SOFT COMPUTING
Subject Code : 18MIT24C
Prepared by Dr. N.Thenmozhi

UNIT-II: Back propagation Networks: Architecture of a Back-Propagation Network – Back propagation Learning- Effect of Tuning parameters of the Back Propagation Neural Network – Selection of various parameters in BPN.

TEXT BOOK
1. S.Rajasekaran & G.A.Vijayalakshmi Pai, "Neural Networks, Fuzzy logic, and Genetic Algorithms
Synthesis and Applications, PHI, 2005.

REFERENCE BOOKS
1. James A. Freeman, David M.Skapura, "Neural Networks-Algorithms, Applications, and Programming Techniques", Pearson Education.
2. Fredric M. Ham, Ivica Kostanic, "Principles of Neuro computing for science of Engineering", TMCH.

2. Back propagation Networks

What is Backpropagation?

Back-propagation is the essence of neural net training. It is the method of fine-tuning the weights of a neural net based on the error rate obtained in the previous epoch (i.e., iteration). Proper tuning of the weights allows you to reduce error rates and to make the model reliable by increasing its generalization.

Backpropagation is a short form for "backward propagation of errors." It is a standard method of training artificial neural networks. This method helps to calculate the gradient of a loss function with respects to all the weights in the network.

Why We Need Backpropagation?

Most prominent advantages of Backpropagation are:

- Backpropagation is fast, simple and easy to program
- It has no parameters to tune apart from the numbers of input
- It is a flexible method as it does not require prior knowledge about the network
- It is a standard method that generally works well
- It does not need any special mention of the features of the function to be learned.

History of Backpropagation

- In 1961, the basics concept of continuous backpropagation were derived in the context of control theory by J. Kelly, Henry Arthur, and E. Bryson.
- In 1969, Bryson and Ho gave a multi-stage dynamic system optimization method.
- In 1974, Werbos stated the possibility of applying this principle in an artificial neural network.
- In 1982, Hopfield brought his idea of a neural network.
- In 1986, by the effort of David E. Rumelhart, Geoffrey E. Hinton, Ronald J. Williams, backpropagation gained recognition.
- In 1993, Wan was the first person to win an international pattern recognition contest with the help of the backpropagation method.

Disadvantages of using Backpropagation

- The actual performance of backpropagation on a specific problem is dependent on the input data.
- Backpropagation can be quite sensitive to noisy data
- You need to use the matrix-based approach for backpropagation instead of mini-Batch

2.1 Architecture of a Backpropagation Network

The Perceptron Model

The **perceptron** is a classification algorithm. Specifically, it works as a linear binary classifier. It was invented in the late 1950s by Frank Rosenblatt. The **perceptron** basically works as a threshold function — non-negative outputs are put into one class while negative ones are put into the other class.

Architecture of a Backpropagation Network

The perceptron model

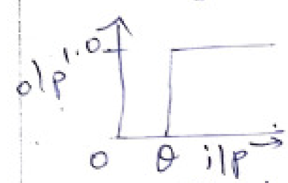
Rosenblatts perceptron was introduced for linearly inseperable (or non linearly seperable) problem.

The initial approach to solve this, more then one perceptron, each setup identifying small linearly seperable section of i/ps. and then combining their outputs into another perceptron. XOR problem was solved.

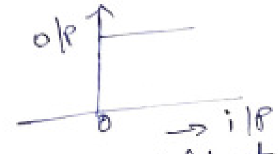
Each neuron in the structure takes the weighted sums of inputs, thresholds it and outputs either a one or zero. for the perceptron in the first layer, the i/p comes from the actual i/p of the problem. for 2nd layer the i/p are o/p of the first layer. The perceptrons of the 2nd layer donot know which of the real i/ps from the first layer were on or off.

It is impossible to strengthen the connections between active i/ps & strengthen the correct parts of the network. The actual i/ps are effectively masked off from the o/ps units of the intermediate layer.

The 2 states of neuron being on or off., do not give us any indication of the scale by which we have to adjust the weights.



Threshold at θ



Threshold at 0

The hard-hitting threshold functions remove the information that is needed if the network is to successfully learn. Hence the network is unable to determine which of the i/p weights should be increased and which one should not and so, it is unable to work to produce a better solution next time.

