به نام خدا

دانشگاه تهران

دانشکده مهندسی برق و کامپیوتر

شبکه های عصبی و یادگیری عمیق

تمرین سری ٤

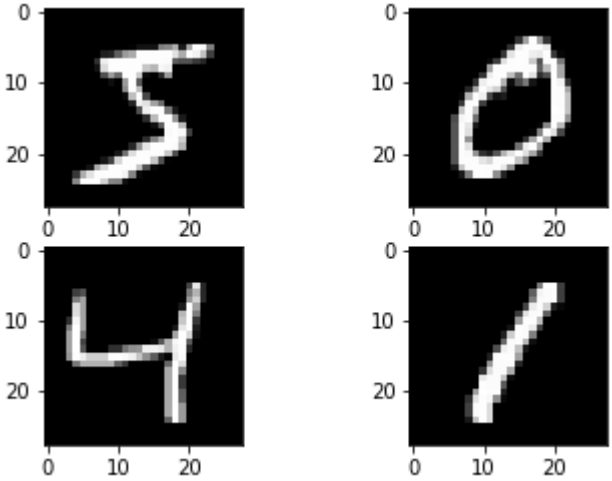
نام و نام خانوادگی	امید واهب
شماره دانشجویی	۸۱۰۱۹۶۵۸۲
تاریخ ارسال گزارش	۱۴۰۰/۰۴/۱۸

فهرست گزارش سوالات

سوال ۱ – SOM

در این سوال الگوریتم SOM را پیاده سازی کرده و با دو شعاع مجاورت و ساختار خطی آن را امتحان می کنیم تا تاثیر این پارامتر را مشاهده کنیم سپس ساختار دوبعدی آن را پیاده می کنیم و نتیجه ی حاصل را بررسی می کنیم. کد این سوال در فایل HW4_Q1.ipynb موجود است.

الف) ابتدا پیش از هر چیز داده را لود کرده و با نرمالایز کردن (بین ۰ و ۱)، آن را آماده یادگیری می کنیم. حال به چهار تا از داده ها را مشاهده می کنیم که به شکل زیر هستند:



شکل ۱ : نمونه ای از داده ها

سپس ماتریس وزن ها را با توجه به اطلاعات داده شده در مورد شبکه در صورت سوال، به صورت 625*784 تعریف می کنیم که ۶۲۵ بعد نورون خروجی و ۷۸۴ هم بعد 28*28 عکس های ورودی است. این بردار وزن را با مقادیر نرمال مقداردهی اولیه می کنیم. همچنین شرط توقف را ۲۵ ایپاک گذاشتم چون نرخ یادگیری در ادامه بیش از حد کوچک است و تغییری ایجاد نمی شود. سپس نورون های فعال که داده های تست در آن ها قرار می گیرند را بررسی کردم . لیبل های داده های تست مربوط به آنها را در ادامه ذکر کردم. برای بررسی اینکه یک داده به کدام cluster تعلق دارد آن را در ماتریس وزن ها ضرب کرده و در بردار ۶۲۵تایی حاصل نورون با حداقل مقدار یا همان فاصله، cluster مدنظر است. نحوه تقسیم داده های تست بین نورون ها به شکل زیر است:

```
[6]
[3]
[1]
[1, 1, 1, 1, 1, 1, 1]
[4, 4, 4, 4, 7]
[1, 1, 1, 1, 1]
[1, 1, 1, 1]
```

[3]
[1, 1]
[7, 7]
[1, 9, 9, 7, 4, 7]
[1, 1, 1]
[4, 1, 7, 9, 7, 7]
[1]
[1, 1, 1, 1, 1, 1]
[1]
[7, 2, 0, 4, 4, 9, 5, 9, 0, 6, 9, 0, 5, 9, 7, 3, 4, 9, 6, 6, 5, 4, 0,
7, 4, 0, 3, 3, 4, 7, 2, 7, 2, 7, 4, 2, 3, 5, 1, 2, 4, 4, 6, 3, 5, 5,
6, 0, 4, 9, 5, 7, 8, 9, 3, 7, 6, 4, 3, 0, 7, 0, 2, 9, 7, 3, 2, 7, 7,
6, 2, 7, 8, 4, 7, 3, 6, 3, 6, 3, 1, 4, 1, 7, 6, 9, 6, 0, 5, 4, 9, 9,
2, 9, 4, 8, 3, 9, 4, 2, 5, 4, 7, 6, 7, 9, 0, 5, 8, 5, 6, 6, 5, 7, 8,
0, 6, 4, 6, 7, 3, 7, 8, 2, 0, 2, 9, 9, 5, 5, 5, 6, 0, 3, 4, 6, 5, 4,
6, 5, 4, 5, 4, 4, 7, 2, 3, 2, 8, 8, 8, 5, 0, 8, 9, 2, 5, 0, 1, 0, 9,
0, 3, 6, 4, 2, 3, 6, 3, 9, 5, 2, 9, 4, 5, 9, 9, 0, 3, 6, 5, 5, 7, 2,
2, 7, 2, 8, 3, 3, 8, 8, 9, 2, 2, 4, 5, 9, 8, 7, 2, 3, 0, 4, 4, 2, 4,
9, 5, 7, 7, 2, 8, 2, 6, 8, 5, 7, 7, 9, 1, 8, 8, 0, 3, 0, 9, 9, 4, 8,
2, 2, 9, 5, 9, 2, 6, 4, 5, 8, 2, 9, 2, 0, 4, 0, 0, 2, 8, 4, 7, 2, 4,
0, 2, 4, 3, 3, 0, 0, 3, 9, 6, 5, 2, 5, 2, 9, 3, 0, 4, 2, 0, 7, 2, 5,
3, 3, 9, 7, 8, 6, 5, 6, 3, 8, 0, 5, 3, 5, 5, 6, 8, 5, 7, 9, 4, 6, 2,
2, 5, 0, 6, 5, 6, 3, 7, 2, 0, 8, 8, 5, 4, 0, 3, 3, 7, 6, 6, 2, 9, 2,
8, 6, 9, 5, 2, 5, 4, 4, 2, 8, 3, 8, 2, 4, 5, 0, 3, 7, 5, 7, 9, 7, 9,
2, 4, 2, 9, 2, 0, 4, 9, 4, 8, 8, 4, 5, 3, 7, 0, 0, 3, 0, 2, 6, 6, 4,
9, 3, 3, 3, 2, 3, 9, 2, 6, 8, 0, 5, 6, 6, 6, 3, 8, 8, 2, 7, 5, 8, 6,
8, 4, 2, 5, 9, 9, 7, 5, 4, 0, 8, 9, 9, 0, 5, 2, 3, 7, 8, 9, 4, 0, 3,
9, 5, 2, 3, 3, 6, 5, 7, 2, 2, 6, 2, 6, 5, 4, 8, 7, 1, 3, 0, 3, 8, 3,
9, 3, 4, 6, 4, 2, 8, 2, 5, 4, 8, 8, 4, 0, 0, 2, 3, 2, 7, 0, 8, 7, 4,
4, 7, 9, 6, 0, 9, 8, 0, 4, 0, 6, 3, 5, 4, 8, 3, 3, 9, 3, 3, 3, 7, 8,
0, 8, 2, 7, 0, 6, 5, 4, 3, 8, 0, 9, 6, 3, 8, 0, 9, 9, 6, 8, 6, 8, 5,
8, 6, 0, 2, 4, 0, 2, 1, 9, 7, 5, 1, 0, 8, 4, 6, 2, 6, 7, 9, 3, 2, 9,
8, 2, 2, 9, 2, 7, 3, 5, 1, 8, 0, 2, 0, 5, 2, 3, 6, 7, 2, 5, 8, 0, 3,
7, 2, 4, 0, 9, 8, 6, 4, 3, 4, 9, 9, 5, 7, 3, 9, 7, 6, 9, 7, 8, 3, 3,
6, 2, 8, 5, 8, 5, 4, 4, 3, 1, 0, 7, 0, 7, 9, 4, 8, 5, 5, 4, 0, 8, 2,
0, 8, 4, 5, 0, 4, 0, 6, 3, 2, 6, 7, 2, 6, 9, 3, 4, 6, 2, 5, 4, 2, 0,
6, 2, 7, 3, 4, 0, 5, 4, 3, 7, 4, 8, 4, 0, 2, 4, 5, 6, 4, 7, 9, 2, 4,
1, 5, 5, 3, 8, 3, 4, 5, 6, 8, 9, 4, 5, 3, 8, 0, 3, 2, 5, 1, 2, 8, 3,
4, 4, 0, 8, 8, 3, 3, 7, 3, 5, 9, 6, 3, 2, 6, 1, 3, 6, 0, 7, 2, 7, 4,
2, 4, 2, 7, 9, 6, 2, 4, 8, 7, 7, 4, 8, 0, 7, 3, 3, 0, 7, 7, 0, 3, 5,
5, 2, 6, 6, 9, 2, 8, 3, 5, 2, 2, 5, 6, 0, 8, 2, 2, 8, 8, 8, 8, 7, 4,
3, 0, 6, 6, 3, 2, 3, 2, 2, 9, 3, 0, 0, 5, 7, 8, 4, 4, 6, 0, 2, 9, 4,
7, 4, 7, 3, 9, 8, 8, 4, 7, 2, 2, 2, 3, 2, 3, 2, 3, 9, 7, 4, 0, 3, 5,
5, 8, 6, 3, 2, 6, 7, 6, 6, 3, 2, 7, 8, 5, 6, 9, 5, 1, 3, 3, 4, 8, 9,
6, 9, 4, 4, 5, 4, 0, 6, 2, 2, 3, 5, 1, 2, 0, 3, 8, 2, 6, 7, 6, 2, 3,
9, 0, 1, 2, 2, 0, 8, 9]
[9, 9]
[7, 9, 7, 7, 7, 9, 9, 7, 1, 9, 4, 7]
[1]
[4, 3]
[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 8, 1, 1, 1, 1, 1, 1, 7, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1]
[9]
[1, 9]
[1, 1, 1, 1, 1, 1, 1, 1, 1, 7, 1, 8, 1, 1, 2, 1, 1, 1, 1, 1, 1, 1, 1,
1]

```
[4]
[9, 6, 6, 4, 9, 1]
```

