University of Tehran

Electrical and Computer Engineering Department

Neural Networks and Deep Learning

Homework 4

Name	Danial Saeedi
Student No	810198571
Date	Tuesday - 2022 01 February

Table of contents

1. Question 4

1. Part A: Mexican-hat

The neurons are attached with the excitatory positive weights to the neurons closer to the reference neuron. Similarly, the neurons are also connected with negative inhibitory weights with neurons at a farther distance. In some architectures, some neurons are further away still which are not connected in terms of weights. “All of these connections are within a particular layer of a neural net.” The neural network acquires external input data apart from these connections. This data is multiplied with the corresponding weights depending on the proximity to obtain the value.

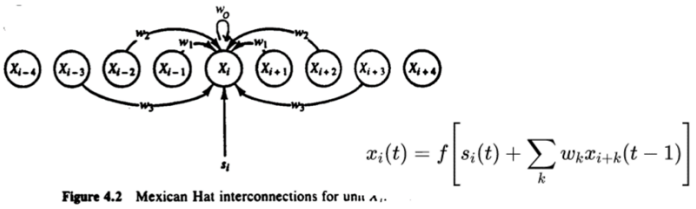

Figure 4.2 Mexican Hat interconnections for unit x_i.

Figure 1 Mexican Hat Architecture

For each node Xi there are:

1. some nearer nodes: X_j ,j∈{ i-R_1 ...i+ R_1} which are wired with positive weights (cooperative nodes).
2. Some farer nodes: X_j ,j∈{ i-R_2 ...i-R_1+1}U{ i+R_1+1,....i+R_2} wired with negative weights (competitive nodes)
3. other nodes which are out of cooperative or competitive nodes : X_j , j∈{....,i-R_2-1}U{i+R_2+1,...}, they do not wired to Xi (zero weight) .

Load dataset

In [ ]:
```
x = np.array([0.24,0.3,0.45,0.57,0.8,0.69,0.42,0.26,0.14])
x_first = np.copy(x)
# x = np.array([0,0.5,0.8,1,0.8,0.5,0])
print("Input data:", x)
```

```
Input data: [0.24 0.3  0.45 0.57 0.8  0.69 0.42 0.26 0.14]
```

Initialization

In [ ]:
```
T_MAX = 8
R1 = 1
R2 = 4
C1 = 0.6
C2 = -0.4
X_MAX = 2

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

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

Updating

In [ ]:
```
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)
```

Finding Max Element

In [ ]:
```
max_index = x_old.argmax()
max = x_first[max_index]
max
```

Out[129]:
```
0.8
```

Plotting

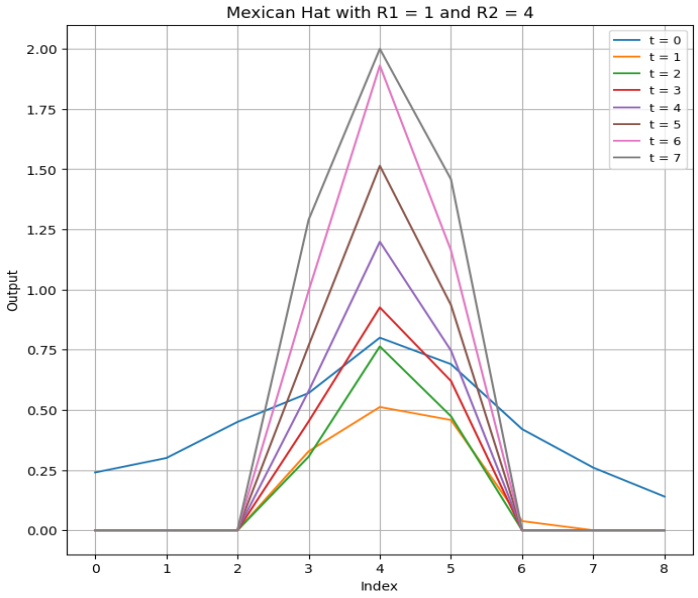

Figure 2 Output in each iteration

```
t: 0
x: [0.24 0.3  0.45 0.57 0.8  0.69 0.42 0.26 0.14]

t: 1
x: [0.   0.   0.   0.33 0.51 0.46 0.04 0.   0.  ]

t: 2
x: [0.   0.   0.   0.31 0.76 0.47 0.   0.   0.  ]

t: 3
x: [0.   0.   0.   0.45 0.93 0.62 0.   0.   0.  ]

t: 4
x: [0.   0.   0.   0.58 1.2  0.75 0.   0.   0.  ]

t: 5
x: [0.   0.   0.   0.77 1.51 0.94 0.   0.   0.  ]

t: 6
x: [0.   0.   0.   0.99 1.93 1.16 0.   0.   0.  ]

t: 7
x: [0.   0.   0.   1.29 2.   1.46 0.   0.   0.  ]
```

Figure 3 Output in each iteration

1. Part B: MaxNet

If we choose $R_1 = 0$, $R_2 = +inf$ and t_max = 5, then mexican hat networks acts like MaxNet.

Initialization

Here R2 = 100 and R2 = +inf are equal statement.

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

Figure 4 Parameters

Finding Max Element

