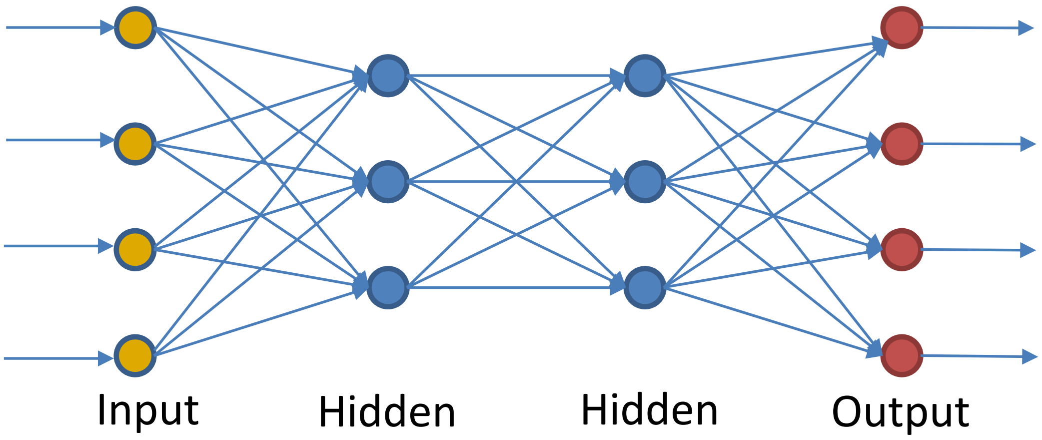
Multilayer Neural Networks

Kietikul Jearanaitanakij

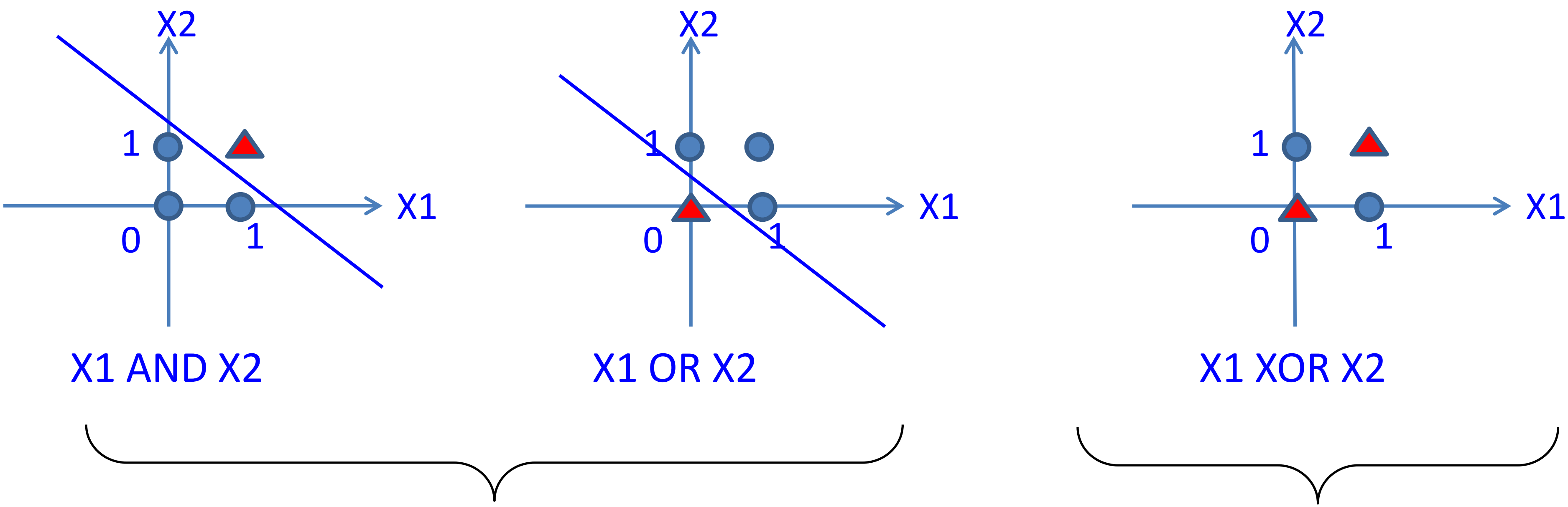
Department of Computer Engineering, KMITL

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**.

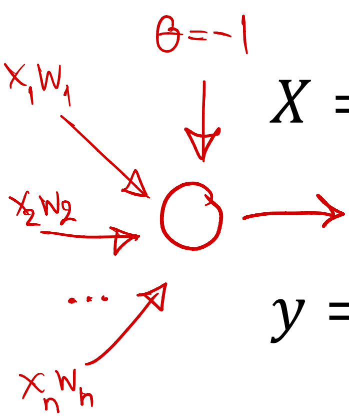
Nonlinearly separable problem



Linearly separable

Non-linearly separable

- With one hidden layer, we can represent the continuous and simple discontinuous functions.
- With two hidden layers, even discontinuous function can be represented.
- Most practical application use three-layer neural network (1-1-1: input-hidden-output) to learn patterns.
- The most popular learning algorithm is "backpropagation" (Bryson & Ho, 1969).
- Each neuron determines its output Y as the following:



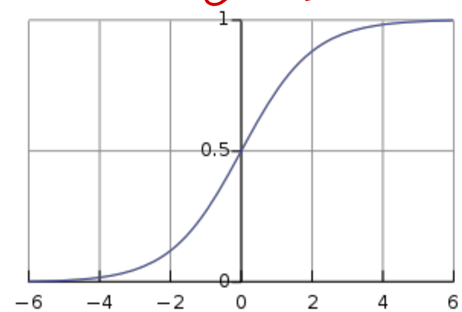
Threshold (Interception)

