به نام خدا

دانشگاه تهران

دانشکده مهندسی برق و کامپیوتر

درس شبکه‌های عصبی و یادگیری عمیق

تمرین سری یک

نام و نام خانوادگی	حمید رضا کاشانی
شماره دانشجویی	810100441
تاریخ ارسال گزارش	1400/12/23

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

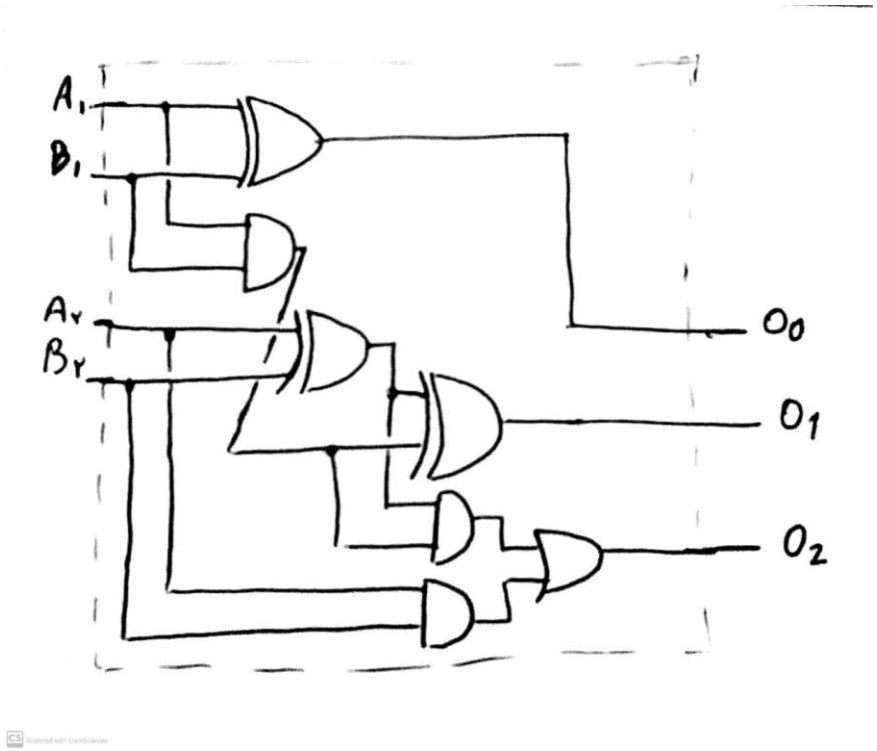
سوال ۱ – Mcculloch Pitts

به کمک نورون Mcculloch Pitts توسعه یافته یک Full Adder بسازید، که دو ورودی دو بیتی را گرفته و آن ها را جمع کند. برای این کار به دو ورودی دو بیتی)در واقع چهار نورون برای همه ورودی ها(نیاز داریم. همچنین سه بیت خروجی (سه نورون) مورد نیاز است. توجه شود که تمام تمامی نورون های ورودی خروجی باینری هستند.(صفر و یک)

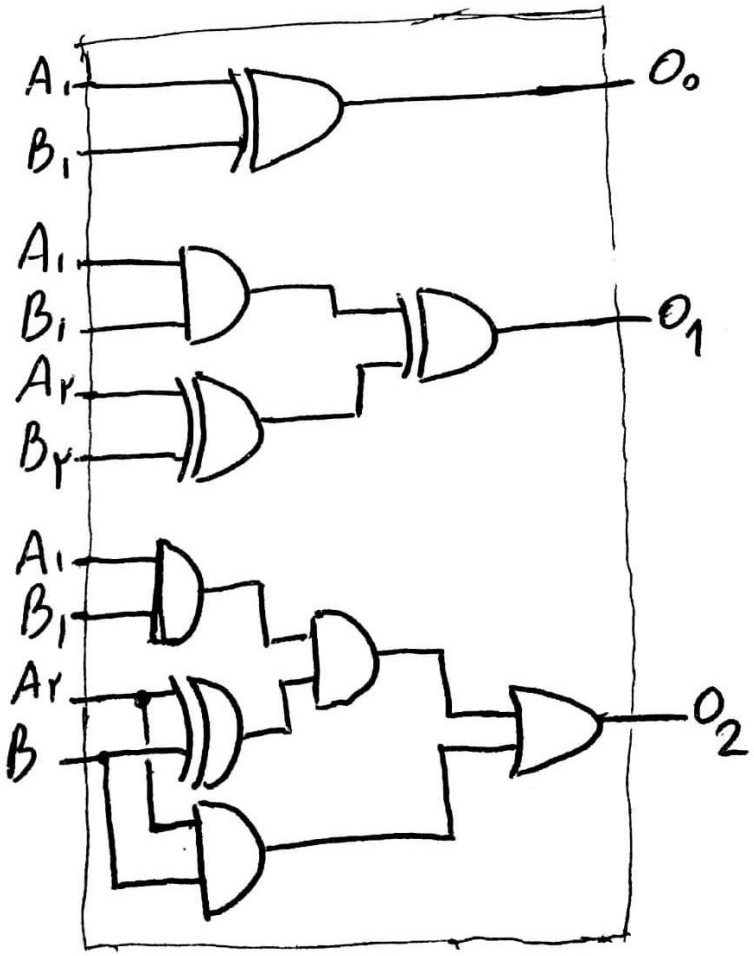
ترتیب زمانی انجام عملیات در این سوال مهم نیست. بنابراین نیازی به در نظر گرفتن تاخیر برای انجام عملیات نیست، با این ترتیب مادامی که در حال جمع دو عدد باینری هستید ورودی ها نیز در حال اعمال شدن هستند. برای سهولت در طراحی شبکه، ابتدا هر سه خروجی را به ترتیب ارزش مکانی مشخص کرده و سپس به صورت جداگانه برای هر خروجی، شبکه متناسب با آن را بدست آورید.

در گزارش خود علاوه بر رسم شبکه نهایی با وزن های آن، شبیه سازی مربوطه را انجام داده و خروجی شبکه برای همه حالت های ورودی ذکر کنید .

چون جمع کننده ما بیت cin ندارد پس با اضافه کردن یه half adder به یه full adder یک بیتی می توان جمع کننده خواسته شده را درست کرد که گیت های آن به صورت شکل زیر می شود.

