CHAPTER 5

Unit V

Artificial Neural Network

Syllabus

5.1 Introduction : Fundamental concept : Basic Models of Artificial Neural Networks : Important Terminologies of ANNs : McCulloch-Pitts Neuron

5.2 Neural Network Architecture: Perceptron, Single layer Feed Forward ANN, Multilayer Feed Forward ANN, Activation functions, Supervised Learning : Delta learning rule, Back Propagation algorithm.

5.3 Un-Supervised Learning algorithm : Self Organizing Maps

5.1 Fundamental Concept - Artificial Neural Networks

5.1.1 Artificial Neural Networks

- Artificial Neural Networks (ANNs) are simplified models of the biological nervous system. They basically mimic the working of a human brain.
- An ANN, in general, is a highly interconnected network of a large number of processing elements called neurons.
- An ANN can be considered as a highly parallel network. Distributed processing is a typical feature of a neural network.
- Neural networks work on the principle of learning by examples. They are presented with known examples of a problem called 'training set', to acquire knowledge about the problem. After training, the network can be effectively employed in solving instances of the problem previously unknown to the network.

Architecture of a simple artificial neural net

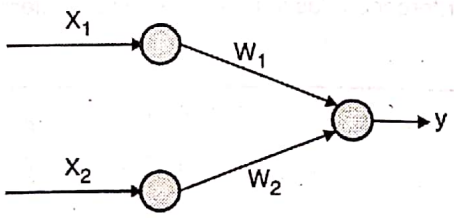


Fig: 5.1.1 Architecture of a simple artificial neural net

- As shown in Fig 5.1.1, There are two input neurons and one output neuron.
- Each neuron has an internal state of its own called **'activation of a neuron'.**
- The activation signal of one neuron is transmitted to another neuron.
- This Activation of a neuron can be considered as a function of the inputs a neuron receives.
- Consider a set of neurons; say X_1 and X_2, transmitting signals to another neuron, Y.
- As shown in Fig 5.1.1 there are two input neurons X_1 and X_2. They transmit signals to output neuron Y.
- Input neurons X_1 and X_2 are connected to the output neuron Y, over a weighted interconnection links (W_1 and W_2).
- For the above simple neural net architecture, the net input is calculated as :

$$Y_{in} = W_1 x_1 + W_2 x_2$$

Scanned by CamScanner

- Where x_1 and x_2 are activations of the input neurons x_1 and x_2 respectively.
- The output y of the output neuron Y can be obtained by applying activations over the net input, i.e.,

$$y = f(y_{in})$$

- The function that we apply over the net input is called activation function.

5.1.2 Biological Neural Networks

- The fundamental computing element in the human brain is called a neuron. There are approximately 10^{11} neurons in a human brain.
- Communication between neurons happens through a connection network of **axons** and **synapses**. There are approximately 10^4 synapses per neuron.
- Neurons communicate with each other by means of electrical impulses.
- A chemical environment surrounds the network of neurons.

Information flow in nervous system

- The huge neural network in the brain is made up of extremely complex interconnections and has a very complicated structure.
- Fig. 5.1.2 shows information flow in the human nervous system.
- As shown in Fig. 5.1.2, sensory receptors provide input to the network. Receptors deliver stimuli both from within the body and from sensory organs, when the stimuli originate in the external world.
- The stimuli are in the form of electrical impulses that convey information into the network of neurons.

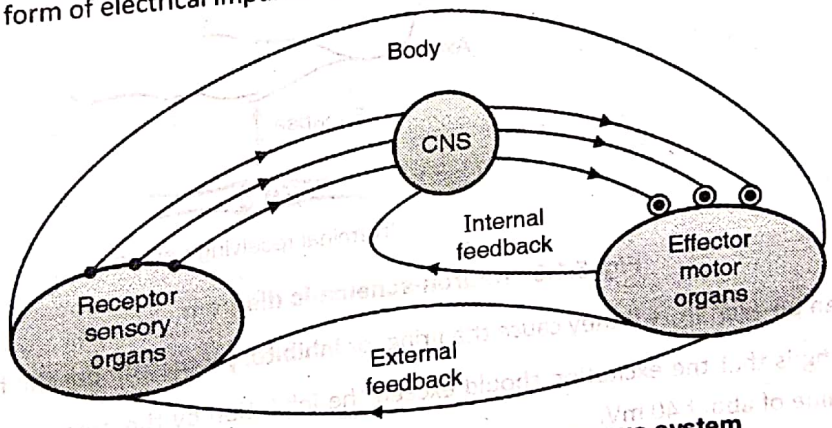


Fig. 5.1.2: Flow of information in the nervous system

- As shown in Fig. 5.1.2, effectors are controlled on account of information processing in the Central Nervous System (CNS). These effectors then produce human responses in the form of various actions.
- When necessary, commands are generated and transmitted to the motor organs.
- Motor organs are monitored in the CNS by both internal and external feedback.
- The overall nervous system structure has the characteristics of a **closed loop system.**

Biological Neuron

- A typical cell has 3 major regions: the cell body, called the soma, the axon and the dendrites. A schematic of a typical biological neuron is shown in Fig.5.1.3.
- Dendrites form a dendrite tree, which is a very fine bush of thin fibres around the neuron's body.

Scanned by CamScanner

- Dendrites receive information from neurons through axons.
- An axon is a long cylindrical fibre that carries impulses from the neuron.
- Towards the end, an axon splits into fine branches. Each branch of an axon terminates in a small end bulb, almost touching the dendrites of neighbouring neurons.
- The contact between an axon and a dendrite is called a **synapse**. Via the synapse, a neuron introduces its signal to a neighbouring neuron. The receiving neuron either generates an impulse to its axon or produces no responses. The neuron is able to respond to the total of its inputs aggregated within a short time interval called the **period of latent summation.**
- The neuron generates a response and sends it to its axon only if the conditions necessary for firing are fulfilled.

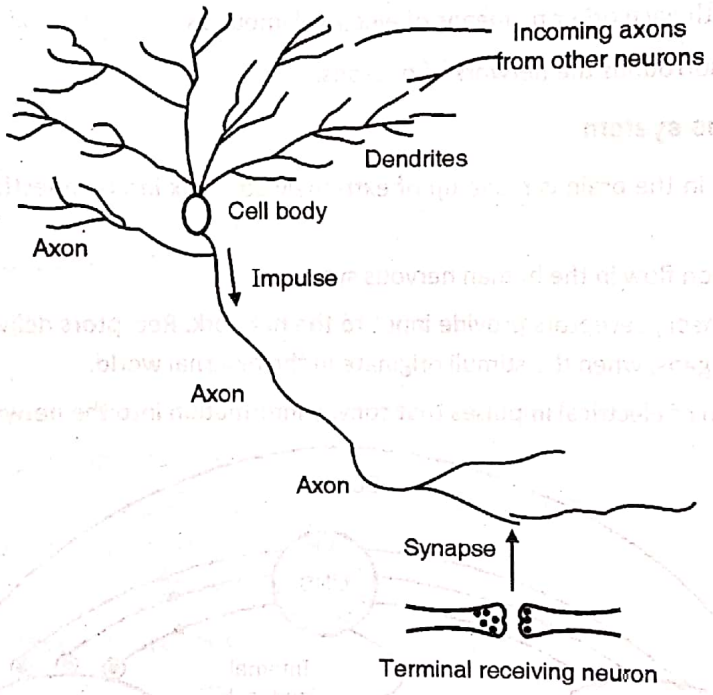


Fig. 5.1.3 : Neuron-schematic diagram

- Incoming impulses can be **excitatory** if they cause the firing, or **inhibitory** if they hinder the firing of the response.
- The condition for firing is that the excitation should exceed the inhibition by the amount called the **threshold** of the neuron, typically a value of about 40 mV.
- The incoming impulse to a neuron can only be generated by neighbouring neurons and by the neuron itself.
- Hence, impulses that are closely spaced in and arrive synchronously are more likely to cause the neuron to fire.
- The characteristic feature of the biological neuron is that the signals generated do not differ significantly in magnitude; the signal in the nerve fibre is either absent or has the maximum value.
- The information is transmitted between the nerve cells by means of binary signals.
- After an axon fibre finishes generating a pulse, it goes into a state of total non-excitability for a certain amount of time, called the **refractory period.** The nerve does not conduct any signals during the refractory period, even if the excitation intensity is very high.
- The time for modelling biological neurons is of the order of milliseconds. However, the refractory period is not uniform over the cells.

Scanned by CamScanner

5.1.3 Brain vs. Computer

Human brain and computer can be compared based on following parameters.

	Biological Neuron	Artificial Neuron
Speed	Low compared to ANN. The speed is in milliseconds. Human Brain may fatigue if too much information and stress is presented.	Speed is High (in nanoseconds). They do not experience 'Fatigue'.
Size	The no. of neurons in Brains is approximately 10^{11} and total no. of connections are 10^{15}. Thus the complexity of human brain is very high .	The size of ANN depends on the application (usually hundreds or thousands) chosen and the developed who develops the application.
Complexity	Can perform Massive parallel computation simultaneously.	Can also perform Massive parallel computation simultaneously but much more faster than Human brain.
Storage capacity	Stores the information in the synapse.	Stores the information in continuous memory locations.
Tolerance	Biological neuron networks due to their topology are fault-tolerant. Information is stored redundantly so minor failures will not result in memory loss.	Artificial neural networks are not modelled for fault tolerance or self-regeneration.
Control Mechanism	No specific control mechanism external to the computing task.	There is a control unit for controlling computing activities
Power consumption	The brain consumes about 20% of all the human body's energy. An adult brain operates on about 20 watts.	A single Nvidia GPU runs on 250 watts alone, and requires a power supply. Machines are way less efficient than biological systems.
Learning	In human brain, Brain fibres grow and reach out to connect to other neurons; neuroplasticity allows new connections to be created and synapses may strengthen or weaken based on their importance	Artificial neural networks in the other hand, have a predefined model, where no further neurons or connections can be added or removed.

Comparison between Biological Neuron and ANN

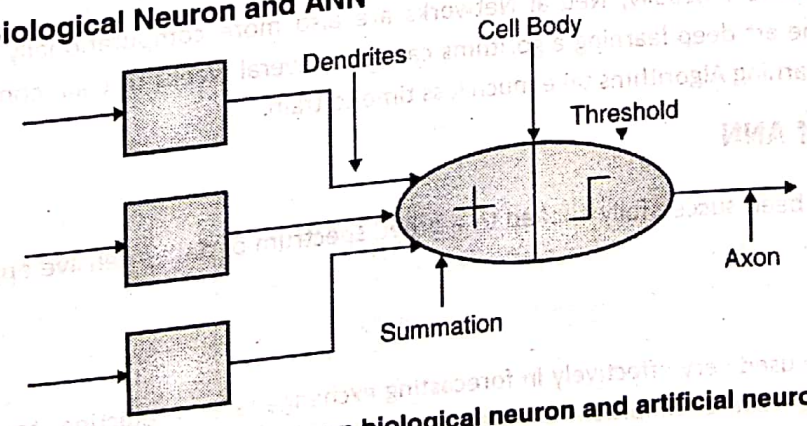


Fig. 5.1.4 : Analogy between biological neuron and artificial neuron

Fig. 5.1.4 shows analogy between Biological and artificial Neuron.

Scanned by CamScanner

- The dendrites in the Biological Neural Network are analogous to the weighted inputs based on their synaptic interconnection in the Artificial Neural Network.
- The cell body is comparable to the artificial neuron unit in the ANN which also comprises of summation and threshold unit.
- Axon carries output that is analogous to the output unit in case of Artificial Neural Network. So, ANN is modeled using the working of basic biological neurons.

5.1.4 Features of ANN

1. **Adaptive learning :** An ANN possess the ability to learn from the existing environment and also adapt to the new environment.
2. **Self-organization :** An ANN can create its own organization or representation of the information it receives during learning time.
3. **Real-time operation :** Since ANN computations may be carried out in parallel, they can be used for real time applications. Special hardware devices are being designed and manufactured to take advantage of this capability of ANNs and to reduce the response time.
4. **Fault tolerance :** Neural networks are fault tolerant in the sense even if some portions of the neural net is removed (For example some connections are removed), there will be only a small degradation in the neural network performance.
5. **Generalization :** After learning from the available inputs and their relationships, ANN has capability to infer unseen relationships on unseen data, thus making the model generalize.
6. **Non-linearity :** ANNs have the ability to learn and model non-linear and complex relationships. In most of the real-life problems, many of the relationships between inputs and outputs are non-linear as well as complex.
7. **Parallel distributed processing :** ANNs have massive parallelism which makes them very efficient.

5.1.5 Disadvantages of ANN

1. The best-known disadvantage of Neural Networks is their "black box" nature. This means that you don't know how and why your NN came up with a certain output. For example, when you put in an image of a cat into a neural network and it predicts it to be a car, it is very hard to understand what caused it to came up with this prediction.
2. Neural Networks usually require much more data than traditional Machine Learning algorithms, as in at least thousands if not millions of labeled samples.
3. Computationally Expensive : Usually, Neural Networks are also more computationally expensive than traditional algorithms. State of the art deep learning algorithms can take several weeks to train completely from scratch. Most traditional Machine Learning Algorithms take much less time to train.

5.1.6 Applications of ANN

Neural networks have been successfully applied to a broad spectrum of data-intensive applications. Few of them are listed below.

1. **Forecasting**

Neural network can be used very effectively in forecasting exchange rates, predicting stock values, inflation and cash forecasting, forecasting weather conditions etc. Researchers have proved that the forecasting accuracy of NN systems tend to excel over that of the linear regression model.

Scanned by CamScanner

2. **Image compression**

- Digital images require a large amount of memory for storage. As a result, the transmission of image from one computer to another can be very expensive in terms of time and bandwidth required.
- With the explosion of Internet, more sites are using images. Image compression is a technique that removes some of the redundant information present in the image without affecting its perceptibility, thus, reducing the storage size required to store the image.
- NN can be effectively used to compress the image. Several NN techniques such as Kohonen's self-organizing maps, Back propagation algorithm, Cellular neural network etc. can be used for image compression.

3. **Industrial process control**

- Neural networks have been applied successfully in industrial process control of dynamic systems.
- Neural networks (especially multi- layer perceptrons) have been proved to be the best choice for modelling non-linear systems and implementing general – purpose non-linear controllers, due to their universal approximation capabilities. For example control and management of agriculture machinery.

4. **Optical Character Recognition**

- Well known application using image recognition is the Optical Character Recognition (OCR) tools that are available with the standard scanning software for the home computer.
- **Scansoft** has had great success in combining NN with a rule based system for correctly recognizing both characters and words, to get a high level of accuracy.

5. **Customer Relationship Management**

- Another popular application for NN is Customer Relationship Management (CRM).
- Customer Relationship Management requires key information to be derived from raw data collected for each individual customer. This can be achieved by building models using historical data information.
- Many companies are now using neural technology to help in their day to day business processes. They are doing this to achieve better performance, greater insight, faster development and increased productivity.
- By using Neural Networks for data mining in the databases, patterns can be identified for the different types of customers, thus giving valuable customer information to the company.
- Also, NN could be useful for important tasks related to CRM, such as forecasting call centre loading, demand and sales levels, monitoring and analysing the market, validating, completing and enhancing databases, clustering and profiling client base etc.
- One example is the airline reservation system AMT, which could predict sales of tickets in relation to destination, time of year and ticket price.

6. **Medical science**

- Medicine is the field that has always taken benefits from the latest and advanced technologies.
- Artificial Neural Networks (ANN) is currently the next promising area of interest in medical science.
- It is believed that neural networks (also Deep networks) will have extensive application to biomedical problems in the next few years.
- ANN has already been successfully applied in medical applications such as diagnostic systems, bio chemical analysis, disease detection, image analysis and drug development.

Scanned by CamScanner

5. 2 Basic Models of Artificial Neural Networks

The basic model of ANN is specified by three basic entities.

1. The Model's synaptic connections.
2. The training or learning rules for updating weights.
3. The activation functions.

5.2.1 Connections

The arrangement of neurons in layers and connection between them defines the neural network architecture. There are four basic types of neuron connection architectures. They are:

1. Single Layer feed forward network
2. Multi-layer feed forward network
3. Single layer Recurrent network
4. Multi-layer Recurrent network

[We will discuss above points in Section 5.5]

5.2.2 Learning

MU - Dec. 11, May 12, Dec. 13, Dec. 14

Q. What is learning ? Compare different learning rules. **(Dec. 11, 10 Marks)**

Q. What is learning in Neural Networks ? Compare different learning rules. **(May 12, 10 Marks)**

Q. What is learning in neural networks ? Differentiate between supervised and unsupervised learning. **(Dec. 13, Dec. 14, 10 Marks)**

- Learning is a relatively permanent change in behaviour brought about by experience.
- Learning in a network is a process that forces the network to yield a particular response to a specific input.
- There are two types of learning used to train the neural network: **batch learning** and **incremental learning.**
- Batch learning takes place when the network weights are adjusted in a single training step.
- In this mode, the complete set of input training data is needed to determine weights. Feedback information produced by the network is not involved in developing the network. This learning technique is called **recording.**
- Incremental learning is most commonly used and can be broadly classified into three basic types.

5.2.2(A) Supervised Learning

MU - May 13, Dec. 13

Q. Explain with neat diagram supervised learning. **(Dec. 13, 3 Marks)**

Q. What is supervised learning ? **(May 13, 3 Marks)**

- In this type of learning, each input pattern has a corresponding output pattern associated with it, which is the target or the desired pattern.
- Here, a comparison is made between the actual output of the network and the desired output to find out the 'error'.
- The computed error can be used to adjust network parameters (like connection weights, threshold etc.). As the network parameters are modified, the performance of the network improves.
- The training could continue until the network is able to produce the expected or desired response.

Scanned by CamScanner

- As shown in Fig. 5.2.1, the distance function $\rho[d, o]$ takes two values as an input. Actual network output o and desired output d for an input X and then computes error measure.
- Since we have assumed adjustable weights, the weights can be adjusted to improve network performance that is to reduce the error.

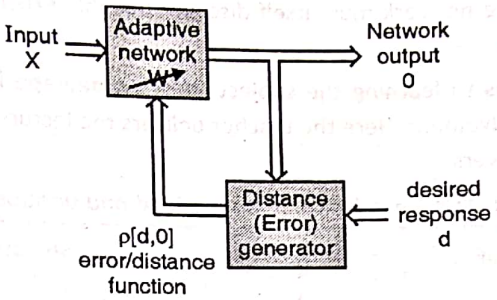


Fig. 5.2.1 : Block diagram of supervised learning

- This is analogous to classroom learning with the teacher's questions answered by students and corrected, if needed by the teacher.

5.2.2(B) Unsupervised Learning

MU - Dec. 12, May 13, Dec. 15

Q.	Explain with neat diagram unsupervised learning.	**(Dec. 12, 3 Marks)**
Q.	What is unsupervised learning ? Compare different learning rules.	**(May 13, 6 Marks)**
Q.	Distinguish between supervised and un-supervised learning.	**(Dec. 15, 5 Marks)**

- In this learning method, the desired output is not presented to the network. It is as if there is no teacher to present the desired or expected output. Hence the system learns on its own by discovering and adapting to structural features present in the input pattern.

