به نام خدا

دانشگاه تهران

دانشکده مهندسی برق و کامپیوتر

شبکه های عصبی و یادگیری عمیق

تمرین سری سوم

نام و نام خانوادگی	**محمدعلی شاکردرگاه**
شماره دانشجویی	810196487
تاریخ ارسال گزارش	1400/03/09

فهرست گزارش سوالات

سوال 1 – شناسایی حروف با استفاده از روش هب

در این سوال، انتظار میرود شبکه ای طراحی شود که به عنوان ورودی ماتریس 7*9 حروف الفبای A و B و C را بگیرد و بتواند در نهایت خروجی متناظر با هر کدام از این حروف الفبا را در ابعاد 3*5 را تولید کند. در این بخش از روش هب Non-iterative استفاده خواهد شد.

عملیات آموزش طبق شکل 1-1 انجام میشد:

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: $s^T=[s_1 \dots s_i \dots s_n]$ $t^T=[t_1 \dots t_j \dots t_m]$

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

شکل 1-1 الگوریتم یادگیری هب

پس از آموزش این شبکه عصبی:

- ورودی X ماتریس 7*9 از 1 و 1- به شکل حروف هستند که reshape میشود.
- از تابع sign به عنوان Activation function استفاده شده است.
- خروجی $Y^T = f(X^T.W)$ است که بعد از reshape ماتریس 3*5 از 1و 1- به شکل حروف میشود که با استفاده از کتابخانه matplotlib نشان داده خواهد شد.

الف) توانایی شبکه در تبدیل ورودی به خروجی مطلوب:

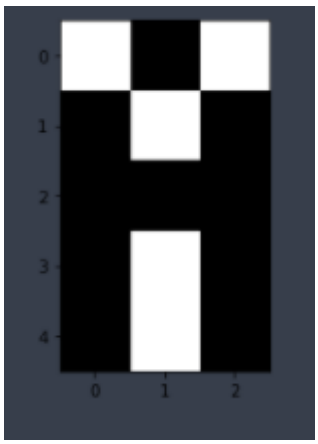
پس از یادگیری شبکه عصبی و دادن ورودی های مشخصه، مشاهده میشود که **"بله"، شبکه میتواند تمامی ورودی های مشخصه را به خروجی مطلوب برساند.**

1- برای ورودی A با شکل ماتریسی شکل 1-2

```
A_s = np.array([   [-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,  1, 1,  1]])
```

شکل 1-2 ورودی A

خروجی مطابق شکل 1-3 را به همراه داشته است



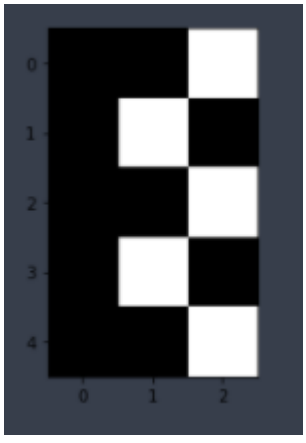
شکل 1-3 خروجی متناظر A

2- برای ورودی B با شکل ماتریسی شکل 1-4

```
B_s = np.array([  [ 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,  1,  1, 1]])
```

شکل 1-4 ورودی B

خروجی مطابق شکل 1-5 را به همراه داشته است



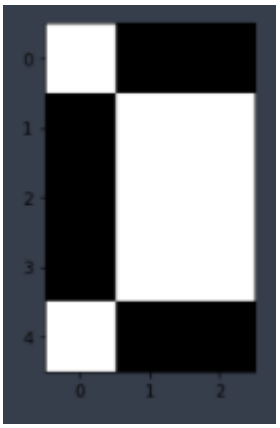
شکل 1-5خروجی متناظر B

3- برای ورودی C با شکل ماتریسی شکل 1-6

```
C_s = np.array([   [-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,  1,  1,-1]])
```

شکل 1-6ورودی C

خروجی مطابق شکل 1-7 را به همراه داشته است



شکل 1-7 خروجی متناظر C

ب) توانایی شبکه در تبدیل ورودی به خروجی مطلوب:

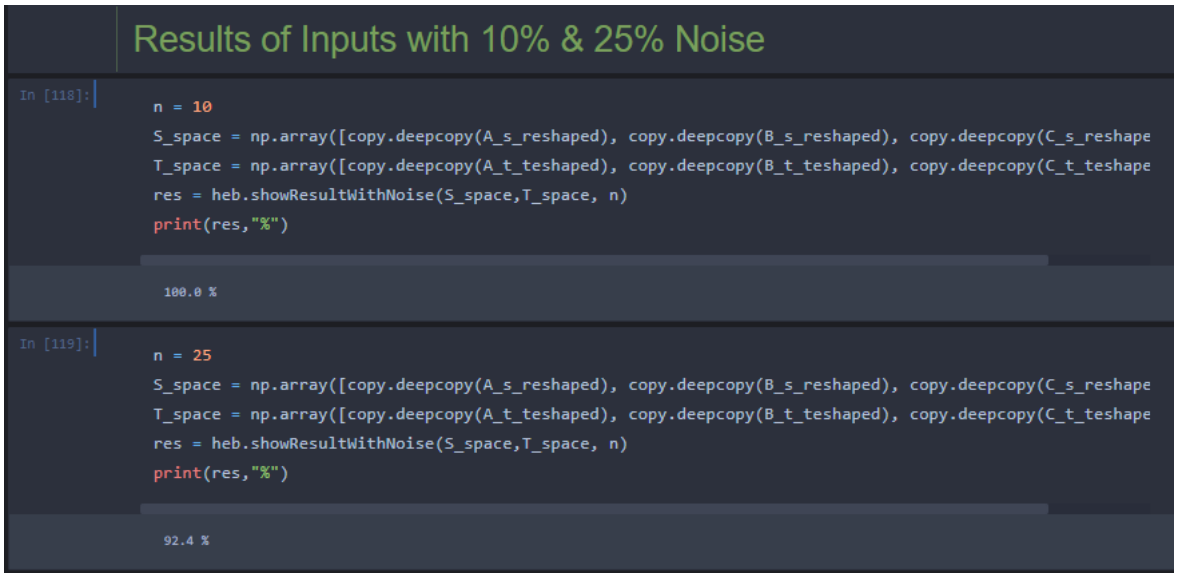
برای این قسمت، دو حالت کلی در نظر گرفته میشود،

1- تست داده ها به همراه 10درصد و 25درصد نویز
2- تست داده ها بدون اطلاعات 10درصد و 25درصد از آن ها (که اطلاعات آن ها از بین میرود یا به نوعی درایه 1 یا 1- تبدیل به 0 میشود)

برای هر کدام از این دو حالت، در 1000 ایپاک، تست صورت میپذیرد و درصد دقت الگوریتم سنجیده میشود .

