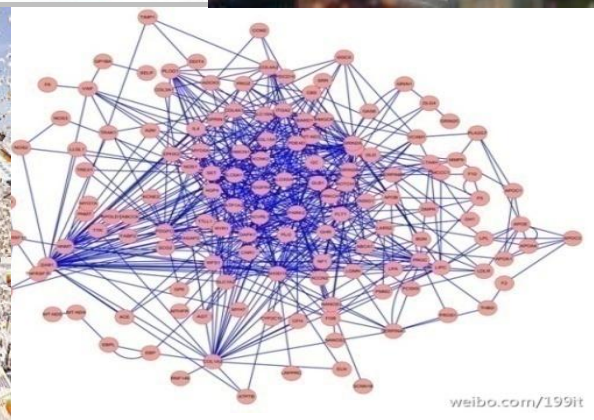


Chapter 7 Artificial Neural Network

2025 Autumn

Lei Sun

01 Perceptron

02 BP Algorithm

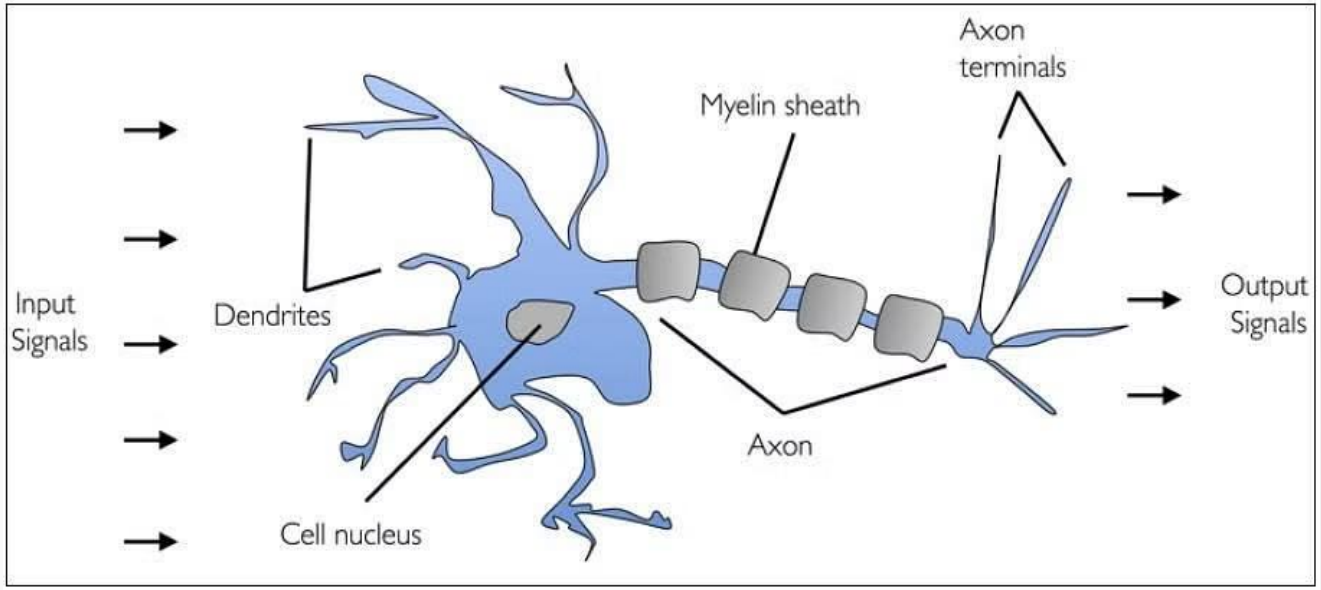
03 Active function and lost function

04 Issues

05 Why—hidden layers

Neural Networks

- ✓ Inspired by brain structure.
- ✓ The neuron(神经元) is the basic building block of a biological brain.
- ✓ Neuron is a device that takes a combination of input signals and produces an output signal.
- ✓ In a typical brain there are millions of neurons connected together: weights



Neural Networks

- The status of every neuron is S_i(i=1,2,...,n),whose value is only 0 or 1, representing depressing and excitement. The status is described by MP function:

$$S_i = f(\sum_j \omega_{ij} S_j + \theta_i)$$

- Hebb rule: according to conditioned response.

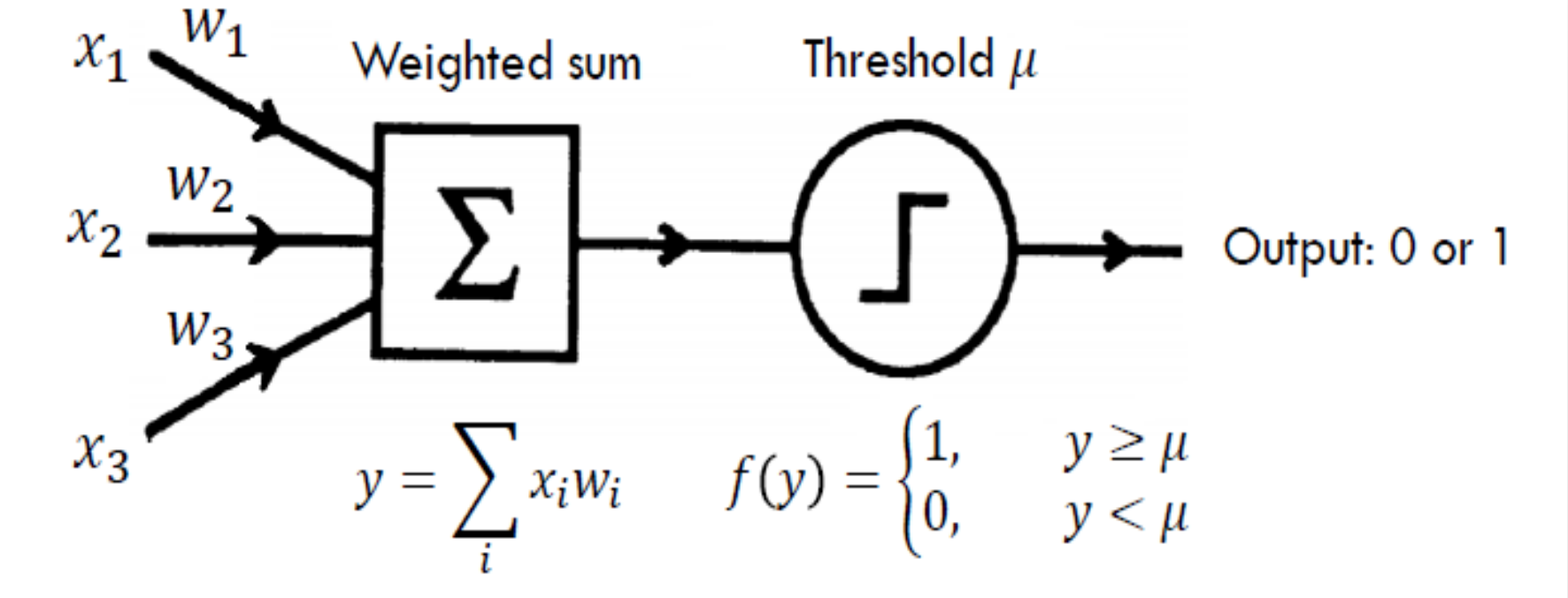
$$\Delta\omega_{ij} = \alpha S_i S_j \qquad \alpha > 0$$

Suppose $\alpha = 1$, when s_i=s_j=1, $\Delta\omega_{ij} = 1$. The connection is strong.

When s_i, s_j=0, $\Delta\omega_{ij} = 0$. The connection is weak.

Artificial neuron（人工神经元）

Inputs (connection) → linear aggregation → transfer → output



Usually inputs are "normalized" so that they takes values in [0,1].

Artificial neuron（人工神经元）

Why do we need Activation Functions?

