به نام خدا

دانشگاه تهران

دانشکده مهندسی برق و کامپیوتر

درس شبکه‌های عصبی و یادگیری عمیق

تمرین سری چهارم

نام و نام خانوادگی	مریم ریاضی
شماره دانشجویی	810197518
تاریخ ارسال گزارش	1401/3/24

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

سوال ۱ – SOM

قسمت الف:

ماتریس وزن ها باید ماتریسی به ابعاد 784 در 225 باشد. یعنی ماتریسی با 784 سطر و 225 ستون.

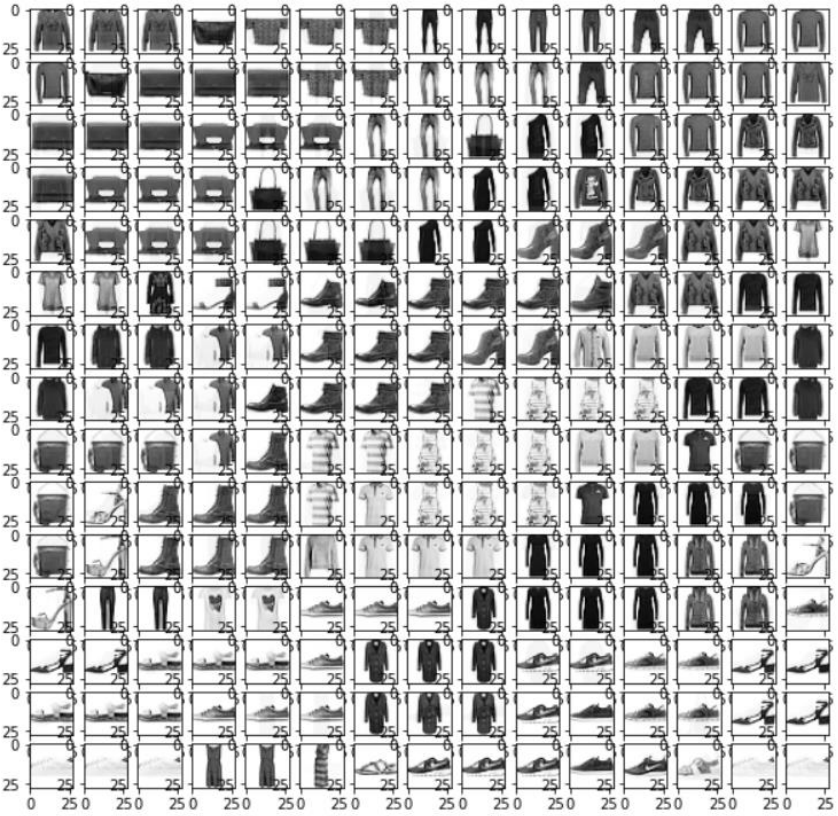
نحوه چینش نورون ها در میزان بهبود شبکه و وزن ها تاثیر می گذارد. علت این موضوع این است که وقتی نورون ها را با چینش مربع یا 6 ضلعی و یا هر شکل دیگری در نظر می گیریم، در هر مرحله تعداد نورون هایی که همکاری می کنند و تعداد نورون هایی که رقابت می کنند تغییر می کند. به همین دلیل است که نحوه چینش نورون ها اهمیت بسیار زیادی دارد زیرا این نحوه چینش و شعاعی که در نظر می گیریم، در تعداد این نورون های همکار و رقیب تاثیر می گذارد.

قسمت ب:

