دانشگاه تهران

پردیس دانشکده‌های فنی

دانشکده مهندسی برق و کامپیوتر

گزارش تمرین سری اول درس شبکه

یلدا فروتن

۸۱۰۱۹۶۲۶۵

استاد درس

جناب آقای دکتر کلهر

پاییز ۹۸

۱.۱ طراحی یک شبکه تک‌لایه برای جداسازی دو نوع گل با روش Linear Perceptron

* توجه شود برای کارکرد کد مربوط به این بخش لازم است فایل دیتاست توسط گوگل درایو آپلود گردد.

در این مساله یک دیتاست شامل دو فیچر طول گلبرگ و کاسبرگ برای دو نوع گل داده شده است. هدف جداسازی این دو دسته است. برای استفاده از دیتاست داده شده از کتابخانه pandas استفاده شده است.

بنابراین لازم است از دیتاست داده شده دو ستون اول به عنوان دو ورودی درنظر گرفته شوند. برای این کار ستون flower از دیتاست که نماینده خروجی شبکه میباشد، حذف شده است. ستون flower شامل اسم دو نوع گل مدنظر است. بدیهی است باید یک لیبل به نام هر گل اختصاص داده شود. برای این منظور یک ماتریس target با ابعاد 1*100 ساخته شده است که شامل لیبل‌های ۱ و ۱- است.

به دلیل وجود دو ورودی (طول گلبرگ و کاسبرگ) ماتریس وزن به صورت 2*1 و بایاس به صورت یک عدد اسکالر درنظر گرفته شده است. همچنین یک پارامتر ترشهلد برای ایجاد حاشیه امن به اندازه ۰.۱ قرار داده شده است.

$$net = w_1 x_1 + \cdots . + w_i x_i + \cdots + w_n x_n + b$$

$$(\#)\quad h = \begin{cases} 1 & if \quad net > \theta \quad Active \\ 0 & if \quad |net| < \theta \quad non-descision \\ -1 & if \quad net < -\theta \quad Passive \end{cases}$$

$$\theta : non-negative\ threshold$$

طبق رابطه بالا، در ۱۰ ایپاک، مقادیر وزن در هر سطر از ماتریس ورودی ضرب شده و با بایاس جمع و با مقدار ترشهلد مقایسه شده است. سپس یک ماتریس خطا برای قدرمطلق اختلاف ماتریس target و predict ساخته شده است. حال اگر خطا برای هرسطر از ورودی مخالف صفر باشد، مقادیر w و b طبق رابطه زیر آپدیت می‌شوند.

```
w    =  w + alfa * x[i,:] * target[i]
b    =  b + alfa * target[i]
```

در این حالت مقادیر ماتریس وزن و بایاس به صورت زیر خواهد بود:

```
Weights =  [[ 0.146 -0.18 ]] , Bias =  [0.06]
```

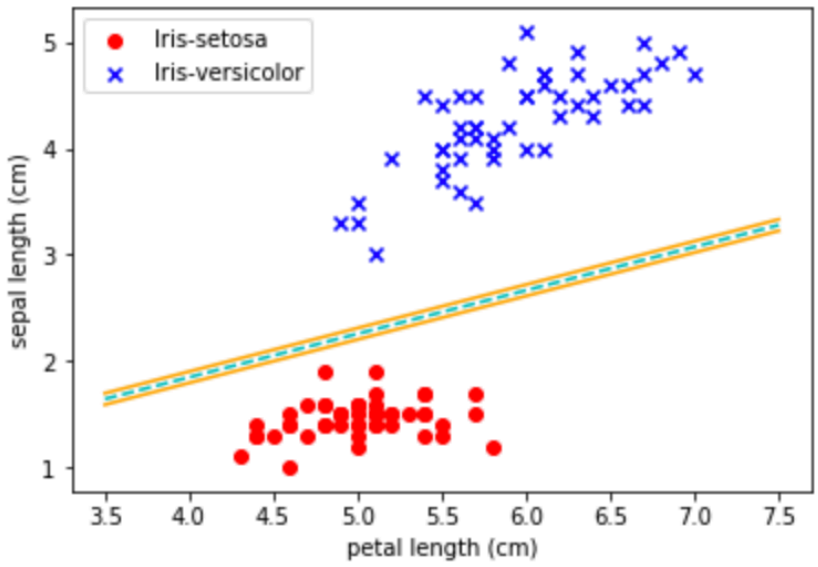
همانطور که پیش‌تر آمده است، در معادله net ورودی‌ها، وزن و بایاس قرار دارد. حال اگر مقادیر train شده در معادله net قرار داده شود، میتوان یک خط جداساز بین دو دسته ورودی قرار داد.

$$net = w_{0,0}.x_1 + w_{0,1}.x_2 + b = 0 \rightarrow \boldsymbol{x2 = \frac{-b - w_{0,0}.x_1}{w_{0,1}}}$$

مقادیر به صورت یک بازه پیوسته به صورت [3.5, 8.5] داده شده است. حال با استفاده از معادله سبزرنگ، مقادیر x2 برای این بازه محاسبه میشود. همچنین در معادله net یکبار مقدار ترشهلد اضافه و بار دیگر کم شده است تا حاشیه امن نیز در شکل نهایی قابل مشاهده باشد. درنهایت سه معادله به صورت زیر نوشته شده است.

```
x1_line = np.arange(3.5,8.5)
x2_line1 = (-b - w[0,0] * x1_line)/w[0,1]
x2_line2 = (-b - w[0,0] * x1_line + threshold)/w[0,1]
x2_line3 = (-b - w[0,0] * x1_line - threshold)/w[0,1]
```

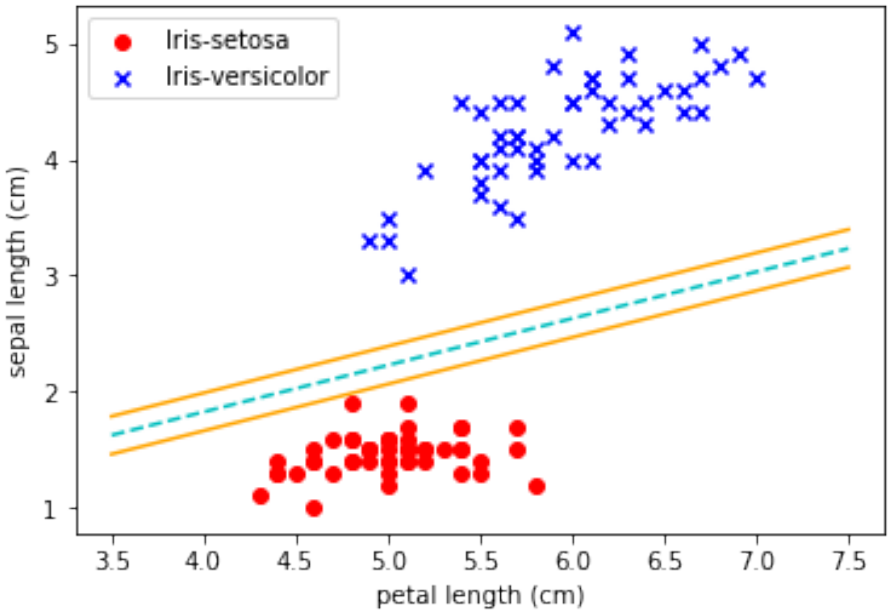
در ادامه نقاط ورودی براساس دو فیچر طول گلبرگ و کاسبرگ رسم شده و خطوط جداساز نیز قابل مشاهده است. شکل زیر برای مقدار lerning rate = 0.6 و threshold = 0.3 رسم شده است.



