سؤال ۱)

در این سؤال می‌خواهیم با استفاده از شبکه SOM داده‌های Fashion mnsit که شامل ۱۰ طبقه هستند را کلاستری کنیم
برای این کار ۸۴۱ نود کلاستر در نظر می‌گیریم و میبینیم که فقط برخی از آن‌ها نود های برنده هستند

برای این کار ابتدا داده‌ها را خوانده و یک ماتریس وزن وزن به اندازه ۸۴۱ که تعداد کلاستر ها است در ۷۸۴ که تعداد پیشکسل های عکس‌های ورودی است درست می‌کنیم و مقادیر آن را رندوم می دهیم:

```
In [3]:
class_names = ['T-shirt/top', 'Trouser', 'Pullover', 'Dress', 'Coat',
               'Sandal', 'Shirt', 'Sneaker', 'Bag', 'Ankle boot']
plt.figure(figsize=(10,10))
for i in range(25):
    plt.subplot(5,5,i+1)
    plt.xticks([])
    plt.yticks([])
    plt.grid(False)
    plt.imshow(x_train_image[i], cmap=plt.cm.binary)
    plt.xlabel(class_names[y_train[i]])
plt.show()
```

می‌توانیم ۲۵ داده اول را با خروجی های آن نیز نمایش دهیم:

In [14]:
```
# Number of Neurons 841
m = 841

# Initial Weights:
print(weights)
print(weights.shape)

#initial learning rate
alpha = 0.6
decay_alpha = 0.5

# Initial neighbour radius
R = 0

# Normalize Input
x_train = x_train / 255.0
x_test = x_test / 255.0
x_test = x_test[:200].reshape(200, 28 * 28)
print(x_train)
```

```
[[0.56527061 0.26954774 0.75263499 ... 0.45780132 0.12272672 0.4058609 ]
 [0.76890253 0.05134861 0.5986841  ... 0.46488896 0.37454181 0.00995268]
 [0.89588719 0.82300959 0.35678014 ... 0.78405991 0.65298173 0.07482362]
 ...
 [0.72343505 0.32393345 0.02801468 ... 0.52497297 0.35437339 0.14095843]
 [0.08488874 0.48305394 0.97106854 ... 0.92086692 0.50767967 0.83175544]
 [0.44498613 0.15189644 0.4659118  ... 0.09484155 0.33644564 0.49500344]]
(784, 841)
[[0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 ...
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]]
```

می‌بینیم که ماتریس وزن و ورودی به شکل زیر می شود:
حال می‌توانیم شبکه را با الفا برابر ۰.۶ و الفا را در هر دور ۰.۵ برابر می‌کنیم برای شبکه داریم:

In [6]:
```
for _ in range(5):
    for i in range(len(x_train)):
        print("input number: ", i)
        d = []
        for j in range(m):
            d.append(compute_distance_square(j, weights, x_train[i]))
#         print(np.array(weights).transpose()[416])
        minpos = d.index(min(d))
        print(min(d))
#         print(minpos)
        for j in range(minpos - R, minpos + R + 1):
            print(j)
            print(y_train[i])
            weights[:, j] = weights[:, j] + alpha * (x_train[i] - weights[:, j])

        print()
    alpha = alpha * decay_alpha

```

بعد از آموزش شبکه داریم:

```
input number:  998
30.70350815305091
559
9

input number:  999
59.94292556489089
559
8
```

```
In [7]: 1 print(weights)
```

```
[[0.56527061 0.26954774 0.75263499 ... 0.45780132 0.12272672 0.4058609 ]
 [0.76890253 0.05134861 0.5986841  ... 0.46488896 0.37454181 0.00995268]
 [0.89588719 0.82300959 0.35678014 ... 0.78405991 0.65298173 0.07482362]
 ...
 [0.72343505 0.32393345 0.02801468 ... 0.52497297 0.35437339 0.14095843]
 [0.08488874 0.48305394 0.97106854 ... 0.92086692 0.50767967 0.83175544]
 [0.44498613 0.15189644 0.4659118  ... 0.09484155 0.33644564 0.49500344]]
```

حال می‌توانیم با استفاده از شبکه آموزش دیده نود ها را انتخاب کنیم و برنده ها را نمایش دهیم:

همان‌طور که میبینیم فقط بخشی از نود ها برنده شده‌اند و بخش بزرگی از آن‌ها خالی هستند که نشان می‌دهد شبکه توانسته تا حدی آموزش ببیند
می‌دانیم که در SOM شبکه نمی‌تواند به طور ۱۰۰ درصد آموزش ببیند

حال می‌توانیم شعاع مجاورت را به ۱ تغییر دهیم و شبکه را دوباره آموزش دهیم و داریم:

حال شبکه را می‌خواهیم به صورت مربعی بچینیم می‌دانیم برای اینگه نود ها را در یک مربع ۲۹ در ۲۹ بچینیم می‌توانیم شعاع مجاورت را ۲۹ قرار دهیم که در این صورت می‌بینیم که نود ها مانند مربع قرار می گیرند

سؤال ۲)

در این سؤال می‌خواهیم با استفاده از الگوریتم MaxNet بین نود های ورودی بیشترین مقدار را پیدا کرده و آن‌ها را طبق ترتیبی Sort کنیم

۱) برای پیدا کردن بیشترین مقدار بین مقادیری که از یک عدد بزرگ‌تر هستند می‌توان ۲ کار انجام داد
یک کار آن است که اعداد را همان‌طور که هستند بگزاریم و شبکه را اجرا کنیم با این کار باز هم شبکه بیشترین مقدار را پیدا می‌کند ولی تعداد iteration بیشتری نسبت به حالت دوم طول می‌کشد تا جواب پیدا شود
کار دوم این است که چون می‌دانیم همه ی مقادیر از یک مقدار به‌خصوص بیشتر است می‌توانیم تمام مقادیر را از آن مقدار ثابت کم کنیم و سپس شبکه را اجرا کنیم

برای آموزش شبکه داریم:
ابتدا مقادیر ورودی اپسیلون و ماتریس وزن ها را تشکیل می‌دهیم

In [12]:
```
a = [1.2, 1.1, 1, 0.9, 0.95, 1.15]
e = 0.1
weights = np.empty(shape=(4,4))
weights.fill(1)
for i in range(4):
    for j in range(4):
        if i == j:
            weights[i][j] = 1
        else:
            weights[i][j] = -e
print(weights)
```

```
[[ 1.  -0.1 -0.1 -0.1]
 [-0.1  1.  -0.1 -0.1]
 [-0.1 -0.1  1.  -0.1]
 [-0.1 -0.1 -0.1  1. ]]
```

سپس با استفاده از اکتیویشن رلو می‌توانیم شبکه را آموزش دهیم:

In [9]:
```
def activation(x):
    if x >= 0:
        return x
    return 0
```

سپس با استفاده از الگوریتم MaxNet شبکه را آموزش می‌دهیم و بیشترین ورودی را پیدا می‌کنیم

In [13]:
```
a_old = np.array(a.copy())
a_new = []
count = 0
while True:
    print('Iteration {} - activations = {}'.format(count, [ '%.2f' % elem for elem in a_old ]))
    temp = np.sum(a_old)
    for i in range(len(a_old)):
        value = a_old[i] - e * temp + e * a_old[i]
        a_new.append(activation(value))
    a_old = a_new.copy()
    count += 1
    if np.sum(a_new) == max(a_new):
        break
    a_new = []

print('Iteration {} - activations = {}'.format(count, [ '%.2f' % elem for elem in a_new ]))


i=0
while a_new[i]==0:
    i=i+1
print ("Winning neuron : ", a[i])
```

و در خروجی داریم:

```
Iteration 0 - activations = ['1.20', '1.10', '1.00', '0.90', '0.95', '1.15']
Iteration 1 - activations = ['0.69', '0.58', '0.47', '0.36', '0.41', '0.63']
Iteration 2 - activations = ['0.44', '0.32', '0.20', '0.08', '0.14', '0.38']
Iteration 3 - activations = ['0.33', '0.20', '0.06', '0.00', '0.00', '0.26']
Iteration 4 - activations = ['0.28', '0.13', '0.00', '0.00', '0.00', '0.21']
Iteration 5 - activations = ['0.24', '0.08', '0.00', '0.00', '0.00', '0.16']
Iteration 6 - activations = ['0.22', '0.04', '0.00', '0.00', '0.00', '0.13']
Iteration 7 - activations = ['0.20', '0.01', '0.00', '0.00', '0.00', '0.10']
Iteration 8 - activations = ['0.19', '0.00', '0.00', '0.00', '0.00', '0.08']
Iteration 9 - activations = ['0.18', '0.00', '0.00', '0.00', '0.00', '0.06']
Iteration 10 - activations = ['0.18', '0.00', '0.00', '0.00', '0.00', '0.05']
Iteration 11 - activations = ['0.17', '0.00', '0.00', '0.00', '0.00', '0.03']
Iteration 12 - activations = ['0.17', '0.00', '0.00', '0.00', '0.00', '0.01']
Iteration 13 - activations = ['0.17', '0.00', '0.00', '0.00', '0.00', '0.00']
Winning neuron :  1.2
```

