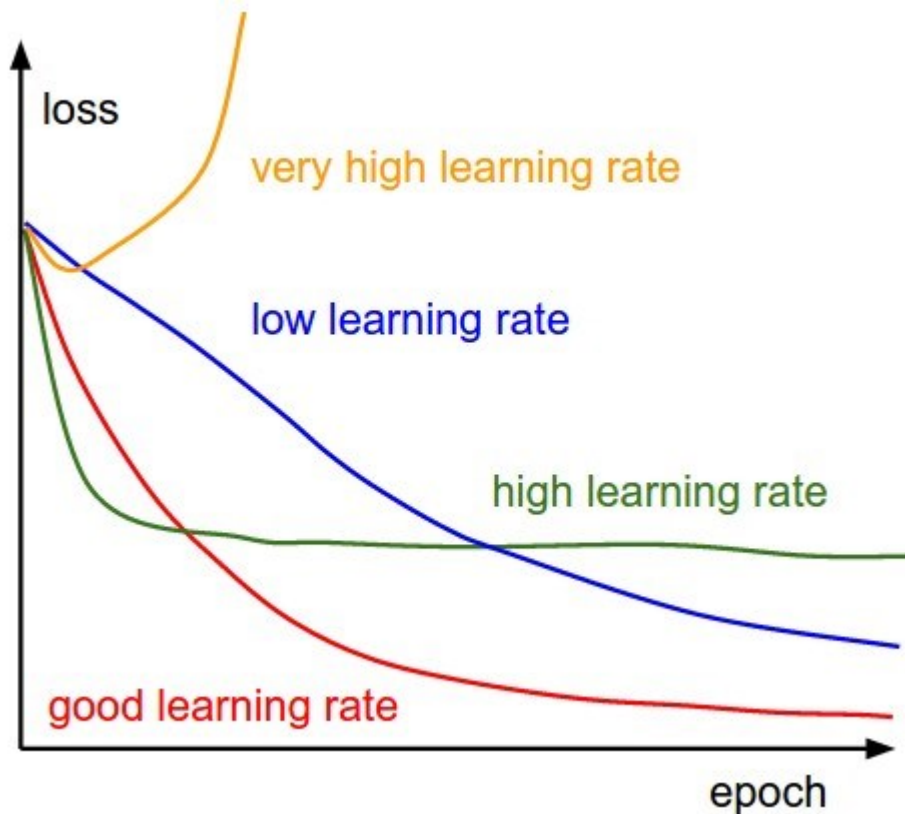


Lecture 6 Recap

Learning Rate: Implications

- What if too high?
- What if too low?



Source: <http://cs231n.github.io/neural-networks-3/>

Training Schedule

Manually specify learning rate for entire training process

- Manually set learning rate every n -epochs
- How?
 - Trial and error (the hard way)
 - Some experience (only generalizes to some degree)

Consider: #epochs, training set size, network size, etc.

Basic Recipe for Training

- Given dataset with ground truth labels
 - $\{\mathbf{x}_i, \mathbf{y}_i\}$
 - \mathbf{x}_i is the i^{th} training image, with label \mathbf{y}_i
 - Often $\text{dim}(\mathbf{X}) \gg \text{dim}(\mathbf{y})$ (e.g., for classification)
 - i is often in the 100-thousands or millions
 - Take network \mathbf{f} and its parameters \mathbf{W}, \mathbf{b}
 - Use SGD (or variation) to find optimal parameters \mathbf{W}, \mathbf{b}
 - Gradients from backprop

Basic Recipe for Machine Learning

- Split your data



Example scenario

Ground truth error 1%

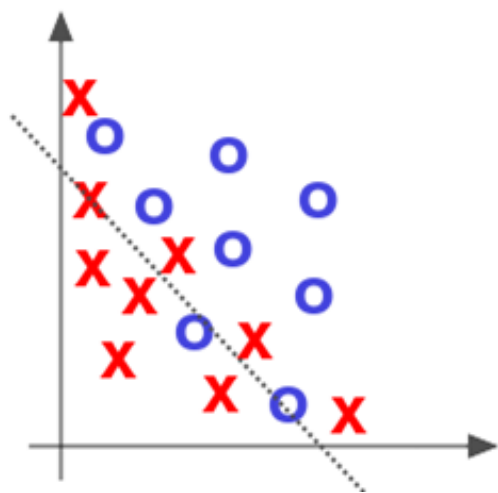
Training set error 5%

Val/test set error 8%

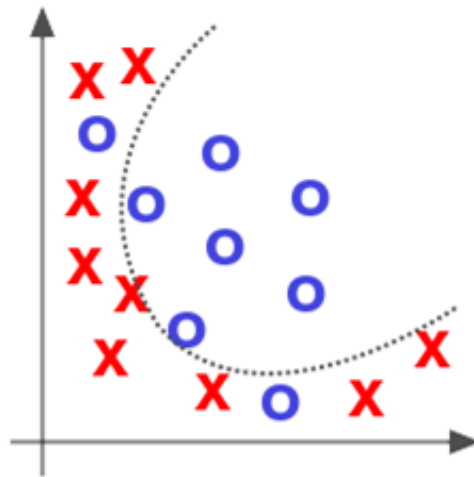
Bias (or underfitting)

Variance
(overfitting)

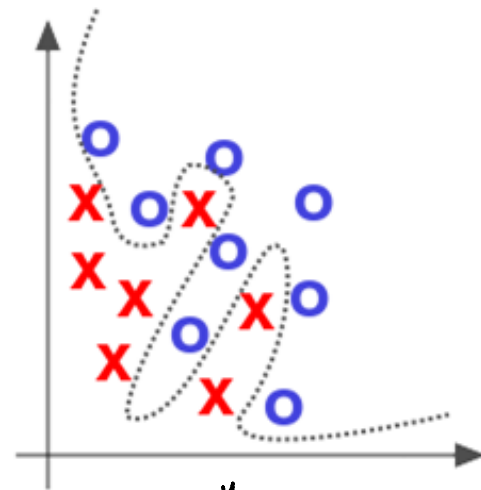
Over and Underfitting



Underfitted



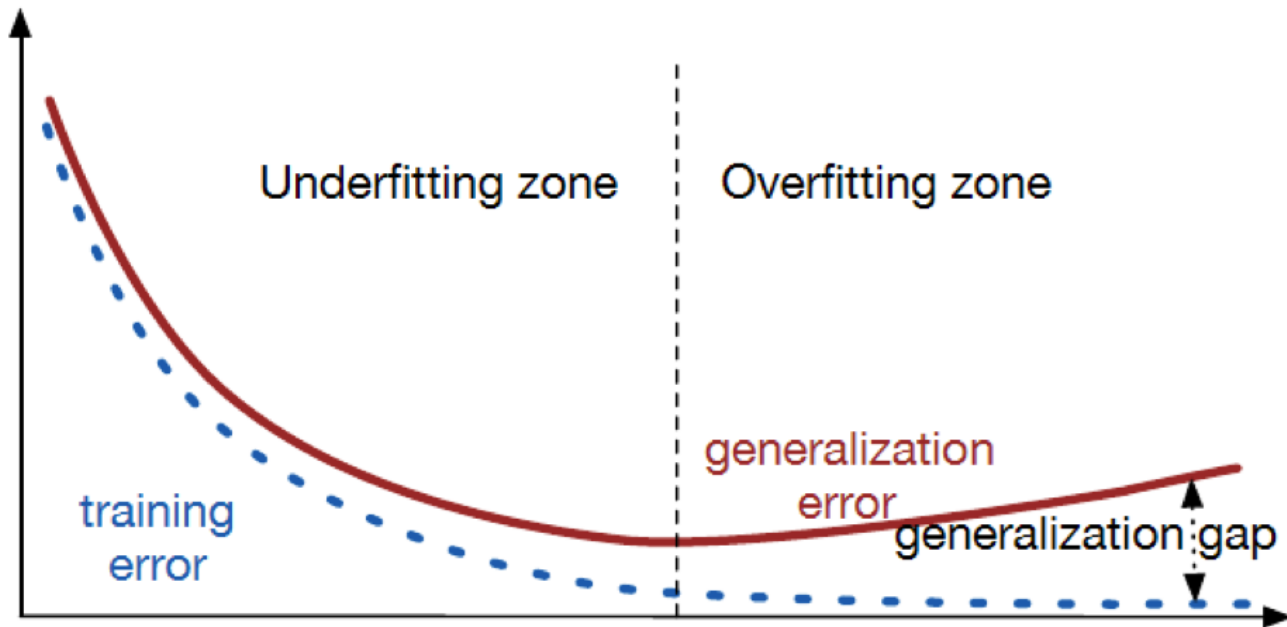
Appropriate



Overfitted

Source: Deep Learning by Adam Gibson, Josh Patterson, O'Reilly Media Inc., 2017

Over and Underfitting



Source:
<https://srdas.github.io/DLBook/ImprovingModelGeneralization.html>

Hyperparameters

- Network architecture (e.g., num layers, #weights)
- Number of iterations
- Learning rate(s) (i.e., solver parameters, decay, etc.)
- Regularization (more later next lecture)
- Batch size
- ...
- Overall: learning setup + optimization = hyperparameters

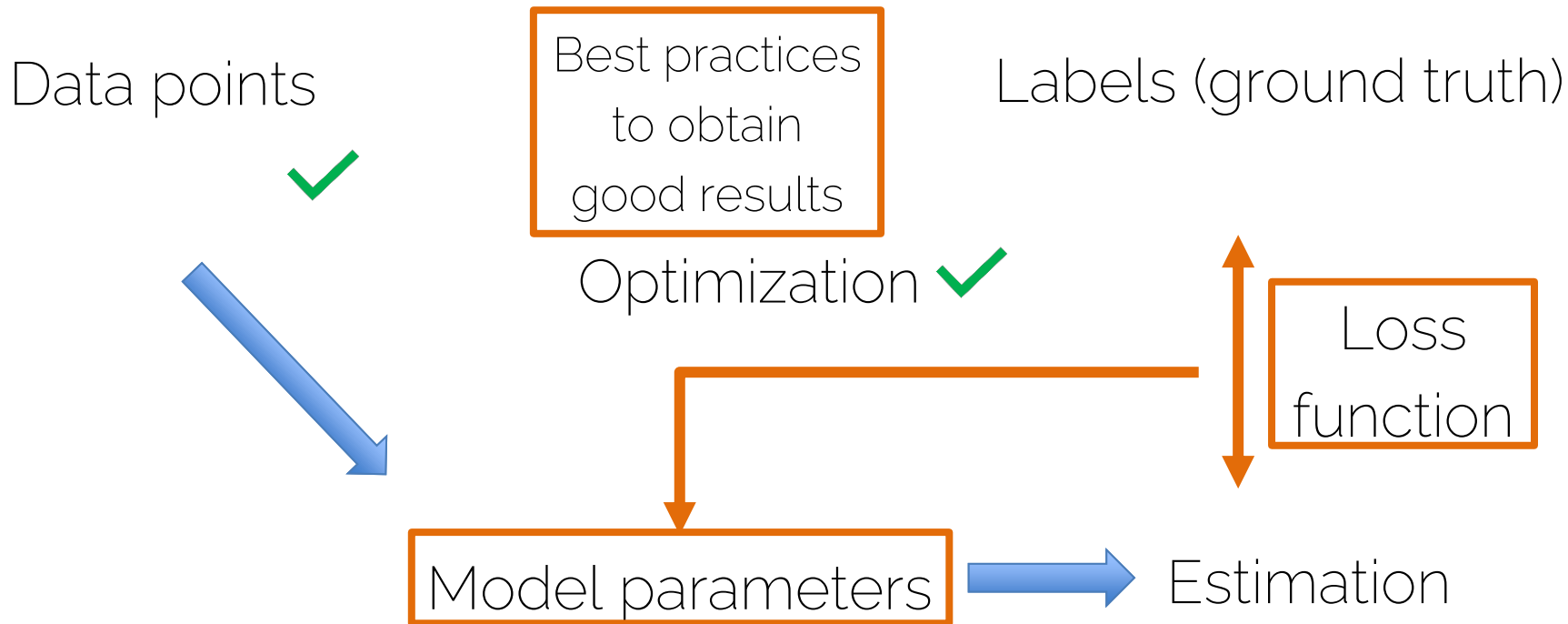
Hyperparameter Tuning

- Methods:
 - Manual search: most common 😊
 - Grid search (structured, for 'real' applications)
 - Define ranges for all parameters spaces and select points
 - Usually pseudo-uniformly distributed
 - Iterate over all possible configurations
 - Random search:
Like grid search but one picks points at random in the predefined ranges
 - Auto-ML:
Bayesian, gradient-based etc