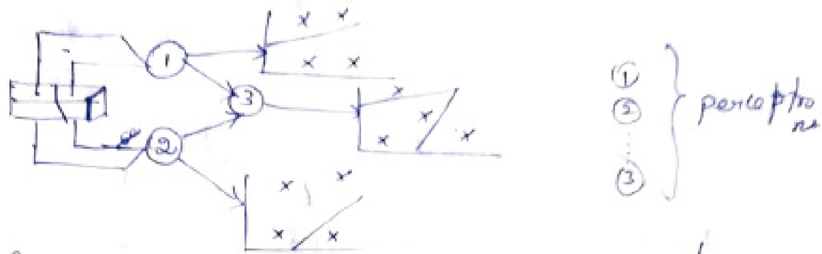
Using the step function as the thresholding process is to adjust it slightly and to use a slightly different nonlinearity.

2. The solution

If we smoothen the threshold function, it more or less turns on or off as before but has a sloping region in the middle that will give us some information on the inputs.

We will be able to determine when it need to strengthen or weaken the relevant weights

Now, the new will be able to learn as required. A couple of possibilities for the new thresholding function are given as



Combining perceptrons to solve XOR problem.

1. If the value of the output = 1, then if the i/p exceeds the value of the threshold a lot.
Else it is = 0 if the i/p is far less than the threshold.

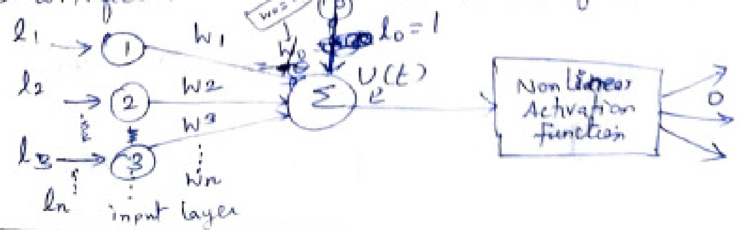
2. when i/p & threshold are almost same, the o/p of the neuron will have a value between 0 & 1. (ie o/p to the neurons can be related to i/p).

⇒ Based on the biological neuron, an artificial neuron receives much i/p representing the o/p of the other neurons.

⇒ Each i/p is multiplied by the corresponding weight analogous to synaptic strengths.

⇒ All of these weighted i/ps are then summed up & passed through an activation function to determine the neuron o/p.

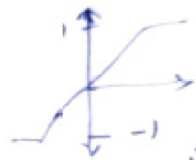
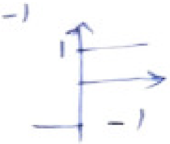
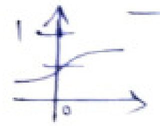
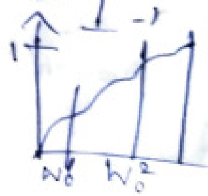
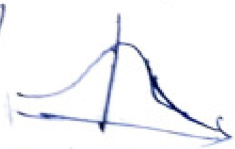
∴ The artificial neural model is as follows.



$u(t) = w_1 I_1 + \dots$

$$u = \{W\}^T \{I\}$$

Typical nonlinear activation operators

Type	Equation	Function Form
Linear	$O = gI$ $g = \tan\phi$	
Piecewise Linear	$O = \begin{cases} 1 & \text{if } mI > 1 \\ gI & \text{if } \lvert mI \rvert < 1 \\ -1 & \text{if } mI > -1 \end{cases}$	
Hard Limiter	$O = \operatorname{sgn}[I]$	
Unipolar Sigmoidal	$O = \dfrac{1}{(1 + \exp(-\lambda I))}$	
Bipolar Sigmoidal	$O = \tanh[\lambda I]$	
Unipolar multimodal	$O = \frac{1}{2}\left[1 + \frac{1}{M}\sum_{m=1}^{M} \tanh(g^m(I - W_0^m))\right]$	
Radial Bases Function (RBF)	$O = \exp(I)$ $I = \left[\dfrac{-\sum_{i=1}^{N}(w_i(t) - X_i(t))^2}{2\sigma^2}\right]$	

Considering threshold θ, the relative input to the neuron is given by.

$$u(t) = W_1 I_1 + W_2 I_2 + \dots + W_n I_n - \theta$$
$$= \sum_{i=0}^{n} W_i I_i \quad \text{where } W_0 = -\theta\ ;\ I_0 = 1$$

The o/p using the nonlinear transfer function 'f' is given by

$$O = f(u)$$

The activation function $f(u)$ is chosen as a nonlinear function to emulate the nonlinear behaviour of conduction current mechanism in a biological neuron. Artificial neuron is not intended to be a xerox copy of the biological neuron, many forms of nonlinear functions are used in various engineering applications. From the above Neurons with sigmoidal fns greater resemblence to biological neuron Hard limiter & radial basis fns are also equally popular.

Even if the sigmoidal fn is differentiated, it gives continuous Val of the output.

