به نام خدا

دانشگاه تهران

دانشکده مهندسی برق و کامپیوتر

شبکه های عصبی و یادگیری عمیق

تمرین سری چهارم

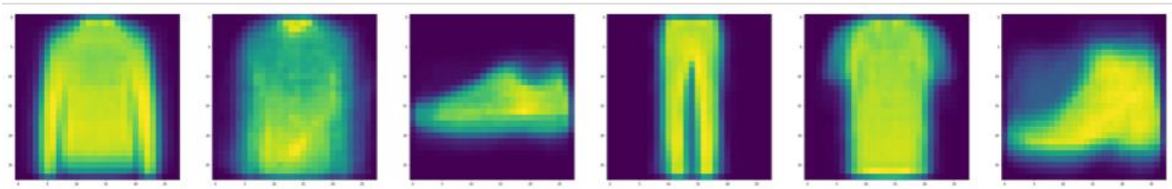
نام و نام خانوادگی	فاطمه سلیقه
شماره دانشجویی	۸۱۰۱۹۸۳۰۶
تاریخ ارسال گزارش	۹۹/۳/۲۹

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

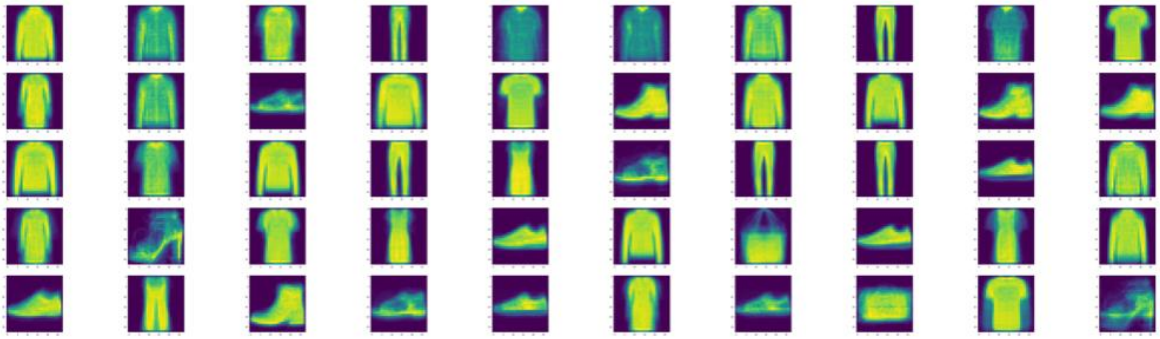
سوال ۱ – SOM

برای طبقه بندی داده های unsupervised می توان از SOM استفاده نمود . ۸۴۱ بردار وزن تصادفی با اندازه ۷۸۴ در نظر می گیریم (زیرا سایز عکس ها ۲۸ در ۲۸ است که آنها را به صورت با به بعد ۷۸۴ می بریم) و آنها را به صورت تصادفی بین 0,1 مقدار دهی می کنیم . هم چنین عکس ها را هم نرمالیزه کرده تا مقادیر بین 0,1پیدا کنند . سپس برای هر بردار maximum square error آن نسبت به هر بردار وزن را محاسبهمی نماییم هر کدام که کمتر بود را انتخاب نموده و به روز رسانی می کنیم . در حالت های اول و دوم و سوم سوال نحوه به روز رسانی تفاوت پیدا می کند . در حالت اول فقط بردار وزنی که کمترین فاصله را دارد به روز می شود در حالت ب اگر بردار وزن ها را روی یک خط بگیریم هر برداری که به روز می شود دو بردار همسایه اش هم به روز می شود . در حالت سوم بردار ها را روی یک مربع ۲۹در۲۹ در نظر می گیریم و زمانی که می خواهیم یک بردار را به روز کنیم ۴ بردار مجاورش را هم به روز می نماییم . در هر iteration هم مقدار learning rate را 0.8 برابر می کنیم . اگر این کار را انجام ندهیم ماتریس وزن به مقدار نامناسبی همگرا می شود . برای شرط توقف هم در نظر می گیریم که ماتریس وزن بیشتر از 0.001 تغییر نکند . یعنی اگر دو iteration پشت سر هم بیشترین میزان تغییر در ماتریس وزن کمتر از 0.001 شود شرط توقف true می شود .

