Unit-7
Self Organizing Maps

By: Arjun Saud
Asst. Prof.: CDCSIT, TU

Introduction of SOM

- A **self-organizing map** (**SOM**) is a type of ANN that is trained using unsupervised learning to produce a low-dimensional (typically two-dimensional) discretized representation of the input space of the training samples. This low dimensional representation can be viewed as a **map**.
- Therefore SOM is a method to do dimensionality reduction. Self-organizing maps differ from other artificial neural networks as they apply competitive learning as opposed to error-correction learning.

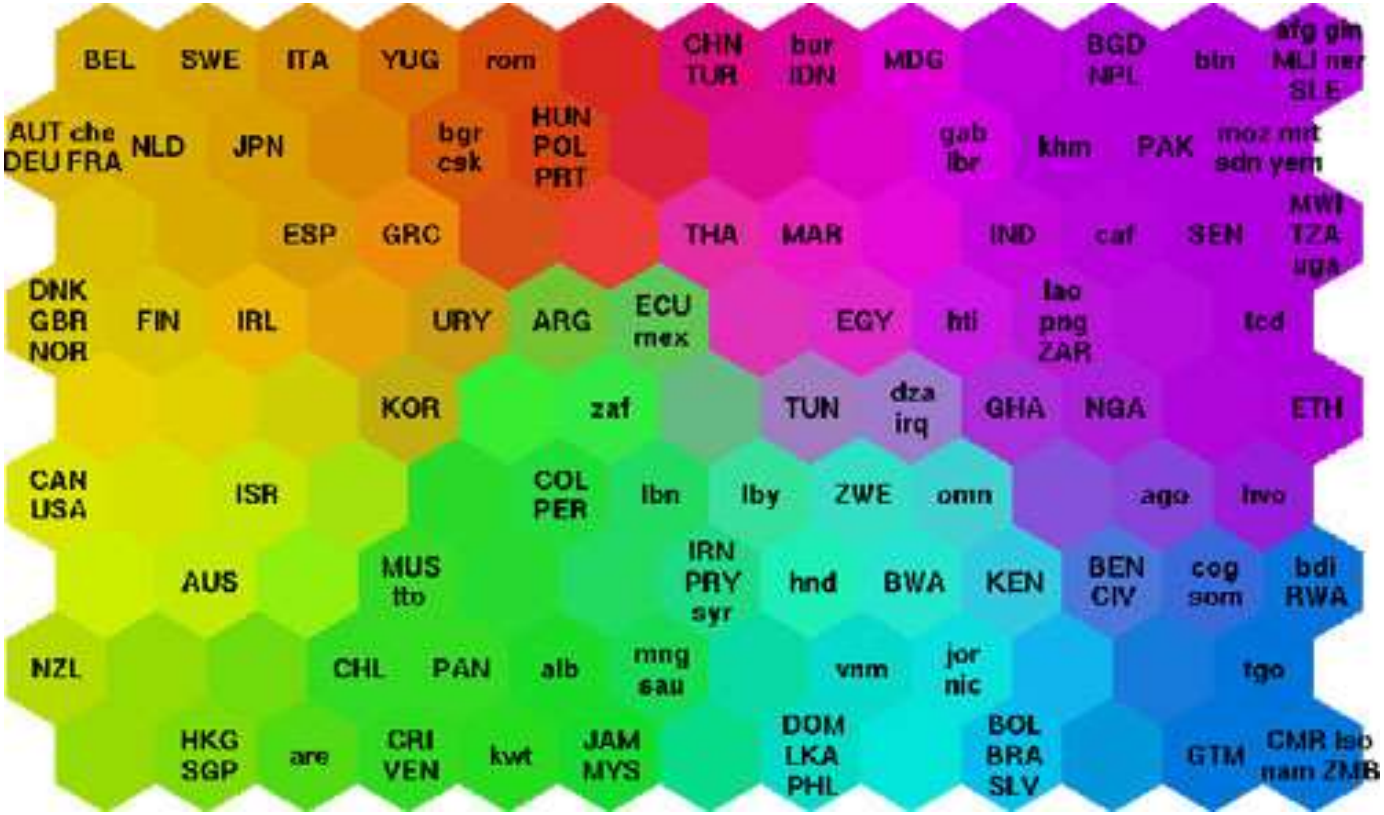
Introduction of SOM

- In SOM all inputs are fully connected with the output neurons and those neurons competes with each other. An output neuron that win the competition is called *winning neuron* or *winner takes all neuron*.
- Synaptic weights are adjusted in the favor of winning neuron so that when same or similar input pattern is presented to the neuron there will be high chance of winning the competition for the neuron.
- This means weights of winning neurons are updated such that Euclidean distance between the input and weights of the neuron is minimized.

Introduction of SOM

- In SOM, output neurons are organized in the form of one or two dimensional lattices. Higher dimensional lattices of neurons are also possible but are not practically common.
- SOMs are commonly used as visualization aids. They can make it easy for us humans to see relationships between vast amounts of data.
- **Example: World Poverty Map**
- A SOM has been used to classify statistical data describing various quality-of-life factors such as state of health, nutrition, educational services etc. Countries with similar quality-of-life factors end up clustered together.

