|  |  |  |
| --- | --- | --- |
|  | به نام خدا |  |
| **دانشگاه تهران**  **دانشکده‌ مهندسی برق و کامپیوتر**  **شبکه های عصبی و یادگیری عمیق**  **تمرین سری اول** | | |

|  |  |
| --- | --- |
| محمدحسین بدیعی | نام و نام خانوادگی |
| 810199106 | شماره‌ دانشجویی |
| 8 آبان 1400 | تاریخ ارسال گزارش |

­

**فهرست گزارش سوالات**

[سوال 1 – Mcculloch pitts 3](#_Toc32834855)

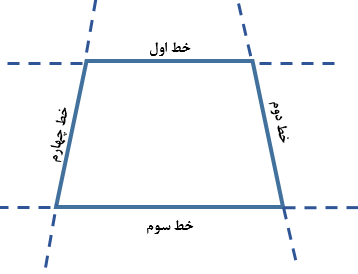
[سوال ۲ – Adaline 7](#_Toc32834856)

[سوال 3 – Perceptron 12](#_Toc32834857)

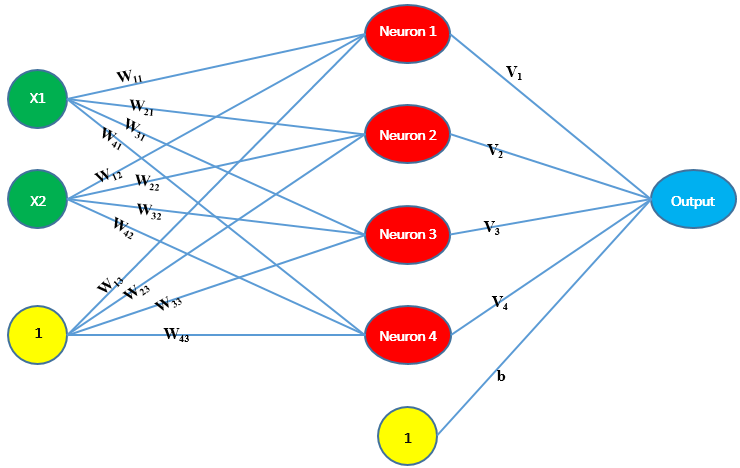
[سوال 4 – Madaline 14](#_Toc32834858)

# سوال 1 – Mcculloch pitts

با توجه به آنکه ناحیه‌ی مربوط به این چهارضلعی با چهار پاره‌خط از محیطِ بیرونی‌اش تفکیک شده است، لذا می‌توان با چهار نورون در لایه‌ی پنهان این چهار خط را ایجاد کرده به گونه‌ای که ناحیه‌ی درونِ این چهار خط چهارضلعیِ خواسته شده در صورت سوال باشد.



هر کدام از این خطوط نشان‌دهنده‌ی مرزِ جداکننده ای برای داده‌ها توسطِ نورون‌های لایه‌ی میانی (لایه‌های پنهان) هستند. نکته‌ای که حائز اهمیت است، فعال شدنِ نورون‌های لایه‌ی پنهان است. لذا معادلات خطوط را به گونه‌ای بدست می‌آوریم که نورونِ متناظر با معادله‌ی هر خط در صورتی فعال شود که ناحیه‌ی چهارضلعی درونِ ناحیهِ فعال شده قرار گیرد. در اینصورت با and کردنِ خروجی‌های نورون‌های میانی می‌توانیم ناحیه‌ی درون چهارضلعی را برای هر زوج داده‌ی ورودی بررسی کنیم. لذا داریم:



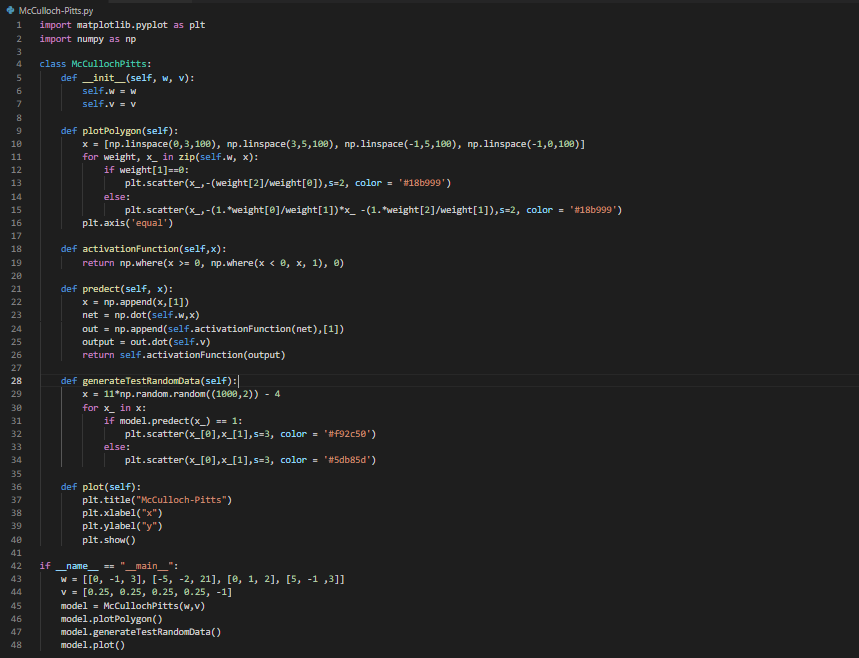
حال تابعِ net های مربوط به هر نورون در لایه‌ی پنهان را به گونه‌ای می‌یابیم که ناحیه‌ی داخل چهارضلعی در ناحیه فعالِ نورون‌های مذکور باشد:

همانطوری که ذکر کردیم در تعیین وزنها به ناحیه‌ی فعال شده توسطِ نورونِ متناظر توجه کردیم و لذا با داشتنِ معادلاتِ net فوق، می توان انتظار داشت که اگر نقطه‌ای درون یا روی ذوزنقه‌ قرار گیرد، هر چهار net فوق دارای مقداری بزرگتر مساوی صفر خواهند داشت و نورون‌های مربوط به همگی آنها فعال خواهد شد و خروجی یک را خواهد داد. در غیر این صورت حداقل یکی از نورون‌ها فعال نمی‌ شود که مبینِ این است که نقطه خارج از ذوزنقه می‌باشد. لذا یک ساختارِ and می بایست بین نورون‌های لایه‌ی پنهان و نورون‌ِ لایه‌ی خروجی قرار دهیم تا ناحیهِ درون و روی ذوزنقه را با یک شدنِ خروجی نشان دهد.

نهایتا وزن‌های شبکه به صورت زیر در می‌آید.

پس وزن‌ها به صورت فوق در آمدند. می‌دانیم تابع فعالساز نورون‌های مک‌کلاچ‌پیتز توسعه یافته نیز در صورتیکه ورودی بزرگتر مساوی صفر باشد، خروجی را 1 و در غیر اینصورت خروجی را صفر بر می‌گرداند. لذا با معماری بالا می‌توانیم اگر به شبکه ورودی‌ای داخل یا روی ذوزنقه دهیم، خروجیِ 1 تحویل بگیریم و برای نقطه‌ای که خارج از ذوزنقه باشد، شبکه خروجی صفر را برمی‌گرداند.

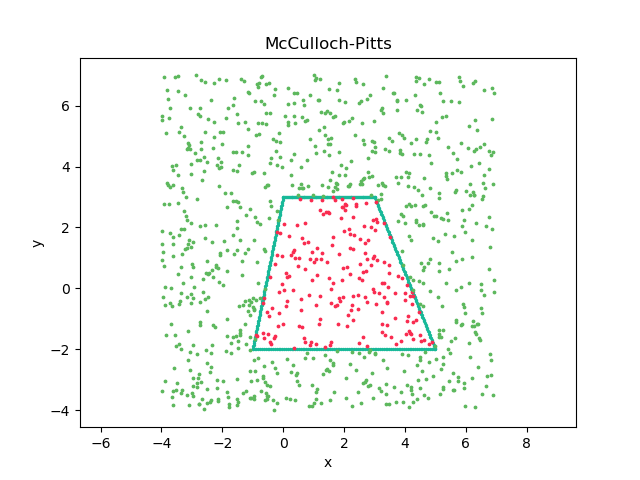
معماریِ این شبکه را با تعریفِ یک کلاس با نام McCullochPitts پیاده کردیم که به صورت زیر است.



همانطور که مشاهده می‌فرمایید یک کلاس با نامِ McCullochPitts ایجاد کردیم که پارامترهای وزن‌ها را به عنوانِ ورودیِ کلاس دریافت می‌کند و یک چند ضلعی رسم می‌نماید. (البته محدوده‌ی چند ضلعی ها را مشخص کردیم که ذوزنقه‌ای مطابق با صورتِ سوال بدست آید.) سپس پیش‌بینی را با داشتنِ وزن‌های میانِ لایه‌ی ورودی و لایه‌ی میانی و همچنین وزن‌های بین لایه‌ی میانی و خروجی مطابق با معماریِ McCulloch-Pitts انجام می‌دهد.

