University of Tehran
Neural Network and Deep Learning
HW4

Milad Mohamadi
Student Number: 810100462

June 2, 2022

Questions List Table

1 Question 1 3

2 Part 1 3

3 Part 2 4

4 Part2-Optional-1 7

5 Part2-Optional-2 8

6 Part 3 9

7 Part 3-Optional 12

8 Part 4 12

9 Question 2 13

10 Part 1 13

11 Part 2 14

12 Part 3 16

13 Question 3 18

14 Part 1 19

15 Part 2 20

16 Question 4 21

17 Part 1 21

18 Part 2 21

19 Part 3 23

1 Question 1

[1]:
```
import numpy as np
from sklearn.preprocessing import MinMaxScaler
import sys

import os
import torch
import matplotlib.pyplot as plt
import torchvision.datasets as datasets
from torch.utils.data.dataloader import DataLoader
```

2 Part 1

[ ]:
```
data = datasets.FashionMNIST('data/', train=True, download=True).data.numpy().
 ↪astype(float)
X_train = data[:1000]
X_test = data[1000:4000]
```

Flattening the 28*28 images

[3]:
```
X_train = X_train.reshape(1000,28*28)
X_test = X_test.reshape(3000,28*28)
```

Normalizing the input images

[4]:
```
sc = MinMaxScaler(feature_range = (0, 1))
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

The weight matrix will be as follows: Number of rows = Number of neurons = 15$*$15 and Number of columns = Number of inputs = 28*28)

In step 3, when calculating the D(j), the minimum amount among the calculated D(j)s depends on how neurons are arranged and which input vector member is subtract and reaches power 2. So at this step arrangement of neurons is important, and as the third step affects the fourth step, it is also important.

3 Part 2

Hyperparameters

[10]:
```
# Number of neurons
M = 225

# Dimension of the input patterns
N = X_train.shape[1]

# Total number of input patterns
P = X_train.shape[0]

learning_rate = 0.3

R = 1

MAX_EPOCHS = 100

MAX_WEIGHT_DIFF = 0.0001

DECAY_FACTOR = 0.0001

RADIUS_REDUCTION_STEP = 200

np.set_printoptions(threshold=sys.maxsize)
```

[12]:
```
# Step 1: Initialization of each node's weights with a random number between 0 and 1
initial_weight = np.random.rand(28*28*(15*15)).reshape(28*28, 15*15)

weight = np.copy(initial_weight)

last_weight = np.copy(weight)
weights_history =[]
weights_history.append(last_weight)
for epoch in range(MAX_EPOCHS):
    print("\r\nEpoch:", epoch)
    print("Learning rate:", np.around(learning_rate, 6))
    print("Neighborhood radius:", R)

    # Step 2: Choosing input patterns ordering
    # use normal ordering
    pattern_ordering = np.arange(P)

    progress = 0

    # For each input pattern do the steps 3-5
    for p in pattern_ordering:

        progress = progress + 1

        # Step 3: Calculating the Best Matching Unit (BMU)

        # initialize distance vector
        distance_vector = np.zeros(M)

        # calculate distance of each weight from each input pattern
        for j in range(M):
            for i in range(N):
                distance_vector[j] = distance_vector[j] + (weight[i,j] -␣
↪X_train[p,i])**2

        # Step 4: find index j such that distance_vector[j] is a minimum
        min_distance_index = np.argmin(distance_vector)

        # Step 5: Update weights for all units j within a specified neighborhood of␣
↪min_distance_index and for all i
        # calculate neighborhood borders
        begin_j = min_distance_index - R
        if (begin_j < 0):
            begin_j = 0

        end_j = min_distance_index + R
        if (end_j > M - 1):
            end_j = M - 1

        for j in range(begin_j, end_j + 1):
            for i in range(N):
                weight[i,j] = weight[i,j] + learning_rate * (X_train[p,i] -␣
↪weight[i,j])

    # Step 6: Update learning rate
    learning_rate = DECAY_FACTOR * learning_rate

    # Step 8: Test stopping condition
    weight_diff = np.amax(np.abs(weight - last_weight))

    if (weight_diff < MAX_WEIGHT_DIFF):
        print("Weight change:", weight_diff, "<", MAX_WEIGHT_DIFF)
        print("Stopping condition is satisfied!")
        break
    else:
        print("Weight change:", weight_diff, ">", MAX_WEIGHT_DIFF)

    last_weight = np.copy(weight)
    weights_history.append(last_weight)
```

```
Epoch: 0
Learning rate: 0.3
Neighborhood radius: 1
Weight change: 0.9999943949405043 > 0.0001

Epoch: 1
Learning rate: 3e-05
Neighborhood radius: 1
Weight change: 0.0016355817704544195 > 0.0001

Epoch: 2
Learning rate: 0.0
Neighborhood radius: 1
Weight change: 1.6316137574357015e-07 < 0.0001
Stopping condition is satisfied!
```

Weights

