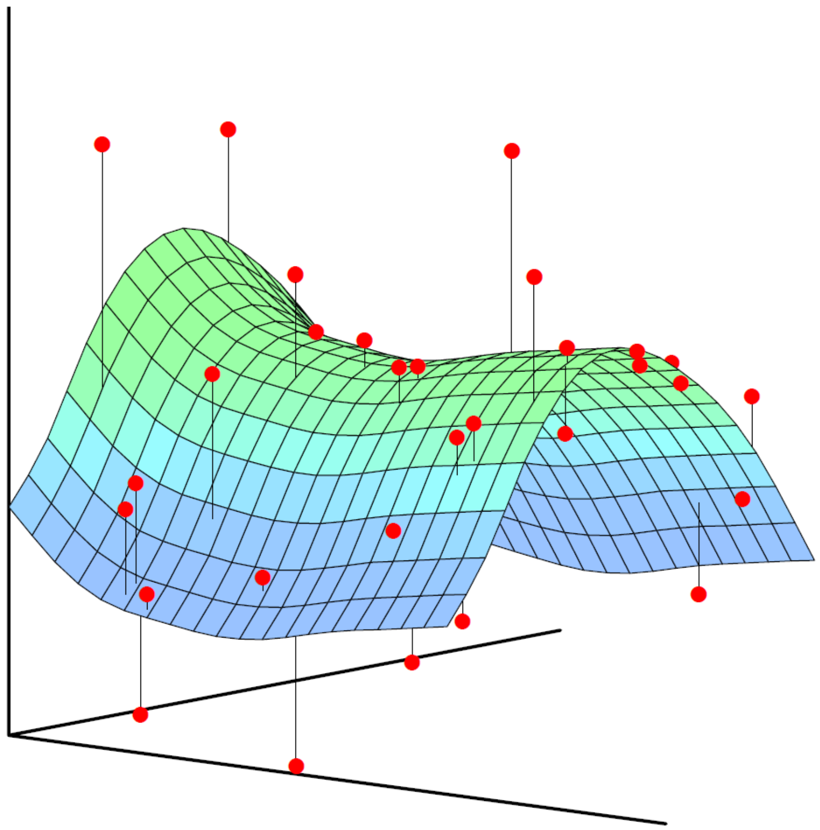
Machine Learning



第11讲 深度学习
Deep Learning

刘 峤

电子科技大学计算机科学与工程学院

11.1 Perceptron

感知机

- 美国学者Rosenblatt在1957年首次提出感知机概念
  ※ 感知机学习算法是Rosenblatt在1958年提出的
- IEEE设立IEEE Frank Rosenblatt Award[1]
- 感知机模型
  ※ 包含一个突触权值可调的神经元
  ※ 属于前向神经网络类型
  ※ 只能区分线性可分的模式，属于线性分类模型

1. https://www.ieee.org/about/awards/tfas/rosenblatt.html

单层感知机

- 单层感知机：Perceptron Learning Algorithm (PLA)

  ※ 是具有单层处理单元的神经网络

  ※ 模拟神经元接受环境信息，通过神经冲动（激活函数）进行信息传递

  ※ 用于求解输入空间的二分类问题：求解分类超平面

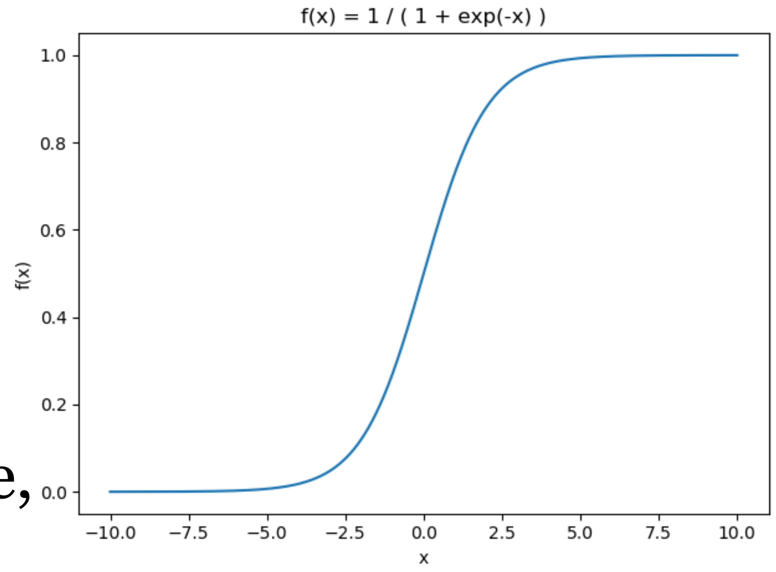
- 设输入向量 $\mathbf{x} \in R^n$，类别标记 $y \in \{-1, +1\}$，则感知机模型表示为：

$$f(\mathbf{x}) = sign(\mathbf{w} \cdot \mathbf{x} + b)$$

- 思考：$\mathbf{w} \cdot \mathbf{x} + b = 0$ 的几何意义是什么?
- 思考：$\mathbf{w}$ 的几何意义是什么?
- 思考：感知机与逻辑斯蒂回归的联系与区别在哪里?

Logistic Regression

- Let's write $p(X) = \Pr(Y = 1|X)$ for short and consider using balance to predict default. Logistic regression uses the form



$$p(X) = \frac{e^{\beta_0+\beta_1 X}}{1+e^{\beta_0+\beta_1 X}}$$

(e ≈ 2.71828 is a mathematical constant [Euler's number.])

- It is easy to see that no matter what values β_0, β_1 or X take,

※ $p(X)$ will have values between 0 and 1.

- A bit of rearrangement gives

$$\log\left(\frac{p(X)}{1-p(X)}\right) = \beta_0 + \beta_1 X$$

- This monotone transformation is called the *log odds* or *logit* transformation of $p(X)$

单层感知机的求解

- 思考：若发生误分类的情况，误分类点到超平面的距离是?

$$d = -\frac{1}{\|\mathbf{w}\|_2} y^i(\mathbf{w} \cdot \mathbf{x}^i + b)$$

- 思考：若发生误分类的情况，误分类点到超平面的距离总和是?

$$\mathcal{L}(\mathbf{w}, b) = -\sum_{\mathbf{x}^i \in D'} y^i(\mathbf{w} \cdot \mathbf{x}^i + b)$$

- 其中: D' 表示模型当前误分类的点的集合
- 思考：如何最小化该损失函数?

$$\nabla_{\mathbf{w}} \mathcal{L}(\mathbf{w}, b) = -\sum_{\mathbf{x}^i \in D'} y^i \mathbf{x}^i \qquad \mathbf{w} = \mathbf{w} + \eta \sum_{\mathbf{x}^i \in D'} y^i \mathbf{x}^i$$

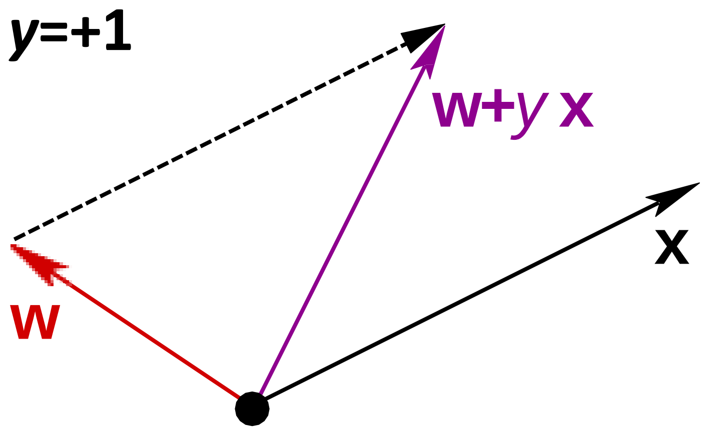
$$\nabla_b \mathcal{L}(\mathbf{w}, b) = -\sum_{\mathbf{x}^i \in D'} y^i \qquad b = b + \eta \sum_{\mathbf{x}^i \in D'} y^i$$

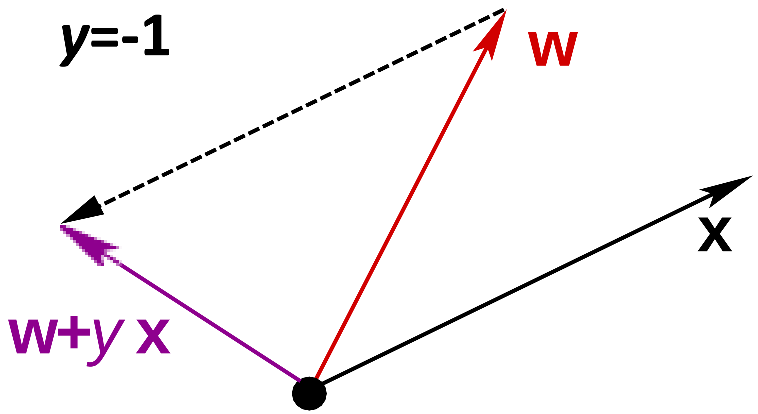
Batch Gradient Descent

单层感知机的求解

- 随机梯度下降（Stochastic Gradient Descent）

$$\mathbf{w}_{t+1} = \mathbf{w}_t + \eta y^i \mathbf{x}^i \qquad b_{t+1} = b_t + \eta y^i$$





Practical Implementation of PLA

- Start from some w_o (say, 0), and 'correct' its mistakes on D
- For t = 0, 1, . . .

- find the next mistake of w_t called (x_t, y_t)

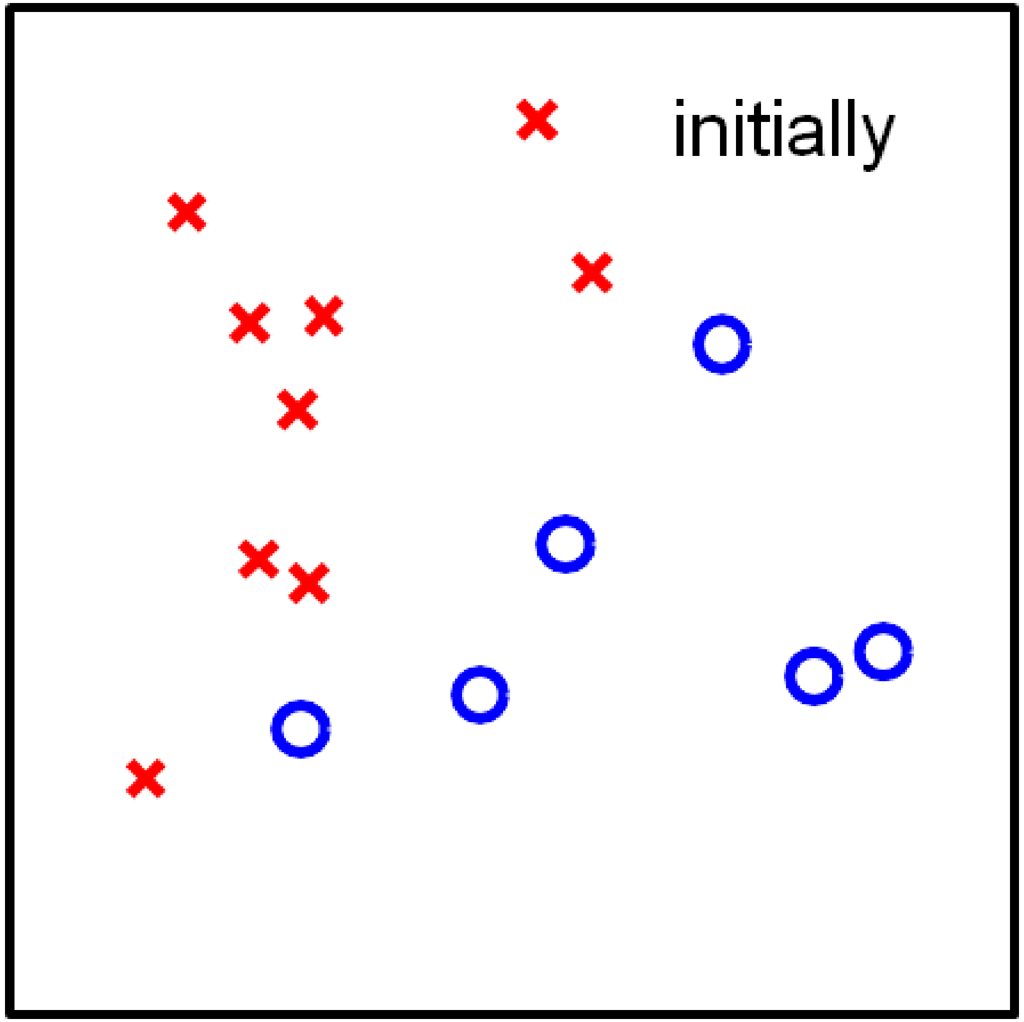
$$sign(\mathbf{w}_t \cdot \mathbf{x}_t + b) \neq y_t$$

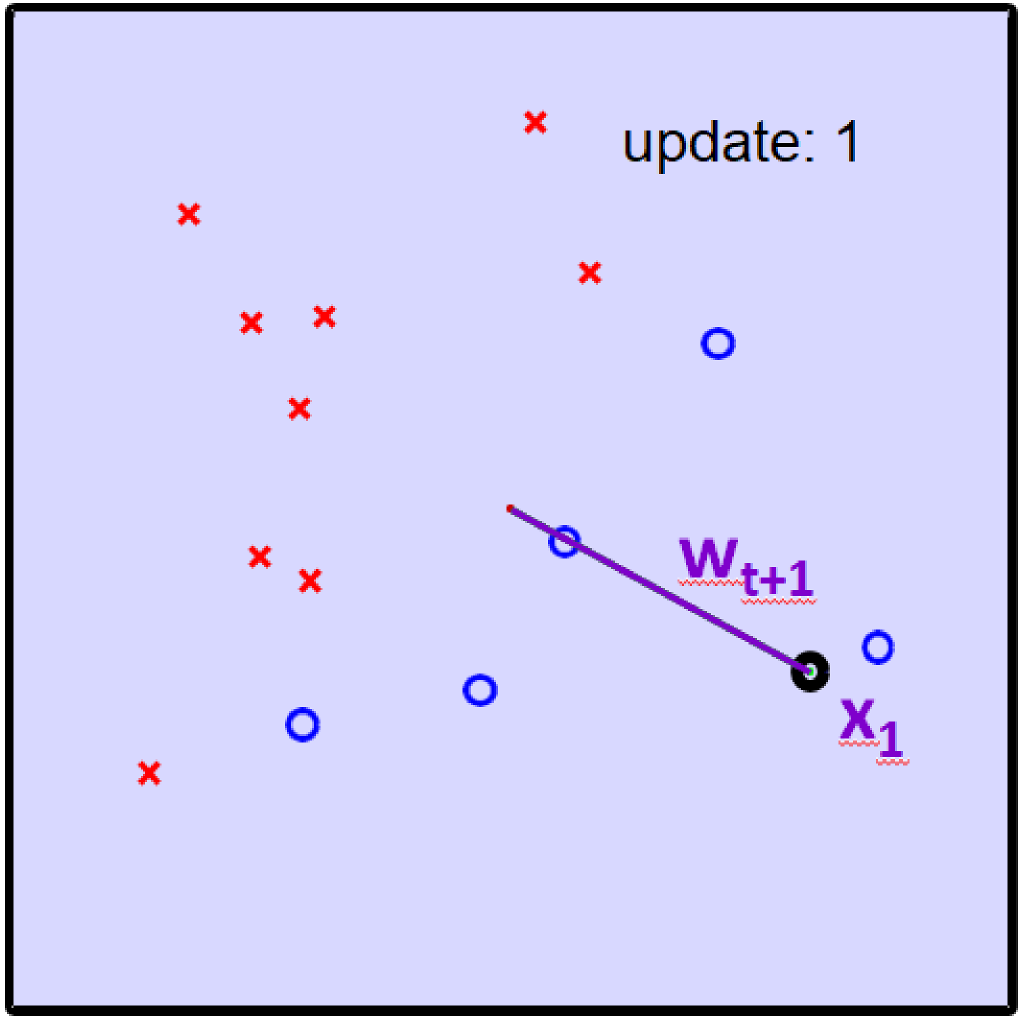
- correct the mistake by

$$\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + y_t \mathbf{x}_t$$

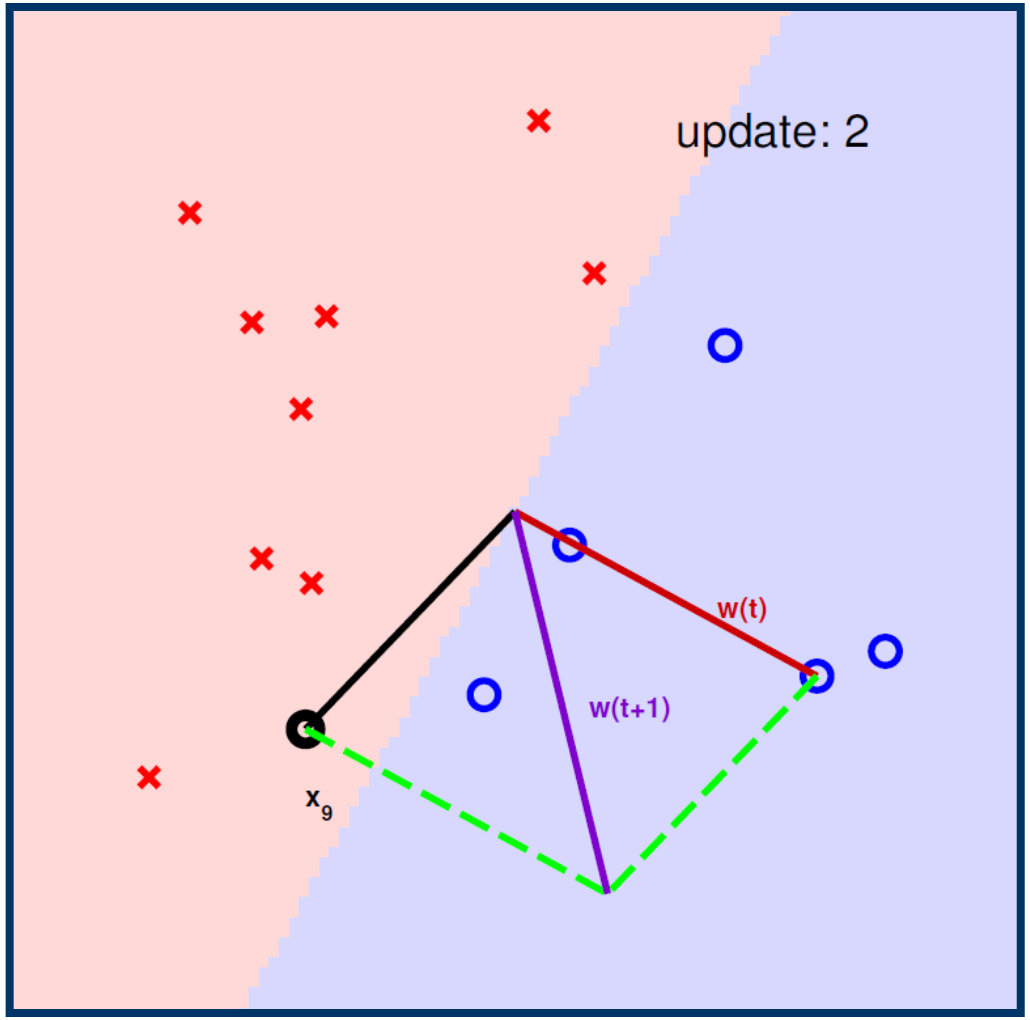
- … until a full cycle of not encountering mistakes

