Yapay Sinir Ağları
3.Ödev

03.01.2021

Mehmet Şerbetçioğlu -- 040160056

Ahmet Hulusi Tarhan -- 040170738

Soru 1: Üç Boyutlu Noktalar Kümesi

İlk problemde 200 noktadan oluşan üç boyutlu üç farklı noktalar kümesi oluşturup Kohonen ağıyla eğitmemiz isteniyor. Kohonen ağı yüksek boyutlu verilerin bilgisini düşük boyutlu bir uzay üzerinde ifade eder. Verilerin yoğun ve seyrek olduğu noktaları göstermeye yarar.

Eğitim sırasında veri, ağdaki tüm nöronlarla karşılaştırılır. Ağırlığı veriye en yakın olan nöron kazanan nöron seçilir ve ağdaki tüm nöronlar, kazanan nörona olan uzaklıkları doğrultusunda güncellenir. Bu işlem her veri için yapılır ve her veride kazanan nöron ve komşuları veriye yaklaşır.

Eğitimin iki farklı aşaması vardır. Bunlar özdüzenleme ve yakınsama aşamalarıdır. Özdüzenleme aşaması ağırlıkların verileri kabaca ifade etmesini sağlar. Her iterasyonda komşuluğun etkisi ve öğrenme hızı düşmekte, ağırlıklar verilere daha dar bir komşulukta ve daha az bir miktarda yaklaşmaktadır. Yakınsama aşamasında ise verilerin daha isabetli bir ifadesi elde edilmeye çalışılır. Bu aşamada komşuluk etkisi ve öğrenme hızı düşük bir seviyededir ve sabittir.

Haykin “Neural Networks – A Comprehensive Foundation” kitabında, bu iki eğitimin sürmesi gereken toplam iterasyon sayısı için, özdüzenlemeye 1000, yakınsamaya ise nöron sayısının 500 katı olması gerektiği tavsiyesinde bulunmuştur. Ayrıca adaptif öğrenme hızı ve komşuluk etkisi için kullanılacak zaman sabitleri ve başlangıç öğrenme hızı için de tavsiyelerde bulunmuştur. Bu tavsiyeler, sebepleriyle birlikte kitabın 474-475. sayfalarında açıklayıcı bir şekilde anlatılmıştır.

Nöron Sınıfı:

Ağı oluşturan nöronların olduğu sınıf. İlk tanıtıldığında (x,y) indisleri, başlangıçtaki öğrenme hızı, ağırlık vektörü için kullanılacak olan veri boyutu ve şimdilik kendisini gösteren sonraki nöron ifadesi bulunmaktadır. Ayrıca yeni nöron oluştuğunda ağırlıkların atanması için bir reset() fonksiyonu bulunmaktadır.

```
# Nöron sınıfı.
class NeuronCls():
    # Her nöron oluşturulduğunda (x,y) indisine, başlangıç öğrenme hızına,
    # w boyutuna ve kendinden sonra gelen nöronun göstergesine sahiptir.
    def __init__(self, x, y, dimension, learning_rate):
        self.x = x
        self.y = y
        self.init_learning_rate = learning_rate
        self.dimension = dimension
        self.nextNeuron = self

        self.reset()
```

Soru 1: Üç Boyutlu Noktalar Kümesi

Reset fonksiyonu içerisinde nöron için boş bir ağırlık dizisi açılır. Ardından bu dizi (-0.1,0.1) aralığında “uniform” dağılımlı rastgele değerlerle doldurulur. Bu değer aralığı Haykin’in daha önce bahsedilen kitabındaki değerler sebebiyle seçilmiştir.

```
def reset(self):
    #Rastgele ağırlık fonksiyonu.
    self.weights = np.zeros((1, self.dimension), dtype=float)
    for i in range(self.dimension):
        # (-0.1, 0.1) arasında rastgele değer atama fonksiyonu
        self.weights[0][i] = 0.2*np.random.rand() - 0.1
```

Nöron sınıfında ayrıca sonraki nöronu atayan bir fonksiyon bulunmaktadır. Bu fonksiyon nöronları birbirine bağlayarak ağın çiziminde kullanılacaktır.

```
# Ağ oluşturulurken nöronlar birbirine bağlanır.
def next(self, Neuron):
    self.nextNeuron = Neuron
```

Son olarak ağırlıkların güncellendiği bir fonksiyon bulunmaktadır. Ağırlıklar güncellenirken komşuluk etkisi, adaptif öğrenme hızı, veri ve güncel ağırlıklar kullanılacaktır. Komşuluk etkisi için kazanan nöronun (x,y) indisleri kullanılır. Güncellenecek nöronun kazanan nörona uzaklığı için (x,y) farkına göre uzaklıkların bulunduğu bir sözlükten faydalanılır. İterasyon bilgisi içeren adaptif öğrenme hızı ve komşuluk etkisi standart sapması da bu fonksiyonun dışında hesaplanıp fonksiyona girilir.

```
# Eğitim esnasında kazanan nöronun (x,y) indislerine göre ağdaki tüm nöronlar güncellenir.
def update(self, winner_x, winner_y, dev, data, learning_adaptive):
    key = (abs(self.x - winner_x), abs(self.y - winner_y))
    # Güncelleme esnasında kazanana uzaklığa ve eğitim iterasyonuna göre bir komşuluk etkisi hesaplanır.
    # Bu etki normal dağılımlıdır ve özdüzenleme aşamasında her iterasyonda daralmaktadır.
    dist = distDict[key]
    self.hdist = math.exp(-(dist**2)/(2*(dev**2)))
    # Öğrenme hızı da özdüzenleme aşamasında her iterasyonda azalmaktadır. Adaptif bir terimin başlangıçtaki
    # öğrenme hızıyla çarpımıyla bulunur.
    learning_rate = self.init_learning_rate*learning_adaptive
    # print(self.weights, self.x, self.y)
    # Ağırlıklar, veriye yaklaşacak şekilde güncellenir. Kazanan nöron ve yakınındakiler daha fazla yaklaşmaktadırlar.
    self.weights = self.weights + learning_rate*self.hdist*(data - self.weights)
    # print(self.weights, self.x, self.y)
```

Verilerin oluşumu:

Veriler “p1data” klasöründe kaydedilip tekrar kullanılacaktır. Eğer önceden oluşturulmuşsa, veriler bu klasörden çekilir. Aksi takdirde yeni veriler oluşturulur.

```
__location__ = os.path.realpath(os.path.join(os.getcwd(), os.path.dirname(__file__)))
dataFile1 = os.path.join(__location__, 'p1data/Data1.pkl')
dataFile2 = os.path.join(__location__, 'p1data/Data2.pkl')
dataFile3 = os.path.join(__location__, 'p1data/Data3.pkl')
dim = 3
```

Soru 1: Üç Boyutlu Noktalar Kümesi

Oluşturulan veriler üç boyutlu olup medyan ve standart sapmaları sırasıyla (2,1,2) – 0.4, (2,1,1) – 0.1, (1,1,2) – 0.3 seçilmiştir.

```
# Önceden oluşturulmuş verileri çeker. Eğer veri bulunmuyorsa
# 3 boyutlu, normal dağılımlı üç farklı noktalar kümesi oluşturur.
# Her kümede 200 nokta vardır. Kümelerde noktalar normal dağılımlıdır.
if os.path.exists(dataFile1):
    dList1 = loadList(dataFile1)
else:
    # Birinci kümede medyan (2,1,2) ve standart sapma 0.4'tür.
    dList1 = generate_normal_data(dim, 200, [2, 1, 2], 0.4)
    saveData(dList1, dataFile1)

if os.path.exists(dataFile2): ···
else:
    # ikincide medyan (2,1,1) ve standart sapma 0.1'dir.
    dList2 = generate_normal_data(dim, 200, [2, 1, 1], 0.1)
    saveData(dList2, dataFile2)

if os.path.exists(dataFile3):
    dList3 = loadList(dataFile3)
else:
    # Üçüncü kümede medyan (1,1,2) ve standart sapma 0.3'tür.
    dList3 = generate_normal_data(dim, 200, [1, 1, 2], 0.3)
    saveData(dList3, dataFile3)
```

Veriler “generate_normal_data” fonksiyonu ile oluşturulmaktadır. Bu fonksiyon, girilen medyan etrafında, girilen standart sapmada normal dağılımlı, girilen boyutta ve girilen sayıda veri oluşturup bunları bir listede toplar.

```
# Normal dağılımlı veri oluşturma fonksiyonu.
def generate_normal_data(dimension, size, mean, deviation):
    dList = []
    # Girilen boyutta, girilen sayı kadar veri oluşturur. Bu verilerin medyanları ve standart
    # sapmaları da fonksiyona girilen büyüklüklerdedir. Daha sonra verilerden oluşan listeyi döndürür.
    for i in range(size):
        data = np.zeros(dimension, dtype=float)
        for j in range(dimension):
            data[j] = np.random.normal(mean[j], deviation)
        dList.append(data)
    return dList
```

Soru 1: Üç Boyutlu Noktalar Kümesi

Bunun ardından daha önce oluşturulan eğitim ve test kümeleri yüklenir. Eğer oluşturulmamışsa, üç veri kümesi yarı yarıya eğitim ve test kümelerine dağıtılır.

```
# Eğitim ve test sırasında kullanılacak veri kümelerinin dağılımı.
trainingFile = os.path.join(__location__, 'p1data/Training.pkl')
testingFile = os.path.join(__location__, 'p1data/Testing.pkl')
if os.path.exists(trainingFile):
    trainingList = loadList(trainingFile)
    testingList = loadList(testingFile)
else:
    trainingList = []
    testingList = []

    for i in range(0, len(dList1)):
        if i < len(dList1)/2:
            trainingList.append(dList1[i])
        else:
            testingList.append(dList1[i])
    for i in range(0, len(dList2)):
        if i < len(dList2)/2:
            trainingList.append(dList2[i])
        else:
            testingList.append(dList2[i])
    for i in range(0, len(dList3)):
        if i < len(dList3)/2:
            trainingList.append(dList3[i])
        else:
            testingList.append(dList3[i])

    saveData(trainingList, trainingFile)
    saveData(testingList, testingFile)
```

Eğitim hazırlıkları:

Eğitimde özdüzenleme ve yakınsama aşamalarının süreceği iterasyon sayıları seçilir. Haykin özdüzenleme için 1000, yakınsama için nöron sayısının 500 katı olmasını tavsiye ettiği için problemin devamı boyunca yakınsama iterasyonu, özdüzenleme iterasyonunun yarısının nöron sayısıyla çarpımı seçilecektir.

```
# Eğitimin süreceği toplam iterasyon sayısı Haykin'in "Neural Networks - A Comprehensive Foundation" kitabındaki tavsiyesi
# özdüzenlemenin 1000, yakınsama aşamasının toplam nöron sayısının 500 katı olduğu yönündedir.
# Ardından, komşuluk etkisinin ve adaptif öğrenme hızının zamanla eksponansiyel olarak azalması için zaman sabitleri belirlenir.
# Bu değerler aynı kitabın 474-475 sayfalarında Haykin'in söylediği şekilde alınır.
epoch = 1000
conv_epoch = epoch/2
```

Ağ boyutu 6’ya 6, başlangıç öğrenme hızı 0.1 seçilir. Adaptif öğrenme hızı ve komşuluk etkisi için kullanılacak olan zaman sabitleri de Haykin’in tavsiye ettiği gibi seçilir.

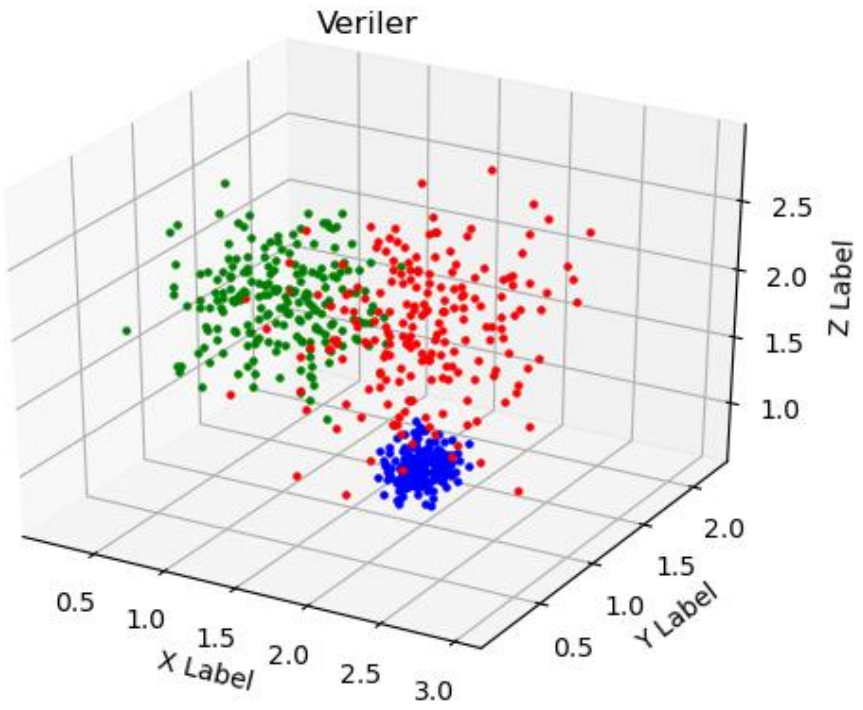
Soru 1: Üç Boyutlu Noktalar Kümesi

```
# 6'ya 6 boyutlu toplam 36 nörondan oluşan, başlangıçtaki öğrenme hızı 0.1 olan bir ağ oluşturulacak.
xdim = 6
ydim = 6
learning_rate = 0.1

tcons1 = epoch/math.log(math.sqrt(xdim**2 + ydim**2))
tcons2 = epoch
```

Ağ boyutu kullanılarak nöronlar arasındaki uzaklıkların bulunduğu bir sözlük hazırlanır. Bu sayede her seferinde uzaklık hesaplanmayacak, indisleri arasındaki farklarla sözlükteki değer seçilecektir.

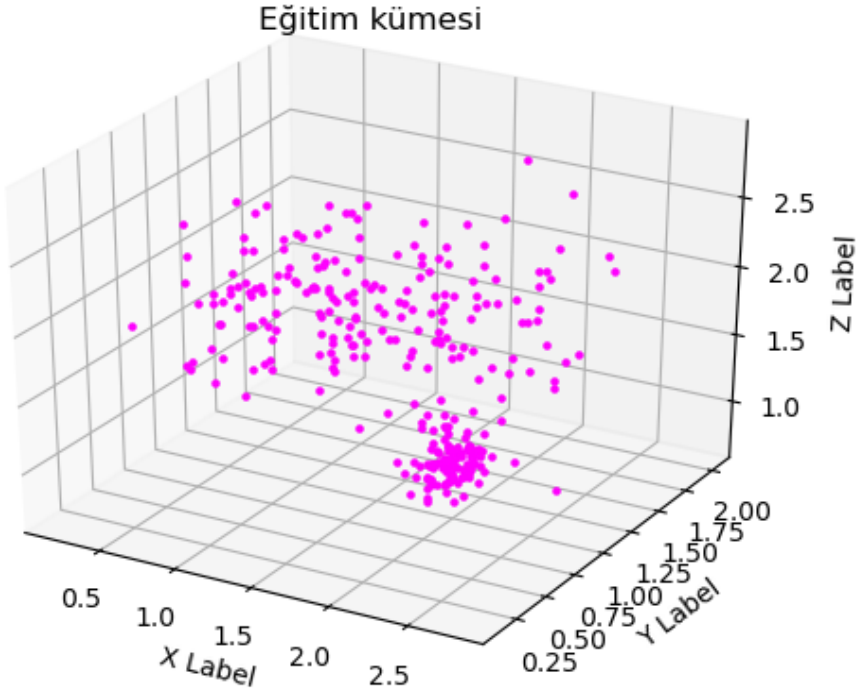
```
# Eğitimde kullanmak üzere, uzaklıkların bulunduğu bir kütüphane oluşturulur.
distDict = {}
for i in range(xdim):
    for j in range(ydim):
        dist = math.sqrt(i**2 + j**2)
        # 0.1 standart sapmaya sahip normal dağılımı. Farklı nöronların komşuluk derecesini
        # ve kazanan nörona göre ağırlıklarının ne kadar değişeceğini belirler.
        distkey = (i, j)
        distDict[distkey] = dist
```

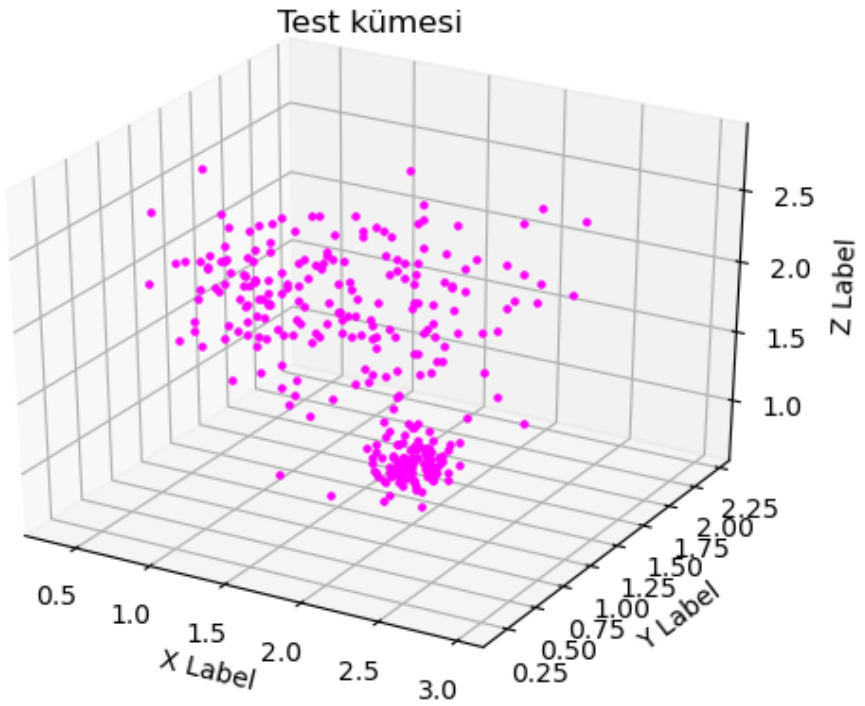
Oluşturulan veri, test ve eğitim kümeleri şekildeki gibidir.