شکل ۱ – جداسازی دو کلاس قرمز و آبی با استفاده از شبکه پرسپترون

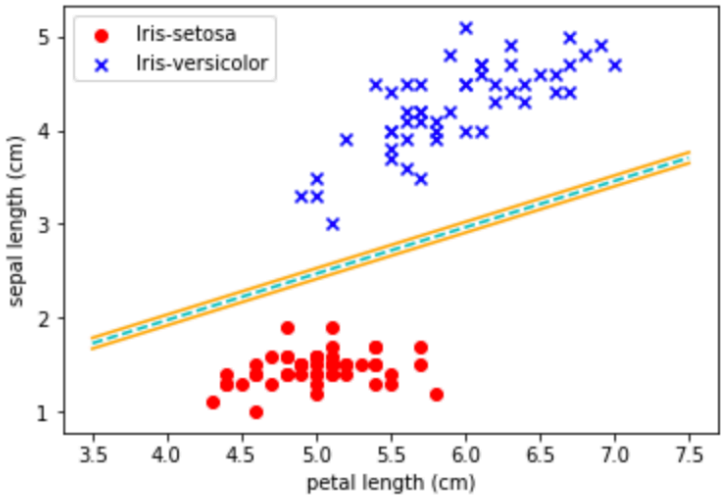
۲.۱ بررسی تاثیرات تغییرات آلفا و حذف بایاس

با کاهش الفا ممکن اس جداسازی دچار مشکل شود. مثلا برای آلفا ۰.۰۲ جداسازی دچار اشتباه می‌شود.



شکل ۲ – جداسازی دو کلاس قرمز و آبی با استفاده از شبکه پرسپترون و اثر تغیر آلفا

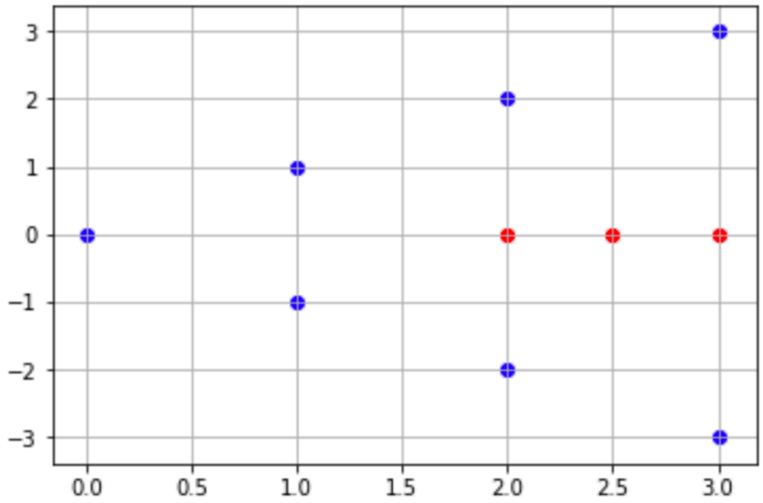
همانطور که در شکل زیر قابل مشاهده است با حذف بایاس تفکیک دو کلاس دچار مشکل نمیشود. البته این موضوع بسته به مقادیر آلفا و ترشهلد دارد. مثلا برای الفا ۰.۱ اگر بایاس حذف شود، جداسازی دچار مشکل میشود.



شکل ۳ – جداسازی دو کلاس قرمز و آبی با استفاده از شبکه پرسپترون و اثر تغییر ترشهلد

۱.۳.۱ جداسازی کلاس قرمز و آبی

هدف از این بخش، جداسازی دو کلاس قرمز و آبی از یکدیگر است. لازم به ذکر است در کلاس قرمز، مختصات نقاط اشتباه است و از مختصات نقاط روی شکل استفاده شده است.



شکل ۴ – دو کلاس قرمز و آبی

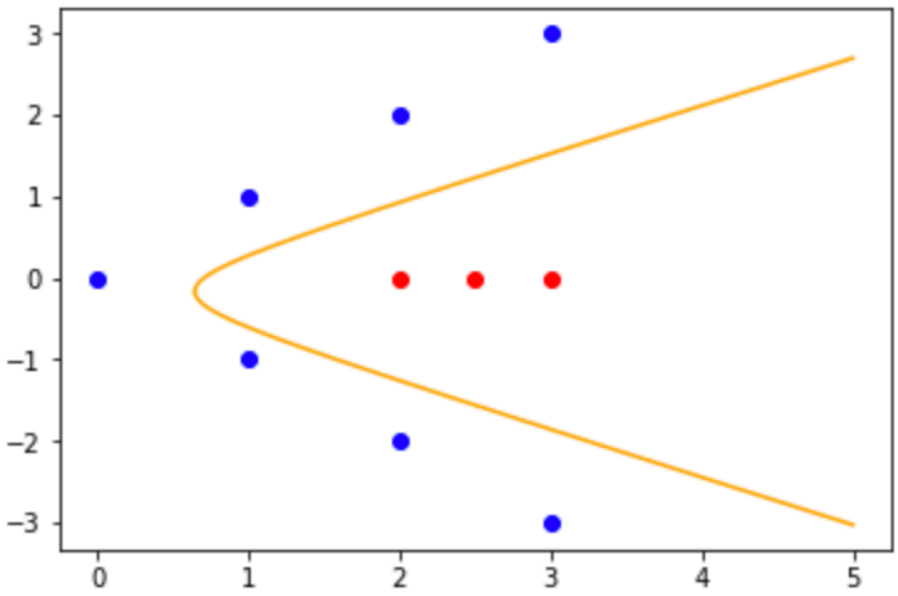
در ابتدا دو کلاس براساس مختصات نقاط درنظر گرفته شد. سپس یک ماتریس بر اساس اتصال دو کلاس و توان مقادیر آنها به عنوان ماتریس ورودی ساخته شد. بدیهی است که ماتریس ورودی دارای ابعاد 4*10 است. براساس ابعاد ماتریس ورودی، مختصات و توان دو مختصات نقاط، یک بردار وزن ۴ تایی و یک بایاس درنظر گرفته شد. همچنین یک بردار target ساخته شد به گونه ای که سه درایه اول آن مربوط به نقاط قرمز است و مقدار 1+ را دارند. ۷ درایه بعدی بردار target مربوط به نقاط آبی بوده و مقدار 1- به آنها اختصاص داده شده است. مقادیر ترشهلد و آلفا نیز به ترتیب ۰.۳ و ۰.۲ درنظر گرفته شده است. سپس برای هر درایه از ماتریس ورودی، بردار وزن در آن ضرب شده و با بایاس جمع شده است. در نهایت با مقدار ترشهلد مقایسه گردیده است و ماتریس predict ساخته شده است. درنهایت با قاپده مربوط به پرسپترون، مقادیر وزن و بایاس آپدیت شده اند که در ادامه آمده است. لازم به ذکر است مقدار ایپاک به طور اختیاری ۲۰ انتخاب شده است.

```
Weights =  [ 0.4 -1.2  0.  -0.4] , Bias =  -0.2
```

