Machine Learning
机器学习

Lecture5：Artificial Neural Network
人工神经网络

李洁
nijanice@163.com

线性回归
Linear Regression

- Given the training dataset of (data,label) pairs,

$$D = \{(x^{(i)}, y^{(i)})\}_{i=1.2,\ldots,N}$$

let the machine learn a function from data to label

$$y^{(i)} \approx \boxed{f_\theta(x^{(i)})}$$

$$\boxed{f_\theta(x) = \theta_1\, x_1 + \theta_2\, x_2 + \cdots + \theta_n\, x_n + \theta_0}$$

- Function set $\{f_\theta(x^{(i)})\}$ is called hypothesis space
- Learning is referred to as updating the parameter θ to make the prediction closed to the corresponding label

逻辑斯蒂回归
Logistic Regression

- Given the training dataset of (data,label) pairs,

$$D = \{(x^{(i)}, y^{(i)})\}_{i=1.2,\ldots,N}$$

let the machine learn a function from data to label

$$y^{(i)} \approx \boxed{f_\theta(x^{(i)})} \Rightarrow \boxed{f_\theta(x) = \frac{1}{1 + e^{-\theta^T x}}}$$

- Function set $\{f_\theta(x^{(i)})\}$ is called hypothesis space
- Learning is referred to as updating the parameter θ to make the prediction closed to the corresponding label

决策树学习
Decision Tree Learning

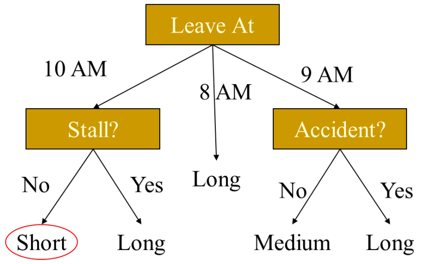
- Given the training dataset of (data,label) pairs,

$$D = \{(x^{(i)}, y^{(i)})\}_{i=1.2,\ldots,N}$$

let the machine learn a function from data to label

$$y^{(i)} \approx \boxed{f_\theta(x^{(i)})}$$

a decision tree

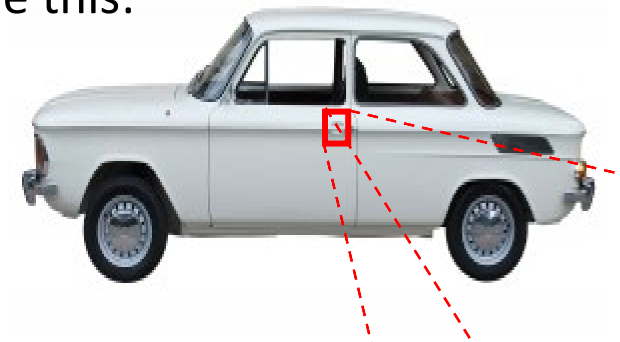


- Function set $\{f_\theta(x^{(i)})\}$ is called hypothesi
- Learning is referred to as updating the pa
prediction closed to the corresponding label

例子

Computer Vision: Car Detection

You see this:

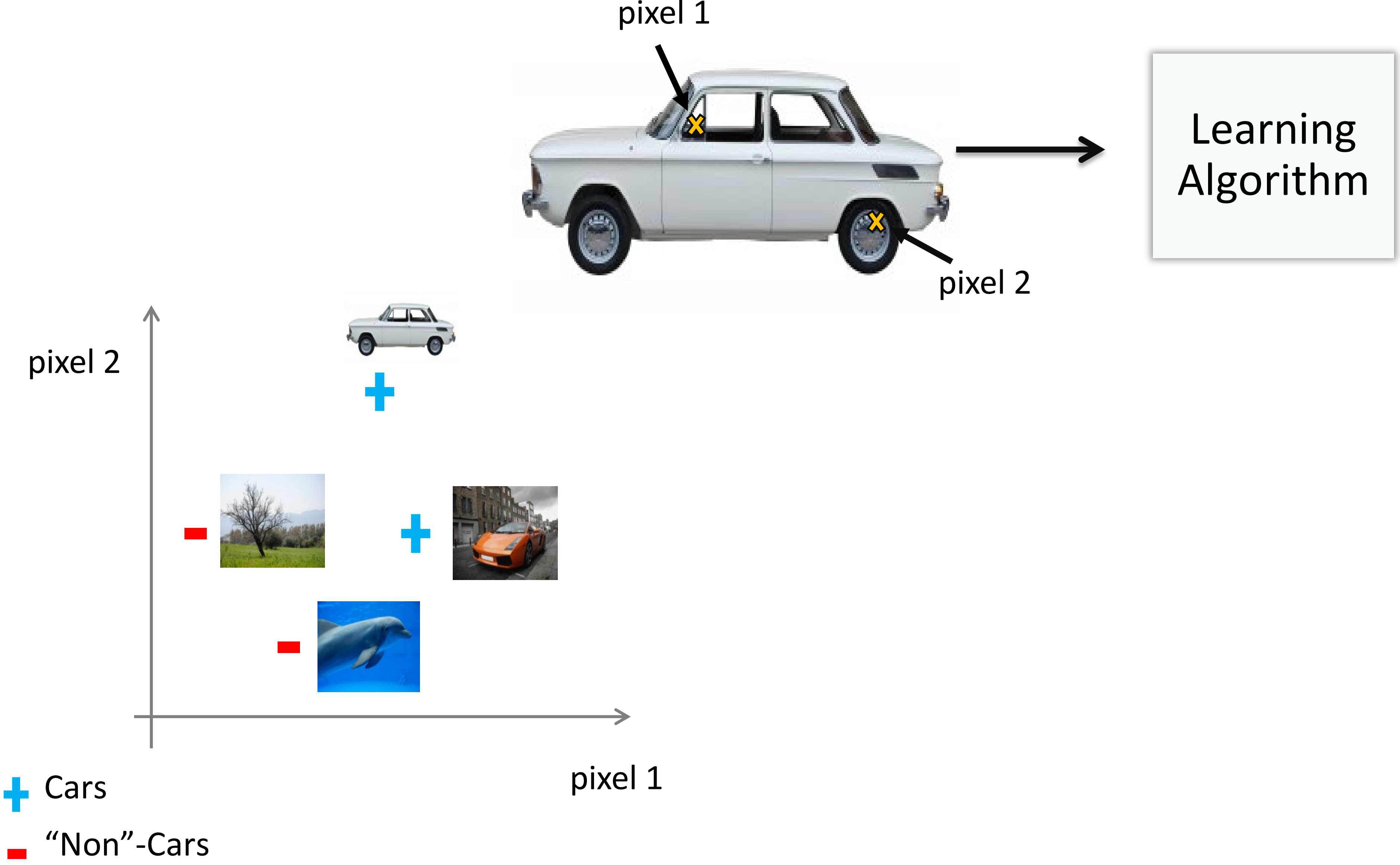


What is this?

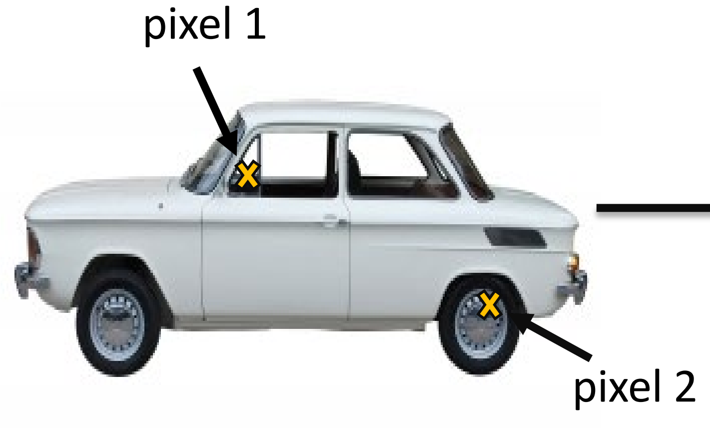
But the camera sees this:

194	210	201	212	199	213	215	195	178	158	182	209
180	189	190	221	209	205	191	167	147	115	129	163
114	126	140	188	176	165	152	140	170	106	78	88
87	103	115	154	143	142	149	153	173	101	57	57
102	112	106	131	122	138	152	147	128	84	58	66
94	95	79	104	105	124	129	113	107	87	69	67
68	71	69	98	89	92	98	95	89	88	76	67
41	56	68	99	63	45	60	82	58	76	75	65
20	43	69	75	56	41	51	73	55	70	63	44
50	50	57	69	75	75	73	74	53	68	59	37
72	59	53	66	84	92	84	74	57	72	63	42
67	61	58	65	75	78	76	73	59	75	69	50

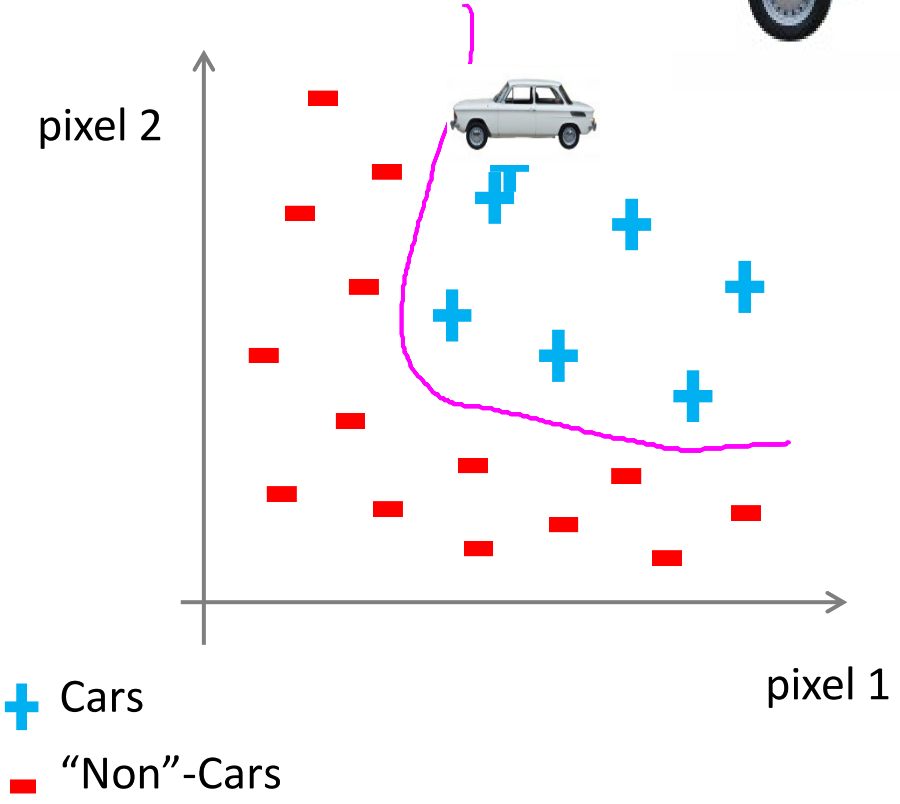
例子
Computer Vision: Car Detection



例子
Computer Vision: Car Detection



Learning Algorithm



50 x 50 pixel images→ 2500 pixels
(7500 if RGB)

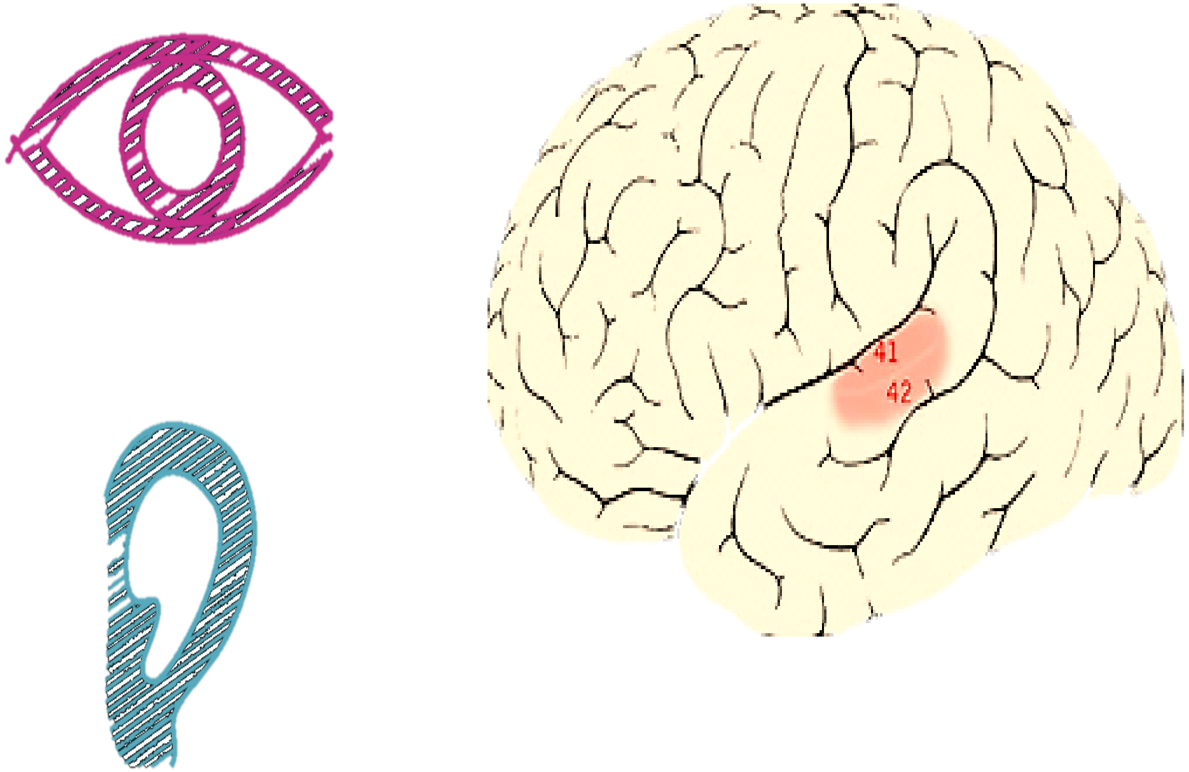
$n = 2500$

$$x = \begin{bmatrix} \text{pixel 1 intensity} \\ \text{pixel 2 intensity} \\ \vdots \\ \text{pixel 2500 intensity} \end{bmatrix}$$

Quadratic features ($x_i \times x_j$): ≈3 million features

大脑的学习算法
Brain Learning Algorithm

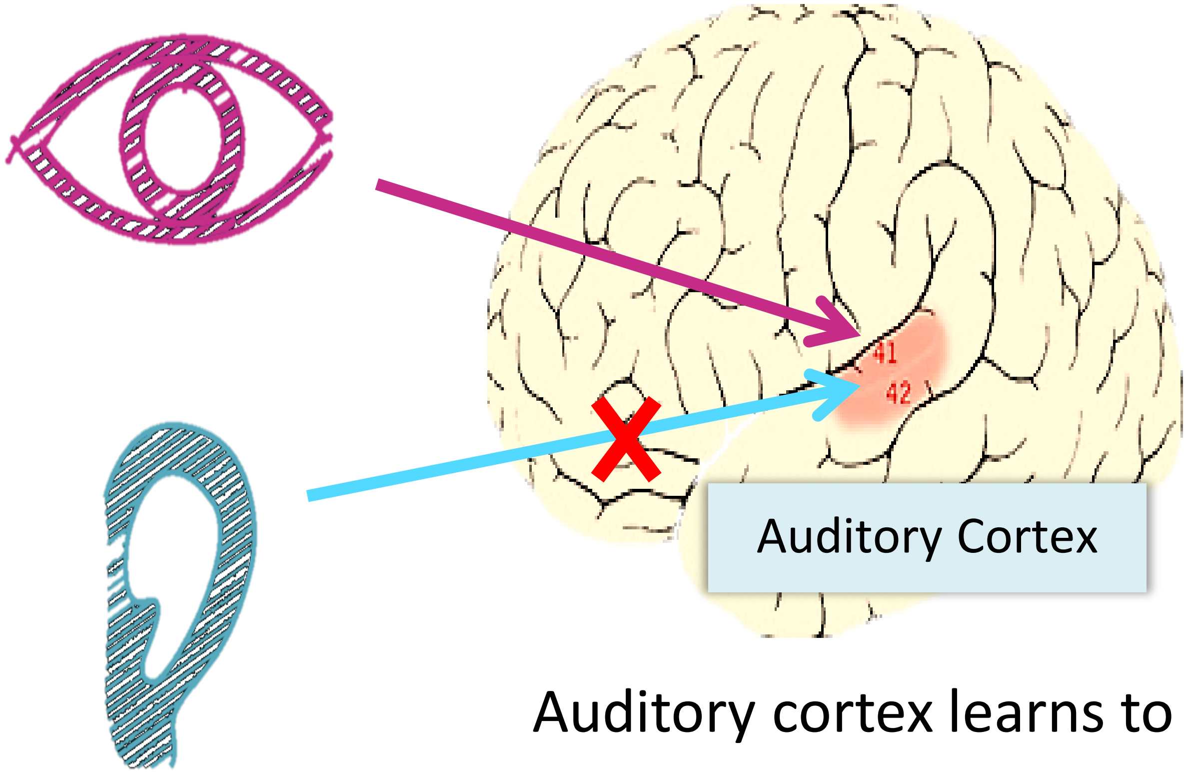
The "one learning algorithm" hypothesis



[Roe et al., 1992]

大脑的学习算法
Brain Learning Algorithm

The “one learning algorithm” hypothesis

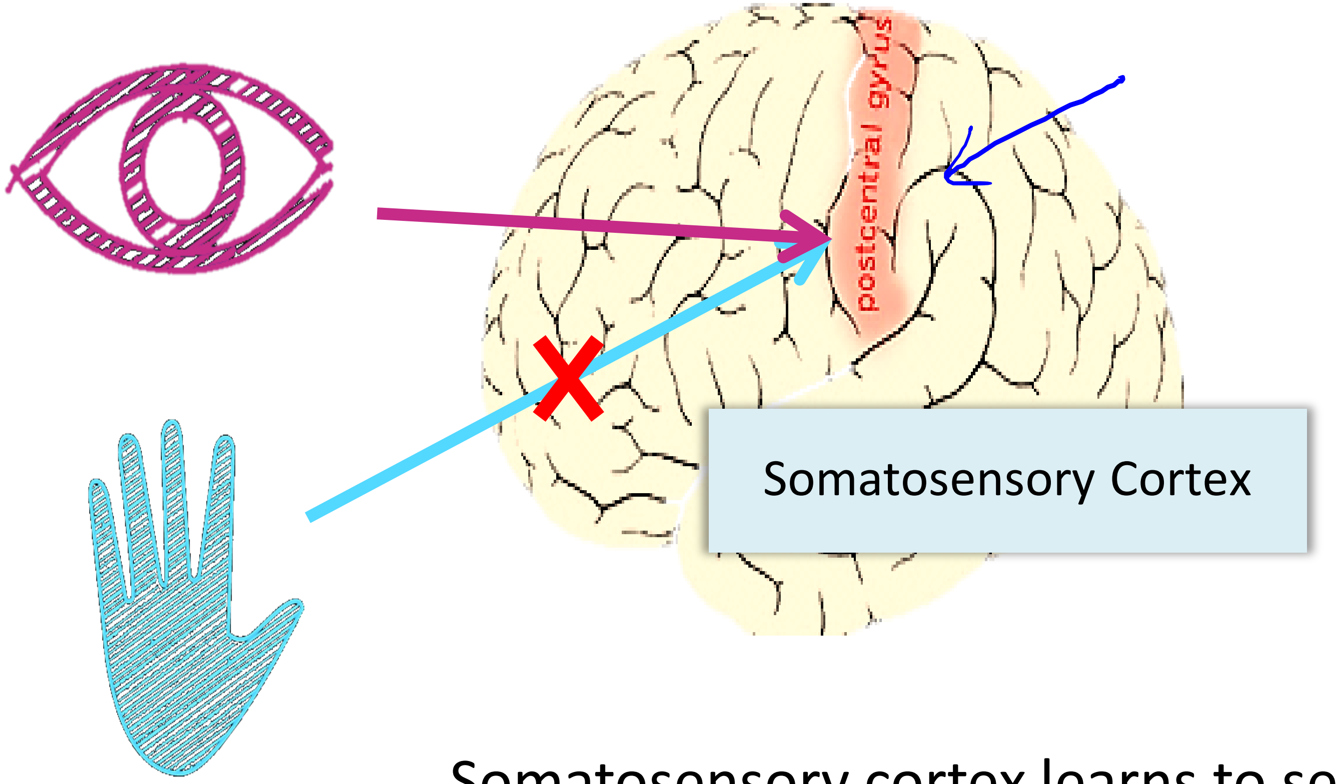


Auditory cortex learns to see

[Roe et al., 1992]

大脑的学习算法
Brain Learning Algorithm

The “one learning algorithm” hypothesis



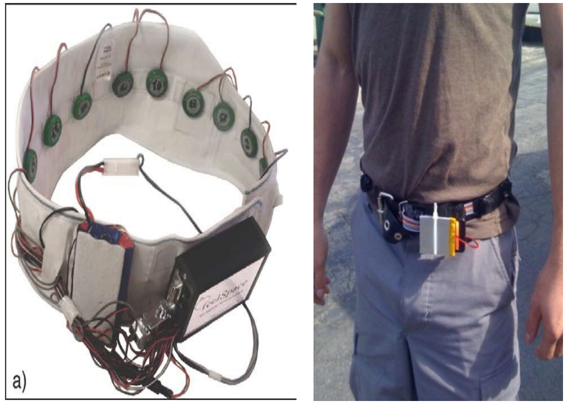
Somatosensory cortex learns to see

[Metin & Frost, 1989]

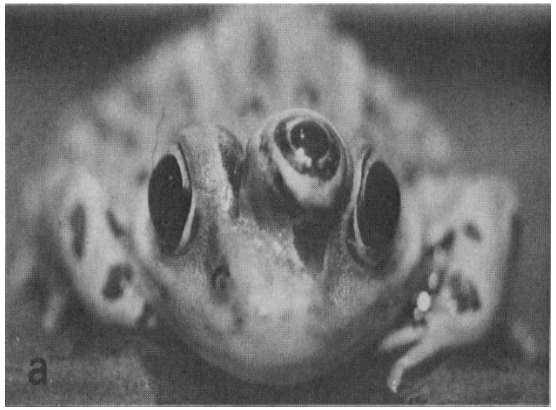
大脑的学习算法
Brain Learning Algorithm

Seeing with your tongue

Human echolocation (sonar)



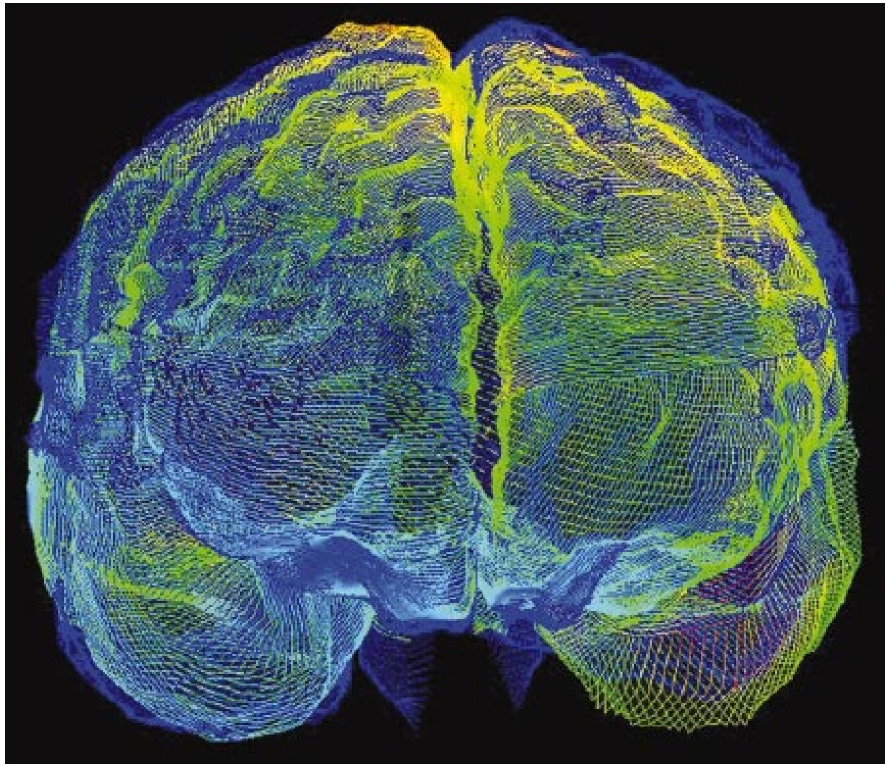
Haptic belt: Direction sense



Implanting a 3rd eye

[BrainPort; Welsh & Blasch, 1997; Nagel et al., 2005; Constantine-Paton & Law, 2009]

大脑中的神经元
Neurons in the Brain



- The brain is composed of a mass of interconnected neurons (about 10^{11} neurons with an average of 10^4 connections each).
  - each neuron is connected to many other neurons

人工神经网络
Artificial Neural Networks

- Neural Networks can be :

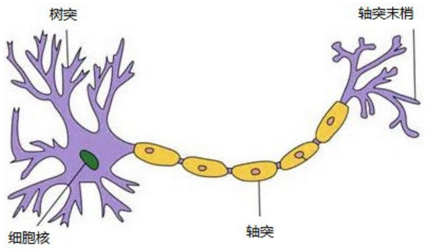
    - **Biological** models

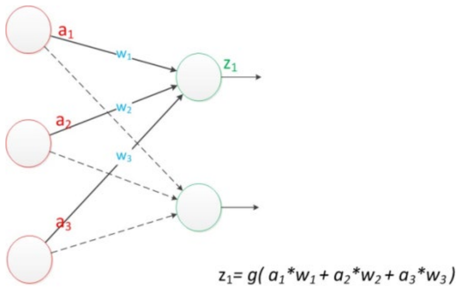
    - **Artificial** models

- Desire to produce artificial systems capable of sophisticated computations similar to the human brain.

神经网络的历史
Brief History of Neural Networks

- The First wave
  - 1943 McCulloch and Pitts proposed the **McCulloch-Pitts neuron model**.
  - 1958 Rosenblatt introduced the simple single layer networks now called **Perceptrons**.
  - 1969 Minsky and Papert's book Perceptrons demonstrated the limitation of single layer perceptrons, and almost the whole field went into hibernation.

