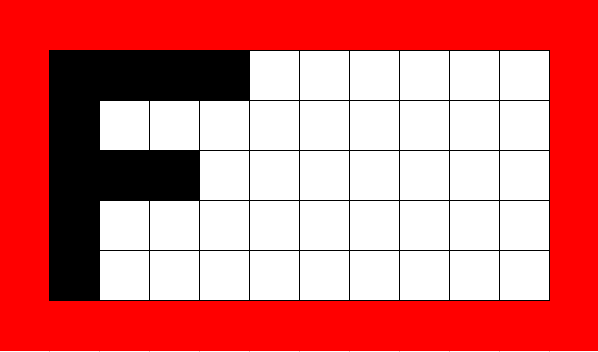
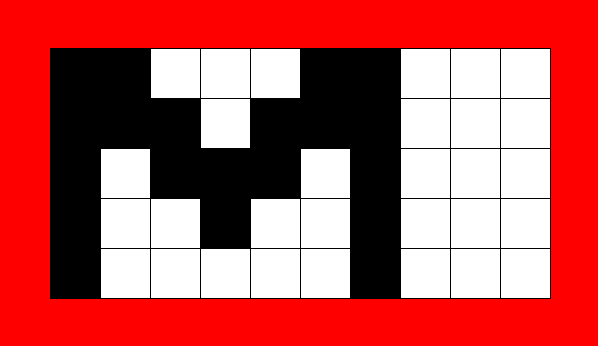
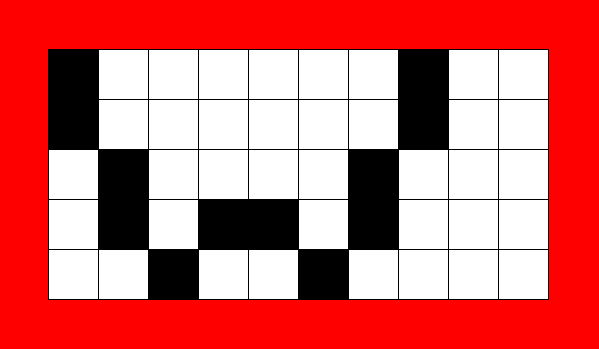
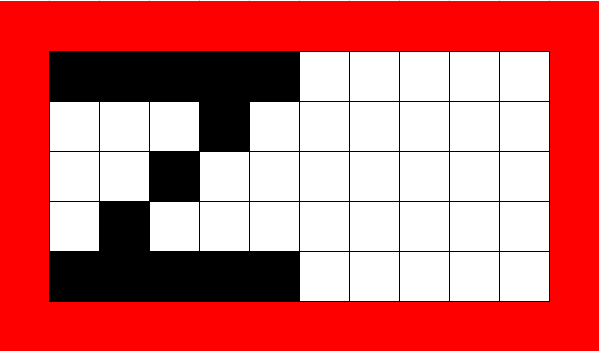
**Yapay Sinir Ağları**

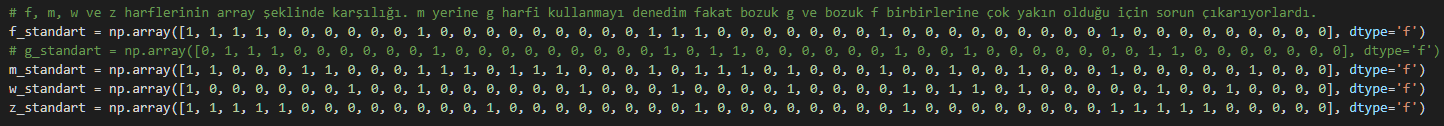
**2.Ödev**

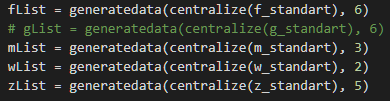
**15.12.2020**

**Mehmet Şerbetçioğlu -- 040160056**

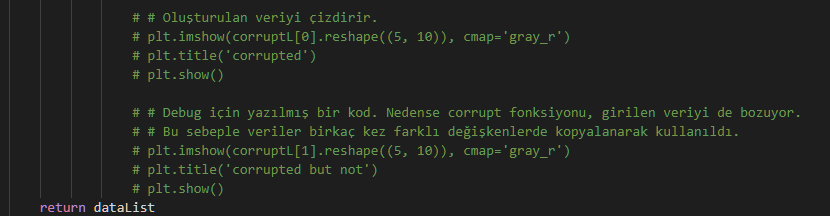
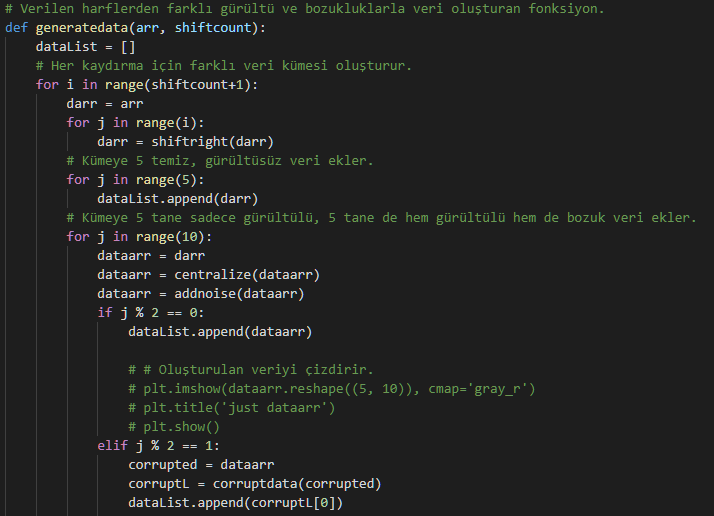
**Ahmet Hulusi Tarhan -- 040170738**

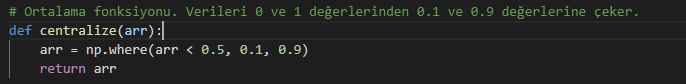
Sorunun ilk kısmında bizden 5x10luk düzlemde oluşturulan 4 farklı örüntü ile bir ağı eğitmemiz isteniliyor. Bunun için, bozulma sonrası dahi birbirine benzemeyecek 4 harf, “f”, “m”, “w” ve “z” seçildi. İlk olarak “g” harfi seçilmişti fakat bozulma sonrası “f” ile çok benzerlik gösterdiği için yerine “m” harfi eklendi. Daha sonra harfler dizi haline getirildi.

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

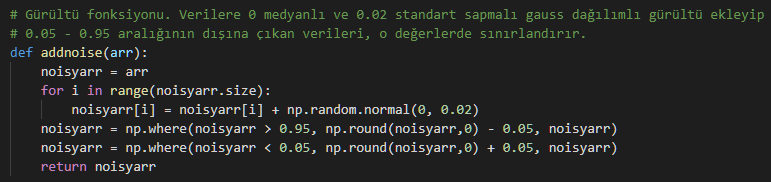
Daha sonra bu diziler, ortalanarak, gürültü eklenerek ve bozularak farklı veriler oluşturuldu. Harflerin düzlemdeki konumu farklı olsa bile sistemin harfi tanıması için harfler düzlem boyunca sağa kaydırılarak, her kaydırmada yeni veri oluşturuldu. Siyah kareler “1” ve beyaz kareler “0” iken bu değerler ağın daha iyi çalışması için “0.9” ve “0.1”e çekildi, buna ek olarak gürültü sonrası oluşan değerler “0.95” ile “0.05” arasında sınırlandırıldı. Bu değişiklikler, ağırlıkların her veri için güncellenmesini sağlayacak ve kullanılacak fonksiyon için verileri belli bir sınırda tutacak. 

**Veri oluşturma fonksiyonu:**

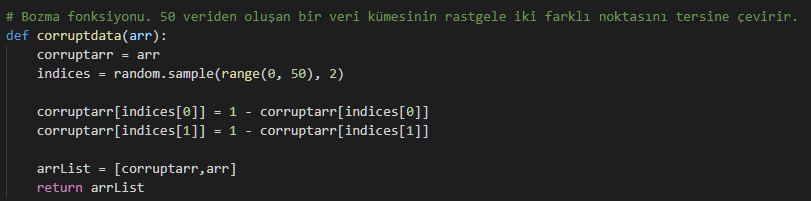
Toplam kaydırma miktarı (shiftcount), döngü miktarını belirler. Her iterasyonda, o anki iterasyon sayısı kadar kaydırılıp veri elde edilecektir.

**Verileri ortalama fonksiyonu:** ****

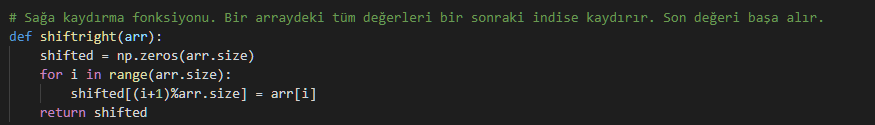
**Gürültü ekleme fonksiyonu:**

Verilere 0 medyan ve 0.02 standart sapma değerlerine sahip gauss dağılımına sahip gürültü ekler. 

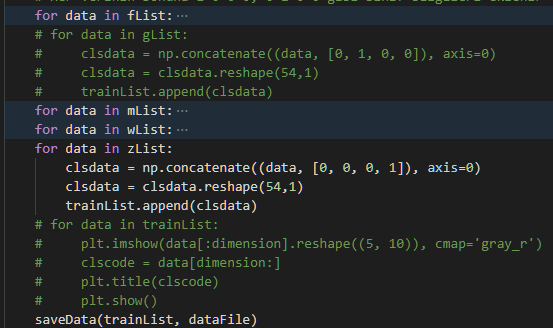
**Bozma fonksiyonu:**

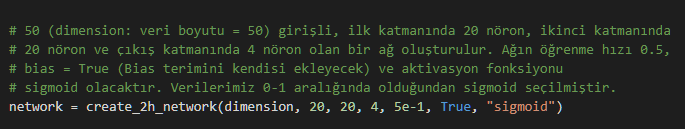
0-49 arasında iki farklı sayı seçilip verideki bu değerler 1’den çıkararak tersine çevirir. 

**Kaydırma fonksiyonu:**

Her elementin indisini bir artırır, son elementi başa alır. Örüntü bu şekilde sağa kaymış olur. **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

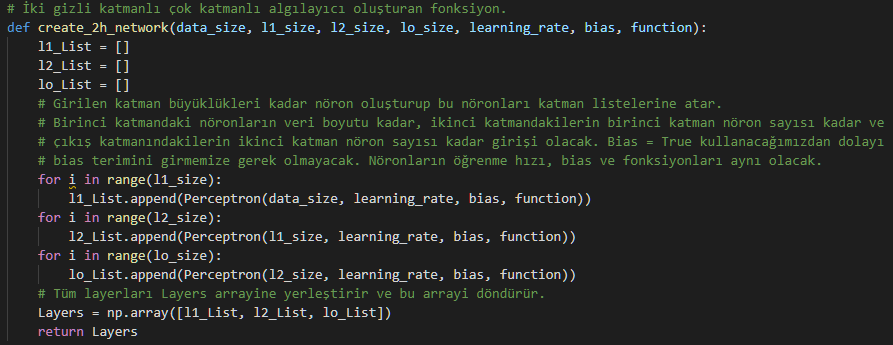
Oluşturulan veriler farklı sınıfların listelerine dahildir. Her verinin sonuna; “f”: [1 0 0 0], “m”: [0 1 0 0], “w”: [0 0 1 0], “z”: [0 0 0 1] olmak üzere sınıf bilgileri eklenir. Sınıf bilgisi eklenen veriler eğitim kümesine yerleştirilir ve bu küme kaydedilir. Ağ tekrar çalıştırıldığında yeni veri üretmek yerine bu veriler kullanılacaktır.

