Unit 4

NEURAL NETWORK

Contents

- Introduction
- Biological Neural Networks
- Perceptron, Multilayer and Recursive Nets
- Gradient Descent
- Back Propagation

Introduction

- Machine Learning involves adaptive mechanism that enable computer to learn from experience, example and analogy
- Learning capability improves performance of an Artificial System over time
- Machine learning mechanisms form the basis for adaptive systems
- The most popular approaches to machine learning are Artificial Neural Network (ANN) and Genetic Algorithms(GA)
- Two types of Machine Learning :
  - Supervised Learning and
  - Unsupervised learning

Biological Neural Networks

- A neural network can be defined as a model of reasoning based on the human brain. The brain consists of a densely interconnected set of nerve cells, or basic information-processing units, called **neurons**
- The human brain incorporates nearly 10 billion neurons and 60 trillion connections, **synapses**, between them. By using multiple neurons simultaneously, the brain can perform its functions much faster than the fastest computers in existence today.

Biological Neural Networks

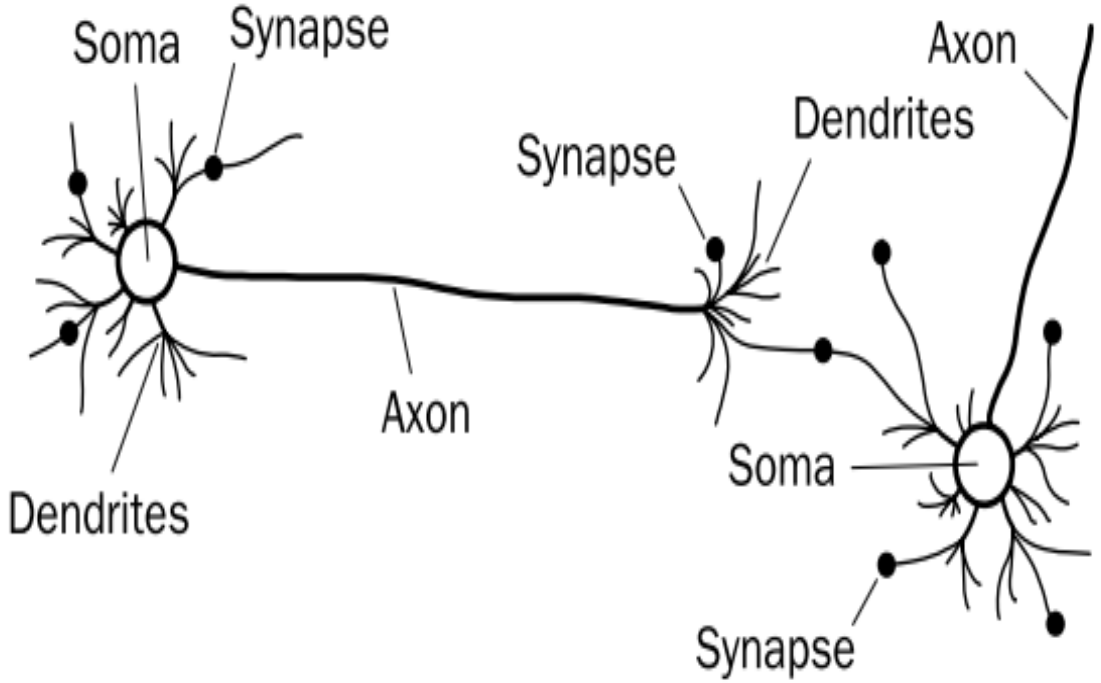
- Each neuron has a very simple structure, but an army of such elements constitutes a tremendous processing power
- **Neuron**: fundamental functional unit of all nervous system tissue

Soma: cell body, contain nucleus

Dendrites: a number of fibres, input

Axon: single long fibre with many branches, output

Synapse: junction of dendrites and axon, each neuron form synapse with 10 to 100000 other neurons



