Unsupervised Learning

Introduction -

In this learning, there exists no feedback from the system (environment) *w* indicate the desired outputs of a network. The network by itself should discover any relationships of interest, such as features, patterns, contours, correlations or categories, classification in the input data, and thereby translate the discovered relationships into outputs. Such networks are also called self-organizing networks. An unsupervised learning can judge how similar a new input pattern is to typical patterns already seen, and the network gradually learns what similarity is; the network may construct a set of axes along which to measure similarity to previous patterns, i.e., it performs principal component analysis, clustering, adaptive vector quantization and feature mapping.

For example, when net has been trained to classify the input patterns into any one of the output classes, say, P, Q, R, S or T, the net may respond to both the classes, P and Q or R and S. In the case mentioned, only one of several neurons should fire, i.e., respond. Hence the network has an added structure by means of which the net is forced to make a decision, so that only one unit will respond. The process for achieving this is called competition. Practically, considering a set of students, if we want to classify them on the basis of evaluation performance, their score may be calculated, and the one whose score is higher than the others should be the winner. The same principle adopted here is followed in the neural networks for pattern classification. In this case, there may exist a tie; a suitable solution is presented even when a tie occurs. Hence these nets may also be called competitive nets, the extreme form of these competitive nets is called winner-take-all. The name itself implies that only one neuron in the competing group will possess a nonzero output signal at the end of competition.

There exist several neural networks that come under this category. To list out a few: Maxnet, Mexican hat, Hamming net, Kohonen self-organizing feature map, counter propagation net, learning vector quantization (LVQ) and adaptive resonance theory (ART).

The learning algorithm used ·m most of these nets is known as Kohonen learning. In this learning, the units update their weights by forming a new weight vector, which is a linear combination of the old weight vector and the new input vector. Also, the learning continues for the unit whose weight vector is closest to the input vector. The weight updation formula used in Kohonen learning for output cluster unit *j* is given as

$$w_j(\text{new}) = w_{\Theta j}(\text{old}) + \alpha \left[x - w_{\Theta j}(\text{old})\right]$$

where x is the input vector; $w_{\Theta j}$ the weight vector for unit j; α the learning rate whose value decreases monotonically as training continues. There exist two methods to determine the winner of the network during competition. One of the methods for determining the winner uses the square of the Euclidean distance between the input vector and weight vector, and the unit whose weight vector is at the smallest Euclidean distance from the input vector is chosen as the winner. The next method uses the dot product of the input vector and weight vector. The dot product between the input vector and weight vector is nothing but the net inputs calculated for the corresponding duster units. The unit with the largest dot product *is* chosen as the winner and the weight updation is performed over it because the one with largest dot product corresponds to the smallest angle between the input and weight vectors, if both are of unit length.

Fixed Weight Competitive Nets

These competitive nets arc those where the weights remain fixed, even during training process. The idea of competition is used among neurons for enhancement of contrast in their activation functions. These are Maxnet, Mexican hat and Hamming net.

Maxnet

The Maxnet serves as a sub net for picking the node whose input is larger.

Architecture of Maxnet

The architecture of Maxnet is shown in Figure 5·1, where fixed symmetrical weights are present over the weighted interconnections. The weights between the neurons are inhibitory and fixed. The Maxnet with this structure can be used as a subnet to select a particular node whose net input is the largest.

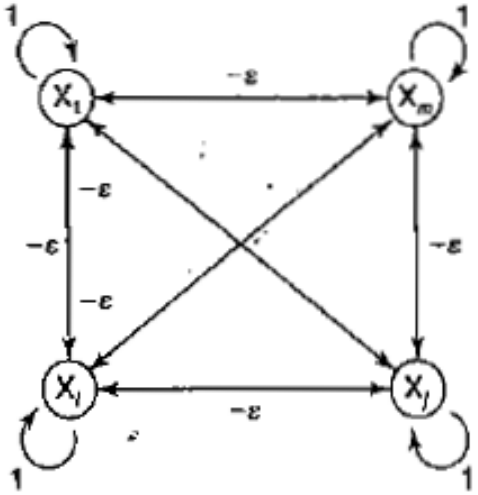


Figure 5-1 Maxnet structure.

Testing/Application Algorithm of Maxnet:

Step 0: Initial weights and initial activations are set. The weight is set as $[0 < \varepsilon < 1/m]$, where "m" is the total number of nodes. Let

$$x_j(0) = \text{input to the node } X_j$$

and

$$w_{ij} = \begin{cases} 1 & \text{if } i = j \\ -\varepsilon & \text{if } i \neq j \end{cases}$$

Step 1: Perform Steps 2–4, when stopping condition is false.

Step 2: Update the activations of each node. For $j = 1$ to m,

$$x_j(\text{new}) = f\left[x_j(\text{old}) - \varepsilon \sum_{i \neq j} x_k(\text{old})\right]$$

Step 3: Save the activations obtained for use in the next iteration. For $j = 1$ to m,

$$x_j(\text{old}) = x_j(\text{new})$$

Step 4: Finally, test the stopping condition for convergence of the network. The following is the stopping condition: If more than one node has a nonzero activation, continue; else stop.

Mexican Hat Net

In 1989, Kohonen developed the Mexican hat network which is a more generalized contrast enhancement network compared to the earlier Maxnet. There exist several "cooperative neighbors" (neurons in close proximity) to which every other neuron is connected by excitatory links. Also each neuron is connected over inhibitory weights to a number of "competitive neighbors" (neurons present farther away). There are several oilier fanher neurons to which the connections between the neurons are nor established. Here, in addition to the connections within a particular layer of neural net, the neurons also receive some other external signals.

This interconnection pattern is repeated for several other neurons in the layer.

Architecture of Mexican Hat Net

The architecture of Mexican hat is shown in Figure 5·2, with the interconnection pattern for node Xi. The neurons here are arranged in linear order; having positive connections between Xi and near neighboring units, and negative connections between Xi and farther away neighboring units. The positive connection region is called region of cooperation and the negative connection region is called region of competition. The size of these regions depends on the relative magnitudes existing between the positive and negative weights and also on the topology of regions such as linear, rectangular, hexagonal grids, ere. In Mexican Hat, there exist two symmetric regions around each individual neuron.

The individual neuron in Figure 5-2 is denoted by Xi. This neuron is surrounded by other neurons $Xi+_1$, $Xi-_1$, $Xi+_2$, $Xi-_2$, ... The nearest neighbors to the individual neuron Xi are $Xi+_1$, $Xi-_1$. $Xi+_2$• and Xi-2· Hence, the weights associated with these are considered to be positive and are denoted by WI and w2. The farthest neighbors m the individual neuron Xi are taken as Xi+3 and Xi-3, the weights associated with these are negative and are denoted by w3. It can be seen chat $Xi+_4$ and $Xi-_4$ are not connected to the individual neuron Xi, and therefore no weighted interconnections exist between these connections. To make it easier, the units present within a radius of 2 [query for unit] to the unit Xi are connected with positive weights, the units within radius 3 are connected with negative weights and the units present further away from radius 3 are not connected in any manner co the neuron Xi.

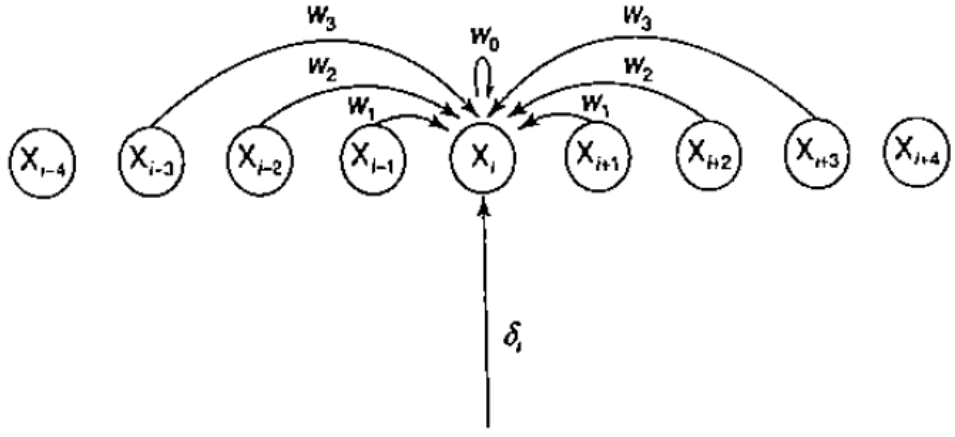


Figure 5-2 Srructure of Mexican hat.

Flowchart of Mexican Hat Net

The flowchart for Mexican hat is shown in Figure 5-3. This dearly depicts the flow of the process performed in Mexican Hat Network.

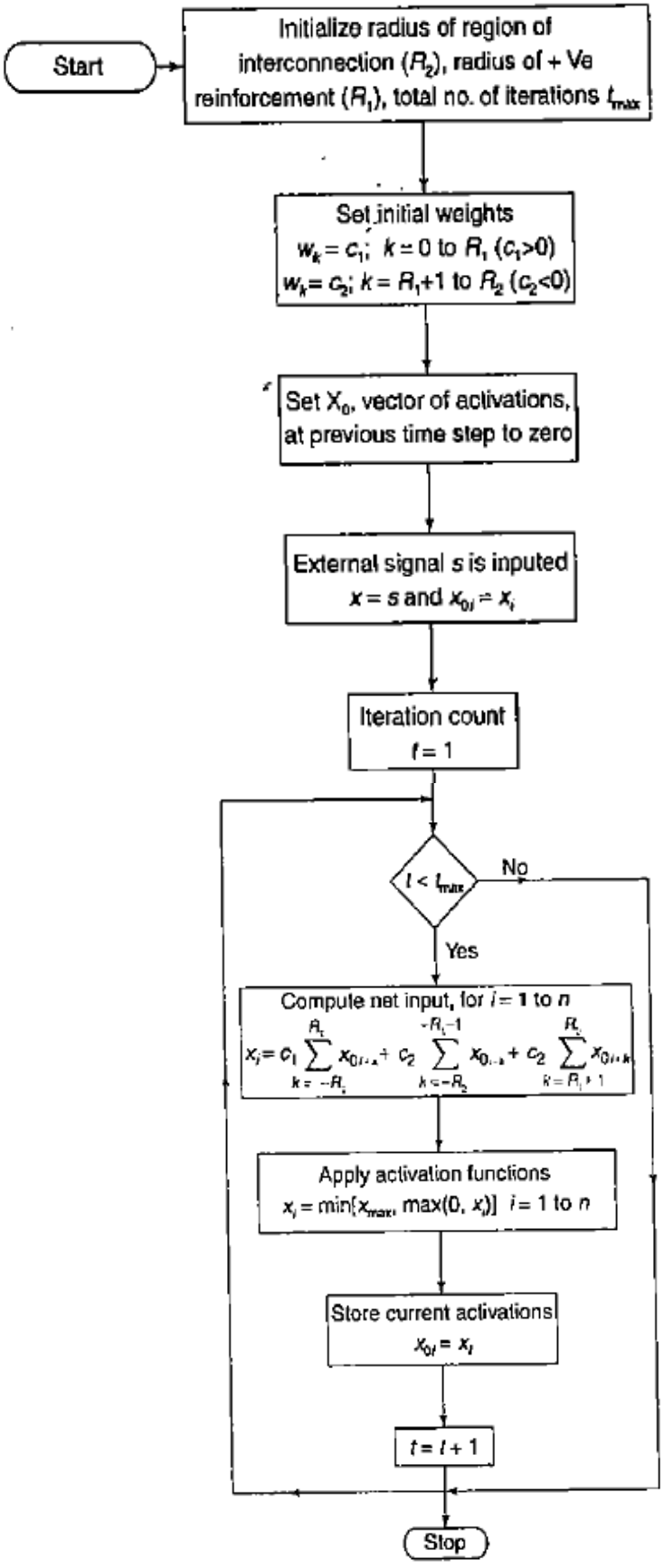


Figure 5-3 Flowchart of Mexican hat.

Algorithm of Mexican Hat Net:

The various parameters used in the training algorithm are as shown below.

R_2 = radius of regions of interconnections

X_{i+k} and X_{i-k} are connected to the individual units X_i for $k = 1$ to R_2.

R_1 = radius of region with positive reinforcement ($R_1 < R_2$)

W_k = weight between X_i and the units X_{i+k} and X_{i-k}

$$0 \leqslant k \leqslant R_1, \quad w_k = \text{positive}$$
$$R_1 \leqslant k \leqslant R_2, \quad w_k = \text{negative}$$

s = external input signal

x = vector of activation

x_0 = vector of activations at previous time step

t_{max} = total number of iterations of contrast enhancement.

Here the iteration is started only with the incoming of the external signal presented to the network.

Step 0: The parameters R_1, R_2, t_{max} are initialized accordingly. Initialize weights as

$$w_k = c_1 \quad \text{for } k = 0, \ldots, R_1 \quad (\text{where } c_1 > 0)$$
$$w_k = c_2 \quad \text{for } k = R_1 + 1, \ldots, R_2 \quad (\text{where } c_2 < 0)$$

Initialize $x_0 = 0$.

Step 1: Input the external signal s:

$$x = s$$

The activations occurring are saved in array x_0. For $i = 1$ to n,

$$x_{0i} = x_i$$

Once activations are stored, set iteration counter $t = 1$.

Step 2: When t is less than t_{max}, perform Steps 3–7.

Step 3: Calculate net input. For $i = 1$ to n,

$$x_i = c_1 \sum_{k=-R_1}^{R_1} x_{0_{i+k}} + c_2 \sum_{k=-R_2}^{-R_1-1} x_{0_{i+k}} + c_2 \sum_{k=R_1+1}^{R_2} x_{0_{i+k}}$$

Step 4: Apply the activation function. For $i = 1$ to n,

$$x_i = \min[x_{max}, \max(0, x_i)]$$

Step 5: Save the current activations in x_0, i.e., for $i = 1$ to n,

$$x_{0i} = x_i$$

Step 6: Increment the iteration counter:

$$t = t + 1$$

Step 7: Test for stopping condition. The following is the stopping condition:
If $t < t_{max}$, then continue
Else stop

The positive reinforcement here has the capacity to increase the activation of units with larger initial activations and the negative reinforcement has the capacity to reduce the activation of units with smaller initial activations. The activation function used here for unit X_i at a particular time instant "t" is given by

$$x_i(t) = f\left[s_i(t) + \sum_k w_k x_{i+k} + k(t-1)\right]$$

The terms present within the summation symbol are the weighted signals that arrived from other units at the previous time step.

Hamming Network

The Hamming network selects stored classes, which are at a maximum Hamming distance (H) from the noisy vector presented at the input (Lippmann, 1987). The vectors involved in this case are all binary and bipolar. Hamming network is a maximum likelihood classifier that determines which of several exemplar vectors (the weight vector for an output unit in a clustering net is exemplar vector or code book vector for the pattern of inputs, which the net has placed on that duster unit) is most similar to an input vector (represented *as* an n-tuple). The weights of the net are determined by the exemplar vectors. The difference between the tom! number of components and the Hamming distance between the vectors gives the measure of similarity between the input vector and stored exemplar vectors. It is already discussed the Hamming distance between the two vectors is the number of components in which the vectors differ.