Lecture 7

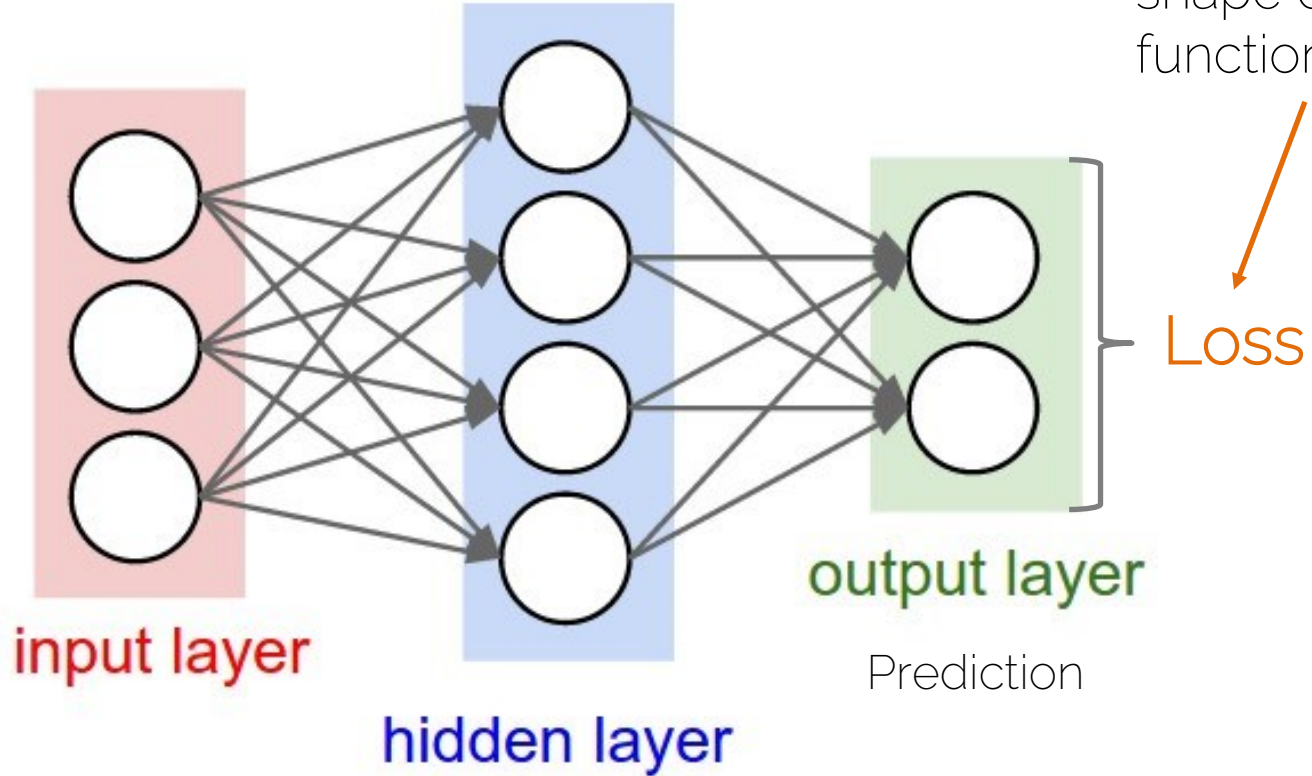
Training NN (part 2)

What we have seen so far



Output and Loss Functions

Neural Networks



Regression Losses

- L2 Loss: $L^2 = \sum_{i=1}^n (y_i - f(\mathbf{x}_i))^2$
 - L1 Loss: $L^1 = \sum_{i=1}^n |y_i - f(\mathbf{x}_i)|$
- training pairs $[\mathbf{x}_i; y_i]$
(input and labels)

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

$f(\mathbf{x}_i)$

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

y_i

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

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

Regression Losses: L2 vs L1

- L2 Loss:

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

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

- L1 Loss:

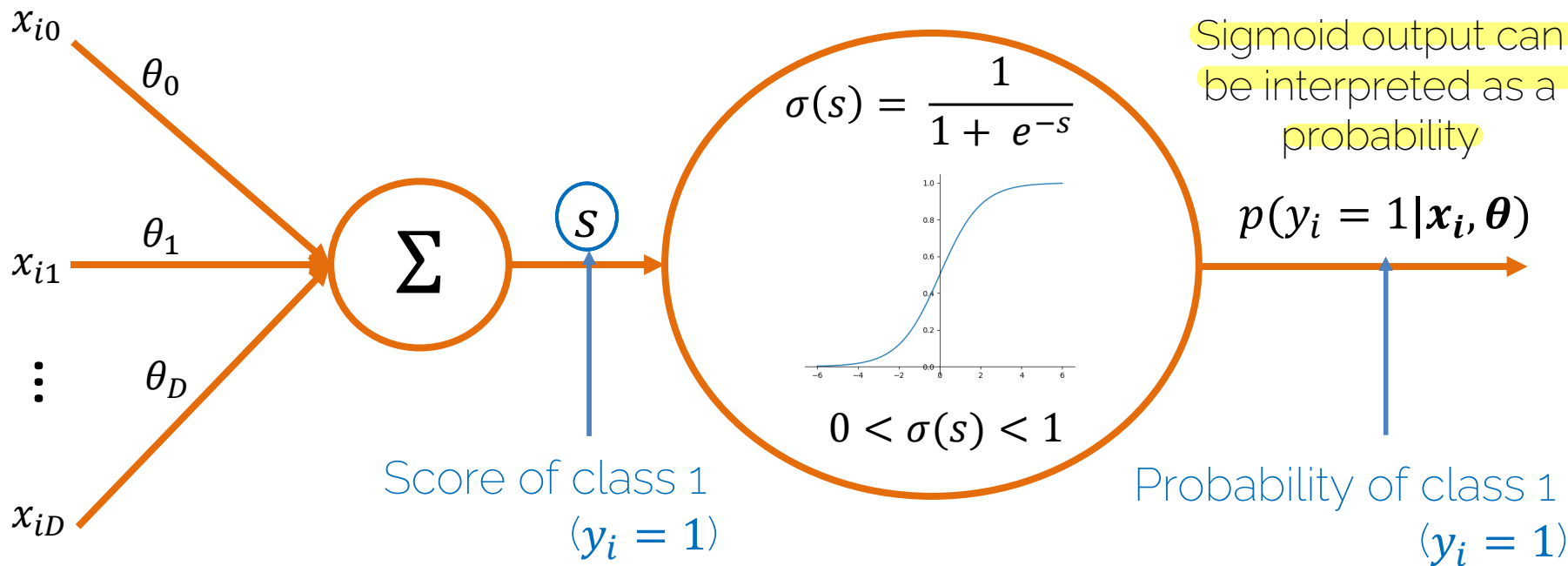
$$L^1 = \sum_{i=1}^n |y_i - f(\mathbf{x}_i)|$$

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

Binary Classification: Sigmoid

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

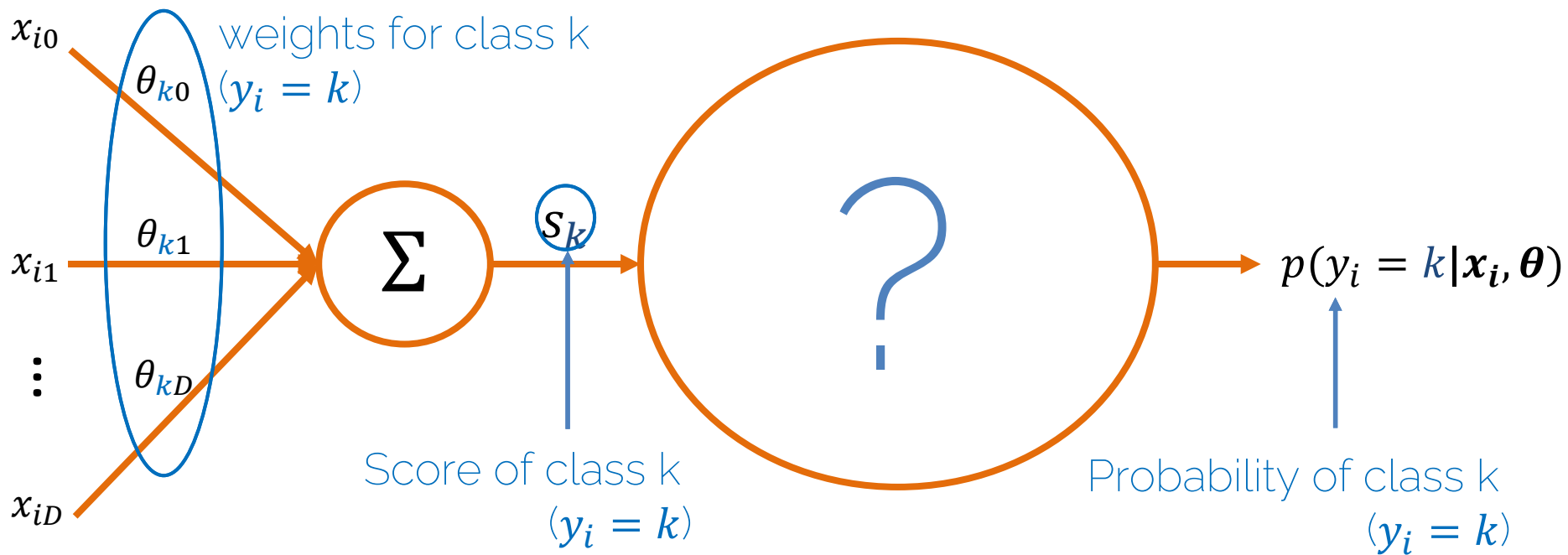
$$p(y_i = 1 | \mathbf{x}_i, \boldsymbol{\theta}) = \sigma(s) = \frac{1}{1 + e^{-\sum_{d=0}^D \theta_d x_{id}}}$$



Multiclass Classification: Softmax

training pairs $[\mathbf{x}_i; y_i]$,

$\mathbf{x}_i \in \mathbb{R}^D, y_i \in \{1, 2 \dots C\}$ (C classes)



Multiclass Classification: Softmax

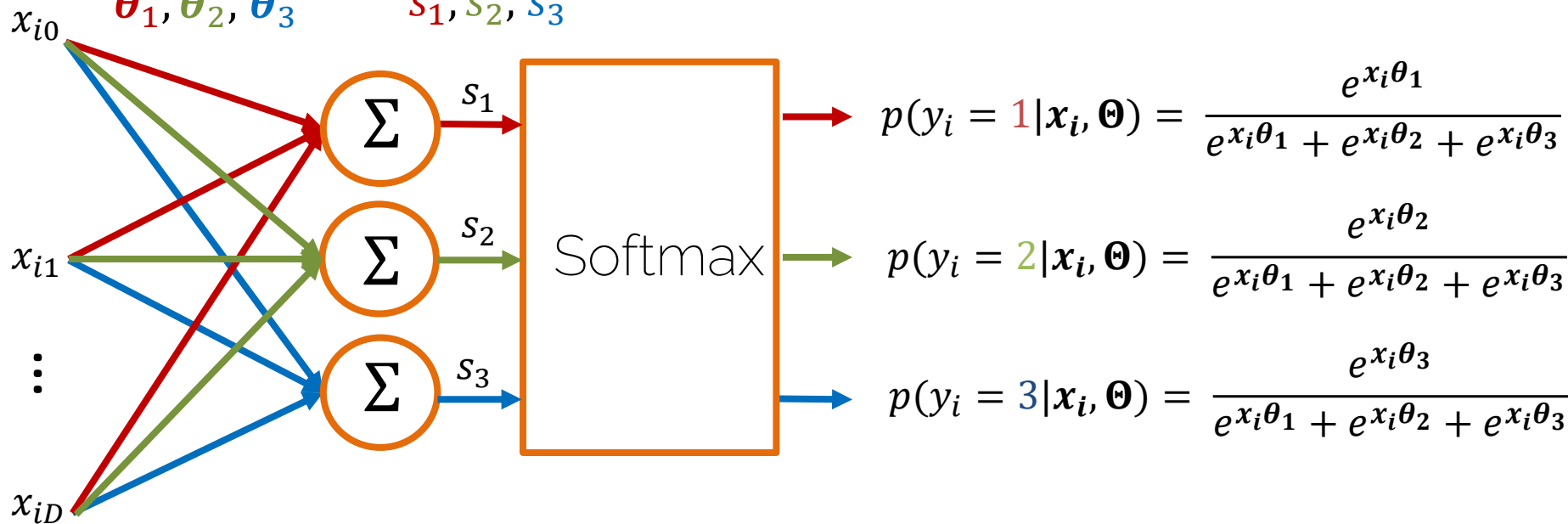
Weights for
each class

$\theta_1, \theta_2, \theta_3$

Scores for
each class

s_1, s_2, s_3

Probabilities
for each class



Multiclass Classification: Softmax

- Softmax

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

Probability of the true class

Exp

normalize

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

Parameters:

$$\Theta = [\boldsymbol{\theta}_1, \boldsymbol{\theta}_2, \dots, \boldsymbol{\theta}_C]$$