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

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

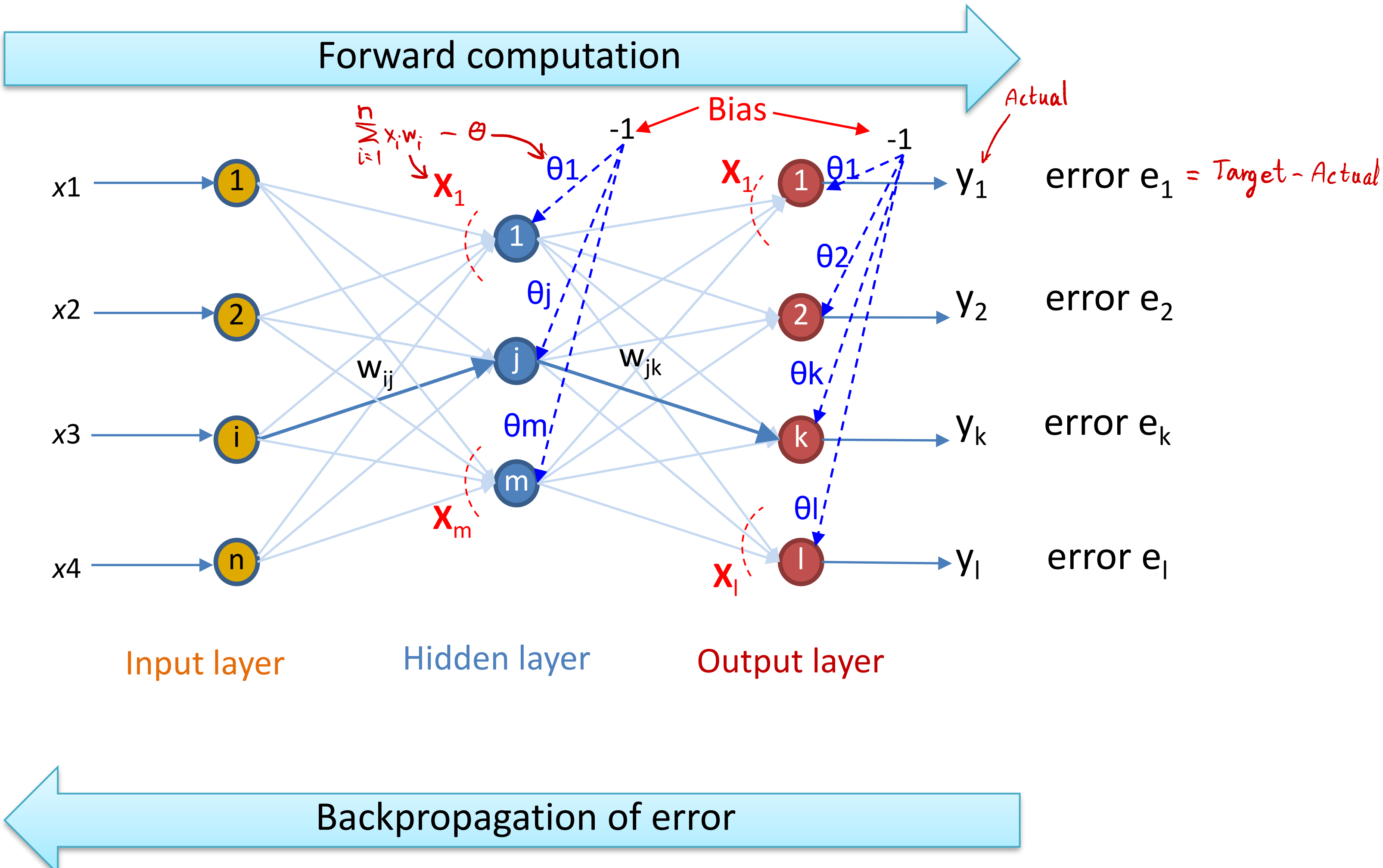
logistic fn

0 ~ 1



Notations

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



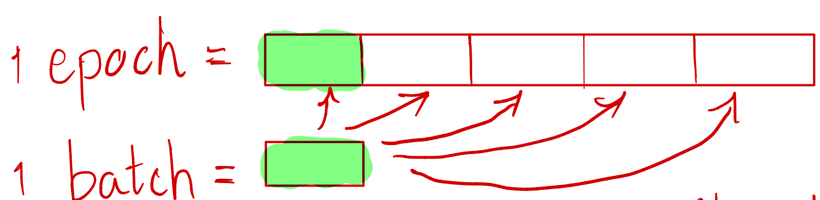
Backpropagation for multilayer NN

- 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: node ที่ k iteration ที่ p

$$e_k(p) = \underbrace{yd_k(p)}_{\text{target}} - \underbrace{y_k(p)}_{\text{actual}}$$

