

# CS 559 Machine Learning

## Lecture 11: Neural Networks

Ping Wang

Department of Computer Science

Stevens Institute of Technology



# Today's Lecture

- Summary and review
- Neural networks
- Implementation

# Reviews

- Unsupervised learning:
  - Density estimation
  - K-means
  - Gaussian mixture models
- Supervised learning:
  - Classification
  - Regression

# Supervised Learning

$$\underbrace{w^T x + b}_{\text{score}}$$

	Classification	Regression
Predictor	Sign	Score
Relate to $y$	Margin(score, $y$ )	Residual(score, $y$ )
Loss	Zero-one, hinge, cross-entropy	Squared, absolute
Algorithm	Gradient descent	Gradient descent

# Review: Optimization Problem

Minimize Training Loss

$$\min_{w \in \mathbb{R}^d} \text{TrainLoss}(w) = \frac{1}{N} \sum_{n=1}^N L(x_n, y_n, w)$$

# Review: Optimization Problem

- Gradient descent:

$$w \leftarrow w - \underbrace{\eta}_{\text{Step size}} * \underbrace{\nabla_w \text{TrainLoss}(w)}_{\text{Gradient}}$$

$\nabla_w \text{TrainLoss}(w)$  denotes the gradient of the (average) total training loss with respect to  $w$

- Stochastic Gradient Descent (SGD):

For each  $(x, y) \in D_{\text{train}}$ ,

$$w \leftarrow w - \eta \nabla_w \text{Loss}(x, y, w)$$

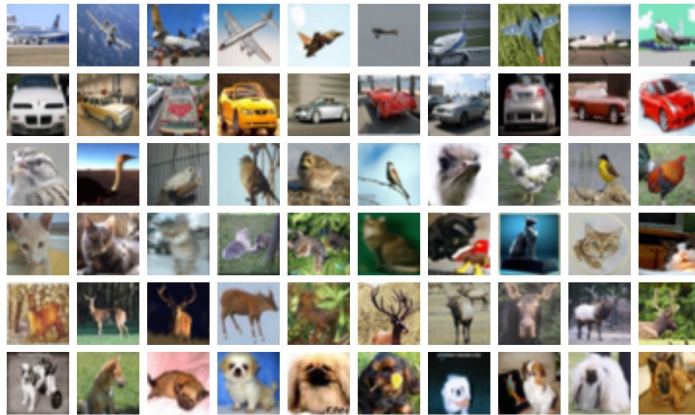
$\nabla_w \text{Loss}(x, y, w)$  denotes the gradient of one example loss with respect to  $w$ .

# Neural Networks

# Everything Has AI Now!

## Image Classification

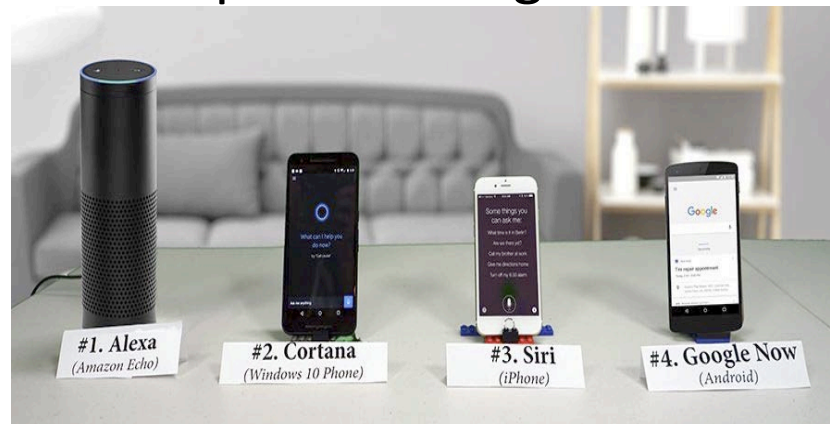
airplane  
automobile  
bird  
cat  
deer  
dog



## Image Captioning



## Speech Recognition



## Machine Translation





# 2016: YEAR of Deep Learning (and AI)



- DeepMind's AlphaGo beat South Korean professional Go player Lee Sedol.
- Out of five matches, Lee lost 4-1.

# 2019: Turing Award in Artificial Intelligence

*Turing Award Won by 3  
Pioneers in Artificial Intelligence*

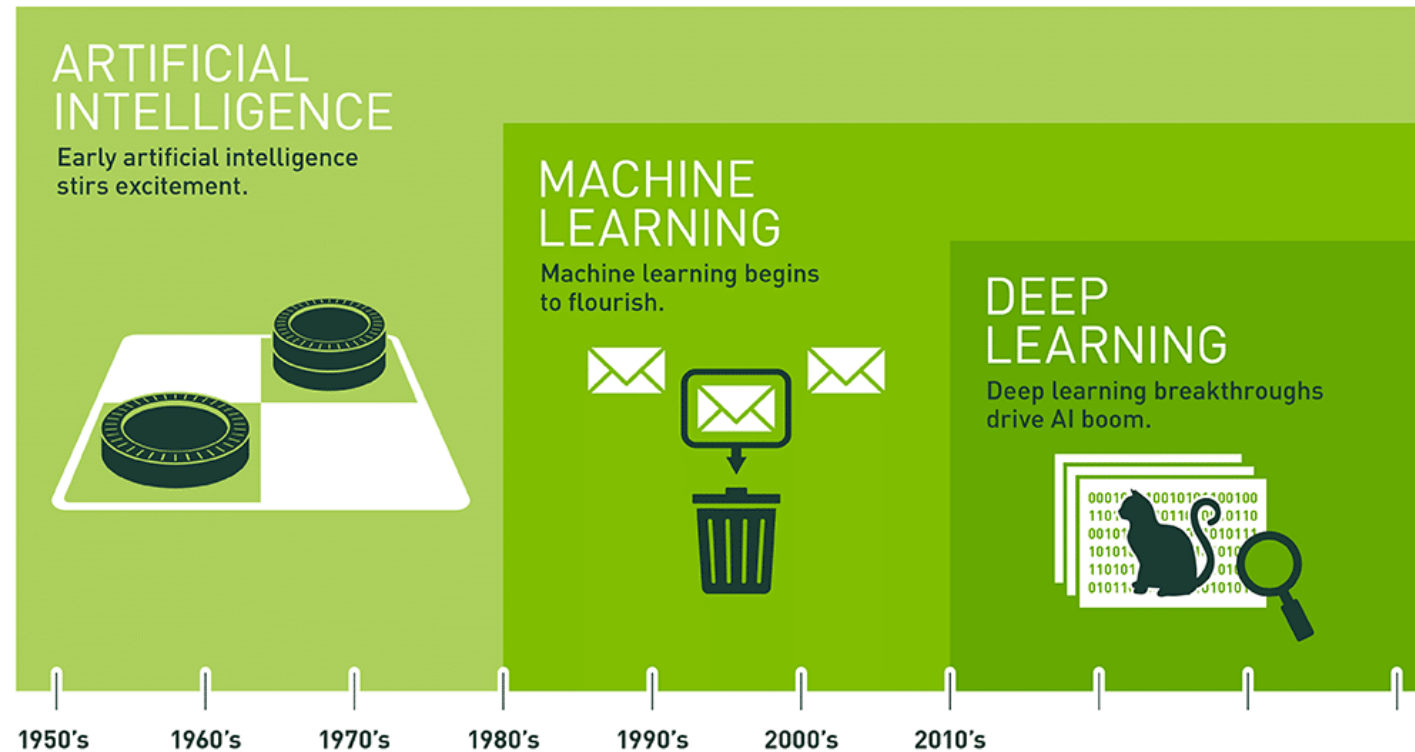


From left, Yann LeCun, Geoffrey Hinton and Yoshua Bengio. The researchers worked on key developments for neural networks, which are reshaping how computer systems are built.

# Myths about Deep Learning

- Deep learning mimics human brain.
  - Deep learning is only “loosely” inspired by the workings of the human brain.
- Deep learning is “magical” and the rest of machine learning models are completely “useless”.
  - The main power of deep learning comes from the fact that deep models can learn efficient representations.
- Any problem can be solved using deep learning.
  - Only in the presence of sufficient data to learn from.
- Deep learning replaces human jobs.
  - It creates more new interesting jobs for humans than it takes.
- Many others ...

# Deep Learning is Only a Small Part!!



Since an early flush of optimism in the 1950s, smaller subsets of artificial intelligence – first machine learning, then deep learning, a subset of machine learning – have created ever larger disruptions.

**Figure:** Nvidia blog about “What’s the Difference Between Artificial Intelligence, Machine Learning and Deep Learning?”.

# Deep Learning Breakthrough

- Large high-quality labeled datasets
- Parallel computing with GPUs
- Backprop-friendly activation functions (ReLU)
- Improved model architectures
- Software platforms
- New regularization techniques
- Robust optimization (momentum, RMSprop, ADAM)

# What is Neural Network?

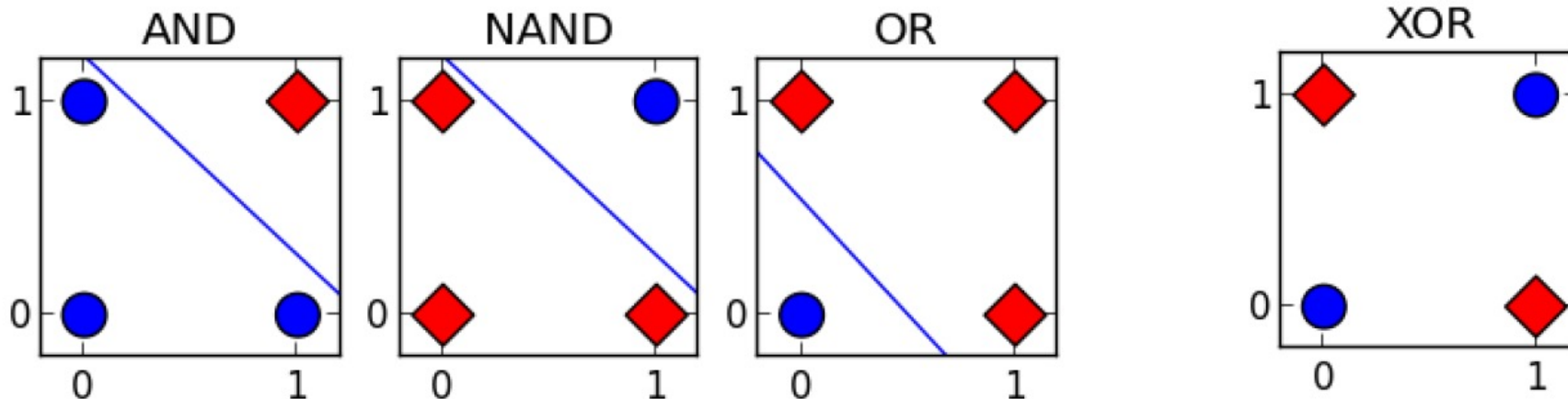
- Typically, it is associated with biological devices (brains), electronic devices, or network diagrams;
- But the best conceptualization in computing area is none of these: we think of a neural network as a **mathematical function**.

