COMP7/8118 M50

Data Mining

Neural Network

Xiaofei Zhang

Slides compiled from Jiawei Han and Raymond C.W. Wong's work

THE UNIVERSITY OF MEMPHIS

Neural Network

- Neural Network
  - A computing system made up of simple and highly interconnected processing elements
- Other terminologies:
  - Connectionist Models
  - Parallel distributed processing models (PDP)
  - Artificial Neural Networks
  - Computational Neural Networks
  - Neurocomputers

Neural Network

- This approach is inspired by the way that brains process information, which is entirely different from the way that conventional computers do
- Information processing occurs at many identical and simple processing elements called **neurons** (or also called **units**, **cells** or **nodes**)
- Interneuron connection strengths known as synaptic **weights** are used to store the knowledge

Advantages of Neural Network

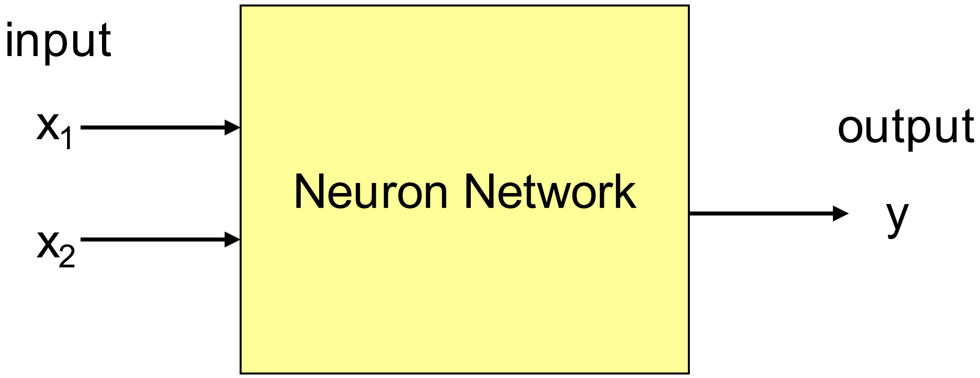
- **Parallel Processing** – each neuron operates individually
- **Fault tolerance** – if a small number of neurons break down, the whole system is still able to operate with slight degradation in performance.

