Yapay Sinir Ağları
1.Ödev

22.11.2020

Mehmet Şerbetçioğlu -- 040160056

Ahmet Hulusi Tarhan -- 040170738

Soru 1:

Öncelikle bir “DataPoints” obje sınıfı oluşturuldu. Bu sınıfın içinde koordinat, anahtar, veri sınıfı ve verinin kullanılacağı küme bilgileri mevcut. Ek olarak bu bilgilerin değişmesini sağlayacak ve koordinatın vektörel uzunluğunu verecek fonksiyonlar var.

```
15  class DataPoints:
16      #DataPoints obje sınıfında koordinat, anahtar, veri sınıfı ve küme bilgileri mevcut.
17      def __init__(self, Coordinate, Key, DataClass, Domain):
18          self.Coordinate = Coordinate
19          self.Key = Key
20          self.DataClass = DataClass
21          self.Domain = Domain
22
23      #
24      def setCoordinate(self, Coordinate):
25          self.Coordinate = Coordinate
26
27      def setClass(self, DataClass):
28          self.DataClass = DataClass
29
30      def setDomain(self, Domain):
31          self.Domain = Domain
32
33      #Verinin koordinat vektörünün büyüklüğünü veren fonksiyon.
34      def CoordinateMag(self):
35          return np.linalg.norm(self.Coordinate)
```

İlk olarak kullanılacak verilerin saklanacağı yeri, veri sayısını ve verilerden bilgi çekebilecek basit fonksiyonlar tanımlandı.

```
381 __location__ = os.path.realpath(os.path.join(os.getcwd(), os.path.dirname(__file__)))
382 dataFile = os.path.join(__location__, 'Data.pkl')
383 max_epoch = 40
384
385 GetCoordinate = attrgetter('Coordinate')
386 GetKey = attrgetter('Key')
387 GetDataClass = attrgetter('DataClass')
388 GetDomain = attrgetter('Domain')
```

Ardından belirtilen lokasyonda bir veri dosyası var mı diye kontrol edilir. Eğer belirtilen dosya bulunmaktaysa dosyadaki veriler “objList” listesine yüklenir. Ardından bu listedeki verilerin beşinci boyut koordinatları ve anahtarlarıyla yeni bir sözlük meydana getirilir.

```
390 if os.path.exists(dataFile):
391     objList = loadList(dataFile)
392     allDict = newDict(objList)
```

```python
#File lokasyonundaki listeyi çeker.
def loadList(File):
    DataList = []
    with open(File, 'rb') as f:
        DataList = pickle.load(f)
    return DataList
```

```python
#Girilen listeden koordinatın beşinci boyut elemanı ve anahtar bilgilerini çeken bir sözlük oluşturur.
def newDict(List):
    d = {}
    for obj in List:
        a = GetCoordinate(obj)
        k = GetKey(obj)
        d[k] = a[4]
    #Oluşturulan sözlük lineer ayrıştırılabilirlik için kullanılacak.
    return d
```

Eğer böyle bir veri yoksa, belirtilen veri sayısında yeni veri oluşturulur. Bu verilerin koordinatları noktadan sonra dört basamaklı, (-5,5) aralığında rastgele seçilir. Oluşturulan veriler “objList” listesine kaydedilir. Bu listeyle yeni bir sözlük oluşturulur. Daha sonra “distClass” komutuyla listedeki veriler yarı yarıya beşinci boyut elemanlarının büyüklüğüne göre iki sınıfa ayırılır. Ardından eğitim ve test kümeleri dağıtılır ve son olarak veri kaydedilir. “distClass” fonksiyonunda dördüncü argümanda False kullanılırsa veriler rastgele sınıflara dağıtılıp lineer ayrıştırılabilir olması kesinleşmeyecektir.

```python
else:
    objList = newData(max_epoch)
    allDict = newDict(objList)
    distClass(max_epoch, allDict, objList, True)
    distDomain(objList)
    saveData(objList, dataFile)
```

```python
#"ep newData: newData boyutlu koordinat oluşturur.
def newData(epoch):
    DataList = []
    for i in range(epoch):
        #Koordinatlar (-1,1) aralığında noktadan sonra 4 basamaklı sayılardan oluşmakta. Uniform dağılımla seçiliyorlar.
        Coord = 10*np.random.random_sample(5,) - 5
        Coord = np.around(Coord, decimals = 4)
        #Kordinatlarla yeni bir veri listesi oluşturur. Yeni verilerin anahtarları iterasyon sayısına eşit olacak ve birbirinden farklı olacaktır.
        DataList.append(DataPoints(Coord, i, 0, "0"))
    return DataList
```

```python
#"epoch" kez yeni beş boyutlu koordinat oluşturur.
def newData(epoch):
    DataList = []
    for i in range(epoch): ...
    return DataList
```

```python
#Gir  newDict: newDict dinatın beşinci boyut elemanı ve anahtar bilgilerini çeken bir sözlük oluşturur.
def newDict(List):
    d = {}
    for obj in List:
        a = GetCoordinate(obj)
        k = GetKey(obj)
        d[k] = a[4]
    #Oluşturulan sözlük lineer ayrıştırılabilirlik için kullanılacak.
    return d
```

```python
#Veri sınıflarını dağıtan fonksiyon
def distClass(epoch, Dict, List, Lin):
    #Lineerizasyon gerçekleşmesi için Lin argümanı doğru olmalıdır.
    if Lin == True:
        totalDict = Dict
        minDict = {}
        #Verilen sözlüğü kullanarak verilerin koordinatlarının beşinci boyut elemanlarını büyüklüklerine göre ikiye ayırır.
        #Bu lineer ayrıştırılmanın kesinlikle mümkün olmasını sağlar.
        for i in range(round(epoch/2)):
            key_min = min(totalDict.keys(), key=(lambda k: totalDict[k]))
            minDict[key_min] = totalDict[key_min]
            del totalDict[key_min]
        for obj in List:
            k = GetKey(obj)
            #Minimum sözlüğündeki değerlerin veri sınıfını 1'e, diğerlerini -1'e atar.
            if k in minDict.keys():
                obj.setClass(1)
            else:
                obj.setClass(-1)

    #Lin argümanı doğru değilse girilen liste çekilir, karıştırılır ve sınıflara dağıtılır.
    elif Lin == False:
        totalDict = Dict
        keys =  list(totalDict.keys())
        random.shuffle(keys)
        Dict1 = dict(list(totalDict.items())[len(totalDict)//2:])
        Dict2 = dict(list(totalDict.items())[:len(totalDict)//2])
        for obj in List:
            k = GetKey(obj)
            #Değerleri rastgele bir şekilde farklı veri sınıflarına atar.
            #Bununla birlikte lineer ayrıştırılabilir bir veri elde etme ihtimali fazlasıyla düşük olacaktır.
            if k in Dict1.keys():
                obj.setClass(1)
            elif k in Dict2.keys():
                obj.setClass(-1)
            else:
                print("None data class detected.")
```

```python
#Data dosyasını File lokasyonuna kaydeder.
def saveData(Data, File):
    with open(File, 'wb') as f:
        pickle.dump(Data, f)
```

Bu şekilde veriler oluşturulup sınıflara ve kümelere dağıtılmış olur. Eğitimi başlatmak için w1 = [1, 1, 1, 1, 1, 1] ağırlığı seçelim. Öncelikle daha önce eğitilmiş ve test kümelerini başarıyla sınıflandırmış DomainEx1 kümesiyle test yapalım.

```
401  #1'lerden oluşan bir ağırlık vektörü.
402  w1 = [1, 1, 1, 1, 1, 1]
403
404  #Önceden atanmış bir test-eğitim kümesi.
405  #w1 = [1, 1, 1, 1, 1, 1] için 22 iterasyonda eğitilmekte ve son ağırlık w = [1.5964 0.8325 2.1872 -1.2123 -21.1314 23].
406  domainFile1 = os.path.join(__location__, 'DomainEx1.pkl')
407  domainFile2 = os.path.join(__location__, 'DomainEx2.pkl')
408  testnum = 1
409  #objList listesindeki verileri w1 = [1 1 1 1 1 1] ilk ağırlığı ve c = 1 öğrenme hızı ile, maksimum 80 iterasyon sürecek şekilde eğiten fonksiyon.
410  #DomainEx1.pkl dosyasındaki eğitim-test kümesi dağılımlarını çekip verilere işler.
411  print("\n\nTest number:", testnum, "_"*150, "\n")
412  testnum += 1
413  insertDomain(objList, domainFile1)
414  extractDomain(objList, domainFile2)
415  Inf = process(objList, w1, 1, max_epoch*2, domainFile2)
416  print("_"*165, "\n")
```

İlk olarak w1, DomainEx1 dosya lokasyonu ve DomainEx2 dosya lokasyonu tanımlanır. Burada DomainEx2 dosyasının amacı DomainEx1in değişmemesidir. DomainEx2, DomainEx1i kopyalar ve eğitim DomainEx2 ile başlar. Kümeler anahtarlarıyla birlikte kaydedilip tekrar yüklenirken anahtarları kullanılarak verilerle eşlenirler.

```
268  #Verilerin küme bilgisini "domainFile" lokasyonundan çeker.
269  def insertDomain(List, domainFile):
270      for obj in List:
271          k = GetKey(obj)
272          domainId = loadDict(domainFile)
273          obj.setDomain(domainId[k])
```

```
258  #Verilerin küme bilgisini "domainFile" lokasyonunda saklar.
259  def extractDomain(List, domainFile):
260      domainId = {}
261      for obj in List:
262          k = GetKey(obj)
263          dom = GetDomain(obj)
264          domainId[k] = dom
265      saveData(domainId, domainFile)
266      return domainId
```

Daha sonra işlemlerimiz başlar. Öncelikle bazı değişkenler tanımlanır. Fonksiyonun argümanında bir veri kümesi olup olmadığına bakılır. Eğer varsa ve veri dosyası bulunuyorsa dosyadaki kümeler kullanılır. Argümanda belirtilip dosyası bulunmuyorsa kümeler dağıtılıp yeni dosya oluşturulur. Eğer argüman “None” olarak girilmişse sadece kümeleri dağıtır, dosya yüklemez ve oluşturmaz. Kümeler ilk seferde rastgele bir şekilde dağıtılır.

```python
def process(objList, w, c, maxiter, domainFile):
    i = 0
    failedTestCount = 0
    totalEdError = 0
    wantedInfo = []
    w = np.array(w)
    #Önceden belirli bir eğitim-test kümesi dağılımı yapar. Yukarıda belirtilen domainFile1'deki eğitim-test kümesini çeker.
    if domainFile != None:
        if os.path.exists(domainFile):
            insertDomain(objList, domainFile)
        else:
            distDomain(objList)
            extractDomain(domainFile)
    #Eğer önceden belirli bir eğitim-test kümesi yoksa, girilen datayı dağıtır. Bu sayede kümeleri belirtilmeyen her veri rastgele kümelere dağıtılır.
    else:
        distDomain(objList)
```

```python
#Eğitim ve test kümelerine rastgele dağıtım gerçekleştirir.
def distDomain(List):
    shuffle = random.sample(range(0, len(List)), 40)
    testArr = np.split(shuffle, [15])[0]
    educationArr = np.split(shuffle, [15])[1]
    for obj in List:
        if obj.Key in testArr:
            obj.setDomain("test")
        elif obj.Key in educationArr:
            obj.setDomain("education")
        else:
            continue
```

Daha sonra eğitim başlatılır. Eğitim sonucu ilk ağırlık, iterasyon sayısı, son ağırlık ve iterasyondaki değişim sayısının bulunduğu bir liste alınır. Eğitimdeki son ağırlık, bir sonraki eğitimde ilk ağırlık olmak üzere güncellenir. Eğer iterasyon sayısı -1'se, yani iterasyon sayısı verilen maksimum iterasyon sayısını geçmişse eğitim başarısız sayılır. Daha sonra eğitimle elde edilen ağırlık test aşamasına girer. Buradan testte gerçekleşen hata miktarı alınır. Eğer hiç hata alınmamışsa döngüden çıkılır ve eğitim tamamlanır. Fakat test aşamasında hata görüldüyse test ve eğitim kümeleri dengelenir, başarısız test sayısı 1 artırılır. Eğer bu işlem "maxiter" sayısından fazla kez gerçekleşirse eğitim durdurulup başarısız sayılır.

```python
#Eğitim ve test işlemlerinin gerçekleştiği fonksiyon
def process(objList, w, c, maxiter, domainFile):
    i = 0
    failedTestCount = 0
    totalEdError = 0
    wantedInfo = []
    w = np.array(w)
    #Önceden belirli bir eğitim-test kümesi dağılımı yapar. Yukarıda belirtilen domainFile1'deki eğitim-test kümesini çeker.
    if domainFile != None:···
    #Eğer önceden belirli bir eğitim-test kümesi yoksa, girilen datayı dağıtır. Bu sayede kümeleri belirtilmeyen her veri rastgele kümelere dağıtılır.
    else:···

    while i < 1:
        #Eğitim ve test kümelerinin eleman sayılarını gösterir. Debug amaçlı
        """domainTest(objList)"""

        #Eğitimi başlatır ve ilk ağırlık, eğitimin sonlandığı iterasyon sayısı, son ağırlığı ve her iterasyon için iterasyondaki hata sayısını verir.
        final = educate(objList, w, c, maxiter)

        #Her iterasyonda toplam hatayı verir. Karşılaştırmada kullanılacak.
        totalEdError = final[3]

        #Verilerin test kümesinden geçememesi durumu için w1 ağırlığını eğitilen ağırlığa eşitler. Sonraki eğitimde ilk ağırlık olarak bu ağırlık kullanılır.
        w = final[2]

        #Eğitim belirlenen iterasyonda sonuçlanmamışsa eğitilemediğine dair hata verir. Bu büyük ihtimalle verinin lineer ayrıştırılamaz olduğunu gösterir.
        if final[1] == -1:
            print("Education failed and likely impossible.")
            break

        #Eğitilen ağırlık test fonksiyonuna girilir. Burada ağırlık değişmez ve dönüt olarak hata veren veri sayısı alınır.
        errorCount = test(objList, final[2])
        print("Tested error count of ", failedTestCount + 1, "th education is ", errorCount)

        #Eğer test sonrası hata alınmadıysa döngüden çıkar.
        if errorCount == 0:
            i += 1

        #Testten geçemediyse veri balanceDomain fonksiyonuna girilir ve test kümesi ve eğitim kümesi birer eleman değişir. Bu değişim iki sınıfın da elemanı olacak şekilde
        #ağırlık koyularak yapılır ancak yine de rastgele bir değişimdir. Ayrıca kümeler değiştiği için iterasyon başına hatayı sıfırlar.
        else:
            balanceDomain(objList)
            failedTestCount += 1
            if failedTestCount > maxiter:
                print("Education failed and likely impossible.")
                break
```