# Advantages of Neural Network

- Successfully used in **a variety of domains**: computer vision, speech recognition, gaming etc.
- Can provide solutions to very **complex and nonlinear** problems.
- If provided with **sufficient amount of data**, it can solve classification and forecasting problems accurately and easily.
- Once trained, **prediction is fast**.

# Motivation of Neural Network

- Perceptron: First learning algorithm for neural networks. (Frank Rosenblatt, 1957)
- Limitations: Marvin Minsky and Seymour Papert, "Perceptrons" 1969: **The perceptron can only solve problems with linearly separable classes.**
  - The functions can be drawn in 2-dim graph and a single straight line separates values in two parts.
  - The perceptron cannot model the non-linearly separable functions: Logical XOR function (computes the logical exclusive).



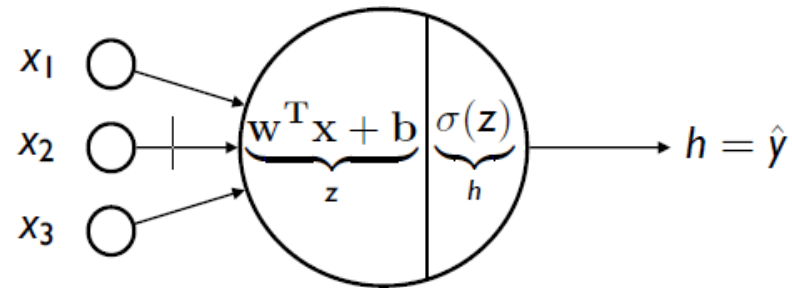


# Motivation of Neural Network

- Allows to learn **nonlinearly** separable transformations from input to output.
- A single **hidden layer** allows to compute any input/output transformation.

# No Hidden Units: Logistic Regression

- Sigmoid activation function:



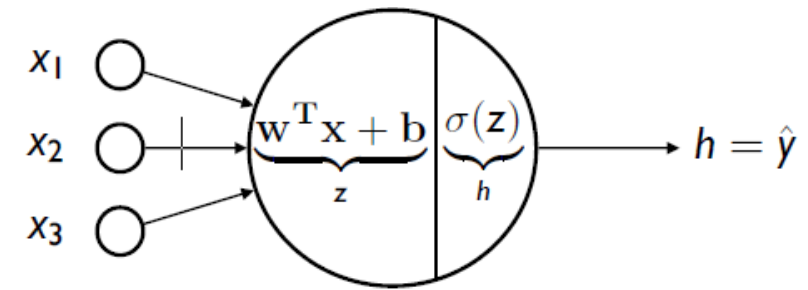
- Output score: sigmoid function  $\hat{y} = \sigma(w^T x + b)$
- Recall sigmoid function:

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

Derivative:  $\sigma'(z) = \sigma(z)(1 - \sigma(z))$

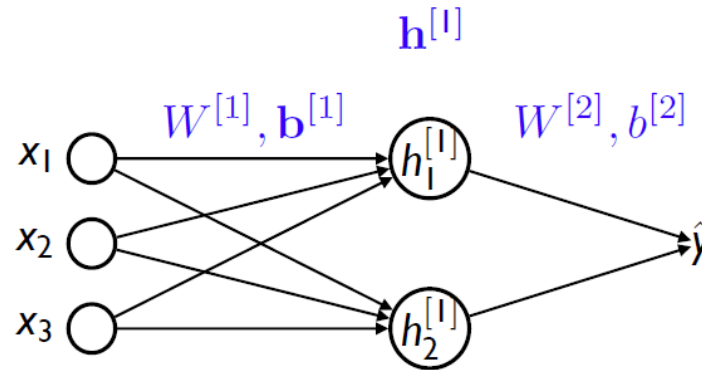
# Learning Algorithm: No hidden units

- Logistic loss:  $J(x, y, w) = -y \log \hat{y} - (1 - y) \log(1 - \hat{y})$
- $\hat{y} = \frac{1}{1 + e^{-w^T x}} = \sigma(w^T x)$
- $\frac{\partial \hat{y}}{\partial w_i} = \hat{y}(1 - \hat{y})x_i$
- $\frac{\partial J}{\partial w_i} = \frac{\partial J}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial w_i} = (\hat{y} - y)x_i$
- Applying gradient descent ( $\eta$  is the learning rate):
- Element (feature) operation:  $w_i^{t+1} = w_i^t - \eta \frac{\partial J}{\partial w_i}$
- Vector operation:  $w^{t+1} = w^t - \eta \frac{\partial J}{\partial w}$



# Neural Networks: One Hidden Layer

- One hidden layer:



- Hidden layer representation:

$$z_1^{[1]} = \mathbf{w}_1^{[1]T} \mathbf{x} + b_1^{[1]}, h_1^{[1]} = \sigma(z_1^{[1]})$$

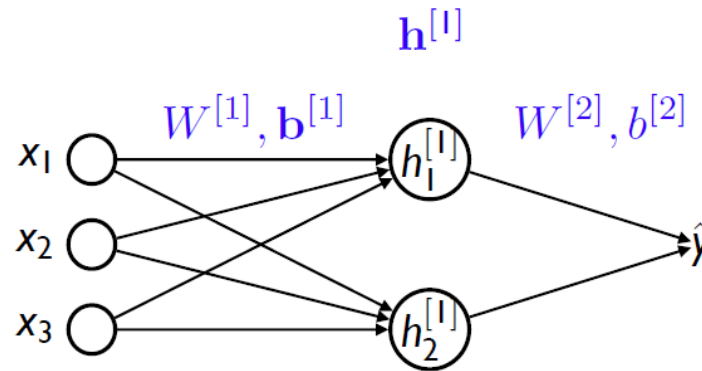
$$z_2^{[1]} = \mathbf{w}_2^{[1]T} \mathbf{x} + b_2^{[1]}, h_2^{[1]} = \sigma(z_2^{[1]})$$

$\Rightarrow$

$$\mathbf{z}^{[1]} = \underbrace{W^{[1]}}_{2 \times 3} \underbrace{\mathbf{x}}_{3 \times 1} + \underbrace{\mathbf{b}^{[1]}}_{2 \times 1}$$

# Neural Networks: One Hidden Layer

- One hidden layer:



- Output layer:

$$z^{[2]} = \underbrace{W^{[2]}}_{1 \times 2} \underbrace{\mathbf{h}^{[1]}}_{2 \times 1} + b^{[2]}$$
$$\hat{y} = \sigma(z^{[2]})$$

# Neural Networks: One Hidden Layer

- We can think of intermediate hidden units as learned features of a linear predictor.
- **Feature learning**: manually specified features:

$\mathbf{x}$

automatically learned features:

$$h(\mathbf{x}) = [h_1(\mathbf{x}), \dots, h_k(\mathbf{x})]$$

# Loss Minimization

- Optimization problem

$$TrainLoss(W, b) = \frac{1}{|D_{train}|} \sum_{(x,y) \in D_{train}} L(x, y, W, b)$$

$$L(x, y, W, b) = -y \log \hat{y} - (1 - y) \log(1 - \hat{y})$$

$$\hat{y} = h^{[2]}$$

- Goal: compute gradient

$$\nabla_{W,b} L(W, b)$$

# Stochastic Gradient Descent

- Goal: compute gradient

$$\nabla_{W,b} L(W, b)$$

- Repeat, given a sample  $(x^{(i)}, y^{(i)})$

$$dW^{[1]} = \frac{\partial L}{\partial W^{[1]}}, \quad W^{[1]} \leftarrow W^{[1]} - \eta dW^{[1]}$$

$$db^{[1]} = \frac{\partial L}{\partial b^{[1]}}, \quad b^{[1]} \leftarrow b^{[1]} - \eta db^{[1]}$$

$$dW^{[2]} = \frac{\partial L}{\partial W^{[2]}}, \quad W^{[2]} \leftarrow W^{[2]} - \eta dW^{[2]}$$

$$db^{[2]} = \frac{\partial L}{\partial b^{[2]}}, \quad b^{[2]} \leftarrow b^{[2]} - \eta db^{[2]}$$

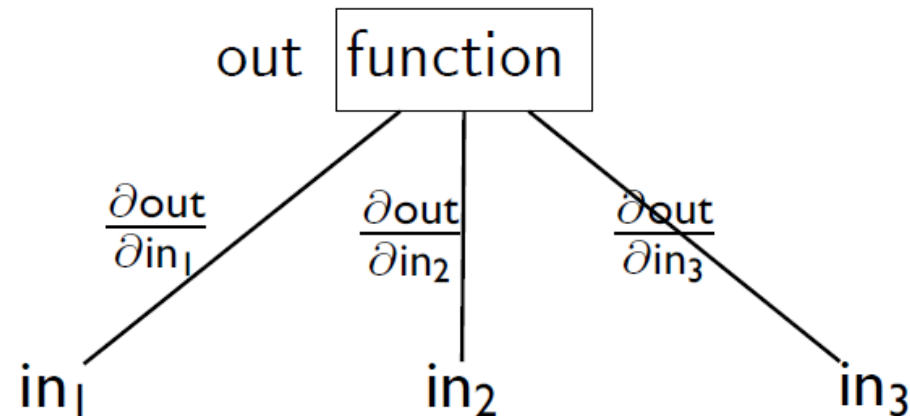


# Approach

- Mathematical: **chain rule**
- Next: visualize the computation using a **computation graph**
- Advantages:
  - Avoid long equations
  - Reveal structure of computations (modularity, efficiency, dependencies)

# Functions As Boxes

- Partial derivatives (gradients): how much does the output change if an input changes?

