به نام خدا

دانشگاه تهران

دانشکده مهندسی برق و کامپیوتر

شبکه های عصبی و یادگیری عمیق

تمرین سری ۳

نام و نام خانوادگی	امید واهب
شماره دانشجویی	۸۱۰۱۹۶۵۸۲
تاریخ ارسال گزارش	۱۴۰۰/۰۳/۱۲

فهرست گزارش سوالات

سوال ۱ – شناسایی حروف با استفاده از روش هب

در این سوال یک شبکه ساده را با قانون Hebbian آموزش می دهیم تا ۳ حرف A، B و C را از هم تمییز دهد. فایل حاوی کد های این سوال در HW3_Q1.ipynb موجود است.

الف) ماتریس W را یک ماتریس ۶۳ در ۱۵ به درایه های ۲فر فرض کردیم و با قانون Hebbian و یک تابع Sign آن را مقداردهی کردیم سپس به ازای سه ورودی مدنظر تست کردیم و هیچ خطایی نداشتیم. در شکل زیر به ترتیب نتایج A، B و C را نمایش می دهیم:

```
Original Output:
[-1  1 -1  1 -1  1  1  1  1  1 -1  1  1 -1  1]
Model Output:
[-1  1 -1  1 -1  1  1  1  1  1 -1  1  1 -1  1]
Error = 0
Original Output:
[ 1  1 -1  1 -1  1  1  1 -1  1 -1  1  1  1 -1]
Model Output:
[ 1  1 -1  1 -1  1  1  1 -1  1 -1  1  1  1 -1]
Error = 0
Original Output:
[-1  1  1  1 -1 -1  1 -1 -1  1 -1 -1 -1  1  1]
Model Output:
[-1  1  1  1 -1 -1  1 -1 -1  1 -1 -1 -1  1  1]
Error = 0
```

شکل ۱ – نتایج مدل بدون نویز و اغتشاش

مشاهده می شود که همه ی حروف بدون هیچ خطایی ساخته می شوند.