Eğitimi başlatırken veriler, ilk ağırlık, eğitim hızı ve maksimum iterasyon sayısı alınır. Eğer ağırlık geçerli değilse rastgele bir ağırlık atanır. Bunun için getrandomw fonksiyonuyla ortalama değeri 0 ve standart sapması 1 olan normal dağılımlı altı boyutlu bir w vektörü üretilir. İlk ağırlık belirlenir, finaliter değişkeni, iterasyon sayısının maksimumu aşması ihtimaliyle -1'e atanır. Eğitim kümesi elemanları listesi oluşturulur ve eğitim başlar.

```python
#Eğitim fonksiyonu
def educate(List, w, eSpeed, maxiter):
    educateList = []
    final = []
    totalEdError = 0

    #Geçerli bir ağırlık vektörü verilmemişse rastgele bir ağırlık vektörü seçer.
    if w.size != 6:···
    #Eğitime başlanılan ağırlığın çıktısını verir.
    print("First weight:", np.around(w, decimals = 4))
    firstw = w
    finaliter = -1
    #Verilerin içinden eğitim kümesine ait elemanları çeker.
    for obj in List:
        if GetDomain(obj) == "education":
            educateList.append(obj)
```

```python
#Normal dağılımlı rastgele bir ağırlık vekötrü atar.
def getrandomw(mean, scale):
    w = np.random.normal(mean, scale, 6)
    return w
```

Eğitim kümesi “maxiter” kez eğitilir. Bu sayı aşılırsa “process” fonksiyonuna son iterasyon sayısı “-1” olarak döner ve eğitim başarısız sayılır. Öncelikle iterasyondaki değişim sayısı 0’a eşitlenir. Eğitim kümesindeki her veri için bias koordinatı eklenir. Daha sonra w ağırlık vektörü ile v = wT*x işlemi yapılır. Elde edilen v 0’dan büyükse y = 1, küçükse y = -1 aktivasyon fonksiyonu tanımlanır. yd değeri verinin sınıfıyla eşitlenir. Daha sonra w = w + (c/2)*(yd – y)*x perceptron algoritması kullanılarak her veri için ağırlık vektörü güncellenir. y değerinin yd’den farklı çıkması durumunda değişim sayısı 1 artırılır. Bu şekilde tüm eğitim kümesi üzerinden geçildiğinde bir eğitim iterasyonu tamamlanmış olur. Eğer iterasyon sonucunda değişim sayısı 0 bulunursa son iterasyon ve son ağırlık çekilip eğitim durdurulur. “process” fonksiyonuna ilk ağırlık, iterasyon sayısı, son ağırlık ve son iterasyondaki toplam hata bilgileri döner.

```python
def educate(List, w, eSpeed, maxiter):
    educateList = []
    final = []
    totalEdError = 0

    #Geçerli bir ağırlık vektörü verilmemişse rastgele bir ağırlık vektörü seçer.
    if w.size != 6:···
    #Eğitime başlanılan ağırlığın çıktısını verir.
    print("First weight:", np.around(w, decimals = 4))
    firstw = w
    finaliter = -1
    #Verilerin içinden eğitim kümesine ait elemanları çeker.
    for obj in List:
        if GetDomain(obj) == "education":
            educateList.append(obj)

    for i in range(maxiter):
        changeCount = 0
        #Eğitim iterasyonu terasyon döngüsü
        for obj in educateList:
            #Bias terimini verinin koordinatına ekler.
            biasCoord = np.concatenate((GetCoordinate(obj),[1]))

            #Perceptron eğitim algoritması
            v = np.dot(w, biasCoord)
            if v > 0:
                y = 1
            else:
                y = -1
            yd = GetDataClass(obj)
            w = w + (eSpeed/2)*(yd - y)*biasCoord
            w = np.around(w, decimals = 4)

            #Değişim gerçekleştiyse bir iterasyondaki değişim sayısına bir ekler.
            if yd != y:
                changeCount += 1
        totalEdError += changeCount
        #Eğitim sırasında her iterasyondaki hata sayısını verir.
        print("Error count on ", i + 1, "th iteration is: ", changeCount)

        #Eğitim hatasız ise eğitimi durdurur.
        if changeCount == 0:
            finaliter = i+1
            finalw = w
            break
    #Sonuç olarak başlanılan ağırlık vektörü, son eğitimdeki iterasyon sayısı, son ağırlık vektörü ve son iterasyondaki değişim sayısını verir.
    final = [firstw, finaliter, finalw, totalEdError]
    return final
```

Daha sonra güncellenen ağırlık test aşamasına girer. Bu, eğitim fonksiyonuyla aynı karşılaştırmayı yapar fakat ağırlıkları güncellemez. Yalnızca yd = y karşılaştırması yapıp farklı çıkınca errorCount değerini 1 artırır. Test sırasında hata bulunmuşsa balanceDomain fonksiyonu ile eğitim kümesi ve test kümesinden birer eleman yer değiştirir. Bu değişim, rastgele olmasına rağmen iki veri sınıfının test ve eğitim kümesinde bulunan elemanlarının sayılarını yakınlaştıracak şekilde ağırlıklandırılmıştır. process fonksiyonunda errorCount = 0 olmadıkça veya hatalı test ya da eğitim sayısı maxiter değerini aşmadıkça eğitim-test döngüsü devam eder.

```python
def test(List, w):
    #Eğitim fonksiyonuyla hemen hemen aynı algoritmaya sahip. Aralarındaki fark test fonksiyonunun ağırlığı değiştirmeyip yalnızca hata sayısına bakması.
    testList = []
    errorCount = 0
    for obj in List:
        if GetDomain(obj) == "test":
            testList.append(obj)

    for obj in testList:
        biasCoord = np.concatenate((GetCoordinate(obj),[1]))
        v = np.dot(w, biasCoord)
        if v > 0:
            y = 1
        else:
            y = -1
        yd = GetDataClass(obj)
        if yd != y:
            errorCount += 1
    #Test sonucu elde edilen hata sayısını döndürür.
    result = errorCount
    return result

#Normal dağılımlı rastgele bir ağırlık vekötrü atar.
def getrandomw(mean, scale):
    w = np.random.normal(mean, scale, 6)
    return w
```

```python
#Eğitim ve Test kümesi arasında eleman alışverişinde bulunur.
def balanceDomain(List):
    Class1Education = []
    Class2Education = []
    Class1Test = []
    Class2Test = []

    for obj in List:
        if GetDataClass(obj) == 1:
            if GetDomain(obj) == "education":
                Class1Education.append(obj)
            elif GetDomain(obj) == "test":
                Class1Test.append(obj)
            else:
                print("None domain object detected.")
        elif GetDataClass(obj) == -1:
            if GetDomain(obj) == "education":
                Class2Education.append(obj)
            elif GetDomain(obj) == "test":
                Class2Test.append(obj)
            else:
                print("None domain object detected.")
        else:
            print("None class object detected.")

    #(0,1) arasında normal dağılımlı rastgele bir c sayısı seçer. Daha sonra bu sayıyı kümelerin eleman sayılarıyla karşılaştırır.
    c = np.random.normal(0.5, 0.1, 1)

    #Eğer birinci veri sınıfına ait eğitim kümesi elemanlarının sayısı, ikinciye ait olanlardan fazlaysa c'nin aşağıdaki değerden küçük olma olasılığı fazladır.

    if c < len(Class1Education)/(len(Class1Education)+len(Class2Education)):
        #Bu durumda eğitim kümesinden birince sınıfa ait bir eleman çıkarılıp test kümesine aktarılır.
        item = random.choice(Class1Education)
        item.setDomain("test")
        #Küme sayılarının aynı kalması için test kümesinden ikinci sınıfa ait bir eleman çıkarılıp eğitim kümesine aktarılır.
        item = random.choice(Class2Test)
        item.setDomain("education")
        #Aksi takdirde ise aynı işlem sınıflar değiştirilerek gerçekleştirilir.
    else:
        item = random.choice(Class2Education)
        item.setDomain("test")
        item = random.choice(Class1Test)
        item.setDomain("education")
        #Sonuçta ortaya çıkan kümelerin öncekilerden daha iyi olma garantisi yoktur ancak c için verdiğimiz koşul
        #ortaya çıkan kümelerin daha iyi olması ihtimalini yükseltir.
```

---

process fonksiyonu sonuncunda kaç testin başarısız olduğu, hangi iterasyonda eğitimin gerçekleştiği ve son ağırlık vektörü görülür. Başarısız test sayısı, iterasyon başına ortalama hata, iterasyon sayısı, ilk ağırlık, son ağırlık ve ilk ve son ağırlıkların oranı bilgileri elde edilir.

```python
#Eğitim ve test işlemlerinin gerçekleştiği fonksiyon
def process(objList, w, c, maxiter, domainFile):
    i = 0
    failedTestCount = 0
    totalEdError = 0
    wantedInfo = []
    w = np.array(w)
    #Önceden belirli bir eğitim-test kümesi dağılımı yapar. Yukarıda belirtilen domainFile1'deki eğitim-test kümesini çeker.
    if domainFile != None:···
    #Eğer önceden belirli bir eğitim-test kümesi yoksa, girilen datayı dağıtır. Bu sayede kümeleri belirtilmeyen her veri rastgele kümelere dağıtılır.
    else:···

    while i < 1:···
    #Test-Eğitim kümesi değişim sayısı
    print("Failed Test Count:", failedTestCount)

    #Eğitim sonucu elde edilen eğitim-test kümesi elemanları çekilir ve domainFile dosyasına kaydedilir.
    if domainFile != None:
        extractDomain(objList, domainFile)

    #Eğer eğitim tamamlanmamışsa
    if len(final) == 4:
        print("Educated on: ", final[1], "th iteration.")
        print("Final weight: ", final[2])
    iterAmount = final[1]
    avgErrorPerIter = totalEdError/iterAmount
    magFirst = np.linalg.norm(final[0])
    magFinal = np.linalg.norm(final[2])
    magRatio = magFinal/magFirst
    wantedInfo = [failedTestCount, avgErrorPerIter, iterAmount, magFirst, magFinal, magRatio]
    return wantedInfo
```

---

w = [1, 1, 1, 1, 1, 1] için c = 1 eğitim hızıyla daha önce belirtilen DomainEx1 küme dağılımıyla eğitim yapıldığında şu sonuç elde edilir:

```
Test number: 1 ____________________________________________________________________

First weight: [1 1 1 1 1 1]
Error count on  1 th iteration is:  7
Error count on  2 th iteration is:  5
Error count on  3 th iteration is:  3
Error count on  4 th iteration is:  4
Error count on  5 th iteration is:  3
Error count on  6 th iteration is:  3
Error count on  7 th iteration is:  2
Error count on  8 th iteration is:  2
Error count on  9 th iteration is:  4
Error count on  10 th iteration is:  1
Error count on  11 th iteration is:  4
Error count on  12 th iteration is:  3
Error count on  13 th iteration is:  4
Error count on  14 th iteration is:  3
Error count on  15 th iteration is:  2
Error count on  16 th iteration is:  3
Error count on  17 th iteration is:  3
Error count on  18 th iteration is:  3
Error count on  19 th iteration is:  1
Error count on  20 th iteration is:  2
Error count on  21 th iteration is:  2
Error count on  22 th iteration is:  0
Tested error count of  1 th education is  0
Failed Test Count: 0
Educated on:  22 th iteration.
Final weight:  [  1.5964   0.8325   2.1872  -1.2123 -21.1314  23.    ]
```

Bu işlem rastgele bir w ve küme dağılım argümanlarıyla gerçekleştirilince örnek olarak böyle bir sonuç elde edilebilir. balanceDomain fonksiyonu rastgele çalıştığı için, doğru bir test-eğitim kümesi dağılımı kullanmadan aynı argümanlarla aynı sonuçları tekrarlama ihtimali çok düşüktür.