Consider two bipolar vectors *x* and y; we use a relation

$$x \cdot y = a - d$$

where *a* is the number of components in which the vectors agree, *d* the number of components in which the vectors disagree. The value *"a - d"* is the Hamming distance existing between two vectors. Since, the total number of components is n, we have,

$$n = a + d$$

$$\text{i.e.,} \quad d = n - a$$

On simplification, we get

$$x \cdot y = a - d$$

$$x \cdot y = a - (n - a)$$

$$x \cdot y = 2a - n$$

$$2a = x \cdot y + n$$

$$a = \frac{1}{2}(x \cdot y) + \frac{1}{2}(n)$$

From the above equation, it is clearly understood that the weights can be set to one-half the exemplar vector and bias can be set initially to *n/2*. By calculating the unit with the largest net input, the net is able to locate a particular unit that is closest to the exemplar. The unit with the largest net input is obtained by the Hamming net using Maxnet as its subnet.

Architecture of Hamming Network:

The architecture of Hamming network is shown in Figure 5-4. The Hamming network consists of two layers. The first layer computes the difference between the total number of components and Hamming distance between the input vector *x* and the stored pattern of vectors in the feed-forward path. The efficient response in this layer of a neuron is the indication of the minimum Hamming distance value between the input and the category, which this neuron represents. The second layer of the Hamming network is composed of Maxnet (used as a subnet) or a Winner-take-all network which is a recurrent network The Maxnet is found to suppress the values at Maxnet output nodes except the initially maximum output node of the first layer.

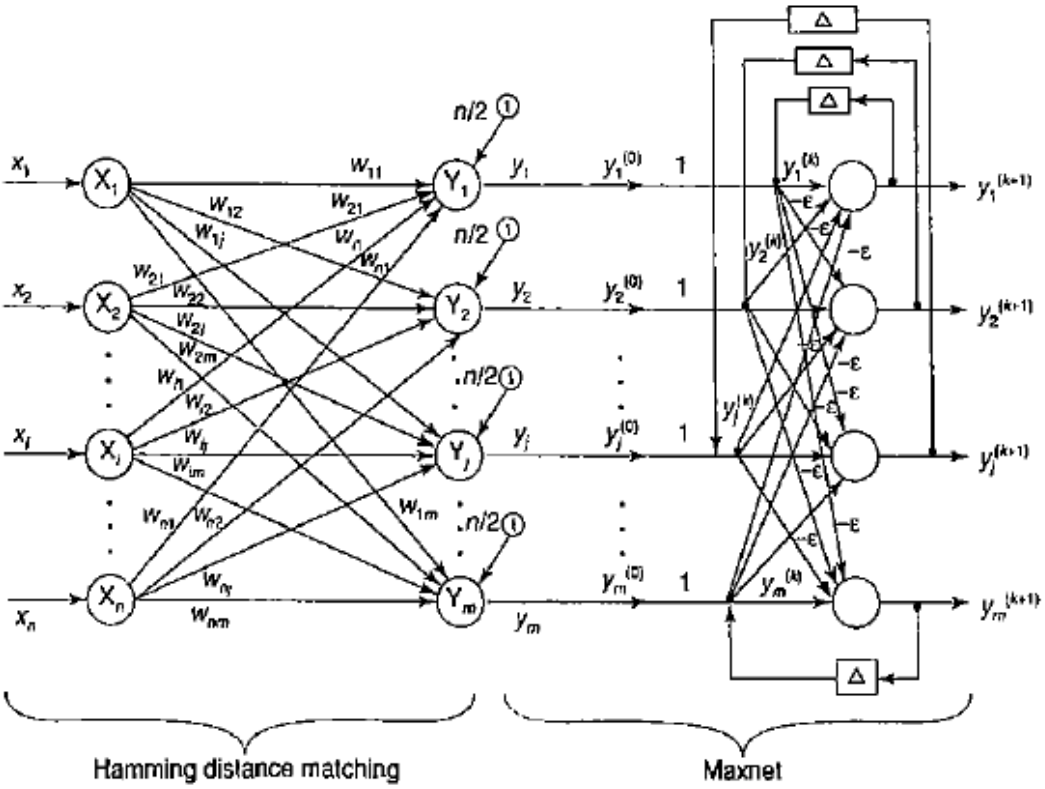


Figure 5-4 Structure of Hamming network.

Testing Algorithm of Hamming Network:

The given bipolar input vector is *x* and for a given set of *"m"* bipolar exemplar vectors say e(l),. *e(j), ... , e(m),* the Hamming network is used to determine the exemplar vector that is closest m the input vector *x.* The net input entering unit Yj gives the measure of the similarity between the input vector and exemplar vector. The parameters used here are the following:

n = number of input units (number of components of *input-output* vector)
m= number of output units (number of components of exemplar vector)
e(j)= jth exemplar vector, i.e.,

$$e(j) = [e1\ (j), \ldots , e_j(j), \ldots , e_n(j)]$$

The testing algorithm for the Hamming Net is as follows:

Step 0: Initialize the weights. For $i = 1$ to n and $j = 1$ to m,

$$w_{ij} = \frac{e_i(j)}{2}$$

Initialize the bias for storing the "m" exemplar vectors. For $j = 1$ to m,

$$b_j = \frac{n}{2}$$

Step 1: Perform Steps 2–4 for each input vector x.

Step 2: Calculate the net input to each unit Y_j, i.e.,

$$y_{inj} = b_j + \sum_{i=1}^{n} x_i w_{ij}, \quad j = 1 \text{ to } m$$

Step 3: Initialize the activations for Maxnet, i.e.,

$$y_j(0) = y_{inj}, \quad j = 1 \text{ to } m$$

Step 4: Maxnet is found to iterate for finding the exemplar that best matches the input patterns.

Kohonen Self-Organizing Feature Maps

Features mapping is a process which converts the patterns of arbitrary dimensionality into a response of one- or two-dimensional arrays of neurons, i.e. it converts a wide pattern space into a typical feature space. The network performing such a mapping is called feature map. Apart from its capability to reduce the higher dimensionality, it has to preserve the neighborhood relations of the input patterns, i.e. it has to obtain a topology preserving map. For obtaining such feature maps, it is required to find a self-organizing array which consist of neurons arranged in a one-dimensional array or a two-dimensional array. To depict this, a typical network structure where each component of the input vector x is connected to each of nodes is shown in Figure 5-5.

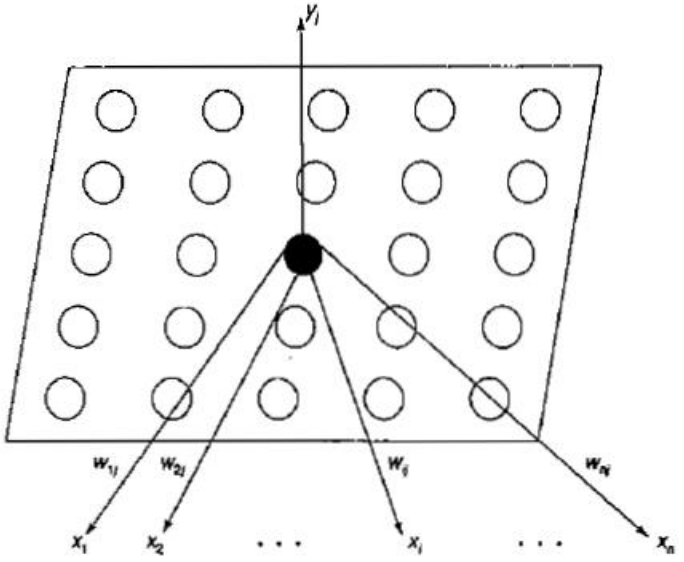


Figure 5-5 One-dimensional feature mapping network.

On the other hand, if the input vector is two-dimensional, the inputs, say *x(a,* b), can arrange themselves in a two-dimensional array defining the input space *(a,* b) as in Figure 5-6. Here, the two layers are fully connected.

The topological preserving property is observed in the brain, bur nor found in any other artificial neural network.

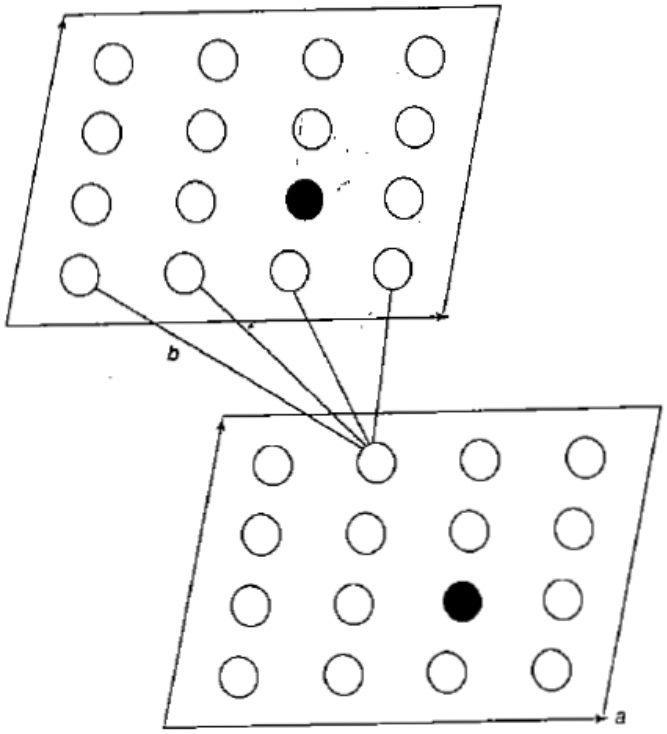


Figure 5-6 Two-dimensional feature mapping network.

Architecture of Kohonen Self-Organizing Feature Maps

Consider a linear array of cluster units as in Figure 5-7. The neighborhoods of the units designated by "o" of radii Ni(k1), Ni(k2) and Ni(k,), *k1* > *k,* > *k,,* where *k1* = 2, *k2* = 1, *k3* = 0.

For a rectangular grid, a neighborhood (Ni) of radii k1, k2, and k3 is shown in Figure 5-8 and for a

hexagonal grid the neighborhood is shown in Figure 5-9. In all the three cases (Figures 5-7-5-9), the unit with "#" symbol is the winning unit and the other units are indicated by "o." In both rectangular and hexagonal grids, *k1* >k2 > *k3,* where k1 = 2, *k2* = 1, *k3* = 0.

For rectangular grid, each unit has eight nearest neighbors but there are only six neighbors for each unit in

the case of a hexagon grid. Missing neighborhoods may just be ignored. A typical architecture of Kohonen self-organizing feature map (KSOFM) is shown in Figure 5-10.

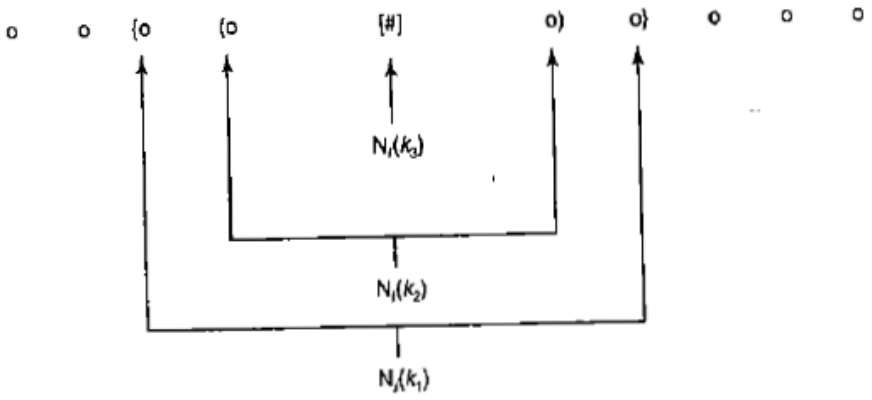


Figure 5-7 Linear array of cluster units.

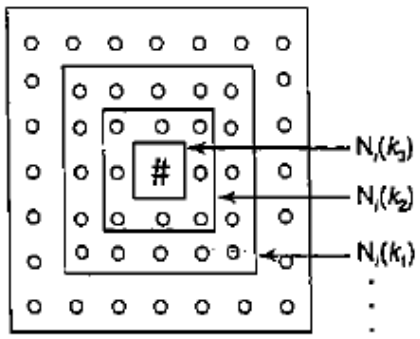


Figure 5-8 Rectangular grid.

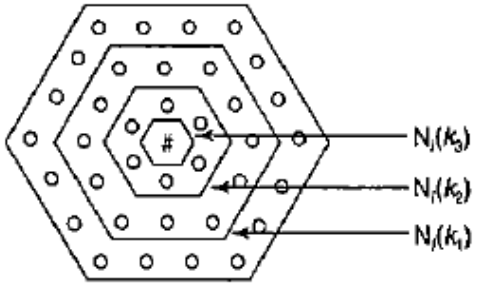


Figure 5-9 Hexagonal grid.

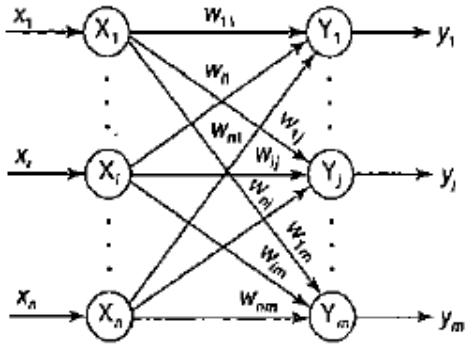


Figure 5-10 Kohonen self-organizing feature map architecture.

