



به نام خدا



دانشگاه تهران  
دانشکده مهندسی برق و کامپیوتر  
شبکه های عصبی و یادگیری عمیق

تمرین سری 4

مهرسا تاجیک	نام و نام خانوادگی
810198126	شماره دانشجویی
99/3/29	تاریخ ارسال گزارش

## فهرست گزارش سوالات

3.....	SOM-1 سوال
24.....	MaxNet – ۲ سوال
29.....	Mexican Hat – ۳ سوال
33.....	Hamming Net – ۴ سوال

## SOM-1 سوال

در ابتدا داده ها دانلود شده و در برنامه وارد می شود:

```
def top_neurons(labels_predicted1, top):
    labels_predicted1 = np.array(labels_predicted1, dtype=int)
    neuron_dict1 = {}
    for i in range(len(labels_predicted1)):
        if labels_predicted1[i] in neuron_dict1.keys():
            neuron_dict1[labels_predicted1[i]] += 1
        else:
            neuron_dict1[labels_predicted1[i]] = 1
    neuron_list1 = sorted(neuron_dict1, key=neuron_dict1.get, reverse=True)[:top]
    return neuron_list1, neuron_dict1

''' Load data '''
import gzip
import matplotlib.pyplot as plt
import numpy as np

f = gzip.open('t10k-images-idx3-ubyte.gz', 'r')

image_size = 28
num_images = 10000
```

سپس داده ها نرمالایز شده و labelها لود می شوند:

```

f.read(16)
buf = f.read(image_size * image_size * num_images)
data = np.frombuffer(buf, dtype=np.uint8).astype(np.float32)
data = data.reshape(num_images, image_size, image_size, 1)
print(data.shape)

data = data.reshape(num_images, 784)

data = data / 255.0
image = np.asarray(data[3].reshape(28, 28)).squeeze()

# plt.imshow(image)

data = data.reshape(num_images, 784)
# data = data[:1000, :]
vectors = data

''' Load labels '''
f = gzip.open('t10k-labels-idx1-ubyte.gz', 'r')
f.read(8)
labels_true = []
for i in range(0, num_images):
    buf = f.read(1)
    labels = np.frombuffer(buf, dtype=np.uint8).astype(np.int64)[0]
    labels_true.append(labels)
labels_true = np.array(labels_true)

```

سپس داده ها shuffle شده و وزن های اولیه داده می شوند:

```

''' Shuffle data '''
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(data, labels_true, test_size=0.1, random_state=42)

data = X_test
vectors = data
labels_true = y_test

''' Initialize parameters'''

m_cluster = 841
alpha = 0.5
lr = 0.6

''' Initialize weights '''

w = np.random.random([m_cluster, vectors.shape[1]])
w = np.round(w, decimals=3)
print(w)

```

الگوریتم SOM نوشته می شود:

```

''' SOM algorithm '''
epoch = 5
for k in range(epoch):
    for i in range(len(vectors)):
        D = np.zeros(len(w))
        for j in range(len(w)):
            D[j] = sum((w[j] - vectors[i]) ** 2)
        J = np.argmin(D)
        w[J] = w[J] + lr * (vectors[i] - w[J])
    ''' Round w till 3 decimal points '''
    w = np.round(w, decimals=5)
    lr *= alpha
w = np.round(w, decimals=1)
print(w)
print(w.shape)

```

در ادامه label ها استخراج می شوند:

```

''' Extract predicted labels '''
clusters_labels = {}
labels_predicted = np.zeros(len(vectors), dtype=int)
for j in range(len(vectors)):
    n_cluster = np.zeros(m_cluster)
    for i in range(len(w)):
        a = vectors[j]
        b = w[i]
        value = np.matmul(a, b)
        n_cluster[i] = value
    labels_predicted[j] = np.argmax(n_cluster)
    if labels_predicted[j] not in clusters_labels.keys():
        sample_list = np.zeros(10)
        sample_list[int(labels_true[j])] += 1
        clusters_labels[int(labels_predicted[j])] = sample_list
    else:
        value = clusters_labels[labels_predicted[j]]
        value1 = labels_true[j]
        sample_list = clusters_labels[labels_predicted[j]]
        sample_list[value1] += 1
        clusters_labels[labels_predicted[j]] = sample_list

labels_predicted = np.array(labels_predicted, dtype=int)
top_number = 5
neuron_list, neuron_dict = top_neurons(labels_predicted, top=top_number)

```

و سپس با فرمت زیر نمایش داده می شوند:

```

print('Table for class discrimination by neurons')
for item in neuron_list:
    print('Cluster: {}, classes: {}'.format(item, clusters_labels[item]))
# print('Classes: ', clusters_labels[item])
print('\nTable for number of class discrimination by neurons')
for item in neuron_list:
    print('Cluster: {}, classes: {}'.format(item, neuron_dict[item]))
counter = 0
for item in neuron_list:
    counter += neuron_dict[item]
print('\nTotal images of top {} neurons are: {}'.format(top_number, counter))
print(neuron_list)

```

در ادامه کد نیز برای درک بهتر نتایج خروجی، یک confusion matrix تشکیل می شود تا دید بهتری نسبت به عملکرد الگوریتم داشته باشیم:

```

''' Calculate confusion matrix '''
from sklearn.metrics import confusion_matrix
from sklearn.utils.multiclass import unique_labels


def plot_confusion_matrix(y_true, y_pred, classes,
                          normalize=False,
                          title=None,
                          cmap=plt.cm.Blues):
    """
    This function prints and plots the confusion matrix.
    Normalization can be applied by setting `normalize=True`.
    """
    if not title:
        if normalize:
            title = 'Normalized confusion matrix'
        else:
            title = 'Confusion matrix, without normalization'

    # Compute confusion matrix
    cm = confusion_matrix(y_true, y_pred)

    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
        # print("Normalized confusion matrix")

```

در این قسمت نیز از تابع به این صورت استفاده می شود:

```

class_names = unique_labels(labels_true, labels_predicted)
print('Here')
print(class_names)

# Plot normalized confusion matrix
plot_confusion_matrix(labels_true, labels_predicted, classes=class_names, normalize=True,
                      title='Normalized confusion matrix')

plt.show()

```

حال برای بخش های مختلف، موارد خواسته شده را اعمال می کنیم:

**الف)** شعاع مجاورت هر نورون را صفر قرار می دهیم:

در این قسمت 10 نورون برنده را نمایش می دهیم :

```
Table for class discrimination by neurons
Cluster: 31, classes: [ 84. 93. 111. 92. 94. 46. 93. 8. 89. 50.]
Cluster: 755, classes: [ 0. 0. 0. 0. 0. 44. 0. 97. 1. 64.]
Cluster: 542, classes: [0. 0. 0. 0. 0. 7. 0. 0. 0. 3.]
Cluster: 231, classes: [0. 6. 0. 2. 0. 0. 0. 0. 0. 0.]
Cluster: 235, classes: [0. 0. 0. 0. 0. 0. 0. 0. 4. 0.]
Cluster: 412, classes: [0. 0. 0. 0. 0. 1. 0. 0. 1. 0.]
Cluster: 592, classes: [0. 0. 0. 0. 0. 2. 0. 0. 0. 0.]
Cluster: 280, classes: [0. 0. 0. 0. 0. 1. 0. 0. 0. 0.]
Cluster: 765, classes: [0. 0. 0. 0. 0. 0. 0. 0. 1. 0.]
Cluster: 30, classes: [0. 0. 0. 0. 0. 1. 0. 0. 0. 0.]
```

به طور مثال خوشة 31 توانسته است 50 تا از عکس های کلاس 10 را دسته بندی کند.

همچنین تعداد آن ها بصورت زیر است :

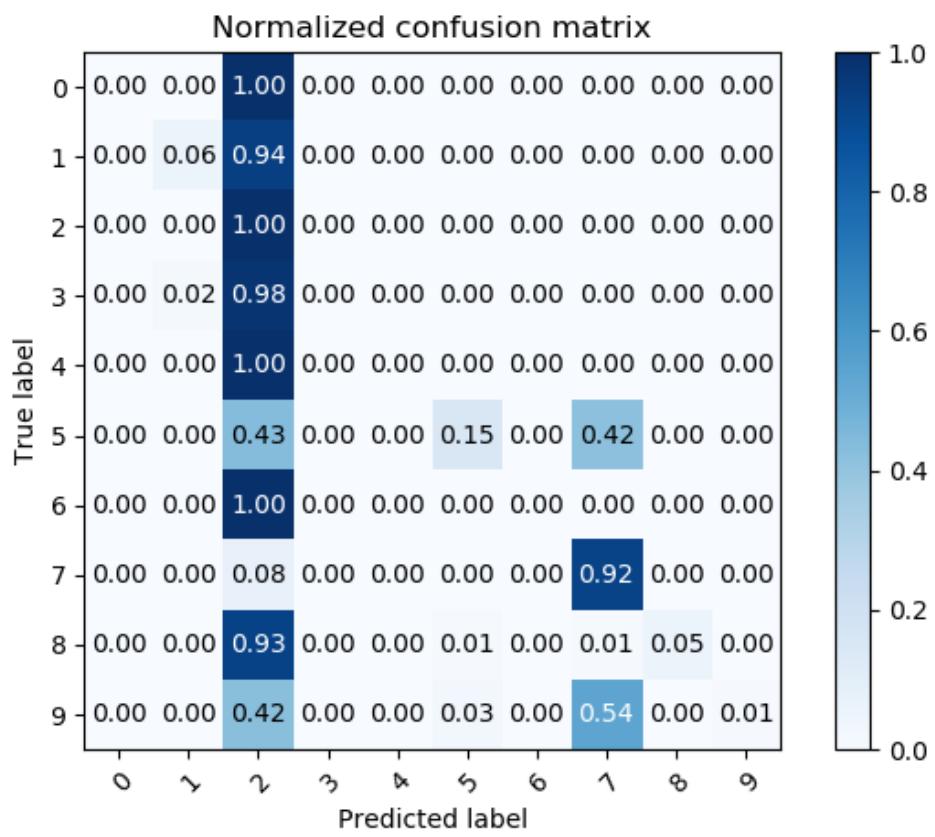
```
Table for number of class discrimination by neurons
Cluster: 31, classes: 760
Cluster: 755, classes: 206
Cluster: 542, classes: 10
Cluster: 231, classes: 8
Cluster: 235, classes: 4
Cluster: 412, classes: 2
Cluster: 592, classes: 2
Cluster: 280, classes: 1
Cluster: 765, classes: 1
Cluster: 30, classes: 1

Total images of top 10 neurons are: 995
```

به طور مثال خوشه 31 توانسته است 760 تصویر مختلف را دسته بندی کند.

