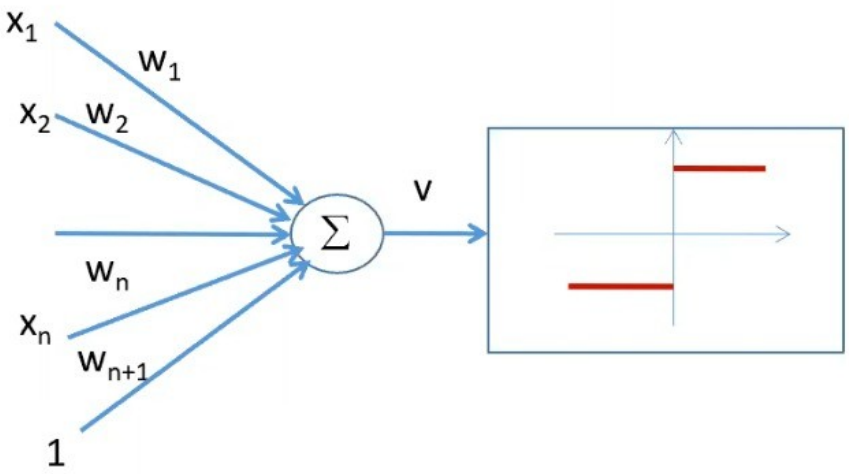
PERCEPTRON

Perceptron, tek bir nörondan oluşan en temel yapay sinir ağ birimidir.

x(giriş) kümesiyle w(ağırlık) kümesi çarpılarak skaler bir v elde edilmektedir. 0'dan küçükse y -1, büyükse y +1 değeri almaktadır..



1)

```
import statistics
from functools import reduce
import numpy as np
#a=np.random.randint(1,10,size=(4,40))
#b=np.random.randint(1,10,size=(19,6))
#np.save('arr', a)
```

np.random.randint koduyla arrayımızı oluşturduk.

np.save koduyla verileri kaydettik.

Farklı ağırlıklara göre sonuçları değerlendirebilmek için 20 farklı ağırlık matrisi oluşturduk.

```
one_arr= np.ones(20).reshape(1,20)
minusone_arr= one_arr* (-1)



kümel=random_matrix[:,0:20]
küme2=random_matrix[:,20:40]

kümel=np.concatenate((kümel,minusone_arr))
küme2=np.concatenate((küme2,one_arr))
kümel=np.concatenate((kümel,one_arr))
küme2=np.concatenate((küme2,one_arr))
```

1X20 boyutlu 1 matris oluşturduk.
1X20 boyutu -1 matris oluşturduk.

Lineer ayrıştırılabilr yapmak için ilk 20 değere 5. boyut değeri olarak -1 ekledik , son 20 değere ise 5. boyut değeri olarak 1 ekledik.

Daha sonra iki kümemize de bias değeri olarak 1 ekledik.

```
egitim=np.concatenate((küme1[:,0:12],küme2[:,0:13]),axis=1)
test=np.concatenate((küme1[:,12:20],küme2[:,13:20]),axis=1)
```

13 tanesi +1 kümesinden, 12 tanesi 1 kümesinden olmak üzere 25 elemanlı eğitim kümemizi oluşturduk.
7 tanesini +1 kümesinden 8 tanesini-1 kümesinden olmak üzere 15 elemanlı test kümemizi oluşturduk.

```
W_list=[[1,1,1,1,1,1],[1.5,1.5,1.5,1.5,1.5,1.5],[2,2,2,2,2,2],[2.5,2.5,2.5,2.5,2.5,2.5],[3,3,3,3,3,3]
       ,[3.5,3.5,3.5,3.5,3.5,3.5],[4,4,4,4,4,4],[4.5,4.5,4.5,4.5,4.5,4.5],[5,5,5,5,5,5]
       ,[5.5,5.5,5.5,5.5,5.5,5.5],[6,6,6,6,6,6],[6.5,6.5,6.5,6.5,6.5,6.5],[7,7,7,7,7,7]
       ,[7.5,7.5,7.5,7.5,7.5,7.5],[8,8,8,8,8,8],[8.5,8.5,8.5,8.5,8.5,8.5],[9,9,9,9,9,9]
       ,[9.5,9.5,9.5,9.5,9.5,9.5],[10,10,10,10,10,10]]
W=[[]]


iter_value=[]
iter_value_test=[]
b=1
```

