YAPAY SİNİR AĞLARINA GİRİŞ

DERSİ ÖDEV2

Öğrenci İsmi ve Numarası: İlker Bedir - 16011036

Ders Sorumlusu: Doç.Dr. SIRMA YAVUZ

Teslim Tarihi: 28.11.2020

Ödev Konusu: Delta Learning Rule ile Karakter Tanıma

1-A)PERCEPTRON ALGORİTMASI NEDİR?

Perceptron, girdi verilerindeki özellikleri tespit etmek için hesaplamalar yapan bir *yapay sinir ağı* birimidir.

Perceptron Öğrenme Algoritması, 1957 yılında *Frank Rosenblatt* tarafından Cornell Havacılık Labaratuarında icat edilmiştir. Algoritma bize, sinir ağındaki ağırlık katsayılarını en uygun şekilde kullanmamızda yardımcı olur. Optimize edilmiş ağırlık katsayıları bulunduktan sonra, girdi özellikleri bir sinirin uyarılıp uyarılmayacağını belirlemek için bu ağırlıklar ile çarpılır, ve bunun neticesinde bir takım çıkarımlar yapılır. Doğrusal bir sınıflandırma algoritmasıdır.

1-B)DELTA RULE ALGORİTMASI NEDİR?

Perceptrona aksine delta rule kuralında ağırlık güncelleme fonksiyonumuz farklı çalışmaktadır.Aşağıdaki fonksiyonlarda daha detaylı şekilde anlatılmıştır.

2)OLUŞTURDUĞUM KOD VE ALGORİTMASI

Öncelikle kodumuz Python yazılım dilinde yazılmıştır. Verilen girdiler, Python'ın open fonksiyonu ile alınmış olup Numpy kütüphanesi Concetenate fonksiyonu ile birleştirilmiştir.

Verilen girdilere baktığımızda ise, girdilerin ilk hali Bipolar girdihalinde olup binary girdi için ise bizim çevirmemiz istenmiştir.

Binary'e çevirme fonksiyonum ise, bipolar verilerdeki –1 değerleri 0 yaparak yazılmıştır. Girdiler ile ilgili kodlar şekil 1a,1b,1c de gösterilmiştir .

```
def bipolar_input(fileName):
    path = 'liste/'
    arr = np.zeros(63)
    with open(path + fileName, 'r') as filestream:
        for line in filestream:
            arr = np.array(line.split(","))
            arr = arr.astype(np.int)
    return arr
```

1a. girdilerin dosyadan alınması

```python
# okuduktan sonra hepsini bir tabloda birleştirme
def tablo_yapma():
    arrA = bipolar_input("Font_1_A.txt")
    arrB = bipolar_input("Font_1_B.txt")
    arrC = bipolar_input("Font_1_C.txt")
    arrD = bipolar_input("Font_1_D.txt")
    arrE = bipolar_input("Font_1_E.txt")
    arrK = bipolar_input("Font_1_K.txt")
    arrJ = bipolar_input("Font_1_J.txt")
    arrA2 = bipolar_input("Font_2_A.txt")
    arrB2 = bipolar_input("Font_2_B.txt")
    arrC2 = bipolar_input("Font_2_C.txt")
    arrD2 = bipolar_input("Font_2_D.txt")
    arrE2 = bipolar_input("Font_2_E.txt")
    arrK2 = bipolar_input("Font_2_K.txt")
    arrJ2 = bipolar_input("Font_2_J.txt")
    arrA3 = bipolar_input("Font_3_A.txt")
    arrB3 = bipolar_input("Font_3_B.txt")
    arrC3 = bipolar_input("Font_3_C.txt")
    arrD3 = bipolar_input("Font_3_D.txt")
    arrE3 = bipolar_input("Font_3_E.txt")
    arrK3 = bipolar_input("Font_3_K.txt")
    arrJ3 = bipolar_input("Font_3_J.txt")
    result = np.concatenate((arrA, arrB, arrC, arrD, arrE, arrJ, arrK, arrA2, arrB2, arrC2, arrD2, arrE2, arrJ2, arrK2,
                             arrA3, arrB3, arrC3, arrD3, arrE3, arrJ3, arrK3), axis=0)
    result = np.reshape(result, (-1, 63))
    return result
```

1a.alınan girdileri tabloya aktarılması

```python
def binary_table(veri):
    for i in range(0, 21):
        for j in range(0, 63):
            if(veri[i][j] == -1):
                veri[i][j] = 0
    return veri
```

1b. bipolar tabloyu binary haline getirme

Girdileri düzenledikten sonraki adım ise, girdi tablosunu eğitmektir. Bu algoritma için ise dışardan girilen parametrelere ihtiyaç duyulmuştur. Bu alt fonksiyonun parametreleri ve parametre işlevleri şunlardır:

Characters; girdi verilerini tutan tablodur.

Weights; eğitmek için gerekli vektörü temsil eder, başlangıçtaki değerleri 0 dır.