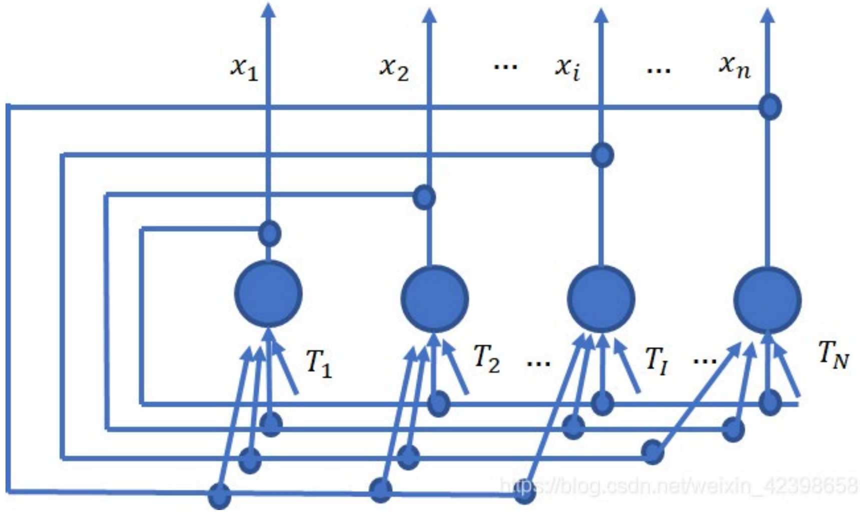




Rosenblat and perceptron

神经网络的历史
Brief History of Neural Networks

- The Second wave
  - 1982 Energy Based Model（EBM), the hopfield network
  - 1984 the traveling salesman problem (abbr. TSP), one of the NP-complete combinatorial optimization problems, can be solved by Hopfield neural network



J. J. Hopfield, "Neural networks and physical systems with emergent collective computational abilities", Proceedings of the National Academy of Sciences of the USA, vol. 79 no. 8 pp. 2554–2558, April 1982.

神经网络的历史
Brief History of Neural Networks

- The Second wave
  - 1986 The **Back-Propagation learning algorithm** for Multi-Layer Perceptrons was rediscovered and the whole field took off again.


David Rumelhart


Geoffery Hinton

Learning representations by back-propagating errors

David E. Rumelhart*, Geoffrey E. Hinton† & Ronald J. Williams*

* Institute for Cognitive Science, C-015, University of California, San Diego, La Jolla, California 92093, USA
† Department of Computer Science, Carnegie-Mellon University, Pittsburgh, Philadelphia 15213, USA

We describe a new learning procedure, back-propagation, for networks of neurone-like units. The procedure repeatedly adjusts the weights of the connections in the network so as to minimize a measure of the difference between the actual output vector of the net and the desired output vector. As a result of the weight adjustments, internal 'hidden' units which are not part of the input or output come to represent important features of the task domain, and the regularities in the task are captured by the interactions of these units. The ability to create useful new features distinguishes back-propagation from earlier, simpler methods such as the perceptron-convergence procedure[1].

There have been many attempts to design self-organizing neural networks. The aim is to find a powerful synaptic modification rule that will allow an arbitrarily connected neural network to develop an internal structure that is appropriate for a particular task domain. The task is specified by giving the desired state vector of the output units for each state vector of

more difficult when we introduce hidden units whose actual or desired states are not specified by the task. (In perceptrons, there are 'feature analysers' between the input and output that are not true hidden units because their input connections are fixed by hand, so their states are completely determined by the input vector: they do not learn representations.) The learning procedure must decide under what circumstances the hidden units should be active in order to help achieve the desired input-output behaviour. This amounts to deciding what these units should represent. We demonstrate that a general purpose and relatively simple procedure is powerful enough to construct appropriate internal representations.

The simplest form of the learning procedure is for layered networks which have a layer of input units at the bottom; any number of intermediate layers; and a layer of output units at the top. Connections within a layer or from higher to lower layers are forbidden, but connections can skip intermediate layers. An input vector is presented to the network by setting the states of the input units. Then the states of the units in each layer are determined by applying equations (1) and (2) to the connections coming from lower layers. All units within a layer have their states set in parallel, but different layers have their states set sequentially, starting at the bottom and working upwards until the states of the output units are determined.

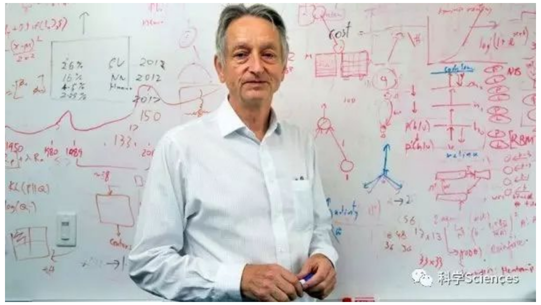
The total input, x_j, to unit j is a linear function of the outputs, y_i, of the units that are connected to j and of the weights, w_{ji}, on these connections

$$x_j = \sum_i y_i w_{ji} \qquad (1)$$

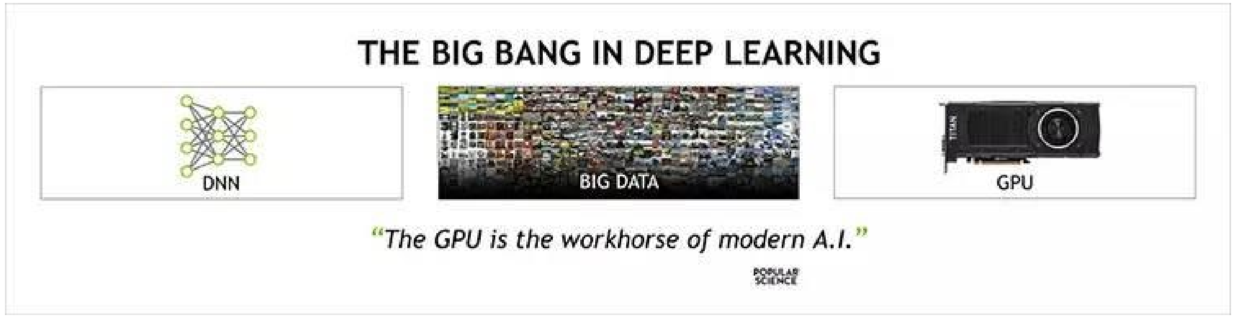
Units can be given biases by introducing an extra input to each

神经网络的历史
Brief History of Neural Networks

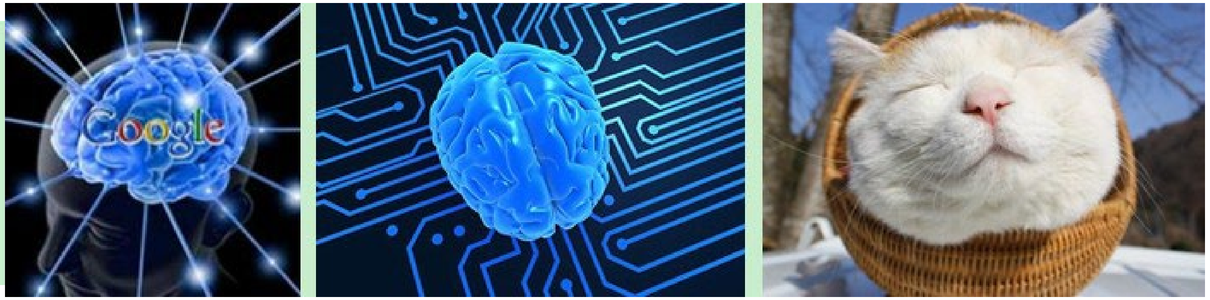
- The Third wave
  - 2006 **Deep (neural networks) Learning** gains popularity and
  - 2012 made significant break-through in many applications.
    - AlexNet—ImageNet champion model



Geoffery Hinton



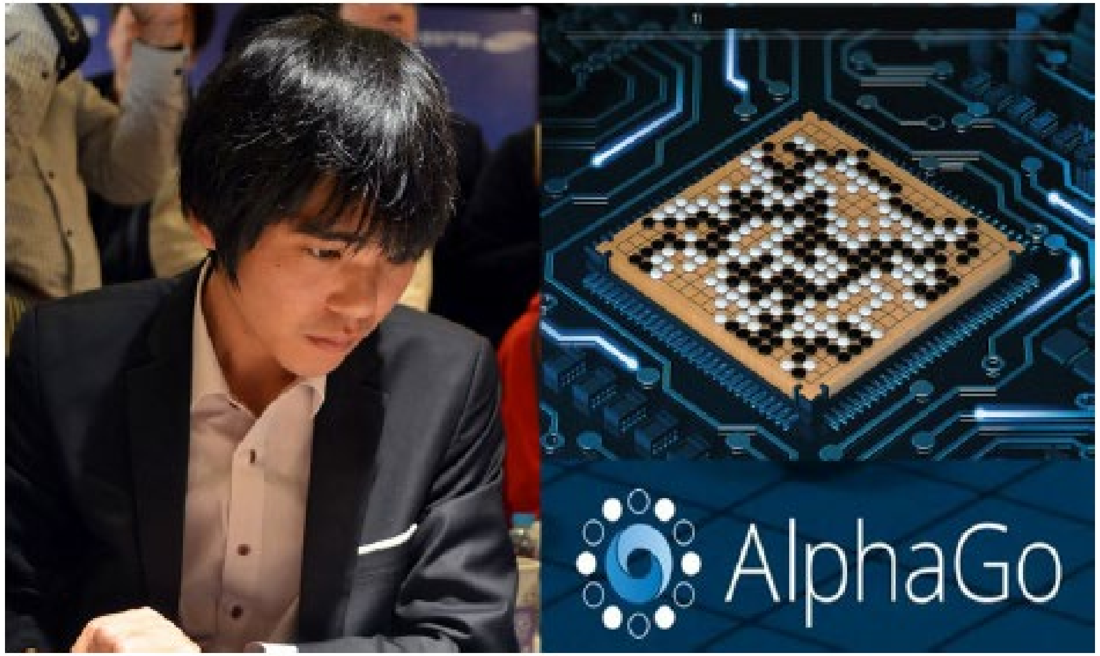
Microsoft DNN research



Google Brain

神经网络的历史
Brief History of Neural Networks

- AlphaGo wins Lee Sedol (4-1)



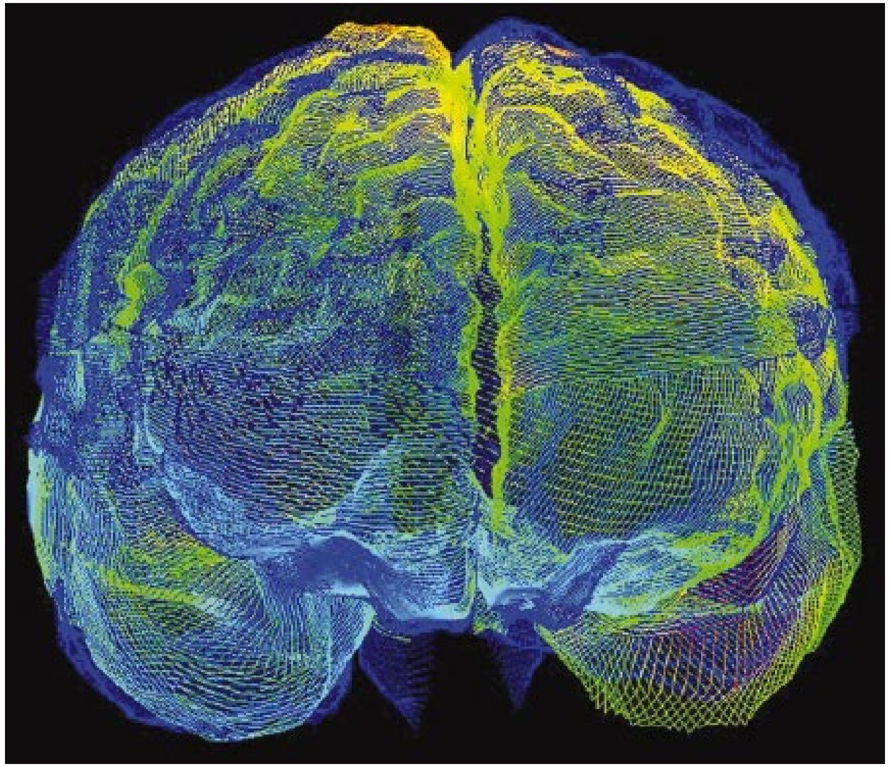
https://deepmind.com/research/alphago/

Breaking News of AI in 2016

Rank	Name	↑↓	Flag	Elo
1	Ke Jie	↑		3628
2	AlphaGo			3598
3	Park Junghwan	↑		3585
4	Tuo Jiaxi	↑		3535
5	Mi Yuting	↑		3534
6	Iyama Yuta	↑		3525
7	Shi Yue	↑		3522
8	Lee Sedol	↑		3521
9	Zhou Ruiyang	↑		3517
10	Shin Jinseo	↑		3503
11	Chen Yaoye	↑		3495
12	Lian Xiao	↑		3493
13	Tan Xiao	↑		3489
14	Kim Jiseok	↑		3489
15	Choi Cheolhan	↑		3482
16	Park Yeonghun	↑		3482
17	Gu Zihao	↑		3468
18	Fan Yunruo	↑		3468
19	Huang Yunsong	↑		3467
20	Li Qincheng	↑		3465
21	Tang Weixing	↑		3461
22	Lee Donghoon	↑		3460
23	Lee Yeongkyu	↑		3459
24	Fan Tingyu	↑		3459
25	Tong Mengcheng	↑		3447
26	Kang Dongyun	↑		3442
27	Wang Xi	↑		3439
28	Weon Seongjin	↑		3439
29	Yang Dingxin	↑		3439
30	Gu Li	↑		3436

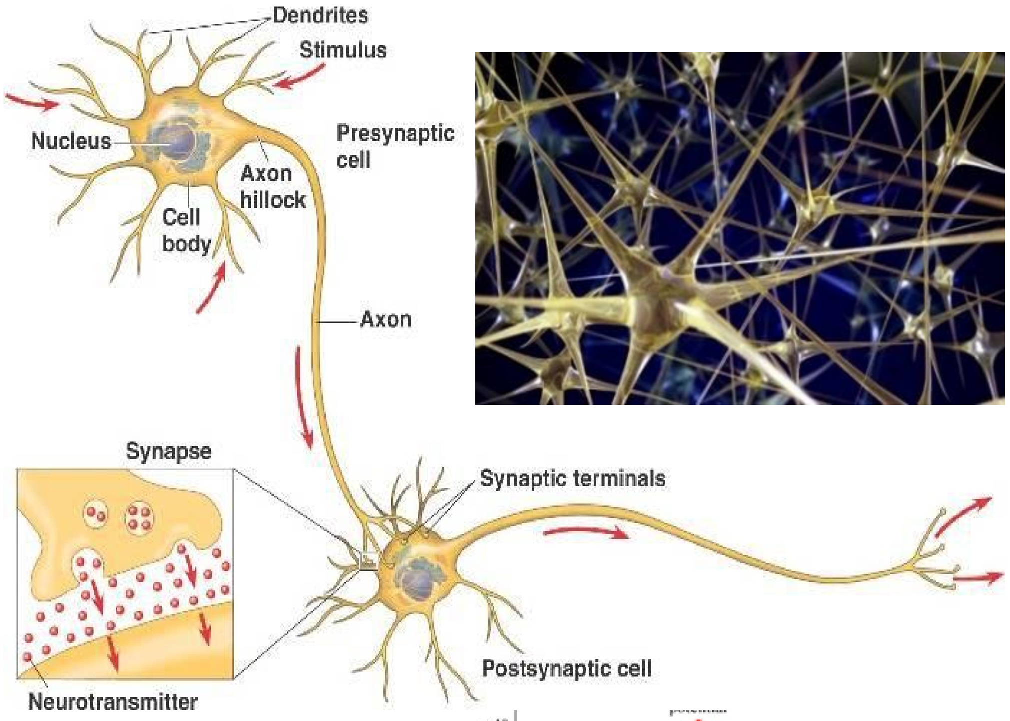
https://www.goratings.org/

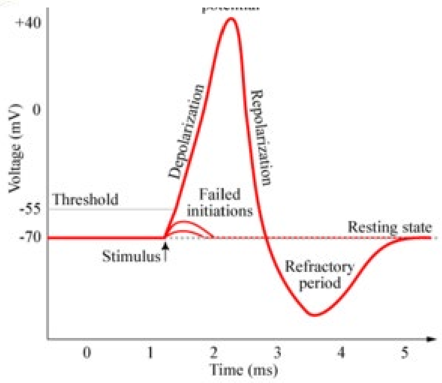
大脑中的神经元
Neurons in the Brain



- The brain is composed of a mass of interconnected neurons (about 10^{11} neurons with an average of 10^4 connections each).
  - each neuron is connected to many other neurons

大脑中的神经元
Neurons in the Brain

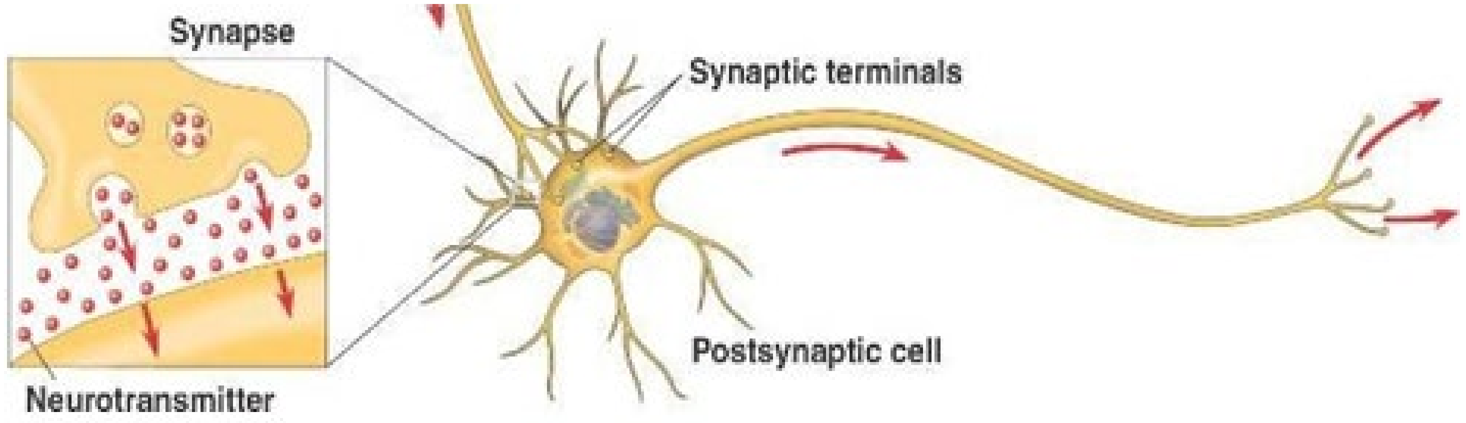


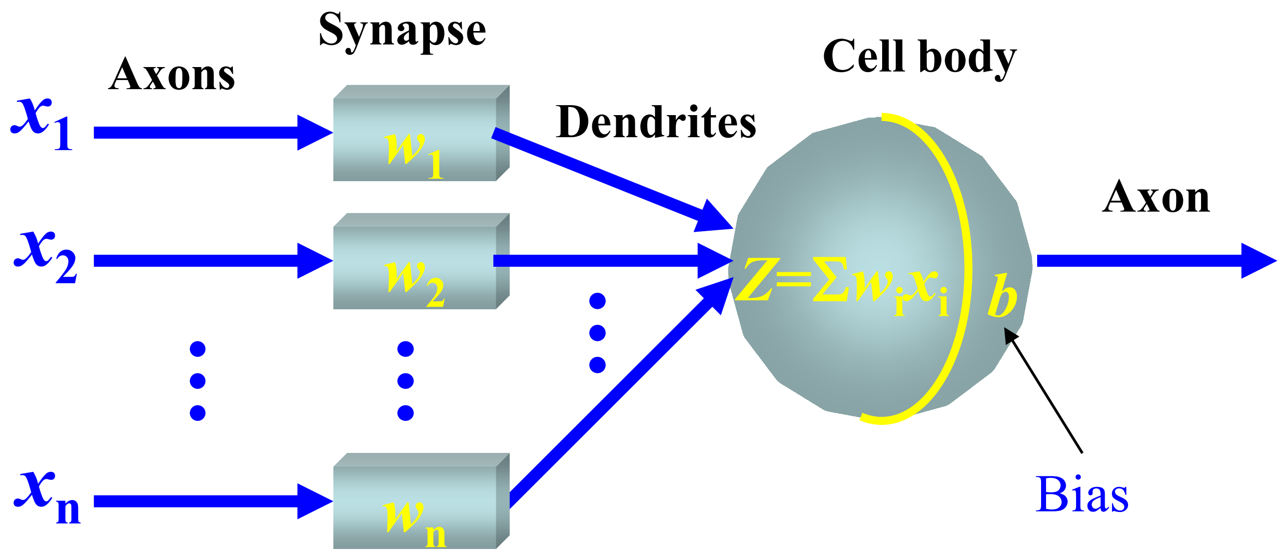


- A neuron receives input from other neurons from its synapses
- Inputs are approximately summed
- When the input exceeds a threshold the neuron sends an electrical spike that travels from the body, down the axon, to the next neuron(s)

[Credit: US National Institutes of Health, National Institute on Aging]

人工神经元
Artificial Neuron Diagram

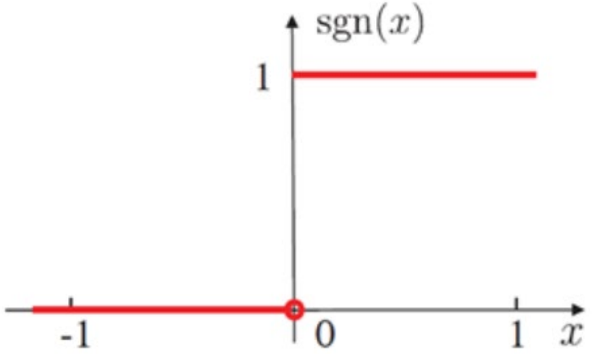




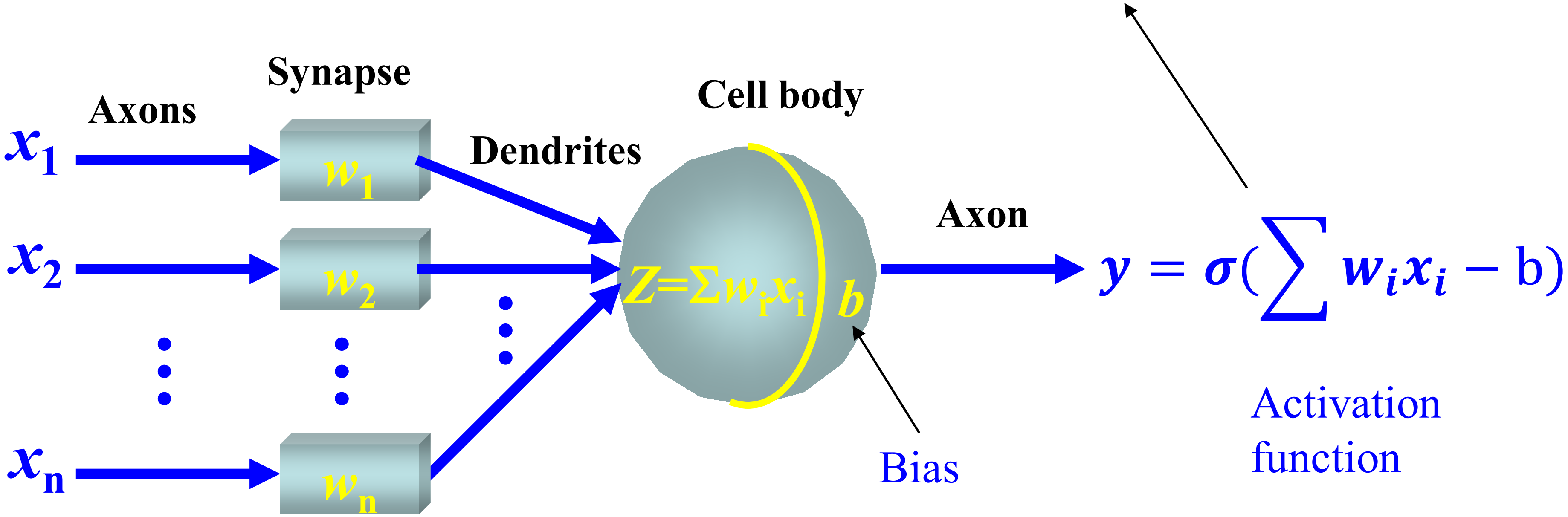
$$y = \sigma(\sum w_i x_i - \mathrm{b})$$

Activation function

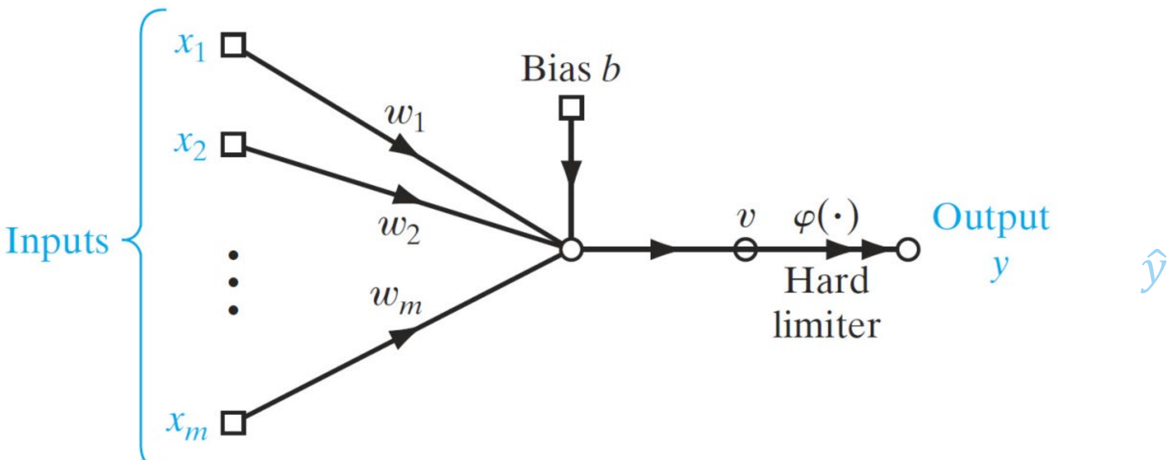
人工神经元
Artificial Neuron



$$sgn(z) = \begin{cases} 1 & \text{if } z \geq 0 \\ 0 & \text{if } z < 0 \end{cases}$$



感知器及其训练法则
Perceptron & Training Rule



Rosenblatt's single layer perceptron [1958]

Training

$$w_i = w_i + \boldsymbol{\eta}(y - \hat{y})\, x_i$$

$$b = b + \boldsymbol{\eta}(y - \hat{y})$$

where η is the "learning rate"

Prediction

$$y = \sigma(\sum w_i x_i - b)$$

Activation function

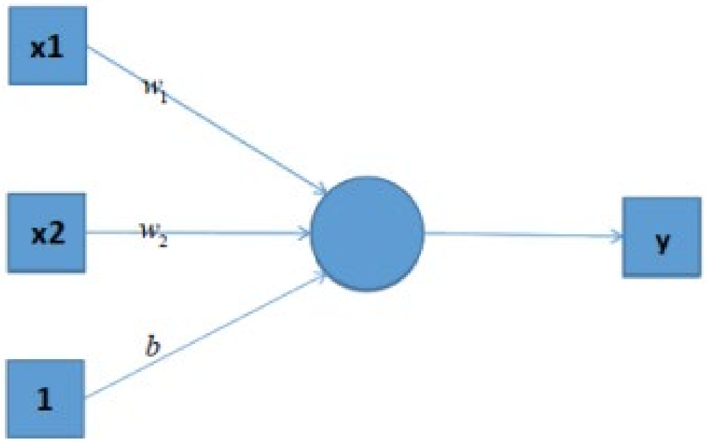
$$\sigma(z) = \begin{cases} 1 & if\ z \geq 0 \\ 0 & otherwise \end{cases}$$

- Equivalent to rules:
  - If output is correct do nothing.
  - If output is high, lower weights on active inputs
  - If output is low, increase weights on active inputs

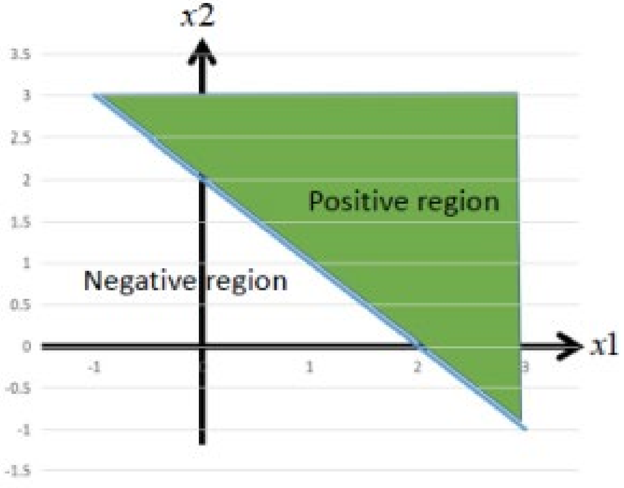
感知器及其训练法则
Perceptron & Training Rule

- Since perceptron uses linear threshold function, it is searching for a linear separator that discriminates the classes.

