



به نام خدا



دانشگاه تهران
دانشکده مهندسی برق و کامپیوتر

درس رمز ارز

تمرین سری دوم

نام و نام خانوادگی	حمید رضا کاشانی
شماره دانشجویی	810100441
تاریخ ارسال گزارش	1402\2\29

فهرست گزارش سوالات (لطفاً پس از تکمیل گزارش، این فهرست را به روز کنید.)

- سوال ۱ _ Pattern Association using Hebbian Learning Rule ۱
- سوال ۲ _ Auto-associative Net ۷
- سوال ۳ _ Discrete Hopfield Network ۱۲
- سوال ۴ _ Bidirectional Associative Memory ۲۰

سوال ۱_ Pattern Association using Hebbian Learning Rule

1) الگوریتم هب به طوری است که هر ورودی که باید حفظ شود رو در یه بعد قرار می دهد با یه مقدار ویژه ، فرض میکند که ورودی هایی که باید بر هم عمود باشند تا الگوریتم به خوبی کار کند ولی در صورت عمود نبودن هم با دقت خوبی الگوریتم کار می کند و هر چه تعداد ورودی ها برای حفظ کردن کمتر باشد الگوریتم بهتر کار می کند این الگوریتم به صورتی است وزن ها برابر حاصل جمع ضرب خروجی در ورودی آن وزن است و با یک بار اجرا شبکه آموزش داده می شود . این شبکه در مقابل نویز ورودی هم تا حدودی مقاوم است.

Pattern Association Algorithm by Hebbian Learning Rule

Step1: Initialize the weight matrix: $w_{ij}=0$ ($i=1..n, j=1..m$)

Step2: For each input-output pair ($\{s(p), t(p)\}$, $p=1,2,...,P$) do following steps: $\underline{s}^T=[s_1 \dots s_i \dots s_n]$ $\underline{t}^T=[t_1 \dots t_j \dots t_m]$

Step3: For $i=1..n$ $s_i \rightarrow x_i$

Step4: For $j=1..m$ $t_j \rightarrow y_j$

Step5: $w_{ij}^+ := w_{ij}^- + x_i y_j$

End

*It can be shown that the weight matrix is sum of the outer-products of input-output patterns:

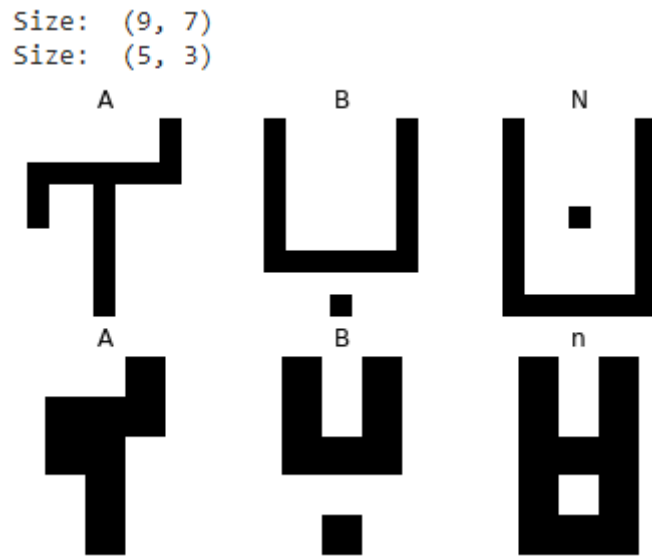
$$W = \sum_{p=1}^P s(p)t(p)^T$$

2) حال تابع های مورد نظر را برای تولید ماتریس شبکه می نویسیم.

ابتدا ورودی و خروجی ها را کنار هم می گذاریم:

```
S = np.hstack((A.reshape(-1, 1), B.reshape(-1, 1), N.reshape(-1, 1)))
T = np.hstack((a.reshape(-1, 1), b.reshape(-1, 1), n.reshape(-1, 1)))
```

ورودی و خروجی ها رو هم زیر هم رسم می کنیم:



سپس کلاس شبکه PAN با الگوریتم هب رو می نویسیم:

```
class PAN:
    def __init__(self, S, T):
        self.S = S
        self.T = T
        self.W = np.zeros((S.shape[0], T.shape[0]))

    def heb_train(self):
        S = self.S
        T = self.T
        for i in range(T.shape[1]):
            s = S[:, i].reshape((S.shape[0], 1))
            t = T[:, i].reshape((T.shape[0], 1))
            self.W += s @ t.T
        return self.W

    def f (self, X):
        for i in range(X.shape[0]):
            for j in range(X.shape[1]):
                X[i][j] = 1 if X[i][j] >= 0 else -1
        return X

    def out_put(self, sample):
        X = sample.reshape(-1, 1)
        return self.func(X.T @ self.W)
```

وزن ها و سایز وزن ها رو بعد یادگیری نمایش می دهیم

```
model = PAN(S,T)
```

خروجی:

3

```

[-1. 3. -3. -3. 1. -3. -3. -3. -1. 1. 1. 1. 1. -3. 1.]
[-1. 3. -3. -3. 1. -3. -3. -3. -1. 1. 1. 1. 1. -3. 1.]
[-3. 1. -1. -1. 3. -1. -1. -1. -3. -1. 3. -1. -1. -1. -1.]
[-1. 3. -3. -3. 1. -3. -3. -3. -1. 1. 1. 1. 1. -3. 1.]
[-1. 3. -3. -3. 1. -3. -3. -3. -1. 1. 1. 1. 1. -3. 1.]
[1. 1. -1. -1. -1. -1. -1. -1. 1. 3. -1. 3. 3. -1. 3.]
[1. 1. -1. -1. -1. -1. -1. -1. 1. 3. -1. 3. 3. -1. 3.]
[1. 1. -1. -1. -1. -1. -1. -1. 1. 3. -1. 3. 3. -1. 3.]
[1. 1. -1. -1. -1. -1. -1. -1. 1. 3. -1. 3. 3. -1. 3.]
[1. 1. -1. -1. -1. -1. -1. -1. 1. 3. -1. 3. 3. -1. 3.]
[1. -3. 3. 3. -1. 3. 3. 3. 1. -1. -1. -1. -1. 3. -1.]
[1. 1. -1. -1. -1. -1. -1. -1. 1. 3. -1. 3. 3. -1. 3.]
[1. 1. -1. -1. -1. -1. -1. -1. 1. 3. -1. 3. 3. -1. 3.]
[1. 1. -1. -1. -1. -1. -1. -1. 1. 3. -1. 3. 3. -1. 3.]
(945, (63, 15))

```

(3) حال شبکه را تست می کنیم:

تابع تست رو می نویسیم اگر خروجی یک بدهد یعنی کاملاً با مقدار اصلی خروجی برابر است:

```

def test_PAN (predA, A):
    return np.sum(predA.reshape(A.shape) == A).astype('int') / 15


```


خروجی برای هر سه ورودی برابر است با:

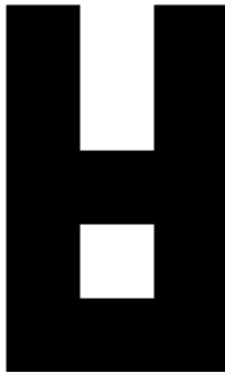
```

similarity precent for letter A: 1.000000
similarity precent for letter B: 1.000000
similarity precent for letter C: 1.000000

```

A_predicted


B_predicted


N_predicted


به طور کامل ورودی ها با آن که باید باشد یکی است.

