Artificial Intelligence

Instructor: Kietikul Jearanaitanakij
Department of Computer Engineering
King Mongkut's Institute of Technology Ladkrabang

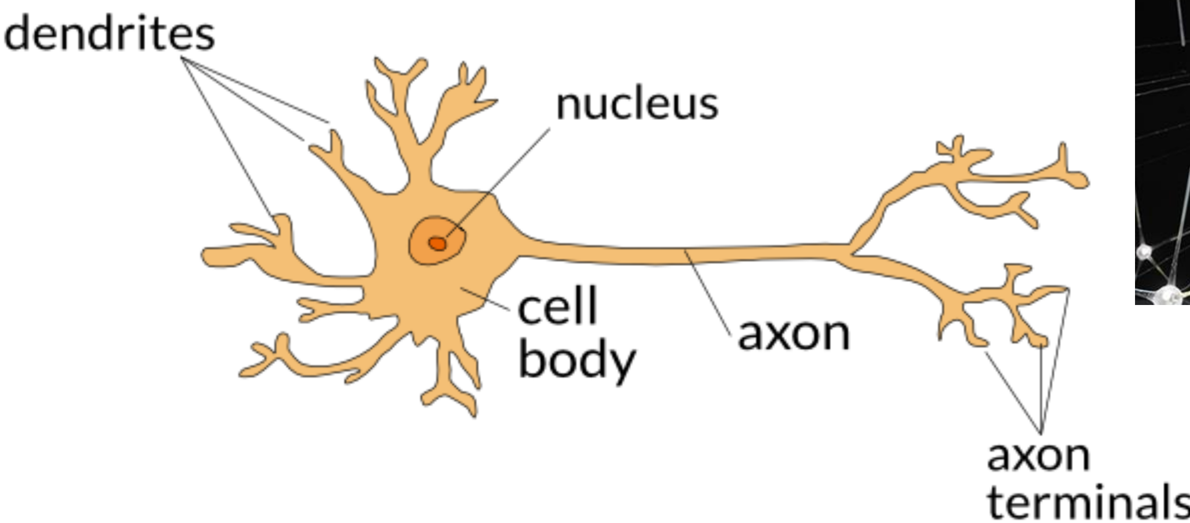
Lecture 9

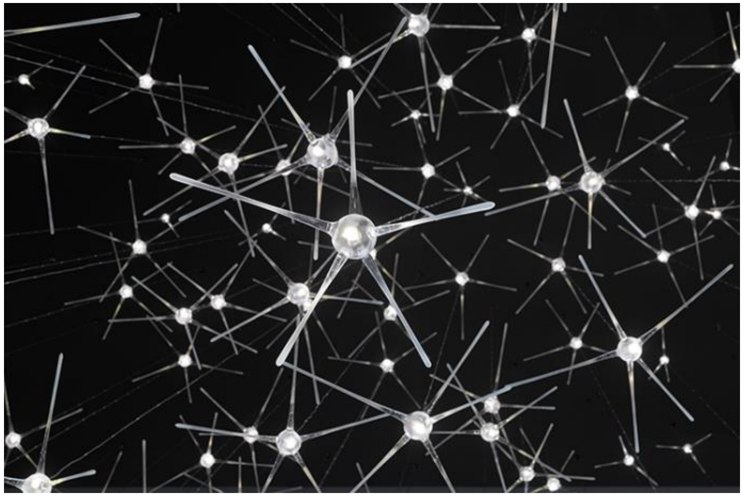
Perceptron And Artificial Neural Networks

- Linear Classification With Perceptron
- Perceptron learning algorithm
- Multilayer neural networks
- Backpropagation for multilayer NN

