



به نام خدا



دانشگاه تهران  
دانشکده مهندسی برق و کامپیوتر  
شبکه های عصبی و یادگیری عمیق

تمرین سری اول

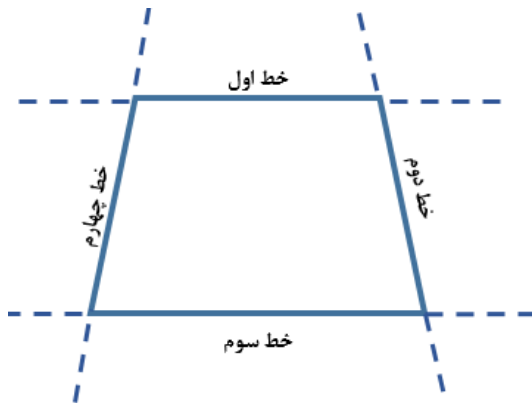
نام و نام خانوادگی	محمدحسین بدیعی
شماره دانشجویی	810199106
تاریخ ارسال گزارش	8 آبان 1400

## فهرست گزارش سوالات

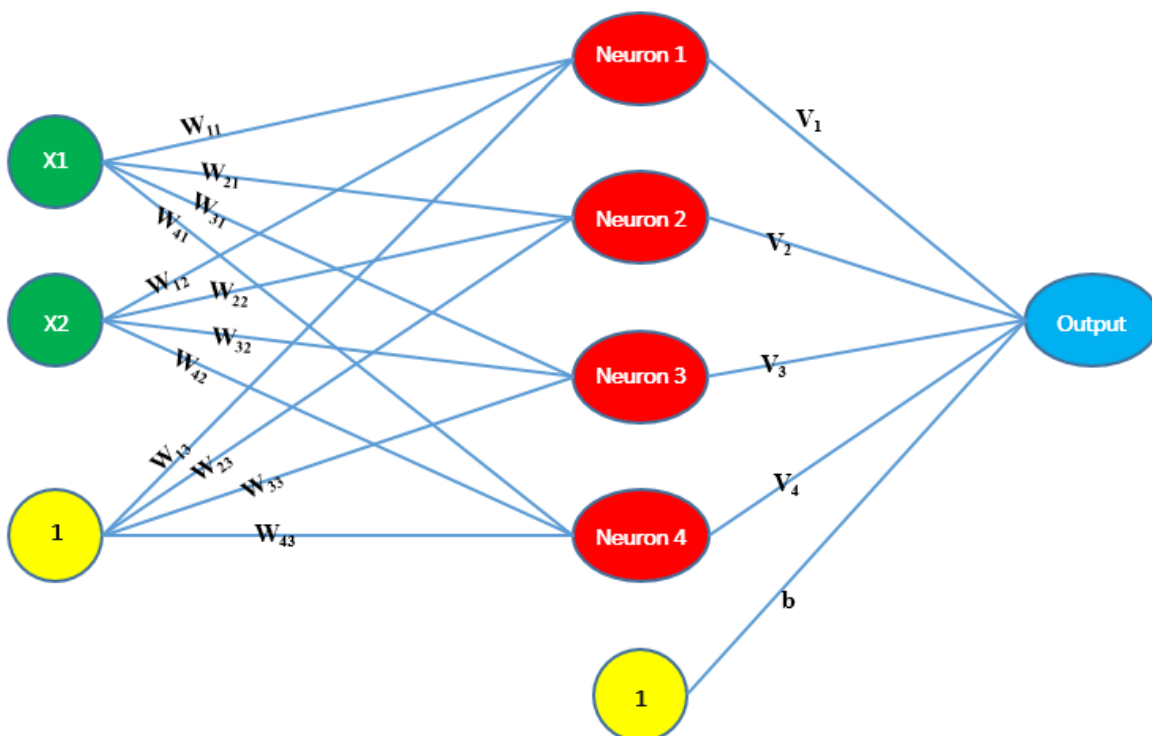
- 3..... سوال 1 - Mcculloch pitts
- 7..... سوال ۲ - Adaline
- 12..... سوال 3 - Perceptron
- 14..... سوال 4 - Madaline

## سوال 1 – Mcculloch pitts

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



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



حال تابع net های مربوط به هر نورون در لایه پنهان را به گونه ای می یابیم که ناحیه داخلی چهارضلعی در ناحیه فعال نورون های مذکور باشد:

$$\begin{cases} \text{first equation} \Rightarrow w_{11}x_1 + w_{12}x_2 + w_{13} = 0_1 \Rightarrow 0 = -x_2 + 3 \\ \text{second equation} \Rightarrow w_{21}x_1 + w_{22}x_2 + w_{23} = 0_2 \Rightarrow 0 = -5x_1 - 2x_2 + 21 \\ \text{third equation} \Rightarrow w_{31}x_1 + w_{32}x_2 + w_{33} = 0 \Rightarrow 0 = x_2 + 2 \\ \text{forth equation} \Rightarrow w_{41}x_1 + w_{42}x_2 + w_{43} = 0 \Rightarrow 0 = 5x_1 - x_2 + 3 \end{cases}$$

$$\Rightarrow \begin{cases} \text{net}_1 = -x_2 + 3 \\ \text{net}_2 = -5x_1 - 2x_2 + 21 \\ \text{net}_3 = x_2 + 2 \\ \text{net}_4 = 5x_1 - x_2 + 3 \end{cases}$$

