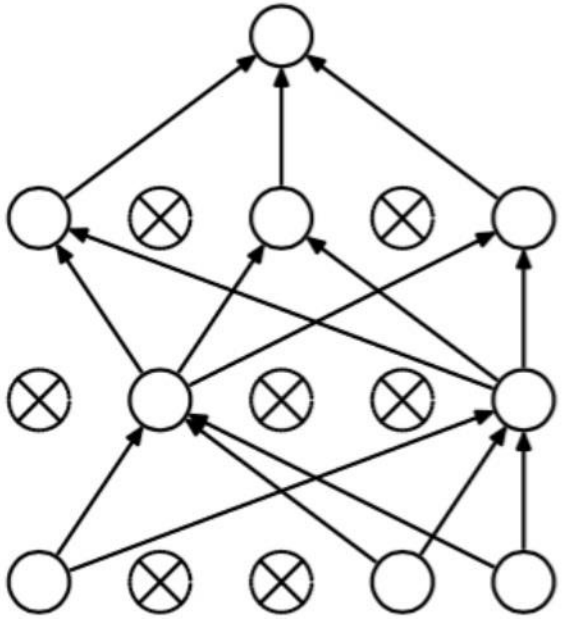


(a) Standard Neural Net (b) After applying dropout.

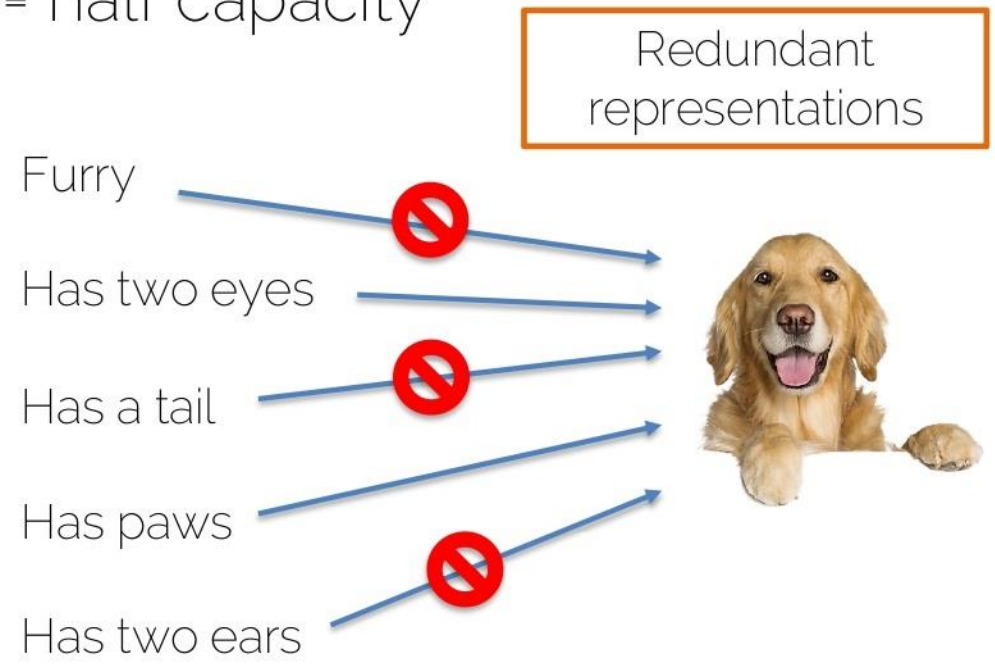
[Srivastava et al., JMLR'14] Dropout

Dropout: Intuition

- Using half the network = half capacity



(b) After applying dropout.



[Srivastava et al., JMLR'14] Dropout

Dropout: Intuition

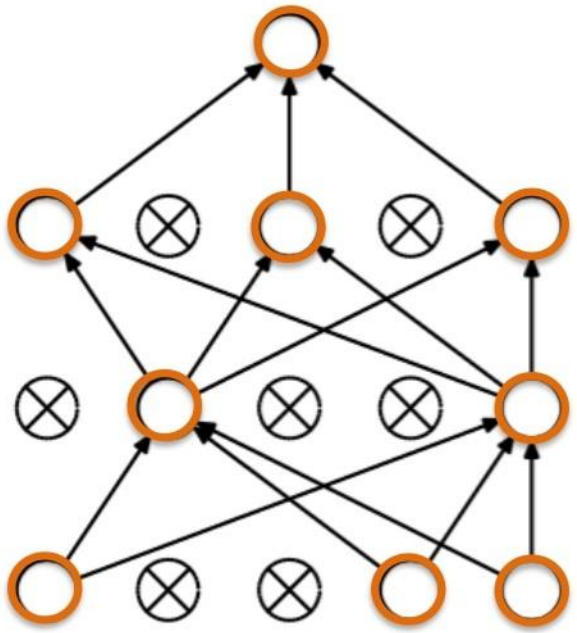
- Using half the network = half capacity
  - Redundant representations
  - Base your scores on more features

