Experiment 7

NAME	**Shreya Shetty**
UID	**2019140059**
CLASS	**TE IT**
BATCH	**B**
ACADEMIC YEAR	**2021-22**
SUBJECT	**SC (Soft Computing)**
COURSE CODE	**IT312**
EXPERIMENT NO.	**7**

Aim:

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

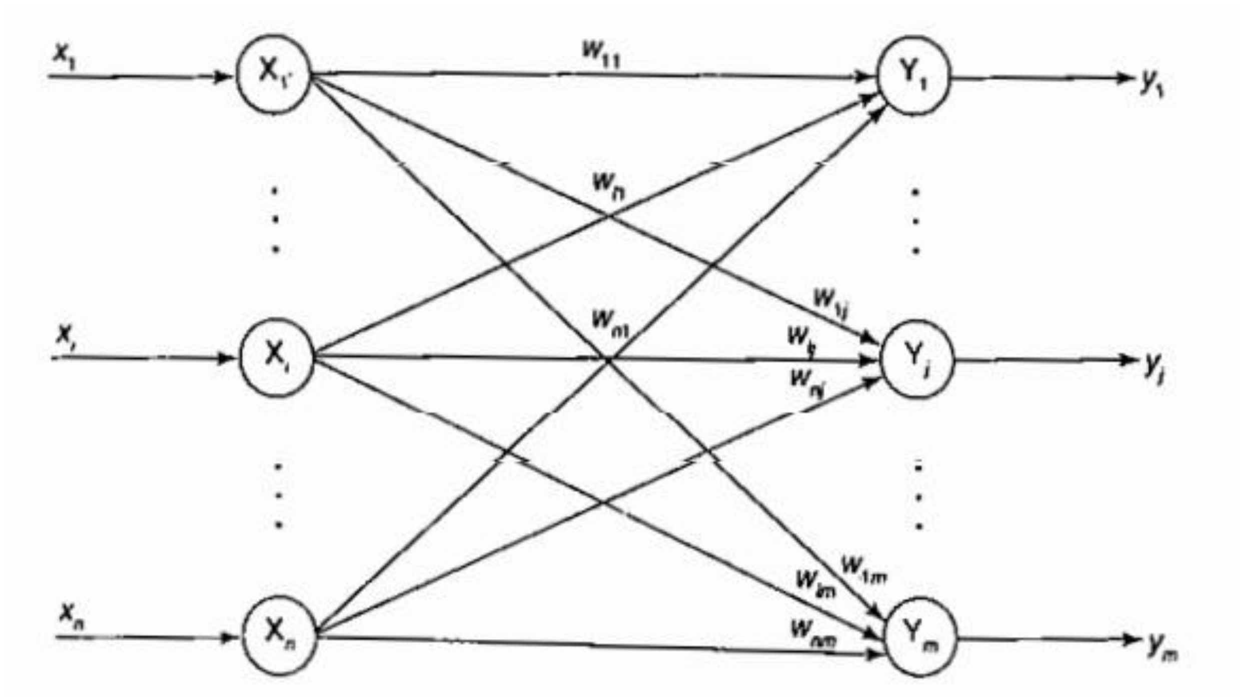
Theory:

LVQ

- Learning vector quantization (LVQ) is a process of classifying the patterns, wherein each output unit represents a particular class. Here, for each class several units should be used.
- The output unit weight vector is called the reference vector or code book vector for the class which the unit represents.
- This is a special case of competitive net, which uses supervised learning methodology.
- During training, the output units are found to be positioned co approximate the decision surfaces of the existing Bayesian classifier.
- Here, the set of training patterns with known classifications is given to the network, along with an initial distribution of the reference vectors.
- When the training process is complete, an LVQ net is found to classify an input vector by assigning it to the same class as that of the output unit, which has its weight vector very close to the input vectors.
- Thus, LVQ is classifier paradigm that adjusts the boundaries between categories to minimize existing misclassification.
- LVQ is used for optical character recognition, converting speech into phonemes and other applications as well.
- LVQ net may resemble KSOFM net. Unlike LVO, KSOFM output nodes do not correspond to the known classes but rather correspond to unknown clusters that the KSOFM finds in the data autonomously.

