Dr. Qadri Hamarsheh

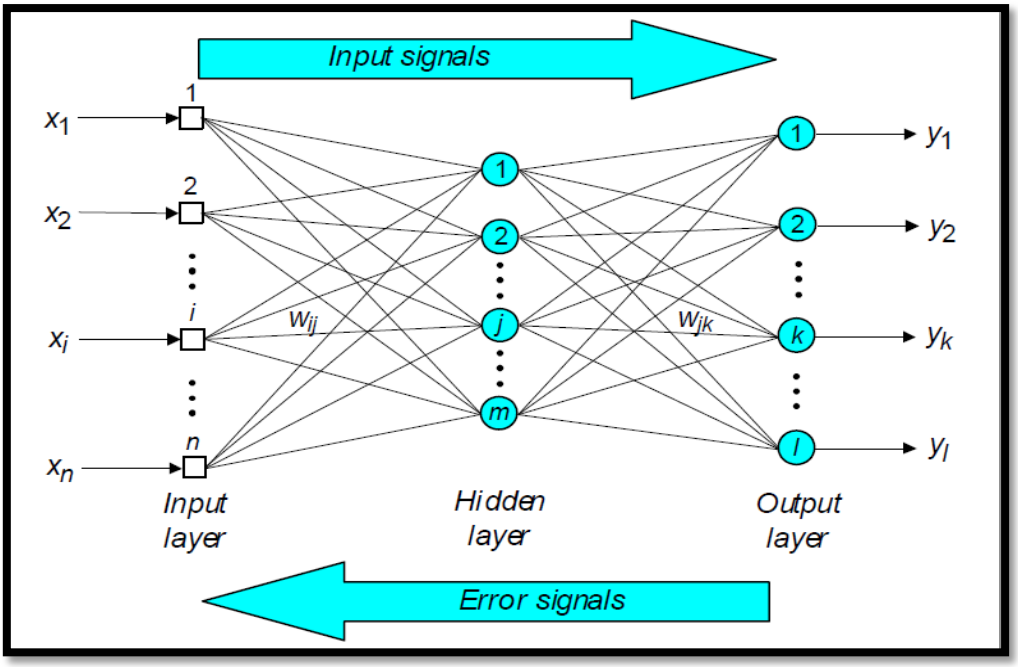
Neural Networks and Fuzzy Logic (630514)

Lecture 9

Supervised Learning in Neural Networks (Part 2)

Multilayer neural networks (back-propagation training algorithm)

- The input signals are **propagated** in a **forward direction** on a layer-by-layer basis.
- Learning in a multilayer network proceeds the same way as for a perceptron.
- A training set of input patterns is presented to the network.
- The network computes its output pattern, and if there is an error - or in other words a difference between actual and desired output patterns - the weights are adjusted to reduce this error.
- In a back-propagation neural network, the learning algorithm has two phases.
  - First, a training input pattern is presented to the network input layer. The network propagates the input pattern from layer to layer until the output pattern is generated by the output layer.
  - Second, if this pattern is different from the desired output, an error is calculated and then propagated backwards through the network from the output layer to the input layer. The weights are modified as the error is propagated.



back-propagation neural network

The back-propagation training algorithm

- Backpropagation is a common method for training a neural network, the goal of backpropagation is to **optimize** the weights so that the neural network can learn how to ***correctly map arbitrary inputs to outputs***.
- Backpropagation method contains the following steps:

Step 1: Initialization; set all the weights and threshold levels of the network to random numbers uniformly distributed inside a range:

$$\left(-\frac{2.4}{F_i}, \quad +\frac{2.4}{F_i}\right)$$

Where $\mathbf{F_i}$ is the total number of inputs of neuron **i** in the network.

Step 2: Activation; activate the back-propagation neural network by applying inputs $\mathbf{x_1(p), x_2(p), \dots, x_n(p)}$ and desired outputs $\mathbf{y_{d,1}(p), y_{d,2}(p), \dots, y_{d,n}(p)}$ (**forward pass**).