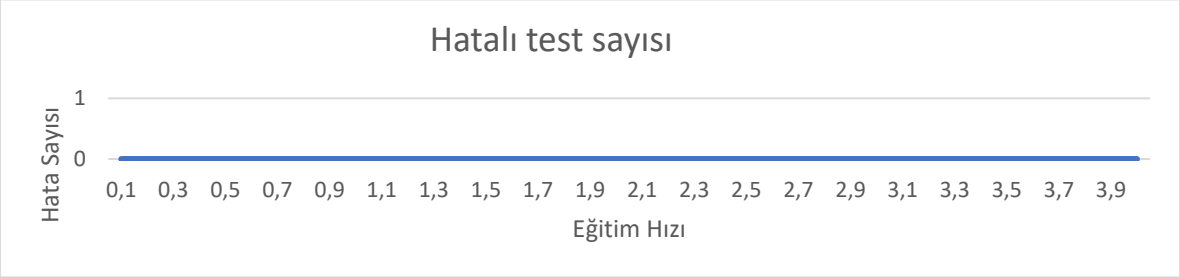
```
413   #objList listesindeki verileri rastgele bir ilk ağırlıkla ve c = 1 öğrenme hızı ile, maksimum 80 iterasyon sürecek şekilde eğiten fonksiyon.
414   #domainFile kısmı "None" olduğu için eğitim sonucunda eğitim-test kümesi dağılımını kaydetmeyecek.
415   """print("\n\nTest number:", testnum, "_"*150, "\n")"""
416   testnum += 1
417   Inf = process(objList, [], 1, max_epoch*2, None)
418   """print("_"*165, "\n")"""
```

```
Test number: 2 ____________________________________________________________________________________________________________

First weight: [-1.6365 -1.4241 -1.0518  1.4459 -1.0647  0.7225]
Error count on  1 th iteration is:  6
Error count on  2 th iteration is:  0
Tested error count of  1 th education is  2
Error count on  1 th iteration is:  0
Tested error count of  2 th education is  2
Error count on  1 th iteration is:  0
Tested error count of  3 th education is  2
Error count on  1 th iteration is:  0
Tested error count of  4 th education is  2
Error count on  1 th iteration is:  1
Error count on  2 th iteration is:  3
Error count on  3 th iteration is:  2
Error count on  4 th iteration is:  2
Error count on  5 th iteration is:  3
Error count on  6 th iteration is:  2
Error count on  7 th iteration is:  2
Error count on  8 th iteration is:  3
Error count on  9 th iteration is:  3
Error count on  10 th iteration is:  1
Error count on  11 th iteration is:  3
Error count on  12 th iteration is:  0
Tested error count of  5 th education is  1
Error count on  1 th iteration is:  4
Error count on  2 th iteration is:  3
Error count on  3 th iteration is:  4
Error count on  4 th iteration is:  1
Error count on  5 th iteration is:  3
Error count on  6 th iteration is:  3
Error count on  7 th iteration is:  4
Error count on  8 th iteration is:  3
Error count on  9 th iteration is:  4
Error count on  10 th iteration is:  3
Error count on  11 th iteration is:  4
Error count on  12 th iteration is:  0
Tested error count of  6 th education is  0
Failed Test Count: 5
Educated on:  12 th iteration.
Final weight:  [  1.1645  -0.4102   0.3968  -0.4178 -19.8912  25.7225]
____________________________________________________________________________________________________________
```

Eğer rastgele seçilen bir ağırlık vektörü ve önceden belirlenmiş bir dağılım kümesi için sadece eğitim hızı değiştirilerek ölçüm alınır. Bu 40 kez gerçekleştirilip elde edeceğimiz sonuçların ortalaması SpeedResult.json dosyasında saklanır.

```
426 domainFile3 = os.path.join(__location__, 'DomainEx3.pkl')
427 domainFile4 = os.path.join(__location__, 'DomainEx4.pkl')
428 # İlk olarak rastgele seçilen bir w = w3 ile aynı veri üzerinde öğrenme hızı değiştirilerek:
429 # Başarısız test sayısı
430 # Son eğitimdeki iterasyon başına düşen ortalama hata sayısı
431 # Son eğitimdeki iterasyon sayısı
432 # Son eğitimdeki başlangıç ağırlık vektörünün boyutu
433 # Son eğitimdeki bitiş ağırlık vektörünün boyutu
434 # Son eğitimdeki bitiş ve başlangıç ağırlık vektörlerinin oranı
435 # karşılaştırılır. Bu 40 kez, c = (0.1,4) aralığında 0.1 aralıklarla gerçekleştirilir.
436 w3 = getrandomw(0, 1)
437 cTestResults = []
438 resultFileC = os.path.join(__location__, 'SpeedResult.json')
439 for i in range(40):
440
441     #Rastgele dağıtılmış DomainEx3 dağılımını kullanır.
442     #DomainEx3'ün değişmemesi için bu dağılım DomainEx4'ye aktarılır.
443     insertDomain(objList, domainFile3)
444     extractDomain(objList, domainFile4)
445     #Elde ettiğimiz değerleri cTestResults'ta toplayıp döngü sonunda SpeedResult.json adlı dosyaya kaydederiz.
446     c = i/10 + 0.1
447     Inf = process(objList, w3, c, max_epoch*2, domainFile4)
448     Inf.append(c)
449     cTestResults.append(Inf)
450
451 for i in range(39):
452     w3 = getrandomw(0, 1)
453     for result in cTestResults:
454         insertDomain(objList, domainFile3)
455         extractDomain(objList, domainFile4)
456         c = result[6]
457         Inf = process(objList, w3, c, max_epoch*2, domainFile4)
458         Inf.append(c)
459         for i in range(len(result)):
460             if Inf[2] != -1:
461                 result[i] = (Inf[i] + result[i])/2
462 with open(resultFileC, 'w') as json_file:
463     json.dump(cTestResults, json_file)
```

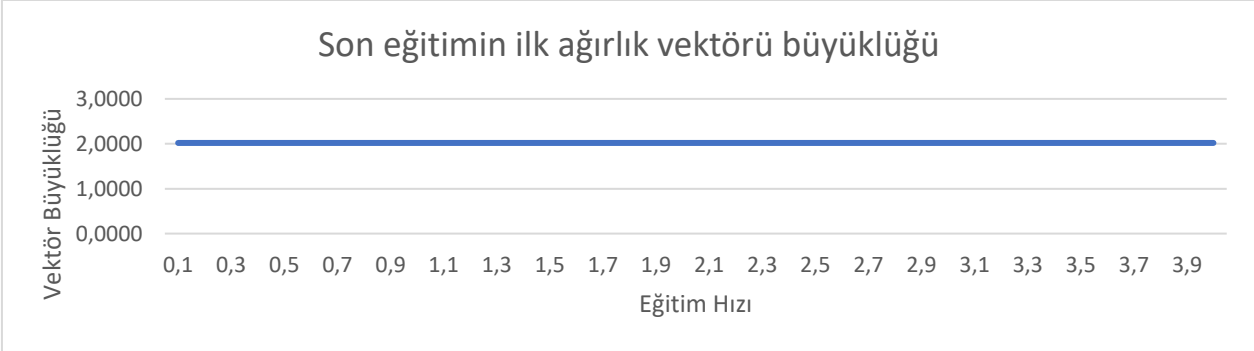
Sonuçlara bakılırsa, hata sayısı her zaman 0'da kalmıştır. Bu seçilen kümelerin iyi dağılmasından kaynaklanır. Bu şekilde kümeleri dengelemeye gerek kalmaz.



Her eğitim iterasyonunda meydana gelen ortalama hataya bakıldığında, 3 ile 4 arasında devam ettiği görülmektedir. Çok küçük eğitim hızı değerlerinde yüksektir, 0.3 ve 1 aralığında optimal değerlere sahiptir ve sonrasında tekrar yükselir. Buna rağmen büyük farklılık göstermez.

Son eğitimdeki iterasyon sayılarına bakıldığında, grafiğin oldukça dalgalı olmasına rağmen c = 2.8 ve üzeri eğitim hızlarının 13 civarı iterasyonda eğitimi tamamladığı ve önceki değerlerden oldukça hızlı olduğu gözlemlenir. Eğitim hızının yükselmesiyle eğitimin hızlanması beklendiktir.

Son eğitimdeki ilk ağırlık vektörünün büyüklüğüne bakılırsa sabit olduğu gözlenir. eğitim esnasında test kümesi hiç başarısız olmadığı için aynı ağırlık vektörü kullanılmıştır. Bu sebeple ilk ağırlık vektörü her zaman testin başında seçilen w3 vektörü olacaktır. Bu vektörün büyüklüğü 2.0188 değerindedir.

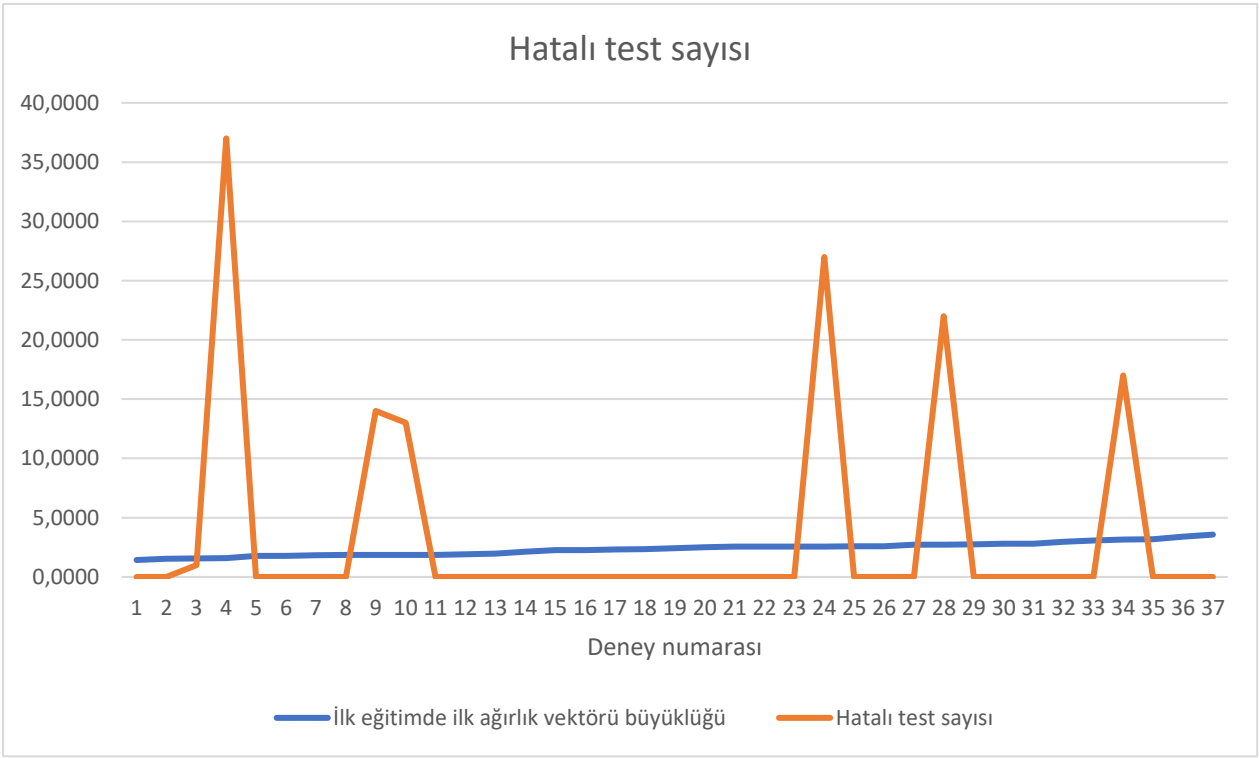


Son ağırlık vektörünün büyüklüğüne bakılırsa lineer olarak arttığı gözlenir. Eğitim sırasında iterasyon sayısının hıza göre çok da fazla değişmemesi ve ağırlık vektörü güncellenirken eğitim hızıyla doğru orantılı bir şekilde büyümesi bunu sağlar. İlk ağırlık vektörü tüm hızlarda sabit olduğu için aynı grafik son-ilk ağırlık vektörü oranlarında da görülür.