```
[48]: weight[0]
```

```
[48]: array([2.69034232e-05, 8.76573213e-08, 2.04149310e-08, 1.75364915e-12,
             3.48122580e-15, 7.59254194e-21, 8.89470281e-27, 1.99780236e-33,
             4.20140646e-37, 3.22327826e-39, 1.91143969e-35, 9.27454509e-35,
             7.35528536e-33, 1.72296322e-29, 3.34215150e-25, 6.83278533e-22,
             4.86521139e-20, 1.69647982e-17, 4.35043613e-17, 4.87453120e-15,
             4.19717197e-13, 1.92499188e-09, 8.97701295e-07, 3.24421126e-05,
             1.70746621e-02, 4.28283924e-01, 1.26779051e-01, 5.18632369e-01,
             3.43066137e-01, 8.27668716e-01, 2.68880839e-01, 8.83260047e-02,
             2.25624361e-01, 9.54098810e-01, 8.24666678e-01, 2.05288537e-01,
             5.21520769e-02, 6.08609934e-01, 7.29437684e-01, 2.75878634e-01,
             8.67155349e-01, 4.11347365e-01, 8.83373145e-02, 2.21612589e-01,
             2.99555438e-01, 3.14222163e-01, 8.58260922e-01, 3.68934098e-01,
             6.99548156e-01, 1.99276481e-01, 7.21131795e-01, 8.85763640e-01,
             9.42911130e-01, 7.62297469e-01, 3.30483585e-01, 9.20935407e-02,
             6.80191187e-01, 1.23506843e-01, 1.25998818e-01, 3.81360234e-01,
             5.45858216e-01, 3.53781689e-01, 6.83843726e-01, 8.83810352e-01,
             3.17590320e-01, 7.39935767e-01, 6.94440726e-01, 7.82618034e-01,
             1.19893646e-01, 1.57157625e-02, 9.87668616e-01, 7.37013214e-01,
             7.19280626e-01, 4.25556807e-01, 9.09288938e-01, 2.15821294e-01,
             1.83024157e-01, 7.23184010e-01, 2.37338074e-01, 3.55054935e-01,
             8.27166328e-01, 2.60354008e-01, 7.25137112e-01, 2.26680757e-01,
             8.87842631e-01, 4.60912619e-01, 9.44552946e-01, 4.31597292e-01,
             5.44624378e-01, 2.69721367e-01, 6.52284065e-01, 2.55882423e-01,
             8.84453825e-01, 1.30099691e-01, 8.18406555e-01, 9.10895607e-01,
             9.40633834e-01, 4.34794706e-01, 2.52863029e-01, 1.62020612e-01,
             7.21760845e-01, 7.05185410e-01, 3.83876228e-01, 5.64729360e-01,
             9.65346708e-01, 1.98554405e-01, 9.54877007e-01, 2.42401563e-01,
             4.71384530e-01, 1.71815532e-01, 5.81456206e-01, 4.99121727e-01,
             5.82992100e-01, 6.48975855e-01, 6.07986045e-02, 2.53370448e-01,
             4.30110945e-02, 9.15878801e-01, 3.24289264e-01, 5.80705107e-01,
             5.47578565e-01, 7.98041232e-01, 5.65689472e-01, 7.83508771e-01,
             5.51525844e-01, 6.75675698e-02, 9.98608518e-01, 2.33272852e-01,
             8.27894323e-01, 5.55555702e-01, 6.85559563e-01, 7.64720192e-01,
             9.81308019e-01, 2.64956188e-01, 3.31682254e-01, 2.88646113e-01,
             1.26881687e-02, 6.67697994e-01, 4.76488131e-01, 7.69290071e-02,
             8.61988760e-01, 1.35052239e-01, 7.06196704e-01, 7.76548910e-01,
             4.95809075e-01, 3.55478454e-01, 4.54522148e-01, 4.32949432e-01,
             5.07411390e-01, 3.60676766e-01, 9.49191461e-01, 4.40946796e-01,
             2.76970476e-01, 4.74956480e-01, 9.28245631e-01, 4.65513367e-01,
             4.79328096e-01, 9.31066337e-01, 2.54840609e-01, 5.01142668e-01,
             9.51872506e-02, 9.04734797e-02, 3.90078918e-01, 5.29325165e-01,
             4.41330688e-01, 7.72504563e-01, 9.36783927e-01, 5.83391476e-01,
             2.39510988e-01, 4.38859510e-01, 6.86977080e-02, 9.02152122e-01,
             1.17957436e-01, 1.08232700e-01, 7.39715676e-01, 5.43390590e-01,
             9.26936778e-02, 1.03407396e-01, 1.81365670e-01, 7.53151161e-01,
             7.45456260e-01, 4.98816887e-01, 4.30559656e-01, 9.46172391e-02,
             3.03556550e-01, 8.97040902e-01, 9.02713369e-01, 2.53343691e-01,
             3.00518891e-01, 5.69068687e-01, 2.55329823e-01, 6.59652875e-01,
             9.56577575e-01, 2.83852552e-02, 5.73057859e-01, 6.22862806e-01,
             1.89472333e-01, 9.06057052e-01, 5.29002892e-01, 8.72563786e-01,
             3.76101940e-02, 7.39041303e-01, 4.48475639e-01, 6.76559260e-01,
             5.82915595e-01, 3.98474484e-01, 6.06927381e-01, 7.62519350e-01,
             7.33221060e-01, 3.25731518e-01, 3.88400194e-01, 7.89512890e-01,
             9.47090263e-01, 4.54099563e-01, 6.59900044e-01, 7.49314651e-01,
             7.05899090e-01, 2.56085269e-01, 7.64063699e-01, 3.76692931e-01,
             8.84110852e-01, 1.21748446e-01, 6.14712505e-01, 3.34999683e-01,
             8.28613499e-01])
```

[13]:
```python
patterns_with_clusters = np.zeros((P, 2), dtype=int)

for p in range(1000):

    distance_vector = np.zeros(M)

    # calculate distance of each weight from each input pattern
    for j in range(M):
        for i in range(N):
            distance_vector[j] = distance_vector[j] + (weight[i,j] - X_test[p,i])**2

    # find index j such that distance_vector[j] is a minimum
    min_distance_index = np.argmin(distance_vector)

    # store pattern index
    patterns_with_clusters[p,0] = p
    # store cluster number associated with pattern
    patterns_with_clusters[p,1] = min_distance_index
```

[38]:
```python
# make an array of clusters size
clusters_size = [0 for j in range(M)]
clusters_members = [[] for j in range(M)]
for p in range(1000):
    # increment cluster size by 1
    clusters_size[patterns_with_clusters[p,1].astype(int)] =␣
 ↪clusters_size[patterns_with_clusters[p,1].astype(int)] + 1
```

4 Part2-Optional-1

Given the diagram of 25 neurons have more than one class.

[42]:
```python
import seaborn as sns
clusters = np.asarray(clusters_size)
# Set position of bar on X axis
br1 = np.arange(len(clusters))