- **Calculate** the actual outputs of the neurons in the **hidden layer**:

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

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

- **Calculate** the actual outputs of the neurons in the **output layer**:

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

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

Step 3: Weight training (back-propagate)

- Update the weights in the back-propagation network propagating backward the errors associated with output neurons.
  - **Calculate** the **error gradient** for the neurons in the **output layer:**

$$\delta_k(p) = y_k(p) \cdot [1 - y_k(p)] \cdot e_k(p)$$

Where

$$e_k(p) = y_{d,k}(p) - y_k(p)$$

  - **Calculate** the **weight corrections:**

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

  - **Update** the **weights** at the **output neurons:**

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

- Calculate the **error gradient** for the neurons in the **hidden layer**:

$$\delta_j(p) = y_j(p) \cdot [1 - y_j(p)] \cdot \sum_{k=1}^{l} \delta_k(p)\, w_{jk}(p)$$

- **Calculate** the **weight corrections:**

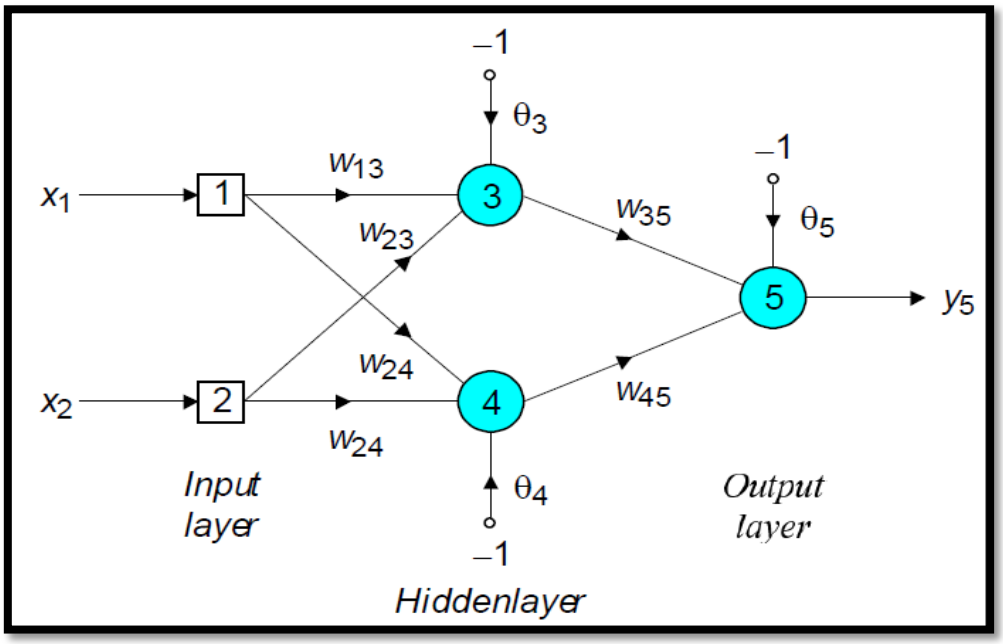
$$\Delta w_{ij}(p) = \alpha \cdot x_i(p) \cdot \delta_j(p)$$

- **Update** the **weights** at the **hidden neurons:**

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

Step 4: Iteration; increase iteration **p** by one, go back to **Step 2** and repeat the process until the selected error criterion is satisfied.

A Step by Step Backpropagation Example: Exclusive-OR.



- The initial weights and threshold levels are set **randomly** as follows:
  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 and θ_5 = 0.3.
- We consider a training set where inputs **x_1** and **x_2** are equal to **1** and desired output **$y_{d,5}$** is **0**.
- The **actual outputs** of neurons **3** and **4** in the hidden layer are calculated as

$$y_3 = sigmoid(x_1 w_{13} + x_2 w_{23} - \theta_3) = 1/\left[1 + e^{-(1\cdot 0.5 + 1\cdot 0.4 - 1\cdot 0.8)}\right] = 0.5250$$

$$y_4 = sigmoid(x_1 w_{14} + x_2 w_{24} - \theta_4) = 1/\left[1 + e^{-(1\cdot 0.9 + 1\cdot 1.0 + 1\cdot 0.1)}\right] = 0.8808$$

- Now the **actual output** of neuron **5** in the output layer is determined as:

$$y_5 = sigmoid(y_3 w_{35} + y_4 w_{45} - \theta_5) = 1/\left[1 + e^{-(-0.5250\,1.2 + 0.8808\,1.1 - 1\cdot 0.3)}\right] = 0.5097$$