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



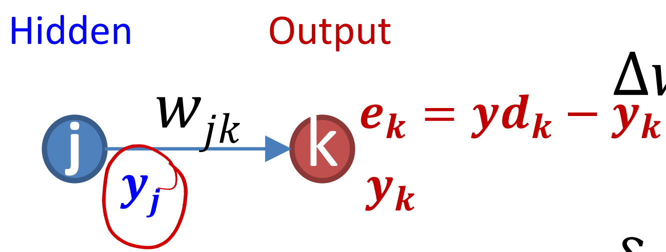
∴ iteration = 5

e = Target − Actual

$\Delta W_1 = \alpha . x_1 . e$

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

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

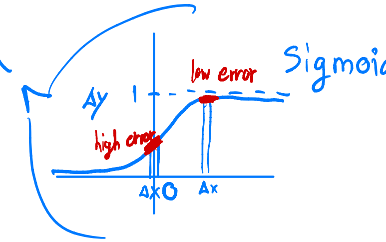


$$\Delta w_{jk}(p) = \alpha . y_j(p) . \delta_k(p),$$ error

$$\delta_k(p) = \frac{\partial y_k(p)}{\partial X_k(p)} . e_k(p),$$

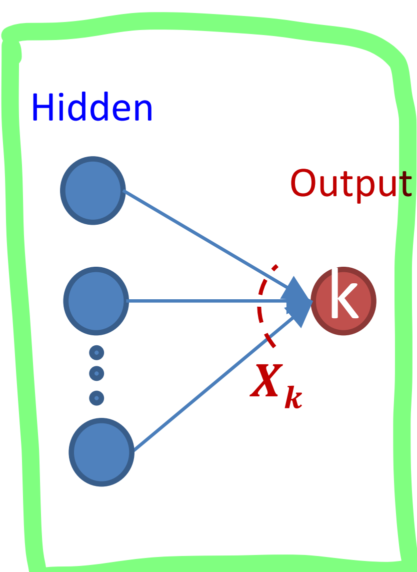
diff sigmoid

slope sigmoid · (Target − Actual)



Error gradient at neuron k

(The reason of derivative will be explained in the later class)

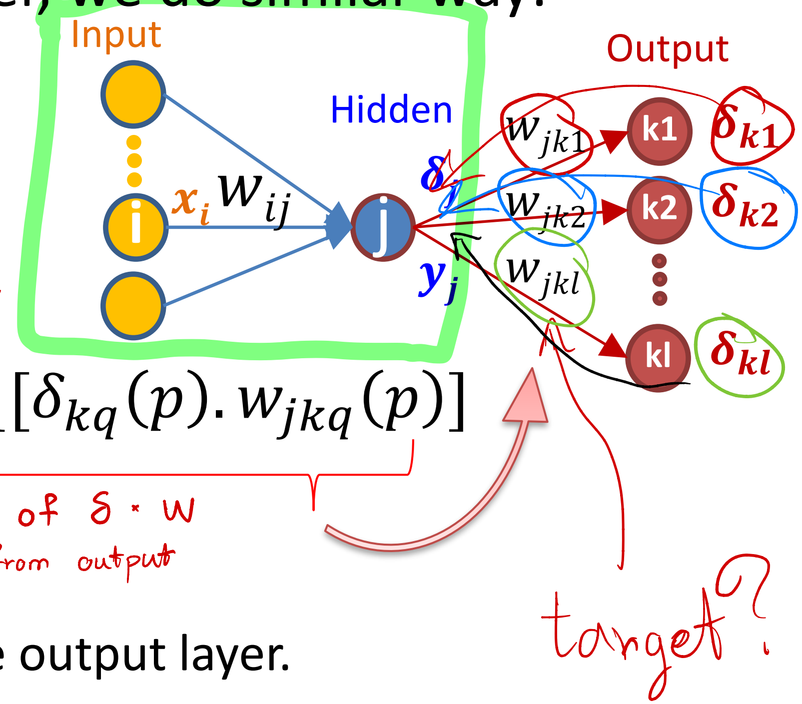


For a sigmoid activation function ($y_k = \frac{1}{1+e^{-X_k}}$),

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

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

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



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

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

$$\delta_j(p) = \underbrace{y_j(p) . \left(1 - y_j(p)\right)}_{\frac{\partial y_j(p)}{\partial X_j(p)}} . \underbrace{\sum_{q=1}^{l} [\delta_{kq}(p) . w_{jkq}(p)]}_{\text{sum of } \delta * w \text{ from output}}$$

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

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

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

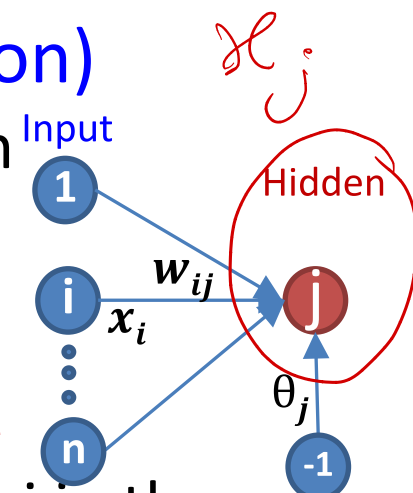
Complete training algorithm

Step1: Weights Initialization

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

Step2: Activation (Feed forward computation)

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



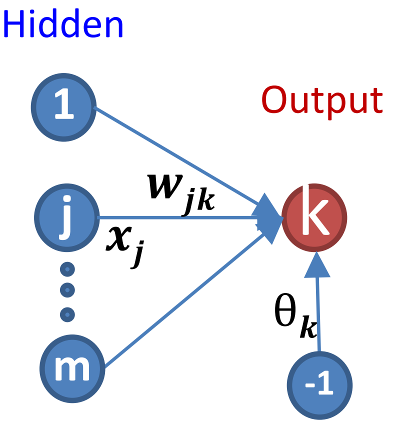
$$y_j(p) = \overset{\frac{1}{1+e^{-x_j}}}{sigmoid} \underbrace{\left[\sum_{i=1}^{n}\left(x_{i(p)}.w_{ij(p)}\right) - \theta_j\right]}_{x_j}$$

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.