C : number of classes

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

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

Try to prove it
by yourself 😊

- Cross-Entropy Loss (Maximum Likelihood Estimate)

$$L_i = -\log(p(y_i|\mathbf{x}_i, \Theta)) = -\log\left(\frac{e^{s_{y_i}}}{\sum_k e^{s_k}}\right)$$

Example: Cross-Entropy Loss

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

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

Given a function with weights θ , training pairs $[x_i; y_i]$ (input and labels)
 $\theta_k = [w_k^{b_k}]$ parameters for each class with C classes



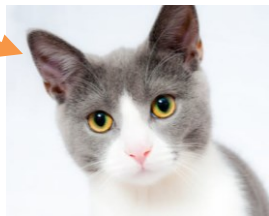
scores	cat	3.2	1.3	2.2
	chair	5.1	4.9	2.5
	car	-1.7	2.0	-3.1

Example: Cross-Entropy Loss

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

Score function $\mathbf{s} = \mathbf{f}(\mathbf{x}_i, \boldsymbol{\theta})$
 e.g., $\mathbf{f}(\mathbf{x}_i, \boldsymbol{\theta}) = [x_{i0}, x_{i2}, \dots, x_{id}] \cdot [\boldsymbol{\theta}_1, \boldsymbol{\theta}_2, \dots, \boldsymbol{\theta}_c]$

Suppose: 3 training examples and 3 classes

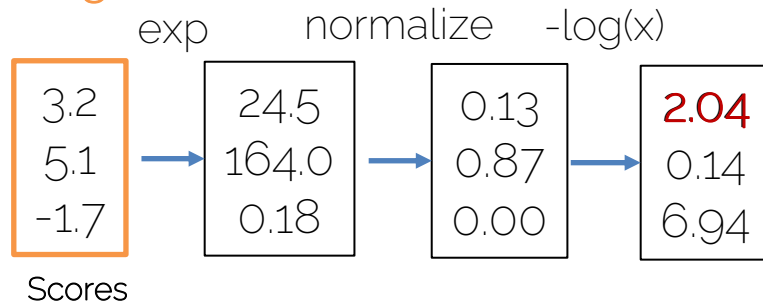


scores	cat	3.2	1.3	2.2
	chair	5.1	4.9	2.5
	car	-1.7	2.0	-3.1

Loss 2.04

Given a function with weights $\boldsymbol{\theta}$, training pairs $[\mathbf{x}_i; \mathbf{y}_i]$ (input and labels)
 $\boldsymbol{\theta}_k = [\mathbf{w}_k^{b_k}]$ parameters for each class with C classes

Image 1

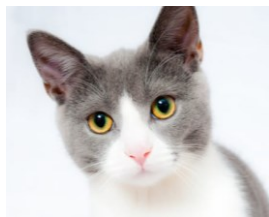


Example: Cross-Entropy Loss

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

Score function $\mathbf{s} = \mathbf{f}(\mathbf{x}_i, \boldsymbol{\theta})$
 e.g., $\mathbf{f}(\mathbf{x}_i, \boldsymbol{\theta}) = [x_{i0}, x_{i2}, \dots, x_{id}] \cdot [\boldsymbol{\theta}_1, \boldsymbol{\theta}_2, \dots, \boldsymbol{\theta}_C]$

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 $\boldsymbol{\theta}$, training pairs $[\mathbf{x}_i; y_i]$ (input and labels)
 $\boldsymbol{\theta}_k = [\mathbf{w}_k^{b_k}]$ parameters for each class with C classes

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

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

$$= 2.76$$

Hinge Loss (SVM Loss)

- Score Function $\mathbf{s} = \mathbf{f}(\mathbf{x}_i, \boldsymbol{\theta})$
 - e.g., $\mathbf{f}(\mathbf{x}_i, \boldsymbol{\theta}) = [x_{i0}, x_{i2}, \dots, x_{id}] \cdot [\boldsymbol{\theta}_1, \boldsymbol{\theta}_2, \dots, \boldsymbol{\theta}_c]$
- 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 $\mathbf{s} = \mathbf{f}(\mathbf{x}_i, \boldsymbol{\theta})$

e.g., $\mathbf{f}(\mathbf{x}_i, \boldsymbol{\theta}) = [x_{i0}, x_{i2}, \dots, x_{id}] \cdot [\boldsymbol{\theta}_1, \boldsymbol{\theta}_2, \dots, \boldsymbol{\theta}_C]$

Suppose: 3 training examples and 3 classes

Given a function with weights $\boldsymbol{\theta}$, training pairs $[\mathbf{x}_i; y_i]$ (input and labels)
 $\boldsymbol{\theta}_k = [\mathbf{w}_k^{b_k}]$ 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

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 $\mathbf{s} = \mathbf{f}(\mathbf{x}_i, \boldsymbol{\theta})$

e.g., $\mathbf{f}(\mathbf{x}_i, \boldsymbol{\theta}) = [x_{i0}, x_{i2}, \dots, x_{id}] \cdot [\boldsymbol{\theta}_1, \boldsymbol{\theta}_2, \dots, \boldsymbol{\theta}_C]$

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 2.9

Given a function with weights $\boldsymbol{\theta}$, training pairs $[\mathbf{x}_i; y_i]$ (input and labels)
 $\boldsymbol{\theta}_k = [\mathbf{w}_k^T; b_k]$ parameters for each class with C classes

$$\begin{aligned}
 &= \max(0, 5.1 - 3.2 + 1) + \max(0, -1.7 - 3.2 + 1) \\
 &= \max(0, 2.9) + \max(0, -3.9) \\
 &= 2.9 + 0 \\
 &= 2.9
 \end{aligned}$$

Example: Hinge Loss (SVM Loss)

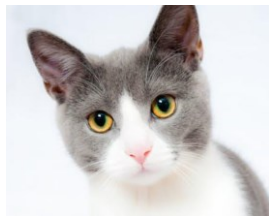
Multiclass SVM loss $L_i = \sum_{k \neq y_i} \max(0, s_k - s_{y_i} + 1)$

Score function $\mathbf{s} = \mathbf{f}(\mathbf{x}_i, \boldsymbol{\theta})$

e.g., $\mathbf{f}(\mathbf{x}_i, \boldsymbol{\theta}) = [x_{i0}, x_{i2}, \dots, x_{id}] \cdot [\boldsymbol{\theta}_1, \boldsymbol{\theta}_2, \dots, \boldsymbol{\theta}_C]$

Suppose: 3 training examples and 3 classes

Given a function with weights $\boldsymbol{\theta}$, training pairs $[\mathbf{x}_i; y_i]$ (input and labels)
 $\boldsymbol{\theta}_k = [\mathbf{w}_k^{b_k}]$ 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
------	-----	---

$$\begin{aligned}
 L_2 &= \max(0, 1.3 - 4.9 + 1) + \max(0, 2.0 - 4.9 + 1) \\
 &= \max(0, -2.6) + \max(0, -1.9) \\
 &= 0 + 0 = 0
 \end{aligned}$$

↓
 because it is
 already classified
 correctly

Example: Hinge Loss (SVM Loss)

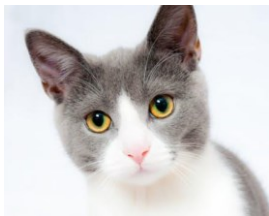
Multiclass SVM loss $L_i = \sum_{k \neq y_i} \max(0, s_k - s_{y_i} + 1)$

Score function $\mathbf{s} = \mathbf{f}(\mathbf{x}_i, \boldsymbol{\theta})$

e.g., $\mathbf{f}(\mathbf{x}_i, \boldsymbol{\theta}) = [x_{i0}, x_{i2}, \dots, x_{id}] \cdot [\boldsymbol{\theta}_1, \boldsymbol{\theta}_2, \dots, \boldsymbol{\theta}_C]$

Suppose: 3 training examples and 3 classes

Given a function with weights $\boldsymbol{\theta}$, training pairs $[\mathbf{x}_i; y_i]$ (input and labels)
 $\boldsymbol{\theta}_k = [\mathbf{w}_k^T]$ 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	12.9
------	-----	---	------

$$\begin{aligned}
 L_3 &= \max(0, 2.2 - (-3.1) + 1) + \max(0, 2.5 - (-3.1) + 1) \\
 &= \max(0, 6.3) + \max(0, 6.6) \\
 &= 6.3 + 6.6 \\
 &= \mathbf{12.9} \\
 &\text{(high value)}
 \end{aligned}$$

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 $\mathbf{s} = \mathbf{f}(\mathbf{x}_i, \boldsymbol{\theta})$

e.g., $\mathbf{f}(\mathbf{x}_i, \boldsymbol{\theta}) = [x_{i0}, x_{i2}, \dots, x_{id}] \cdot [\boldsymbol{\theta}_1, \boldsymbol{\theta}_2, \dots, \boldsymbol{\theta}_C]$

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	2.9	0	12.9
------	-----	---	------

Given a function with weights $\boldsymbol{\theta}$, training pairs $[\mathbf{x}_i; \mathbf{y}_i]$ (input and labels)
 $\boldsymbol{\theta}_k = [\mathbf{w}_k^b]$ parameters for each class with C classes

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

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

$$= 5.3$$

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

Example: Hinge vs Cross-Entropy

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

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

For image \mathbf{x}_i (assume $y_i = 0$):

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

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 \mathbf{x}_i (assume $y_i = 0$):

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

For image \mathbf{x}_i (assume $y_i = 0$):

	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	$s = [5, 10, 10]$	$\max(0, 10 - 5 + 1) + \max(0, 10 - 5 + 1) = 12$	
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 : $L_i = -\log\left(\frac{e^{s_{y_i}}}{\sum_k e^{s_k}}\right)$

For image \mathbf{x}_i (assume $y_i = 0$):

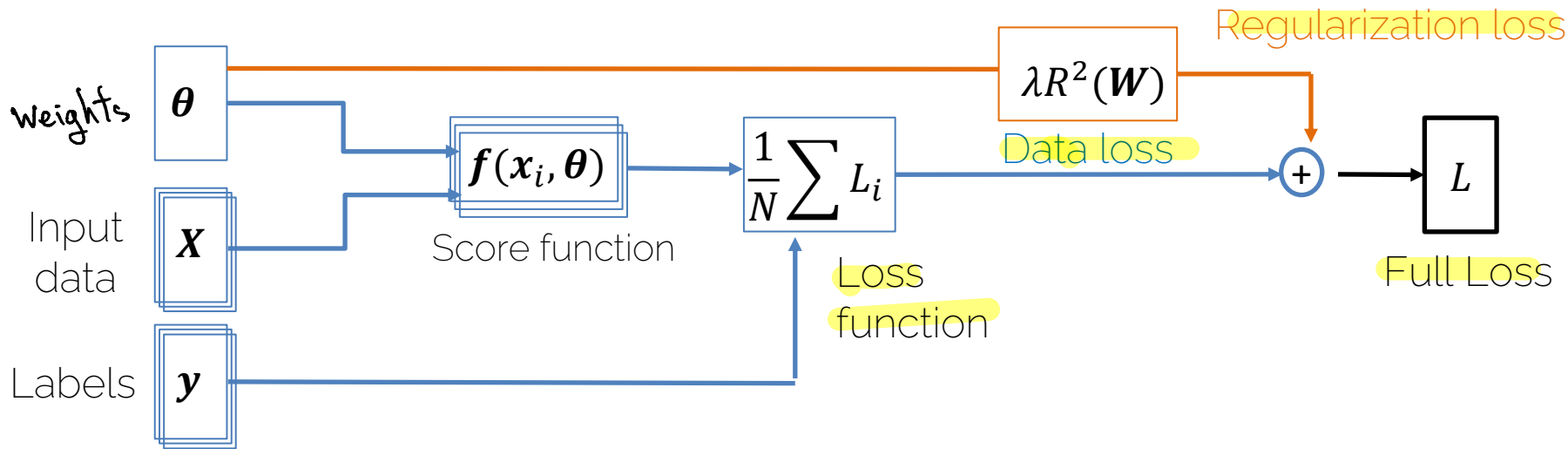
	Scores	Hinge loss:	Cross Entropy loss:
Model 1	$s = [5, -3, 2]$	$\max(0, -3 - 5 + 1) + \max(0, 2 - 5 + 1) = 0$	$-\ln\left(\frac{e^5}{e^5 + e^3 + e^2}\right) = 0.05$
Model 2	$s = [5, 10, 10]$	$\max(0, 10 - 5 + 1) + \max(0, 10 - 5 + 1) = 12$	$-\ln\left(\frac{e^5}{e^5 + e^{10} + e^{10}}\right) = 5.70$
Model 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 θ . (weights are unknowns of optimization problem)

