

# **Yapay Sinir Ağları**

## **4.Ödev**

**16.01.2021**

**Mehmet Şerbetçioğlu -- 040160056**

**Ahmet Hulusi Tarhan -- 040170738**

## Soru 1: Lineer Regresyon

İkinci ödevde verilen fonksiyon üzerinde yaklaşacak Elman Ağrı oluşturulacaktır.

Yaklaşılacak fonksiyon:

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

Fonksiyonun argümanları arasında  $y(k-1)$  ve  $y(k-2)$  bulunmaktadır.  $e(k)$  gürültü olup küçük rastgele bir değer atanır. Eğitilecek ağda girişler  $[y(k-1), y(k-2), y(k-3), y(k-4), y(k-5)]$  seçilir. Başlangıçta ağ boyutları belirlenir.

```
__location__ = os.path.realpath(os.path.join(os.getcwd(), os.path.dirname(__file__)))
# Girilen veri boyutu 3 seçilir. Eğitim için çıkış boyutu 1, gizli katman nöron sayısı 5 seçilir.
dim = 5
y_dim = 1
x_dim = 10
```

Burada “dim” girişteki verinin boyutu, “y\_dim” ağ çıkışının boyutu ve “x\_dim” gizli katmandaki nöron sayısıdır.

### Verilerin Saklanması:

Bu aşamanın ardından ikinci ödevdekiyle aynı şekilde test ve eğitim kümeleri oluşturulur. Eğitim kümesinde  $k = [1, 99]$  değerleri, test kümesinde ise  $k = [100, 299]$  değerleri bulunur.

```
# Veriler ikinci ödevdekiyle aynı şekilde oluşturulur.
trainList = []
# Başlangıç koşulu oluşturulur ve  $y(0) = 0$  seçilir.
yinit = YNode(0)
yinit.setnode(0.1, yinit)
yinit.pred = 0.1
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)
```

Oluşturulan veriler düğümler şeklinde saklanır. Bu düğümlerde “yd” değeri, düğümün “k” indisini ve ağın veri için tahmin ettiği “y” değeri bulunmaktadır. Veri kümeleri oluşturuluren başlangıç değeri  $y(0) = 0.1$  seçilir ve kendisine bağlanır. Ek olarak, düğümler içinde bir önceki düğümlle bağlantı bulunmaktadır. Bu şekilde girişteki  $[y(k-1), y(k-2), \dots, y(k-5)]$  verisi her düğüm için çekilebilmektedir.

## Soru 1: Lineer Regresyon

```
# 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 indisini 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
```

Veri kümeleri oluşturulduktan sonra eğitim kümesi [0,1] aralığında ölçeklendirilir. Maksimum ve minimum değerleri daha sonra geri ölçeklemede kullanılmak üzere saklanır.

```
# 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)
```

## Eğitim Hazırlıkları:

Öncelikle eğitimin maksimum iterasyon sayısı ve hata limiti seçilir. Bu değerler ikinci ödevde karşılaştırılmak adına, ikinci ödevde kullanılan değerlerle aynı seçilir. Bu değerler  $epoch = 1000$ ,  $eth = 0.001$  olarak seçilir.

```
# Hata limiti ve maksimum iterasyon sayısı ikinci ödevdekiyle
# karşılaştırılmak amacıyla aynı değerler alınır.
epoch = 1000
eth = 1e-4
```

## Soru 1: Lineer Regresyon

Eğitim için kullanılacak ağ oluşturulur. Ağın öğrenme hızı 0.1 olup kullanılacak aktivasyon fonksiyonu sigmoid olacaktır.

```
# Öğrenme hızı 0.1 ve aktivasyon fonksiyonu sigmoid seçilir ve ağ oluşturulur.  
network = generate_network(0.1, "sigmoid")
```

## Ağ Oluşumu ve Nöron Yapısı:

Ağ oluşturulurken “x\_dim” sayısı kadar nöron oluşturulur. Ağda aktivasyon fonksiyonu kullanan tek katman gizli katman olduğu için, çıkış ve giriş katmanı için ayrıca nöron oluşturulması gerekmekz.

```
# Ağ oluşturma fonksiyonu.  
def generate_network(learning_rate, activation_type):  
    network = []  
    # Seçilen öğrenme hızı ve aktivasyon fonksiyonu ile yeni bir ağ oluşturulur.  
    # Elman ağında aktivasyon fonksiyonunun bulunduğu tek aşama  $x(k)$  değerinin bulunduğu aşamadır.  
    # Bu sebeple yalnızca gizli katmandaki nöron oluşturulup diğer katmanların ağırlıkları da  
    # bu nöronda saklanabilir.  
    for i in range(x_dim):  
        #  $x\_dim$ ,  $y\_dim$  ve  $dim$  sırayla gizli katmandaki nöron sayısı, ağ çıkışı boyutu ve girişteki  
        # verinin boyutu olup küresel değişkenlerdir. Fonksiyonun dışında belirlenir.  
        new_Neuron = Neuron(dim, x_dim, y_dim, learning_rate, activation_type)  
        network.append(new_Neuron)  
    return network
```

Ağ oluşumunda kullanmak üzere nöron sınıfı tanımlıdır. Nöron sınıfında girişteki verinin boyutu ( $u\_dim$ ), gizli katmandaki nöron sayısı ( $x\_dim$ ), ağ çıkışı boyutu ( $y\_dim$ ), öğrenme hızı ve aktivasyon fonksiyonu bulunur. Nöron öncelikle sıfırlanır.

```
#Nöron yapısı.  
class Neuron():  
    def __init__(self, u_dim, x_dim, y_dim, learning_rate = 1e-2, activation_type = "tanh"):  
        # Nöronun ilk değerleri atanır.  
        self.u_dim = u_dim  
        self.x_dim = x_dim  
        self.y_dim = y_dim  
        self.learning_rate = learning_rate  
        self.activation_type = activation_type  
  
        self.reset()  
  
    def reset(self): ...  
  
    def forward(self, data, prevx): ...  
  
    def activation_function(self, data): ...  
  
    def updateN(self, w_u, w_x, w_y): ...
```

## Soru 1: Lineer Regresyon

Nöron sıfırlandığında giriş katmanı, gizli katman ve çıkış katmanı ağırlıklarının başlangıç değerleri atanır. Bu değerler  $[-0.05, 0.05]$  aralığında rastgele bir şekilde seçilir. Ayrıca veri için boş bir dizi açılıp önceki ağırlıklar güncel ağırlıklarla eşitlenir. Önceki ağırlıklar momentum için gereklidir.

```
def reset(self):
    # Nöronun ilk ağırlıkları [-0.5,0.5] arasında rastgele seçilir.
    # Nöronla ilgili içerik katmanı, çıkış katmanı ve giriş katmanı ağırlıkları tek nöronda saklanır.
    self.w_u = np.zeros((1, self.u_dim), dtype=float)
    for i in range(self.u_dim):
        self.w_u[0,i] = np.random.rand()*0.1 - 0.05

    self.w_x = np.zeros((1, self.x_dim), dtype=float)
    for i in range(self.x_dim):
        self.w_x[0,i] = np.random.rand()*0.1 - 0.05

    self.w_y = np.zeros((self.y_dim, 1), dtype=float)
    for i in range(self.y_dim):
        self.w_y[i,0] = np.random.rand()*0.1 - 0.05

    # Ağırlıkların önceki değerleri momentum için kullanılacaktır.
    self.w_u_prev = self.w_u
    self.w_x_prev = self.w_x
    self.w_y_prev = self.w_y

    self.data = np.zeros((self.u_dim, 1), dtype=float)
```

Bunlar dışında nöron yapısında üç fonksiyon bulunmaktadır. İlk olarak “forward” fonksiyonu, ağıın ileri yol fonksiyonudur. Ağ girişi  $u(k)$  ve nöronun önceki iterasyondaki çıkıştı  $x(k-1)$  ile nöron çıkışındaki  $x(k)$  değerini elde eder.

```
def forward(self, data, prevx):
    # İleri yolda girilen veri ( $u(k)$ ) ve önceki çıkış değerlerini ( $x(k-1)$ ) kullanarak
    # tek nörondaki çıkış değerini ( $x_i(k)$ ) elde eder.
    self.data = np.array(data).reshape((self.u_dim, 1))
    self.lin_comb = np.dot(self.w_u, self.data) + np.dot(self.w_x, prevx)
    self.activation = self.activation_function(self.lin_comb)
    return self.activation
```

“activation\_function” fonksiyonu ise seçilen aktivasyon fonksiyonuna göre çıkışı ve aktivasyon fonksiyonunun türevini elde eder.

```
def activation_function(self, data):
    # Aktivasyon fonksiyonunun kullanıldığı fonksiyon. Türev bilgisi de elde edilir.
    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
```

## Soru 1: Lineer Regresyon

Aktivasyon fonksiyonları için, nöronun içindeki fonksiyona ek olarak farklı aktivasyon fonksiyonlarının tanımlandığı fonksiyonlar bulunmaktadır.

```
# Tanh aktitasyon fonksiyonu. Türev bilgisi de bu fonksiyonla elde edilir.
def tanh(x):
    t=(np.exp(x)-np.exp(-x))/(np.exp(x)+np.exp(-x))
    dt=1-t**2
    return t,dt

# Sigmoid aktitasyon fonksiyonu. Türev bilgisi de bu fonksiyonla elde edilir.
def sigmoid(x):
    s=1/(1+math.exp(-x))
    ds=s*(1-s)
    return s,ds
```

Son olarak nöronların ağırlıklarının güncellendiği bir fonksiyon bulunmaktadır. Bu fonksiyonda momentum da uygulanır. Momentum 0.9 seçilmiş olup çıkarılması için 0'a eşitlenmesi yeterlidir.

```
def updateN(self, w_u, w_x, w_y):
    # Nöron ağırlıklarının güncellenir.
    # Momentum kullanılır.
    # Momentumun çıkarılması için 0'a eşitlenmesi yeterlidir.
    momentum = 0.9
    moment_u = momentum*(self.w_u - self.w_u_prev)
    moment_x = momentum*(self.w_x - self.w_x_prev)
    moment_y = momentum*(self.w_y - self.w_y_prev)
    self.w_u_prev = self.w_u
    self.w_x_prev = self.w_x
    self.w_y_prev = self.w_y
    self.w_u = w_u + moment_u
    self.w_x = w_x + moment_x
    self.w_y = w_y + moment_y
```