Linear Classification With Perceptron

Biological neuron

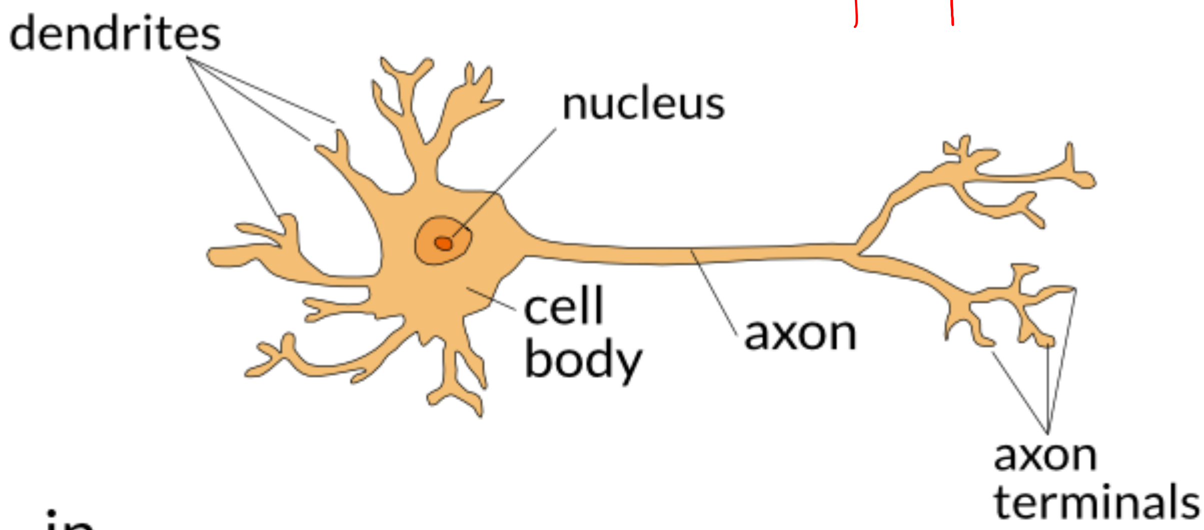


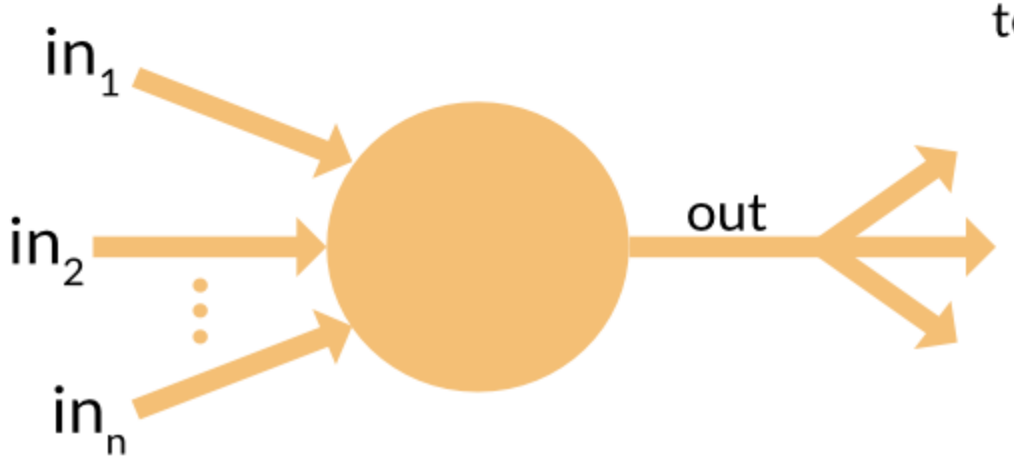


- Dendrites receive signals
- Axon sends signals out to other neurons

Artificial neuron

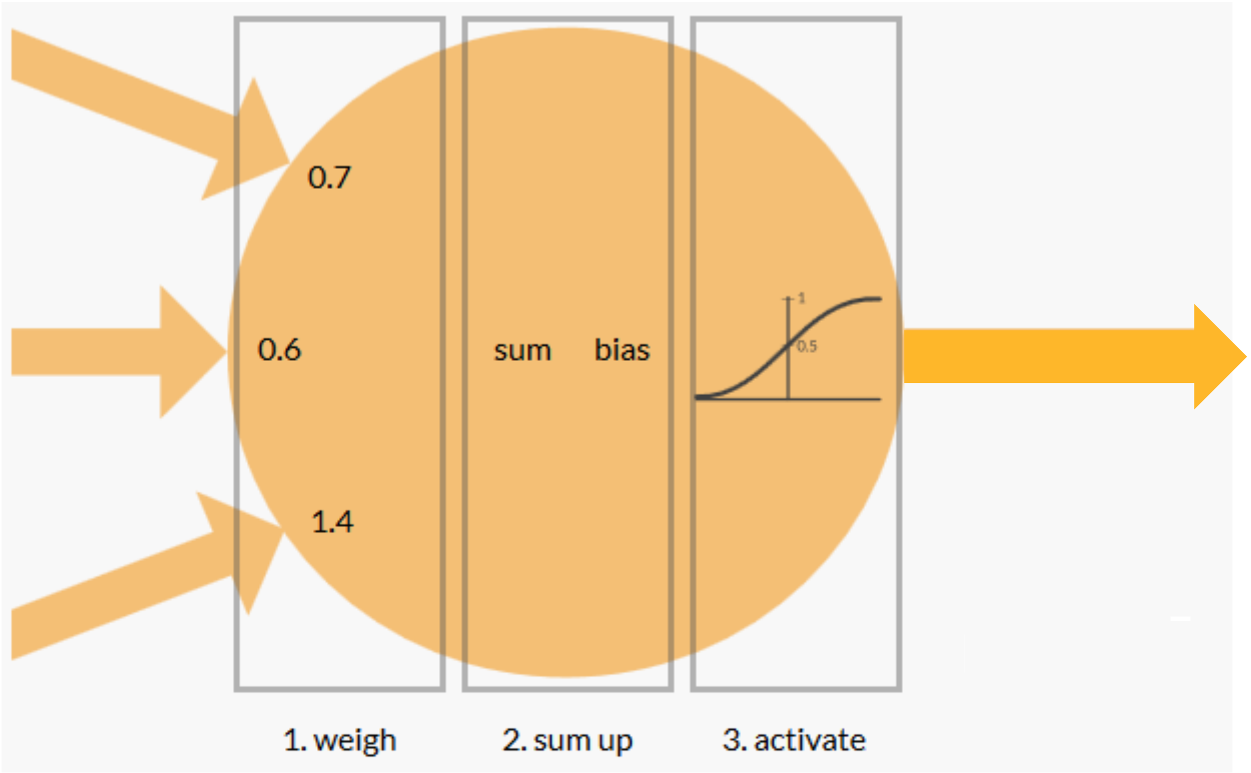
perceptron (cin)





Inside artificial neuron

Input signals

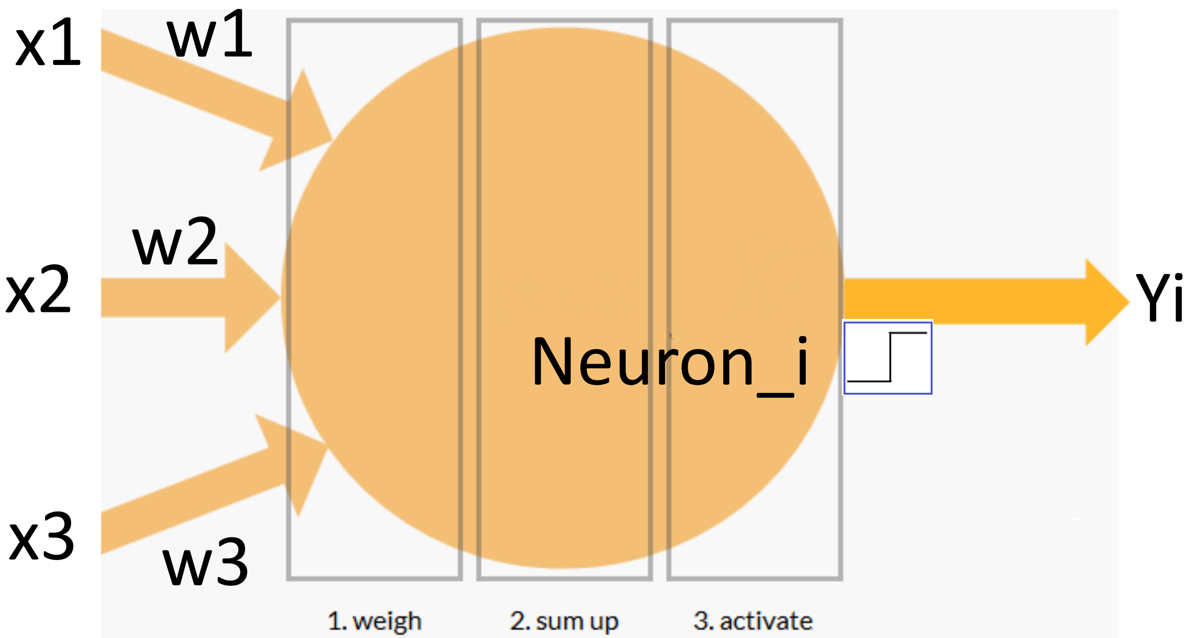


Output signal

We will call this artificial neuron as a **perceptron**.

Notations

Perceptron is the simplest form of a neural network. It consists of a single neuron with adjustable weights and a hard limit activation function.

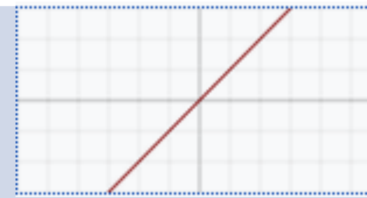
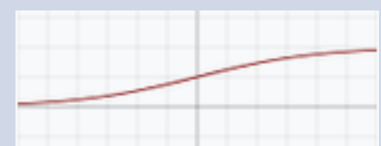
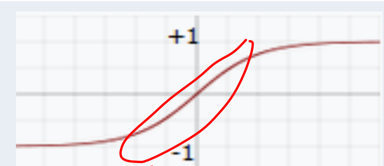


There are many kinds of activation function.