می توان مشاهده کرد که کلاس های ۱، ۷، ۴ و ۹ تا حدی جذب چند تا از نورون ها شده اند و باقی داده های کلاس ها جذب یکی از نورون ها شده اند و اکثریت داده یک جا تجمع کرده است که حاصل کم بودن شعاع مجاورت است. حال برای بررسی و کلاسیفای کردن به کمک مدل ساخته شده رای داده های train در هر cluster لیبلی که بیشترین تکرار را دارد را به عنوان لیبل آن cluster انتخاب می کنیم و به این صورت برای هر داده ی تست لیبل clusterی که در آن میافتد را به آن می دهیم. با این پروسه میزان دقت برای داده ی train و test به شرح زیر خواهد بود:

```
Accuracy on train data is 20.95 percent.

Accuracy on test data is 22.90 percent.
```

به دقت بالای ۲۰ درصد رسیدیم که خیلی دقت بدی نیست و از baseline ۱۰ درصد بالاتر است. علت کم بودن را می توان به ایپاک کم و شعاع مجاورت صفر ربط داد همچنین عدم استفاده از لیبل ها برای آموزش مدل نیز مشکل این روش است.

ب) حال شعاع مجاورت را به ۲ افزایش می دهیم به این صورت که وقتی نورونی را می خواهیم آپدیت کنیم دو نورون چپ و دو نورون راست آن را نیز آپدیت می کنیم. طبق انتظار در همان ۲۵ ایپاک نتیجه ی بهتری می گیریم. لیبل داده های تست قرار گرفته در هر یک از cluster ها به شرح زیر است:

```
[1, 1]
[2, 3, 3, 1, 3, 8, 3, 3, 5, 2, 3, 3, 2, 3, 3, 2, 3, 8, 3, 3, 1, 3, 1,
3, 3, 8, 3]
[5, 3, 9, 3, 7, 3, 1, 7, 2, 5, 8, 8, 5, 6, 3, 3, 3, 5, 8, 3, 8, 5, 9,
2, 8, 2, 2, 3, 5, 3, 3, 3, 8, 5, 3, 5, 8, 3, 8, 3, 6, 3, 6, 3, 3, 6,
5, 2, 9, 5, 3, 6, 3, 3, 8, 8, 2, 3, 3, 2, 8, 3, 3, 3, 3, 6, 2, 3, 5,
6, 3, 8, 3, 3, 5, 2, 6, 3, 3, 3, 5, 6, 3, 3, 1, 3, 3, 8, 4, 3, 3]
[9, 5, 3, 8, 5, 5, 8, 8, 5, 5, 2, 8, 2, 5, 3, 5, 8, 3, 0, 5, 5, 3, 9,
3, 8, 3, 6, 5, 9, 3, 8, 2, 7, 8, 8, 8, 8, 8, 3, 2, 8, 3, 8, 2, 3, 8,
5, 6]
[9, 5]
[3, 2, 2, 2, 7, 8, 1, 2, 3, 2, 2, 2, 2, 3, 2, 2, 2]
[2, 3, 3, 3, 3, 3, 3, 3]
[3, 0]
[3, 0, 2, 3, 0, 5, 3, 5, 3, 7, 1, 7, 8, 2, 5, 8, 5, 1, 2, 1, 2, 3, 7,
1]
[7, 0, 5, 9, 7, 1, 3, 7, 7, 3, 1, 6, 5, 9, 7, 7, 6, 1, 6, 2, 5, 4, 7,
6, 4, 6, 6, 5, 1, 1, 6, 9, 5, 7, 7, 7, 8, 2, 7, 0, 6, 5, 9, 0, 7, 4,
0, 5, 4, 5, 7, 1, 5, 2, 7, 0, 3, 7, 6, 9, 8, 5, 7, 4, 9, 8, 7, 3, 3,
6, 0, 8, 9, 5, 7, 3, 3, 7, 0, 7, 7, 0, 4, 8, 9, 7, 0, 5, 9, 3, 9, 1,
6, 7, 5, 5, 7, 6, 7, 0, 5, 7, 6, 7, 3, 0, 7, 7, 9, 4, 2, 6, 3, 5, 4,
6, 2, 7, 7, 9, 5, 8, 5, 3, 3, 7, 9, 7, 9, 7, 8, 0, 7, 8, 4, 6, 6, 7,
7, 5, 7, 8, 5, 6, 9, 7, 6, 1, 8]
```

```
[4, 9, 9, 6, 5, 7, 3, 6, 6, 5, 9, 4, 4, 9, 6, 7, 7, 6, 9, 5, 6, 0, 9,
0, 9, 0, 3, 9, 4, 9, 7, 3, 7, 6, 0, 9, 9, 0, 3, 9, 0, 6, 6, 5, 6, 6,
9, 7, 0, 4, 5, 6, 6, 9, 6, 5, 7, 0, 0, 6, 3, 6, 8, 4, 6, 3, 7, 0, 6,
0, 6, 4, 0, 0, 6, 9, 7, 0, 7, 9, 9, 6, 8, 4, 4, 0, 6, 7, 7, 6, 0, 4,
9, 7, 3, 6, 0, 6, 6, 7, 7, 0, 7, 6, 6, 9, 5, 0, 9, 0, 6, 4, 4, 9, 9,
5, 6, 6, 6, 5, 9, 4, 9, 9]
[4, 0, 4, 4, 0]
[4, 9, 4, 7, 4, 4, 4, 4, 4, 0, 4, 9, 4, 7, 4, 4, 9, 9, 4, 4, 7, 4, 4,
4, 4, 4, 4, 4, 4, 4, 9, 7, 8, 4, 0, 9, 0, 6, 9, 4, 0, 4, 4, 4, 7, 9,
0, 0, 9, 9, 4, 9, 5, 7, 4, 4, 4, 4, 8, 4, 9, 8, 4, 6, 4, 9, 5, 6, 9,
4, 2, 5, 4, 9, 4, 4, 5, 5, 4, 4, 0, 4, 4, 4, 9, 4, 4, 4, 9, 4, 2, 4,
9, 4, 9, 7, 2, 4, 4, 6]
[0]
[0, 6, 0, 4, 0, 0, 7, 2, 6, 7, 6, 0, 7, 2, 8, 4, 6, 0, 8, 6, 7, 2, 0,