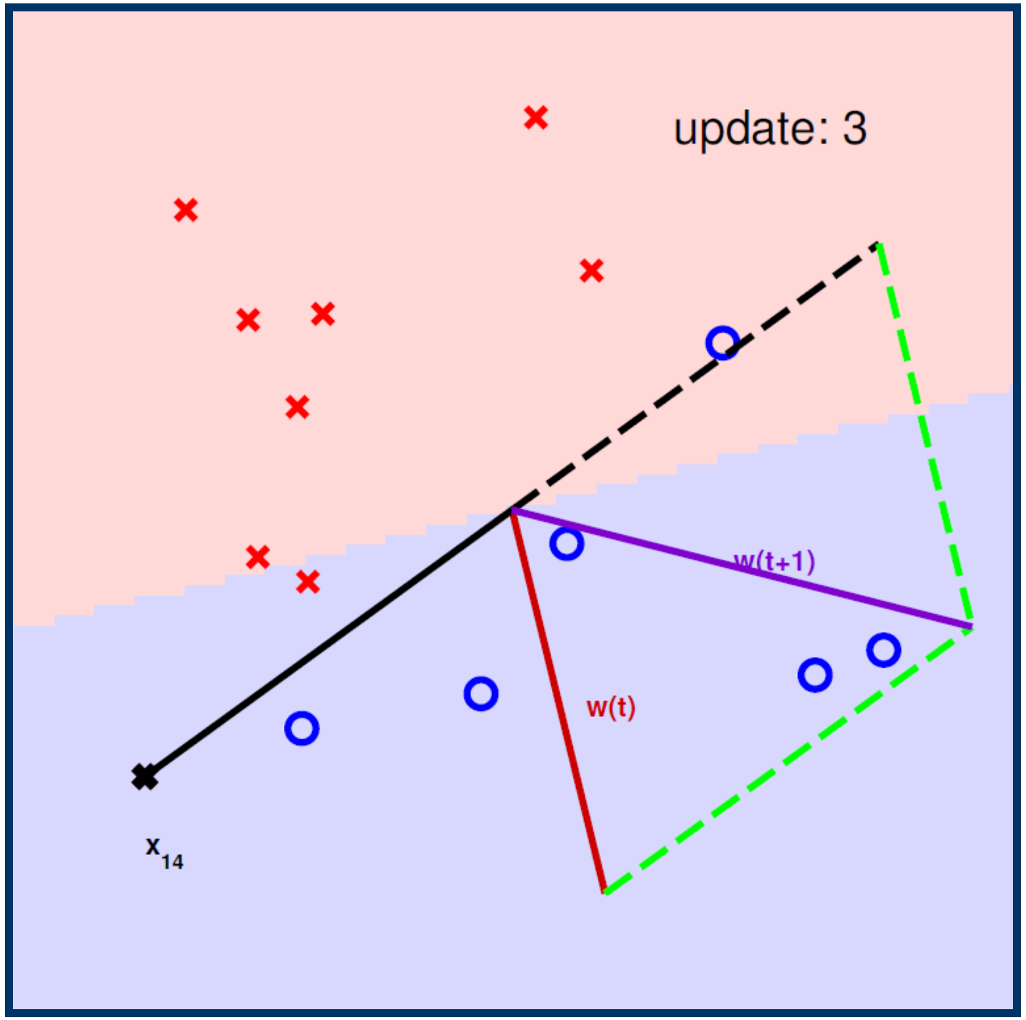
Seeing is Believing



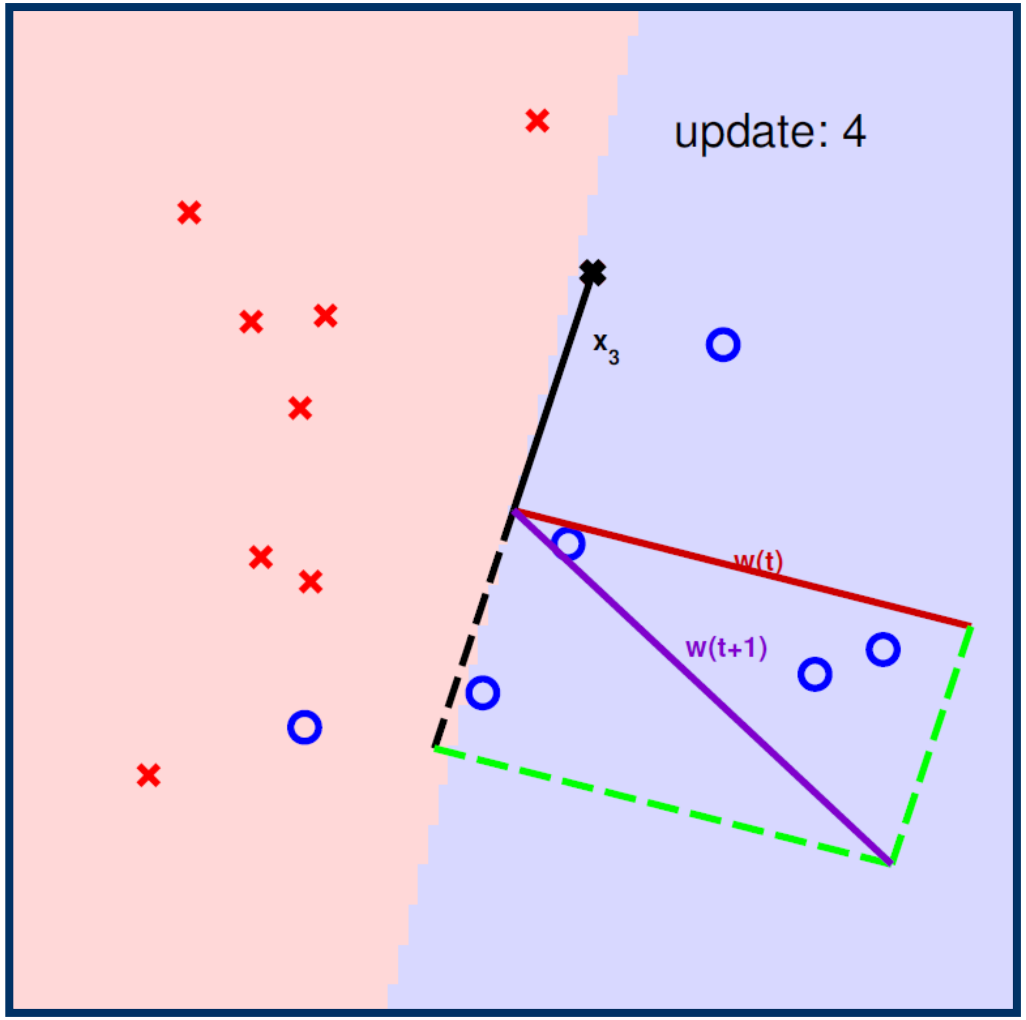


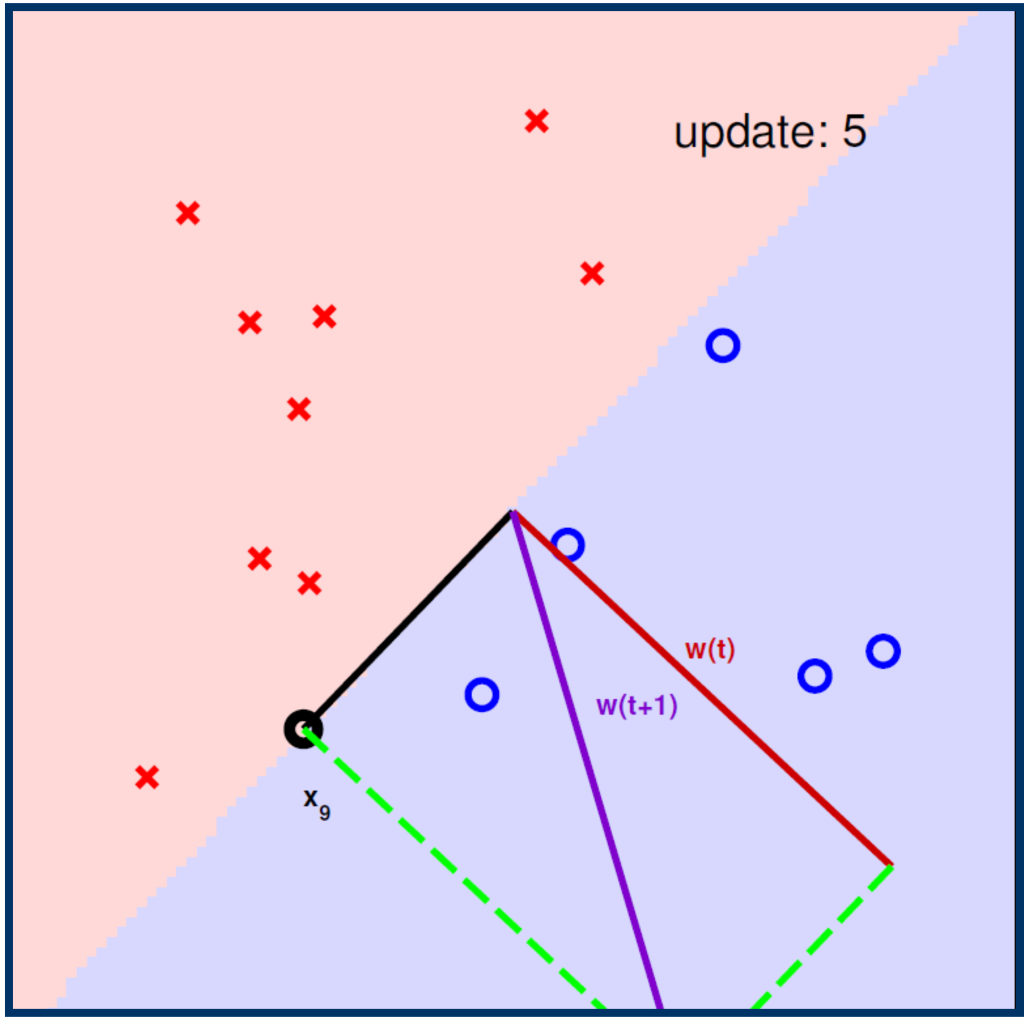
Seeing is Believing



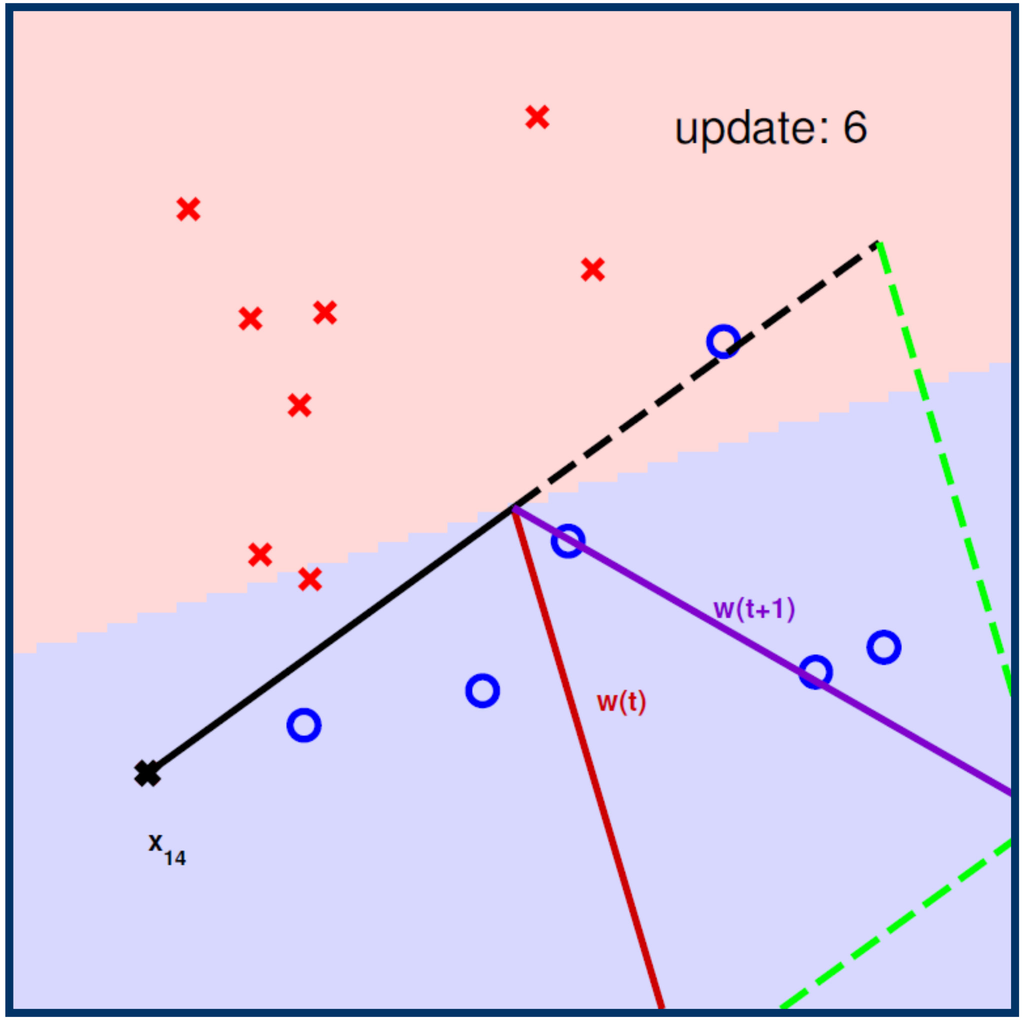


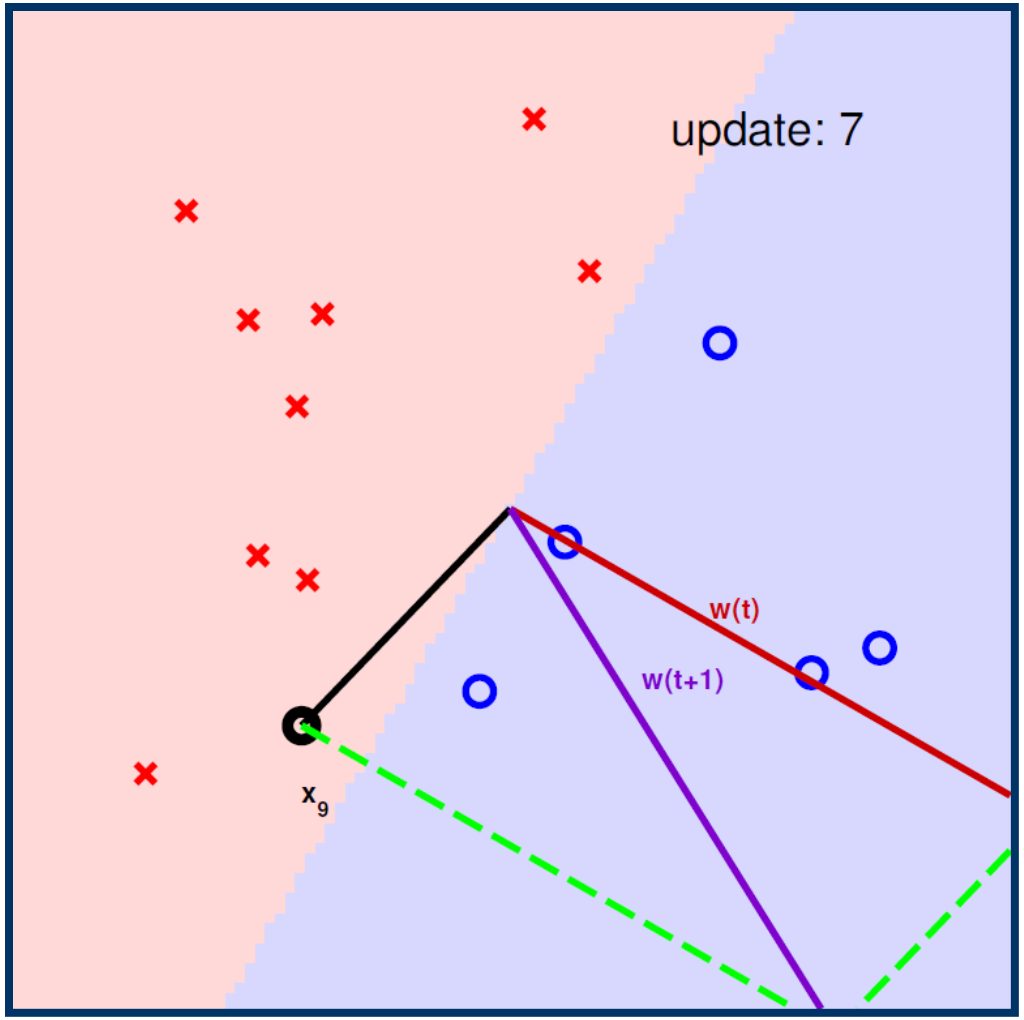
Seeing is Believing



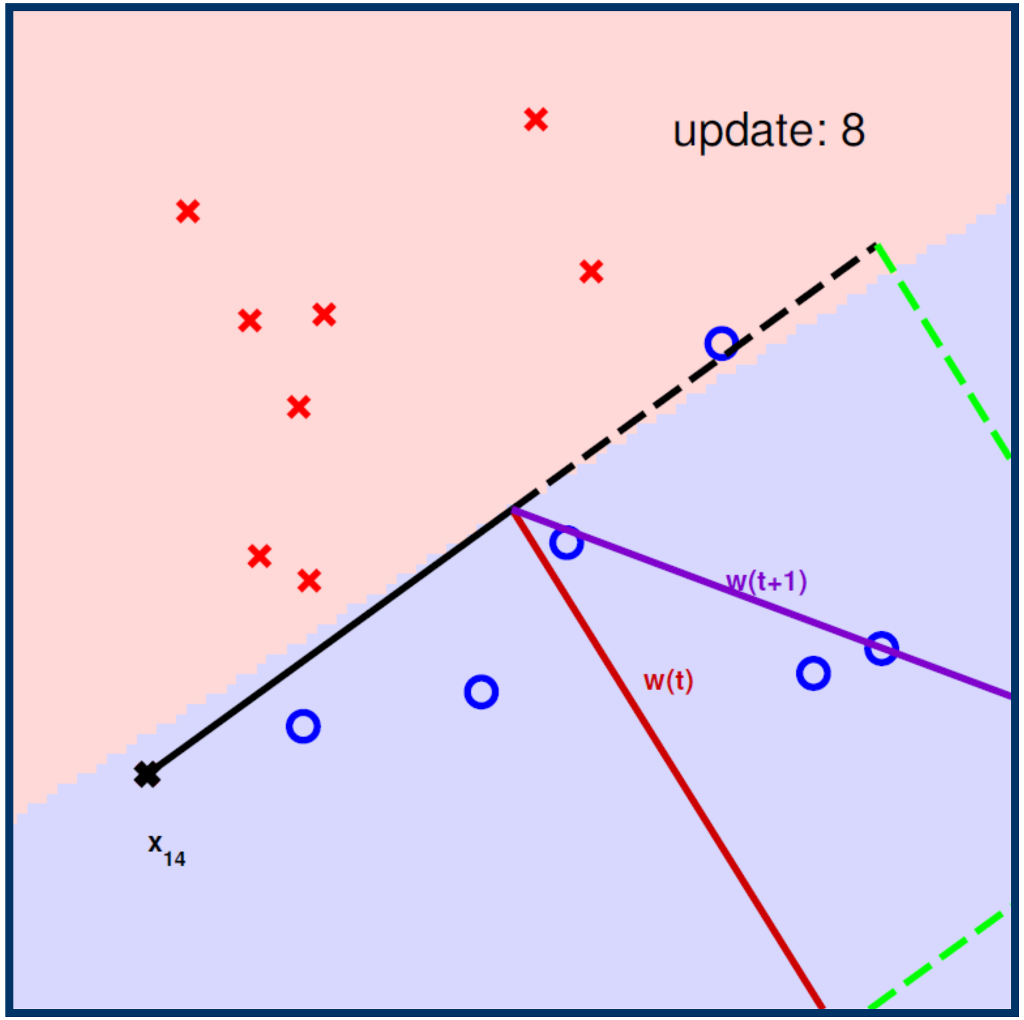


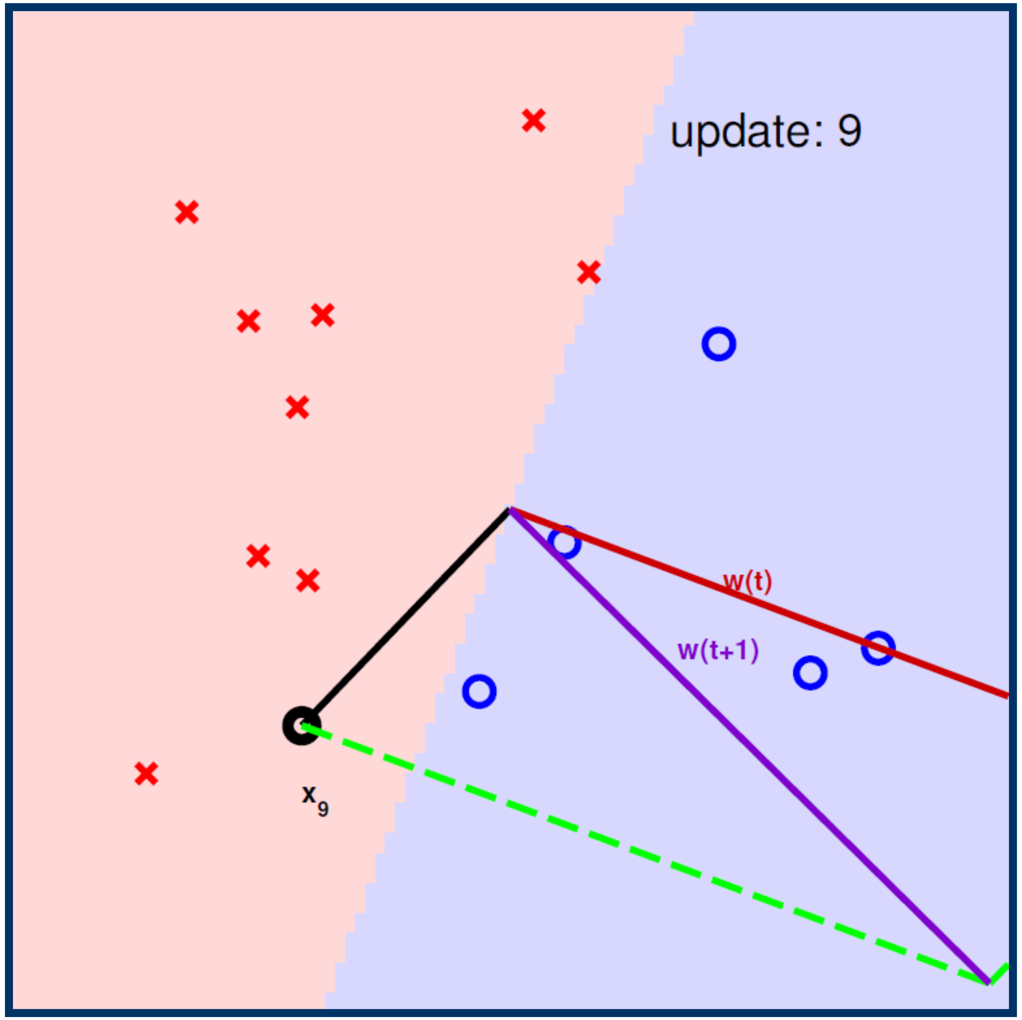
Seeing is Believing



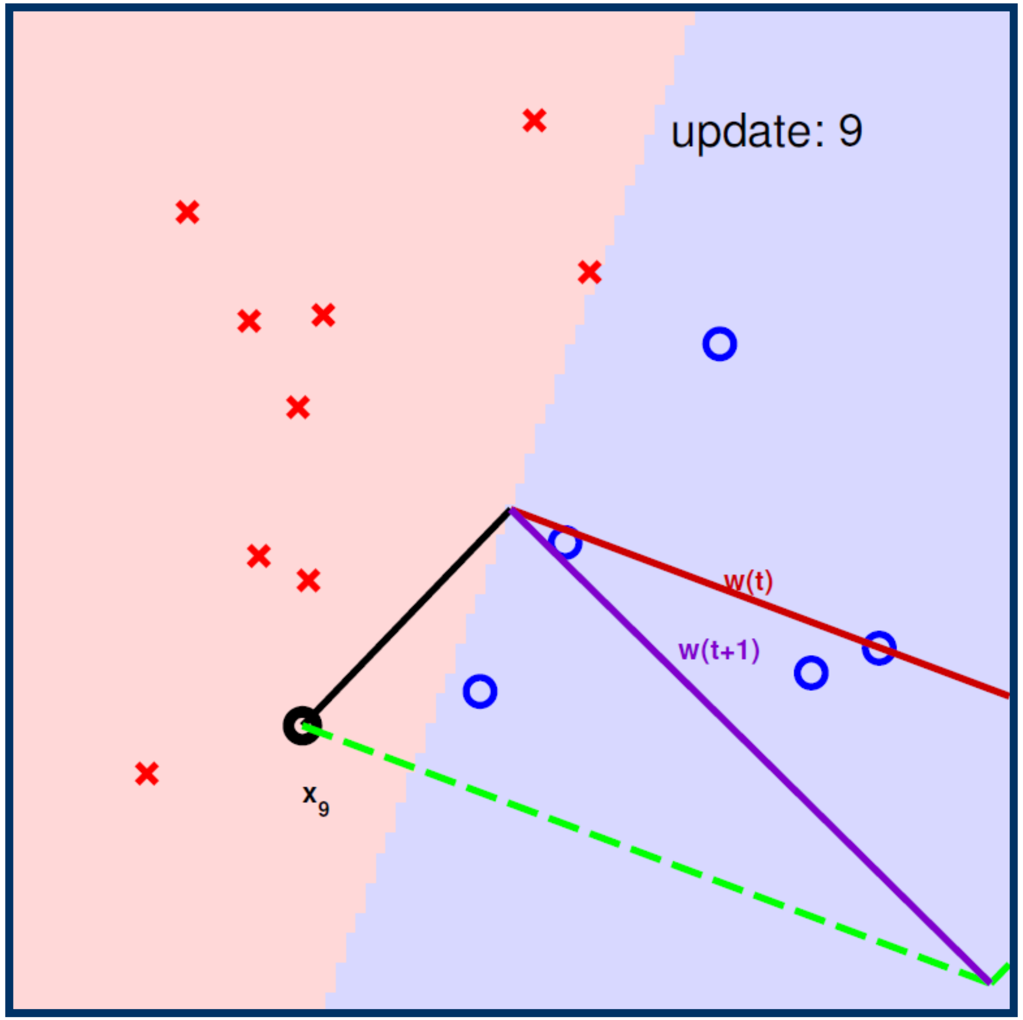


Seeing is Believing





Seeing is Believing



11.2 Deep Learning

Ups and downs of Deep Learning

- 1958: Perceptron (linear model)
- 1969: Perceptron has limitation