- 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

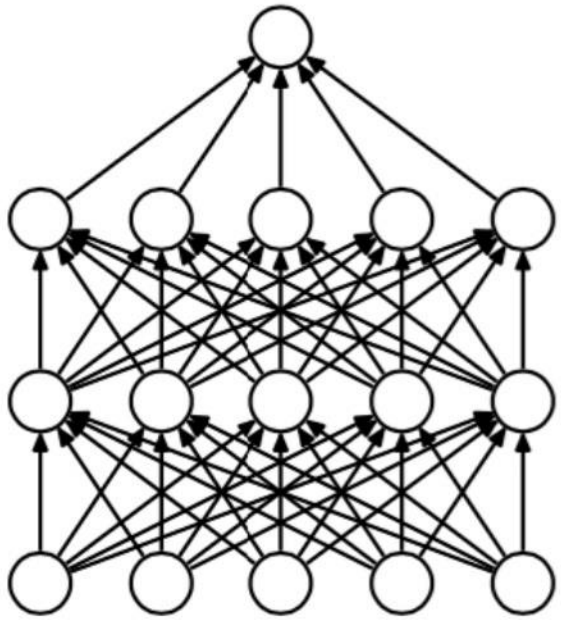
- Consider it as two models in one
  - Training a large ensemble of models, each on different set of data (mini-batch) and with SHARED parameters

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

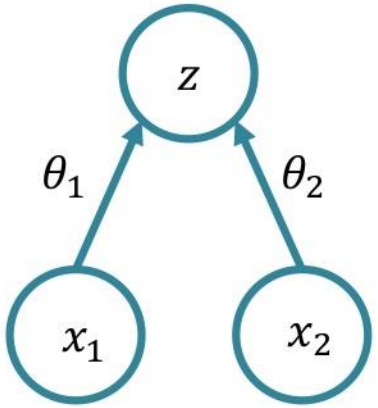
Dropout: Test Time

- Test:

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

Dropout probability $p = 0.5$

- Train:



Weight scaling inference rule

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

[Srivastava et al., JMLR'14] Dropout

Dropout: Verdict

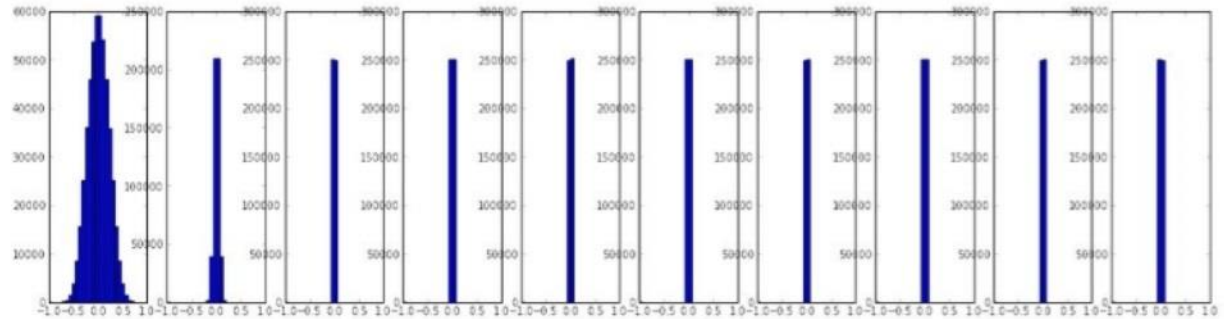
- Efficient bagging method with parameter sharing
- Try it!
- Dropout reduces the effective capacity of a model → larger models, more training time