در مجموع 995 تصویر توسط این 10 نورون برتر دسته بندی شده اند.

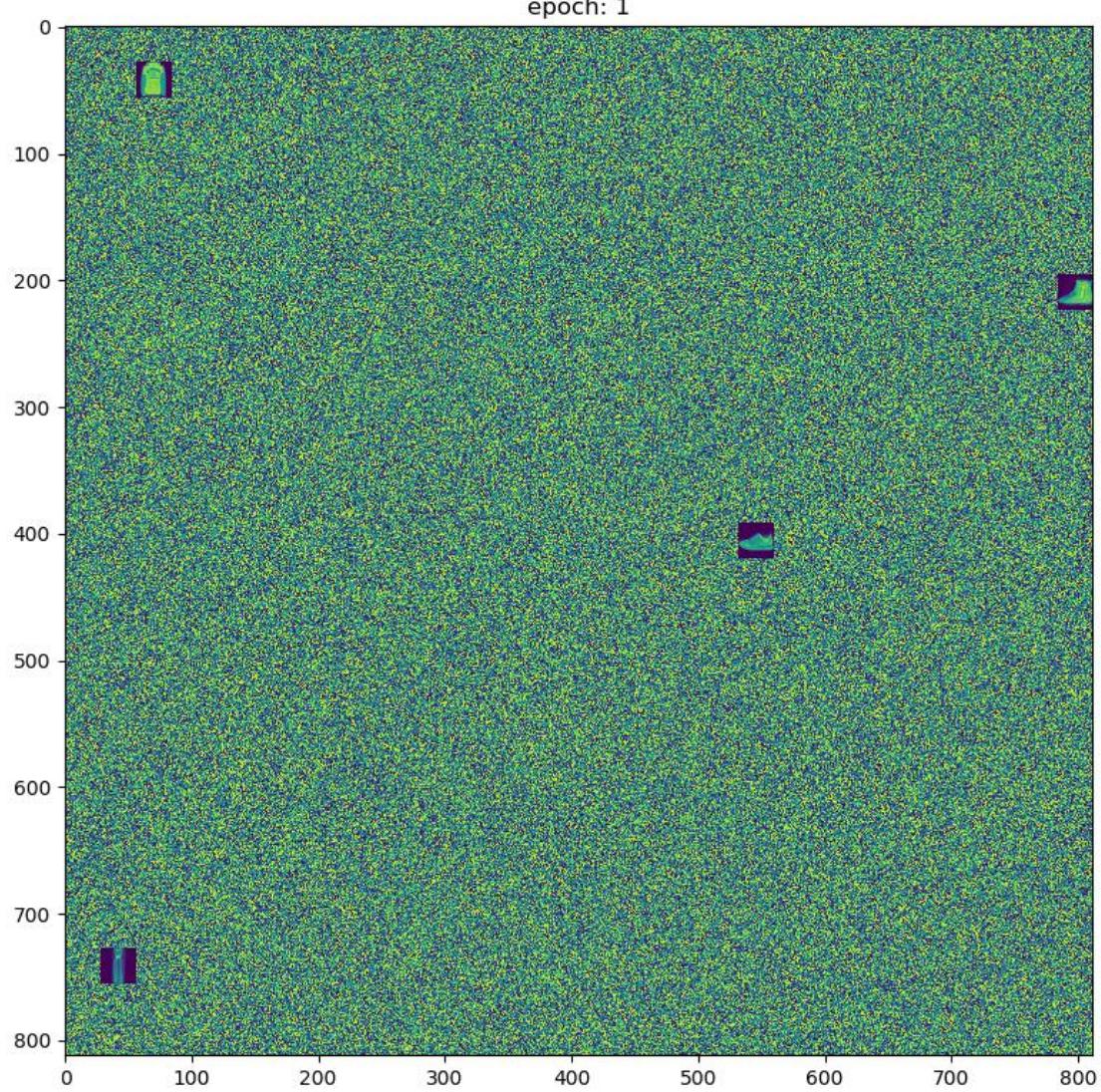
این الگوریتم به صورت زیر است:

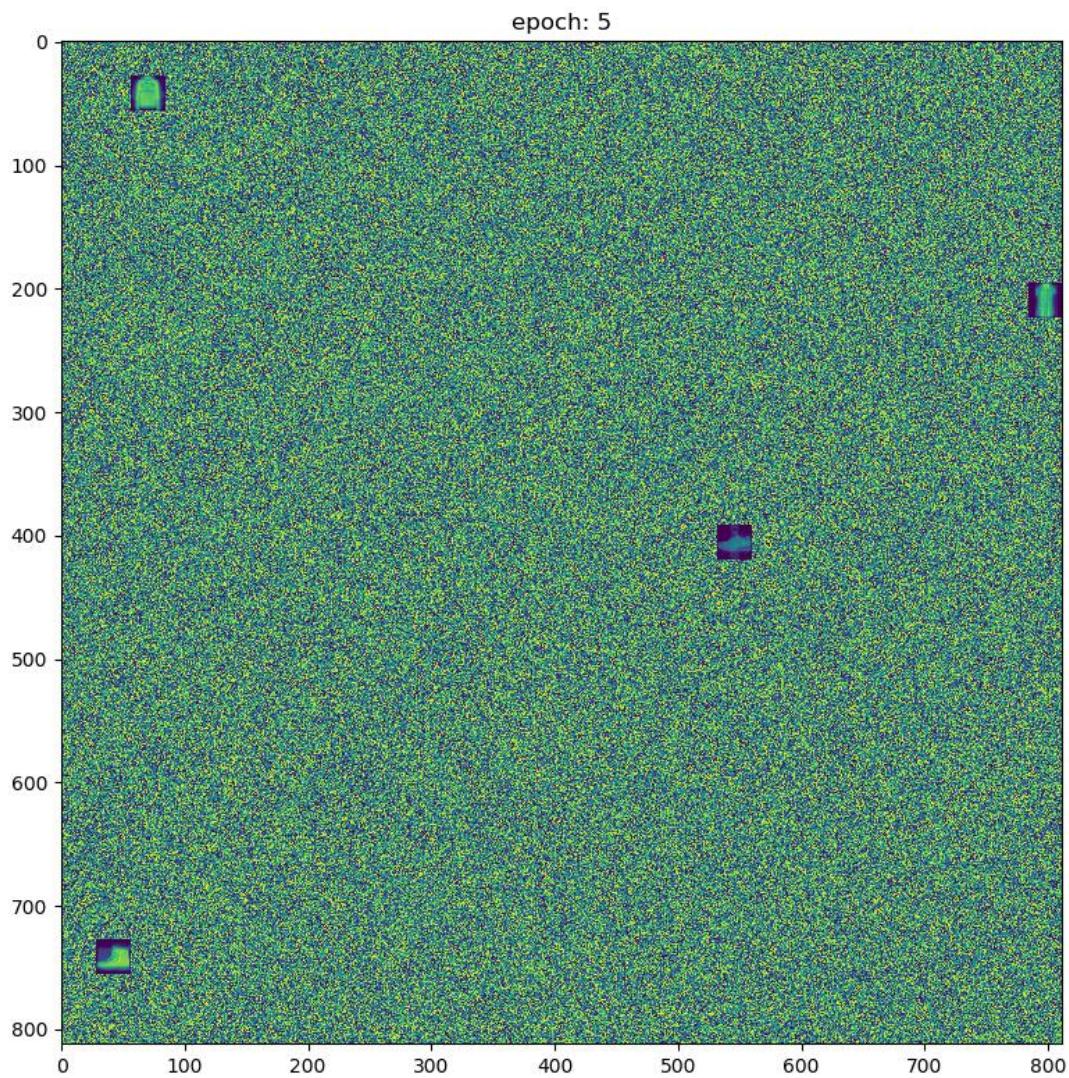


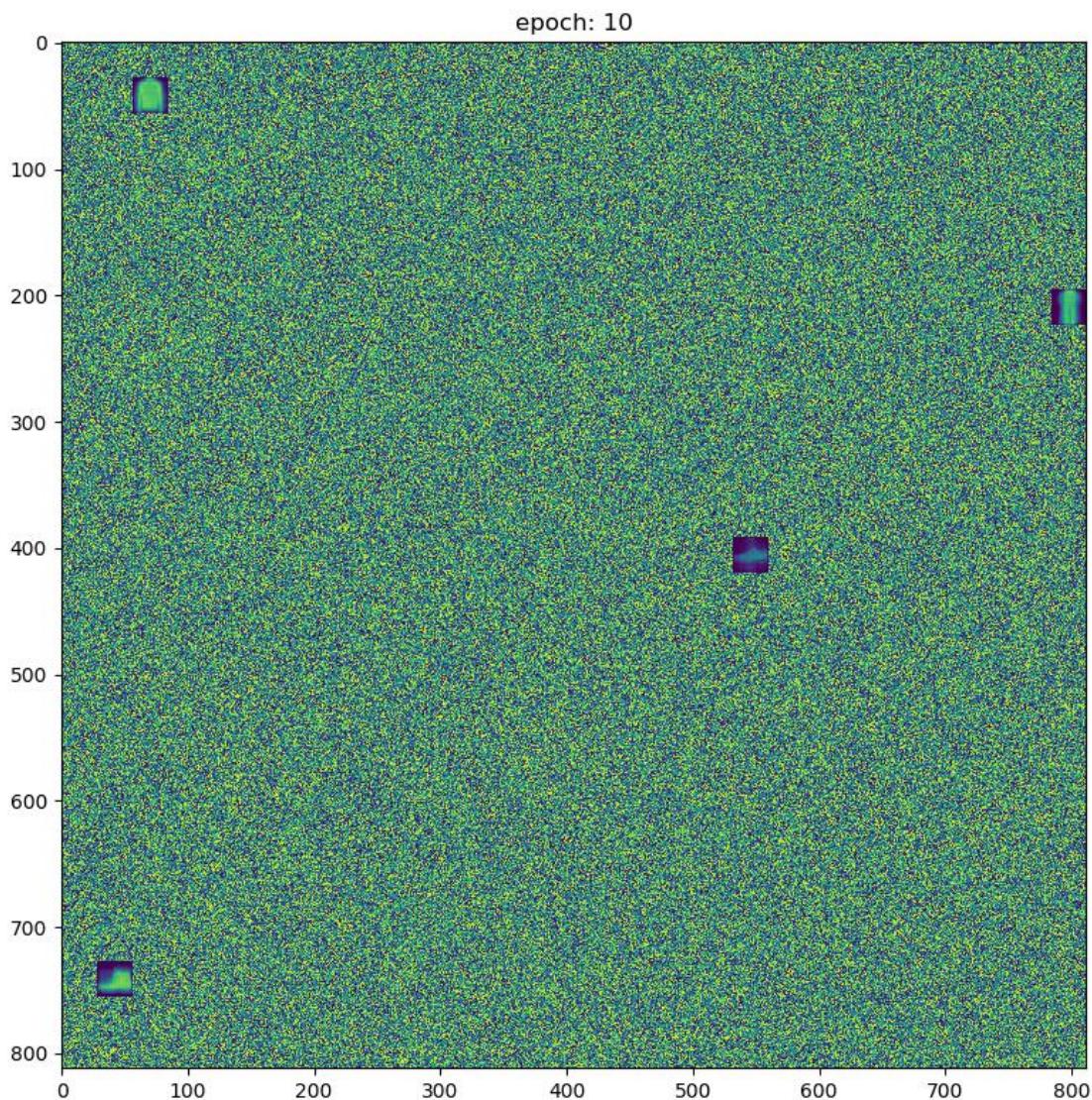
در اینجا اگر به صورت چشمی نگاه کنیم چند ردیف از سطر ها که نورون ها هستند اپدیت می شوند و مابقی حدودا ثابت می مانند.

در این بخش برنامه برای 10 ایپاک اجرا شده است و تصویر اپیاک های 1، 5 و 10 را در زیر می آوریم و همانطور که مشاهده می کنیم تغییر چندانی را شاهد نیستیم.

epoch: 1







ب) در این حالت نورون ها بصورت خطی با شعاع مجاورت  $R=1$  قرار می دهیم.

```
def gen_adjacent_node(arr, node=(0)):
    rows = len(arr)
    for r in [-1, 0, 1]:
        if r == 0:
            continue
            # check valid index
        if 0 <= node + r < rows:
            yield [node + r]
```

این تابع یک بردار را می گیرد و نودهای هم جوار را با شعاع  $R=1$  برمیگرداند.

بخش زیر را به الگوریتم SOM که قبل از نوشته اضافه می کنیم:

```
''' SOM algorithm '''
counter = 0
epoch = 10
for k in range(epoch):
    for i in range(len(vectors)):
        D = np.zeros(len(w))
        for j in range(len(w)):
            D[j] = sum((w[j] - vectors[i]) ** 2)
        J = np.argmin(D)
        '''Update with R = 1'''
        sample_arr = np.ones([(m_cluster)])
        new_list = node_giver(sample_arr, J)
        for item in new_list:
            w[item] = w[item] + lr * (vectors[i] - w[item])
        '''Update itself'''
        w[J] = w[J] + lr * (vectors[i] - w[J])
    if counter <= 150:
        my_img_show(w, k + 1)
    ''' Round w till 3 decimal points '''
    w = np.round(w, decimals=5)
    lr *= alpha
w = np.round(w, decimals=1)
print(w)
print(w.shape)
```

10 نورون برتر در زیر آورده شده اند:

```
Table for class discrimination by neurons
Cluster: 827, classes: [70. 98. 66. 88. 55. 3. 67. 0. 19. 12.]
Cluster: 807, classes: [ 1. 0. 0. 0. 0. 81. 0. 105. 44. 70.]
Cluster: 826, classes: [10. 0. 25. 1. 7. 10. 12. 0. 26. 24.]
Cluster: 828, classes: [ 1. 0. 20. 1. 32. 0. 14. 0. 5. 0.]
Cluster: 806, classes: [0. 0. 0. 0. 0. 7. 0. 0. 2. 6.]
Cluster: 814, classes: [0. 0. 0. 0. 0. 3. 0. 0. 0. 6.]
Cluster: 833, classes: [2. 0. 0. 2. 0. 0. 0. 0. 0. 0.]
Cluster: 834, classes: [0. 1. 0. 2. 0. 0. 0. 0. 0. 0.]
Cluster: 26, classes: [0. 0. 0. 0. 0. 1. 0. 0. 0. 0.]
Cluster: 813, classes: [0. 0. 0. 0. 0. 1. 0. 0. 0. 0.]
```

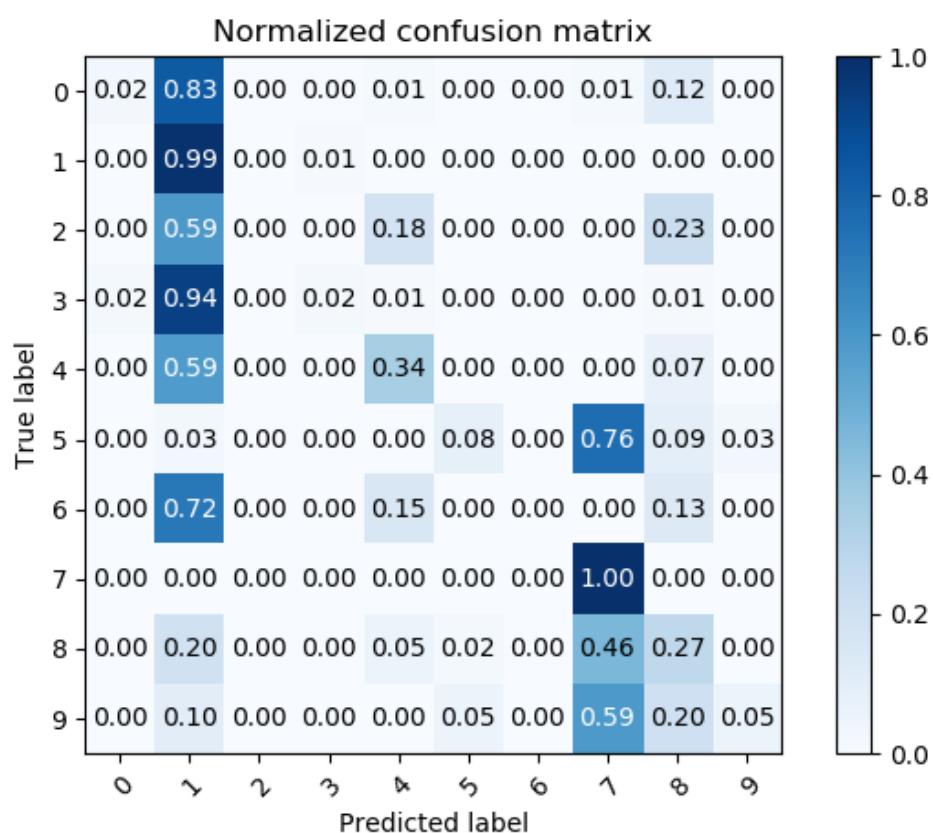
به طور مثال خوشه 827 توانسته است 12 تصویر از کلاس 10 را دسته بندی کند.