- 1980s: Multi-layer perceptron

  ※ Do not have significant difference from DNN today
- 1986: **Backpropagation**

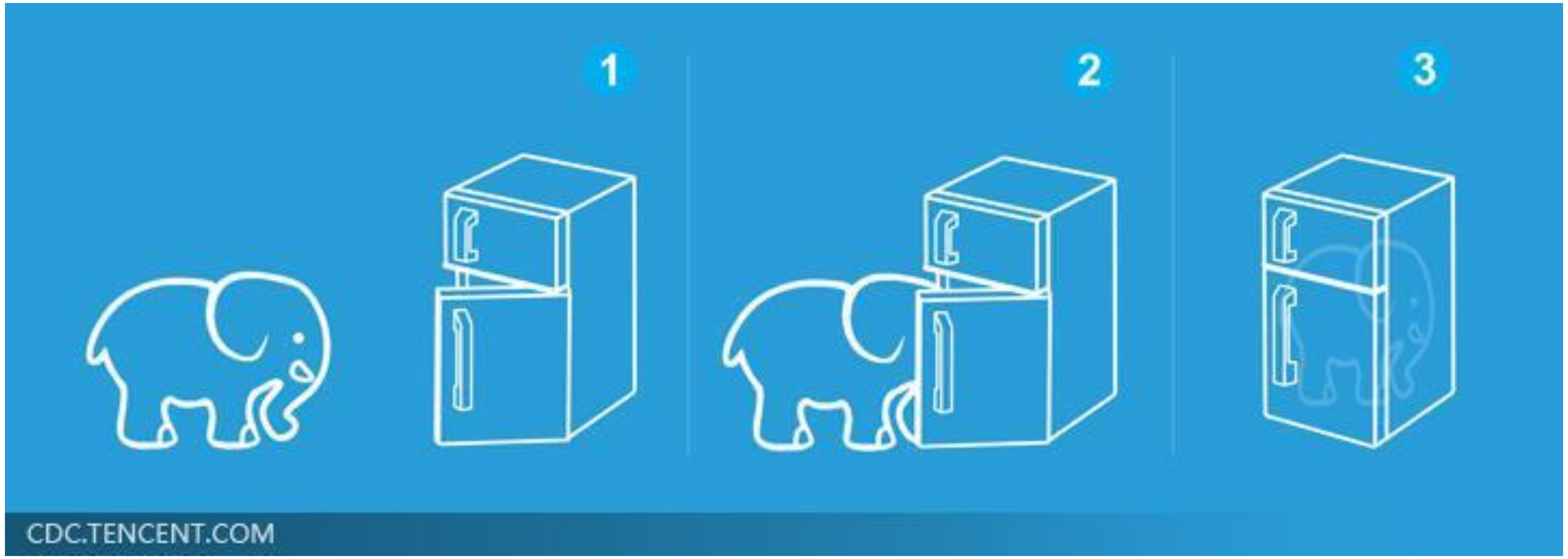
  ※ Usually more than 3 hidden layers is not helpful
- 1989: 1 hidden layer is "good enough", why deep?
- 2006: RBM initialization (breakthrough)
- 2009: GPU
- 2011: Start to be popular in speech recognition
- 2012: win ILSVRC image competition

Three Steps for Deep Learning

- Deep Learning is so simple ……

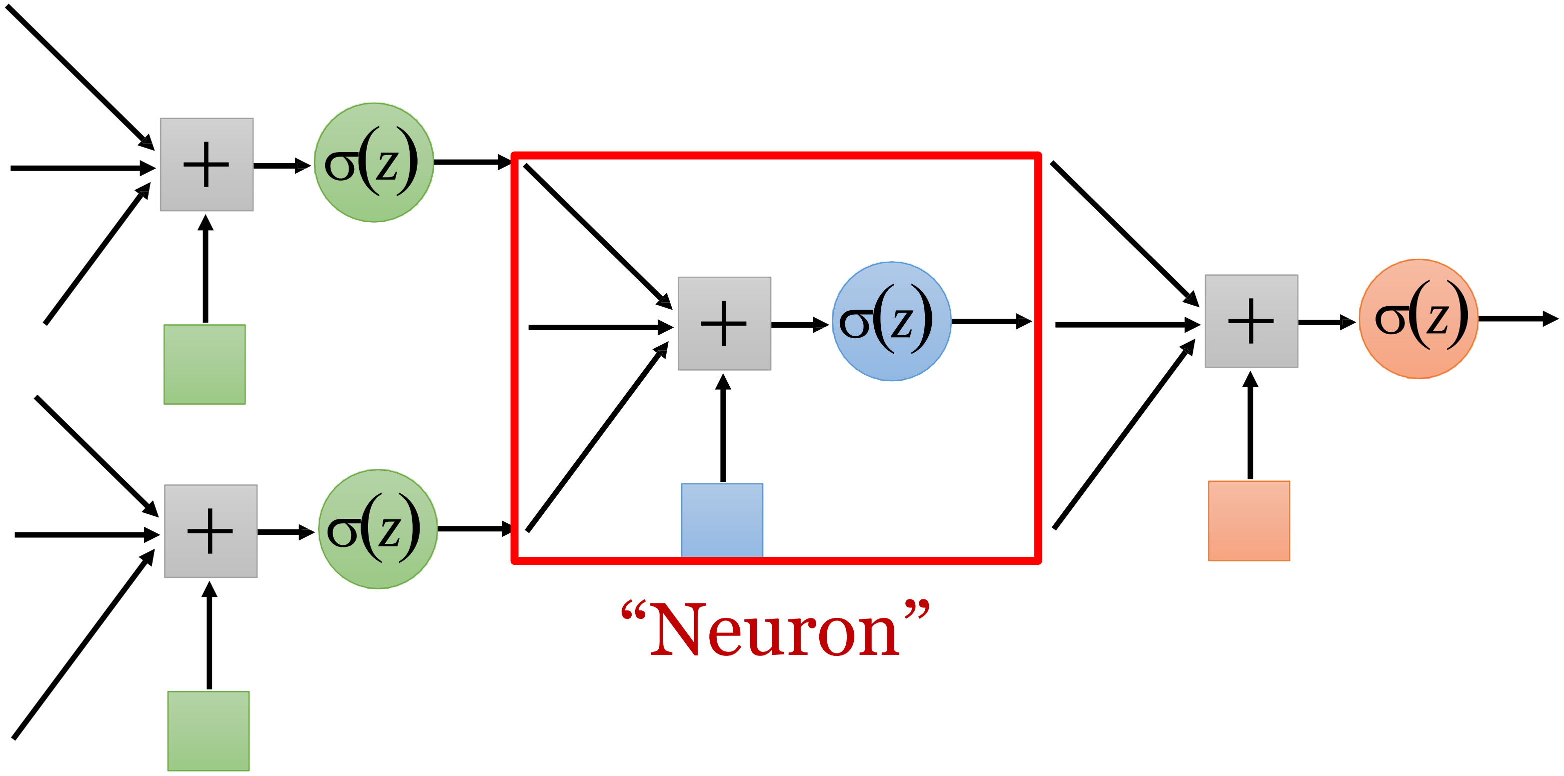
Step 1: define a set of function → **Step 2: goodness of function** → **Step 3: pick the best function**