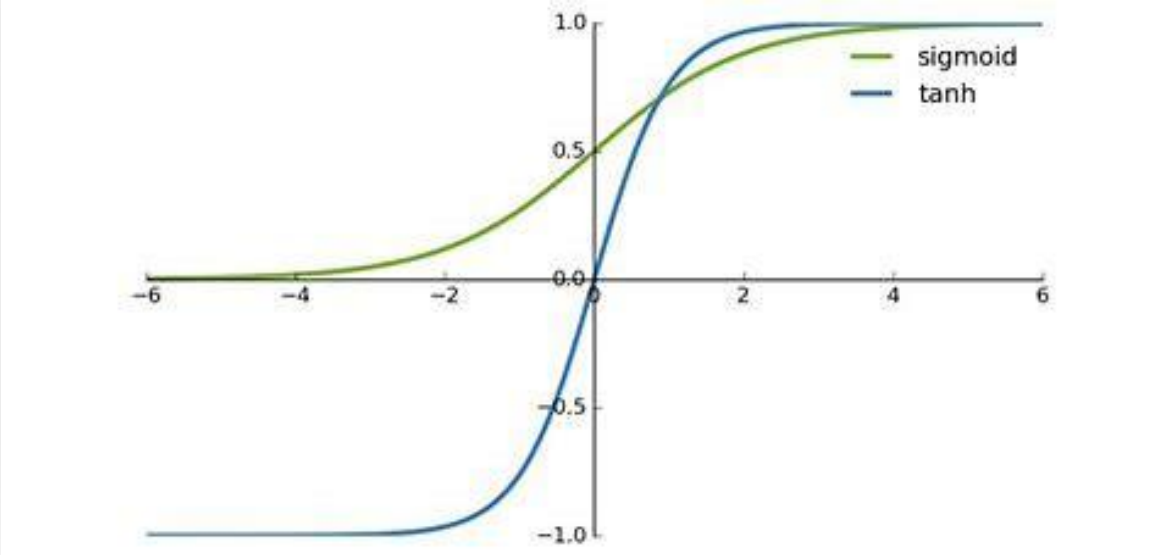
Without activation function, weight and bias would only have a linear transformation, or neural network is just a linear model, a linear equation is polynomial of one degree（一次多项式） only which is simple to solve but limited in terms of ability to **solve complex problems**.

In addition to that, Activation functions are differentiable due to which they can easily implement back propagations, optimized strategy while performing backpropagations to measure **gradient(梯度)** loss functions in the neural networks.

Artificial neuron（人工神经元）

Sigmoid Function

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

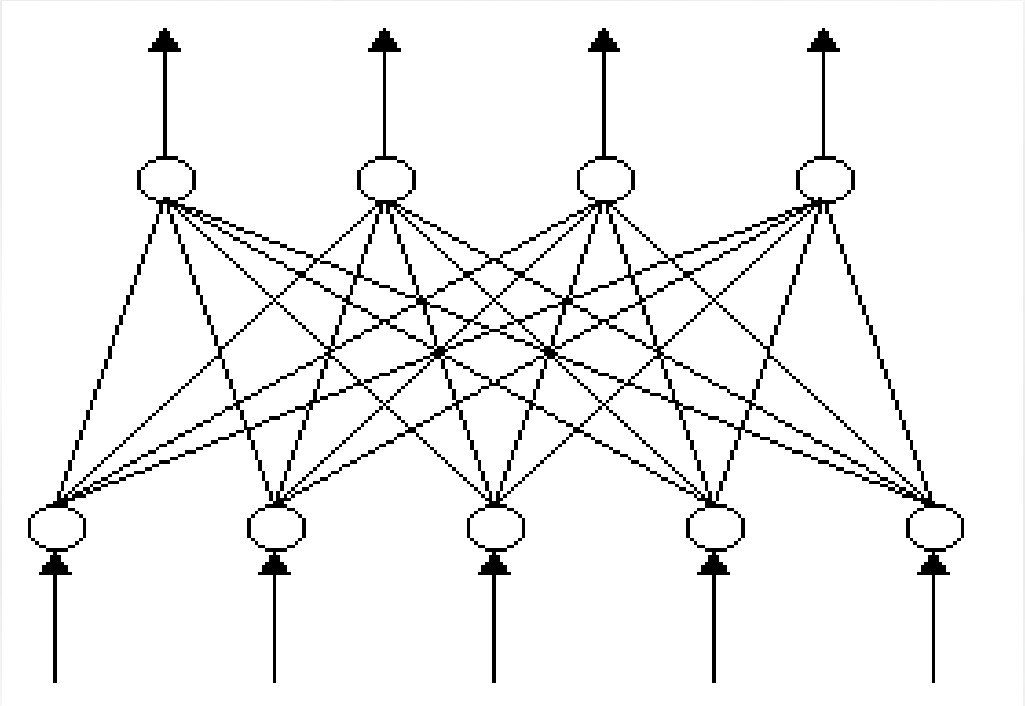


Hyperbolic Tangent Function(Tanh)

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

ranges in between -1 to 1

Single-Layer Perceptron （感知器）



Single-Layer Perceptron （感知器）

X_1	X_2	Expected d
0	0	0
0	1	1
1	0	1
1	1	1

Boolean add
逻辑加

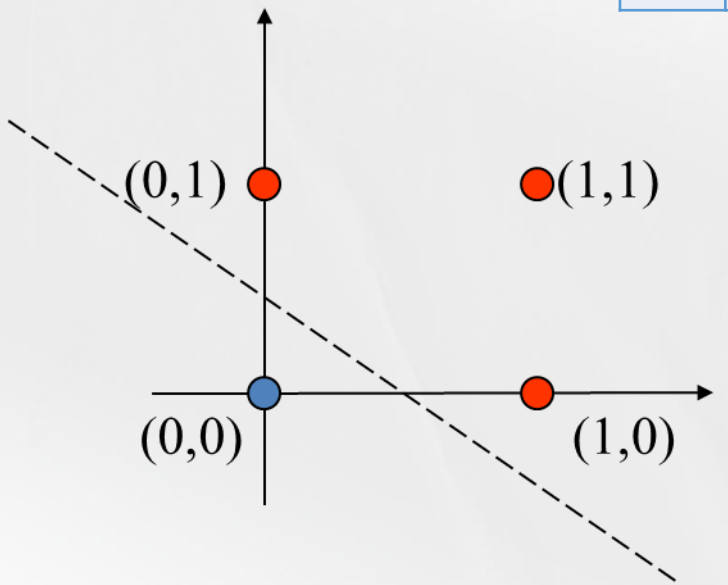
formula:

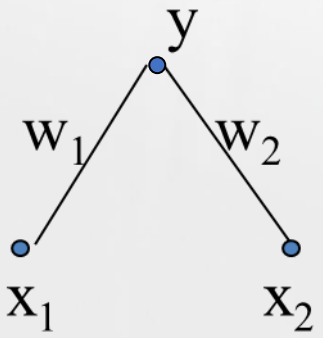
$$\begin{bmatrix} W1 \\ W2 \end{bmatrix}^{(k+1)} = \begin{bmatrix} W1 \\ W2 \end{bmatrix}^{(k)} + c\,(d-y) \begin{bmatrix} x1 \\ x2 \end{bmatrix}$$

initial: $\begin{bmatrix} W1 \\ W2 \end{bmatrix} \quad \begin{bmatrix} 0 \\ 0 \end{bmatrix} \quad c = 1$

d: expected output

y: computed output





Linearly separable: A set of two-class sample in the space is linearly separable if a hyperplane (straight line) can be drawn to separate all of the tuples of two classes.

computing process:

- K = 1: y = f (0 + 0) = 0

$$\begin{bmatrix} W1 \\ \\ W2 \end{bmatrix}^{(1)} = \begin{bmatrix} W1 \\ \\ W2 \end{bmatrix}^{(0)} + (0-0) \qquad \begin{bmatrix} 0 \\ \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ \\ 0 \end{bmatrix}$$

- K = 2, y = f (0 + 0) = 0

$$\begin{bmatrix} W1 \\ \\ W2 \end{bmatrix}^{(2)} = \begin{bmatrix} W1 \\ \\ W2 \end{bmatrix}^{(1)} + (1-0) \qquad \begin{bmatrix} 0 \\ \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ \\ 1 \end{bmatrix}$$

- K = 3, y = f (0 + 0) = 0

$$\begin{bmatrix} W1 \\ \\ W2 \end{bmatrix}^{(3)} = \begin{bmatrix} W1 \\ \\ W2 \end{bmatrix}^{(2)} + (1-0) \qquad \begin{bmatrix} 1 \\ \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ \\ 1 \end{bmatrix} + \begin{bmatrix} 1 \\ \\ 0 \end{bmatrix} = \begin{bmatrix} 1 \\ \\ 1 \end{bmatrix}$$

- K = 4, y = f (1 + 1) = f (2) = 1

$$\begin{bmatrix} W1 \\ \\ W2 \end{bmatrix}^{(4)} = \begin{bmatrix} W1 \\ \\ W2 \end{bmatrix}^{(3)} + (1-1) \qquad \begin{bmatrix} 1 \\ \\ 1 \end{bmatrix} = \begin{bmatrix} 1 \\ \\ 1 \end{bmatrix} + \begin{bmatrix} 0 \\ \\ 0 \end{bmatrix} = \begin{bmatrix} 1 \\ \\ 1 \end{bmatrix}$$

Cycle again, all values of $(d - y)$ are zero, that is, weights $(W_1 = 1,\ W_2 = 1)$ satisfy all the samples.