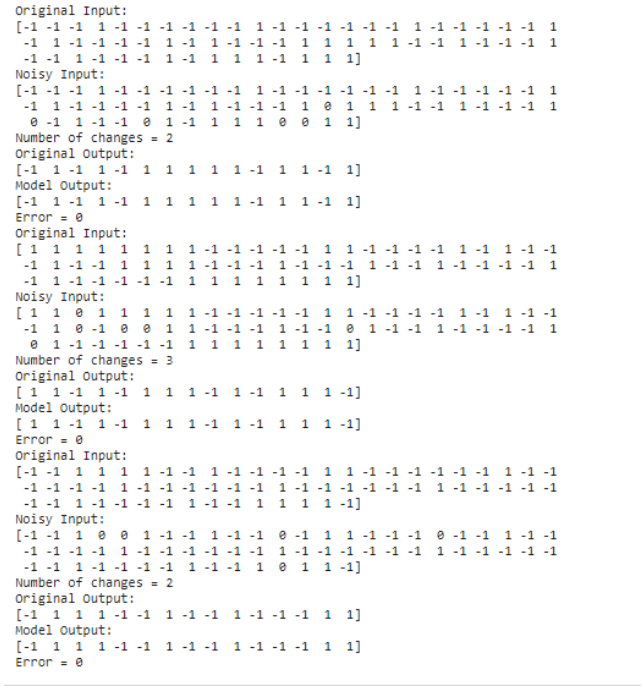
ب) حال اثر نویز و اغتشاش را روی ورودی ها می بینیم:

```
Original Input:
[-1 -1 -1  1 -1 -1 -1 -1 -1 -1  1 -1 -1 -1 -1 -1 -1  1 -1 -1 -1 -1 -1  1
 -1  1 -1 -1 -1 -1 -1  1 -1  1 -1 -1 -1 -1  1  1  1  1 -1 -1  1 -1 -1 -1  1
 -1 -1  1 -1 -1 -1 -1  1 -1  1  1  1 -1 -1  1  1  1]
Noisy Input:
[-1 -1 -1  1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1  1  1 -1 -1 -1 -1  1
 -1  1 -1 -1 -1 -1 -1  1 -1  1 -1 -1 -1 -1  1  1 -1  1 -1 -1 -1  1  1  1 -1 -1
 -1 -1  1 -1 -1 -1 -1  1 -1  1 -1  1  1  1  1  1]
Number of changes = 9
Original Output:
[-1  1 -1  1 -1  1  1  1  1 -1  1  1 -1  1]
Model Output:
[-1  1 -1  1 -1  1  1  1  1 -1  1  1 -1  1]
Error = 0
Original Input:
[ 1  1  1  1  1  1  1  1 -1 -1 -1 -1 -1  1  1 -1 -1 -1 -1  1 -1  1 -1 -1
 -1  1 -1 -1  1  1  1  1 -1 -1 -1  1 -1 -1 -1 -1  1 -1 -1  1 -1 -1 -1 -1  1
 -1  1 -1 -1 -1 -1 -1 -1  1  1  1  1  1  1  1  1]
Noisy Input:
[ 1  1 -1  1  1  1  1  1 -1 -1 -1 -1 -1 -1  1 -1 -1 -1  1  1  1 -1  1 -1 -1
 -1  1  1 -1  1  1  1  1 -1 -1 -1 -1 -1 -1  1 -1  1 -1 -1 -1  1 -1 -1 -1 -1  1
 -1  1 -1 -1 -1 -1 -1 -1  1  1 -1  1 -1  1  1  1]
Number of changes = 8
Original Output:
[ 1  1 -1  1 -1  1  1  1 -1  1 -1  1  1  1 -1]
Model Output:
[ 1  1 -1  1 -1  1  1  1 -1  1 -1  1  1  1 -1]
Error = 0
Original Input:
[-1 -1  1  1  1  1 -1 -1  1 -1 -1 -1 -1  1  1 -1 -1 -1 -1 -1  1 -1 -1
 -1 -1 -1 -1  1 -1 -1 -1 -1 -1 -1 -1  1 -1 -1 -1 -1 -1 -1 -1  1 -1 -1 -1 -1 -1
 -1 -1  1 -1 -1 -1 -1 -1  1 -1 -1  1  1  1  1 -1]
Noisy Input:
[-1 -1  1  1  1  1 -1 -1  1 -1 -1 -1 -1  1  1 -1 -1 -1 -1 -1  1 -1 -1
 -1 -1 -1 -1  1 -1 -1 -1 -1 -1 -1 -1  1 -1 -1 -1 -1 -1 -1 -1  1 -1 -1 -1 -1 -1
  1 -1  1 -1 -1 -1 -1 -1 -1 -1  1  1  1  1 -1]
Number of changes = 5
Original Output:
[-1  1  1  1 -1 -1  1 -1 -1  1 -1 -1 -1  1  1]
Model Output:
[-1  1  1  1 -1 -1  1 -1 -1  1 -1 -1 -1  1  1]
Error = 0
```

