به نام خدا

دانشگاه تهران

دانشکده مهندسی برق و کامپیوتر

شبکه های عصبی و یادگیری عمیق

تمرین سری 3

نام و نام خانوادگی	مهسا مسعود
شماره دانشجویی	810196635
تاریخ ارسال گزارش	1400.3.12

فهرست گزارش سوالات

سوال 1 – شناسایی حروف با استفاده از روش هب.....3

سوال ۲ – شبکه خود انجمنی.....8

سوال 3 – شبکه هاپفیلد.....11

سوال 4 – شبکه BAM.....14

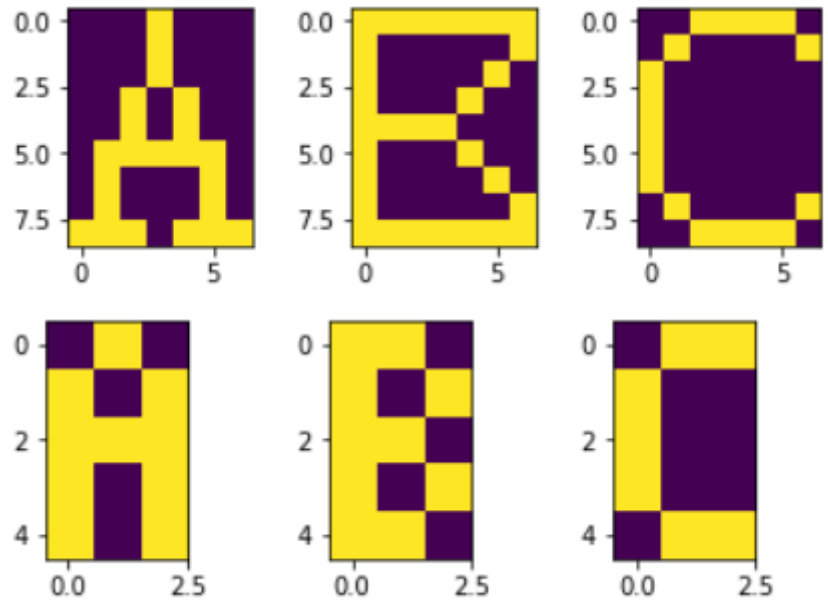
سوال 1 – شناسایی حروف با استفاده از روش هب

در طراحی های شبکه های عصبی با هدف تداعی کردن یک پترن ورودی نیاز به حافظه است و خروجی مورد نظر را می دهد. حال پترن ورودی می تواند به صورت explicit و implicitباشد. منظور از ورودی implicit آن است که ورودی ها توسط یک اردر زمانی یا مکانی به شبکه داده می شوند . در غیر این صورت ورودی ها explicit است. شاید بتوان گفتن شبکه ای که برای ورودی های explicit طراحی می گردد، از جنس مسایل طبقه بندی است با این تفاوت که تعداد کلاس ها بیشتر می باشد. در واقع شبکه های عصبی با کاربرد حافظه در صورتی که ورودی ها explicit باشند، نیاز به یک حافظه استاتیک دارند. بدیهی است که طراحی یک شبکه عصبی برای ورودی های explicit ساده تر از طراحی شبکه برای داده های implicit است. یکی از الگوریتم های متداول برای شبکه های عصبی با حافظه استاتیکی ، قانون یادگیری هب است. در ادامه یک شبکه عصبی تک لایه با قانون هب طراحی می گردد.

الف)

هدف از طراحی این قسمت، کاهش ابعاد سه حرف A, B,C است. در ابتدا حروف مذکور به صورت ماتریس 7*9 داده شده اند و انتظار می رود که به ابعاد 3*5 تداعی شوند. در ادامه هر حرف و تارگت مورد نظر آن آمده است.

زرد نماد درایه 1 و بنفش نماد درایه -1 است.



شکل 1-1 – ورودی های explicit به صورت ماتریس 7*9

در ابتدا هر حرف 7*9 به صورت یک ماتریس با ابعاد -1 و 1 در شبکه به صورت یک بردار 63 تایی ذخیره شده است و همه به عنوان ورودی شبکه کانکت شده و ماتریس ورودی با ابعاد 63*3 ساخته شد. بدیهی است سه تعداد نمونه های ورودی است. از طرفی ماتریس تارگت نیز ساخته شد . به صورتی که هر حرف خروجی با ابعاد 3*5 به یک بردار 15 درایه ای تبدیل شده و پس از کانکت شدن به یک بردار هدف با ابعاد 15*3 تبدیل می گردد.

برای محاسبه ماتریس وزن، بردار نمونه ها یا همان ورودی به صورت ترنسپوز شده در ماتریس هدف ضرب شد. از آنجایی که ماتریس هاس ورودی و تارگت برای هر نمونه به ترتیب دارای 63 و 15 درایه بودند، ابعاد ماتریس وزن به صورت 15*63 در آمد.