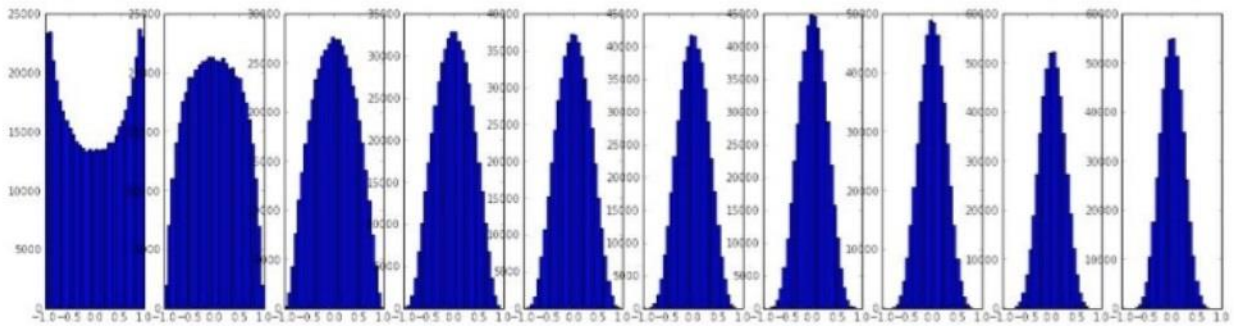
[Srivastava et al., JMLR'14] Dropout

Batch Normalization

Our Goal

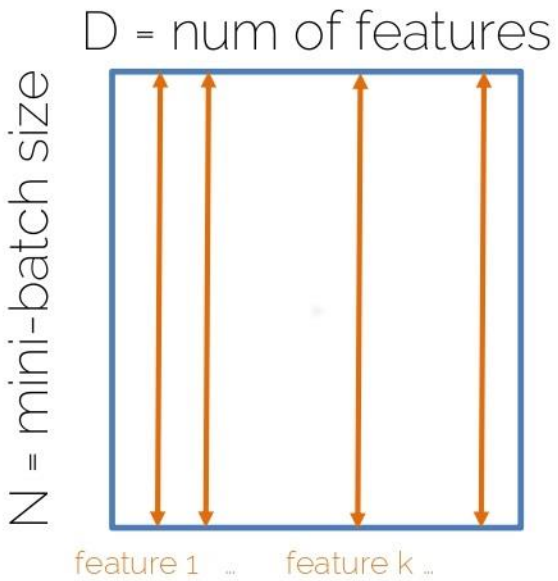
- All we want is that our activations do not die out





Batch Normalization

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



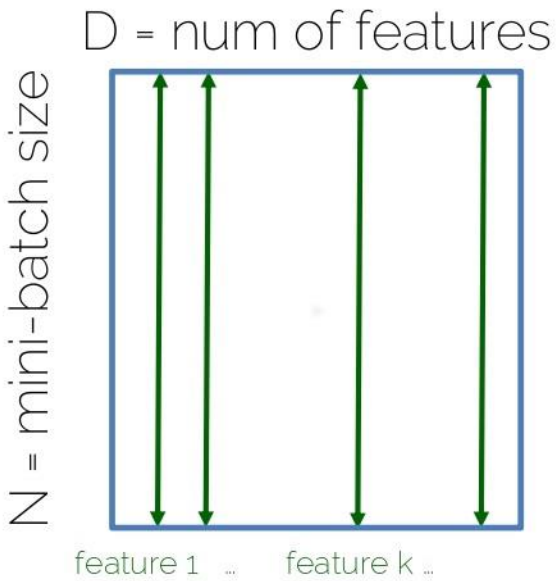
Mean of your mini-batch examples over feature k

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

[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

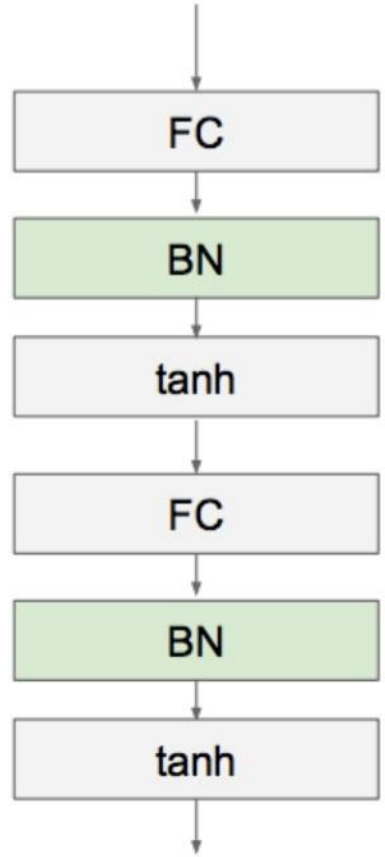
Batch Normalization

- 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

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

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

Differentiable function so we can backprop through it….

- 2. Allow the network to change the range

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

These parameters will be optimized during backprop

[Ioffe and Szegedy, PMLR'15] Batch Normalization

Batch Normalization

- 1. Normalize

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

- 2. Allow the network to change the range

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

backprop

The network *can* learn to undo the normalization

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

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

[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

BN: Train vs Test

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

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

- 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 σ^2_{test} and μ_{test} .