(4)

(5) ابتدا تابعی رو برای ایجاد نویز در ورودی های درست می کنیم. سپس آخر برای 100 بار نویز

دادن خروجی را تست میکنیم و در آخر درصد درست تشخیص دادن خروجی را می آوریم:

```

def Noise_Net(input, percent):
    noisyOutput = input.copy()
    choosenIndices = np.random.choice(noisyOutput.size, int(percent *
noisyOutput.size / 100), replace=False)

```

```

for index in choosenIndices:
    if noisyOutput[index] == 1:
        noisyOutput[index] = -1
    else:
        noisyOutput[index] = 1

return noisyOutput

```

با 100 بار امتحان حاصل برابر:

```

khoroji sahih baray A ba 20 darsad noise 0.960000 darsad ast
khoroji sahih baray B ba 20 darsad noise 1.000000 darsad ast
khoroji sahih baray N ba 20 darsad noise 0.830000 darsad ast
khoroji sahih baray A ba 60 darsad noise 0.000000 darsad ast
khoroji sahih baray B ba 60 darsad noise 0.000000 darsad ast
khoroji sahih baray N ba 60 darsad noise 0.000000 darsad ast

```

(6) همین کار را با از بین بردن اطلاعات انجام می دهیم:

تابع از بین بردن اطلاعات

```

def loss_Net(input, percent):
    lossyOutput = input.copy()
    choosenIndices = np.random.choice(lossyOutput.size, int(percent *
lossyOutput.size / 100), replace=False)
    for index in choosenIndices:
        lossyOutput[index] = 0
    return lossyOutput

```

خروجی برای 100 بار خروجی گرفتن با از بین بردن اطلاعات:

```

khoroji sahih baray A ba 20 darsad noise 1.000000 darsad ast
khoroji sahih baray B ba 20 darsad noise 1.000000 darsad ast
khoroji sahih baray N ba 20 darsad noise 0.980000 darsad ast
khoroji sahih baray A ba 60 darsad noise 0.980000 darsad ast
khoroji sahih baray B ba 60 darsad noise 1.000000 darsad ast
khoroji sahih baray N ba 60 darsad noise 0.880000 darsad ast

```

(7) مشاهده میشود که شبکه در مقابل از دست رفتن اطلاعات بسیار مقاوم تر است نسبت به نویز

و تغییر علامت پیکسل و هر چه خروجی تعداد کمتری داشته باشد ابعاد آزاد و پوچ زیاد وجود

دارد و این ابعاد پوچ برای از بین بردن نویز و مقاومت در مقابل نویز کارا هستند یعنی هر چه

تعداد خروجی کمتر قدرت شبه در حذف اعوجاج و نویز و از بین رفتن اطلاعات بیشتر می شود.

سوال ۲ _ Auto-associative Net

برای تبدیل عکس ها به آرایه های بایپولار ابتدا عکس ها را آورده تبدیل به آرایه میکنیم معکوس میکنیم اعداد بزرگتر از 0 را به یک تبدیل میکنیم تا باینری شود سپس نوع داده را به int32 تبدیل می کنیم تا بتوانیم اعداد منفی هم بدهیم سپس صفر ها را به 1- تبدیل می کنیم:

```
from PIL import Image
import numpy as np
np_img=[]
for i in range(3):
    img = Image.open('/content/photo/Image_%i.png'%(i+1)).convert('L')

    np_img += [np.array(img)]
    np_img[i]= ~np_img[i] # invert B&W
    np_img[i][np_img[i] > 0] = 1
img1=np_img[0]
img2=np_img[1]
img3=np_img[2]
img1 = img1.astype('int32')
img2 = img2.astype('int32')
img3 = img3.astype('int32')
img1[img1==0] = -1
img2[img2==0] = -1
img3[img3==0] = -1
```

حال برای استفاده از کتابخانه های سوال قبل ورودی ها را به شکل ستون های کنار هم در می آوریم:

```
S = np.hstack((img1.reshape(1, 1), img2.reshape(1, 1), img3.reshape(-1, 1)))
```

سپس کلاس Auto-associative Net را پیاده سازی می کنیم.

```
class auto_associative_net:
    def __init__(self, S):
        self.S = S
        self.W = np.zeros((S.shape[0], S.shape[0]))

    def heb_train(self):
        S = self.S
        for i in range(S.shape[1]):
            s = S[:, i].reshape((S.shape[0], 1))
            self.W += s @ s.T
        return self.W

    def heb_train_modified(self):
```

```

S = self.S
for i in range(S.shape[1]):
    s = S[:, i].reshape((S.shape[0], 1))
    self.W += s @ s.T - np.eye(S.shape[0])
return self.W

def f (self, X):
    for i in range(X.shape[0]):
        for j in range(X.shape[1]):
            X[i][j] = 1 if X[i][j] >= 0 else -1
    return X

def out_put(self, sample):
    X = sample.reshape(-1, 1)
    return self.f(X.T @ self.W)

```

1) حال برای ورودی های داده شده شبکه را تشکیل داده و هر دو نوع یادگیری را برای آن انجام می دهیم:

```

model = auto_associative_net(S)
print(model.heb_train())
model1 = auto_associative_net(S)
print(model1.heb_train_modified())

```

که حاصل آن که ضرایب آموزش داده شده اند به صورت زیر اند:

```

[[ 3. -1. -3. ... -3. -3.  1.]
 [-1.  3.  1. ...  1.  1.  1.]
 [-3.  1.  3. ...  3.  3. -1.]
 ...
 [-3.  1.  3. ...  3.  3. -1.]
 [-3.  1.  3. ...  3.  3. -1.]
 [ 1.  1. -1. ... -1. -1.  3.]]