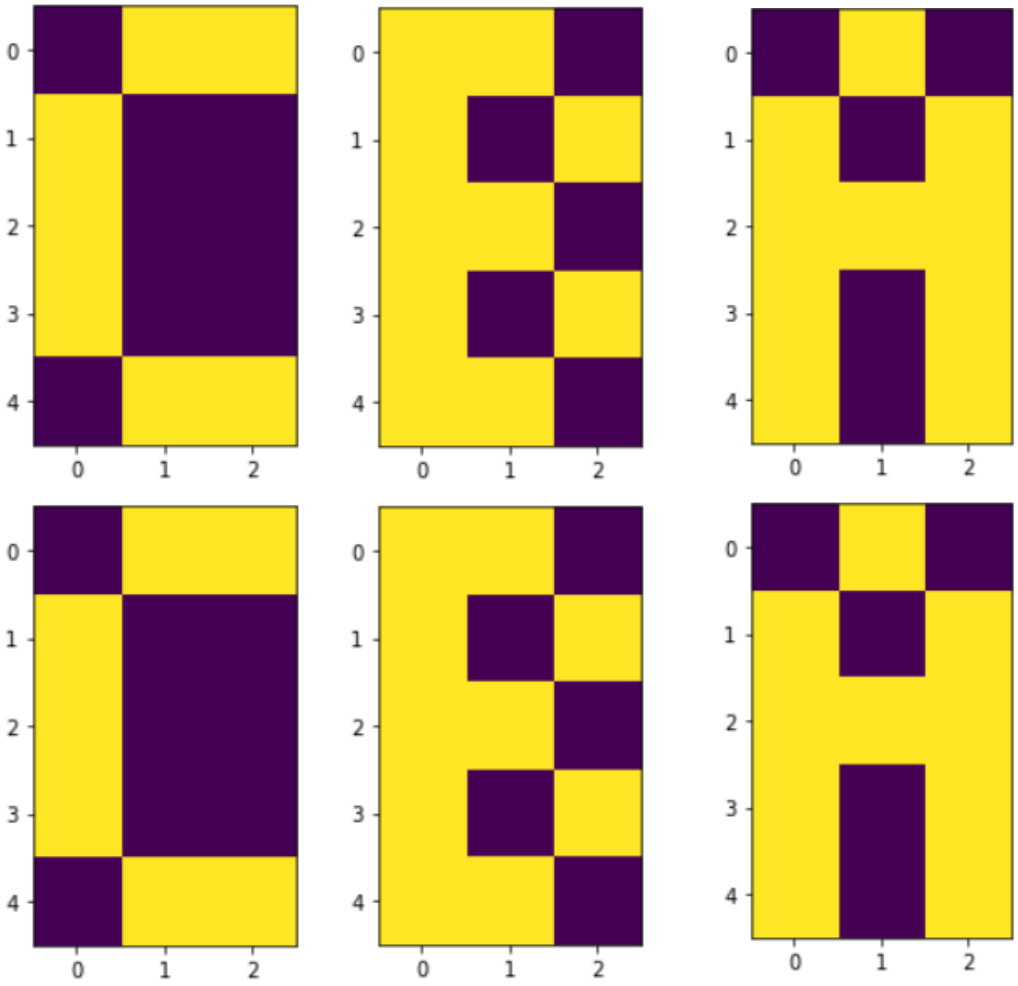
برای آزمودن اینکه آیا به ازای هر نمونه ، تارگت مورد نظر تداعی شده یا نه کافیست نمونه مورد نظر در ماتریس وزن ضرب گردد و از یک تافع فعالساز عبور کند.

بدیهی است بدون این تابع شبکه خطی بوده و فاقد نقاط اکسترمم خواهد بود. شبکه یک تله میخواهد که به ازای ورودی های اغتشاش دار شده هم در آن گیر کند. در واقع شبکه کاملا خطی نمی تواند حافظه ایجاد کند. تابع فعالساز استفاده شده در این بخش تابع sign به صورت bipolar است. به گونه ای که اگر هر یک از درایه های بردار حاصل از ضرب نمونه در ماتریس وزن بزرگتر از صفر باشد، درایه 1 و در

غیر این صورت درایه -1 گردد. در نهایت پس از اعمال هر یک از نمونه ها به شبکه خروجی متناظر دیده شد.

همچنین از مفهوم hamming و distance نیز استفاده شده است. به طوری که خروجی هر شبکه به ازای هر ورودی با تارگت ها مقایسه شد که برای هر 3 نمونه این فاصله 0 بود.

مشاهده میشود برای این 3 حرف تارگت ها و خروجی های شبکه کاملا بر هم منطبق هستند.



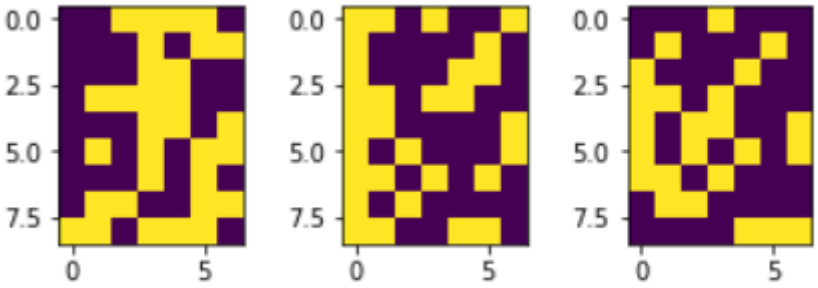
شکل 1-2 – خروجی های explicit به صورت ماتریس 3*5

ب -1) افزودن اغتشاش به شبکه

هدف این قسمت آزمودن قدرت شبکه برای داده هایی متفائت از داده ها ورودی است به گونه ای که برخی از داده ها با نویز ترکیب و یا برخی اطلاعات حذف میشوند.