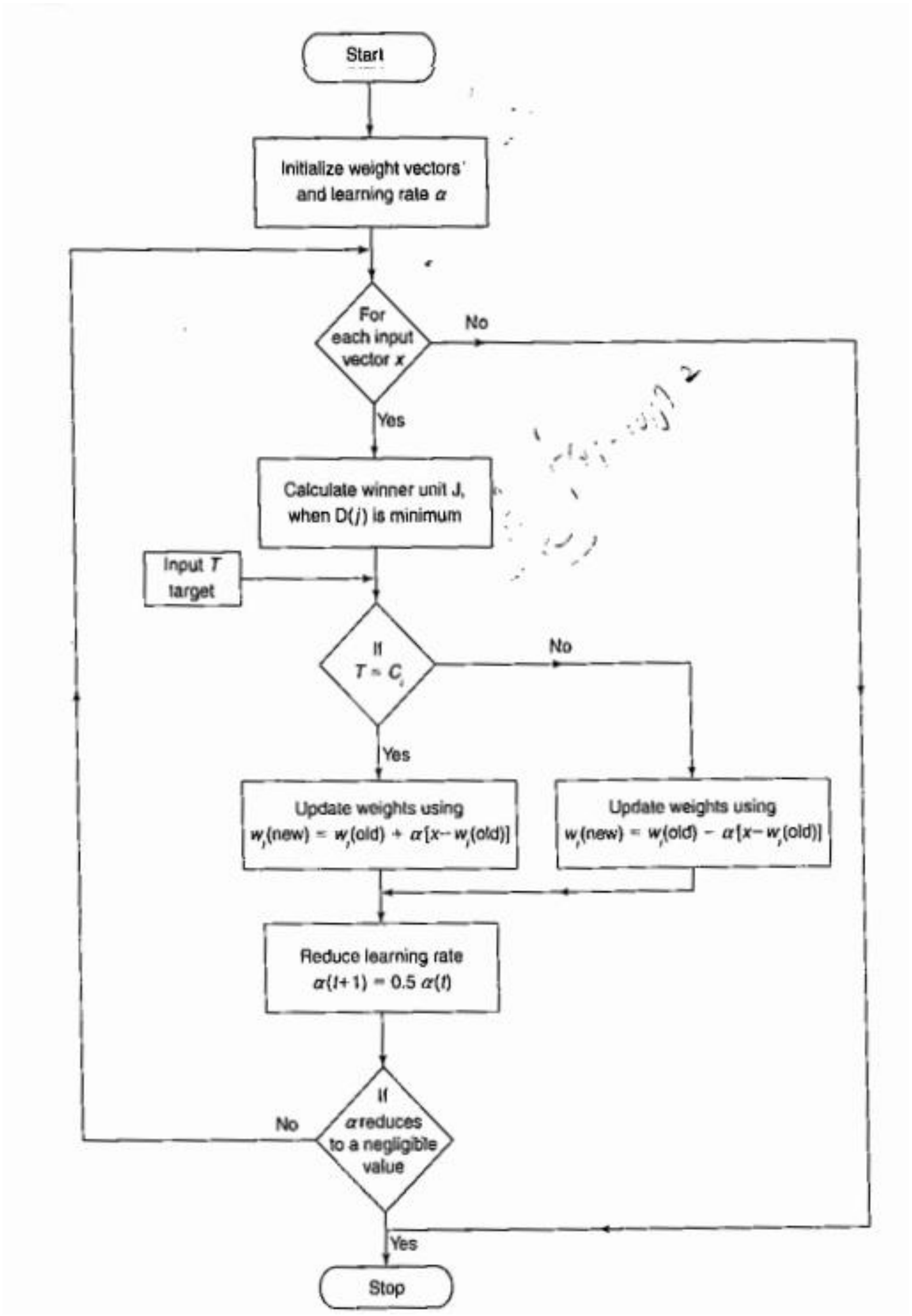
LVQ Architecture

Below figure shows the architecture of LVQ, which is almost the same as that of KSOFM, with the difference being that in the case of LVQ/the topological structure at the output unit is not being considered. Here, each output unit has knowledge about what a known class represents.
From below figure, it can be noticed that there exists input layer with "n" units and output layer with “m" units. The layers are found to be fully interconnected with weighted linkage acting over the links.



Flowchart for LVQ

The parameters used for the training process of a LVQ include the following:
x= training vector (x1,..., ..., xo)
T= category or class for the training vector x
w_j = weight vector for j^{th} output unit (w_{1j}..., w_{ij}..., w_{nj})
c_j = cluster or class or category associated with j^{th} output unit.
The Euclidean distance of j^{th} output unit is $D(j) = \sum(x_i - w_{ij})^2$
The flowchart indicating the flow of training process is shown in below figure.


Start
Initialize weight vectors and learning rate α
For each input vector x
No
Yes
Calculate winner unit J, when D(j) is minimum
Input T target
If $T = C_j$
No
Yes
Update weights using $w_j(\text{new}) = w_j(\text{old}) + \alpha[x - w_j(\text{old})]$
Update weights using $w_j(\text{new}) = w_j(\text{old}) - \alpha[x - w_j(\text{old})]$
Reduce learning rate $\alpha(t+1) = 0.5\,\alpha(t)$
If α reduces to a negligible value
No
Yes
Stop

Training Algorithm

In case of training, a set of training input vectors with a known classification is provided with some initial distribution of reference vector. Here, each output unit will have a known class. The objective of the algorithm is to find the output unit that is closest to the input vector.

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

- From the given sec 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 α.