```

```

[[ 0. -1. -3. ... -3. -3.  1.]
 [-1.  0.  1. ...  1.  1.  1.]
 [-3.  1.  0. ...  3.  3. -1.]
 ...
 [-3.  1.  3. ...  0.  3. -1.]
 [-3.  1.  3. ...  3.  0. -1.]
 [ 1.  1. -1. ... -1. -1.  0.]]

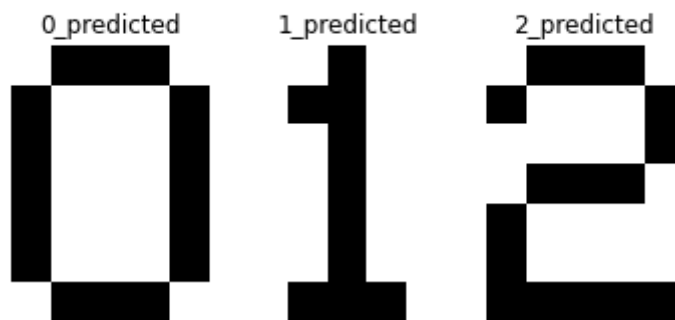
```

2) این الگوریتم این سه عکس را حفظ می کند و علاوه بر حفظ قابلیت تشخیص و حذف نویز ها واعوجاج ها را نیز دارد به این صورت است که ویژگی های هر عکس را در راستا برداری قرار می دهد که این بردار ها با هم عمود اند حال اگر اعوجاج و نویزی در ورودی باشد به علت هم راستا نبودن با بردار ورودی حذف می شود و عکس بدون اعوجاج در خروجی نمایش داده می شود .

برای هر دو مدل درستشده نتیجه ورودی بدون اعوجاج کاملاً صحیح است:

```
simA = test_NET(model.out_put(img1), img1)
simB = test_NET(model.out_put(img2), img2)
simC = test_NET(model.out_put(img3), img3)
print('similarity precent for letter A: %f'%simA)
print('similarity precent for letter B: %f'%simB)
print('similarity precent for letter C: %f'%simC)
plt.subplot(131)
plt.imshow(model.out_put(img1).reshape(img1.shape), cmap='binary')
plt.axis('off')
plt.title('0_predicted')
plt.subplot(132)
plt.imshow(model.out_put(img2).reshape(img2.shape), cmap='binary')
plt.axis('off')
plt.title('1_predicted')
plt.subplot(133)
plt.imshow(model.out_put(img3).reshape(img3.shape), cmap='binary')
plt.axis('off')
plt.title('2_predicted')
plt.show()
```

```
similarity precent for letter A: 1.000000
similarity precent for letter B: 1.000000
similarity precent for letter C: 1.000000
```



این الگوریتم برای ورودی بدون اعوجاج کاملاً درست کار میکند

3) تابع ایجاد نویز همان تابع سوال قبل است و ما هم برای هر دو مدل این تست ایجاد نویز را برای 100

سمپل انجام می دهیم:

کد آن به صورت زیر است:

```
def Noise_Net(input, percent):
    noisyOutput = input.copy()
    choosenIndices = np.random.choice(noisyOutput.size, int(percent *
noisyOutput.size / 100), replace=False)
    for index in choosenIndices:
```

```

if noisyOutput[index] == 1:
    noisyOutput[index] = -1
else:
    noisyOutput[index] = 1

return noisyOutput

```

حاصل برای Hebbian Learning Rule:

```

khoroji sahih baray 0 ba 20 darsad noise 0.940000 darsad ast
khoroji sahih baray 1 ba 20 darsad noise 1.000000 darsad ast
khoroji sahih baray 2 ba 20 darsad noise 0.920000 darsad ast
khoroji sahih baray 0 ba 80 darsad noise 0.000000 darsad ast
khoroji sahih baray 1 ba 80 darsad noise 0.000000 darsad ast
khoroji sahih baray 2 ba 80 darsad noise 0.000000 darsad ast

```

شبکه در مقابل 80 درصد نویز دوام نمی آورد

ولی برای 20 درصد نویز شبکه برای عدد 1 خوب کار می کند ولی برای عدد 0 و 2 خیر زیرا پیکسل بیشتری برای شناسایی آن مورد نیاز است

برای 0 و 2 تقریباً شبیه هم عمل می کند ولی مقداری برای 0 بهتر تشخیص می دهد

پس همه ی اعداد به یک میزان به نویز حساس نیستند اعدادی که برای شناسایی آنها به تعداد بیشتری بیت نیاز است در مقابل نویز حساس ترند

حاصل برای Modified Hebbian Learning Rule:

```

khoroji sahih baray 0 ba 20 darsad noise 0.840000 darsad ast
khoroji sahih baray 1 ba 20 darsad noise 0.960000 darsad ast
khoroji sahih baray 2 ba 20 darsad noise 0.910000 darsad ast
khoroji sahih baray 0 ba 80 darsad noise 0.000000 darsad ast
khoroji sahih baray 1 ba 80 darsad noise 0.000000 darsad ast
khoroji sahih baray 2 ba 80 darsad noise 0.000000 darsad ast

```

در الگوریتم modified شبکه برای 2 بهتر از 0 کار می کند

3) حال به جای تابع نویز از تابع loss استفاده می کنیم همان تابع سوال قبل

نتیجه برای Hebbian Learning Rule:

```

khoroji sahih baray 0 ba 20 darsad loss 1.000000 darsad ast
khoroji sahih baray 1 ba 20 darsad loss 1.000000 darsad ast
khoroji sahih baray 2 ba 20 darsad loss 1.000000 darsad ast
khoroji sahih baray 0 ba 80 darsad loss 0.810000 darsad ast
khoroji sahih baray 1 ba 80 darsad loss 0.950000 darsad ast
khoroji sahih baray 2 ba 80 darsad loss 0.760000 darsad ast

```

20 درصد از دست دادن همه قایل تشخیص اند ولی برای 80 درصد به ترتیب 1 بهترین و بعد 0 بهتر از 2 است اما فرق چندانی ندارند

نتیجه برای Modified Hebbian Learning Rule :

```
khoroji sahih baray 0 ba 80 darsad loss 1.000000 darsad ast  
khoroji sahih baray 1 ba 80 darsad loss 1.000000 darsad ast  
khoroji sahih baray 2 ba 80 darsad loss 1.000000 darsad ast  
khoroji sahih baray 0 ba 80 darsad loss 0.470000 darsad ast  
khoroji sahih baray 1 ba 80 darsad loss 0.540000 darsad ast  
khoroji sahih baray 2 ba 80 darsad loss 0.490000 darsad ast
```