همانطوری که ذکر کردیم در تعیین وزنها به ناحیه ای فعال شده توسط نورون متناظر توجه کردیم و لذا با داشتن معادلات net فوق، می توان انتظار داشت که اگر نقطه ای درون یا روی دوزنقه قرار گیرد، هر چهار net فوق دارای مقداری بزرگتر مساوی صفر خواهند داشت و نورون های مربوط به همگی آنها فعال خواهد شد و خروجی یک را خواهد داد. در غیر این صورت حداقل یکی از نورون ها فعال نمی شود که مبین این است که نقطه خارج از دوزنقه می باشد. لذا یک ساختار and می بایست بین نورون های لایه پنهان و نورون لایه خروجی قرار دهیم تا ناحیه درون و روی دوزنقه را با یک شدن خروجی نشان دهد.

$$\Rightarrow \begin{cases} v_1 = 0.25 \\ v_2 = 0.25 \\ v_3 = 0.25 \\ v_4 = 0.25 \\ b = -1 \end{cases}$$

نهایتا وزن های شبکه به صورت زیر در می آید.

$$\begin{bmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \\ w_{31} & w_{32} & w_{33} \\ w_{41} & w_{42} & w_{43} \end{bmatrix} = \begin{bmatrix} 0 & -1 & 3 \\ -5 & -2 & 21 \\ 0 & 1 & 2 \\ 5 & -1 & 3 \end{bmatrix}, \quad \begin{bmatrix} v_1 \\ v_2 \\ v_3 \\ v_4 \\ b \end{bmatrix} = \begin{bmatrix} 0.25 \\ 0.25 \\ 0.25 \\ 0.25 \\ -1 \end{bmatrix}$$

پس وزن ها به صورت فوق در آمدند. می دانیم تابع فعال ساز نورون های مک کلاچ پیترز توسعه یافته نیز در صورتیکه ورودی بزرگتر مساوی صفر باشد، خروجی را 1 و در غیر این صورت خروجی را صفر بر می گرداند.

لذا با معماری بالا می‌توانیم اگر به شبکه ورودی‌ای داخل یا روی دوزنقه دهیم، خروجی 1 تحویل بگیریم و برای نقطه‌ای که خارج از دوزنقه باشد، شبکه خروجی صفر را برمی‌گرداند.

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

```

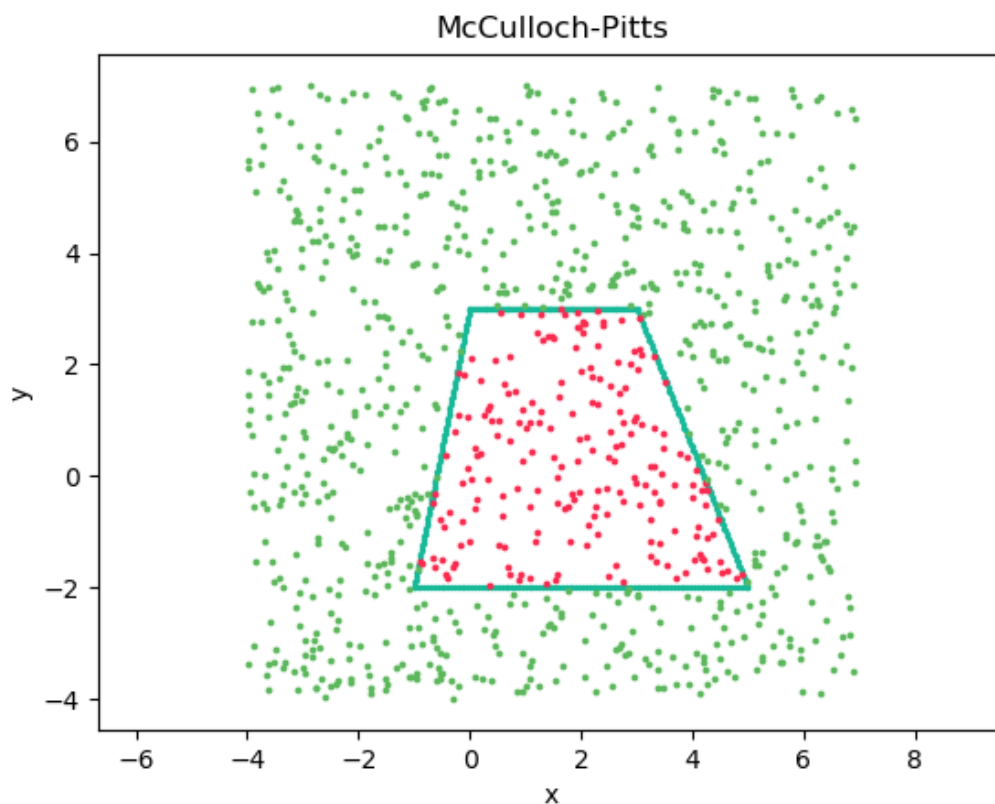
1  import matplotlib.pyplot as plt
2  import numpy as np
3
4  class McCullochPitts:
5      def __init__(self, w, v):
6          self.w = w
7          self.v = v
8
9      def plotPolygon(self):
10         x = [np.linspace(0,3,100), np.linspace(3,5,100), np.linspace(-1,5,100), np.linspace(-1,0,100)]
11         for weight, x_ in zip(self.w, x):
12             if weight[1]==0:
13                 plt.scatter(x_,-(weight[2]/weight[0]),s=2, color = '#18b999')
14             else:
15                 plt.scatter(x_,-(1.*weight[0]/weight[1])*x_ -(1.*weight[2]/weight[1]),s=2, color = '#18b999')
16         plt.axis('equal')
17
18     def activationFunction(self,x):
19         return np.where(x >= 0, np.where(x < 0, x, 1), 0)
20
21     def predelect(self, x):
22         x = np.append(x,[1])
23         net = np.dot(self.w,x)
24         out = np.append(self.activationFunction(net),[1])
25         output = out.dot(self.v)
26         return self.activationFunction(output)
27
28     def generateTestRandomData(self):
29         x = 11*np.random.random((1000,2)) - 4
30         for x_ in x:
31             if model.predelect(x_) == 1:
32                 plt.scatter(x_[0],x_[1],s=3, color = '#f92c50')
33             else:
34                 plt.scatter(x_[0],x_[1],s=3, color = '#5db85d')
35
36     def plot(self):
37         plt.title("McCulloch-Pitts")
38         plt.xlabel("x")
39         plt.ylabel("y")
40         plt.show()
41
42 if __name__ == "__main__":
43     w = [[0, -1, 3], [-5, -2, 21], [0, 1, 2], [5, -1, 3]]
44     v = [0.25, 0.25, 0.25, 0.25, -1]
45     model = McCullochPitts(w,v)
46     model.plotPolygon()
47     model.generateTestRandomData()
48     model.plot()

