شبکه های عصبی و یادگیری عمیق

تمرین شماره یک

محمد هاشمی
810197423
Mamadh1993@gmail.com

چکیده – در این گذارش سعی بر این است که سوال‌ها و مثال‌هایی از شبکه‌هایی نظیر *Perceptron Linear* و *AdaLine* و *Madline* و *Mcculloch_Pitts* حل شود و نتیجه این تمارین گذارش شود. برنامه‌ی به کار رفته در حل این سوالات نرم‌افزار*Python* می‌باشد و جهت رسم شکل‌ها از نرم‌افزار *Visio* و برای رسم نمودارها از *Matlab* استفاده شده است.

کلید واژه- AdaLine, Madline, Mcculloch_Pitts, Perceptron Linear,Python,Visio

1- مقدمه

شبکه‌های عصبی مصنوعی یا شبکه‌های عصبی صناعی (Artificial Neural Networks - ANN) یا به زبان ساده‌تر شبکه‌های عصبی سیستم‌ها و روش‌های محاسباتی نوین برای یادگیری ماشینی، نمایش دانش و در انتها اعمال دانش به دست آمده در جهت بیش‌بینی پاسخ‌های خروجی از سامانه‌های پیچیده هستند. ایده اصلی این گونه شبکه‌ها تا حدودی الهام‌گرفته از شیوه کارکرد سیستم عصبی زیستی برای پردازش داده‌ها و اطلاعات به منظور یادگیری و ایجاد دانش قرار دارد. عنصر کلیدی این ایده، ایجاد ساختارهایی جدید برای سامانه پردازش اطلاعات است.

این سیستم از شمار زیادی عناصر پردازشی فوق‌العاده بهم‌پیوسته با نام نورون تشکیل شده که برای حل یک مسئله با هم هماهنگ عمل می‌کنند و توسط سیناپس‌ها (ارتباطات الکترومغناطیسی) اطلاعات را منتقل می‌کنند. در این شبکه‌ها اگر یک سلول آسیب ببیند بقیه سلول‌ها می‌توانند نبود آن را جبران کرده، و نیز در بازسازی آن سهیم باشند. این شبکه‌ها قادر به یادگیری‌اند. مثلاً با اعمال سوزش به سلول‌های عصبی لامسه، سلول‌ها یادمی‌گیرند که به طرف جسم داغ نروند و با این الگوریتم سیستم می‌آموزد که خطای خود را اصلاح کند. یادگیری در این سیستم‌ها به صورت تطبیقی صورت می‌گیرد، یعنی با استفاده از مثال‌ها وزن سیناپس‌ها به گونه‌ای تغییر می‌کند که در صورت دادن ورودی‌های جدید، سیستم پاسخ درستی تولید کند.

2- پیاده‌سازی شبکه Perceptron و استفاده از آن برای کلاس‌بندی کردن

در این تمرین هدف این است که یک شبکه Perceptron با دو ورودی طراحی کنیم و از یک فایل Excel، تعدادی ورودی بخوانیم و با تغییر وزن‌ها، شبکه را آموزش دهیم:

2-1- آموزش شبکه Perceptron برای جداسازی داده‌های دو گلبرگ گل

پس از پیاده‌سازی شبکه در برنامه پایتون، به ترتیب 100 ورودی موجود در دیتاست را به شبکه اعمال می‌کنیم و وزن‌ها را در جهت هدف (Target) تغییر می‌دهیم. در اینجا جروجی مربوط به گلبرگ Iris-setosa را -1 و خروجی گلبرگ Iris-versicolor را 1 در نظر می‌گیریم. پس از اجرای کد مشاهده

می‌کنیم پس از 6 گردش یا 600 ورودی، شبکه ما آموزش داده شده است. وزن های خروجی و مقدار بایاس در شکل 2.1 آماده است.

```
[-3.4  9.1]
-2
```