- Thus, the following **error** is obtained:

$$e = y_{d,5} - y_5 = 0 - 0.5097 = -0.5097$$

- The next step is **weight training**. To update the weights and threshold levels in our network, we propagate the error, **e**, from the output layer backward to the input layer.
- First, we **calculate** the **error gradient** for neuron **5** in the output layer:

$$\delta_5 = y_5 (1 - y_5)\, e = 0.5097 \cdot (1 - 0.5097) \cdot (-0.5097) = -0.1274$$

- Then we determine the **weight corrections** assuming that the learning rate parameter, α, is equal to **0.1**:

$$\Delta w_{35} = \alpha \cdot y_3 \cdot \delta_5 = 0.1 \cdot 0.5250 \cdot (-0.1274) = -0.0067$$
$$\Delta w_{45} = \alpha \cdot y_4 \cdot \delta_5 = 0.1 \cdot 0.8808 \cdot (-0.1274) = -0.0112$$
$$\Delta \theta_5 = \alpha \cdot (-1) \cdot \delta_5 = 0.1 \cdot (-1) \cdot (-0.1274) = -0.0127$$

- Next we calculate the **error gradients** for neurons **3** and **4** in the hidden layer:

$$\delta_3 = y_3 (1 - y_3) \cdot \delta_5 \cdot w_{35} = 0.5250 \cdot (1 - 0.5250) \cdot (-0.1274) \cdot (-1.2) = 0.0381$$
$$\delta_4 = y_4 (1 - y_4) \cdot \delta_5 \cdot w_{45} = 0.8808 \cdot (1 - 0.8808) \cdot (-0.1274) \cdot 1.1 = -0.0147$$

- We then determine the **weight corrections**:

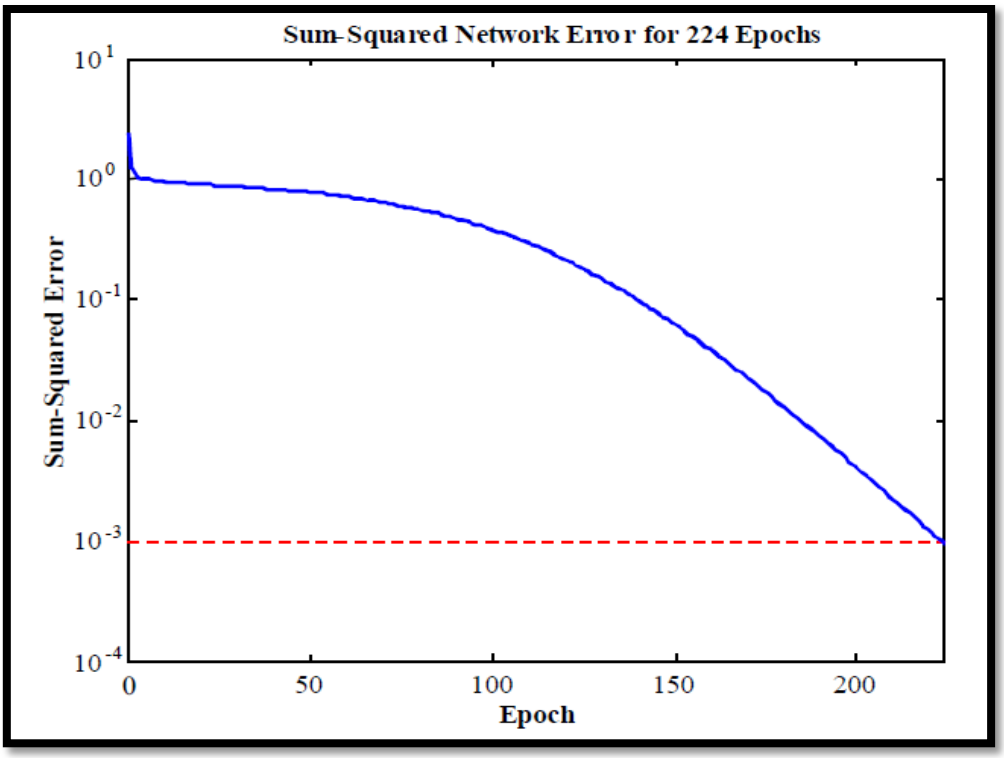
$$\Delta w_{13} = \alpha \cdot x_1 \cdot \delta_3 = 0.1 \cdot 1 \cdot 0.0381 = 0.0038$$
$$\Delta w_{23} = \alpha \cdot x_2 \cdot \delta_3 = 0.1 \cdot 1 \cdot 0.0381 = 0.0038$$
$$\Delta \theta_3 = \alpha \cdot (-1) \cdot \delta_3 = 0.1 \cdot (-1) \cdot 0.0381 = -0.0038$$
$$\Delta w_{14} = \alpha \cdot x_1 \cdot \delta_4 = 0.1 \cdot 1 \cdot (-0.0147) = -0.0015$$
$$\Delta w_{24} = \alpha \cdot x_2 \cdot \delta_4 = 0.1 \cdot 1 \cdot (-0.0147) = -0.0015$$
$$\Delta \theta_4 = \alpha \cdot (-1) \cdot \delta_4 = 0.1 \cdot (-1) \cdot (-0.0147) = 0.0015$$