1- تست داده ها به همراه 10درصد و 25درصد نویز:

نتایج در تصویر 1-7 قابل مشاهده است:



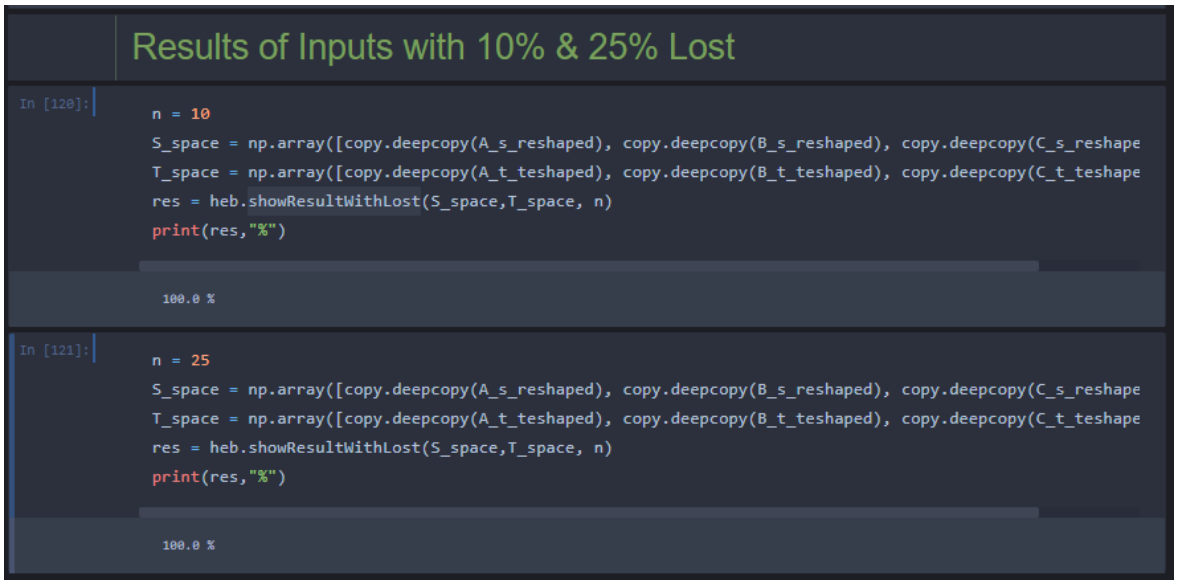
شکل 1-7 نتایج تست داده با 10 و 25% نویز

همانطور که قابل مشاهده است:

- با 10% نویز بر روی ورودی، حاصل دقت، از تست بر روی هر سه کلاس A و B و C مشخص میشود که الگوریتم توانسته است در 100% مواقع (1000 بار از 1000 بار) عملکرد درستی از خود نشان دهد و پاسخ درست را برگرداند.
- با 25% نویز بر روی ورودی، حاصل دقت، از تست بر روی هر سه کلاس A و B و C مشخص میشود که الگوریتم توانسته است در 92.4% مواقع (924 بار از 1000 بار) عملکرد درستی از خود نشان دهد و پاسخ درست را برگرداند.

2- تست داده ها بدون اطلاعات 10درصد و 25درصد از آن ها:

نتایج در تصویر 1-8 قابل مشاهده است:



شکل 1-8 نتایج تست داده بدون اطلاعات 10 و 25% آن ها

همانطور که قابل مشاهده است:

- با حذف اطلاعات 10% از داده های ورودی، حاصل دقت، از تست بر روی هر سه کلاس A و B و C مشخص میشود که الگوریتم توانسته است در 100% مواقع (1000 بار از 1000 بار) عملکرد درستی از خود نشان دهد و پاسخ درست را برگرداند.
- با حذف اطلاعات 25% از داده های ورودی، حاصل دقت، از تست بر روی هر سه کلاس A و B و C مشخص میشود که الگوریتم توانسته است در 100% مواقع (1000 بار از 1000 بار) عملکرد درستی از خود نشان دهد و پاسخ درست را برگرداند.

قابل شهود است که فقط در حالتی که 25% نویز روی داده هایمان قرار گرفته بودو اطلاعات 1 را به 1- و برعکس تبدیل کرده بود، نتوانستیم تماما به خروجی مطلوب برسیم و فقط 924 مورد از 1000 بار این مطلوب برآورده شد در حالی که با 25% Missing data که به جای 1 و 1- عدد صفر قرار داده شده بود، 100% دقت داشتیم (1000 مورد از 1000 بار).

نتیجه گیری جانبی : عملکرد شبکه عصبی در برابر حذف درصدی از داده ها، بهتر از زمانی بود که روی داده ها به همان درصد نویز قرار گرفته بود.

سوال ۲ – شبکه خود انجمنی

در این بخش مقصود آن است که یک شبکه Auto-Associative ساخته شود و در حالت های مختلف محیط نویزی مورد تست قرار گیرد و صحت شبکه در تولید خروجی درست سنجیده شود و عملکرد آن به بررسی گذاشته شود.

الف) دقت شبکه عصبی، میانگین تشخیص درست و انحراف معیار آن ها

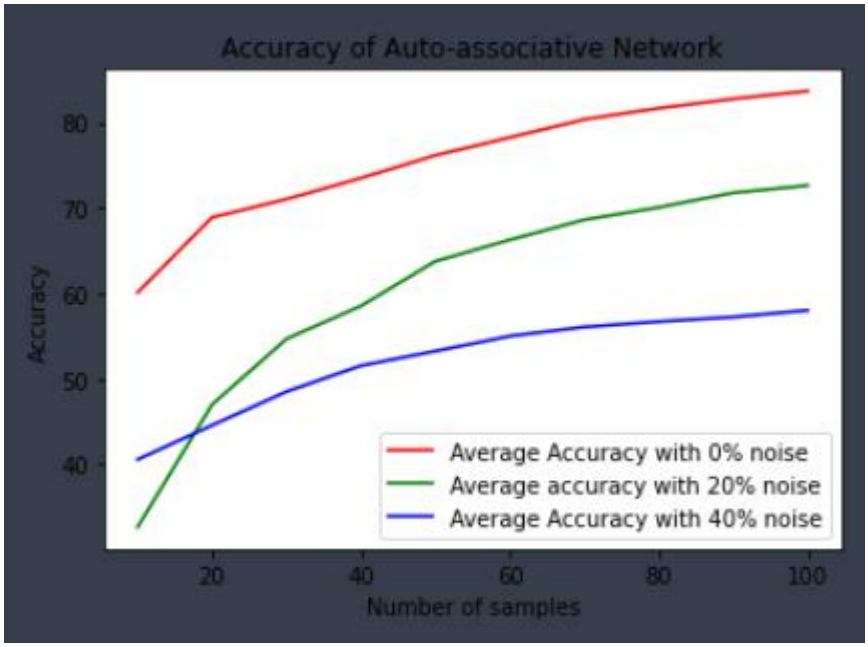
در این شبکه، فضای ورودی و فضای مطلوب خروجی یکی هستند. حال سعی میشود با تعداد 10 و 20 و 30 و 40 و 50 و 60 و 70 و 80 و 90 و100 Sample از ورودی های 100 تایی رندوم که دیتای 1 یا 1- (Bipolar) را در خود ذخیره کرده اند را با قاعده هب تعمیم یافته آموزش دهیم که برای وزن از رابطه زیر استفاده میکنیم:

$$W = \sum_{p} x_p y_p - PI$$

و سپس با 0 درصد و 20 درصد و 40 درصد Error روی این محموعه داده، با 30 بار آزمایش دقت شبکه عصبی خود را به آزمون بگذاریم.