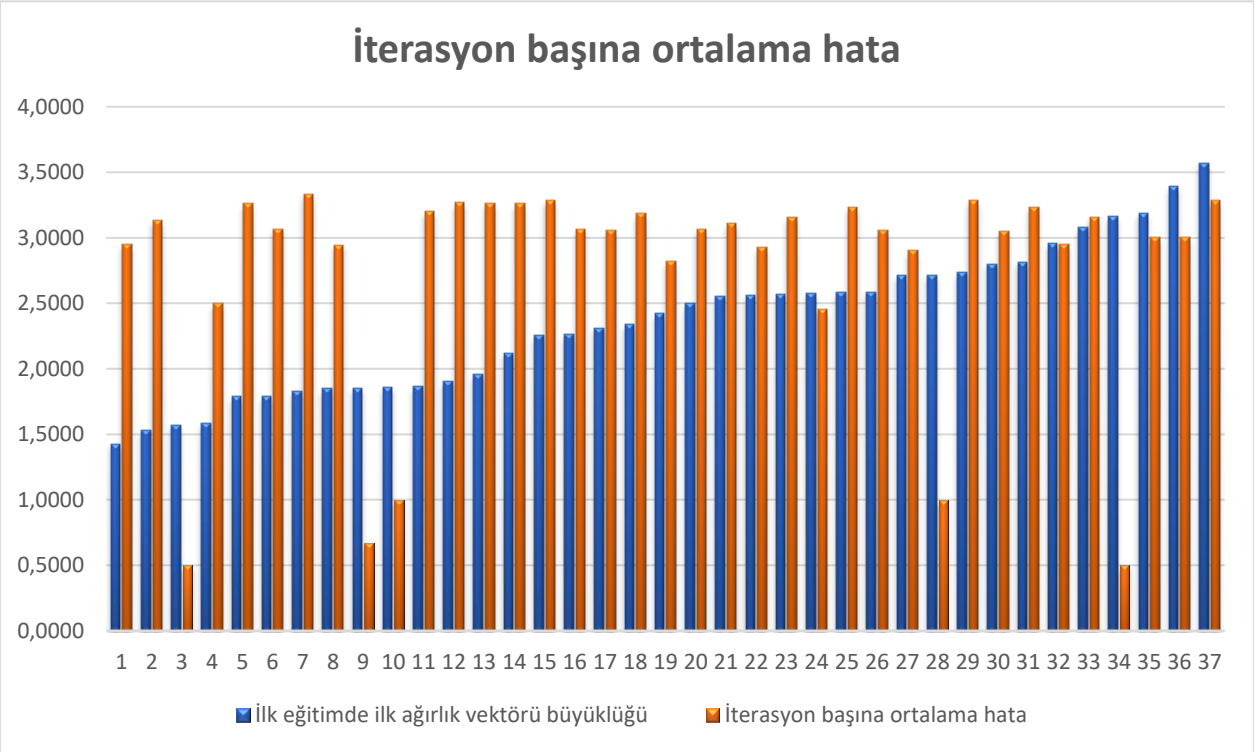
Her seferinde rastgele bir ağırlık vektörü seçip önceden belirlenmiş bir dağılım kümesi ve sabit eğitim hızı kullanır ve sonuçlar ilk seçilen ağırlık vektörünün büyüklüğüyle karşılaştırılır. Bu 40 kez gerçekleştirilip elde edeceğimiz sonuçların ortalaması WeightResult.json dosyasında saklanır.

```
467   domainFile5 = os.path.join(__location__, 'DomainEx5.pkl')
468   domainFile6 = os.path.join(__location__, 'DomainEx6.pkl')
469   # Daha sonra hız c = 1 ve aynı eğitim-test kümesiyle başlayıp farklı ağırlık vektörleriyle:
470   # Başarısız test sayısı
471   # Son eğitimdeki iterasyon başına düşen ortalama hata sayısı
472   # Son eğitimdeki iterasyon sayısı
473   # Son eğitimdeki başlangıç ağırlık vektörünün boyutu
474   # Son eğitimdeki bitiş ağırlık vektörünün boyutu
475   # Son eğitimdeki bitiş ve başlangıç ağırlık vektörlerinin oranı
476   # karşılaştırılır. Bu 40 kez, her döngüde farklı w değerleri alınarak gerçekleştirilir.
477   # Sonuç w vektörlerinin büyüklüklerine göre karşılaştırılacaktır.
478   wTestResults = []
479   resultFileW = os.path.join(__location__, 'WeightResult.json')
480   for i in range(40):
481       w4 = getrandomw(0, 1)
482       #Rastgele dağıtılmış DomainEx4 dağılımını kullanır.
483       #DomainEx5'in değişmemesi için bu dağılım DomainEx6'ya aktarılır.
484       insertDomain(objList, domainFile5)
485       extractDomain(objList, domainFile6)
486       #Elde ettiğimiz değerleri wTestResults'ta toplayıp döngü sonunda WeightResult.json adlı dosyaya kaydederiz.
487       Inf = process(objList, w4, 1, max_epoch*2, domainFile6)
488       Inf.append(np.linalg.norm(w4))
489       wTestResults.append(Inf)
490
491   with open(resultFileW, 'w') as json_file:
492       json.dump(wTestResults, json_file)
```

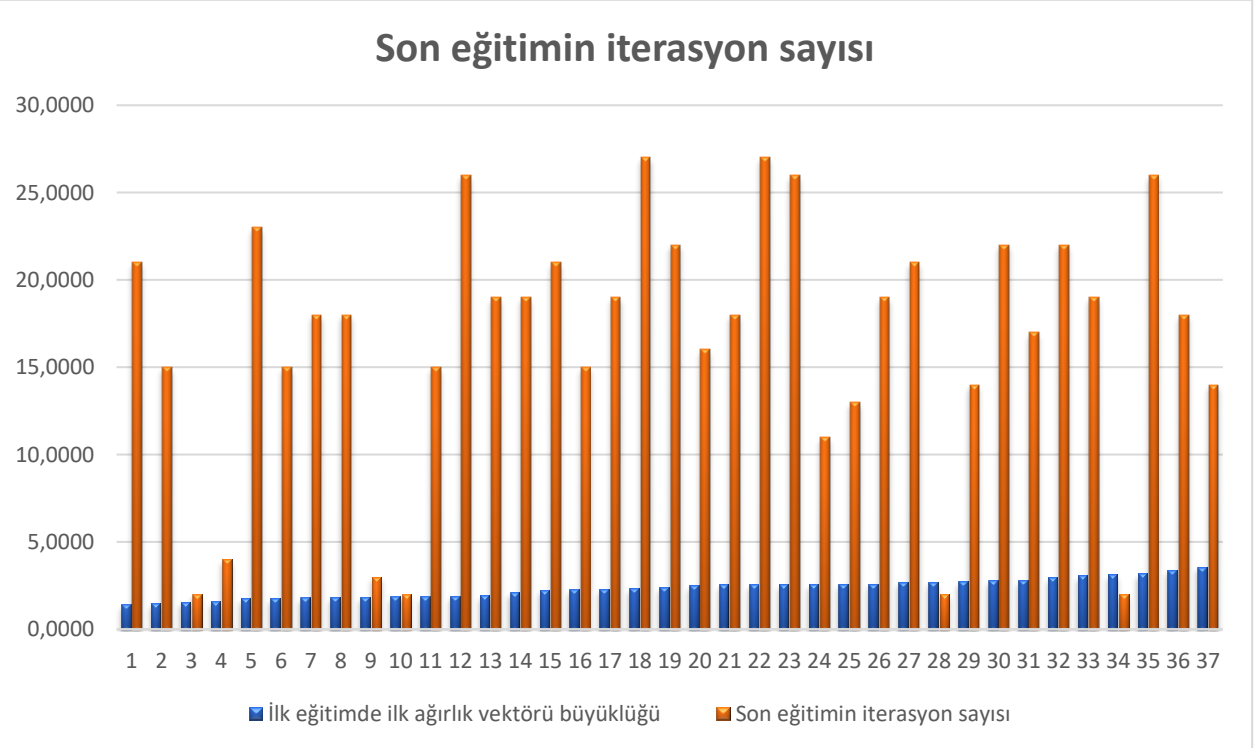
Bu deneyde test kümesinden birkaç kez hata alınmıştır. Hatalı olan kümelerin ortalama 20 seferde dengelendiği görülür. Eğitim ve test kümeleri iyi dağılmıştır.



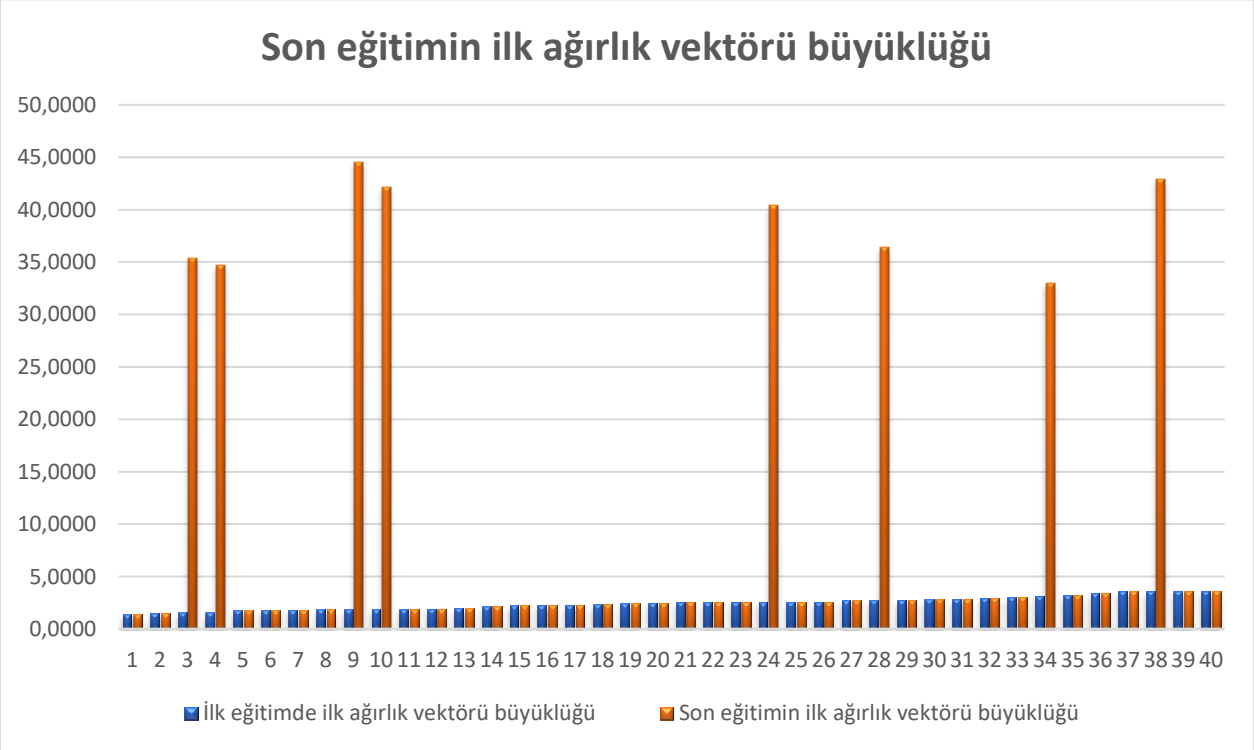
İterasyon başına düşen ortalama hata ilk ağırlık vektörünün büyüklüğünden ilişkisiz durmaktadır. Test kümesinin hatalı olduğu deneylerde düşük olduğu gözlemlenmektedir. Bunun sebebi test kümesindeki bir veya iki elemanın hatalı vermesi, bu elemanlar eğitim kümesine geçene kadar da ortalama hatanın düşmesidir.



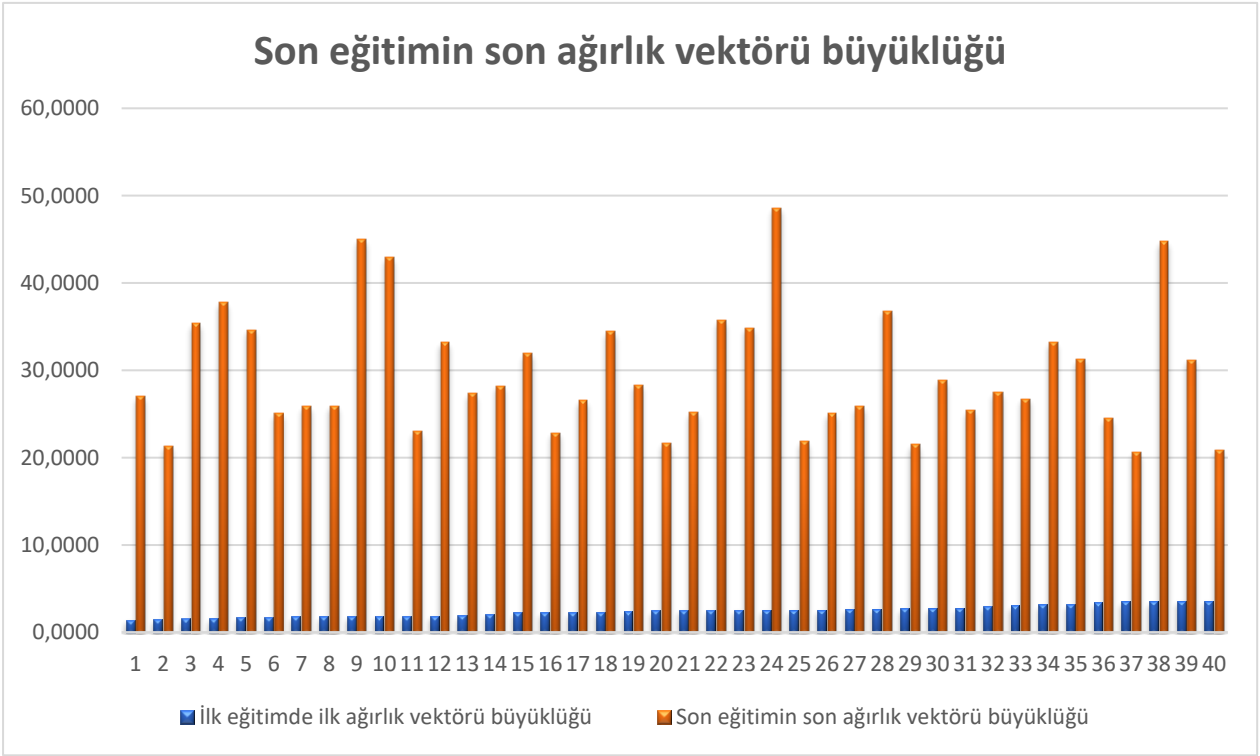
Son eğitimin iterasyon sayısı da ilk ağırlık vektörünün büyüklüğünden ilişkisiz durmaktadır fakat test kümesinin hatalı olduğu deneylerde düşüktür. Bunun sebebi test ve eğitim kümelerinin, eğitim tamamlanana kadar optimize olmasıdır. Diğer deneylerde de eğitim tamamlanmasına rağmen, verimsiz kümeler kullanılmıştır.