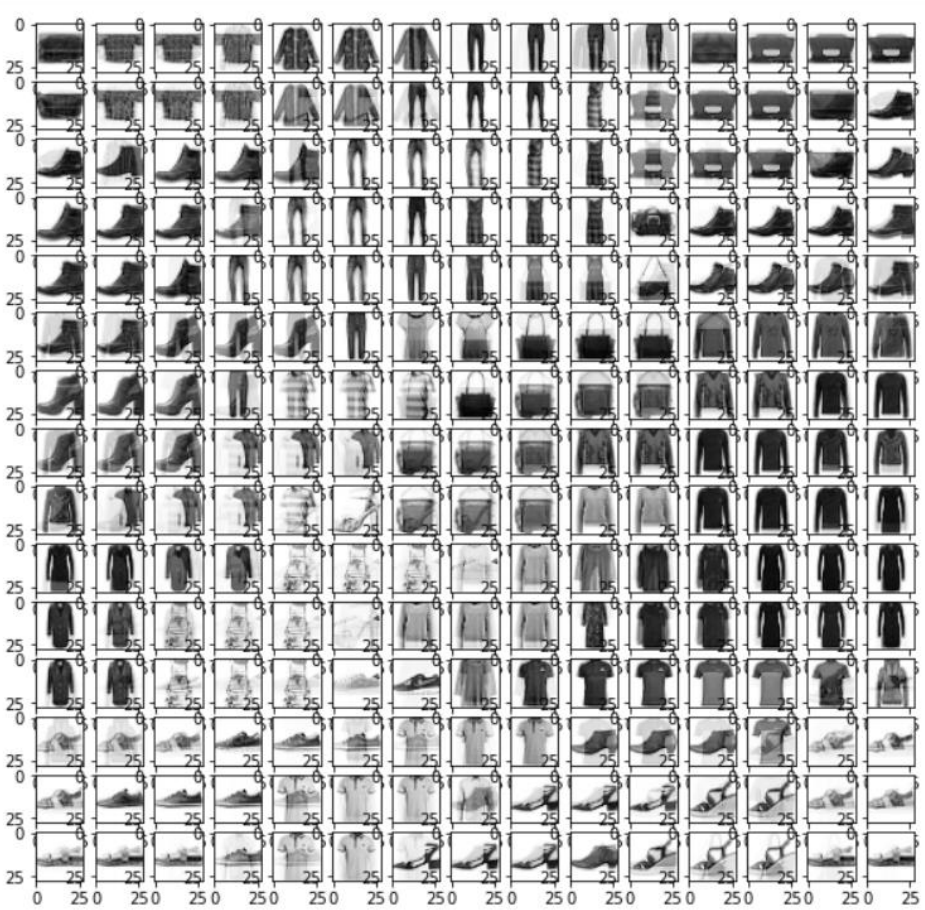
در این قسمت ابتدا ماتریس وزن ها به صورت اعداد رندوم بین 0 و 1 مقداردهی شد. در ادامه در 10 ایپاک به ازای تمام ورودی ها در هر ایپاک، ماتریس وزن ها را به روز کردم. در نهایت برای ایپاک های 1و 2و5و10 ماتریس وزن ها را نگه داشتم که در ادامه تصاویر هر کدام را جداگانه رسم کنم و نحوه پیشرفت شبکه را مشاهده کنم.

در این قسمت مقدار alpha = 0.9 در ابتدا در نظر گرفته شد و در انتهای هر ایپاک نیز در 0.7 ضرب می شود.

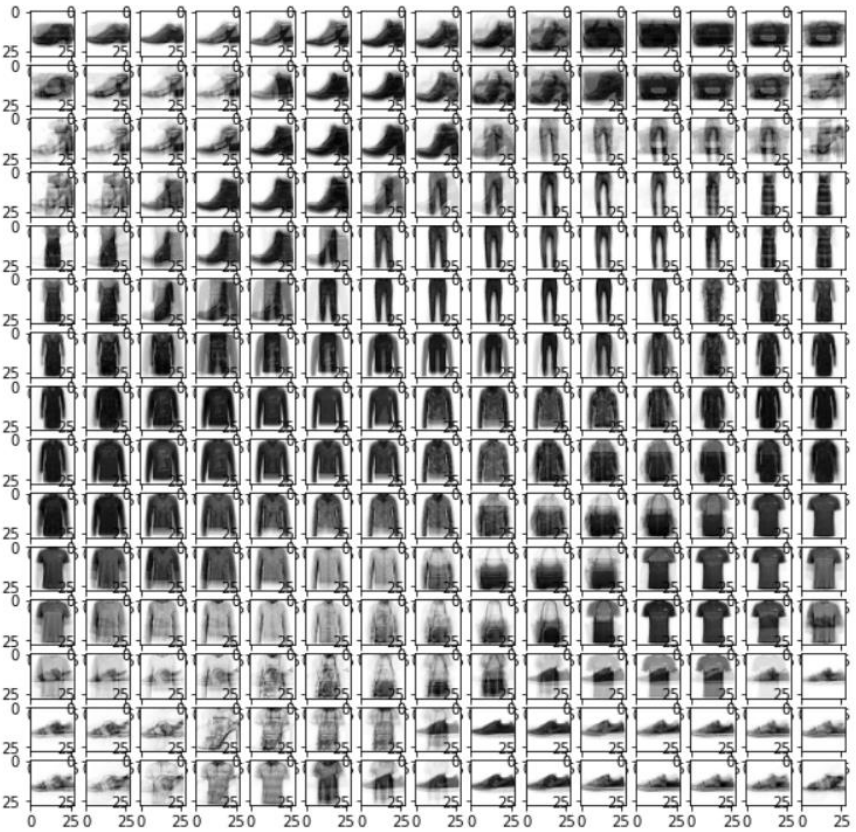
تصاویر نهایی به صورت زیر هستند:



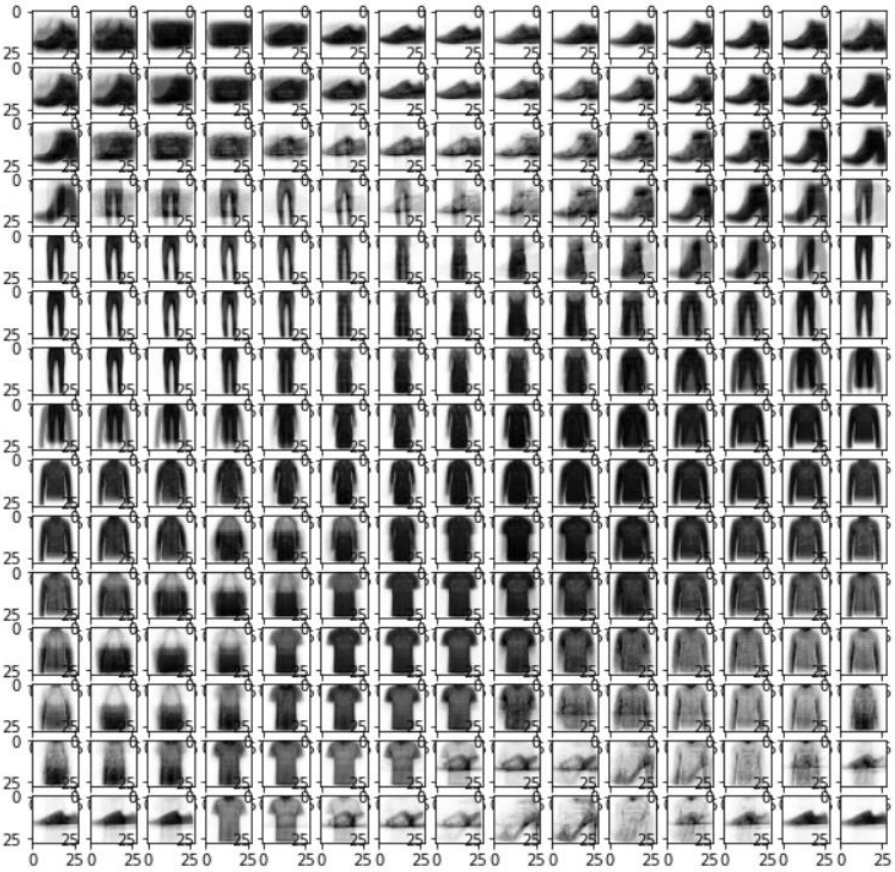
تصاویر وزن ها در ایپاک 1



تصاویر وزن ها در ایپاک 2

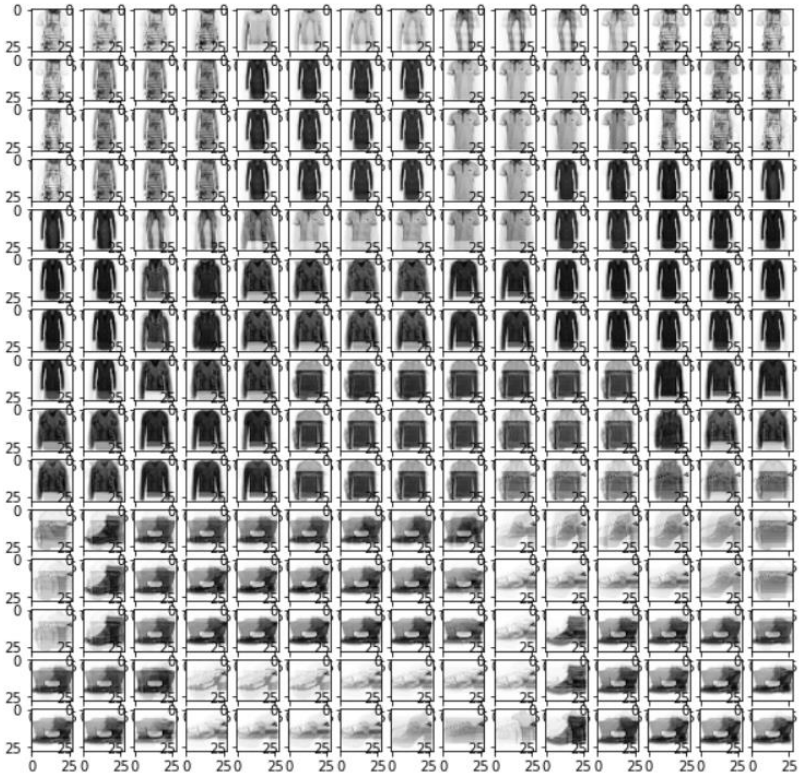