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

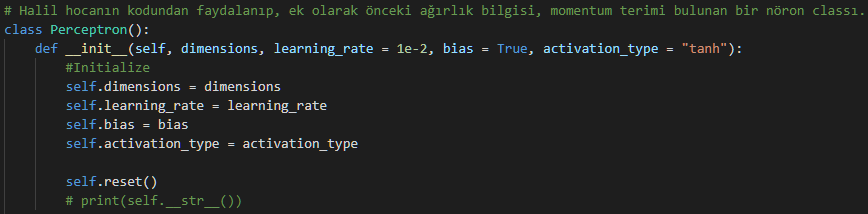
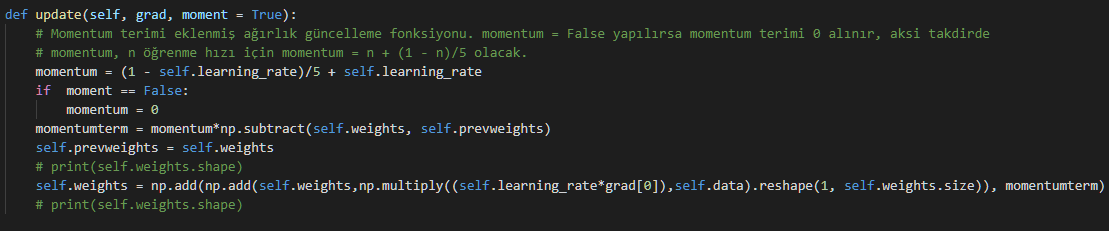
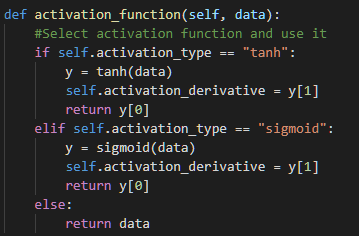
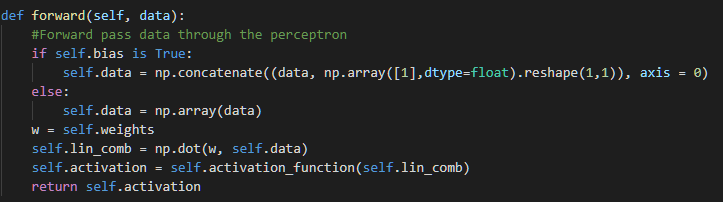
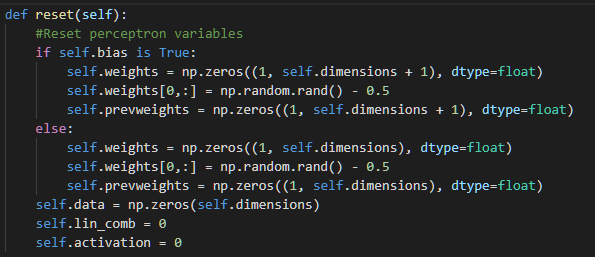
Eğitim kümesi oluşturulduktan sonra, eğitilmek üzere iki gizli katmana sahip bir yapay sinir ağı yapısı oluşturulur. Veriler 0-1 aralığında olduğu için aktivasyon fonksiyonu olarak sigmoid seçilmiştir.

**Yapay sinir ağı oluşturma fonksiyonu:**

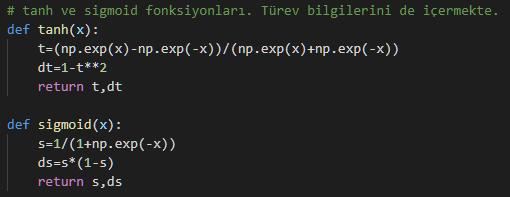
Bu katmanın oluşumunda “Perceptron” sınıfı kullanılmıştır. Bu sınıf, Halil Durmuş’un paylaştığı ağ yapısından alınmış ve biraz değişiklik yapılmıştır.

****

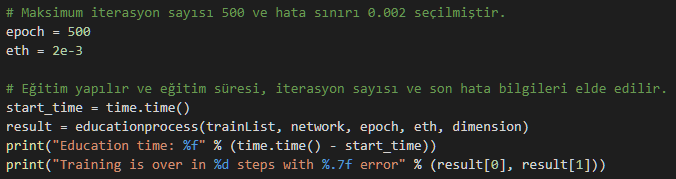
**Perceptron sınıfı:**

Sınıfta ek olarak momentum terimi ve önceki iterasyonun ağırlık bilgisi bulunmaktadır. Ayrıca aktivasyon fonksiyonu için kullanılan fonksiyonlar, lokal gradyan hesaplamasında kolaylık sağlaması için türevlerinin de bulunduğu farklı fonksiyonlar haline getirilmiştir.****

**Aktivasyon fonksiyonları:**

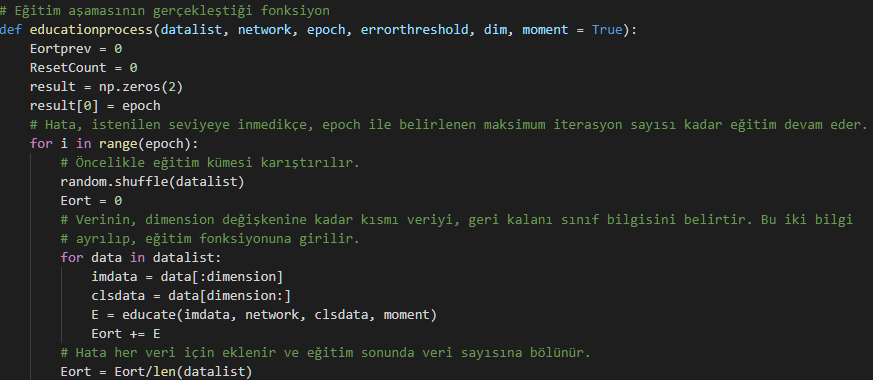
****

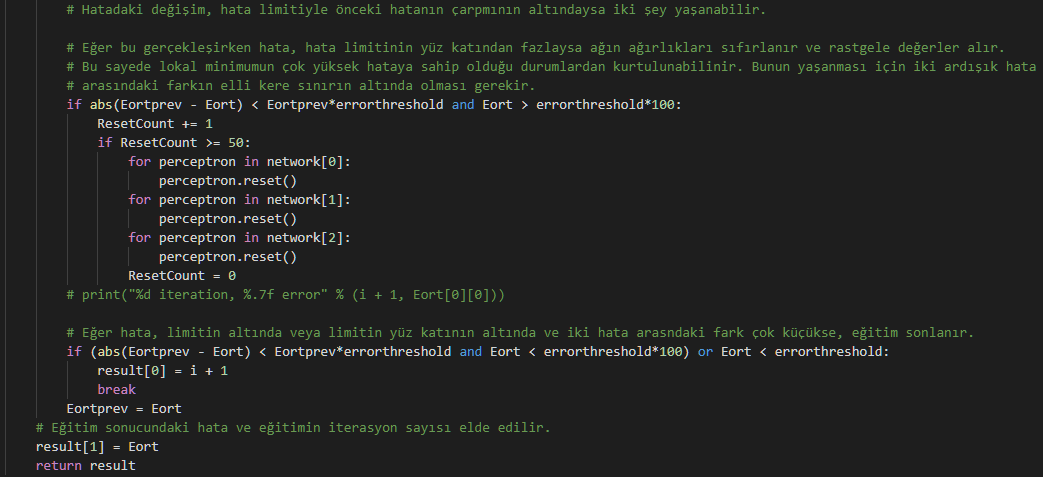
**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

Maksimum iterasyon sayısı 500, hata limiti ise 0.002 belirlenip eğitim başlatılır. 

**Eğitim işlemi fonksiyonu:**

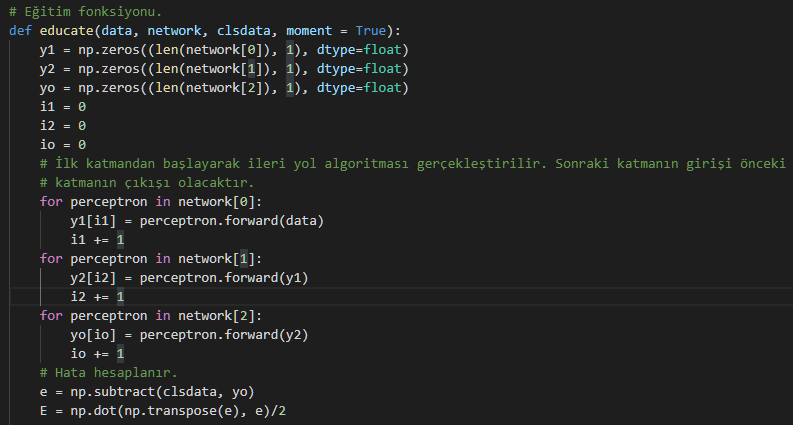
Eğitimin her iterasyonunda eğitilen veriler karıştırılır. Her veri, sınıf bilgisi ayrılarak ağı eğitir. Her iterasyonda ortalama hata elde edilir.

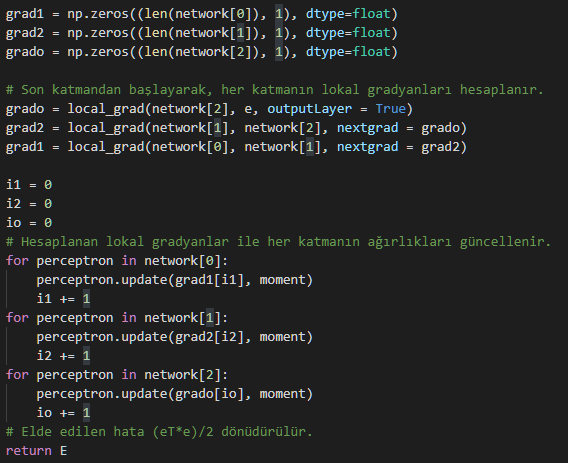


Eğitimin durması için ya hatadaki değişimin belli bir değerin altına düşmesi ya da hatanın belirlenen hata limitinin altına düşmesi lazımdır. Uzun süre yüksek bir hata değerinde kalması sonucunda ağırlıklar rastgele bir şekilde yeniden atanır.