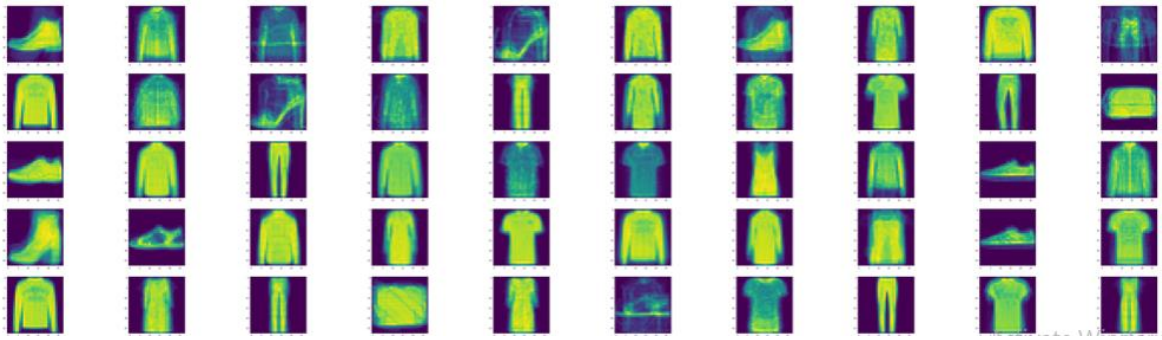
حالت اول :



حالت دوم :



حالت سوم :



همان طور که از نتایج به دست آمد تعداد نورون های چگال در حالت اول ۶ درحالت دوم ۷۲ و در حالت سوم ۳۸۴ تاست . حالت اول به خوبی عمل نکرده است چون ما می دانیم که ۱۰ کلاس مختلف داریم . اما چون ۶ نورون در نظر گرفته شده پس به خوبی کلاس بندی نشده و انواع مختلف پوشاک در یک کلاس قرار گرفته است . اما در حالت دوم و سوم بیش از ۱۰ نورون چگال داریم . هر دو تا حدی به خوبی عمل نموده اند و انواع مختلف پوشاک را نشان داده اند اما به نظر تنوع پوشاک در حالت سوم بیشتر از حالت دوم است یعنی بهتر توانسته نورون ها را به بخش های چگال تر ببرد . اما نکته ای که وجود دارد این است که در در چند نورون چگال تر هر حالت . تصاویر حالت دوم وضوح بیشتری نسبت به حالت سوم دارند . شاید به این دلیل است که در حالت دوم رقابت بیشتر از حالت سوم است در واقع در حالت دوم هر نورون به دو نورون مجاور همکاری کرده و با بقیه نورون ها رقابت می کند اما در حالت سوم ، هر نورون با چهار نورون همکاری نموده و با بقیه نورون ها رقابت می کند .

سوال ۲ – MaxNet

الگوریتم MaxNet پیاده سازی شده است و مراحل اجرای الگوریتم بر روی x به صورت زیر است :

```
[0.435  0.32   0.205  0.09   0.1475 0.3775]
[0.264    0.13175  0.       0.       0.       0.197875]
[0.21455625 0.06246875 0.         0.         0.         0.1385125 ]
[0.18440906 0.00950844 0.         0.         0.         0.09695875]
[0.16843898 0.         0.         0.         0.         0.06787112]
[0.15825832 0.         0.         0.         0.         0.04260528]
[0.15186752 0.         0.         0.         0.         0.01886653]
[0.14903754 0.         0.         0.         0.         0.        ]
```

از نتایج دیده می شود که هر بار از تمام نودها به اندازه ϵ برابر مجموع سایر نورون ها کم می کنیم . بنابراین برای کوچکترین نود چون مجموع سایر نودها بیشتر می شود بنابراین مقداری هم که از کوچکترین نورون کم می شود بیشتر از سایر نود هاست و از طرفی مقدار نود کوچکتر هم از همه کمتر است بنابراین نودها هر چه کوچکتر باشند زودتراز بقیه نودها به سمت 0 و مقادیر منفی میل پیدا می کنند . به همین دلیل نودی که از همه کوچکتر است ابتدا 0 می شود و به همین ترتیب از کوچک به بزرگ به استثنای بزرگترین نود 0 می شوند و در اخر بزرگترین نود باقی می ماند . همانطور که می بینیم به درستی توانست عدد 1.2 را به عنوان بزرگترین عدد انتخاب کند .

