به نام خدا

دانشگاه تهران

دانشکده مهندسی برق و کامپیوتر

شبکه های عصبی و یادگیری عمیق

تمرین سری اول

نام و نام خانوادگی	علی عدالت
شماره دانشجویی	۸۱۰۱۹۹۳۴۸
تاریخ ارسال گزارش	

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

سوال 1 – McCulloch-Pitts

الف و ج

در این جا ما دو نرون ورودی داریم. هر یک متناظر یکی از دو بمب مورد نظر برای انفجار است. این نرون‌ها با x_1 و x_2 نام گذاری می‌شوند. هر وقت بمب اول منفجر شود، نرون ورودی متناظر آن یک می‌شود و گرنه این نرون صفر است. در زمان انفجار هر بمب نرون ورودی متناظر یک می‌شود و یک step زمانی بعد، مقدار آن صفر می‌شود. در زمان انفجار بلافاصله نور انفجار دیده می‌شود اما صدای آن یک step بعد به گوش می‌رسد. پس یک نرون تعیین صدای انفجار داریم که نرون ورودی هر بمب با یک یال با وزن یک به آن متصل می‌شود. این نرون یک step بعد از انفجار یک می‌شود که به گوش رسیدن صدا را مدل می‌کند. این نرون را h_i می‌نامیم. i متناظر با شماره بمب تعیین می‌شود. فرد برای تشخیص انفجار هر بمب باید، هم نور انفجار و هم صدای آن را شنیده باشد. پس فرد باید در حافظه خود نور انفجار هر بمب را در صورت وقوع نگه دارد تا بتواند انفجار را تشخیص دهد. دلیل این موضوع هم زمان نبودن صدا و نور است. برای مدل کردن حافظه برای نور انفجار از یک نرون با نام m_i برای هر بمب استفاده می‌کنیم. این نرون با یال با وزن یک به ورودی متصل می‌شود. این نرون یک step بعد از انفجار یک می‌شود که هم زمان با یک شدن نرون شنیدن صدا است. ورودی این نرون تعیین کننده‌ی نور انفجار است. اگر انفجار رخ دهد، نرون ورودی یک می‌شود و هم زمان یال ورودی نرون m_i نیز یک می‌شود. این یعنی نور انفجار دیده شده است. یک step بعد نرون m_i یک می‌شود که یعنی یک step قبل نور انفجار دیده شده است. پس اگر هم m_i و هم h_i یک شوند یعنی هم صدا و هم نور انفجار بمب i دیده شده است و این یعنی بمب i منفجر شده است. برای جمع بندی تشخیص انفجار از خروجی نرون‌های m_i و h_i ، یک نرون s_i داریم که با یال به وزن یک به m_i و با یال به وزن یک به h_i متصل است. این نرون زمانی که انفجار رخ دهد ۲ می‌شود و گرنه صفر است. در این جا دو بمب با فاصله ۲ step منفجر می‌شوند و ما تداخل نوری و صدایی نداریم. این بدان معنا است که بعد از انفجار حتما نرون‌های حافظه نور و شنیده شدن صدا هم زمان یک می‌شوند. زمانی که s_i مقدار جدید در اثر انفجار بمب متناظر می‌گیرد، ۲ step از انفجار آن بمب گذشته است و بمب دوم همزمان منفجر می‌شود. یک step بعد صدای انفجار دوم نیز شنیده خواهد شد و فرد باید دکمه را بزند. پس به اندازه‌ی یک step باید مقدار s_i را نگه داریم تا بتوانیم انفجار بعدی را تشخیص دهیم. پس نرون z_i را تعریف می‌کنیم که با یال به وزن یک به s_i متصل است. فرض کنیم که اول بمب یک و بعد بمب دو منفجر شود. زمانی که z_1 برابر دو می‌شود، h_2 و m_2 همزمان هر دو یک هستند. پس در گام بعد می‌توان دکمه را فشار داد. برای مدل کردن این حالت از h_2 و m_2 به صورت مستقیم یال به وزن یک به

نرون خروجی وصل می‌کنیم. از z_1 نیز یال با وزن یک به نرون خروجی وصل می‌کنیم. ممکن است اول بمب دو منفجر شود. پس متناسب با حالت قبل یال‌هایی به نرون خروجی متصل می‌کنیم. نرون خروجی را y می‌نامیم که آستانه آن یا θ را برابر ۴ قرار می‌دهیم. اگر مجموع ورودی های این نرون بزرگتر مساوی ۴ باشد، خروجی آن یک می‌شود و گرنه خروجی آن صفر است. خروجی یک در این نرون به معنی فشار دادن دکمه است. شبکه طراحی شده در Figure 1 آمده است.

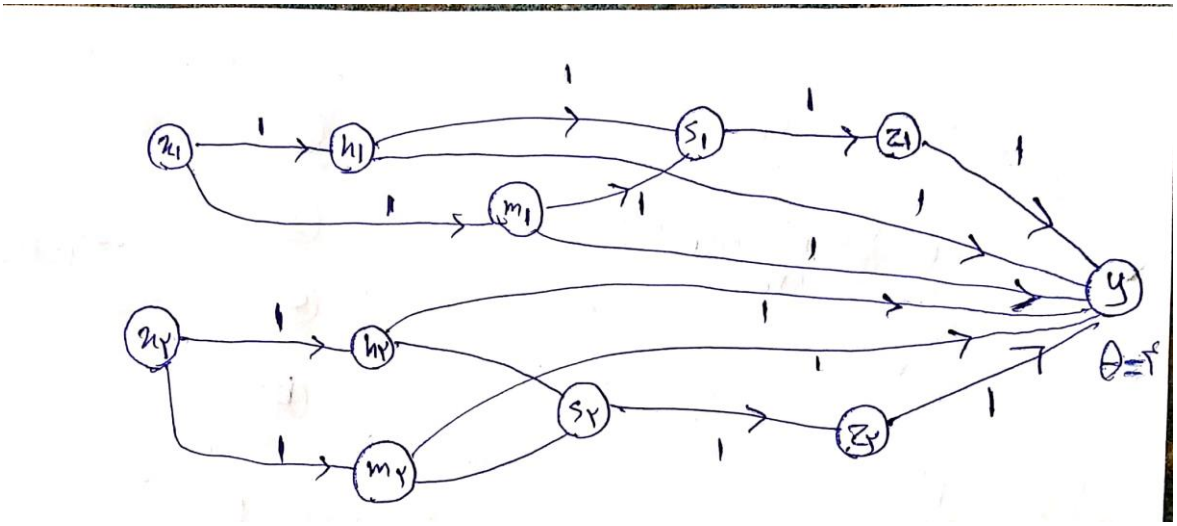


Figure 1 شبکه طراحی شده برای سوال

ب

برای توضیح منطق، یک سناریو را توضیح می‌دهیم. ما دو بمب را با فاصله دو step منفجر می‌کنیم. فرض کنیم ابتدا بمب اول و سپس بمب دوم منفجر می‌شود. در Table 1 در step های مختلف مقادیر نرون ها آمده است.

Table 1 جدول مقادیر نرون‌ها با انفجار بمب اول و سپس بمب دوم

step	x_1	x_2	h_1	h_2	m_1	m_2	s_1	s_2	z_1	z_2	y
0	1	0	0	0	0	0	0	0	0	0	0
1	0	0	1	0	1	0	0	0	0	0	0
2	0	1	0	0	0	0	2	0	0	0	0
3	0	0	0	1	0	1	0	0	2	0	0
4	0	0	0	0	0	0	0	2	0	0	1

در Table 1 منطق شبکه دیده می‌شود. نرون‌های h_i و m_i یک step بعد از تغییر مقدار x_i، مقدار آن را می‌گیرند. نرون z_i یک step بعد از تغییر مقدار s_i، مقدار آن را می‌گیرد. نرون s_i یک step بعد از تغییر مقدار نرون‌های h_i و m_i ، مقدار مجموع نرون‌های ورودی را می‌گیرد. نرون خروجی اگر مقدار نرون‌های h_1 و m_1 برابر یک شود و مقدار z_2 برابر دو شود، یک می‌شود. یا اگر مقدار نرون‌های h_2 و m_2 برابر یک شود و مقدار z_1 برابر دو شود، یک می‌شود. زمانی که مقدار نرون‌های h_i و m_i برابر یک است، مقدار s_i و z_i هر دو صفر است. مقادیر این نرون‌ها نشان دهنده‌ی انفجار قبل از انفجار آخر بمب است. چون هر بمب فقط یک بار منفجر می‌شود، به همین دلیل مقدار s_i و z_i هر دو صفر است. پس مقدار s_i و z_i در تعیین مقدار خروجی تاثیری ندارد. چون اولین لحظه تشخیص دکمه را باید بزنیم، پس خروجی به مقدار s و z متناظر بمب منفجر شده‌ی دوم مرتبط نیست. جدول مقادیر برای نرون‌های h، m، z و y در Table 2 و Table 3 آمده است.