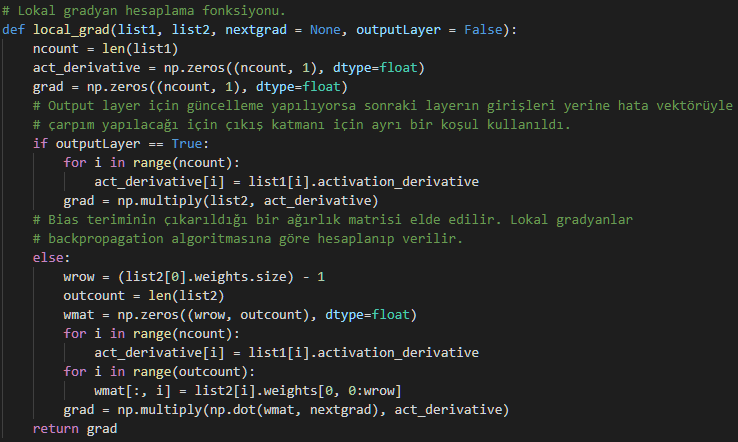
**Eğitim fonksiyonu:**

Sırayla ileri yol izlenir. İlk katmana veriler, ikinci katmana birinci katman çıkışları ve son katmana ikinci katman çıkışları verilir. Son katman çıkışları sınıf bilgisiyle karşılaştırılıp hata bilgisi elde edilir.



Elde edilen hata bilgisi ile her katmanın lokal gradyanı elde edilir. Ardından lokal gradyanlar da kullanılarak ağdaki ağırlıklar güncellenir. 

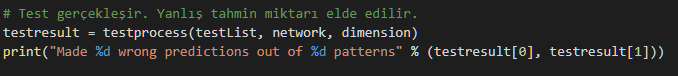
**Lokal gradyan fonksiyonu:**



**Eğitim sonucu:**

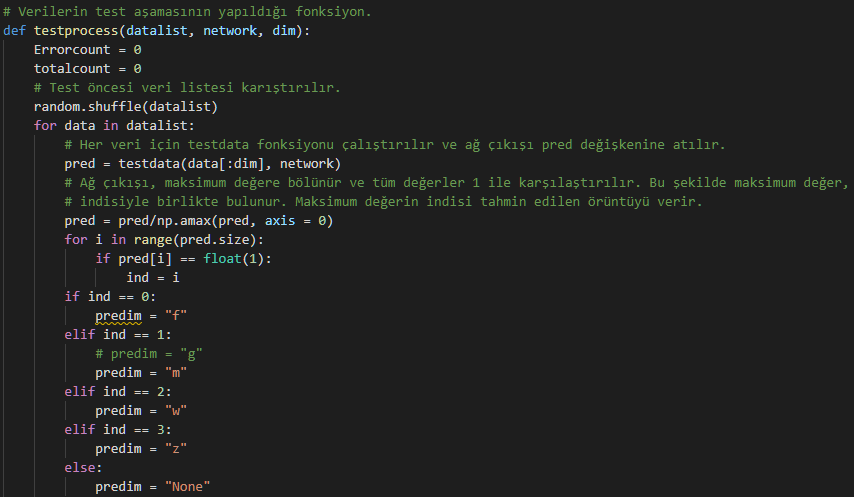
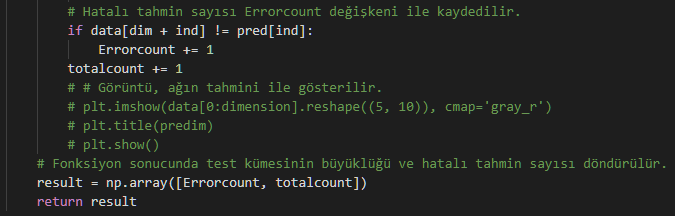
****

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

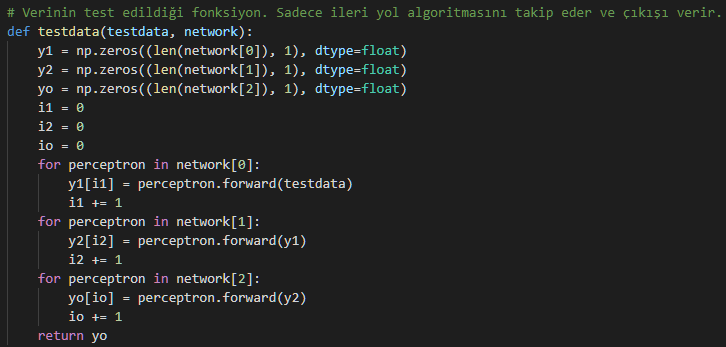
Eğitim sonrası, eğitim kümesinde olduğu gibi yeni veriler oluşturulur ve test kümesine eklenir. Bu veriler kullanılarak ağın örüntüleri tahmin etmesi sağlanır. 

**Test işlemi fonksiyonu:**

Test kümesindeki verileri ağdan geçirip ağ sonucunu sınıf bilgisiyle karşılaştırır.

****

**Test fonksiyonu:**

Eğitim fonksiyonunun ilk aşaması ile verileri ileri yoldan geçirip ağ çıkışını elde eder. 

**Test sonucu:**

300 veri içinden sadece 1 yanlış tahmin ile, %0.33’lük bir hata oranına sahiptir. Bu hata farklı denemelerde değişebileceği gibi hata limitinin düşürülmesi ve eğitimin optimize edilmesiyle de geliştirilebilir.

****

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Momentum varken değişken öğrenme hızı:**

Öğrenme hızı 0.1-0.4 aralığında iken iterasyon sayısının düşük olduğu, hatta 0.4’e kadar azaldığı görülür. 0.4’ten sonra düzenli bir artış yaşanmaktadır. Bunun sebeplerinden biri seçilen durdurma kriteridir. Hatanın, belirlenen sınırın altına inmesinin yanısıra hatadaki değişimin belli bir sınırın altına inmesi de eğitimi durduran bir faktördür ve bu kriterin gerçekleşmesi düşük öğrenme hızları için çok daha kolaydır. Ayrıca öğrenme hızı 0.8 ve 0.9’ken maksimum iterasyon sayısı olan 500’e ulaşıldığı, eğitimin tamamlanamadığı görülür.

Eğitim süresi de, ağ yapısı değişmediği için eğitim iterasyonuyla aynı orantıda ilerlemektedir. 0.8 ve 0.9 noktalarında eğitim maksimum iterasyona ulaştığı için, maksimum eğitim süresinin olarak yaklaşık 192 saniye sürdüğü görülür.

Hata oranı 0.3’ten itibaren sonra artış göstermektedir. 0.3 noktasına kadar hata oranı %1 değerinin altındayken bu noktadan sonra %73’e kadar artmaya devam eder. 0.8 ve 0.9 noktalarındaki yüksek hata eğitimin tamamlanamamasından kaynaklanmaktadır fakat eğitim tamamlansa dahi 0.5-0.7 aralığındaki hızlarda %10un üzerinde hata görülmektedir.

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Momentum yokken değişken öğrenme hızı:**

Momentum terimi yokken, öğrenme hızının küçük olduğu değerlerde iterasyon sayısı momentumun olduğu iterasyon sayısına benzemektedir. Fakat beklenmedik bir şekilde momentum teriminin çıkarılmasıyla öğrenme hızı yükseldikçe iterasyon sayısı da düşer. Momentum varken minimum iterasyon sayısı 0.4 öğrenme hızında 34.2’yken, olmadığı durumda öğrenme hızı 0.9’da 15.4’tür.

Eğitim süresi yine ağ yapısı değişmediği için iterasyon sayısıyla doğru orantılıdır. Öğrenme hızının artmasıyla iterasyon sayısının azalması, eğitim süresini fazlasıyla geliştirmektedir. Momentum varken eğitim minimum 13 saniyede tamamlanırken, olmadığı durumda 0.9 öğrenme hızında yaklaşık 6 saniyede tamamlanmaktadır. Minimum eğitim hızı hemen hemen yarıya düşmüştür.

Aynı eğilim hata oranında da görülmektedir. Momentumla 0.4 noktasından itibaren hatada artış görülürüken olmadığı durumda hata 0.2’den itibaren %0.5’in üstüne çıkmamıştır. 0.1 noktasında %4lük bir hata görülmektedir. 0.9 noktasında %0.066’lık bir hata görülür. Hem eğitim süresi hem de hata oranı olarak 0.9 noktası bu deneyde optimum noktadır, hatta daha yüksek eğitim hızları daha iyi sonuçlar verebilir.

Momentum teriminin eğitim süresinin kısaltarak hata oranını düşürmesi beklenmektedir. Elde edilen sonuçlar momentumun eksik bir şekilde uygulandığına işaret eder.

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Momentum yokken değişken birinci katman nöron sayısı:**

Öğrenme hızı 0.5’te sabitlenip momentum terimi olmadan, ilk katmandaki nöron sayısı değiştirilerek ölçümler yapılır. Eğitim iterasyonunda fazla bir değişim görülmemiştir. 5 nörondan 20 nörona kadar yaklaşık 7 iterasyon azalmıştır. Daha sonra 50 nörona kadar yaklaşık 16 iterasyonluk bir artış yaşanmıştır. Genel anlamda iterasyon sayısı birbirine çok uzak değildir.

Eğitim süresi ise iterasyonun aksine sürekli artan bir yol izlemektedir. İterasyon sayısı tüm değerlerde birbirine yakın olsa da, nöron sayısının değişmesiyle güncellenmesi gereken ağırlık sayısı değiştiği için, her iterasyonun süresi değişir. Bu sebeple nöron sayısı arttıkça eğitim süresi de artmaktadır.

Hata oranı, 15 nörona kadar azalmış ve sonrasında artış göstermiştir. 10-40 aralığında hata oranı %1’in altındayken diğer noktalarda %1.5’in üstündedir. Bu optimal nöron sayısı seçimini güçleştirir. Hata oranı, az nöron kullanıldığında düşük veya çok nöron kullanıldığında düşük gibi ifadelerde bulunulamaz. Ayrıca eğitim süresinin de hesaba katılması gerekir. Bu deney için, 10 nöronun eğitim süresinde, 15’in ise hata oranında avantajlı olduğu, ikisinin de farklı kriterlerde en uygun noktalar olduğu görülür.