تعداد کل تصاویری که هر نورون دسته بندی می کند نیز در زیر آورده شده است:

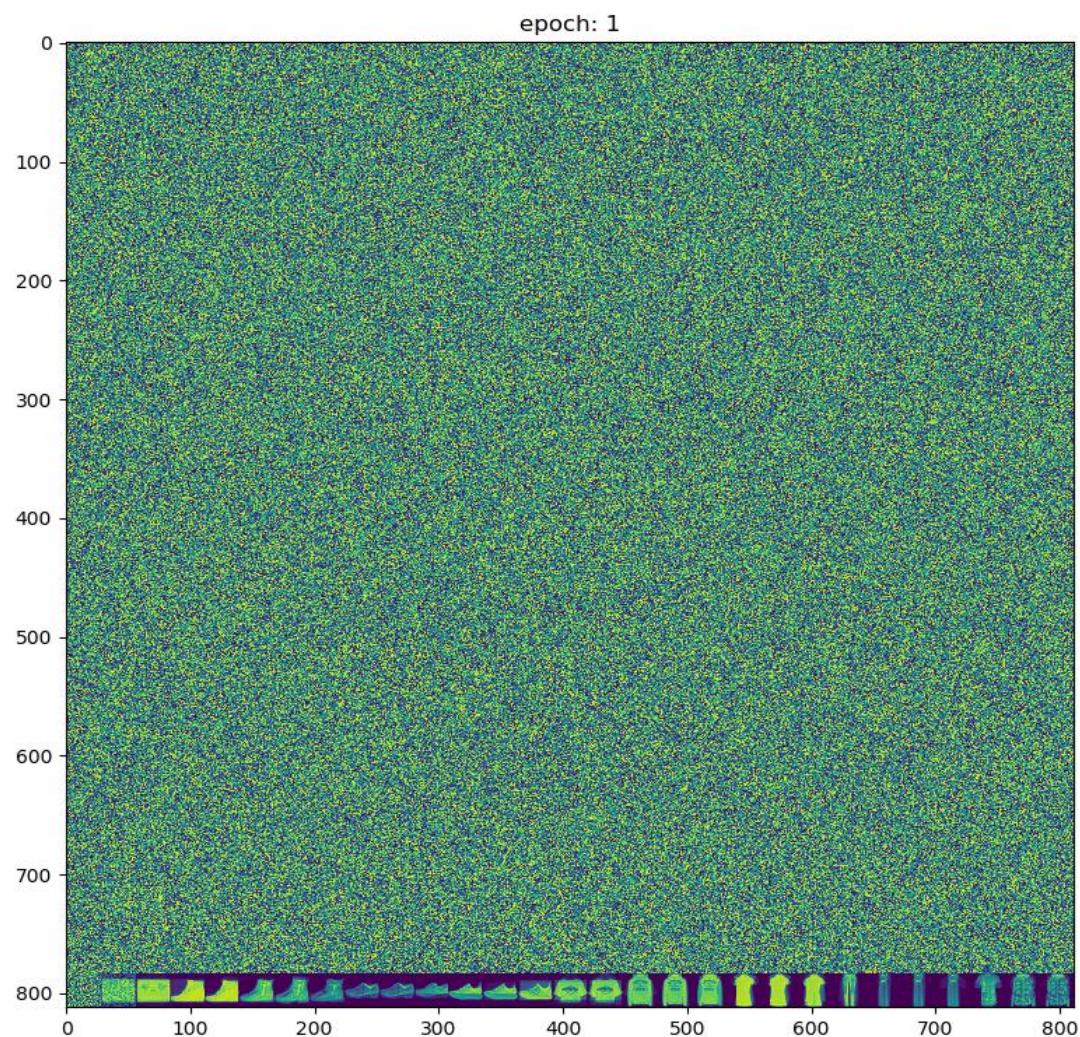
Table for number of class discrimination by neurons	
Cluster: 827,	classes: 478
Cluster: 807,	classes: 301
Cluster: 826,	classes: 115
Cluster: 828,	classes: 73
Cluster: 806,	classes: 15
Cluster: 814,	classes: 9
Cluster: 833,	classes: 4
Cluster: 834,	classes: 3
Cluster: 26,	classes: 1
Cluster: 813,	classes: 1
Total images of top 10 neurons are: 1000	

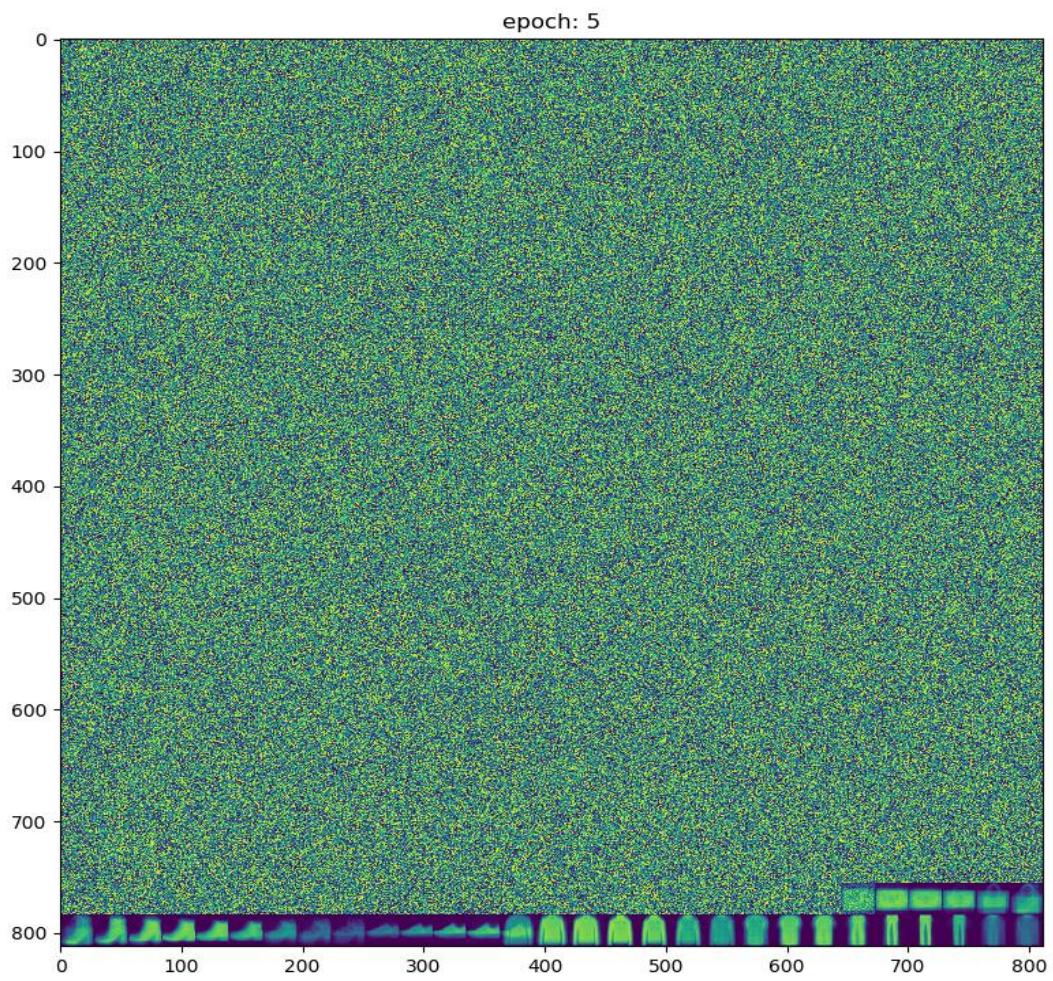
برای مثال خوشه 827 توانسته در مجموع 478 تصویر از کلاس های مختلف را دسته بندی کند.

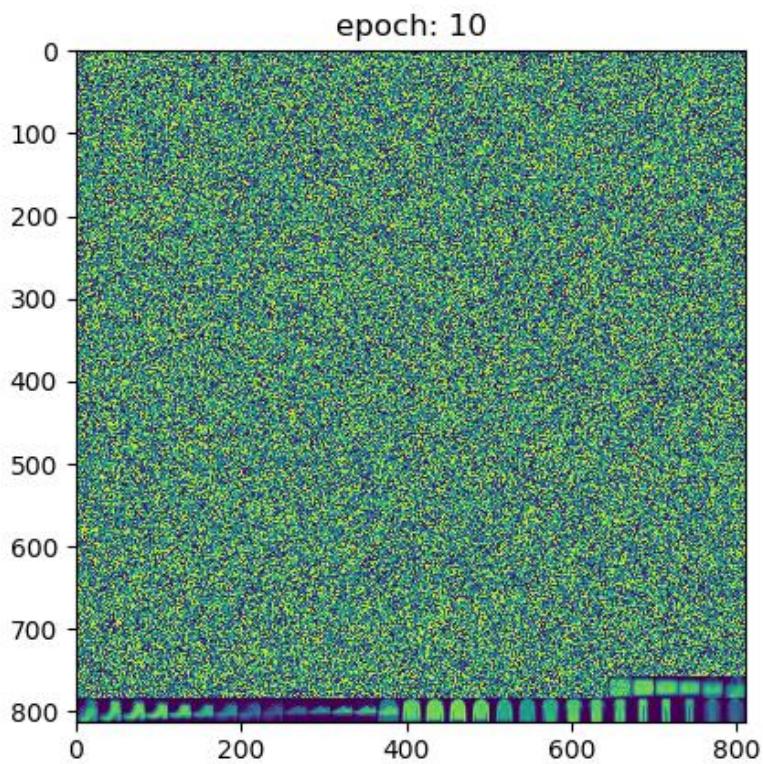
این الگوریتم به صورت زیر است:



در این بخش برنامه برای 10 ایپاک اجرا شده است و تصویر اپیاک های 1، 5 و 10 را در زیر می آوریم.