Table 2 مقدار خروجی بر اساس نرون‌های مرتبط، زمانی که در ابتدا بمب اول منفجر شود

y	$m_1 = 1$	$h_1 = 1$	$z_2 = 2$
0	0	0	0
0	0	1	0
0	0	1	1
0	0	0	1
0	1	0	0
0	1	1	0
0	1	0	1
1	1	1	1

Table 3 مقدار خروجی بر اساس نرون‌های مرتبط، زمانی که در ابتدا بمب دوم منفجر شود

y	$m_2 = 1$	$h_2 = 1$	$z_1 = 2$
0	0	0	0
0	0	1	0
0	0	1	1
0	0	0	1
0	1	0	0
0	1	1	0
0	1	0	1

1	1	1	1

بر اساس جداول بالا، معادله منطقی شبکه یا معادله منطقی خروجی به صورت زیر است. یک متغیر منطقی به ازای هر نرون مرتبط تعیین می‌کنیم که همنام آن‌ها است. متغیرهای z اگر نرون‌های z متناظر آن‌ها ۲ باشد، یک می‌شوند وگرنه صفر هستند. متغیرهای h و m اگر نرون‌های متناظر آن‌ها ۱ باشد، یک می‌شوند وگرنه صفر هستند.

$$y = (z_1\ and\ m_2\ and\ h_2)\ or\ (z_2\ and\ m_1\ and\ h_1)$$

برای پیاده سازی این منطق، از آستانه ۴ در خروجی استفاده کردیم. اگر z_2 برابر ۲ و h_1 برابر ۱ و m_1 برابر ۱ شود، مجموع ورودی‌های نرون خروجی ۴ می‌شود و خروجی یک می‌شود. اگر هر حالت دیگری اتفاق بیفتد، مجموع ورودی‌های نرون خروجی کمتر از ۴ می‌شود و خروجی صفر است. به این شکل عبارت منطقی $(z_2\ and\ m_1\ and\ h_1)$ پیاده سازی شده است. یا اگر z_1 برابر ۲ و h_2 برابر ۱ و m_2 برابر ۱ شود، مجموع ورودی‌های نرون خروجی ۴ می‌شود و خروجی یک می‌شود. اگر هر حالت دیگری اتفاق بیفتد، مجموع ورودی‌های نرون خروجی کمتر از ۴ می‌شود و خروجی صفر است. به این شکل عبارت منطقی $(z_1\ and\ m_2\ and\ h_2)$ پیاده سازی شده است.

سوال ۲ – Perceptron

الف

داده‌ها در scatter plot رسم شده اند. داده‌های کلاس‌های مختلف با رنگ متفاوت به نمایش در آمده اند. Plot مورد نظر در Figure 2 آمده است.

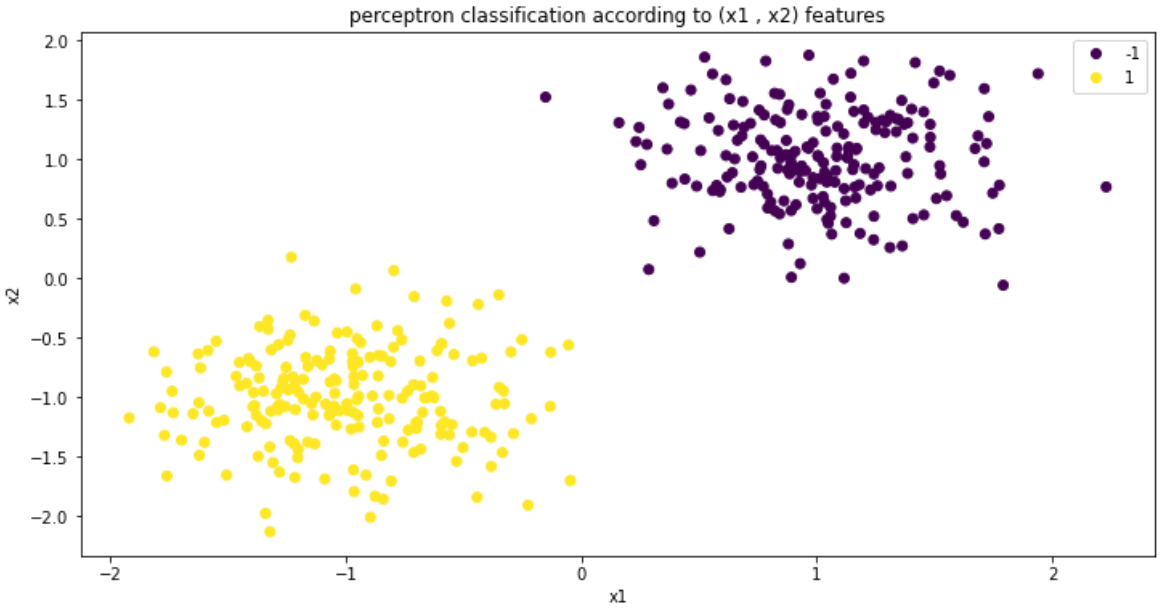


Figure 2 نمودار **scatter plot** داده‌ها

همان طور که دیده می‌شود، داده‌های هر دست حول مرکز متمرکز هستند و همچنین مرکز دسته‌ها فاصله قابل توجهی از هم دارند. توزیع دسته‌ها از هم جدا هستند. همان طور که دیده می‌شود، با یک خط به طور کامل می‌توان داده‌های دو دسته را از هم جدا کرد. پس انتظار داریم که به accuracy یک برسیم.

ب

در ابتدا وزن‌ها و بایاس را برابر صفر قرار می‌دهیم و مقدار α یا learning rate را برابر یک قرار می‌دهیم. به ازای هر داده آموزش، اگر پیش بینی ما با داده برابر نبود به بروز رسانی وزن‌ها و بایاس می‌پردازیم. این کار را آنقدر ادامه می‌دهیم تا به ازای تمام داده‌های آموزش، به درستی پیش بینی کنیم. در Perceptron ما دو خط موازی را برای جدا سازی دسته‌ها یاد می‌گیریم که می‌خواهیم margin مناسبی داشته باشیم. یعنی هر خط چسبیده به هر دسته باشد. این موضوع داشتن margin مناسب به جنرالیزیشن کمک می‌کند. می‌تواند باعث شود که برای داده‌های دیده نشده، پیش بینی بهتری داشته باشیم. که برای این موضوع

اندازه margin یا فاصله دو خط، مقدار θ ماثر است. آستانه بزرگتر باعث فاصله بیشتر بین خطوط می شود. شکاف در نظر گرفته شده بین خطوط باعث ایجاد یک حاشیه محافظه کارانه در جداسازی دو طبقه می شود. ما در این جا برای یادگیری از مقدار ۷ برای تتا استفاده می‌کنیم. در Figure 3 کد نحوه یادگیری و تابع فعال ساز وابسته به تتا آمده است.

```
def active(net, theta):
  if net > theta:
    return 1
  elif theta >= net >= (-1*theta):
    return 0
  elif net < (-1*theta):
    return -1
```

learn Perceptron

```
def learn(x, y, theta):
  w = np.zeros(x.shape[1])
  b = 0
  a = 1
  epoch_count = 0
  while True:
    print('epoch ', epoch_count)
    epoch_count += 1
    zero_samples = set()
    for i, s in enumerate(x.values):
      h = active(np.dot(s, w)+b, theta)
      error = h-y[i]
      if error != 0:
        w = w + a*y[i]*s
        b = b + a*y[i]
        zero_samples = set()
      else:
        zero_samples.add(i)
    if zero_samples == set(range(x.shape[0])):
      break
  return w, b

w, b = learn(X_train, y_train, 7)
```

Figure 3 نحوه یادگیری در **perceptron**

ج

در Figure 4 خطوط جداکننده که در بخش قبل توسط نرون آموخته شده است، در کنار scatter plot داده‌های تست آمده است.