Numof_epochs; modeldeki verileri kaç adım eğiteceğimizi belirtir.

Learning rate; ağırlıkları değiştirirken, değişim hassasiyetini belirleyen parametredir.

Threshold; girdi ve weights dizileri çarpıldığında sonucun karşılaştırıldığı bir eşik değeridir.

Bipolar Flag; Girdinin bipolar olup olmadığını belirtir.

Fonksiyonumuz algoritması ise, öncelik bir girdi verisi için her output verisinin ağırlık vektörü, threshold ile bipolar flag birlikte predict adı verilen girdilerin out değerini veren bir fonksiyona sokulmuştur. Bu fonksiyonun yapısı, girdi dizisinin elemanları ile ağırlıklık dizisi girdi dizisin aynı indisli elemanlarını çarpıp, çarpma işlemindeki sonuçları birbirine eklediğimizde bir aktivasyon değeri elde edilmektedir. Bu aktivasyon değerini, değerini threshold olan bir eşik değerinden geçirdiğimizde out değeri, bipolar veri için (-1,0,1) , binary için ise(0,1) olmaktadır. Predict fonksiyonu bize out sonuçlarını verir.Bu fonksiyonla ilgili veriler şöyledir.

$$f(\mathbf{x}) = \begin{cases} 1 & \text{if } \mathbf{w} \cdot \mathbf{x} + b > 0, \\ 0 & \text{otherwise} \end{cases}$$

Binary girdi ve threshold=0 için out formül.

$$f(x) = 1 \; if \; w.x + b > threshold f(x) = \begin{cases} 1 \rightarrow wx + b > th \\ 0 \rightarrow -\text{th} < wx + b < th \\ -1 \rightarrow wx + b < -th \end{cases}$$

Bipolar girdi için out formülü.

Predict fonksiyonundan dönen out değerini target değeri ile kıyaslamak için ise veri tablosundaki değerlerin indisinin 7'ye göre modunu aldığımız da hangi out ait olduğunu bulabiliriz. Böylelik target ve out değerini kıyaslayabiliriz.

Kıyaslamada ise, sıradaki girdi için sıra değeri mod 7 'e göre değeri ile output nöronun sıra değeri arasında karşılaştırma yapılır. Yapılan karşılaştırma eşit ise o girdi aslında nöronumuzun sınıfındadır. Eğer out ile target değerimiz 1 ise öğrenmiş olur. Ancak farklı ise sonuç burdan bir hata alırız. Eğer sıra değerler farklı ise target ve out değerlerimiz -1 veya 0 olmalıdır, eğer farklı ise bir hata alırız.

Fonksiyonumuzun sonuncu adımında ise, ağırlık ve bias değerleri güncelleme işlemi yapılmaktadır. Delta rule kuralı için güncelleme formülü ise aşağıdaki gibidir;

```
# 1: delta rule
elif update_function == 1:
    if j == (i % 7):  # only true for A->A, B->B ...
        error = 1 - prediction
    elif bipolar_flag == True:
        error = -1 - prediction
    else:
        error = 0 - prediction
    totalError += abs(error)
    inputIndex = 0
    # range(0,63) | range(63,63*2)
    for k in range(0, 63):
        # update weights
        weights[j][k] += learning_rate * \
            error * inputs[inputIndex]
        inputIndex += 1
```

2-a) Delta kuralı ile ağırlık güncelleme

Perceptron algoritmamızda ise ağırlık güncelleme aşağıdaki resimdeki gibi verilmiştir.

```
    # 0: perceptron rule
    if update_function == 0:
        if j == (i % 7):  # only true for A->A, B->B ..
            target = 1
        else:
            if bipolar_flag == True:
                target = -1
            else:
                target = 0
        inputIndex = 0
        if prediction != target:
            # range(0,63) | range(63,63*2)
            for k in range(0, 63):

                # update weights
                weights[j][k] += learning_rate * \
                    target * inputs[inputIndex]
                inputIndex += 1
```