برای رسم خط جداکننده دو کلاس، هدف آن بود که یک بیضی دور نقاط کشده شود. بنابراین معادله مربوط به بیضی با ضرایب ماتریس وزن نوشته شد. اما درنهایت یک هذلولی کشیده شده است. به آن دلیل که معادله بیضی و هذلولی در یک ضریب منفی تفاوت دارند. معادله مربوط به هذلولی یا همان بیضی در ادامه آمده

است. برای این منظور ابتدا به متغیر x نقاطی در بازه ۰ تا ۵ اختصاص داده شده است و مقدار y طبق معادله زیر محاسبه گردیده است. درنهایت شکل زیر مشاهده شد.

$$y = \pm\sqrt{-w[0](\frac{x + w[2]}{2w[0]})^2 + \frac{(w[2])^2}{4w[0]} + \frac{(w[3])^2}{4w[1]} - \frac{b}{w[1]}} - \frac{w[3]}{2w[0]}$$

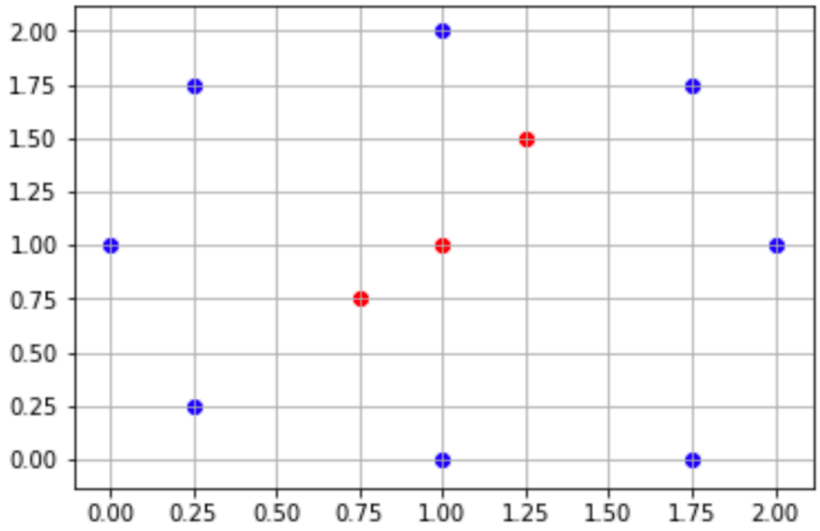


شکل ۵ – چگونگی جداسازی دو کلاس قرمز و آبی با هذلولی

بدیهی است اگر بازه x بیشتر مقادیر منفی را دربرگیرد، قرینه خط نارنجی قابل مشاهده خواهد بود.

۲.۳.۱ جداسازی کلاس قرمز و آبی

در این قسمت نیز هدف جداسازی نقاط قرمز از آبی است. نحوه قرار گرفتن این نقاط در شکل زیر آمده است.

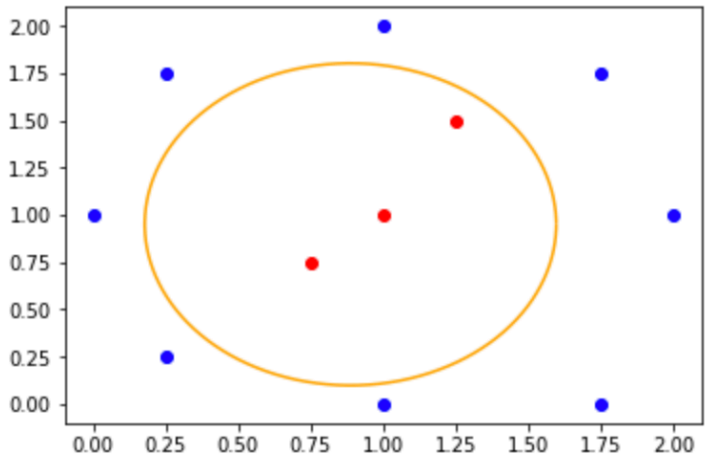


شکل ۶ – دو کلاس قرمز و آبی

همانند قسمت قبل یک ماتریس از مختصات و توان آنها ساخته شده است. بدیهی است ماتریس ورودی دارای ابعاد ۴*۱۱ است. همچنین بردار target برای نقاط ساخته شده که ۳ درایه اول آن دارای مقادیر ۱ و سایر درایه ها دارای مقادیر ۱- هستند. درنهایت مقادیر وزن ها و بایاس پس از ۸۰ ایپاک به صورت زیر مشاهده شد.

```
Weights =  [-2.2   -1.525  3.9    2.9  ] , Bias =  -1.9999999999999998
```

نحوه مقداردهی برای رسم خط جداساز همانند حال قبل بود. به گونه‌ای که به ورودی از بازه ۰ تا ۲ مقدار داده و براساس معادله بیضی و قراردادن مقادیر وزن‌ها و بایاس در معادله آن، همانند حالت قبل، جداسازی به صورت زیر مشاهده گردید.

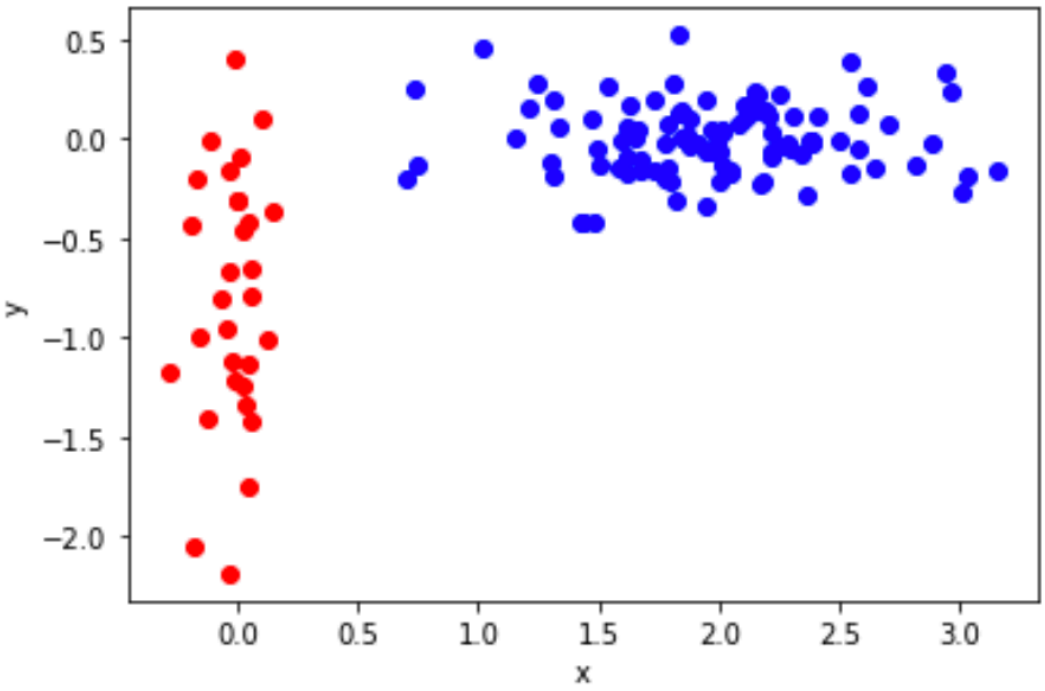


شکل ۷ – چگونگی جداسازی دو کلاس قرمز و آبی با بیضی

۱.۲ جداسازی دو کلاس با توزیع متفاوت توسط Adaline

هدف از این بخش،جداسازی دو شبکه با استفاده از روش آدلاین است. برخلاف قسمت قبل، دو کلاس با اطلاعات داده شده ساخته شدند. در ادامه نحوه ساخت این دو کلاس آمده است. همچنین شکل مربوط به داده‌های تعریف شده برای دو کلاس آبی و قرمز قابل مشاهده است.

```
x1 =  2 + 0.5 * np.random.randn(100, 1)
y1 =  0 + 0.2 * np.random.randn(100, 1)
xi1 = np.concatenate((x1, y1), axis = 1)

x2 =  0 + 0.1 * np.random.randn(30, 1)
y2 = -1 + 0.7 * np.random.randn(30, 1)
xi2 = np.concatenate((x2, y2), axis = 1)

xi = np.concatenate((xi1, xi2), axis = 0)
```