Step3: Weight training (Backpropagation)

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

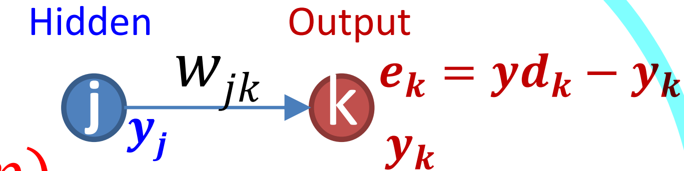
$$e_k(p) = yd_k(p) - y_k(p),$$

diff sigmoid

$$\delta_k(p) = \boxed{y_k(p).\left(1 - y_k(p)\right)}.\, e_k(p),$$

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

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



ต้องใช้ $w_{jk}(p)$ ในการ backpropagate

wait ⇒

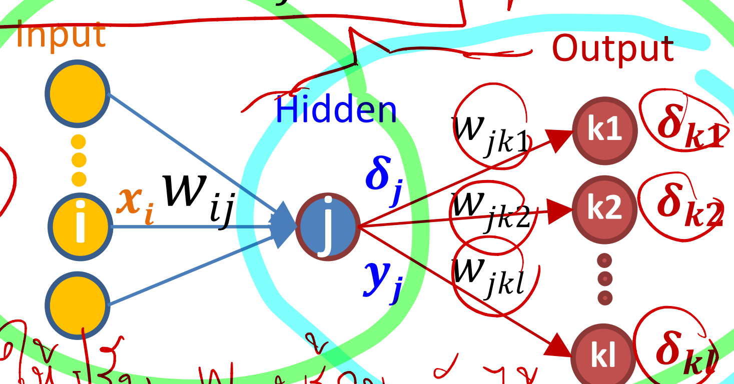
จน backpropagate ไปถึง input layer ก่อน

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

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