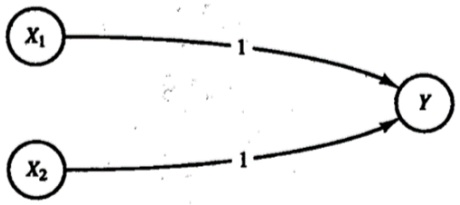


اگر بخواهیم مطابق با سوال عمل کنیم برای هر بیت خروجی باید گیت های جدا گانه در نظر بگیریم که مدار منطقی ما به شکل زیر میشه:

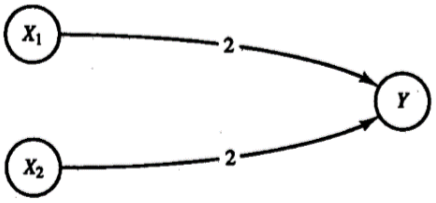


حال تنها کاری که باید انجام داد این است که به جای گیت های منطقی معادل شبکه Mcculloch Pitts آنها را قرار داد:

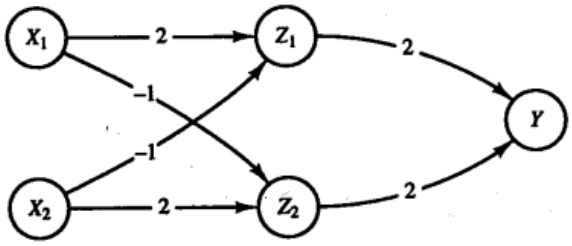
And



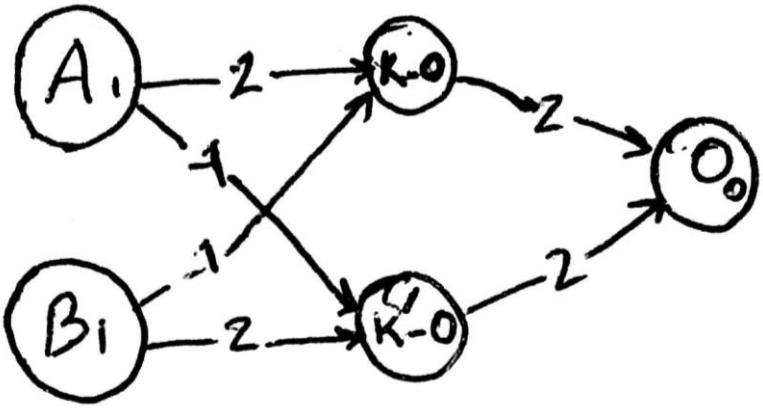
Or



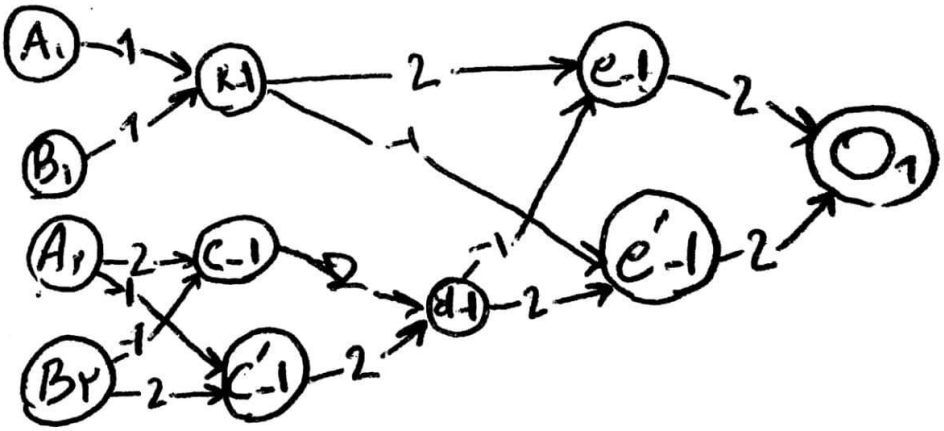
Xor



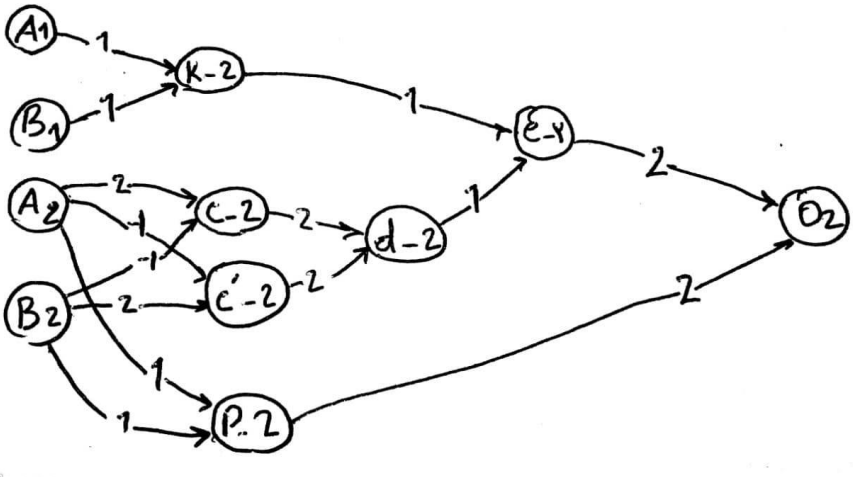
ابتدا برای خروجی اول شبکه عصبی را تشکیل می دهیم:



سپس شبکه عصبی برای خروجی دوم



و همچنین شبکه عصبی برای خروجی سوم:



اسامی نوشته شده بر روی نورون های شبکه عصبی با اسامی به کار رفته در کد پایتون آن یکی است.

شبیه شازی:

برای شبیه سازی این شبکه ابتدا یکه کلاس درست کردیم به اسم logic که دو ورودی را می گیرد و میتواند تمام گیت های منطقی را شبیه سازی کند و با فرا خوانی هر کدام از متد های آن متغییر مورد نظر با تایپ این کلاس به گیت منطقی متناظر با آن تبدیل می شود و خروجی که بر یگرداند خروجی آن گیت است که از ورودی های داده شده به آن کلاس به دست آمده است.

کلاس logic شامل متد های and,or ,andnot و xor(که خود آن از ترکیب سه گیت منطقی تشکیل شده) می باشد.

```
class logic:
  def __init__(self,in1,in2):
    self.in1 = in1
    self.in2 = in2
    self.teta = 2
  def And (self):
    net = 1*self.in1 + 1* self.in2;
    if net >= self.teta :
      out =1
    else:
      out=0
    return out
  def Or (self):
    net = 2*self.in1 + 2* self.in2;
    if net >= self.teta :
      out =1
    else:
      out=0
    return out
  def AndNot (self):
    net = 2*self.in1 + (-1)* self.in2;
    if net >= self.teta :
      out =1
    else:
      out=0
    return out
  def Xor (self):
    Z1= self.AndNot()
    a = self.in1
    b=self.in2
    self.in1=b
    self.in2=a
    Z2= self.AndNot()
    self.in1=Z1
    self.in2=Z2
    Y= self.Or()
    self.in1=a
    self.in2=b
```

```
    return Y
```