6, 7, 8, 0, 2, 9, 0, 2, 2, 9, 8, 0, 2, 8, 8, 2, 9, 4, 4, 0, 2, 2, 4,
0, 2, 9, 8, 6, 7, 6, 6, 8, 2, 8, 2, 0, 5, 2, 2, 4, 4, 8, 0, 2, 6, 4,
8, 0, 8, 7, 8, 4, 9, 8, 9, 2, 7, 6, 2, 1, 0, 8, 4, 2, 0, 0, 0, 5, 0,
8, 6, 8, 9, 8, 0, 0, 9, 6, 4, 4, 9, 7, 2, 0, 8, 2, 5, 0, 9, 4, 2, 4,
2, 8, 4, 5, 8, 2, 0, 8, 2, 4, 4, 7, 0, 5, 2, 2, 8, 2, 2, 8, 8, 8, 8,
7, 0, 2, 2, 0, 5, 8, 4, 0, 2, 4, 2, 2, 4, 0, 6, 4, 0, 2, 2, 0, 2, 0]
[5, 2, 4, 5, 7, 7, 2]
[1, 1, 1, 1, 1, 5, 1, 1, 1, 7, 9, 1, 1, 1, 1, 1, 1, 1, 5, 1, 1, 1, 1,
1, 5, 1, 1, 1, 1]
[1, 1, 1, 1, 5, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 3, 1, 1, 1, 1, 1, 5,
1, 1, 1, 1]
[1, 1, 1, 1, 1, 7, 8, 1, 1, 5, 1, 1, 1]
[1, 1, 1, 2, 1, 8, 1, 9, 7, 1, 1, 2, 7, 1, 1, 1, 1, 6, 1, 2, 2, 2, 1,
1, 6, 2, 1, 1, 8, 1, 1, 1, 1, 2, 2, 1, 1, 1, 8, 1, 1, 1, 8, 1, 2, 2,
2, 1, 2, 2, 2, 8, 7, 1, 8, 5, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1, 2,
7, 1, 1, 1, 7, 6, 7, 1, 1, 1, 1, 8, 2]
[2, 2, 2, 8, 8, 2, 8, 2, 2, 2]
[2, 2, 0, 2, 2, 2, 0]
[2, 0]
[8]
[5, 7, 5, 9, 9, 5, 3, 9, 3, 5, 7, 5]
[9, 9, 9, 3, 9, 3, 5, 6, 9, 3, 7, 9, 7, 7, 2, 3]
[6, 9]
[4, 7, 9, 7, 4, 5, 4, 4, 4, 9, 4, 4, 4, 9, 9, 4, 6, 4, 9, 9, 7]
[6]
[0]
```

می توان مشاهده کرد که تقسیم داده ها بین نورون ها خیلی بهتر انجام شده است و تعداد نورون های فعال کمتر شده و به مقدار ۱۰ که تعداد کلاس های واقعی است نزدیکتر شده است. می توان برای ساختن یک کلاسیفایر بهتر بعضی از این نورون ها را با هم تجمیع کنیم. اگر مدلی مثل حالت قبل بسازیم دقت هایی به شرح زیر می گیریم:

```
Accuracy on train data is 39.80 percent.

Accuracy on test data is 37.10 percent.
```

می توان دید که دقت مدل دو برابر شده است و دقت نزدیک ۴۰ درصد برای این مدل روی همچین داده ای خیلی خوب است و می توانستیم با ادامه ی ایپاک ها و دو بعدی کردن این دقت را بهبود دهیم.

ج) حال در این قسمت شبکه را به صورت دوبعدی با ابعاد ۲۵*۲۵ طراحی می کنیم و هنگام آپدیت کردن وزن ها علاوه بر نورون چپ و راست، نورون بالا و پایین در معماری گفته شده را نیز آپدیت می کنیم. از نظر تعداد نورون آپدیت شده در هر مرحله این حالت با حالت قبلی یکسان است پس انتظار جواب خیلی بهتری به نسبت حالت قبل در ۲۵ ایپاک نداریم. نحوه توزیع داده های تست بین نورون های فعال به شرح زیر است:

```
[3, 9]
[3, 4, 5, 4]
[5, 6, 8]
[4, 7, 6, 7, 4, 6, 6, 7, 6, 0, 1, 7, 7, 6]
[6, 2]
[1, 8, 8, 8, 4, 8]
[0, 2]
[1, 1, 1, 1]
[7, 7, 4]
[7, 7]
[9, 9]
[7, 8, 6, 4, 3, 8, 3, 3, 4, 3, 3]
[5]
[5, 5, 4, 5]
[1, 7, 1, 1, 9, 1, 1]
[6, 8, 8, 8, 2, 8, 9, 8]
[9, 5, 5, 1, 0]
[0]
[7, 7, 7, 7, 9, 8, 7, 7, 7, 5, 5, 5, 9, 7, 5, 0, 7, 7]
[5, 5, 5, 0, 5, 3, 9, 7]
[7]
[1, 3, 2, 3, 2, 1, 5, 3, 3, 3, 2, 3, 1, 5, 5, 7, 0, 5, 7, 1, 5, 5, 6,
3, 1]
[4, 3]
[6]
[7]
[1, 3, 1, 3]
[3]
[7, 3, 7, 3]
[6]
[0, 0]
[7, 2, 2]
[3]
[9, 9, 6, 3, 9, 5, 1, 1, 8, 8, 2, 4, 4, 8, 1, 9, 7, 7, 1, 1, 1, 5, 1,
1, 7, 1, 8, 1, 4, 6, 6, 3, 1, 5, 3, 9, 3, 8, 8, 9, 4, 9, 5, 8, 2, 1,
1, 3, 3, 3, 6, 4, 4, 1, 3, 6, 4, 1, 4, 2, 3, 6, 1, 1, 5, 3, 1, 3, 1,
6, 8, 5, 6, 1, 6]
[7, 1]
[3, 8, 3, 5, 9, 8, 9, 9, 3, 9, 5, 8, 8, 5, 5, 3, 5, 4, 5, 9, 3, 5, 4,
3, 9, 5, 5, 5, 3, 9, 8, 8, 5, 8, 3, 3, 3]
[7, 7]
```