2-b)Perceptron ile ağırlık güncelleme

```
def trainWeights(characters, weights, numof_epochs, learning_rate, threshold, bipolar_flag, update_function):
    log_accuracy = [0.0] * numof_epochs

    for epoch in range(numof_epochs):
        totalError = 0.0
        for i in range(len(characters)):
            inputs = characters[i]
            for j in range(7):
                prediction = predict(inputs, weights, j,
                                     threshold, bipolar_flag)

                # 0: perceptron rule
                if update_function == 0:
                    if j == (i % 7):  # only true for A->A, B->B ...
                        target = 1
                    else:
                        if bipolar_flag == True:
                            target = -1
                        else:
                            target = 0
                    inputIndex = 0
                    if prediction != target:
                        # range(0,63) | range(63,63*2)
                        for k in range(0, 63):

                            # update weights
                            weights[j][k] += learning_rate * \
                                target * inputs[inputIndex]
                            inputIndex += 1
```

```
            # 1: delta rule
            elif update_function == 1:
                if j == (i % 7):  # only true for A->A, B->B ...
                    error = 1 - prediction
                elif bipolar_flag == True:
                    error = -1 - prediction
                else:
                    error = 0 - prediction
                totalError += abs(error)
                inputIndex = 0
                # range(0,63) | range(63,63*2)
                for k in range(0, 63):
                    # update weights
                    weights[j][k] += learning_rate * \
                        error * inputs[inputIndex]
                    inputIndex += 1

    log_accuracy[epoch] = accuracy(
        characters, weights, threshold, bipolar_flag)
    print('epoch:', epoch + 1, 'current accuracy:', log_accuracy[epoch])

    if log_accuracy[epoch] == 1.0:  # if we reach 100% success rate we can stop
        count = 0  # resize the log_accuracy list with removing zeros
        for i in range(len(log_accuracy)):
            if log_accuracy[i] != 0.0:
                count += 1
        print('count:', count)
        log_accuracy_no_zeros = [0.0] * count
        for i in range(count):
            log_accuracy_no_zeros[i] = log_accuracy[i]
        return weights, log_accuracy_no_zeros

return weights, log_accuracy
```

VERİ EĞİTİM FONKSİYONU

```
def predict(inputs, weights, bias, weightsIndex, threshold, bipolar_flag):
    activation = bias
    for i in range(len(inputs)):
        activation += weights[weightsIndex][i] * inputs[i]
    if (activation >= threshold):
        return 1.0
    elif bipolar_flag == True:
        if activation < (-1) * threshold:
            return -1
    return 0.0
```

Tahmin Fonksiyonu

```python
def accuracy(characters, weights, threshold, bipolar_flag):

    total_correct_predict = 0
    for i in range(len(characters)):
        for j in range(0, 7):  # num of output neurons
            result = predict(characters[i], weights,
                             j, threshold, bipolar_flag)
            if j == (i % 7):  # means same letter -> 1: correct ** 0, -1: wrong answers
                if result == 1:
                    total_correct_predict += 1
            elif bipolar_flag == True:
                if result != 1:
                    total_correct_predict += 1
            else:
                if result == 0:
                    total_correct_predict += 1
    return total_correct_predict / (7 * len(characters))
```

Veri eğitim Accuracy fonksiyonum

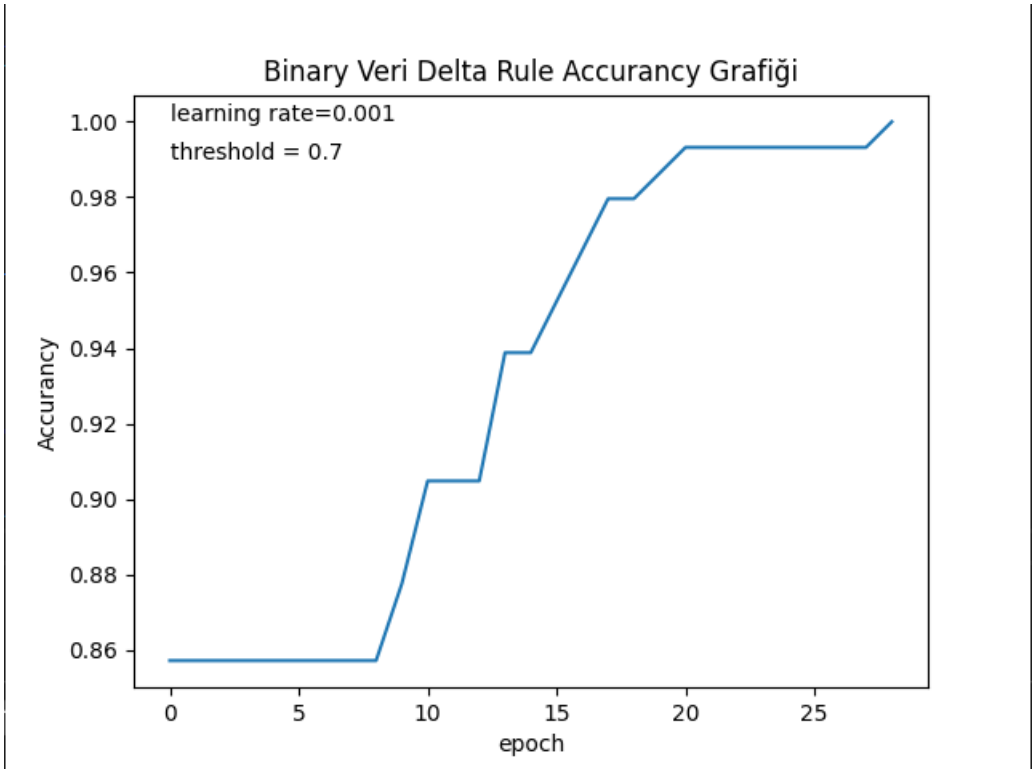
Test fonksiyonunda ise, veriyi eğittikten sonra oluşan sonraki ağırlık değerleri, girdi verilerimiz ile çarpılıp toplandıktan sonra bir out değeri oluşmaktadır. Oluşan out değeri threshold ile kıyasladığımız zaman bir net değer ortaya çıkar. Bu değer bir ise o girdinin sıra değerinin mod 7'e göre değeri ile output nöron değeri eşit olmalıdır. Eğer eksi bir ise o girdinin sıra değerlerinin farklı olması gerekir. Bunlardan başka bir durum ise hata kabul edilir.

