Program : **B.E**

Subject Name: **Soft Computing**

Subject Code: **CS-8001**

Semester: **8th**

UNIT-2 Notes

Introduction of MLP (Multi Layer Perceptron)

1. The multilayer perceptron (MLP) is a hierarchical structure of several perceptrons, and overcomes the shortcomings of these single-layer networks see figure 1.7.
2. The multilayer perceptron is an artificial neural network that learns nonlinear function mappings. The multilayer perceptron is capable of learning a rich variety of nonlinear decision surfaces.
3. Nonlinear functions can be represented by multilayer perceptrons with units that use nonlinear activation functions. Multiple layers of cascaded linear units still produce only linear mappings.
4. A neural network with one or more layers of nodes between the input and the output nodes is called multilayer network.
5. The multilayer network structure, or architecture, or topology, consists of an input layer, two or more hidden layers, and one output layer. The input nodes pass values to the first hidden layer, its nodes to the second and so on till producing outputs.
6. A network with a layer of input units, a layer of hidden units and a layer of output units is a two-layer network. A network with two layers of hidden units is a three-layer network, and so on. A justification for this is that the layer of input units is used only as an input channel and can therefore be discounted.

A two-layer neural network that implements the function:

$$f(x) = \Sigma (w_{jk} \Sigma (w_{ij} x_i + w_{0j}) + w_{0k})$$

where: x is the input vector,
w_{0j} and w_{0k} are the thresholds,
w_{ij} are the weights connecting the input with the hidden nodes
w_{jk} are the weights connecting the hidden with the output nodes
s is the sigmoid activation function.

These are the hidden units that enable the multilayer network to learn complex tasks by extracting progressively more meaningful information from the input examples.

7. The multilayer network MLP has a highly connected topology since every input is connected to all nodes in the first hidden layer, every unit in the hidden layers is connected to all nodes in the next layer, and so on.
8. The input signals, initially these are the input examples, propagate through the neural network in a forward direction on a layer-by-layer basis, that is why they are often called feed forward multilayer networks.

Two kinds of signals pass through these networks:
- function signals: the input examples propagated through the hidden units and processed by their activation functions emerge as outputs;
- error signals: the errors at the otuput nodes are propagated backward layer-by-layer through the network so that each node returns its error back to the nodes in the previous hidden layer.

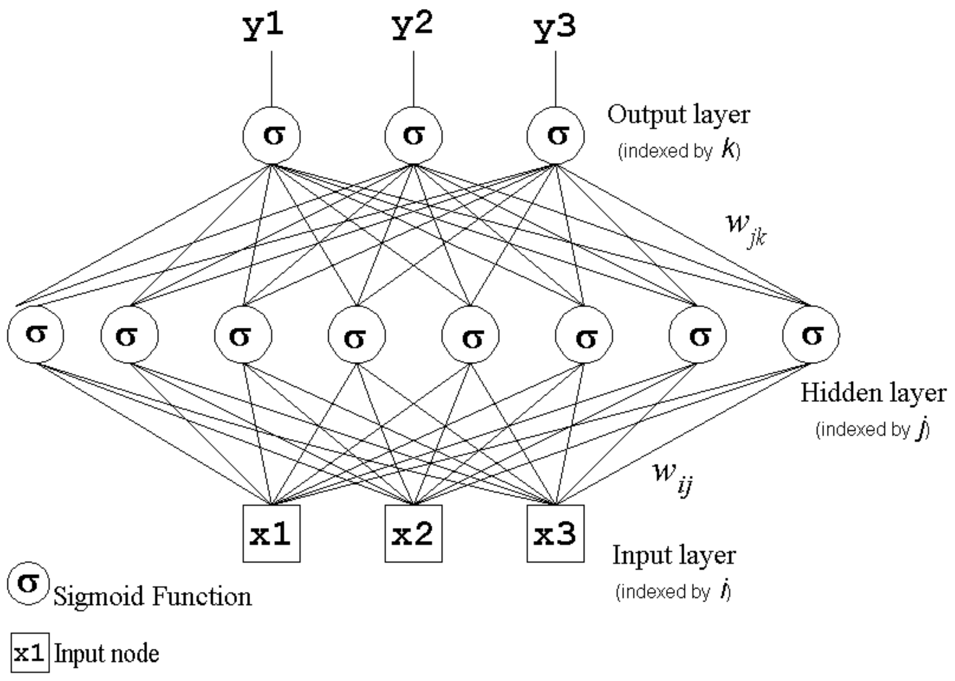

Fig 1.7: MLP Neural Network

Different activation functions:

An activation function performs a mathematical operation on the signal output. It decides the criteria and nature of output to be generated. Two most common activation functions are:

1. Threshold Function,

A step function is a function is likely used by the original Perceptron. The output is a certain value, A1, if the input sum is above a certain threshold and A0 if the input sum is below a certain threshold. The values used by the Perceptron were A1 = 1 and A0 = 0 see figure 1.8.