# Make the plot
plt.figure(figsize=(15,10))
plt.bar(br1, clusters, color ='r', width = 1,
        edgecolor ='grey', label ='IT')
```

[42]:
```
<BarContainer object of 225 artists>
```

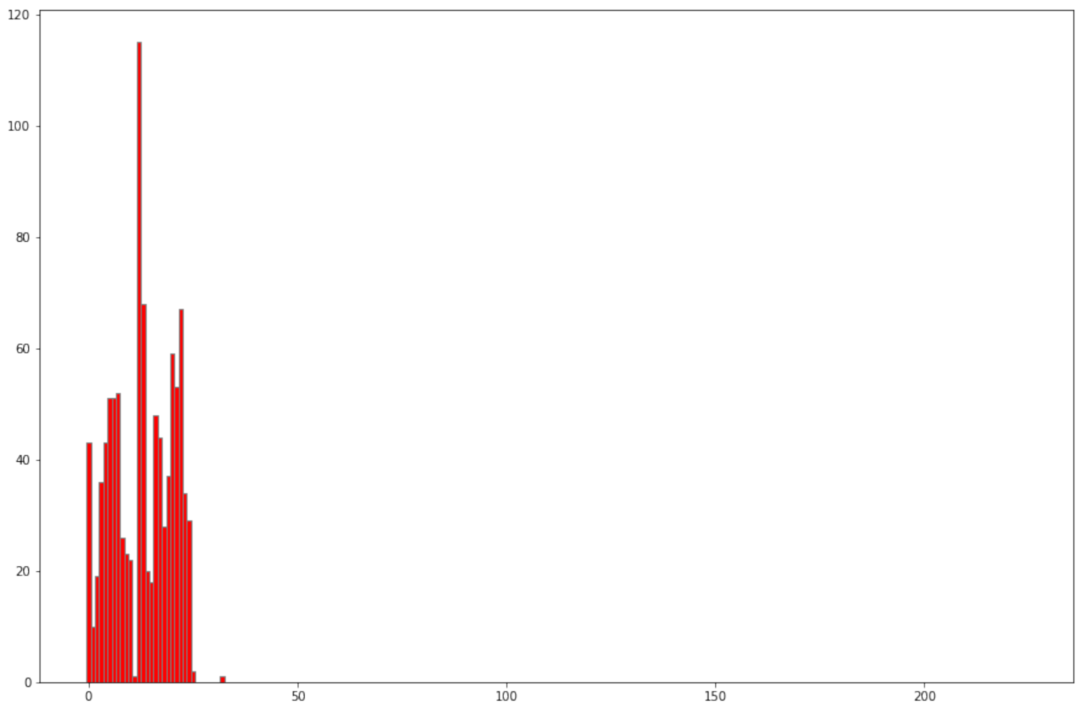

Figure 1: Diagram of classes mapping to neurons(with R = 1)

5 Part2-Optional-2

The weight matrix will be as follows: Number of rows = Number of neurons = 15$*$15 and Number of columns = Number of inputs = 28*28) and Network performance will be weak given that R = 0.

6 Part 3

Hyperparameters

[5]:
```python
# Number of neurons
M = 225

# Dimension of the input patterns
N = X_train.shape[1]

# Total number of input patterns
P = X_train.shape[0]

learning_rate = 0.3

R = 3

MAX_EPOCHS = 100

MAX_WEIGHT_DIFF = 0.0001

DECAY_FACTOR = 0.0001

RADIUS_REDUCTION_STEP = 200

np.set_printoptions(threshold=sys.maxsize)
```

[6]:
```python
initial_weight = np.random.rand(28*28*(15*15)).reshape(28*28, 15*15)

# Step 1: Initialization of each node's weights with a random number between 0 and 1
weight = np.copy(initial_weight)
#print("Initial weights:")
#print(np.around(weight, 2))

last_weight = np.copy(weight)
weights_history =[]
weights_history.append(last_weight)
for epoch in range(MAX_EPOCHS):
    print("\r\nEpoch:", epoch)
    print("Learning rate:", np.around(learning_rate, 6))
    print("Neighborhood radius:", R)

    # Step 2: Choosing input patterns ordering
    # use normal ordering
    pattern_ordering = np.arange(P)
    # use random ordering
    #np.random.shuffle(pattern_ordering)
    #print("Random input patterns ordering:", pattern_ordering)

    progress = 0

    # For each input pattern do the steps 3-5
    for p in pattern_ordering:

        progress = progress + 1

        #print("\r\n\tCurrent pattern index is", p, "and", np.around(progress/
↪P*100, 2), "% of patterns has been processed in Epoch", epoch)
```

```
        # Step 3: Calculating the Best Matching Unit (BMU)

        # initialize distance vector
        distance_vector = np.zeros(M)

        # calculate distance of each weight from each input pattern
        for j in range(M):
            for i in range(N):
                distance_vector[j] = distance_vector[j] + (weight[i,j] -␣
↪X_train[p,i])**2

        #print("\tdistance_vector", np.around(distance_vector, 2))

        # Step 4: find index j such that distance_vector[j] is a minimum
        min_distance_index = np.argmin(distance_vector)

        #print("\tmin_distance_index", min_distance_index)

        # Step 5: Update weights for all units j within a specified neighborhood of␣
↪min_distance_index and for all i
        # calculate neighborhood borders
        begin_j = min_distance_index - R
        if (begin_j < 0):
            begin_j = 0

        end_j = min_distance_index + R
        if (end_j > M - 1):
            end_j = M - 1

        #print("neighberhood", begin_j, end_j)

        for j in range(begin_j, end_j + 1):
            for i in range(N):
                weight[i,j] = weight[i,j] + learning_rate * (X_train[p,i] -␣
↪weight[i,j])

    # Step 6: Update learning rate
    learning_rate = DECAY_FACTOR * learning_rate

    # Step 7: Reduce radius of topological neighborhood at specified times
    if R > 0 :
            R = R - 1

    # Step 8: Test stopping condition
    weight_diff = np.amax(np.abs(weight - last_weight))

    if (weight_diff < MAX_WEIGHT_DIFF):
        print("Weight change:", weight_diff, "<", MAX_WEIGHT_DIFF)
        print("Stopping condition is satisfied!")
        break
    else:
        print("Weight change:", weight_diff, ">", MAX_WEIGHT_DIFF)

    last_weight = np.copy(weight)
    weights_history.append(last_weight)
```

```
Epoch: 0
Learning rate: 0.3
Neighborhood radius: 3
Weight change: 0.9998354951980238 > 0.0001