Biological Neural Network

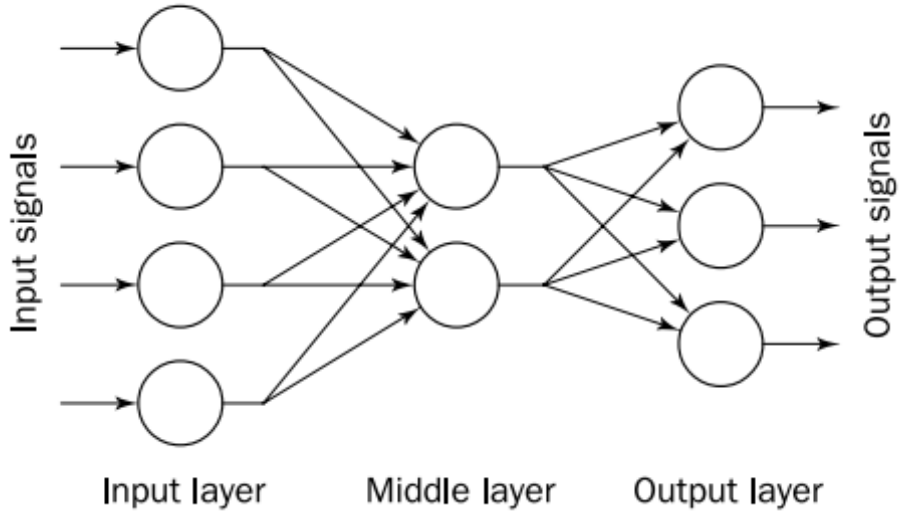
HOW DOES BRAIN WORKS

Signals are propagated from neuron to neuron by electrochemical reaction:

1. Chemical substances are released from the synapses and enter the dendrite, raising or lowering the electrical potential of the cell body
2. When a potential reaches a threshold, an electrical pulse or **action potential** is sent down the axon
3. The pulse spreads out along the branches of the axon, eventually reaching synapses and releasing transmitters into the bodies of other cells
    1. **Excitory** Synapses: Increase potential
    2. **Inhibitory** Synapses : Decreases potential

Artificial Neural Network (ANN)

- Consists of a number of very simple and highly interconnected processors called neurons
- The neurons are connected by weighted links passing signals from one neuron to another.
- The output signal is transmitted through the neuron's outgoing connection. The outgoing connection splits into a number of branches that transmit the same signal. The outgoing ranches terminate at the incoming connections of other neurons in the network



Analogy Between Biological and ANN

Biological Neural Network	Artificial Neural Network
Soma	Neuron
Dendrite	Input
Axon	Output
Synapse	Weight

Learning in ANN

Supervised learning

- Uses a set of inputs for which the desired outputs are known
- Example: Back propagation algorithm

Unsupervised learning

- Uses a set of inputs for which no desired output are known.
- The system is self-organizing; that is, it organizes itself internally. A human must examine the final categories to assign meaning and determine the usefulness of the results.
- Example: Self-organizing map

The Neuron as a simple computing element: Diagram of a neuron

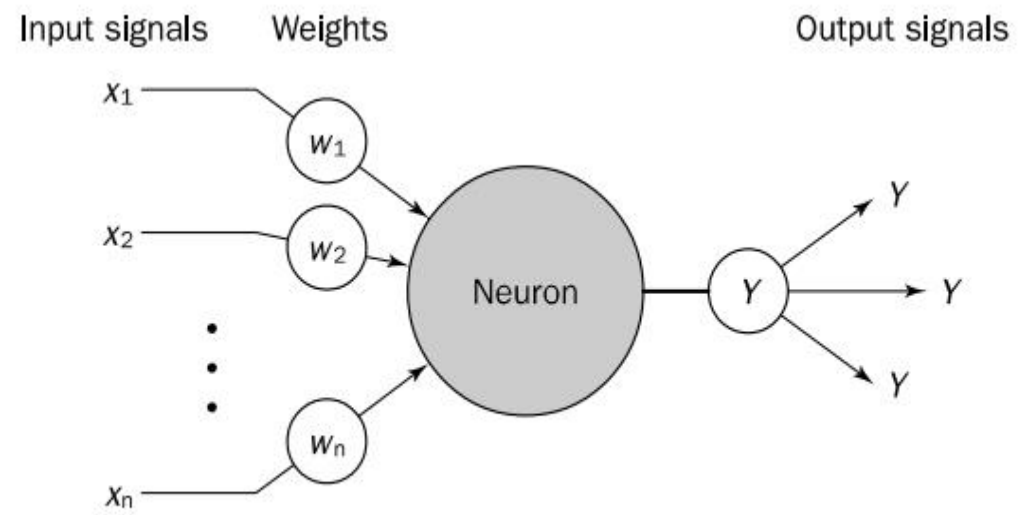
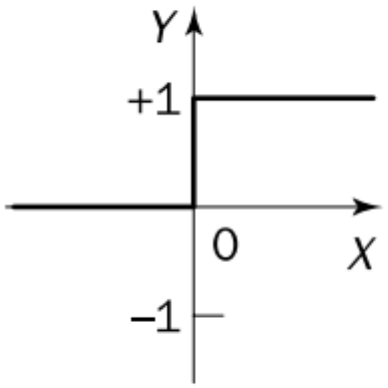


