Національний технічний університет України

«Київський політехнічний інститут імені Ігоря Сікорського»

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

**Лабораторна робота №1**

з дисципліни " Програмні засоби проектування та реалізаціїї нейромережевих систем"

Тема: "Парцептрон"

Виконав: Перевірив:

студент групи ІП-93 Шимкович Володимир

Домінський Валентин Миколайович

Олексійович

Київ 2022

Зміст:

[Мета: 3](#_Toc96437032)

[Хід роботи: 3](#_Toc96437033)

[Вихідний код 3](#_Toc96437034)

[Результат роботи: 8](#_Toc96437035)

[Висновки: 8](#_Toc96437036)

## Мета:

Написати програму, що реалізує нейронну мережу Парцептрон та навчити її виконувати функцію XOR

## Хід роботи:

## Вихідний код

Program:

using Lab1;

using SML.Matrices;

namespace Labs;

internal class Program

{

private static readonly double[,] s\_input = new double[,]

{

{ 0, 0 },

{ 0, 1 },

{ 1, 0 },

{ 1, 1 }

};

private static readonly double[,] s\_outputs =

{

{ 0 },

{ 1 },

{ 1 },

{ 0 }

};

private static readonly double[,] s\_xTest = { { 1, 1 } };

private static readonly double[,] s\_xTest2 = { { 0, 0 } };

private static readonly double[,] s\_xTest3 = { { 0, 1 } };

private static readonly double[,] s\_xTest4 = { { 1, 0 } };

private static void Main(string[] args)

{

Lab1();

}

private static void Lab1()

{

Perceptron perceptron = new(s\_input);

perceptron.Start();

Console.WriteLine("Predictions before training:\n");

RunPredictions(perceptron);

perceptron.Train(s\_input, s\_outputs, 10000);

Console.WriteLine("//////////////////////////////\n");

Console.WriteLine("Predictions after training:\n");

RunPredictions(perceptron);

}

private static void RunPredictions(Perceptron perceptron)

{

Matrix firstPrediction = new(perceptron.Predict(s\_xTest));

Matrix secondPrediction = new(perceptron.Predict(s\_xTest2));

Matrix thirdPrediction = new(perceptron.Predict(s\_xTest3));

Matrix fourthPrediction = new(perceptron.Predict(s\_xTest4));

Console.WriteLine("Prediction for 1, 1 is:\n");

Console.WriteLine(firstPrediction.ToString());

Console.WriteLine("Prediction for 0, 0 is:\n");

Console.WriteLine(secondPrediction.ToString());

Console.WriteLine("Prediction for 0, 1 is:\n");

Console.WriteLine(thirdPrediction.ToString());

Console.WriteLine("Prediction for 1, 0 is:\n");

Console.WriteLine(fourthPrediction.ToString());

}

}

Perceptron:

using SML.Matrices;

namespace Lab1;

public class Perceptron

{

#region Fields

public double[,] Input { get; set; }

public int RunTimes { get; set; } = 10000;

private readonly double \_bias = 0.03;

private readonly Random \_random = new();

private double[,] \_firstLayerWeights = new double[0, 0];

private double[,] \_secondLayerWeights = new double[0, 0];

#endregion Fields

#region Constructors

public Perceptron(double[,] input)

{

Input = input;

}

#endregion Constructors

#region Methods

public void Start()

{

GenerateWeights();

}

private void GenerateWeights()

{

// During the training phase, the network is trained by adjusting

// these weights to be able to predict the correct class for the input.

int firstLayerLength = Input.GetUpperBound(0) + 1;

int secondLayerLength = Input.GetUpperBound(1) + 1;

\_firstLayerWeights = new double[secondLayerLength, firstLayerLength];

for (int i = 0; i < secondLayerLength; i++)

{

for (int j = 0; j < firstLayerLength; j++)

{

\_firstLayerWeights[i, j] = \_random.NextDouble();

}

}

\_secondLayerWeights = new double[firstLayerLength, 1];

for (int i = 0; i < firstLayerLength; i++)

{

for (int j = 0; j < 1; j++)

{

\_secondLayerWeights[i, j] = \_random.NextDouble();

}

}

}

private static double Sigmoid(double x)

{

return 1 / (1 + (float)Math.Exp(-x));

}

private static double SigmoidDerivative(double x)

{

return Sigmoid(x) \* (1 - Sigmoid(x));

}

public double[,] Predict(double[,] xTest)

{

Matrix xTestMatrix = new(xTest);

Matrix firstLayerWeightsMatrix = new(\_firstLayerWeights);

Matrix xTestDotfirstLayerWeights = xTestMatrix.Multiply(firstLayerWeightsMatrix);

double[,] firstLayer = xTestDotfirstLayerWeights.Array;

for (int i = 0; i < xTestDotfirstLayerWeights.Rows; i++)

{

for (int j = 0; j < xTestDotfirstLayerWeights.Columns; j++)

{

firstLayer[i, j] = Sigmoid(firstLayer[i, j]);

}

}

Matrix firstLayerMatrix = new(firstLayer);

Matrix secondLayerWeightsMatrix = new(\_secondLayerWeights);

Matrix firstLayerDotsecondLayerWeights = firstLayerMatrix

.Multiply(secondLayerWeightsMatrix);

double[,] secondLayer = firstLayerDotsecondLayerWeights.Array;

for (int i = 0; i < firstLayerDotsecondLayerWeights.Rows; i++)

{

for (int j = 0; j < firstLayerDotsecondLayerWeights.Columns; j++)

{

secondLayer[i, j] = Sigmoid(secondLayer[i, j]);

}

}

return secondLayer;