در راستای انجامِ فرایندِ ارزیابیِ عملکرد شبکه، 1000 ورودی را به صورتِ رندوم حولِ چندضلعیِ صورت سوال ایجاد کرده و پس از ساختِ مدل و رسمِ چندضلعی، اقدام به پیشبینیِ خروجیِ شبکه متناظر با هر یک از ورودی‌های مورد نظر می‌کنیم. نهایتا نقاطِ را با توجه به اینکه خارج یا داخلِ پلیگان هستند، رنگ‌دهی می‌کنیم تا نقاطِ داخل پلیگان از نقاط خارج تفکیک شوند.

لذا همانطور که در شکل زیر مشاهده می‌فرمایید تمامی نقاط به درستی پیش‌بینی می‌شوند.



همانطور که مشاهده می‌فرمایید به خوبی نقاطِ رندومِ داخل پلیگان را از ناحیه‌ی خارجی چندضلعی جدا کردیم و هر یک را با رنگ‌های متفاوتی نمایش دادیم. لذا معماری طراحی شده برای شبکه، نتیجه‌ی درستی را به ما بر می‌گرداند. ( روش دوم: لازم به ذکر است که می‌توانستیم خروجی شبکه را به ازای ناحیه‌ی بیرونی یک و به ازای ناحیه‌ی داخلی صفر برگردانیم که در اینصورت ضرایبِ w در یک منفی می‌بایست ضرب شود و ضرایبِ v و b به تشکیلِ یک ساختاری or منجر می‌شدند که با روش ما متفاوت و در عین حال روشی صحیح است)

لطفا برای اجرای کد به پوشه‌ی Codes و فایل McCulloch-Pitts.py مراجعه بفرمایید.

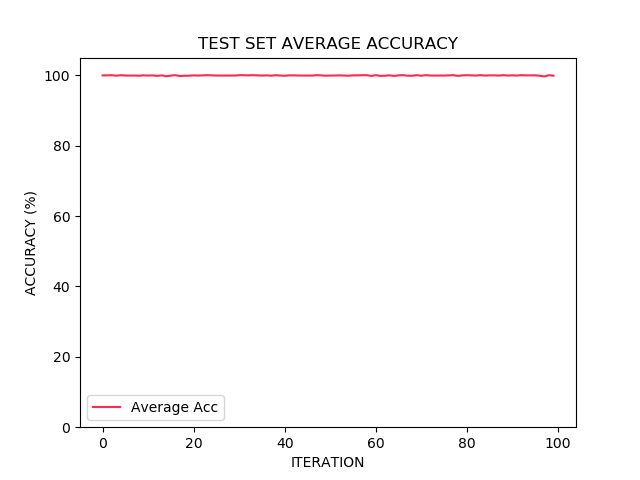
# سوال ۲ – Adaline

در این سوال کلاسی با نام Adaline را پیاده‌سازی کردیم که پس از آموزشِ شبکه توسط داده‌های رندوم با توزیع نرمال به جواب زیر برای وزن‌ها رسیدیم. (توجه داشته باشید که هر ساختار دو تابع sign() و tanh() را در کد پیاده کردیم لاکن نتایج sign() را در ابتدا نشان داده و سپس بهبودِ الگوریتم توسطِ tanh() را ارائه خواهیم کرد.)

وزن‌های بدست آمده به صورت زیر است:

لازم به ذکر است که تعداد حداکثر ایپاک برابر 500 و در عین حال شرط توقف که همگرایی وزن‌ها است در ایپاک 45 رخ می‌دهد و دیگر آموزش ادامه نمی‌یابد.

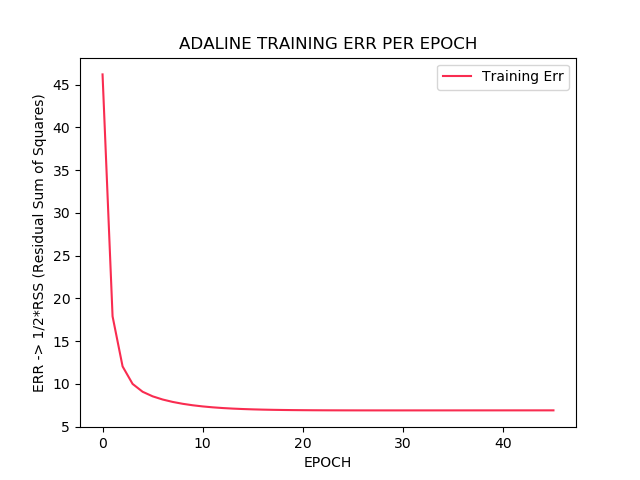
میانگین صحت بدست آمده با وزن‌های فوق برای صد مشاهده و بر روی داده های رندوم تست برابر 679/99 درصد می‌باشد. نمودار میانگین صحت برای مشاهدات به صورت زیر است.