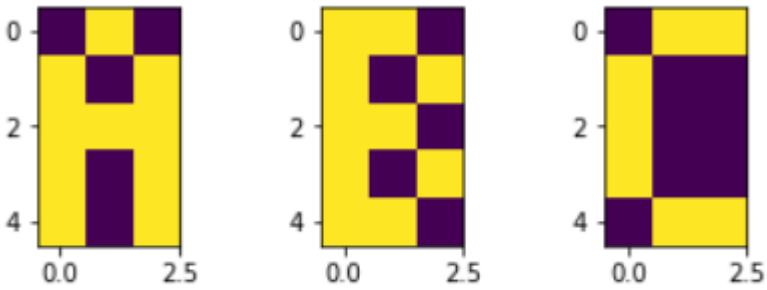
- افزودن نویز به نمونه های ورودی

بدیهی است که این افزودن نویز باید به صورت تصادفی باشد. برای تحقق این امر، ابتدا 10 و 25 درصد از خانه های ورودی به صورت رندوم انتخاب شد اما در نهایت شبکه به 33 درصد نویز مقاوم بود. اینکار به این صورت انجام شد که خانه های انتخاب شده اگر مقدار 1 دارند به -1 و بالعکس تبدیل شدند.



شکل 1-3 – سمپل های ورودی با نویز 33 درصد

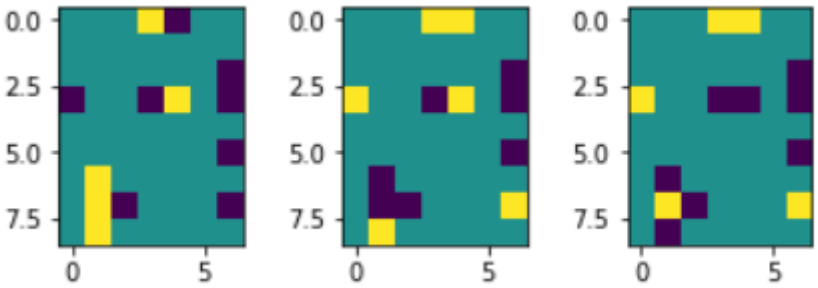
حال با محاسبه حاصلضرب هر نمونه در ماتریس وزن خروجی شبکه و گذراندن ار تابع فعالساز، خروجی شبکه به صورت زیر مشاهده گردید. بنابراین همانظور که از قبل گفته شد، اگر ورودی های شبکه طراحی شده تا حدود 33 درصد دچار نویز شوند ، شبکه مقاوم بوده و به درستی جواب می دهد.



شکل 1-4 – خروجی شبکه برای سمپل های ترکیب شده با نویز

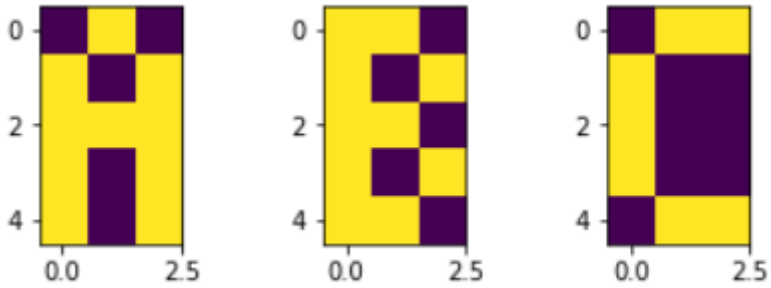
- افزودن Missing Point به ورودی های شبکه

در این بخش پس از انتخاب خانه های ماتریس های ورودی، برای ایجاد Missing Point ، خانه های انتخاب شده فارغ از مقادیری که دارند به صفر تبدیل شده اند که در ادامه آمده است.



شکل 1-5 – سمپل های شبکه با حذف مقادیر برخی خانه ها

در این حالت نیز پس از محاسبه حاصلضرب ماتریس وزن و نمونه ها و عبور از تابع فعالساز، خروجی به صورت زیر مشاهده گردید. در این حالت شبکه تا 79.3 درصد قابلیت باز شناسی داشت.