Diagram of a neuron

- The neuron computes the weighted sum of the input signals and compares the result with a **threshold value**, θ. If the net input is less than the threshold, the neuron output is –1. But if the net input is greater than or equal to the threshold, the neuron becomes activated and its output attains a value +1.
- The neuron uses the following transfer or **activation function**
- $X = \sum_{i=1}^{n} x_i w_i \qquad Y = \begin{cases} +1 \; if \; X \geq \theta \\ -1 \; if \; X \leq \theta \end{cases}$
- This type of activation function is called a **sign function**

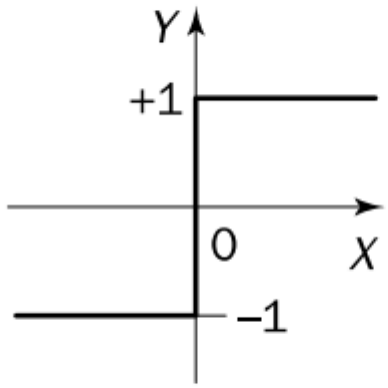
Activation Functions for Neuron

Step function



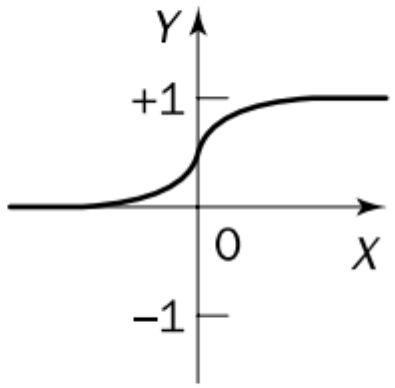
$$Y^{step} = \begin{cases} 1, \text{ if } X \geq 0 \\ 0, \text{ if } X < 0 \end{cases}$$

Sign function



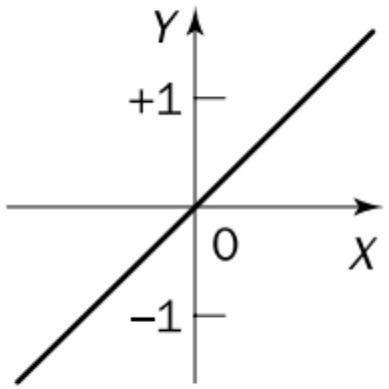
$$Y^{sign} = \begin{cases} +1, \text{ if } X \geq 0 \\ -1, \text{ if } X < 0 \end{cases}$$

Sigmoid function



$$Y^{sigmoid} = \frac{1}{1 + e^{-X}}$$

Linear function



$$Y^{linear} = X$$

Perceptron

- In 1958, Frank Rosenblatt introduced a training algorithm that provided the first procedure for training a simple ANN : a perceptron
- The perceptron is the simplest form of a neural network. It consists of a single neuron with *adjustable* synaptic weights and a **hard limiter**
- The operation of Rosenblatt's perceptron is based on the McCulloch and Pitts neuron model. The model consists of a **linear combiner** followed by a hard limiter
- The weighted sum of the inputs is applied to the hard limiter, which produces an output equal to +1 if its input is positive and -1 if its input is negative