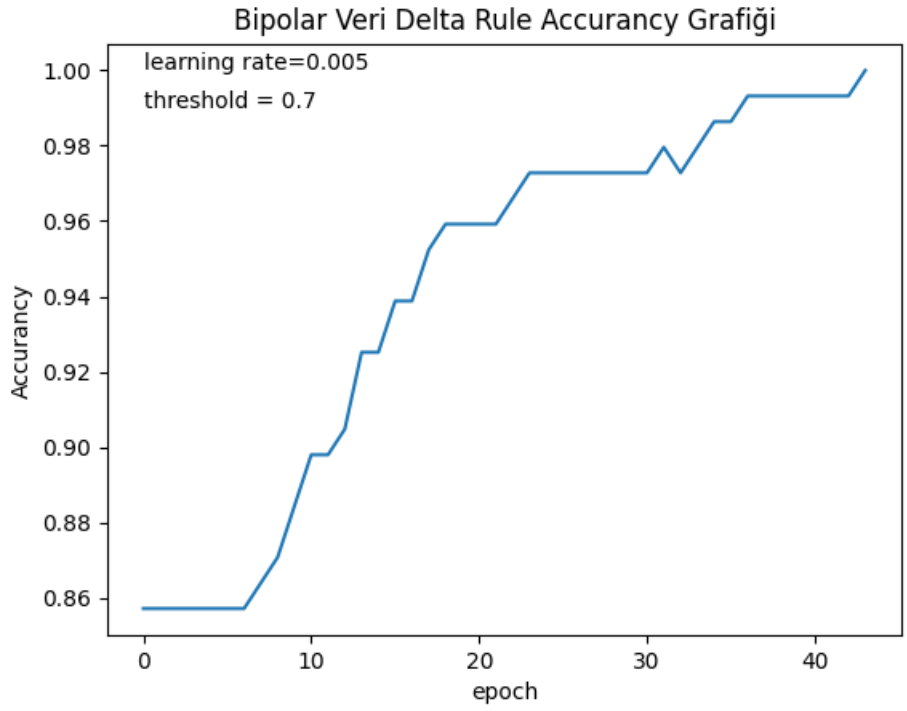
```python
def test(characters, weights, hata_out, hata_input, threshold, bipolar_flag):
    if bipolar_flag == False:
        characters = binary_table(characters)
    error = 0
    for j in range(0, 21):
        for k in range(0, 7):
            c = 0
            out = 0
            for i in range(0, 63):
                out = out + characters[j][i]*weights[k][i]
            if(out > threshold and k == (j % 7)):
                out = 1
            elif(out < threshold and k != (j % 7)):
                out = -1
            else:
                hata_input.append(j)
                hata_out.append(k)
                error = error+1
    return error, hata_out, hata_input
```

