Yapay Sinir Ağları
5.Ödev

21.01.2021

Mehmet Şerbetçioğlu -- 040160056

Ahmet Hulusi Tarhan -- 040170738

Soru 1: Pekiştirmeli Öğrenme – Araba-Çubuk Sistemi ve Ağ Yapısı

Hafif rüzgarlı bir ortamda, düz bir yol üzerinde bir araba ve arabanın üstünde bir çubuk bulunmaktadır. Arabanın ilerisinde ve arkasında duvar bulunmaktadır. Arabaya kuvvet uygulanarak arabanın duvara çarpmaması ve arabanın üzerindeki çubuğun düşmemesi istenmektedir. Amaç, farklı zamanlarda bu kuvvetin ne yönde uygulanacağına karar verecek bir ağ üretilmesidir.

Bu problemde kullanılacak ağ, pekiştirmeli öğrenme uygulayacaktır. Pekiştirmeli öğrenmede bir sistem, ortamdan aldığı cevap ve bulunduğu duruma göre durumu değiştirecek kararlar verir. Bu sırada ortamdan alacağı cevabı tahmin ederek, istediği cevabı almak için kararlar verir.

Kurulacak sistemin ve ortamın detayları Tez_Kuyumcu.pdf dosyasında bulunmaktadır. Arabanın konumu, hızı, ivmesi ve çubuğun açısı, açısal hızı ve açısal ivmesine dair fiziksel eşitlikler de bu dosyadadır. Bunlara ek olarak, ortam koşulları, makalede kullanılan değerlerle alınacaktır.

Fiziksel Eşitlikler:

Ortamın kurulmasında kullanılan, durum değişkenlerini etkileyecek olan fiziksel eşitlikler şu şekildedir:

$$\ddot{\theta}t = \frac{g \sin\theta t + \cos\theta t \left[\frac{-F - m\, l\, \dot{\theta}t^2 \sin\theta t + \mu c\, sgn(\dot{x}t)}{mc + m}\right] - \frac{\mu p\, \dot{\theta}t}{m\, l}}{l\left[\frac{4}{3} - \frac{m\cos^2\theta t}{mc + m}\right]}$$

$$\ddot{x}t = \frac{F + m\, l\left[\dot{\theta}t + \sin\theta t - \ddot{\theta}t\cos\theta t\right] - \mu c\, sgn(\dot{x}t)}{mc + m}$$

$$m1 = mc + m$$

$$f1 = m\, l\, \dot{\theta}t^2 \sin\theta t - \mu c\, sgn\,(\dot{x}t)$$

$$f2 = g\sin\theta t + \cos\theta t\left[\frac{-F - f1}{m1}\right] - \frac{\mu p\, \dot{\theta}t}{m\, l}$$

Bu diferansiyel denklemler açık euler ile çözülecek olup, şu hali almaktadırlar:

$$\theta1(k+1) = \theta1(k) + \beta\,[\theta2(k)]$$

$$\theta2(k+1) = \theta2(k) + \beta\left[\frac{f2(k)}{l\left[\frac{4}{3} - \frac{m\cos^2\theta1(k)}{m1}\right]}\right]$$

Soru 1: Pekiştirmeli Öğrenme – Araba-Çubuk Sistemi ve Ağ Yapısı

$$x1(k+1) = x1(k) + \beta\,[x2(k)]$$

$$x2(k+1) = x2(k) + \beta\left[\frac{F(k) + (f1(k) - m\,l\,f2(k)\cos\theta1(k))}{m1}\right]$$

Burada θ_1 çubuk açısı, θ_2 çubuk açısının hızı, x_1 arabanın konumu ve x_2 arabanın hızıdır. ß değeri ise açık euler için kullanılan adım boyutu olup bu sistem için 0.005 alınmıştır.

Fiziksel parametreler ise şu şekilde alınmıştır:

$g = -9.8\,m/s^2$, $m_c = 1\,kg$, $m = 0.1\,kg$, $l = 0.5\,m$, $\mu_c = 0.0005$, $\mu = 0.000002$

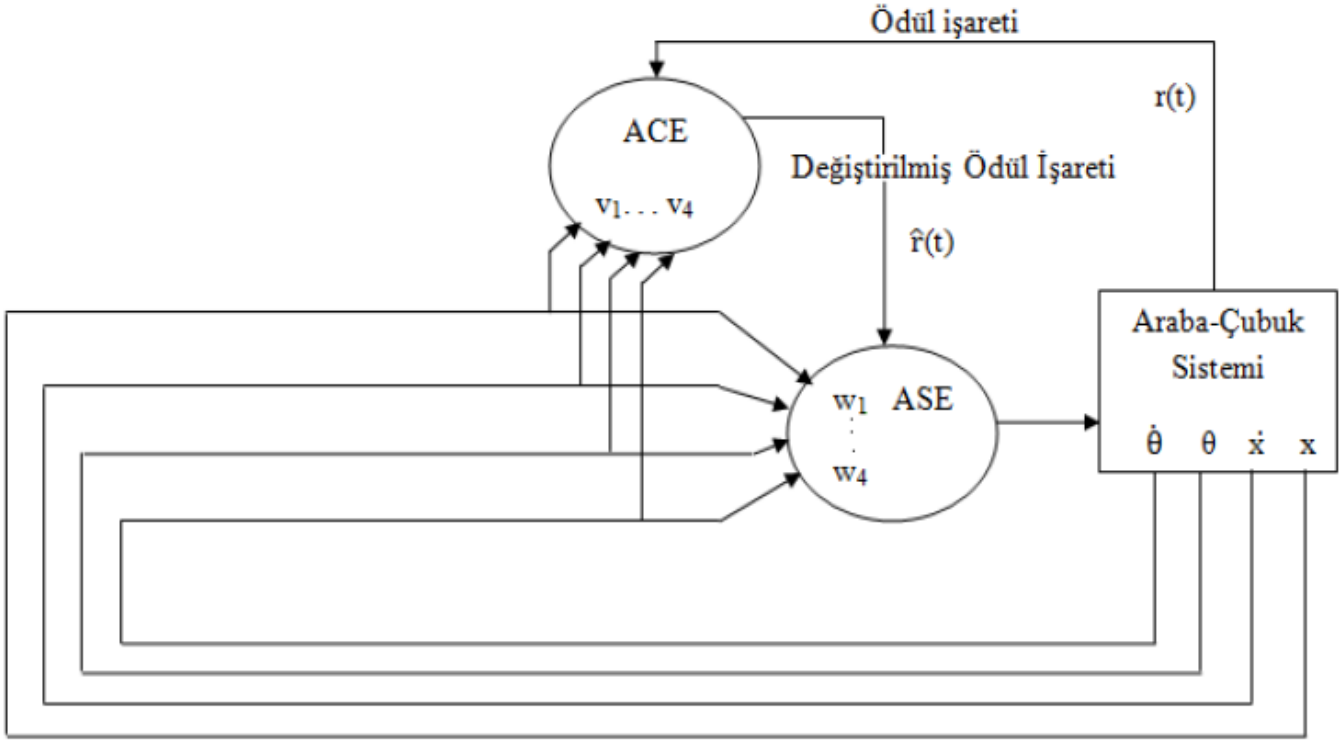
Burada g yerçekimi ivmesi, m_c araba kütlesi, m çubuk kütlesi, l çubuk uzunluğu, μ_c araba ve yer arasındaki sürtünme katsayısı ve μ çubuk ve araba arasındaki sürtünme katsayısıdır. Ayrıca, arabaya uygulanacak $F(k)$ kuvveti, ödül beklentisine göre $+10\,N$ veya $-10\,N$ olacaktır.

Ağ Yapısı:

Ağ temel olarak iki parçadan oluşur. Birinci parça uyarlanabilir eleştiri bileşenidir (ACE). Bu bileşen ortamdan alınan tepki ve sistemin durumuna göre ödül beklentisi değerini oluşturur. Bu değeri ikinci parça olan uyarlanabilir karar bileşenine (ASE) verir.

Uyarlanabilir karar bileşeni sistemin davranışını belirler. Bu kararı belirlerken ödül beklentisini kullanır. Ödül alabildiği durum doğrultusunda karar vermeye çalışır.

Ağ yapısı, Kuyumcu'nun tezinde şu şekilde ifade edilmiştir:



Soru 1: Pekiştirmeli Öğrenme – Araba-Çubuk Sistemi ve Ağ Yapısı

Uyarlanabilir Eleştiri Bileşeni:

Bu bileşenin amacı ödül beklentisini oluşturmaktır. Bu sırada şu eşitlikleri kullanır:

$$p(t) = [\, v1 \quad v2 \cdots \cdots \cdots vn \,] \begin{bmatrix} x1 \\ \cdot \\ \cdot \\ xn \end{bmatrix}$$

$$\hat{r}(t) = r(t) + \gamma\, p(t) - p(t\text{-}1)$$

Burada v vektörü bileşenin ağırlıkları, x vektörü ise girişindeki durum değişkenleridir. Ödül beklentisinin elde edildiği denklemdeki γ değişkeni, eleştiri bileşeninin yaptığı tahminin bir önceki tahminle benzerliğini gösterir. $0 \leq \gamma < 1$ olup, makaledeki sistemde $\gamma = 0.95$ değeri kullanılmıştır.

Eleştiri bileşeninin vektörleri şu şekilde güncellenmektedir:

$$v(t+1) = v(t) + \beta\, [\, r(t) + \gamma\, p(t) - p(t\text{-}1) \,]\, \bar{x}(t)$$

$$\bar{x}(t+1) = \lambda\, \bar{x}(t) + (1 - \lambda)\, x(t)$$

Burada $\bar{x}$ terimi giriş işaretinin seçilebilirliğini göstermektedir. $\bar{x}$ teriminin güncellenmesinde kullanılan λ ise bozulma oranı olup $0 \leq \lambda < 1$ koşullarına sahiptir. Makaledeki sistemde $\lambda = 0.8$ kullanılmıştır.

Uyarlanabilir Karar Bileşeni:

Ödül beklentisi ve sistem durumuna göre, sisteme uygulanacak kuvvetin yönüne karar verir. Bu işlem yapılırken kullanılan denklemler şu şekildedir:

$$y = f\,[w^T(t) + x(t) + n(t)\,] \qquad y = f(x)$$

Buradaki $f(x)$ fonksiyonu aktivasyon fonksiyonudur. Makalede Bu fonksiyon için $sgn(-x)$ kullanılmış olup, kurulacak sistemde çoğunlukla $tanh(-x)$ olmak üzere, $tanh(-x)$ ve $sgn(-x)$ kullanılacaktır.

Aktivasyon fonksiyonu argümanları içinde bulunan $n(t)$ ise gürültü olup sistemin durgun kalmasına engel olur. Kurulacak sistemde gürültü, $N(0,0.1)$ dağılımlı rastgele bir değer alır.

Soru 1: Pekiştirmeli Öğrenme – Araba-Çubuk Sistemi ve Ağ Yapısı

Karar bileşeni ağırlıkları ise şu şekilde güncellenmektedir:

$$w(t+1) = w(t) + \alpha\, \hat{r}(t)\, e(t)$$

$$e(t+1) = \delta\, e(t) + (1-\delta)\, y(t)\, x(t)$$

Ağırlıkların güncellendiği denklemdeki α değeri ağırlıkların değişme oranını gösterir. $e(t)$ terimi davranışlar ve sistemden gelecek hata işareti arasındaki uygunluk derecesini göstermektedir. Bu terimin güncellenmesinde kullanılan δ değeri ise bozulma sabitidir ve $0 \leq \delta < 1$ ilişkisine sahiptir.

Karar bileşeni çıkışı, sistemde uygulanacak kuvvetin yönünü belirler. Bu ilişki;

$y \geq 0 \Longrightarrow F = 10$

$y < 0 \Longrightarrow F = -10$

şeklinde ifade edilir.

Soru 1: Pekiştirmeli Öğrenme – Sistemin Python'da Kurulumu

Sistemin Kurulumu:

Öncelikle, iki bileşen ve ortam için üç obje oluşturulur.

```
class ACE():···

class ASE():···

class CarStick():···
```

ACE:

Uyarlanabilir eleştiri bileşeni (ACE) ilk başlatıldığında giriş verisinin (durum değişkenleri) boyutu, λ ve γ değerlerini atar. Ek olarak, açık euler çözümünde kullanılacak adım büyüklüğü ß atanır. Değerler atandıktan sonra sistem ağırlıklarının sıfırlandığı "reset" fonksiyonu çağrılır.

```
# Uyarlanabilir eleştiri bileşeni. Ödüle ve sistem durumlarına göre uyarlanabilir
# karar bileşenine (ASE) hata işaretini verir. Sistem, bu işarete göre duvarlara
# çarpmaması ve çubuğu düşürmemesi için kendini ayarlamaktadır.
def __init__(self, dimension, lambda_val, gamma, dif_step):
    # ACE içerisinde kullanılacak değişkenler atanır.
    self.dimension = dimension # Durum değişkeni boyutu.
    self.lambda_val = lambda_val # Bozulma oranı.
    self.gamma = gamma # Hata işaretindeki benzerliği belirlemeye yarayan sabit.
    self.dif_step = dif_step # Açık euler için kullanılacak olan adım boyutu.

    self.reset()
```

Bu fonksiyonda sistemde kullanılacak $\bar{x}$, $p(t)$, $p(t-1)$, $x(t)$, $\hat{r}(t)$ terimlerinin ilk değerleri atanır ve başlangıçtaki $v(t)$ ağırlıkları belirlenir. Başlangıçta ağırlıklara $[-0.5,0.5]$ aralığında rastgele sayılar atanır.

```
def reset(self):
    self.v_weights = np.zeros((self.dimension, 1), dtype=float) # ACE ağırlıkları.
    # Başlangıç ağırlıkları [-0.5,0.5] aralığında rastgele sayılardır.
    self.v_weights[:,0] = np.random.rand(self.dimension)*0.1 - 0.05

    # Giriş işaretinin seçilebilirliğini gösterecek terim. Başlangıçta 0'dır.
    self.xhat_selection = np.zeros((self.dimension, 1), dtype=float)

    # Giriş, aktivasyon (p), önceki aktivasyon(p(k-1)) ve ödül beklentisi için
    # değişkenler oluşturulur. Başlangıçta 0'lardır.
    self.data = np.zeros(self.dimension)
    self.p_activation = 0
    self.prev_p_activation = 0
    self.r_int_reinf = 0
```