Soru 1: Üç Boyutlu Noktalar Kümesi

Eğitim ve test kümelerinde sınıf bilgisi olmadığı için tüm veriler aynı renkte çizilmiştir. (epoch = 1000 değerindeki eğitim kümesinin ölçeklenmesinde bir sorun yaşandığı için epoch = 50 değerindeki grafikler kullanılacaktır.)





Soru 1: Üç Boyutlu Noktalar Kümesi

Çizim için kullanılan fonksiyonlar:

Sadece veriler çizilirken “plot3ddata” kullanılır. Bu fonksiyonda üç farklı veri kümesi girilip farklı renklerde veya tek veri kümesi girilip mor/pembe renkte çizdirilebilir.

```
def plot3ddata(title, dList1, dList2 = [], dList3 = [], single_list = True):
    # Girilen listedeki tüm verileri scatter plot şeklinde çizdirir. Tek liste girilmişse mor/pembe bir renkte
    # çizim yapar, üç farklı liste girilmişse listedeki verileri kırmızı, mavi ve yeşil renklerle çizer.
    fig = plt.figure()
    ax = fig.add_subplot(111, projection='3d')
    ax.set_title(title)
    if single_list == False:
        for data in dList1:
            ax.scatter(data[0], data[1], data[2], s=5, c="red", depthshade=True)
        for data in dList2:
            ax.scatter(data[0], data[1], data[2], s=5, c="blue", depthshade=True)
        for data in dList3:
            ax.scatter(data[0], data[1], data[2], s=5, c="green", depthshade=True)
    else:
        for data in dList1:
            ax.scatter(data[0], data[1], data[2], s=5, c="magenta", depthshade=True)

    ax.set_xlabel('X Label')
    ax.set_ylabel('Y Label')
    ax.set_zlabel('Z Label')
```

Ağın çizimi biraz daha karmaşıktır. Öncelikle nöronların ağırlıkları nokta halinde çizdirilir.

```
def plotnetwork(network, xdim, ydim, title):
    # Ağ çizimini yapan fonksiyon. Öncelikle ağdaki nöronları ağırlıklarına göre scatter plot şeklinde çizdirir.
    fig = plt.figure()
    ax = fig.add_subplot(111, projection='3d')
    ax.set_title(title)
    for Neuron in network:
        weights = Neuron.weights[0]
        ax.scatter(weights[0], weights[1], weights[2], s=10, c="black", depthshade=True)
```

Ardından nöronlar arasındaki bağlar çizdirilir. (0,0) nöronundan (1,0) ve (0,1) nöronuna bağ çizilir. Bu, sınırlardaki nöronlar haricinde tüm nöronlarda uygulandıktan sonra bağlar çizilmiş olur. Bu çizimler yapılırken ağ oluşturulurken nöronlara atanan sonraki nöronlar kullanılır. Ağ oluşturulurken önce y indisindeki nöronlar atanıp, daha sonra x indisi kullanıldığı için (x,y) nöronu ile (x+1,y) nöronu arasındaki bağın çizimi için (x,y) nöronunun y boyutu kadar sonrasındaki nörona ulaşılır. Bu nöron (x+1,y) nöronu olacaktır. (x,y) nöronu ile (x,y+1) nöronu arasındaki bağ için ise nörondan hemen sonraki nörona ulaşılır.

Soru 1: Üç Boyutlu Noktalar Kümesi

```
# Ardından yanındaki nöronlarla olan bağını çizer. Bunun için bir nörondan iki farklı çizgi çizdirilir.
for Neuron in network:
    # Birinci çizgi hemen sonraki nörona çizilir. (x,y) nöronundan (x,y+1) nöronuna olan çizgidir.
    weights = Neuron.weights[0]
    nextNeuron = Neuron.nextNeuron
    next_weights1 = nextNeuron.weights[0]

    xcord1 = np.linspace(weights[0], next_weights1[0], 5)
    ycord1 = np.linspace(weights[1], next_weights1[1], 5)
    zcord1 = np.linspace(weights[2], next_weights1[2], 5)

    # İkincisi ise y boyutu kadar sonraki nöronla arasındaki çizgidir. (x,y) nöronundan (x+1,y) nöronuna
    # olan çizgidir. Bunun sebebi, nöronlar sıralanırken önce y değerlerine göre, sonra x değerlerine göre sıralanmalarıdır.
    nextNeuron = Neuron
    for i in range(ydim):
        nextNeuron = nextNeuron.nextNeuron
    next_weights2 = nextNeuron.weights[0]

    xcord2 = np.linspace(weights[0], next_weights2[0], 5)
    ycord2 = np.linspace(weights[1], next_weights2[1], 5)
    zcord2 = np.linspace(weights[2], next_weights2[2], 5)
    # x ve y boyutlarına göre sınırda olan nöronlardan çizgi çizdirilmez.
    if Neuron.y != ydim - 1:
        ax.plot(xcord1, ycord1, zcord1, linewidth=1.2, c='0.12')
    if Neuron.x != xdim - 1:
        ax.plot(xcord2, ycord2, zcord2, linewidth=1.2, c='0.12')

ax.set_xlabel('X Label')
ax.set_ylabel('Y Label')
ax.set_zlabel('Z Label')
```

“plotnetworkanddata” fonksiyonu ise iki fonksiyonun birleşimidir. Aynı figür üzerinde çizilmesi için iki fonksiyon ayrı ayrı kullanılamamaktadır.

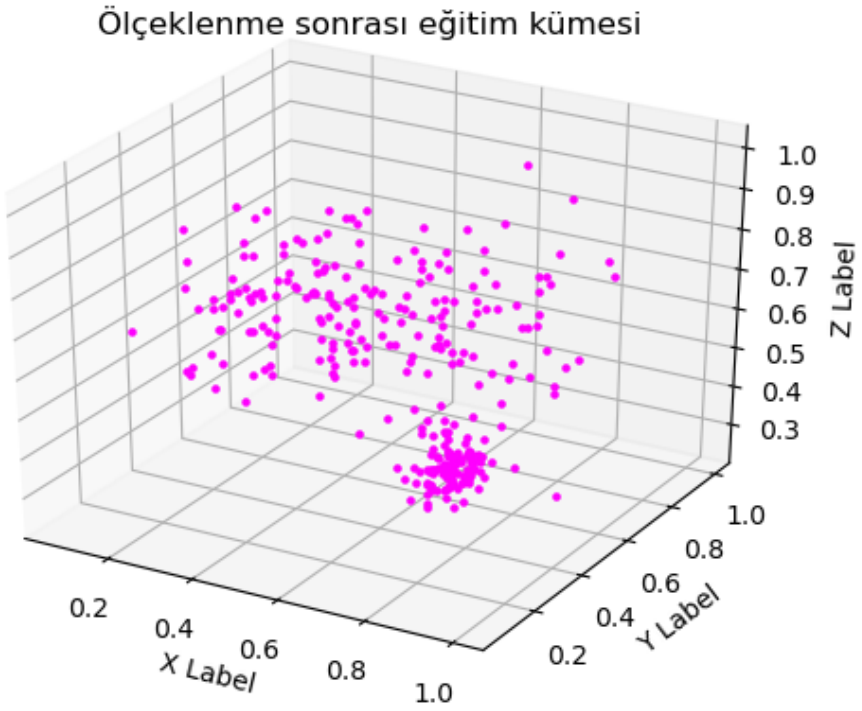
```
# Verilerin çizimini yapan fonksiyonlar
def plotnetworkanddata(network, xdim, ydim, title, dList1, dList2 = [], dList3 = [], single_list = True):···
```

Soru 1: Üç Boyutlu Noktalar Kümesi

Eğitim hazırlıkları devamı:

Eğitim kümesi eğitime girmeden önce (0,1) aralığında ölçeklenir. Ölçeklenme öncesindeki minimum ve maksimum değerleri geri ölçekleme için saklanır.

```
normalized_training = normalize_List(trainingList, dim, np.zeros(dim), np.ones(dim))
trainingList_normal = normalized_training[0]
maxL_training = normalized_training[1]
minL_training = normalized_training[2]
```



Eğitim üç kere tekrarlanıp özdüzenleme aşaması, yakınsama aşaması ve toplam eğitim süresinin ortalaması alınıp kaydedilecektir. Eğitim sonrasında daha önce elde edilen minimum ve maksimum değerleriyle ağ üzerinde geri ölçekleme yapılır.

```
# Belirlenen değerlerle eğitim başlatılır.
total_ordering_time = 0
total_convergence_time = 0
for j in range(3):
    print("test: %d" % (j+1))

    # Eğitilecek ağ, verilen değerlere göre oluşturulur ve eğitim başlatılır.
    network = generate_network(xdim, ydim, dim, learning_rate)
    [ordering_time, convergence_time] = education_process(network, trainingList_normal, epoch, conv_epoch, xdim, ydim, tcons1, tcons2, current_result_folder)
    total_ordering_time += ordering_time
    total_convergence_time += convergence_time

    # # Eğitim kümesindeki verilerin ve Kohonen Ağı'nın ağırlıklarının geri ölçeklemesi. Veriler orjinal boyutlarına döner.
    # for data in trainingList:
    #     data = denormalize_data(data, dim, np.zeros(dim), np.ones(dim), minL_training, maxL_training)

    for Neuron in network:
        Neuron.weights[0] = denormalize_data(Neuron.weights[0], dim, np.zeros(dim), np.ones(dim), minL_training, maxL_training)
```

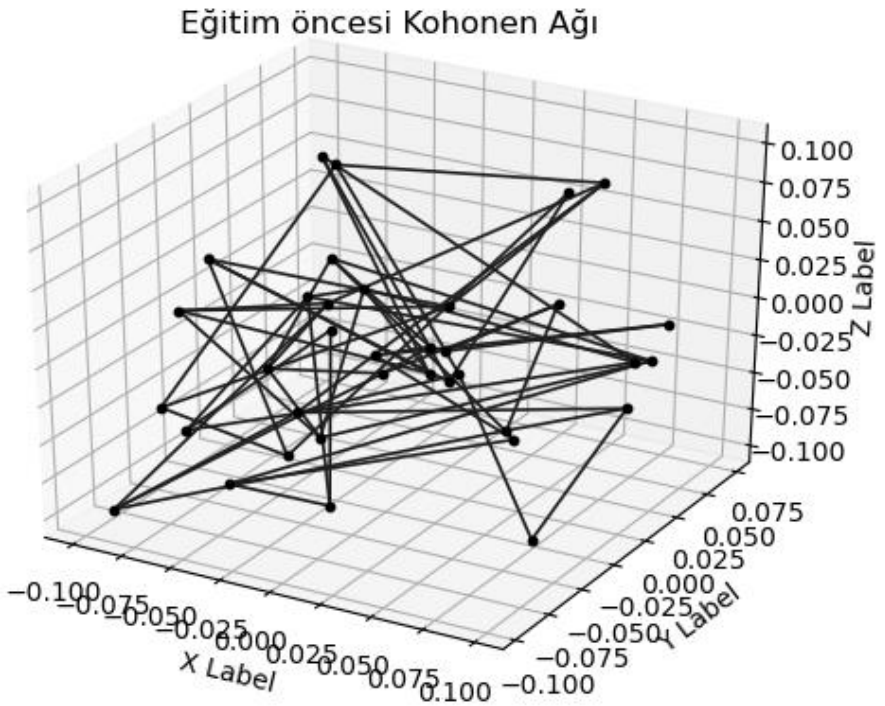
Soru 1: Üç Boyutlu Noktalar Kümesi

Eğitim:

İlk olarak özdüzenleme aşaması gerçekleşir. Her iterasyonun başında eğitim listesi karıştırılır. Her iterasyonda adaptif öğrenme hızı terimi ve komşuluk etkisi standart sapması hesaplanarak eğitimde kullanılır.

```
def education_process(network, dList, epoch, conv_epoch, xdim, ydim, tcons1, tcons2, current_result_folder):
    plotnetwork(network, xdim, ydim, "Eğitim öncesi Kohonen Ağı")
    plt.savefig(current_result_folder + '/fig5.png')
    # plt.show()
    plt.close()
    # Özdüzenleme aşaması. "epoch" iterasyonu kadar sürecektir.
    start_time = time.time()
    for i in range(epoch):
        # Her iterasyonun başında eğitim listesi karıştırılır.
        random.shuffle(dList)
        # dev: Eğitimin bulunduğu iterasyondaki komşuluk etkisi standart sapması. Her iterasyonda eksponansiyel
        # olarak küçülmekte olup sonraki iterasyonların ağ üzerindeki etksini azaltır.
        # learning_adaptive: Adaptif öğrenme hızı. Öğrenme hızı da her iterasyonda eksponansiyel olarak azalmaktadır.
        dev = math.exp(-(i/tcons1))
        learning_adaptive = math.exp(-(i/tcons2))

        # Listedeki her veri için ağ eğitime girer.
        for data in dList:
            education(data, network, dev, learning_adaptive)
    ordering_time = (time.time() - start_time)
    print("Özdüzenleme aşaması süresi: %f" % ordering_time)
```

Eğitimden önce ağ iç içe geçmiş bir haldedir. Ağırlıklar rastgele dağıtıldığı için herhangi bir düzen gözlenmez.



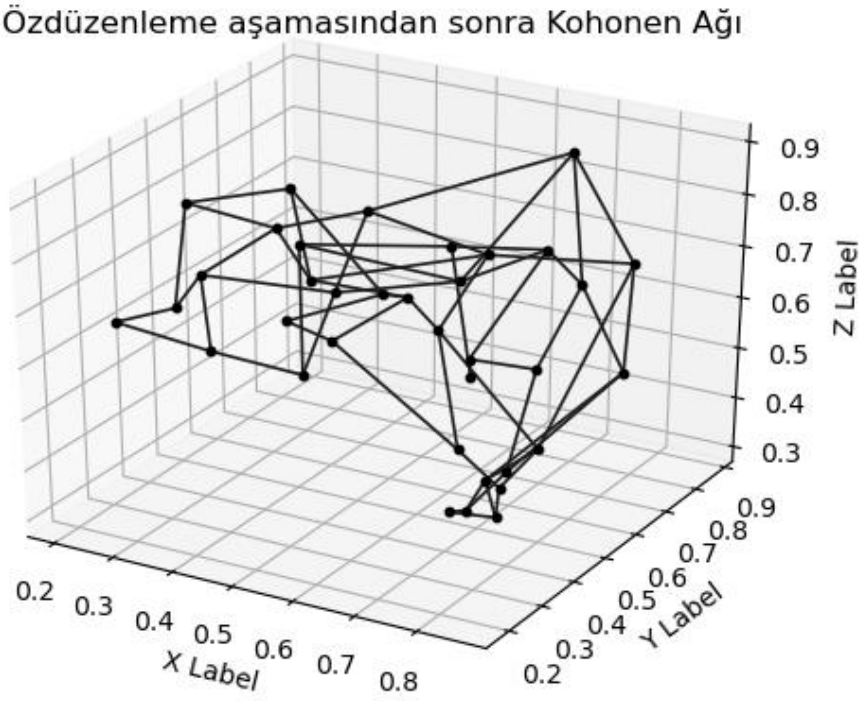
Soru 1: Üç Boyutlu Noktalar Kümesi

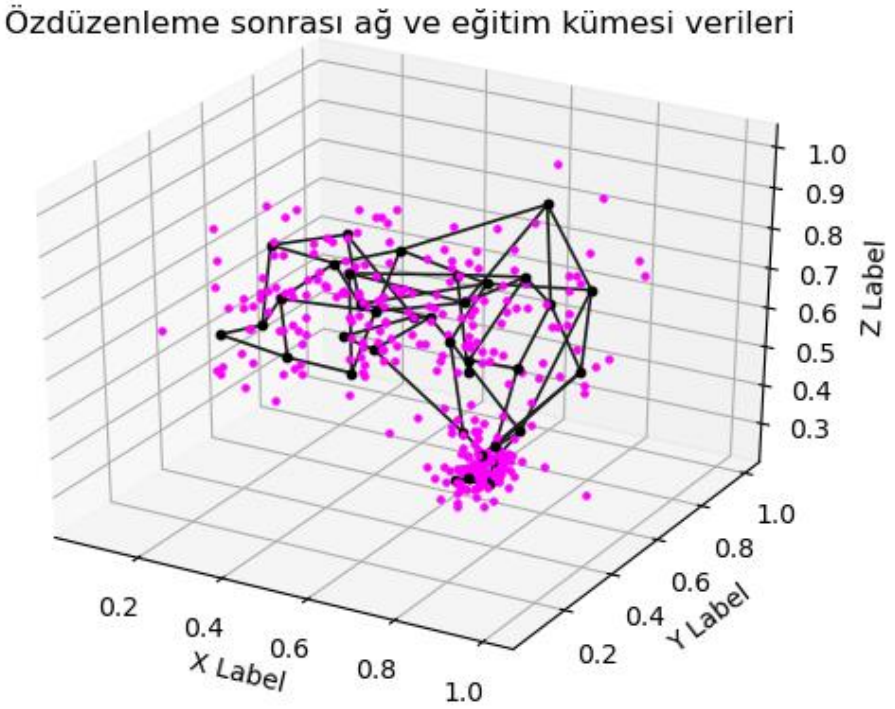
Eğitim sırasında kazanan nöronun belirlenmesi için veri ile nöron ağırlıkları karşılaştırılır. Veriye en yakın nöron kazanan nöron seçilir ve (x,y) indisleri kaydedilir. Ardından ağdaki tüm nöronların ağırlıkları kazanan nöronun indisine göre güncellenir. Kazanan nörona yakın nöronların ağırlıkları, veriye daha çok yaklaşacaktır.

```python
def education(data, network, dev, learning_adaptive):
    # Eğitim için öncelikle kazanan belirlenir. Kazananın indislerine göre ağdaki nöronlar güncellenir.
    winner = 999
    winner_x = 0
    winner_y = 0
    for Neuron in network:
        current = 0
        # Kazananın belirlenmesi için veri ile nöronların ağırlıkları arasındaki uzaklık ölçülür ve en düşük olan seçilir.
        for i in range(len(Neuron.weights[0])):
            current += (Neuron.weights[0][i] - data[i])**2
        # Kazananın indisleri kaydedilir.
        if current < winner:
            winner = current
            winner_x = Neuron.x
            winner_y = Neuron.y

    #     print("kazanan nöron uzaklığı", winner)
    #     print("kazanan nöron indisleri, x: %d, y: %d" % (winner_x, winner_y))
    #     print("_____________________________________________________")
    # print("kazanan indisler", winner_x, winner_y)

    # Kazananın indislerine göre ağdaki her nöron güncellenir. Bu işlem için veri, güncel iterasyondaki komşuluk
    # etkisi ve adaptif öğrenme hızı da kullanılır.
    for Neuron in network:
        # print("Nöron indisleri: x: %d, y: %d" % (Neuron.x, Neuron.y))
        # print("Güncelleme öncesi ağırlıklar:", Neuron.weights[0])
        Neuron.update(winner_x, winner_y, dev, data, learning_adaptive)
        # print("Güncelleme sonrası ağırlıklar:", Neuron.weights[0])
        # print("_____________________________________________________")
```

Özdüzenleme sonrasında ağ, Haykin'in örneğindeki kadar düzenli olmasa da gayet düzenli bir yapıdadır. Eğitilen verilerin oluşturduğu öbeklere yaklaştığı açık bir şekilde gözlenebilir. Standart sapması yüksek olan kümelerdeki nöronlar daha seyrekken, standart sapması 0.1 olan kümede gayet yakınlardır. Nöronlar bir şekilde verilerin dağılımını taklit ederler.