Introduction of SOM

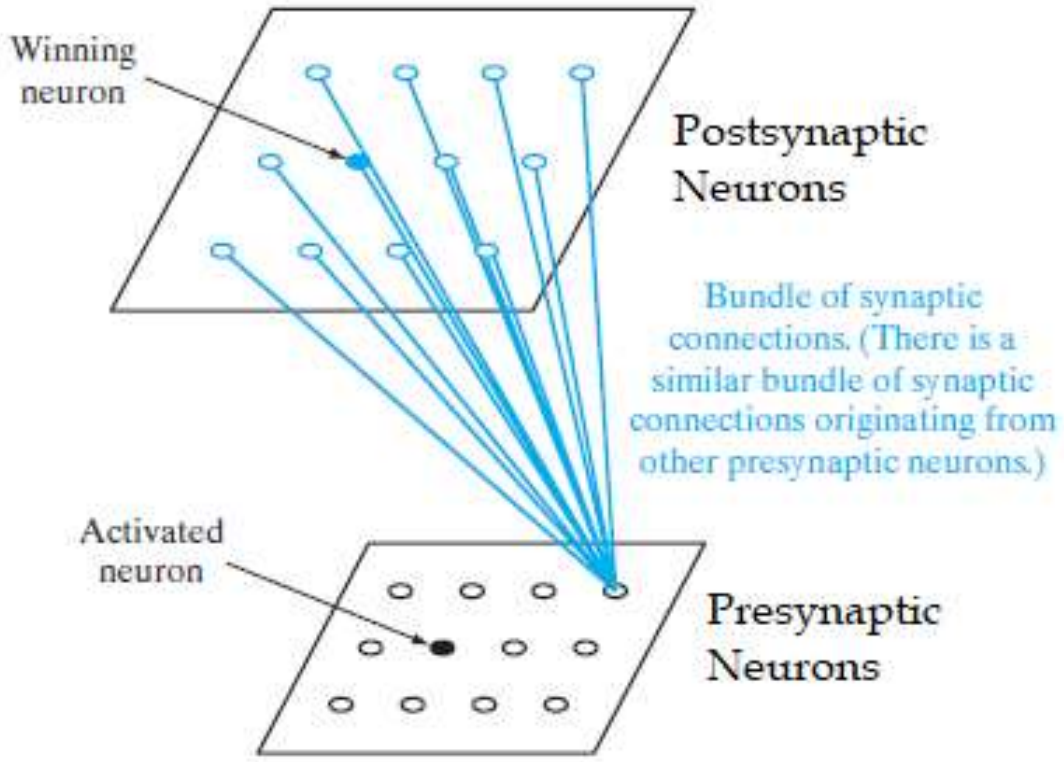


Two Basic Feature Mapping Models

- There are two different models for the self-organizing map:
  - Willshaw-von der Malsburg model;
  - Kohonen model.
- In both models the output neurons are placed in a 2D lattice. They differ in the way input is given:
  - In the Willshaw-von der Malsburg model the input is also a 2D lattice of equal number of neurons;
  - In the Kohonen model there isn't any input lattice, but an array of input neurons

Two Basic Feature Mapping Models

Willshaw-von der Malsburg Model



Two Basic Feature Mapping Models

Willshaw-von der Malsburg Model

- This is based the retino topic mapping from the retina to the visual cortex. It consists two layers of neurons with each input neuron fully connected to the output neurons layer.
- The postsynaptic lattice uses a *short-range excitatory mechanism* as well as a *long-range inhibitory mechanism*. The two lattices are interconnected by modifiable synapses of a Hebbian type.
- The postsynaptic neurons are not winner-takes-all neurons; rather, a threshold is used to ensure that only a few postsynaptic neurons will fire at any one time.

Two Basic Feature Mapping Models

Willshaw-von der Malsburg Model

- Thus, for each neuron some synaptic weights increase, while others decrease.
- Electric signals of presynaptic neurons shown geometrical proximities. Therefore the basic idea of Willshaw–von der Malsburg model is use the geometric proximity of presynaptic neurons in the postsynaptic neurons.
- This model is specialized to mappings for which the input dimension is the same as the output dimension. Hence, this model is not widely applicable.

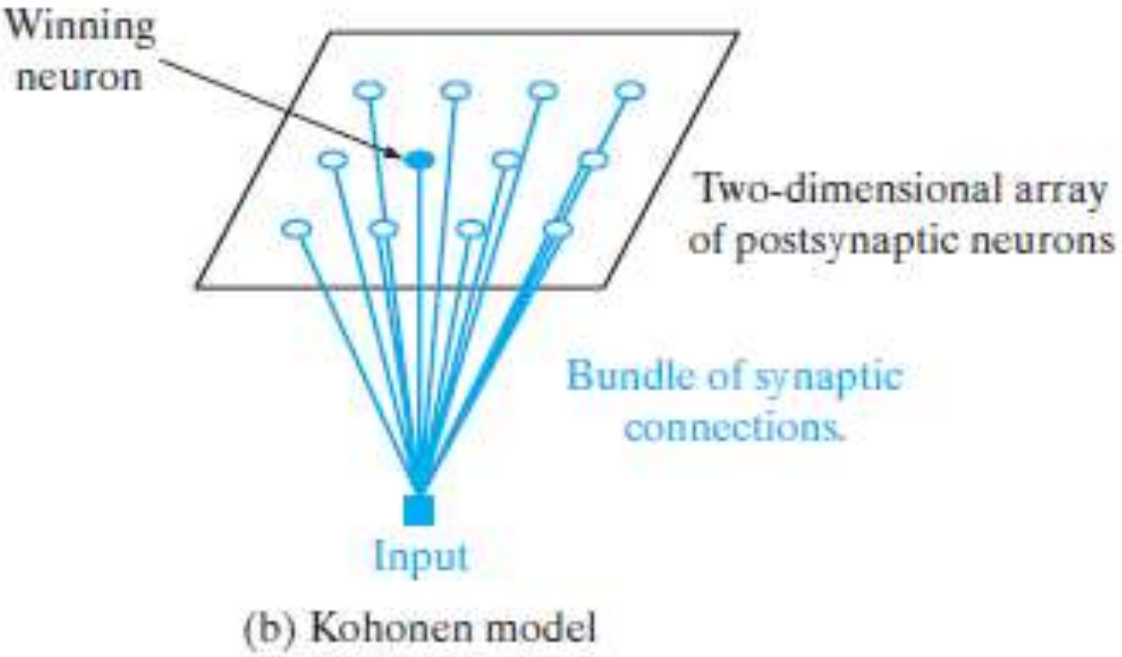
Two Basic Feature Mapping Models

Kohonen Model

- The Kohonen model is a more general version of the Willshaw-von der Malsburg model.
- It allows for compression of information. It belongs to a class of vector-coding algorithms. This means it provides a topological mapping that optimally places a fixed number of vectors into a higher-dimensional input space and thereby facilitates data compression (on the input).