Obje içerisinde ayrıca "activate" fonksiyonu bulunur. Bu fonksiyon, beklenen ödül işaretini oluşturup dışarı verir. Aynı zamanda ağırlıklar bu fonksiyon içinde güncellenmektedir.

```
def activate(self, data, reinf):

    # İleri yol, ACE aktivasyonu ve beklenti hatasının hesaplanması.
    self.p_activation = np.dot(np.transpose(self.v_weights), data)
    self.r_int_reinf = reinf + self.gamma*self.p_activation - self.prev_p_activation

    # Ağırlık ve seçilebilirlik teriminin güncellenmesi.
    self.v_weights = self.v_weights + self.dif_step*(reinf + gamma*self.p_activation - self.prev_p_activation)*self.xhat_selection
    self.xhat_selection = self.lambda_val*self.xhat_selection + (1 - self.lambda_val)*data
    self.prev_p_activation = self.p_activation

    # Beklenti hatasını döndürür.
    return self.r_int_reinf
```

ASE:

Uyarlanabilir karar bileşeni (ASE) de ACE gibi ilk başlatıldığında kullanılacak sabitleri atar. Giriş boyutu, α ve δ değerleri atanır. Ayrıca kullanılacak aktivasyon fonksiyonu ve açık euler çözümünde kullanılacak adım büyüklüğü atanır. Daha sonra "reset" fonksiyonu çağrılır.

```
# Uyarlanabilir karar bileşeni. Durum değişkenlerine ve ödül beklentisine göre karar verir. Bu karar
# durumun değişiminde kullanılacaktır.
def __init__(self, dimension, alpha, delta, activation_type, dif_step):
    # ASE içerisinde kullanılacak değişkenler atanır.
    self.dimension = dimension # Durum değişkeni boyutu.
    self.alpha = alpha # Ağırlıkların değişme oranı.
    self.delta = delta # Bozulma sabiti.
    self.activation_type = activation_type # ASE'nin kullanacağı aktivasyon fonksiyonu.
    # Aktivasyon fonksiyonu için perceptron (signum) ve tanh fonksiyonları tanımlıdır.
    self.dif_step = dif_step # Açık euler için kullanılacak olan adım boyutu.

    self.reset()
```

Bu fonksiyonda $e(t)$, $y(t)$ ve $x(t)$ terimleri atanır. Ayrıca başlangıçtaki ağırlıklar ACE'de olduğu gibi $[-0.5,0.5]$ aralığında rastgele değerler alır.

```
def reset(self):
    self.w_weights = np.zeros((self.dimension, 1), dtype=float) # ASE ağırlıkları.
    # Başlangıç ağırlıkları [-0.5,0.5] aralığında rastgele sayılardır.
    self.w_weights[:,0] = np.random.rand(self.dimension)*0.1 - 0.05

    # Seçilen davranış ve sistemin cevabı arasındaki ilişkiyi belirten terim. Başlangıçta 0'dır.
    self.e_relevance = np.zeros((self.dimension, 1), dtype=float)

    # Giriş, ve aktivasyon (y) için değişkenler oluşturulur. Başlangıçta 0'lardır.
    self.data = np.zeros(self.dimension)
    self.y_activation = 0
```

Soru 1: Pekiştirmeli Öğrenme – Sistemin Python'da Kurulumu

ASE'de bulunan "activate" fonksiyonu, ödül beklentisi ve durum değişkenlerine göre karar verilmesini sağlar. Bu karar verilirken durgunluk olmaması için gürültü de bulunmaktadır. Daha sonra sistem ağırlıkları ve $e(t)$ uygunluk terimi güncellenir ve aktivasyon verilir.

```
def activate(self, data, int_reinf, noise):

    # İleri yol, ASE aktivasyonu.
    self.y_activation = self.activation_function(np.dot(np.transpose(self.w_weights), data) + noise)

    # Ağırlık ve uygunluk teriminin güncellenmesi.
    self.w_weights = self.w_weights + self.alpha*int_reinf*self.e_relevance
    self.e_relevance = self.delta*self.e_relevance + (1 - self.delta)*self.y_activation*data

    # Aktivasyonu döndürür.
    return self.y_activation
```

Aktivasyonun bulunması için seçilen aktivasyon fonksiyonu kullanılır. Tanımlı $sgn(-x)$ ve $tanh(-x)$ fonksiyonları bulunmaktadır. Kodun bu kısmında numpy.exp fonksiyonu kullanılmaktadır. Bu fonksiyonun "overflow" yaşaması durumunda x'in işaretine göre -1 veya +1 değeri seçilir.

```
def activation_function(self, x):
    # Aktivasyon fonksiyonu olarak tanh ve perceptron (signum) tanımlıdır. Fakat, bu fonksiyonların,
    # Tez_Kuyumcu.pdf'te belirtildiği gibi, y eksenine göre simetrileri alınmıştır.
    if self.activation_type == "tanh":
        if np.isinf((np.exp(x) + np.exp(-x))):
            # Kodda, x çok büyük veya çok küçük değerler alabiliyor. numpy.exp fonksiyonu
            # "overflow" hatası verdiğinde "t" değeri "nan" oluyor. Bu durumda, x değeri
            # zaten çok büyük veya çok küçük olduğu için tanh(x) fonksiyonu -1 veya 1 değerine
            # sahip. Bu nedenle "t"nin değerinin ölçülemediği durumlarda, değerin atanması için
            # perceptronda da kullanılan algoritma kullanılır. Bu işlem "t" hesaplanırken paydanın
            # sonsuzla karşılaştırılmasıyla yapılarak, hatadan önce saptanır.
            return -np.sign(-x)
        t = (np.exp(-x) - np.exp(x))/(np.exp(-x) + np.exp(x))
        return t
    elif self.activation_type == "perceptron":
        return np.sign(-x)
    else:
        return -x
```

Soru 1: Pekiştirmeli Öğrenme – Sistemin Python'da Kurulumu

Araba-Çubuk Sistemi:

Araba ve çubuğun durum değişkenlerini güncelleyeceği ve ACE'nin kullanması için ödül verecek olan objedir. Başlangıçta sistemle ilgili fiziksel parametreler girilir. Ardından durum değişkenleri için başlangıç koşullarının atanacağı "reset" fonksiyonu çağrılır.

```
# Araba ve çubuk sistemi.
def __init__(self, gravity, m_car, m_stick, l_stick, fs_ground, fs_car, dif_step, theta1_init, theta2_init, x1_init, x2_init):
    # Problemle ilgili fiziksel sabitler atanır.
    self.gravity = gravity # Yerçekimi ivmesi.
    self.m_car = m_car # Arabanın kütlesi.
    self.m_stick = m_stick # Çubuğun kütlesi.
    self.l_stick = l_stick # Çubuğun uzunluğu.
    self.fs_ground = fs_ground # Araba-yer arasındaki sürtünme sabiti.
    self.fs_car = fs_car # Çubuk-araba arasında sürtünme sabiti.
    self.dif_step = dif_step # Açık euler için kullanılacak olan adım boyutu.
    self.m1 = self.m_car + self.m_stick # Araba ve çubuğun toplam kütlesi.

    self.reset(theta1_init, theta2_init, x1_init, x2_init)
```

```
def reset(self, theta1_init, theta2_init, x1_init, x2_init):
    # Sistemdeki araba ve çubuğun başlangıç koşulları.
    self.theta1 = theta1_init
    self.theta2 = theta2_init
    self.x1 = x1_init
    self.x2 = x2_init
```

"activate" fonksiyonu, ASE'de alınan karara göre kuvvetin uygulanıp yeni durum değişkenlerinin elde edilmesini sağlar. Bu işleme $f1$ ve $f2$ terimlerinin ve F kuvvetinin hesaplanmasıyla başlar. Ardından açık euler çözümüyle yeni durum değişkenlerini elde eder ve günceller. Ortamın vereceği ödül veya cezayı seçerek, yeni durum değişkenleriyle birlikte dışarı verir.

```
def activate(self, y):
    # İleri yol, sistemdeki kuvvetin ve f1,f2 terimlerinin hesaplanması.
    F = 10*np.sign(y)
    f1 = self.m_stick*self.l_stick*(self.theta2**2)*math.sin(self.theta1) - self.fs_ground*np.sign(self.x2)
    f2 = self.gravity*math.sin(self.theta1) + math.cos(self.theta1)*((-F - f1)/self.m1) - self.fs_car*self.theta2/(self.m_stick*self.l_stick)

    # Araba konumu ve çubuk açısının sonraki durumlarının hesaplanması.
    new_theta1 = self.theta1 + self.dif_step*self.theta2
    new_theta2 = self.theta2 + self.dif_step*f2/(self.l_stick*(4/3 - self.m_stick*(math.cos(self.theta1)**2)/self.m1))
    new_x1 = self.x1 + self.dif_step*self.x2
    new_x2 = self.x2 + self.dif_step*((F + (f1 - self.m_stick*self.l_stick*f2)*math.cos(self.theta1))/self.m1)

    # Sistem durumunun güncellenmesi.
    self.update(new_theta1, new_theta2, new_x1, new_x2)

    # Ödül fonksiyonu. Açı büyüklüğü 8 dereceden küçük veya araba konumu 1.5 metreden küçükse
    # sistem ödül verir, bu sınırların dışındayken ceza verir.
    if abs(new_theta1) < 8 or abs(new_x1) < 1.5:
        reinf = 1
    else:
        reinf = -1

    # Bir sonraki iterasyonda veri olarak sistemin yeni durumu sunulur. Ödülle birlikte sistem durumu çekilir.
    return [np.array([new_theta1, new_theta2, new_x1, new_x2], dtype=float).reshape(4,1), reinf]
```

```
def update(self, new_theta1, new_theta2, new_x1, new_x2):
    # Sistem durumunun güncellenmesi.
    self.theta1 = new_theta1
    self.theta2 = new_theta2
    self.x1 = new_x1
    self.x2 = new_x2
```

Soru 1: Pekiştirmeli Öğrenme – Sistemin Python'da Kurulumu

Sistemin Başlatılması:

Öncelikle, sistemin kullanacağı λ ve γ değerleri makalede kullanıldığı gibi 0.8 ve 0.95 olarak atanır. Ardından sistemin başlangıç koşulları seçilir. Başlangıç koşulları farklı testler için farklı şekilde seçilecektir. Daha sonra ASE'nin kullanacağı aktivasyon fonksiyonu ve simülasyonun devam edeceği maksimum iterasyon sayısı seçilir.

```
__location__ = os.path.realpath(os.path.join(os.getcwd(), os.path.dirname(__file__)))

# Sistemin kullanacağı lambda ve gamma değişkenleri Tez_Kuyumcu.pdf'te belirtildiği gibi alınır.
lambda_val = 0.8
gamma = 0.95

# Sistemin başlangıç koşulları farklı testler için değiştirilebilir.
theta1_init = 10
theta2_init = 0.25
x1_init = 1.9
x2_init = 0.25

# Kullanılacak aktivasyon fonksiyonu ve simülasyonun maksimum iterasyon sayısı belirlenir.
# Tanh ve perceptron (signum) fonksiyonları tanımlıdır.
activation_type = "tanh"
epoch = 500
```

Sonuçların kaydedilmesi için kaydedilecek dosya adresi, test numarası ve boş bir sonuç listesi oluşturulur.

```
# Sonuçların kaydedileceği klasör oluşturulur.
resultFile = os.path.join(__location__, "Sonuclar")
if not os.path.exists(resultFile):
    os.makedirs(resultFile)

# Tez_Kuyumcu.pdf'te değeri verilmeyen fakat sistemin kullandığı iki sabit vardır. Bunlar "delta" (bozulma oranı)
# ve "alpha"dır (ağırlıkların değişme oranı). Bu değerlerin bu problemde optimal değerlerinin bulunması için test
# düzenlenir. Testte değerlerin 0, 0.2, 0.4, 0.6 ve 0.8 değerleri aldığı 10 farklı simülasyon yapılır. Bu simülasyonlar
# sonucunda elde edilen simülasyon iterasyon sayısı ve beklenti hata performansları bir excel dosyasında kaydedilir.
# Ayrıca her farklı değer çifti için sonuncu testte elde edilen durum değişkenleri ve beklenti hatası çizdirilir.
testnumber = 1
resultList = []
```

Soru 1: Pekiştirmeli Öğrenme – Sistemin Python'da Kurulumu

ASE'nin kullanacağı α ve δ değerleri seçilir. Bu değerler makalede verilmediği için, farklı değerler seçilerek farklı durumlarda sistemin performansı karşılaştırılacaktır. Figürlerin kaydedileceği dosya adresi ve başlıkları oluşturulur. Simülasyon başlatılır. Simülasyon sonucunda araba-çubuk sisteminin başarısız olduğu iterasyon sayısı ve simülasyonun beklenti hata performansı elde edilir. Bu değerler, sistemin koşullarıyla birlikte sonuç listesine eklenir. Tüm testler bittikten sonra sonuç listesi bir excel dosyasına kaydedilecektir.

```
for i in range(5):
    for j in range(5):
        for k in range(10):
            # Test edilecek olan alpha ve delta değerleri atanır.
            alpha = i*0.2
            delta = j*0.2

            # Çizimlerin kaydedileceği klasör belirlenir.
            savefolder = os.path.join(resultFile, "alpha0" + str(int(10*alpha)) + "_delta0" + str(int(10*delta)))

            # Çizimlerin başlıkları belirlenir.
            title = "alpha = " + str(alpha) + ", delta = " + str(delta)

            # Sistem aktive edilir ve simülasyon başlatılır.
            result = activate_system(lambda_val, gamma, alpha, delta, theta1_init, theta2_init, x1_init, x2_init, activation_type, epoch, title, savefolder)
            # Simülasyon sonucu elde edilen sonuçlar yazdırılır.
            print("delta:%.1f ---- alpha:%.1f ---- başarısız olunan iterasyon:%d ---- beklenti hata performansı:%d" % (delta, alpha, result[0], result[1]))

            # Sonuçlar kaydedilmek üzere bir listeye eklenir ve bir sonraki teste geçilir.
            resultList.append([testnumber, alpha, delta, theta1_init, theta2_init, x1_init, x2_init, result[0], result[1]])
            testnumber += 1

# Test sonuçları kaydedilir.
saveResults(resultList, resultFile, 'Test_Sonuclari.xls')
```