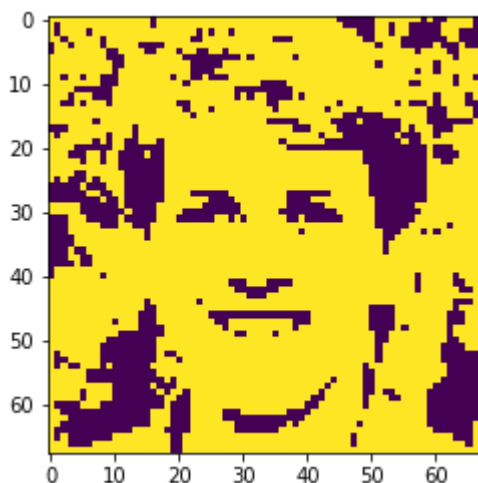
برای این مدل هم 1 بهتر از 0 و 2 و همینطور 2 بهتر از 0 است

سوال ۳ _ Discrete Hopfield Network

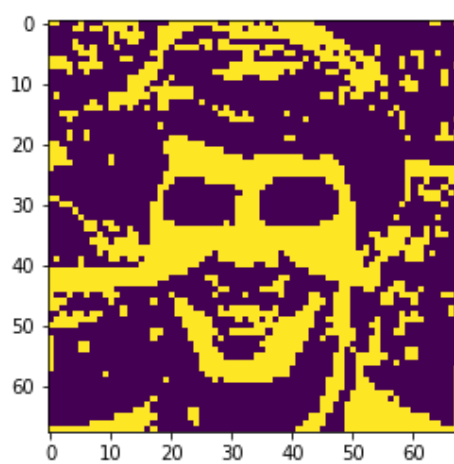
- (1) شبیه شبکه Recurrent Auto-Associative NET است که در آن ورودی به صورت بازگشتی از خروجی به دست می آید تا آن که شبکه به خروجی ثابتی میل کند و در آن تنها از ورودی در بار اول استفاده می شد و ورودی های بعدی شبکه خروجی های لحظه قبل آن بودند اما تفاوت آن با این شبکه Discrete Hopfield Net این است که یک هر بار تنها یکی از یونیت های شبکه به روز رسانی می شوند و تفاوت دوم این است که در هر بار بروز رسانی علاوه بر خروجی لحظه قبل ورودی اعمال شده به شبکه هم در الگوریتم بروز رسانی دخالت دارد. و هر یونیت به صورت اتفاقی از بین یونیت های بروز نشده انتخاب می شوند تا یک ایپاک تمام شود. این شبکه بازدهی بهتری نسبت به شبکه های قبلی دارد و در برابر نویز و اعوجاج مقاومت بیشتری از خود نشان می دهند.
- (2) عکس را آورده تبدیل باینری می کنیم و سپس سائز آن را به سائز استاندارد تغییر می دهیم
برای چندین ترشولد خروجی را می آوریم:

```
img = Image.open('/content/train.jpg').convert('L')
np_img = np.array(img)
print(np_img)
#np_img= ~np_img # invert B&W
#print(np_img)
teta=85
np_img[np_img<= teta] = 0
np_img[np_img > teta] = 1
import cv2
height = 68
width = 68
img_resized = cv2.resize(np_img, (width, height))
```

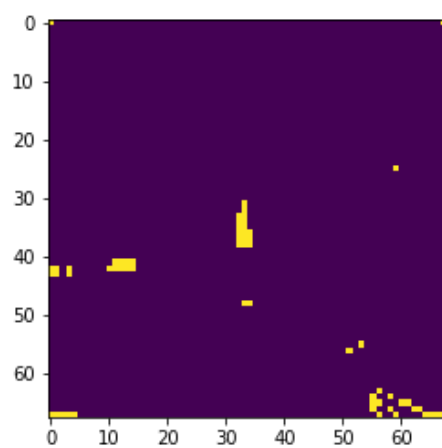
برای ترشولد 85:



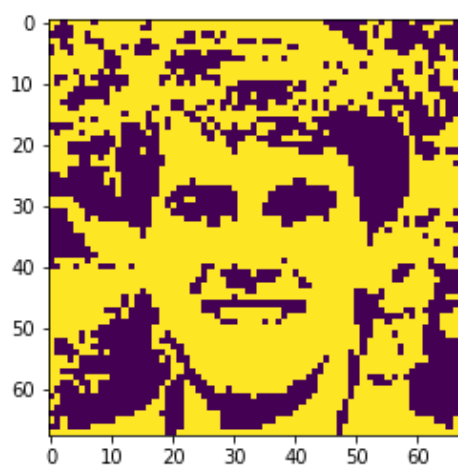
برای ترشولد 125:



برای ترشولد 185:



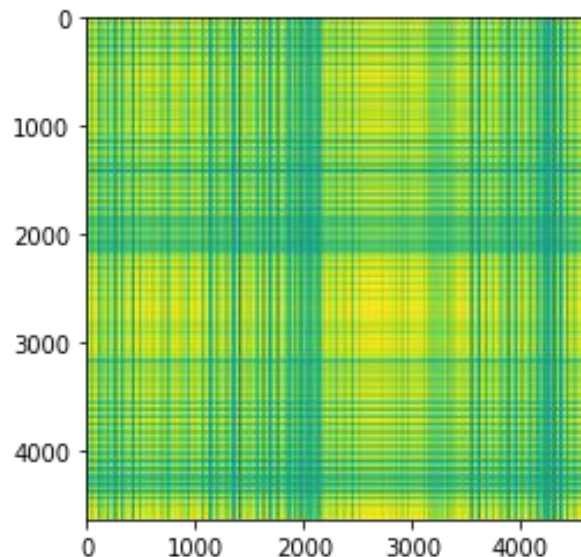
برای ترشولد 100 :



به نظر ترشولد 85 برای این مدل وضوح بیشتری دارد.

