# Experiment 6

| | |
|---|---|
| **NAME** | **Shreya Shetty** |
| **UID** | **2019140059** |
| **CLASS** | **TE IT** |
| **BATCH** | **B** |
| **ACADEMIC YEAR** | **2021-22** |
| **SUBJECT** | **SC (Soft Computing)** |
| **COURSE CODE** | **IT312** |
| **EXPERIMENT NO.** | **6** |

## *Aim:*

To implement an unsupervised learning algorithm (KSOFM) for pattern classification problem.

## *Theory:*

***KSOFM (Kohonen Self-Organizing Feature Maps)***

- Feature mapping is a process which converts the patterns of arbitrary dimensionality into a response of one or two-dimensional arrays of neurons, i.e., it converts a wide pattern space into a typical feature space.
- The network performing such a mapping is called feature map.
- Apart from its capability to reduce the higher dimensionality, it has to preserve the neighborhood relations of the input patterns, i.e., it has to obtain a topology preserving map.
- For obtaining such feature maps, it is required to find self-organizing neuralarcay which consists of neurons arranged in one-dimensional array or a two-dimensional array.
- To depict this, a typical network structure where each component of the input vector x is connected to each of the nodes is shown in below Figure.

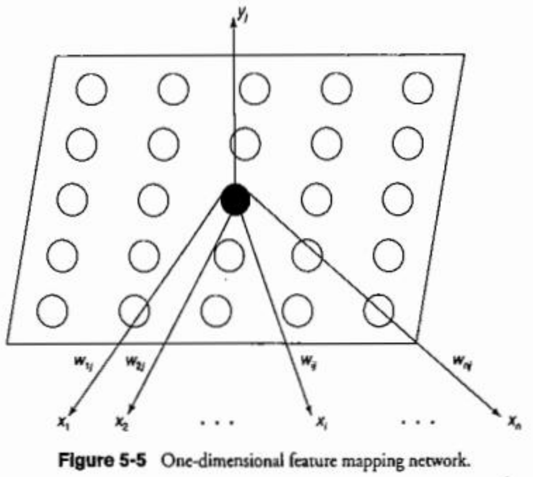


**Figure 5-5** One-dimensional feature mapping network.

- On the other hand, if the input vector is two-dimensional, the inputs, say la, b), can arrange themselves in a two-dimensional array defining the input space (a, b) as in Figure 5-6. Here, the two layers are fully connected.

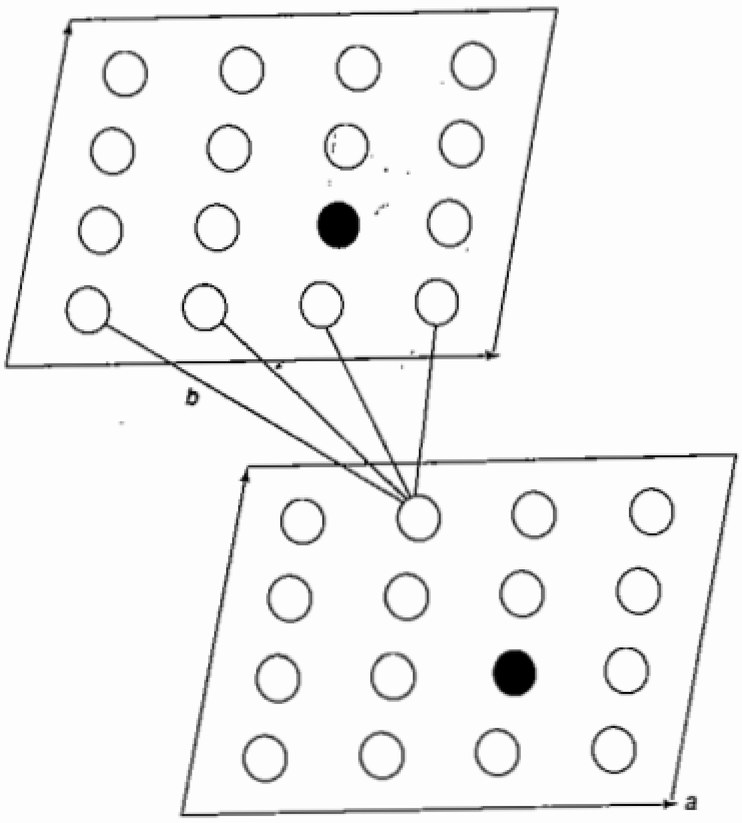


**Figure 5-6** Two-dimensional feature mapping network.

- The topological preserving property is observed in the brain, but not found in any other artificial neural network.
- Here, there are in ourpur cluster units arranged in a one-or two-dimensional array and the input signals are n-tuples. The cluster (output) units' weight vector serves as an exemplar of the inpul pattern char is associated with that cluster.
- At the time of self-organization, the weight vector of the cluster unit which matches the input pattern very closely is chosen as the winner unit.
- The closeness of weight vector of cluster unit to the input pattern may be based on the square of the minimum Euclidean distance.
- The weights are updated for the winning unic and its neighboring units.
- It should be noced that the weight vectors of the neighboring units are not close to the input pattern and the connective weights do not multiply the signal sent from the input units to the cluster units until dor product measure of similarity is being used.
- The extension of Kohonen feature map for a multilayer network involves the addition of an association layer to the output of the self-organizing feature map layer.

- The output node is found to associate the desired output values with certain input vectors.
- This type of architecture is called as Kohonen self-organizing motor map (KSOMM; Ritter, 1992) and layer that is added is called a motor map in which the movement command, are being mapped into two-dimensional locations of excitation.
- Here, the feature map is a hidden layer this acts as a competitive network which classifies the Input vectors.
- The motor map formation is based on the learning of a control task.
- The motor map learning may be either supervised or unsupervised learning and can be performed by delta learning rule or outstar learning rule.
- The motor map learning is an extension of Kohonen's original learning algorithm.