3. Single Layer Artificial Neural Network

Single Layer feed forward neural network consisting of an i/p layer to receive the i/p & an o/p layer to o/p the vector respectively.

i/p layers — n neurons & o/p layer — m neurons

Indicate the weight of the synapse connecting i^{th} i/p neuron to the j^{th} o/p neurons as W_{ij}. The i/ps of the i/p layer & the corresponding o/ps of the o/p layer are given as

$$I_i = \begin{bmatrix} I_{I1} \\ I_{I2} \\ \vdots \\ I_{In} \end{bmatrix}_{n\times 1} \qquad O_0 = \begin{Bmatrix} O_{01} \\ O_{02} \\ \vdots \\ O_{0m} \end{Bmatrix}_{m\times 1}$$

Linear transfer function for the neurons in the i/p layer and the unipolar sigmoidal function for the neurons in the o/p layer.

$$\underset{n\times 1}{\{O_1\}} = \underset{m\times 1}{\{I_1\}} \quad (\text{linear transfer function})$$

$$I_{0j} = W_{1j} I_{11} + W_{2j} I_{12} + \cdots W_{nj} I_{1N}$$

Hence, the i/p to the o/p layer can be given as

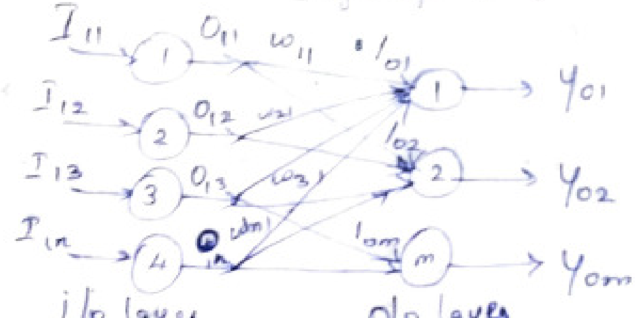
$$\underset{m\times 1}{\{I_0\}} = \underset{m\times n}{[W]^T} \{O_1\} = [W]^T \underset{n\times 1}{\{I_1\}}$$

→ Weight matrix or Connection matrix.

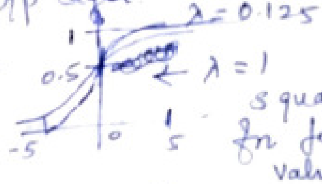
$I \rightarrow \boxed{f(I,W)} \rightarrow O$

Block diagram of a single layer feed forward neural network.

Single layer feedforward neural net



Unipolar Sigmoidal $\longrightarrow$



squashed-S fn for various values of λ.

& slope of this fn is

squashed-S slope

Neurons in the o/p layer, the o/p is given by

$$O_{ok} = \frac{1}{(1+e^{-\lambda I_{ok}})} \rightarrow \text{Non Linear activation fn.}$$

$$\{O_0\} = f\{WI\}$$

λ is known as sigmoidal gain.

Each activation value is in turn a scalar product of the i/p with respect to weight vectors.

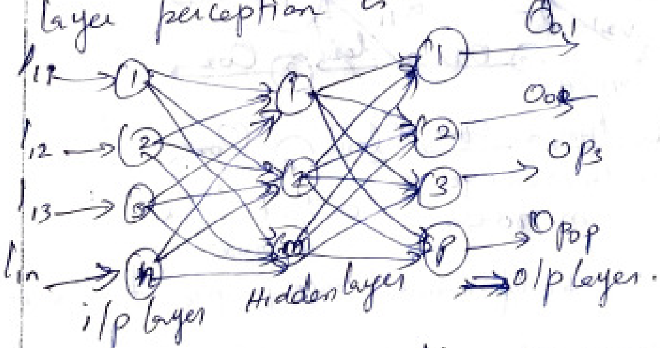
$\therefore$ Sigmoidal fn is

$$f(I) = \frac{1}{(1+e^{-\lambda I})}$$

& $f'(I) = \lambda f(I)(1-f(I))$

4. Model for multilayer perceptron

Adapted perceptrons are arranged ~~in~~ layers are called multilayer perceptron. This model has 3 layers : an i/p layer & o/p layer & a layer in between not connected directly to the i/p or the o/p is called hidden layer.
For the perceptrons in the i/p layer, use linear transfer function & for the hidden & o/p layer we use sigmoidal or squashed-S functions.
The i/p layer serves to distribute the values they receive to the next layer & so, does not perform a weighted sum or theshold. Because we have modified the single layer perceptron by changing the nonlinearity from a step function to sigmoidal function and added a hidden layer, so we are forced to change the learning rules as well.
So the nw should be able to learn to recognise more complex things. The i/p – o/p mapping of multi layer perception is



ie

$$O = N_3\left[N_2\left[N_1\left[I\right]\right]\right]$$

N_1, N_2, N_3 are nonlinear mapping provided by i/p, hidden & o/p layer respectivity

multilayer perceptron provides
no increase in computational power
over single layer.
i/p layer represents raw information that is
fed into the n/w. Hidden layer is deter-
-mined by the activities of the neurons
in the i/p layer & connecting weights between
i/p & hidden units.
Ilty the activity of o/p units depends on the
activity in the hidden layer & the weight
between the hidden & o/p layers.
Neurons in the hidden layer are free to
construct their own representations of the
o/p.