Flowchart of Kohonen Self-Organizing Feature Maps

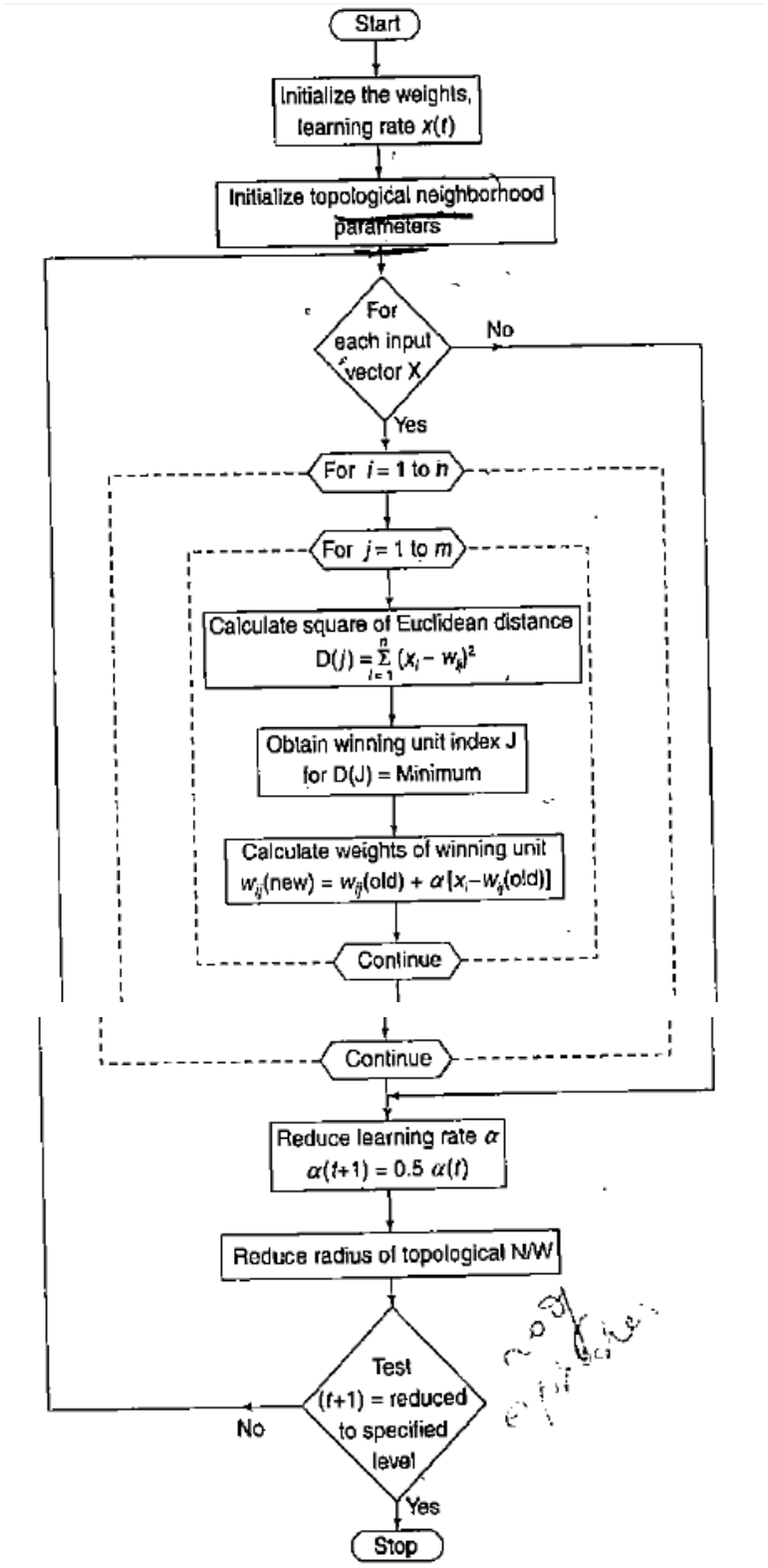


Figure 5-11 Flowchart for training process of KSOFM.

Training Algorithm of Kohonen Self-Organizing Feature Maps:

Step 0: • Initialize the weights w_{ij}: Random values may be assumed. They can be chosen as the same range of values as the components of the input vector. If information related to distribution of clusters is known, the initial weights can be taken to reflect that prior knowledge.

• Set topological neighborhood parameters: As clustering progresses, the radius of the neighborhood decreases.

• Initialize the learning rate α: It should be a slowly decreasing function of time.

Step 1: Perform Steps 2–8 when stopping condition is false.

Step 2: Perform Steps 3–5 for each input vector x.

Step 3: Compute the square of the Euclidean distance, i.e., for each $j = 1$ to m,

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

Step 4: Find the winning unit index J, so that D(J) is minimum. (In Steps 3 and 4, dot product method can also be used to find the winner, which is basically the calculation of net input, and the winner will be the one with the largest dot product.)

Step 5: For all units j within a specific neighborhood of J and for all i, calculate the new weights:

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

or

$$w_{ij}(\text{new}) = (1 - \alpha) w_{ij}(\text{old}) + \alpha x_i$$

Step 6: Update the learning rate α using the formula $\alpha(t + 1) = 0.5\alpha(t)$.

Step 7: Reduce radius of topological neighborhood at specified time intervals.

Step 8: Test for stopping condition of the network.

Kohonen Self-Organizing Motor Map :

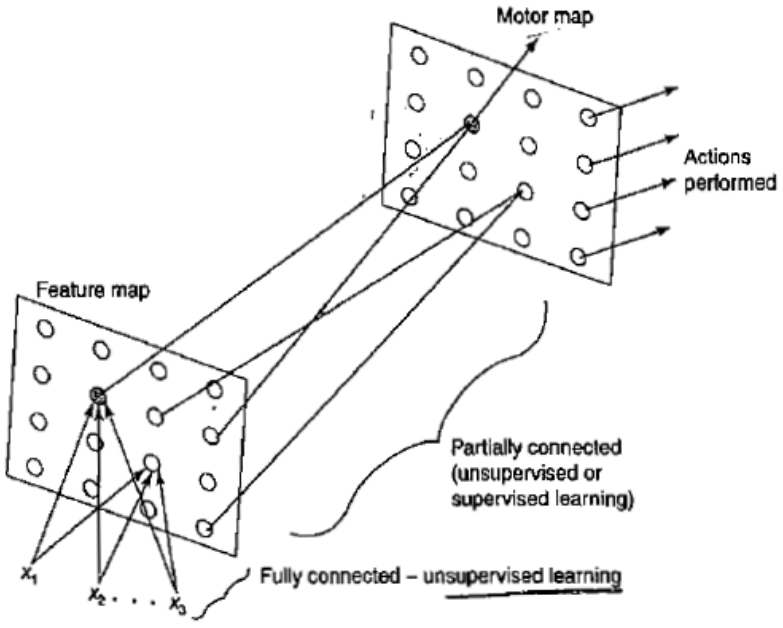


Figure 5-12 Architecture of Kohonen self-organizing motor map.

The extension of Kohonen feature map for a multilayer network involve the addition of an association layer to the output of the self-organizing feature map layer. The output node is found to associate the desired output values with certain input vectors. This type of architecture is called as Kohonen self-organizing motor map and layer that is added is called a motor map in which the movement command, are being mapped into two-dimensional locations of excitation. The architecture of KSOMM is shown in Figure 5-12. Here, the feature map is a hidden layer and this acts as a competitive network which classifies the input vectors.

Learning Vector Quantization (LVQ)

LVQ is a process of classifying the patterns, wherein each output unit represents a particular class. Here, for each class several units should be used. The output unit weight vector is called the reference vector or code book vector for the class which the unit represents. This is a special case of competitive net, which uses supervised learning methodology. During the training the output units are found to be positioned to approximate the decision surfaces of the existing Bayesian classifier. Here, the set of training patterns with known classifications is given to the network, along with an initial distribution of the reference vectors. When the training process is complete, an LVQ net is found to classify an input vector by assigning it to the same class as that of the output unit, which has its weight vector *very close to the* input vector. Thus LVQ is a classifier paradigm that adjusts the boundaries between categories to minimize existing misclassification. LVQ is used for optical character recognition, converting speech mro phonemes and other application as well.

Architecture of LVQ:

Figure 5-13 shows the architecture of LVQ. From Figure 5-13 it can be noticed that there exists input layer with *"n"* unit; and output layer with *"m"* units. The layers are found to be fully interconnected with weighted linkage acting over the links.

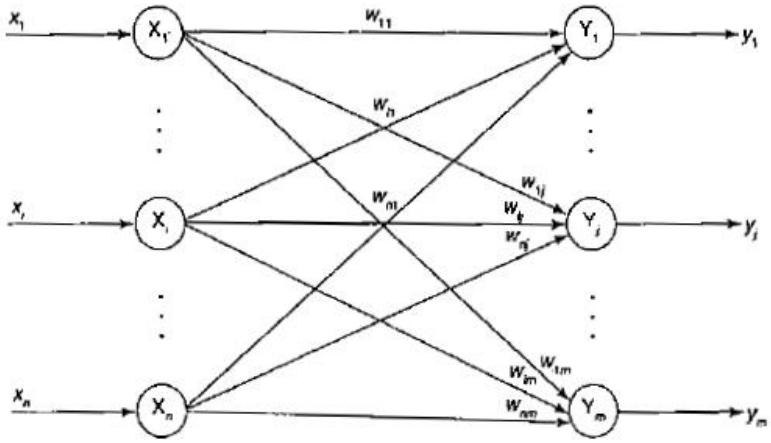


Figure 5-13 Architecture of LVQ.

Flowchart of LVQ:

The parameters used for the training process of a LVQ include the following:

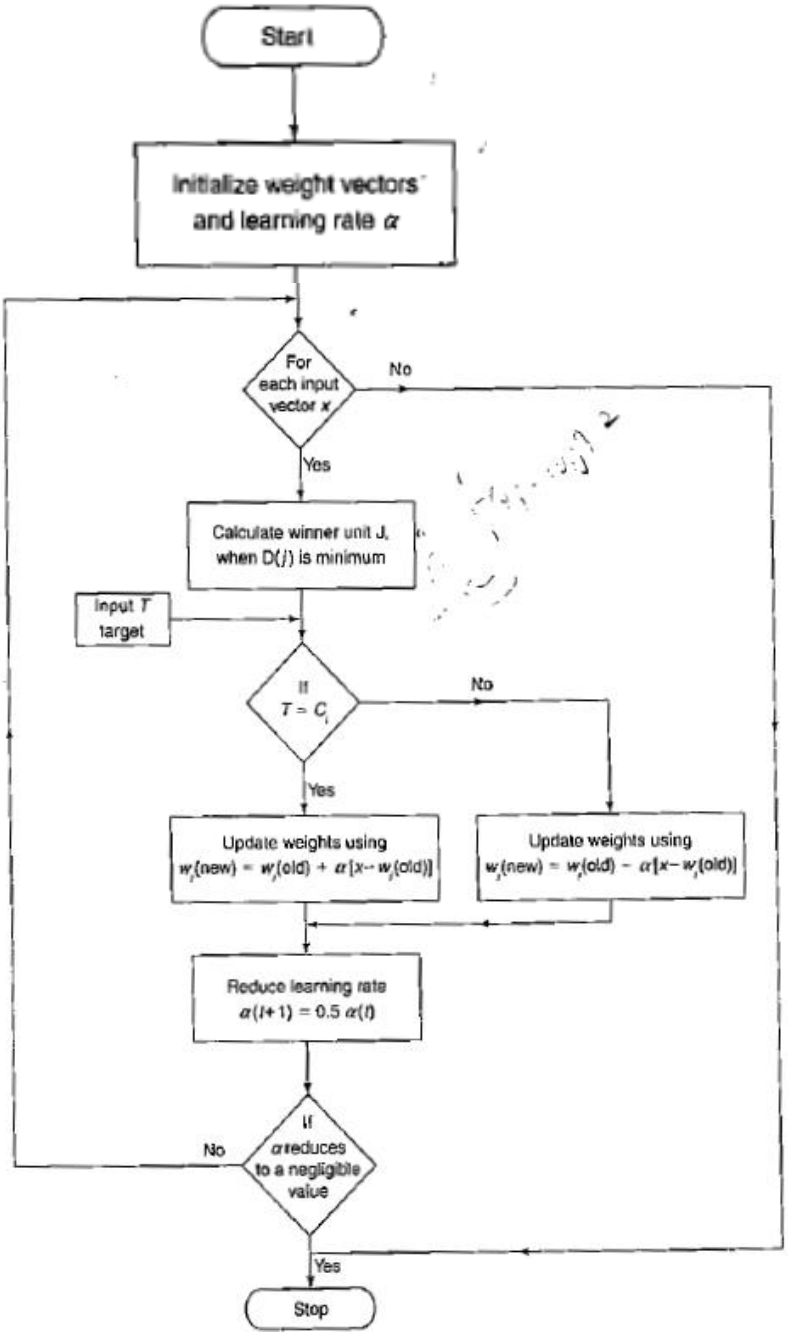
x = training vector $(x_1, \ldots, x_i, \ldots, x_n)$

T = category or class for the training vector x

w_j = weight vector for jth output unit $(w_{1j}, \ldots, w_{ij}, \ldots, w_{nj})$

c_j = cluster or class or category associated with jth output unit.

The Euclidean distance of jth output unit is $D(j) = \sum (x_i - w_{ij})^2$. The flowchart indicating the flow of training process is shown in Figure 5-14.



Training Algorithm of LVQ:

Step 0: Initialize the reference vectors. This can be done using the following steps.

- From the given set of training vectors, take the first "m" (number of clusters) training vectors and use them as weight vectors, the remaining vectors can be used for training.
- Assign the initial weights and classificarions randomly.
- K-means clustering method.

Set initial learning rate α.

Step 1: Perform Steps 2–6 if the stopping condition is false.

Step 2: Perform Steps 3–4 for each training input vector x.

Step 3: Calculate the Euclidean distance; for $i = 1$ to n, $j = 1$ to m,

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

Find the winning unit index J, when D(J) is minimum.

Step 4: Update the weights on the winning unit, w_J using the following conditions.

If $T = q$, then $w_J(\text{new}) = w_J(\text{old}) + \alpha [x - w_J(\text{old})]$

If $T \neq q$, then $w_J(\text{new}) = w_J(\text{old}) - \alpha [x - w_J(\text{old})]$

Step 5: Reduce the learning rate α.

Step 6: Test for the stopping condition of the training process. (The stopping conditions may be fixed number of epochs or if learning rate has reduced to a negligible value.)

Counter propagation Networks

They are multilayer networks based on the combinations of the input, output and clustering layers. The applications of counter propagation nets are data compression, function approximation and pattern association. The counter propagation network is basically constructed from an instar-outstar model. This model is a three-layer neural network that performs input-output data mapping, producing an output vector yin response to an input vector *x,* on the basis of competitive learning. The three layers in an instar-outstar model are the input layer, the hidden (competitive) layer and the output layer. The connections between the input layer and the competitive layer are the instar structure, and the connections existing between the competitive layer and the output layer are the outstar structure.

There are two stages involved in the training process of a counter propagation net. The input vectors are clustered in the first stage. Originally, it is assumed that there is no topology included in the counter propagation network. However, on the inclusion of a linear topology, the performance of the net can be improved. The dusters are formed using Euclidean distance method or dot product method. In the second stage of training, the weights from the cluster layer units to the output units are tuned to obtain the desired response.

There are two types of counter propagation nets:

(i) Full counter propagation net

(ii) Forward-only counter propagation net

Full Counter propagation Net:

Full counter propagation net (full CPN) efficiently represents a large number of vector pairs *x:y* by adaptively constructing a look-up-table. The approximation here is x*.y*, which is based on the vector pairs x:*y,* possibly with some distorted or missing elements in either vector or both vectors. The network is defined to approximate a continuous function, defined on a compact set A. The full CPN works best if the inverse function f^{-1} exists and is continuous. The vectors *x* and *y* propagate through the network in a counterflow manner to yield output vectors x* and y*, which are the approximations of *x* and y, respectively. During competition, the winner can be determined either by Euclidean distance or by dot product method. In case of dot product method, the one with the largest net input is the winner. Whenever vectors are to be compared using the dot product metric, they should be normalized. Even though the normalization can be performed without loss of information by adding an extra component, yet to avoid

the complexity Euclidean distance method can be used. On the basis of this, direct comparison can be made between the full CPN and forward-only CPN.

