به نام خدا

دانشگاه تهران

دانشکده مهندسی برق و کامپیوتر

شبکه های عصبی و یادگیری عمیق

تمرین سری چهارم

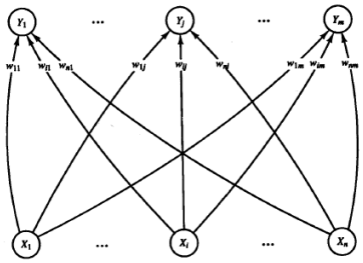
نام و نام خانوادگی	**محمدعلی شاکردرگاه**
شماره دانشجویی	810196487
تاریخ ارسال گزارش	1400/04/18

فهرست گزارش سوالات

سوال 1 – SOM

در این سوال با شبکه Self Organizing Map کار خواهیم کرد که به اختصار به آن SOM گفته میشود.

این شبکه تمام ورودی های n بعدی را به m طبقه بند تقسیم میکند. در این شبکه، هر نورون نماینده یک خوشه بند میباشد، به عبارتی هر نورون یک فرم همسایگی دارد.



شکل 1-1 ساختار کلی SOM

در این مساله از داده های MNIST استفاده خواهیم کرد که به ما تصاویری 28*28 از اعداد 0 تا 9 انگلیسی میدهد، مقصود ما آن است که این ها را در 10 کلاس خوشه بندی نماییم.با توجه به شبکه گفته شده در سوال، لایه خروجی شبکه SOM میبایست دارای 625 نورون باشد.

برای داده های آموزش و تست، به ترتیب 2000 داده و 1000 داده از MNIST در مجموعه داده های Keras بارگزاری میکنیم و از آن ها برای آموزش شبکه و ارزیابی عملکرد آن استفاده خواهیم کرد.

حل مسئله – الف:

در قسمت الف، از ما خواسته شده است شبکه SOM را در حالتی برای آموزش بسازیم که شعاع مجاورت برای R برابر با 0 باشد. یعنی هر نورون به تنهایی و بدون مجاورت نورون های دیگر.

مراحل زیر را طی میکنیم:

0- مقدار دهی اولیه:

در ابتدا میبایست بردار وزن را تعریف کنیم، اگر ورودی n بعدی (ابعاد تصویر داده) و خروجی ما m بعدی (تعداد نورون های خواسته شده) باشد، آنگاه بردار بردار وزن ما شامل n ردیف و m ستون خواهد بود و برای تعریف اولیه آن، از انتخاب اعداد به صورت تصادفی استفاده میکنیم، در اینجا توپولوژی همسایگی به علت صفر بودن شعاع مجاورت، نخواهیم داشت و در نهایت نرخ یادگیری خودمان را انتخاب مینماییم.

1- آموزش

به شیوه Gradient Based وزن های خود را بروز مینماییم بدن صورت که تا قبل از شرایط توقف، برای هر ورودی x_i ای که داریم می آییم و بردار D را همانند زیر میسازیم:

$$D(j) = \sum_i \left(W_{ij} - x_i\right)^2$$

بدین ترتیب بردار D بدست می آید، حال اندیس مقدار مینیمم آن را در J ثبت میکنیم و برای تمام همسایه های J با شعاع همسایگی R (که در اینجا صفر میباشد)، بروز رسانی وزن را به مانند زیر بروز میکنیم:

$$W_{ij}(new) = W(old) + \alpha(x_i - W_{ij}(old))$$

که در آن α نرخ آموزش است.

سپس، نرخ آموزش را به روزرسانی میکنیم (در این پیاده سازی، بروزرسانی نرخ آموزش از هندسی بروز میشود) و شعاع همسایگی را کاهش میدهیم.

اجرای این عملیات را 15 بار انجام میدهیم.

2- ارزیابی

حال کافیست بردار Y= x * W را بیابیم و ایندکس مینیمم آن را برای هر کدام از x ها بیابیم، این مینیمم درواقع مینیمم فاصله میباشد. این ایندکس همان نورونی از خروجی شبکه ما میباشد که کمترین فاصله را با داده x_i ما داشته است. حال سعی میکنیم برای تمام این نورون ها ببینیم هر کدام، چند تعداد از کلاس های اعداذ(0 تا 9) را در خود گنجانده اند بدین ترتیب ماتریس 10*625 این میسازیم که نماینده تمام 625 نورون میباشند که دارای در هر کدام تعداد تکرار کلاس 0 و 1 و ... و 9 را در خود ذخیره کرده اند. بیشترین فرکانس کلاس در هر نورون Y_pred ما را مشخص مینماید.

پس از انجام عملیات فوق، عملکرد دقت شبکه برای داده های تست در زیر آورده شده است:

```
cnt = 0
for i in range(len(y_test)):
  if labelOfNeuron[neuronMinDif_forXi[i]] == y_test[i]:
    cnt += 1
accuracy = (cnt/len(y_test))*100
print(f"Accuracy on test data is {accuracy}")
```

```
Accuracy on test data is 13.900000000000002
```

شکل 1-2 دقت شبکه برای داده های تست در حالت R=0

همانگونه که مشاهده میشود، دقت شبکه بسیار ضعیف میباشد، دلیل این امر آن است که شعاع همسایگی صفر انتخاب شده است و این نتیجه قابل انتظاری نیز بود.

حل مسئله – ب:

در قسمت ب، از ما خواسته شده است شبکه SOM را در حالتی برای آموزش بسازیم که شعاع مجاورت برای R برابر با 2 باشد.

مراحل زیر را طی میکنیم:

0- مقدار دهی اولیه:

در ابتدا میبایست بردار وزن را تعریف کنیم، اگر ورودی n بعدی (ابعاد تصویر داده) و خروجی ما m بعدی (تعداد نورون های خواسته شده) باشد، آنگاه بردار بردار وزن ما شامل n ردیف و m ستون خواهد بود و برای تعریف اولیه آن، از انتخاب اعداد به صورت تصادفی استفاده میکنیم، **در اینجا توپولوژی همسایگی را داریم و شعاع همسایگی را برابر با 2 قرار میدهیم** و در نهایت نرخ یادگیری خودمان را انتخاب مینماییم.