Neuron Network

- Neuron Network for OR

x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	1

OR Function

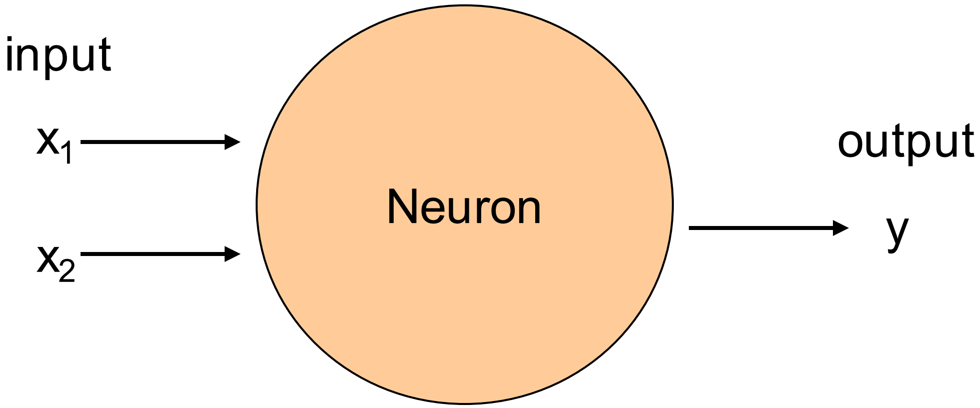


Neuron Network

x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	1

OR Function

- Neuron Network for OR

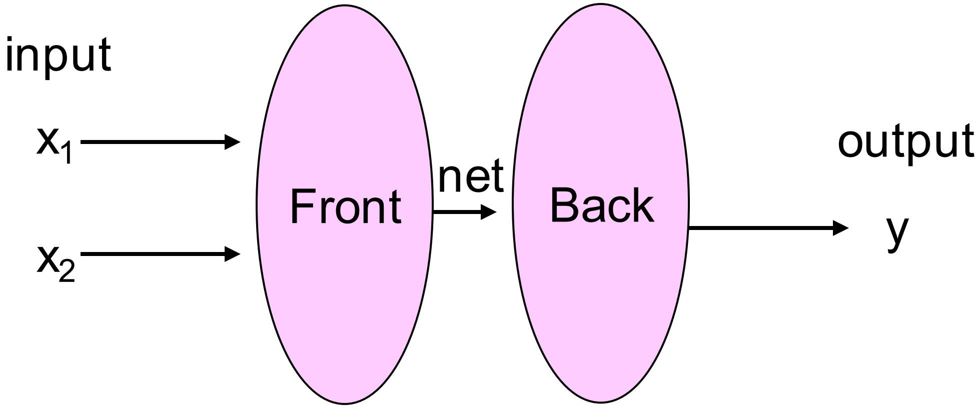


Neuron Network

x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	1

OR Function

- Neuron Network for OR

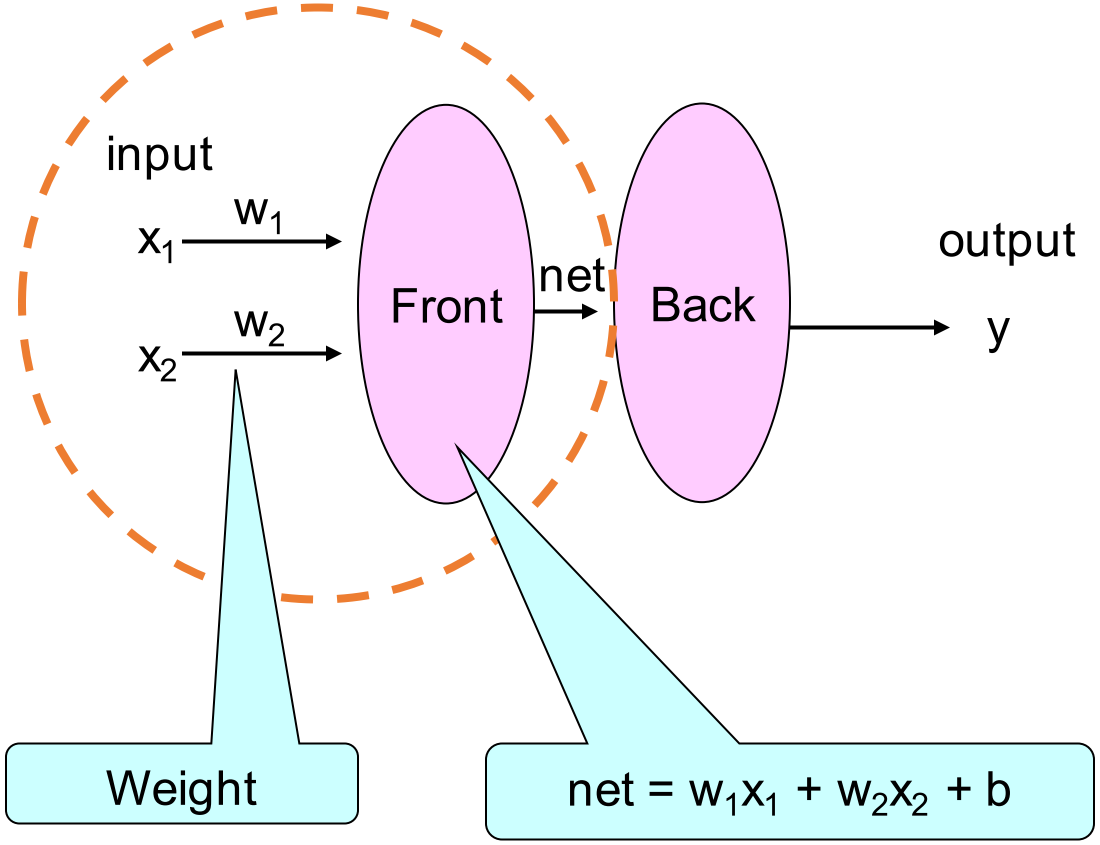


Neuron Network

x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	1

OR Function

- Neuron Network for OR

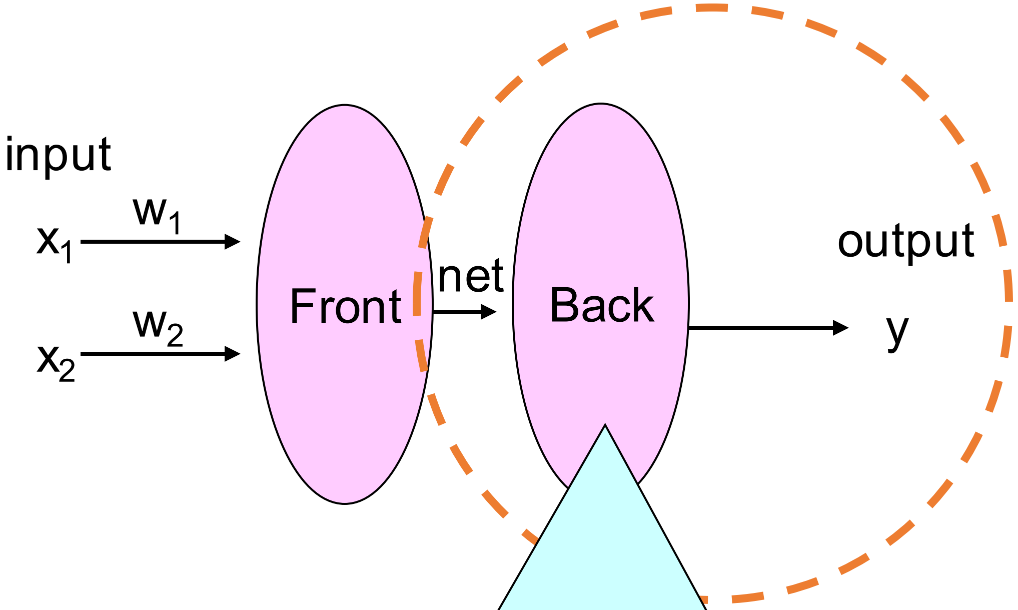


Neuron Network

- Neuron Network for OR

x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	1

OR Function



Activation function

-Linear function: $y = net$ or $y = a \cdot net$

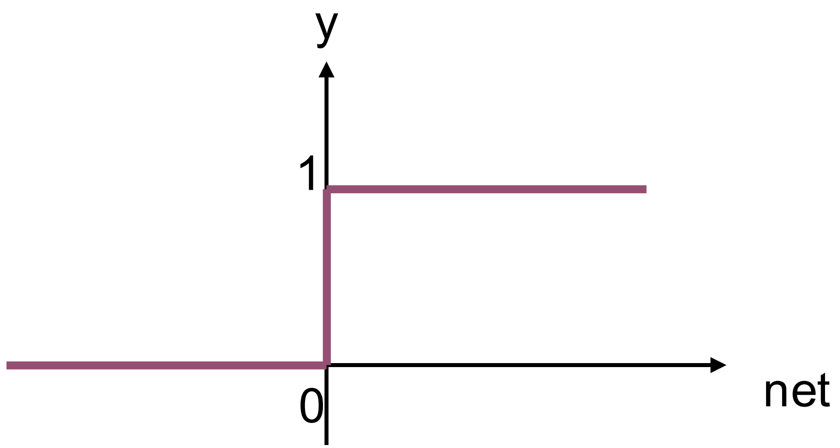
-Non-linear function

Activation Function

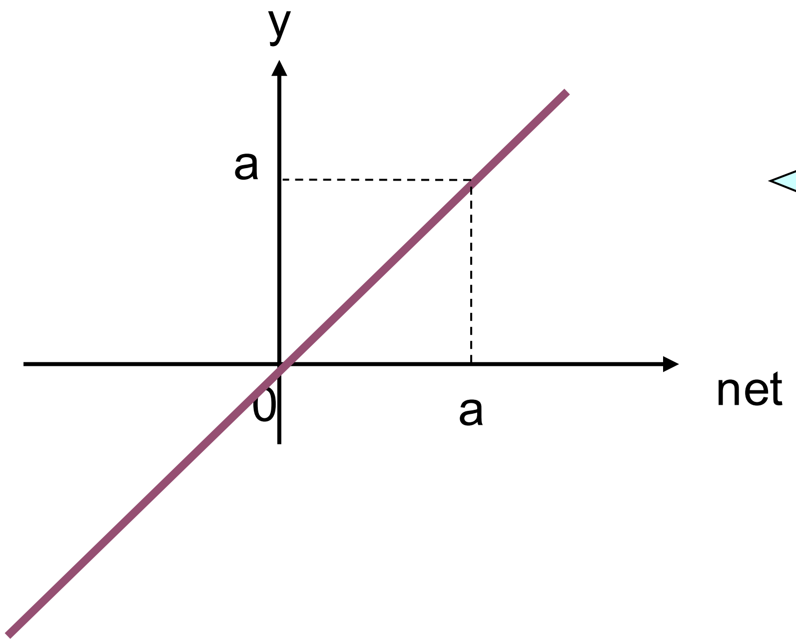
- Non-linear functions
  - Threshold function, Step Function, Hard Limiter
  - Linear Function (or Identity Function)
  - Rectifier Function/Rectified Linear Unit (ReLU)
  - Sigmoid Function
  - tanh Function

Threshold function, Step Function, Hard Limiter

$$y = \begin{cases} 1 & \text{if net } \geq 0 \\ 0 & \text{if net } < 0 \end{cases}$$



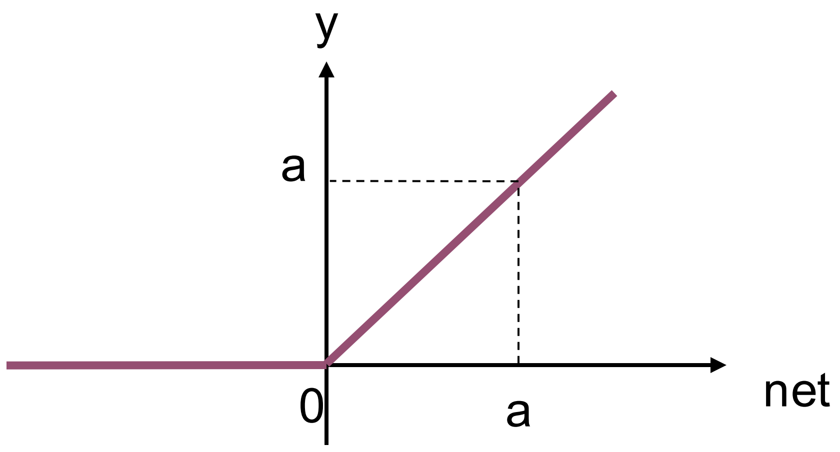
Linear function (or Identity function)



Some variants:
linear functions with
different slopes

Rectifier Function/Rectified Linear Unit (ReLU)

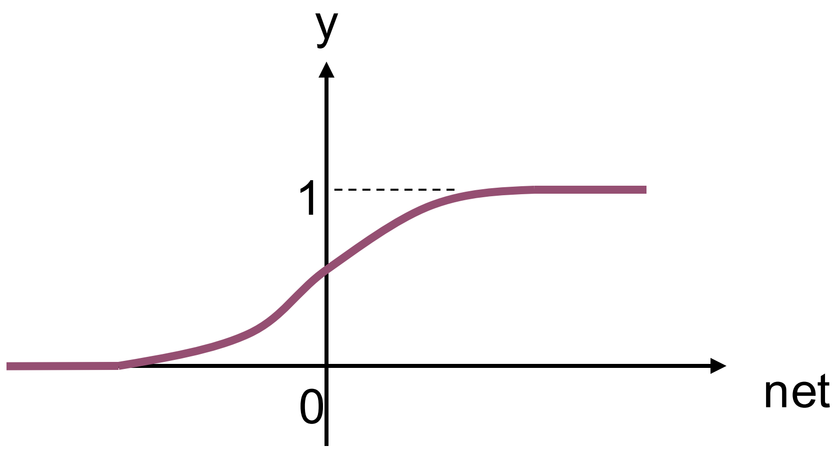
$$y = \begin{cases} net & \text{if } net \geq 0 \\ 0 & \text{if } net < 0 \end{cases}$$



Sigmoid Function

$$y = \frac{1}{1 + e^{-net}}$$

e is Euler's number (which is equal to 2.718)