For continuous function, the CPN is as efficient as the back-propagation net; it is a universal continuous function approximator. In case of CPN, the number of hidden nodes required to achieve a particular level of accuracy is greater than the number required by the back-propagation network. The greatest appeal of CPN is its speed of learning. Compared to various mapping networks, it requires only fewer steps of training to achieve best performance. This is common for any hybrid learning method that combines unsupervised learning (e.g., instar learning) and supervised learning (e.g., outstar learning).
As already discussed, the training of CPN occurs in two phases. In the input phase, the units in the duster layer and input layer are found to be active. In CPN, no topology is assumed for the cluster layer units; only the winning units are allowed to learn. The weight updation learning rule on the winning duster units is

$$v_{ij}(\text{new}) = v_{ij}(\text{old}) + \alpha\,[x_i - v_{ij}(\text{old})], \quad i = 1 \text{ to } n$$
$$w_{kj}(\text{new}) = w_{kj}(\text{old}) + \beta\,[y_k - w_{kj}(\text{old})], \quad k = 1 \text{ to } m$$

In the second phase of training, only the winner unit J remains active in the cluster layer. The weights between the winning cluster unit J and the output units are adjusted so that the vector of activations of the units in the Y-output layer is y* which is an approximation to the input vector *y* and X* which is an approximation to the input vector *x.* The weight updations for the units in the Y-output and X-output layers are

$$u_{jk}(\text{new}) = u_{jk}(\text{old}) + a[y_k - u_{jk}(\text{old})], \quad k = 1 \text{ to } m$$
$$t_{ji}(\text{new}) = t_{ji}(\text{old}) + b[x_i - t_{ji}(\text{old})], \quad i = 1 \text{ to } n$$

Architecture of Full Counter propagation Net

The general structure of full CPN is shown in Figure 5-15. The complete architecture of full CPN is shown in Figure 5-16.

The four major components of the instar-outstar model are the input layer, the instar, the competitive layer and the outstar. For each node *i* in the input layer, there is an input value x_i;. An instar responds maximally to the input vectors from a particular duster. All the instar are grouped into a layer called the competitive layer.

Each of the instar responds maximally to a group of input vectors in a different region of space. This layer of instars classifies any input vector because, for a given input, the winning instar with the strongest response identifies the region of space in which the input vector lies. Hence, it is necessary that the competitive layer single outs the winning instar by setting its output to a nonzero value and also suppressing the other outputs to zero. That is, it is a winner-take-all or a Maxnet-type network. An outstar model is found to have all the nodes in the output layer and a single node in the competitive layer. The outstar looks like the fan-out of a node. Figures 5-17 and 5-18 indicate the units that are active during each of the two phases of training a full CPN.

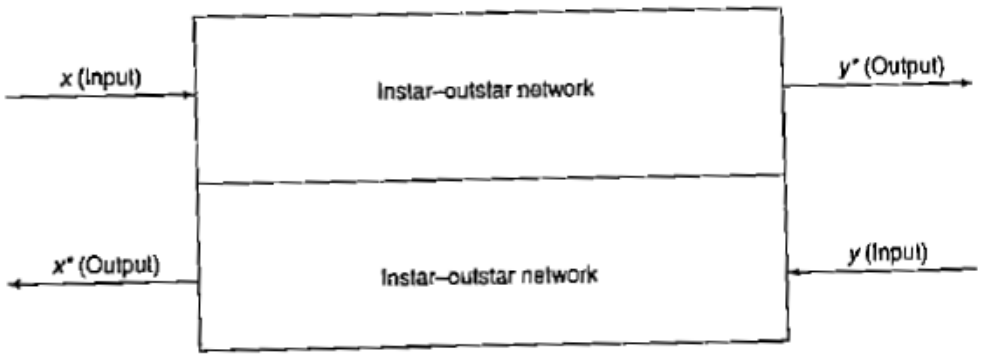


Figure 5-15 General structure of full CPN.

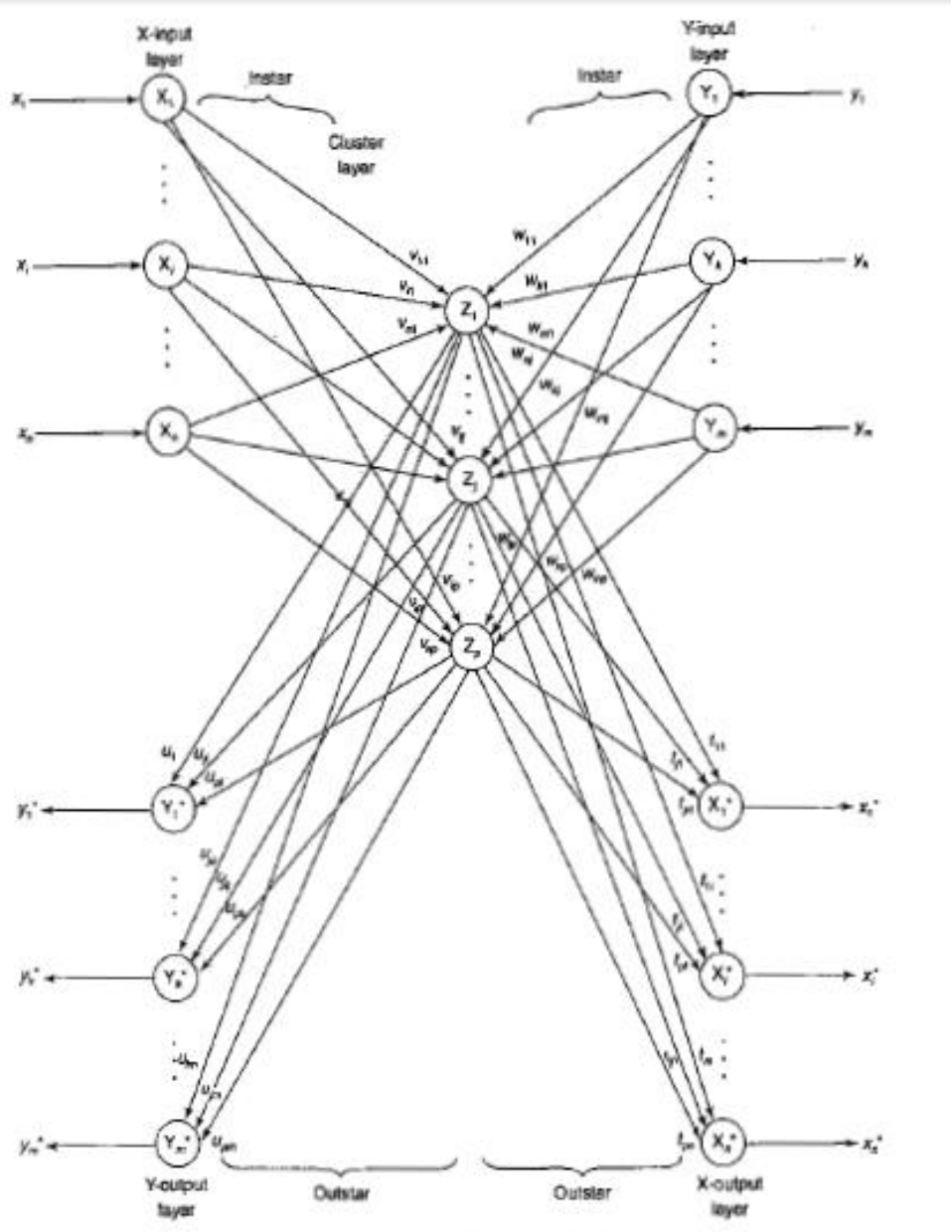


Figure 5-16 Architecture of full CPN.

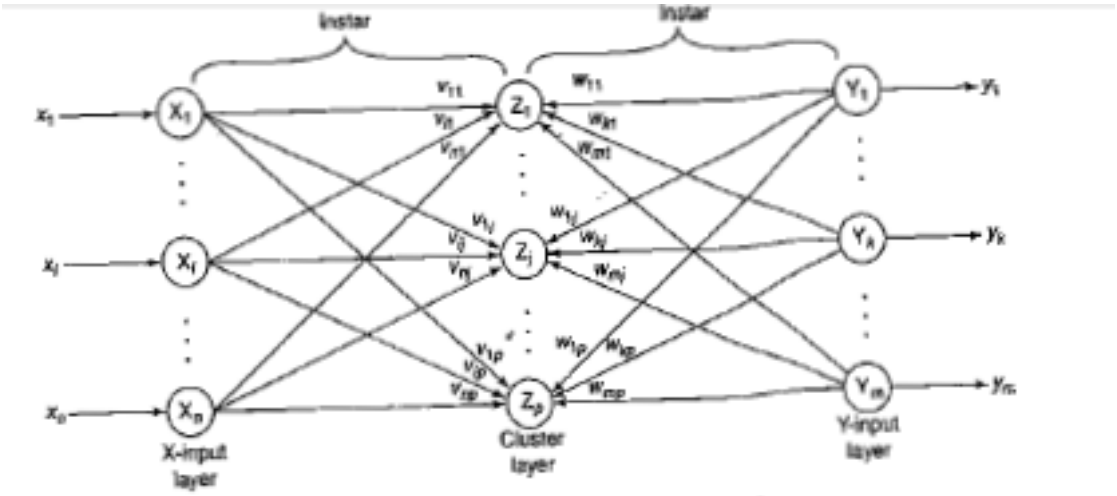


Figure 5-17 First phase of training of full CPN.

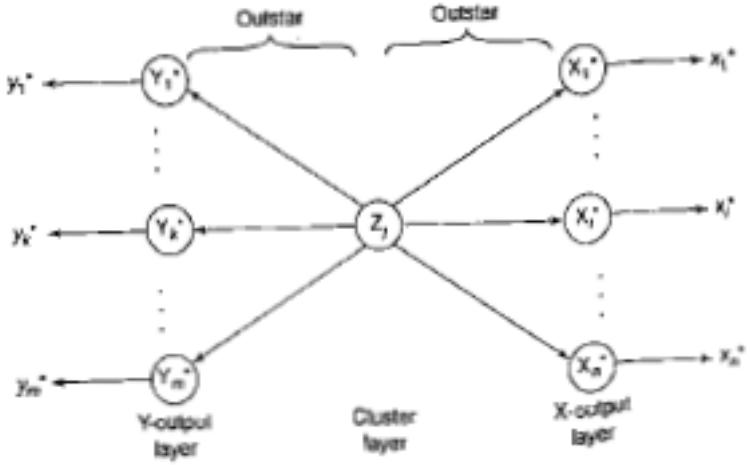


Figure 5-18 Second phase of training of full CPN.

Training Algorithm of Full Counter propagation Net:

Step 0: Set the initial weights and the initial learning rate.

Step 1: Perform Steps 2–7 if stopping condition is false for phase I training.

Step 2: For each of the training input vector pair $x : y$ presented, perform Steps 3–5.

Step 3: Make the X-input layer activations to vector X.
Make the Y-input layer activations to vector Y.

Step 4: Find the winning cluster unit.
If dot product method is used, find the cluster unit z_j with target net input; for $j = 1$ to p,

$$z_{inj} = \sum_{i=1}^{n} x_i v_{ij} + \sum_{k=1}^{m} y_k w_{kj}$$

If Euclidean distance method is used, find the cluster unit z_j whose squared distance from input vectors is the smallest:

$$D_j = \sum_{i=1}^{n} (x_i - v_{ij})^2 + \sum_{k=1}^{m} (y_k - w_{kj})^2$$

If there occurs a tie in case of selection of winner unit, the unit with the smallest index is the winner. Take the winner unit index as J.

Step 5: Update the weights over the calculated winner unit z_j.

$$\text{For } i = 1 \text{ to } n, \quad v_{ij}(\text{new}) = v_{ij}(\text{old}) + \alpha[x_i - v_{ij}(\text{old})]$$
$$\text{For } k = 1 \text{ to } m, \quad w_{kj}(\text{new}) = w_{kj}(\text{old}) + \beta[y_k - w_{kj}(\text{old})]$$

Step 6: Reduce the learning rates.

$$\alpha(t+1) = 0.5\,\alpha(t); \quad \beta(t+1) = 0.5\,\beta(t)$$

Step 7: Test stopping condition for phase I training.

Step 8: Perform Steps 9–15 when stopping condition is false for phase II training.

Step 9: Perform Steps 10–13 for each training input pair $x : y$. Here α and β are small constant values.

Step 10: Make the X-input layer activations to vector x. Make the Y-input layer activations to vector y.

Step 11: Find the winning cluster unit (use formulas from Step 4). Take the winner unit index as J.

Step 12: Update the weights entering into unit z_j.

$$\text{For } i = 1 \text{ to } n, \quad v_{ij}(\text{new}) = v_{ij}(\text{old}) + \alpha[x_i - v_{ij}(\text{old})]$$
$$\text{For } k = 1 \text{ to } m, \quad w_{kj}(\text{new}) = w_{kj}(\text{old}) + \beta[y_k - w_{kj}(\text{old})]$$

Step 13: Update the weights from unit z_j to the output layers.

$$\text{For } i = 1 \text{ to } n, \quad t_{ji}(\text{new}) = t_{ji}(\text{old}) + b[x_i - t_{ji}(\text{old})]$$
$$\text{For } k = 1 \text{ to } m, \quad u_{jk}(\text{new}) = u_{jk}(\text{old}) + a[y_k - u_{jk}(\text{old})]$$

Step 14: Reduce the learning rates a and b.

$$a(t+1) = 0.5\,a(t); \quad b(t+1) = 0.5\,b(t)$$

Step 15: Test stopping condition for phase II training.

Testing Algorithm of Full Counter propagation Net:

Step 0: Initialize the weights (from training algorithm).

Step 1: Perform Steps 2–4 for each input pair X : Y.

Step 2: Set X-input layer activations to vector X.
Set Y-input layer activations to vector Y.

Step 3: Find the cluster unit z_j that is closest to the input pair.

Step 4: Calculate approximations to x and y:

$$x_i^* = t_{ji}; \quad y_k^* = u_{jk}$$

Forward Only Counterpropagation Net:

A simplified version of full CPN is the forward-only CPN. The approximation of the function $y = f(x)$ but not of $x = f(y)$ can be performed using forward-only CPN, i.e., it may be used if the mapping from x to y is well defined but mapping from y to x *is* not defined. In forward-only CPN only the x-vectors are used to form the clusters on the Kohonen units. Forward-only CPN uses only the x vectors to form the clusters on the Kohonen units during first phase of training.

