BURSA TEKNİK ÜNİVERSİTESİ

LİSANSÜSTÜ EĞİTİM ENSTİTÜSÜ

BİLGİSAYAR MÜHENDİSLİĞİ (T) BÖLÜMÜ

BİLGİSAYAR OYUNLARDA YAPAY ZEKA

Ödev-6 Raporu

Github: https://github.com/MoussaBane/BOYZ-Artificial-Neural-Networks-ANNs

MOUSSA BANE

24435004029

Giriş:

Bu rapor, Unity kullanarak bir Yapay Sinir Ağı (ANN) oluşturan ve XOR fonksiyonunu eğiten bir projeyi, ilk kez kullanan birinin bile anlayıp tekrar edebileceği şekilde detaylandırır.

1. Proje Ayarları

Unity'de Yeni Bir Proje Oluşturun:

- Unity Hub'ı açın ve yeni bir 2D veya 3D proje oluşturun. Projenin adı "XOR ANN Projesi" olabilir.
- **Proje Türü:** 3D önerilir, ancak 2D de kullanılabilir.

Boş Bir GameObject Oluşturun:

- **Hierarchy** penceresinde sağ tıklayın ve "Create Empty" seçeneğini seçin.
- Bu nesneyi **Brain** olarak yeniden adlandırın.

Script Dosyalarını Ekleyin:

Proje klasöründe bir **Scripts** klasörü oluşturun. Aşağıdaki 4 C# script'ini oluşturun ve içeriğini ekleyin:

- Neuron_sc.cs
- Layer_sc.cs
- ANN_sc.cs
- Brain_sc.cs

2. Kodların Açıklaması ve Eklenmesi

Neuron_sc.cs : Bu sınıf, tek bir nöronun yapı taşıdır. Giriş sayısını, ağırlıkları, bias'ı ve çıktıları içerir.

```
public class Neuron_sc
{
    // Nöronun sahip olduğu giriş sayısı.
    public int numberOfInputs;

    // Nöronun bias değeri; genelde ağırlıkları dengelemek için kullanılır.
    public double bias;

    // Nöronun hesaplanan çıktısı.
    public double output;

    // Hata gradyanı; genelde geri yayılım (backpropagation) sırasında öğrenme için kullanılır.
    public double errorGradient;

    // Nöronun her bir girişi için ayrı ayrı ağırlık değerlerini tutan liste.
    public List<double> weights = new List<double>();

    // Nörona verilen girişlerin değerlerini tutan liste.
    public List<double> inputs = new List<double>();

    // Kurucu metod; bir Neuron_sc nesnesi oluşturulduğunda çağrılır.
    // 'numberOfInputs' parametresi, nöronun kaç giriş alacağını belirtir.
    public Neuron_sc(int numberOfInputs)
    {
        // Bias değeri -1.0 ile 1.0 arasında rastgele bir değer olarak atanıyor.
        bias = UnityEngine.Random.Range(-1.0f, 1.0f);

        // Parametreden gelen giriş sayısı, sınıfın numberOfInputs değişkenine atanıyor.
        this.numberOfInputs = numberOfInputs;

        // Giriş sayısı kadar rastgele ağırlık oluşturuluyor.
        // Her giriş için bir ağırlık değeri gereklidir.
        for (int i = 0; i < numberOfInputs; i++)
        {
            weights.Add(UnityEngine.Random.Range(-1.0f, 1.0f));
        }
    }
}
```