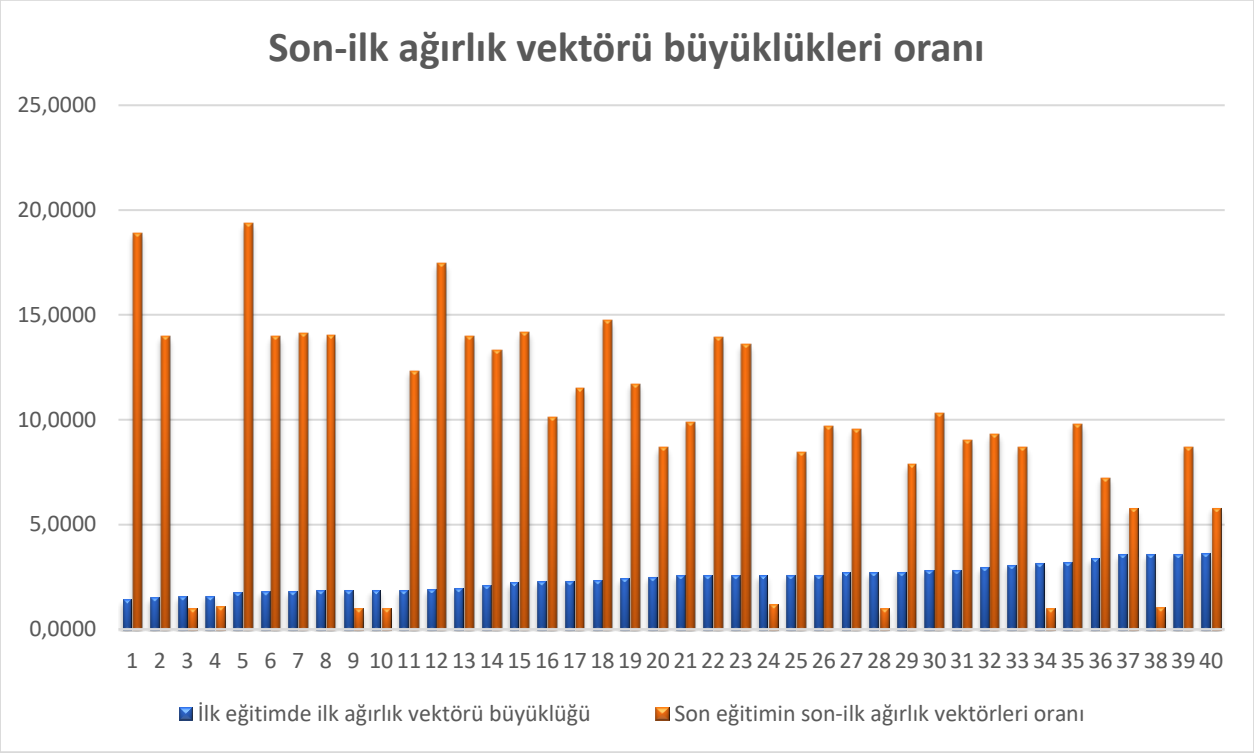
Son eğitimin ilk ağırlık vektörü, hata olmadığı sürece ilk girilen ağırlık vektörünün büyüklüğüne eşittir. Hata yaşanan deneylerde ilk ağırlık gitgide büyümektedir.



İlginç bir şekilde son ağırlık vektörü büyüklüğü, hatalı testlerden ve ilk vektörün ağırlığından ilişkisiz bir sonuç vermektedir. Bu vektörün büyüklüğünü yalnızca veri setinin belirlediği gözlemlenir.



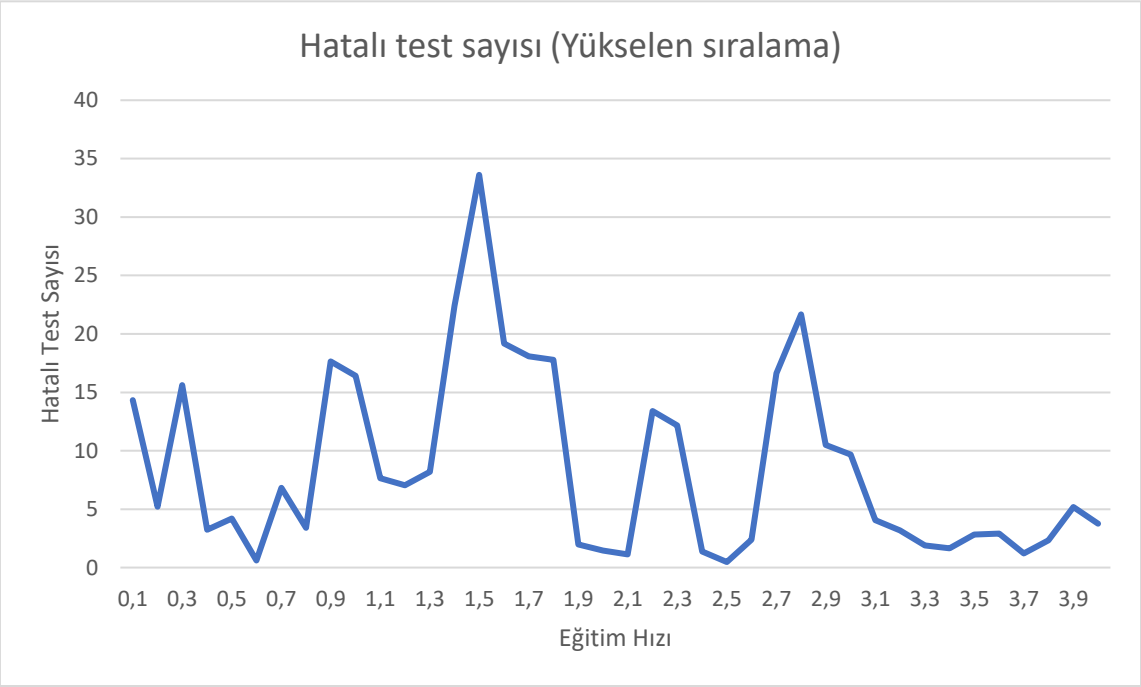
Son ağırlık vektörünün büyüklüğünün diğer parametrelerden ilişkisiz olması ve hatalı deneylerde ilk ağırlık vektörünün büyümesi sebebiyle, son eğitimin son-ilk ağırlık vektörleri büyüklükleri oranı çok düşük olacaktır. Bunun deneylerin dışında, yine son ağırlık vektörü nispeten sabit hareket ettiği için, ilk ağırlık vektörü büyüklüğü artarken son-ilk ağırlık vektörü büyüklükleri oranı düşmektedir.

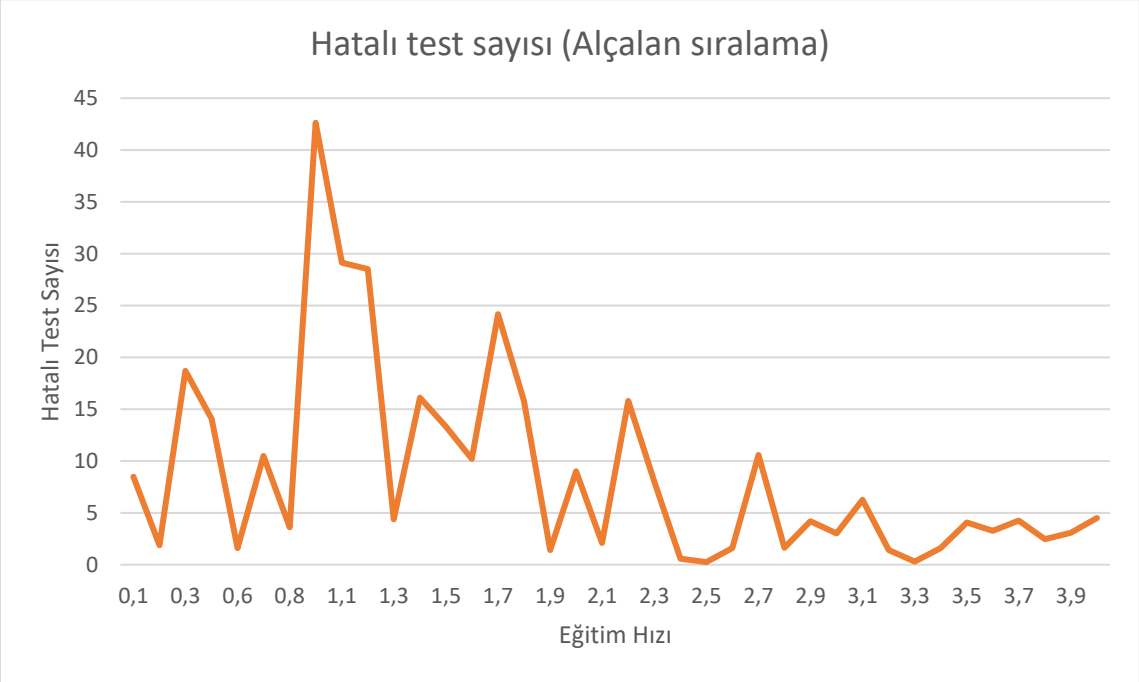


Farklı eğitim kümesi sıralamalarına göre, eğitim hızı değişirken gözlem yapılır. Rastgele seçilmiş w4 ağırlık vektörü ve DomainEx7 küme dağılımı sabit tutulur.

```
domainFile7 = os.path.join(__location__, 'DomainEx7.pkl')
domainFile8 = os.path.join(__location__, 'DomainEx8.pkl')
# Son olarak aynı eğitim-test kümesi ve ağırlıkla başlayıp
# eğitim kümesi vektörlerin büyüklüklerine göre farklı şekilde sıralandığında:
# Başarısız test sayısı
# Son eğitimdeki iterasyon başına düşen ortalama hata sayısı
# Son eğitimdeki iterasyon sayısı
# Son eğitimdeki başlangıç ağırlık vektörünün boyutu
# Son eğitimdeki bitiş ağırlık vektörünün boyutu
# Son eğitimdeki bitiş ve başlangıç ağırlık vektörlerinin oranı
# karşılaştırılır. Bu 40 kez c = (0.1,4) aralığında 0.1 değişirken eğitim kümesi
# elemanları büyükten küçüğe ve küçükten büyüğe sıralıyken 40 kez test edilir ve karşılaştırılır.
w4 = getrandomw(0, 1)
sortTestResults = []
resultFileSort = os.path.join(__location__, 'SortResult.json')
for i in range(40):
    #Rastgele dağıtılmış DomainEx6 dağılımını kullanır.
    #DomainEx7'nin değişmemesi için bu dağılım DomainEx8'e aktarılır.
    insertDomain(objList, domainFile7)
    extractDomain(objList, domainFile8)
    #Elde ettiğimiz değerleri sortTestResults'ta toplayıp döngü sonunda SortResult.json adlı dosyaya kaydederiz.
    c = i/10 + 0.1
    sort = "descending"
    descList = sortList(objList, -1)
    Inf = process(descList, w4, c, max_epoch*2, domainFile8)
    Inf.append(c)
    Inf.append(sort)
    if Inf[2] != -1:
        sortTestResults.append(Inf)

    insertDomain(objList, domainFile7)
    extractDomain(objList, domainFile8)
    sort = "ascending"
    ascList = sortList(objList, 1)
    Inf = process(ascList, w4, c, max_epoch*2, domainFile8)
    Inf.append(c)
    Inf.append(sort)
    if Inf[2] != -1:
        sortTestResults.append(Inf)

for i in range(39):
    for result in sortTestResults:
        insertDomain(objList, domainFile7)
        extractDomain(objList, domainFile8)
        c = result[6]
        Inf = process(objList, w4, c, max_epoch*2, domainFile8)
        Inf.append(c)
        Inf.append(sort)
        for j in range(len(result)-1):
            if Inf[2] != -1:
                result[j] = (Inf[j] + result[j])/2
with open(resultFileSort, 'w') as json_file:
    json.dump(sortTestResults, json_file)
```

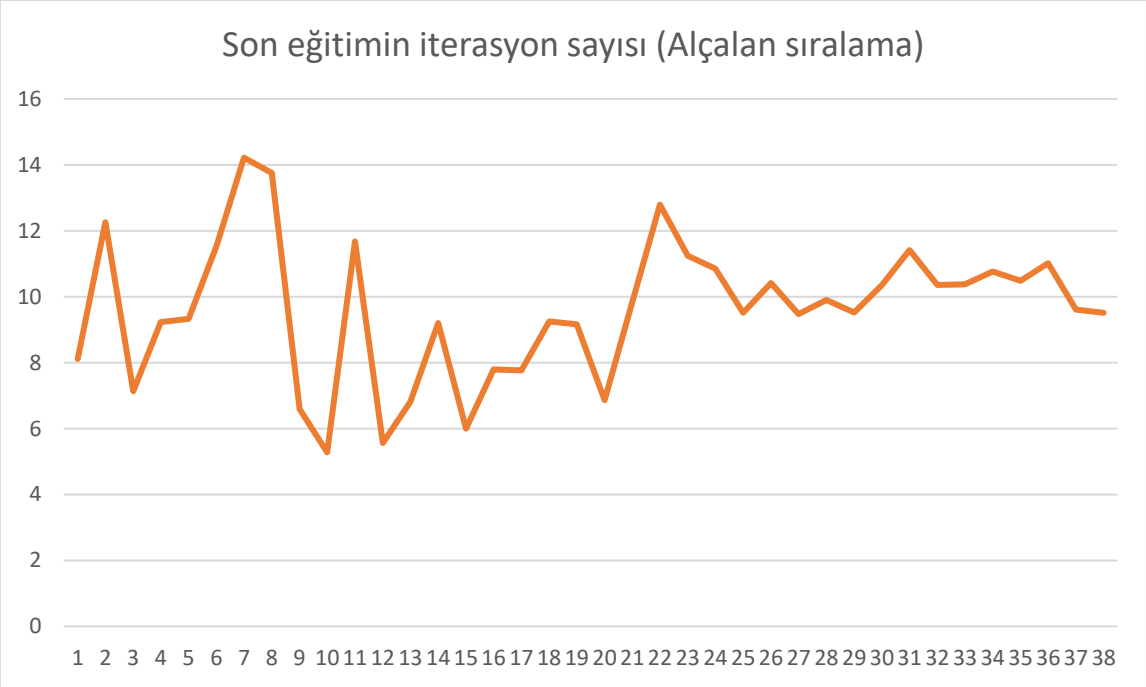
Sonuçlara bakıldığında, hatalı test sayısının yükselen sıralamada daha düşük olmasıyla birlikte iniş ve çıkışların sıralamadan fazla etkilenmediği görülür. Eğitim hızı 1.5-2.5 aralığında hatalı test sayısında bir artış vardır ve aralığın dışında genel olarak düşüktür.