شکل 2.1 خروجی وزن‌ها و بایاس پس از آموزش شبکه

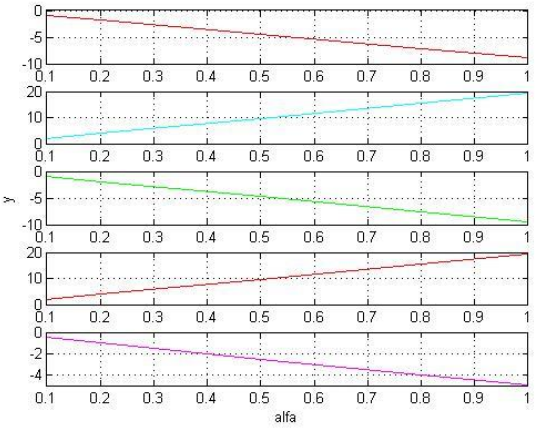
2-2- تاثیر حذف بایاس و تغییر ضریب آلفا (Learning-Rate) بر خروجی وزن‌ها

در ابتدا کار مقدار بایاس را از شبکه حذف می‌کنیم. مشاهده می‌کنیم نسبت به حالت قبل وزن‌ها مقداری افزایش داشته اند. تعداد دفعات تکرار برای رسیدن به هدف دقیقا مانند سابق است، اما وزن‌ها برای جبران مقدار بایاس تغییر می‌کنند که در شکل 2.2 می‌توان مقادیر آن‌ها را مشاهده کرد.

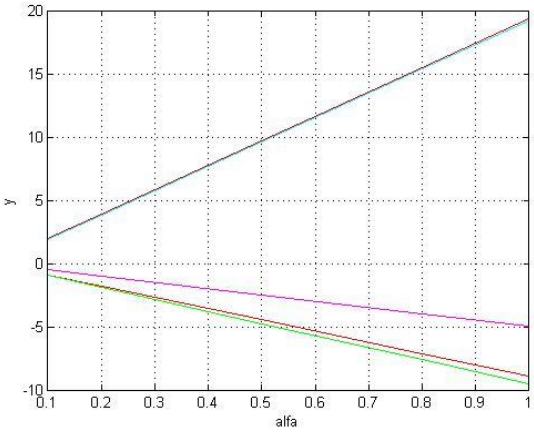
```
[-4.75  9.7 ]
```

شکل 2.2 مقادیر وزن‌ها بدون وجود بایاس

با تغییر مقدار آلفا مشاهده می‌کنیم که تمامی وزن‌ها و بایاس به یک مقیاس افزایش می‌یابد. دلیلی این امر این است که شیب خطی که قرار است کلاس‌ها را از یکدیگر جدا کند، باید ثابت باشد. با توجه به نمودارهای موجود در شکل 2.3 و 2.4 مشاهده می‌کنیم تغییرات شیب هر دو وزن یکدیگر را خنثی می‌کند. در 2.3 اول از بالا به پایین وزن اول و دوم با وجود بایاس در آلفاهای مختلف است و سومی و چهارمی بدون وجود بایاس است و نمودار آخر هم به تغییرات بایاس اشاره دارد.



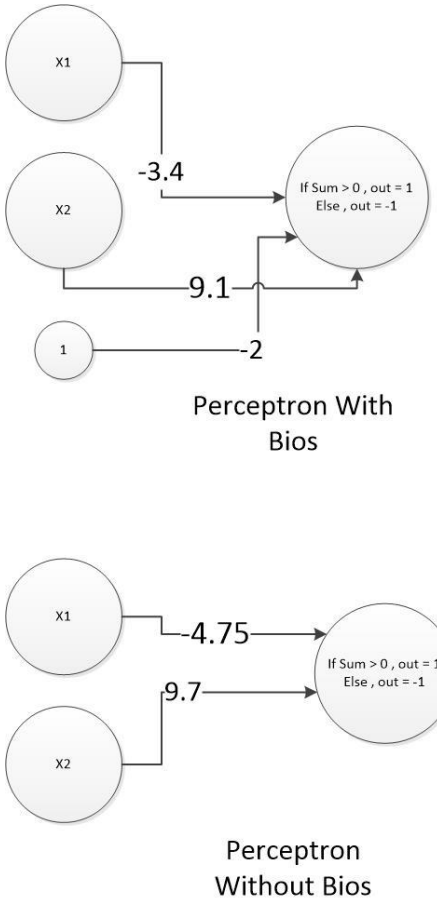
شکل 2.3 نمودار تغییرات وزن‌ها و بایاس نسبت به آلفا



شکل 2.4 نمودار تغییرات وزن‌ها و بایاس در کنار هم

از این تغییرات این نتیجه را می‌گیریم که تغییر آلفا (Learning-Rate) تاثیری در شیب خط خروجی ندارد و تنها پرش‌های بزرگ‌تری را ایجاد می‌کند. از طرفی حذف بایاس باعص می‌شود که وزن‌ها خود را به گونه‌ای متفاوت‌تر سازگار کنند.