حال برای پیاده سازی این ادر فقط کافی است هر کدام از نورون ها را نام گذاری که رده و برابر متغییری قرار داده و به هم متصل کرد تا جمع کننده مورد نظر ساخته شود:

```
x1=[0,0,0,0,1,1,1,0,0,1]
x2=[0,0,0,0,0,0,0,1,1,1]
y1=[0,1,0,1,1,0,1,0,1,1]
y2=[0,0,1,1,0,1,1,1,1,1]
for i in range(0,len(x1)) :
  k_0 = logic(x1[i],y1[i])
  O0= k_0.Xor()

  k_1= logic(x1[i],y1[i])
  j_1=k_1.And()
  c_1=logic(x2[i],y2[i])
  d_1=c_1.Xor()
  e_1=logic(j_1,d_1)
  O1=e_1.Xor()

  k_2= logic(x1[i],y1[i])
  j_2=k_2.And()
  p_2= logic(x2[i],y2[i])
  q_2=p_2.And()
  c_2=logic(x2[i],y2[i])
  d_2=c_2.Xor()
  e_2=logic(j_2,d_2)
  w_2=e_2.And()
  m_2=logic(w_2,q_2)
  O2= m_2.Or()

  print("hasel %s%s ba %s%s = %s%s%s hast" %(x2[i],x1[i],y2[i],y1[i],
O2,O1,O0))
```

کد مورد نظر این کار را انجام داده و حاصل جمع تمام ورودی ها را هم به ما می دهد که برابر:

```
hasel 00 ba 00 = 000 hast
hasel 00 ba 01 = 001 hast
hasel 00 ba 10 = 010 hast
hasel 00 ba 11 = 011 hast
hasel 01 ba 01 = 010 hast
hasel 01 ba 10 = 011 hast
hasel 01 ba 11 = 100 hast
hasel 10 ba 10 = 100 hast
hasel 10 ba 11 = 101 hast
hasel 11 ba 11 = 110 has
```

همان طور که ملاحظه می شود جواب تمام ورودی ها درست است پس شبکه پیاده سازی شده درست کار کرده و کد نوشته شده نیز صحیح است

سوال ۲ – Adaline

فرض کنید داده های ما در دو بعد، به صورت زیر تعریف شده اند. (x, y)

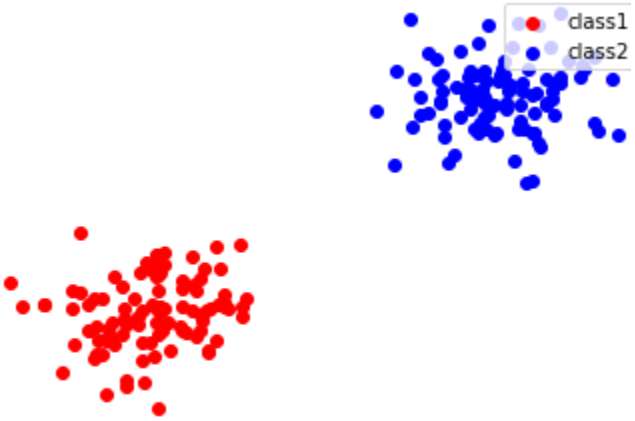
x : متغییر تصادفی نرمال با میانگین mx و انحراف معیار σx

y : متغییر تصادفی نرمال با میانگین my و انحراف معیار σy

الف (دو دسته داده به صورت زیر تعریف کنید و نمودار پراکندگی آنها را رسم نمایید.)نتیجه باید چیزی شبیه شکل زیر باشد).

دسته اول: شامل 100 داده است، که متغییر x آن دارای میانگین صفر و انحراف معیار 0.3 و متغییر y آن هم دارای میانگین صفر و انحراف معیار 0.3 است.

دسته دوم: شامل 100 داده ا ست، که متغییر x آن دارای میانگین 2 و انحراف معیار 0.3 و متغییرy آن هم دارای میانگین 2 و انحراف معیار 0.3 است.



ب) با استفاده از روش Adaline یک شبکه عصبی را آموزش دهید که این دو دسته داده را از هم جدا نماید. نمودار تغییرات خطا یعنی ½(t-net)^2را رسم نمایید. دلیل خوب یا بد جدا شدن داده ها را توجیه نمایید.

ج) فرض کنید تعداد داده ها به صورت زیر تغییر نماید. قسمت ب را برای این داده های جدید تکرار نمایید.

دسته اول: شامل 100 داده است، که متغییر x آن دارای میانگین 2 و انحراف معیار 1 و متغییر y آن هم دارای میانگین 1 و انحراف معیار 0.1 است.

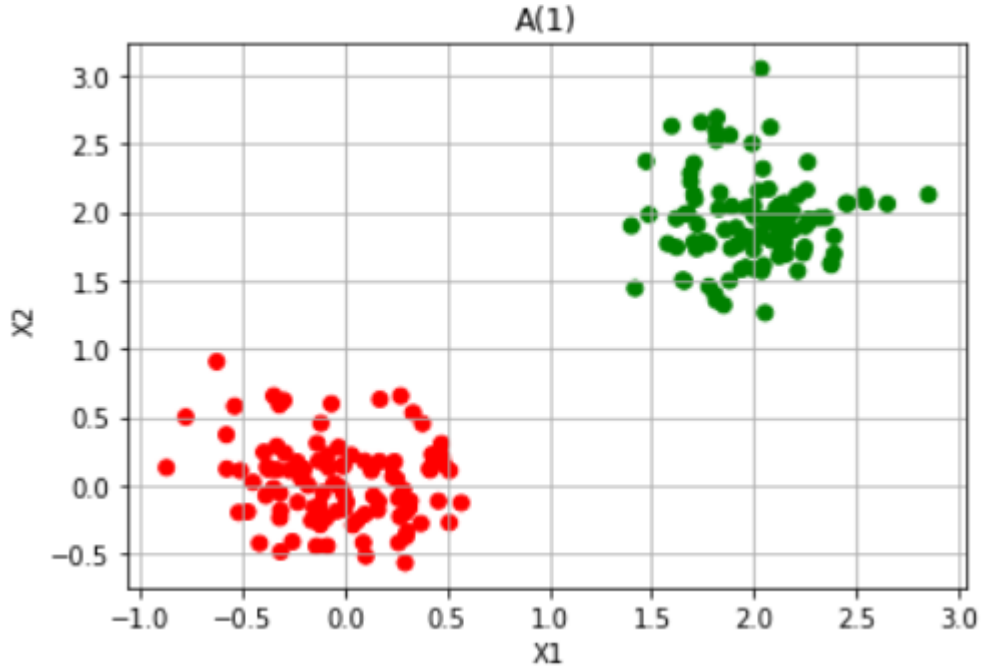
دسته دوم: شامل 20 داده است، که متغییر x آن دارای میانگین 1 - و انحراف معیار 0.4 و متغییر y آن هم دارای میانگین 2 و انحراف معیار 0.4 است.

الف) با استفاده از کتابخانه numpy ، 4 دسته آرایه با درایه های رندوم نرمال همان طور که در سوال گفته شده درست می کنیم و با استفاده از کتابخانه pandas این 4 دسته را داده را دسته بندی می کنیم دو تای اول را برای نقاط مربوط به یک لیبل و دو دسته ی بعد را برای نقاط مربوط به لیبل دیگر در نظر می گیریم سپس این دو دسته را در یک فریم قرار می دهیم و لیبل بندی می کنیم سر آخر هم در آرایه ای نقاط را می ریزیم در آرایه ای دیگر لیبل ها را که یکی می شود ورودی و دیگری خروجی شبکه ما.