1- آموزش

به شیوه Gradient Based وزن های خود را بروز مینماییم بدن صورت که تا قبل از شرایط توقف، برای هر ورودی x_i ای که داریم می آییم و بردار D را همانند زیر میسازیم:

$$D(j) = \sum_i (W_{ij} - x_i)^2$$

بدین ترتیب بردار D بدست می آید، حال اندیس مقدار مینیمم آن را در J ثبت میکنیم و برای تمام همسایه های J با شعاع همسایگی R (که در اینجا 2 میباشد)، بروز رسانی وزن را به مانند زیر بروز میکنیم:

$$W_{ij}(new) = W(old) + \alpha(x_i - W_{ij}(old))$$

که در آن α نرخ آموزش است.

سپس، نرخ آموزش را به روزرسانی میکنیم (در این پیاده سازی، بروزرسانی نرخ آموزش از شیوه هندسی بروز میشود) و شعاع همسایگی را کاهش میدهیم.

اجرای این عملیات را 15 بار انجام میدهیم.

2- ارزیابی

حال کافیست بردار Y= x * W را بیابیم و ایندکس مینیمم آن را برای هر کدام از x ها بیابیم، این مینیمم درواقع مینیمم فاصله میباشد. این ایندکس همان نورونی از خروجی شبکه ما میباشد که کمترین فاصله را با داده x_i ما داشته است. حال سعی میکنیم برای تمام این نورون ها ببینیم هر کدام، چند تعداد از کلاس های اعداد (0 تا 9) را در خود گنجانده اند بدین ترتیب ماتریس 625*10 این میسازیم که نماینده تمام 625 نورون میباشند که دارای در هر کدام تعداد تکرار کلاس 0 و 1 و ... و 9 را در خود ذخیره کرده اند. بیشترین فرکانس کلاس در هر نورون Y_pred ما را مشخص مینماید.

پس از انجام عملیات فوق، عملکرد دقت شبکه برای داده های تست در زیر آورده شده است:

```
cnt = 0
for i in range(len(y_test)):
  if labelOfNeuron[neuronMinDif_forXi[i]] == y_test[i]:
    cnt += 1
accuracy = (cnt/len(y_test))*100
print(f"Accuracy on test data is {accuracy}")
```

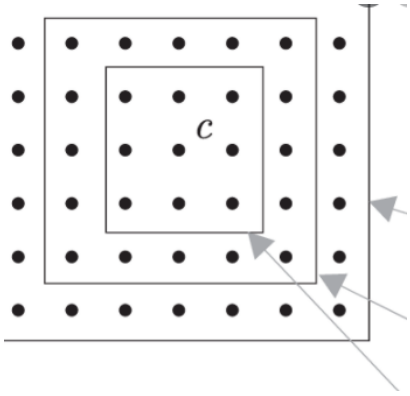
```
Accuracy on test data is 49.811100000000002
```

شکل 1-3 دقت شبکه برای داده های تست در حالت R=2

همانگونه که مشاهده میشود، دقت شبکه با شعاع همسایگی بالاتر، دقت بسیار بهبود پیدا کرده است نسبت به حالتی که شعاع همسایگی برابر با صفر بوده است.

حل مسئله – ج:

در قسمت ج، از ما خواسته شده است شبکه SOM را در حالتی برای آموزش بسازیم که **با یک فرم مربعی و با شعاع مجاورت 1** وزن ها به روز شوند. به مانند شکل زیر:



شکل 1-4 خوشه بندی با R مختلف در فرم مربعی

مراحل زیر را طی میکنیم:

0- مقدار دهی اولیه:

در ابتدا میبایست بردار وزن را تعریف کنیم، اگر ورودی n بعدی (ابعاد تصویر داده) و خروجی ما m بعدی (تعداد نورون های خواسته شده) باشد، آنگاه بردار بردار وزن ما شامل n ردیف و m ستون خواهد بود و برای تعریف اولیه آن، از انتخاب اعداد به صورت تصادفی استفاده میکنیم، **در اینجا توپولوژی همسایگی به صورت مربعی را داریم و شعاع همسایگی را برابر با 1 قرار میدهیم** و در نهایت نرخ یادگیری خودمان را انتخاب مینماییم

1- آموزش

به شیوه Gradient Based وزن های خود را بروز مینماییم بدن صورت که تا قبل از شرایط توقف، برای هر ورودی x_i ای که داریم می آییم و بردار D را همانند زیر میسازیم:

$$D(j) = \sum_i \left(W_{ij} - x_i\right)^2$$

بدین ترتیب بردار D بدست می آید، حال اندیس مقدار مینیمم آن را در J ثبت میکنیم و برای تمام همسایه های J با شعاع همسایگی R **(برای حالتی که فرم مربعی قرار دارد با مقدار 1)، بروز رسانی وزن را به مانند برای حالاتی که در شعاع همسایگی (که به صورت مربعی در نظر گرفتیم:**

$$W_{ij}(new) = W(old) + \alpha(x_i - W_{ij}(old))$$

که در آن α نرخ آموزش است.

سپس، نرخ آموزش را به روزرسانی میکنیم (در این پیاده سازی، بروزرسانی نرخ آموزش از شیوه هندسی بروز میشود) و شعاع همسایگی را کاهش میدهیم. **در این روند آموزش، فاصله اقلیدسی وزن ها با الگوی تصاویر ورودی کمینه خواهد شد.** اجرای این عملیات را 15 بار انجام میدهیم.

