UNIVERSITY OF TEHRAN

COLLEGE OF ENGINEERING

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

NEURAL NETWORK & DEEP LEARNING

ASSIGNMENT#4

MOHAMMAD HEYDARI

810197494

UNDER SUPERVISION OF:

DR. AHMAD KALHOR

SCHOOL OF ELECTRICAL AND COMPUTER ENGINEERING

UNIVERSITY OF TEHRAN

June. 2022

1 CONTENTS

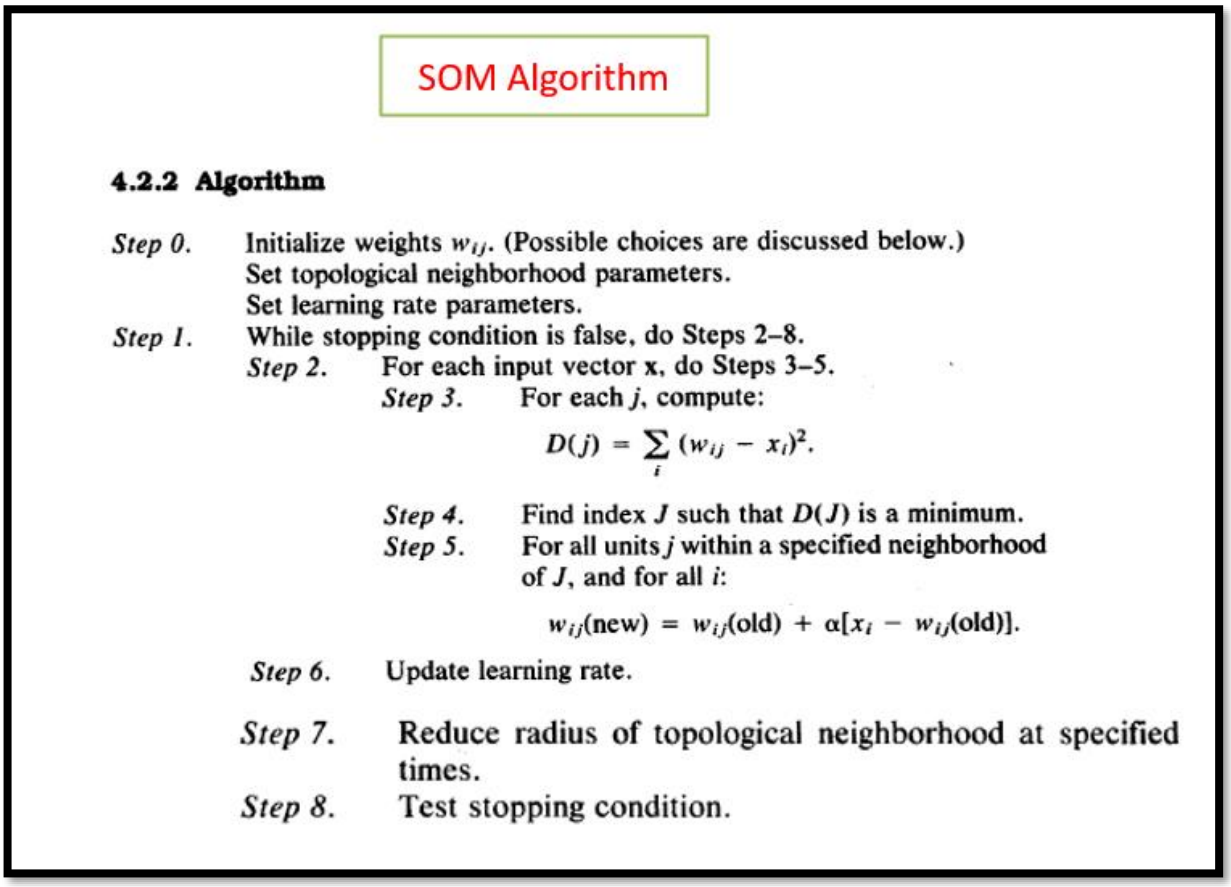
2 QUESTION #1: SOM

In this part we intend to implement SOM on fashion_mnist , first of all, we pick up the first 1000 rows as of our dataset as of our train data and also first 3000 data from test vector as of our test data.

Please note that we have 15*15 square shape formation of neurons and also we know that every pictures have a 28*28 size which is a 784 flat vector.

So, the input shape for the weight matrix is **(784,225).**

The neuron formation has an effect on stage five because we determine the neighbourhood according to the neuron formation and also neighbourhood radius and then the update step has been issued on the specific set of neurons.


SOM Algorithm

4.2.2 Algorithm

```
Step 0.  Initialize weights $w_{ij}$. (Possible choices are discussed below.)
         Set topological neighborhood parameters.
         Set learning rate parameters.
Step 1.  While stopping condition is false, do Steps 2–8.
         Step 2.  For each input vector x, do Steps 3–5.
                  Step 3.  For each j, compute:
                               $D(j) = \sum_i (w_{ij} - x_i)^2$.
                  Step 4.  Find index J such that D(J) is a minimum.
                  Step 5.  For all units j within a specified neighborhood
                           of J, and for all i:
                               $w_{ij}(\text{new}) = w_{ij}(\text{old}) + \alpha[x_i - w_{ij}(\text{old})]$.
         Step 6.  Update learning rate.
         Step 7.  Reduce radius of topological neighborhood at specified
                  times.
         Step 8.  Test stopping condition.
```

2.1 CONSTANT NEIGHBOURHOOD RADIUS

in this section I have implemented the algorithm using constant neighbourhood radius with R=1 and then you can see the results which mention in question description.

I have tested different value for learning rate and also decay rate and then I have chosen the **learning-rate=0.6** and **decay-rate=0.5.**