***ARCHITECTURE***

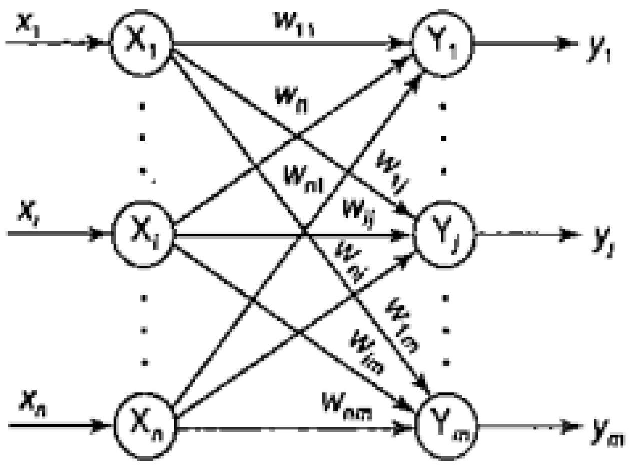


**Figure 5-10** Kohonen self-organizing feature map architecture.

## ***Training Algorithm for KSOFM***

Step 0: Initialize the reference vectors. This can be done using the following steps.

- From the given set of training vectors, take the first "m" (number of clusters) training vectors and use them as weight vectors, the remaining vectors can be used for training.
- Assign the initial weights and classifications randomly.
- K-means clustering method.

Set initial learning rate alpha.

Step 1: Perform Steps 2-6 if the stopping condition is false.

Step 2: Perform Steps 3-4 for each training input vector x.

Step 3: Calculate the Euclidean distance; for i = 1 to n, j = 1 to m.

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

Find the winning unit index J, when D(J) is minimum.

Step 4: Update the weights on the winning unit, $W_j$ using the following conditions.

If T = y then $W_j$(new) = $W_j$(old) + α($x_j$ - $W_j$(old)]

If T != y then $W_j$(new) = $W_j$(old) - α ($x_j$ - $W_j$(old)]

Step 5: Reduce the learning rate α.

Step 6: Test for the stopping condition of the training process. (The stopping conditions may be fixed number of epochs or if learning rate has reduced to a negligible value)

***FlowChart:***

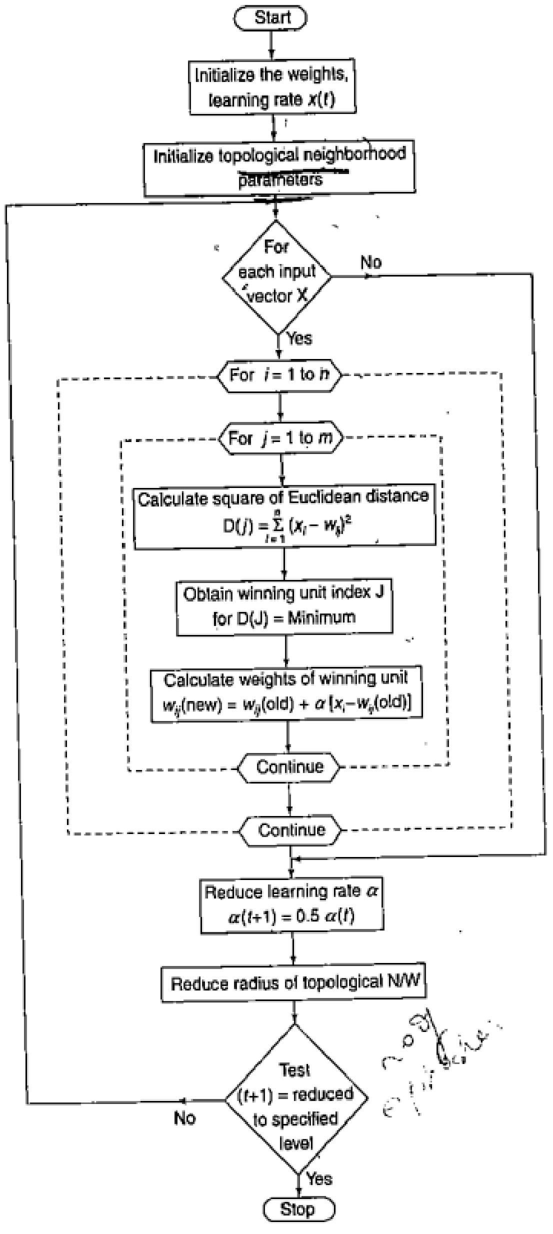
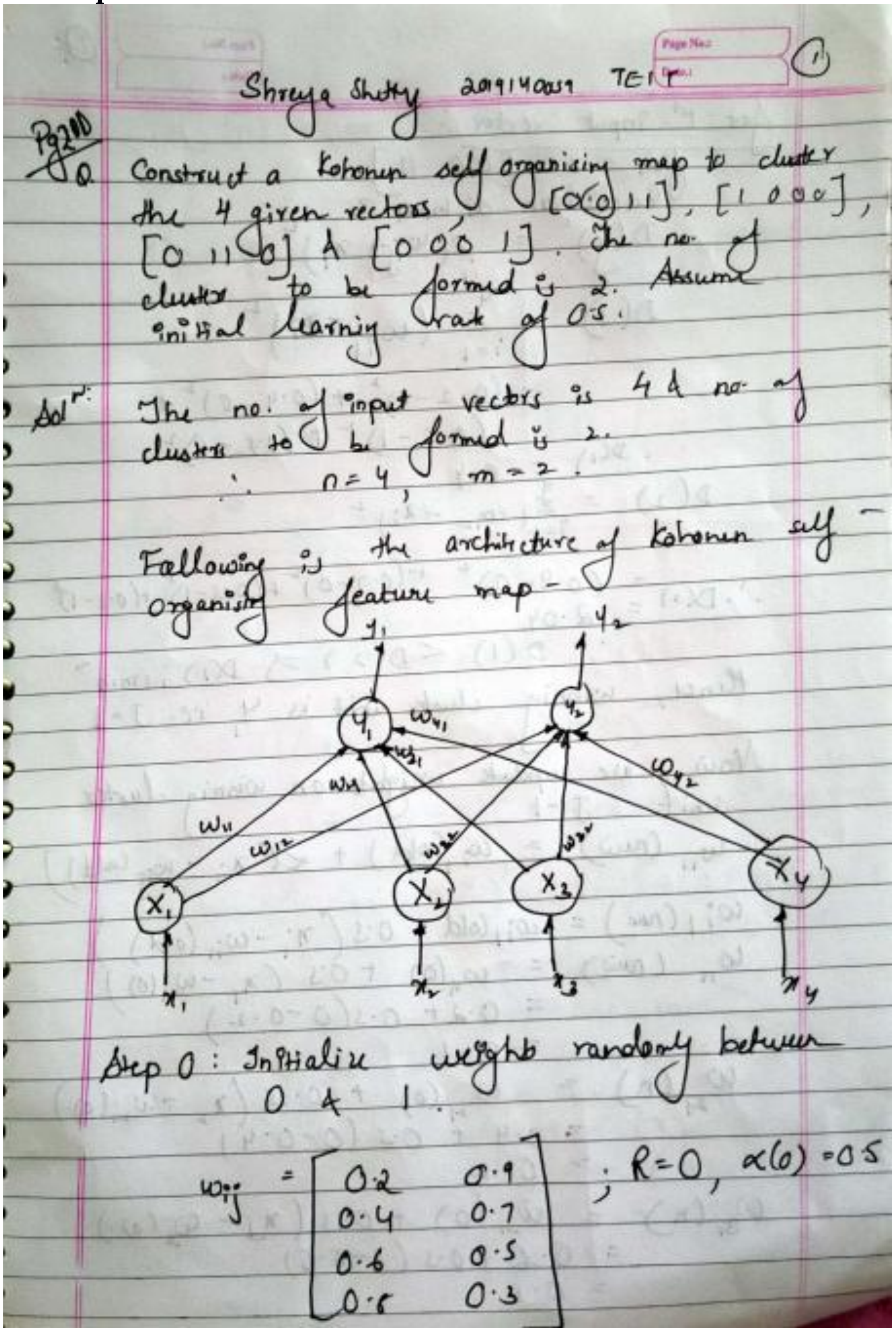

