

Back Propagation Network : Soft Computing Course Lecture 15 – 20, notes, slides
www.myreaders.info/ , RC Chakraborty, e-mail rcchak@gmail.com , Aug. 10, 2010
http://www.myreaders.info/html/soft_computing.html



Back Propagation Network

Soft Computing

Back-Propagation Network, topics : Background, what is back-prop network ? learning AND function, simple learning machines - Error measure , Perceptron learning rule, Hidden Layer, XOR problem. Back-Propagation Learning : learning by example, multi-layer feed-forward back-propagation network, computation in input, hidden and output layers, error calculation. Back-propagation algorithm for training network - basic loop structure, step-by-step procedure, numerical example.

Back Propagation Network

Soft Computing

Topics

(Lectures 15, 16, 17, 18, 19, 20 6 hours)

	Slides
1. Back-Propagation Network - Background	03-11
What is back-prop network ?; Learning : AND function; Simple learning machines - Error measure , Perceptron learning rule ; Hidden Layer , XOR problem.	
2. Back-Propagation Learning - learning by example	12-16
Multi-layer Feed-forward Back-propagation network; Computation of Input, Hidden and Output layers ; Calculation of Error.	
3. Back-Propagation Algorithm	17-32
Algorithm for training Network - Basic loop structure, Step-by-step procedure; Example: Training Back-prop network, Numerical example.	
4. References	33

Back-Propagation Network

What is BPN ?

- A single-layer neural network has many restrictions. This network can accomplish very limited classes of tasks.

Minsky and Papert (1969) showed that a two layer feed-forward network can overcome many restrictions, but they did not present a solution to the problem as "how to adjust the weights from input to hidden layer" ?

- An answer to this question was presented by Rumelhart, Hinton and Williams in 1986. The central idea behind this solution is that the errors for the units of the hidden layer are determined by back-propagating the errors of the units of the output layer.

This method is often called the **Back-propagation learning rule**.

Back-propagation can also be considered as a generalization of the delta rule for non-linear activation functions and multi-layer networks.

- Back-propagation is a systematic method of training multi-layer artificial neural networks.

1. Back-Propagation Network – Background

Real world is faced with a situations where data is incomplete or noisy. To make reasonable predictions about what is missing from the information available is a difficult task when there is no a good theory available that may to help reconstruct the missing data. It is in such situations the Back-propagation (Back-Prop) networks may provide some answers.

- A BackProp network consists of at least three layers of units :
 - an **input layer**,
 - at least one intermediate **hidden layer**, and
 - an **output layer**.
- Typically, units are connected in a **feed-forward** fashion with input units fully connected to units in the hidden layer and hidden units fully connected to units in the output layer.
- When a BackProp network is cycled, an input pattern is propagated forward to the output units through the intervening input-to-hidden and hidden-to-output weights.
- The output of a BackProp network is interpreted as a classification decision.

[Continued in next slide]

[Continued from previous slide]

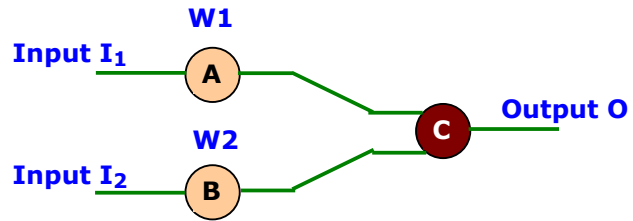
- With BackProp networks, learning occurs during a training phase. The steps followed during learning are :
 - each input pattern in a training set is applied to the input units and then propagated forward.
 - the pattern of activation arriving at the output layer is compared with the correct (associated) output pattern to calculate an error signal.
 - the error signal for each such target output pattern is then back-propagated from the outputs to the inputs in order to appropriately adjust the weights in each layer of the network.
 - after a BackProp network has learned the correct classification for a set of inputs, it can be tested on a second set of inputs to see how well it classifies untrained patterns.
- An important consideration in applying BackProp learning is how well the network generalizes.

4.1 Learning :

AND function

Implementation of AND function in the neural network.

AND		
X1	X2	Y
0	0	0
0	1	0
1	0	0
1	1	1



AND function implementation

- there are 4 inequalities in the AND function and they must be satisfied.

$$w_1 0 + w_2 0 < \theta, \quad w_1 0 + w_2 1 < \theta,$$

$$w_1 1 + w_2 0 < \theta, \quad w_1 1 + w_2 1 > \theta$$

- one possible solution :

if both weights are set to 1 and the threshold is set to 1.5, then

$$(1)(0) + (1)(0) < 1.5 \text{ assign } 0, \quad (1)(0) + (1)(1) < 1.5 \text{ assign } 0$$

$$(1)(1) + (1)(0) < 1.5 \text{ assign } 0, \quad (1)(1) + (1)(1) > 1.5 \text{ assign } 1$$

Although it is straightforward to explicitly calculate a solution to the AND function problem, but the question is "how the network can learn such a solution". That is, given random values for the weights can we define an incremental procedure which will cover a set of weights which implements AND function.