İlk ağırlık olarak W_one arrayini kullandık.
6 boyutlu 20 farklı değerli ağırlık kümemizi oluşturduk.
iter_value hangi ağırlık başlangıçta kullanılırsa kaç iterasyonla doğru sonuca ulaştığımızı belirten listedir.
iter_value_test her başlangıç ağırlığında test kümesinden kaç tane doğru bildiğini gösteren listedir.

```
for w in W_list:
    W.append(w)
    for i in range(50):
        truth = 0
        for j in range(25):
            a = round(sum(W[b]*egitim[:,j]),2)
            #print(a)

            if a>0:

                a=1
            else:
                a=-1
            if (egitim[4,j]-a)==0:
                truth=truth+1
            W.append(W[b]+(0.5*c*(egitim[4,j]-a)*egitim[:,j]))
            b=b+1
        if truth == 25:
            iter_value.append(i+1)
            break
    truth_test=0
    for k in range(15):
        a = round(sum(W[b]*test[:,k]),2)
            #print(a)

        if a>0:

            a=1
        else:
            a=-1
        if (test[4,k]-a)==0:
            truth_test=truth_test+1
        if k==14:
            iter_value_test.append(truth_test)
```

İlk for döngüsünde başlangıç ağırlık değerlerimizi döndürüyoruz.
İkinci for döngüsünde iterasyon yapıyoruz.
İkinci for'un içindeki for döngüsünde ise eğitim kümemizi eğitiyoruz.
Üçüncü for döngüsünde en son elde ettiğimiz ağırlık kümemizi, test kümemizde test ediyoruz.

```
def Average(lst):
    return reduce(lambda a, b: a + b, lst) / len(lst)
Avarage_iter=Average(iter_value)
standart_devision=statistics.stdev(iter_value)
```

İterasyon kümesindeki ağırlıkları alıyoruz ve standart sapmasını alıyoruz.

Sıralı Eğitim Kümesi için:

c	1	0.8	0.5	0.4	0.2
Average iter	14	12.631	11.05	9.89	11.26
İterasyon stan. sapması	8.61	6.64	6.89	6.36	7.19

Average iter: w'ya göre iterasyon ortalaması

c 'yi değiştirdiğimiz zaman 1'de 0.4'e kadar iterasyon ortalamasında düşüş görülmekteyken 0.4'ten sonra artış meydana geldi.

Ağırlık listesi

0	list	6	[1, 1, 1, 1, 1, 1]
1	list	6	[1.5, 1.5, 1.5, 1.5, 1.5, 1.5]
2	list	6	[2, 2, 2, 2, 2, 2]
3	list	6	[2.5, 2.5, 2.5, 2.5, 2.5, 2.5]
4	list	6	[3, 3, 3, 3, 3, 3]
5	list	6	[3.5, 3.5, 3.5, 3.5, 3.5, 3.5]
6	list	6	[4, 4, 4, 4, 4, 4]
7	list	6	[4.5, 4.5, 4.5, 4.5, 4.5, 4.5]
8	list	6	[5, 5, 5, 5, 5, 5]
9	list	6	[5.5, 5.5, 5.5, 5.5, 5.5, 5.5]
10	list	6	[6, 6, 6, 6, 6, 6]
11	list	6	[6.5, 6.5, 6.5, 6.5, 6.5, 6.5]
12	list	6	[7, 7, 7, 7, 7, 7]
13	list	6	[7.5, 7.5, 7.5, 7.5, 7.5, 7.5]
14	list	6	[8, 8, 8, 8, 8, 8]
15	list	6	[8.5, 8.5, 8.5, 8.5, 8.5, 8.5]
16	list	6	[9, 9, 9, 9, 9, 9]
17	list	6	[9.5, 9.5, 9.5, 9.5, 9.5, 9.5]
18	list	6	[10, 10, 10, 10, 10, 10]