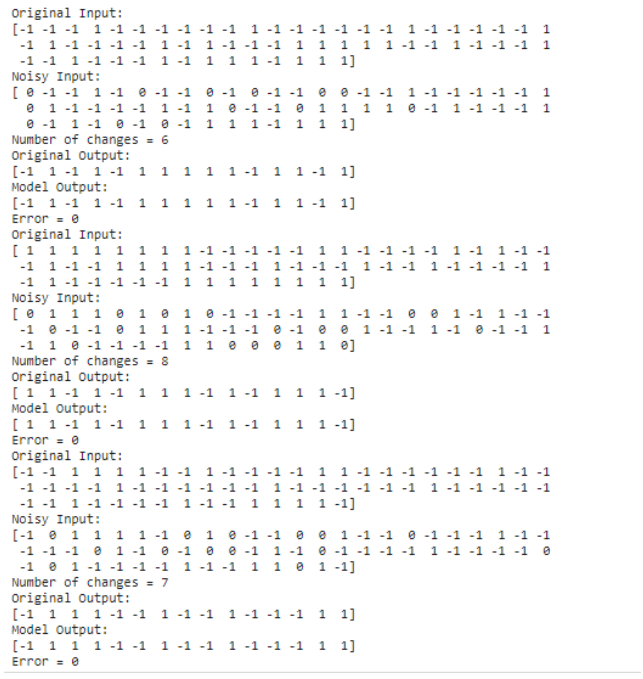
شکل ۲ – نتایج مدل به ازای ۱۰ درصد نویز

```
Original Input:
[-1 -1 -1  1 -1 -1 -1 -1 -1  1 -1 -1 -1 -1 -1  1 -1 -1 -1 -1 -1  1
 -1  1 -1 -1 -1 -1  1 -1  1 -1 -1 -1  1  1  1 -1 -1 -1  1 -1 -1 -1  1
 -1 -1  1 -1 -1 -1  1 -1  1  1  1 -1  1  1  1]
Noisy Input:
[-1 -1 -1  1 -1 -1 -1  1 -1 -1 -1 -1 -1  1 -1 -1 -1  1 -1 -1  1 -1  1  1
 -1  1 -1 -1 -1 -1  1  1  1 -1 -1 -1  1  1  1 -1  1 -1 -1 -1 -1 -1 -1  1
 -1 -1  1 -1 -1 -1  1 -1 -1 -1  1 -1  1  1 -1]
Number of changes = 11
Original Output:
[-1  1 -1  1 -1  1  1  1  1 -1  1  1 -1  1]
Model Output:
[-1  1 -1  1 -1  1  1  1  1 -1  1  1 -1  1]
Error = 0
Original Input:
[ 1  1  1  1  1  1  1  1 -1 -1 -1 -1 -1  1  1 -1 -1 -1 -1  1 -1  1 -1 -1
 -1  1 -1 -1  1  1  1  1 -1 -1 -1  1 -1 -1 -1  1 -1 -1  1 -1 -1 -1 -1  1
 -1  1 -1 -1 -1 -1 -1 -1  1  1  1  1  1  1  1  1]
Noisy Input:
[ 1  1  1  1  1  1  1  1  1 -1 -1 -1 -1  1  1 -1  1 -1 -1  1  1 -1 -1 -1
 -1  1 -1  1  1  1 -1  1 -1 -1 -1 -1  1 -1 -1  1  1 -1 -1 -1 -1 -1 -1 -1
 -1 -1 -1 -1 -1  1 -1  1  1  1 -1  1  1  1 -1]
Number of changes = 12
Original Output:
[ 1  1 -1  1 -1  1  1  1 -1  1 -1  1  1  1 -1]
Model Output:
[ 1  1 -1  1 -1  1  1  1 -1  1 -1  1  1  1 -1]
Error = 0
Original Input:
[-1 -1  1  1  1  1 -1 -1  1 -1 -1 -1 -1  1  1 -1 -1 -1 -1 -1  1 -1 -1
 -1 -1 -1 -1  1 -1 -1 -1 -1 -1 -1 -1  1 -1 -1 -1 -1 -1 -1 -1  1 -1 -1 -1 -1 -1
 -1 -1  1 -1 -1 -1 -1 -1  1 -1 -1  1  1  1  1 -1]
Noisy Input:
[-1 -1  1  1  1  1 -1  1  1  1 -1  1 -1  1  1  1 -1 -1 -1  1 -1  1 -1 -1
  1 -1  1 -1  1 -1 -1 -1 -1 -1 -1  1  1 -1 -1 -1  1 -1 -1 -1 -1 -1  1 -1
 -1 -1  1 -1 -1 -1 -1 -1 -1 -1 -1  1  1  1  1 -1]
Number of changes = 12
Original Output:
[-1  1  1  1 -1 -1  1 -1 -1  1 -1 -1 -1  1  1]
Model Output:
[-1  1  1  1 -1 -1  1 -1 -1  1 -1 -1 -1  1  1]
Error = 0
```

شکل ۳ – نتایج مدل به ازای ۲۵ درصد نویز



شکل ۴ – نتایج مدل به ازای ۱۰ درصد اغتشاش



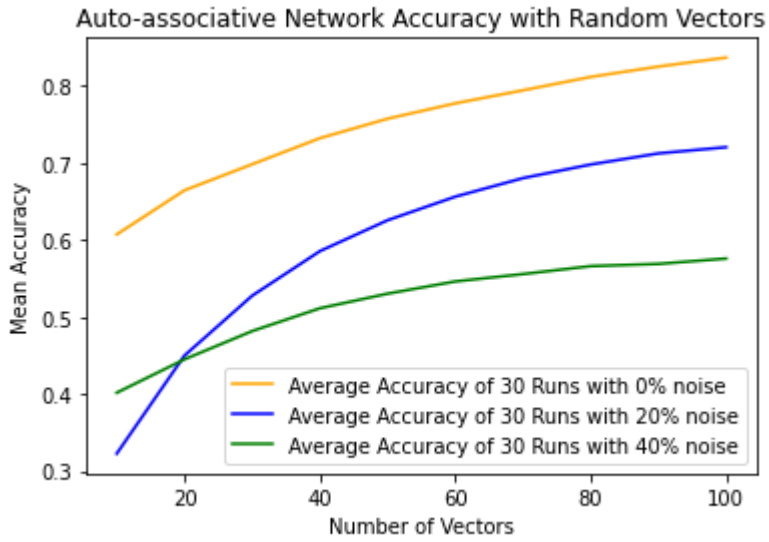
شکل ۵ – نتایج مدل به ازای ۲۵ درصد اغتشاش

