Neural Networks

Overview

- **Basics of Neural Network**
- Advanced Features of Neural Network
- Applications I-II
- Summary

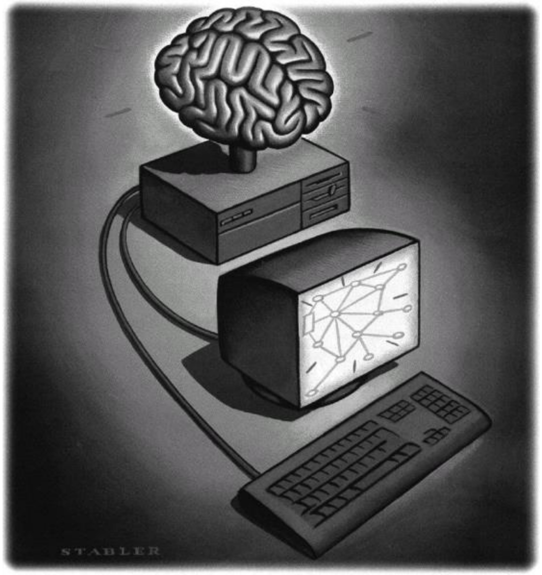
Basics of Neural Network

- What is a Neural Network
- Neural Network Classifier
- Data Normalization
- Neuron and bias of a neuron
- Single Layer Feed Forward
- Limitation
- Multi Layer Feed Forward
- Back propagation

Neural Networks

What is a Neural Network?