- At last, we **update** all weights and threshold:

$$w_{13} = w_{13} + \Delta w_{13} = 0.5 + 0.0038 = 0.5038$$
$$w_{14} = w_{14} + \Delta w_{14} = 0.9 - 0.0015 = 0.8985$$
$$w_{23} = w_{23} + \Delta w_{23} = 0.4 + 0.0038 = 0.4038$$
$$w_{24} = w_{24} + \Delta w_{24} = 1.0 - 0.0015 = 0.9985$$
$$w_{35} = w_{35} + \Delta w_{35} = -1.2 - 0.0067 = -1.2067$$
$$w_{45} = w_{45} + \Delta w_{45} = 1.1 - 0.0112 = 1.0888$$
$$\theta_3 = \theta_3 + \Delta \theta_3 = 0.8 - 0.0038 = 0.7962$$
$$\theta_4 = \theta_4 + \Delta \theta_4 = -0.1 + 0.0015 = -0.0985$$
$$\theta_5 = \theta_5 + \Delta \theta_5 = 0.3 + 0.0127 = 0.3127$$

- The training process is **repeated** until the **sum of squared errors** is less than **0.001**.

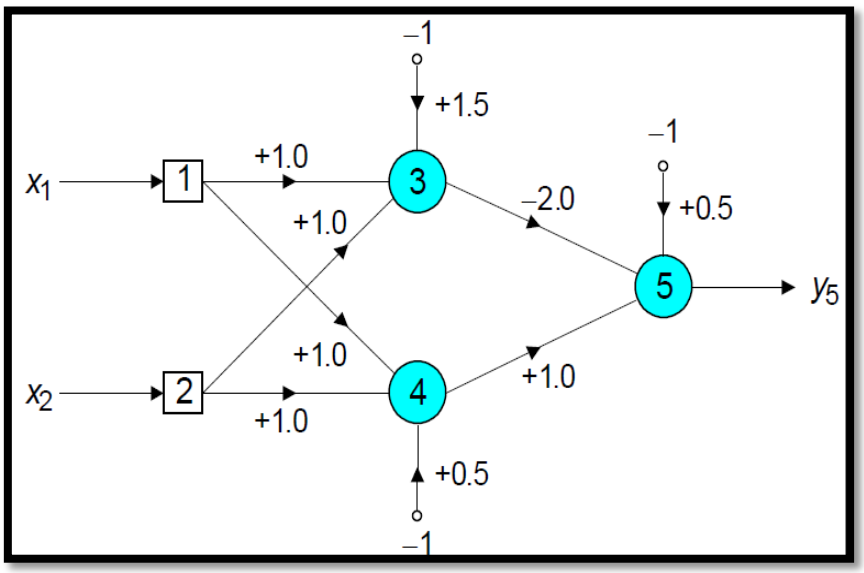
- **Learning curve** for operation ***Exclusive-OR***



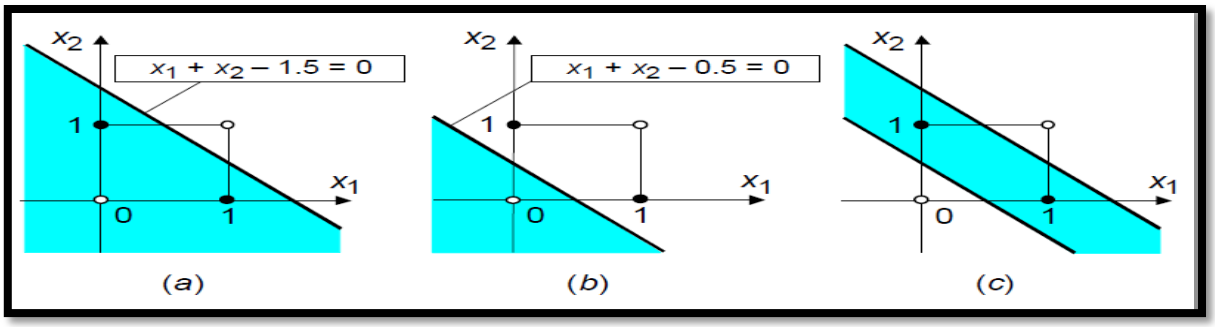
- **Final results** of network learning:

Inputs		Desired output	Actual output	Error	Sum of squared errors
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	

- Network represented by McCulloch-Pitts model for solving the Exclusive-OR operation.



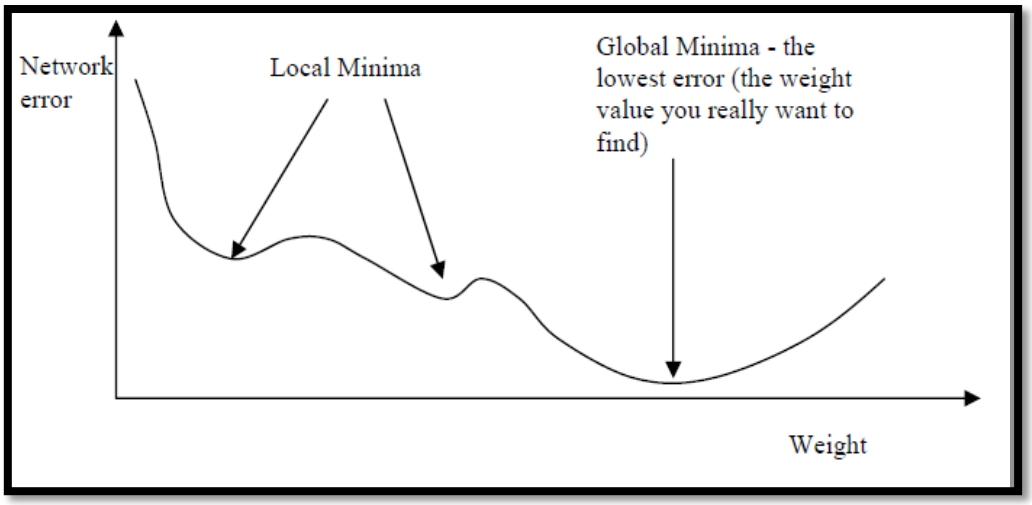
- **Decision boundaries**



(a) Decision boundary constructed by hidden neuron 3;
(b) Decision boundary constructed by hidden neuron 4;
(c) Decision boundaries constructed by complete network

Problems with Backpropagation

1) "**Local Minima**": This occurs because the algorithm always changes the weights in such a way as to cause the **error to decrease**. But the error might briefly have to **increase**, as shown in figure. If this is the case, the algorithm will "**gets stuck**" and the error will not decrease further.



- **Solutions to this problem:**
  - **Reset the weights to different random numbers and train again.**
  - **Add "momentum" to the weight correction: weight correction depends not just on the current error, but also on previous changes, For example**

W^+ =W+Current change+(Change on previous iteration * constant)
Constant is < 1.

$$\Delta w_{jk}(p) = \beta \cdot \Delta w_{jk}(p-1) + \alpha \cdot y_j(p) \cdot \delta_k(p)$$

$0 \le \beta < 1$; Typically $\beta = 0.95$

This equation is called the **generalized delta rule**.

Learning with momentum for operation Exclusive-OR: **126 Epochs**

2) Biological neurons ***do not work backward*** to adjust the synaptic weights, so Backpropagation Algorithm ***can't emulate brain-like learning.***

3) In Backpropagation Algorithm, calculations are extensive (***training is slow***).

- **Solutions to this problem:**
  - **Use the sigmoidal activation function (hyperbolic tangent).**

$$Y^{tanh} = \frac{2a}{1+e^{-bX}} - a$$

where a and b are constants.

  - **Include a momentum term.**
  - **Adjust the learning rate parameter during training (not constant).**

Learning with adaptive learning rate: **103 Epochs.**

  - **Learning with momentum and adaptive learning rate: 85 Epochs.**

- Many **variations** on the standard Backpropagation Algorithm have been developed over the years to overcome such problems.