Single-Layer Perceptron （感知器）

X_1	X_2	Expected d
0	0	0
0	1	1
1	0	1
1	1	0

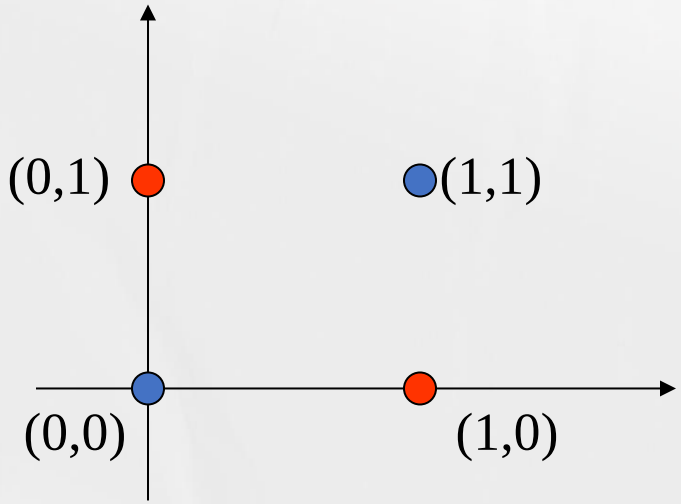
For XOR problem

Nonlinearity samples: can't find a hyperplane to separate the two classes samples.

K＝1， 2， 3， **when** K＝**4:**

$$y = f(1+1) = f(2) = 1$$

$$\begin{bmatrix} W1 \\ W2 \end{bmatrix}^{(4)} = \begin{bmatrix} W1 \\ W2 \end{bmatrix}^{(3)} + (0-1) \begin{bmatrix} 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \end{bmatrix} + \begin{bmatrix} -1 \\ -1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$



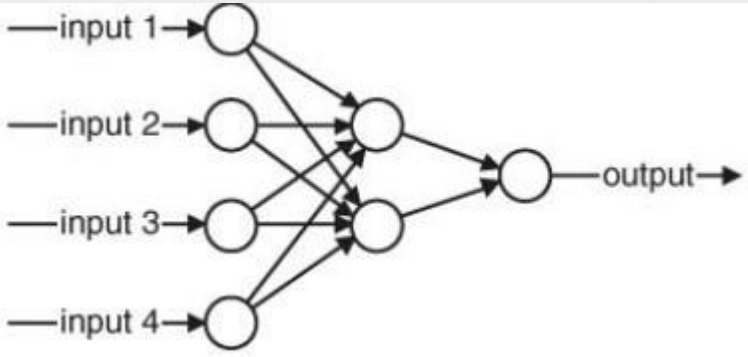
Corrected weights come back to initial value. If continually computing, we will see endless loop, and weights are never convergence.

Single-layer perceptron is invalid to nonlinearity samples.

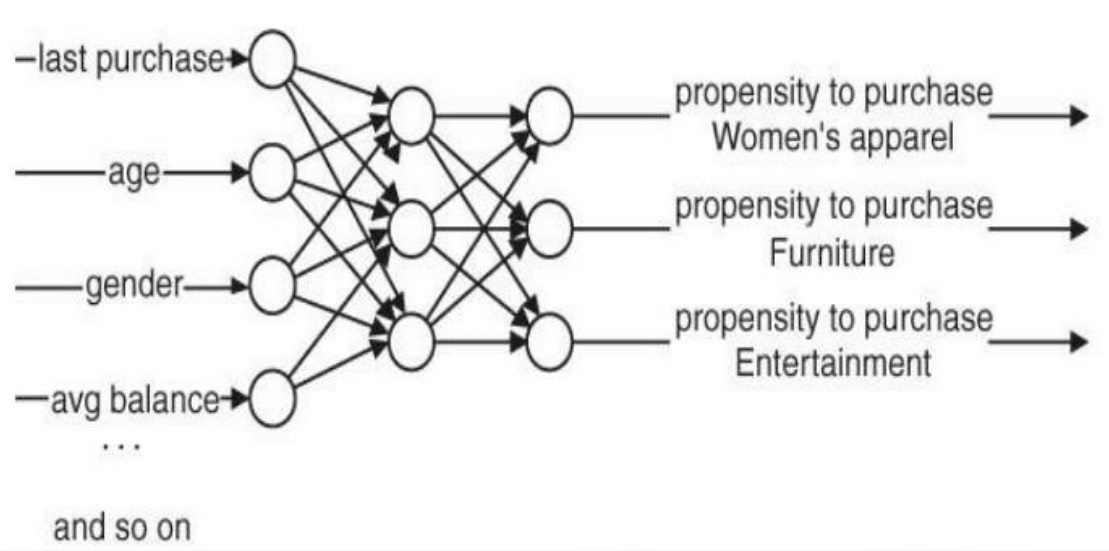
What is an Artificial Neural Network

- ✓ A network of many simple unites(neurons, nodes)
  - ---The neurons are connected by *connections*.
    - Each neuron is both a storage unit of information and a processing unit of information.
  - --- Each connection has an associated numeric **weight.**
  - ---Messages are passed along the connections and can change based on the weights and the values in the nodes.
  - ---ANN can be represented as a directed graph.
  - --- **Modifying the weights** of connections in order to perform a certain task.

What is an Artificial Neural Network



This network has a middle layer called the hidden layer, which makes the network more powerful by enabling it to recognize more patterns



A neural network can produce multiple output values.

Back Propagation Algorithm

Major Steps for BP

① **Constructing a network**

- input data representation
- selection of number of layers and number of nodes in each layer.

② **Training the network using training data**

Forward pass Backward pass

③ **Pruning the network**

The ultimate objective of training

obtain a set of weights that makes almost all the tuples in the training data classified correctly

Back Propagation Algorithm

Working of Backpropagation Algorithm

The BP algorithm works by two different passes:

Forward pass Backward pass

✓ **Forward pass:**

- initially the input(raw data) is fed into the input layer.
- The inputs and their corresponding weights are passed to the hidden layer. The hidden layer performs the computation on the data it receives.
- To the weighted sum of inputs, the activation function is applied in the hidden layer to each of its neurons.
- And finally, the weighted outputs from the last hidden layer are fed into the output to compute the final prediction, same as the computation of hidden layer.

Back Propagation Algorithm

Working of Backpropagation Algorithm

- ✓ **Backward pass:**
  - the error is transmitted back to the network which helps to improve its performance by learning and adjusting the internal weights.
  - most commonly used methods is mean squared error which calculates the difference between the predicted output and class label.
  - Once we have done the calculation at the output layer, we then propagate the error backward through the network, layer by layer.

Back Propagation Algorithm

The key calculation during the backward pass is determining the gradients for each weight and bias in the network.

This gradient is responsible for telling us how much each weight/bias should be adjusted to minimize the error in the next forward pass. The chain rule(链式法则) is used iteratively to calculate this gradient efficiently.