که همان‌طور که می‌بینیم شبکه توانسته بعد از ۱۳ دور بزرگترین عدد را پیدا کرده است و بزرگترین عدد ۱.۲ است.

برای قسمت بعد می‌خواهیم اعداد را به ترتیب صعودی مرتب کنیم برای اینکار اعداد را به ترتیب ای که ۰ می‌شوند ذخیره می کنیم:

Sort ascending with MaxNet ¶

In [14]:
```
a_old = np.array(a.copy())
a_new = []
count = 0
order_list = []
while True:
    for i in range(len(a_old)):
        if a_old[i] == 0 and not order_list.__contains__(a[i]):
            order_list.append(a[i])
    print('Iteration {} - activations = {}'.format(count, [ '%.2f' % elem for elem in a_old ]))
    temp = np.sum(a_old)
    for i in range(len(a_old)):
        value = a_old[i] - e * temp + e * a_old[i]
        a_new.append(activation(value))
    a_old = a_new.copy()
    count += 1
    if np.sum(a_new) == max(a_new):
        break
    a_new = []

print('Iteration {} - activations = {}'.format(count, [ '%.2f' % elem for elem in a_new ]))


i=0
while a_new[i]==0:
    i=i+1
print ("Winning neuron : ", a[i])
order_list.append(a[i])
print ("ascending order neuron : ", order_list)
```

```
Iteration 0 - activations = ['1.20', '1.10', '1.00', '0.90', '0.95', '1.15']
Iteration 1 - activations = ['0.69', '0.58', '0.47', '0.36', '0.41', '0.63']
Iteration 2 - activations = ['0.44', '0.32', '0.20', '0.08', '0.14', '0.38']
Iteration 3 - activations = ['0.33', '0.20', '0.06', '0.00', '0.00', '0.26']
Iteration 4 - activations = ['0.28', '0.13', '0.00', '0.00', '0.00', '0.21']
Iteration 5 - activations = ['0.24', '0.08', '0.00', '0.00', '0.00', '0.16']
Iteration 6 - activations = ['0.22', '0.04', '0.00', '0.00', '0.00', '0.13']
Iteration 7 - activations = ['0.20', '0.01', '0.00', '0.00', '0.00', '0.10']
Iteration 8 - activations = ['0.19', '0.00', '0.00', '0.00', '0.00', '0.08']
Iteration 9 - activations = ['0.18', '0.00', '0.00', '0.00', '0.00', '0.06']
Iteration 10 - activations = ['0.18', '0.00', '0.00', '0.00', '0.00', '0.05']
Iteration 11 - activations = ['0.17', '0.00', '0.00', '0.00', '0.00', '0.03']
Iteration 12 - activations = ['0.17', '0.00', '0.00', '0.00', '0.00', '0.01']
Iteration 13 - activations = ['0.17', '0.00', '0.00', '0.00', '0.00', '0.00']
Winning neuron :  1.2
ascending order neuron :  [0.9, 0.95, 1, 1.1, 1.2]
```