•Biologically motivated approach to machine learning



Similarity with biological network

Fundamental processing elements of a neural network is a neuron

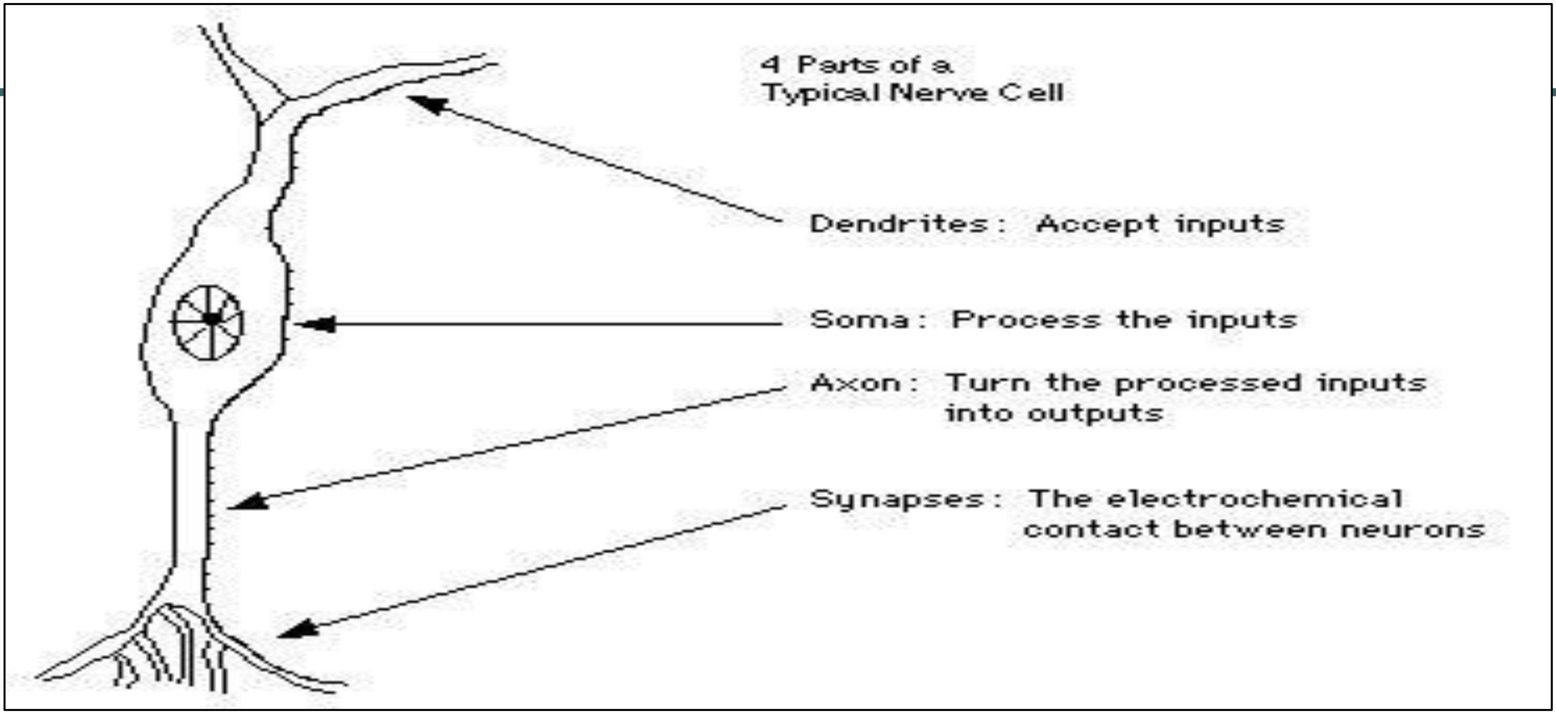
1.Receives inputs from other source

2.Combines them in someway

3.Performs a generally nonlinear operation on the result

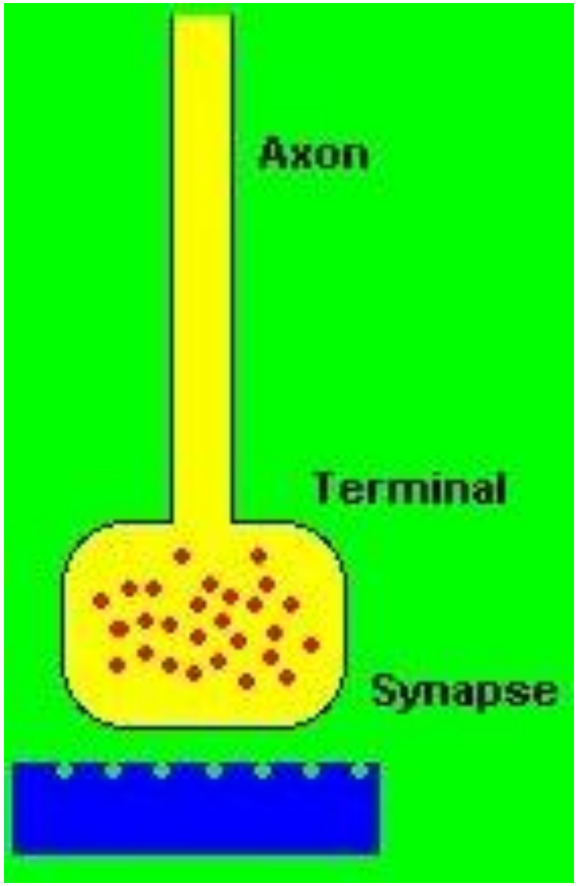
4.Outputs the final result

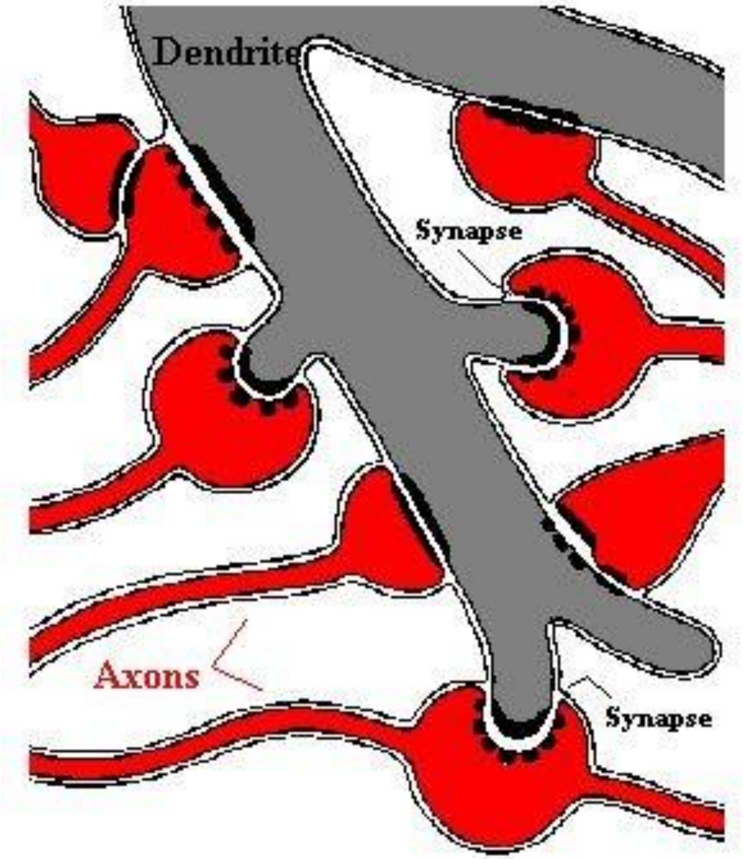
Similarity with Biological Network



- Fundamental processing element of a neural network is a neuron
- A human brain has 100 billion neurons
- An ant brain has 250,000 neurons

Synapses, the basis of learning and memory





Neural Network

- Neural Network is a set of connected INPUT/OUTPUT UNITS, where each connection has a WEIGHT associated with it.

- Neural Network learning is also called CONNECTIONIST learning due to the connections between units.

- It is a case of SUPERVISED, INDUCTIVE or CLASSIFICATION learning.

Neural Network

- Neural Network learns by adjusting the weights so as to be able to correctly classify the training data and hence, after testing phase, to classify unknown data.

- Neural Network needs long time for training.

- Neural Network has a high tolerance to noisy and incomplete data

Neural Network Classifier

- **Input: Classification data**
  It contains classification attribute
- Data is divided, as in any classification problem. [Training data and Testing data]
- **All data must be normalized.**
  (i.e. all values of attributes in the database are changed to contain values in the internal [0,1] or[-1,1])
  Neural Network can work with data in the range of (0,1) or (-1,1)
- **Two basic normalization techniques**
  1 Max-Min normalization
  2 Decimal Scaling normalization

Data Normalization

[1] Max- Min normalization formula is as follows:

$$v' = \frac{v - \min A}{\max A - \min A}(new\_\max A - new\_\min A) + new\_\min A$$

[minA, maxA , the minimun and maximum values of the attribute A max-min normalization maps a value v of A to v' in the range {new_minA, new_maxA}]

Example of Max-Min Normalization

Max- Min normalization formula

$$v' = \frac{v - \min A}{\max A - \min A}(new\_\max A - new\_\min A) + new\_\min A$$

Example: **We want to normalize data to range of the interval [0,1].**
We put: new_max A= 1, new_minA =0.

Say, max A was 100 and min A was 20 (That means maximum and minimum values for the attribute).