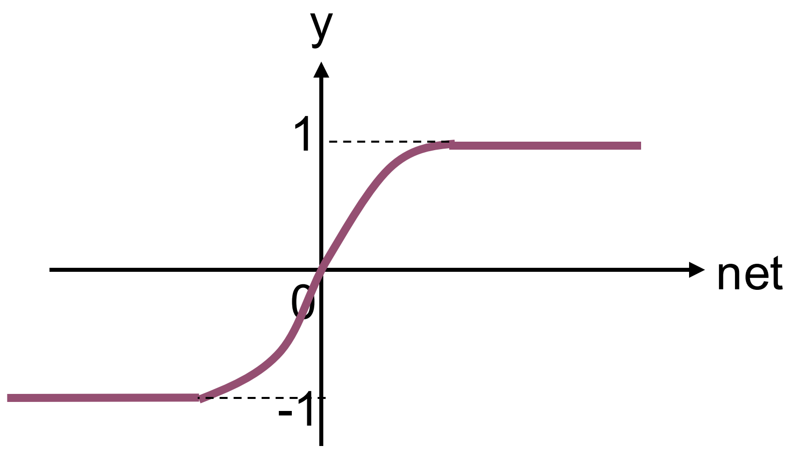


tanh Function

Hyperbolic tangent

$$y = \frac{e^{2\,net} - 1}{e^{2\,net} + 1}$$

e is Euler's number (which is equal to 2.718)

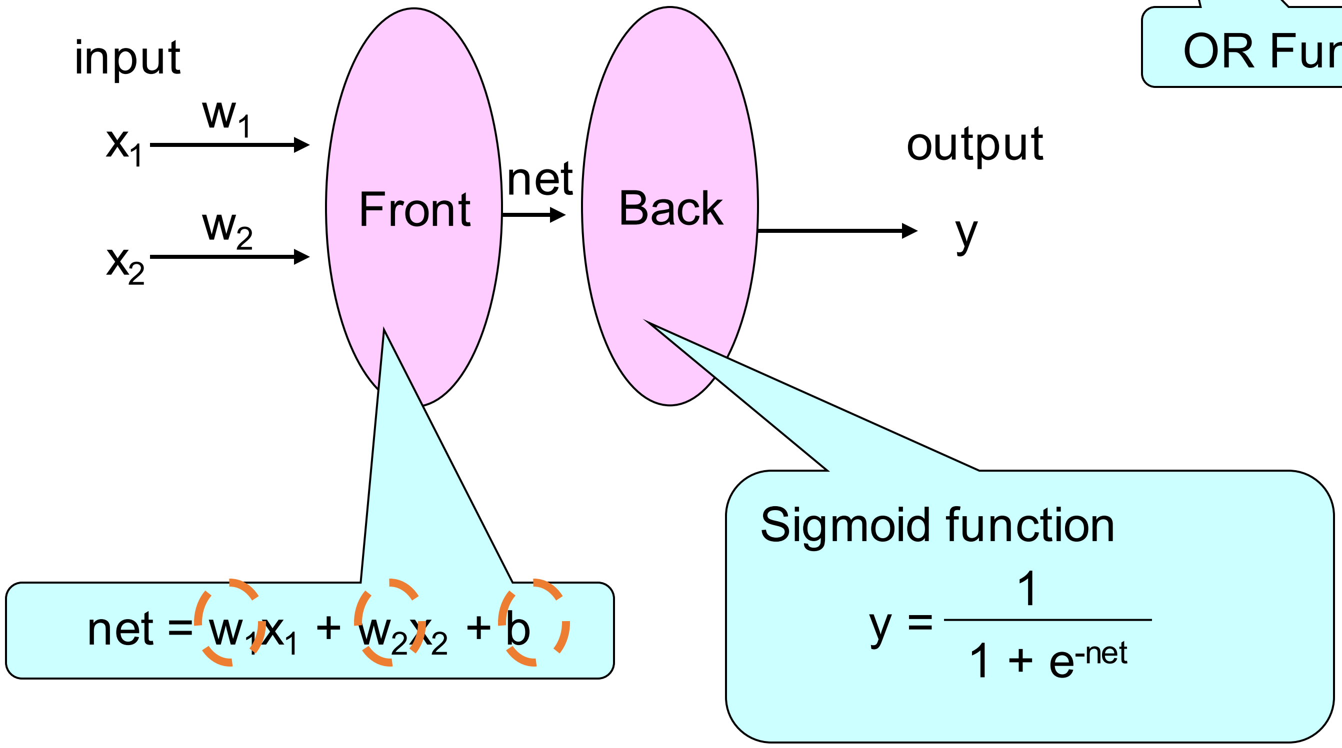


Neuron Network

- Neuron Network for OR

x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	1

OR Function



Learning

- Let α be the learning rate (a real number)
- Learning is done by
  - $w_i \leftarrow w_i + \alpha(d - y)x_i$
  - where
    - d is the desired output
    - y is the output of our neural network
  - $b \leftarrow b + \alpha(d - y)$

$$net = w_1x_1 + w_2x_2 + b$$

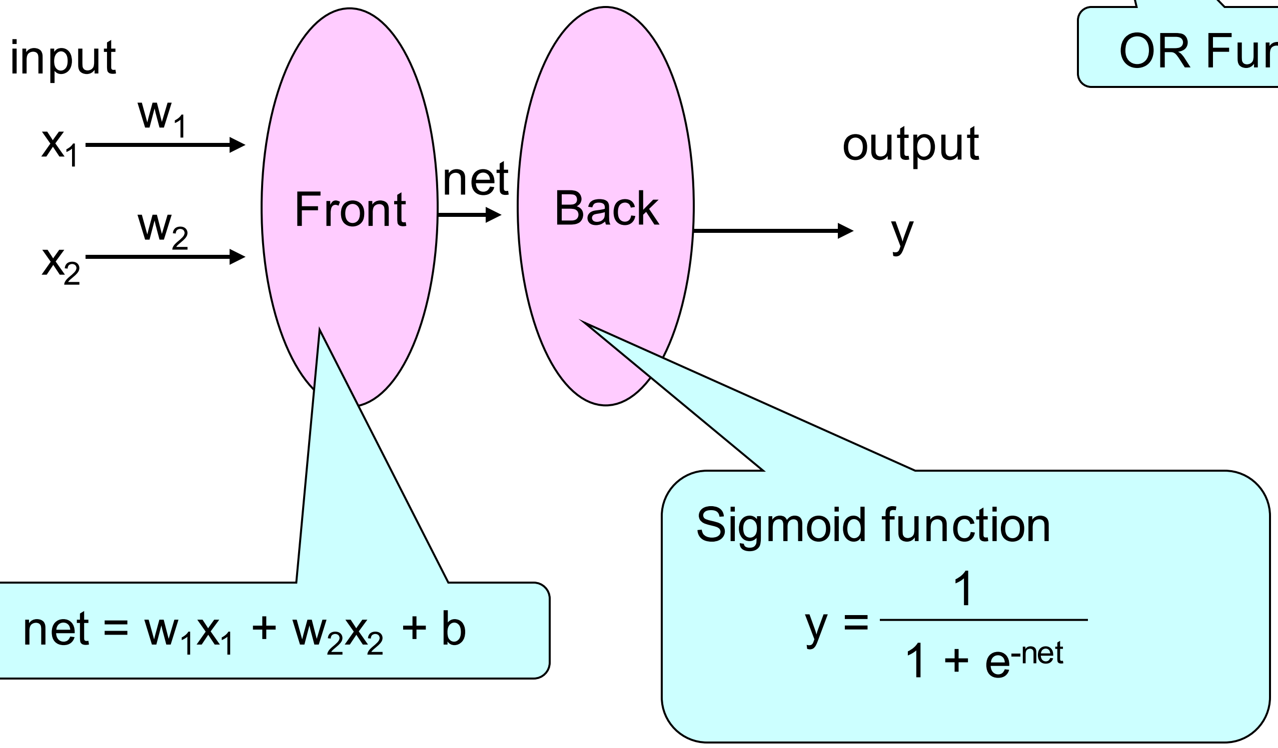
$$y = \frac{1}{1 + e^{-net}}$$

Neuron Network

x_1	x_2	d
0	0	0
0	1	1
1	0	1
1	1	1

OR Function

- Neuron Network for OR



Neuron Network

$net = w_1x_1 + w_2x_2 + b$

$$y = \frac{1}{1 + e^{-net}}$$

x_1	x_2	d
0	0	0
0	1	1
1	0	1
1	1	1

$\alpha = 0.8$

Step 1 (Input Forward Propagation)