همانطور که مشاهده می‌کنید صحت بدست آمده برای داده‌های تست بسیار مطلوب است. حال به سراغ پاسخ بخش‌های تمرین می‌رویم.

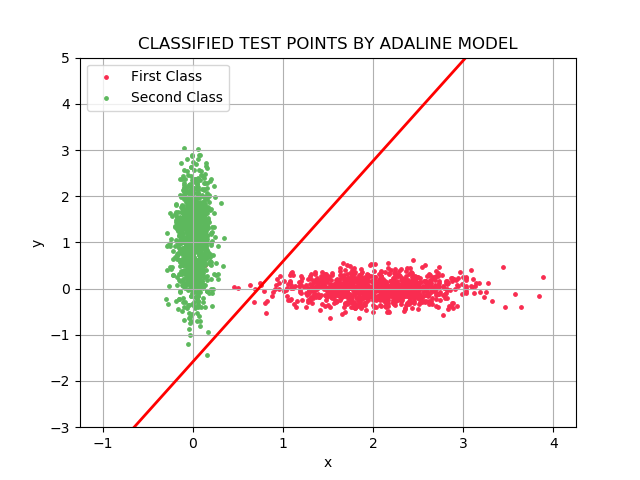
**بخش الف)**

در این بخش خواسته شده خطای مورد نظر در صورت سوال را بدست آوریم. لذا ما خطای (1/2)\*RSS برآمده از هر ایپاک را ترسیم کردیم که به صورت زیر می باشد.



همانطور که ذکر کردیم همگرایی در 45 مین ایپاک رخ داد و در واقع شرط توقفِ آموزش در این ایپاک فعال می‌شود. به همین دلیل تعداد مشاهدات موجود بر محور x، 45 عدد می‌باشد. طبیعتا انتظار داریم که الگوریتمِ ادلاین با فیت کردنِ خطی طبقه بندی را به خوبی انجام دهد و لذا به همین دلیل است که با افزایشِ روندِ آموزش، خطای خواسته شده در صورت سوال کم و کمتر می‌شود.

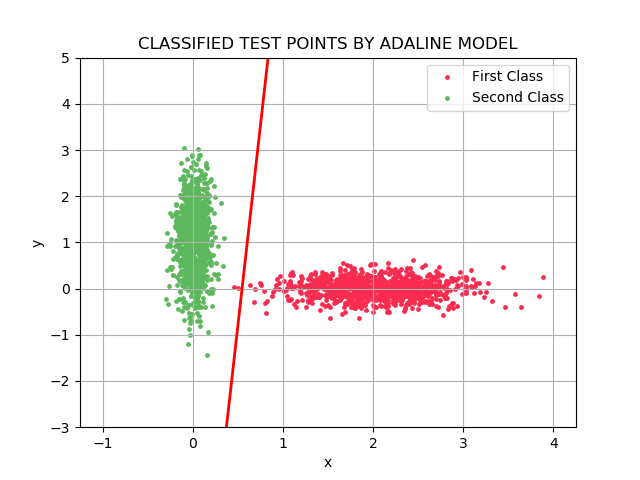
در نهایت آخرین مجموعه‌ی داده‌های تست را که متشکل از 1000 نمونه از هر دسته است، به همراه خط جدا کننده (معادله net) در یک شکل ترسیم نمودیم. توجه داشته باشید که داده‌های ورودی را متناسب با مقدار تارگت (مقدار مورد انتظار) که دارند رنگ بندی کردیم به گونه‌ای که داده‌های همرنگ که در حقیقت در یک دسته هستند، به درستی مشخص باشند که بتوانیم تاثیر الگوریتم را در دسته بندی به خوبی درک کنیم. شکل زیر خروجی بدست آمده از این آزمایش است.



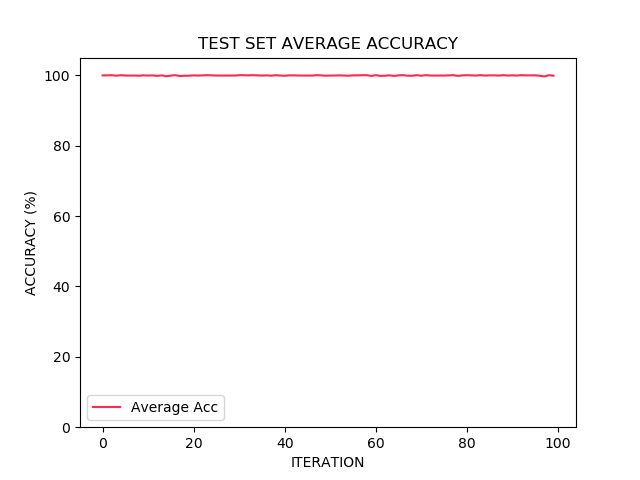
در این آزمایش در بین 1000 داده از هر یک از دسته‌ها، 8 پیش‌بینیِ اشتباه توسطِ الگوریتم داشتیم.

