# Yapay Sinir Ağları
# 2.Ödev

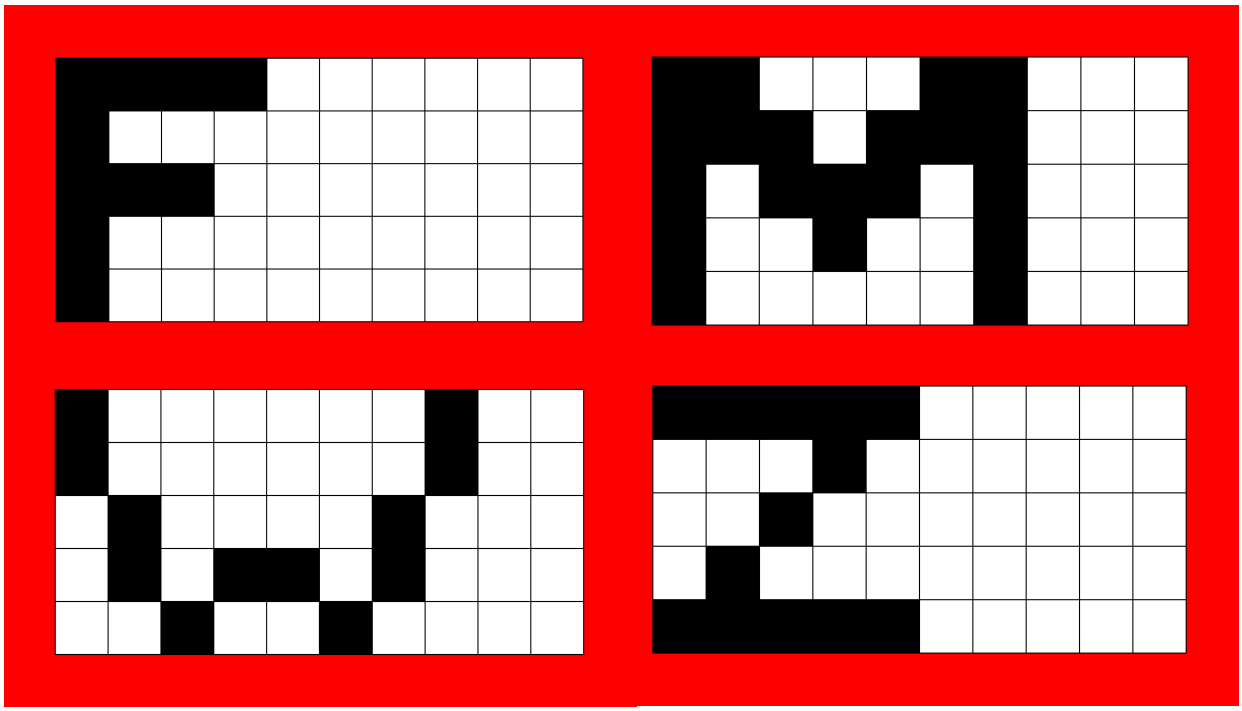
**15.12.2020**

**Mehmet Şerbetçioğlu -- 040160056**

**Ahmet Hulusi Tarhan -- 040170738**

**Soru 1: Örüntü Tanıma**

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.



```
# f, m, w ve z harflerinin array seklinde karşılığı. m yerine g harfi kullanmayı denedim fakat bozuk g ve bozuk f birbirlerine çok yakın olduğu için sorun çıkarıyorlardı.
f_standart = np.array([1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0], dtype='f')
# g_standart = np.array([0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0], dtype='f')
m_standart = np.array([1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 1, 0, 1, 1, 1, 0, 0, 0, 1, 0, 1, 1, 1, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0], dtype='f')
w_standart = np.array([1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0], dtype='f')
z_standart = np.array([1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0], dtype='f')
```

---

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.

```
fList = generatedata(centralize(f_standart), 6)
# gList = generatedata(centralize(g_standart), 6)
mList = generatedata(centralize(m_standart), 3)
wList = generatedata(centralize(w_standart), 2)
zList = generatedata(centralize(z_standart), 5)
```

## Soru 1: Örüntü Tanıma

### Veri oluşturma fonksiyonu:

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

```
# Verilen harflerden farklı gürültü ve bozukluklarla veri oluşturan fonksiyon.
def generatedata(arr, shiftcount):
    dataList = []
    # Her kaydırma için farklı veri kümesi oluşturur.
    for i in range(shiftcount+1):
        darr = arr
        for j in range(i):
            darr = shiftright(darr)
        # Kümeye 5 temiz, gürültüsüz veri ekler.
        for j in range(5):
            dataList.append(darr)
        # Kümeye 5 tane sadece gürültülü, 5 tane de hem gürültülü hem de bozuk veri ekler.
        for j in range(10):
            dataarr = darr
            dataarr = centralize(dataarr)
            dataarr = addnoise(dataarr)
            if j % 2 == 0:
                dataList.append(dataarr)

                # # Oluşturulan veriyi çizdirir.
                # plt.imshow(dataarr.reshape((5, 10)), cmap='gray_r')
                # plt.title('just dataarr')
                # plt.show()
            elif j % 2 == 1:
                corrupted = dataarr
                corruptL = corruptdata(corrupted)
                dataList.append(corruptL[0])
```

```
                # # Oluşturulan veriyi çizdirir.
                # plt.imshow(corruptL[0].reshape((5, 10)), cmap='gray_r')
                # plt.title('corrupted')
                # plt.show()

                # # Debug için yazılmış bir kod. Nedense corrupt fonksiyonu, girilen veriyi de bozuyor.
                # # Bu sebeple veriler birkaç kez farklı değişkenlerde kopyalanarak kullanıldı.
                # plt.imshow(corruptL[1].reshape((5, 10)), cmap='gray_r')
                # plt.title('corrupted but not')
                # plt.show()
    return dataList
```

### Verileri ortalama fonksiyonu:

```
# Ortalama fonksiyonu. Verileri 0 ve 1 değerlerinden 0.1 ve 0.9 değerlerine çeker.
def centralize(arr):
    arr = np.where(arr < 0.5, 0.1, 0.9)
    return arr
```

## Soru 1: Örüntü Tanıma

### Gürültü ekleme fonksiyonu:

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

```
# Gürültü fonksiyonu. Verilere 0 medyanlı ve 0.02 standart sapmalı gauss dağılımlı gürültü ekleyip
# 0.05 - 0.95 aralığının dışına çıkan verileri, o değerlerde sınırlandırır.
def addnoise(arr):
    noisyarr = arr
    for i in range(noisyarr.size):
        noisyarr[i] = noisyarr[i] + np.random.normal(0, 0.02)
    noisyarr = np.where(noisyarr > 0.95, np.round(noisyarr,0) - 0.05, noisyarr)
    noisyarr = np.where(noisyarr < 0.05, np.round(noisyarr,0) + 0.05, noisyarr)
    return noisyarr
```

### Bozma fonksiyonu:

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

```
# Bozma fonksiyonu. 50 veriden oluşan bir veri kümesinin rastgele iki farklı noktasını tersine çevirir.
def corruptdata(arr):
    corruptarr = arr
    indices = random.sample(range(0, 50), 2)

    corruptarr[indices[0]] = 1 - corruptarr[indices[0]]
    corruptarr[indices[1]] = 1 - corruptarr[indices[1]]

    arrList = [corruptarr,arr]
    return arrList
```

### Kaydırma fonksiyonu:

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

```
# Sağa kaydırma fonksiyonu. Bir arraydeki tüm değerleri bir sonraki indise kaydırır. Son değeri başa alır.
def shiftright(arr):
    shifted = np.zeros(arr.size)
    for i in range(arr.size):
        shifted[(i+1)%arr.size] = arr[i]
    return shifted
```

---

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.

**Soru 1: Örüntü Tanıma**

```
    for data in fList:···
    # for data in gList:
    #     clsdata = np.concatenate((data, [0, 1, 0, 0]), axis=0)
    #     clsdata = clsdata.reshape(54,1)
    #     trainList.append(clsdata)
    for data in mList:···
    for data in wList:···
    for data in zList:
        clsdata = np.concatenate((data, [0, 0, 0, 1]), axis=0)
        clsdata = clsdata.reshape(54,1)
        trainList.append(clsdata)
    # for data in trainList:
    #     plt.imshow(data[:dimension].reshape((5, 10)), cmap='gray_r')
    #     clscode = data[dimension:]
    #     plt.title(clscode)
    #     plt.show()
    saveData(trainList, dataFile)
```

---

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.

```
# 50 (dimension: veri boyutu = 50) girişli, ilk katmanında 20 nöron, ikinci katmanında
# 20 nöron ve çıkış katmanında 4 nöron olan bir ağ oluşturulur. Ağın öğrenme hızı 0.5,
# bias = True (Bias terimini kendisi ekleyecek) ve aktivasyon fonksiyonu
# sigmoid olacaktır. Verilerimiz 0-1 aralığında olduğundan sigmoid seçilmiştir.
network = create_2h_network(dimension, 20, 20, 4, 5e-1, True, "sigmoid")
```

## Soru 1: Örüntü Tanıma

### Yapay sinir ağı oluşturma fonksiyonu:

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

```
# İki gizli katmanlı çok katmanlı algılayıcı oluşturan fonksiyon.
def create_2h_network(data_size, l1_size, l2_size, lo_size, learning_rate, bias, function):
    l1_List = []
    l2_List = []
    lo_List = []
    # Girilen katman büyüklükleri kadar nöron oluşturup bu nöronları katman listelerine atar.
    # Birinci katmandaki nöronların veri boyutu kadar, ikinci katmandakilerin birinci katman nöron sayısı kadar ve
    # çıkış katmanındakilerin ikinci katman nöron sayısı kadar girişi olacak. Bias = True kullanacağımızdan dolayı
    # bias terimini girmemize gerek olmayacak. Nöronların öğrenme hızı, bias ve fonksiyonları aynı olacak.
    for i in range(l1_size):
        l1_List.append(Perceptron(data_size, learning_rate, bias, function))
    for i in range(l2_size):
        l2_List.append(Perceptron(l1_size, learning_rate, bias, function))
    for i in range(lo_size):
        lo_List.append(Perceptron(l2_size, learning_rate, bias, function))
    # Tüm layerları Layers arrayine yerleştirir ve bu arrayi döndürür.
    Layers = np.array([l1_List, l2_List, lo_List])
    return Layers
```

### 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.

```
# Halil hocanın kodundan faydalanıp, ek olarak önceki ağırlık bilgisi, momentum terimi bulunan bir nöron classı.
class Perceptron():
    def __init__(self, dimensions, learning_rate = 1e-2, bias = True, activation_type = "tanh"):
        #Initialize
        self.dimensions = dimensions
        self.learning_rate = learning_rate
        self.bias = bias
        self.activation_type = activation_type

        self.reset()
        # print(self.__str__())
```

```
def reset(self):
    #Reset perceptron variables
    if self.bias is True:
        self.weights = np.zeros((1, self.dimensions + 1), dtype=float)
        self.weights[0,:] = np.random.rand() - 0.5
        self.prevweights = np.zeros((1, self.dimensions + 1), dtype=float)
    else:
        self.weights = np.zeros((1, self.dimensions), dtype=float)
        self.weights[0,:] = np.random.rand() - 0.5
        self.prevweights = np.zeros((1, self.dimensions), dtype=float)
    self.data = np.zeros(self.dimensions)
    self.lin_comb = 0
    self.activation = 0
```

## Soru 1: Örüntü Tanıma

```
def forward(self, data):
    #Forward pass data through the perceptron
    if self.bias is True:
        self.data = np.concatenate((data, np.array([1],dtype=float).reshape(1,1)), axis = 0)
    else:
        self.data = np.array(data)
    w = self.weights
    self.lin_comb = np.dot(w, self.data)
    self.activation = self.activation_function(self.lin_comb)
    return self.activation
```

```
def activation_function(self, data):
    #Select activation function and use it
    if self.activation_type == "tanh":
        y = tanh(data)
        self.activation_derivative = y[1]
        return y[0]
    elif self.activation_type == "sigmoid":
        y = sigmoid(data)
        self.activation_derivative = y[1]
        return y[0]
    else:
        return data
```

```
def update(self, grad, moment = True):
    # Momentum terimi eklenmiş ağırlık güncelleme fonksiyonu. momentum = False yapılırsa momentum terimi 0 alınır, aksi takdirde
    # momentum, n öğrenme hızı için momentum = n + (1 - n)/5 olacak.
    momentum = (1 - self.learning_rate)/5 + self.learning_rate
    if  moment == False:
        momentum = 0
    momentumterm = momentum*np.subtract(self.weights, self.prevweights)
    self.prevweights = self.weights
    # print(self.weights.shape)
    self.weights = np.add(np.add(self.weights,np.multiply((self.learning_rate*grad[0]),self.data).reshape(1, self.weights.size)), momentumterm)
    # print(self.weights.shape)
```

## Aktivasyon fonksiyonları:

```
# tanh ve sigmoid fonksiyonları. Türev bilgilerini de içermekte.
def tanh(x):
    t=(np.exp(x)-np.exp(-x))/(np.exp(x)+np.exp(-x))
    dt=1-t**2
    return t,dt

def sigmoid(x):
    s=1/(1+np.exp(-x))
    ds=s*(1-s)
    return s,ds
```