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

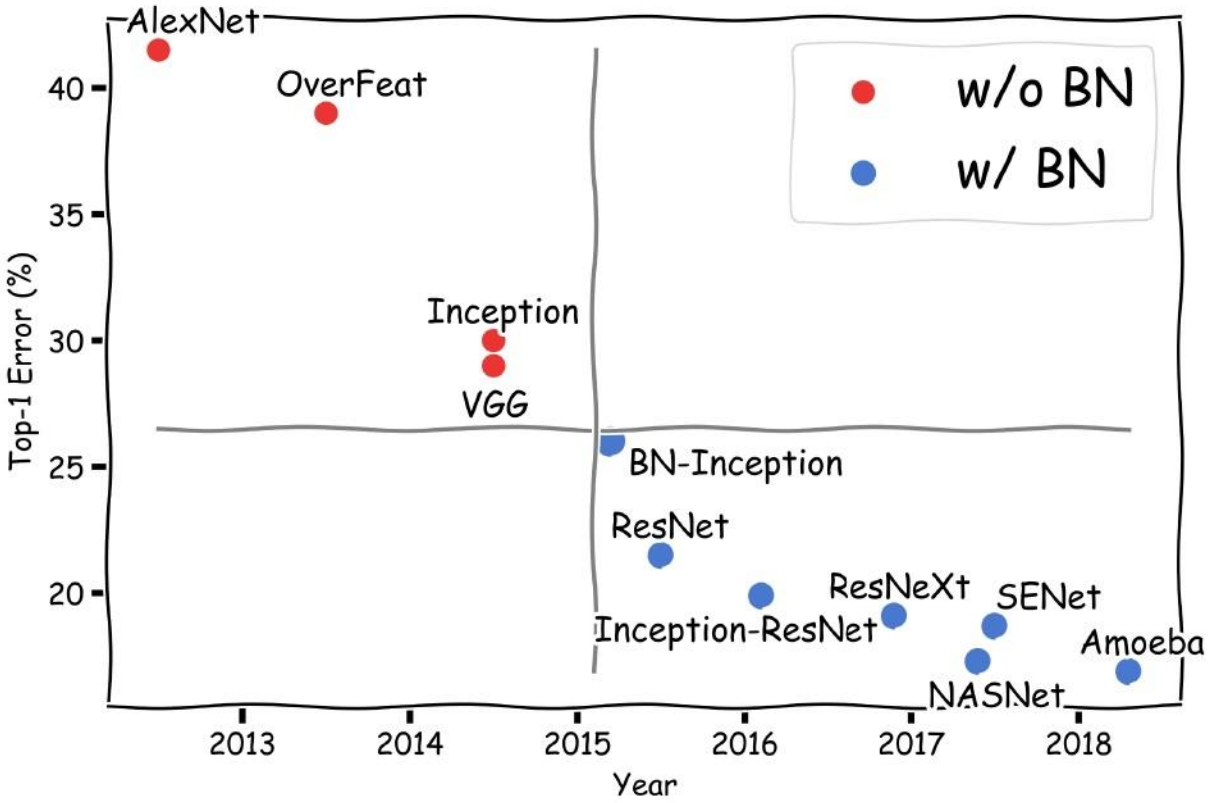
β_m: momentum (hyperparameter)

BN: What do you get?

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

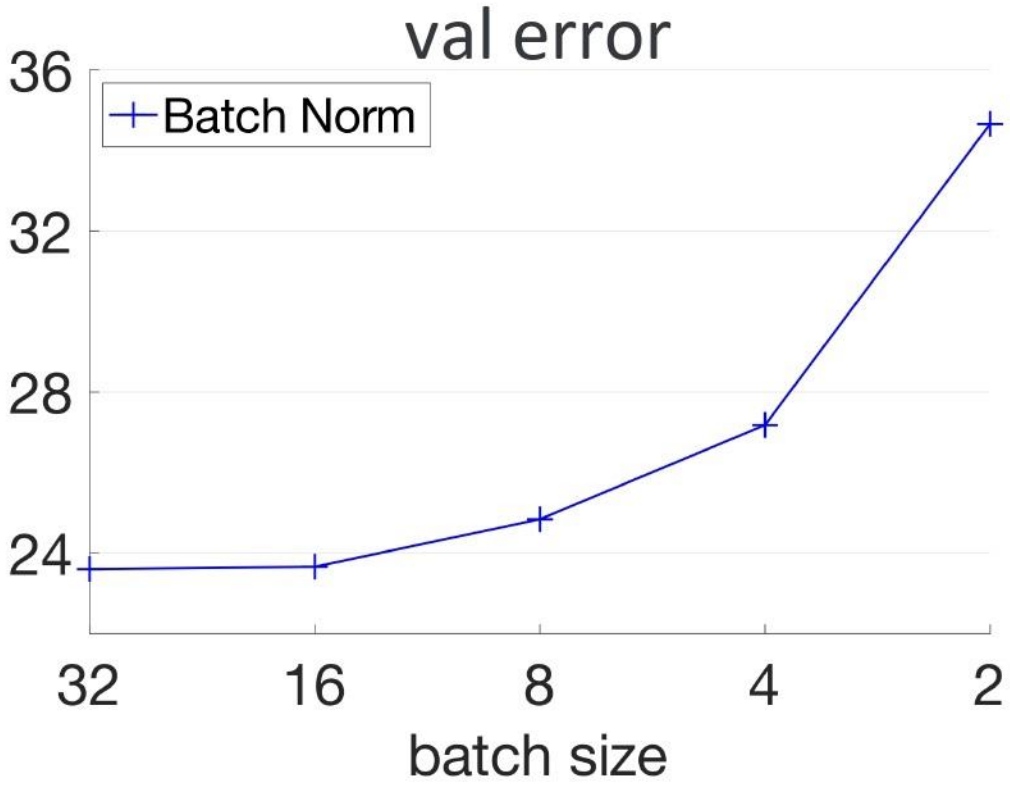
- A much larger range of hyperparameters works similarly when using BN