iterasyon miktarı

Ind	Type	Size	
0	int	1	20
1	int	1	22
2	int	1	17
3	int	1	16
4	int	1	6
5	int	1	30
6	int	1	24
7	int	1	8
8	int	1	13
9	int	1	13
10	int	1	8
11	int	1	9
12	int	1	11
13	int	1	33
14	int	1	4
15	int	1	7
16	int	1	16
17	int	1	4
18	int	1	5

Eğitim kümemizi küme_1 ve küme_2'den sırasıyla oluşturmak suretiyle karıştırdık:

```
egitim_mix=[]

for i in range(12):
    egitim_mix.append(küme1[:,i].tolist())
    egitim_mix.append(küme2[:,i].tolist())
egitim_mix.append(küme2[:,13])

egitim_mix_array=np.array(egitim_mix)
egitim_mix_array=np.transpose(egitim_mix_array)
```

İterasyon döngümüzün içine karışık eğitim kümemizi yerleştirdik:

```
truth = 0
for j in range(25):
    a = round(sum(W[b]*egitim_mix_array[:,j]),2)
    #print(a)

    if a>0:

        a=1
    else:
        a=-1
    if (egitim_mix_array[4,j]-a)==0:
        truth=truth+1
    W.append(W[b]+(0.5*c*(egitim_mix_array[4,j]-a)*egitim_mix_array[:,j]))
    b=b+1
if truth == 25:
    iter_value.append(i+1)
    break
```

Karışık Eğitim Kümesi için:

c	1	0.8	0.5	0.4	0.2
Average iter_mix	12.05	9.05	9.42	10.31	8.57
İterasyon stan. sapması	14.71	7.79	12.56	11.29	10.18

Average iter: w'ya göre iterasyon ortalaması
c'ye bağlı olarak:
-Karışık eğitim kümesiyle iterasyona soktuğumuzda daha verimli sonuç elde ettik.
-İterasyon ortalamalarında düşüş gerçekleşti ve daha hızlı öğrendi.

Başlangıç Ağırlık Listesi

Indı	Type	Size	
0	list	6	[1, 1, 1, 1, 1, 1]
1	list	6	[1.5, 1.5, 1.5, 1.5, 1.5, 1.5]
2	list	6	[2, 2, 2, 2, 2, 2]
3	list	6	[2.5, 2.5, 2.5, 2.5, 2.5, 2.5]
4	list	6	[3, 3, 3, 3, 3, 3]
5	list	6	[3.5, 3.5, 3.5, 3.5, 3.5, 3.5]
6	list	6	[4, 4, 4, 4, 4, 4]
7	list	6	[4.5, 4.5, 4.5, 4.5, 4.5, 4.5]
8	list	6	[5, 5, 5, 5, 5, 5]
9	list	6	[5.5, 5.5, 5.5, 5.5, 5.5, 5.5]
10	list	6	[6, 6, 6, 6, 6, 6]
11	list	6	[6.5, 6.5, 6.5, 6.5, 6.5, 6.5]
12	list	6	[7, 7, 7, 7, 7, 7]
13	list	6	[7.5, 7.5, 7.5, 7.5, 7.5, 7.5]
14	list	6	[8, 8, 8, 8, 8, 8]
15	list	6	[8.5, 8.5, 8.5, 8.5, 8.5, 8.5]
16	list	6	[9, 9, 9, 9, 9, 9]
17	list	6	[9.5, 9.5, 9.5, 9.5, 9.5, 9.5]
18	list	6	[10, 10, 10, 10, 10, 10]