Epoch: 1
Learning rate: 3e-05
Neighborhood radius: 2
Weight change: 0.0025957416211894735 > 0.0001

Epoch: 2
Learning rate: 0.0
Neighborhood radius: 1
Weight change: 1.9475723433970415e-07 < 0.0001
Stopping condition is satisfied!
```

[7]:
```
patterns_with_clusters = np.zeros((P, 2), dtype=int)

for p in range(1000):

    distance_vector = np.zeros(M)

    # calculate distance of each weight from each input pattern
    for j in range(M):
        for i in range(N):
            distance_vector[j] = distance_vector[j] + (weight[i,j] - X_test[p,i])**2

    # find index j such that distance_vector[j] is a minimum
    min_distance_index = np.argmin(distance_vector)

    # store pattern index
    patterns_with_clusters[p,0] = p
    # store cluster number associated with pattern
    patterns_with_clusters[p,1] = min_distance_index
```

[8]:
```
# make an array of clusters size
clusters_size = [0 for j in range(M)]
clusters_members = [[] for j in range(M)]
for p in range(1000):
    # increment cluster size by 1
    clusters_size[patterns_with_clusters[p,1].astype(int)] =␣
 ↪clusters_size[patterns_with_clusters[p,1].astype(int)] + 1
```

7 Part 3-Optional

[9]:
```python
import seaborn as sns
clusters = np.asarray(clusters_size)
# Set position of bar on X axis
br1 = np.arange(len(clusters))

# Make the plot
plt.figure(figsize=(15,10))
plt.bar(br1, clusters, color ='r', width = 1,
        edgecolor ='grey', label ='IT')
```

[9]:
```
<BarContainer object of 225 artists>
```

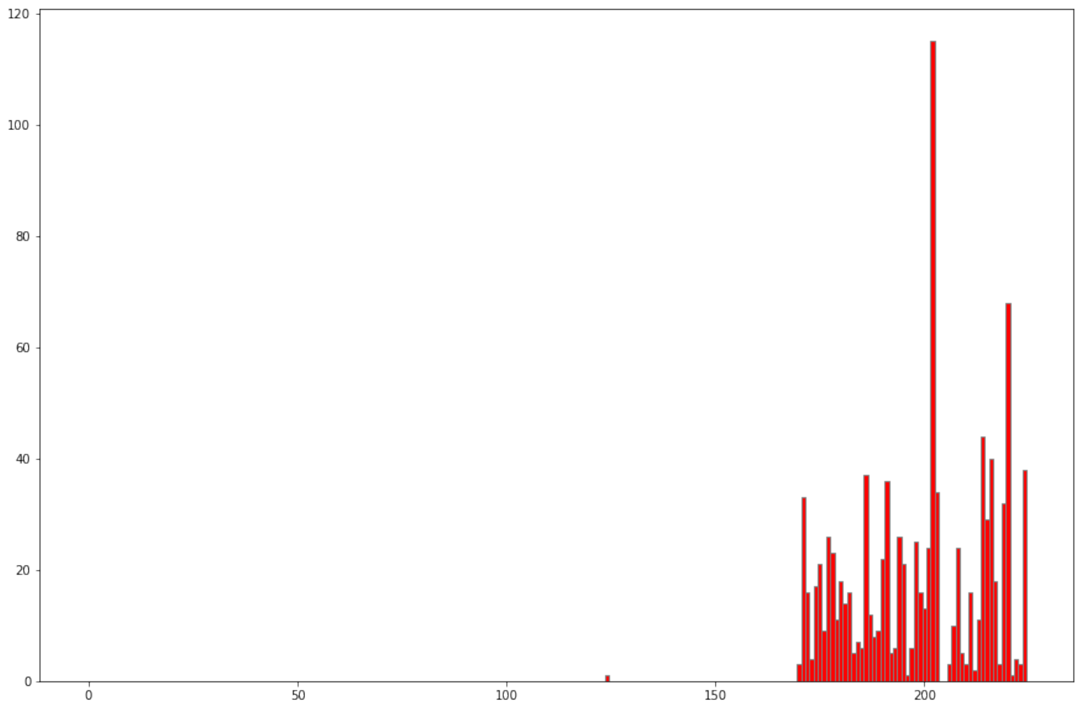

Figure 2: Diagram of classes mapping to neurons(with variable R)

8 Part 4

Yes the recovery has been achieved. As we can see, it is more suitable than part b.In this section, more classes are mapped to more neurons. In fact, the distribution of classes to neurons is normal. And it's not like a neuron has a lot of classes, but many neurons do not map them.

When we use variable neighborhood radius, our calculations for a neuron such as X are compared to all neurons, not just its neighbors, and therefore we get a desirable result.

9 Question 2

10 Part 1

If all the numbers are larger than that is a real number, and want to find the biggest of them by the Maxnet network, it is enough to obtain the difference between these numbers with the real number and give the numbers to the network. In other words, if A is one of the numbers in this collection, we give the Maxnet network B = A - and do this on the rest of the numbers as well.

[1]:
```
import numpy as np

input = np.array([1.2, 1.1, 1, 0.9, 0.95, 1.15])
eps = 0.15
np.set_printoptions(formatter={'float': '{: 0.3f}'.format})
```

[2]:
```
def activation_function(x):
  for i in range(0, len(x)):
        if x[i] < 0.0:
            x[i] = 0.0
        else:
            x[i] = x[i]
  return x
```

[3]:
```
def MaxNet_Matrix(m=6, eps=0.15):
    matrix = np.full((m, m), -1*(eps))
    np.fill_diagonal(matrix, 1)
    return matrix
```

[4]:
```
def MaxNet(data, eps):
    a_old = data.copy()
    maxnet_mat = MaxNet_Matrix(m=6, eps=eps)
    num_non_zero = 0
    i = 0
    while num_non_zero!=1:
        temp_data = np.inner(maxnet_mat, a_old)
        a_new = activation_function(temp_data)
        print("Epoch {0}:===> {1}".format(i+1, a_new))
        print('----------------------------------')
        num_non_zero = np.count_nonzero(a_new)
        a_old = a_new
        if num_non_zero == 1:
            break
        i+=1

    out_tpl = np.nonzero(a_new)
    print("Index of the Max Element :=> ", out_tpl[0].item())
    print("The Value of the Max element is :=> ", data[out_tpl[0].item()])