کد ها به صورت زیر است:

```
 import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
mu1 ,sigma1 = 0 , 0.3
mu2 ,sigma2 = 2 , 0.3
np.random.seed(30)

xp1 = np.random.normal(mu1 , sigma1 , 100)
xp2 = np.random.normal(mu1 , sigma1 , 100)

xn1 =np.random.normal(mu2 , sigma2 , 100)
xn2 =np.random.normal(mu2 , sigma2 , 100)
XP = pd.DataFrame([],columns=['x1','x2'])
XP['x1']=xp1
XP['x2']=xp2
XP['label']=np.ones((len(xp1),1) , dtype=int)

XN = pd.DataFrame([],columns=['x1','x2'])
XN['x1']=xn1
XN['x2']=xn2
XN['label']=(-1)*(np.ones((len(xn1),1) , dtype=int))

X=XP.append(XN, ignore_index=True)
x_train=X[['x1','x2']]
x_train=x_train.to_numpy()
y_train=X[['label']]
y_train=y_train.to_numpy()
```

سپس در نموداری نقاط را به همرا تفکیک رنگ برای هر لیبل رسم میکنیم:



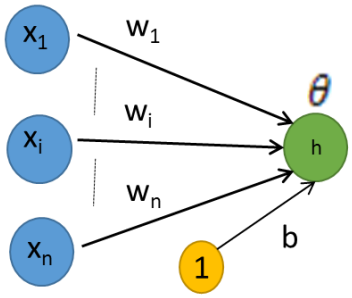
ب) کلاسی درست می کنیم به اسم Adailne که متد های مختلفی دار که هر کدام مختصر توضیح داده می شود:

کلاس Adaline ورودی های بایاس و وزن های اولیه همینطور مقدار حداکثر خطا و ضریب یاد گیری و همینطور تعداد حداکثر ایپاک را می گیرد.

متد net : جمع وزن دار ورودی ها را به همراه بایاس محاسبه می کند.

متد h : خروجی شبکه را با استفاده از جمع وزن دار و تابع مقایسه محاسبه می کند.

متد update : با توجه به نحوه ی یاد گیری Adaline وزن ها را آپدیت می کند.



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

$$w_i^+ = w_i^- + \alpha(t - net)x_i$$

$$b^+ = b^- + \alpha(t - net)$$

متد error : مقدار خطا را با توجه به فرمولی که در سوال گفته شده محاسبه می کند.

متد ok : این متد محاسبه می کند ورودی داده شده از مقدار حداکثر خطا خطایش کمتر است یا نه.

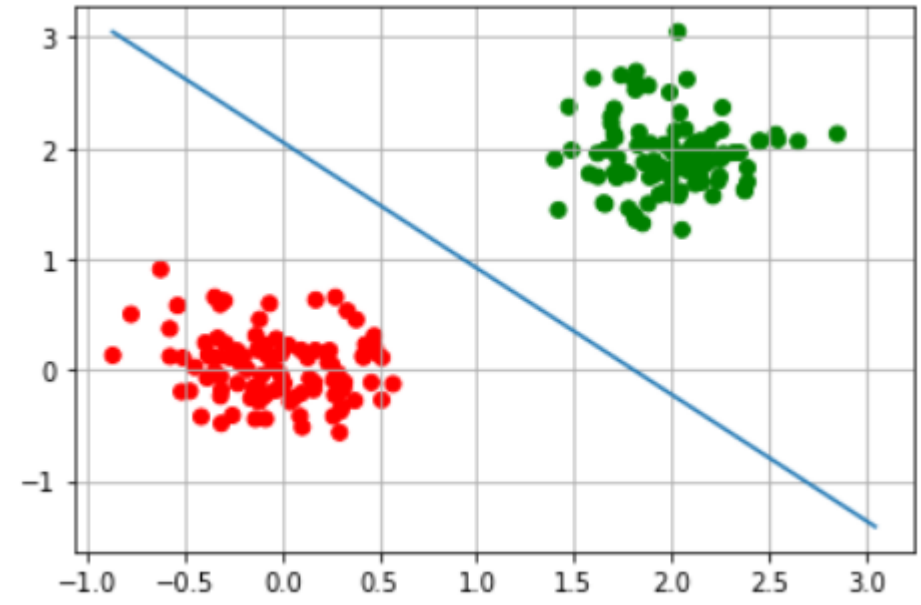
متد train : ورودی ها را به آن داده می دهیم و وزن ها را آموزش می دهد.

کد کلاس مورد نظر:

```
class Adaline:
  def __init__(self, b, w, alpha,threshold,epoch=5000):
    self.epoch=epoch
    self.alpha=alpha
    self.w=w
    self.b =b
    self.threshold=threshold
  def net (self,xin):
    #NET = np.dot(xin,self.w)+self.b
    NET= xin.dot(self.w)+self.b
    return NET
  def h (self,xin):
    if self.net(xin)>=0:
      H=1
    else:
      H=0
    return H
  def update(self , xin , t_out):
    self.w = self.w + self.alpha*(t_out - self.net(xin))*xin
    self.b = self.b + self.alpha*(t_out - self.net(xin))
  def error (self,xin,t_out):
    return 0.5 *((t_out.T - self.net(xin))**2)
  def ok (self,xin,t_out):
    if self.error(xin , t_out) >= self.threshold :
      OK=0
    else:
      OK=1
    return OK
  def train(self , xin , t_out):
    error =[]
    for i in range(1,self.epoch +1):
      e=0
      for s in range(0,len(xin)) :
        self.update(xin[s],t_out[s])
        ee=self.ok(xin[s],t_out[s])
        e=e+ee
      error +=[self.error(xin,t_out).max()]
```

```
        if self.error(xin,t_out).max() <= self.threshold:#e==len(xin):
            break
    #@ print("__________")
     #print(self.epoch)
     return error
```

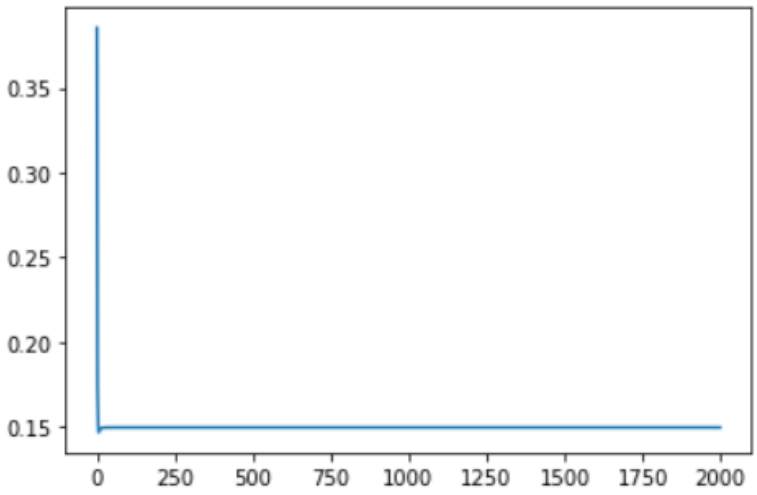
حال ورودی های مورد نظر خود را با حداکثر خطای 0.1 که به این معنی است که اگر به این خطا بتواند برسد دیگر آموزش را ادامه نمی دهد و ضریب یاد گیری 0.01 (اگر ضریب را مقدار بزرگی انتخاب کنید باز دهی کمتر می شود) و ضریب های اولیه رندم و حداکثر ایپاک 2000 به شبکه می دهیم تا آموزش ببیند حاصل آن نمودار زیر می شود:




```
0.1496449953935933
```