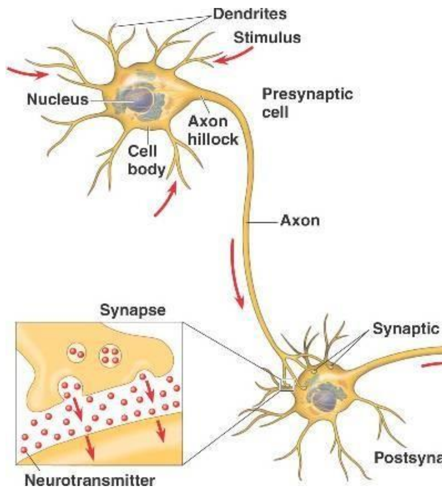
Neural Network



Neural Network

- Neural Network: Different connection leads to different network structures



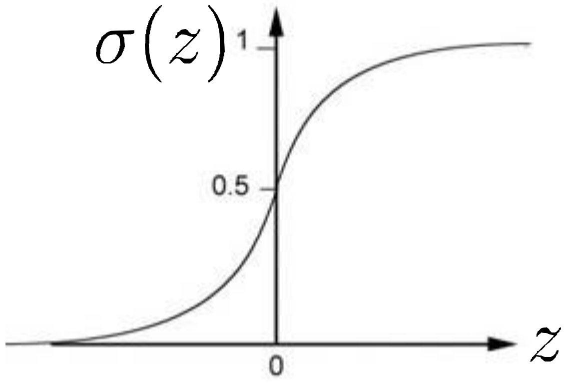


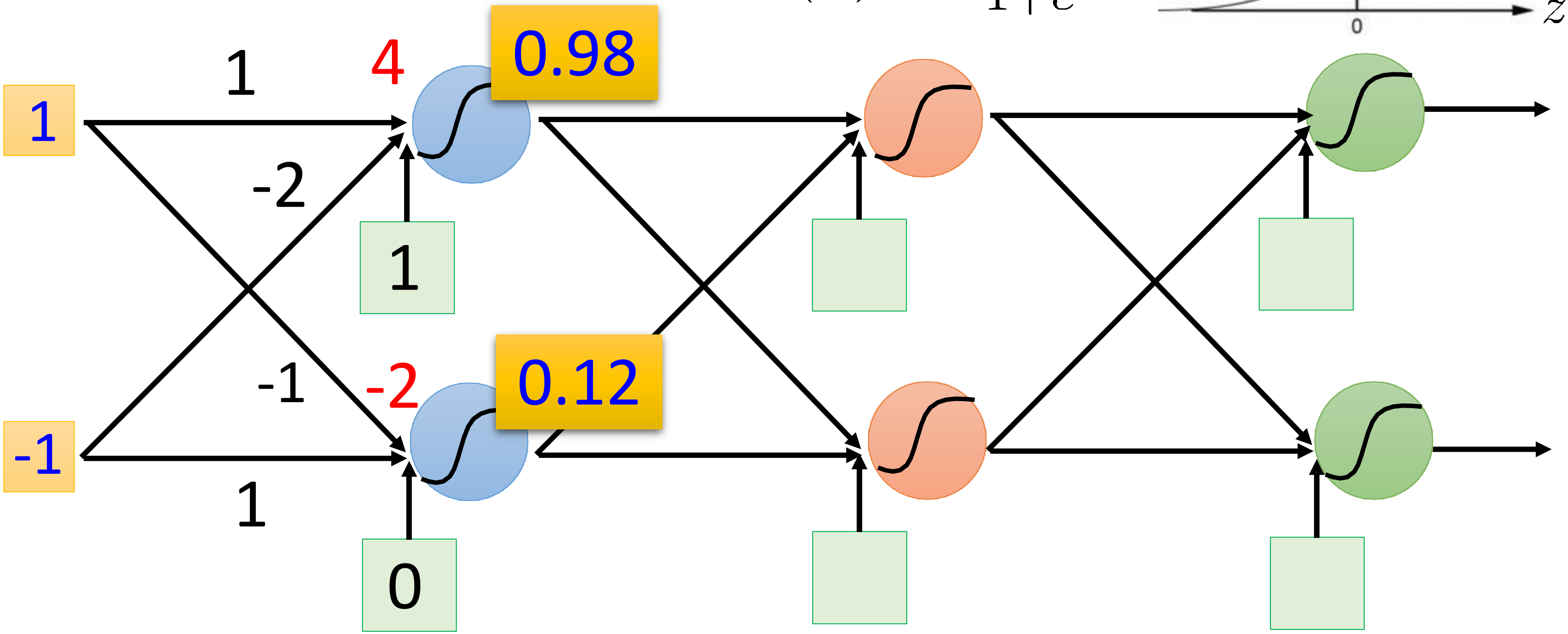
- Network parameter θ: all the weights and biases in the "neurons"

Fully Connect Feedforward Network

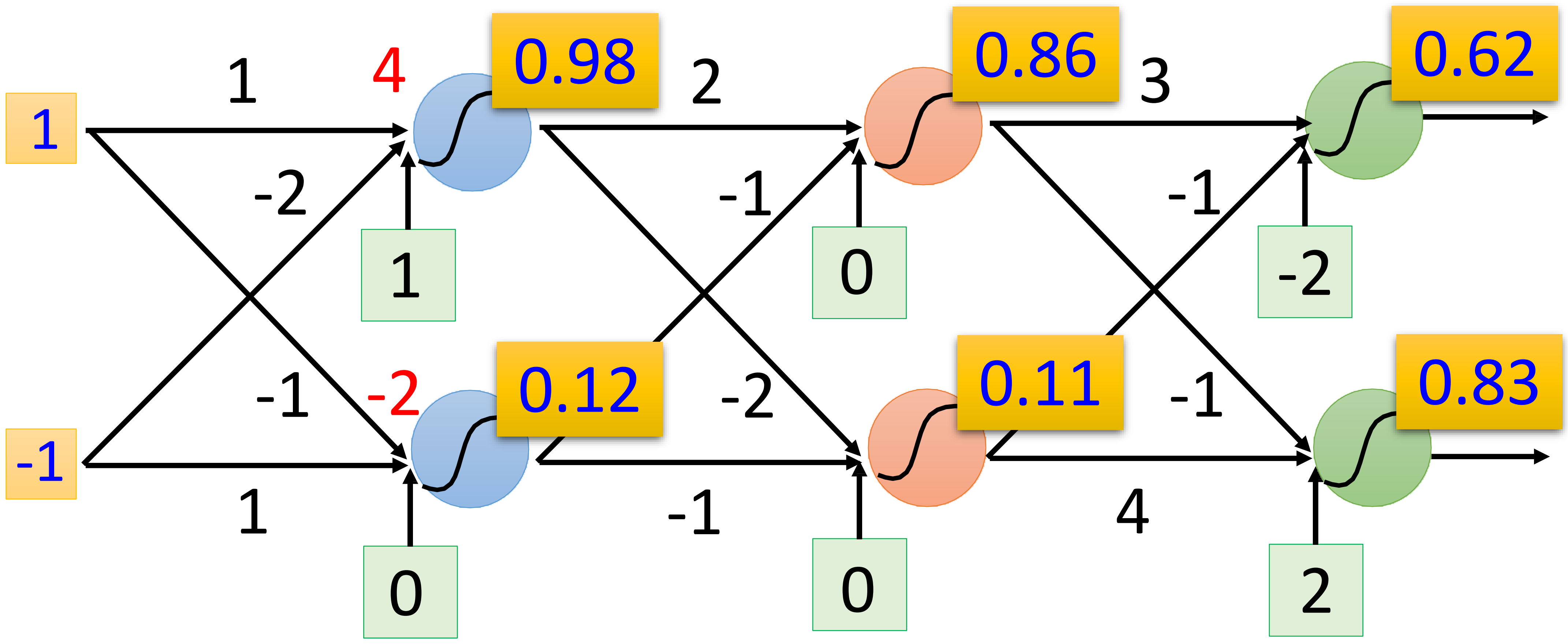
Sigmoid Function

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

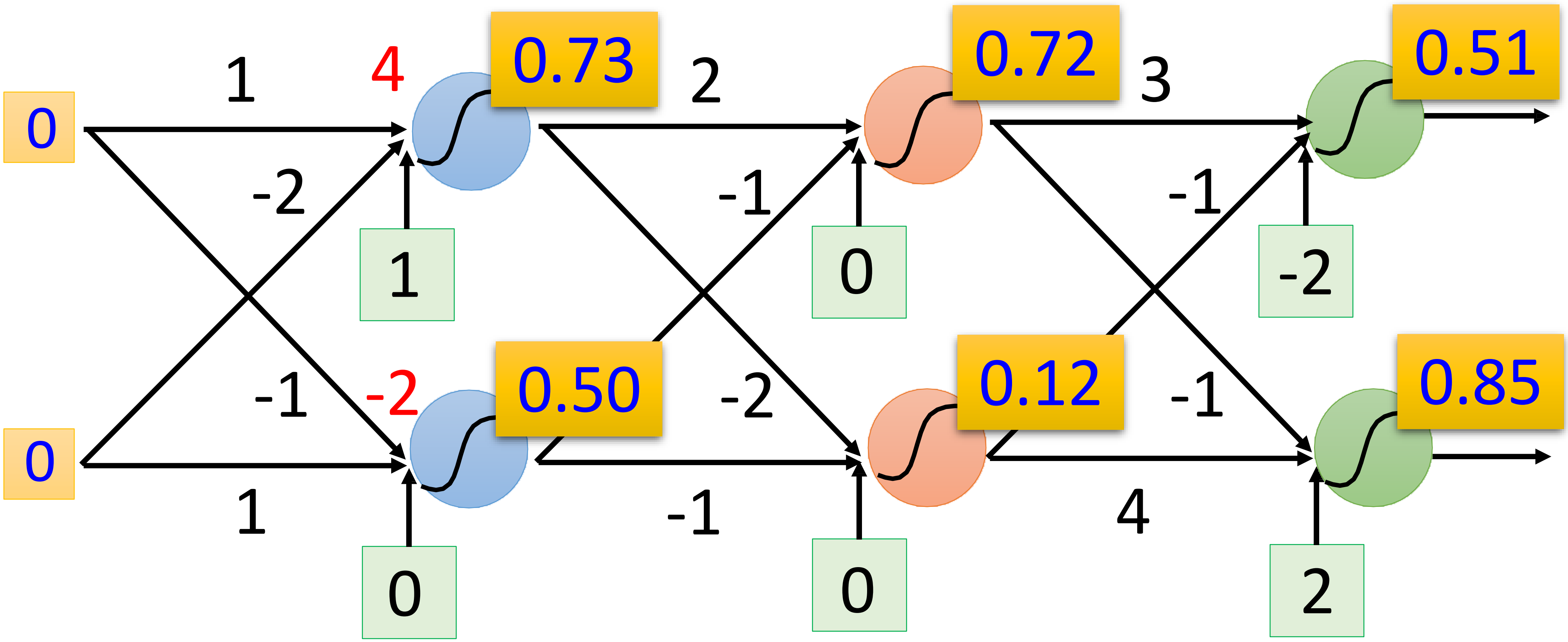




Fully Connect Feedforward Network



Fully Connect Feedforward Network



This is a function： Input vector, output vector

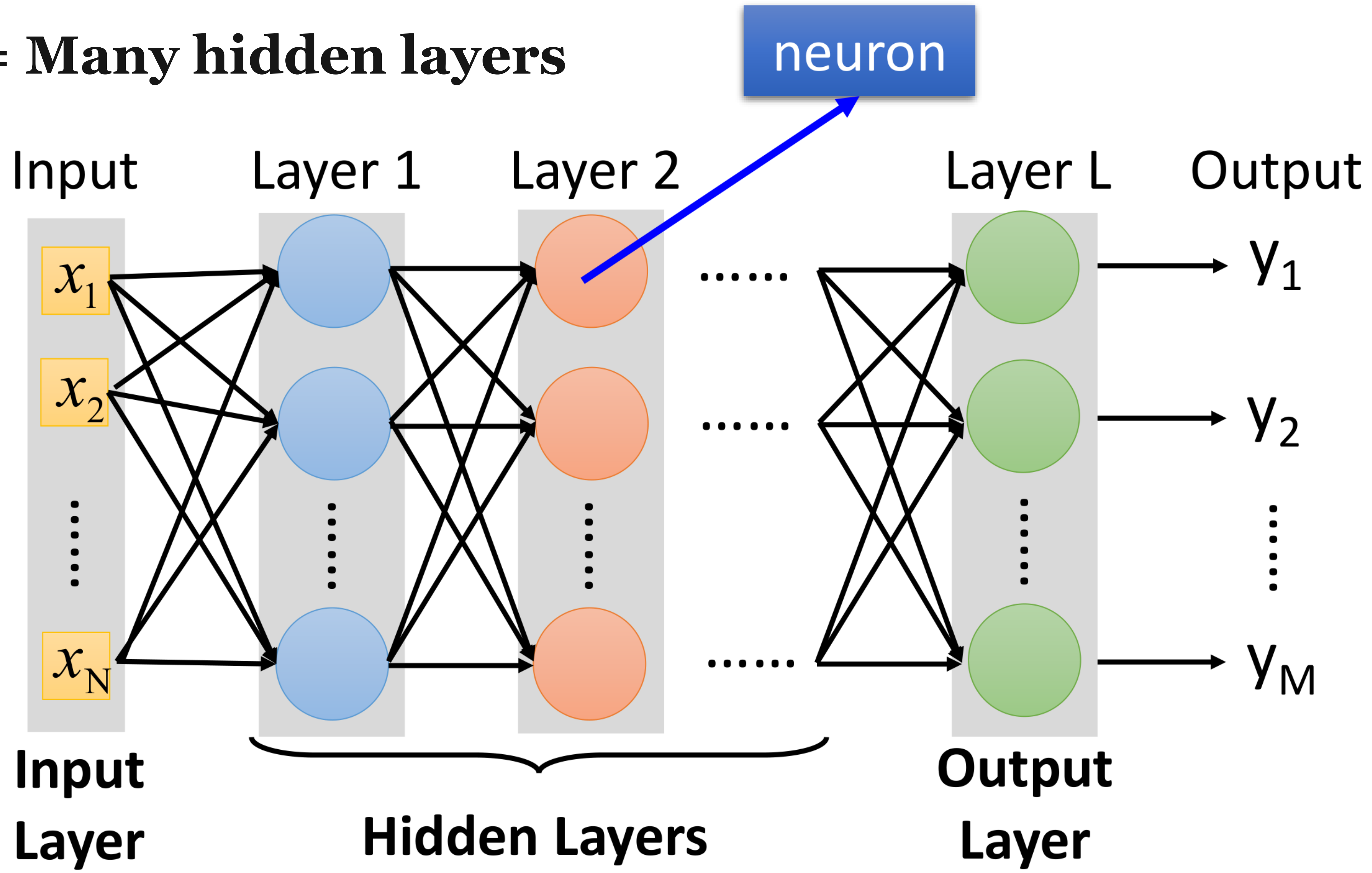
$$f\left(\begin{bmatrix}1\\-1\end{bmatrix}\right)=\begin{bmatrix}0.62\\0.83\end{bmatrix}$$

Given network structure, define a function set

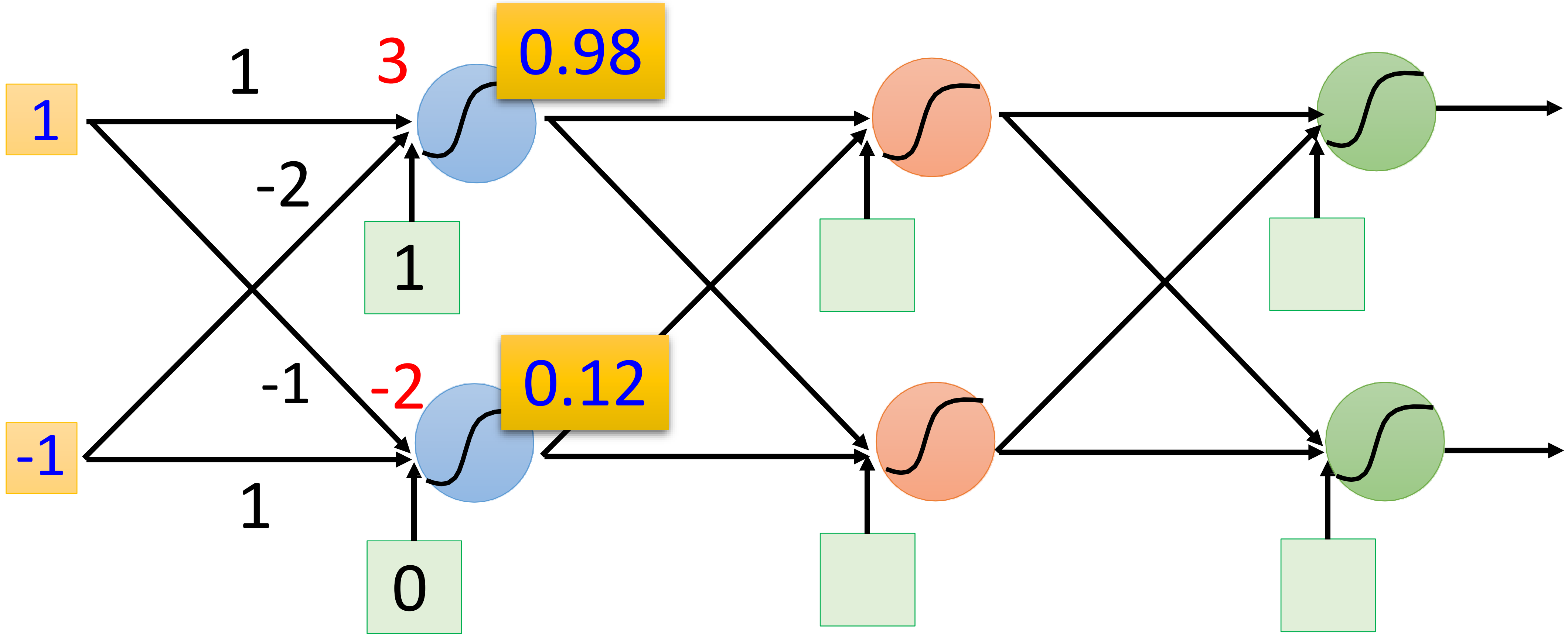
$$f\left(\begin{bmatrix}0\\0\end{bmatrix}\right)=\begin{bmatrix}0.51\\0.85\end{bmatrix}$$