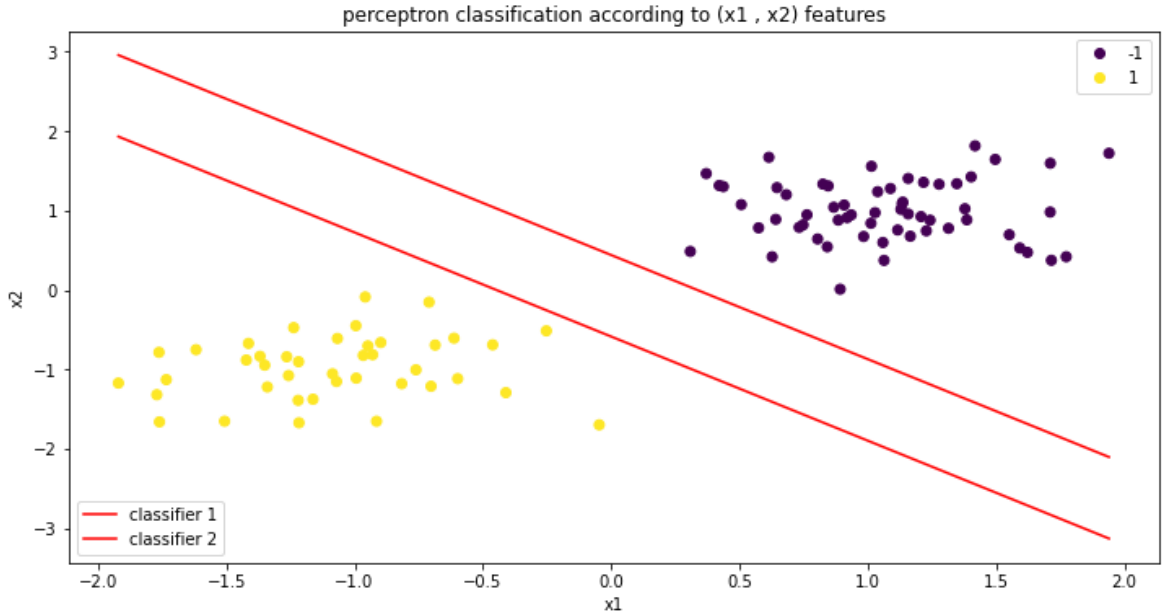


Figure 4 خطوط جداکننده که توسط **perceptron** آموخته شده است و **scatter plot** داده‌های تست

در زیر دقت و مقدار دیگر متریک‌های طبقه بندی آمده است.

	precision	recall	f1-score	support
-1	1.00	1.00	1.00	59
1	1.00	1.00	1.00	41
accuracy			1.00	100
macro avg	1.00	1.00	1.00	100
weighted avg	1.00	1.00	1.00	100

Figure 5 مقدار دقت و دیگر متریک‌ها در طبقه بندی با نرون **perceptron** با تتا ۷

همانطور که در Figure 4 دیده می‌شود، خطوط جدا کننده به داده‌های دسته‌های مختلف چسبیده اند و داده‌های دسته‌های مختلف کاملا توسط این خطوط از هم جدا شده اند. به همین دلیل مقدار accuracy و دیگر متریک‌های طبقه بندی برای ۱۰۰ درصد است. چون تمام داده‌ها درست دسته بندی شده اند. داده‌ها کاملا با خط از هم جدا پذیر اند. به همین دلیل نرون توانسته در تعداد ۱۵ تا epoch این خط جدا کننده را پیدا کند و دقت آن ۱۰۰ درصد شود.

د

در این جا تیتا را برای آموزش نرون برابر صفر قرار می‌دهیم. دوباره نرون را آموزش می‌دهیم تا خطوط جدا کننده را بیاموزد. مانند قسمت قبل خطوط جداکننده در کنار داده‌های تست در Figure 6 آمده است.

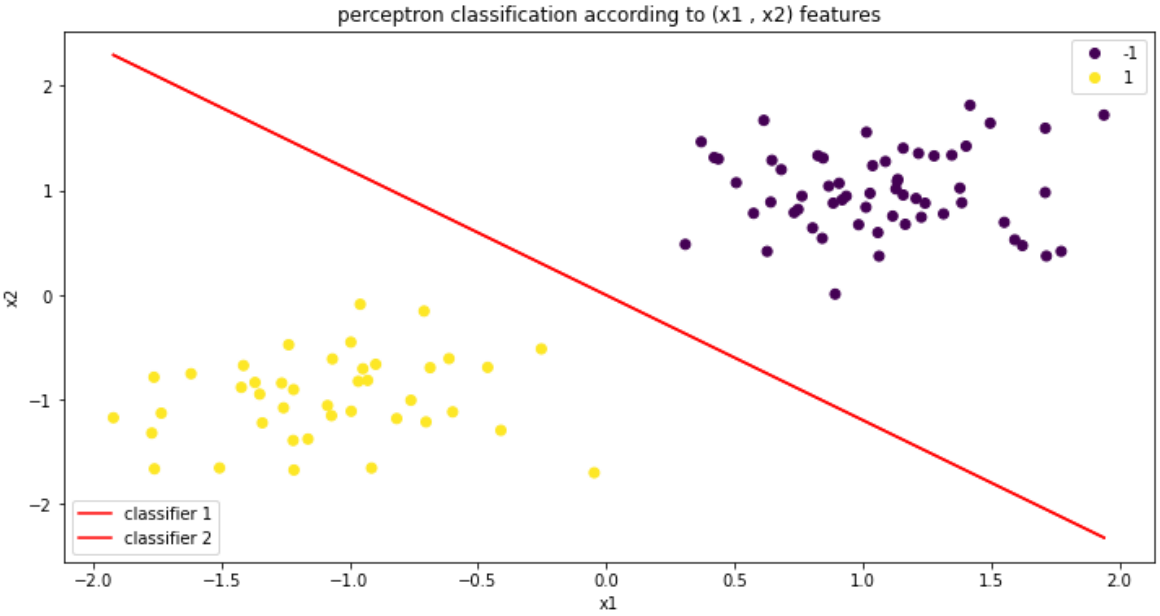


Figure 6 خطوط جدا کننده به همراه داده‌های تست با آستانه‌ی صفر

همانطور که در Figure 6 دیده می‌شود، خطوط بر روی هم قرار دارند که دلیل آن تتای صفر است. خطوط کاملا داده‌های دو دسته را از هم جدا کرده اند و به همین دلیل دقت و دیگر متریک‌های طبقه بندی همگی برابر ۱۰۰ درصد هستند. مقدار متریک‌های طبقه بندی در Figure 7 آمده است. نرون دو باره توانسته خط جدا کننده داده‌های جداپذیر خطی را بیابد. نکته قابل توجه شیب و عرض از مبدا متفاوت خطوط در این حالت با حالت قبل است.

	precision	recall	f1-score	support
-1	1.00	1.00	1.00	59
1	1.00	1.00	1.00	41
accuracy			1.00	100
macro avg	1.00	1.00	1.00	100
weighted avg	1.00	1.00	1.00	100

Figure 7 دقت و دیگر متریک‌های دسته بندی برای نرون **perceptron** با تتا صفر

همانطور که در دو شکل Figure 4 و Figure 6 دیده می‌شود، در حالت تتا یا آستانه برابر صفر شیب بیشتر است و عرض از مبدا خطوط نیز متفاوت است. خط در حالت تتا صفر موازی خطوط تتا ۷ نیست. این متفاوت شدن خطوط به دلیل تغییر تتا است. تغییر آستانه تاثیری بر روی نتیجه طبقه بندی ندارد. درصورتی که آستانه را روی مقدار دیگری تنظیم کنیم، وزنها فقط خود را تنظیم می کنند تا معادله را تنظیم کنند، یعنی وزنها (با احتساب بایاس) اثرات آستانه را جذب می کنند. در حالت کلی ما به دنبال a و b و c در معادله زیر هستیم.

$$a.x_1 + b.x_1 + c = \theta$$

اگر تتا را به طرف چپ ببریم معادله جداکننده به شکل زیر می‌شود که $c - \theta$ را c' می‌نامیم. در این جا ما به دنبال یادگیری a و b و c' هستیم.

$$a.x_1 + b.x_1 + c - \theta = 0$$
$$a.x_1 + b.x_1 + c' = 0$$

به ازای هر مقدار تتا، نرون وزن‌ها و بایاس (a, b, c) را به گونه‌ای تغییر می‌دهد تا معادله خط جدا کننده مانند زمانی که تتا صفر است، بدست بیاید. مقدار تتا فقط باعث می‌شود که مقدار اولیه c' فرق کند. نرون از این مقدار اولیه جدید شروع می‌کند و ادامه می‌دهد تا به (a, b, c') مناسب برسد که با حالت‌های دیگر تتا نیز برابر هستند. پس به ازای هر تتا باز خطوط مانند گذشته قدرت جداسازی دسته‌های مختلف را دارند و در نتیجه طبقه بندی تغییری ایجاد نمی‌شود.