Soru 1: Üç Boyutlu Noktalar Kümesi



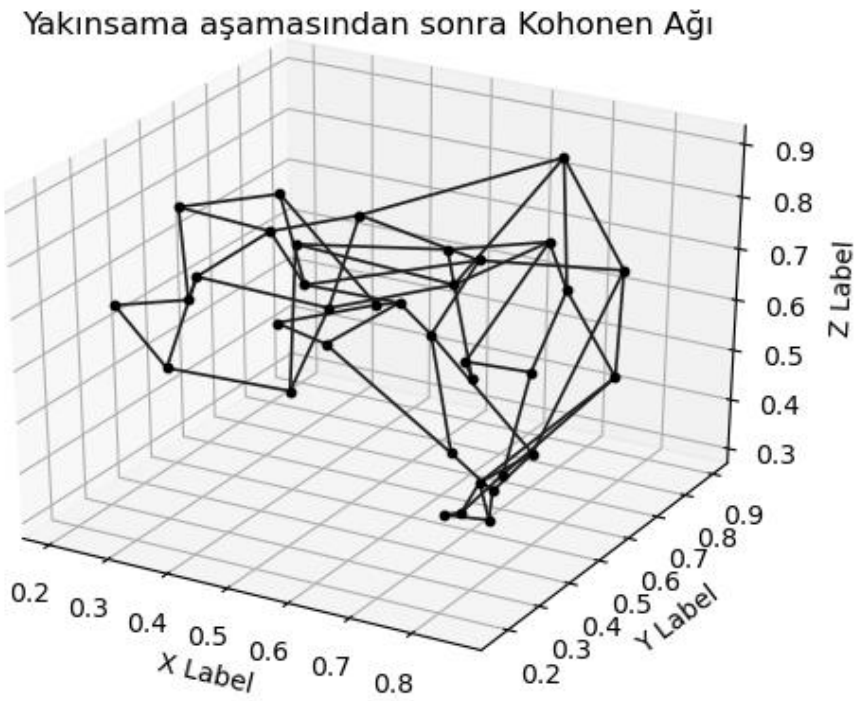


Soru 1: Üç Boyutlu Noktalar Kümesi

Özdüzenleme aşamasının ardından yakınsama aşaması gerçekleşir. Bu aşamada komşuluk etkisi standart sapması ve adaptif öğrenme hızı terimi sabit kalacaktır.

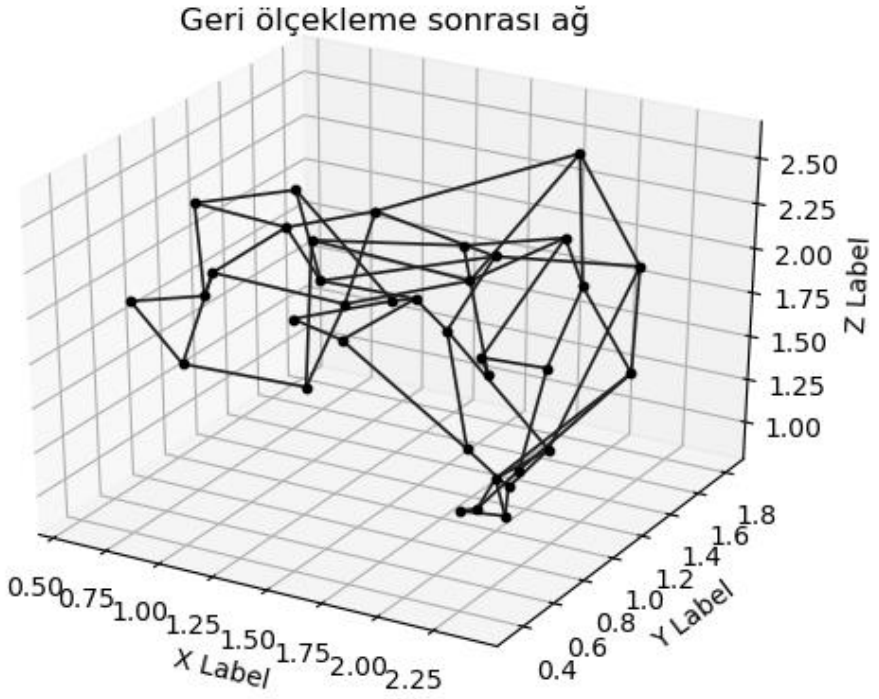
```
# Yakınsama aşamasında ise komşuluk etkisi ve öğrenme hızı sabitken ağ eğitilir. Haykin, kitabında bu aşama
# için nöron sayısının 500 katı iterasyon sürmesi gerektiğini söylemiştir.
dev = math.exp(-((epoch-1)/tcons1))
learning_adaptive = math.exp(-((epoch-1)/tcons2))
start_time2 = time.time()
for i in range(round(conv_epoch*len(network))):
    random.shuffle(dList)
    for data in dList:
        education(data, network, dev, learning_adaptive)
convergence_time = (time.time() - start_time2)
print("Yakınsama aşaması süresi: %f" % convergence_time)
print("Eğitim süresi: %f" % (ordering_time + convergence_time))
plotnetwork(network, xdim, ydim,  "Yakınsama aşamasından sonra Kohonen Ağı")
plt.savefig(current_result_folder + '/fig8.png')
# plt.show()
plt.close()

return [ordering_time, convergence_time]
```

Yakınsama aşaması sonrası ağda daha detaylı değişimler gözlenmektedir.

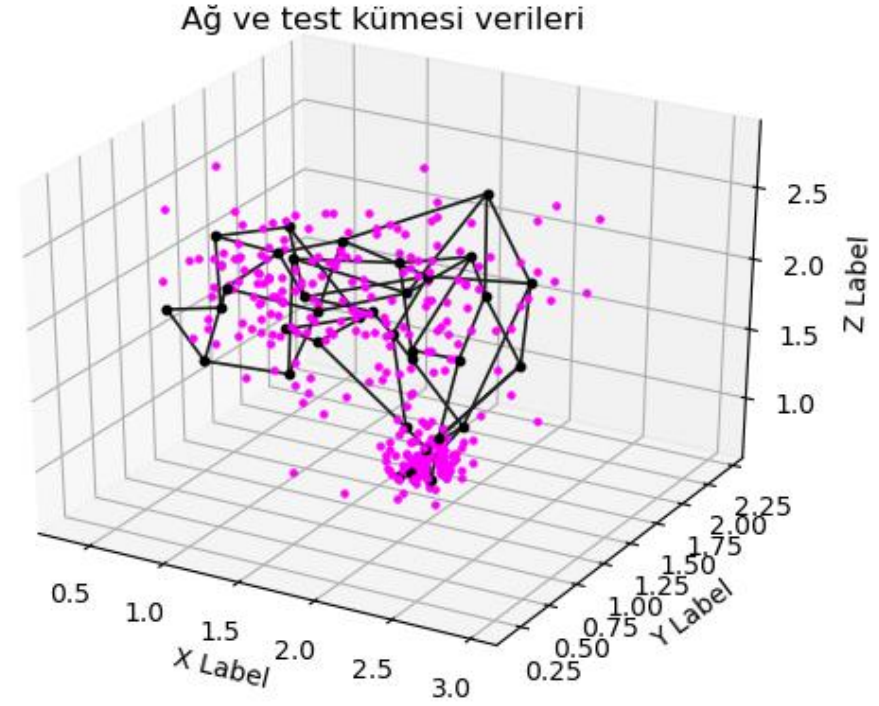


Soru 1: Üç Boyutlu Noktalar Kümesi

Ardından ağ geri ölçeklenerek veri boyutlarına döndürülür.

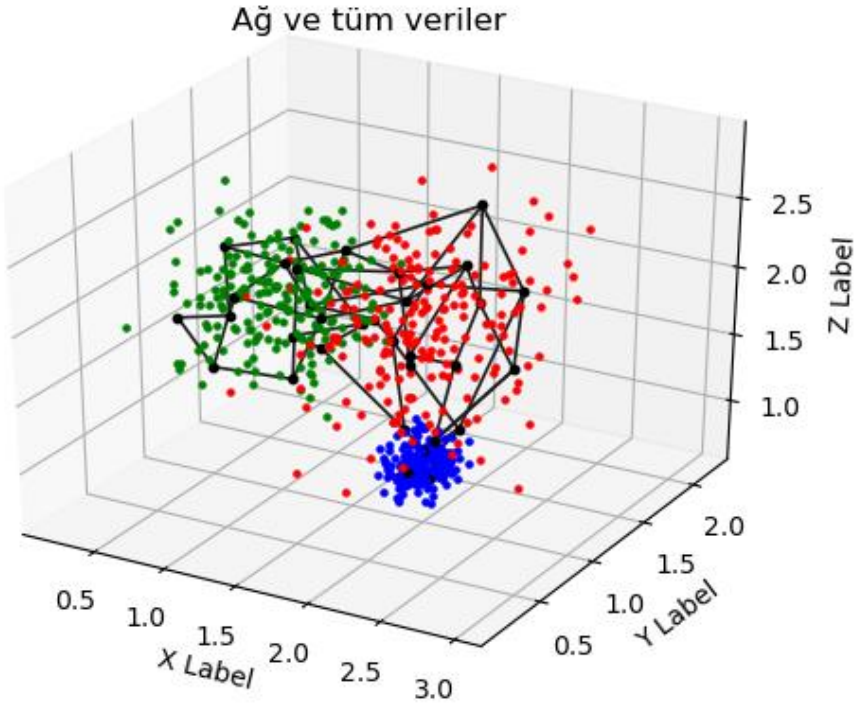


Test kümesiyle karşılaştırıldığında da nöronların öbeklerde yoğunlaştığı görülebilir.,



Soru 1: Üç Boyutlu Noktalar Kümesi

Tüm verilerle karşılaştırıldığında, ağdaki bazı nöronların veri kümelerinin iç içe olduğu konumlarda bulunduğu görülür. Ağ, öbekleri ifade etse de verilerin iç içe olduğu durumları ayırt edememektedir.



Maalesef, eğitim kümesi listesi ölçeklendirme fonksiyonun bu global eğitim kümesini etkilemiştir. Bu sebeple diğer testlerdeki verilerden elde edilen ağlar geri ölçeklenememiş, test kümesiyle karşılaştırılamaz haldedirler. Fakat eğitim sonucu elde edilen ağ yapılarına bakıldığında ilk testtekine benzer olduğu ve öbekleri başarılı bir şekilde ifade ettikleri görülür. Bu grafiklerin tümü “p1results” klasöründe ilgili test klasörleri altında bulunmaktadır. Yapılan testler;

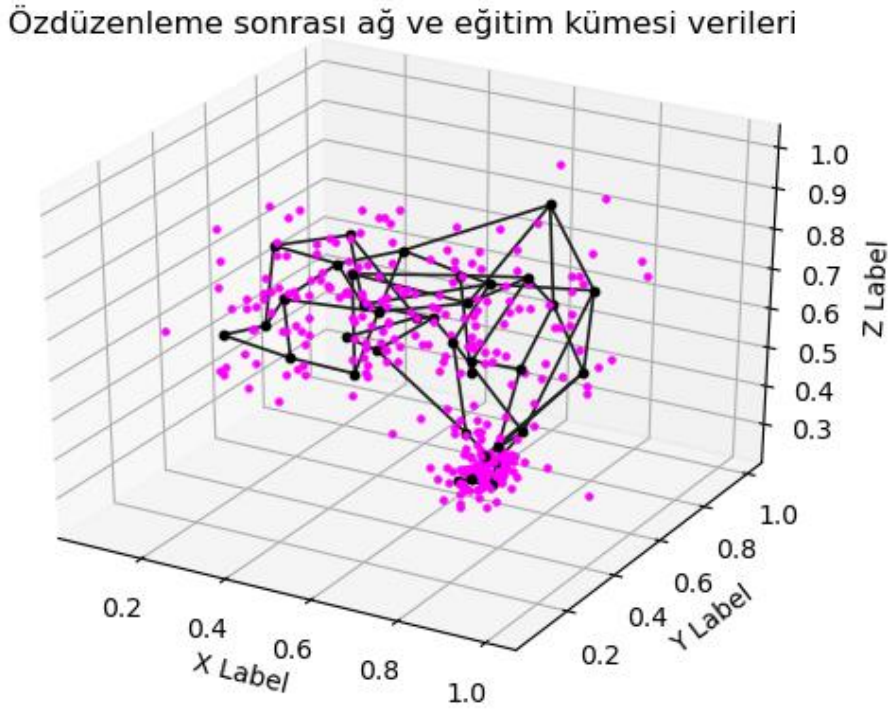
- özdüzenleme iterasyon sayısının {50, 250, 500, 750, 1000} seçilmesi,
- öğrenme hızının {0.001, 0.01, 0.1, 0.3, 0.5} seçilmesi,
- ağ boyutunun 4’e 4, 5’e 5, 6’ya 6, 7’ye 7, 8’e 8 seçilmesidir.

Aksi belirtilmedikçe özdüzenleme iterasyon sayısı 250, başlangıç öğrenme hızı 0.1 ve ağ boyutu 6’ya 6’dır. Test verileriyle olan sonuçlar bozuk olduğu için normalize eğitim kümesiyle ağın karşılaştırmaları yapılacaktır. Bu karşılaştırmalar yine aynı sebepten dolayı özdüzenleme sonrası ağ ile yapılacaktır.

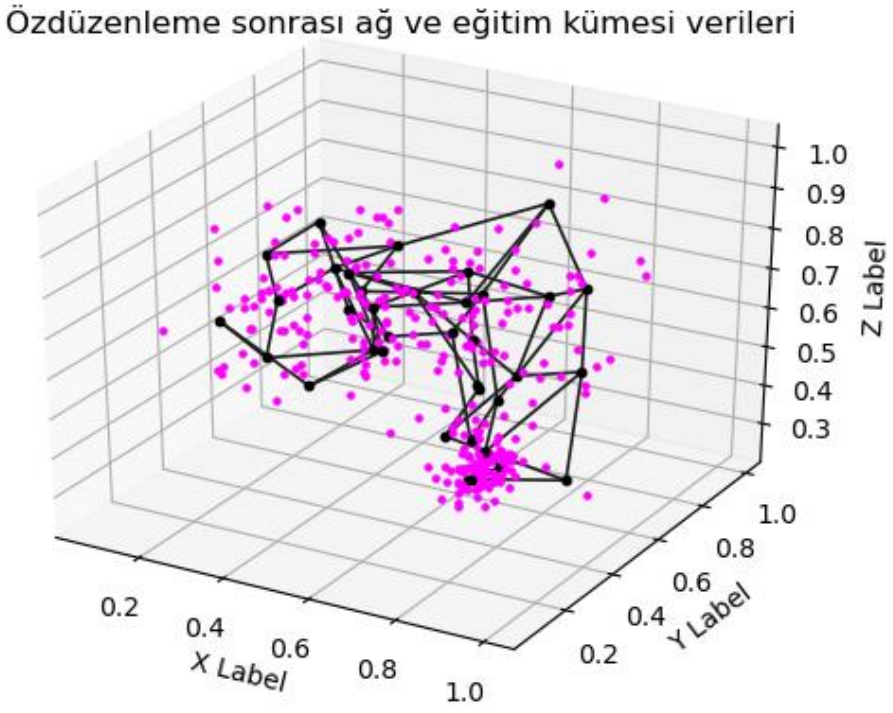
Soru 1: Üç Boyutlu Noktalar Kümesi

Özdüzenleme iterasyon sayısı testleri:

- Epoch = 50

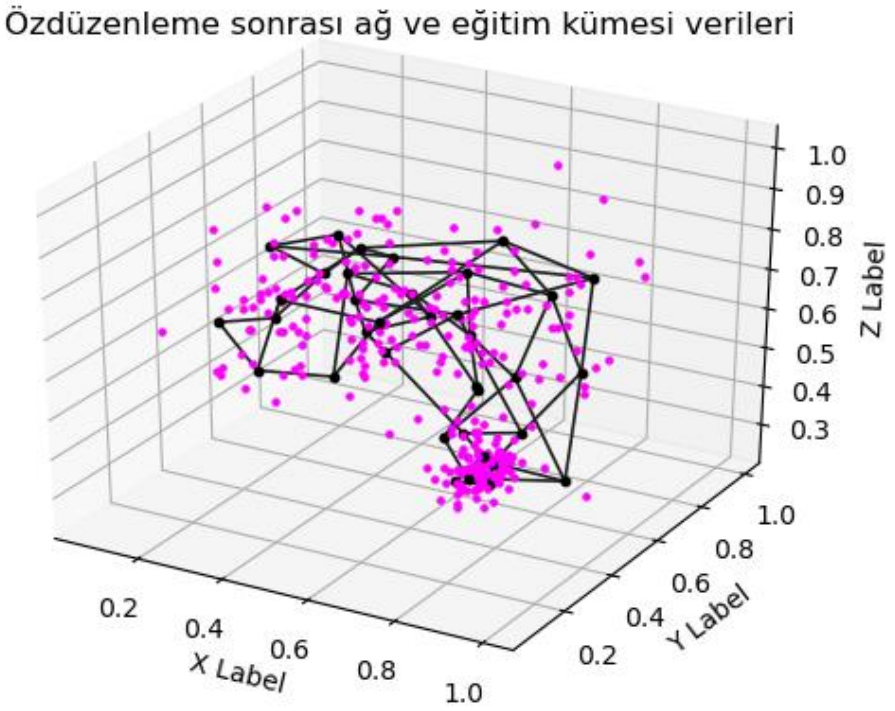


- Epoch = 250

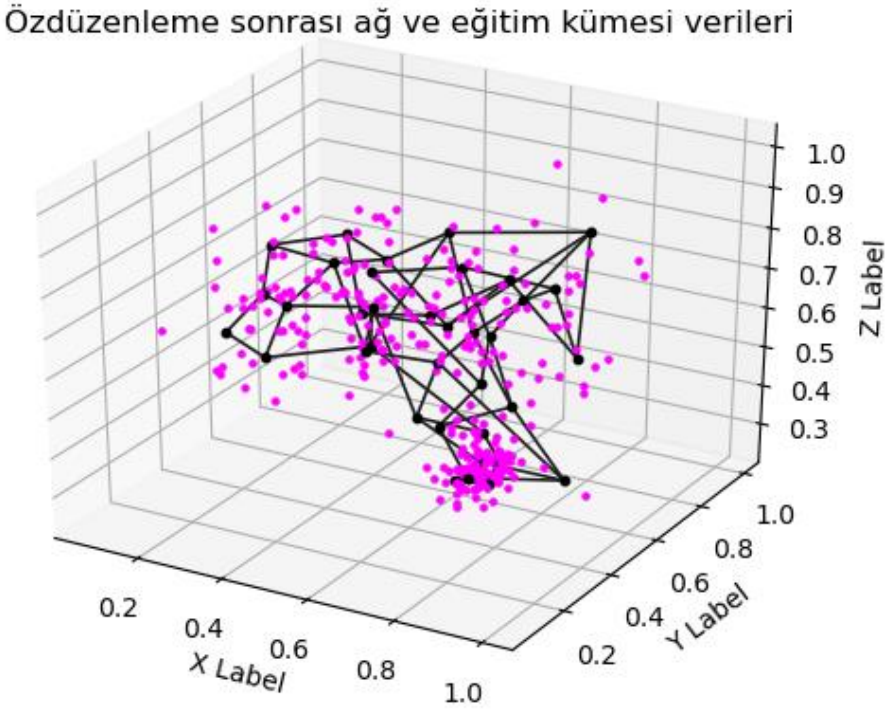


Soru 1: Üç Boyutlu Noktalar Kümesi

- Epoch = 500:

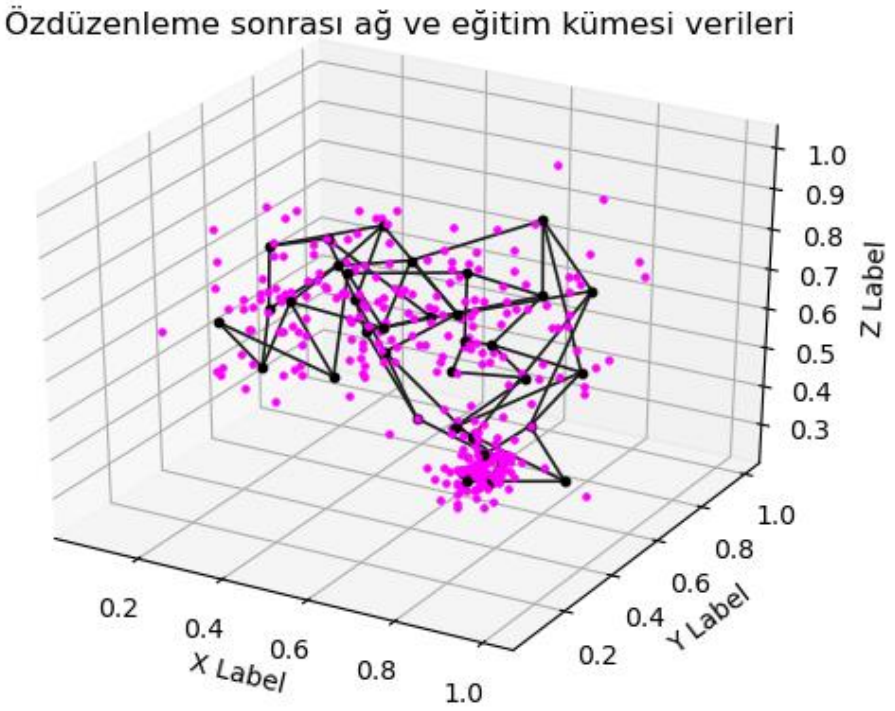


- Epoch = 750:



Soru 1: Üç Boyutlu Noktalar Kümesi

- Epoch = 1000:

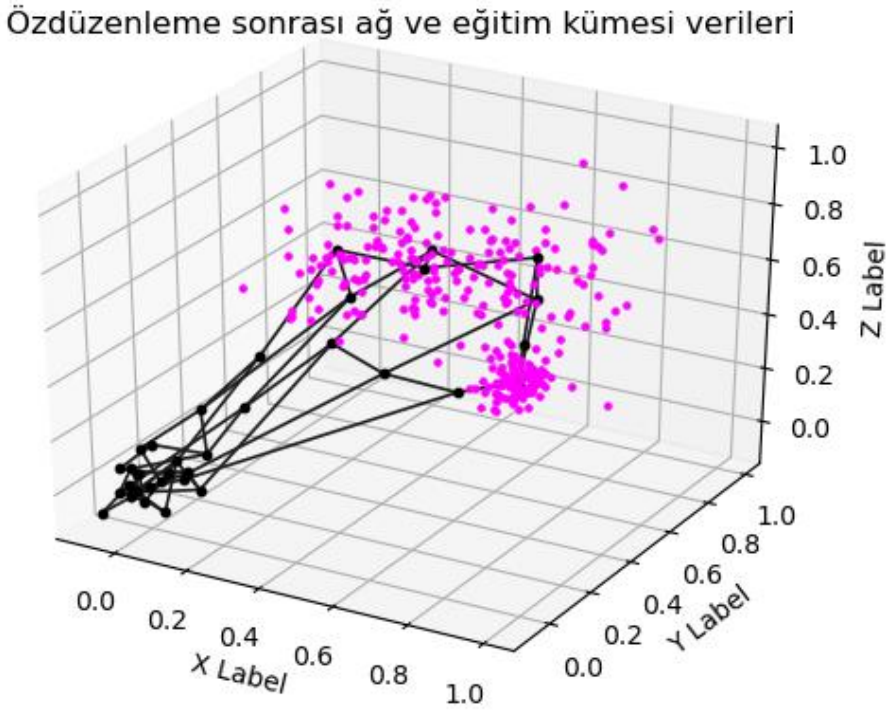


Elde edilen ağlar fazlasıyla çeşit gösterirken her seferinde öbekleri ifade etmeyi başarmıştır. Çeşitliliğin sebebi iterasyon sayısındaki farklılık yerine ağırlıkların başlangıç koşulları olabilir.

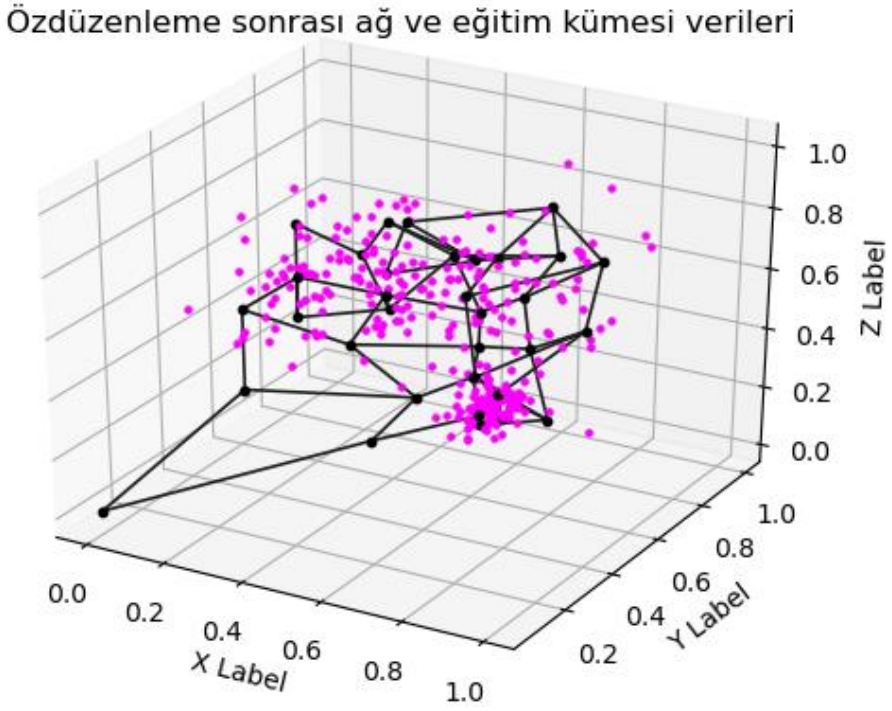
Soru 1: Üç Boyutlu Noktalar Kümesi

Başlangıç öğrenme hızı testleri:

- Öğrenme Hızı = 0.001:

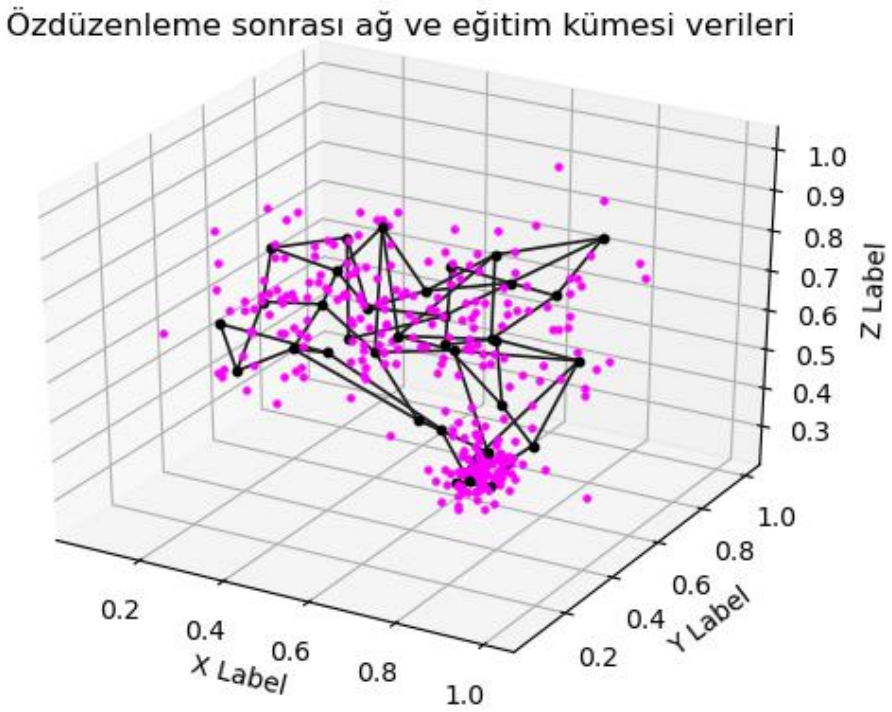


- Öğrenme Hızı = 0.01:

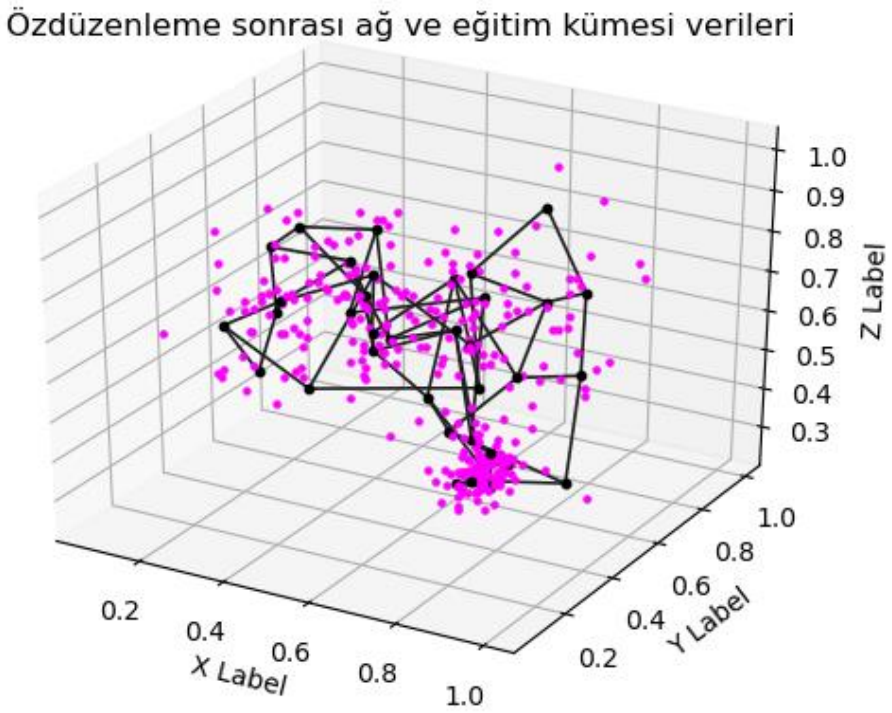


Soru 1: Üç Boyutlu Noktalar Kümesi

- Öğrenme Hızı = 0.1:

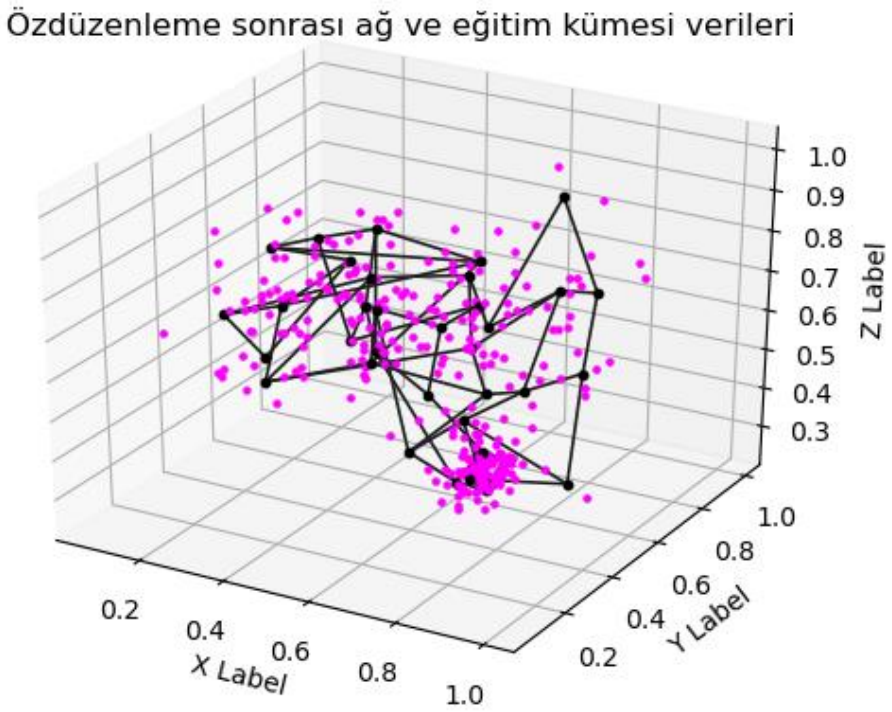


- Öğrenme Hızı = 0.3:



Soru 1: Üç Boyutlu Noktalar Kümesi

- Öğrenme Hızı = 0.5:



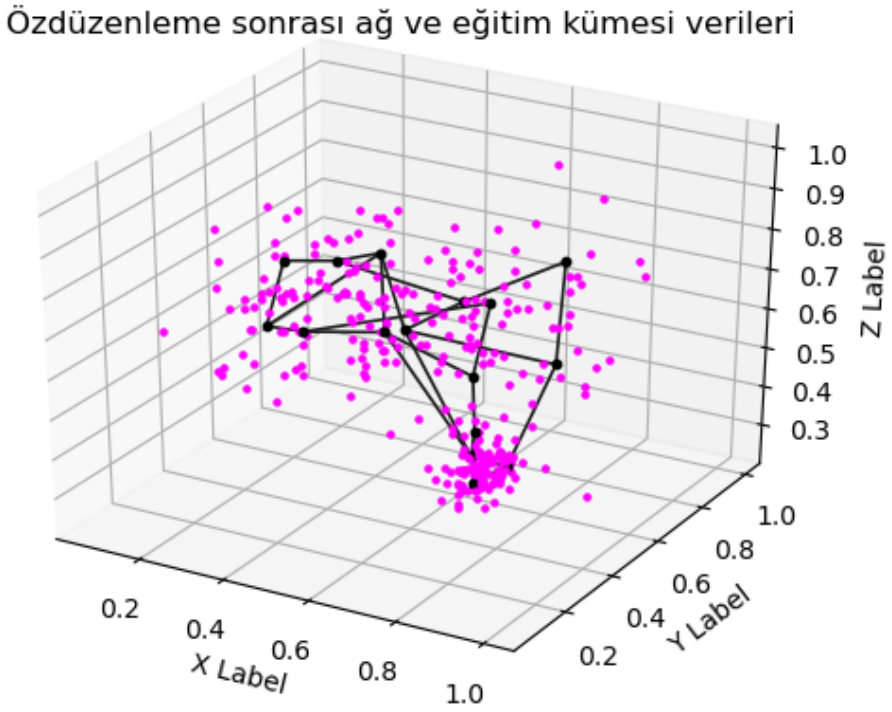
Öğrenme hızı 0.1'in altındaki değerlerde kazanamayan nöronların dışarıda kaldıkları ve kazanmamaya devam ettikleri gözlenir. Bu durum 0.001 öğrenme hızı için çok daha çarpıcı bir şekilde gözlenir. Ağın yarısından azı öbekleri temsil etmektedir. Haykin'in tavsiyesinin başarısı bu testte gözlenebilir.

Öğrenme hızı yüksek olduğunda ise ağ öbekleri temsil edebilirken, 0.1 olduğu durumdaki kadar isabetli bir temsil olmadığı gözlemlenmektedir.