اما گاهی اوقات میشود که ممکن است چند نود با هم 0 شوند . زیرا مقدار آن ها با هم منفی شده در نتیجه با هم 0 می شوند . برای حل این مشکل می توان مقدار ϵ را کاهش داد . مثلا اگر 0.05 بگذاریم مشکل حل می شود . نتیجه به صورت زیر در می آید :

```
[0.945  0.84   0.735  0.63   0.6825 0.8925]
[0.756    0.64575  0.5355   0.42525  0.480375 0.700875]
[0.6166125  0.50085    0.3850875  0.269325   0.32720625 0.55873125]
[0.5145525  0.39300188 0.27145125 0.14990063 0.21067594 0.45377719]
[0.44061216 0.312984   0.18535584 0.05772769 0.12154177 0.37679808]
[0.38789179 0.25388222 0.11987266 0.         0.05286788 0.32088701]
[0.3505163  0.20980626 0.06909621 0.         0.         0.28016128]
[0.32256311 0.17481757 0.02707202 0.         0.         0.24869034]
[0.30003412 0.14490129 0.         0.         0.         0.2224677 ]
[0.28166567 0.1187762  0.         0.         0.         0.20022093]
[0.26571581 0.09468187 0.         0.         0.         0.18019884]
[0.25197177 0.07238614 0.         0.         0.         0.16217896]
[0.24024352 0.0516786  0.         0.         0.         0.14596106]
[0.23036153 0.03236837 0.         0.         0.         0.13136495]
[0.22217487 0.01428205 0.         0.         0.         0.11822846]
[0.21554934 0.         0.         0.         0.         0.10640561]
[0.21022906 0.         0.         0.         0.         0.09562815]
[0.20544765 0.         0.         0.         0.         0.08511669]
[0.20119182 0.         0.         0.         0.         0.07484431]
[0.1974496  0.         0.         0.         0.         0.06478472]
[0.19421037 0.         0.         0.         0.         0.05491224]
[0.19146476 0.         0.         0.         0.         0.04520172]
[0.18920467 0.         0.         0.         0.         0.03562848]
[0.18742325 0.         0.         0.         0.         0.02616825]
[0.18611483 0.         0.         0.         0.         0.01679709]
[0.18527498 0.         0.         0.         0.         0.00749135]
[0.18490041 0.         0.         0.         0.         0.        ]
```

اگر بدانیم همه داده ها از β بزرگتر هستند راهی که ابتدا به ذهن می رسد این است که در activation function ، مقدار threshold را بزرگتر مساوی β در نظر بگیریم. فقط نکته ای که باید توجه کنیم این است که مقدار ϵ را باید بسیار کوچک بگیریم . زیرا ممکن است با سرعت به مقدار β همگرا شویم و عدد بزرگ شناسایی نشود . الگوریتم به صورت زیر پیاده سازی شده است :

```python
E = 0.01
while cond:
    a_last = np.copy(a)
    for i in range(len(a)):
        sumj = 0
        for j in range(len(a)):
            if i != j:
                sumj = sumj + a_last[j]
        if a_last[i] - E * sumj <= 0.6:
            a[i] = 0.6
        else:
            a[i] = a_last[i] - E * sumj
    print(a)
    n = np.count_nonzero(a == 0.6, axis = 0)
    if(6 - n <= 1):
        cond = False
```

خروجی به صورت زیر است :

```
[1.149  1.048  0.947  0.846  0.8965 1.0985]
[1.10064  0.99863  0.89662  0.79461  0.845615 1.049635]
[1.0547889  0.9517588  0.8487287  0.7456986  0.79721365 1.00327385]
[1.01132216 0.90726176 0.80320136 0.69914096 0.75117116 0.95929196]
[0.97012149 0.86502049 0.75991948 0.65481848 0.70736898 0.91757099]
[0.93107451 0.82492249 0.71877048 0.61261846 0.66569447 0.8779985 ]
[0.89407446 0.78686093 0.67964739 0.6        0.62604063 0.8404677 ]
[0.8587443  0.75045863 0.64217296 0.6        0.6        0.80460146]
[0.82477197 0.71540344 0.60603491 0.6        0.6        0.7700877 ]
[0.79185671 0.68139449 0.6        0.6        0.6        0.7366256 ]
[0.75967651 0.64810967 0.6        0.6        0.6        0.70389309]
[0.72815648 0.61547397 0.6        0.6        0.6        0.67181523]
[0.69728359 0.6        0.6        0.6        0.6        0.64037892]
[0.6668798  0.6        0.6        0.6        0.6        0.60940609]
[0.63678574 0.6        0.6        0.6        0.6        0.6       ]
```