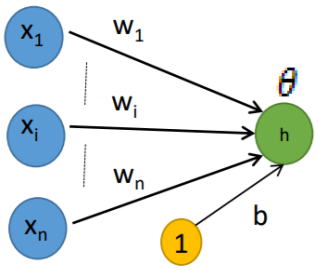
سوال 3 – Adaline

الف

معماری شبکه های Adaline و Perceptron مشابه است. آنها از قوانین مختلف یادگیری برای وزن‌ها و بایاس استفاده می کنند. Perceptron از برچسب های کلاس برای یادگیری ضرایب مدل استفاده می‌کند. Adaline از مقادیر پیش بینی شده پیوسته (از ورودی خالص net) برای یادگیری ضرایب مدل استفاده می‌کند، که قدرتمندتر است زیرا از این طریق به ما می گوید که چقدر درست یا غلط عمل کرده‌ایم. در Adaline به ازای هر داده‌ی آموزش، ما به بروز رسانی وزن‌ها و بایاس می‌پردازیم. این درحالی هست که در perceptron اگر پیش بینی برای یک داده با لیبل واقعی یکی نباشد، ما به بروز رسانی می‌پردازیم. در perceptron ما تابع فعال ساز برحسب تتا داریم و دو خط را برای جدا سازی پیدا می‌کنیم. . شکاف در نظر گرفته شده بین خطوط باعث ایجاد یک حاشیه محافظه کارانه در جداسازی دو طبقه می شود. در Adaline ما یک خط برای جداسازی پیدا می‌کنیم و تابع فعالساز به پارامتری مثل تتا وابسته نیست.

ب

ساختار شبکه به صورت Figure 8 است. در اینجا ما به دنبال یادگیری وزن‌ها و بایاس به شکلی هستیم که تابع هزینه را کمینه کنیم.



(1)

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

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

Figure 8 ساختار شبکه **adaline**

تابع هرینه برای داده آموزش در Figure 9 آمده است که ما می‌خواهیم وزن‌ها و بایاس را به گونه‌ای بیابیم که این تابع را کمینه کنیم. از این تابع در الگوریتم یادگیری برای پایان نیز استفاده می‌کنیم.

$$J_p(\boldsymbol{w}, b) = 0.5(t(p) - net(x(p), w, b))^2$$

Figure 9 تابع هزینه **adaline** برای یک داده آموزش

برای پیاده سازی یادگیری این نرون، گام‌هایی را انجام می‌دهیم که در ادامه آمده است. ابتدا وزن‌ها را به صورت راندم مقدار می‌دهیم. بایاس را نیز همین گونه مقدار می‌دهیم که البته ما بایاس را در ابتدا صفر قرار دادیم. مقدار α یا learning rate را برابر 0.002 که مقدار مثبت کوچکی است قرار دادیم. سپس در هر epoch تمام داده‌های آموزش را می‌بینیم. به ازای هر داده وزن‌ها و بایاس را به شکل Figure 10 به روز می‌کنیم.

```
net = np.dot(s,w)+b
w = w + a*(y[i]-net)*s
b = b + a*(y[i]-net)
```

Figure 10 نحوه به روز کردن وزن‌ها و بایاس در یادگیری نرون **adaline**

در انتهای هر epoch میزان تغییر مقدار cost را بررسی می‌کنیم. تا زمانی که تغییر هزینه بیش از 0.0001 است به انجام epoch های بیشتر می‌پردازیم. در ادامه خطوط جداکننده برای شکل‌های ۱ و ۲ به ترتیب در Figure 11 و Figure 12 آمده است.