Fully Connect Feedforward Network

Deep = Many hidden layers



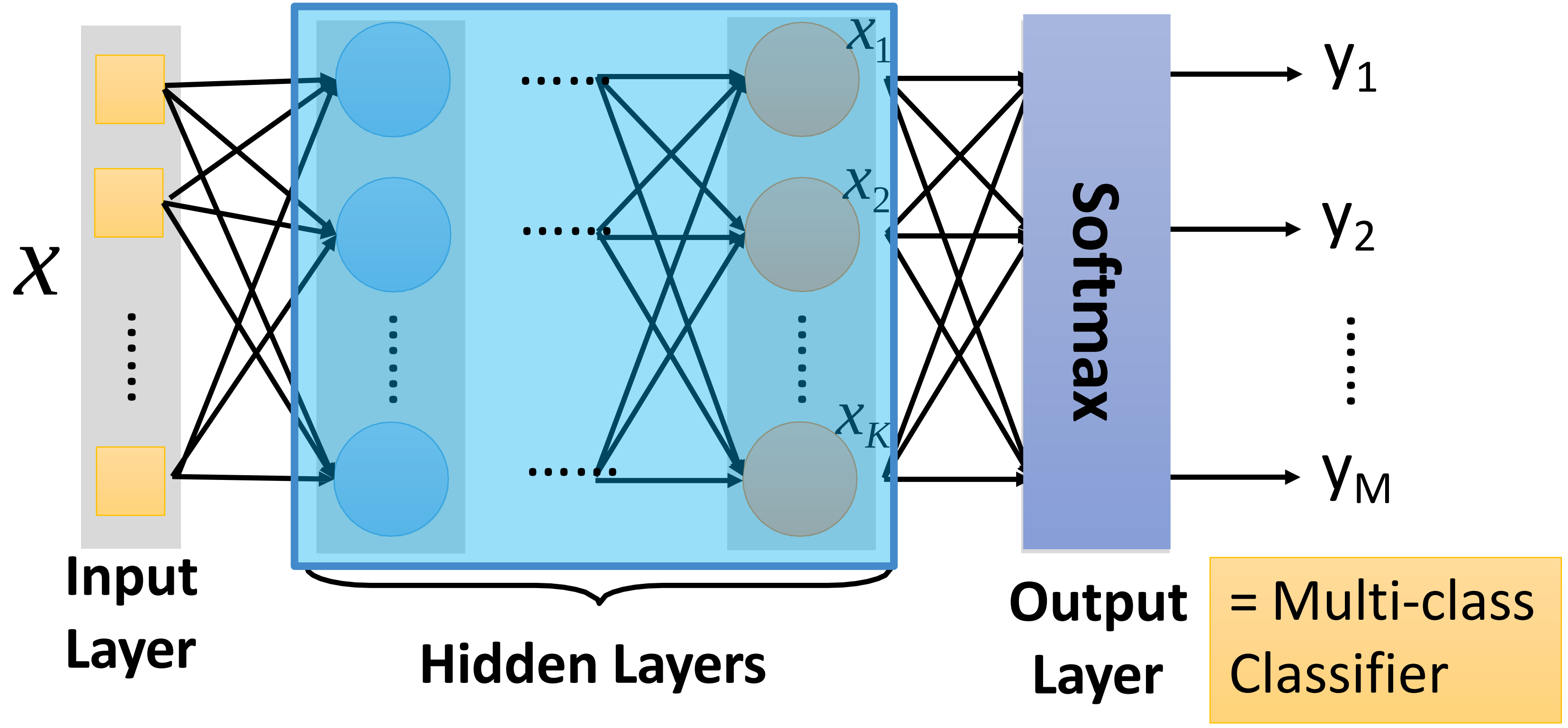
Matrix Operation



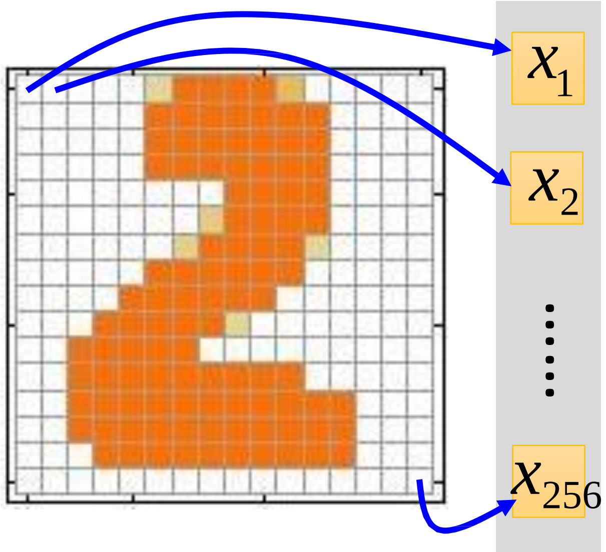
$$\sigma\left(\begin{bmatrix} 1 & -2 \\ -1 & 1 \end{bmatrix}\begin{bmatrix} 1 \\ -1 \end{bmatrix} + \begin{bmatrix} 1 \\ 0 \end{bmatrix}\right) = \begin{bmatrix} 0.98 \\ 0.12 \end{bmatrix}$$

Output Layer

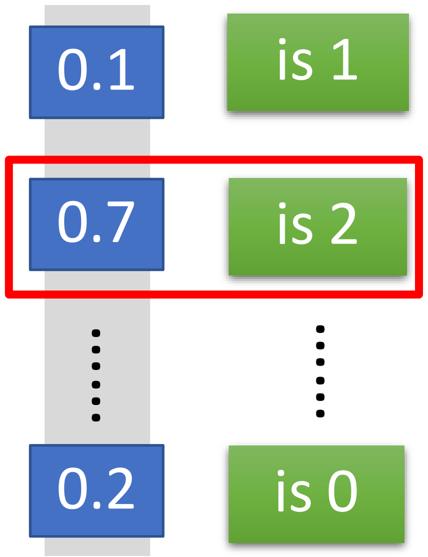
Feature extractor replacing feature engineering



Input

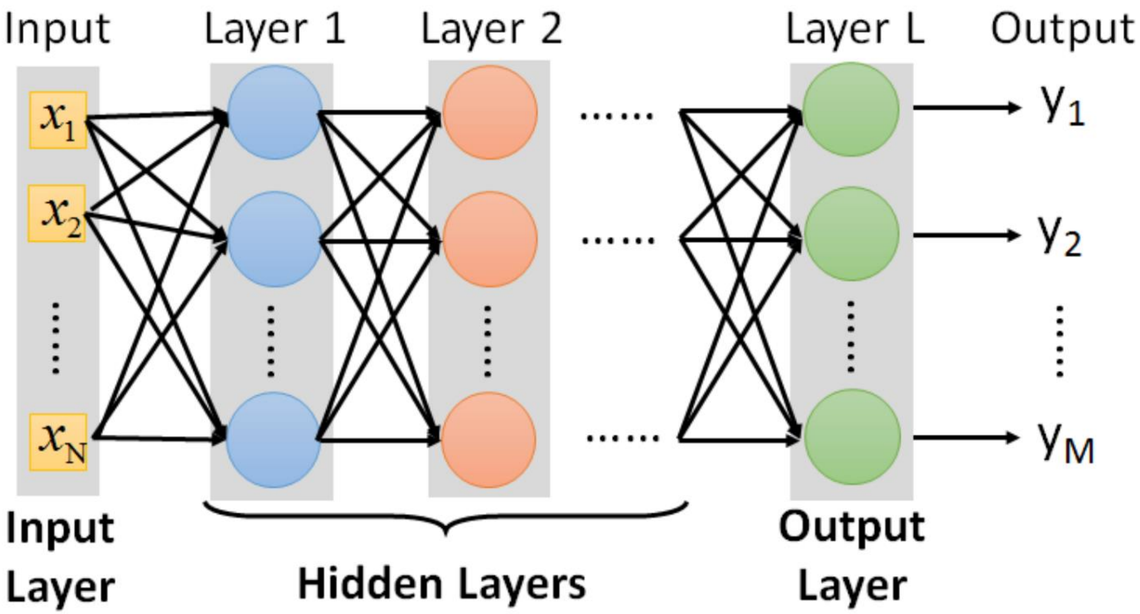


Output



Each dimension represents the confidence of a digit.

FAQ



- Q: How many layers? How many neurons for each layer?

  ※ Trial and Error + Intuition

- Q: Can the structure be automatically determined?

  ※ E.g. Evolutionary Artificial Neural Networks

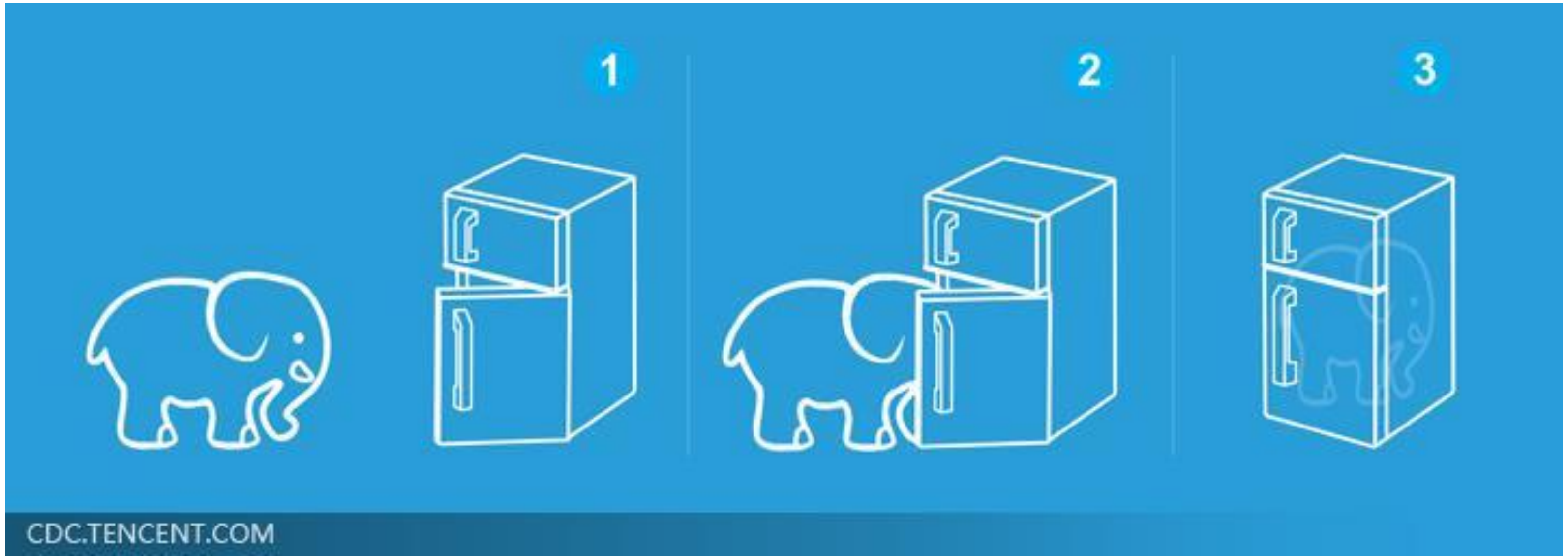
- Q: Can we design the network structure?

Three Steps for Deep Learning

- Deep Learning is so simple ……

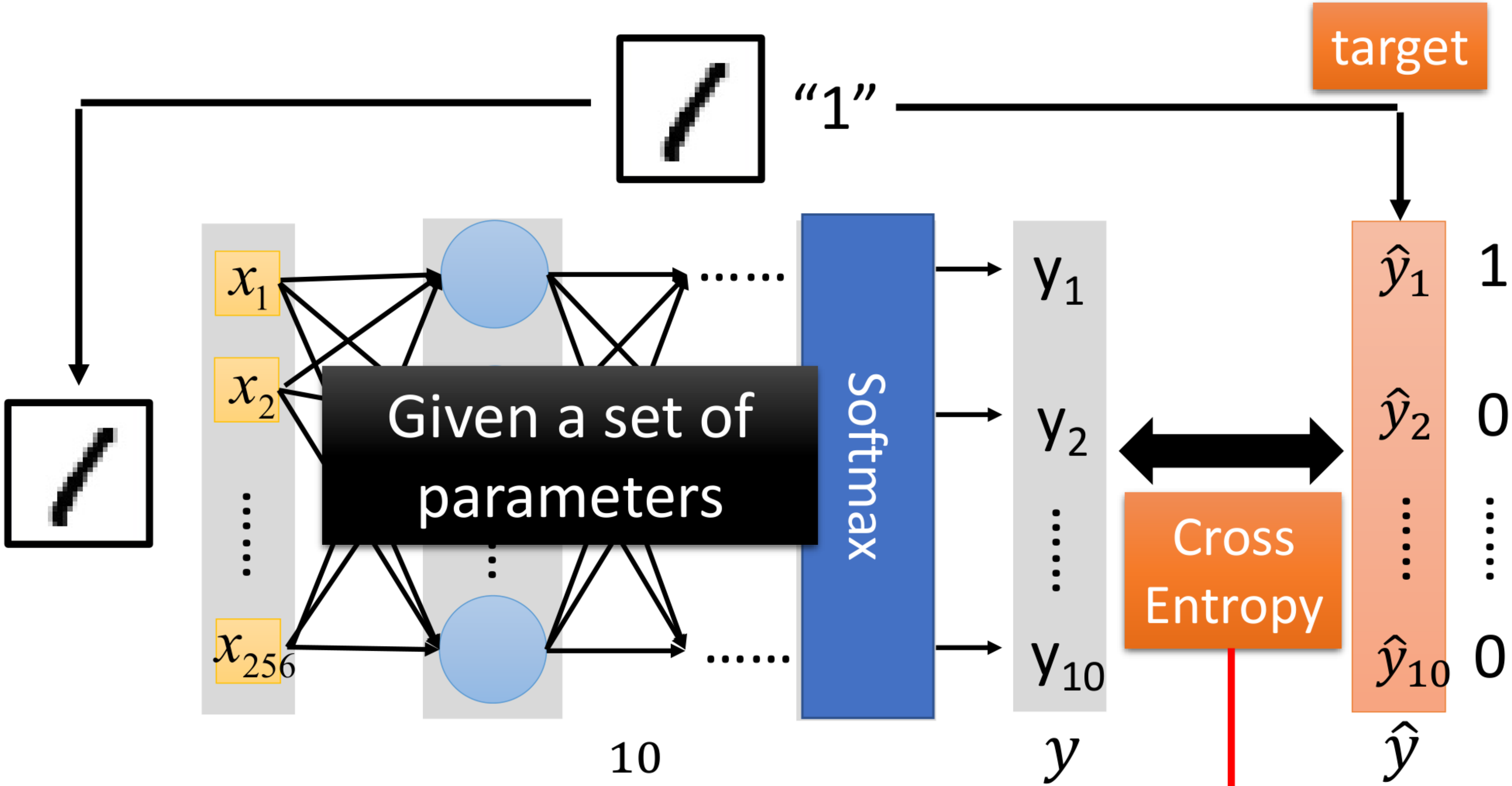
Step 1: define a set of function → **Step 2: goodness of function** → **Step 3: pick the best function**

Neural Network



11.3 The Loss Function

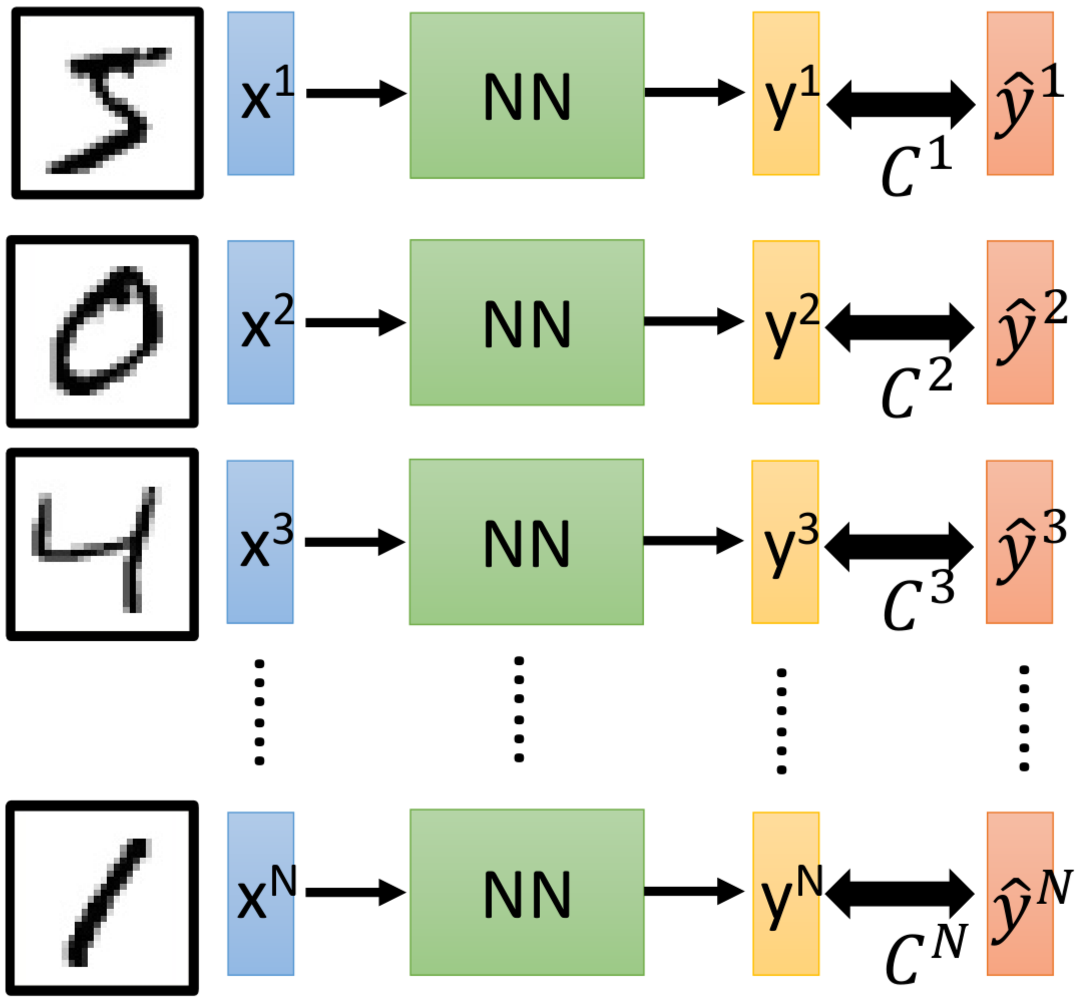
Loss for an Example



$$C(y, \hat{y}) = -\sum_{i=1}^{10} \hat{y}_i \ln y_i$$

Total Loss

- For all training data …



- Total Loss:

$$L = \sum_{n=1}^{N} C^n$$

$\Downarrow$

Find a function in function set that minimizes total loss L

$\Downarrow$

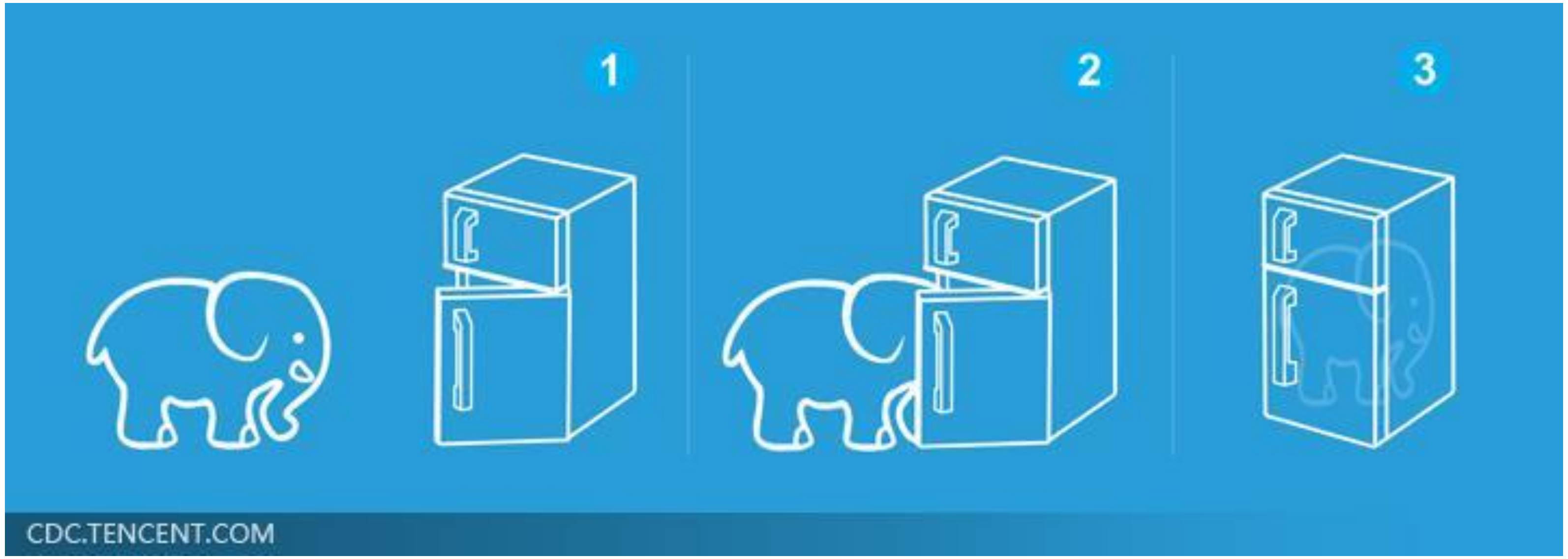
Find the network parameters $\boldsymbol{\theta}*$ that minimize total loss L