شکل 1-6 - خروجی شبکه با حذف مقادیر برخی خانه ها

سوال ۲ – شبکه خود انجمنی

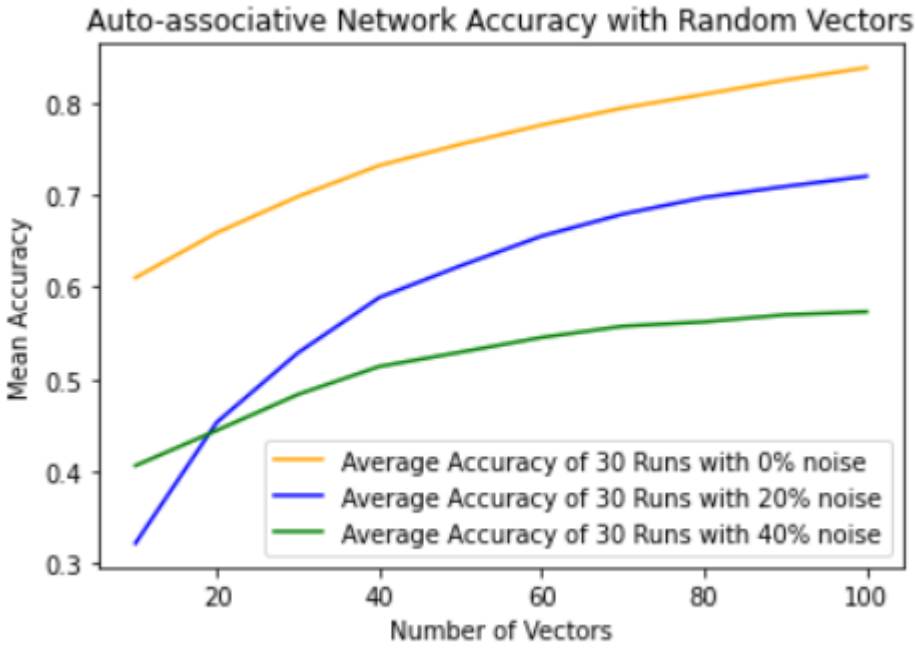
در این سوال یک شبکه خود انجمنی می سازیم که تعدادی بردار با مقادیر bipolar و رندوم را به خاطر بسپارد.

سپس به آن ها نویز و تعداد بردار متفاوت اعمال کرده و نتیجه را مشاهده می کنیم.

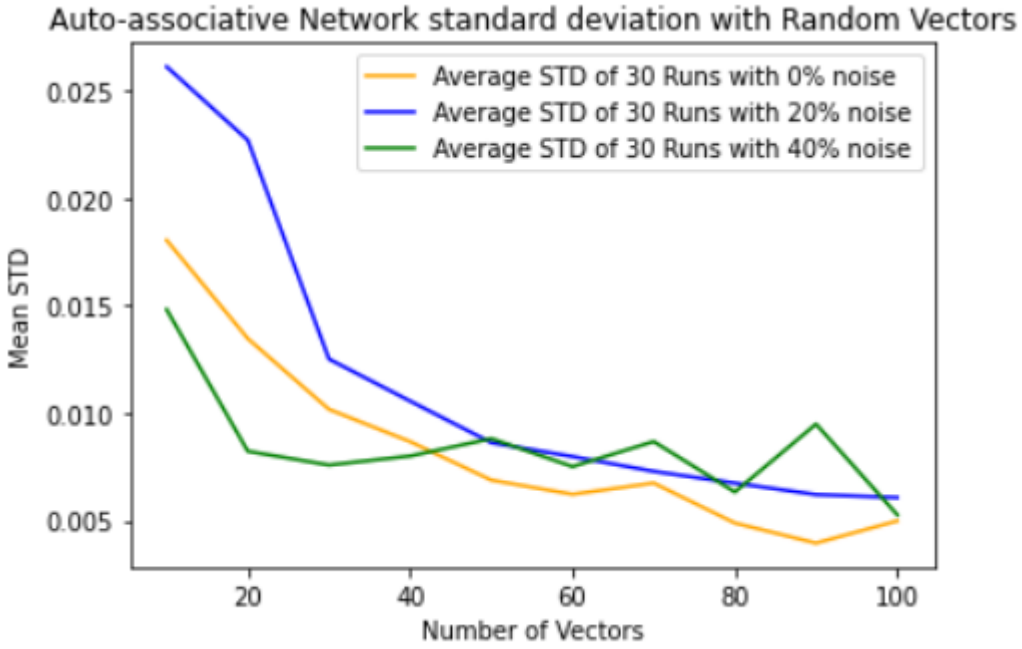
هدف در شبکه های خود انجمنی این است که خروجی را در ورودی تداعی کنیم. در واقع فایده این شبکه ها این است که بتوان بعد از disturb شدن ورودی، آن را بازیابی کرد. شبکه خود تداعی گر شبیه auto encoder است با این تفاوت که هدف در اتوانکودر ها این است که با dimension reduction ، همبستگی ها را بشکنند. اما در این شبکه ها ، ابعاد کاهش نمیابد و هدف disturbance rejection است. بنابراین در این شبکه ها حافظه ای ساخته می شود که نسبت به disturbance مقاوم هستند.

الف)

با کمک قانون هب تغییر یافته ماتریس وزن ها را تشکیل می دهیم.