Simülasyon başlamadan önce ACE, ASE ve araba-çubuk sistemleri seçilen fiziksel sabitler, başlangıç koşulları ve sistem katsayıları kullanılarak oluşturulur. Sistemler oluşturulduktan sonra simülasyon başlatılır.

```
def activate_system(lambda_val, gamma, alpha, delta, theta1_init, theta2_init, x1_init, x2_init, activation_type, epoch, title, savefolder):
    # Araba-çubuk sisteminin başladığı fonksiyon.

    # Sistem için sabitler atanır. Sabitler Tez_Kuyumcu.pdf'ten alınmıştır.
    dimension = 4 # Durum değişkenleri boyutu = 4.
    gravity = -9.8 # Yerçekimi ivmesi = -9.8 m/s^2.
    m_car = 1 # Araba kütlesi = 1 kg.
    m_stick = 0.1 # Çubuk kütlesi = 0.1 kg.
    l_stick = 0.5 # Çubuk uzunluğu = 0.5 m.
    fs_ground = 5e-4 # Araba-yer arasındaki sürtünme katsayısı = 0.0005.
    fs_car = 2e-6 # Çubuk-araba arasındaki sürtünme katsayısı = 0.000002.
    dif_step = 0.005 # Açık euler çözümünde kullanılacak adım büyüklüğü = 0.005.

    # Karar bileşeni, eleştiri bileşeni ve araba-çubuk sistemi verilen parametrelerle oluşturulur.
    my_ACE = ACE(dimension, lambda_val, gamma, dif_step)
    my_ASE = ASE(dimension, alpha, delta, activation_type, dif_step)
    my_CarStick = CarStick(gravity, m_car, m_stick, l_stick, fs_ground, fs_car, dif_step, theta1_init, theta2_init, x1_init, x2_init)

    # Simülasyon başlatılır.
    sim_result = start_sim(my_ACE, my_ASE, my_CarStick, epoch, title, savefolder)

    # Simülasyon sonuçları döndürülür.
    return sim_result
```

Soru 1: Pekiştirmeli Öğrenme – Sistemin Python'da Kurulumu

Simülasyonda öncelikle sonuçların çizdirilmesi için beklenti hatası ve durum değişkenlerinin boş listeleri oluşturulur. Daha sonra başlangıçtaki durum değişkenleri çekilir ve bir dizide toplanır. Başlangıç ödülü 0 alınır.

```
def start_sim(ACE, ASE, CarStick, epoch, title, savefolder):
    y_delta = []
    y_state = []

    # Başlangıçtaki durum değişkenleri çekilir ve veri olarak birleştirilir.
    theta1 = CarStick.theta1
    theta2 = CarStick.theta2
    x1 = CarStick.x1
    x2 = CarStick.x2
    data = np.array([theta1, theta2, x1, x2], dtype=float).reshape(4,1)

    # Başlangıç ödülü 0 alınır.
    reinf = 0
```

Sonuçların elde edilmesi ve kaydedilmesi için kullanılacak değişkenler atanır. "lastiter" simülasyonun sürdüğü iterasyon sayısını, "delta_performance" beklenti hatasının 0.5'in altında en uzun süre kaldığı iterasyon sayısını ve "performance_iter" simülasyon sırasında o an için beklenti hatasının ardışık olarak 0.5'in altında kaldığı iterasyon sayısını belirtir. "simdone" ise simülasyonu durdurma kararıdır. Eğer durdurma kararı alınmışsa son ölçümler yapılır ve simülasyon sonlandırılır. Bu kararın alınması için durum değişkenlerinin belirlenen limitlerin dışına çıkması gerekir.

```
# Sistem performansının ölçütleri olarak simülasyonun sürdüğü iterasyon sayısı ve
# beklenti hatasının düşük olduğu seri iterasyon sayısı ile ilgili değişkenler atanır.
lastiter = epoch
delta_performance = 0
performance_iter = 0
simdone = False
```

Soru 1: Pekiştirmeli Öğrenme – Sistemin Python'da Kurulumu

İlk iterasyon başlatılır. İterasyon başında gürültü değeri için daha önce belirtildiği koşullarda rastgele bir sayı seçilir.

```
# Simülasyon, araba çarpmadıkça veya çubuk düşmedikçe, "epoch" iterasyon devam edecektir.
for i in range(epoch):

    # 0 medyan ve 0.1 standart sapmalı normal dağılımlı rastgele bir gürültü tanımlanır.
    noise = np.random.normal(0, 0.1)
```

İlk sistem olarak ACE aktive edilerek ödül beklentisi bulunur.

```
# Öncelikle eleştiri bileşeni çalıştırılarak durum değişkenlerine göre
# ödül beklentisi bulunur.
int_reinf = ACE.activate(data, reinf)
```

Bulunan ödül beklentisi, durum değişkenleri ve gürültü ile ASE'ye verilir. ASE kararı belirler.

```
# Bulunan ödül bileşeni, gürültü ve durum değişkenleri kullanılarak karar bileşeni
# çalıştırılır. Karar bileşeni çıkışı arabaya uygulanacak kuvveti belirler.
y_activation = ASE.activate(data, int_reinf, noise)
```

ASE'nin verdiği karar ile araba-çubuk sisteminde kuvvet uygulanır. Değişen açı, açı hızı, konum ve hız değerleri ACE'ye verilecek ödül/ceza ile birlikte elde edilir. Ödül beklentisi ve elde edilen ödülün farkı alınarak beklenti hatası hesaplanır.

```
# Karar bileşeni çıkışı ile sistem çalıştırılır. Uygulanan kuvvet ve sistemin durumuna
# göre, sistemin yeni durumu ve eleştiri bileşenine iletilecek ödül bulunur.
[data, reinf] = CarStick.activate(y_activation)

# Beklenti hatası hesaplanır.
delta = int_reinf - reinf
```

Durum değişkenlerinin belirlenen limitler içinde olup olmadığı kontrol edilir. Bu limitler $-2.4 < x < 2.4$ ve $-12 < \theta < 12$ şeklindedir. Limitler dışına çıkılmışsa simülasyonu durdurma kararı alınır. Karar alındıktan sonra son ölçümler ve beklenti hata performansı hesaplanıp simülasyon durdurulur.

```
# Araba 2.4 metrenin dışında veya çubuk 12 dereceden daha eğik ise simülasyonu durdurma kararı alınır.
if (abs(CarStick.x1) > 2.4 or abs(CarStick.theta1) > 12):
    simdone = True
```

Soru 1: Pekiştirmeli Öğrenme – Sistemin Python'da Kurulumu

Beklenti hatasının büyüklüğünün 0.5'in altında olması durumunda "performance_iter" değeri artırılır. Eğer beklenti hatası bu sınırın dışına çıkmışsa veya simülasyonun son iterasyonuysa, "performance_iter" ve "delta_performance" karşılaştırılır. Eğer "performance_iter" daha büyükse, "delta_performance" değerine atanır. Bu şekilde simülasyon süresince beklenti hatasının büyüklüğünün 0.5'in dışına çıkmadığı en uzun iterasyon sayısı elde edilir.

```
if abs(delta) < 0.5:
    performance_iter += 1

if (abs(delta) >= 0.5 and performance_iter > 0) or i + 1 == epoch or simdone == True:
    if performance_iter > delta_performance:
        # Beklenti hatasının 0.5'ten küçük olduğu ardışık iterasyon sayısı kaydedilir.
        # Simülasyon boyunca en uzun süre bu aralıkta kalınan sayı, simülasyonun
        # beklenti hata performansını belirler.
        delta_performance = performance_iter
    performance_iter = 0
```

Çizdirilmek üzere beklenti hatası ve durum değişkenleri kaydedilir. Durdurma kararı kontrol edilir ve duruma göre simülasyon durdurulur veya sonraki iterasyona geçilir.

```
# Simülasyon sonunda çizilmek üzere beklenti hatası ve durum değişkenleri bir listede kaydedilir.
y_delta.append(delta)
y_state.append([CarStick.theta1, CarStick.theta2, CarStick.x1, CarStick.x2])

# Simülasyonu durdurma kararı alındıysa son iterasyon kaydedilir ve simülasyon durdurulur.
if simdone == True:
    # Simülasyonun sürdüğü iterasyon sayısı kaydedilir.
    lastiter = i + 1
    break
```

Soru 1: Pekiştirmeli Öğrenme – Sistemin Python'da Kurulumu

Simülasyon boyunca beklenti hatası ve durum değişkenleri çizdirilip testin kullandığı α ve δ değerlerine göre belirlenen isimle kaydedilir.

```
plt.figure()
plt.plot(x, np.array(y_delta, dtype=float).reshape(lastiter))
plt.ylim(-3, 3)
plt.xlim(0,epoch)
plt.title(title)
plt.savefig(savefolder + '_Hata.png')
# plt.show()
plt.close()

fig, axs = plt.subplots(4)
fig.suptitle('Durum Değişkenlerinin Değişimi')
axs[0].plot(x, y_state[:,0])
axs[0].set_title('Çubuk Açısı')
axs[1].plot(x, y_state[:,1])
axs[1].set_title('Çubuk Açısının Hızı')
axs[2].plot(x, y_state[:,2])
axs[2].set_title('Araba Konumu')
axs[3].plot(x, y_state[:,3])
axs[3].set_title('Araba Hızı')
fig.text(0.5, 0.04, 'İterasyon', ha='center')
fig.savefig(savefolder + '_Durum.png')
plt.close()
```

Soru 1: Pekiştirmeli Öğrenme – Sistemin Python'da Kurulumu

Simülasyon sonunda, simülasyonun sürdüğü iterasyon sayısı ve beklenti hatası performansı döndürülür.

```
# Simülasyonun sürdüğü iterasyon sayısı ve beklenti hata performansı döndürülür.
return [lastiter, delta_performance]
```

Elde edilen sonuçlar daha sonra yorumlanmak üzere xlwt ve xlrd modülleri kullanılarak excel dosyasında kaydedilir.

```
# Elde edilen sonuçları excel dosyasına kaydeden fonksiyon.
def saveResults(results, savefolder, filename):
    if not os.path.exists(savefolder):
        os.makedirs(savefolder)
    filedir = os.path.join(savefolder,filename)
    wb = Workbook()
    sheet1 = wb.add_sheet('Sheet1')
    col = 0
    row = 0
    sheet1.write(row, col, 'Test Numarası')
    sheet1.write(row, col+1, 'Alpha')
    sheet1.write(row, col+2, 'Delta')
    sheet1.write(row, col+3, 'Theta1_init')
    sheet1.write(row, col+4, 'Theta2_init')
    sheet1.write(row, col+5, 'X1_init')
    sheet1.write(row, col+6, 'X2_init')
    sheet1.write(row, col+7, 'Başarısız Olunan İterasyon')
    sheet1.write(row, col+8, 'Beklenti Hata Performansı')
    row += 1
    for result in results:
        for i in range(len(result)):
            sheet1.write(row, col+i, result[i])
        row +=1

    wb.save(filedir)
```

Soru 1: Pekiştirmeli Öğrenme – Test Sonuçları

Test Sonuçları:

Test edilecek durumlar;

- α ve δ değerleri {0, 0.2, 0.4, 0.6, 0.8} kümesinden seçilirken sistemin merkezi ve sınıra yakın koşullarda davranışı, seçilen değer çiftlerinin performansları,
- araba ve çubuk merkezi bir konumdayken signum ve tanh fonksiyonlarının performansları,

şeklindedir. Öncelikle tanh fonksiyonu kullanılan bir sistemde α ve δ değer çiftleri karşılaştırılarak iyi performansa sahip değerler seçilecek, ardından bu değerler kullanılarak aktivasyon fonksiyonu olarak signum ve tanh fonksiyonları kullanılacak ve karşılaştırılacaklardır. Karşılaştırılacak iki ölçüt simülasyonun sürdüğü iterasyon sayısı ve beklenti hatası performansıdır. Ayrıca, karşılaştırma yapılırken aynı durumlar 10 kere test edilip karşılaştırma ölçütlerinin ortalamaları alınıp kullanılacaktır.

α, δ Testi:

Test daha önce belirtildiği gibi α ve δ değerlerinin {0, 0.2, 0.4, 0.6, 0.8} kümesinden seçilip farklı başlangıç koşullarında simüle edilmesiyle yapılacaktır. Başlangıç koşulları dört şekilde seçilecektir:

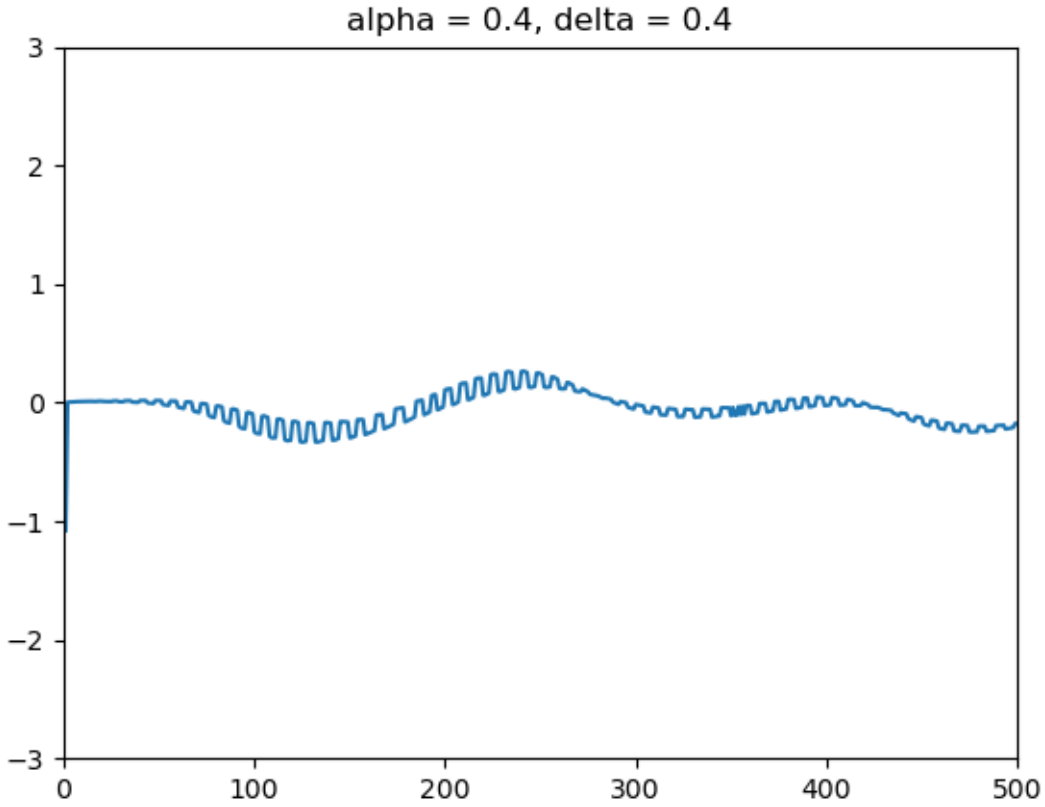
	Merkezi	θ Sınırda	x Sınırda	θ ve x Sınırda
θ	1	10	1	10
$\dot{\theta}$	0	0.5	0	0.25
x	1	1	1.9	1.9
$\dot{x}$	0	0	0.5	0.25