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

Loss in Compute Graph

Given a function with weights θ ,
Training pairs $[\mathbf{x}_i; \mathbf{y}_i]$ (input and labels)

- Score function $\mathbf{s} = \mathbf{f}(\mathbf{x}_i, \theta)$

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

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

- Regularization Loss: e.g., L2-Reg: $R^2(\mathbf{W}) = \sum \mathbf{w}_i^2$

- Full Loss
$$L = \frac{1}{N} \sum_{i=1}^N L_i + \lambda R^2(\mathbf{W})$$

- Full Loss = Data Loss + Reg Loss

Example: Regularization & SVM Loss

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

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

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

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

Example:

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

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

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

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

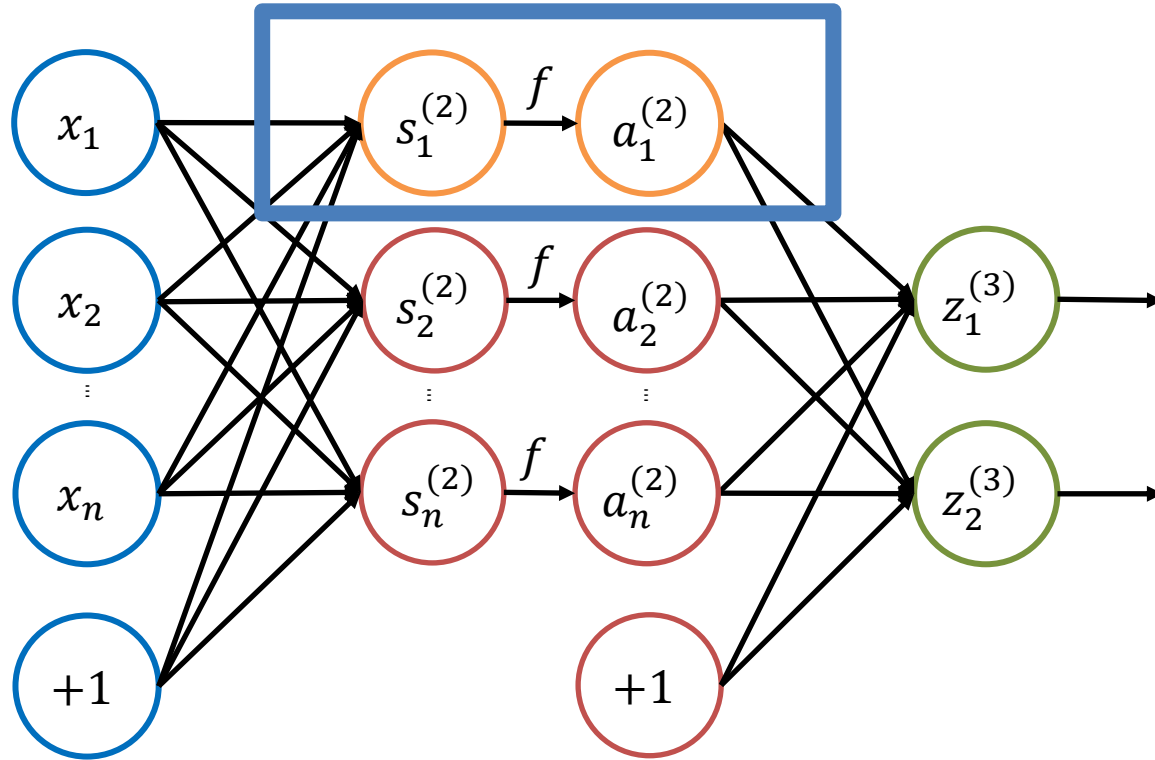
$$R^2(\mathbf{w}_1) = 1$$

$$\begin{aligned} R^2(\mathbf{w}_2) &= 0.25^2 + 0.25^2 + 0.25^2 + 0.25^2 \\ &= 0.25 \end{aligned}$$