Layer_sc.cs : Bu sınıf, bir nöron katmanını temsil eder ve nöron listesini içerir.

```
public class Layer_sc
{
    // Katmandaki nöron sayısını tutar.
    public int numberOfNeurons;

    // Katmandaki nöronları temsil eden bir liste. Her eleman bir Neuron_sc nesnesidir.
    public List<Neuron_sc> neurons = new List<Neuron_sc>();

    // Kurucu metod; bir Layer_sc nesnesi oluşturulduğunda çağrılır.
    // 'numberOfNeurons' katmanda kaç nöron olacağını, 'numberOfInputs' ise her nöronun kaç giriş alacağını belirtir.
    public Layer_sc(int numberOfNeurons, int numberOfInputs)
    {
        // Parametreden gelen nöron sayısı, sınıfın değişkenine atanır.
        this.numberOfNeurons = numberOfNeurons;

        // Nöron sayısı kadar döngü oluşturulur.
        for (int i = 0; i < numberOfNeurons; i++)
        {
            // Her bir nöron için yeni bir Neuron_sc nesnesi oluşturulur ve listeye eklenir.
            // 'numberOfInputs', bu nöronun kaç giriş alacağını belirtir.
            neurons.Add(new Neuron_sc(numberOfInputs));
        }
    }
}
```

