Module 3

KSOM

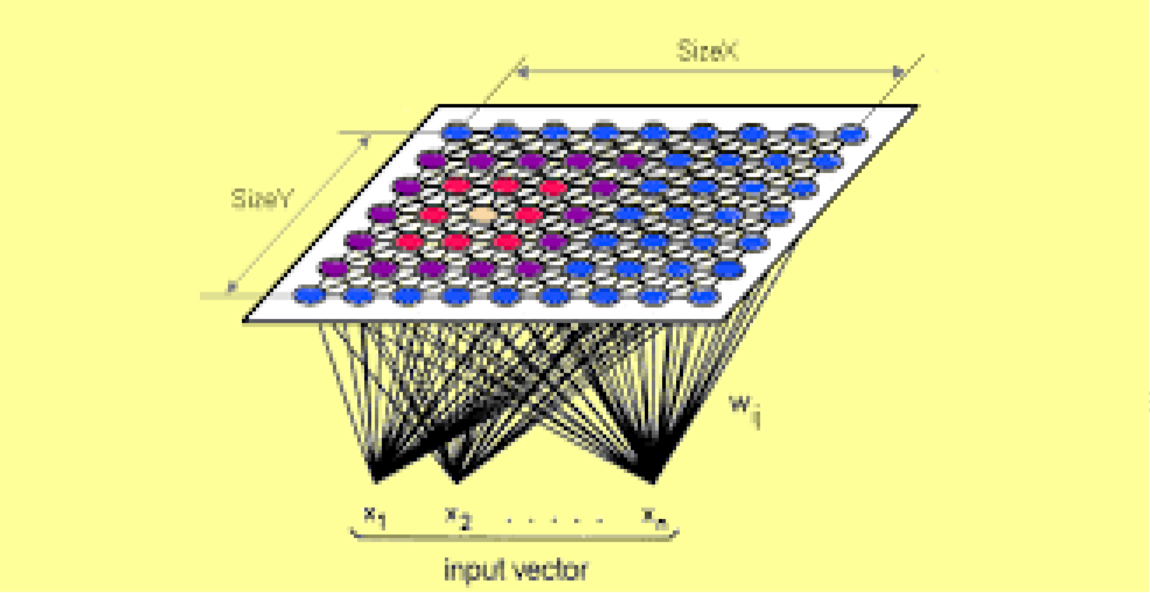
Topic	Suggested Books	Remarks
Fixed weights competitive nets.	S. N. Sivanandam and S. N. Deepa, “Principles of Soft Computing,” Chapter 5	Theory
Kohonen Self-organizing Feature Maps, Learning Vector Quantization.	S. N. Sivanandam and S. N. Deepa, “Principles of Soft Computing,” Chapter 5	Theory/Numericals No numerical on LVQ
Adaptive Resonance Theory –1.	S. N. Sivanandam and S. N. Deepa, “Principles of Soft Computing,” Chapter 5	Theory

Self Organizing Maps – Kohonen Maps

- It is a type of Artificial Neural Network which is also inspired by biological models of neural systems from the 1970s.
- It **follows an unsupervised learning** approach and
- it **trains** its network **through a competitive learning algorithm.**

Self Organizing Maps – Kohonen Maps

- SOM has two layers,
- one is the Input layer and
- the other one is the Output layer.



Competition

- It is a type of unsupervised artificial neural network
- It uses **competetive learning** to update its weights.

- Competetive learning is based on three processes :
  - **I. Competetion**
  - **II. Cooperation**
  - **III. Adaptation**

Competition

- It is a type of unsupervised artificial neural network
- It uses **competetive learning** to update its weights.

- Competetive learning is based on three processes :
    I. **Competetion**
    II. **Cooperation**
    III. **Adaptation**

I Competition

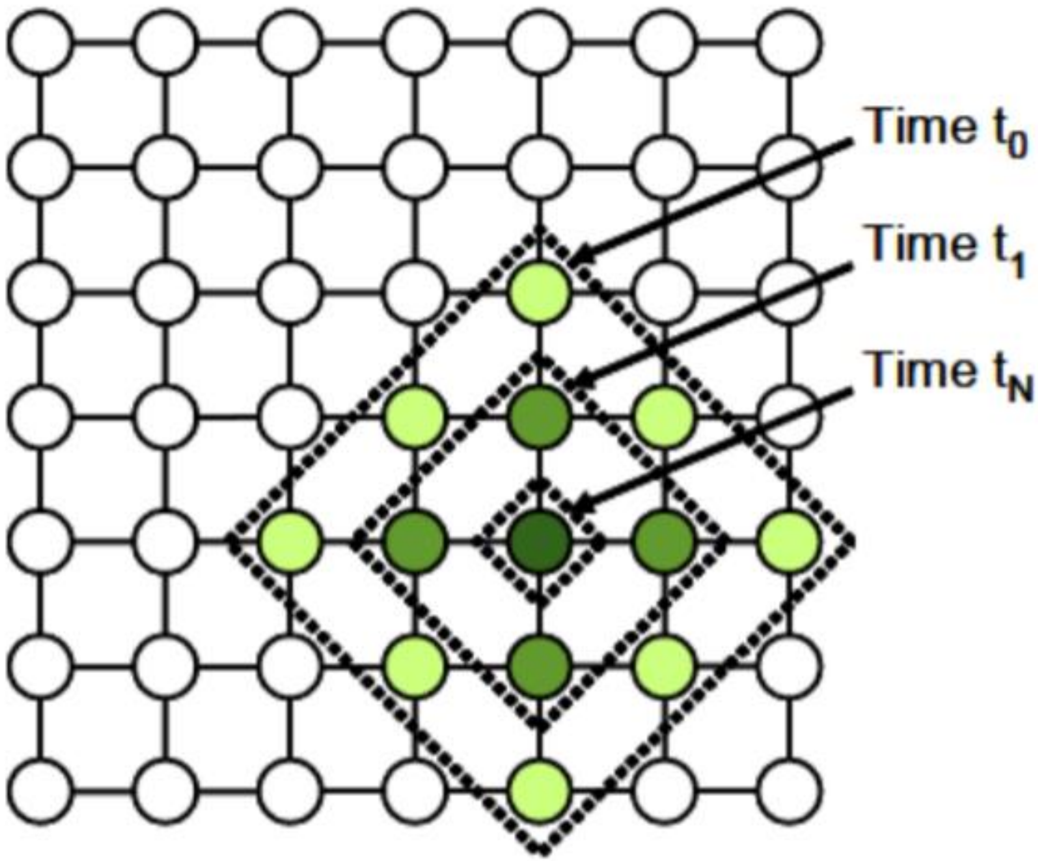
- It is a type of unsupervised artificial neural network
- It uses **competitive learning** to update its weights.

- Competitive learning is based on three processes :
  - I. **Competition**
  - II. **Cooperation**
  - III. **Adaptation**

I Competition

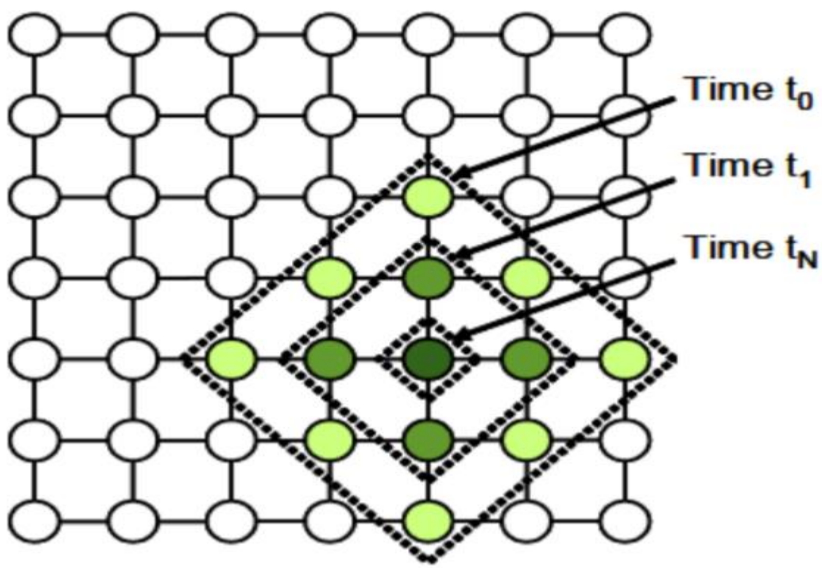
- Each neuron in a SOM is assigned **a weight vector** with the **same dimensionality** as **the input space.**
- In the example below, in each neuron of the output layer we will have a vector with dimension n.
- Compute distance **between each neuron (neuron from the output layer) and the input data**, and
- The neuron **with the lowest distance** will be the **winner of the competition.**

II Co OPERATION



II Co OPERATION

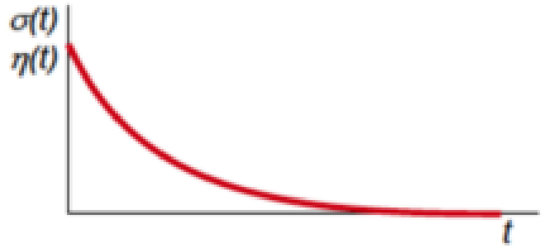
To choose neighbors we use **neighborhood kernel function**, this function depends on **two factor** : *time (* time incremented each new input data) and **distance between the winner neuron and the other neuron** (How far is the neuron from the winner neuron).



