COUNTER PROPAGATION NETWORK

The **counterpropagation network** is a hybrid network. It consists of an outstar network and a competitive filter network. It was developed in 1986 by Robert Hecht-Nielsen. It is guaranteed to find the correct weights, unlike regular back propagation networks that can become trapped in local minimums during training.

The input layer neurode connect to each neurode in the hidden layer. The hidden layer is a Kohonen network which categorizes the pattern that was input. The output layer is an outstar array which reproduces the correct output pattern for the category.

Training is done in two stages. The hidden layer is first taught to categorize the patterns and the weights are then fixed for that layer. Then the output layer is trained. Each pattern that will be input needs a unique node in the hidden layer, which is often too large to work on real world problems.

ARCHITECTURE

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. It allows the output of a pattern rather than a simple category number. It can also be viewed as a bidirectional associative memory. Figure 1 shows architecture of the 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.

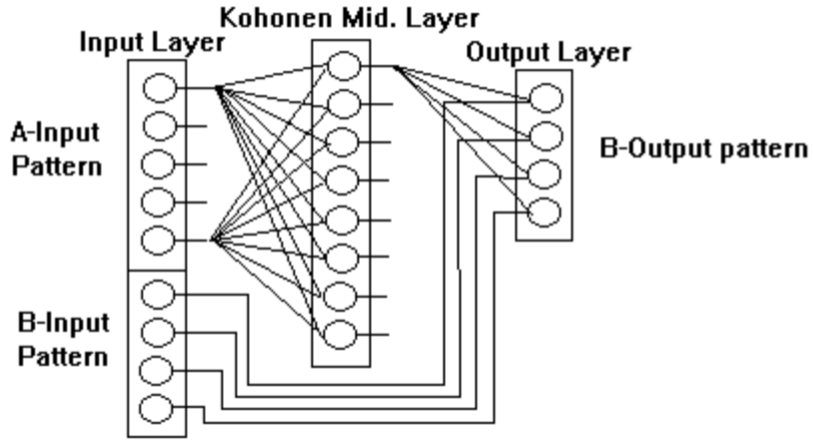


Fig 1 Architecture of Unidirectional counter propagation network

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.

TRAINING ALGORITHM STAGE 1:

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

$$\mathbf{w}(t+1) = \mathbf{w}(t) + \alpha(\mathbf{x} - \mathbf{w})$$

Step 4 Repeat steps 1 through 3 until all vectors have been processed.

Step 5 Repeat steps 1 to 4 until all input vectors have been learned.

Step 1 Apply normalized input vector **x** and its corresponding output vector **y**, to inputs *A* and *B* respectively.

Step 2 Determine winning node in the Kohonen layer.

Step 3 Update weights on the connections from the winning node to the output unit -

$$w_i(t+1) = w_i(t) + \alpha(y_i - w_i)$$

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

BIDIRECTIONAL COUNTERPROPAGATION NETWORK

A bidirectional Counterpropagation network is capable of a two-way mapping. For example, an *A* pattern input produces a *B* pattern output and a *B* pattern input produces an *A* pattern output.

The Figure 2 illustrates the connections in a bidirectional Counterpropagation network. The input and output layers are now of the same size, equal to the sum of the sizes of the *A* and *B* patterns. Both *A* and *B* sections have full connections to the middle layer, and one-to-one connections to the corresponding neurons in the output layer. The middle layer receives input from all elements of the input layer and transmits its output to the entire output layer.