Two Basic Feature Mapping Models

Kohonen Model



(b) Kohonen model

Self Organizing Maps

- The main goal of the SOM is to transform an incoming pattern of arbitrary dimension into a one- or two-dimensional discrete map, and to perform this transformation adaptively in a topologically ordered fashion.
- Each output neuron is fully connected to all the source nodes in the input layer.
- This network represents a feedforward structure with a single computational layer consisting of neurons arranged in a 2D or 1D grid.

Self Organizing Maps

- The algorithm which is responsible for the self-organization of the network is based on three essential steps:
- **Competition:** For each input pattern, the neurons in the network compute their respective values of a discriminant. The neuron with the largest value of discriminant function is declared winner of the competition.
- **Cooperation:** The winning neuron determines the spatial location of a topological neighborhood of excited neurons. This is definitely cooperation because exited neuron not only strengthen itself but also it strengthens neighboring neurons.

Self Organizing Maps

- **Synaptic Adaptation**: This mechanism enables the excited neurons to increase their individual values of the discriminant function in relation to the input pattern through suitable adjustments applied to their synaptic weights.
- Let us examine mathematical modeling of each of above mechanism used in self organizing maps.

Self Organizing Maps

Competition

- Let m be the dimension of the input space. A pattern chosen randomly from input space is denoted by:

  $\mathbf{x}=[x_1, x_2,\ldots, x_m]^T$

- The synaptic weight of each neuron in the output layer has the same dimension as the input space. We denote the weight of neuron j as:

  $\mathbf{w}_j=[w_{j1}, w_{j2},\ldots, w_{jm}]^T$, $j=1,2,\ldots,l$

  Where l is the total number of neurons in the output layer.

Self Organizing Maps

Competition

- To find the best match of the input vector **x** with the synaptic weights $\mathbf{w}_j$ we use the Euclidean distance. The neuron with the smallest distance is called i(**x**) and is given by:

$$i(\mathbf{x}) = \arg\min_j ||\mathbf{x} - \mathbf{w}_j||, \quad j = 1,2,\ldots,l$$

- The neuron (i) that satisfies the above condition is called *best-matching* or *winning neuron* for the input vector **x**.
- The above equation leads to the following observation: *A continuous input space of activation patterns is mapped onto a discrete output space of neurons by a process of competition among the neurons in the network.*

Self Organizing Maps

Cooperation

- The winning neuron effectively locates the center of a *topological neighborhood*.
- The neighborhood should be a decreasing function of the *lateral distance* between the neurons.
- Let d_{ij} is the lateral distance between neurons i and j and let us denote h_{ji} as the *topological neighborhood around neuron i*.

Self Organizing Maps

Cooperation

- Thus h_{ji} must be *function of distance* which satisfies the following two requirements:
  - *The topological neighborhood h_{ji} is symmetric about the maximum point defined by d_{ij}=0; in other words, it attains its maximum value at the winning neuron i for which the distance is zero.*
  - *The amplitude of the topological neighborhood h_{ji} decreases monotonically with increasing lateral distance d_{ij} decaying to zero for $d_{ij} \rightarrow \infty$; this is a necessary condition for convergence.*

Self Organizing Maps

Cooperation

- A typical choice of h_{ji} is the Gaussian function which is given by:

$$h_{ji(\vec{x})} = \exp\left(-\frac{d_{ij}^{2}}{2\sigma^{2}}\right)$$

The parameter σ is the width of the neighborhood.

- Another characteristic feature of the SOM algorithm is that the size of the neighborhood *shrinks* with time. This requirement is satisfied by making the width of the Gaussian function decreasing with time.

Self Organizing Maps

Cooperation

- A popular choice is the exponential decay described by:

$$\sigma(n) = \sigma_0 \exp(-\frac{n}{\tau_1}) \quad n = 0,1,2,...$$

Where σ_0 is the value of σ at the initialization of the SOM algorithm and τ_1 is a *time (iteration) constant*.

- Correspondingly the neighborhood function assumes a time dependent form of its own:

$$h_{ji(\vec{x})}(n) = \exp(-\frac{d_{ji}^{\ 2}}{2\sigma(n)^2}) \quad n = 0,1,2,...$$

Self Organizing Maps

Adaptation

- The adaptive process modifies the weights of the network so as to achieve the self-organization of the network.
- Only the winning neuron and neurons inside its neighborhood have their weights adapted. All the other neurons have no change in their weights.
- A method for deriving the weight update equations for the SOM model is based on a modified form of Hebbian learning. There is a forgetting term in the standard Hebbian weight equations.

Self Organizing Maps

Adaptation

- Let us assume that the *forgetting term* has the form $g(y_j)\mathbf{w}_j$ where y_j is the response of neuron j and $g(\bullet)$ is a positive scalar function of y_j.
- The only requirement for the function $g(y_j)$ is that the constant term in its Taylor series expansion to be zero when the activity is zero, i.e.: $g(y_j)=0$ for $y_j=0$.
- The modified Hebbian rule for the weights of the output neurons is given by:

$$\Delta\mathbf{w}_j = \alpha y_j \mathbf{x} - g(y_j)\,\mathbf{w}_j$$

Where α is the *learning rate parameter* of the algorithm.

Self Organizing Maps

Adaptation

- To satisfy the requirement for a zero constant term in the Taylor series we choose the following form for the function $g(y_j)$:

  $$g(y_j) = \alpha\, y_j$$

- We can simplify further by setting:

  $$y_j = h_{ji(x)}$$

- Combining the previous equations we get:

  $$\Delta \mathbf{w}_j = \alpha\, h_{ji(x)}\, (\mathbf{x} - \mathbf{w}_j)$$

- Finally using a discrete representation for time we can write:

  $$\mathbf{w}_j(n+1) = \mathbf{w}_j(n) + \alpha(n)\, h_{ji(x)}(n)\, (\mathbf{x} - \mathbf{w}_j(n))$$

Self Organizing Maps

Adaptation

- The learning rate must also be time varying. A suitable form is given by:

$$\alpha(n) = \alpha_0 \exp(-\frac{n}{\tau_2}) \quad n = 0,1,2,\ldots$$

Where η_0 is an initial value and τ_2 is another time constant of the SOM algorithm.