شکل 2-1 – دقت شبکه خود انجمنی با بردار های تصادفی



شکل 2-2– انحراف از معیار شبکه خود انجمنی با بردار های تصادفی

ب)

می توان مشاهده کرد که میانگین و انحراف از معیار دقت در داده های بدون نویز بهتر بوده است. همچنین هر چه تعداد بردار ها زیاد شدند ، دقت نیز بیشتر شده است و از حدی تعداد الگوهایی که شبکه میتواند به خاطر بسپرد عبور نکردیم. افزایش دقت به این دلیل بوده است که با افزودن تعداد بردار ها از توانایی شبکه در به یاد سپاری بیشتر استفاده شده است.

ج)

جدول 1 – روش ها و ظرفیت های متناظر

Weight matrix	Hetro-Associative	Auto-Associative (bipolar patterns)
Hebbian matrix	Capacity = n	Capacity = n
Modified Hebbian Matrix	-	Capacity = n-1

طبق جدول بالا ظرفیت شبکه 99 می باشد و ما از این حد تقریبا عبور نکرده ایم. به همین دلیل است که دقت خوبی در تعداد بالای بردار ها داشتیم. عنصر موثر دیگر نیز رابطه بین شبکه هاست زیرا این مقادیر

برای ورودی های عمود برهم است و اگر کاملا برهم عمود بودند ، در حالت بدون نویز به 100 درصد دقت می رسیدیم. بردار ها به صورت رندوم مقدار دهی شده اند برای همین لزوما همه بر هم عمود نیستند و دقت شبکه حدود 80 درصد شده است.

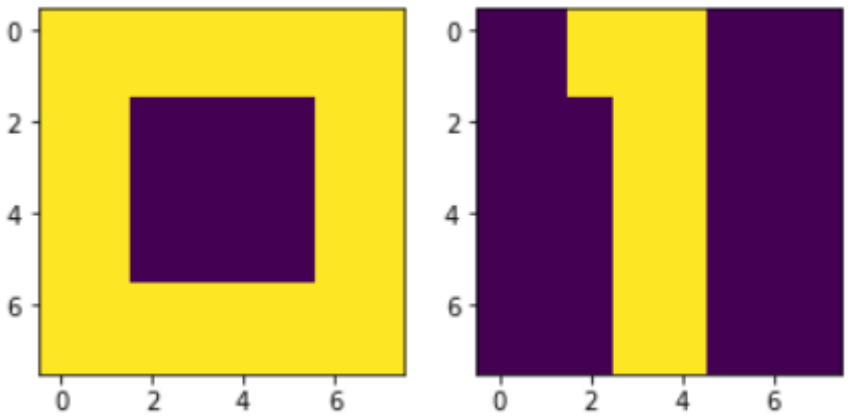
سوال 3 – شبکه هاپفیلد

در قسمت های قبل شبکه های عصبی با کاربرد حافظه طراحی گردید. شبکه های طراحی شده در بخش اول توسط قانون هب و در بخش دوم یک شبکه خود تداعی گر توسط قانون هب اصلاح شده آموزش داده شده اند. در واقع یادگیری به روش هب یک قابلیت برای به حافظه سپاری بردار های ورودی نسبت به بردار خروجی می دهد. برای قوی تر کردن شبکه ها طراحی شده میتوان خروجی را مجدد در ورودی اعمال کرد. بنابراین شبکه ها به صورت recurrent می شوند. به عنوان مثال شبکه هاپفیلد از خروجی ها نیز در ساخت شبکخ بهره می برد. لازم به ذکر است ماتریس وزن شبکه هاپفیلد همان ماتریس وزن هب است. شبکه های recurrent قدرت به حافظه سپاری را افزایش می دهند و با تعداد محدودی تکرار همگرا خواهند شد. در ادامه یک شبکه هاپفیلد برای ذخیره سازی اعداد 0 و 1 طراحی می گردد.

الف)

هدف این بخش طراحی یک شبکه recurrent توسط الگوریتم هاپفیلد است. در این شبکه قرار است اعداد 0 و 1 که به صورت ماتریس های 8*8 هستند ذخیره شود. لازم به ذکر است تارگت این شبکه همان سمپل های ورودی بوده و شبکه به صورت خود تداعی گر است.

در ابتدا سمپل های ورودی به صورت ماتریس های 8*8 و سپس به صورت یک بردار 1*64 ذخیره شده و پس از کانکت شدن دو نمونه، بردار 2*64 را میسازیم.