شکل نهایی شبکه بعد از آموزش در شکل 2.5 آمده است.

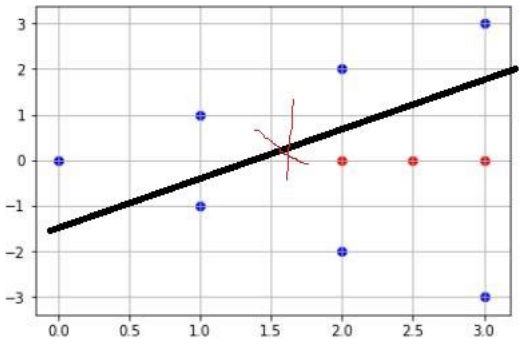


شکل 2.5 شکل نهایی شبکه با وجود بایاس و بدون بایاس با آلفا برابر با 1

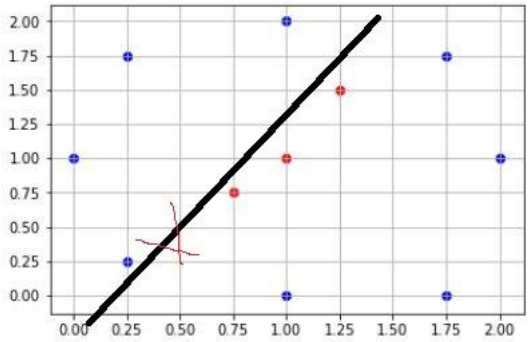
تمامی کدهای متلب در قسمت کد وجود دارد.

3-2- پیاده‌سازی شبکه Perceptron جهت تشخیش دو حالت خاص از کلاس‌ها

در این قسمت از تمرین به طراحی شبکه Perceptron می‌پردازیم که قادر است دو دسته داده در شکل‌های 2.6 و 2.7 را از هم جدا کند. تفاوت این حالت با حالت اول این است که دیگر نمی‌توانیم با یک خط بین این دو کلاس تفاوت قایل شویم.



شکل 2.6



شکل 2.7

بر طبق گفته کتاب برای حل این سوال فضا را به یک فضای 49 بعدی تبدیل میکنیم به گونه ای که هرجا کلاس ما قرار داشت مقدار برابر با یک و باقی نقاط مقداری برابر با منفی یک در نظر می‌گیریم. با توجه به داشتن دو کلاس در کل 2 ورودی داریم که در ابتدای کد پایتون مقدار دهی شده است. نحوه مپ کردن داده‌ها به کلاس ما به صورت زیر خواهد بود:

$$x = 3 - int(i/7)$$

$$y = 0.5 * (i\%7)$$

پس از اجرای آموزش مشاهده میکنیم بعد از سه ایپاک شبکه ما آموزش می‌بیند و مقادیر وزن‌ها و بایاس برای دسته اول داده‌ها (شکل 2.6) به صورت شکل 2.8 در می‌آید.

```
[0 0]
[0 0]
w1 [ 0.38598079  0.03326562  0.92576386  0.77627292  0.05164216  0.38191358
  1.50833398  0.19408428  0.51715565  0.01200611  0.2058635   1.80419132
  0.50832617  0.70807355  0.75367212  0.84340943  1.07539927  0.36202759
  0.60163192  0.60918358  0.87577788  1.98405959  0.72401744  0.68970031
  0.7119943  -0.69912086 -0.46826795 -0.42629368  0.51054925  0.23374013
  1.37463865  0.37947088  0.32667183  0.99354757  0.51303351  0.69857459
  0.69238037  0.46550546  0.359203    1.04323834  0.14481668  0.50351373
  0.52946043  0.51994262  0.85184161  0.08595388  0.74863815  0.75262346
  1.03687256]
w2 [ 0.67358922  0.99127112  0.71444354  0.98731891  0.22291393  0.72110213
 -1.38514646  0.04981161  0.71071461  0.62849283  0.99160217 -1.04476962
  0.39411701  0.68697671  0.73254782  0.12940921 -1.38707213  0.85507765
  0.65049325  0.21967662  0.66671841 -1.06653251  0.11106278  0.51220867
  0.41812032  2.18251131  2.89524978  2.02456634  0.58610397  0.58780023
 -1.80946078  0.34937528  0.32665874  0.09013166  0.03831037  0.50242365
  0.9454067   0.91073293  0.68615681 -1.3769647   0.39550318  0.35589081
  0.39973882  0.5329978   0.8697486   0.39467617  0.58418061  0.29703985
 -1.10969066]
b1 0.18117138501421182
b2 0.7537732486570072
epoch 3
```