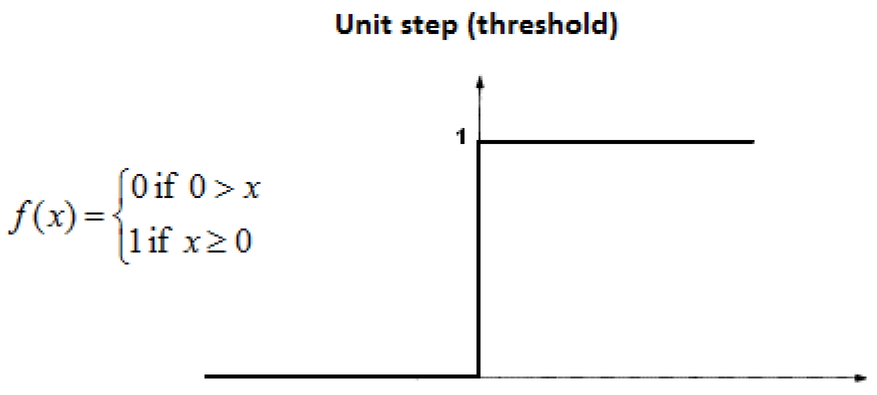

Fig 1.8: Threshold Function

2. Sigmoidal (S shaped) function,

A wide variety of sigmoid functions have been used as the activation function of artificial neurons, including the logistic and hyperbolic tangent functions, see figure 1.9.

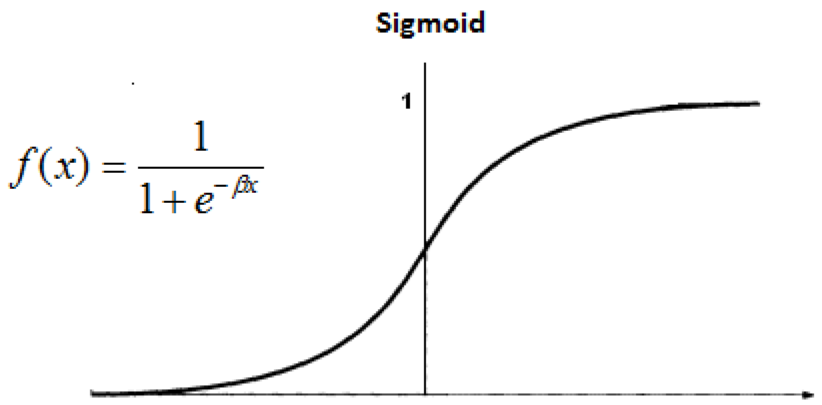

Fig 1.9: Sigmoidal Function

Error back propagation algorithm

1. This algorithm was discovered and rediscovered a number of times. This reference also contains the mathematical details of the derivation of the backpropagation equations, which we shall omit. (This is covered in COMP9444 Neural Networks.)
2. Like perceptron learning, back-propagation attempts to reduce the errors between the output of the network and the desired result.
3. However, assigning blame for errors to hidden nodes (i.e. nodes in the intermediate layers), is not so straightforward. The error of the output nodes must be propagated back through the hidden nodes.
4. The contribution that a hidden node makes to an output node is related to the strength of the weight on the link between the two nodes and the level of activation of the hidden node when the output node was given the wrong level of activation.
5. This can be used to estimate the error value for a hidden node in the penultimate layer, and that can, in turn, be used in making error estimates for earlier layers see figure 1.9.

Derivation of EBPA

- The basic algorithm can be summed up in the following equation (the *delta rule*) for the change to the weight w_{ji} from node *i* to node *j*:

weight change		learning rate	local gradient	input signal to node *j*
Δw_{ji}	=	η	$\times\ \delta_j$	$\times\ y_i$

- where the local gradient δ_j is defined as follows:

1. If node j is an output node, then δ_j is the product of $\phi'(v_j)$ and the error signal e_j, where $\phi(\_)$ is the logistic function and v_j is the total input to node j (i.e. $\Sigma_i\ w_{ji}y_i$), and e_j is the error signal for node j (i.e. the difference between the desired output and the actual output);
2. If node j is a hidden node, then δ_j is the product of $\phi'(v_j)$ and the weighted sum of the δ's computed for the nodes in the next hidden or output layer that are connected to node j.
3. [The actual formula is $\delta_j = \phi'(v_j)$ &Sigma$_k$ $\delta_k w_{kj}$ where k ranges over those nodes for which w_{kj} is non-zero (i.e. nodes k that actually have connections from node j. The δ_k values have already been computed as they are in the output layer (or a layer closer to the output layer than node j).]