شکل ۸ – دو کلاس قرمز و آبی با توزیع نامتقارن

سپس ماتریس target به صورت ۱۰۰ درایه اول با مقدار ۱ و ۳۰ درایه بعدی با مقدار ۱- ساخته شده است. بدیهی است که ماتریس ورودی یک ماتریس 130*2 است. بنابراین بردار وزن دارای دو ورودی است و بردار بایاس تک درایه‌ای است. در ابتدا برای هر داده، مقدار net براساس رابطه زیر محاسبه گردید و تابع sign بر آن‌ها پیاده شد.

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

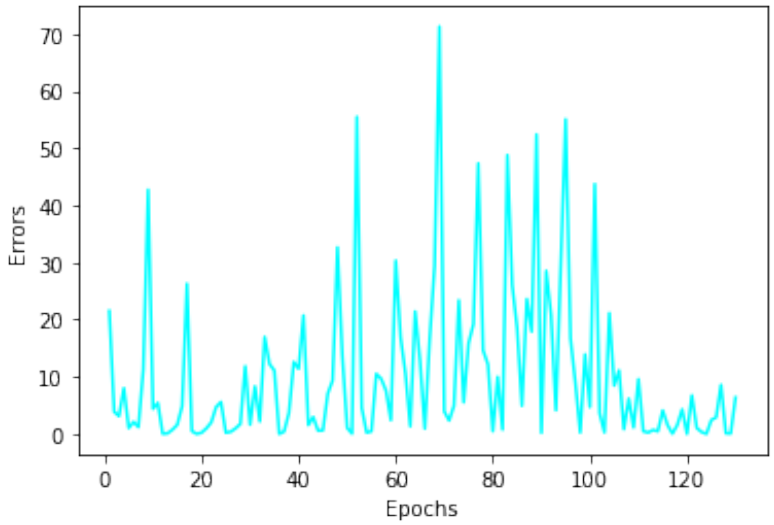
$$h = f(net) = \begin{cases} +1 & net \geq 0 \\ -1 & net < 0 \end{cases}$$

سپس براساس رابطه داده شده برای خطا، در ادامه قابل‌مشاهده است، مقادیر وزن و بایاس آپدیت شدند. درنهایت مقادیر آپدیت‌شده به صورت زیر مشاهده گردید:

errors = ½ (target-net)^2

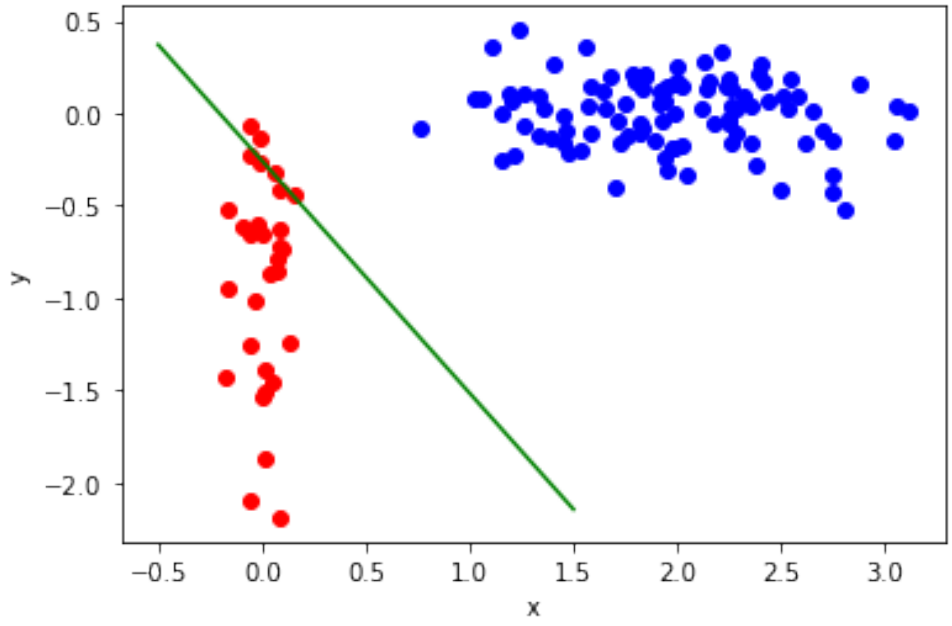
```
Weights =  [3.97150706 3.15808151] , Bias =  0.8138654732616875
```

درنهایت نمودار خطا براساس ایپاک به صورت زیر شد:



شکل ۹ – نمودار خطا برحسب تعداد ایپاک

همچنین الگوریتم، جداسازی را به صورت زیر انجام داد:



شکل ۱۰ – چگونگی جداسازی دو کلاس قرمز و آبی با یک خط

۲.۲ بررسی الگوریتم Adaline برای داده‌های با توزیع نامتقارن

استفاده از روش آدلاین برای دو کلاس ساخته شده در قسمت قبل مناسب نیست. زیرا روش آدلاین تاجایی که توزیع داده‌ها متقارن است خوب عمل میکند و یک hyperplane مناسب در وسط داده‌ها قرار میدهد که فاصله قابل توجهی از هر دو کلاس دارد. اما هنگامی که توزیع کلاس‌ها یکسان نیست، ممکن است جداساز را جایی قرار دهد که حتی جداسازی را به درستی انجام ندهد. حال این یکسان نبودن توزیع میتواند به شباهت داده‌ها یا جمعیت آنها برگردد. دلیل این اتفاق آن است که می‌خواهد عمده خطای ناشی از جمعیت بیشتر را کم کند و جایی قرار بگیرد که جداساز را به سمت جمعیت بیشتر خم کند. در قسمت قبل نیز چون جمعیت یک کلاس ۳۰ و دیگری ۱۰۰ بود جداسازی به درستی انجام نشد. بنابراین استفاده از روش آدلاین برای داده‌های با توزیع متفاوت (جمعیت و یا شباهت متفاوت) نامناسب است.

