YAPAY SİNİR AĞLARI

ÖDEV-2 RAPORU

Prof. Dr.Neslihan Serap Şengör

Osman Kaan Kurtça

040160090

Hasan Göktuğ Kayan

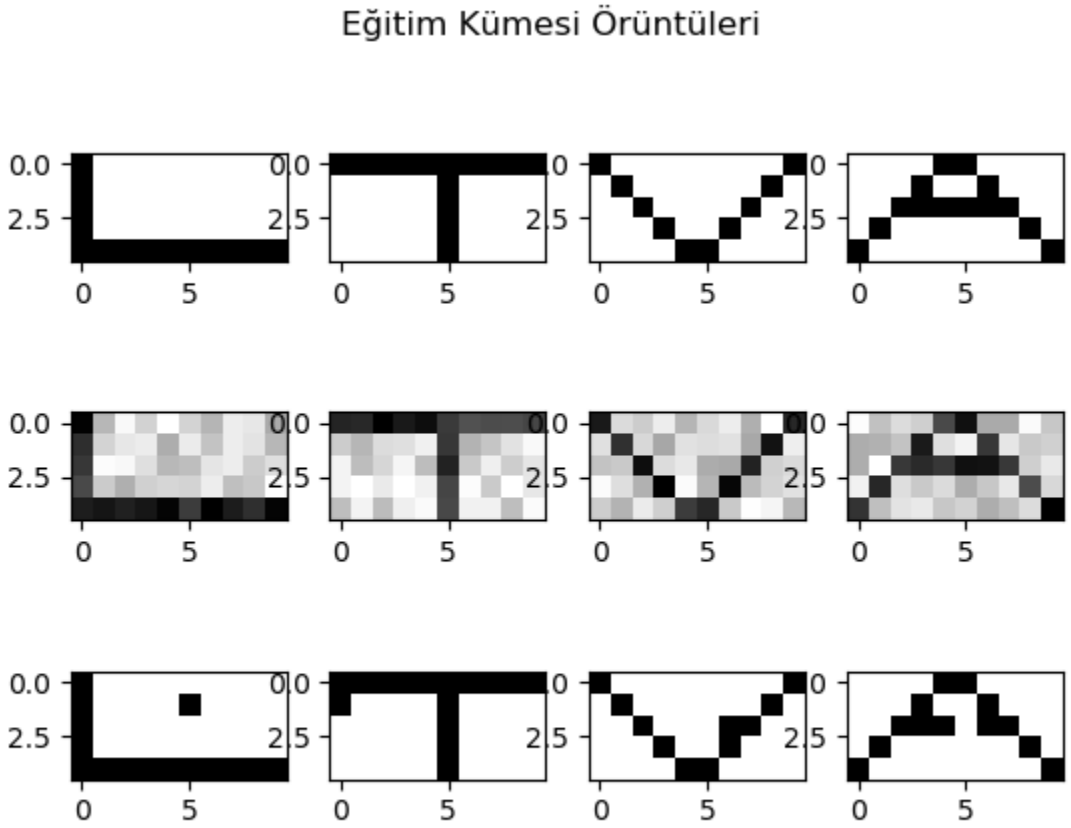
040160072

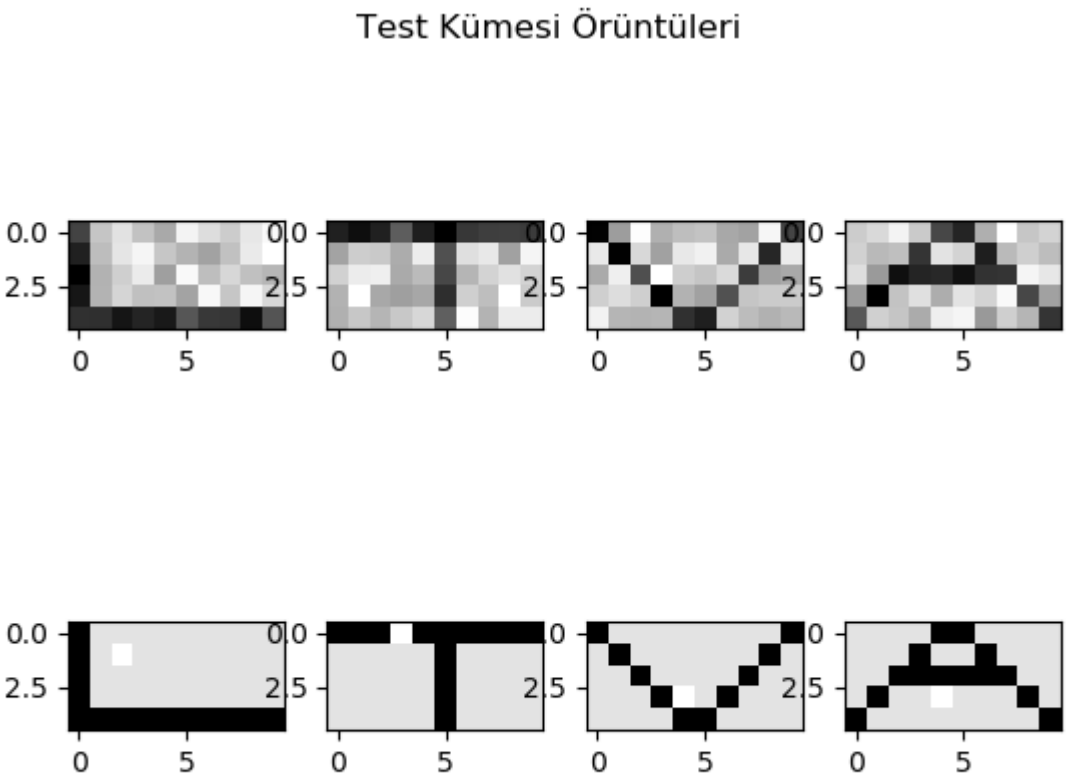
SORU-1)

Çok katmanlı algılayıcı tek katmanlı algılayıcıdan farklı olarak giriş ve çıkış katmanları arasında birden farklı sayıda her biri farklı sayıda nörondan oluşan gizli katmanlar bulunduran yapay sinir ağı çeşididir. Bu ağda yapılanları ileri yol ve geriye yayılım olarak iki aşamada açıklayabiliriz. İleri yol ağında her katmanın çıkışındaki değerler o katmana ilişkin ağırlıklarla çarpılarak bir değerler kümesi elde edilir bu kümenin tüm elemanları probleme göre belirlenen aktivasyon fonksiyonundan geçirilerek bir sonraki katmana giriş olarak sunulur. Aktivasyon fonksiyonu son katmanda çıkış değerlerine göre verinin sınıflandırılmasını sağlar. Her katman için farklı aktivasyon fonksiyonları tanımlanabileceği gibi bizim de bu problem çözerken kullandığımız gibi tek bir aktivasyon fonksiyonu da seçilebilir. Geriye yayılım algoritmasıyla yapılan , bu problem bir eğiticili öğrenme örneği olduğu için test kümesindeki her örüntünün tahmin edilen çıktıları