T.C.

ONDOKUZ MAYIS ÜNİVERSİTESİ

MÜHENDİSLİK FAKÜLTESİ

BİLGİSAYAR MÜHENDİSLİĞİ BÖLÜMÜ

MAKİNE ÖĞRENİMİ DERSİ

DÖNEM SONU PROJESİ

YAPAY SİNİR AĞLARI

ÖĞRENCİ

Doğukan KALENDER

19060419

Proje Kodunun Açıklamaları:

```python
import numpy as np
import random

random.seed(2)

# Sigmoid aktivasyon fonksiyonu
def sigmoid(x):
    return 1 / (1 + np.exp(-x))

# Sigmoid aktivasyon fonksiyonunun türevi
def sigmoid_derivative(x):
    return x * (1 - x)
```

Projeyi yaparken **numpy** ve **random** kütüphaneleri kullanılmıştır. Random kütüphanesinin kullanım nedeni, gerekli testler ve denemeler için verilen rastgele değerlerin karşılaştırılabilmesi için istikrar sağlamaktır. Yazılan **sigmoid** fonksiyonu ise ileri yayılım esnasında giriş değerlerine sigmoid aktivasyonu uygular. Hemen altındaki **sigmoid_derivative** ise geri yayılım sırasında değişimi ölçmek ve hatayı bulmak için sigmoid fonksiyonunun türevi alınmış halidir.

```python
# Yapay sinir ağı sınıfı
class NeuralNetwork:
    def __init__(self, input_size, hidden_size1, hidden_size2, output_size):
        # Ağırlıkları rastgele başlat
        self.weights_input_hidden1 = np.random.rand(input_size, hidden_size1)
        self.weights_hidden1_hidden2 = np.random.rand(hidden_size1, hidden_size2)
        self.weights_hidden2_output = np.random.rand(hidden_size2, output_size)

    def forward(self, inputs):
        # İleri yayılım (forward propagation)
        self.hidden1_input = np.dot(inputs, self.weights_input_hidden1)
        self.hidden1_output = sigmoid(self.hidden1_input)

        self.hidden2_input = np.dot(self.hidden1_output, self.weights_hidden1_hidden2)
        self.hidden2_output = sigmoid(self.hidden2_input)

        self.output_layer_input = np.dot(self.hidden2_output, self.weights_hidden2_output)
        self.output = sigmoid(self.output_layer_input)

        return self.output
```

Kodun bu kısmında ise bir sinir ağı sınıfı oluşturularak sınıfımıza **__init__, forward, backward** ve **train** fonksiyonları eklendi. Bu fonksiyonları açıklayacak olursak __init__ fonksiyonu sınıfın bir örneği oluşturulduğunda ilk çağırılan metottur. Ağırlıkların başlangıç değerlerini rastgele atar ve ağın katmanlarını tanımlar.

İkinci fonksiyonumuz forward'ın görevi verilen giriş verilerini kullanarak ağın çıkışını hesaplamaktır. İleri yayılım, ağın öğrenmiş olduğu ağırlıklar ve sigmoid fonksiyonu aracılığıyla gerçekleşir. Kısaca açıklamak gerekirse giriş değerleri ve ağırlıklar çarpılır, ardından bu değer üzerinden sigmoid aktivasyon fonksiyonuna uygulanarak çıkış elde edilir.

```python
    def backward(self, inputs, targets, learning_rate):
        # Geri yayılım (backward propagation)
        # Hata hesaplama
        output_error = targets - self.output

        # Çıkış katmanındaki hata ve sigmoid türevi kullanarak delta hesaplama
        output_delta = output_error * sigmoid_derivative(self.output)

        # Gizli katmanlardaki hata ve sigmoid türevi kullanarak delta hesaplama
        hidden2_error = output_delta.dot(self.weights_hidden2_output.T)
        hidden2_delta = hidden2_error * sigmoid_derivative(self.hidden2_output)

        hidden1_error = hidden2_delta.dot(self.weights_hidden1_hidden2.T)
        hidden1_delta = hidden1_error * sigmoid_derivative(self.hidden1_output)

        # Ağırlıkları güncelleme
        self.weights_hidden2_output += self.hidden2_output.T.dot(output_delta) * learning_rate
        self.weights_hidden1_hidden2 += self.hidden1_output.T.dot(hidden2_delta) * learning_rate
        self.weights_input_hidden1 += inputs.T.dot(hidden1_delta) * learning_rate
```