---

## Soru 1: Örüntü Tanıma

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

```
# Maksimum iterasyon sayısı 500 ve hata sınırı 0.002 seçilmiştir.
epoch = 500
eth = 2e-3

# Eğitim yapılır ve eğitim süresi, iterasyon sayısı ve son hata bilgileri elde edilir.
start_time = time.time()
result = educationprocess(trainList, network, epoch, eth, dimension)
print("Education time: %f" % (time.time() - start_time))
print("Training is over in %d steps with %.7f error" % (result[0], result[1]))
```

### Eğitim işlemi fonksiyonu:

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

```
# Eğitim aşamasının gerçekleştiği fonksiyon
def educationprocess(datalist, network, epoch, errorthreshold, dim, moment = True):
    Eortprev = 0
    ResetCount = 0
    result = np.zeros(2)
    result[0] = epoch
    # Hata, istenilen seviyeye inmedikçe, epoch ile belirlenen maksimum iterasyon sayısı kadar eğitim devam eder.
    for i in range(epoch):
        # Öncelikle eğitim kümesi karıştırılır.
        random.shuffle(datalist)
        Eort = 0
        # Verinin, dimension değişkenine kadar kısmı veriyi, geri kalanı sınıf bilgisini belirtir. Bu iki bilgi
        # ayrılıp, eğitim fonksiyonuna girilir.
        for data in datalist:
            imdata = data[:dimension]
            clsdata = data[dimension:]
            E = educate(imdata, network, clsdata, moment)
            Eort += E
        # Hata her veri için eklenir ve eğitim sonunda veri sayısına bölünür.
        Eort = Eort/len(datalist)
```

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.

```
        # Hatadaki değişim, hata limitiyle önceki hatanın çarpmının altındaysa iki şey yaşanabilir.

        # Eğer bu gerçekleşirken hata, hata limitinin yüz katından fazlaysa ağın ağırlıkları sıfırlanır ve rastgele değerler alır.
        # Bu sayede lokal minimumun çok yüksek hataya sahip olduğu durumlardan kurtulunabilinir. Bunun yaşanması için iki ardışık hata
        # arasındaki farkın elli kere sınırın altında olması gerekir.
        if abs(Eortprev - Eort) < Eortprev*errorthreshold and Eort > errorthreshold*100:
            ResetCount += 1
            if ResetCount >= 50:
                for perceptron in network[0]:
                    perceptron.reset()
                for perceptron in network[1]:
                    perceptron.reset()
                for perceptron in network[2]:
                    perceptron.reset()
                ResetCount = 0
        # print("%d iteration, %.7f error" % (i + 1, Eort[0][0]))

        # Eğer hata, limitin altında veya limitin yüz katının altında ve iki hata arasındaki fark çok küçükse, eğitim sonlanır.
        if (abs(Eortprev - Eort) < Eortprev*errorthreshold and Eort < errorthreshold*100) or Eort < errorthreshold:
            result[0] = i + 1
            break
        Eortprev = Eort
    # Eğitim sonucundaki hata ve eğitimin iterasyon sayısı elde edilir.
    result[1] = Eort
    return result
```

## Soru 1: Örüntü Tanıma

### 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.

```
# Eğitim fonksiyonu.
def educate(data, network, clsdata, moment = True):
    y1 = np.zeros((len(network[0]), 1), dtype=float)
    y2 = np.zeros((len(network[1]), 1), dtype=float)
    yo = np.zeros((len(network[2]), 1), dtype=float)
    i1 = 0
    i2 = 0
    io = 0
    # İlk katmandan başlayarak ileri yol algoritması gerçekleştirilir. Sonraki katmanın girişi önceki
    # katmanın çıkışı olacaktır.
    for perceptron in network[0]:
        y1[i1] = perceptron.forward(data)
        i1 += 1
    for perceptron in network[1]:
        y2[i2] = perceptron.forward(y1)
        i2 += 1
    for perceptron in network[2]:
        yo[io] = perceptron.forward(y2)
        io += 1
    # Hata hesaplanır.
    e = np.subtract(clsdata, yo)
    E = np.dot(np.transpose(e), e)/2
```

**Soru 1: Örüntü Tanıma**

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

```
grad1 = np.zeros((len(network[0]), 1), dtype=float)
grad2 = np.zeros((len(network[1]), 1), dtype=float)
grado = np.zeros((len(network[2]), 1), dtype=float)

# Son katmandan başlayarak, her katmanın lokal gradyanları hesaplanır.
grado = local_grad(network[2], e, outputLayer = True)
grad2 = local_grad(network[1], network[2], nextgrad = grado)
grad1 = local_grad(network[0], network[1], nextgrad = grad2)

i1 = 0
i2 = 0
io = 0
# Hesaplanan lokal gradyanlar ile her katmanın ağırlıkları güncellenir.
for perceptron in network[0]:
    perceptron.update(grad1[i1], moment)
    i1 += 1
for perceptron in network[1]:
    perceptron.update(grad2[i2], moment)
    i2 += 1
for perceptron in network[2]:
    perceptron.update(grado[io], moment)
    io += 1
# Elde edilen hata (eT*e)/2 dönüdürülür.
return E
```

## Soru 1: Örüntü Tanıma

### Lokal gradyan fonksiyonu:

```
# Lokal gradyan hesaplama fonksiyonu.
def local_grad(list1, list2, nextgrad = None, outputLayer = False):
    ncount = len(list1)
    act_derivative = np.zeros((ncount, 1), dtype=float)
    grad = np.zeros((ncount, 1), dtype=float)
    # Output layer için güncelleme yapılıyorsa sonraki layerın girişleri yerine hata vektörüyle
    # çarpım yapılacağı için çıkış katmanı için ayrı bir koşul kullanıldı.
    if outputLayer == True:
        for i in range(ncount):
            act_derivative[i] = list1[i].activation_derivative
        grad = np.multiply(list2, act_derivative)
    # Bias teriminin çıkarıldığı bir ağırlık matrisi elde edilir. Lokal gradyanlar
    # backpropagation algoritmasına göre hesaplanıp verilir.
    else:
        wrow = (list2[0].weights.size) - 1
        outcount = len(list2)
        wmat = np.zeros((wrow, outcount), dtype=float)
        for i in range(ncount):
            act_derivative[i] = list1[i].activation_derivative
        for i in range(outcount):
            wmat[:, i] = list2[i].weights[0, 0:wrow]
        grad = np.multiply(np.dot(wmat, nextgrad), act_derivative)
    return grad
```

### Eğitim sonucu:

```
Education time: 38.292275
Training is over in 70 steps with 0.0016870 error
```

---

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

```
# Test gerçekleşir. Yanlış tahmin miktarı elde edilir.
testresult = testprocess(testList, network, dimension)
print("Made %d wrong predictions out of %d patterns" % (testresult[0], testresult[1]))
```

## Soru 1: Örüntü Tanıma

### Test işlemi fonksiyonu:

Test kümesindeki verileri ağdan geçirip ağ sonucunu sınıf bilgisiyle karşılaştırır.

```
# Verilerin test aşamasının yapıldığı fonksiyon.
def testprocess(datalist, network, dim):
    Errorcount = 0
    totalcount = 0
    # Test öncesi veri listesi karıştırılır.
    random.shuffle(datalist)
    for data in datalist:
        # Her veri için testdata fonksiyonu çalıştırılır ve ağ çıkışı pred değişkenine atılır.
        pred = testdata(data[:dim], network)
        # Ağ çıkışı, maksimum değere bölünür ve tüm değerler 1 ile karşılaştırılır. Bu şekilde maksimum değer,
        # indisiyle birlikte bulunur. Maksimum değerin indisi tahmin edilen örüntüyü verir.
        pred = pred/np.amax(pred, axis = 0)
        for i in range(pred.size):
            if pred[i] == float(1):
                ind = i
        if ind == 0:
            predim = "f"
        elif ind == 1:
            # predim = "g"
            predim = "m"
        elif ind == 2:
            predim = "w"
        elif ind == 3:
            predim = "z"
        else:
            predim = "None"
```

```
        # Hatalı tahmin sayısı Errorcount değişkeni ile kaydedilir.
        if data[dim + ind] != pred[ind]:
            Errorcount += 1
        totalcount += 1
        # # Görüntü, ağın tahmini ile gösterilir.
        # plt.imshow(data[0:dimension].reshape((5, 10)), cmap='gray_r')
        # plt.title(predim)
        # plt.show()
    # Fonksiyon sonucunda test kümesinin büyüklüğü ve hatalı tahmin sayısı döndürülür.
    result = np.array([Errorcount, totalcount])
    return result
```

**Soru 1: Örüntü Tanıma**

**Test fonksiyonu:**

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

```
# Verinin test edildiği fonksiyon. Sadece ileri yol algoritmasını takip eder ve çıkışı verir.
def testdata(testdata, network):
    y1 = np.zeros((len(network[0]), 1), dtype=float)
    y2 = np.zeros((len(network[1]), 1), dtype=float)
    yo = np.zeros((len(network[2]), 1), dtype=float)
    i1 = 0
    i2 = 0
    io = 0
    for perceptron in network[0]:
        y1[i1] = perceptron.forward(testdata)
        i1 += 1
    for perceptron in network[1]:
        y2[i2] = perceptron.forward(y1)
        i2 += 1
    for perceptron in network[2]:
        yo[io] = perceptron.forward(y2)
        io += 1
    return yo
```

**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.

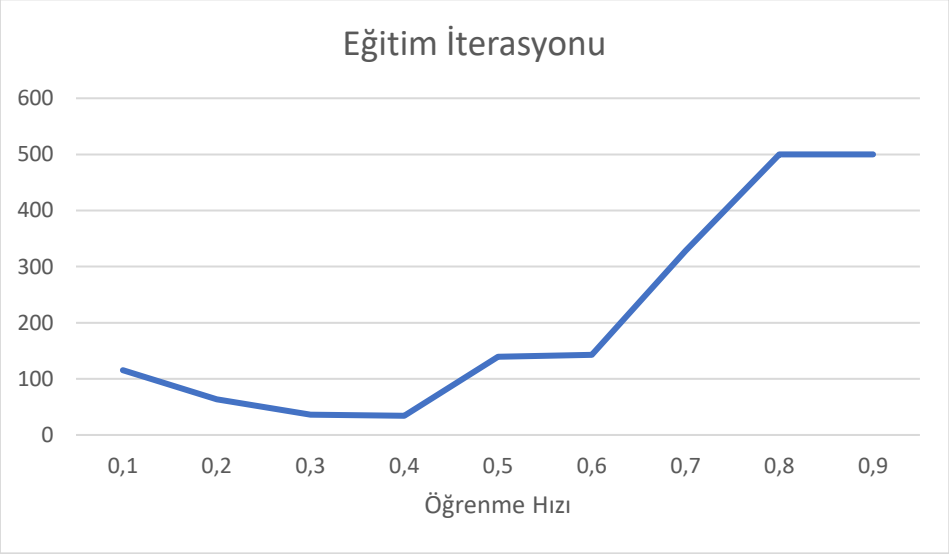
```
Made 1 wrong predictions out of 300 patterns
```

---

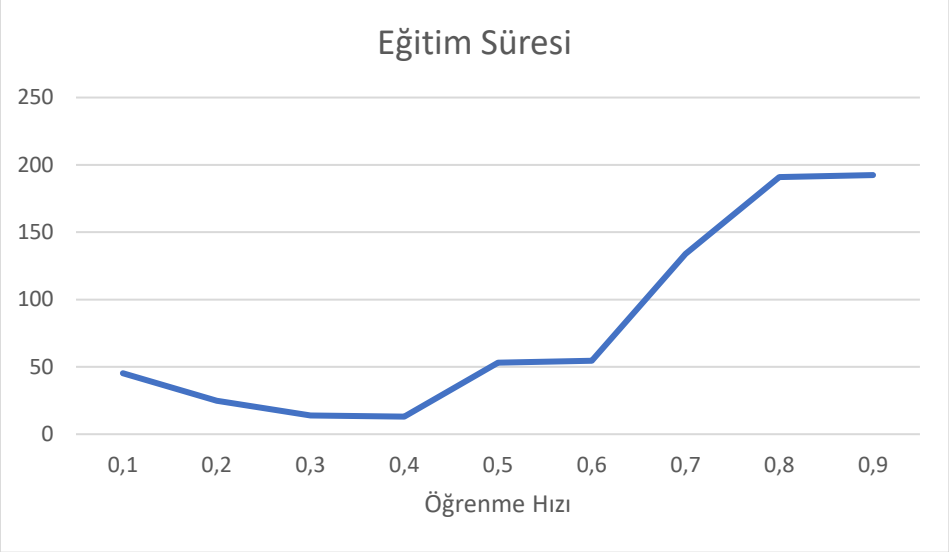
**Soru 1: Örüntü Tanıma**

**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.

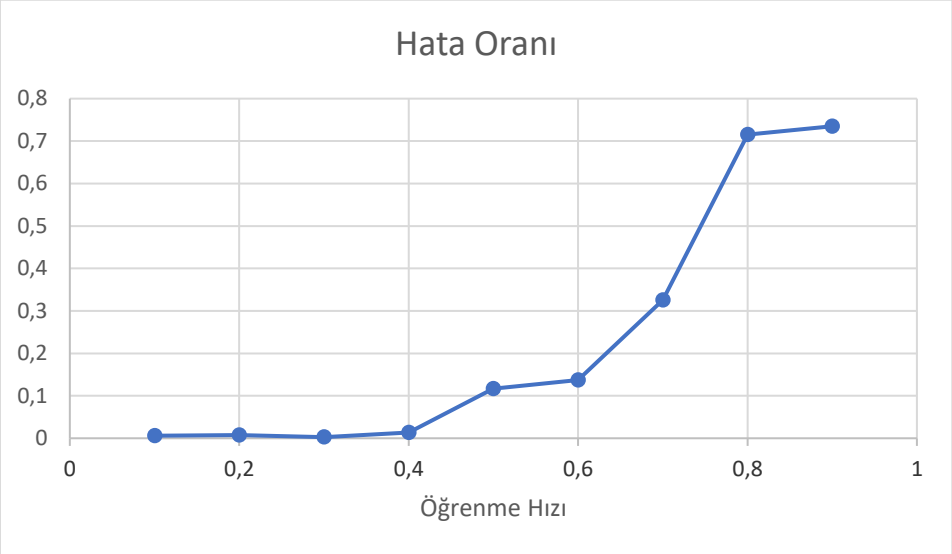



Eğitim süresi de, ağ yapısı değişmediği için eğitim iterasyonuyla aynı orantıda ilerlemektedir. 0.8 ve 0.9 noktalarında eğitim maksimum iterasyona ulaştığı için, maksimum eğitim süresinin olarak yaklaşık 192 saniye sürdüğü görülür.