```

همانطور که مشاهده می‌فرمایید یک کلاس با نام McCullochPitts ایجاد کردیم که پارامترهای وزن‌ها را به عنوان ورودی کلاس دریافت می‌کند و یک چند ضلعی رسم می‌نماید. (البته محدوده‌ی چند ضلعی‌ها را مشخص کردیم که دوزنقه‌ای مطابق با صورت سوال بدست آید). سپس پیش‌بینی را با داشتن وزن‌های میان‌لایه‌ی ورودی و لایه‌ی میانی و همچنین وزن‌های بین لایه‌ی میانی و خروجی مطابق با معماری McCulloch-Pitts انجام می‌دهد.

در راستای انجام فرایند ارزیابی عملکرد شبکه، 1000 ورودی را به صورت رندوم حول چندضلعی صورت سوال ایجاد کرده و پس از ساخت مدل و رسم چندضلعی، اقدام به پیش‌بینی خروجی شبکه متناظر با هر یک از ورودی‌های مورد نظر می‌کنیم. نهایتاً نقاط را با توجه به اینکه خارج یا داخل پلیگان هستند، رنگ‌دهی می‌کنیم تا نقاط 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 داخلِ 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 تشکیل یک ساختاری  $or$  منجر می‌شدند که با روش ما متفاوت و در عین حال روشی صحیح است)

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

## سوال ۲ – Adaline

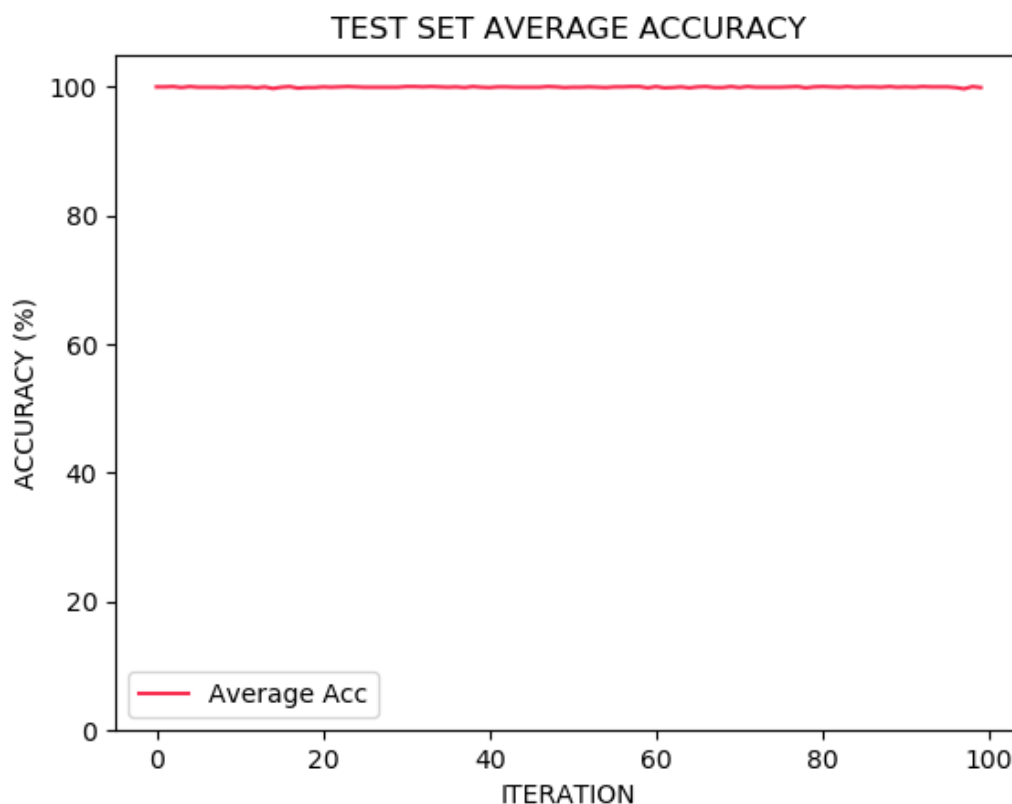
در این سوال کلاسی با نام Adaline را پیاده‌سازی کردیم که پس از آموزش شبکه توسط داده‌های رندوم با توزیع نرمال به جواب زیر برای وزن‌ها رسیدیم. (توجه داشته باشید که هر ساختار دو تابع  $\text{sign}()$  و  $\tanh()$  را در کد پیاده کردیم لکن نتایج  $\text{sign}()$  را در ابتدا نشان داده و سپس بهبود الگوریتم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