اما اگر ϵ را بزرگتر مثلا 0.05 بگیریم به صورت زیر خروجی می دهد :

```
[0.945  0.84   0.735  0.63   0.6825 0.8925]
[0.756    0.64575  0.6      0.6      0.6      0.700875]
[0.6 0.6 0.6 0.6 0.6 0.6]
```

همان طور که می بینیم نمی توان بزرگترین نود را شناسایی نمود .

راه دیگر این است که مقدار β را از تمام داده های کم کنیم سپس الگوریتم را اجرا نماییم . در این صورت زمان همگرایی کمتر می شود .

```
[0.235  0.12   0.005  0.     0.     0.1775]
[0.189625 0.057375 0.       0.       0.       0.1235  ]
[0.16249375 0.01040625 0.         0.         0.         0.08645   ]
[0.14796531 0.         0.         0.         0.         0.060515  ]
[0.13888806 0.         0.         0.         0.         0.0383202 ]
[0.13314003 0.         0.         0.         0.         0.01748699]
[0.13051698 0.         0.         0.         0.         0.        ]
```

حال اگر بخواهیم اعداد را از بزرگ به کوچک مرتب کنیم لازم است تا الگوریتم را چندبار اجرا کنیم . الگوریتم را به این صورت تغییر می دهیم که در ابتدا بزرگترین عدد را پیدا می کنیم سپس وقتی بزرگترین عدد را پیدا کردیم در یک ارایه قرار می دهیم سپس نود بزرگترین مقدار را صفر می کنیم و مجدد الگوریتم را اجرا می کنیم چون بزرگترین نود الان مقدار صفر دارد با اجرای مجدد دومین بزرگترین عدد را پیدا می کنیم . به همین صورت ادامه می دهیم تا تمام نودها صفر شوند .

```
def MaxNet(x):
    cond = True
    a = np.copy(x)
    E = 0.05
    while cond:
        a_last = np.copy(a)
        for i in range(len(a)):
            sumj = 0
            for j in range(len(a)):
                if i != j:
                    sumj = sumj + a_last[j]
            if a_last[i] - E * sumj <= 0:
                a[i] = 0
            else:
                a[i] = a_last[i] - E * sumj
        print(a)
        n = np.count_nonzero(a == 0, axis = 0)
        if(len(x) - n <= 1):

            cond = False
            maxind = np.where(a != 0)
            if(len(maxind[0])>0):
                sortL.append(maxind[0][0])
                x[maxind[0][0]] = 0
                MaxNet(x)
```

خروجی به صورت زیر است :

```
input :  [1.2  1.1  1.   0.9  0.95 1.15]
output :  [0.18490041 0.         0.         0.         0.         0.        ]

input :  [0.   1.1  1.   0.9  0.95 1.15]
output :  [0.         0.         0.         0.         0.         0.20484974]

input :  [0.   1.1  1.   0.9  0.95 0.  ]
output :  [0.         0.29274552 0.         0.         0.         0.        ]

input :  [0.   0.   1.   0.9  0.95 0.  ]
output :  [0.         0.         0.21687606 0.         0.         0.        ]

input :  [0.   0.   0.   0.9  0.95 0.  ]
output :  [0.         0.         0.         0.         0.29068366 0.        ]

input :  [0.  0.  0.  0.9 0.  0. ]
output :  [0.  0.  0.  0.9 0.  0. ]
```

آرایه اعداد از بزرگ به کوچک به صورت زیر است :

```
1.2
1.15
1.1
1.0
0.95
0.9
```

نکته ای که وجود دارد این است که شرط لازم برای اجرای درست الگوریتم بالا این است که دو مقدار بزرگ با هم به دست نیاید که در اینصورت باعث می شود همه اعداد با هم 0 شده و بزرگترین مقدار مشخص نشود . برای جلوگیری از این مشکل لازم است تا مقدار ϵ را کوچک بگیریم . اگر ϵ کوچکتر مساوی کوچکترین اختلاف بین دو نود باشد درست جواب می دهد .