In case of forward-only CPN, first input vectors are presented to the input units. The cluster layer units compete with each other using winner-take-all policy to learn the input vector. Once entire set of training vectors has been presented, there exist reduction in learning rate and the vectors are presented again, performing several iterations. First the weights between the input layer and duster layer are trained. Then the weights between the cluster layer and output layer are trained. This is a specific competitive network, with target known. Hence, when each input vector is presented m the input vector, its associated target vectors are presented to the output layer. The winning duster unit sends its signal to the output layer. Thus each of the output unit has a computed signal (w_{jk}) and die target value (y_k). The difference between these values is calculated; based on this, the weights between the winning layer and output layer are updated. The weight updation from input units to cluster units is done using the learning rule given below:

For i= 1 to n,

$$v_{ij}(\text{new}) = v_{ij}(\text{old}) + \alpha[x_i - v_{ij}(\text{old})] = (1-\alpha)v_{ij}(\text{old}) + \alpha x_i$$

The weight updation from cluster units to output units is done using following the learning rule: For $k = 1$ to m,

$$w_{jk}(\text{new}) = w_{jk}(\text{old}) + \alpha[y_k - w_{jk}(\text{old})] = (1 - \alpha)w_{jk}(\text{old}) + \alpha y_k$$

The learning rule for weight updation from the cluster units to output units can be written in the form of delta rule when the activations of the cluster units (z_j) are included, and is given as

$$w_{jk}(\text{new}) = w_{jk}(\text{old}) + \alpha z_j[y_k - w_{jk}(\text{old})]$$

where

$$z_j = \begin{cases} 1 & \text{if } j = J \\ 0 & \text{if } j \neq J \end{cases}$$

This occurs when w_{jk} is interpreted as the computed output (i.e., $y_k = w_{jk}$). In the formulation of forward-only CPN also, no topological structure was assumed.

Architecture of Forward Only Counterpropagation Net:

Figure 5-20 shows the architecture of forward-only CPN. It consists of three layers: input layer, cluster (competitive) layer and output layer. The architecture of forward-only CPN resembles the back-propagation network, but in CPN there exists interconnections between the units in the duster layer (which are nor connected in Figure 5-20). Once competition is completed in a forward-only CPN, only one unit will be active in that layer and it sends signal to the output layer. As inputs are presented m the network, the desired outputs will also be presented simultaneously.

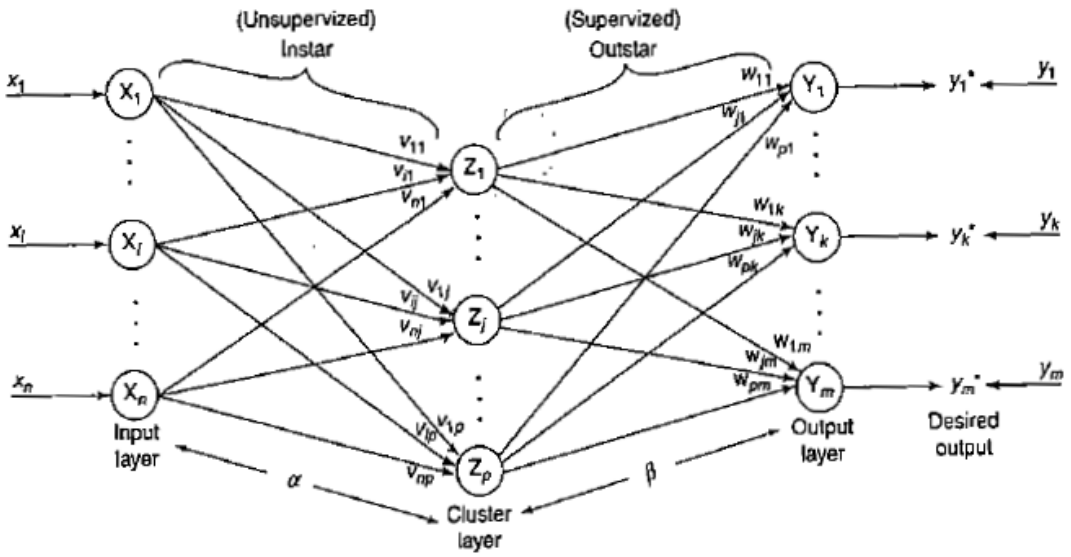


Figure 5-20 Architecture of forward-only CPN.

Training Algorithm of Forward Only Counterpropagation Net:

Step 0: Initialize the weights and learning rates.

Step 1: Perform Steps 2–7 when stopping condition for phase I training is false.

Step 2: Perform Steps 3–5 for each of training input X.

Step 3: Set the X-input layer activations to vector X.

Step 4: Compute the winning cluster unit (J). If dot product method is used, find the cluster unit z_j with the largest net input:

$$z_{inj} = \sum_{i=1}^{n} x_i v_{ij}$$

If Euclidean distance is used, find the cluster unit z_j square of whose distance from the input pattern is smallest:

$$D_j = \sum_{i=1}^{n} (x_i - v_{ij})^2$$

If there exists a tie in the selection of winner unit, the unit with the smallest index is chosen as the winner.

Step 5: Perform weight updation for unit z_j. For $i = 1$ to n,

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

Step 6: Reduce learning rate α

$$\alpha(t+1) = 0.5\,\alpha(t)$$

Step 7: Test the stopping condition for phase I training.

Step 8: Perform Steps 9–15 when stopping condition for phase II training is false. (Set α a small constant value for phase II training.)

Step 9: Perform Steps 10–13 for each training input pair x:y.

Step 10: Set X-input layer activations to vector X. Set Y-output layer activations to vector Y.

Step 11: Find the winning cluster unit (J) [use formulas as in Step 4].

Step 12: Update the weights into unit z_J. For $i = 1$ to n,

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

Step 13: Update the weights from unit z_J to the output units. For $k = 1$ to m,

$$w_{Jk}(\text{new}) = w_{Jk}(\text{old}) + \beta[y_k - w_{Jk}(\text{old})]$$

Step 14: Reduce learning rate β, i.e.,

$$\beta(t+1) = 0.5\beta(t)$$

Step 15: Test the stopping condition for phase II training.

Testing Algorithm of Forward Only Counterpropagation Net:

Step 0: Set initial weights. (The initial weights here are the weights obtained during training.)

Step 1: Present input vector X.

Step 2: Find unit J that is closest to vector X.

Step 3: Set activations of output units:

$$y_k = w_{Jk}$$

Adaptive Resonance Theory Network

The adaptive resonance theory (ART) network, developed by Steven Grossberg and Gail Carpenter (1987), is consistent with behavioral models. This is an unsupervised learning, based on competition, that finds categories autonomously and learns new categories if needed. The adaptive resonance model was developed to solve the problem of instability occurring in feed-forward systems. There are two types of ART: ART 1 and ART 2. ART 1 is designed for clustering binary vectors and ART 2 is designed to accept continuous-valued vectors. In both the nets, input patterns can be presented in any order. For each pattern, presented to the network, an appropriate cluster unit is chosen and the weights of the cluster unit are adjusted to let the cluster unit learn the pattern. This network controls the degree of similarity of the patterns placed on the same cluster units. During training, each training pattern may be presented several times. It should be noted that the input patterns should not be presented on the same cluster unit, when it is presented each time. On the basis of this, the stability of the net is defined as that wherein a pattern is not presented to previous cluster units.

The adaptive resonance theory (ART) network, developed by Steven Grossberg and Gail Carpenter (1987), is consistent with behavioral models. This is an unsupervised learning, based on competition, that finds categories autonomously and learns new categories if needed. The adaptive resonance model was developed to solve the problem of instability occurring in feed-forward systems. There are two types of ART: ART 1 and ART 2. ART 1 is designed for clustering binary vectors and ART 2 is designed to accept continuous-valued vectors. In both the nets, input patterns can be presented in any order. For each pattern, presented to the network, an appropriate cluster unit is chosen and the weights of the cluster unit are adjusted to let the cluster unit learn the pattern. This network controls the degree of similarity of the patterns placed on the same cluster units. During training, each training pattern may be presented several times. It should be noted that the input patterns should not be presented on the same cluster unit, when it is presented each time. On the basis of this, the stability of the net is defined as that wherein a pattern is not presented to previous cluster units. The stability may be achieved by reducing the learning rates. The ability of the network to respond to a new pattern equally at any stage of learning is called as plasticity. ART nets are designed to possess the properties, stability and plasticity. The key concept of ART is that the stability plasticity can be resolved by a system in which the network includes bottom-up (input–output) competitive learning combined with top-down (output–input) learning. The instability of instar–outstar networks could be solved by reducing the learning rate gradually to zero by freezing the learned categories. But, at this point, the net may lose its plasticity or the ability to react to new data. Thus it is difficult to possess both stability and plasticity. ART networks are designed particularly to resolve the stability–plasticity dilemma, that is, they are stable to preserve significant past learning but nevertheless remain adaptable to incorporate new information whenever it appears.

Fundamental architecture of ART-

Three groups of neurons reused to build an ART network. These include:

1. Input processing neurons (F1 layer).
2. Clustering units (F2 layer).
3. Control mechanism (controls degree of similarity of patterns placed on the same duster

The input processing neuron (F_1) layer consists of two portions: Input portion and interface portion. The input portion may perform some processing based on the inputs it receives. This is especially performed in the case of ART 2 compared to ART 1. The interface portion of the F_1 layer combines the input from input portion of F_1 and F_2 layers for comparing the similarity of the input signal with the weight vector for the cluster unit that has been selected as a unit for learning. F_1 layer input portion may be denoted as F_1(a) and interface portion as F_1(b).

There exist two sets of weighted interconnections for controlling the degree of similarity between the units in the interface portion and the cluster layer. The bottom-up weights are used for the connection from F_1(b) layer to F_2 layer and are represented by b_{ij} (*i*th F_1 unit to *j*th F_2 unit). The top-down weights are used for the connection from F_2 layer to F_1(b) layer and are represented by t_{ji} (*j*th F_2 unit to *i*th F_1 unit). The competitive layer in this case is the cluster layer and the cluster unit with largest net input is the victim to learn the input pattern, and the activations of all other F_2 units are made zero. The interface units combine the data from input and cluster layer units. On the basis of the similarity between the top-down weight vector and input vector, the cluster unit may be allowed to learn the input pattern. This decision is done by reset mechanism unit on the basis of the signals it receives from interface portion and input portion of the F_1 layer. When cluster unit is not allowed to learn, it is inhibited and a new cluster unit is selected as the victim.

Fundamental algorithm of ART-

Step 0: Initialize the necessary parameters.

Step 1: Perform Steps 2–9 when stopping condition is false.

Step 2: Perform Steps 3–8 for each input vector.

Step 3: F_1 layer processing is done.

Step 4: Perform Steps 5–7 when reset condition is true.

Step 5: Find the victim unit to learn the current input pattern. The victim unit is going to be the F_2 unit (that is not inhibited) with the largest input.

Step 6: F_1(b) units combine their inputs from F_1(a) and F_2.

Step 7: Test for reset condition.
If reset is true, then the current victim unit is rejected (inhibited); go to Step 4. If reset is false, then the current victim unit is accepted for learning; go to next step (Step 8).

Step 8: Weight updation is performed.

Step 9: Test for stopping condition.

5.6.2 Adaptive Resonance Theory 1

Adaptive resonance theory 1 (ART 1) network is designed for binary input vectors. As discussed generally, the ART 1 net consists of two fields of units–input unit (F_1 unit) and output unit (F_2 unit)–along with the reset control unit for controlling the degree of similarity of patterns placed on the same cluster unit. There exist two sets of weighted interconnection path between F_1 and F_2 layers. The supplemental unit present in the net provides the efficient neural control of the learning process. Carpenter and Grossberg have designed ART 1 network as a real-time system. In ART 1 network, it is not necessary to present an input pattern in a particular order; it can be presented in any order. ART 1 network can be practically implemented by analog circuits governing the differential equations, i.e., the bottom-up and top-down weights are controlled by differential equations. ART 1 network runs throughout autonomously. It does not require any external control signals and can run stably with infinite patterns of input data.

ART 1 network is trained using fast learning method, in which the weights reach equilibrium during each learning trial. During this resonance phase, the activations of F_1 units do not change; hence the equilibrium weights can be determined exactly. The ART 1 network performs well with perfect binary input patterns, but it is sensitive to noise in the input data. Hence care should be taken to handle the noise.

Fundamental architecture of ART1-

The ART 1 network is made up of two units:

1. Computational units.
2. Supplemental units.

In this section we will discuss in detail about these two units.

Computational units

The computational unit for ART 1 consists of the following:

1. Input units (F_1 unit – both input portion and interface portion).
2. Cluster units (F_2 unit – output unit).
3. Reset control unit (controls degree of similarity of patterns placed on same cluster).

The basic architecture of ART 1 (computational unit) is shown in Figure 5-22. Here each unit present in the input portion of F_1 layer (i.e., F_1(a) layer unit) is connected to the respective unit in the interface portion of F_1 layer (i.e., F_1(b) layer unit). Reset control unit has connections from each of F_1(a) and F_1(b) units. Also, each unit in F_1(b) layer is connected through two weighted interconnection paths to each unit in F_2 layer and the reset control unit is connected to every F_2 unit. The X_i unit of F_1(b) layer is connected to Y_j unit of F_2 layer through bottom-up weights (b_{ij}) and the Y_j unit of F_2 is connected to X_i unit of F_1 through top-down weights (t_{ji}). Thus ART 1 includes a bottom-up competitive learning system combined with a top-down outstar learning system. In Figure 5-22 for simplicity only the weighted interconnections b_{ij} and t_{ji} are shown, the other units' weighted interconnections are in a similar way. The cluster layer (F_2 layer) unit is a competitive layer, where only the uninhibited node with the largest net input has nonzero activation.

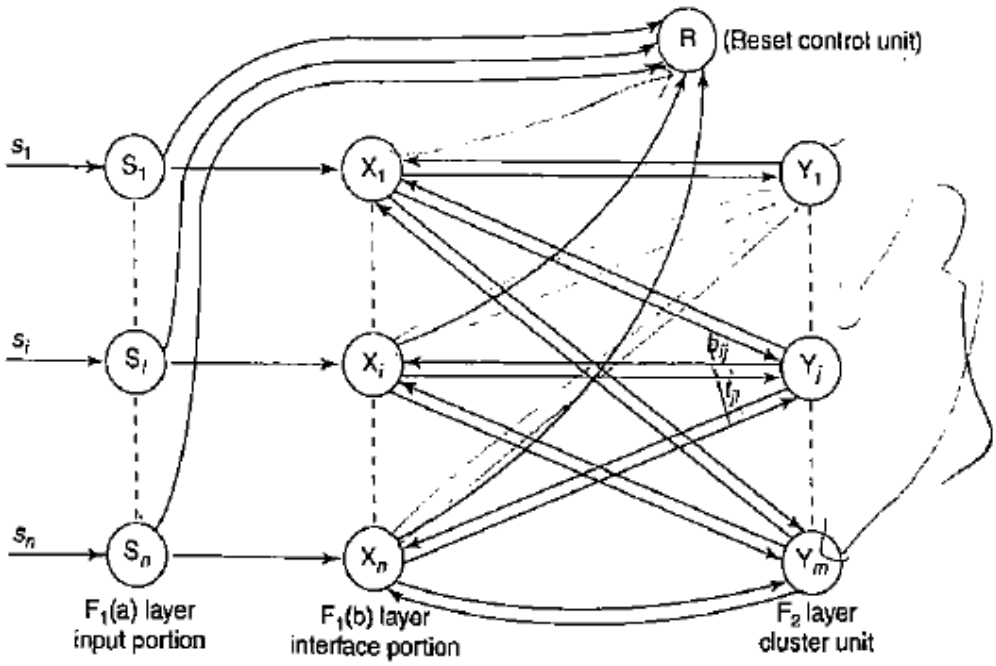


Figure 5-22 Basic architecture of ART 1.