Start
Initialize the weights, learning rate x(t)
Initialize topological neighborhood parameters
For each input vector X
No
Yes
For i = 1 to n
For j = 1 to m
Calculate square of Euclidean distance D(j) = Σ (x_i − w_ij)²
Obtain winning unit index J for D(J) = Minimum
Calculate weights of winning unit w_ij(new) = w_ij(old) + α [x_i − w_ij(old)]
Continue
Continue
Reduce learning rate α α(t+1) = 0.5 α(t)
Reduce radius of topological N/W
Test (t+1) = reduced to specified level
No
Yes
Stop

**Figure 5-11** Flowchart for training process of KSOFM.

***Procedure:***

1. Import necessary libraries
2. Take input of weights
3. Take input of vectors
4. For each input vector x, calculate winner unit j and update the weights on winning cluster

***Solved Example:***



Shreya Shetty 2019140059 TE IT

1

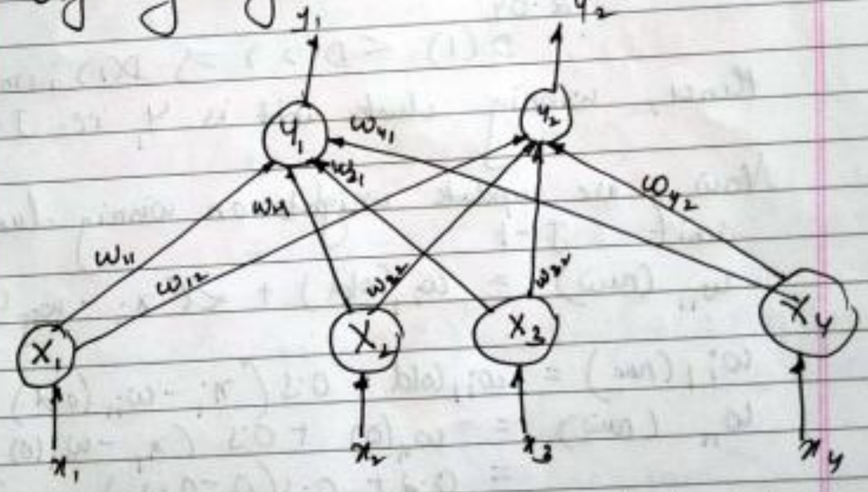
Pg 200

Q. Construct a Kohonen self organising map to cluster the 4 given vectors [0 0 1 1], [1 0 0 0], [0 1 1 0] & [0 0 0 1]. The no. of cluster to be formed is 2. Assume initial learning rate of 0.5.

Sol$^n$: The no. of input vectors is 4 & no. of clusters to be formed is 2.