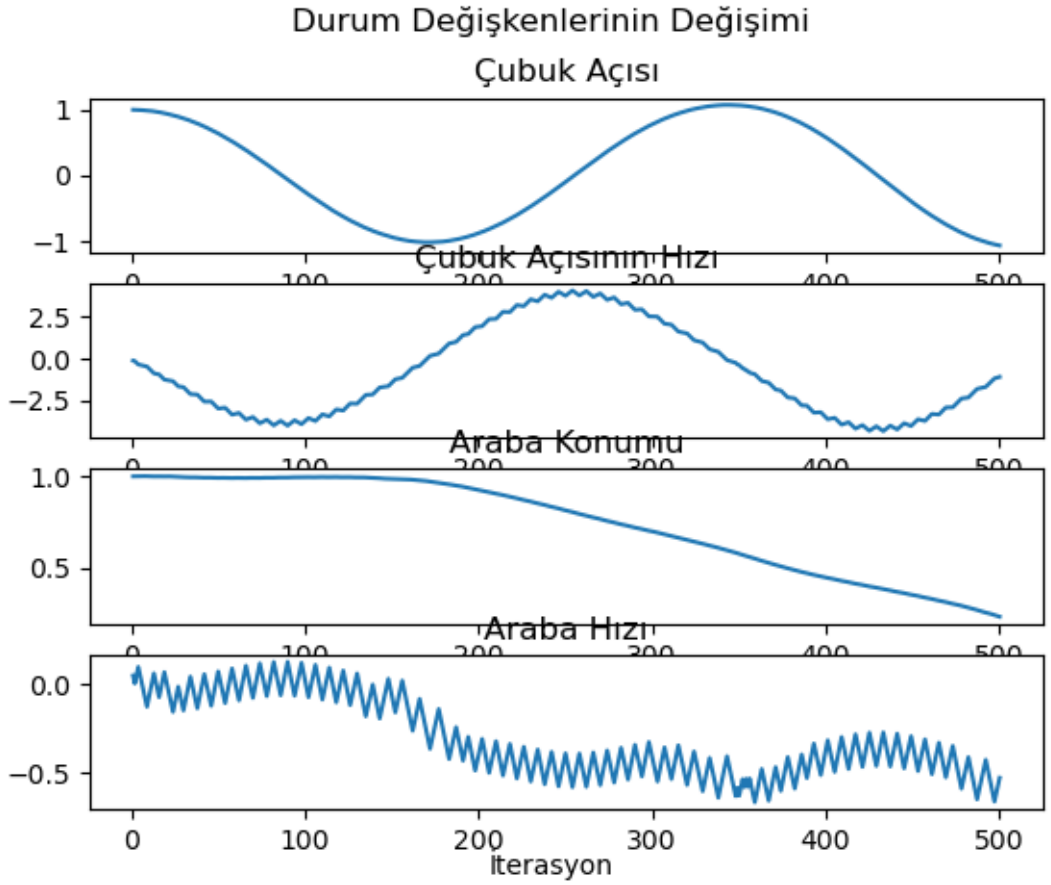
Merkezi Durum:

Bu durumda araba ve çubuk herhangi bir zorlukla karşılaşmazlar. Rahat bir pozisyonda başlayıp sadece küçük düzenlemeler yaparak simülasyona devam ederler.

Soru 1: Pekiştirmeli Öğrenme – Test Sonuçları

α, δ değerleri 0.4 iken (0.4 değeri ortalama değer olduğu için seçilmiştir, diğer değer çiftlerinin sonuçları 'Sonuclar_Merkez' klasöründe bulunmaktadır.) durum değişkenleri ve beklenti hatası şu şekilde değişmiştir:



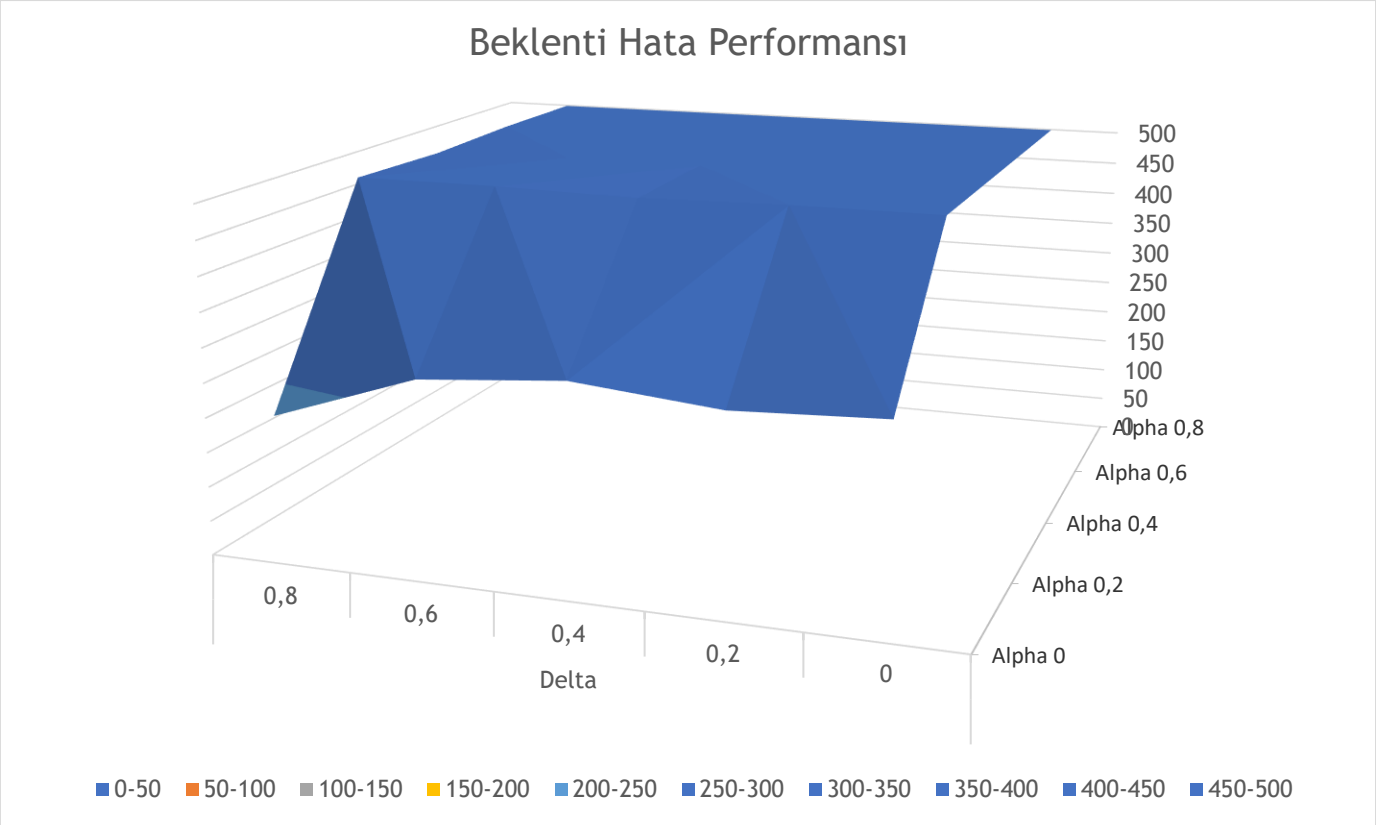


Soru 1: Pekiştirmeli Öğrenme – Test Sonuçları

Beklenti hatasının 0'a yakın bir şekilde ilerlediği görülür. Ayrıca çubuk açısının 0 etrafında değişmekte olup araba konumunun 0'a daha da yaklaştığı görülür.

Bu test sonucunda elde edilen beklenti hata performansı tablosu şu şekildedir:

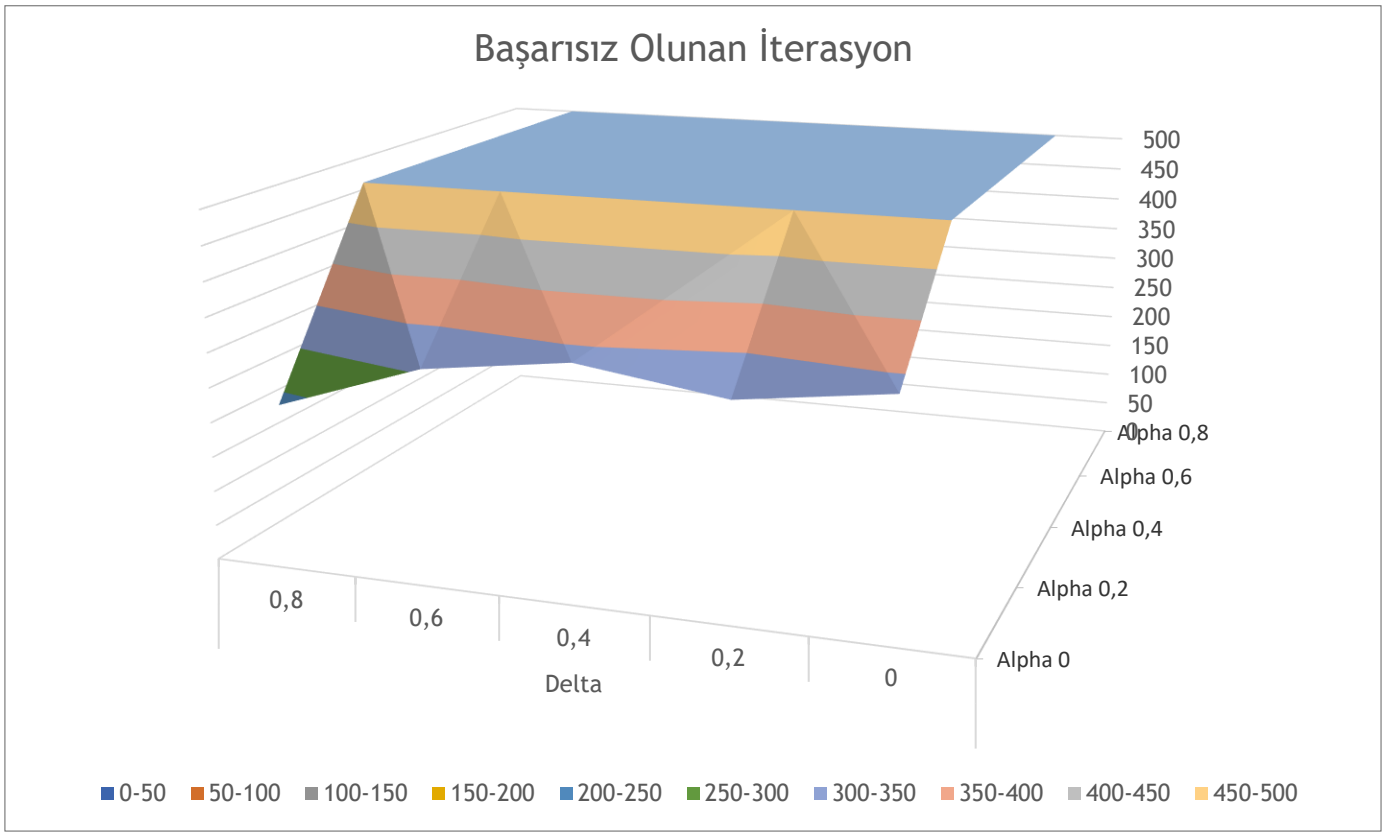
$\alpha\backslash\delta$	0	0,2	0,4	0,6	0,8
0	293,5	284	302,8	284,9	213,6
0,2	499	499	495,1	499	499
0,4	499	499	499	499	494,5
0,6	499	499	499	499	499
0,8	499	499	499	499	499



Simülasyonun sürdüğü iterasyon sayısı ise şöyledir:

$\alpha\backslash\delta$	0	0,2	0,4	0,6	0,8
0	332	304,1	333,4	305,6	235,5
0,2	500	500	500	500	500
0,4	500	500	500	500	500
0,6	500	500	500	500	500
0,8	500	500	500	500	500

Soru 1: Pekiştirmeli Öğrenme – Test Sonuçları



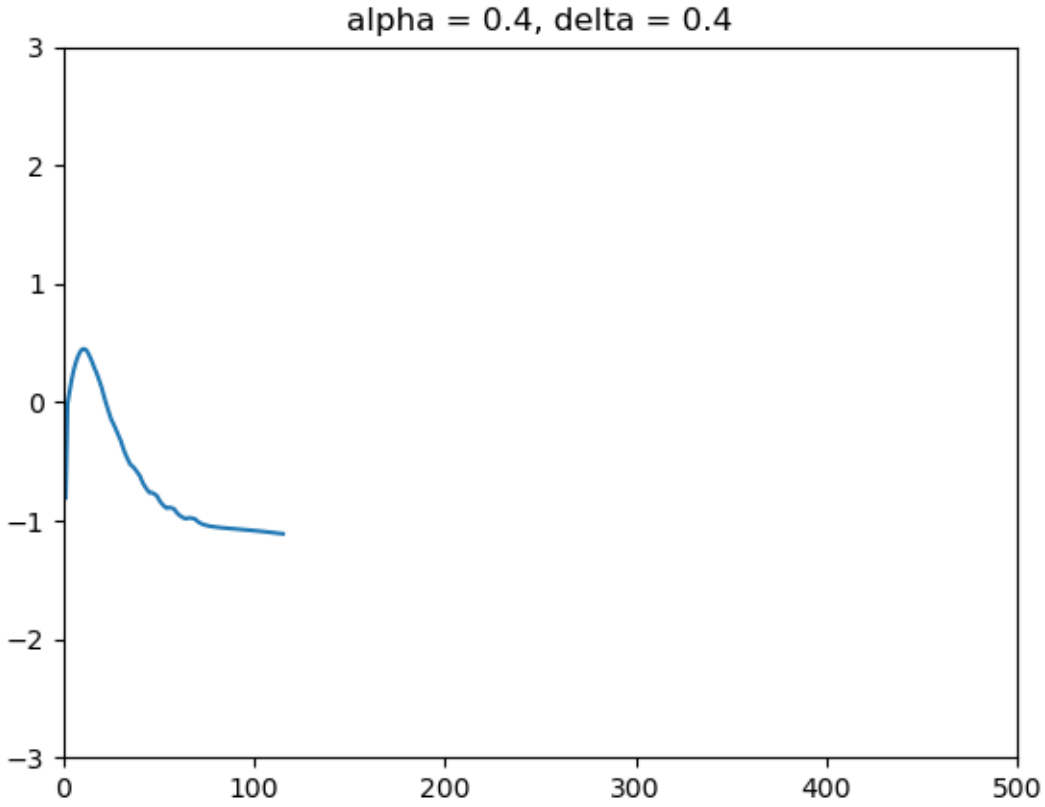
Görülebileceği gibi, α = 0 olduğu durumlarda sistem istenilen şekilde çalışmamıştır. Araba ve çubuk belirtilen sınırların dışına çıkmış ve simülasyon erken bitmiştir. Buna rağmen beklenti hata performansı, simülasyonun sürdüğü iterasyon sayısına yakın değerler almıştır. Bu testten δ değerinin sisteme etkisi hakkında yorumlanabilecek bir veri elde edilmemiştir.