**Soru 1: Örüntü Tanıma**

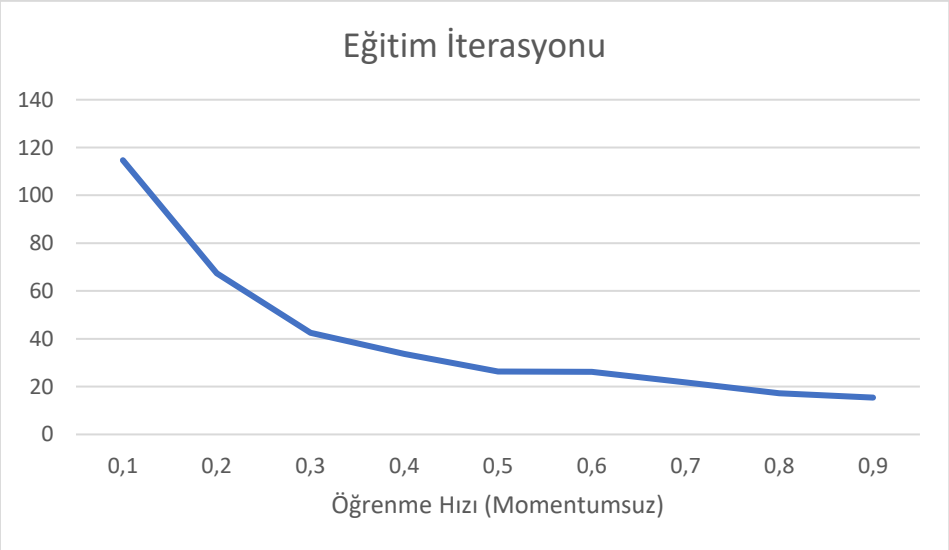
Hata oranı 0.3'ten itibaren sonra artış göstermektedir. 0.3 noktasına kadar hata oranı %1 değerinin altındayken bu noktadan sonra %73'e kadar artmaya devam eder. 0.8 ve 0.9 noktalarındaki yüksek hata eğitimin tamamlanamamasından kaynaklanmaktadır fakat eğitim tamamlansa dahi 0.5-0.7 aralığındaki hızlarda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




---

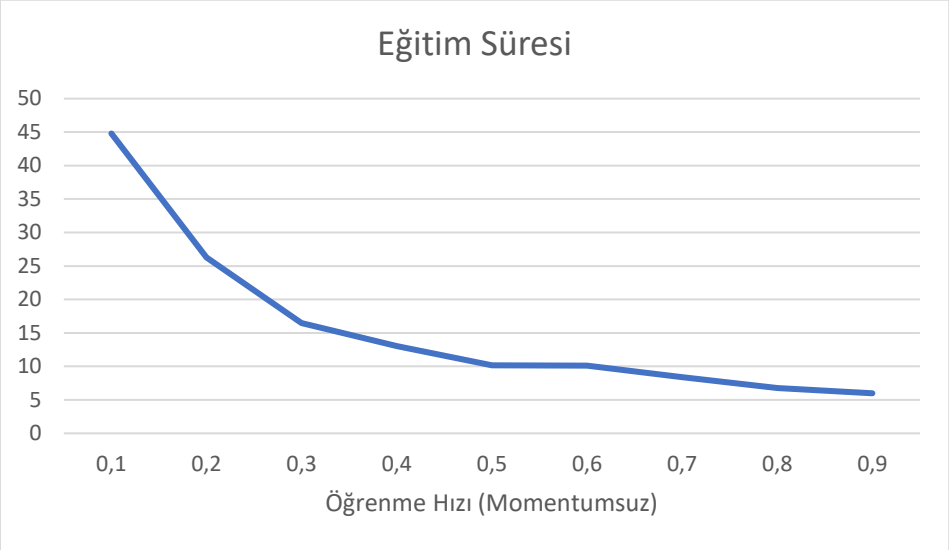
**Momentum yokken değişken öğrenme hızı:**

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.

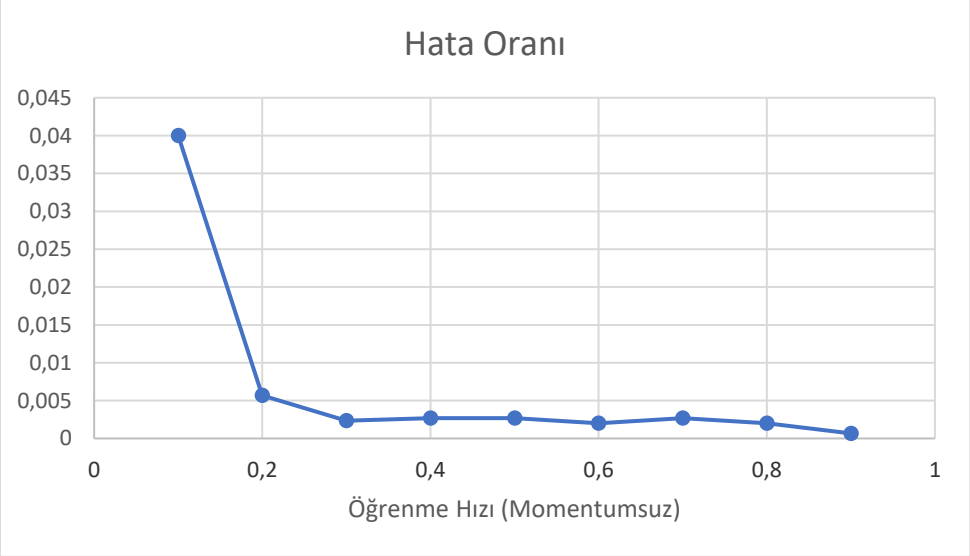


**Soru 1: Örüntü Tanıma**

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.



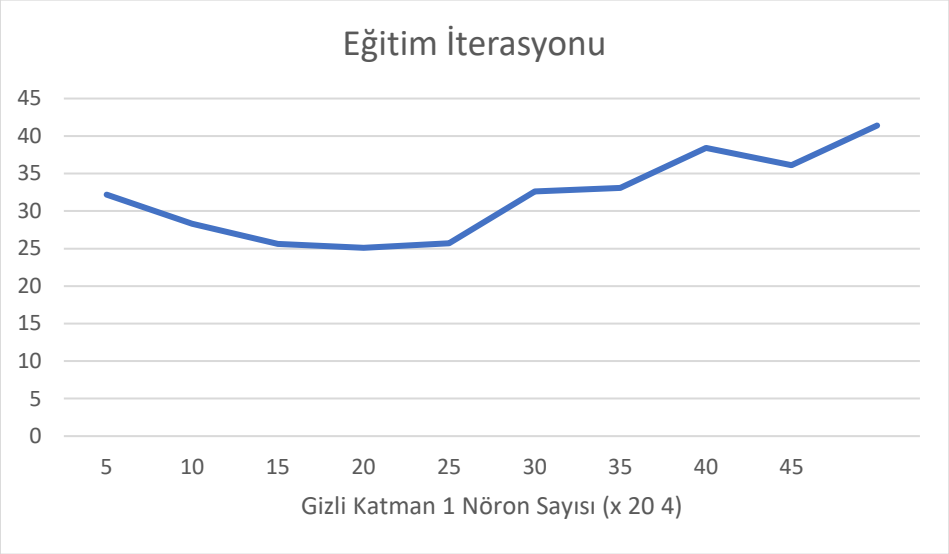

Momentum teriminin eğitim süresinin kısaltarak hata oranını düşürmesi beklenmektedir. Elde edilen sonuçlar momentumun eksik bir şekilde uygulandığına işaret eder.

**Soru 1: Örüntü Tanıma**

---

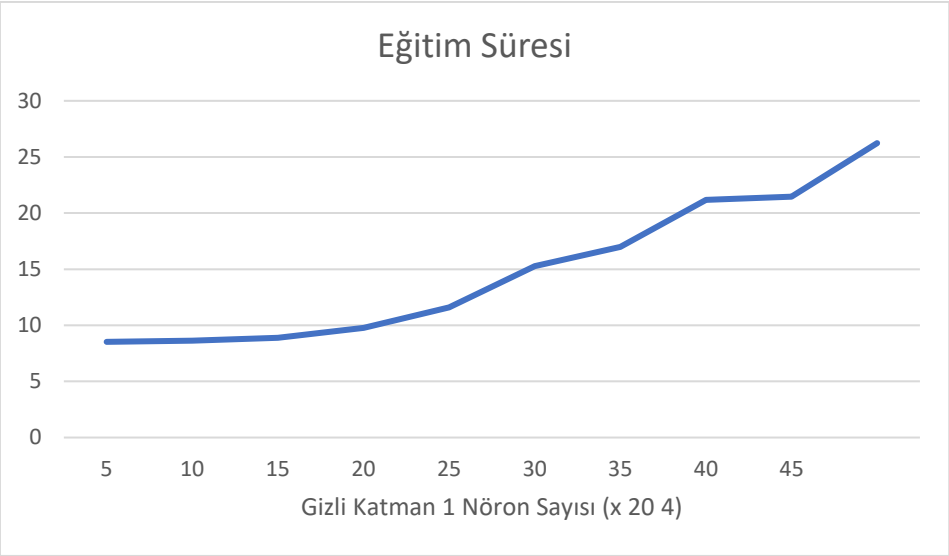
**Momentum yokken değişken birinci katman nöron sayısı:**

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.

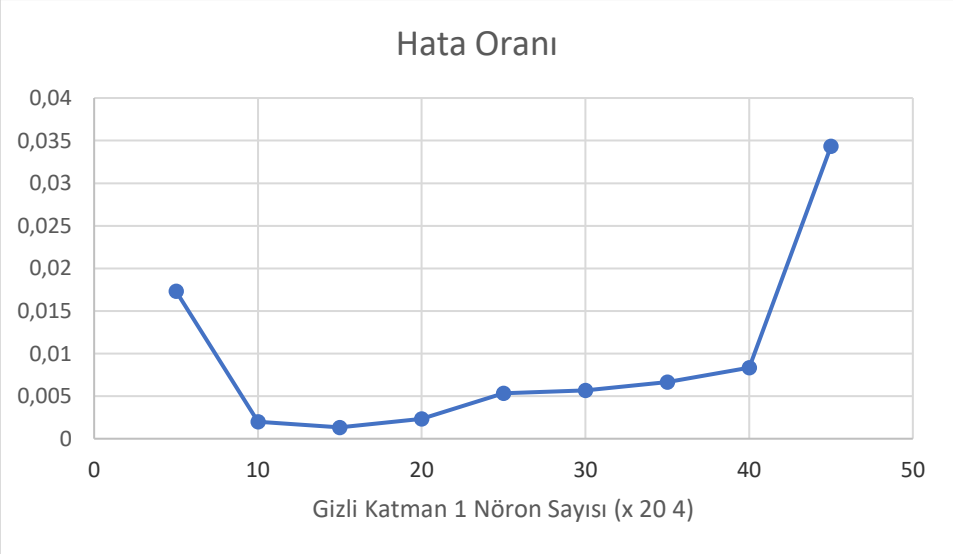



Eğitim süresi ise iterasyonun aksine sürekli artan bir yol izlemektedir. İterasyon sayısı tüm değerlerde birbirine yakın olsa da, nöron sayısının değişmesiyle güncellenmesi gereken ağırlık sayısı değiştiği için, her iterasyonun süresi değişir. Bu sebeple nöron sayısı arttıkça eğitim süresi de artmaktadır.

**Soru 1: Örüntü Tanıma**




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.

