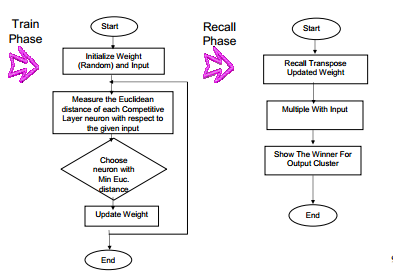
# **SELF ORGANISING MAPS**

**Flow chart**



**Pseudo code**

Initialize weights  
 For 0 to N number of training epochs

Select a sample from the input data set  
 Find the "winning" neuron for the sample input  
 Adjust the weights of nearby neurons

End for loop

The stages of the SOM algorithm that achieves this can be summarised as follows:

1. Initialization – Choose random values for the initial weight vectors wj

2. Sampling – Draw a sample training input vector x from the input space.

3. Matching – Find the winning neuron I(x) that has weight vector closest to the input vector

4. Updating – Apply the weight update equation

5. Continuation – keep returning to step 2 until the feature map stops changing.

**ALGORITHM AND IMPLEMENTATION**

**Training input**

[[1, 1, 0, 0],

[0, 0, 0, 1],

[1, 0, 0, 0],

[0, 0, 1, 1]]

**Weight initialisation**

self.w = [[0.2, 0.6, 0.5, 0.9],

[0.8, 0.4, 0.7, 0.3]]

**Calculate the distance of input vector and weight**

for i in range(self.maxClusters):

for j in range(self.mVectors):

self.mD[i] += math.pow((self.w[i][j] - trainingTests[vectorNumber][j]), 2)

**See which is smaller, mD(0) or mD(1)?**

dMin = 1 if self.mD[0] > self.mD[1] else 0

**Update the weights on the winning unit.**

for j in range(self.mVectors):

self.w[dMin][j] = self.w[dMin][j] + (self.mAlpha \* (patterns[i][j] - self.w[dMin][j]))

**Reduce the learning rate.**

self.mAlpha = self.decayRate \* self.mAlpha

**SCREEN SHOT**