حال برای مرتب کردن اعداد از کوچک به بزرگ لازم است تا هر بار که مقدار یک نود برابر صفر می شود آن را در یک لیست قرار دهیم .

```
while cond:
    a_last = np.copy(a)
    for i in range(len(a)):
        sumj = 0
        for j in range(len(a)):
            if i != j:
                sumj = sumj + a_last[j]
        if a_last[i] - E * sumj <= 0:
            a[i] = 0
            if (i not in sortL):
                sortL.append(i)
        else:
            a[i] = a_last[i] - E * sumj
    print(a)
    n = np.count_nonzero(a == 0, axis = 0)
    if(6 - n <= 1):
        sortL.append(np.where(a != 0)[0][0])
        cond = False
```

نکته ای که وجود دارد و در قسمت های قبل هم مشابه آن ذکر شد این است که مشکلی که ممکن است ایجاد شود این است که دو نود با هم صفر شوند در این صورت الگوریتم به درست یعمل نمی کند و مقدار ϵ باید کوچک انتخاب شود تا الگوریتم به درستی عمل کند . در این قسمت 0.05 در نظر گرفته شده است .

خروجی به صورت زیر است :

```
[0.9, 0.95, 1.0, 1.1, 1.15, 1.2]
```

سوال ۳ – Mexican Hat

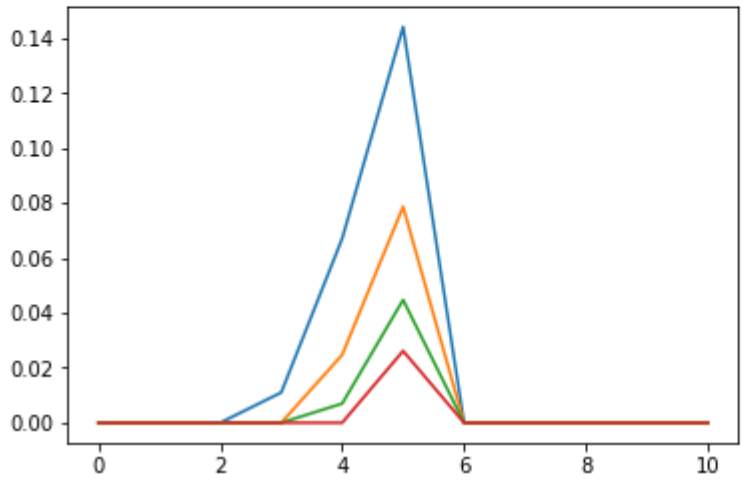
MexicanHat به این صورت عمل می کند به جای پیدا کردن بیشینه مقدار نودی را پیدا می کند که خودش و همسایه هایش اعداد بزرگتری را شامل می شوند . برای هر نود تا شعاع R1 را شعاع هم جواری نودهای همکار می نامیم یعنی در هر به روز رسانی هر نود ، مقدار خود نود و نودهای همسایه با یک تحریک مثبت باعث افزایش مقدار آن نود می شوند . از R1 تا R2 نود هایی هستند که با نود مورد نظر رقابت می کنند و تحریک منفی دارند و از R2 به بعد نوهای خنثی هستند .

حال اگر R1 =0 و R2 = ∞ باشد . در این صورت نود همکار و نود خنثی نداریم و همه نودها با هم رقابت می کنند یعنی مانند MaxNet عمل می شود .