**بخش ب)**

اینکه روشِ ادلاین برای جداسازیِ داده‌ها مناسب باشد یا نباشه مسلما به نوع داده و کاربردِ آن حیطه بستگی دارد. لاکن می توان ادعا کرد که در مقابلِ شبکه‌های عصبیِ پیشرفته‌ی کنونی مانند CNN و RNN و LSTM و ...، این الگوریتم ضعیف عمل می‌کند و در مثال آخر هم مشاهده کردید که هشت پیشبینی اشتباه در 1000 داده از هر دسته بر آمد در صورتیکه در بسیاری از کاربردها همین میزان خطا، خطای زیادی محسوب می شود. مسلما عوامل زیادی در کارایی الگوریتم تاثیرگذار هستند از جمله لرنینگ ریت یا مقدار اولیه‌ی وزن‌ها که می‌بایست مقادیر کوچکی انتخاب شوند. با این حال نکته‌ی کلیدی که به نظرم می رسد عدمِ توازنِ تعداد داده‌های آموزش در هر دسته است. این حقیقت در الگوریتمِ ادلاین با توجه به تابعِ فعالسازِ sign() باعث می‌شود که الگوریتم، تاثیرگذاری دسته‌ای را که دارای تعداد عضوِ آموزشِ کمتری است در آموزشِ شبکه لحاظ نکند. به همین جهت از توابعی استفاده می‌کنند که تاثیرگذاریِ داده‌ها نه با توجه به مجموع تعداشان در دسته، بلکه با لحاظِ فاصله‌ی داده‌ها از خطِ جداکننده تبیین شود. به همین جهت تصمیم گرفتیم برای استدلالِ خود اثباتی ارائه کنیم. شکل زیر نتیجه‌ی اعمالِ ساختارِ tanh() در فعالسازیِ نورون‌ها و آپدیتِ وزن‌ها بر روی همان داده‌های تستِ قبلی می باشد.

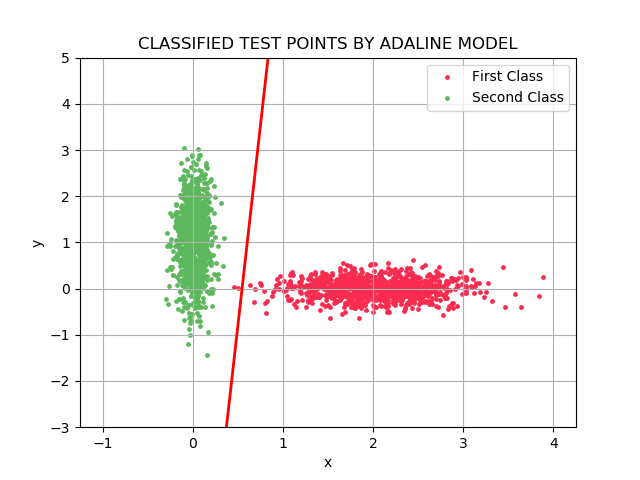
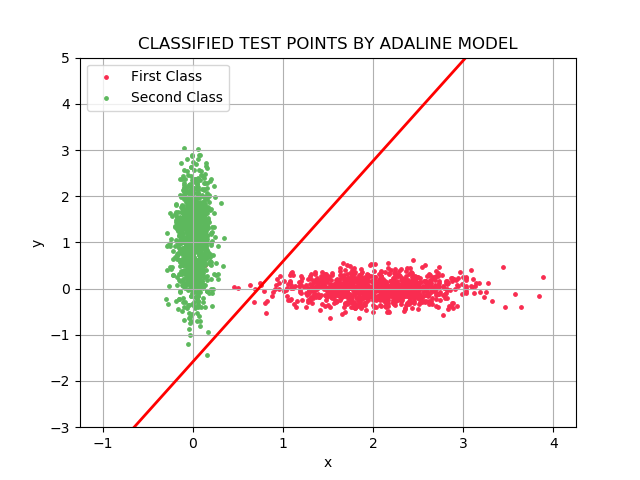


نتیجه‌ی این ساختار این شد که تعداد پیش‌بینی‌های نادرست از 8 عدد به 3 عدد کاهش یابد. لذا بر آن شدیم که بار دیگر شبکه را با این ساختار آموزش دهیم و میانگین صحت را با ساختار قبلی مقایسه کنیم.