شکل2.8: وزن‌ها و بایاس برای دسته اول

حال به دو کلاس دیگر می‌رویم. فضای داده ما در این قسمت شامل 81 نورون می‌باشد که نحوه مپ شدن داده‌ها به فضای ما به صورت زیر است:

$$x = 0.25* int(i/9)$$

$$y = 0.25 * (i\%9)$$

حال به سراغ یادگیری می‌رویم. مشاهده می‌کنیم پس از 3 ایپاک شبکه ما آموزش دیده شده است.نتایج در شکل 2.9 آمده است.

```
[0 0]
[0 0]
w1 [ 0.09807275  0.47708841  0.00846208  0.99381365  1.01381897  0.40458932
  0.90146771  0.44062939  0.62638039  0.05470749  1.10251632  0.91101532
  0.01130639  0.87395802  0.24360567  0.90787998  0.4760586   0.45414555
  0.37567617  0.11883941  0.10212538  0.3193811   0.30426809 -0.10747059
  0.36717148  0.99678241  0.8027393   0.45287631  0.81446552  0.77915167
  0.85116995  0.17560447  0.97259845  0.41156484  0.75732591  0.81952307
  1.56579968  0.60155295  0.33701423  0.17896512 -0.21670287  0.95043556
  0.34782233  0.06055807  1.83063525  0.66973814  0.92179013  0.88912923
 -0.58600691  0.97804477  0.54163967  0.13435261  0.96410378  0.79085425
  0.41645947  0.31446536  0.08560767  0.05277967  0.72191922  0.13073868
  0.13869701  0.98070627  0.51493305  0.79931715  1.74636952  0.38827813
  0.12268775  0.27500015  0.22838928  0.68258359  0.87153908  0.20779218
  0.62751201  0.7211763   0.4295185   0.38871836  1.73905954  0.56114187
  0.06085567  1.00819613  0.26246283]
w2 [ 1.1834252   0.78433801  1.13673706  0.66171027 -2.96831177  1.21642251
  1.00237275  0.70886412  0.66128229  1.14015054 -3.31884787  0.55435931
  1.11928792  0.64466365  1.48420934  0.68325146  0.66679683  0.92267347
  0.9831505   1.45257526  1.26537623  0.83331392  1.41796717  4.03696279
  1.43796363  1.29569643  1.16574679  1.47552831  0.92364596  1.02061988
  0.73723186  0.73035442  0.83237669  0.83788597  0.99056265  1.24480525
 -3.08362477  1.24127301  0.98254088  1.03115256  4.21365214  0.89739553
  1.02980391  0.57610543 -3.38664686  0.74096688  0.78810925  1.2251997
  3.56538571  0.86033756  1.47846493  1.22586333  0.9654172   0.68086039
  1.29123014  1.17416735  0.87752838  1.19452019  0.77954345  1.12565043
  1.18794937  0.64935091  0.62346623  0.97399911 -3.04271442  0.70753413
  0.88680216  1.47983468  0.84585346  0.5270807   0.53373518  1.16554819
  0.9127064   0.78416474  1.067487    1.1251959  -3.39505599  0.84717207
  0.72627447 -2.50519259  0.88622314]
b1 0.37821131974133393
b2 0.46811837593952366
epoch 5
```

شکل 2.9: نتایج برای دسته ورودی‌های دوم

با توجه به کوچک بودن عکس‌ها نتایج در کد پایتون و به صورت عکس‌های جداگانه 1.3.1و1.3.2 در پوشه Pics موجود است.