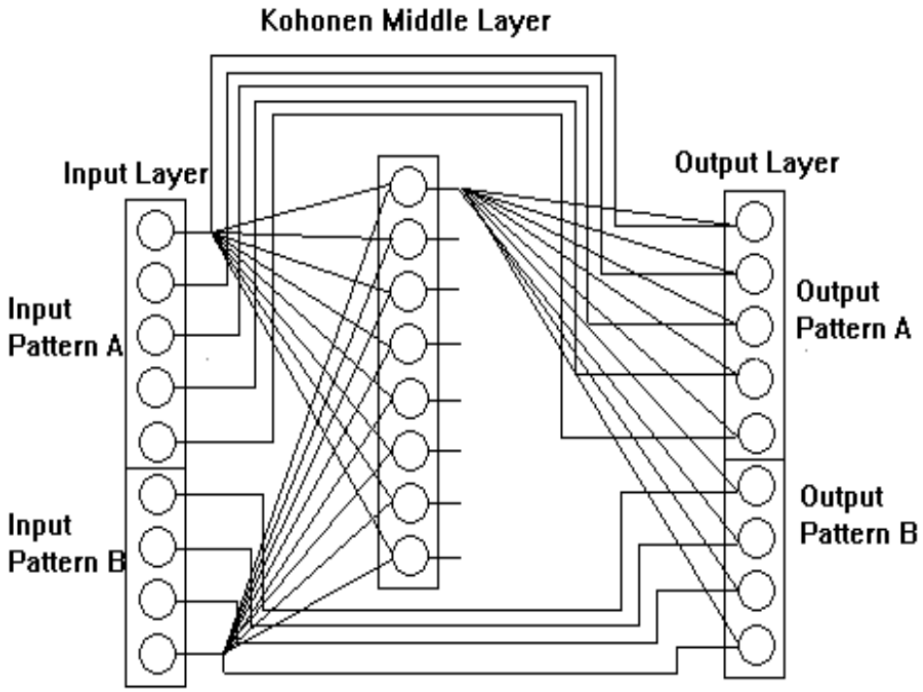


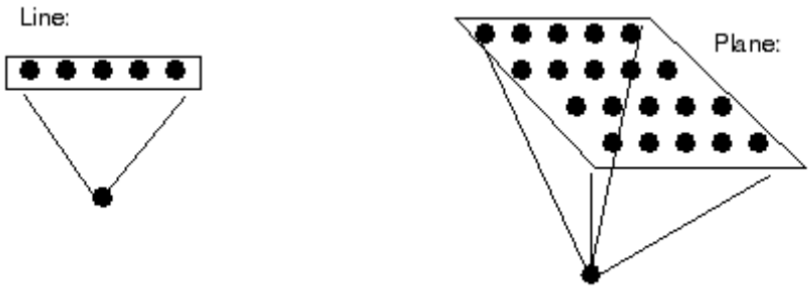
Fig 2. Architecture of Bidirectional counter propagation network

KOHONEN SELF ORGANIZATION MAPS

Kohonon's SOMs are a type of unsupervised learning. The goal is to discover some underlying structure of the data. Kohonen's SOM is called a topology-preserving map because there is a topological structure imposed on the nodes in the network. A topological map is simply a mapping that preserves neighborhood relations.

In the nets we have studied so far, we have ignored the geometrical arrangements of output nodes. Each node in a given layer has been identical in that each is connected with all of the nodes in the upper and/or lower layer. We are now going to take into consideration that physical arrangement of these nodes. Nodes that are "close" together are going to interact differently than nodes that are "far" apart.

What do we mean by "close" and "far"? We can think of organizing the output nodes in a line or in a



planar configuration.

The goal is to train the net so that nearby outputs correspond to nearby inputs.

E.g. if x1 and x2 are two input vectors and t1 and t2 are the locations of the corresponding winning output nodes, then t1 and t2 should be close if x1 and x2 are similar. A network that performs this kind of mapping is called a feature map.

In the brain, neurons tend to cluster in groups. The connections within the group are much greater than the connections with the neurons outside of the group. Kohonen's network tries to mimick this in a simple way.

The main difference between them and conventional models is that the correct output cannot be defined *a priori*, and therefore a numerical measure of the magnitude of the mapping error cannot be used. However, the learning process leads, as before, to the determination of well-defined network parameters for a given application.

The grid of computing elements allows us to identify the immediate neighbors of a unit. This is very important, since during learning the weights of computing units and their neighbors are updated. The objective of such a learning approach is that neighboring units learn to react to closely related signals.

LEARNING ALGORITHM

Consider the problem of charting an *n*-dimensional space using a one-dimensional chain of Kohonen units. The units are all arranged in sequence and are numbered from
1 to *m* as shown in Figure 3. Each unit becomes the *n*-dimensional input x and computes the corresponding excitation. The *n*-dimensional weight vectors $w_1, w_2, \ldots, w_m$ are used for the computation. The objective of the charting process is that each unit learns to specialize on different regions of input space. When an input from such a region is fed into the network, the corresponding unit should compute the maximum excitation. Kohonen's learning algorithm is used to guarantee that this effect is achieved.