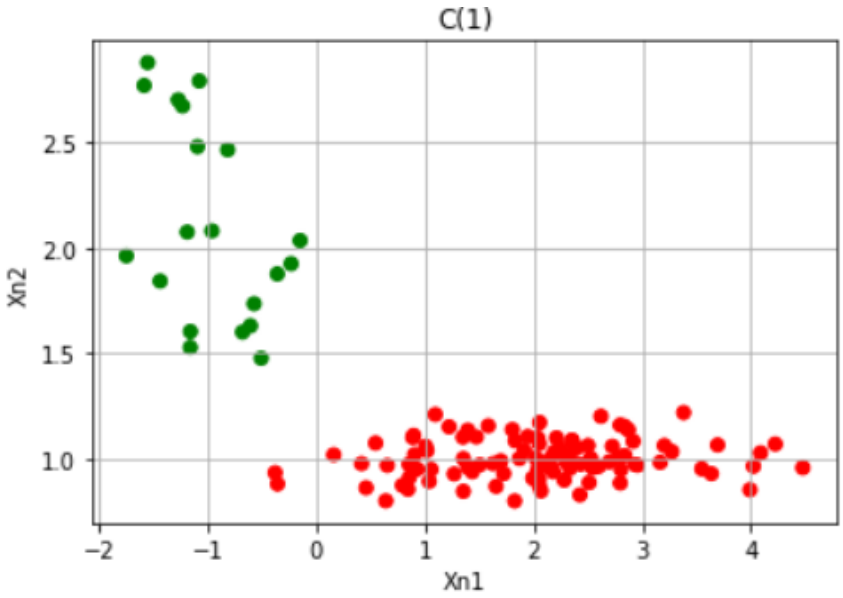
عدد نوشته شده در زیر نمودار مقدار بیشترین خطای برای تمام ورودی ها است بعد از یادگیری

نمودار خطا بر حسب ایپاک:



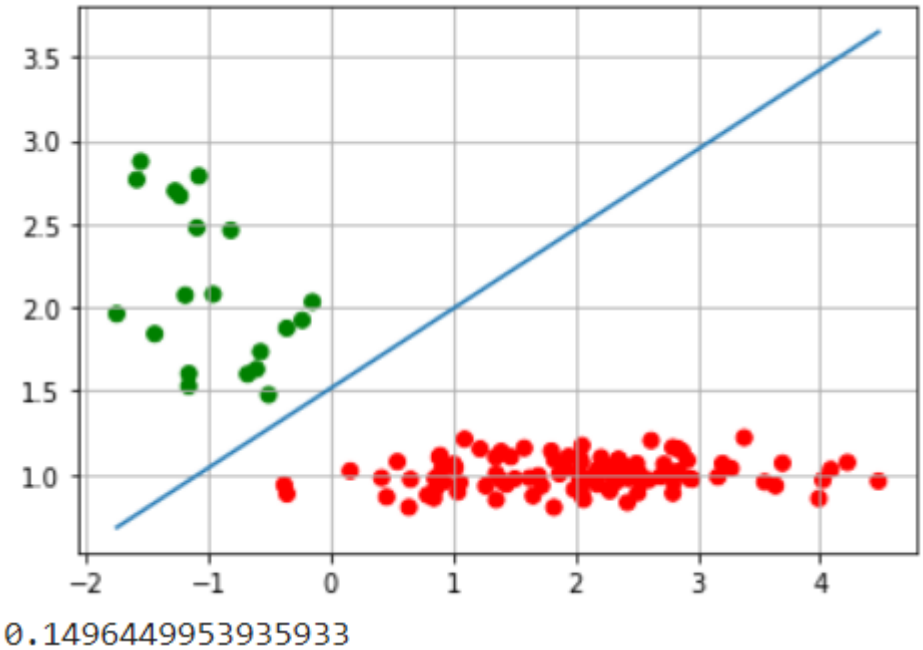
همانطور که مشاهده می کنید مقدار خطا در ایپاک های اولیه تا حدود 0.15 کم شده و مقدار آن ثابت مانده است که یعنی کمتر مقدار ممکن همان 0.14 است که زیر نمودار قبلی نوشته شده

ج) حال دوباره داده ها با توجه به صورت سوال آماده می کنیم که نمودار آن شکل زیر است:



شبکه به علت نا متقارن بودن سخت آموزش می بیند پس از سعی وخطا به این اعداد نوشته شده برای پارامتر ها رسیده شده

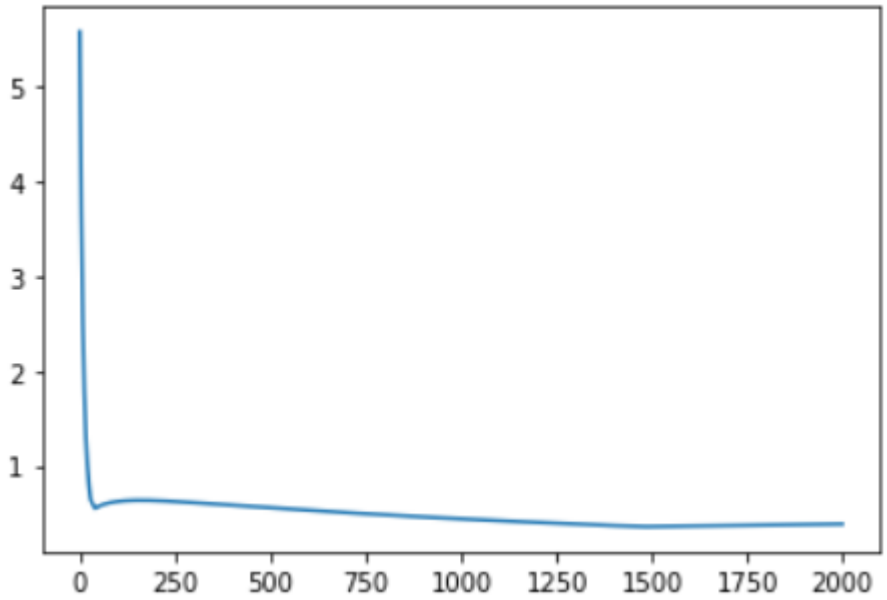
بیشترین خطا 0.1 و ضریب یاد گیری 0.001 و حداکثر ایپاک ها 2000 تا



0.1496449953935933

همانطور که مشاهده می شود 0.14 خطای حداکثر است

نمودار خطا بر حسب ایپاک:



همان طور که مشاهده کردید به شبکه ضریب یاد گیری کوچک تری را اعمال کردیم تا دقیق قبلی برسیم و تعداد ایپاک خیلی بیشتری طول می کشد تا شبکه ثابت شود نسبت به قبل که بعنی شبکه در حالت نا متقارن خوب عمل نمی کند.