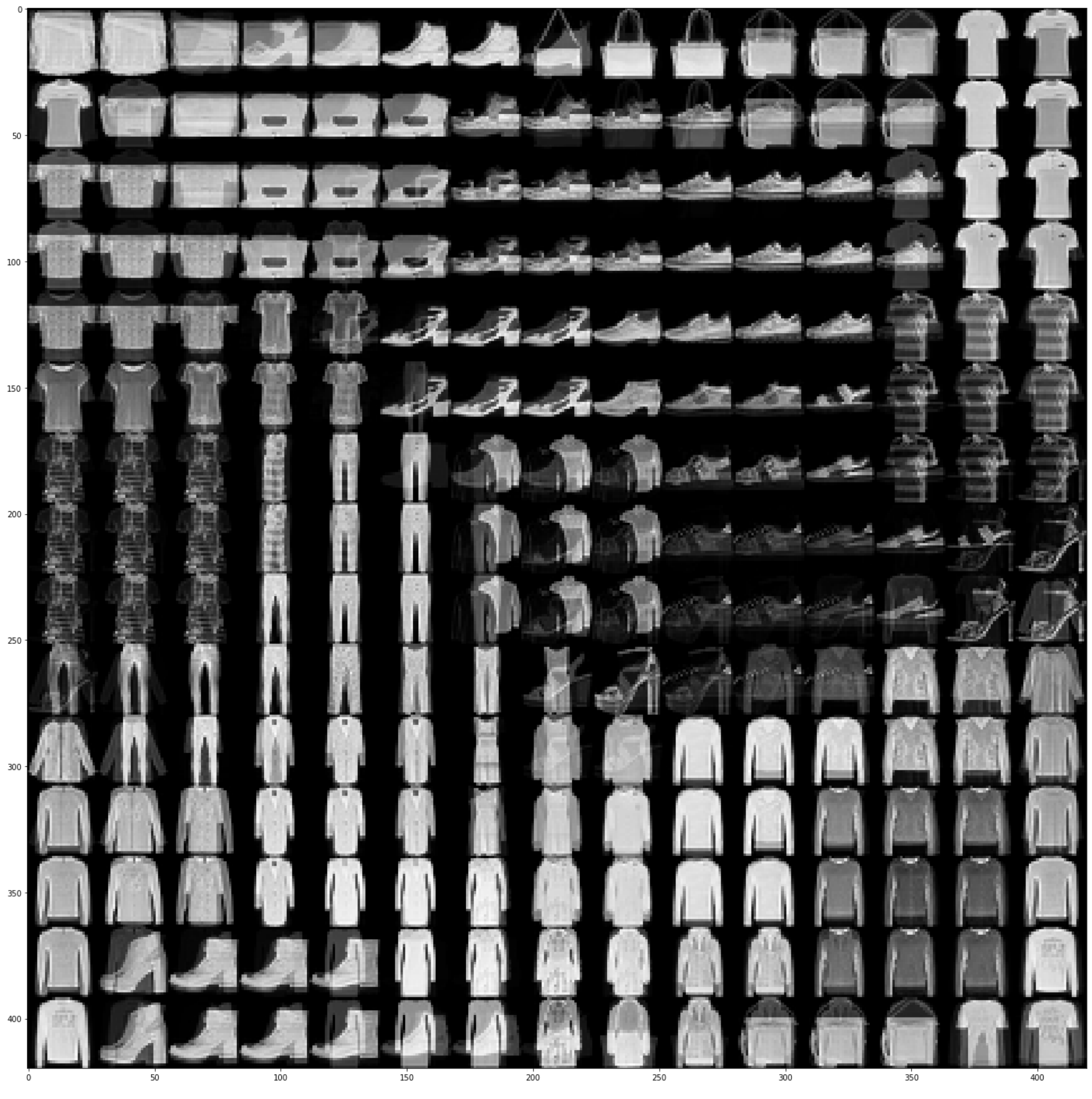


Figure1. Epoch==1

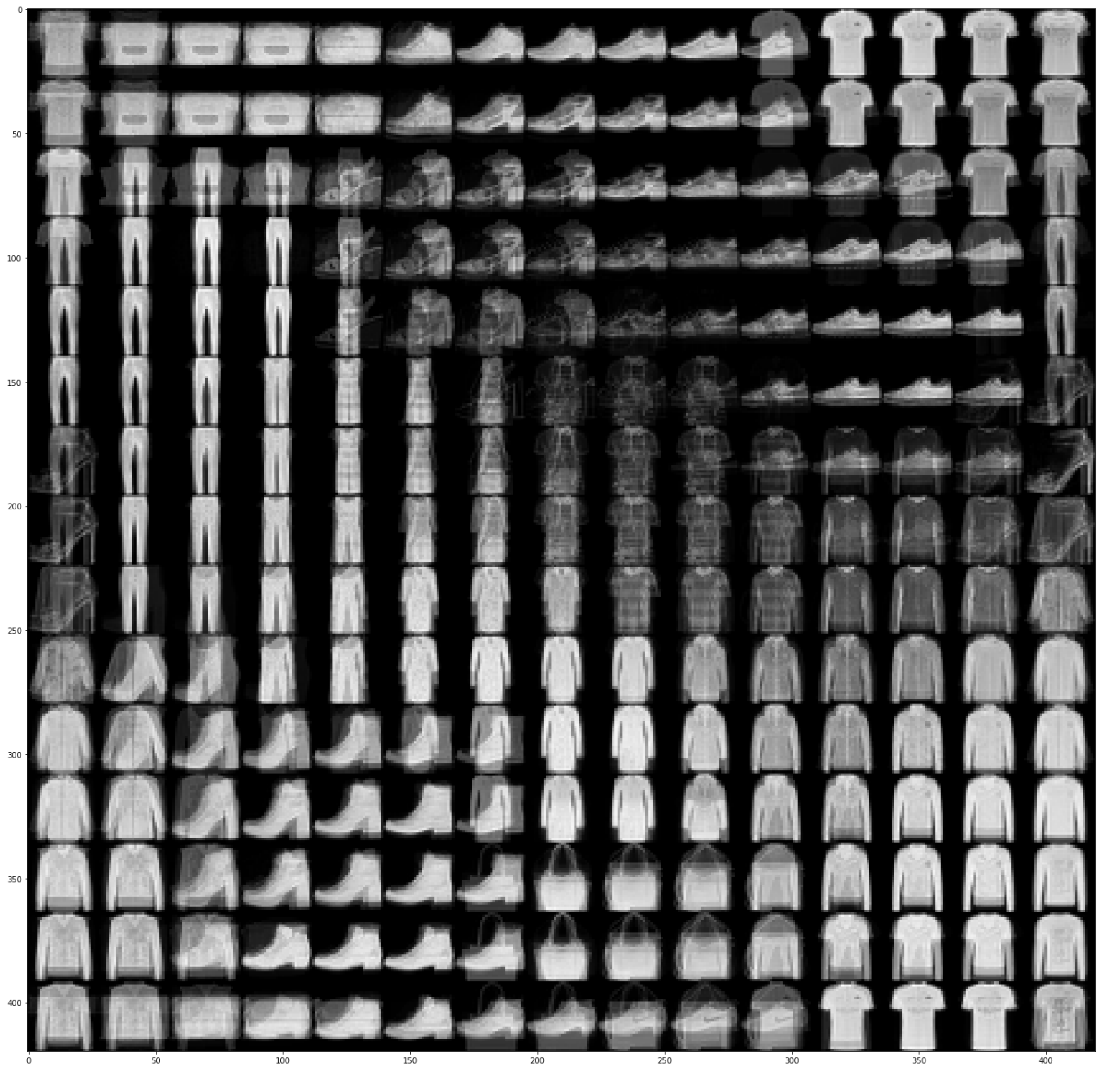


Figure2. Epoch==5

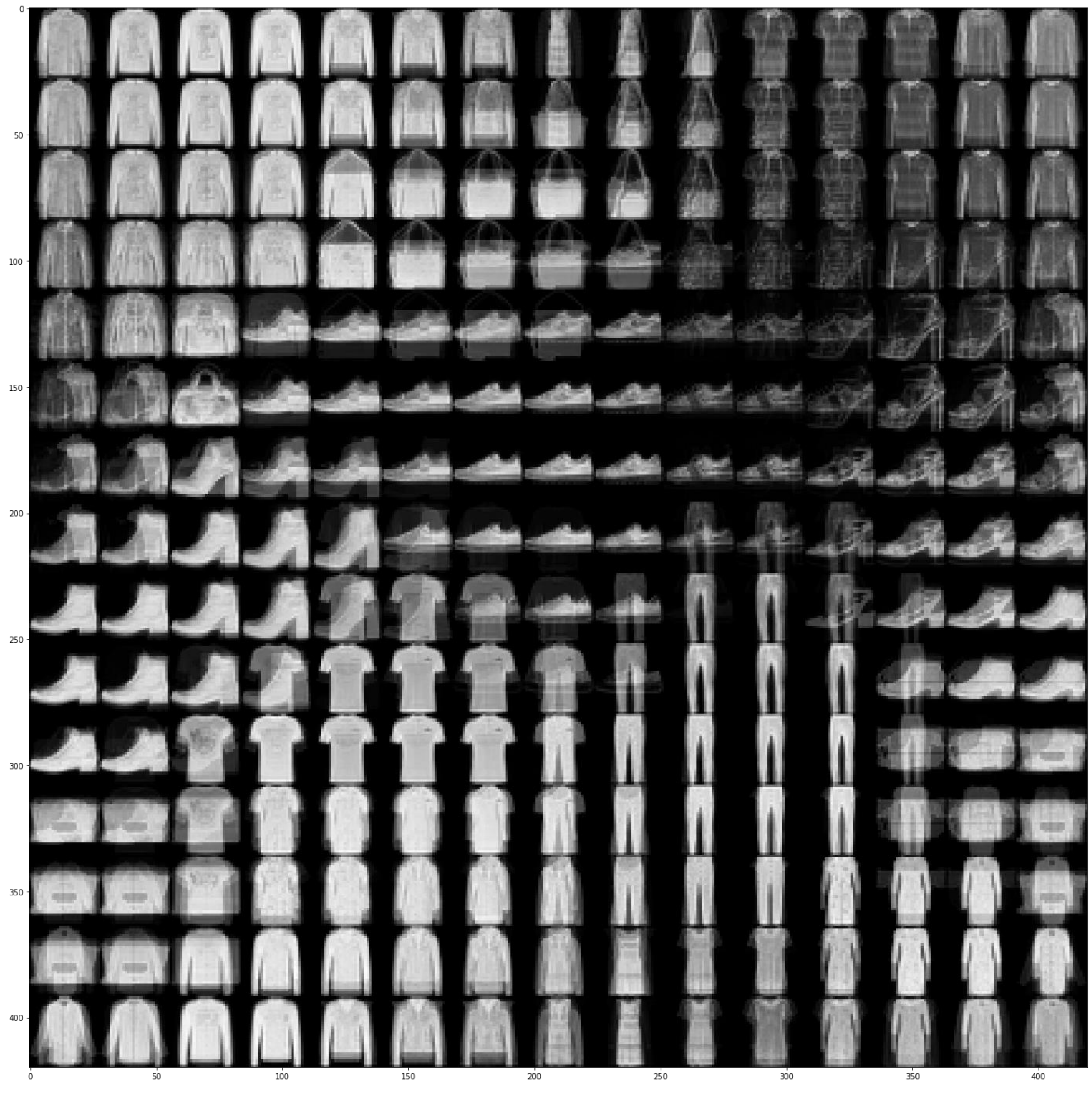
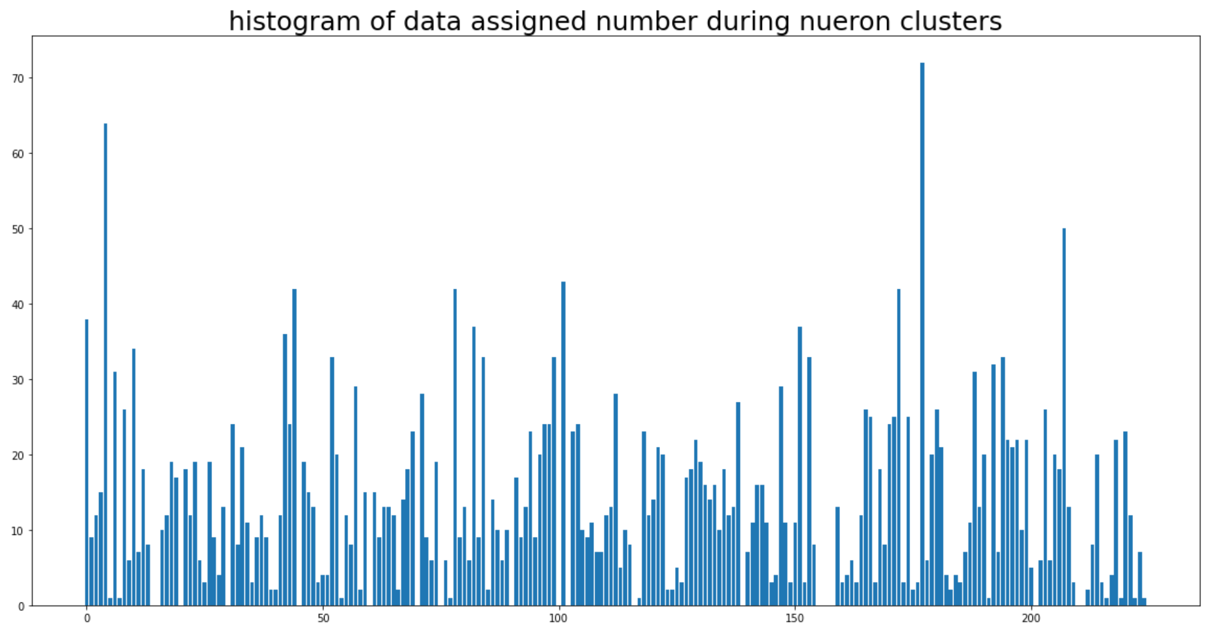
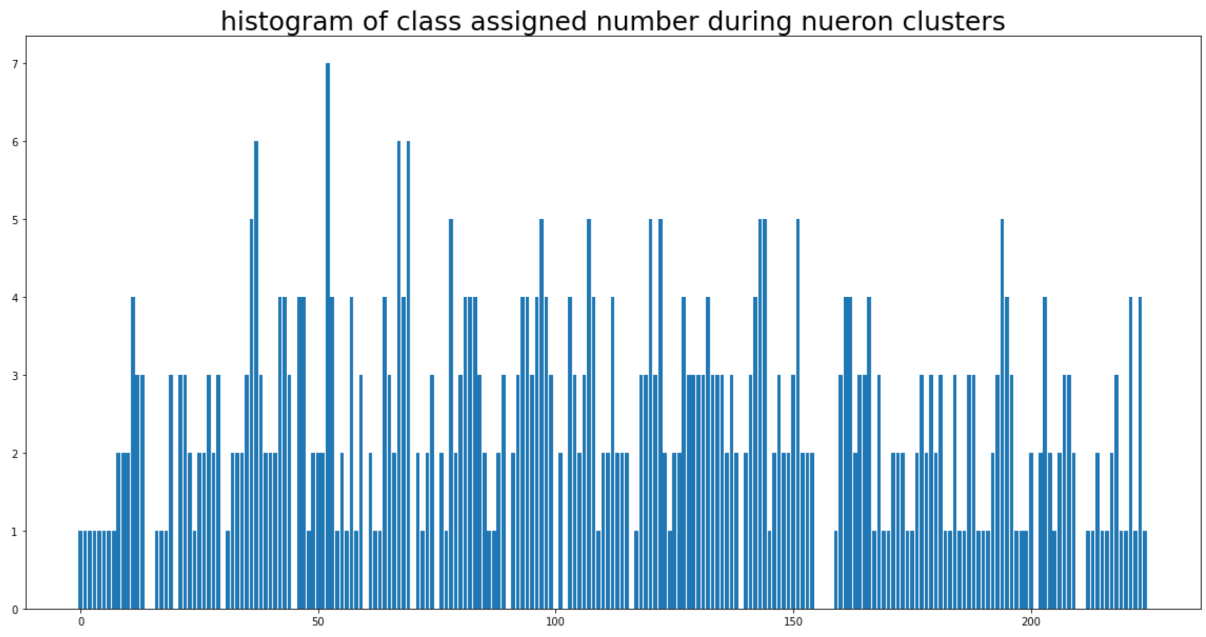


Figure3. Epoch==10


histogram of data assigned number during nueron clusters
0
10
20
30
40
50
60
70
0
50
100
150
200


histogram of class assigned number during nueron clusters
0
1
2
3
4
5
6
7
0
50
100
150
200

2.2 VARIABLE NEIGHBOURHOOD RADIUS

In this part I have reported the results for variable neighbourhood radius.