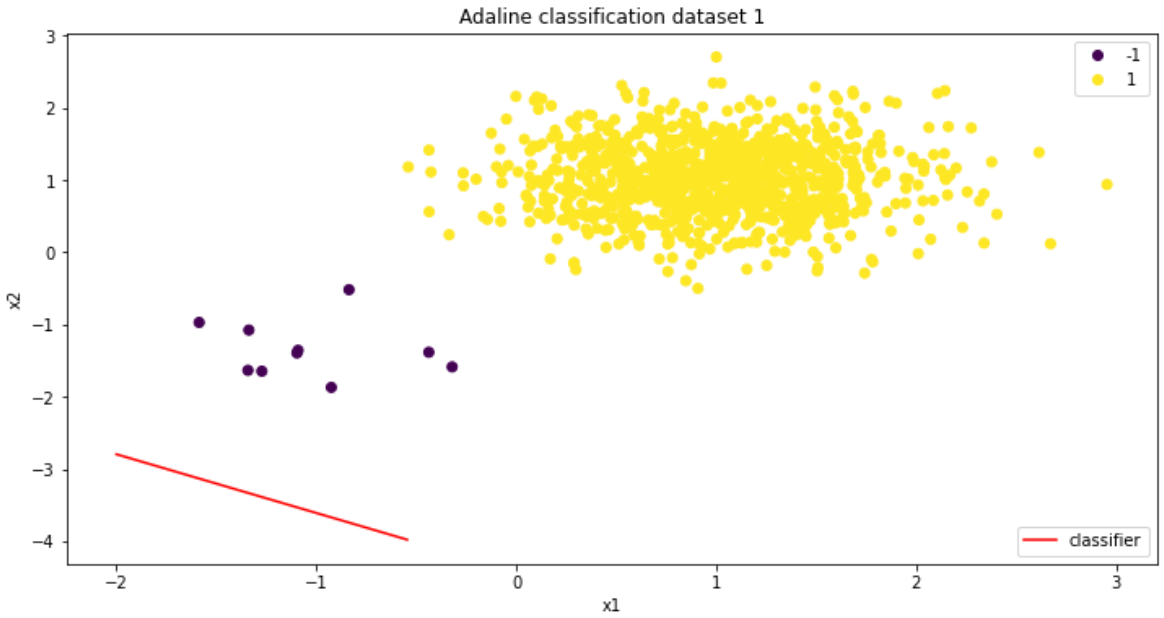


Figure 11 خط جدا کننده برای شکل ۱ صورت سوال

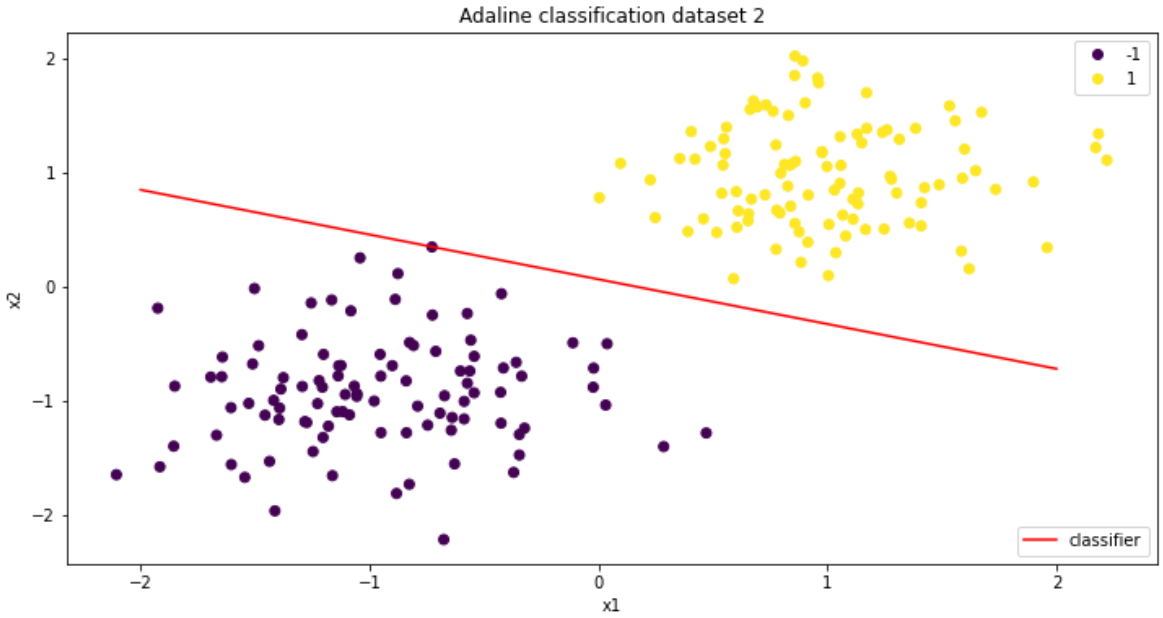


Figure 12 خط جداکننده برای شکل ۲ صورت سوال

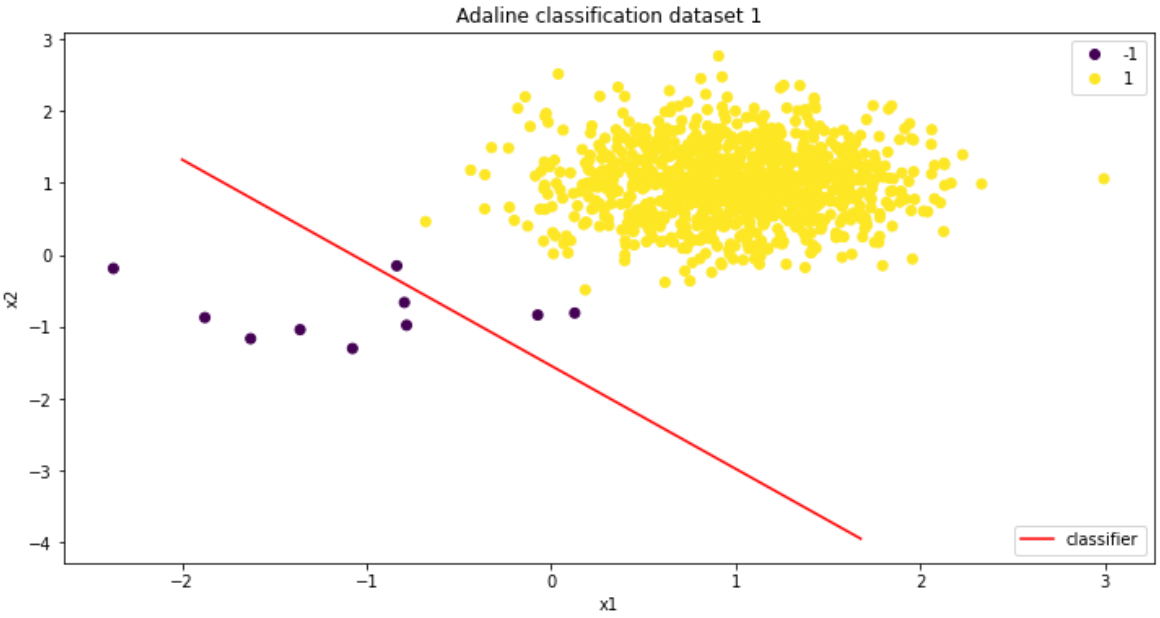


Figure 13 خط جداکننده بهتر برای شکل ۱ سوال

در Figure 11 خط جداکننده کاملا اشتباه کلاس ها را طبقه بندی می‌کند و توانایی برای طبقه بندی ندارد و دقت طبقه بندی آن صفر است. چون تمام داده‌ها در یک طرف هستند. در Figure 13 خروجی دیگری برای شکل ۱ است که باز هم خط جداکننده دقت بالایی ندارد. باز خط در طبقه بندی اشتباه دارد و خط تقریبا از وسط کلاس با تعداد کمتر گذشته است. این اتفاق به خاطر توزیع و تعداد متفاوت داده‌های دو دسته است. از آنجا که خطا به عنوان تفاوت هدف و net تعریف شده است (نه تفاوت هدف و خروجی)،

net (نه لزوماً خروجی شبکه) نزدیک به هدف می شود. اگر توزیع الگوها در دو کلاس از نظر شکل و جمعیت نزدیک به هم نباشد، روش یادگیری بیان شده (روش دلتا) تضمینی برای پیدا کردن خط جدا کننده‌ی واقعی نمی‌دهد. به همین دلیل است که خط پیدا شده اصلا دو کلاس را جدا نمی‌کند. در Figure 12 توزیع و تعداد داده‌های دو دسته یکسان است. به همین دلیل الگوریتم توانسته خط جدا کننده‌ی واقعی دو دسته را پیدا کند.

ج

خیر. همان طور که در قسمت قبل تلاش کردیم ولی نتوانستیم خط جدا کننده‌ی واقعی را بیابیم. از آنجا که خطا به عنوان تفاوت هدف و net تعریف شده است (نه تفاوت هدف و خروجی)، net (نه لزوماً خروجی شبکه) نزدیک به هدف می شود. اگر توزیع الگوها در دو کلاس از نظر شکل و جمعیت نزدیک به هم نباشد، روش یادگیری بیان شده (روش دلتا) تضمینی برای پیدا کردن خط جدا کننده‌ی واقعی نمی‌دهد. برای حل این مشکل از تابع فعالساز دیگری به جز sign استفاده می‌کنیم که مشتق پذیر باشد. Sign نرم مانند تابع $tanh$. و در این حالت ارور را تفاوت فعال شده‌ی net با tanh و لیبل واقعی تعریف می‌کنیم. سعی می‌کنیم خروجی را به لیبل ها نزدیک کنیم. روش یادگیری نرون مانند قبل است و فقط نحوه‌ی بروز رسانی وزن‌ها و بایاس به صورت Figure 14 تغییر می‌کند که g همان γ است.

```
net = np.dot(s,w)+b
h = np.tanh(g*net)
w = w + a*(1-(h**2))*g*(y[i]-h)*s
b = b + a*(1-(h**2))*g*(y[i]-h)
```

Figure 14 نحوه به روز رسانی وزن‌ها و بایاس با تابع فعال ساز **tanh**

تابع هزینه برای یک داده به شکل Figure 15 است. اگر γ یا g خیلی بزرگ انتخاب شود، tanh نزدیک به تابع sign خواهد بود. اکنون، با کاهش هزینه، خروجی شبکه به مقدار هدف همگرا خواهد شد (همانطور که در perceptron انجام شد).

$$J_p(\boldsymbol{w}, b) = 0.5\big(t(p) - \tanh(\gamma net(x(p), w, b)\big)^2$$

Figure 15 تابع هزینه برای تابع فعالساز **tanh**

Tanh تابع نرم است و می توان قوانین به روزرسانی جدید را با gradient descent محاسبه کرد که مانند Figure 14 می‌شود. از $\gamma = 100$ برای حل استفاده شده است. خطوط جداکننده شکل ۱ و ۲ سوال به ترتیب در Figure 16 و Figure 17 طبق این قوانین به روز رسانی جدید آمده است.