MaxNet(data=input, eps=eps)
```

```
Epoch 1:===> [ 0.435  0.320  0.205  0.090  0.147  0.377]
----------------------------------
Epoch 2:===> [ 0.264  0.132  0.000  0.000  0.000  0.198]
----------------------------------
Epoch 3:===> [ 0.215  0.062  0.000  0.000  0.000  0.139]
----------------------------------
Epoch 4:===> [ 0.184  0.010  0.000  0.000  0.000  0.097]
----------------------------------
```

```
Epoch 5:===> [ 0.168  0.000  0.000  0.000  0.000  0.068]
----------------------------------
Epoch 6:===> [ 0.158  0.000  0.000  0.000  0.000  0.043]
----------------------------------
Epoch 7:===> [ 0.152  0.000  0.000  0.000  0.000  0.019]
----------------------------------
Epoch 8:===> [ 0.149  0.000  0.000  0.000  0.000  0.000]
----------------------------------
Index of the Max Element :=>  0
The Value of the Max element is :=>  1.2
```

11 Part 2

To be able to arrange a set of numbers from large to small by the Maxnet network, we can use this network as a circulation. This way, each time it runs through the remaining numbers, the largest number is found and the numbers can be arranged from large to small. The largest number was found in part one. Now at each step we remove the largest number from the input numbers and find the largest number between the remaining numbers.

[5]:
```python
def MaxNetDescending(data, eps):
    sorted_array = []
    m = len(data)
    a_old = data.copy()
    data1 = data.copy()
    maxnet_mat = MaxNet_Matrix(m=m, eps=eps)
    num_non_zero = 0
    i = 0
    for i in range(0, m):
        a_old = data1.copy()
        num_non_zero = 0
        i = 0
        while num_non_zero!=1:
            temp_data = np.inner(maxnet_mat, a_old)
            a_new = activation_function(temp_data)
            print("Epoch {0}:===> {1}".format(i+1, a_new))
            print('----------------------------------')
            num_non_zero = np.count_nonzero(a_new)
            a_old = a_new
            if num_non_zero == 1 or num_non_zero==0:
                break
            i+=1

        out_tpl = np.nonzero(a_new)
        print("Index of the Max Element :=> ", out_tpl[0].item())
        print("The Value of the Max element is :=> ", data1[out_tpl[0].item()])
        sorted_array.append(data1[out_tpl[0].item()])
        data1[out_tpl[0].item()] = 0
        print('----------------------------')
    print("Entered Vector :=> ", data)
    print("Final Sorted Descending Array :=> ", sorted_array)


MaxNetDescending(data=input, eps=eps)
```

```
Epoch 1:===> [ 0.435  0.320  0.205  0.090  0.147  0.377]
---------------------------------
Epoch 2:===> [ 0.264  0.132  0.000  0.000  0.000  0.198]
---------------------------------
```

```
Epoch 3:===> [ 0.215  0.062  0.000  0.000  0.000  0.139]
---------------------------------
Epoch 4:===> [ 0.184  0.010  0.000  0.000  0.000  0.097]
---------------------------------
Epoch 5:===> [ 0.168  0.000  0.000  0.000  0.000  0.068]
---------------------------------
Epoch 6:===> [ 0.158  0.000  0.000  0.000  0.000  0.043]
---------------------------------
Epoch 7:===> [ 0.152  0.000  0.000  0.000  0.000  0.019]
---------------------------------
Epoch 8:===> [ 0.149  0.000  0.000  0.000  0.000  0.000]
---------------------------------
Index of the Max Element :=>  0
The Value of the Max element is :=>  1.2
----------------------------
Epoch 1:===> [ 0.000  0.500  0.385  0.270  0.327  0.557]
---------------------------------
Epoch 2:===> [ 0.000  0.269  0.137  0.005  0.071  0.335]
---------------------------------
Epoch 3:===> [ 0.000  0.187  0.035  0.000  0.000  0.263]
---------------------------------
Epoch 4:===> [ 0.000  0.142  0.000  0.000  0.000  0.230]
---------------------------------
Epoch 5:===> [ 0.000  0.108  0.000  0.000  0.000  0.208]
---------------------------------
Epoch 6:===> [ 0.000  0.077  0.000  0.000  0.000  0.192]
---------------------------------
Epoch 7:===> [ 0.000  0.048  0.000  0.000  0.000  0.181]
---------------------------------
Epoch 8:===> [ 0.000  0.021  0.000  0.000  0.000  0.174]
---------------------------------
Epoch 9:===> [ 0.000  0.000  0.000  0.000  0.000  0.170]
---------------------------------
Index of the Max Element :=>  5
The Value of the Max element is :=>  1.15
----------------------------
Epoch 1:===> [ 0.000  0.673  0.557  0.443  0.500  0.000]
---------------------------------
Epoch 2:===> [ 0.000  0.448  0.315  0.183  0.249  0.000]
---------------------------------
Epoch 3:===> [ 0.000  0.335  0.183  0.031  0.107  0.000]
---------------------------------
Epoch 4:===> [ 0.000  0.287  0.112  0.000  0.025  0.000]
---------------------------------
Epoch 5:===> [ 0.000  0.267  0.065  0.000  0.000  0.000]
---------------------------------
Epoch 6:===> [ 0.000  0.257  0.025  0.000  0.000  0.000]
---------------------------------
Epoch 7:===> [ 0.000  0.253  0.000  0.000  0.000  0.000]
---------------------------------
Index of the Max Element :=>  1
The Value of the Max element is :=>  1.1
----------------------------
Epoch 1:===> [ 0.000  0.000  0.723  0.608  0.665  0.000]
---------------------------------
Epoch 2:===> [ 0.000  0.000  0.532  0.399  0.465  0.000]
---------------------------------
Epoch 3:===> [ 0.000  0.000  0.402  0.250  0.326  0.000]
```