Frank Rosenblatt

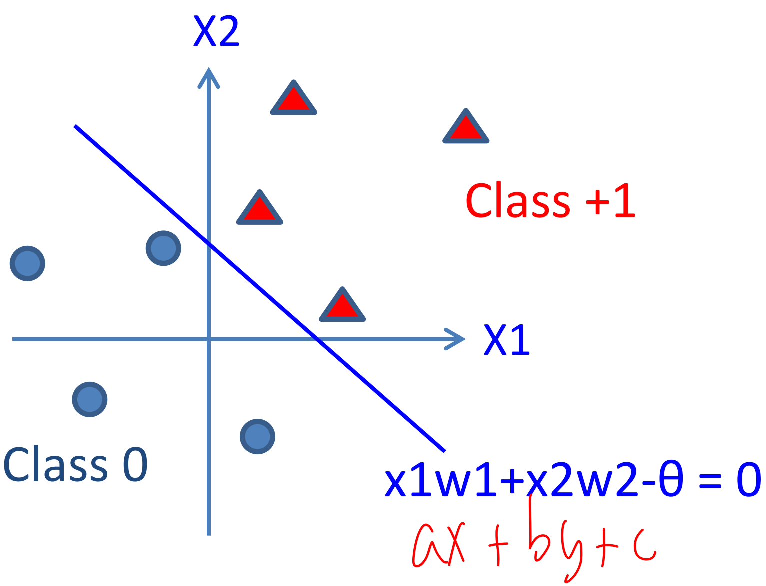
invented perceptron in 1957

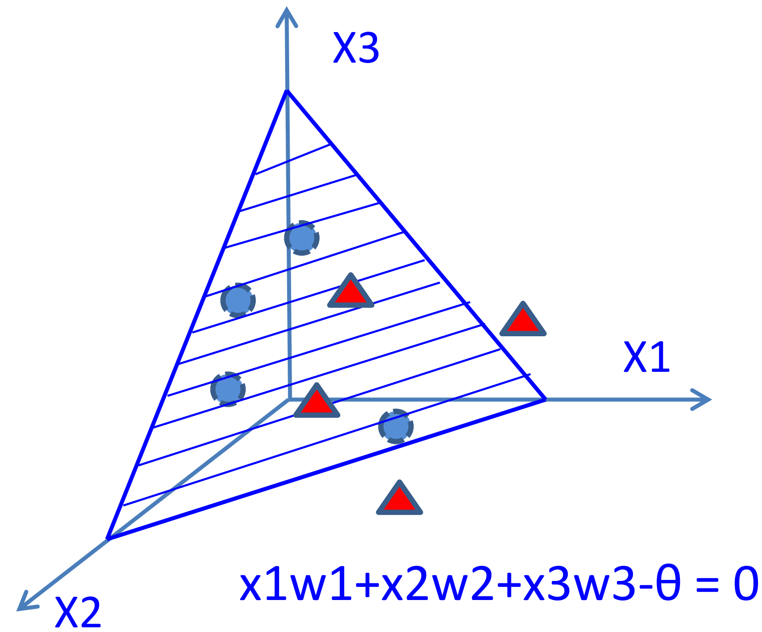
Activation functions

Name	Equation	Plot
Identity (Linear)	$f(x) = x$ useless	
Binary step	$f(x) = \begin{cases} 0 & \text{for} & x < 0 \\ 1 & \text{for} & x \geq 0 \end{cases}$	 step
Sigmoid (Logistic)	$f(x) = \frac{1}{1+e^{-x}}$	
TanH	$f(x) = \tanh(x) = \frac{2}{1+e^{-2x}} - 1$	 slope
Softmax	$f_i(\vec{x}) = \frac{e^{x_i}}{\sum_{j=1}^{J} e^{x_j}}$ for $i = 1, \ldots, J$	

Linear separability

Data in the n-dimensional space is **linearly separable** if two classes of data are divided by a hyperplane into two decision regions.





In general, the hyperplane is defined by the linearly separable function.

$$\sum_{i=1}^{n} x_i w_i - \theta = 0$$

(The threshold θ can be used to shift the decision boundary.)

- The perceptron learns its classification task by making small adjustments in the weights to reduce the difference between the actual and desired (target) outputs of the perceptron.

Target value   Actual value

$$e(p) = Y_d(p) - Y(p)$$

error   desire   output ที่ออกมาจริง ๆ

Where

p is the training pattern (example)

$Y_d(p)$ is the desired (target) output of pattern p

Y(p) is the actual output

e(p) is the difference (error) between $Y_d(p)$ and Y(p)

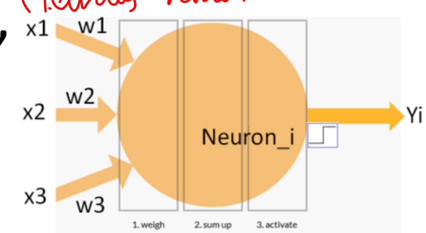
น้ำหนักใหม่ = น้ำหนักเก่า + ไปทางไหน x ระยะ

(learning rate)

- It uses e(p) to update weights of the next iteration,

$$w_i(p+1) = w_i(p) + \alpha . x_i(p) . e(p)$$

where α is the learning rate (0~1)



Perceptron learning algorithm

Step 1: Initialization เริ่มสร้าง weight

– Set initial weights $w_1, w_2, \ldots, w_n$ and thresholds (θ) to small random numbers in the range [-0.5,+0.5]

– p = 1 ; the first pattern

ช่วงแคบๆ

Step 2: Activation คำนวณค่า

Step function:

$$f(x) = \begin{cases} 0 & \text{for} \quad x < 0 \\ 1 & \text{for} \quad x \geq 0 \end{cases}$$

$$Y(p) = step\left(\sum_{i=1}^{n} x_i(p).w_i(p) - \theta\right)$$

where n is the number of perceptron inputs.

Step 3: Weight training by using gradient descent

$$w_i(p+1) = w_i(p) + \Delta w_i(p),$$

$$\Delta w_i(p) = \alpha . x_i(p) . e(p) \qquad \text{where } e(p) = Yd(p) - Y(p)$$

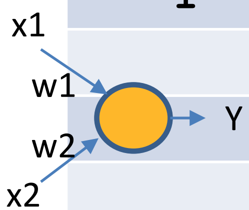
Step 4: Iteration เริ่ม p = 2

Increase p by one, go back to step 2 until each pattern is trained.

Step 5: If the perceptron doesn’t converge, p = 1 and repeat steps 2 – 4.

Example: Train a perceptron on AND

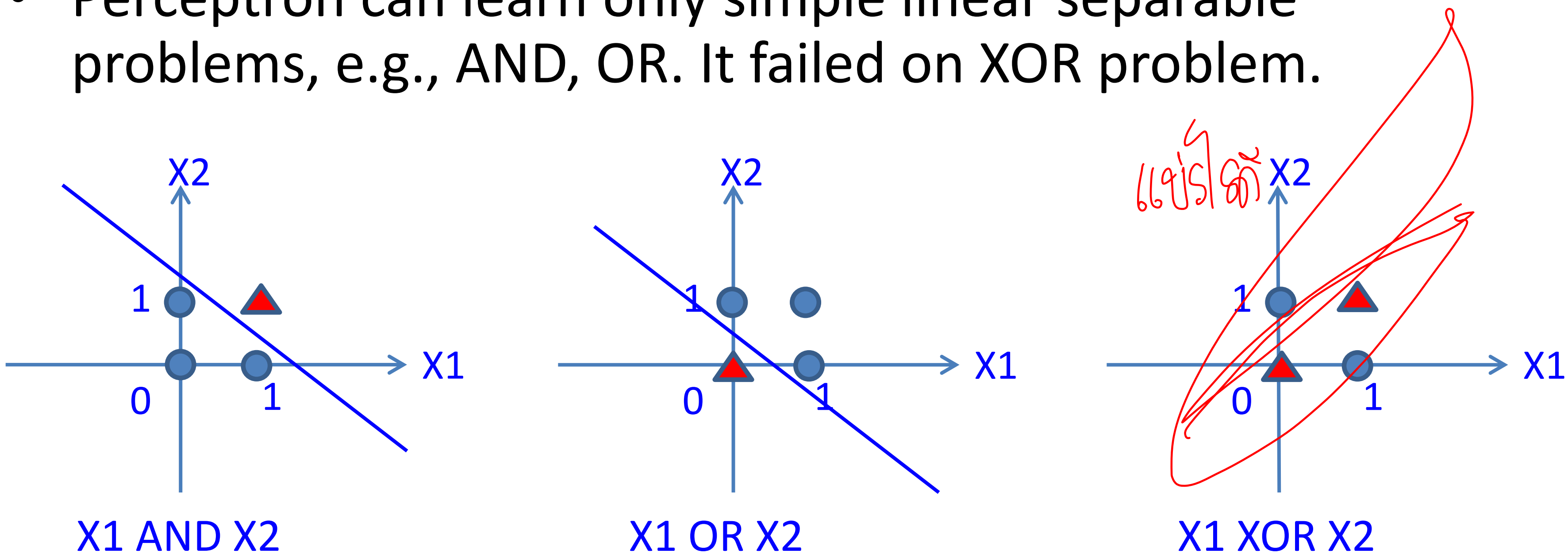
θ= 0.3, α = 0.1



Epoch	Inputs		Yd	Weights		Y	Error	Weights	
	x1	x2	(desired)	w1	w2	(actual)		w1	w2
1	0	0	0	0.3	-0.1	0	0	0.3	-0.1
	0	1	0	0.3	-0.1	0	0	0.3	-0.1
	1	0	0	0.3	-0.1	1	-1	0.2	-0.1
	1	1	1	0.2	-0.1	0	1	0.3	0.0
2	0	0	0	0.3	0.0	0	0	0.3	0.0
	0	1	0	0.3	0.0	0	0	0.3	0.0
	1	0	0	0.3	0.0	1	-1	0.2	0.0
	1	1	1	0.2	0.0	1	0	0.2	0.0
.	.		.	.		.	.	.	
.	.		.	.		.	.	.	
.	.		.	.		.	.	.	
5	0	0	0	0.1	0.1	0	0	0.1	0.1
	0	1	0	0.1	0.1	0	0	0.1	0.1
	1	0	0	0.1	0.1	0	0	0.1	0.1
	1	1	1	0.1	0.1	1	0	0.1	0.1