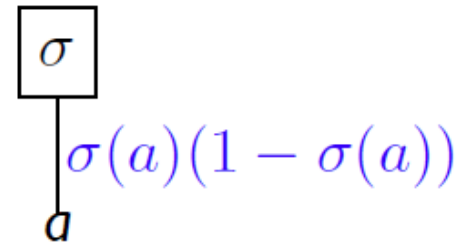
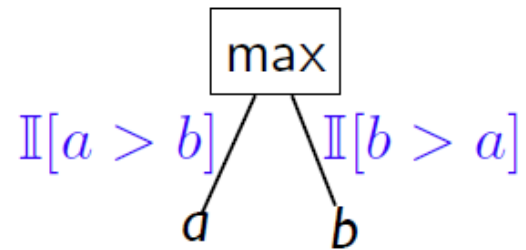
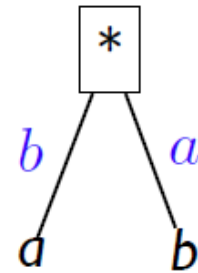
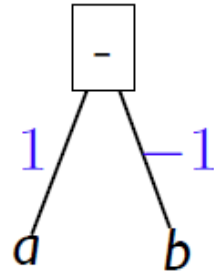
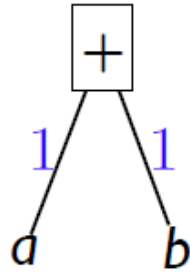


- Example:

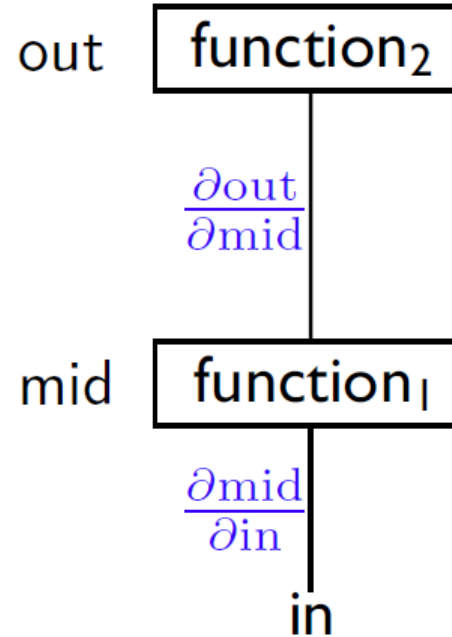
$$out = 2in_1 + in_2in_3$$
$$2in_1 + (in_2 + \epsilon)in_3 = out + \epsilon in_3$$

- Partial derivatives (gradients): a measure of **sensitivity**

# Basic Building Blocks



# Composing Functions



- Chain rule:

$$\frac{\partial \text{out}}{\partial \text{in}} = \frac{\partial \text{out}}{\partial \text{mid}} \frac{\partial \text{mid}}{\partial \text{in}}$$

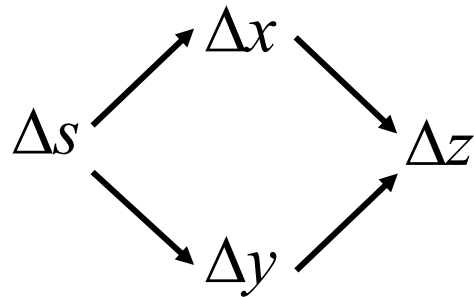
# Chain Rule

Case 1       $y = g(x) \quad z = h(y)$

$$\Delta x \rightarrow \Delta y \rightarrow \Delta z \qquad \frac{dz}{dx} = \frac{dz}{dy} \frac{dy}{dx}$$

Case 2

$$x = g(s) \qquad y = h(s) \qquad z = k(x, y)$$

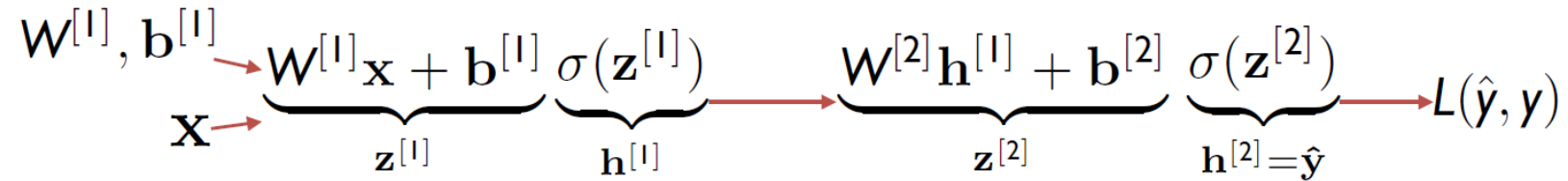


$$\frac{\partial z}{\partial s} = \frac{\partial z}{\partial x} \frac{\partial x}{\partial s} + \frac{\partial z}{\partial y} \frac{\partial y}{\partial s}$$

# Back Propagation

- Goal: learn the weights so that the loss (error) is minimized.
- It provides an efficient procedure to compute derivatives.
- For a fixed sample  $(x, y)$ , we want to learn  $\hat{y} = f(x)$
- Negative Log-likelihood over input  $x_n$  is  $L_n = -y_n \log \hat{y}_n - (1 - y_n) \log(1 - \hat{y}_n)$
- Total error of training examples is  $E = \sum_n L_n$
- **Backpropagation**: an efficient way to compute the gradients of the network parameters.

# Back Propagation

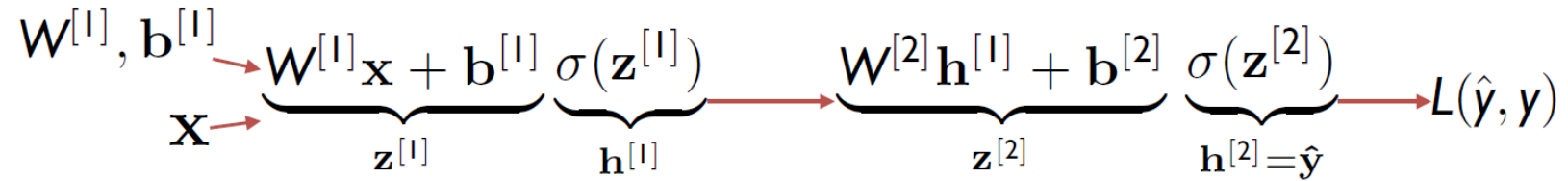


- Loss function for each data sample:  $L_n = -y_n \log \hat{y}_n - (1 - y_n) \log(1 - \hat{y}_n)$

$$\frac{\partial L}{\partial W^{[1]}} = \frac{\partial L}{\partial z^{[1]}} \frac{\partial z^{[1]}}{\partial W^{[1]}}, \quad \frac{\partial L}{\partial b^{[1]}} = \frac{\partial L}{\partial z^{[1]}} \frac{\partial z^{[1]}}{\partial b^{[1]}}$$

$$\frac{\partial L}{\partial W^{[2]}} = \frac{\partial L}{\partial z^{[2]}} \frac{\partial z^{[2]}}{\partial W^{[2]}}, \quad \frac{\partial L}{\partial b^{[2]}} = \frac{\partial L}{\partial z^{[2]}} \frac{\partial z^{[2]}}{\partial b^{[2]}}$$

# Back Propagation



$$\frac{\partial L}{\partial z^{[2]}} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z^{[2]}} = \frac{\hat{y} - y}{\hat{y}(1 - \hat{y})} \hat{y}(1 - \hat{y}) = \hat{y} - y$$

$$\frac{\partial L}{\partial \hat{y}} = \frac{\hat{y} - y}{\hat{y}(1 - \hat{y})}$$

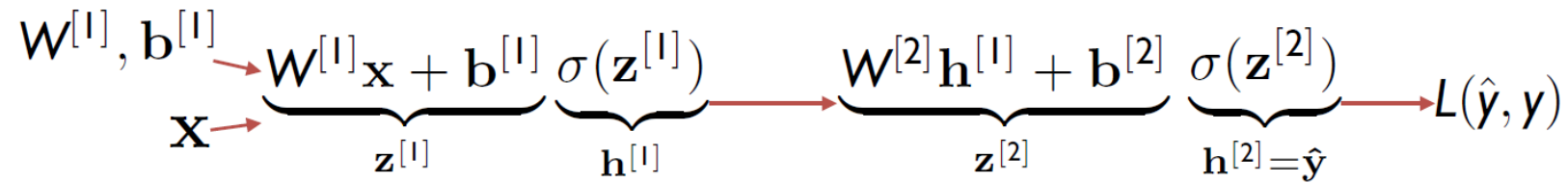
$$\frac{\partial z^{[2]}}{\partial W^{[2]}} = h^{[1]}, \quad \frac{\partial z^{[2]}}{\partial b^{[2]}} = 1$$

$$\frac{\partial L}{\partial W^{[2]}} = \frac{\partial L}{\partial z^{[2]}} \frac{\partial z^{[2]}}{\partial W^{[2]}} = (\hat{y} - y) h^{[1]}$$

$$\frac{\partial L}{\partial b^{[2]}} = \frac{\partial L}{\partial z^{[2]}} \frac{\partial z^{[2]}}{\partial b^{[2]}} = \hat{y} - y$$



# Back Propagation



$$\frac{\partial L}{\partial \mathbf{z}^{[1]}} = \frac{\partial L}{\partial \mathbf{z}^{[2]}} \frac{\partial \mathbf{z}^{[2]}}{\partial \mathbf{z}^{[1]}} = (\hat{y} - y) W^{[2]} h^{[1]} (1 - h^{[1]})$$

$$\frac{\partial \mathbf{z}^{[2]}}{\partial \mathbf{h}^{[1]}} = \frac{\partial \mathbf{z}^{[2]}}{\partial \mathbf{h}^{[1]}} \frac{\partial \mathbf{h}^{[1]}}{\partial \mathbf{z}^{[1]}} = W^{[2]} h^{[1]} (1 - h^{[1]})$$

$$\frac{\partial \mathbf{z}^{[1]}}{\partial W^{[1]}} = \mathbf{x}, \quad \frac{\partial \mathbf{z}^{[1]}}{\partial \mathbf{b}^{[1]}} = 1$$

$$\frac{\partial L}{\partial W^{[1]}} = \frac{\partial L}{\partial \mathbf{z}^{[1]}} \frac{\partial \mathbf{z}^{[1]}}{\partial W^{[1]}} = (\hat{y} - y) W^{[2]} h^{[1]} (1 - h^{[1]}) \mathbf{x}$$

$$\frac{\partial L}{\partial \mathbf{b}^{[1]}} = \frac{\partial L}{\partial \mathbf{z}^{[1]}} \frac{\partial \mathbf{z}^{[1]}}{\partial \mathbf{b}^{[1]}} = (\hat{y} - y) W^{[2]} h^{[1]} (1 - h^{[1]})$$

# Summary: Computing Derivatives

- **Forward** propagation:

$$z^{[1]} = W^{[1]}x + b^{[1]}$$

$$h^{[1]} = \sigma(z^{[1]})$$

$$z^{[2]} = W^{[2]}h^{[1]} + b^{[2]}$$

$$\hat{y} = h^{[2]} = \sigma(z^{[2]})$$

- **Backpropagation:**

$$\frac{\partial L}{\partial z^{[2]}} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z^{[2]}} = \hat{y} - y$$

$$\frac{\partial L}{\partial \hat{y}} = \frac{\hat{y} - y}{\hat{y}(1 - \hat{y})}$$

$$\frac{\partial z^{[2]}}{\partial W^{[2]}} = h^{[1]}, \quad \frac{\partial z^{[2]}}{\partial b^{[2]}} = 1$$

$$\frac{\partial L}{\partial z^{[1]}} = \frac{\partial L}{\partial z^{[2]}} \frac{\partial z^{[2]}}{\partial z^{[1]}} = (\hat{y} - y)W^{[2]}h^{[1]}(1 - h^{[1]})$$