Perceptron



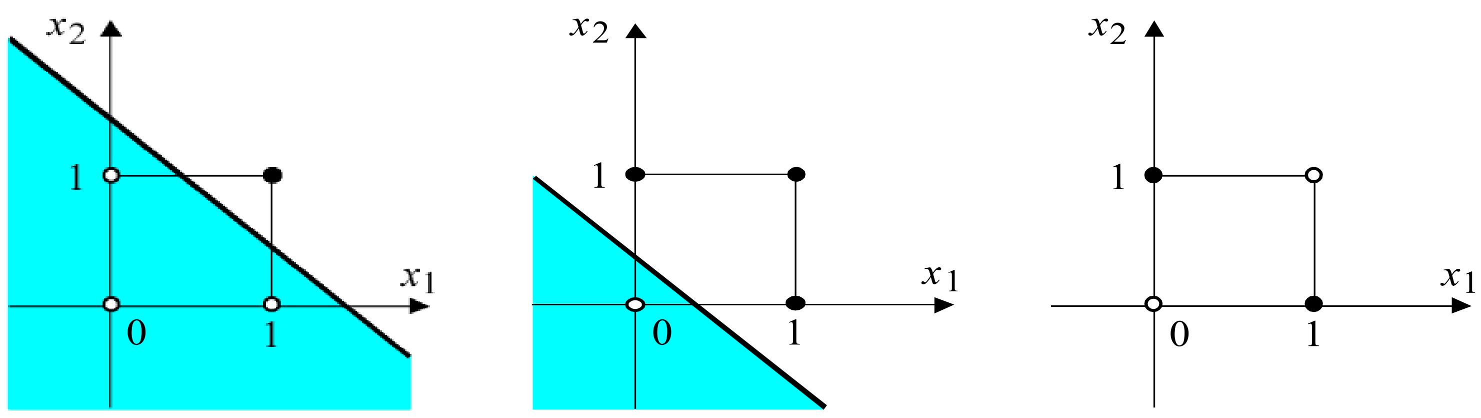
$$y = w_1 x_1 + w_2 x_2 + b$$



$$w_1 = 1, w_2 = 1, b = -2$$

感知器及其局限
Perceptron Limitation

- Can prove it will converge
  - If training data is linearly separable
  - And η sufficiently small
- However, a single layer perceptron can only learn linearly separable concepts.

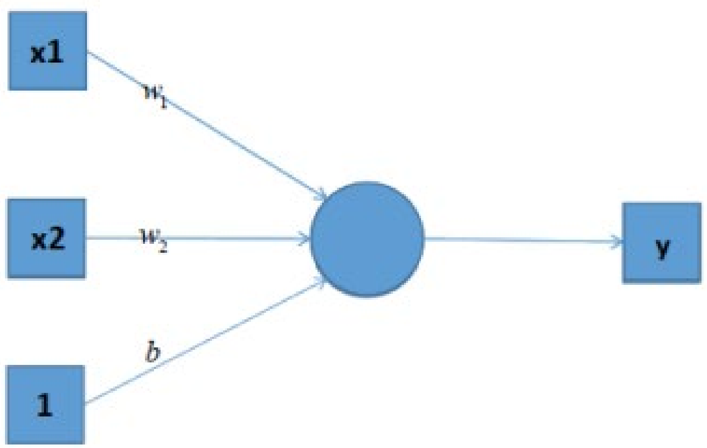


(a) *AND* ($x_1 \cap x_2$) (b) *OR* ($x_1 \cup x_2$) (c) *Exclusive-OR* ($x_1 \oplus x_2$)

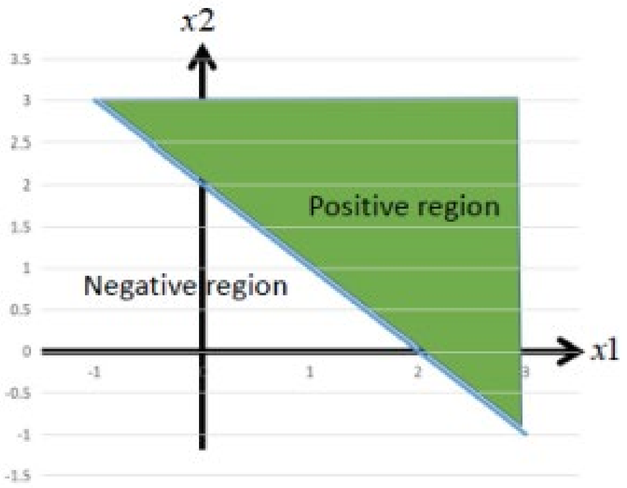
感知器及其训练法则
Perceptron & Training Rule

- Since perceptron uses linear threshold function, it is searching for a linear separator that discriminates the classes.

Perceptron



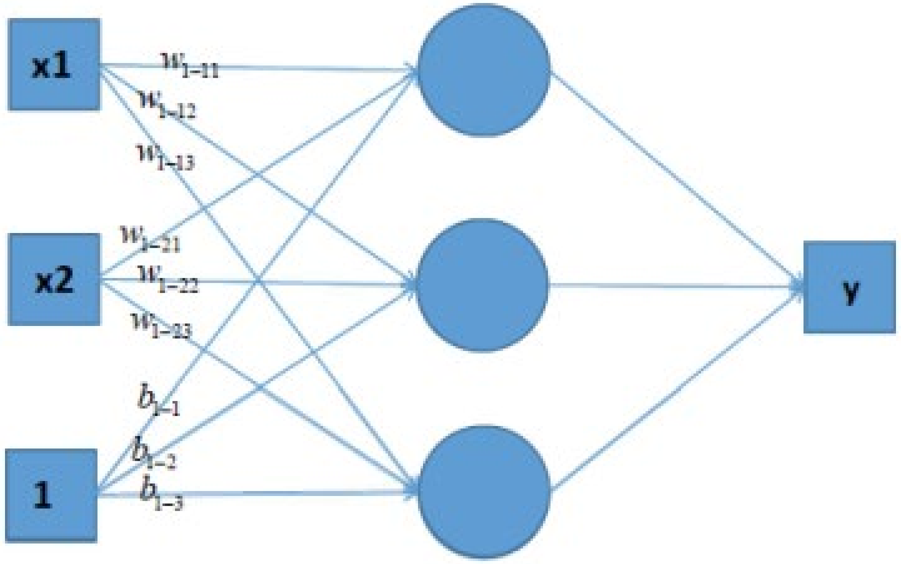
$$y = w_1 x_1 + w_2 x_2 + b$$



$$w_1 = 1, w_2 = 1, b = -2$$

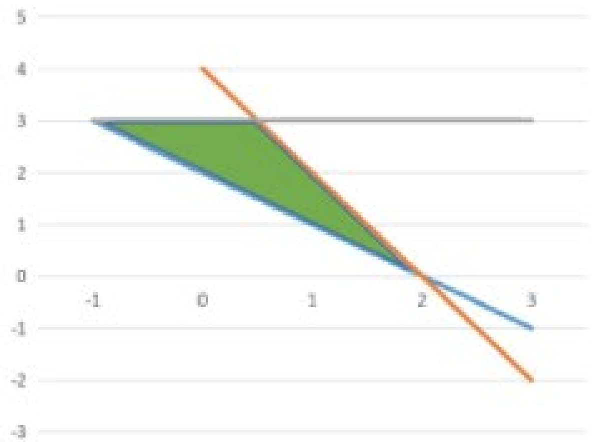
加入隐藏层的神经网络
Network with One Hidden Layer

Perceptron



linear combination of three decision lines

single layer perceptron is a linear classifier



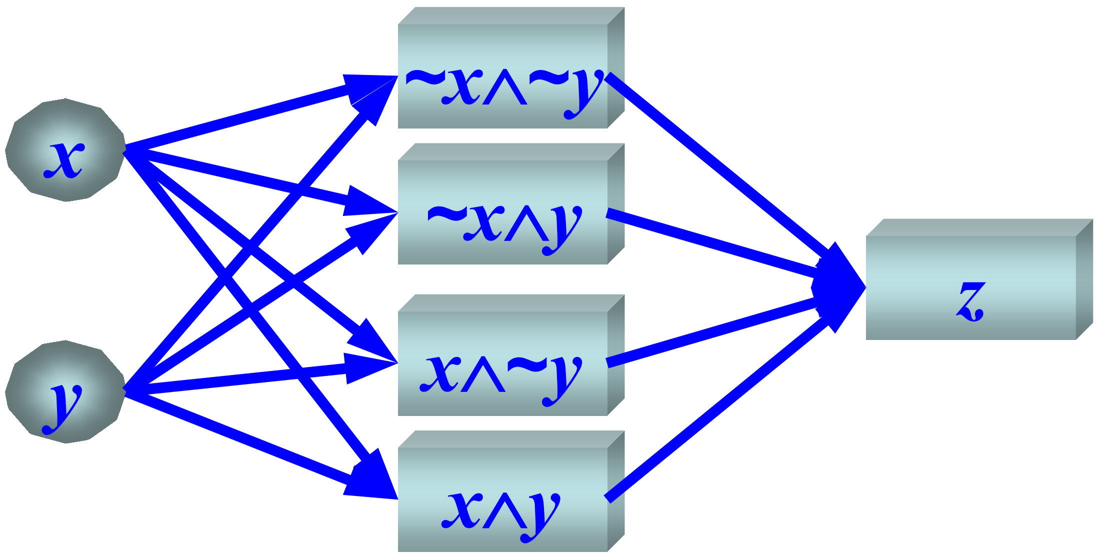
$$w_{1-11}=1, w_{1-12}=1, b_{1-1}=-2$$

$$w_{1-21}=2, w_{1-22}=1, b_{1-2}=4$$

$$w_{1-31}=0, w_{1-32}=1, b_{1-3}=3$$

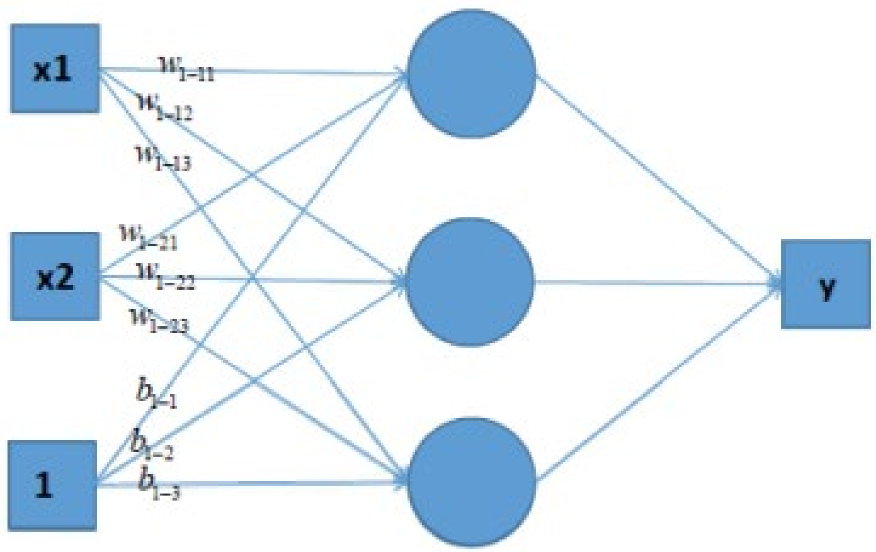
加入隐藏层的神经网络
Network with One Hidden Layer

- All Boolean functions can be represented with two-layers network.



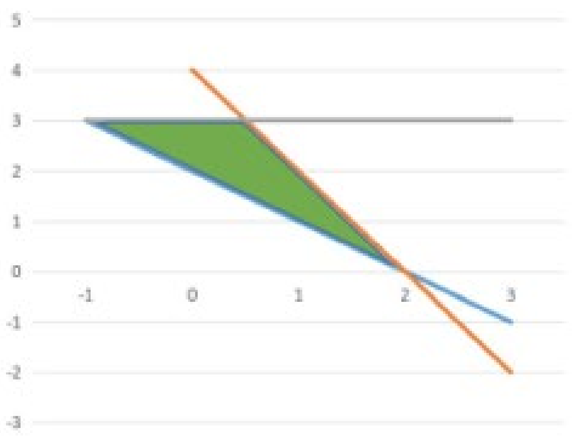
加入隐藏层的神经网络
Network with One Hidden Layer

Perceptron



linear combination of three decision lines

single layer perceptron is a linear classifier

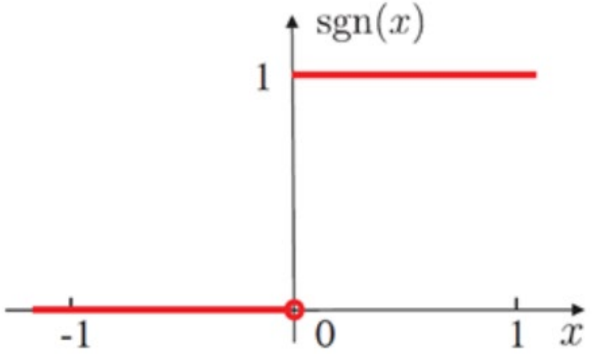


$$w_{1-11}=1, w_{1-12}=1, b_{1-1}=-2$$

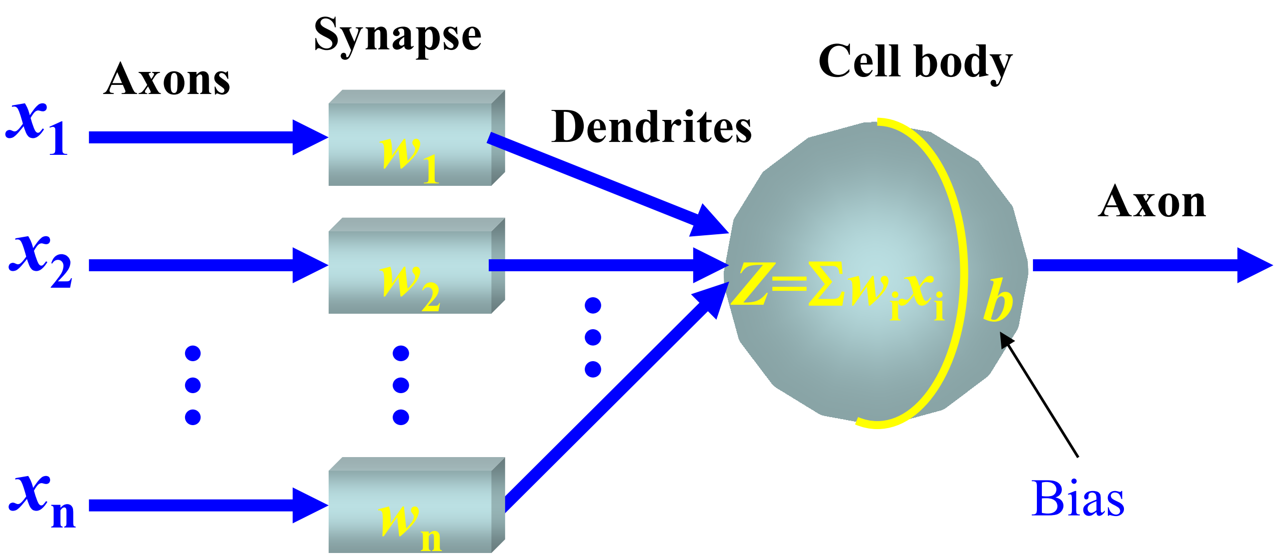
$$w_{1-21}=2, w_{1-22}=1, b_{1-2}=4$$

$$w_{1-31}=0, w_{1-32}=1, b_{1-3}=3$$

人工神经元
Artificial Neuron



$$sgn(z) = \begin{cases} 1 & \text{if } z \geq 0 \\ 0 & \text{if } z < 0 \end{cases}$$



$$y = \sigma\left(\sum w_i x_i - \mathrm{b}\right)$$

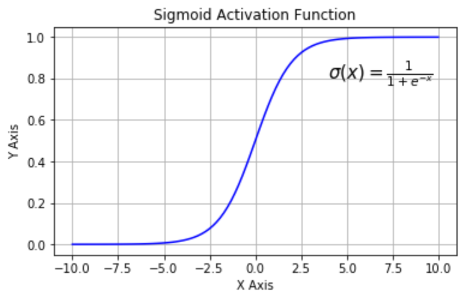
Activation function

激活函数选择
Activation Function

- Activation function must be continuous, differentiable, non-linear, and easy to compute

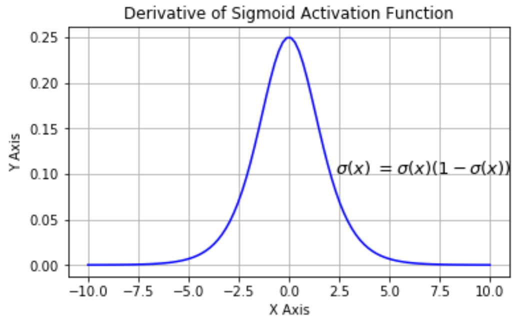
Logistic Sigmoid:

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



Its derivative:

$$\sigma(z)' = \frac{e^{-z}}{(1+e^{-z})^2}$$

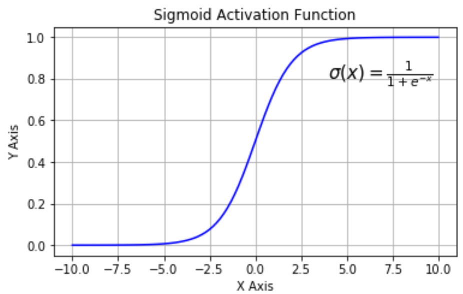


激活函数选择
Activation Function

- Activation function must be continuous, differentiable, non-linear, and easy to compute

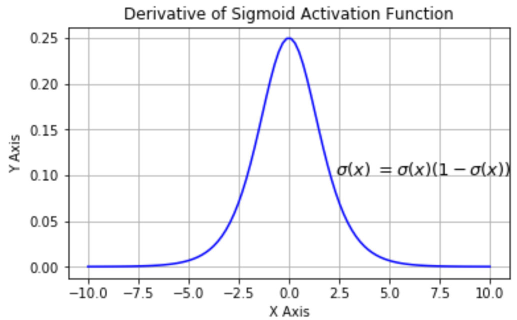
Logistic Sigmoid:

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



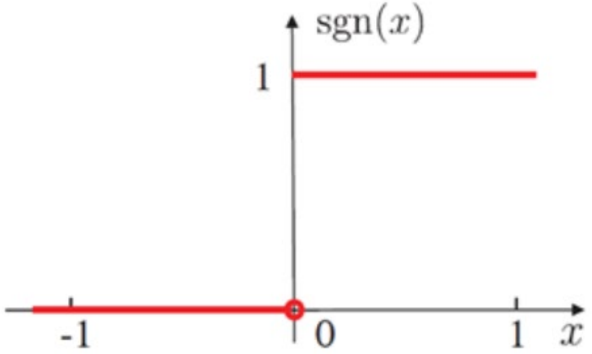
Its derivative:

$$\sigma(z)' = \frac{e^{-z}}{(1+e^{-z})^2} = \sigma(z)(1-\sigma(z))$$

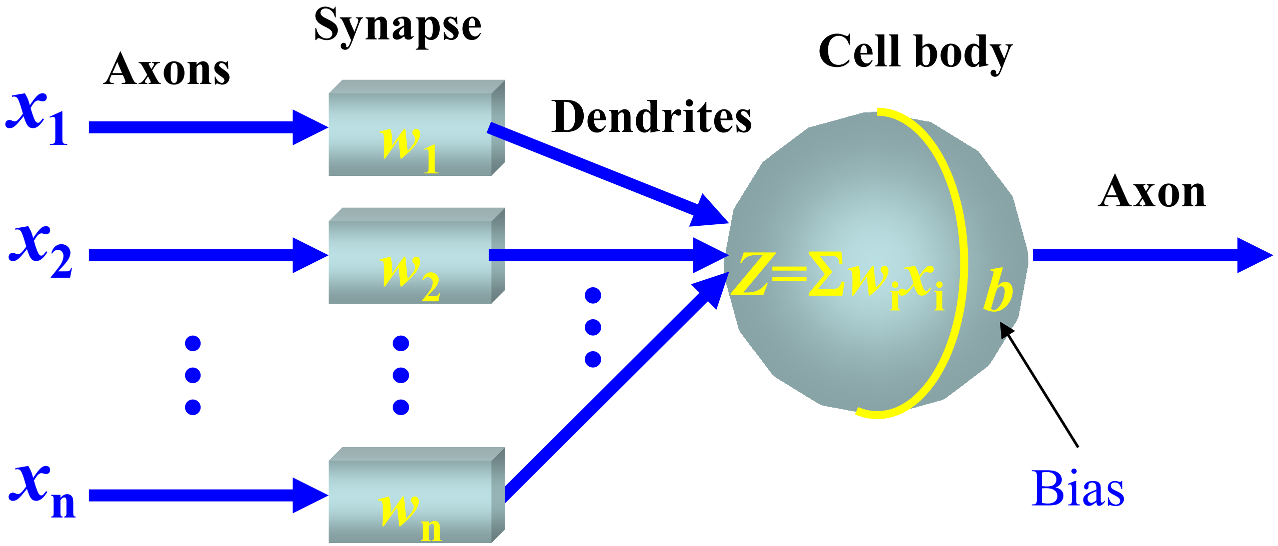


Output range [0,1];
Motivated by biological neurons and can be interpreted as the probability of an artificial neuron 'firing' given its inputs

人工神经元
Artificial Neuron



$$sgn(z) = \begin{cases} 1 & \text{if } z \geq 2 \\ 0 & \text{if } z < 0 \end{cases}$$