Problem of perceptron

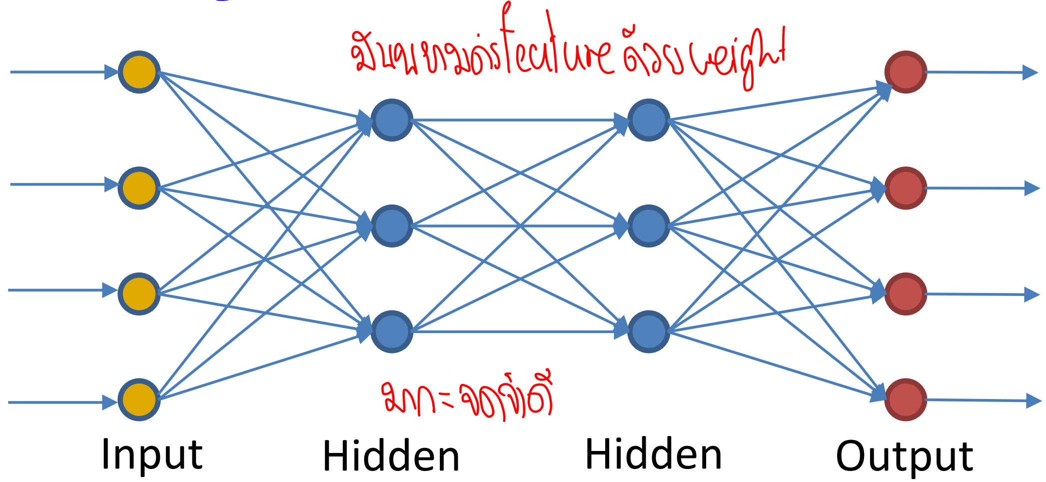
- Perceptron can learn only simple linear separable problems, e.g., AND, OR. It failed on XOR problem.



- A single perceptron can classify only linear separable problems, regardless of whether we use a hard-limit or soft-limit activation functions.
- Moreover, increasing the number of perceptrons in the same layer doesn't help.

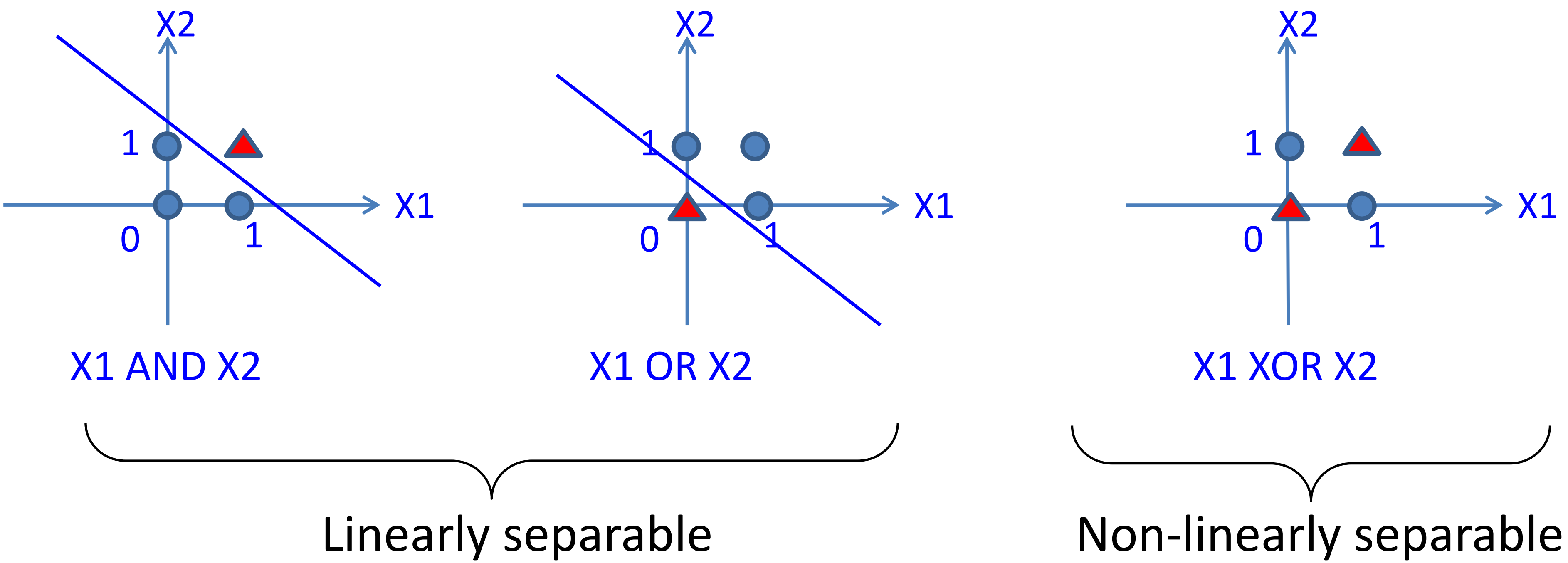
ยุคแรกของ nn

Multilayer neural networks



- Input layer accepts input signals from the outside world and redistributes these signals to all neurons in the hidden layer.
- Hidden layers detect the features of the input patterns by adjusting weights of the neurons.
- Output layer accepts signals from the hidden layer and establishes the output pattern of the network.
- Multilayer neural networks can solve the **non-linearly separable problem**.

Nonlinearly separable problem



- With one hidden layer, we can represent the continuous and simple discontinuous functions.
- With two hidden layers, even discontinuous function can be represented.

จับไป

ปกติไม่มีใครใช้ถึงสอง 55555

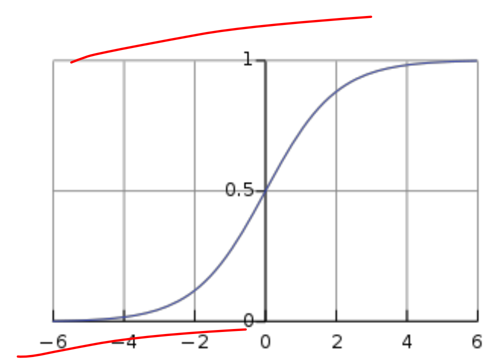
- Most practical application use three-layer neural network (1-1-1: input-hidden-output) to learn patterns. เพราะต่อยาว เพิ่ม hidden เยอะไป
- The most popular learning algorithm is "backpropagation" (Bryson & Ho, 1969).
- Each neuron determines its output Y as the following:

การคำนวณอื่น ๆ ก็แบบนี้

Threshold (Interception)