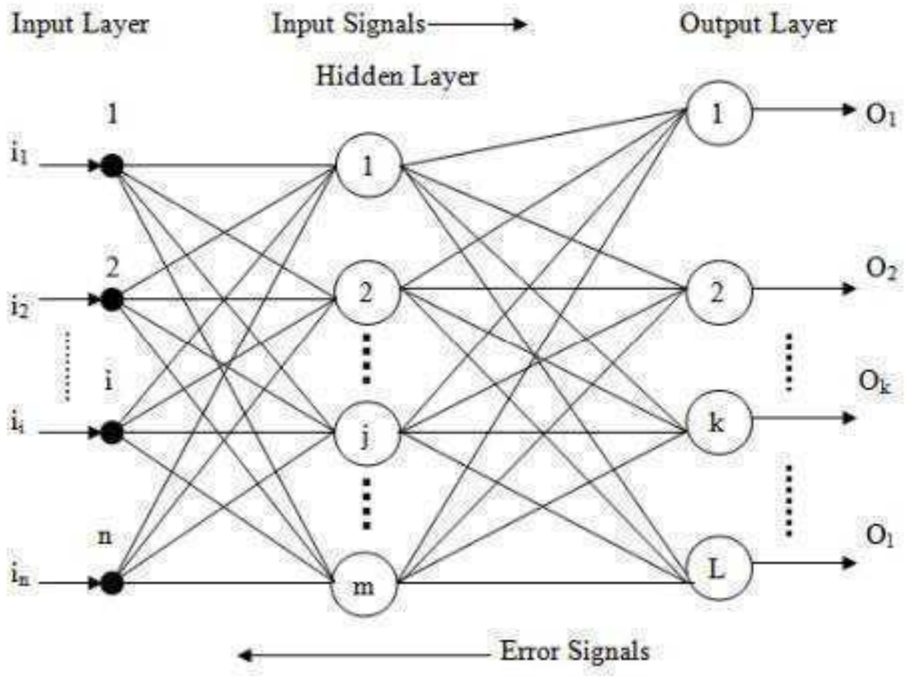

Fig 1.9: Error Back Propagation Network

-Two Passes of Computation

FORWARD PASS: weights fixed, input signals propagated through network and outputs calculated. Outputs o_j are compared with desired outputs d_j; the error signal $e_j = d_j - o_j$ is computed.

BACKWARD PASS: starts with output layer and recursively computes the local gradient δ_j for each node. Then the weights are updated using the equation above for Δw_{ji}, and back to another forward pass.

-Sigmoidal Nonlinearity

With the sigmoidal function $\phi(x)$ defined above, it is the case that $\phi'(v_j) = y_j(1 - y_j)$, a fact that simplifies the computations.

Momentum

- If the learning rate η is very small, then the algorithm proceeds slowly, but accurately follows the path of steepest descent in weight space.
- If η is largish, the algorithm may oscillate ("bounce off the canyon walls").
  A simple method of effectively increasing the rate of learning is to modify the delta rule by including a *momentum* term:
  $\Delta w_{ji}(n) = \alpha \Delta w_{ji}(n-1) + \eta \delta_j(n) y_i(n)$
  where α is a positive constant termed the *momentum constant*. This is called the *generalized delta rule*.
- The effect is that if the basic delta rule is consistently pushing a weight in the same direction, then it gradually gathers "momentum" in that direction.

-Stopping Criterion
Two commonly used stopping criteria are:

- stop after a certain number of runs through all the training data (each run through all the training data is called an *epoch*);
- stop when the total sum-squared error reaches some low level. By total sum-squared error we mean $\Sigma_p \Sigma_i e_i^2$ where *p* ranges over all of the training patterns and *i* ranges over all of the output units.

-Initialization

- The weights of a network to be trained by backprop must be initialized to some non-zero values.
- The usual thing to do is to initialize the weights to small random values.
- The reason for this is that sometimes backprop training runs become "lost" on a plateau in weight-space, or for some other reason backprop cannot find a good minimum error value.
- Using small random values means different starting points for each training run, so that subsequent training runs have a good chance of finding a suitable minimum.

Limitation of EBPA
A back-propagation neural network is only practical in certain situations. Following are some guidelines on when you should use another approach:

- Can you write down a flow chart or a formula that accurately describes the problem? If so, then stick with a traditional programming method.
- Is there a simple piece of hardware or software that already does what you want? If so, then the development time for a NN might not be worth it.
- Do you want the functionality to "evolve" in a direction that is not pre-defined? If so, then consider using a Genetic Algorithm (that's another topic!).
- Do you have an easy way to generate a significant number of input/output examples of the desired behavior? If not, then you won't be able to train your NN to do anything.
- Is the problem is very "discrete"? Can correct answer be found in a look-up table of reasonable size? A look-up table is much simpler and more accurate.
- Is precise numeric output values required? NN's are not good at giving precise numeric answers.

Characteristics of EBPA

There are the characteristics of EBPA:

1. After briefly describing linear threshold units, neural network computation paradigm in general, and the use of the logistic function (or similar functions) to transform weighted sums of inputs to a neuron.
2. Backprop's performance on the XOR problem was demonstrated using the tlearn backprop simulator.
3. A number of refinements to backprop were looked at briefly, including momentum and a technique to obtain the best generalization ability.
4. Backprop nets learn slowly but compute quickly once they have learned.
5. They can be trained so as to generalize reasonably well.

Applications of EBPA

- Backprop tends to work well in some situations where human experts are unable to articulate a rule for what they are doing - e.g. in areas depending on raw perception, and where it is difficult to determine the attributes (in the ID3 sense) that are relevant to the problem at hand.
- For example, there is a proprietary system, which includes a backprop component, for assisting in classifying Pap smears.
    - The system picks out from the image the most suspicious-looking cells.
    - A human expert then inspects these cells.
    - This reduces the problem from looking at maybe 10,000 cells to looking at maybe 100 cells - this reduces the boredom-induced error rate.
- Other successful systems have been built for tasks like reading handwritten postcodes.

Counter propagation network architecture

- An example of a hybrid network which combine the features of two or more basic network designs. Proposed by Hecht-Nielsen in 1986.
- The hidden layer is a Kohonen network with unsupervised learning and the output layer is a Grossberg (outstar) layer fully connected to the hidden layer. The output layer is trained by the Widrow-Hoff rule.
- Allows the output of a pattern rather than a simple category number.
- Can also be viewed as a bidirectional associative memory.
- Figure 2.1 shows a unidirectional counter propagation network used for mapping pattern *A* of size n to pattern *B* of size m.
- The output of the *A* subsection of the input layer is fanned out to the competitive middle layer. Each neuron in the output layer receives a signal corresponding to the input pattern's category along one connection from the middle layer.
- The *B* subsection of the input layer has zero input during actual operation of the network and is used to provide input only during training.