3- پیاده‌سازی شبکه Adaline جهت تشخیص دو دسته داده با توضیع‌های غیر یکسان

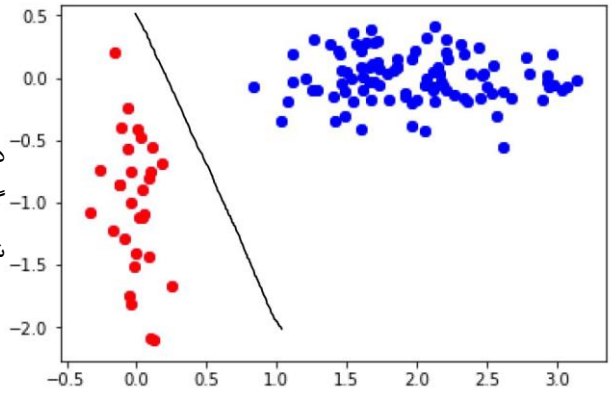
در این بخش از تمرین سعی بر این است که یک شبکه

Adaline را برای دسته‌بندی دو دسته داده موجود در شکل 3.1 با خصوصیات:

دسته اول: شامل 100 داده با میانگین عرض 2 و انحراف معیار 0.5 و میانگین طول 0 و انحراف معیار 0.2

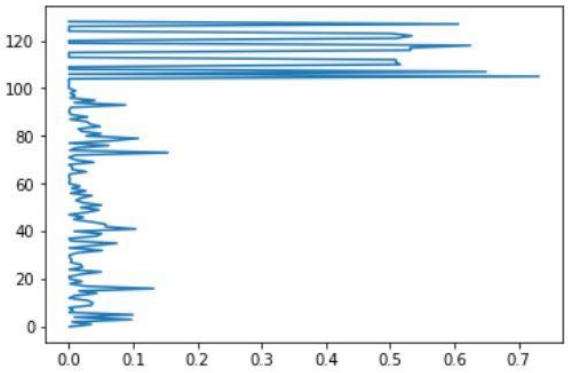
دسته دوم: شامل 30 داده با میانگین عرض 0 و انحراف معیار 0.1 و میانگین طول 1 و انحراف معیار 0.7

را پیاده‌سازی کنیم.



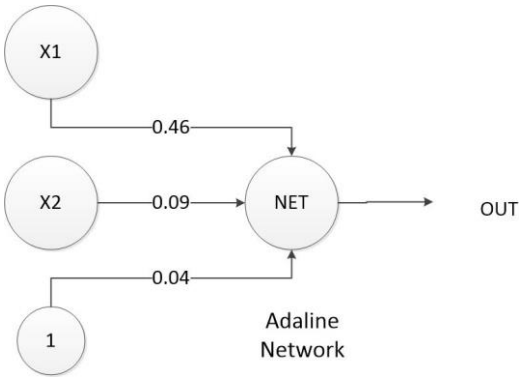
شکل 3.1 نمودار پراکندگی داده‌ها و خط جداساز مد نظر

برای بدست آوردن این دو دسته از کتابخانه random توابع gauss را فراخوانی میکنیم و با فراخوانی توزیع نورمال دو دسته داده را ایجاد کرده و بهم می‌چسبانیم. سپس به دسته داده اول لیبل 1 و به دسته داده دوم لیبل -1 می‌زنیم. پس از نوشتن برنامه و اجرا مشاهده کردیم که پس از 11 ساعت !!!!!!!! اجرای پشت هم برنامه به جوابی نرسید و این به این دلیل بود که شرط صفر شدن برای تمامی داده‌ها فراهم نمی‌شد. ازین رو شرط را به سمت همگرا شدن 0.5*(x[i,2]-summ)(x[i,2]-summ) بردیم و با اجرای 25000 ایپاک مشاهده کردیم که این مقدار به عدد 0.4347464605972482 نزدیک می‌شود. به نظر می‌رسد که از اینجا به بعد شیب نمودار کند تر شده است. برای بررسی این موضوع نمودار مورد نظر در سوال را رسم می‌کنیم. نمودار مورد نظر در شکل 3.2 آمده است.



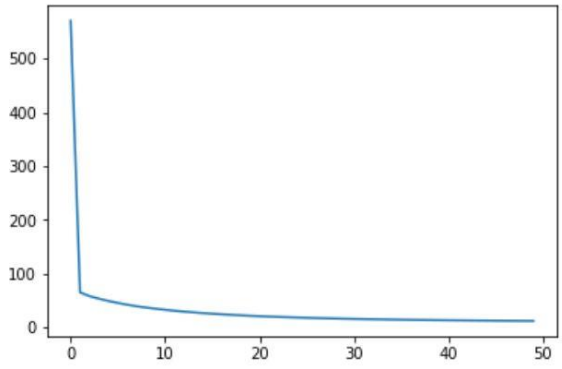
شکل 3.2 نمودار تغییرات خواسته شده (تنها در یک ایپاک)