1-2 : دقت

در شکل 2-1 نمودار دقت شبکه عصبی برای داده ها آورده شده است:



تصویر 2-1 نمودار دقت شبکه عصبی

همچنین مقدار مقیق دقت ها نیز در تصویر 2-2 قابل مشاهده است:

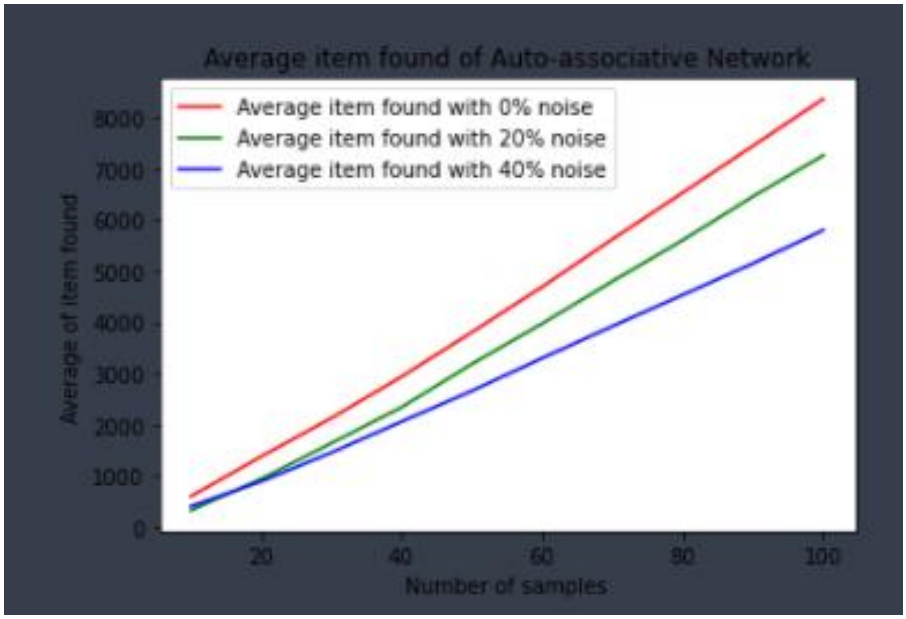
```
print("Accuracy with 0% noise:",accuracy_E0)
print("Accuracy with 20% noise:",accuracy_E20)
print("Accuracy with 40% noise:",accuracy_E40)
```

```
Accuracy with 0% noise: [60.099999999999994, 68.89999999999999, 71.03333333333333, 73.5, 76.16000000000001, 78.25, 80.3714285714285
7, 81.675, 82.77777777777777, 83.67999999999999]
Accuracy with 20% noise: [32.663333333333334, 46.99166666666667, 54.681111111111115, 58.53583333333333, 63.77, 66.27388888888889, 6
8.62238095238095, 70.07666666666667, 71.77703703703703, 72.637]
Accuracy with 40% noise: [40.55666666666667, 44.56666666666667, 48.507777777777775, 51.545833333333334, 53.24399999999999, 55.003333
33333332, 56.069047619047616, 56.705416666666665, 57.237037037037034, 58.013000000000005]
```

تصویر 2-2 مقدار دقیق دقت شبکه عصبی

2-2 : میانگین تشخیص درست تعداد داده bipolar توسط شبکه عصبی

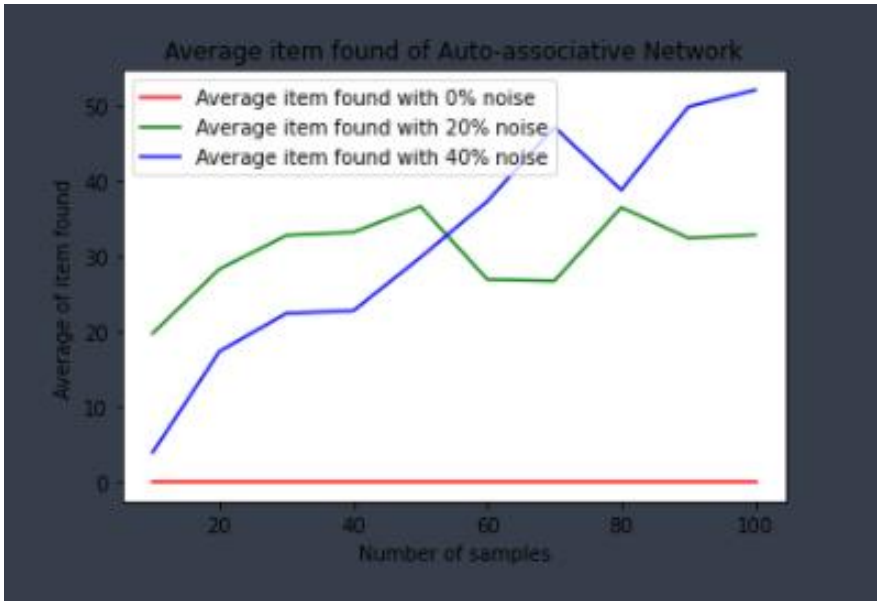
در تصویر 2-3 میتوان میانگین تعداد تشخیص توسط شبکه عصبی را مشاهده نمود.



تصویر 2-3 میتوان تعداد تشخیص توسط شبکه عصبی

2-3 : انحراف معیار تشخیص درست تعداد داده bipolar توسط شبکه عصبی

در تصویر 2-4 میتوان انحراف معیار تعداد تشخیص توسط شبکه عصبی را مشاهده نمود.



تصویر 2-4 میتوان انحراف معیار تعداد تشخیص توسط شبکه عصبی

ب) توضیح نتایج بند الف

2-1: دقت

مشاهده میشود که هنگامی که 0% خطا به داده ها اضافه میکنیم بهترین عملکرد را دارد که کاملا هم امری منطقی میباشد. برای هنگامی که 10 Sample ورودی داریم، عملکرد شبکه برای حالت 40% نویز از 20% نویز بهتر بوده اما برای تمامی تعداد Sample های دیگر، عملکرد شبکه در حضور 20% نویز بهتر از 40% نویز بوده است.

همچنین مشاهده میشود که با افزایش تعداد Sample ها، دقت میز افزایش پیدا کرده است

2-2 : میانگین تشخیص درست تعداد داده bipolar توسط شبکه عصبی

همانند قبل، مشاهده میشود که هنگامی که 0% خطا به داده ها اضافه میکنیم بهترین عملکرد را دارد. برای هنگامی که 10 Sample ورودی داریم، عملکرد شبکه برای حالت 40% نویز از 20% نویز بهتر بوده اما برای تمامی تعداد Sample های دیگر، عملکرد شبکه در حضور 20% نویز بهتر از 40% نویز بوده است.

همچنین مشاهده میشود که با افزایش تعداد Sample ها، میانگین تعداد تشخیص توسط شبکه عصبی میز افزایش پیدا کرده است

3-2 : انحراف معیار تشخیص درست تعداد داده bipolar توسط شبکه عصبی

برای 0% نویز، مشخصا تغییری در داده های ما ایجاد نمیشود، پس در 30 بار آزمایش برای هر کدام از Data set ها، تغییری در جواب مشاهده نمیشود و اعداد با میانگین خود یکی میشوند و Standard deviation برابر با صفر میشود.