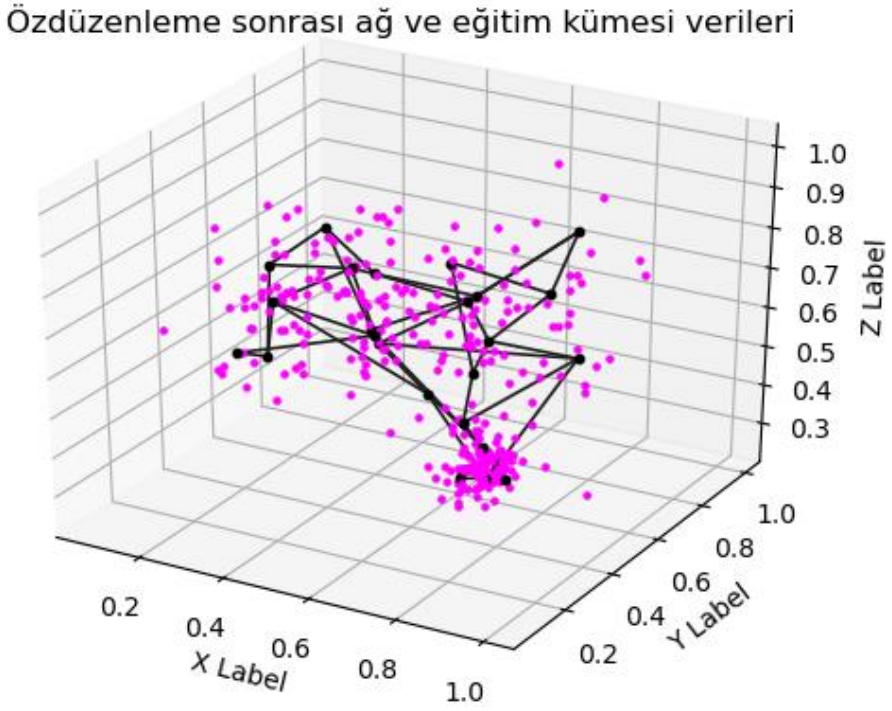
Soru 1: Üç Boyutlu Noktalar Kümesi

Ağ boyutu testleri:

- Ağ boyutu = 4x4:

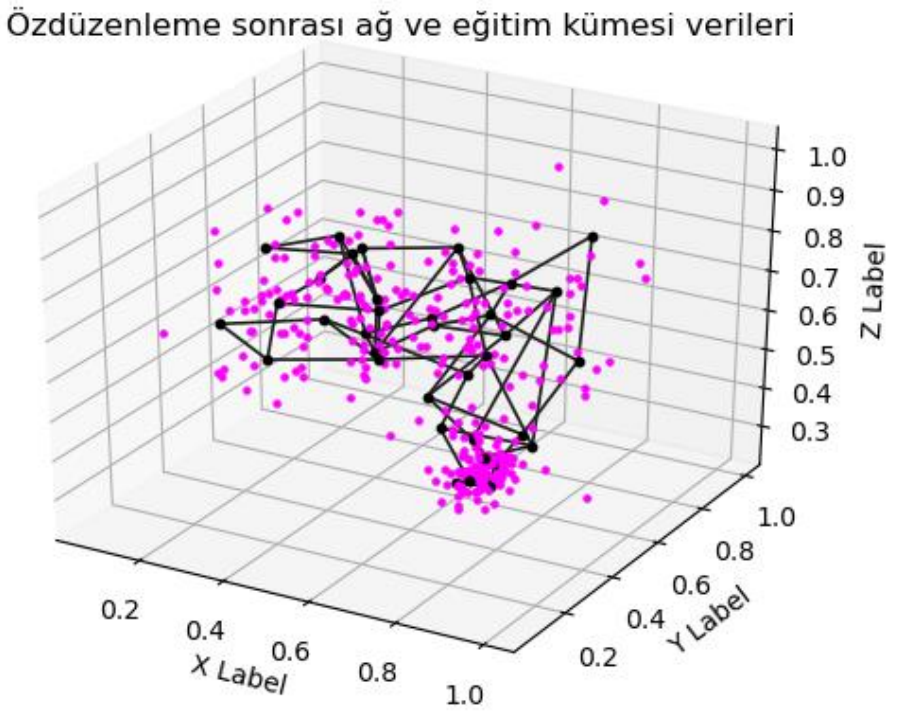


-
- Ağ boyutu = 5x5:

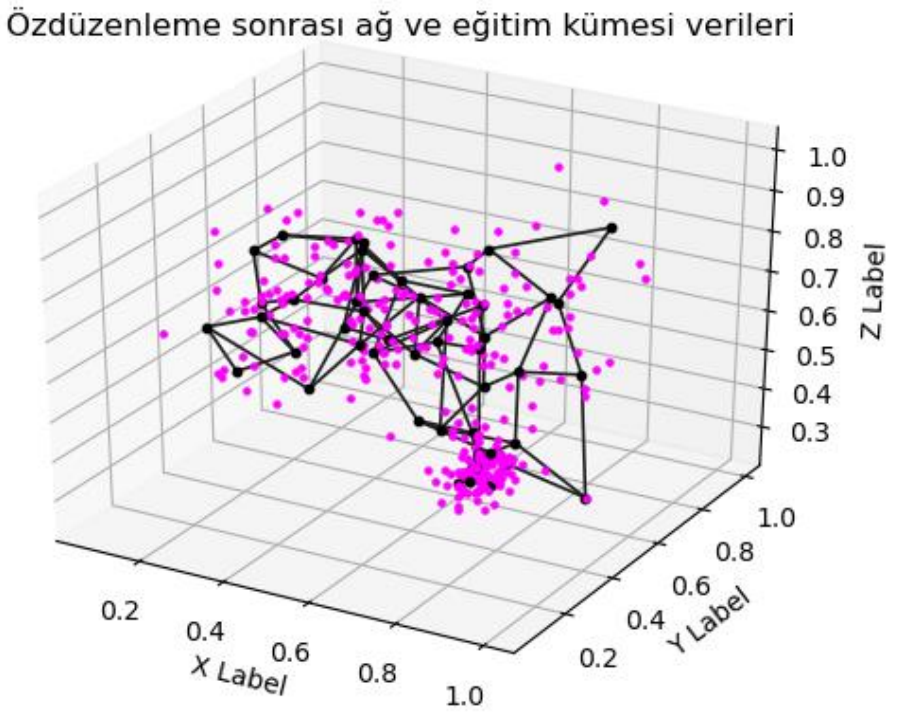


Soru 1: Üç Boyutlu Noktalar Kümesi

- Ağ boyutu = 6x6:

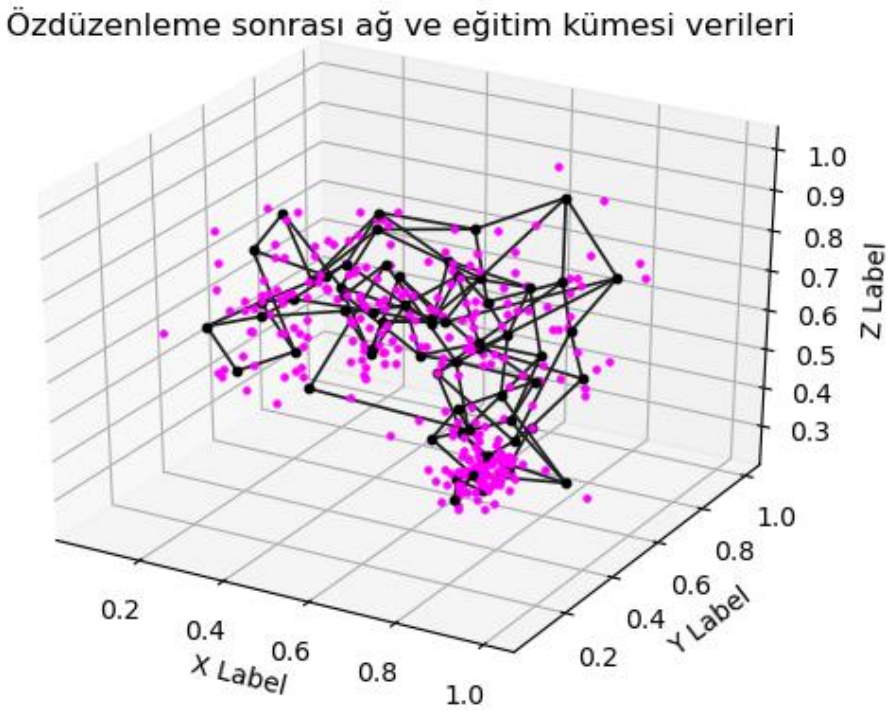


- Ağ boyutu = 7x7:



Soru 1: Üç Boyutlu Noktalar Kümesi

- Ağ boyutu = 8x8:



Ağ boyutunun büyümesi öbeklerin daha detaylı bir şekilde temsil edilmesini sağlarken, 4'e 4'lük ağ bile bu konuda başarılı olmaktadır.

Eğitim süreleri:

Eğitim süresi için epoch = özdüzenleme iterasyon sayısı ise

özdüzenleme aşamasının epoch ile,

yakınsama aşamasının (nöron sayısı)*epoch/2 ile,

toplam eğitim süresinin ise iki aşamanın toplamı ile epoch*(1 + (nöron sayısı)/2) ile doğru orantılı olması gerekir. Ölçülen eğitim süreleri tablodaki gibidir.

Soru 1: Üç Boyutlu Noktalar Kümesi

Epoch	50	250	500	750	1000
Özdüzenleme aşaması [sn]	7,0825	35,0963	70,9410	134,2541	160,4658
Yakınsama aşaması [sn]	127,7791	641,3150	1333,4954	2347,0576	2844,2079
Eğitim süresi [sn]	134,8616	676,4113	1404,4364	2481,3118	3004,6737
Başlangıç öğrenme hızı	0,001	0,010	0,100	0,300	0,500
Özdüzenleme aşaması [sn]	12,4954	14,1313	13,7148	13,8666	15,2210
Yakınsama aşaması [sn]	235,8309	261,3103	270,7707	236,6021	244,8789
Eğitim süresi [sn]	248,3263	275,4417	284,4855	250,4686	260,0999
Ağ boyutu	4x4	5x5	6x6	7x7	8x8
Nöron sayısı	16	25	36	49	64
Özdüzenleme aşaması [sn]	7,7551	10,6876	13,2846	19,6483	22,3913
Yakınsama aşaması [sn]	58,7694	182,4266	257,8314	474,3659	678,3452
Eğitim süresi [sn]	66,5245	193,1143	271,1160	494,0141	700,7365

Tabloda, beklenen doğrultuda bir eğitim süresi gerçekleştiği gözlenlemlenebilir. İterasyon sayısı eğitim süresini her 250 iterasyonda yaklaşık 800 saniye artırmış, başlangıç öğrenme hızı eğitim süresini etkilememiş ve ağdaki nöron sayısı, nöron sayısıyla hemen hemen doğru orantılı bir şekilde eğitim süresini artırmıştır.

Ayrıca eğitim sırasında birkaç for döngüsü iç içe kullanıldığı ve eğitim her nöron sınıfının içinde yapıldığı için oldukça uzun sürebildiği görülmüştür. Eğitim süresinin 50 dakikaya kadar çıktığı gözlenir.

Soru 2: İris Çiçeklerinin Kümelendirilmesi

Kurulan Kohonen Ağı ile bir önceki ödevdeki İris verilerinin ayrıştırılması için bir yapı oluşturulur. Çok katmanlı algılayıcı ile iyi ihtimalle %3 civarında hata oranı sağlarken bu ağ yapısı ile nasıl bir performans aldığı gözlemlenecektir.

Verilerin oluşumu:

Veriler ilk olarak iris.data dosyasından çekilip setosa, versicolor ve virginica çiçekleri için ayrı oluşturulan listelere yerleştirilir.

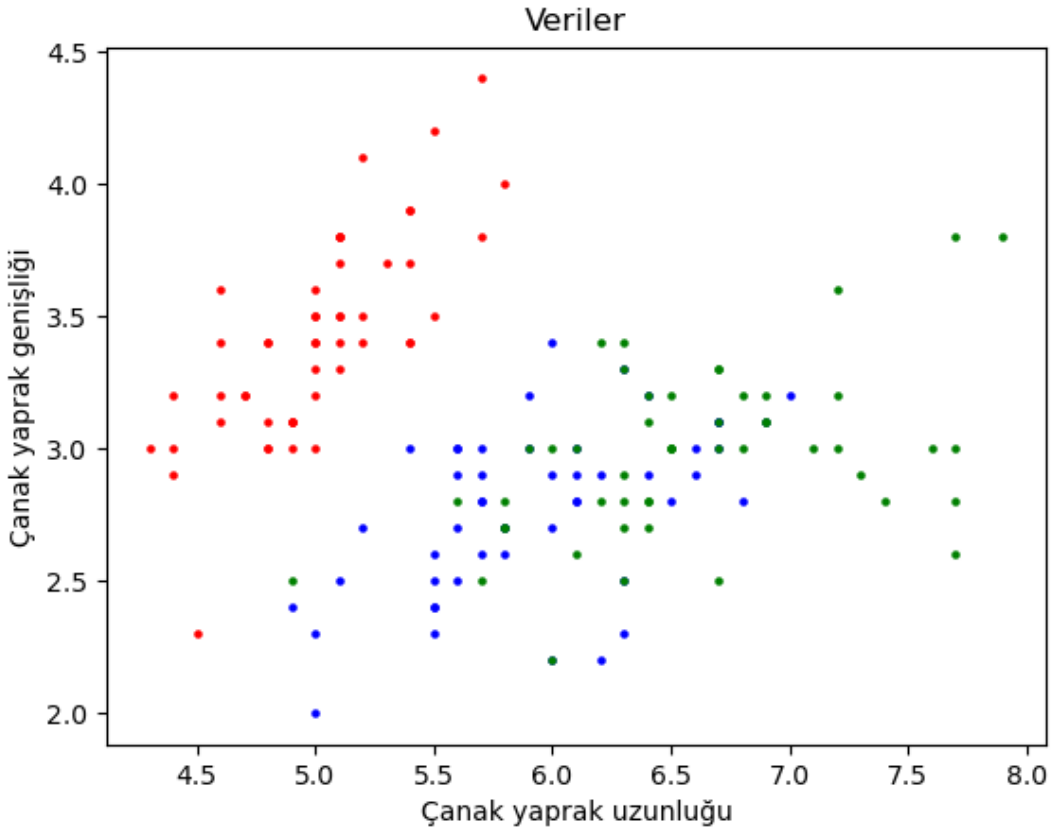
```
__location__ = os.path.realpath(os.path.join(os.getcwd(), os.path.dirname(__file__)))
dataFile = os.path.join(__location__, 'p2data/iris.data')
dim = 4

irisfile = open(dataFile, 'r')
Lines = irisfile.readlines()

setosaL = []
versicolorL = []
virginicaL = []
for line in Lines:
    rawdata = line.split(',')
    if len(rawdata) < 5:
        break

    data = np.zeros(dim, dtype=float)
    data[0] = rawdata[0]
    data[1] = rawdata[1]
    data[2] = rawdata[2]
    data[3] = rawdata[3]

    # Çiçeğin sınıfına göre farklı listelere dağıtılır.
    if rawdata[4].rstrip("\n") == "Iris-setosa":
        setosaL.append(data)
    elif rawdata[4].rstrip("\n") == "Iris-versicolor":
        versicolorL.append(data)
    elif rawdata[4].rstrip("\n") == "Iris-virginica":
        virginicaL.append(data)
    else:
        continue
```

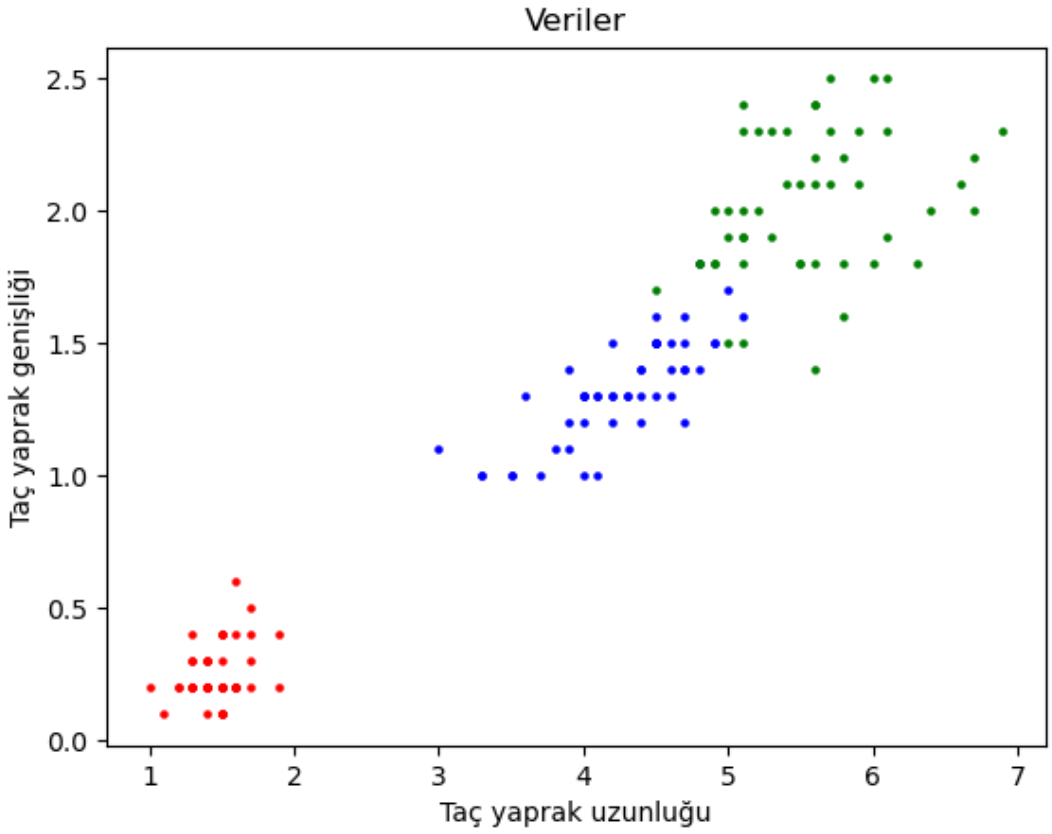
Soru 2: İris Çiçeklerinin Kümelendirilmesi

Daha sonra her çiçek kümesinin yarısı test, yarısı eğitim kümesine atanır. Eğitim kümesindeki verilerde hala sınıf bilgisi bulunmuyorken, bu sefer test kümesine bu bilgi verilir. Çiçek sınıfına göre verinin sonuna setosa için [0], versicolor için [1] ve virginica için [2] eklenir. Bu değerler test aşamasında indis olarak kullanılacaktır.