- The role of the output layer is to produce the pattern corresponding to the category output by the middle layer. The output layer uses a supervised learning procedure, with direct connection from the input layer's *B* subsection providing the correct output.
- Training is a two-stage procedure. First, the Kohonen layer is trained on input patterns. No changes are made in the output layer during this step. Once the middle layer is trained to correctly categorise all the input patterns, the weights between the input and middle layers are kept fixed and the output layer is trained to produce correct output patterns by adjusting weights between the middle and output layers.

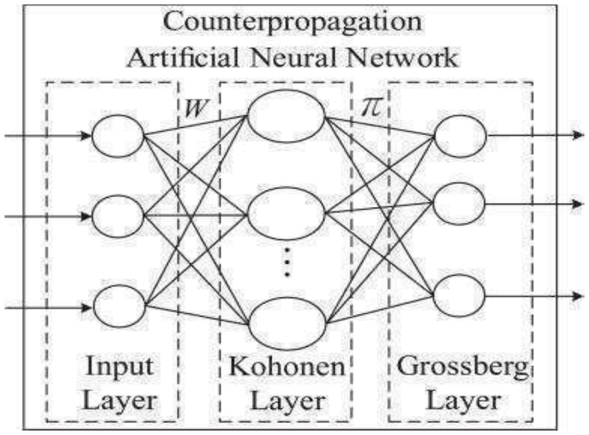

Fig 2.1: Counter Propagation Network

CPN Functioning and Training algorithm

Training algorithm stage 1:

1. Apply normalised input vector **x** to input *A*.
2. Determine winning node in the Kohonen layer.
3. Update winning node's weight vector –

w(t+1) = **w**(t) + (**x** – **w**)

4. Repeat steps 1 through 3 until all vectors have been processed.
5. Repeat steps 1 to 4 until all input vectors have been learned.

Training algorithm stage 2:

1. Apply normalised input vector **x** and its corresponding output vector **y**, to inputs *A* and *B* respectively.
2. Determine winning node in the Kohonen layer.
3. Update weights on the connections from the winning node to the output unit –

wi(t+1) = *wi*(t) + (*yi* – *wi*)

4. Repeat steps 1 through 3 until all vectors of all classes map to satisfactory outputs.

Characteristics of CPN

- In the first training phase, if a hidden-layer unit does not win for a long period of time, its weights should be set to random values to give that unit a chance to win subsequently.
- There is no need for normalizing the training output vectors.

• After the training has finished, the network maps the training vectors onto output vectors that are close to the desired ones.
• The more hidden units, the better the mapping.
• Thanks to the competitive neurons in the hidden layer, the linear neurons can realize nonlinear mappings.

Hopfield/ Recurrent network

The Hopfield network is an implementation of a learning matrix with recurrent links. The learning matrix is a weight matrix which actually stores associations between inputs and targets. This network generalizes in the sense that it identifies general dependencies in the given incomplete and noisy training data, in this sense it resembles a learning matrix. This kind of a network is a linear model as it can model only linearly separable data.

The Hopfield Type Network is a multiple-loop feedback neural computation system. The neurons in this network are connected to all other neurons except to themselves that is there are no self-feedbacks in the network. A connection between two neurons N_i and N_j is two way which is denoted by w_{ij}. The connection w_{ij} from the output of neuron i to the input of neuron j has the same strength as the connection w_{ji} from the output of neuron j to the input of neuron i, in other words the weight matrix is symmetric. Each neuron computes the summation:

$$\mathbf{s_i = \sum_{j=1}^{n} w_{ji}\, x_j}$$

where: n is the number of neurons in the network, w_{ji} are the weights, and x_j are inputs, 1<=j<=n.

The Hopfield network can be made to operate in either continuous or discrete mode. Here it is considered that the network operates in discrete mode using neurons with discrete activation functions. When a neuron fires, its discrete activation function is evaluated and the following output is produced: $x'_i = 1$ if $s_i = \sum_{j=1}^{n} w_{ji}\, x_j >= 0$ or $x'_i = 0$ if $s_i < 0$.

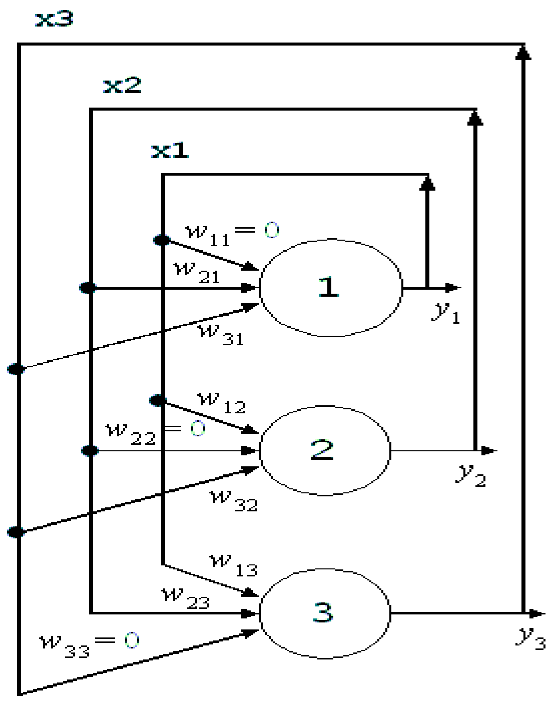

Fig 2.2: Hopfield Network

Configuration:
Hopfield networks can be implemented to operate in two modes:
- Synchronous mode of training Hopfield networks means that all neurons fire at the same time.
- Asynchronous mode of training Hopfield networks means that the neurons fire at random.