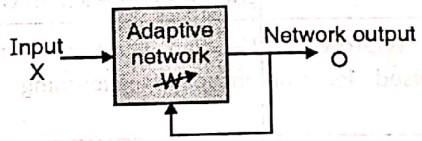
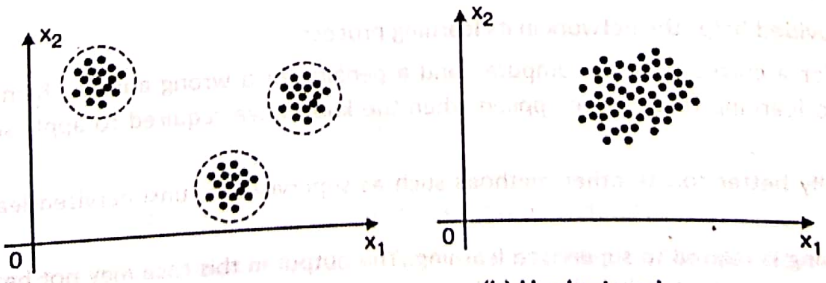


Fig. 5.2.2 : Block diagram of unsupervised learning

- Since no information is available to correct the network, learning must somehow be accomplished based on the observations of responses to inputs.
- For example, unsupervised learning can be used for finding the boundary between classes of input patterns distributed as shown in Fig. 5.2.3.
- In favourable case, as shown in Fig. 5.2.3 (a), cluster boundaries can be found based on the large and representative sample of inputs.



(a) Clustered **(b) Unclustered**

Fig. 5.2.3 : Two dimensional patterns

Scanned by CamScanner

- Since no external instructions regarding potential clusters is available, suitable weights self-adaption mechanism needs to be embedded in the trained network. One possible network adaptation rule that can be applied is "A pattern added to a cluster has to be closer to the cluster centre than to the other clusters' centres".
- Thus in unsupervised learning, the network must itself discover possibly existing patterns, regularities or separating properties.
- This type of learning is analogous to learning the subject from a videotape lecture covering the material but not including any other teacher's involvement. Here the teacher delivers the lecture but is not available for clarification of unclear questions or to check answers.

Table 5.2.1 : Difference between supervised and unsupervised learning

Sr. No.	Supervised Learning	Unsupervised Learning
1.	Supervised Learning is that type of learning where the desired output is known	Unsupervised Learning is a type of learning where the desired output is unknown
2.	A supervised learning algorithm analyses the training data and produces an inferred function, which can be used for mapping new examples.	Unsupervised learning uses procedures that attempt to find natural partitions of patterns given to the neural network.
3.	Supervised learning is achieved by means of classification or regression	Unsupervised learning is achieved by means of clustering
4.	Supervised learning is known as learning with a teacher's presence as the desired output is known beforehand	Unsupervised learning is known as learning without a teacher's presence as desired output is unknown
5.	Supervised learning is limited to learning small and simple models	With unsupervised learning it is possible to learn larger and more complex models than with supervised learning
6.	Perceptron learning, Delta learning, Widrow-Hoff learning, are examples of supervised learning algorithms	Hebbian learning, Winner-take-all learning are examples of unsupervised learning algorithms
7.	Example: Given several images of faces and not-faces (i.e. labelled data), a supervised algorithm will eventually learn and be able to predict whether or not an unseen image is a face.	Example: Given several images and there is no labelling on them, an unsupervised algorithm clusters the data into different groups, e.g. it can distinguish that faces are very different from landscapes, which are very different from horses.

5.2.2(C) Reinforced Learning

- In this method, a teacher though available, does not present the expected answer but only indicates if the computed output is correct or incorrect.
- The information provided helps the network in its learning process.
- A reward is given for a correct answer computed and a penalty for a wrong answer. Reinforced learning is a very general approach to learning that can be applied when the knowledge required to apply supervised learning is not available.
- However, it is usually better to use other methods such as supervised or unsupervised learning, because they are more direct.
- Reinforcement training is related to supervised learning. The output in this case may not be indicated as the desired output, but the condition whether it is **'success'**(1) or **'failure'** (0) may be indicated. Based on this, error may be calculated. The error signal produced here is binary.

Scanned by CamScanner

- Basically, this learning attempts to learn the input-output mapping through trial and error. Here the system knows whether the output is correct or not, but does not know the correct output.
- Table 5.2.1 shows the difference between supervised and unsupervised learning.

5.2.3 Activation Functions

- The output response of a neuron is calculated using an **activation function** (also called transfer function).
- The sum of weighted inputs (net input to the neuron) is applied with an activation to obtain the neuron's response.
- Neurons placed in the same layer use same activation.
- There are basically two types of activation functions: linear and non-linear.
- The non-linear activation functions are used in a multi-layer net.

The most commonly used activation functions are :

1. **Unipolar Binary**
   - It's unipolar in nature meaning it generates only two values 0 or 1.
   - It is used only at output layer.
   - It is computed as,

$$f(net) = \begin{cases} 1 & net > 0 \\ 0 & net < 0 \end{cases}$$

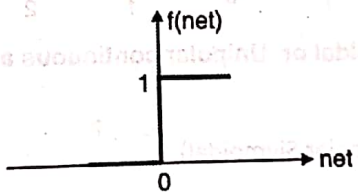


Fig. 5.2.3 : Unipolar binary activation function

2. **Bipolar Binary**
   - It's bipolar in nature meaning it generates two values +1 or -1.
   - It is used only at output layer.
   - Its computed as,

$$f(net) = sgn\,(net) = \begin{cases} +1 & net > 0 \\ -1 & net < 0 \end{cases}$$

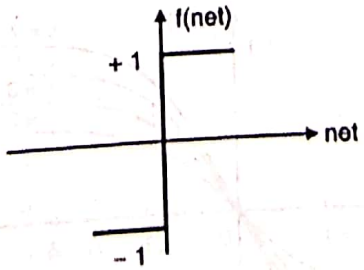


Fig. 5.2.4: Bipolar binary activation function

- The two functions bipolar binary and unipolar binary are called **hard limiting** activation functions.

Scanned by CamScanner

3. Sigmoidal Function / Unipolar Sigmoidal

- Also known as unipolar Continuous or (Binary Sigmoidal)
- It's range is 0 to 1.
- It's non-linear in nature.
- Usually used in output layer of a binary classification, where result is either 0 or 1, as value for sigmoid function lies between 0 and 1 only.
- It is computed as,

$$f(net) = \frac{1}{1 + e^{(-\lambda net)}} \qquad \text{Where, } net = \sum_{i=1}^{n} w_i x_i$$

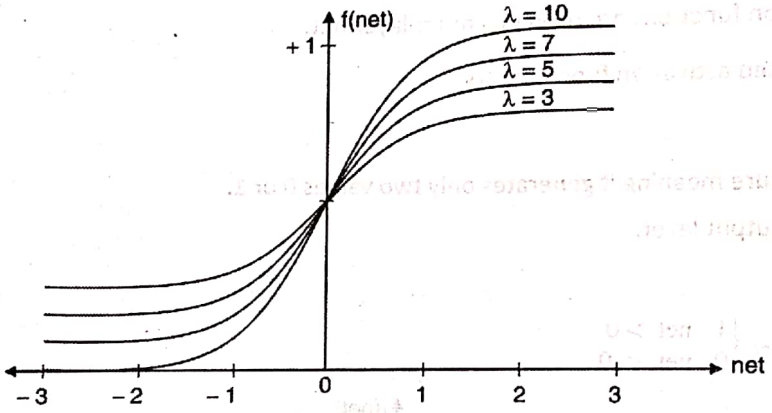


Fig. 5.2.5 : Sigmoidal or Unipolar continuous activation function

4. Tanh Function / Bipolar Sigmoidal

- Also called Bipolar Continuous (or Bipolar Sigmoidal).
- The range is between + 1 and – 1.
- It is computed as,

$$f(net) = \frac{2}{1 + e^{(-\lambda net)}} - 1 \qquad \text{Where, } net = \sum_{i=1}^{n} w_i x_i$$

- This function is related to the hyperbolic tangent function. λ is called the **steepness parameter.**
- Usually used in hidden layers of a neural network as it's values lies between **-1 to 1.**
- It's output is zero centered because its range in between -1 to 1. Hence in practice it is always preferred over Sigmoid function .

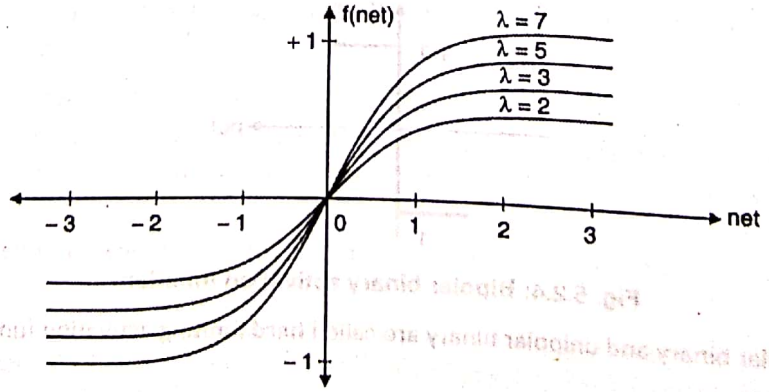


Fig. 5.2.6 : Tanh or bipolar continuous activation function

Scanned by CamScanner

- We can easily observe that at $\lambda \to \infty$ in bipolar continuous (sigmoidal) function it becomes the sgn (net) function.
- The two activation functions bipolar continuous and unipolar continuous have sigmoidal characteristics and hence called **soft limiting** activation functions.

Note : Both sigmoidal and tanh functions suffer from Vanishing gradient problem.

5. ReLu- Rectified Linear units

- It is mathematically represented as

$$R(x) = \max(0,x)$$

i.e if $x < 0$, $R(x) = 0$ and if $x >= 0$, $R(x) = x$.

- It is simple and efficient.
- It was recently proved that it had 6 times improvement in convergence from Tanh function.
- it avoids and rectifies **vanishing gradient** problem . Almost all deep learning Models use **ReLu** nowadays. But its limitation is that it should only be used within Hidden layers of a Neural Network Model.

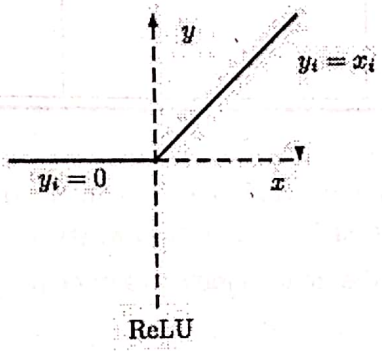
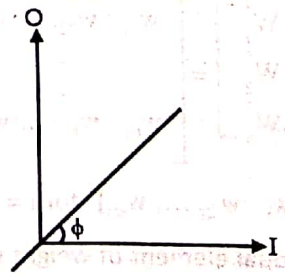


Fig. 5.2.7 : ReLu

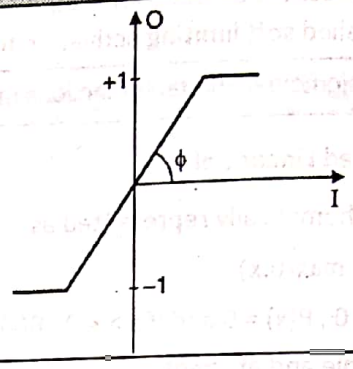
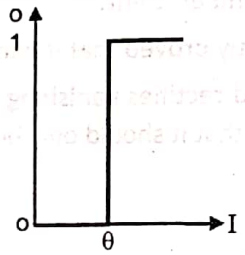
6. Softmax Function

- The softmax function is also a type of sigmoid function.
- The sigmoid function can handle only two classes. What if there are multiple classes.
- Softmax is commonly used for multi - classification problems.
- It is non-linear in nature.
- The softmax function would squeeze the outputs for each class between 0 and 1 and would also divide it by the sum of the outputs. This gives the probability of the input being in a particular class.
- The softmax function is ideally used in the output layer of the classifier where we actually try to attain the probabilities to define the class of each input.

Other activation functions:

Type	Equation	Functional form
Linear	O = gI g = (tan ϕ)	

Scanned by CamScanner

Type	Equation	Functional form
Piecewise linear	$O = \begin{cases} 1 & \text{if } mI > 1 \\ gI & \text{if } \lvert mI < 1 \rvert \\ -1 & \text{if } mI > -1 \end{cases}$ $g = \tan\phi$	
Step function (unipolar binary)	$O = \begin{cases} 1 & \text{if } I >= \theta \\ 0 & \text{if } I < \theta \end{cases}$	

Note :

Sigmoidal functions (unipolar continuous and bipolar continuous) are used in multilayer nets like back-propagation networks, Radial Basis Function Networks (RBFN) etc.

The reason is, in multilayer network, the actual inputs are effectively masked off from the output units by the intermediate layer.

The hard-limiting threshold functions (unipolar binary and bipolar binary) remove the information that is needed if the network is to successfully learn.

Hence the network is unable to determine which of the input weights should be increased and which ones should not.

On the other hand, soft-limiting activation functions (sigmoidal functions) give us some information on the inputs, so that the network will be able to determine when to strengthen or weaken the relevant weights.

5.3 Important Terminologies of ANN

5.3.1 Weights

In the architecture of neural networks, each neuron is connected to other neurons by means of directed communication links. Each of these links are associated with weights. These weights carry the information about the input signals. The weights can be presented in terms of a matrix called the **weight matrix**. The weight matrix is also called **connection matrix**, and is defined as,

$$W = \begin{bmatrix} W_1^t \\ W_2^t \\ W_n^t \end{bmatrix} = \begin{bmatrix} w_{11} & w_{12} & \dots & w_{1m} \\ w_{21} & w_{22} & \dots & w_{2m} \\ & \vdots & & \\ w_{n1} & w_{n2} & \dots & w_{nm} \end{bmatrix}$$

Where $W_i = [w_{i1}, w_{i2}, \dots, w_{im}]^T$ for i = 1, 2, 3, ..., n is the weight vector for processing element.

w_{ij} is the individual element of weight vector that represents the weight on the communication link from i^{th} neuron to j^{th} neuron.

Scanned by CamScanner

5.3.2 Bias

- A bias is a weight on a connection from an additional input unit whose activation is always 1.
- Increasing the bias increases the net input to the unit.

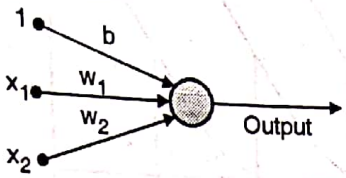


Fig. 5.3.1 : A single neuron with a bias

- If the bias is present, the net value is calculated as,

$$net = b + \sum_{i=1}^{n} x_i w_i; \; i = 1, 2, \ldots, n$$

- For the above example, net is computed as,

$$net = b + w_1 x_1 + w_2 x_2$$

- Similar to initialization of weights, bias should also be initialized to some specific value.

Why do we need to use bias?

- Using bias in a neural network helps to improve performance of the net.
- For example, the bias value allows us to shift the activation function to the left or right which may be critical for successful learning.
- Usually, when a bias is not used, the separating line (or plane) passes through the origin. This may not be appropriate for solving a particular problem. Adding an adjustable bias helps us to shift the separating line (or plane) either to the right or to the left of the origin.
- Consider the following network shown in Fig. 5.3.2.

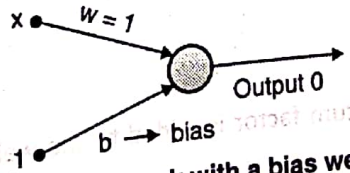


Fig. 5.3.2: Network with a bias weight

- If we use sigmoidal activation function, then output o is calculated as,

$$o = \text{sigmoid}\,(1 \times x + b)$$
$$= \text{sigmoid}\,(x + b)$$

where b is the bias.

- Using different values of bias, we can shift the entire curve to the left or to the right as shown in the Fig. 5.3.4.

Scanned by CamScanner

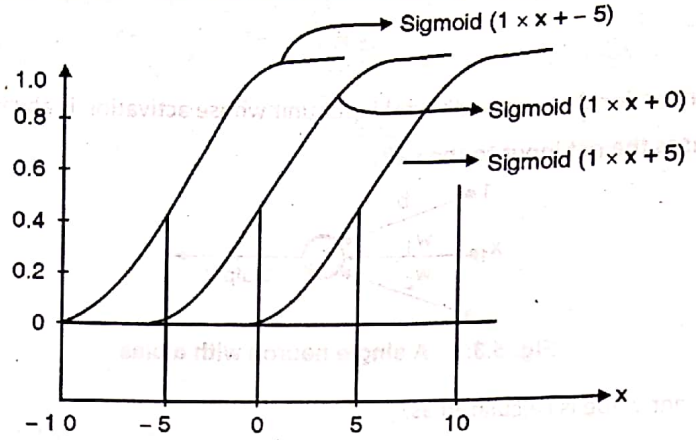


Fig. 5.3.4: Sigmoid functions with different bias values

5.3.3 Threshold

- Threshold is a set value based upon which the final output of the network is calculated.
- Usually a net input to the neuron is calculated and then compared with the threshold value.
- If the net value is greater than the threshold, then the neuron fires, otherwise, it does not fire.

5.3.4 Learning Constant

- The learning constant is used to control the amount of weight adjustment at each step of training. It controls the rate of learning.
- The effectiveness and convergence of learning algorithm depends significantly on the value of the learning constant. However, the optimum value of learning rate depends on the problem being solved and therefore there is no single learning constant that can be used for different training cases.
- For example, to solve a problem with broad minima, a large value of learning constant will result in a more rapid convergence. However, for problems with steep and narrow minima, a small value of learning constant must be chosen to avoid overshooting the solution. But choosing a small value of learning constant also increases the total number of steps in training.

5.3.5 Momentum Factor

- Convergence is made faster if a momentum factor is added to the weight update process. This is generally done in back propagation algorithm.
- The method involves supplementing the current weight adjustment with a fraction of the most recent weight adjustment.

$$\Delta W(t) = -\eta \nabla E(t) + \alpha \Delta W(t-1)$$

where, argument t and t – 1 indicate the current and the most recent training step respectively.

- η is a learning constant and α is the user defined selected positive momentum constant.
- In the above equation, the term $\alpha \Delta W(t-1)$ indicates a scaled most recent adjustment of weights and is called the **momentum term.**
- Typically α is chosen between 0.1 and 0.8.

Scanned by CamScanner

5.3.6 Vigilance Parameter

The vigilance parameter denoted by **ρ** is used in ART networks. It is used to control the degree of similarity required for pattern to be assigned to the same cluster unit. The range of vigilance parameter is usually 0.7 to 1.

5.4 McCulloch-Pitts Neuron Model

MU – Dec. 11, May 12, Dec. 12

Q. Explain Mc-Culloch Pitts Neuron Model with help of an example. **(Dec. 11, May 12, 5/6 Marks)**

Q. Explain with an example McCulloch-Pitts neuron model. **(Dec. 12, 6 Marks)**

- The first formal definition of a synthetic neuron model based on highly simplified considerations of the biological model was formulated by McCulloch and Pitts in 1943.
- The model is shown in the Fig. 5.4.1.
- The inputs x_i for i = 1, 2, 3, ..., n are either 0 or 1, depending on the absence or presence of the input impulse at instant k.