2- ارزیابی

حال کافیست بردار Y= x * W را بیابیم و ایندکس مینیمم آن را برای هر کدام از x ها بیابیم، این مینیمم درواقع مینیمم فاصله میباشد. این ایندکس همان نورونی از خروجی شبکه ما میباشد که کمترین فاصله را با داده x_i ما داشته است. حال سعی میکنیم برای تمام این نورون ها ببینیم هر کدام، چند تعداد از کلاس های اعداد (0 تا 9) را در خود گنجانده اند بدین ترتیب ماتریس 625*10 این میسازیم که نماینده تمام 625 نورون میباشند که دارای در هر کدام تعداد تکرار کلاس 0 و 1 و ... و 9 را در خود ذخیره کرده اند. بیشترین فرکانس کلاس در هر نورون Y_pred ما را مشخص مینماید.

پس از انجام عملیات فوق، عملکرد دقت شبکه برای داده های تست در زیر آورده شده است:

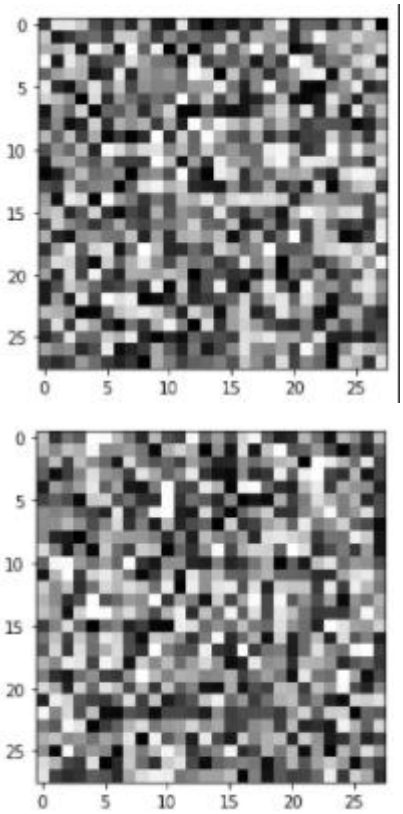
```
cnt = 0
for i in range(len(y_test)):
  if labelOfNeuron[neuronMinDif_forXi[i]] == y_test[i]:
    cnt += 1
accuracy = (cnt/len(y_test))*100
print(f"Accuracy on test data is {accuracy}")
```

```
Accuracy on test data is 51.311
```

شکل 1-5 دقت شبکه برای داده های تست در حالت همسایگی مربعی با R = 1

که مشاهده میشود، دقت شبکه با اینگونه شعاع همسایگی، دقت بهبود پیدا کرده است نسبت به حالت های دیگر.

همچنین دو عکس از وبردار وزن رسم شده است که ابعادش 28*28 است:



شکل1-6 دو بردار وزن برای حالت شعاع همسایگی 1

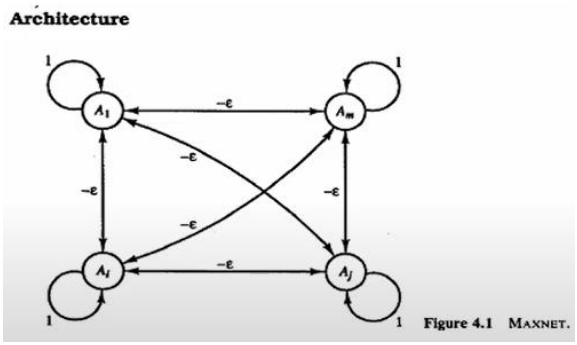
همانگونه که قابل مشاهده میباشد، این دو بردار، از وزن، آنچنان که باید شبیه به اعداد نیستند ولی توانایی بهتری برای تشخیص و خوشه بندی کردن ارائه دده اند. این در حالی است که اگر شعاع همسایگی را افزایش میدادیم، میتوانستیم خیلی بهتر ویژگی ها و تفاوتشان را دریابیم و بهتر خوشه بندی کنیم. پس دلیل بسیار شبیه به عدد نبودن این دو بردار از ماتریس وزن، به همین دلیل است.

سوال 2 – MaxNet

در این سوال به بررسی عملکرد شبکه MaxNet میپردازیم که یک شبکه تک لایه Fixed-Weight Competetive Net که M تا نورون دارد میباشد. MaxNet مکانیزم یادگیری ای است که برای مشخص نمودن بزرگترین Node مطلق استفاده میشود.

اطلاعات کلی شبکه:

- ساختار این شبکه در شکل زیر قابل رویت میباشد:



شکل 2-1 ساختار شبکه MaxNet

- این شبکه از تابع فعال ساز ReLU بهره میبرد.
- ماتریس وزن و آستاه به مانند زیر میباشد:

$$w_{ij} = \begin{cases} 1 & if\ i = j \\ -\epsilon & if\ i = j \end{cases}$$

که در آن $0 < \epsilon < \frac{1}{m}$

- قانون به روز رسانی نیز در این شبکه به مانند زیر خواهد بود:

$$a_i(new) = f[\, a_i(old) - \epsilon \sum_{k \neq i} a_k(old)]$$

$$\boldsymbol{a}^{new} = \boldsymbol{f}\left(\begin{bmatrix} 1 & -\varepsilon & \cdots & -\varepsilon \\ -\varepsilon & 1 & \cdots & -\varepsilon \\ \vdots & \vdots & \ddots & \vdots \\ -\varepsilon & -\varepsilon & \cdots & 1 \end{bmatrix} \boldsymbol{a}^{old}\right)$$

شکل 2-2 به روز رسانی در شبکه MaxNet

حل مسئله – قسمت اول :

با استفاده از شبکه MaxNet سعی میکنیم مقدا بیشینه بردار زیر را بیابیم:

$$x = [1.2, 1.1, 0.5, 1.5, 1.13, 0.8]$$

همچنین مراحل پیشروی الگوریتم و از دور خارج شدن نود های بازنده را به نمایش بگذاریم.