Self Organizing Maps

Summarized Algorithm

1. *Initialization:* Choose random values for the initial weight vectors $\boldsymbol{w}_j(0)$. The weight vectors must be different for all neurons. Usually we keep the magnitude of the weights small.
2. *Sampling:* Draw a sample $\boldsymbol{x}$ from the input space with a certain probability; the vector $\boldsymbol{x}$ represents the activation pattern that is applied to the lattice. The dimension of $\mathbf{x}$ is equal to m.
3. *Similarity Matching:* Find the best-matching (winning) neuron $i(\boldsymbol{x})$ at time step n by using the minimum Euclidean distance criterion.

Self Organizing Maps

Summarized Algorithm

$$i(\boldsymbol{x}) = arg\ min_j\ ||\boldsymbol{x} - \boldsymbol{w}_j||,\ j=1,2,\ldots,l$$

4. *Updating:* Adjust the synaptic weight vectors of all neurons by using the update formula:

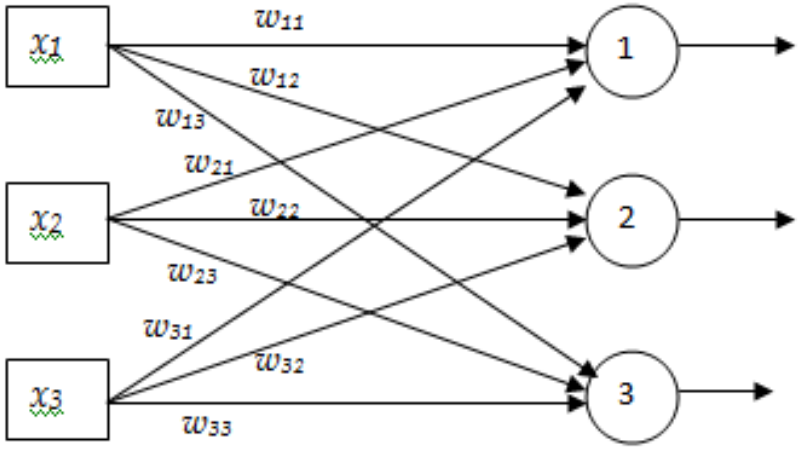
$$\boldsymbol{w}_j(n+1) = \boldsymbol{w}_j(n) + \alpha(n)\, h_{ji(x)}(n)\, (\boldsymbol{x}(n) - \boldsymbol{w}_j))$$

5. *Continuation:* Continue with step 2 until no noticeable changes in the feature map are observed.

Self Organizing Maps

Example

Consider following 1-D SOM and 3-D inputs. Show the working of SOM for the given two inputs.



x_1	x_2	x_3
0.1	0.2	0.13
0.5	0.3	0.7

Initial weight matrix

w_{11}=0.1	w_{12}=0.2	w_{13}=0.3
w_{21}=0.2	w_{22}=0.4	w_{23}=0.6
w_{31}=0.1	w_{32}=0.3	w_{33}=0.5

Self Organizing Maps

Solution

Iteration 1: input (0.1,0.2,0.13)

Find Euclidean distance between the input and weight vector of each output neuron

$$d_1 = d(x, w_1) = \sqrt{(0.1-0.1)^2 + (0.2-0.2)^2 + (0.13-0.1)^2} = 0.03$$

$$d_2 = d(x, w_2) = \sqrt{(0.1-0.2)^2 + (0.2-0.4)^2 + (0.13-0.3)^2} = 0.28$$

$$d_3 = d(x, w_3) = \sqrt{(0.1-0.3)^2 + (0.2-0.6)^2 + (0.13-0.5)^2} = 0.58$$

Clearly neuron 1 is winner

Self Organizing Maps

Update weights: Assume $\alpha = 1$ $\sigma = 1$

We know that

$$w_j(n+1) = w_j(n) + \alpha h_{ji}(x - w_j(n)) \qquad h_{ji(\vec{x})} = \exp(-\frac{d_{ji}^{\ 2}}{2\sigma^2})$$

=>

$$h_{11} = 1$$

$$w_{11} = w_{11} + 1*1*(0.1 - 0.1) = 0.1$$

$$w_{21} = w_{21} + 1*1*(0.2 - 0.2) = 0.2$$

$$w_{31} = w_{31} + 1*1*(0.13 - 0.1) = 0.13 + 0.03 = 0.16$$

Self Organizing Maps

Similarly

$$h_{21} = \exp\left(-\frac{1}{2}\right) = 0.6$$

$$w_{12} = w_{12} + 1 * 0.6 * (0.1 - 0.2) = 0.2 - 0.06 = 0.14$$

$$w_{22} = w_{22} + 1 * 0.6 * (0.2 - 0.4) = 0.4 - 0.12 = 0.28$$

$$w_{32} = w_{32} + 1 * 0.6 * (0.13 - 0.3) = 0.6 - 0.102 = 0.498$$

$$h_{31} = \exp\left(-\frac{4}{2}\right) = 0.135$$

$$w_{13} = w_{13} + 1 * 0.135 * (0.1 - 0.3) = 0.3 - ? = ?$$

$$w_{23} = w_{23} + 1 * 0.37 * (0.2 - 0.6) = 0.6 - ? = ?$$

$$w_{33} = w_{33} + 1 * 0.37 * (0.13 - 0.5) = 0.5 - ? = ?$$

Self Organizing Maps

Iteration 2: input (0.5,0.3,0.7)