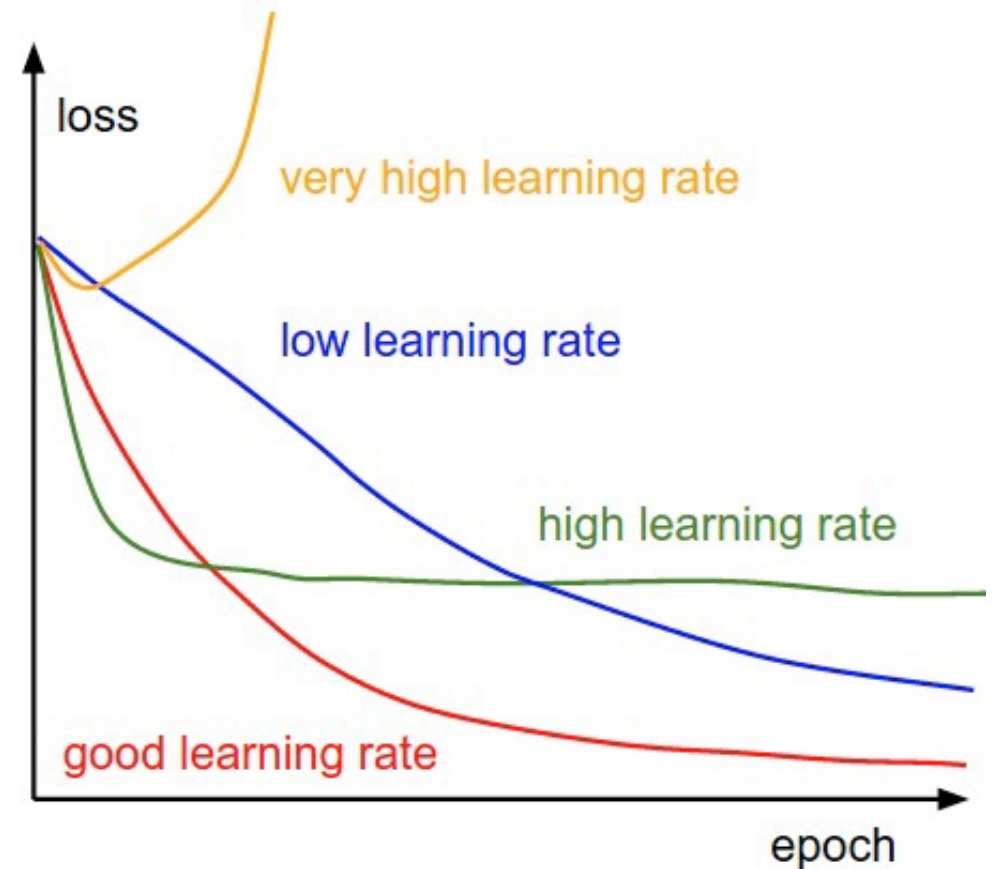
$$\frac{\partial z^{[2]}}{\partial z^{[1]}} = \frac{\partial z^{[2]}}{\partial h^{[1]}} \frac{\partial h^{[1]}}{\partial z^{[1]}} = W^{[2]}h^{[1]}(1 - h^{[1]})$$

$$\frac{\partial z^{[1]}}{\partial W^{[1]}} = x, \quad \frac{\partial z^{[1]}}{\partial b^{[1]}} = 1$$

# Learning Rate

Set the learning rate carefully:

- If there are more than 3 parameters, it is hard to visualize the loss w.r.t the parameters.
- But you can visualize the loss w.r.t the # of parameter updates (epoch).
  - If the loss increases, we need to decrease the value of the learning rate.
  - If the loss is decreasing at a very slow rate, we need to increase the value of the learning rate.

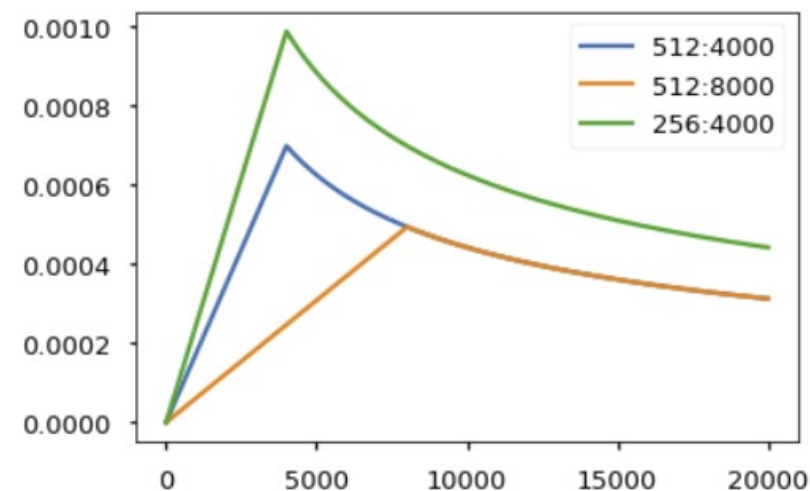


# Adaptive Learning Rates

- Popular & Simple Idea: Reduce the learning rate by some factor every few epochs.
  - At the beginning, we are far from the destination, so we use larger learning rate
  - After several epochs, we are close to the destination, so we reduce the learning rate
- Learning rate cannot be one-size-fits-all
  - Giving different parameters different learning rates
  - Adagrad, RMSprop, Adam, etc

# Warm-up Learning Rate

- **Motivation:** At the beginning of training, the weights of the model are randomly initialized. If you choose a larger learning rate, it may lead to instability (oscillation) of the model.
- **Solution:** At the beginning of training, it uses a small learning rate to train some epochs or steps, and then modifies it to the preset learning for training.
- The model can gradually become stable, which makes the convergence speed of the model become faster and the model effect is better.