مرحله 0:

- ابتدا سعی میکنیم مقدار ϵ را بین 0 و $\frac{1}{m}$ انتخاب کنیم که طبق فرض سوال آن را برابر با 0.13 میگیریم
- مقادر a_old را برابر با $[1.2, 1.1, 0.5, 1.5, 1.13, 0.8]$ میگذاریم
- بردار W را به مانند زیر بدست می آوریم:

```
Weight:
 [[ 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.  ]]
```

شکل 2-3 مقادیر ماتریس وزن

مرحله 1:

تا زمانی که به شرط خروج نرسیده ایم مراحل زیر را میپیماییم:

- ابتدا مقدار جدید بردار از طریق معادله زیر بدست می آوریم و از Actication function عبور میدهیم.

$$a_i(new) = f[\, a_i(old) - \epsilon \sum_{k \neq i} a_k(old)]$$

اگر فقط یکی از نود ها (درایه های بردار) غیر صفر مانده بود، برنامه را به پایان میرسانیم، در غیر این صورت، در غیر این صورت باز این کار را تکرار میکنیم:

نتیجه الگوریتم در زیر آمده است:

```
iteration =  0
  Res:  [1.2, 1.1, 0.5, 1.5, 1.13, 0.8]

iteration =  1
  Res:  [0.5460999999999999, 0.43310000000000004, 0, 0.8851, 0.46699999999999986, 0.09410000000000002]

iteration =  2
  Res:  [0.3017909999999999, 0.1741010000000001, 0, 0.684861, 0.21240799999999985, 0]

iteration =  3
  Res:  [0.16251289999999993, 0.018223200000000106, 0, 0.5953820000000001, 0.06151010999999984, 0]

iteration =  4
  Res:  [0.07474790969999995, 0, 0, 0.5638899927000001, 0, 0]

iteration =  5
  Res:  [0.0014422106489999365, 0, 0, 0.5541727644390001, 0, 0]

iteration =  6
  Res:  [0, 0, 0, 0.5539852770546301, 0, 0]
```

شکل 2-4 نتیجه شبکه MaxNet برای ورودی اول

همانطور که مشاهده میشود، شبکه توانست مقدار بیشینه (1.5) را به درستی نگه دارد و آن را برنده اعلام کند.

حل مسئله – قسمت دوم :

با استفاده از شبکه MaxNet سعی میکنیم مقدا **بیشینه مطلق** بردار زیر را بیابیم:

$$x = [1.2, 1.1, 0.5, -1.5, 1.13, -0.8]$$

عملکرد اصلی ما در این بخش آن است که یک بار برای بردار x می آییم و MaxNet را اجرا میکنیم و یکبار دیگر هم برای x- اینکار را انجام میدهیم، نتایج هرکدام از MaxNet ها، اندیس بزرگترین عدد در آن حالت را به ما میتواند بدهد، یک بردار دیگر هم سایز با x به نام MaxOfBothCon را پر از صفر میکنیم و سپس با با توجه به اندیسی که در هر حالت اول و دوم بدست آوردیم، از مقدار x و x- آن را پر میکنیم این به ما $[1.2, 0, 0, 1.5, 0, 0]$ میدهد، حال دوباره روی این بردار MaxNet میزنیم و بزرگترین مقدار را مشخص مینماییم.

مرحله 0:

- ابتدا سعی میکنیم مقدار ϵ را بین 0 و $\frac{1}{m}$ انتخاب کنیم که طبق فرض سوال آن را برابر با 0.13 میگیریم
- مقادر a_old را برابر با $[1.2, 1.1, 0.5, -1.5, 1.13, -0.8]$ میگذاریم
- بردار W را به مانند زیر بدست می آوریم:

```
Weight:
 [[ 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.  ]]
```

شکل 2-5 مقادیر ماتریس وزن

مرحله 1:

تا زمانی که به شرط خروج نرسیده ایم مراحل زیر را میپیماییم:

- ابتدا مقدار جدید بردار از طریق معادله زیر بدست می آوریم و از Actication function عبور میدهیم.

$$a_i(new) = f[\, a_i(old) - \epsilon \sum_{k \neq i} a_k(old)]$$

اگر فقط یکی از نود ها (درایه های بردار) غیر صفر مانده بود، برنامه را به پایان میرسانیم، در غیر این صورت، در غیر این صورت باز این کار را تکرار میکنیم:

- برای x :

```
..................... First Part ............................
iteration =  0
  Res:  [1.2, 1.1, 0.5, -1.5, 1.13, -0.8]

iteration =  1
  Res:  [1.1441, 1.0311000000000003, 0.3531000000000001, 0, 1.065, 0]

iteration =  2
  Res:  [0.8257039999999999, 0.6980140000000004, 0, 0, 0.7363209999999999, 0]

iteration =  3
  Res:  [0.6392404499999998, 0.4949507500000004, 0, 0, 0.5382376599999998, 0]

iteration =  4
  Res:  [0.5049259566999998, 0.34187859570000045, 0, 0, 0.39079280399999977, 0]

iteration =  5
  Res:  [0.40967867473899977, 0.2254351568090005, 0, 0, 0.28070821218799974, 0]

iteration =  6
  Res:  [0.3438800367693897, 0.13568486150849057, 0, 0, 0.1981434140867597, 0]

iteration =  7
  Res:  [0.3004823609420072, 0.06522181289719114, 0, 0, 0.13579997731063526, 0]

iteration =  8
  Res:  [0.2743495282149898, 0.008505108924347624, 0, 0, 0.08825843471153946, 0]

iteration =  9
  Res:  [0.2617702675423245, 0, 0, 0, 0.051487331883425594, 0]

iteration =  10
  Res:  [0.2550769143974792, 0, 0, 0, 0.017457197102923407, 0]

iteration =  11
  Res:  [0.25280747877409915, 0, 0, 0, 0, 0]
```