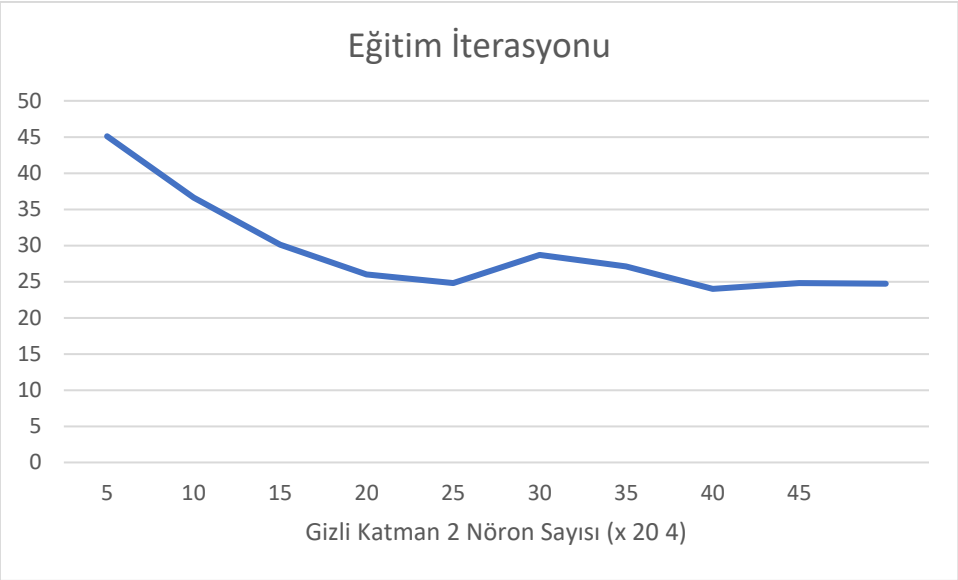



---

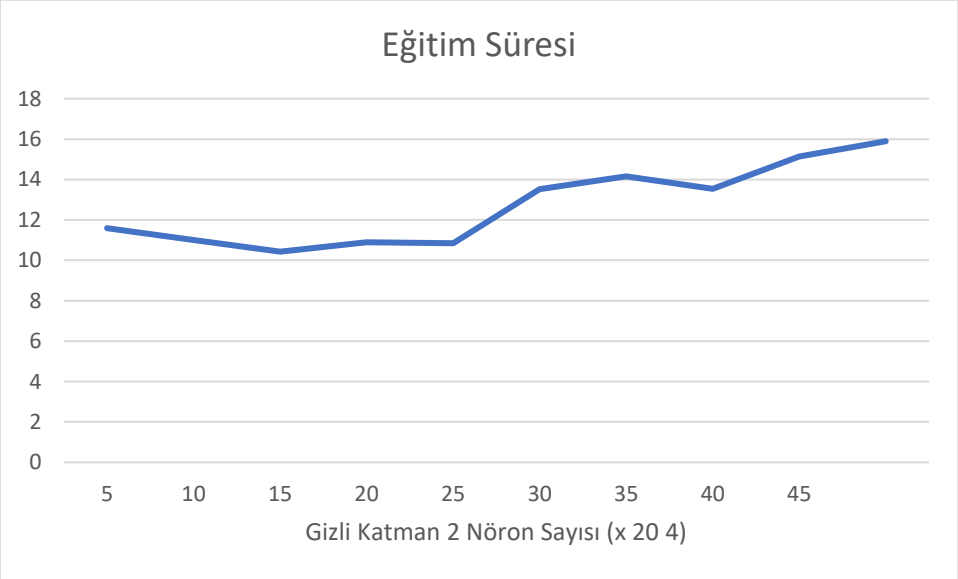
**Momentum yokken değişken ikinci katman nöron sayısı:**

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.

**Soru 1: Örüntü Tanıma**

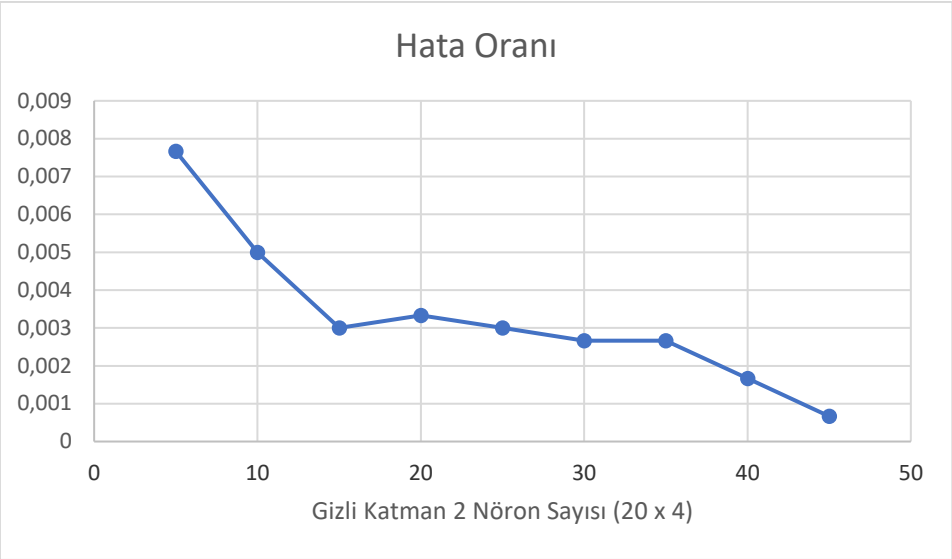



İterasyondaki bu azalmaya rağmen nöron sayısı ve beraberinde ağırlık sayısının artışı daha hızlıdır. Eğitim süresi, önceki deneye nispeten daha yavaş da olsa, genel anlamda sürekli olarak artmaktadır.




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.

## Soru 1: Örüntü Tanıma




---

## Deney Kodları:

### Momentum varken değişken öğrenme hızı:

```
# Öğrenme hızı 0.1-0.9 aralığında test yapılıp iterasyon sayısı, eğitim süresi ve hata oranı gibi bilgiler elde edilir.
# Bu testler, her öğrenme hızı için 10 kere tekrarlanıp elde edilen verilerin ortalaması alınarak excele kaydedilir.
cresult = np.zeros((9,3), dtype=float)
for i in range(9):
    eduiter = 0
    testacc = 0
    edutime = 0
    for j in range(10):
        network = create_2h_network(dimension, 20, 20, 4, (i+1)*1e-1, True, "sigmoid")
        start_time = time.time()
        result = educationprocess(trainList, network, epoch, eth, dimension)
        edutime += time.time() - start_time
        print("Education time for cresult, %d iteration: %f" % (i+1, (time.time() - start_time)))
        eduiter += result[0]
        testresult = testprocess(testList, network, dimension)
        print("Made %d wrong predictions out of %d patterns" % (testresult[0], testresult[1]))
        testacc += testresult[0]/testresult[1]
    cresult[i,0] = eduiter/10
    cresult[i,1] = edutime/10
    cresult[i,2] = testacc/10
```

## Soru 1: Örüntü Tanıma

### Momentum yokken değişken öğrenme hızı:

```
# Az önceki deneyin aynısı momentum terimi olmadan tekrarlanır.
cwomresult = np.zeros((9,3), dtype=float)
for i in range(9):
    eduiter = 0
    testacc = 0
    edutime = 0
    for j in range(10):
        network = create_2h_network(dimension, 20, 20, 4, (i+1)*1e-1, True, "sigmoid")
        start_time = time.time()
        result = educationprocess(trainList, network, epoch, eth, dimension, False)
        edutime += time.time() - start_time
        print("Education time for cwomresult, %d iteration: %f" % (i+1, (time.time() - start_time)))
        eduiter += result[0]
        testresult = testprocess(testList, network, dimension)
        print("Made %d wrong predictions out of %d patterns" % (testresult[0], testresult[1]))
        testacc += testresult[0]/testresult[1]
    cwomresult[i,0] = eduiter/10
    cwomresult[i,1] = edutime/10
    cwomresult[i,2] = testacc/10
```

### Momentum yokken değişken birinci katman nöron sayısı:

```
# Birinci katmandaki nöron sayısı 5-50 aralığında 5'in katlarıyken sonuçlar elde edilir. Momentum terimi
# yine çıkarılmıştır.
h1ncountresult = np.zeros((10,3), dtype=float)
for i in range(10):
    eduiter = 0
    testacc = 0
    edutime = 0
    for j in range(10):
        network = create_2h_network(dimension, (i+1)*5, 20, 4, 5e-1, True, "sigmoid")
        start_time = time.time()
        result = educationprocess(trainList, network, epoch, eth, dimension, False)
        edutime += time.time() - start_time
        print("Education time for h1ncountresult, %d iteration: %f" % (i+1, (time.time() - start_time)))
        eduiter += result[0]
        testresult = testprocess(testList, network, dimension)
        print("Made %d wrong predictions out of %d patterns" % (testresult[0], testresult[1]))
        testacc += testresult[0]/testresult[1]
    h1ncountresult[i,0] = eduiter/10
    h1ncountresult[i,1] = edutime/10
    h1ncountresult[i,2] = testacc/10
```

## Soru 1: Örüntü Tanıma

### Momentum yokken değişken ikinci katman nöron sayısı:

```
# Az önceki deney, birinci katman yerine ikinci katmanda tekrarlanır.
h2ncountresult = np.zeros((10,3), dtype=float)
for i in range(10):
    eduiter = 0
    testacc = 0
    edutime = 0
    for j in range(10):
        network4 = create_2h_network(dimension, 20, (i+1)*5, 4, 5e-1, True, "sigmoid")
        start_time = time.time()
        result = educationprocess(trainList, network4, epoch, eth, dimension, False)
        edutime += time.time() - start_time
        print("Education time for h2ncountresult, %d iteration: %f" % (i+1, (time.time() - start_time)))
        eduiter += result[0]
        testresult = testprocess(testList, network4, dimension)
        print("Made %d wrong predictions out of %d patterns" % (testresult[0], testresult[1]))
        testacc += testresult[0]/testresult[1]
    h2ncountresult[i,0] = eduiter/10
    h2ncountresult[i,1] = edutime/10
    h2ncountresult[i,2] = testacc/10

print("h2ncountresult is done")
```

---

## Soru 1: İris Çiçeklerinin Tanınması

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.

```
__location__ = os.path.realpath(os.path.join(os.getcwd(), os.path.dirname(__file__)))
dataFile = os.path.join(__location__, 'iris.data')
dimension = 4
# Koddaki fonksiyonlar önceki soruyla birebir aynıdır.

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

# iris.data dosyası okunup satır satır ayrılır. Bu satırlarda virgülle ayrılmış 4 veri ve çiçeğin sınıfı bulunmaktadır.
for line in Lines:
    rawdata = line.split(',')
    if len(rawdata) < 5:
        break

    # Çiçeğin sınıfına göre 1 0 0, 0 1 0 gibi arrayler oluşturulur.
    if rawdata[4].rstrip("\n") == "Iris-setosa":
        clsdata = np.array([1, 0, 0], dtype=float)
    elif rawdata[4].rstrip("\n") == "Iris-versicolor":
        clsdata = np.array([0, 1, 0], dtype=float)
    elif rawdata[4].rstrip("\n") == "Iris-virginica":
        clsdata = np.array([0, 0, 1], dtype=float)
    else:
        continue
```

```
    # Çiçek verileri ve sınıf arrayi birleştirilip veri elde edilir.
    dataArr = np.array([rawdata[0], rawdata[1], rawdata[2], rawdata[3]], dtype=float)
    clsdata = np.concatenate((dataArr, clsdata), axis=0).reshape(dimension+3,1)

    # Bu veriler irisdata listesinde toplanır.
    if 'irisdata' in globals():
        irisdata.append(clsdata)
        # print("new data", dataArr)
    else:
        irisdata = []
        irisdata.append(clsdata)
        # print("first data", dataArr)
```

---

**Soru 1: İris Çiçeklerinin Tanınması**

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

```
# Verilerin minimum ve maksimum değerleri kontrol edilir.
maxele = 0
minele = 0
i = 0
rowcount = irisdata[0].shape[0]
colcount = len(irisdata)
for i in range(colcount*dimension):
    rowindex = i % dimension
    colindex = math.floor(i/dimension)
    ele = irisdata[colindex][rowindex]
    if ele > maxele:
        maxele = ele
    if ele < minele:
        minele = ele

print("max", maxele)
print("min", minele)
```

```
# Kontrol sonucunda minimum değerin 0 olduğu görülür. Ölçeklemek için veriler maksimum değere bölünür.
for i in range(colcount*dimension):
    rowindex = i % dimension
    colindex = math.floor(i/dimension)
    irisdata[colindex][rowindex] = irisdata[colindex][rowindex]/maxele
```

Ölçeklenen veriler test ve eğitim kümelerine dağıtılır.

```
# Verilerin yarısı test, yarısı eğitim listelerine dağıtılır.
trainList = []
testList = []
for i in range(colcount):
    if i%2 == 0:
        trainList.append(irisdata[i])
    elif i%2 == 1:
        testList.append(irisdata[i])
```

---

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.

```
# İlk katmanda 20, ikincide 20 ve çıkış katmanında 3 nöron bulunan, 0.1 öğrenme hızına sahip, bias terimini kendisi
# ekleyen ve sigmoid aktivasyon fonksiyonuna sahip bir ağ oluşturulur.
network = create_2h_network(dimension, 20, 20, 3, 1e-1, True, "sigmoid")
```

---

**Soru 1: İris Çiçeklerinin Tanınması**

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

```
# Maksimum iterasyon sayısı 1000 ve hata sınırı 0.001 seçilir.
epoch = 1000
eth = 1e-3

# Önceki sorudaki gibi eğitim ve eğitilen ağ ile test yapılır.
start_time = time.time()
result = educationprocess(trainList, network, epoch, eth, dimension)
print("Education time: %f" % (time.time() - start_time))
print("Training is over in %d steps with %.7f error" % (result[0], result[1]))

testresult = testprocess(testList, network, dimension)
print("Made %d wrong predictions out of %d patterns" % (testresult[0], testresult[1]))
```