İterasyon başına düşen hata da sıralamadan bağımsız olmakla birlikte, eğitim hızı 0.8-1.9 aralığında düşüktür.

Son eğitimde gerçekleşen iterasyon sayısı da verilerin sıralamasıdan etkilenmemektedir.



Son eğitimdeki ilk ağırlık vektörü de sıralamadan etkilenmemektedir. Büyüklüğü, yüksek ve düşük eğitim hızlarında, 1-3 aralığındakinden daha düşüktür.

Son eğitimin son ağırlık vektörü; ilk ağırlık vektörü, iterasyon sayısı ve eğitim hızıyla doğru orantılı olduğundan sıralamadan bağımsız bir şekilde, eğitim hızı arttıkça artmaktadır.

Son olarak, son ağırlık vektörlerinin ilk ağırlık vektörlerine oranı karşılaştırıldığında bunların da sıralamadan bağımsız olduğu görünür.

Sorunun diğer kısmı için malesef lineer ayrıştırılamaz bir veri seti elde edilemedi.

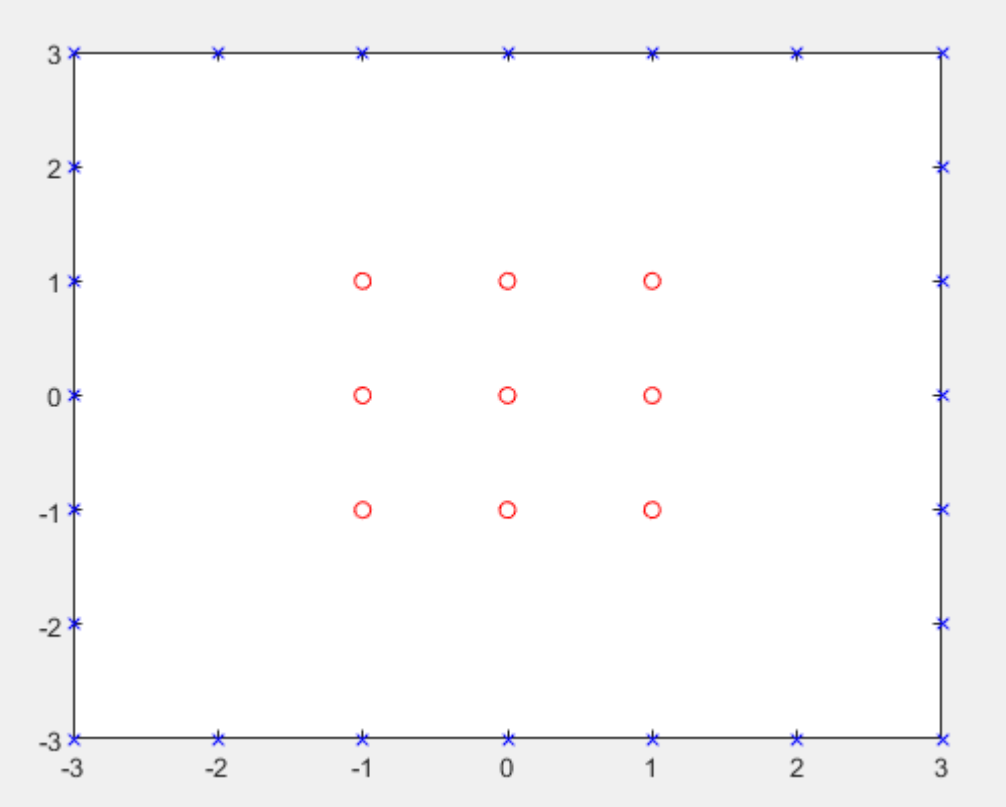
```
nonLinFile = os.path.join(__location__, 'NonLinearData.pkl')

if os.path.exists(nonLinFile):
    nonLinList = loadList(nonLinFile)
    nonLinDict = newDict(nonLinList)

else:
    nonLinList = newData(max_epoch)
    nonLinDict = newDict(nonLinList)
    distDomain(nonLinList)
    distClass(max_epoch, nonLinDict, nonLinList, 0)
    saveData(nonLinList, nonLinFile)

#Sorunun ikinci kısmı için lineer ayrıştırılamaz bir küme oluşturulur. Kodun bu kısmı bilinmeyen bir hatadan dolayı çalışmamakta.
#Verilerin kümeleri karışıyor ve eğitim kümesi anında eğitiliyor.
Inf = process(nonLinList, [], 1, max_epoch*2, nonLinFile)
domainTest(nonLinList)
#Verilerin son halini kaydeder
saveData(nonLinList, nonLinFile)
saveData(objList, dataFile)
```

```
First weight: [-1.1096  0.7838 -1.1416  0.7309  0.7168 -1.1141]
Tested error count of  1 th education is  0
Failed Test Count: 0
Educated on:  1 th iteration.
Final weight:  [-1.10957975  0.78384397 -1.14157372  0.73091549  0.71676368 -1.11405434]
  -1  -1  -1  -1  1  -1  -1  1  -1  1  1  1  -1  1  1  -1  -1  -1  1  1  1  1  -1  1  -1  -1  1  -1  -1  -1  -1  1  -1  1  1  1  1  1  -1  1
E1 count: 20    T1 count: 0
E2 count: 20    T2 count: 0
PS D:\Mehmet\Python>
```

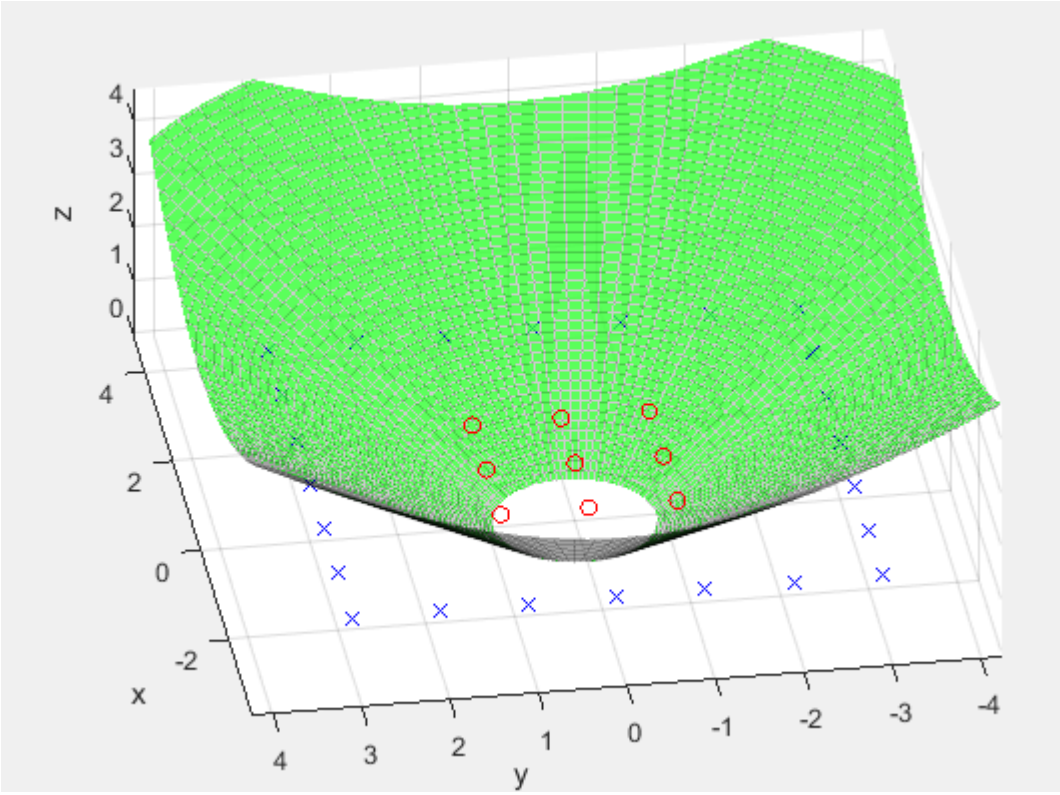
Soru 2:

Bu soru diğerlerinden farklı olarak MATLAB üzerinden çözülmüştür.

Veriler çizildiğinde, lineer ayrıştırılamadığı görülebiliyor. Lineer ayrıştırılabilir hale getirmek için verilere yeni bir boyut eklenir, bu şekilde üç boyutlu bir uzayda veriler bir düzlem ile ayrıştırılabilir.



Bunun için birçok düzlem seçilebilir. Bizim seçtiğimiz düzlem, verilerle aynı z koordinatındayken r = 2 çemberi kesitine sahip koni düzlemi oldu.



Bu işlemlerin yapıldığı MATLAB kodu aşağıdaki gibidir.

```
i = 1;
A = zeros(24,4);
B = zeros(9,4);

%Yd = -1 olan veri matrisi
for x = -3:3
    for y = -3:3
        if (abs(x) == 3)||(abs(y) == 3)
            A(i,:) = [x;y;1;1];
            i = i+1;
        end
    end
end

%Yd = 1 olan veri matrisi
i = 1;
for x = -1:1
    for y = -1:1
        B(i,:) = [x;y;1;1];
        i = i+1;
    end
end

%Çember tanımlanıyor.
r = linspace(0,10) ;
th = linspace(0,2*pi) ;
[R,T] = meshgrid(r,th) ;
X = R.*cos(T) ;
Y = R.*sin(T) ;
%Verilere Z = 1 boyutu eklendiği için
%Z = 1 => R = 2 olması isteniyor. R, Z'ye bağımlı olduğu için bir koni
%düzlemi tanımlanıyor.
Z = R - 1 ;

s1 = surf(X,Y,Z,'FaceAlpha',0.8,'EdgeAlpha',0.2,'FaceColor',[50 255 50]/255);
xlabel('x');
ylabel('y');
zlabel('z');
hold on
plot3(A(:,1),A(:,2),A(:,3),'x','MarkerEdgeColor','b');
plot3(B(:,1),B(:,2),B(:,3),'o','MarkerEdgeColor','r');
hold off
```

Soru 3:

f(x1, x2) = 3*x1 + 2*cos(x2) fonksiyonu x1 = [0,1] ve x = [0, π/2] aralığındayken yaklaşık olarak ifade edilmeli. Fonksiyonun minimumu f(0, π/2) = 0, maksimumu f(1, 0) = 5. g(x1, x2) = f(x1, x2)/5 fonksiyonu tanımlanıp kullanılırsa [0,1] aralığını kapsayacak logit Adaline fonksiyonu kullanarak yaklaşılabilir. $phi(v) = \frac{1}{1+e^{-v}}$
(x1*w1 + x2*w2 + 1*w3 = v)

Veri setinin oluşumu için rastgele 100 farklı x1-x2 seti seçilir, rastgele bir şekilde test ve eğitim kümelerine dağıtılır ve veriler oluşturulur.

```python
class DataPoints:
    #DataPoints obje sınıfında koordinat, anahtar, veri sınıfı mevcut.
    def __init__(self, Coordinate, Key, Domain):
        self.Coordinate = Coordinate
        self.Key = Key
        self.Domain = Domain

    def setDomain(self, Domain):
        self.Domain = Domain

    def setCoordinate(self, Coordinate):
        self.Coordinate = Coordinate
```

```python
__location__ = os.path.realpath(os.path.join(os.getcwd(), os.path.dirname(__file__)))
dataFile = os.path.join(__location__, 'DataQ3.pkl')
max_epoch = 100

GetCoordinate = attrgetter('Coordinate')
GetKey = attrgetter('Key')
GetDomain = attrgetter('Domain')

if os.path.exists(dataFile):
    objList = loadList(dataFile)

else:
    objList = newData(max_epoch)
    distDomain(objList)
    saveData(objList, dataFile)
```

```python
#Yeni verilerin oluşturulduğu fonksiyon
def newData(epoch):
    DataList = []
    for i in range(epoch):
        #x1 ve x2'yi soruda verilen aralıklarda rastgele sayılardan oluşturur. Daha sonra bias ekleyip veri koordinatına geçirir.
        x1 = random.random()
        x2 = np.pi*random.random()/2
        Coord = [x1, x2, 1]
        Coord = np.around(Coord, decimals = 9)
        #Kordinatlarla yeni bir veri listesi oluşturur. Yeni verilerin anahtarları iterasyon sayısına eşit olacak ve birbirinden farklı olacaktır.
        DataList.append(DataPoints(Coord, i, "None"))
    return DataList
```

Ardından veri setiyle eğitim başlatılır. Eğitim için w1 = [1 1 1] ağırlık vektörü ve c = 0.5 eğitim hızı kullanılır. Eğitim, ortalama kare hatasının önceki iterasyonla farkı, kendisinin 10^-12 katından aşağıya düşmedikçe devam edecektir. Ayrıca bu koşul sağlandıktan sonra test döngü tekrar gerçekleşir ve test kümesinde aynı koşulun sağlanması gerekir. Bununla birlikte eğitim tamamlanır.

```python
def educate(objList, w, c):
    print("İlk ağırlık vektörü:", w)
    count = 1
    countT = -1
    i = 0
    Eortprev = 0
    EortTprev = 0
    Eort = 0
    w = np.array(w)
    while i < 1:
        E = 0
        for obj in objList:
            if GetDomain(obj) == "E":
                Coord = GetCoordinate(obj)
                yd = f(Coord[0], Coord[1])/5
                v = np.dot(w, Coord)
                y = phi(v)
                e = yd - y
                E += (1/2)*(e**2)
                phiDerivative = phi(v)*(1 - phi(v))
                w = w + e*phiDerivative*Coord
        Eort = E/len(objList)
        EortR = Eort
        if abs(Eort - Eortprev) < Eort/(10**12):
            i +=1

        if i == 1:
            E = 0
            for obj in objList:
                if GetDomain(obj) == "T":
                    yd = f(Coord[0], Coord[1])/5
                    v = np.dot(w, Coord)
                    y = phi(v)
                    e = yd - y
                    E += (1/2)*(e**2)
            Eort = E/len(objList)
            if abs(Eort - EortTprev) < Eort/(10**12):
                i +=1
            else:
                countT += 1

            EortTprev = Eort
        print("İterasyon:",count,"\tOrtalama hata:", EortR, "\tAğırlık vektörü:", w)
        count += 1
        Eortprev = Eort
        Eort = 0
    if countT > 0:
            print("Başarısız Test Sayısı:", countT)
    result = [w, count, EortR]
    return result
```

Eğitim sırasında her iterasyonda her veri için yd = f(x1, x2)/5 değeri hesaplanır. Daha sonra wT*x ile v bulunup phi(v) değeri elde edilir. e = yd – y hesaplanıp toplam veri hatası E = (1/2)*e^2, iterasyonun toplam hatası E'ye eklenir. phi(v) fonksiyonu türevi hesaplanır. Ardından bu değerler ağırlık vektörünün güncellenmesi için w = w + e*phiDerivative*Coord şeklinde toplanır. Daha sonra w güncellenmeden, aynı işlem test kümesinde yapılır.