تصاویر وزن ها در ایپاک 5

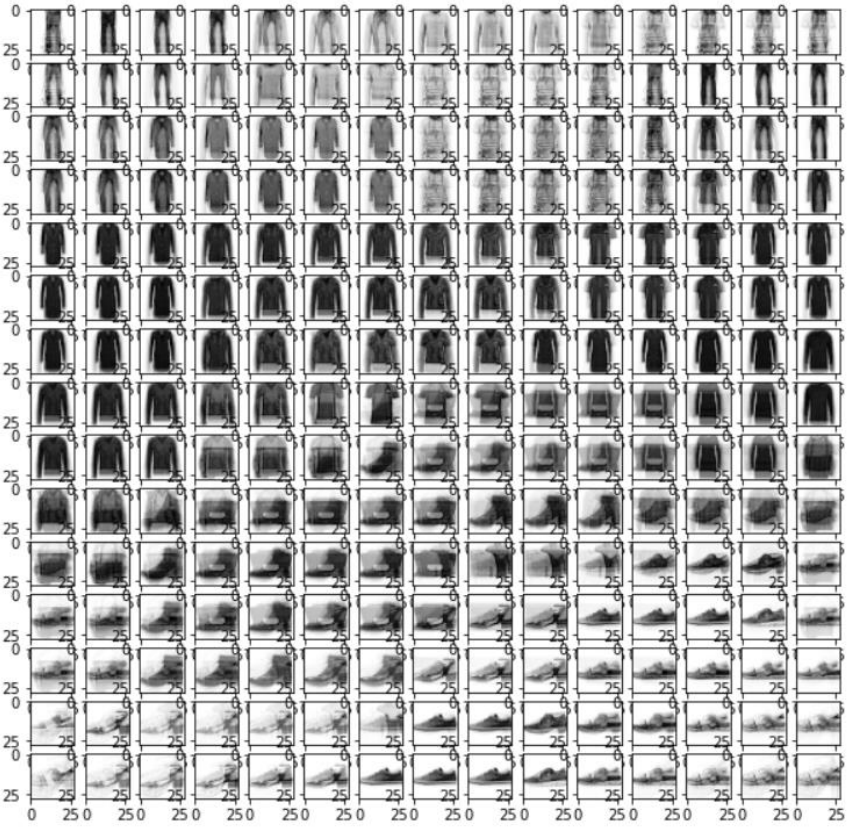


تصاویر وزن ها در ایپاک 10

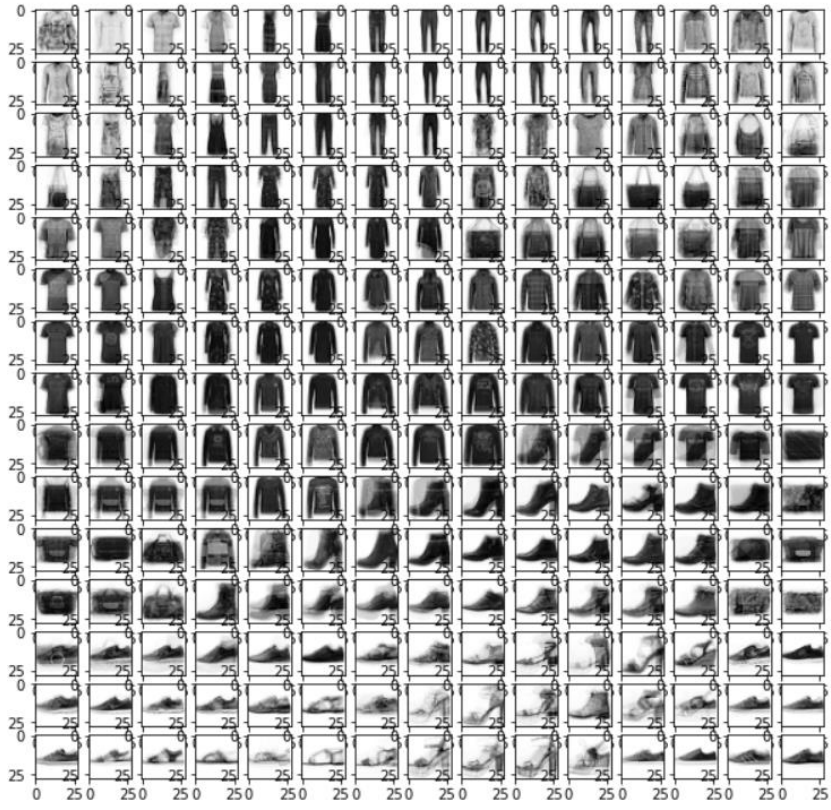
قسمت پ:



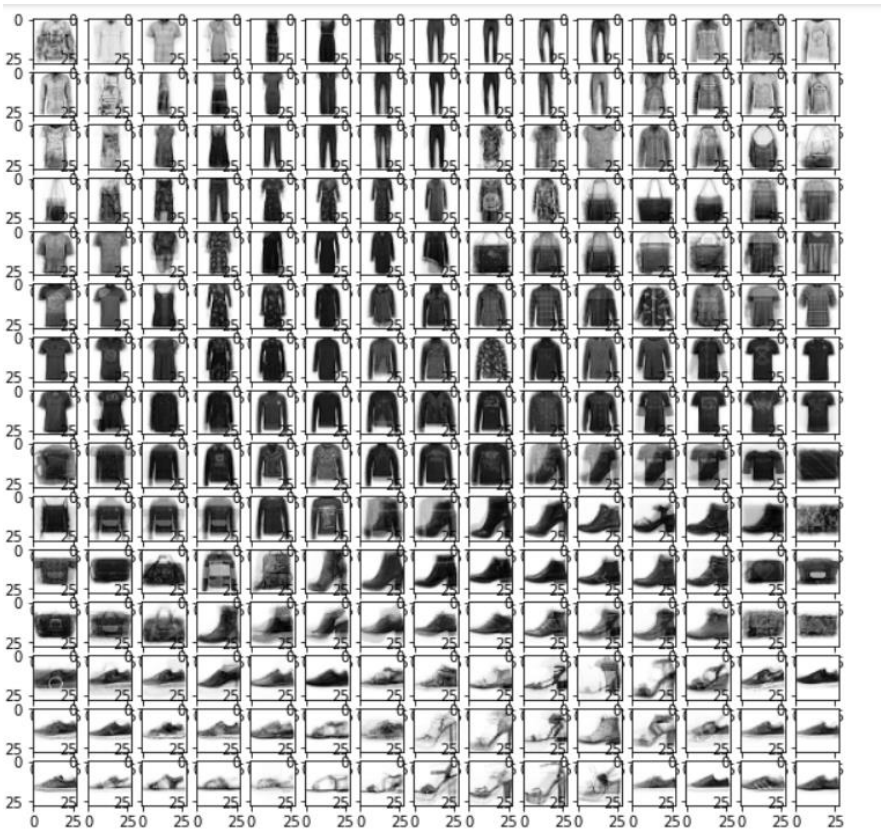
تصاویر وزن ها در ایپاک 1



تصاویر وزن ها در ایپاک 2



تصاویر وزن ها در ایپاک 5



تصاویر وزن ها در ایپاک 10

در این قسمت مقدار alpha = 0.5 در نظر گرفته شده است.

قسمت ت:

با مقایسه قسمت ب و پ می بینیم که نتایج در قسمت پ بهتر شده اند و این موضوع به این دلیل است که شعاع در هر ایپاک کمتر شده است. در ابتدا که شعاع زیا د است نورون های همکار زیاد هستند و همگی آنها تلاش می کنند تا به مرکزیت برسند. وقتی شعاع کم کم کاهش پیدا می کند تعداد نورون های همکار کمتر شده و رقیب یکدییگر می شوند. این خاصیت باعث می شود که شبکه سریعتر به کلاسترهای متمرکز برسد و سریعتر خود را بهبود دهد.

سوال ۲ – Max Net

بخش 1:

توضیح کد: در این قسمت، ابتدا ارایه ورودی و ماتریس وزن ها را تعریف کردم. در ادامه برا یتابع فعالسازی و خود الگوریتم، توابع جداگانه ای نوشتم. تابع الگوریتم که maxnet نام دارد، دو خروجی دادر. اولین خروجی، لیستی است که آپدیت های ارایه را در هر مرحله ذخیره کرده است. خروجی دوم نیز ایندکس المان بیشینه است. شرط توقف را به صورت عددی تعریف کردم که تعداد المان های غیرصفر ارایه را نشان می دهد. در ابتدا این عدد به اندازه همان طول ارایه اولیه است و در هر مرحله به روز می شود. در هر مرحله نیز خروجی تابع فعالسازی مجددا در ارایه اولهی قرار می گیرد . برای اینکه ارایه اصلی تغییر نکند، تمام تغییرات روی کپی ارایه اصلی صورت می گیرد.

```
array in 0 update is : [1.2  1.1  1.   0.9  0.95 1.15]
array in 1 update is : [0.435  0.32   0.205  0.09   0.1475 0.3775]
array in 2 update is : [0.264    0.13175  0.       0.       0.       0.197875]
array in 3 update is : [0.21455625 0.06246875 0.         0.         0.         0.1385125 ]
array in 4 update is : [0.18440906 0.00950844 0.         0.         0.         0.09695875]
array in 5 update is : [0.16843898 0.         0.         0.         0.         0.06787112]
array in 6 update is : [0.15825832 0.         0.         0.         0.         0.04260528]
array in 7 update is : [0.15186752 0.         0.         0.         0.         0.01886653]
array in 8 update is : [0.14903754 0.         0.         0.         0.         0.        ]

The max element is 1.2
```

خروجی های شبکه در هر مرحله و نتیجه نهایی

همانطور که مشاهده می شود، شبکه توانست المان بیشینه را که همان 1.2 بود، تشخیص دهد.

بخش 2:

در این قسمت، برای اینکه المان بیشینه را در ارایه ای ئیدا کنیم که تمام المان هایش از عددی مانند betta بزرگتر هستند، کافی است که مقدار betta را از تمام المان های ارایه کم کنیم و شبکه را روی ارایه نهایی اعمال کنیم.

در این سوال، می بینیم که تمام المان ها از 0.8 بیشتر هستند. بنابراین شبکه را روی ارایه [0.4, 0.3, 0.2, 0.1. 0.15, 0.35] اجرا می کنیم.

```
array in 0 update is : [0.4  0.3  0.2  0.1  0.15 0.35]
array in 1 update is : [0.235  0.12   0.005  0.     0.     0.1775]
array in 2 update is : [0.189625 0.057375 0.       0.       0.       0.1235  ]
array in 3 update is : [0.16249375 0.01040625 0.         0.         0.         0.08645   ]
array in 4 update is : [0.14796531 0.         0.         0.         0.         0.060515  ]
array in 5 update is : [0.13888806 0.         0.         0.         0.         0.0383202 ]
array in 6 update is : [0.13314003 0.         0.         0.         0.         0.01748699]
array in 7 update is : [0.13051698 0.         0.         0.         0.         0.        ]

The max element is 1.2
```

خروجی های شبکه در هر مرحله و نتیجه نهایی

همانطور که مشاهده می شود، شبکه همچنان توانست المان بیشینه را تشخیص دهد و نسبت به حالتت عادی نیز یک مرحله کمتر طول کشید تا به نتیجه برسد.

بخش 3:

برای مرتب کردن یک ارایه از بزرگ به کوچک، باید شبکه را به تعداد المان های ارایه تکرار کنیم. در هر مرحله که المان بیشینه پیدا شد، مقدار آن را در ارایه ای که به تابع maxnet می دهیم، صفر می کنیم. ارایه مرتب شده نیز در ابتدا یک لیست خالی است که در هر مرحله، المان بیشینه را به انتهای آن اضافه می کنیم.

```
max_to_min = []
temp_a = a.copy()
for i in range(m):
   [a_new, max_idx] = maxnet(temp_a, w)
   max_to_min.append(a[max_idx])
   temp_a[max_idx] = 0
print(f'Sorted array is : {max_to_min}')
```

```
Sorted array is : [1.2, 1.15, 1.1, 1.0, 0.95, 0.9]
```

مرتب کردن ارایه از بزرگ به کوچک و نتیجه نهایی

بخش 4:

برای مرتب کردن ارایه ای از کوچک به بزرگ، تمام مراحل مانند قسمت قبل است با این تفاوت که در هر مرحله، به جای اینکه المان بیشینه را به انتهای لیست اضافه کنیم، در ابتدای لیست اضافه م یکنیم. به این صورت، المان اول ارایه نهایی، کوچکترین المان و المان آخر نیز بزرگترین خواهد بود.

```
min_to_max = []
temp_a = a.copy()
for i in range(m):
   [a_new, max_idx] = maxnet(temp_a, w)
   min_to_max.insert(0, a[max_idx])
   temp_a[max_idx] = 0
print(f'Sorted array is : {min_to_max}')
```

```
Sorted array is : [0.9, 0.95, 1.0, 1.1, 1.15, 1.2]
```