● **Example 1**

AND Problem

Consider a simple neural network made up of two inputs connected to a single output unit.

AND		
X1	X2	Y
0	0	0
0	1	0
1	0	0
1	1	1

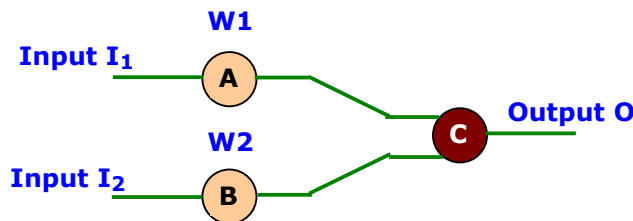


Fig A simple two-layer network applied to the AND problem

- the output of the network is determined by calculating a weighted sum of its two inputs and comparing this value with a threshold θ .
- if the net input (**net**) is greater than the threshold, then the output is **1**, else it is **0**.
- mathematically, the computation performed by the output unit is **net = w1 I1 + w2 I2** if **net > θ** then **O = 1**, otherwise **O = 0**.

● **Example 2**

Marital status and occupation

In the above example 1

- the input characteristics may be : **marital Status (single or married)** and their **occupation (pusher or bookie)**.
- this information is presented to the network as a 2-D binary input vector where 1st element indicates **marital status (single = 0, married = 1)** and 2nd element indicates **occupation (pusher = 0, bookie = 1)**.
- the output, comprise "**class 0**" and "**class 1**".
- by applying the **AND operator** to the inputs, we classify an individual as a member of the "**class 0**" only if they are both married and a bookie; that is the output is **1** only when both of the inputs are **1**.

2 Simple Learning Machines

Rosenblatt (late 1950's) proposed learning networks called **Perceptron**. The task was to discover a set of connection weights which correctly classified a set of binary input vectors. The basic architecture of the perceptron is similar to the simple AND network in the previous example.

A **perceptron** consists of a set of input units and a single output unit. As in the AND network, the output of the perceptron is calculated by comparing the net input $\text{net} = \sum_{i=1}^n w_i I_i$ and a threshold θ . If the net input is greater than the threshold θ , then the output unit is turned **on**, otherwise it is turned **off**.

To address the learning question, Rosenblatt solved two problems.

- first, defined a cost function which measured **error**.
- second, defined a procedure or a **rule** which reduced that error by appropriately adjusting each of the weights in the network.

However, the procedure (or **learning rule**) required to assesses the relative contribution of each weight to the total error.

The **learning rule** that Roseblatt developed, is based on determining the difference between the actual output of the network with the target output (**0** or **1**), called "**error measure**" which is explained in the next slide.

● **Error Measure** (learning rule)

Mentioned in the previous slide, the error measure is the difference between actual output of the network with the target output (**0** or **1**).

- If the input vector is correctly classified (i.e., zero error), then the weights are left unchanged, and the next input vector is presented.
- If the input vector is incorrectly classified (i.e., not zero error), then there are two cases to consider :

Case 1 : If the output unit is **1** but need to be **0** then

- ◇ the threshold is incremented by **1** (to make it less likely that the output unit would be turned on if the same input vector was presented again).
- ◇ If the input **I_i** is **0**, then the corresponding weight **W_i** is left unchanged.
- ◇ If the input **I_i** is **1**, then the corresponding weight **W_i** is decreased by **1**.

Case 2 : If output unit is **0** but need to be **1** then the opposite changes are made.

● Perceptron Learning Rule : Equations

The perceptron learning rules are governed by two equations,

- one that defines the change in the **threshold** and
- the other that defines change in the **weights**,

The **change in the threshold** is given by

$$\Delta \theta = - (t_p - o_p) = - d_p$$

where **p** specifies the presented input pattern,

o_p actual output of the input pattern **I_{pi}**

t_p specifies the correct classification of the input pattern i.e. target,

d_p is the difference between the target and actual outputs.