Now, if v = 40 (If for this particular pattern , attribute value is 40), v' will be calculated as , v' = (40-20) x (1-0) / (100-20) + 0

=> v' = 20 x 1/80

=> v' = 0.4

Decimal Scaling Normalization

[2]Decimal Scaling Normalization

Normalization by decimal scaling normalizes by moving the decimal point of values of attribute A.

$$\boxed{v' = \frac{v}{10^j}}$$

Here j is the smallest integer such that max|v'|<1.

Example :

A – values range from -986 to 917.   Max |v| = 986.

v = -986 normalize to v' = -986/1000 = -0.986

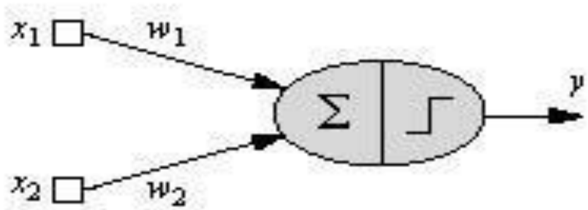


Fig1: an artificial neuron

One Neuron as a Network

- Here x1 and x2 are normalized attribute value of data.
- y is the output of the neuron , i.e the class label.
- x1 and x2 values multiplied by weight values w1 and w2 are input to the neuron x.
- Value of x1 is multiplied by a weight w1 and values of x2 is multiplied by a weight w2.
- Given that
  - w1 = 0.5 and w2 = 0.5
  - Say value of x1 is 0.3 and value of x2 is 0.8,
  - So, weighted sum is :
  - sum= w1 x x1 + w2 x x2 = 0.5 x 0.3 + 0.5 x 0.8 = 0.55

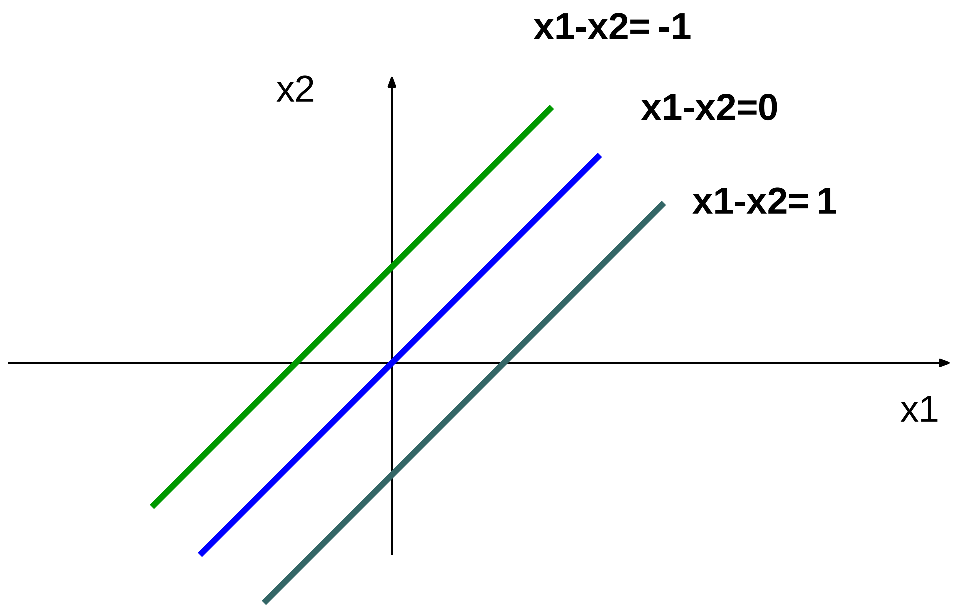
One Neuron as a Network

- **The neuron receives the weighted sum as input and calculates the output as a function of input as follows :**
- **y = f(x) , where f(x) is defined as**
- **f(x) = 0 { when x< 0.5 }**
- **f(x) = 1 { when x >= 0.5 }**
- **For our example, x (weighted sum) is 0.55, so y = 1 ,**
- That means corresponding input attribute values are classified in class 1.
- **If for another input values , x = 0.45 , then f(x) = 0,**
- **so we could conclude that input values are classified to class 0.**