Stability constraints:
The recurrent networks of the Hopfield type are complex dynamical systems whose behavior is determined by the connectivity pattern between the neurons. The inputs to the neurons are not simply externally provided inputs but outputs from other neurons that change with the time. The temporal behavior of the network implies characteristics that have to be taken into consideration in order to examine the network performance.
The Hopfield networks are dynamical systems whose state changes with the time. The state of the neural network is the set of the outputs of all neurons at a particular moment, time instant. When a neuron fires then its output changes and so the network state also changes. Therefore the sequence of neuron firings leads to a corresponding sequence of modified neuron outputs, and modified system states. Acquiring knowledge of the state space allows us to study the motion of the neural network in time. The trajectories that the network leaves in the time may be taken to make a state portrait of the system.
A Hopfield net with n neurons has 2^n possible states, assuming that each neuron output produces two values 0 and 1. Performance analysis of the network behavior can be carried out by developing a state table that lists all possible subsequent states.

Associative Memory:

These kinds of neural networks work on the basis of pattern association, which means they can store different patterns and at the time of giving an output they can produce one of the stored patterns by matching them with the given input pattern. These types of memories are also called **Content-Addressable Memory** (CAM). Associative memory makes a parallel search with the stored patterns as data files.
Following are the two types of associative memories we can observe −

- Auto Associative Memory
- Hetero Associative memory

Auto Associative Memory

This is a single layer neural network in which the input training vector and the output target vectors are the same. The weights are determined so that the network stores a set of patterns.
Architecture
As shown in the following figure, the architecture of Auto Associative memory network has 'n' number of input training vectors and similar 'n' number of output target vectors.

Training Algorithm
For training, this network is using the Hebb or Delta learning rule.
Step 1 − Initialize all the weights to zero as $w_{ij} = 0$ (i = 1 to n, j = 1 to n)
Step 2 − Perform steps 3-4 for each input vector.
Step 3 − Activate each input unit as follows −

$$xi=si(i=1 \text{ to } n)$$

Step 4 − Activate each output unit as follows −

$$yj=sj(j=1 \text{ to } n)$$

Step 5 − Adjust the weights as follows −

$$wij(new)=wij(old)+xiyj$$

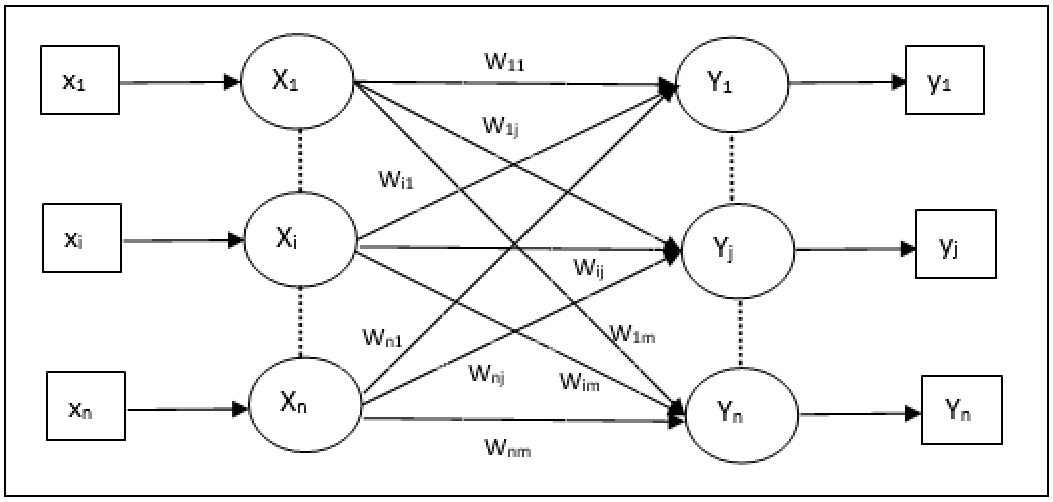

Fig 2.3: Auto Associative Memory

Hetero Associative memory

Similar to Auto Associative Memory network, this is also a single layer neural network. However, in this network the input training vector and the output target vectors are not the same. The weights are determined so that the network stores a set of patterns. Hetero associative network is static in nature, hence, there would be no non-linear and delay operations.

As shown in the following figure, the architecture of Hetero Associative Memory network has 'n' number of input training vectors and 'm' number of output target vectors.

Training Algorithm

For training, this network is using the Hebb or Delta learning rule.

Step 1 − Initialize all the weights to zero as $w_{ij} = 0$ (i = 1 to n, j = 1 to m)

Step 2 − Perform steps 3-4 for each input vector.