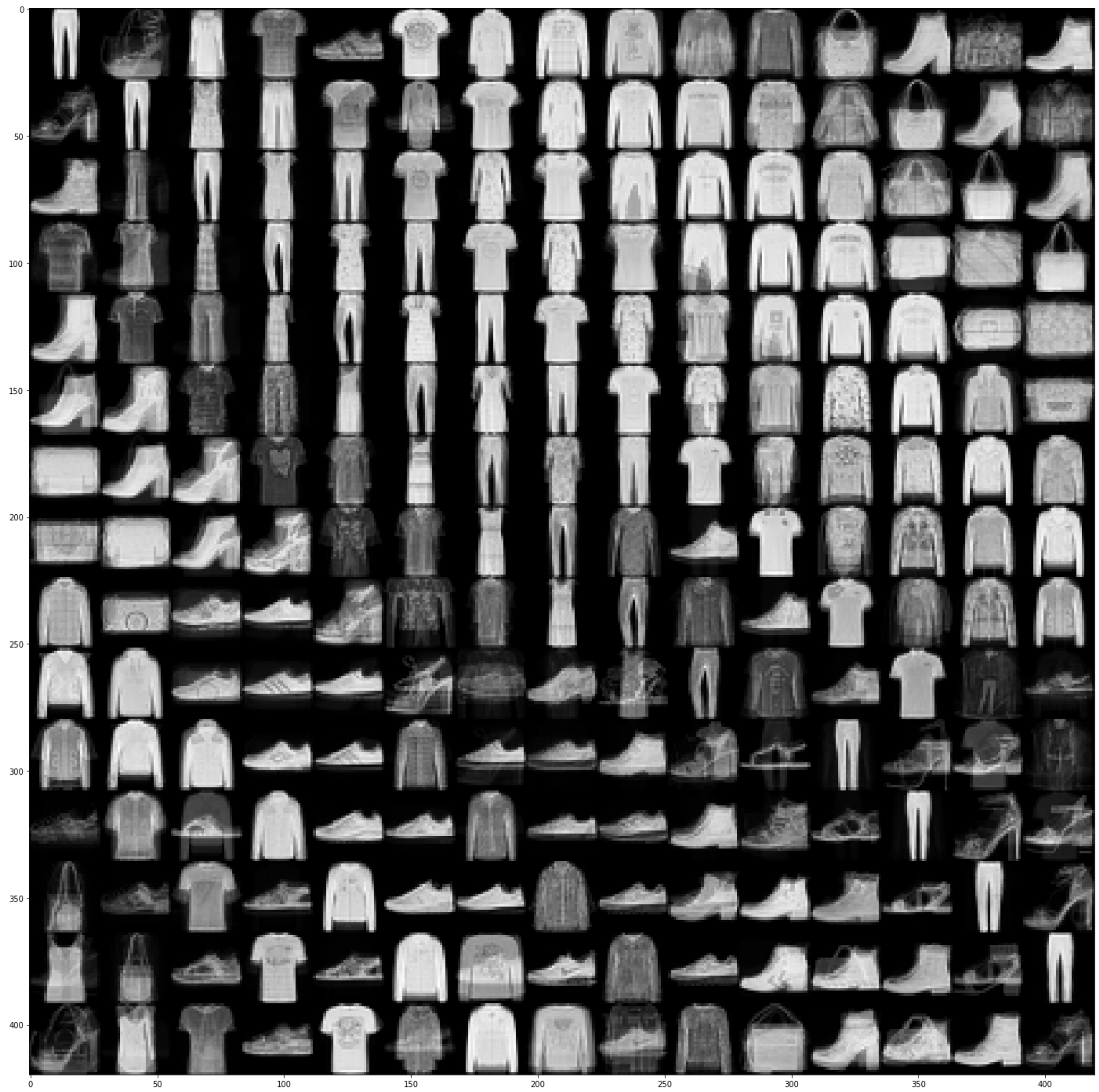


Figure4. Epoch==1

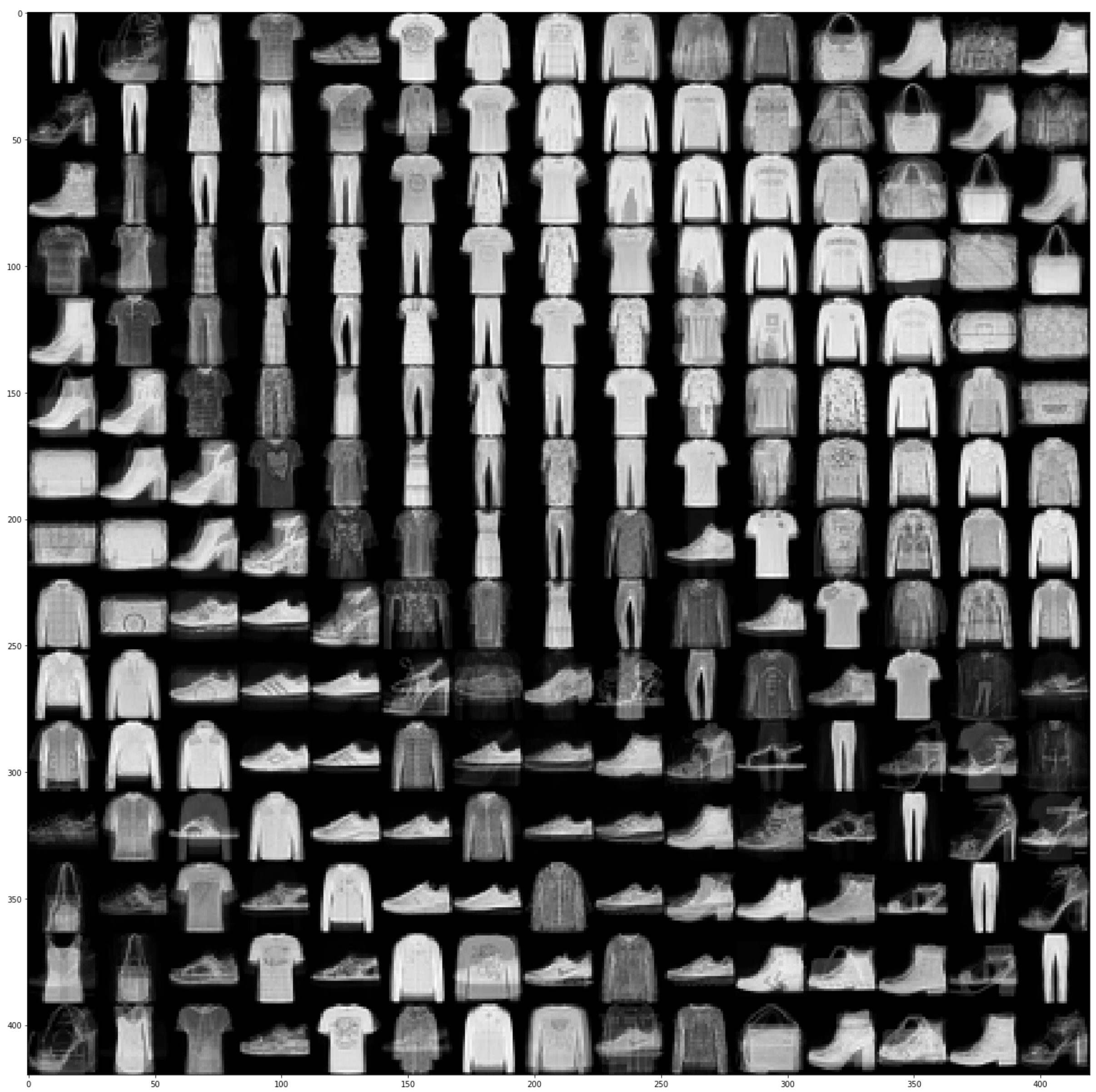


Figure5. Epoch==2

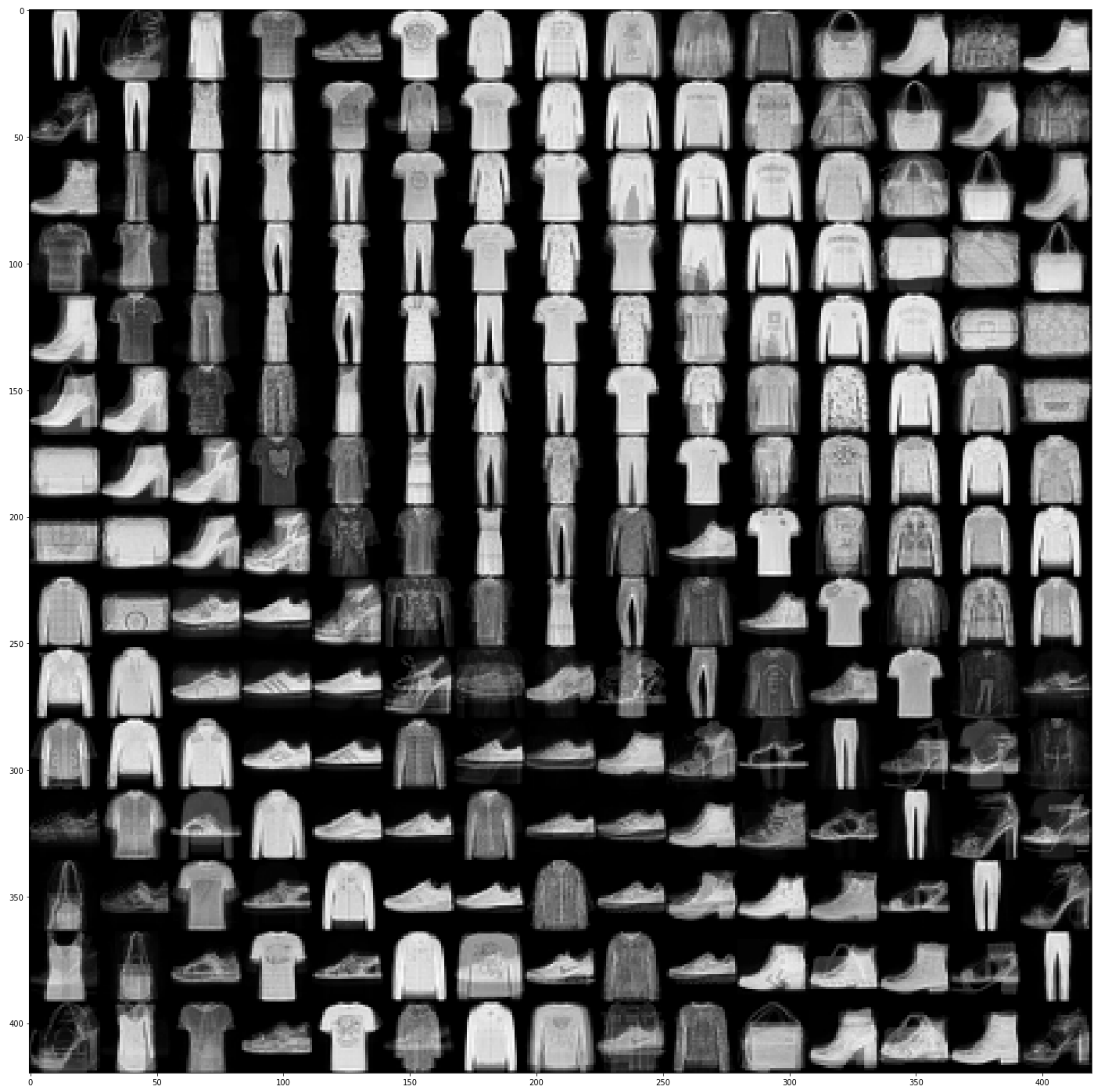


Figure6. Epoch==5

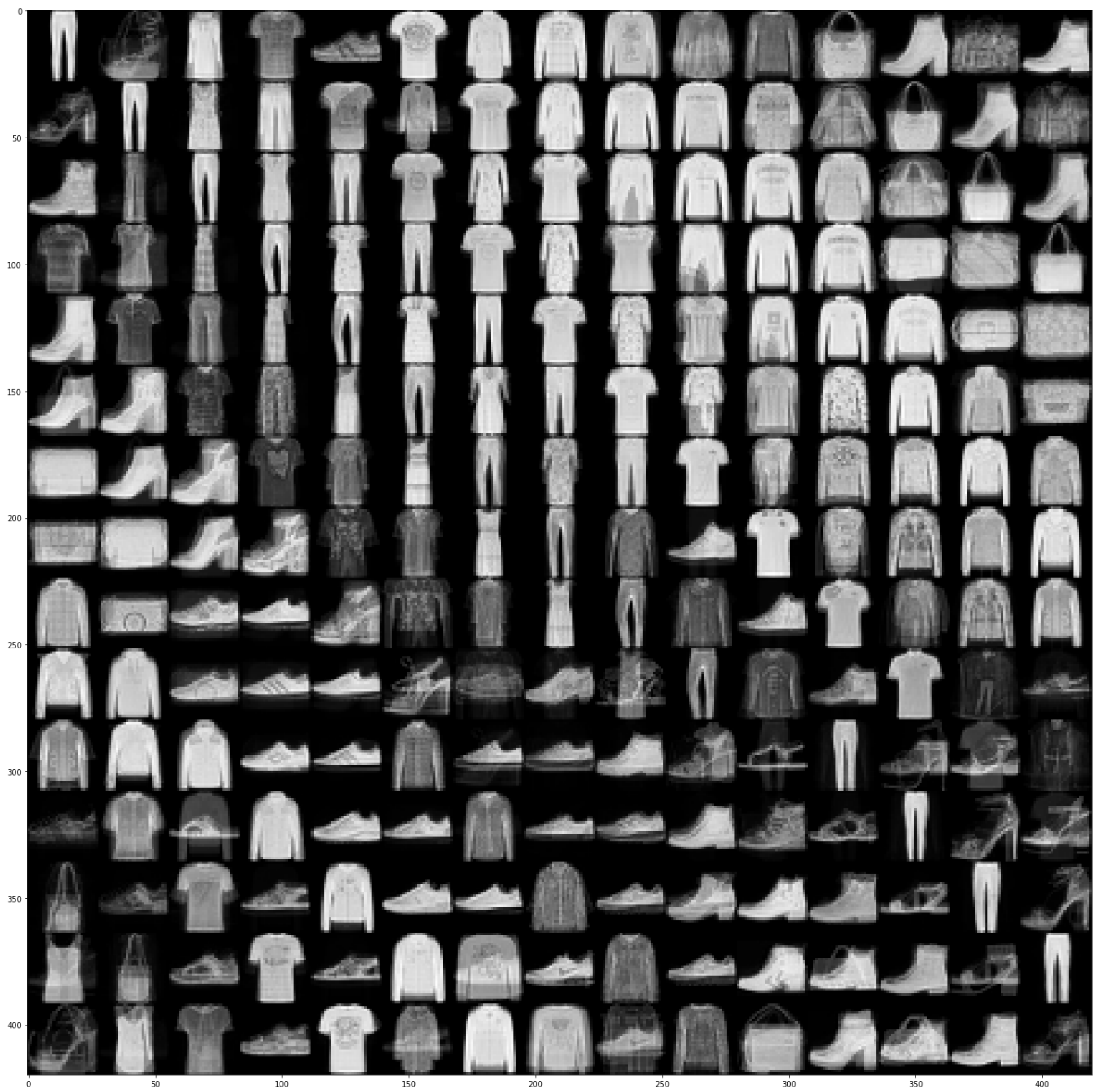
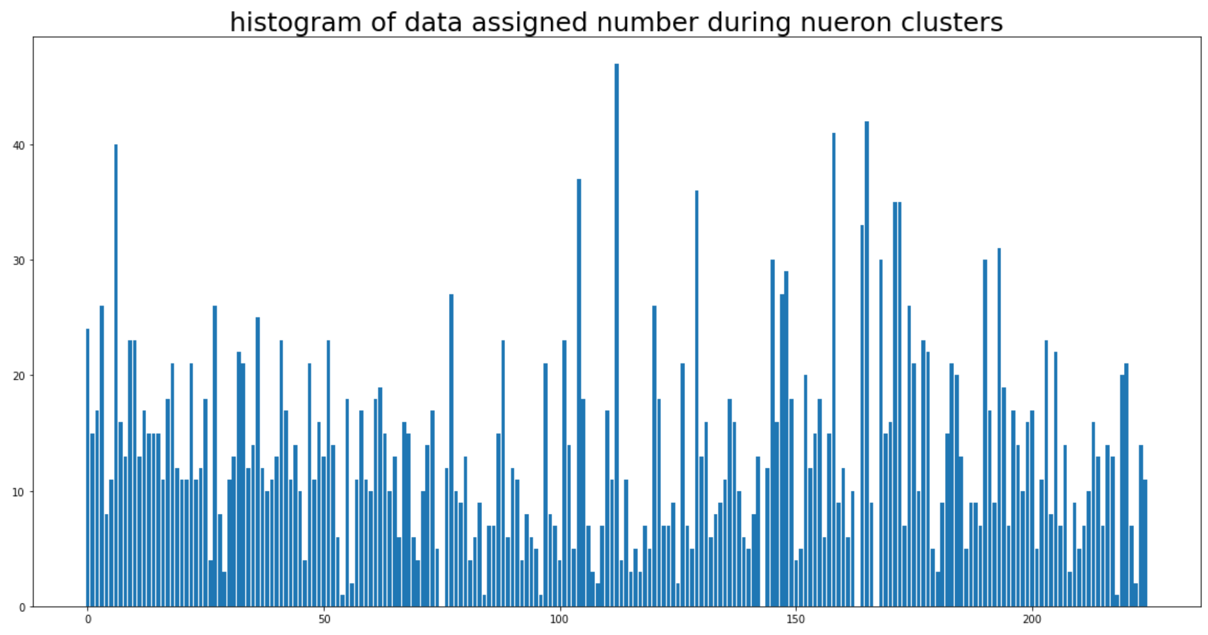