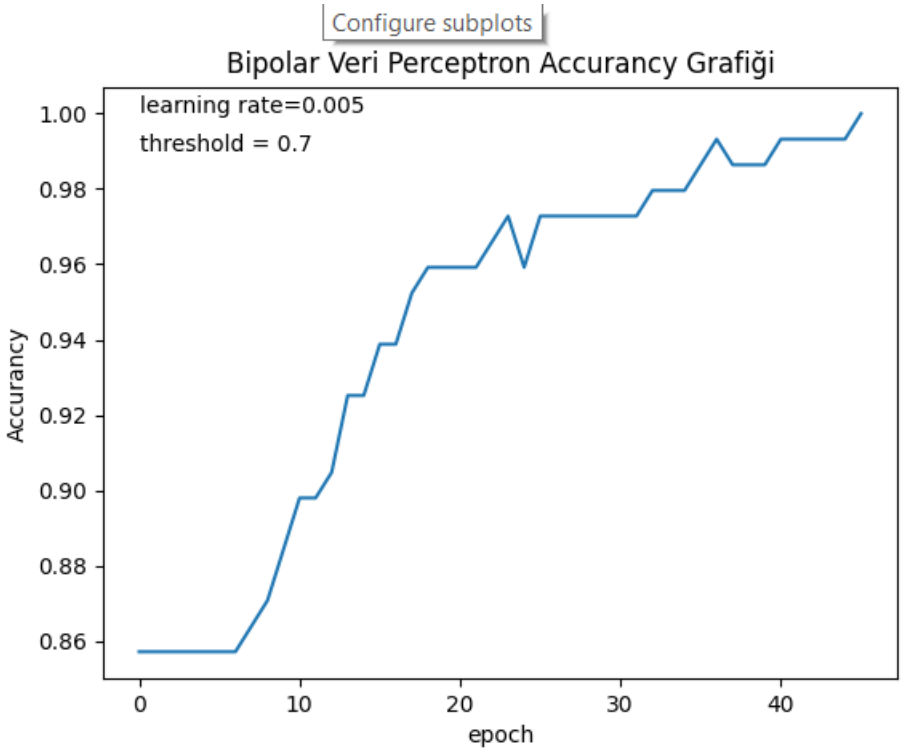
TEST FONKSİYON

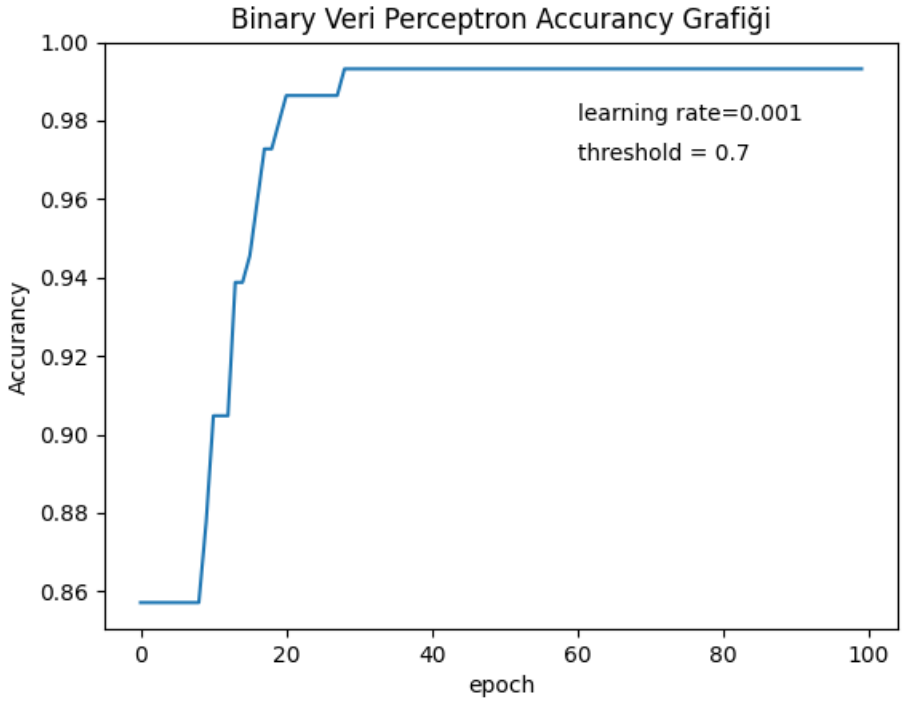
ANALİZ KISMI

Analiz kısmına hata kısımlarından başlayacak olursak, hataları veriyi eğitirken ardından ise test fonksiyonunda görmekteyiz. Ancak veriyi eğitirken ve test ederken ayrı yerlerde, farklı sayılarda hata alabilir. Aşağıdaki grafiklerde ise veriyi eğitirken her epoch da oluşan başarı oranını sayısını göstermektedir.