```
---------------------------------
Epoch 4:===> [ 0.000  0.000  0.316  0.141  0.228  0.000]
---------------------------------
Epoch 5:===> [ 0.000  0.000  0.260  0.059  0.160  0.000]
---------------------------------
Epoch 6:===> [ 0.000  0.000  0.227  0.000  0.112  0.000]
---------------------------------
Epoch 7:===> [ 0.000  0.000  0.211  0.000  0.078  0.000]
---------------------------------
Epoch 8:===> [ 0.000  0.000  0.199  0.000  0.046  0.000]
---------------------------------
Epoch 9:===> [ 0.000  0.000  0.192  0.000  0.016  0.000]
---------------------------------
Epoch 10:===> [ 0.000  0.000  0.190  0.000  0.000  0.000]
---------------------------------
Index of the Max Element :=>  2
The Value of the Max element is :=>  1.0
----------------------------
Epoch 1:===> [ 0.000  0.000  0.000  0.758  0.815  0.000]
---------------------------------
Epoch 2:===> [ 0.000  0.000  0.000  0.635  0.701  0.000]
---------------------------------
Epoch 3:===> [ 0.000  0.000  0.000  0.530  0.606  0.000]
---------------------------------
Epoch 4:===> [ 0.000  0.000  0.000  0.439  0.527  0.000]
---------------------------------
Epoch 5:===> [ 0.000  0.000  0.000  0.360  0.461  0.000]
---------------------------------
Epoch 6:===> [ 0.000  0.000  0.000  0.291  0.407  0.000]
---------------------------------
Epoch 7:===> [ 0.000  0.000  0.000  0.230  0.363  0.000]
---------------------------------
Epoch 8:===> [ 0.000  0.000  0.000  0.176  0.329  0.000]
---------------------------------
Epoch 9:===> [ 0.000  0.000  0.000  0.126  0.302  0.000]
---------------------------------
Epoch 10:===> [ 0.000  0.000  0.000  0.081  0.283  0.000]
---------------------------------
Epoch 11:===> [ 0.000  0.000  0.000  0.038  0.271  0.000]
---------------------------------
Epoch 12:===> [ 0.000  0.000  0.000  0.000  0.265  0.000]
---------------------------------
Index of the Max Element :=>  4
The Value of the Max element is :=>  0.95
----------------------------
Epoch 1:===> [ 0.000  0.000  0.000  0.900  0.000  0.000]
---------------------------------
Index of the Max Element :=>  3
The Value of the Max element is :=>  0.9
----------------------------
Entered Vector :=>  [ 1.200  1.100  1.000  0.900  0.950  1.150]
Final Sorted Descending Array :=>  [1.2, 1.15, 1.1, 1.0, 0.95, 0.9]
```

12 Part 3

In order to arrange the numbers by the Maxnet network from small to large, we first negate the numbers. Then we change the negative numbers by answering the first question. In other words, by

this method we convert negative numbers into positive numbers. Now, by the Maxnet network, we arrange the set of numbers from large to small with the method of second question. The numbers are arranged from large to small, the small arranged to large raw numbers. To convert numbers into their original form, reduce the positive value of each and then negate each. Now we see that raw numbers are arranged from small to large

[7]:
```python
def MaxNetAscending(data, eps):
    a_old = data.copy()
    data1 = data.copy()
    maxnet_mat = MaxNet_Matrix(m=6, eps=eps)
    num_non_zero = 0
    i = 0
    all_zeros = np.array([], dtype=int)
    while num_non_zero!=1:
        temp_data = np.inner(maxnet_mat, a_old)
        a_new = activation_function(temp_data)
        print("Epoch {0}:===> {1}".format(i+1, a_new))
        print("-------------------------------")
        num_non_zero = np.count_nonzero(a_new)
        zeros = np.argwhere(a_new == 0).reshape(-1)
        zeros_diff = np.setdiff1d(zeros, all_zeros)
        all_zeros = np.append(all_zeros, zeros_diff)
        a_old = a_new
        if num_non_zero == 1:
            break
        i+=1

    out_tpl = np.nonzero(a_new)
    all_zeros = np.append(all_zeros, out_tpl[0].item())
    print("The Entered Vector :=> ", data)
    print("The Ascending Sorted Vector :=> ", data[all_zeros])


MaxNetAscending(data=input, eps=0.09)
```

```
Epoch 1:===> [ 0.741  0.632  0.523  0.414  0.468  0.686]
-------------------------------
Epoch 2:===> [ 0.496  0.377  0.258  0.139  0.199  0.436]
-------------------------------
Epoch 3:===> [ 0.369  0.239  0.110  0.000  0.045  0.304]
-------------------------------
Epoch 4:===> [ 0.306  0.165  0.024  0.000  0.000  0.235]
-------------------------------
Epoch 5:===> [ 0.268  0.114  0.000  0.000  0.000  0.191]
-------------------------------
Epoch 6:===> [ 0.240  0.073  0.000  0.000  0.000  0.157]
-------------------------------
Epoch 7:===> [ 0.220  0.037  0.000  0.000  0.000  0.128]
-------------------------------
Epoch 8:===> [ 0.205  0.006  0.000  0.000  0.000  0.105]
-------------------------------
Epoch 9:===> [ 0.195  0.000  0.000  0.000  0.000  0.086]
-------------------------------
Epoch 10:===> [ 0.187  0.000  0.000  0.000  0.000  0.069]
-------------------------------
Epoch 11:===> [ 0.181  0.000  0.000  0.000  0.000  0.052]
-------------------------------
Epoch 12:===> [ 0.176  0.000  0.000  0.000  0.000  0.036]
```