## Eğitim Aşaması:

Ağ oluşturulduktan sonra, ağ ve eğitim kümesi eğitime girer.

```
# Eğitim başlatılır.
start_time = time.time()
result = educationprocess(network, trainList[:,], epoch, eth)
print("Eğitim süresi: %.3f saniye" % (time.time() - start_time))
print("Eğitim %d adımda yüzde %.7f hata oranı ile tamamlandı" % (result[0], 100*result[1]))
```

## Soru 1: Lineer Regresyon

Eğitimde, her iterasyonda ağırlıklar ve nöron çıkışları dışarı çekilip eğitildikten sonra nöronlar güncellenecektir. Eğitim tamamlandığında, son iterasyon ve son iterasyondaki ortalama hata döndürülür. Başlangıçtaki  $x(k - 1)$  değeri 0.5 olarak alınır.

```
# Eğitim aşaması fonksiyonu.
def educationprocess(network, nodeList, epoch, eth, shuffle = True):
    prevx = np.zeros((x_dim,1), dtype=float) + 0.5
    currx = np.zeros(x_dim, dtype=float).reshape(x_dim,1)
    Eortprev = 0
    last_iter = epoch
    # 'epoch' iterasyonuna kadar eğitim yapılır.
    for i in range(epoch):
        # Eğitim sonunda son iterasyon ve son iterasyondaki ortalama hata döndürülür.
    return [last_iter, Eort]
```

Eğitimin durması için birden fazla koşul vardır. Bu koşullardan birinin sağlanması eğitimi durduracaktır. Bu koşullar;

- Ortalama hatanın, hata limitinin altına düşmesi,
- Ortalama hatadaki değişimin, ortalama hatanın hata limitiyle çarpımının altına düşmesi,
- Maksimum iterasyon sayısına ulaşılmasıdır.

Her iterasyonun başında ortalama hata sıfırlanır. Ardından eğer “shuffle” değeri doğruysa (varsayılan durumda doğru), liste karıştırılır. Bu işlem çok katmanlı algılayıcı gibi ağların eğitiminin sağlıklı geçmesini sağlarken, Elman Ağı’nda ağın eğitildiği verilerin sırasının önemi olduğu için, ağın etkisiz kullanılmasını sağlayabilir. Beklenti, eğitim listesinin karıştırılması durumunda gizli katman ağırlıklarının 0'a yaklaşırılıp etkisiz bırakılması ve ağın tek katmanlı bir ağ gibi çalışmasıdır. Bu durum test edilecek ve karıştırılan ve karıştırılmayan durumlar karşılaştırılacaktır.

```
for i in range(epoch):
    Eort = 0
    # Eğitim yapılırken listenin karıştırılıp karıştırılmaması belirlenir. Diğer ağlarda her iterasyonda
    # eğitim kümesinin karıştırılması sağlıklı iken Elman ağında 'context unit' ile ağın eğitildiği bir önceki
    # veri önem taşımaktadır. Bu sebeple ağa girilen verinin sıralaması önem kazanmaktadır. Bu sadece
    # speküasyon olup karıştırılması ve karıştırılmaması durumunda test edilecektir.
    if shuffle == True:
        random.shuffle(nodeList)

    # Eğitim kümesi bu problem için düğümlerden oluşur.
    for node in nodeList:
        # Ortalama hata bulunur.
        Eort = Eort/len(nodeList)
        # print("İterasyon ", i+1, " hata oranı: %", "%7f" % (100*Eort[0][0]))
        # Ortalama hata ile seçilen hata limiti karşılaştırılır. Hata, önceki hataya göre yeterince değişimmiş veya
        # seçilen hata limitinin altına düşmüştse eğitim sonlandırılır.
        if abs(Eortprev - Eort) < (Eort*eth) or Eort < eth:
            last_iter = i+1
            break
    Eortprev = Eort
```

## Soru 1: Lineer Regresyon

Her veri düğümü için bir eğitim döngüsü vardır. Öncelikle ağa girilecek  $[y(k-1), y(k-2), \dots, y(k-5)]$  verileri bu düğüm ile çekilir. Ardından bu veriler ve önceki nöron çıkışları (ilk adımda 0) kullanılarak ağı çıkışı elde edilir. Ağı çıkışı ile hata vektörü hesaplanır ve hesaplanan hata kullanılarak ağı ağırlıkları güncellenir.

```
# Eğitim kümesi bu problem için düğümlerden oluşur.
for node in nodeList:
    j = 0
    # Düğümlerden bir önceki düğümlerin değerleri, karesi ve ondan da bir önceki düğümün değeri çekilir.
    # Ağa girilecek veri bu şekilde atanır. u(k) = [y(k-2), y(k-1), y(k-1)^2]
    data = np.array([node.prev.prev.prev.prev.val,
                    node.prev.prev.prev.val,
                    node.prev.prev.val,
                    node.prev.val], dtype=float).reshape(dim, 1)
    # Ardından agdaki gizli katman nöron çıkışları (x(k)) bulunur.
    for neuron in network:
        x_j = neuron.forward(data, prevx)
        currx[j] = x_j
```

Ortalama hata eklenir, nöron çıkışları “context unit” olarak atanır ve sonraki veri düğümüne geçilir.

```
# Hata, ortalama hata için kaydedilir. Gizli katman çıkışı (x(k)) 'context unit' olarak
# kullanılmak üzere (x(k-1)) kaydedilir.
Eort += E
prevx = currx
```

## Test Aşaması:

Eğitim bittikten sonra test aşamasına geçilir. Bu aşamada eğitim ve test kümesi, eğitilen ağı ile “testNetwork” fonksiyonuna girilir. Eğitim kümesinin de test aşamasına girmesinin sebebi, karşılaştırma amacıyla ağıın eğitim kümesi verileriyle tahmini değerler oluşturmasıdır. Ayrıca test listesinin başlangıcındaki  $x(k-1)$  değeri, eğitim listesiyle olan testin son adımdan çekilecektir.

```
# Eğitilen ağı ile test yapılır. Test, hem eğitim hem test kümesi ile yapılmaktadır. Bunun sebebi
# veri düğümü içinde bulunan 'pred' değeridir. Veriler ve ağıın tahmini çizilirken bu değer kullanılmaktadır.
# Ağ test aşamasında herhangi bir veri için çıkışı tahmin etmektedir.
edresult = testNetwork(trainList, network)
result = testNetwork(testList, network, edresult[1])
print("x = 100-299 için test edildi ve ortalama yüzde %.7f hata elde edildi." % (100*result[0][0]))
network.clear()
# Ağ çıkışı, eğitim kümesi verileri ve test kümesi verileri çizdirilip 'Liste_Karistirildiginda' klasörüne kaydedilir.
plotcomparison(trainList, testList, maxval, minval, os.path.join(__location__, 'Liste_Karistirildiginda'))
plt.close()
```

## Soru 1: Lineer Regresyon

Test fonksiyonu eğitim fonksiyonuna oldukça benzemektedir. Yine ileri yol izlenip ağ çıkışı bulunur ve düğümlerde tahmin değerlerini belirten “pred”e atılır. Ağırlıklar güncellenmez. Elde edilen hata döndürülür. Başlangıçtaki  $x(k - 1)$  değeri eğer belirtilmemişse 0.5 olarak alınır.

```
def testNetwork(nodeList, network, prevx = 0):
    # Girilen bir veri kümesi (bu problem için düğüm kümesi) ile girilen ağ test edilir.
    # Test aşaması eğitimde ileri yol aşamasıyla aynıdır. Veriler ağa girilir ve çıkış elde edilir.
    # Elde edilen çıkışla hata hesaplanır. Hata tüm verilerde hesaplanıp bu hatanın ortalaması alınır.
    # Test esnasında ağıın ağırlıkları güncellenmez.

    # Test kümesi, eğitimden sonra geleceği için eğitimdeki son  $x(k)$  değerleri çekilir ve
    # testin başlangıcında  $x(k-1)$  olarak kullanılır.
    if not isinstance(prevx, np.ndarray):
        prevx = np.zeros((x_dim,1), dtype=float) + 0.5
    currx = np.zeros(x_dim, dtype=float).reshape(x_dim,1)
    Eort = 0
    for node in nodeList:
        j = 0
        data = np.array([node.prev.prev.prev.prev.val,
                        node.prev.prev.prev.prev.val,
                        node.prev.prev.prev.val,
                        node.prev.prev.val,
                        node.prev.val], dtype=float).reshape(dim, 1)
        for neuron in network:
            x_j = neuron.forward(data, prevx)
            currx[j] = x_j
            if j == 0:
                w_y = neuron.w_y
            else:
                w_y = np.concatenate((w_y, neuron.w_y), axis=1)
            j += 1
        curry = np.dot(w_y, currx)
        e = node.val - curry
        E = (e**2)/2
        Eort += E
        node.pred = curry
        prevx = currx
    Eort = (Eort/len(nodeList)).reshape(1)
    return Eort, prevx
```

## Verilerin Karşılaştırılması:

Gerçekte olan değerler ve ağıın test aşamasında tahmin ettiği değerler çizdirilerek karşılaştırılır.

```
# Ağ çıkışı, eğitim kümesi verileri ve test kümesi verileri çizdirilip 'Liste_Karistirildiginda' klasörüne kaydedilir.
plotcomparison(trainList, testList, maxval, minval, os.path.join(__location__, 'Liste_Karistirildiginda'))
```

