Artificial Neural Networks

Topic-06

Unsupervised Learning

Unsupervised Learning

- There exist problems where the training data consist of only input patterns and the desired output pairs are not available.
- Where the only information is provided by a set of input patterns.
- In these cases the relevant information has to be found within the redundant training samples .

Example Problems

- **Clustering:** Here the input data are to be grouped in to clusters and the data processing system has to find these inherent clusters pertaining to the input data. The output of the system should give the cluster label of the input pattern (discrete output)
- **Vector Quantization:** This problem occurs when a continuous space has to be discretized. The input of the system is the n-dimensional vector x. The output is a discrete representation of the input space, The system has to find optimal discretization of the input space.
- **Dimensionality Reduction:** The input data are grouped in a subspace which has lower dimensionality than the dimensionality of the data. The system has to learn an optimal mapping such that most of the variance in the input data is preserved in the output data.
- **Feature Extraction:** The system has to extract features from the input signal. This often means a dimensionality reduction as described above.

Neuro-Computational Solution

There is a special classes of ANN known as are self organizing networks that are suitable for solving these kind of problems. In these networks the training is done without the presence of an external teacher using unsupervised weight adapting algorithms. These algorithms are usually based on some form of global competition between the neurons.

- One of the most basic schemes is competitive learning as proposed by Rumelhart and Zipser
- A very similar network but with different emergent properties is the topology conserving map devised by Kohonen
- Other self-organising networks are ART proposed by Carpenter and Grossberg, and Fukushima's cognitron by Fukushima.

Competitive Learning Network

In competitive learning network all the output neurons are connected to every input neuron with associated interconnection weights. When an input pattern is presented , the output neurons of the network compete among themselves to be activated with the result only one output neuron is ***ON*** at any one time. The output neuron that wins the competition is called winner-takes all-neuron or simply a winning neuron.

There are two methods for the determination of the winner.

Winner selection Using *Dot Product*

Here both the input vectors and weight vectors are normalized to unit length.

Step-1: Each output unit calculates its activation according to the dot product of input and weight vector. (this nothing but the weighted sum)

Step-2: The output neuron *'k'* with maximum activation is selected as a winning neuron.

Step-3: Activation of the output neurons are reset such that the winning neuron will have the value of ***+1*** and other neurons will have the value of ***0*** for the given input.

Winner Selection Using *Euclidean Distance*

The dot product method would fail if un-normalised vectors were to be used.

Naturally one would like to accommodate the algorithm for un-normalized input data.

For this end winning neuron '*k*' is selected with its weight vector closest to the input pattern using the Euclidean distance measure.

Self Organizing Map

In a self-organizing map (SOM), the neurons are placed at the nodes of a lattice that is usually one or two dimensional.

The neurons are selectively tuned to various input patterns.

SOM is characterized by the formation of a topological map of input patterns in which the spatial location of a neurons in the lattice are indicative of intrinsic statistical feature contained in the input pattern. Hence the name self-organizing map.

Feature Mapping Models

Motivation

The development of self organizing map as neural model is motivated by distinct feature of the human brain: the brain is organized in many places in such a way that different sensory inputs are mapped on to corresponding areas in a cerebral cortex in a topologically ordered fashion. Thus the computational map constitutes a basic building block in the information processing infrastructure of the nervous system.

Properties of Brain Computational Map

- At each stage of representation each incoming piece of information is kept in its proper context.
- Neurons dealing with closely related pieces of information are close together sot hat they can interact via short synaptic connections.
- The principle of artificial topological map states that " the spatial location of an output neuron is topological map corresponds to a particular domain or feature of data drawn from the input space".