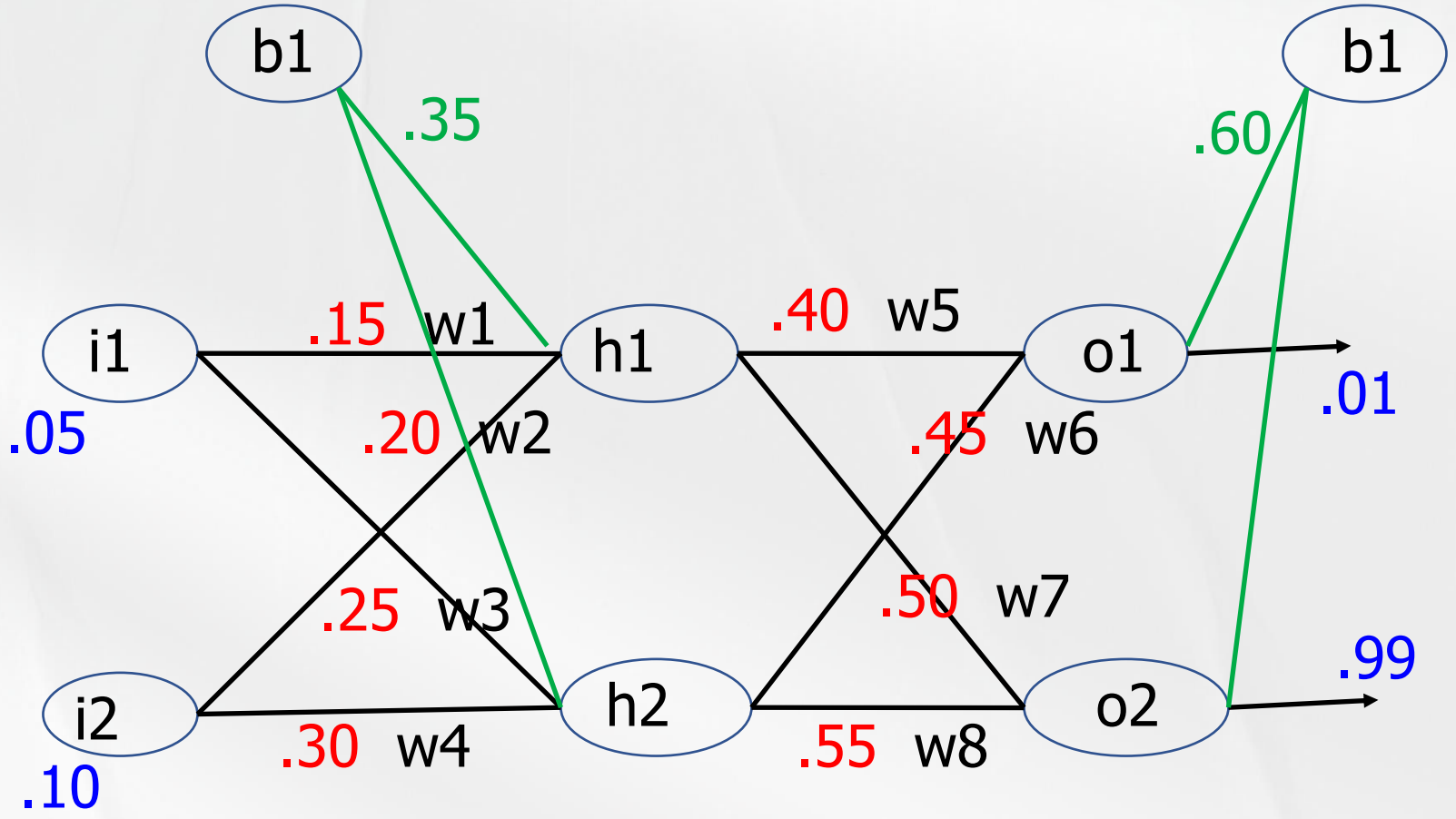
$$\Delta T_{li} = -\eta \frac{\partial E}{\partial T_{li}}$$

$$\Delta w_{ij} = -\eta \frac{\partial E}{\partial w_{ij}}$$

η : learning rate, defines the step; typically in the range (0,1)

Back Propagation Algorithm

example



Step1--- forward computation

- input layer→ hidden layer

$$\text{net}_{h1} = w_1 * i_1 + w_2 * i_2 + b_1 \qquad \text{net}_{h1} = 0.15 * 0.05 + 0.2 * 0.1 + 0.35 = 0.3775$$

$$\text{out}_{h1} = \frac{1}{1 + e^{-\text{net}_{h1}}} = \frac{1}{1 + e^{-0.3775}} = 0.5933 \qquad \text{out}_{h2} = 0.5969$$

- hidden layer→ output layer

$$\text{net}_{o1} = w_5 * \text{out}_{h1} + w_6 * \text{out}_{h2} + b_2$$
$$= 0.4 * 0.5933 + 0.45 * 0.5969 + 0.6 = 1.1059$$

$$\text{out}_{o1} = \frac{1}{1 + e^{-\text{net}_{o1}}} = \frac{1}{1 + e^{-1.1059}} = 0.7514 \qquad \text{out}_{o2} = 0.7729$$

Step2--- backward propagation

$$E_{total} = \frac{1}{2}(target - output)^2$$

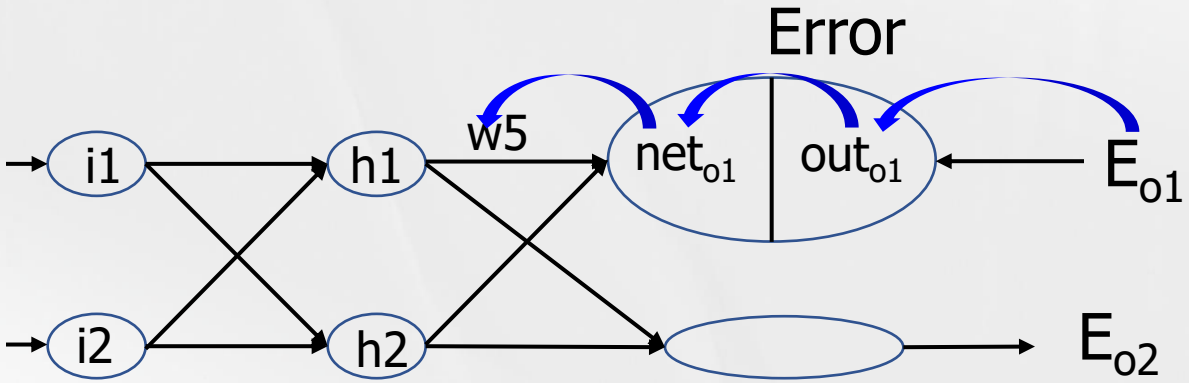
- total error (square error)

$$E_{o1} = \sum \frac{1}{2}(target_{o1} - out_{o1})^2 = \frac{1}{2}(0.01 - 0.7514)^2 = 0.2748$$

$$E_{total} = E_{o1} + E_{o2} = 0.2984$$

- adjustment of weights

① output layer→ hidden layer



$$\frac{\partial E_{total}}{\partial w_5} = \frac{\partial net_{o1}}{\partial w_5} * \frac{\partial out_{o1}}{\partial net_{o1}} * \frac{\partial E_{total}}{\partial out_{o1}}$$

$$E_{o1} = \frac{1}{2}(target_{o1} - out_{o1})^2 \qquad E_{total} = E_{o1} + E_{o2}$$

$$E_{total} = \frac{1}{2}(target_{o1} - out_{o1})^2 + \frac{1}{2}(target_{o2} - out_{o2})^2$$

$$\frac{\partial E_{total}}{\partial out_{o1}} = -(target_{o1} - out_{o1}) = -(0.01 - 0.7514) = 0.7414$$

$$\frac{\partial out_{o1}}{\partial net_{o1}} = out_{o1}(1 - out_{o1}) = 0.7514(1 - 0.7514) = 0.1868$$

$$net_{o1} = w_5 * out_{h1} + w_6 * out_{h2} + b_2$$

$$\frac{\partial net_{o1}}{\partial w_5} = 1 * out_{h1} = 0.5933$$

$$\frac{\partial E_{total}}{\partial w_5} = \frac{\partial net_{o1}}{\partial w_5} * \frac{\partial out_{o1}}{\partial net_{o1}} * \frac{\partial E_{total}}{\partial out_{o1}} = 0.0822$$

$$\frac{\partial E_{total}}{\partial w_5} = -(target_{o1} - out_{o1})\underbrace{\text{out}_{o1}(1 - out_{o1})}_{f'(x)}\text{out}_{h1}$$