Perceptron

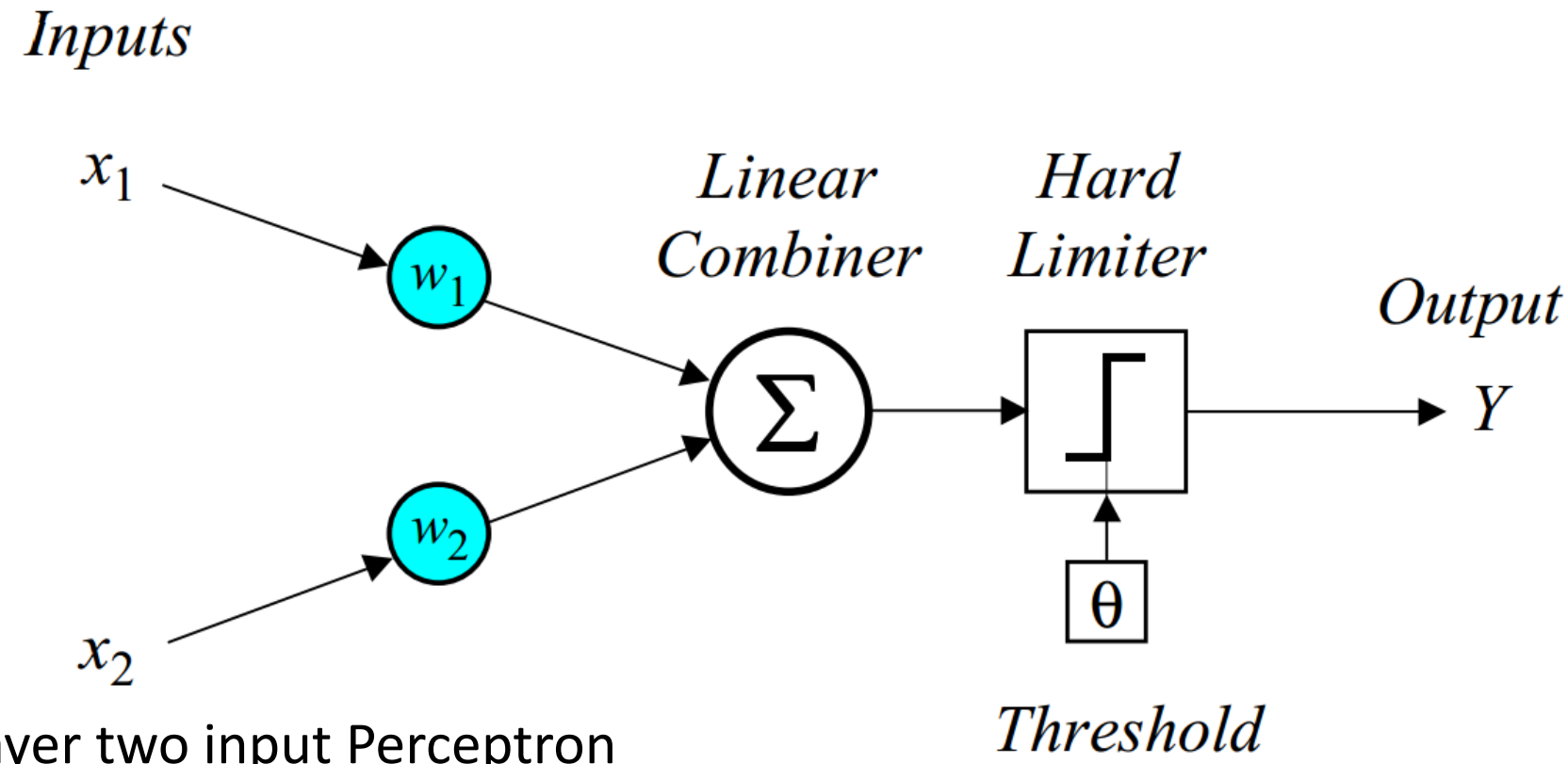


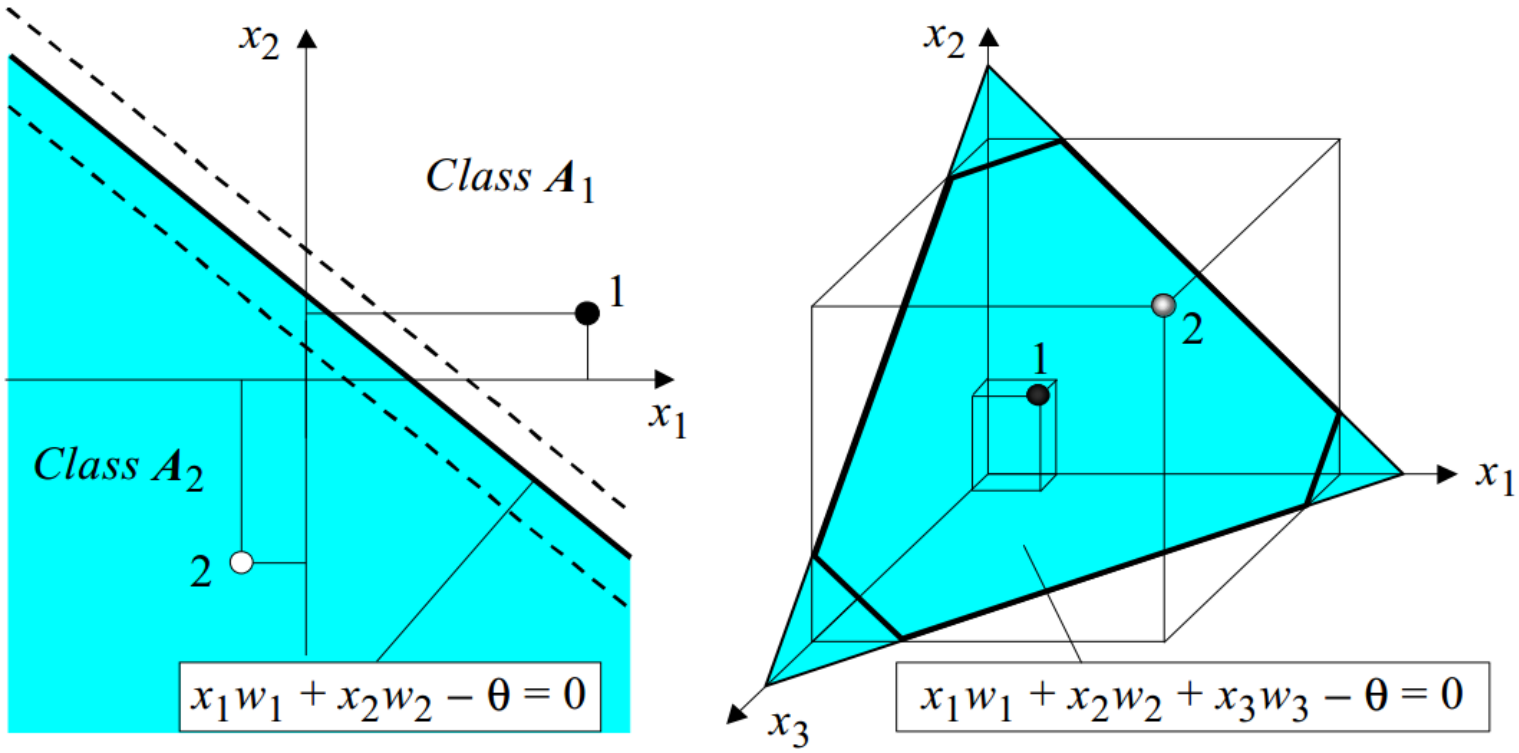
Fig: Single Layer two input Perceptron

Perceptron

- The aim of the perceptron is to classify inputs, x_1 , x_2 , x_3 , x_4 , … , x_n. Into one of two classes, say A_1, A_2.
- In the case of an elementary perceptron, the n-dimensional space is divided into a *hyperplane* into two decision regions. The hyperplane is defined by ***linearly separable function***

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

Perceptron



(*a*) Two-input perceptron. (*b*) Three-input perceptron.

Fig: Linear Separability in the perceptron

Perceptron

How does the perceptron learn its classification tasks?

- This is done by making small adjustment in the weights to reduce the difference between the actual and desired outputs of the perceptron. The initial weights are randomly assigned, usually in the range [-0.5, +0.5], and then updated to obtain the output consistent with the training examples.

- If at iteration p, the actual output is , $Y_{(p)}\ and$ the desired output is , $Y_{d(p)}$, then the error is given by :

$$e_p = Y_{d(p)} - Y_{(p)}$$, where *p = 1, 2, 3 , …*

Iteration *p* here refers to the *pth* training example presented to the perceptron

- If the error, e_p is positive, we need to increase perceptron output $Y_{(p)}$, but if it is negative, we need to decrease $Y_{(p)}$

Perceptron

Perceptron Learning Rule

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

where $p = 1, 2, 3, \ldots$