الگوریتم مورد نظر پیاده سازی شده است . پارامتر ها به صورت t_max = 4 و C1 = 0.6 وC2 = -0.1

خروجی به صورت زیر است:

```
[0.    0.    0.    0.011 0.067 0.144 0.    0.    0.    0.    0.   ]
[0.     0.     0.     0.     0.0247 0.0786 0.     0.     0.     0.
 0.    ]
[0.      0.      0.      0.      0.00696 0.04469 0.      0.      0.
 0.      0.     ]
[0.       0.       0.       0.       0.       0.026118 0.       0.
 0.       0.       0.      ]
```

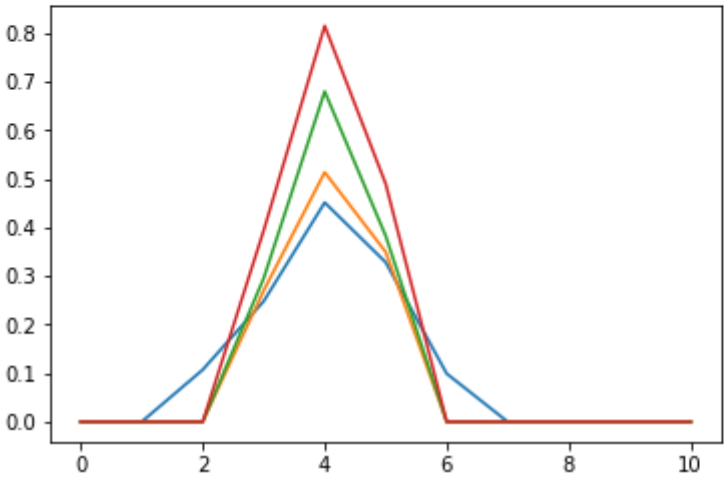


هر بردار نشان دهنده خروجی یک ایپاک است که نمودار آن ها نیز در زیر آن ها رسم شده است . همانطور که مشاهده می کنیم در انتها تنها یک مقدار بزگتر از 0 باقی مانده است که مانند MaxNet عمل شده و بزرگترین عدد شناسایی شده است.

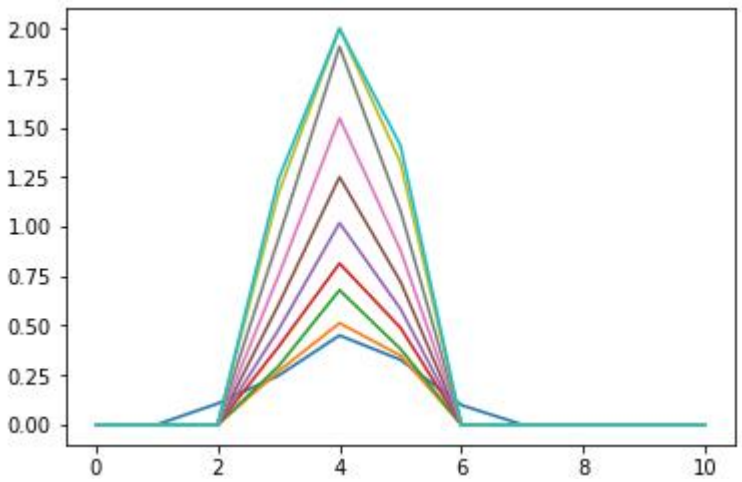
حال اگر R1= 1 و R2 = 3 باشد . یعنی نودهای کناری نودهای همکار و نودهای همسایه دوم و سوم نودها ی رقیب و از نودهای همسایه چهار به بعد نودهای خنثی هستند . پارامترها را به صورت t_max = 4 و C1 = 0.6 و C2 = -0.5 در نظر می گیریم .

خروجی به صورت زیر خواهد بود :

```
[0.    0.    0.107 0.248 0.451 0.328 0.099 0.    0.    0.    0.   ]
[0.     0.     0.     0.2701 0.5132 0.3493 0.     0.     0.     0.
 0.    ]
[0.      0.      0.      0.29533 0.67956 0.38245 0.      0.      0.
 0.      0.     ]
[0.       0.       0.       0.393709 0.814404 0.489541 0.       0.
 0.       0.       0.      ]
```



حال اگر t_max را زیادتر کرده مثلا برابر ۱۰ بگذاریم به صورت زیر خواهد بود ودر نتیجه تاثیری ندارد :



همان طور که می بینیم بر خلاف قسمت قبل به مقدار maximum مطلق نمی رسیم و نودی را پیدا می کنیم که خودش و همسایه هایش بیشترین مقدار را دارند .