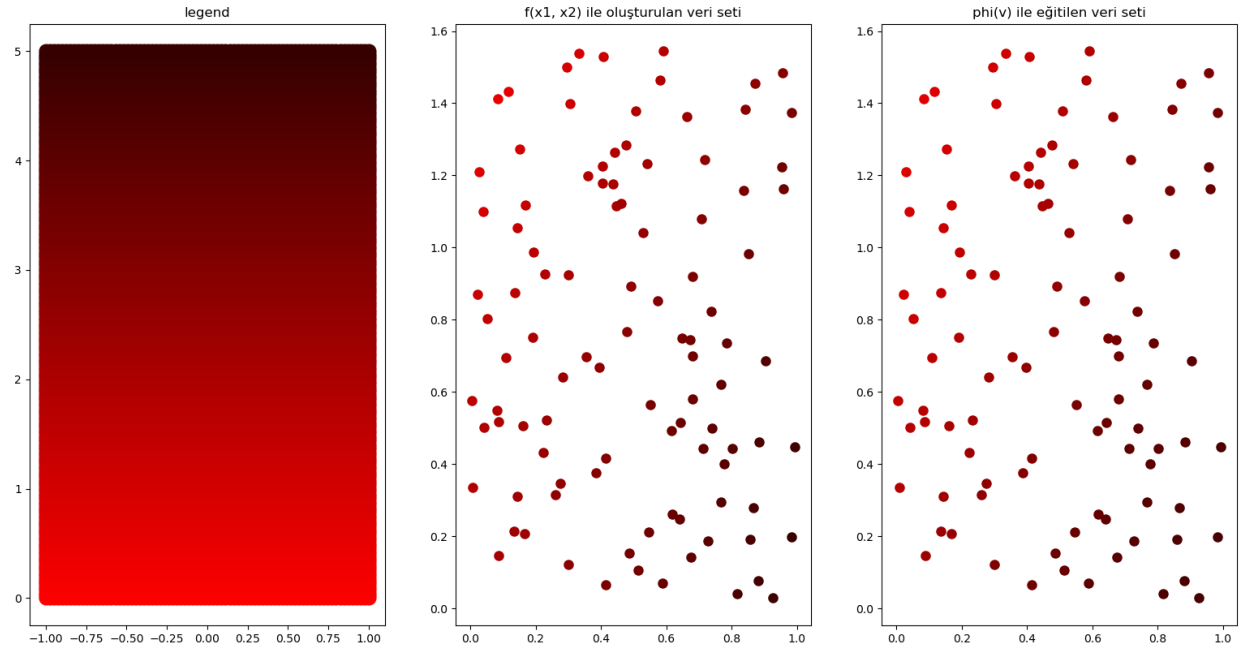
```
while i < 1:
    E = 0
    for obj in objList:
        if GetDomain(obj) == "E":
            Coord = GetCoordinate(obj)
            yd = f(Coord[0], Coord[1])/5
            v = np.dot(w, Coord)
            y = phi(v)
            e = yd - y
            E += (1/2)*(e**2)
            phiDerivative = phi(v)*(1 - phi(v))
            w = w + e*phiDerivative*Coord
    Eort = E/len(objList)
    EortR = Eort
    if abs(Eort - Eortprev) < Eort/(10**12):
        i +=1
```

```
    if i == 1:
        E = 0
        for obj in objList:
            if GetDomain(obj) == "T":
                yd = f(Coord[0], Coord[1])/5
                v = np.dot(w, Coord)
                y = phi(v)
                e = yd - y
                E += (1/2)*(e**2)
        Eort = E/len(objList)
        if abs(Eort - EortTprev) < Eort/(10**12):
            i +=1
        else:
            countT += 1

        EortTprev = Eort
    print("İterasyon:",count,"\tOrtalama hata:", EortR, "\tAğırlık vektörü:", w)
    count += 1
    Eortprev = Eort
    Eort = 0
if countT > 0:
        print("Başarısız Test Sayısı:", countT)
result = [w, count, EortR]
return result
```

Kod çalıştığında 199. Iterasyonda başarılı olup, yaklaşık % 0.02266 ortalama hata ile eğitimin tamamlandığı görülür. Son ağırlık vektörü w = [2.76695842 -1.2814749 -0.09802192] elde edilir.

```
İterasyon: 192  Ortalama hata: 0.00022663226123325632    Ağırlık vektörü: [ 2.76695842 -1.2814749  -0.09802192]
İterasyon: 193  Ortalama hata: 0.00022663226123329429    Ağırlık vektörü: [ 2.76695842 -1.2814749  -0.09802192]
İterasyon: 194  Ortalama hata: 0.00022663226123332801    Ağırlık vektörü: [ 2.76695842 -1.2814749  -0.09802192]
İterasyon: 195  Ortalama hata: 0.00022663226123358028    Ağırlık vektörü: [ 2.76695842 -1.2814749  -0.09802192]
İterasyon: 196  Ortalama hata: 0.00022663226123384748    Ağırlık vektörü: [ 2.76695842 -1.2814749  -0.09802192]
İterasyon: 197  Ortalama hata: 0.00022663226123340848    Ağırlık vektörü: [ 2.76695842 -1.2814749  -0.09802192]
İterasyon: 198  Ortalama hata: 0.00022663226123429594    Ağırlık vektörü: [ 2.76695842 -1.2814749  -0.09802192]
Son ağırlık vektörü: [ 2.76695842 -1.2814749  -0.09802192]      İterasyon sayısı: 199   Son iterasyonda ortalama hata: 0.00022663226123429594
```

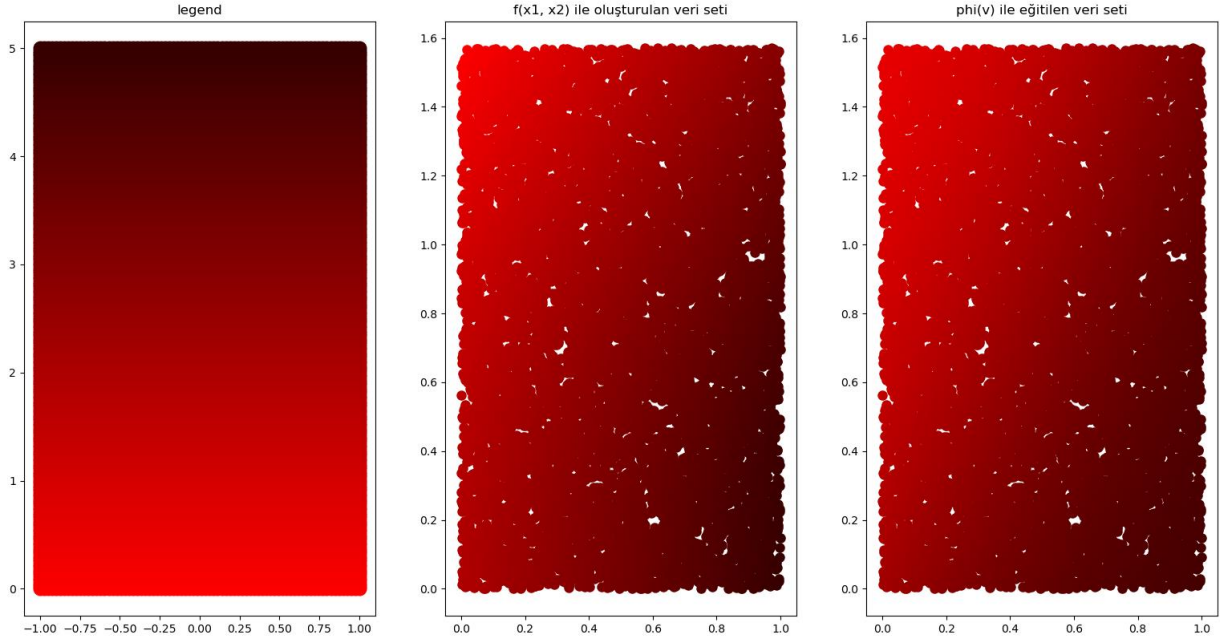
Ek olarak, sonuçlar çizdirilmek istenirse aşağıdaki şekil elde edilir.



Buradan da eğitilen sistemin istenilen fonksiyona yakınlığı gözlemlenebilir. Daha iyi gözlem yapabilmek için veri sayısını artırırsak, örneğin 10000 veri alırsak eğitimin çok daha hızlı gerçekleştiğini görürüz. Aynı parametrelerle işlem yapıldığında yeni veri setiyle eğitim 5. Iterasyonda gerçekleşir. Elde edilen son ağırlık [2.72203588 -1.25285761 -0.1246531] ve yaklaşık ortalama hata oranı %0.023575 olarak gözlemlenir.

```
İlk ağırlık vektörü: [1 1 1]
İterasyon: 1    Ortalama hata: 0.0005214433184262373    Ağırlık vektörü: [ 2.72202975 -1.25285643 -0.12465089]
İterasyon: 2    Ortalama hata: 0.0002357527208975145    Ağırlık vektörü: [ 2.72203588 -1.25285761 -0.1246531 ]
İterasyon: 3    Ortalama hata: 0.00023575270154065731   Ağırlık vektörü: [ 2.72203588 -1.25285761 -0.1246531 ]
İterasyon: 4    Ortalama hata: 0.00023575270154056508   Ağırlık vektörü: [ 2.72203588 -1.25285761 -0.1246531 ]
Son ağırlık vektörü: [ 2.72203588 -1.25285761 -0.1246531 ]      İterasyon sayısı: 5     Son iterasyonda ortalama hata: 0.00023575270154056508
```

Hata oranının önceki deneyden yüksek olması, fonksiyonun yaklaşmaya çalıştığı çok daha fazla veriye sahip olmasından kaynaklanır. Buna rağmen iki fonksiyon arasındaki yakınlık, sonuçlar çizdirilince gözlemlenebilir.



Soru 4:

Genlikte sürekli algılayıcı ile böyle bir fonksiyonun elde edilebilmesi için, seçilecek aktivasyon fonksiyonu phi(v) de bu tipte olmalıdır. v = w1x1 + w2x2 + … + wnxn + w(n+1) olarak açılırsa, phi(v) fonksiyonunun

$f(x1, x2, \dots, xn) = \sum_{k=i} c_i x_i + \sum_{k=i} \sum_{k=i} c_{ij} x_i x_j$ gibi bir fonksiyona benzetilmesi

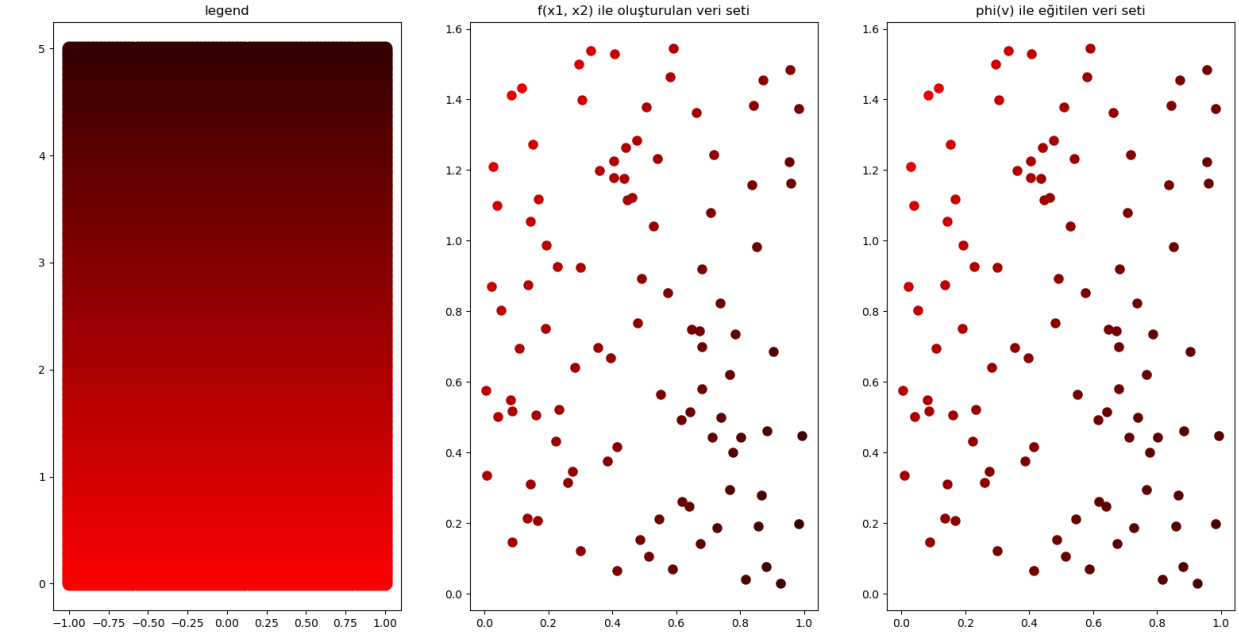
Için verilen x1 ve x2 aralığı için $phi(v) = \tanh(v)$ aktivasyon fonksiyonu kullanılabilir.

```
def phi(v):
    return abs(np.tanh(v))
```

```
while i < 1:
    E = 0
    for obj in objList:
        if GetDomain(obj) == "E":
            Coord = GetCoordinate(obj)
            yd = f(Coord[0], Coord[1])/5
            v = np.dot(w, Coord)
            y = phi(v)
            e = yd - y
            E += (1/2)*(e**2)
            phiDerivative = np.sinh(v)/(abs(np.sinh(v)/np.cosh(v))*(np.cosh(v)**3))
            w = w + e*phiDerivative*Coord
    Eort = E/len(objList)
    EortR = Eort
    if abs(Eort - Eortprev) < Eort/(10**12):
        i +=1
```