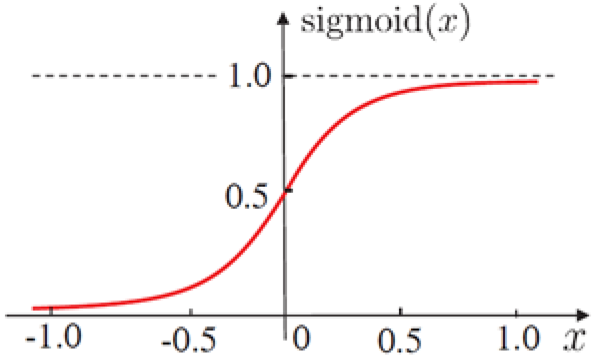


$$y = \sigma(\sum w_i x_i - b)$$

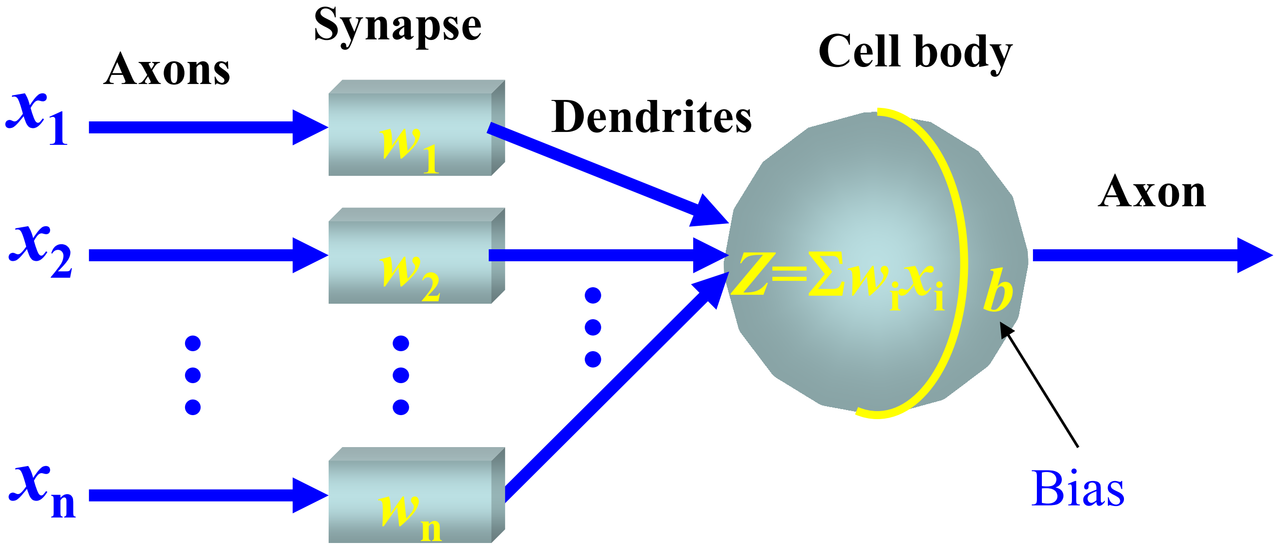
Bias

Activation function

人工神经元
Artificial Neuron



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



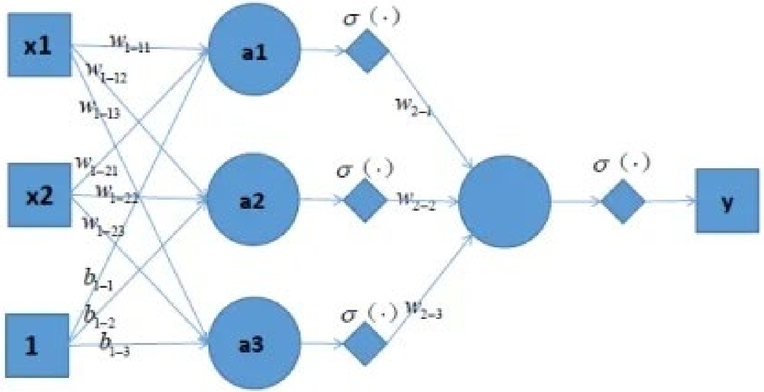
$$y = \sigma(\sum w_i x_i - \text{b})$$

Bias

Activation function

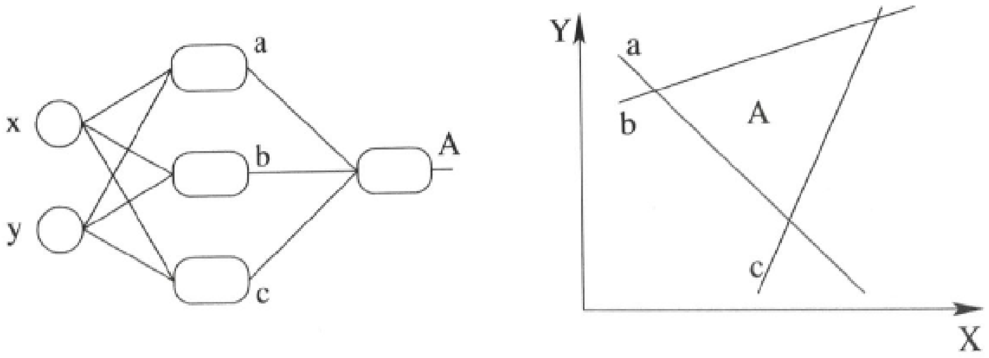
非线性激活函数
Non-linear activation function

Perceptron with non-linear activation function

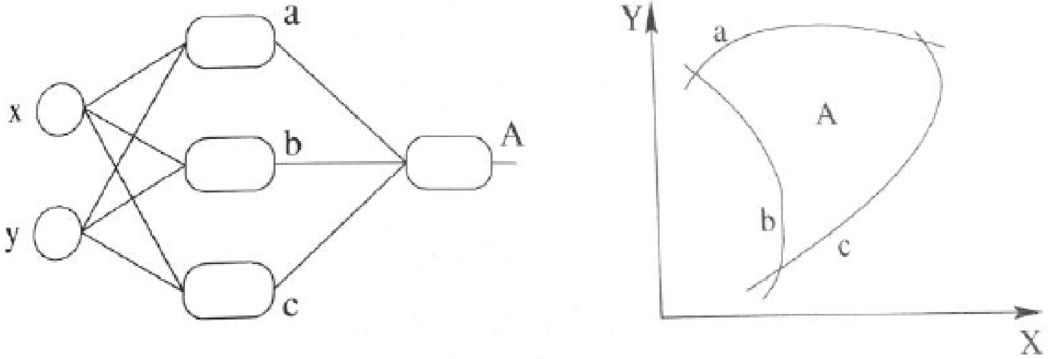


$$a1 = w_{1\text{-}11}x_1 + w_{1-21}x_2 + b_{1\text{-}1}$$
$$a2 = w_{1\text{-}12}x_1 + w_{1-22}x_2 + b_{1\text{-}2}$$
$$a3 = w_{1\text{-}13}x_1 + w_{1-23}x_2 + b_{1\text{-}3}$$

$$y = \sigma(w_{2-1}\sigma(a1) + w_{2-2}\sigma(a2) + w_{2-3}\sigma(a3))$$



with step activation function



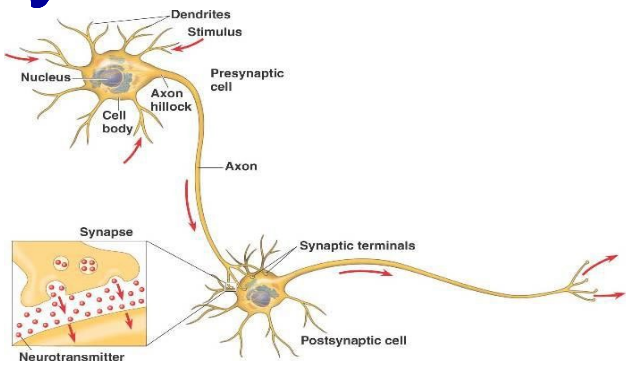
with sigmoid activation function

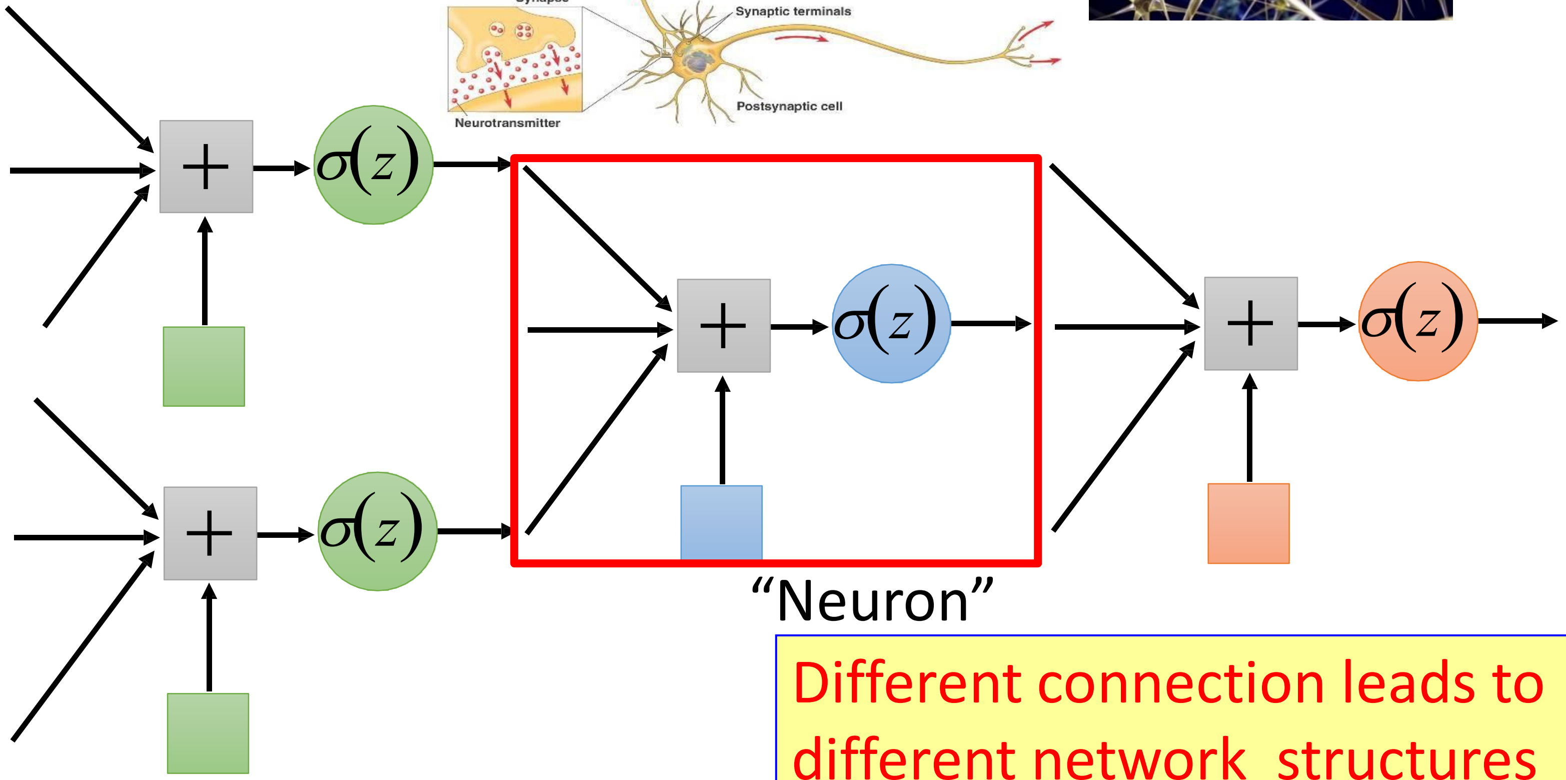
神经网络的标准结构
Standard structure of an ANN

- Input units
  - represents the input as a fixed-length vector of numbers (user defined)
- Hidden units
  - represent intermediate calculations that the network learns;
  - Usually just one (i.e., a 2-layer net)
- Output units
  - represent the output as a fixed length vector of numbers

- *Units are connected by links.*
- *Each link has a numeric weight.*

多层神经网络
Multi-layers Neural Network





Different connection leads to different network structures

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

监督学习
Supervised Learning

- Given the training dataset of (data,label) pairs,

$$D = \{(x^{(i)}, y^{(i)})\}_{i=1.2,\ldots,N}$$

let the machine learn a function from data to label

$$y^{(i)} \approx \boxed{f_\theta(x^{(i)})}$$

- Function set $\{f_\theta(x^{(i)})\}$ is called hypothesis space
- Learning is referred to as updating the parameter θ to make the prediction closed to the corresponding label

监督学习
Supervised Learning

- Given the training dataset of (data,label) pairs,

$$D = \{(x^{(i)}, y^{(i)})\}_{i=1.2,\ldots,N}$$

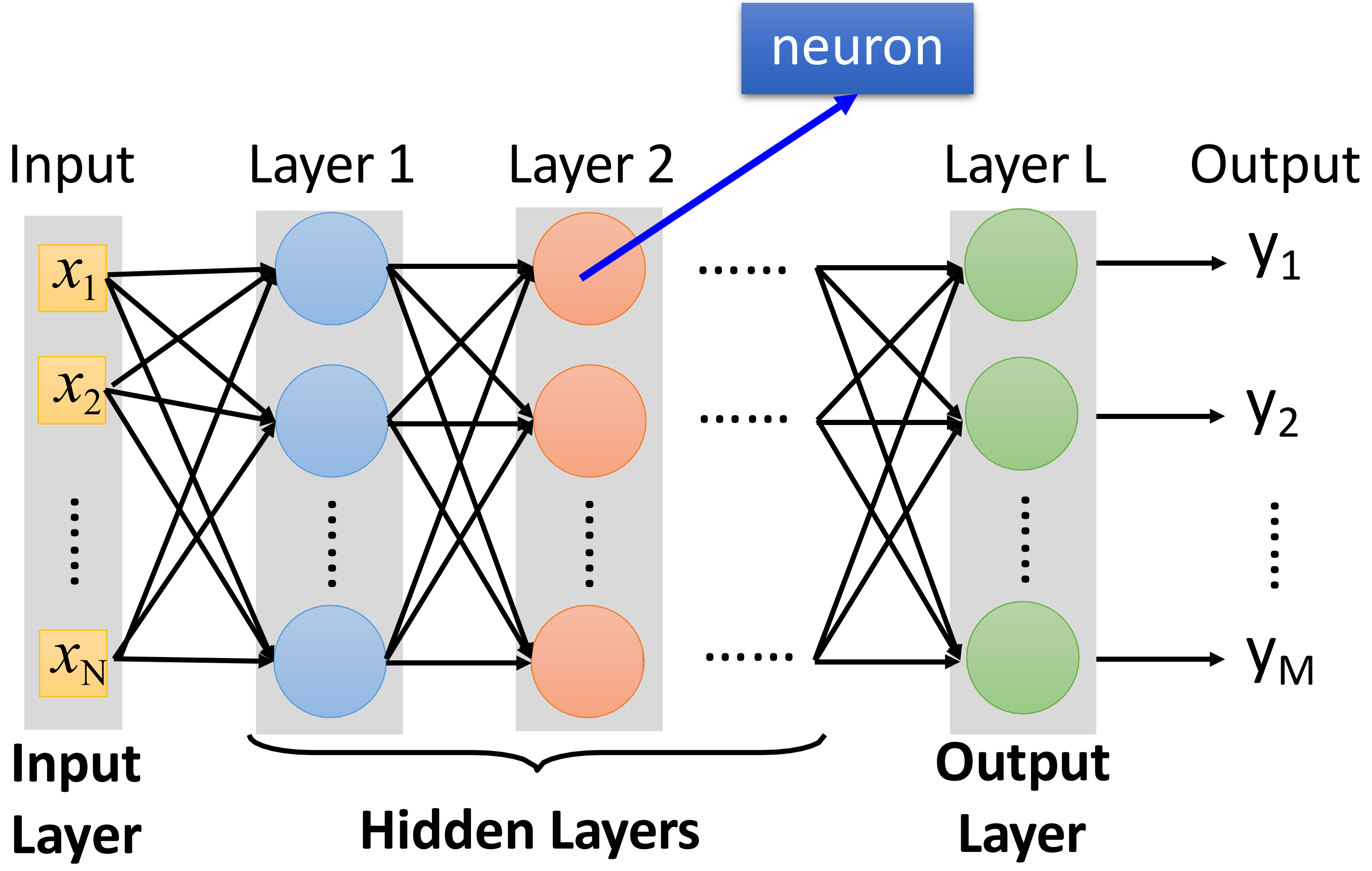
let the machine learn a function from data to label

$$y^{(i)} \approx \boxed{f_\theta(x^{(i)})} \Rightarrow$$

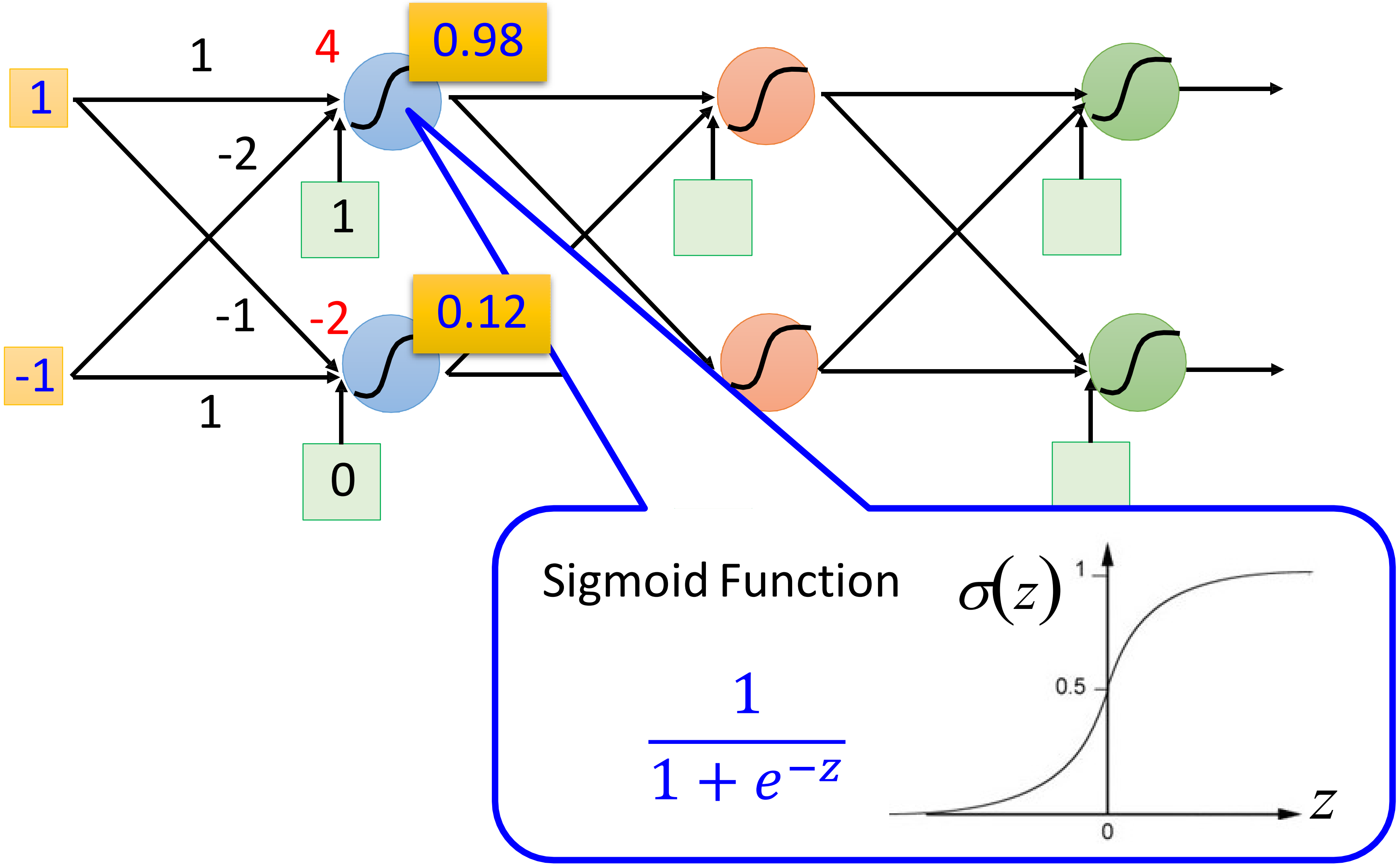
neural network

- Function set $\{f_\theta(x^{(i)})\}$ is called hypothesis s
- Learning is referred to as updating the parar ediction closed to the corresponding label

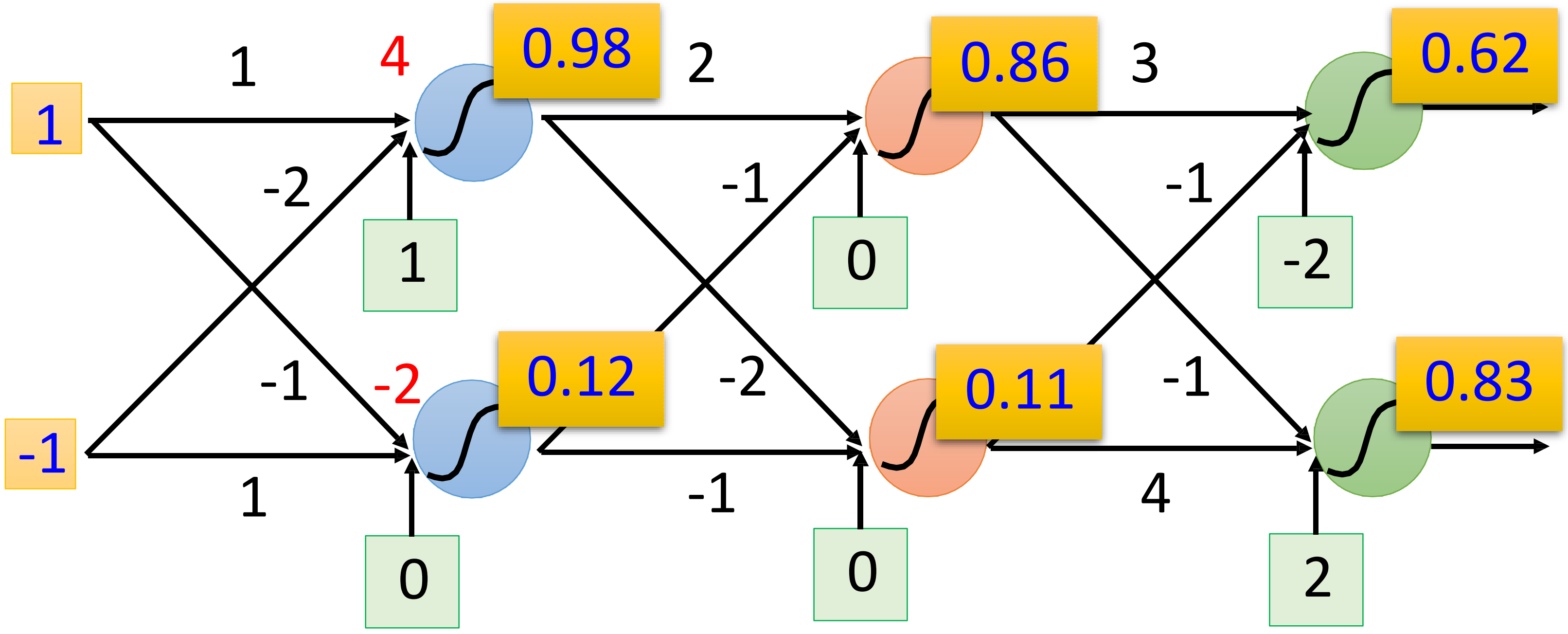
全连接前馈神经网络
Fully Connect Feedforward Network



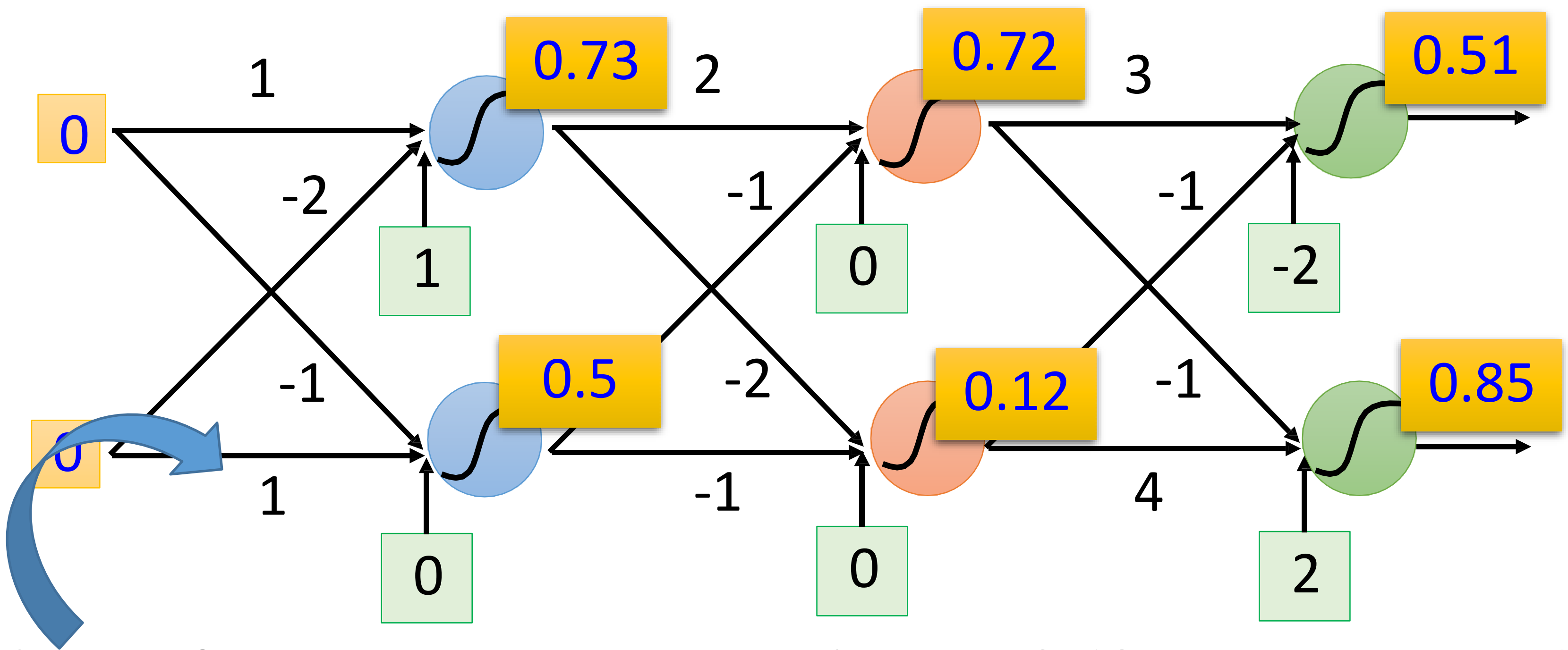
全连接前馈神经网络
Fully Connect Feedforward Network



全连接前馈神经网络
Fully Connect Feedforward Network



全连接前馈神经网络
Fully Connect Feedforward Network



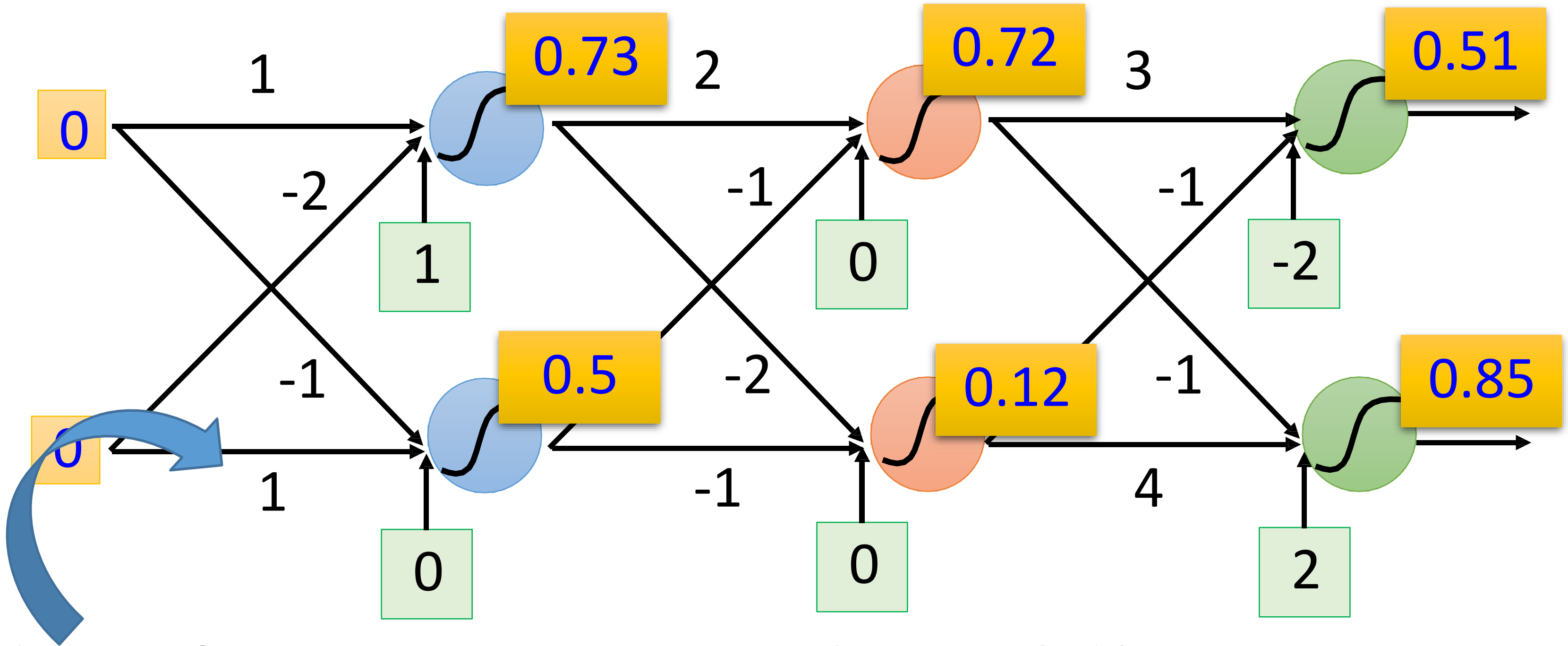
This is a function.