BN: A Milestone



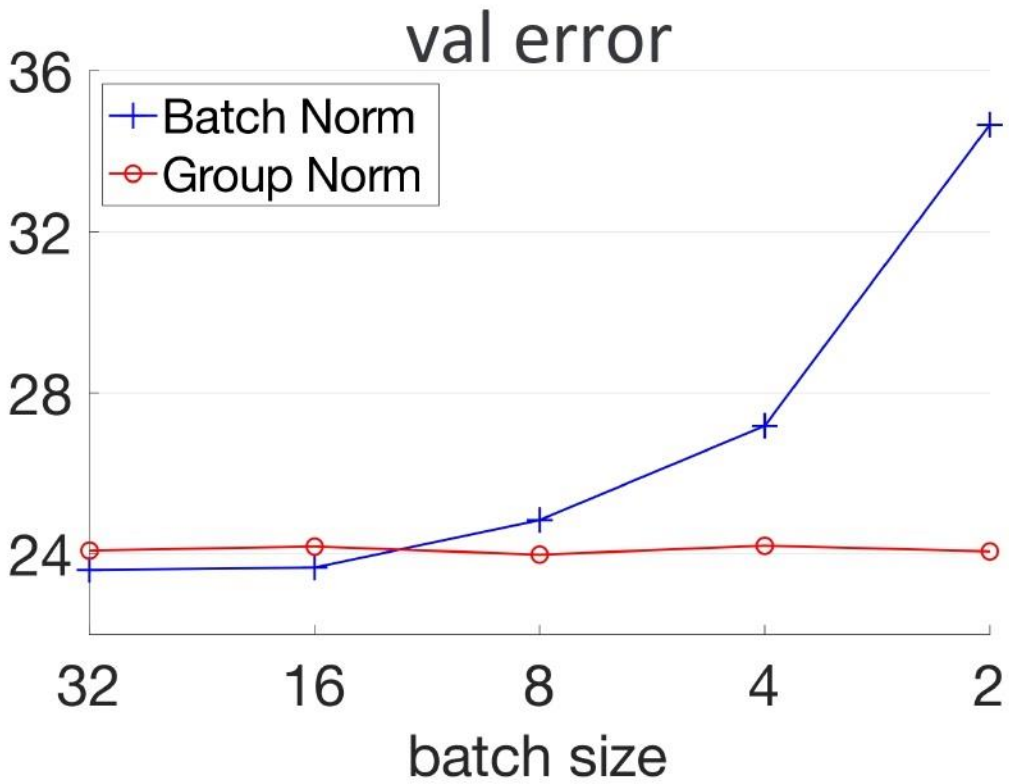
[Wu and He, ECCV'18] Group Normalization