وزن‌های بدست آمده به صورت زیر است:

$$[w_1 \ w_2 \ b] = [0.69172907 \ -0.31801947 \ -0.50465315]$$

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

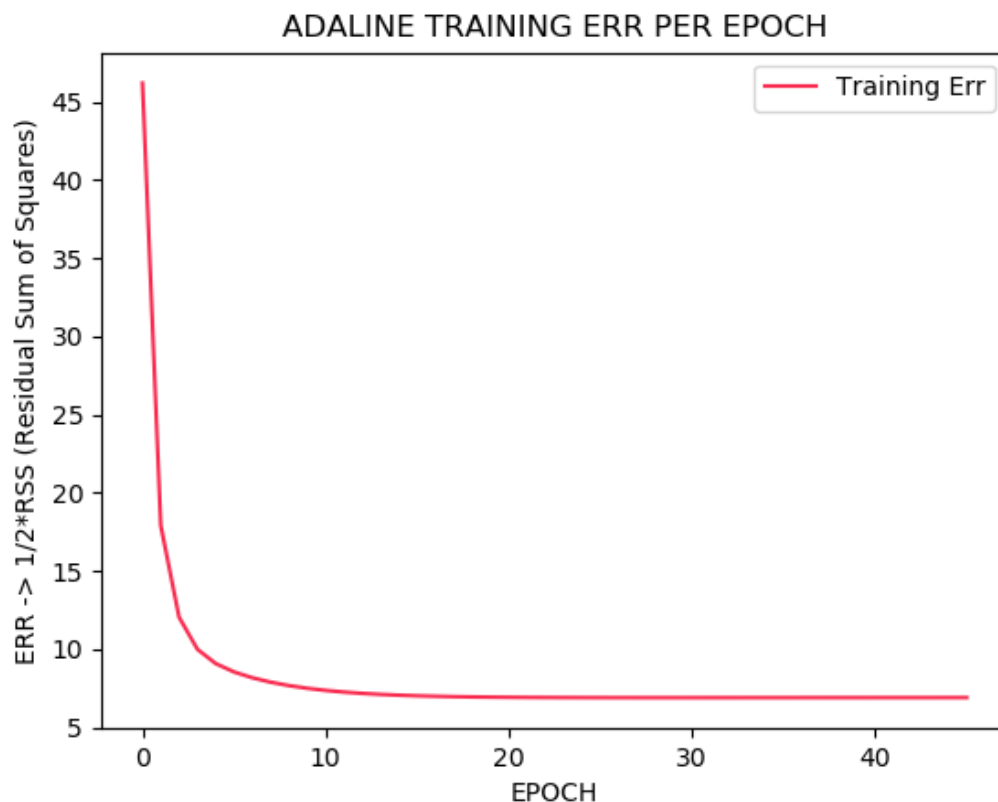
میانگین صحت بدست آمده با وزن‌های فوق برای صد مشاهده و بر روی داده‌های رندوم تست برابر 99/679 درصد می‌باشد. نمودار میانگین صحت برای مشاهدات به صورت زیر است.



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

## بخش الف)

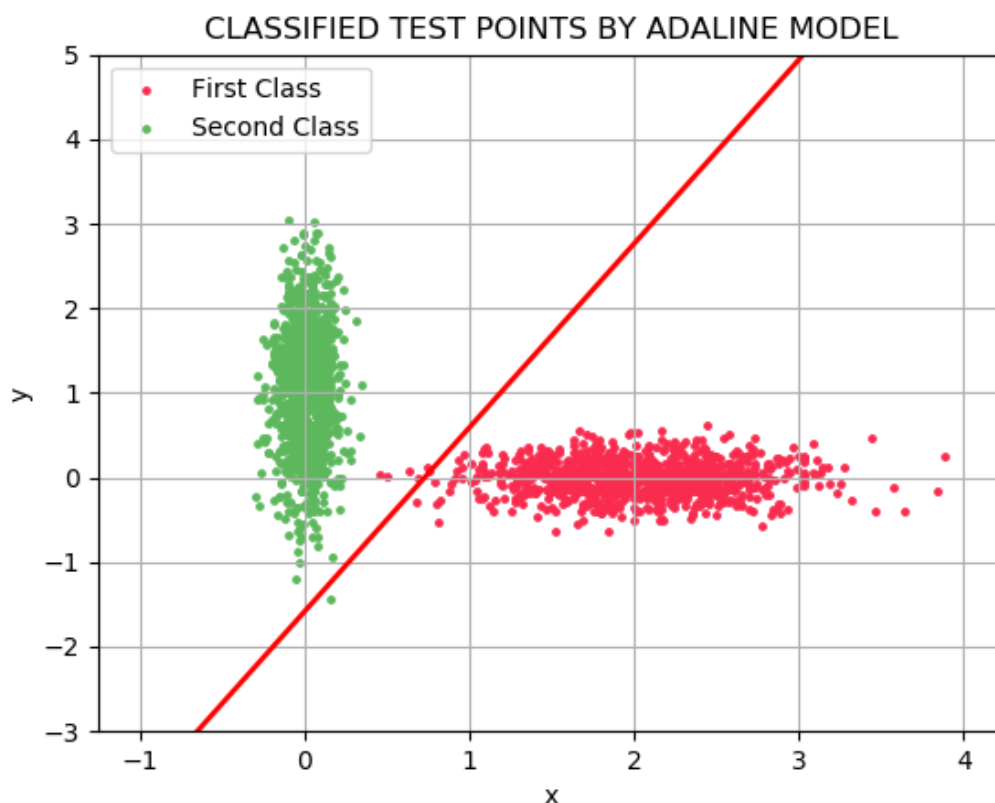
در این بخش خواسته شده خطای مورد نظر در صورت سوال را بدست آوریم. لذا ما خطای  $(1/2)*RSS$  برآمده از هر ایپاک را ترسیم کردیم که به صورت زیر می باشد.



همانطور که ذکر کردیم همگرایی در 45 مین ایپاک رخ داد و در واقع شرط توقف آموزش در این ایپاک فعال می شود. به همین دلیل تعداد مشاهدات موجود بر محور  $x$ ، 45 عدد می باشد. طبیعتاً انتظار داریم که الگوریتم ادالین با فیت کردن خطی طبقه بندی را به خوبی انجام دهد و لذا به همین دلیل است که با افزایش روند آموزش، خطای خواسته شده در صورت سوال کم و کمتر می شود.