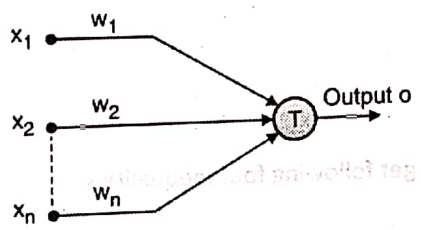


Fig. 5.4.1 : McCulloch Pitts Neuron model

- The neuron's output signal is denoted by *o*.
- The inputs are connected by direct weighted paths to neuron.
- The weights associated with the communication links may be excitatory(weight is positive)or inhibitory (weight is negative).
- There is a fixed threshold for each neuron, and if the net input to the neuron is greater than the threshold, then the neuron fires. If the net input to the neuron is less than the threshold, then the neuron does not fire i.e., It goes into an inhibition state.
- Accordingly, the firing rule for the neuron is defined as follows,

$$o^{k+1} = \begin{cases} 1 & \text{if } \sum_{i=1}^{n} w_i x_i^k \geq T \\ 0 & \text{if } \sum_{i=1}^{n} w_i x_i^k < T \end{cases}$$

- In the above equation, superscript *k* denotes discrete time instances, *k = 1, 2, 3, ...*
- By analogy with the biological neuron, an effective synapse which transmits a stronger signal will have correspondingly larger weights, while a weak synapse will have smaller weights.
- In McCulloch Pitts' model,

  w_i = + 1 for excitatory synapses,

  w_i = – 1 for inhibitory synapses.

Scanned by CamScanner

And *T* is the neuron's threshold value, which needs to be exceeded by the weighted sum of input signals for the neuron to fire.

- Although, this neuron model is very simplistic, it has substantial computing potential. It can perform basic logical operations such as AND, OR, NOT etc. provided its weights and threshold are appropriately selected.
- The McCulloch Pitts neuron does not have any particular training algorithm. An analysis has to be performed to determine the values of weights and the threshold.

Problems based on McCulloch Pitts Model :

Ex. 5.4.1 : Design two input AND logic using McCulloch Pitts Neuron model.

Soln.: Consider the truth table for the AND logic function shown in Table P. 5.4.1.

Here we need to find the appropriate values of w_1, w_2 and threshold T such that, they satisfy the truth table of AND logic.

Here, the firing rule is,

$$w_1 x_1 + w_2 x_2 \geq T \quad \text{...(1)}$$

And inhibition rule is,

$$w_1 x_1 + w_2 x_2 < T \quad \text{...(2)}$$

From the truth table of AND logic, we get following four inequalities.

$$0 < T$$

$$w_2 < T$$

$$w_1 < T$$

$$w_1 + w_2 \geq T$$

Table P.5.4.1 : Truth table of AND

x_1	x_2	Output y
0	0	0
0	1	0
1	0	0
1	1	1

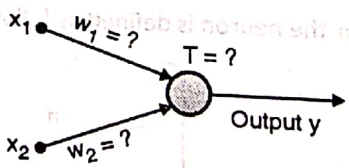


Fig. P.5.4.1: MCP Neuron Model for Logical AND

In order to get above four inequalities, use Equation (1) when neuron must fire and use Equation (2) when neuron must be inhibitory.

For $x_1 = 0, x_2 = 0 \Rightarrow y = 0$ (i.e. inhibitory)

So substitute $x_1 = 0, x_2 = 0$ in Equation (2), we get,

$$0 < T$$

For $x_1 = 0, x_2 = 1 \Rightarrow y = 0$ (i.e. inhibitory)

Scanned by CamScanner

So substitute $x_1 = 0$, $x_2 = 1$ in Equation (2),we get,

$$w_2 < T$$

For $x_1 = 1$, $x_2 = 0 \Rightarrow y = 0$ (i.e. inhibitory)

So, substitute $x_1 = 1$ and $x_2 = 0$ in Equation (2), we get,

$$w_1 < T$$

For $x_1 = 1$, $x_2 = 1 \Rightarrow y = 1$ (i.e. Firing)

So, substitute $x_1 = 1$ and $x_2 = 1$ in Equation (1), we get,

$$w_1 + w_2 \geq T$$

Here w_1 and w_2 can only take either + 1 or –1values.

So, for $w_1 = 1$, $w_2 = 1$ and T = 2, all of the four inequalities are satisfied. Hence, solution is,

$$w_1 = 1$$

$$w_2 = 1$$

$$T = 2$$

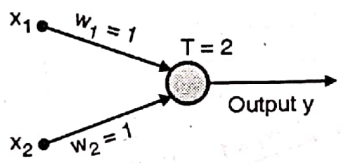


Fig. P.5.4.1(a) : Final weights and threshold for AND logic

Ex. 5.4.2 : Design two inputs OR logic using McCulloch Pitts Neuron model.

Soln.:

Consider the truth table for OR logic function shown in the Table P.5.4.2.

Table P.5.4.2 : Truth Table of OR

x_1	x_2	Output y
0	0	0
0	1	1
1	0	1
1	1	1

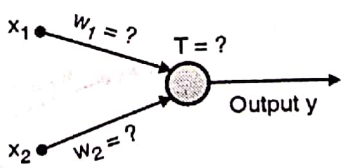


Fig. P.5.4.2 :MCP Neuron Model for Logical OR

We need to find the appropriate values of w_1, w_2 and T such that they satisfy the truth table of OR logic.

Here, the firing rule is, ...(1)

$$w_1 x_1 + w_2 x_2 \geq T$$

and the inhibitory rule is, ...(2)

$$w_1 x_1 + w_2 x_2 < T$$

From the truth table of OR logic, we get the following four inequalities.

$$0 < T$$

$$\ldots > T$$

Scanned by CamScanner

$$w_1 \geq T$$

$$w_1 + w_2 \geq T$$

To derive the above four inequalities, use Equation (1) when the neuron must fire and use Equation (2) when the neuron must be inhibitory.

For $x_1 = 0, x_2 = 0 \Rightarrow y = 0$ (i.e. inhibitory), so substitute $x_1 = 0, x_2 = 0$ in Equation (2), we get,

$$0 < T$$

For $x_1 = 0, x_2 = 1 \Rightarrow y = 1$ (i.e. firing), so substitute $x_1 = 0, x_2 = 1$ in Equation (1),we get,

$$w_2 \geq T$$

For $x_1 = 1, x_2 = 0 \Rightarrow y = 1$ (i.e. firing) so substitute $x_1 = 1, x_2 = 0$ in Equation (1) we get,

$$w_1 \geq T$$

For $x_1 = 1, x_2 = 1 \Rightarrow y = 1$ (i.e. firing).

So, substitute $x_1 = 1, x_2 = 1$ in Equation (1) we get,

$$w_1 + w_2 \geq T$$

For $w_1 = 1, w_2 = 1$ and $T = 1$, all of the above inequalities are satisfied. Hence, the solution is,

$$w_1 = 1$$

$$w_2 = 1$$

$$T = 1$$

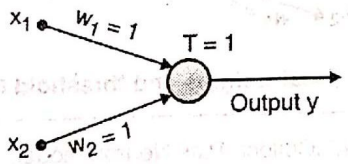


Fig. P.5.4.2(a) : Final weights and threshold for OR logic

Ex. 5.4.3 : Implement the logic function shown in Table P.5.4.3 using McCulloch-Pitts neuron.

Soln. :

Table P.5.4.3 : Truth table

x_1	x_2	Output y
0	0	0
0	1	0
1	0	1
1	1	0

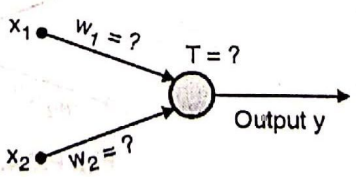


Fig. P.5.4.3: MCP Neuron Model for given Logical function

We need to find the appropriate values of w_1, w_2 and T such that they satisfy the truth table.

The firing rule is,

$$w_1 x_1 + w_2 x_2 \geq T \quad \text{...(1)}$$

And the inhibitory rule is,

$$w_1 x_1 + w_2 x_2 < T \quad \text{...(2)}$$

Scanned by CamScanner

From the truth table , we get the following four inequalities,

$$0 < T$$

$$w_2 < T$$

$$w_1 \geq T$$

$$w_1 + w_2 < T$$

So for $w_1 = 1$, $w_2 = -1$ and $T = 1$, all of the above inequalities are satisfied. So, the solution is,

$$w_1 = 1$$

$$w_2 = -1$$

$$T = 1$$

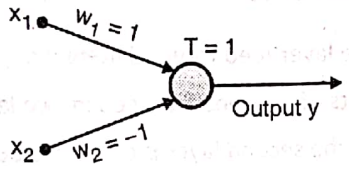


Fig. P.5.4.3(a) : Final weights and threshold for given logic

Ex. 5.4.4 : Design NOT logic using McCulloch Pitts neuron model.

Soln.:

Consider the truth table of NOT logic function.

Table P. 5.4.4 : Truth table of NOT

x	y
0	1
1	0

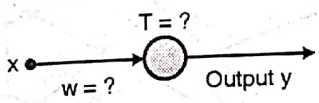


Fig. P.5.4.4 : MCP Neuron Model for Logical NOT

Here, we need to find out the appropriate values of w and T, such that, they satisfy the truth table of NOT logic.

The firing rule can be,

$$w_x \geq T \quad \text{...(1)}$$

And inhibitory rule can be,

$$w_x < T \quad \text{...(2)}$$

From the truth table of NOT logic, we get following two inequalities.

$$0 \geq T$$

$$w < T$$

So, for $T = 0$ and $w = -1$, the above mentioned two inequalities are satisfied.

So, solution is,

$$w = -1$$

$$T = 0$$

Scanned by CamScanner

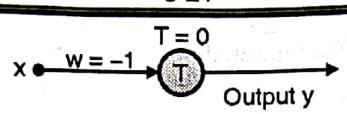


Fig. P.5.4.4(a) : Final weights and threshold for NOT

5.5 Neural Network Architecture

- As we know, an artificial neuron is defined as a data processing system consisting of a large number of simple highly interconnected processing elements inspired by the biological neuron.
- Generally, an ANN structure is represented using a directed graph, where neuron outputs are connected, through weights, to all other neurons including themselves; Both lag-free and delay connections are allowed.
- Based on these connection types, there are four fundamental architectures of neural networks.

5.5.1 Single Layer Feed Forward Networks

- Fig 5.5.1 shows the architecture of a single layer feed forward network.
- A single layer feed forward network consists of neurons arranged in two layers.
- The first layer is called the ***input layer*** and the second layer is called the ***output layer***.
- The input signals are fed to the neurons of the input layer and neurons of the output layer produce output signals.
- Every input neuron is connected to every output neuron via synaptic links or weights.

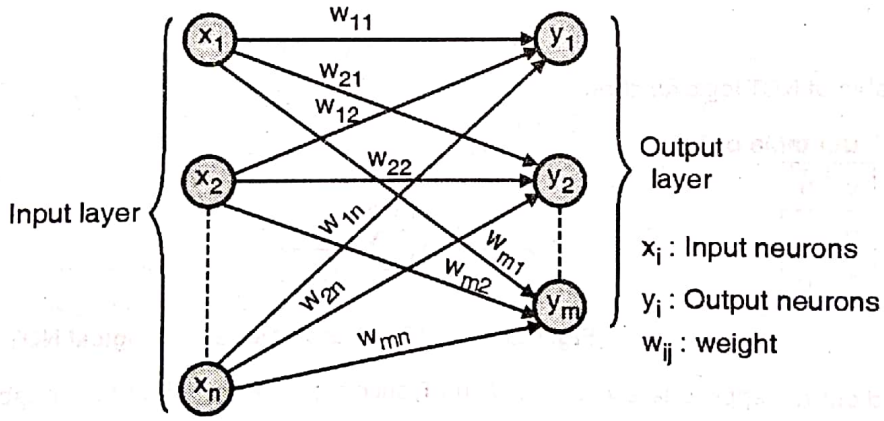


Fig. 5.5.1 : Single Layer feed forward network

- The output vector is given as,

$$O = [o_1\ o_2\ o_3 \ldots\ldots o_m]^t$$

And input vector is,

$$X = [x_1 x_2 x_3 \ldots\ldots x_n]^t$$

- Weight W_{ij} connects the ith output neuron with the jth input neuron.
- Then the activation value of the ith neuron is given as,

$$net_i = \sum_{j=0}^{n} w_{ij}\, x_j, \quad \text{For } i = 1, 2, 3, \ldots, m$$

- The transformation performed by each of the *m* neurons in the network, is non-linear mapping and expressed as,

$$O_i = f\left(W_i^t X\right), \text{ for } i = 1,2,3,\ldots,m$$

Scanned by CamScanner

Where weight vector W_i contains weights leading toward the ith output node and is defined as,

$$W_i = [\, w_{i1} \;\; w_{i2} \;\; w_{i3} \ldots w_{in} \,]^t$$

- In spite of having two layers, the network is still named as 'single layer' because out of the two layers, only the output layer performs computations. The input signal is simply transmitted to the output layer by the input layer.

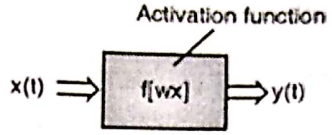


Fig. 5.5.2 : Feed forward network block diagram

- Thus, feed forward networks are characterised by the lack of feedback. That is, the output of any given neuron is not fed back to itself directly or indirectly or through other neurons. Thus, present output does not influence future output.

5.5.2 Multilayer Feed Forward Networks

- In a multilayer feed forward network, there are multiple layers. Thus, besides having input layer and output layer, this network has one or more intermediary layers called hidden layers.

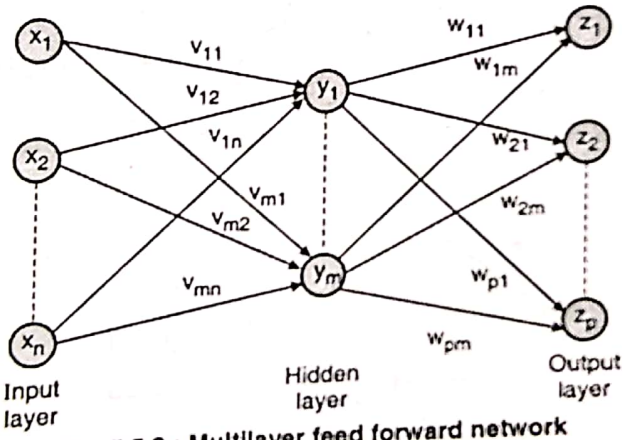


Fig. 5.5.3 : Multilayer feed forward network

Input layer	Hidden layer	Output layer
x_i	Input neurons	i = 1, 2, 3, ..., n
y_j	Hidden neurons	j = 1, 2, 3, ...,.m
z_k	Output neurons	k = 1, 2, 3, ... p
v_{ij}	Input- hidden layer weight	
w_{jk}	Hidden-output layer weight	

- Fig. 5.5.3 shows a multilayer feed forward network.
- The computational units of the hidden layer are known as hidden neurons or hidden units. Before directing the input to the output layer the hidden layer performs useful intermediary computations.
- Neurons in input layer are connected to the neurons in hidden layer and the weights on these connections are referred to as "Input-hidden layer weights".

Scanned by CamScanner

- Similarly, the hidden layer neurons are connected to the output layer neurons and the corresponding weights are referred to as "Hidden-output layer weights".
- A multi-layer feed forward network with m input neurons, n_1 neurons in the first hidden layer, n_2 neurons in the second hidden layer and k output neurons is written as $m - n_1 - n_2 - k$ network.

5.5.3 Single Layer Feedback Network

- The feedback networks differ from the feed forward network in the sense that there exists at least one feedback loop.
- When the output of the output neurons is fed back as an input to the same or preceding layer nodes, then this type of network is called feedback network.
- Fig. 5.5.4 shows one such type of feedback network.

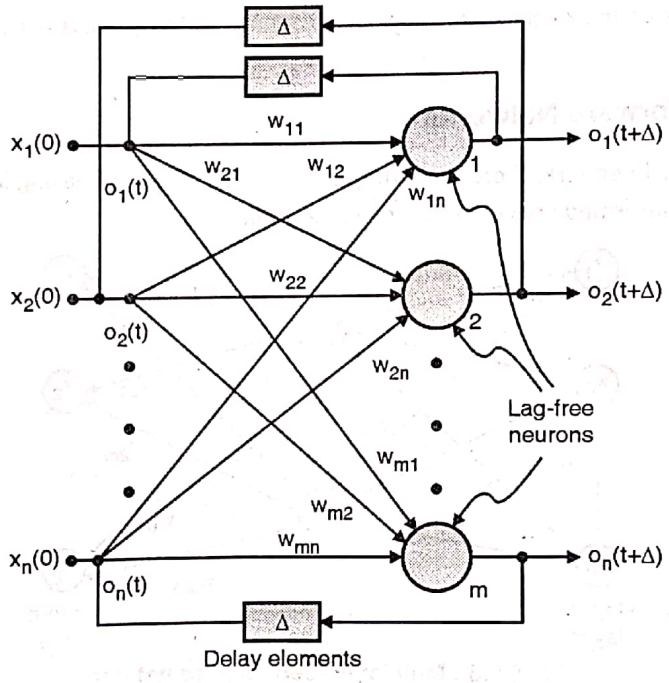


Fig. 5.5.4 : Single layer discrete time feedback network

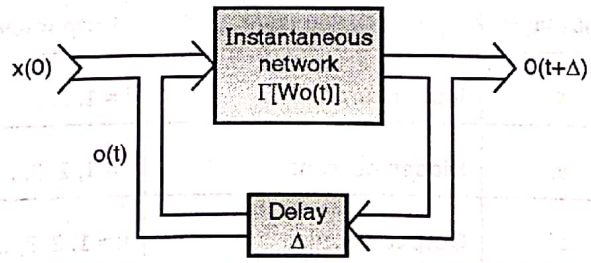


Fig. 5.5.5 : Block diagram of feedback network

1. The present output, say $o(t)$, controls the output at the following instant, $o(t + \Delta)$.
2. Δ indicates the time elapsed between t and $t + \Delta$. Here the time delay Δ is in an analogy to the refractory period of a basic biological neuron model.
3. Thus, the mapping of $o(t)$ into $o(t+\Delta)$ can be written as,

$$o(t + \Delta) = \Gamma\,[Wo(t)]$$

Scanned by CamScanner

5.5.4 Multilayer Feedback Networks

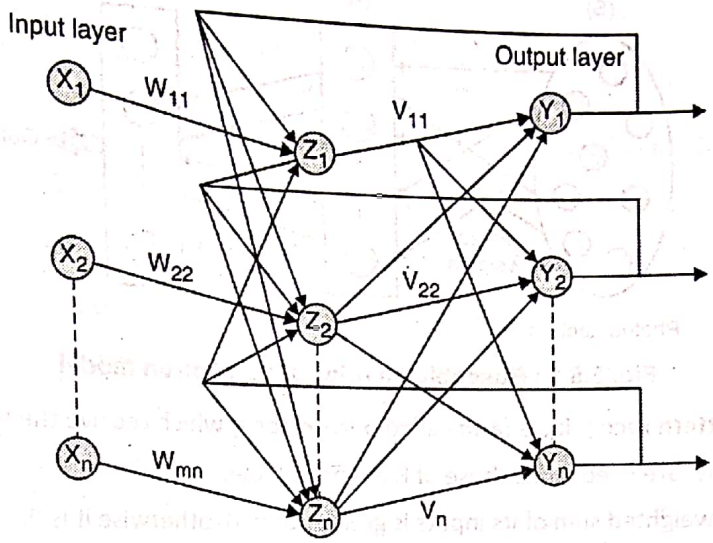


Fig. 5.5.6 : Multiplayer recurrent network (Redraw Figure)

- Fig 5.5.6 shows architecture of multilayer feedback network.
- It can be noted that a processing element output can be directed back to the nodes in a preceding layer, forming a multilayer recurrent network.
- Also, in these networks, a processing element output can be directed back to the processing element itself and to other processing elements in the same layer.