α is the **learning rate ,** a positive constant less than unity

The perceptron learning rule was first proposed by Rosenblatt in 1960. Using this rule we can derive perceptron training algorithm for classification task

Perceptron training algorithm

Step 1 : Initialisation

Set initial weights w_1, w_2, …, w_n and threshold θ to random numbers in the range [-0.5, +0.5].

If error . e_p is positive, we need to increase perceptron output Y_p , but if it is negative, we need to decrease Y_p

Perceptron training algorithm (contd…)

Step 2 : Activation

Activate the perceptron by applying inputs $x_1(p)$, $x_2(p)$, …, $x_n(p)$ and desired output $Y_d(p)$

Calculate the actual output at iteration p = 1

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

Where *n* is the number of the perceptron inputs, and *step* is activation function

Perceptron training algorithm (contd…)

Step 3: Weight Training

Update the weights of the perceptron

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

Where $\Delta w_i(p)$ is the weight correction at iteration *p* . The weight correction is computed by the delta rule:

$$\Delta w_i(p) = \alpha \times x_i(p) \times e(p)$$

Step 4: Iteration

Increase iteration *p* by 1, go back to *Step 2* and repeat the process until convergence

Multilayer Neural Network

- A multi layer perceptron is a feed forward neural network with one or more hidden layers
- The network consists of :
  - Input Layer
  - Hidden Layer
  - Output Layer
- The input signal is propagated in a forward direction in a layer-by-layer basis