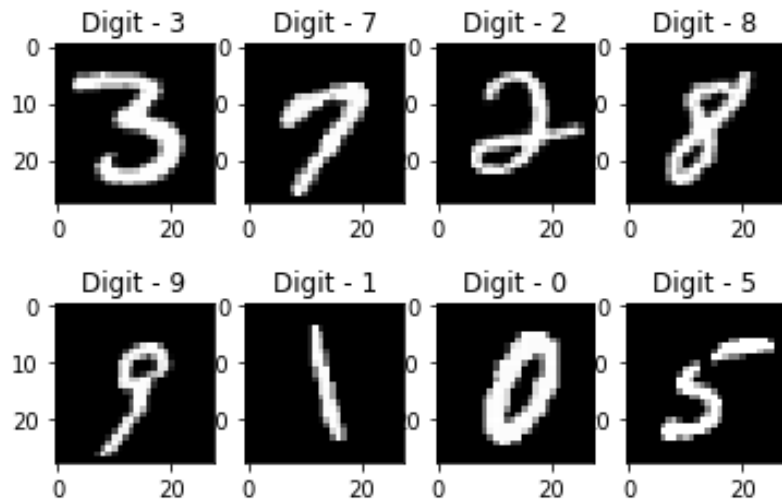


# Optimization

- Gradient descent
- Stochastic gradient descent
- Mini-batch gradient descent

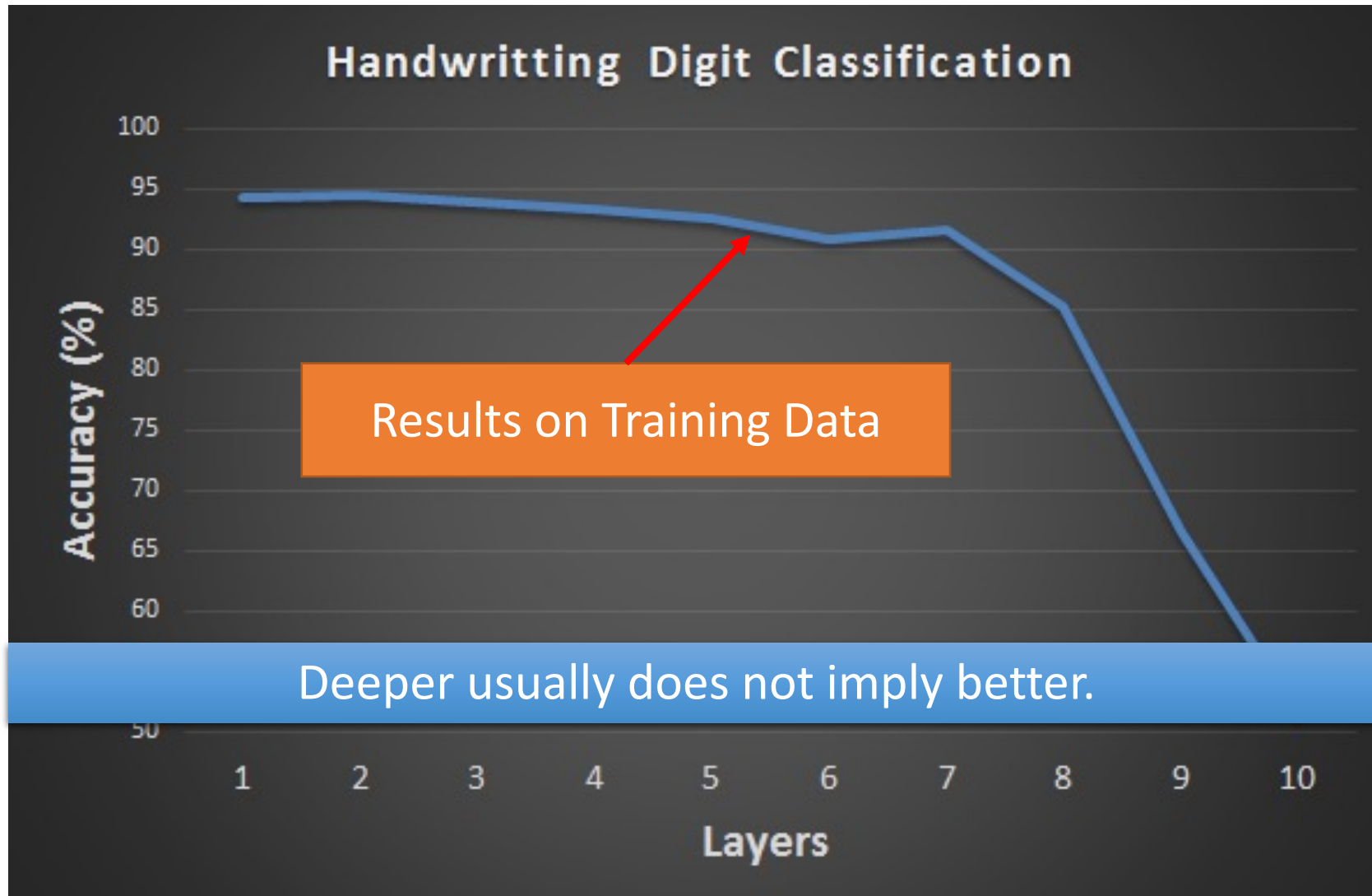
# Comparisons

- Handwriting Digit Classification



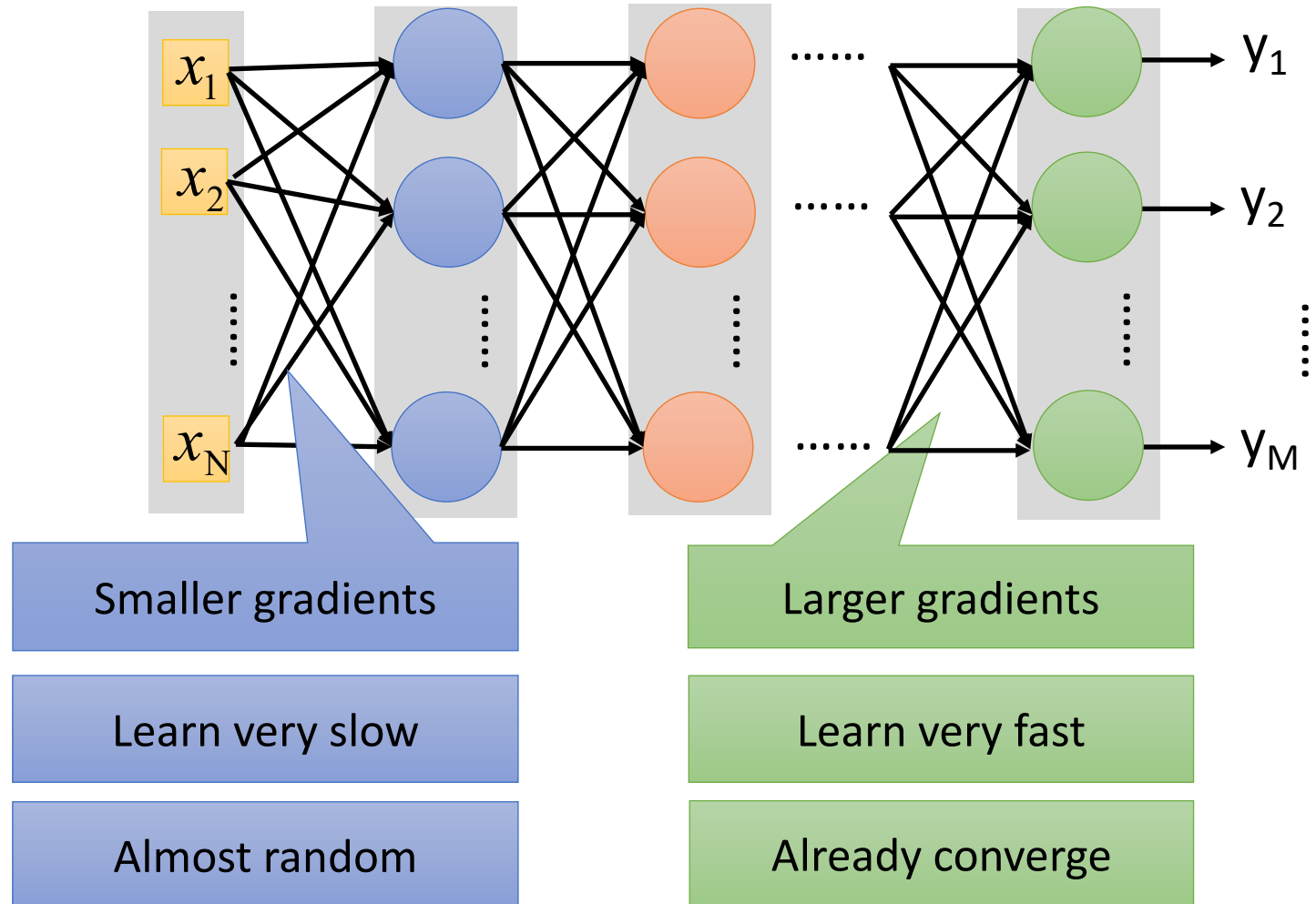
Gradient  
Descent

# Hard to Get the Power of Deep ...

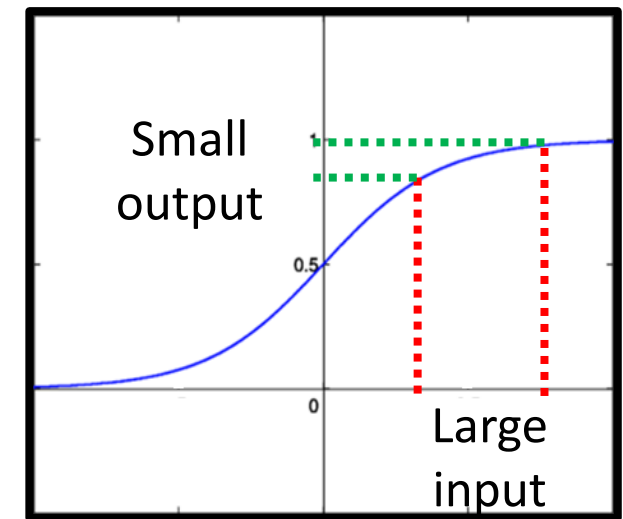
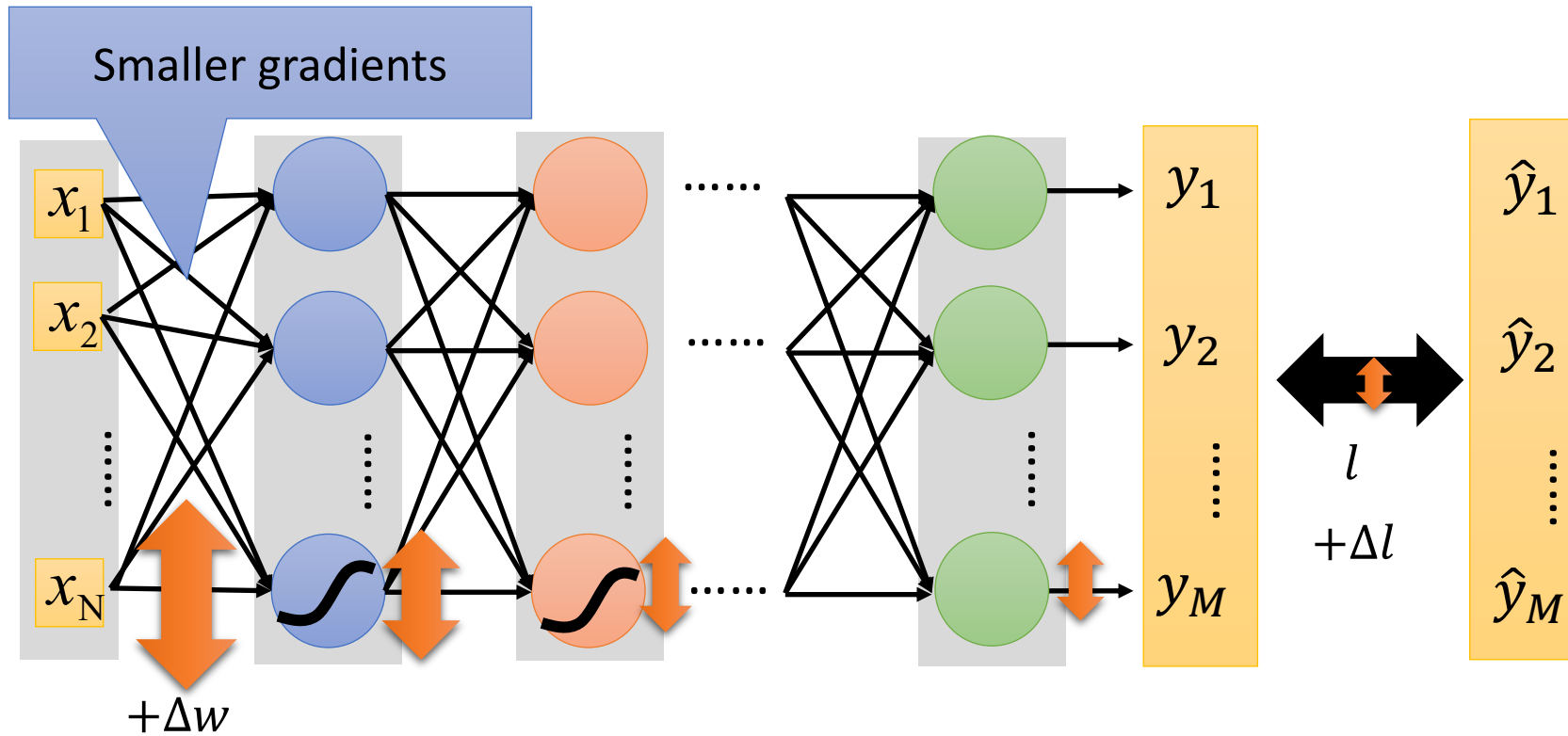




# Vanishing Gradient Problem



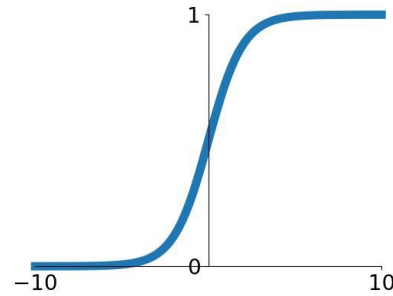
# Vanishing Gradient Problem



# Activation Functions

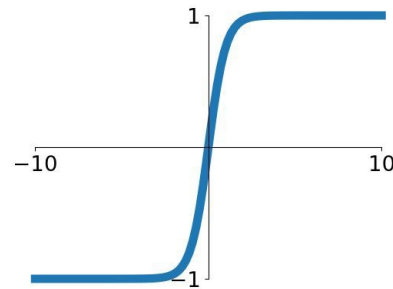
## Sigmoid

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



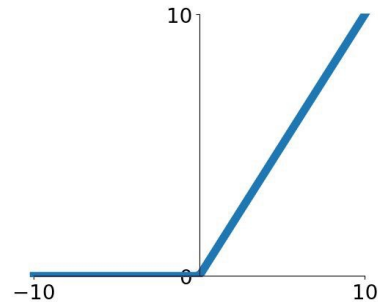
## tanh

$$\tanh(x)$$



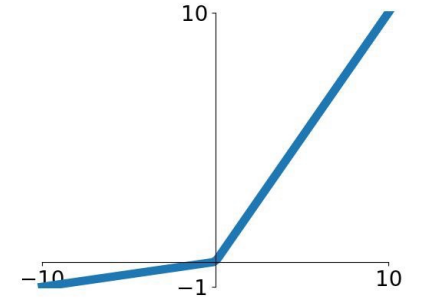
## ReLU

$$\max(0, x)$$



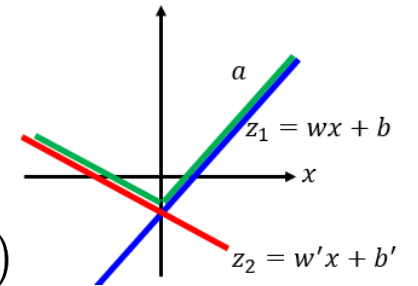
## Leaky ReLU

$$\max(0.1x, x)$$



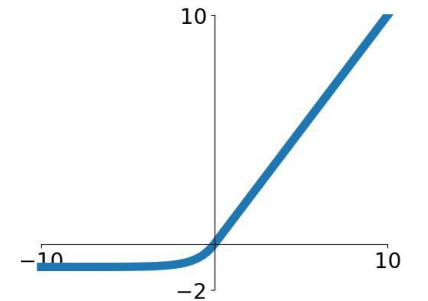
## Maxout

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

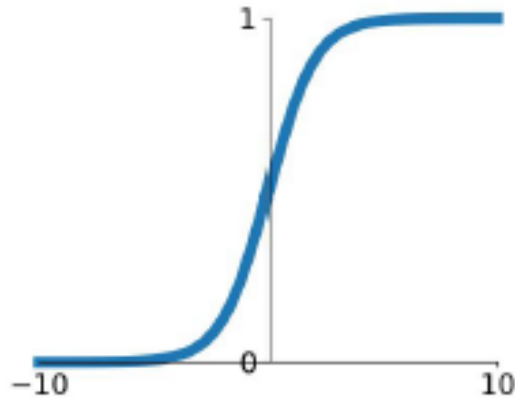


## ELU

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$

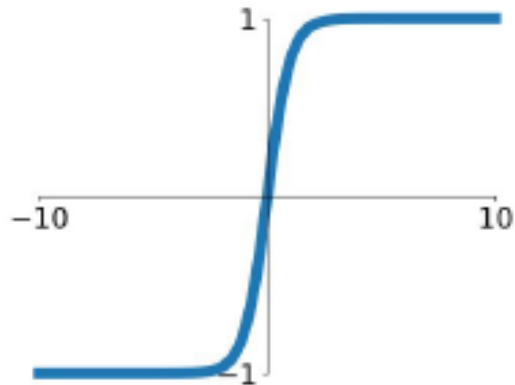


# Sigmoid and tanh



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

1. Vanishing gradient problem
2. The output of sigmoid function is not zero centered.
3. It is expensive to calculate  $e^{-x}$ .