5.6 Supervised Learning

As discussed earlier, supervised learning works on labeled data meaning, for every input, the corresponding output (or class) is known.

Following section discusses three main supervised algorithms.

1. Perceptron learning rule
2. Delta learning rule
3. Back propagation algorithm

5.6.1 Perceptron Learning Rule

MU – Dec. 11, May 12

Q. Explain with example perceptron learning rule. **(Dec. 11, 10 Marks)**

Q. Explain perceptron learning with the help of an example. **(May 12, 8 Marks)**

5.6.1(A) Rosenblatt's Perceptron

As shown in Fig.5.6.1, the network consists of 3 units, the Sensory unit S, Association unit A and Response unit R.

- The S unit receives input images and generates 0 or 1 electrical signal as output .It contains 400 photo-detectors.
- These input signals are then compared with threshold. If the input signals exceed a threshold, then the photo-detector outputs 1 otherwise 0.
- The photo-detectors are randomly connected to the Association unit A. The association unit A comprises of **feature demons or predicates.**

Scanned by CamScanner

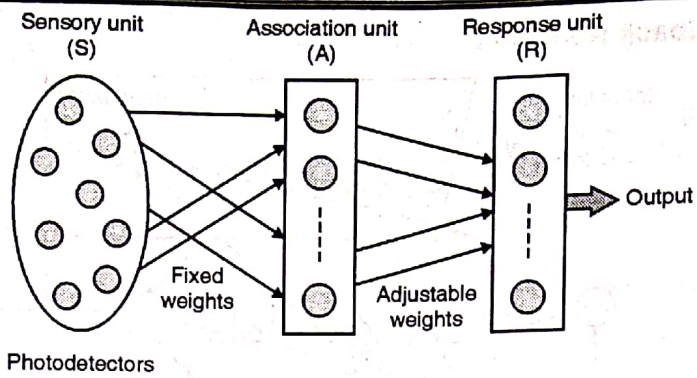


Fig. 5.6.1 : Rosenblatt's original perceptron model

- The 3rd unit R contains **pattern recognizers (also called perceptrons)** which receive the results of the predicate.
- The weights of S and A units are fixed while those of R are adjustable.
- The output of R is 1 if the weighted sum of its inputs is greater than 0, otherwise it is 0.

5.6.1(B) The Perceptron Model

- The perceptron model is based on the perceptron learning rule in which the learning signal is the difference between the desired output and the neuron's actual output.
- Here learning is obtained in **supervised** environment and the learning signal is equal to:

$$r = d_i - o_i$$

Where $o_i = \text{sgn}(W_i^t X)$

and d_i = desired (or target) output

- Here the weights are adjusted using following weight update formula.

$$\Delta W_i = c\left[d_i - \text{sgn}\left(W_i^t X\right)\right] X$$

$$\Delta W_{ij} = c\left[d_i - \text{sgn}\left(W_i^t X\right)\right] X_j \text{ ; for } j = 1, 2, \ldots n$$

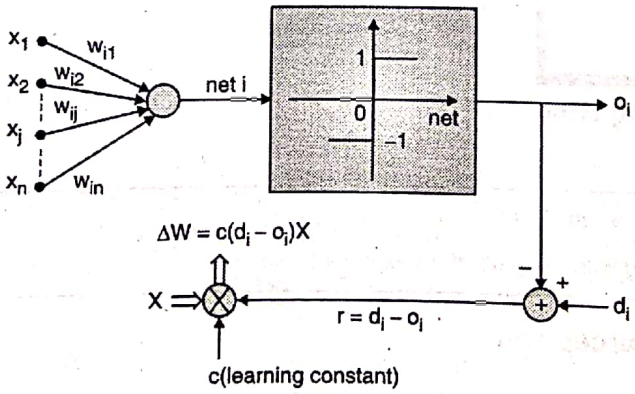


Fig. 5.6.2 : The perceptron model

- This rule is applicable for ***binary neuron*** response only.
- From the above formula of ΔW_i it is clear that ,weights are adjusted if and only if o_i is incorrect (i.e. if $d_i \neq o_i$)
- Since the desired output/response is either 1 or – 1, the weight adjustment reduces to

$$\Delta W_i = \pm 2cX$$

Scanned by CamScanner

- Where $\Delta W_i = +2cX$ when $d_i = 1$ and $Sgn\left(W_i^t X\right) = o_i = -1$

  and $\Delta W_i = -2cX$ when $d_i = -1$ and $Sgn\left(W_i^t X\right) = o_i = 1$

- The weight adjustment is inherently zero, when $d_i = o_i$ (i.e. $d_i = sgn\left(W_i^t X\right)$). Thus, when the desired output is the same as the actual output, there will not be any change in weights.

5.6.1(C) Algorithm for Training a Single Perceptron Network

- A single perceptron can solve a classification problem with 'n' input features and two output classes (0/1).
- Given below is the algorithm for training such a network and is called SDPTA (Single Discrete Perceptron Training Algorithm)

SDPTA (Single Discrete Perceptron Training Algorithm)

- Given are P training pairs

  $\{X_1, d_1, X_2, d_2, X_3, d_3, \ldots, X_p, d_p\}$ where X_i is $(n \times 1)$, d_i is (1×1), $i = 1, 2, \ldots P$
- Here augmented input vectors are used.

$$Y_i = \begin{bmatrix} X_i \\ 1 \end{bmatrix} \qquad \text{for } i = 1, 2, \ldots, P$$

i.e. there is an additional input for the bias and it is 1.

In the following, *k* denotes the training steps and *p* denotes the step counter within a cycle.

Step 1 : $c > 0$ is chosen

Step 2 : Weights are initialized as ***W*** at small random values.

W is $(n + 1) \times 1$

Counters are initialized

$K \leftarrow 1, p \leftarrow 1, \quad E \leftarrow 0$

Step 3 : The training cycle begins here. Input is presented and output is computed

$Y \leftarrow Y_p, \quad d \leftarrow d_p$

$O \leftarrow sgn\,(\mathbf{W}^t Y)$

Step 4 : Weights are updated

$$\mathbf{W} \leftarrow \mathbf{W} + \frac{1}{2} c\,(d - o)\,Y$$

Step 5 : Cycle error is computed

$$E \leftarrow \frac{1}{2}(d - o)^2 + E$$

Step 6 : If $p < P$ then

$p \leftarrow p + 1 \qquad k \leftarrow k + 1$

and go to step 3 ; otherwise go to step 7

Step 7 : The training cycle is completed.

For $E = 0$, terminate the training session. Display weights and *k*.

If $E > 0$, then $E \leftarrow 0, p \leftarrow 1$

and enter the new training cycle by going to step 3

Scanned by CamScanner

5.6.1(D) Model for Multilayer Perceptron

- In multilayer perceptron networks, the perceptrons are arranged in layers.
- This model has three (or more) layers:
  a. An input layer
  b. An output layer and
  c. One (or more) layers between input layer and output layer. These layers are called hidden layers.
- For perceptrons in the input layer, we use linear transfer function and for the perceptrons in the hidden and output layers, we use **sigmoid (continuous)** function.
- Here, since the single layer perceptron's model is modified by adding a hidden layer and changing the transfer (activation) function from linear function to a nonlinear function, we need to alter the learning rule as well. So now the multilayer perceptron network should be able to recognize more complex things.
- The input-output mapping of multilayer perceptron is shown in Fig. 5.6.3 and is represented by,

$$O = f_2 [W [f_1[XV]]$$ Where, f_1 and f_2 represent non-linear mapping.

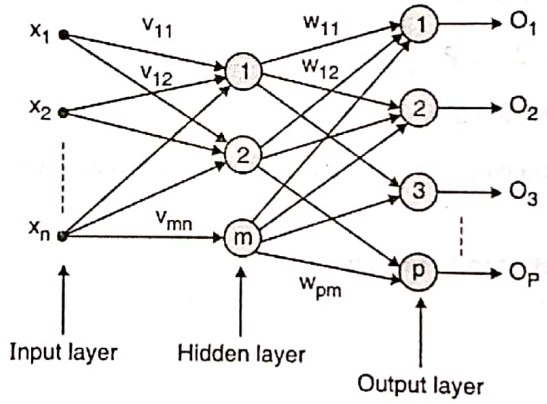


Fig. 5.6.3 : Multilayer perceptron

5.6.1(E) The EX-OR Problem and Need for Multi-layer Perceptron

- The simplest and most well-known pattern recognition problem in neural network literature is the Exclusive-OR problem.
- The truth table of Ex-OR function is given below.

Table 5.6.1: EX-OR Truth Table

x_1	x_2	Output (y)
1	1	– 1
1	– 1	1
– 1	1	1
– 1	– 1	– 1

- The task of the network is to classify a binary input vector to output class = – 1, if the vector has similar inputs (Both the inputs +1 Or both the inputs -1) or assign it to output class = 1 otherwise.
- Let us try to solve this problem using a single-layer perceptron model. Consider the following single-layer perceptron model shown in Fig. 5.6.4.

Scanned by CamScanner

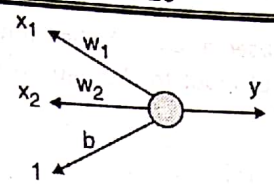


Fig. 5.6.4 : Single layer preceptron model

- For the above model, the decision boundary is

$$b + w_1 x_1 + w_2 x_2 = 0$$

$$\text{i.e. net} = 0$$

If net > 0, y (the output)= 1, else y = – 1

- So from the truth-table and above conditions, we have the following four inequalities.

$$b + w_1 + w_2 \leq 0$$

$$b + w_1 - w_2 > 0$$

$$b - w_1 + w_2 > 0$$

$$b - w_1 - w_2 \leq 0$$

which must be satisfied.

- However, the set of inequalities is clearly self-contradictory when considered as a whole.
- This is because the EX-OR problem is not linearly separable. This can easily be observed from the plot given in Fig. 5.6.5.

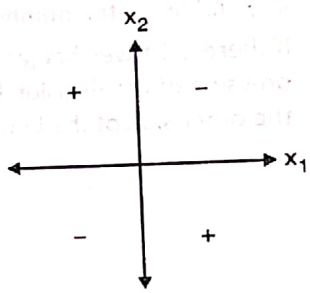


Fig. 5.6.5 : EX-OR function plot

- In other words, we cannot use a single layer perceptron to construct a straight line to partition the 2D input space into two regions (class 1 and class 2).

Solving EX-OR problem using Multilayer perceptron

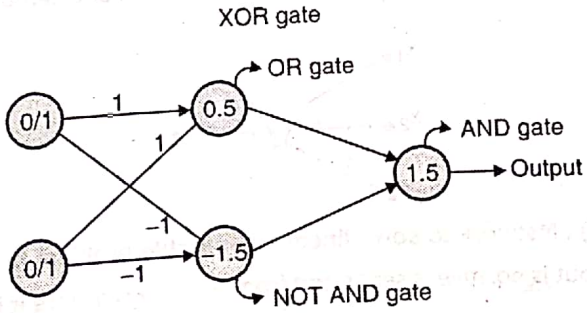


Fig. 5.6.6 : EX-OR using multi - layer perceptron

5.6.1(F) Linearly and Non-Linearly Separable Patterns

MU – Dec. 10, May 12, Dec. 13, Dec. 15

Q. Explain linearly separable and non-linearly separable patterns with examples. **(Dec. 10, 5 Marks)**

Q. Explain with examples linearly separable and non-linearly separable pattern classification. **(May 12, Dec. 13, Dec.15, 10 Marks)**

a. Linearly Separable Patterns

- Consider a particular classification problem, where if the input pattern is a member of its class, then the desired response (output) is "yes", otherwise it is "no".
- A "yes" is represented by an output signal of 1, a "no" by an output signal of – 1 (considering bipolar signals).

Scanned by CamScanner

- To achieve this, a step function can be used as an activation function. That is, the output is + 1, if the net input to the neuron is positive, and -1 if the net input to the neuron is negative. Since the net input to the neuron is computed as

$$net = b + \sum_{i=1}^{n} x_i w_i$$

- It is clear that the boundary between the region where net > 0 and the region where net < 0 is determined by the relation.

$$b + \sum_{i=1}^{n} x_i w_i = 0$$

- This boundary is also called **decision boundary.**
- Depending on the number of input units in the network, this equation represents a line, a plane or a hyperplane.
- If there exist weights (and a bias) such that all of the training input vectors for which correct response is + 1 lie on one side of the decision boundary and all of the training input vectors for which the correct response is – 1 lie on the other side of the boundary then we say that the problem is **"linearly separable".**

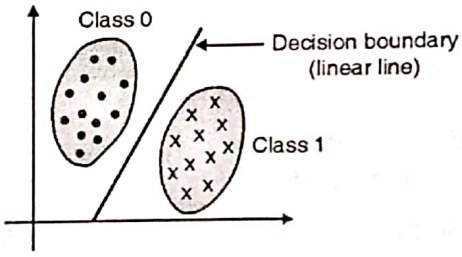


Fig. 5.6.7 (a) : Linearly separable pattern

- The problem shown in Fig. 5.6.7 (a) is a linearly separable problem and can be solved using the network shown in Fig. 5.6.7 (b).

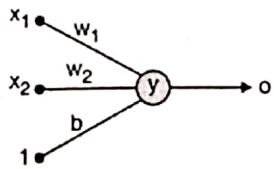


Fig. 5.6.7 (b) : Network to solve linearly separable problem in Fig. 5.6.7 (a)

- The region where the output is positive is separated from the region where it is negative by the line,

$$x_2 = \frac{-w_1}{w_2} x_1 - \frac{b}{w_2}$$

- Here,

∵ net is computed as

$$net = b + \sum_{i=1}^{n} w_i x_i$$

- Here,

$$net = b + w_1 x_1 + w_2 x_2$$

- So, the decision boundary is

$$b + w_1 x_1 + w_2 x_2 = 0$$

Scanned by CamScanner

$$\therefore \quad x_2 = \frac{-w_1}{w_2} x_1 - \frac{b}{w_2}$$

- In the above example, there are many different lines that will serve to separate the input vectors into two classes. There could also be many choices of w_1, w_2 and b that give exactly the same line.
- To understand the concept of linear separability further, consider simple logic gates AND and OR.
- The AND gate can be represented by the truth-table shown in Table 5.6.2.

Table 5.6.2 : AND Truth Table

x_1	x_2	y(output)
1	1	1
1	−1	−1
−1	1	−1
−1	−1	−1

- Then, the desired response for the AND function can be represented as shown in Fig. 5.6.8(a).

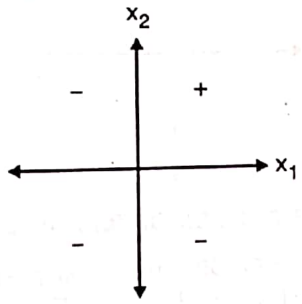


Fig. 5.6.8(a) : AND function plot

- The possible decision boundary for AND function could be as shown in Fig. 5.6.8 (b).

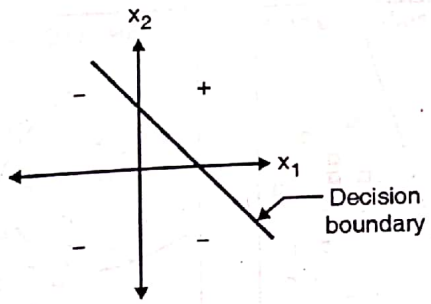


Fig. 5.6.8 (b) : AND function decision boundary

- Hence it is linearly separable.
- Similarly, the OR function is also linearly separable which is clear from the Fig. 5.6.9.

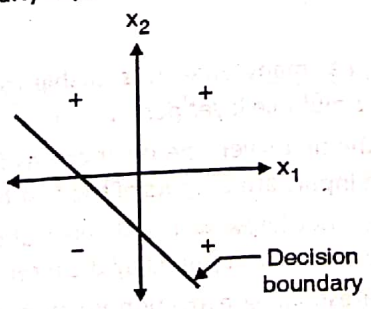


Fig. 5.6.9: OR Function solution

Scanned by CamScanner

- The examples that we have seen are in 2D space.
- When we say that a set of points in two dimensional space are linearly separable, we mean that they can be separated by a straight line.
- More generally, a set of points in n-dimensional space are said to be linearly separable if they can be separated by a hyper plane.
- A linearly separable problem can be solved by a perceptron network.
- However, problems which are non-linearly separable cannot be solved by a single layered perceptron network.

b. Non-Linearly Separable Patterns

- To understand non-linearly separable patterns, consider the Exclusive-OR problem.
- Here, the task of the network is to classify a binary input vector to class 1 (output = – 1), if the vector has even number of 1s, or assign it to class 2 (output = 1) otherwise.

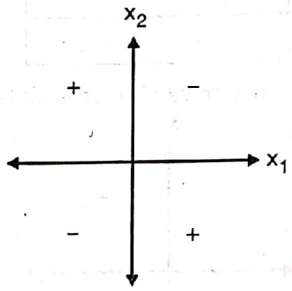


Fig. 5.6.10 : EX-OR function plot

- Fig. 5.6.10 shows EX-OR function plot. The EX-OR is nonlinearly separable since there is no line that can partition the 2D input space into two regions (class1 and class2). This can easily be observed from the plot given in Fig. 5.6.10.
- Fig. 5.6.11 shows examples of linearly separable and non-linearly separable patterns.

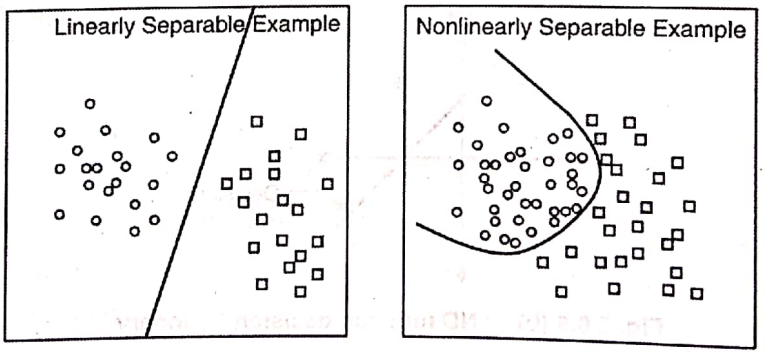


Fig. 5.6.11 : Linearly separable and non-linearly separable patterns

5.6.2 Delta Learning Rule

- The problem with perceptron learning is that many times it is unable to solve non-linearly separable problems. The initial solution to such problems was to use multiple layer perceptron.
- In such a network, for the perceptron in the first layer, the input comes from the actual inputs of the problem, while for the perceptron in the second layer, the inputs are outputs of the first layer.
- So, the perceptrons of the second layer do not know which of the real inputs of the first layer were on or off. The actual inputs are effectively masked off from the output units by the intermediate layer.
- The problem here is, the hard limiting thresholding activation function removes the information that is needed to train the network.