Find Euclidean distance between the input and weight vector of each output neuron

$$d_1 = d(x, w_1) = \sqrt{?} = ?$$

$$d_2 = d(x, w_2) = \sqrt{?} = ?$$

$$d_3 = d(x, w_3) = \sqrt{?} = ?$$

Clearly neuron ? is winner

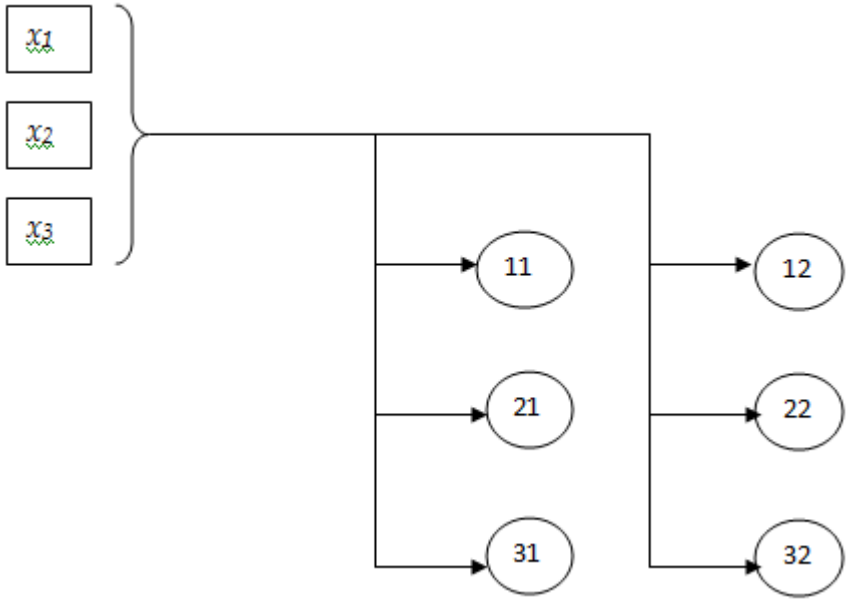
Self Organizing Maps

Update weights

Self Organizing Maps

Example

Consider following 2-D SOM and 3-D inputs. Show the working of SOM for the inputs (0.25,0.15,0.55)



Initial weight matrix

0.1	0.2	0.4	0.2	0.1	0.2
0.4	0.3	0.2	0.5	0.7	0.6
0.7	0.5	0.6	0.3	0.4	0.3

Self Organizing Maps

Properties of Feature Map

- **Approximation of the Input Space:** The feature map Φ, represented by the set of synaptic weight vectors $\{\mathbf{w}_j\}$ in the output space A, provides a good approximation to the input space H.
- **Topological Ordering:** The feature map Φ computed by the SOM algorithm is topologically ordered in the sense that the spatial location of a neuron in the lattice corresponds to a particular domain or feature of the input patterns.

Self Organizing Maps

Properties of Feature Map

- **Density Matching:** The feature map Φ reflects variations in the statistics of the input distribution: regions in the input space H from which sample vectors **x** are drawn with a high probability of occurrence are mapped onto larger domains of the output space A, and therefore with better resolution than regions in H from which sample vectors **x** are drawn with a low probability of occurrence.
- **Feature Selection:** Given data from an input space with a nonlinear distribution, the self-organizing map is able to select a set of best features for approximating the underlying distribution.

Contextual Maps

- It is in the vary nature of a symbol that its meaning is dissociated from its encoding.
- Hence logical relatedness between different symbols will in general not be directly detectable from their encodings and one may thus not presume any metric relations between the symbols, even when they represent similar items.
- Thus to create topographic map of symbols, they must frequently be presented in due context i.e. in conjunction with all or part of the attribute values of the item it encodes

Contextual Maps

- The simplest system model for symbol maps assumes each data vector x as a concatenation of two fields, one specifying the symbol code, denoted by x_s, and the other the attribute set, denoted x_a, respectively.

$$\mathbf{x} = \begin{bmatrix} \mathbf{x}_s \\ \mathbf{x}_a \end{bmatrix} = \begin{bmatrix} \mathbf{x}_s \\ \mathbf{0} \end{bmatrix} + \begin{bmatrix} \mathbf{0} \\ \mathbf{x}_a \end{bmatrix}$$

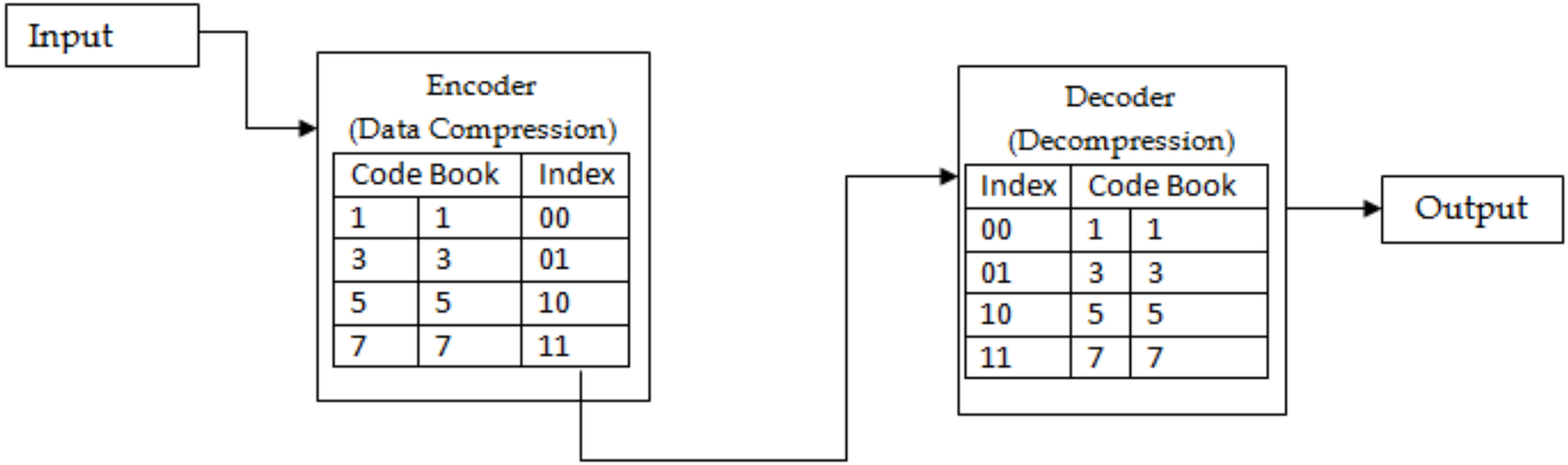
- Topographical maps of symbols created using SOM with above technique is called contextual map.