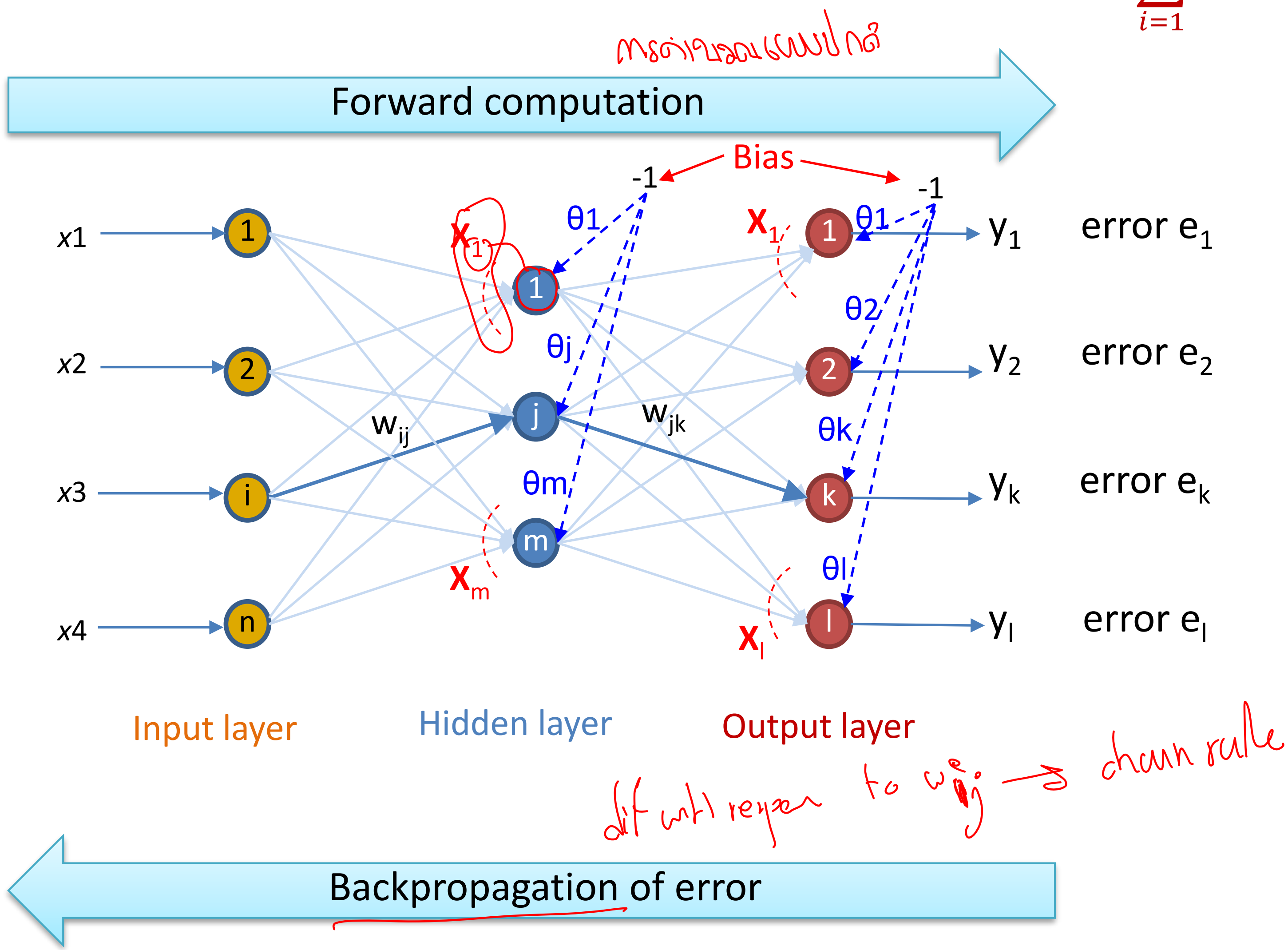
$$X = \sum_{i=1}^{n} x_i w_i - \theta,$$

$$y = \frac{1}{1+e^{-x}} \quad ; 0 < y < 1 \qquad \text{(Sigmoid)}$$



Notations

$$X = \sum_{i=1}^{n} x_i w_i - \theta,$$



Backpropagation for multilayer NN

- To propagate error signals, we start at the output layer and work backward to the hidden layer.

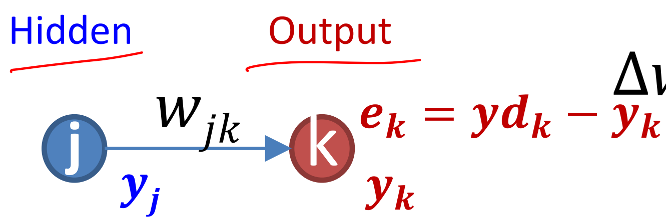
- The error signal at the output of neuron k at iteration p is defined by:

error ต้องการ จริง

$$e_k(p) = yd_k(p) - y_k(p)$$

Where $yd_k(p)$ is the desired output of neuron k at iteration p.

- To update weights, the weight correction (Δw) is adjusted to the previous weights.



$$w_{jk}(p+1) = w_{jk}(p) + \Delta w_{jk}(p),$$

$$\Delta w_{jk}(p) = \alpha . \overset{\text{input}}{y_j(p)} . \delta_k(p),$$ ปรับweight base

gradient ตรงนี้ → diff

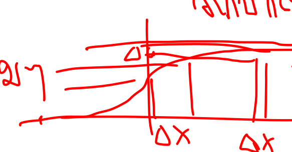
จากไหน error

$$\delta_k(p) = \overset{\text{sigmoid}}{\frac{\partial y_k(p)}{\partial X_k(p)}} . e_k(p),$$ × error

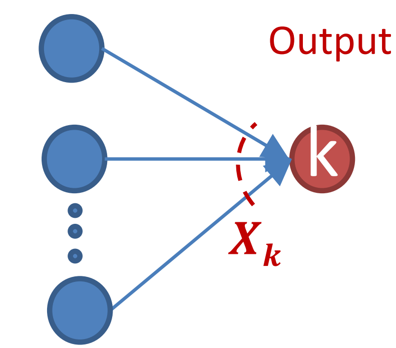
Error gradient at neuron k

หาdiff (sigmoid)

เยอะๆ



ค่าน้อยมากๆเมื่อxไกล



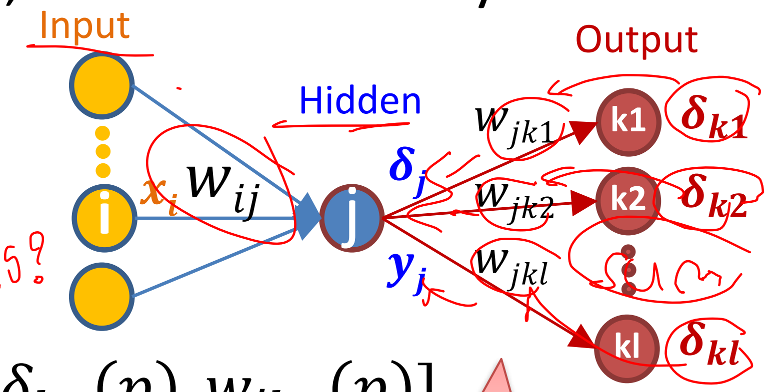
For a sigmoid activation function ($y_k = \frac{1}{1+e^{-X_k}}$),

$$\frac{\partial y_k(p)}{\partial X_k(p)} = \frac{\partial \left[\frac{1}{1+e^{-X_k(p)}}\right]}{\partial X_k(p)} = \frac{e^{-X_k(p)}}{\left(1+e^{-X_k(p)}\right)^2}$$

$$\delta_k(p) = y_k(p).\left(1 - y_k(p)\right).e_k\ (p)$$

sigmoid (1 - sigmoid) . error #

- To update weights in the hidden layer, we do similar way:



$$w_{ij}(p+1) = w_{ij}(p) + \Delta w_{ij}(p),$$

$$\Delta w_{ij}(p) = \alpha . x_i(p) . \delta_j(p),$$

er x_i gradient j ได้มาไร?

$$\delta_j(p) = \underbrace{y_j(p) . \left(1 - y_j(p)\right)}_{\frac{\partial y_j(p)}{\partial X_j(p)}} . \underbrace{\sum_{q=1}^{l} [\delta_{kq}(p) . w_{jkq}(p)]}$$

backpropagation

Where l is the number of neurons in the output layer.

$$y_j(p) = \frac{1}{1+e^{-X_j(p)}},$$

$$X_j(p) = \sum_{i=1}^{n} [x_i(p) . w_{ij}(p)] - \theta_j$$

Complete training algorithm

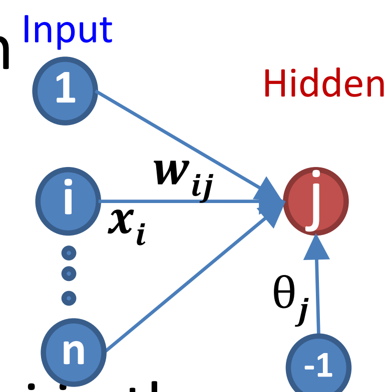
Step1: Weights Initialization

- Set all weights and thresholds (θ) to random number uniformly distributed inside a small range, e.g., (-0.5, +0.5).

compute

Step2: Activation (Feed forward computation)

- Calculate the actual outputs of neurons in the hidden layer.



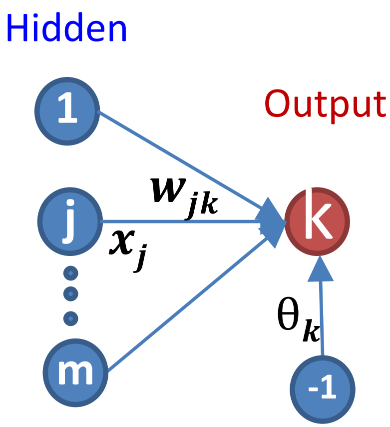
$$y_j(p) = sigmoid\left[\sum_{i=1}^{n}\left(x_{i(p)} . w_{ij(p)}\right) - \theta_j\right]$$

Where n is the number of inputs of neuron j in the hidden layer.

