UNSUPERVISED LEARNING NEURAL NETWORKS

- Artificial neural networks have been developed to model the pattern association ability of the human brain.
- These networks are referred to as *associative memory* NNs.
- Associative memory NNs are usually two-layer NNs, where the objective is to adjust the weights such that the network can store a set of pattern associations – without any external help from a teacher.
- Unsupervised learning NNs are functions that map an input pattern to an associated target pattern.

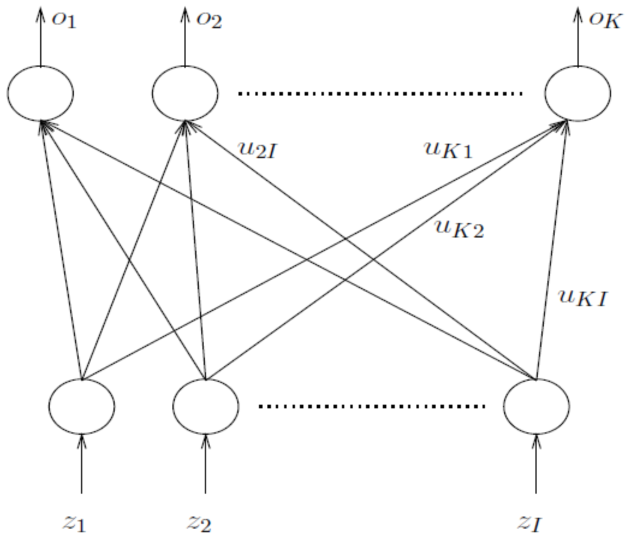


Figure 4.1 Unsupervised Neural Network

Hebbian Learning Rule

- With Hebbian learning, weight values are adjusted based on the correlation of neuron activation values.
- In such cases the weight between the two correlated neurons is strengthened (or increased).
- The change in weight at time step t is given as:

$$\Delta u_{ki}(t) = \eta o_{k,p} z_{i,p} \tag{4.2}$$

Weights are then updated using

$$u_{ki}(t) = u_{ki}(t-1) + \Delta u_{ki}(t) \tag{4.3}$$

where η is the learning rate.

- From equation (4.2), the adjustment of weight values is larger for those input-output pairs for which the input value has a greater effect on the output values.

- The Hebbian learning rule is summarized in Algorithm 4.1. The algorithm terminates when there is no significant change in weight values, or when a specified number of epochs has been exceeded.

Algorithm 4.1 Hebbian Learning Algorithm

Initialize all weights such that $u_{ki} = 0, \ \forall i = 1, \cdots, I$ and $\forall k = 1, \cdots, K$;
while *stopping condition(s) not true* **do**
  for *each input pattern* $\mathbf{z}_p$ **do**
    Compute the corresponding output vector $\mathbf{o}_p$;
  end
  Adjust the weights using equation (4.3);
end

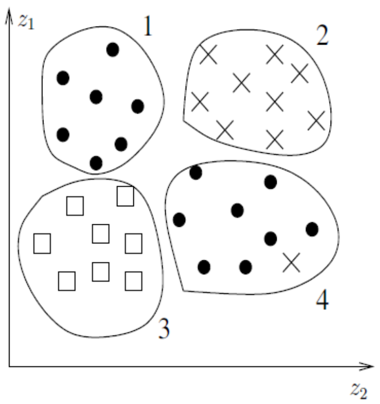
- A problem with Hebbian learning is that repeated presentation of input patterns leads to an exponential growth in weight values, driving the weights into ***saturation***.
- To prevent saturation, a limit is posed on the increase in weight values. One type of limit is to introduce a nonlinear ***forgetting factor***.

$$\Delta u_{ki}(t) = \eta o_{k,p} z_{i,p} - \gamma o_{k,p} u_{ki}(t-1)$$

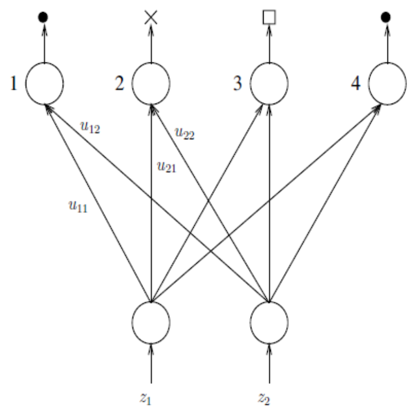
where γ is a positive constant.