Scanned by CamScanner

The solution to this problem is to use non-linear (continuous) activation function such as sigmoidal or radial basis function.

Hence Widrow-Hoff introduced the Delta learning rule that uses non-linear differentiable activation function for training the multilayer perceptron networks.

6.2(A) Delta Learning Model

Delta learning is a supervised form of learning which uses ***continuous activation functions.***

The learning signal for this rule is called delta and is defined as

$$r = [d_i - f(W_i^t X)]\, f'\left(W_i^t X\right)$$

The term $f'\left(W_i^t X\right)$ is the derivative of the activation function f(net) computed for,

$net = W_i^t X$

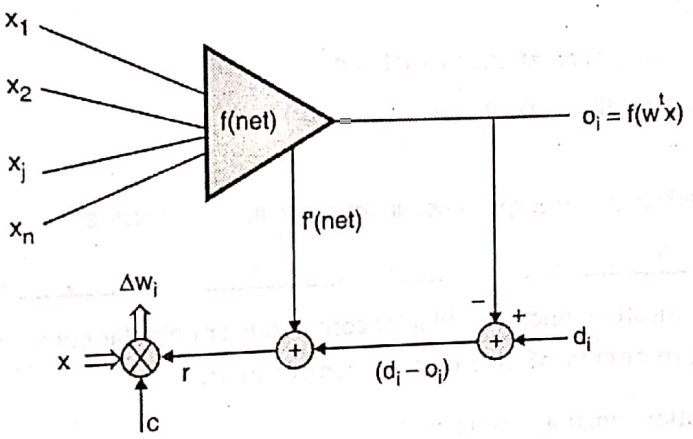


Fig. 5.6.12: Delta learning model

The learning rule can be readily derived from the condition of least squared error between o_i and d_i.

Calculating the gradient vector with respect to W_i of the squared error defined as

$$E = \frac{1}{2}(d_i - o_i)^2 \qquad \text{...(5.6.1)}$$

which is equivalent to

$$E = \frac{1}{2}\left[d_i - f\left(W_i^t X\right)\right]^2 \qquad \text{...(5.6.2)}$$

We obtain the error gradient vector value which is obtained by taking partial derivative of E with respect to each of the weights.

$$\nabla E = -(d_i - o_i)\, f'\left(W_i^t X\right) X \qquad \text{...(5.6.3)}$$

The components of the gradient vector are

$$\frac{\partial E}{\partial W_{ij}} = -(d_i - o_i)\, f'\left(W_i^t X\right) x_j \;;\; j = 1, 2, \ldots, n \qquad \text{...(5.6.4)}$$

Since the minimization of the error requires the weight changes to be in the negative gradient direction, we take

$$\therefore\ \Delta W_i = -\eta \nabla E \qquad \text{...(5.6.5)}$$

Where η is a positive constant

$$\therefore \Delta W_i = \eta\,(d_i - o_i)\, f'\left(W_i^t X\right) X \qquad \text{...(5.6.6)}$$

Scanned by CamScanner

For a single weight, the adjustment becomes.

$$\Delta w_{ij} = \eta\,(d_i - o_i)\,f'(net_i)\,x_j\;;\; j = 1, 2, 3, \ldots n \qquad \ldots(5.6.7)$$

Thus,

$$\Delta W_i = c\,(d_i - o_i)\,f'(net_i)\,X$$

where $c = \eta$ (learning constant)

- The initial weights can be any random values.
- This rule is sometimes also called **continuous perceptron learning** rule.

5.6.2(B) Proofs

MU – Dec. 12, Dec. 13, Dec. 15

Q. Prove the following identities :

(i) For unipolar continuous activation function f' (net) = o (1-o)

(ii) For bipolar continuous activation function f'(net) = o $(1-o^2)/2$

Where o is output. **(Dec. 12, Dec. 13 10 Marks)**

Q. Prove that the first order derivative of a unipolar continuous activation function is f' (net) = O (1-O) **(Dec. 15, 5 Marks)**

- Depending on whether the activation function is bipolar continuous or unipolar continuous, and further assuming that $\lambda = 1$, we can express $f'(net)$ in terms of the output f(net) or *o*.
- For bipolar continuous activation function, we have,

$$f'(net) = \frac{1}{2}(1 - o^2) \qquad \ldots(5.6.8)$$

- Similarly, for unipolar continuous activation function.

$$f'(net) = o(1 - o) \qquad \ldots(5.6.9)$$

- Now, we will prove results (5.6.8) and (5.6.9)

Proofs

(I) $f'(net) = \frac{1}{2}(1 - o^2)$ **...[Bipolar continuous]**

Proof :

$$o = f(net) = \frac{2}{1 + \exp(-\lambda net)} - 1$$

$$\text{Assume } \lambda = 1$$

$$\therefore\; o = f(net) = \frac{2}{1 + \exp(-net)} - 1$$

$$\text{L.H.S} = f'(net)$$

$$= \frac{d}{d(net)}\left\{\frac{2}{1 + \exp(-net)} - 1\right\} = \frac{d}{d(net)}\left\{\frac{2}{1 + \exp(-net)}\right\} - (1)$$

$$= 2\frac{d}{d(net)}\left\{\frac{1}{1 + \exp(-net)}\right\}$$

Scanned by CamScanner

$$= 2\left[\frac{-\frac{d}{d(net)}(1+\exp(-net))}{(1+\exp(-net))^2}\right]$$

$$= \frac{-2(-\exp(-net))}{(1+\exp(-net))^2}$$

$$\text{L. H.S} = \frac{2\exp(-net)}{(1+\exp(-net))^2} \quad \text{...(5.6.10)}$$

$$\text{R. H. S} = \frac{1}{2}(1-o^2)$$

$$= \frac{1}{2}\left[1-\left\{\frac{2}{1+\exp(-net)}-1\right\}^2\right]$$

$$= \frac{1}{2}-\frac{1}{2}\left[\frac{2}{1+\exp(-net)}-1\right]^2$$

$$= \frac{1}{2}-\frac{1}{2}\left[\frac{4}{(1+\exp(-net))^2}-(2)\frac{2}{(1+\exp(-net))}+1\right]$$

$$= \frac{-2}{(1+\exp(-net))^2}+\frac{2}{1+\exp(-net)}$$

$$= 2\left[\frac{-1+(1+\exp(-net))}{(1+\exp(-net))^2}\right]$$

$$= \frac{2\exp(-net)}{(1+\exp(-net))^2} \quad \text{...(5.6.11)}$$

From Equations (5.6.10) and (5.6.11)

L.H.S = R.H.S.

Hence proved $f'(net) = \frac{1}{2}(1-o^2)$

(II) $f'(net) = o(1-o)$ ... (Unipolar continuous)

Proof

$$o = f(net)$$

$$= \frac{1}{1+\exp(-\lambda net)}$$

$$\text{Assume } \lambda = 1$$

$$o = f(net)$$

$$= \frac{1}{1+\exp(-net)}$$

$$\text{L. H. S} = f'(net)$$

$$= \frac{d}{d(net)}\left[\frac{1}{1+\exp(-net)}\right]$$

$$= \frac{-1\frac{d}{d(net)}(1+\exp(-net))}{(1+\exp(-net))^2}$$

$$= \frac{-(-\exp(-net))}{(1+\exp(-net))^2}$$

Scanned by CamScanner

$$\text{L. H.S.} = \frac{\exp(-\text{net})}{(1+\exp(-\text{net}))^2} \quad \ldots(5.6.12)$$

$$\text{R.H.S.} = o(1-o)$$

$$= \frac{1}{1+\exp(-\text{net})}\left[1-\frac{1}{1+\exp(-\text{net})}\right]$$

$$= \frac{1+\exp(-\text{net})-1}{(1+\exp(-\text{net}))^2}$$

$$= \frac{\exp(-\text{net})}{(1+\exp(-\text{net}))^2} \quad \ldots(5.6.13)$$

From Equations (5.6.12) and (5.6.13),

L.H.S. = R.H.S.

Hence proved, $f'(\text{net}) = o(1-o)$

5.6.2(C) Algorithm-Delta Learning

MU – Dec. 15

Q. Explain Single Continuous Perceptron Training Algorithm (SCPTA). **(Dec. 15, 5 Marks)**

SCPTA-Single Continuous Perception Training Algorithm

Given are P training pairs $\{X_1, d_1, X_2, d_2, \ldots X_p, d_p\}$

Where X_i is $(n \times 1)$ and d_i is (1×1)

$$i = 1, 2, 3 \ldots, P$$

Here, augmented input is used,

$$Y_i = \begin{bmatrix} X_i \\ 1 \end{bmatrix}_{(n+1)\times 1} \text{ for } i = 1, 2, \ldots, P$$

In the following, *k* denotes the no. of training steps and *p* denotes the no. of steps within a training cycle.

Step 1 : $\eta > 1$, $\lambda = 1$, $E_{max} > 0$ chosen

Step 2 : Weights W are initialized at some small random values.

W is $(n+1) \times 1$

Counters an initialized

$p \leftarrow 1$, $k \leftarrow 1$, $E \leftarrow 0$

Step 3 : The training cycle beings here.

Input is presented and output is computed.

$Y \leftarrow Y_p$, $d \leftarrow d_p$

$o \leftarrow f(\mathbf{W}^t Y)$

$$\text{With } f(\text{net}) = \frac{2}{1+e^{-\lambda\,\text{net}}} - 1$$

Scanned by CamScanner

Step 4 : Weights are updated

$W \leftarrow W + 1/2\, \eta\, (d - o)\, (1 - o^2) Y$

Step 5 : Cycle error is computed

$E \leftarrow E + 1/2\, (d - o)^2$

Step 6 : If $p < P$ then

$k \leftarrow k + 1$

$p \leftarrow p + 1$ and go to Step 3

Otherwise, go to Step 7.

Step 7 : The training cycle is completed. For $E < E_{max}$ terminate the training session. Output weights, *k* and *E*.

If $E \geq E_{max}$ then,

$E \leftarrow 0$

$p \leftarrow 1$ and enter the new training cycle by going to Step 3.

5.6.3 Back Propagation Algorithm

MU – Dec. 10, Dec. 11, May 12, Dec. 12, May 13, Dec. 13, Dec. 14

Q. Explain Error back Propagation Algorithm with the help of flowchart. **(Dec. 10, 12 Marks)**

Q. Explain Error back propagation training algorithm with the help of a flowchart. **(Dec. 11, May 12, Dec. 12, Dec. 13, Dec. 14, 10 Marks)**

Q. Explain Error Back Propagation Training Algorithm (EBPTA). **(May 13, 10 Marks)**

- The back propagation algorithm is one of the most important developments in neural networks. This learning algorithm is applied to multilayer feed-forward networks. Multi layer feed forward networks consist of processing elements with continuous differentiable activation functions.
- The back propagation algorithm is different from the other networks in respect to the process by which the weights are calculated during learning process.
- For the multilayer perception calculating the weights of the hidden layer in an efficient way that would result in a very small or zero output error is a challenging task.
- We can easily measure the error between the actual and desired output at the output layer. But at the hidden layer there is no direct information of the error available. Therefore, adjusting the weights of hidden layer perception is difficult.
- The back propagation algorithm solves this problem by providing the equation for updating the weights at hidden layer along with weight update formula at output layers such that the overall output error is minimized.
- The back-propagation algorithm contains two phases:
  1. Feedforward recall and
  2. Feedback phase that calculates the back-propagation of the error.

Scanned by CamScanner

5.6.3(A) Architecture of Back Propagation Network

Consider the layered feed forward neural network with two continuous perceptron layers shown in Fig. 5.6.13.

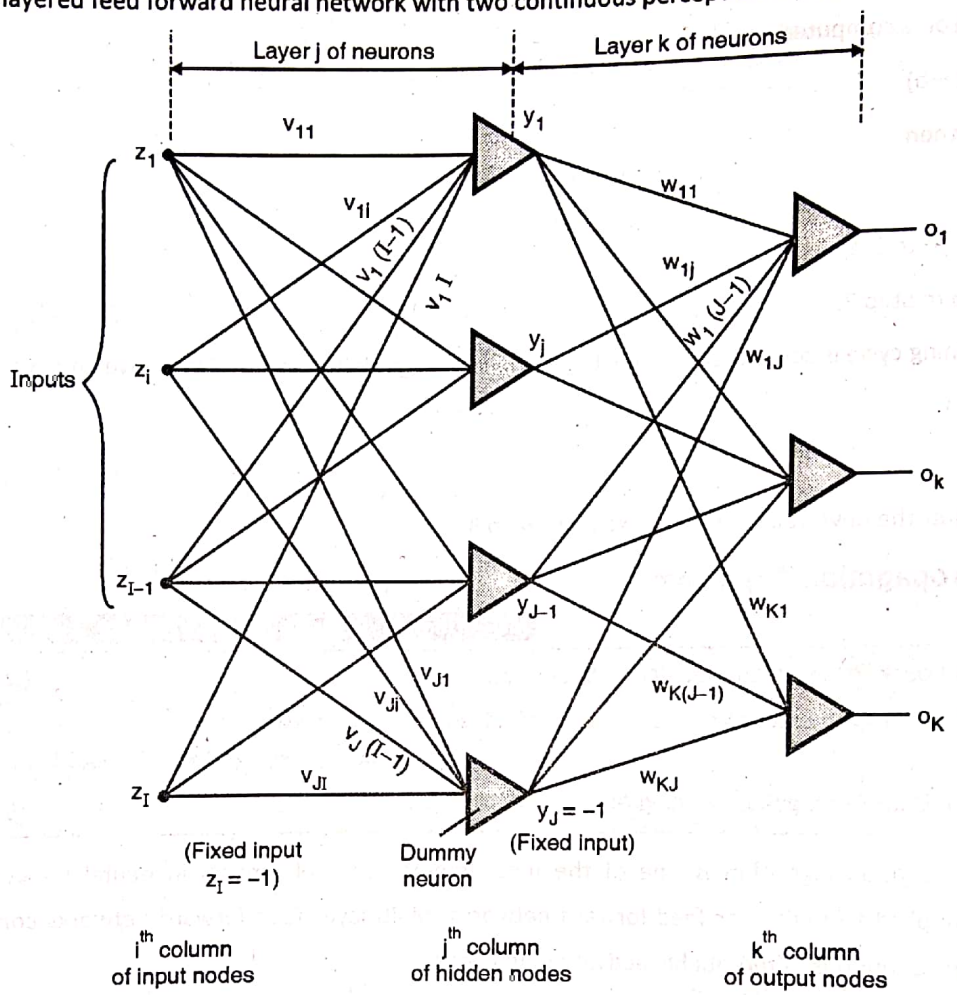


Fig. 5.6.13 : Back propagation network architecture

- In the above network, I input neurons are connected to each of the Jth hidden layer neuron.
- Weights connecting the input layer to the hidden layer are represented by matrix **V**. Similarly, outputs of J hidden neurons are connected to each of the K output neurons, with weight matrix **W**.
- EBPTA uses supervised learning mode, therefore the training patterns *Z* should be arranged in pairs with desired response *d* provide by the teacher.

Feed forward recall phase

- In this phase, the input pattern vectors are presented and mapped to the output vector *O* as follows :

$$O = \Gamma [W \cdot \Gamma [VZ]]$$

Where, Γ [VZ] is the internal mapping and relates to the hidden layer mapping $Z \rightarrow Y$

- The operator Γ is a non-linear diagonal operator.
- **V** and **W** are adjusted so that error value proportional to $||d - o||^2$ is minimized.

Scanned by CamScanner

Error back propagation training

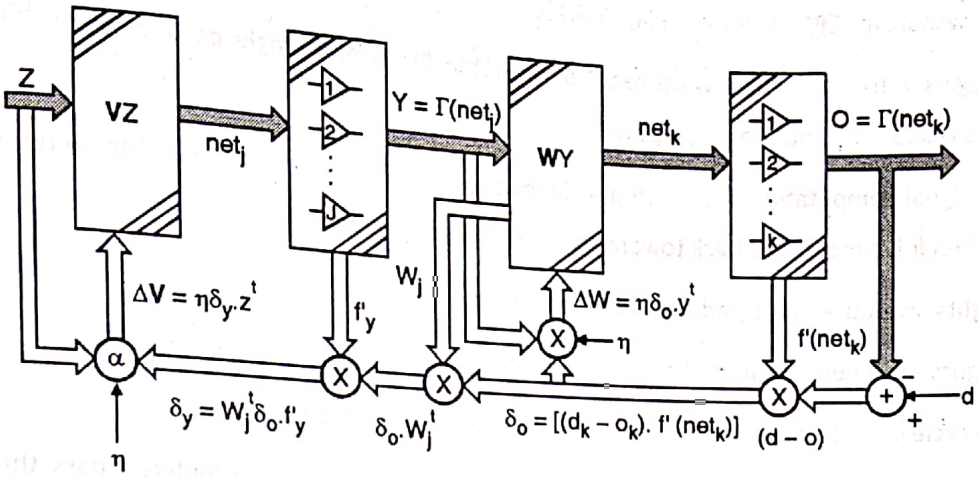


Fig. 5.6.14 : Block diagram : EBPT (Error back propagation training)