(3) ماتریس وزن ها از ضرب داخلی ورودی در خودش منهای ماتریس همانی حاصل می شود برای درک بهتر عکس ماتریس وزن ها را چاپ می شود

```
w = S.reshape(-1, 1) @ S.reshape(1, -1) - np.eye(S.size)
plt.imshow(w)
w
```



(4) الگوریتم bipolar داده شده در اسلاید ها را پیاده سازی می کنیم والگوریتم حوری است که نود بعدی برای یادگیری به صورت رندوم انتخاب می شود و خروجی را در هر 50 بار بروز رسانی می کشیم:

کد مربوط:

```
x = S.reshape(1, -1)
n=0
nn=0
y = np.copy(x)
out = None
def comparator( input, threshold):
    if input > threshold:
        return 1
    elif input < threshold:
        return -1
    else:
        return input
randiter = np.random.choice(np.arange(64*64), 64*64, replace=False)
axes=[]
```

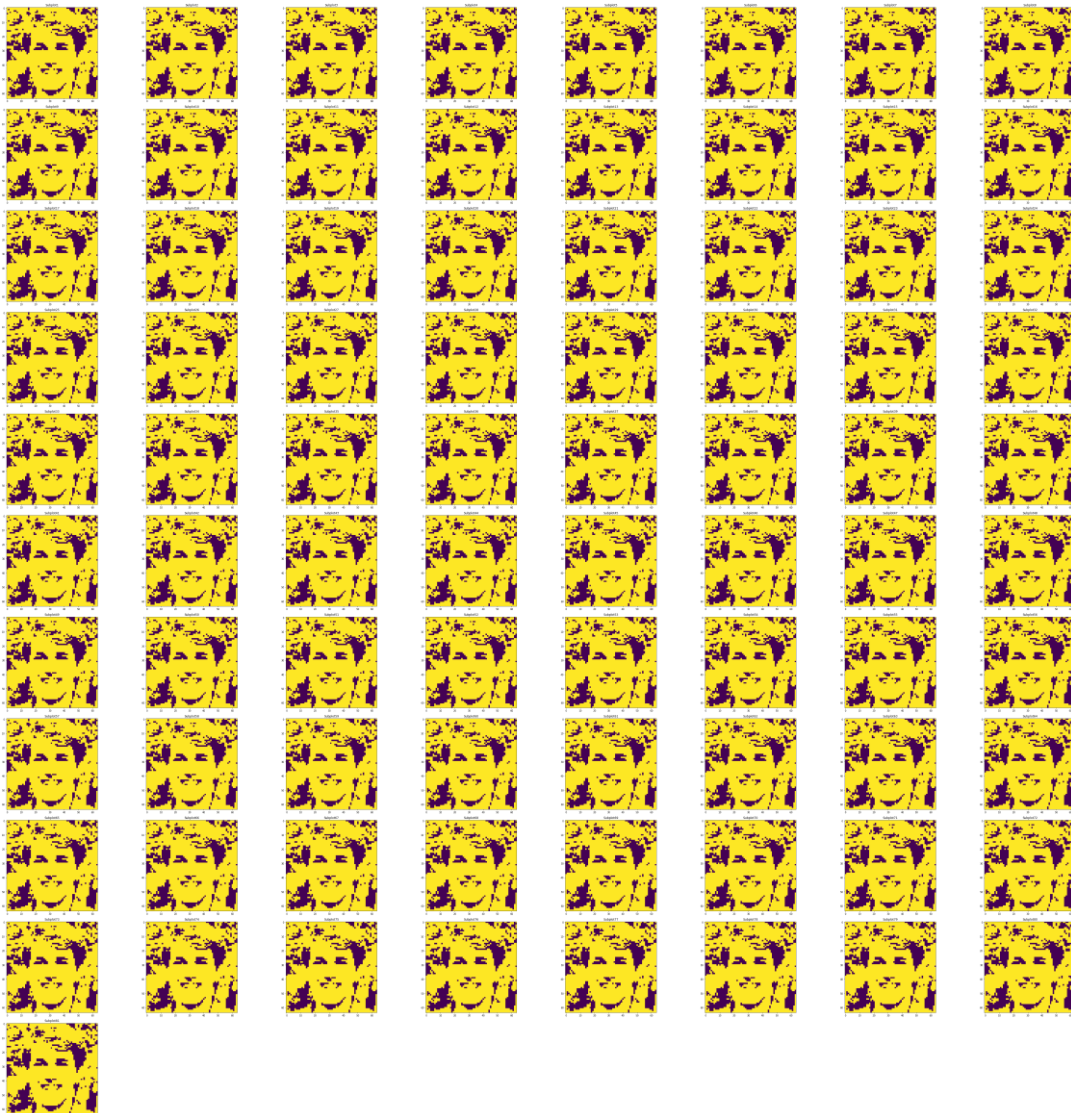


```

fig=plt.figure(figsize=(64, 64))
for i in randiter:
    y_in = x[0][i] + np.sum(y[0] * w[:, i])
    y[0][i] = comparator(y_in, 0)
    out = np.copy(y)
    n=n+1
    if n%50 == 0 :
        nn =nn+1
        axes.append( fig.add_subplot(11, 8, nn) )
        subplot_title=("Subplot"+str(nn))
        axes[-1].set_title(subplot_title)
        plt.imshow(out.reshape(64,64))
        #print(out.reshape(64,64))
fig.tight_layout()
plt.show()

```

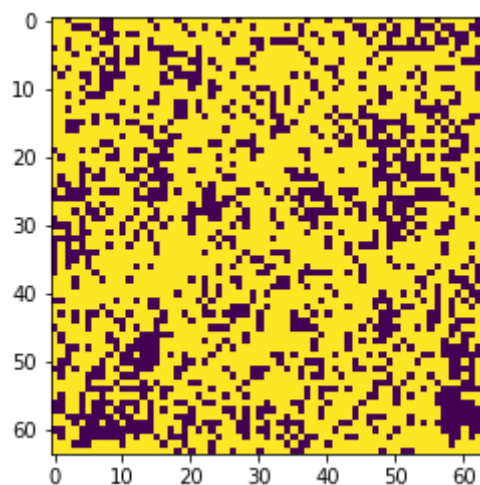
خروجی به صورت زیر است:



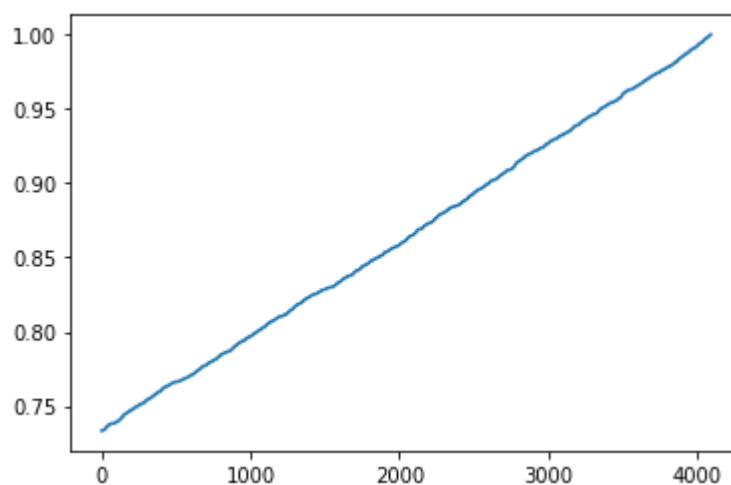
همان طور که مشاهده می شود خروجی از همان تکرار های اولیه به صورت صحیح کار می کند مطابق انتظار.