شکل 2-6 نتیجه شبکه MaxNet برای x

- برای x- :

```
..................... Second Part ..........................
iteration =  0
  Res:  [-1.2, -1.1, -0.5, 1.5, -1.13, 0.8]

iteration =  1
  Res:  [0, 0, 0, 1.9068999999999998, 0, 1.1159]

iteration =  2
  Res:  [0, 0, 0, 1.7618329999999998, 0, 0.8680029999999999]

iteration =  3
  Res:  [0, 0, 0, 1.6489926099999999, 0, 0.6389647099999999]

iteration =  4
  Res:  [0, 0, 0, 1.5659271976999998, 0, 0.4245956706999999]

iteration =  5
  Res:  [0, 0, 0, 1.5107297605089998, 0, 0.22102513499899995]

iteration =  6
  Res:  [0, 0, 0, 1.4819964929591298, 0, 0.02463026613282998]

iteration =  7
  Res:  [0, 0, 0, 1.478794558361862, 0, 0]
```

شکل 2-7 نتیجه شبکه MaxNet برای x-

- برای MaxOfBoth : که نتیجه نهایی را برای ما معین میسازد

```
..................... Final Part ..........................
iteration =  0
  Res:  [1.2, 0, 0, 1.5, 0, 0]

iteration =  1
  Res:  [1.005, 0, 0, 1.344, 0, 0]

iteration =  2
  Res:  [0, 0, 0, 1.478794558361862, 0, 0]
```

شکل 2-8 نتیجه شبکه MaxNet برای MaxOfBoth (نتیجه نهایی)

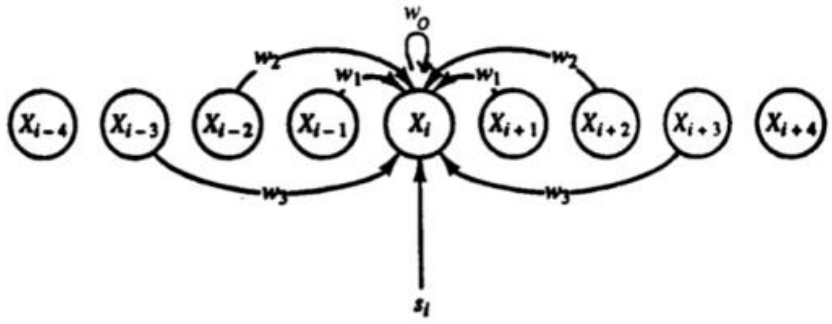
همانطور که مشاهده میشود در نهایت، شبکه توانست مقدار بیشینه (1.5) را به درستی نگه دارد و آن را برنده اعلام کند.

سوال 3 – Mexican Hat

در این سوال به بررسی عملکرد شبکه Mexican Hat میپردازیم که یک Fixed-Weight Competetive Net میباشد. Mexican Hat مکانیزم یادگیری ای است که برای مشخص نمودن بزرگترین Node نرم استفاده میشود.

اطلاعات کلی شبکه:

- ساختار ارتباطات درونی این شبکه در شکل زیر قابل رویت میباشد:



شکل 3-1 ساختار ارتباطات درونی شبکه Mexican Hat

- این شبکه از تابع فعال ساز زیر بهره میبرد:

$$f = \begin{cases} 0 & if\ x < 0 \\ x & if\ 0 \le x \le 2 \\ 2 & if\ 2 < x \end{cases} \quad -$$

در این شبکه اگر پارامتر های زیر را در نظر بگیریم:

R_2	Radius of region of interconnections; X_i is connected to units X_{i+k} and X_{i-k} for $k = 1, \ldots, R_2$.
R_1	Radius of region with positive reinforcement; $R_1 < R_2$.
w_k	Weight on interconnections between X_i and units X_{i+k} and X_{i-k}: w_k is positive for $0 \le k \le R_1$, w_k is negative for $R_1 < k \le R_2$.

x	Vector of activations.
x_old	Vector of activations at previous time step.
t_max	Total number of iterations of contrast enhancement.
s	External signal.

شکل 3-2 پارامتر های الگوریتم شبکه Mexican Hat

آنگاه برای این شبکه الگوریتم زیر قابل اجرا خواهید بود:

Step 0. Initialize parameters t_max, R_1, R_2 as desired.
Initialize weights:

$w_k = C_1$ for $k = 0, \ldots, R_1$ $(C_1 > 0)$

$w_k = C_2$ for $k = R_1 + 1, \ldots, R_2$ $(C_2 < 0)$.

Initialize **x_old** to **0**.

Step 1. Present external signal **s**:

$\mathbf{x} = \mathbf{s}$.

Save activations in array **x_old** (for $i = 1, \ldots, n$):

$x\_old_i = x_i$.

Set iteration counter: $t = 1$.

Step 2. While t is less than t_max, do Steps 3–7.

Step 3. Compute net input ($i = 1, \ldots, n$):

$$x_i = C_1 \sum_{k=-R_1}^{R_1} x\_old_{i+k} + C_2 \sum_{k=-R_2}^{-R_1-1} x\_old_{i+k} + C_2 \sum_{k=R_1+1}^{R_2} x\_old_{i+k}.$$

Step 4. Apply activation function (ramp function from 0 to x_max, slope 1):

$x_i = \min(x\_max, \max(0, x_i))$ $(i = 1, \ldots, n)$.

Step 5. Save current activations in **x_old**:

$x\_old_i = x_i$ $(i = 1, \ldots, n)$.

Step 6. Increment iteration counter:

$t = t + 1$.

Step 7. Test stopping condition:
If $t < t\_max$, continue; otherwise, stop.

شکل 3-3 الگوریتم در Mexican Hat

حل مسئله:

در این سوال، برداری که میبایست عملیات ماکسیمم یابی را روی آن اجرا نماییم ب صورت زیر میباشد:

X = [0.32, 0.33, 0.28, 0.47, 0.66, 0.80, 0.4, 0.33, 0.1, 0.26]

طبق صورت مسئله دو حالت برای شعاع های همکاری و رقابت را برمیگزینیم. این دو حالت در زیر آمده اند:

R1 = 0 & R2 = inf -

R1 = 1 & R2 = 5 -

سعی میکنیم در هر دو حالت اشاره شده، الگوریتم را پیاده سازی کنیم.