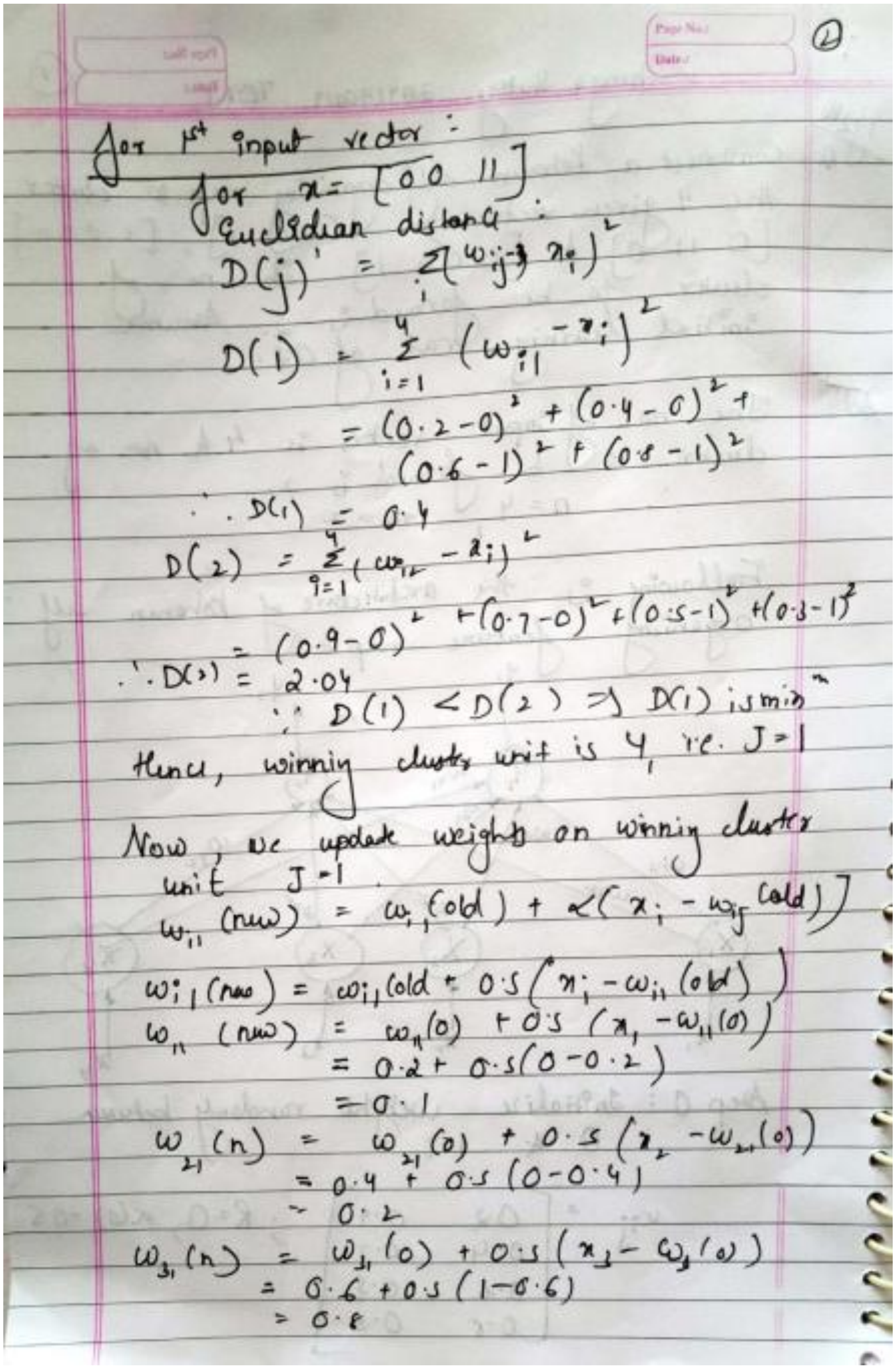
$\therefore\ n = 4,\ m = 2.$

Following is the architecture of Kohonen self-organising feature map -



Step 0 : Initialize weights randomly between 0 & 1.

$$w_{ij} = \begin{bmatrix} 0.2 & 0.9 \\ 0.4 & 0.7 \\ 0.6 & 0.5 \\ 0.8 & 0.3 \end{bmatrix} ;\ R = 0,\ \alpha(0) = 0.5$$



2

For 1st input vector :

for $x = [0\ 0\ 1\ 1]$

Euclidian distance :

$D(j) = \sum_i (w_{ij} - x_i)^2$

$D(1) = \sum_{i=1}^{4} (w_{i1} - x_i)^2$

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

$\therefore D(1) = 0.4$

$D(2) = \sum_{i=1}^{4} (w_{i2} - x_i)^2$

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

$\therefore D(2) = 2.04$

$\because D(1) < D(2) \Rightarrow D(1)$ is minm

Hence, winning cluster unit is $Y_1$ i.e. $J=1$

Now, we update weights on winning cluster unit $J=1$.