- Calculate the actual outputs of neurons in the output layer.

$$y_k(p) = sigmoid\left[\sum_{j=1}^{m}\left(x_{j(p)}.\,w_{jk(p)}\right) - \theta_k\right]$$

Where m is the number of inputs of neuron k in the output layer.



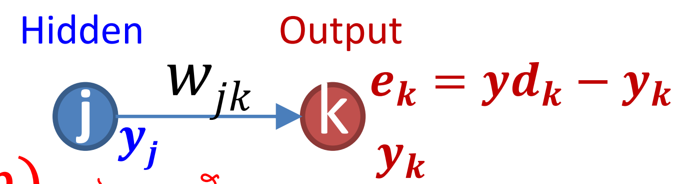
ทำไปเรื่อย ๆ

เริ่มการปรับ

Step3: Weight training (Backpropagation)

- Calculate the error gradient of neurons in the hidden-output layer.

$$e_k(p) = yd_k(p) - y_k(p),$$ ค่าerror



$$\delta_k(p) = y_k(p).\left(1 - y_k(p)\right).e_k(p),$$ ค่าความชัน

$$\Delta w_{jk}(p) = \alpha.y_j(p).\delta_k(p),$$ ตัวที่จะปรับ

$$w_{jk}(p+1) = w_{jk}(p) + \Delta w_{jk}(p)$$ เริ่มปรับ ← อยู่ที่เป็นระยะๆ

backprop ของ state ที่ผ่านมา

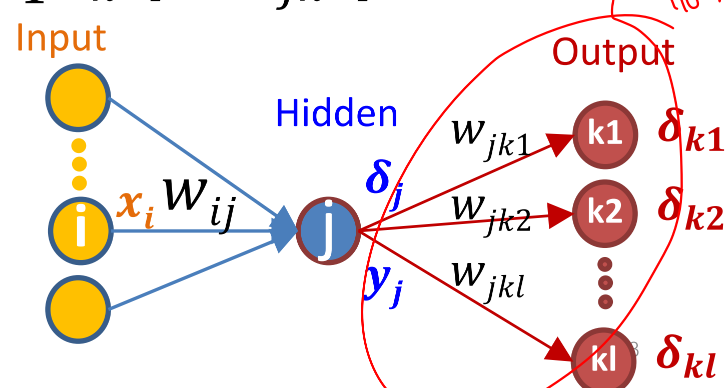
- Calculate the error gradient of neurons in the input-hidden layer.

$$\delta_j(p) = y_j(p).\left(1 - y_j(p)\right).\sum_{k=1}^{l}\delta_k(p).w_{jk}(p),$$ ความชันรวม ถ้าปรับแรง

$$\Delta w_{ij}(p) = \alpha.x_i(p).\delta_j(p),$$ จะปรับ

$$w_{ij}(p+1) = w_{ij}(p) + \Delta w_{ij}(p)$$

เริ่มใหม่



จนครบทุกรอบ

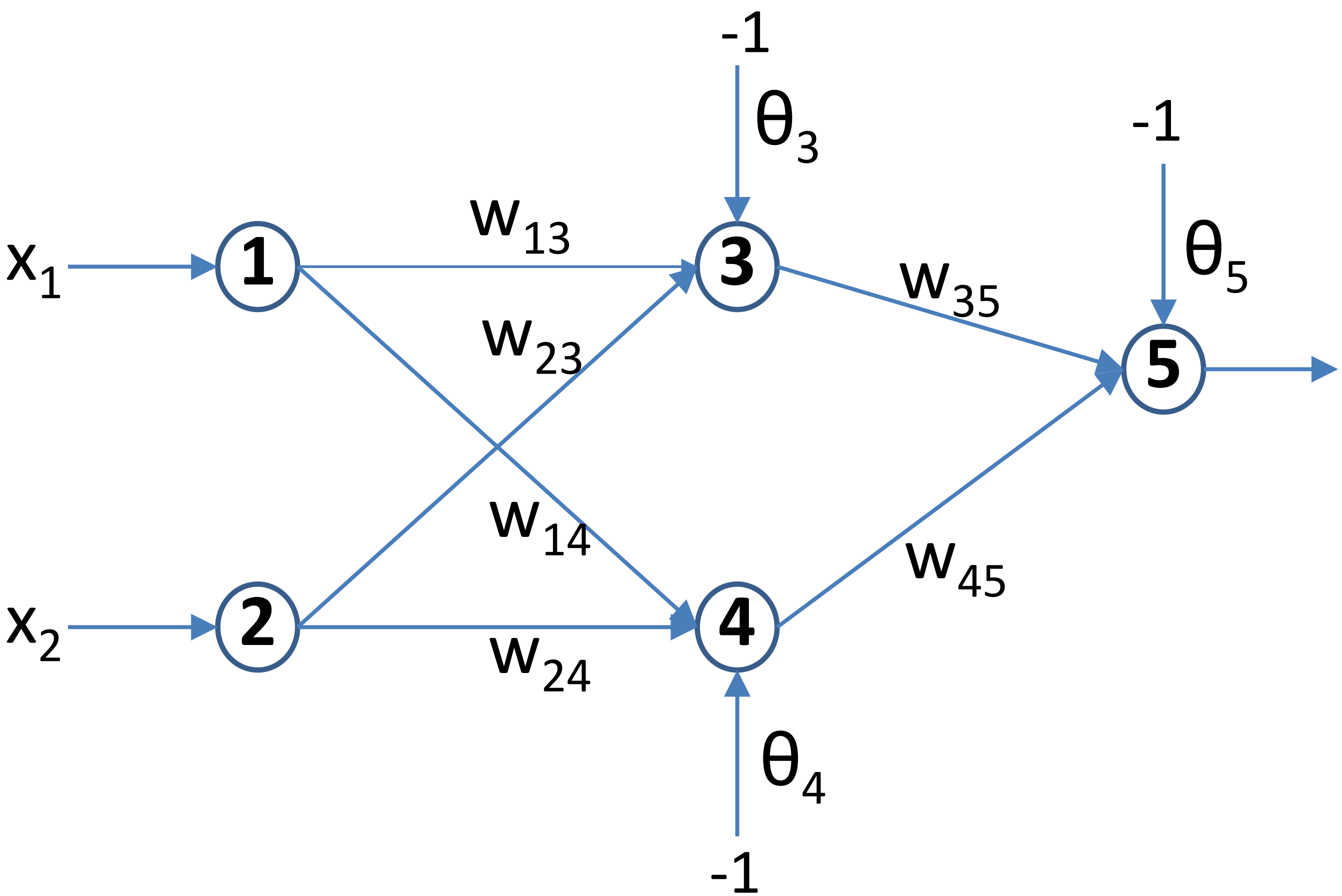
Step4: Iterations

- Take the next training pattern, p+1, and go back to step 2. Then repeat the process until the selected error criterion is satisfy.