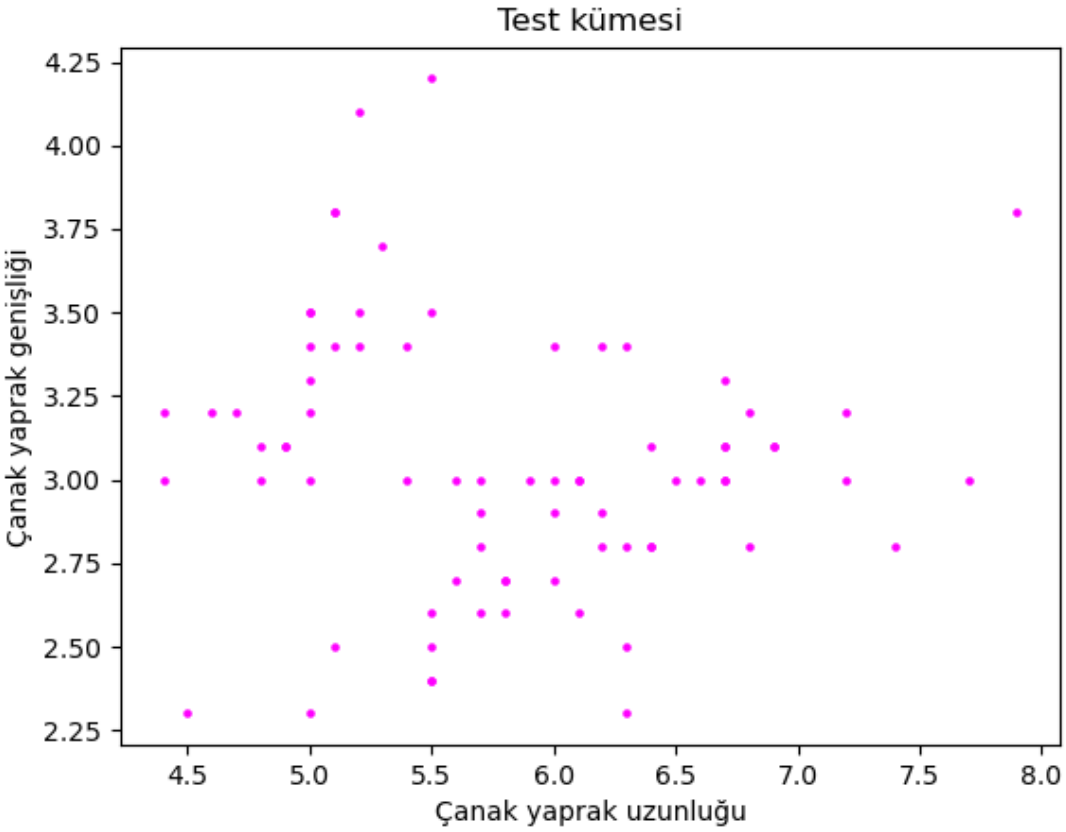
```
# Eğitim ve test sırasında kullanılacak veri kümelerinin dağılımı.
trainingFile = os.path.join(__location__, 'p2data/Training.pkl')
testingFile = os.path.join(__location__, 'p2data/Testing.pkl')
if os.path.exists(trainingFile):
    trainingList = loadList(trainingFile)
    testingList = loadList(testingFile)
else:
    trainingList = []
    testingList = []

    for i in range(0, len(setosaL)):
        if i < len(setosaL)/2:
            trainingList.append(setosaL[i])
        else:
            testdata = np.concatenate([setosaL[i], [0]])
            testingList.append(testdata)
    for i in range(0, len(versicolorL)):
        if i < len(versicolorL)/2:
            trainingList.append(versicolorL[i])
        else:
            testdata = np.concatenate([versicolorL[i], [1]])
            testingList.append(testdata)
    for i in range(0, len(virginicaL)):
        if i < len(virginicaL)/2:
            trainingList.append(virginicaL[i])
        else:
            testdata = np.concatenate([virginicaL[i], [2]])
            testingList.append(testdata)

    saveData(trainingList, trainingFile)
    saveData(testingList, testingFile)
```

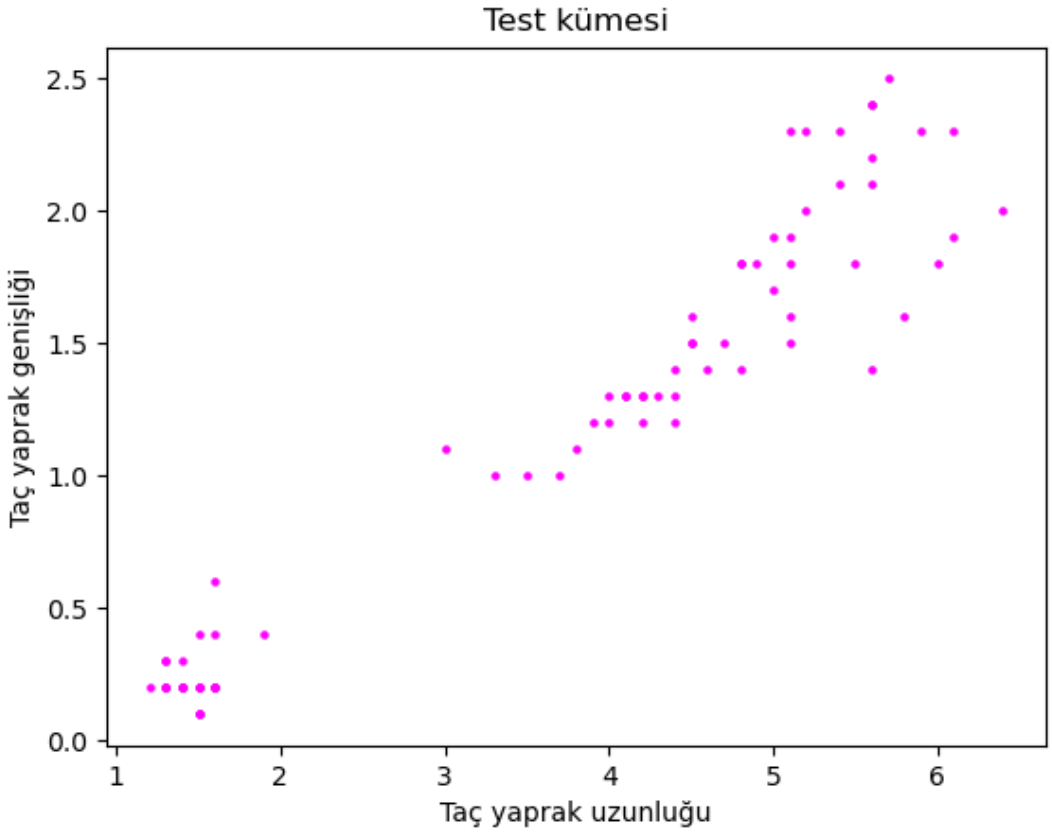
Soru 2: İris Çiçeklerinin Kümelendirilmesi



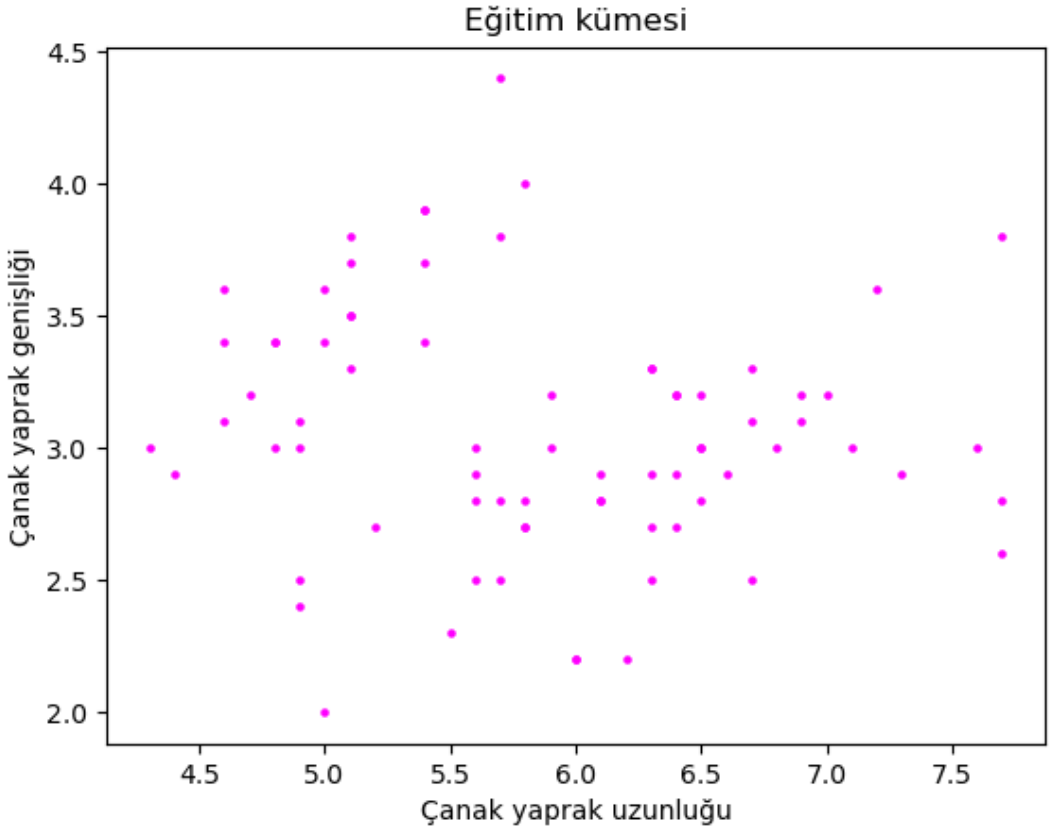


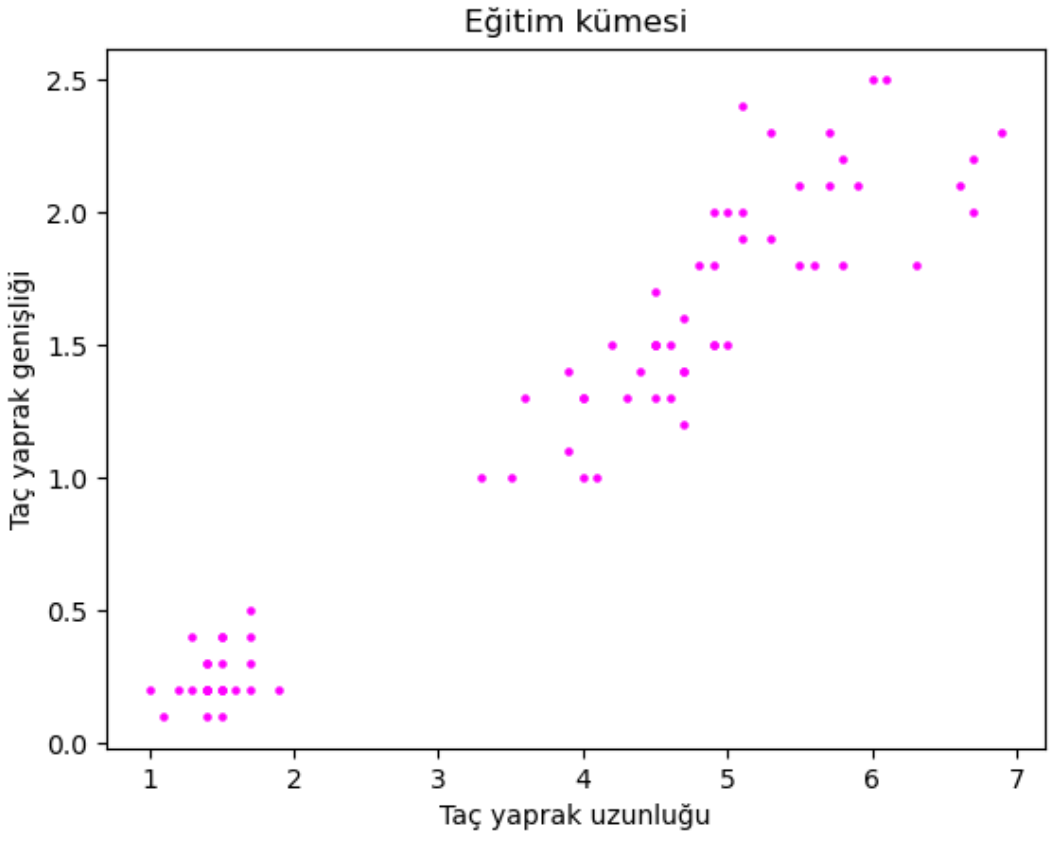
Soru 2: İris Çiçeklerinin Kümelendirilmesi





Soru 2: İris Çiçeklerinin Kümelendirilmesi





Soru 2: İris Çiçeklerinin Kümelendirilmesi

Verilerin çizimi:

Verilerin çizimi için önceki problemde kullanılan çizim fonksiyonları iki boyutlu çizim için değiştirilmiştir. Veriler 4 boyutlu olup çanak ve taç yaprakların genişlik ve uzunluklarını bulundurmaktadır. Çanak yaprak verileri ve taç yaprak verileri ayrı grafiklerle ifade edilir.

```python
def plotnetwork(network, xdim, ydim, xind, yind, xlabel, ylabel, title):
    # Önceki problemdeki ağ çizdirme fonksiyonuyla hemen hemen aynı şekilde çalışır. Önce ağdaki nöronları,
    # daha sonra nöronlar arasındaki bağları çizdirir.
    fig = plt.figure()
    ax = fig.add_subplot(111)
    ax.set_title(title)
    for Neuron in network:
        weights = Neuron.weights[0]
        ax.scatter(weights[xind], weights[yind], s=10, c="black")
    for Neuron in network:
        weights = Neuron.weights[0]
        nextNeuron = Neuron.nextNeuron
        next_weights1 = nextNeuron.weights[0]

        xcord1 = np.linspace(weights[xind], next_weights1[xind], 5)
        ycord1 = np.linspace(weights[yind], next_weights1[yind], 5)

        nextNeuron = Neuron
        for i in range(ydim):
            nextNeuron = nextNeuron.nextNeuron
        next_weights2 = nextNeuron.weights[0]

        xcord2 = np.linspace(weights[xind], next_weights2[xind], 5)
        ycord2 = np.linspace(weights[yind], next_weights2[yind], 5)
        if Neuron.y != ydim - 1:
            ax.plot(xcord1, ycord1, linewidth=1.2, c='0.12')
        if Neuron.x != xdim - 1:
            ax.plot(xcord2, ycord2, linewidth=1.2, c='0.12')

    ax.set_xlabel(xlabel)
    ax.set_ylabel(ylabel)
```

```python
def plot2ddata(title, xind, yind, xlabel, ylabel, dList1, dList2 = [], dList3 = [], single_list = True):
    # Verilerin girilen indislerini iki boyutlu düzlemde çizdirir. Eğer üç farklı veri listesi girilmişse
    # bu verileri kırmızı, mavi ve yeşil renkte çizdirir. Tek liste girilmişse tüm verileri mor/pembe renkte çizdirir.
    fig = plt.figure()
    ax = fig.add_subplot(111)
    ax.set_title(title)
    if single_list == False:
        for data in dList1:
            ax.scatter(data[xind], data[yind], s=5, c="red")
        for data in dList2:
            ax.scatter(data[xind], data[yind], s=5, c="blue")
        for data in dList3:
            ax.scatter(data[xind], data[yind], s=5, c="green")
    else:
        for data in dList1:
            ax.scatter(data[xind], data[yind], s=5, c="magenta")

    ax.set_xlabel(xlabel)
    ax.set_ylabel(ylabel)
```

Soru 2: İris Çiçeklerinin Kümelendirilmesi

Eğitim hazırlıkları:

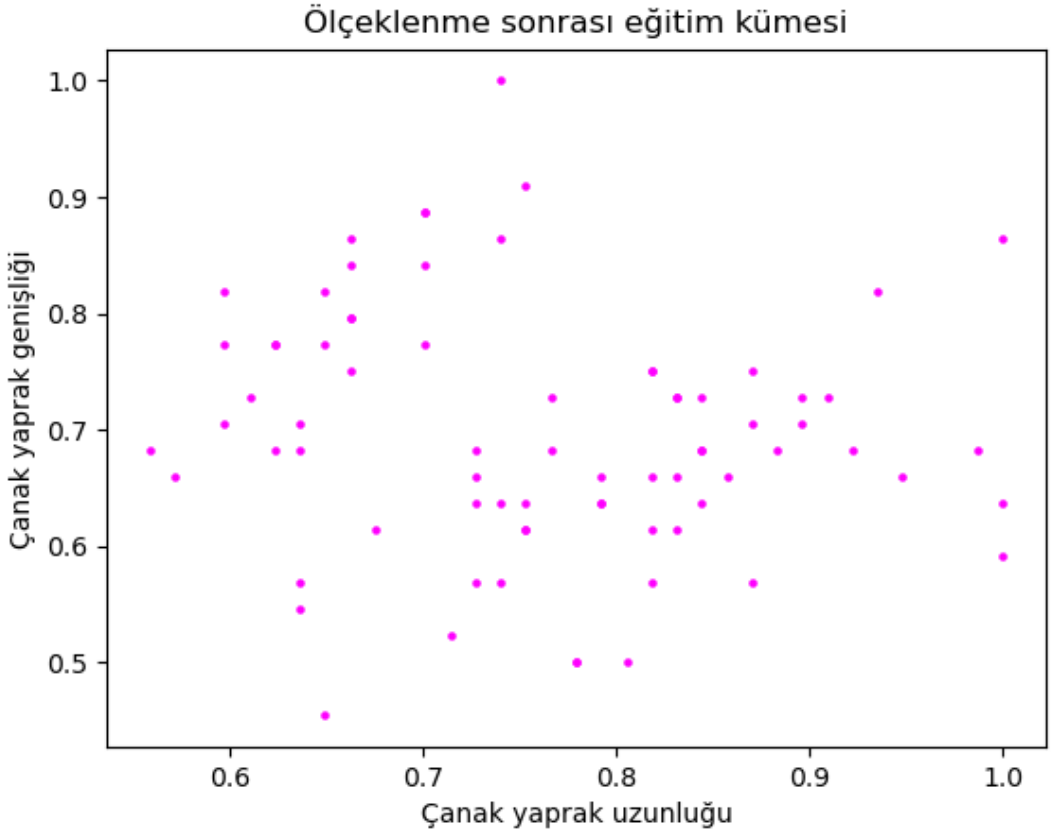
Önceki problemle birebir aynıdır. Epoch = 500, ağ boyutları 6x6 ve başlangıç öğrenme hızı 0.1 seçilir. Zaman sabitleri hesaplanır ve uzaklık sözlüğü oluşturulur.