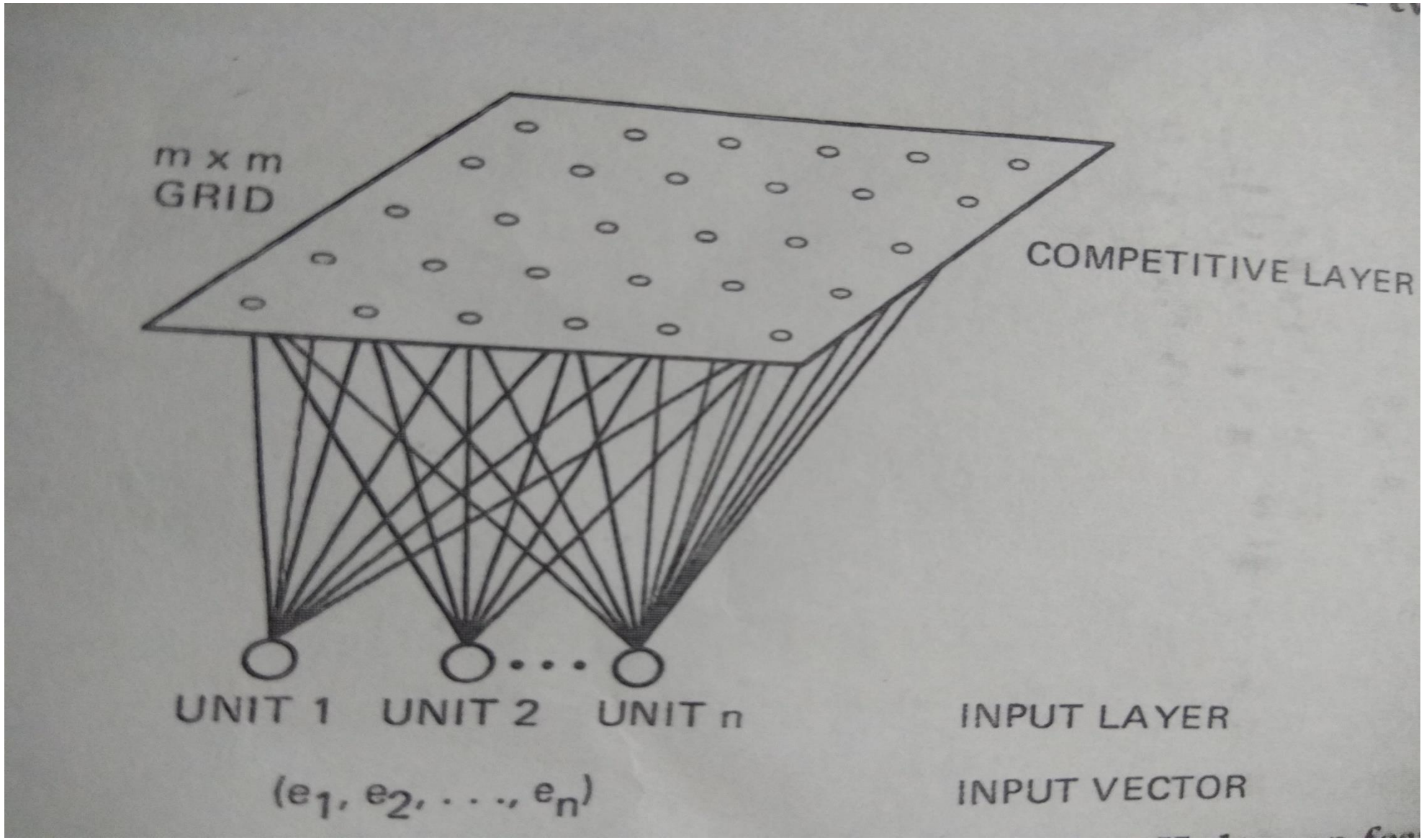
Kohonen Feature map

Kohanen's self-organizing feature map is a two layered network that can organize a topological map from random starting point. The resulting map shows the natural relationship among the pattern that are given to the network. This network combines the input layer with a competitive layer of processing unit and trained by unsupervised learning

The Kohonen feature map finds organization of relationship among patterns.