Training Algorithm of ART1-

Step 0: Initialize the parameters:

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

Initialize the weights:

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

Step 1: Perform Steps 2–13 when stopping condition is false.

Step 2: Perform Steps 3–12 for each of the training input.

Step 3: Set activations of all F_2 units to zero. Set the activations of F_1(a) units to input vectors.

Step 4: Calculate the norm of s:

$$\|s\| = \sum_i s_i$$

Step 5: Send input signal from F_1(a) layer to F_1(b) layer:

$$x_i = s_i$$

Step 6: For each F_2 node that is not inhibited, the following rule should hold: If $y_j \neq -1$, then

$$y_j = \sum_i b_{ij} x_i$$

Step 7: Perform Steps 8–11 when reset is true.

Step 8: Find J for $y_J \geq y_j$ for all nodes j. If $y_J = -1$, then all the nodes are inhibited and note that this pattern cannot be clustered.

Step 9: Recalculate activation X of F_1(b):

$$x_i = s_i t_{Ji}$$

Step 10: Calculate the norm of vector x:

$$\|x\| = \sum_i x_i$$

Step 11: Test for reset condition.
If $\|x\|/\|s\| < \rho$, then inhibit node J, $y_J = -1$. Go back to step 7 again.
Else if $\|x\|/\|s\| \geq \rho$, then proceed to the next step (Step 12).

Step 12: Perform weight updation for node J (fast learning):

$$b_{ij}(\text{new}) = \frac{\alpha x_i}{\alpha - 1 + \|x\|}$$

$$t_{ji}(\text{new}) = x_i$$

Step 13: Test for stopping condition. The following may be the stopping conditions:

a. No change in weights.
b. No reset of units.
c. Maximum number of epochs reached.

Adaptive Resonance Theory 2 (ART2):

Adaptive resonance theory 2 (ART 2) is for continuous-valued input vectors. In ART 2 network complexity is higher than ART 1 network because much processing is needed in F 1 layer. ART 2 network was developed by Carpenter and Grossberg in 1987. ART 2 network was designed to self-organize recognition categories for analog as well as binary input sequences. The major difference between ART l and ART 2 networks is the input layer. On the basis of the stability criterion for analog inputs, a three-layer feedback system in the input layer of ART 2 network is required: A bottom layer where the input patterns are read in, a top layer where inputs coming from the output layer are read in and a middle layer where the top and bottom patterns are combined together to form a marched pattern which is then fed back to the top and bottom input layers. The complexity in the F1 layer is essential because continuous-valued input vectors may be arbitrarily dose together. The F1 layer consists of normalization and noise suppression parameter, in addition to comparison of the bottom-up and top-down signals, needed for the reset mechanism.

The continuous-valued inputs presented to the ART 2 network may be of two forms. The first form is a "noisy binary" signal form, where the information about patterns is delivered primarily based on the components which are "on" or "off," rather than the differences existing in the magnitude of the components chat are positive. In this case, fast learning mode is best adopted. The second form of patterns are those, in which the range of values of the components carries significant information and the weight vector for a cluster is found to be interpreted as exemplar for· the patterns placed-on chat unit. In this type of pattern, slow learning mode is best adopted. The second form of data is "truly continuous."

Fundamental architecture of ART2-

A typical architecture of ART 2 network is shown in Figure 5-25. From the figure, we can notice that F_1 layer consists of six types of units – W, X, U, V, P, Q – and there are "n" units of each type. In Figure 5-25, only one of these units is shown. The supplemental part of the connection is shown in Figure 5-26.

The supplemental unit "N" between units W and X receives signals from all "W" units, computes the norm of vector w and sends this signal to each of the X units. This signal is inhibitory signal. Each of this $(X_1, \ldots, X_i, \ldots, X_n)$ also receives excitatory signal from the corresponding W unit. In a similar way, there exists supplemental units between U and V, and P and Q, performing the same operation as done between W and X. Each X unit and Q unit is connected to V unit. The connections between P_i of the F_1 layer and Y_j of the F_2 layer show the weighted interconnections, which multiplies the signals transmitted over those paths. The winning F_2 units' activation is d $(0 < d < 1)$. There exists normalization between W and X, V and U, and P and Q. The normalization is performed approximately to unit length.

The operations performed in F_2 layer are same for both ART 1 and ART 2. The units in F_2 layer compete with each other in a winner-take-all policy to learn each input pattern. The testing of reset condition differs for ART 1 and ART 2 networks. Thus in ART 2 network, some processing of the input vector is necessary because the magnitudes of the real valued input vectors may vary more than for the binary input vectors.

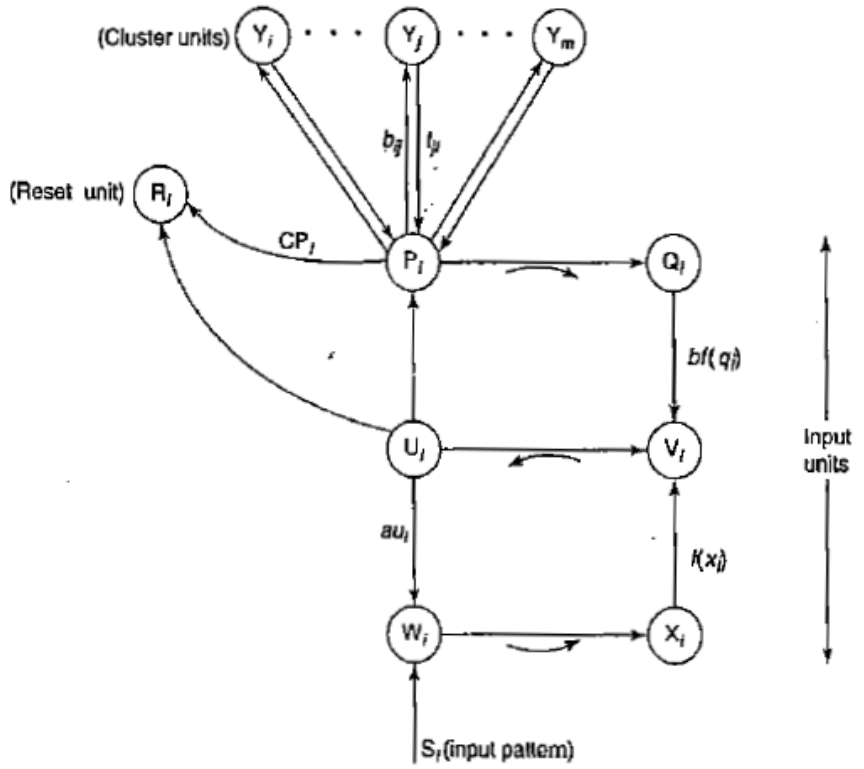


Figure 5-25 Architecture of ART 2 network.

Training Algorithm of ART2:

Step 0: Initialize the following parameters: a, b, c, d, e, α, ρ, θ. Also, specify the number of epochs of training (nep) and number of learning iterations (nit).

Step 1: Perform Steps 2–12 (nep) times.

Step 2: Perform Steps 3–11 for each input vector s.

Step 3: Update F_1 unit activations:

$$u_i = 0; \quad w_i = s_i; \quad P_i = 0; \quad q_i = 0; \quad v_i = f(x_i);$$

$$x_i = \frac{s_i}{e + \|s\|}$$

Update F_1 unit activations again:

$$u_i = \frac{v_i}{e + \|v\|}; \quad w_i = s_i + au_i;$$

$$P_i = u_i; \quad x_i = \frac{w_i}{e + \|w\|};$$

$$q_i = \frac{p_i}{e + \|p\|}; \quad v_i = f(x_i) + bf(q_i)$$

In ART 2 networks, norms are calculated as the square root of the sum of the squares of the respective values.

Step 4: Calculate signals to F_2 units:

$$y_j = \sum_{i=1}^{n} b_{ij} p_i$$

Step 5: Perform Steps 6 and 7 when reset is true.

Step 6: Find F_2 unit Y_J with largest signal (J is defined such that $y_j \geq y_j, j = 1$ to m).

Step 7: Check for reset:

$$u_i = \frac{v_i}{e + \|v\|}; \quad P_i = u_i + dt_{ji}; \quad r_i = \frac{u_i + cP_i}{e + \|u\| + c\|p\|}$$

If $\|r\| < (\rho - e)$, then $y_J = -1$ (inhibit J). Reset is true; perform Step 5.
If $\|r\| \geq (\rho - e)$, then

$$w_i = s_i + au_i; \quad x_i = \frac{w_i}{e + \|w\|};$$

$$q_i = \frac{p_i}{e + \|p\|}; \quad v_i = f(x_i) + bf(q_i)$$

Reset is false. Proceed to Step 8.

Step 8: Perform Steps 9–11 for specified number of learning iterations.

Step 9: Update the weights for winning unit J:

$$t_{Ji} = \alpha d u_i + \{[1 + \alpha d(d-1)]\} t_{Ji}$$
$$b_{iJ} = \alpha d u_i + \{[1 + \alpha d(d-1)]\} b_{iJ}$$

Step 10: Update F_1 activations:

$$u_i = \frac{v_i}{e + \|v\|}; \quad w_i = s_i + au_i;$$

$$P_i = u_i + dt_{Ji}; \quad x_i = \frac{w_i}{e + \|w\|};$$

$$q_i = \frac{P_i}{e + \|p\|}; \quad v_i = f(x_i) + bf(q_i)$$

Step 11: Check for the stopping condition of weight updation.

Step 12: Check for the stopping condition for number of epochs.

Special Networks

Simulated Annealing Network

The concept of simulated annealing has it origin in the physical annealing process performed over metals and other substances. In metallurgical annealing, a metal body is heated almost to its melting point and then cooled back slowly to room temperature. This process eventually makes the metal's global energy function reach an absolute minimum value. If the metal's temperature is reduced quickly, the energy of the metallic lattice will be higher than this minimum value because of the existence of frozen lattice dislocations that would otherwise disappear due to thermal agitation. Analogous to the physical annealing behavior, simulated annealing can make a system change its state to a higher energy state having a chance to jump from local minima or global maxima. There exists a *cooling procedure* in the simulated annealing process such that the system has a higher probability of changing to an increasing energy state in the beginning phase of convergence. Then, as time goes by, the system becomes stable and always moves in the direction of decreasing energy state as in the case of normal minimization produce.

With simulated annealing, a system changes its state from the original state SA^{old} to a new state SA^{new} with a probability P given by

$$P = \frac{1}{1 + \exp(-\Delta E/T)}$$

where $\Delta E = E^{old} - E^{new}$ (energy change = difference in new energy and old energy) and T is the non-negative parameter (acts like temperature of a physical system). The probability P as a function of change in energy (ΔE) obtained for different values of the temperature T is shown in Figure 6-1. From Figure 6-1, it can be noticed that the probability when $\Delta E > 0$ is always higher than the probability when $\Delta E < 0$ for any temperature.

An optimization problem seeks to find some configuration of parameters $\dot{X} = (X_1, \ldots, X_n)$ that minimizes some function $f(\dot{X})$ called cost function. In an artificial neural network, configuration parameters are associated with the set of weights and the cost function is associated with the error function.

The simulated annealing concept is used in statistical mechanics and is called Metropolis algorithm. *As* discussed earlier, this algorithm is based on a material that anneals into a solid as temperature is slowly decreased. To understand this, consider the slope of a hill having local valleys. A stone is moving down the hill. Here, the local valleys are local minima, and the bottom of the hill is going to be the global or universal minimum. It is possible that the stone may stop at a local minimum and never reaches the global minimum. In neural nets, this would correspond to a set of weights that correspond to that of local minimum, but this is nm the desired solution. Hence, to overcome this kind of situation, simulated annealing perturbs the stone such that if it is trapped in a local minimum, it escapes from it and continues falling till it reaches its global minimum (optimal solution). At that point, further perturbations cannot move the stone to a lower position.

Figure 6-2 shows the simulated annealing between a stone and a hill.

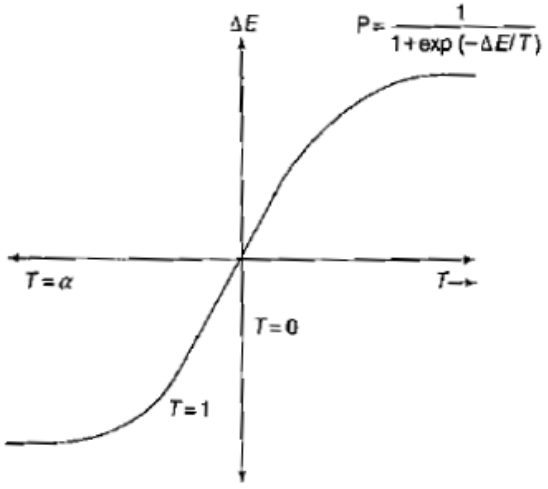


Figure 6-1 Probability "P" as a function of change in energy (ΔE) for different values of temperature T.

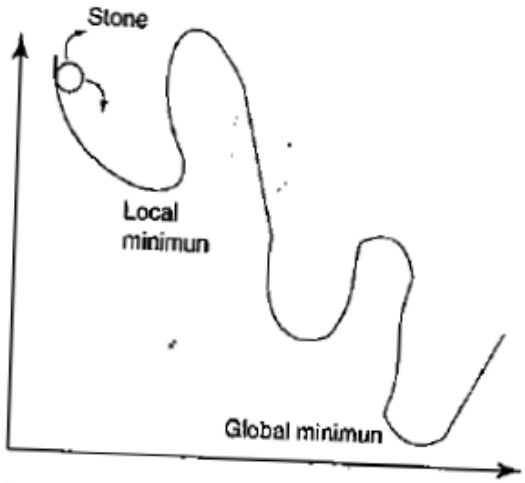


Figure 6-2 Simulated annealing—stone and hill.

The components required for annealing algorithm are the following.

1. *A basic system configuration:* The possible solution of a problem over which we search for a best (optimal) answer. (In a neural net, this is optimum steady-state weight.)
2. *The move set:* A set of allowable moves that permit us to escape from local minima and reach all possible configurations.
3. *A cost function* associated with the error function.
4. *A cooling schedule:* Starting of the cost function and rules to determine when it should be lowered and by how much, and when annealing should be terminated.