Vector Quantization

- Vector quantization (VQ) is a lossy data compression technique.
- In VQ, a codebook is used to approximate each input vector.
- Every input vector is assigned to the nearest vector in the codebook.
- The quantization process can be decomposed into two operations: *Encoder and Decoder*.
- Encoder maps every input vector with some index and decoder maps every index with nearest code vector of the input vector.

Vector Quantization

- Example: Compress the input {(1,5),(2,3),(4,3),(4,1),(6,3), (5,4), (2,3), (2,4)} using codebook {(1,1),(3,3),(5,5), (7,7)}



Vector Quantization

Lets take input x=(1,5)

$$d(x,c_1) = \sqrt{(1-1)^2 + (5-1)^2} = 4$$

$$d(x,c_2) = \sqrt{(1-3)^2 + (5-3)^2} = 2.83$$

$$d(x,c_3) = \sqrt{(1-5)^2 + (5-5)^2} = 4$$

$$d(x,c_4) = \sqrt{(1-7)^2 + (5-7)^2} = 6.32$$

Thus, (1,5) is matched with (3,3)=> code for (1,5) is 01

Similarly, other inputs can be quantized

Vector Quantization

Without VQ

- We need 3 bits to represent 8 inputs
- Thus, total number of bits=8 x 3=24 bits

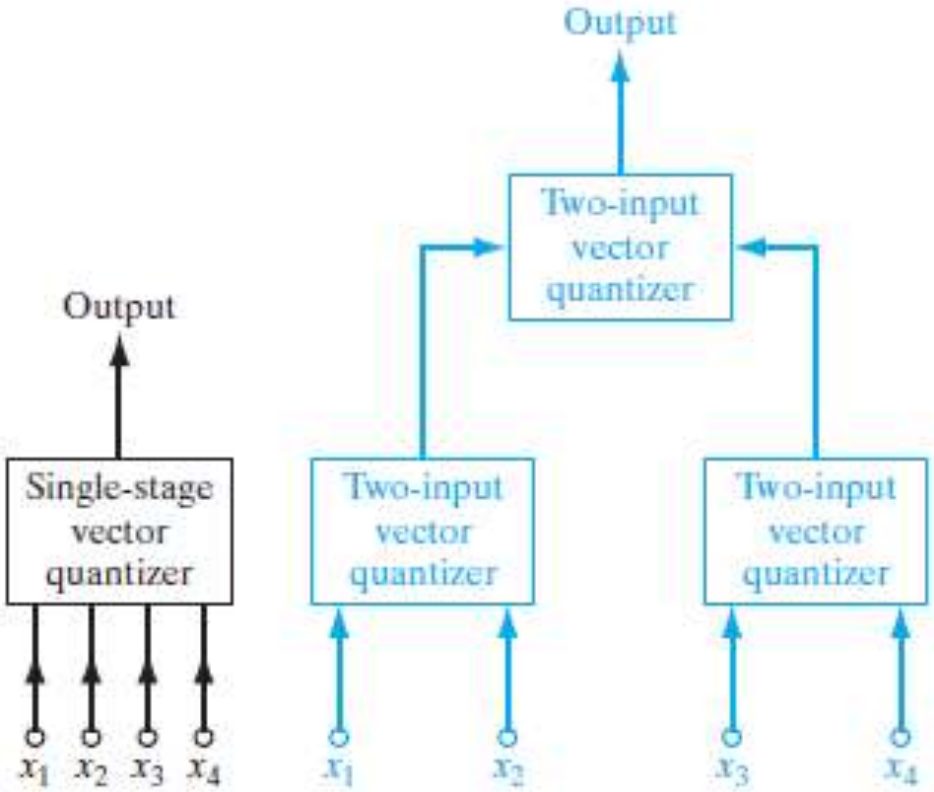
After VQ:

- 2 bits are needed to represent 4 code vectors
- Thus, total number of bits=8 x 2=16 bits

Vector Quantization

- The main problem with conventional vector quantization is that its encoding operation is time consuming.
- For a code book containing N code vectors, for example, the time taken for encoding a single input is on the order of N.
- The multistage hierarchical vector quantizer attempts to factorize the overall vector quantization into a number of sub-operations, each of which requires very little computation.