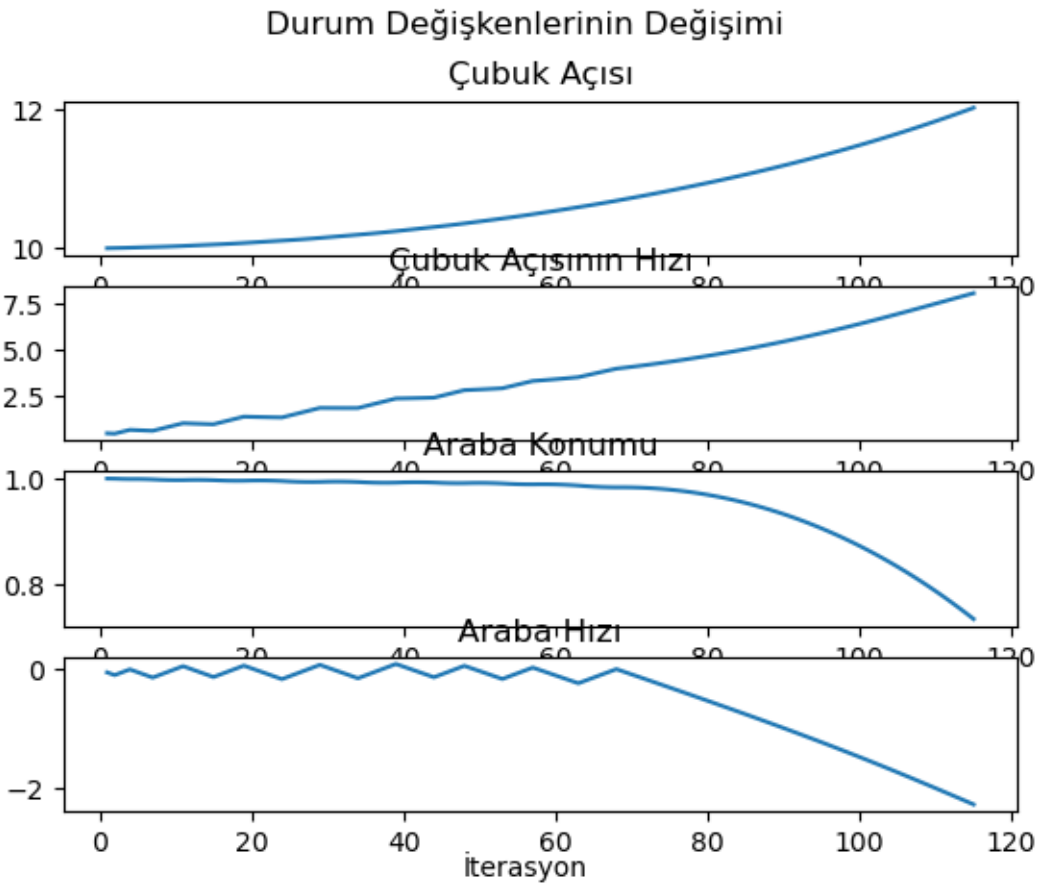
Soru 1: Pekiştirmeli Öğrenme – Test Sonuçları

θ'nın Sınırda Olması Durumu:

Bu durumda başlangıçta çubuğun açısı 10 derece ve açının hızı 0.5 seçilir. Arabanın, çubuk düşmeden açısını düşürmeye çalışması gerekir.

α, δ değerleri 0.4 iken durum değişkenleri ve beklenti hatası şu şekilde değişmiştir:



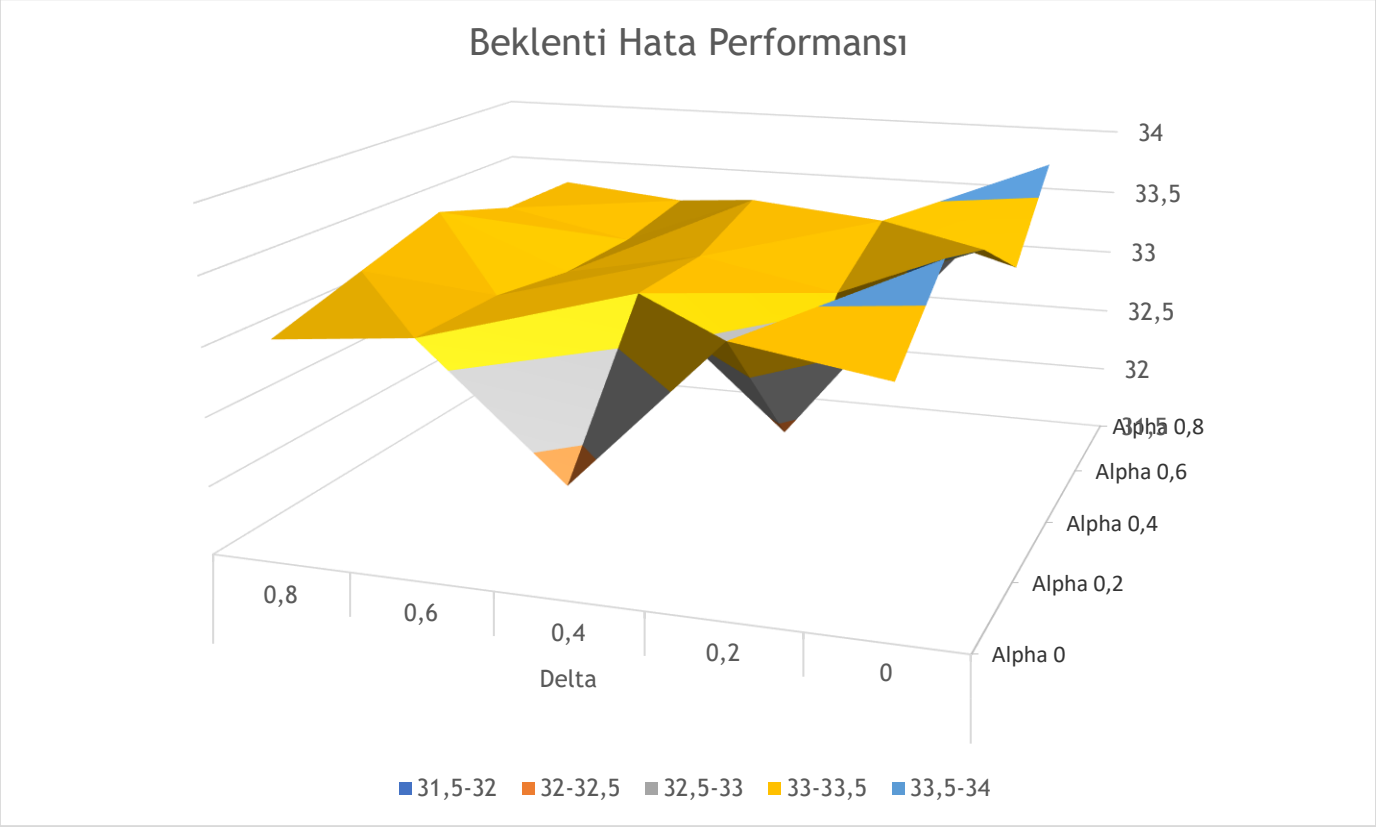


Soru 1: Pekiştirmeli Öğrenme – Test Sonuçları

Simülasyon sırasında arabanın çubuk açısını düzeltmek için yeterince hızlı karar vermediği ve ilerlemek istemediği, çubuğun düşmesine izin verip simülasyonun bitmesine sebep olduğu görülür. Bu durum, diğer değer çiftleri için gerçekleştirilen simülasyonlarda da gözlemlenmektedir.

Test sonucunda elde edilen beklenti hata performansları:

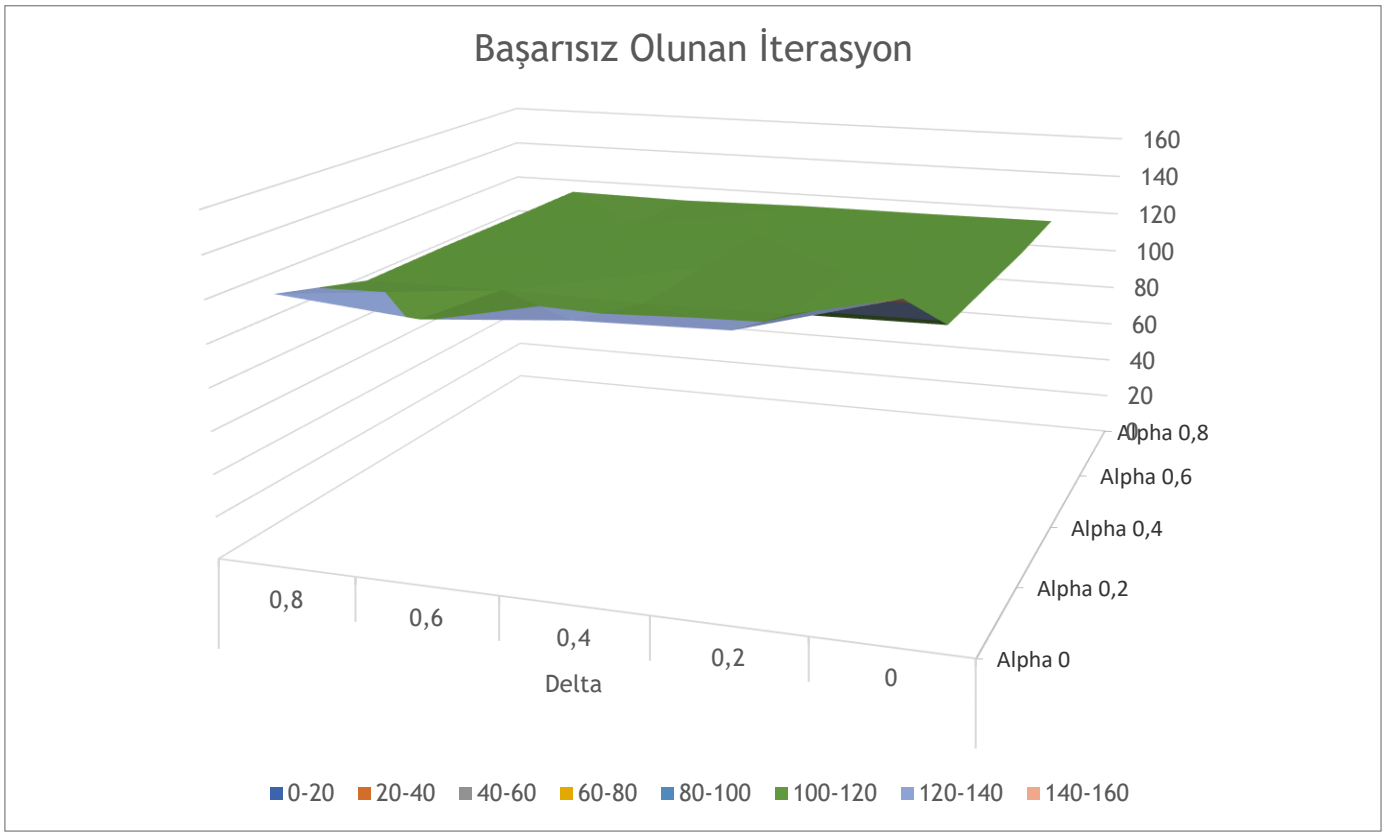
$\alpha\backslash\delta$	0	0,2	0,4	0,6	0,8
0	33,2	33,36	32,3	33,2	33,1
0,2	33,7	32,4	33,3	33,2	33,3
0,4	33,5	33,1	33,3	33,1	33,5
0,6	33,1	33,4	33,5	33,1	33,3
0,8	33,7	32,5	33,2	33,2	33,3



Test sonucunda elde edilen simülasyon iterasyonu:

$\alpha\backslash\delta$	0	0,2	0,4	0,6	0,8
0	144,4	126,1	124,6	119,5	125,2
0,2	114,6	113,9	113	114,2	113,5
0,4	113,8	114,4	113	113,5	113,8
0,6	112,9	113	113,4	113,4	113,4
0,8	113,8	113,5	113,5	112,4	113,3

Soru 1: Pekiştirmeli Öğrenme – Test Sonuçları



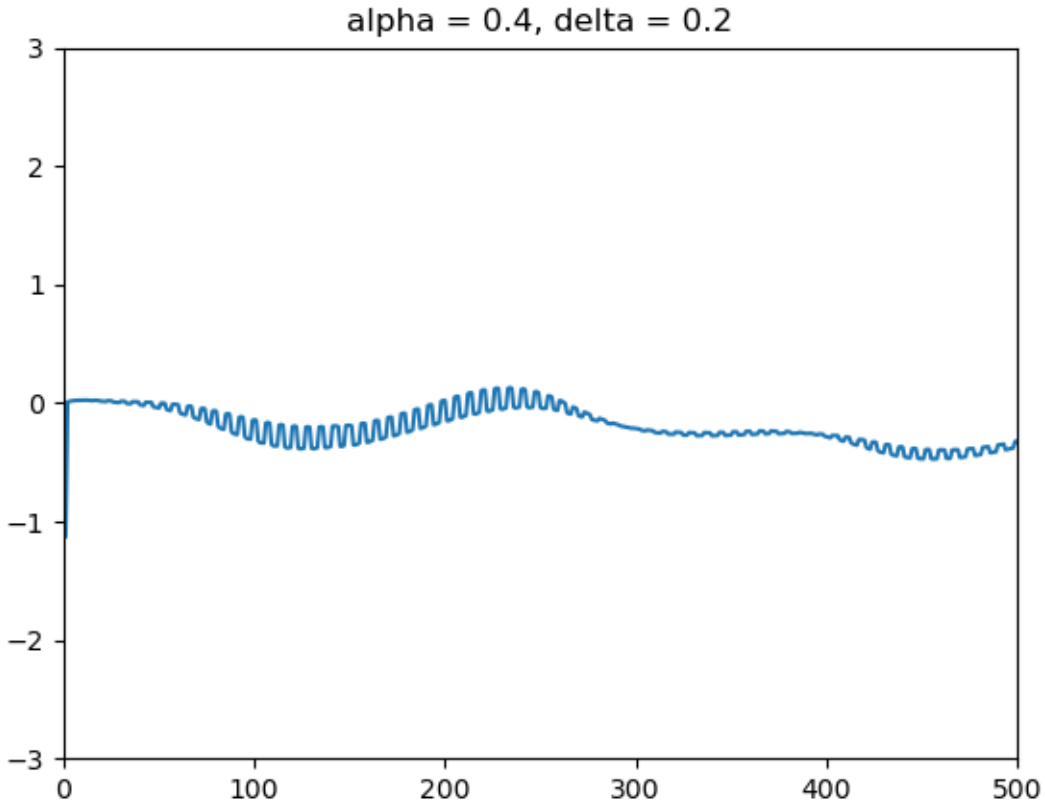
Test sonucunda yeterince değerli veriler elde edilememiştir. Arabanın başlangıç koşullarından toparlanması fazlasıyla zordur ve bu sebeple tüm testlerde yakın veriler elde edilmiştir. Göze çarpan tek sonuç, α ve δ düşük olduğu değerlerde simülasyonun daha uzun süre devam etmesidir. Bu özellikle α değeri için geçerlidir.

