COMPETITIVE ANN

UNSUPERVISED LEARNING

As the name suggests, this type of learning is done without the supervision of a teacher. This learning process is independent. During the training of ANN under unsupervised learning, the input vectors of similar type are combined to form clusters. When a new input pattern is applied, then the neural network gives an output response indicating the class to which input pattern belongs. In this, there would be no feedback from the environment as to what should be the desired output and whether it is correct or incorrect. Hence, in this type of learning the network itself must discover the patterns, features from the input data and the relation for the input data over the output.

Winner-Takes-All Networks

These kinds of networks are based on the competitive learning rule and will use the strategy where it chooses the neuron with the greatest total inputs as a winner. The connections between the output neurons show the competition between them and one of them would be 'ON' which means it would be the winner and others would be 'OFF'.

Following are some of the networks based on this simple concept using unsupervised learning.

Hamming Network

In most of the neural networks using unsupervised learning, it is essential to compute the distance and perform comparisons. This kind of network is Hamming network, where for every given input vectors, it would be clustered into different groups. Following are some important features of Hamming Networks

- Lippmann started working on Hamming networks in 1987.
- It is a single layer network.
- The inputs can be either binary {0, 1} or bipolar {-1, 1}.
- The weights of the net are calculated by the exemplar vectors.
- It is a fixed weight network which means the weights would remain the same even during training.

MAXNET

MAXNET is a specific example of a neural net based on competition. It can be used as a subnet to pick the node whose input is the largest. The m nodes in this subnet are completely interconnected, with symmetric weights. There is no training algorithm for the MAXNET; the weights are fixed.

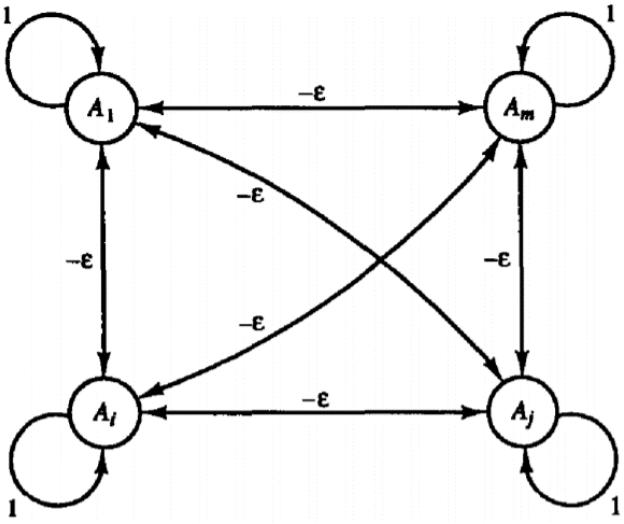


Figure 1: MAXNET Architecture

Application

The activation function for the MAXNET is

$$f(x) = \begin{cases} x & if\ x \geq 0; \\ 0 & otherwise \end{cases}$$

The application procedure is as follows:

Step 0. Initialize activations and weights (set $0 < \epsilon < \frac{1}{m}$):

$$a_j(0) \qquad \text{input to node } A_j,$$

$$w_{ij} = \begin{cases} 1 & if\ i = j; \\ -\epsilon & if\ i \neq j. \end{cases}$$

Step 1. While stopping condition is false, do steps 2-4.

Step 2. Update the activation of each node: For $j = 1, \dots, m$,

$$a_j(new) = f[a_j(old) - \epsilon \sum_{\lambda \neq j} a_\lambda(old)].$$

Step 3. Save activations for use in next iteration:

$$a_j(old) = a_j(new), j = 1, \dots, m.$$

Step 4. Test stopping condition:

If more than one node has a nonzero activation, continue; otherwise, stop.

Note that in Step 2, the input to the function f is simply the total input to node A_j from all nodes, including itself. Some precautions should be incorporated to handle the situation in which two or more units have the same, maximal, input.

Example: Using a MAXNET

Consider the action of a MAXNET with four neurons and inhibitory weights $\epsilon = 0.2$ when given the initial activations (input signals)