سوال ۳ – Madaline

در این سوال به شبکه Madaline میپردازیم .

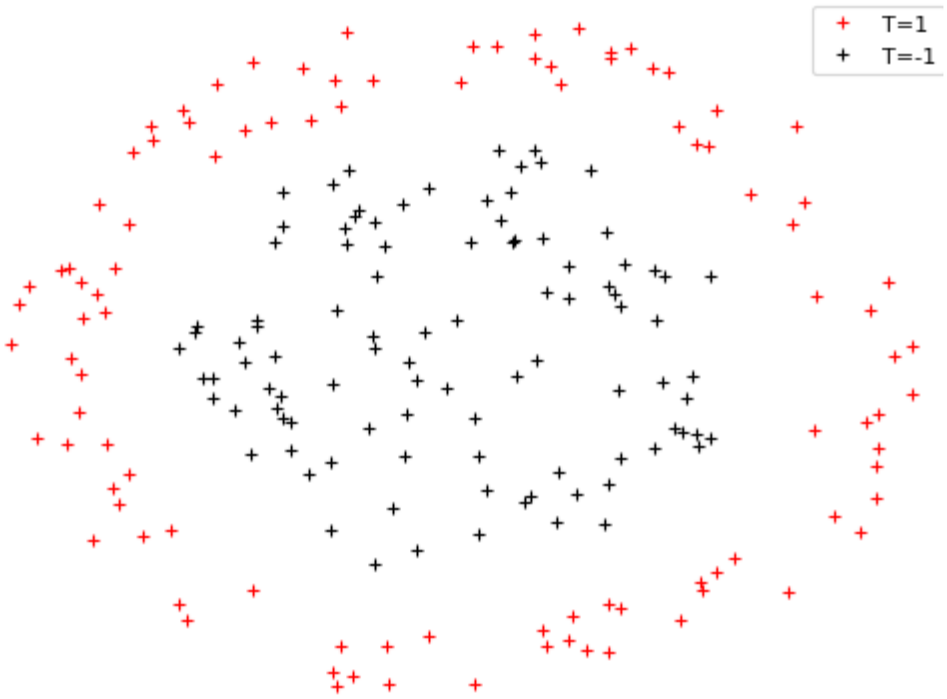
الف) در ابتدا به دلخواه یکی از الگوریتم های MRI یا MRII را که در کتاب مرجع موجود است، توضیح مختصری دهید.

ب) با استفاده از کتابخانه های آماده مانند Pandas ، ابتدا مجموعه داده ای که مربوط به این سوال است (Question3.csv) را بارگزاری نمایید و منحنی پراکندگی داده ها را رسم نمایید. (ستون اول داده ها مربوط به ویژگی اول، ستون دوم مربوط به ویژگی دوم و ستون سوم کلاس هر داده را مشخص میکند.)

ج) با استفاده از الگوریتمی که در قسمت الف مطالعه نمودید، شبکه ای بر اساس آن الگوریتم آموزش دهید.

- این نقاط را یک بار با 3 نورون، یک بار با 4 نورون و یک بار با ۸ نورون جدا کنید.
- خط های جدا کننده را در هر حالت به صورت مجزا رسم نمایید.

د) هر سه نمودار حاصل شده، دقت و تعداد ایپاک های هر سه حالت را با هم مقایسه و تحلیل نمایید .



الف) شبکه madailine شامل دو لایه نورون است که لایه آخر آن بک نورون است که تمام خروجی های لایه ی اول در آن وارد می شود که معمولا یا or هست یا and که برای الگوریتم یادگیری MR1 این نورون یک نورون or هست که وزن های آن ثابت بوده و لرن نمی شود . تابع فعال سازی تمام نورون های آن یه تابع مقایسه گر با صفر است که خروجی 1- , 1 را می دهد.

الگوریتم : مرحله 0: وزن ها را به جز در لایه آخر در ابتدا به صورت رندوم انتخاب می شود

مرحله 1: تا زمانی که stoping condition برابر true نباشد مرحله 2 تا 8 را انجام می دهیم.

مرحله 2: برای هر ایپاک مرحله های 3 تا 7 رو انجام می دهیم

مرحله 3: ورودی را وارد شبکه می کنیم

مرحله 4: net های نورون های لایه مخفی رو حساب می کنیم

مرحله 5: خروجی توابع لایه های مخفی رو حساب می کنیم

مرحله6 : خروجی لایه آخر را محاسبه می کنیم

مرحله 7: ارور را محاسبه می کنیم و با توجه به آن وزن ها را آپدیت می کنیم:

Step 7. **Determine error and update weights:**
If $t = y$, no weight updates are performed.
Otherwise:
If $t = 1$, then update weights on Z_J,
the unit whose net input is closest to 0,

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

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

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

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

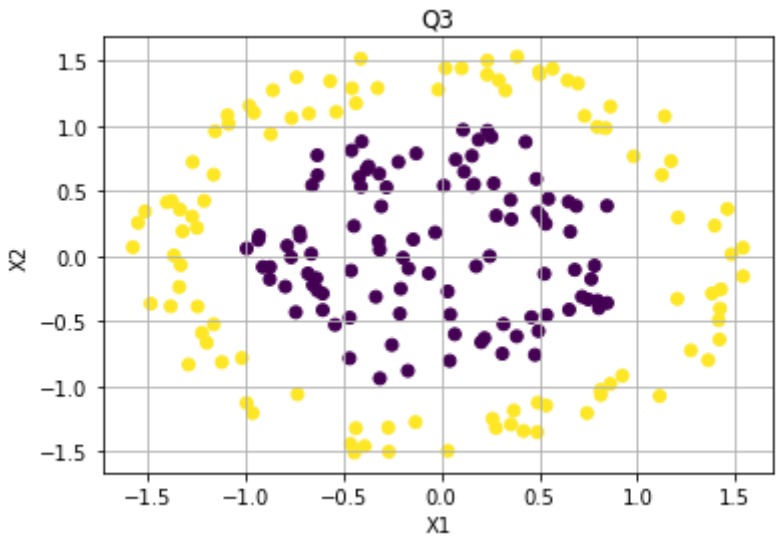
$$w_{ik}(\text{new}) = w_{ik}(\text{old}) + \alpha(-1 - z\_in_k)x_i.$$

مرحله 8: stoping codition را محاسبه می کنیم و تصمیم می گیریم ادامه بدیم یا نه.