ج) در این حالت نورون ها را روی نودهای یک شبکه  $29 \times 29$  و با فرم مجاورت مربعی با شعاع  $R=1$  قرار می دهیم. کد های زیر را باید اضافه کنیم :

```
def gen_adjacent_node(matrix_2d, node=(0, 0)):
    rows = len(matrix_2d)
    columns = len(matrix_2d[0])
    for r in [-1, 0, 1]:
        for c in [-1, 0, 1]:
            if r == c == 0:
                continue
            # check valid index
            if 0 <= node[0] + r < rows and 0 <= node[1] + c < columns:
                yield [node[0] + r, node[1] + c]
```

این تابع یک ماتریس دوبعدی می گیرد و نودهای همچوار را در انتهای چاپ می کند.

```

def node_giver(arr, given_num1):
    # given_num = 29
    dimension1 = (arr.shape[0]) ** 2
    x_y = int(np.sqrt(dimension1))
    x_axis = given_num1 // x_y
    y_axis = given_num1 % x_y
    # print(x_axis)
    # print(y_axis)
    final_list1 = []
    a = gen_adjacent_node(arr, node=(x_axis, y_axis))
    for item in a:
        final_list1.append(item)
        # print(item)
    new_list1 = []
    for item in final_list1:
        value_x1 = item[0]
        value_y1 = item[1]
        final_value1 = value_x1 * int(np.sqrt(dimension1)) + value_y1
        new_list1.append(final_value1)
    return new_list1

```

این تابع نیز نودهای موردنظر را می گیرد و آن را flat می کند و در انتهای لیستی از نودهایی که باید تعویض شوند را می دهد.

در ادامه تنها با اضافه کردن بخش کوچکی به SOM میتوان الگوریتم را پیاده سازی کرد:

```

''' SOM algorithm '''
epoch = 5
for k in range(epoch):
    for i in range(len(vectors)):
        D = np.zeros(len(w))
        for j in range(len(w)):
            D[j] = sum((w[j] - vectors[i]) ** 2)
        J = np.argmin(D)
        ''' Update with R = 1 '''
        sample_arr = np.ones([int(np.sqrt(m_cluster)), int(np.sqrt(m_cluster))])
        new_list = node_giver(sample_arr, J)
        for item in new_list:
            w[item] = w[item] + lr * (vectors[i] - w[item])
        '''Update itself '''
        w[J] = w[J] + lr * (vectors[i] - w[J])
    ''' Round w til 3 decimal points '''
    w = np.round(w, decimals=5)
    lr *= alpha
w = np.round(w, decimals=1)
print(w)
print(w.shape)

```

در این بخش نودهای خروجی را گرفته و در ماتریس وزن ها آن را اپدیت می کنیم.

خروجی کد نیز به صورت زیر می باشد:

نودهای برتر شامل لیست زیر می باشند که در آن کلاس های جدا کننده موردنظر آورده شده اند:

Table for class discrimination by neurons									
Cluster: 125, classes: [ 8. 1. 31. 0. 12. 31. 20. 2. 53. 65.]									
Cluster: 20, classes: [ 2. 1. 63. 21. 78. 0. 47. 0. 4. 2.]									
Cluster: 463, classes: [ 0. 0. 0. 0. 0. 44. 0. 97. 20. 22.]									
Cluster: 216, classes: [ 1. 75. 0. 25. 0. 0. 2. 0. 0. 0.]									
Cluster: 157, classes: [ 51. 2. 0. 9. 1. 0. 4. 0. 1. 1.]									
Cluster: 186, classes: [ 13. 9. 0. 33. 1. 0. 8. 0. 1. 0.]									
Cluster: 492, classes: [ 0. 0. 0. 0. 0. 8. 0. 3. 13. 16.]									
Cluster: 126, classes: [ 6. 1. 15. 5. 1. 0. 9. 0. 0. 0.]									
Cluster: 101, classes: [ 1. 0. 0. 0. 0. 18. 0. 0. 4. 0.]									
Cluster: 217, classes: [ 0. 10. 0. 0. 0. 0. 0. 0. 0. 0.]									

به طور مثال خوش 125 تعداد 65 تصویر از کلاس 10 را دسته بندی می کند.

مجموع عکس های طبقه بندی شده توسط نورون ها نیز به صورت زیر می باشد:

Table for number of class discrimination by neurons

Cluster: 125, classes: 223

Cluster: 20, classes: 218

Cluster: 463, classes: 183

Cluster: 216, classes: 103

Cluster: 157, classes: 69

Cluster: 186, classes: 65

Cluster: 492, classes: 40

Cluster: 126, classes: 37

Cluster: 101, classes: 23

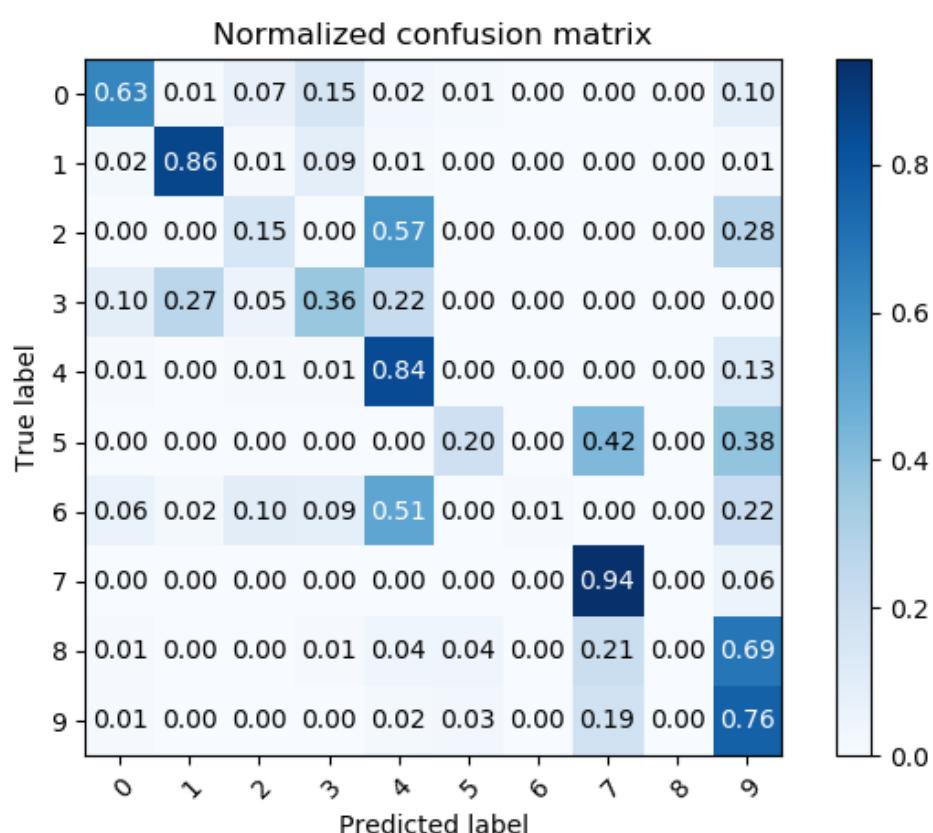
Cluster: 217, classes: 10

Total images of top 10 neurons are: 971

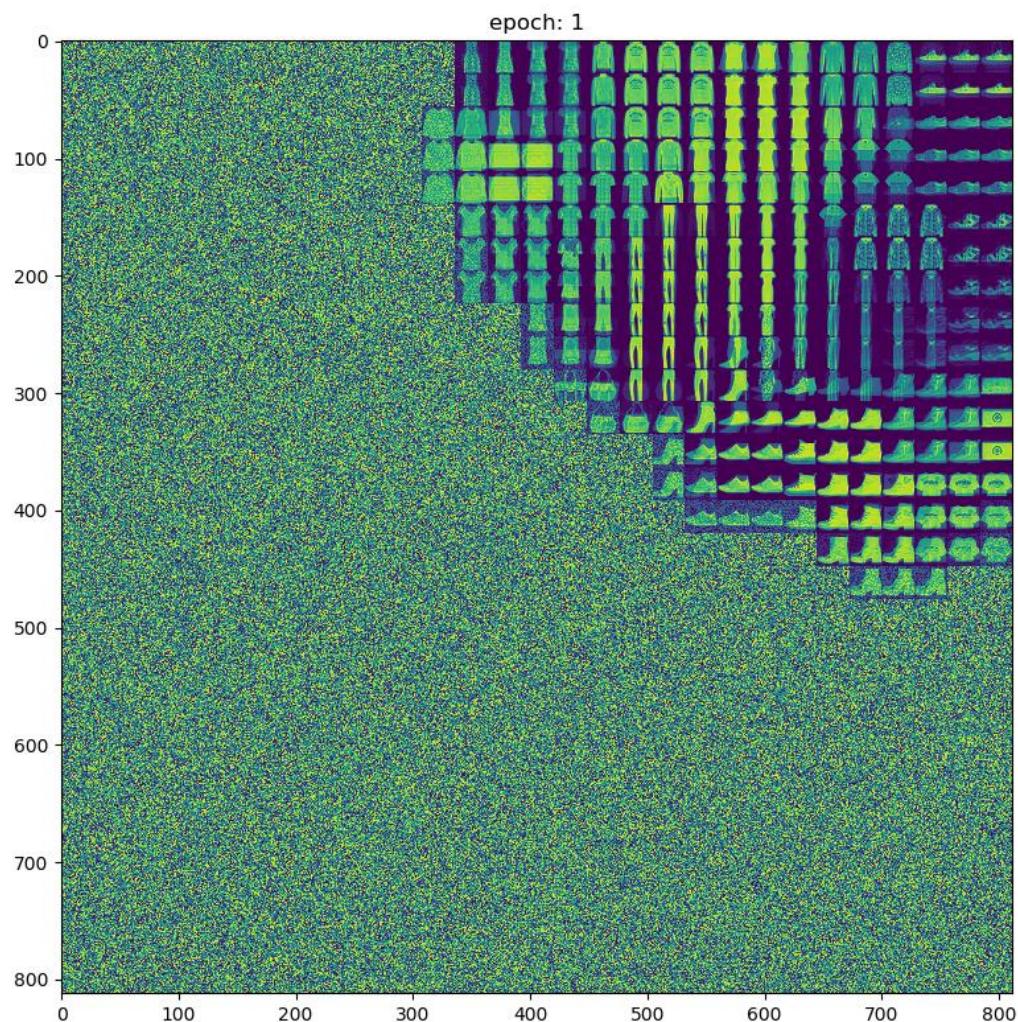
در عکس بالا کلاستر 125 تعداد 223 تصویر از کلاس های مختلف را دسته بندی می کند.