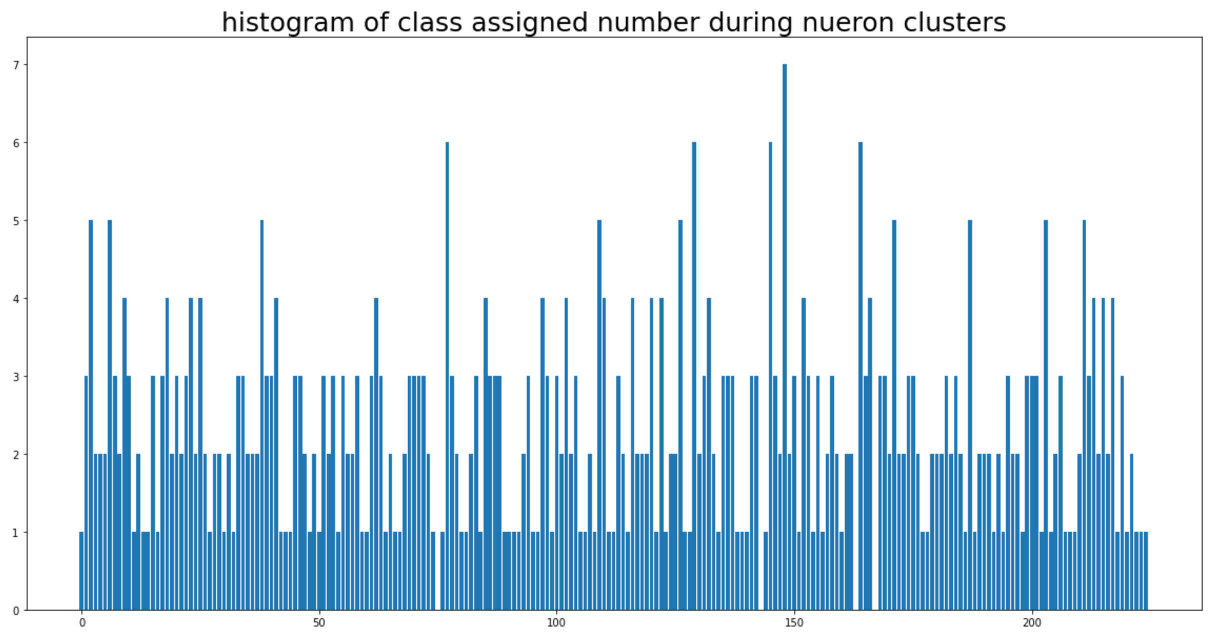


Figure7. Epoch==10


histogram of data assigned number during nueron clusters
0
10
20
30
40
0
50
100
150
200


histogram of class assigned number during nueron clusters
0
1
2
3
4
5
6
7
0
50
100
150
200