$net = w_1x_1 + w_2x_2 + b$ $=1$

$y = 0.7311$

Step 2 (Error Backward Propagation)

$w_1 = w_1 + \alpha(d - y)x_1$

$= 1+0.8*(0-0.7311)*0$ $= 1$

$w_2 = w_2 + \alpha(d - y)x_2$

$= 1+0.8*(0-0.7311)*0$ $= 1$

$b = b + \alpha(d - y)$

$= 1+0.8*(0-0.7311)$ $= 0.4151$

b	w_1	w_2
1	1	1
0.4151	1	1

$net = w_1x_1 + w_2x_2 + b$

$$y = \frac{1}{1 + e^{-net}}$$

Neuron Network

x_1	x_2	d
0	0	0
0	1	1
1	0	1
1	1	1

$\alpha = 0.8$

Step 1 (Input Forward Propagation)

$net = w_1x_1 + w_2x_2 + b$ $=1.4151$

$y = 0.8046$

Step 2 (Error Backward Propagation)

$w_1 = w_1 + \alpha(d - y)x_1$

$= 1+0.8*(1-0.8046)*0$ $= 1$

$w_2 = w_2 + \alpha(d - y)x_2$

$= 1+0.8*(1-0.8046)*1$ $= 1.1563$

$b = b + \alpha(d - y)$

$= 0.4151+0.8*(1-0.8046)$ $= 0.5714$

b	w_1	w_2
0.4151	1	1

0.5714 1 1.1563

$net = w_1x_1 + w_2x_2 + b$

$$y = \frac{1}{1 + e^{-net}}$$

Neuron Network

x_1	x_2	d
0	0	0
0	1	1
1	0	1
1	1	1

$\alpha = 0.8$

Step 1 (Input Forward Propagation)

$net = w_1x_1 + w_2x_2 + b$ $=1.5714$

$y = 0.8280$

Step 2 (Error Backward Propagation)

$w_1 = w_1 + \alpha(d - y)x_1$

$= 1+0.8*(1-0.8280)*1$ $= 1.1376$

$w_2 = w_2 + \alpha(d - y)x_2$

$= 1.1563+0.8*(1-0.8280)*0$ $= 1.1563$

$b = b + \alpha(d - y)$

$= 0.5714+0.8*(1-0.8280)$ $= 0.7090$

b	w_1	w_2
0.5714	1	1.1563
0.7090	1.1376	1.1563

Neuron Network

$net = w_1x_1 + w_2x_2 + b$

$$y = \frac{1}{1 + e^{-net}}$$

x_1	x_2	d
0	0	0
0	1	1
1	0	1
1	1	1

$\alpha = 0.8$

Step 1 (Input Forward Propagation)

$net = w_1x_1 + w_2x_2 + b$ $=3.0029$

$y = 0.9527$

Step 2 (Error Backward Propagation)

$w_1 = w_1 + \alpha(d - y)x_1$

$= 1.1376+0.8*(1-0.9527)*1$ $= 1.1754$

$w_2 = w_2 + \alpha(d - y)x_2$

$= 1.1563+0.8*(1-0.9527)*1$ $= 1.1941$

$b = b + \alpha(d - y)$

$= 0.7090+0.8*(1-0.9527)$ $= 0.7468$

b	w_1	w_2
0.7090	1.1376	1.1563
0.7468	1.1754	1.1941

Neuron Network

- When we processed all data points (i.e., 4 data points) in the whole dataset, we say that we processed the dataset in one **epoch**.
- We could continue to process the data points again with the updated weight and bias variables (i.e., w_1, w_2 and b)

Neuron Network

$net = w_1x_1 + w_2x_2 + b$

$$y = \frac{1}{1 + e^{-net}}$$

x_1	x_2	d
0	0	0
0	1	1
1	0	1
1	1	1

$\alpha = 0.8$

Step 1 (Input Forward Propagation)

$net = w_1x_1 + w_2x_2 + b$ $=0.7468$

$y = 0.6785$

Step 2 (Error Backward Propagation)

$w_1 = w_1 + \alpha(d - y)x_1$

$= 1.1754+0.8*(0-0.6785)*0$ $= 1.1754$

$w_2 = w_2 + \alpha(d - y)x_2$

$= 1.1941+0.8*(0-0.6785)*0$ $= 1.1941$

$b = b + \alpha(d - y)$

$= 0.7468+0.8*(0-0.6785)$ $= 0.2040$

b	w_1	w_2
0.7468	1.1754	1.1941
0.2040	1.1754	1.1941

We repeat the above process until a stopping condition is satisfied

Neuron Network

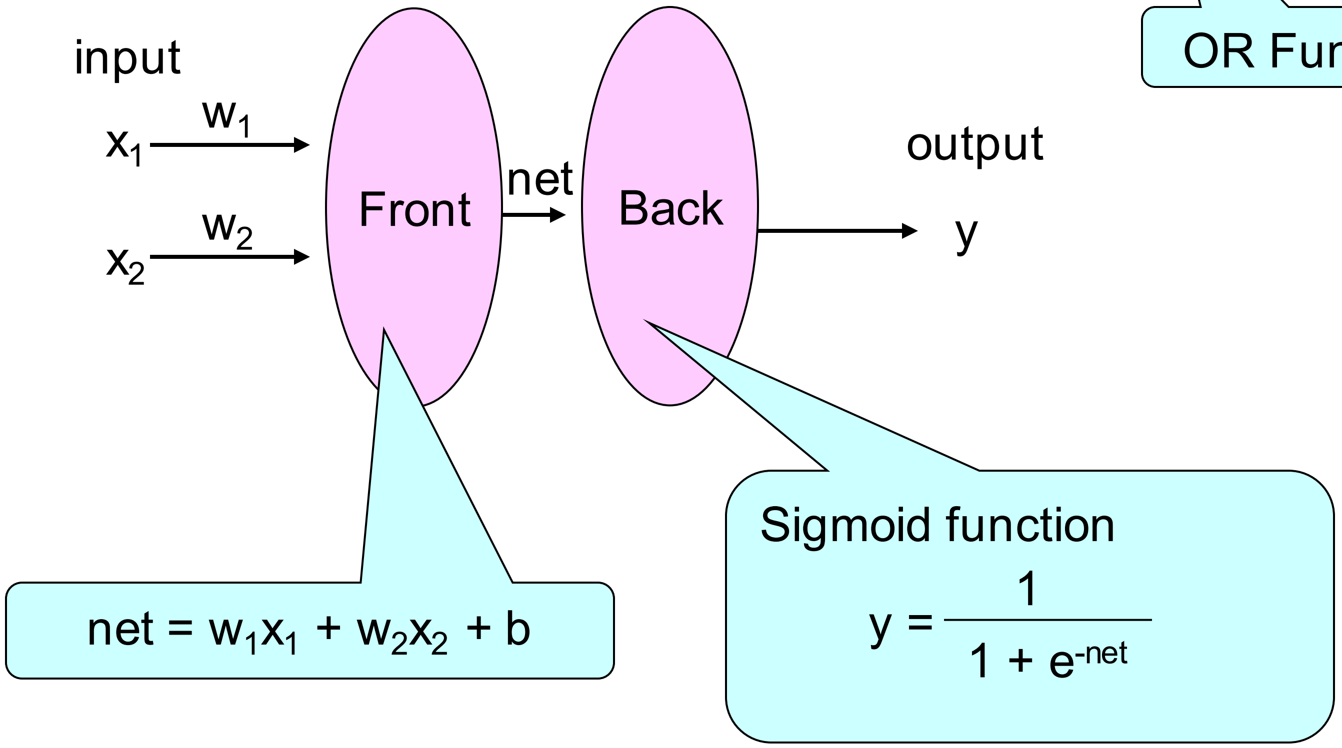
- The stopping condition could be specified with the following.
  - The **no. of epochs** is equal to 100