BN: Drawbacks



[Wu and He, ECCV'18] Group Normalization

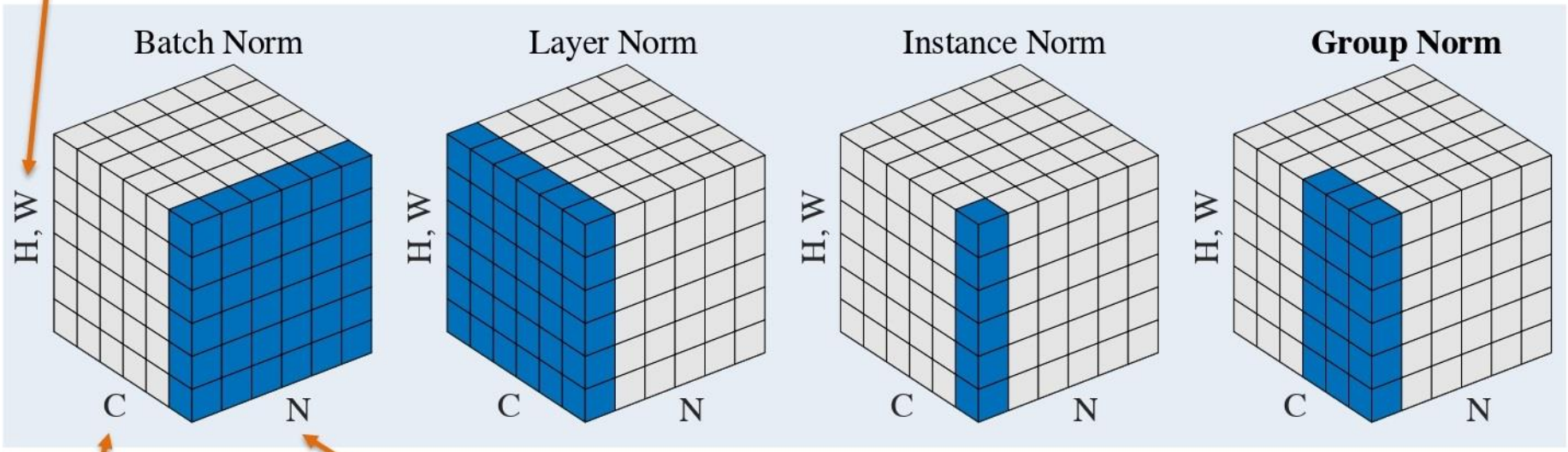
Other Normalizations



[Wu and He, ECCV'18] Group Normalization

Other Normalizations

Image size



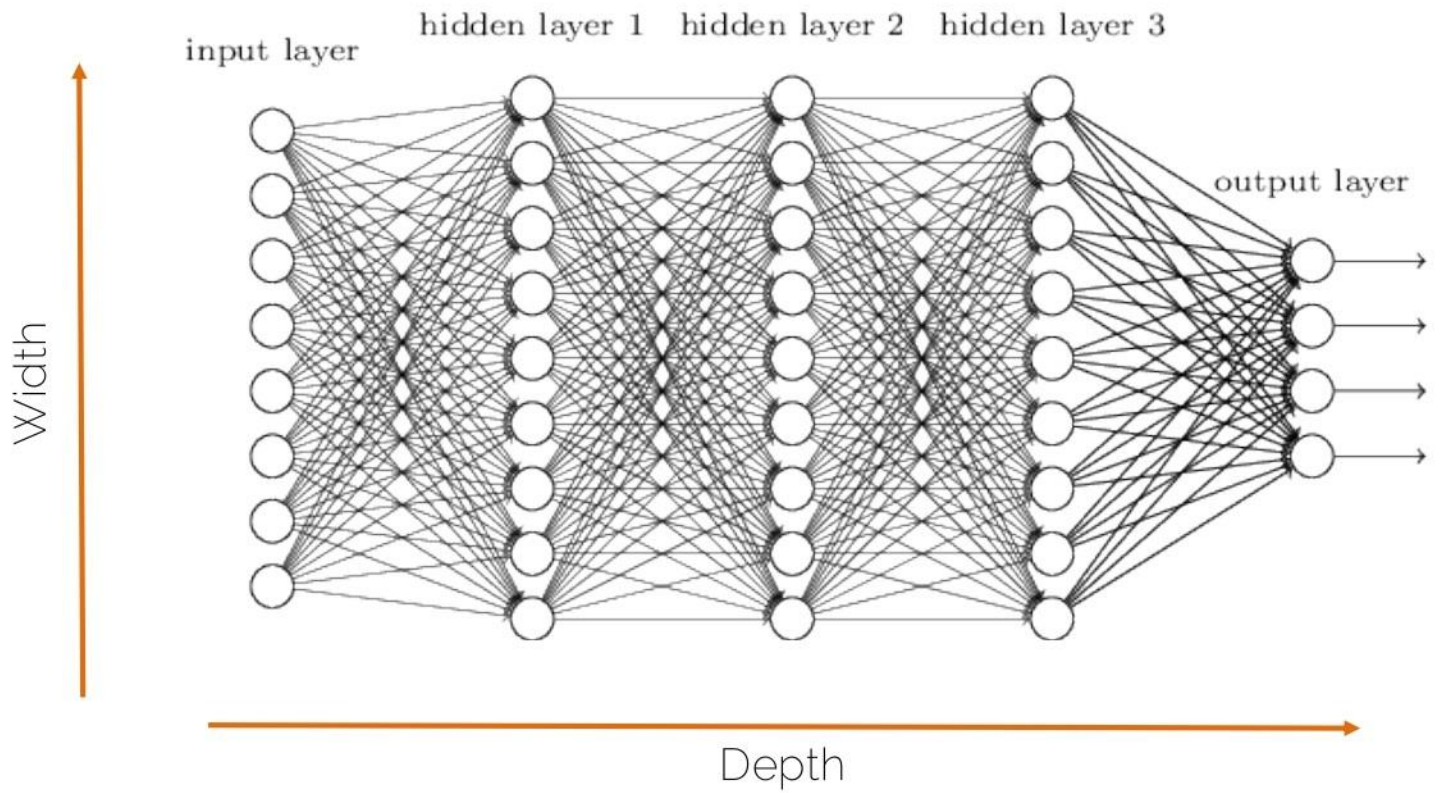
Number of elements in the batch

Number of channels

[Wu and He, ECCV'18] Group Normalization