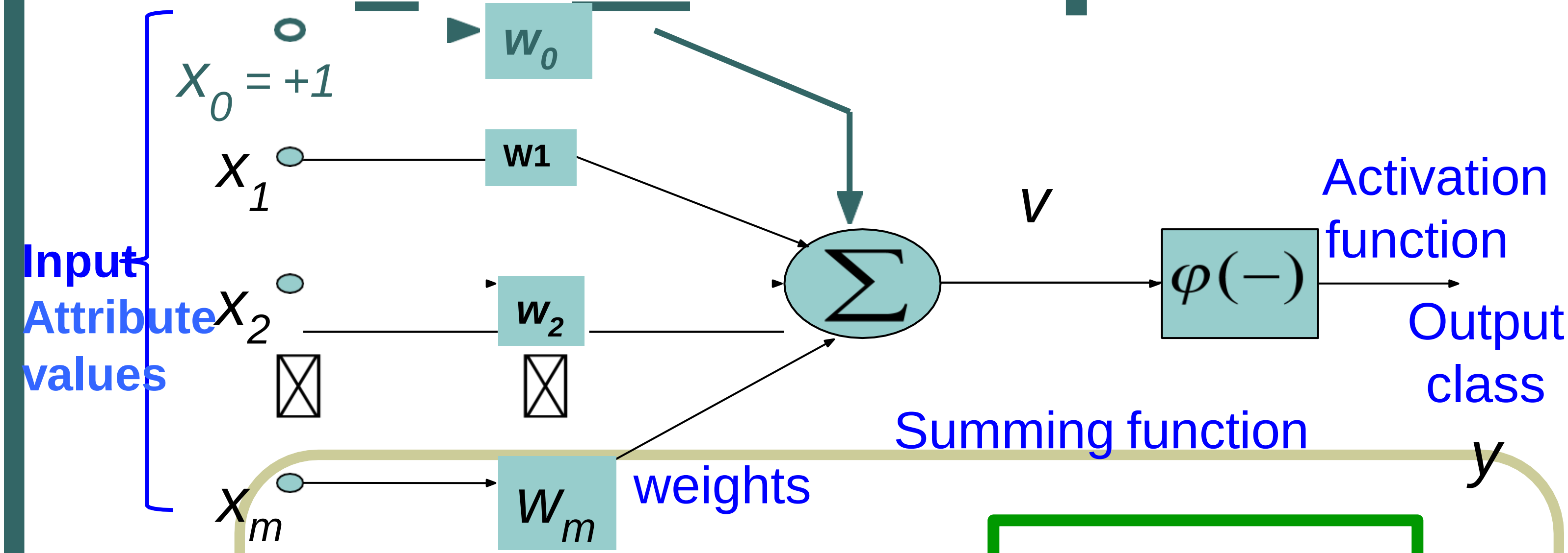
Bias of a Neuron

- We need the bias value to be added to the weighted sum $\sum w_i x_i$ so that we can transform it from the origin.

  ***v* = $\sum w_i x_i$ + *b*, here *b* is the bias**



Bias as extra input



$$v = \sum_{j=0}^{m} w_j x_j$$

$$w_0 = b$$

Neuron with Activation

- The neuron is the basic information processing unit of a NN. It consists of:

---

1 A set of links, describing the neuron inputs, with weights $W_1, W_2, \ldots, W_m$

2. An adder function (linear combiner) for computing the weighted sum of the inputs (real numbers):

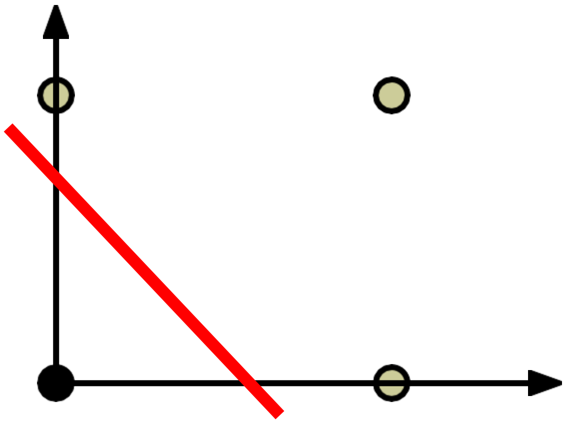
$$\mathbf{u} = \sum_{j=1}^{\mathbf{m}} \mathbf{w_j x_j}$$

3 Activation function : for limiting the amplitude of the neuron output.

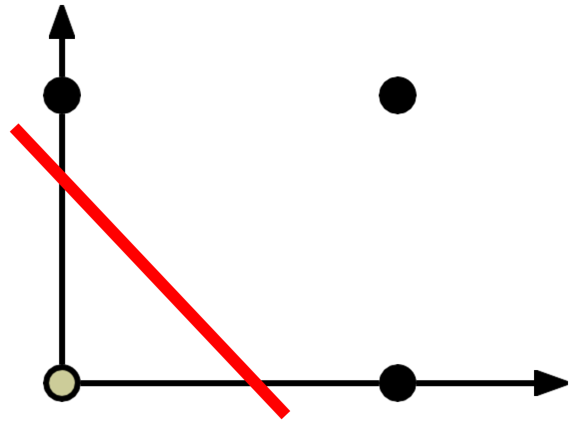
$$\mathrm{y} = \varphi(\mathrm{u} + b)$$

Why We Need Multi Layer ?

- Linear Separable:

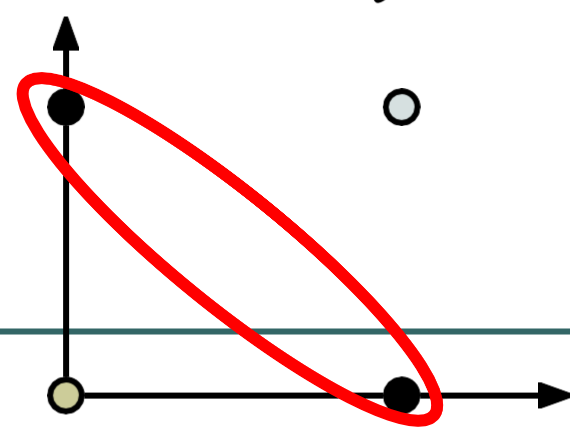


$x \wedge y$



$x \vee y$

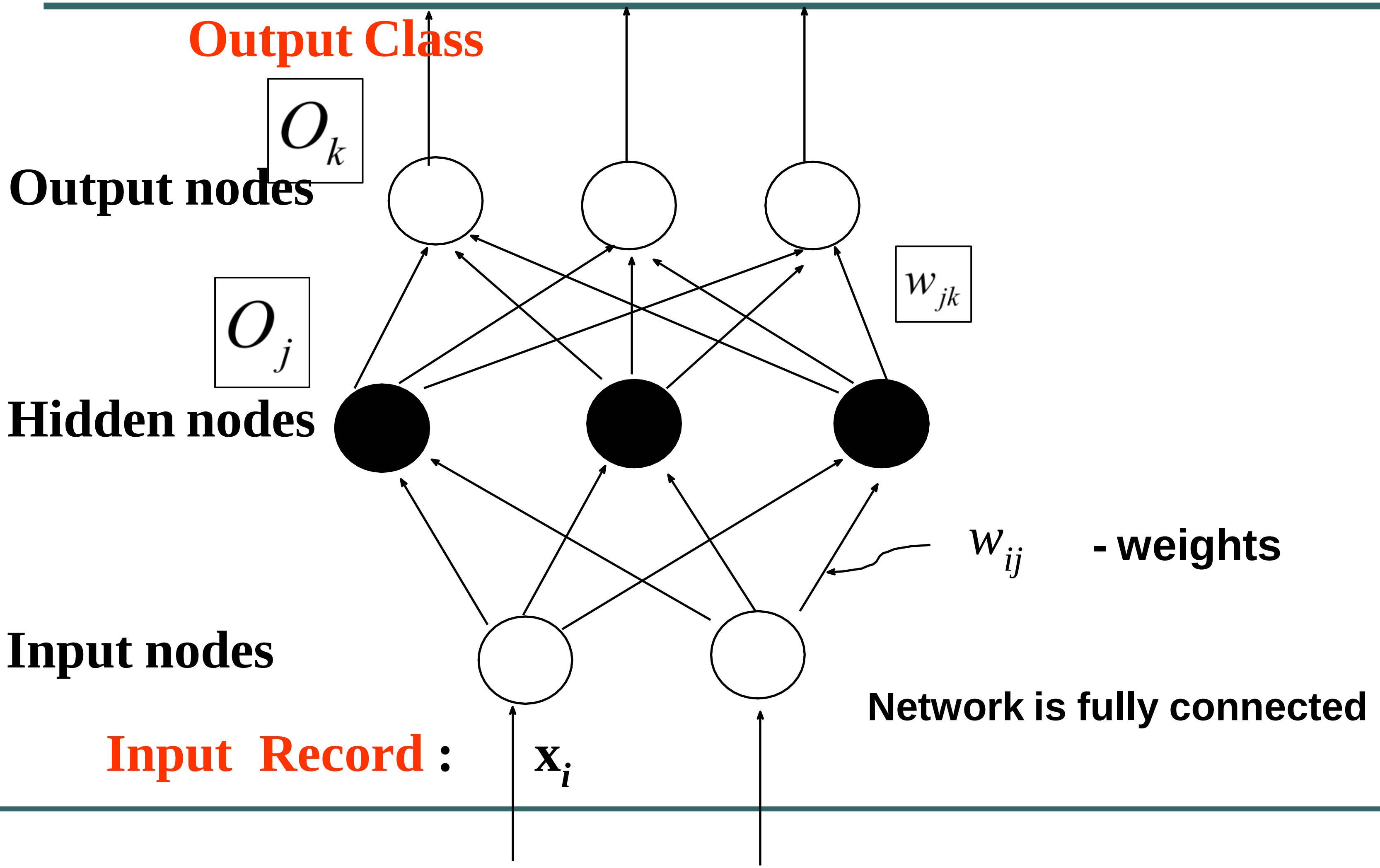
- Linear inseparable:



$x \veebar y$

- Solution?

A Multilayer Feed-Forward Neural Network



Neural Network Learning

- The inputs are fed simultaneously into the input layer.
- The weighted outputs of these units are fed into hidden layer.
- The weighted outputs of the last hidden layer are inputs to units making up the output layer.

A Multilayer Feed Forward Network

- The units in the hidden layers and output layer are sometimes referred to as neurodes, due to their symbolic biological basis, or as output units.
- A network containing two hidden layers is called a three-layer neural network, and so on.
- The network is feed-forward in that none of the weights cycles back to an input unit or to an output unit of a previous layer.

A Multilayered Feed – Forward Network

- INPUT: records without class attribute with normalized attributes values.

- INPUT VECTOR: $X = \{ x1, x2, \ldots. xn\}$

  where n is the number of (non class) attributes.

- INPUT LAYER – there are as many nodes as non-class attributes i.e. as the length of the input vector.

- HIDDEN LAYER – the number of nodes in the hidden layer and the number of hidden layers depends on implementation.

A Multilayered Feed–Forward Network

- **OUTPUT LAYER** – corresponds to the class attribute.
- There are as many nodes as classes (values of the class attribute).

$$\boxed{O_k}$$

k= 1, 2,.. #classes

•Network is fully connected, i.e. each unit provides input to each unit in the next forward layer.

Classification by Back propagation

- *Back Propagation learns by iteratively processing a set of training data (samples).*
- For each sample, weights are modified to minimize the error between network's classification and actual classification.

Steps in Back propagation Algorithm

- STEP ONE: initialize the weights and biases.
- The weights in the network are initialized to random numbers from the interval [-1,1].
- Each unit has a BIAS associated with it
- The biases are similarly initialized to random numbers from the interval [-1,1].
- STEP TWO: feed the training sample.

Steps in Back propagation Algorithm (cont..)