[5, 2, 2, 2, 2, 5, 2, 2, 3, 0, 2, 2, 0, 2, 5, 6, 2, 8, 2, 3, 2, 0, 5, 6, 2, 0, 2, 0, 2, 3, 2]
[4, 4]
[7, 7, 8, 7, 0, 5]
[3]
[4, 2, 0, 9, 8, 2]
[5, 9, 8]
[4, 6, 0, 0, 0, 0, 0, 6]
[6, 4, 5, 2, 5, 5, 5, 5, 8, 9, 5, 5, 7, 2]
[6, 6, 3]
[0]
[6, 9, 4, 6, 4, 9, 4, 9, 9, 7, 9]
[8, 0, 0]
[1]
[7, 7, 7]
[9, 7, 9, 4, 7, 4, 4, 9, 4, 7, 2, 9, 4, 3, 7, 2, 2, 3, 9, 2, 4, 4, 4, 6, 2, 9, 9, 2]
[4, 6, 4, 6, 6, 6, 6, 6, 9, 6, 9, 6, 4, 6, 6, 9]
[5, 9, 1, 0, 6]
[5, 3, 6]
[4]
[3]
[0]
[3]
[3, 0]
[6]
[6, 6]
[3, 6, 1, 1, 1, 1, 5, 1, 2, 2, 2, 8, 6, 1, 1, 1, 2, 8, 6, 1, 1]
[5, 5]
[6, 0, 5, 0, 6, 0]
[0, 0, 0, 0, 0, 0, 0, 5, 9, 9, 0, 9, 4, 0, 9, 8, 5, 8, 4, 7, 4, 8, 0, 6, 9, 5, 8]
[2, 7, 2, 7]
[0, 0, 0, 5, 0, 6, 3, 2, 0, 0, 9, 0, 0, 5, 2, 3, 0, 3, 8, 8, 8, 3, 6, 0, 6, 2, 0, 0, 2, 3, 2, 5, 3, 8, 9, 0, 0, 6]
[9, 9, 7, 9, 7, 1, 8, 1, 8, 5, 4, 8, 7, 4, 8, 2, 8, 8, 8, 9, 8, 6, 8, 8, 7, 4, 6, 7, 8, 9, 7, 7, 7, 7, 9]
[3, 5, 9, 3, 3, 3, 3]
[4]
[6]
[4, 5]
[4, 2, 7, 2, 6, 6, 2, 6]
[7, 7, 7]
[9, 9, 0, 2, 2, 7]
[5]
[2, 1, 1, 1, 1, 1, 2, 2]
[0]
[3, 3, 3, 2]
[5, 0, 5, 0, 5, 0, 0, 0, 8]
[8, 1, 1, 0, 4]
[5, 5, 3]
[4, 5]
[9, 4, 4, 8, 8, 9, 4, 9, 8, 4, 8, 4, 4, 8, 1, 8, 8]
[6]
[4, 4]
[4, 9, 7, 4, 4, 7, 7, 7, 7, 7, 7, 9, 6, 7, 9, 4, 7, 7, 7, 4, 7, 4, 9, 2]

```
[3, 0, 8, 0, 8, 4, 5, 0, 0, 0, 0, 4, 0, 0, 4, 0]
[1, 1, 1, 1, 1, 1, 1]
[3, 8, 5, 5, 3, 3, 3, 3]
[2, 2]
[3, 3]
[1, 2]
[5, 6, 5]
[1, 1]
[7, 2, 2, 2, 2, 2, 6, 2, 7, 2, 2]
[1, 1, 1, 1, 1, 1, 1, 1, 1, 6, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1]
[2]
[1, 6, 6, 6, 2, 2, 9, 1, 6, 1, 0, 1, 9, 2, 6, 2, 2]
[4, 7]
[6, 6]
[9, 4]
[1, 1, 1, 1, 7, 1, 1]
[6, 2, 3, 2, 2, 3, 2]
[2, 6, 9, 2, 2, 8, 2, 3, 8, 2, 2, 0, 2, 3, 3, 8, 3, 6, 2]
[0]
[1]
[3]
[3, 2, 0, 2, 2, 2, 2]
[4]
[3]
[3, 1, 3, 1, 4, 1]
[5]
[4]
[4, 6, 2]
[6]
[0, 2, 0, 0]
[4, 4, 4, 8, 8, 2, 4, 4, 2, 1, 4, 9, 8, 3, 8, 4, 8, 4, 2, 4, 4, 4, 8,
4, 8, 8, 2]
[4, 7, 1, 7, 9, 9, 6, 4, 6, 4, 4, 2, 9, 2, 9, 9, 9, 2, 4, 9, 1, 8, 9,
9, 4, 8, 9, 9, 4, 7, 4, 9, 9, 2, 9, 1, 8, 9, 1, 4, 9, 7, 9, 4, 7, 8,
9, 8, 3, 7, 4, 4, 4, 4, 3, 7, 9, 9]
[3, 5, 4, 0]
[4, 4]
[0, 6, 4, 8]
[9]
[2, 2, 8, 2, 1]
[7, 7, 7, 7]
[7, 3, 3, 3, 3, 7, 7, 5]
[7, 2, 2, 2, 2, 0, 8, 8]
[1]
[6, 4, 6, 6]
```

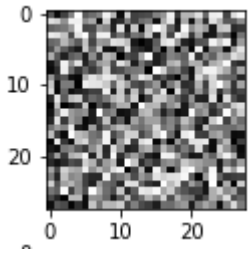
می توان دید که تعداد نورون های فعال در این حالت بیشتر است و می بایست تعداد ایپاک های بیشتری یادگیری را انجام دهیم. دقت ان حالت نیز به شرح زیر است:

```
Accuracy on train data is 44.50 percent.

Accuracy on test data is 36.20 percent.
```

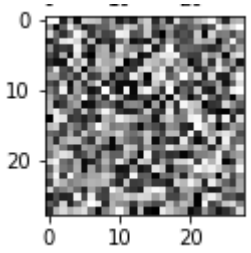
می توان مشاهده کرد که به دلیل دوبعدی کردن اندکی در دقت train بهبود داشتیم که با افزایش ایپاک ها (به دلیل محدودیت colab و زمان زیاد یادگیری ۲۵ انتخاب شد) می توانستیم بهبود بیشتری نیز داشته باشیم همچنین شعاع مجاورت نیز بهتر بود بزرگتر انتخاب میشد و به مرور زمان کاهش می یافت.

وزن دو تا از نورون ها را به صورت شکل ۲۸*۲۸ می کشیم که به صورت زیر خواهد بود:



شکل ۲ : وزن های یکی از نورون های خروجی

می توان دید که در این نورون داده های 3، 4 و 5 قرار دارند که همگی شکلی گرد دارند و دایره ی وسط نیز قابل تشخیص است. مثال دیگر عکس زیر است که کلاسی است که اکثر داده های آن مربوط به عدد ۳ هستند و می توان در آن شکل عدد ۳ را تشخیص داد:



شکل ۳ : وزن های یکی از نورون های خروجی

اگر تعداد ایپاک ها و شعاع مجاورت بیشتر می بود قطعا تصاویر خیلی بهتری را می توانستیم رسم کنیم.