Basic Structure

- The Kohonen feature map is a two-layered network.
- The first layer of the network is the input layer.
- The second layer is the competitive layer and is organized as two-dimensional grid
- All connections go from the first layer to the second layer and these two layers are fully interconnected [each input neuron is connected to all the neurons in the competitive layer]
- Each interconnection has an associated weight value
- The typical initial weights are set by adding a small random number to the average entries in the input patterns.
- These weight values get updated during the training of the network.


m x m
GRID
COMPETITIVE LAYER
UNIT 1
UNIT 2
UNIT n
INPUT LAYER
$(e_1, e_2, \ldots, e_n)$
INPUT VECTOR

Training

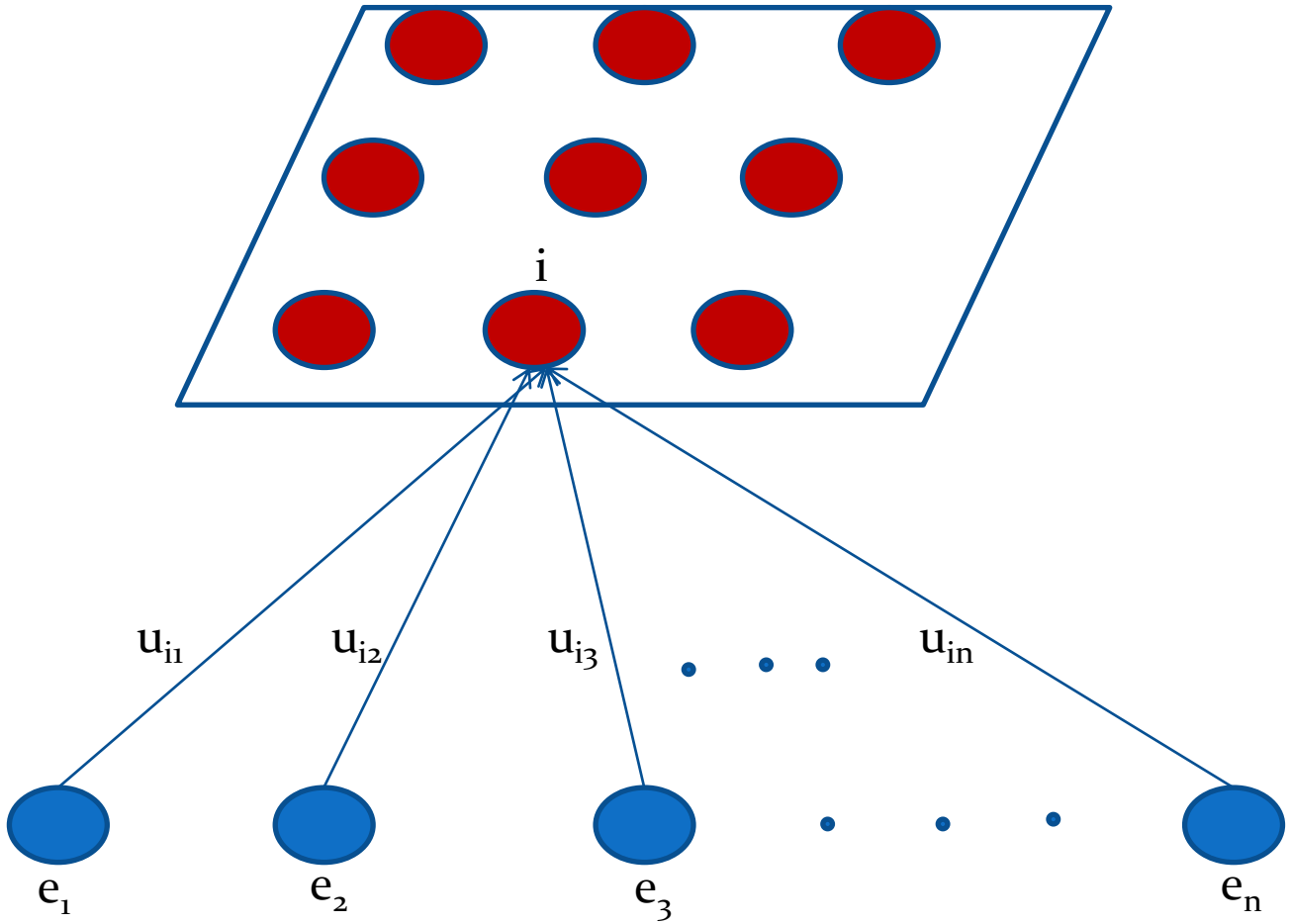
- When an input pattern is presented, each unit in the first layer takes on the value of the corresponding entry in the input pattern. This input pattern is denoted by