The **change in the weights** are given by

$$\Delta w_i = (t_p - o_p) I_{pi} = - d_p I_{pi}$$

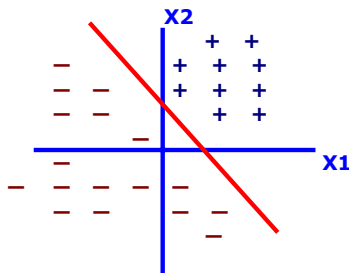
3 Hidden Layer

Back-propagation is simply a way to determine the error values in hidden layers. This needs to be done in order to update the weights.

The best example to explain where back-propagation can be used is the XOR problem.

Consider a simple graph shown below.

- all points on the right side of the line are +ve, therefore the output of the neuron should be **+ve**.
- all points on the left side of the line are **-ve**, therefore the output of the neuron should be **-ve**.



With this graph, one can make a simple table of inputs and outputs as shown below.

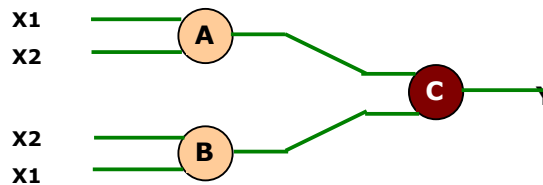
AND		
X1	X2	Y
1	1	1
1	0	0
0	1	0
0	0	0

Training a network to operate as an AND switch can be done easily through only one neuron (see previous slides)

But a XOR problem can't be solved using only one neuron.

If we want to train an XOR, we need 3 neurons, fully-connected in a feed-forward network as shown below.

XOR		
X1	X2	Y
1	1	0
1	0	1
0	1	1
0	0	0



2. Back Propagation Network

Learning By Example

Consider the Multi-layer feed-forward back-propagation network below.

The subscripts **I**, **H**, **O** denotes input, hidden and output neurons.

The weight of the arc between i^{th} input neuron to j^{th} hidden layer is V_{ij} .

The weight of the arc between i^{th} hidden neuron to j^{th} out layer is W_{ij}

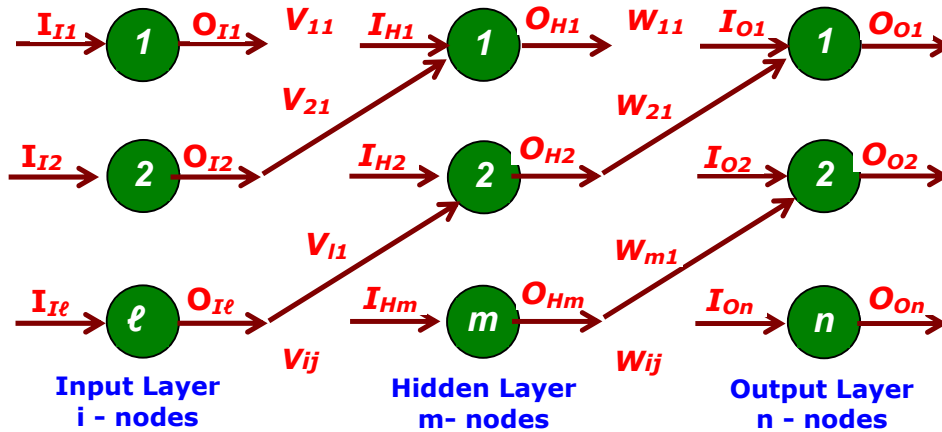


Fig Multi-layer feed-forward back-propagation network

The table below indicates an 'nset' of input and out put data.

It shows ℓ inputs and the corresponding n output data.

Table : 'nset' of input and output data

No	Input				Ouput			
	I_1	I_2	I_ℓ	O_1	O_2	O_n
1	0.3	0.4	0.8	0.1	0.56	0.82
2								
:								
nset								

In this section, over a three layer network the computation in the input, hidden and output layers are explained while the step-by-step implementation of the BPN algorithm by solving an example is illustrated in the next section.

1 Computation of Input, Hidden and Output Layers

(Ref. Previous slide, Fig. Multi-layer feed-forward back-propagation network)

● Input Layer Computation

Consider linear activation function.

If the output of the input layer is the input of the input layer and the transfer function is **1**, then

$$\{O\}_I = \{I\}_I$$

$\ell \times 1 \quad \ell \times 1 \quad (\text{denotes matrix row, column size})$

The hidden neurons are connected by synapses to the input neurons.