}

public void Train(double[,] xTrain, double[,] yTrain, int iterations)

{

for (var k = 0; k < iterations; k++)

{

Matrix xTrainMatrix = new(xTrain);

Matrix firstLayerWeightsMatrix = new(\_firstLayerWeights);

Matrix dotXTrainAndFirstLayerWeigth = xTrainMatrix.Multiply(firstLayerWeightsMatrix);

double[,] firstLayer = dotXTrainAndFirstLayerWeigth.Array;

// Adjusting with bias and activating training

for (int i = 0; i < dotXTrainAndFirstLayerWeigth.Rows; i++)

{

for (int j = 0; j < dotXTrainAndFirstLayerWeigth.Columns; j++)

{

firstLayer[i, j] += \_bias;

firstLayer[i, j] = Sigmoid(firstLayer[i, j]);

}

}

Matrix firstLayerMatrix = new(firstLayer);

Matrix secondLayerWeightsMatrix = new(\_secondLayerWeights);

Matrix dotFirstLayerAndSecondLayerWeights =

firstLayerMatrix.Multiply(secondLayerWeightsMatrix);

double[,] secondLayer = dotFirstLayerAndSecondLayerWeights.Array;

for (int i = 0; i < dotFirstLayerAndSecondLayerWeights.Rows; i++)

{

for (int j = 0; j < dotFirstLayerAndSecondLayerWeights.Columns; j++)

{

secondLayer[i, j] = Sigmoid(secondLayer[i, j]);

}

}

// Calculate the prediction error

double[,] secondLayerError = dotFirstLayerAndSecondLayerWeights.Array;

for (int i = 0; i < dotFirstLayerAndSecondLayerWeights.Rows; i++)

{

for (int j = 0; j < dotFirstLayerAndSecondLayerWeights.Columns; j++)

{

secondLayerError[i, j] = yTrain[i, j] - secondLayer[i, j];

}

}

Matrix secondLayerErrorMatrix = new(secondLayerError);

for (int i = 0; i < secondLayer.GetUpperBound(0)+1; i++)

{

for (int j = 0; j < secondLayer.GetUpperBound(1)+1; j++)

{

secondLayer[i, j] = SigmoidDerivative(secondLayer[i, j]);

}

}

Matrix secondLayerMatrix = new(secondLayer);

Matrix secondLayerDeltaMatrix = secondLayerMatrix.

Hadamard(secondLayerErrorMatrix);

Matrix secondLayerWeightsMatrixTransposed =

secondLayerWeightsMatrix.Transpose();

Matrix firstLayerErrorMatrix = secondLayerDeltaMatrix.

Multiply(secondLayerWeightsMatrixTransposed);

double[,] firstLayerDerivative = firstLayer;

for (int i = 0; i < firstLayerDerivative.GetUpperBound(0) + 1; i++)

{

for (int j = 0; j < firstLayerDerivative.GetUpperBound(1) + 1; j++)

{

firstLayerDerivative[i, j] = SigmoidDerivative(firstLayer[i, j]);

}

}

Matrix firstLayerDerivativeMatrix = new(firstLayerDerivative);

Matrix firstLayerDeltaMatrix = firstLayerDerivativeMatrix.

Hadamard(firstLayerErrorMatrix);

// Adjusting the weights

// Second Weights

Matrix dotFirstLayerAndSecondLayerDelta = firstLayerMatrix.Transpose()

.Multiply(secondLayerDeltaMatrix);

for (int i = 0; i < \_secondLayerWeights.GetUpperBound(0) + 1; i++)

{

for (int j = 0; j < \_secondLayerWeights.GetUpperBound(1) + 1; j++)

{

\_secondLayerWeights[i, j] += dotFirstLayerAndSecondLayerDelta[i, j];

}

}

// First Weights

Matrix xTrainTransposedMatrix = xTrainMatrix.Transpose();

Matrix dotXTrainTransposedAndFirstLayerDeltaMatrix =

xTrainTransposedMatrix.Multiply(firstLayerDeltaMatrix);

for (int i = 0; i < \_firstLayerWeights.GetUpperBound(0) + 1; i++)

{

for (int j = 0; j < \_firstLayerWeights.GetUpperBound(1) + 1; j++)

{

\_firstLayerWeights[i, j] += dotXTrainTransposedAndFirstLayerDeltaMatrix[i, j];

}

}

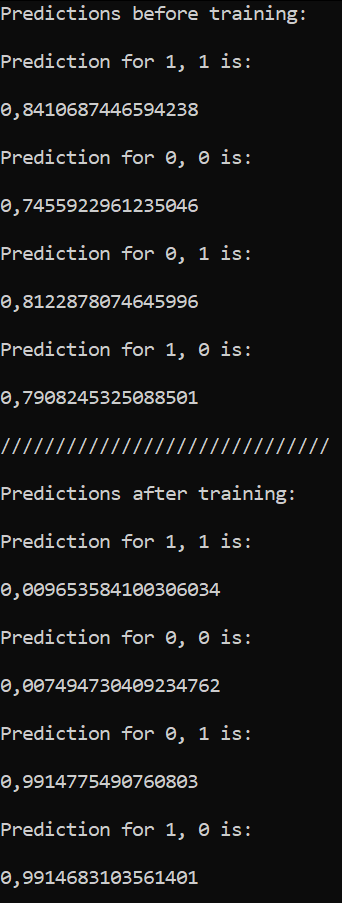
}

}

#endregion Methods

}

## Результат роботи:



## Висновки:

Я дізнався більше інформації про нейронні мережі, як вони навчаються. Створив власний перцептрон та навчив його розв’язувати проблему XOR