$w_{i1}(new) = w_{i1}(old) + \alpha(x_i - w_{ij}(old))]$

$w_{i1}(new) = w_{i1}(old + 0.5(x_i - w_{i1}(old))$

$w_{11}(new) = w_{11}(0) + 0.5(x_1 - w_{11}(0))$

$= 0.2 + 0.5(0 - 0.2)$

$= 0.1$

$w_{21}(n) = w_{21}(0) + 0.5(x_2 - w_{21}(0))$

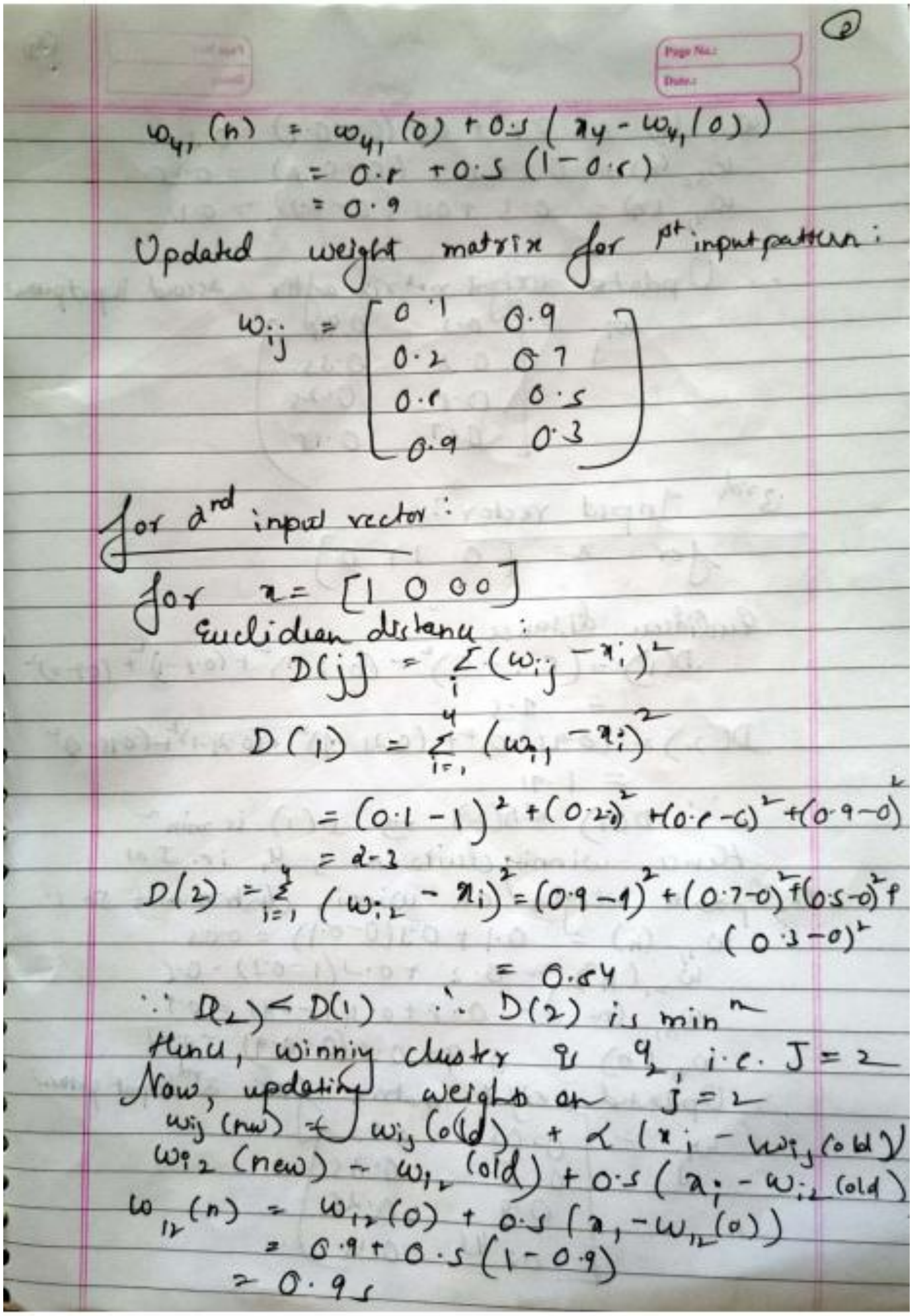
$= 0.4 + 0.5(0 - 0.4)$

$= 0.2$

$w_{31}(n) = w_{31}(0) + 0.5(x_3 - w_{31}(0))$

$= 0.6 + 0.5(1 - 0.6)$

$= 0.8$



$\omega_{41}(n) = \omega_{41}(0) + 0.5(x_4 - \omega_{41}(0))$

$= 0.8 + 0.5(1 - 0.8)$

$= 0.9$

Updated weight matrix for 1st input pattern:

$$w_{ij} = \begin{bmatrix} 0.1 & 0.9 \\ 0.2 & 0.7 \\ 0.8 & 0.5 \\ 0.9 & 0.3 \end{bmatrix}$$

For 2nd input vector:

For $x = [1\ 0\ 0\ 0]$

Euclidean distance:

$D(j) = \sum_i (w_{ij} - x_i)^2$

$D(1) = \sum_{i=1}^{4} (w_{i1} - x_i)^2$

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

$= 2.3$

$D(2) = \sum_{i=1}^{4} (w_{i2} - x_i)^2 = (0.9 - 1)^2 + (0.7 - 0)^2 + (0.5 - 0)^2 + (0.3 - 0)^2$

$= 0.84$

$\because D(2) < D(1)$ $\therefore D(2)$ is $\min^n$

Hence, winning cluster is $y_2$, i.e. $J = 2$

Now, updating weights on $J = 2$

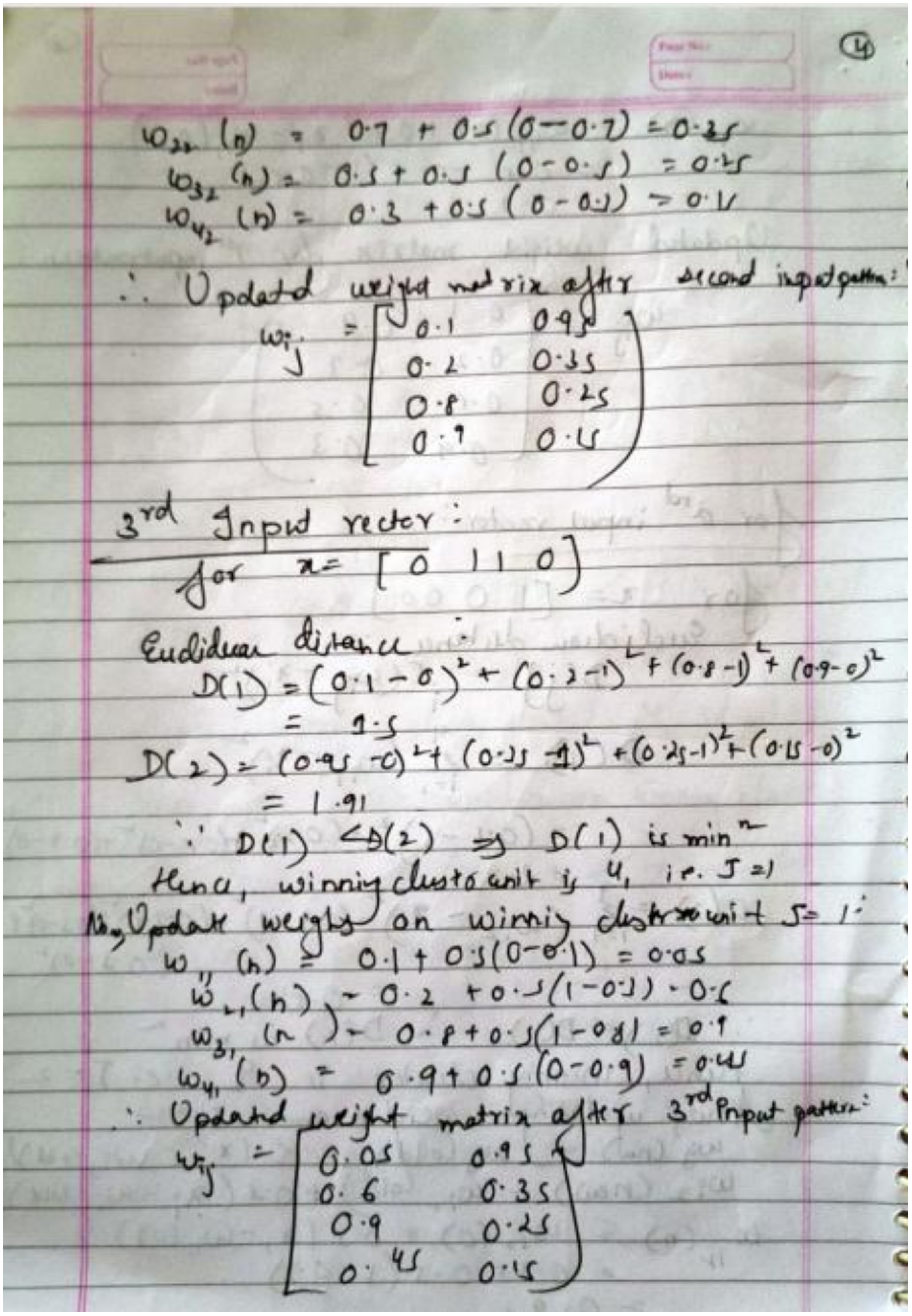
$w_{ij}(new) = w_{ij}(old) + \alpha(x_i - w_{ij}(old))$