## Soru 1: Lineer Regresyon

Bu karşılaştırmada eğitim ve test kümeleri için gerçek değerler ve tahmini değerler ayrı ayrı ve birlikte çizdirilir. Bu çizimler belirtilen bir klasörde kaydedilecektir.

```
# Ağın ve verinin karşılaştırıldığı grafiği çizen fonksiyon.
def plotcomparison(trainList, testList, maxval, minval, savefolder):
    # Ağlı, eğitim ve test kümelerini ayrı ayrı ve karşılaştırarak çizer.
    # Figürleri 'savefolder' adresine kaydeder.
    if not os.path.exists(savefolder):
        os.makedirs(savefolder)

    # Verileri çizerken her veri düğümü ve önce gelen veri düğümünü alır. Bu iki düğümün gerçek
    # değerleri arasında aralıklı bir dizi oluşturur. Bu şekilde figür çizilir.
    plt.figure()
    for node in trainList:...
        plt.title("Yaklaşılan fonksiyon (Eğitim kümesi)")
        plt.savefig(os.path.join(savefolder, 'Gerçek_Egitim_Kumesi.png'))

    # Ağın çiziminde de benzer bir şekilde iki ardışık düğümün tahmini değerleri arasında
    # aralıklı bir dizi oluşturularak çizdirilir. Bu tahmini değer eğitilmiş ağ ile test sonucu
    # elde edilir.
    plt.figure()
    for node in trainList:...
        plt.title("Tahmin edilen fonksiyon (Eğitim kümesi)")
        plt.savefig(os.path.join(savefolder, 'Tahmini_Egitim_Kumesi.png'))

    # İki çizim aynı anda uygulanıp karşılaştırma çizilir.
    plt.figure()
    for node in trainList:...
        plt.title("İki fonksiyonun karşılaştırması (Eğitim kümesi)")
        plt.savefig(os.path.join(savefolder, 'Egitim_Kumesi_Karsilastirma.png'))

    # Aynı işlemler test kümesi için de yapılır.
    plt.figure()
    for node in testList:...
        plt.title("Yaklaşılan fonksiyon (Test kümesi)")
        plt.savefig(os.path.join(savefolder, 'Gerçek_Test_Kumesi.png'))

    plt.figure()
    for node in testList:...
        plt.title("Tahmin edilen fonksiyon (Test kümesi)")
        plt.savefig(os.path.join(savefolder, 'Tahmini_Test_Kumesi.png'))

    plt.figure()
    for node in testList:...
        plt.title("İki fonksiyonun karşılaştırması (Test kümesi)")
        plt.savefig(os.path.join(savefolder, 'Test_Kumesi_Karsilastirma.png'))
    # plt.show()
```

## Soru 1: Lineer Regresyon

Düğümlerden önce gelen düğümlerin, gerçek ve tahmini değerleri arasında aralıklı diziler oluşturularak çizim yapılır. Düğümlerden çekilen değerler çizdirilmeden önce geri ölçekleme yapılır.

```
# Verileri çizerken her veri düğümü ve önce gelen veri düğümünü alır. Bu iki düğümün gerçek  
# değerleri arasında aralıklı bir dizi oluşturur. Bu şekilde figür çizilir.  
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)  
    y = y.reshape(100)  
    plt.plot(x, y, color = 'blue', label='Gerçek Değer')  
plt.title("Yaklaşılan fonksiyon (Eğitim kümesi)")  
plt.savefig(os.path.join(savefolder, 'Gercek_Egitim_Kumesi.png'))  
plt.close()
```

```
# Ağın çiziminde de benzer bir şekilde iki ardışık düğümün tahmini değerleri arasında  
# aralıklı bir dizi oluşturularak çizdirilir. Bu tahmini değer eğitilmiş ağ ile test sonucu  
# elde edilir.  
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  
    # Çizilen y değeri geri ölçeklendirilir. 'maxval' ve 'minval' veri ölçeklendirilirken  
    # verinin maksimum ve minimumlarının saklandığı değişkenlerdir ve geri ölçeklendirmede  
    # kullanılır.  
    y = minval + normalized_y*(maxval - minval)  
    y = y.reshape(100)  
    plt.plot(x, y, color = 'red')  
plt.title("Tahmin edilen fonksiyon (Eğitim kümesi)")  
plt.savefig(os.path.join(savefolder, 'Tahmini_Egitim_Kumesi.png'))  
plt.close()
```

```
# İki çizim aynı anda uygulanıp karşılaştırma çizilir.  
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)  
    y = y.reshape(100)  
    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)  
    y = y.reshape(100)  
    plt.plot(x, y, color = 'blue', label='Gerçekte olan')  
plt.title("İki fonksiyonun karşılaştırması (Eğitim kümesi)")  
plt.savefig(os.path.join(savefolder, 'Egitim_Kumesi_Karsilastirma.png'))  
plt.close()
```

## Soru 1: Lineer Regresyon

Ek olarak, durum portrelerinin çizimi için eğitim ve test kümelerindeki düğümlerden  $y(k)$  ve  $y(k - 1)$  değerleri, gerçek ve tahmini değerler için çekilipleri ile ölçekleme yapılarak listelerde toplanır.

```
# Ardından  $y(k)$  ve  $y(k-1)$  gerçek ve tahmini değerleri çekilir. Bu değerler kullanılarak
# durum portreleri çizdirilir ve kaydedilir.
# Veriler çekilişken geri ölçeklendirme uygulanmaktadır.

trainingList_x = []
trainingList_y = []
for node in trainList:
    trainingList_x.append(float(minval + node.prev.val*(maxval - minval)))
    trainingList_y.append(float(minval + node.val*(maxval - minval)))

trainingListPred_x = []
trainingListPred_y = []
for node in trainList:
    trainingListPred_x.append(float(minval + node.prev.pred*(maxval - minval)))
    trainingListPred_y.append(float(minval + node.pred*(maxval - minval)))

testingList_x = []
testingList_y = []
for node in testList:
    testingList_x.append(float(minval + node.prev.val*(maxval - minval)))
    testingList_y.append(float(minval + node.val*(maxval - minval)))

testingListPred_x = []
testingListPred_y = []
for node in trainList:
    testingListPred_x.append(float(minval + node.prev.pred*(maxval - minval)))
    testingListPred_y.append(float(minval + node.pred*(maxval - minval)))
```

Daha sonra bu listeler ile durum portreleri çizilir.

```
plt.figure()
plt.plot(trainingListPred_x, trainingListPred_y, color = 'red')
plt.title("Tahmini değerlerin durum portesi (Eğitim kümesi)")
plt.savefig(os.path.join(savefolder, 'Egitim_Kumesi_Tahmini_Durum_Portresi.png'))
plt.close()

plt.figure()
plt.plot(trainingList_x, trainingList_y, color = 'blue')
plt.title("Gerçek değerlerin durum portesi (Eğitim kümesi)")
plt.savefig(os.path.join(savefolder, 'Egitim_Kumesi_Gercek_Durum_Portresi.png'))
plt.close()

plt.figure()
plt.plot(trainingListPred_x, trainingListPred_y, color = 'red', label='Tahmin edilen')
plt.plot(trainingList_x, trainingList_y, color = 'blue', label='Gerçekte olan')
plt.title("Tahmini ve gerçek değerlerin durum portrelerinin karşılaştırılması (Eğitim kümesi)")
plt.savefig(os.path.join(savefolder, 'Egitim_Kumesi_Durum_Portresi_Karsilastirma.png'))
plt.close()
```

## Soru 1: Lineer Regresyon

```
plt.figure()
plt.plot(testingListPred_x, testingListPred_y, color = 'red')
plt.title("Tahmini değerlerin durum portesi (Test kümesi)")
plt.savefig(os.path.join(savefolder, 'Test_Kumesi_Tahmini_Durum_Portresi.png'))
plt.close()

plt.figure()
plt.plot(testingList_x, testingList_y, color = 'blue', label='Gerçekte olan')
plt.title("Gerçek değerlerin durum portesi (Test kümesi)")
plt.savefig(os.path.join(savefolder, 'Test_Kumesi_Gercek_Durum_Portresi.png'))
plt.close()

plt.figure()
plt.plot(testingListPred_x, testingListPred_y, color = 'red', label='Tahmin edilen')
plt.plot(testingList_x, testingList_y, color = 'blue', label='Gerçekte olan')
plt.title("Tahmini ve gerçek değerlerin durum portrelerinin karşılaştırılması (Test kümesi)")
plt.savefig(os.path.join(savefolder, 'Test_Kumesi_Durum_Portresi_Karsilastirma.png'))
plt.close()
```

## Eğitim ve Test Sonucu:

Eğitim kümesi, her iterasyonun başında karıştırıldığında aşağıdaki sonuç elde edilir.

```
Eğitim süresi: 67.924 saniye
Eğitim 1000 adımada yüzde 3.5139527 hata oranı ile tamamlandı
x = 100-299 için test edildi ve ortalama yüzde 3.5133917 hata elde edildi.
```

Kümenin oluşturulduğu sırayla ağa verildiğinde elde edilen sonuç ise aşağıdaki gibidir.

```
Eğitim süresi: 67.541 saniye
Eğitim 1000 adımada yüzde 0.0838765 hata oranı ile tamamlandı.
x = 100-299 için test edildi ve ortalama yüzde 0.0443365 hata elde edildi.
```

Görüldüğü üzere eğitimdeki verilerin karıştırılmaması Elman Ağrı'nı oldukça iyi bir şekilde etkilemiştir. Eğitim süresi iki durumda da hemen hemen aynıdır. Test aşamasında elde edilen hata ise ilk durumda %3.513 civarındayken ikinci durumda %0.044'e kadar düşmüştür. Eğitim kümelerinin sırasının Elman Ağrı için önemli olduğu gözlemlenir.

Gözlemlenen bir başka durum ise eğitimin maksimum iterasyon sayısı seçilen 1000 iterasyonda sonlanmasıdır. Daha sağlıklı testler yapılabilmesi için maksimum iterasyon sayısı 3000'e çıkarılır.