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

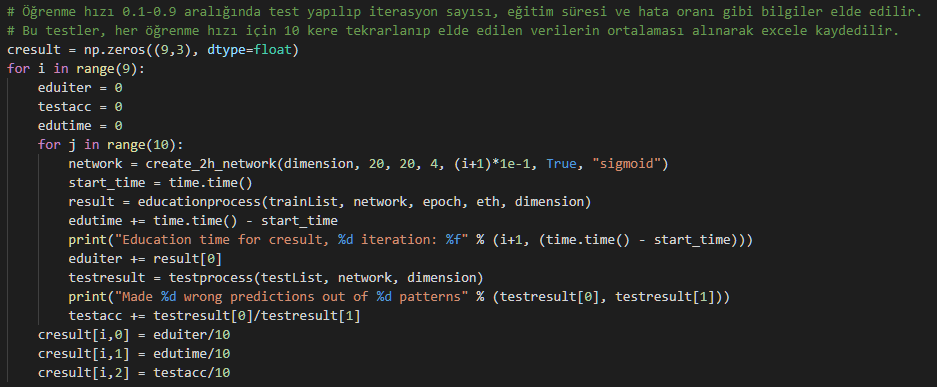
**Momentum yokken değişken ikinci katman nöron sayısı:**

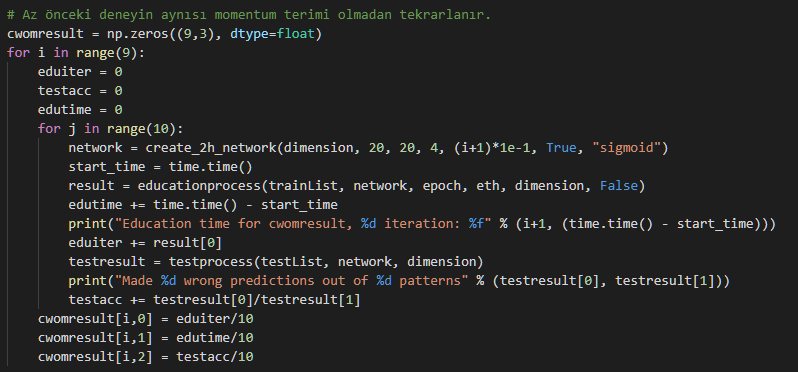
İlk katmanın aksine, ikinci katmandaki nöron sayısı değiştiridiğinde eğitim iterasyonu genel anlamda azalmaktadır. 5 nöron kullanıldığın eğitim 45 iterasyonda bitiyorken 25 nöron ve sonrasında yaklaşık 25 iterasyonda bitmeye başlamaktadır.

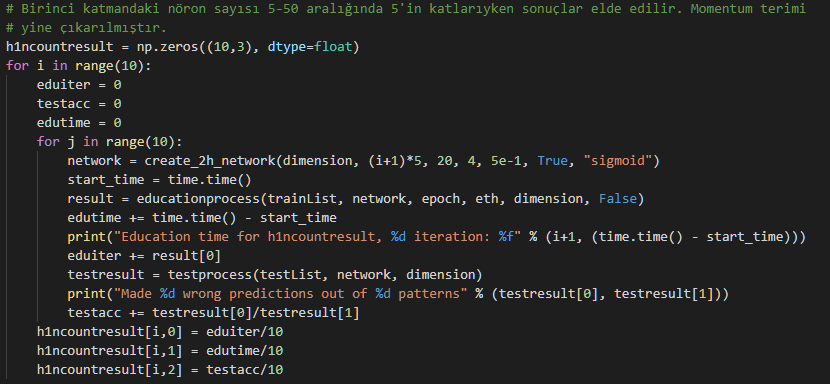
İterasyondaki bu azalmaya rağmen nöron sayısı ve beraberinde ağırlık sayısının artışı daha hızlıdır. Eğitim süresi, önceki deneye nispeten daha yavaş da olsa, genel anlamda sürekli olarak artmaktadır.

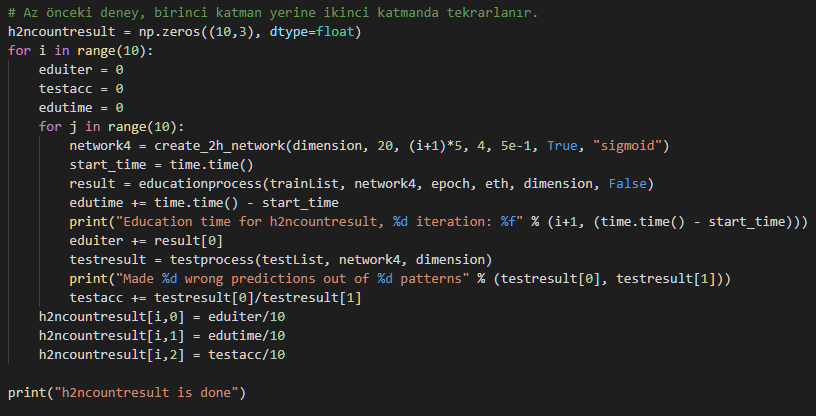
Hata oranı ise daha nöron sayısının artmasıyla sürekli azalmaktadır. 5 nöron kullanıldığında %0.8’lik bir hata oranı söz konusuyken bu oran %0.1’in altına düşebilmektedir. Hata oranının eğitim süresinden daha önemli olduğu uygulamalarda ikinci katmanda daha çok nöron kullanmak daha iyi sonuçlar verebilir.

**Deney Kodları:**

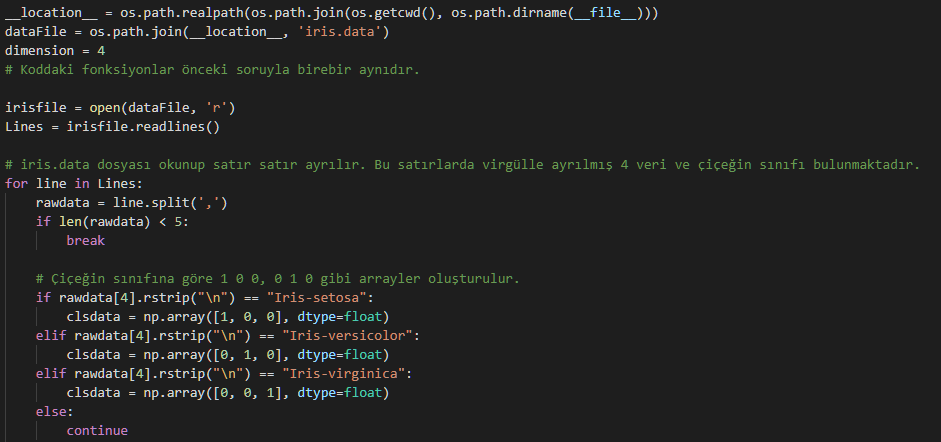
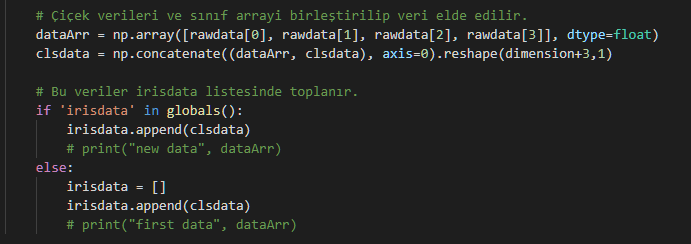
**Momentum varken değişken öğrenme hızı:** 

**Momentum yokken değişken öğrenme hızı:** 

**Momentum yokken değişken birinci katman nöron sayısı:** ****

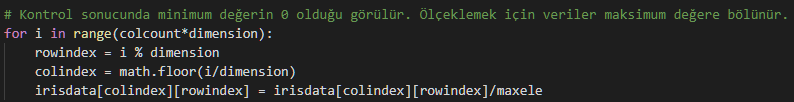
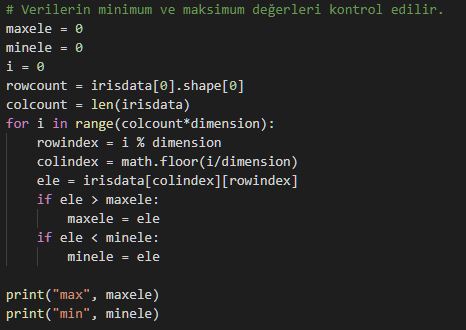
**Momentum yokken değişken ikinci katman nöron sayısı:** ****

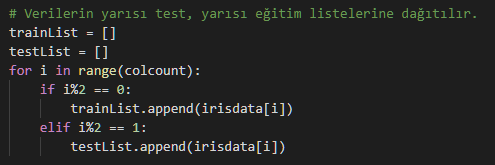
**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

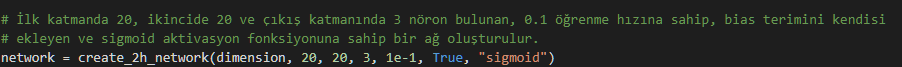
İlk kısımda elde edilen ağ, verilen iris.data dosyasındaki veriler ile eğitilip üç farklı iris çiçeğinin ayırt edilmesi için kullanılacaktır. Öncelikle iris.data dosyası okunur ve farklı veriler olarak bir listeye eklenir. Sınıf bilgisi ilk kısımdaki gibi verinin sonunda olup, üç ayrı sınıf olduğundan dolayı iris setosa: [1 0 0], iris versicolor: [0 1 0] ve iris virginica: [0 0 1] şeklindedir.  

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

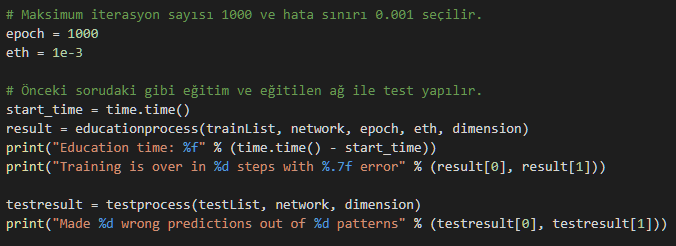
Ölçeklendirilmek üzere verilerin maksimum ve minimum değerleri hesaplanır. Veri büyüklükleri 0-1 aralığına getirilir, ağda aktivasyon fonksiyonu için sigmoid fonksiyon kullanılacaktır.



Ölçeklenen veriler test ve eğitim kümelerine dağıtılır. **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

İlk katmanında 20, ikincide 20 ve çıkış katmanında 3 nöronlu bir ağ oluşturulur. Eğitim hızı veriler birbirine yakın olduğundan daha isabetli bir eğitim için düşük (0.1) seçilmiştir.  **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

Maksimum 1000 iterasyon ve 0.001 hata sınırı ile ağ eğitilir. Eğitilen ağ test kümesiyle test edilir.



**Eğitim ve test sonuçları:**

Eğitim, biraz uzun bir süre olarak 35 saniyede tamamlanmıştır. Eğitim 183 iterasyon sürüp 0.04 hata ile sonlanmıştır. Hatanın hata sınırından yüksek olmasının sebebi, hatadaki değişimin çok azalmasındandır. Test sonucunda ise 75 çiçekten 2’si yanlış tahmin edilmiştir. Eğitimin uzun sürmesi, eğitim ve testteki yüksek hata oranları çiçeklerin birbirine çok benzer özellikler göstermesinden dolayıdır. Eğitilen veriler bu seviyede benzerlik gösterdiğinde hata miktarı lokal minimumlarda takılmaya daha yatkındır. Bu isabetli eğitimi fazlasıyla güçleştirir.

****

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Momentum varken değişken öğrenme hızı:**

Bu problemde de öncekinde olduğu gibi iterasyon sayısı öğrenme hızının artmasıyla artmaktadır. Öğrenme hızı 0.1’ken yaklaşık 264 iterasyonda eğitim tamamlanıyorken, 0.8 ve 0.9 değerlerinde maksimum iterasyon sayısı olan 1000’e kadar yükselmiş, başarılı bir şekilde tamamlanamamıştır. Önceki problemde iterasyon sayısı çoğunlukla 200’ün altındayken bu problemde minimum değerin 264 olması, eğitilen sınıfların birbirine yakın olmasından ve bu problemde öncekinin yarısı kadar hata limiti kullanılmasından kaynaklanır.