- Let V_{ij} be the weight of the arc between i^{th} input neuron to j^{th} hidden layer.
- The input to the hidden neuron is the weighted sum of the outputs of the input neurons. Thus the equation

$$I_{Hp} = V_{1p} O_{I1} + V_{2p} O_{I2} + \dots + V_{\ell p} O_{I\ell} \text{ where } (p = 1, 2, 3 \dots, m)$$

denotes weight matrix or connectivity matrix between input neurons and a hidden neurons as $[V]$.

we can get an input to the hidden neuron as $\ell \times m$

$$\{I\}_H = [V]^T \{O\}_I$$

$m \times 1 \quad m \times \ell \quad \ell \times 1 \quad (\text{denotes matrix row, column size})$

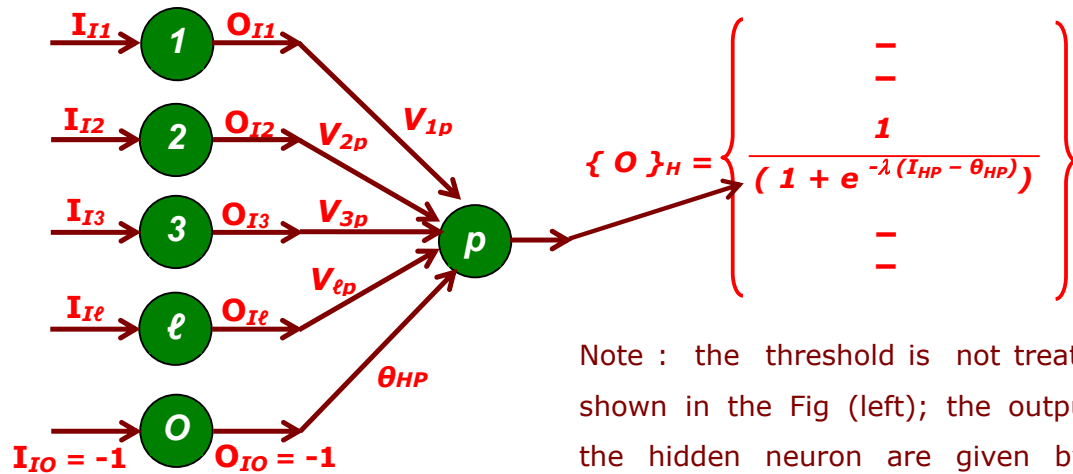
Hidden Layer Computation

Shown below the p^{th} neuron of the hidden layer. It has input from the output of the input neurons layers. If we consider transfer function as sigmoidal function then the output of the p^{th} hidden neuron is given by

$$O_{HP} = \frac{1}{(1 + e^{-\lambda(I_{HP} - \theta_{HP})})}$$

where O_{HP} is the output of the p^{th} hidden neuron,
 I_{HP} is the input of the p^{th} hidden neuron, and
 θ_{HP} is the threshold of the p^{th} neuron;

Note : a non zero threshold neuron, is computationally equivalent to an input that is always held at -1 and the non-zero threshold becomes the connecting weight value as shown in Fig. below.



Note : the threshold is not treated as shown in the Fig (left); the outputs of the hidden neuron are given by the above equation.

Fig. Example of Treating threshold in hidden layer

Treating each component of the input of the hidden neuron separately, we get the outputs of the hidden neuron as given by above equation .

The input to the output neuron is the weighted sum of the outputs of the hidden neurons. Accordingly, I_{Oq} the input to the q^{th} output neuron is given by the equation

$$I_{Oq} = W_{1q} O_{H1} + W_{2q} O_{H2} + \dots + W_{mq} O_{Hm}, \text{ where } (q = 1, 2, 3 \dots, n)$$

It denotes weight matrix or connectivity matrix between hidden neurons and output neurons as $[W]$, we can get input to output neuron as

$$\{I\}_O = [W]^T \{O\}_H$$

$n \times 1 \quad n \times m \quad m \times 1$ (denotes matrix row, column size)

Output Layer Computation

Shown below the q^{th} neuron of the output layer. It has input from the output of the hidden neurons layers.

If we consider transfer function as sigmoidal function then the output of the q^{th} output neuron is given by

$$O_{Oq} = \frac{1}{(1 + e^{-\lambda(I_{Oq} - \theta_{Oq})})}$$

where O_{Oq} is the output of the q^{th} output neuron,
 I_{Oq} is the input to the q^{th} output neuron, and
 θ_{Oq} is the threshold of the q^{th} neuron;

Note : A non zero threshold neuron, is computationally equivalent to an input that is always held at -1 and the non-zero threshold becomes the connecting weight value as shown in Fig. below.

Note : Here again the threshold may be tackled by considering extra O^{th} neuron in the hidden layer with output of -1 and the threshold value θ_{Oq} becomes the connecting weight value as shown in Fig. below.