HERE IS REPRESENTATION OF OUTPUT FOR R=0:

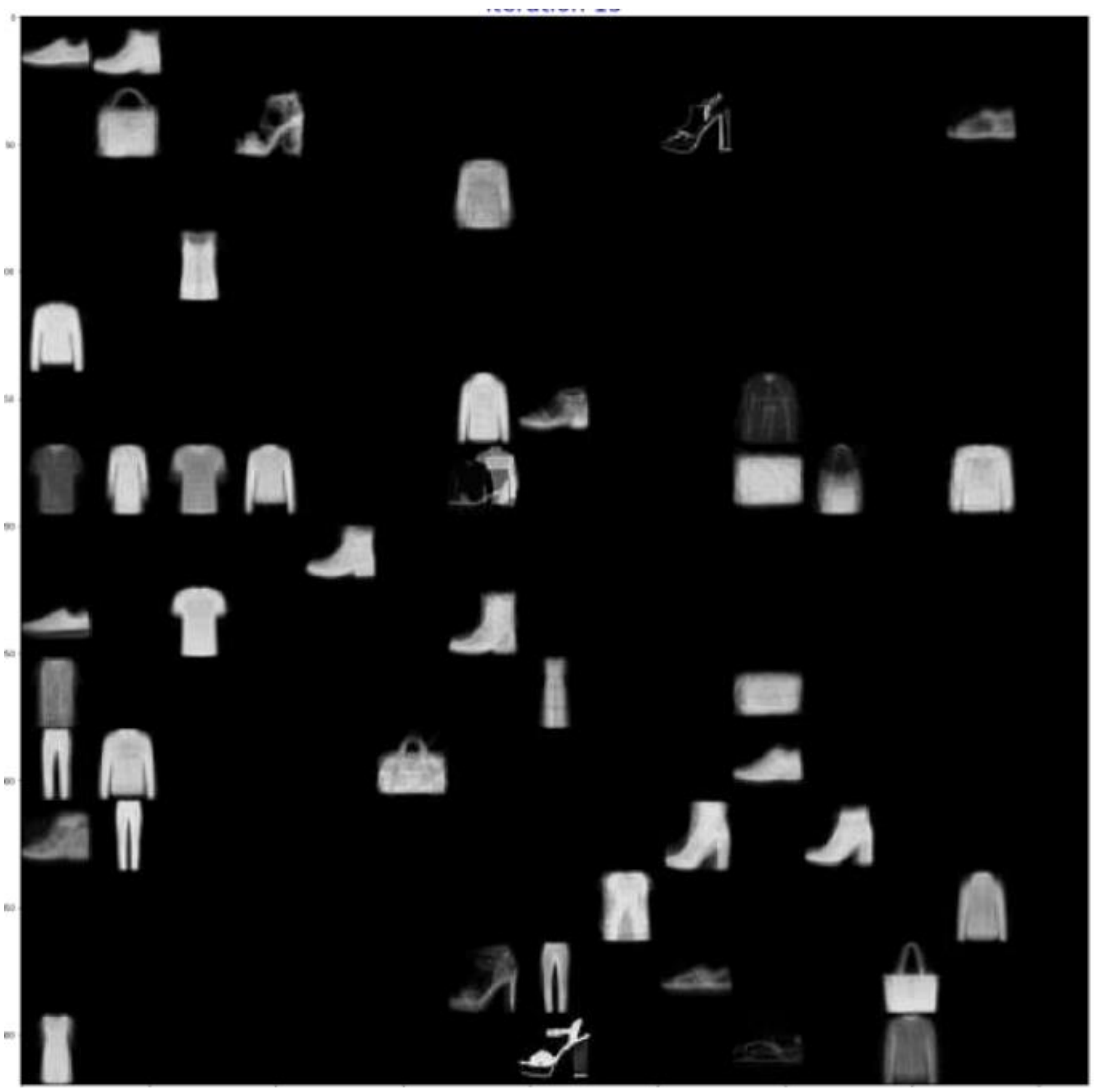


Figure8. Epoch==10

2.3 COMPARING CONSTANT RADIUS AND VARIABLE RADIUS AND INTERPRETING THE RESULTS:

we can see that the variable radius acts pretty much better because when we are dealing with variable radius in the start of algorithm it's a chance for boundary neurons to do better and then we have a better performance and better results but when we are dealing with constant radius it could be so bad for the boundary neurons because we have updating just for a near zone around the neighbourhood and then the boundary neurons tends to be updated by two neighbourhood and then we can see that some of the neurons have made by two or more classes and it’s not a good happening.

And also, when we are dealing with R=0 radius we are just update the neuron which has the minimum distance and then most of the neurons doesn’t have a same formation with side neurons and this is all about that fact we are just update the winner neuron!

Furthermore, when we are dealing with variable radius it handles the issues as well as we expect and then we have neurons with better performance.

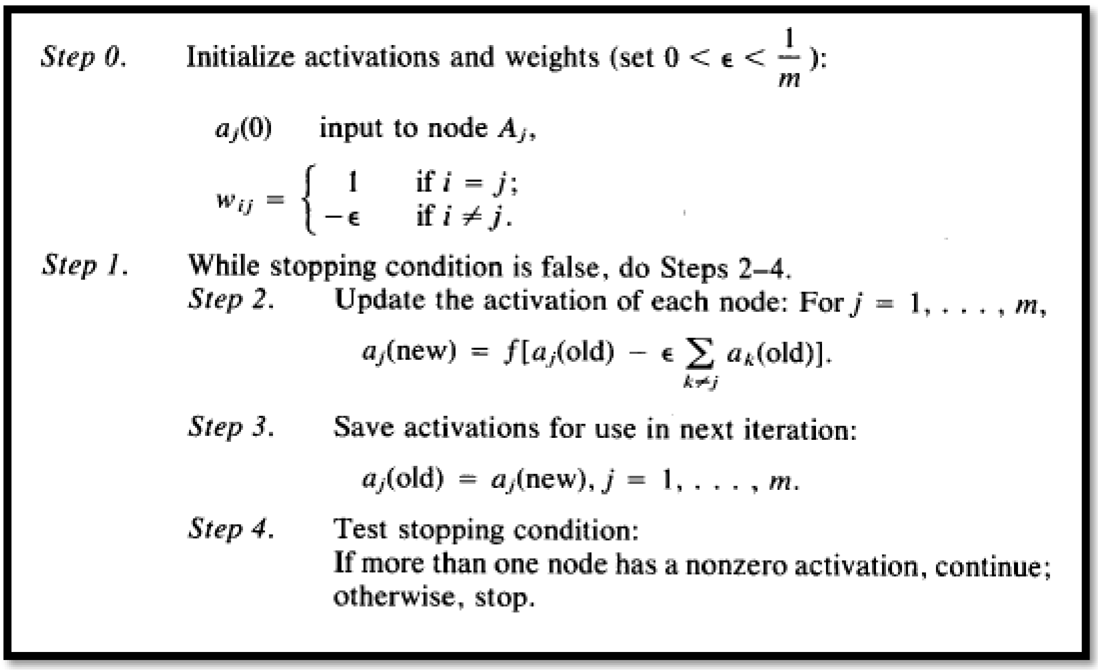
3 QUESTION #2: MAX-NET

In this part we intend to implement Max-Net in case of finding the maximum value.

Further, we are going to report the results of this section one by one.

First of all, we should have a brief explanation about the algorithm and then investigate the results.

Max-Net algorithm:



```
Step 0.  Initialize activations and weights (set $0 < \epsilon < \frac{1}{m}$):
             $a_j(0)$   input to node $A_j$,
             $w_{ij} = \begin{cases} 1 & \text{if } i = j; \\ -\epsilon & \text{if } i \neq j. \end{cases}$
Step 1.  While stopping condition is false, do Steps 2–4.
    Step 2.  Update the activation of each node: For $j = 1, \dots, m$,
                 $a_j(\text{new}) = f[a_j(\text{old}) - \epsilon \sum_{k \neq j} a_k(\text{old})]$.
    Step 3.  Save activations for use in next iteration:
                 $a_j(\text{old}) = a_j(\text{new}), j = 1, \dots, m$.
    Step 4.  Test stopping condition:
             If more than one node has a nonzero activation, continue;
             otherwise, stop.
```

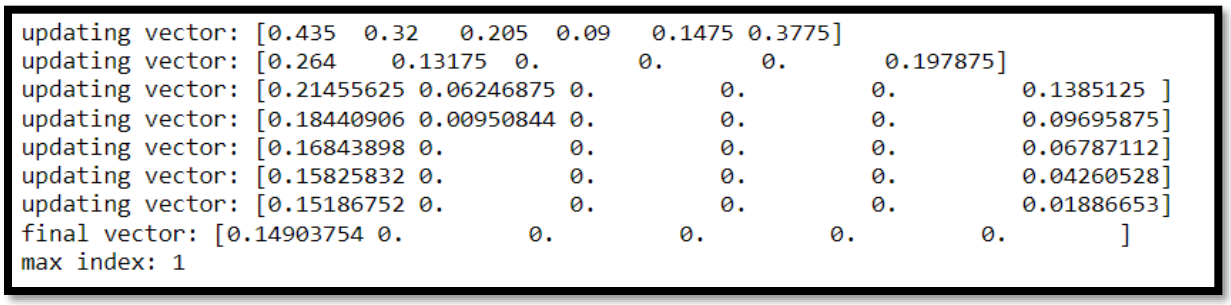
Figure9. Max-Net algorithm

I have implemented the algorithm using above notes and I have used **below parameters**:

$\in = 0.15$

$x = [1.2 , 1.1 , 1 , 0.9 , 0.95 , 1.15]$

Now according to the question description, we do the above steps and calculate the max-net output:



```
updating vector: [0.435   0.32    0.205   0.09    0.1475 0.3775]
updating vector: [0.264    0.13175  0.       0.       0.       0.197875]
updating vector: [0.21455625 0.06246875 0.         0.         0.         0.1385125 ]
updating vector: [0.18440906 0.00950844 0.         0.         0.         0.09695875]
updating vector: [0.16843898 0.         0.         0.         0.         0.06787112]
updating vector: [0.15825832 0.         0.         0.         0.         0.04260528]
updating vector: [0.15186752 0.         0.         0.         0.         0.01886653]
final vector: [0.14903754 0.         0.         0.         0.         0.        ]
max index: 1
```

Figure10. Max-Net algorithm

3.1 FINDING MAXIMUM USING MAX-NET

In this part we are going to investigate situation which is about determining the maximum when all values tend to be greater than $\beta \in R$.

The most important condition here is:

We must never have a vector with all negative value!

Because if all vector values be negative according to the activation function and type of algorithm it tends to be more and more negative and then the algorithm doesn't work.

Solution: we should subtract all value from β value and then all of negative values tend to be positive and also we have the right formation of values in case of order!