II Co OPERATION

$$w_k = w_k + \eta(t)\cdot h_{ik}(t)\cdot\left(x^{(n} - w_k\right)$$

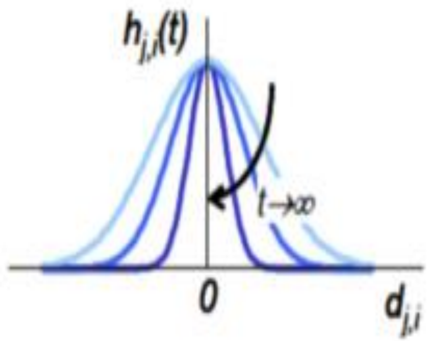
A learning rate decay rule $\eta(t) = \eta_0 \exp\left(-\frac{t}{T_1}\right)$



A neighborhood kernel function $h_{ik}(t) = \exp\left(-\frac{d_{ik}^2}{2\sigma^2(t)}\right)$

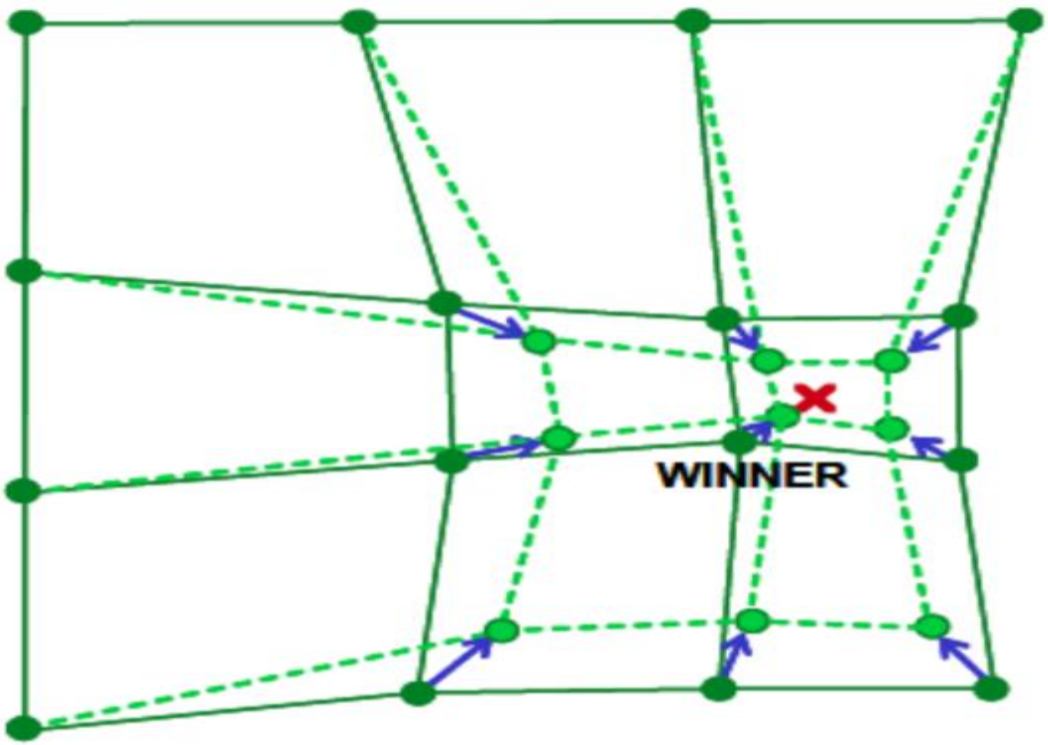
- where d_{ik} is the lattice distance between w_i and w_k

A neighborhood size decay rule $\sigma(t) = \sigma_0 \exp\left(-\frac{t}{T_2}\right)$



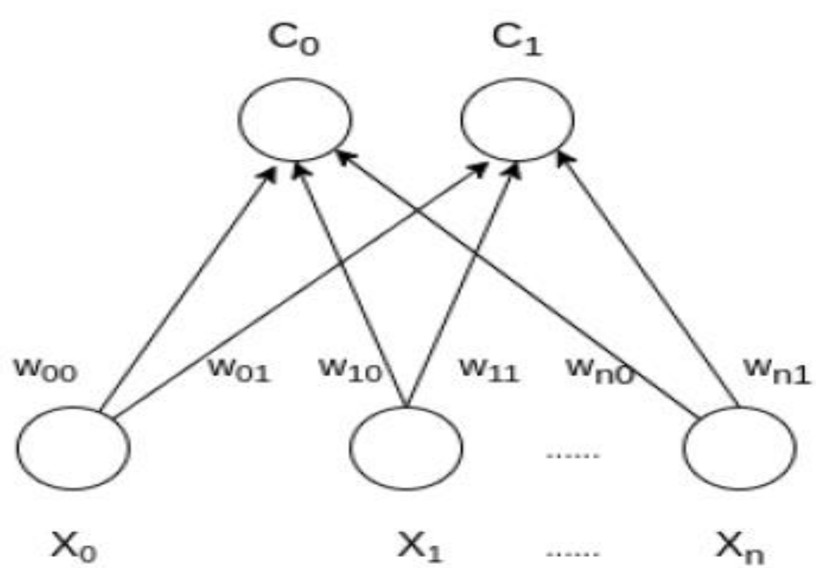
III ADAPTATION

- choosen neurons will be updated but not the same update, **more the distance between neuron and the input data grow less we adjust it** like shown in the image below :



Self Organizing Maps – Kohonen Maps

- **How do SOM works?**
- Let's say an input data of size (m, n) where m is the number of training examples and n is the number of features in each example.
- First, it initializes the weights of size (n, C) where C is the number of clusters.
- Here **it is three**
- **input neurons**
- **and two output**
- **neurons 3X2**

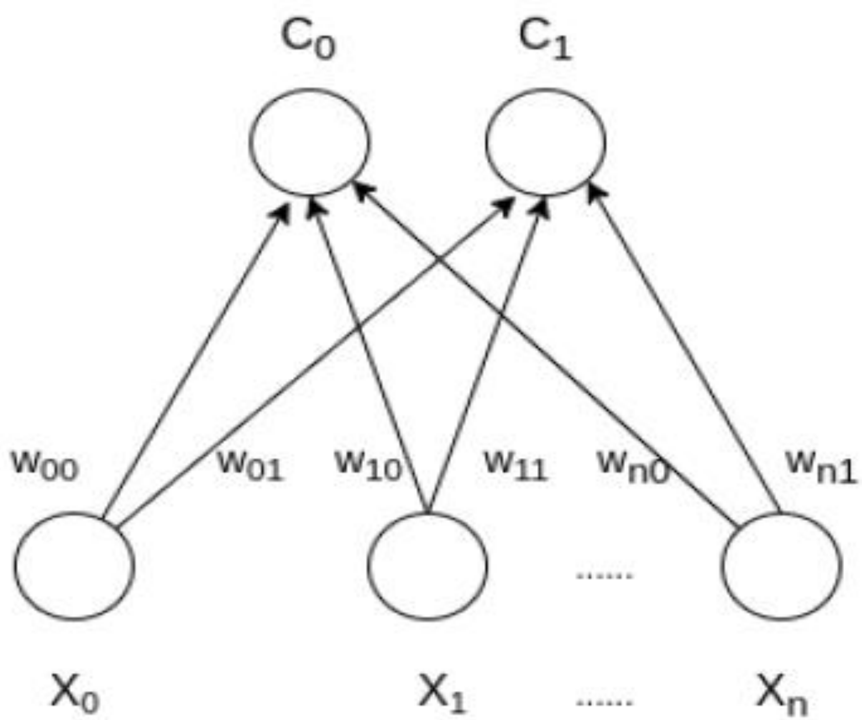


Self Organizing Maps – Kohonen Maps

- **How do SOM works?**
- Then iterating over the input data, for each training example, it updates the **winning vector** (weight vector with the shortest distance (e.g Euclidean distance) from training example).
- **Weight updation** rule is given by :
- **$w_{ij} = w_{ij}(old) + alpha(t) * (x_i^k - w_{ij}(old))$**
- where **alpha** is a learning rate at time t,
- **j** denotes the winning neuron,
- **i** denotes the i^{th} feature of training example and k denotes the k^{th} training example from the input data.
- **After training** the SOM network, **trained weights are used for clustering new examples**. A new example falls in the cluster of winning vectors.