Step 3 − Activate each input unit as follows −

$$xi=si(i=1 \text{ to } n)$$

Step 4 − Activate each output unit as follows −

$$yj=sj(j=1 \text{ to } m)$$

Step 5 − Adjust the weights as follows −

$$wij(new)=wij(old)+xiyj$$

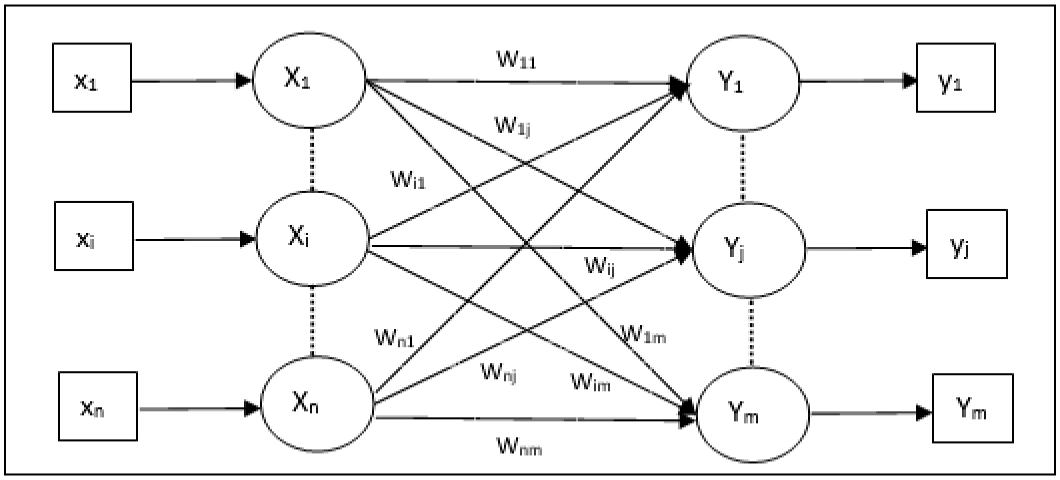

Fig 2.4: Hetero Associative Memory

Characteristics of Associative Memory

Characteristics of an autocorrelation or cross correlation associative memory largely depend on how items are encoded in pattern vectors to be stored. When most of the components of encoded patterns to be stored are 0 and only a small ratio of the components is 1, the encoding scheme is said to be sparse. The memory capacity and information capacity of a sparsely encoded associative memory are analyzed in detail, and are proved to be in proportion of $n^2 logn^2$, *n* being the number of neurons, which is very large compared to the ordinary non-sparse encoding scheme of about 0.15*n*. Moreover, it is proved that the sparsely encoded associative memory has a large basin of attraction around each memorized pattern, when and only when an activity control mechanism is attached to it.

Limitations of Associative Memory

Main limitation of associative memory is to efficiency of access and retrieval of pattern stored in database. In case of damaged pattern, it is unable to restore it.

Applications of Associative Memory

Following are the application areas of associative memory:

1. Patter Recognition i.e. face, signature etc.
2. Content Addressable Storage (CAS)
3. Clustering
4. Encoding and Decoding of Data

Hopfield machine

The standard binary Hopfield network is a recurrently connected network with the following features:

- symmetrical connections: if there is a connection going from unit j to unit i having a connection weight equal to W_ij then there is also a connection going from unit i to unit j with an equal weight.
- linear threshold activation: if the total weighted summed input (dot product of input and weights) to a unit is greater than or equal to zero, its state is set to 1, otherwise it is -1. Normally, the threshold is zero. Note that the Hopfield network for the travelling salesman problem (assignment 3) behaved slightly differently from this.
- asynchronous state updates: units are visited in random order and updated according to the above linear threshold rule.
- Energy function: it can be shown that the above state dynamics minimizes an energy function.
- Hebbian learning

The most important features of the Hopfield network are:

- Energy minimization during state updates guarantees that it will converge to a stable attractor.
- The learning (weight updates) also minimizes energy; therefore, the training patterns will become stable attractors (provided the capacity has not been exceeded).

However, there are some serious drawbacks to Hopfield networks:

- Capacity is only about .15 N, where N is the number of units.
- Local energy minima may occur, and the network may therefore get stuck in very poor (high Energy) states which do not satisfy the "constraints" imposed by the weights very well at all. These local minima are referred to as spurious attractors if they are stable attractors which are not part of the training set. Often, they are blends of two or more training patterns.

Boltzmann machine

The binary Boltzmann machine is very similar to the binary Hopfield network, with the addition of three features:

- Stochastic activation function: the state a unit is in is probabilistically related to its Energy gap. The bigger the energy gap between its current state and the opposite state, the more likely the unit will flip states.
- Temperature and simulated annealing: the probability that a unit is on is computed according to a sigmoid function of its total weighted summed input divided by T. If T is large, the network behaves very randomly. T is gradually reduced and at each value of T, all the units' states are updated. Eventually, at the lowest T, units are behaving less randomly and more like binary threshold units.
- Contrastive Hebbian Learning: A Boltzmann machine is trained in two phases, "clamped" and "unclamped". It can be trained either in supervised or unsupervised mode. Only the supervised mode was discussed in class; this type of training proceeds as follows, for each training pattern:
    1. Clamped Phase: The input units' states are clamped to (set and not permitted to change from) the training pattern, and the output units' states are clamped to the target vector. All other units' states are initialized randomly, and are then permitted to update until they reach "equilibrium" (simulated annealing). Then Hebbian learning is applied.
    2. Unclamped Phase: The input units' states are clamped to the training pattern. All other units' states (both hidden and output) are initialized randomly, and are then permitted to update until they reach "equilibrium". Then anti-Hebbian learning (Hebbian learning with a negative sign) is applied.

  The above two-phase learning rule must be applied for each training pattern, and for a great many iterations through the whole training set. Eventually, the output units' states should become identical in the clamped and unclamped phases, and so the two learning rules exactly cancel one another. Thus, at the point when the network is always producing the correct responses, the learning procedure naturally converges and all weight updates approach zero.

  The stochasticity enables the Boltzmann machine to overcome the problem of getting stuck in local energy minima, while the contrastive Hebb rule allows the network to be trained with hidden features and thus overcomes the capacity limitations of the Hopfield network. However, in practice, learning in the Boltzmann machine is hopelessly slow.