سپس اعداد را به ترتیب مرتب می‌کنیم و همان‌طور که میبینیم در خروجی میاریم

همین کار را برای مرتب کردن اعداد به ترتیب نزولی می‌توانیم استفاده کنیم

In [5]:

```
a_old = np.array(a.copy())
a_new = []
count = 0
order_list = []
while True:
    for i in range(len(a_old)):
        if a_old[i] == 0 and not order_list.__contains__(a[i]):
            order_list.insert(0, a[i])
    print('Iteration {} - activations = {}'.format(count, [ '%.2f' % elem for elem in a_old ]))
    temp = np.sum(a_old)
    for i in range(len(a_old)):
        value = a_old[i] - e * temp + e * a_old[i]
        a_new.append(activation(value))
    a_old = a_new.copy()
    count += 1
    if np.sum(a_new) == max(a_new):
        break
    a_new = []

print('Iteration {} - activations = {}'.format(count, [ '%.2f' % elem for elem in a_new ]))


i=0
while a_new[i]==0:
    i=i+1
print ("Winning neuron : ", a[i])
order_list.insert(0, a[i])
print ("ascending order neuron : ", order_list)
```

```
Iteration 0 - activations = ['1.20', '1.10', '1.00', '0.90', '0.95', '1.15']
Iteration 1 - activations = ['0.69', '0.58', '0.47', '0.36', '0.41', '0.63']
Iteration 2 - activations = ['0.44', '0.32', '0.20', '0.08', '0.14', '0.38']
Iteration 3 - activations = ['0.33', '0.20', '0.06', '0.00', '0.00', '0.26']
Iteration 4 - activations = ['0.28', '0.13', '0.00', '0.00', '0.00', '0.21']
Iteration 5 - activations = ['0.24', '0.08', '0.00', '0.00', '0.00', '0.16']
Iteration 6 - activations = ['0.22', '0.04', '0.00', '0.00', '0.00', '0.13']
Iteration 7 - activations = ['0.20', '0.01', '0.00', '0.00', '0.00', '0.10']
Iteration 8 - activations = ['0.19', '0.00', '0.00', '0.00', '0.00', '0.08']
Iteration 9 - activations = ['0.18', '0.00', '0.00', '0.00', '0.00', '0.06']
Iteration 10 - activations = ['0.18', '0.00', '0.00', '0.00', '0.00', '0.05']
Iteration 11 - activations = ['0.17', '0.00', '0.00', '0.00', '0.00', '0.03']
Iteration 12 - activations = ['0.17', '0.00', '0.00', '0.00', '0.00', '0.01']
Iteration 13 - activations = ['0.17', '0.00', '0.00', '0.00', '0.00', '0.00']
Winning neuron :  1.2
ascending order neuron :  [1.2, 1.1, 1, 0.95, 0.9]
```

سؤال ۳)

در این سؤال می‌خواهیم با استفاده از شبکه Mexican Hat
می‌خواهیم بیشترین مقدار را با استفاده از ۲ شبکه با شعاع های:
R1 = 1 R2 = 3 R1=0 R2=11
برای این کار ابتدا برای شعاع های همگرایی تابعی را می نویسیم تا برای شعاع همگرایی که به آن داده می‌شود نود های رقیب و نود های رفیق را پیدا کند.

In [44]:
```
def corrections(i, r1, r2, m):
    k = i-r1
    h = i-r2
    j = i+r1+1
    l = i+r2+1
    if k<0:
        k = 0
    if h<0:
        h = 0
    if l>m:
        l = m
    if j>m:
        j = m
    return k, h, j, l

def activation_fn(x):
    if x<0:
        return 0
    elif x>2:
        return 2
    else:
        return x
```

و سپس با تعیین C1 C2 و شعاع های همگرایی داریم:

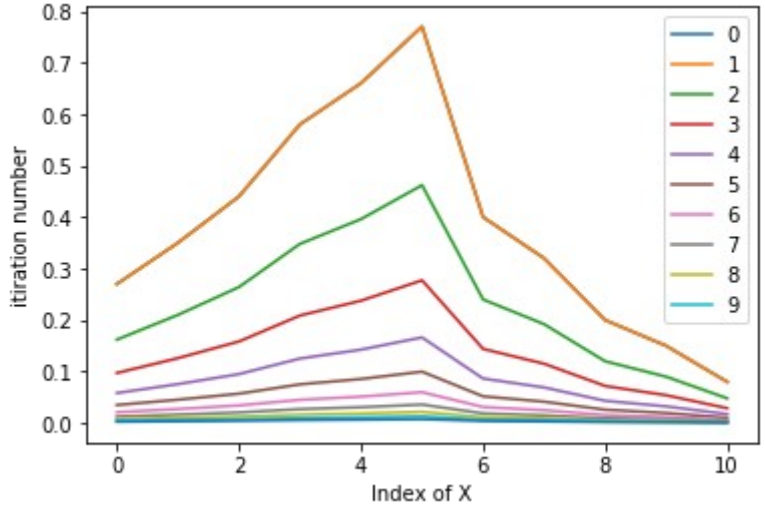
In [7]:
```
x = [0.27, 0.35, 0.44, 0.58, 0.66, 0.77, 0.4, 0.32, 0.20, 0.15, 0.08]
r1 = 0
r2 = 12
c1 = 0.6
c2 = -0.4
t_max = 10
```