ج) ظرفیت شبکه با در دست داشتن N و روابط

ابتدا در شکب 2-5، ظرفیت شبکه را بررسی کنیم

Capacity of one layer MNN in restoring input patterns

For n-dimensional orthogonal input patterns

Weight Matrix	Hetro-Associative	Auto-Associative Bipolar patterns
Hebbian Matrix	"Capacity =n"	"Capacity =n"
Modified Hebbian Matrix	-	"Capacity =n-1"

شکل 2-5 ظرفیت شبکه ها

در شبکه Auto-associative که در این بخش ما از نسخه Modified Hebbian Matrix استفاده کردیم، برای ورودی nبعدی عمود بر هم، ظرفیت n-1 (به دلیل آنکه یکی از مقادیر ویژه ماتریس هبین را برابر با صفر میکند) دارد.در اینجا دو فاکتور در حقیقت بررسی میشود، در اینجا ظرفیت ما برابر با 99 میباشد، که میبایست رعایت شود وگرنه عنصر دقت را از دست میدهیم، در اینجا به دلیل آنکه زیاد از آن عبور نکردیم، دقت مناسبی گرفتیم. عامل دیگر هم رابطه ورودی ها میباشد، این جدول برای داده های "عمود بر هم" میباشد که تاثیر ایجاد میکند. ما برای Generate کردن داده ها، به صورت رندوم عمل کردیم و این ممکن است باعث شود که داده ها بر همدیگر عمود نباشند. همانطور که مشاهده میشود، به دلیل عدم عمود بودن داده ها بر هم، به نهایت دقتی که رسیده ایم، 83.679% بوده است.

سوال 3 – شبکه هاپفیلد

در این بخش به بررسی شبکه Hopfield که یک شبکه Auto-associative میباشد پرداخته میشود.

در این مسئله دو ماتریس 8*8 که نماینده اعداد 0 و 1 میباشند به عنوان ورودی شبکه برای یادگیری آن انتخاب شده اند، بعد از آموزش این شبکه، 30% نویز روی این دو ورودی گذاشته میشود و عملکرد شبکه مورد بررسی قرار گرفته میشود:

الف) توانایی شبکه در تبدیل ورودی به خروجی مطلوب:

پس از یادگیری شبکه عصبی و تشکیل بردار وزن به شکل تصویر زیر:

To store a set of bipolar patterns $\mathbf{s}(p)$, $p = 1, \ldots, P$, where

$$\mathbf{s}(p) = (s_1(p), \ldots, s_i(p), \ldots, s_n(p)),$$

the weight matrix $\mathbf{W} = \{w_{ij}\}$ is given by

$$w_{ij} = \sum_p s_i(p) s_j(p) \qquad \text{for } i \neq j$$

and

$$w_{ii} = 0.$$

تصویر 3-1 یادگیری شبکه Hopfield

و دادن ورودی های مشخصه، مشاهده میشود که، **شبکه توانست تمامی ورودی های مشخصه را به خروجی مطلوب برساند.**

برای ورودی 0 با شکل ماتریسی شکل 3-2

```
zero_s = np.array( [[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, 1, 1, 1, 1]])
```

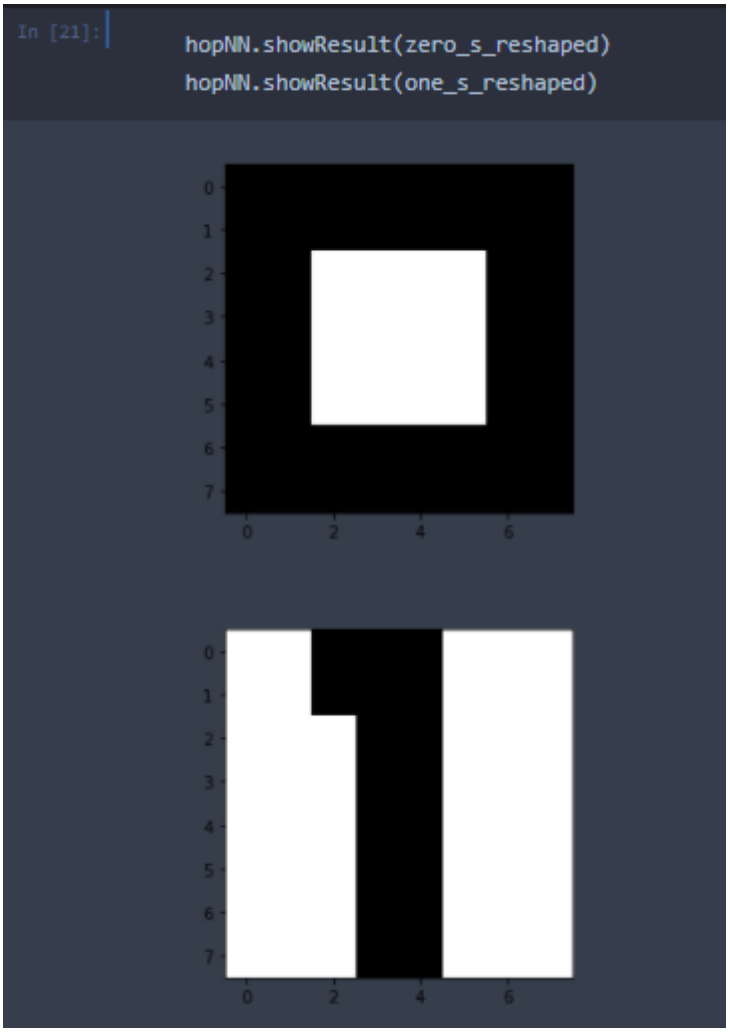
شکل 3-2 ورودی 0

رای ورودی 1 با شکل ماتریسی شکل 3-3

```
one_s = np.array( [[-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, 1, -1, -1, -1]])
```

شکل 3-3 ورودی 1

خروجی های مطلوب مطابق شکل 3-4 را به همراه داشته است



شکل 3-4 خروجی های مطلوب

ب) توانایی شبکه در تبدیل ورودی به خروجی مطلوب در حضور نویز:

در این بخش به هر دو داده ماتریس 0 و 1 که ماتریس های 8*8 هستند، به 30% آن نویز اضافه میشود، یعنی اگر unit ای 1 است 1- میشود و برعکس. سپس با استفاده از قاعده تصویر زیر، عملکرد شبکه خود را بررسی میکنیم.

Algorithm

Application Algorithm for the Discrete Hopfield Net

Step 0. Initialize weights to store patterns.
(Use Hebb rule.)

While activations of the net are not converged, do Steps 1–7.

Step 1. For each input vector x, do Steps 2–6.

Step 2. Set initial activations of net equal to the external input vector x:

$$y_i = x_i, (i = 1, \ldots n)$$

Step 3. Do Steps 4–6 for each unit Y_i.
(Units should be updated in random order.)

Step 4. Compute net input:

$$y\_in_i = x_i + \sum_j y_j w_{ji}.$$

Step 5. Determine activation (output signal):

$$y_i = \begin{cases} 1 & \text{if } y\_in_i > \theta_i \\ y_i & \text{if } y\_in_i = \theta_i \\ 0 & \text{if } y\_in_i < \theta_i. \end{cases}$$

Step 6. Broadcast the value of y_i to all other units.
(This updates the activation vector.)