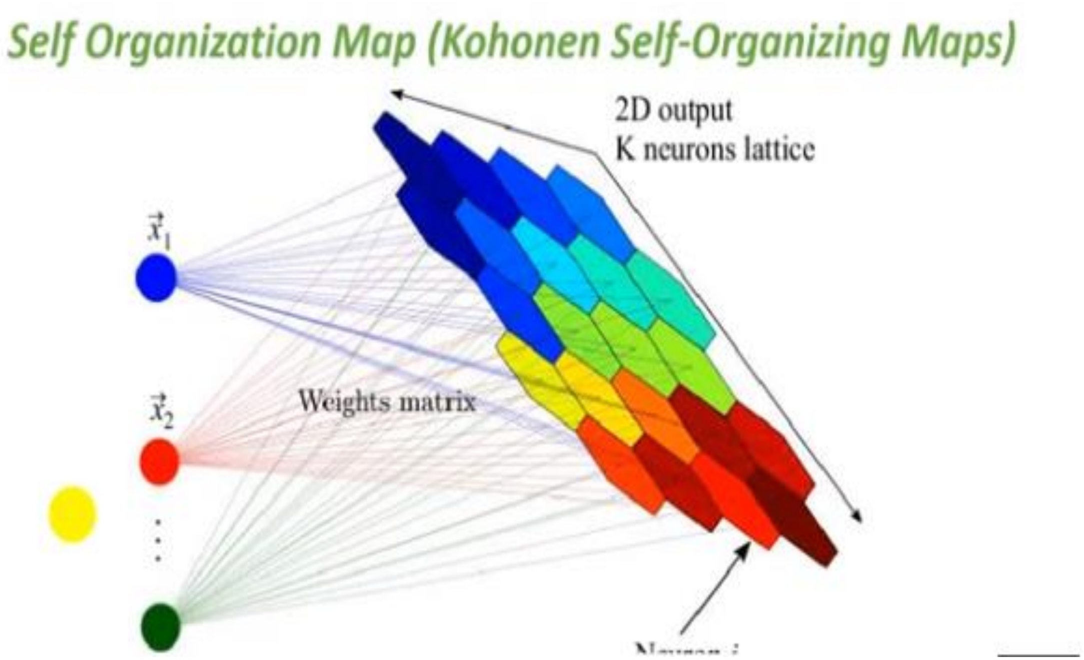
Self Organizing Maps – Kohonen Maps

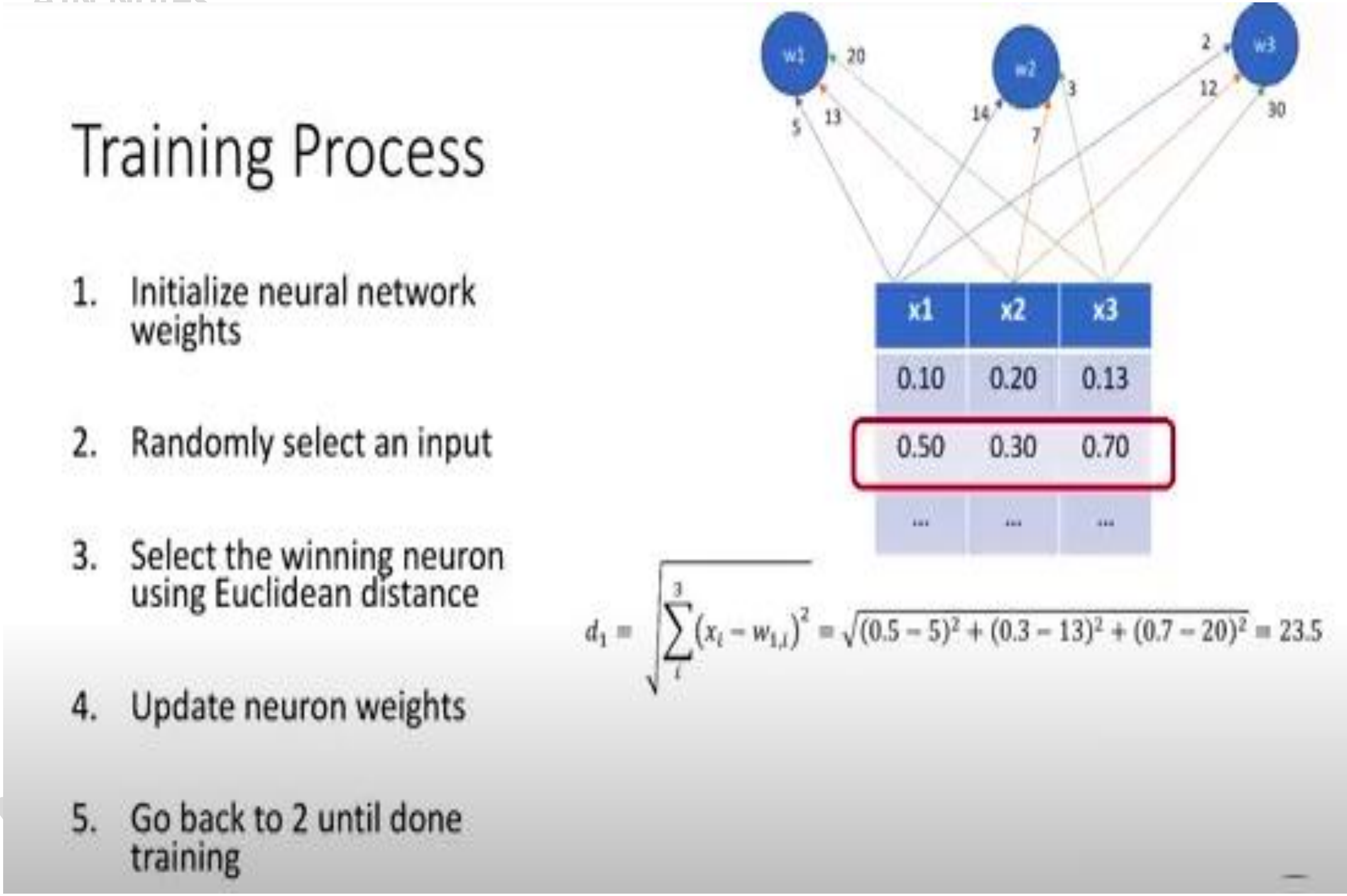
- SOM is used for clustering



Self Organizing Maps – Kohonen Maps

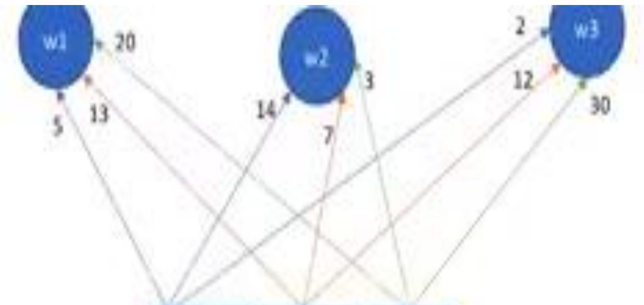
- SOM is used **mapping techniques to map multidimensional data onto lower-dimensional data**
- to reduce
- complex
- Problems
- for easy
- interpretation




Training Process
1. Initialize neural network weights
2. Randomly select an input
3. Select the winning neuron using Euclidean distance
4. Update neuron weights
5. Go back to 2 until done training
w1
w2
w3
20
5
13
14
7
3
2
12
30
x1 x2 x3
0.10 0.20 0.13
0.50 0.30 0.70
... ...
$d_1 = \sqrt{\sum_i^3 (x_i - w_{1,i})^2} = \sqrt{(0.5-5)^2 + (0.3-13)^2 + (0.7-20)^2} = 23.5$

Training Process

1. Initialize neural network weights
2. Randomly select an input
3. Select the winning neuron using Euclidean distance
4. Update neuron weights
5. Go back to 2 until done training



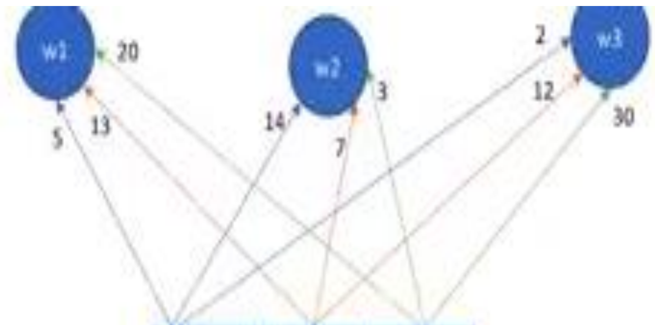
x1	x2	x3
0.10	0.20	0.13
0.50	0.30	0.70
...	...	...

$$d_1 = \sqrt{\sum_i^3 (x_i - w_{1,i})^2} = \sqrt{(0.5-5)^2 + (0.3-13)^2 + (0.7-20)^2} = 23.5$$

$$d_2 = \sqrt{\sum_i^3 (x_i - w_{2,i})^2} = \sqrt{(0.5-14)^2 + (0.3-7)^2 + (0.7-3)^2} = 15.2$$

Training Process

1. Initialize neural network weights
2. Randomly select an input
3. Select the winning neuron using Euclidean distance
4. Update neuron weights
5. Go back to 2 until done training



x1	x2	x3
0.10	0.20	0.13
0.50	0.30	0.70
...	...	...