Multilayer Neural Network

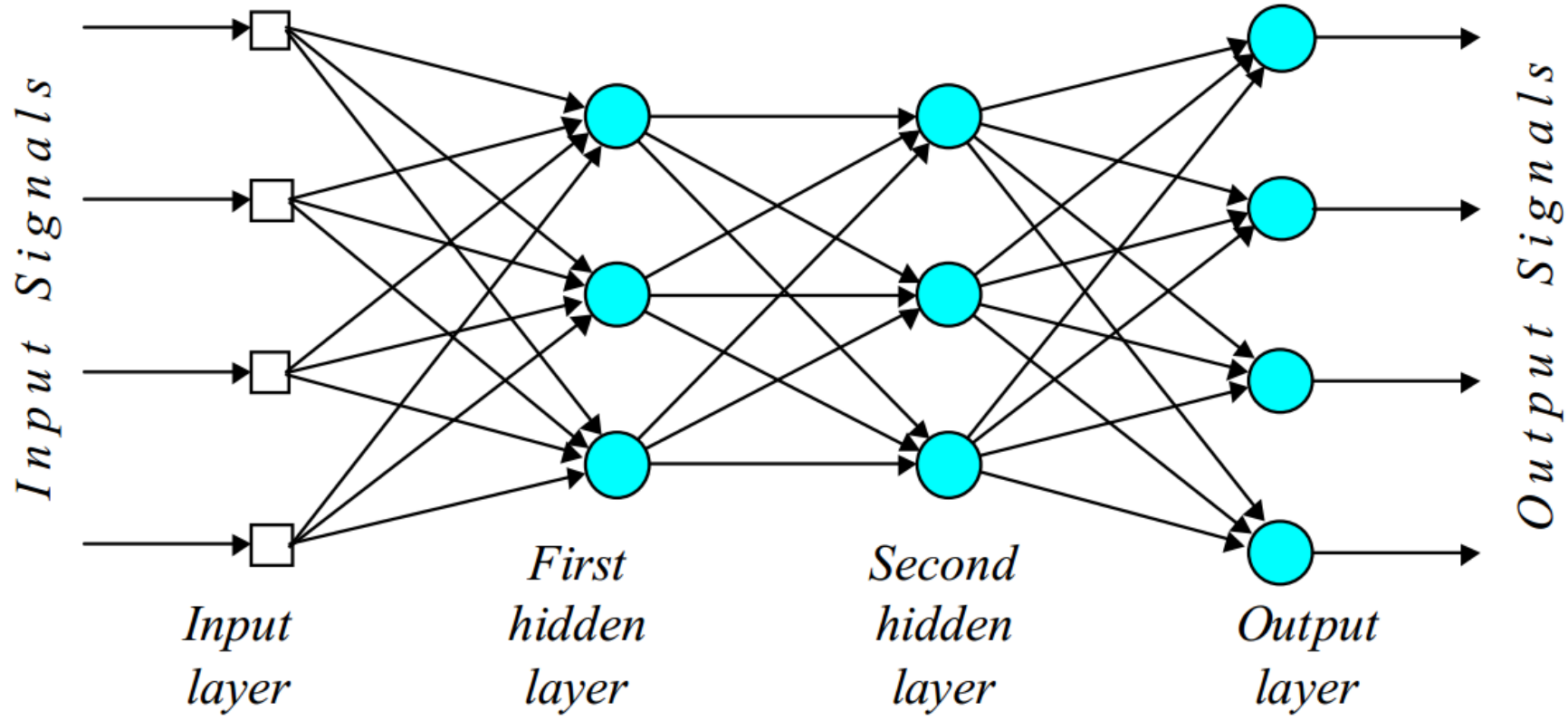


Fig: A multilayer Perceptron with two hidden layers

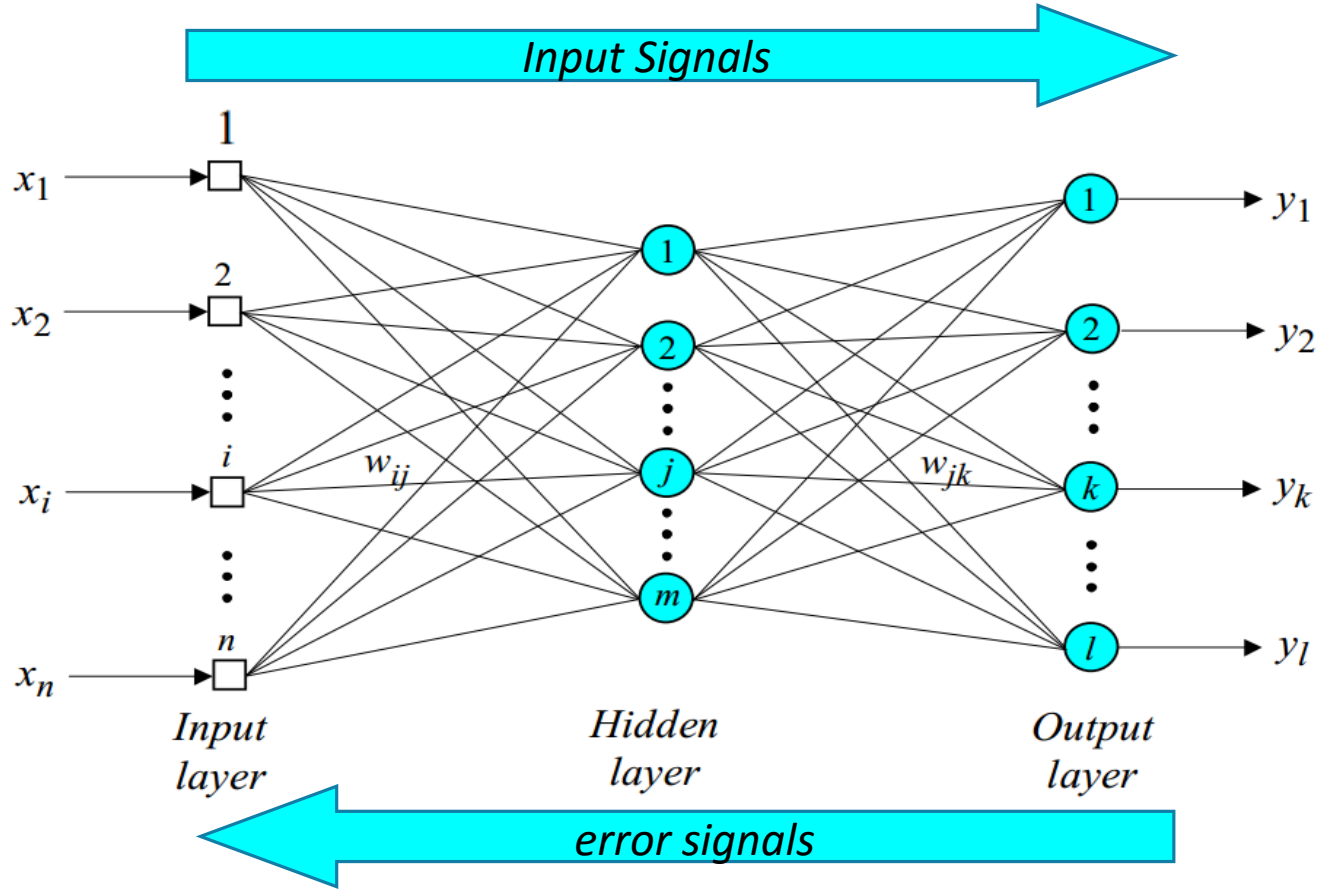
Multilayer Neural Network

- The hidden layer “hides” its desired output. Neurons in the hidden layer can not be observed through the input/output behavior of the network. There is no obvious way to know what the desired output of the hidden layer should be.
- Commercial ANNs incorporate three and sometimes four layers, including one or two hidden layers. Each layer can contain from 10 to 1000 neurons. Experimental neural networks may have fie or six layers, including three or four hidden layers, and utilise millions of neurons.

Back Propagation

- 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 other word difference between actual and desired output pattern – the weight are adjusted to reduce the 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 out layer
- 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



Back Propagation Training Algorithm

Step 1: Initialization

Set all the weights and threshold levels of the network to random numbers uniformly distributed inside a small range:

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

where F_i is the total number of inputs of neuron i in the network. The weight initialization is done on a neuron-by-neuron basis.

Back Propagation Training Algorithm

Step 2: Activation

Activate the back-propagation neural network by applying inputs $x_1(p)$, $x_2(p)$, …, $x_n(p)$ and desired outputs $y_{d,1}(p)$, $y_{d,2}(p)$,…, $y_{d,n}(p)$

A) Calculate the actual output of the neurons in the hidden layers:

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

Where *n* is the number of inputs of neuron *j* in the hidden layer, and *sigmoid* is the *sigmoid* activation function

Back Propagation Training Algorithm

Step 2: Activation(contd…)

b) 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.

Back Propagation Training Algorithm

Step 3: weight Training

Update the weights in the back-propagation network propagating backward the errors associated with output neurons.

(a) 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:

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

Back Propagation Training Algorithm

Step 3: weight Training(contd…)

(b) 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:

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

Back Propagation Training Algorithm

Step 3: Iteration

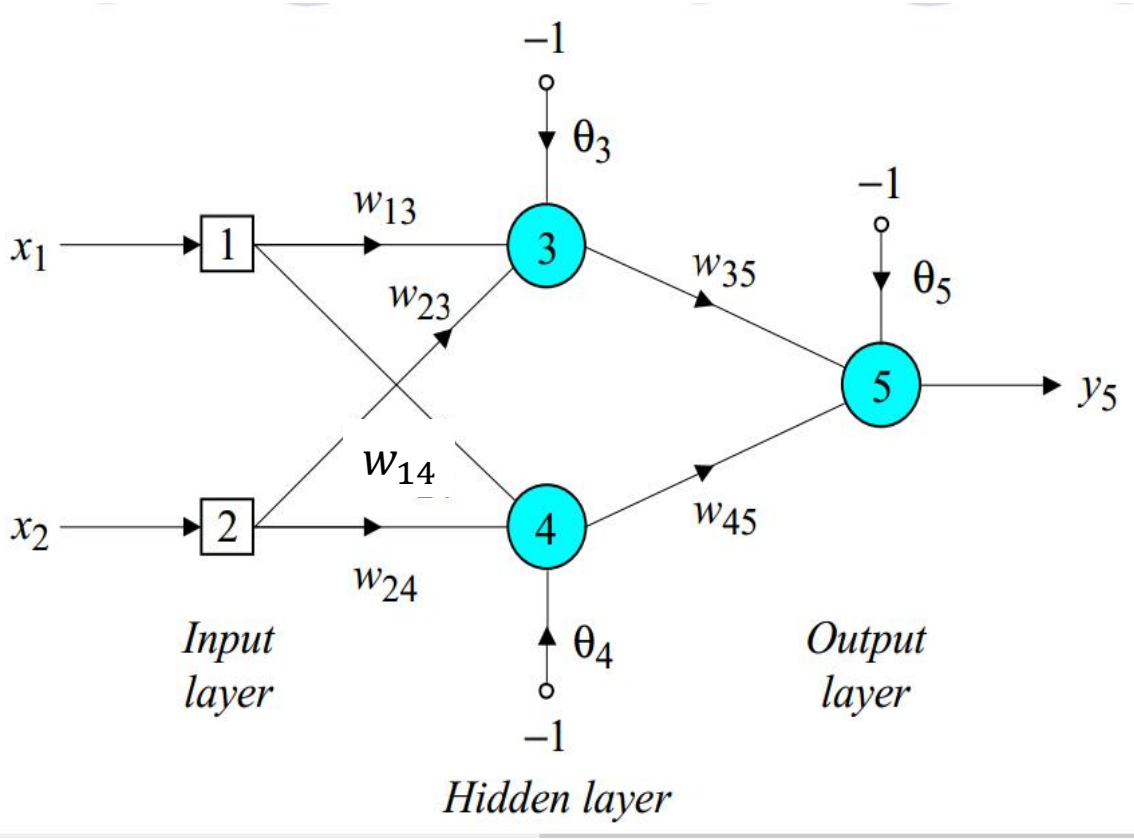
Increase iteration *p* by one, go back to *Step 2* and repeat the process until the selected error criterion is satisfied.

As an example, we may consider the three layer back-propagation network. Suppose that the network is required to perform logical operation Exclusive-OR. Recall that a single-layer perceptron could not do this operation. Now we will apply the three layer net.

Example: Three-layer network for solving the Exclusive-OR operation

- The effect of the threshold applied to a neuron in the hidden layer is represented by its weight, θ, connected to a fixed input equal to -1
- The initial weights and threshold levels are set randomly as follows:

$w_{13} = 0.5, w_{14} = 0.9, w_{23} = 0.4, w_{24} = 1.0, w_{35} = -1.2,$
$w_{45} = 1.1, \theta_3 = 0.8, \theta_4 = -0.1$ and $\theta_5 = 0.3$.



Example: Three-layer network for solving the Exclusive-OR operation

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 output of neurons 3 and 4 in the hidden layers 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:

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

Example: Three-layer network for solving the Exclusive-OR operation

- 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

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

Example: Three-layer network for solving the Exclusive-OR operation

- Now we calculate the error gradients for neuron 3 and 4 in the hidden layer:

$$\delta_3 = y_3(1-y_3)\cdot\delta_5\cdot w_{35} = 0.5250\cdot(1-0.5250)\cdot(-0.1274)\cdot(-1.2) = 0.0381$$

$$\delta_4 = y_4(1-y_4)\cdot\delta_5\cdot w_{45} = 0.8808\cdot(1-0.8808)\cdot(-0.127\,4)\cdot 1.1 = -0.0147$$

- We, then, determine the weight corrections:

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

Example: Three-layer network for solving the Exclusive-OR operation

- At last, we update all weights and thresholds
- The training process is updated till the sum of squared error is less than 0.001

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

Example: Three-layer network for solving the Exclusive-OR operation

- The Final results of three layer network learning is:

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	

Gradient Descent

- Gradient descent is an iterative minimisation method. The gradient of the error function always shows in the direction of the steepest ascent of the error function.
- It is determined as the derivative of the activation function multiplied by the error at the neuron output

For Neuron *k* in the output layer

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

Where, yk(p) is the output of neuron k at iteration p, and Xk(p) is the net weighted input of neuron k at same iteration.

Characteristics of ANN

- Adaptive learning
- Self-organization
- Error tolerance
- Real-time operation
- Parallel information processing

Benefits and Limitations of ANN

Benefits	Limitations
Ability to tackle new kind of problems	Performs less well at tasks humans tend to find difficult
Robustness	Lack of explanation facilities
	Require large amount of test data