Learning Vector Quantizer-I

- One of the most frequently used unsupervised clustering algorithms is the *learning vector quantizer* (LVQ) developed by Kohonen.
- Clustering algorithms divide a set on *n* observations into *m* groups such that members of the same group are more alike than members of different groups.
- The aim of a clustering algorithm is therefore to construct *clusters* of similar input vectors (patterns), where similarity is usually measured in terms of *Euclidean distance*.



(a) Clustering Problem



(b) LVQ-I network

- The training process of LVQ-I to construct clusters is based on *competition*.
- Each output unit *ok* represents a single cluster.
- The competition is among the cluster output units.

- During training, the cluster unit whose weight vector is the "closest" to the current input pattern is declared as the winner. The weight update is given as:

$$\Delta u_{ki}(t) = \begin{cases} \eta(t)[z_{i,p} - u_{ki}(t-1)] & \text{if } k \in \kappa_{k,p}(t) \\ 0 & \text{otherwise} \end{cases} \tag{4.19}$$

where $\eta(t)$ is a decaying learning rate, and $\kappa_{k,p}(t)$ is the set of neighbors of the winning cluster unit o_k for pattern p.

- The Kohonen LVQ-I algorithm is summarized in Algorithm 4.2.
- For the LVQ-I, weights are either initialized to random values, sampled from a uniform distribution, or by taking the first input patterns as the initial weight vectors.
- Stopping conditions may be
  1- a maximum number of epochs is reached.
  2- stop when weight adjustments are sufficiently small.
  3- a small enough quantization error has been reached, where the quantization error is defined as

$$\mathcal{Q}_T = \frac{\sum_{p=1}^{P_T} ||\mathbf{z}_p - \mathbf{u}_k||_2^2}{P_T}$$

Algorithm 4.2 Learning Vector Quantizer-I Training Algorithm

Initialize the network weights, the learning rate, and the neighborhood radius;
while *stopping condition(s) not true* **do**
  for *each pattern p* **do**
    Compute the Euclidean distance, $d_{k,p}$, between input vector $\mathbf{z}_p$ and each weight vector $\mathbf{u}_k = (u_{k1}, u_{k2}, \cdots, u_{KI})$ as

$$d_{k,p}(\mathbf{z}_p, \mathbf{u}_k) = \sqrt{\sum_{i=1}^{I}(z_{i,p} - u_{ki})^2} \tag{4.20}$$

    Find the output unit o_k for which the distance $d_{k,p}$ is the smallest;
    Update all the weights for the neighborhood $\kappa_{k,p}$ using equation (4.19);
  end
  Update the learning rate;
  Reduce the neighborhood radius at specified learning iterations;
end

Self-Organizing Feature Maps

- Kohonen developed the *self-organizing feature map* (SOM).
- The self-organizing feature map projects an *I*-dimensional input space to a discrete output space, effectively performing a compression of input space onto a set of codebook vectors.
- The output space is usually a two-dimensional grid.

Stochastic Training Rule

- SOM training is based on a competitive learning strategy.
- The first step of the training process is to define a map structure, usually a two-dimensional grid.
- The number of elements (neurons) in the map is less than the number of training patterns.