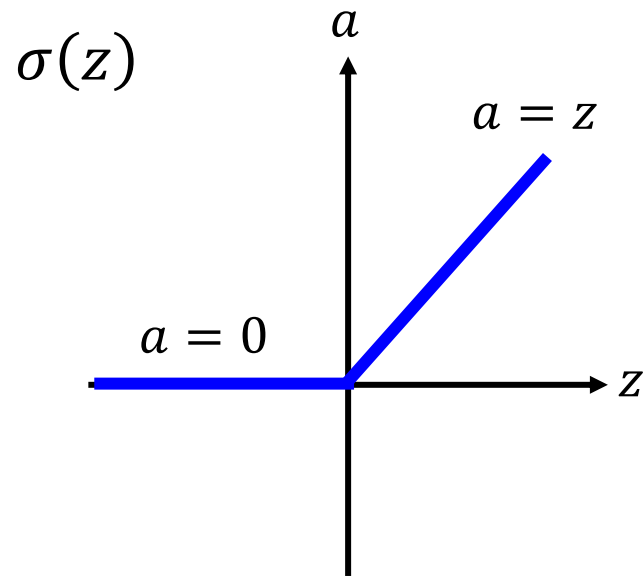


$\tanh(x)$

1. The output is zero centered.
2. Suffer from vanishing gradient problem.

# ReLU

- Rectified Linear Unit (ReLU)



[Xavier Glorot, AISTATS'11]

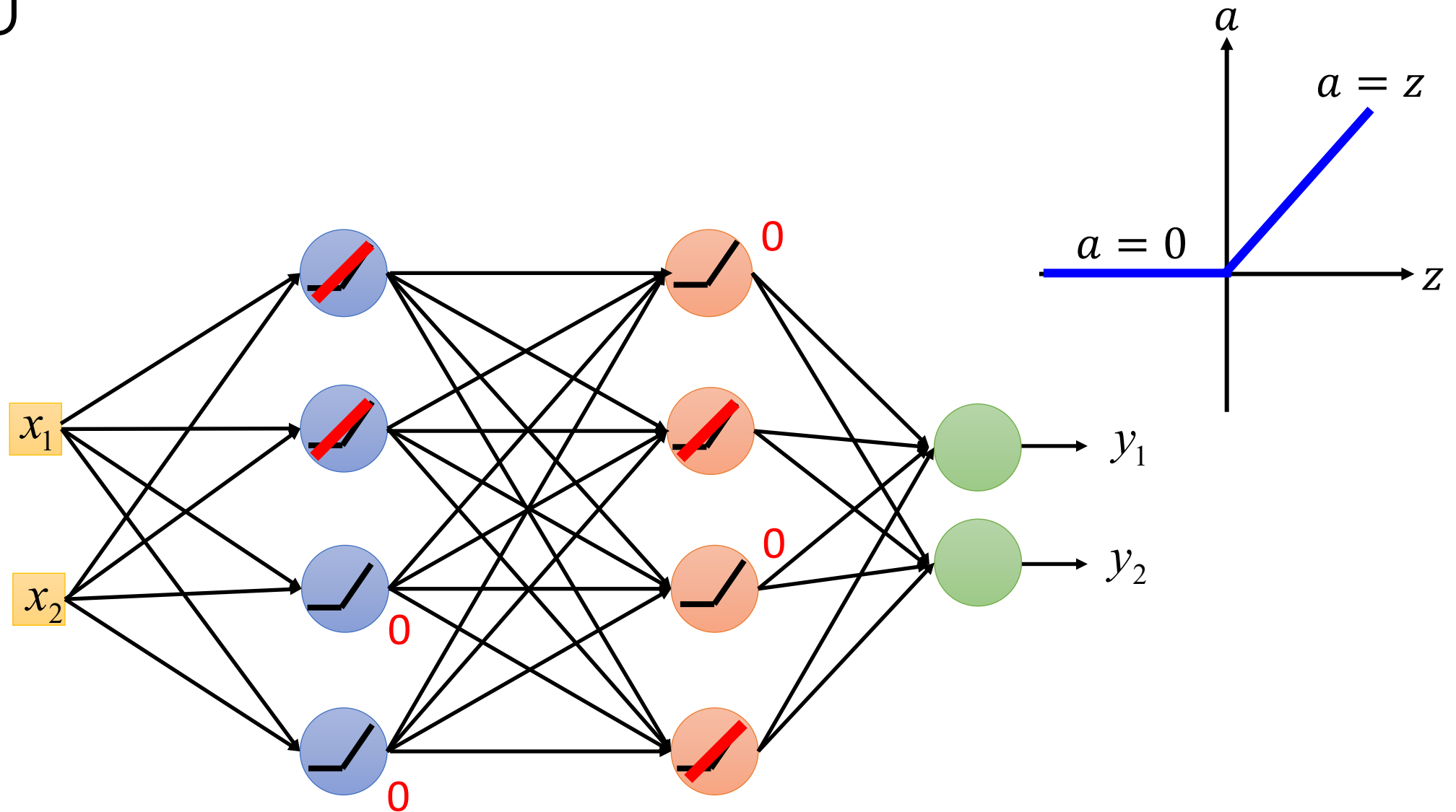
[Andrew L. Maas, ICML'13]

[Kaiming He, arXiv'15]

## Reason:

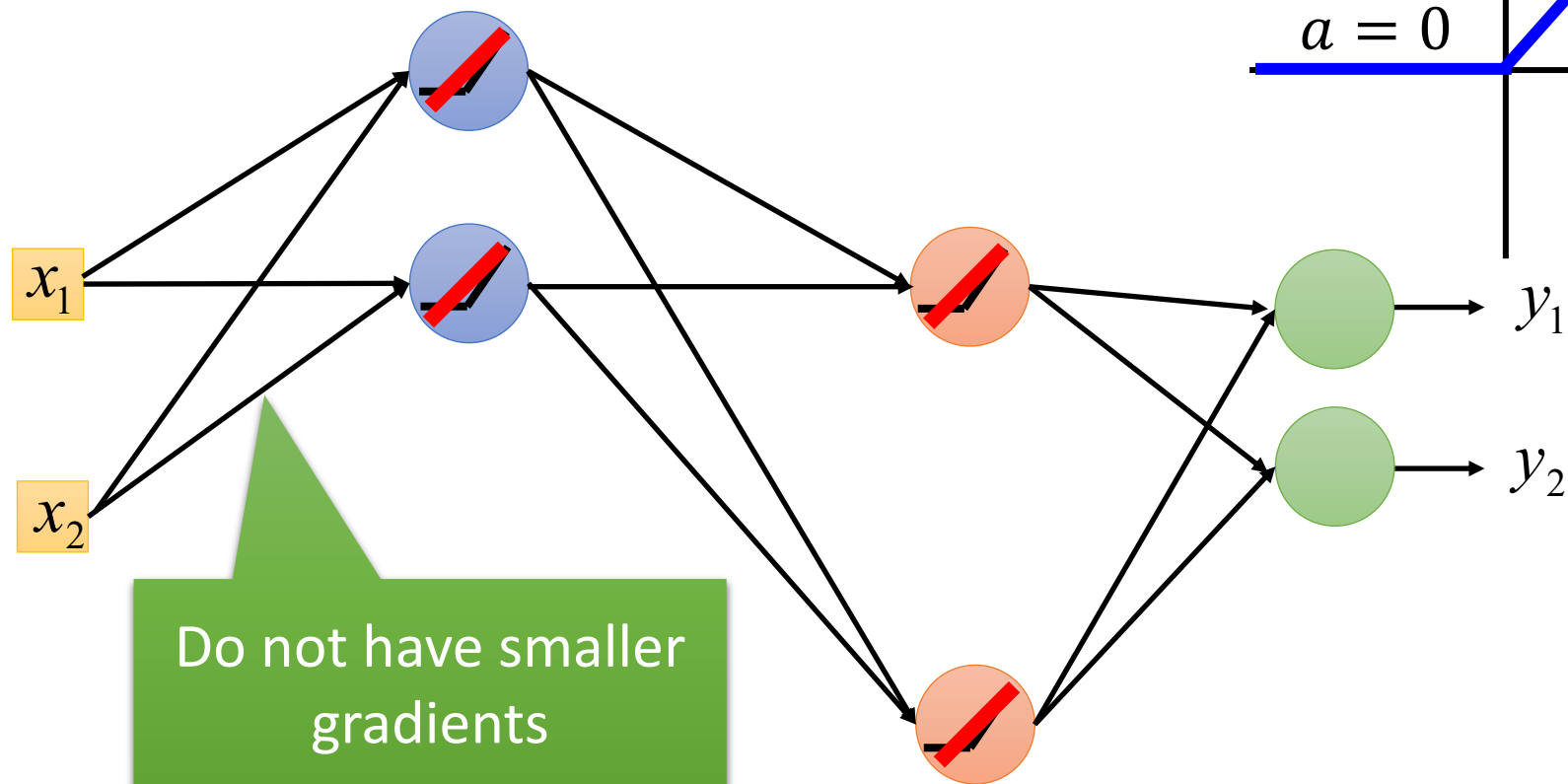
1. Fast to compute: no exponential computations
2. Solves the vanishing gradient problem