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,

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

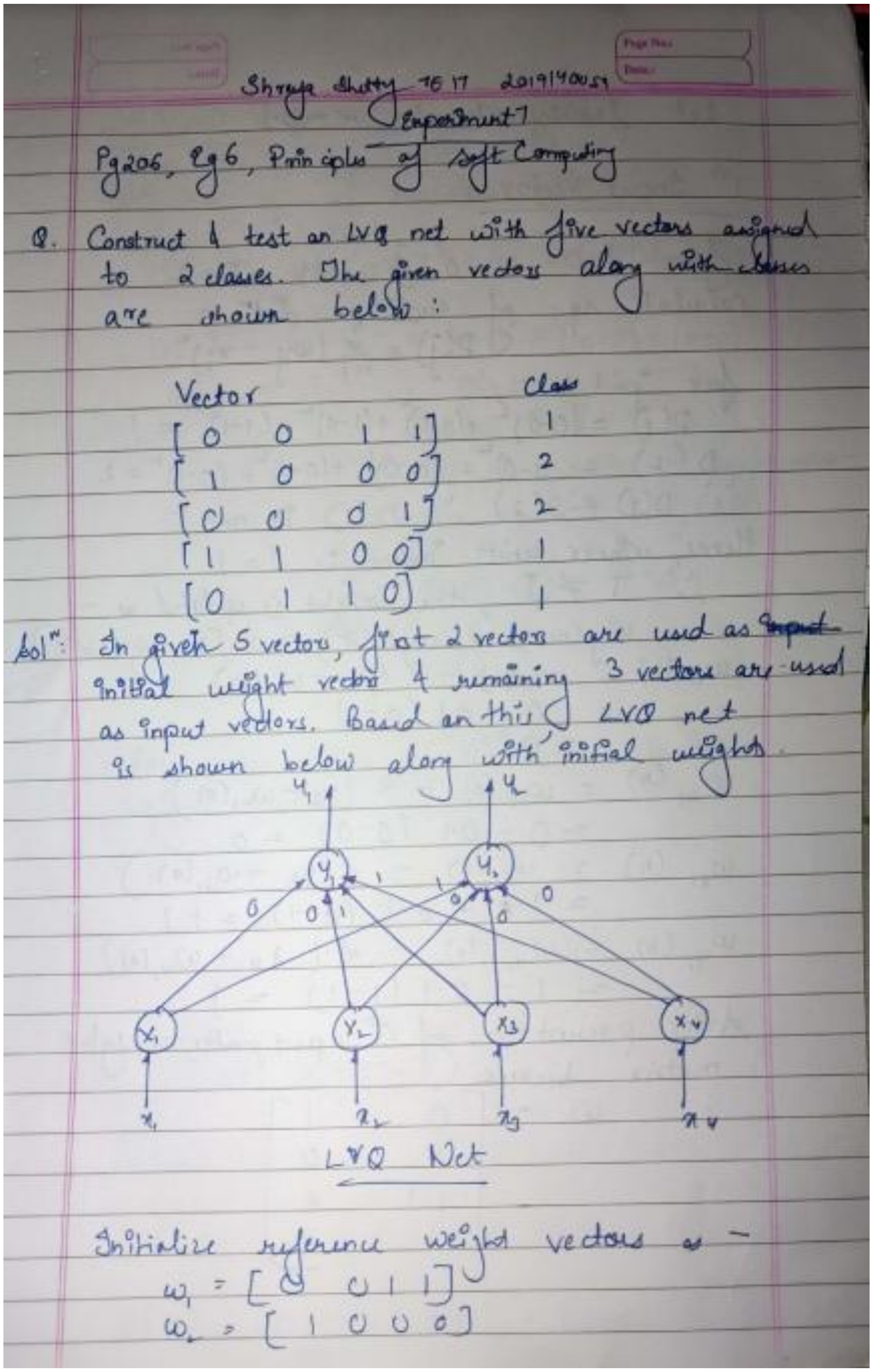
Find the winning unit index J, when D(J) is minimum

Step 4: Update the weights on the winning unit, w_j using the following conditions.
If $T = c_j$, then $w_j(new) = w_j(old) + \alpha (x_i - w_j(old)]$
If $T \neq c_j$, then $w_j(new) = w_j(old) - \alpha (x_i – w_j(old)]$

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.)

Solved Example:



Shreya Shetty TE IT 2019140059

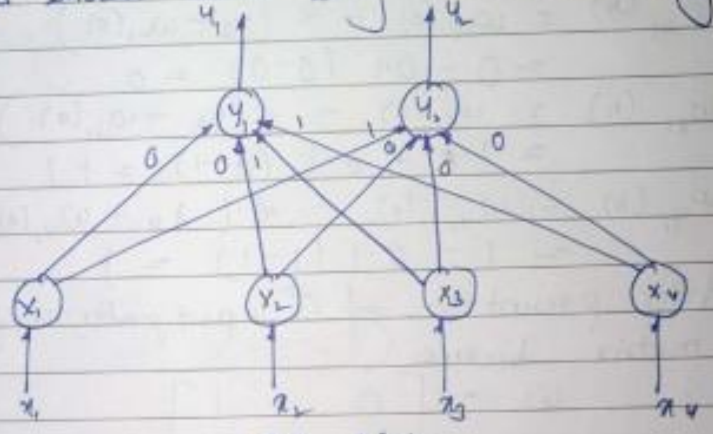
Experiment 7

Pg 206, Eg 6, Principles of Soft Computing

Q. Construct & test an LVQ net with five vectors assigned to 2 classes. The given vectors along with classes are 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

Sol^n: In given 5 vector, first 2 vectors are used as ~~input~~ initial weight vector & remaining 3 vectors are used as input vectors. Based on this LVQ net is shown below along with initial weights.