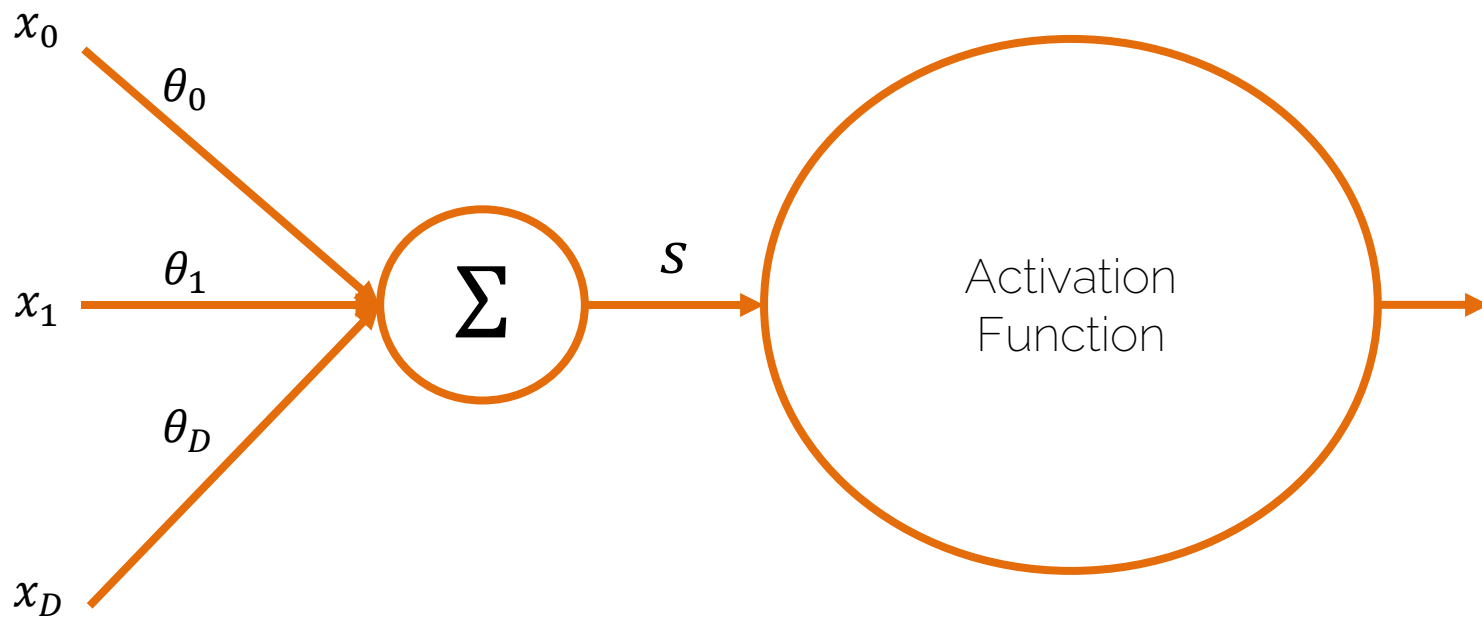
$$R^2(\mathbf{W}) = 1 + 0.25 = 1.25$$

Activation Functions

Neural Networks

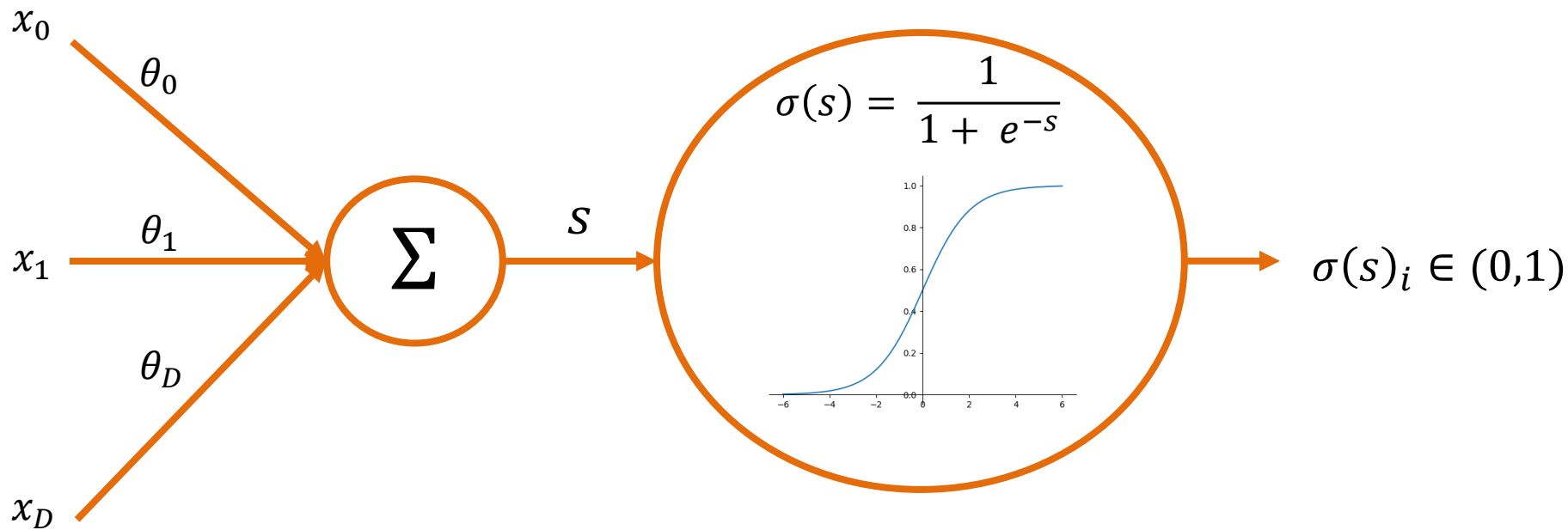


Activation Functions or Hidden Units



Sigmoid Activation

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



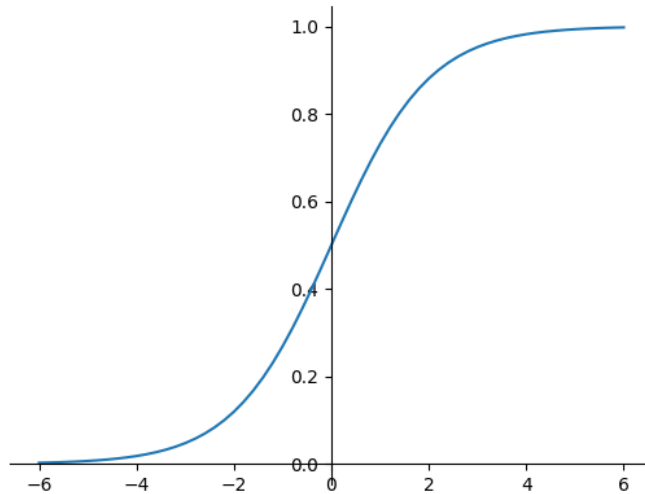
Sigmoid Activation

Forward

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

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

$\uparrow \quad \uparrow$
 $\mathbf{x}^T \quad ?$



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

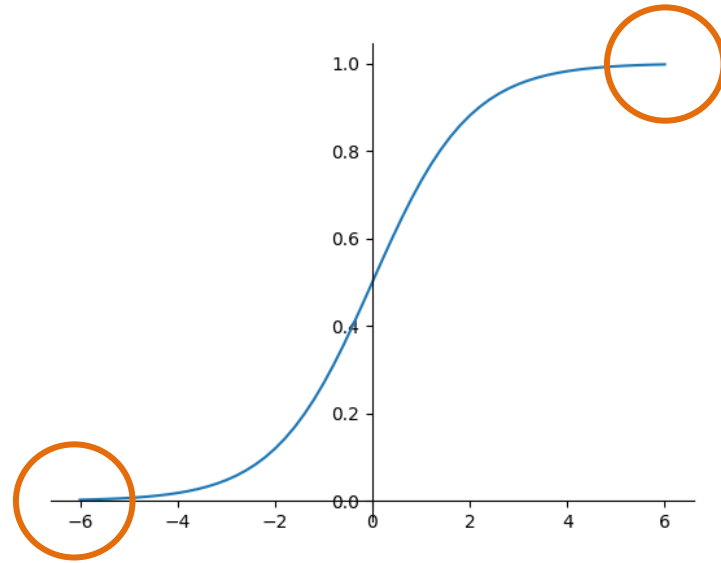
$$\frac{\partial \sigma}{\partial s}$$

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

Sigmoid Activation

Forward

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



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

→ gradient almost 0

× Saturated neurons kill the gradient flow

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

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

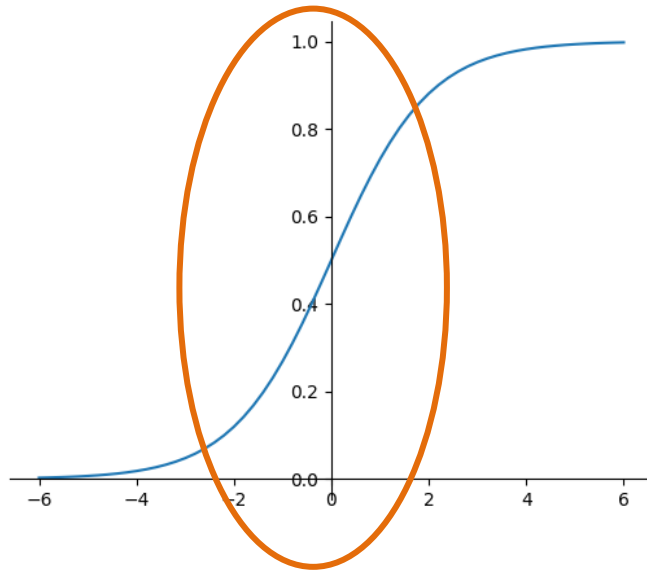
Sigmoid Activation

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

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

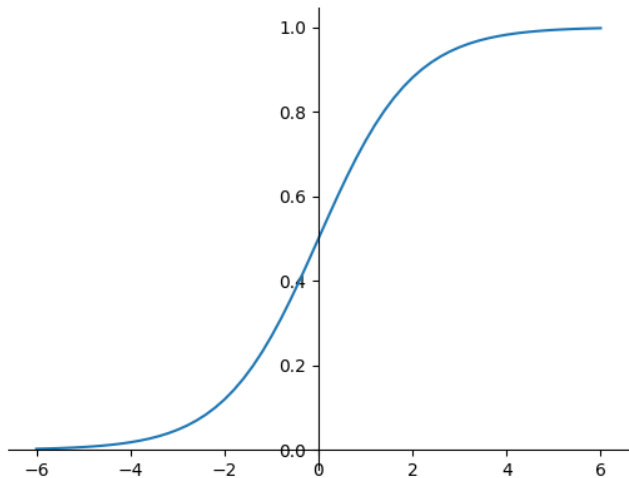


$$\frac{\partial \sigma}{\partial s}$$



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

Sigmoid Activation



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

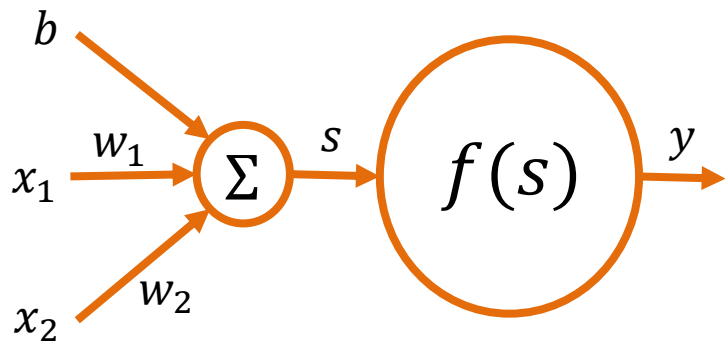
Output is always positive!

- 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:

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

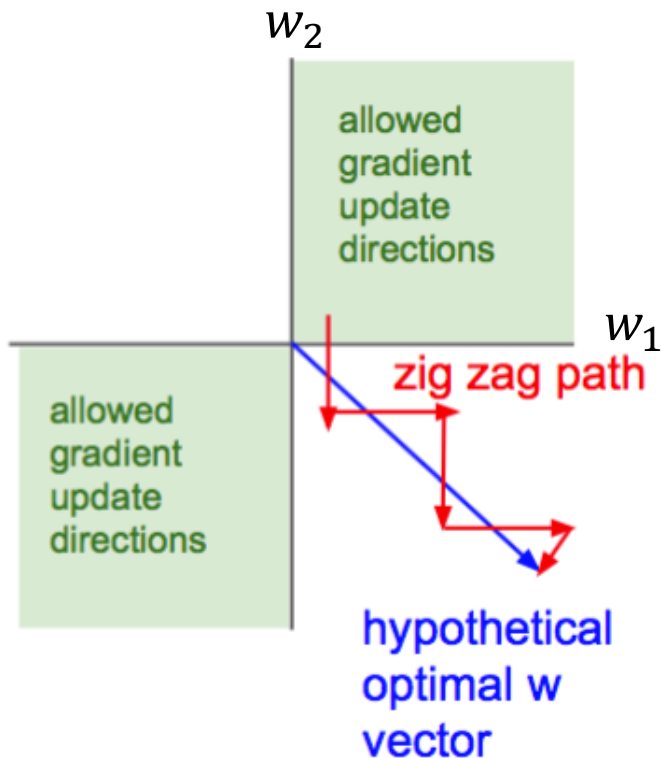
all
either positive
or negative

$$\frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial y} \cdot \frac{\partial y}{\partial s} \cdot \frac{\partial s}{\partial w_1}$$
$$\frac{\partial L}{\partial w_2} = \frac{\partial L}{\partial y} \cdot \frac{\partial y}{\partial s} \cdot \frac{\partial s}{\partial w_2}$$

$x_1 > 0$
 $x_2 > 0$

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

Sigmoid Output not Zero-centered



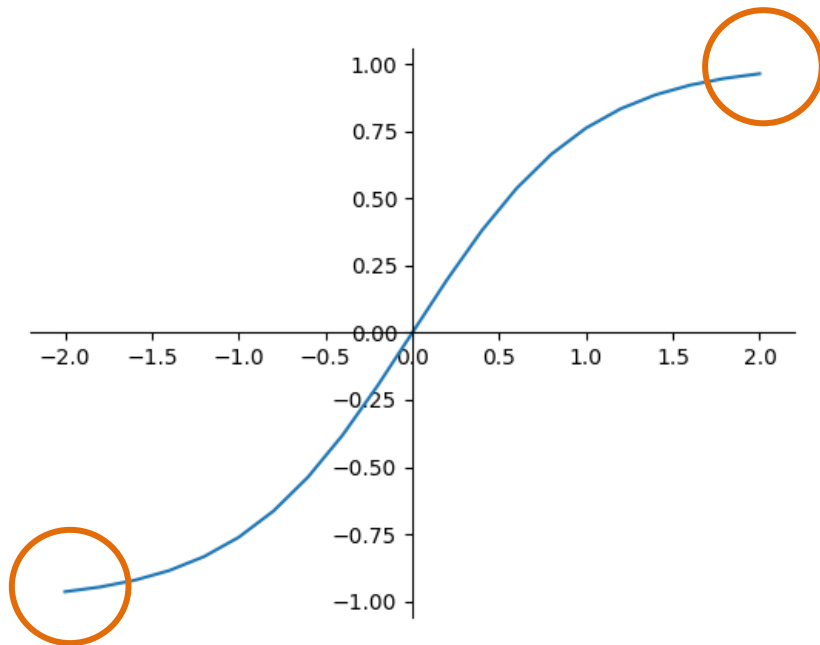
w_1 , w_2 can only be increased or decreased at the same time, which is not good for update.

That is also why you need zero-centered data.

Source :

http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture6.pdf

TanH Activation



Still saturates

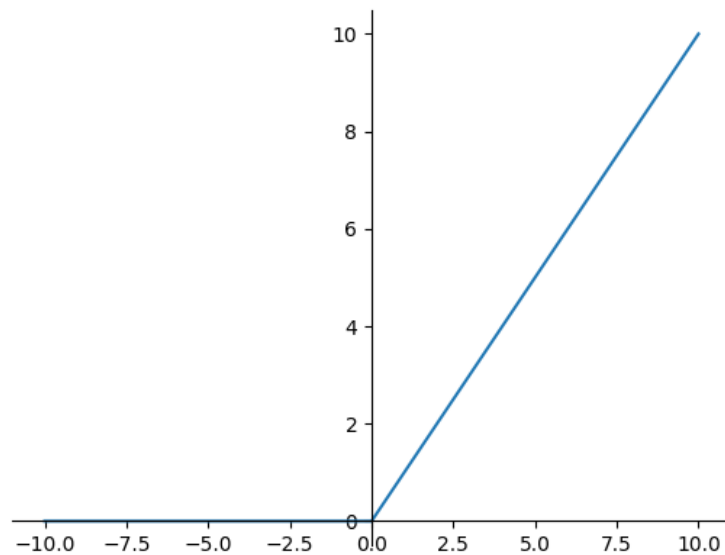


Zero-centered

[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

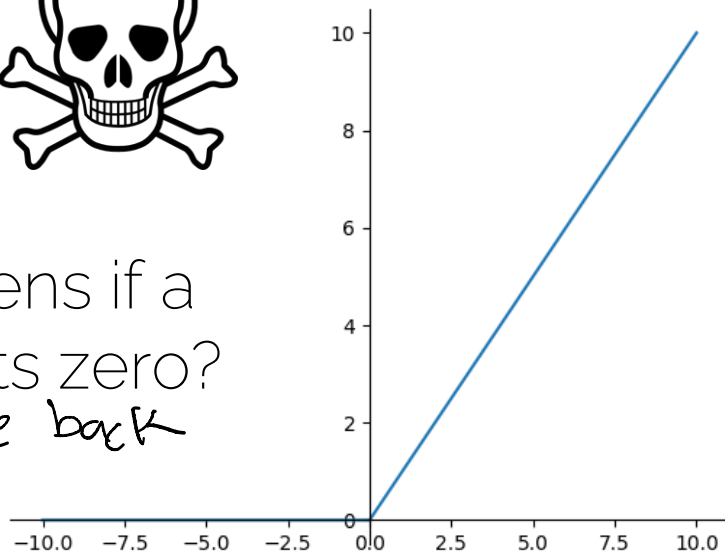
Rectified Linear Units (ReLU)



Dead ReLU



What happens if a
ReLU outputs zero?
no gradients update back



Large and
consistent
gradients



Fast convergence



Does not saturate

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

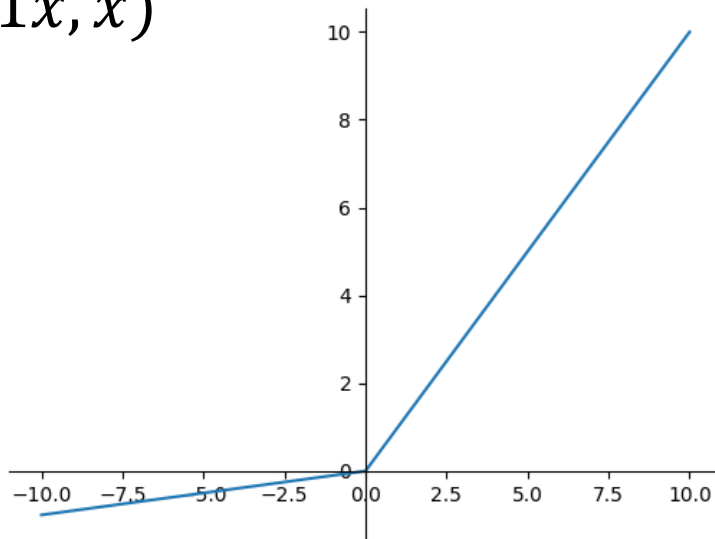
Rectified Linear Units (ReLU)

- Initializing ReLU neurons with slightly positive biases (0.01) makes it likely that they stay active for most inputs

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

Leaky ReLU

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



Does not die

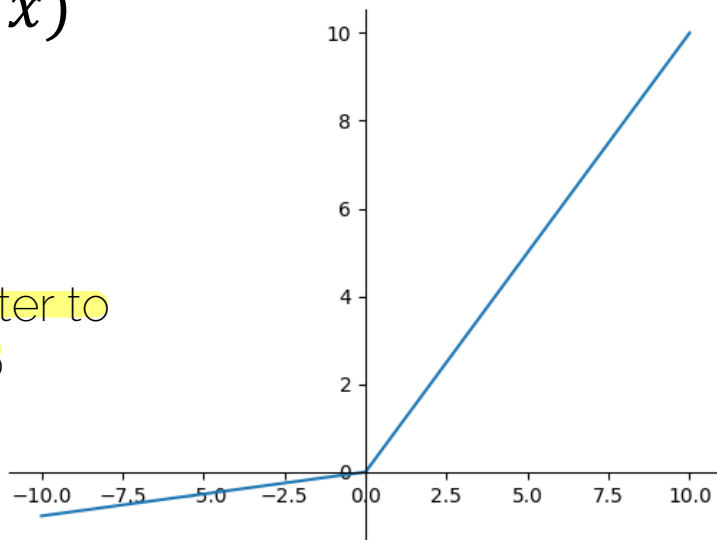
[Mass et al., ICML 2013] Rectifier Nonlinearities Improve Neural Network Acoustic Models