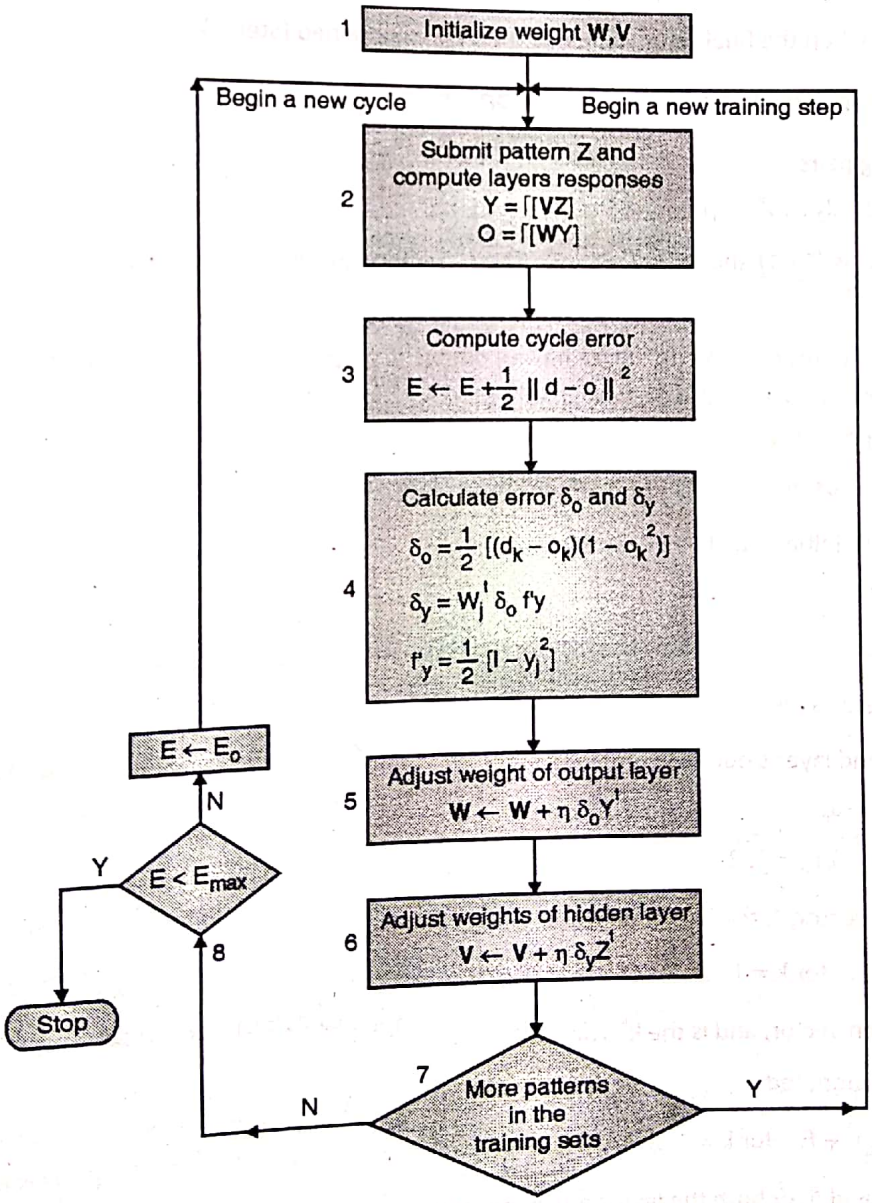


Fig. 5.6.15 : Flowchart Error Back propagation

Scanned by CamScanner

- Consider the flowchart for EBPTA shown in Fig. 5.6.15.
- The training begins with the feed forward recall phase (Step 2). After a single pattern vector Z is submitted at the input, the layers response Y and O are computed.
- Then the error signal computation phase (Step 4) follows. The error signal vector must be determined in the output layer first and then it is propagated back towards the network input nodes.
- Next, $K \times J$ weights are subsequently adjusted within the matrix W (Step 5).
- Finally $J \times I$ weights are adjusted within the matrix V (Step 6).
- The cumulative cycle error of input to output mapping is computed (in Step 3)
- The final error value for the entire training cycle is calculated after each completed pass through the training set $\{Z_1, Z_2, Z_3, \ldots Z_p\}$
- The learning stops when the final error value less than E_{max} is obtained (Step 8)

5.6.3(B) Algorithm (Error Back Propagation Training)

- Given are P training pairs

$$\{Z_1, d_1, Z_2, d_2 \ldots Z_p, d_p\}$$

Where Z_i is $(I \times 1)$, d_i is (1×1) and i = 1, 2, 3, ..., P. The ith component of each Z_i is of value (– 1) since input vector have been augmented.

- There are total $J - 1$ neurons in hidden layer having output Y. Note that the Jth component of Y is also of value (– 1), since hidden layer outputs have also been augmented.
- Thus Y is $(J \times 1)$ and O is $(K \times 1)$

Step 1 : $\eta > 0$ and E_{max} chosen.

Weights **W** and **V** are initialized at small random values.

W is $(K \times J)$, **V** is $(J \times I)$

$q \leftarrow 1, p \leftarrow 1, E \leftarrow 0$

Step 2 : Training step starts here

Input is presented and layer's outputs computed.

$$Z \leftarrow Z_p, \quad d \leftarrow d_p$$

$$y_j \leftarrow f(V_j^t Z), \quad \text{for } j = 1, 2, \ldots, J$$

where V_j, a column vector, is the jth row of vector V and

$$o_k \leftarrow f(W_k^t Y), \quad \text{for } k = 1, 2, \ldots, K$$

Where W_k is a column vector, and Is the kth row of W.

Step 3 : Error value is computed

$$E \leftarrow \frac{1}{2}(d_k - o_k)^2 + E, \quad \text{for } k = 1, 2, 3, \ldots K$$

Step 4 : Error signal δ_o and δ_y of both the layers are computed.

Scanned by CamScanner

Vector δ_o, is $(K \times 1)$, δ_y is $(J \times 1)$

$$\delta_{ok} = \frac{1}{2}(d_k - o_k)\left(1 - o_k^2\right), \text{ for } k = 1, 2, \ldots, K$$

$$\text{and } \delta_{yj} = \frac{1}{2}\left(1 - y_j^2\right)\sum_{k=1}^{K} \delta_{ok} w_{kj}, \text{ for } j = 1, 2, \ldots J$$

Step 5 : Output layer weights are adjusted.

$$w_{kj} \leftarrow w_{kj} + \eta \delta_{ok} y_j, \quad \text{for } k = 1, 2, 3, \ldots K; \; j = 1, 2, 3, \ldots, J$$

Step 6 : Hidden layer weights are updated

$$v_{ji} \leftarrow v_{ji} + \eta \delta_{yj} z_i, \quad \text{for } j = 1, 2, 3, \ldots, J; \; i = 1, 2, 3, \ldots, I$$

Step 7 : If $p < P$ then $p \leftarrow p + 1$, $q \leftarrow q + 1$ and go to step 2, otherwise go to step8.

Step 8 : The training cycle is completed.

For $E < E_{max}$ terminate the training and

Output weights **W**, **V**, q and E.

If $E > E_{max}$, then $E \leftarrow 0$, $p \leftarrow 1$, and initiate the new training cycle by going to step 2.

5.7 Un-Supervised Learning Algorithm: Self Organizing Maps

MU – Dec. 10, Dec. 11, May 12, May 13, Dec. 14, Dec. 15

Q. Write a short note on Kohonen's self organizing network. **(Dec. 10, Dec. 11, May 12, May 13, Dec. 14, 5/10 Marks)**

Q. What is self-organizing map? Draw and explain architecture of Kohonen's Self Organization Feature Map KSOFM. **(Dec. 15, 10 Marks)**

Self-Organizing Maps:

- A Self-Organizing Map (SOM) uses unsupervised learning to build a two-dimensional map of a problem space.
- It uses competitive learning as opposed to error-correction learning (which is used in backpropagation with gradient descent)
- A self-organizing map can generate a visual representation of data on a hexagonal or rectangular grid.
- SOM also represents clustering concepts by grouping similar data together.
- Applications include meteorology, oceanography, project prioritization, and oil and gas exploration.
- A self-organizing map is also known as a Self-Organizing Feature Map (SOFM) or a Kohonen map.

Kohonen's Self Organizing maps:

- Kohonens's self organizing networks, also called Kohonen's feature maps or topology preserving maps, are used for data clustering.
- Networks of this type impose a neighbourhood constraint on the output unit such that a certain topological property in the input data is reflected in the output unit's weight.

Scanned by CamScanner

- Fig. 5.7.1 presents relatively simple Kohonen's self organizing network with two inputs and 49 outputs.

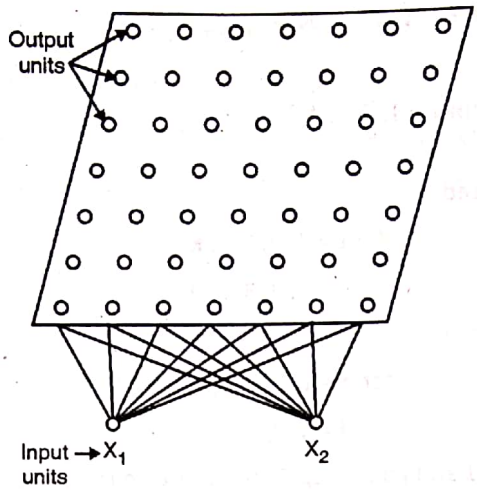


Fig. 5.7.1 : Kohonen'sself organizing network with two inputs and 49 outputs

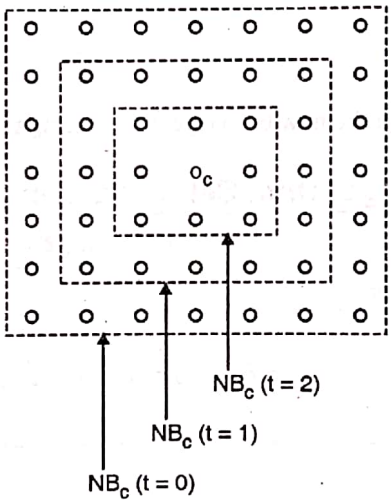


Fig. 5.7.2 : Neighbourhood of unit C

- The learning process here is similar to that of competitive learning networks. That is, a similarity (or dissimilarity) measure is selected and the winning unit is considered to be the one with the largest (or smallest) activation function.
- However, for Kohonen's feature maps, we update not only the winning unit's weight but also all of the weights in a neighbourhood around the winning unit.
- The neighbourhood size generally decreases slowly with each iteration, as shown in Fig. 5.7.2.
- Instead of defining a neighbourhood of a winning neuron, we can also use a neighbourhood function Ωc (i) around a winning unit *c*.

Where,

$$\Omega c\ (i) = \exp\left(\frac{-(P_i - P_c)^2}{2\sigma^2}\right)$$

- Here, σ reflects the scope of the neighbourhood and P_i and P_c are the positions of the output unit *i* and *c* respectively.

Algorithm (Kohonen's self organization map)

Step 1 : Initialize the weights at some random values → set topological neighborhood parameters. As clustering progresses, the radius of the neighborhood decreases. Initialize the learning rate α. It should be slowly decreasing function of time.

Scanned by CamScanner

Step 2 : Perform step 3-9 till stopping condition is true.

Step 3 : Perform steps 4-6 for each input vector X.

Step 4 : Compute the square of the Euclidean distance i.e. for each j = 1 to m (there are m no. of output neurons)

$$D(j) = \sum_{i=1}^{n} \sum_{j=1}^{m} (x_i - w_{ij})^2$$

Step 5 : Find the winning unit index J, so that D(J) is minimum.

Note : Instead of Euclidean distance, dot product method can also be used to finalize the winner, in which case the winner will be the one with the largest dot product.

Step 6 : For all units *j* within a specific neighborhood of *J* and for all *i*, calculate new weights.

$$w_{ij} = w_{ij}(\text{old}) + \alpha\,[x_i - w_{ij}(\text{old})]$$

Step 7 : Update the learning rate α using formula $\alpha(t+1) = 0.5\,\alpha(t)$

Step 8 : Reduce radius of topological neighborhood at specified time intervals.

Step 9 : Test for stopping condition.

5.8 Solved Problems

Ex. 5.8.1 : A neuron with 4 inputs has the weight vector $w = [1\ 2\ 3\ 4]^t$. The activation function is linear that is, the activation function is given by f(net) = 2 * net. If the input vector is $X = [5\ 6\ 7\ 8]^t$, then find the output of the neuron. **MU - Dec. 11, 5 Marks**

Soln. :

Fig.P. 5.8.1 shows the architecture of the given network.

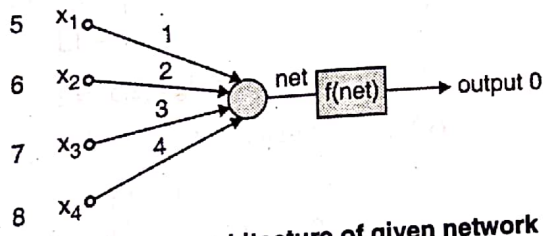


Fig. P. 5.8.1 : Architecture of given network

$$\text{net} = W^t \cdot X$$

$$= [1\ 2\ 3\ 4]\begin{bmatrix} 5 \\ 6 \\ 7 \\ 8 \end{bmatrix}$$

$$= 5 + 12 + 21 + 32$$

$$= 70$$

$$O = f(\text{net}) = 2 * \text{net} = 2 \times 70 = 140$$

Scanned by CamScanner

Ex. 5.8.2 : Use perception learning rule to train the network. The set of input training vectors are as follows.

$$X_1 = \begin{bmatrix} 1 \\ -2 \\ 0 \\ -1 \end{bmatrix} \quad X_2 = \begin{bmatrix} 0 \\ 1.5 \\ -0.5 \\ -1 \end{bmatrix} \quad X_3 = \begin{bmatrix} -1 \\ 1 \\ 0.5 \\ -1 \end{bmatrix}$$

and the initial weight vector W_1 is,

$$W_1 = \begin{bmatrix} 1 \\ -1 \\ 0 \\ 0.5 \end{bmatrix}$$

The learning, constant c = 0.1. The teacher's desired response for $X_1 X_2$ and X_3 are d1 = – 1, d2 = – 1 and d3 = 1 respectively. Calculate the weights after one complete cycle.

Soln. :

For perceptron learning rule,

$$\text{net}_i = W_i^t X$$

$$o_i = \text{sign}(\text{net}_i)$$

$$\Delta W_i = c(d - o_i) X$$

Step 1: Take first training pair $X = X_1$, $d = d_1$

$$X = \begin{bmatrix} 1 \\ -2 \\ 0 \\ -1 \end{bmatrix}, \quad d = -1$$

Compute :

$$\text{net}_1 = W_i^t X = [1 \ -1 \ 0 \ 0.5] \begin{bmatrix} 1 \\ -2 \\ 0 \\ -1 \end{bmatrix} = 2.5$$

$$o_1 = \text{sign}(\text{net}_1) = \text{sign}(2.5) = 1$$

$$\Delta W_1 = c(d - o_1) X$$

$$= 0.1(-1-1) \begin{bmatrix} 1 \\ -2 \\ 0 \\ -1 \end{bmatrix}$$

$$= \begin{bmatrix} -0.2 \\ 0.4 \\ 0 \\ 0.2 \end{bmatrix}$$

$$W_2 = W_1 + \Delta W_1$$

$$= \begin{bmatrix} 1 \\ -1 \\ 0 \\ 0.5 \end{bmatrix} + \begin{bmatrix} -0.2 \\ 0.4 \\ 0 \\ 0.2 \end{bmatrix} = \begin{bmatrix} 0.8 \\ -0.6 \\ 0 \\ 0.7 \end{bmatrix}$$

Step 2 : Take second training pair,

$$\text{Set } X = X_2, \quad d = d_2$$

Scanned by CamScanner

$$X = \begin{bmatrix} 0 \\ 1.5 \\ -0.5 \\ -1 \end{bmatrix}, d = -1$$

$$\text{Compute} \rightarrow net_2 = W_2^t X = [0.8 \; -0.6 \; 0 \; 0.7] \begin{bmatrix} 0 \\ 1.5 \\ -0.5 \\ -1 \end{bmatrix} = -1.6$$

$O_2 = \text{Sign}(net_2) = \text{Sign}(-1.6) = -1$

Here $d = -1$ and $o_2 = -1$

So, $(d - o_2) = 0$

Hence, $\Delta W_2 = 0$. Thus there is no change in weight

$$\therefore W_3 = W_2 = \begin{bmatrix} 0.8 \\ -0.6 \\ 0 \\ 0.7 \end{bmatrix}$$

Step 3 : Take third training pair,

$$\text{Set } X = X_3, \; d = d_3$$

$$X = \begin{bmatrix} -1 \\ 1 \\ 0.5 \\ -1 \end{bmatrix}, \quad d = 1$$

$$\text{Compute} \rightarrow net_3 = W_3^t X$$

$$= [0.8 \; -0.6 \; 0 \; 0.7] \begin{bmatrix} -1 \\ 1 \\ 0.5 \\ -1 \end{bmatrix} = -2.1$$

$o_3 = \text{Sign}(net_3) = \text{sign}(-2.1) = -1$

Here $d = 1$ and $o_3 = -1$

$$\therefore \Delta W_3 = c\,(d - o_3)\,X$$

$$= (0.1)(1 - (-1)) \begin{bmatrix} -1 \\ 1 \\ 0.5 \\ -1 \end{bmatrix} = \begin{bmatrix} -0.2 \\ 0.2 \\ 0.1 \\ -0.2 \end{bmatrix}$$

$$W_4 = W_3 + \Delta W_3$$

$$= \begin{bmatrix} 0.8 \\ -0.6 \\ 0 \\ 0.7 \end{bmatrix} + \begin{bmatrix} -0.2 \\ 0.2 \\ 0.1 \\ -0.2 \end{bmatrix} = \begin{bmatrix} 0.6 \\ -0.4 \\ 0.1 \\ 0.5 \end{bmatrix}$$

Ex. 5.8.3 : Determine the weights after four steps of training for perceptron learning rule of a single neuron network starting with initial weights :
$W = [0 \; 0]^t$, inputs as $X_1 = [2 \; 2]^t$,
$X_2 = [1 \; -2]^t$, $X_3 = [-2 \; 2]^t$, $X_4 = [-1 \; 1]^t$,
$d_1 = 0$, $d_2 = 1$, $d_3 = 0$, $d_4 = 1$ and $c = 1$.