Simulated annealing networks can be used to make a network converge to its global minimum.

Boltzmann Machine

The early optimization technique used in artificial neural networks is based on the Boltzmann machine. When the simulated annealing process is applied *w* the discrete Hopfield network, it becomes a Boltzmann machine. The network is configured as the vector of the states of the units, and the stares of the units are binary valued with probabilities state transition. The Boltzmann machine described in this

section has fixed weights *wij*. On applying the Boltzmann machine to a constrained optimization problem, the weights represent the constraints of the problem and the quantity to be optimized. The discussion here is based on the fact of maximization of a consensus function (CF).

The Boltzmann machine consists of a set of units (Xi, and Xj) and a set of bi-directional connections between pairs of units. This machine can be used as an associative memory. If the units *Xi;* and Xj are connected, then $w_{ij} \neq 0$. There exists symmetry in the weighted interconnections based on the directional nature. It can be represented as $w_{ij}=w_{ji}$. There also may exist a self-connection for a unit (w_{ij}). For unit *Xi,* its State x_i; may be either 1 or 0. The objective of the neural net is to maximize the CF given by

$$CF = \sum_i \sum_{j \leq i} w_{ij} x_i x_j$$

The maximum of the CF can be obtained by letting each unit attempt to change its state (alter between "1" and "0" or "0" and "1"). The change of state can be done either in parallel or sequential manner. However, in this case all the description is based on sequential manner. The consensus change when unit X_i changes its state is given by

$$\Delta CF(i) = (1 - 2x_i)\left(w_{ij} + \sum_{j \neq i} w_{ij} x_i\right)$$

where x_i is the current state of unit X_i. The variation in coefficient $(1 - 2x_i)$ is given by

$$(1 - 2x_i) = \begin{cases} +1, & X_i \text{ is currently off} \\ -1, & X_i \text{ is currently on} \end{cases}$$

If unit X_i were to change its activations then the resulting change in the CF can be obtained from the information that is local to unit X_i. Generally, X_i does not change its state, but if the states are changed, then this increases the consensus of the net. The probability of the network that accepts a change in the state for unit X_i is given by

$$AF(i, T) = \frac{1}{1 + \exp[-\Delta CF(i)/T]}$$

where T (temperature) is the controlling parameter and it will gradually decrease as the CF reaches the maximum value. Low values of T are acceptable because they increase the net consensus since the net accepts a change in state. To help the net not to stick with the local maximum, probabilistic functions are used widely.

Architecture of Boltzmann Machine

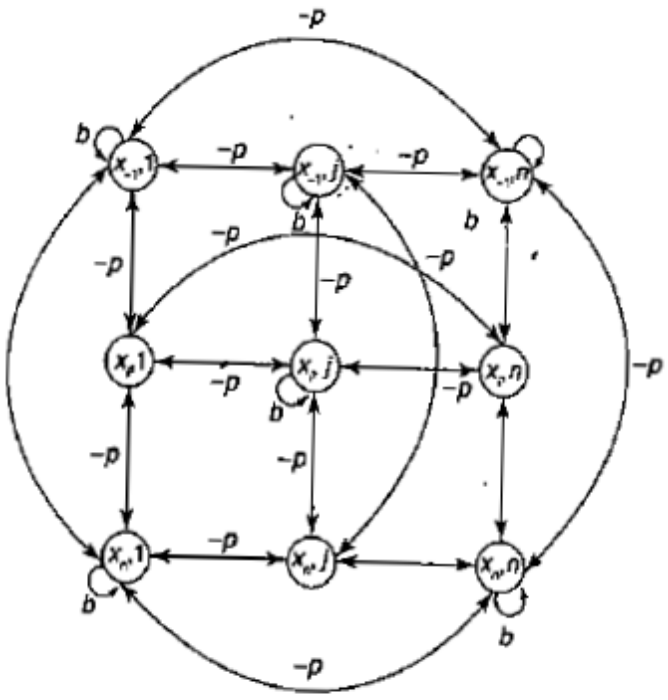


Figure 6-3 Architecture of Boltzmann machine.

Testing Algorithm of Boltzmann Machine

Step 0: Initialize the weights representing the constraints of the problem. Also initialize control parameter T and activate the units.

Step 1: When stopping condition is false, perform Steps 2–8.

Step 2: Perform Steps 3–6 n^2 times. (This forms an epoch.)

Step 3: Integers I and J are chosen random values between 1 and n. (Unit $U_{I,J}$ is the current victim to change its state.)

Step 4: Calculate the change in consensus:

$$\Delta CF = (1 - 2X_{I,J}) \left[w(I,J : I,J) + \sum_{i,j \neq} \sum_{I,J} w(i,j : I,J) X_{I,J} \right]$$

Step 5: Calculate the probability of acceptance of the change in state:

$$AF(T) = 1/1 + \exp[-(\Delta CF/T)]$$

Step 6: Decide whether to accept the change or not. Let R be a random number between 0 and 1. If $R < AF$, accept the change:

$X_{I,J} = 1 - X_{I,J}$ (This changes the state $U_{I,J}$.)

If $R \geq AF$, reject the change.

Step 7: Reduce the control parameter T.

$$T(\text{new}) = 0.95\ T(\text{old})$$

Step 8: Test for stopping condition, which is:
If the temperature reaches a specified value or if there is no change of state for specified number of epochs then stop, else continue.

Gaussian Machine

Gaussian machine is one which includes Boltzmann machine, Hopfield net and other neural networks. The Gaussian machine is based on the following three parameters: (a) a slope parameter of sigmoidal function α, (b) a time step Δt, (c) temperature T.

The steps involved in the operation of the Gaussian net are the following:

Step 1: Compute the net input to unit X_i:

$$\text{net}_i = \sum_{j=1}^{N} w_{ij} v_j + \theta_i + \epsilon$$

where θ_i is the threshold and ϵ the random noise which depends on temperature T.

Step 2: Change the activity level of unit X_i:

$$\frac{\Delta x_i}{\Delta t} = -\frac{x_i}{t} + \text{net}_i$$

Step 3: Apply the activation function:

$$v_i = f(x_i) = 0.5[1 + \tanh(x_i)]$$

where the binary step function corresponds to $\alpha = \infty$ (infinity).

The Gaussian machine with $T = 0$ corresponds the Hopfield net. The Boltzmann machine can be obtained by setting $\Delta t = \tau = 1$ to get

$$\Delta x_i = -x_i + \text{net}_i$$

$$\text{or} \quad x_i(\text{new}) = \text{net}_i = \sum_{j=1}^{N} w_{ij} v_j + \theta_i + \epsilon$$

The approximate Boltzmann acceptance function is obtained by integrating the Gaussian noise distribution

$$\int_0^{\infty} \frac{1}{\sqrt{2\pi\sigma^2}} \exp \frac{(x - x_i^2)}{2\sigma^2} dx \approx \text{AF}(i, T) = \frac{1}{1 + \exp(-x_i/T)}$$

where $x_i = \Delta \text{CF}(i)$. The noise which is found to obey a logistic rather than a Gaussian distribution produces a Gaussian machine that is identical to Boltzmann machine having Metropolis acceptance function, i.e., the output set to 1 with probability,

$$\text{AF}(i, T) = \frac{1}{1 + \exp(-x_i/T)}$$

Cauchy Machine

Cauchy machine can be called fast simulated annealing, and it is based on including more noise to the net input for increasing the likelihood of a unit escaping from a neighborhood of local minimum. Larger changes in the system's configuration can be obtained due to the unbounded variance of the Cauchy distribution. Noise involved in Cauchy distribution is called "colored noise" and the noise involved in the Gaussian distribution is called "white noise."

By setting $\Delta t = \tau = 1$, the Cauchy machine can be extended into the Gaussian machine, to obtain

$$\Delta x_i = -x_i + \text{net}_i$$

$$\text{or} \quad x_i(\text{new}) = \text{net}_i = \sum_{j=1}^{N} w_{ij} v_j + \theta_i + \in$$

The Cauchy acceptance function can be obtained by integrating the Cauchy noise distribution:

$$\int_0^{\infty} \frac{1}{\pi} \frac{T\,dx}{T^2 + (x - x_i)^2} = \frac{1}{2} + \frac{1}{\pi} \arctan\left(\frac{x_i}{T}\right) = \text{AF}(i, T)$$

where $x_i = \Delta \text{CF}(i)$. The cooling schedule and temperature have to be considered in both Cauchy and Gaussian machines.

Probabilistic Neural Net

The probabilistic neural net *is* based on the idea of conventional probability theory, such as Bayesian classification and other estimators for probability density functions, to construct a neural net for classification. This net instantly approximates optimal boundaries between categories. It assumes that the training data are original representative samples. The probabilistic neural net consists of two hidden layers as shown in Figure 6-4. The first hidden layer contains a dedicated node for each training p:mern :md the second hidden layer contains a dedicated node for each class. The two hidden layers are connected on a class-by-class basis, that is, the several examples of the class in the first hidden layer are connected only to a single machine unit in the second hidden layer.

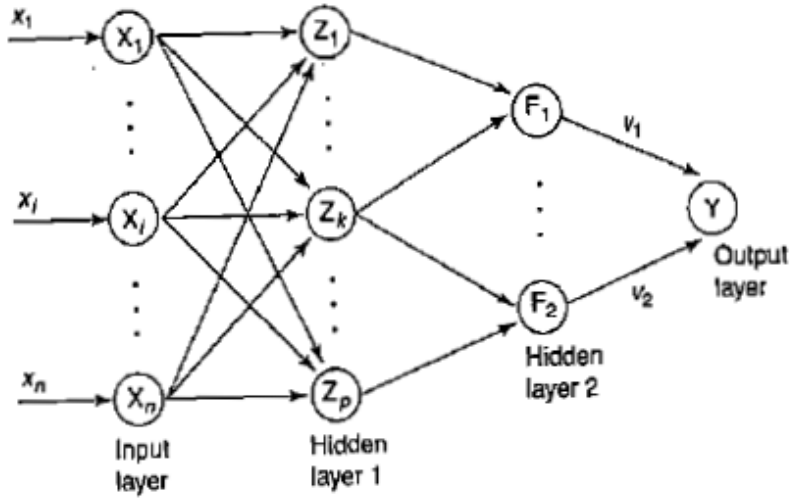


Figure 6-4 Probabilistic neural network.

The algorithm for the construction of the net is as follows:

Step 0: For each training input pattern $x(p)$, $p = 1$ to P, perform Steps 1 and 2.

Step 1: Create pattern unit z_k (hidden-layer-1 unit). Weight vector for unit z_k is given by

$$w_k = x(p)$$

Unit z_k is either z-class-1 unit or z-class-2 unit.

Step 2: Connect the hidden-layer-1 unit to the hidden-layer-2 unit.
If $x(p)$ belongs to class 1, then connect the hidden layer unit z_k to the hidden layer unit F_1.
Otherwise, connect pattern hidden layer unit z_k to the hidden layer unit F_2.

Cascade Correlation Network:

Cascade correlation is a network which builds its own architecture as the training progresses. Figure 6-5 shows the cascade correlation architecture. The network begins with some inputs and one or more output nodes, but it has no hidden nodes. Each and every input is connected to every output node. There may be linear units or some nonlinear activation function such as bipolar sigmoidal activation function in the output nodes. During training process, new hidden nodes are added to the network one by one. For each new hidden node, the correlation magnitude between the new node's output and the residual error signal is maximized. The connection is made to each node from each of the network's original inputs and also from every preexisting hidden node. During the time when the node is being added to the network, the input weights of the hidden nodes are-frozen, and only the output connections are trained repeatedly. Each new node thus adds a new one-node layer to the network.

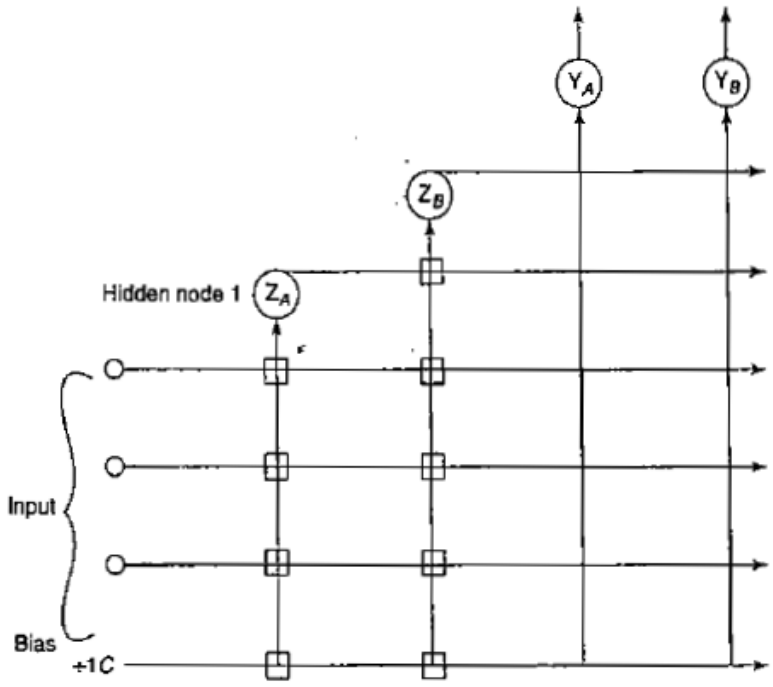


Figure 6-5 Cascade architecture after two hidden nodes have been added.

In Figure 6-5, the vertical lines sum all incoming activations. The rectangular boxed connections are frozen and "0" connections are trained continuously. In the beginning of the training, there are no hidden nodes, and the network is trained over the complete training set. Since there is no hidden node, a simple learning rule, Widrow-Hofflearning rule, is used for training. After a certain number of training cycles, when there is no significant error reduction and the final error obtained is unsatisfactory, we try to reduce

the residual errors further by adding a new hidden node. For performing this task, we begin with a candidate node that receives trainable input connections from the network's external inputs and from all pre-existing hidden nodes. The output of this candidate node is not yet connected to the active network. After this, we run several numbers of epochs for the training set. We adjust the candidate node's input weights after each -epoch to maximize *C* which is defined as

$$C = \sum_i \left| \sum_j (v_j - \bar{v})(E_{j,i} - \bar{E}_o) \right|$$

where i is the network output at which error is measured, j the training pattern, v the candidate node's output value, E_o the residual output error at node o, $\bar{v}$ the value of v averaged over all patterns, $\bar{E}_o$ the value of E_o averaged over all patterns. The value "C" measures the correlation between the candidate node's output value and the calculated residual output error. For maximizing C, the gradient $\partial c/\partial w_i$ is obtained as

$$\frac{\partial c}{\partial w_i} = \sum_{j,i} \sigma_i (E_{j,i} - \bar{E}_i) d_j' I_{m,j}$$

where σ_i is the sign of the correlation between the candidate's value and output i; d_j' the derivative for pattern j of the candidate node's activation function with respect to sum of its inputs; $I_{m,j}$ the input the candidate node receives from node m for pattern j. When gradient $\partial c/\partial w_i$ is calculated, perform gradient ascent to maximize C. As we are training only a single layer of weights, simple delta learning rule can be applied. When C stops improving, again a new candidate can be brought in as a node in the active network and its input weights are frozen. Once again all the output weights are trained by the delta learning rule as done previously, and the whole cycle repeats itself until the error becomes acceptably small.