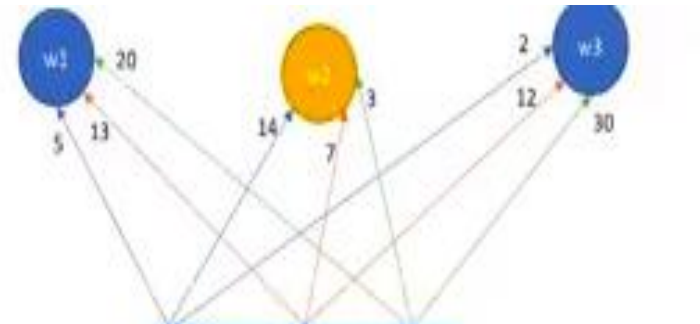
$$d_1 = \sqrt{\sum_{i}^{3}(x_i - w_{1,i})^2} = \sqrt{(0.5-5)^2 + (0.3-13)^2 + (0.7-20)^2} = 23.5$$

$$d_2 = \sqrt{\sum_{i}^{3}(x_i - w_{2,i})^2} = \sqrt{(0.5-14)^2 + (0.3-7)^2 + (0.7-3)^2} = 15.2$$

$$d_3 = \sqrt{\sum_{i}^{3}(x_i - w_{3,i})^2} = \sqrt{(0.5-2)^2 + (0.3-12)^2 + (0.7-30)^2} = 31.6$$

Training Process

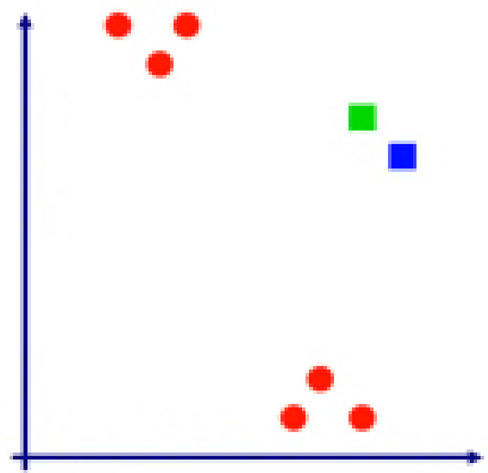
1. Initialize neural network weights
2. Randomly select an input
3. Select the winning neuron using Euclidean distance
4. Update neuron weights
5. Go back to 2 until done training



x1	x2	x3
0.10	0.20	0.13
0.50	0.30	0.70
...	...	

$$d_2 = \sqrt{\sum_i^3 (x_i - w_{2,i})^2} = \sqrt{(0.5-14)^2 + (0.3-7)^2 + (0.7-3)^2} = 15.2$$

Self Organizing Maps

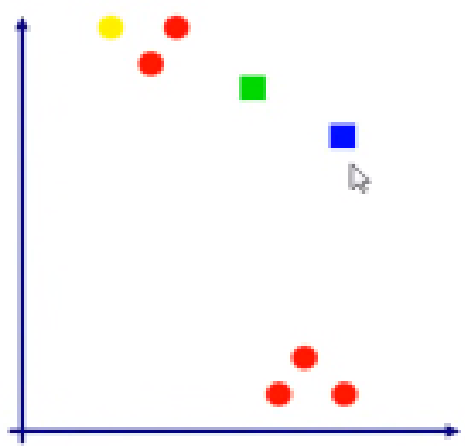


Operations

- Select random input
- Compute winner neuron
- Update neurons
- Repeat for all input data
- Classify input data

□ 2 Neurons
○ 6 Inputs

Self Organizing Maps

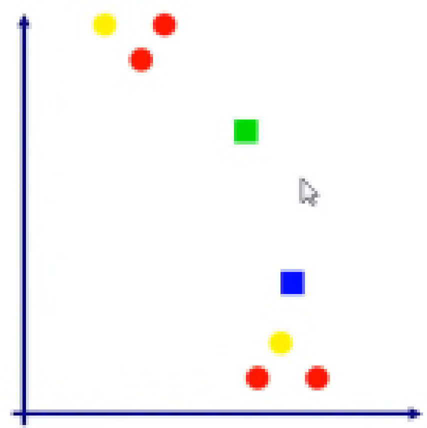


Operations

- Select random input
- Compute winner neuron
- Update neurons
- Repeat for all input data
- Classify input data

□ 2 Neurons
○ 6 Inputs

Self Organizing Maps

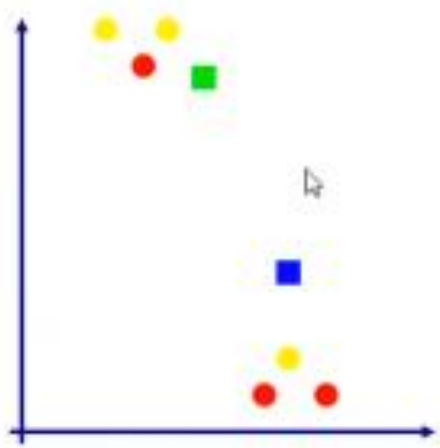


Operations

- Select random input
- Compute winner neuron
- Update neurons
- Repeat for all input data
- Classify input data

□ 2 Neurons
○ 6 Inputs

Self Organizing Maps

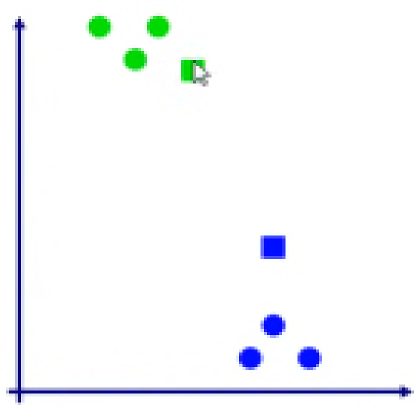


Operations

- Select random input
- Compute winner neuron
- Update neurons
- Repeat for all input data
- Classify input data

□ 2 Neurons
○ 6 Inputs

Self Organizing Maps



Operations
Select random input
Compute winner neuron
Update neurons
Repeat for all input data
Classify input data

□ 2 Neurons
○ 6 Inputs

Why SOM?

- Basically, SOMs are characterized as **a nonlinear,** ordered, smooth mapping of high-dimensional input data manifolds onto the elements of a regular, **low-dimensional array².**