همانطور که مشاهده می‌فرمایید میانگین خطا برای 100 مشاهده برابر با % 917/99 شد که از میانگین خطا در حالتِ قبل (ساختارِ sign() به عنوان تابع فعالساز) بیشتر است. لذا نشان دادیم در چنین مواردی که توازن بینِ دسته‌های آموزش یکسان نیست، می‌توانیم به ساختار‌های دیگری مراجعه کنیم.

در نهایت همان دو شکل مربوط به ارزیابیِ هر یک از ساختارها را با توجه به خط جداکننده در کنار یکدیگر نمایش می‌دهیم که صحت ادعای خود را در غالب این شکل نیز مشاهده نماییم.

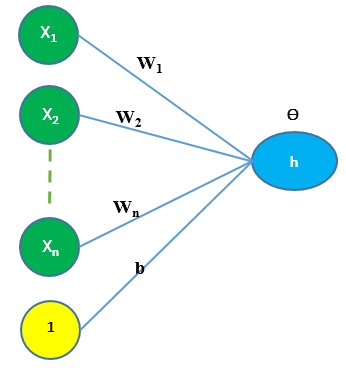


Adaline using tanh() activation function Adaline using sing() activation function

لطفا برای اجرای کد به پوشه‌ی Codes و فایل adaline.py مراجعه بفرمایید.

# سوال 3 – Perceptron

**بخش الف)**

پرسپترون شبکه‌ای با ساختار زیر است که در سال 1958 توسط فرانک روزنبلات معرفی شد.

همانطور که مشاهده می فرمایید ابتدا ورودی‌ها در وزن‌های متنظرشان ضرب شده و به همراه بایاس، از آنها جمع گرفته می‌شود و سپس به تابع فعالساز برای تصمیم‌گیری در رابطه با خروجی نورون داده می‌شود. تابع فعالساز نیز بدین صورت عمل می‌کند که در صورتیکه مقدارِ جمع شده از حد آستانه‌ی انتخابی بیشتر باشد، نورون فعال شده و خروجیِ یک می‌دهد و در صورتیکه از منفیِ آن حدِ آستانه کمتر باشد، نورون غیرفعال و نتیجتا خروجی منفی یک را می‌دهد و در صورتیکه جمعِ مذکور در بینِ این دو عدد (حد آستانه و منفیِ آن) قرار گیرد، نورون قادر به تصمیم‌گیری در بابِ خروجی نخواهد بود و خروجی صفر را تولید می‌نماید. در واقع این محدوده‌ی میانی یک بازه‌ی نایقینی در تصمیم‌گیری برای نورون محسوب می‌شود.

نحوه‌ی کار بدین صورت است که:

گام صفر: مقدار اولیه‌ای برای وزن‌ها، بایاس، نرخ یادگیری و ترشولد در نظر می‌گیریم.

گام اول: به ازای هر (input, target) در مجموعه‌ی آموزش (یعنی {(s,t)}) گام‌های 2، 3 و 4 را انجام می‌دهیم.

گام دوم: مقدار si را به xi نگاشت می‌کنیم.

گام سوم: مقدار خروجی یعنی h را با در نظر داشتنِ معادلاتِ زیر محاسبه می‌کنیم.

گام چهارم: در صورتیکه خطا (h-t) مخالف با صفر بود، ورن‌ها و بایاس طبق معادلات زیر آپدیت می‌شوند.

گام پنج: در صورتیکه ارور به ازای تمامی جفت ورودی‌ها (s,t) برابر با صفر شد، به گام شش می‌رویم در غیر اینصورت به گام اول بر‌میگردیم.