$$a_1(0) = 0.2 \quad a_2(0) = 0.4 \quad a_3(0) = 0.6 \quad a_4(0) = 0.8$$

The activations found as the net iterates are

$$a_1(1) = 0.0 \quad a_2(1) = 0.08 \quad a_3(1) = 0.32 \quad a_4(1) = 0.56$$

$$a_1(2) = 0.0 \quad a_2(2) = 0.0 \quad a_3(2) = 0.192 \quad a_4(2) = 0.48$$

$$a_1(3) = 0.0 \quad a_2(3) = 0.0 \quad a_3(3) = 0.096 \quad a_4(3) = 0.442$$

$$a_1(4) = 0.0 \quad a_2(4) = 0.0 \quad a_3(4) = 0.008 \quad a_4(4) = 0.422$$

$$a_1(5) = 0.0 \quad a_2(5) = 0.0 \quad a_3(5) = 0.0 \quad a_4(5) = 0.421$$

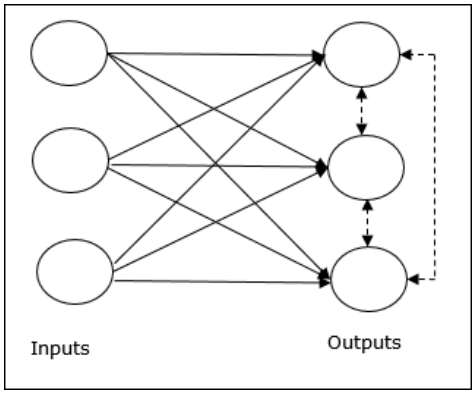
Although we have shown the activations as a function of the iteration, it is not necessary in general to save all of the previous values; only the activations from the previous step are actually needed, as shown in the algorithm.

Competitive Learning in ANN

It is concerned with unsupervised training in which the output nodes try to compete with each other to represent the input pattern. To understand this learning rule we will have to understand competitive net which is explained as follows –

Basic Concept of Competitive Network

This network is just like a single layer feed-forward network having feedback connection between the outputs. The connections between the outputs are inhibitory type, which is shown by dotted lines, which means the competitors never support themselves.



Basic Concept of Competitive Learning Rule

As said earlier, there would be competition among the output nodes so the main concept is - during training, the output unit that has the highest activation to a given input pattern, will be declared the winner. This rule is also called Winner-takes-all because only the winning neuron is updated and the rest of the neurons are left unchanged.

Mathematical Formulation

Following are the three important factors for mathematical formulation of this learning rule –

- Condition to be a winner

Suppose if a neuron $\boldsymbol{y}_k$ wants to be the winner, then there would be the following condition

$$y_k = \begin{cases} 1 & if\ v_k > v_j\ for\ all\ j, j \neq k \\ 0 & otherwise \end{cases}$$

It means that if any neuron, say, $\boldsymbol{y}_k$ wants to win, then its induced local field the output of the summation unit, say v_k, must be the largest among all the other neurons in the network.

- Condition of the sum total of weight

Another constraint over the competitive learning rule is the sum total of weights to a particular output neuron is going to be 1. For example, if we consider neuron k then

$$\sum_k w_{kj} = 1 \quad for\ all\ k$$

- Change of weight for the winner

If a neuron does not respond to the input pattern, then no learning takes place in that neuron. However, if a particular neuron wins, then the corresponding weights are adjusted as follows –

$$\Delta w_{kj} = \begin{cases} -\alpha(x_j - w_{kj}), if\ neuron\ k\ wins \\ 0 \qquad\qquad if\ neuron\ k\ losses \end{cases}$$

Here α is the learning rate.

This clearly shows that we are favoring the winning neuron by adjusting its weight and if a neuron is lost, then we need not bother to re-adjust its weight.

1. Vector Quantization

Vector quantization is used in many applications such as image and voice compression, voice recognition (in general statistical pattern recognition)

A vector quantizer maps k-dimensional vectors in the vector space R^K into a finite set of vectors $Y = \{y_i: i = 1, 2, \ldots, N\}$. Each vector y_i is called a code vector or a codeword. and the set of all the codewords is called a codebook. Associated with each codeword, y_i, is a nearest neighbor region called Voronoi region, and it is defined by:

$$V_i = \left\{x \in R^K: \|x - y_i\| \leq \left\|x - y_j\right\|, for\ all\ j \neq i\right\}$$

The set of Voronoi regions partition the entire space R^K such that:

$$\bigcup_{i=1}^{N} V_i = R^K$$

$$\bigcap_{i=1}^{N} V_i = \emptyset, \qquad for\ all\ i \neq j$$

As an example we take vectors in the two dimensional case without loss of generality. Figure 2 shows some vectors in space. Associated with each cluster of vectors is a representative codeword. Each codeword resides in its own Voronoi region. These regions are separated with imaginary lines in figure 2 for illustration. Given an input vector, the codeword that is chosen to represent it is the one in the same Voronoi region.