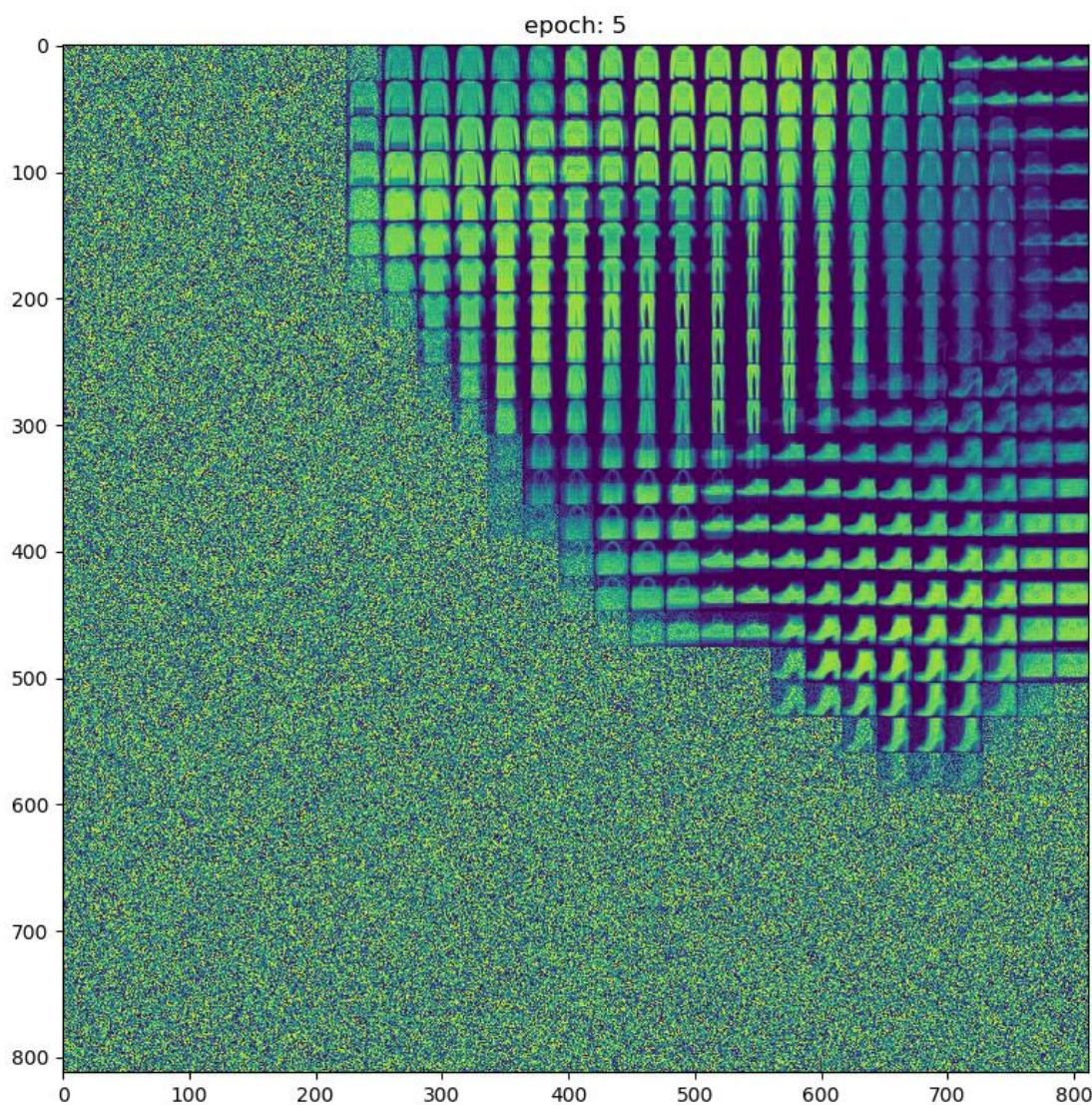
در مجموع 971 تصویر توسط این 10 نورون برتر دسته بندی شده اند.

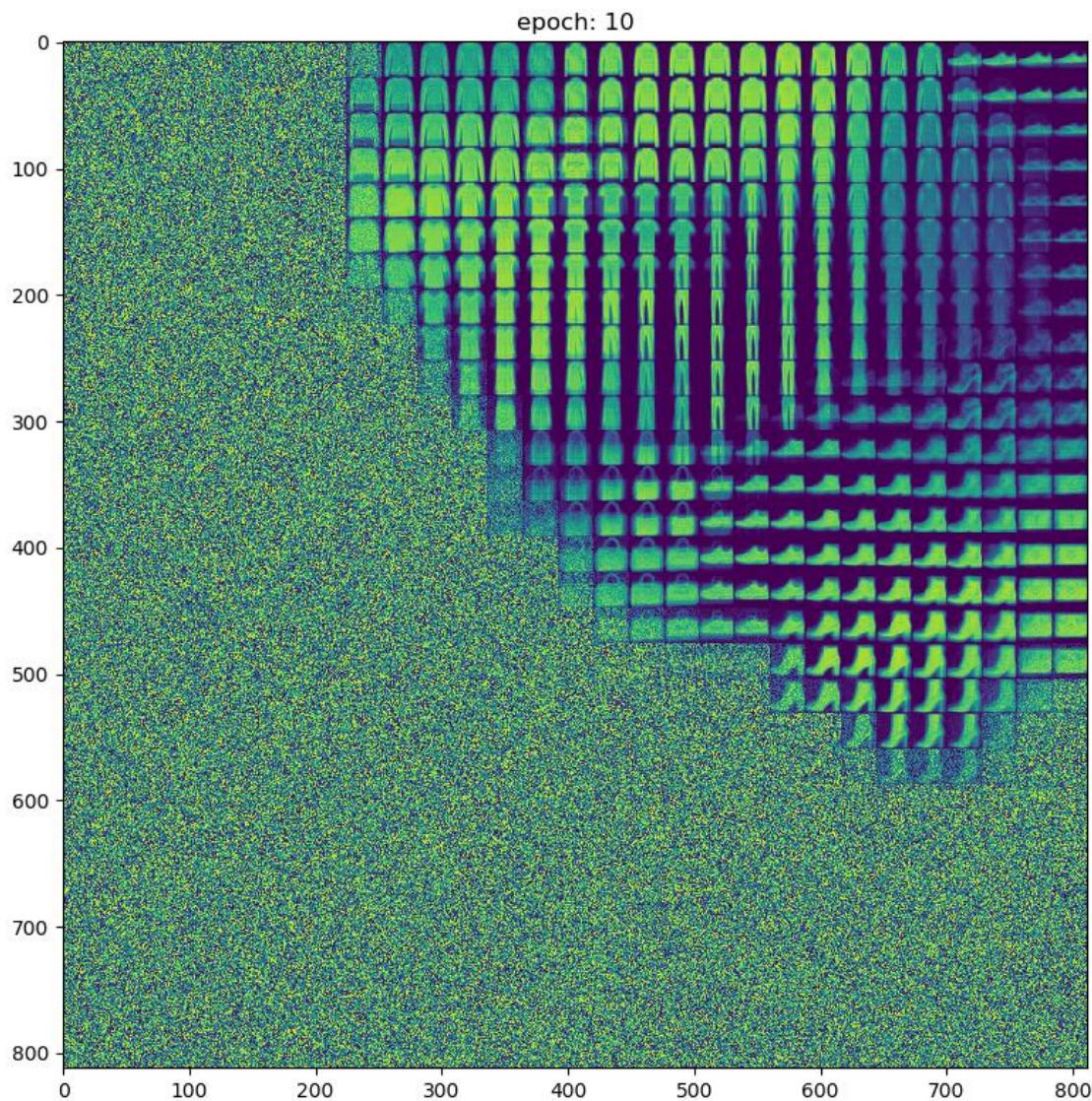
این الگوریتم به صورت زیر است:



در این بخش برنامه برای 10 ایپاک اجرا شده است و تصویر اپیاک های 1، 5 و 10 را در زیر می آوریم.