Eğitim süresi de önceki problemdeki gibi iterasyon sayısıyla doğru orantılıdır. Veri sayısı az olduğundan, maksimum iterasyon sayısı önceki problemdekinin iki katı da olsa, eğitim en fazla yaklaşık 93 saniye sürmektedir.

Hata oranında da benzer trendler görülür. Öğrenme hızı 0.4 olana kadar %5’in altında bir hata oranı gözlenirken sonrasında yükselmiş, 0.6’dan sonra ise hızlı bir yükselişle %66’ya kadar çıkmıştır. 0.8 ve 0.9 değerlerinde eğitim başarısız tamamlandığı için yüksek hata oranları gözlenir fakat başarılı tamamlanmış olmasına rağmen 0.7 değeri %31.7 ile yüksek bir hata oranına sahiptir. Hata oranı düşükken bile nispeten yüksek değerlere sahip olması, verinin bu ağ ile eğitilmesinin verimsiz olduğunu gösterir.

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Momentum yokken değişken öğrenme hızı:**

Önceki problemin aksine, momentum terimi çıkarıldığında eğitim iterasyonu düzenli bir eğilim göstermez. Genel anlamda düşük öğrenme hızlarında düşük, yüksek değerlerde yüksek olduğu söylenebilir fakat buna karşı çıkan 0.3, 0.8 değerleri de görülmektedir. Momentumun olduğu durumla farkı, yüksek öğrenme hızlarında bile eğitimin başarılı tamamlanamabilmesidir. Önceki problemde belirtildiği gibi, bu beklenmeyen bir davranıştır. Momentum teriminin eksik uygulanmasından kaynaklanabilir.

Eğitim süresi de, ağ yapısı değişmediği için iterasyonla aynı hareket ettiği görülür. Eğitim süresi olarak, 0.1 noktası haricinde, momentumun olduğu durumdan daha avantajlı olduğu görülür.

Hata oranında da düzenli bir değişim gözlenmez. Hatanın %3’ün altına düşmediği, düşük öğrenme hızlarında momentumlu durumdan daha avantajsız olduğu görülür.

Momentum olmadan kullanılan ağ, eğitim süresi açısından çoğunlukla daha avantajlıyken hata oranında, düşük öğrenme hızlarında, momentumun kullanılması küçük bir oranla daha doğru sonuçlar elde edilmesini sağlıyor. Önceliğin farklı olduğu uygulamalarda, momentumun kullanılıp kullanılmaması fark gösterecektir.

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Momentum yokken değişken birinci katman nöron sayısı:**

Birinci katmandaki nöron sayısının değiştirildiği durumda, önceki problemin aksine iterasyond düzenli bir azalma görülmemektedir. 230-700 iterasyon arasında değişim göstermektedir.

Eğitim süresi de iterasyon sayısı ve ağırlık sayısıyla orantılı olacağı için, iterasyon sayısının biraz daha artan eğilimli bir fonksiyonu olarak görülmektedir. Şaşırtıcı bir şekilde 5 ve 50 nöron sayılarının aradakilerden daha ideal eğitim süreleri verdiği görülmektedir.

Hata oranı açısından da ara değerlerin pek iyi bir performansa sahip olmadığı, eğitim süreleri fazla olmasına rağmen 15, 35 ve 40 nöron sayılarının %3’ün altında hata verdiği görülür.

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Momentum yokken değişken ikinci katman nöron sayısı:**

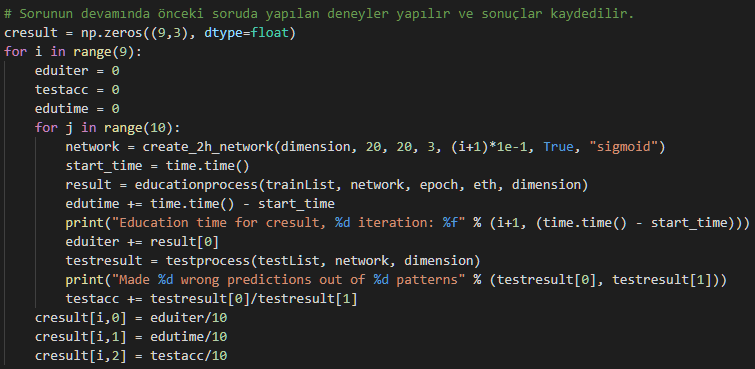
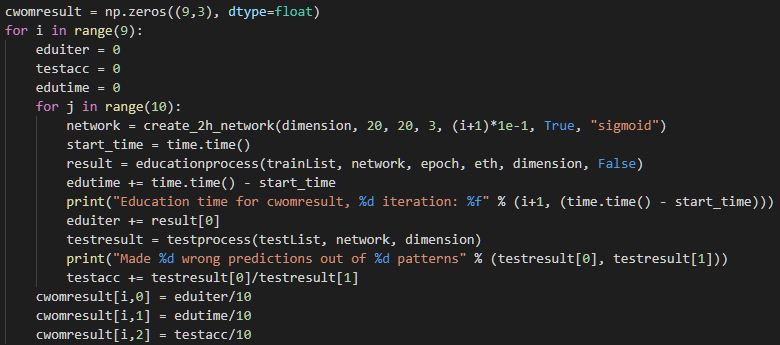
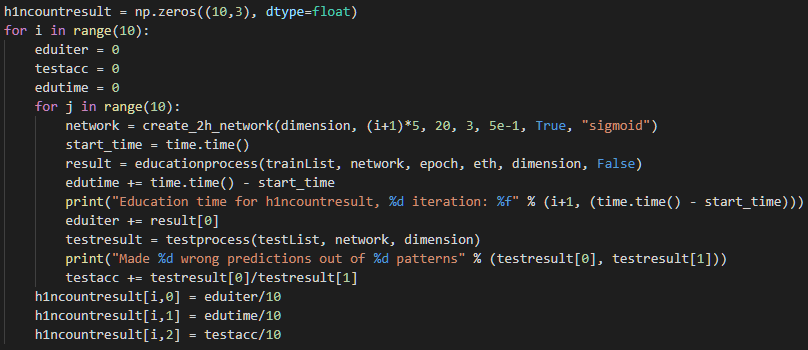
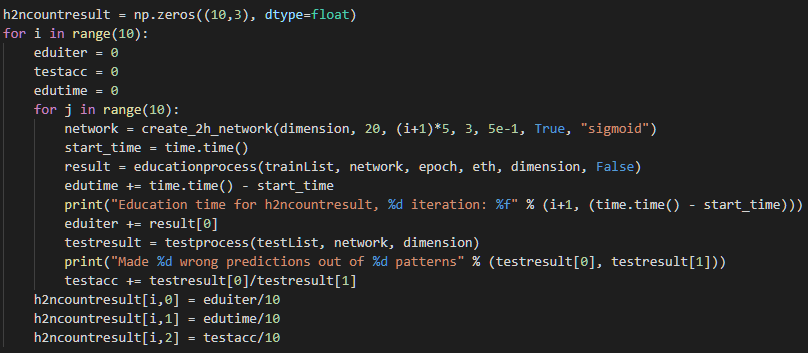
İkinci katmandaki nöron sayılarının değişmesiyle de iterasyon sayısında düzenli bir değişim görülemez. 15 nörona kadar azalıp 459’dan 315’e inmiş, hemen ardından 612’ye yükselip düzensiz bir şekilde azalıp artmaya başlamıştır. Genel olarak 315-612 iterasyon arasındadır.

Eğitim süresi de önceki problemdeki gibi iterasyon sayısının ağırlık sayısıyla çarpımı şeklinde, genel anlamda artan bir yapıya sahiptir. 20-23 saniye civarı sürelerle başlayıp, nöron sayısı arttıkça yaklaşık 65 saniyeye kadar yükselmiştir.

Hata oranı için de düzenli bir değişim olmadığı görülür. Genel olarak %3-%5 civarındayken nöron sayısı 15, 25 ve 40 olduğu durumalrda %5.5 üzerindedir. İlginç olarak, eğitim iterasyonunun en yüksek olduğu 20 nöron durumunda hata oranı %3.2 ile üç minimum değerden biridir. Diğer minimum değerleri 30 ve 35 nörondadır. Hata olarak en optimal değerin 30 nöron kullanıldığında, süre olarak ise 10 nöron kullanıldığında elde edildiği görülür.

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Deney Kodları:**

Önceki problemle aynı kodlara sahiptir. Ağ aynı kalmış olup sadece girilen veriler ve verilerin çekilmesi farklılık göstermiştir.    

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

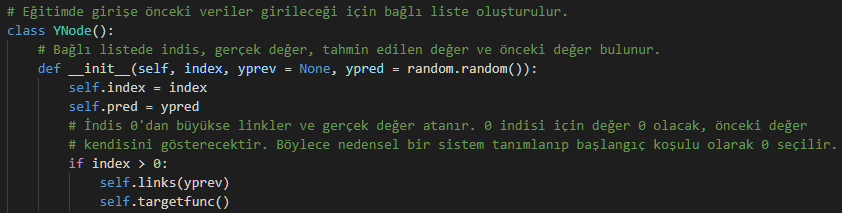
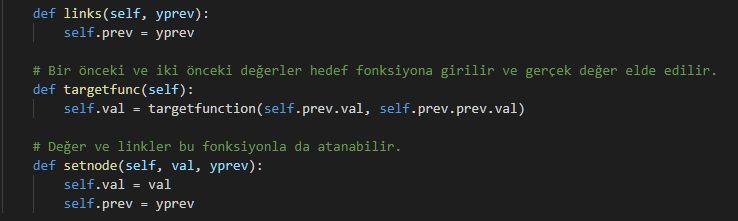
Bu soru için, çok katmanlı algılayıcı kullanarak 

sistemin tanınması ve yaklaşılması isteniyor. Denkleme göz atınca ilk olarak bir fark denklemi olduğu, geçmiş sonuçlara bağımlı dinamik bir sistem olduğu dikkat çekiyor. Bu yapıyı çok katmanlı algılayıcıda gerçeklemek için girişe önceki çıkış bilgileri verilmesi gerecek. Girişteki değerleri için seçilir.

Ek olarak, sistemin bir başlangıç koşuluna ihtiyacı olacağı için y(0) ve negatif k değerleri 0 alınarak sistem aynı zamanda nedensel bir hale getirilmiştir.

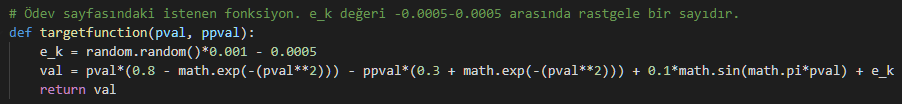
**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

Geçmiş verileri saklamak ve ulaşabilmek amacıyla dizi kullanmak çok mantıklı olmayabilir. Dizide indisler verinin k değerini belirleyebilirken, eğitim sırasında dizi karıştırılmaktadır ve bu verilerin sırasını bozabilir. Bir başka yaklaşım sözlük kullanarak verileri sıralarıyla birlikte saklamak olabilir fakat bunda da verinin geçmiş değerlerini bulmak için tüm sözlüğün aranması gerekebilir ve performansı zayıf olacaktır.