İterasyon sayısı

Indı	Type	Size	
0	int	1	4
1	int	1	1
2	int	1	10
3	int	1	1
4	int	1	3
5	int	1	59
6	int	1	8
7	int	1	31
8	int	1	8
9	int	1	1
10	int	1	25
11	int	1	1
12	int	1	23
13	int	1	1
14	int	1	10
15	int	1	21
16	int	1	6
17	int	1	1
18	int	1	15

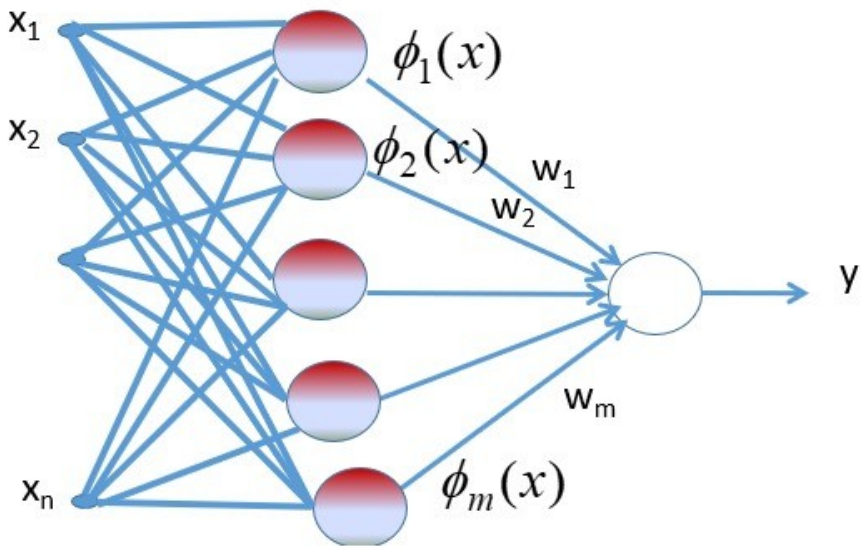
c=1 için maksimum 70 iterasyonda eğittik. Karışık eğitim kümemiz sıralı eğitim kümemizden daha az iterasyonda sonuca ulaştı.

1.b)

```
import numpy as np
random_matrix_new=np.random.randint(1,10,size=(5,40))


one_arr = np.ones(40).reshape(1,40)
random_matrix_new=np.concatenate((random_matrix_new,one_arr))


W=[[1,1,1,1,1,1]]

b=0
c=1

for i in range(500):
    truth = 0
    for j in range(40):
        a = round(sum(W[b]*random_matrix_new[:,j]),2)
        #print(a)

        if a>0:

            a=1
        else:
            a=-1
        if (random_matrix_new[4,j]-a)==0:
            truth=truth+1
        W.append(W[b]+(0.5*c*(random_matrix_new[4,j]-a)*random_matrix_new[:,j]))
        b=b+1
    if truth is 40:
        break
```

Lineer ayrıştırılamayan küme oluşturduk ve 500 defa iterasyona sokmamıza rağmen kümemizi eğitemedik.

Rosenblatt'ın Genlikte Ayrık Algılayıcısı



Giriş katmanı birinci katman ve çıkış katmanı olmak üzere 3 birimden oluşur. Giriş katmanında sadece girişler bulunur, birinci katmanda sabit ağırlık ve fonkiyonlara bulunur yani değişmez yapıdadır. Çıkış katmanı bağlantı ağırlıklarının güncellendiği katmandır.

2)

```
column=np.array([[0,-1],[0,0],[0,1],[1,-1],[1,0],[1,1],[-1,-1],[-1,0]
                 ,[-1,1],[-3,3],[-3,1],[-3,0],[-3,-1],[-3,-3],[-1,3],[-1,-3]
                 ,[0,3],[0,-3],[1,3],[1,-3],[3,3],[3,1],[3,0],[3,-1],[3,-3]
                 ,[-2,3],[-3,2],[-3,-2],[-2,-3],[2,3],[3,2],[3,-2],[2,-3]])
array=column.transpose()
one_arrl = np.ones(33).reshape(1,33)
one_arr = np.ones(9)
one_arr_mines = (-1)*np.ones(24)
y=np.concatenate((one_arr,one_arr_mines)).reshape(1,33)
array_bias=np.concatenate((array,one_arr1))
W=[[1,1,1]]
b=0
```

Öncelikle 2 boyutlu arraylarimizi koda aktardık.