Input vector, output vector

$$f\left(\begin{bmatrix}1\\-1\end{bmatrix}\right)=\begin{bmatrix}0.62\\0.83\end{bmatrix}\quad f\left(\begin{bmatrix}0\\0\end{bmatrix}\right)=\begin{bmatrix}0.51\\0.85\end{bmatrix}$$

Given network structure, define ***a function set***

全连接前馈神经网络
Fully Connect Feedforward Network



This is a function.

Input vector, output vector

$$f\left(\begin{bmatrix}1\\-1\end{bmatrix}\right)=\begin{bmatrix}0.62\\0.83\end{bmatrix}\quad f\left(\begin{bmatrix}0\\0\end{bmatrix}\right)=\begin{bmatrix}0.51\\0.85\end{bmatrix}$$

Given network structure, define ***a function set***

Different w and bias

问题1：学习目标
What is the Learning Objective?

- Make the prediction closed to the corresponding label

$$\min_{\theta} \quad \frac{1}{N}\sum_{i=1}^{N} L(y^{(i)}, f_{\theta}(x^{(i)}))$$

(Empirical Risk Minimization, ERM)

Loss function $L(y^{(i)}, f_{\theta}(x^{(i)}))$ measures the error between the label and prediction for single sample.

The definition of loss function depends on the data and task

损失函数
Loss function

0-1 Loss Function

$$L\left(y^{(i)}, f\left(x^{(i)}\right)\right) = \begin{cases} 1, & \text{if } y^{(i)} \neq f\left(x^{(i)}\right) \\ 0, & \text{if } y^{(i)} = f\left(x^{(i)}\right) \end{cases}$$

Mean Squared Error, MSE

$$L\left(y^{(i)}, f\left(x^{(i)}\right)\right) = \left(y^{(i)} - f\left(x^{(i)}\right)\right)^2$$

Absolute Loss Function

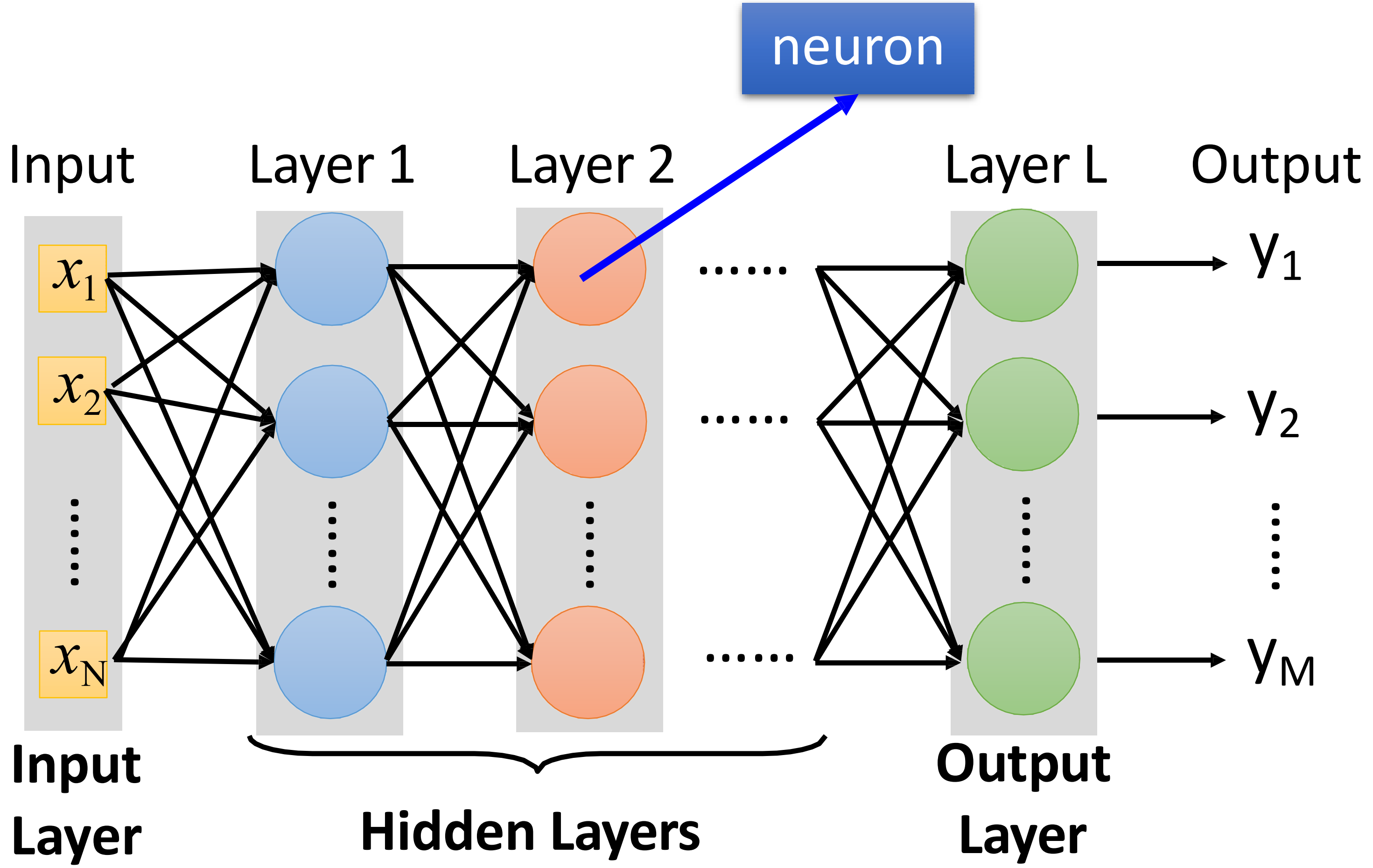
$$L\left(y^{(i)}, f\left(x^{(i)}\right)\right) = \left|y^{(i)} - f\left(x^{(i)}\right)\right|$$

Logarithmic Loss Function (Cross-Entropy Loss Function)

$$L\left(y^{(i)}, p^{(i)}\right) = -\left[y^{(i)} \log\left(p^{(i)}\right) + \left(1 - y^{(i)}\right) \log\left(1 - p^{(i)}\right)\ \right]$$

$y^{(i)} \in \{0,1\}$, $p^{(i)}=f\left(x^{(i)}\right)$ is the predicted probability that the i th sample belongs to the positive class (usually denoted as class 1))

全连接前馈神经网络
Fully Connect Feedforward Network



梯度下降求解
Gradient Descent

Network parameters $\theta = \{w_1, w_2, \cdots, b_1, b_2, \cdots\}$

Starting Parameters $\theta^0 \longrightarrow \theta^1 \longrightarrow \theta^2 \longrightarrow$ ......

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

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

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

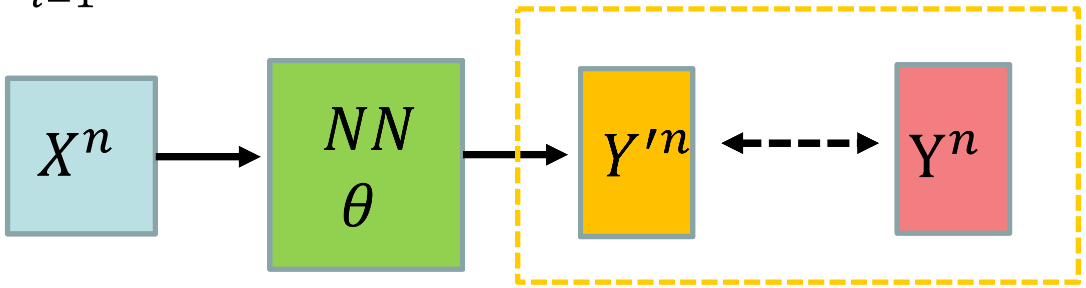
Millions of parameters ......

To compute the gradients efficiently, we use ***backpropagation***.

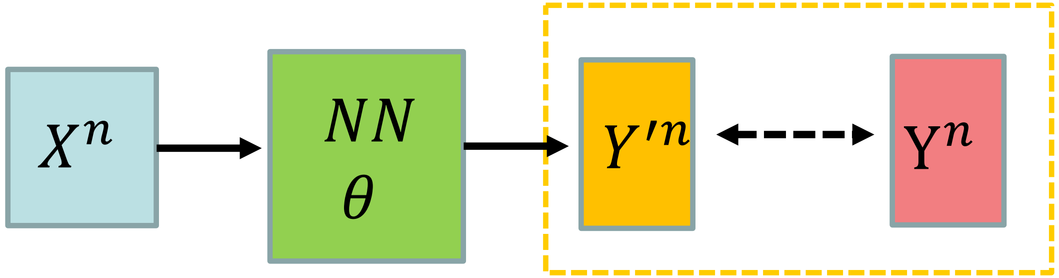
反向传播算法
Backpropagation

Cost function: $\frac{1}{N}\sum_{i=1}^{N} L(y^{(i)}, y'^{(i)})$

Regression:



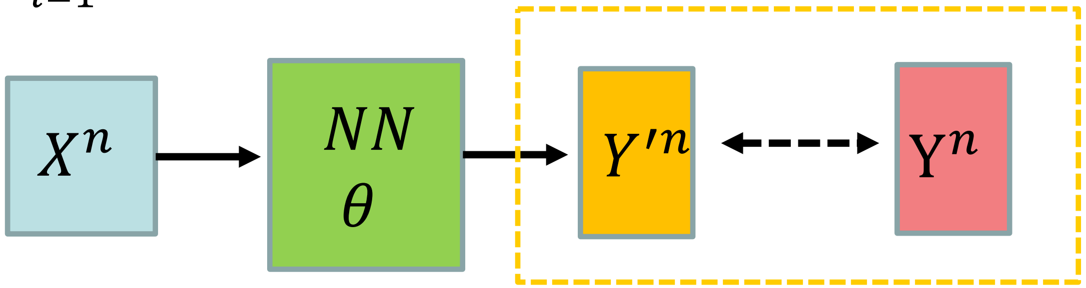
Classification:



反向传播算法
Backpropagation

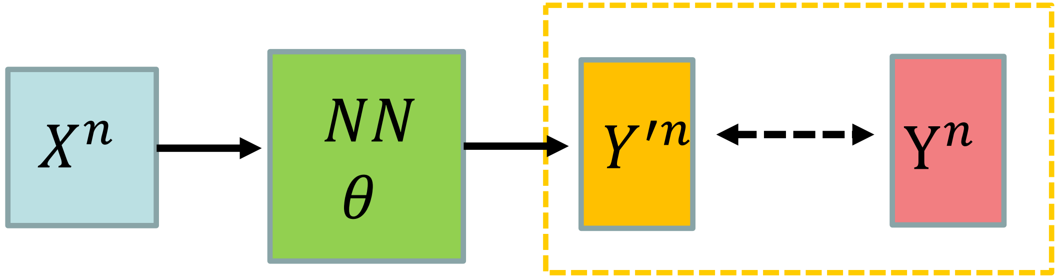
Cost function: $\frac{1}{N}\sum_{i=1}^{N} L(y^{(i)}, y'^{(i)})$

Regression:



Mean Square Error (MSE): $L(y', y) = (y' - y)^2$

Classification:



$$\begin{array}{cccccc} & \text{Class 1} & & \text{Class 2} & & \text{Class 3} \\ \hat{\boldsymbol{y}} = & \begin{bmatrix}1\\0\\0\end{bmatrix} & \text{or} & \begin{bmatrix}0\\1\\0\end{bmatrix} & \text{or} & \begin{bmatrix}0\\0\\1\end{bmatrix} \end{array}$$

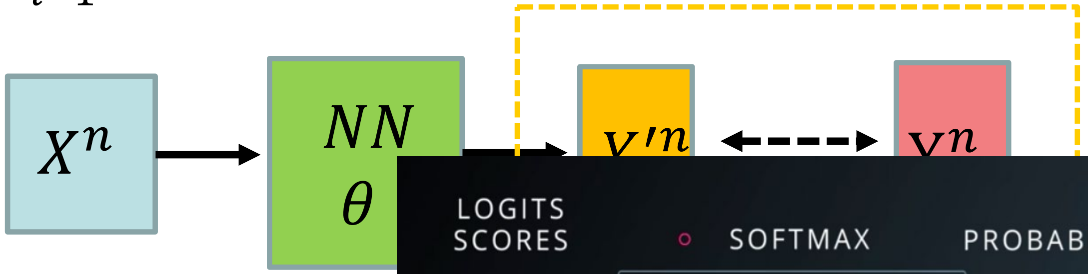
Class as One hot vector

Cross-entropy: $L(y', y) = -\sum y \log y'$

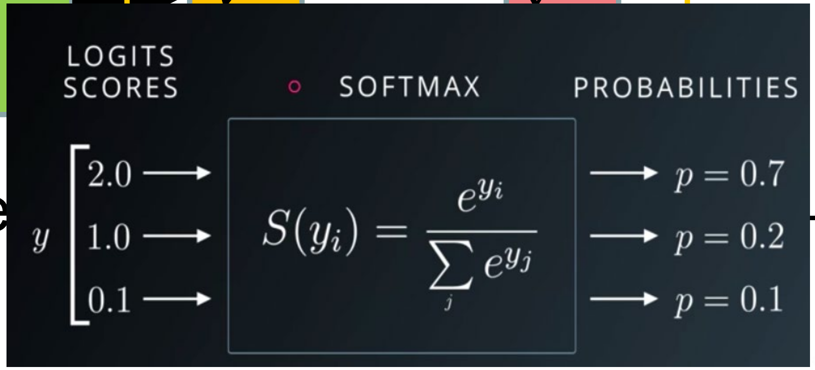
反向传播算法
Backpropagation

Cost function: $\frac{1}{N}\sum_{i=1}^{N} L(y^{(i)}, y'^{(i)})$

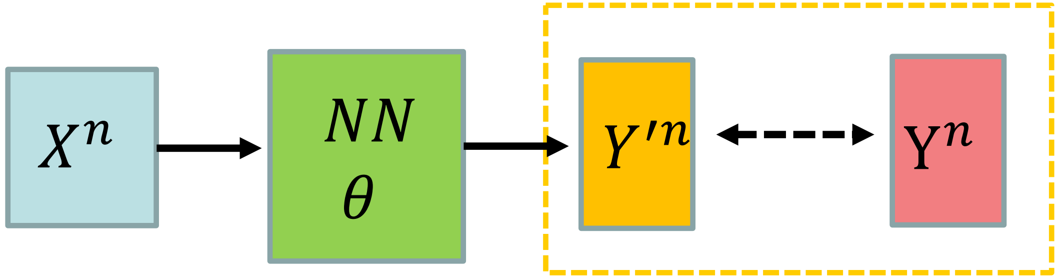
Regression:



Mean Square ... $- y)^2$



Classification:



$$\begin{array}{ccccc} & \text{Class 1} & & \text{Class 2} & & \text{Class 3} \\ \hat{\boldsymbol{y}} = & \begin{bmatrix}1\\0\\0\end{bmatrix} & \text{or} & \begin{bmatrix}0\\1\\0\end{bmatrix} & \text{or} & \begin{bmatrix}0\\0\\1\end{bmatrix}\end{array}$$

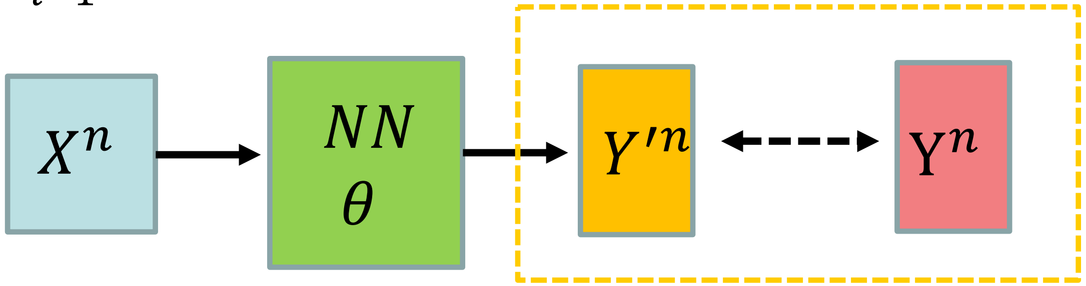
Class as One hot vector

Cross-entropy: $L(y', y) = -\sum y \log y'$

反向传播算法
Backpropagation

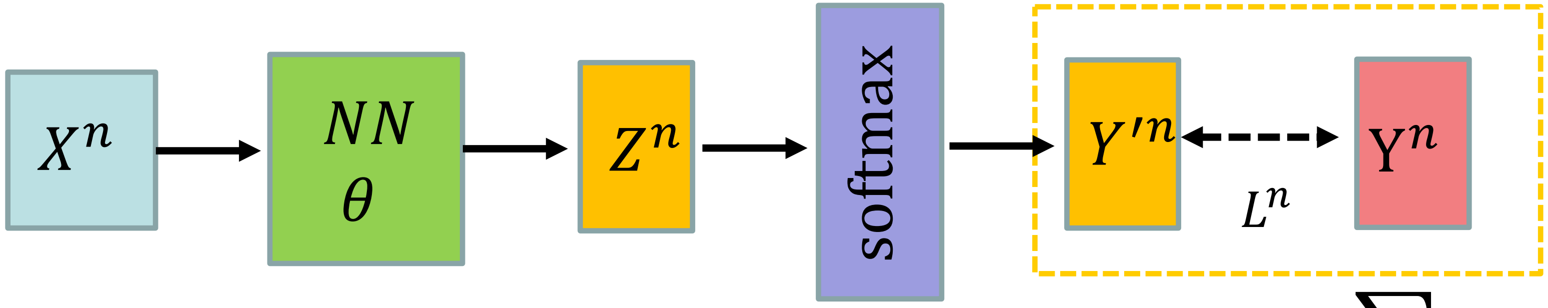
Cost function: $\frac{1}{N}\sum_{i=1}^{N} L(y^{(i)}, y'^{(i)})$

Regression:



Mean Square Error (MSE): $L(y', y) = (y' - y)^2$

Classification:



Class 1 Class 2 Class 3

$$\hat{\boldsymbol{y}} = \begin{bmatrix}1\\0\\0\end{bmatrix} \text{ or } \begin{bmatrix}0\\1\\0\end{bmatrix} \text{ or } \begin{bmatrix}0\\0\\1\end{bmatrix}$$

Class as One hot vector

Cross-entropy: $L(y', y) = -\sum y \log y'$

Softmax回归
Softmax Regression

- Input: $x = (x_1, x_2, \dots x_n)^T$
- Output: class label $\in \{0,1,..K\}$ represented by one-hot vector

$$y = (I(1 = k)\ , I(2 = k) \dots I(K = k))^T$$

- For each class k, there is a weight vector θ_k
- For each class k, the predicted probability is : $p = \dfrac{e^{\theta_k^{\mathrm{T}} x}}{\sum_{\mathrm{j=1}}^{\mathrm{K}} e^{\theta_j^{\mathrm{T}} x}}$
- Using cross-entropy, the cost function:

$$-\sum_{i=1}^{N} \sum_{j=1}^{K} y_j^i \log(p_j^i)$$

- Predicting label:

$$\hat{y} = arg \max_{\mathrm{j=1} \dots K} p_j(x)$$

Softmax回归
Softmax Regression

- Input: $x = (x_1, x_2, \dots x_n)^T$
- Output: class label $\in \{0,1,..K\}$ represented by one-hot vector

$$y = (I(1 = k), I(2 = k) \dots I(K = k))^T$$

e.g. K=3,then $label1 = (1,0,0)^T$
$label2 = (0,1,0)^T$
$label3 = (0,0,1)^T$

- For each class k, there is a weight vector θ_k
- For each class k, the predicted probability is : $p = \frac{e^{\theta_k^{\mathrm{T}} x}}{\sum_{\mathrm{j=1}}^{\mathrm{K}} e^{\theta_j^{\mathrm{T}} x}}$
- Using cross-entropy, the cost function:

$$-\sum_{i=1}^{N} \sum_{j=1}^{K} y_j^i \log(p_j^i)$$

e.g. $y = (0,0,1)^T$, $\mathrm{p} = (0.3,0.3,0.4)^T$

$\mathrm{loss} = -(0 * \log(0.3) + 0 * \log(0.3) + 1 * \log(0.4))$

≈0.916

$y = (0,0,1)^T$, $\mathrm{p} = (0.1,0.1,0.8)^T$

$\mathrm{loss} = -(0 * \log(0.1) + 0 * \log(0.1) + 1 * \log(0.8))$

≈ 0.223

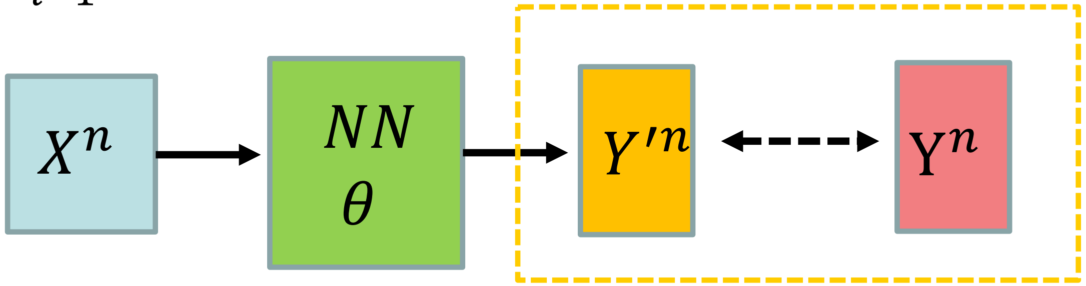
- Predicting label:

$$\hat{y} = arg \max_{\mathrm{j=1\dots}K} p_j(x)$$

反向传播算法
Backpropagation

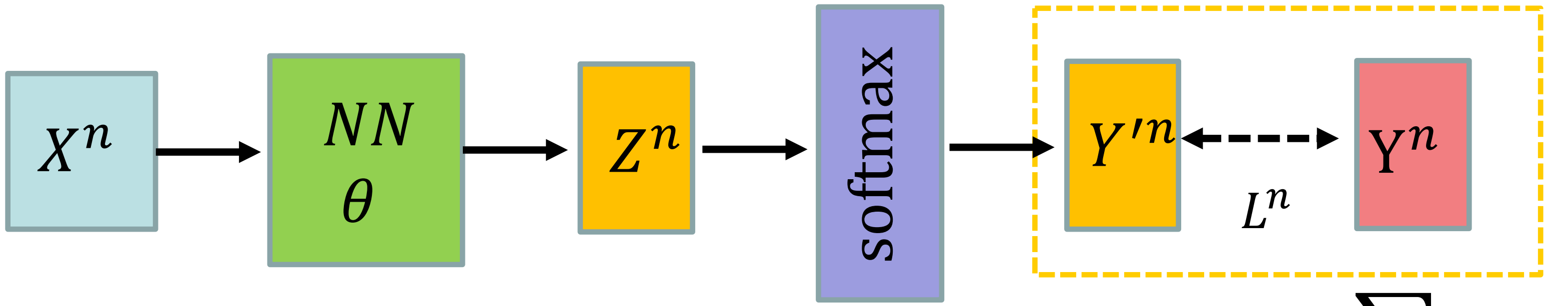
Cost function: $\frac{1}{N}\sum_{i=1}^{N} L(y^{(i)}, y'^{(i)})$

Regression:



Mean Square Error (MSE): $L(y', y) = (y' - y)^2$

Classification:

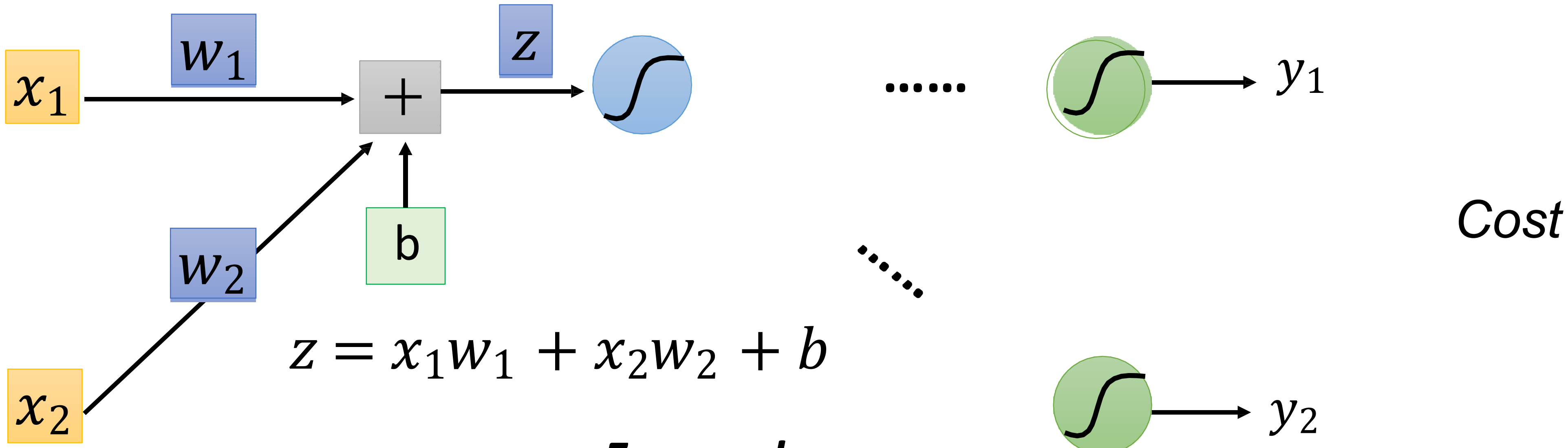


$$\begin{array}{cccccc} & \text{Class 1} & & \text{Class 2} & & \text{Class 3} \\ \hat{\mathbf{y}} = & \begin{bmatrix}1\\0\\0\end{bmatrix} & \text{or} & \begin{bmatrix}0\\1\\0\end{bmatrix} & \text{or} & \begin{bmatrix}0\\0\\1\end{bmatrix} \end{array}$$

Class as One hot vector

Cross-entropy: $L(y', y) = -\sum y \log y'$

反向传播算法
Backpropagation



$$z = x_1 w_1 + x_2 w_2 + b$$

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

(Chain rule)

__Forward pass:__

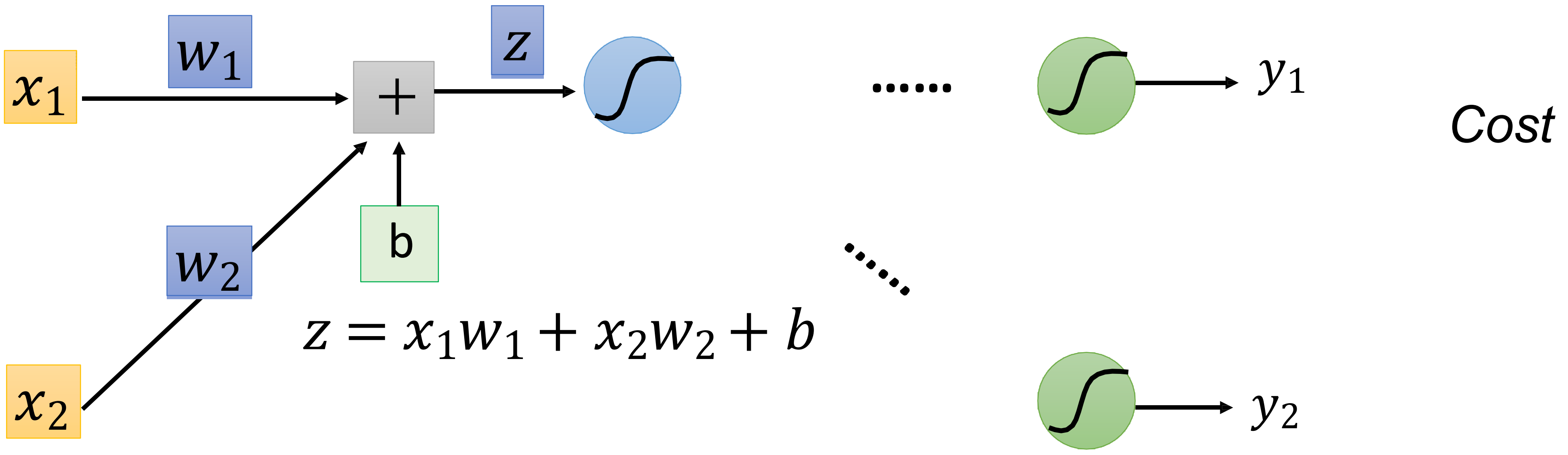
Compute $\partial z / \partial w$ for all parameters

__Backward pass:__

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

反向传播算法之正向传递
Backpropagation-Forward Pass

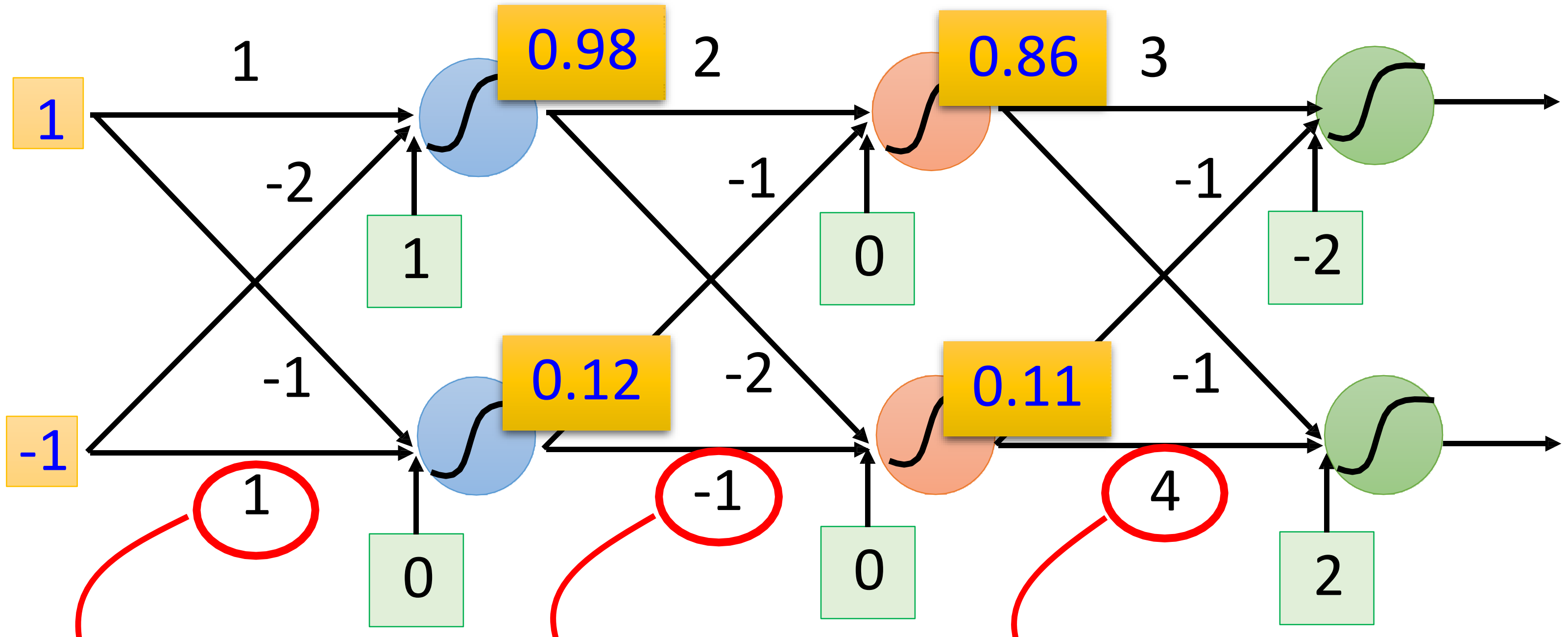
Compute $\partial z/\partial w$ for all parameters



$$\left.\begin{array}{l}\partial z/\partial w_1 = ?\quad x_1 \\ \partial z/\partial w_2 = ?\quad x_2\end{array}\right\}$$ The value of the input connected by the weight

反向传播算法之正向传递
Backpropagation-Forward pass

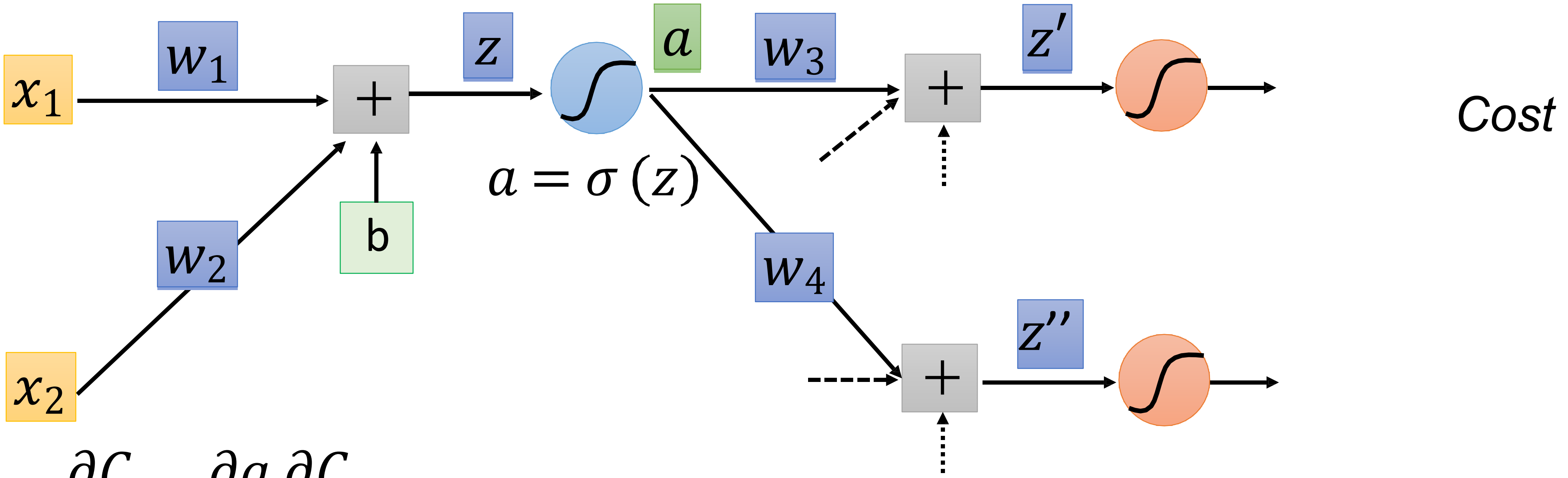
Compute $\partial z/\partial w$ for all parameters



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

反向传播算法-反向传递
Backpropagation-backward Pass

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

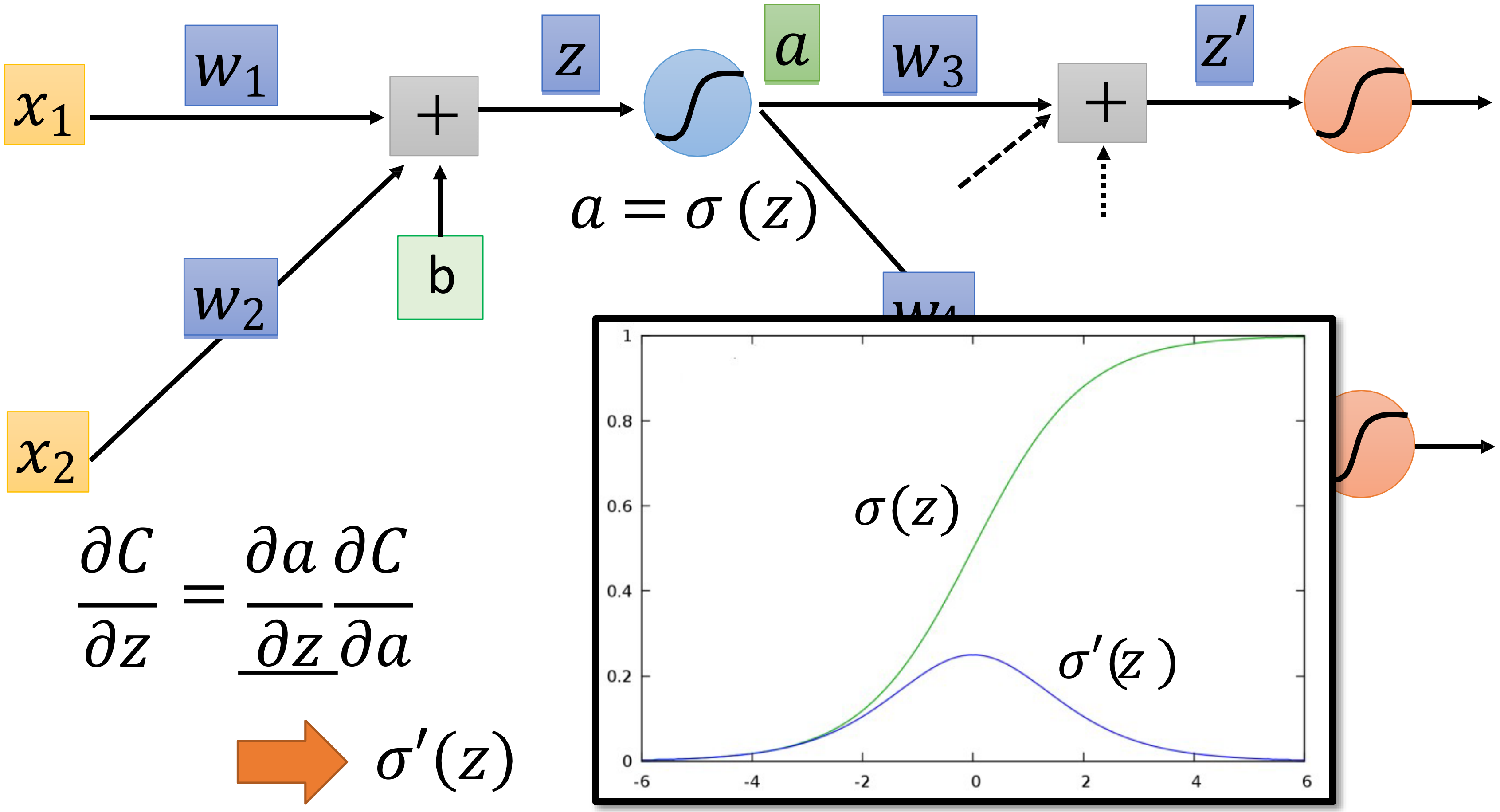


$$\frac{\partial C}{\partial z} = \frac{\partial a}{\partial z}\frac{\partial C}{\partial a}$$

$\Rightarrow \sigma'(z)$

反向传播算法-反向传递
Backpropagation-backward Pass

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

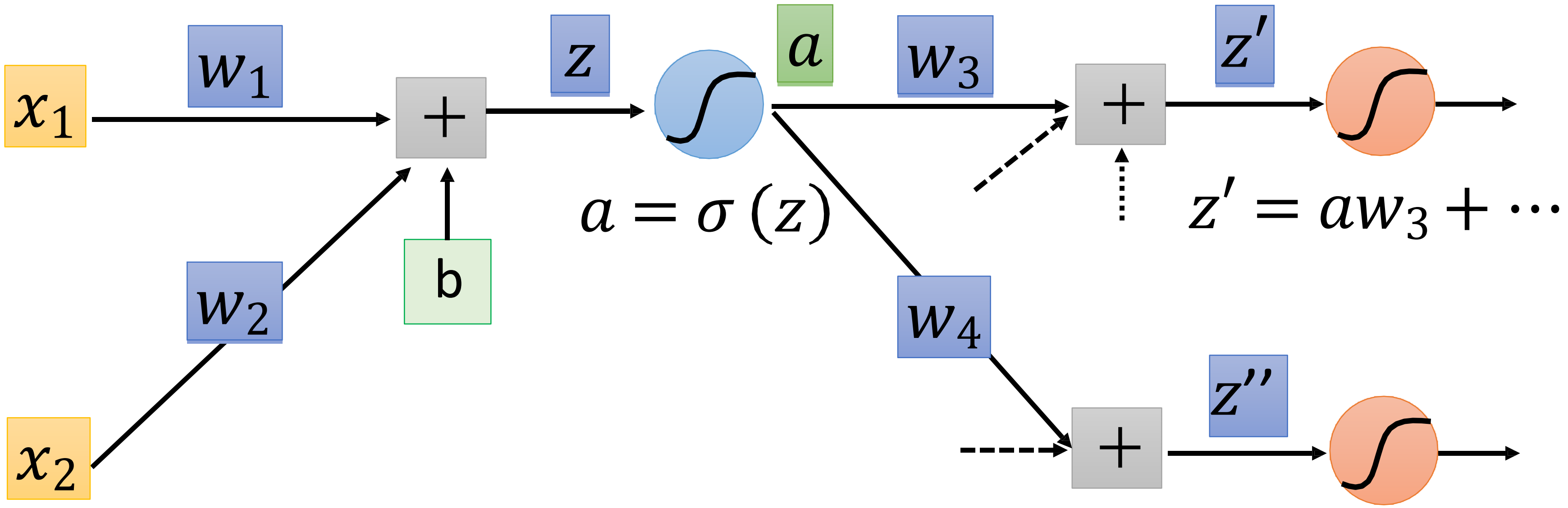


$$\frac{\partial C}{\partial z} = \frac{\partial a}{\partial z}\frac{\partial C}{\partial a}$$

$\sigma'(z)$

反向传播算法-反向传递
Backpropagation-backward Pass

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



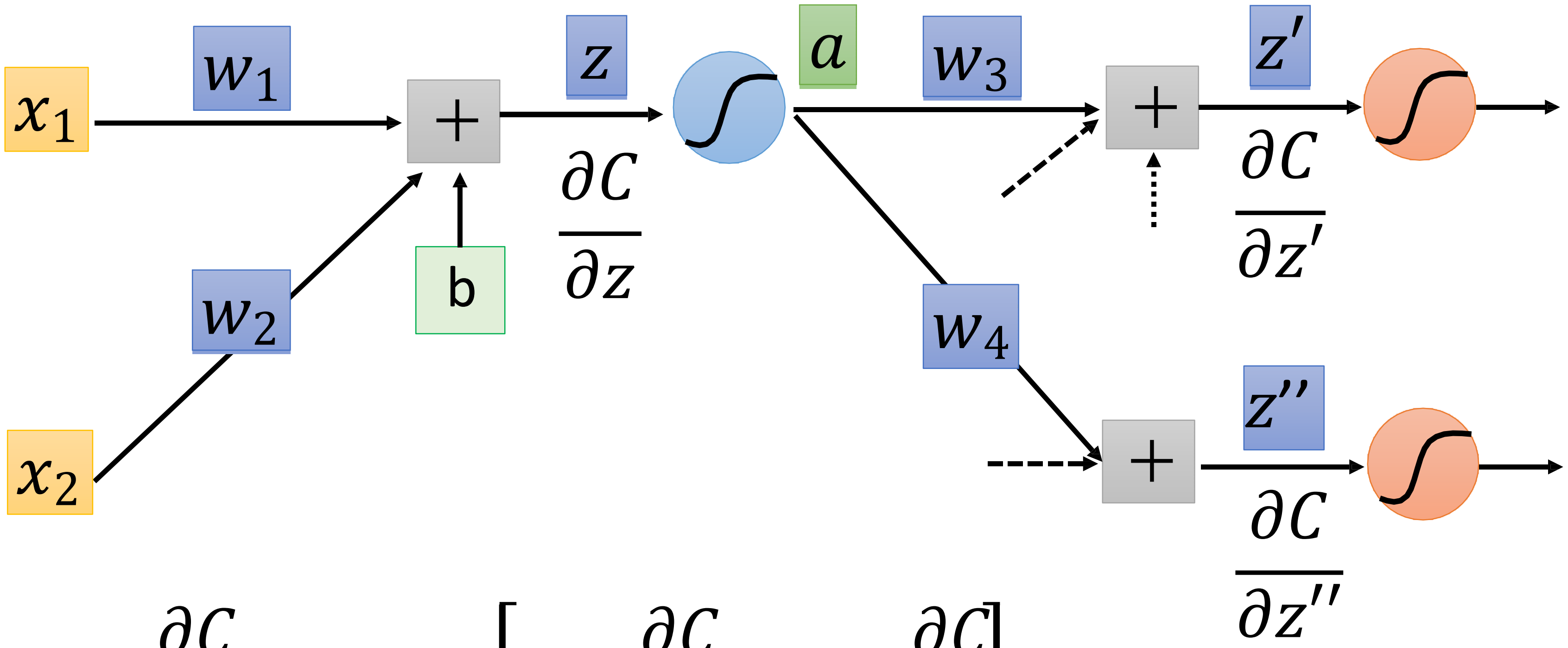
$$\frac{\partial C}{\partial z} = \frac{\partial a}{\partial z}\frac{\partial C}{\partial a} \qquad \frac{\partial C}{\partial a} = \frac{\partial z'}{\partial a}\frac{\partial C}{\partial z'} + \frac{\partial z''}{\partial a}\frac{\partial C}{\partial z''}$$ (Chain rule)

w_3 ? w_4 ?

Assumed it's known

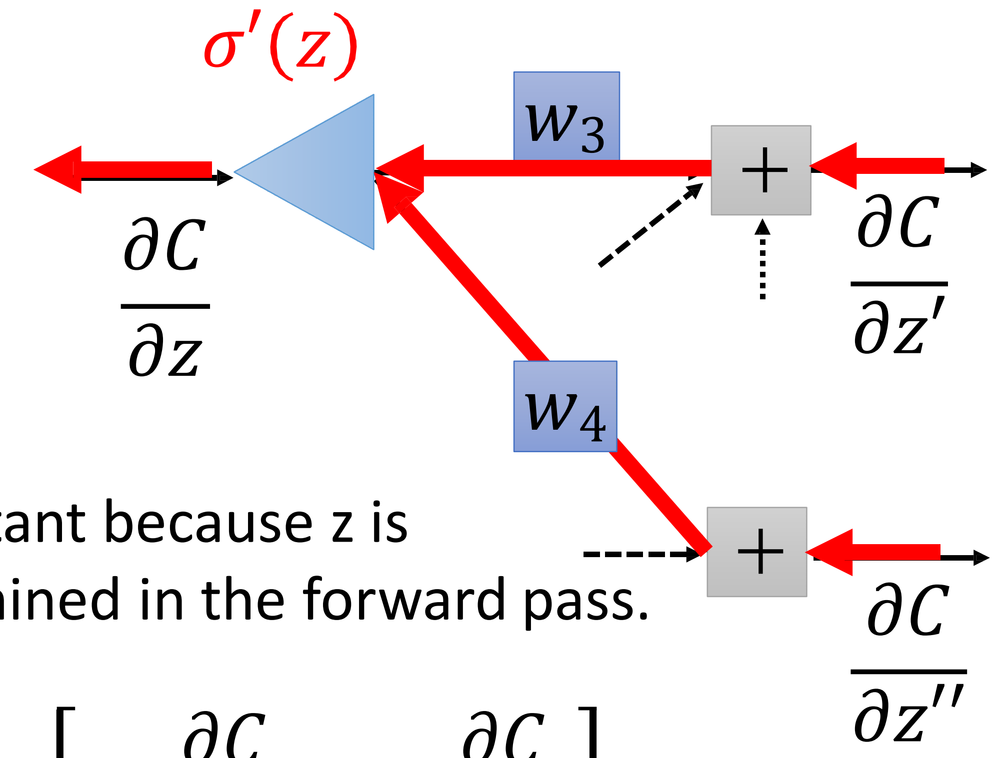
反向传播算法–反向传递
Backpropagation-backward Pass

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



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

反向传播算法-反向传递
Backpropagation-backward Pass

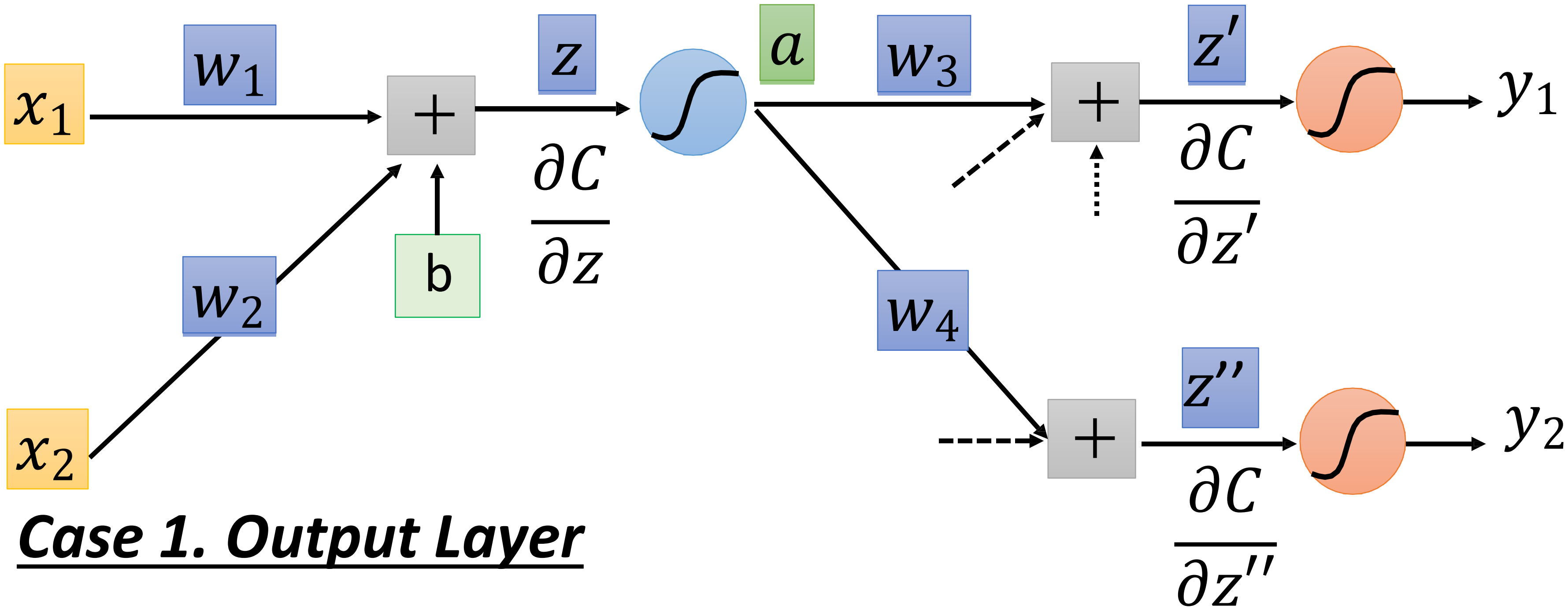


$\sigma'(z)$ is a constant because z is already determined in the forward pass.