ANN_sc.cs : Bu sınıf, yapay sinir ağını oluşturur. XOR işlevini eğitmek ve çalıştırmak için gerekli tüm işlevleri içerir.

```
// Yapıcı metod: Sinir ağı yapısını başlatır.
    public ANN_sc(int numberOfInputs, int numberOfOutputs,
                  int numberOfHiddenLayers,
                  int numberOfNeuronsPerHiddenLayer,
                  double alpha)
    {
        this.numberOfInputs = numberOfInputs;                       // Giriş sayısını ayarla.
        this.numberOfOutputs = numberOfOutputs;                     // Çıkış sayısını ayarla.
        this.numberOfHiddenLayers = numberOfHiddenLayers;           // Gizli katman sayısını ayarla.
        this.numberOfNeuronsPerHiddenLayer = numberOfNeuronsPerHiddenLayer; // Gizli katman nöron sayısını ayarla.
        this.alpha = alpha;                                         // Öğrenme oranını ayarla.

        // Gizli katman varsa oluştur.
        if (numberOfHiddenLayers > 0)
        {
            layers.Add(new Layer_sc(numberOfNeuronsPerHiddenLayer, numberOfInputs)); // İlk gizli katman.

            // Ek gizli katmanları oluştur.
            for (int i = 0; i < numberOfHiddenLayers - 1; i++)
            {
                layers.Add(new Layer_sc(numberOfNeuronsPerHiddenLayer, numberOfNeuronsPerHiddenLayer));
            }

            // Çıkış katmanını oluştur.
            layers.Add(new Layer_sc(numberOfOutputs, numberOfNeuronsPerHiddenLayer));
        }
        else
        {
            // Gizli katman yoksa girişten direkt çıkış katmanına bağla.
            layers.Add(new Layer_sc(numberOfOutputs, numberOfInputs));
        }
    }
```

```
// Ağın çalıştırılması: Giriş değerlerini alır, işler ve çıktıları döndürür.
    public List<double> Run(List<double> inputValues, List<double> desiredOutput)
    {
        List<double> inputs = new List<double>();   // Girişler.
        List<double> outputs = new List<double>();  // Çıktılar.

        // Giriş sayısını kontrol et.
        if (inputValues.Count != numberOfInputs)
        {
            Debug.Log("ERROR: Number of Inputs must be " + numberOfInputs);
            return outputs; // Hatalı giriş durumu.
        }

        inputs = new List<double>(inputValues); // Girişleri ayarla.

        // Her katmanı sırayla işle.
        for (int i = 0; i < numberOfHiddenLayers + 1; i++)
        {
            if (i > 0)
            {
                inputs = new List<double>(outputs); // Bir önceki katmanın çıktısını giriş yap.
            }
            outputs.Clear(); // Çıktıları temizle.

            // Her nöron için işlemleri uygula.
            for (int j = 0; j < layers[i].numberOfNeurons; j++)
            {
                double N = 0; // Nöronun toplam girdisi.
                layers[i].neurons[j].inputs.Clear(); // Nöron girişlerini temizle.

                // Nöron ağırlıklarını ve girdileri hesapla.
                for (int k = 0; k < layers[i].neurons[j].numberOfInputs; k++)
                {
```

```
                // Nöron ağırlıklarını ve girdileri hesapla.
                for (int k = 0; k < layers[i].neurons[j].numberOfInputs; k++)
                {
                    layers[i].neurons[j].inputs.Add(inputs[k]); // Girişi ekle.
                    N += layers[i].neurons[j].weights[k] * inputs[k]; // Ağırlık ve giriş çarpımı.
                }

                N -= layers[i].neurons[j].bias; // Bias çıkar.
                layers[i].neurons[j].output = ActivationFunction(N); // Aktivasyon fonksiyonu uygula.
                outputs.Add(layers[i].neurons[j].output); // Çıkışı ekle.
            }
        }

        UpdateWeights(outputs, desiredOutput); // Ağırlıkları güncelle.

        return outputs; // Sonuçları döndür.
    }
```

```
// Geri yayılım algoritmasıyla ağırlıkları günceller.
    void UpdateWeights(List<double> outputs, List<double> desiredOutput)
    {
        double error; // Hata değeri.

        // Katmanları geriye doğru işle.
        for (int i = numberOfHiddenLayers; i >= 0; i--)
        {
            for (int j = 0; j < layers[i].numberOfNeurons; j++)
            {
                if (i == numberOfHiddenLayers)
                {
                    // Çıkış katmanındaki hata hesaplama.
                    error = desiredOutput[j] - outputs[j];
                    layers[i].neurons[j].errorGradient = outputs[j] * (1 - outputs[j]) * error;
                }
                else
                {
                    // Gizli katmandaki hata hesaplama.
                    layers[i].neurons[j].errorGradient = layers[i].neurons[j].output *
                                                         (1 - layers[i].neurons[j].output);
                    double errorGradSum = 0;

                    // Sonraki katmandan hata katkısı topla.
                    for (int p = 0; p < layers[i + 1].numberOfNeurons; p++)
                    {
                        errorGradSum += layers[i + 1].neurons[p].errorGradient *
                                        layers[i + 1].neurons[p].weights[j];
                    }
                    layers[i].neurons[j].errorGradient *= errorGradSum;
                }
```

```
                // Ağırlıkları güncelle.
                for (int k = 0; k < layers[i].neurons[j].numberOfInputs; k++)
                {
                    if (i == numberOfHiddenLayers)
                    {
                        error = desiredOutput[j] - outputs[j];
                        layers[i].neurons[j].weights[k] += alpha * layers[i].neurons[j].inputs[k] *
                                                           error;
                    }
                    else
                    {
                        layers[i].neurons[j].weights[k] += alpha * layers[i].neurons[j].inputs[k] *
                                                           layers[i].neurons[j].errorGradient;
                    }
                }

                // Bias güncelle.
                layers[i].neurons[j].bias += alpha * -1 * layers[i].neurons[j].errorGradient;
            }
        }
    }
```

```
// Aktivasyon fonksiyonu: Sigmoid kullanır.
    double ActivationFunction(double value)
    {
        return Sigmoid(value);
    }

    // Binary step aktivasyon fonksiyonu.
    double Step(double value)
    {
        if (value < 0) return 0;
        else return 1;
    }

    // Sigmoid aktivasyon fonksiyonu.
    double Sigmoid(double value)
    {
        double k = (double)System.Math.Exp(value);
        return k / (1.0f + k);
    }
}
```