Sonra bias terimimizi ekleyip array_bias'ı oluşturduk.

Ağırlık kümemizin başlangıç değerini oluşturduk.

```
for i in range(500):
    truth = 0
    for j in range(33):
        a = round(sum(W[b]*array_bias[:,j]),2)
        #print(a)

        if a>0:

            a=1
        else:
            a=-1
        if (y[0,j]-a)==0:
            truth=truth+1
        W.append(W[b]+(0.5*c*(y[0,j]-a)*array_bias[:,j]))
        b=b+1
```

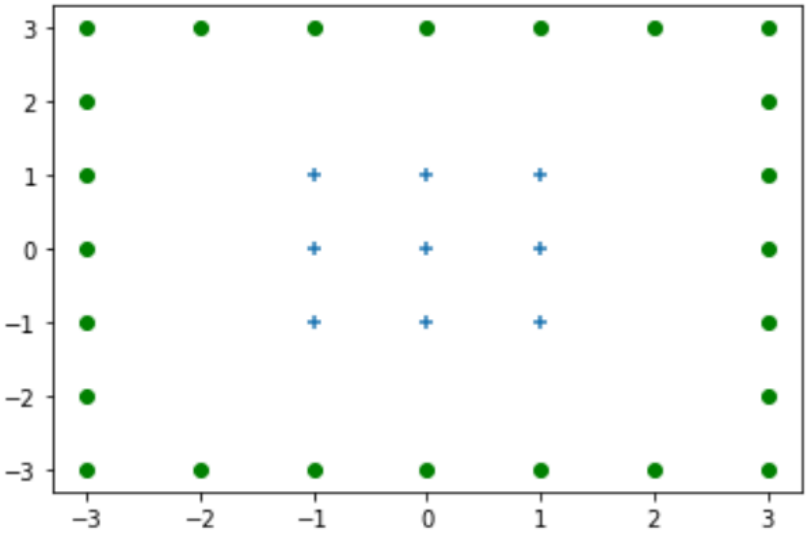
Kümemizi Perceptron'a soktuk ancak 500 iterasyona rağmen eğitemedik.

truth	int	1	17

En fazla 17 doğru sonuca ulaştık.

```
fig = plt.figure()


plt.scatter(column[0:9,0],column[0:9,1], marker='+')
plt.scatter(column[9:33,0],column[9:33,1], c= 'green', marker='o')
```

Kümeyi 2 boyutlu düzlemde çizdirdik.



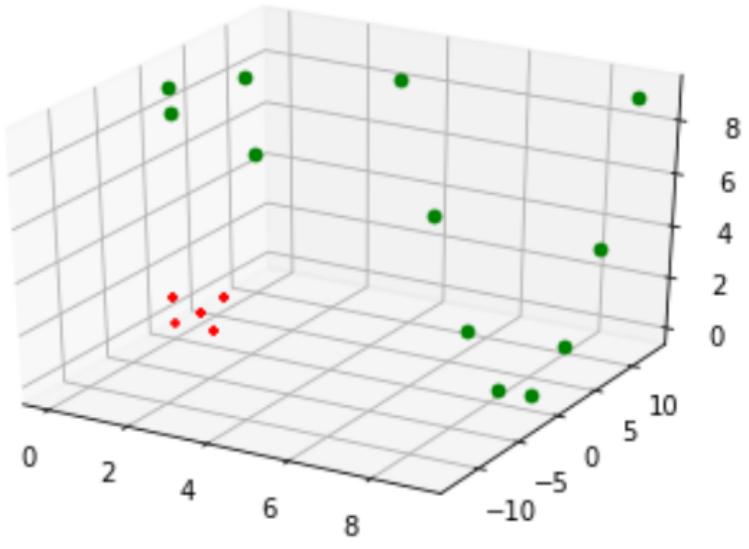
Şekilden de görüldüğü üzere 2 boyutlu düzlemde lineer ayrıştırılamamaktadır. Lineer ayrıştırılabilir hale getirmek için iki seçeneğimiz var:
-Boyut arttırmak
-Örüntü sayısını azaltmak.
Biz 3 boyuta taşıyarak (boyut arttırarak) lineer ayrıştırılabilir hale getirme seçeneğini tercih ettik:

```
for i in range(33):
    fi.append([(column[i,0]*column[i,0]),math.sqrt(2)*column[i,0]*column[i,1],column[i,1]*column[i,1]])
```