```
In [ ]: max_index = x_old.argmax()
        max = x_first[max_index]
        max

Out[122]: 0.8
```

Plotting

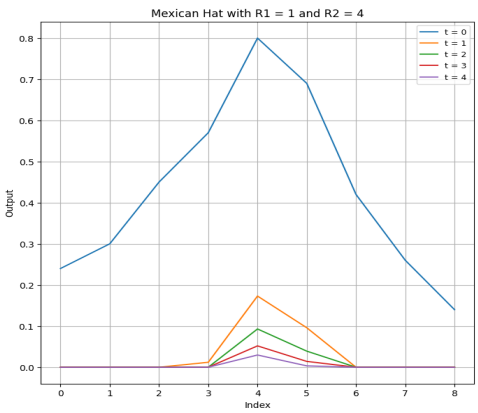

Figure 5 Vector value per iteration

```
t: 0
x: [0.24 0.3  0.45 0.57 0.8  0.69 0.42 0.26 0.14]

t: 1
x: [0.   0.   0.   0.01 0.17 0.1  0.   0.   0.  ]

t: 2
x: [0.   0.   0.   0.   0.09 0.04 0.   0.   0.  ]

t: 3
x: [0.   0.   0.   0.   0.05 0.01 0.   0.   0.  ]

t: 4
x: [0.   0.   0.   0.   0.03 0.   0.   0.   0.  ]
```

Figure 6 Output in each iteration

Part 2: HammingNet

Hamming neural network is composed of two layers of processing. The first layer is a neural network with direct links, in which the Hamming distance is calculated by comparing the reference image and supplied to the space image input. The second layer is a modified Hopfield neural network.

Input size is 6 and the output layer has 4 neurons because there are 4 exemplar vectors.

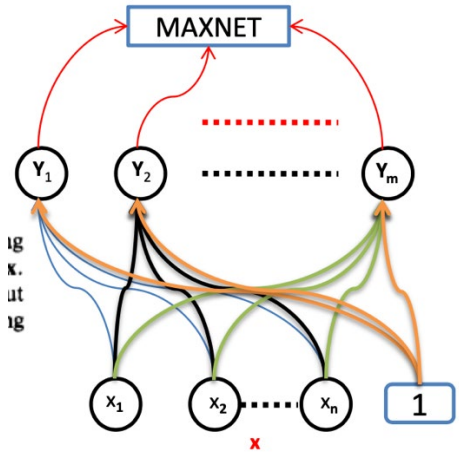

Figure 7 Hamming Net Architecture

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

Figure 8 Training Algorithm

Helper Functions

In [ ]:
```
def find_percentage_agreement(s1, s2):
    assert len(s1)==len(s2), "Lists must have the same shape"
    nb_agreements = 0  # initialize counter to 0
    for idx, value in enumerate(s1):
        if s2[idx] == value:
            nb_agreements += 1

    percentage_agreement = nb_agreements/len(s1)

    return percentage_agreement
```

Define Hamming Network

In [ ]:
```
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], '    Agreement Percentage: ',
            str(find_percentage_agreement(x,self.exemplar_vectors[closest_vector]))  + '%' )
```

In [11]:
```
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]
]
```

In [12]:
```
hamming_net = HammingNet(exemplar_vectors)
```

Testing the model

In [13]:
```
v1 = [-1,-1,1,-1,1,-1]
v2 = [1,1,1,1,-1,-1]
v3 = [-1,-1,-1,1,1,-1]
v4 = [1,-1,1,1,-1,1]
v5 = [1,1,1,-1,-1,-1]
v6 = [1,-1,-1,1,1,1]
v7 = [-1,1,-1,-1,-1,1]
```

In [14]:
```
hamming_net.find_closest(v1)
hamming_net.find_closest(v2)
hamming_net.find_closest(v3)
hamming_net.find_closest(v4)
hamming_net.find_closest(v5)
hamming_net.find_closest(v6)
hamming_net.find_closest(v7)
```