Neuron Network

- Neuron Network for OR

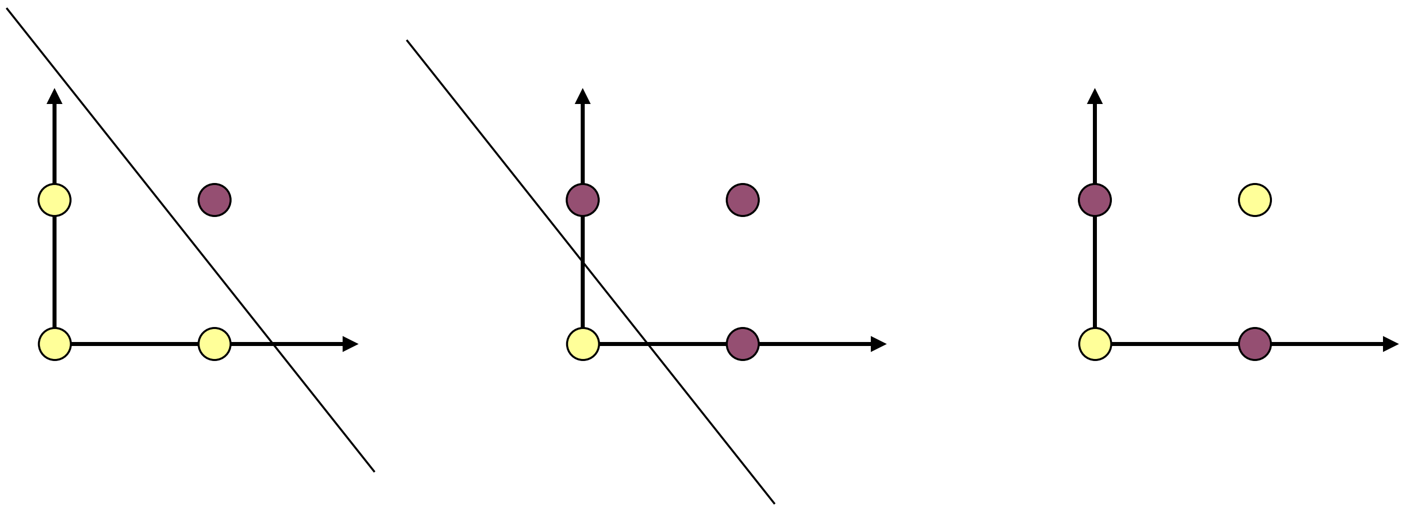
x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	1

OR Function

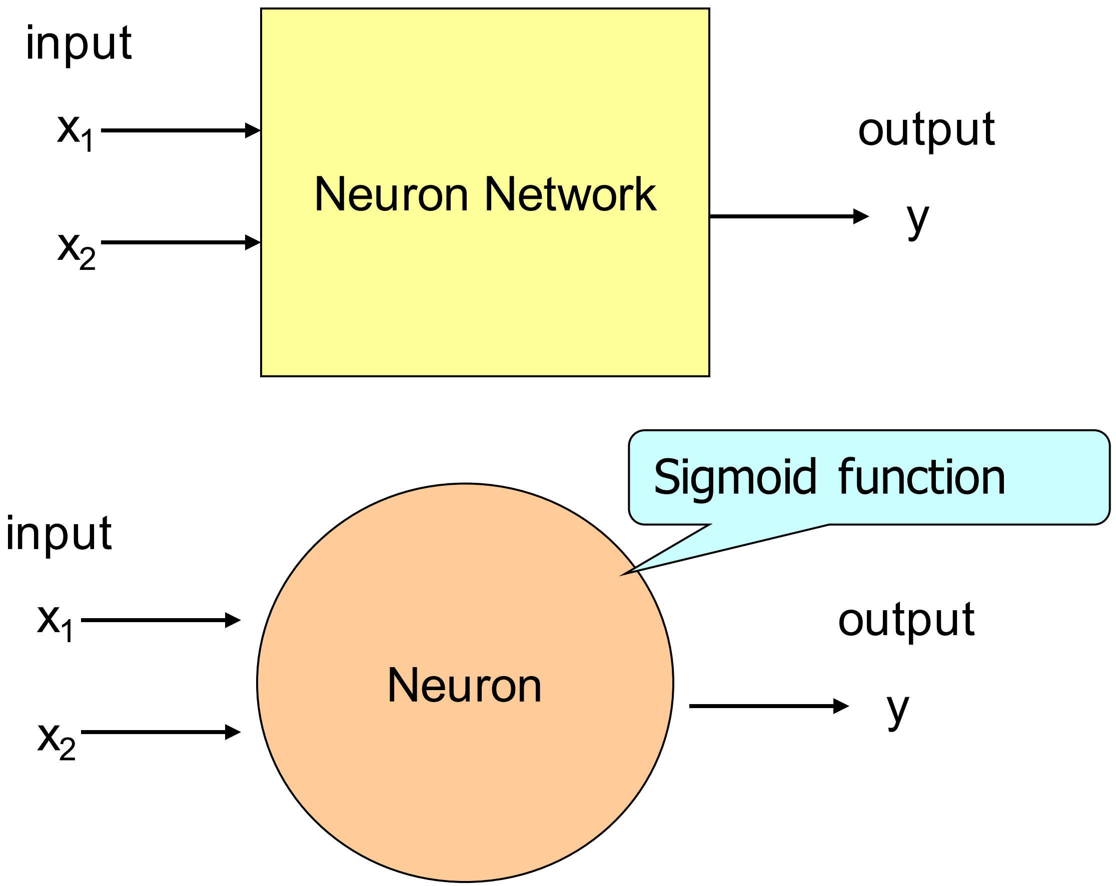


Limitation

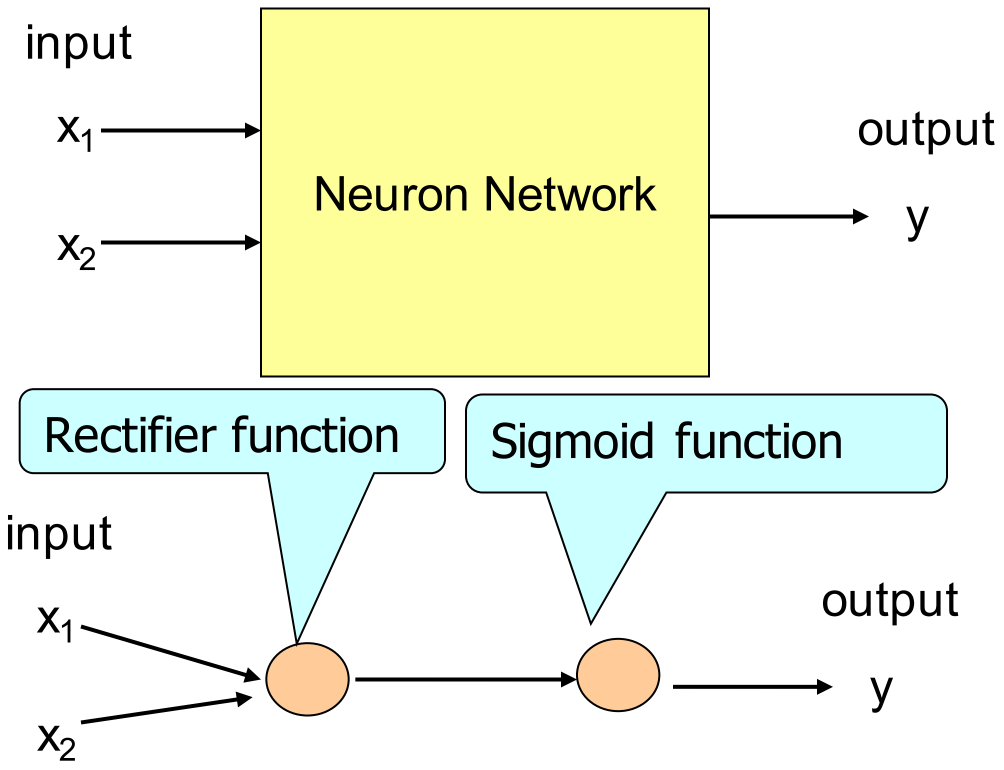
- It can only solve linearly separable problems