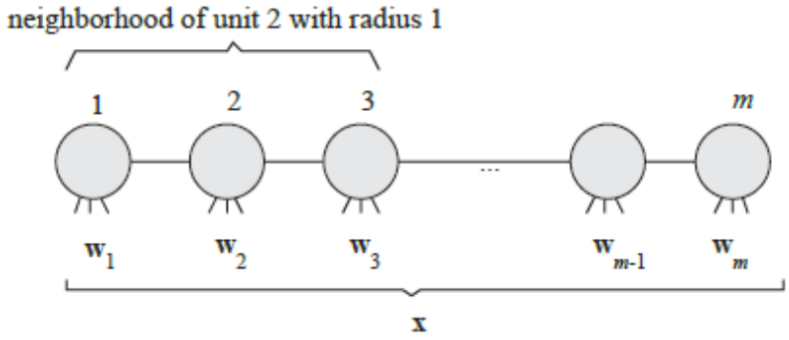


Figure 3. One dimensional lattice of computing units

A Kohonen unit computes the Euclidian distance between an input x and its weight vector w. This new definition of excitation is more appropriate for certain applications and also easier to visualize. In the Kohonen one-dimensional network, the neighborhood of radius 1 of a unit at the *k*-th position consists of the units at the positions $k - 1$ and $k + 1$. Units at both ends of the chain have asymmetrical neighborhoods. The neighborhood of radius *r* of unit *k* consists of all units located up to *r* positions from *k* to the left or to the right of the chain.

Kohonen learning uses a neighborhood function φ, whose value φ(i, k) represents the strength of the coupling between unit i and unit k during the training process. A simple choice is defining φ(i, k) = 1 for all units i in a neighborhood of radius r of unit k and φ(i,k) = 0 for all other units. We will later discuss the problems that can arise when some kinds of neighborhood functions are chosen. The learning algorithm for Kohonen networks is the following:

start : The *n*-dimensional weight vectors $w_1, w_2, \ldots, w_m$ of the *m* computing units are selected at random. An initial radius *r*, a learning constant *η*, and a neighborhood function *φ* are selected.

step 1 : Select an input vector *ξ* using the desired probability distribution over the input space.

step 2 : The unit *k* with the maximum excitation is selected (that is, for which the distance between w_i and *ξ* is minimal, $i = 1, \ldots, m$).

step 3 : The weight vectors are updated using the neighborhood function and the update rule

$$w_i \leftarrow w_i + \eta\varphi(i, k)(\xi - w_i), \qquad \text{for } i = 1, \ldots, m.$$

step 4 : Stop if the maximum number of iterations has been reached; otherwise modify *η* and *φ* as scheduled and continue with step 1.

The modifications of the weight vectors (step 3) attracts them in the direction of the input *ξ*. By repeating this simple process several times, we expect to arrive at a uniform distribution of weight vectors in input space (if the inputs have also been uniformly selected). The radius of the neighborhood is reduced according to a previous plan, which we call a *schedule*. The effect is that each time a unit is

updated, neighboring units are also updated. If the weight vector of a unit is attracted to a region in input space, the neighbors are also attracted, although to a lesser degree. During the learning process both the size of the neighborhood and the value of φ fall gradually, so that the influence of each unit upon its neighbors is reduced. The learning constant controls the magnitude of the weight updates and is also reduced gradually. The net effect of the selected schedule is to produce larger corrections at the beginning of training than at the end.

CPN (COUNTERPROPAGATION NETWORK) REDEFINED

CPN (Counter propagation networks) are multilayer network based on the combinations of the input, output, and clustering layers. The application of counter propagation net are data compression, function approximation and pattern association. The counter propagation network is basically constructed from an instar out star model. This model is three-layer neural network that performs input-output data mapping, producing an output vector y in response to input vector x, on the basis of competitive learning. The three layer in an instar - outstar model are the input layer, the hidden(competitive) layer and the output layer.