```
Input Vector:  [-1, -1, 1, -1, 1, -1]    Closest Vector:  e1= [1, -1, 1, -1, -1, -1]     Agreement Percentage:  0.666
66666666666666%
Input Vector:  [1, 1, 1, 1, -1, -1]    Closest Vector:  e4= [1, 1, -1, 1, 1, -1]     Agreement Percentage:  0.6666666
666666666%
Input Vector:  [-1, -1, -1, 1, 1, -1]    Closest Vector:  e2= [-1, 1, -1, 1, 1, -1]     Agreement Percentage:  0.8333
333333333334%
Input Vector:  [1, -1, 1, 1, -1, 1]    Closest Vector:  e3= [-1, -1, 1, 1, -1, 1]     Agreement Percentage:  0.833333
3333333334%
Input Vector:  [1, 1, 1, -1, -1, -1]    Closest Vector:  e1= [1, -1, 1, -1, -1, -1]     Agreement Percentage:  0.8333
333333333334%
Input Vector:  [1, -1, -1, 1, 1, 1]    Closest Vector:  e4= [1, 1, -1, 1, 1, -1]     Agreement Percentage:  0.6666666
666666666%
Input Vector:  [-1, 1, -1, -1, -1, 1]    Closest Vector:  e3= [-1, -1, 1, 1, -1, 1]     Agreement Percentage:  0.5%
```

2. Question 5: SOM

Part A

Loading Dataset

In [ ]:
```
# Load dataset with labels
raw_data = datasets.FashionMNIST('data/', train=True, download=True).data.numpy().astype(float)
test_data = raw_data[1000:1300]
```

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

In [ ]:
```
raw_data = raw_data[:1000]
```

In [ ]:
```
raw_data = raw_data.reshape(1000,28*28)
test_data = test_data.reshape(300,28*28)
```

Hyper-parameters

In [ ]:
```
# Number of neurons (1-dimensional)
M = 225

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

# Total number of input patterns
P = raw_data.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)
```

Pre-processing

```
In [ ]: # Normalize input vectors
sc = MinMaxScaler(feature_range = (0, 1))
data = sc.fit_transform(raw_data)
test_data = sc.transform(test_data)
```

Training

```
Epoch: 0
Learning rate: 0.3
Neighborhood radius: 1
Weight change: 0.9995507571793345 > 0.0001

Epoch: 1
Learning rate: 3e-05
Neighborhood radius: 1
Weight change: 0.0034967576816120616 > 0.0001

Epoch: 2
Learning rate: 0.0
Neighborhood radius: 0
Weight change: 1.2292408380076836e-07 < 0.0001
Stopping condition is satisfied!
```

Figure 9 Training Result

Part B

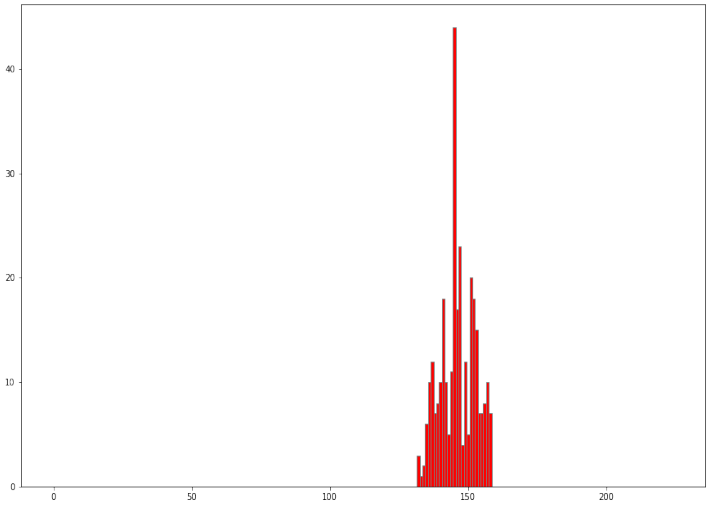

Figure 10 How many data is classified in each cluster

```
20 Largest Clusters
Cluster 145     =>       44 patterns.
Cluster 147     =>       23 patterns.
Cluster 151     =>       20 patterns.
Cluster 152     =>       18 patterns.
Cluster 141     =>       18 patterns.
Cluster 146     =>       17 patterns.
Cluster 153     =>       15 patterns.
Cluster 137     =>       12 patterns.
Cluster 149     =>       12 patterns.
Cluster 144     =>       11 patterns.
Cluster 157     =>       10 patterns.
Cluster 136     =>       10 patterns.
Cluster 140     =>       10 patterns.
Cluster 142     =>       10 patterns.
Cluster 156     =>       8 patterns.
Cluster 139     =>       8 patterns.
Cluster 155     =>       7 patterns.
Cluster 154     =>       7 patterns.
Cluster 138     =>       7 patterns.
Cluster 158     =>       7 patterns.

Total 274 patterns are in top 20 clusters.
```

Figure 11 - 20 Largest Clusters