Brain_sc.cs : Bu script, XOR işlevini eğitmek için ağı kurar ve çalıştırır.

```
public class Brain_sc : MonoBehaviour
{
    ANN_sc ann;                     // Yapay sinir ağı nesnesi.
    double sumSquareError = 0;      // Toplam kare hata değişkeni.

    void Start()
    {
        // Ağ yapısının parametreleri.
        int numberOfInputs = 2;                     // Giriş sayısı.
        int numberOfOutputs = 1;                    // Çıkış sayısı.
        int numberOfHiddenLayers = 1;               // Gizli katman sayısı.
        int numberOfNeuronsPerHiddenLayer = 2;      // Her gizli katmandaki nöron sayısı.
        double alpha = 0.8;                         // Öğrenme oranı.

        int epoch = 1000;                           // Eğitim döngüsü sayısı.

        // Yapay sinir ağı nesnesini oluştur.
        ann = new ANN_sc(numberOfInputs, numberOfOutputs,
                         numberOfHiddenLayers,
                         numberOfNeuronsPerHiddenLayer,
                         alpha);

        List<double> result; // Çıkış sonuçlarını tutmak için liste.

        // Eğitim süreci.
        for (int i = 0; i < epoch; i++)
        {
            sumSquareError = 0; // Toplam kare hatayı sıfırla.

            // Eğitim verileriyle ağı eğit ve hataları hesapla.
            result = Train(1, 1, 0);
            sumSquareError += Mathf.Pow((float)result[0] - 0, 2); // Hata hesaplama.
```

```
33
34              result = Train(1, 0, 1);
35              sumSquareError += Mathf.Pow((float)result[0] - 1, 2);
36
37              result = Train(0, 1, 1);
38              sumSquareError += Mathf.Pow((float)result[0] - 1, 2);
39
40              result = Train(0, 0, 0);
41              sumSquareError += Mathf.Pow((float)result[0] - 0, 2);
42          }
43
44          Debug.Log("SSE: " + sumSquareError); // Toplam kare hatayı yazdır.
45
46          // Eğitim sonrası doğruluk testi.
47          result = Train(1, 1, 0);
48          Debug.Log(" 1 1 " + result[0]);
49
50          result = Train(1, 0, 1);
51          Debug.Log(" 1 0 " + result[0]);
52
53          result = Train(0, 1, 1);
54          Debug.Log(" 0 1 " + result[0]);
55
56          result = Train(0, 0, 0);
57          Debug.Log(" 0 0 " + result[0]);
58      }
59
60      // Eğitim fonksiyonu: Ağın girişlerini ve beklenen çıkışını işler.
61      List<double> Train(double input1, double input2, double output)
62      {
63          List<double> inputs = new List<double>() { input1, input2 };    // Girişleri listele.
64          List<double> outputs = new List<double>() { output };          // Beklenen çıktıyı listele.
65          return (ann.Run(inputs, outputs));                             // Sinir ağını çalıştır ve çıktıla
    rı döndür.
66      }
67  }
```

3. Kodların Çalıştırılması

Brain Script'ini Bağlayın:

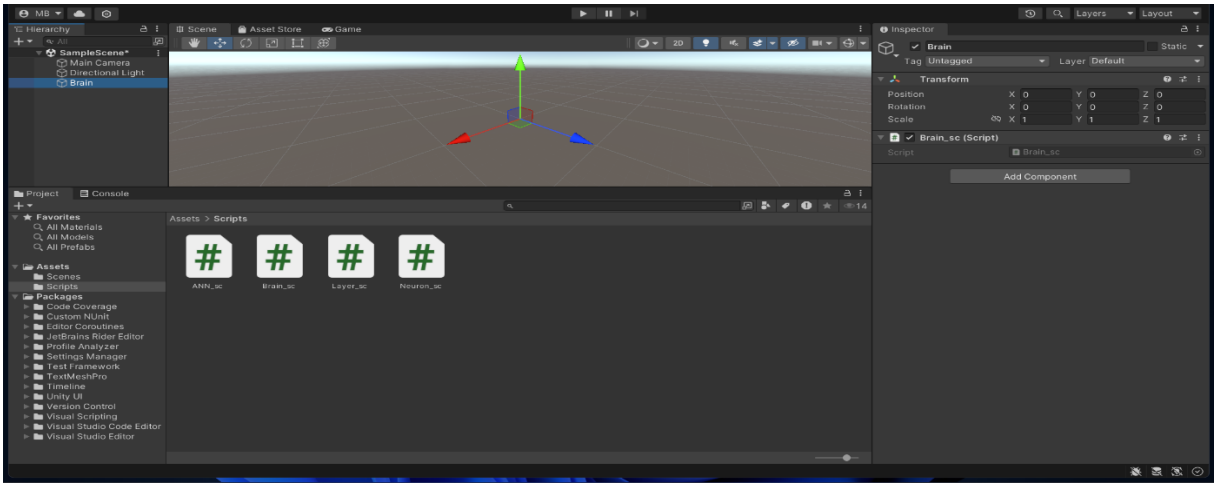
- **Brain** adlı boş nesneye Brain_sc script'ini ekleyin.
- Brain_sc içindeki parametreleri değiştirerek XOR eğitimini inceleyebilirsiniz.