Three Steps for Deep Learning

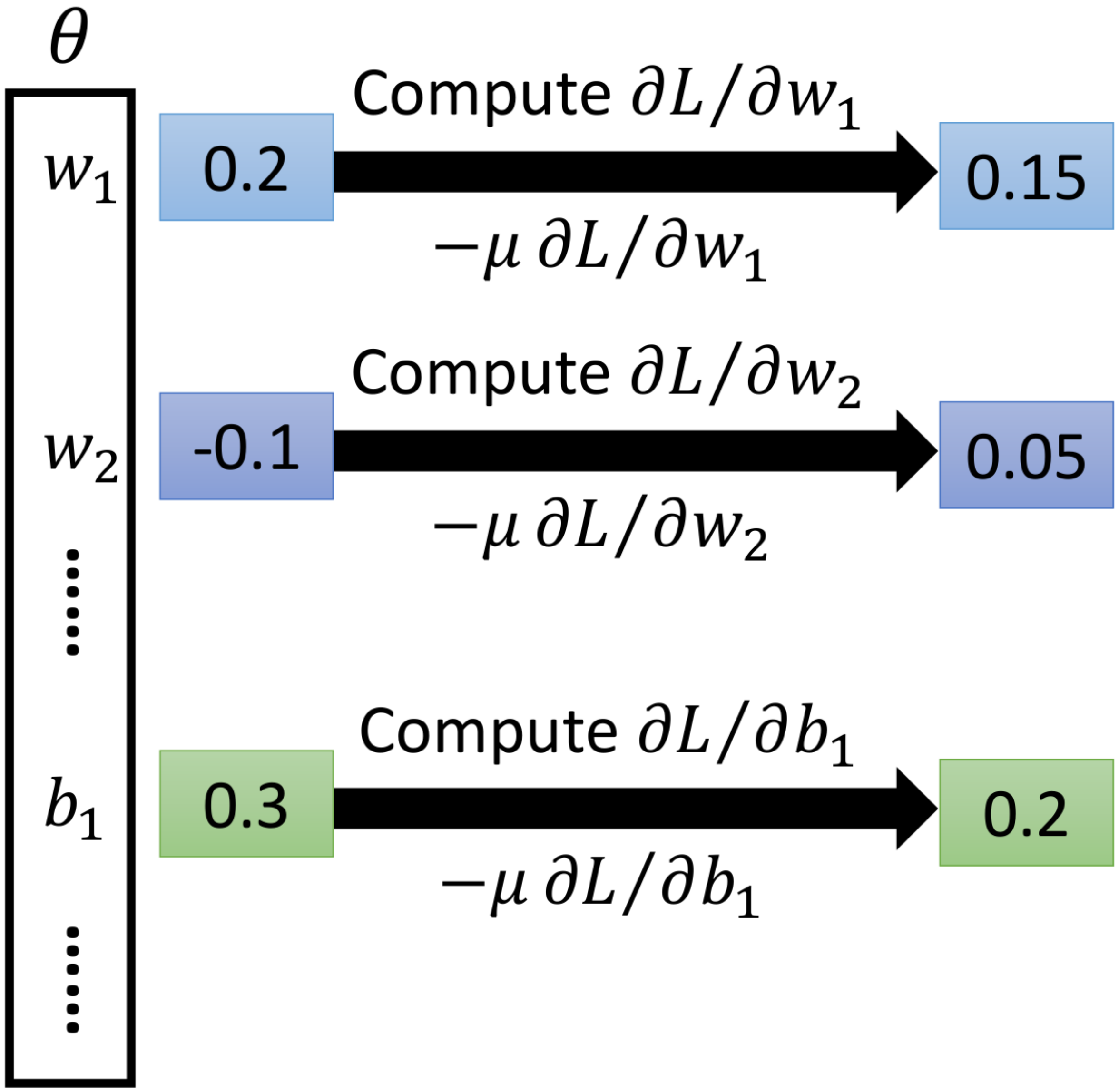
- Deep Learning is so simple ……

Step 1: define a set of function → Step 2: goodness of function → Step 3: pick the best function

Neural Network

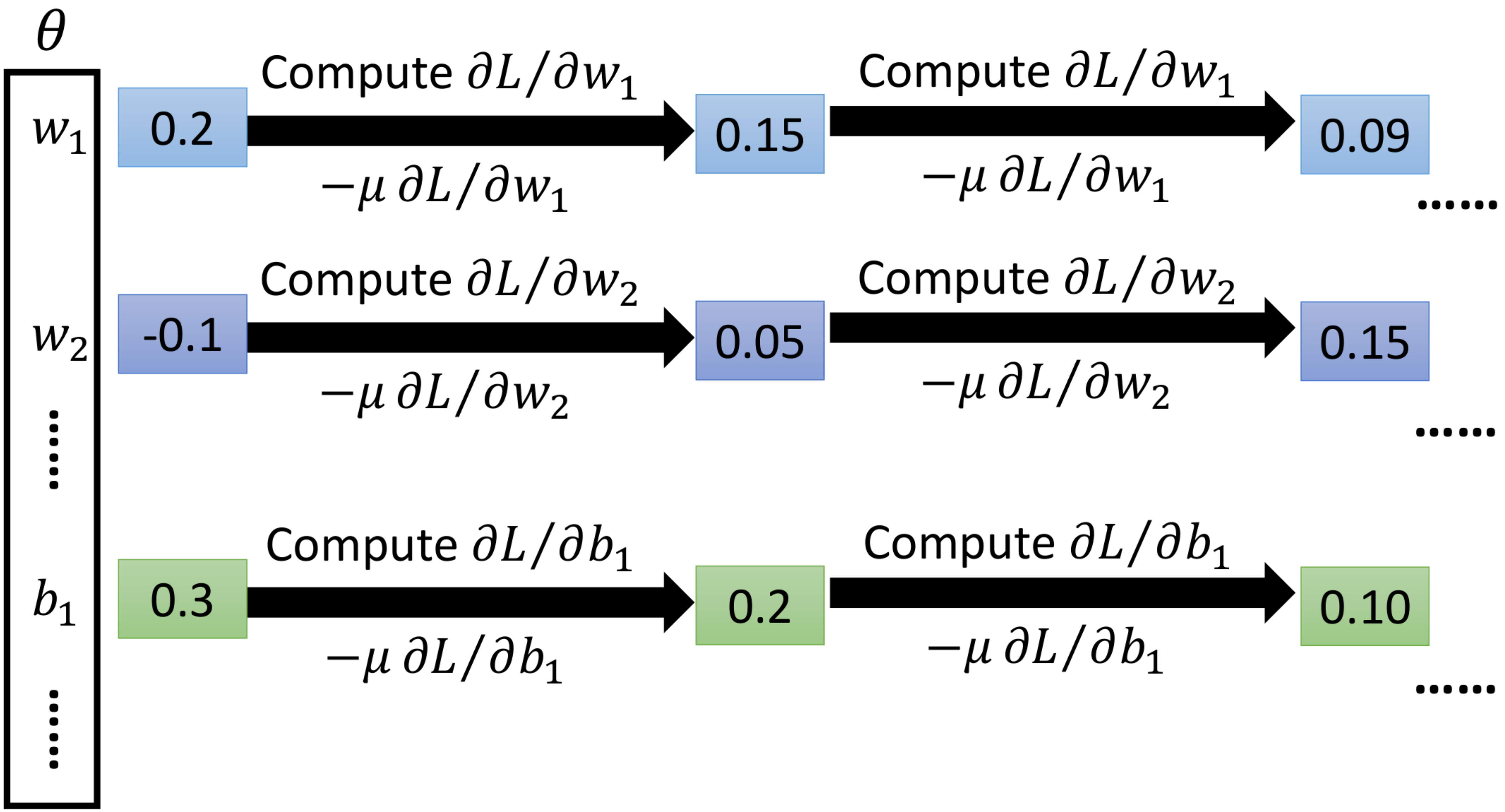


Gradient Descent



gradient $$\nabla L = \begin{bmatrix} \frac{\partial L}{\partial w_1} \\ \frac{\partial L}{\partial w_2} \\ \vdots \\ \frac{\partial L}{\partial b_1} \\ \vdots \end{bmatrix}$$

Gradient Descent



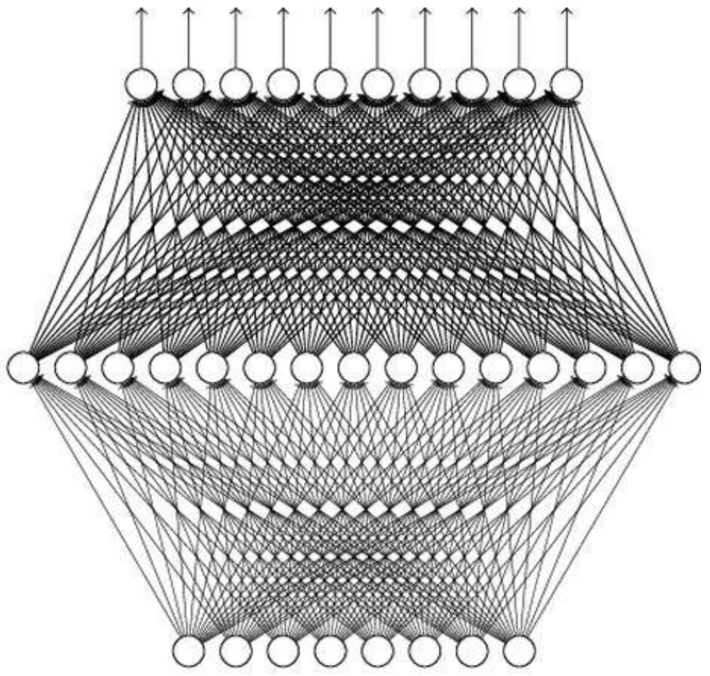
Universality Theorem

- Any continuous function f

$$f : \mathbb{R}^N \Rightarrow \mathbb{R}^M$$

- Can be realized by a network with one hidden layer

  ※ given enough hidden neurons



- Reference for the reason:

  ※ http://neuralnetworksanddeeplearning.com/chap4.html

11.4 Backpropagation

Gradient Descent

- Network parameters: $\theta = \{w_1, w_2, \cdots, b_1, b_2, \cdots\}$
- Starting Parameters: $\theta^0 \rightarrow \theta^1 \rightarrow \theta^2 \rightarrow \cdots$

$$\nabla L(\theta) = \begin{bmatrix} \partial L(\theta)/\partial w_1 \\ \partial L(\theta)/\partial w_2 \\ \vdots \\ \partial L(\theta)/\partial b_1 \\ \partial L(\theta)/\partial b_2 \\ \vdots \end{bmatrix}$$

$Compute\ \nabla L(\theta^0)$ $\qquad \theta^1 = \theta^0 - \eta \nabla L(\theta^0)$

$Compute\ \nabla L(\theta^1)$ $\qquad \theta^2 = \theta^1 - \eta \nabla L(\theta^1)$

Millions of parameters ......

To compute the gradients efficiently, we use:

backpropagation.

Chain Rule

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

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

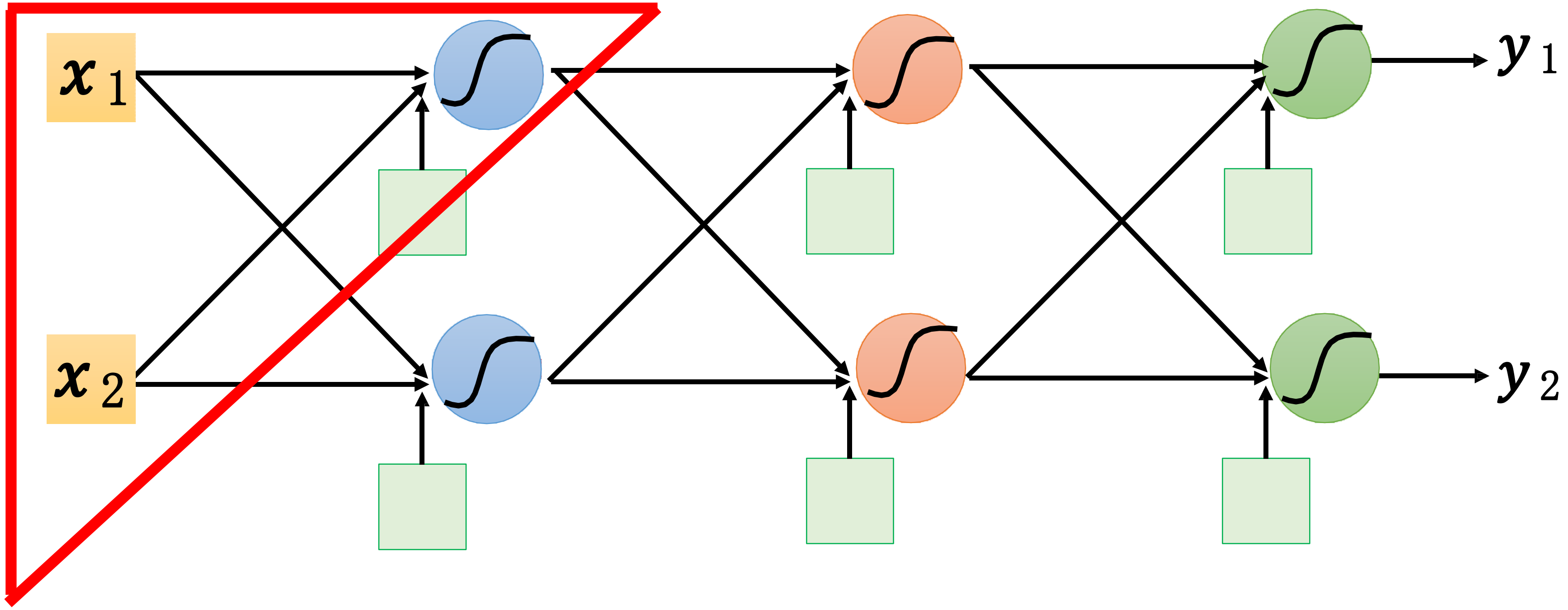
- Case 2: $x = g(s), \quad y = h(s), \quad z = k(x, y)$

$$\begin{tikzcd}
 & \Delta x \arrow[dr] & \\
\Delta s \arrow[ur] \arrow[dr] & & \Delta z \\
 & \Delta y \arrow[ur] &
\end{tikzcd}$$

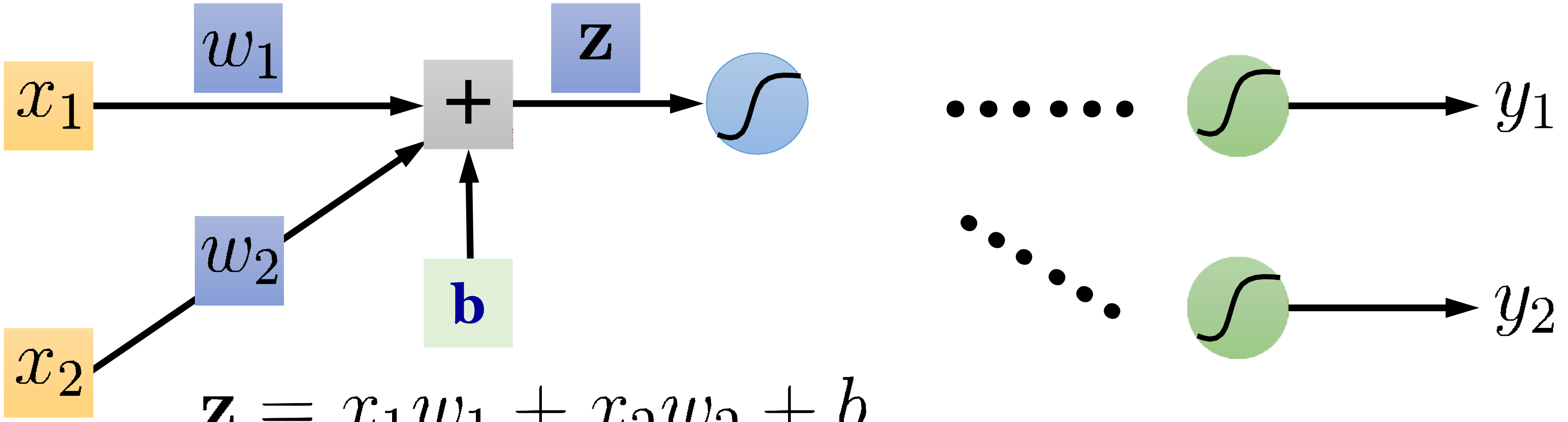
$$\frac{dz}{ds} = \frac{\partial z}{\partial x}\frac{\partial x}{\partial s} + \frac{\partial z}{\partial y}\frac{\partial y}{\partial s}$$