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

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



∴ ต้องรอ → พอถึง layer input ให้ปรับ w พร้อมๆกันได้เลย

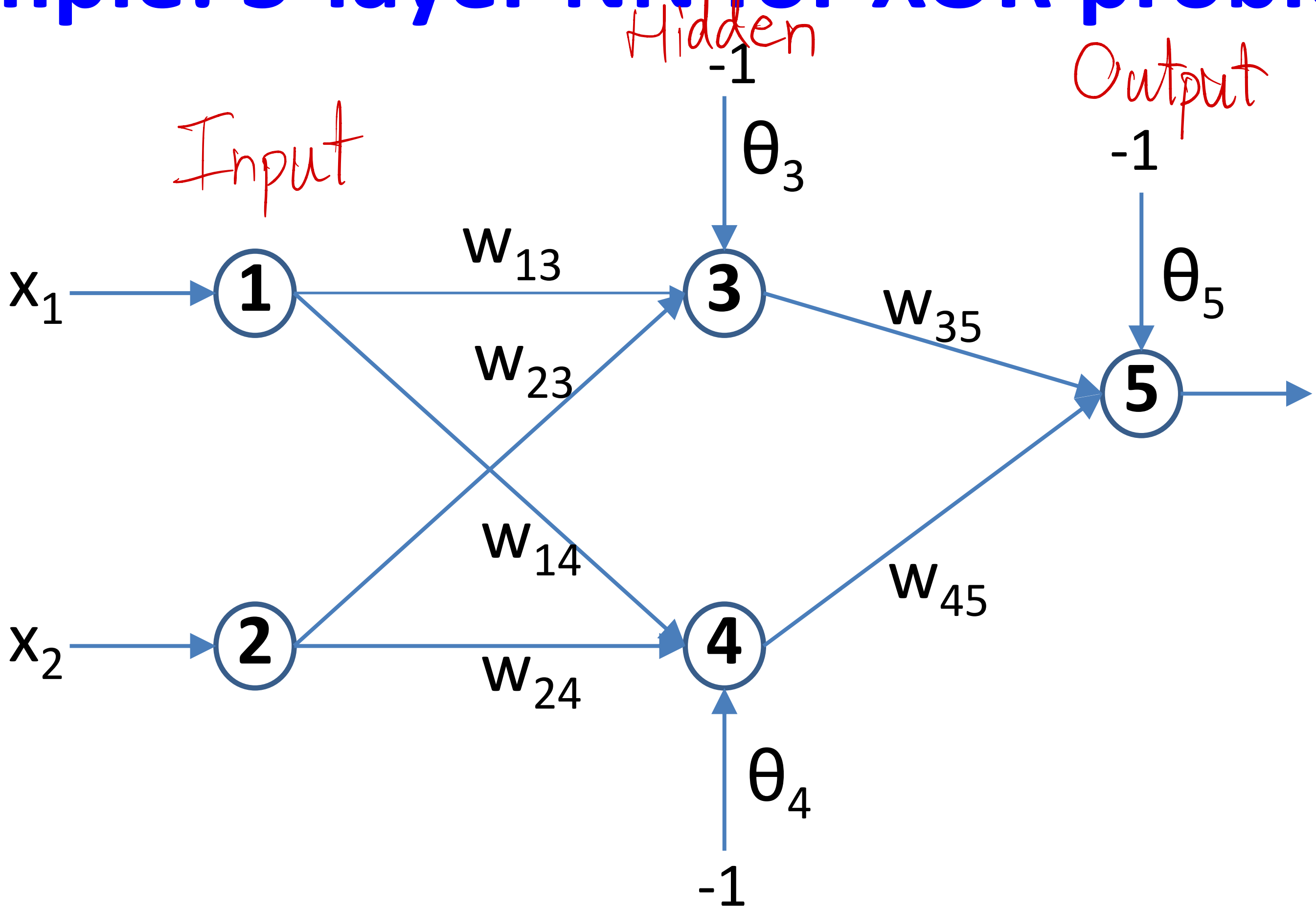
Step4: Iterations

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

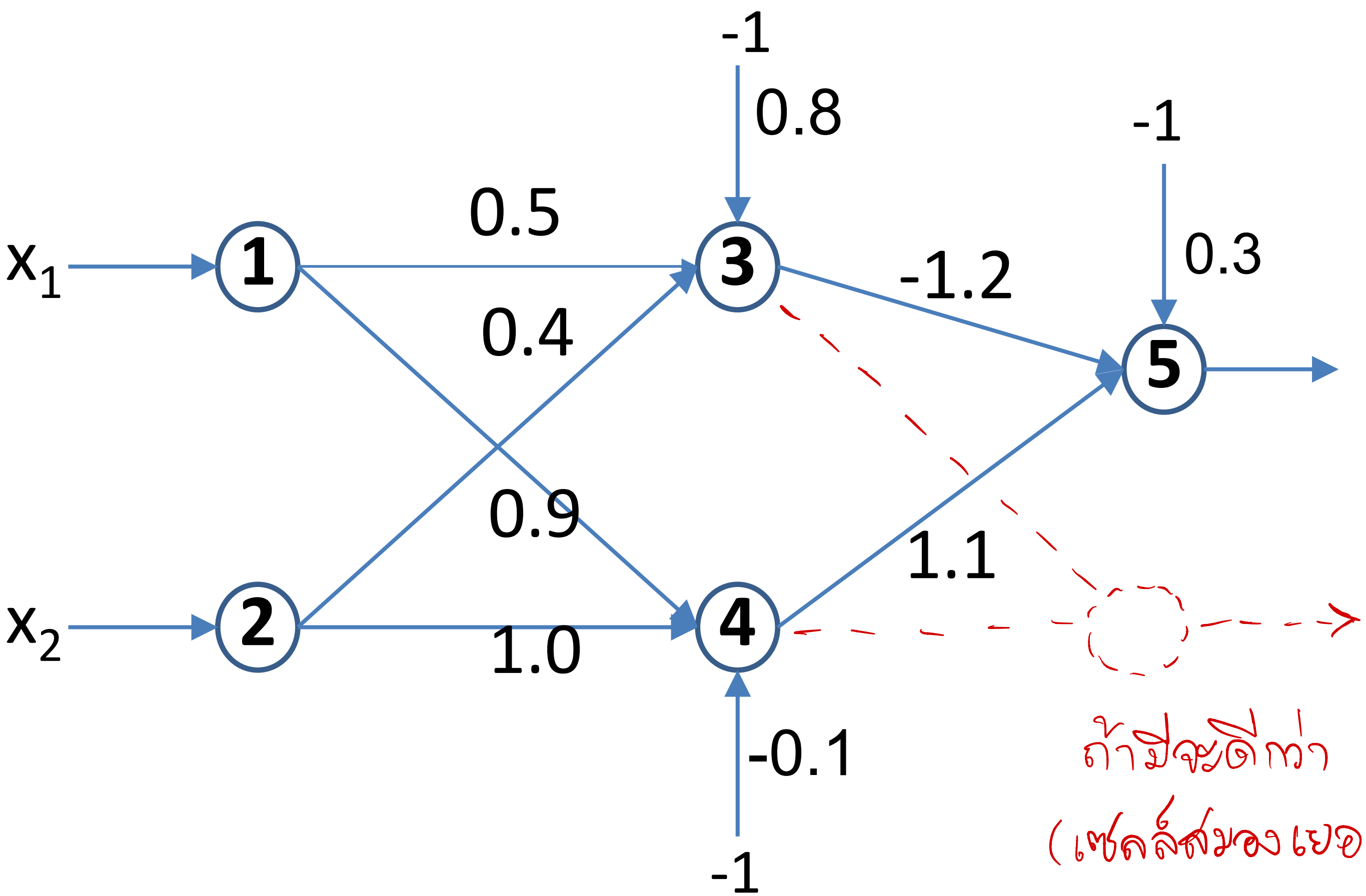
- A simple stop criterion is when the sum-squared error (SSE) less than a certain number, e.g., 0.1.

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

Example: 3-layer NN for XOR problem



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



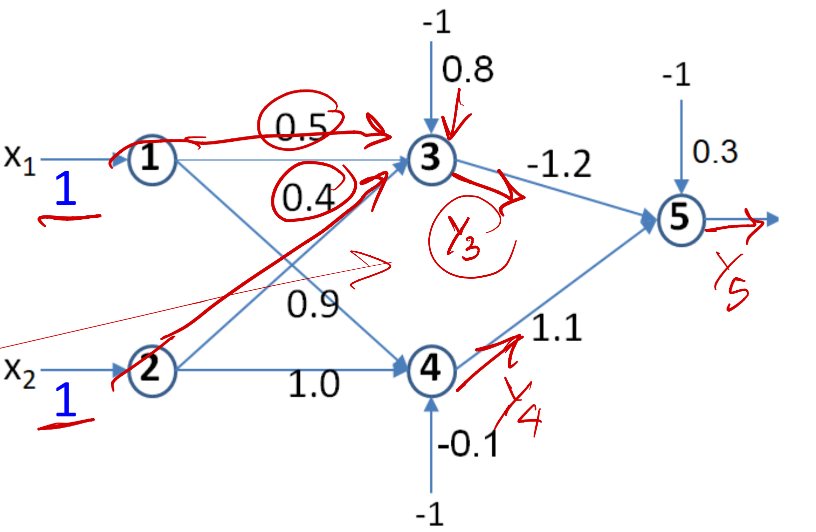
ถ้ามีจะดีกว่า
(เซลล์สมองเยอะกว่า)