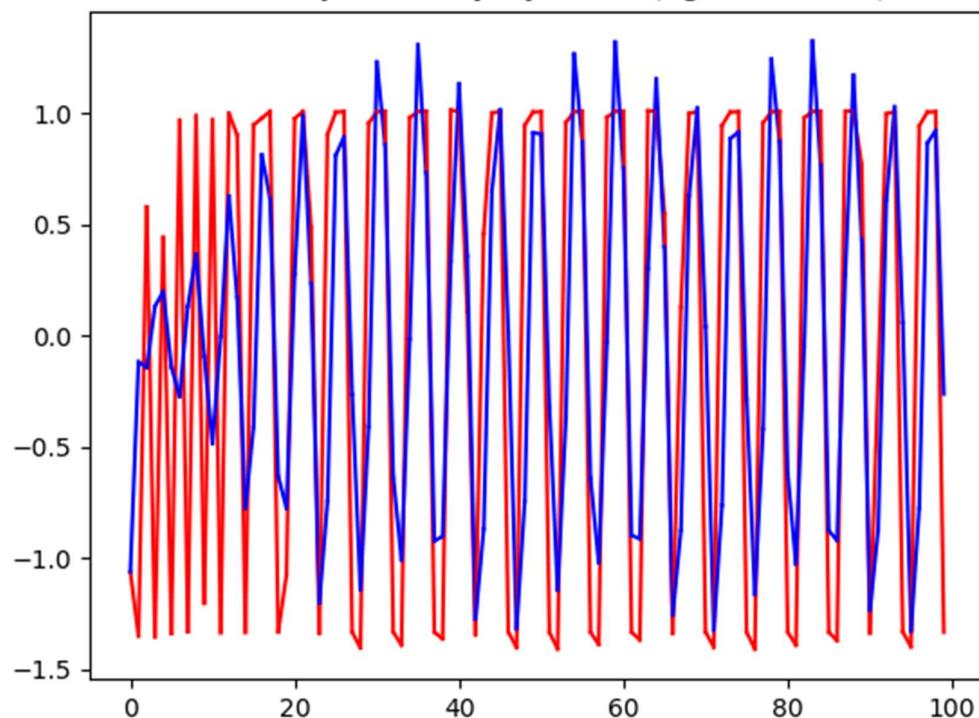
## Soru 1: Lineer Regresyon

### Elde Edilen Figürler:

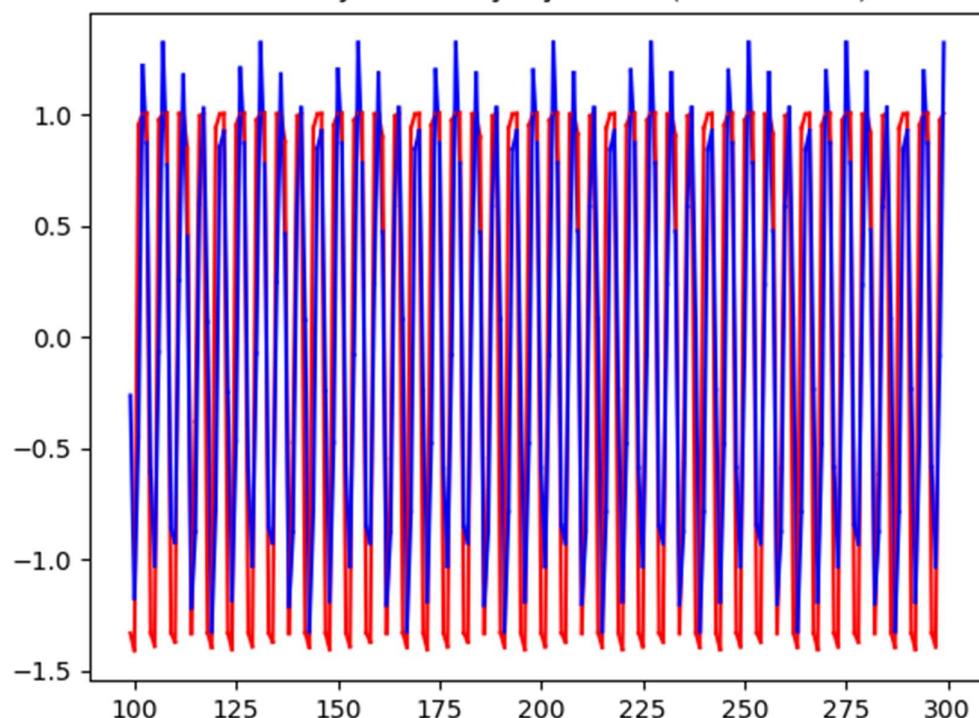
Bu bölümde ağın başarısını incelemek için oluşturulan figürler incelenecaktır. Elde edilen figürlerde kırmızı renk tahmin edilen, ağ çıkışındaki sonuçları temsil eder. Mavi renk ise fonksiyon çıkışında elde edilen gerçek değerleri temsil eder.

Öncelikle, başlangıçtaki liste karşılaştırıldığında eğitilen ağın sonuçları şu şekildedir:

İki fonksiyonun karşılaştırması (Eğitim kümlesi)



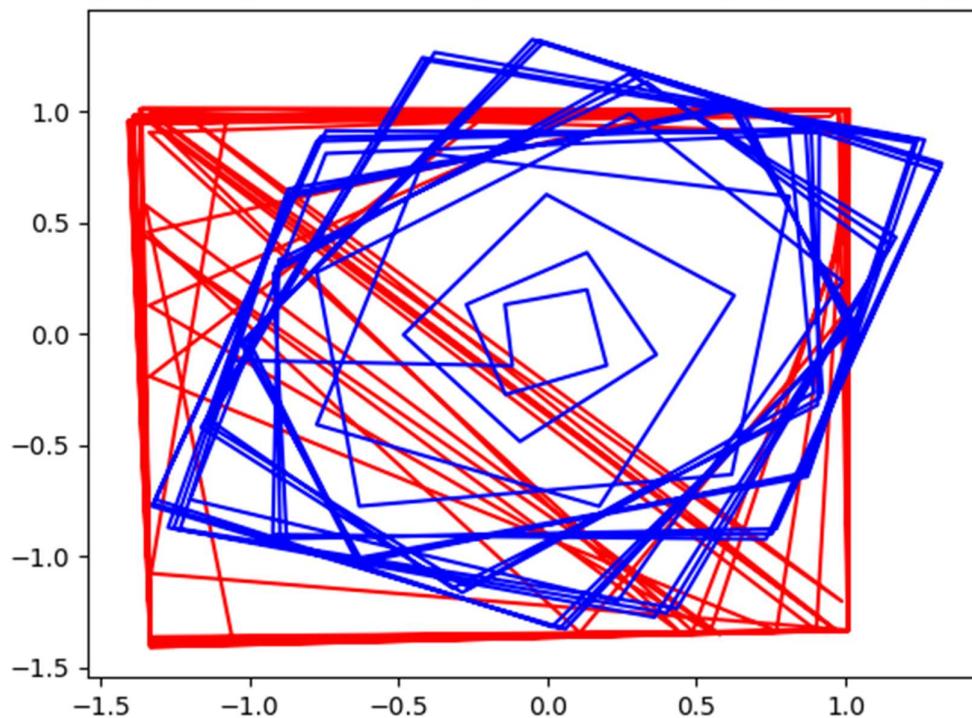
İki fonksiyonun karşılaştırması (Test kümlesi)



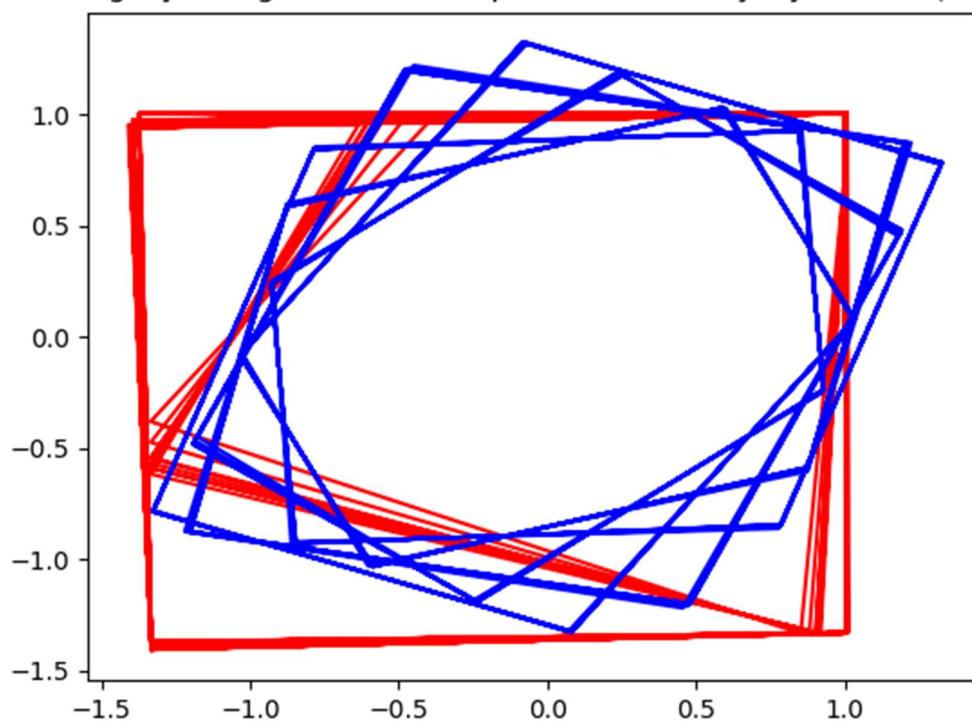
## Soru 1: Lineer Regresyon

Göründüğü üzere, liste karıştırıldığında ağ eğitim kümelerine yaklaşamamış, test kümelerindeki verileri de oldukça yanlış bir şekilde tahmin etmiştir. Bu durum, durum portresine bakıldığında daha net bir şekilde gözlemlenir.