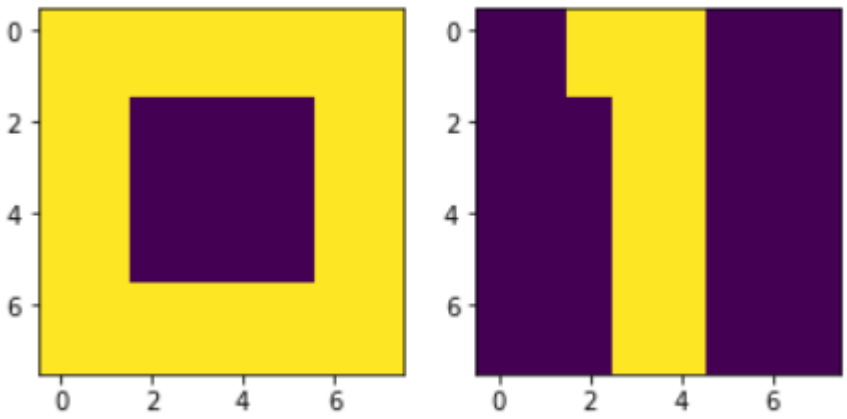


شکل 3-1 – سمپل های ذخیره شده در شبکه هاپفیلد طراحی شده

از آنجایی که شبکه خود تداعی گر بوده ، لازم است برای ساختن ماتریس وزن، قطر ماتریس 0 گردد تا از تبدیل ماتریس وزن به ماتریس همانی جلوگیری شود. بنابراین برای ساخت ماتریس وزن ماتریس سمپل های ورودی به صورت ترنسپوز در خودش ضرب می شود و قطر آن صفر می گردد. برای حذف قطر ماتریس وزن، از 2 برابر یک ماتریس همانی استفاده کردیم. در نهایت ماتریس وزن بر 250 تقسیم گردید تا فواصل بین ورودی و خروجی زیاد نشود و شبکه بتواند مقادیر تارگت را با ورودی های 1 و -1 تداعی کند.

برای این بخش تابع signرا طراحی کردیم. سپس الگوریتم هاپفلد به صورت زیر پیاده می شود:

به ازای هر سمپل، تا زمانی که همگرایی صورت نگرفته باشد، عملیات را ادامه می دهد. در ابتدا یک random order تعریف می شود که خروجی هر سمپل به ازای آن به روز رسانی گردد. حال برای هر random order ، ماتریس خروجی در ستون nام از ماتریس وزن ضرب می گردد. منظور از ستون n ام ماتریس وزن، ستون مربوط به random order است. البته ماتریس خروجی و وزن به صورت درایه ای در هم ضرب شده اند و در نهایت یک عدد اسکالر می دهد. خروجی بعد از عبور تابع فعالساز آماده شده و به ازای هر دو ورودی خودش را تداعی میکند.



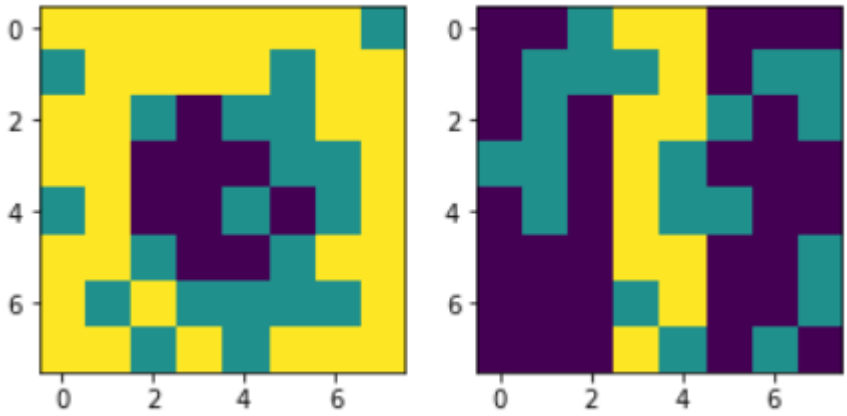
شکل 3-2– خروجی شبکه خود تداعی گر به ازای سمپل و ماتریس وزنی که بر 250 تقسیم شده است

ب) افزودن نویز:

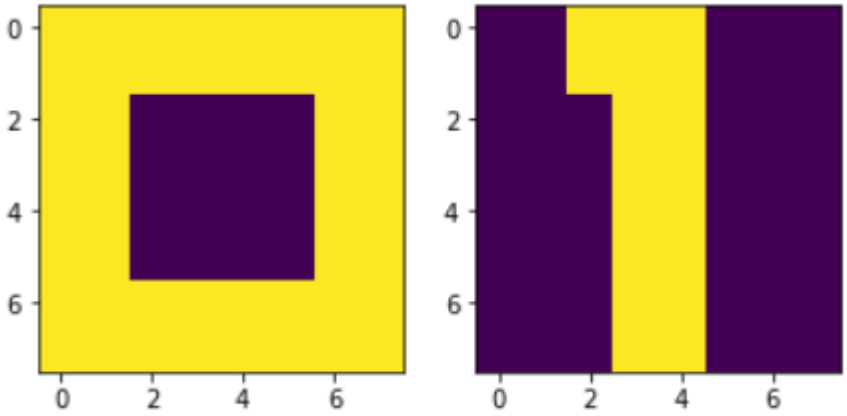
برای 20 درصد از تعداد خانه های نویزی به صورت قرینه کردن مقدار آن خانه اعمال گردید.

این رقم را تا 20 خانه از 64 تا که حدود 31 درصد نویز می باشد نیز بالا بردیم و مشاهده کردیم که خروجی به درستی تداعی می شود.

برای نویزی کردن خانه های انتخاب شده را صفر کردیم.



شکل 3-3 – سمپل های نویزی شده



شکل 3-4 – خروجی شبکه خود تداعی گر برای سمپل های نویزی شده

ج)

برای محاسبه Hamming Distance ، این مقدار حدودا برابر 46.64 و درصد آن 72٪ می باشد. نشان دهنده این است که فاصله این دو کاراکتر از هم زیاد است و برای همین است که نسبت به نویز مقاومت وجود دارد. (تنها در 18 خانه اشتراک دارند).