در نهایت شکل کلی شبکه به صورت شکل 3.3 می‌باشد که در این بر اساس شرط تابع Loss زیر 7.44 آموزش متوفق گردیده است. (عدد 7.44 بر اساس هیچ قانونی در نظر گرفته شده است که در کد قابل تغییر است)



شکل 3.3 شبکه Adaline پس از آموزش با شرط Loss زیر 0.44

حال به سراغ بررسی تابع مورد نظر می‌رویم. در اینجا تابع Loss (0.5 * (error)^2) را برای 50 ایپاک رسم کردیم.



شکل3.4: تابع Loss برای 50 ایپاک

برای بررسی مناسب بودن این نوع شبکه تعداد ایپاک‌ها را به صورت دستی مقایسه کردیم. به صورت کلی مقدار تابع Loss از 586.56 تا ایپاک 8000 به مقداری حدود 7.6 می‌رسد ولی مشاهده میکنیم که از ایپاک 8000 تا 25000 تنها مقدار 0.2 از این مقدار کم می‌شود. ازین‌رو می‌توان دید که روش Adaline

شاید برای رسیدن به جواب دلخواه ما که باز هم 100٪ نیست چه تعداد بالایی ایپاک و زمان مصرف می‌کند پس با قاطعیت می‌توان گفت که این روش، روش هذینه بری است.

4- پیاده سازی شبکه Madaline جهت اجرای منطق XOR , XNOR

در این بخش سعی بر این است که منطق XOR را در سیستم توسط Madaline شبکه بررسی کنیم. در ابتدا کار به بررسی ساختار یادگیری MRI می‌پردازیم. براساس توضیحات موجود درکتاب فاست، این یادگری 7 مرحله کلی دارد که دز ریز مشاهده میکنیم:

مرحله اول : در این مرحله ابتدا تمامی مقادیر اولیه را به صورت تصادفی قرار می‌دهیم که در این سوال مقادیر پیش‌فرض مانند کتاب درنظر گرفته شده است.

Step 0. Initialize weights:
Weights v_1 and v_2 and the bias b_3 are set as described;
small random values are usually used for ADALINE weights.
Set the learning rate α as in the ADALINE training algorithm (a small value).

مرحله دوم تا ششم: مانند شبکه‌های قبلی ورودی‌ها را یکی یکی به مدار وارد کرده و خروجی را با تارگت مورد نظر می‌سنجیم.

Step 3. Set activations of input units:

$$x_i = s_i.$$

Step 4. Compute net input to each hidden ADALINE unit:

$$z\_in_1 = b_1 + x_1w_{11} + x_2w_{21},$$

$$z\_in_2 = b_2 + x_1w_{12} + x_2w_{22}.$$

Step 5. Determine output of each hidden ADALINE unit:

$$z_1 = f(z\_in_1),$$

$$z_2 = f(z\_in_2).$$

Step 6. Determine output of net:

$$y\_in = b_3 + z_1v_1 + z_2v_2;$$

$$y = f(y\_in).$$

مرحله 7 : در صورت عدم برابری دو حالت پیش می‌آید:

حالت اول اینکه مقدار هدف برابر -1 است در این صورت نورونی که ورودیش به صفر نزدیک تر است را به فرمول پایین به‌روز می‌کنیم.

Determine error and update weights:
If $t = y$, no weight updates are performed.
Otherwise:
If $t = 1$, then update weights on Z_J,
the unit whose net input is closest to 0,

$$b_J(\text{new}) = b_J(\text{old}) + \alpha(1 - z\_in_J),$$

$$w_{iJ}(\text{new}) = w_{iJ}(\text{old}) + \alpha(1 - z\_in_J)x_i;$$

در صورتی که مقدار هدف برابر با 1 بود تمامی وزن‌هایی که مقدار مثبت دارند را به‌روز می‌کنیم.