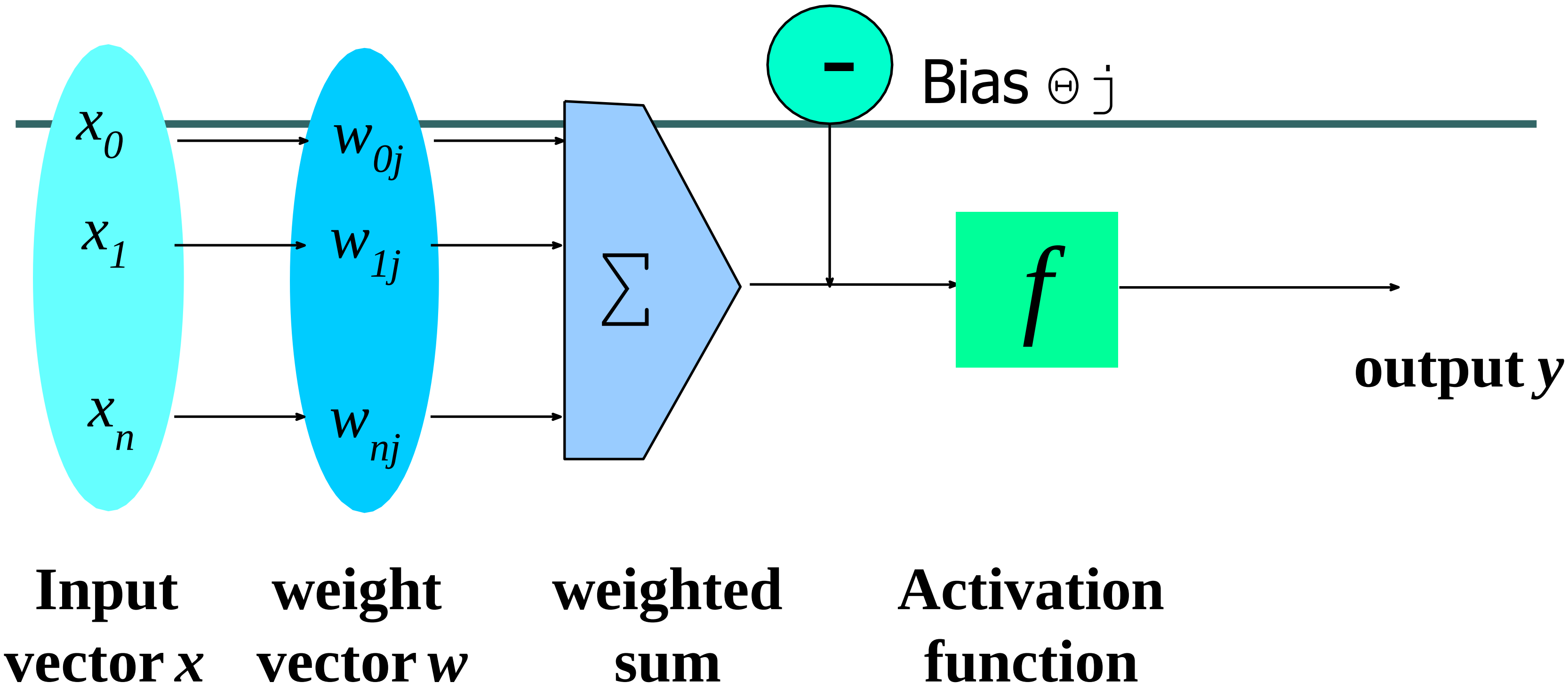
- STEP THREE: Propagate the inputs forward; we compute the net input and output of each unit in the hidden and output layers.

- STEP FOUR: back propagate the error.

- STEP FIVE: update weights and biases to reflect the propagated errors.

- STEP SIX: terminating conditions.

Propagation through Hidden Layer (One Node)



- The inputs to unit j are outputs from the previous layer. These are multiplied by their corresponding weights in order to form a weighted sum, which is added to the bias associated with unit j.
- A nonlinear activation function f is applied to the net input.

Propagate the inputs forward

- For unit j in the input layer, its output is equal to its input, that is, $O_j = I_j$

for input unit j.

•The net input to each unit in the hidden and output layers is computed as follows.

•Given a unit j in a hidden or output layer, the net input is

$$I_j = \sum_i w_{ij} O_i + \theta_j$$

where wij is the weight of the connection from unit i in the previous layer to unit j; Oi is the output of unit I from the previous layer;

θ_j is the bias of the unit

Back propagate the error

- When reaching the Output layer, the error is computed and propagated backwards.
- For a unit k in the output layer the error is computed by a formula:

$$\boxed{Err_k = O_k(1 - O_k)(T_k - O_k)}$$

Where O k – actual output of unit k (computed by activation function.

$$\boxed{O_k = \frac{1}{1 + e^{-I_k}}}$$

Tk – True output based of known class label; classification of training sample

Ok(1-Ok) – is a Derivative (rate of change) of activation function.

Back propagate the error

- The error is propagated backwards by updating weights and biases to reflect the error of the network classification .
- For a unit j in the hidden layer the error is computed by a formula:

$$\boxed{Err_j = O_j(1-O_j)\sum_k Err_k w_{jk}}$$

where wjk is the weight of the connection from unit j to unit k in the next higher layer, and Errk is the error of unit k.

Update weights and biases

- Weights are updated by the following equations, where l is a constant between 0.0 and 1.0 reflecting the learning rate, this learning rate is fixed for implementation.

$$\Delta w_{ij} = (l) Err_j O_i$$

$$w_{ij} = w_{ij} + \Delta w_{ij}$$

- Biases are updated by the following equations

$$\Delta \theta_j = (l) Err_j$$

$$\theta_j = \theta_j + \Delta \theta_j$$

Update weights and biases

- We are updating weights and biases after the presentation of each sample.

- This is called case updating.

- Epoch --- One iteration through the training set is called an epoch.