[(x1)^2, (√ 2).x1.(x2)^2] bu formülüzasyonu kullanarak 3 boyutlu düzleme taşıdık.

```
fig2 = plt.figure()
ax = fig2.add_subplot(111, projection='3d')
for i in range(0,9):
     ax.scatter(fi[i][0],fi[i][1],fi[i][2],c='red', marker='+')

    #plt.scatter(fi[i][0],fi[i][1],fi[i][2], marker='+')
for j in range(9,33):
     ax.scatter(fi[j][0],fi[j][1],fi[j][2], c= 'green', marker='o')
   # plt.scatter(fi[i][0],fi[i][1],fi[i][2], c= 'green', marker='o')
```

Yeni kümemizi 3 boyutlu düzlemde çizdirdik.



Resimde de görüldüğü gibi lineer ayrıştırılabilir olmuştur.

```
b=0
W1=[[1,1,1,1]]
three_d= np.asarray(fi)
three_d_ones=np.concatenate((three_d, one_arr1.transpose()),axis=1)
array_bias1=three_d_ones.transpose()
for i in range(500):
    truth1 = 0
    for j in range(33):
        a = round(sum(W1[b]*array_bias1[:,j]),2)
        #print(a)

        if a>0:

            a=1
        else:
            a=-1
        if (y[0,j]-a)==0:
            truth1=truth1+1
        W1.append(W1[b]+(0.5*c*(y[0,j]-a)*array_bias1[:,j]))
        b=b+1
```

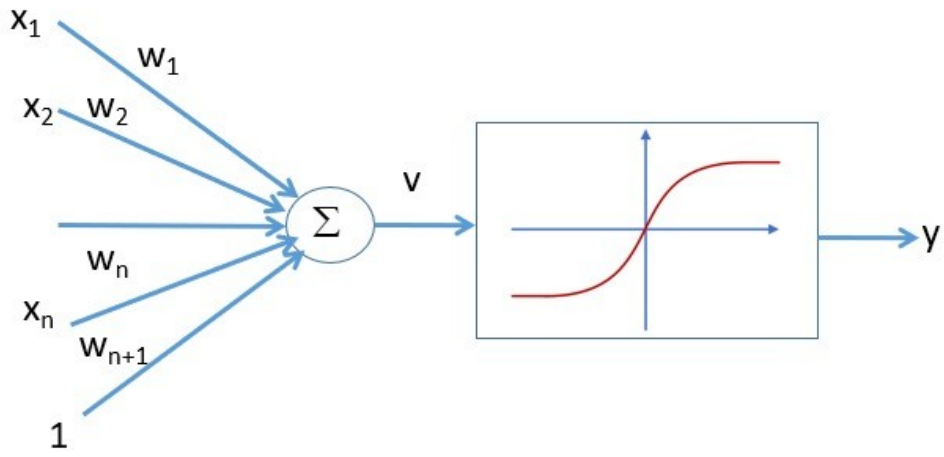
Rosenblatt'ın Genlikte Ayrık Algılayıcısını kullanarak sınıflandırıyoruz.

ii) Giriş katmanında doğrudan girişlere bağlı olan(x1-xn) bir yapıdır.