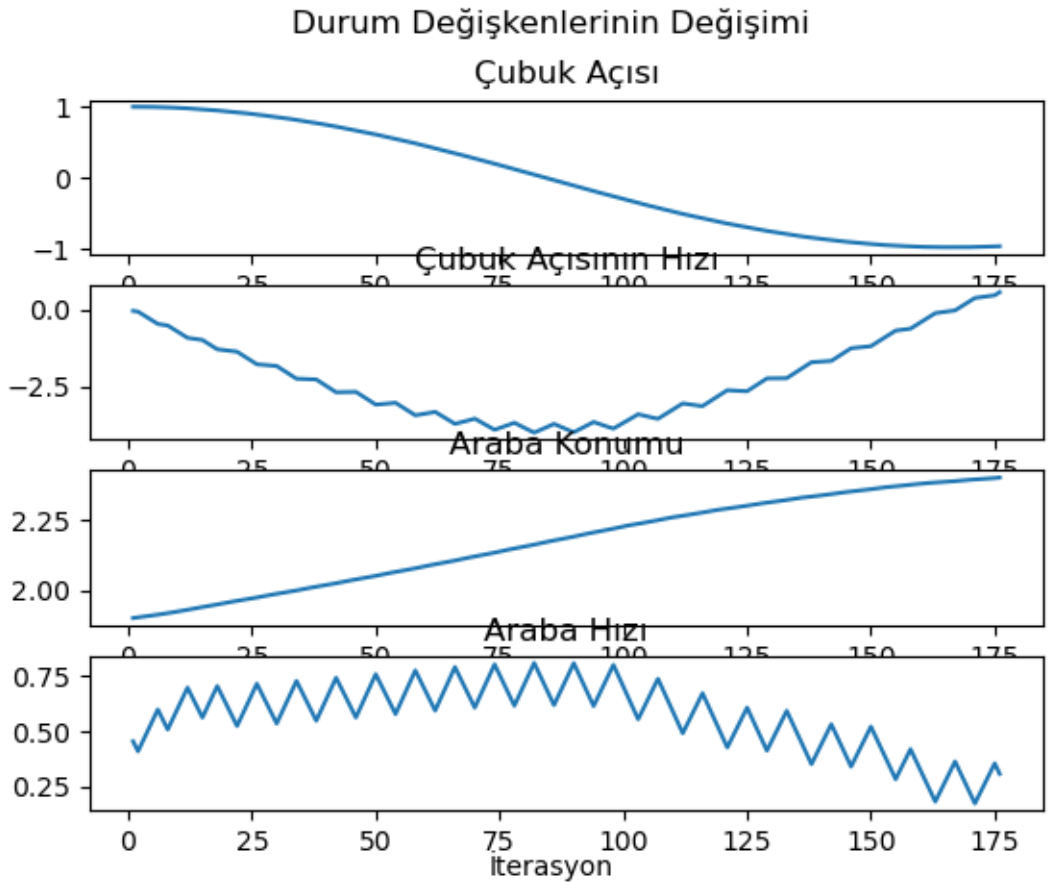
Soru 1: Pekiştirmeli Öğrenme – Test Sonuçları

x'in Sınırda Olması Durumu:

Önceki test gibi sınır koşullarda başlatılan bir durumdur. Bu sefer çubuk rahat bir açıdayken araba duvara yakın bir konumda bulunmakta ve hızlanmaktadır.

α, δ değerleri 0.4 iken durum değişkenleri ve beklenti hatası şu şekilde değişmiştir:



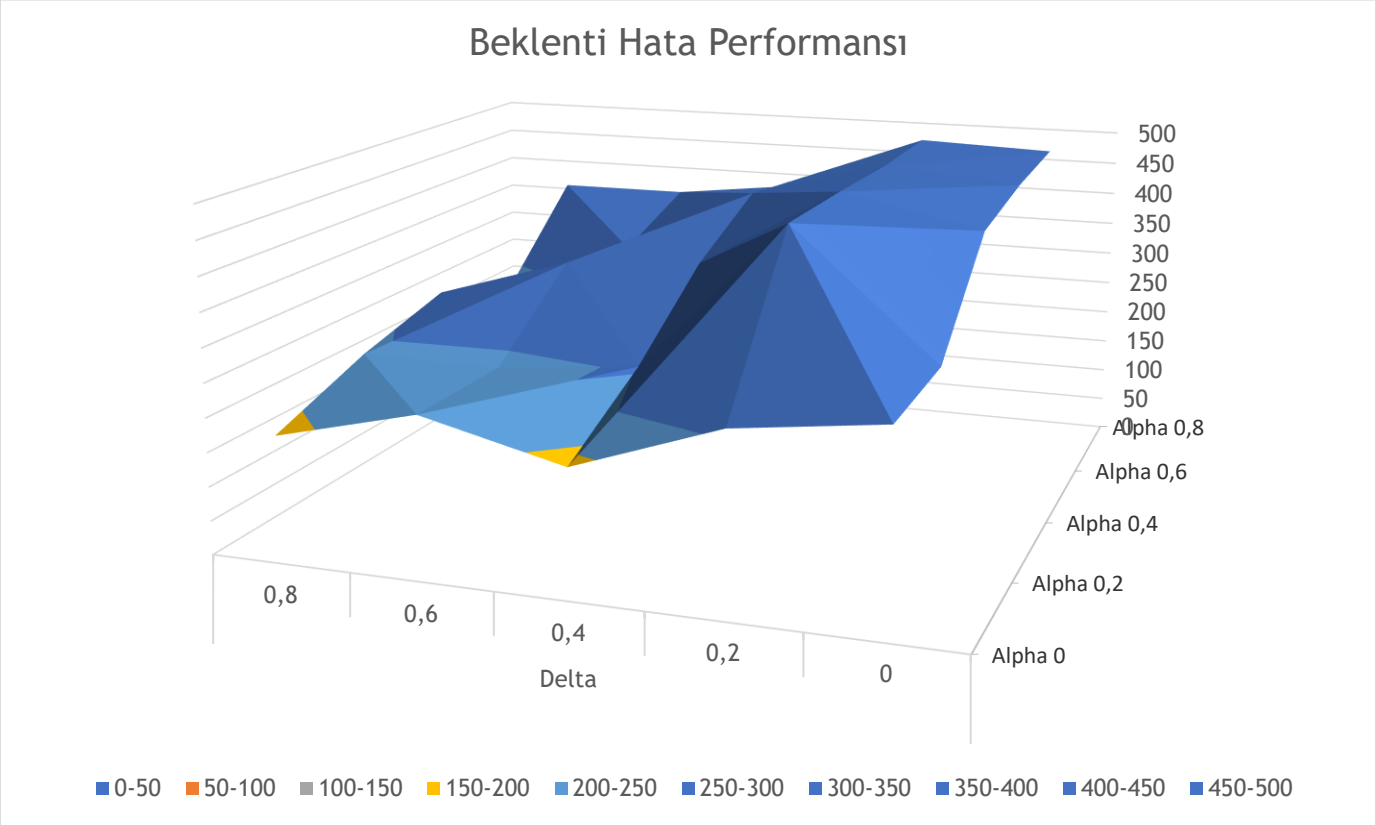


Soru 1: Pekiştirmeli Öğrenme – Test Sonuçları

Beklenti hatasının 0'a yakın bir değerde ilerlediği görülür. Ayrıca arabanın hızını yeterince çabuk düşüremeyip önündeki duvara çarptığı ve simülasyonun durdurulduğu gözlenir.

Test sonucunda elde edilen beklenti hata performansları:

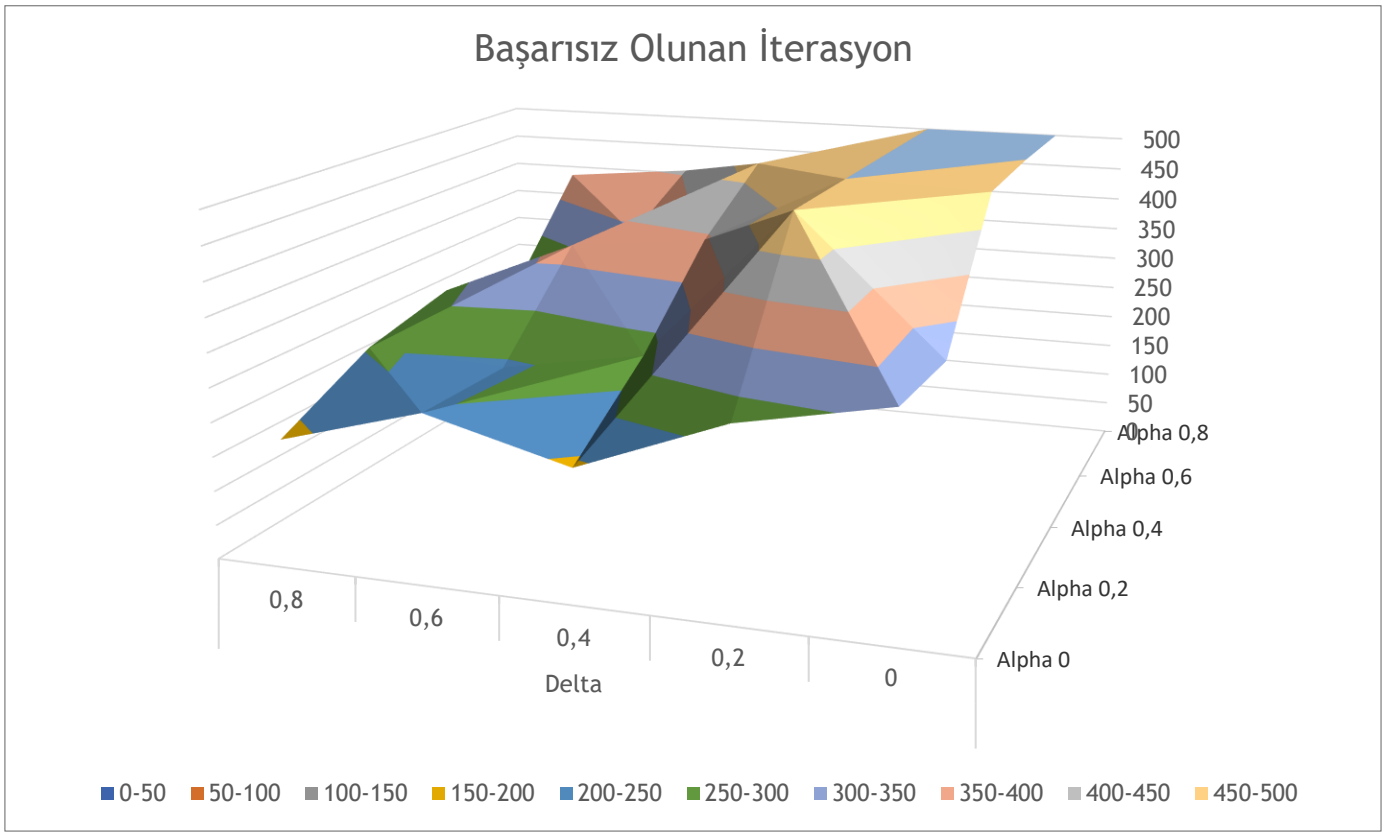
$\alpha\backslash\delta$	0	0,2	0,4	0,6	0,8
0	287	260,27	186,2	236,3	185,2
0,2	293,6	474,6	255,6	234,1	236,3
0,4	429	473	350,6	336,1	272,3
0,6	451,8	468,4	412,7	307,9	221,8
0,8	463,8	471,4	380,9	356,7	356,4



Test sonucunda elde edilen simülasyon iterasyonu:

$\alpha\backslash\delta$	0	0,2	0,4	0,6	0,8
0	315,6	273,2	191,5	245,1	186,2
0,2	308,7	500	278,3	241,4	253,2
0,4	493,9	500	395	371,8	283,8
0,6	500	500	470,3	341,6	228,7
0,8	500	500	430,9	405	384,9