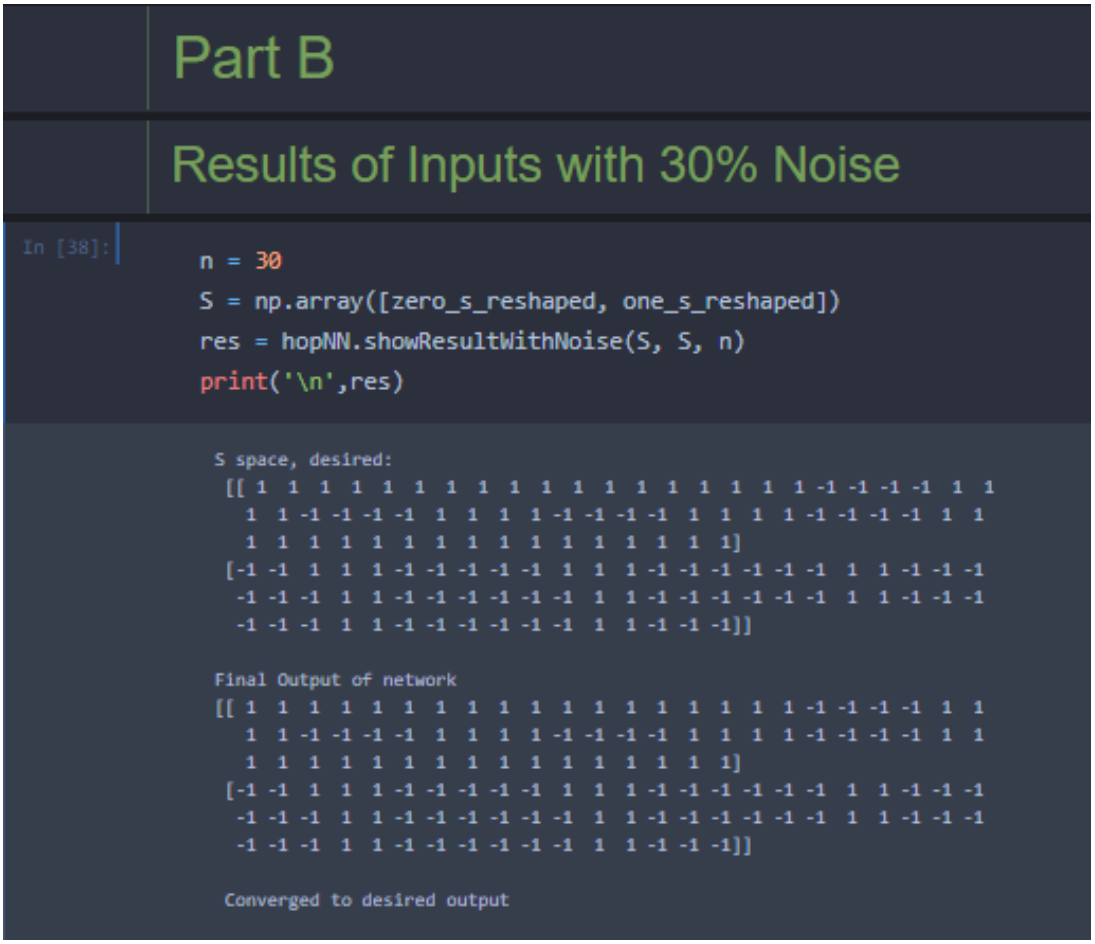
Step 7. Test for convergence.

شکل 3-5 الگوریتم شبکه هاپفیلد

در این بخش شرط Convergence به سه صورت گذارده شده است،

- رسیدن به برداری است که از قبل در S_space[sample] قرارداده شده است.
- رسیدن به برداری مشابه در محاسبات قبلی که با ادامه دادن الگوریتم به دیبتای جدیدی نمیرسیم.
- تعداد ایتریشن ها از حد تعریف شده فراتر رفته باشد.

شکل زیر خروجی شبکه برای این حالت را نشان میدهد:

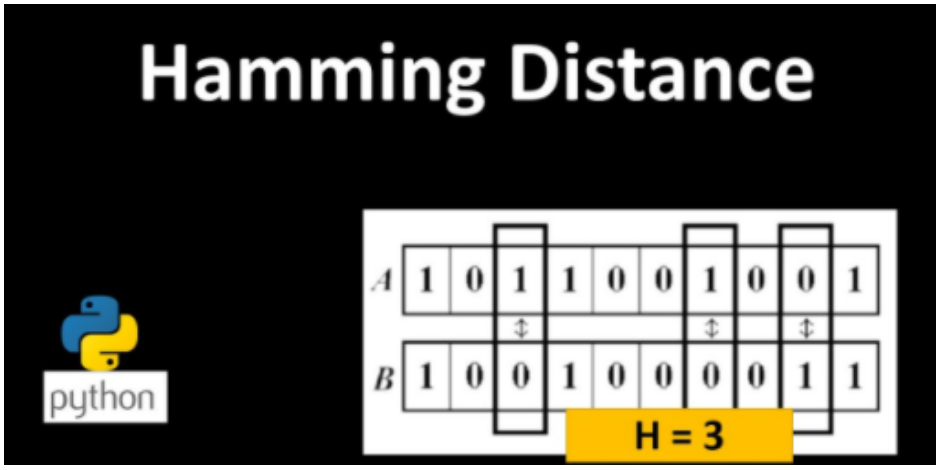


شکل 3-6 خروجی شبکه هاپفیلد برای دیتای همراه نویز

همانطور که مشاهده میشود شبکه ما در تهایت خروجی مطلوبی را حاصل کرده است و به آن همگرا شده است.

ج) فاصله Hamming و وابستگی ورودی ها

ابتدا تعریف Hamming Distance را مطرح میکنیم، فاصله Hamming در واقع بین دو دیتا، برابر با تعداد Unit های متفاوت در آن دو داده است، مثال آن در شکل زیر مشهود است:



شکل 3-7 تعریف فاصله Hamming

در شکل زیر فاصله Hamming را برای داده های خود محاسبه میکنیم:

```
hamming_dist = 0
for i in range(len(zero_s)):
    for j in range(len(zero_s[0])):
        if(zero_s[i][j] != one_s[i][j]):
            hamming_dist += 1
print(hamming_dist)
```

```
46
```

شکل 3-8 فاصله Hamming

مشاهده میشود که فاصله Hamming برابر با 46 شده است و از 64 Unit ، 71.8% متفاوت داشته ایم، این به ما نشان میدهد که ورودی هایی که انتخاب کرده ایم با تقریب خوبی متفاوت از یکدیگرند و با تقریب میتوان بیان نمود که داده ها وابستگی خیلی کمی نسبت به هم دارند (در 18 unit از 64 unit مشابه اند) و به همین علت عملکرد شبکه عصبی ما در برابر تغییر در چندین unit میتواند بسیار خوب و مطمئن باشد(همانطور که در بخش قبل نیز مشاهده نمودیم) .

سوال 4 – شبکه BAM

در این بخش به سراغ شبکه عصبی Bidirectional-Associative میرویم که یک فرم Recurrent از شبکه Hetro-Associative میسازد. در این پرسش 8 کاراکتر الفبای انگلیسی A, B, C, D, E, F, G, H که در ماتریس های 3*5 هستند به عنوان ورودی شبکه انتخاب شده اند و خروجی شبکه ماتریس 1*3 میباشد که چون 8 ورودی داریم میتوانیم آن را به $2^3 = 8$ به خروجی Map کنیم.
حال قسمت به قسمت، به پرسش های این بخش پاسخ میدهیم.