Example 2: Triangle Input Distribution

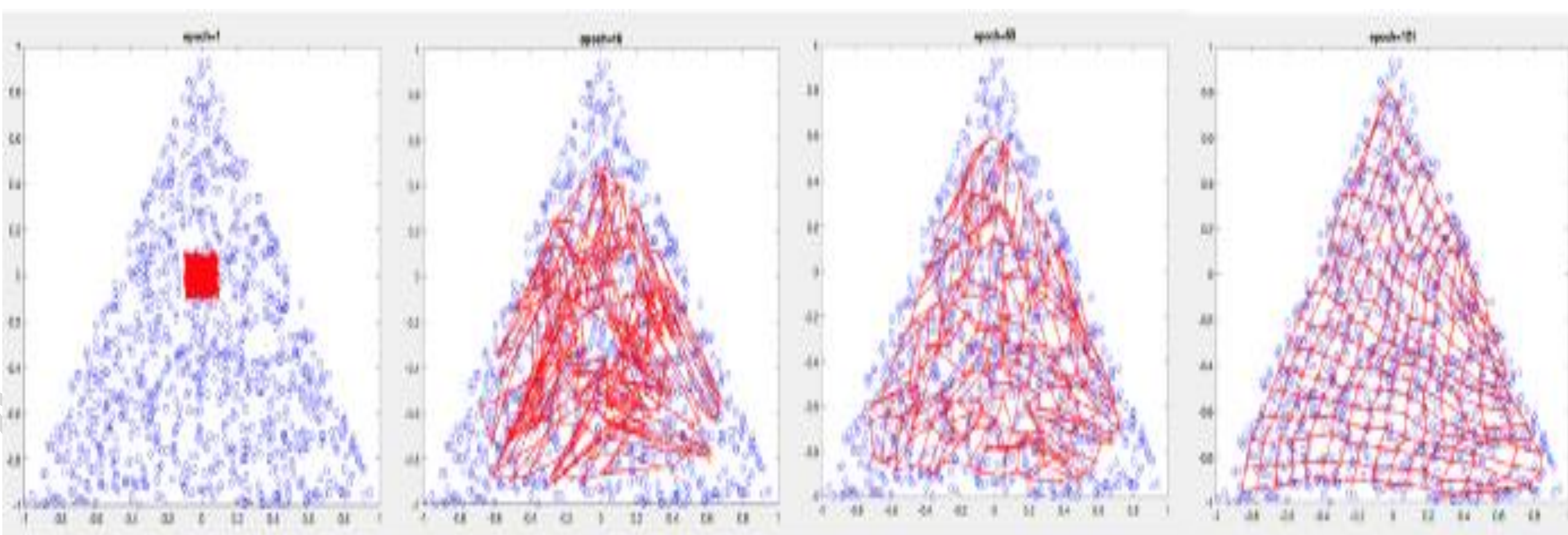
A simple illustration of the learning process is given in the next slide figure and we can understand the specialty of SOMs from this representation easily. Initially, input data(blue dots) occupy a special distribution in 2D space, and un-learned neuron(weights) values (red dots) are randomly distributed in a small area and after neurons get modified and learned by inputs, it gets the shape of the input data distribution step by step in the learning process.

Example 2: Triangle Input Distribution

In addition, each neuron became a representation of one small cluster of input data space. Therefore in this demonstration, we were able to represent 1000 data points with 100 neurons, preserving the topology of the input data. That means we have built a relationship between high-dimension data to low-dimensional representation (map). For further calculations and predictions, we can utilize these few neuron values to represent the tremendous input data space which makes processes much faster.

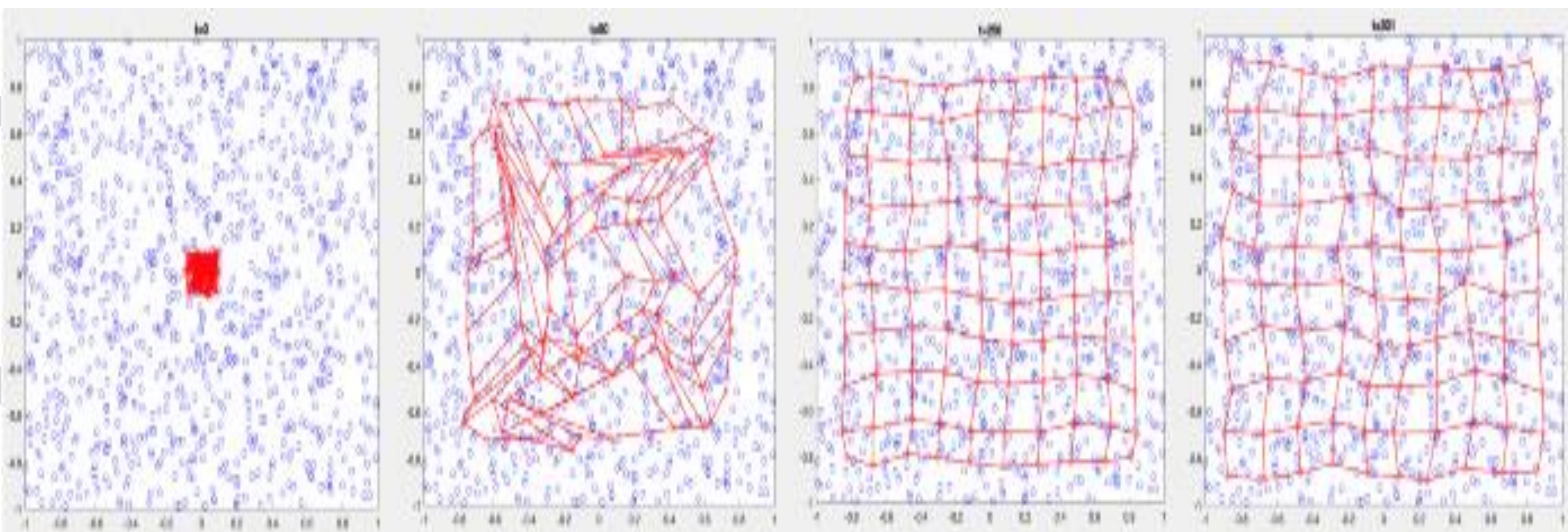
Example 2: Triangle Input Distribution

- After training the SOM's neurons we get a low dimensional representation of high dimensional input data **without disturbing the shape** of the data distribution **and relationship between each input data element**.



Example 1: Square Input Distribution

- input data values are square shapely random distributed over the 2-dimensional space.
- — **Inputs are given in blue dot** and **model's neuron values are given in red dots** for epoch 1, 50, 250 and 300



Self-organizing maps and other ANNs difference

- Self-organizing maps differ from other ANNs as they **apply unsupervised learning** as compared to error-correction learning (backpropagation with gradient descent etc),
- SOM **use a neighbourhood function to** preserve the topological properties of the input space.

KSOM solved example:Clustering

Ex- Construct SOFM to cluster following given vectors

X1=(0 0 1 1), X2=(1 0 0 0), X3=(0 1 1 0), X4=(0 0 0 1)

Number of clusters to be formed is two. Assume initial learning rate is 0.5.

Ans-

Initialize weights and learning rate α

Learning rate α = 0.5 (Given)

Number of input vectors are, n=4

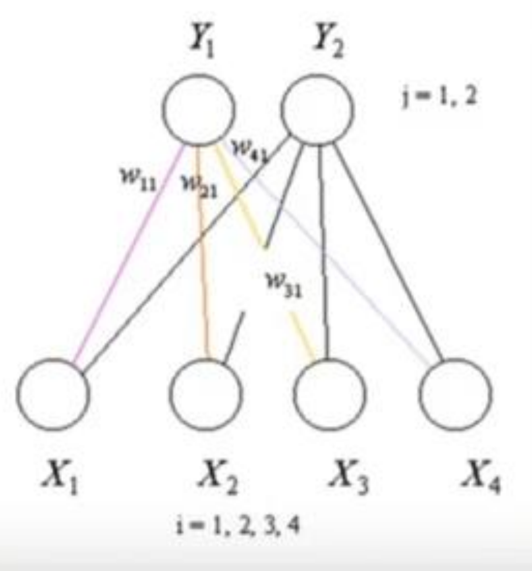
Therefore number of rows for weight matrix are 4,

i=1, 2, 3 & 4

Number of output clusters m=2

Therefore number of column for weight matrix are 2,

j=1 & 2



Here X are inputs and Y are output clusters.
W are weights. Weight is strength of that connection

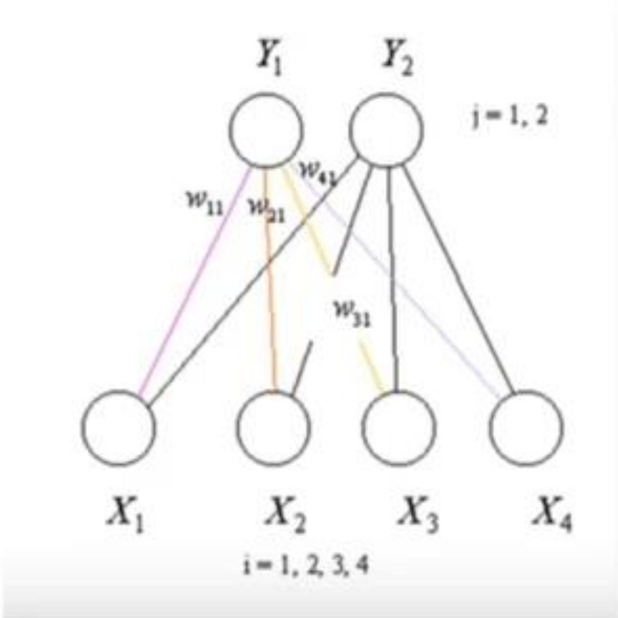
KSOM solved example:Clustering

Let us initialize weight matrix randomly with the values in between 0 to 1.