Soru 1: Pekiştirmeli Öğrenme – Test Sonuçları



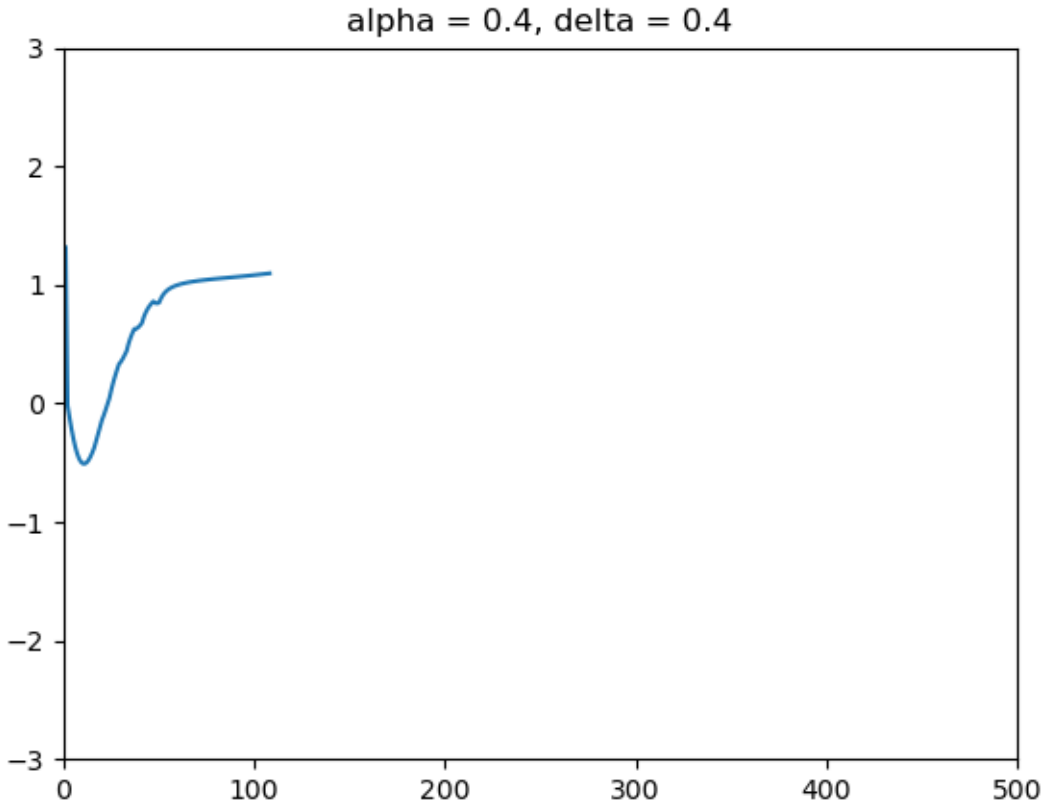
Beklenti hata performansları incelendiğinde, α değerinin yüksek ve δ değerinin düşük olmasının bu değeri iyi yönde etkilediği görülebilir. Bunu özellikle α değerinin yüksek olması etkiler. Beklenti hata performansındaki farklılığın sebebi ise çoğunlukla simülasyonun sürdüğü iterasyon sayısındaki farklılıktır. α'nın yüksek ve δ'nın düşük olduğu durumlarda araba-çubuk sistemi limitlerin dışına çıkmayıp maksimum iterasyona ulaşmayı başarmışlardır. Bu test için en başarılı değer ikilisi $\alpha = 0.2,\ \delta = 0.2$ olup maksimum iterasyona ulaşılan diğer simülasyonlar arasındaki fark fazlasıyla düşüktür.

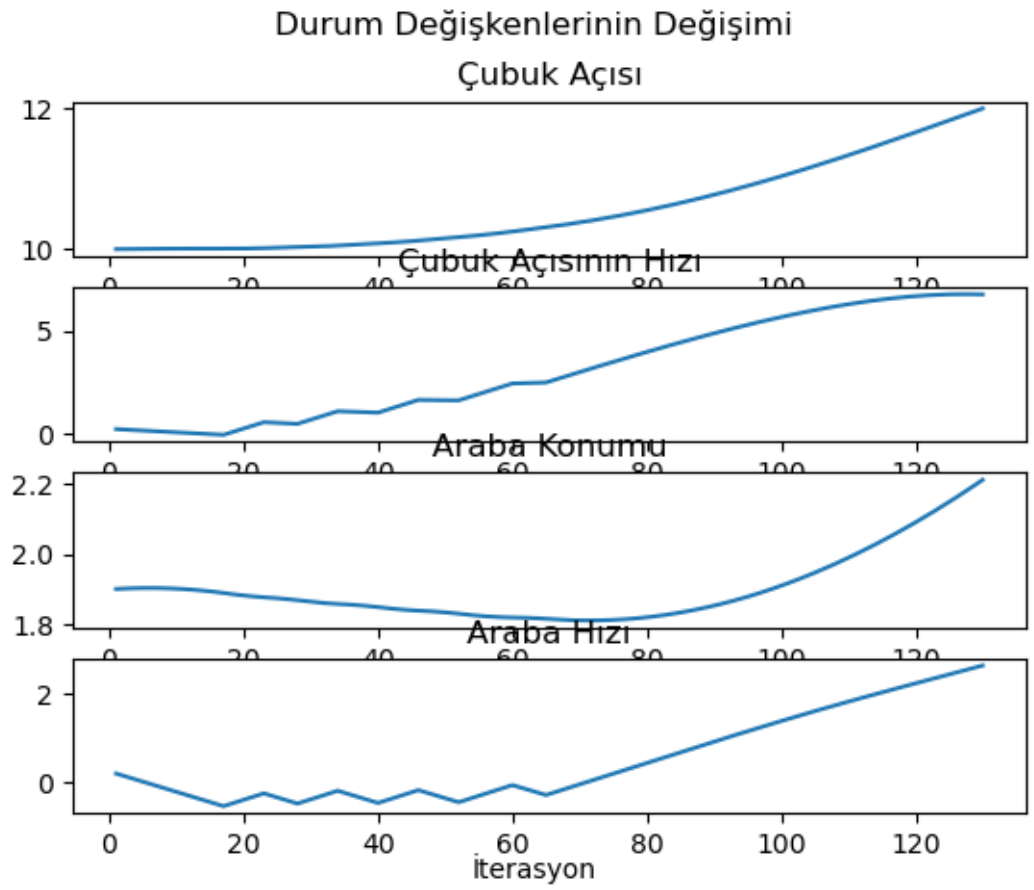
Soru 1: Pekiştirmeli Öğrenme – Test Sonuçları

θ ve x'in Sınırda Olması Durumu:

Bu durumda hem araba hem de çubuk limitlerine fazlasıyla yakın bir durumda başlamaktadırlar. Çubuğun düşmemesi için arabanın hızlanması gerekir fakat hızlanırsa duvara çarpma ihtimali vardır.

α, δ değerleri 0.4 iken durum değişkenleri ve beklenti hatası şu şekilde değişmiştir:



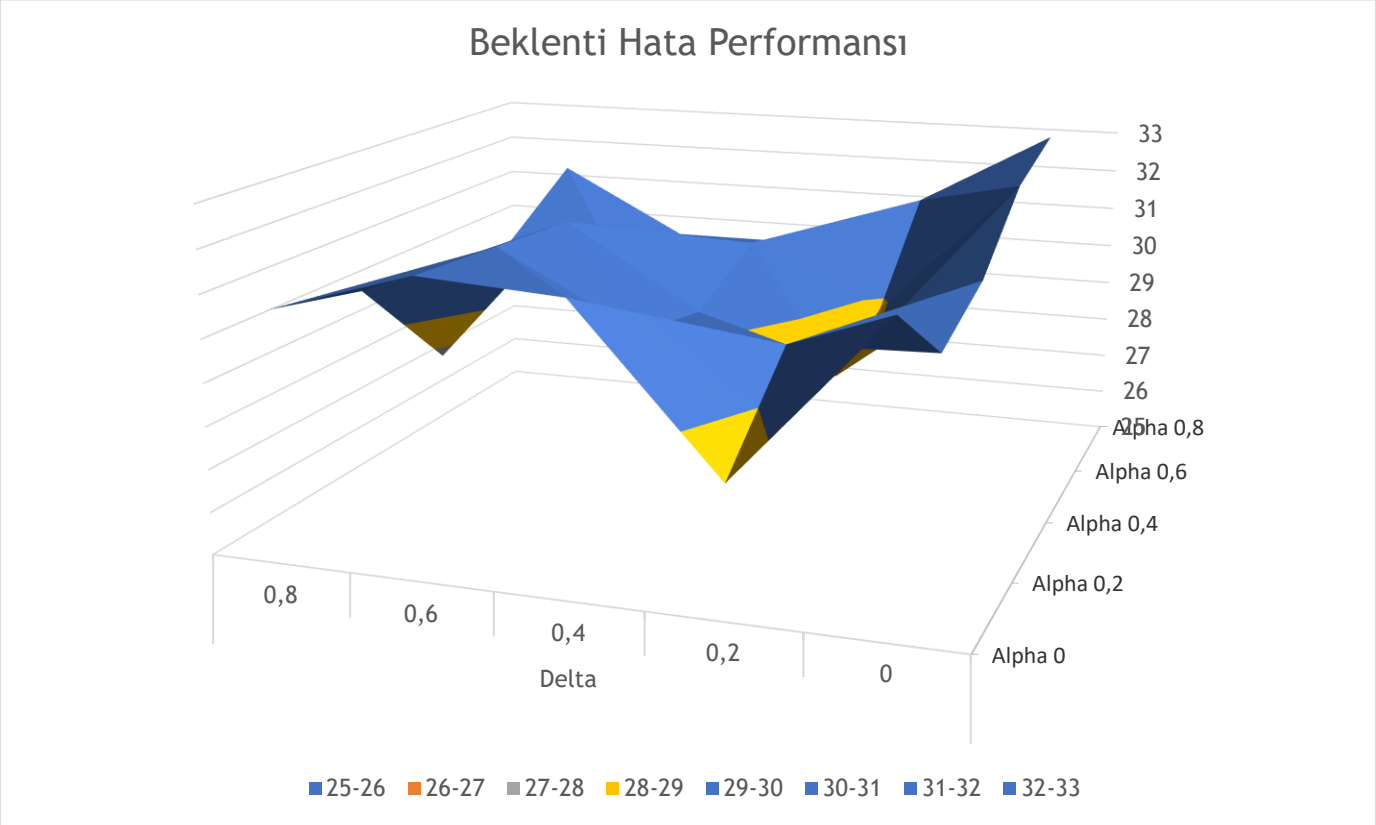


Soru 1: Pekiştirmeli Öğrenme – Test Sonuçları

Koşullar, sistemin dengeyi bulması ve simülasyonu bitirebilmesi için fazlasıyla zordur. Buna rağmen bazı testlerde sistemlerin diğerlerinden daha iyi performans sergilediği görülebilir.

Test sonucunda elde edilen beklenti hata performansları:

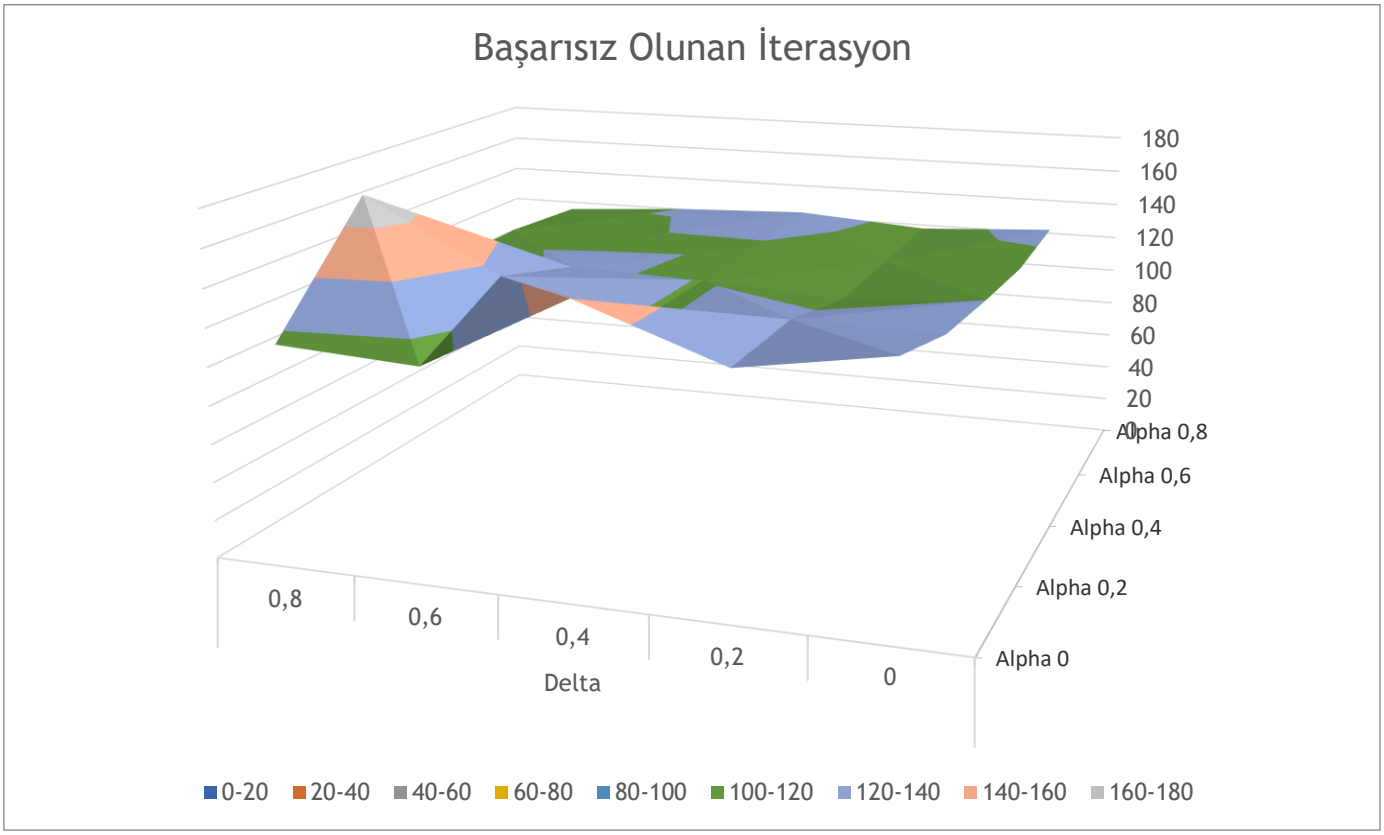
$\alpha\backslash\delta$	0	0,2	0,4	0,6	0,8
0	31,8	28	31,6	31,8	30,8
0,2	30	29,9	29,9	31,6	30,3
0,4	30,7	28,1	29,4	31,4	27,7
0,6	32,2	28,8	30,3	28,9	29,8
0,8	32,8	30,9	29,5	29,5	31,2



Test sonucunda elde edilen simülasyon iterasyonu:

$\alpha\backslash\delta$	0	0,2	0,4	0,6	0,8
0	136,3	123,9	150,1	110,8	115
0,2	123,9	124,9	119	135,2	173,1
0,4	120	114,1	119,1	120,9	116,7
0,6	117,4	115,7	119,4	119,2	118,9
0,8	122,1	117,8	123	120,3	115,4

Soru 1: Pekiştirmeli Öğrenme – Test Sonuçları



Görüldüğü üzere testler arasında pek bir farklılık bulunmamaktadır. Buna iki istisna simülasyonun $\alpha = 0,\ \delta = 0.4$ durumunda 150 iterasyon, $\alpha = 0.2,\ \delta = 0.8$ durumunda 173 iterasyon devam etmesidir. Bu iki değer bir düzen göstermedikleri için α, δ değerleri seçilimi için fazla bilgi vermezler. Ek olarak bu iki değerin beklenti hata performansları diğer testlere oldukça yakın, hatta birçoğundan düşüktür.