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>

$\underbrace{1,\ 1}$ 1 XOR 1 = 0



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

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

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

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

(gradient)

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

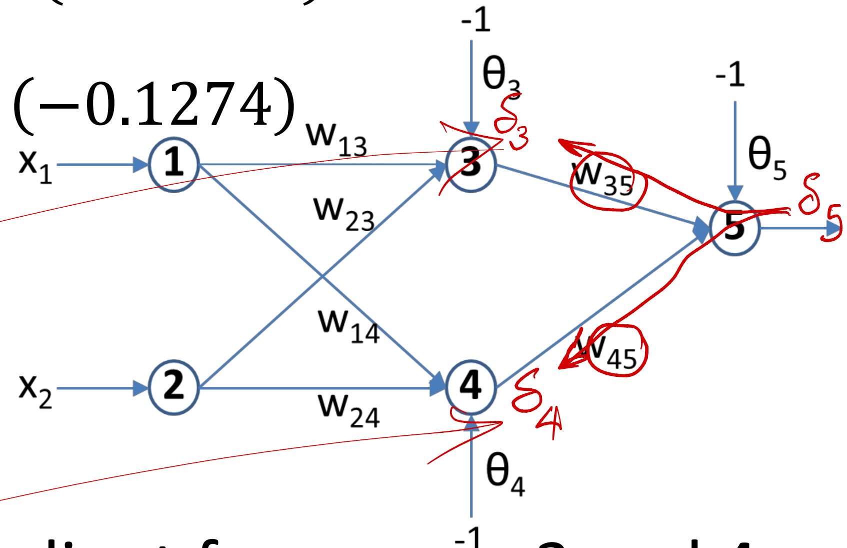
- Assume that the learning rate, α, is equal to 0.1.

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

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

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

$$= 0.0127$$



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

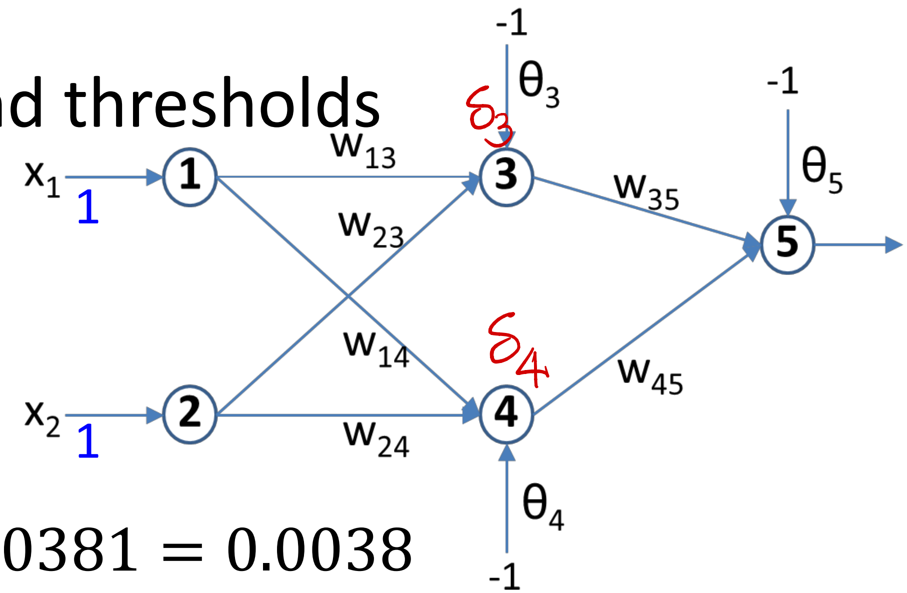
$$\delta_3 = \overbrace{y_3 . (1 - y_3)}^{\text{diff sigmoid (slope sigmoid)}} . \delta_5 . w_{35}$$

$$= 0.525 * (1 - 0.525) * (-0.1274) * (-1.2) = 0.0381$$

$$\delta_4 = y_4 . (1 - y_4) . \delta_5 . w_{45}$$

$$= 0.8808 * (1 - 0.8808) * (-0.1274) * 1.1 = -0.0147$$

- 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.