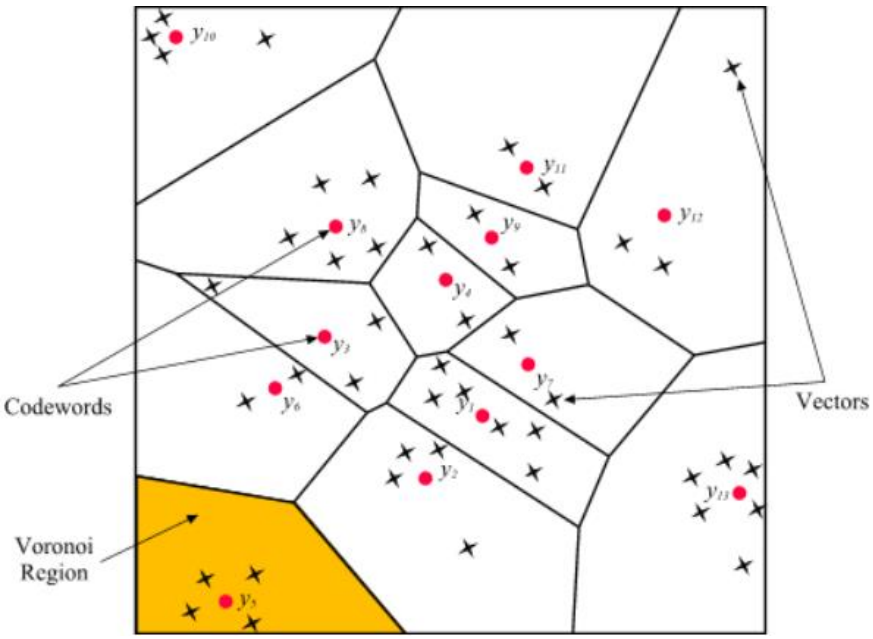


Figure 2: Codewords in 2-dimensional space. Input vectors are marked with an x, codewords are marked with red circles, and the Voronoi regions are separated with boundary lines.

The representative codeword is determined to be the closest in Euclidean distance from the input vector. The Euclidean distance is defined by:

$$d(x, y_i) = \sqrt{\sum_{j=1}^{k} (x_j - y_{ij})^2}$$

where x_j is the jth component of the input vector, and y_{ij} is the jth is component of the codeword y_i.

2. Kohonen Self-Organizing Feature Maps (SOM)

It has been known for over hundred years that various cortical areas of the brain are specialized to different modalities of cognitive functions. However, it was not until, e.g., Mountcastle (1957) as well as Hubel and Wiesel (1962) found that certain single neural cells in the brain respond selectively to some specific sensory stimuli. These cells often form local assemblies, in which their topographic location corresponds to some feature value of a specific stimulus in an orderly fashion. Such systems of cells are called brain maps.

It was believed first that the brain maps are determined genetically, like the other bodily formations and organizations. It was not until many of these maps, at least their fine structures and feature scales were found to depend on sensory experiences and other occurrences. Studies of brain

maps that are strongly modified by experiences have been reported especially by Merzenich et al.(1983).

Among some theoretical biologists in the 1970s, e.g. Grossberg (1976), Nass and Cooper (1975), and Perez, Glass, and Shlaer (1975), the question arose whether feature-sensitive cells could be formed also in artificial systems automatically, by learning (i.e., adaptation to simulated sensory stimuli). However, already Malsburg (1973), and later Amari (1980) demonstrated that their topographic order may also ensue from the input data.

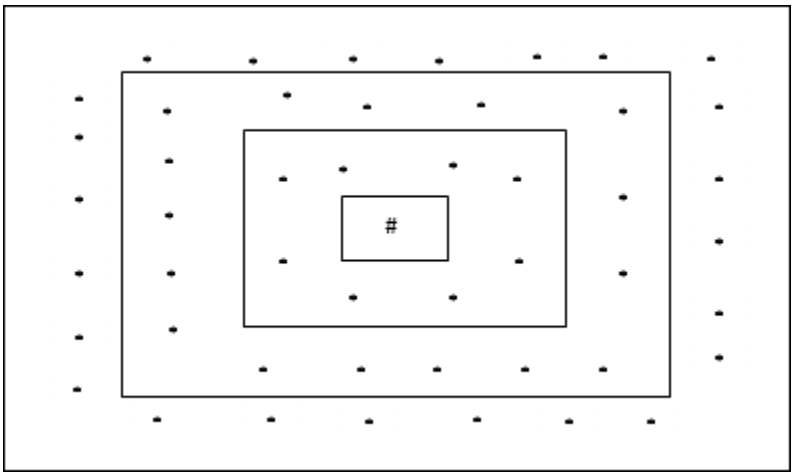
Suppose we have some pattern of arbitrary dimensions, however, we need them in one dimension or two dimensions. Then the process of feature mapping would be very useful to convert the wide pattern space into a typical feature space. Now, the question arises why do we require self-organizing feature map? The reason is, along with the capability to convert the arbitrary dimensions into 1-D or 2-D, it must also have the ability to preserve the neighbor topology.