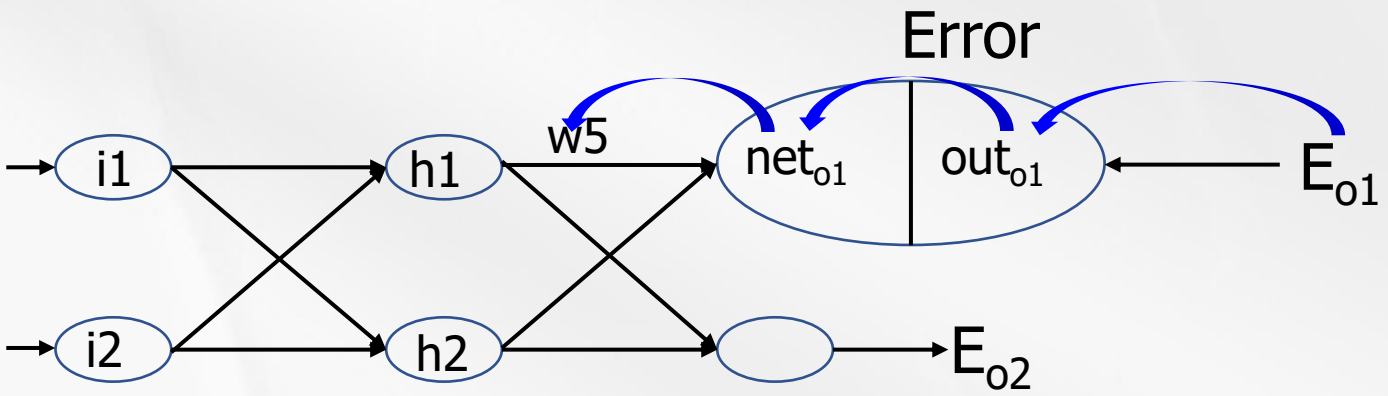
$$\delta_{o1}$$

隐层输出

$$\frac{\partial E_{total}}{\partial w_5} = \delta_{o1}\text{out}_{h1}$$

$$\text{w}_5^+ = \text{w}_5 - \eta\frac{\partial E_{total}}{\partial w_5}$$

$$= 0.4 - 0.5 * 0.0822 = 0.3589$$



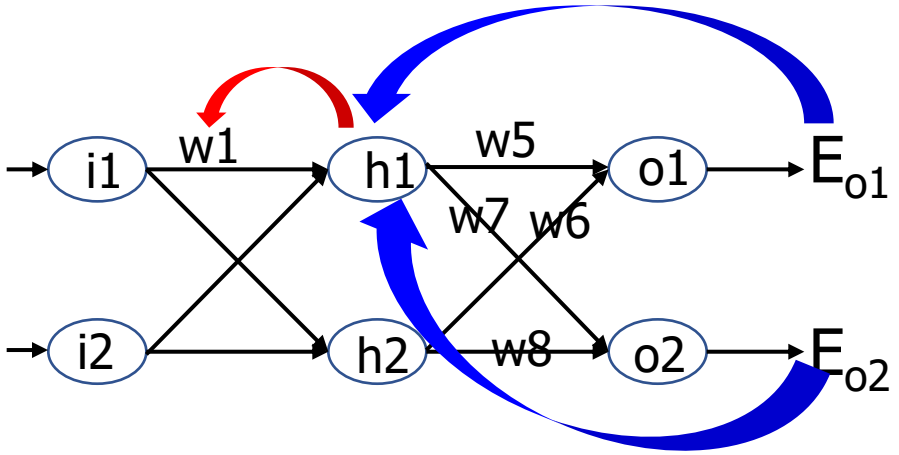
② Hidden layer→ input layer

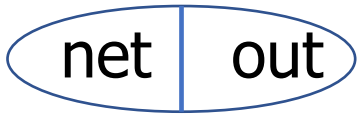
$$\frac{\partial E_{total}}{\partial out_{h1}} = \frac{\partial E_{o1}}{\partial out_{h1}} + \frac{\partial E_{o2}}{\partial out_{h1}}$$

$$\frac{\partial E_{total}}{\partial w_1} = \frac{\partial E_{total}}{\partial out_{h1}} * \frac{\partial out_{h1}}{\partial net_{h1}} * \frac{\partial net_{h1}}{\partial w_1}$$

$f'(net_{h1})$

i1





h1

$E_{total}=E_{o1}+E_{o2}$

$$\frac{\partial E_{o1}}{\partial out_{h1}} = \frac{\partial E_{o1}}{\partial net_{o1}} * \frac{\partial net_{o1}}{\partial out_{h1}}$$

$$\frac{\partial E_{o1}}{\partial net_{o1}} = \frac{\partial E_{o1}}{\partial out_{o1}} * \frac{\partial out_{o1}}{\partial net_{o1}} = 0.7414 * 0.1868 = 0.1385$$

e $f'(net_{o1})$

$$net_{o1} = w_5 * out_{h1} + w_6 * out_{h2} + b_2$$

$$\frac{\partial net_{o1}}{\partial out_{h1}} = w_5 = 0.4$$

$$\frac{\partial E_{o2}}{\partial out_{h1}} = -0.019$$

$$\frac{\partial E_{o1}}{\partial out_{h1}} = 0.4 * 0.1385 = 0.0554$$

$$\frac{\partial E_{o2}}{\partial out_{h1}} = -0.019$$

$$\frac{\partial E_{total}}{\partial w_1} = \frac{\partial E_{total}}{\partial out_{h1}} * \frac{\partial out_{h1}}{\partial net_{h1}} * \frac{\partial net_{h1}}{\partial w_1}$$

$$\frac{\partial E_{total}}{\partial out_{h1}} = \frac{\partial E_{o1}}{\partial out_{h1}} + \frac{\partial E_{o2}}{\partial out_{h1}} = 0.0554 - 0.019 = 0.0364$$

$$\frac{\partial out_{h1}}{\partial net_{h1}} = \mathrm{out}_{h1}(1 - out_{h1}) = 00.5933(1 - 0.5933) = 0.2413$$

$$\mathrm{net}_{h1} = \mathrm{w}_1 * i_1 + w_2 * i_2 + \mathrm{b}_1 \qquad \frac{\partial net_{h1}}{\partial w_1} = \mathrm{i}_1 = 0.05$$

$$\frac{\partial E_{total}}{\partial w_1} = \frac{\partial E_{total}}{\partial out_{h1}} * \frac{\partial out_{h1}}{\partial net_{h1}} * \frac{\partial net_{h1}}{\partial w_1} = 0.0364 * 0.2413 * 0.05 = 0.0004$$

$$\mathrm{w}_1^+ = \mathrm{w}_1 - \eta \frac{\partial E_{total}}{\partial w_1} = 0.15 - 0.5 * 0.0004 = 0.1498$$

$$\Delta w = \frac{\partial \mathrm{E}_{\mathrm{total}}}{\partial w_1} = [\underbrace{\mathrm{e}_1 \mathrm{f}'(\mathrm{net}_{\mathrm{o1}})}_{\delta_{\mathrm{o1}}} \mathrm{w}_5 + \mathrm{e}_2 \mathrm{f}'(\mathrm{net}_{\mathrm{o2}}) \mathrm{w}_7] f'(net_{h1}) \mathrm{x}_1$$

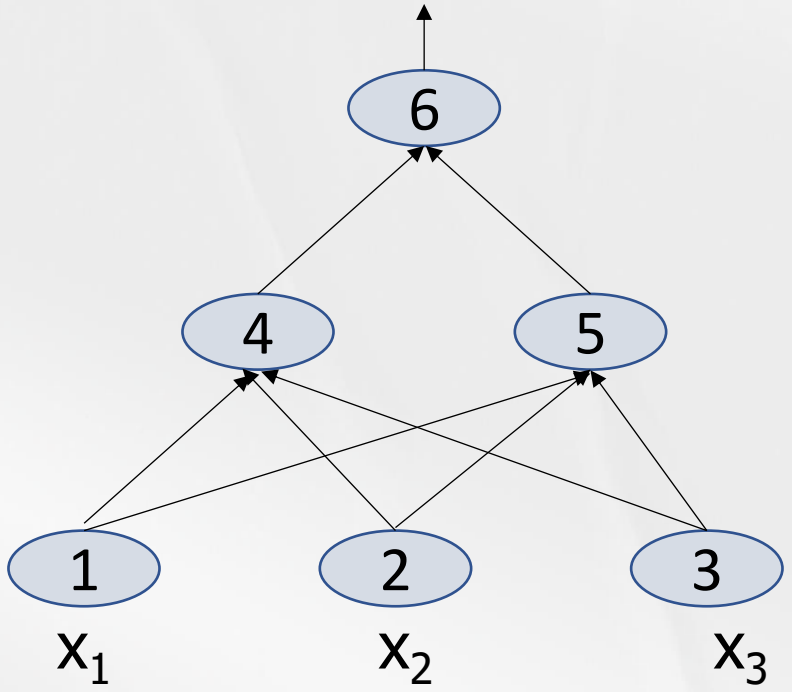
Example

inputs			weights								bias		
x_1	x_2	x_3	w_{14}	w_{15}	w_{24}	w_{25}	w_{34}	w_{35}	w_{46}	w_{56}	θ_4	θ_5	θ_6
1	0	1	0.2	-0.3	0.4	0.1	-0.5	0.2	-0.3	-0.2	-0.4	0.2	0.1