**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.

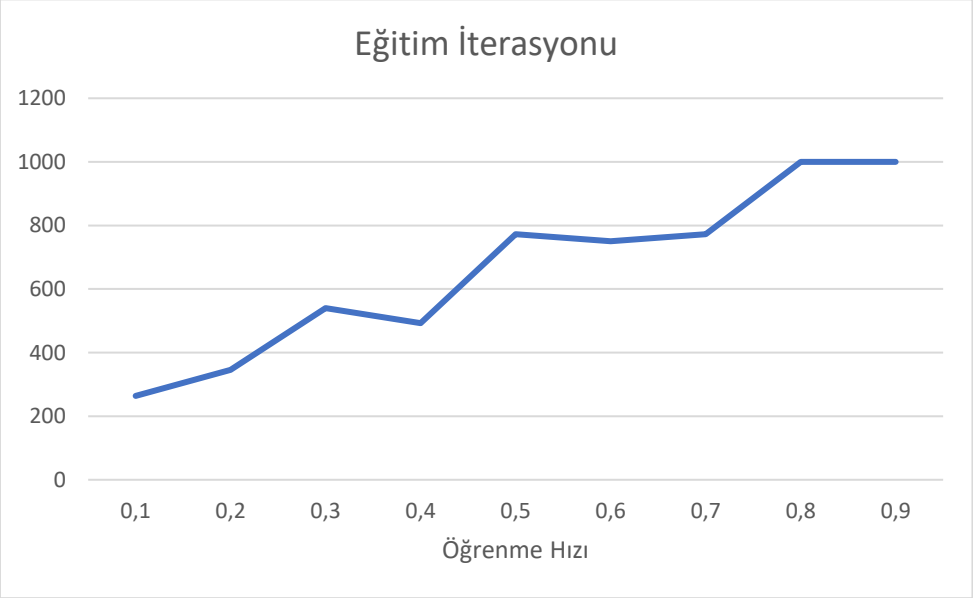
```
Education time: 35.090211
Training is over in 183 steps with 0.0402392 error
Made 2 wrong predictions out of 75 patterns
```

---

## Soru 1: İris Çiçeklerinin Tanınması

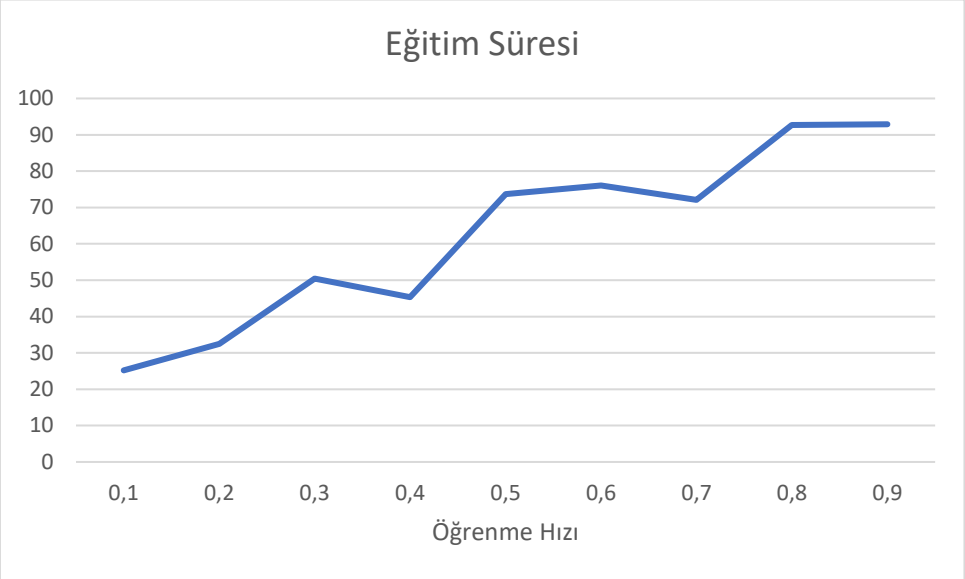
### Momentum varken değişken öğrenme hızı:

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.

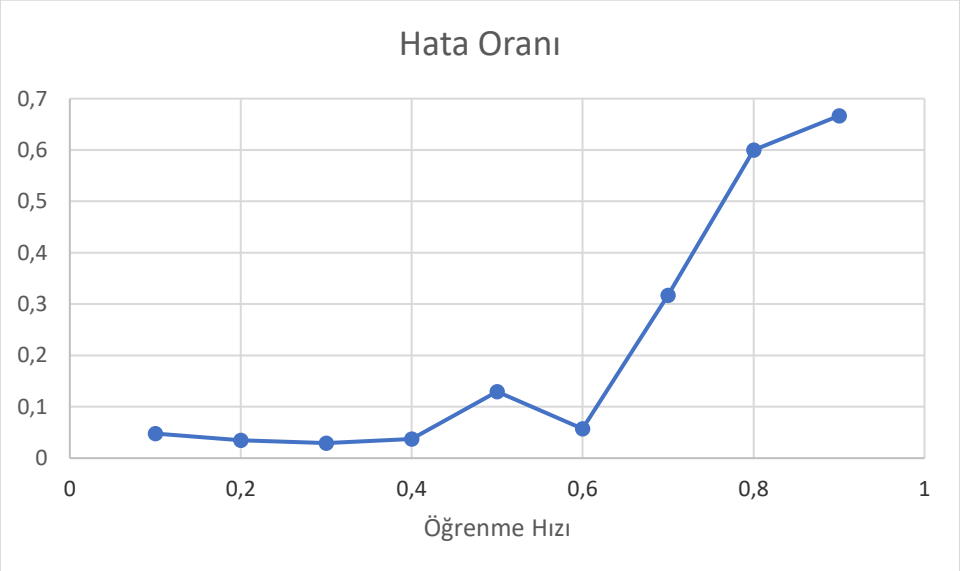


**Soru 1: İris Çiçeklerinin Tanınması**

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.



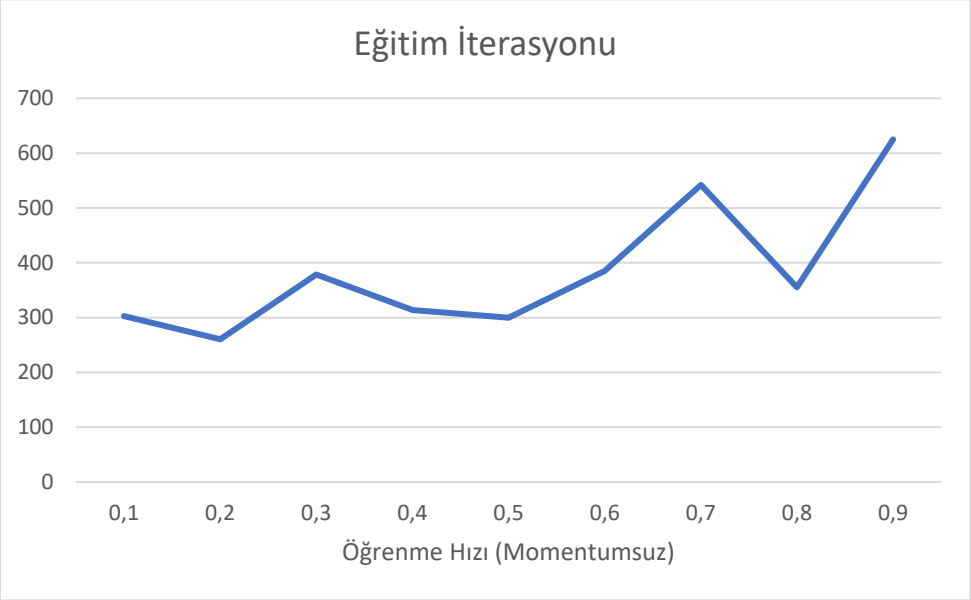

Hata oranında da benzer trendler görülür. Öğrenme hızı 0.4 olana kadar %5'in altında bir hata oranı gözlenirken sonrasında yükselmiş, 0.6'dan sonra ise hızlı bir yükselişle %66'ya kadar çıkmıştır. 0.8 ve 0.9 değerlerinde eğitim başarısız tamamlandığı için yüksek hata oranları gözlenir fakat başarılı 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.



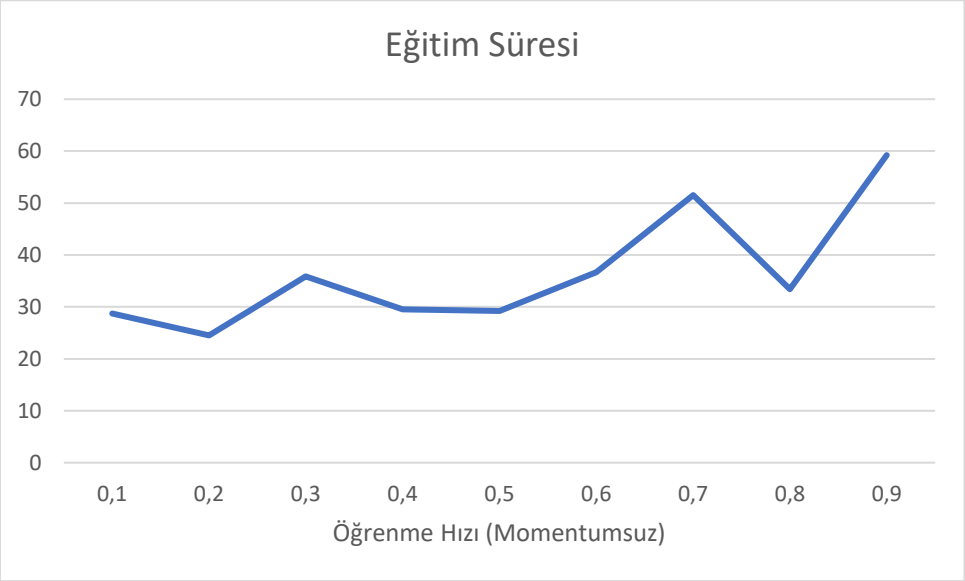
## Soru 1: İris Çiçeklerinin Tanınması

### 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.

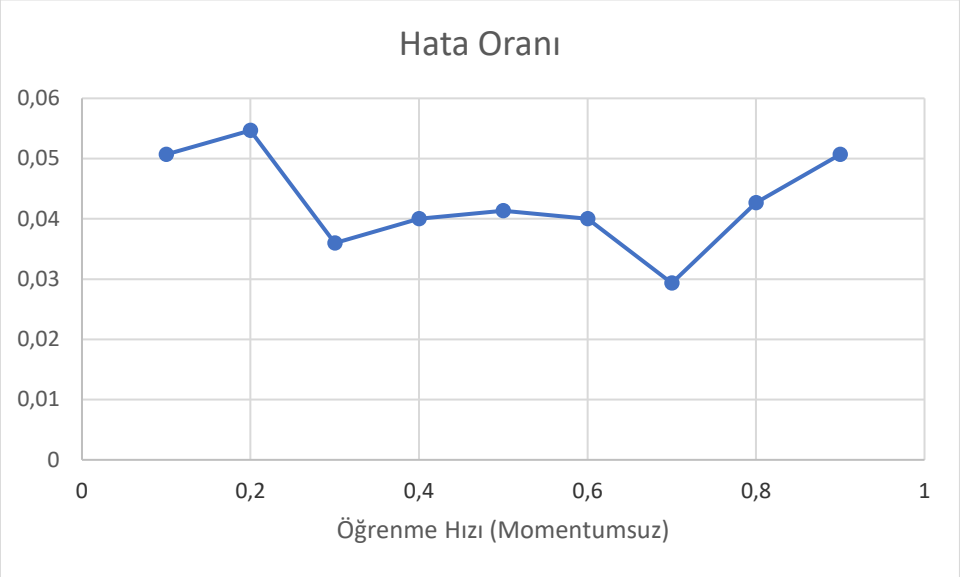



Eğitim süresi de, ağ yapısı değişmediği için iterasyonla aynı hareket ettiği görülür. Eğitim süresi olarak, 0.1 noktası haricinde, momentumun olduğu durumdan daha avantajlı olduğu görülür.



**Soru 1: İris Çiçeklerinin Tanınması**

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



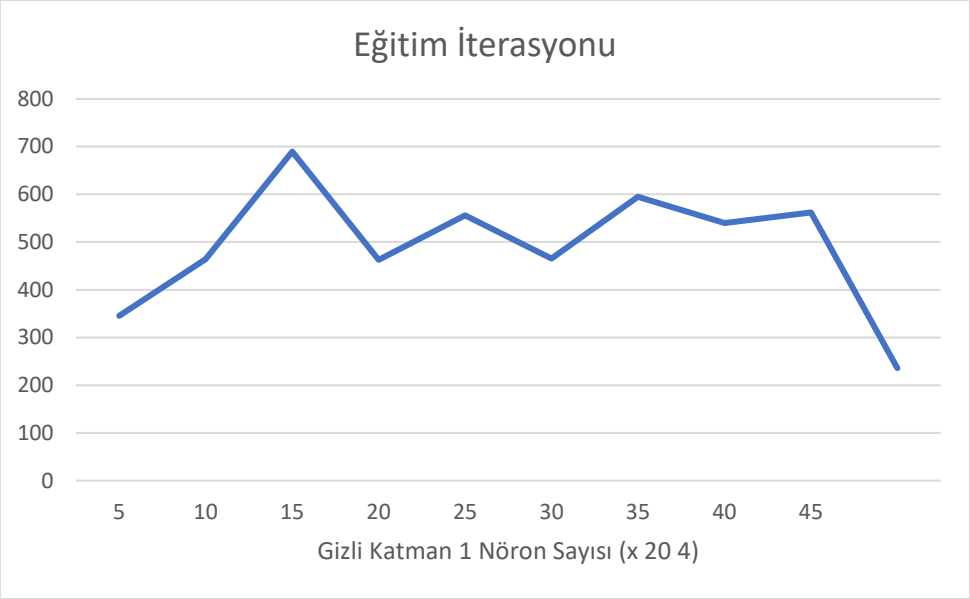

Momentum olmadan kullanılan ağ, eğitim süresi açısından çoğunlukla daha avantajlıyken hata oranında, düşük öğrenme hızlarında, momentumun kullanılması küçük bir oranla daha doğru sonuçlar elde edilmesini sağlıyor. Önceliğin farklı olduğu uygulamalarda, momentumun kullanılıp kullanılmaması fark gösterecektir.