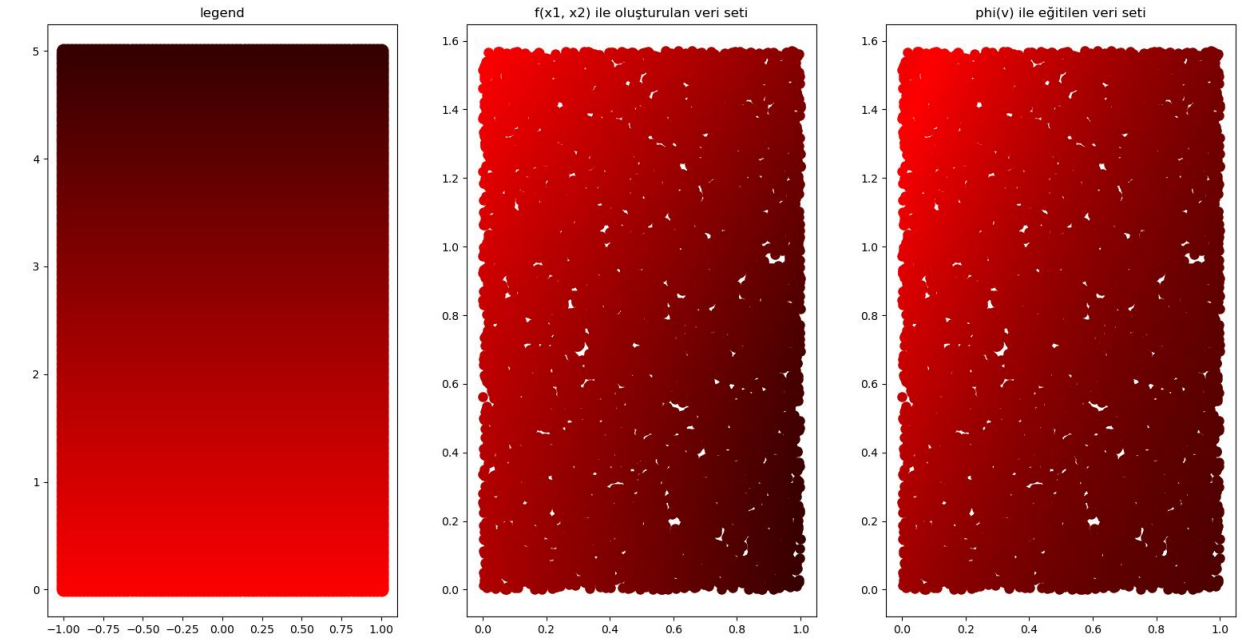
Bu şekilde bir eğitim gerçekleştirildiğinde önceki sorudaki 100 verilik veri seti ile 21 iterasyonda yaklaşık %0.1148 hata ile eğitim tamamlanır ve son ağırlık vektörü [0.86449066 -0.4097381 0.59531946] bulunur. Elde edilen veri çizdirildiğinde, birbirine yakın çizimler ortaya çıkar.

```
İterasyon: 15   Ortalama hata: 0.0011480737359049632    Ağırlık vektörü: [ 0.86449066 -0.4097381   0.59531946]
İterasyon: 16   Ortalama hata: 0.0011480737361513934    Ağırlık vektörü: [ 0.86449066 -0.4097381   0.59531946]
İterasyon: 17   Ortalama hata: 0.001148073736204048     Ağırlık vektörü: [ 0.86449066 -0.4097381   0.59531946]
İterasyon: 18   Ortalama hata: 0.0011480737362152967    Ağırlık vektörü: [ 0.86449066 -0.4097381   0.59531946]
İterasyon: 19   Ortalama hata: 0.001148073736217701     Ağırlık vektörü: [ 0.86449066 -0.4097381   0.59531946]
İterasyon: 20   Ortalama hata: 0.0011480737362182126    Ağırlık vektörü: [ 0.86449066 -0.4097381   0.59531946]
Son ağırlık vektörü: [ 0.86449066 -0.4097381   0.59531946]      İterasyon sayısı: 21    Son iterasyonda ortalama hata: 0.0011480737362182126
```



Eğitim 10000 verilik veri seti ile gerçekleştirildiğinde ise 4. Iterasyonda yaklaşık %0.1803 hata oranı ile eğitim tamamlanır ve son ağırlık vektörü [-0.88434009 0.39164808 -0.52397232] bulunur. Bu veri çizdirilince yine birbirine yakın şekiller ortaya çıkar.

```
İlk ağırlık vektörü: [1 1 1]
İterasyon: 1    Ortalama hata: 0.001979286490299345     Ağırlık vektörü: [-0.88434009  0.39164808 -0.52397232]
İterasyon: 2    Ortalama hata: 0.001802687739602909     Ağırlık vektörü: [ 0.88434009 -0.39164808  0.52397232]
İterasyon: 3    Ortalama hata: 0.001802687739602909     Ağırlık vektörü: [-0.88434009  0.39164808 -0.52397232]
Son ağırlık vektörü: [-0.88434009  0.39164808 -0.52397232]      İterasyon sayısı: 4     Son iterasyonda ortalama hata: 0.001802687739602909
```



Sonuca bakacak olursak w vektörü yaklaşık olarak ilk veri setiyle

w = [0.8645 -0.4097   0.5953] = [w11 w12 w13]

ikinci veri setiyle

w = [-0.8843 0.3917 -0.524] = [w21 w22 w23]

bulunmuştur. Fonksiyona koyulduğunda

phi(v) = phi(w1*x1 + w2*x2 + w3*1) = c1*x1 + c2*x2 + c3*x1*x2 = f(x1, x2) elde edilir.

İki tarafında;

x1'e göre türevi alındığında phi'(v)*w1 = c1 + c3*x2

x2'ye göre türevi alındığında phi'(v)*w2 = c2 + c3*x1 olacaktır. c1 ve c2 x1 ve x2'ye bağımlılardır.

Wolfram kullanılarak ilk veri seti için çizildiğinde c11 + c13*x2:

plot $\frac{\partial(0.8645\,|\tanh(0.59532 + 0.8645\,x1 - 0.4097\,x2)|)}{\partial x1}$

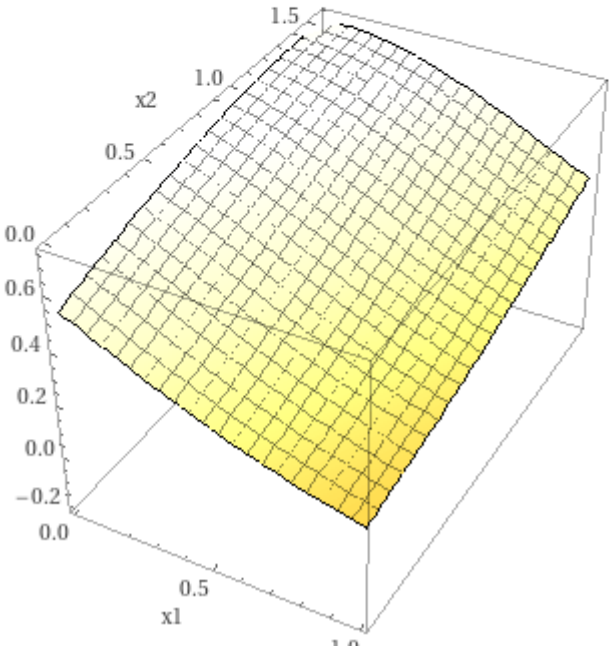
$x1 = 0$ to 1

$x2 = 0$ to $\frac{\pi}{2}$

tanh

3D plot:

Enla



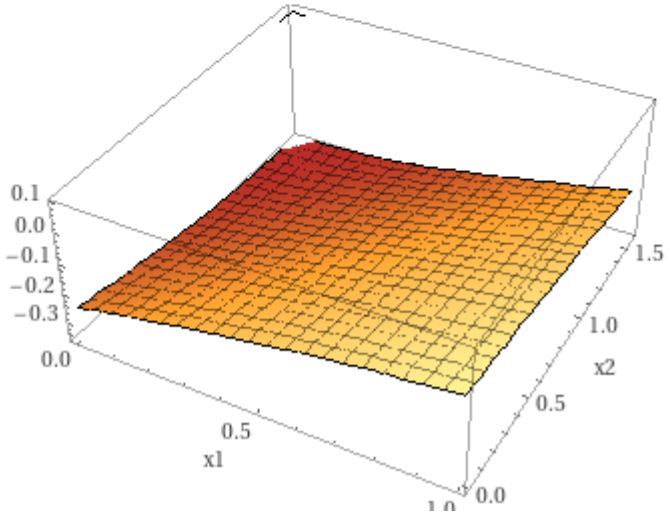
c12 + c13*x1:

Input interpretation:

plot	$\frac{\partial(-0.4097\,\lvert\tanh(0.59532 + 0.8645\,x1 - 0.4097\,x2)\rvert)}{\partial x1}$	$x1 = 0$ to 1 $x2 = 0$ to $\frac{\pi}{2}$

$\tanh(x)$ is

3D plot:

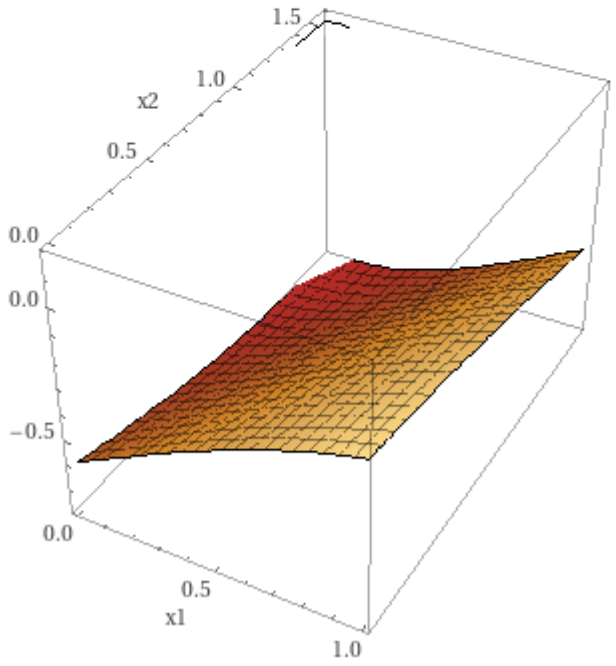


İkinci veri seti için c21 + c23*x1:

plot	$\frac{\partial(-0.8843\,\lvert\tanh(-0.524-0.8843\,x1+0.3917\,x2)\rvert)}{\partial x1}$	$x1 = 0$ to 1 $x2 = 0$ to $\frac{\pi}{2}$

$\tanh(x)$

3D plot:

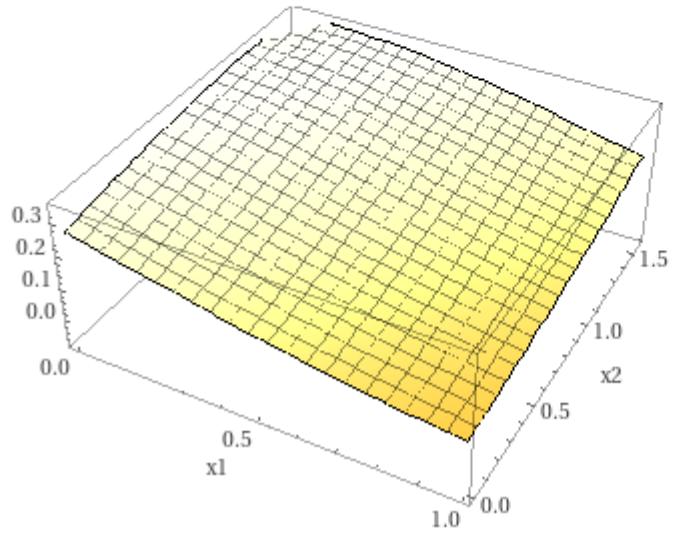


c22 + c23*x2:

plot	$\frac{\partial(0.3917\,\lvert\tanh(-0.524-0.8843\,x1+0.3917\,x2)\rvert)}{\partial x1}$	$x1 = 0$ to 1 $x2 = 0$ to $\frac{\pi}{2}$

tanh

3D plot:



Kullanılan kaynak ve kütüphaneler:

- Python
  - Numpy
  - Matplotlib
  - random
  - math
  - json
  - pickle
  - scipy
- WolframAlpha
- Stackoverflow

Kullanılan kodlara, ödev dosyasına ek olarak https://github.com/meserbetcioglu/ehb420hw1.git linkini kullanarak erişebilirsiniz.