If $t = -1$, then update weights on all units Z_k that have positive net input,

$$b_k(\text{new}) = b_k(\text{old}) + \alpha(-1 - z\_in_k),$$

$$w_{ik}(\text{new}) = w_{ik}(\text{old}) + \alpha(-1 - z\_in_k)x_i.$$

حال از داده‌های کتاب به عنوان Verify استفاده می‌کنیم به این صورت که شبکه XOR را آموزش میدهیم و با داده‌های کتاب مقایسه می‌کنیم در صورت برابری اطمینان حاصل می‌کنیم که شبکه ما درست آموزش داده شده است. پس از پیاده‌سازی شبکه در برنامه پایتون، مقادیر خروجی به صورت زیر درآمد و شکل خروجی شبکه نیز در شکل 4.1 آمده است.

$w11 = -0.73$

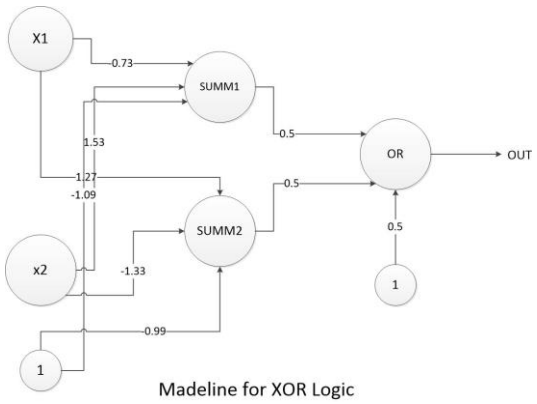
$w12 = 1.27$

$w21 = 1.53$

$w22 = -1.33$

$b1 = -0.99$

$b2 = -1.09$



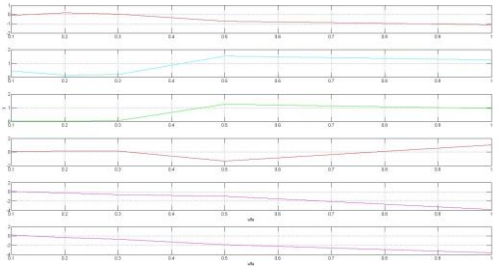
Madeline for XOR Logic

شکل 4.1 شبکه نهایی پس از یادگیری

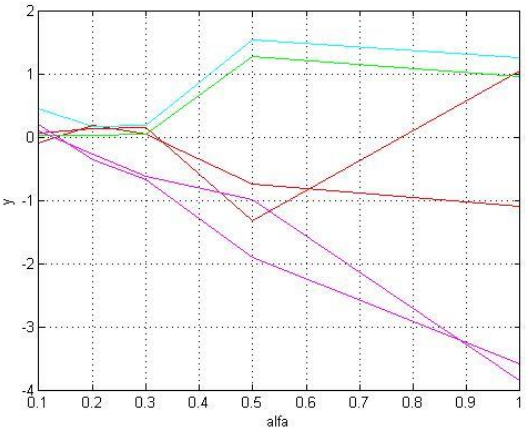
نتیجه می‌گیریم که شبکه درست عمل می‌کند پس از بررسی تاثیر آلفا به سراغ آموزش شبکه با منطق XNOR می-

رویم.

حال به سراغ تاثیر ضریب یادگیری بر وزن‌ها می‌پردازیم. برای این‌کار ضریب را از مقدار 0.1 تا 1 افزایش می‌دهیم. اینگونه که به نظر می‌آید پرش‌های مختلفی با تغییر این ضریب ایجاد می‌شود. در صورت نزدیک شدن به مقدار 0.5 شبکه حالت بهتر با وزن‌های پایدارتر (نسبت به دفعات تکرار و فاصله هر پرش) پیدا می‌کند. با توجه به دو نمودار 4.1 و 4.2 می‌توان دریافت که هرچه ضریب یادگیری افزایش یابد، میزان دو بایاس کاهش می‌یابد و وزن‌ها این کاهش را با افزایش خود جبران می‌کنند. با نزدیک کردن آلفا به میزان 1 به نظر می‌آید که سیستم ناپایدار شده و حتی نتواند خود را آموزش دهد. یک روش بهتر برای آموزش این نوع شبکه‌ها استفاده از شبکه‌های Multi Layer Perceptron (MLP) است زیرا تابع خروجی آنها نرم است.