Tuple: X={1，0，1}

class: $1(t_i)$

η=0.9



Example

Neuron j	Input	output
4	0.2+0-0.5-0.4=-0.7	$1/(1+e^{-0.7})=0.332$
5	-0.3+0+0.2+0.2=0.1	$1/(1+e^{-0.1})=0.525$
6	(-0.3)(0.332)-(0.2)(0.525)+0.1=-0.105	$1/(1+e^{-0.105})=0.474$

$$\frac{\partial E_{total}}{\partial w_5} = -(target_{o1} - out_{o1})\underbrace{out_{o1}(1 - out_{o1})}_{f'(x)}out_{h1}$$

δ_{o1}

隐层输出

Neuron j	δ_l	δ_i
6	(1-0.474)(0.474)(1-0.474) =0.1311	
5		(0.1311)(-0.2) (0.525)(1-0.525)=-0.0065
4		(0.1311)(-0.3) (0.332)(1-0.332) =-0.0087

$$\Delta w = \frac{\partial E_{total}}{\partial w_1} = [\underbrace{e_1 f'(net_{o1})w_5}_{\delta_{o1}} + e_2 f'(net_{o2})w_7]f'(net_{h1})x_1$$

δ_i

$\Delta T_{li}=\eta\delta_l y_i$

$\Delta w_{ij}=\eta\delta_i x_j$

W_{46}	-0.3+(0.9)(0.1311)(0.332)=-0.261
W_{56}	-0.2+(0.9)(0.1311)(0.525)=-0.138
W_{14}	0.2+(0.9)(-0.0087)(1)=0.192
W_{15}	-0.3+(0.9)(-0.0065)(1)=-0.306
W_{24}	0.4+(0.9)(-0.0087)(0)=0.4
W_{25}	0.1+(0.9)(-0.0065)(0)=0.1
W_{34}	-0.5+(0.9)(-0.0087)(1)=-0.508
w_{35}	0.2+(0.9)(-0.0065(1))=0.194
θ_6	0.2+(0.9)(0.1311)=0.218
θ_5	0.2+(0.9)(-0.0065)=0.194
θ_4	-0.4+(0.9)(-0.0087)=0.408

Back Propagation Algorithm

Backpropagation is the essence of neural net training.

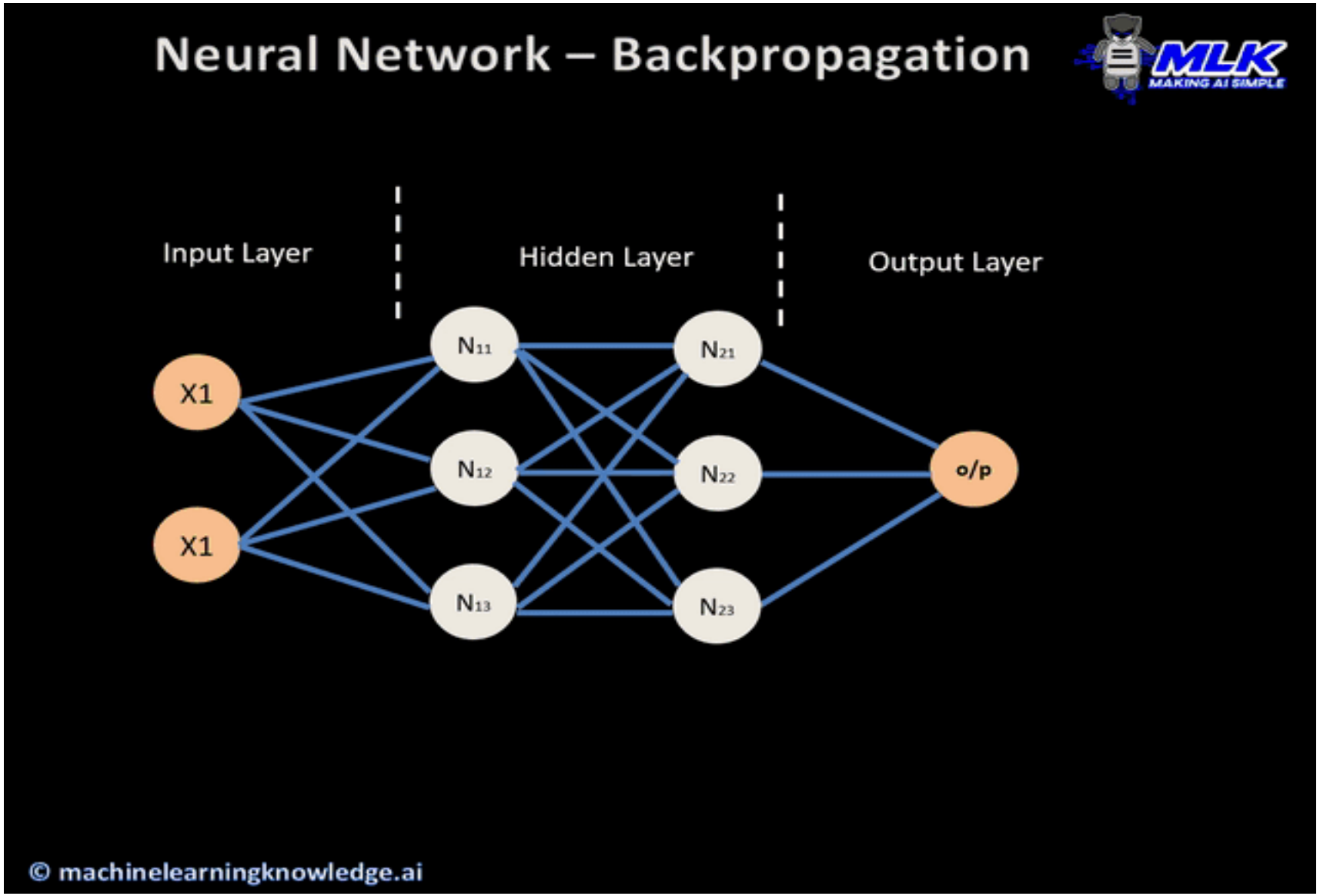
It is the practice of fine-tuning the weights of a neural net based on the error rate (i.e. loss) obtained in the previous epoch (i.e. iteration.)

Proper tuning of the weights ensures lower error rates, making the model reliable by increasing its generalization.

Back propagation Algorithm

https://machinelearningknowledge.ai/wp-content/uploads/2019/10/Backpropagation.gif

Backpropagation illustration

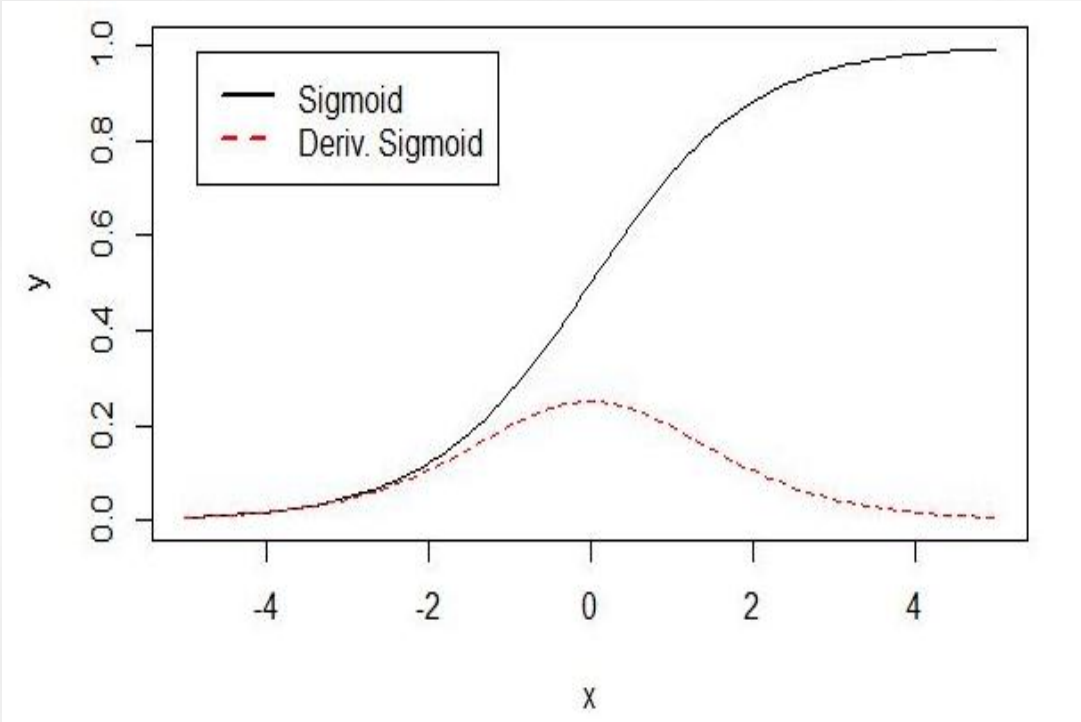


Back propagation Algorithm

Problem of Sigmoid

Vanishing gradient: the gradient can become small as it is propagated through time steps.