تحلیل : همانطور که در تصاویر مشاهده می کنیم عملکرد حالت ج بهتر از ب و ب بهتر از الف است.

زمانیکه با یک شعاع مجاورتی نورون ها را کنار هم قرار می دهیم نورون ها با هم همکاری دارند و در نتیجه به دسته بندی بهتری از تصاویر میرسیم و هر چه این همکاری کمرنگ تر می شود میبینیم که کیفیت دسته بندی پایینتر می آید.

## سوال ۲ – MaxNet

در این قسمت می خواهیم الگوریتم MaxNet را پیاده سازی کنیم که این الگوریتم با دریافت یک بردار ورودی بیشینه‌ی آن را پیدا می‌کند و همچنین می خواهیم الگوریتم را بگونه‌ای تغییر دهیم که از آن برای مرتب سازی عناصر بردار هم استفاده کنیم.

در این الگوریتم در هر بار اجرا برای هر یک از عناصر بردار ، مجموع بقیه‌ی عناصر را با یک ضریبی از مقدار آن عنصر بردار کم کرده و از یک تابع فعال ساز عبور می‌دهیم و اینکار را تا جایی ادامه می‌دهیم که همه‌ی عناصر بجز بیشینه‌ی بردار صفر شوند و یا همه‌ی مقادیر باقی مانده با هم برابر باشند. اگر اندازه‌ی بردار  $m$  باشد ضریبی که انتخاب می‌کنیم باید در بازه‌ی  $[0,1/m]$  باشد.

در این سوال ضریب برابر با  $0.15$  انتخاب شده است. هر چقدر ضریب کوچکتر باشد تعداد گام‌ها تا رسیدن به بیشینه‌ی بردار بیشتر خواهد شد.

تابع فعال ساز برای این سوال بصورت زیر است:

```
def f(input):
    if input >= 0:
        return input
    else:
        return 0
```

در گام اول الگوریتم را برای پیدا کردن بیشینه پیاده سازی می‌کنیم :

```

x_old = [1.2, 1.1, 1, 0.9, 0.95, 1.15]
x = [1.2, 1.1, 1, 0.9, 0.95, 1.15]
e = 0.15
x = np.array(x)
x_new = np.zeros(6)
iter = 0
while True:
    for i in range(len(x)):
        sum = 0
        for j in range(len(x)):
            if i != j:
                sum += x[j]
        x_new[i] = f(x[i] - e * sum)
    x = np.copy(x_new)
    non_zero_index = np.nonzero(x)
    for k in range(len(non_zero_index) - 1):
        if np.equal(x[non_zero_index[k]], x[non_zero_index[k + 1]]):
            break
    if (np.count_nonzero(x)) == 1:
        break
    iter += 1
    print("iteration", iter, ":", x)
print("iteration", iter + 1, ":", x)
index = np.nonzero(x)
for ind in range(len(index)):
    print("maximum element :" ,x_old[ind])

```

برای مثال داده شده مراحل به روز رسانی واحدها و همچنین بیشینه مقدار به صورت زیر است :

```

C:\Users\Mahsa\anaconda3\envs\PythonGPU\python.exe C:/Users/Mahsa/Desktop/University/te
iteration 1 : [0.435  0.32   0.205  0.09   0.1475  0.3775]
iteration 2 : [0.264   0.13175  0.        0.        0.        0.197875]
iteration 3 : [0.21455625 0.06246875 0.        0.        0.        0.1385125 ]
iteration 4 : [0.18440906 0.00950844 0.        0.        0.        0.09695875]
iteration 5 : [0.16843898 0.        0.        0.        0.        0.06787112]
iteration 6 : [0.15825832 0.        0.        0.        0.        0.04260528]
iteration 7 : [0.15186752 0.        0.        0.        0.        0.01886653]
iteration 8 : [0.14903754 0.        0.        0.        0.        0.          ]
maximum element : 1.2

Process finished with exit code 0

```

زمانیکه تمامی مقادیر از یک مقداری مثل بتا بزرگتر هستند میتوانیم در ابتدا به اندازه بتا از همه عناصر آرایه کم کنیم سپس همین الگوریتم را اجرا کنیم.

در گام دوم الگوریتم را برای مرتب کردن اعداد از بزرگ به کوچک پیاده سازی می کنیم. برای اینکار در الگوریتم قبلی در هر مرحله که یکی از عناصر بردار صفر شد آن را بعنوان کوچکترین عضو به انتهای یک بردار با ابعاد بردار ورودی اضافه می کنیم و اینکار را برای همه عناصری که صفر می شوند انجام میدهیم و در پایان الگوریتم که بزرگترین عضو پیدا شد آن را به ابتدای بردار اضافه می کنیم.

مشکلی که ممکن است در این حالت پیش بیاید اینست که چند عنصر همزن با هم صفر شوند که در این مثال برای مقدار اپسیلون  $0.15$  این اتفاق می افتد. بنابراین باید مقدار اپسیلون را کوچکتر کنیم تا مشکل حل شود و در اینجا مقدار آن را  $0.09$  قرار می دهیم.

کد این قسمت را در زیر می بینیم قسمت های مشخص شده به کد قسمت اول اضافه شده است :

```
x_old = [1.2, 1.1, 1, 0.9, 0.95, 1.15]
x = [1.2, 1.1, 1, 0.9, 0.95, 1.15]
e = 0.09
x = np.array(x)
x_new = np.zeros(6)
sorted_descending = [] # max to min
while True:
    for i in range(len(x)):
        sum = 0
        for j in range(len(x)):
            if i != j:
                sum += x[j]
        x_new[i] = f(x[i] - e * sum)
        if x_new[i] == 0 and not (x_old[i] in sorted_descending):
            sorted_descending = np.insert(sorted_descending, 0, x_old[i])
    x = np.copy(x_new)
    non_zero_index = np.nonzero(x)
    for k in range(len(non_zero_index) - 1):
        if np.equal(x[non_zero_index[k]], x[non_zero_index[k + 1]]):
            break
    if (np.count_nonzero(x)) == 1:
        break
    # print(x)
    temp = 0
    index = np.nonzero(x)
    for ind in range(len(index)):
        print(x_old[ind])
        temp = x_old[ind]
        sorted_descending = np.insert(sorted_descending, 0, temp)
print(sorted_descending)
```

خروجی الگوریتم برای اپسیلون برابر  $0.15$  :

```
C:\Users\Mahsa\anaconda3\envs\Python6  
[1.2 1.15 1.1 0.95 0.9 1. ]  
  
Process finished with exit code 0
```

همانطور که میبینیم برای مرتب کردن سه عدد آخر که همزمان صفر شده بودند به مشکل می خورد.

خروجی الگوریتم برای اپسیلون برابر 0.09 :

```
C:\Users\Mahsa\anaconda3\envs\Python6  
[1.2 1.15 1.1 1. 0.95 0.9 ]  
  
Process finished with exit code 0
```

در گام سوم الگوریتم را برای مرتب کردن اعداد از کوچک به بزرگ پیاده سازی می کنیم. برای اینکار مشابه حالت قبل در هر بار اجرای الگوریتم که یکی از عناصر صفر شد باید آن را در یک بردار جدید ذخیره کنیم و اینبار آن را به انتهای بردار جدید اضافه میکنیم و در نهایت آخرین عددی که به انتهای بردار اضافه می شود ، بزرگترین عدد است که الگوریتم پیدا می کند و به این ترتیب اعداد بصورت صعودی مرتب می شوند.

کد این قسمت را در زیر می بینیم قسمت های مشخص شده به کد قسمت اول اضافه شده است:

```

x_old = [1.2, 1.1, 1, 0.9, 0.95, 1.15]
x = [1.2, 1.1, 1, 0.9, 0.95, 1.15]
e = 0.09
x = np.array(x)
x_new = np.zeros(6)
sortedAscending = [] # min to max
while True:
    for i in range(len(x)):
        sum = 0
        for j in range(len(x)):
            if i != j:
                sum += x[j]
        x_new[i] = f(x[i] - e * sum)
    if x_new[i] == 0 and not (x_old[i] in sortedAscending):
        sortedAscending = np.append(sortedAscending, x_old[i])
    x = np.copy(x_new)
    non_zero_index = np.nonzero(x)
    for k in range(len(non_zero_index) - 1):
        if np.equal(x[non_zero_index[k]], x[non_zero_index[k + 1]]):
            break
    if (np.count_nonzero(x)) == 1:
        break
    # print(x)
    temp = 0
    index = np.nonzero(x)
    for ind in range(len(index)):
        print(x_old[ind])
        sortedAscending = np.append(sortedAscending, x_old[ind])
    temp = x_old[ind]
    print(sortedAscending)

```

خروجی الگوریتم برای اپسیلوون برابر 0.15 :

```

C:\Users\Mahsa\anaconda3\envs\Python
[1.  0.9  0.95 1.1  1.15 1.2 ]

Process finished with exit code 0

```

مشکل هم زمان صفر شدن چند عنصر را در اینجا هم داریم بنابراین اپسیلوون را کوچکتر می کنیم.