در نهایت آخرین مجموعه داده های 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 داده های ورودی را متناسب با مقدار تارگت (مقدار مورد انتظار) که دارند رنگ بندی کردیم به گونه ای که داده های هم رنگ 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.



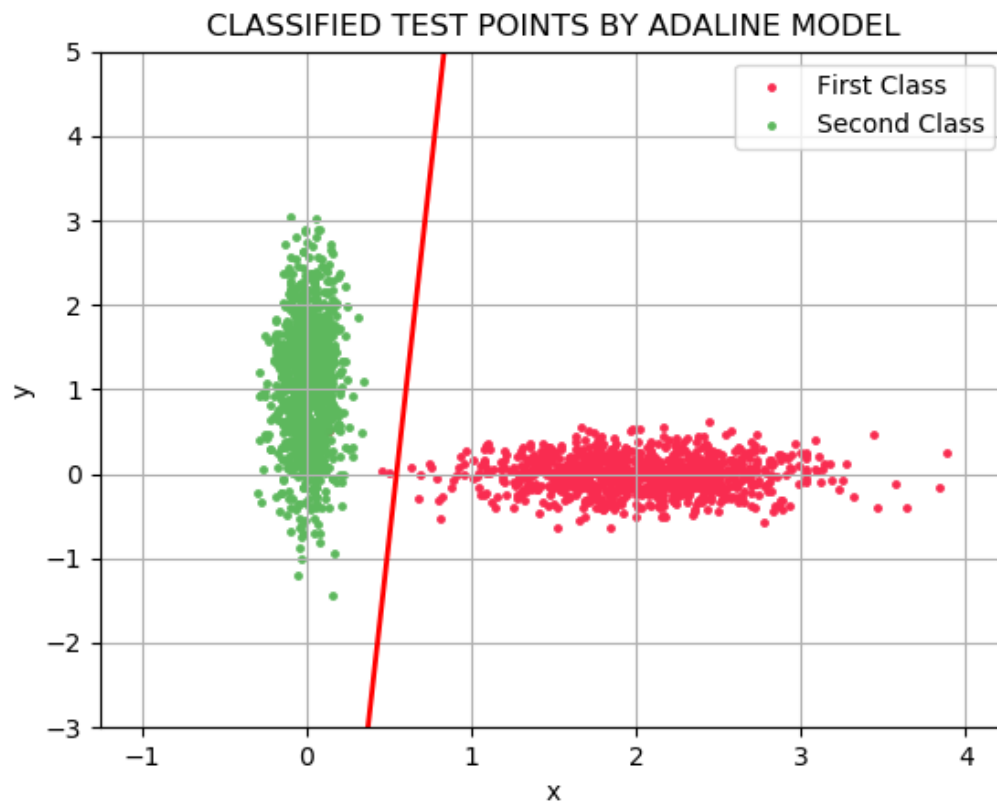


در این آزمایش در بین 1000 داده از هر یک از دسته‌ها، 8 پیش‌بینی اشتباه توسط الگوریتم داشتیم.

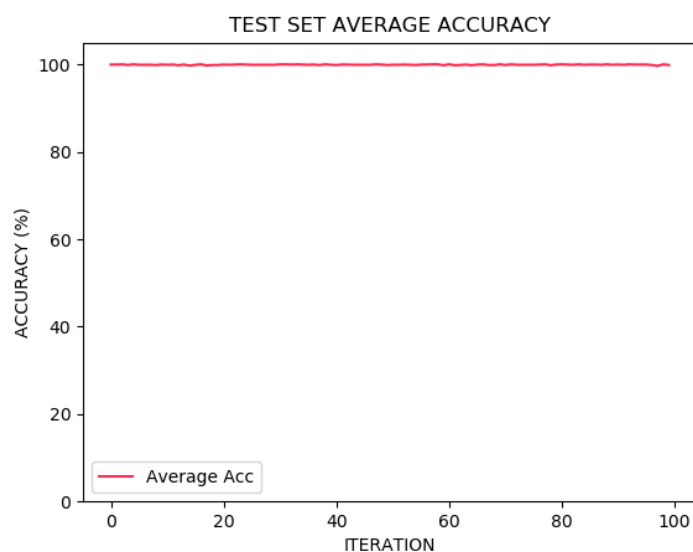
### بخش ب)

اینکه روش ادالین برای جداسازی داده‌ها مناسب باشد یا نباشد مسلماً به نوع داده و کاربرد آن حیطه بستگی دارد. لاکن می‌توان ادعا کرد که در مقابل شبکه‌های عصبی 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 می‌شود. مسلماً 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 نظر می‌رسد عدم توازن تعداد داده‌های آموزش در هر دسته است. این حقیقت در الگوریتم ادالین با توجه به تابع فعالساز  $\text{sign}()$  باعث می‌شود که الگوریتم، تاثیرگذاری دسته‌ای را که دارای تعداد عضو آموزش کمتری است در آموزش شبکه لحاظ نکند. به همین جهت از توابعی استفاده می‌کنند که تاثیرگذاری داده‌ها نه با توجه به مجموع تعدادشان در دسته، بلکه با لحاظ فاصله‌ی داده‌ها از خط جداکننده تبیین شود. به همین

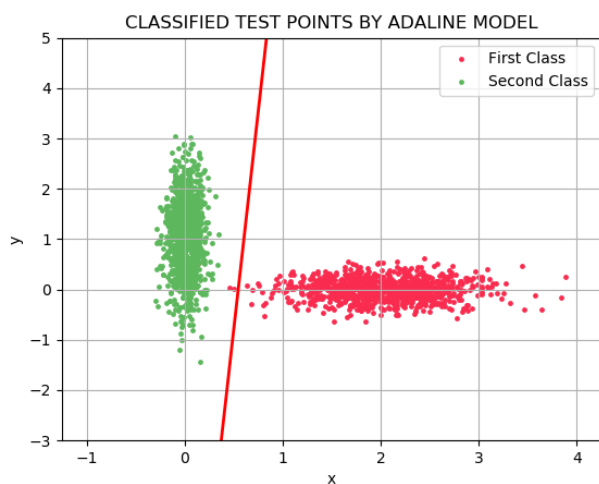
جهت تصمیم گرفتیم برای استدلال خود اثباتی ارائه کنیم. شکل زیر نتیجه‌ی اعمال ساختار  $\tanh()$  در فعالسازی نوروها و آپدیت وزن‌ها بر روی همان داده‌های تست قبلی می باشد.