ile eğitim için ağa verilen örüntülerin etiketleri arasındaki farkı hata olarak hesaplayıp bu hatayı her katmana ilişkin yerel gradyen terimleri, öğrenme hızı ve giriş değerlerini dikkate alarak ağırlıkların güncellenmesi için kullanılmasıdır.

a) Eğitim ve test kümelerinin seçilmesi

L,T,V ve A harflerinden oluşan test ve eğitim kümelerimiz aşağıdaki gibi şeçildi.





Eğitim kümesini harflerin orijinal halleri, gri gürültülü halleri ve orijinal hallerinden 1 piksel farklı halleri olmak üzere 12 örüntüden oluşturduk. Test kümesini de aynı şekilde harflerin gri gürültü eklenmiş halleri ile 1 piksel değişmiş hallerinden oluşan 8 örüntü şeklinde ağa giriş olarak sunduk.

b) Ağın parametrelerinin belirlenmesi

Modelimizde gizli katman sayımızı 2 olarak belirledik. Bu problem için ilk gizli katman 20 ikinci gizli katman 10 nörondan oluşuyor. Örüntüleri 0 ve 1 olarak sınıflandırdığımız için aktivasyon fonksiyonumuzu sigmoid seçtik. Durdurma koşulumuz ise 0.001 eğitim ortalama kare hatası olarak belirledik.

c) Katmanlardaki nöron sayılarının eğitim sürecine olan etkisini görmek için ağı oluşturmak için kullandığımız sınıfa farklı nöron sayıları vererek denedik.