MU - Dec. 13, 10 M

Scanned by CamScanner

Soln. :

Here, we use **unipolar binary** activation function because desired values given are 0's and 1's for perceptron learning,

$$net_i = W_i^t X$$

$$o_i = f(net_i)$$

$$\Delta W_i = c(d - o_i) X$$

$$\text{Here } o_i = \begin{cases} 1 , & net_i \geq 0 \\ 0 , & net_i < 0 \end{cases}$$

Step 1 : Take first training pair,

$$\text{set } X = X_1 \text{ and } d = d_1 \Rightarrow X = \begin{bmatrix} 2 \\ 2 \end{bmatrix} \text{ and } d = 0$$

$$net_1 = W_1^t X$$

Here W_1 is the initial weight, $W_1^t = [0 \;\; 0]$

$$\therefore \; net_1 = [0 \;\; 0] \begin{bmatrix} 2 \\ 2 \end{bmatrix}$$

$$= 0$$

$$o_1 = f(0) = +1$$

$$\Delta W_1 = c(d - o_1) X = (1)(0 - 1) \cdot \begin{bmatrix} 2 \\ 2 \end{bmatrix} = \begin{bmatrix} -2 \\ -2 \end{bmatrix}$$

$$\therefore \; W_2 = W_1 + \Delta W_1 = \begin{bmatrix} 0 \\ 0 \end{bmatrix} + \begin{bmatrix} -2 \\ -2 \end{bmatrix} = \begin{bmatrix} -2 \\ -2 \end{bmatrix}$$

Step 2 : Take second training pair,

$$\text{set } X = X_2 \quad \text{and } d = d_2$$

$$X = \begin{bmatrix} 1 \\ -2 \end{bmatrix} \text{ and } d = 1$$

$$net_2 = W_2^t X = [-2 \;\; -2] \begin{bmatrix} 1 \\ -2 \end{bmatrix}$$

$$= -2 + 4 = 2$$

$$o_2 = f(2) = +1$$

$$\Delta W_2 = c \cdot (d - o_2) \cdot X$$

$$= (1)(1 - 1) \begin{bmatrix} 1 \\ -2 \end{bmatrix} = 0$$

Hence there is no change in weights.

$$\therefore \; W_3 = W_2 = \begin{bmatrix} -2 \\ -2 \end{bmatrix}$$

Step 3 : Take third training pair

$$\text{set } X = X_3 \quad \text{and } d = d_3$$

$$X = \begin{bmatrix} -2 \\ 2 \end{bmatrix} \text{ and } d = 0$$

$$net_3 = W_3^t X$$

Scanned by CamScanner

$$= [-2 \ -2]\begin{bmatrix}-2\\2\end{bmatrix}$$

$$= 4-4=0$$

$$o_3 = f(0) = +1$$

$$\Delta W_3 = c\ (d-o_3)\cdot X$$

$$= (1)\ (0-1)\begin{bmatrix}-2\\2\end{bmatrix} = \begin{bmatrix}2\\-2\end{bmatrix}$$

$$W_4 = W_3 + \Delta W_3 = \begin{bmatrix}-2\\-2\end{bmatrix} + \begin{bmatrix}2\\-2\end{bmatrix} = \begin{bmatrix}0\\-4\end{bmatrix}$$

Step 4 : Take fourth training pair

Set

$$X = X_4 \quad \text{and} \quad d = d_4$$

$$X = \begin{bmatrix}-1\\1\end{bmatrix} \quad \text{and} \quad d = 1$$

$$net_4 = W_4^t \cdot X = [0 \ -4]\begin{bmatrix}-1\\1\end{bmatrix} = 0-4=-4$$

$$o_4 = f(-4) = 0$$

$$\Delta W_4 = c\ (d-o_4)\ X$$

$$= (1)\ (1-0)\begin{bmatrix}-1\\1\end{bmatrix} = \begin{bmatrix}-1\\1\end{bmatrix}$$

$$W_5 = W_4 + \Delta W_4$$

$$= \begin{bmatrix}0\\-4\end{bmatrix} + \begin{bmatrix}-1\\1\end{bmatrix} = \begin{bmatrix}-1\\-3\end{bmatrix}$$

Ex. 5.8.4 : A neuron with 3 inputs has the weight vector w = [0.1 0.3 −0.2]. The activation function is binary sigmoidal activation function. If input vector is [0.8 0.6 0.4] then find the output of neuron. **MU - May 13, 5 Marks**

Soln. :

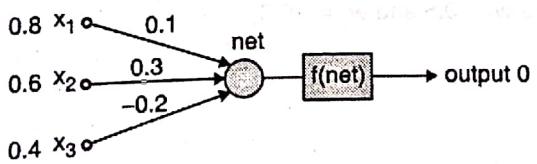


Fig. P. 5.8.4

Here $f(net) = \left[\dfrac{1}{1+e^{(-\lambda\, net)}}\right]$ [Binary sigmoid means the unipolar continuous]

Assume $\lambda = 1$

Computenet = $W^t \cdot X = [0.1 \ 0.3 \ -0.2]\begin{bmatrix}0.8\\0.6\\0.4\end{bmatrix} = 0.08 + 0.18 - 0.08 = 0.18$

Compute o = f (net)

$$= \left[\frac{1}{1+e^{(-0.18)}}\right] = \frac{1}{1+0.8352} = \mathbf{0.5448}$$

Scanned by CamScanner

Ex. 5.8.5 : Implement OR function using perceptron networks for bipolar inputs and targets. **MU - Dec. 11, 10 Marks**

Soln. :

The Truth table for a 2 input OR function is (assuming bipolar inputs target).

x_1	x_2	d
−1	−1	−1
−1	+1	+1
+1	−1	+1
+1	+1	+1

Where x_1, x_2 are the 2 inputs and 'd' is the desired output (target)

Perceptron N/W for OR

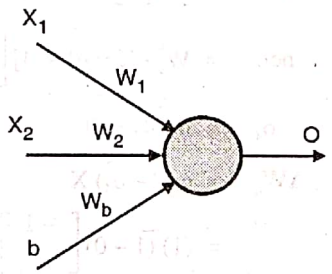


Fig. P. 5.8.5

Shown in Fig. P. 5.8.5 is a perceptron to represent OR function.

The output 'O' of the network is given by

$$O = \text{sign}\,(x_1w_1 + x_2w_2 + bw_b)$$

Now, let us train the n/w to obtain the OR function.

Assume that the initial weights are $w_1 = 0.5$ and $w_2 = -0.5$

Assume bias weight $w_b = 0.5$

Further assume bias input, b = 1

∴ Our initial weight vector

$$w_1 = \begin{bmatrix} 0.5 \\ -0.5 \\ 0.5 \end{bmatrix}$$

And, our training pairs would be :

$$\left(x_1 = \begin{bmatrix} -1 \\ -1 \\ +1 \end{bmatrix}, d_1 = -1\right), \left(x_2 = \begin{bmatrix} -1 \\ +1 \\ +1 \end{bmatrix}, d_2 = +1\right)$$

$$\left(x_3 = \begin{bmatrix} +1 \\ -1 \\ +1 \end{bmatrix}, d_3 = +1\right), \left(x_4 = \begin{bmatrix} +1 \\ +1 \\ +1 \end{bmatrix}, d_4 = +1\right)$$

Assume c = 1 (learning constant)

Scanned by CamScanner

Step 1 :

$$\text{Compute net}_1 = w_1^t x_1$$

$$w_1^t x_1 = [0.5 \quad -0.5 \quad 0.5]\begin{bmatrix}-1\\-1\\+1\end{bmatrix}$$

$$= -0.5 + 0.5 + 0.5 = 0.5$$

$$O_1 = \text{sgn}(\text{net}_1) = \text{sgn}(0.5) = +1$$

$$\text{Now, } d_1 = -1$$

$$\therefore \Delta w_1 = -2c\, x_1$$

$$= -2(1)\begin{bmatrix}-1\\-1\\+1\end{bmatrix} = \begin{bmatrix}2\\2\\-2\end{bmatrix}$$

$$\therefore \Delta w_2 = w_1 + \Delta w_1$$

$$= \begin{bmatrix}0.5\\-0.5\\0.5\end{bmatrix} + \begin{bmatrix}2\\2\\-2\end{bmatrix} = \begin{bmatrix}2.5\\1.5\\-1.5\end{bmatrix}$$

Step 2 :

$$\text{net}_2 = w_2^t x_2$$

$$= [2.5 \quad 1.5 \quad -1.5]\begin{bmatrix}-1\\+1\\+1\end{bmatrix}$$

$$= -2.5 + 1.5 - 1.5$$

$$= -2.5$$

$$O_2 = \text{sgn}(\text{net}_2) = -1$$

$$d_2 = +1$$

$$\therefore \Delta w_2 = 2c\, x_2 = 2(1)\begin{bmatrix}-1\\1\\1\end{bmatrix} = \begin{bmatrix}-2\\2\\2\end{bmatrix}$$

$$\therefore w_3 = w_2 + \Delta w_2$$

$$= \begin{bmatrix}2.5\\1.5\\-1.5\end{bmatrix} = \begin{bmatrix}-2\\2\\2\end{bmatrix} = \begin{bmatrix}0.5\\3.5\\0.5\end{bmatrix}$$

Step 3 :

$$\text{net}_3 = w_3^t x_3$$

$$= [0.5 \quad 3.5 \quad 0.5]\begin{bmatrix}1\\-1\\1\end{bmatrix}$$

$$= 0.5 - 3.5 + 0.5 = -2.5$$

$$O_3 = f(\text{net}_3) = -1$$

$$d_3 = +1$$

Scanned by CamScanner

$$\therefore \Delta w_3 = 2\,c\,x_3 = 2(1)\begin{bmatrix}1\\-1\\1\end{bmatrix} = \begin{bmatrix}2\\-2\\2\end{bmatrix}$$

$$w_4 = w_3 + \Delta w_3$$

$$= \begin{bmatrix}0.5\\3.5\\0.5\end{bmatrix} = \begin{bmatrix}2\\-2\\2\end{bmatrix} = \begin{bmatrix}2.5\\1.5\\2.5\end{bmatrix}$$

Step 4 :

$$net_4 = w_4^t\, x_4$$

$$= [2.5 \;\; 1.5 \;\; 2.5]\begin{bmatrix}+1\\+1\\+1\end{bmatrix}$$

$$= 2.5 + 1.5 + 2.5 = 6.5$$

$$O_4 = sgn(net_4) = sgn\,(6.5) = +1$$

$$d_4 = +1$$

$\therefore$ No change in weights

$$\text{i.e. } \Delta w_4 = 0$$

$$w_5 = w_4 = \begin{bmatrix}2.5\\1.5\\2.5\end{bmatrix}$$

Step 5 :

$$net_5 = w_5^t\, x_1$$

$$= [2.5 \;\; 1.5 \;\; 2.5]\begin{bmatrix}-1\\-1\\+1\end{bmatrix}$$

$$= -2.5 - 1.5 + 2.5 = -1.5$$

$$O_5 = sgn\,(net\ 5) = sgn\,(-1.5) = -1$$

$$d_1 = -1$$

$\therefore$ No change in weights

$$\Delta w_5 = 0 \qquad w_6 = w_5 = \begin{bmatrix}2.5\\1.5\\2.5\end{bmatrix}$$

Now, after Step 5, we have,

$$w_6 = w_5 = w_4$$

i.e. Weight vector is not changing after 2 consecutive steps. Hence, we stop training and this means that the training in complete final weight vector after training is

$$W = \begin{bmatrix}2.5\\1.5\\2.5\end{bmatrix}$$

Scanned by CamScanner

∴ The perceptron network for OR function is as shown in Fig. P. 5.8.5(a).

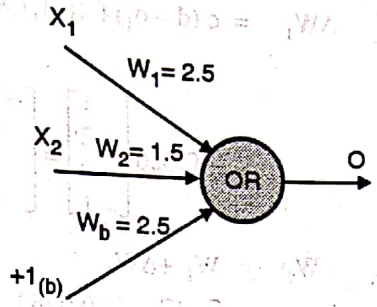


Fig. P. 5.8.5 (a)

Where $O = \text{sgn}(2.5x_1 + 1.5x_2 + 2.5)$

Ex. 5.8.6 : Consider the following set of input training vectors.

$$X_1 = \begin{bmatrix} 1 \\ -2 \\ 0 \\ -1 \end{bmatrix} \quad X_2 = \begin{bmatrix} 0 \\ 1.5 \\ -0.5 \\ -1 \end{bmatrix} \quad X_3 = \begin{bmatrix} -1 \\ 1 \\ 0.5 \\ -1 \end{bmatrix}$$

$d_1 = -1$, $d_2 = -1$ and $d_3 = +1$ are desired responses for X_1, X_2 and X_3 respectively. Initial weight vector

$$W_1 = \begin{bmatrix} 1 \\ -1 \\ 0 \\ 0.5 \end{bmatrix}$$

The learning constant c = 0.1 and λ = 1.Use delta learning rule to calculate the final weights. Use bipolar continuous activation function.

Soln. :

For delta learning with bipolar continuous activation function.

$$net_i = W_i^t X \quad ...(1)$$

$$o_i = f(net_i) = \frac{2}{1 + \exp(-\lambda net)} - 1 \quad ...(2)$$

$$f'(net_i) = \frac{1}{2}\left(1 - o_i^2\right) \quad ...(3)$$

$$\Delta W_i = c\,(d - o_i) f'(net_i)\, X \quad ...(4)$$

Step 1 : Take 1st training pair, $X = X_1$ and $d = d_1$

$$X = \begin{bmatrix} 1 \\ -2 \\ 0 \\ -1 \end{bmatrix}, \; d = -1$$

Compute → $net_1 = W_1^t X = [1 \;\; -1 \;\; 0 \;\; 0.5] \begin{bmatrix} 1 \\ -2 \\ 0 \\ -1 \end{bmatrix} = [1 + 2 - 0.5] = 2.5$

$$o_1 = f(net_1) = \frac{2}{1 + \exp(-2.5)} - 1 = \frac{2}{1 + 0.0820} - 1 = 0.848$$

$$f'(net_1) = \frac{1}{2}\left(1 - o_1^2\right) = \frac{1}{2}\left(1 - 0.848^2\right) = 0.140$$

Scanned by CamScanner

$$\Delta W_1 = c\,(d - o_1)\, f'(net_1)\, X = (0.1)\,(-1 - 0.848)\,(0.14)\begin{bmatrix}1\\-2\\0\\-1\end{bmatrix}$$

$$= -0.026\begin{bmatrix}1\\-2\\0\\-1\end{bmatrix} = \begin{bmatrix}-0.026\\0.052\\0\\0.026\end{bmatrix}$$

$$W_2 = W_1 + \Delta W_1$$

$$= \begin{bmatrix}1\\-1\\0\\0.5\end{bmatrix} + \begin{bmatrix}-0.026\\0.052\\0\\0.026\end{bmatrix} = \begin{bmatrix}0.974\\-0.948\\0\\0.526\end{bmatrix}$$

Step 2: Take 2nd training pair, $X = X_2$ and $d = d_2$

$$X = \begin{bmatrix}0\\1.5\\-0.5\\-1\end{bmatrix}, d = -1$$

Compute $\rightarrow net_2 = W_2^t\, X = [0.974 \;\; -0.948 \;\; 0 \;\; 0.526]\begin{bmatrix}0\\1.5\\-0.5\\-1\end{bmatrix}$

$$= -1.422 - 0.526$$

$$= -1.948$$

$$O_2 = f(net_2) = \frac{2}{1 + \exp(1.948)} - 1 = \frac{2}{8.0146} - 1 = -0.75$$

$$f'(net_2) = \frac{1}{2}\left(1 - o_2^2\right) = \frac{1}{2}\,(1 - (-0.75)^2) = 0.2187$$

$$\Delta W_2 = c\,(d - o_2)\, f'(net_2)\, X = (0.1)\,(-1 - (-0.75)) \times (0.2187)\begin{bmatrix}0\\1.5\\-0.5\\-1\end{bmatrix}$$

$$= (-0.00546)\begin{bmatrix}0\\1.5\\-0.5\\-1\end{bmatrix} = \begin{bmatrix}0\\-0.00819\\0.00273\\0.00546\end{bmatrix}$$

$$W_3 = W_2 + \Delta W_2 = \begin{bmatrix}0.974\\-0.948\\0\\0.526\end{bmatrix} + \begin{bmatrix}0\\-0.00819\\0.00273\\0.00546\end{bmatrix} = \begin{bmatrix}0.974\\-0.956\\0.0027\\0.5315\end{bmatrix}$$

Step 3 : Take 3rd training pair, $X = X_3$ and $d = d_3$

$$X = \begin{bmatrix}-1\\1\\0.5\\-1\end{bmatrix}, d = 1$$

Compute $\rightarrow net_3 = W_3^t\, X = [0.974 - 0.956 \;\; 0.0027 \;\; 0.531]\begin{bmatrix}-1\\1\\0.5\\-1\end{bmatrix}$

Scanned by CamScanner

$$= -0.974 - 0.956 + 0.00135 - 0.5315 = -2.462$$

$$o_3 = f(net_3) = \frac{2}{1 + \exp(2.462)} - 1 = \frac{2}{12.728} - 1 = -0.843$$

$$f'(net_3) = \frac{1}{2}(1 - o_3^2) = \frac{1}{2}(1 - (-0.843)^2) = 0.145$$

$$\Delta W_3 = (0.1)(d - o_3) f'(net_3) X = (0.1)(1 - (-0.843))(0.145) \begin{bmatrix} -1 \\ 1 \\ 0.5 \\ -1 \end{bmatrix}$$

$$\Delta W_3 = (0.0267) \begin{bmatrix} -1 \\ +1 \\ 0.5 \\ -1 \end{bmatrix} = \begin{bmatrix} -0.0267 \\ 0.0267 \\ 0.0133 \\ -0.0267 \end{bmatrix}$$

$$W_4 = W_3 + \Delta W_3 = \begin{bmatrix} 0.974 \\ -0.956 \\ 0.0027 \\ 0.5315 \end{bmatrix} + \begin{bmatrix} -0.0267 \\ 0.0267 \\ 0.0133 \\ -0.0267 \end{bmatrix} = \begin{bmatrix} 0.947 \\ -0.929 \\ 0.016 \\ 0.504 \end{bmatrix}$$

Ex. 5.8.7 : Perform two training steps of the network using delta learning rule for $\lambda = 1$, $c = 0.25$. Train the network using the following data pairs.

$\left(X_1 = \begin{bmatrix} 2 \\ 0 \\ -1 \end{bmatrix}, d_1 = -1\right)\left(X_2 = \begin{bmatrix} 1 \\ -2 \\ -1 \end{bmatrix}, d_2 = +1\right)$. The initial weights are $W_1 = [1\ 0\ 1]^t$

Soln. :

For delta learning rule

$$net_i = W_i^t X \quad \text{...(1)}$$

$$o_i = f(net_i) = \frac{2}{1 + \exp(-\lambda\, net_i)} - 1 \quad \text{...(2)}$$

$$f'(net_i) = \frac{1}{2}\left(1 - o_i^2\right) \quad \text{...(3)}$$

$$\Delta W_i = c(d - o_i)\left[\frac{1}{2}\left(1 - o_i^2\right)\right] \quad \text{...(4)}$$

Step 1 : Take 1st training pair $X = X_1$ and $d = d_1$

$$X = \begin{bmatrix} 2 \\ 0 \\ -1 \end{bmatrix}, d = -1$$

Compute $\rightarrow net_1 = W_1^t X = [1\ \ 0\ \ 1] \begin{bmatrix} 2 \\ 0 \\ -1 \end{bmatrix} = 2 - 1 = 1$

$$o_1 = \frac{2}{1 + \exp(-1)} - 1 = 0.463$$

$$f'(net_1) = \frac{1}{2}(1 - (0.463)^2) = 0.393$$

$$\Delta W_1 = (0.25)(-1 - 0.463)(0.393) \begin{bmatrix} 2 \\ 0 \\ -1 \end{bmatrix}$$

Scanned by CamScanner

$$= (-0.1437)\begin{bmatrix}2\\0\\-1\end{bmatrix} = \begin{bmatrix}-0.287\\0\\0.1437\end{bmatrix}$$

$$W_2 = W_1 + \Delta W_1 = \begin{bmatrix}1\\0\\1\end{bmatrix} + \begin{bmatrix}-0.287\\0\\0.1437\end{bmatrix} = \begin{bmatrix}0.713\\0\\1.1437\end{bmatrix}$$

Step 2 : Take 2nd training pair, $X = X_2$ and $d = d_2$

$$X = \begin{bmatrix}1\\-2\\-1\end{bmatrix}, d = 1$$

$$\text{Compute} \rightarrow net_2 = W_2^t X = [0.713 \quad 0 \quad 1.1437]\begin{bmatrix}1\\-2\\-1\end{bmatrix}$$

$$= [0.713 \quad -1.1437] = -0.4307$$

$$o_2 = -0.2119$$

$$f'(net_2) = \frac{1}{2}(1-(-0.2119)^2) = \frac{1}{2}(1-0.0449) = 0.477$$

$$\Delta W_2 = (0.25)(1-(-0.2119)) \times (0.477)\, X_2$$

$$= (0.145)\begin{bmatrix}1\\-2\\-1\end{bmatrix} = \begin{bmatrix}0.145\\-0.29\\-0.145\end{bmatrix}$$

$$W_3 = W_2 + \Delta W_2 = \begin{bmatrix}0.713\\0\\1.143\end{bmatrix} + \begin{bmatrix}0.145\\-0.29\\-0.145\end{bmatrix} = \begin{bmatrix}0.858\\-0.29\\0.998\end{bmatrix}$$

Ex. 5.8.8 : A single neuron network using f (net) = sgn (net) has been trained using the pairs of (x_i, d_i) as given below :

$X_1 = [1 \quad -2 \quad 3 \quad -1]^t, d_1 = -1$

$X_2 = [0 \quad -1 \quad 2 \quad -1]^t, d_2 = 1$

$X_3 = [-2 \quad 0 \quad -3 \quad -1]^t, d_3 = -1$

The final weights obtained using perception rule are :

$W_4 = [3 \quad 2 \quad 6 \quad 1]^t$

Knowing that correction has been performed at each step for c = 1, determine the following weights :

(a) W_3, W_2, W_1 by backtracking the training

(b) W_5, W_6, W_7 obtained for step 4, 5, 6 of training by reusing the sequence $(X_1, d_1), (X_2, d_2), (X_3, d_3)$.