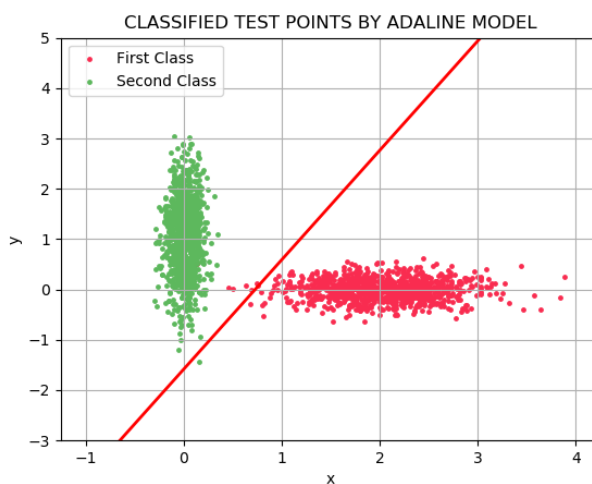
نتیجه‌ی این ساختار این شد که تعداد پیش‌بینی‌های نادرست از 8 عدد به 3 عدد کاهش یابد. لذا بر آن شدیم که بار دیگر شبکه را با این ساختار آموزش دهیم و میانگین صحت را با ساختار قبلی مقایسه کنیم.



همانطور که مشاهده می‌فرمایید میانگین خطا برای 100 مشاهده برابر با  $99/917\%$  شد که از میانگین خطا در حالت قبل (ساختار  $\text{sign}()$  به عنوان تابع فعالساز) بیشتر است. لذا نشان دادیم در چنین مواردی که توازن بین دسته‌های آموزش یکسان نیست، می‌توانیم به ساختارهای دیگری مراجعه کنیم. در نهایت همان دو شکل مربوط به ارزیابی 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



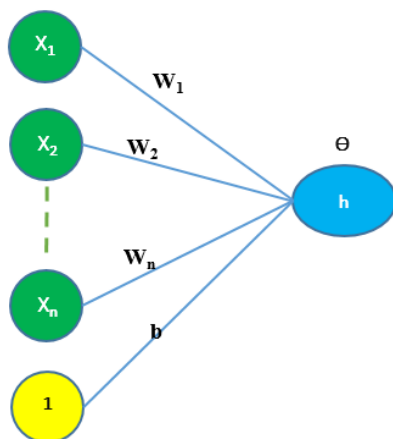
Adaline using  $\text{sing}()$  activation function

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

### سوال 3 – Perceptron

#### بخش الف)

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



$$net = w_1x_1 + w_2x_2 + \dots + w_nx_n + b$$

$$h = \begin{cases} 1 & \text{if } net > \theta \\ 0 & \text{if } |net| < \theta \\ -1 & \text{if } net < -\theta \end{cases} \quad \begin{matrix} Active \\ non - decision \\ Passive \end{matrix}$$

همانطور که مشاهده می‌فرمایید 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 مقدار جمع شده از حد آستانه‌ی انتخابی بیشتر باشد، نورون فعال شده و خروجی یک می‌دهد و در صورتیکه از منفی آن حد آستانه کمتر باشد، نورون غیرفعال و نتیجتاً خروجی منفی یک را می‌دهد و در صورتیکه جمع مذکور در بین این دو عدد (حد آستانه

و منفی آن) قرار گیرد، نورون قادر به تصمیم‌گیری در باب 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  $s_i$  را به  $x_i$  نگاشت می‌کنیم.

گام سوم: مقدار خروجی یعنی  $h$  را با در نظر داشتن معادلات زیر محاسبه می‌کنیم.

$$net = w_1x_1 + w_2x_2 + \dots + w_3x_3 + b$$

$$h = \begin{cases} 1 & \text{if } net > \theta & \text{Active} \\ 0 & \text{if } |net| < \theta & \text{non-decision} \\ -1 & \text{if } net < -\theta & \text{Passive} \end{cases}$$

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

$$w_i(new) = w_i(old) + \alpha * x_i * t \quad \alpha: \text{learning rate}$$

$$b(new) = b(old) + \alpha * t$$

گام پنجم: 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

$$w_1 = 0.1, \quad w_2 = 0.6, \quad w_3 = 0.3, \quad b = 0.5, \quad \alpha = 0.2, \quad \theta = 0$$

$$h = \begin{cases} 1 & \text{if } net > 0 & \text{Active} \\ 0 & \text{if } net = 0 & \text{non-decision} \\ -1 & \text{if } net < 0 & \text{Passive} \end{cases}$$

## مرحله اول-

$$FIRST STEP: net = w_1x_1 + w_2x_2 + w_3x_3 + b$$

$$\Rightarrow net = (0.1) * (0) + (0.6) * (1) + (0.3) * (1) - 0.5 = 0.4$$

$\Rightarrow h = 1$  and  $h$  is non zero so go ahead to update weights

$$\Rightarrow \begin{cases} w_1 = 0.1 + 0.2 * (0) * (-1) \\ w_2 = 0.6 + 0.2 * (1) * (-1) \\ w_3 = 0.3 + 0.2 * (1) * (-1) \\ b = -0.5 + 0.2 * (-1) \end{cases} \Rightarrow \begin{cases} w_1 = 0.1 \\ w_2 = 0.4 \\ w_3 = 0.1 \\ b = -0.7 \end{cases}$$