Vector Quantization



Single-stage and two-stage vector quantizer

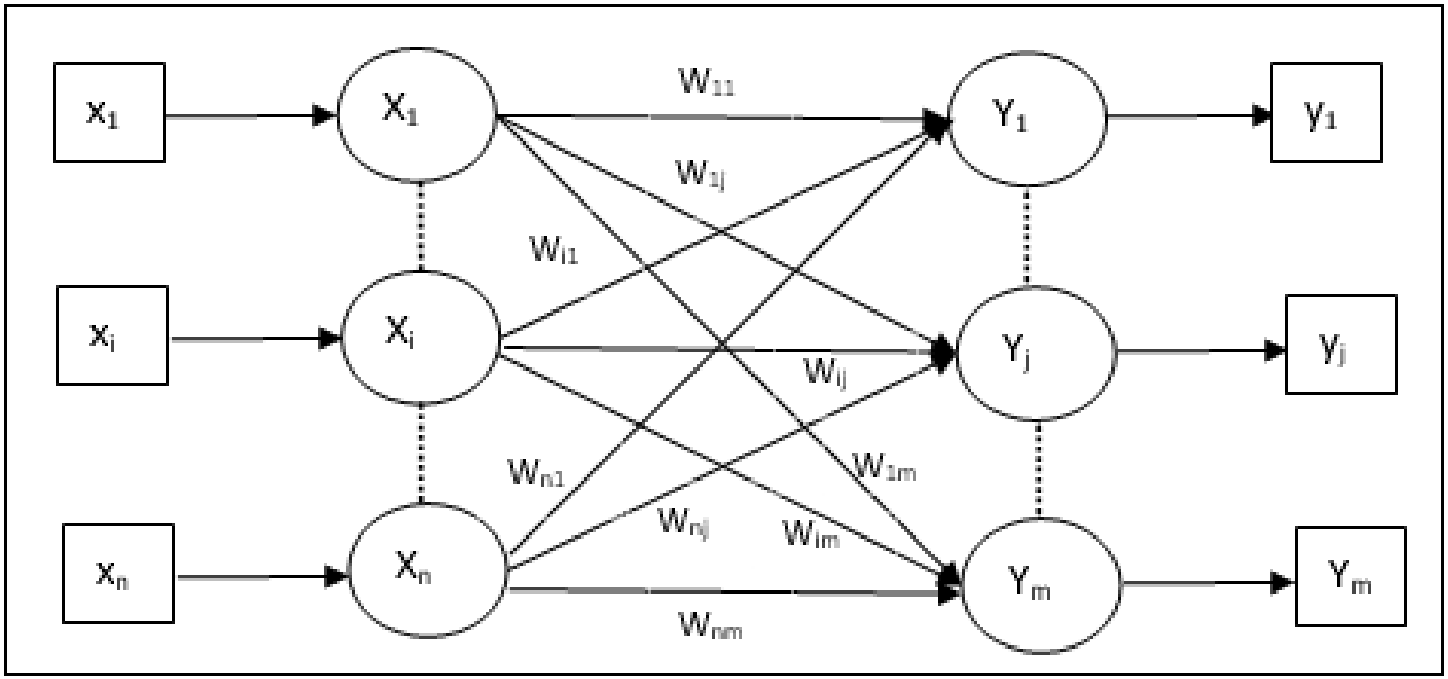
Vector Quantization

- We show a single-stage vector quantizer for x. Alternatively, we may use a two-stage hierarchical vector quantizer, as depicted in the second figure.
- The significant difference between these two schemes is that the input dimension of the quantizer in first figure is four, whereas for the quantizer in second figure it is two.
- Accordingly, the quantizer of second figure requires a lookup table of smaller size and is therefore simpler to implement than that of first figure. This is the advantage of a hierarchical quantizer over a conventional quantizer.
- SOM algorithm can be used to train vector quantizer at each level.

Learning Vector Quantization (LVQ)

- LVQ is prototype based supervised classification algorithm that uses competitive learning algorithm for training neural network.
- Training algorithm used by LVQ is similar to SOM except slight different in weight update.
- LVQ has two layers, one is the Input layer and the other one is the Output layer.
- The architecture of the Learning Vector Quantization with the m number of classes in an input data and n number of input features for any sample is given below:

Learning Vector Quantization (LVQ)



Learning Vector Quantization (LVQ)

Algorithm

1. From the given set of training vectors, take the first m (i.e number of clusters) training vectors and use them as weight vectors. The remaining vectors can be used for training.
2. For each training vector x
   - Calculate Euclidean distance between x and weight vector of each neuron

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

   - Find index of winning neuron (say, J) such that Euclidean distance is minimum.

Learning Vector Quantization (LVQ)

Algorithm contd..

3. Update weights of winning neuron as below.

$$If\ T = J \quad w_i = w_i + \alpha(x - w_i)$$

$$If\ T \neq J \quad w_i = w_i - \alpha(x - w_i)$$

4. Repeat steps 2-3 until stopping condition is satisfied.

Learning Vector Quantization (LVQ)

Example

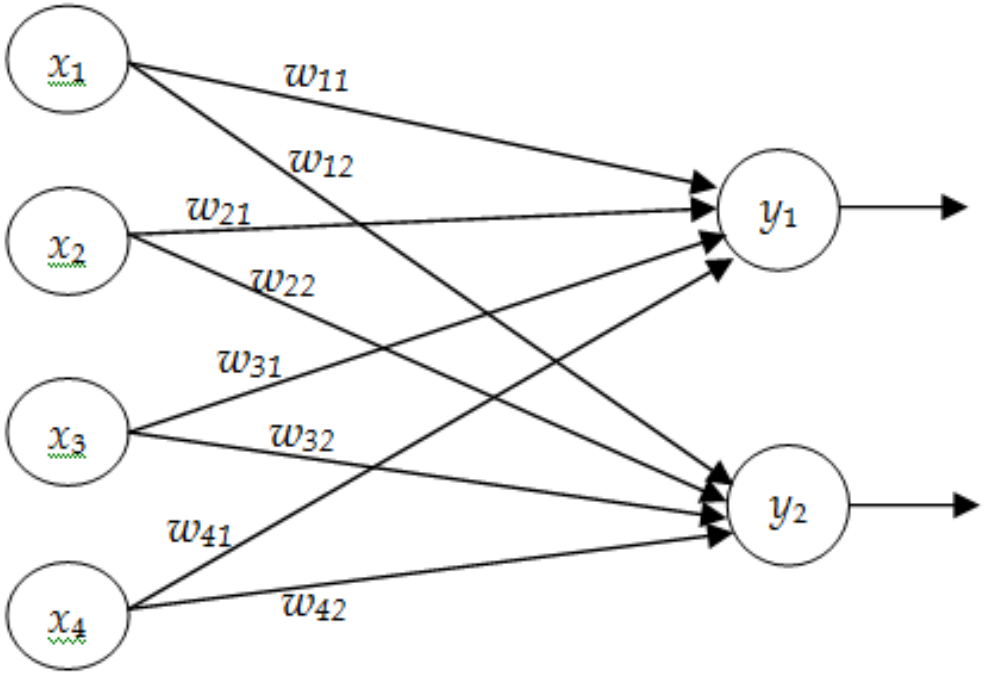
- Construct an LVQ net with five vectors assigned to two classes. Given vectors along with classes is shown below.