ب) با توجه به کد های زیر داده ها را وارد پایتون کرده و نقاط را از لیبل های آن جدا می کنیم و با تابع scatter آنها را رسم می نماییم.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.metrics import accuracy_score

data=pd.read_csv('Question3.csv',header=None)#.sample(frac=1).values
xin=data[[0,1]]
yout=data[[2]]
X=xin.to_numpy()
Y=yout.to_numpy()
Y[Y==0]=-1

plt.scatter(X[:,0],X[:,1] , c=Y)
plt.grid()
plt.title("Q3")
plt.xlabel('X1')
plt.ylabel('X2')
plt.show()
```

که حاصل آن نمودار زیر میشود:



ج) در این سوال از یک کلاس MAdaLine و برای نرونها از کلاس AdaLine استفاده میکنیم. توضیحات کلاس AdaLine در سوال قبلی موجود میباشند. در این سوال فقط متدهای net، update وh از کلاس AdaLine استفاده میشوند. یک کلاس MAdaLine نیز برای مقداردهی اولیه و آموزش شبکه نیز داریم.

برای آموزش این شبکه از الگوریتم MR1 استفاده میشود. در ذیل به اختصار هریک از متدهای این کلاس را توضیح میدهیم.

متد out_madaline : این متد در واقع پیاده سازی گیت or با استفاده از نورون McCulloch-Pitts برای لایه آخر شبکه می باشد.

متد train : این متد پیاده سازی الگوریتم MR1 میباشد. ابتدا جمع وزندار ورودیهای آنها (zin) محاسبه میشوند. با توجه به خروجی هریک از نورونها که ورودی گیت or نورون McCulloch-Pitts میباشند خروجی کلی شبکه y محاسبه میگردد. در صورتی که این خروجی با برچسب اصلی برابر نباشد دو حالت را در نظر میگیریم و با توجه به اینکه کدام روی داده است ضرایب نورونهای AdaLine را به روز رسانی میکنیم در غیر این صورت به روز رسانی انجام نمیشود

حالت اول: در صورتی که برچسب اصلی یک باشد، ضرایب نورونی که قدر مطلق جمع وزندار ورودیهای (zin) از همه کمتر باشد را به روز رسانی میشوند.

حالت دوم: در صورتی که برچسب اصلی یک نباشد، ضرایب نورونهایی که جمع وزندار ورودی های آنها (zin) مثبت دارند به روز رسانی میشوند.

در صورتی که در یک دور بررسی تمام ورودیها خطایی نداشته باشیم الگوریتم خاتمه میابد در غیر این صورت دوباره از اول اجرا میشود.

کد های دو کلاس به صورت زیر است:

```
class Adaline:
  def __init__(self, b, w, alpha):
    self.alpha=alpha
    self.w=w
    self.b =b
  def net (self,xin):
    NET = np.dot(xin,self.w)+self.b
    return NET
#    return sum(i * w for i, w in zip(xin, self.w)) + self.b
  def h (self,xin):
    if self.net(xin)>=0:
      H=1
    else:
```

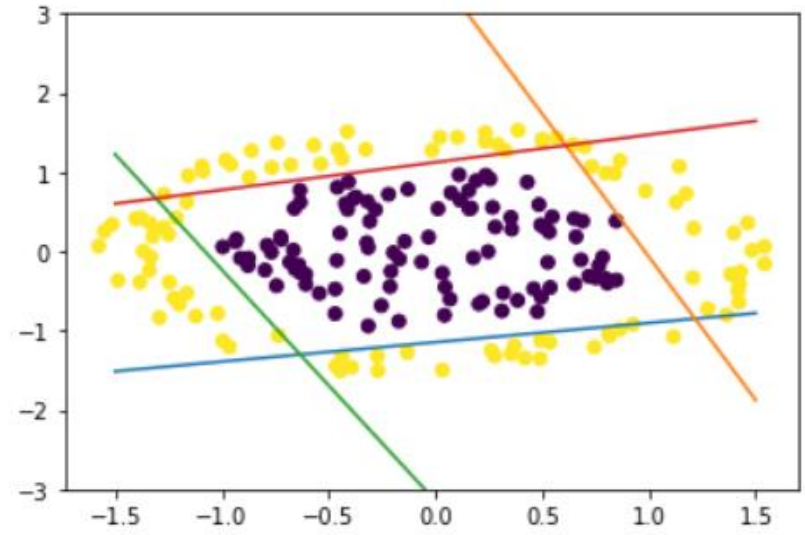
```
      H=-1
    return H
  def update(self , xin , t_out ):
      self.w = self.w + self.alpha*(t_out - self.net(xin))*xin
      self.b = self.b + self.alpha*(t_out - self.net(xin))

class Madaline:
  def __init__(self,neurons,alpha,dimension):
    self.neurons=neurons
    self.alpha=alpha
    self.dimension=dimension
    adalins=[]
    for i in range(neurons):
      adalins += [Adaline(np.random.rand(1) , np.random.rand(dimensio
n), alpha)]
    self.adalins = adalins
  def out_madaline (self, x_in):
    n=[]
    t=-1
    for i in range(self.neurons):
      n +=[self.adalins[i].h(x_in)]
    #if sum(n) >= -1*(self.neurons-1) :
      if n[i]>= 0:
         t=1
    #print(n)
    return t
  def train(self,x_in,t_out ,max_epoch=1000):
   for epoch in range(max_epoch):
     e=0
     for k in range(len(x_in)):
       z_in=[]
       for i in range(self.neurons):
         z_in += [self.adalins[i].net(x_in[k])]
       y=self.out_madaline(x_in[k])
       if y != t_out[k] :
         e +=1
         if t_out[k]==1:
           zin=np.asarray(z_in)
           idx=np.argmin(abs(zin))
           self.adalins[idx].update(x_in[k],t_out[k])
         else:
           for l in range(len(z_in)):
             if z_in[l] >0:
               self.adalins[l].update(x_in[k],t_out[k])
     if e==1:
       return epoch
   return epoch
```