Number of rows, i=4,

Number of columns, j = 2

$$W_{ij} = \begin{bmatrix} 0.2 & 0.9 \\ 0.4 & 0.7 \\ 0.6 & 0.5 \\ 0.8 & 0.3 \end{bmatrix}$$



A) **Take first input vector X1= (x1, x2, x3, x4) = (0 0 1 1)**

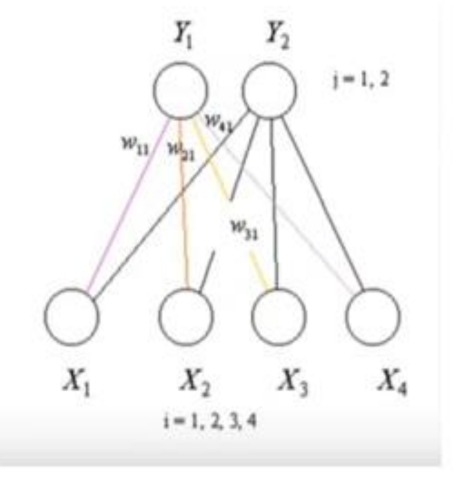
Calculate Euclidean Distance between clusters j=1, 2 and first input vector using

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

KSOM solved example:Clustering

A) Take first input vector X1= (x1, x2, x3, x4)

= (0 0 1 1)



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

Calculate distance between cluster j=1 and first input vector, $D_j = D_1$

$$D_1 = \sum_{i=1}^{n}(w_{i1} - x_i)^2$$

$$D_1 = (0.2-0)^2+(0.4-0)^2+(0.6-1)^2+(0.8-1)^2$$

$$D_1 = 0.4$$

Calculate distance between cluster j=2 and first input vector, $D_j = D_2$

$$D_2 = \sum_{i=1}^{n}(w_{i2} - x_i)^2$$

$$D_2 = (0.9-0)^2+(0.7-0)^2+(0.5-1)^2+(0.3-1)^2$$

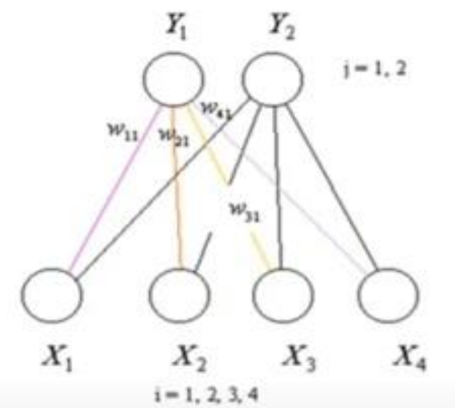
$$D_2 = 2.04$$

Here, $D_1 << D_2$

KSOM solved example:Clustering

Here, $D_1 << D_2$, So winning cluster is j=1 by considering minimum value.

So update weights of only column j=1 of above weight matrix.



Equation to update the weights is

$$W_{ij}(new) = w_{ij}(old) + \alpha [x_i - w_{ij}(old)]$$

Here, α=Learning rate=0.5 and j=1

$$W_{11}(new) = 0.2 + 0.5(0-0.2) = 0.1$$

$$W_{21}(new) = 0.4 + 0.5(0-0.4) = 0.2$$

$$W_{31}(new) = 0.6 + 0.5(1-0.6) = 0.8$$

$$W_{41}(new) = 0.8 + 0.5(1-0.8) = 0.9$$

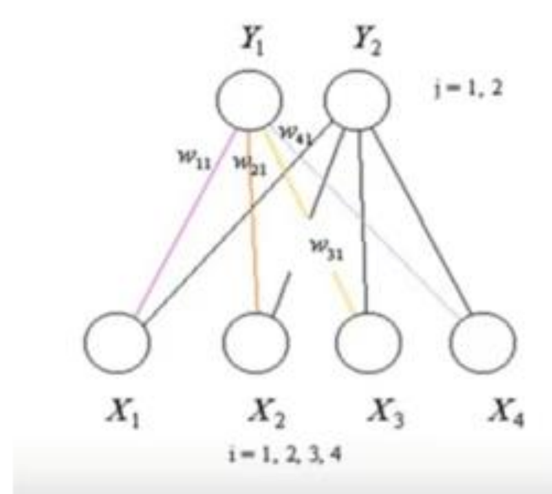
So new weight matrix is, $W_{ij} = \begin{bmatrix} 0.1 & 0.9 \\ 0.2 & 0.7 \\ 0.8 & 0.5 \\ 0.9 & 0.3 \end{bmatrix}$

KSOM solved example:Clustering

B) Take second input vector X2= (x1, x2, x3, x4)

= (1 0 0 0)

And new weight matrix, $W_{ij} = \begin{bmatrix} 0.1 & 0.9 \\ 0.2 & 0.7 \\ 0.8 & 0.5 \\ 0.9 & 0.3 \end{bmatrix}$



Calculate distance between cluster and second input vector

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

Calculate distance between cluster j=1 and second input vector, $D_j = D_1$

$$D_1 = \sum_{i=1}^{n} (w_{i1} - x_i)^2$$

$$D_1 = (0.1-1)^2 + (0.2-0)^2 + (0.8-0)^2 + (0.9-0)^2$$

Here, $D_2 << D_1$

Calculate distance between cluster j=2 and second input vector, $D_j = D_2$

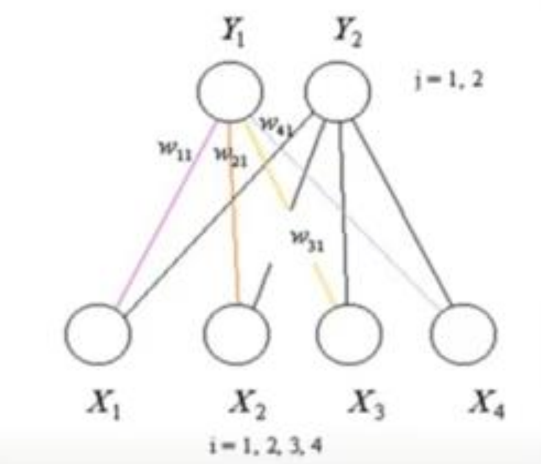
$$D_2 = \sum_{i=1}^{n} (w_{i2} - x_i)^2$$

KSOM solved example:Clustering

B) Take second input vector X2= (x1, x2, x3, x4)

= (1 0 0 0)

And new weight matrix, $W_{ij} = \begin{bmatrix} 0.1 & 0.9 \\ 0.2 & 0.7 \\ 0.8 & 0.5 \\ 0.9 & 0.3 \end{bmatrix}$



Calculate distance between cluster and second input vector

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

Calculate distance between cluster j=1 and second input vector, D_j=D_1

$$D_1 = \sum_{i=1}^{n}(w_{i1} - x_i)^2$$

$$D_1 = (0.1\text{-}1)^2+(0.2\text{-}0)^2+(0.8\text{-}0)^2+(0.9\text{-}0)^2$$

$$D_1 = 2.3$$

Calculate distance between cluster j=2 and second input vector, D_j=D_2

$$D_2 = \sum_{i=1}^{n}(w_{i2} - x_i)^2$$

$$D_2 = (0.9\text{-}1)^2+(0.7\text{-}0)^2+(0.5\text{-}0)^2+(0.3\text{-}0)^2$$

KSOM solved example:Clustering

B) Take second input vector X2= (x1, x2, x3, x4)

= (1 0 0 0)

And new weight matrix, $W_{ij} = \begin{bmatrix} 0.1 & 0.9 \\ 0.2 & 0.7 \\ 0.8 & 0.5 \\ 0.9 & 0.3 \end{bmatrix}$

Calculate distance between cluster and second input vector

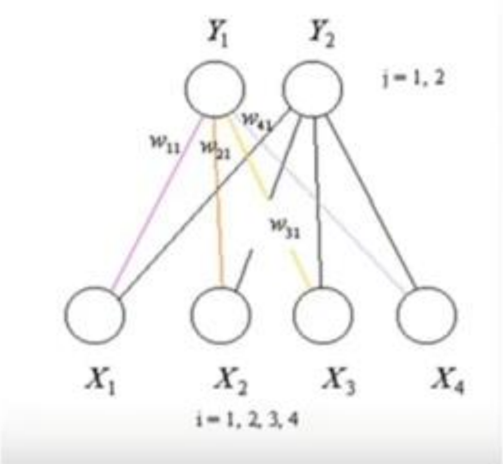
$$D_j = \sum_{i=1}^{n}(w_{ij} - x_i)^2$$

Calculate distance between cluster j=1 and second input vector, $D_j = D_1$

$$D_1 = \sum_{i=1}^{n}(w_{i1} - x_i)^2$$

$$D_1 = (0.1-1)^2+(0.2-0)^2+(0.8-0)^2+(0.9-0)^2$$

$$D_1 = 2.3$$



Calculate distance between cluster j=2 and second input vector, $D_j = D_2$

$$D_2 = \sum_{i=1}^{n}(w_{i2} - x_i)^2$$

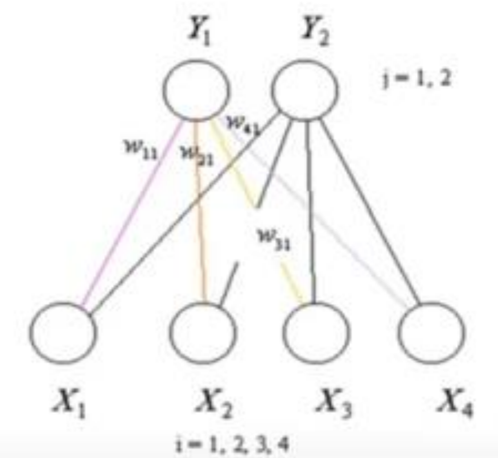
$$D_2 = (0.9-1)^2+(0.7-0)^2+(0.5-0)^2+(0.3-0)^2$$

$$D_2 = 0.84$$

Here, $D_2 << D_1$

KSOM solved example:Clustering

Here, $D_2 << D_1$, So winning cluster is j=2 by considering minimum value. So update only column j=2 of above weight matrix.