# ReLU



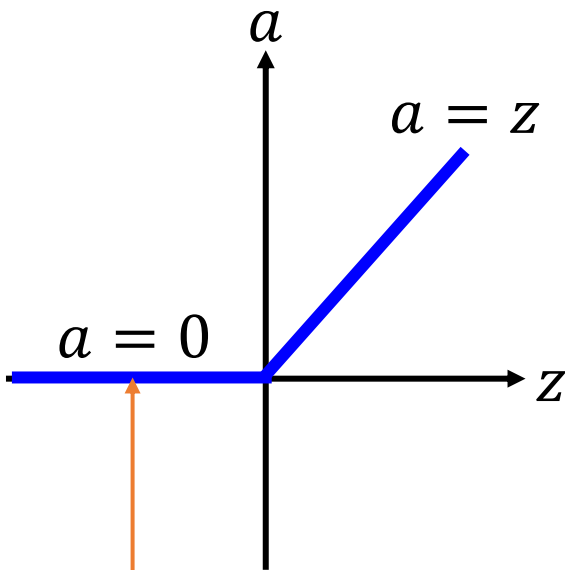
# ReLU

A Thinner linear network



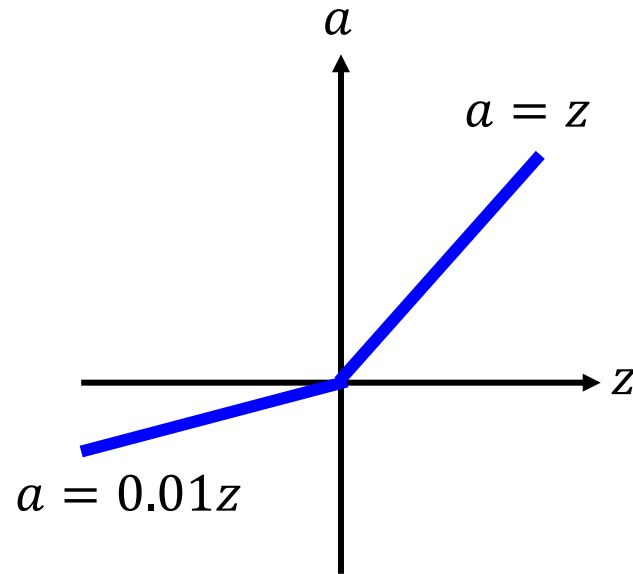
# ReLU: Variant

*ReLU*

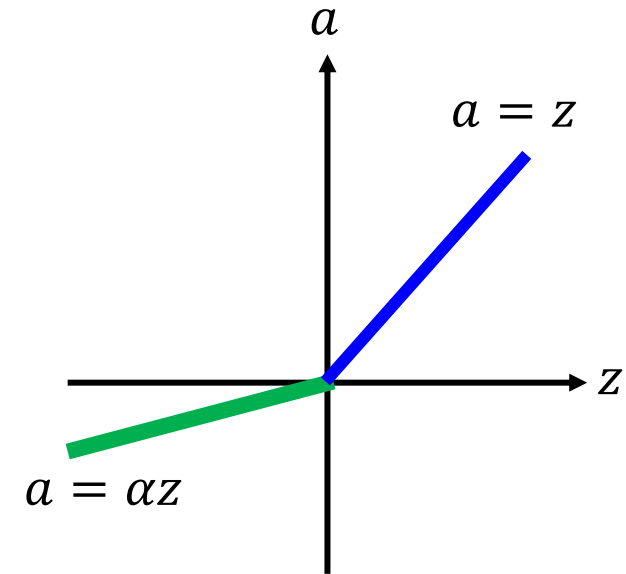


May never be  
activated

*Leaky ReLU*



*Parametric ReLU*



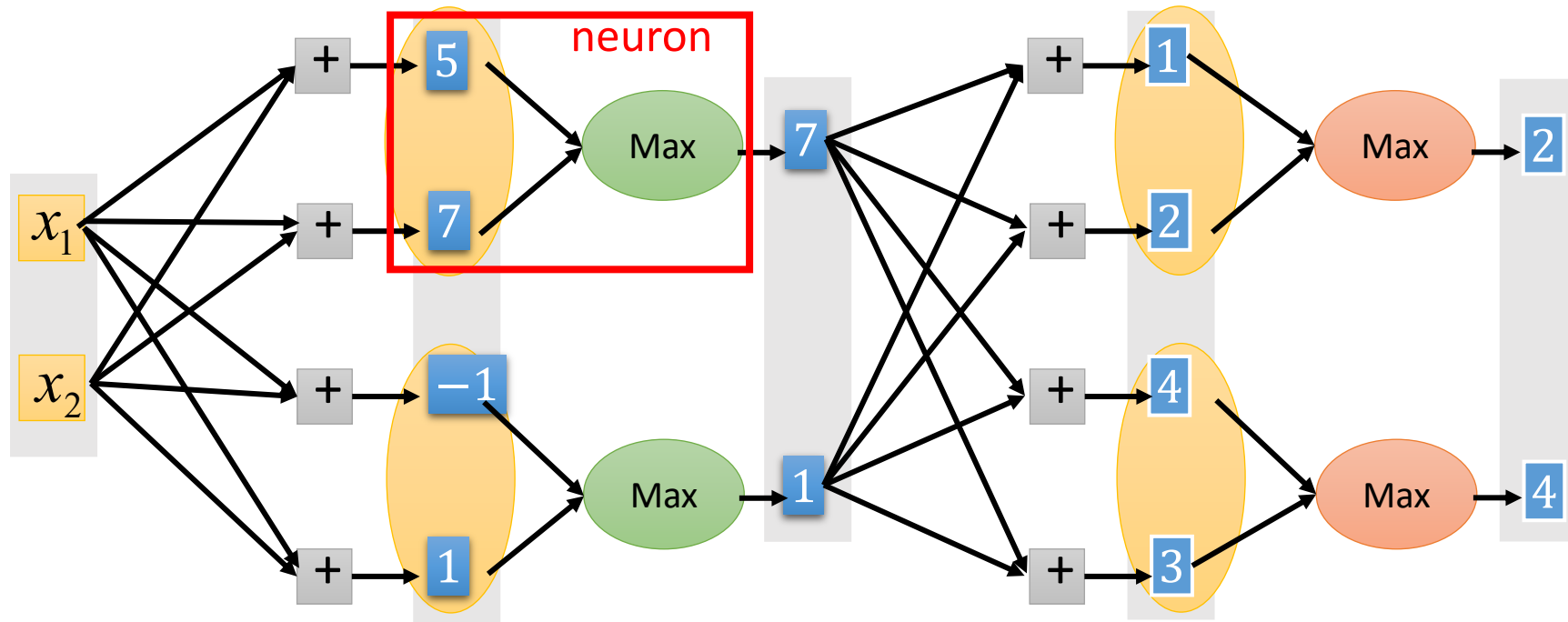
$\alpha$  also learned by  
gradient descent



# Maxout

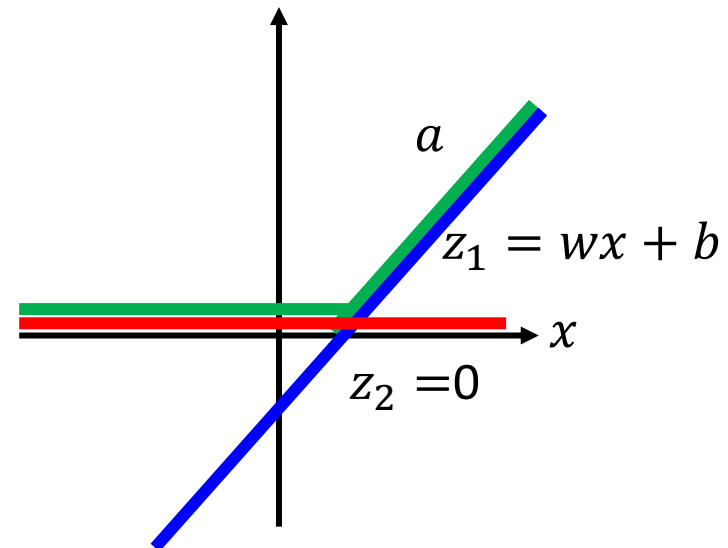
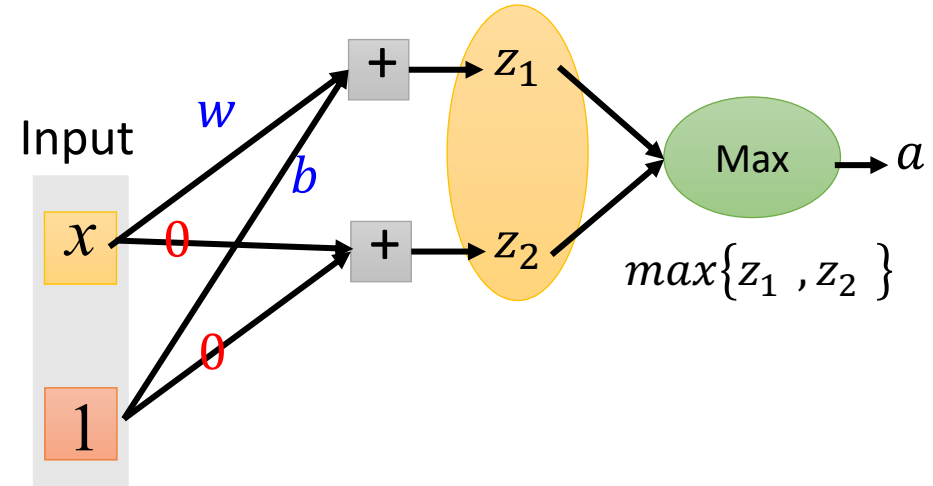
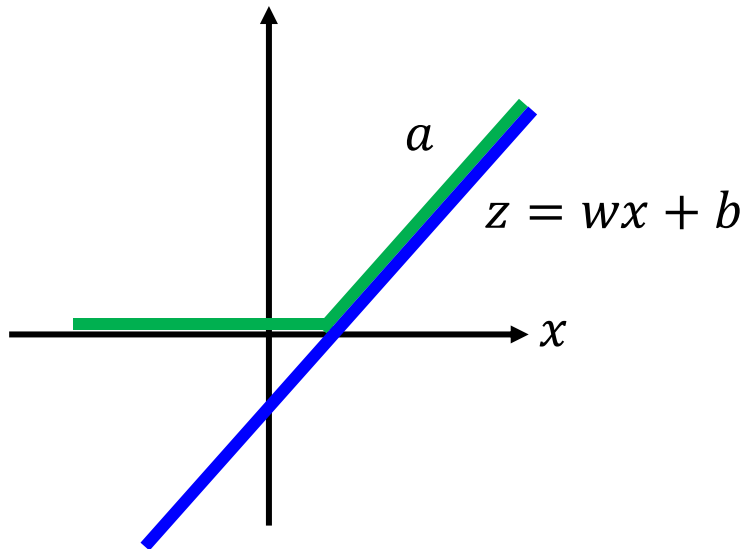
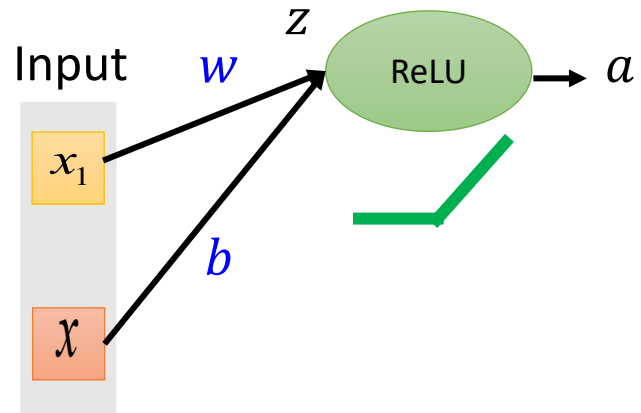
- Learnable activation function [Ian J. Goodfellow, ICML'13]

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



You can have more than 2 elements in a group.