Açının sınırda olması, sistemin toparlanmasını fazlasıyla zorlaştırmaktadır. Sistem açıyı toparlamakta yeterli cevabı veremez. Bu durum konumun sınırda olması durumunda farklı olup, doğru α, δ değerlerinin seçilmesiyle sistem durum değişkenlerini rahat bir konuma getirip simülasyonu tamamlayabilmektedir.

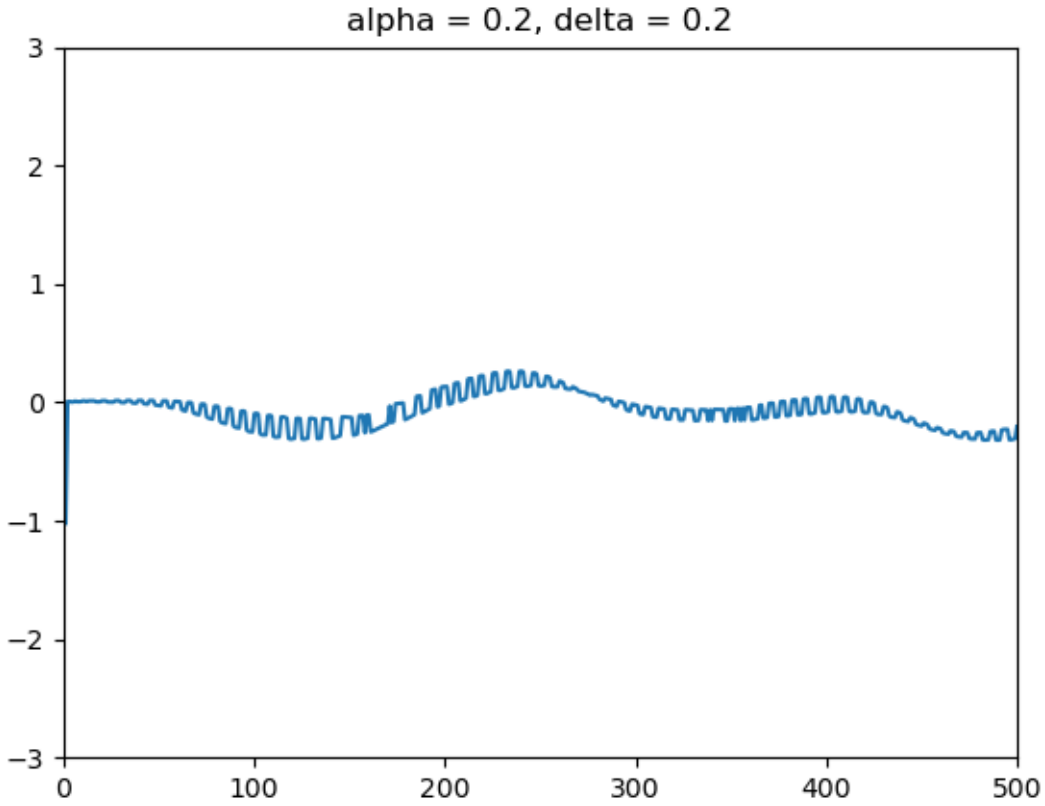
Test sonuçları incelendiğinde için α, δ değerleri için en değerli bilgiyi konumun sınırda olduğu testin verdiği görülür ve bir sonraki test için $\alpha = 0.2,\ \delta = 0.2$ değerleri seçilir.

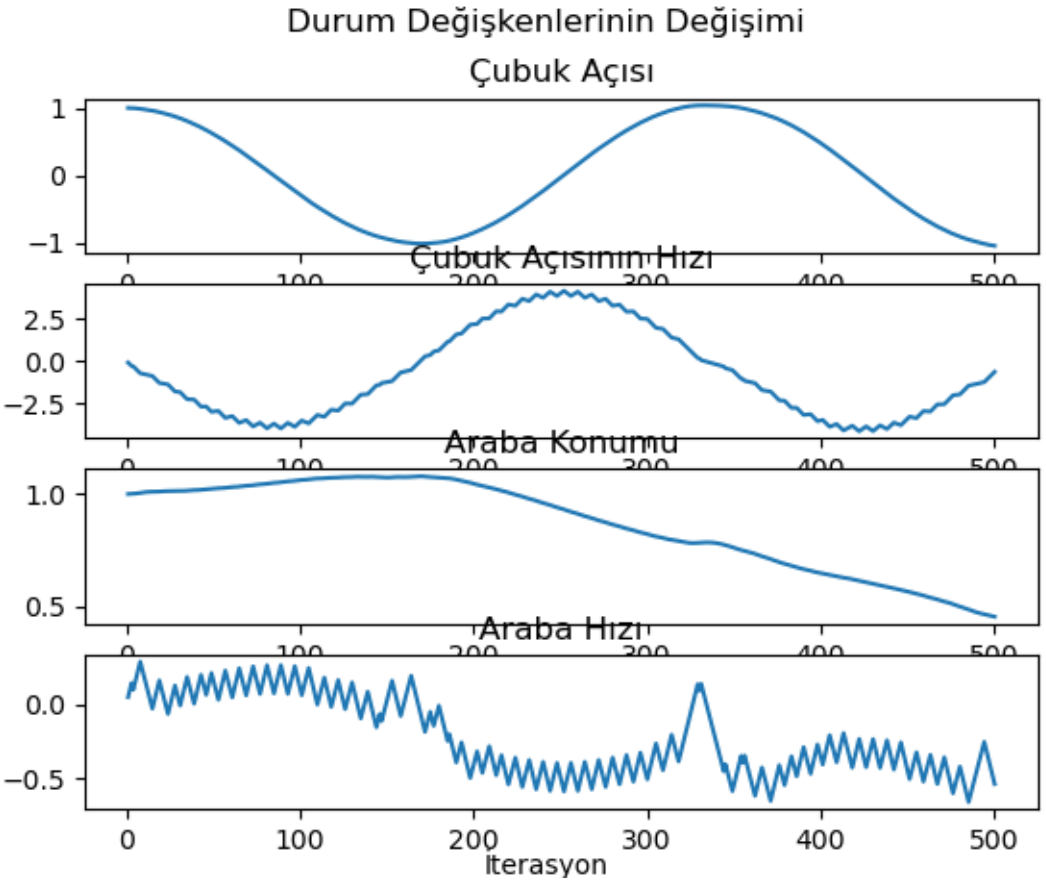
Soru 1: Pekiştirmeli Öğrenme – Test Sonuçları

Aktivasyon Fonksiyonu Seçimi:

ASE'nin kullanacağı aktivasyon fonksiyonu için $tanh(-x)$ ve $sgn(-x)$ seçenekleri arasındaki farklılık incelenecektir. En iyi performansı verdiği gözlenen $\alpha = 0.2,\ \delta = 0.2$ değer çifti seçilir. Merkezi durumdaki başlangıç koşulları kullanılacaktır.

Tanh seçilmesi durumunda:

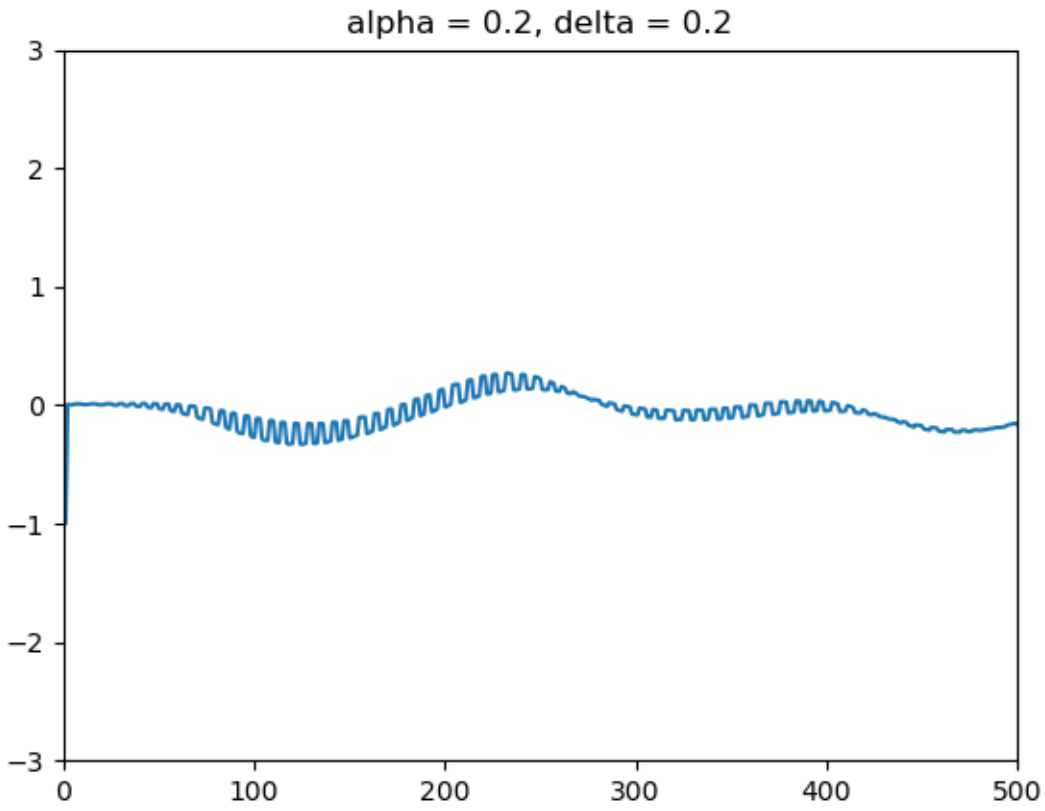


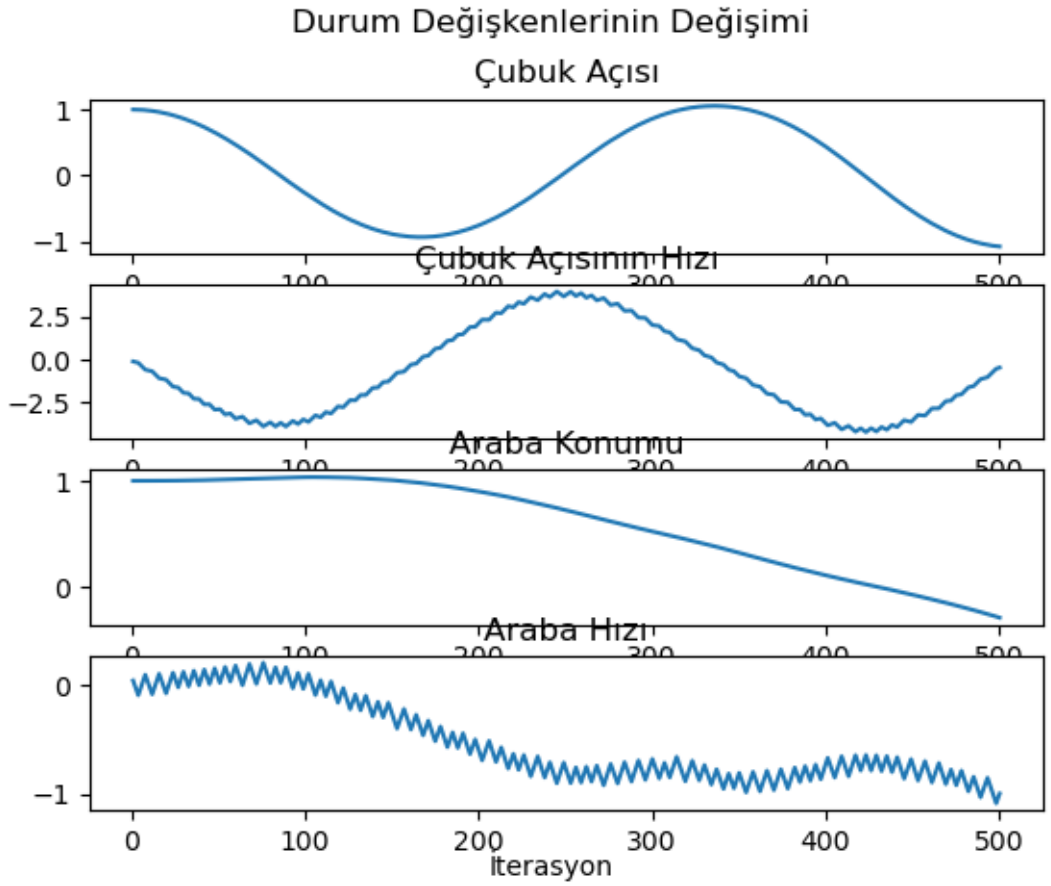


Soru 1: Pekiştirmeli Öğrenme – Test Sonuçları

Araba, konumunu 0 noktasına yaklaştırmış ve üstündeki çubuğu dengede tutmuştur.

Signum seçilmesi durumunda:





Soru 1: Pekiştirmeli Öğrenme – Test Sonuçları

Araba ve çubuk, signum fonksiyonu seçildiğinde de dengede kalmışlardır. Beklenti hatasındaki değişimin çok az bir farkla da olsa 0'a daha yakın ilerlediği görülür. Ayrıca sistem daha hızlı ve daha stabil kararlar almaktadır. Bu özellikle arabanın hızındaki değişimlerde görülür. Bu problem için signum fonksiyonunun daha iyi bir performans sergilediği görülebilir.