الف) ماتریس وزن، برای ورودی های A, B, C

از قاعده هب که در تصویر زیر آمده است برای بدست آوردن ماتریس وزن استفاده میکنیم.

Setting the Weights. The weight matrix to store a set of input and target vectors $\mathbf{s}(p):\mathbf{t}(p)$, $p = 1, \ldots, P$, where

$$\mathbf{s}(p) = (s_1(p), \ldots, s_i(p), \ldots, s_n(p))$$

and

$$\mathbf{t}(p) = (t_1(p), \ldots, t_j(p), \ldots, t_m(p)),$$

can be determined by the Hebb rule. The formulas for the entries depend on whether the training vectors are binary or bipolar. For binary input vectors, the weight matrix $\mathbf{W} = \{w_{ij}\}$ is given by

$$w_{ij} = \sum_p (2s_i(p) - 1)(2t_j(p) - 1).$$

For bipolar input vectors, the weight matrix $\mathbf{W} = \{w_{ij}\}$ is given by

$$w_{ij} = \sum_p s_i(p)t_j(p).$$

شکل 4-1 قاعده بدست آوردن وزن

چون فضایS و فضای T به صورت Bipolar تعریف شده اند از قاعده

$$w_{ij} = \sum_p s_i(p)\, t_j(p)$$

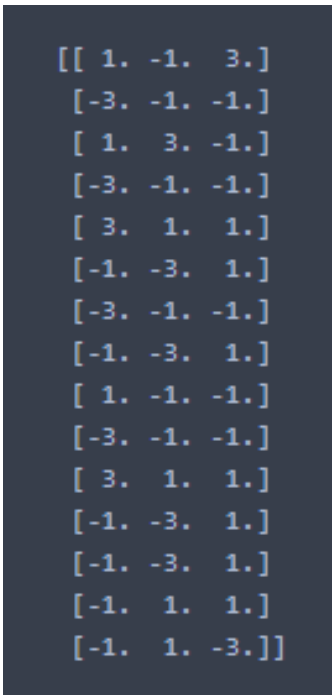
استفاده میشود.

نتیجه یادگیری، ماتریس وزن زیر را به دنبال داشت:

Part A

```
S = np.array([copy.deepcopy(A_s_reshaped), copy.deepcopy(B_s_reshaped), copy.deepcopy(C_s_reshaped)])
T = np.array([copy.deepcopy(A_t), copy.deepcopy(B_t), copy.deepcopy(C_t)])
bam = BAMNN(S, T)
bam.train()
bam.printWeight()
```

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



شکل 4-2 ماتریس وزن پس از یادگیری شبکه BAM

ب) تداعی ورودی به خروجی مطلوب از هر دو سمت

بعد از آموزش شبکه عصبی، حال میبایست برای هر دو طرف از S space به T space و برعکس، عملکرد شبکه عصبی را بررسی کنیم.

1- ابتدا از 3 حروف اول الفبا که در ماتریس های 3*5 میباشند، آن ها را Reshape میکنیم و به 15*1 تبدیل میکنیم و به عنوان ورودی به شبکه میدهیم، مطلوب آن است که شبکه خروجی متناظر با Mapped اولیه را به ما بدهد. نتیجه شبکه عصبی در شکل زیر قابل مشاهده میباشد.

```
Input given to Network
 [[-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]]
Desired output:
 [[-1 -1 -1]
 [-1 -1  1]
 [-1  1 -1]]
Network's output:
 [[-1 -1 -1]
 [-1 -1  1]
 [-1  1 -1]]

 => Converged to desired output
```

شکل 4-3 خروجی شبکه برای ورودی الفبا

مشاهده میشود که شبکه عصبی توانسته است به درستی به خروجی مطلوب برسد و به آن همگرا شود.

2- اینبار برعکس دفعه قبل، ماتریس 3*1 هایی که نماینده 3 حرف اول الفبا هستند را به عنوان ورودی به شبکه میدهیم، مطلوب آن است که شبکه خروجی 3 حروف اول الفبا که در ماتریس های 3*5 میباشند، آن ها را Reshape کرده ایم و به 15*1 تبدیل کرده ایم را برای ما باز گرداند.

```
Input given to Network
 [[-1 -1 -1]
 [-1 -1  1]
 [-1  1 -1]]
Desired output:
 [[-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]]
Network's output:
 [[-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.]]

 => Converged to desired output
```

شکل 4-3 خروجی شبکه برای ورودی الفبا

مشاهده میشود که شبکه عصبی توانسته است به درستی به خروجی مطلوب برسد و به آن همگرا شود.

ج) تداعی ورودی به خروجی مطلوب در حضور نویز:

در این بخش سعی میشود عملکرد شبکه عصبی در حضور نویز بررسی شود، در 100 بار تکرار، ابتدا به داده هایمان یکبار 10% و بار دیگر، 40% نویز اضافه میکنیم بدان معنا که اگر داده ای در Unitی برابر با 1 بود آن را 1- میکنیم و برعکس.

حال ابتدا برای حالت 10% نویز روی ورودی، دقت را میسنجیم.

شکل زیر نشان دهنده دقت تعداد پیکسل های درست تشخیص داده شده با وجود 10% نویز میباشد:

```
cnt = 0
for i in range(len(Y_t_array)):
    for j in range(len(Y_t_array[0])):
        if(Y_t_array[i][j] == _T_space[i][j]):
            cnt += 1
print("Correct detected pixels: ",(cnt*100)/(len(Y_t_array) * len(Y_t_array[0])),"%")
print("accuracy: ",numTrue, "%")
```

```
Correct detected pixels:  100.0 %
accuracy:  100 %
```

شکل 4-4 توانایی شبکه دقت با 10% نویز

نتیجه نشان میدهد که شبکه آموزش دیده، در برابر 10% نویز کاملا مقاوم بوده و به 100% دقت توانسته است برسد.

حال برای حالت 40% نویز روی ورودی، دقت را میسنجیم.

شکل زیر نشان دهنده دقت تعداد پیکسل های درست تشخیص داده شده با وجود 40% نویز میباشد:

```
cnt = 0
for i in range(len(Y_t_array)):
    for j in range(len(Y_t_array[0])):
        if(Y_t_array[i][j] == _T_space[i][j]):
            cnt += 1
print("Correct detected pixels: ",cnt/(len(Y_t_array) * len(Y_t_array[0])))
print("accuracy: ",numTrue, "%")
```

```
Correct detected pixels:  0.7777777777777778
accuracy:  5 %
```

شکل 4-5 توانایی شبکه دقت با 40% نویز

نتیجه نشان میدهد که شبکه آموزش دیده، در برابر 40% عملکرد خوبی از خود نشان نداده است و به دقت مطلوب ما نرسیده است در حالی که شرطی برای خاتمه یک Convergence را 10000 بار ادامه دادن بدون رسیدن به حلقه تکراری داده بودیم.

د) تداعی خروجی به ورودی در حضور Missing data

در این بخش بررسی میشود که در حالی که Missing data موجود باشد، یعنی اطلاعات 1 یا 1- تبدیل به 0 شده باشد، عملکرد شبکه به چه گونه ای خواهد بود.