Vector	Class
[0 0 1 1]	1
[1 0 0 0]	2
[0 0 0 1]	2
[1 1 0 0]	1
[0 1 1 0]	1

Learning Vector Quantization (LVQ)

Solution

- Here every input has four features and two classes, so draw a network with four inputs and 2 outputs.



Learning Vector Quantization (LVQ)

Solution

Initialize weigh vector $w = \begin{bmatrix} 0 & 1 \\ 0 & 0 \\ 1 & 0 \\ 1 & 0 \end{bmatrix}$ let $\alpha = 0.1$

Take First Input vector x={0 0 0 1} T=2 (i.e class 2)

Calculate Euclidean distances:

$$D_1 = (0-0)^2 + (0-0)^2 + (0-1)^2 + (1-1)^2 = 1$$

$$D_2 = (0-1)^2 + (0-0)^2 + (0-0)^2 + (1-0)^2 = 2$$

Thus winning neuron is j=1 (i.e. y_1)

Learning Vector Quantization (LVQ)

Solution

Update weights (T≠J)

$$w_{11} = w_{11} - \alpha(x_1 - w_{11}) = 0 - 0.1(0 - 0) = 0$$

$$w_{21} = w_{21} - \alpha(x_2 - w_{21}) = 0 - 0.1(0 - 0) = 0$$

$$w_{31} = w_{31} - \alpha(x_3 - w_{31}) = 1 - 0.1(0 - 1) = 1.1$$

$$w_{41} = w_{41} - \alpha(x_4 - w_{41}) = 1 - 0.1(1 - 1) = 1$$

Thus, updated weight matrix is:

$$w = \begin{bmatrix} 0 & 1 \\ 0 & 0 \\ 1.1 & 0 \\ 1 & 0 \end{bmatrix}$$

Learning Vector Quantization (LVQ)

Take Second Input vector x={1 1 0 0} T=1

Calculate Euclidean distances:

$$D_1 = (1-0)^2 + (1-0)^2 + (0-1.1)^2 + (0-1)^2 = 4.21$$

$$D_2 = (1-1)^2 + (1-0)^2 + (0-0)^2 + (0-0)^2 = 1$$

Thus, winning neuron is: j=2 (i.e. y_2)

Update weights (T≠J)

$$w_{12} = w_{12} - \alpha(x_1 - w_{12}) = 1 - 0.1(1-1) = 1$$

$$w_{22} = w_{22} - \alpha(x_2 - w_{22}) = 0 - 0.1(1-0) = -0.1$$

$$w_{32} = w_{32} - \alpha(x_3 - w_{32}) = 0 - 0.1(0-0) = 0$$

$$w_{42} = w_{42} - \alpha(x_4 - w_{42}) = 0 - 0.1(0-0) = 0$$

Learning Vector Quantization (LVQ)

Thus, updated weight matrix is:

$$w = \begin{bmatrix} 0 & 1 \\ 0 & -0.1 \\ 1.1 & 0 \\ 1 & 0 \end{bmatrix}$$

Take Third Input vector x={0 1 1 0} T=1

Calculate Euclidean distances:

$$D_1 = (0-0)^2 + (1+0)^2 + (1-1.1)^2 + (0-1)^2 = 2.01$$

$$D_2 = (0-1)^2 + (1+0.1)^2 + (1-0)^2 + (0-0)^2 = 3.21$$

Thus, winning neuron is: j=1 (i.e. y_1)

Learning Vector Quantization (LVQ)

Update weights (T=J)

$$w_{11} = w_{11} + \alpha(x_1 - w_{11}) = 0 + 0.1(0 - 0) = 0$$
$$w_{12} = w_{12} + \alpha(x_2 - w_{12}) = 0 + 0.1(1 - 0) = 0.1$$
$$w_{13} = w_{13} + \alpha(x_3 - w_{13}) = 1.1 + 0.1(1 - 1.1) = 1.09$$
$$w_{14} = w_{14} + \alpha(x_4 - w_{14}) = 1 + 0.1(0 - 1) = 0.9$$

Thus, weigh matrix is

$$w = \begin{bmatrix} 0 & 1 \\ 0.1 & -0.1 \\ 1.09 & 0 \\ 0.9 & 0 \end{bmatrix}$$

Concept of Kernel Self Organizing Map

- Two limitations of Kohenon self-organizing maps (KSOM) are:
  - The formulation of the algorithm has no objective function that could be optimized.
  - The estimate of the probability density function of the input space provided by the algorithm lacks accuracy.
- The main concept behind the kernel SOM is to accommodate neurons with overlapping activation regions in the form of kernel functions.

Concept of Kernel Self Organizing Map

- There are many researches that have proposed variants of kernel SOM.
- Van Hulle proposed a kernel-based formulation of the self-organizing map. The motivation of which is improved topographic mapping.
- MacDonald and Fyfe proposed a kernel SOM from kernelizing the k-means clustering algorithm with added neighborhood learning.
- Andras et al. proposed a kernel SOM by transforming the input space to a feature space and then applying nonlinear kernel functions to the mapped data and obtained improved classification results.

Kullback–Leibler Divergence

- It is the measure of difference between two probability distributions over the same variable.
- The KL divergence between two distributions p and q is often stated using the following notation:

$$KL(p||q)$$

- It is calculated using following formula.

$$KL(p||q) = \sum_{i=1}^{n} p(x_i).\log \frac{p(x_i)}{q(x_i)}$$

Kullback–Leibler Divergence

- **Example:** Find KL divergence for following set of probability distributions

  p=[0.1, 0.4, 0.5]

  q=[0.8,0.15,0.05]

  Solution

$$KL(p||q) = 0.1 \times \log(0.1/0.8) + 0.4 \times \log(0.4/0.15) + 0.5 \times \log(0.5/0.05)$$
$$= 1.927$$

- Weight update rule of Kernel SOM can be derived from KL Divergence. Thus, we can say that KL divergence is helpful for Kernel SOM.