Bu durum için uygun olan bir veri saklama methodu bağlı listelerdir. Bir bağlı liste sınıfı oluşturup geçmiş değerler bu yapılara konulabilir. Oluşturulan bağlı listede verinin indisi, ağ ile tahmin edilen değeri, fonksiyon sonucu gerçek değeri ve önceki değerle linki bulunmaktadır. 

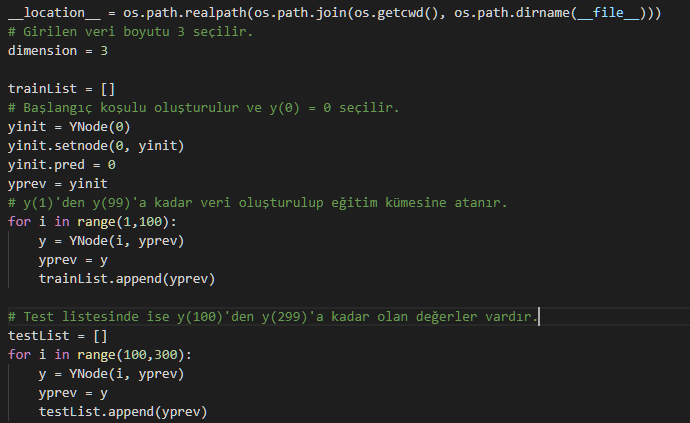
**Hedef fonksiyon:**

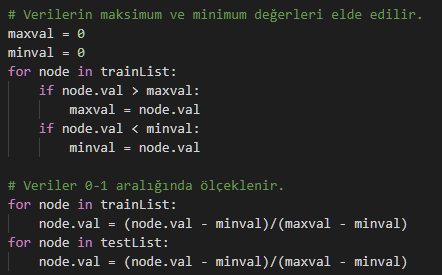
e(k) değeri -5e-4 ile 5e-4 arasında rastgele bir sayı seçilir.

****

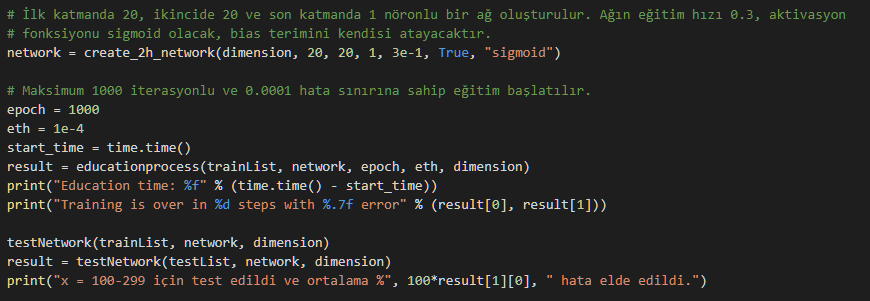
**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

Boyut, 3 giriş olduğu için 3 seçilir. Öncelikle başlangıç değeri olarak y(0) = 0 atanır. 0 değerinin düğümü, geçmiş değer olarak kendini gösterir ve böylece k’nın 0 veya 0’dan küçük değerleri için y(k) = 0 sağlanır.

Daha sonra her iterasyonda bağlanarak k=99’a kadar düğümler oluşturulur ve bu düğümler eğitim kümesine eklenir. Aynı işlem k=100’den k=299’a kadar yapılır ve bu değerlerle de test kümesi oluşturulur. **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

Elde edilen değerlerin maksimum ve minimum değerleri bulunup, sigmoid aktivasyon fonksiyonu kullanmak üzere veriler [0,1] aralığında ölçeklenir. **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

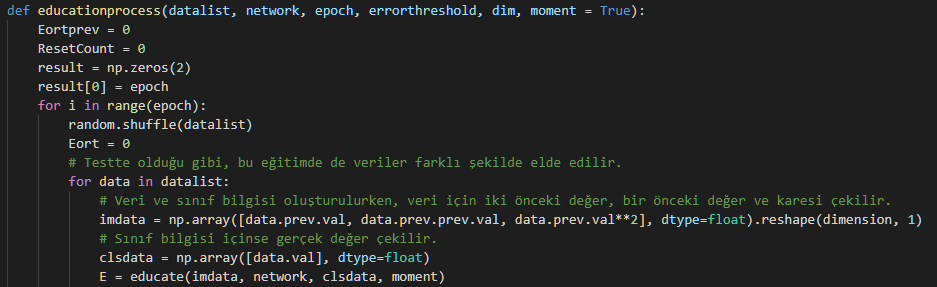
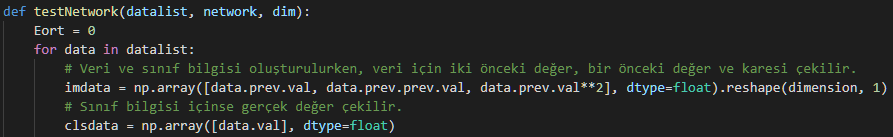
Ardından 3 boyutlu girişe sahip, ilk katmanında 20, ikincide 20 ve son katmanda 1 nöronu olan bir ağ oluşturulur. Ağın eğitim hızı 0.3, aktivasyon fonksiyonu sigmoid seçilir ve ağ bias terimini kendi oluşturacaktır.

Ağ, maksimum 1000 iterasyon ve 0.0001 hata limiti ile eğitime sokulur. Ardından test kümesiyle karşılaştırmak üzere test işlemi yapılır. 

Eğitim; 454 iterasyonla, 70 saniyede 0.002 hata oranıyla bitmiş, test kümesinden yaklaşık %0.32’ye yakın bir hata alınmıştır. Bu değerler eğitimde birçok rastgele işlem olmasından dolayı değişiklik gösterebilir. 

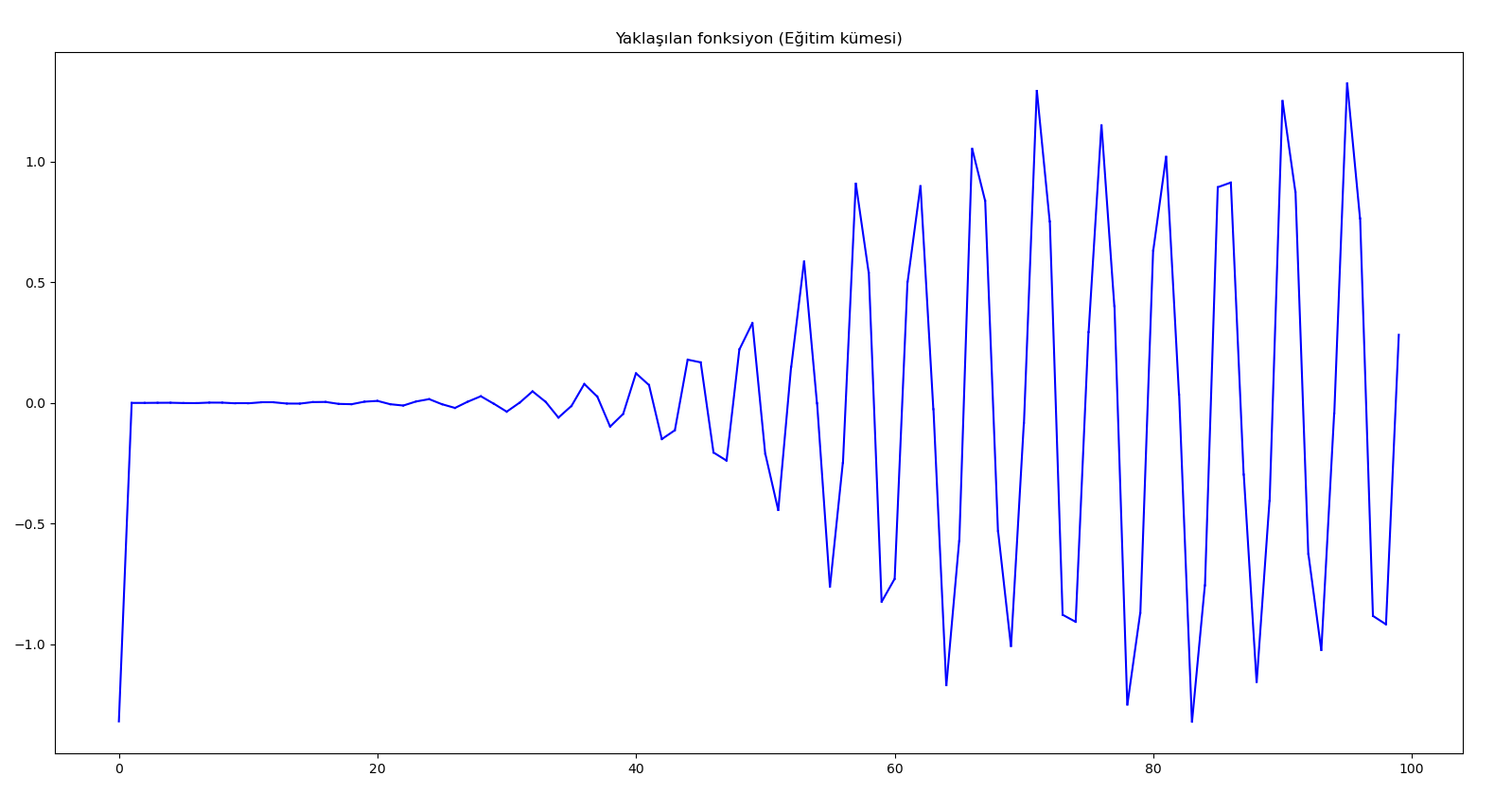
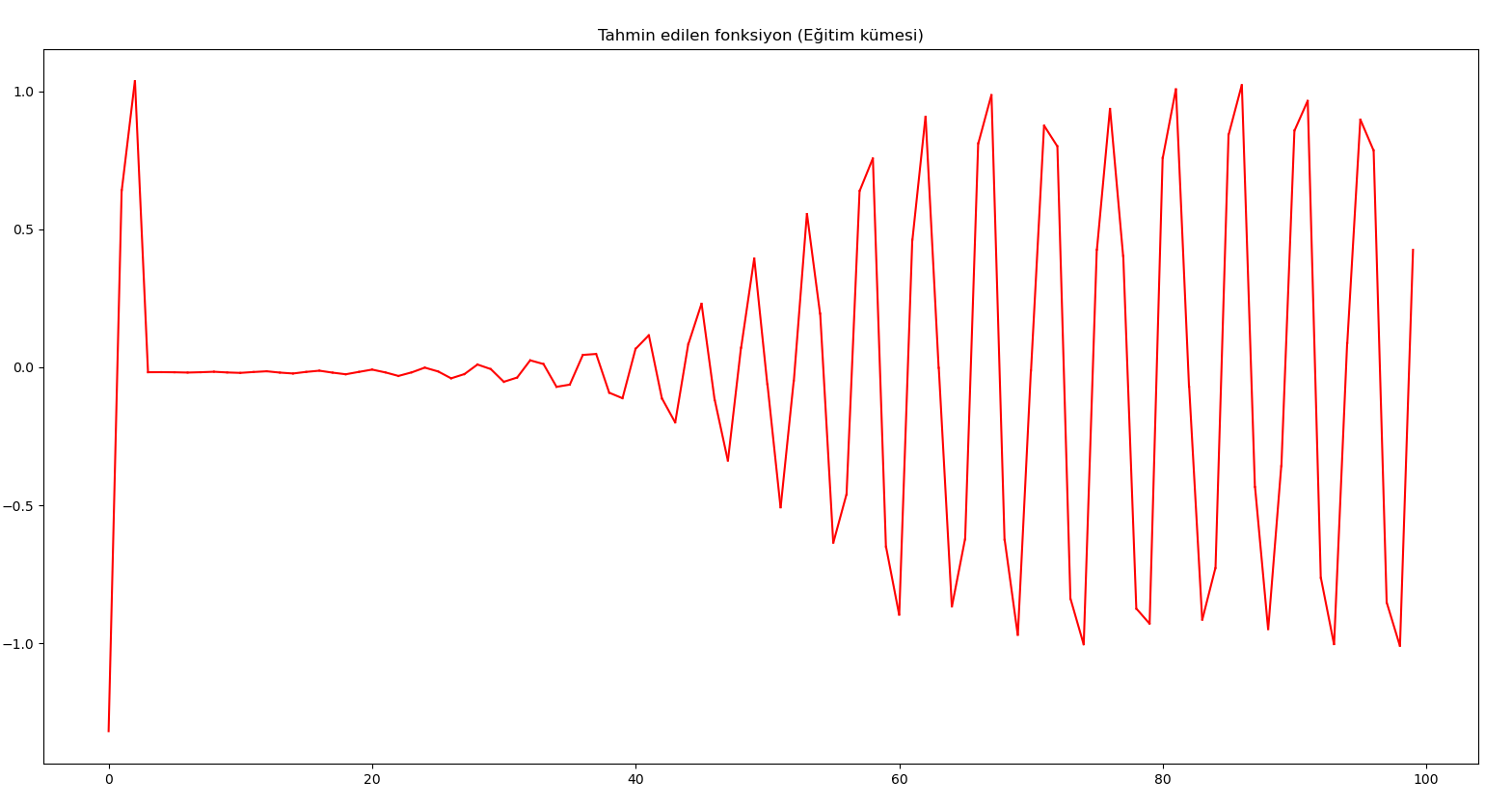
**Eğitim ve test fonksiyonları:**