İlk gizli katman nöron sayısı 20 ikinci gizli katman nöron sayısını 10 olarak seçtik bunun sonucunda

```
Eğitim için Ortalama Kare Hata: 0.0023010332415234 57, iterasyon sayısı: 100
```

```
Test ortalama kare hatası:  0.02019575
8 veriden 8 tanesi doğru sınıflandırıldı.
```

İlk gizli katman nöron sayısı 10 ikinci gizli katman nöron sayısını 5 olarak seçtik bunun sonucunda

```
Eğitim için Ortalama Kare Hata: 0.028986468092471813, iterasyon sayısı: 100
```

```
Test ortalama kare hatası:  0.21968262500000002
8 veriden 6 tanesi doğru sınıflandırıldı.
```

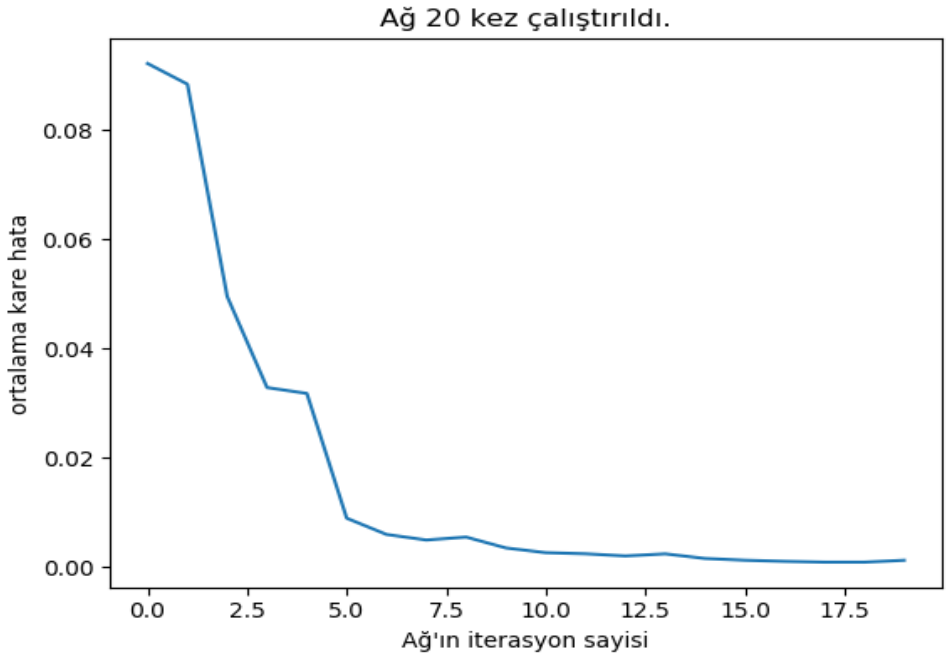
İlk gizli katman nöron sayısı 5 ikinci gizli katman nöron sayısını 3 olarak seçtik bunun sonucunda