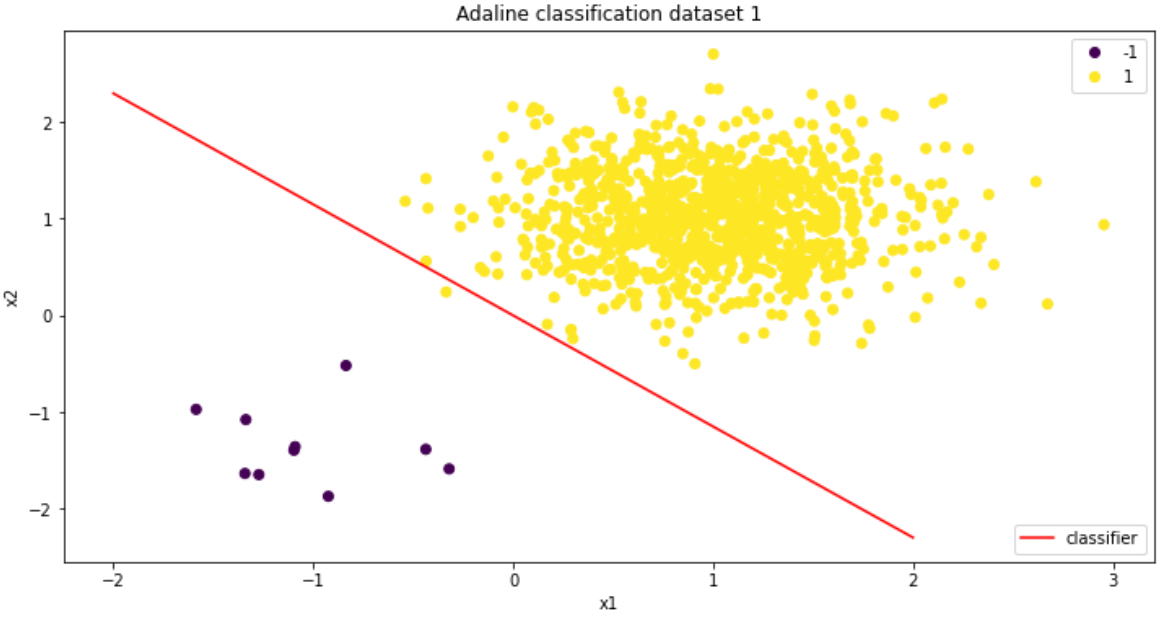


Figure 16 خط جدا کننده شکل ۱ سوال با تابع فعالساز **tanh**

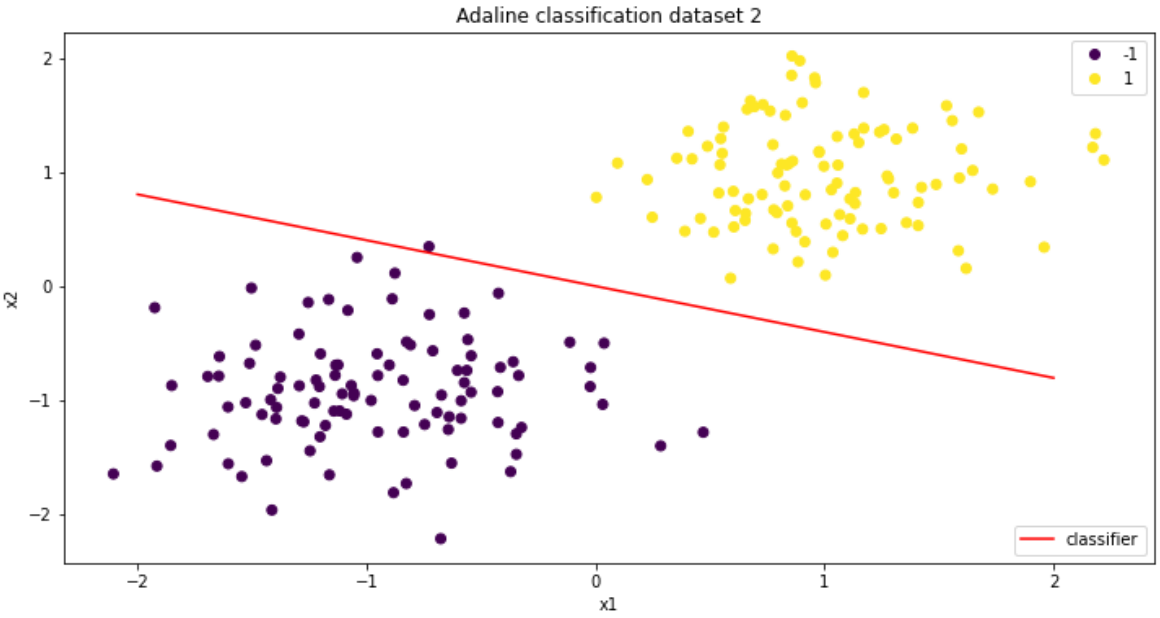


Figure 17 خط جدا کننده شکل ۲ سوال با تابع فعالساز **tanh**

همان طور که دیده می‌شود، پاسخ Figure 16 از Figure 11 و Figure 13 برای شکل یک سوال robust تر است. همان طور که دیده می‌شود خط بین دو دسته قرار دارد و دو دسته را با دقت بالایی جدا کرده است. در حالت تعداد برابر دو دسته نیز که در Figure 17 آمده است، عملکرد مانند گذشته است.

د

مقداری که وزن ها در حین آموزش به روز می شوند به عنوان اندازه گام یا میزان یادگیری یا learning rate ذکر می شود. میزان یادگیری کنترل می کند که چگونه مدل سریعاً با مشکل سازگار شود. نرخ یادگیری کوچکتر با توجه به تغییرات کمتری که در هر به روزرسانی در اوزان ایجاد می شود، به دوره های آموزشی بیشتری احتیاج دارد، در حالی که میزان یادگیری بیشتر منجر به تغییرات سریع می شود و به دوره های آموزشی کمتری نیاز دارد. نرخ یادگیری بیش از حد زیاد می تواند باعث شود مدل به سرعت به یک راه حل غیربهینه همگرا شود، در حالی که یک میزان یادگیری بسیار کوچک می تواند باعث گیر افتادن روند شود. اگر نرخ یادگیری بیش از حد زیاد باشد، ممکن است اطراف یک مینیمم پرش داشته باشیم و به مقدار مینیمم همگرا نشویم. یک میزان یادگیری خیلی کم یا باعث همگرایی خیلی طولانی می شود یا باعث می‌شود در حداقل محلی نامطلوب گیر کند.

سوال 4 – Madaline

الف

برای یادگیری madaline با تعداد متفاوتی نرون از یک الگوریتم یادگیری جامع به ازای هر تعداد نرون استفاده می‌کنیم. در این الگوریتم از روش MRI برای حل استفاده می‌کنیم. در الگوریتم MRI، وزن یال‌ها از نرون های لایه پنهان به خروجی ثابت هستند. نورون خروجی به عنوان یک منطق OR تنظیم می شود و فقط وزن ها و بایاس‌های نورون های پنهان آموزش داده می شوند. هر نرون پنهان یک نرون Adaline است که معادل یک خط یا یک ضلع چندضلعی convex ما است. ساختار نمونه این شبکه در Figure 18 آمده است.

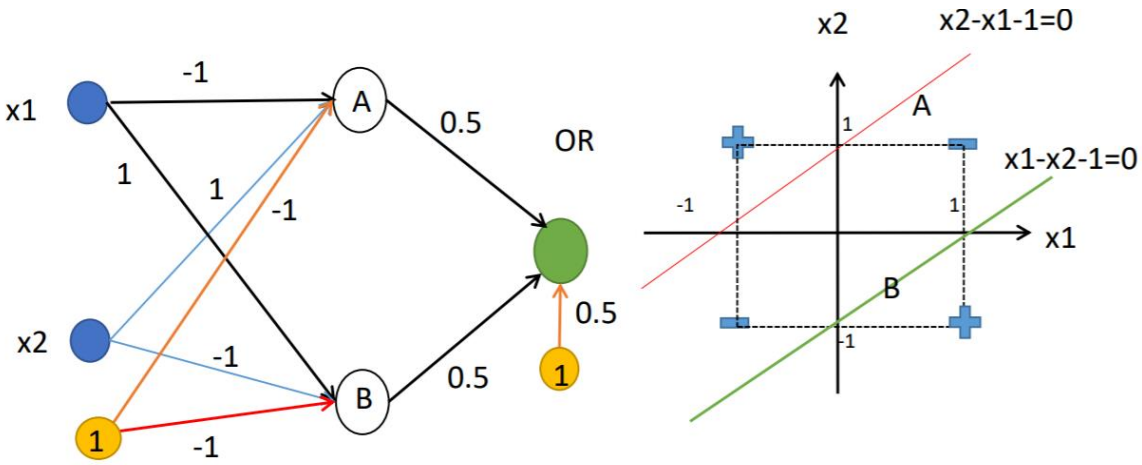


Figure 18 ساختار نمونه‌ی شبکه **madaline** با منطق **or** در خروجی

تعداد نرون‌های پنهان را n که متغیر است در نظر می‌گیریم. چون می‌خواهیم خروجی منطق or را پیاده سازی کند، وزن هر یال از هر نرون پنهان به خروجی را $\frac{2}{n}$ در نظر می‌گیریم . مقدار بایاس برای نرون خروجی را با توجه به این که حداکثر ۸ ضلع داریم، برابر 1.8 قرار می‌دهمیم. خروجی هر نرون پنهان برابر ۱ یا 1- است. منطق or مورد نظر ما در خروجی این است که فقط زمانی که تمام نرون‌ها خروجی 1- داشته باشند، خروجی کلی 1- باشد. در غیر این صورت خروجی یک باشد. با توجه به این تعریف، لیبل 1- باید برای داده‌هایی باشد که قرار است داخل چندضلعی قرار داشته باشند (داده‌های آبی در صورت سوال که در دیتاست لیبل صفر دارند). چون این داده‌ها، داده‌هایی هستند که تمام خطوط اضلاع باید هم زمان بگویند که داخل چند ضلعی هستند. به همین دلیل داخل چند ضلعی را ناحیه‌ی منفی تعریف می‌کنیم. ما حداکثر ۸ ضلع داریم. پس در صورتی که ۷ نرون جواب منفی دهند، مجموع وزن‌دار ورودی‌های نرون خروجی از نرون‌های پنهان برابر $\frac{-2\times(n-2)}{n} = -1.5$ می‌شود. مقدار بایاس باید عددی بزرگتر از $\frac{2\times(n-2)}{n} = 1.5$