Slow training speed



a comparison between the sigmoid function itself and its derivative.

Back propagation Algorithm

Why is the vanishing gradient problem significant?

can significantly hinder the training and performance of ANN models.

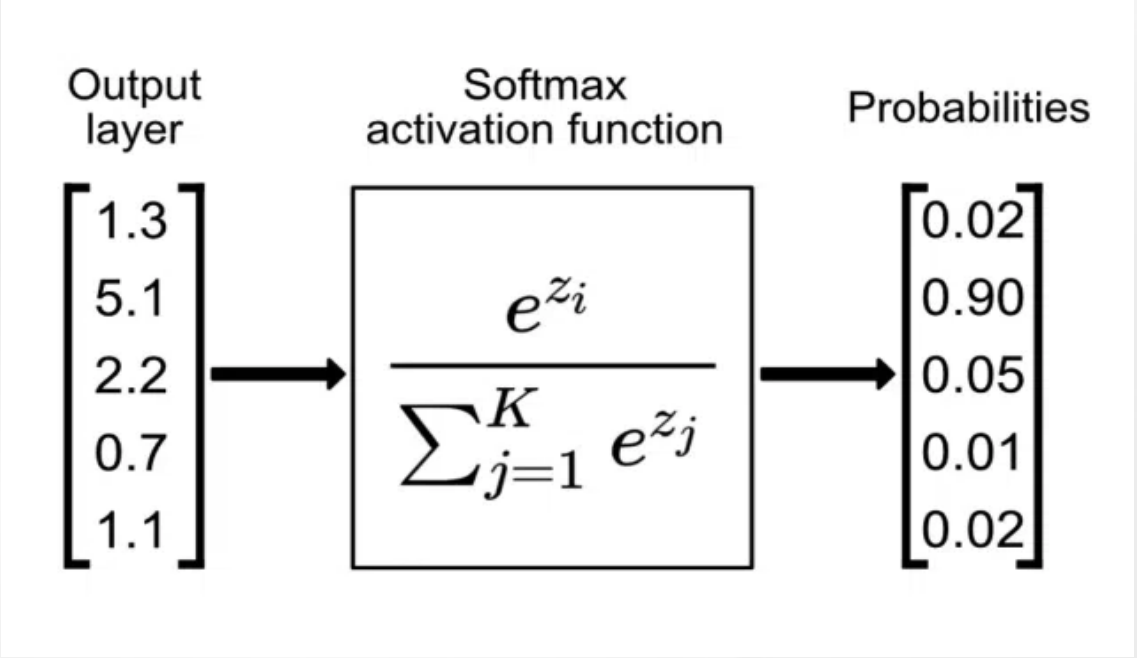
When the gradients become too small during backpropagation, the optimizer has difficulty updating the parameters, leading to slow convergence or non-convergence of the model. This can make it hard for the models to learn from the data, leading to erroneous results.

Softmax Function and Cross Entropy

Softmax

convert a vector of raw scores into a probability distribution.
It is particularly useful in the final layer of a neural network designed for classification tasks。

① **Exponentiation:** Each element z_i in the output vector is exponentiated.
② **Normalization:** Each exponentiated value is divided by the sum of all exponentiated values in the vector.
③ **Output:** The resulting values form a probability distribution, where each value is between 0 and 1, and the sum of all values is 1.



Softmax Function and Cross Entropy

Cross-Entropy Loss Function

Cross-entropy loss refers to the contrast between two random variables.

$$H(p,q) = \sum_x p(x) log \frac{1}{q(x)} = -\sum_x p(x) log q(x)$$

$$\text{Entropy } = H(p) = -\sum_i p_i * \log_2 p_i$$

$$H(p,q) = -0.35 * \log\left(\frac{1}{8}\right) - \cdots \cdots \cdots - 0.01 * \log\left(\frac{1}{8}\right)$$
$$= 3 \text{ bits}$$

$$\text{Entropy, } H(p) = -0.35 * \log 0.35 - \cdots \cdots \cdots - 0.01 * \log 0.01$$
$$= 2.23 \text{ bits}$$

where, ***p*** is the true distribution and **q** is the predicted distribution. It is exactly equal to entropy if ***p*** is equal to ***q***. when our predicted probability distribution is same as our true probability distribution.

Softmax Function and Cross Entropy

The cross entropy will always be greater than or equal to entropy

$$H(p,q) \geqslant H(p)$$

The extra information is called **relative entropy**

交叉熵可以用来计算预测分布和数据分布之间的差异。
交叉熵可以来衡量在给定的真实分布下，使用非真实分布所指定的策略消除系统的不确定性所需要付出的努力的大小。

Why are loss functions important in machine learning?

Loss functions quantify（量化）the difference between predicted and actual values in a machine learning model. They guide the optimization process by providing feedback on how well the model fits the data.损失函数通过提供关于模型对数据拟合程度的反馈，来指导（模型的）优化过程。

Softmax Function and Cross Entropy

- Binary classification

$$Loss = -[y \cdot \log(p) + (1-y)\log(1-p)]$$

average cross-entropy across all data samples:

$$\text{cross entropy} = -\frac{1}{N}\sum_{i=1}^{N}(y_i \log(p_i) - (1-y_i)\log(1-p_i))$$

- multiple classification

$$Loss = -\sum_{1}^{M} y_c \log(p_c)$$

- M: number of categories
- y: target
- p: prediction probability belonging to c

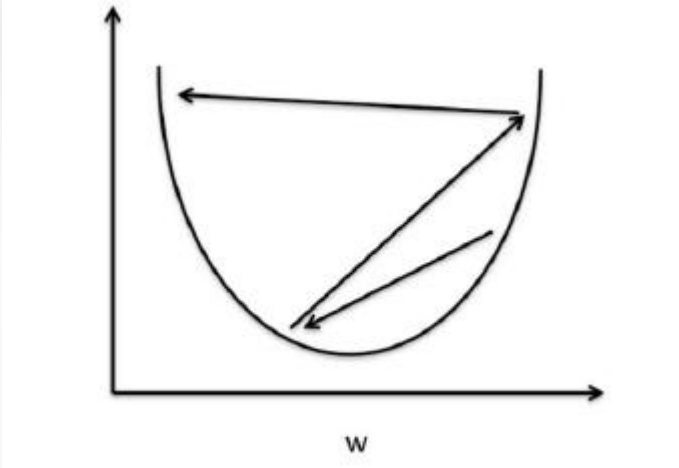
Softmax Function and Cross Entropy

Entropy: calculates the degree of randomness or disorder within a system to measure the uncertainty of an event. If an outcome is certain, the measure of entropy will be low.

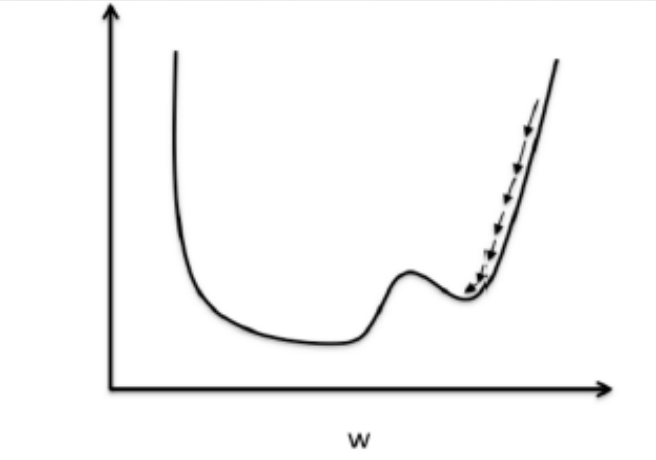
Cross-entropy: also known as logarithmic loss or log loss, is a popular loss function used in machine learning to measure the performance of a classification model. Namely, it measures the difference between the discovered probability distribution of a classification model and the predicted values.

Issues– learning rate

- ✓ If the learning rate is too large, gradient descent will overshoot the minima and diverge.
- ✓ If the learning rate is too small, the algorithm will require too many epochs to converge and can become trapped in local minima more easily.