Equation to update the weights is

$$W_{ij}(new) = w_{ij}(old) + \alpha [x_i - w_{ij}(old)]$$

Here, α=Learning rate=0.5 and j=2

$$W_{12}(new) = 0.9 + 0.5(1-0.9) = 0.95$$

$$W_{22}(new) = 0.7 + 0.5(0-0.7) = 0.35$$

$$W_{32}(new) = 0.5 + 0.5(0-0.5) = 0.25$$

$$W_{42}(new) = 0.3 + 0.5(0-0.3) = 0.15$$

So updated weight matrix is,

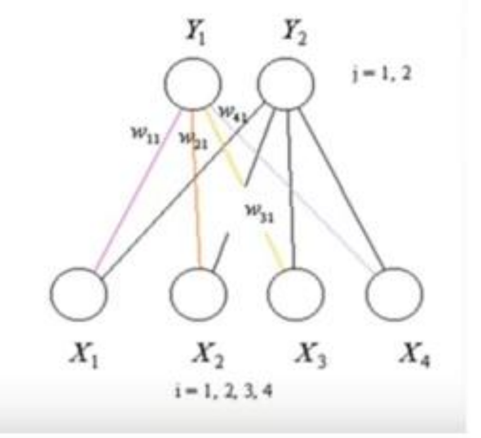
$$W_{ij} = \begin{bmatrix} 0.1 & 0.95 \\ 0.2 & 0.35 \\ 0.8 & 0.25 \\ 0.9 & 0.15 \end{bmatrix}$$

KSOM solved example:Clustering

C) Take third input vector X3= (x1, x2, x3, x4)

= (0 1 1 0)

And new weight matrix, $W_{ij} = \begin{bmatrix} 0.1 & 0.95 \\ 0.2 & 0.35 \\ 0.8 & 0.25 \\ 0.9 & 0.15 \end{bmatrix}$



Calculate distance between cluster and Third input vector

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

Calculate distance between cluster j=1 and third input vector, D_j=D_1

$$D_1 = \sum_{i=1}^{n}(w_{i1} - x_i)^2$$

$$D_1 = (0.1\text{-}0)^2+(0.2\text{-}1)^2+(0.8\text{-}1)^2+(0.9\text{-}0)^2$$

$$D_1 = 1.5$$

Calculate distance between cluster j=2 and third input vector, D_j=D_2

$$D_2 = \sum_{i=1}^{n}(w_{i2} - x_i)^2$$

$$D_2 = (0.95\text{-}0)^2+(0.35\text{-}1)^2+(0.25\text{-}1)^2+(0.15\text{-}0)^2$$

$$D_2 = 1.91$$

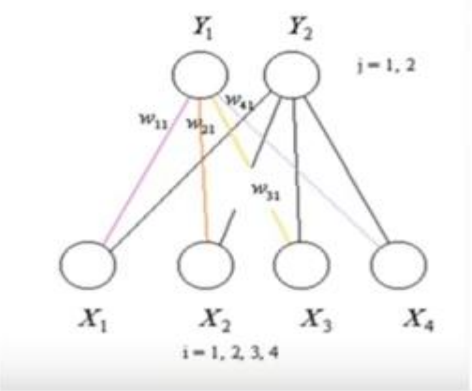
Here, $D_1 << D_2$

KSOM solved example:Clustering

C) Take third input vector X3= (x1, x2, x3, x4)

= (0 1 1 0)

And new weight matrix, $W_{ij} = \begin{bmatrix} 0.1 & 0.95 \\ 0.2 & 0.35 \\ 0.8 & 0.25 \\ 0.9 & 0.15 \end{bmatrix}$



Calculate distance between cluster and Third input vector

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

Calculate distance between cluster j=1 and third input vector, D_j=D_1

$$D_1 = \sum_{i=1}^{n}(w_{i1} - x_i)^2$$

$$D_1 = (0.1\text{-}0)^2+(0.2\text{-}1)^2+(0.8\text{-}1)^2+(0.9\text{-}0)^2$$

$$D_1 = 1.5$$

Calculate distance between cluster j=2 and third input vector, D_j=D_2

$$D_2 = \sum_{i=1}^{n}(w_{i2} - x_i)^2$$

$$D_2 = (0.95\text{-}0)^2+(0.35\text{-}1)^2+(0.25\text{-}1)^2+(0.15\text{-}0)^2$$

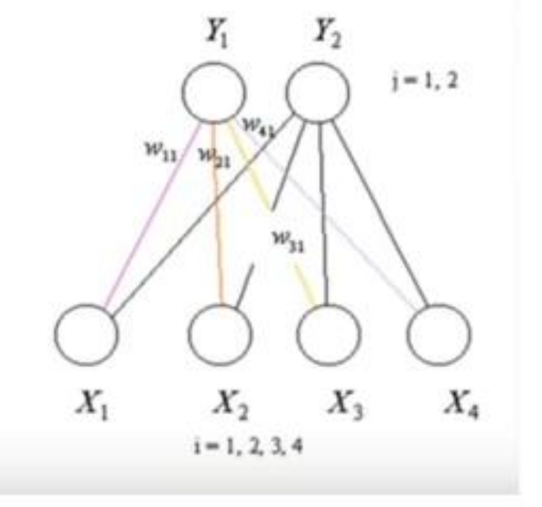
Here, $D_1 << D_2$

KSOM solved example:Clustering

C) Take third input vector X3= (x1, x2, x3, x4)

= (0 1 1 0)

And new weight matrix, $W_{ij} = \begin{bmatrix} 0.1 & 0.95 \\ 0.2 & 0.35 \\ 0.8 & 0.25 \\ 0.9 & 0.15 \end{bmatrix}$



Calculate distance between cluster and Third input vector

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

Calculate distance between cluster j=1 and third input vector, $D_j = D_1$

$$D_1 = \sum_{i=1}^{n}(w_{i1} - x_i)^2$$

$$D_1 = (0.1-0)^2+(0.2-1)^2+(0.8-1)^2+(0.9-0)^2$$

$$D_1 = 1.5$$

Here, $D_1 << D_2$

Calculate distance between cluster j=2 and third input vector, $D_j = D_2$

$$D_2 = \sum_{i=1}^{n}(w_{i2} - x_i)^2$$

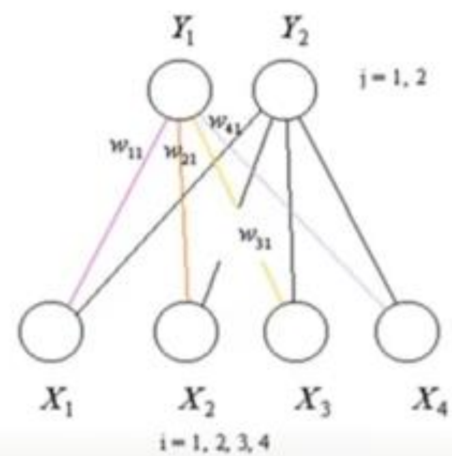
$$D_2 = (0.95-0)^2+(0.35-1)^2+(0.25-1)^2+(0.15-0)^2$$

$$D_2 = 1.91$$

KSOM solved example:Clustering

Here, $D_1 << D_2$, So winning cluster is j=1 by considering minimum value.

So update weights of only column j=1 of above weight matrix.