Koddaki tüm fonksiyonlar, önceki sorularda kullanılan fonksiyonlarla hemen hemen aynıdır. İki farklılık testNetwork (önceki soruda testprocess) ve educationprocess fonksiyonlarındadır. Bu fonksiyonlarda veri bilgisi seçilirken

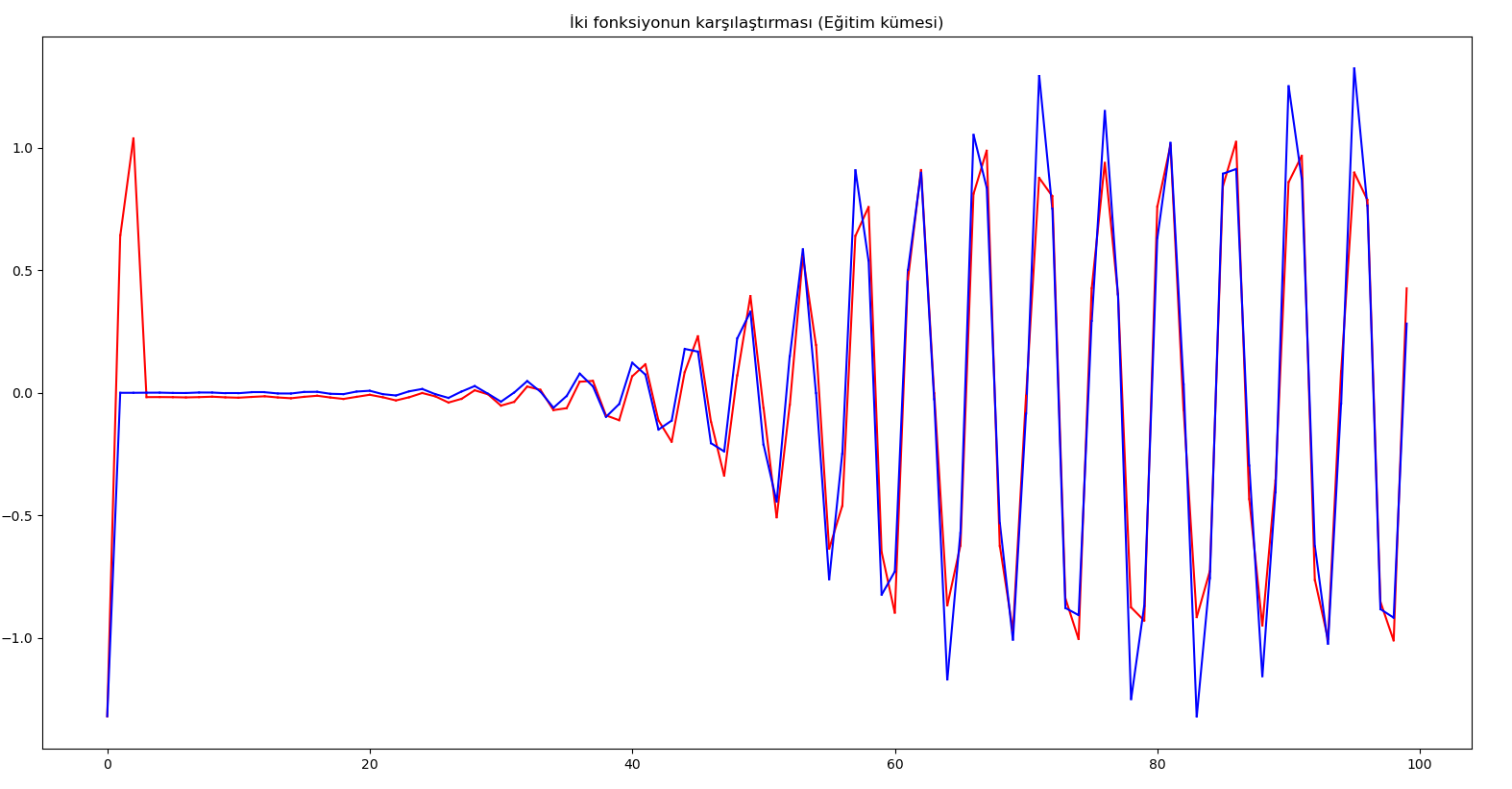
numpy arrayi oluşturulur. Bu, bağlı listeden çekilerek sağlanır. Sınıf bilgisi de aynı şekilde numpy arrayi şeklindedir. Ağın çalışması için veri yapısının numpy arrayi şeklinde olması önemlidir. 

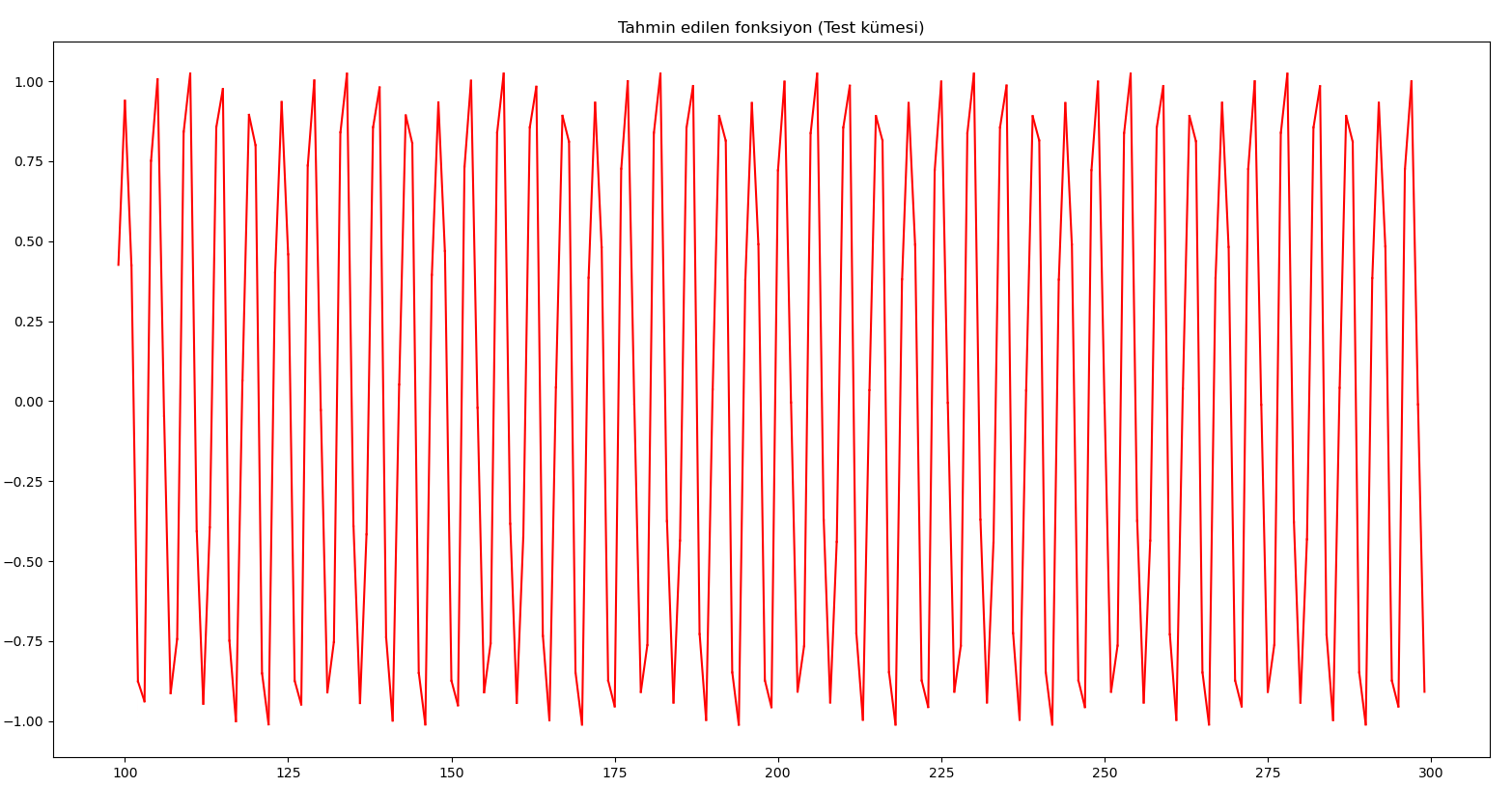
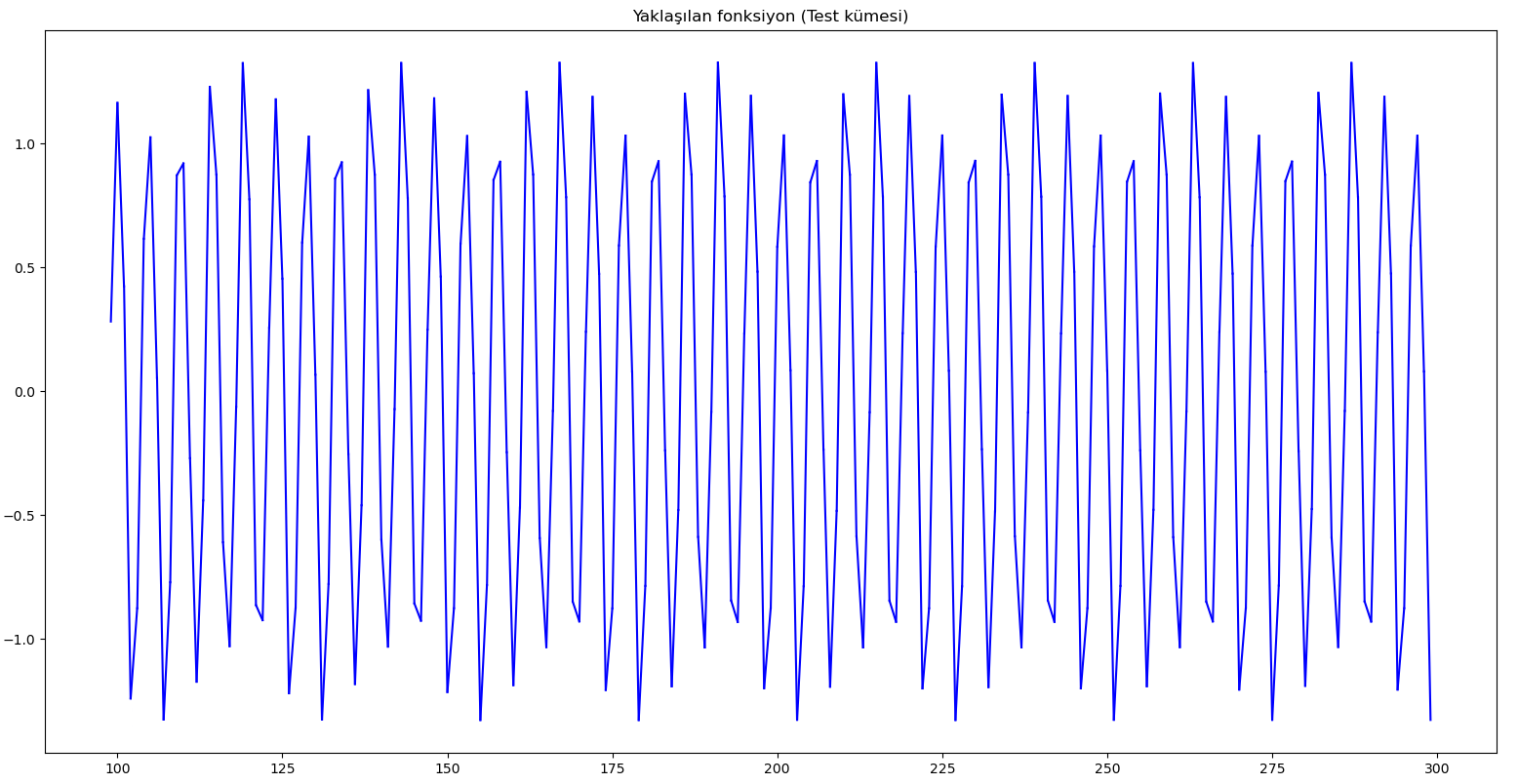
**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