İhmini ve gerçek değerlerin durum portrelerinin karşılaştırılması (Eğitim küm



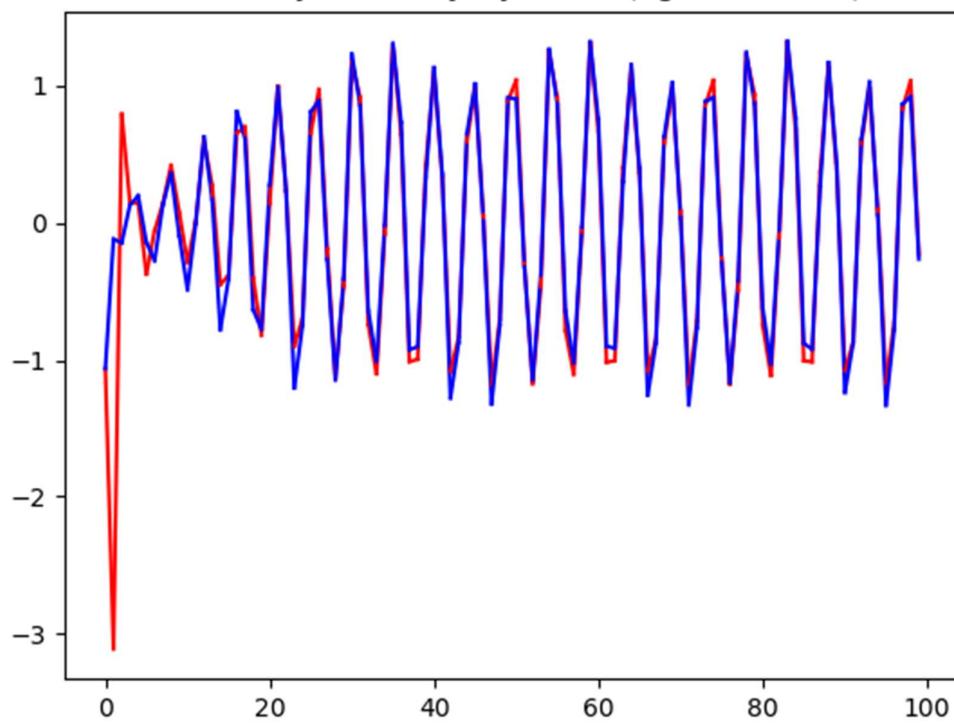
Tahmini ve gerçek değerlerin durum portrelerinin karşılaştırılması (Test küme



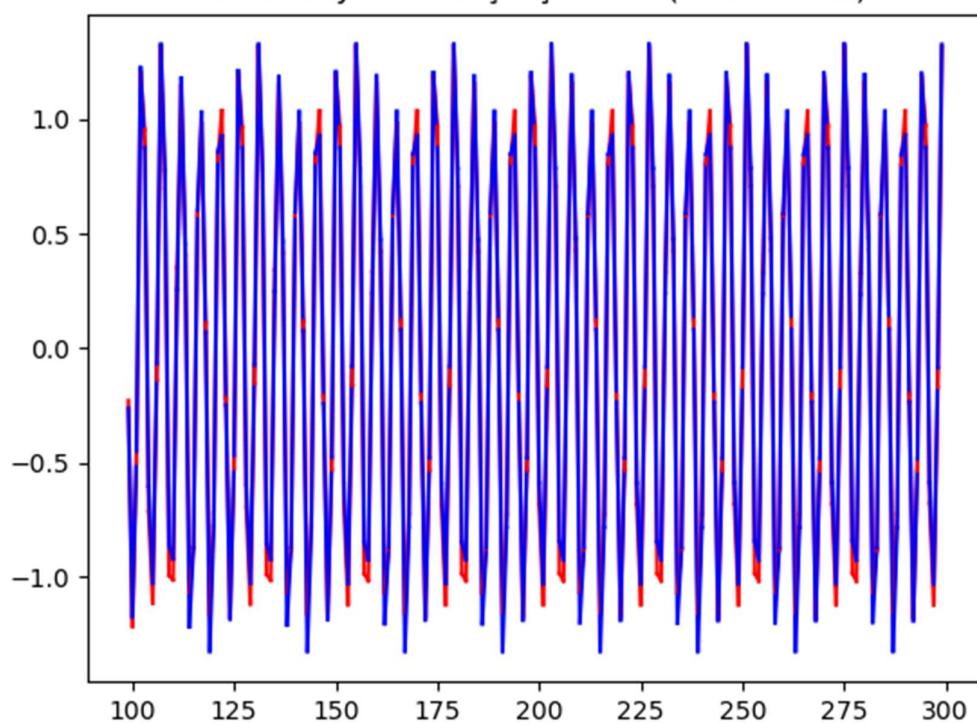
### Soru 1: Lineer Regresyon

Şekillerden de görüldüğü gibi, ağ sistem davranışını temsil edememektedir. Eğitim sırasında verilerin karıştırılmadığı durum ise aşağıdaki gibidir.

İki fonksiyonun karşılaştırması (Eğitim kümesi)



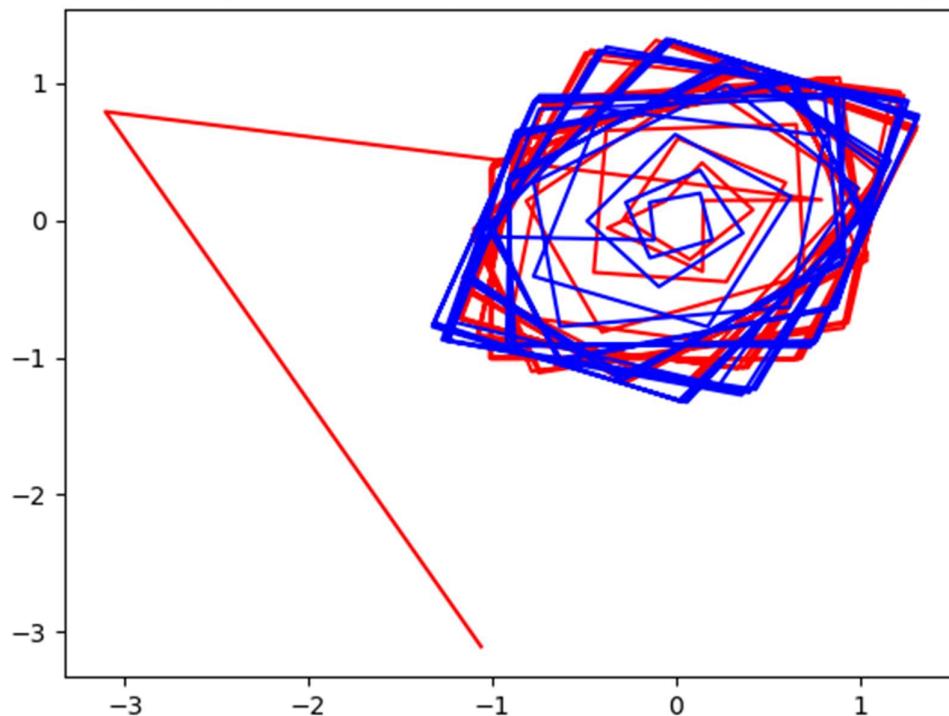
İki fonksiyonun karşılaştırması (Test kümesi)



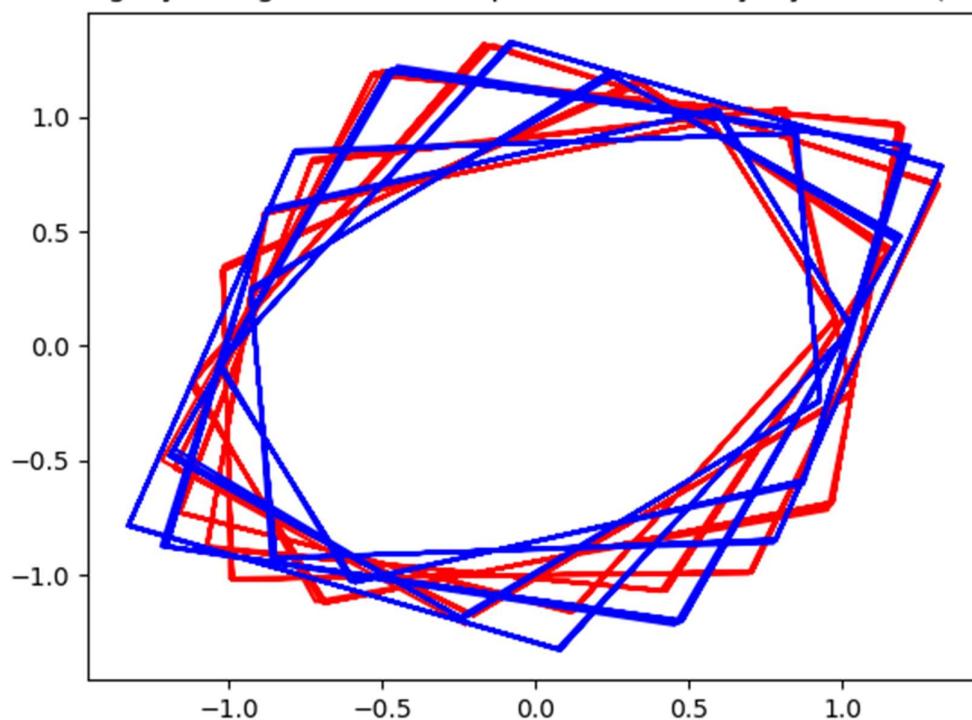
### Soru 1: Lineer Regresyon

Gördüğü gibi, eğitim kümesiyle karşılaştırıldığında başlangıçtaki aykırı davranış dışında sisteme oldukça yakındır. Eğitim başarılı bir şekilde tamamlanmış, ağ sistemi temsil edebilmektedir. Ağın, test kümesindeki verilerin de yakın bir şekilde tahmin ettiği görülür. Bazı üç noktalarda ayrınlık gösterse de iki şekil arasında çok büyük bir farklılık yoktur. Durum portreleri de ağın başarısını göstermektedir.