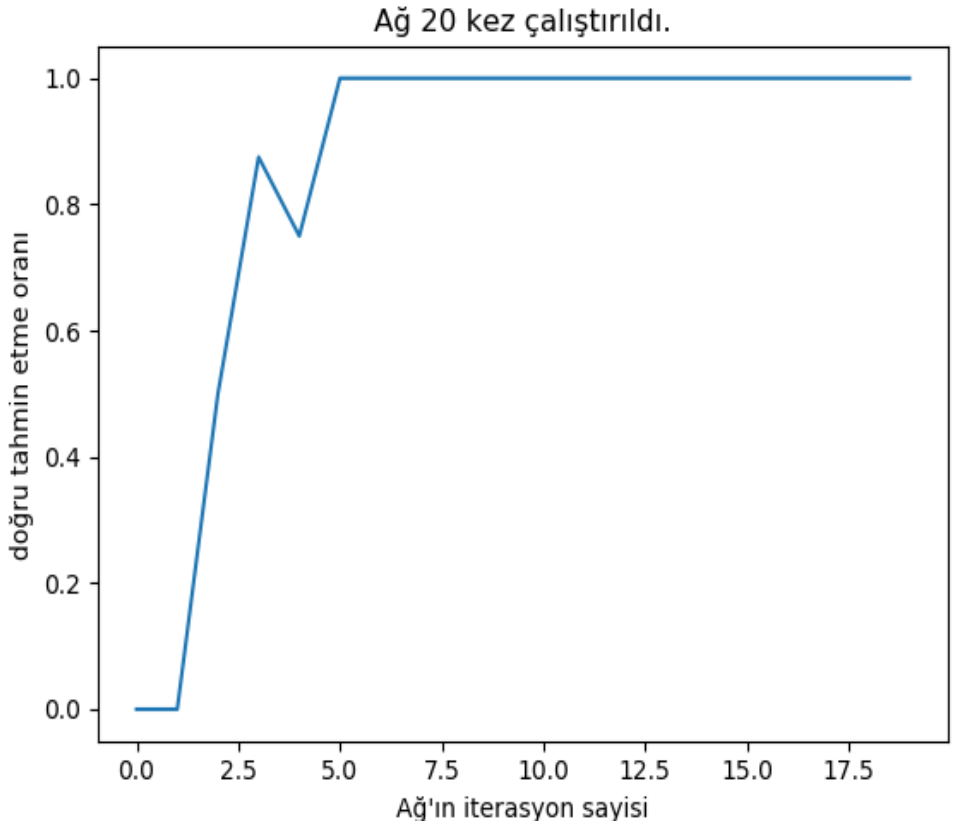
```
Eğitim için Ortalama Kare Hata: 0.07097091234572515, iterasyon sayısı: 100
```

```
Test ortalama kare hatası:  0.5567571250000001
8 veriden 2 tanesi doğru sınıflandırıldı.
```

Değerlere bakarsak; gizli katman nöron sayılarını 20 ve 10'un daha altında seçersek eğitim ve test hatamızın arttığını ve örüntüleri doğru sınıflandıramadığımızı söyleyebiliriz.

Öğrenme hızının eğitim sürecine etkisini görmek için ağı 20 kere çalıştırdık ve her seferinde ağa 0.1 ile 0.9 arasında düzenli artacak şekilde olmak üzere 20 farklı öğrenme hızı verdik. Öğrenme hızını arttırmak bu problemde hatayı daha hızlı düşürmemizi sağlıyor fakat optimum noktadan uzaklaşmamıza sebep olmuyor.