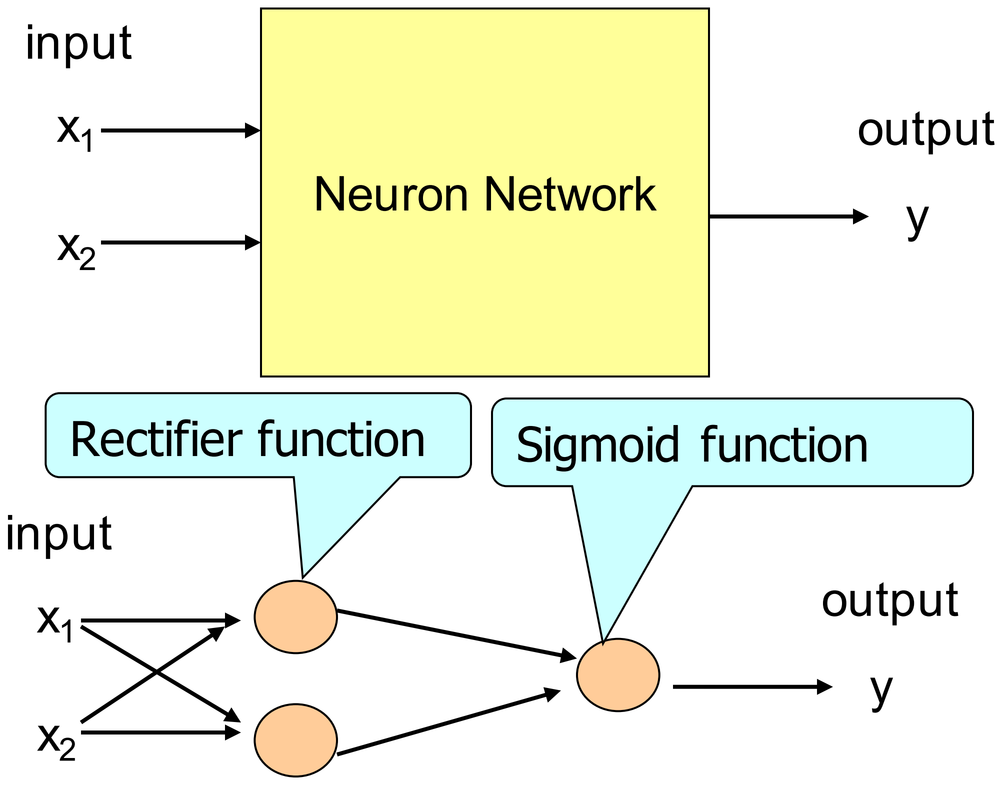
Multi-layer Perceptron (MLP)



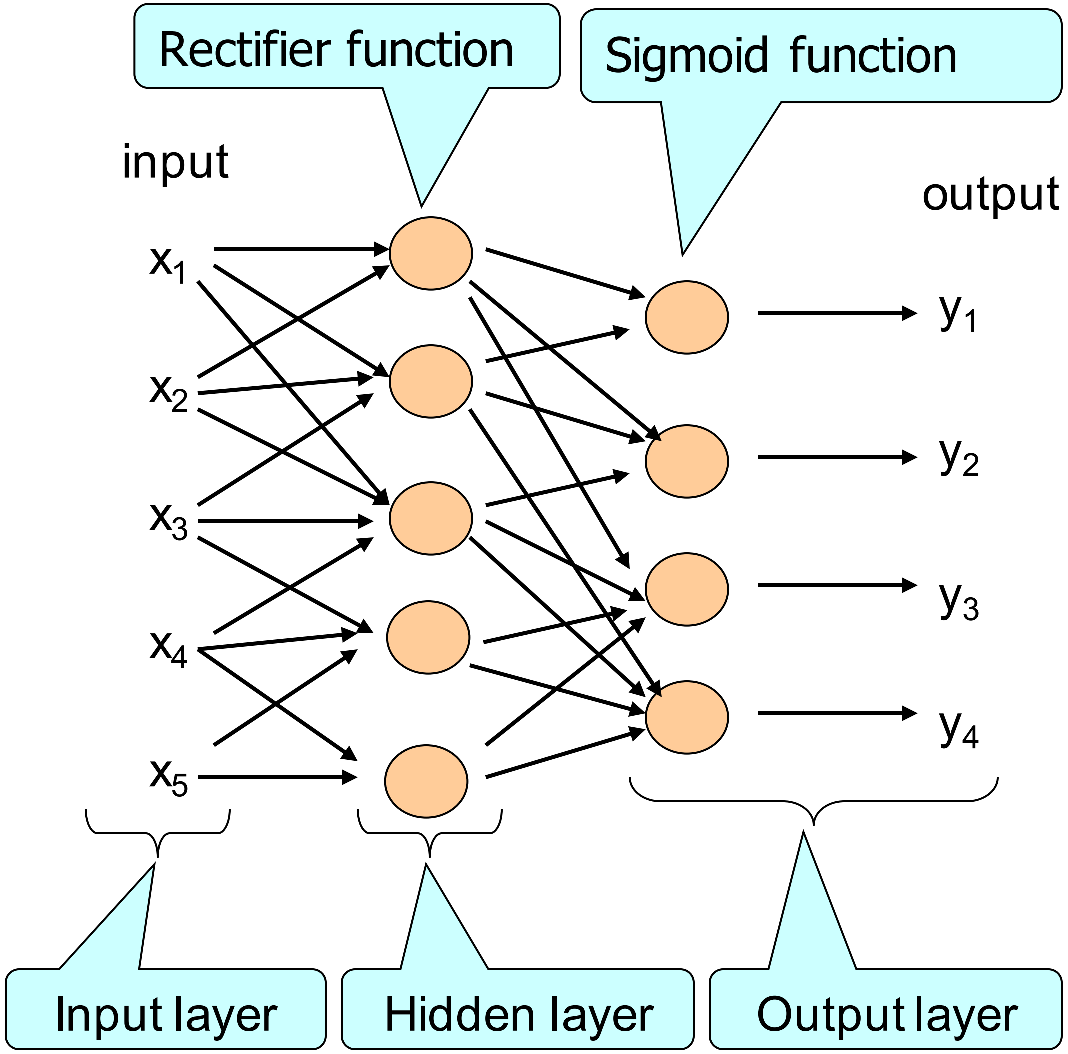
Multi-layer Perceptron (MLP)



Multi-layer Perceptron (MLP)



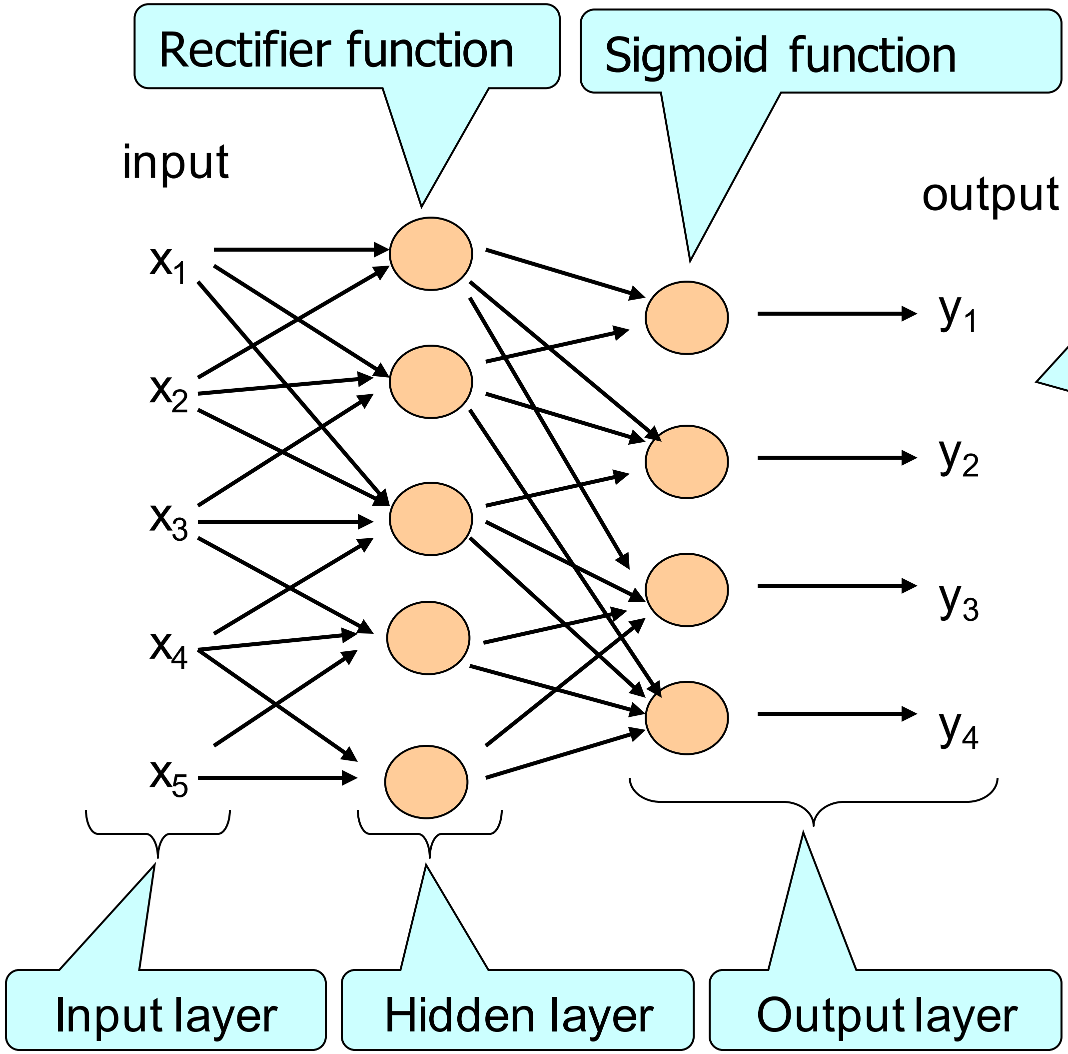
Multi-layer Perceptron (MLP)