There are two stages involved in the training process of a counter propagation net. The input vector is clustered in the first stage. 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 net:

1. Full counter propagation network

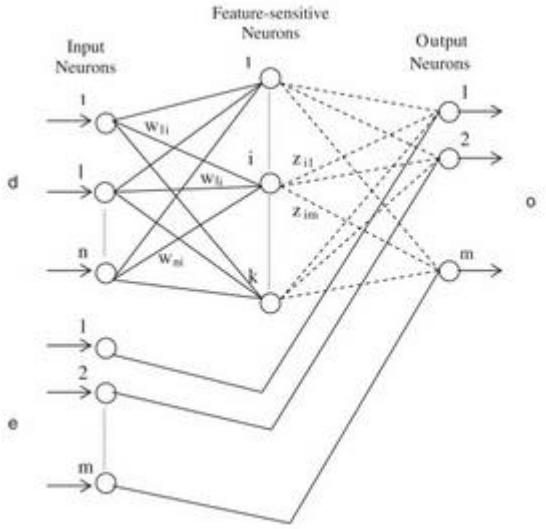
2. Forward-only counter propagation network

Full counter propagation network:

Full CPN efficiently represents a large number of vector pair x:y by adaptively constructing a look-up-table. The full CPN works best if the inverse function exists and is continuous. The vector x and y propagate through the network in a counterflow manner to yield output vector x* and y*.

Architecture of Full CPN:

The four major components of the instar-outstar model are the input layer, the instar, the competitive layer and the outstar. For each node in the input layer there is an input value xi. 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. 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.

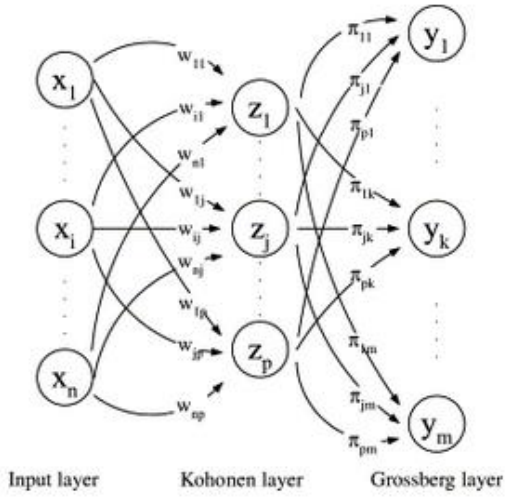


Forward-only Counter propagation network:

A simplified version of full CPN is the forward-only CPN. Forward-only CPN uses only the x vector to form the cluster on the Kohonen units during phase I training. In case of forward-only CPN, first input vectors are presented to the input units. First, the weights between the input layer and cluster layer are trained. Then the weights between the cluster layer and output layer are trained. This is a specific competitive network, with target known.

Architecture of forward-only CPN:

It consists of three layers: input layer, cluster layer and output layer. Its architecture resembles the back-propagation network, but in CPN there exists interconnections between the units in the cluster layer.



REFERENCE

David, Wolpert (1992). "Stacked generalization". Neural Networks. 5 (2): 241–259. CiteSeerX 10.1.1.133.8090.

Bengio, Y. (2009-11-15). "Learning Deep Architectures for AI". Foundations and Trends in Machine Learning. 2 (1): 1–127. CiteSeerX 10.1.1.701.9550.

Deng, Li; Yu, Dong; Platt, John (2012). "Scalable stacking and learning for building deep architectures" (PDF). 2012 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP): 2133–2136.

Werbos, P. J. (1988). "Generalization of backpropagation with application to a recurrent gas market model". Neural Networks

R. J. Williams and D. Zipser. Gradient-based learning algorithms for recurrent networks and their computational complexity. In Back-propagation: Theory, Architectures and Applications. Hillsdale, NJ: Erlbaum, 1994.

Mass, Wolfgang; Nachtschlaeger, T.; Markram, H. (2002). "Real-time computing without stable states: A new framework for neural computation based on perturbations". Neural Computation.

Kemp, Charles; Perfors, Amy; Tenenbaum, Joshua (2007). "Learning over hypotheses with hierarchical Bayesian models". Developmental Science.