می توان مشاهده کرد که در تمامی حالات بدون هیچ خطایی مدل حروف را از هم تشخیص داده و خروجی های مدنظر را ساخته است. می توان علت این اتفاق را متفاوت بودن شکل کلی حروف دانست همچنین تعداد کلاس ها نیز کم بود و ممکن بود در صورت افزایش تعداد خطا داشته باشیم. در ۱۰۰ درصد مواقع به جواب درست رسیدیم(البته در یکی از اجراها یک خطا داشتیم ولی در مابقی هیچ خطایی نبود).

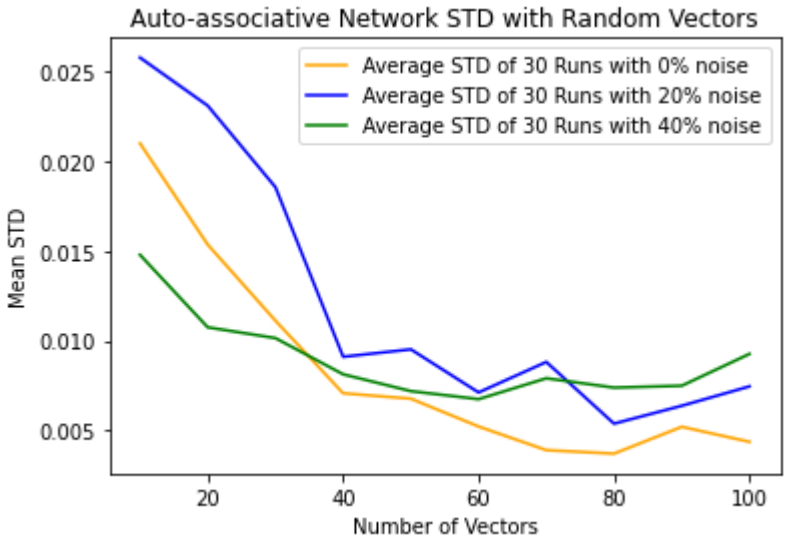
سوال ۲- شبکه خودانجمنی

در این سوال یک شبکه ی auto-associative می سازیم که تعدادی بردار با مقادیر bipolar و رندوم را به یاد بسپارد سپس عملکرد آن را در حالات مختلف با نویزها و تعداد بردار های متفاوت می سنجیم. کد این سوال در فایل HW3_Q2.ipynb موجود است.

الف) کد خواسته شده را پیاده سازی می کنیم و به کمک قانون هب تغییر یافته ماتریس وزن ها را تشکیل می دهیم. نمودار های خواسته شده به شرح زیر هستند:



شکل ۶ – دقت شبکه خودانجمنی با بردار های رندوم



شکل ۷ – انحراف از معیار شبکه خودانجمنی با بردار های رندوم

ب) می توان دید که میانگین و انحراف از معیار دقت در داده های بدون نویز بهتر بوده است که مورد انتظار ما هم بود. همچنین هر چقدر تعداد بردار ها افزایش یافته اند نیز دقت افزوده شده است یعنی از حد تعداد پترن هایی که شبکه ما می تواند به یاد بسپرد عبور نکرده ایم. افزایش در دقت نیز به این دلیل بوده است که با افزودن تعداد بردار ها از توانایی شبکه در به یاد سپاری بیشتر استفاده شده ست.