eg 3 Layer ANN ⇒

Backpropagation Learning

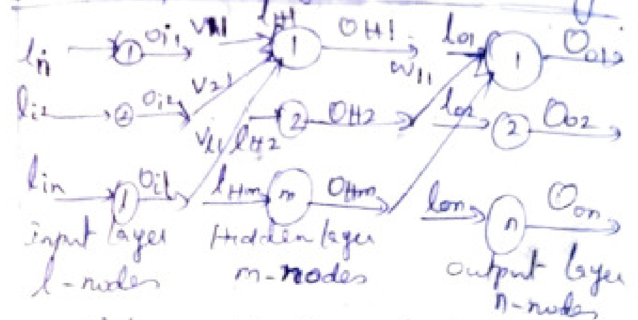


fig. multilayer feedforward backpropagation network

consider the problem ⇒ an n set of 'l' i/p
corresponding n set of 'n' o/p data

n set of i/p and output data

NO	Input			output		
1 → 2 . n set	I_1	I_2 ...	I_l	O_1 ...	O_2 ...	O_n
	0.3	0.4 ...	0.8	0.1	0.56	0.82

Input Layer computation

consider linear activation function, the o/p of the i/p layer is i/p of i/p Layer. $(g = \tan\phi = 1)$

Taking one set of data

$$\{O\}_I = \{I\}_I$$
$$l \times 1 = l \times 1$$

The hidden Layer is connected by synapses to i/p neurons. & V_{ij} = weight of the arc between ith i/p neuron to jth hidden neuron

The i/p of hidden neuron = weighted sum of the o/p of the i/p neurons to get I_{Hp}

ie pth hidden neuron as

$$I_{Hp} = V_{1p} O_{I1} + V_{2p} O_{I2} + \cdots V_{lp} O_{Il}$$
$$(p = 1, 2, 3 \ldots m)$$

defined in weighted matrix. $[V]_{l \times m}$.

We can get an i/p to the hidden neuron as

$$\{I\}_H = \{V\}^T \{O\}_I$$
$$m \times l \quad m \times l \quad l \times 1$$

Hidden Layer Computation

Sigmoidal fn. $O_{Hp} = \dfrac{1}{1 + e^{-\lambda(I_{Hp} - \theta_{Hp})}}$. ie o/p of the pth Hidden neuron

O_{Hp} – o/p of the pth hidden neuron

I_{Hp} – i/p " " "

θ_{Hp} – threshold of the pth neuron.

A non zero threshold neuron = to an i/p always at –1

& " " becomes the connecting weight values.

The derivations → o/p to the hidden neuron is

$$\{O\}_H = \left\{ \frac{1}{1+e^{-\lambda(I_{Hp}-\theta_{Hp})}} \right\}$$

The weighted sum of the o/p of the hidden neurons

To get I_{Oq} (input of qth output neuron)

$$\underset{n\times 1}{\{I\}_O} = \underset{n\times m}{\{W\}^T} \; \underset{m\times 1}{\{O\}_H}$$

o/p Layer Computation.

(1) →

2.3 Training Algorithm

For training, BPN will use binary sigmoid activation function. The training of BPN will have the following three phases.

- **Phase 1 –** Feed Forward Phase
- **Phase 2 –** Back Propagation of error
- **Phase 3 –** Updating of weights

All these steps will be concluded in the algorithm as follows

Step 1 – Initialize the following to start the training –

- Weights
- Learning rate αα

For easy calculation and simplicity, take some small random values.

Step 2 – Continue step 3-11 when the stopping condition is not true.

Step 3 – Continue step 4-10 for every training pair.

Phase 1

Step 4 – Each input unit receives input signal $\mathbf{x_i}$ and sends it to the hidden unit for all **i = 1 to n**

Step 5 – Calculate the net input at the hidden unit using the following relation –

Qinj=b0j+∑i=1nxivijj=1topQinj=b0j+∑i=1nxivijj=1top

Here $\mathbf{b_{0j}}$ is the bias on hidden unit, $\mathbf{v_{ij}}$ is the weight on **j** unit of the hidden layer coming from **i** unit of the input layer.