$w_{i2}(new) = w_{i2}(old) + 0.5(x_i - w_{i2}(old))$

$w_{12}(n) = w_{12}(0) + 0.5(x_1 - w_{12}(0))$

$= 0.9 + 0.5(1 - 0.9)$

$= 0.95$



4

$w_{22}(n) = 0.7 + 0.5(0-0.7) = 0.35$

$w_{32}(n) = 0.5 + 0.5(0-0.5) = 0.25$

$w_{42}(n) = 0.3 + 0.5(0-0.3) = 0.15$

∴ Updated weight matrix after second input pattern:

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

3rd Input vector:

for $x = [0\ 1\ 1\ 0]$

Euclidean distance:

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

$= 1.5$

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

$= 1.91$

$\because D(1) < D(2) \Rightarrow D(1)$ is min$^n$

Hence, winning cluster unit is $u_1$ i.e. $J=1$

Now, Update weights on winning cluster unit $J=1$:

$w_{11}(n) = 0.1 + 0.5(0-0.1) = 0.05$

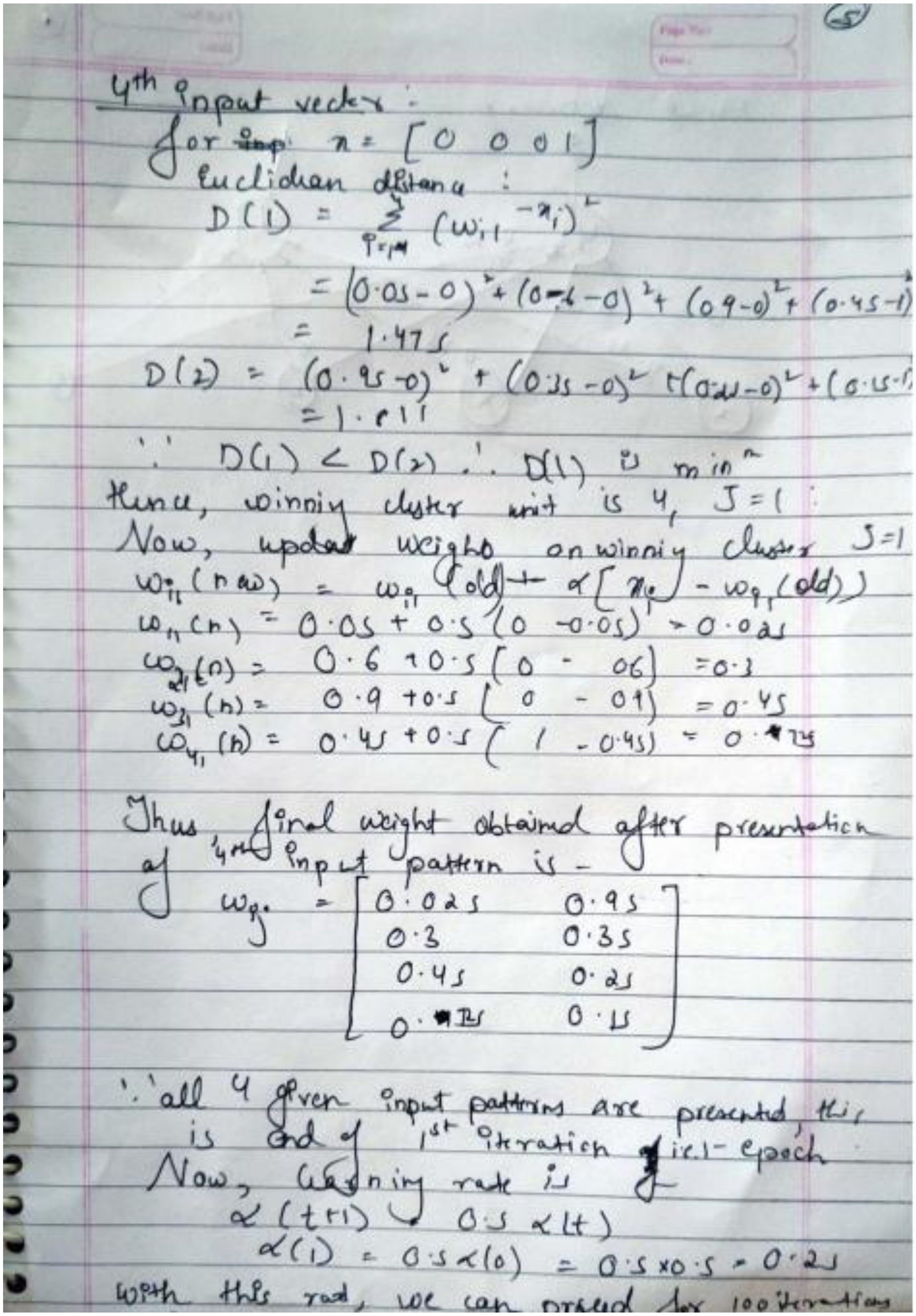
$w_{21}(n) = 0.2 + 0.5(1-0.2) = 0.6$

$w_{31}(n) = 0.8 + 0.5(1-0.8) = 0.9$

$w_{41}(n) = 0.9 + 0.5(0-0.9) = 0.45$

∴ Updated weight matrix after 3rd input pattern:

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



(5)

4th input vector:

for ~~inp~~ $x = [0\ 0\ 0\ 1]$

Euclidean distance:

$D(1) = \sum_{i=1}^{4} (w_{i1} - x_i)^2$

$= (0.05-0)^2 + (0.6-0)^2 + (0.9-0)^2 + (0.45-1)$

$= 1.475$

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

$= 1.811$

$\because D(1) < D(2) \therefore D(1)$ is $\min^n$

Hence, winning cluster unit is $Y_1$, $J = 1$

Now, update weights on winning cluster $J = 1$

$w_{i1}(new) = w_{i1}(old) + \alpha[x_i - w_{i1}(old)]$

$w_{11}(n) = 0.05 + 0.5(0 - 0.05) = 0.025$

$w_{21}(n) = 0.6 + 0.5[0 - 0.6] = 0.3$

$w_{31}(n) = 0.9 + 0.5(0 - 0.9) = 0.45$

$w_{41}(n) = 0.45 + 0.5(1 - 0.45) = 0.725$

Thus, final weight obtained after presentation of 4th input pattern is -

$$w_{ij} = \begin{bmatrix} 0.025 & 0.95 \\ 0.3 & 0.35 \\ 0.45 & 0.25 \\ 0.725 & 0.15 \end{bmatrix}$$

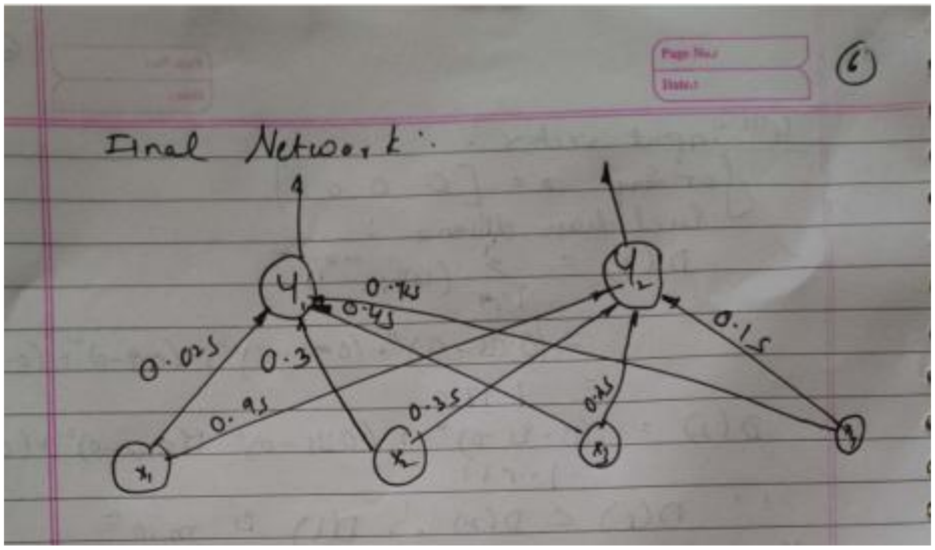
$\because$ all 4 given input patterns are presented, this is end of 1st iteration ~~of~~ i.e. 1 - epoch