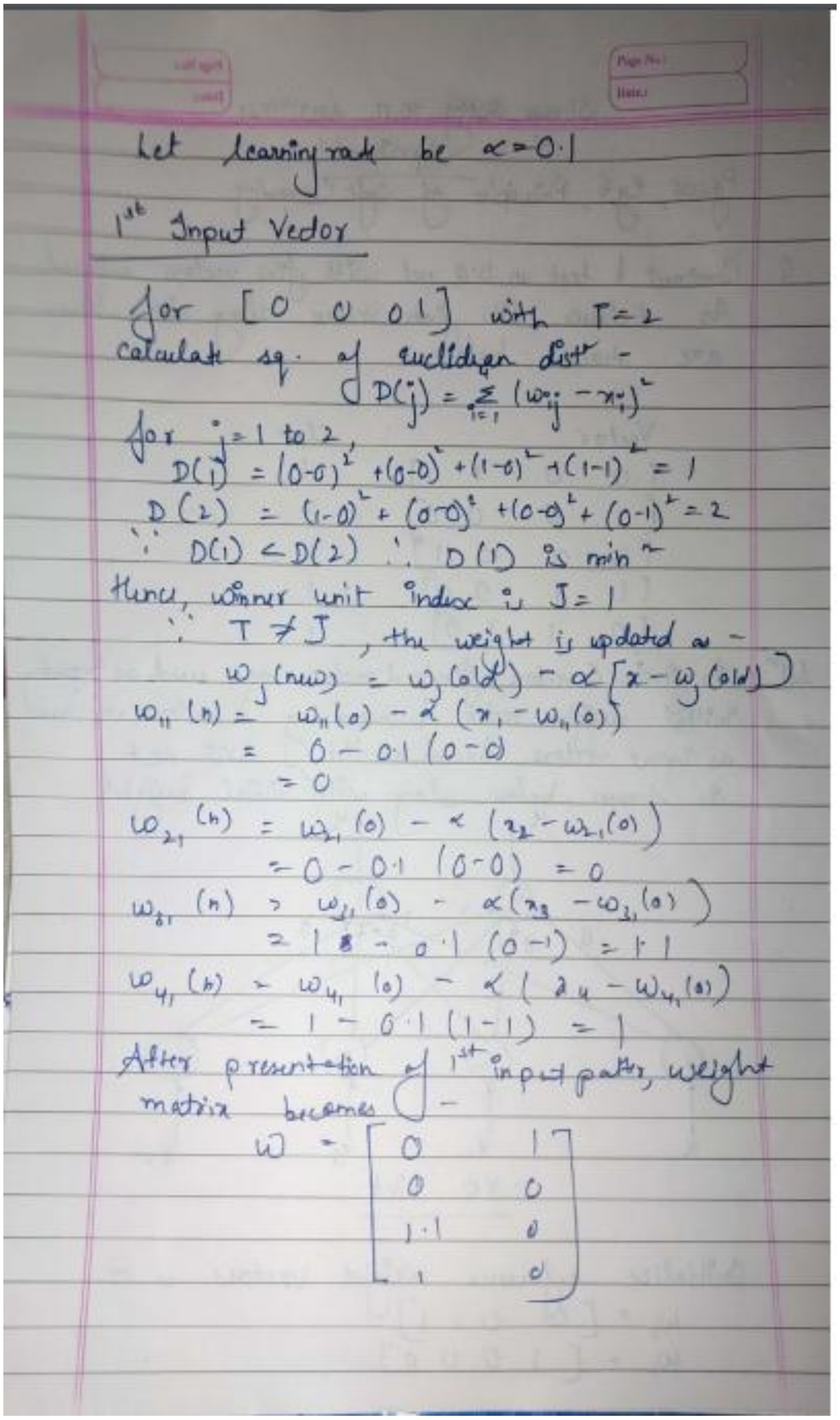


LVQ Net

Initialize reference weight vectors as —

$w_1 = [0\ 0\ 1\ 1]$

$w_2 = [1\ 0\ 0\ 0]$



Let learning rate be $\alpha = 0\cdot1$

<u>1st Input Vector</u>

for $[0 \ 0 \ 0 \ 1]$ with $T = 2$

calculate sq. of euclidean distⁿ -

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

for $j = 1$ to 2,

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

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

$\because D(1) < D(2) \therefore D(1)$ is minⁿ

Hence, winner unit index is $J = 1$

$\because T \neq J$, the weight is updated as -

$w_J(new) = w_J(old) - \alpha[x - w_J(old)]$

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

$= 0 - 0\cdot1(0-0)$

$= 0$

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

$= 0 - 0\cdot1(0-0) = 0$

$w_{31}(n) = w_{31}(0) - \alpha(x_3 - w_{31}(0))$