سوال 4 – شبکه BAM

در این سوال به طراحی شبکه هاپفیلد برای به حافظه سپردن و تشخیص کاراکتر های دو عدد یک و صفر از هم می پردازیم. ابتدا با کمک قانون هب، یادگیری را کامل میکنیم و ورودی های نویزی را به مقادیر اصلی می رسانیم.

الف)

ابتدا تنها با حروف A , B , C ، شبکه را می سازیم و آموزش می دهیم و ماتریس وزن ها به شکل زیر است:

```
array([[ 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 4-1 – ماتریس وزن شبکه ساخته شده با 3 حرف

ب)

نتیجه شبکه با 3 حرف در دو پیمایش به دست می آید و خطا نیز 0 است.

```
iterations: 2
Real Input:  [-1  1 -1  1 -1  1  1  1  1  1 -1  1  1 -1  1]
Final input: [-1  1 -1  1 -1  1  1  1  1  1 -1  1  1 -1  1]
Real Output:  [-1 -1 -1]
Final output: [-1 -1 -1]
Error :  0
Error : 0
iterations: 2
Real Input:  [ 1  1 -1  1 -1  1  1  1 -1  1 -1  1  1  1 -1]
Final input: [ 1  1 -1  1 -1  1  1  1 -1  1 -1  1  1  1 -1]
Real Output:  [-1 -1  1]
Final output: [-1 -1  1]
Error :  0
Error : 0
iterations: 2
Real Input:  [-1  1  1  1 -1 -1  1 -1 -1  1 -1 -1 -1  1  1]
Final input: [-1  1  1  1 -1 -1  1 -1 -1  1 -1 -1 -1  1  1]
Real Output:  [-1  1 -1]
Final output: [-1  1 -1]
Error :  0
Error : 0
```

شکل 4-2 – خروجی شبکه ساخته شده با 3 حرف

ج)

ورودی های نویزی را به ترتیب با 10 و 40 درصد اعمال می کنیم.

10 درصد نویز:

```
Average accuracy for number of correct pixels of input:  0.9897777777777778
Average acuracy of predicted output: 0.9466666666666667
Average number of iterations: 2
```

شکل 4-3– خروجی با 10 درصد نویز روی شبکه

40 درصد نویز:

```
Average accuracy for number of correct pixels of input:  0.8917777777777778
Average accuracy of predicted output: 0.4733333333333334
Average number of iterations: 3
```

شکل 4-4- خروجی با 40 درصد نویز روی شبکه

مشاهده می شود که با بالاتر رفتن نویز کمی از دقت کم می شود(حدود 10 درصد).

تعداد سه پیمایش طول کشیده و دقت تشخیص هم کمتر شده و علت متفاوت بودن اردر کاهش دقت X , Y این است که با تغییر چند خانه کلاسی که تشخیص داده می شود اشتباه میشود.

د)

نتیجه شبکه روی داده های تست به شکل زیر است که به درستی تشخیص داده شد:

بردار تست = (0,-1,-1)

```
iterations: 4
Real Input:  [-1  1 -1  1 -1  1  1  1  1  1 -1  1  1 -1  1]
Final input: [-1  1 -1  1 -1  1  1  1  1  1 -1  1  1 -1  1]
Real Output:  [-1 -1 -1]
Final output: [-1 -1 -1]
Error :  0
Error of output: 0
```

شکل 4-5– خروجی شبکه با داده تست

ه)

اگر با همه ی حروف A تا H این شبکه را پیاده سازی کنیم ، بردار وزنی به شکل زیر خواهیم داشت:

```
array([[ 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 4-6– بردار وزن ها برای شبکه 8 حرفی

و)