ج) همانطور که در حین درس مشاهده شد ظرفیت این شبکه ها به شرح زیر است:

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 ۸ – ظرفیت شبکه ها

می توان دید که ظرفیت شبکه ما ۹۹ بردار بوده است 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 ها است زیرا همانطور که می توان مشاهده کرد این مقادیر برای ورودی های عمود بر هم است و اگر کاملا بر هم عمود بودند در حالت بدون نویز به ۱۰۰ درصد دقت می رسیدیم با این تعداد بردار ورودی اما بردار ها به صورت رندوم مقداردهی شده اند به همین دلیل لزوما همگی بر هم عمود نیستند ولی به حدی بوده اند که شبکه بتواند دقت نزدیک ۸۰ درصد را بگیرد.

سوال ۳ – شبکه هاپفیلد

در این سوال به طراحی شبکه ی 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 انجام می دهیم سپس به کمک الگوریتم گفته شده ورودی های دارای نویز را به مقادیر اصلی شان میرسانیم.

کد این قسمت در فایل HW3_Q3.ipynb موجود است.

الف) به کمک قانون هب ماتریس وزن ها را مقداردهی اولیه می کنیم تا دو عدد مدنظر در شبکه ذخیره شوند.

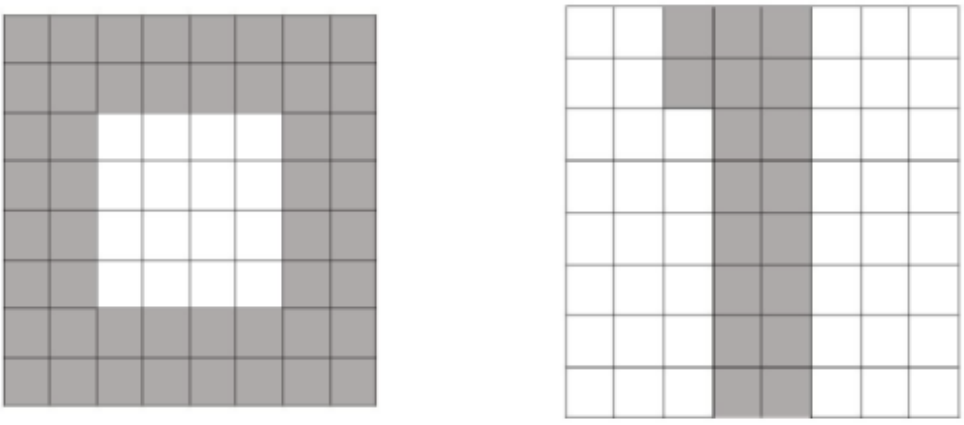
ب) حال نویزی با احتمال ۳۰ درصد به دو عدد ذخیره شده ی صفر و یک اضافه می کنیم که بالای ۲۰ تا از ورودی ها را دچار تغییر می کند سپس بردار حاصل را به عنوان ورودی به شبکه هاپفیلد می دهیم و با ۳ iteration توسط این شبکه آموزش داده شده ورودی های دارای اشتباه را بدون هیچ خطایی به مقادیر اصلی شان میرسانیم. تمامی این اطلاعات در شکل زیر قابل مشاهده اند به این صورت که نصفه ی اول اطلاعات برای عدد صفر و نصفه ی دوم برای عدد یک هستند:

```
Number of changes in input signal: 22
Number of iterations: 3
Data without noie: [ 1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1 -1 -1 -1 -1  1  1
  1  1 -1 -1 -1 -1  1  1  1  1 -1 -1 -1 -1  1  1  1  1 -1 -1 -1 -1  1  1
  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1]
Output of Model  : [ 1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1 -1 -1 -1 -1  1  1
  1  1 -1 -1 -1 -1  1  1  1  1 -1 -1 -1 -1  1  1  1  1 -1 -1 -1 -1  1  1
  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1]
Error :  0
Number of changes in input signal: 29
Number of iterations: 3
Data without noie: [-1 -1  1  1  1 -1 -1 -1 -1 -1  1  1  1 -1 -1 -1 -1 -1 -1  1  1 -1 -1 -1
 -1 -1 -1  1  1 -1 -1 -1 -1 -1 -1  1  1 -1 -1 -1 -1 -1 -1 -1  1  1 -1 -1 -1
 -1 -1 -1  1  1 -1 -1 -1 -1 -1 -1  1  1 -1 -1 -1]
Output of Model  : [-1 -1  1  1  1 -1 -1 -1 -1 -1  1  1  1 -1 -1 -1 -1 -1 -1  1  1 -1 -1 -1
 -1 -1 -1  1  1 -1 -1 -1 -1 -1 -1  1  1 -1 -1 -1 -1 -1 -1 -1  1  1 -1 -1 -1
 -1 -1 -1  1  1 -1 -1 -1 -1 -1 -1  1  1 -1 -1 -1]
Error :  0
```

شکل ۹ – نتیجه ی تست شبکه هاپفیلد با داده های دارای نویز ۳۰ درصد

می توان دید که بدون هیچ خطایی و تنها در سه iteration ورودی هایی با ۲۹ تغییر را به ورودی اصلی رسانده ایم.