برای مثال شبکه ی آخر را یکبار با ایپاک بیشتر (۵۰) اجرا کردم و نزدیک ۵۰ درصد دقت گرفته شد.

سوال ۲- MaxNet

در این سوال الگوریتم MaxNet را پیاده سازی کرده و با ایجاد تغییراتی در این الگوریتم آن را برای یافتن حداکثر مقدار قدر مطلق اعداد آماده می کنیم. کد این سوال در فایل HW4_Q2.ipynb موجود است.

الگوریتم MaxNet را مرحله به مرحله پیش می بریم و مقدار ماکسیمم را پیدا می کنیم. ابتدا ماتریس وزن ها را تشکیل می دهیم که چون اپسیلون برابر۰.۱۳ به شرح زیر است:

$$W = \begin{bmatrix} 1 & -0.13 & -0.13 & -0.13 & -0.13 & -0.13 \\ -0.13 & 1 & -0.13 & -0.13 & -0.13 & -0.13 \\ -0.13 & -0.13 & 1 & -0.13 & -0.13 & -0.13 \\ -0.13 & -0.13 & -0.13 & 1 & -0.13 & -0.13 \\ -0.13 & -0.13 & -0.13 & -0.13 & 1 & -0.13 \\ -0.13 & -0.13 & -0.13 & -0.13 & -0.13 & 1 \end{bmatrix}$$

همچنین داریم که بردارx برابر است با $x = [1.2,1.1,0.5,1.5,1.13,0.8]^T$ پس به کمک فرمول زیر مراحل را پیش می بریم:

$$x_{new} = f(Wx_{old})$$

پس داریم:

$$x_{new} = Wx_{old} = f(\begin{bmatrix} 1 & -0.13 & -0.13 & -0.13 & -0.13 & -0.13 \\ -0.13 & 1 & -0.13 & -0.13 & -0.13 & -0.13 \\ -0.13 & -0.13 & 1 & -0.13 & -0.13 & -0.13 \\ -0.13 & -0.13 & -0.13 & 1 & -0.13 & -0.13 \\ -0.13 & -0.13 & -0.13 & -0.13 & 1 & -0.13 \\ -0.13 & -0.13 & -0.13 & -0.13 & -0.13 & 1 \end{bmatrix} \begin{bmatrix} 1.2 \\ 1.1 \\ 0.5 \\ 1.5 \\ 1.13 \\ 0.8 \end{bmatrix})$$

$$= f\left(\begin{bmatrix} 0.5461 \\ 0.4331 \\ -0.2449 \\ 0.8851 \\ 0.4670 \\ 0.0941 \end{bmatrix}\right) = \begin{bmatrix} 0.5461 \\ 0.4331 \\ 0 \\ 0.8851 \\ 0.4670 \\ 0.0941 \end{bmatrix}$$

Activation function این الگوریتم مقادیر منفی را برابر صفر قرار می دهد. حال مرحله بعدی را انجام می دهیم:

$$x_{new} = Wx_{old} = f\left(\begin{bmatrix} 1 & -0.13 & -0.13 & -0.13 & -0.13 & -0.13 \\ -0.13 & 1 & -0.13 & -0.13 & -0.13 & -0.13 \\ -0.13 & -0.13 & 1 & -0.13 & -0.13 & -0.13 \\ -0.13 & -0.13 & -0.13 & 1 & -0.13 & -0.13 \\ -0.13 & -0.13 & -0.13 & -0.13 & 1 & -0.13 \\ -0.13 & -0.13 & -0.13 & -0.13 & -0.13 & 1 \end{bmatrix}\begin{bmatrix} 0.5461 \\ 0.4331 \\ 0 \\ 0.8851 \\ 0.4670 \\ 0.0941 \end{bmatrix}\right)$$

$$= f\left(\begin{bmatrix} 0.3018 \\ 0.1741 \\ -0.3153 \\ 0.6849 \\ 0.2124 \\ -0.2090 \end{bmatrix}\right) = \begin{bmatrix} 0.3018 \\ 0.1741 \\ 0 \\ 0.6849 \\ 0.2124 \\ 0 \end{bmatrix}$$

نتایج مرحله بعد به شرح زیر است:

$$x_{new} = Wx_{old} = f\left(\begin{bmatrix} 1 & -0.13 & -0.13 & -0.13 & -0.13 & -0.13 \\ -0.13 & 1 & -0.13 & -0.13 & -0.13 & -0.13 \\ -0.13 & -0.13 & 1 & -0.13 & -0.13 & -0.13 \\ -0.13 & -0.13 & -0.13 & 1 & -0.13 & -0.13 \\ -0.13 & -0.13 & -0.13 & -0.13 & 1 & -0.13 \\ -0.13 & -0.13 & -0.13 & -0.13 & -0.13 & 1 \end{bmatrix}\begin{bmatrix} 0.3018 \\ 0.1741 \\ 0 \\ 0.6849 \\ 0.2124 \\ 0 \end{bmatrix}\right)$$

$$= f\left(\begin{bmatrix} 0.1625 \\ 0.0182 \\ -0.1785 \\ 0.5954 \\ 0.0615 \\ -0.1785 \end{bmatrix}\right) = \begin{bmatrix} 0.1625 \\ 0.0182 \\ 0 \\ 0.5954 \\ 0.0615 \\ 0 \end{bmatrix}$$

نتیجه مرحله بعد نیز به شکل زیر است:

$$x_{new} = Wx_{old} = f\left(\begin{bmatrix} 1 & -0.13 & -0.13 & -0.13 & -0.13 & -0.13 \\ -0.13 & 1 & -0.13 & -0.13 & -0.13 & -0.13 \\ -0.13 & -0.13 & 1 & -0.13 & -0.13 & -0.13 \\ -0.13 & -0.13 & -0.13 & 1 & -0.13 & -0.13 \\ -0.13 & -0.13 & -0.13 & -0.13 & 1 & -0.13 \\ -0.13 & -0.13 & -0.13 & -0.13 & -0.13 & 1 \end{bmatrix}\begin{bmatrix} 0.1625 \\ 0.0182 \\ 0 \\ 0.5954 \\ 0.0615 \\ 0 \end{bmatrix}\right)$$

$$= f\left(\begin{bmatrix} 0.0747 \\ -0.0883 \\ -0.1089 \\ 0.5639 \\ -0.0394 \\ -0.1089 \end{bmatrix}\right) = \begin{bmatrix} 0.0747 \\ 0 \\ 0 \\ 0.5639 \\ 0 \\ 0 \end{bmatrix}$$

سپس خواهیم داشت:

$$x_{new} = Wx_{old} = f\left(\begin{bmatrix} 1 & -0.13 & -0.13 & -0.13 & -0.13 & -0.13 \\ -0.13 & 1 & -0.13 & -0.13 & -0.13 & -0.13 \\ -0.13 & -0.13 & 1 & -0.13 & -0.13 & -0.13 \\ -0.13 & -0.13 & -0.13 & 1 & -0.13 & -0.13 \\ -0.13 & -0.13 & -0.13 & -0.13 & 1 & -0.13 \\ -0.13 & -0.13 & -0.13 & -0.13 & -0.13 & 1 \end{bmatrix}\begin{bmatrix} 0.0747 \\ 0 \\ 0 \\ 0.5639 \\ 0 \\ 0 \end{bmatrix}\right)$$

$$= f\left(\begin{bmatrix} 0.0014 \\ -0.0830 \\ -0.0830 \\ 0.5542 \\ -0.0830 \\ -0.0830 \end{bmatrix}\right) = \begin{bmatrix} 0.0014 \\ 0 \\ 0 \\ 0.5542 \\ 0 \\ 0 \end{bmatrix}$$