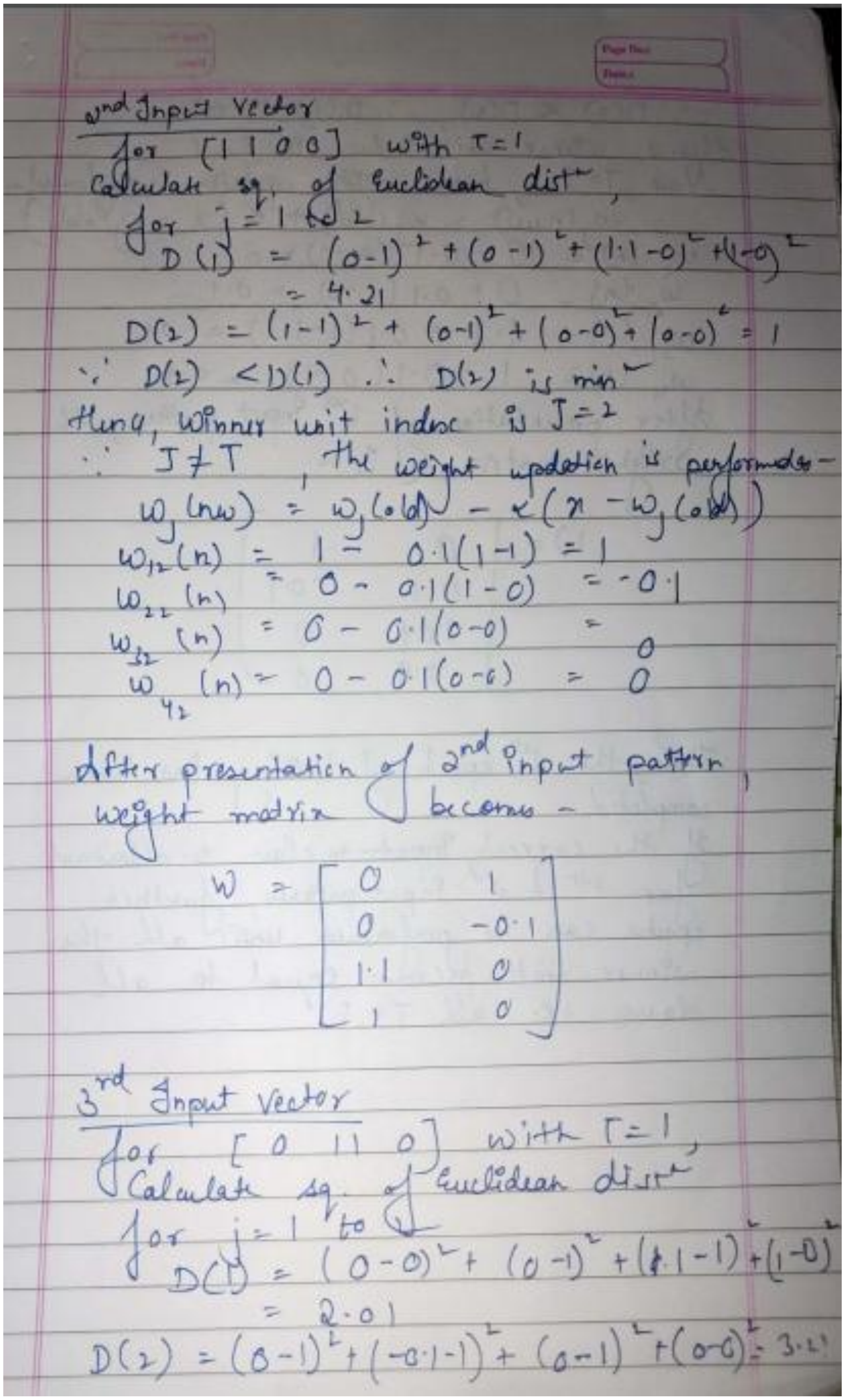
$= 1 - 0\cdot1(0-1) = 1\cdot1$

$w_{41}(n) = w_{41}(0) - \alpha(x_4 - w_{41}(0))$

$= 1 - 0\cdot1(1-1) = 1$

After presentation of 1st input patter, weight matrix becomes -

$$W = \begin{bmatrix} 0 & 1 \\ 0 & 0 \\ 1\cdot1 & 0 \\ 1 & 0 \end{bmatrix}$$



2nd Input Vector

for [1 1 0 0] with T=1

Calculate sq. of Euclidean dist^n,

for j = 1 to 2

$D(1) = (0-1)^2 + (0-1)^2 + (1.1-0)^2 + (1-0)^2$

$= 4.21$

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

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

Hence, Winner unit index is $J = 2$

$\because$ $J \neq T$, the weight updation is performed -

$w_j(new) = w_j(old) - \alpha(x - w_j(old))$

$w_{12}(n) = 1 - 0.1(1-1) = 1$

$w_{22}(n) = 0 - 0.1(1-0) = -0.1$

$w_{32}(n) = 0 - 0.1(0-0) = 0$

$w_{42}(n) = 0 - 0.1(0-0) = 0$