ج) حال Hamming Distance بین کاراکتر های صفر و یک داده شده را محاسبه می کنیم:



شکل ۱۰ – کاراکتر های صفر و یک داده شده

می توان محاسبه کرد که فاصله ی hamming این دو ورودی برابر ۴۶ است و Average Hamming Distance نیز برابر 46/64 تقریبا ۷۲٪ است که مقدار خیلی زیادی است و نشان می دهد فاصله ی این و کاراکتر از هم خیلی زیاد است و تغییر در چند پیکسل نمی تواند ما را به اشتباه بیندازد و همچنان قابل تمییز از یکدیگر هستند به عبارت دیگر می توان گفت وابستگی به هم ندارند و تنها در ۱۸ پیکسل از ۶۴ پیکسل کلی با هم اشتراک دارند که اکثر این اشتراکات نیز در مرکز تصاویر جمع شده اند و با تمرکز بر حاشیه ها می توان به خوبی آن ها را از هم جدا کرد.

سوال ۴ – شبکه BAM

در این سوال به طراحی شبکه ی 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 انجام می دهیم سپس به کمک الگوریتم گفته شده ورودی های دارای نویز را به مقادیر اصلی شان میرسانیم.

کد این قسمت در فایل HW3_Q4.ipynb موجود است.

الف) ابتدا تنها با حروف A، B و C شبکه را می سازیم و آموزش می دهیم و ماتریس وزن های زیر به دست می آید:

```
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 ۱۱ – ماتریس وزن های شبکه با سه حرف

ب) حال توانایی شبکه را بررسی می کنیم و مشاهده می شود که بدون هیچ مشکلی کار می کند:

```
Number of 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 input:  0
Error of output: 0
Number of 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 input:  0
Error of output: 0
Number of 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 input:  0
Error of output: 0
```

شکل ۱۲ – نتیجه ی بازیابی اطلاعات به کمک شبکه آموزش دیده با ۳ حرف

در دو iteration به مقادیر مدنظر میرسیم و خطایی نیز نداریم.

ج) حال ورودی ها را با نویز اعمال می کنیم به ترتیب نتیجه ی نویز ۱۰ درصد و ۴۰ درصد به شرح زیر است:

```
Average accuracy of number of correct pixels of input:  0.99
Average acuracy of predicted output: 0.95
Average number of iterations: 2
```

شکل ۱۳ – نتیجه بازیابی اطلاعات با ۱۰ درصد نویز توسط شبکه

```
Average accuracy of number of correct pixels of input:  0.8768888888888888
Average accuracy of predicted output: 0.42666666666666664
Average number of iterations: 3
```

شکل ۱۴ – نتیجه بازیابی اطلاعات با ۴۰ درصد نویز توسط شبکه

می توان مشاهده کرد که زمانی که نویز بالاتر رفته خطای شبکه در تشخیص کلاس ورودی خیلی کاهش داشته است همچنین به جای دو iteration سه تا طول کشیده است و دقت تشخیص پیکسل ها هم کمتر شده است و علت متفاوت بودن اردر کاهش دقت X و Y این است که با تغییر چند پیکسل کلاسی که شخیص داده می شود، عوض می شود و در محاسبه خطا اثرگذار است.

د) حال بررسی می کنیم اگر (0, -1, -1) را به شبکه بدهیم توانایی تشخیص دارد که نتیجه ی مدنظر مشاهده می شود:

```
Number of 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 of input:  0
Error of output: 0
```

شکل ۱۴ – نتیجه تشخیص شبکه با ورودی (0, -1, -1)

شبکه به درستی تشخیص داده است اما اگر در حالتی بودیم که ۸ حرف داشتیم نمی توانست زیرا هم می توانست (-1, -1, -1) و هم (1, -1, -1) تشخیص داده شود وی در این حالت مشکلی در تشخیص نداریم.

ه) حال با ۸ حرف اقدام به پیاده سازی شبکه می کنیم و بردار وزن ها به شرح زیر می شود:

```
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 ۱۵ – بردار وزن ها برای شبکه با ۸ حرف

و) شبکه در به حافظه سپاری ورودی های درست اندکی مشکل دارد و در نصف موارد کلاس را اشتباه تشخیص داده و تعداد کمی از پیکسل ها را نیز اشتباه تشخیص داده است:

```
Number of 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 of input:  1
Error of output: 0
Number of 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 input:  0
Error of output: 0
Number of 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 of input:  7
Error of output: 1
Number of 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 of input:  2
Error of output: 1
```

```
Number of 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 of input:  2
Error of output: 0
Number of 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 of input:  1
Error of output: 0
Number of 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 of input:  1
Error of output: 1
Number of 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 of input:  4
Error of output: 2
```

شکل ۱۶ – نتیجه شبکه با ۸ حرف به ورودی بدون نویز

```
Average accuracy of number of correct pixels of input:  0.9715
Average acuracy of predicted output: 0.625
Average number of iterations: 2
```

شکل ۱۷ – عملکرد شبکه با ۸ حرف به ورودی با ۱۰ درصد نویز

```
Average accuracy of number of correct pixels of input:  0.9456666666666667
Average accuracy of predicted output: 0.73375
Average number of iterations: 3
```

شکل ۱۸ – عملکرد شبکه با ۸ حرف به ورودی با ۴۰ درصد نویز

مشاهده می شود که با اضافه شدن نویز اندکی دقت شبکه در تشخیص پیکسل ها کاهش یافته ولی هنوز هم قابل قبول است اما می توان دید که این اشتباه در تشخیص پیکسل در تشخیص کلاس ها اثر زیادی داشته است و اکنون که تعداد کلاس ها و اشکال شبیه هم زیادتر شده است دقت تشخیص کلاس خیلی کمتر شده است.

می توان نتیجه گرفت که ۸ پترن به هیچ وجه قابل به حافظه سپاری نیست و حتی بدون نویز هم دارای مشکلاتی است حال آنکه با نویز مشکلات زیادتر هم می شوند. حال شبکه با دو حرف را مشاهده می کنیم که به درصد خیلی خوبی میرسد و می توان گفت کاملا از هم تشخیص می دهد البته با توجه به تفاوت شکل حروف از یکدیگر می توان دیتاست هایی با ۳ و یا حتی ۴ حرف هم تشکیل دهیم و کاملا به حافظه بسپاریم.

```
Number of 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 input:  0
Error of output: 0
Number of 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 input:  0
Error of output: 0
```

شکل ۱۹ – نتیجه شبکه با ۲ حرف به ورودی بدون نویز

```
Average accuracy of number of correct pixels of input:  0.9986666666666667
Average acuracy of predicted output: 0.995
Average number of iterations: 2
```

شکل ۲۰ – عملکرد شبکه با ۲ حرف به ورودی با ۱۰ درصد نویز

```
Average accuracy of number of correct pixels of input:  0.99
Average accuracy of predicted output: 0.96
Average number of iterations: 3
```

شکل ۲۱ – عملکرد شبکه با ۲ حرف به ورودی با ۴۰ درصد نویز

علت این محدودیت کوچک بودن پیکسل های ما است و البته بهتر بود اعداد را با تعداد بیت های بیشتری نمایش میدادیم یا حداقل کلاس هایی با بیت متفاوت زیادتر را به اعداد با اشکال شبیه به هم می دادیم. نکته دیگر هم محدودیت این شبکه است و می توانستیم با شبکه های قوی تر و ورودی های بزرگتر نتایج خیلی بهتری بگیریم.

ز) حال hamming distance را برای جفت اعداد مختلف محاسبه می کنیم:

جدول ۱ – فاصله ی hamming بین حروف

	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	

جدول ۲ – وضعیت فاصله ی حروف

	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	

می توان مشاهده کرد که حروف A، B، C و H تا حد خوبی (به جز A و H) از هم قابل تشخیص هستند و می توانند بهترین انتخاب باشند اگر تنها ۴ حرف را بخواهیم انتخاب کنیم. در بین این حروف E و F خیلی به هم شبیه هستند و کلا کاراکتر E احتمال اینکه تشخیص داده شود کمترین و کاراکتر C بیشترین است.