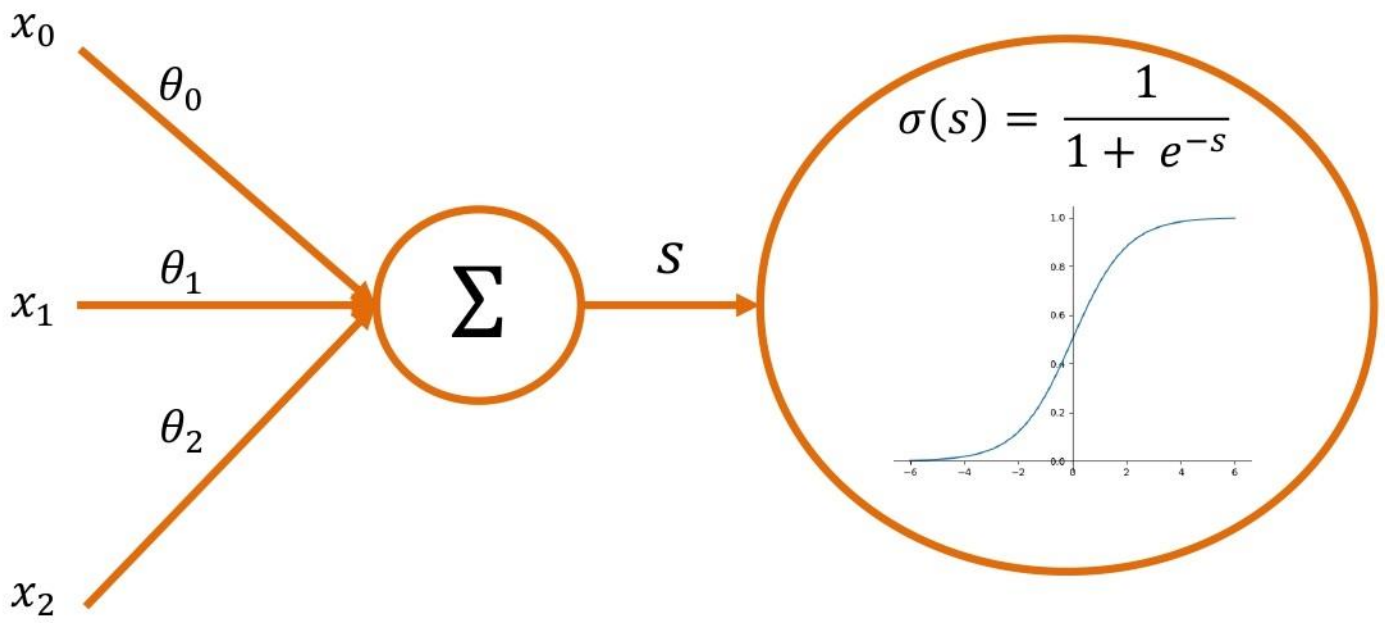
What We Know

What do we know so far?



What do we know so far?

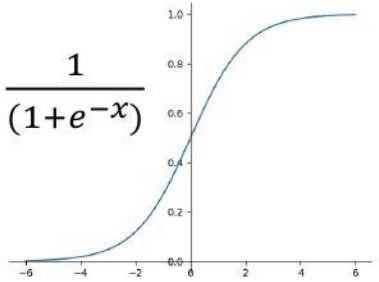
Concept of a 'Neuron'



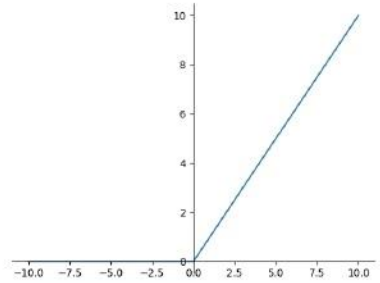
What do we know so far?

Activation Functions (non-linearities)

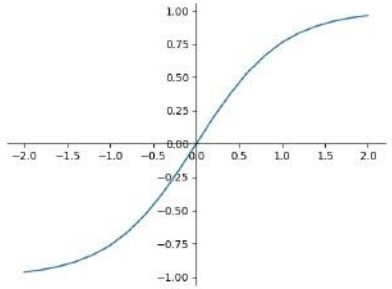
- Sigmoid: $\sigma(x) = \frac{1}{(1+e^{-x})}$