Tahmini ve gerçek değerlerin durum portrelerinin karşılaştırılması (Eğitim küm



Tahmini ve gerçek değerlerin durum portrelerinin karşılaştırılması (Test küm

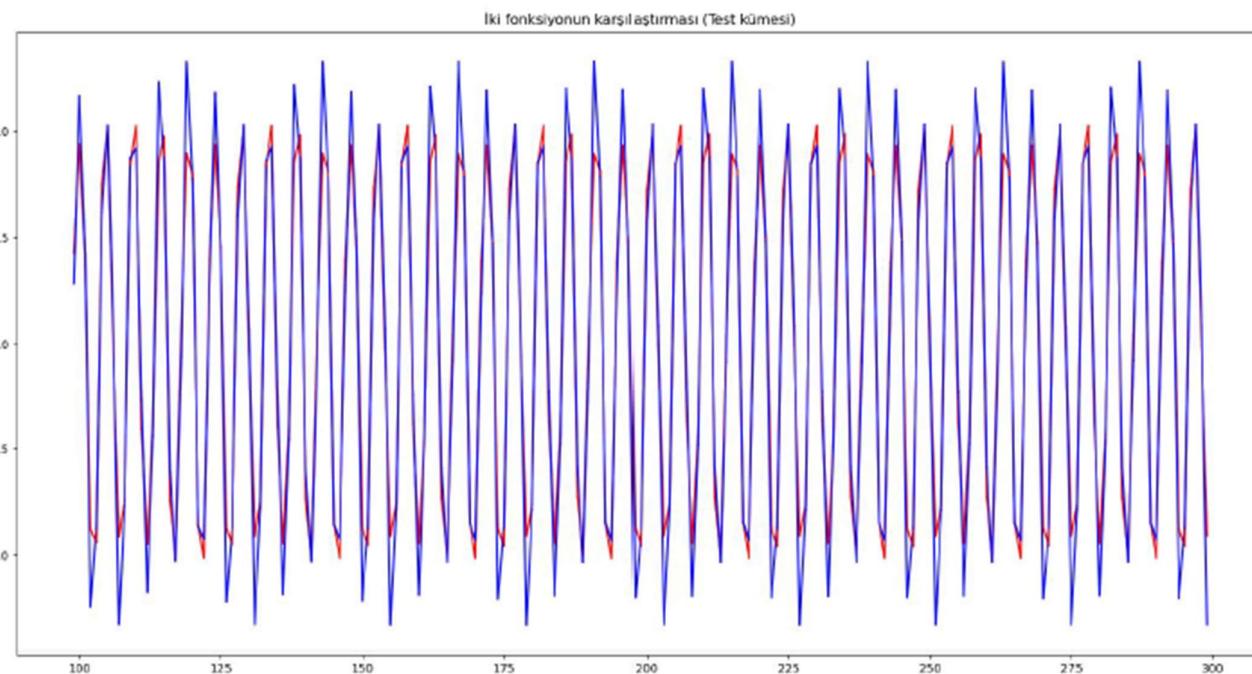
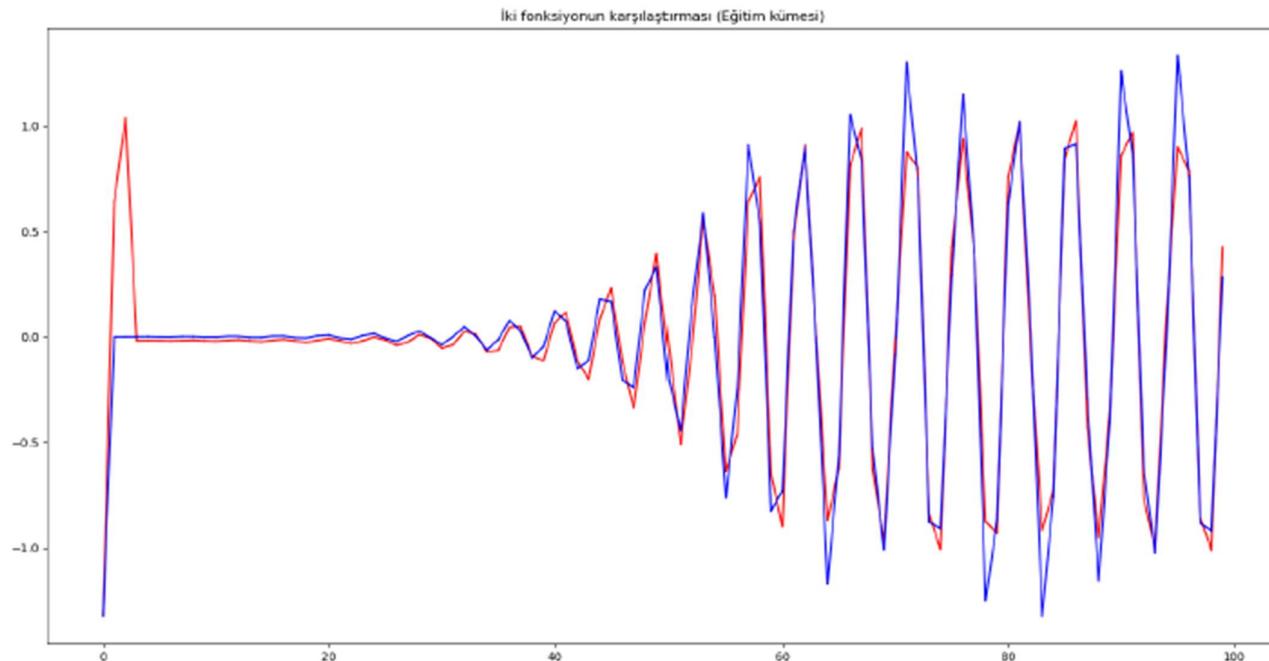


## Soru 1: Lineer Regresyon

Göründüğü gibi, başlangıçtaki ayrıklık dışında ağıın ve sistemin durum portreleri birbirine oldukça yakındır. Yakın davranışları sergilerler fakat ağıın, sistemin bir miktar ötelenmiş bir durum portresi sergilediği de görülmektedir.

### Çok Katmanlı Algılayıcı Karşılaştırması:

Bu probleme daha önce çok katmanlı algılayıcı ile de yaklaşılmıştır. Çok katmanlı algılayıcı kullanıldığında elde edilen ağı sonuçları şu şekilde dir:



## Soru 1: Lineer Regresyon

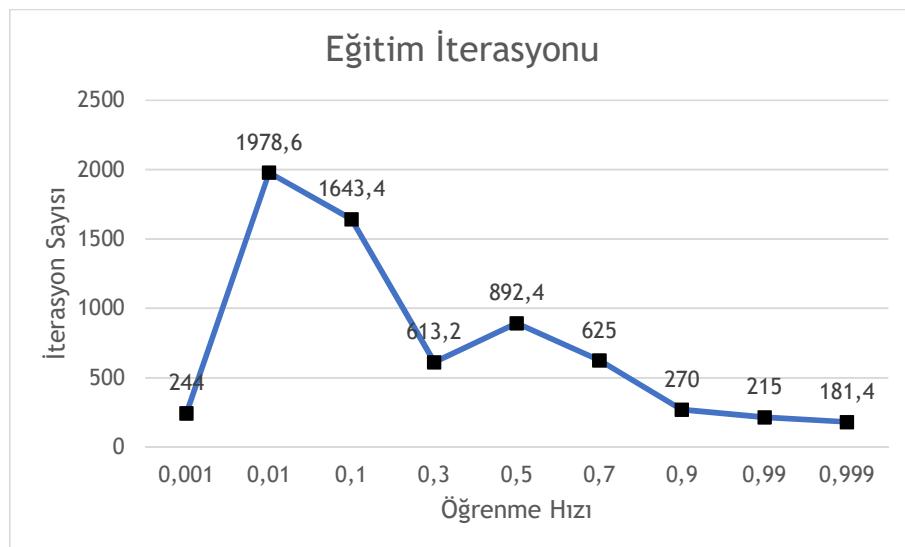
Elman Ağının elde ettiği sonuçla karşılaştırıldığında, iki ağın da sisteme benzer bir başarıyla yaklaşığı görülmektedir. Fakat çok katmanlı algılayıcı kullanıldığında test sonucunda yaklaşık %0.32 hata elde edilmişken aynı parametrelerle kurulan Elman Ağısı %0.044 hata elde edilerek, Elman Ağısının bu problem için daha uygun bir yapı olduğu görülür.

### Farklı Öğrenme Hızları:

Kurulan ağ, farklı öğrenme hızları kullanılarak eğitim süresi, eğitim iterasyon sayısı ve hat oranı için test edilir. Maksimum iterasyon sayısı 3000, momentum 0.9 ve hata limiti 0.001 kullanılacaktır. Elde edilen sonuçlar tablodaki gibidir.