---

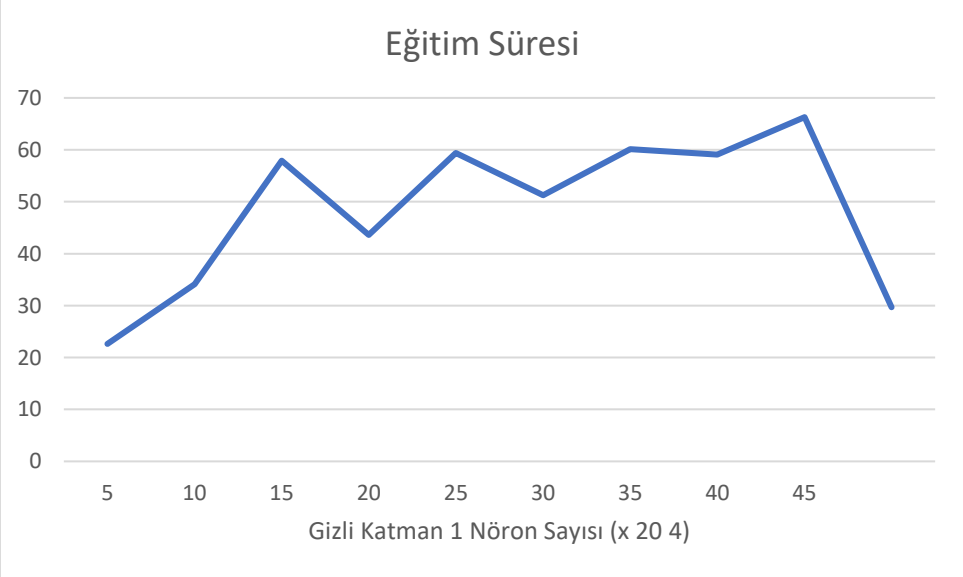
**Soru 1: İris Çiçeklerinin Tanınması**

**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.

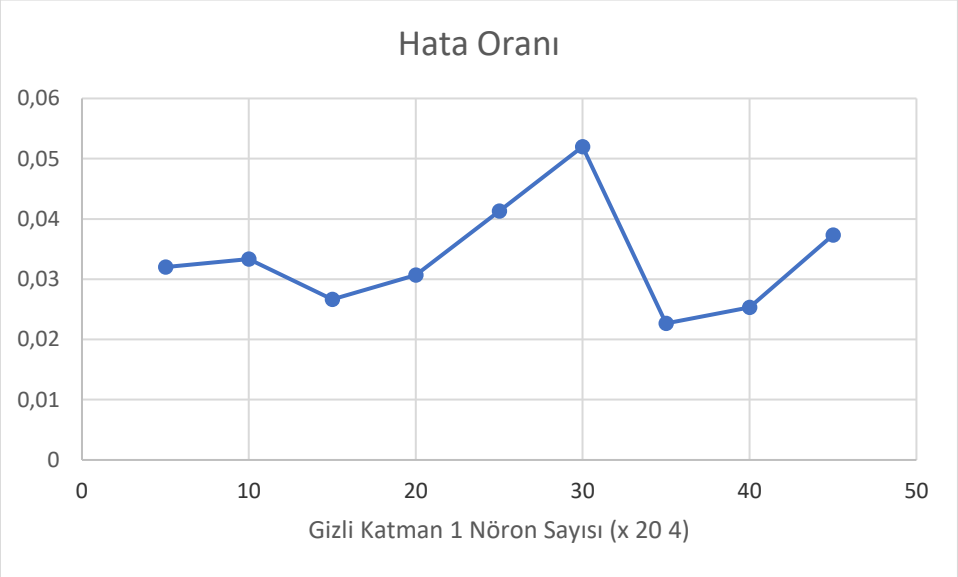



Eğitim süresi de iterasyon sayısı ve ağırlık sayısıyla orantılı olacağı için, iterasyon sayısının biraz daha artan eğilimli bir fonksiyonu olarak görülmektedir. Şaşırtıcı bir şekilde 5 ve 50 nöron sayılarının aradakilerden daha ideal eğitim süreleri verdiği görülmektedir.



**Soru 1: İris Çiçeklerinin Tanınması**

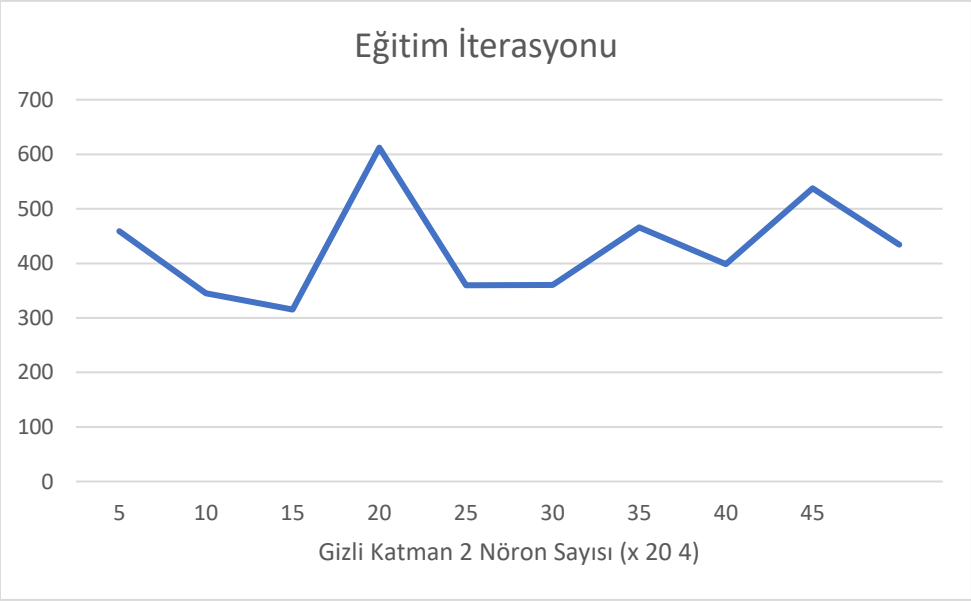
Hata oranı açısından da ara değerlerin pek iyi bir performansa sahip olmadığı, eğitim süreleri fazla olmasına rağmen 15, 35 ve 40 nöron sayılarının %3'ün altında hata verdiği görülür.




---

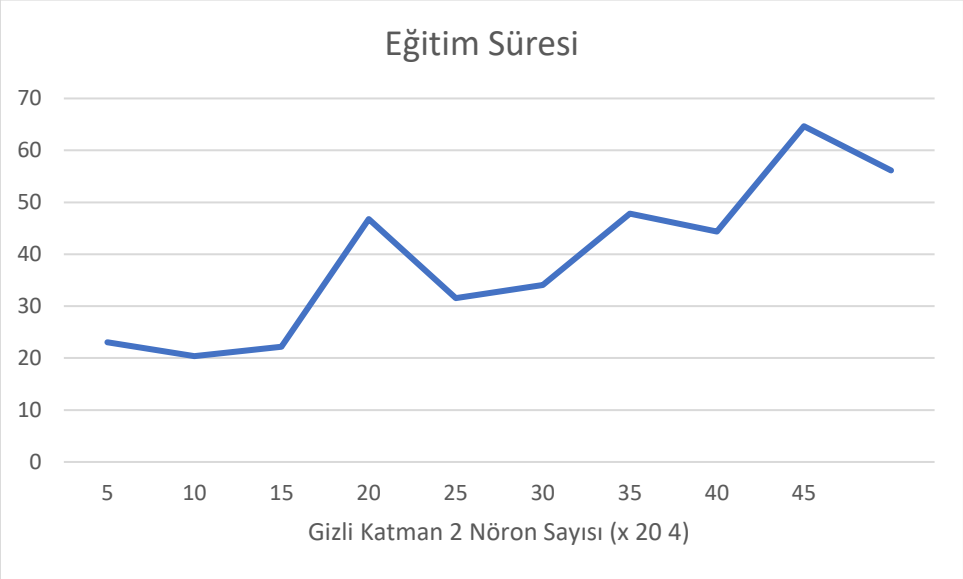
**Momentum yokken değişken ikinci katman nöron sayısı:**

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.

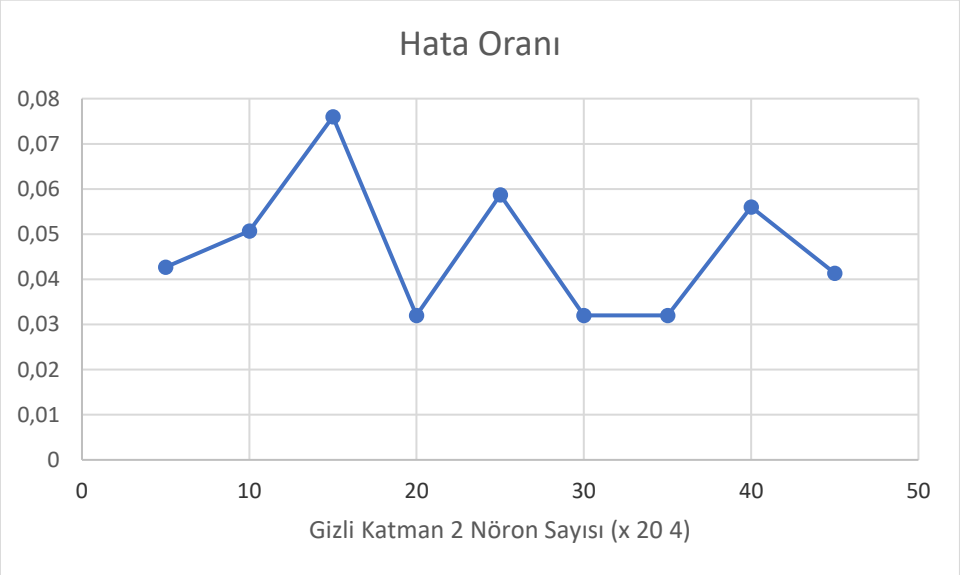


**Soru 1: İris Çiçeklerinin Tanınması**

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.



## Soru 1: İris Çiçeklerinin Tanınması

### Deney Kodları:

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

```
# Sorunun devamında önceki soruda yapılan deneyler yapılır ve sonuçlar kaydedilir.
cresult = np.zeros((9,3), dtype=float)
for i in range(9):
    eduiter = 0
    testacc = 0
    edutime = 0
    for j in range(10):
        network = create_2h_network(dimension, 20, 20, 3, (i+1)*1e-1, True, "sigmoid")
        start_time = time.time()
        result = educationprocess(trainList, network, epoch, eth, dimension)
        edutime += time.time() - start_time
        print("Education time for cresult, %d iteration: %f" % (i+1, (time.time() - start_time)))
        eduiter += result[0]
        testresult = testprocess(testList, network, dimension)
        print("Made %d wrong predictions out of %d patterns" % (testresult[0], testresult[1]))
        testacc += testresult[0]/testresult[1]
    cresult[i,0] = eduiter/10
    cresult[i,1] = edutime/10
    cresult[i,2] = testacc/10
```

```
cwomresult = np.zeros((9,3), dtype=float)
for i in range(9):
    eduiter = 0
    testacc = 0
    edutime = 0
    for j in range(10):
        network = create_2h_network(dimension, 20, 20, 3, (i+1)*1e-1, True, "sigmoid")
        start_time = time.time()
        result = educationprocess(trainList, network, epoch, eth, dimension, False)
        edutime += time.time() - start_time
        print("Education time for cwomresult, %d iteration: %f" % (i+1, (time.time() - start_time)))
        eduiter += result[0]
        testresult = testprocess(testList, network, dimension)
        print("Made %d wrong predictions out of %d patterns" % (testresult[0], testresult[1]))
        testacc += testresult[0]/testresult[1]
    cwomresult[i,0] = eduiter/10
    cwomresult[i,1] = edutime/10
    cwomresult[i,2] = testacc/10
```