Advantages of MLP

- Can solve
  - Linear separable problems
  - Non-linear separable problems
- A universal approximator
  - MLP has proven to be a universal approximator, i.e., it can model all types function y = f(x)

Multi-layer Perceptron (MLP)



If the total number of layers is very large, this neural network is called a "**deep network**"

If we use this deep network for training and then perform prediction or classification, we call this process "**deep learning**".

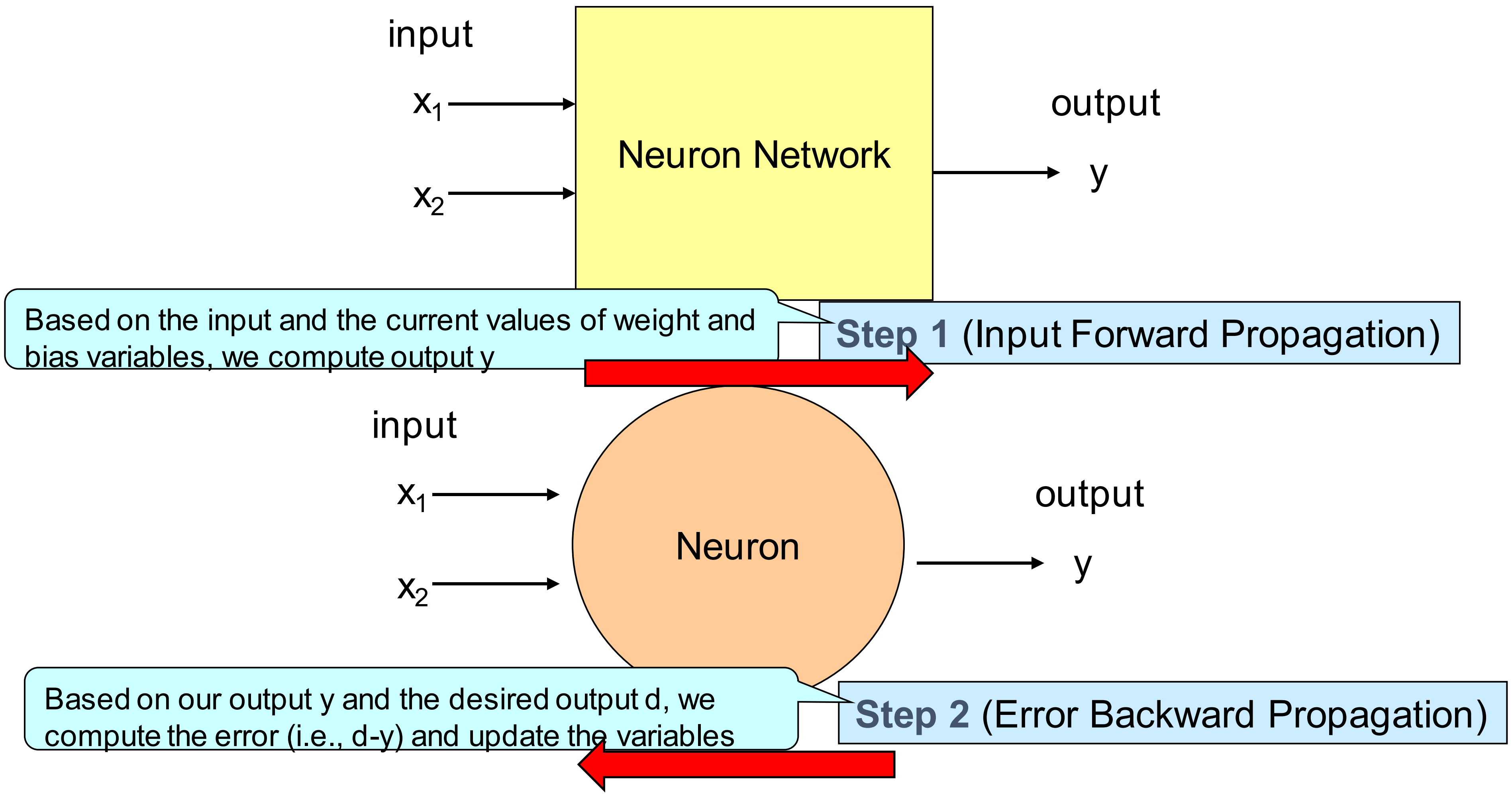
Multi-layer Perceptron (MLP)

- Nowadays, "deep learning" is very popular.
- It attracts a lot of attention from both academic and industries
- This is because it could solve a lot of problems "magically".
- One possible reason is that this deep network is "similar" to our human brain.
- Another possible reason is that it could solve a lot of problems with different types (e.g., classification and prediction/regression)

Multi-layer Perceptron (MLP)

- Next, we will present how to perform the following steps among "multiple nodes" in MLP (not just a single neuron)
  - **Step 1** (Input Forward Propagation)
  - **Step 2** (Error Backward Propagation)

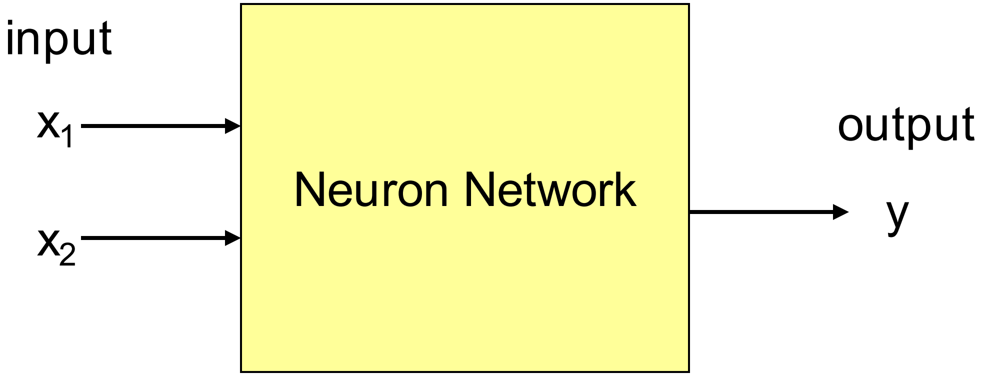
Multi-layer Perceptron (MLP)