گام شش: توقف الگوریتم و break

**بخش دوم)**

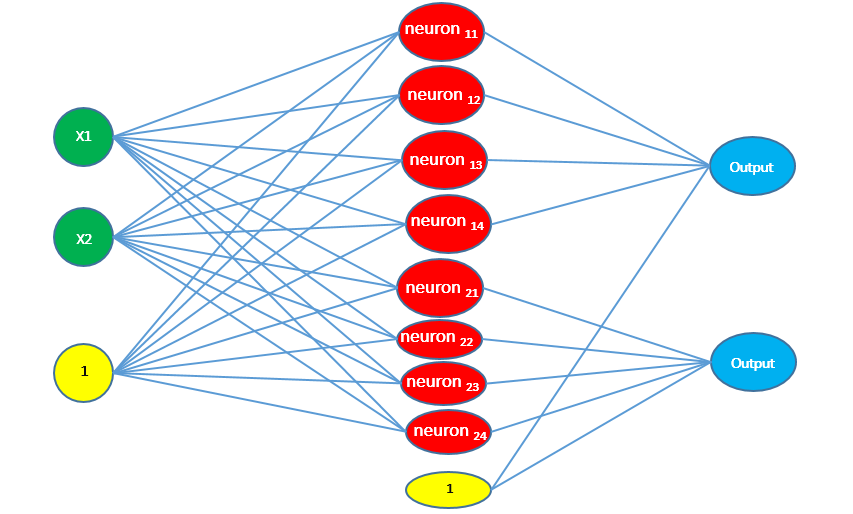
**مرحله اول-**

**مرحله دوم (وزن‌های نهایی پس از پایان مرحله دوم)**

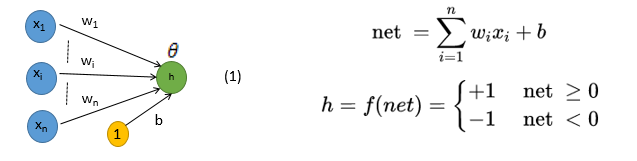
# سوال 4 – Madaline

**بخش الف)**

معماری طراحی شده در این بخش دقیقا مشابه با ایده‌ی مطرح شده در جزوه استاد است. ابتدا معماری شبکه را مشاهده کنیم سپس به توضیحات خواهیم پرداخت.



تمامی نورون‌های موجود در لایه میانی و لایه‌ی خروجی از نوع نورون ادلاین هستند. عملکردِ این نورون به صورت زیر می‌باشد. (عکس زیر از جزوه اقتباس شده است)



در واقع هر یک از این نورون‌ها یک جمع وزن‌دار از ورودی‌ها و بایاس خواهند گرفت و در صورتی که حاصل بیشتر یا مساوی صفر باشد، خروجی +1 را تولید می‌کنند و در غیر اینصورت خروجی منفی یک را می‌دهد.

برای طراحی این معماری بدین گونه عمل کردیم که دو نورون رو به عنوان خروجی نهایی شبکه قرار دهیم به این منظور که هر یک از نورون‌ها در صورتی فعال شود ورودی عضوِ یکی از دسته داده‌ی داخل رینگ باشد. بدین صورت به راحتی می‌توانیم داده‌هایی را که متعلق به یکی از دسته‌های داخل رینگ باشند را با فعال شدنِ نورونِ مربوطه‌اش بیابیم. طبیعتا اگر هیچ‌یک از نورون‌ها فعال نشدند، می‌توانیم استدلال کنیم که چون ورودی متعلق به دسته‌های داخل رینگ نیست، پس عضوی از دسته داده‌های رینگ می‌باشد.

لذا با این دو نورون سه خروجی را انتظار داریم.

[1, 0] : داده‌ی ورودی عضو دسته‌ی اول (چون خروجیِ یک را فعال کرده)

[0, 1] : داده‌ی ورودی عضو دسته‌ی دوم (چون خروجیِ دو را فعال کرده)

[0, 0] : داده‌ی ورودی عضو دسته‌ی سوم یعنی رینگ دایره‌ای (چون هیچ یک از خروجی‌های مربوط به دسته‌های داخل رینگ را فعال نکرده است)

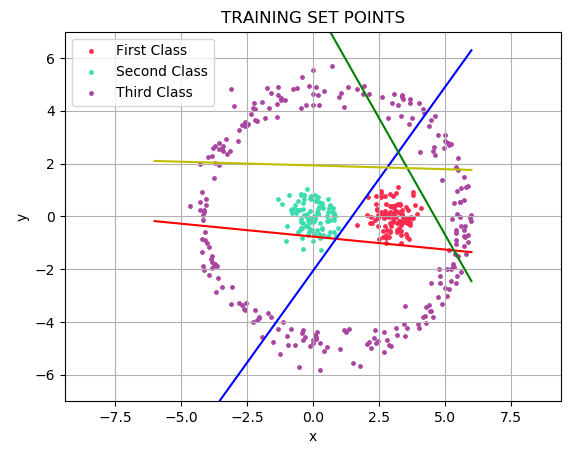
لذا با این تقسیم بندی کافی است که یک سری از نورون‌های لایه‌ی میانی را به خروجی یک متصل می‌کردیم و یک سری را به خروجی دو و هر قسمت را به صورت جداگانه آموزش دهیم.

برای شفافیت موضوع شکل ورودی را ترسیم می‌کنیم.