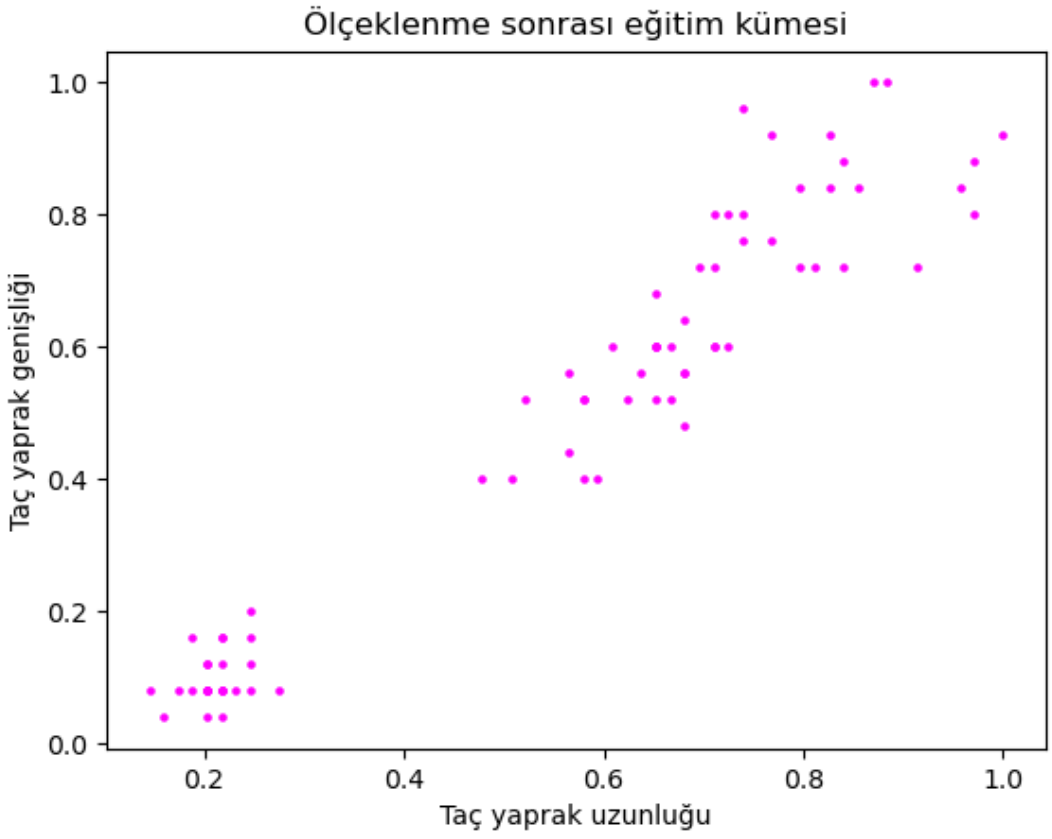
```
# Eğitimin süreceği toplam iterasyon sayısı Haykin'in "Neural Networks - A Comprehensive Foundation" kitabındaki tavsiyesi
# özdüzenlemenin 1000, yakınsama aşamasının toplam nöron sayısının 500 katı olduğu yönündedir. Bu problem için 500 özdüzenleme iterasyonu
# seçilecek, yakınsama iterasyonu da toplam nöron sayısının, seçilen iterasyonun yarısıyla çarpımı olarak seçilecek.
# Ardından, komşuluk etkisinin ve adaptif öğrenme hızının zamanla eksponansiyel olarak azalması için zaman sabitleri belirlenir.
# Bu değerler aynı kitabın 474-475 sayfalarında Haykin'in söylediği şekilde alınır.
epoch = 500
conv_epoch = epoch/2

# 6'ya 6 boyutlu toplam 36 nörondan oluşan, başlangıçtaki öğrenme hızı 0.1 olan bir ağ oluşturulacak.
xdim = 6
ydim = 6
learning_rate = 0.1

tcons1 = epoch/math.log(math.sqrt(xdim**2 + ydim**2))
tcons2 = epoch

# Eğitimde kullanmak üzere, uzaklıkların bulunduğu bir kütüphane oluşturulur.
distDict = {}
for i in range(xdim):
    for j in range(ydim):
        dist = math.sqrt(i**2 + j**2)
        # 0.1 standart sapmaya sahip normal dağılımı. Farklı nöronların komşuluk derecesini
        # ve kazanan nörona göre ağırlıklarının ne kadar değişeceğini belirler.
        distkey = (i, j)
        distDict[distkey] = dist
```

Eğitim kümesi (0,1) arasında ölçeklenir.

```
normalized_training = normalize_List(trainingList, dim, np.zeros(dim), np.ones(dim))
trainingList_normal = normalized_training[0]
maxL_training = normalized_training[1]
minL_training = normalized_training[2]
```

Soru 2: İris Çiçeklerinin Kümelendirilmesi





Soru 2: İris Çiçeklerinin Kümelendirilmesi

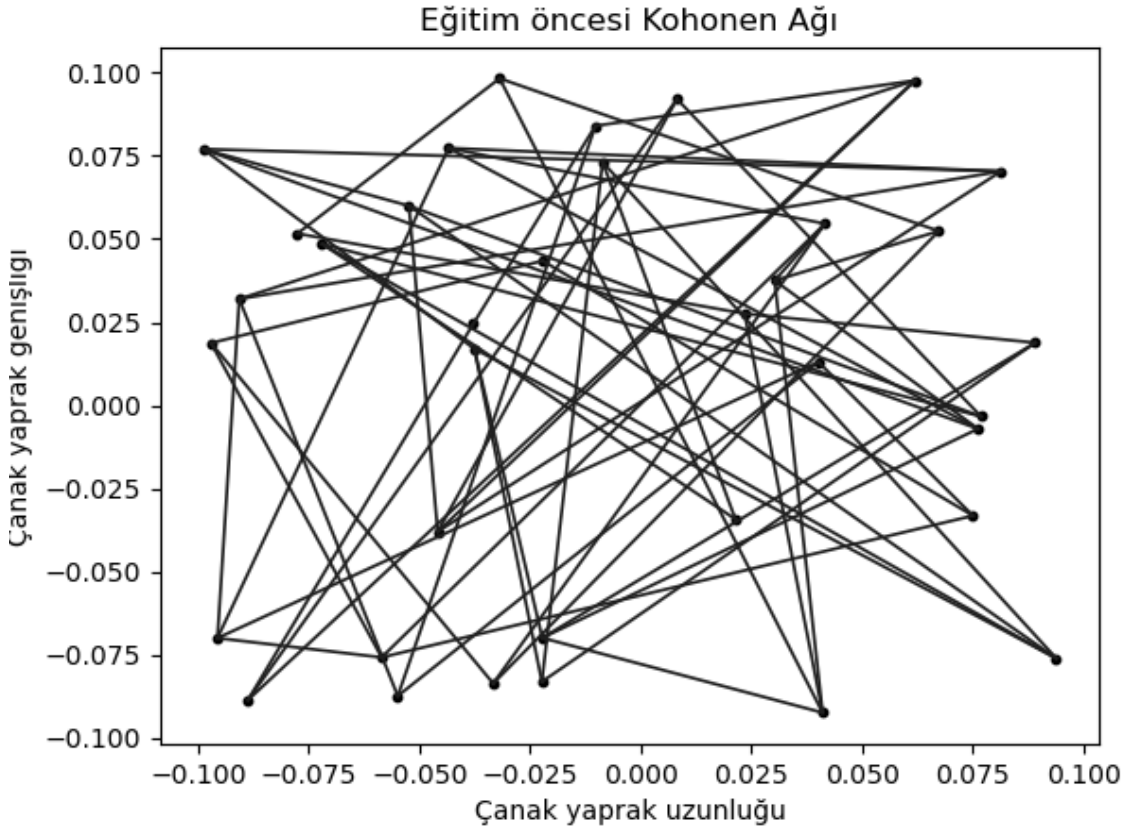
Eğitim için ağ oluşturulur ve eğitim başlatılır. Eğitim aşaması önceki problemle tamamen aynıdır, sadece veri boyutu ve ağın eğitildiği veriler farklıdır. Başlangıçtaki ağ yapısı beklendiği gibi düzensizdir.

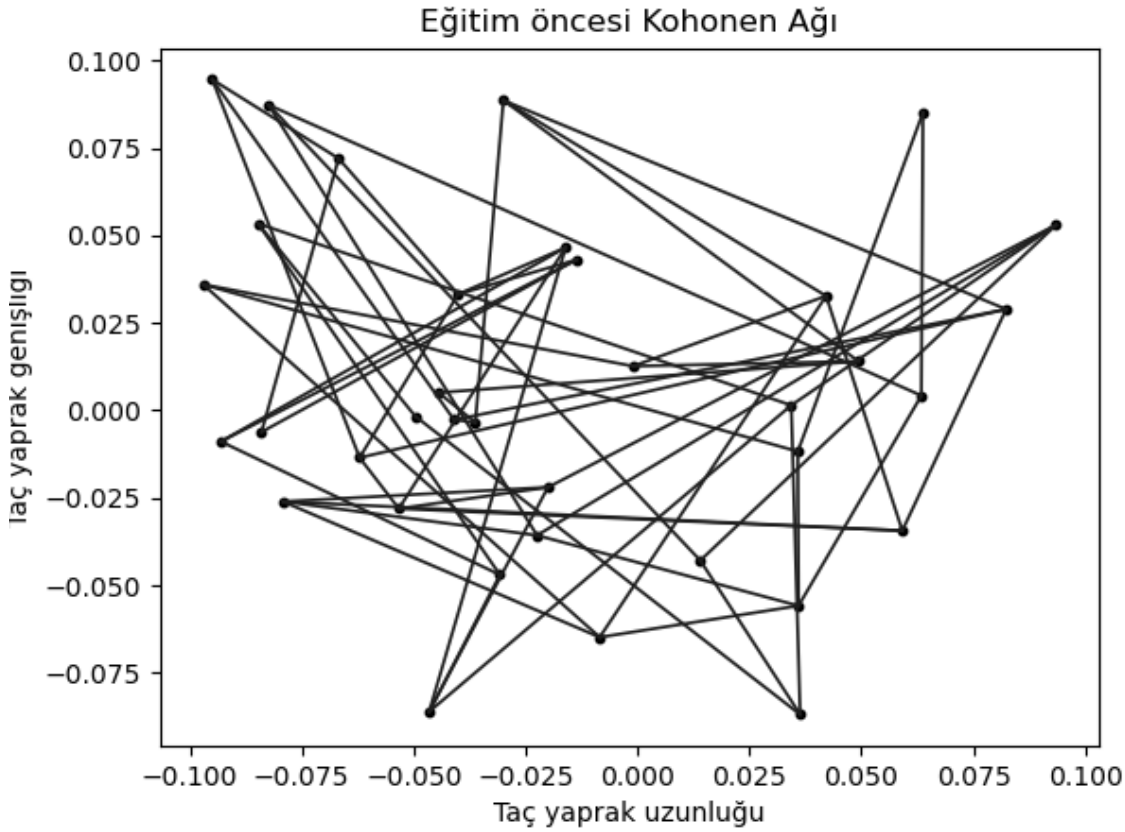
Özdüzenleme aşamasından sonra ağın veri öbeklerine göre düzenlendiği görülür. 4 boyutlu bir veri üzerinde eğitim yapıldığı için bu düzen hala Haykin'in örneğindeki gibi açık bir şekilde görülemez.

Yakınsama aşaması sonrası yine detaya yönelik değilimler olsa da bu problemde önceki problemden daha büyük değişimler görülür. Ağ, verileri temsil etmekte optimize olmaktadır.

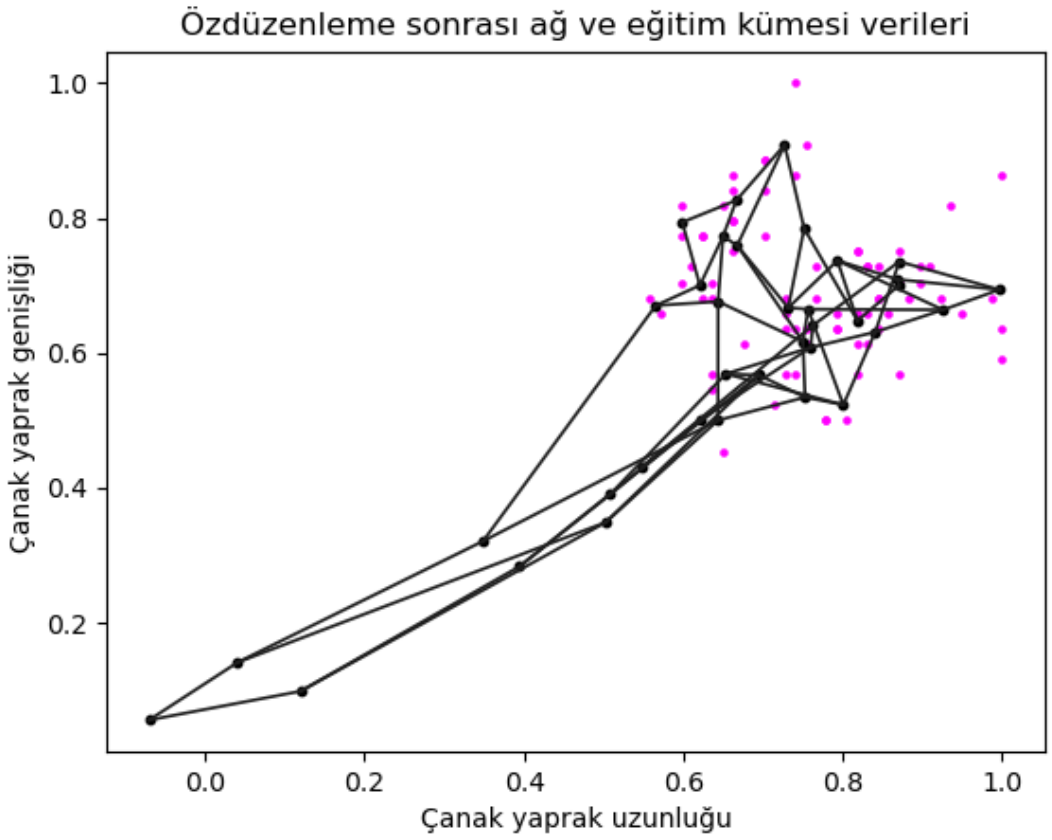
Eğitildikten sonra geri ölçeklenen ağ test kümesiyle karşılaştırıldığında veri öbeklerini başarılı bir şekilde temsil ettiği görülmektedir.

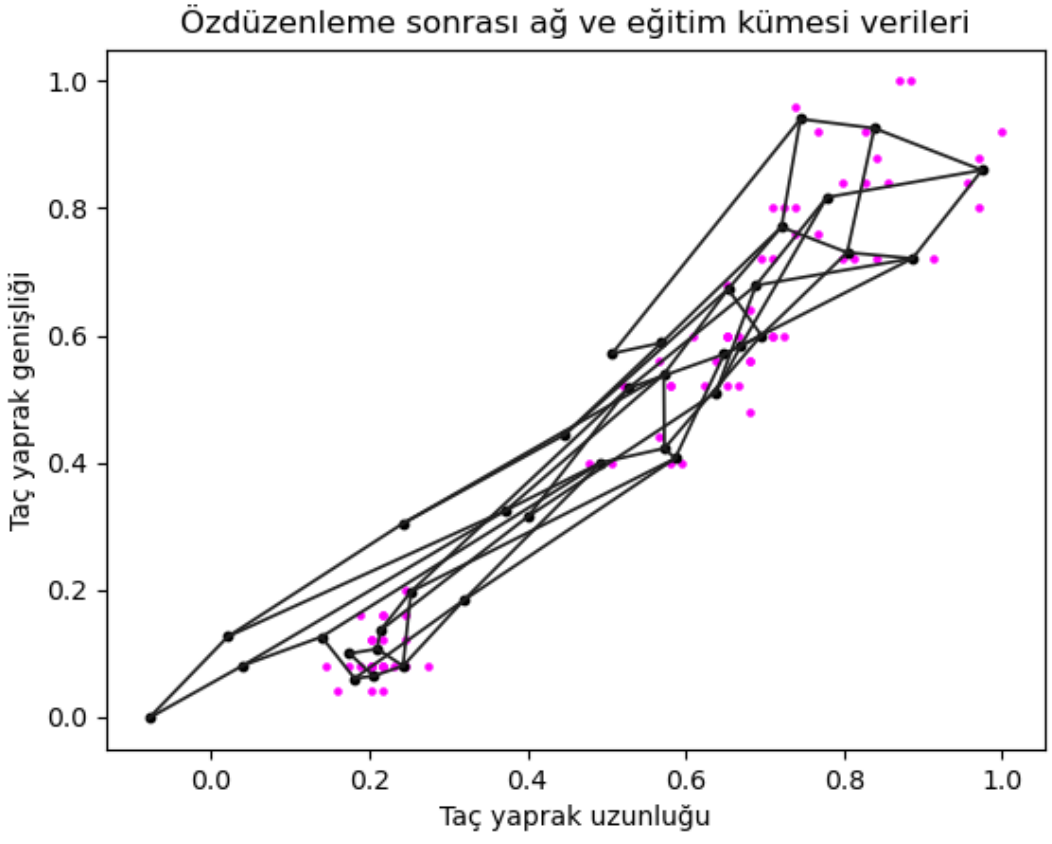
Soru 2: İris Çiçeklerinin Kümelendirilmesi