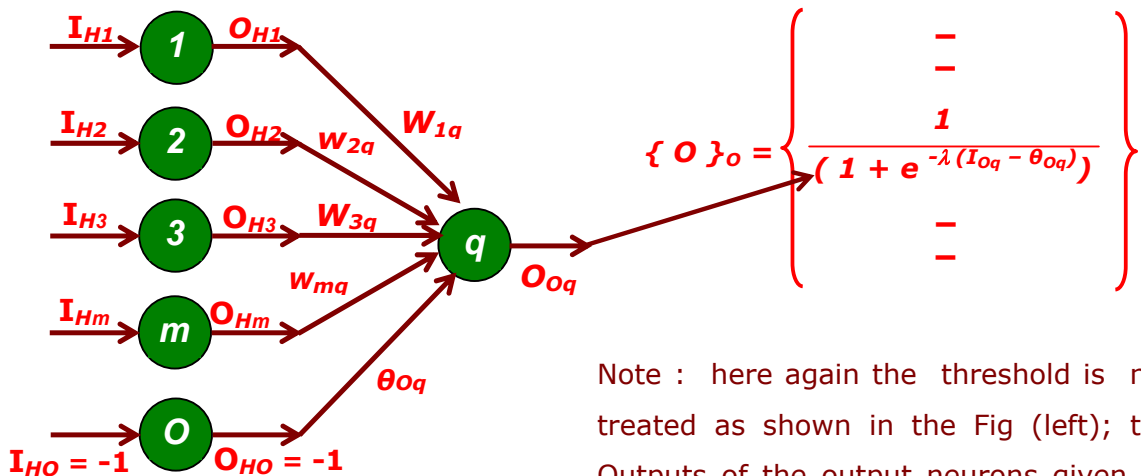


Fig. Example of Treating threshold in output layer

Note : here again the threshold is not treated as shown in the Fig (left); the Outputs of the output neurons given by the above equation.

2 Calculation of Error

(refer the earlier slides - Fig. "Multi-layer feed-forward back-propagation network" and a table indicating an 'nset' of input and output data for the purpose of training)

Consider any r^{th} output neuron. For the target out value T , mentioned in the table- 'nset' of input and output data" for the purpose of training, calculate output O .

The error norm in output for the r^{th} output neuron is

$$E^1_r = (1/2) e^2_r = (1/2) (T - O)^2$$

where E^1_r is $1/2$ of the second norm of the error e_r in the r^{th} neuron for the given training pattern.

e^2_r is the square of the error, considered to make it independent of sign **+ve** or **-ve**, ie consider only the absolute value.

The Euclidean norm of error E^1 for the first training pattern is given by

$$E^1 = (1/2) \sum_{r=1}^n (T_{or} - O_{or})^2$$

This error function is for one training pattern. If we use the same technique for all the training pattern, we get

$$E(V, W) = \sum_{r=1}^{nset} E^j(V, W, I)$$

where E is error function depends on $m(1+n)$ weights of $[W]$ and $[V]$.

All that is stated is an **optimization** problem solving, where the objective or cost function is usually defined to be maximized or minimized with respect to a set of parameters. In this case, the network parameters that optimize the error function E over the 'nset' of pattern sets $[I^{nset}, t^{nset}]$ are synaptic weight values $[V]$ and $[W]$ whose sizes are

$$\begin{matrix} [V] & \text{and} & [W] \\ l \times m & & m \times n \end{matrix}$$

3. Back-Propagation Algorithm

The benefits of hidden layer neurons have been explained. The hidden layer allows ANN to develop its own internal representation of input-output mapping. The complex internal representation capability allows the hierarchical network to learn any mapping and not just the linearly separable ones.

The step-by-step algorithm for the training of Back-propagation network is presented in next few slides. The network is the same, illustrated before, has a three layer. The input layer is with l nodes, the hidden layer with m nodes and the output layer with n nodes. An example for training a BPN with five training set have been shown for better understanding.

1 Algorithm for Training Network

The basic algorithm loop structure, and the step by step procedure of Back- propagation algorithm are illustrated in next few slides.

- **Basic algorithm loop structure**

Initialize the weights

Repeat

 For each training pattern

 "Train on that pattern"

 End

Until the error is acceptably low.

● **Back-Propagation Algorithm** - Step-by-step procedure

■ **Step 1 :**

Normalize the I/P and O/P with respect to their maximum values.