Backpropagation

$$L(\theta) = \sum_{n=1}^{N} C^n(\theta) \quad \Rightarrow \quad \frac{\partial L(\theta)}{\partial w} = \sum_{n=1}^{N} \frac{\partial C^n(\theta)}{\partial w}$$



Backpropagation



$$\mathbf{z} = x_1 w_1 + x_2 w_2 + b$$

$$\frac{\partial C}{\partial w} = ? \qquad \frac{\partial z}{\partial w}\frac{\partial C}{\partial z}$$

(Chain rule)

- Forward pass:

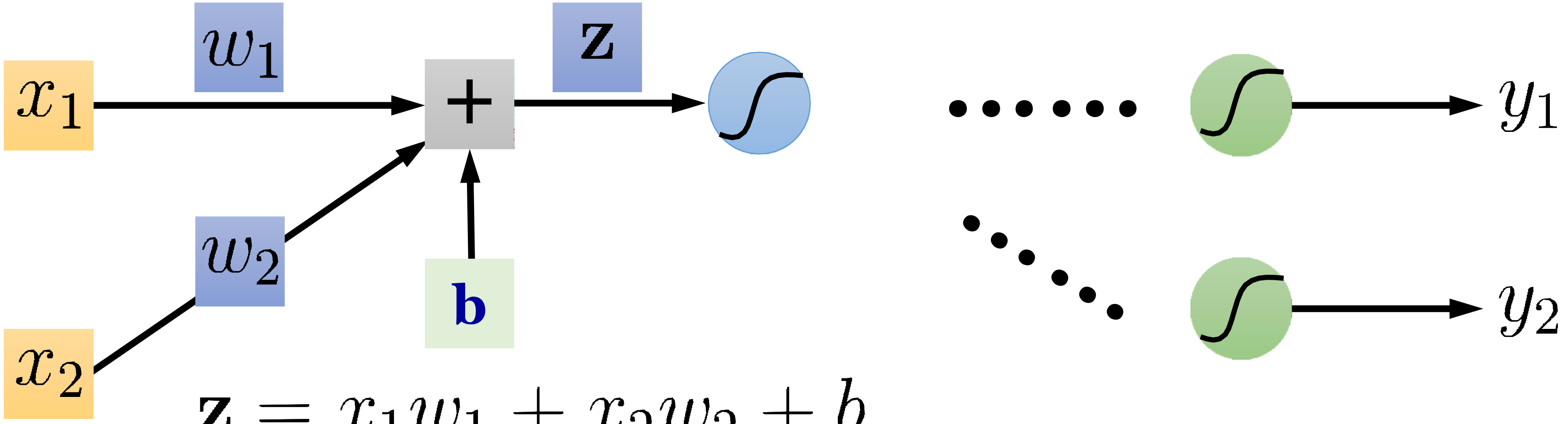
  ※ Compute $\partial z/\partial w$ for all parameters

- Backward pass:

  ※ Compute $\partial C/\partial z$ for all activation function inputs z

Backpropagation – Forward pass

- Compute $\partial z/\partial w$ for all parameters

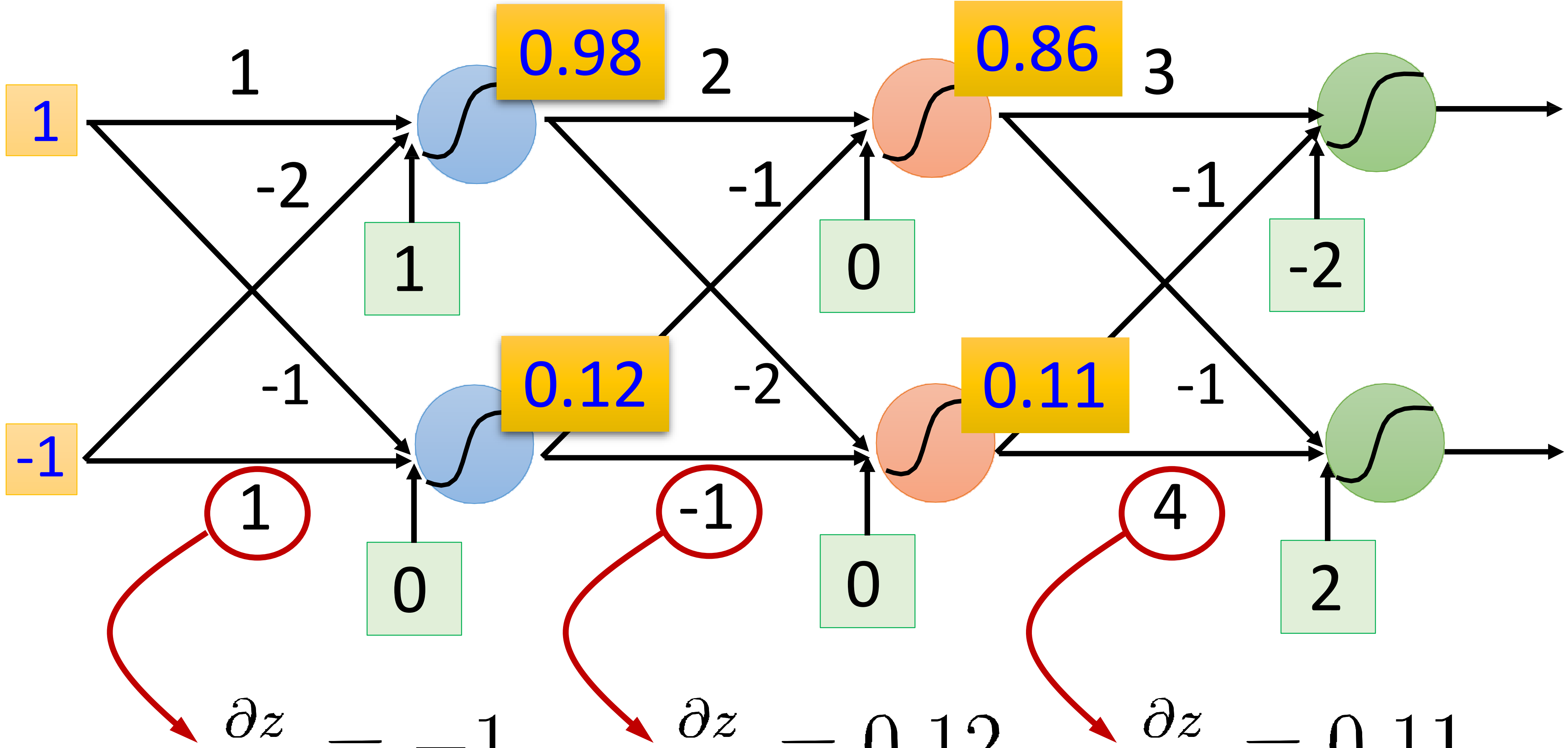


$$\mathbf{z} = x_1 w_1 + x_2 w_2 + b$$

$$\left.\begin{aligned} \frac{\partial z}{\partial w_1} &= ? \quad x_1 \\ \frac{\partial z}{\partial w_2} &= ? \quad x_2 \end{aligned}\right\} \text{The value of the input connected by the weight}$$

Backpropagation – Forward pass

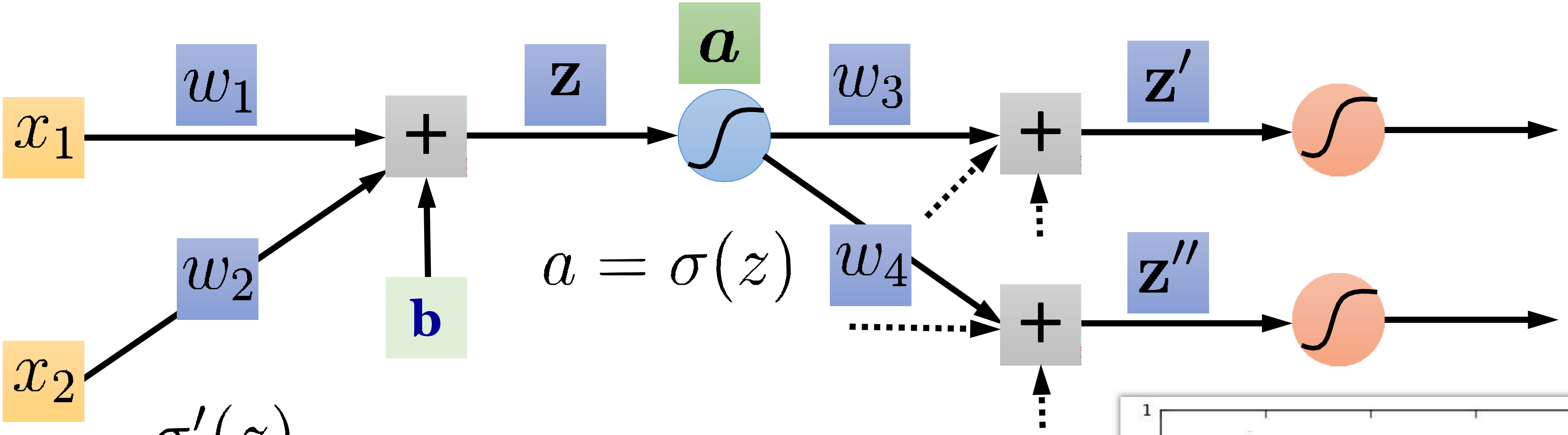
- Compute $\partial z / \partial w$ for all parameters



$$\frac{\partial z}{\partial w} = -1 \qquad \frac{\partial z}{\partial w} = 0.12 \qquad \frac{\partial z}{\partial w} = 0.11$$

Backpropagation – Backward pass

- Compute $\partial C / \partial z$ for all activation function inputs z

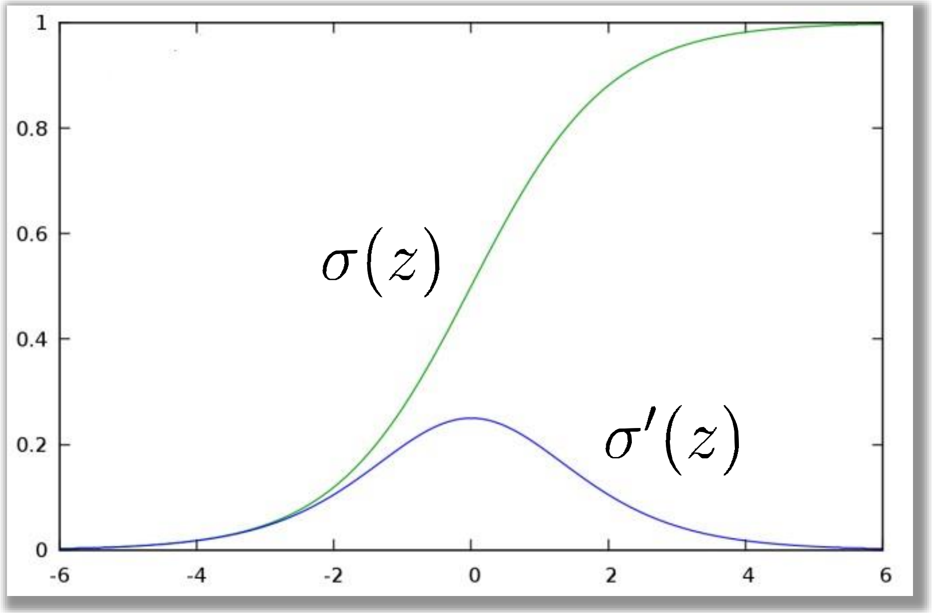


$\sigma'(z)$

$$\frac{\partial C}{\partial z} = \frac{\partial a}{\partial z} \frac{\partial C}{\partial a}$$

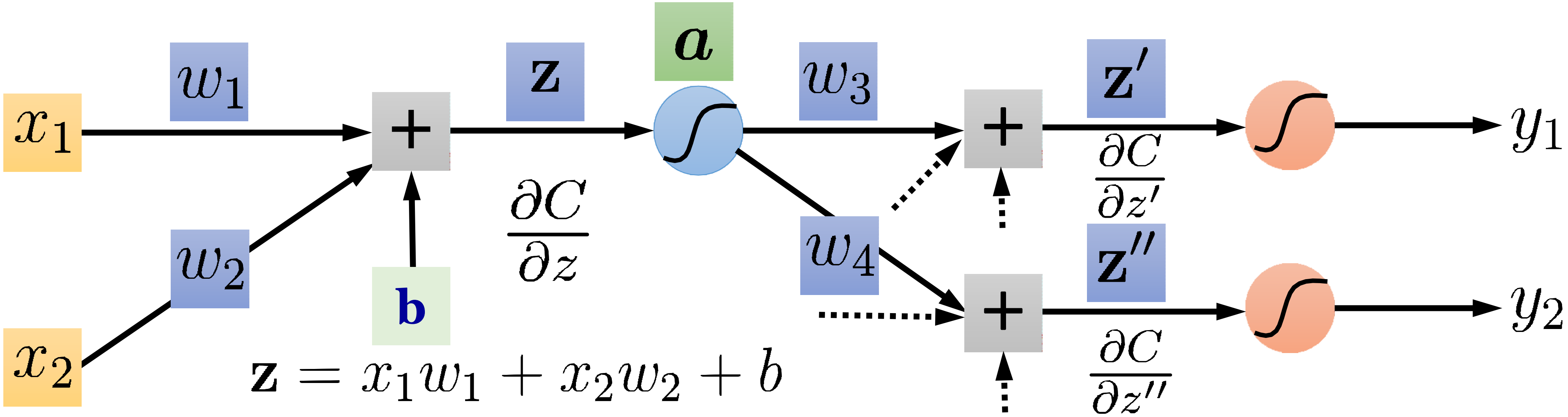
Assumed it's known

Chain rule: $\frac{\partial C}{\partial a} = \underbrace{\frac{\partial z'}{\partial a}}_{w_3} \frac{\partial C}{\partial z'} + \underbrace{\frac{\partial z''}{\partial a}}_{w_4} \frac{\partial C}{\partial z''}$



Backpropagation – Backward pass

- Compute $\partial C/\partial z$ for all activation function inputs z

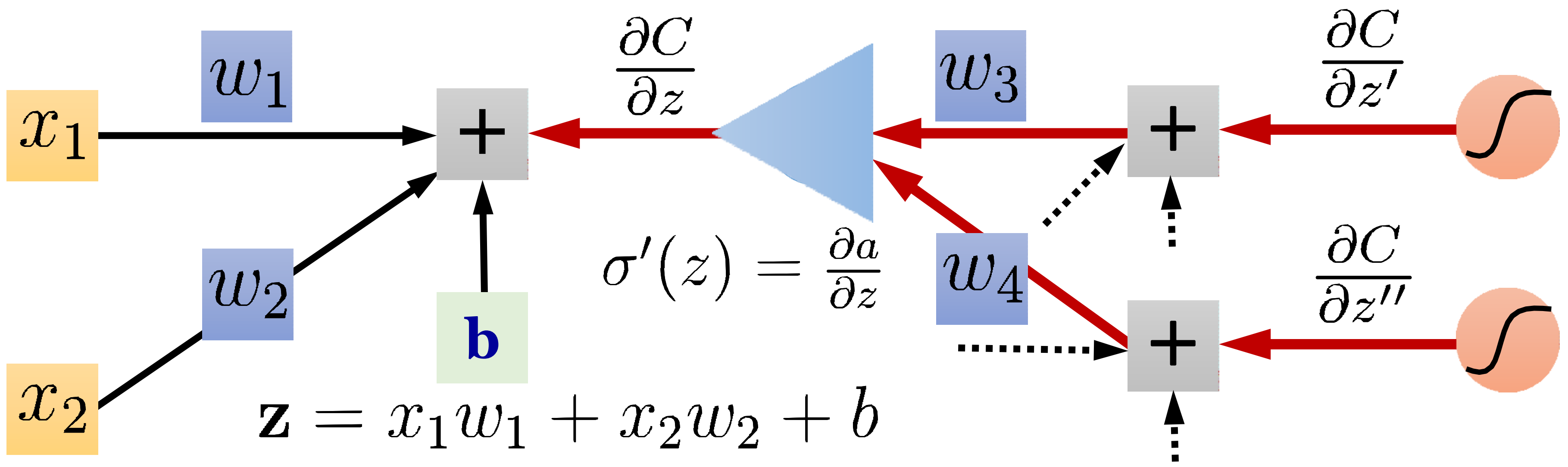


$$\frac{\partial C}{\partial z} = \sigma'(z)\left[w_3 \frac{\partial C}{\partial z'} + w_4 \frac{\partial C}{\partial z''}\right]$$

$$\sigma'(z) = \frac{\partial a}{\partial z}$$

Backpropagation – Backward pass

- Compute $\partial C/\partial z$ for all activation function inputs z

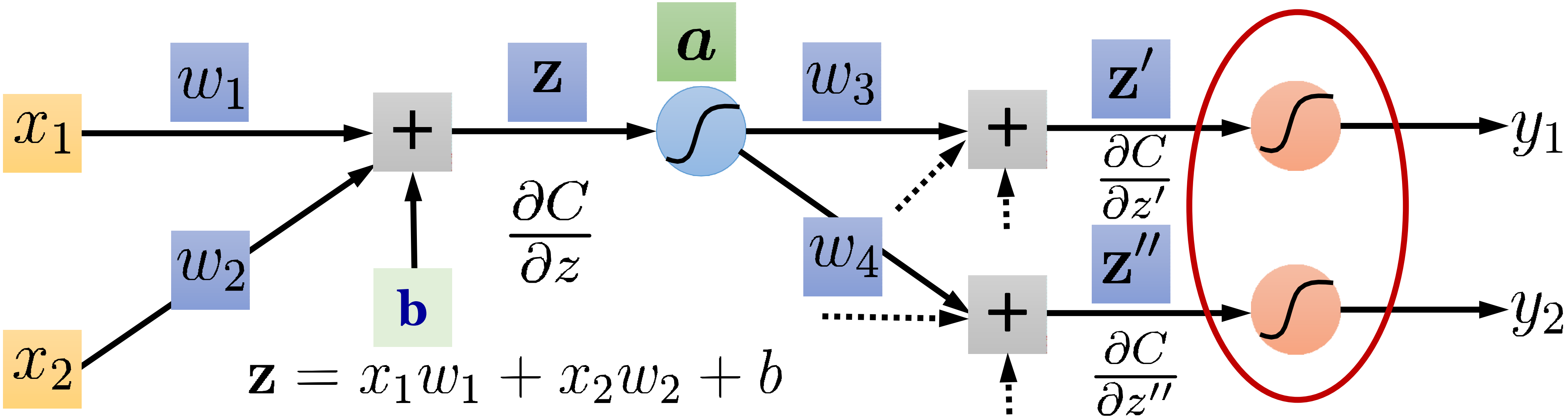


- $\sigma'(z)$ is a constant because z is already determined in the forward pass.