در راستای بهبود آدلاین برای حذف وابستگی آن به شکل و جمعیت توزیع‌ها، در محاسبه خطا، به جای استفاده از net از h استفاده می‌شود:

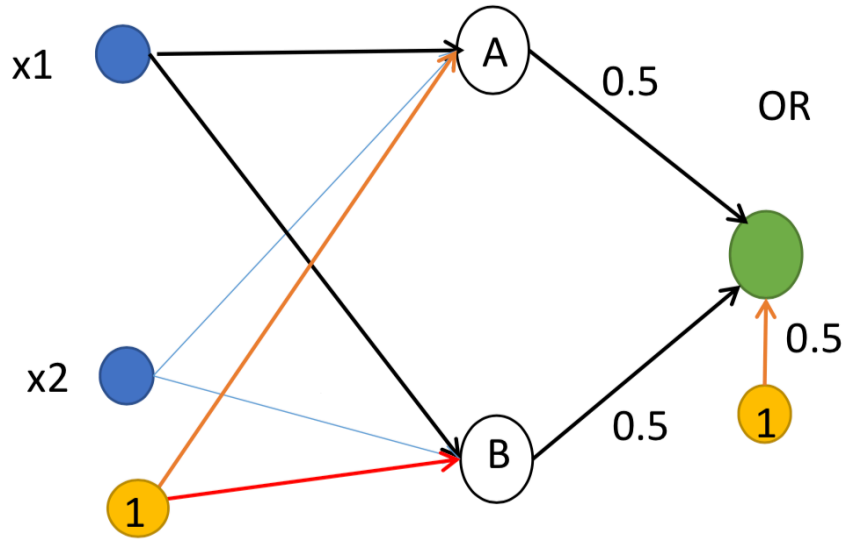
$$errors = \frac{1}{2} (target-h)^2$$

اما از آنجایی که گرادیان نیاز به مشتق h دارد شاید تابع sign مناسب نباشد و از توابع مشتق پذیر همانند tanh استفاده می‌شود.

۳. پیاده‌سازی گیت XNOR با استفاده از شبکه Madaline

در این بخش با استفاده از روش Madaline یک شبکه تعریف شده که میتواند عمل xor را انجام دهد. درواقع Madaline توسعه جدی‌تری از شبکه Adaline است. طبق الگوریتم MRI موجود در مرجع درس، که برای xor طراحی شده بود استفاده شده است. شبکه دارای دو لایه است. لایه اول از دو نرورن W&H ساخته شده است. درواقع وزن این دو نورون و بایاس همین لایه اول باید train شود. هر نورون گفته شده یک خط می‌سازند که فضا را به سه ناحیه تقسیم میکند. دو ناحیه از این سه ناحیه متعلق به کلاس - بوده و ناحیه دیگر مربوط به کلاس + است. بنابراین در لایه دوم نیاز از نواحی هم‌کلاس، با یکدیگر جمع یا همان or شوند. به این دلیل است که در لایه دوم از یک نورون برای ساخت or استفاده شده است. البته در الگوریتم MRI به این گونه است. در الگوریتم MEII لایه مربوط به or نیز train می‌شود.

معماری شبکه طراحی شده در ادامه آمده است.



شکل ۱۱ – معماری استفاده شده برای پیاده‌سازی گیت XNOR

در ابتدا دو ورودی x1 و x2 با مقادیر ۱ و -۱ برای تعیین ۴ حالت مختلف، تعریف شده‌اند. یک ماتریس target به عنوان جواب xnor نیز درنظر گرفته شده است. بدیهی است xnor درحالتی که ورودی‌ها یکسان

باشند، مقدار ۱ و درغیراین‌صورت مقدار ۱- را دارد. یک ماتریس وزن 2*2 و یک بردار بایاس 1*2 برای لایه اول لازم است که به صورت اختیاری مقداردهی شده اند. لایه دوم به یک ماتریس وزن 1*2 و یک بایاس احتیاج دارد. البته یک ماتریس بایاس 1*3 برای دو لایه درنظر گرفته شده است. در لایه اول دو مقدار z_in1 و z_in2 با استفاده از مجموع حاصلضرب وزن‌ها در هر دو ورودی ساخته شده است و علامت آنها تعیین میگردد و به عنوان ورودی لایه دوم مورداستفاده قرار می‌گیرند. ورودی‌های جدید در وزن مربوطه (۰.۵) ضرب شده و بایاس مربوطه (۰.۵) جمع شده و علامت آن محاسبه میگردد. حال مقدار خروجی شبکه با target مقایسه می‌شود و اگر یکسان نبود، حالت‌های زیر بررسی میشود.

- اگر target=-1، یعنی حداقل یکی از z1 یا z2 (علامت‌های z_in1 و z_in2)، ۱+ بوده است. درصورتی‌که باید هردو ۱- می‌بودند، بنابراین هرکدام که ۱+ بوده را باید آپدیت کرد.
- اگر target1، یعنی z1 و z2 هر دو ۱- بودند و باید یکی از آن‌ها ۱+ شود. هرکدوم که net آن بزرگتر است، آپدیت می‌شود.

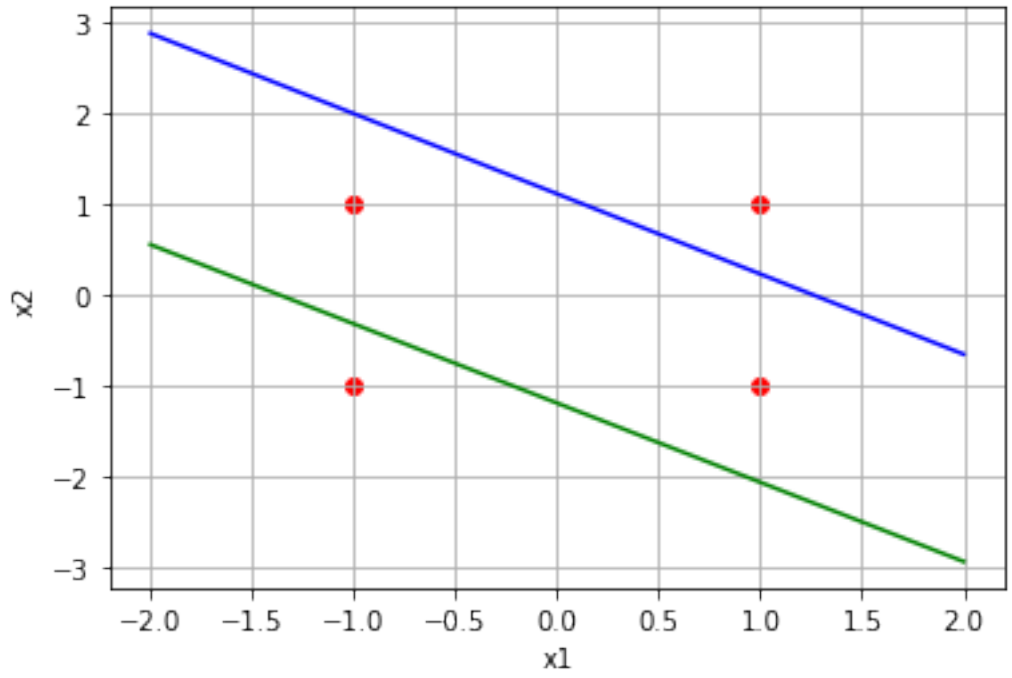
درنهایت مقادیر وزن و بایاس برای لایه اول به صورت زیر مشاهده گردید. این مقادیر برای الفا 0.69 است.

```
Weights =
[[ 1.92340564 -1.94108215]
 [ 2.17888506 -2.22584675]]
Bias =
[[-2.40436623]
 [-2.66978769]]
```

سپس معادله مربوط به دو خط جداکننده با استفاده از ضرایب وزن و بایاس رسم شد. یکی از ورودی‌ها در بازه ۲- تا ۳ مقداردهی شد. در ادامه معادله این دو خطوط آمده است.

```
x1_line = np.arange(-2, 3)
x2_line1 = (-b[0,0] - w[0,0] * x1_line)/w[1,0]
x2_line2 = (-b[1,0] - w[0,1] * x1_line)/w[1,1]
```