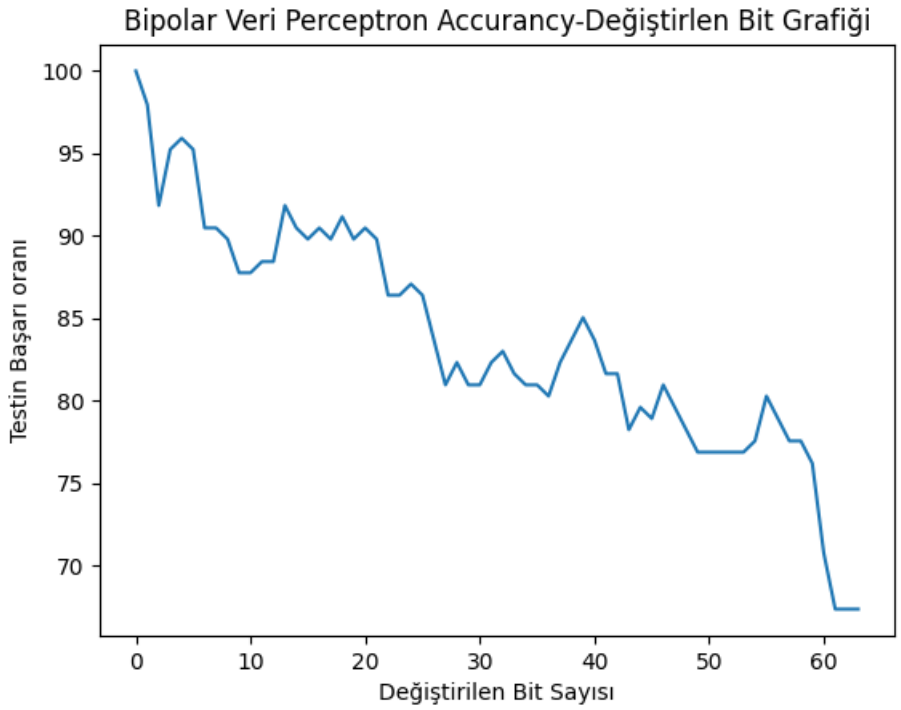


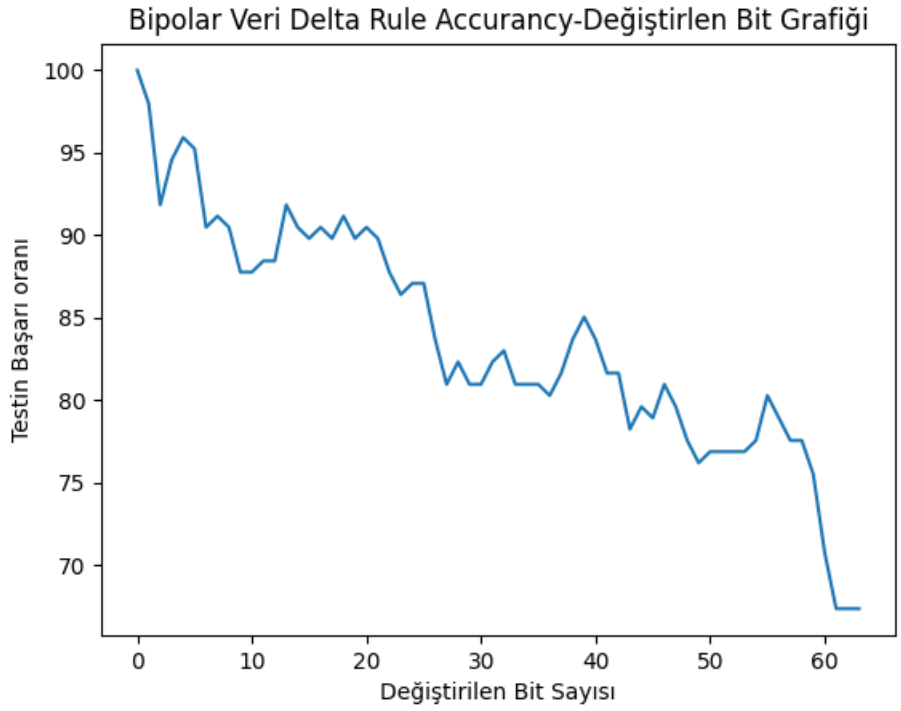


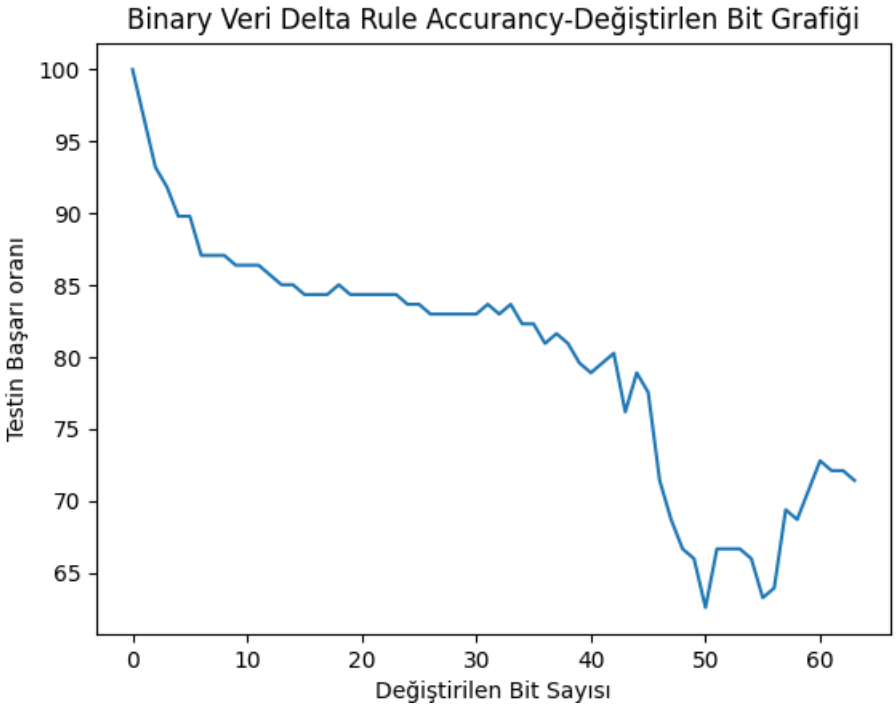


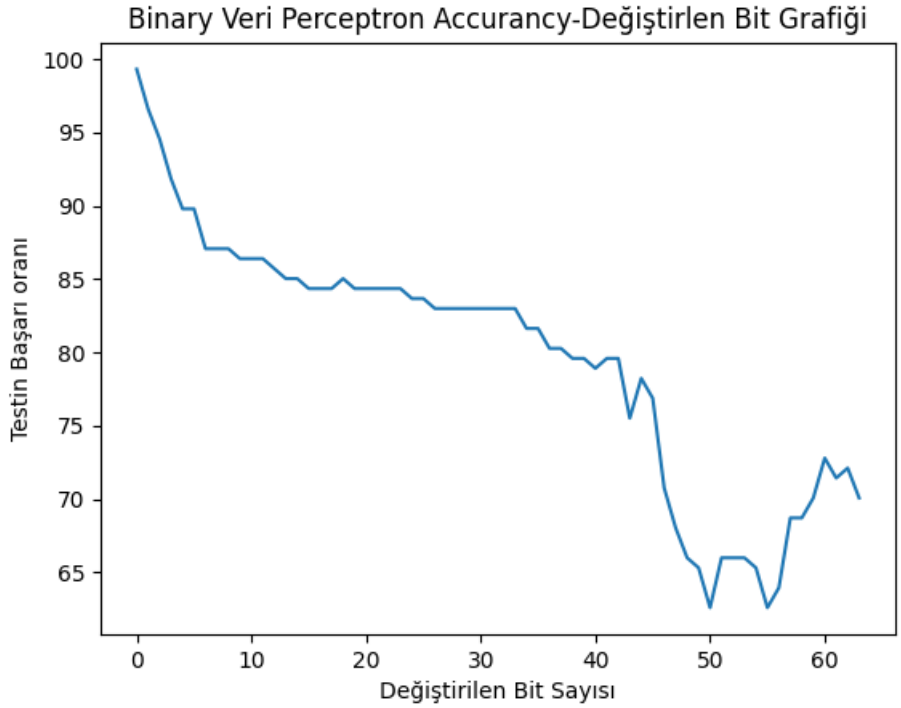
Veriler eğitilirken başarı oranları yukarıdaki gibidir. Delta Rule'de iki inputta da 100% eğitim başarı oranı yakalarken ,perceptron öğrenmesinde bipolar veri için 100% eğitim başarı oranı varken binary veride 100% oranını yakalayamamaktadır.

Oynanmış test verileri için ise başarı grafikleri ise aşağıdaki gibidir.









Bitlerin değişimi indise göre yapılmaktadır.

Yukarıdaki grafiklerden çıkan sonuç ise,bipolar input için veri en düşük %67 olurken,binary tipindeki input için ise %62 ye kadar inmektedir.Bu yüzden bipolar veri daha iyi sonuçlar verebilir.

SİSTEMİN GENEL ÇIKTISI:

```
(.venv) C:\Users\ilker\OneDrive\Masaüstü\test> cd c:\Users\ilker\OneDrive\Masaüstü\test && cmd /C "c:\Users\ilker\OneDrive\Masaüstü\test\.venv\Scripts\python.e
xe c:\Users\ilker\.vscode\extensions\ms-python.python-2020.11.371526539\pythonFiles\lib\python\debugpy\launcher 61187 -- c:\Users\ilker\OneDrive\Masaüstü\test\
16011036_ödev1.py "
accurancy:  [0.8571428571428571, 0.8571428571428571, 0.8571428571428571, 0.8571428571428571, 0.8571428571428571, 0.8571428571428571, 0.8571428571428571, 0.8571
428571428571, 0.8571428571428571, 0.8775510204081632, 0.9047619047619048, 0.9047619047619048, 0.9047619047619048, 0.9387755102040817, 0.9387755102040817, 0.945
5782312925171, 0.9591836734693877, 0.9727891156462585, 0.9727891156462585, 0.9795918367346939, 0.9863945578231292, 0.9863945578231292, 0.9863945578231292, 0.98