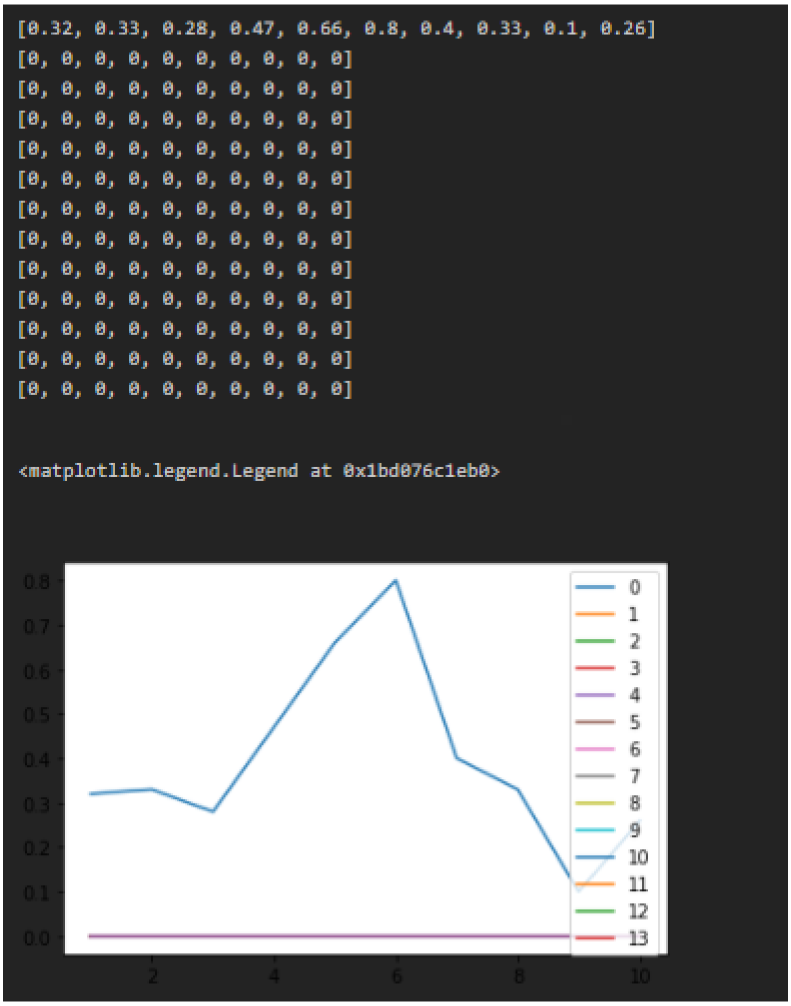
در این مساله، میبایست مقادیری برای C1 و C2 که وزن های همکاری و رقابت هستند را انتخاب کنیم که همانند کتاب عمل میکنیم و C1 = 0.6 و C2 = -0.4 انتخاب میکنیم.

حال طبق الگوریتم عمل میکنیم، در حالت اول شعاع های همکاری و رقابت را به ترتیب 0 و عددی بزرگ، و در حالت دوم شعاع های همکاری و رقابت را به ترتیب 1 و 5 قرار میدهیم، همانطو که اشاره شد، برای هر دو حالت، ما وزن های همکاری و رقابت را همانند کتاب برمیگزینیم و C1 = 0.6 و C2 = -0.4 انتخاب میکنیم. طبق فرض مسئله، ما مقدار x_max را برابر با 2 قرار میدهیم و برای آنکه حاصل جمع، Iterable باشد، نیازمند آن خواهیم بود که به دو طرف x مان که به سراغ بروزرسانی رفته است، به اندازه R2 (چرا که

R2 بزرگتر از R1 میباشد) 0 اضافه کنیم، بدین صورت به راحتی میتوانیم طبق Bound های مجموع های ذکر شده در الگوریتم، x را بروز رسانی کنیم و انتخاب نهایی را با توجه Boumd داده شده در تابع فعال سازی انجام دهیم که بدین صورت خواهد بود که ما بین x_max که در این مساله 2 میباشد و ماکزیمم 0 و x_i بروز شده، مینیمم میگیریم و آن را در x_i قرار میدهیم. حال اینکار را t_max بار در یک حلقه انجام میدهیم، تا بتوانیم به نتیجه برسیم.

حالت اول: R2 = inf   ,   R1 = 0

نتیجه برای این حالت در تصویر زیر آمده است:

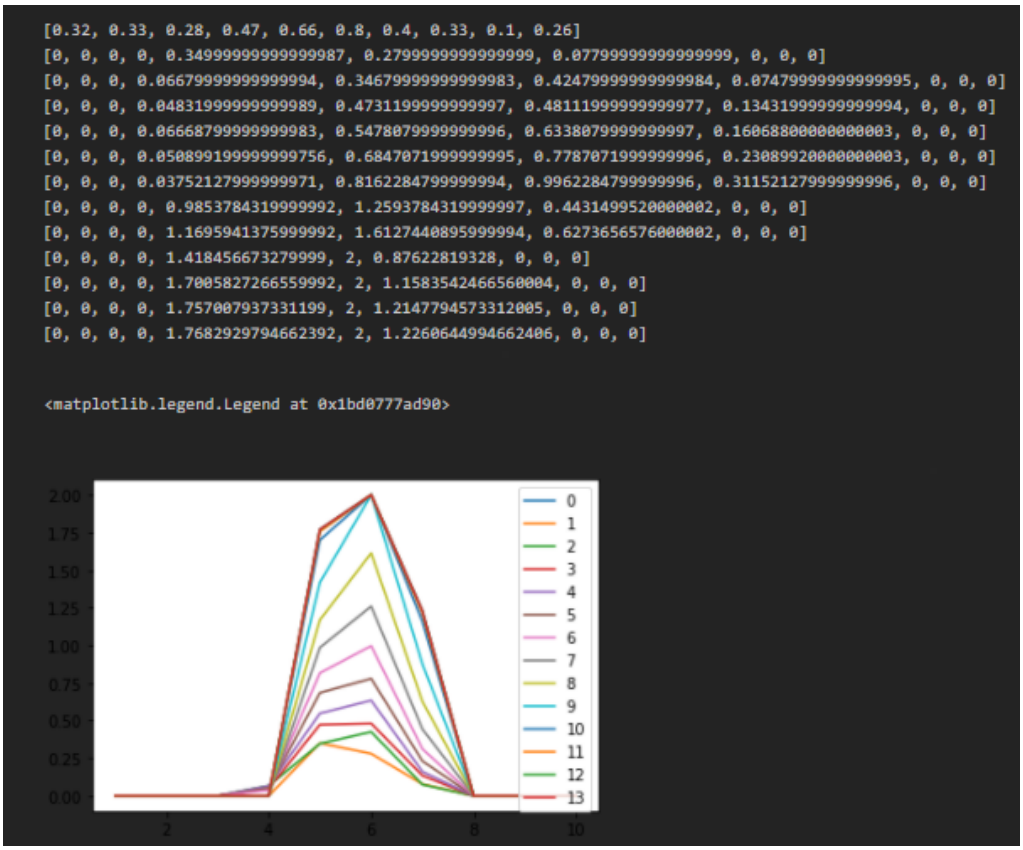


شکل 3-4 نتیجه در Mexican Hat برای حالت اول

همانطور که مشاهده مینماییم، با انتخاب کردن مقداری بزرگ برای شعاع رقابت R2 (در اینجا سایز بردار x هم کافی است) و صفر قرار دادن شعاع همکاری R1 مشاهده مینماییم که شبکه قدرتش را به طور کلی از دست داده است و نتیجه x به روزرسانی شده، بعد از یک بار اجرای الگوریتم 0 شده است.

حالت دوم: R1 = 1 , R2 = 5

نتیجه برای این حالت در تصویر زیر آمده است:



شکل 3-5 نتیجه در Mexican Hat برای حالت دوم

همانطور که قابل مشاهده میباشد، با انتخاب کردن مقدار 5 برای شعاع رقابت R2 و 1 قرار دادن شعاع همکاری R1 مشاهده مینماییم که عملکرد شبکه بسیار مطلوب میباشد و در طی Iteration های قابل رویت در بالا، علاوه بر Local maximumهایی که قابل رویت است، اندیس ماکزیمم مقدار را که 0.8 بوده است را یافته و در نوک قله در سیگنال خروجی نشان داده است.

سوال 4 – Hamming Net

در این سوال به بررسی عملکرد شبکه Hamming Net میپردازیم که یک Fixed-Weight Competetive Net میباشد. Hamming Net مکانیزم یادگیری ای است که برای assign کردن بردار هایی به بردار های مرجعمان استفاده میشود.

حل مسئله – الف)

در این بخش از پاسخ به سوال چهارم، به بررسی شبکه Hamming Net میپردازیم و عملکرد آن را شرح میدهیم، این شبکه در سال 1987 معرفی شد و عملکرد اصلی این شبکه بدین گونه است که یک سری "بردار های مرجع" تعریف شده وجود دارند و این شبکه سعی میکند با استفاده از Hamming Distance Index بتواند هر بردار جدیدی را به یکی از بدار های مرجع Assign کند که کمترین Hamming Distance را داشته باشند.

بردار های مرجع ما در این سوال به مانند مقابل هستند:

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

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

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

اگر تعریف کنیم که a برابر با تعداد بیت های برابر و d تعداد بردار های نابرابر برای دو بردار Bipolar x و y باشد، آنگاه

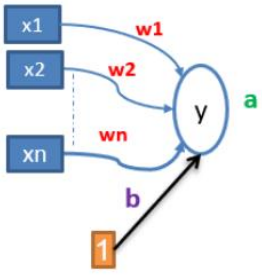
$$x \cdot y = a - d$$

$$d = n - a$$

خواهد بود و معادله به صورت زیر را خواهیم داشت:

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

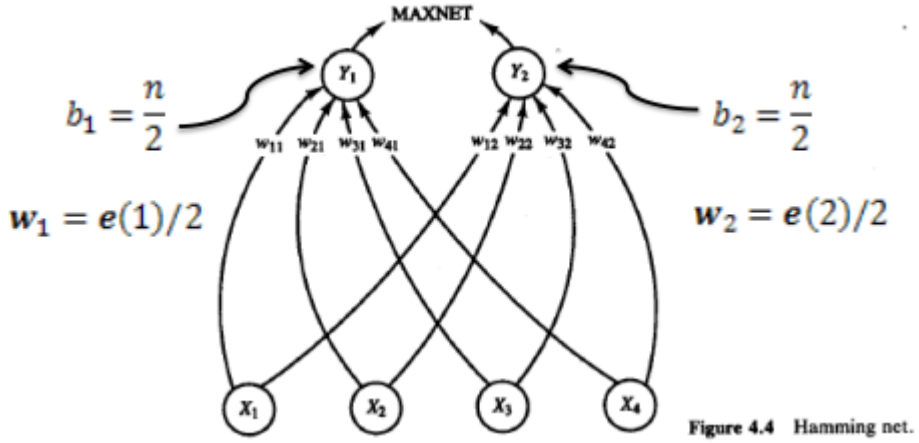
حال کافی است با داشتن یک نورون مکولاش-پیتز میتوان فرمول بالا را برای a پیاده سازی و تداعی کرد به این صورت که $\frac{y}{2}$ برابر با بردار وزن میشود و $\frac{n}{2}$ برابر با بردار بایاس میشود، شکل این تداعی را در زیر میتوان مشاهده نمود:



شکل 4-1 نورون مکولاش-پیتز برای تداعی شبکه *Hamming Net*

سپس میزان شباهت بردار x را با هرکدام از بردار های مرجع محاسبه مینماییم و این تشابهات را در یک بردار ذخیره مینماییم و به یک شبکه *MaxNet* میدهیم و بیشترین شباهت را با بردار مرجع مربوطه (کمترین فاصله *Hamming*) (یا همان بیشترین بیت های برابر) پیدا و اعلام مینماییم.