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

反向传播算法–反向传递
Backpropagation-backward Pass

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



<u>Case 1. Output Layer</u>

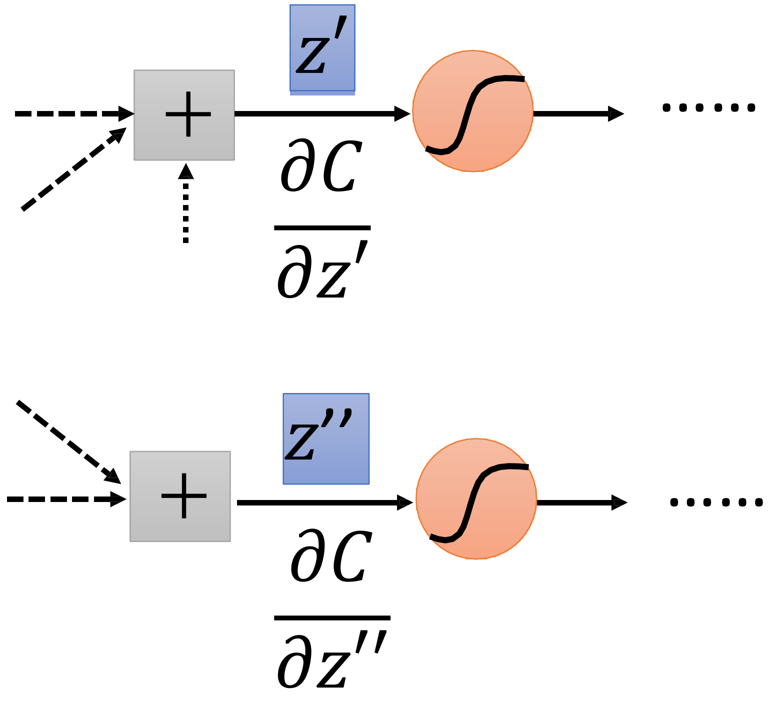
$$\frac{\partial C}{\partial z'} = \frac{\partial y_1}{\partial z'} \frac{\partial C}{\partial y_1} \qquad \frac{\partial C}{\partial z''} = \frac{\partial y_2}{\partial z''} \frac{\partial C}{\partial y_2}$$

Done!

反向传播算法–反向传递
Backpropagation-backward Pass

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

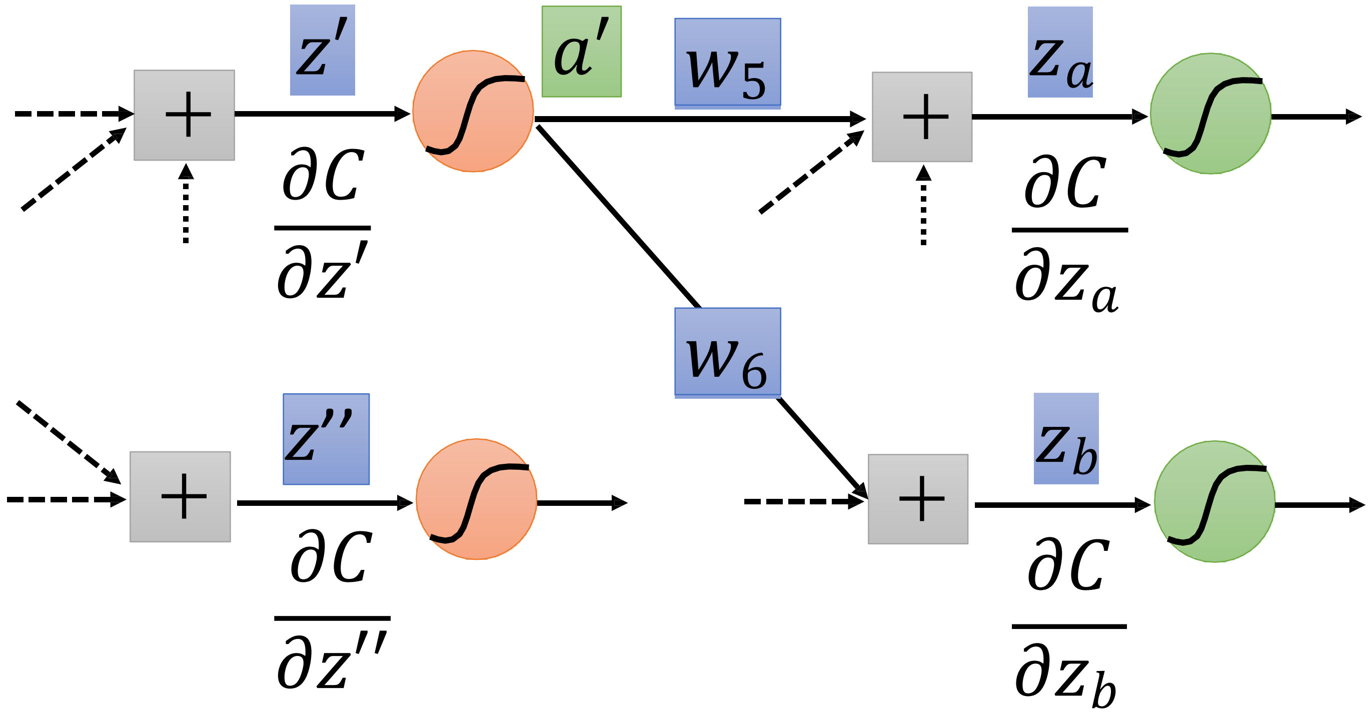
<u>Case 2. Not Output Layer</u>



反向传播算法–反向传递
Backpropagation-backward Pass

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

<u>Case 2. Not Output Layer</u>



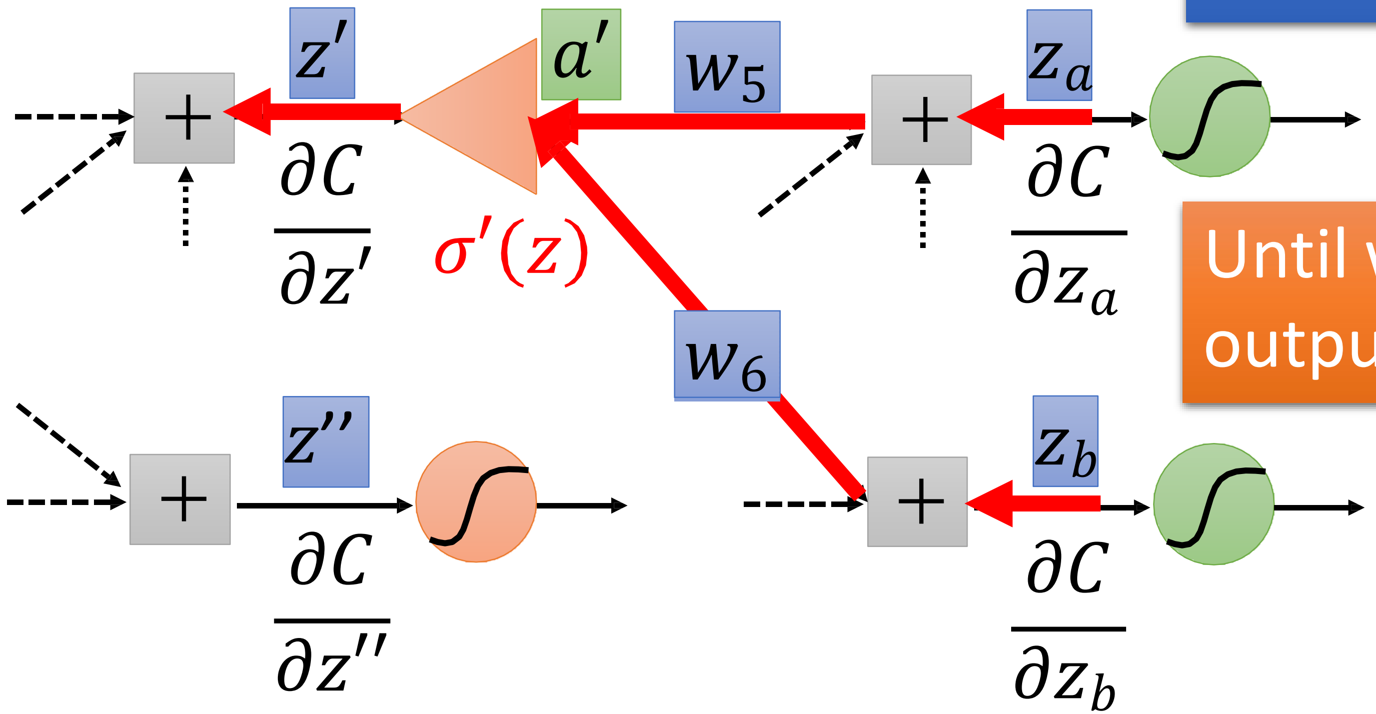
反向传播算法-反向传递
Backpropagation-backward Pass

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

<u>Case 2. Not Output Layer</u>

Compute $\partial C/\partial z$ recursively

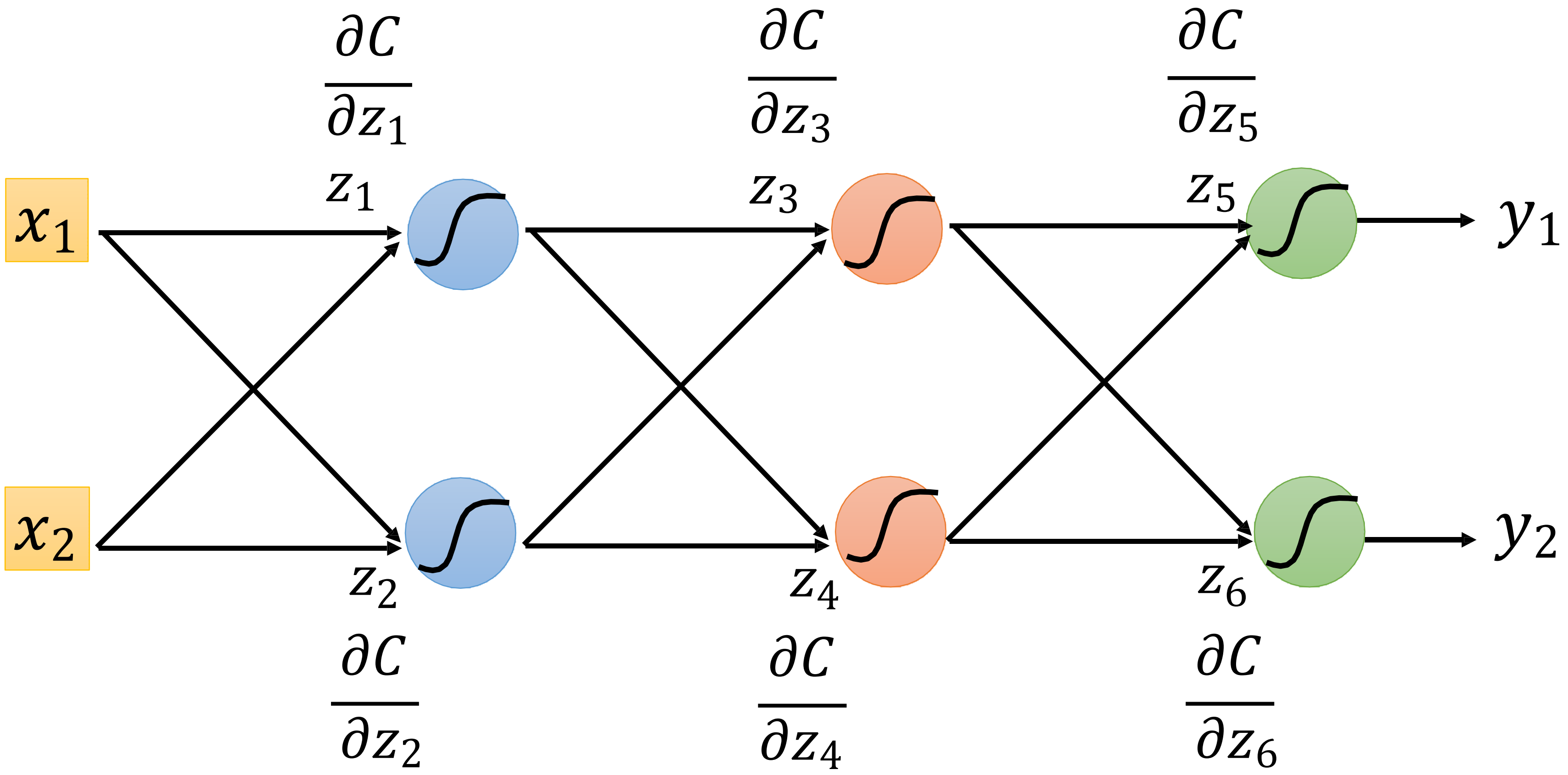
Until we reach the output layer ……



反向传播算法–反向传递
Backpropagation-backward Pass

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

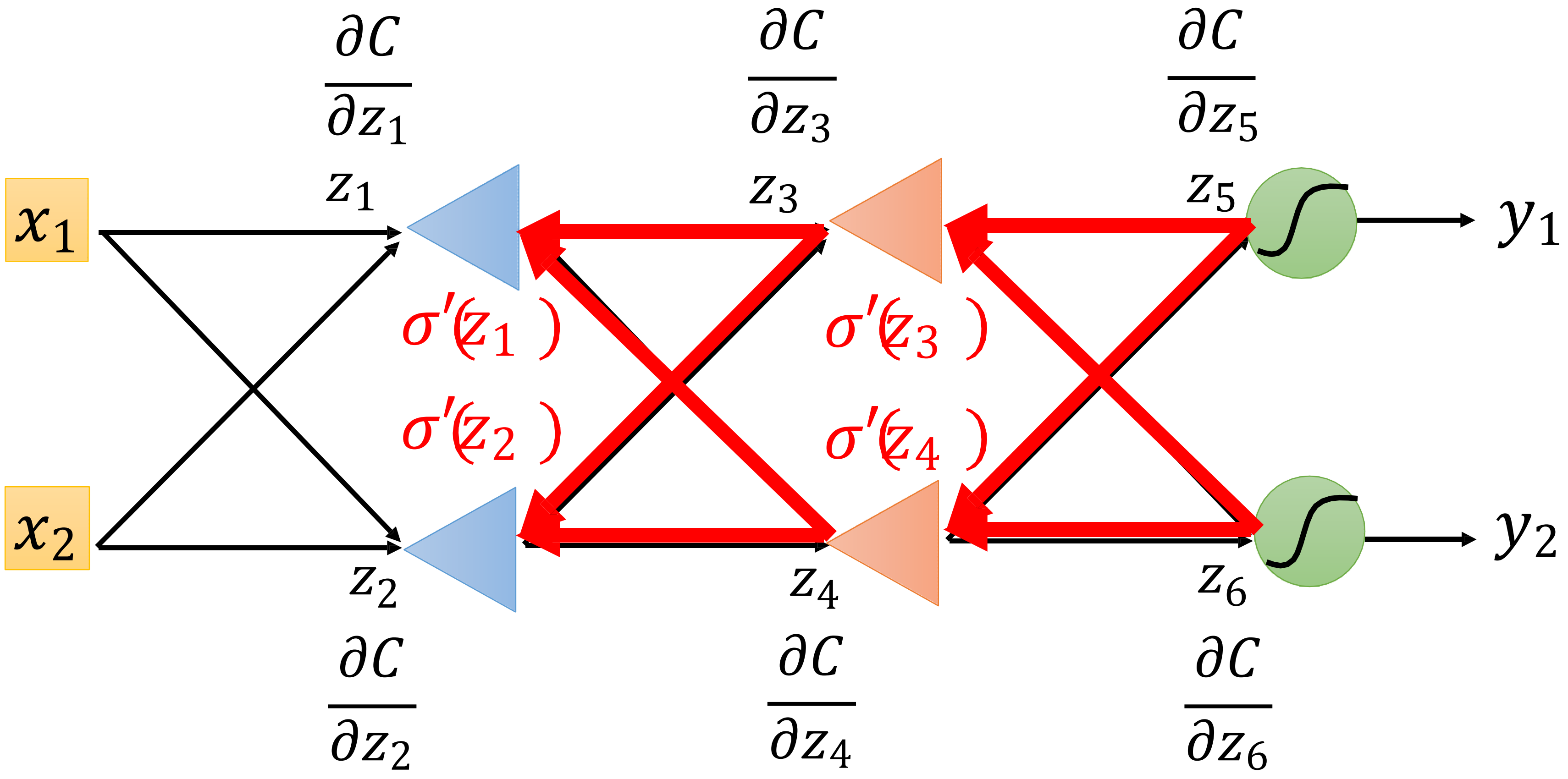
Compute $\partial C/\partial z$ from the output layer



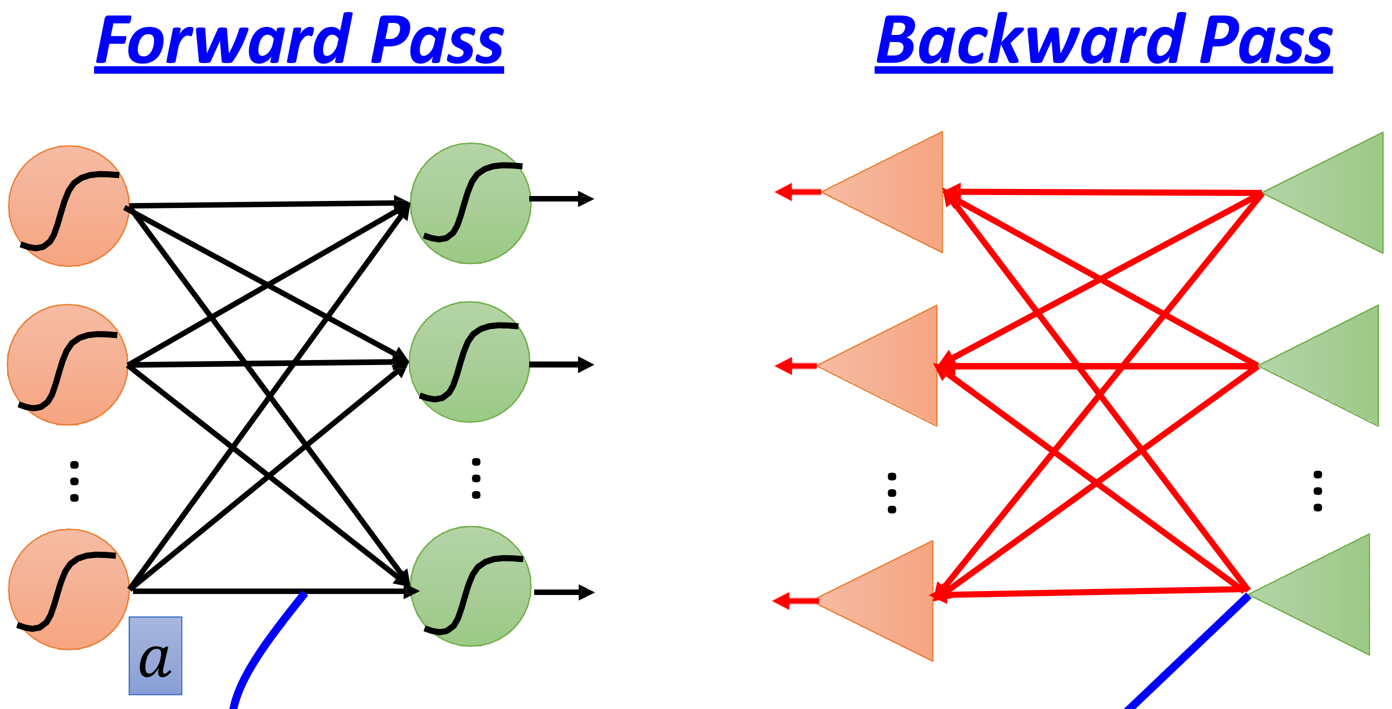
反向传播算法-反向传递
Backpropagation-backward Pass

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

Compute $\partial C/\partial z$ from the output layer



反向传播法
Backpropagation

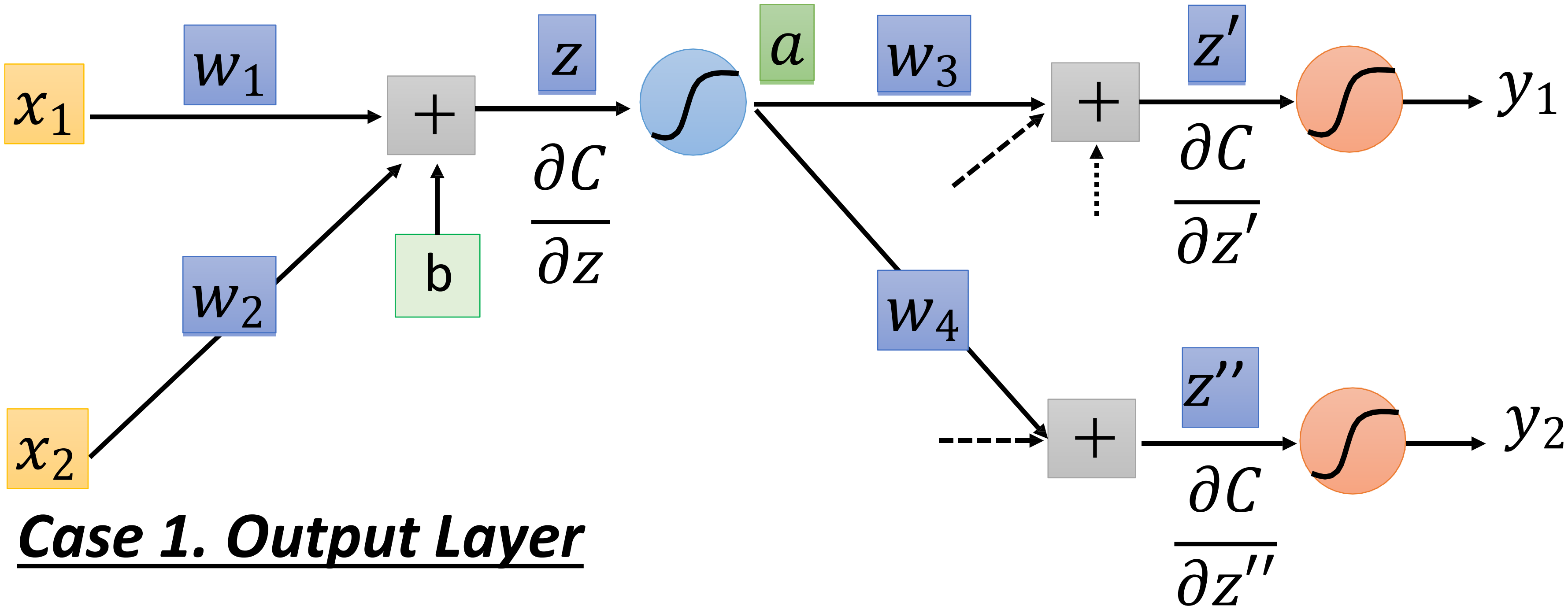


$$\frac{\partial z}{\partial w} = a \quad \times \quad \frac{\partial C}{\partial z} \quad = \frac{\partial C}{\partial w}$$

for all w

反向传播算法–反向传递
Backpropagation-backward Pass

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



Case 1. Output Layer

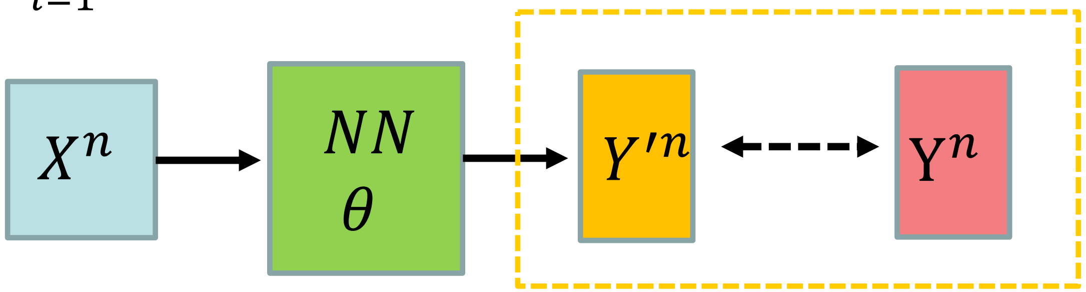
$$\frac{\partial C}{\partial z'} = \frac{\partial y_1}{\partial z'} \frac{\partial C}{\partial y_1} \qquad \frac{\partial C}{\partial z''} = \frac{\partial y_2}{\partial z''} \frac{\partial C}{\partial y_2}$$

Done!