در اینجا به عنوان S space به شبکه عصبی [1- ,1- ,0] میدهیم که اگر دقت کنیم داده ای است که مپ شده حرف الفبای A که برابر باشد با [1- ,1- ,1-] درایه اولش Miss شده است.

حال میخواهیم ببینیم آیا شبکه عصبی آموزش داده شده ما قادر هست که آن را بازیابی کند یا خیر.
شکل زیر نتیجه خروجی شبکه عصبی میباشد:

```
Input given to Network
 [[ 0 -1 -1]]
Desired output:
 [[-1  1 -1  1 -1  1  1  1  1  1 -1  1  1 -1  1]]
Network's output:
 [[-1.  1. -1.  1. -1.  1.  1.  1.  1.  1. -1.  1.  1. -1.  1.]]

 => Converged to desired output
```

شکل 4-ج1 خروجی شبکه عصبی با Missing data

مشاهده میشود که شبکه عصبی توانسته است که خروجی مطلوب را بدست آورد، حال نکته ای که مطرح است این است که در آموزش این شبکه عصبی برای درایه اول فقط1- قرار گرفته بود که ادامه آن نماینده حرف انگلیسی A بود، این در حالی است که حرف E با بردار Map شده [1,-1,-1] نیز میتوانست در شبکه عصبی موجود باشد که در این صورت شبکه احتمالا دچار خطا میشد.

ه) ماتریس وزن، برای ورودی های هر 8 حرف اول الفبای انگلیسی

از قاعده هب که در تصویر زیر آمده است برای بدست آوردن ماتریس وزن استفاده میکنیم.

Setting the Weights. The weight matrix to store a set of input and target vectors $\mathbf{s}(p):\mathbf{t}(p)$, $p = 1, \ldots, P$, where

$$\mathbf{s}(p) = (s_1(p), \ldots, s_i(p), \ldots, s_n(p))$$

and

$$\mathbf{t}(p) = (t_1(p), \ldots, t_j(p), \ldots, t_m(p)),$$

can be determined by the Hebb rule. The formulas for the entries depend on whether the training vectors are binary or bipolar. For binary input vectors, the weight matrix $\mathbf{W} = \{w_{ij}\}$ is given by

$$w_{ij} = \sum_p (2s_i(p) - 1)(2t_j(p) - 1).$$

For bipolar input vectors, the weight matrix $\mathbf{W} = \{w_{ij}\}$ is given by

$$w_{ij} = \sum_p s_i(p)t_j(p).$$

شکل 4-ه-1 قاعده بدست آوردن وزن

چون فضایS و فضای T به صورت Bipolar تعریف شده اند از قاعده

$$w_{ij} = \sum_p s_i(p)\, t_j(p)$$

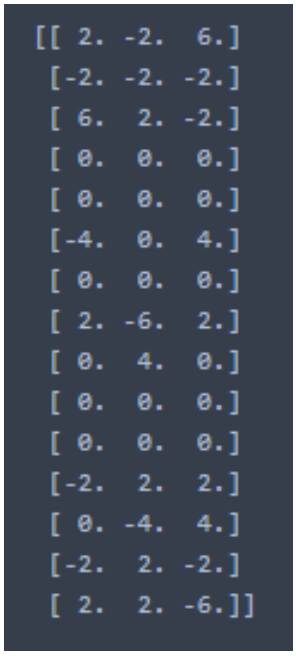
استفاده میشود.

نتیجه یادگیری، ماتریس وزن زیر را به دنبال داشت:

Part E

```
S = np.array([copy.deepcopy(A_s_reshaped), copy.deepcopy(B_s_reshaped), copy.deepcopy(C_s_reshaped),
              copy.deepcopy(D_s_reshaped), copy.deepcopy(E_s_reshaped), copy.deepcopy(F_s_reshaped),
              copy.deepcopy(G_s_reshaped), copy.deepcopy(H_s_reshaped)])
T = np.array([copy.deepcopy(A_t), copy.deepcopy(B_t), copy.deepcopy(C_t),
              copy.deepcopy(D_t), copy.deepcopy(E_t), copy.deepcopy(F_t),
              copy.deepcopy(G_t), copy.deepcopy(H_t)])
bam = BAMNN(S, T)
bam.train()
W = bam.giveWeight()
print(W)
```

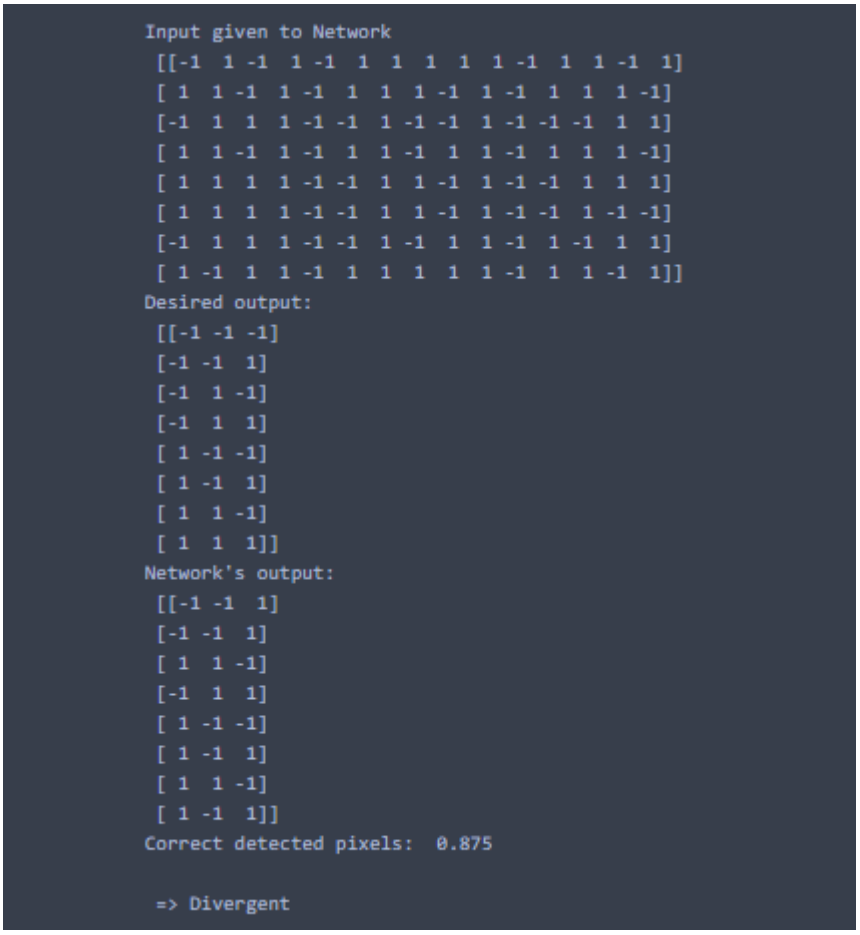
```
[[ 2. -2.  6.]
 [-2. -2. -2.]
 [ 6.  2. -2.]
 [ 0.  0.  0.]
 [ 0.  0.  0.]
 [-4.  0.  4.]
 [ 0.  0.  0.]
 [ 2. -6.  2.]
 [ 0.  4.  0.]
 [ 0.  0.  0.]
 [ 0.  0.  0.]
 [-2.  2.  2.]
 [ 0. -4.  4.]
 [-2.  2. -2.]
 [ 2.  2. -6.]]
```



شکل 4-ه-2 ماتریس وزن پس از یادگیری شبکه BAM

و) توانایی ذخیره هر 8 پترن توسط شبکه عصبی

در این قسمت سعی میشود توانایی شبکه با هر 8 حرف انگلیسی سنجیده شود، یعنی به عنوان ورودی در S space به عنوان ورودی داده میشود و خروجی های مطلوب متناظر آن ها طلب میشوند که انتظار میرود شبکه عصبی به آن ها برسد.