63945578231292, 0.9863945578231292, 0.9863945578231292, 0.9863945578231292, 0.9863945578231292, 0.9931972789115646, 0.9931972789115646, 0.9931972789115646, 0.9
931972789115646, 0.9931972789115646, 0.9931972789115646, 0.9931972789115646, 0.9931972789115646, 0.9931972789115646, 0.9931972789115646, 0.9931972789115646, 0.
9931972789115646, 0.9931972789115646, 0.9931972789115646, 0.9931972789115646, 0.9931972789115646, 0.9931972789115646, 0.9931972789115646, 0.9931972789115646, 0
.9931972789115646, 0.9931972789115646, 0.9931972789115646, 0.9931972789115646, 0.9931972789115646, 0.9931972789115646, 0.9931972789115646, 0.9931972789115646,
0.9931972789115646, 0.9931972789115646, 0.9931972789115646, 0.9931972789115646, 0.9931972789115646, 0.9931972789115646, 0.9931972789115646, 0.9931972789115646,
 0.9931972789115646, 0.9931972789115646, 0.9931972789115646, 0.9931972789115646, 0.9931972789115646, 0.9931972789115646, 0.9931972789115646, 0.9931972789115646
, 0.9931972789115646, 0.9931972789115646, 0.9931972789115646, 0.9931972789115646, 0.9931972789115646, 0.9931972789115646, 0.9931972789115646, 0.993197278911564
6, 0.9931972789115646, 0.9931972789115646, 0.9931972789115646, 0.9931972789115646, 0.9931972789115646, 0.9931972789115646, 0.9931972789115646, 0.99319727891156
46, 0.9931972789115646, 0.9931972789115646, 0.9931972789115646, 0.9931972789115646, 0.9931972789115646, 0.9931972789115646, 0.9931972789115646, 0.9931972789115
646, 0.9931972789115646, 0.9931972789115646, 0.9931972789115646, 0.9931972789115646, 0.9931972789115646]
```