در صورتی که دو حالت بالا را مقایسه کنیم می بینیم که نود ۶ ام که مقدار 0.77 را دارد از همه بیشتر است که در حالت اول به درستی انتخاب شده است اما در حالت دوم نود 5 ام که مقدار 0.66 را دارد بالاتر است و همسایه های آن یعنی 0.77 و 0.58 قرار دارند . در این قسمت 0.66 بالاتر است چون خودش و همسایه هایش عدد های بالاتری هستند اما نود 0.77 دارای همسایه های 0.66 و 0.4 است . بنابراین موفق نشده که بالاتر از0.66 قرار گیرد . اما با این حال بالاتر از 0.58 قرار دارد .

سوال ۴ – Hamming Net

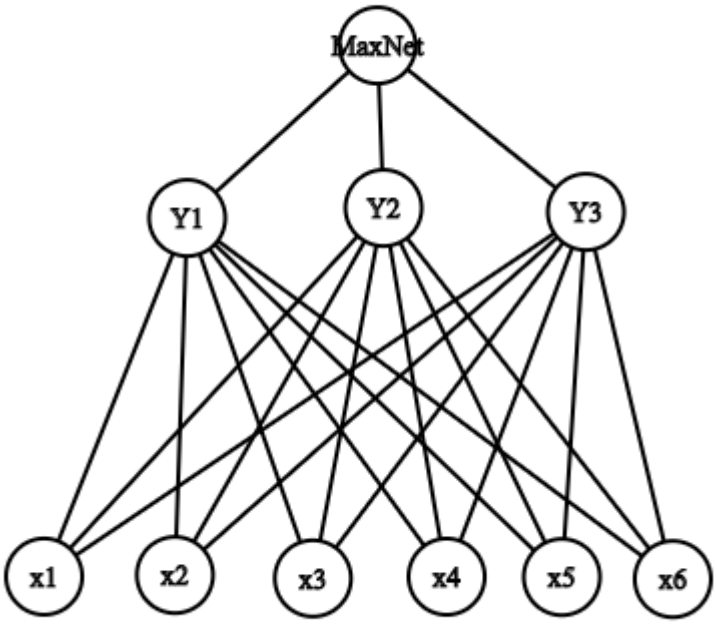
در این شبکه سه بردار مرجع داریم . که در هر بار یک بردار خارجی به عنوان ورودی داده شده و فاصله hamming آن تا سه بردار مرجع محاسبه شده و هر کدام کمتر باشد پیروز می شود . d که همان فاصله hamming است برابر n-a است . n ثابت است پس هر چه a بزرگتر باشد آن برنده است .

$a= x.w + b$

برای انتخاب بزرگترین a از شبکه MaxNet استفاده می کنیم .

الف)

به طور کلی معماری شبکه به صورت زیر خواهد بود :



همانطور که می بینیم طول بردار ورودی برابر ۶ خواهد بود و سه بردار مرجع داریم و خروجی به maxNet وارد شده تا بزرگترین مقدار را تشخیص دهیم .

ب) برای تک تک بردار های ورودی a را برای سه بردار مرجع را محاسبه می نماییم و با استفاده ازMaxNet بیشترین مقدار را انتخاب می کنیم. خروجی MaxNet برای هر بردار به صورت زیر است :

بردار اول :

```
[1.95 1.95 3.1 ]
[1.1925 1.1925 2.515 ]
[0.636375 0.636375 2.15725 ]
[0.21733125 0.21733125 1.9663375 ]
[0.         0.         1.90113813]
```

بنابراین بردار اول متعلق به Y3 است .

بردار دوم :

```
[3.25 0.95 2.1 ]
[2.7925 0.1475 1.47  ]
[2.549875 0.       1.029   ]
[2.395525   0.         0.64651875]
[2.29854719 0.         0.28719   ]
[2.25546869 0.         0.        ]
```

بنابراین بردار اول متعلق به Y1 است .

بردار سوم :

```
[3.25 0.95 2.1 ]
[2.7925 0.1475 1.47  ]
[2.549875 0.       1.029   ]
[2.395525   0.         0.64651875]
[2.29854719 0.         0.28719   ]
[2.25546869 0.         0.        ]
```

بنابراین بردار اول متعلق به Y1 است .

و به همین صورت نتایج نهایی به فرم زیر است و بردار ها به ترتیب زیر به Y1وY2 و Y3 متعلق هستند :

```
['Y3', 'Y1', 'Y1', 'Y2', 'Y3', 'Y1', 'Y2', 'Y3', 'Y3', 'Y3']
```