ابتدا نتیجه شبکه جدید را با ورودی بدون نویز خواهیم دید:

```
iterations: 3
Real Input:  [-1  1 -1  1 -1  1  1  1  1  1 -1  1  1 -1  1]
Final input: [ 1  1 -1  1 -1  1  1  1 -1  1 -1  1  1 -1 -1]
Real Output:  [-1 -1 -1]
Final output: [-1 -1  1]
Error :  3
Error of output: 1
iterations: 3
Real Input:  [ 1  1 -1  1 -1  1  1  1 -1  1 -1  1  1  1 -1]
Final input: [ 1  1 -1  1 -1  1  1  1 -1  1 -1  1  1 -1 -1]
Real Output:  [-1 -1  1]
Final output: [-1 -1  1]
Error :  1
Error of output: 0
iterations: 3
Real Input:  [-1  1  1  1 -1 -1  1 -1 -1  1 -1 -1 -1  1  1]
Final input: [-1 -1  1  1 -1 -1  1 -1  1  1 -1 -1 -1  1  1]
Real Output:  [-1  1 -1]
Final output: [ 1  1 -1]
Error :  2
Error of output: 1
iterations: 3
Real Input:  [ 1  1 -1  1 -1  1  1 -1  1  1 -1  1  1  1 -1]
Final input: [ 1 -1 -1  1 -1  1  1 -1  1  1 -1  1  1  1 -1]
Real Output:  [-1  1  1]
Final output: [-1  1  1]
Error :  1
Error of output: 0
iterations: 3
Real Input:  [ 1  1  1  1 -1 -1  1  1 -1  1 -1 -1  1  1  1]
Final input: [-1  1  1  1 -1 -1  1  1 -1  1 -1 -1  1 -1  1]
Real Output:  [ 1 -1 -1]
Final output: [ 1 -1 -1]
Error :  2
Error of output: 0
iterations: 3
Real Input:  [ 1  1  1  1 -1 -1  1  1 -1  1 -1 -1  1 -1 -1]
Final input: [ 1 -1  1  1 -1 -1  1  1 -1  1 -1 -1  1 -1 -1]
Real Output:  [ 1 -1  1]
Final output: [ 1 -1  1]
Error :  1
Error of output: 0
iterations: 3
Real Input:  [-1  1  1  1 -1 -1  1 -1  1  1 -1  1 -1  1  1]
Final input: [-1 -1  1  1 -1 -1  1 -1  1  1 -1 -1 -1  1  1]
Real Output:  [ 1  1 -1]
Final output: [ 1  1 -1]
Error :  2
Error of output: 0
iterations: 3
Real Input:  [ 1 -1  1  1 -1  1  1  1  1  1 -1  1  1 -1  1]
Final input: [ 1 -1  1  1 -1  1  1  1 -1  1 -1 -1  1 -1 -1]
Real Output:  [1 1 1]
Final output: [ 1 -1  1]
Error :  3
Error of output: 1
```

شکل 4-7– خروجی شبکه با 8 حرف به ورودی بدون نویز

شبکه در به حافظه سپاری ورودی های درست خطا دارد و در نصف موارد کلاس را اشتباه تشخیص می دهد.

```
Average accuracy for number of correct pixels of input:  0.97
Average acuracy of predicted output: 0.8
Average number of iterations: 2
```

شکل 4-8– خروجی شبکه 8 حرفی با 10 درصد نویز

```
Average accuracy for number of correct pixels of input:  0.9410833333333334
Average accuracy of predicted output: 0.73875
Average number of iterations: 2
```

شکل 4-9– خروجی شبکه 8 حرفی با 40 درصد نویز

مشاهده می شود با اضافه شدن نویز اندکی دقت شبکه در تشخیص خانه های درست کم می شود که تاثیر زیادی در تشخیص کلاس ها نیز دارد زیراکه تعداد کلاس ها و اشکال شبیه به هم زیاد شده است.

می توان نتیجه گرفت که 8 پترن را نمیتوان همزمان به حافظه سپرد و حتی بدون نویز هم دارای مشکل است. اما شبکه 2 حرفی به دقت بسیار خوبی می رسد .

```
iterations: 2
Real Input:  [-1  1 -1  1 -1  1  1  1  1  1 -1  1  1 -1  1]
Final input: [-1  1 -1  1 -1  1  1  1  1  1 -1  1  1 -1  1]
Real Output:  [-1 -1 -1]
Final output: [-1 -1 -1]
Error of:  0
Error of output: 0
iterations: 2
Real Input:  [ 1  1 -1  1 -1  1  1  1 -1  1 -1  1  1  1 -1]
Final input: [ 1  1 -1  1 -1  1  1  1 -1  1 -1  1  1  1 -1]
Real Output:  [-1 -1  1]
Final output: [-1 -1  1]
Error of:  0
Error of output: 0
```

شکل 4-10– خروجی شبکه 2 حرفی بدون نویز(ساده)

```
Average accuracy for number of correct pixels of input:  1.0
Average acuracy of predicted output: 1.0
Average number of iterations: 2
```

شکل 4-11– خروجی شبکه 2 حرفی با 10 درصد نویز

```
Average accuracy for number of correct pixels of input:  0.883
Average accuracy of predicted output: 0.565
Average number of iterations: 3
```

شکل 4-12– خروجی شبکه 2 حرفی با 40 درصد نویز

علت این محدودیت کوچک بودن خانه ها می باشد که با نمایش دادن توسط تعداد بیت های بیشتر قابل حل است. همچنین این شبکه به نسبت ضعیف است و اگر تقویت شود (با ورودی بزرگتر) ، عملکرد بهتری خواهد داشت.

ز)

جدول 2- فاصله Hamming بین کاراکتر ها

	A	B	C	D	E	F	G	H
A		4	7	4	6	6	5	3
B	4		7	2	4	4	7	5
C	7	7		7	3	5	2	8
D	4	2	7		5	6	5	5
E	6	4	3	5		2	5	5
F	6	4	5	6	2		7	5
G	5	7	2	5	5	7		6
H	3	5	8	5	5	5	6	

مشاهده می شود که حروف A,B,C,H به نسبت قابل تشخیص هستند و بهترین انتخاب می باشند. بدترین انتخاب E است زیرا شباهت زیادی به بقیه حروف مثل F دارد. بیشترین احتمال انتخاب نیز برای C است.