4-و-1 وضعیت خروجی شبکه با تمام Letter ها

پس از 4000 بار، باز هم Divergent شد یعنی تمامی اطلاعات بدون نویز بازیابی نشدند و دقت پیکسل های بازیابی شده حدود 87.5% شده است.

پس از انجام برنامه، متوجه شدیم که هیچ گاه به دقت 100 درصدی برای تمامی 8 Letter ها نمیرسیم. علت این امر این است که تعداد بیت هایی که برای Map کردن استفاده نمودیم خیلی کم بود و 3 عدد بیت Bipolar بود، در حالی که میتوانست خیلی بیشتر باشد، همچنین میتوانستیم سایز داده هایمان را نیز بزرگتر بگیریم، به امید آنکه Hamming Difference بیشتری ایجاد شود و توانایی شبکه ما برای Recognition الفبا بالاتر رود.

در بخش های قبل دیدیم که شبکه توانست در حضور 3 حرف عملکرد مناسبی از خود نشان دهد و در حالتی که بدون نویز بودیم، دقت 100% و در حالت 10% نویز هم دقت 100%ی از خود نشان داد.

سعی میکنیم برای حالتی که 4 Letter است هم امتحان کنیم، (از قسمت بعدی کمک میگیریم و متوجه میشویم که برای آنکه چهار حرف الفبا را انتخاب کنیم، سعی میکنیم حروفی را انتخاب نماییم که بیشترین Hamming Distance ها را در برابر هم دارا هستند. بنظر میرسد با انتخاب C

و B و G و A که نسبت به هم Hamming Distance زیادی دارند، پترن به احتمال زیاد با موفقیت ذخیره میشود.)

وزن با Train این 4 حرف الفبا:

```
[[ 0. -2.  4.]
 [-2.  0. -2.]
 [ 2.  4. -2.]
 [-2.  0. -2.]
 [ 2.  0.  2.]
 [-2. -4.  2.]
 [-2.  0. -2.]
 [-2. -4.  2.]
 [ 2.  0. -2.]
 [-2.  0. -2.]
 [ 2.  0.  2.]
 [ 0. -2.  0.]
 [-2. -4.  2.]
 [ 0.  2.  0.]
 [ 0.  2. -4.]]
```

4-و-2 وزن با 4 حرف

```
Input given to Network
 [[-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]]
Desired output:
 [[-1 -1 -1]
 [-1 -1  1]
 [-1  1 -1]
 [ 1  1 -1]]
Network's output:
 [[-1 -1 -1]
 [-1 -1  1]
 [-1  1 -1]
 [-1  1 -1]]
Correct detected pixels:  0.9166666666666666

 => Divergent
```

4-و-3 خروجی شبکه با 4 letter

میبینیم که با اینکه دقت در پیکسل ها افزایش یافت اما برای داده های بدون نویز، به دقت 100%ی نرسیدیم و باز هم میتوان گفت Divergent شدیم.

نتیجه گیری: شبکه توانست در حضور 3 حرف عملکرد مناسبی از خود نشان دهد و در حالتی که بدون نویز بودیم، دقت 100% و در حالت 10% نویز هم دقت 100%ی از خود نشان داد.

پس ماکسیمم 3 حرف دقت 100%ی میرسیم

ز) توانایی ذخیره هر 8 پترن توسط شبکه عصبی

در این قسمت سعی میشود ابتدا Hamming Distance را بدست آورده آن را بررسی کنیم و نتیجه مطلوب را از آن بدست آوریم .

در شکل زیر Hamming Distance را برای هشت حرف الفبای انگلیسی به ترتیب بدست آورده ایم:

Part F:

```
S = np.array([copy.deepcopy(A_s_reshaped), copy.deepcopy(B_s_reshaped), copy.deepcopy(C_s_reshaped),
              copy.deepcopy(D_s_reshaped), copy.deepcopy(E_s_reshaped), copy.deepcopy(F_s_reshaped),
              copy.deepcopy(G_s_reshaped), copy.deepcopy(H_s_reshaped)])
char_cnt = 1
hamming_dist = []
for char1 in S:
    hamming_dist_char_i = []
    for char2 in S:
        cnt = 0
        for i in range(len(char1)):
            if(char1[i] != char2[i]):
                cnt += 1
        hamming_dist_char_i.append(cnt)
    hamming_dist.append(hamming_dist_char_i)
    char_cnt += 1
for i in range(len(hamming_dist)):
    print(hamming_dist[i])
```

```
[0, 4, 7, 4, 6, 6, 5, 3]
[4, 0, 7, 2, 4, 4, 7, 5]
[7, 7, 0, 7, 3, 5, 2, 8]
[4, 2, 7, 0, 6, 6, 5, 5]
[6, 4, 3, 6, 0, 2, 5, 5]
[6, 4, 5, 6, 2, 0, 7, 5]
[5, 7, 2, 5, 5, 7, 0, 6]
[3, 5, 8, 5, 5, 5, 6, 0]
```

```
   A  B  C  D  E  F  G  H
A [0, 4, 7, 4, 6, 6, 5, 3]
B [4, 0, 7, 2, 4, 4, 7, 5]
C [7, 7, 0, 7, 3, 5, 2, 8]
D [4, 2, 7, 0, 6, 6, 5, 5]
E [6, 4, 3, 6, 0, 2, 5, 5]
F [6, 4, 5, 6, 2, 0, 7, 5]
G [5, 7, 2, 5, 5, 7, 0, 6]
H [3, 5, 8, 5, 5, 5, 6, 0]
```

شکل 4-ز-1 ماتریس Hamming Distance

بدیهتا قطر اصلی این ماتریس برابر با صفر میباشد، که بدان معناست که حروف الفبای مشابه، در تمامی unit ها همانند یکدیگرند.

مشاهده میشود که بزرگترین Hamming distance به ترتیب برابر با 8 و 7 میباشد که 8 متعلق به (C,H) است و 7 متعلق به داده های (A,C) و (B,C) و (C,D) و (B,G) و (F,G) میباشد.

برای آنکه سه حرف الفبا را انتخاب کنیم، سعی میکنیم حروفی را انتخاب نماییم که بیشترین Hamming Distance ها را در برابر هم دارا هستند. بنظر میرسد با انتخاب C و B و A که نسبت به هم Hamming Distance زیادی دارند، پترن به احتمال زیاد با موفقیت ذخیره میشود.

برای آنکه چهار حرف الفبا را انتخاب کنیم، سعی میکنیم حروفی را انتخاب نماییم که بیشترین Hamming Distance ها را در برابر هم دارا هستند. بنظر میرسد با انتخاب C و B و G و A که نسبت به هم Hamming Distance زیادی دارند، پترن به احتمال زیاد با موفقیت ذخیره میشود.