After presentation of 2nd input pattern, weight matrix becomes -

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

3rd Input Vector

for [0 1 1 0] with T=1,

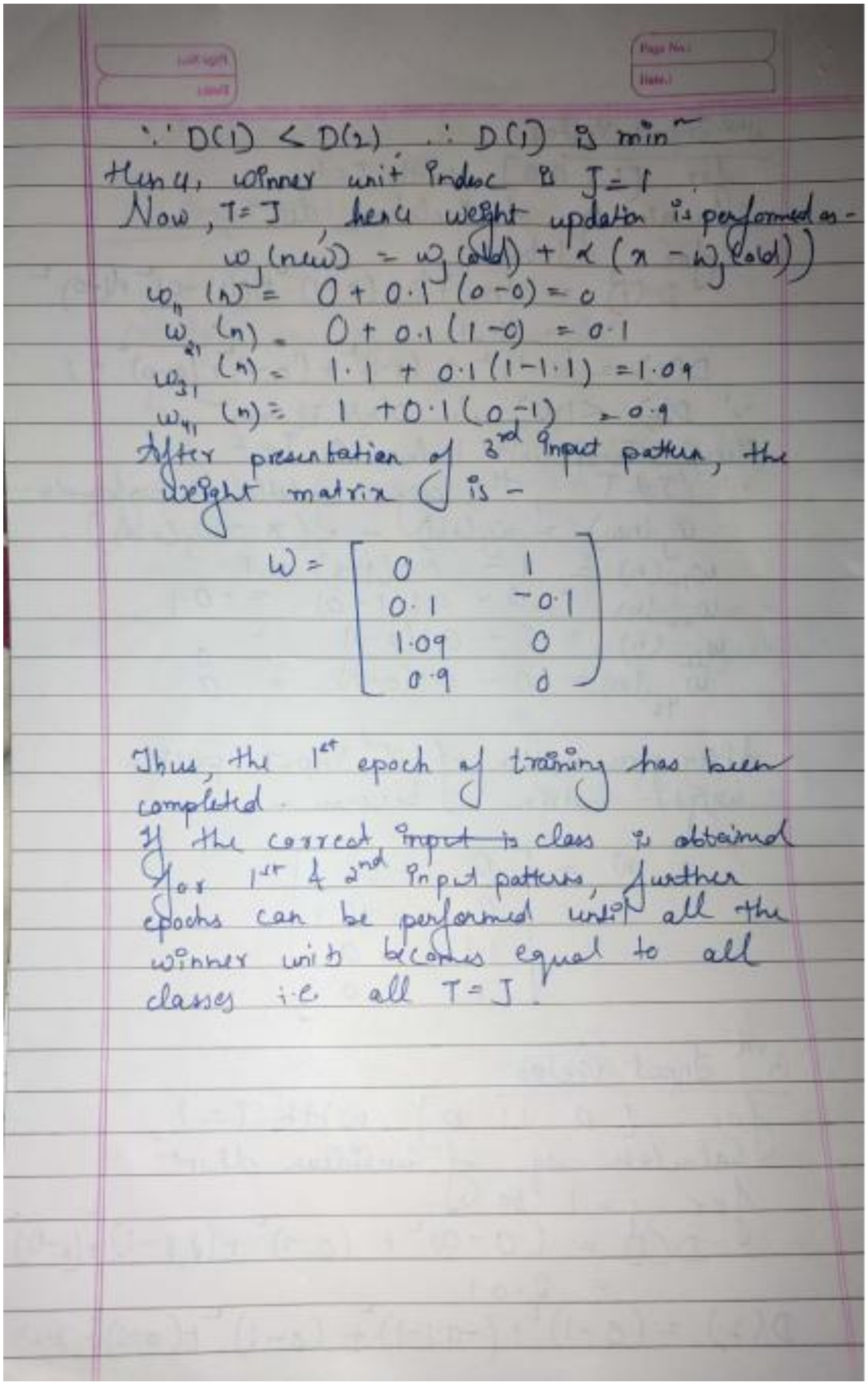
Calculate sq. of Euclidean dist^n

for j = 1 to 2

$D(1) = (0-0)^2 + (0-1)^2 + (1.1-1)^2 + (1-0)^2$

$= 2.01$

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



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

Hence, winner unit index is $J = 1$.

Now, $T = J$, hence weight updation is performed as -

$$w_j(new) = w_j(old) + \alpha(x - w_j(old))$$

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

$w_{21}(n) = 0 + 0.1(1-0) = 0.1$

$w_{31}(n) = 1.1 + 0.1(1-1.1) = 1.09$

$w_{41}(n) = 1 + 0.1(0-1) = 0.9$

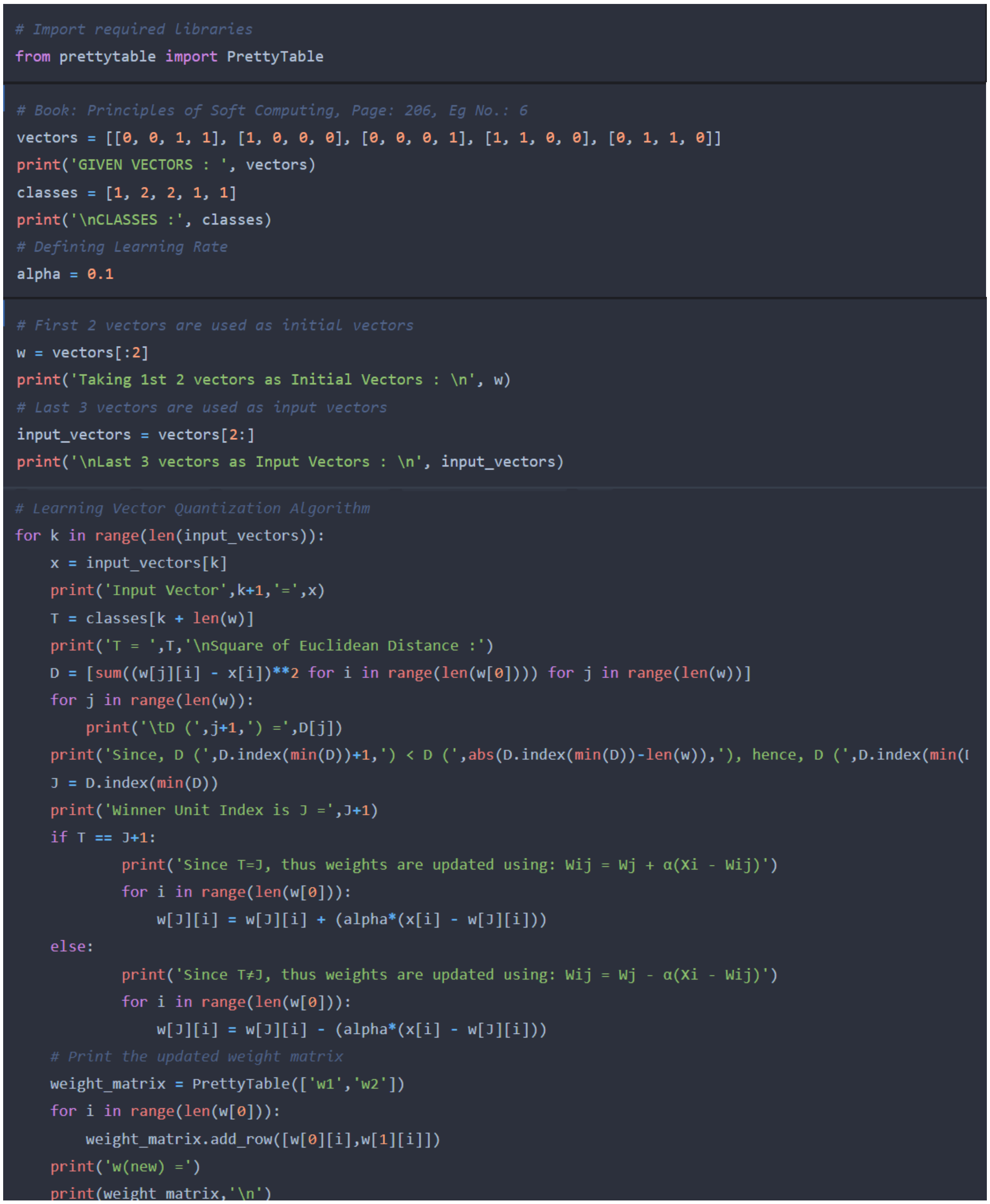
After presentation of 3rd input pattern, the weight matrix is -