```
-------------------------------
Epoch 13:===> [ 0.173  0.000  0.000  0.000  0.000  0.020]
-------------------------------
Epoch 14:===> [ 0.171  0.000  0.000  0.000  0.000  0.004]
-------------------------------
Epoch 15:===> [ 0.171  0.000  0.000  0.000  0.000  0.000]
-------------------------------
The Entered Vector :=>  [ 1.200  1.100  1.000  0.900  0.950  1.150]
The Ascending Sorted Vector :=>  [ 0.900  0.950  1.000  1.100  1.150  1.200]
```

13 Question 3

[ ]:
```
import matplotlib.pyplot as plt
```

[ ]:
```
def activation_function(x):
  for i in range(len(x)):
    if x[i] < 0:
        x[i] = 0
    elif x[i] > 2:
        x[i] = 2
  return x
```

[ ]:
```
class MexicanHat:
  def __init__(self, R1, R2, c1, c2, x, t_max, activation_function):
    self.R1 = R1
    self.R2 = R2
    self.c1 = c1
    self.c2 = c2
    self.t_max = t_max
    self.activation_function = activation_function
    self.x_old = [0 for i in range(len(x))]
    self.x = x
    self.counter = 0

  def identify_border(self, index, max_length):
    r1_left = index - self.R1
    r1_right = index + self.R1 + 1
    r2_left = index - self.R2
    r2_right = index + self.R2 + 1
    if r1_left < 0:
      r1_left = 0
    if r2_left < 0:
      r2_left = 0
    if r1_right > max_length:
      r1_right = max_length
    if r2_right > max_length:
      r2_right = max_length
    return r1_left, r1_right, r2_left, r2_right



  def update(self):

    for t in range(self.t_max):
        self.x_old = self.x.copy()
        if t > 0:
            plt.plot(self.x_old)
            # plt.show()
        for i in range(len(self.x)):
            a, b, c = 0, 0, 0
            for k in range(-1 * self.R1, self.R1 + 1):
                if i + k > 0 and i + k < len(self.x) - 1:
                    a += self.x_old[i + k]

            for k in range(-1 * self.R2, -1 * self.R1):
                if i + k > 0 and i + k < len(self.x) - 1:
                    b += self.x_old[i + k]

            for k in range(self.R1 + 1, self.R2 + 1):
                if i + k > 0 and i + k < len(self.x) - 1:
                    b += self.x_old[i + k]

            self.x[i] = self.c1 * a + self.c2 * (b + c)

        self.x = activation_function(self.x)
```

14 Part 1

```
[ ]: x = [0.27, 0.35, 0.44, 0.58, 0.66, 0.77, 0.4, 0.32, 0.2, 0.15, 0.08]
     Mh = MexicanHat(0, len(x), 1, -0.1 ,x, 20, activation_function)
     Mh.update()
```

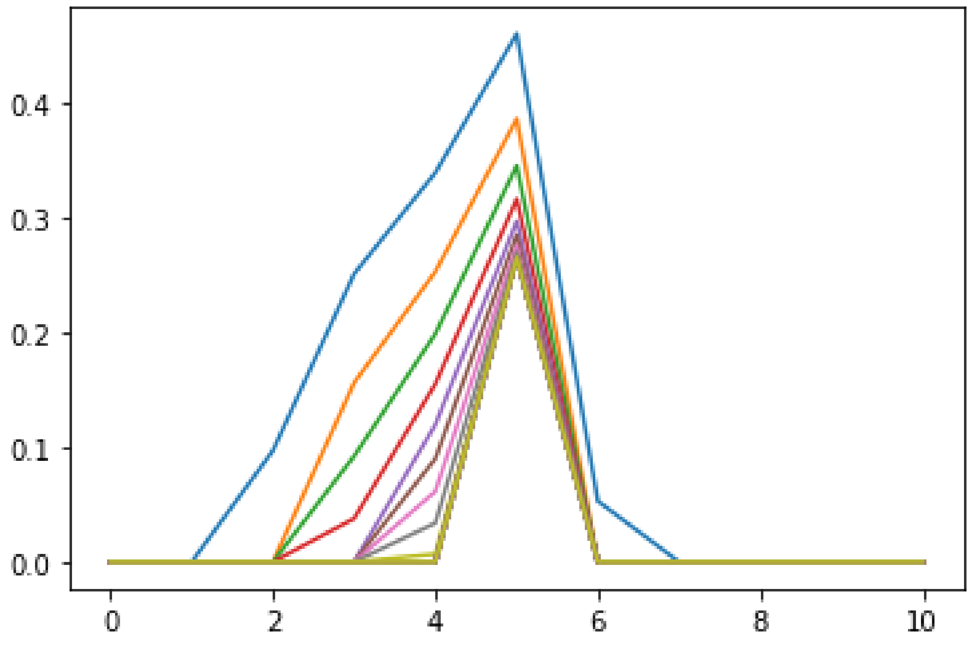

Figure 3: Index chart of array members and output signal value

15 Part 2