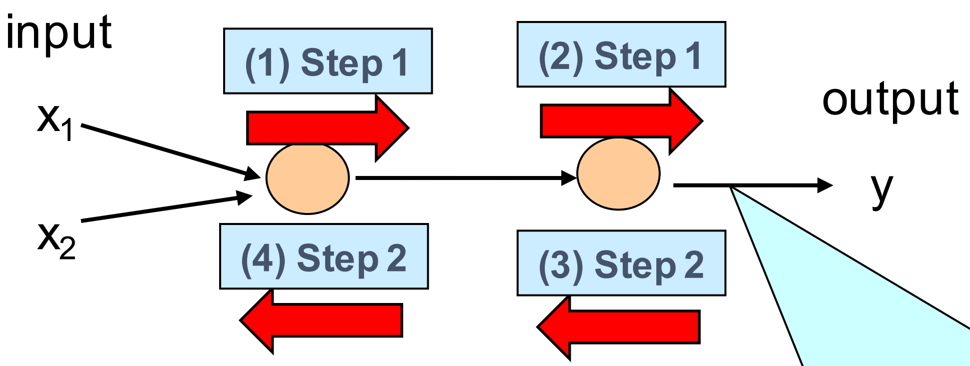


Multi-layer Perceptron (MLP)

Step 1 (Input Forward Propagation)

Step 2 (Error Backward Propagation)



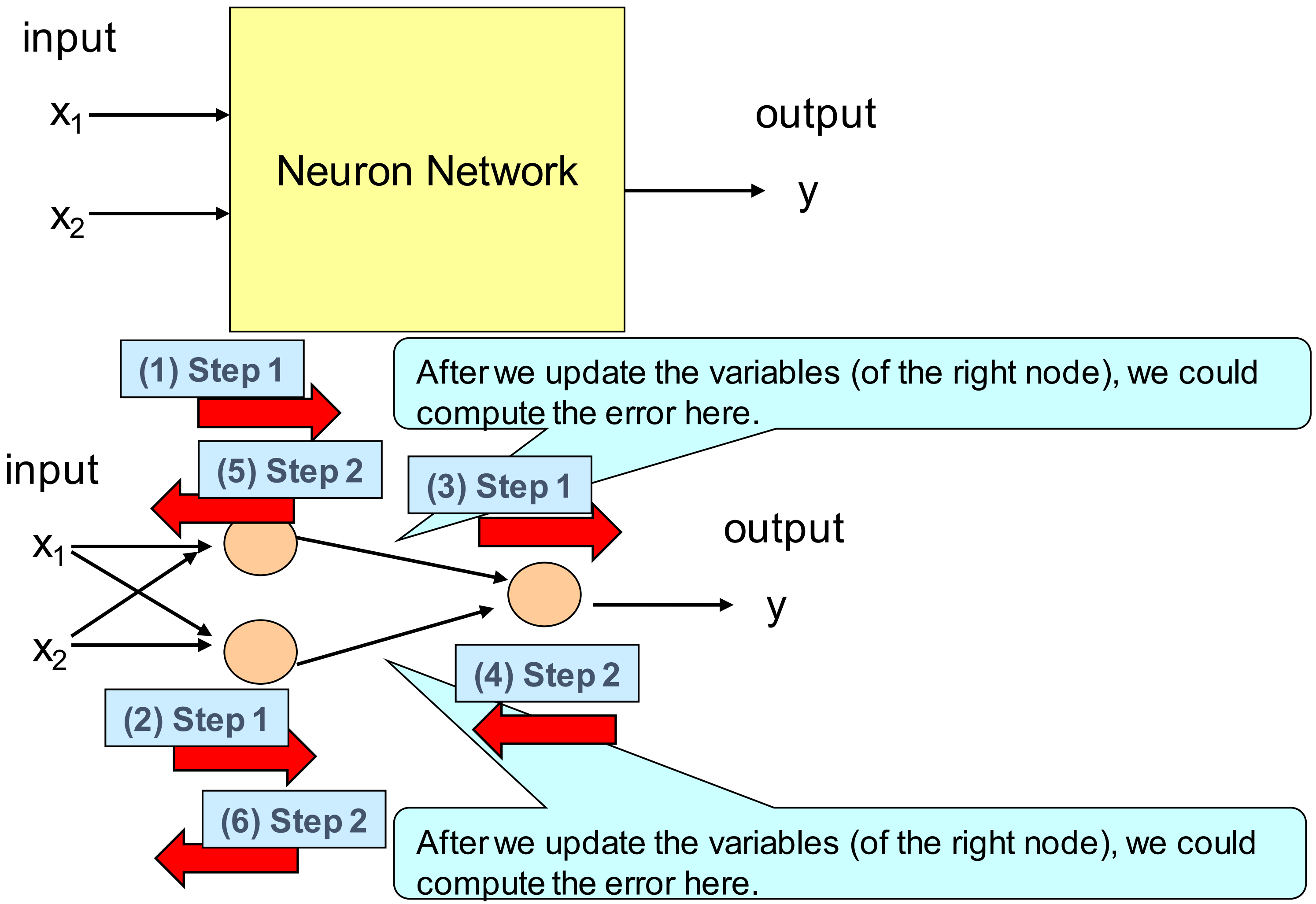


After we update the variables (of the right node), we could compute the error here.

Multi-layer Perceptron (MLP)

Step 1 (Input Forward Propagation)

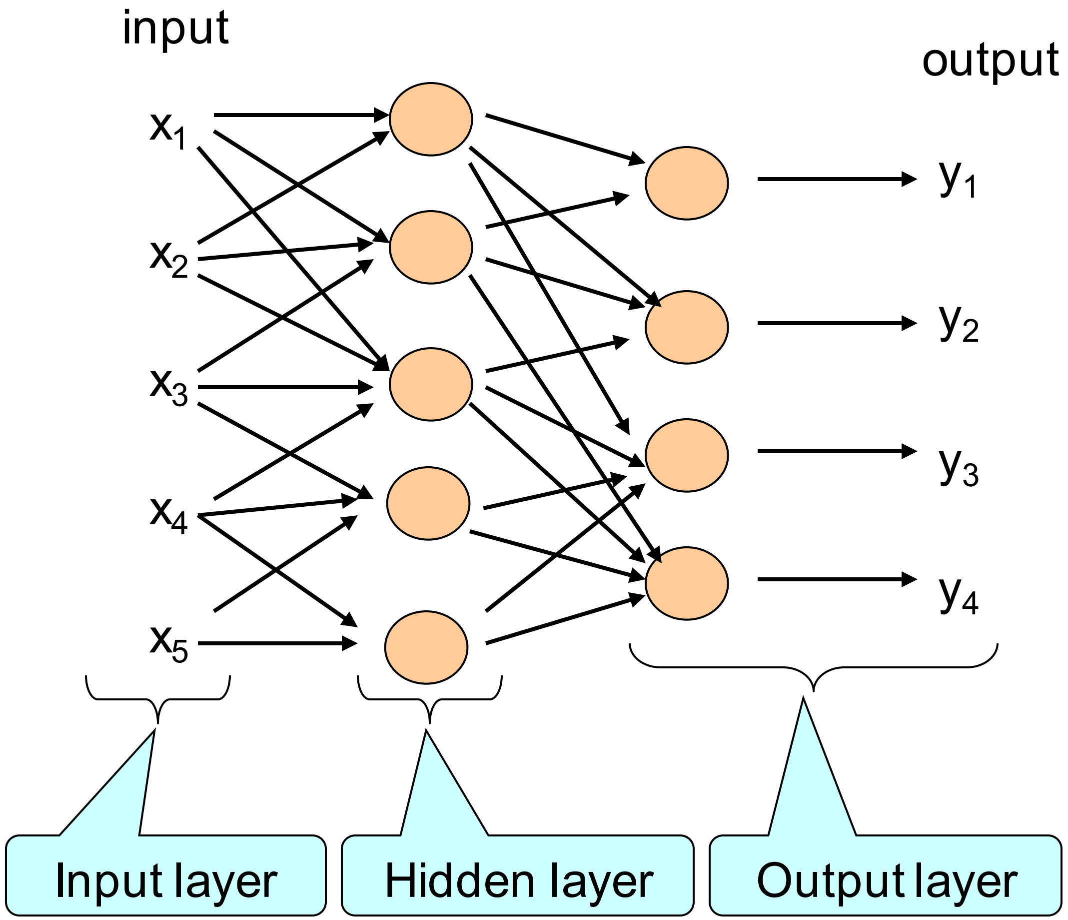
Step 2 (Error Backward Propagation)



Multi-layer Perceptron (MLP)

Step 1 (Input Forward Propagation)

Step 2 (Error Backward Propagation)



First, we perform "Step 1" for each node in the hidden layer

Then, we perform "Step 1" for each node in the output layer

We perform "Step 2" for each node in the output layer

Then, we perform "Step 2" for each node in the hidden layer