HARFLERİ BİRBİRİNE BENZERLİK SAYILARI(HAMING DISTANCE)

Hamming distance algoritması genelde kelimelerin birbirine benzerliği için kullanılır. Ancak burada harflerin benzerliğini bulmak için kullandım. Hamming Distance Formülü:

$$hd = |x_1 - y_2| + \cdots + |x_i - y_i|$$

	A	B	C	D	E	J	K
A	0.0	28.0	29.0	27.0	24.0	23.0	19.0
B	28.0	0.0	23.0	11.0	10.0	27.0	21.0
C	29.0	23.0	0.0	26.0	19.0	18.0	28.0
D	27.0	11.0	26.0	0.0	17.0	22.0	20.0
E	24.0	10.0	19.0	17.0	0.0	25.0	13.0
J	23.0	27.0	18.0	22.0	25.0	0.0	28.0
K	19.0	21.0	28.0	20.0	13.0	28.0	0.0

FONT 1 İÇİN BENZERLİK TABLOSU(EN BENZER E İLE B)

	A	B	C	D	E	J	K
A	0.0	38.0	29.0	36.0	33.0	21.0	28.0
B	38.0	0.0	21.0	12.0	9.0	33.0	24.0
C	29.0	21.0	0.0	11.0	20.0	18.0	25.0
D	36.0	12.0	11.0	0.0	15.0	27.0	24.0
E	33.0	9.0	20.0	15.0	0.0	30.0	19.0
J	21.0	33.0	18.0	27.0	30.0	0.0	29.0
K	28.0	24.0	25.0	24.0	19.0	29.0	0.0

FONT 2 İÇİN BENZERLİK TABLOSU(EN BENZER E İLE B)

	A	B	C	D	E	J	K
A	0.0	30.0	34.0	33.0	27.0	31.0	29.0
B	30.0	0.0	26.0	11.0	9.0	29.0	21.0
C	34.0	26.0	0.0	19.0	27.0	17.0	29.0
D	33.0	11.0	19.0	0.0	18.0	24.0	18.0
E	27.0	9.0	27.0	18.0	0.0	28.0	14.0
J	31.0	29.0	17.0	24.0	28.0	0.0	26.0
K	29.0	21.0	29.0	18.0	14.0	26.0	0.0

FONT 3 İÇİN BENZERLİK TABLOSU(EN BENZER E İLE B)

BİPOLAR VERİ İLE BİNARY VERİ ARASINDAKİ FARKLAR

Bipolar veri -1 ve 1 değerlerinden oluşur. Binary veri ise, 0 ve 1 değerlerinden oluşmuştur. Bipolar veri, binary veriden değer aralığı yüksek olduğu için bipolar veri de daha kesin ve daha doğru sonuçlar çıkmaktadır. Ancak binary sonuçlarında ise daha hızlı bir eğitim söz konusudur. Eğer bir veri seçme hakkım olsaydı daha kesin ve daha doğru sonuç için bipolar veriyi tercih ederim.

VERİYİ EĞİTMEK İÇİN KULLANILAN PARAMETLERİNİN ETKİSİ

Threshold değeri genelde verileri bölen bir hiperdüzlem olarak düşünürsek thresholdu üst değerlerinin arasındaki bir değer yaparsak daha iyi sonuçlar alabiliriz.

Learning rate ise, ağırlık değişiminde büyük etkisi vardır ve katsayısıdır. Learning rate ne kadar az olursa algoritmadaki doğruluğa yavaş ulaşılır ancak tam değişim noktaları ve en yakınsak noktalar bulunabilir. Learning rate büyük olursa, sistem daha hızlı çalışır ancak en yakınsak noktayı kaçırabilir.

Epoch sayısı ne kadar çok o kadar fazla adım ve kesinlik olur.

DELTA RULE VS PERCEPTRON

Delta rule algoritması perceptron algoritmasına göre verileri eğitirken binary tipindeki veri içinde daha doğru çalışır. Ayrıca Delta Rule Perceptron algoritmasına göre her iki veri tipi için daha az iterasyon ile doğru sonucu bulur.