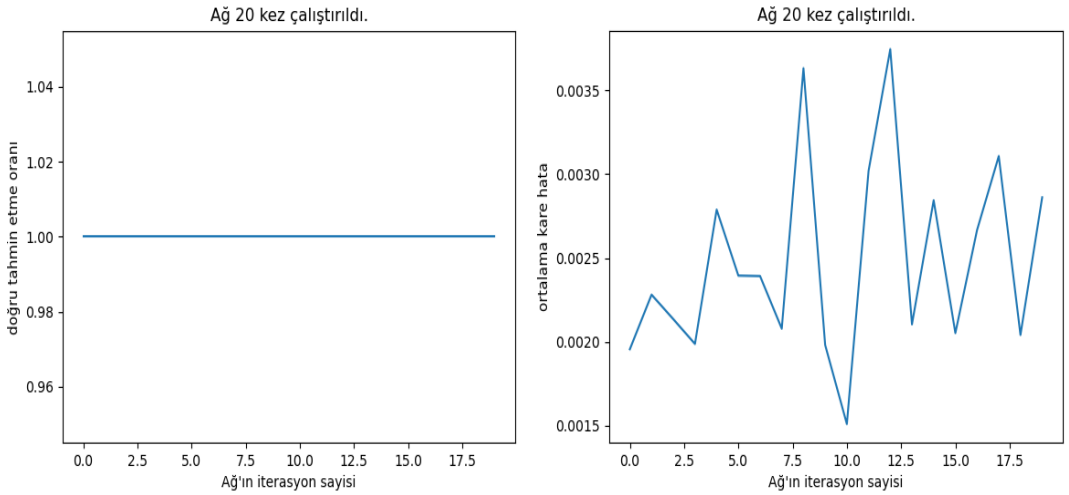




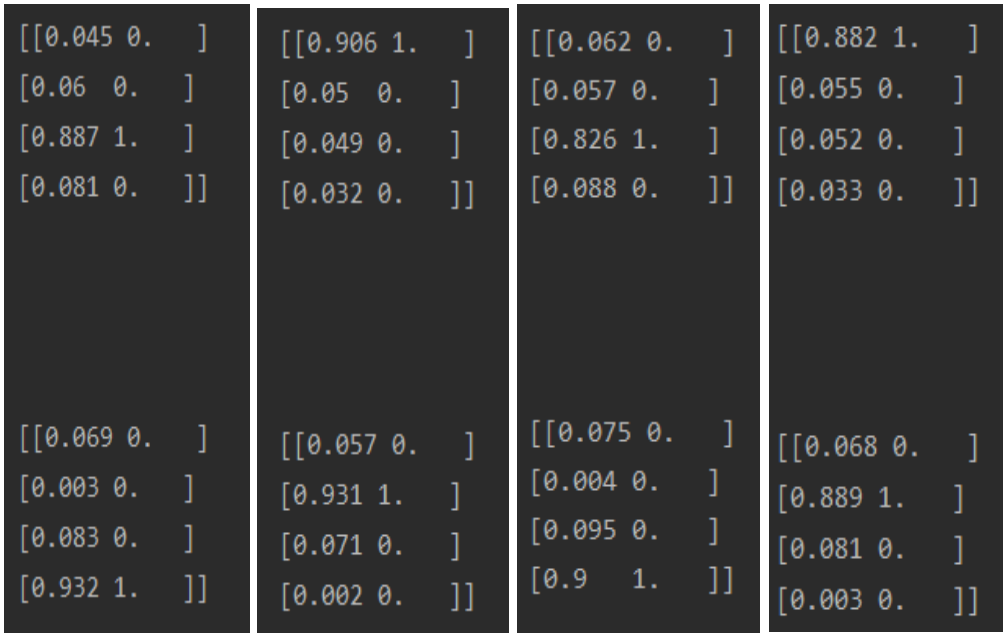
Eğitim kümesindeki örüntüleri doğru tahmin etme grafiği de ortalama kare hata grafiğine uyumlu bir şekilde takip ediyor.

d) Test kümesi ile performans analizi

Analiz için öğrenme hızı 0.4 olarak seçildi. Ağı 20 kere çalıştırdığımızda aldığımız doğruluk oranı ve test ortalama hataları:



20 ve 10 nörondan oluşan gizli katmanlar kullandığımız ağ ile örüntülerimizi doğru sınıflandırdığımızı gözlemliyoruz. Test hata Ortalamalarımızın da doğru tahmin etme grafiğini doğrular şekilde olduğunu söyleyebiliriz.