Max-net Procedure:

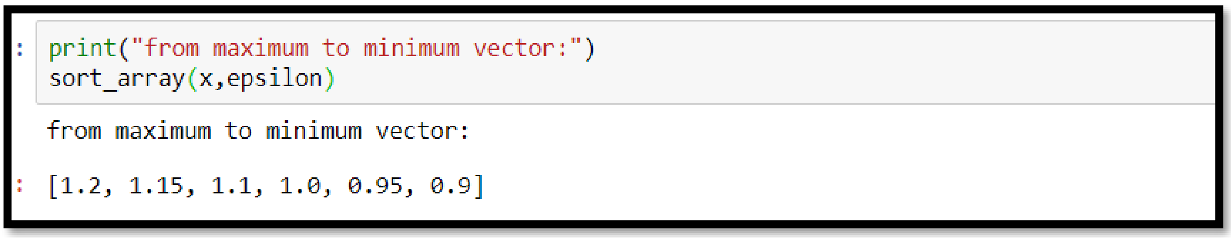
If we have care about above notes the max-net gives us the maximum value index so we can find which one has the max value among the others.

3.2 SORTING FROM MAXIMUM TO MINIMUM USING MAX-NET

in this section I have added a trick which let us use max-net network to sort a special array from maximum to minimum.

We have a while loop during our code which in a every iteration gives us the maximum value of vector and then by deleting the value from mentioned vector we repeat the steps for the new vector until the length of tuple tends to be 1 and in every single iteration we store the values in a vector which is called the sorted-vector.

Further I have reported the results:


```
: print("from maximum to minimum vector:")
  sort_array(x,epsilon)

  from maximum to minimum vector:

: [1.2, 1.15, 1.1, 1.0, 0.95, 0.9]
```

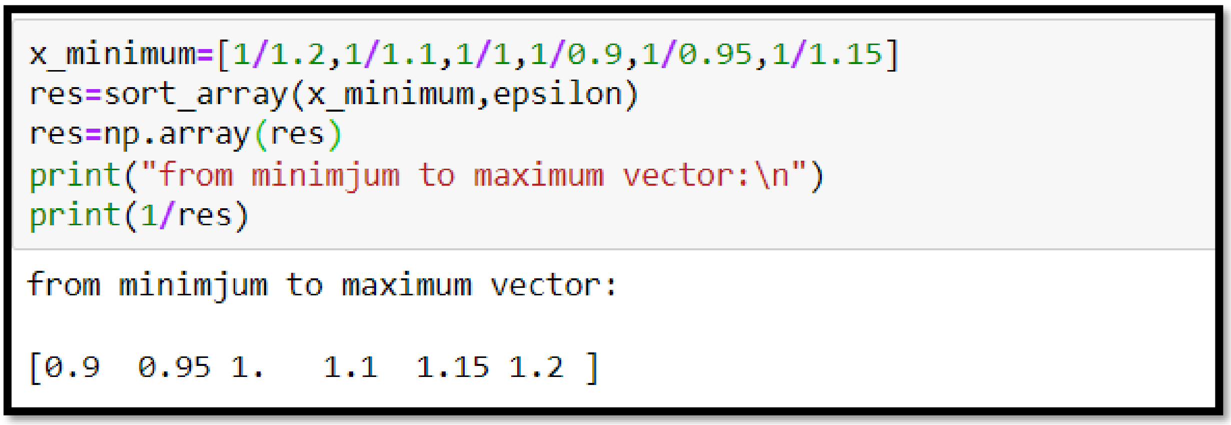
3.3 SORTING FROM MINIMUM TO MAXIMUM USING MAX-NET

in this section I have added a trick which let us use max-net network to sort a special array from minimum to maximum.

In this part we should use the inverse of input matrix as of our new input which change the problem space to the task of finding the minimum using max-net network and rest of steps are the same.

We have a while loop during our code which in a every iteration gives us the maximum value of vector and then by deleting the value from mentioned vector we repeat the steps for the new vector until the length of tuple tends to be 1 and in every single iteration we store the values in a vector which is called the sorted-vector.

Further I have reported the results:



```
x_minimum=[1/1.2,1/1.1,1/1,1/0.9,1/0.95,1/1.15]
res=sort_array(x_minimum,epsilon)
res=np.array(res)
print("from minimjum to maximum vector:\n")
print(1/res)
```

```
from minimjum to maximum vector:

[0.9  0.95 1.   1.1  1.15 1.2 ]
```

For the conditions we must make sure that the input vector should not be all negative values.

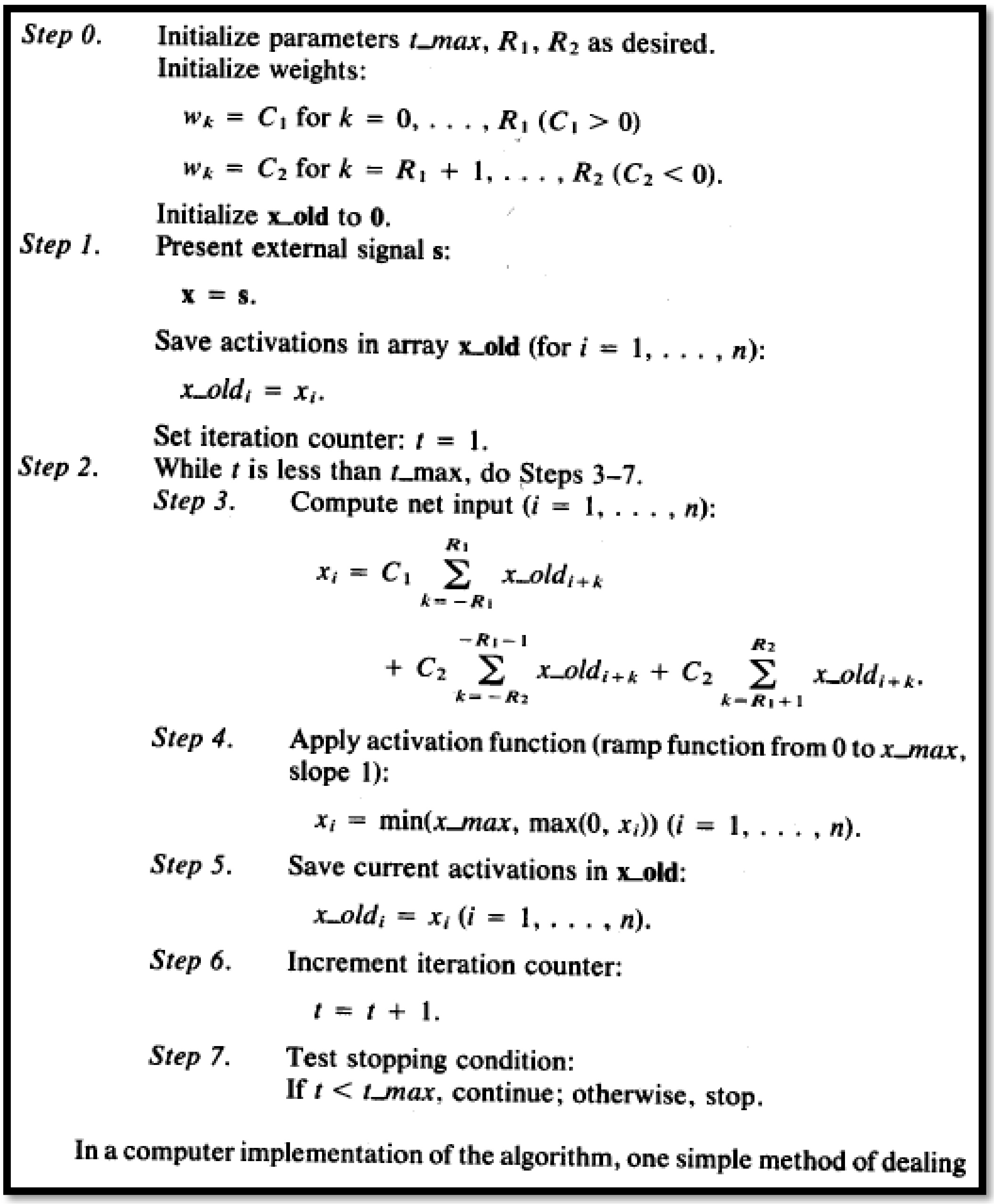
4 QUESTION #3: MEXICAN-HAT

In this part we intend to implement Mexican-Hat in case of finding the maximum value.

Further, we are going to report the results of this section one by one.

First of all, we should have a brief explanation about the algorithm and then investigate the results for two of the mentioned state:

Mexican-Hat algorithm:



```
Step 0.  Initialize parameters $t\_max$, $R_1$, $R_2$ as desired.
         Initialize weights:
             $w_k = C_1$ for $k = 0, \ldots, R_1$ $(C_1 > 0)$
             $w_k = C_2$ for $k = R_1 + 1, \ldots, R_2$ $(C_2 < 0)$.
         Initialize x_old to 0.
Step 1.  Present external signal s:
             $\mathbf{x} = \mathbf{s}$.
         Save activations in array x_old (for $i = 1, \ldots, n$):
             $x\_old_i = x_i$.
         Set iteration counter: $t = 1$.
Step 2.  While $t$ is less than $t$_max, do Steps 3–7.
    Step 3.  Compute net input ($i = 1, \ldots, n$):
                 $x_i = C_1 \sum_{k=-R_1}^{R_1} x\_old_{i+k} + C_2 \sum_{k=-R_2}^{-R_1-1} x\_old_{i+k} + C_2 \sum_{k=R_1+1}^{R_2} x\_old_{i+k}$.
    Step 4.  Apply activation function (ramp function from 0 to $x\_max$,
             slope 1):
                 $x_i = \min(x\_max, \max(0, x_i))$ $(i = 1, \ldots, n)$.
    Step 5.  Save current activations in x_old:
                 $x\_old_i = x_i$ $(i = 1, \ldots, n)$.
    Step 6.  Increment iteration counter:
                 $t = t + 1$.
    Step 7.  Test stopping condition:
             If $t < t\_max$, continue; otherwise, stop.
```

In a computer implementation of the algorithm, one simple method of dealing

Figure11. Mexican-Hat algorithm

I have implemented the algorithm using above notes and I have used **below parameters**:

$$t_{max} = 10$$

$$F(x)= \begin{cases} 0 & x < 0 \\ x & 0 \le x < 2 \\ 2 & 2 \le x \end{cases}$$

4.1 State One:

First of all, we tend to run the model using below parameters:

$$R1 = 0 \ \& \ R2 = \infty \, , C1 = 0.7 \, , C2 = -0.1$$

According to the question description I have provided the results:

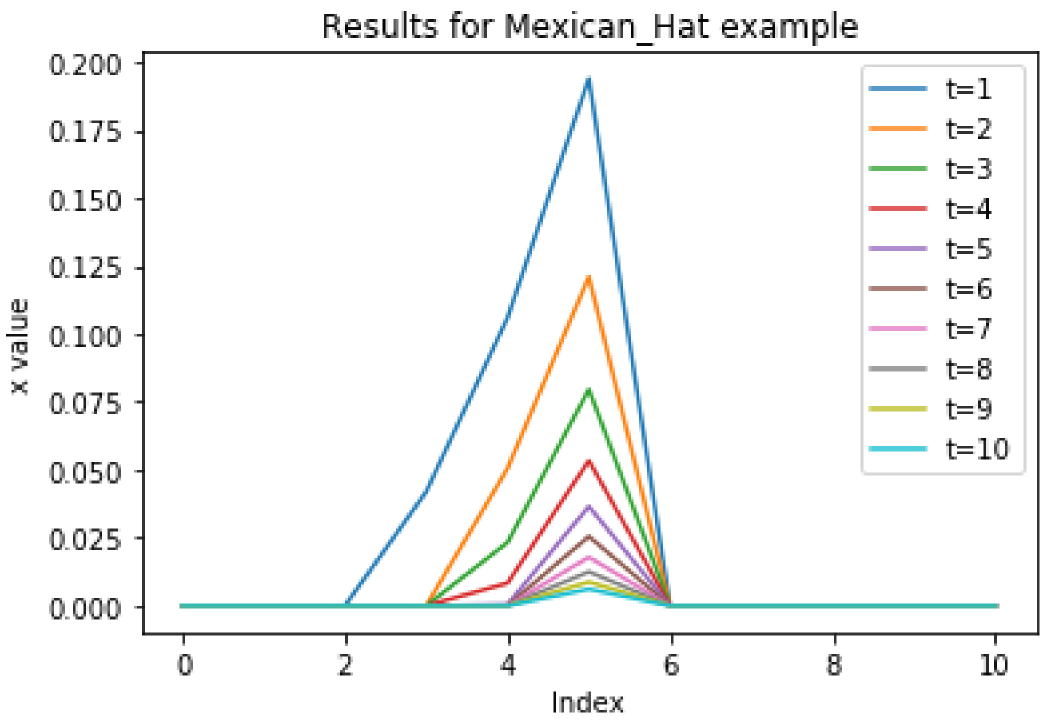


Figure12. Mexican-Hat algorithm

4.2 State Two:

First of all, we tend to run the model using below parameters:

$\boldsymbol{R1 = 1}$ **&** $\boldsymbol{R2 = 3\,,\, C1 = 0.6\,,\, C2 = -0.4}$

According to the question description I have provided the results:

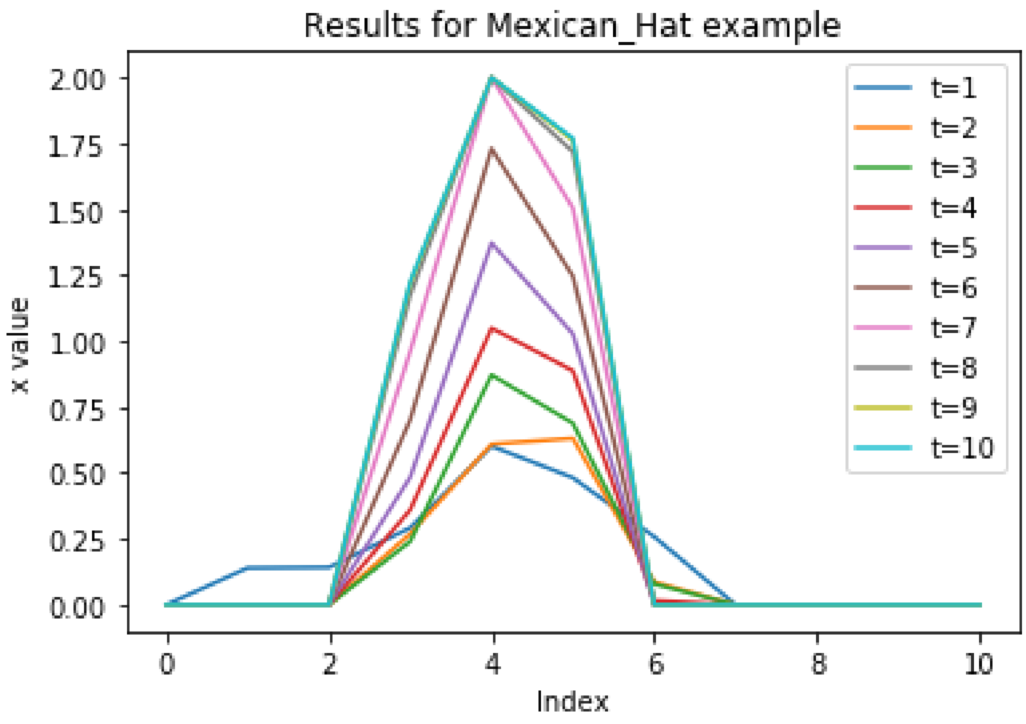


Figure13. Mexican-Hat algorithm

Comparing the results:

As we can see in state one we have converged to the maximum value of vector which is 0.77 , in other word when we are dealing with R1=0 and R2=inf it seems that we are dealing with max-net network which is same with Mexican-hat in this special case.

But when we are dealing with R1=1 and R2=3 it seems that we just investigate vector data around this neighborhood and then the output result consists three value and doesn't pass the maximum value in comparison with previous state and as you can see the results obviously shows mentioned trend.

5 QUESTION #4: HAMMING-NET

In this part we intend to implement Hamming-Net in case of Pattern Association.

Further, we are going to report the results of this section one by one:

5.1 STORING THE EXEMPLARS AND CALCULATING THE DISTANCES:

In this section we are dealing with a Hamming-distance which is used in task of pattern similarity finding.

First let's make the exemplars vectors.

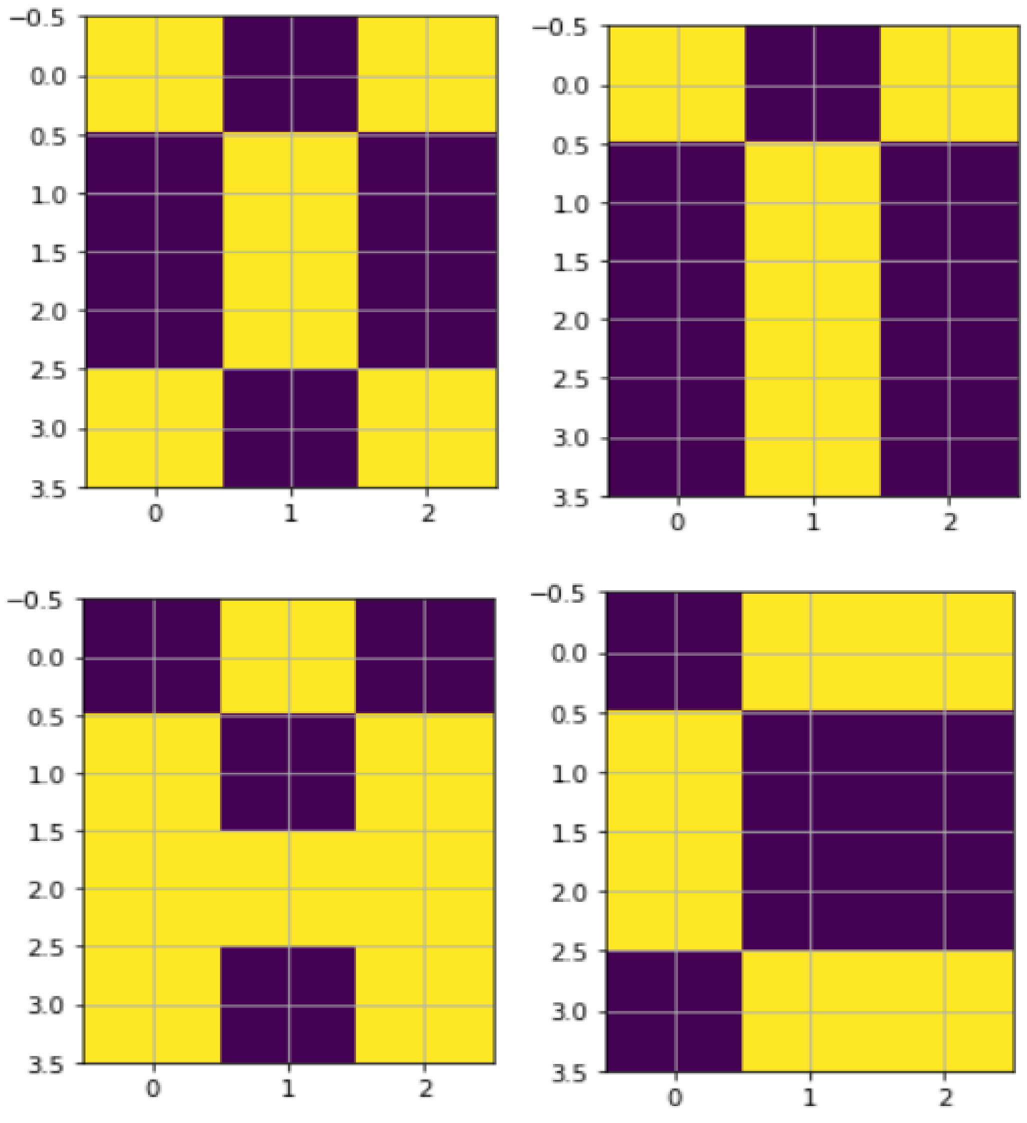
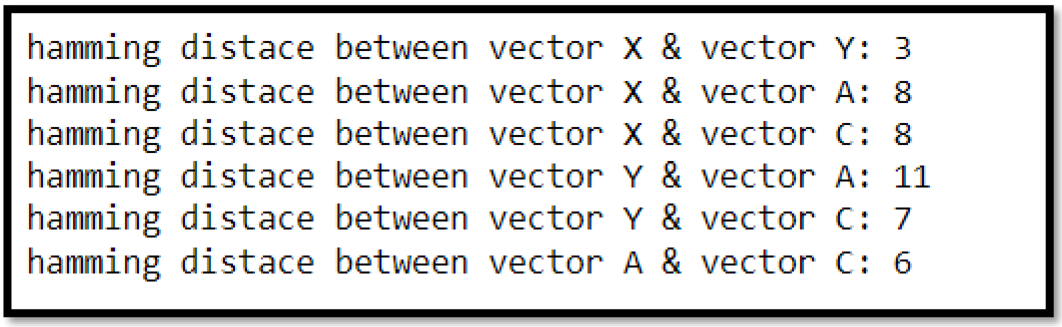


Figure14. Exemplar representations

and then we are going to calculate the hamming-distance which results have been abbreviated below:

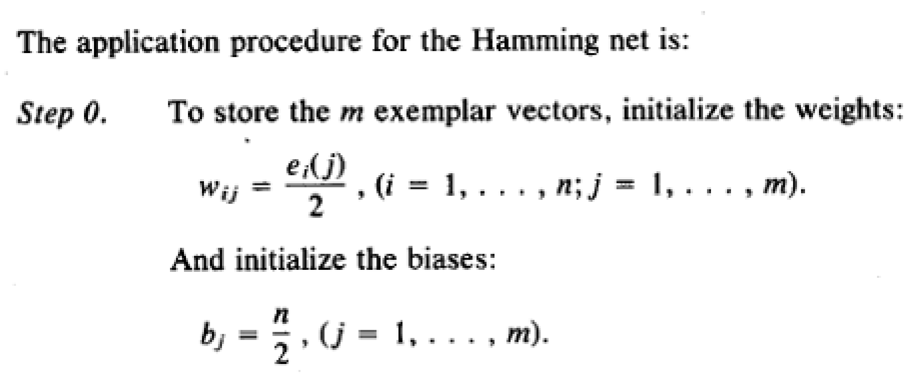


```
hamming distace between vector X & vector Y: 3
hamming distace between vector X & vector A: 8
hamming distace between vector X & vector C: 8
hamming distace between vector Y & vector A: 11
hamming distace between vector Y & vector C: 7
hamming distace between vector A & vector C: 6
```

Figure15. Calculating hamming-distance between different exemplars

5.2 DETERMINING THE WEIGHTS AND BIAS MATRIX

In this section we are going to implement the hamming-net using below notes about weight and bias matrix:



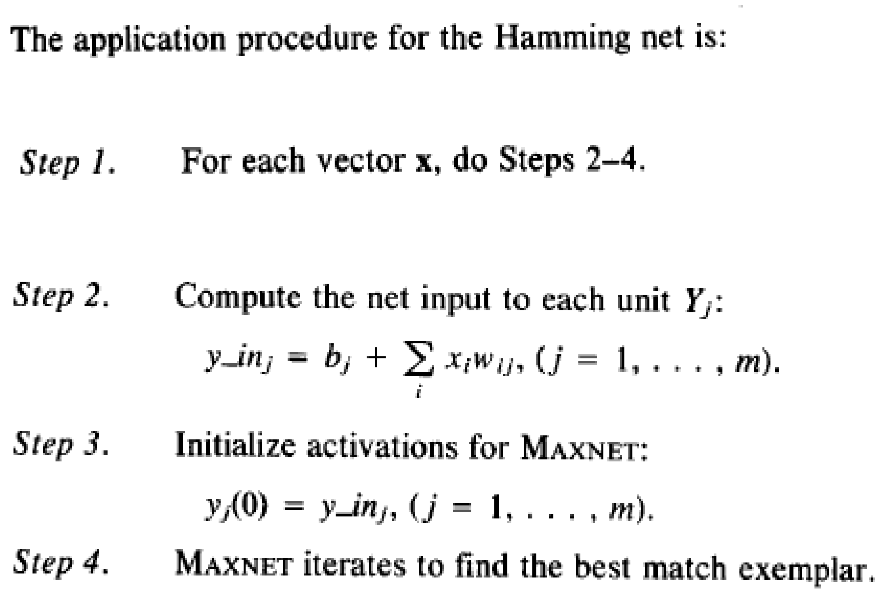
The application procedure for the Hamming net is:

Step 0. To store the m exemplar vectors, initialize the weights:

$$w_{ij} = \frac{e_i(j)}{2}, (i = 1, \ldots, n; j = 1, \ldots, m).$$

And initialize the biases:

$$b_j = \frac{n}{2}, (j = 1, \ldots, m).$$



The application procedure for the Hamming net is:

Step 1. For each vector **x**, do Steps 2–4.

Step 2. Compute the net input to each unit Y_j:

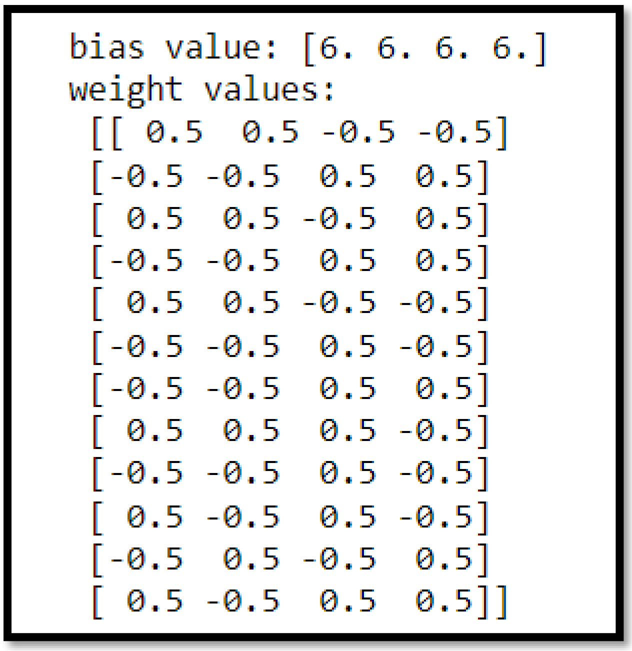
$$y\_in_j = b_j + \sum_i x_i w_{ij}, (j = 1, \ldots, m).$$

Step 3. Initialize activations for MAXNET:

$$y_j(0) = y\_in_j, (j = 1, \ldots, m).$$

Step 4. MAXNET iterates to find the best match exemplar.

Further I have calculated the weight and bias matrix:



```
bias value: [6. 6. 6. 6.]
weight values:
 [[ 0.5  0.5 -0.5 -0.5]
 [-0.5 -0.5  0.5  0.5]
 [ 0.5  0.5 -0.5  0.5]
 [-0.5 -0.5  0.5  0.5]
 [ 0.5  0.5 -0.5 -0.5]
 [-0.5 -0.5  0.5 -0.5]
 [-0.5 -0.5  0.5  0.5]
 [ 0.5  0.5  0.5 -0.5]
 [-0.5 -0.5  0.5 -0.5]
 [ 0.5 -0.5  0.5 -0.5]
 [-0.5  0.5 -0.5  0.5]
 [ 0.5 -0.5  0.5  0.5]]
```

Figure16. Representation of weight and bias matrix in hamming-net

5.3 DETERMINING THE RESULTS AND REPORTING THE MOST SIMILARITY ONE

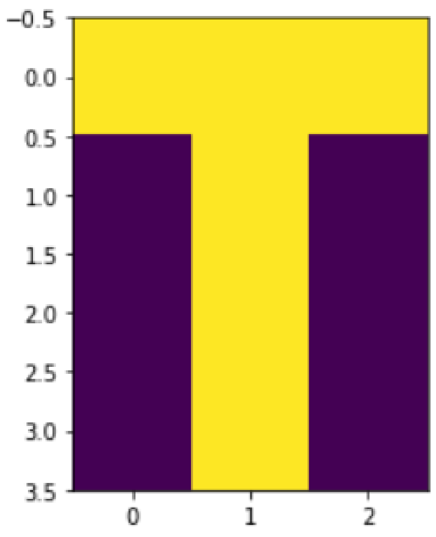
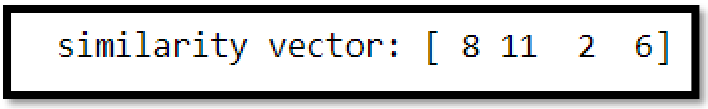


Figure17. Representation of input vector

Using hamming-distance we can determine the similarity vector which is equal to the below vector:

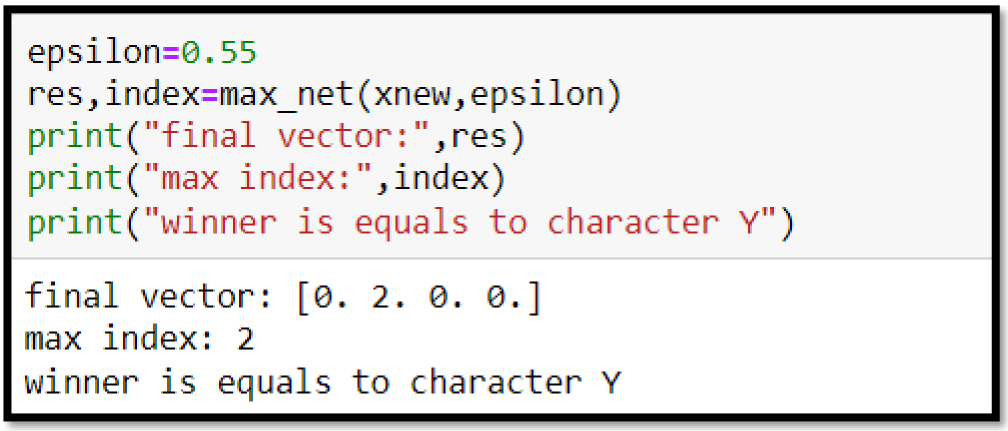


```
similarity vector: [ 8 11  2  6]
```

Figure18. Similarity-Vector

And then we should pass the mentioned vector to the max-net to find out which index is corresponded to the maximum similarity and then determine the character which has the most similarity with input.

The result of max-net is according to the below vector:



```
epsilon=0.55
res,index=max_net(xnew,epsilon)
print("final vector:",res)
print("max index:",index)
print("winner is equals to character Y")
```

```
final vector: [0. 2. 0. 0.]
max index: 2
winner is equals to character Y
```

Figure19. Using max-net to find most similarity-one

as we can expect the most similarity is character “Y” which is same with the character that our network predicts!

6 REFERENCES

[1] https://www.geeksforgeeks.org/hopfield-neural-network/

[2] https://stackoverflow.com/questions/21465988/python-equivalent-to-hold-on-in-matlab

[3] https://www.geeksforgeeks.org/bidirectional-associative-memory-bam-implementation-from-scratch/

[4] https://www.codegrepper.com/code-examples/python/change+image+size+python

[5] https://www.hackerearth.com/practice/notes/extracting-pixel-values-of-an-image-in-python/

[6] https://numpy.org/doc/stable/reference/generated/numpy.concatenate.html

[7] https://www.geeksforgeeks.org/how-to-compute-the-pseudoinverse-of-a-matrix-in-pytorch/

[8] https://www.geeksforgeeks.org/moore-penrose-pseudoinverse-mathematics/

[9] https://www.geeksforgeeks.org/association-rule/

[10]https://www.researchgate.net/publication/236093245_Learning_the_pseudoinverse_solution_to_network_weights

[11] https://stackoverflow.com/questions/21254472/multiple-plots-in-one-figure-in-python

[12] https://stackoverflow.com/questions/9638826/plot-a-black-and-white-binary-map-in-matplotlib

[13] https://matplotlib.org/3.5.0/api/_as_gen/matplotlib.pyplot.legend.html