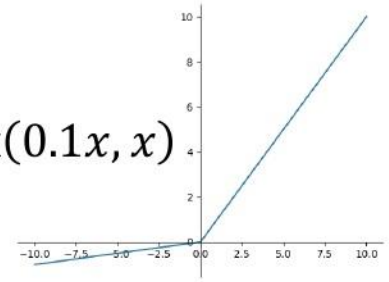
- ReLU: $\max(0, x)$



- TanH: $\tanh(x)$

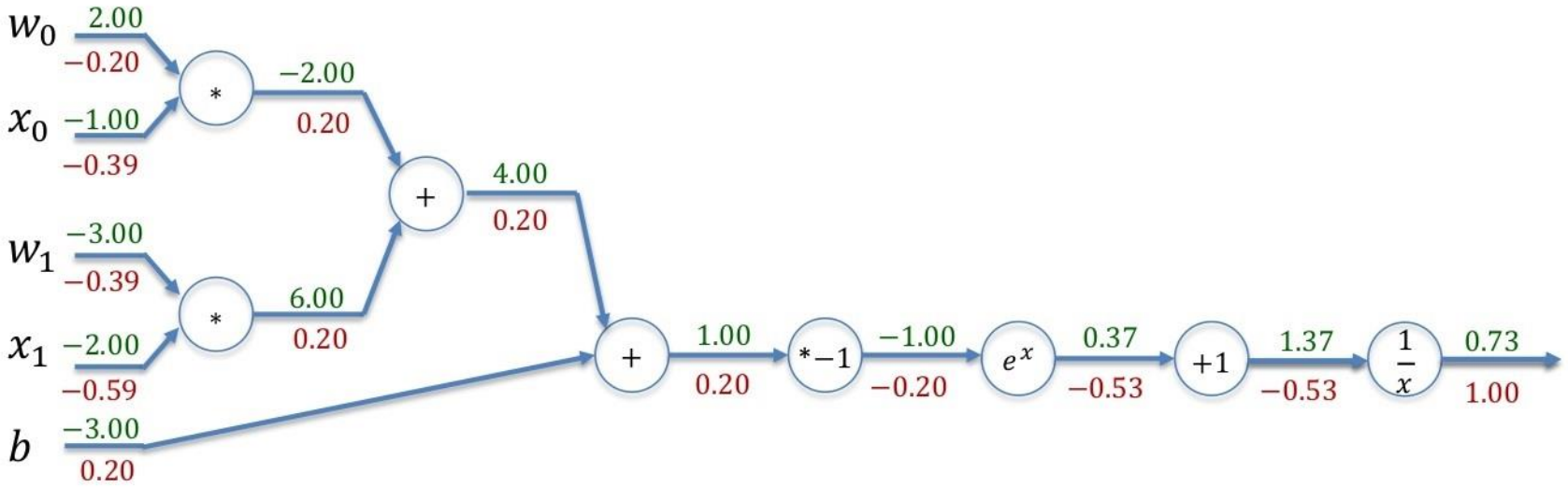


- Leaky ReLU: $\max(0.1x, x)$



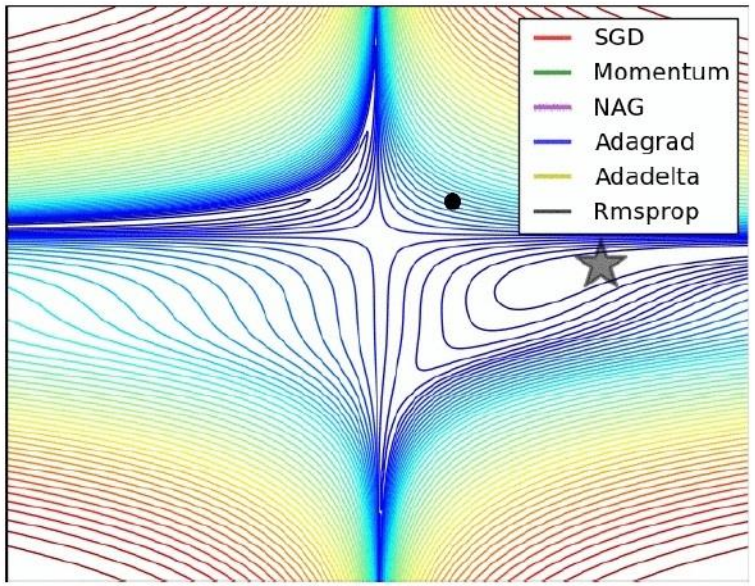
What do we know so far?

Backpropagation



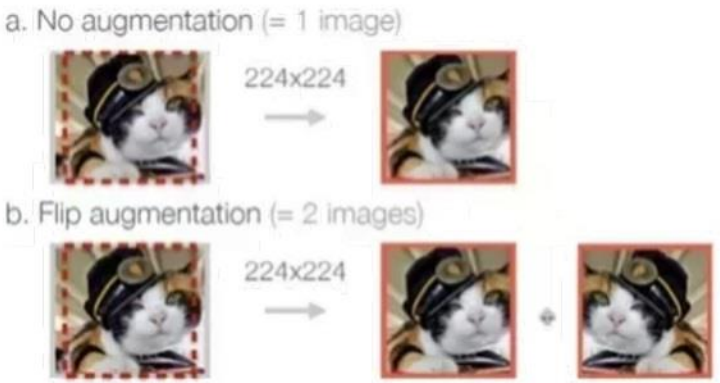
What do we know so far?

SGD Variations (Momentum, etc.)



What do we know so far?

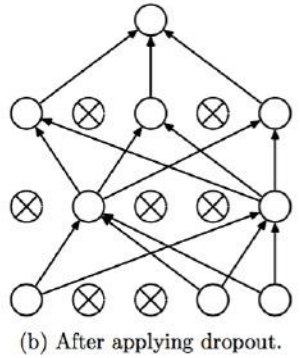
Data Augmentation



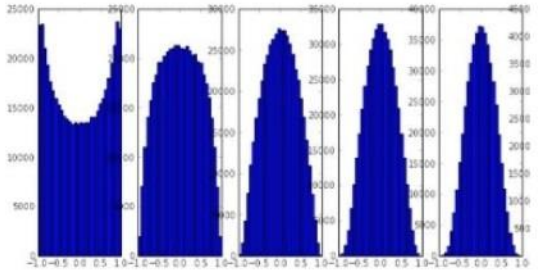
Batch-Norm

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

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!