Birinci katman sabit bir fonksiyon kullanarak kümemizi lineer ayrıştırılabilir hale getirebilmek için kullandığımız yapıdır. [(x1)^2, (√ 2).x1.(x2)^2] fonksiyonunu boyut arttırmak için kullandık.

Çıkış katmanı ise oluşturduğumuz lineer ayrıştırılabilir kümeyle bağlantı ağırlıklarını belirleyen tek bir nörondur.

ADALİNE



Temel olarak Perceptron'a benzemekle beraber Perceptron'un Continuos halidir. Skaler v'ye kadar Perceptron'la aynıdır. Daha sonra aktivasyon fonksiyonu eklenir. -1 ve 1 arasında tanh(a*v) 0-1 arasında 1/(1+e^(-a*v) aktivasyon fonksiyonu kullanılır. Böylece daha ayrıntılı bir ayrıştırma gerçekleştirilir.

3)

```
z=30
#a_1=np.random.rand(1,z)
#a_2=np.random.rand(1,z)
#np.save('arrx_1', a_1)
#np.save('arrx_2', a_2)
x_1= np.load('arrx_1.npy')
x_2= np.load('arrx_2.npy')
x=np.concatenate((x_1,x_2),axis=0)
one_arr = np.ones(z).reshape(1,z)
```

(x_1) 0-1 arasında , (x_2) 0-1 arasında olmak üzere 30'ar tane değer atadık.
Sonra x_1 ile x_2'yi birleştirdik ve x kümemizi oluşturduk.

```
yd=[]
for i in range(z):
    yd.append(3*x[0,i]+2*math.cos(x[1,i]))

max_yd=max(yd)
min_yd=min(yd)
x_bias=np.concatenate((x,one_arr))
yd=(yd-min_yd)/(max_yd-min_yd)
```

Yd kümemizi oluşturup 3(x1)*2cos(x2) formuluyle elde ettiğimiz değerleri yd kümesine atadık.
Yd'yi normalize ederek 0-1 arasına yerleştirdik.

```
y=np.zeros([1,30])
a=1
b=0
for j in range(150):
    truth=0
    for i in range(z):
        v=round(sum(W[b]*x_bias[:,i]),2)
        y[0,i]=1/(1+math.exp(-a*v))
        f=(math.exp(-a*v)*a)/pow((1+math.exp(-a*v)),2)
```

e=yd-y ve E=(1/2)*(e^2) olmak üzere minimum hatayı bulmak için türevini alıyoruz.
Öğrenme kuralımız ==> (yd-y)*(1/(1+e^(-a*v))'*(x^T)
y=1/(1+e^(/a*v) ve f=(e^(-a*v)*a/(1+e^(-a*v))^2 eğitim kümemizi bu öğrenim kurallarına göre eğittik.

```
#a_3=np.random.rand(1,z)
#a_4=np.random.rand(1,z)
#np.save('arrx_3', a_3)
#np.save('arrx_4', a_4)
x_3= np.load('arrx_3.npy')
x_4= np.load('arrx_4.npy')
x_test=np.concatenate((x_3,x_4),axis=0)
one_arr = np.ones(z).reshape(1,z)
yd_test=[]
```