مرحله دوم (وزن های نهایی پس از پایان مرحله دوم)

FIRST STEP:  $net = w_1x_1 + w_2x_2 + w_3x_3 + b$

$$\Rightarrow net = (0.1) * (0) + (0.4) * (1) + (0.1) * (1) - 0.7 = -0.2$$

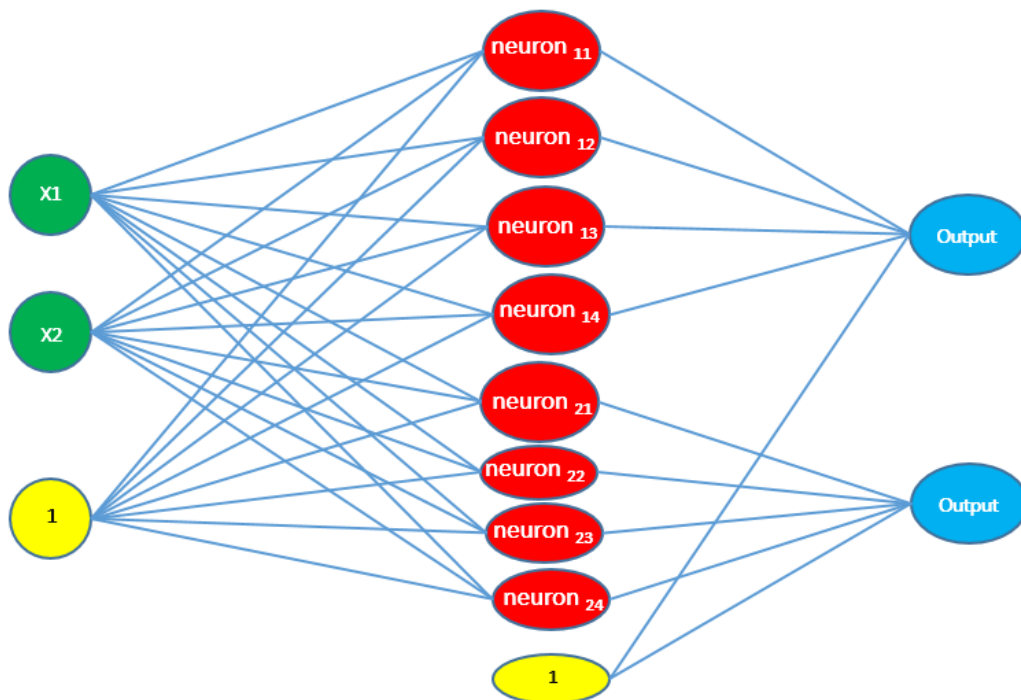
$\Rightarrow h = 1$  and  $h$  is non zero so go ahead to update weights

$$\Rightarrow \begin{cases} w_1 = 0.1 + 0.2 * (0) * (-1) \\ w_2 = 0.4 + 0.2 * (1) * (-1) \\ w_3 = 0.1 + 0.2 * (1) * (-1) \\ b = -0.7 + 0.2 * (-1) \end{cases} \Rightarrow \begin{cases} w_1 = 0.1 \\ w_2 = 0.2 \\ w_3 = -0.1 \\ b = -0.9 \end{cases}$$

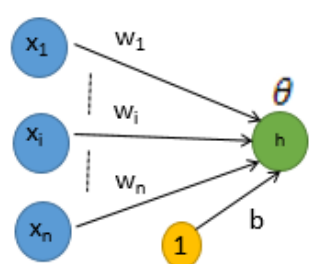
## سوال 4 – Madaline

بخش الف)

معماری طراحی شده در این بخش دقیقاً 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 لایه خروجی از نوع نورون ادلاین هستند. عملکرد این نورون به صورت زیر می‌باشد. (عکس زیر از جزوه اقتباس شده است)



(1)

$$net = \sum_{i=1}^n w_i x_i + b$$

$$h = f(net) = \begin{cases} +1 & net \geq 0 \\ -1 & net < 0 \end{cases}$$

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.

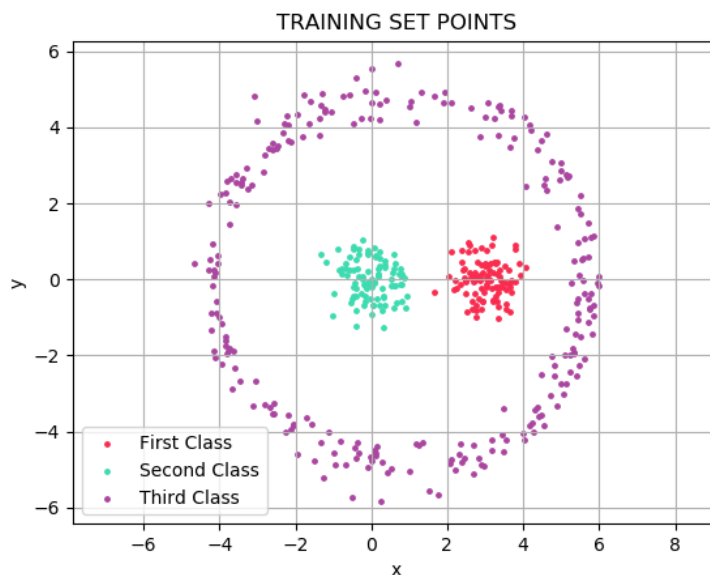
برای طراحی این معماری بدین گونه عمل کردیم که دو نورون رو به عنوان خروجی نهایی شبکه قرار دهیم به این منظور که هر یک از نورون‌ها در صورتی فعال شود ورودی عضو یکی از دسته داده‌ی داخل رینگ باشد. بدین صورت به راحتی می‌توانیم داده‌هایی را که متعلق به یکی از دسته‌های داخل رینگ باشند را با فعال شدن نورون مربوطه‌اش بیابیم. طبیعتاً 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 خروجی یک را فعال کرده)