و در نهایت داریم:

$$x_{new} = Wx_{old} = f\left(\begin{bmatrix} 1 & -0.13 & -0.13 & -0.13 & -0.13 & -0.13 \\ -0.13 & 1 & -0.13 & -0.13 & -0.13 & -0.13 \\ -0.13 & -0.13 & 1 & -0.13 & -0.13 & -0.13 \\ -0.13 & -0.13 & -0.13 & 1 & -0.13 & -0.13 \\ -0.13 & -0.13 & -0.13 & -0.13 & 1 & -0.13 \\ -0.13 & -0.13 & -0.13 & -0.13 & -0.13 & 1 \end{bmatrix}\begin{bmatrix} 0.0014 \\ 0 \\ 0 \\ 0.5542 \\ 0 \\ 0 \end{bmatrix}\right)$$

$$= f\left(\begin{bmatrix} -0.0706 \\ -0.0722 \\ -0.0722 \\ 0.5540 \\ -0.0722 \\ -0.0722 \end{bmatrix}\right) = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0.5540 \\ 0 \\ 0 \end{bmatrix}$$

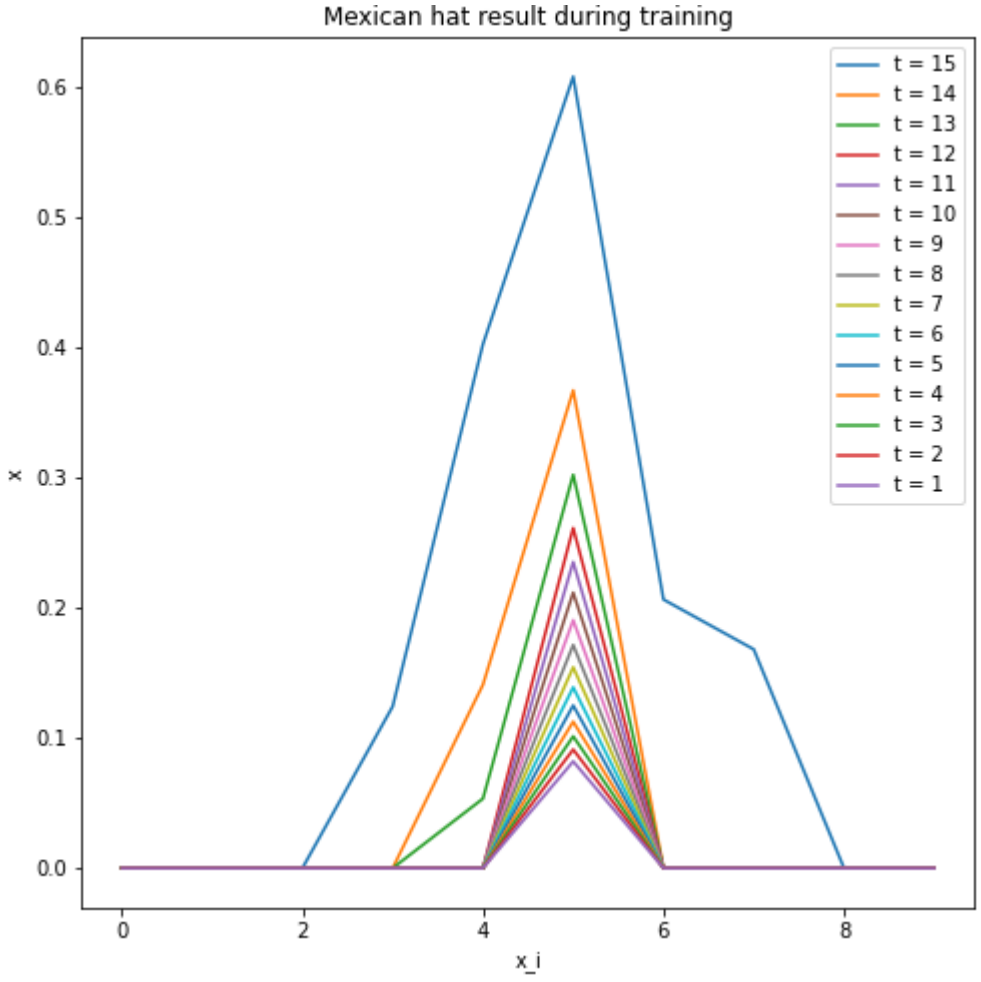
می توان نتیجه گرفت که به ترتیب بزرگی درایه چهارم، اول، دوم و پنجم، ششم، سوم قرار دارند پس بزرگترین عنصر درایه چهارم است. این نتیجه منطبق با اعداد ورودی است و ۱.۵ که عنصر چهارم است بزرگترین است.

حال در قسمت دوم این سوال می خواهیم الگوریتمی طراحی کنیم که ماکزیمم قدر مطلق اعداد برداری را مشخص کند. برای این کار به سادگی یکبار روی اعداد بردار MaxNet می زنیم و درایه ای که عدد آن از همه بزرگتر است را مشخص می کنیم. سپس همه ی اعداد را در 1- ضرب می کنیم و روی بردار حاصل نیز MaxNet می زنیم و بزرگترین درایه از بردار جدید را نیز پیدا می کنیم سپس بین مقدار درایه بزرگترین عدد حالت اول و دوم ماکسیمم می گیریم و عدد حاصل ماکسیمم قدر مطلق همه ی اعداد است. الگوریتم گفته شده در مرحله اول ۱.۲ را پیدا می کند سپس در مرحله دوم ۱.۵ را می یابد و در انتها ۱.۵ را به عنوان ماکسیمم قدر مطلق این اعداد بر می گرداند.