Parametric ReLU

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



One more parameter to
backprop into

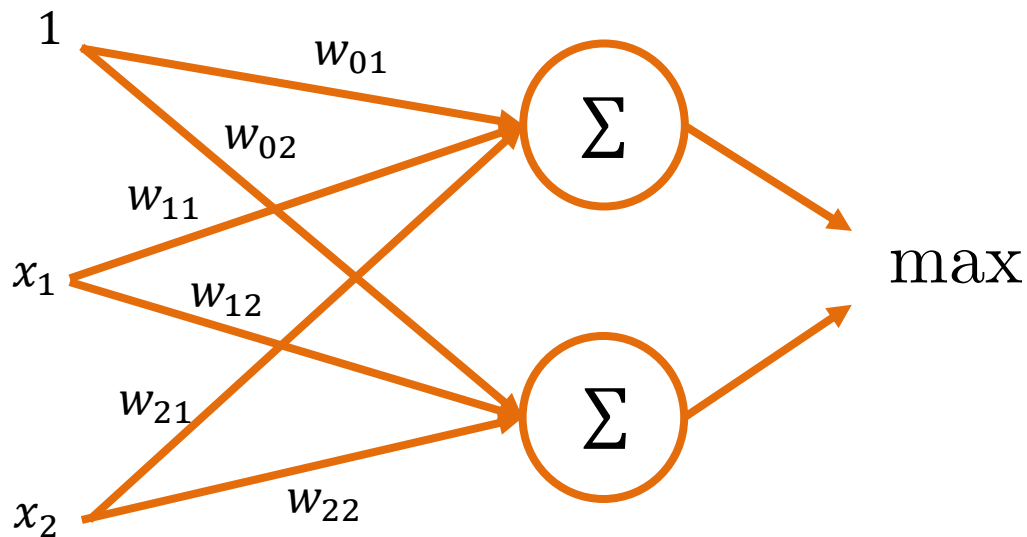


Does not die

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

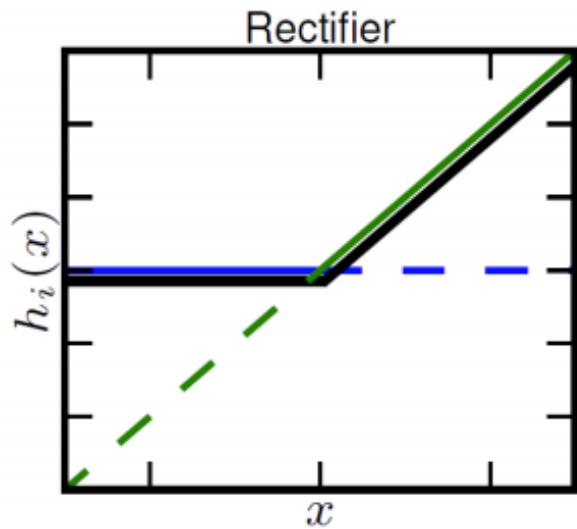
Maxout Units

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

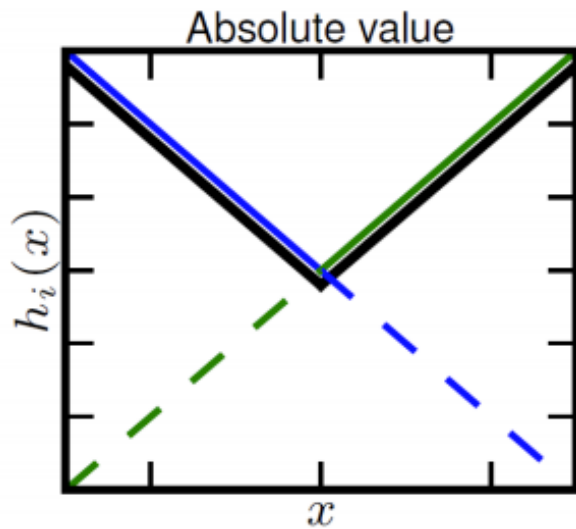


[Goodfellow et al. ICML 2013] Maxout Networks

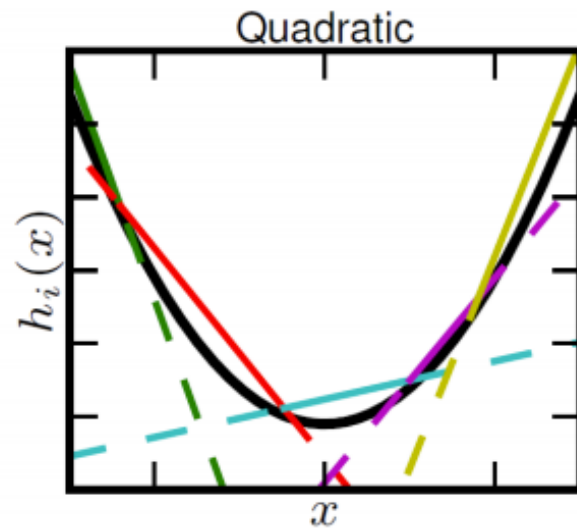
Maxout Units



$k=2$



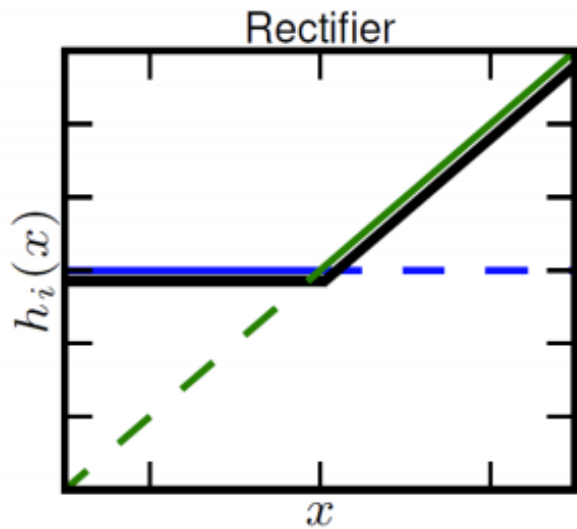
$k=2$



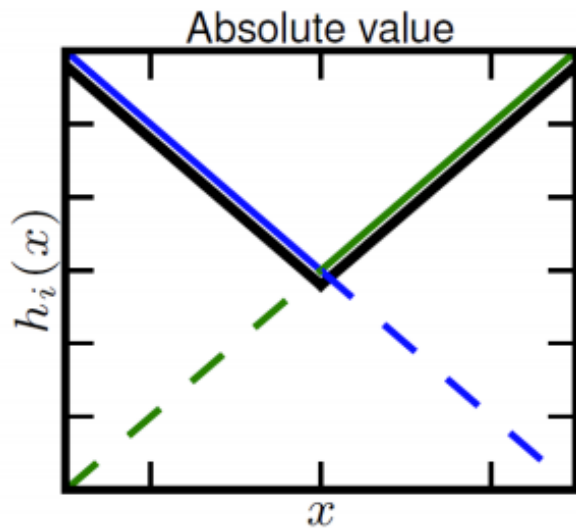
$k=5$

Piecewise linear approximation of
a convex function with N pieces

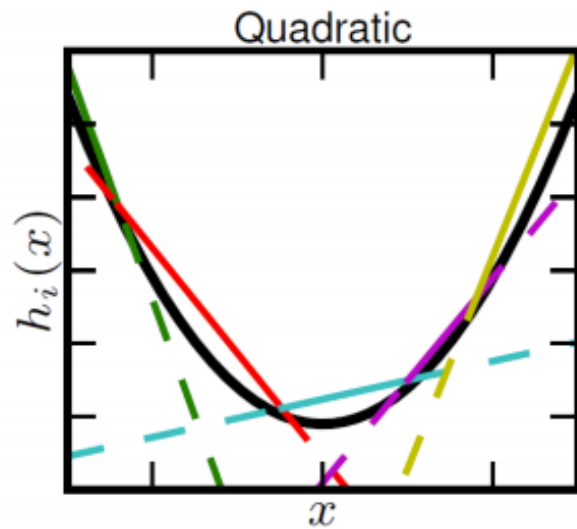
Maxout Units



$k=2$



$k=2$



$k=5$

✓ Generalization
of ReLUs

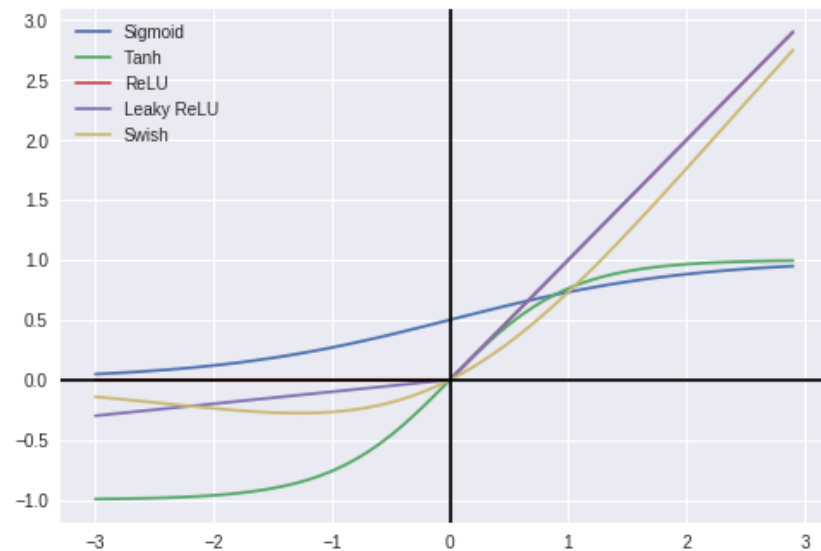
✓ Linear
regimes

✓ 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	$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$	$\{0, 1\}$
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	$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$[0, \infty)$
Leaky ReLU	$f(x) = \begin{cases} 0.01 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$(-\infty, \infty)$
Swish Function	$f(x) = 2x\sigma(\beta x) = \begin{cases} \beta = 0 & \text{for } f(x) = x \\ \beta \rightarrow \infty & \text{for } f(x) = 2\max(0, x) \end{cases}$	$(-\infty, \infty)$

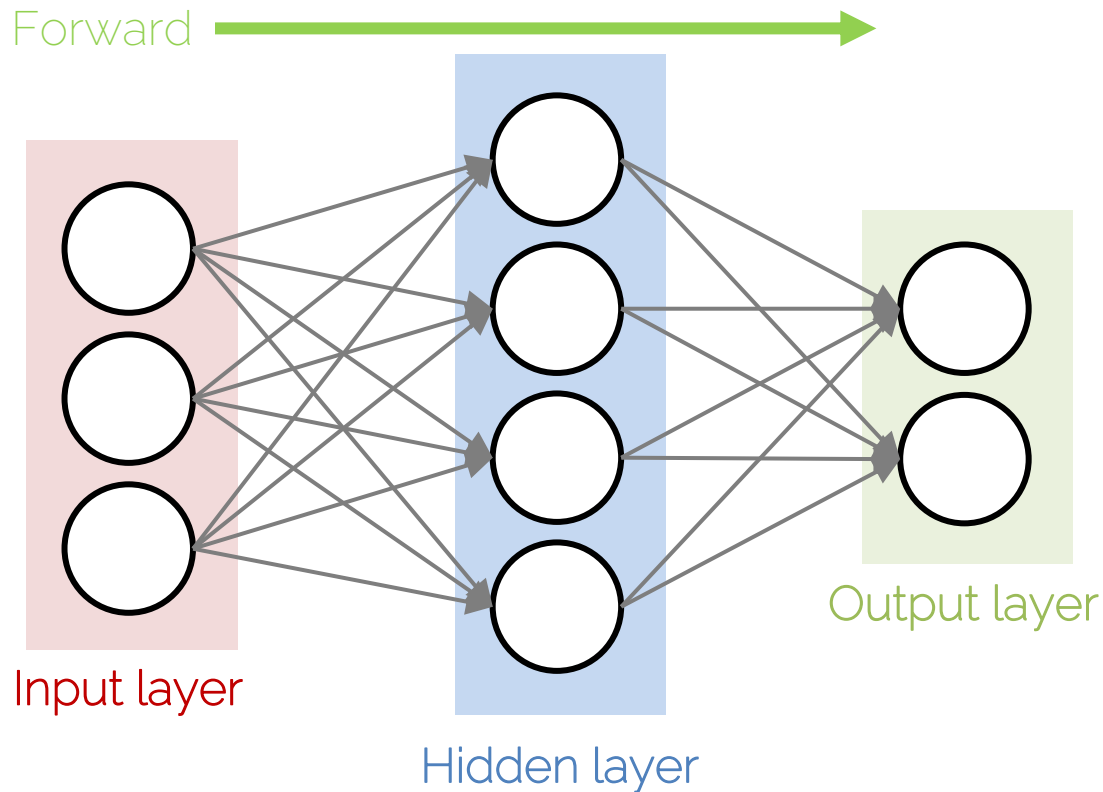
Source : <https://towardsdatascience.com/comparison-of-activation-functions-for-deep-neural-networks-706ac4284c8a>