MU - Dec. 11. 10 marks

Soln. :

(a) Backtracking

Step 3 : $W_4 = \begin{bmatrix}3\\2\\6\\1\end{bmatrix}$

$$W_4 = W_3 + \Delta W_3$$

$$W = c \cdot (d_3 - o_3) \cdot X_3$$

Scanned by CamScanner

We know that the correction has been performed at every step this means, actual output and desired outputs were not same.

This means they we exactly opposite of each other.

$$\text{Since } d_1 = -1 \Rightarrow o_1 = +1$$

$$d_2 = +1 \Rightarrow o_2 = -1$$

$$d_3 = -1 \Rightarrow o_3 = +1$$

$$\therefore \Delta W_3 = (1)(-1-1)\begin{bmatrix} -2 \\ 0 \\ -3 \\ -1 \end{bmatrix} = \begin{bmatrix} 4 \\ 0 \\ 6 \\ 2 \end{bmatrix}$$

$$\therefore W_4 = W_3 + \Delta W_3$$

$$\therefore W_3 = W_4 - \Delta W_3$$

$$= \begin{bmatrix} 3 \\ 2 \\ 6 \\ 1 \end{bmatrix} - \begin{bmatrix} 4 \\ 0 \\ 6 \\ 2 \end{bmatrix} = \begin{bmatrix} -1 \\ 2 \\ 0 \\ -1 \end{bmatrix}$$

Step 2 : $W_3 = W_2 + \Delta W_2$

$$\therefore \Delta W_2 = c(d_2 - o_2) X_2$$

$$= (1)(1-(-1))\begin{bmatrix} 0 \\ -1 \\ 2 \\ -1 \end{bmatrix} = \begin{bmatrix} 0 \\ -2 \\ 4 \\ -2 \end{bmatrix}$$

$$\text{Now } W_2 = W_3 - \Delta W_2$$

$$= \begin{bmatrix} -1 \\ 2 \\ 0 \\ -1 \end{bmatrix} - \begin{bmatrix} 0 \\ -2 \\ 4 \\ -2 \end{bmatrix} = \begin{bmatrix} -1 \\ 4 \\ -4 \\ 1 \end{bmatrix}$$

Step 1 : $W_2 = W_1 + \Delta W_1$ and $\Delta W_1 = c(d_1 - o_1) X_1$

$$= (1)(1-(-1))\begin{bmatrix} 1 \\ -2 \\ 3 \\ -1 \end{bmatrix} = (-2)\begin{bmatrix} 1 \\ -2 \\ 3 \\ -1 \end{bmatrix} = \begin{bmatrix} -2 \\ 4 \\ -6 \\ 2 \end{bmatrix}$$

$$\therefore W_1 = W_2 - \Delta W_1$$

$$= \begin{bmatrix} -1 \\ 4 \\ -4 \\ 1 \end{bmatrix} - \begin{bmatrix} -2 \\ 4 \\ -6 \\ 2 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 2 \\ -1 \end{bmatrix}$$

Scanned by CamScanner

Thus,

$$W_1 = \begin{bmatrix} 1 \\ 0 \\ 2 \\ -1 \end{bmatrix}, \quad W_2 = \begin{bmatrix} -1 \\ 4 \\ -4 \\ 1 \end{bmatrix} \quad W_3 = \begin{bmatrix} -1 \\ 2 \\ 0 \\ -1 \end{bmatrix}$$

(b) Find W_5, W_6 and W_7

Step 4 : Take first training pair

set $X = X_1$ and $d = d_1$

$$X = \begin{bmatrix} 1 \\ -2 \\ 3 \\ -1 \end{bmatrix} \quad \text{and} \quad d = -1$$

$$net_4 = W_4^t X$$

$$= [3 \ 2 \ 6 \ 1] \begin{bmatrix} 1 \\ -2 \\ 3 \\ -1 \end{bmatrix}$$

$$= 3 - 4 + 18 - 1 = 16$$

$$o_4 = \text{sgn}(net_4) = +1$$

$$\Delta W_4 = c(d - o_4) \cdot X$$

$$= (1)(-1-1) \begin{bmatrix} 1 \\ -2 \\ 3 \\ -1 \end{bmatrix} = (1)(-2) \begin{bmatrix} 1 \\ -2 \\ 3 \\ -1 \end{bmatrix} = \begin{bmatrix} -2 \\ 4 \\ -6 \\ 2 \end{bmatrix}$$

$$\therefore \ W_5 = W_4 + \Delta W_4 = \begin{bmatrix} 3 \\ 2 \\ 6 \\ 1 \end{bmatrix} + \begin{bmatrix} -2 \\ 4 \\ -6 \\ 2 \end{bmatrix} = \begin{bmatrix} 1 \\ 6 \\ 0 \\ 3 \end{bmatrix}$$

Step 5 : Take second training pair,

set $X = X_2$ and $d = d_2$

$$X = \begin{bmatrix} 0 \\ -1 \\ 2 \\ -1 \end{bmatrix} \quad d = 1$$

$$net_5 = W_5^t X = [1 \ 6 \ 0 \ 3] \begin{bmatrix} 0 \\ -1 \\ 2 \\ -1 \end{bmatrix}$$

$$= 0 - 6 + 0 - 3 = -9$$

Scanned by CamScanner

$$o_5 = \text{sgn}(-9) = -1$$

$$\Delta W_5 = c(d - o_5) \cdot X$$

$$= (1)(1-(-1)) \begin{bmatrix} 0 \\ -1 \\ 2 \\ -1 \end{bmatrix} = 2 \begin{bmatrix} 0 \\ -1 \\ 2 \\ -1 \end{bmatrix} = \begin{bmatrix} 0 \\ -2 \\ 4 \\ -2 \end{bmatrix}$$

$$\therefore \; W_6 = W_5 + \Delta W_5$$

$$= \begin{bmatrix} 1 \\ 6 \\ 0 \\ 3 \end{bmatrix} + \begin{bmatrix} 0 \\ -2 \\ 4 \\ -2 \end{bmatrix} = \begin{bmatrix} 1 \\ 4 \\ 4 \\ 1 \end{bmatrix}$$

Step 6 : Take third training pair,

set $X = X_3$ and $d = d_3$

$$X = \begin{bmatrix} -2 \\ 0 \\ -3 \\ -1 \end{bmatrix} \quad d = -1$$

$$net_6 = W_6^t X$$

$$= [1 \;\; 4 \;\; 4 \;\; 1] \begin{bmatrix} -2 \\ 0 \\ -3 \\ -1 \end{bmatrix} = -2 + 0 - 12 - 1 = -14$$

$$o_6 = \text{sgn}(-14) = -1$$

$$\Delta W_6 = c(d - o_6) \cdot X = (1)(-1-(-1)) \begin{bmatrix} -2 \\ 0 \\ -3 \\ -1 \end{bmatrix} = 0$$

$$\therefore \; W_7 = W_6 = \begin{bmatrix} 1 \\ 4 \\ 4 \\ 1 \end{bmatrix}$$

$$\text{Hence } W_5 = \begin{bmatrix} 1 \\ 6 \\ 0 \\ 3 \end{bmatrix} \quad W_6 = \begin{bmatrix} 1 \\ 4 \\ 4 \\ 1 \end{bmatrix} \quad W_7 = \begin{bmatrix} 1 \\ 4 \\ 4 \\ 1 \end{bmatrix}$$

Ex. 5.8.9 : Apply Back propagation Algorithm to find the final weights for the following net. Inputs : x = [0.0, 1.0]. Weights between hidden and output layers : w = [0.4, 0.2]. Blass on the Output Node O is W_o = [– 0.4]. Weights between input and hidden layer : v = [2, 1 ; 1.2]. Bias on Hidden Unit nodes are V_o = [0.1 0.3]. Desired output : d = 1.0.

MU - Dec. 15. 10 Marks

Scanned by CamScanner

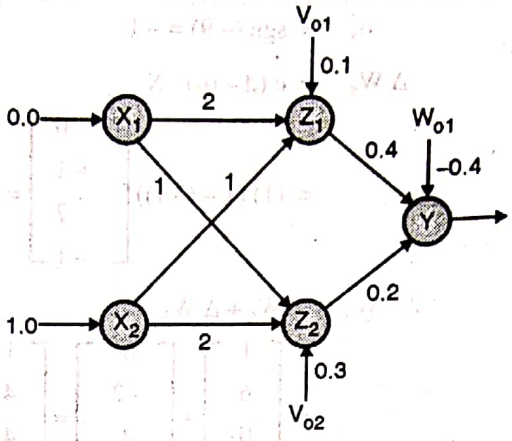


Fig. P.5.8.9

Soln. :

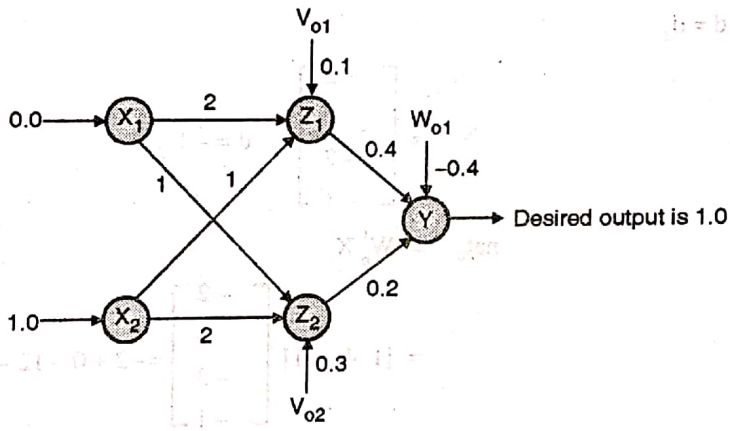


Fig. P. 5.8.9(a)

(I) Forward Pass :

Computing forward activations.

(i) Computing hidden layer activations (Z_1 and Z_2)

$$\text{net } z_1 = V_{11}X_1 + V_{12}x_2 + V_{01}$$

$$= 2(0) + 1(1) + 0.1$$

$$= 0 + 1 + 0.1$$

$$= 1.1$$

$$Z_1 = f(\text{net}z_1)$$

Using bipolar continuous activation function, we have

$$f(\text{net}) = \frac{2}{1+e^{-\lambda \text{net}}} - 1 \qquad [\text{Assume } \lambda = 1]$$

$$= \frac{2}{1+e^{-\text{net}}} - 1$$

$$Z_1 = f(\text{net}z_1) = \frac{2}{1+e^{-\text{net}z_1}} - 1 = \frac{2}{1+e^{-1.1}} - 1$$

$$z_1 = 0.5005$$

Scanned by CamScanner

$$\text{Net } z_2 = v_{21} x_1 + v_{22} x_2 + V_{o2}$$

$$= 1(0) + 2(1) + 0.3$$

$$= 2.3$$

$$z_2 = f(\text{net } z_2) = \frac{2}{1+e^{-2.3}} - 1$$

$$z_2 = 0.8178$$

(ii) Computing output layer activations (y)

$$\text{net}_y = w_1 z_1 + w_2 z_2 + w_{01}$$

$$= 04(0.5005) + 0.2\,(0.8178) + (-0.4)$$

$$\text{net}_y = -0.03624$$

$$y = f(\text{net}_y) = \frac{2}{1+e^{-0.03624}} - 1$$

$$y = -0.0181$$

(II) Backward Pass :

Computing error signal vectors for both layers.

(i) Computing error signal terms δ_{ok} for output layer

$$\delta_{ok} = \frac{1}{2}(d_k - o_k)\left(1 - O_k^2\right), \text{ for } k = 1, 2, \ldots K$$

$$\delta_y = \frac{1}{2}(d - y)(1 - y^2)$$ Since, there is only one neuron 'y'

$$\delta_y = \frac{1}{2}(1 - (-0.0181))(1 - (-0.0181)^2)$$

$$\delta_y = 0.5089$$

(ii) Computing error signal terms of the hidden layer (δzj)

$$\delta_{zj} = \frac{1}{2}\left(1 - z_j^2\right) \sum_{k=1}^{k} \delta_{ok} W_{kj}, j = 1, 2, \ldots J$$

$$\delta_{z1} = \frac{1}{2}\left(1 - z_1^2\right) \delta_y w_1$$

$$= \frac{1}{2}(1 - 0.5005^2)(0.5089)(0.4)$$

$$\delta_{z1} = 0.0763$$

$$\delta_{z2} = \frac{1}{2}\left(1 - z_2^2\right) \delta_y w_2$$

$$= \frac{1}{2}(1 - 0.8178^2)(0.5089)(0.2)$$

$$\delta_{z2} = 0.0169$$

(iii) Adjusting output layer weights (w_1 and w_2)

$$W_{kj} = W_{kj} + \eta \delta_y z_j \quad (\text{Assume } \eta = 1)$$

$$W_{1new} = W_{1old} + \delta_y z_1$$

Scanned by CamScanner

$$= 0.4 + (0.5089)\ (0.5005)$$

$$\mathbf{w_1} = \mathbf{0.6547}$$

$$w_2 = w_2 + \delta_y z_2$$

$$= 0.2 + (0.5089)\ (0.8178)$$

$$\mathbf{w_2} = \mathbf{0.6162}$$

$$\text{Bias } w_{01} = w_{01} + \delta_y(1)$$

$$= -0.4 + 0.5089$$

$$W_{01} = 0.1089$$

(iv) Adjusting hidden layer weights ($v_{11}, v_{12}, v_{21}, v_{22}$)

$$v_{ji} = v_{ji} + \eta\delta_{zj}x_i \text{ for } j = 1, 2 \quad J \text{ and } i = 1, 2, ..I$$

$$v_{11} = v_{11} + \delta_{z1}x_1 = 2 + 0.0763\ (0) = 2$$

$$v_{12} = v_{12} + \delta_{z1}\ x_2 = 1 + 0.0763(1) = 1.0763$$

$$v_{21} = v_{21} + \delta_{z2}\ x_1 = 1 + 00169\ (0) = 1$$

$$v_{22} = v_{22} + \delta_{z2}x_2 = 2 + 0.0169(1) = 2.0169$$

$$v_{o1} = v_{o1} + \delta_{z1}\ (1) = 0.1 + 0.0763(1) = 0.1763$$

$$v_{o2} = v_{o2} + \delta_{z2}\ (1) = 0.3 + 0.0169(1) = 0.3169$$

$$\text{Cycle error E} = ½\,(d - y)^2 + E = \frac{1}{2}\ (1 - (-0.0181))^2 = 0.5183$$

Final weights:

$$w_1 = 0.6547,\ w_2 = 0.6162,\ wo1 = 0.1089,\ v_{11} = 2,\ v_{12} = 1.0763$$

$$v_{21} = 1,\ \ v_{22} = 2.0169,\ v_{o1} = 0.1763,\ v_{o2} = 0.3169$$

Review Questions

Q. 1 Explain linearly separable and non-linearly separable patterns with examples.

Q. 2 Explain Error back Propagation Algorithm with the help of flowchart.

Q. 3 Write a short note on Kohonen self organizing network.

Q. 4 Explain McCulloch Pitts Neuron Model with help of an example.

Q. 5 A neuron with 4 inputs has the weight vector $w = [1, 2, 3, 4]^t$. The activation function is linear, that is, the activation function is given by f (net) = 2* net. If the input vector is $X = [5, 6, 7, 8]^t$ then find the output of the neuron.

Q. 6 Explain Error back propagation training algorithm with the help of a flowchart.

Q. 7 A single neuron network using f (net) = sgn (net) has been trained using the pairs of (X_i, d_i) as given below :

$X_1 = [1 - 2\ 3 - 1]^t$, $d_1 = -1$

$X_2 = [0 - 1\ 2 - 1]^t$, $d_2 = 1$

$X_3 = [-2\ 0 - 3 - 1]^t$, $d_3 = -1$

The final weights obtained using the perceptron rule are :

Scanned by CamScanner

$W_4 = [3\ 2\ 6\ 1]^t$

Knowing that correction has been performed in each step for c = 1, determine the following weights :

(a) W_3, W_2, W_1 by backtracking the training.

(b) W_5, W_6, W_7 obtained for steps 4, 5, 6 of training by reusing the sequence $(X_1, d_1), (X_2, d_2), (X_3, d_3)$,

Q. 8 Explain with example perceptron learning rule.

Q. 9 Determine the weights after one iteration for hebbian learning of a single neuron network starting with initial weights $w = [1, -1]$, inputs as $X_1 = [1, -2]$, $X_2 = [2, 3]$, $X_3=[1, -1]$ and c = 1.

Use (i) Bipolar binary activation function

(ii) Bipolar continuous activation function.

Q. 10 Explain Single Continuous Perceptron Training Algorithm (SCPTA).

Q. 11 Explain with neat diagram supervised and unsupervised learning.

Q. 12 A neuron with 3 inputs has the weight vector w = [0.1 0.3 – 0.2]. The activation function is binary sigmoidal activation function. If input vector is [0.8 0.6 0.4] then find the output of neuron.

Q. 13 Differentiate between supervised and unsupervised learning.

Q. 14 Explain Kohonen's self organizing neural network.

Q. 15 Determine the weights after four steps of training for percenptron learning rule of a single neuron network starting with initial weights :

$W = [0\ 0]^t$, inputs as $X_1 = [2\ 2]^t$,

$X_2 = [1\ -2]^t$, $X_3 = [-2\ 2]^t$, $X_4 = [-1\ 1]^t$,

$d_1 = 0$, $d_2 = 1$, $d_3 = 0$, $d_4 = 1$ and c = 1.

Q. 16 Prove the following identities :

(i) For unipolar continuous activation function

$f^1(net) = 0\ (1 - 0)$

(ii) For bipolar continuous activation function :

$f^1\ (net) = \frac{1}{2}(1 - 0^2)$

Q. 17 What is learning in neural networks ? Differentiate between supervised and unsupervised learning.

□□□

Scanned by CamScanner