Now, learning rate is

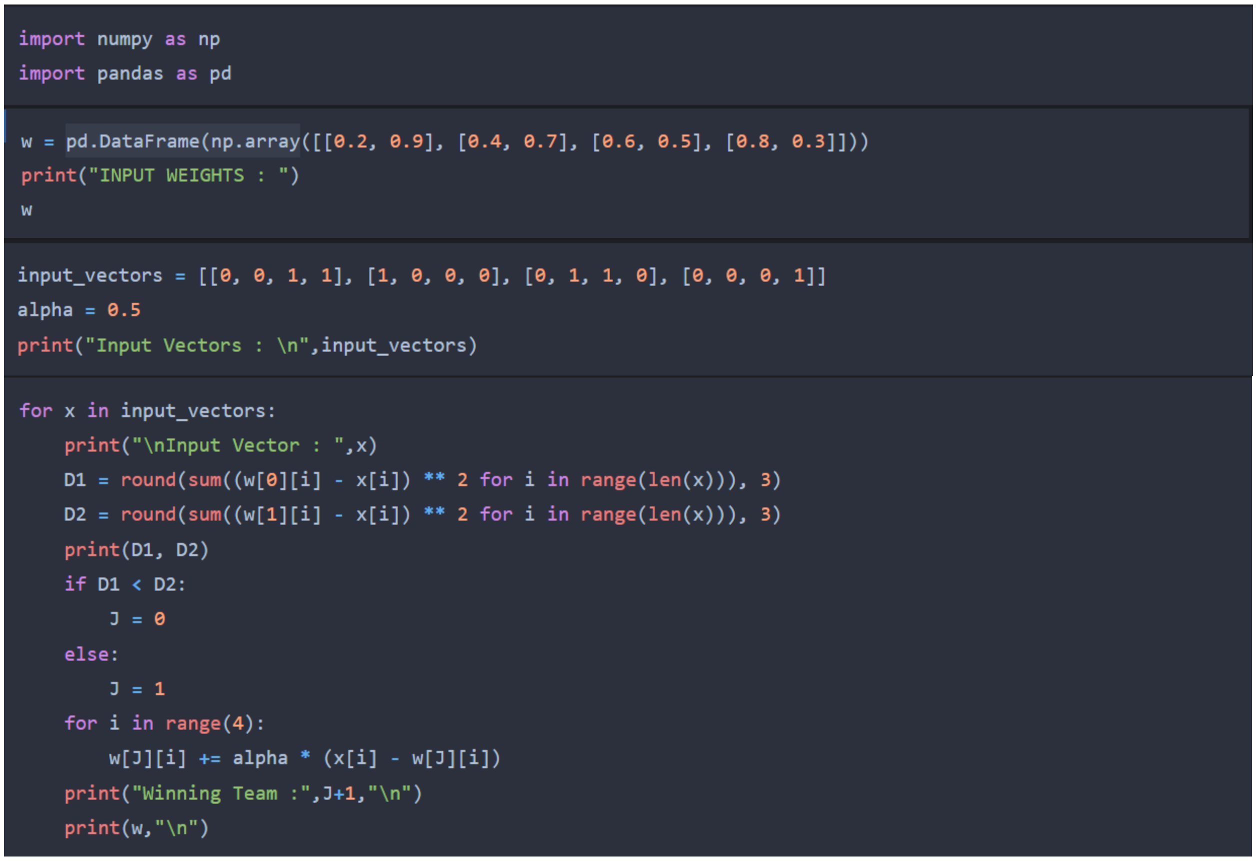
$\alpha(t+1) = 0.5\,\alpha(t)$

$\alpha(1) = 0.5\,\alpha(0) = 0.5 \times 0.5 = 0.25$

With this rate, we can proceed for 100 iterations



***Code:***



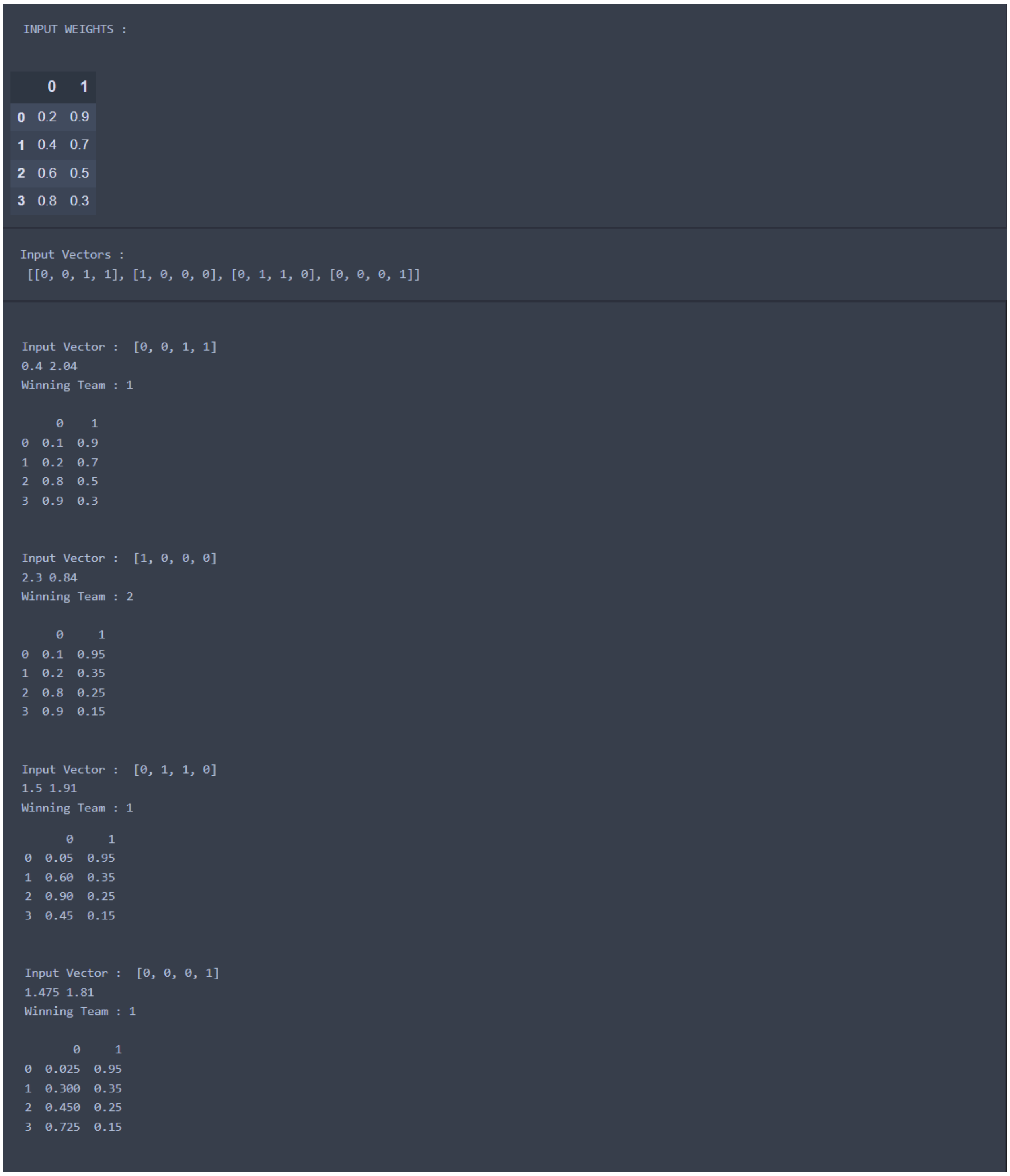
```
import numpy as np
import pandas as pd

w = pd.DataFrame(np.array([[0.2, 0.9], [0.4, 0.7], [0.6, 0.5], [0.8, 0.3]]))
print("INPUT WEIGHTS : ")
w

input_vectors = [[0, 0, 1, 1], [1, 0, 0, 0], [0, 1, 1, 0], [0, 0, 0, 1]]
alpha = 0.5
print("Input Vectors : \n",input_vectors)

for x in input_vectors:
    print("\nInput Vector : ",x)
    D1 = round(sum((w[0][i] - x[i]) ** 2 for i in range(len(x))), 3)
    D2 = round(sum((w[1][i] - x[i]) ** 2 for i in range(len(x))), 3)
    print(D1, D2)
    if D1 < D2:
        J = 0
    else:
        J = 1
    for i in range(4):
        w[J][i] += alpha * (x[i] - w[J][i])
    print("Winning Team :",J+1,"\n")
    print(w,"\n")
```

***Output:***



```
INPUT WEIGHTS :

     0    1
0  0.2  0.9
1  0.4  0.7
2  0.6  0.5
3  0.8  0.3

Input Vectors :
 [[0, 0, 1, 1], [1, 0, 0, 0], [0, 1, 1, 0], [0, 0, 0, 1]]


Input Vector :  [0, 0, 1, 1]
0.4 2.04
Winning Team : 1

     0    1
0  0.1  0.9
1  0.2  0.7
2  0.8  0.5
3  0.9  0.3


Input Vector :  [1, 0, 0, 0]
2.3 0.84
Winning Team : 2

     0     1
0  0.1  0.95
1  0.2  0.35
2  0.8  0.25
3  0.9  0.15


Input Vector :  [0, 1, 1, 0]
1.5 1.91
Winning Team : 1

      0     1
0  0.05  0.95
1  0.60  0.35
2  0.90  0.25
3  0.45  0.15


Input Vector :  [0, 0, 0, 1]
1.475 1.81
Winning Team : 1

       0     1
0  0.025  0.95
1  0.300  0.35
2  0.450  0.25
3  0.725  0.15
```

***Conclusion:***

In this experiment, I have learnt and implemented KSOFM algorithm and how to select the winning team and then update the weights of the under winning team.