Now calculate the net output by applying the following activation function

Qj=f(Qinj)Qj=f(Qinj)

Send these output signals of the hidden layer units to the output layer units.

Step 6 – Calculate the net input at the output layer unit using the following relation –

yink=b0k+∑j=1pQjwjkk=1tomyink=b0k+∑j=1pQjwjkk=1tom

Here $\mathbf{b_{0k}}$ is the bias on output unit, $\mathbf{w_{jk}}$ is the weight on **k** unit of the output layer coming from **j** unit of the hidden layer.

Calculate the net output by applying the following activation function

yk=f(yink)yk=f(yink)

Phase 2

Step 7 – Compute the error correcting term, in correspondence with the target pattern received at each output unit, as follows –

δk=(tk−yk)f′(yink)δk=(tk−yk)f′(yink)

On this basis, update the weight and bias as follows –

Δvjk=αδkQijΔvjk=αδkQij

Δb0k=αδkΔb0k=αδk

Then, send δkδk back to the hidden layer.

Step 8 – Now each hidden unit will be the sum of its delta inputs from the output units.

δinj=∑k=1mδkwjkδinj=∑k=1mδkwjk

Error term can be calculated as follows –

δj=δinjf′(Qinj)δj=δinjf′(Qinj)

On this basis, update the weight and bias as follows –

Δwij=αδjxiΔwij=αδjxi

Δb0j=αδjΔb0j=αδj

Phase 3

Step 9 – Each output unit ***(y_kk = 1 to m)*** updates the weight and bias as follows –

vjk(new)=vjk(old)+Δvjkvjk(new)=vjk(old)+Δvjk

b0k(new)=b0k(old)+Δb0kb0k(new)=b0k(old)+Δb0k

Step 10 – Each output unit ***(z_jj = 1 to p)*** updates the weight and bias as follows –

wij(new)=wij(old)+Δwijwij(new)=wij(old)+Δwij

b0j(new)=b0j(old)+Δb0jb0j(new)=b0j(old)+Δb0j

Step 11 – Check for the stopping condition, which may be either the number of epochs reached or the target output matches the actual output.

Generalized Delta Learning Rule

Delta rule works only for the output layer. On the other hand, generalized delta rule, also called as **back-propagation** rule, is a way of creating the desired values of the hidden layer.

Mathematical Formulation

For the activation function yk=f(yink)yk=f(yink) the derivation of net input on Hidden layer as well as on output layer can be given by

yink=∑iziwjkyink=∑iziwjk

And yinj=∑ixivijyinj=∑ixivij

Now the error which has to be minimized is

E=12∑k[tk−yk]2E=12∑k[tk−yk]2

By using the chain rule, we have

∂E∂wjk=∂∂wjk(12∑k[tk−yk]2)∂E∂wjk=∂∂wjk(12∑k[tk−yk]2)

=∂∂wjk· 12[tk−t(yink)]2· =∂∂wjk· 12[tk−t(yink)]2·

=−[tk−yk]∂∂wjkf(yink)=−[tk−yk]∂∂wjkf(yink)

=−[tk−yk]f(yink)∂∂wjk(yink)=−[tk−yk]f(yink)∂∂wjk(yink)

=−[tk−yk]f′(yink)zj=−[tk−yk]f′(yink)zj

Now let us say δk=−[tk−yk]f′(yink)δk=−[tk−yk]f′(yink)

The weights on connections to the hidden unit $\mathbf{z_j}$ can be given by –

∂E∂vij=−∑kδk∂∂vij(yink)∂E∂vij=−∑kδk∂∂vij(yink)

Putting the value of yinkyink we will get the following

δj=−∑kδkwjkf′(zinj)δj=−∑kδkwjkf′(zinj)

Weight updating can be done as follows –

For the output unit

$$\Delta wjk=-\alpha\partial E\partial wjk\Delta wjk=-\alpha\partial E\partial wjk$$

$$=\alpha\delta kzj=\alpha\delta kzj$$

For the hidden unit

$$\Delta vij=-\alpha\partial E\partial vij\Delta vij=-\alpha\partial E\partial vij$$

$$=\alpha\delta jxi$$

2.4 Effect of Tuning parameters of the Back Propagation Neural Network

- Sigmoidal gain
- Threshold value

2.5 Selection of various parameters in BPN

- Number of hidden nodes
- Momentum coefficient
- Sigmoidal Gain
- Local Minima
- Learning Coefficient