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

Thus, the 1st epoch of training has been completed.

If the correct ~~input~~ is class is obtained for 1st & 2nd input patterns, further epochs can be performed until all the winner units becomes equal to all classes i.e. all $T = J$.

Code:



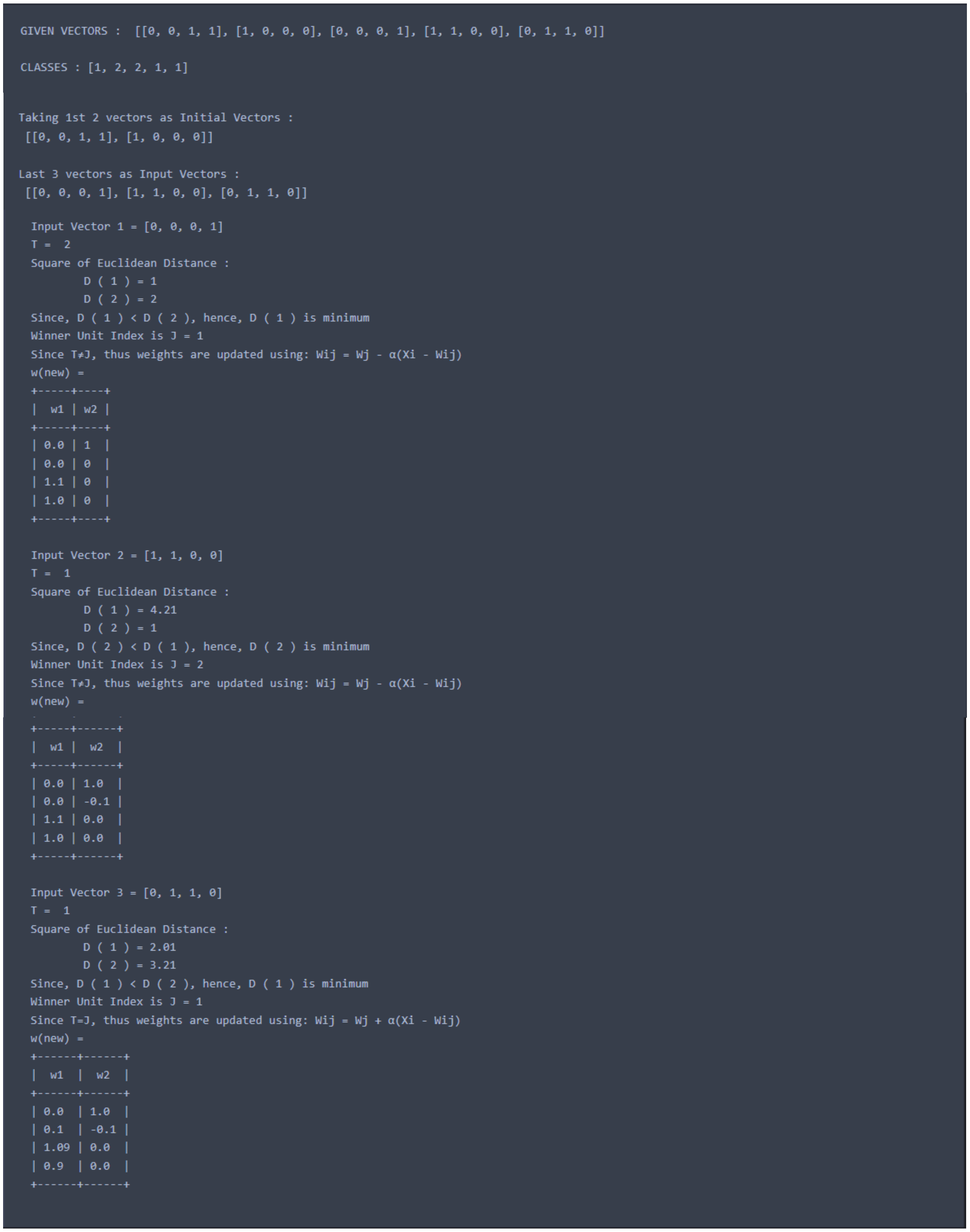
```
# Import required libraries
from prettytable import PrettyTable

# Book: Principles of Soft Computing, Page: 206, Eg No.: 6
vectors = [[0, 0, 1, 1], [1, 0, 0, 0], [0, 0, 0, 1], [1, 1, 0, 0], [0, 1, 1, 0]]
print('GIVEN VECTORS : ', vectors)
classes = [1, 2, 2, 1, 1]
print('\nCLASSES :', classes)
# Defining Learning Rate
alpha = 0.1

# First 2 vectors are used as initial vectors
w = vectors[:2]
print('Taking 1st 2 vectors as Initial Vectors : \n', w)
# Last 3 vectors are used as input vectors
input_vectors = vectors[2:]
print('\nLast 3 vectors as Input Vectors : \n', input_vectors)

# Learning Vector Quantization Algorithm
for k in range(len(input_vectors)):
    x = input_vectors[k]
    print('Input Vector',k+1,'=',x)
    T = classes[k + len(w)]
    print('T = ',T,'\nSquare of Euclidean Distance :')
    D = [sum((w[j][i] - x[i])**2 for i in range(len(w[0]))) for j in range(len(w))]
    for j in range(len(w)):
        print('\tD (',j+1,') =',D[j])
    print('Since, D (',D.index(min(D))+1,') < D (',abs(D.index(min(D))-len(w)),'), hence, D (',D.index(min(
    J = D.index(min(D))
    print('Winner Unit Index is J =',J+1)
    if T == J+1:
            print('Since T=J, thus weights are updated using: Wij = Wj + α(Xi - Wij)')
            for i in range(len(w[0])):
                w[J][i] = w[J][i] + (alpha*(x[i] - w[J][i]))
    else:
            print('Since T≠J, thus weights are updated using: Wij = Wj - α(Xi - Wij)')
            for i in range(len(w[0])):
                w[J][i] = w[J][i] - (alpha*(x[i] - w[J][i]))
    # Print the updated weight matrix
    weight_matrix = PrettyTable(['w1','w2'])
    for i in range(len(w[0])):
        weight_matrix.add_row([w[0][i],w[1][i]])
    print('w(new) =')
    print(weight_matrix,'\n')
```