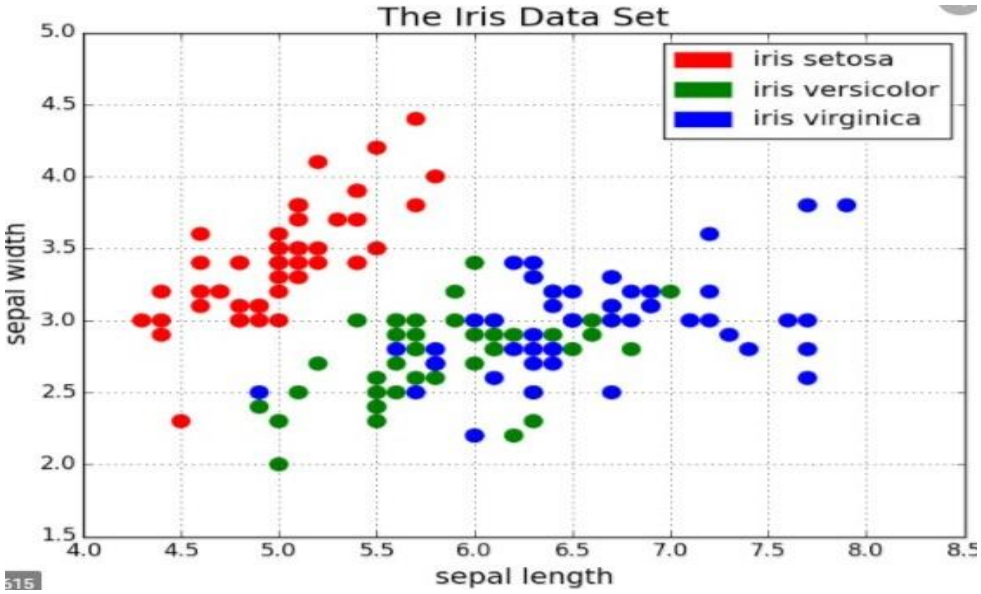


Ağ'ı 1 kez çalıştırarak gerçek test değerleri çıktıları ve tahmin edilen değerleri bu şekilde kıyasladık.

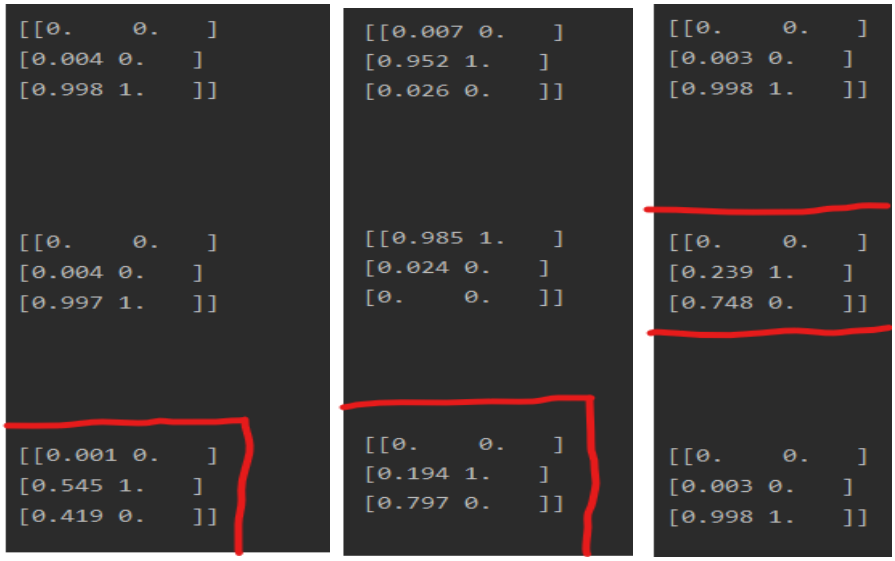
SORU-2) İris Veri Kümesi

Bu veri kümesini sınıflandırmak için 1. Sorudaki ağ yapısını kullandık. Her bir çiçeğin çanak yaprak uzunluğu, çanak yaprak genişliği, taç yaprak uzunluğu ve taç yaprak genişliği olmak üzere 4 özelliği olduğu için giriş katmanındaki nöron sayısını 4 olarak seçtik, 3 farklı çiçek olduğu için de çıkış katmanındaki nöron sayısını 3 olarak seçtik. Başlangıç ağırlık değerlerini -0.5 ile 0.5 arasında rastgele sayılar seçtik. İlk gizli katman nöron sayısını 30 ikinci gizli katman nöron sayısını 20 olarak seçtik. Iris Setosa için [1 0 0] Iris Versicolor için [0 1 0] Iris Virginica için [0 0 1] etiketlerini kullandık. Giriş değerlerini 0 ve 1 arasına sıkıştırmak için verimizi normalize ettik. Veri setindeki ilk 120 örüntüyü eğitim için geriye kalan 30 örüntüyü de test kümesi için aldık. Ağı bir defa öğrenme hızı=0.4 olacak şekilde çalıştırdığımızda ağımız çoğunlukla test kümesindeki örüntüleri doğru tahmin ediyor, bazen Iris Versicolor ile Iris Virginica türlerini birbirine

karıştırıyor. Böyle bir durumun söz konusu olabileceğini aşağıda türlerin çanak yaprak genişliği ile uzunluklarına göre çizdirilen grafikten anlayabiliyoruz:



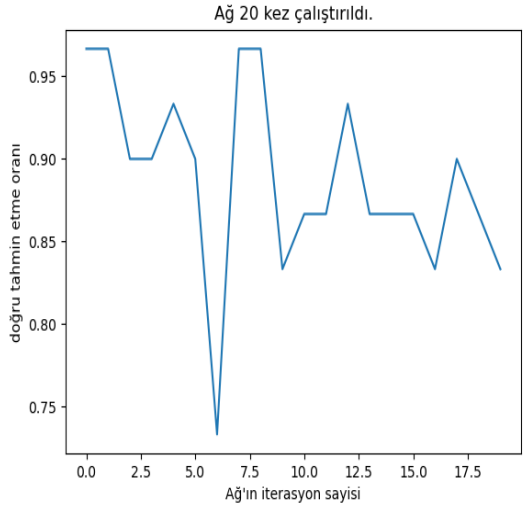
Ağı 1 kez çalıştırarak tahmin ve gerçek test çıktıları arasındaki farkları inceledik.

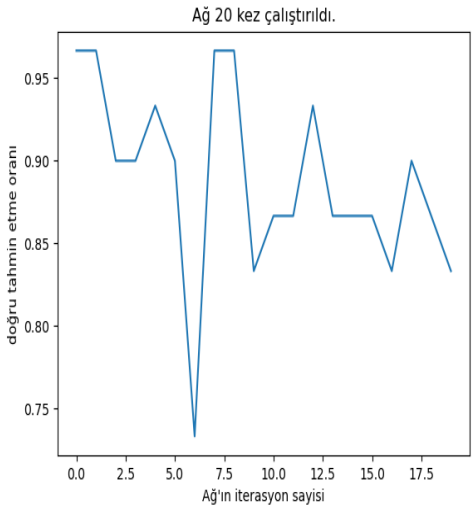


Ağımız bazi tahminlerde Versicolor ve Virginica'yı birbirine karıştırıyor.

Test Ortalama Kare Hatası ve Doğruluk Oranı Grafikleri

Ağımızı 20 defa öğrenme hızı her seferinde 0.4 olacak şekilde çalıştırdığımızda test için ortalama kare hata grafiği ve doğruluk oranı:





Görüldüğü üzere, örüntü tanımadaki kadar iyi doğruluk oranı ve ortalama hata çıktısı alamadık. Fakat yine de modelimiz yüksek bir oranla verileri doğru sınıflandırıyor.

SORU-3) Billings Sistemi

a) İlk önce 150 değerden oluşan bir boş y vektörü oluşturduk. Başlangıç değerlerini yani y[0] ve y[1]'i 0 ve 0.5 arasında rastgele sayılar olarak seçtik. Ardından e[k] gürültü vektörümüzde aynı şekilde oluşturuldu. Sonrasında, sistemin formülüne göre gürültü de eklenerek çıkış değerleri oluşturuldu. Ağın girişine verilecek olanlar y[k-2], y[k-1] ve e[k] seçilirken çıkışı y[k] seçtik. Giriş katmanında 3, çıkış katmanında 1 nöronumuz var. Değerlerimiz -1 ile 1 aralığında değiştiği için aktivasyon fonksiyonumuzu hiberpolik tanjant olarak belirledik, bu problem için gizli katman nöron sayılarını da 8 ve 4 seçtik.