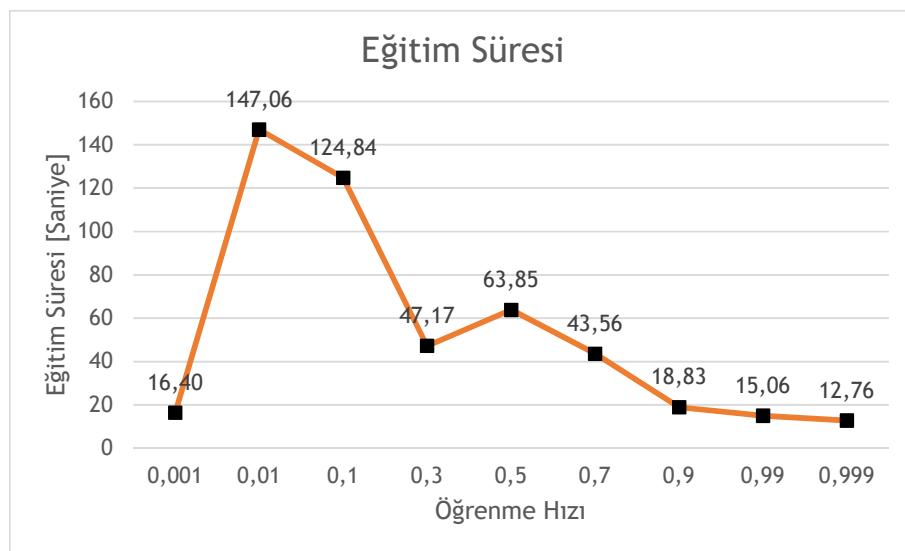
Öğrenme Hızı	0,001	0,01	0,1	0,3	0,5	0,7	0,9	0,99	0,999
Eğitim İterasyonu	244	1978,6	1643,4	613,2	892,4	625	270	215	181,4
Eğitim Süresi	16,40236	147,0553	124,8397	47,17149	63,84811	43,5614	18,8339	15,05942	12,76322
Hata Oranı	0,001187	0,000674	0,000721	0,000284	0,000248	0,000716	0,001239	0,03775	0,002354

Öncelikle, eğitim iterasyonu sayısı incelendiğinde 0.01 ve 0.1 öğrenme hızlarındaki testlerin diğer testlerden çok daha yüksek iterasyonlara ulaşlığı görülür. Çok düşük ve çok yüksek öğrenme hızlarında iterasyon sayısı en düşüktür.

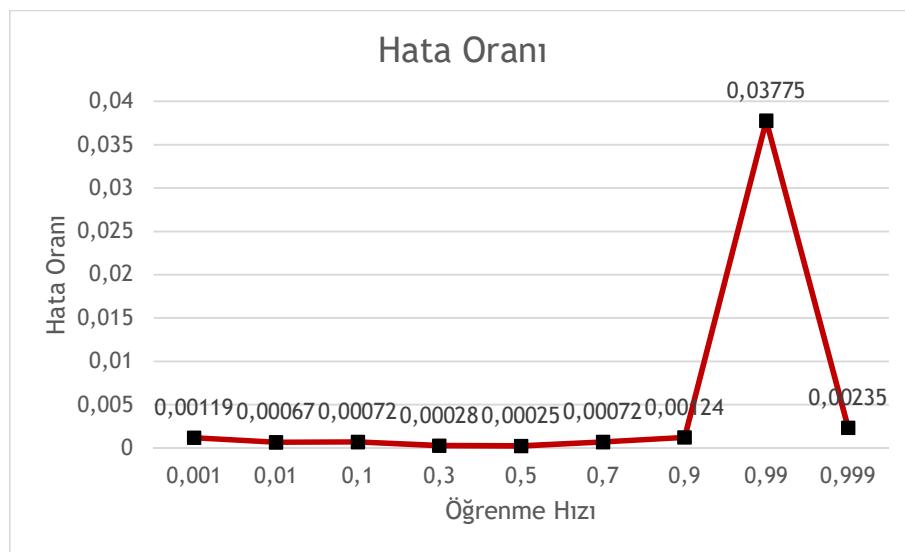


## Soru 1: Lineer Regresyon

Bu test için, testler arasında ağ yapısında bir değişiklik olmadığı için eğitim süresi, iterasyon sayısıyla aynı şekilde hareket etmektedir. Eğitim ortalama 50 saniye civarında gerçekleşmekte olup, en hızlı 0.999 öğrenme hızıyla 12.76 saniyede gerçekleşmiş ve en yavaş 147.06 saniyeye 0.01 öğrenme hızıyla gerçekleşmiştir.



Hata oranları karşılaştırıldığında ise 0.99 öğrenme hızının diğerlerine göre çok daha yüksek bir hata oranına sahip olduğu görülür. Eğitim süresi en kısa olan diğer iki test, 0.001 ve 0.999 öğrenme hızları, 0.03775 kadar yüksek bir değer olmasa da diğer testlere göre kötü bir performans sergilemiştir. 0.3 ve 0.5 öğrenme hızlarının bu problemde eğitim süresi ve hata oranı açısından iyi performans sergilediği görülmektedir.



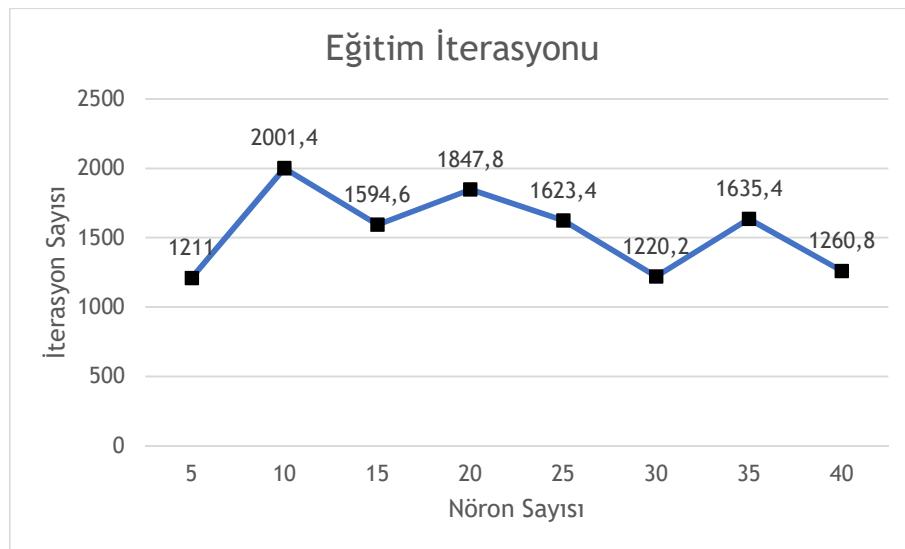
## Soru 1: Lineer Regresyon

### Farklı Nöron Sayıları:

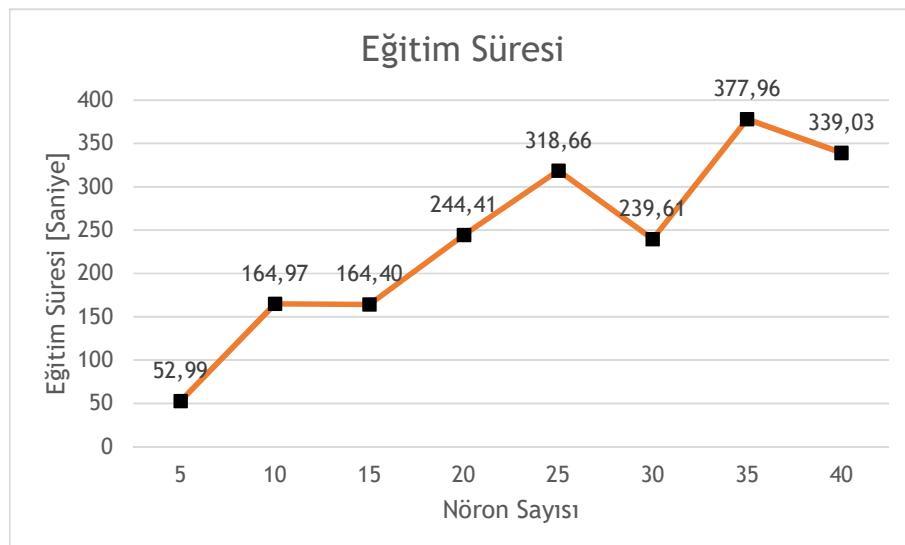
Kurulan ağ, farklı gizli katman nöron sayıları kullanılarak eğitim süresi, eğitim iterasyon sayısı ve hat oranı için test edilir. Maksimum iterasyon sayısı 3000, momentum 0.9 ve hata limiti 0.001 kullanılacaktır. Sonuçlar tablodaki gibidir.

Gizli Katman Nöron Sayısı	5	10	15	20	25	30	35	40
Eğitim Iterasyonu	1211	2001,4	1594,6	1847,8	1623,4	1220,2	1635,4	1260,8
Eğitim Süresi	52,99369	164,9695	164,4044	244,414	318,6643	239,6137	377,9592	339,0323
Hata Oranı	0,000462	0,000131	0,000759	0,000698	0,000672	0,00231	0,000677	0,001021

Eğitim iterasyonları karşılaştırıldığında düzenl bir değişim gözlenmez.

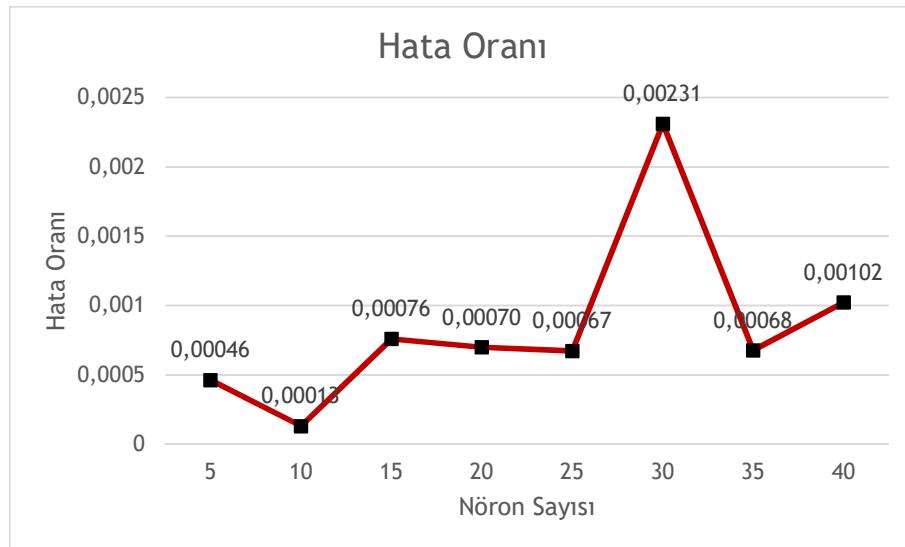


Fakat eğitim süreleri, farklı testler yapılırken ağ yapısı değiştiği için iterasyon sayısıyla aynı davranışı sergilememektedir. Aynı değerler etrafında değişen iterasyon sayılarına rağmen, her teste artan nöron sayısı eğitim süresinin de artmasına sebep olur. En düşük eğitim süresi 5 nöron kullanıldığı testteki 52.99 saniyeyken bu süre 35 nöron kullanıldığından 377.96 saniyeye kadar yükselmiştir.