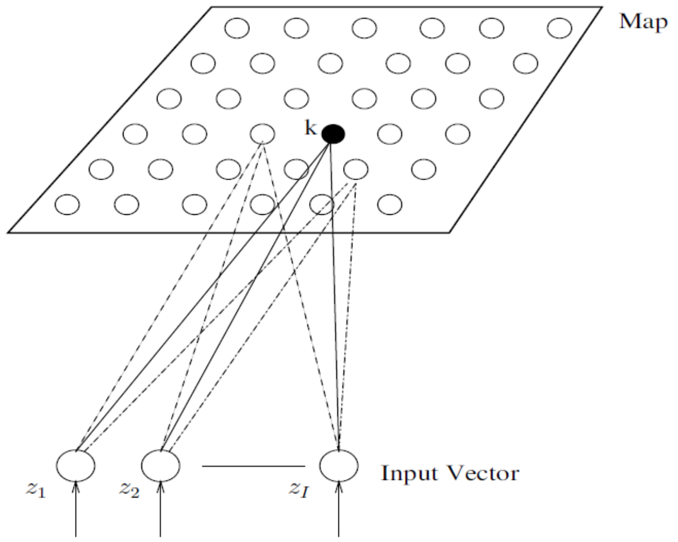


Figure 4.3 Self-organizing Map

- Each neuron on the map is associated with *I*-dimensional weight vector that formsthe centroid of one cluster.
- Initialization of the codebook vectors can occur in various ways:
  1- Random values. $w_{kj} = (w_{kj1}, w_{kj2}, \cdots, w_{KJI})$, with K the number of rows and J the number of columns of the map.
  2- Random input patterns. $\mathbf{w}_{kj} = \mathbf{z}_p$
- Codebook vectors are updated after each pattern is presented to the network. For each neuron, the associated codebook vector is updated as:

$$\mathbf{w}_{kj}(t+1) = \mathbf{w}_{kj}(t) + h_{mn,kj}(t)[\mathbf{z}_p - \mathbf{w}_{kj}(t)]$$

where*mn*is the row and column index of the winning neuron.

- The winning neuron is found by computing the Euclidean distance from each codebook vector to the input vector, and selecting the neuron closest to the input vector.
- The function $h_{mn,kj}(t)$ in equation is referred to as the neighborhood function. Thus, only those neurons within the neighborhood of the winning neuron *mn*have their codebook vectors updated.

- The neighborhood function is usually a function of the distance between the coordinates of the neurons as represented on the map. The smooth Gaussian kernel is mostly used to implement the neighborhood function.

$$h_{mn,kj}(t) = \eta(t)e^{-\frac{||c_{mn}-c_{kj}||_2^2}{2\sigma^2(t)}}$$

Where, $\eta(t)$ is the learning rate and $\sigma(t)$ is the width of the kernel. Both $\eta(t)$ and $\sigma(t)$ are monotonically decreasing functions. Thus, $h_{mn,kj}(t) \to 0$ when $t \to \infty$.

- The learning process is iterative, continuing until a "good" enough map has been found. The quantization error is usually used as an indication of map accuracy:

$$\mathcal{E}_T = \sum_{p=1}^{P_T} ||\mathbf{z}_p - \mathbf{w}_{mn}(t)||_2^2$$

- Training stops when ε_T is sufficiently small.

Batch Map

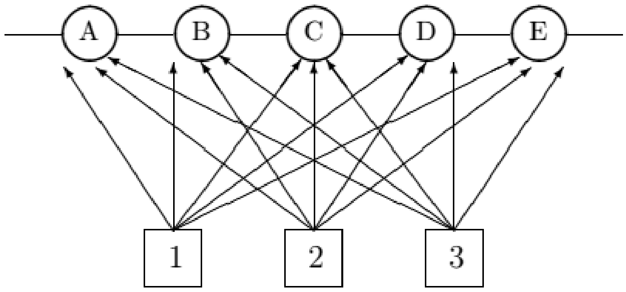
- The stochastic SOM training algorithm is slow due to the updates of weights after each pattern presentation: all the weights are updated.
- Batch versions of the SOM training rule have been developed that update weight values only after all patterns have been presented.