For each training pair, assume that in normalized form there are

l inputs given by **$\{ I \}_I$** and
 $l \times 1$

n outputs given by **$\{ O \}_o$**
 $n \times 1$

■ **Step 2 :**

Assume that the number of neurons in the hidden layers lie between **$1 < m < 21$**

■ Step 3 :

Let $[\mathbf{V}]$ represents the weights of synapses connecting input neuron and hidden neuron

Let $[\mathbf{W}]$ represents the weights of synapses connecting hidden neuron and output neuron

Initialize the weights to small random values usually from **-1 to +1**;

$$[\mathbf{V}]^0 = [\text{random weights}]$$

$$[\mathbf{W}]^0 = [\text{random weights}]$$

$$[\Delta \mathbf{V}]^0 = [\Delta \mathbf{W}]^0 = [\mathbf{0}]$$

For general problems λ can be assumed as **1** and threshold value as **0**.

■ **Step 4 :**

For training data, we need to present one set of inputs and outputs.

Present the pattern as inputs to the input layer $\{ \mathbf{I} \}_I$.

then by using linear activation function, the output of the input layer may be evaluated as

$$\{ \mathbf{O} \}_I = \{ \mathbf{I} \}_I$$

$l \times 1 \quad l \times 1$

■ **Step 5 :**

Compute the inputs to the hidden layers by multiplying corresponding weights of synapses as

$$\{ \mathbf{I} \}_H = [\mathbf{V}]^T \{ \mathbf{O} \}_I$$

$m \times 1 \quad m \times l \quad l \times 1$

■ **Step 6 :**

Let the hidden layer units, evaluate the output using the sigmoidal function as

$$\{ \mathbf{O} \}_H = \left\{ \begin{array}{c} - \\ - \\ \mathbf{1} \\ (\mathbf{1} + e^{- (IHi)}) \\ - \\ - \\ m \times 1 \end{array} \right\}$$

■ **Step 7 :**

Compute the inputs to the output layers by multiplying corresponding weights of synapses as

$$\{ \mathbf{I} \}_o = [\mathbf{W}]^T \{ \mathbf{O} \}_H$$

$n \times 1 \quad n \times m \quad m \times 1$

■ **Step 8 :**

Let the output layer units, evaluate the output using sigmoidal function as

$$\{ \mathbf{O} \}_o = \left\{ \begin{array}{c} - \\ - \\ \mathbf{1} \\ \hline (\mathbf{1} + e^{- (I_{oj})}) \\ - \\ - \end{array} \right\}$$

Note : This output is the network output

■ **Step 9 :**

Calculate the error using the difference between the network output and the desired output as for the j^{th} training set as

$$E^p = \sqrt{\frac{\sum (T_j - O_{oj})^2}{n}}$$

■ **Step 10 :**

Find a term $\{ d \}$ as

$$\{ d \} = \left\{ \begin{array}{c} - \\ - \\ (T_k - O_{ok}) O_{ok} (1 - O_{ok}) \\ - \\ - \\ n \times 1 \end{array} \right\}$$

■ Step 11 :

Find $[Y]$ matrix as

$$[Y] = \{O\}_H \langle d \rangle$$

$m \times n$ $m \times 1$ $1 \times n$

■ Step 12 :

Find

$$[\Delta W]^{t+1} = \alpha [\Delta W]^t + \eta [Y]$$

$m \times n$ $m \times n$ $m \times n$

■ Step 13 :

Find

$$\{e\} = [W] \{d\}$$

$m \times 1$ $m \times n$ $n \times 1$

$$\{d^*\} = \left\{ \begin{array}{c} - \\ - \\ e_i (O_{Hi}) (1 - O_{Hi}) \\ - \\ - \\ m \times 1 \quad m \times 1 \end{array} \right\}$$

Find $[X]$ matrix as

$$[X] = \{O\}_I \langle d^* \rangle = \{I\}_I \langle d^* \rangle$$

$1 \times m$ $l \times 1$ $1 \times m$ $l \times 1$ $1 \times m$

■ **Step 14 :**

Find
$$\begin{matrix} [\Delta \mathbf{V}]^{t+1} & = & \alpha [\Delta \mathbf{V}]^t + \eta [\mathbf{X}] \\ 1 \times m & & 1 \times m \quad 1 \times m \end{matrix}$$

■ **Step 15 :**

Find
$$\begin{aligned} [\mathbf{V}]^{t+1} &= [\mathbf{V}]^t + [\Delta \mathbf{V}]^{t+1} \\ [\mathbf{W}]^{t+1} &= [\mathbf{W}]^t + [\Delta \mathbf{W}]^{t+1} \end{aligned}$$

■ **Step 16 :**

Find error rate as

$$\text{error rate} = \frac{\sum E_p}{nset}$$

■ **Step 17 :**

Repeat steps 4 to 16 until the convergence in the error rate is less than the tolerance value

■ **End of Algorithm**

Note : The implementation of this algorithm, step-by-step 1 to 17, assuming one example for training BackProp Network is illustrated in the next section.