## Soru 1: İris Çiçeklerinin Tanınması

```
h1ncountresult = np.zeros((10,3), dtype=float)
for i in range(10):
    eduiter = 0
    testacc = 0
    edutime = 0
    for j in range(10):
        network = create_2h_network(dimension, (i+1)*5, 20, 3, 5e-1, True, "sigmoid")
        start_time = time.time()
        result = educationprocess(trainList, network, epoch, eth, dimension, False)
        edutime += time.time() - start_time
        print("Education time for h1ncountresult, %d iteration: %f" % (i+1, (time.time() - start_time)))
        eduiter += result[0]
        testresult = testprocess(testList, network, dimension)
        print("Made %d wrong predictions out of %d patterns" % (testresult[0], testresult[1]))
        testacc += testresult[0]/testresult[1]
    h1ncountresult[i,0] = eduiter/10
    h1ncountresult[i,1] = edutime/10
    h1ncountresult[i,2] = testacc/10
```

```
h2ncountresult = np.zeros((10,3), dtype=float)
for i in range(10):
    eduiter = 0
    testacc = 0
    edutime = 0
    for j in range(10):
        network = create_2h_network(dimension, 20, (i+1)*5, 3, 5e-1, True, "sigmoid")
        start_time = time.time()
        result = educationprocess(trainList, network, epoch, eth, dimension, False)
        edutime += time.time() - start_time
        print("Education time for h2ncountresult, %d iteration: %f" % (i+1, (time.time() - start_time)))
        eduiter += result[0]
        testresult = testprocess(testList, network, dimension)
        print("Made %d wrong predictions out of %d patterns" % (testresult[0], testresult[1]))
        testacc += testresult[0]/testresult[1]
    h2ncountresult[i,0] = eduiter/10
    h2ncountresult[i,1] = edutime/10
    h2ncountresult[i,2] = testacc/10
```

---

**Soru 2: Sistem Yaklaşımı**

Bu soru için, çok katmanlı algılayıcı kullanarak

$$y(k) = (0.8 - 0.5e^{-y^2(k-1)})y(k-1) - (0.3 + 0.9e^{-y^2(k-1)})y(k-2) + 0.1\sin(\pi y(k-1)) + e(k)$$

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 $y^2(k-1), y(k-1)\ ve\ y(k-2)$ 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.

```
# Eğitimde girişe önceki veriler girileceği için bağlı liste oluşturulur.
class YNode():
    # Bağlı listede indis, gerçek değer, tahmin edilen değer ve önceki değer bulunur.
    def __init__(self, index, yprev = None, ypred = random.random()):
        self.index = index
        self.pred = ypred
        # İndis 0'dan büyükse linkler ve gerçek değer atanır. 0 indisi için değer 0 olacak, önceki değer
        # kendisini gösterecektir. Böylece nedensel bir sistem tanımlanıp başlangıç koşulu olarak 0 seçilir.
        if index > 0:
            self.links(yprev)
            self.targetfunc()
```

```
    def links(self, yprev):
        self.prev = yprev

    # Bir önceki ve iki önceki değerler hedef fonksiyona girilir ve gerçek değer elde edilir.
    def targetfunc(self):
        self.val = targetfunction(self.prev.val, self.prev.prev.val)

    # Değer ve linkler bu fonksiyonla da atanabilir.
    def setnode(self, val, yprev):
        self.val = val
        self.prev = yprev
```

## Soru 2: Sistem Yaklaşımı

### Hedef fonksiyon:

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

```
# Ödev sayfasındaki istenen fonksiyon. e_k değeri -0.0005-0.0005 arasında rastgele bir sayıdır.
def targetfunction(pval, ppval):
    e_k = random.random()*0.001 - 0.0005
    val = pval*(0.8 - math.exp(-(pval**2))) - ppval*(0.3 + math.exp(-(pval**2))) + 0.1*math.sin(math.pi*pval) + e_k
    return val
```

---

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.

```
__location__ = os.path.realpath(os.path.join(os.getcwd(), os.path.dirname(__file__)))
# Girilen veri boyutu 3 seçilir.
dimension = 3

trainList = []
# Başlangıç koşulu oluşturulur ve y(0) = 0 seçilir.
yinit = YNode(0)
yinit.setnode(0, yinit)
yinit.pred = 0
yprev = yinit
# y(1)'den y(99)'a kadar veri oluşturulup eğitim kümesine atanır.
for i in range(1,100):
    y = YNode(i, yprev)
    yprev = y
    trainList.append(yprev)

# Test listesinde ise y(100)'den y(299)'a kadar olan değerler vardır.
testList = []
for i in range(100,300):
    y = YNode(i, yprev)
    yprev = y
    testList.append(yprev)
```

---

**Soru 2: Sistem Yaklaşımı**

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

```
# Verilerin maksimum ve minimum değerleri elde edilir.
maxval = 0
minval = 0
for node in trainList:
    if node.val > maxval:
        maxval = node.val
    if node.val < minval:
        minval = node.val

# Veriler 0-1 aralığında ölçeklenir.
for node in trainList:
    node.val = (node.val - minval)/(maxval - minval)
for node in testList:
    node.val = (node.val - minval)/(maxval - minval)
```

---

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.

```
# İlk katmanda 20, ikincide 20 ve son katmanda 1 nöronlu bir ağ oluşturulur. Ağın eğitim hızı 0.3, aktivasyon
# fonksiyonu sigmoid olacak, bias terimini kendisi atayacaktır.
network = create_2h_network(dimension, 20, 20, 1, 3e-1, True, "sigmoid")

# Maksimum 1000 iterasyonlu ve 0.0001 hata sınırına sahip eğitim başlatılır.
epoch = 1000
eth = 1e-4
start_time = time.time()
result = educationprocess(trainList, network, epoch, eth, dimension)
print("Education time: %f" % (time.time() - start_time))
print("Training is over in %d steps with %.7f error" % (result[0], result[1]))

testNetwork(trainList, network, dimension)
result = testNetwork(testList, network, dimension)
print("x = 100-299 için test edildi ve ortalama %", 100*result[1][0], " hata elde edildi.")
```

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.

```
Education time: 70.110885
Training is over in 454 steps with 0.0021441 error
x = 100-299 için test edildi ve ortalama % [0.31843219]  hata elde edildi.
```

## Soru 2: Sistem Yaklaşımı

### 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

$[y(k-1),\ y(k-2),\ y^2(k-1)]$ numpy arrayi oluşturulur. Bu, bağlı listeden çekilerek sağlanır. Sınıf bilgisi de aynı şekilde $[y(k)]$ numpy arrayi şeklindedir. Ağın çalışması için veri yapısının numpy arrayi şeklinde olması önemlidir.

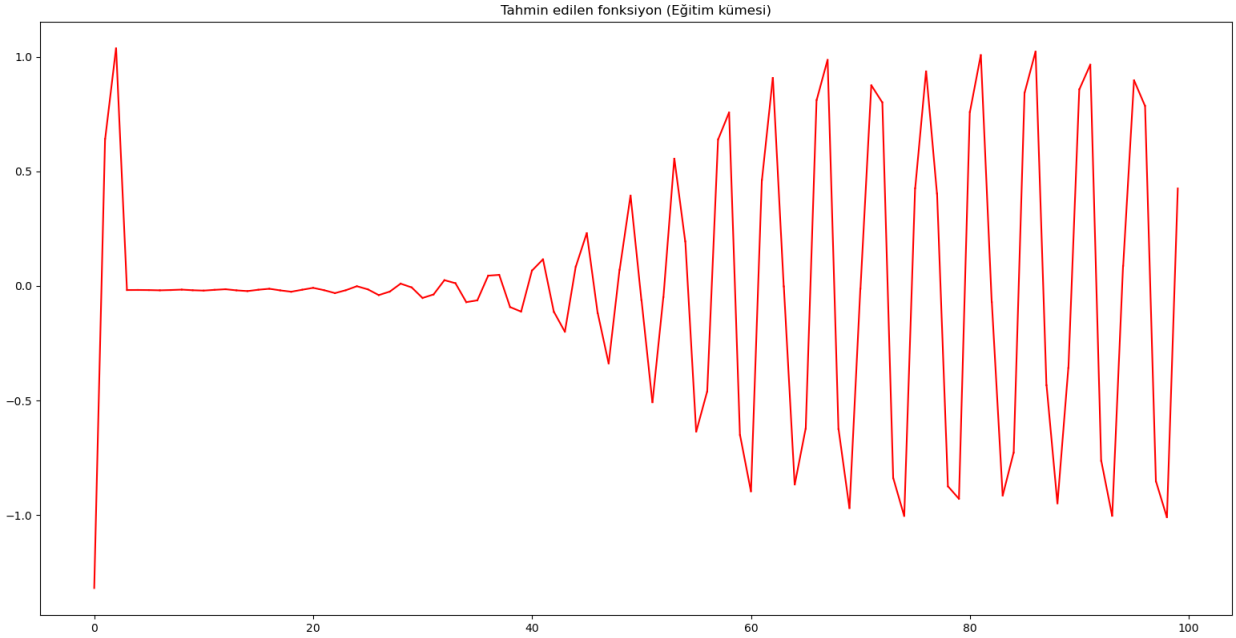
```python
def testNetwork(datalist, network, dim):
    Eort = 0
    for data in datalist:
        # Veri ve sınıf bilgisi oluşturulurken, veri için iki önceki değer, bir önceki değer ve karesi çekilir.
        imdata = np.array([data.prev.val, data.prev.prev.val, data.prev.val**2], dtype=float).reshape(dimension, 1)
        # Sınıf bilgisi içinse gerçek değer çekilir.
        clsdata = np.array([data.val], dtype=float)
```

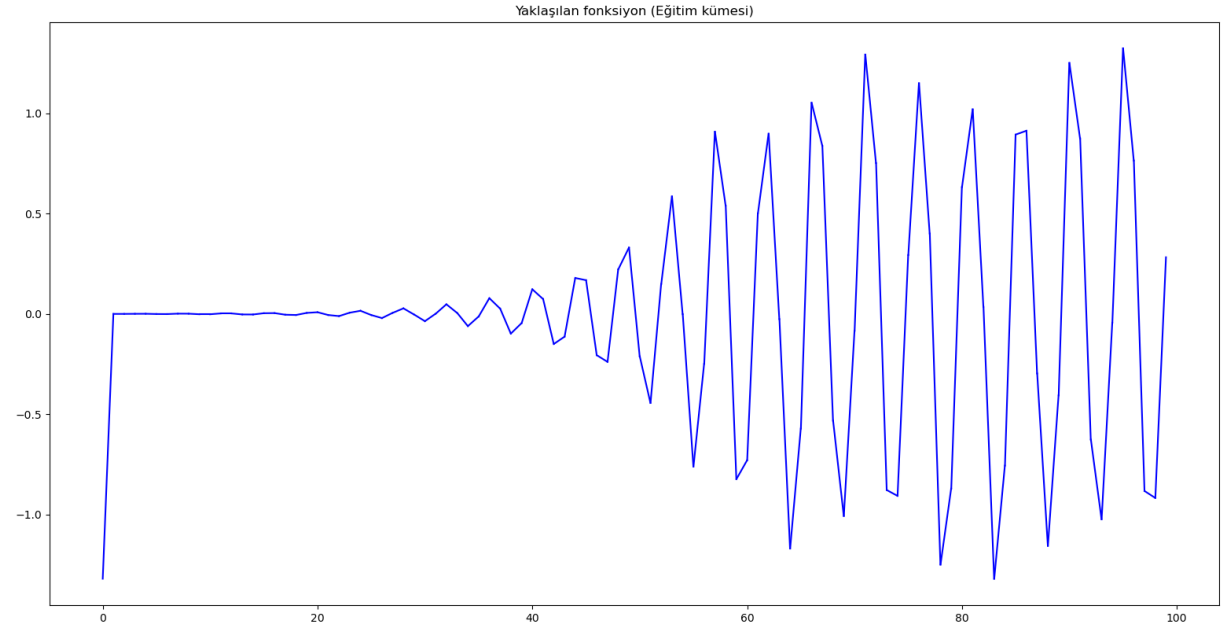
```python
def educationprocess(datalist, network, epoch, errorthreshold, dim, moment = True):
    Eortprev = 0
    ResetCount = 0
    result = np.zeros(2)
    result[0] = epoch
    for i in range(epoch):
        random.shuffle(datalist)
        Eort = 0
        # Testte olduğu gibi, bu eğitimde de veriler farklı şekilde elde edilir.
        for data in datalist:
            # Veri ve sınıf bilgisi oluşturulurken, veri için iki önceki değer, bir önceki değer ve karesi çekilir.
            imdata = np.array([data.prev.val, data.prev.prev.val, data.prev.val**2], dtype=float).reshape(dimension, 1)
            # Sınıf bilgisi içinse gerçek değer çekilir.
            clsdata = np.array([data.val], dtype=float)
            E = educate(imdata, network, clsdata, moment)
```