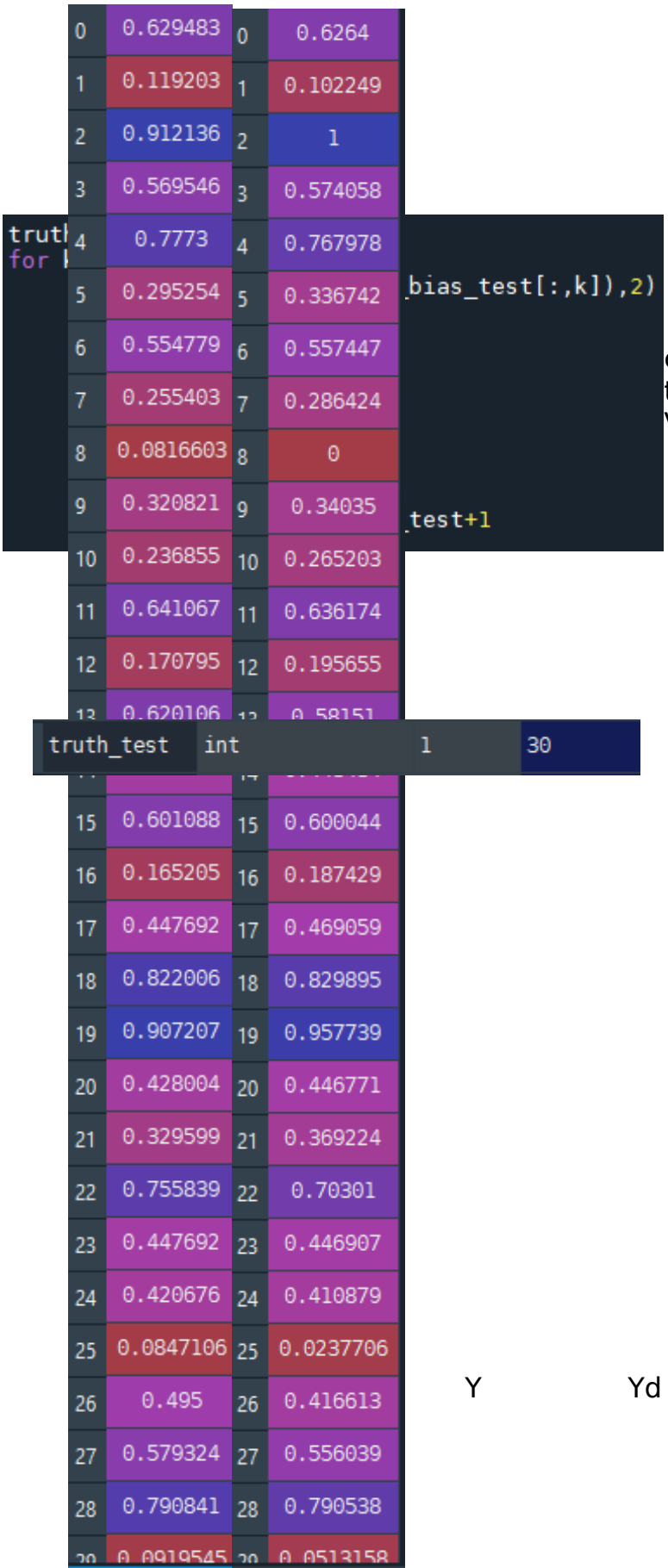
Test kümemizi oluşturduk. Test kümemizdeki x_3 ve x_4'ü test fonksiyonuna soktuk.

```
yd_test=[]
for i in range(z):
    yd_test.append(3*x_test[0,i]+2*math.cos(x_test[1,i]))
max_yd_test=max(yd_test)
min_yd_test=min(yd_test)
x_bias_test=np.concatenate((x_test,one_arr))
yd_test=(yd_test-min_yd_test)/(max_yd_test-min_yd_test)
```

Test fonksiyonuna sokulan x_3 ve x_4 'ü yd_test kümesine dahil ettik.
yd_test kümemizi 0-1 arasına normalize ettik.

```
for i in range(z):
    if yd_test[i] <= 0.5:
        yd_test[i]=0
    else:
        yd_test[i]=1
```

Test kümemizi sınıflandırmak için 0.5 ve altını 0 kümesine diğerlerini 1 kümesine dahil ettik.



Ağırlıklarımızla x değerlerimizi çarptık. A 0.5'ten büyükse 1 tanımladık, küçükse 0'a tanımladık. Ve test etmiş olduk.

Son iterasyondaki y değerimiz yd değerimize yakın bir sonuç verdi.

4) Çok katmanlı perceptron kullanacağız.