$$\frac{\partial C}{\partial z} = \sigma'(z) \left[w_3 \frac{\partial C}{\partial z'} + w_4 \frac{\partial C}{\partial z''} \right]$$

Backpropagation – Backward pass

- Compute $\partial C/\partial z$ for all activation function inputs z



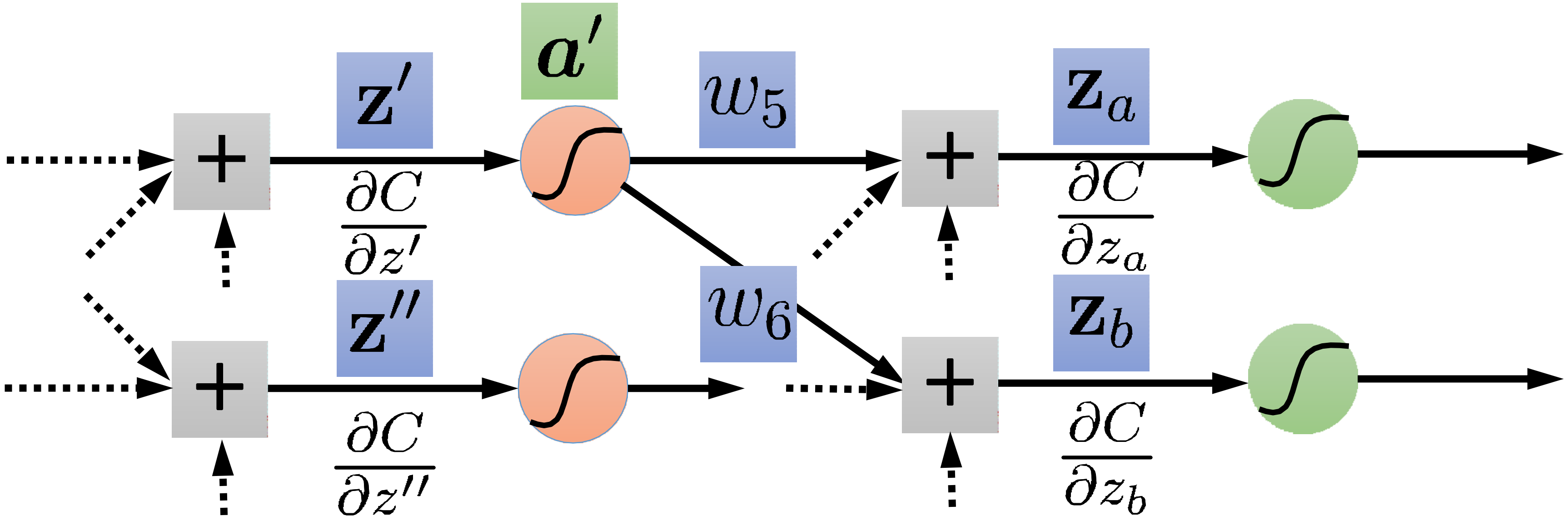
- Case 1. Output Layer $\frac{\partial C}{\partial z} = \sigma'(z)\left[w_3 \frac{\partial C}{\partial z'} + w_4 \frac{\partial C}{\partial z''}\right]$

$$\frac{\partial C}{\partial z'} = \frac{\partial y_1}{\partial z'} \frac{\partial C}{\partial y_1} \qquad \frac{\partial C}{\partial z''} = \frac{\partial y_2}{\partial z''} \frac{\partial C}{\partial y_2}$$

Done!

Backpropagation – Backward pass

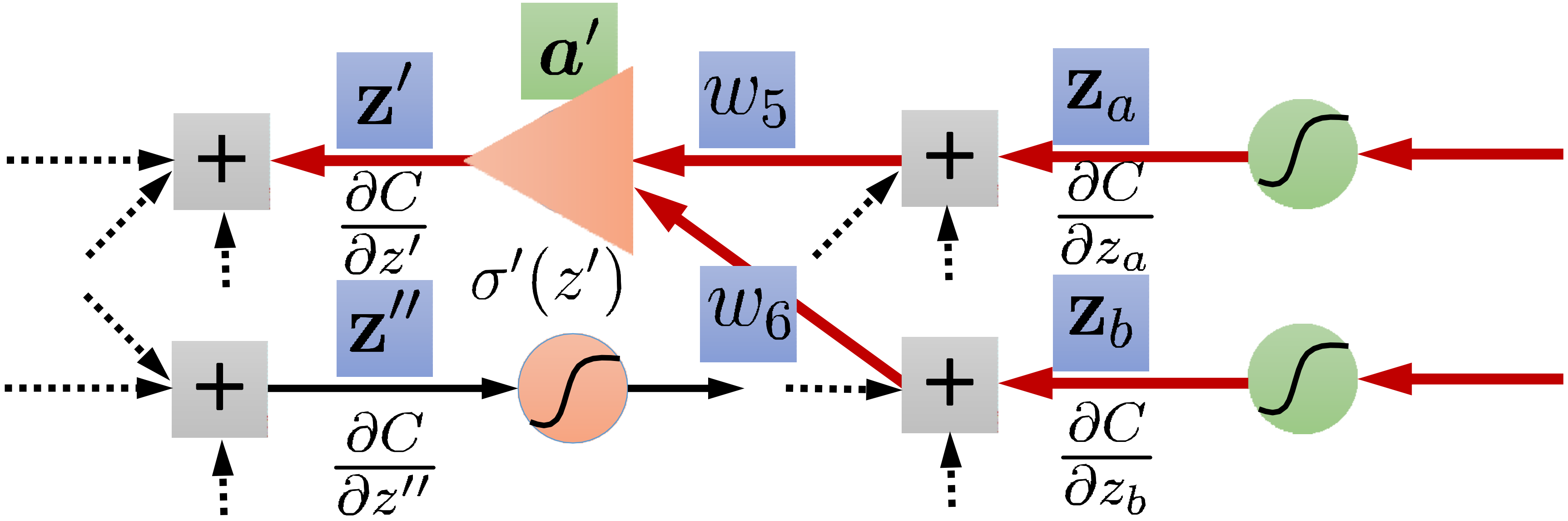
- Compute $\partial C/\partial z$ for all activation function inputs z
- Case 2. Not Output Layer



$$\frac{\partial C}{\partial z} = \sigma'(z)\left[w_3 \frac{\partial C}{\partial z'} + w_4 \frac{\partial C}{\partial z''}\right]$$

Backpropagation – Backward pass

- Compute $\partial C/\partial z$ for all activation function inputs z
- Case 2. Not Output Layer

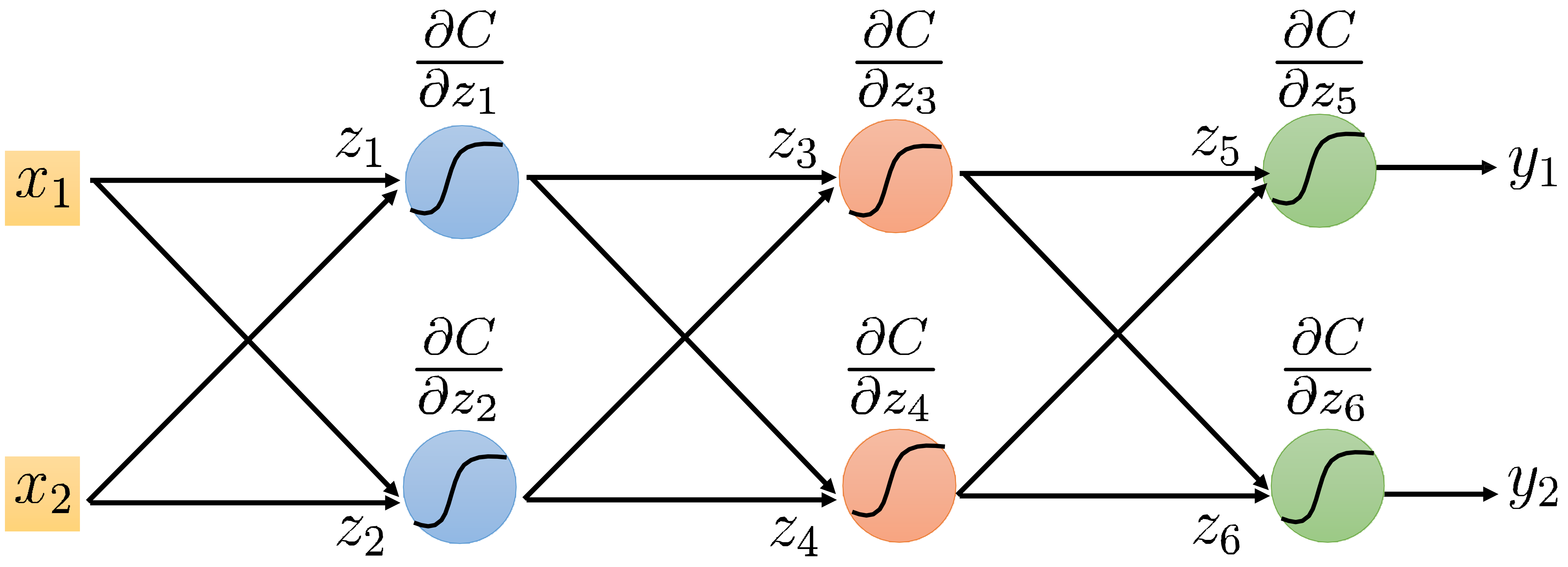


- Compute $\partial C/\partial z$ recursively

※ Until we reach the output layer ……

Backpropagation – Backward pass

- Compute $\partial C/\partial z$ for all activation function inputs z
- Compute $\partial C/\partial z$ from the output layer

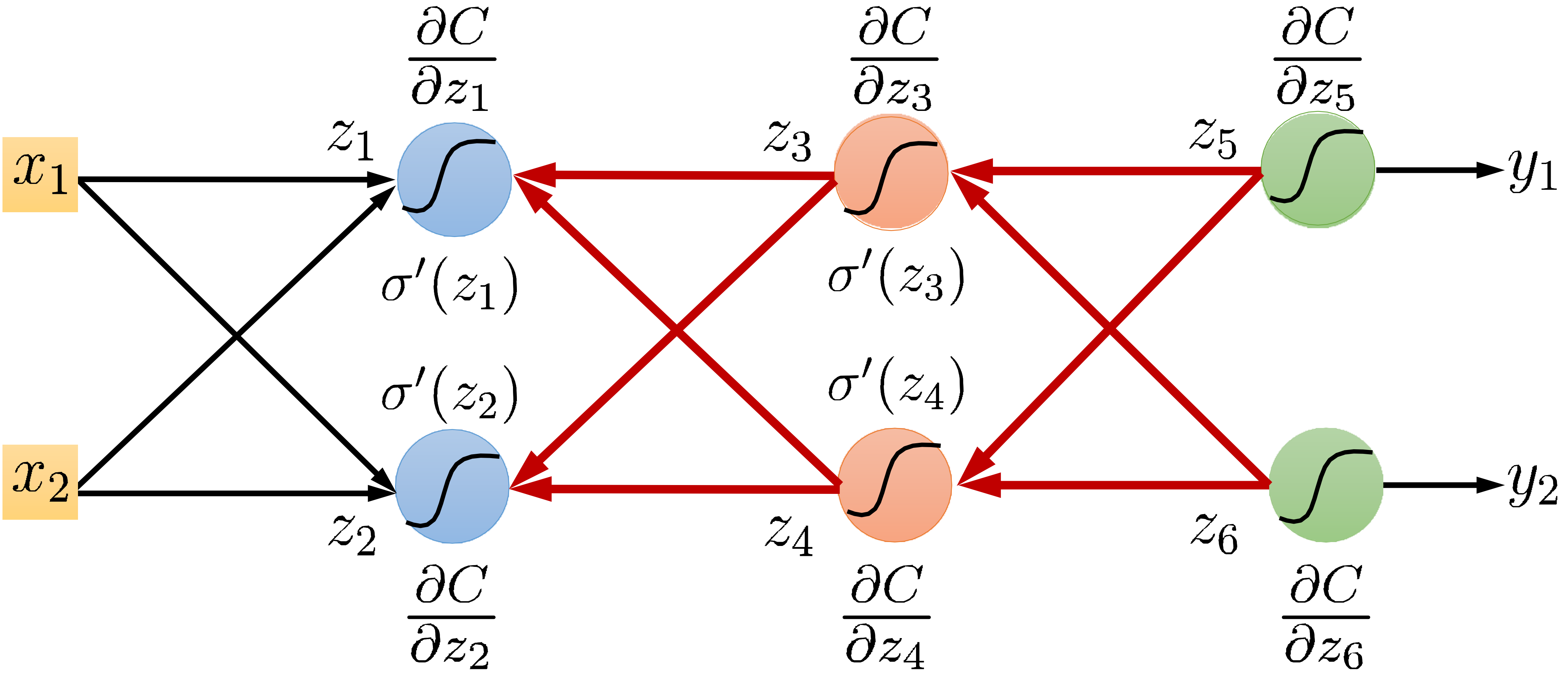


$$\frac{\partial C}{\partial z} = \sigma'(z)\left[w_3\frac{\partial C}{\partial z'} + w_4\frac{\partial C}{\partial z''}\right]$$

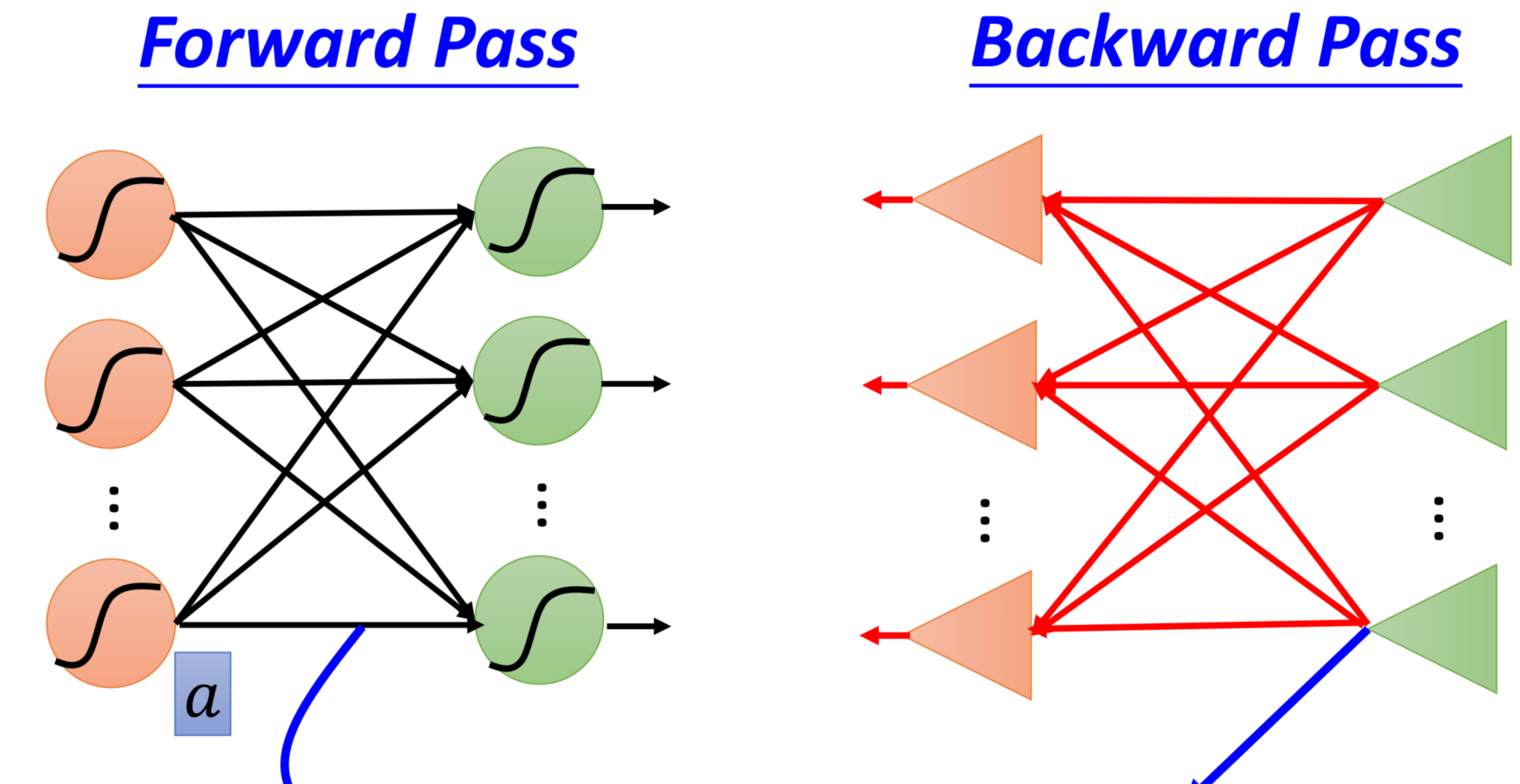
Backpropagation – Backward pass

- Compute $\partial C/\partial z$ for all activation function inputs z
- Compute $\partial C/\partial z$ from the output layer

$$\frac{\partial C}{\partial z} = \sigma'(z)\left[w_3\frac{\partial C}{\partial z'} + w_4\frac{\partial C}{\partial z''}\right]$$



Backpropagation – Summary



$$\frac{\partial z}{\partial w} = a \quad \times \quad \frac{\partial C}{\partial z} \quad = \frac{\partial C}{\partial w}$$

for all w

References

- 《 Neural Networks and Deep Learning 》

  ※ written by Michael Nielsen

  ※ http://neuralnetworksanddeeplearning.com/

- 《 Deep Learning 》

  ※ written by Yoshua Bengio, Ian J. Goodfellow and Aaron Courville

  ※ http://www.deeplearningbook.org

- 《 Machine learning 》

  ※ https://speech.ee.ntu.edu.tw/~hylee/ml/2022-spring.php

Next chapter: Reinforcement Learning