Cognitron Network:

The synaptic strength from cell X to cell Y is reinforced if and only if the following two conditions are true:

l. Cell X- presynaptic cell fires.
2. None of the postsynaptic cells present near cell Y fire stronger than Y.

The model developed *by* Fukushima was called cognitron as a successor to the perceptron which can perform cognizance of symbols from any alphabet after training. Figure 6-6 shows the connection between presynaptic cell and postsynaptic cell.

The cognitron network is a self-organizing multilayer neural network. Its nodes receive input from the defined areas of the previous layer and also from units within its own area. The input and output neural elements can rake the form of positive analog values, which are proportional to the pulse density of firing biological neurons. The cells in the cognitron model use a mechanism of shunting inhibition, i.e., a cell is bound in terms of a maximum and minimum activities and is driven toward these extremities. The area from which the cell receives input is called connectable area. The area formed by the inhibitory cluster is called the vicinity area. Figure 6. 7 shows the model of a cognitron. Since the connectable areas for cells in the same vicinity are defined to overlap, but are not exactly the same, there will be-a slight difference appearing between the cells which is reinforced so that the gap becomes more apparent. Like this, each cell is allowed to develop its own characteristics.

Cognitron network can be used in neurophysiology and psychology. Since this network closely resembles the natural characteristics of a biological neuron, this is best suited for various kinds of visual and auditory information processing systems. However, a major drawback of cognitron net is that it

cannot deal with the problems of orientation or distortion. To overcome this drawback, an improved version called neocognitron was developed.

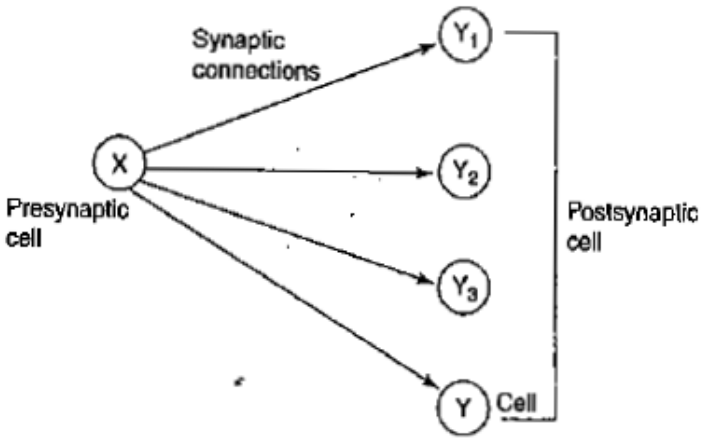


Figure 6-6 Connection between presynaptic cell and postsynaptic cell.

Figure 6-7 Model of a cognitron network.

Neocognitron Network

Neocognitron is a multilayer feed-forward network model for visual pattern recognition. It is a hierarchical net comprising many layers and there is a localized pattern of connectivity between the layers. It is an extension of cognitron network. Neocognitron net can be used for recognizing hand-written characters. A neocognitron model is shown in Figure 6·8.

The algorithm used in cognitron and neocognitron is same, except that neocognicron model can recognize patterns that are position-shifted or shape-distorted. The cells used in neocognitron are of two types:

1. *S·-cell:* Cells that are trained suitably to. respond to only certain features in the previous layer.
1. *C-cell·* A C-cell displaces the result of an S-cell in space, i.e., son of "spreads" the features recognized by the S-cell.

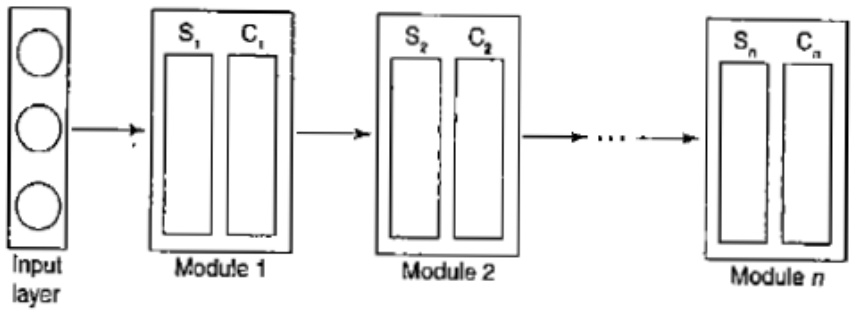


Figure 6-8 Neocognitron models.

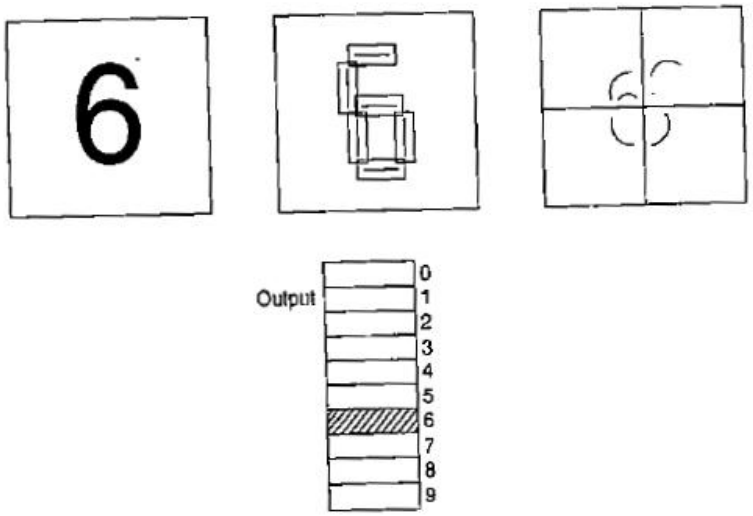


Figure 6-9 Spreading effect in neocognitron.

Neocognitron net consists of many modules with the layered arrangement of S-cells and C-cells. The S-cells receive the input from the previous layer, while C-cells receive the input from the S-layer. During training, only the inputs to the S-layer are modif1ed. The S-layer helps in the detection of spccif1c features and their complexities. The feature recognized in the S_1 layer may be a horizontal bar or a vertical bar but the feature in the Sn layer may be more complex. Each unit in the C-layer corresponds to one relative position independent feature. For the independent feature, C-node receives the inputs from a subset of S-layer nodes. For instance, if one node in C-layer detects a vertical line and if four nodes in the preceding S-layer detect a vertical line, then these four nodes will give the input to the specific node in C-layer to spatially distribute the extracted features. Modules present near the input layer (lower in hierarchy) will be trained before the modules that are higher in hierarchy, i.e., module 1 will be trained before module 2 and so on.

The users have to fix the "receptive field" of each C-node before training starts because the inputs to C-node cannot be modified. The lower level modules have smaller receptive fields while the higher level modules indicate complex independent features present in the hidden layer. The spreading effect used in neocognitron is shown in Figure 6-9.

Cellular Neural Network –

cellular neural network (CNN), introduced in 1988, is based on cellular automata, i.e., every cell in the network is connected only to its neighbor cells. Figures 6-10 (A) and (B) show 2 x 2 CNN and 3 x 3 CNN, respectively. The basic unit of a CNN is a cell. In Figures 6-10(A) and (B), C(l, l) and C(2, 1) are called as cells.

Even if the cells are not directly connected with each other, they affect each other indirectly due to propagation effects of the network dynamics. The CNN can be implemented by means of a hardware model. This is achieved by replacing each cell with linear capacitors and resistors, linear and nonlinear controlled sources, and independent sources. An electronic circuit model can be constructed for a CNN. The CNNs are used in a wide variety of applications including image processing, pattern recognition and array computers.

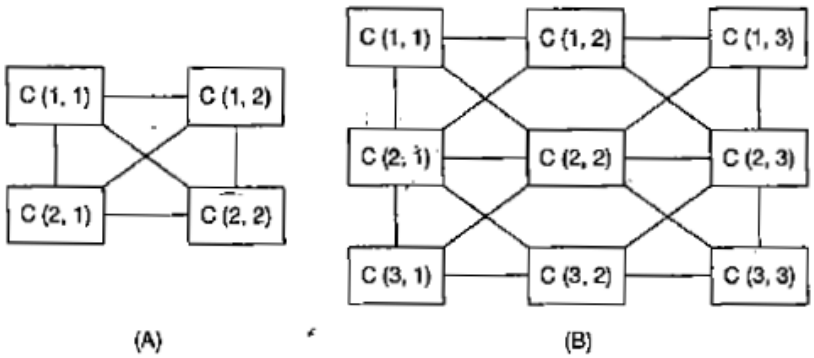


Figure 6-10 (A) A 2 × 2 CNN; (B) a 3 × 3 CNN.

Optical Neural Networks

Optical neural networks interconnect neurons with light beams. Owing to this interconnection, no insulation is required between signal paths and the light rays can pass through each other without interacting. The path of the signal travels in three dimensions. The transmission path density is limited by the spacing of light sources, the divergence effect and the spacing, of detectors. *A$* a result, all signal paths operate simultaneously, and true data rare results are produced. In holograms with high density, the weighted strengths are stored.

These stored weights can be modified during training for producing a fully adaptive system. There are two classes of this optical neural network. They are:

1. **electro-optical multipliers;**
2. **holographic correlators**.

Electro-optical multipliers

Electro-optical multipliers, also called electro-optical matrix multipliers, perform matrix multiplication in parallel. The network speed is limited only by the available electro-optical components; here the computation time is potentially in the nanosecond range. A model of electro-optical matrix multiplier is shown in Figure 6-11.

Figure 6-11 shows a system which can multiply a nine-element input vector by a 9 X 7 matrix, which produces a seven-element NET vector. There exists a column of light sources that passes its rays through a lens; each light illuminates a single row of weight shield. The weight shield is a photographic film where transmittance of each square (as shown in Figure 6-11) is proportional to the weight. There is another lens that focuses the light from each column of the shield m a corresponding photoelectron. The NET is calculated as

$$\mathrm{NET}_k = \sum_i w_{ik} x_i$$

where NET_k is the net output of neuron *k;* w_{ik} the weight from neuron *i* to neuron *k;* x_i the input vector component *i.* The output of each photodetector represents the dot product between the input vector and a column of the weight matrix. The output vector set is equal to the produce of the input vector with weight matrix. Hence, matrix multiplication is performed parallelly. The speed is independent of the size of the array.

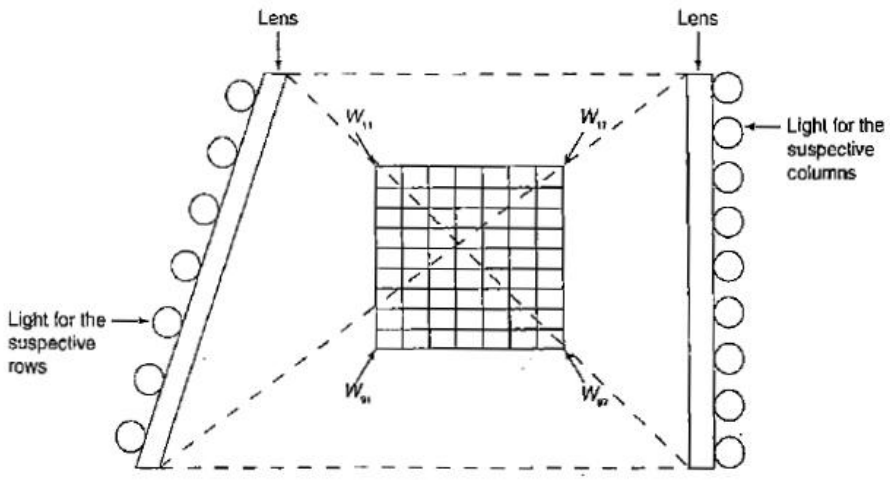


Figure 6-11 Electro-optical multiplier.

Holographic Correlators

In holographic correlators, the reference images are stored in a thin hologram and are retrieved in a coherently illuminated feedback loop. The input signal, either noisy or incomplete, may be applied to the system and can simultaneously be correlated optically with all the stored reference images. These. correlations can be threshold and are fed back to the input, where the strongest correlation reinforces the input image. The enhanced image passes around the loop repeatedly, which approaches the stored image more closely on each pass, until the system gets stabilized on the desired image. The best performance of optical correlators is obtained when they are used for image recognition. A generalized optical image recognition system with holograms is shown in Figure 6- 12.

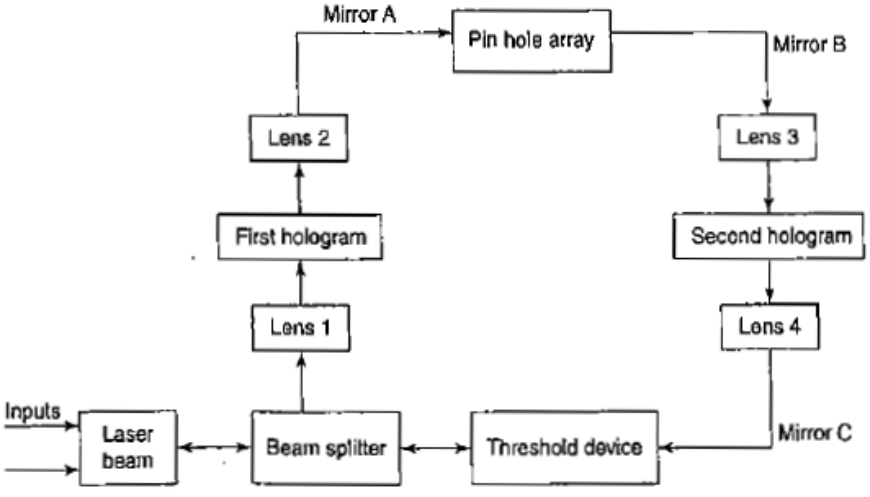


Figure 6-12 Optical image recognition system.

The system input is an image from a laser beam. This passes through a beam splitter, which sends it to the threshold device. The image is reflected, then gets reflected from the threshold device, passes back to the beam splitter, then goes to lens 1, which makes it fall on the first hologram. There are several stored images in first hologram. The image then gets correlated with each stored image. This correlation produces light patterns. The brightness of the patterns varies with the degree of correlation. The projected images from lens 2 and mirror A pass through pinhole array, where they are spatially separated. From this array, light patterns go to mirror B through lens 3 and then are applied to the second hologram. Lens 4

and mirror C then produce superposition of the multiple correlated images o1nto the back side of the threshold device.

The front surface of the threshold device reflects most strongly that pattern which is brightest on its rear surface. Its rear surface has projected on it the set of four correlations of each of the four stored images with the input image. The stored image that is similar to the input image possesses highest correlation. *This* reflected image again passes through the beam splitter and reenters the loop for further enhancement. The system gets converged on the stored patterns most like the input pattern.