[0, 1]: داده‌ی ورودی عضو دسته‌ی دوم (چون خروجی 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)

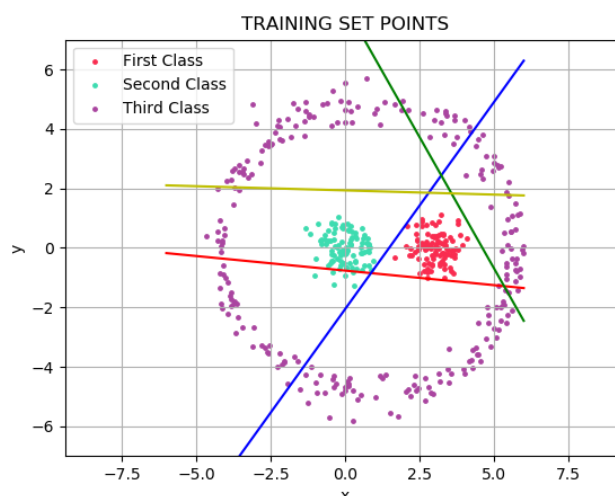
لذا با این تقسیم بندی کافی است که یک سری از نورون‌های لایه میانی را به خروجی یک متصل می‌کردیم و یک سری را به خروجی دو و هر قسمت را به صورت جداگانه آموزش دهیم. برای شفافیت موضوع شکل ورودی را ترسیم می‌کنیم.



در شکل فوق داده‌های ورودی را ترسیم کردیم که پیاده‌سازی را دقیق‌تر شرح دهیم. مثلاً در صورتیکه مجموعه داده‌ی قرمز را دسته اول بگیریم و بخواهیم وزن‌های نورون‌های متصل به خروجی یک را آموزش دهیم، کافی است بقیه‌ی داده‌ها 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 آموزش می‌یابد. لازم به ذکر است که ما در ابتدا برای هر نورون خروجی 3 نورون در لایه‌ی میانی (مجموعه‌اش نورون در لایه میانی) در نظر گرفتیم که وزن‌ها به سختی آموزش می‌دیدند و لذا تعداد نورون متصل به هر خروجی را یکی اضافه کردیم. نهایتاً شبکه‌ای با 8 نورون میانی توانست جواب مطلوبی را پس از 609 ایتريشن 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 ساختار and را برای هر یک از خروجی‌ها در این قسمت تعریف کردیم تا چهار خط 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 نورون خروجی استخراج کنند تا این چهارضلعی به درستی فیت شود. بدین منظور ساختار آپدیت وزن‌ها در الگوریتم MRI باعث می‌شد که تا زمانی که خروجی نورون‌های میانی متصل به نورون خروجی یک نشوند، خروجی نهایی هم یک نشود و این دقیقاً با ساختار and ما تطابق داشت. حساسیت این الگوریتم به نرخ یادگیری بسیار بالاست و به همین جهت نرخ یادگیری را بسیار پایین و در اردر پنج هزارم گرفتیم.

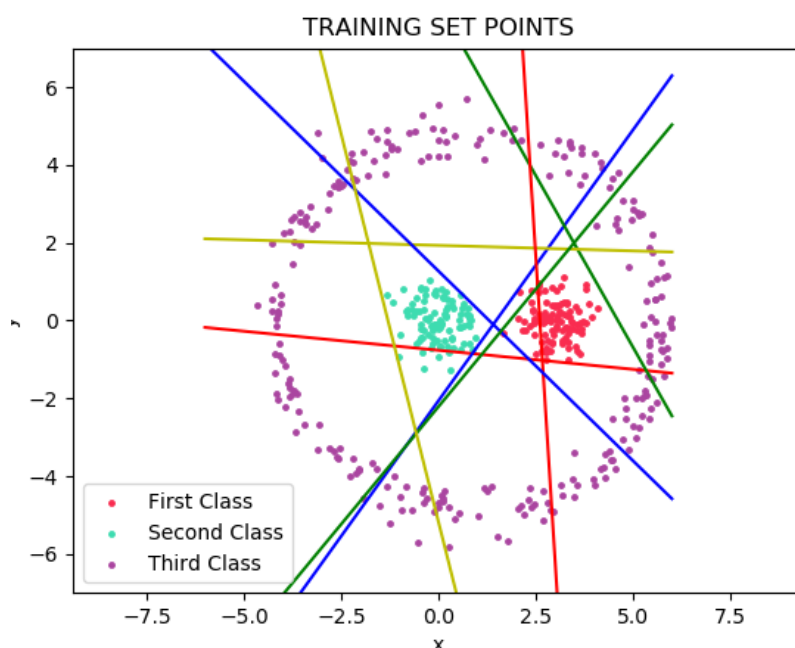


## بخش ب



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 نورن خروجی مربوط به خود تفکیک می‌شوند و در صورتیکه هیچ یک از نورن‌های فعال نشود نتیجه می‌گیریم که داده‌ی ورودی متعلق به رینگ است که توضیحات 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 مانع عبور پارامترها از مقدار بهینه شود. و لذا اگر همین لرنینگ ریت زیاد بود باعث ایجاد پرش در وزن ها هنگام آپدیت نمودن آنها می شد. لذا این عامل می تواند در زمانیکه نزدیک به نقطه ی بهینه است نه تنها کند بلکه سریعتر ما را به جواب برساند.

---