- **Epoch updating ------------**
- **Alternatively, the weight and bias increments could be accumulated in variables and the weights and biases updated after all of the samples of the training set have been presented.**

- **Case updating is more accurate**

Terminating Conditions

- Training stops

• All $\boxed{\Delta w_{ij}}$ in the previous epoch are below some threshold, or

•The percentage of samples misclassified in the previous epoch is below some threshold, or

• a pre specified number of epochs has expired.

•In practice, several hundreds of thousands of epochs may be required before the weights will converge.

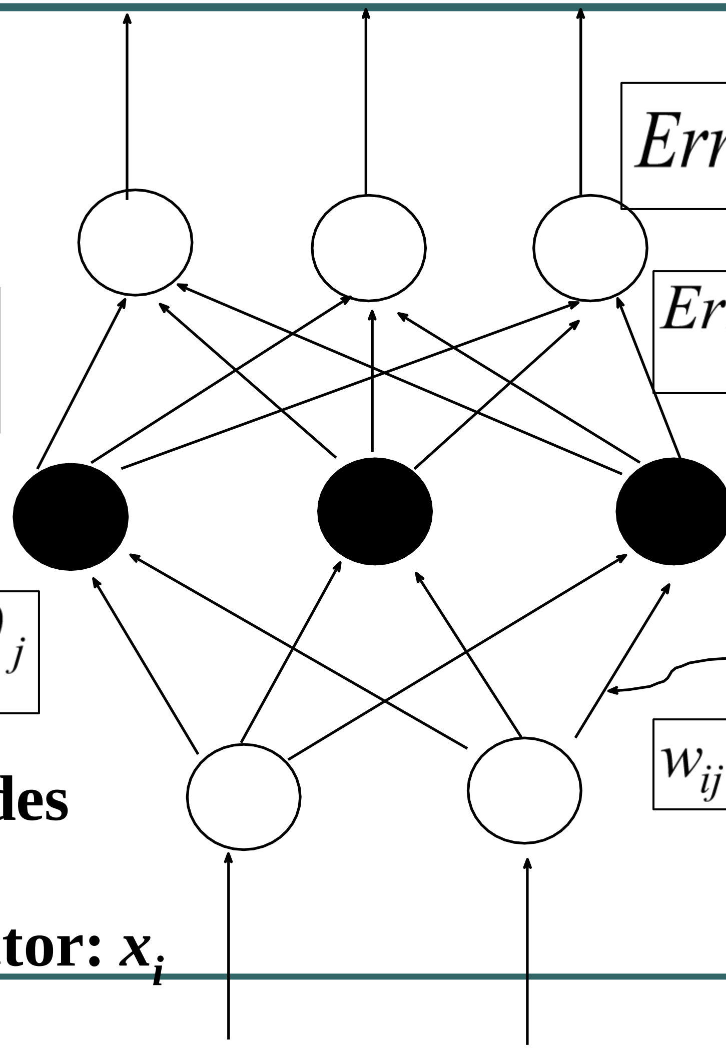
Backpropagation Formulas

Output vector

$$Err_k = O_k(1-O_k)(T_k - O_k)$$

Output nodes

$$O_j = \frac{1}{1+e^{-I_j}}$$



Hidden nodes

$$Err_j = O_j(1-O_j)\sum_k Err_k w_{jk}$$

$$I_j = \sum_i w_{ij} O_i + \theta_j$$

w_{ij}

$$\theta_j = \theta_j + (l) Err_j$$

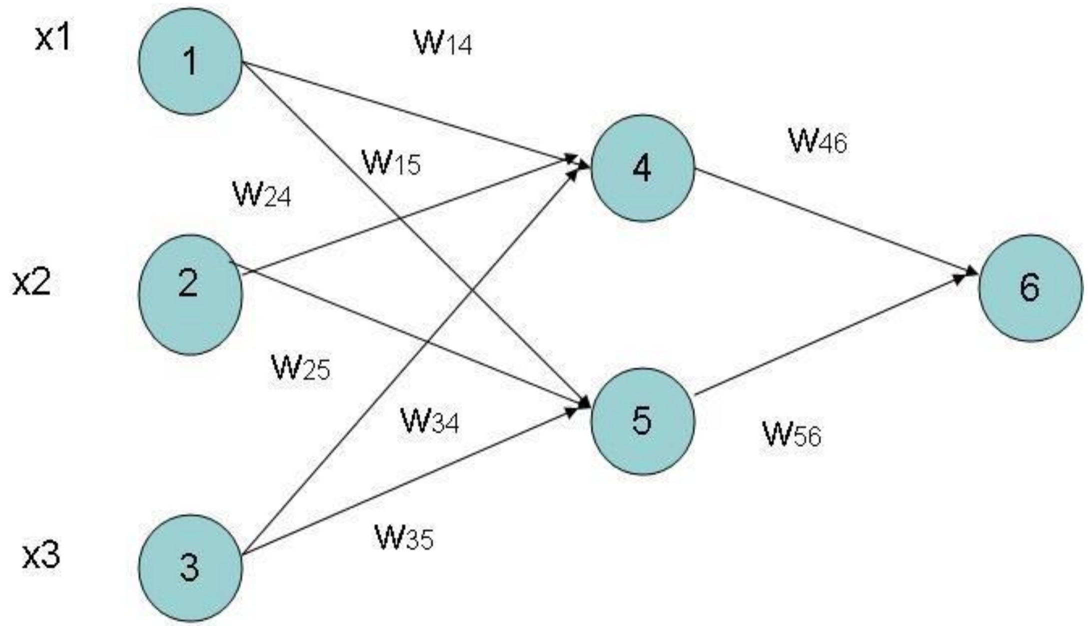
$$w_{ij} = w_{ij} + (l) Err_j O_i$$

Input nodes

Input vector: x_i

Example of Back propagation

Input = 3, Hidden Neuron = 2 Output =1



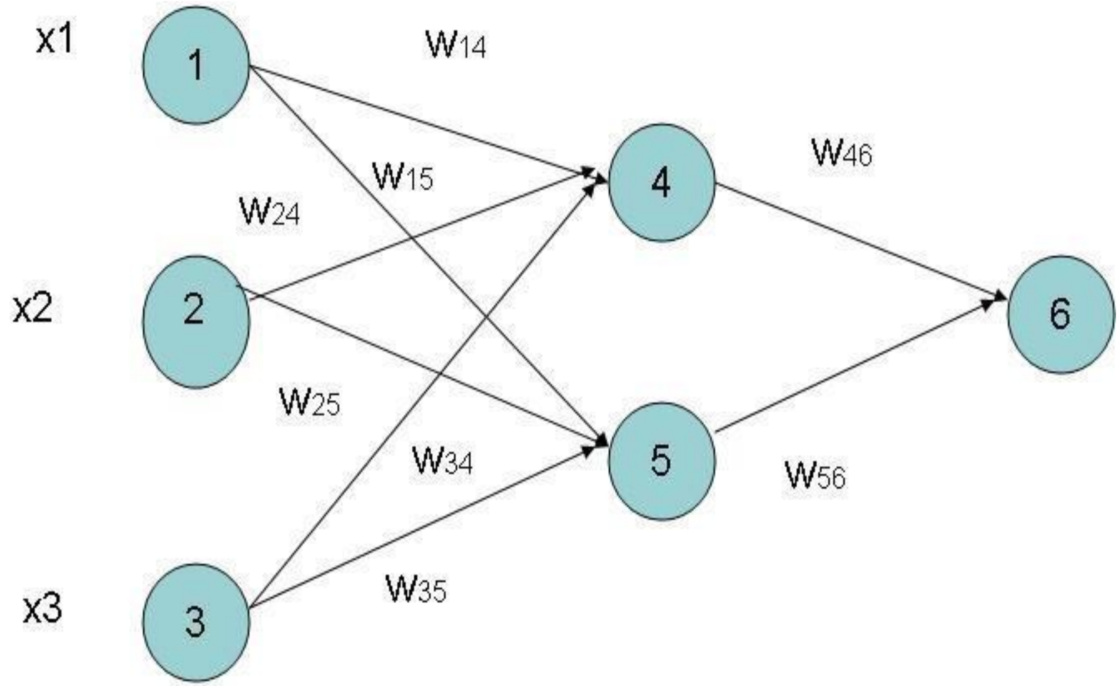
Initialize weights :

Random Numbers from -1.0 to 1.0

Initial Input and weight

x1	x2	x3	W_{14}	W_{15}	W_{24}	W_{25}	W_{34}	W_{35}	W_{46}	W_{56}
1	0	1	0.2	-0.3	0.4	0.1	-0.5	0.2	-0.3	-0.2

Example (cont..)



- Bias added to Hidden
- + Output nodes
- Initialize Bias
- Random Values from
- -1.0 to 1.0

- Bias (Random)

θ_4	θ_5	θ_6
-0.4	0.2	0.1

Net Input and Output Calculation

Unit j	Net Input I_j	Output O_j
4	**0.2 + 0 + 0.5 -0.4 = -0.7**	$O_j = \frac{1}{1+e^{0.7}}$ = 0.332
5	**-0.3 + 0 + 0.2 + 0.2 =0.1**	$O_j = \frac{1}{1+e^{-0.1}}$ = 0.525