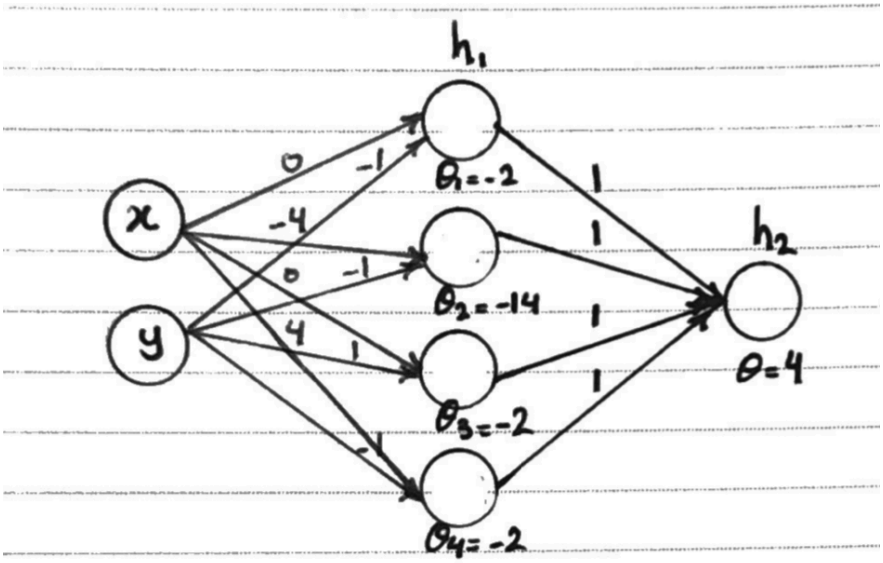
درنهایت دوخط جداساز به صورت شکل زیر عمل طبقه‌بندی را انجام دادند. بدیهی است بین خطوط مربوط به کلاس ۱+ و در طرفین مربوط به کلاس ۱- است.



شکل ۱۲ – چگونگی عملکرد گیت XNOR با دو خط

۴ – طراحی یک شبکه عصبی برای جداسازی درون و بیرون یک چهارضلعی محدب

طبق صورت مساله، باید امکان جداسازی درون و بیرون یک چهارضلعی محدب، در اینجا ذوزنقه، با استفاده از نورون‌های Mcculloch Pitts فراهم شود. این کار توسط دو لایه صورت گرفته است. در لایه اول با استفاده از هر دو نقطه مجاور معادله یک خط نوشته میشود، که در ادامه آمده است. حال با استفاده از ضرایب x و y در معادله هر خط، دو بردار وزن مربوط به x و y ساخته میشود. بدیهی است که مقادیر ثابت همان بایاس هستند. بنابراین دو بردار وزن و یک بردار بایاس با ابعاد 1*4 ساخته شده است. درواقع از هر نورون M&P برای رسم یک خط استفاده شده است. لازم به ذکر است که ماتریس‌های وزن، یک ماتریس با ابعاد 2*4 است که ستون اول برای ورودی x و ستون دوم برای ورودی y درنظر گرفته شده است. اما در محاسبات برای راحتی از دو ماتریس w_x و w_y استفاده شده است.



شکل ۱۳ – معماری شبکه برای تفکیک درون و برون یک چهارضلعی محدب

در لایه دوم، با استفاده از یک نورون M&P یک گیت and چهار ورودی طراحی شده است. حال میتوان دو کار کرد. معادله خط را مساوی صفر قرار داد و مقدار بایاس‌ها را با سایر ترم‌ها جمع کرد و عمل sign را بر روی معادله پیاده کرد و یا ترم بایاس به عنوان ترشهلد استفاده شود. در اینجا بایاس در معادله مربوط به خطوط به کار رفته و درنهایت net با صفر مقایسه شده است. بدیهی است مقدار بایاس‌ها برای گیت and، همان تعداد ورودی‌ها خواهد بود یعنی در اینجا ـ۴ است.

مجموع حاصل ضرب های بردار وزن هر ورودی در بردار ورودی مربوطه محاسبه شده است و با بردار بایاس جمع شده است.

در لایه دوم یک بردار وزن 1*4 با مقادیر یک و بایاس ۴- درنظر گرفته شده است. بدیهی است در این لایه، خروجی لایه قبل به عنوان ورودی است.

همچنین میتوان از سه لایه استفاده کرد به گونه ای که از چهار خروجی لایه اول، هر دو خروجی را به عنوان ورودی های نورون های M&P برای ساخت یک and استفاده کرد و در لایه سوم یک and دیگر استفاده کرد.

در ادامه جدول مربوط به ضرایب برای دو لایه آمده است.

جدول ۱ – مقادیر وزن‌ها و بایاس برای دو لایه شبکه

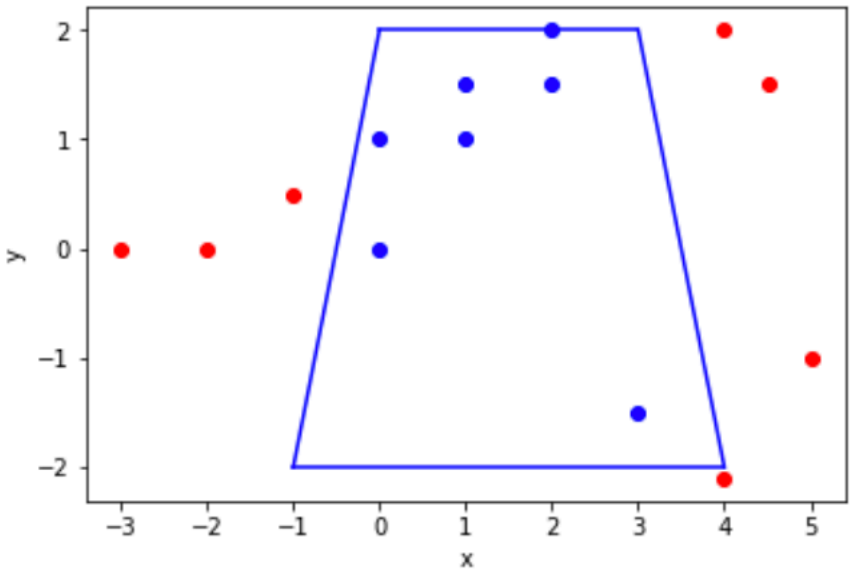
Line		Layer 1			Layer 2	
		w_x	w_y	b	w_and	b_and
y-2=0	below	0	1	-2	1	-4
y+4x-14=0	below	4	1	-14	1	
y+2=0	abow	0	1	2	1	
y-4x-2=0	below	-4	1	-2	1	

همچنین ۱۴ نقطه برای تست الگوریتم داده شده است. درصورتی که نقاط درون ذوزنقه باشند، به رنگ آبی نشان داده می‌شوند و اگر خارج از چهارضلعی باشند، به رنگ قرمز درمی‌آیند.

```
#inside
input_( 1,   1)
input_( 3,-1.5)
input_( 1, 1.5)
input_( 0,   0)
input_( 2, 1.5)
input_( 2,   2)
input_( 0,   1)
#outside
input_(4 ,-2.1)
input_(-2,   0)
input_(-3,   0)
input_(-1, 0.5)
input_(4 ,   2)
input_(5 ,  -1)
input_(4.5,1.5)

plt.show()
```

```
( 1 , 1 ) IS inside of the convex polygon.
( 3 , -1.5 ) IS inside of the convex polygon.
( 1 , 1.5 ) IS inside of the convex polygon.
( 0 , 0 ) IS inside of the convex polygon.
( 2 , 1.5 ) IS inside of the convex polygon.
( 2 , 2 ) IS inside of the convex polygon.
( 0 , 1 ) IS inside of the convex polygon.
( 4 , -2.1 ) ISN'T inside of the convex polygon.
( -2 , 0 ) ISN'T inside of the convex polygon.
( -3 , 0 ) ISN'T inside of the convex polygon.
( -1 , 0.5 ) ISN'T inside of the convex polygon.
( 4 , 2 ) ISN'T inside of the convex polygon.
( 5 , -1 ) ISN'T inside of the convex polygon.
( 4.5 , 1.5 ) ISN'T inside of the convex polygon.
```