$$w_{13} = w_{13} - \Delta w_{13} = 0.5 - 0.0038 = 0.4962$$

$$w_{14} = w_{14} - \Delta w_{14} = 0.9 + 0.0015 = 0.9015$$

$$w_{23} = w_{23} - \Delta w_{23} = 0.4 - 0.0038 = 0.3962$$

$$w_{24} = w_{24} - \Delta w_{24} = 1.0 + 0.0015 = 1.0015$$

$$w_{35} = w_{35} - \Delta w_{35} = -1.2 + 0.0067 = -1.1933$$

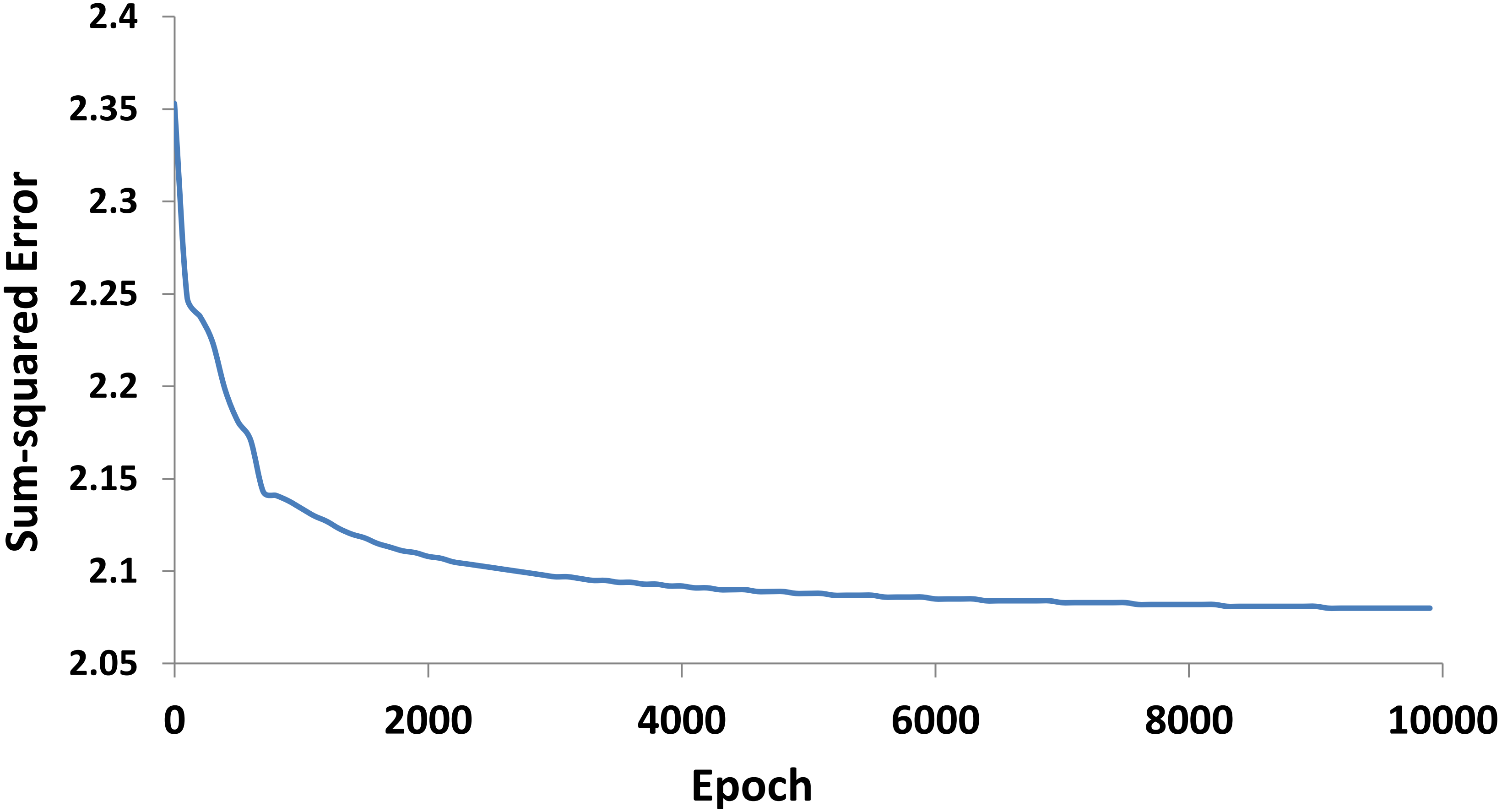
$$w_{45} = w_{45} - \Delta w_{45} = 1.1 + 0.0112 = 1.1112$$

$$\theta_3 = \theta_3 - \Delta\theta_3 = 0.8 + 0.0038 = 0.8038$$

$$\theta_4 = \theta_4 - \Delta\theta_4 = -0.1 - 0.0015 = -0.1015$$

$$\theta_5 = \theta_5 - \Delta\theta_5 = 0.3 - 0.0127 = 0.2873$$

- 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

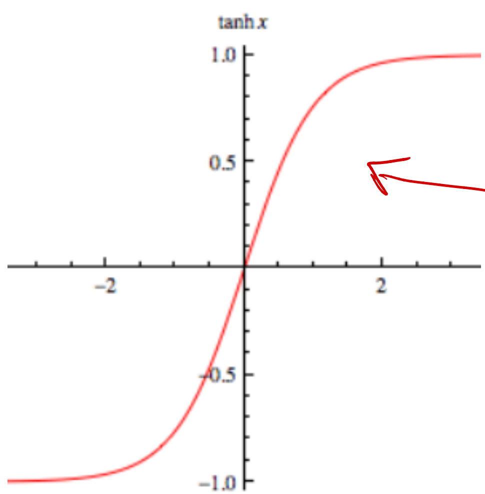
Input		Desired Output	Actual Output	Error	SSE
x_1	x_2	y_d	y_5	e	
1	1	0	0.0155	-0.0155	0.0010
0	1	1	0.9849	0.0151	
1	0	1	0.9849	0.0151	
0	0	0	0.0175	-0.0175	

Techniques for improving multilayer NN

- Using a steeper activation function, i.e. tanh (hyperbolic tangent function), to accelerate the convergence.

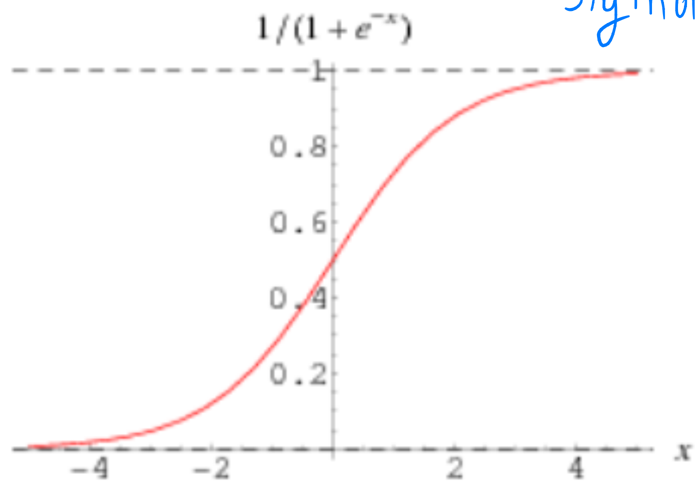
$$\tanh(x) = \frac{\sinh(x)}{\cosh(x)}$$

$$= \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}} = \frac{e^{2x}-1}{e^{2x}+1}$$



ชันกว่า

Sigmoid



Techniques for improving multilayer NN

- Momentum

The momentum term can reduce the local minima and smooth the variation of the output value.