Quick Guide

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

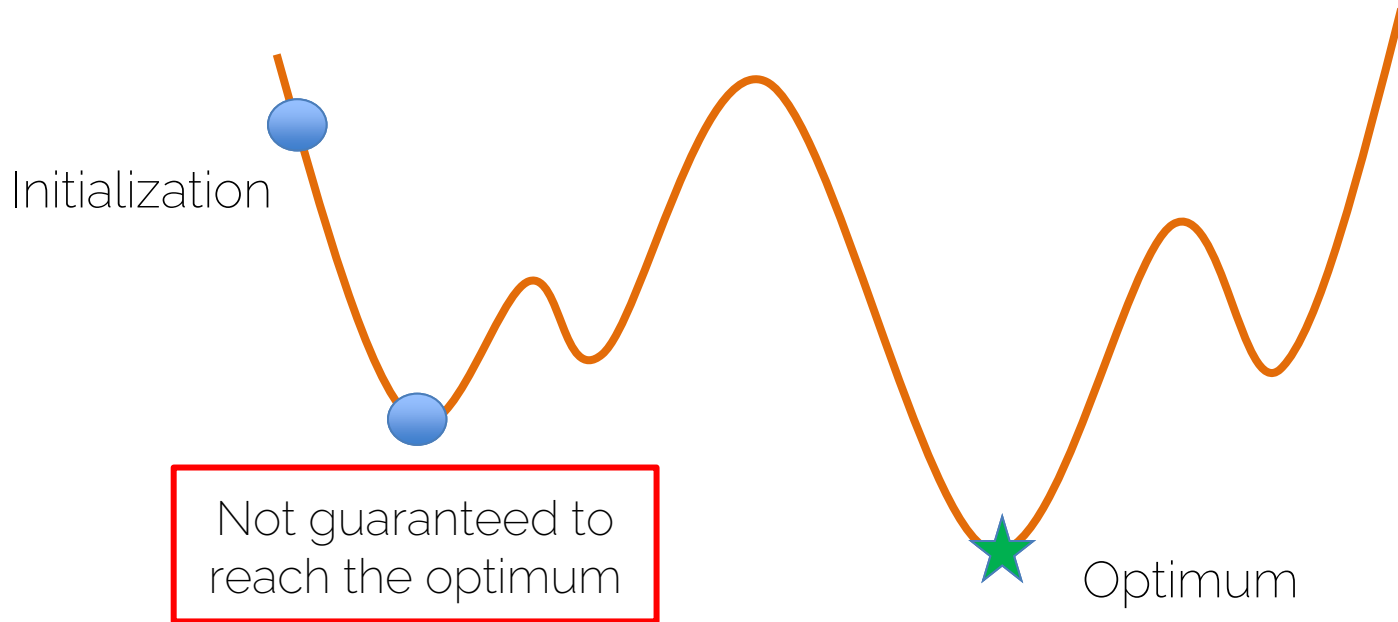
Weight Initialization

How do I start?



Initialization is Extremely Important!

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



How do I start?

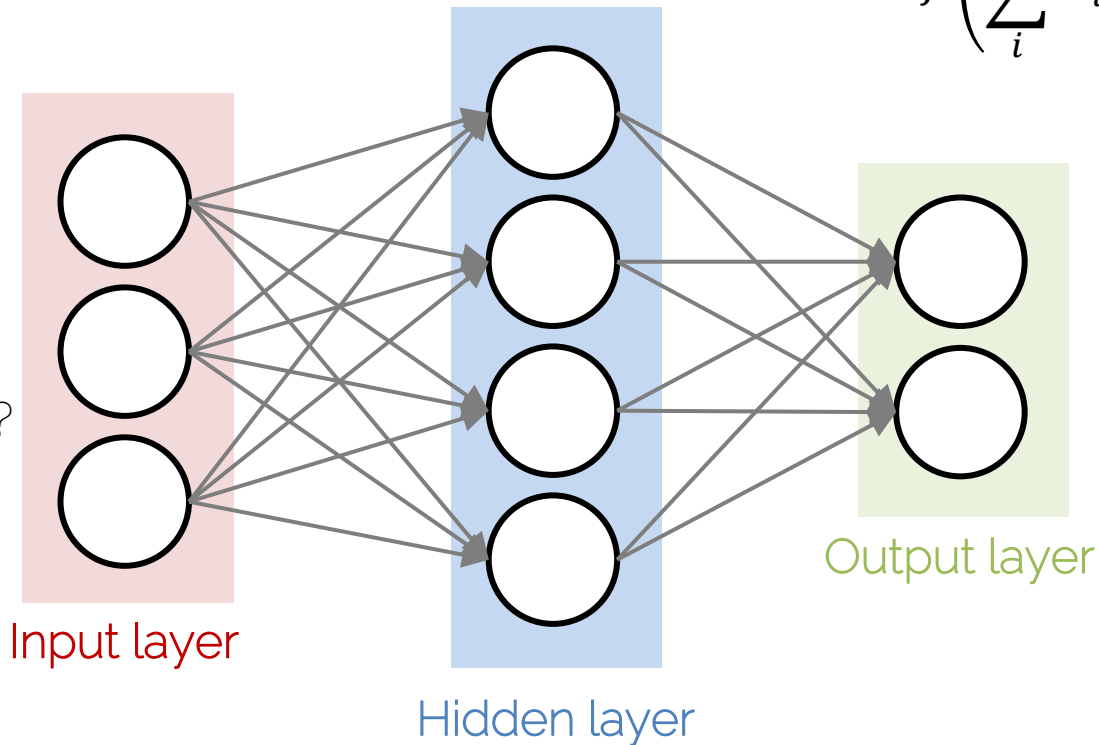
Forward



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

all weights
set to 0
 $w = 0$

What happens
to the gradients?



All Weights Zero

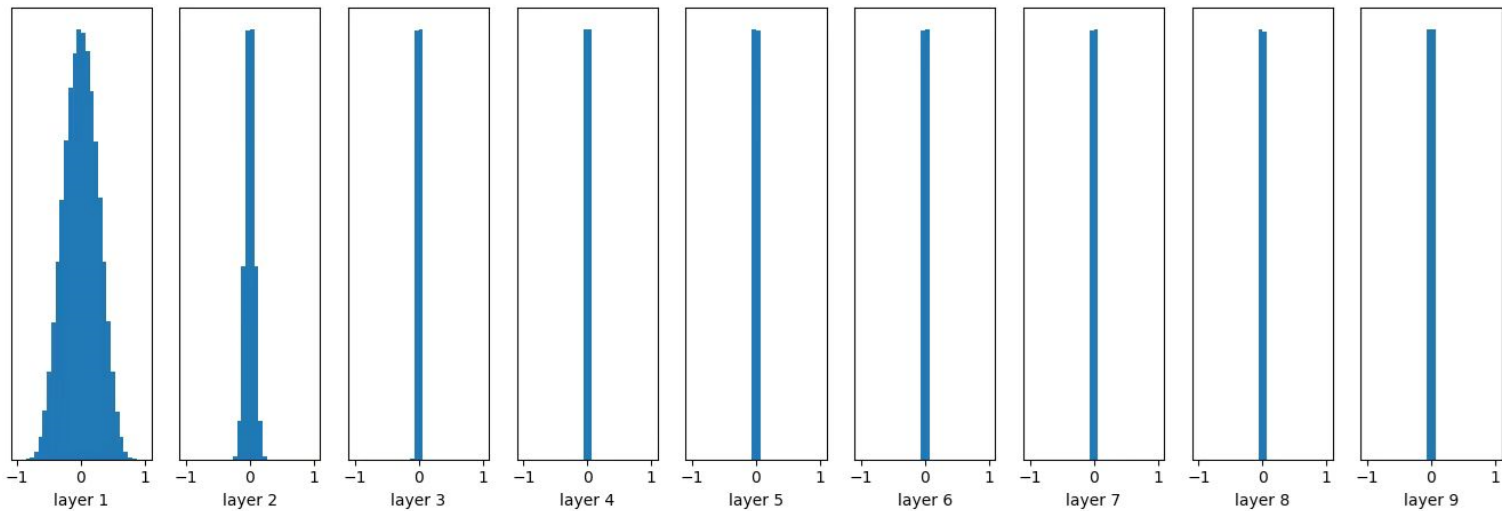
- What happens to the gradients?
- The hidden units are all going to compute the same function, gradients are going to be the same
 - No symmetry breaking

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

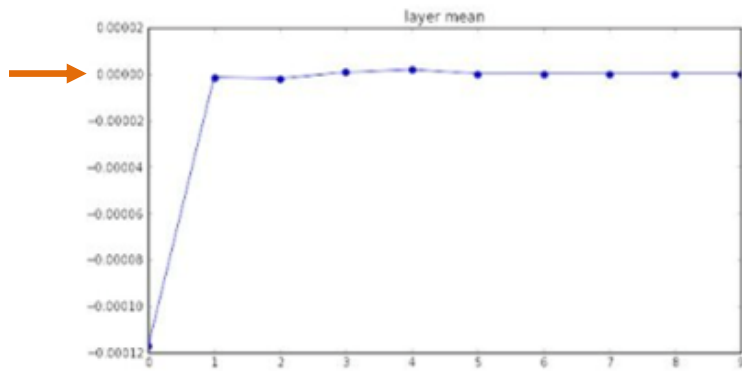


Output goes to zero

Forward

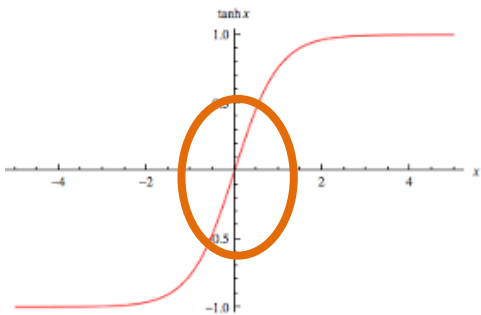


Small Random Numbers



Small w_i^l cause small output for layer l :

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

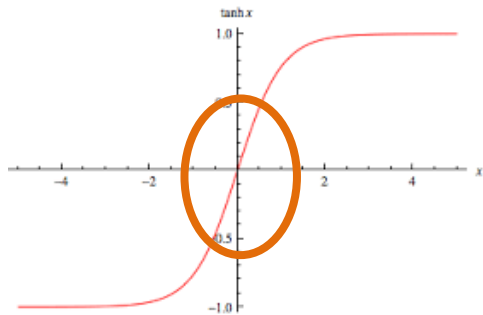


Forward



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

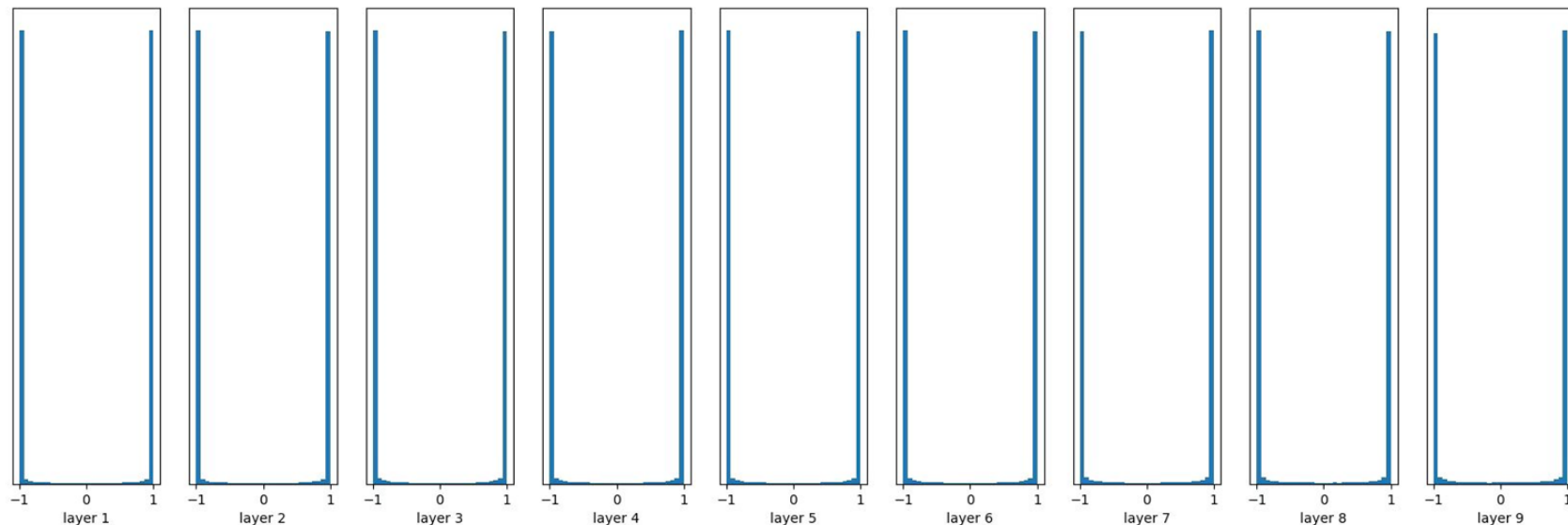


Big Random Numbers

- 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

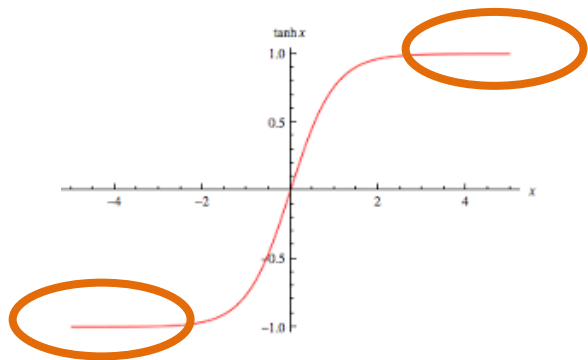
tanh as activation functions



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?