反向传播算法
Backpropagation

Cost function: $\frac{1}{N}\sum_{i=1}^{N} L(y^{(i)}, y'^{(i)})$

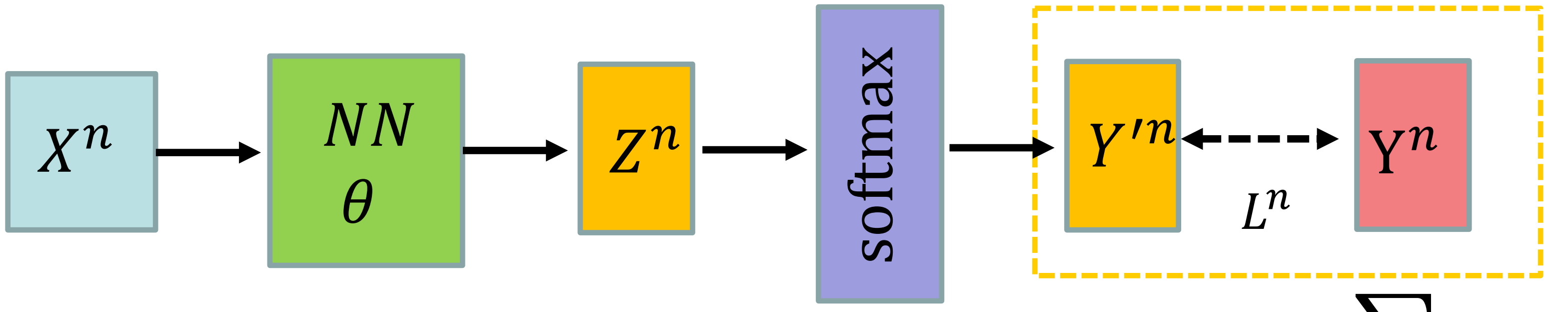
Regression:



$$\frac{\partial L}{\partial y_i'} = y_i' - y_i$$

Mean Square Error (MSE): $L(y', y) = (y' - y)^2$

Classification:

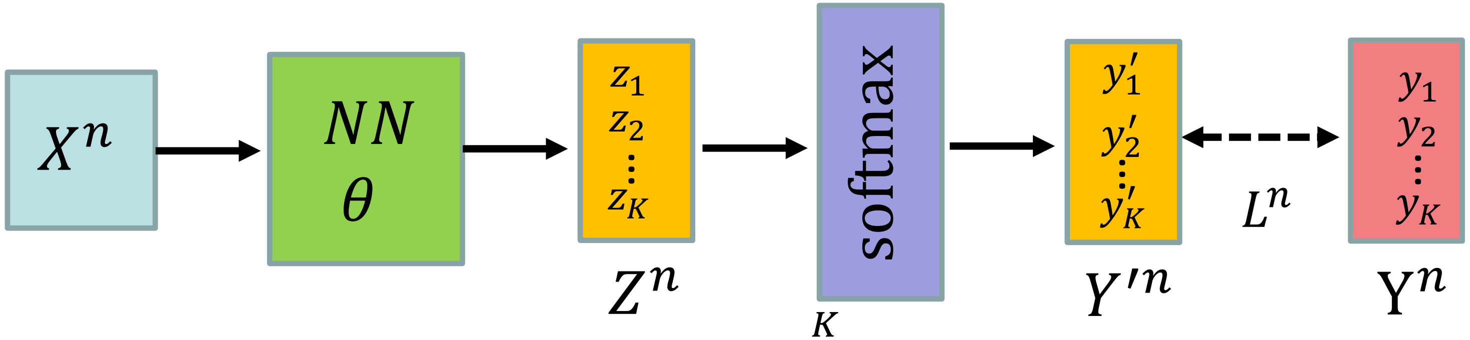


Class 1		Class 2		Class 3
$\hat{\mathbf{y}} = \begin{bmatrix}1\\0\\0\end{bmatrix}$	or	$\begin{bmatrix}0\\1\\0\end{bmatrix}$	or	$\begin{bmatrix}0\\0\\1\end{bmatrix}$

Class as One hot vector

Cross-entropy: $L(y', y) = -\sum y \log y'$

反向传播法
Backpropagation

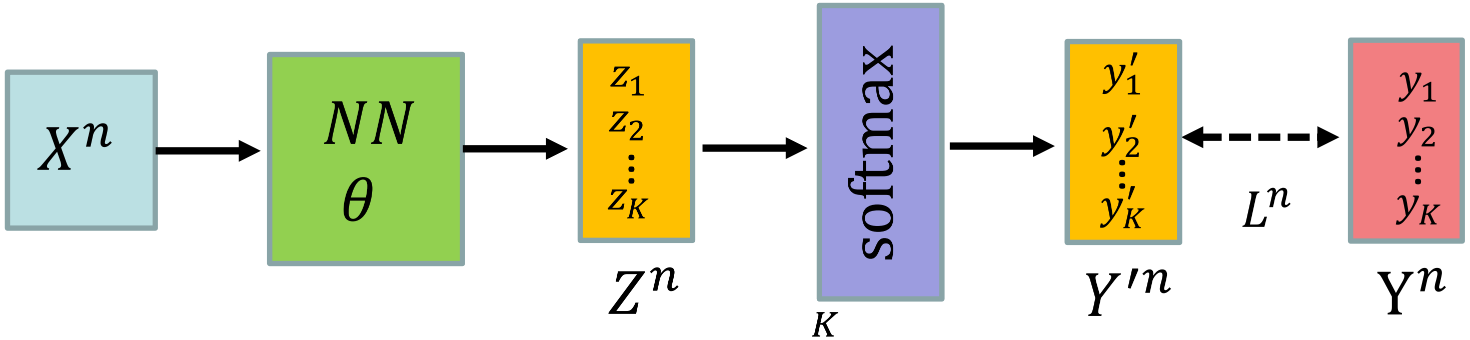


Cross-entropy: $L(Y', Y) = -\sum_{j=1}^{K} y_j \log y'_j$ $\qquad y'_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}}$

$$\frac{\partial y'_j}{\partial z_i} = \frac{\partial \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}}}{\partial z_i} = \frac{\frac{\partial e^{z_j}}{\partial z_i}\sum_{k=1}^{K} e^{z_k} - \frac{\partial \sum_{k=1}^{K} e^{z_k}}{\partial z_i} e^{z_j}}{\left(\sum_{k=1}^{K} e^{z_k}\right)^2} = \frac{-e^{z_i}e^{z_j}}{\left(\sum_{k=1}^{N} e^{z_k}\right)^2} = -y'_i y'_j, \quad i \neq j$$

$$\frac{\partial y'_j}{\partial z_i} = \frac{\partial \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}}}{\partial z_i} = \frac{\frac{\partial e^{z_j}}{\partial z_i}\sum_{k=1}^{K} e^{z_k} - \frac{\partial \sum_{k=1}^{K} e^{z_k}}{\partial z_i} e^{z_j}}{\left(\sum_{k=1}^{N} e^{z_k}\right)^2} = \frac{e^{z_i}\sum_{k=1}^{N} e^{z_k} - e^{z_i}e^{z_j}}{\left(\sum_{k=1}^{N} e^{z_k}\right)^2} = y'_i - y'_i y'_j, \qquad i = j$$

反向传播法
Backpropagation

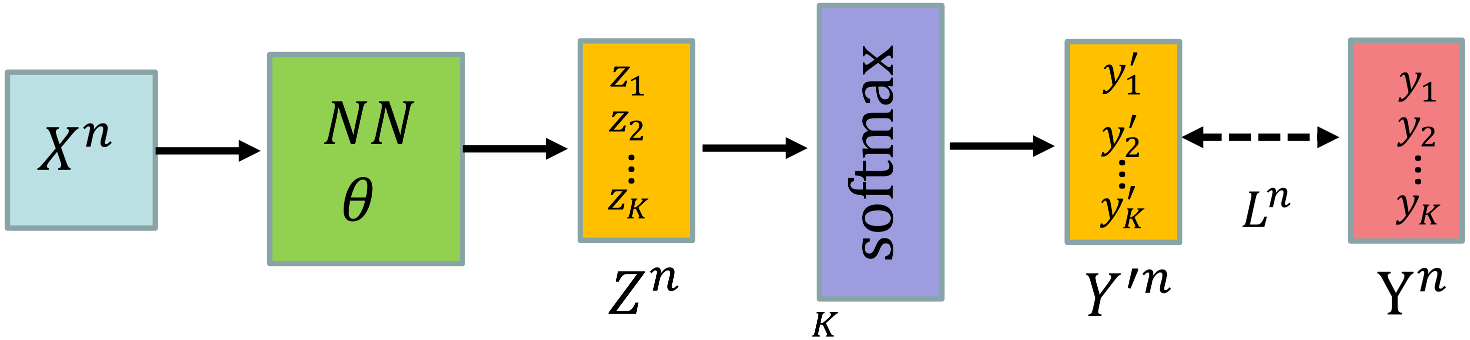


Cross-entropy: $L(Y', Y) = -\sum_{j=1}^{K} y_j \log y'_j$ $\qquad y'_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}}$

$$\frac{\partial y'_j}{\partial z_i} = -y'_i y'_j \qquad , \qquad i \neq j$$

$$\begin{aligned} \frac{\partial y'_j}{\partial z_i} &= y'_i - y'_i y'_j \\ &= y'_i - y'_i y'_i \qquad , \qquad i = j \end{aligned}$$

反向传播法
Backpropagation



Cross-entropy: $$L(Y', Y) = -\sum_{j=1}^{K} y_j \log y'_j \qquad y'_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}}$$

$$\frac{\partial y'_j}{\partial z_i} = -y'_i y'_j \quad , \quad i \neq j$$

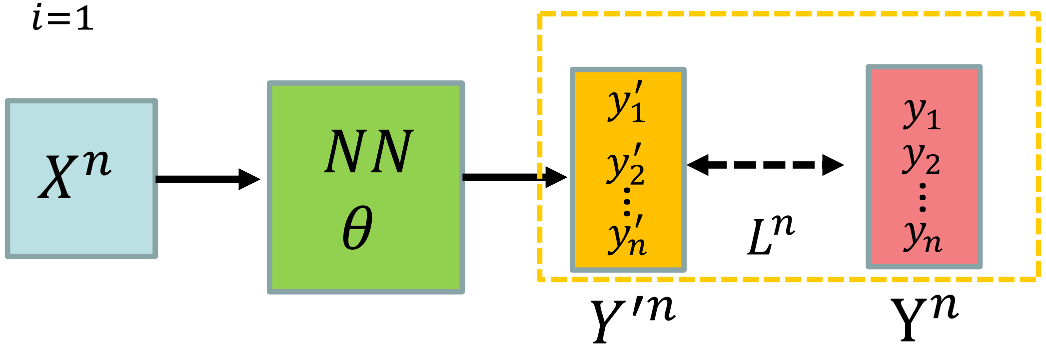
$$\frac{\partial y'_j}{\partial z_i} = y'_i - y'_i y'_j$$
$$= y'_i - y'_i y'_j \quad , \quad i = j$$

$$\frac{\partial L}{\partial z_i} = -\sum_{j=1}^{K} \frac{\partial y_j \log y'_j}{\partial y'_j} \frac{\partial y'_j}{\partial z_i}$$
$$= -\frac{y_i}{y'_i}(y'_i - y'_i y'_i) - \sum_{j=1, j\neq i}^{K} \frac{y_j}{y'_j}(-y'_i y'_j)$$
$$= -y_i + y_i y'_i + \sum_{j=1, j\neq i}^{N} y_j\, y'_i$$
$$= -y_i + y'_i \quad = y'_i - y_i$$

反向传播算法
Backpropagation

Cost function: $\frac{1}{N}\sum_{i=1}^{N} L(y^{(i)}, y'^{(i)})$

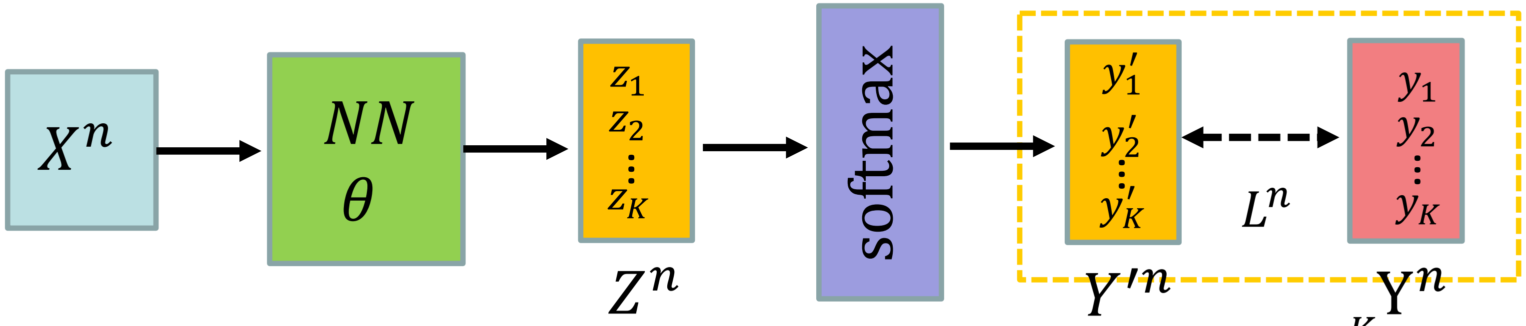
Regression:



$$\frac{\partial L}{\partial y'_i} == y'_i - y_i$$

Mean Square Error (MSE): $L(y', y) = (y' - y)^2$

Classification:



$$\frac{\partial L}{\partial z_i} == y'_i - y_i$$

Cross-entropy: $L(Y', Y) = -\sum_{j=1}^{K} y_j \log y'_j$

反向传播法
Backpropagation

- Backpropagation: an efficient way to compute $\partial L / \partial w$ in neural network

TensorFlow
torch
theano
Caffe
Microsoft CNTK
Chainer
Deep Learning library produced by Amazon
DSSTNE
mxnet

反向传播算法评价
Remarks on the Backprop

- Very powerful - can learn any function with enough hidden units, we can generate any function.

- BP is only guaranteed to converge toward some local minimum and not necessarily to the global minimum error.
  - momentum term
  - stochastic gradient descent
  - Train multiple networks using the same data

多层神经网络的表征能力
Representational Power of ANN

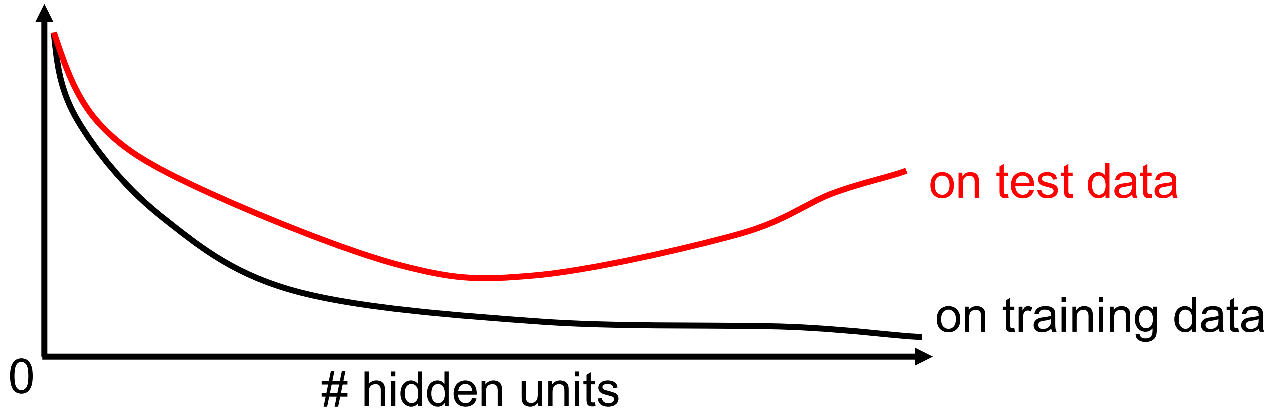
- **Boolean functions**
  - Any boolean function can be represented by a two-layer network with sufficient hidden units.
- **Continuous functions**
  - Any bounded continuous function can be approximated with arbitrarily small error by a two-layer network.
- **Arbitrary function**
  - Any function can be approximated to arbitrary accuracy by a three-layer network.

神经网络的建立
Establishment of Neural Networks

- Decisions must be taken on the following:
  - The number of units to use.(hyperparameters)
    - How many hidden units in the layer?
    - Too few ==> can’t learn
    - Too many ==> poor generalization
  - The type of units required.(hyperparameters)
  - Connection between the units.

选择隐藏层节点个数
Determining the Number of Hidden Units

- Too few hidden units prevents the network from adequately fitting the data.
- Too many hidden units can result in over-fitting.



- Use internal cross-validation to empirically determine an optimal number of hidden units.

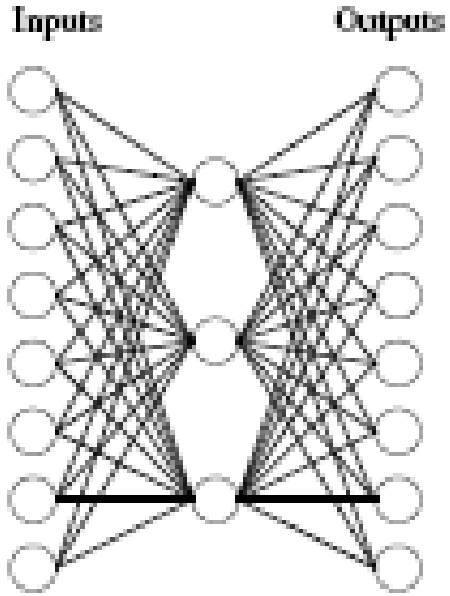
隐藏单元的表征
Hidden Unit Representations

- Trained hidden units can be seen as newly constructed features that make the target concept linearly separable in the transformed space.
- On many real domains, hidden units can be interpreted as representing meaningful features such as vowel detectors or edge detectors, etc..
- However, the hidden layer can also become a distributed representation of the input in which each individual unit is not easily interpretable as a meaningful feature.

隐藏单元的表征举例
Example - Hidden Unit Representations

A target function:

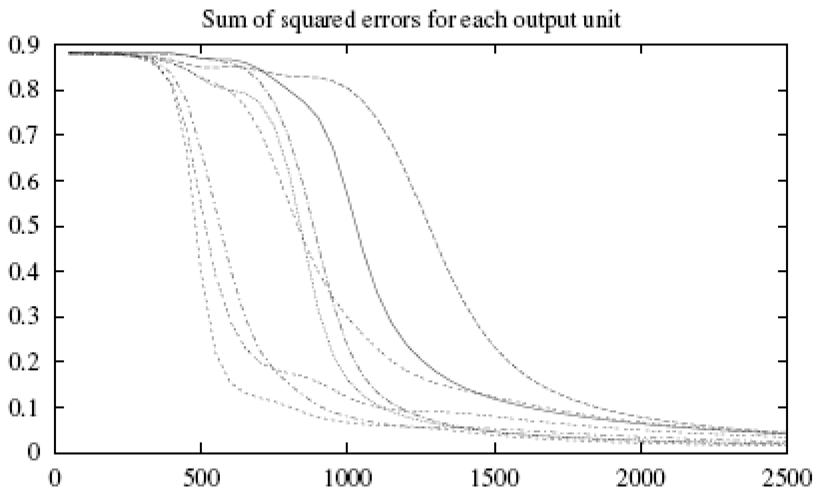
Input		Output
10000000	→	10000000
01000000	→	01000000
00100000	→	00100000
00010000	→	00010000
00001000	→	00001000
00000100	→	00000100
00000010	→	00000010
00000001	→	00000001

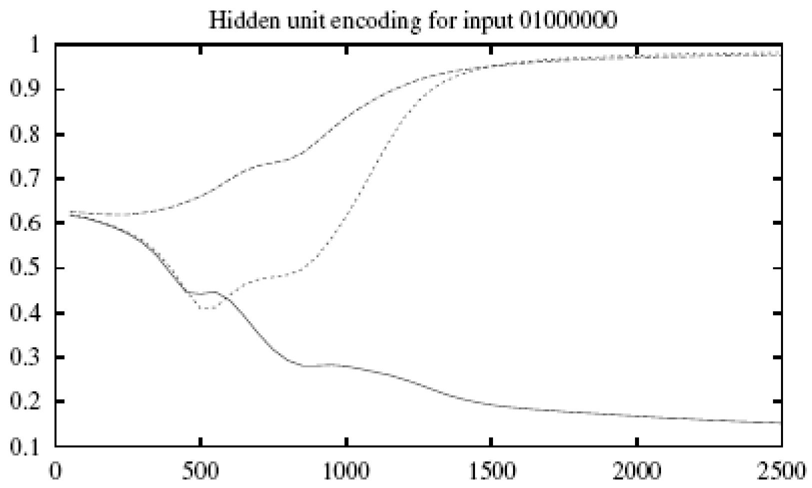


Can this be learned by a 8*3*8 neural network?

http://www.cs.cmu.edu/~tom/mlbook

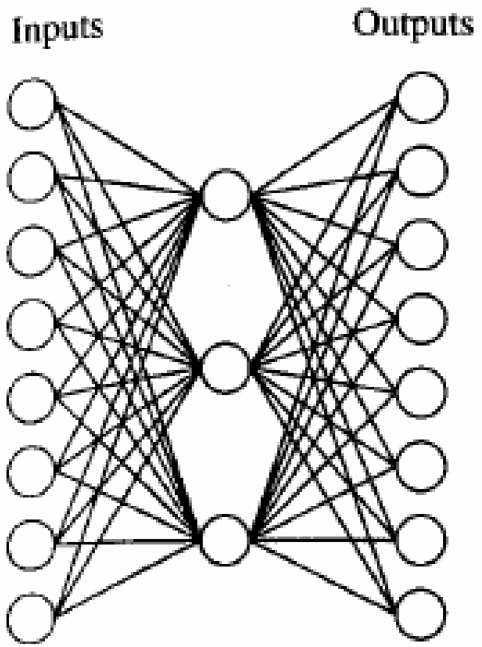
隐藏单元的表征举例
Example - Hidden Unit Representations





http://www.cs.cmu.edu/~tom/mlbook

隐藏单元的表征举例
Example - Hidden Unit Representations

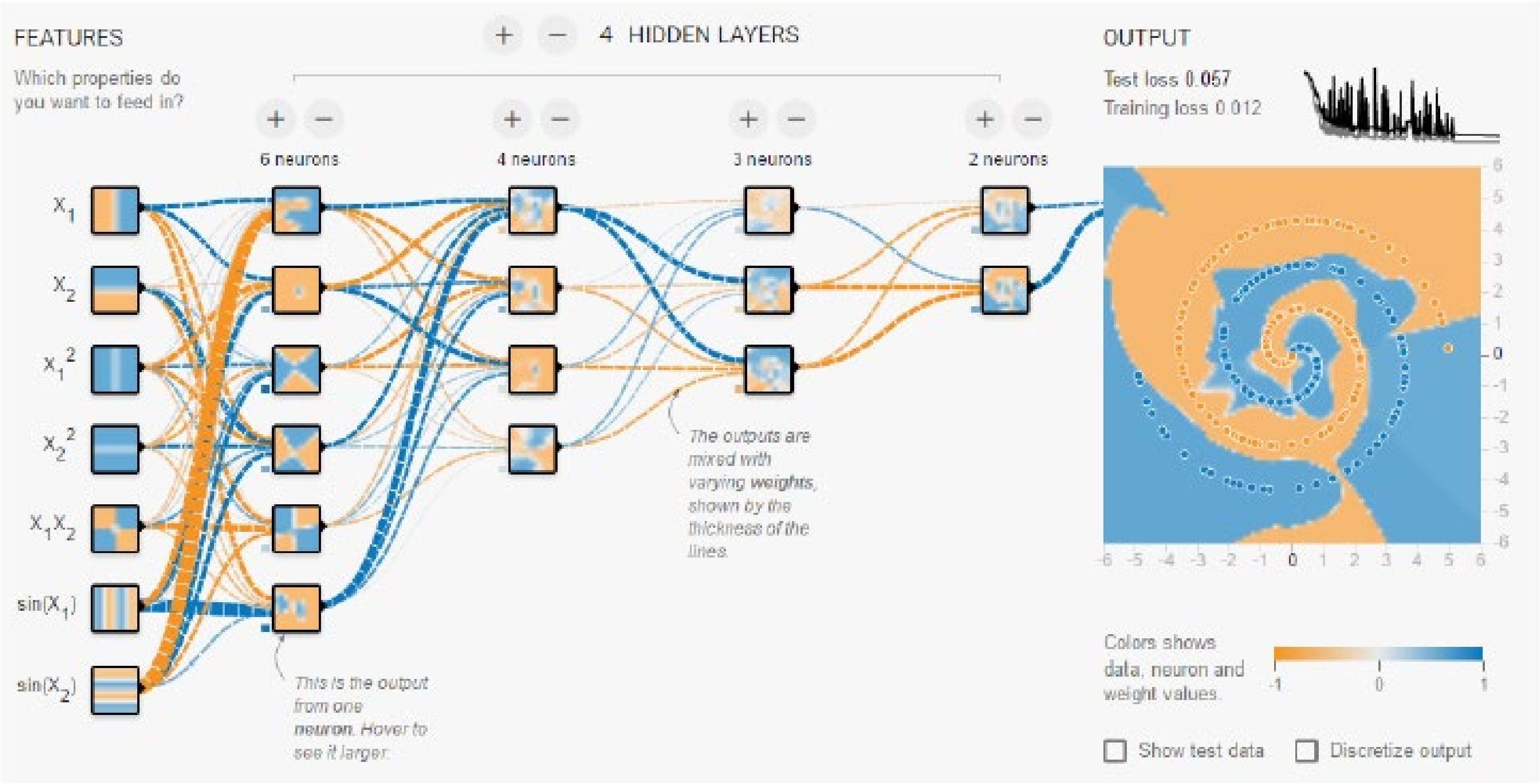


Input		Hidden Values				Output
10000000	→	.89	.04	.08	→	10000000
01000000	→	.15	.99	.99	→	01000000
00100000	→	.01	.97	.27	→	00100000
00010000	→	.99	.97	.71	→	00010000
00001000	→	.03	.05	.02	→	00001000
00000100	→	.01	.11	.88	→	00000100
00000010	→	.80	.01	.98	→	00000010
00000001	→	.60	.94	.01	→	00000001

Learned hidden layer representation after 5000 times training

http://www.cs.cmu.edu/~tom/mlbook

A demo from Google



http://playground.tensorflow.org/

小结

- Neural network is a computational model that simulate some properties of the human brain.
- The connections and nature of units determine the behavior of a neural network.
- Perceptrons are feed-forward networks that can only represent linearly separable functions.
- Given enough units, any function can be represented by Multi-layer feed-forward networks.
- Backpropagation learning works on multi-layer feed-forward networks.
- Neural Networks are widely used in developing artificial learning systems.