## Soru 1: Lineer Regresyon

Testlerdeki hata oranları karşılaştırıldığında, yüksek nöron sayısının çoğunlukla daha isabetli bir eğitime yol açmadığı görülür. 15 nöron ve sonrasındaki testlerde hata oranı yüksek kalmış, 30 nöronda 0.00231'e kadar yükselmiştir. Büyüklük bir farkla en isabetli ağ yapısının 0.00013 hata oranı ile 10 nöron kullanılan ağ olduğu görülür.

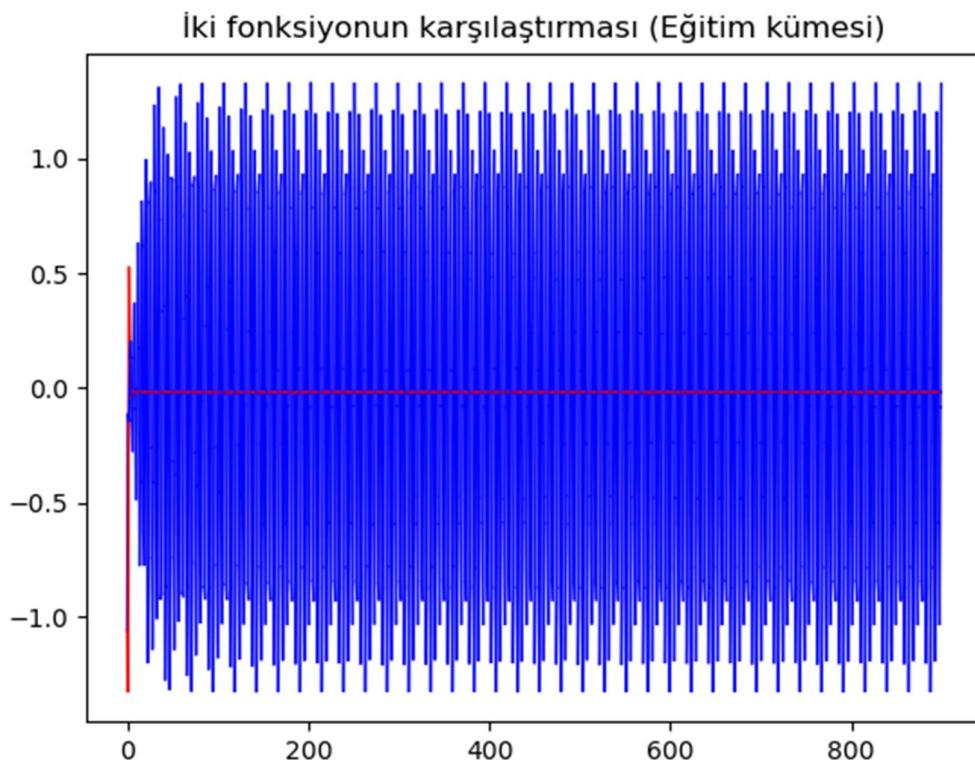


## Soru 1: Lineer Regresyon

### Girişe e(k) Verilmesi:

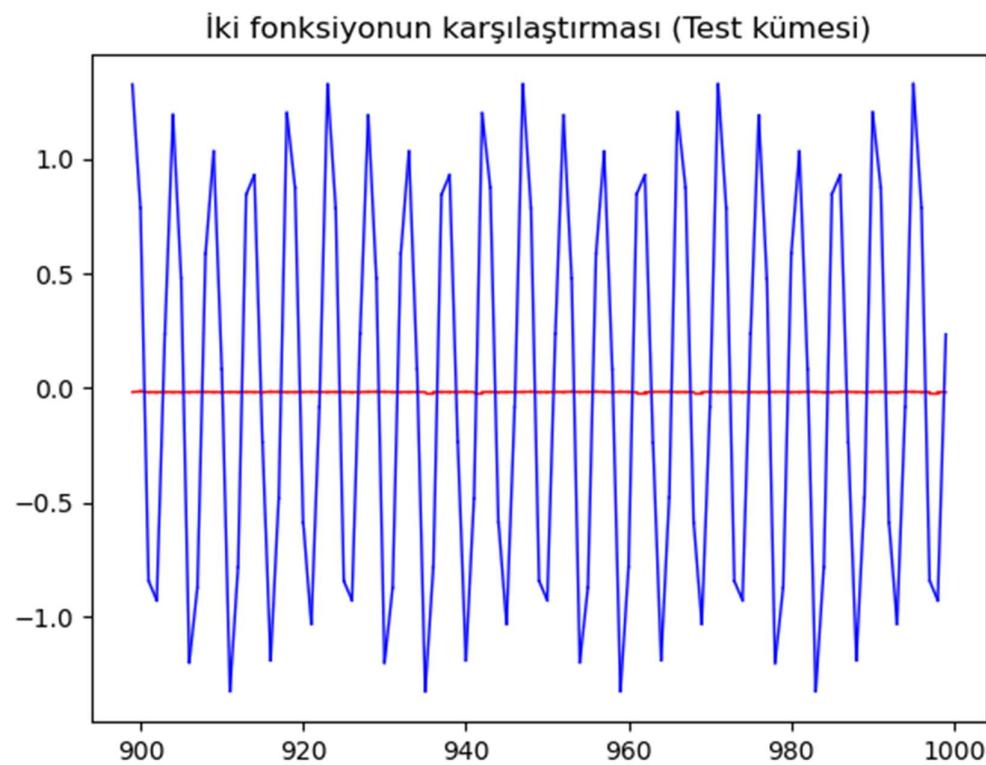
Ağ girişi olan  $u(k)$ , yalnızca beyaz gürültü olduğunda ağ çalışmamaktadır. Bu yapının denendiği kod, önceki kodla aynı olup yalnızca eğitim ve test aşamalarında data için önceki veriler yerine `np.random.normal(0,0.1)` şeklinde 0 medyan ve 0.1 standart sapmalı bir beyaz gürültü kullanılmıştır. Bu kod “hw4test.py” olarak klasörde bulunmaktadır. Başlangıç ağırlıkları  $[-0.5, 0.5]$  aralığına çekilmiş, eğitim kümesi ilk 900 veriden ve test kümesi sonraki 100 veriden oluşturulmuştur. Verilerin başlangıç değeri de önceki ağıdaki gibi 0.1 alınmıştır. Bu değerler okuma önerilerinde bulunan ve bu ağı anlatan Pham\_96 makalesinden tavsiyelerle seçilmiştir.

Ağ çalıştırıldığında, 25 deneme içinden başarılı olan hiçbir yapı olmamış, hepsi benzer özellikler göstermiştir. Farklı denemelerin sonuçları “Test” klasöründe bulunmaktadır. Herhangi bir ağın eğitimi sonrası alınan sonuç şekildeki gibidir.



### Soru 1: Lineer Regresyon

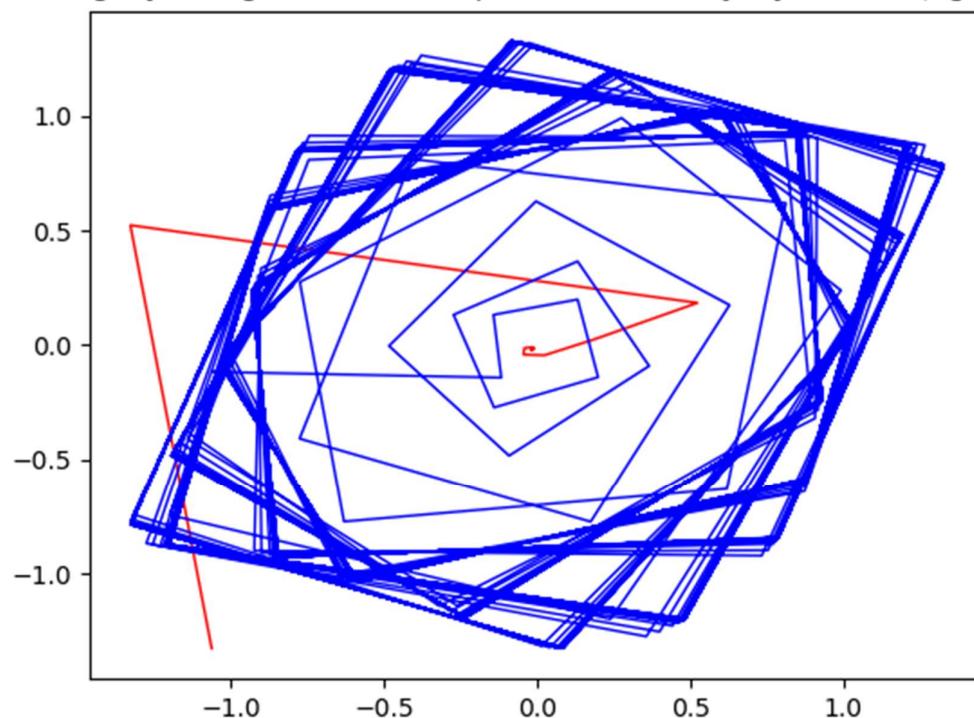
Görüldüğü gibi ağ çıkışı hızlı bir şekilde 0'a yakınsayıp, ağa benzer bir özellik göstermemiştir. Bu durum test kümesiyle karşılaştırıldığında belirgindir.



### Soru 1: Lineer Regresyon

Durum portreleri incelendiğinde de ağıın hızlı bir şekilde bir noktaya yakınsadığı görülür.

Tahmini ve gerçek değerlerin durum portrelerinin karşılaştırılması (Eğitim küm



Tahmini ve gerçek değerlerin durum portrelerinin karşılaştırılması (Test küm