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

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

Tips:

$$E[X^2] = \text{Var}[X] + E[X]^2$$

If X, Y are independent:

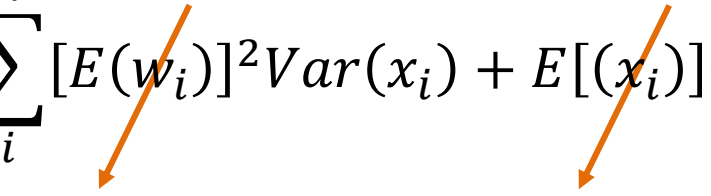
$$\text{Var}[XY] = E[X^2 Y^2] - E[XY]^2$$

$$E[XY] = E[X]E[Y]$$

Xavier Initialization


- Gaussian with zero mean, but what standard deviation?

$$\begin{aligned} \text{Var}(s) &= \text{Var}\left(\sum_i^n w_i x_i\right) = \sum_i^n \text{Var}(w_i x_i) \\ &= \sum_i^n [E(w_i)]^2 \text{Var}(x_i) + E(x_i)^2 \text{Var}(w_i) + \text{Var}(x_i) \text{Var}(w_i) \end{aligned}$$


Zero mean Zero mean

Xavier Initialization

- Gaussian with zero mean, but what standard deviation?

$$\begin{aligned} Var(s) &= Var\left(\sum_i^n w_i x_i\right) = \sum_i^n Var(w_i x_i) \\ &= \sum_i^n [E(w_i)]^2 Var(x_i) + E[(x_i)]^2 Var(w_i) + Var(x_i) Var(w_i) \\ &= \sum_i^n Var(x_i) Var(w_i) = n(Var(w) Var(x)) \end{aligned}$$


Identically distributed
(each random variable has the same
distribution)

Xavier Initialization

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

Goal:

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

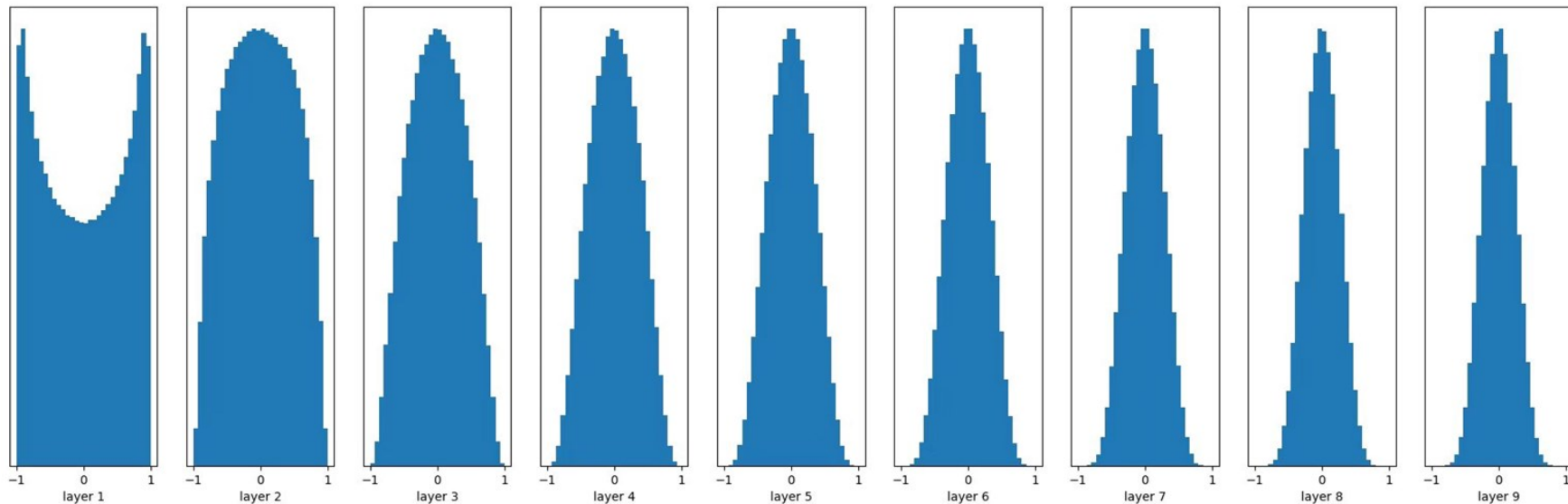
$$\longrightarrow \quad \text{Var}(w) = \frac{1}{n}$$

n : number of input neurons

Xavier Initialization

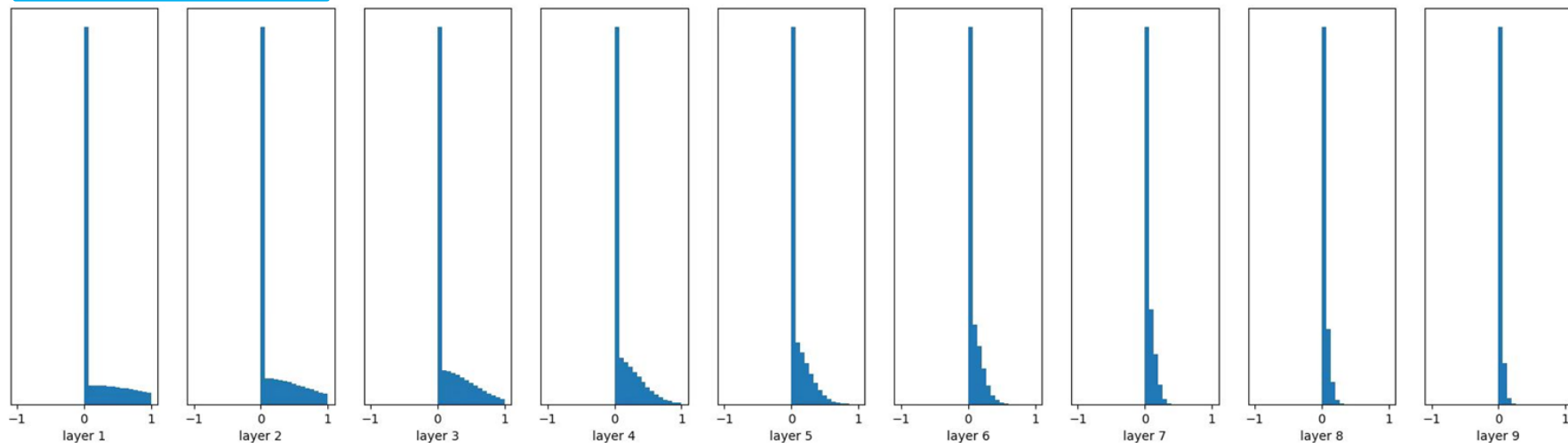
$$\text{Var}(w) = \frac{1}{n}$$

tanh as activation functions



Xavier Initialization with ReLU (Kaiming Initialization)

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

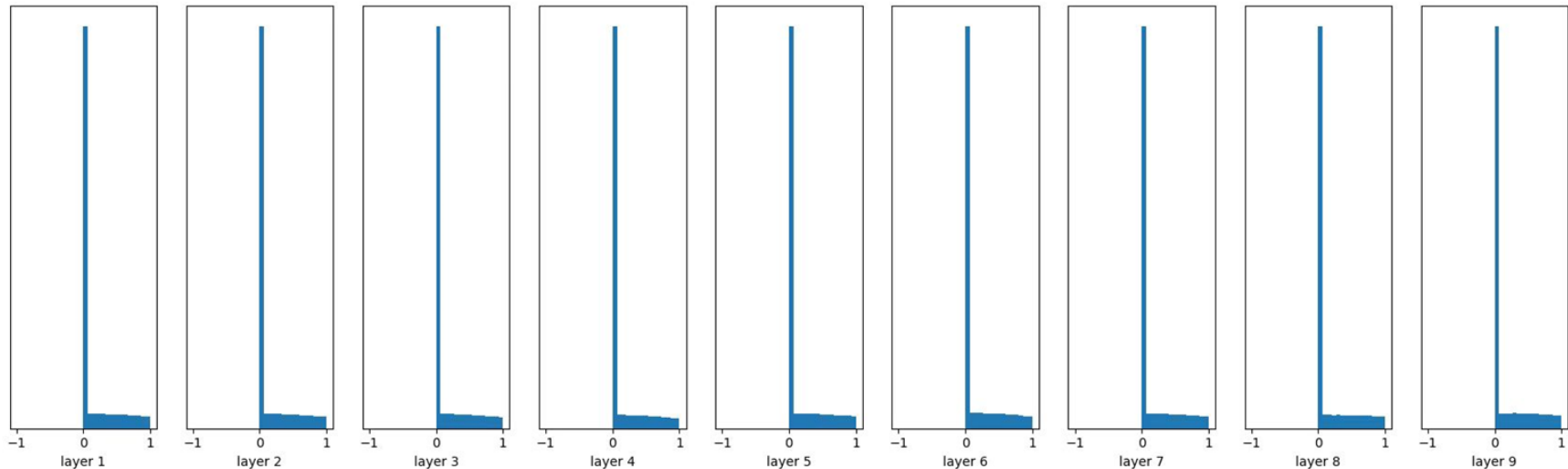


ReLU kills Half of the Data
What's the solution?

When using ReLU, output
close to zero again 😞

Kaiming Initialization with ReLU

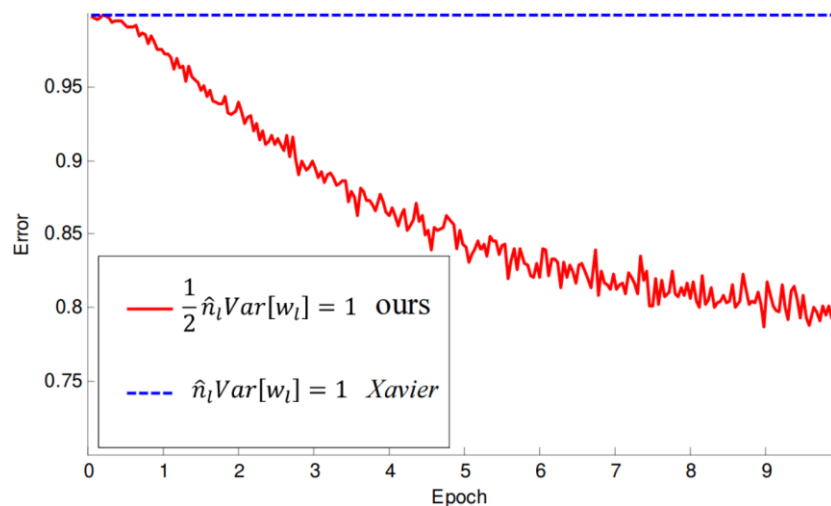
$$\text{Var}(w) = \frac{1}{n/2} = \frac{2}{n}$$



Kaiming Initialization with ReLU

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

It makes a huge difference!



- Use ReLU and Xavier/2 initialization

Summary

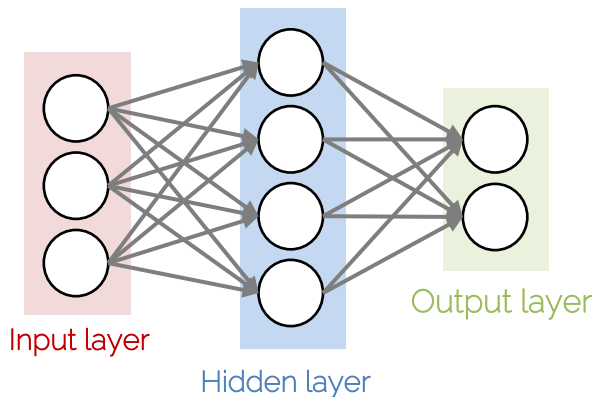


Image Classification	Output Layer	Loss function
Binary Classification	Sigmoid	Binary Cross entropy
Multiclass Classification	Softmax	Cross entropy

Other Losses:

SVM Loss (Hinge Loss), L1/L2-Loss

Initialization of optimization

- How to set weights at beginning

Next Lecture

- Next lecture
 - More about training neural networks: regularization, dropout, data augmentation, batch normalization, etc.
 - Followed by CNNs

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/>