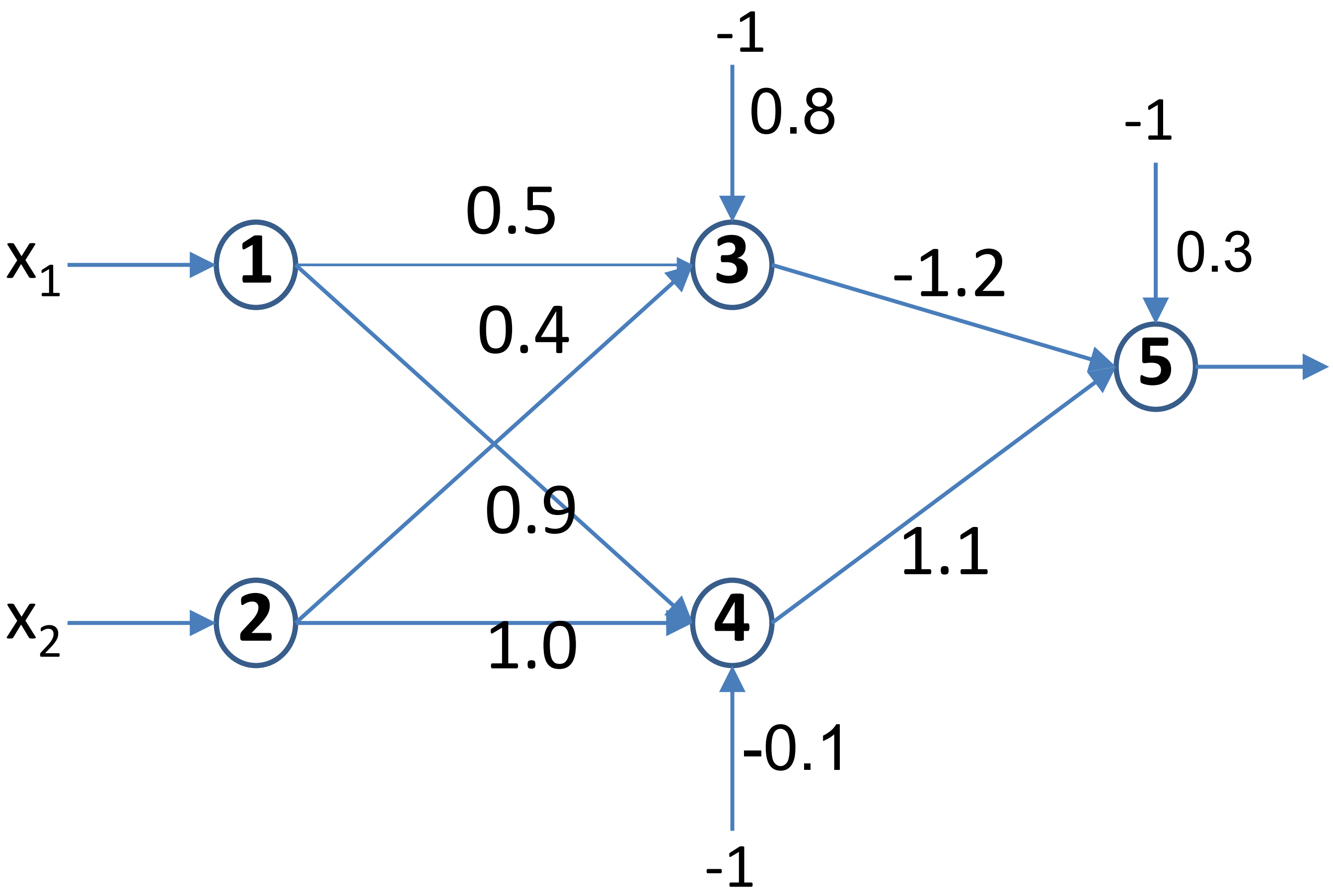
- A simple stop criterion is when the sum-squared error (SSE) less than a certain number, e.g., 0.1.

$$SSE = \sum_{p=1}^{\#patterns} \sum_{k=1}^{\#outputs} [yd_k(p) - y_k(p)]^2$$

Example: 3-layer NN for XOR problem



Recall that a single-layer perceptron cannot solve XOR problem.

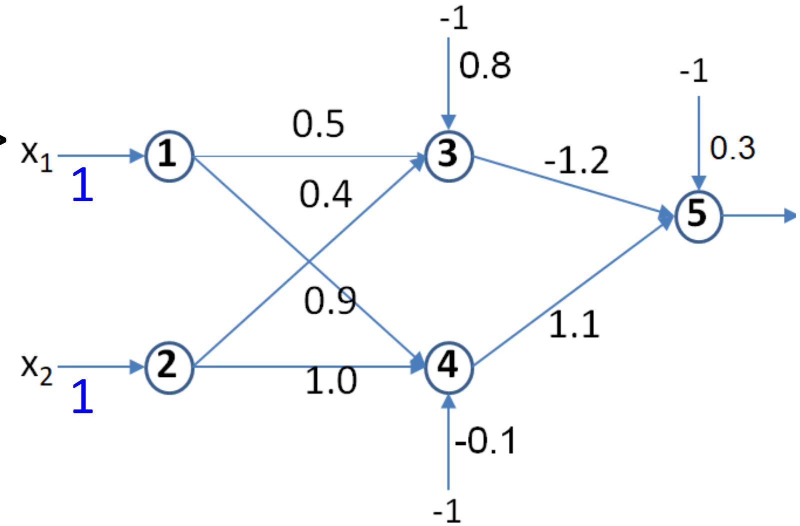


-1
0.8
-1
0.5
0.3
x_1
1
3
-1.2
0.4
5
0.9
1.1
x_2
2
4
1.0
-0.1
-1

All weights and thresholds are randomly initialized as follows:

Parameter	**Initial Value**
w_{13}	0.5
w_{14}	0.9
w_{23}	0.4
w_{24}	1.0
w_{35}	-1.2
w_{45}	1.1
θ_3	0.8
θ_4	-0.1
θ_5	0.3

- Consider the training pattern, <1, 1, 0>



$$y_3 = \frac{1}{\left[1 + e^{-(1*0.5+1*0.4-1*0.8)}\right]} = 0.5250$$

คำนวณไป

$$y_4 = \frac{1}{\left[1 + e^{-(1*0.9+1*1.0+1*0.1)}\right]} = 0.8808$$

$$y_5 = \frac{1}{\left[1 + e^{-(-0.525*1.2+0.8808*1.1-1*0.3)}\right]} = 0.5097$$

ค่าต้องการ   ค่าแท้จริง

$$\mathrm{e} = yd_5 - y_5 = 0 - 0.5097 = -0.5097$$

$$\begin{aligned}\delta_5 &= y_5 \cdot (1 - y_5) \cdot e \\ &= 0.5097 * (1 - 0.5097) * (-0.5097) \\ &= -0.1274\end{aligned}$$

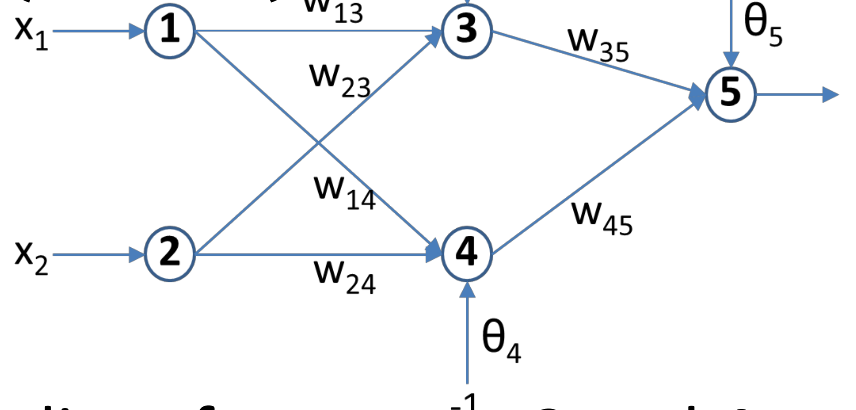
- Assume that the learning rate, α, is equal to 0.1.