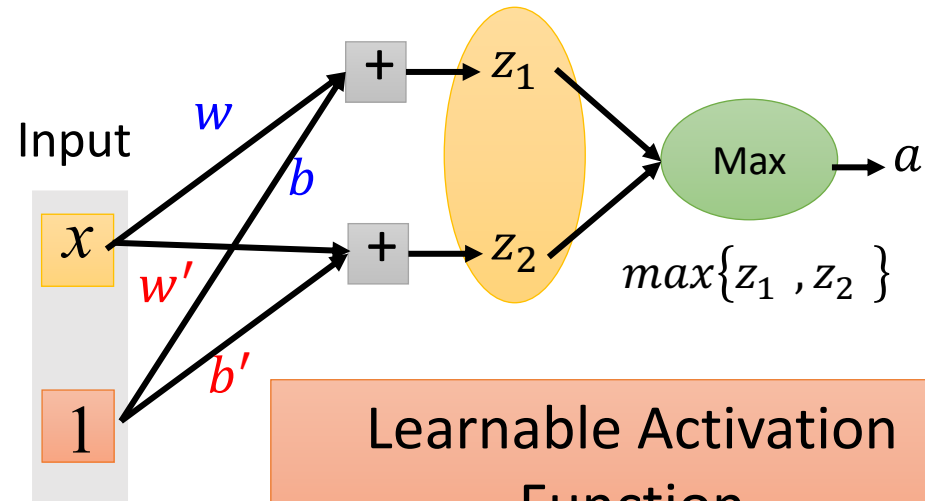
# Maxout



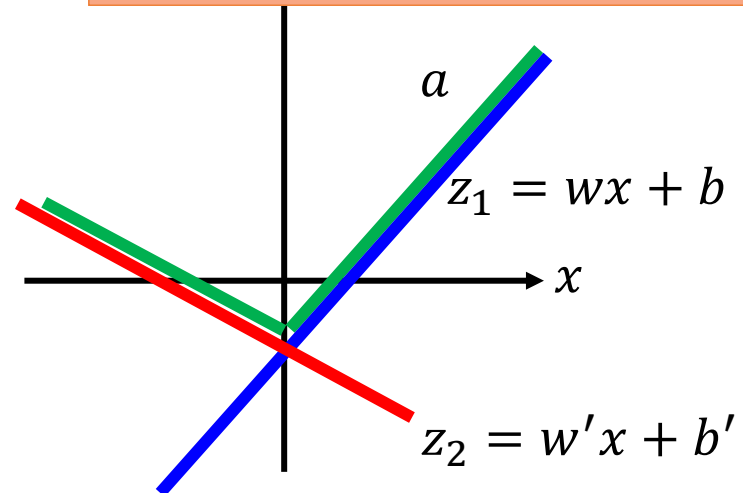
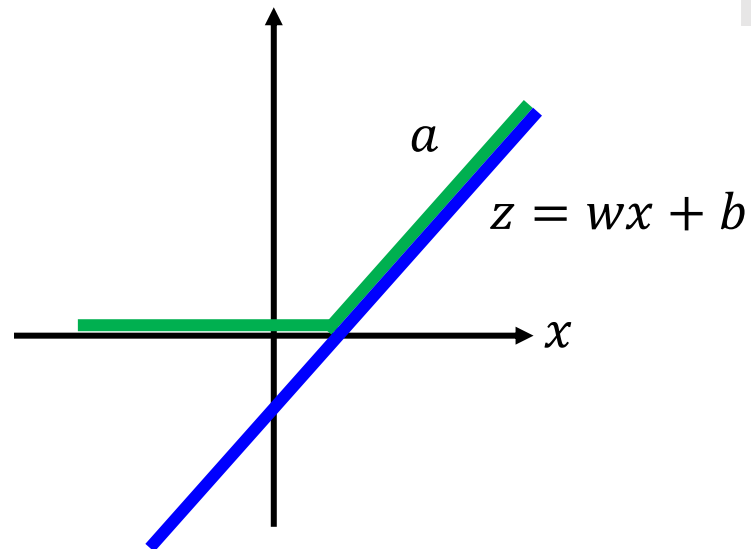
ReLU is a special cases of Maxout

# Maxout

More than ReLU



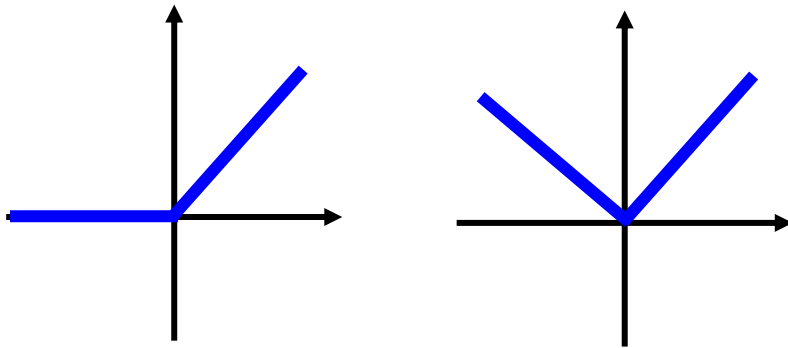
Learnable Activation Function



# Maxout

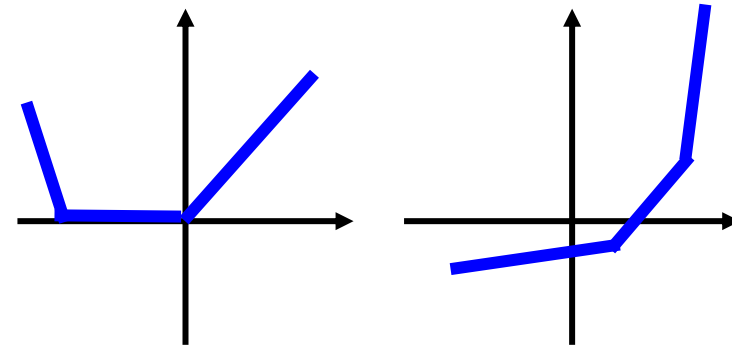
- Learnable activation function [\[Ian J. Goodfellow, ICML'13\]](#)
  - Activation function in maxout network can be any **piecewise** linear convex function
  - How many pieces depending on how many elements in a group

2 elements in a group



Problem: The number of parameters are doubled.

3 elements in a group



# Regularization

- New loss function to be minimized
- Find a set of weights not only minimizing original loss but also reducing overfitting.

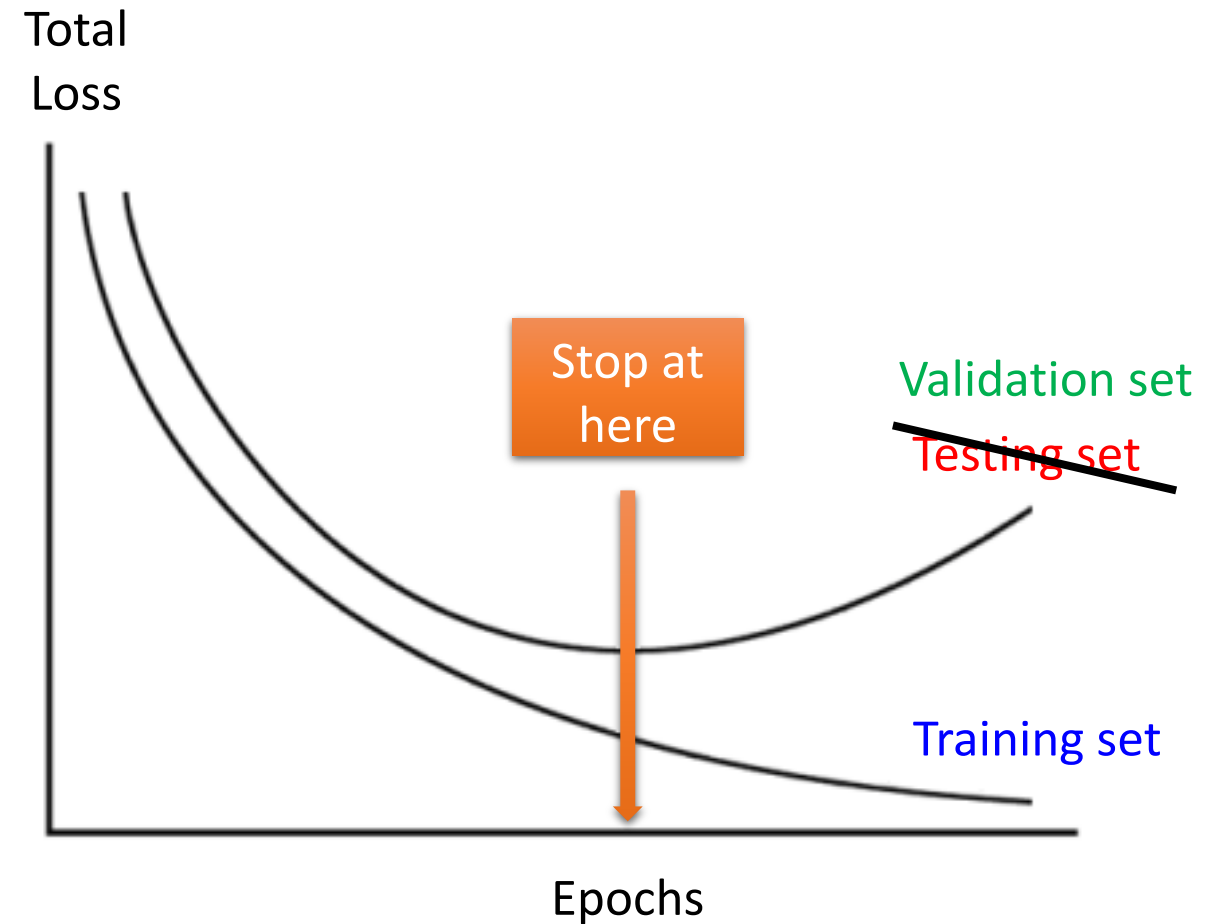
$$L'(\theta) = \underbrace{L(\theta)}_{\substack{\text{Original loss} \\ \text{(e.g. minimize squared} \\ \text{error, cross entropy ...)}}} + \lambda \frac{1}{2} \underbrace{\|\theta\|_2}_{\substack{\text{Regularization term} \\ \theta = \{w_1, w_2, \dots\}}} \rightarrow$$

L2 regularization:  $\|\theta\|_2 = (w_1)^2 + (w_2)^2 + \dots$

L1 regularization:  $\|\theta\|_1 = |w_1| + |w_2| + \dots$

# Early Stopping

- One of the most widely used regularization techniques to overcome the **overfitting issue**.
- Monitors the performance of the model for every epoch on the **validation** set during the training.
- Terminates the training as soon as the validation error reaches a minimum.



# Implementation

- Deep learning packages: [Tensorflow](#), [Pytorch](#) ...
- [Visualization of Learning a Neural Network](#)
- [Implementation of a 1-hidden layer Neural Network](#)

# Summary of Today's Lecture

- Summary and review
- Neural networks
- Implementation