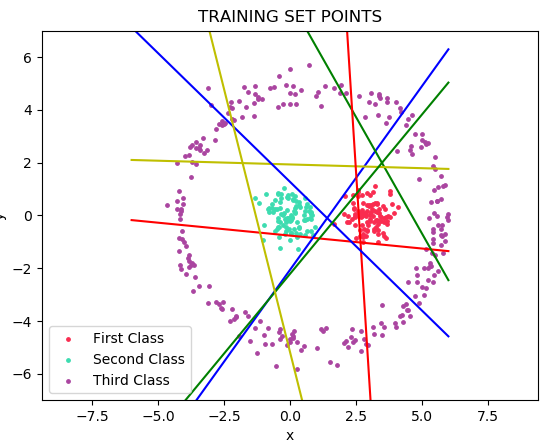
در شکل فوق داده‌های ورودی را ترسیم کردیم که پیاده‌سازی را دقیق‌تر شرح دهیم. مثلا در صورتیکه مجموعه داده‌ی قرمز را دسته اول بگیریم و بخواهیم وزن‌های نورون‌های متصل به خروجی یک را آموزش دهیم، کافی است بقیه‌ی داده ها را شامل دسته‌ی دوم و دسته‌ی سوم (رینگ) در یک مجموعه قرار دهیم و خروجی مورد انتظار -1 برایشان لحاظ کنیم. حال برای عضو‌های دسته‌ی اول خروجی مورد انتظار +1 را در نظر میگیریم و شبکه را برای خروجی اول و وزن‌های متصل به نورون‌های مربوطه به این قسمت از شبکه را آموزش می‌دهیم. سپس همین روند را برای خروجی دوم و وزن‌های مربوطه‌اش پیاده می‌کنیم به گونه‌ای که این خروجی برای داده‌های مربوط به دسته‌ی دوم فعال شود. بدین صورت تمامی وزن‌های شبکه اموزش می‌یابد. لازم به ذکر است که ما در ابتدا برای هر نورون خروجی 3 نورون در لایه‌ی میانی (مجموعا شش نورون در لایه میانی) در نظر گرفتیم که وزن‌ها به سختی آموزش می‌دیدند و لذا تعداد نورونِ متصل به هر خروجی را یکی اضافه کردیم. نهایتا شبکه ای با 8 نورونِ میانی توانست جواب مطلوبی را پس از 609 ایتریشن به ما بدهد و خطای آموزش را صفر کند. طبق الگوریتم وزن‌های بین لایه‌ی میانی و خروجی در ابتدا تعریف می‌شوند و به همین جهت ما یک ساختارِ and را برای هر یک از خروجی‌ها در این قسمت تعریف کردیم تا چهار خطِ آموزش دیده بتوانند یک چهارضلعی را بر روی داده‌های هر یک از دسته‌های یک و دو محاط کنند. بدین صورت نورون‌های میانی باید خروجی‌ای مشابه با نورونِ خروجیِ استخراج کنند تا این چهارضلعی به درستی فیت شود. بدین منظور ساختارِ آپدیتِ وزن‌ها در الگوریتمِ MRI باعث می‌شد که تا زمانیکه خروجی نورون‌های میانیِ متصل به نورون خروجی یک نشوند، خروجی نهایی هم یک نشود و این دقیقا با ساختارِ and ما تطابق داشت. حساسیتِ این الگوریتم به نرخ یادگیری بسیار بالاست و به همین جهت نرخ یادگیری را بسیار پایین و در اردرِ پنج هزارم گرفتیم.

**بخش ب**



نتیجه آموزش جهت تفکیک داده‌های دسته اول نتیجه آموزش جهت تفکیک داده‌های دسته دوم

بدلیل وضوح بیشتر در تفکیک بندی نتیجه ی آموزش قسمت ب را در دو شکل نمایش دادیم. همانطور که مشاهده می‌فرمایید هر یک از دسته‌ها به خوبی توسط نورنِ خروجیِ مربوط به خود تفکیک می‌شوند و در صورتیکه هیچ یک از نورون‌های خروجی فعال نشود نتیجه میگیریم که داده‌ی ورودی متعلق به رینگ است که توضیحاتِ آن داده شد.



چون شکل فوق ابهام داشت اشکال قبلی را ارائه کردیم که به تفکیک مشاهده بفرمایید.

بخش ج)

برای لرنینگ ریت 0.005

زمان : 11.8431775

برای لرنینگ ریت 0.001

زمان: 4.4479976

به نظر من هر چه لرنینگ ریت کم شود سرعت یادگیری کمتر خواهد شد ولی ممکن است تاثیراتی بگذارد که زودتر به جواب برسیم مانند آنچه در مساله ما مشاهده شد.

لذا چیزی که مشاهده کردیم این حقیقت را بیان می کند که ممکن است سرعت یادگیری کم شود ولی همین عامل مانعِ عبورِ پارامترها از مقدار بهینه شود. و لذا اگر همین لرنینگ ریت زیاد بود باعث ایجاد پرش در وزن‌ها هنگام آپدیت نمودن آنها می‌شد. لذا این عامل می‌تواند در زمانیکه نزدیک به نقطه‌ی بهینه است نه تنها کند بلکه سریعتر ما را به جواب برساند.