انتخاب شود تا حاصل net در این شرایط مثبت شود. همچنین مقدار بایاس باید از ۲ کمتر باشد تا در صورتی که تمام ۸ نرون یا نرون‌ها خروجی منفی دادند، net منفی شود. پیاده سازی مقدار دهی اولیه در Figure 19 آمده است.

```
def learn(x, y, n=4, a=0.005, tol=3e-8, w=[]):
  v = [2/float(n) for i in range(n)]
  bo = 1.8
  ws = []
  if w == []:
    ws = [np.array([(2*np.random.rand()-1),(2*np.random.rand()-1)]) for j in range(n)]
  else:
    ws = w

  bs = [np.random.rand() for i in range(n)]
```

Figure 19 پیاده سازی مقدار دهی اولیه برای **madaline**

برای یادگیری وزن‌ها و بایاس‌های نرون‌های لایه پنهان از MRI و انجام epoch استفاده می‌کنیم. در هر epoch تمام داده‌های آموزش را می‌بینیم. به ازای هر داده، مقدار net و خروجی نرون‌های پنهان و خروجی کل را تعیین می‌کنیم. اگر خروجی کل با لیبل واقعی یکسان نبود، تعیین می‌کنیم کدام نرون‌های پنهان به بروزرسانی بپردازند.

اگر لیبل واقعی منفی یا داخل چند ضلعی باشد، در این صورت تعدادی از نرون‌های پنهان خروجی ۱ داده اند که اشتباه بوده است. تمام نرون‌ها باید خروجی منفی می‌دانند. در این جا تمام نرون‌های با خروجی یک را به روز می‌کنیم. هر نرون پنهان یک نرون Adaline با تابع فعالساز sign است. برای بروز کردن این نرون‌ها از قوانین بروزرسانی نرون‌های Adaline با تابع فعالساز sign استفاده می‌کنیم.

اگر لیبل واقعی یک یا خارج چند ضلعی باشد و خروجی نهایی منفی یا داخل باشد، تمام نرون‌های پنهان به اشتباه خروجی منفی داده اند. کافی بود یک نرون خروجی مثبت می‌داد تا پیش بینی درست می‌شد. پس در این جا یک نرون پنهان را باید برای بروز کردن انتخاب کنیم. ما نرونی که net آن از همه به صفر نزدیکتر است را انتخاب می کنیم و به روش به روز کردن نرون Adaline به روز می‌کنیم. این قدر به انجام epoch ادامه می‌دهیم تا تغییر وزن‌ها و بایاس‌های نرون‌های پنهان از یک آستانه مشخص کمتر شود. کد یادگیری وزن‌ها و بایاس‌های نرون‌های لایه پنهان در Figure 20 آمده است.

```
for i, s in enumerate(x):
  nets = [np.dot(s,ws[i])+bs[i] for i in range(n)]
  zs = [active(nets[i]) for i in range(n)]
  yout = active(np.dot(np.array(zs),np.array(v))+bo)
  if yout == -1:
    yout = 0
  if yout != y[i]:
    if y[i] == 0:
      for j in range(n):
        if zs[j] == 1:
          t = -1 if y[i] == 0 else 1
          ws[j] = ws[j] + a*(t-nets[j])*s
          bs[j] = bs[j] + a*(t-nets[j])
    else:
      j = np.argmin(np.array(nets)**2)
      t = -1 if y[i] == 0 else 1
      ws[j] = ws[j] + a*(t-nets[j])*s
      bs[j] = bs[j] + a*(t-nets[j])
```

Figure 20 نحوه یادگیری وزن‌ها و بایاس‌های نرون‌های لایه پنهان در **madaline**

ب و ج

در Figure 21 نمودار خطوط جدا کننده با ۴ نرون در کنار مجموعه داده‌ها آمده است. همان طور که دیده می‌شود ۴ خط یک ۴ ضلعی convex ایجاد کرده اند. در title به اشتباه perceptron نوشته شده است.

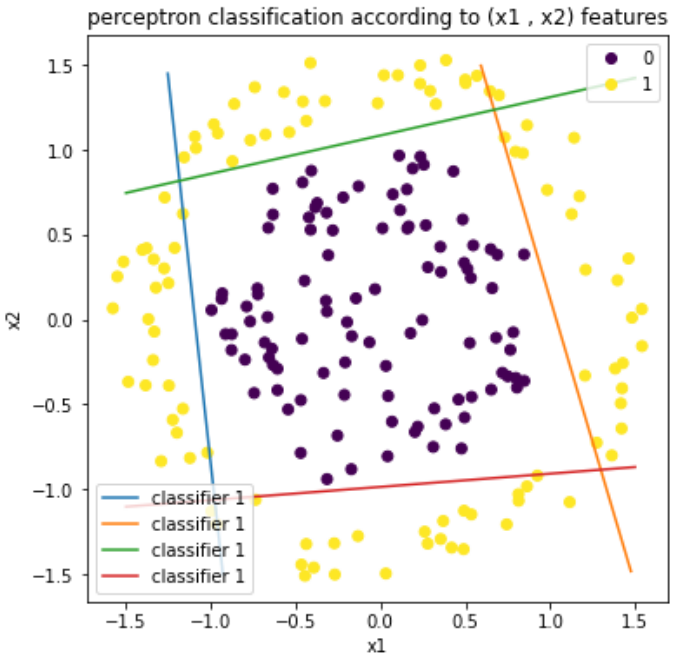


Figure 21 نمودار خطوط جدا کننده با ۴ نرون با **madaline**

همان طور که دیده می شود، خطوط تمام داده‌های داخل را به درستی از داده‌های بیرون جدا کرده‌اند. دسته بندی داده‌ها به داخل و بیرون کاملا درست است. به همین دلیل دفت و دیگر متریک‌های دسته بندی ۱۰۰ درصد است. در Figure 22 مقدار دقت و سایر متریک‌های طبقه بندی آمده است. این نتیجه بعد از انجام ۹۹ تا epoch بدست آمده است.

	precision	recall	f1-score	support
0.0	1.00	1.00	1.00	100
1.0	1.00	1.00	1.00	100
accuracy			1.00	200
macro avg	1.00	1.00	1.00	200
weighted avg	1.00	1.00	1.00	200

Figure 22 مقدار دقت و سایر متریک‌های طبقه بندی برای **madaline** با ۴ نرون

در Figure 23 نمودار خطوط جدا کننده با ۶ نرون در کنار مجموعه داده‌ها آمده است. همان طور که دیده می‌شود ۶ خط یک ۶ ضلعی convex ایجاد کرده اند. در title به اشتباه perceptron نوشته شده است.

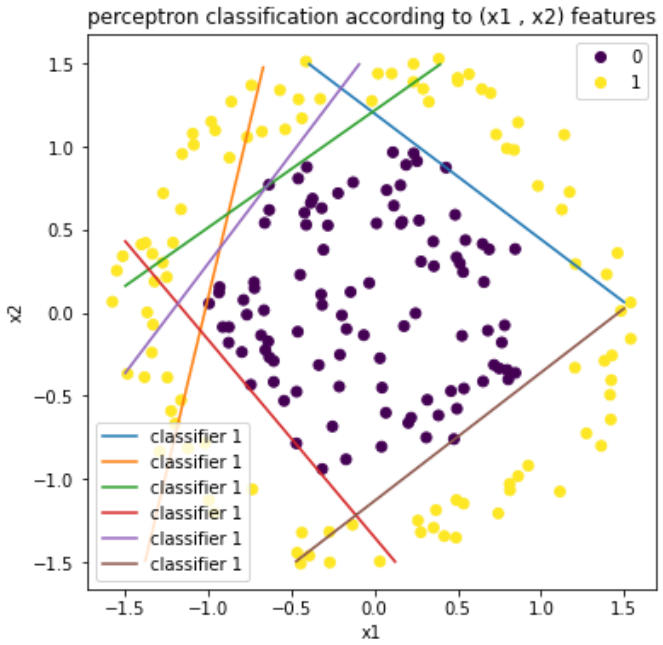


Figure 23 نمودار خطوط جدا کننده با ۶ نرون با **madaline**

همان طور که دیده می شود، خطوط تمام داده‌های داخل را به درستی از داده‌های بیرون جدا کرده‌اند. دسته بندی داده‌ها به داخل و بیرون کاملا درست است. به همین دلیل دفت و دیگر متریک‌های دسته بندی ۱۰۰ درصد است. در Figure 24 مقدار دقت و سایر متریک‌های طبقه بندی آمده است. این نتیجه بعد از انجام ۳۶ تا epoch بدست آمده است.

	precision	recall	f1-score	support
0.0	1.00	1.00	1.00	100
1.0	1.00	1.00	1.00	100
accuracy			1.00	200
macro avg	1.00	1.00	1.00	200
weighted avg	1.00	1.00	1.00	200

Figure 24 مقدار متریک و سایر متریک‌های طبقه بندی برای **madaline** با ۶ نرون

در Figure 25 نمودار خطوط جدا کننده با ۸ نرون در کنار مجموعه داده‌ها آمده است. همان طور که دیده می‌شود ۸ خط یک ۸ ضلعی convex ایجاد کرده اند. در title به اشتباه perceptron نوشته شده است.