در شکل زیر، ساختار شبکه *Hamming Net* برای یک حالت 2 بردار مرجع آورده شده است:



شکل 4-2 *ساختار* کلی شبکه *Hamming Net* برای دو بردار مرجع

حل مسئله – ب)

در این بخش، به پیاده سازی این شبکه میپردازیم و بررسی میکنیم که بردار های داده شده، به چه گروهی تعلق خواهند گرفت. عمانطور که در قسمت قبل نیز اشاره شد، بردار های مرجع ما در این سوال به مانند مقابل هستند:

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

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

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

همچنین بردار های که میبایست بررسی شود به مانند زیر هستند:

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

$$V2 = [-1, 1, -1, -1, 1, 1]$$

$$V3 = [-1, -1, 1, 1, 1, 1]$$

$$V4 = [-1, -1, 1, 1, -1, 1]$$

$$V5 = [-1, 1, 1, -1, -1, -1]$$

برای پیاده سازی الگوریتم، ابتدا بردار وزن و بایاس را به همانگونه که توضیح داده شد، از طریق فرمول زیر بدست می آوریم:

$$W_{ij} = \frac{e_i(j)}{2}$$

$$b_j = \frac{n}{2}$$

سپس کافی است برای هر ورودی V_i سه گام را انجام دهیم:

1- تشکیل بردار Y از طریق فرمول زیر:

$$Y = [y_{in_1}, \dots, y_{in_m}]_{m*1} = b + xW^T$$

2- شبکه MaxNet را با بردار Y ، Initialize میکنیم.

3- شبکه MaxNet در طی Iteration ، بهترین match را برای ما انتخاب مینماید.

نتایج شبیه سازی در زیر آورده شده است:

وزن و بایاس بدست آمده:

```
W:  [[ 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]]
b:  [3.0, 3.0, 3.0]
```

شکل 4-3 وزن و بایاس شبکه HammingNet

- نتیجه برای بردار اول:

```
#V1
Y_1 = b + np.dot(V1, np.transpose(W))
print("Result of Y_1", Y_1)
resultOfMaxNet = MaxNet(Y_1)
print("indexes of e_i that has most similarity:", resultOfMaxNet)
```

```
Result of Y_1 [3. 2. 3.]
stopped after 20000 iterations, There are more than one maximum
indexes of e_i that has most similarity: [1, 3]
```

شکل 4-4 نتیجه شبکه HammingNet برای ورودی V1

همانطور که مشخص میباشد، دو بردار e1 و e3 با V1 در 3 بیت مشترک میباشند و بعد از 20000 تکرار، قسمت MaxNet این دو بردار را شبیه ترین به V1 اعلام نموده است.

- نتیجه برای بردار دوم:

```
#V2
Y_2 = b + np.dot(V2, np.transpose(W))
print("Result of Y_2", Y_2)
resultOfMaxNet = MaxNet(Y_2)
print("indexes of e_i that has most similarity:", resultOfMaxNet)
```

```
Result of Y_2 [2. 3. 2.]
indexes of e_i that has most similarity: [2]
```

شکل 4-5 نتیجه شبکه HammingNet برای ورودی V2

همانطور که مشخص میباشد، شبکه HammingNet ، بردار V2 را به e2 نسبت داده که با تحلیل هم متوجه تشخیص درست شبکه میباشیم چرا که V2 و e2 در 3 بیت با یکدیگر مشترکند، بیشتر از بردار های مرجع دیگر.

- نتیجه برای بردار سوم:

```
#V3
Y_3 = b + np.dot(V3, np.transpose(W))
print("Result of Y_3", Y_3)
resultOfMaxNet = MaxNet(Y_3)
print("indexes of e_i that has most similarity:", resultOfMaxNet)
```

```
Result of Y_3 [3. 2. 1.]
indexes of e_i that has most similarity: [1]
```

شکل 4-6 نتیجه شبکه HammingNet برای ورودی V3

همانطور که مشخص میباشد، شبکه HammingNet ، بردار V3 را به e1 نسبت داده که با تحلیل هم متوجه تشخیص درست شبکه میباشیم چرا که V3 و e1 در 3 بیت با یکدیگر مشترکند، بیشتر از بردار های مرجع دیگر.

- نتیجه برای بردار چهارم:

```
#V4
Y_4 = b + np.dot(V4, np.transpose(W))
print("Result of Y_4", Y_4)
resultOfMaxNet = MaxNet(Y_4)
print("indexes of e_i that has most similarity:", resultOfMaxNet)
```

```
Result of Y_4 [2. 3. 2.]
indexes of e_i that has most similarity: [2]
```

شکل 4-7 نتیجه شبکه HammingNet برای ورودی V4

همانطور که مشخص میباشد، شبکه HammingNet ، بردار V4 را به e2 نسبت داده که با تحلیل هم متوجه تشخیص درست شبکه میباشیم چرا که V4 و e2 در 3 بیت با یکدیگر مشترکند، بیشتر از بردار های مرجع دیگر.

- نتیجه برای بردار پنجم:

```
#V5
Y_5 = b + np.dot(V5, np.transpose(W))
print("Result of Y_4", Y_5)
resultOfMaxNet = MaxNet(Y_5)
print("indexes of e_i that has most similarity:", resultOfMaxNet)


 Result of Y_4 [3. 4. 5.]
 indexes of e_i that has most similarity: [3]
```

شکل 4-7 نتیجه شبکه HammingNet برای ورودی V5

همانطور که مشخص میباشد، شبکه HammingNet ، بردار V5 را به e3 نسبت داده که با تحلیل هم متوجه تشخیص درست شبکه میباشیم چرا که V5 و e3 در 5 بیت با یکدیگر مشترکند، بیشتر از بردار های مرجع دیگر.