شکل ۱۴ – مقادیر استفاده شده برای آزمودن شبکه



شکل ۱۵ – نحوه جداسازی درون و برون یک چهارضلعی محدب

Codes

1.1.

```
[1] #upload csv file from google drive
    import pandas as pd
    url = '/content/HW1_Q1_data.csv'
    dataset = pd.read_csv(url)

    #dataset
```

```
[2] import numpy as np
    x = dataset.drop(columns = 'flower')
    x = x.values
    y = dataset['flower'].values

    #x.shape, x, y
```

```
[ ] target = np.where(y == 'Iris-setosa', 1, -1)
    target = np.reshape(target, (100,1))

    #target.shape, target
```

```
[ ] import numpy as np
    w = np.zeros([1, 2])
    b = 0
    threshold = 0.3
    alfa = 0.2
    predict = np.zeros([100, 1])
    net = np.zeros([100, 1])
    epochs = 0
    errors = np.zeros([100,1])

    w,b
```

```
[ ] i=0
    for epochs in range (10):
      for i in range(100):
        net = np.matmul(w, x[i,:])+ b
        if net > threshold:
          predict = 1
        elif  net < threshold:
          predict = -1
        else:
          predict = 0
        errors[i] = abs(predict-target[i])
        if errors[i]!=0:
          w   =  w + alfa * x[i,:] * target[i]
          b   =  b + alfa * target[i]

    print("Weights = ",w,", Bias = ",b)
```

```
[ ] x1_line = np.arange(3.5,8.5)
    x2_line1 = (-b - w[0,0] * x1_line)/w[0,1]
    x2_line2 = (-b - w[0,0] * x1_line + threshold)/w[0,1]
    x2_line3 = (-b - w[0,0] * x1_line - threshold)/w[0,1]

    #x1_line
```

```
    import matplotlib.pyplot as plt
    plt.scatter(x[:50, 0], x[:50, 1], color='red', marker='o', label='Iris-setosa')
    plt.scatter(x[50:100, 0], x[50:100, 1], color='blue', marker='x', label='Iris-versicolor')
    plt.plot(x1_line,x2_line1,'c--')
    plt.plot(x1_line,x2_line2,'orange')
    plt.plot(x1_line,x2_line3,'orange')
    plt.xlabel('petal length (cm)')
    plt.ylabel('sepal length (cm)')
    plt.legend(loc='upper left')
    plt.show()
```

1.3.1.

hw1_part1_c1.ipynb - Colaboratory

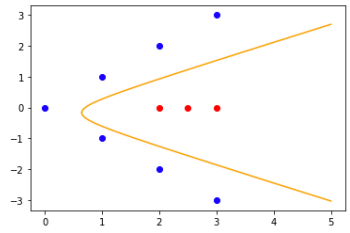
```
, 0], [3, 0]])
class2 = np.array([[0, 0], [1, 1], [2, 2], [3, 3], [1, -1], [2, -2], [3, -3]])

#class1.shape, class2.shape
```

```
[ ] x_class = np.concatenate((class1,class2),axis=0)
x_power = np.power(x_class,2)
x = np.concatenate((x_power, x_class),axis=1)

#x.shape, x.shape
```

```
[ ] w = np.zeros(4)
b = 0
target = np.array([1, 1, 1, -1, -1, -1, -1, -1, -1, -1])
threshold = 0.3
alfa = 0.2
epochs = 0
for epochs in range (20):
  for i in range (0, 10, 1):
    net = np.matmul(w, x[i]) + b
    if net > threshold:
      predict = 1
    elif  net < threshold:
      predict = -1
    else:
      predict = 0
    if predict != target[i]:
      for j in range(0,4,1):
        w[j]   =  w[j] + alfa * target[i] * x[i,j]
      b    =  b + alfa * target[i]

print("Weights = ",w,", Bias = ",b)
```

```
Weights =  [ 0.4 -1.2  0.  -0.4] , Bias =  -0.2
```

```
import matplotlib.pyplot as plt

plt.plot(x[target ==  1, 2], x[target ==  1, 3], 'ro')
plt.plot(x[target == -1, 2], x[target == -1, 3], 'bo')

x_line = np.linspace(0,5,50000)
delta  = (-w[0]*np.power(x_line+w[2]/(2*w[0]),2)+ np.power(w[2],2)/(4*w[0]) + np.power(w[3],2)/(4*w[1]) - b)/w[1]
x_line = x_line[delta > 0]
delta  = delta [delta > 0]
y_line_p =  np.sqrt (delta) - w[3]/(2*w[1])
y_line_n = -np.sqrt (delta) - w[3]/(2*w[1])
plt.plot(x_line, y_line_p, 'orange')
plt.plot(x_line, y_line_n, 'orange')
plt.show()
```



1.3.2.

```
import numpy as np
class1 = np.array([[0.75, 0.75], [1, 1], [1.25, 1.5]])
class2 = np.array([[0, 1], [0.25, 1.75], [1, 2], [1.75, 1.75], [2, 1], [1.75, 0], [1, 0], [0.25, 0.25]])

#class1.shape, class2.shape
```

```
x_class = np.concatenate((class1,class2),axis=0)
x_power = np.power(x_class,2)
x = np.concatenate((x_power, x_class),axis=1)

#x_class, x_power
#x.shape, x
```

```
w = np.zeros(4)
target = np.array([1, 1, 1, -1, -1, -1, -1, -1, -1, -1, -1])
b = 0
threshold = 0.1
alfa = 0.2
epochs = 0
for epochs in range (80):
  for i in range (0, 11, 1):
    net = np.matmul(w, x[i]) + b
    if net > threshold:
      predict = 1
    elif  net < threshold:
      predict = -1
    else:
      predict = 0
    if predict != target[i]:
      for j in range(0, 4, 1):
        w[j]   =  w[j] + alfa * target[i] * x[i,j]
      b =  b + alfa * target[i]

print("Weights = ",w,", Bias = ",b)
```

```
import matplotlib.pyplot as plt

plt.plot(x[target ==  1, 2], x[target ==  1, 3], 'ro')
plt.plot(x[target == -1, 2], x[target == -1, 3], 'bo')

x_line = np.linspace(0, 2, 50000)
delta  = (-w[0]*np.power(x_line+w[2]/(2*w[0]),2)+ np.power(w[2],2)/(4*w[0]) + np.power(w[3],2)/(4*w[1]) - b)/w[1]
x_line = x_line[delta > 0]
delta  = delta [delta > 0]
y_line_p =  np.sqrt (delta) - w[3]/(2*w[1])
y_line_n = -np.sqrt (delta) - w[3]/(2*w[1])
plt.plot(x_line, y_line_p, 'orange')
plt.plot(x_line, y_line_n, 'orange')
plt.show()
```