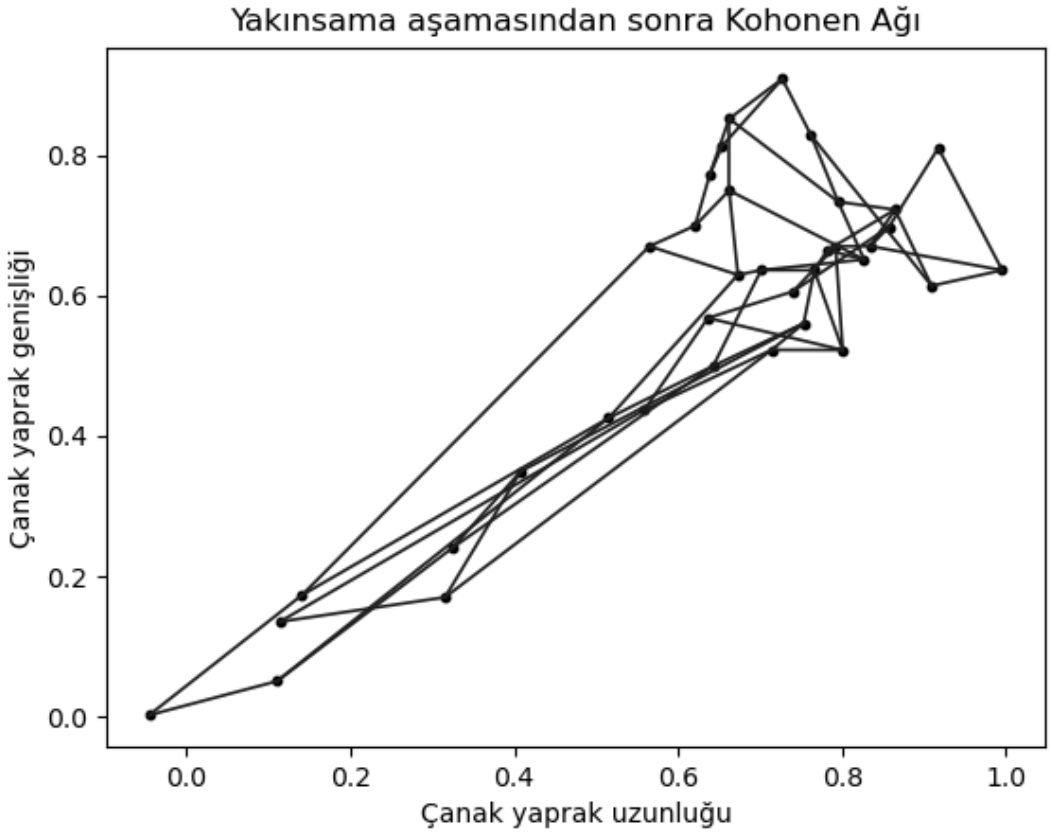


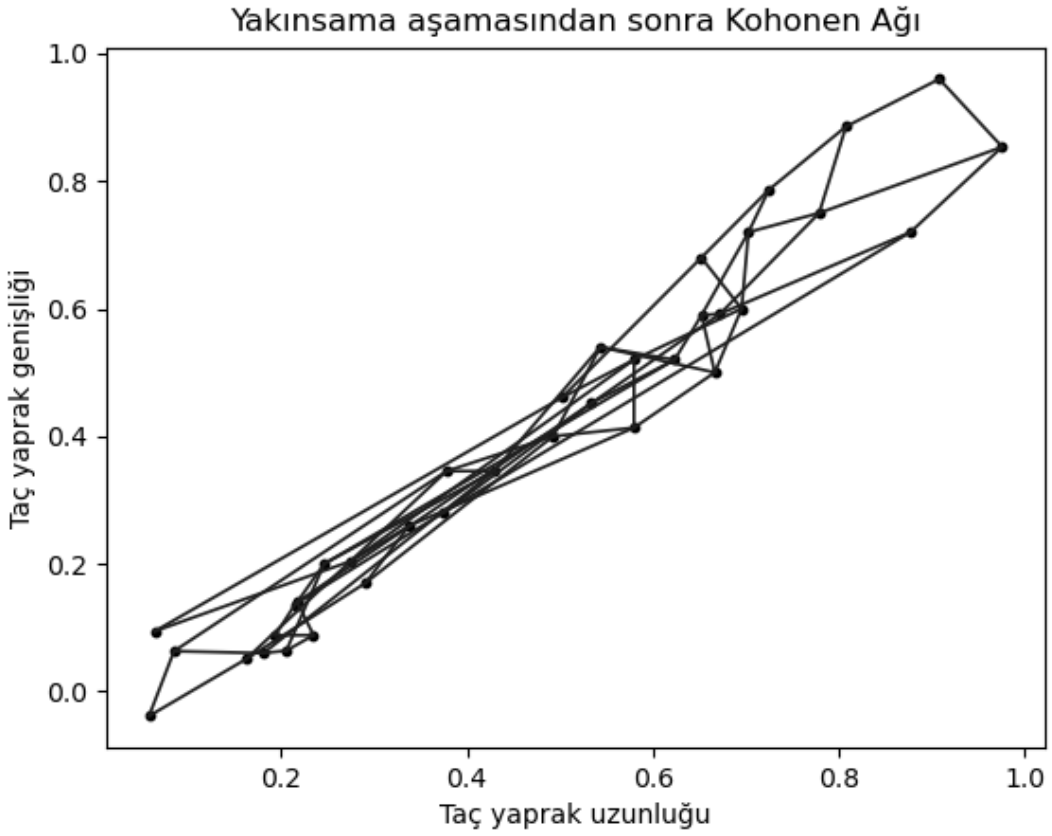
Soru 2: İris Çiçeklerinin Kümelendirilmesi



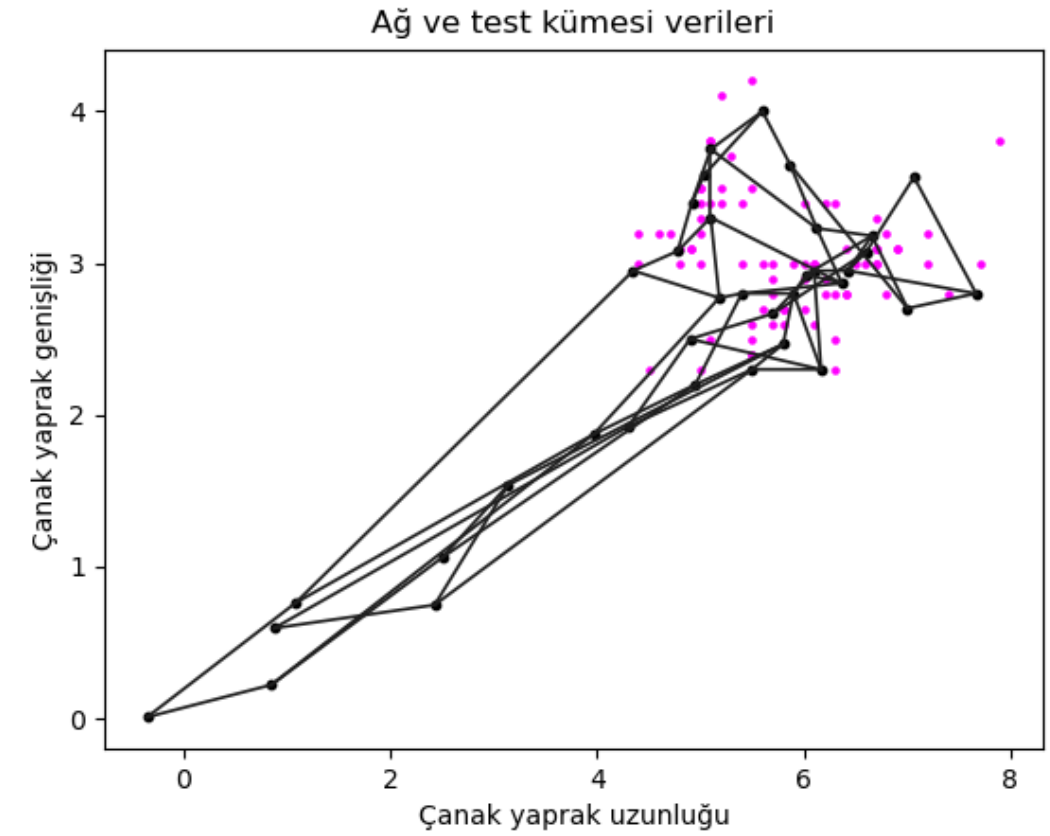


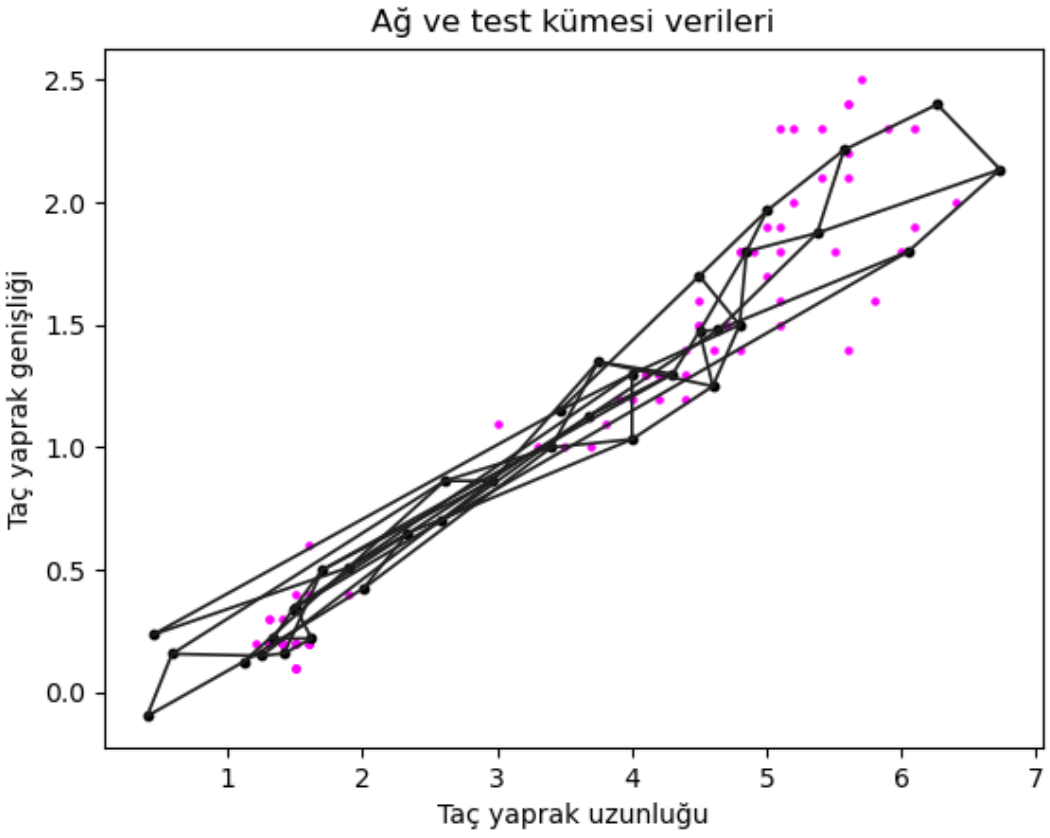
Soru 2: İris Çiçeklerinin Kümelendirilmesi





Soru 2: İris Çiçeklerinin Kümelendirilmesi





Soru 2: İris Çiçeklerinin Kümelendirilmesi

Test aşaması:

Eğitim fonksiyonuyla aynı şekilde her veri için kazanan nöron belirlenir. Bu nöron, (x,y) indislerinin anahtar ve setosa, versicolor ve virginica sınıflarının değer olduğu bir sözlükteki kazanılan veri sınıfının artırılmasında kullanılır.

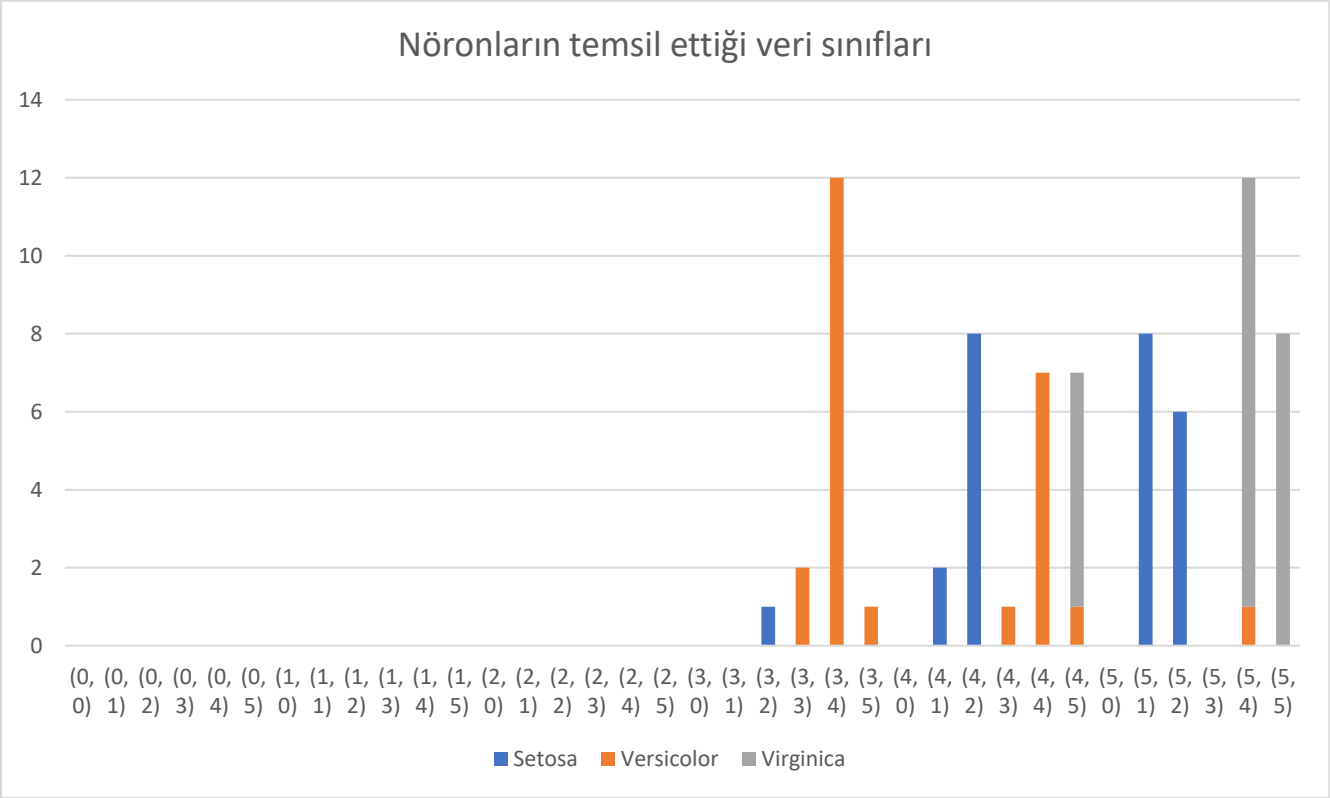
```
def testing_process(network, dList, dim):
    # Test aşamasında öncelikle nöronların x ve y kordinatları anahtar olarak kullanılıp bir sözlük oluşturulur.
    # Bu sözlükte her nöronun hangi tip veride kaç kere kazandığı bilgisi olacaktır.
    winnerDict = {}
    for Neuron in network:
        # Nöronun x ve y koordinatları anahtar olarak alınır.
        key = (Neuron.x, Neuron.y)
        # Nöronun hangi sınıftan veride kaç kere kazandığı ise değerler olacaktır.
        # Bu değerler sırayla setosa, versicolor ve virginica çiçeklerini ifade eder.
        winnerDict[key] = [0, 0, 0]

    for data in dList:
        for Neuron in network:
            winner = 999
            winner_x = 0
            winner_y = 0
            current = 0
            # Kazananın belirlenmesi için veri ile nöronların ağırlıkları arasındaki uzaklık ölçülür ve en düşük olan seçilir.
            for i in range(len(Neuron.weights[0])):
                current += (Neuron.weights[0][i] - data[i])**2
            # Kazananın indisleri kaydedilir.
            if current < winner:
                winner = current
                winner_x = Neuron.x
                winner_y = Neuron.y
        # Kazanan nöronun kazandığı sınıftaki değeri artar. Test kümesi oluşturulurken farklı çiçekler için
        # tüm verilere farklı değerler eklenmiştir. Verinin sonuna setosa için 0, versicolor için 1 ve virginica
        # için 2 eklenmiştir. Bu değerler, kazanan nöronun hangi sınıftaki değerini artıracağını seçerken kullanılır.
        winner_key = (winner_x, winner_y)
        winner_val = winnerDict[winner_key]
        winner_cls = int(data[dim])
        winner_val[winner_cls] = winner_val[winner_cls] + 1
        winnerDict[winner_key] = winner_val
    return winnerDict
```

Elde edilen kazananlar sözlüğü, “p2results” klasöründe “irispred.xls” excel dosyasına yazdırılır.

```
# Nöronların kazandığı veri sınıflarını excel dosyasına kaydeden fonksiyon.
def saveResults(result, xdim, ydim, filedir):
    if os.path.exists(filedir):
        wb = open_workbook(filedir)
    else:
        wb = Workbook()
        sheet1 = wb.add_sheet('Sheet1')
        sheet1.write(0, 0, 'Nöron (x,y)')
        sheet1.write(0, 1, 'Setosa')
        sheet1.write(0, 2, 'Versicolor')
        sheet1.write(0, 3, 'Virginica')
        for i in range(xdim):
            for j in range(ydim):
                row = i*ydim + j + 1
                sheet1.write(row, 0, str((i,j)))
                for k in range(3):
                    sheet1.write(row, (k+1), result[i,j][k])
    wb.save(filedir)
```

Soru 2: İris Çiçeklerinin Kümelendirilmesi

Nöronların temsil ettiği veri sınıfların karşılaştırıldığında, iki nöron haricinde birden fazla sınıfı temsil eden nöron olmadığı görülür. İki nöronda da azınlıkta olan sınıftan sadece tek bir veri temsil edilmektedir. Ek olarak çoğu nöronun herhangi bir veride kazanmadığı görülür.



Test kümesi 75 veriden oluşmaktadır. Bu durumda 2 verinin hatalı sınıflandırıldığı düşünülürse hata oranı %2.67'lik bir hata oranı gözlenir. Bu oran daha büyük bir ağ veya daha yüksek iterasyonlu bir eğitim ile düşürülebilir. Çok katmanlı algılayıcının %3'lük hata oranı ile karşılaştırıldığında daha isabetli bir sınıflandırma yaptığı görülür.