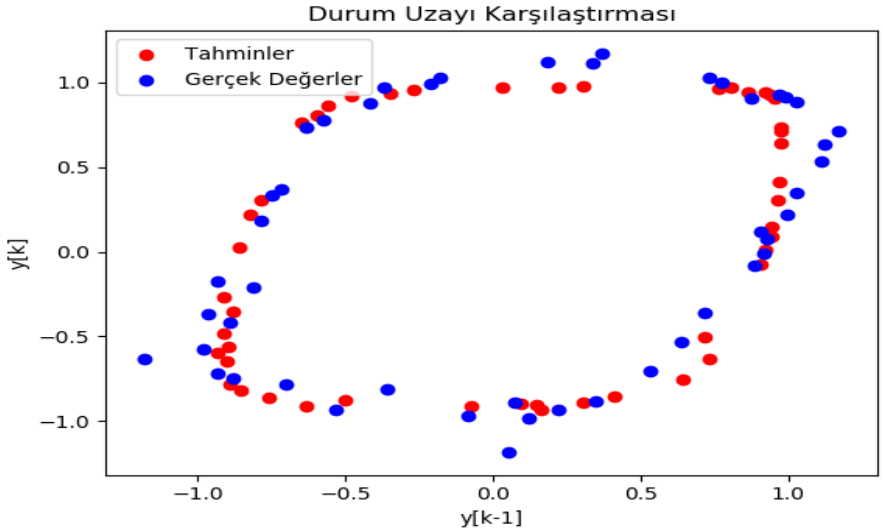
```
firstFeatureTrain = y[:100].reshape(-1, 1)          # y[k-2]
secondFeatureTrain = y[1:101].reshape(-1, 1)        # y[k-1]
thirdFeatureTrain = e[2:102].reshape(-1, 1)         # e[k]
# Eğitim kümesinin sütunları(feature) oluşturuldu.
```

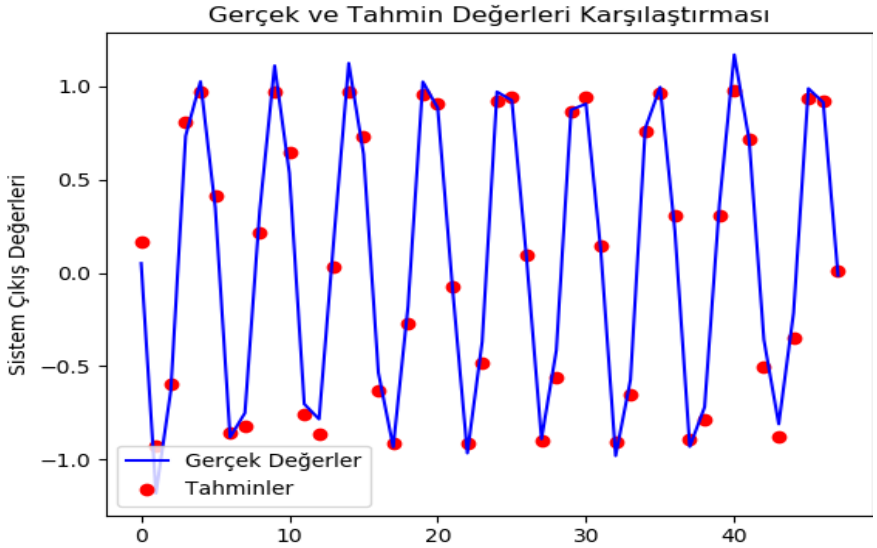
```
trainSet_y = y[2:102].reshape(-1,1)      # Eğitim kümesi çıktıları oluşturuldu. #y[k]
```

b) Sistemimizin ilk 100 çıkışını eğitim kümesi kalan 50 çıkışını da test kümesi olarak aldık. İlk gizli katman nöron sayısı 8, ikinci gizli katman nöron sayısı 4 olmak üzere 0.5 öğrenme hızıyla ağımızı eğittik. Eğitim ve test için ortalama kare hata sonuçlarımız:

```
Eğitim için Ortalama Kare Hata: 0.004023966574844921, iterasyon sayısı: 300
Test ortalama kare hatası:  0.0074151875
```

Bu hata değerlerine karşılık düşen verilen sistemin gerçek çıkış değerleri ile ağımızın tahmin ettiği çıkış değerlerini ve durum uzayında y[k-1] ve y[k] değerlerini karşılaştırdığımızda elde ettiğimiz grafikler de yukarıdaki hata oranlarını doğrular nitelikteydi.





Aktivasyon fonksiyonumuz tanh olduğu için gerçek sistemin -1 ve 1 değerlerinden daha küçük ve daha büyük olduğu bazı noktalarda ağımız ufak bir sapma göstermiştir.