Algorithm 4.3 Batch Self-Organizing Map

```
Initialize the codebook vectors by assigning the first KJ training patterns to them,
where KJ is the total number of neurons in the map;
while stopping condition(s) not true do
    for each neuron, kj do
        Collect a list of copies of all patterns z_p whose nearest codebook vector
        belongs to the topological neighborhood of that neuron;
    end
    for each codebook vector do
        Compute the codebook vector as the mean over the corresponding list of
        patterns;
    end
end
```

Assignment

1- Below is a diagram of a self–organising map:



By looking at the diagram answer the following questions:

a) How many input nodes does this SOM have?

b) How many output nodes does this SOM have?

c) The input to an SOM can be represented by a point in an m–dimensionalspace (or m–dimensional vector). How many dimensions are in thespace that this SOM is analysing?

d) How many weights does each of the output nodes have?

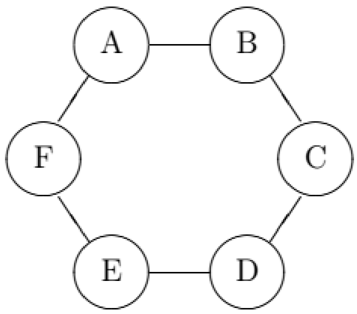
e) How many output nodes can fire simultaneously?

f) Is it important what value the output node sends when it fires?

g) Is there any limit on how many data points (input patterns) this SOMcan analyse?

h) How many clusters can this SOM detect in the input data?

2- Consider the following self–organising map:



The output layer of this map consists of six nodes, A, B, C, D, E and F, whichare organised into a two–dimensional lattice with neighbours connected bylines. Each of the output nodes has two inputs x1and x2(not shown on thediagram). Thus, each node has two weights corresponding to these inputs:w1and w2. The values of the weights for all output in the SOM nodes aregiven in the table below:

Node	A	B	C	D	E	F
w_1	-1	0	3	-2	3	4
w_2	2	4	-2	-3	2	-1

For an input pattern x = (x1, x2) the winner is determined using Euclideandistance:

$$d(\mathbf{x}, \mathbf{w}) = \sqrt{|x_1 - w_1|^2 + |x_2 - w_2|^2}$$

a- Calculate which of the six output nodes is the winner if the inputpattern is x = (2, −4) ?

b- After the winner for a given input x has been identified, the weightsof the nodes in SOM are adjusted using adaptation formula:

$$\mathbf{w}' = \mathbf{w} + \alpha\, h[\mathbf{x} - \mathbf{w}] \ ,$$

where w' is the new weight vector, α is the learning rate, h is the neighbourhood function. Let α = 0.5 and the neighbourhoodbe defined as:

$$h = \begin{cases} 1 & \text{if the node is the winner} \\ 0.5 & \text{if the node is immediate neighbour of the winner} \\ 0 & \text{otherwise} \end{cases}$$

Adjust the weights in the SOM.

3- What are the main similarities and differences between feed–forward neuralnetworks and self–organising maps?

4- Suppose that the SOM, shown in Question 1, is used to classify types ofairplanes based on three parameters: Size, speed and passenger load. Theweights of the output nodes are shown in the table below:

Node→	A	B	C	D	E
w_1	3	5	1	2	5
w_2	2	1	5	3	2
w_3	5	1	1	2	5

Each of the three parameters is assessed on a scale from 1 to 5. For example, small airplanes have size 1, while huge planes would have value 5. Each planeis represented as a three–dimensional vector with coordinates correspondingto these three parameters. Answer each of the following questions justifyingyour answers:

a- How many types of planes can this SOM classify?

b- Which node will be the winner, if a vector (1, 5,1) isfed into the input?

c- Suppose you were asked to change the design of the SOM in order totake into account two additional parameters: Price and fuel consumption. What would you need to change in this SOM?