نمودار 4.1 تغییرات وزن‌ها و بایاس‌ها نسبت به آلفا

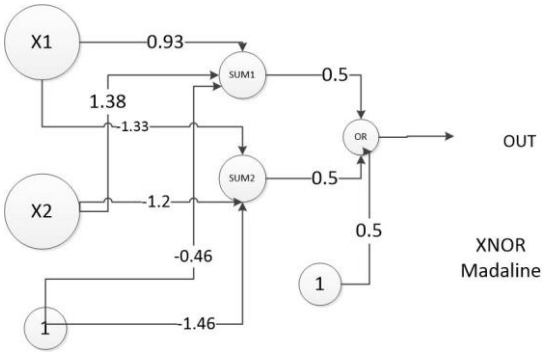


نمودار 4.2 تغییرات کلی وزن‌ها و بایاس‌ها در کنار هم

پس از پیاده سازی گیت XOR و اطمینان از درستی شبکه بر اساس داده‌های کتاب حال به سراغ گیت XNOR می‌رویم. ازین‌رو کافیست ورودی‌های مدار را Not می‌کنیم و شبکه را با ورودی‌های جدید آموزش می‌دهیم. مشاهده می‌کنیم پس از هشت ایپاک آموزش خروجی‌های ما به صورت زیر درآمد.

```
error 0
6
0.9375000000000001
1.3875000000000002
-1.3328125000000002
-1.2046875000000004
-0.46250000000000013
-1.5015625000000004
```

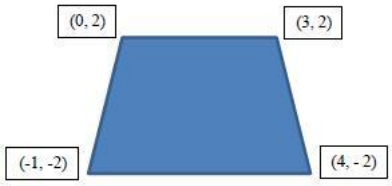
شکل شبکه XNOR به صورت شکل 4.2 می‌باشد.



شکل 4.2: شبکه آموزش دیده با منطق XNOR

5- پیاده‌سازی آلگریتم خاص توسط نورون Mcculloch_Pitts جهت تشخیص نقاط بیرون و درون یک شکل هندسی

در این تمرین سعی بر این است که با استفاده از نورون‌های Mcculloch_Pits و قرار دادن این نورون‌ها در کنار هم در شبکه خاص و طراحی وزن‌ها و تابع Activation بتوانیم تمامی نقاط بیرون و درون شکل 5.1 را تشخیص دهیم.



شکل 5.1

با توجه با شکل 5.1 در میابیم که این شکل ار تقاطع چهار خط با رابطه 5.1 به وجود آمده است.

$$y = 4x + 2$$
$$y = 2$$
$$y = -4x + 14$$
$$y = -2$$

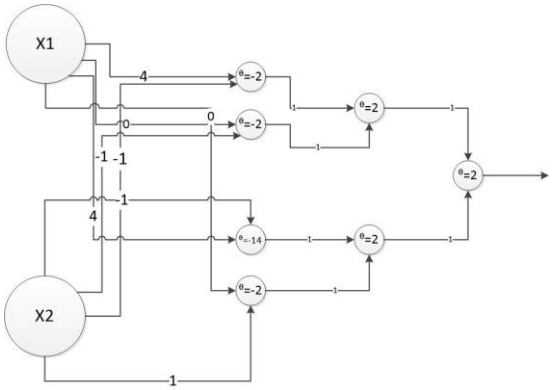
رابطه 5.1

شرایط نقاط داخل این ذوذنقه در مقایسه با این 4 خط باید شرایط نسبت به سه خط اول سمت راست و خط چهارم سمت چپ باشد. برای برقرار کردن این رابطه باید سه رابطه اول را در رابطه 5.1 به یک سمت انتقال دهیم و آخری را دست نخورده نگه داریم. شرایط درون این ذوذنقه در رابطه 5.2 آمده است.

$$-y + 4x > -2$$
$$-y > -2$$
$$-y + 4x > -14$$
$$y > -2$$

رابطه 5.2 شرایط لازم جهت قرار گرفتن نقاط در درون ذوذنقه

برای برقراری رابطه فوق نیاز به 4 نورون مکلاچ پیتر می-باشد که همگی آنها باید با یکدیگر AND شوند. ازین‌رو به سه نورون دیگر نیاز است که عملیات AND را انجام دهند. شکل نهایی شبکه پیشنهادی در شکل 5.2 موجود است.



شکل 5.2 شبکه پیشنهادی برای مساله

پس از پیاده‌سازی این شبکه در پایتون، تمامی مقادیر درون ذوذنقه با اندیس 1 و خارج آن با اندیس 0 شناسایی شده است. کد پایتون این شبکه در پوشه Mccolluch_Pitts موجود است.

مراجع

[1] ."Fundamentals of Neural Networks" by Laurene Fausett, 1994