Neighbor Topologies in Kohonen SOM

There can be various topologies, however the following two topologies are used the most:

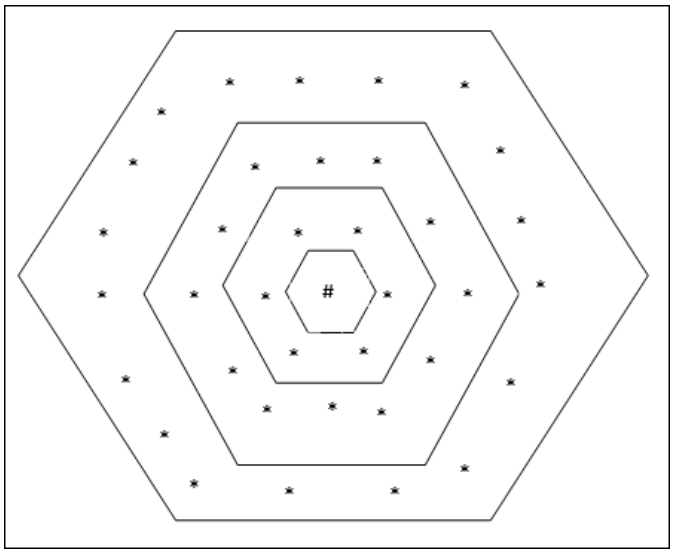
Rectangular Grid Topology

This topology has 24 nodes in the distance-2 grid, 16 nodes in the distance-1 grid, and 8 nodes in the distance-0 grid, which means the difference between each rectangular grid is 8 nodes. The winning unit is indicated by #.



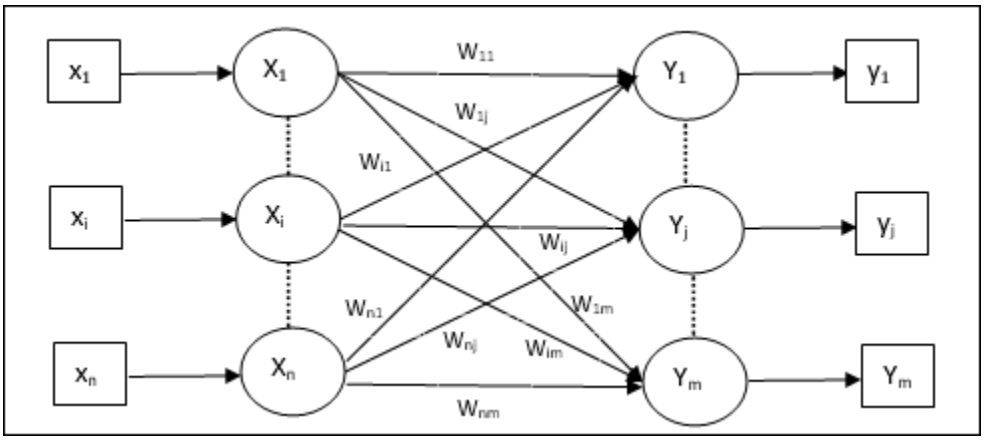
Hexagonal Grid Topology

This topology has 18 nodes in the distance-2 grid, 12 nodes in the distance-1 grid, and 6 nodes in the distance-0 grid, which means the difference between each rectangular grid is 6 nodes. The winning unit is indicated by #.



Architecture

The architecture of KSOM is similar to that of the competitive network. With the help of neighborhood schemes, discussed earlier, the training can take place over the extended region of the network.



Algorithm for training

Step 1 − Initialize the weights, the learning rate **α** and the neighborhood topological scheme.

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

Step 3 − Continue step 4-6 for every input vector **x**.

Step 4 − Calculate Square of Euclidean Distance for **j = 1 to m**

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

Step 5 − Obtain the winning unit **J** where $D(j)$ is minimum.

Step 6 − Calculate the new weight of the winning unit by the following relation

$$w_{ij}(new) = w_{ij}(old) + \propto \left[x_i - w_{ij}(old)\right]$$

Step 7 − Update the learning rate **α** by the following relation

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

Step 8 − Reduce the radius of topological scheme.

Step 9 − Check for the stopping condition for the network.

References and further reading:

Fausett, L. (1994). Fundamentals of neural networks: architectures, algorithms, and applications. Prentice-Hall, Inc..

http://www.mqasem.net/vectorquantization/vq.html [accessed in November 2019]

Kohonen, T. (2013). Essentials of the self-organizing map. Neural networks, 37, 52-65.

https://www.tutorialspoint.com/artificial_neural_network/artificial_neural_network_kohonen_selforganizing_feature_maps.htm [accessed in November 2019]