Comparison between Hopfield v/s Boltzman

Hopfield networks have following limitations:

1. suffer from spurious local minima that form on the energy hyper surface
2. require the input patterns to be uncorrelated
3. are limited in capacity of patterns that can be stored
4. are usually fully connected and not stacked

Restricted Boltzmann Machines (RBMs) avoid the spurious solutions that arise in Hopfield Networks by adding in hidden nodes and then sampling over all possible nodes using Boltzman statistics:

This is not the only difference however, as older studies of Hopfield networks also looked at adding temperature (i.e the Little model for spin glasses)

Spin Glass Models of Neural Networks
(* these models have spurious states that form if you are too far below the critical temperature; if you place too many patterns in a Hopfield network, it undergoes a spin glass transition.)
It is known, however, that introducing even simple statistical thermal fluctuations leads to 2p ground stable that are correlated with the input patterns. This behavior is not seen in T=0 Hopfield networks and it appears that introducing the Boltzman stats is critical. Furthermore, this behavior emerges in 2 very different models, the generalized Hopfield model and the little model, so it seems it would also appear in the RBM model also. So adding the hidden variables + temperature has the effect of introducing these kinds of thermal fluctuations, which create a large number of stable memories without low lying meta stable states.

Adaptive Resonance Theory: Architecture

ART1 neural networks cluster binary vectors, using unsupervised learning. The neat thing about adaptive resonance theory is that it gives the user more control over the degree of relative similarity of patterns placed on the same cluster.

An ART1 net achieves stability when it cannot return any patterns to previous clusters (in other words, a pattern oscillating among different clusters at different stages of training indicates an unstable net. Some nets achieve stability by gradually reducing the learning rate as the same set of training patterns is presented many times. However, this does not allow the net to readily learn a new pattern that is presented for the first time after a number of training epochs have already taken place. The ability of a net to respond to (learn) a new pattern equally well at any stage of learning is called plasticity (e.g., this is a computational corollary of the biological model of neural plasticity). Adaptive resonance theory nets are designed to be both stable and plastic.

The basic structure of an ART1 neural network involves:

- an input processing field (called the F_1 layer) which happens to consist of two parts:
    - an input portion (F_1(a))
    - an interface portion (F_1(b))
- the cluster units (the F_2 layer)
- and a mechanism to control the degree of similarity of patterns placed on the same cluster
- a reset mechanism
- weighted bottom-up connections between the F_1 and F_2 layers
- weighted top-down connections between the F_2 and F_1 layers

F_1(b), the interface portion, combines signals from the input portion and the F_2 layer, for use in comparing the similarity of the input signal to the weight vector for the cluster unit that has been selected as a candidate for learning.