Üçüncü fonksiyonumuz backward'ın görevi ağın eğitimi sırasında geri yayılım algoritması kullanarak ağı güncelleme işlemini gerçekleştirir. Ağın çıkışındaki hatayı geri yayarak ağırlıklar güncellenmesini ve daha optimal değerler verilmesini amaçlar.

```python
def train(self, inputs, targets, epochs):
    for epoch in range(epochs):
        # Ağı eğitme
        for i in range(len(inputs)):
            input_data = np.array([inputs[i]])
            target_data = np.array([targets[i]])

            # İleri ve geri yayılım
            output = self.forward(input_data)
            self.backward(input_data, target_data, 1 - (epoch/epochs))

            # Belirlenen epoch'ta bir hata kontrolü, training'de kullanılan 70 tanesinin hatası
            if epoch % 10000 == 0:
                error = np.mean(np.abs(target_data - output))
                print(f"Epoch {epoch}, Error: {error}\n")
```

Sınıfımızdaki dördüncü ve son fonksiyonun amacı ise belirli bir veri kümesi üzerinde belirli bir epoach sayısı uygulanarak modelimizin eğitilmesini amaçlar. Her epoach için ileri ve geri yayılım algoritmaları uygulanarak ağırlıkların güncellenmesi ve daha optimal değerler verilmesini sağlar.

```
71  input_size = 2
72  hidden_size1 = 4
73  hidden_size2 = 3
74  output_size = 1
75
76  nn = NeuralNetwork(input_size, hidden_size1, hidden_size2, output_size)
77
78  # Eğitim verileri
79  inputs = np.array([[0.3,1.0],[0.3,0.2],[0.3,0.1],[0.1,0.3],[0.9,0.8],[0.9,0.2],[0.1,0.4],[1.0,0.3],[0.4,0.2]
80  targets = np.array([[0.3],[0.06],[0.03],[0.03],[0.72],[0.18],[0.04],[0.3],[0.08],[0.04],[0.3],[0.42],[0.24],
81
82  # Ağı eğitme
83  nn.train(inputs, targets, epochs= 50000)
84
85  # Test verisi
86  test_input = np.array([[0.9,0.7],[0.5,0.7],[0.5,0.1],[0.5,0.2],[0.4,0.8],[0.9,0.3],[0.6,0.4],[0.1,0.5],[1,0.
87  test_output = np.array([63,35,5,10,32,27,24,5,50,20,80,36,18,28,42,40,45,49,81,60,12,80,2,14,1,8,25,20,12,16
88  result = nn.forward(test_input)
89  normalize_result = result * 100 # Normalize etme
90
91  for i in range(len(test_input)):
92      test_error = np.mean(np.abs(test_output[i] - normalize_result[i]))
93      print(f"\nTest output: {test_output[i]},\nNetwork output: {normalize_result[i]},\nError: {test_error}")
```

Kodumuzun kalan kısmı ekran görüntüsündeki gibidir. Bu kısımda 71-74 satıları arasında modelin katmanları ve düğümleri tanımlanmıştır ve 76. satırda modelimiz oluşturularak nn'e atanmıştır. 79-80 satırlarında toplam yetmiş tane olmak üzere eğitim verileri ve bu verilerin hedeflenen çıkış değerleri verilmiştir. 83. satırda modelimiz train fonksiyonu ile 50.000 epoach kadar tekrar ederek eğitime başlatılmıştır. 85-87 satırları arasında toplam otuz tane olmak üzere test verileri ve bu verilerin hedeflenen çıkış değerleri verilmiştir. 88 numaralı satırda bu test verileri eğitilmiş modelde ileri yayılım algoritması uygulanarak elde edilen değerler bir değişkene atanmıştır ve ardından 89 numaralı satırda normalize edilerek 10'a bölünmüş girdileri eski haline getimek için sonuçlar 100 ile çarpılmıştır. Son olarak 91-93 satırlarında test verileri üstündeki başarı oranını görmek için hata oranı hesaplanarak yazdırılmıştır.