سوال ۳ – Mexican Hat

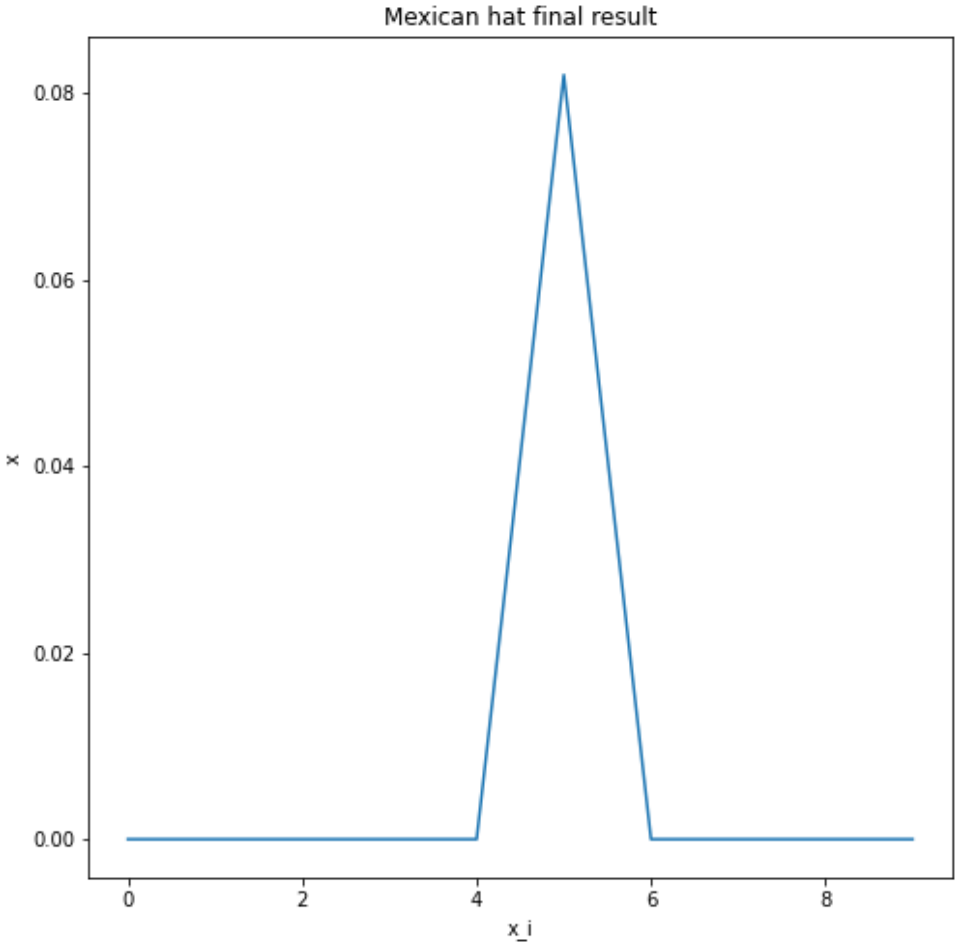
در این سوال الگوریتم Mexican Hat را پیاده سازی کرده و آن را برای دو جفت شعاع مختلف امتحان می کنیم. از این ساختار و الگوریتم برای یافتن مقدار حداکثر یک لیست و مجاورتش استفاده می کنیم. کد این سوال در فایل HW4_Q3.ipynb موجود است.

در حالت اول مقدار R1 را برابر ۰ و مقدار R2 را برابر ۱۰۰۰ یا بی نهایت برای این مجموعه داده می گذارم و داده های مدنظر نیز برابر x = [0.32, 0.33, 0.28, 0.47, 0.66, 0.80, 0.4, 0.33, 0.1 , 0.26] هستند. حداکثر iteration را برابر ۱۵ قرار دادم و تنها تعیین مقادیر C1 و C2 باقی می ماند که سعی کردم مقادیری شبیه سورس قرار دهم و با اندکی آزمون و خطا مقادیری مناسب پیدا کنم که 0.9 و 0.2- را انتخاب کردم. سپس نمودار تغییرات را برای این حالت رسم کردم که به شکل زیر است و می توان مصداق Mexican hat که عنوان این الگوریتم است را مشاهده کرد:



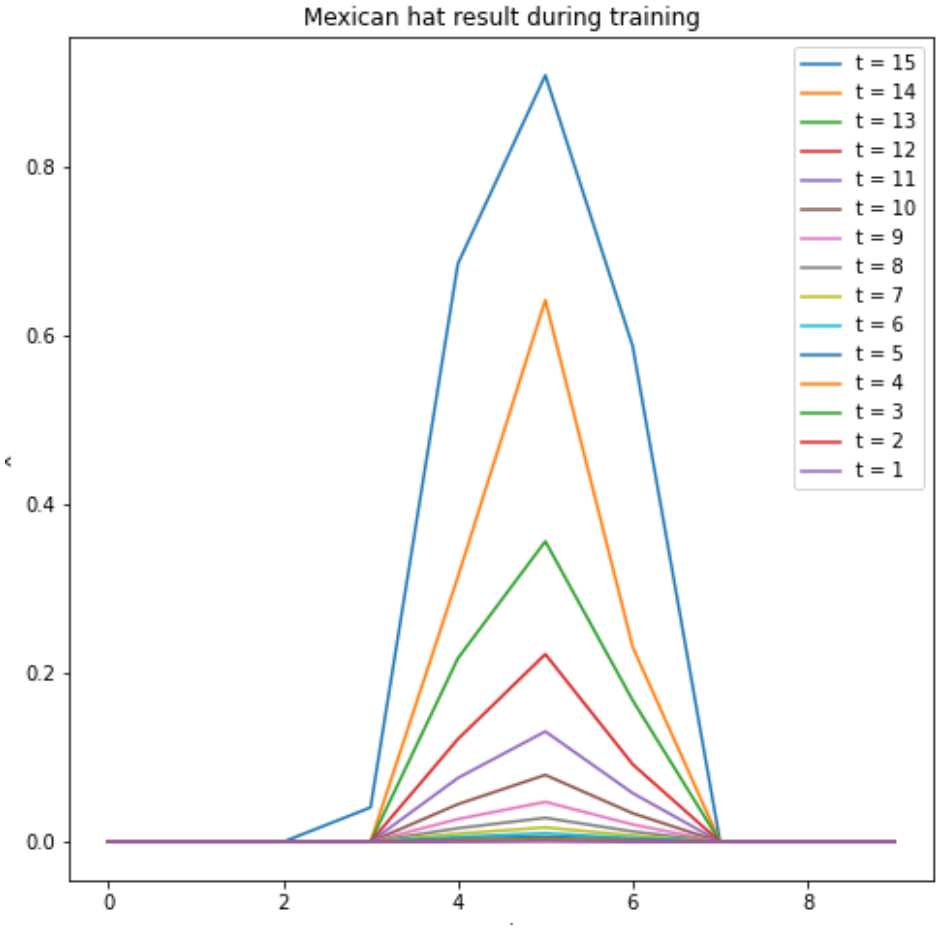
شکل ۴ : سیر تغییرات اعداد در حالت اول الگوریتم

و حالت نهایی بردار اعداد نیز به شکل زیر است:

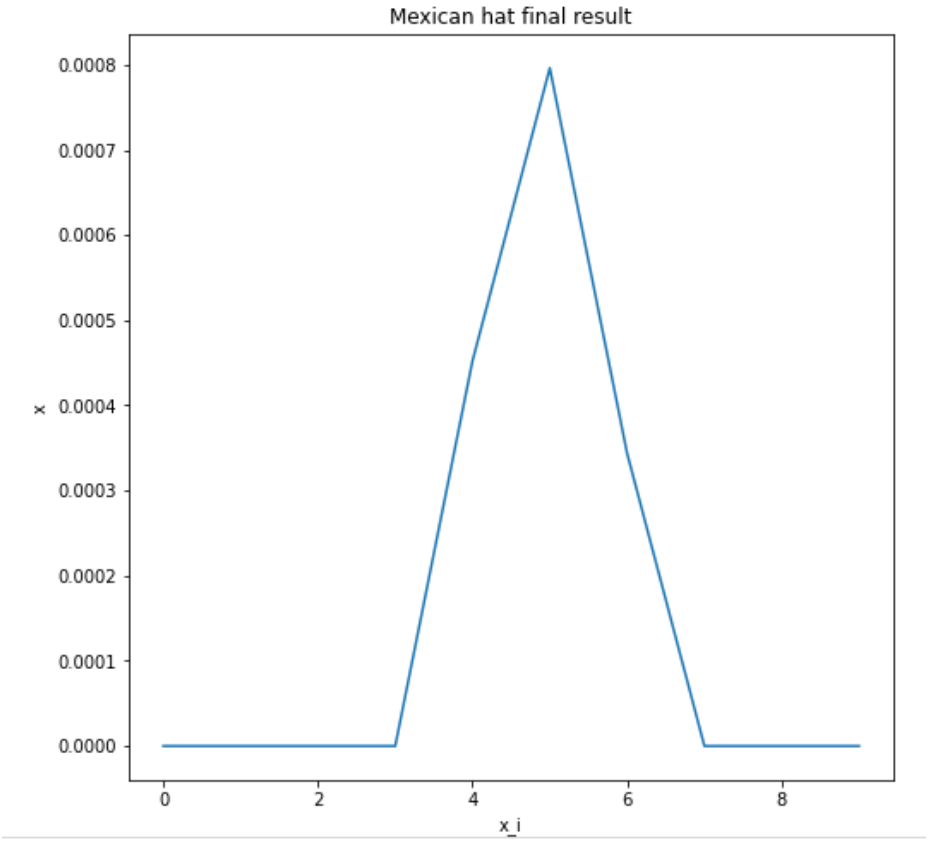


شکل ۵ : خروجی الگوریتم در حالت اول

می توان مشاهده کرد که عدد پنجم به عنوان خروجی مشخص شده است که جواب درست است و می بایست این جواب را می گرفتیم. حال برای حالت دوم الگوریتم را با همان تعداد iteration انجام می دهیم و R1 و R2 را برابر ۱ و ۵ قرار می دهیم و مقدار C1 و C2 را نیز با آزمون و خطا مقادیر 0.3 و 0.3- قرار می دهیم و نتیجه حاصل به شکل زیر است:



شکل ۶ : سیر تغییرات اعداد در حالت دوم الگوریتم



شکل ۷ : خروجی الگوریتم در حالت دوم الگوریتم

می توان دید که باز هم الگوریتم خوب عمل کرده است و از حالت قبل راحت تر به جواب رسیده است و مقدار خانه پنجم یا همان ۰.۸ که ماکسیمم است را پیدا می کند. همچنین شکل کلی Mexican hat نیز دوباره قابل رویت است.

سوال ۴ – Hamming Net

در این سوال الگوریتم Hamming Net را پیاده سازی کرده و ۳ بردار ۶ بعدی را به یاد سپرده سپس ۵ تا بردار را به عنوان ورودی به این شبکه می دهیم و از بین سه بردار قبلی نزدیک ترین به هر یک را پیدا می کنیم. کد این سوال در فایل HW4_Q4.ipynb موجود است.

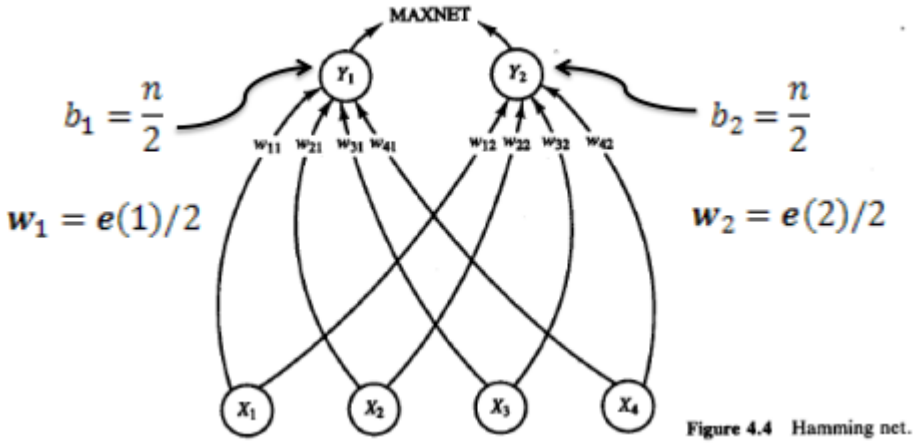
الف) در شبکه Hamming Net یک سری "بردار مرجع" تعریف کرده سپس این شبکه سعی می کند با استفاده از Hamming Distance Index هر بردار جدید ورودی را به یکی از بردار های مرجع که از بقیه نزدیک تر است، نسبت دهد. بردار های مرجع ما در این سوال به شکل زیر هستند:

$$e1 = [1, -1, 1, -1, 1, -1]$$
$$e2 = [-1, 1, -1, 1, -1, -1]$$
$$e3 = [1, 1, 1, -1, -1, -1]$$

اگر a را برابر تعداد عناصر یکسان دو بردار و d را برابر تعداد عناصر نابرابر دو بردار x و y که bipolar هستند، باشد داریم:

$$x \cdot y = a - d$$
$$d = n - a$$
$$a = x\left(\frac{y}{2}\right) + \left(\frac{n}{2}\right)$$

به کمک یک نورون mac clutch pitts می توان معادله ی بالا را پیاده کرد و شکل کلی معماری این شبکه به صورت زیر خواهد بود:



شکل ۸ : معماری شبکه Hamming Net

ب) حال شبکه خواسته شده را پیاده سازی می کنیم و نتیجه زیر حاصل می شود:

```
The closest vector to V1 is vector e3.
The closest vector to V2 is vector e2.
The closest vector to V3 is vector e1.
The closest vector to V4 is vector e2.
The closest vector to V5 is vector e3.
```

شکل ۹ : خروجی شبکه hamming net برای بردار ها

می توان دید که شبکه به خوبی کار کرده است و همه ی بردار ها را به درستی به نزدیک ترین بردار مرجع مرتبط کرده است که در زیر این نکته را بررسی کرده ایم:

V1= [1, 1, 1, 1, 1, 1]

Distance to (e1 = [1, -1, 1, -1, 1, -1]) is 3
Distance to (e2 = [-1, 1, -1, 1, -1, -1]) is 4
Distance to (e3 = [1, 1, 1, -1, -1, -1]) is 3

که شبکه ما به بردار سوم ربط داده است که درست است.

V2 = [-1, 1, -1, -1, 1, 1]
Distance to (e1 = [1, -1, 1, -1, 1, -1]) is 4
Distance to (e2 = [-1, 1, -1, 1, -1, -1]) is 3
Distance to (e3 = [1, 1, 1, -1, -1, -1]) is 4

که شبکه ما به بردار دوم ربط داده است که درست است.

V3 = [-1, -1, 1, 1, 1, 1]
Distance to (e1 = [1, -1, 1, -1, 1, -1]) is 3
Distance to (e2 = [-1, 1, -1, 1, -1, -1]) is 4
Distance to (e3 = [1, 1, 1, -1, -1, -1]) is 5

که شبکه ما به بردار اول ربط داده است که درست است.

V4 = [-1, -1, 1, 1, -1, 1]
Distance to (e1 = [1, -1, 1, -1, 1, -1]) is 4
Distance to (e2 = [-1, 1, -1, 1, -1, -1]) is 3
Distance to (e3 = [1, 1, 1, -1, -1, -1]) is 4

که شبکه ما به بردار دوم ربط داده است که درست است.

V5 = [-1, 1, 1, -1, -1, -1]
Distance to (e1 = [1, -1, 1, -1, 1, -1]) is 3
Distance to (e2 = [-1, 1, -1, 1, -1, -1]) is 2
Distance to (e3 = [1, 1, 1, -1, -1, -1]) is 2

که شبکه ما به بردار سوم ربط داده است که درست است.