حال شبکه را باری ۱۰ iteration اجرا می کنیم:

In [72]:
```
legend_plt = []
print ('t = 0')
for i in range(len(x)):
    print ('x{} = {}'.format(i, x[i]))
plt.plot(x)
for t in range(t_max): #step 3 step 7 & 8
    x_old = x.copy() #step 6
    plt.plot(x_old)
    print ('t = {}'.format(t+1))
    legend_plt.append(t)
    for i in range(len(x_old)):
        k, h, j, l = corrections(i, r1, r2, len(x_old))
        sum1 = sum((x_old[k:j]))
        left = sum(x_old[h:k])
        right = sum((x_old[j:l]))
        sum2 = left + right
        x[i] = activation_fn((c1*sum1 + c2*sum2)) # step 4 & 5
        print ('x{} = {}'.format(i, x[i]))
plt.legend(legend_plt)
plt.xlabel("Index of X")
plt.ylabel("itiration number")
plt.show()
```

می‌بینیم با شعاع رفیق ۰ و شعاع رقیب ۱۲ که شامل کل نود ها می‌شود در خروجی داریم:

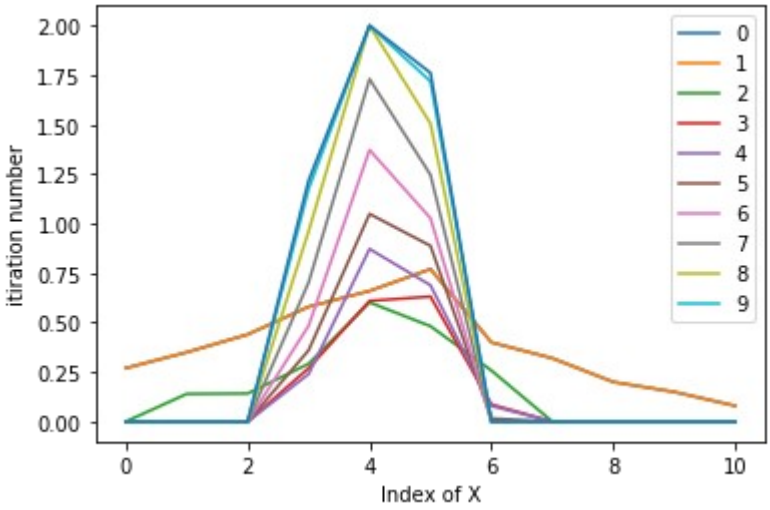
```
t = 10
x0 = 0.0016325867519999995
x1 = 0.0021163161599999994
x2 = 0.0026605117439999996
x3 = 0.0035070382079999984
x4 = 0.0039907676159999985
x5 = 0.004655895551999998
x6 = 0.0024186470399999993
x7 = 0.0019349176319999996
x8 = 0.0012093235199999997
x9 = 0.0009069926399999998
x10 = 0.0004837294079999999
```



می‌بینیم شبکه تقریباً مانند MaxNEt شده است و بیشترین مقدار را پیدا کرده است ک همان ایندکس ۵ بین اعداد است

حال اگر شبکه را با شعاع رفیق ۱ و رقیب ۳ اجرا کنیم داریم:

```
t = 10
x0 = 0
x1 = 0
x2 = 0
x3 = 1.2261985392639996
x4 = 2
x5 = 1.769526539264
x6 = 0
x7 = 0
x8 = 0
x9 = 0.0
x10 = 0.0
```



که می‌بینیم soft maximum را برای ایندکس ۴ پیدا کرده است
می‌دانیم بیشترین عدد ایندکس ۵ است ولی چون نود های رفیق
رو هم تأثیر گذاشته‌اند ایندکس ۴ بیشتر شده است