Equation to update the weights is

$$W_{ij}(new) = w_{ij}(old) + \alpha\,[x_i - w_{ij}(old)]$$

Here, α=Learning rate=0.5 and j=1

$$W_{11}(new) = 0.1 + 0.5(0-0.1) = 0.05$$

$$W_{21}(new) = 0.2 + 0.5(1-0.2) = 0.6$$

$$W_{31}(new) = 0.8 + 0.5(1-0.8) = 0.9$$

$$W_{41}(new) = 0.9 + 0.5(0-0.9) = 0.45$$

So updated weight matrix is,

$$W_{ij} = \begin{bmatrix} 0.05 & 0.95 \\ 0.6 & 0.35 \\ 0.9 & 0.25 \\ 0.45 & 0.15 \end{bmatrix}$$

KSOM solved example:Clustering

D) Take fourth input vector X4= (x1, x2, x3, x4)

= (0 0 0 1)

And new weight matrix, $W_{ij} = \begin{bmatrix} 0.05 & 0.95 \\ 0.6 & 0.35 \\ 0.9 & 0.25 \\ 0.45 & 0.15 \end{bmatrix}$

Calculate distance between cluster and fourth input vector

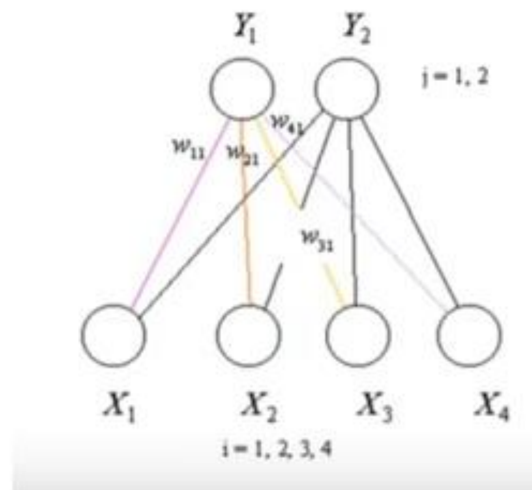
$$D_j = \sum_{i=1}^{n}(w_{ij} - x_i)^2$$

Calculate distance between cluster j=1 and fourth input vector, $D_j=D_1$

$$D_1 = \sum_{i=1}^{n}(w_{i1} - x_i)^2 = 1.475$$

Calculate distance between cluster j=2 and fourth input vector, $D_j=D_2$

$$D_2 = \sum_{i=1}^{n}(w_{i2} - x_i)^2 = 1.81$$



Here, $D_1 << D_2$

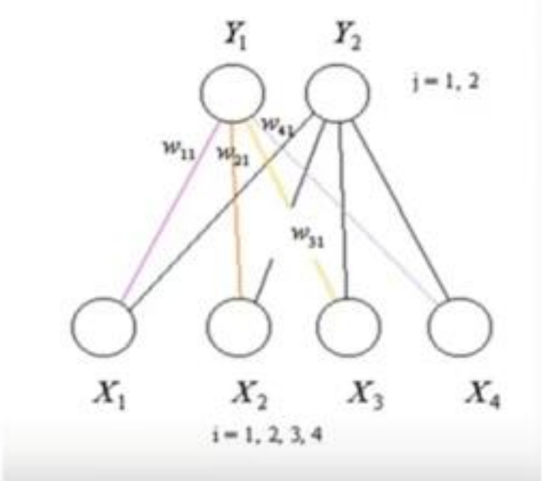
So winning cluster is j=1 by considering minimum value.

So update only column j=1 of above weight matrix.

KSOM solved example:Clustering

Here, $D_1 << D_2$, So winning cluster is j=1 by considering minimum value.

So update weights of only column j=1 of above weight matrix.



Equation to update the weights is

$$W_{ij}(new) = w_{ij}(old) + \alpha\,[x_i - w_{ij}(old)]$$

Here, α=Learning rate=0.5 and j=1

$W_{11}(new) =$

$W_{21}(new) =$

$W_{31}(new) =$

$W_{41}(new) =$

So updated weight matrix is,

W_{ij}

KSOM solved example:Clustering

Here, $D_1 << D_2$, So winning cluster is j=1 by considering minimum value.

So update weights of only column j=1 of above weight matrix.

Equation to update the weights is

$$W_{ij}(new) = w_{ij}(old) + \alpha\,[x_i - w_{ij}(old)]$$

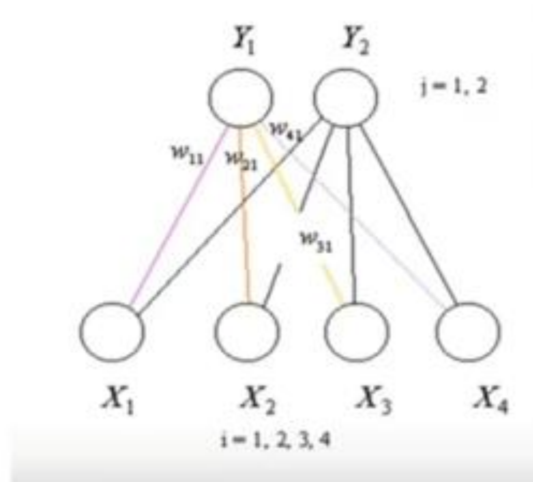
Here, α=Learning rate=0.5 and j=1

$W_{11}(new) = 0.05 + 0.5(0-0.05) =$

$W_{21}(new) = 0.6 + 0.5(0-0.6) =$

$W_{31}(new) = 0.9 + 0.5(0-0.9) =$

$W_{41}(new) = 0.45 + 0.5(1-0.45) =$



So updated weight matrix is,

W_{ij}

KSOM solved example:Clustering

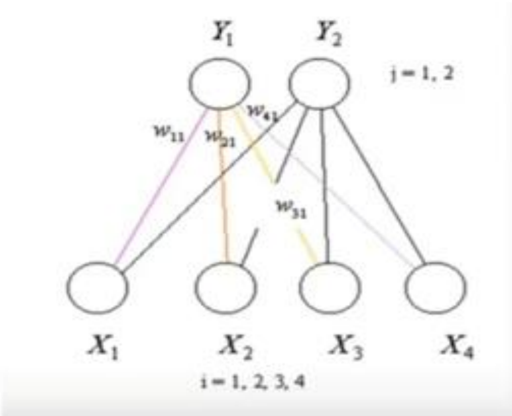
Here, $D_1 << D_2$, So winning cluster is j=1 by considering minimum value.

So update weights of only column j=1 of above weight matrix.

Equation to update the weights is

$$W_{ij}(new) = w_{ij}(old) + \alpha [x_i - w_{ij}(old)]$$

Here, α=Learning rate=0.5 and j=1



$W_{11}(new) = 0.05 + 0.5(0-0.05) = 0.025$

$W_{21}(new) = 0.6 + 0.5(0-0.6) = 0.3$

$W_{31}(new) = 0.9 + 0.5(0-0.$

$W_{41}(new) = 0.45 + 0.5(1-$

So updated weight matrix is,

$$W_{ij} = \begin{bmatrix} 0.025 & 0.95 \\ 0.3 & 0.35 \end{bmatrix}$$

KSOM solved example:Clustering

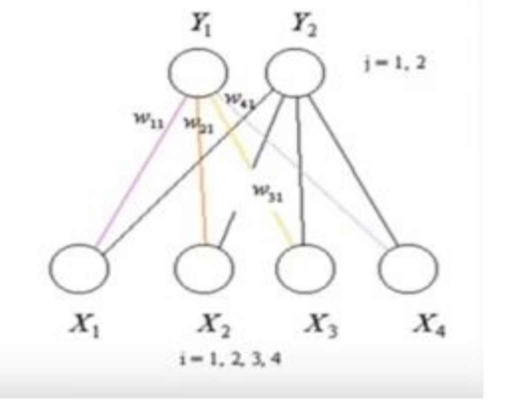
Here, $D_1 << D_2$, So winning cluster is j=1 by considering minimum value.

So update weights of only column j=1 of above weight matrix.

Equation to update the weights is

$$W_{ij}(new) = w_{ij}(old) + \alpha [x_i - w_{ij}(old)]$$

Here, α=Learning rate=0.5 and j=1



$$W_{11}(new) = 0.05 + 0.5(0-0.05) = 0.025$$

$$W_{21}(new) = 0.6 + 0.5(0-0.6) = 0.3$$

$$W_{31}(new) = 0.9 + 0.5(0-0.9) = 0.45$$

$$W_{41}(new) = 0.45 + 0.5(1-0.45) = 0.475$$

So updated weight matrix is,

$$W_{ij} = \begin{bmatrix} 0.025 & 0.95 \\ 0.3 & 0.35 \\ 0.45 & 0.25 \\ 0.475 & 0.15 \end{bmatrix}$$

AJN NOTES

AJN NOTES

AJN NOTES