Output:



```
GIVEN VECTORS :  [[0, 0, 1, 1], [1, 0, 0, 0], [0, 0, 0, 1], [1, 1, 0, 0], [0, 1, 1, 0]]

CLASSES : [1, 2, 2, 1, 1]

Taking 1st 2 vectors as Initial Vectors :
 [[0, 0, 1, 1], [1, 0, 0, 0]]

Last 3 vectors as Input Vectors :
 [[0, 0, 0, 1], [1, 1, 0, 0], [0, 1, 1, 0]]

  Input Vector 1 = [0, 0, 0, 1]
  T =  2
  Square of Euclidean Distance :
         D ( 1 ) = 1
         D ( 2 ) = 2
  Since, D ( 1 ) < D ( 2 ), hence, D ( 1 ) is minimum
  Winner Unit Index is J = 1
  Since T≠J, thus weights are updated using: Wij = Wj - α(Xi - Wij)
  w(new) =
  +-----+----+
  |  w1 | w2 |
  +-----+----+
  | 0.0 | 1  |
  | 0.0 | 0  |
  | 1.1 | 0  |
  | 1.0 | 0  |
  +-----+----+

  Input Vector 2 = [1, 1, 0, 0]
  T =  1
  Square of Euclidean Distance :
         D ( 1 ) = 4.21
         D ( 2 ) = 1
  Since, D ( 2 ) < D ( 1 ), hence, D ( 2 ) is minimum
  Winner Unit Index is J = 2
  Since T≠J, thus weights are updated using: Wij = Wj - α(Xi - Wij)
  w(new) =
  +-----+------+
  |  w1 |  w2  |
  +-----+------+
  | 0.0 | 1.0  |
  | 0.0 | -0.1 |
  | 1.1 | 0.0  |
  | 1.0 | 0.0  |
  +-----+------+

  Input Vector 3 = [0, 1, 1, 0]
  T =  1
  Square of Euclidean Distance :
         D ( 1 ) = 2.01
         D ( 2 ) = 3.21
  Since, D ( 1 ) < D ( 2 ), hence, D ( 1 ) is minimum
  Winner Unit Index is J = 1
  Since T=J, thus weights are updated using: Wij = Wj + α(Xi - Wij)
  w(new) =
  +------+------+
  |  w1  |  w2  |
  +------+------+
  | 0.0  | 1.0  |
  | 0.1  | -0.1 |
  | 1.09 | 0.0  |
  | 0.9  | 0.0  |
  +------+------+
```

Conclusion:

In this experiment, I have implemented the Learning Vector Quantization algorithm in python. LVQ is a process of classifying the patterns, wherein each output unit represents a particular class. I learned how to select the winning team find update the weights of the winning team, when input of vectors and their respective classes is given.