$$\Delta w_{35} = \alpha . y_3 . \delta_5 = 0.1 * 0.5250 * (-0.1274) = -\ 0.0067$$

$$\Delta w_{45} = \alpha . y_4 . \delta_5 = 0.1 * 0.8808 * (-0.1274) = -\ 0.0112$$

$$\Delta \theta_5 \ = \alpha . (-1) . \delta_5 = 0.1 * (-1) * (-0.1274)$$

$$= 0.0127$$

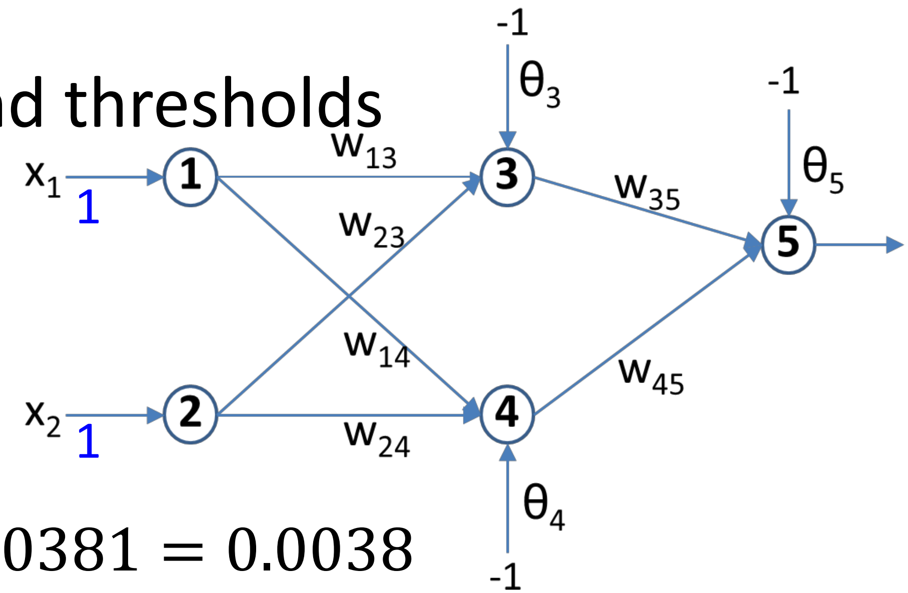


- Next, we calculate the error gradient for neuron 3 and 4.

$$\delta_3 = y_3 . (1 - y_3) . \delta_5 . w_{35}$$
$$= 0.525 * (1 - 0.525) * (-0.1274) * (-1.2) = 0.0381$$

$$\delta_4 = y_4 . (1 - y_4) . \delta_5 . w_{45}$$
$$= 0.8808 * (1 - 0.8808) * (-0.1274) * 1.1 = -0.0147$$

- Calculating Δ of weights and thresholds



$$\Delta w_{13} = \alpha . x_1 . \delta_3 = 0.1 * 1 * 0.0381 = 0.0038$$

$$\Delta w_{23} = \alpha . x_2 . \delta_3 = 0.1 * 1 * 0.0381 = 0.0038$$

$$\Delta \theta_3 = \alpha . (-1) . \delta_3 = 0.1 * (-1) * 0.0381 = -0.0038$$

$$\Delta w_{14} = \alpha . x_1 . \delta_4 = 0.1 * 1 * (-0.0147) = -0.0015$$

$$\Delta w_{24} = \alpha . x_2 . \delta_4 = 0.1 * 1 * (-0.0147) = -0.0015$$

$$\Delta \theta_4 = \alpha . (-1) . \delta_4 = 0.1 * (-1) * (-0.0147) = 0.0015$$

- Lastly, we update all weights and thresholds in the network.

$$w_{13} = w_{13} + \Delta w_{13} = 0.5 + 0.0038 = 0.5038$$

$$w_{14} = w_{14} + \Delta w_{14} = 0.9 - 0.0015 = 0.8985$$

$$w_{23} = w_{23} + \Delta w_{23} = 0.4 + 0.0038 = 0.4038$$

$$w_{24} = w_{24} + \Delta w_{24} = 1.0 - 0.0015 = 0.9985$$

$$w_{35} = w_{35} + \Delta w_{35} = -1.2 - 0.0067 = -1.2067$$

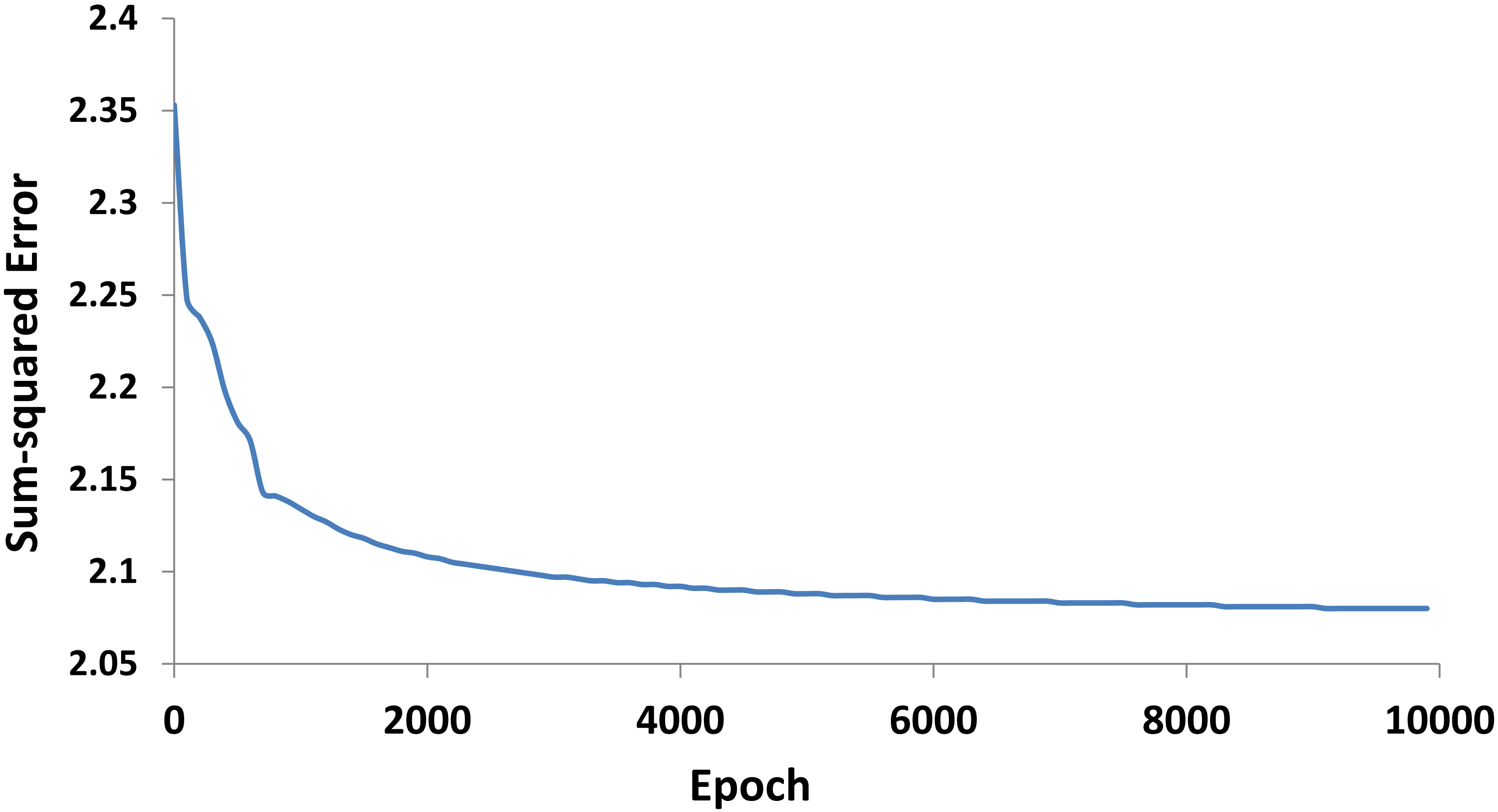
$$w_{45} = w_{45} + \Delta w_{45} = 1.1 - 0.0112 = 1.0888$$

$$\theta_3 = \theta_3 + \Delta\theta_3 = 0.8 - 0.0038 = 0.7962$$

$$\theta_4 = \theta_4 + \Delta\theta_4 = -0.1 + 0.0015 = -0.0985$$

$$\theta_5 = \theta_5 + \Delta\theta_5 = 0.3 + 0.0127 = 0.3127$$

- Repeat the same computation for all training patterns (1 epoch)
- Repeat the process for another epoch until the sum of squared error (SSE) is less than a certain number, e.g. 0.001.



- Sum of squared errors (SSE) of the final network

Input		Desired Output	Actual Output	Error	SSE
x_1	x_2	y_d	y_5	e	
1	1	0	0.0155	-0.0155	0.0010
0	1	1	0.9849	0.0151	
1	0	1	0.9849	0.0151	
0	0	0	0.0175	-0.0175	