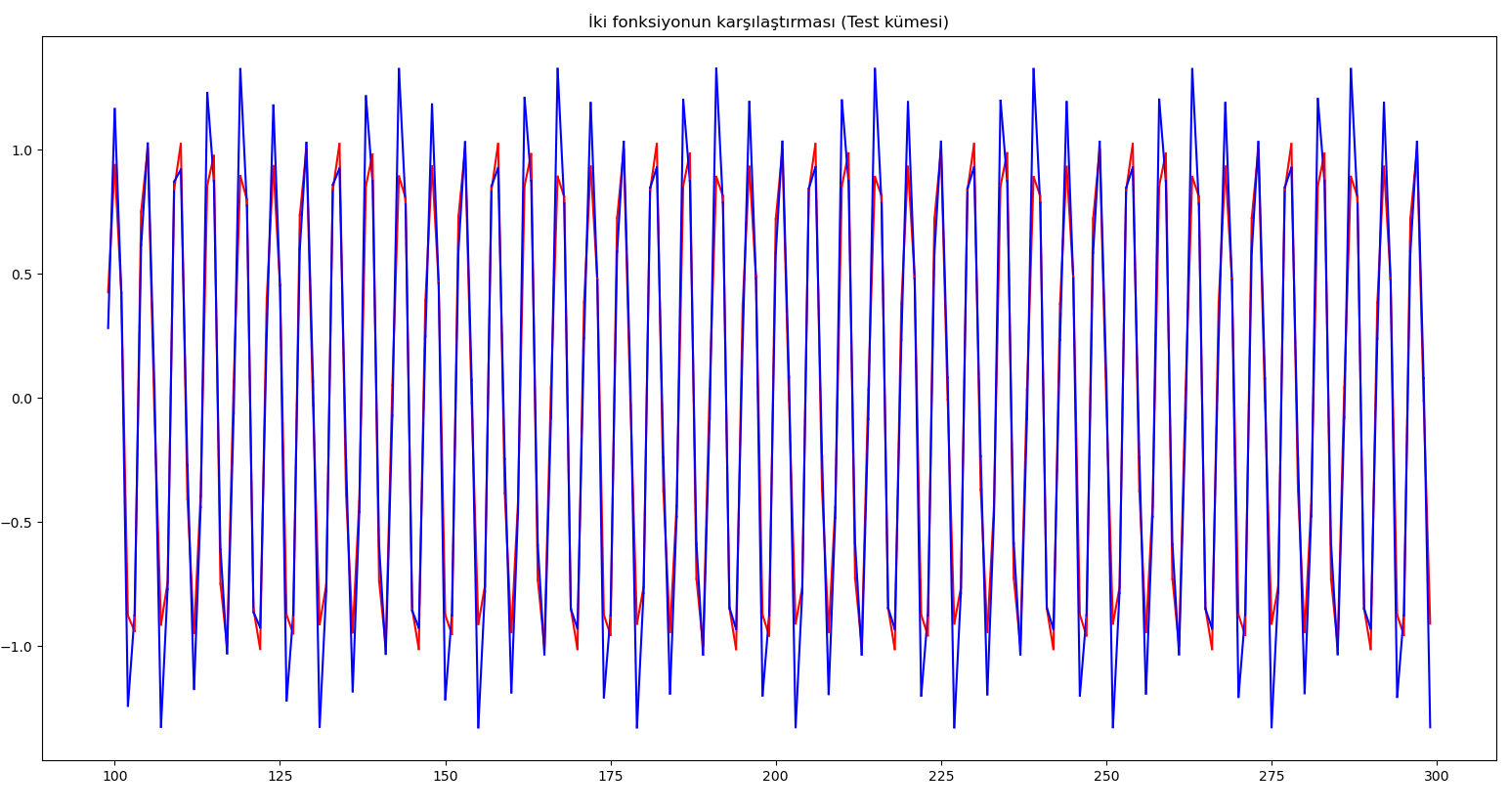
Veriler, ters ölçeklenip karşılaştırılmak üzere çizildiğinde aşağıdaki şekiller ortaya çıkar. Sigmoid fonksiyon kullanılması ve verilerin [0,1] arasına çekilmesinden dolayı, ilk değerlerde garip bir yol izlendiği görülür. Belki tanh fonksiyonunun bu problem için daha uygun olabileceği görülür. Başlangıçta bir düzensizlik de olsa sistem yaklaşık 60. adımda limit değerlerine normal genliğine ulaşmış bir şekilde salınıma girmektedir.

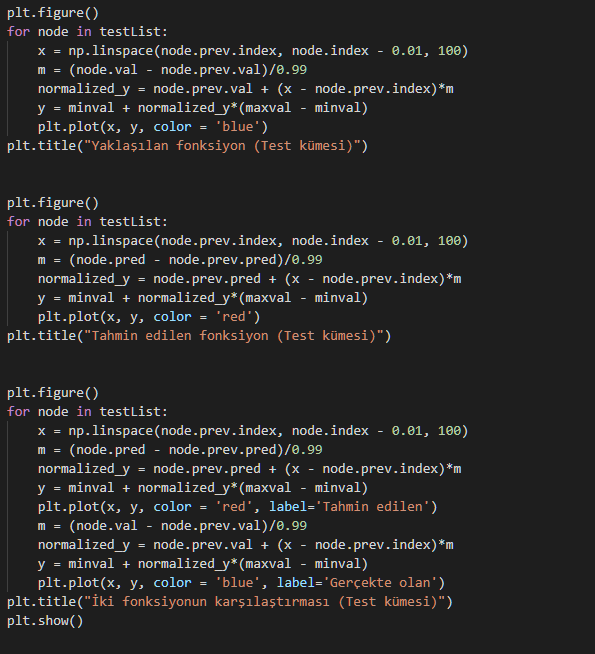
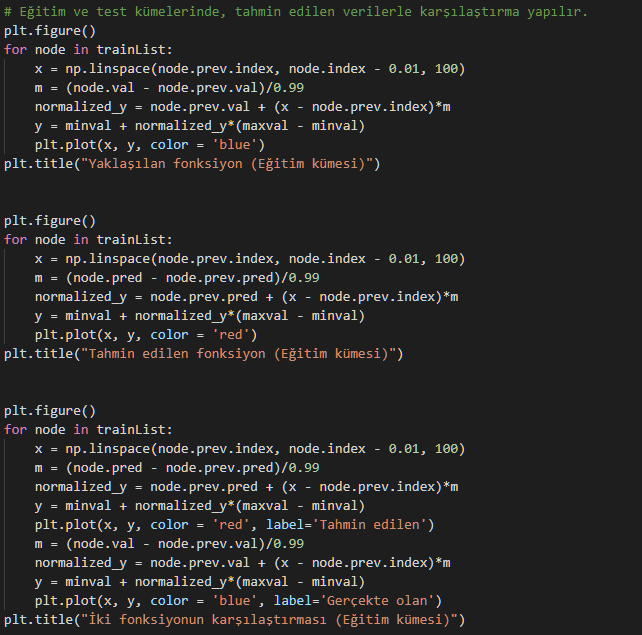


Eğitilen ağ, beklendiği gibi eğitim kümesiyle yaklaşık olarak örtüşmektedir. Eğitimin başarısıyla ilgili gerçek bilgi test kümesiyle olan karşılaştırmadan gelmektedir.



Test kümesindeki veriler başlangıç koşulundan bağımsız bir şekilde salınımdadırlar. Sistem kararlılığını sürdürmektedir, bu eğitimin başarılı olmasında büyük katkı sağlamaktadır.  

Eğitim sonrası test aşamasında bulunan yaklaşık %0.32’lik hata, eğitimin başarılı olduğu anlamına gelir. Bu, tahmin edilen ve gerçekte olan fonksiyon aynı grafikte karşılaştırıldığında daha belirgindir. Genlik olarak uç noktalarda kısmen fark göstermekte olsa da çoğunlukla yakın fonksiyonlardır.

**Çizim için kullanılan kodlar:**