$$\boldsymbol{E} = [e_1, e_2, \ldots, e_n]$$

- The weights associated with connections from the input units to single unit (say i^{th} unit, here we identify each unit in the competitive layer by a single index, even though there is a two-dimensional grid of units in this layer) in the competitive layer are given by

$$\boldsymbol{U}_i = [\, u_{i1}, u_{i2}, \ldots, u_{in}]$$

Connections From the Input Vector to Single Unit in the Competitive Layer



Training Steps

The first step is to compute matching value for each unit in the competitive layer. This value measure the extent to which the weights each unit match the corresponding values of the input pattern. The matching value for the unit i is calculated by the following equation

$$\|E - U_i\| = \sqrt{\sum_j (e_j - u_{ij})^2}$$

which is the distance between the vectors E and U_i

The unit with lowest matching value(the best match) wins the competition. Hence the unit with the best match is denoted as unit k and k is chosen such that

$$\|E - U_k\| = \min_i \left(\|E - U_i\| \right)$$

where the minimum is taken over all units i in the competitive layer.

If two or more units have same matching value then by convention, the unit with lower index value i is chosen

The next step after identifying the winning unit is to identify the neighborhood around it. The neighborhood consist of the units that are close to winner in the competitive layer grid. Here the neighborhood consists of a set of units that are within a square that is centered around the winning unit ***k*** and denoted by $N_{k..}$ The of the neighborhood can vary depending up on the distance between ***k*** to the edge of the neighborhood and denoted by ***d.*** The weights are updated for all the units that are in the neighborhood of the winning unit $N_{k.}$ by using the equations

$$\Delta u_{ij} = \begin{cases} \alpha\left(e_j - u_{ij}\right) & if \quad unit \quad i \quad is \quad in \quad the \quad neighborhood \quad N_k \\ 0 & otherwise \end{cases}$$

$$u_{ij}^{new} = u_{ij}^{old} + \Delta u_{ij}$$

This adjustment results in the winning unit and its neighbors having their weights modified , becoming more like input pattern. Now the winner become more likely to win the competition when the same or a similar input pattern will be presented subsequently.

Here two parameter need to be specified:

- The value of α, the learning rate parameter in the weight adjustment equation
- The size of the neighborhood N_k

Setting the Value for α and N_c

- Initially the learning rate α takes relative larger value and being decreased over the span of many iterations.
- This initial value of α is set by choice and is denoted by α_0 (the typical choices are in the range0.2 to 0.5).
- The value of α for t^{th} is denoted by α_t determined by the equation

$$\alpha_t = \alpha_0\left(1 - \frac{t}{T}\right)$$

where t → is the current training iteration

T → is the total number of training iteration to be done,

The value begins with α_0 and decreases until it reaches zero

- The initial neighborhood width is relatively large, and is decreases over the progress of the training iterations. The center of the neighborhood is the winning unit k and at the position (x_k, y_k). Let d be the distance from k to the edge of the neighborhood. The neighborhood is then all (x, y) such that

  $k - d < x < k + d$ and $k - d < y < k + d$

- Sometimes the calculated neighborhood goes outside the grid of units in the competitive layer; in this case the actual neighborhood will be cut off at the edge.
- Initially d is assigned with a chosen value denoted by d_0 (the typical value for d_o can be a half or a third of the width of the competitive layer of processing units. The value of d will be decreased according to the equation

$$d = \left\lceil d_0 \left(1 - \frac{t}{T} \right) \right\rceil$$

where t $\rightarrow$ is the current training iteration and

T $\rightarrow$ is the total number of training iteration

This process assures a gradual linear decrease in $\boldsymbol{d}$ starting with $\boldsymbol{d_o}$ and going down to 1

Summary

1. Locate the unit in the competitive layer whose weights best match the input pattern.
2. Increase matching at this unit and its neighbors by adjusting their weights
3. Gradually decrease the size of the neighborhood and the amount of change to the weights as learning iteration progress