```
def out_madaline_all (self ,X):
  y=[]
  for n in range(len(X)):
    y +=[self.out_madaline(X[n])]
  return y
```

حال حاصل نمودار های این شبکه را برای 3 و 4 و 8 نورون مخفی مشاهده می کنید:

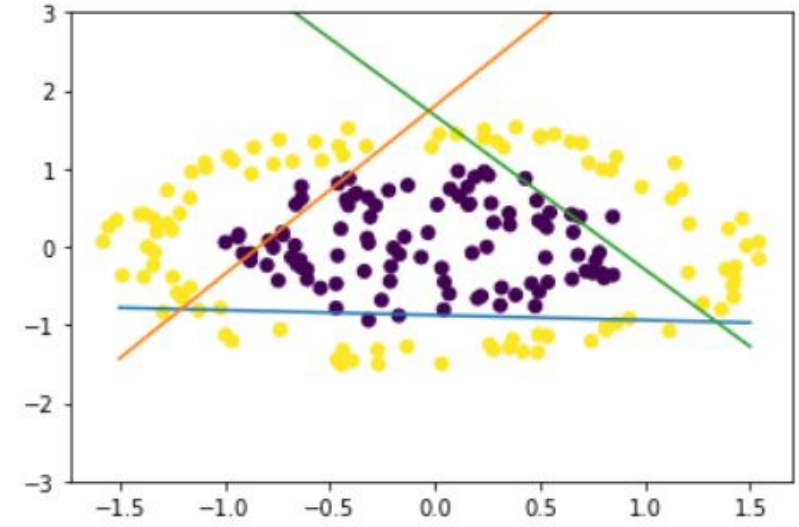
آلفا برابر 0.1

4 نورون:



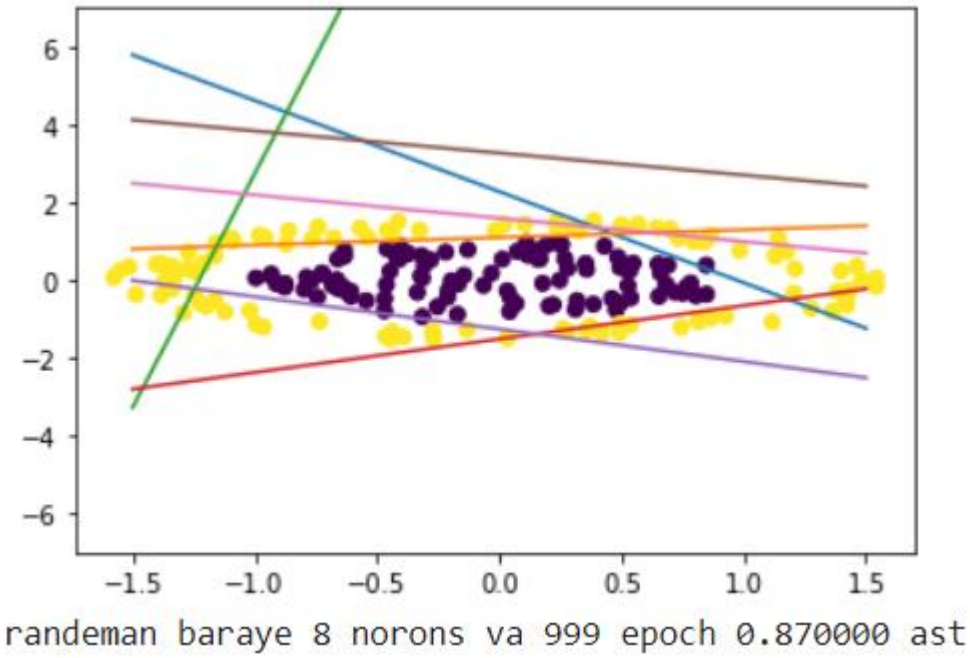
randeman baraye 4 norons va 85 epoch 1.000000 ast

3 نورون: آلفا برابر 0.5



randeman baraye 3 norons va 1999 epoch 0.870000 ast

8 نورون : آلفا برابر 0.5



ج) تعداد ایپاک ها برای 8 و 3 نورون متوقف نمی شود و الگوریتم نمی تواند حالت بهینه ای برای آنها پیدا کند که ازتمامی خطوط برای جدا سازی استفاده شود و تمام داده ها هم جدا شود اما برای 4 نورون با 85 ایپاک می شود شبکه را به درستی جدا سازی کرد به طور کلی نباید تعداد خطوط را بیش از نیاز زیاد کرد و همچنین نباید انقدر کم گرفت که جدا سازی غیر ممکن شود و تعداد نورون ها باید با تجوه به شکل حداقل ممکن انتخاب شود تا شبکه به خوبی لرن شود

سوال ۴ – Perceptron

یک شبکه پرسپترون با سه ورودی x1,x2,x3 در نظر بگیرید. وزن های و بایاس این شبکه به صورت زیر است.

وزن ها	مقدار ها
W1	0.2
W2	0.7
W3	0.9
bias	-0.7

حال به ازای ورودی های x1=0, x2=1, x3=1 خروجی مورد انتظار ما برابر 1 - است. نرخ یادگیری را 0.3 در نظر بگیرید و به روز رسانی وزن ها را تا سه مرحله ادامه دهید.)توجه شود که تابع فعال ساز را یک واحد مقایسه گر با عدد صفر در نظر بگیرید. (

حل این سوال، حل تشریحی است و نیازمند پیاده سازی نمی باشد.

ابتدا ورودی ها را به شبکه می دهیم تا خروجی را محاسبه کنیم .

X1	X2	X3	W1	W2	W3	B	H
0	1	1	0.2	0.7	0.9	-0.7	

$$\text{net} = w_1x_1 + \cdots . + w_ix_i + \cdots + w_nx_n + b$$

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

$$\theta : \text{non} - \text{negative threshold}$$

Teta را به علت این که سوال مشخص نکرده است صفر می گیریم.

مرحله یک:

ابتدا با ورودی های داده شده و وزن های موجود خروجی تابع net محاسبه می شود :

net = 0.9

حال net را به تایع مقایسه ای h وارد می کنیم تا خروجی شبکه عصبی مشخص شود:

h = 1

خروجی شبکه باید 1- باشد ولی ما خروجی شبکه خود را 1 دریافت کردیم پس error ما غیر صفر است و باید ژروتکل را ادامه دهیم و وزن های موجود را با استفاده از فرمول شبکه پرسپترون بروز رسانی کنیم.

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

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

W1(old)	W2(old)	W3(old)	B(old)	H	W1(new)	W2(new)	W3(new)	B(new)
0.2	0.7	0.9	-0.7	1	0.2	0.4	0.6	**-1**

مرحله دو:

حال با وزن های جدید دوباره خروجی سیستم را مشاهده میکنیم.

net = 0 , h=0

خروجی شبکه برابر صفر است زیرا شبکه پرسپترون به علت وجود تتا هر چند که ما تتا را صفر گرفتیم باز هم داده روی خط جدا کننده را خطا حساب میکند.

پس باید بروز رسانی وزن ها را ادامه دهیم

W1(old)	W2(old)	W3(old)	B(old)	H	W1(new)	W2(new)	W3(new)	B(new)
0.2	0.4	0.6	-1	0	0.2	0.1	0.3	**-1.3**

مرحله سه :

حال دوباره خروجی شبکه رو محاسبه می کنیم.

net = 0.9 , h=1

شبکه خروجی درست را پیدا کرد ولی چون در صورت سوال گفته شده سه بار به روز رسانی را انجام دهید باید یک بار دیگر هم محاسبات به روز رسانی وزن ها را انجام داد.

W1(old)	W2(old)	W3(old)	B(old)	H	W1(new)	W2(new)	W3(new)	B(new)
0.2	0.1	0.3	-1.3	-1	0.2	-0.2	0	**-1.6**

محاسبات خروجی دوباره انجام می شود :

net = -1.8 , h =-1

شبکه سه بار لرن شد و با دو بار هم می توانست به عدد درست برسد.

(کد های کمکی برای محاسبه در فایل مربوط به سوال 4 آمده است)