```
[ ]: x = [0.27, 0.35, 0.44, 0.58, 0.66, 0.77, 0.4, 0.32, 0.2, 0.15, 0.08]
     Mh = MexicanHat(1, 3, 0.6, -0.4 ,x, 20, activation_function)
     Mh.update()
```

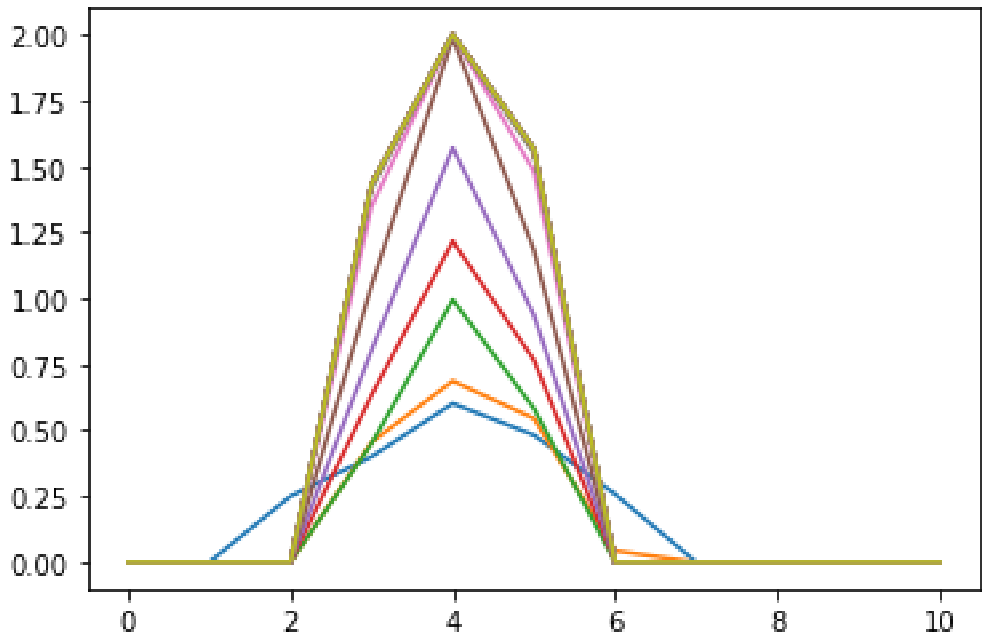

Figure 4: Index chart of array members and output signal value

16 Question 4

[1]:
```
import numpy as np
import sys
```

[2]:
```
X = np.array([1, -1, 1, -1, 1, -1, -1, 1, -1, 1, -1, 1])
Y = np.array([1, -1, 1, -1, 1, -1, -1, 1, -1, -1, 1, -1])
A = np.array([-1, 1, -1, 1, -1, 1, 1, 1, 1, 1, -1, 1])
C = np.array([-1, 1, 1, 1, -1, -1, 1, -1, -1, -1, 1, 1])
```

17 Part 1

[3]:
```
def compute_hamming_distance(a, b):
   return len(np.where(a != b)[0])
```

[4]:
```
print("Hamming distance of X and Y:", compute_hamming_distance(X, Y))
print("Hamming distance of X and A:", compute_hamming_distance(X, A))
print("Hamming distance of X and C:", compute_hamming_distance(X, C))
print("Hamming distance of Y and A:", compute_hamming_distance(Y, A))
print("Hamming distance of Y and C:", compute_hamming_distance(Y, C))
print("Hamming distance of A and C:", compute_hamming_distance(A, C))
```

```
Hamming distance of X and Y: 3
Hamming distance of X and A: 8
Hamming distance of X and C: 8
Hamming distance of Y and A: 11
Hamming distance of Y and C: 7
Hamming distance of A and C: 6
```

18 Part 2

[5]:
```
def activation_function(x):
  mask = 1*(x >= 0)
  x = x * mask
  return x
```

[6]:
```
def max_net(x):
    x_first = np.copy(x)

    T_MAX = 5
    R1 = 0
    R2 = 100
    C1 = 0.6
    C2 = -0.1
    X_MAX = 2

    # size of the input dataset
    N = x.shape[0]

    np.set_printoptions(threshold=sys.maxsize)

    history = np.zeros((T_MAX, N))

    x_old = np.copy(x)
    history[0] = x[:]

    for t in range(1, T_MAX):
        for i in range(N):

            # Calculate neighborhood borders
            B1_BEGIN = i - R1
            if (B1_BEGIN < 0):
                B1_BEGIN = 0

            B1_END = i + R1
            if (B1_END > N - 1):
                B1_END = N - 1

            B2_BEGIN = i - R2
            if (B2_BEGIN < 0):
                B2_BEGIN = 0

            B2_END = i + R2
            if (B2_END > N - 1):
                B2_END = N - 1

            # Create current weight matrix
            weight = np.zeros(N)
            for j in range(B2_BEGIN, B2_END + 1):
                weight[j] = C2
            for j in range(B1_BEGIN, B1_END + 1):
                weight[j] = C1

            x[i] = np.dot(weight, x_old)

        # Apply activation function
        x = np.minimum(X_MAX, np.maximum(0, x))

        history[t] = x[:]

        x_old = np.copy(x)

    return x_old.argmax()
```

[7]:
```
class HammingNet:
    def __init__(self,exemplar_vectors):
      self.exemplar_vectors = exemplar_vectors
      self.m = len(exemplar_vectors)
      self.n = len(exemplar_vectors[0])

      self.init_weights_bias()

    def init_weights_bias(self):
      self.bias = self.n/2
      self.weights = np.array(exemplar_vectors) / 2

    def find_closest(self,x):
      y_in = []
      for col in range(self.m):
        y_in.append(self.bias + np.dot(x,self.weights[col]))
      closest_vector = max_net( np.array(y_in) )
      # print()
      print('Input Vector: ',x ,'   Closest Vector: ',f'e{closest_vector+1}=',
            self.exemplar_vectors[closest_vector])
```

[13]:
```
exemplar_vectors = [[1, -1, 1, -1, 1, -1, -1, 1, -1, -1, 1, -1],
                    [1, -1, 1, -1, 1, -1, -1, 1, -1, 1, -1, 1],
                    [-1, 1, -1, 1, -1, 1, 1, 1, 1, 1, -1, 1],
                    [-1, 1, 1, 1, -1, -1, 1, -1, -1, -1, 1, 1]
]

input  = [1, 1, 1, -1, 1, -1, -1, 1, -1, -1, 1, -1]
```

[14]:
```
hamming_net = HammingNet(exemplar_vectors)
```

Weights

[15]:
```
hamming_net.weights
```

[15]:
```
array([[ 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]])
```

Bias

[16]:
```
hamming_net.bias
```

[16]:
```
6.0
```

19 Part 3

Output of network: Y

[17]:
```
hamming_net.find_closest(input)
```

```
Input Vector:  [1, 1, 1, -1, 1, -1, -1, 1, -1, -1, 1, -1]
Closest Vector: e1= [1, -1, 1, -1, 1, -1, -1, 1, -1, -1, 1, -1]
```

As we can see, the output of the network is the Y vector, and according to the input vector, the same was expected because the hamming distance of Y and input is 1 that is shown below.

[19]:
```
print("Hamming distance of input and Y:", compute_hamming_distance(Y, input))
print("Hamming distance of input and X:", compute_hamming_distance(X, input))
print("Hamming distance of input and C:", compute_hamming_distance(C, input))
print("Hamming distance of input and A:", compute_hamming_distance(A, input))
```

```
Hamming distance of input and Y: 1
Hamming distance of input and X: 4
Hamming distance of input and C: 6
Hamming distance of input and A: 10
```