6	**(-0.3)0.332-(0.2)(0.525)+0.1= -0.105**	$O_j = \frac{1}{1+e^{0.105}}$ = 0.475

Calculation of Error at Each Node

Unit j	Error j
6	0.475(1-0.475)(1-0.475) =0.1311 We assume T_6 = 1
5	0.525 x (1- 0.525)x 0.1311x (-0.2) = 0.0065
4	0.332 x (1-0.332) x 0.1311 x (-0.3) = -0.0087

Calculation of weights and Bias Updating

Learning Rate l =0.9

Weight	New Values
w_{46}	-0.3 + 0.9(0.1311)(0.332) = -0.261
w_{56}	-0.2 + (0.9)(0.1311)(0.525) = -0.138
w_{14}	0.2 + 0.9(-0.0087)(1) = 0.192
w_{15}	-0.3 + (0.9)(-0.0065)(1) = -0.306
……..similarly	………similarly
θ_6	0.1 +(0.9)(0.1311)=0.218
……..similarly	………similarly

Network Pruning and Rule Extraction

- Network pruning
  - Fully connected network will be hard to articulate
  - *N* input nodes, *h* hidden nodes and *m* output nodes lead to $h(m+N)$ weights
  - Pruning: Remove some of the links without affecting classification accuracy of the network

Applications-I

- Handwritten Digit Recognition
- Face recognition
- Time series prediction
- Process identification
- Process control
- Optical character recognition

Application-II

- Forecasting/Market Prediction: finance and banking
- Manufacturing: quality control, fault diagnosis
- Medicine: analysis of electrocardiogram data, RNA & DNA sequencing, drug development without animal testing
- Control: process, robotics

Remember…..

- ANNs perform well, generally better with larger number of hidden units
- More hidden units generally produce lower error
- Determining network topology is difficult
- Choosing single learning rate impossible
- Difficult to reduce training time by altering the network topology or learning parameters
- NN(Subset) often produce better results