2 Example : Training Back-Prop Network

● Problem :

Consider a typical problem where there are 5 training sets.

Table : Training sets

S. No.	Input		Output
	I_1	I_2	O
1	0.4	-0.7	0.1
2	0.3	-0.5	0.05
3	0.6	0.1	0.3
4	0.2	0.4	0.25
5	0.1	-0.2	0.12

In this problem,

- there are two inputs and one output.
- the values lie between **-1** and **+1** i.e., no need to normalize the values.
- assume two neurons in the hidden layers.
- the NN architecture is shown in the Fig. below.

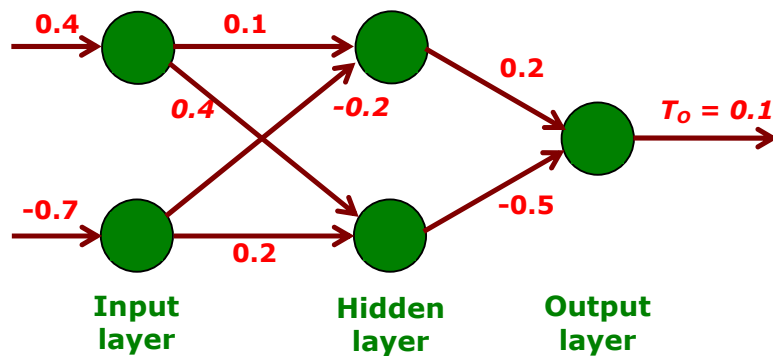


Fig. Multi layer feed forward neural network (MFNN) architecture with data of the first training set

The solution to problem are stated step-by-step in the subsequent slides.

- **Step 1 :** Input the first training set data

$$\{ \mathbf{O} \}_I = \{ \mathbf{I} \}_I = \begin{Bmatrix} 0.4 \\ -0.7 \end{Bmatrix}_{2 \times 1}$$

from training set s.no 1

- **Step 2 :** Initialize the weights as

(ref eq. of step 3 & Fig)

$$[\mathbf{V}]^0 = \begin{Bmatrix} 0.1 & 0.4 \\ -0.2 & 0.2 \end{Bmatrix}_{2 \times 2}; \quad [\mathbf{W}]^0 = \begin{Bmatrix} 0.2 \\ -0.5 \end{Bmatrix}_{2 \times 1}$$

from fig initialization

from fig initialization

- **Step 3 :** Find $\{ \mathbf{I} \}_H = [\mathbf{V}]^T \{ \mathbf{O} \}_I$ as

(ref eq. of step 5)

$$\{ \mathbf{I} \}_H = \begin{Bmatrix} 0.1 & -0.2 \\ -0.4 & 0.2 \end{Bmatrix} \begin{Bmatrix} 0.4 \\ -0.7 \end{Bmatrix} = \begin{Bmatrix} 0.18 \\ 0.02 \end{Bmatrix}$$

Values from step 1 & 2

■ **Step 4 :**

$$\{ \mathbf{O} \}_H = \left\{ \begin{array}{c} \frac{1}{(1 + e^{-(0.18)})} \\ \frac{1}{(1 + e^{-(0.02)})} \end{array} \right\} = \left\{ \begin{array}{c} 0.5448 \\ 0.505 \end{array} \right\}$$

Values from step 3 values

■ **Step 5 :**

$$\{ I \}_o = [W]^T \{ O \}_H = (0.2 - 0.5) \begin{Bmatrix} 0.5448 \\ 0.505 \end{Bmatrix} = - 0.14354$$

Values from step 2 , from step 4

■ **Step 6 :**

(ref eq. of step 8)

$$\{ O \}_o = \left\{ \frac{1}{(1 + e^{-(0.14354)})} \right\} = 0.4642$$

Values from step 5

■ **Step 7 :**

(ref eq. of step 9)

$$\text{Error} = (T_o - O_{o1})^2 = (0.1 - 0.4642)^2 = 0.13264$$

table first training set o/p   from step 6

■ **Step 8 :**

$$d = (T_o - O_{o1}) (O_{o1}) (1 - O_{o1})$$

$$= (0.1 - 0.4642) (0.4642) (0.5358) = - 0.09058$$

Training o/p

all from step 6

(ref eq. of step 11)