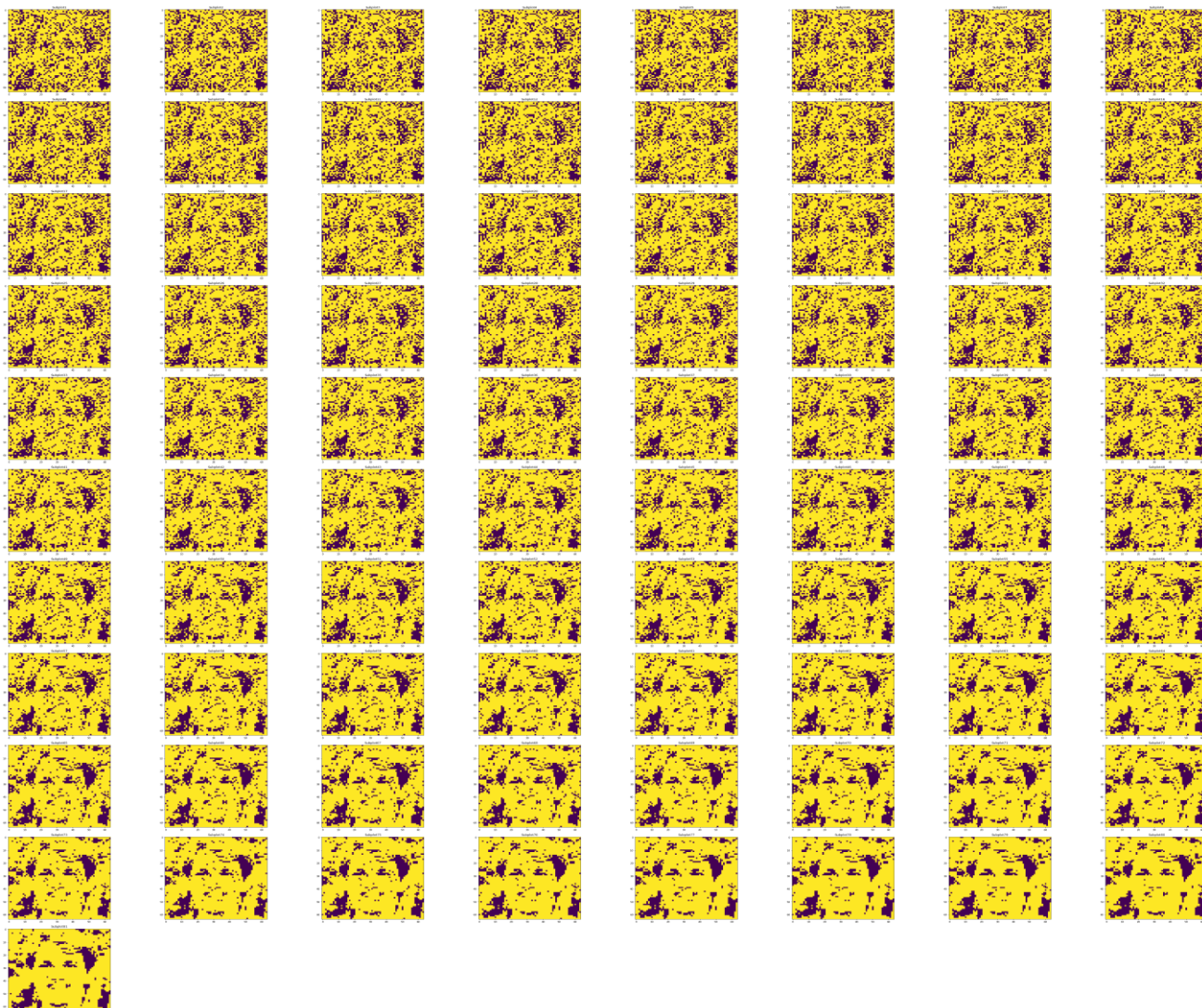
5) حال یک تابع مقایسه گر می سازیم تا نمودار آن را بر حسب تکرار ها رسم کنیم برای هر کدام از داده های نویزی تست.

عکس تست یک:

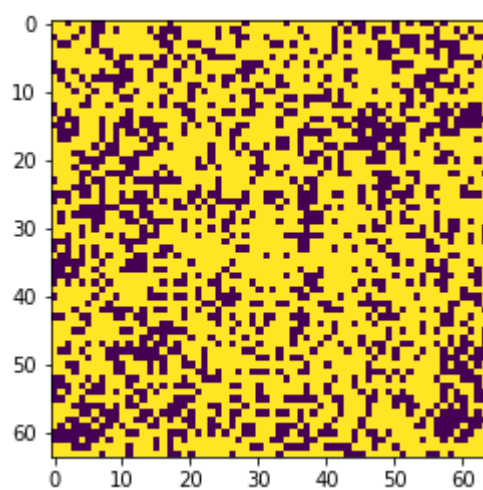


نتایج شبکه:

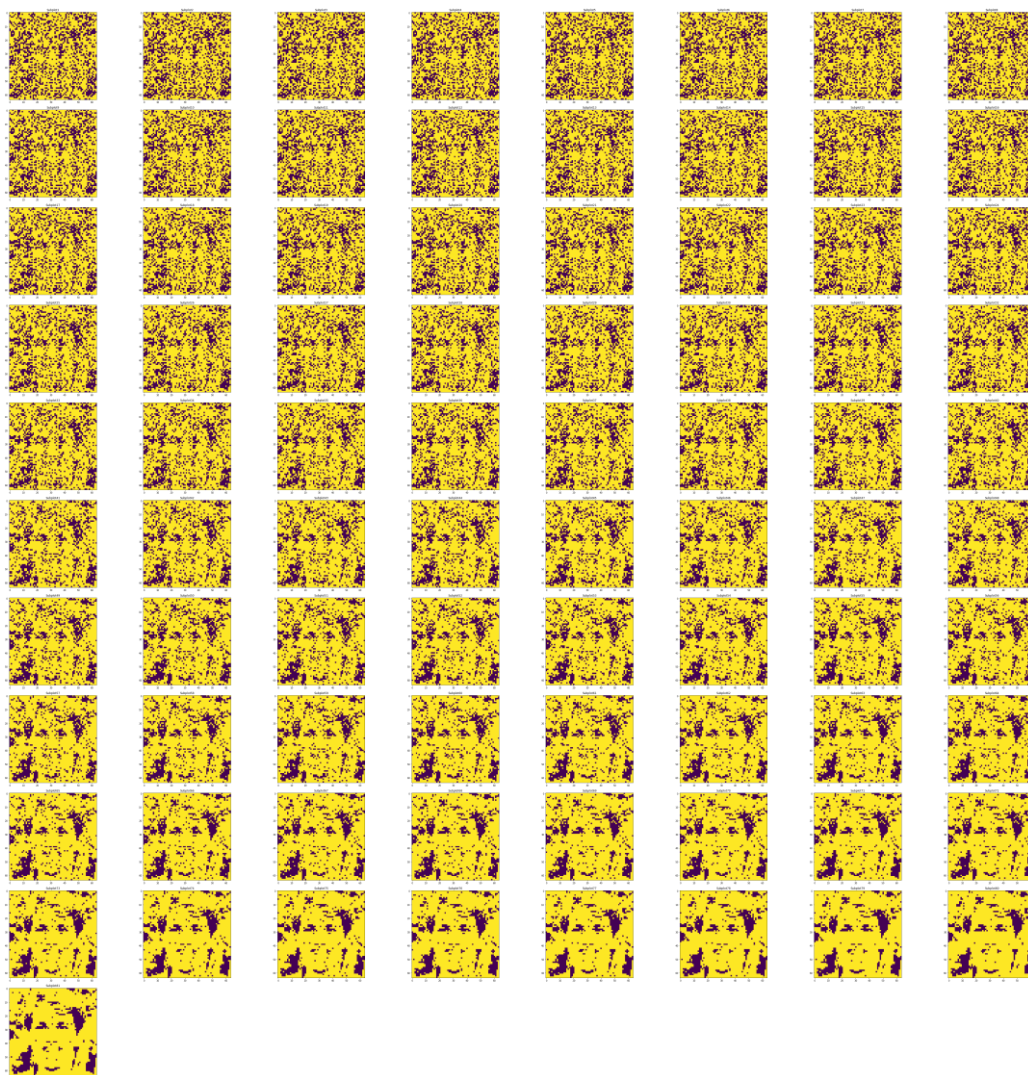
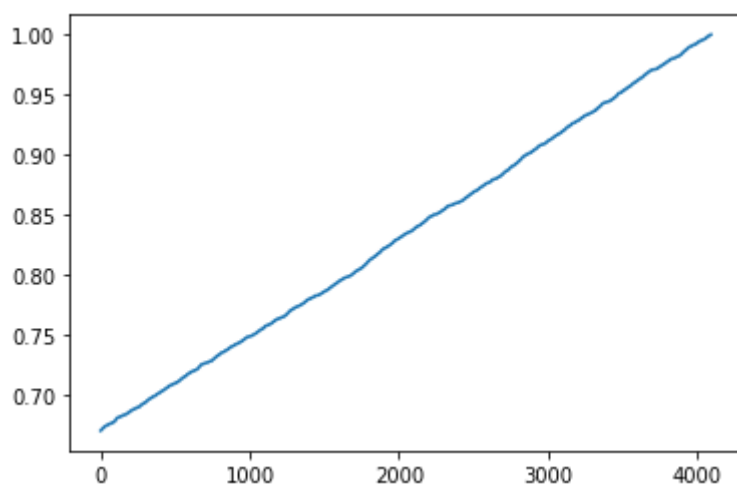




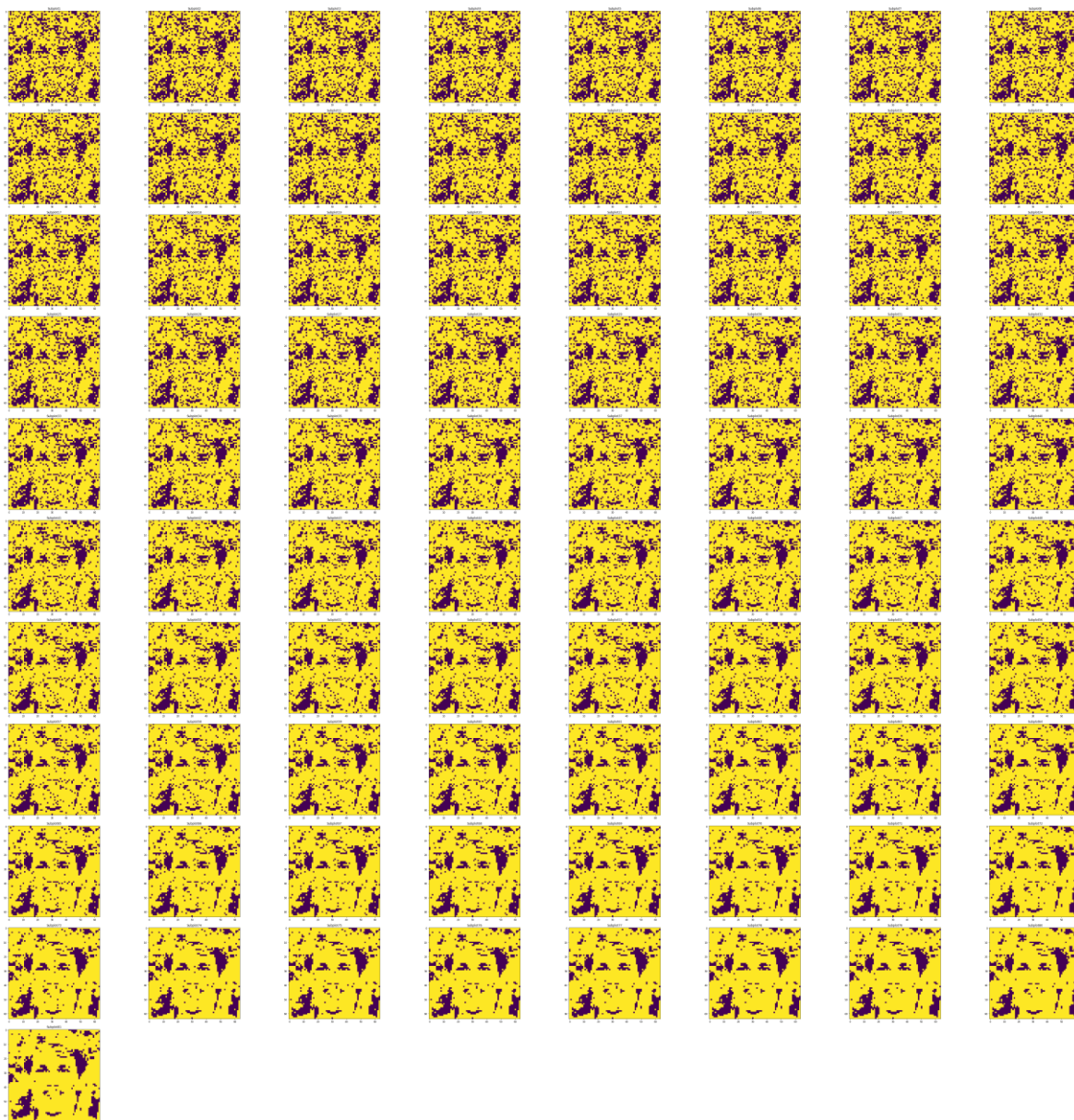
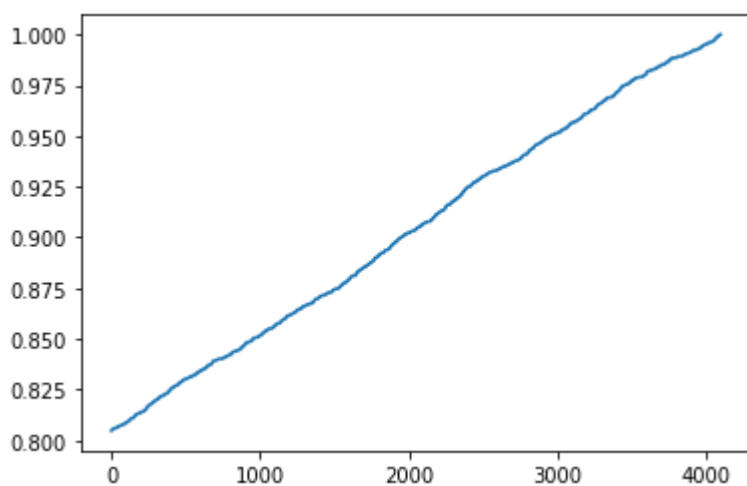
عکس تست 2:



نتایج:



برای تست سوم:



سوال ۴ Bidirectional Associative Memory

1) ما الگوریتم باپولار را انتخاب کردیم و ماتریس وزن ها مطابق فرمول پاور کد زدیم و به نتیجه زیر رسیدیم:

```
W = np.zeros((S.shape[0], T.shape[0]))
for i in range(3):
    W += S[:, i].reshape(-1, 1) @ T[:, i].reshape(-1, 1).T
print(W)
```

```
[[ 3. -3. -1. ... 1. -1. -1.]
 [-3.  3.  1. ... -1.  1.  1.]
 [-3.  3.  1. ... -1.  1.  1.]
 ...
 [-1.  1.  3. ... 1. -1. -1.]
 [ 1. -1.  1. ... 3.  1. -3.]
 [-1.  1. -1. ... -3. -1.  3.]]
```

2) ما شبکه را برای داده های ورودی و خروجی امتحان کردیم و هم برای ورودی ها و هم برای خروجی ها شبکه به درستی کار کرد:

ما ورودی را یا استفاده از دیکشنری حروف بایپولاری که درست کردیم وارد شبکه می کنیم و خروجی را با استفاده از یک پیرینت نمایش می دهیم و همین کار را برای خروجی انجام داده و با ورودی مقایسه می کنیم:

```
semat President baraye Clintonast
semat FirstLady baraye Hillaryast
semat Gentleman baraye Kenstarast
```

```
Clinton semat Presidentra darad
Hillary semat FirstLadyra darad
Kenstar semat Gentlemanra darad
```

3) حال با استفاده از تابع نویز ساز و مقایسه گر تشابه ساخته شده در سوالات قبل مقدار خروجی با نویز را با مقدار اصل آن مقایسه می کنیم و در صورت درستی کامل یکی به نمونه های درست ضافه میکنیم و

این کار را برای صد بار تکرار میکنیم تا متوجه شویم در چند درصد مواقع شبکه درست کار می کند
نتایج به دست آمده را پرینت کردیم:

برای 10 درصد نویز:

```
dar 100.000000 darsad mavaghe baraye 10 darsad khata  
hefz semat Clinton shabake dorost kar karde  
dar 100.000000 darsad mavaghe baraye 10 darsad khata  
hefz semat Hillary shabake dorost kar karde  
dar 100.000000 darsad mavaghe baraye 10 darsad khata  
hefz semat Kenstar shabake dorost kar karde  
dar 100.000000 darsad mavaghe baraye 10 darsad khata  
hefz name President shabake dorost kar karde  
dar 99.000000 darsad mavaghe baraye 10 darsad khata  
hefz name FirstLady shabake dorost kar karde  
dar 100.000000 darsad mavaghe baraye 10 darsad khata  
hefz name Gentleman shabake dorost kar karde
```

برای 20 درصد نویز:

```
dar 80.000000 darsad mavaghe baraye 20 darsad khata  
hefz semat Clinton shabake dorost kar karde  
dar 70.000000 darsad mavaghe baraye 20 darsad khata  
hefz semat Hillary shabake dorost kar karde  
dar 55.000000 darsad mavaghe baraye 20 darsad khata  
hefz semat Kenstar shabake dorost kar karde  
dar 69.000000 darsad mavaghe baraye 20 darsad khata  
hefz name President shabake dorost kar karde  
dar 66.000000 darsad mavaghe baraye 20 darsad khata  
hefz name FirstLady shabake dorost kar karde  
dar 49.000000 darsad mavaghe baraye 20 darsad khata  
hefz name Gentleman shabake dorost kar karde
```

3) حال شخصیت ها را یکی اضافه می کنیم و حفظیات شبکه را از 6 تا به 8 تا افزایش می دهیم
ورودی و خروجی ها را جدید کرده و دوباره وزن ها را به دست آورده و شبکه را امتحان می کنیم
که با خطا مواجه می شویم که به این معنی است که شبکه نتوانسته شخصیت بعدی را به خوبی
حفظ کند و برای درک بهتر کتر گرد شبکه آن را با نویز امتحان می کنیم و گزارش می دهیم ،
علت حفظ نشدن هم گنجایش پایین شبکه است و شبکه نمی تواند ویژگی ها را به طوری پیدا
کند که بر هم عمود شوند برای بیش از سه شخصیت این ضعف مشخص می شود:

dar 73.000000 darsad mavaghe baraye 10 darsad khata
hefz semat Lewisky shabake dorost kar karde
dar 37.000000 darsad mavaghe baraye 10 darsad khata
hefz name SweetGirl shabake dorost kar karde
dar 57.000000 darsad mavaghe baraye 20 darsad khata
hefz semat Lewisky shabake dorost kar karde
dar 35.000000 darsad mavaghe baraye 20 darsad khata
hefz name President shabake dorost kar karde