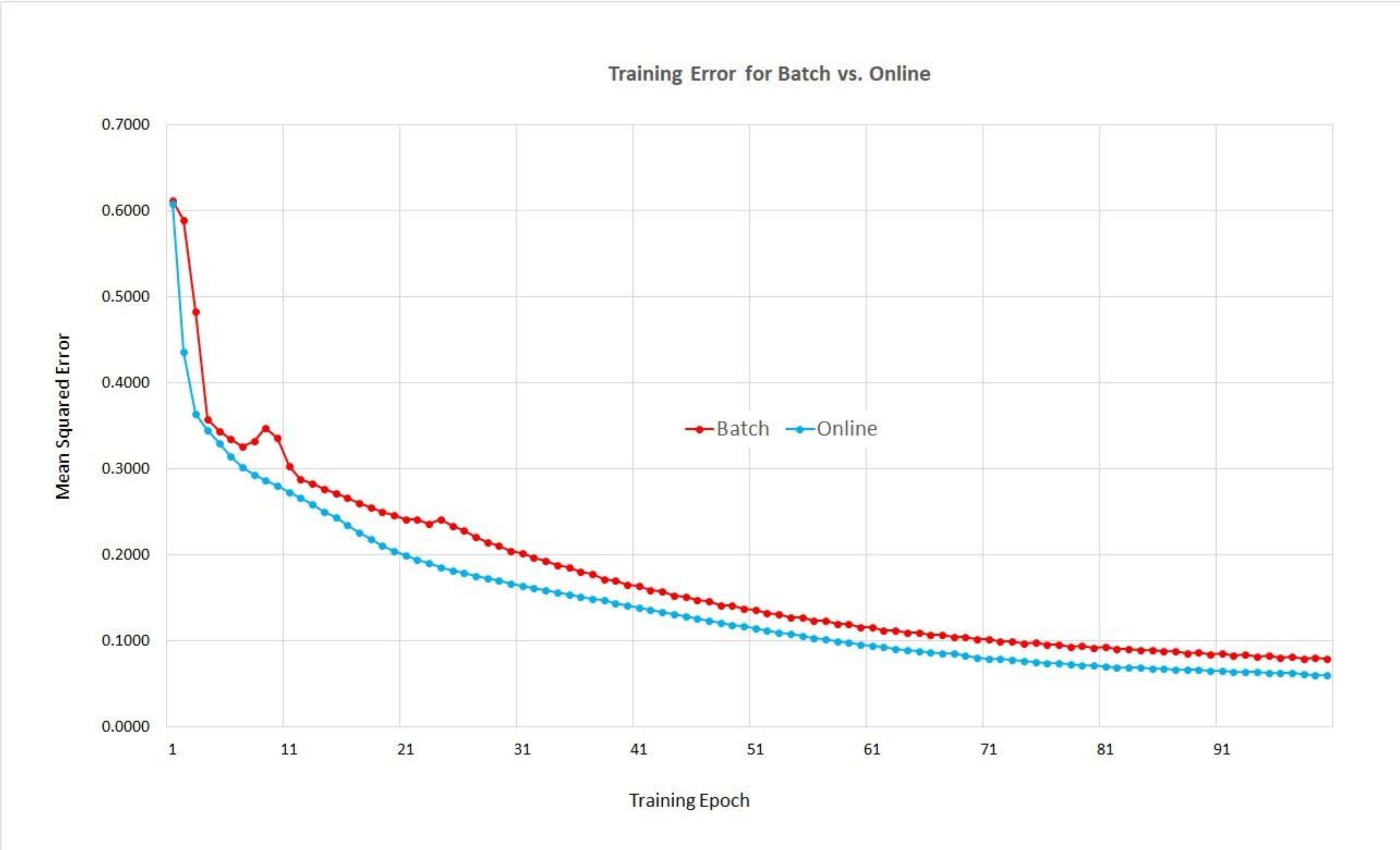
$$\Delta w_{jk} = \boldsymbol{\beta} \Delta w_{jk} - \propto . y_j . \delta_k$$

Where β is a positive number $(0 \leq \beta \leq 1)$ called the momentum constant.

If the momentum term is large then the learning rate should be kept smaller. Otherwise, you might skip the minimum spot with a huge step.

Techniques for improving multilayer NN

- Batch learning ต.ย. XOR ข้างบนนั่นแหละ
  - In **online training**, weights and bias values are adjusted for every training item based on the difference between computed outputs and the training data target outputs.
  - In **batch training,** the adjustment delta values are accumulated over all training items, to give an aggregate set of deltas, and then the aggregated deltas are applied to each weight and bias. $W_{ij}(p+1) = W_{ij}(p) + \Delta W$ $\Delta W_1 + \Delta W_2 + \ldots \Delta W_n$ iteration
  - **Online update** = **Batch updates** x No. of samples in training set
  - The batch algorithm is also slightly more efficient in terms of number of computations.


Training Error for Batch vs. Online
Mean Squared Error
0.7000
0.6000
0.5000
0.4000
0.3000
0.2000
0.1000
0.0000
Batch
Online
1
11
21
31
41
51
61
71
81
91
Training Epoch