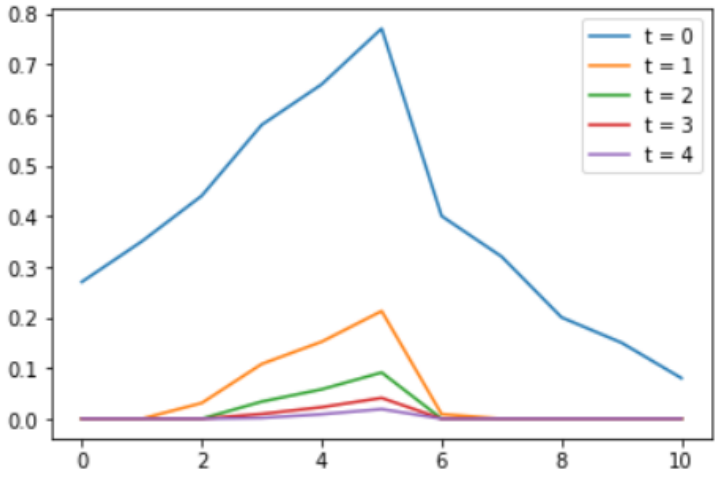
مرتب کردن ارایه از کوچک به بزرگ و نتیجه نهایی

سوال ۳ – Mexican Hat

بخش الف:

طبق آنچه در درس آموختیم، می دانیم که اگر R1 = 0, R2 = ∞ باشد، متد Mexican hat در واقع همان max net است.

توضیح کد: برای این قسمت، ماتریس وزن ها را مانند سوال قبل به دست می اوریم. در ادامه الگوریتم Mexican hat را برای 4 بار تکرار انجام دادم. نتایج هر بار آپدیت در لیستی ذخیره شده است که اولین المان لیست، داده های اصلی اولیه هستند. در ادامه نیز نمودار هر بار آپدیت را رسم کردم .



نمودار نتایج هربار آپدیت

بخش ب: در این بخش همه چیز مانند بخش قبل است با این تفاوت که ماتریس وزن ها متفاوت سات و تابع جداگانه ای برای محاسبه آن در نظر گرفته شده است. پارامترهای این بخش هم عبارتند از:

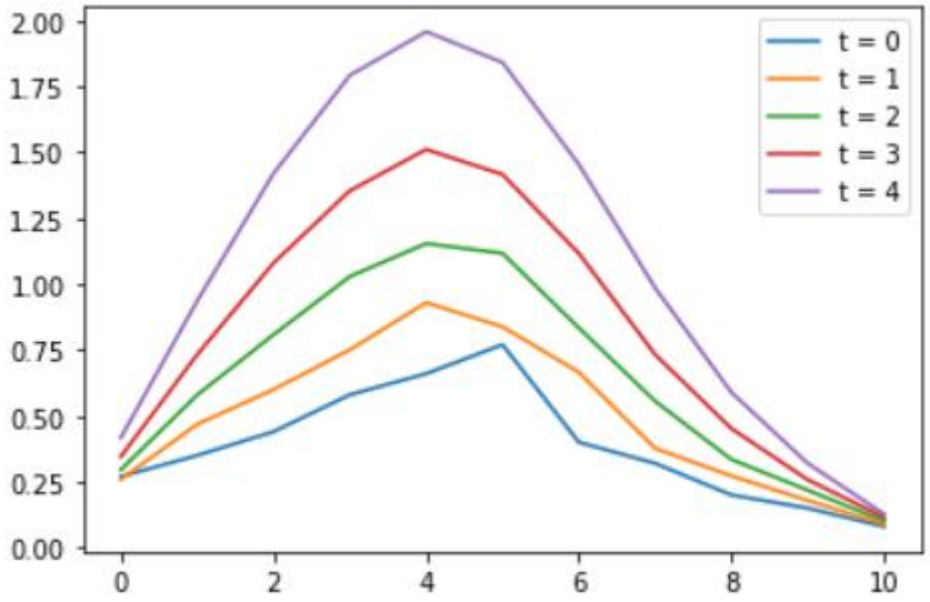
T max = 4

W1 = 0.5

W2 = -0.05

```
[[ 0.5   0.5  -0.05 -0.05  0.    0.    0.    0.    0.    0.    0.  ]
 [ 0.5   0.5   0.5  -0.05 -0.05  0.    0.    0.    0.    0.    0.  ]
 [-0.05  0.5   0.5   0.5  -0.05 -0.05  0.    0.    0.    0.    0.  ]
 [-0.05 -0.05  0.5   0.5   0.5  -0.05 -0.05  0.    0.    0.    0.  ]
 [ 0.   -0.05 -0.05  0.5   0.5   0.5  -0.05 -0.05  0.    0.    0.  ]
 [ 0.    0.   -0.05 -0.05  0.5   0.5   0.5  -0.05 -0.05  0.    0.  ]
 [ 0.    0.    0.   -0.05 -0.05  0.5   0.5   0.5  -0.05 -0.05  0.  ]
 [ 0.    0.    0.    0.   -0.05 -0.05  0.5   0.5   0.5  -0.05 -0.05]
 [ 0.    0.    0.    0.    0.   -0.05 -0.05  0.5   0.5   0.5  -0.05]
 [ 0.    0.    0.    0.    0.    0.   -0.05 -0.05  0.5   0.5   0.5 ]
 [ 0.    0.    0.    0.    0.    0.    0.   -0.05 -0.05  0.5   0.5 ]]
```

ماتریس وزن ها



نمودار نتایج هربار آپدیت

با مقایسه بخش الف و ب می بینیم که هر الگوریتم کار خود را انجام داده است. در بخش الف که در واقع max net در حال اجراست، قله نمودارها به سمت اندیس 5 رفته است که درایه ماکزیمم ورودی است . در بخش ب نیز که می خواهیم soft maximum را به دست آوریم، قله نمودارها به سمت اندیس 4 رفت است که کاملا مطابق انتظار ماست چراکه در المان های ارایه، المان چهارم که 0.66 است، بیشترین همبستگی را با المان های اطراف خود دارد.

سوال ۴ – Hamming Net

بخش الف:

توضیح کد: در ابتدا تصاویر به صورت ارایه هایی 3*4 ذخیره شدند و در ادامه نیز به صورت ارایه هایی سطری 12 تایی تغییر سایز داده شدند. در ادامه نیز با کنار هم قرار دادن مقدار نصف ارایه های سطری تصاویر، ماتریس وزن را ساختم. برای محاسبه فاصله hamming، تابع جداگانه ای نوشته شده است و از این تابع در اجرای الگوریتم استفاده شده است. در تابعی به نام HammingNat ابتدا فاصله hamming محاسبه می شود و سپس با پاس دادن ارایه نهایی به دست امده به تابع maxnet، ایندکس شبیه ترین تصویر به دست می آید.

```
[[ 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 -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]
 [ 0.5 -0.5  0.5 -0.5]
 [-0.5  0.5 -0.5  0.5]
 [ 0.5 -0.5  0.5  0.5]]
```

ماتریس وزن ها

```
Hamming Distance between X&Y is : 9.0
Hamming Distance between X&A is : 4.0
Hamming Distance between X&C is : 4.0
Hamming Distance between Y&A is : 1.0
Hamming Distance between Y&C is : 5.0
Hamming Distance between A&C is : 6.0
```

فاصله hamming بین هر جفت تصویر

قسمت ب:

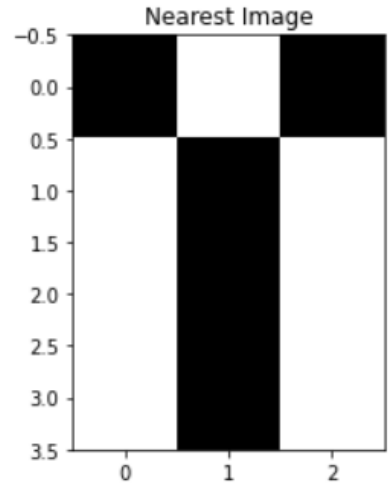
ماتریس وزن ها در قسمت قبل نشان داده شد. مقدار بایاس نیز نصف تعداد درایه های تصاویر داده شده است. در این بخش تمام عکس ها را در لیستی به نام exemplas قرار دادم تا بتوانم آن را به تابع بدهم .

قسمت پ:

در این قسمت ورودی جدیدی که همان تصویر حرف T است تعریف کردم. این ورودی و لیست exemplas های قبلی را به تابع HammimgNet دادم تا شبیه ترین تصویر را شناسایی کنید. نتایج به صورت زیر هستند:

```
The most similar img is : Y

Text(0.5, 1.0, 'Nearest Image')
```



شبیه ترین تصویر به ورودی.

طبق این نتیجه، تصویر حرف T بیشترین شباهت را به تصویر حرف Y دارد. برای اطمینان از درستی کارکرد الگوریتم، فاصله hamming ورودی را با تمام تصاویر قبلی محاسبه کردم و طبق این نتایج نیز تصویر Y بیشترین نزدیکی را داشت.

```
Hamming Distance between T&Y is : 11.0
Hamming Distance between T&A is : 2.0
Hamming Distance between T&C is : 6.0
```

فاصله hamming تصویر ورودی و تصاویر قبلی