$$[Y] = \{ O \}_H (d) = \begin{Bmatrix} 0.5448 \\ 0.505 \end{Bmatrix} (- 0.09058) = \begin{Bmatrix} -0.0493 \\ -0.0457 \end{Bmatrix}$$

from values at step 4

from values at step 8 above

■ **Step 9 :**

(ref eq. of step 12)

$$[\Delta W]^1 = \alpha [\Delta W]^0 + \eta [Y] \quad \text{assume } \eta = 0.6$$

$$= \begin{Bmatrix} -0.02958 \\ -0.02742 \end{Bmatrix}$$

from values at step 2 & step 8 above

■ **Step 10 :**

(ref eq. of step 13)

$$\{ e \} = [W] \{ d \} = \begin{Bmatrix} 0.2 \\ -0.5 \end{Bmatrix} (- 0.09058) = \begin{Bmatrix} -0.018116 \\ -0.04529 \end{Bmatrix}$$

from values at step 2

from values at step 8 above

■ Step 11 :

$$\{ \mathbf{d}^* \} = \left\{ \begin{array}{l} (-0.018116) \quad (0.5448) \quad (1 - 0.5448) \\ (0.04529) \quad (0.505) \quad (1 - 0.505) \end{array} \right\} = \left\{ \begin{array}{l} -0.00449 \\ -0.01132 \end{array} \right\}$$

from values at step 10
at step 4
at step 8

■ Step 12 :

(ref eq. of step 13)

$$[\mathbf{X}] = \{ \mathbf{O} \}_I (\mathbf{d}^*) = \left\{ \begin{array}{l} 0.4 \\ -0.7 \end{array} \right\} \left(\begin{array}{cc} -0.00449 & 0.01132 \end{array} \right)$$

from values at step 1
from values at step 11 above

$$= \left\{ \begin{array}{cc} -0.001796 & 0.004528 \\ 0.003143 & -0.007924 \end{array} \right\}$$

■ Step 13 :

(ref eq. of step 14)

$$[\Delta \mathbf{V}]^1 = \alpha [\Delta \mathbf{V}]^0 + \eta [\mathbf{X}] = \left\{ \begin{array}{cc} -0.001077 & 0.002716 \\ 0.001885 & -0.004754 \end{array} \right\}$$

from values at step 2 & step 8 above

■ **Step 14 :**

$$[V]^1 = \begin{Bmatrix} 0.1 & 0.4 \\ -0.2 & 0.2 \end{Bmatrix} + \begin{Bmatrix} -0.001077 & 0.002716 \\ 0.001885 & -0.004754 \end{Bmatrix}$$

from values at step 2
from values at step 13

$$= \begin{Bmatrix} -0.0989 & 0.04027 \\ 0.1981 & -0.19524 \end{Bmatrix}$$

$$[W]^1 = \begin{Bmatrix} 0.2 \\ -0.5 \end{Bmatrix} + \begin{Bmatrix} -0.02958 \\ -0.02742 \end{Bmatrix} = \begin{Bmatrix} 0.17042 \\ -0.52742 \end{Bmatrix}$$

from values at step 2,
from values at step 9

■ **Step 15 :**

With the updated weights $[V]$ and $[W]$, error is calculated again and next training set is taken and the error will then get adjusted.

■ **Step 16 :**

Iterations are carried out till we get the error less than the tolerance.

■ **Step 17 :**

Once the weights are adjusted the network is ready for inferencing new objects .

4. References : Textbooks

1. "Neural Network, Fuzzy Logic, and Genetic Algorithms - Synthesis and Applications", by S. Rajasekaran and G.A. Vijayalaksmi Pai, (2005), Prentice Hall, Chapter 3, page 34-86.
2. "Soft Computing and Intelligent Systems Design - Theory, Tools and Applications", by Fakhreddine karray and Clarence de Silva (2004), Addison Wesley, chapter 5, page 249-293.
3. "Elements of Artificial Neural Networks", by Kishan Mehrotra, Chilukuri K. Mohan and Sanjay Ranka, (1996), MIT Press, Chapter 3, page 65-106.
4. "Fundamentals of Neural Networks: Architecture, Algorithms and Applications", by Laurene V. Fausett, (1993), Prentice Hall, Chapter 6, page 289-332.
5. "Neural Network Design", by Martin T. Hagan, Howard B. Demuth and Mark Hudson Beale, (1996) , PWS Publ. Company, Chapter 11-12, page 11-1 to 12-50.
6. Related documents from open source, mainly internet. An exhaustive list is being prepared for inclusion at a later date.