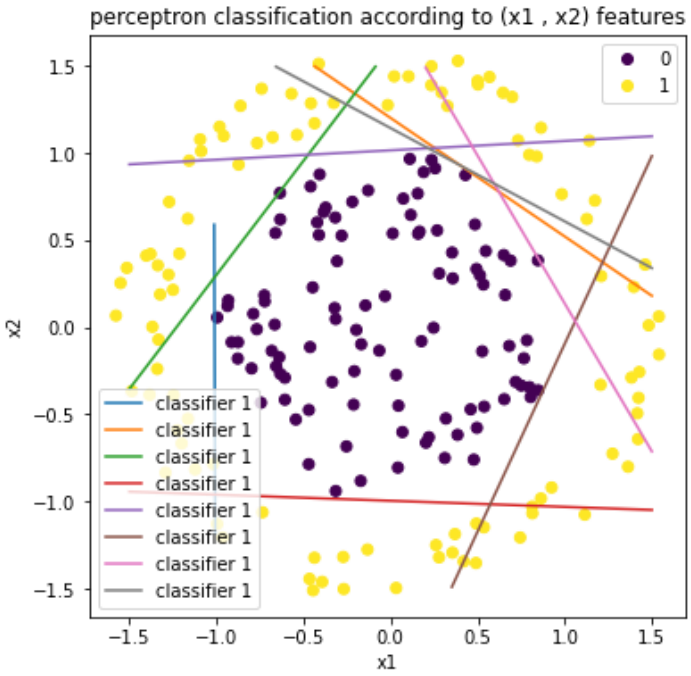


Figure 25 نمودار خطوط جدا کننده با ۸ نرون با **madaline**

همان طور که دیده می شود، خطوط تمام داده‌های داخل را به درستی از داده‌های بیرون جدا کرده‌اند. دسته بندی داده‌ها به داخل و بیرون کاملا درست است. به همین دلیل دفت و دیگر متریک‌های دسته بندی ۱۰۰ درصد است. در Figure 26 مقدار دقت و سایر متریک‌های طبقه بندی آمده است. این نتیجه بعد از انجام ۹ تا epoch بدست آمده است.

	precision	recall	f1-score	support
0.0	1.00	1.00	1.00	100
1.0	1.00	1.00	1.00	100
accuracy			1.00	200
macro avg	1.00	1.00	1.00	200
weighted avg	1.00	1.00	1.00	200

Figure 26 مقدار دقت و دیگر متریک‌های طبقه بندی برای **madaline** با ۸ نرون

همان طور که دیده می‌شود، هم با ۴، هم با ۶ و هم با ۸ نرون به دقت ۱۰۰ درصد رسیدیم. همانطور که در Figure 21 دیده می‌شود، داده‌های دو دسته توسط یک ۴ ضلعی convex کاملا جدا پذیر هستند. به همین دلیل madaline توانسته یک چهار ضلعی برای این کار بیابد. از آن جا که داده‌ها با چهار ضلع جدا پذیر هستند، پس می‌توان با ۶ و ۸ ضلعی‌های convex نیز دسته‌ها را جدا کرد. برای این کار کافی است که دو ضلع از چهار ضلعی جدا کننده را به ۴ ضلع تبدیل کنیم. یعنی هر ضلع را به دو ضلع کوچکتر که فاصله کمی از ضلع اصلی دارند تبدیل کنیم. یا می‌توانیم یک ضلع از چهار ضلعی جدا کننده را به ۳ ضلع تبدیل کنیم. در Figure 22 می‌بینیم که ۶ ضلعی پیدا شده در اصل یک چهار ضلعی بوده که یک ضلع آن به سه ضلع کوچکتر تبدیل شده است. این اضلاع جدید فاصله کمی از ضلع اصلی دارند که تقریبا بر آن منطبق هستند. چون اضلاع جدید فاصله‌ی کمی از ضلع اصلی دارند، به همین دلیل ناحیه داخل چند ضلعی تقریبا با چهار ضلعی اولیه برابر است. چون ناحیه داخل چهار ضلعی اولیه داده‌ها را درست دسته بندی می‌کرد، به همین دلیل ۶ ضلعی حاصل نیز دقت ۱۰۰ درصد دارد. به همین دلیل می‌توان گفت که داده‌ها با ۶ ضلعی هم جدا پذیر هستند. در Figure 25 می‌بینیم که ۸ ضلعی پیدا شده در اصل یک ۶ ضلعی است که دو ضلع آن به ۴ ضلع کوچکتر تبدیل شده است. این اضلاع جدید فاصله کمی از ضلع اصلی دارند که تقریبا بر آن منطبق هستند. چون اضلاع جدید فاصله‌ی کمی از ضلع اصلی دارند، به

همین دلیل ناحیه داخل چند ضلعی تقریبا با ۶ ضلعی اولیه برابر است. چون ناحیه داخل ۶ ضلعی اولیه داده‌ها را درست دسته بندی می‌کرد، به همین دلیل ۸ ضلعی حاصل نیز دقت ۱۰۰ درصد دارد. به همین دلیل می‌توان گفت که داده‌ها با ۸ ضلعی هم جدا پذیر هستند.

تعداد epoch ها و learning rate برای تعداد مختلف نرون برای madaline در Table 4 آمده است. اندازه‌ی learning rate در مشخص کردن تعداد epoch ماثر است و به همین دلیل در جدول آمده است.

Table 4 تعداد **epoch** ها و **learning rate** برای تعداد مختلف نرون برای **madaline**

n	Learning rate	epoch
4	0.05	99
6	0.002	36
8	0.01	9

همان طور در Table 4 دیده می‌شود، با افزایش تعداد نرون‌ها تعداد epoch کم می‌شود. برای تعداد نرون ۴ و ۸ مقدار learning rate از یک کلاس یعنی 0.01 است. همچنین learning rate برای ۴ نرون بیشتر است. از آن جا که افزایش learning rate می‌تواند باعث کاهش تعداد epoch شود، پس می‌توان نتیجه گرفت که کاهش تعداد epoch بین ۴ و ۸ نرون به خاطر تعداد نرون بوده است. برای تعداد نرون ۴ و ۶ مقدار learning rate کاملا متفاوت است. همچنین learning rate برای ۴ نرون بسیار بیشتر است. از آن جا که افزایش learning rate می‌تواند باعث کاهش تعداد epoch شود، پس می‌توان نتیجه گرفت که کاهش تعداد epoch بین ۴ و ۶ نرون به خاطر تعداد نرون بوده است. پس می‌توان گفت افزایش تعداد نرون باعث کاهش تعداد epoch نسبت به تعداد ۴ نرون شده است.

این موضوع می‌تواند به دلیل این موضوع باشد که با افزایش نرون یا خطوط، به طور متوسط تعداد داده‌هایی که هر خط می‌خواهد از دو دسته جدا کند کمتر می‌شود. هم چنین توزیع داده‌های دو دسته اطراف خطوط ساده تر می‌شود. هر چه تعداد کمتری داده‌ی دو دسته در کنار هم را در نظر بگیریم، توزیع داده‌ها ساده تر و تعداد داده‌های دسته‌ها کمتر می‌شود. این موضوع باعث می‌شود که در تعداد epoch کمتر بتوان خط جدا کننده برای این تعداد داده‌ی کمتر پیدا کرد و در کل تعداد epoch برای جداسازی کل کمتر می‌شود. بین ۶ و ۸ نرون باز کاهش تعداد epoch دیده می‌شود که بخشی از آن می‌تواند به خاطر افزایش تعداد نرون‌ها و بخش دیگر می‌تواند به خاطر افزایش learning rate باشد.