---

## Soru 2: Sistem Yaklaşımı

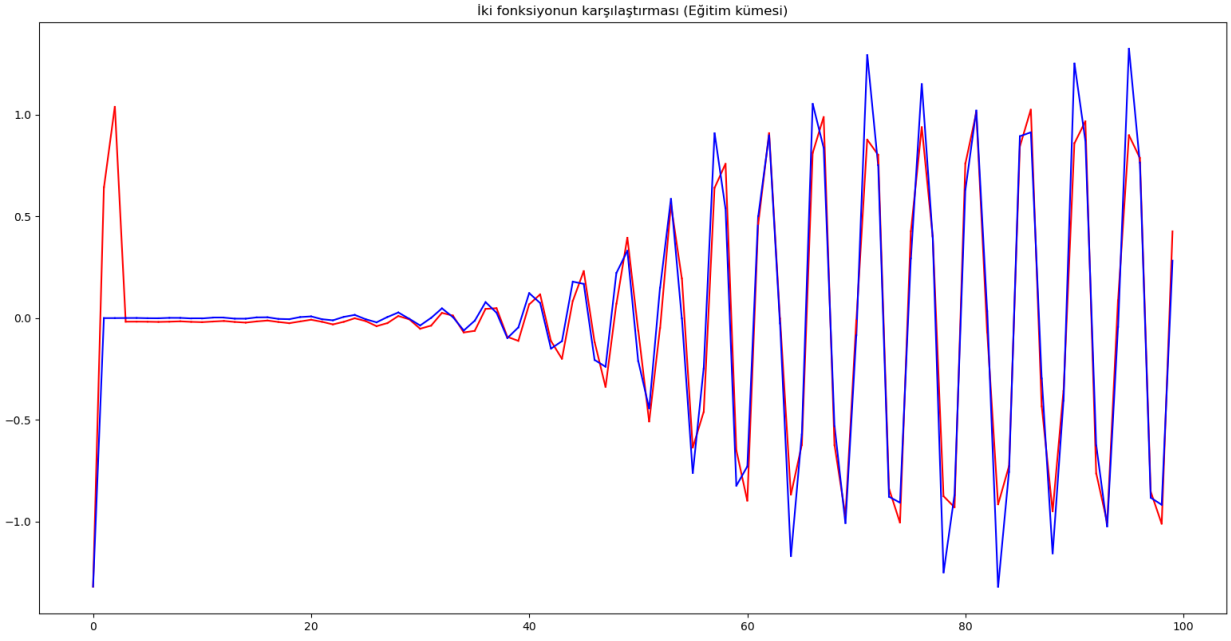
Veriler, ters ölçeklenip karşılaştırılmak üzere çizildiğinde aşağıdaki şekiller ortaya çıkar. Sigmoid fonksiyon kullanılması ve verilerin [0,1] arasına çekilmesinden dolayı, ilk değerlerde garip bir yol izlendiği görülür. Belki tanh fonksiyonunun bu problem için daha uygun olabileceği görülür. Başlangıçta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



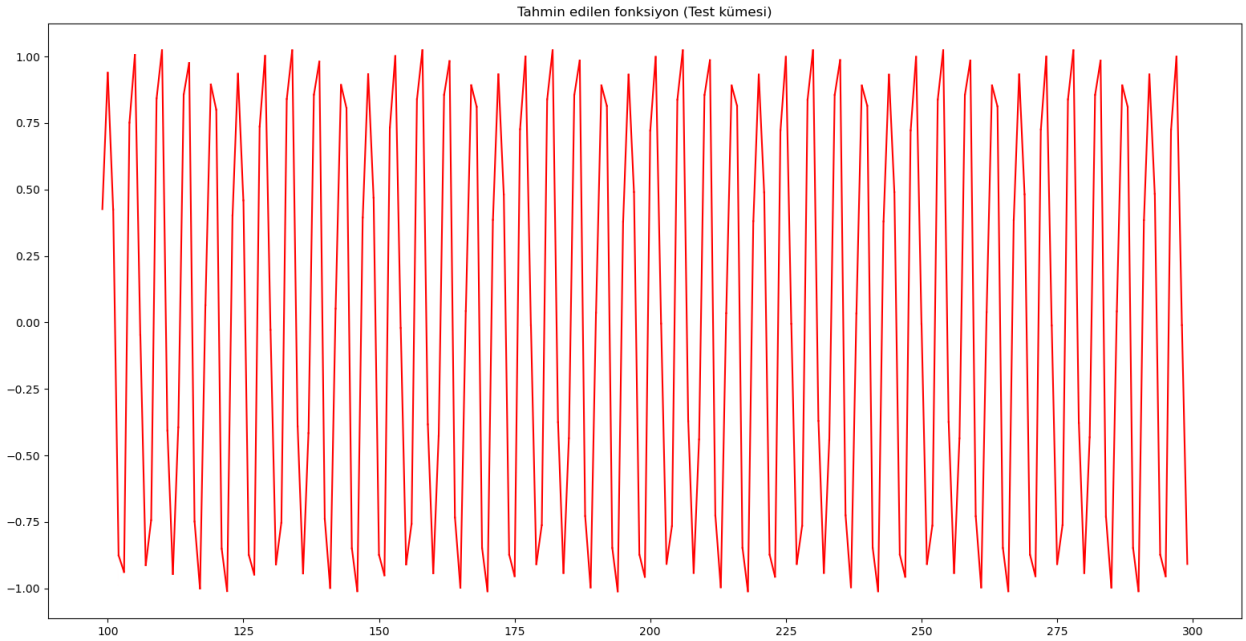
## Soru 2: Sistem Yaklaşımı

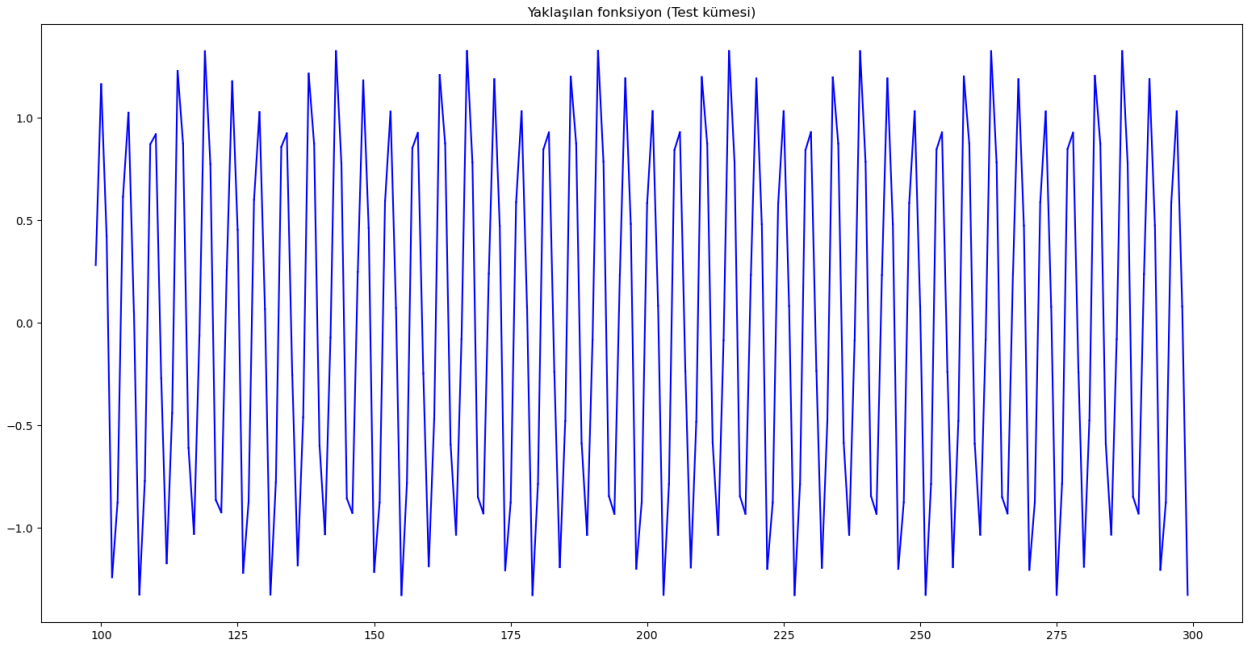
Eğitilen ağ, beklendiği gibi eğitim kümesiyle yaklaşık olarak örtüşmektedir. Eğitimin başarısıyla ilgili gerçek bilgi test kümesiyle olan karşılaştırmadan gelmektedir.



## Soru 2: Sistem Yaklaşımı

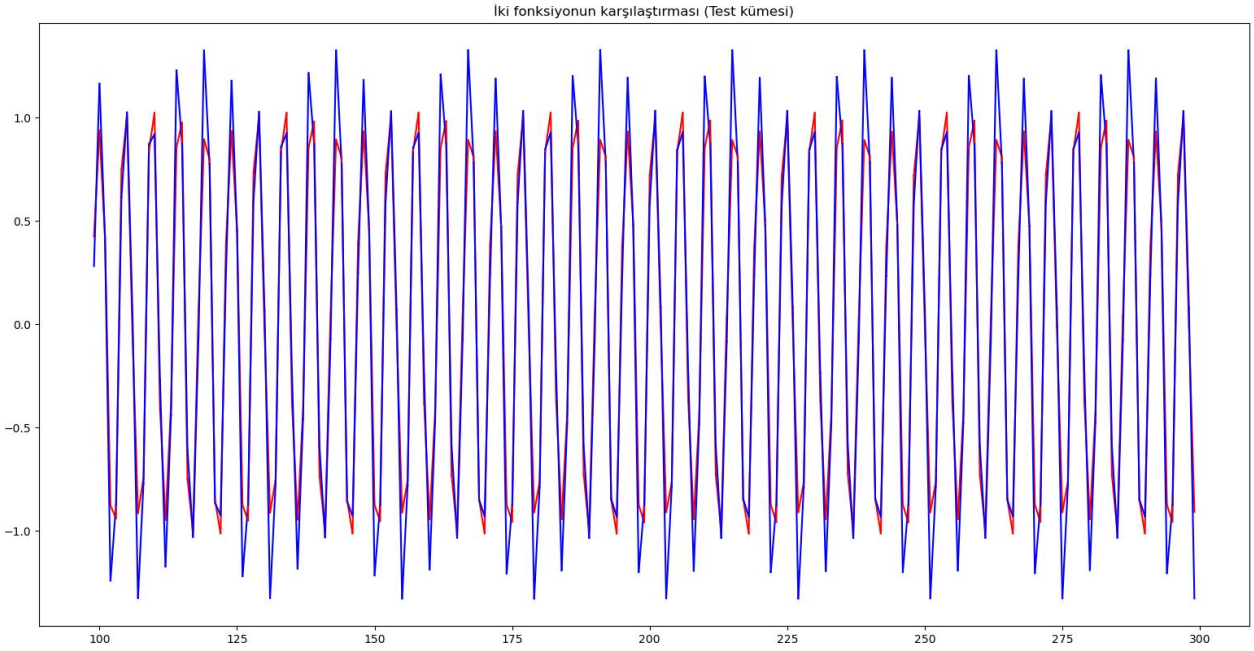
Test kümesindeki veriler başlangıç koşulundan bağımsız bir şekilde salınımdadırlar. Sistem kararlılığını sürdürmektedir, bu eğitimin başarılı olmasında büyük katkı sağlamaktadır.



## Soru 2: Sistem Yaklaşımı

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.



**Soru 2: Sistem Yaklaşımı**

**Çizim için kullanılan kodlar:**

```
# Eğitim ve test kümelerinde, tahmin edilen verilerle karşılaştırma yapılır.
plt.figure()
for node in trainList:
    x = np.linspace(node.prev.index, node.index - 0.01, 100)
    m = (node.val - node.prev.val)/0.99
    normalized_y = node.prev.val + (x - node.prev.index)*m
    y = minval + normalized_y*(maxval - minval)
    plt.plot(x, y, color = 'blue')
plt.title("Yaklaşılan fonksiyon (Eğitim kümesi)")


plt.figure()
for node in trainList:
    x = np.linspace(node.prev.index, node.index - 0.01, 100)
    m = (node.pred - node.prev.pred)/0.99
    normalized_y = node.prev.pred + (x - node.prev.index)*m
    y = minval + normalized_y*(maxval - minval)
    plt.plot(x, y, color = 'red')
plt.title("Tahmin edilen fonksiyon (Eğitim kümesi)")


plt.figure()
for node in trainList:
    x = np.linspace(node.prev.index, node.index - 0.01, 100)
    m = (node.pred - node.prev.pred)/0.99
    normalized_y = node.prev.pred + (x - node.prev.index)*m
    y = minval + normalized_y*(maxval - minval)
    plt.plot(x, y, color = 'red', label='Tahmin edilen')
    m = (node.val - node.prev.val)/0.99
    normalized_y = node.prev.val + (x - node.prev.index)*m
    y = minval + normalized_y*(maxval - minval)
    plt.plot(x, y, color = 'blue', label='Gerçekte olan')
plt.title("İki fonksiyonun karşılaştırması (Eğitim kümesi)")
```

**Soru 2: Sistem Yaklaşımı**

```
plt.figure()
for node in testList:
    x = np.linspace(node.prev.index, node.index - 0.01, 100)
    m = (node.val - node.prev.val)/0.99
    normalized_y = node.prev.val + (x - node.prev.index)*m
    y = minval + normalized_y*(maxval - minval)
    plt.plot(x, y, color = 'blue')
plt.title("Yaklaşılan fonksiyon (Test kümesi)")


plt.figure()
for node in testList:
    x = np.linspace(node.prev.index, node.index - 0.01, 100)
    m = (node.pred - node.prev.pred)/0.99
    normalized_y = node.prev.pred + (x - node.prev.index)*m
    y = minval + normalized_y*(maxval - minval)
    plt.plot(x, y, color = 'red')
plt.title("Tahmin edilen fonksiyon (Test kümesi)")


plt.figure()
for node in testList:
    x = np.linspace(node.prev.index, node.index - 0.01, 100)
    m = (node.pred - node.prev.pred)/0.99
    normalized_y = node.prev.pred + (x - node.prev.index)*m
    y = minval + normalized_y*(maxval - minval)
    plt.plot(x, y, color = 'red', label='Tahmin edilen')
    m = (node.val - node.prev.val)/0.99
    normalized_y = node.prev.val + (x - node.prev.index)*m
    y = minval + normalized_y*(maxval - minval)
    plt.plot(x, y, color = 'blue', label='Gerçekte olan')
plt.title("İki fonksiyonun karşılaştırması (Test kümesi)")
plt.show()
```