سؤال 4)

در این سؤال می‌خواهیم شیک شبکه hamming را آموزش دهیم می‌دانیم شبکه همینگ به تعداد بردار های پایه نود میانی (نود کلاستر) دارد و به تعداد المنت های ورودی نود ورودی دارد در اینجا چون ما ۳ بردار پایه داریم درواقع ۳ نود میانی (۳ نود برای کلاستر کردن) داریم و چون ورودی های ما ۶ عضو دارند ۶ نود ورودی داریم که هر کدام نشان دهنده یکی از اعداد ورودی است
و سپس به کمک یک شبکه Maxnet که ورودی آن همان ۳ بردار کلاستر ما هست شبیه ترین کلاستر را به ورودی تست مان پیدا می کنیم

برای آموزش شبکه ابتدا با استفاده از نود های پایه ماتریس وزن را که شامل نصف اعضای بردار های پایه است تشکیل می دهیم:

In [4]:
```
example_vectors = [[1, -1, 1, -1, 1, -1], [-1, 1, -1, 1, -1, 1], [1, 1, 1, 1, 1, 1]]
test_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],
                [1, 1, -1, 1, 1, 1], [1, 1, 1, -1, 1, 1]]
```

In [14]:
```
# initial weights
weights = []
for i in range(len(example_vectors)):
    w = [number / 2 for number in example_vectors[i]]
    weights.append(w)
weights = np.array(weights).transpose()
print(weights)
```

و ماتریس وزن ما به شکل زیر می شود:

```
[[ 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]]
```

سپس بردار بایاس را تشکیل می‌دهیم که به تعداد بردار های پایه ما عضو دارد و هر عضو همان نصف تعداد المنت های بردار است

سپس با تعیین اکتیویشن فانکشن رلو و اپسیلون داریم:

In [35]:
```
# initial biases
bias = [3, 3, 3]
y_in = np.zeros((3,1))
y = np.zeros((3,1))
```

In [36]:
```
def activation(x):
    if x >= 0:
        return x
    return 0

e = 0.1
```

حال برای شبکه داریم:

In [9]:
```
for i in range(len(test_vectors)):
    print("Test Vector Number: ", i)
    for j in range(len(example_vectors)):
        y_in[j] = bias[j] + np.sum(np.array(weights).transpose()[j] * test_vectors[i])

    y = copy.deepcopy(y_in)

    y_old = np.array(y.copy())
    y_new = []
    count = 0
    while True:
#         print('Iteration {} - activations = {}'.format(count, [ '%.2f' % elem for elem in y_old ]))
        temp = np.sum(y_old)
        for i in range(len(y_old)):
            value = y_old[i] - e * temp + e * y_old[i]
            y_new.append(activation(value))
        y_old = y_new.copy()
        count += 1
        if np.sum(y_new) == max(y_new):
            break
        y_new = []

#     print('Iteration {} - activations = {}'.format(count, [ '%.2f' % elem for elem in y_new ]))


    i=0
    while y_new[i]==0:
        i=i+1
    print ("Winning neuron : ", y[i])
    print ("Winning neuron number : ", i+1)
    print()

```

برای خروجی داریم:

```
Test Vector Number:  0
Winning neuron :  [4.]
Winning neuron number :  3

Test Vector Number:  1
Winning neuron :  [4.]
Winning neuron number :  1

Test Vector Number:  2
Winning neuron :  [4.]
Winning neuron number :  1

Test Vector Number:  3
Winning neuron :  [4.]
Winning neuron number :  2

Test Vector Number:  4
Winning neuron :  [6.]
Winning neuron number :  3

Test Vector Number:  5
Winning neuron :  [4.]
Winning neuron number :  1

Test Vector Number:  6
Winning neuron :  [4.]
Winning neuron number :  2

Test Vector Number:  7
Winning neuron :  [4.]
Winning neuron number :  3

Test Vector Number:  8
Winning neuron :  [5.]
Winning neuron number :  3

Test Vector Number:  9
Winning neuron :  [5.]
Winning neuron number :  3
```

همان‌طور که میبینیم نورون برنده برای هر بردار تست بین ۱ ۲ ۳ شده است که همان بردار های پایه ما هستند و نشان می‌دهد بردار تست به کدام یک از بردار های وایه نزدیک‌تر بوده است