2.

```
[262] import numpy as np


      x1 =  2 + 0.5 * np.random.randn(100, 1)
      y1 =  0 + 0.2 * np.random.randn(100, 1)
      xi1 = np.concatenate((x1, y1), axis = 1)

      x2 =  0 + 0.1 * np.random.randn(30, 1)
      y2 = -1 + 0.7 * np.random.randn(30, 1)
      xi2 = np.concatenate((x2, y2), axis = 1)

      xi = np.concatenate((xi1, xi2), axis = 0)

      target = np.concatenate((np.ones((1, 100)), -1*np.ones((1, 30))), axis=1)
      target = np.squeeze(target)

      #xi.shape
      #xi1, xi2 ,xi
      #target.shape, target
```

```
[263] w = np.zeros(2)
      b = 0
      alfa = 0.09
      errors = np.zeros((130))

      #w, wp, b, alfa, errors
```

```
[264] for epochs in range (0,130,1):
        for i in range(0, 130, 1):
          net = np.matmul(w, xi[i]) + b
          errors[i] = 0.5 * ((target[i] - net)**2)
          for j in range(0, 2, 1):
            w[j] = w[j] + alfa * (target[i] - net)*xi[i, j]
          b = b +  alfa * target[i] - net

      print("Weights = ",w,", Bias = ",b)
```

```
[266] import matplotlib.pyplot as plt
      epochs = np.arange(1,131)
      plt.figure(0)
      plt.plot(epochs, errors, 'cyan')
      plt.xlabel('Epochs')
      plt.ylabel('Errors')
      plt.show()
```

```
      x_line = np.arange(-0.5, 2)
      y_line = (-b - w[0] * x_line)/w[1]
      plt.figure(1)
      plt.plot(xi[target == -1, 0], xi[target == -1, 1], 'ro')
      plt.plot(xi[target ==  1, 0], xi[target ==  1, 1], 'bo')
      plt.xlabel('x')
      plt.ylabel('y')
      plt.plot(x_line, y_line, 'green')
      plt.show()
```

3.

```
import numpy as np

x1 = np.array([1,  1, -1, -1])
x2 = np.array([1, -1,  1, -1])

target = np.array([1, -1, -1, 1])

v = np.array([[0.5],[0.5]])
b = np.array([[0.3],[0.15],[0.5]])
w = np.array([[0.05,0.1],[0.2,0.2]])

alfa = 0.69
```

```
[52] def sign(x):
       if x >= 0:
         yt =  1
       else:
         yt = -1
       return(yt)
```

```
[53] flag = True
     epoch =0
     while(flag):
       flag = False
       for i in range (0,4):
         z_in1 = x1[i] * w[0,0] + x2[i] * w[1,0] + b[0,0]
         z_in2 = x1[i] * w[0,1] + x2[i] * w[1,1] + b[1,0]
         z1 = sign(z_in1)
         z2 = sign(z_in2)
         y_in = v[0,0] * z1 + v[1,0] * z2 + b[2,0]
         y = sign(y_in)
         if (-target[i]+y) == 0:
           error=0
         else:
           if target[i] ==  -1:
               if z_in1>0:
                   b[0,0] = b[0,0] + alfa*(-1-z_in1)
                   w[0,0] = w[0,0] + alfa*(-1-z_in1)*x1[i]
                   w[1,0] = w[1,0] + alfa*(-1-z_in1)*x2[i]
               if z_in2>0 :
                   b[1,0] = b[1,0] + alfa*(-1-z_in2)
                   w[0,1] = w[0,1] + alfa*(-1-z_in2)*x1[i]
                   w[1,1] = w[1,1] + alfa*(-1-z_in2)*x2[i]

               #flag = True
           if target[i] == 1:
               if abs(z_in1)<abs(z_in2):
                   b[0,0] = b[0,0] + alfa*(1-z_in1)
                   w[0,0] = w[0,0] + alfa*(1-z_in1)*x1[i]
                   w[1,0] = w[1,0] + alfa*(1-z_in1)*x2[i]
               else:
                   b[1,0] = b[1,0] + alfa*(1-z_in2)
                   w[0,1] = w[0,1] + alfa*(1-z_in2)*x1[i]
                   w[1,1] = w[1,1] + alfa*(1-z_in2)*x2[i]
               flag = True
       epoch = epoch +1


     print("Weights = ")
     print(w)
     print("Bias =" )
     print(b[0:2])
```

```
[54] x1_line = np.arange(-2, 3)
     x2_line1 = (-b[0,0] - w[0,0] * x1_line)/w[1,0]
     x2_line2 = (-b[1,0] - w[0,1] * x1_line)/w[1,1]
```

```
[55] import matplotlib.pyplot as plt
     plt.scatter(x1, x2, color='red', marker='o')
     plt.plot(x1_line,x2_line1,'b')
     plt.plot(x1_line,x2_line2,'g')
     plt.xlabel('x1')
     plt.ylabel('x2')
     #plt.xlim(-1,0.5)
     plt.grid()
     plt.show()
```

4.

```
import numpy as np

x = np.array([ 0,  3,   4,  -1])
y = np.array([ 2,  2,  -2,  -2])

w_x = np.array([  0,    4,   0,  -4])
w_y = np.array([  1,    1,   1,   1])
b   = np.array([ -2,  -14,   2,  -2])

w_and = np.ones(4)
b_and = -4
```

```
def line(x, y, w_x, w_y, b):
  net = w_x * x + w_y * y + b
  if net <= 0:
    return True
  else:
    return False

def convex(x, y):
  a_line = line(x, y,  w_x[0],  w_y[0],  b[0])
  b_line = line(x, y,  w_x[1],  w_y[1],  b[1])
  c_line = line(x, y, -w_x[2], -w_y[2],  -b[2])
  d_line = line(x, y,  w_x[3],  w_y[3],  b[3])

  out = a_line + b_line + c_line + d_line + b_and
  if out >= 0 :
    return True
  else:
    return False
```

```
def input_(a, b):
  if convex(a, b)==True:
    print("(",a,",",b,")","IS inside of the convex polygon.")
    plt.plot(a, b, 'bo')
  else:
    print("(",a,",",b,")","ISN'T inside of the convex polygon.")
    plt.plot(a, b, 'ro')
```

```
import matplotlib.pyplot as plt

x_line1 = np.linspace( 0,  3, 500)
y_line1 = np.linspace( 2,  2, 500)
plt.plot(x_line1, y_line1, 'blue')

x_line2 = np.linspace( 3,  4, 500)
y_line2 = np.linspace( 2, -2, 500)
plt.plot(x_line2, y_line2, 'blue')

x_line3 = np.linspace( 4, -1, 500)
y_line3 = np.linspace(-2, -2, 500)
plt.plot(x_line3, y_line3, 'blue')

x_line4 = np.linspace(-1,  0, 500)
y_line4 = np.linspace(-2,  2, 500)
plt.plot(x_line4, y_line4, 'blue')
plt.xlabel('x')
plt.ylabel('y')

#inside
input_( 1,   1)
input_( 3,-1.5)
input_( 1, 1.5)
input_( 0,   0)
input_( 2, 1.5)
input_( 2,   2)
input_( 0,   1)
#outside
input_(4 ,-2.1)
input_(-2,   0)
input_(-3,   0)
input_(-1, 0.5)
input_(4 ,   2)
input_(5 ,  -1)
input_(4.5,1.5)

plt.show()
```