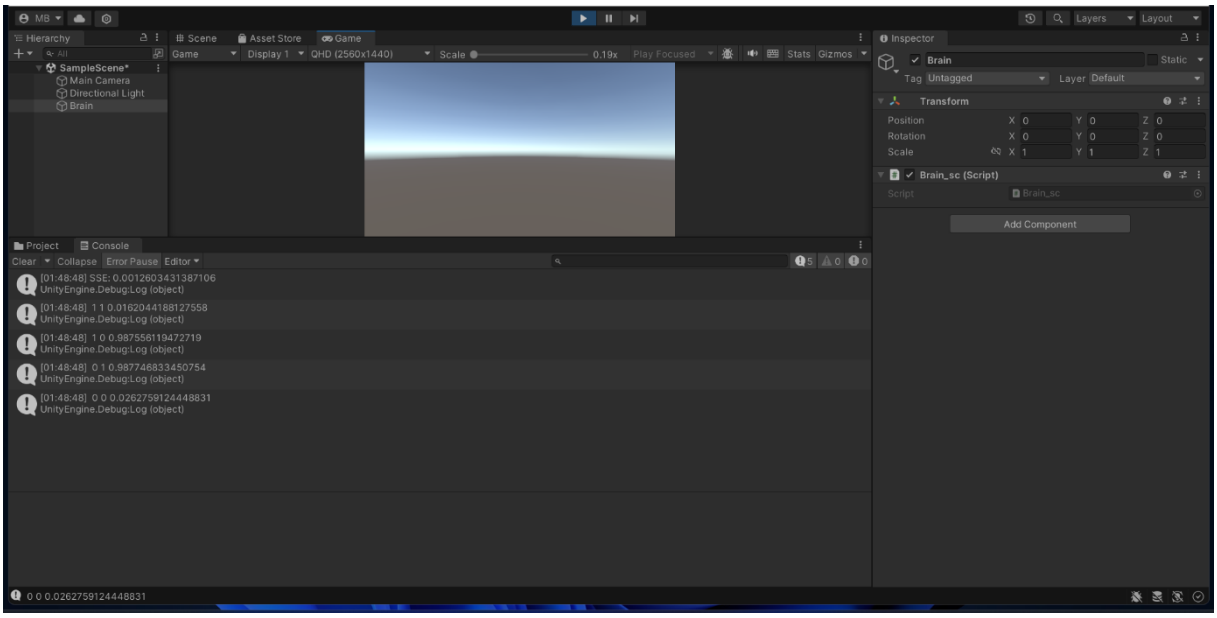
Oyunu Çalıştırın:

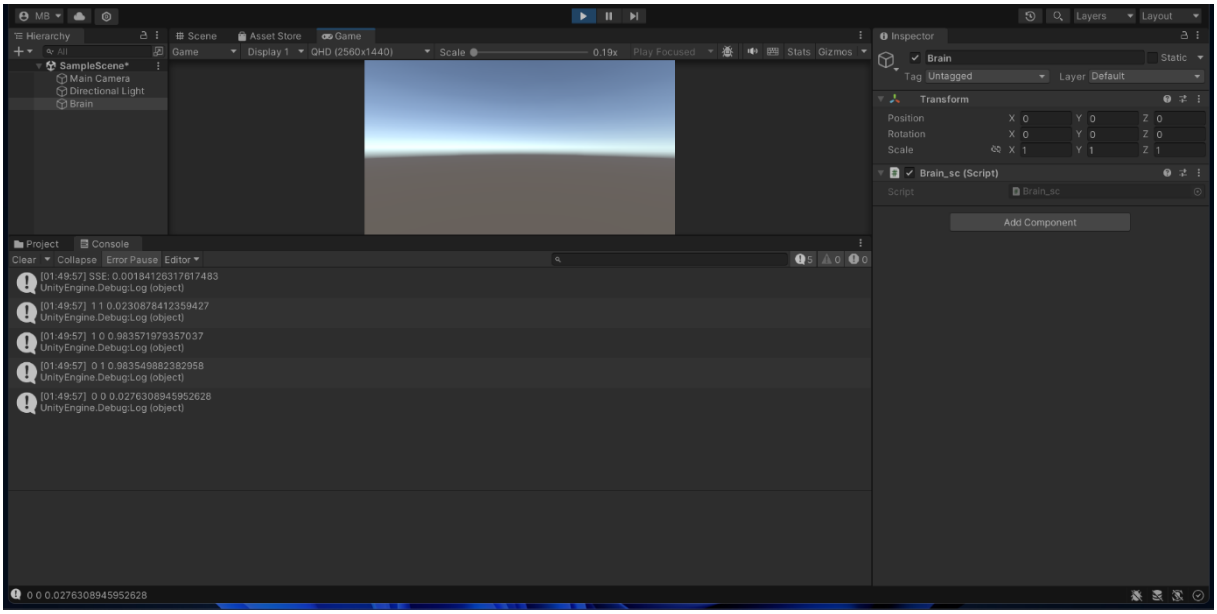
- Unity’de "Play" butonuna basın.
- Konsolda "Sum of Squared Error" ve XOR sonuçları yazdırılacaktır.

4. Ekran Görüntüleri ve Sonuçlar

Eğitim sonuçlarının konsol çıktıları aşağıdaki gibidir:




Hierarchy
SampleScene*
Main Camera
Directional Light
Brain
Scene
Asset Store
Game
Display 1
QHD (2560x1440)
Scale
0.19x
Play Focused
Stats
Gizmos
Inspector
Brain
Static
Tag Untagged
Layer Default
Transform
Position
Rotation
Scale
Brain_sc (Script)
Script
Brain_sc
Add Component
Project
Console
Clear
Collapse
Error Pause
Editor
[01:48:48] SSE: 0.0012603431387106
UnityEngine.Debug:Log (object)
[01:48:48] 1 1 0.01620441881275558
UnityEngine.Debug:Log (object)
[01:48:48] 1 0 0.987556119472719
UnityEngine.Debug:Log (object)
[01:48:48] 0 1 0.987746833450754
UnityEngine.Debug:Log (object)
[01:48:48] 0 0 0.0262759124448831
UnityEngine.Debug:Log (object)
0 0 0.0262759124448831


Hierarchy
SampleScene*
Main Camera
Directional Light
Brain
Scene
Asset Store
Game
Display 1
QHD (2560x1440)
Scale
0.19x
Play Focused
Stats
Gizmos
Inspector
Brain
Static
Tag Untagged
Layer Default
Transform
Position
Rotation
Scale
Brain_sc (Script)
Script
Brain_sc
Add Component
Project
Console
Clear
Collapse
Error Pause
Editor
[01:49:57] SSE: 0.00184126317617483
UnityEngine.Debug:Log (object)
[01:49:57] 1 1 0.0230878412359427
UnityEngine.Debug:Log (object)
[01:49:57] 1 0 0.983571979357037
UnityEngine.Debug:Log (object)
[01:49:57] 0 1 0.983549882382958
UnityEngine.Debug:Log (object)
[01:49:57] 0 0 0.0276308945952628
UnityEngine.Debug:Log (object)
0 0 0.0276308945952628