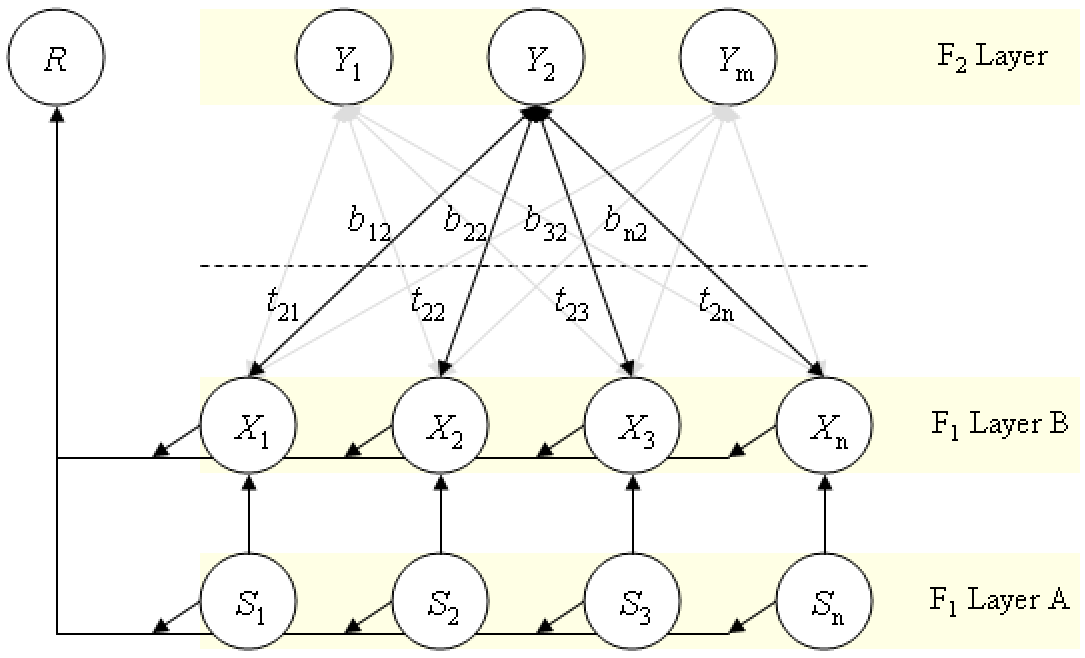

Fig 2.5: ART Architecture

To control the similarity of patterns placed on the same cluster, there are two sets of connections (each with its own weights) between each unit in the interface portion of the input field and the cluster unit. The F_1(b) layer is connected to the F_2 layer by bottom-up weights (b_{ij}). The F_2 layer is connected to the F_1(b) layer by top-down weights (t_{ij}).
The F_2 layer is a competitive layer: The cluster unit with the largest net input becomes the candidate to learn the input pattern. The activations of all other F_2 units are set to zero. The interface units, F_1(b), now combine information from the input and cluster units. Whether or not this cluster unit is allowed to learn the input pattern depends on how similar its top-down weight vector is to the input vector. This decision is made by the reset unit, based on signals it receives from the input F_1(a) and interface F_1(b) layers. If the cluster unit is not allowed to learn, it is inhibited and a new cluster unit is selected as the candidate. If a cluster unit is allowed to learn, it is said to classify a pattern class. Sometimes there is a tie for the winning neuron in the F_2 layer, when this happens, then an arbitrary rule, such as the first of them in a serial order, can be taken as the winner.
During the operation of an ART1 net, patterns emerge in the F_1a and F_1b layers and are called traces of STM (short–term memory. Traces of LTM (long–term memory) are in the connection weights between the input layers (F_1) and output layer (F_2).

Classifications of ART

- **ART-1**

This is a binary version of ART, i.e., it can cluster binary input vectors.

- **ART-2**

This is an analogue version of ART, i.e. it can cluster real-valued input vectors.

- **ART-2A**

This refers to a fast version of the ART2 learning algorithm.

- **ART-3**

This network is an ART extension that incorporates "chemical transmitters" to control the search process in a hierarchical ART structure.

- **ARTMAP**

This is a supervised version of ART that can learn arbitrary mappings of binary patterns.

- **Fuzzy ART**

This network is a synthesis of ART and fuzzy logic.

- **Fuzzy ARTMAP**

This is supervised fuzzy ART.

- **Distributed ART and ARTMAP (dART and dARTMAP)**

These models learn distributed code representations in the F2 layer. In the special case of winner-take-all F2 layers, they are equivalent to Fuzzy ART and ARTMAP, respectively.

- **ART adaptations**

We are aware that this list is far from complete. Contributions in the form of a short description together with references are much appreciated!

- **ARTMAP-IC**

This network adds distributed prediction and category instance counting to the basic fuzzy ARTMAP system.

- **Gaussian ARTMAP**

A supervised-learning ART network that uses Gaussian-defined receptive fields.

- **Hierarchical (modular) ART models**

These are ART-based modular networks that learn hierarchical clusterings of arbitrary sequences of input patterns.

- **arboART**

In this network, the prototype vectors at each layer are used as input to the following layer (agglomerative method). The architecture is similar to HART-J (see below). It has been applied to automatic rule generation of Kansei engineering expert systems.

- **Cascade Fuzzy ART**

A cascade of Fuzzy ART networks that develop hierarchies of analogue and binary patterns through bottom-up learning guided by a top-down search process. It has been applied to model-based 3D object recognition.

- **HART(-J), HART-S**

Modular Hierarchical ART (HART) models. HART-J (also known as HART) implements an agglomerative clustering method (similar to arboART above). HART-S implements a divisive clustering method with each ART layer learning the differences between the input and the matching prototype of the previous layer.

- **SMART**

Self-consistent Modular ART network, which is capable of learning self-consistent cluster hierarchies through explicit links and an internal feedback mechanism (much like those of the ARTMAP network).

- **LAPART**

An ART-based neural architecture for pattern sequence verification through inferencing.

- **MART**

Multichannel ART, for adaptive classification of patterns through multiple inputs channels.

- **PROBART**

A modification to the Fuzzy ARTMAP network (by building up probabilistic information regarding interlayer node associations) which allows it to learn to approximate noisy mappings.

- **R2MAP**

An ARTMAP-based architecture capable of learning complex classification tasks by re-iteratively creating novel (relational) input features to represent the same problem with fewer input categories. It takes motivation from the representational redescription (RR) hypothesis in cognitive science.

- **TD-ART**

Time-Delay ART for learning spatio-temporal patterns.

Implementation and Training

Step 1 − Initialize the learning rate, the vigilance parameter, and the weights as follows −

$$\alpha > 1 \text{ and } 0 < \rho \leq 1$$

$$0 < b_{ij}(0) < \alpha / (\alpha - 1 + n) \text{ and } t_{ij}(0) = 1$$

Step 2 − Continue step 3-9, when the stopping condition is not true.

Step 3 − Continue step 4-6 for every training input.

Step 4 − Set activations of all F_1(a) and F_1 units as follows

$$\mathbf{F_2 = 0 \text{ and } F_1(a) = \text{input vectors}}$$

Step 5 − Input signal from F_1(a) to F_1(b) layer must be sent like

$$si=xi$$

Step 6 − For every inhibited F_2 node

$y_j=\sum_i b_{ij} x_i$, the condition is $y_j \neq -1$

Step 7 − Perform step 8-10, when the reset is true.

Step 8 − Find **J** for $y_J \geq y_j$ for all nodes **j**

Step 9 − Again calculate the activation on F_1(b) as follows

$$xi=sitJi$$

Step 10 − Now, after calculating the norm of vector **x** and vector **s**, we need to check the reset condition as follows −

If $||x||/||s||$ < vigilance parameter **ρ**, then inhibit node **J** and go to step 7

Else If $||x||/||s|| \geq$ vigilance parameter **ρ**, then proceed further.

Step 11 − Weight updating for node **J** can be done as follows −

$$bij(new)=(\alpha xi)/(\alpha-1+||x||)$$

$$tij(new)=xi$$

Step 12 − The stopping condition for algorithm must be checked and it may be as follows −

- Do not have any change in weight.
- Reset is not performed for units.
- Maximum number of epochs reached.