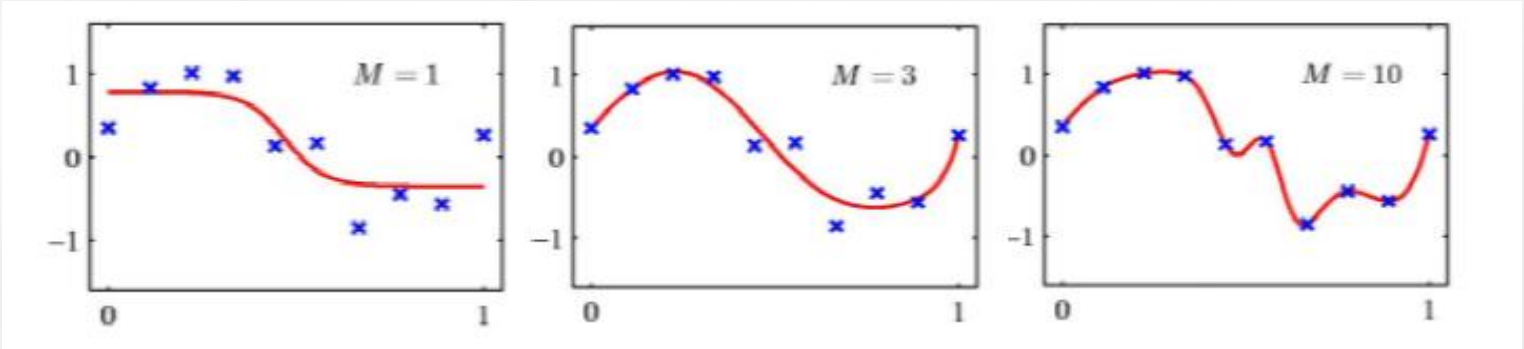
overshooting



Many iterations until convergence and trapping in local minima

Issues– Number of hidden layers

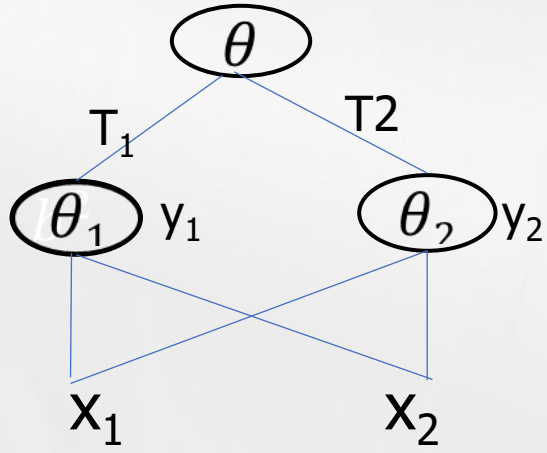
- ✓ Number of hidden units M (weights and biases) is a free parameter
  - – Adjusted to get best predictive performance



- ✓ One way to determine M is using cross-validation.
- ✓ There are other ways (regularization), to control the complexity of a neural network to avoid over-fitting.

Why—hidden layes

input	x1	x1	output
	0	1	0
	0	1	1
	1	0	1
	1	1	0



Running results:

iterations：16745；overall error：0.05

Weights and bias in the hidden layer：

w_{11}=5.24，　w_{12}=5.23，　w_{21}=6.68，w_{22}=6.64　θ_1=8.01　　θ_2=2.98

Weights and bias in the output layer：

T_1=-10，T_2=10，ϕ=4.79

Why—hidden layes

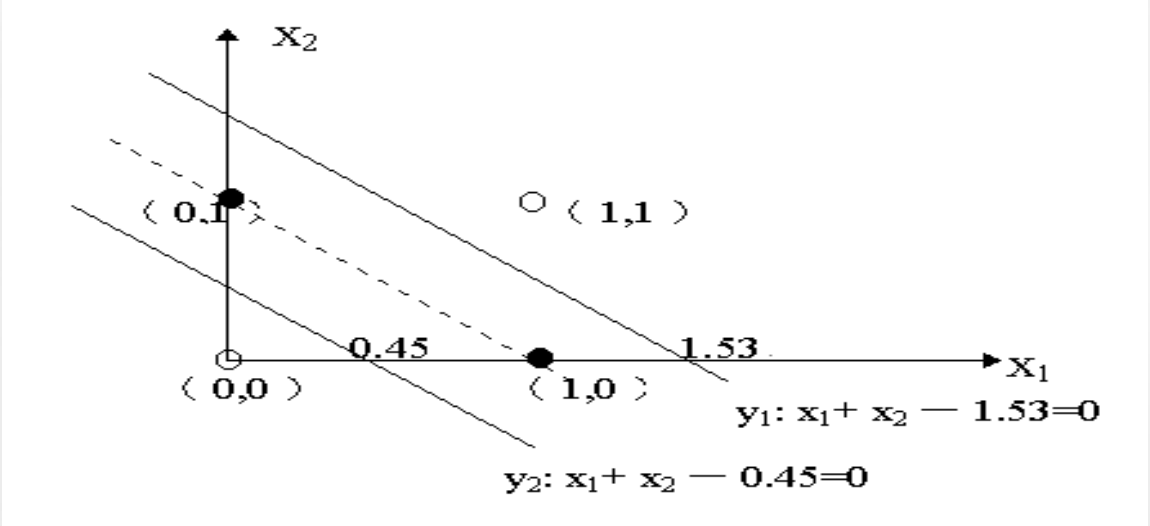
Linear equation of nodes in hidden layer:

y_1: $5.24x_1+5.23\ x_2-8.01=0$

$x_1+0.998\ x_2-1.529=0$,

y_2: $6.68x_1+6.64\ x_2-2.98=0$

$x_1+0.994\ x_2-0.446=0$



Three parts by y_1 and y_2:

- Beyond y_1, $x_1+x_2-1.53>0$, $x_1+x_2-0.45>0$
- Between y_1 and y_2, $x_1+x_2-1.53<0$, $x_1+x_2-0.45>0$
- Below y_2, $x_1+x_2-1.53<0$, $x_1+x_2-0.45<0$

Why—hidden layes

- ✓ conclution： The nodes in the hidden layer divided the four sample points

  (0,0)，(0,1)，(1,0)，(1,1),

  into three sample points：

  (0,0)，(0,1)，(1,1)

- ✓ The three points are linearly separable training patterns.

Why—hidden layes

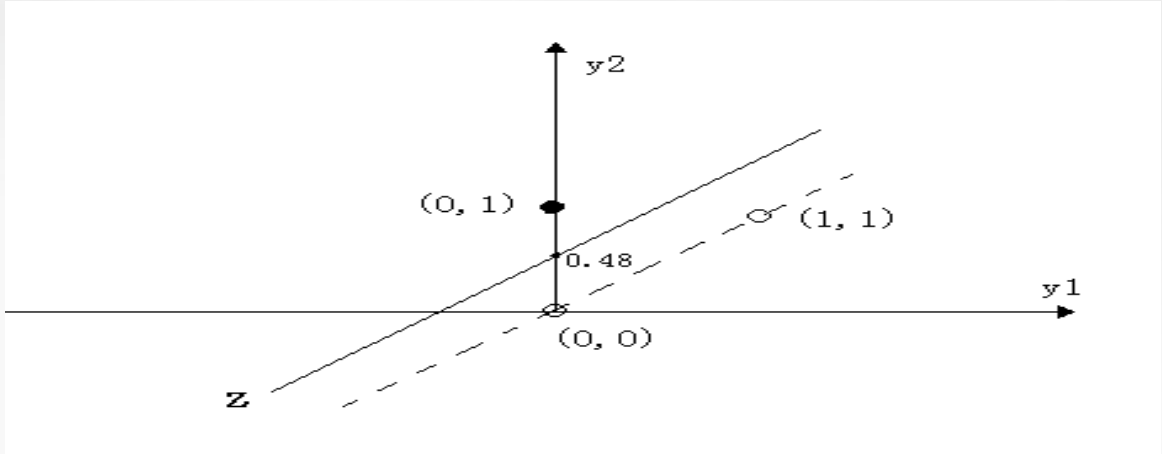
- ✓ Linear equation of nodes in output layer Z:

$-10y_1+10y_2-4.79=0$, $-y_1+y_2-0.479=0$

- ✓ Straight line Z divided the plane (y1,y2) into two parts:

$-y1+y2-0.479>0$; $-y_1+y_2-0.479<0$

- ✓ Conclution: output node classifies the three points (y1,y2plane) (0,0), (0,1), (1,1) into two categories, Z=1 and Z=0.



Why—hidden layes

The role of the neural network nodes

- Hidden layer nodes change nonlinear separated samples into linearly separable samples.
- Output layer nodes classify linearly separable samples into two categories.