خروجی الگوریتم برای اپسیلوون برابر 0.09 :

```

C:\Users\Mahsa\anaconda3\envs\Python
[0.9  0.95 1.   1.1  1.15 1.2 ]

Process finished with exit code 0

```

### سوال 3 Mexican Hat

این شبکه توسعه ای از MaxNet هست و به جای اینکه بین  $m$  نود ، بیشینه را پیدا کند ، نودی را پیدا می کند که خودش و همسایگانش اعداد بزرگتری را شامل باشد به عبارتی خود نود و همسایگانش به شکل نرم بیشینه را تعیین می کند. در این سوال داده ها به شکل خطی کنار هم قرار دارند و با درنظر گرفتن شعاع هم جواری تعیین شده در سوال سعی می کنیم مقدار بیشینه را پیدا کنیم. نود هایی که داخل شعاع همجواری  $R1$  قرار می گیرند ، نود های همکار نامیده می شوند و در هر بروزرسانی نودی مثل  $i$  ، خود نود  $i$  و نودهای همکارش با یک تحریک مثبت باعث هم افزایی نود  $i$  می شوند. شعاع همجواری دیگری داریم که با  $R2$  مشخص می کنیم . نودهایی که در شعاع  $(R1, R2)$  قرار می گیرند در هر بروزرسانی با تحریک منفی باعث کاهش مقدار نود  $i$  می شوند و نودهایی که خارج از شعاع  $R2$  هستند، نود های خنثی در بروزرسانی نود  $i$  هستند.

در این سوال خواسته شده تا در بردار زیر بیشینه را با شبکه Mexican Hat پیدا کنیم :

[ 0.27, 0.35, 0.44, 0.58, 0.66, 0.77, 0.4, 0.32, 0.20, 0.15, 0.08 ]

تابع فعال ساز هم همانطور که در صورت سوال خواسته شده به شکل زیر تعریف می کنیم:

```
def f(input):
    if input < 0:
        return 0
    elif 0 <= input < 2:
        return input
    else:
        return 2
```

الگوریتم Mexican Hat بصورت زیر پیاده شده است:

```

while t < t_max:
    for i in range(len(x)):
        x1, x2, x3 = 0, 0, 0
        v = 0
        for k in range(-R1, R1+1):
            if i + k - R1 < 0 or i + k + R1 > 10:
                x1 += 0
            else:
                x1 += C1 * x_old[i + k]
        for k in range(-R2, -R1 - 1+1):
            if i + k - R2 < 0 or i + k - R1 - 1 > 10:
                x2 += 0
            else:
                x2 += C2 * x_old[i + k]
        for k in range(R1 + 1, R2+1):
            if i + k - R1 - 1 < 0 or i + k + R2 > 10:
                x3 += 0
            else:
                x3 += C2 * x_old[i + k]
        x_new[i] = f(x1+x2+x3)

    x_old = x_new
    x_old_per_iter[t - 1] = x_old
    plt.plot(x_old_per_iter[t-1], label='t = {}'.format(t-1))
    t += 1

plt.legend()
plt.show()

```

الف) مقدار  $R1=0$  و مقدار  $R2=\infty$  :

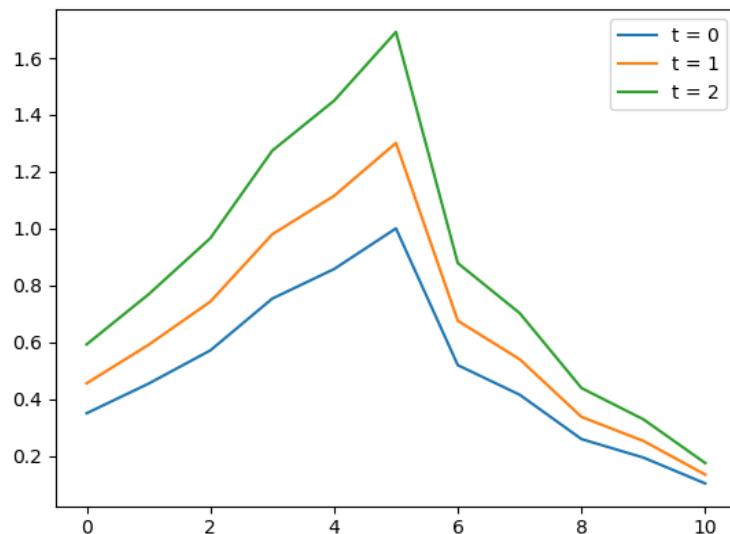
در این حالت انتظار داریم مشابه الگوریتم MaxNet عمل کند. مقداردهی اولیه ی وزن ها و تعداد iteration ها را در شکل زیر میبینیم :

```

R1 = 0
R2 = 11
C1 = 1.3
C2 = -1
t = 1
t_max = 4

```

نمودار مقدار سیگنال خروجی برحسب index اعضای آرایه در هر iteration را در شکل زیر میبینیم:



: R2=3 و R1=1 مقدار

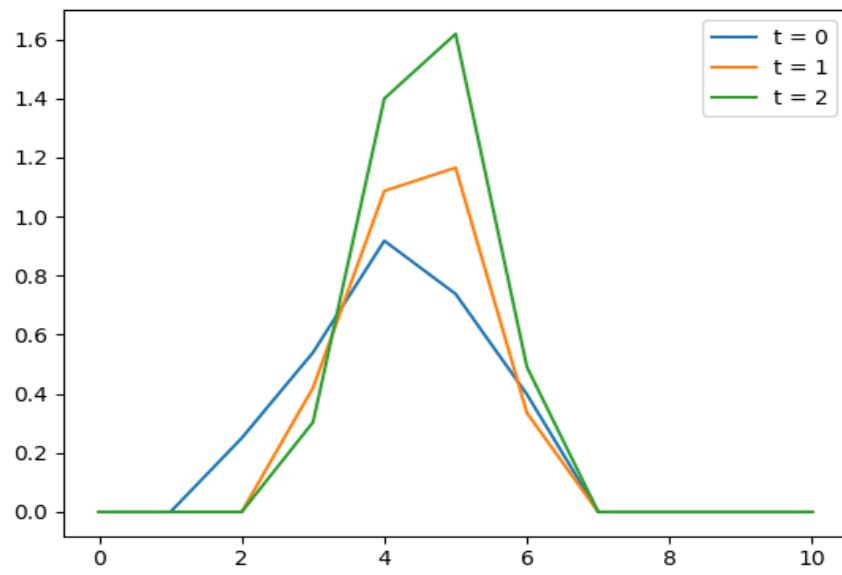
مقداردهی اولیه ای وزن ها و تعداد iterationها را در شکل زیر میبینیم :

```

R1 = 1
R2 = 3
C1 = 0.6
C2 = -0.4
t = 1
t_max = 4

```

نمودار مقدار سیگنال خروجی برحسب index اعضای آرایه در هر iteration را در شکل زیر میبینیم:



همانطور که انتظار داشتیم بیشینه در نمودار حالت ب نرم تر است ولی در حالت الف مانند شبکه maxNet تنها یک نقطه‌ی بیشینه را می‌بینیم ولی در حالت ب یک همسایگی نرم از نقطه‌ی بیشینه می‌بینیم. و به مرور با تکرار الگوریتم دامنه کوچکتر و مقدار بیشینه بیشتر می‌شود.

## سوال 4 Hamming Net

در این سوال پیاده سازی یک شبکه Hamming Net خواسته شده است. 10 بردار ورودی به ابعاد 6 داریم که می خواهیم ببینیم هر کدام از آنها به کدامیک از 3 بردار مرجع با همین ابعاد، نزدیک تر است و در خروجی شماره‌ی بردار مرجع نزدیکتر به آن را بر می‌گردانیم.

این شبکه شامل یک لایه ورودی است که همان 10 بردار ورودی ما هستند و یک ورودی بایاس که مقدار آن برابر است با  $n/2 = 3$  و لایه‌ی خروجی که شامل 3 نورون است.

مقادیر بردارهای ورودی و مرجع به شکل زیر است:

```
v = np.zeros((10, 6))
v[0] = [1, -1, 1, 1, -1, 1]
v[1] = [-1, 1, 1, -1, 1, -1]
v[2] = [1, 1, 1, -1, -1, -1]
v[3] = [-1, -1, -1, 1, 1, 1]
v[4] = [1, 1, 1, 1, 1, 1]
v[5] = [-1, -1, 1, -1, -1, -1]
v[6] = [-1, -1, -1, 1, -1, -1]
v[7] = [1, 1, -1, -1, 1, 1]
v[8] = [1, 1, -1, 1, 1, 1]
v[9] = [1, 1, 1, -1, 1, 1]

e = np.zeros((3, 6))
e[0] = [1, -1, 1, -1, 1, -1]
e[1] = [-1, 1, -1, 1, -1, 1]
e[2] = [1, 1, 1, 1, 1, 1]
```

مقدار دهی اولیه ماتریس وزن و مقدار بایاس هم بصورت زیر انجام می‌شود:

```
w = [e[0], e[1], e[2]]
w = 0.5 * np.array(w)
w = w.T
b = 3
```

و در نهایت الگوریتم این شبکه به شکل کد زیر است و از بین فواصلی که برای هر ورودی با بردارهای مرجع مختلف پیدا می‌کنیم کمینه‌ی آن را انتخاب می‌کنیم بعنوان شبیه ترین بردار مرجع به آن ورودی و چاپ می‌کنیم.

```
y_in = np.zeros(len(e))
out = np.zeros(len(v))

for i in range(len(v)):
    for j in range(len(e)):
        y_in[j] = b + np.sum(v[i] * w[:, j])
    out[i] = np.argmax(y_in)

print(out)
```

خروجی الگوریتم بصورت زیر است :

```
C:\Users\Mahsa\anaconda3\envs\Pyt
[2. 0. 0. 1. 2. 0. 1. 2. 2. 2.]
```

