**МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ**

**УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ**

**ГОМЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ ИМЕНИ П. О. СУХОГО**

Факультет автоматизированных и информационных систем

Кафедра «Информационные технологии»

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №2

по дисциплине «Введение в нейронные сети»

**Контролируемое обучение нейронных сетей.**

Выполнила: студент гр. ИТИ-21

Тесёлкин Б.А.

Принял: преподаватель

Карабчикова E.A.

Гомель 2020

**Цель работы:** изучить работу с задачами бинарной классификации с учителем с использованием простейших нейронных сетей.

**Задание**

Разработать программное обеспечение для решения задачи бинарной классификации с учителем с использованием простейших нейронных сетей. Архитектуру нейронной сети выбрать самостоятельно. На входном слое не должно быть менее 25 нейронов и не более 36. Количество скрытых слоёв не должно быть менее 1 и более 3-х. Изображения использовать бинаризованные. В качестве первичного алгоритма обучения использовать алгоритм, указанный в таблице, согласно варианта. Дополнить разработанное программное обеспечение методом обратного распространения ошибки (backpropagation). Сравнить с помощью графиков качество и скорость обучения одной и той же сети различными методами.

Вариант 24

Цифры от 0 до 9

Алгоритм: однослойный персептрон (алгоритм Уидроу-Хоффа)

**Ход работы**

В ходе выполнения лабораторной работы была создана структура нейронной системы и подготовлена обучающая выборка, которая будет использоваться и для тестирования нейронной системы.

Для реализации структуры будет использоваться объектно-ориентированный язык С# и среда разработки Visual Studio 2019. Была создана библиотека классов NeuronLibrary и помещены туда созданные классы. Был создан абстрактный класс Neuron, описывающий нейрон, его веса (веса инициализируются рандомно при создании слоя) и входные сигналы, а также выходной сигнал, и реализованы наследуемые классы InputNeuron и HiddenNeuron. Непосредственно за сборку самой нейронной сети отвечает класс NN\_Hoff.

После был разработан графический интерфейс, загружающий обучающую выборку и позволяющий производить обучение нейронной системы с отслеживанием процесса обучения с помощью графика и тестировать обученную нейронную сеть на правильность результатов обучения. Интерфейс приложения представлен на рисунке 1. График обучения нейронной системы представлен на рисунке 2. Результат тестирования нейронной системы на изображении представлено на рисунке 3.

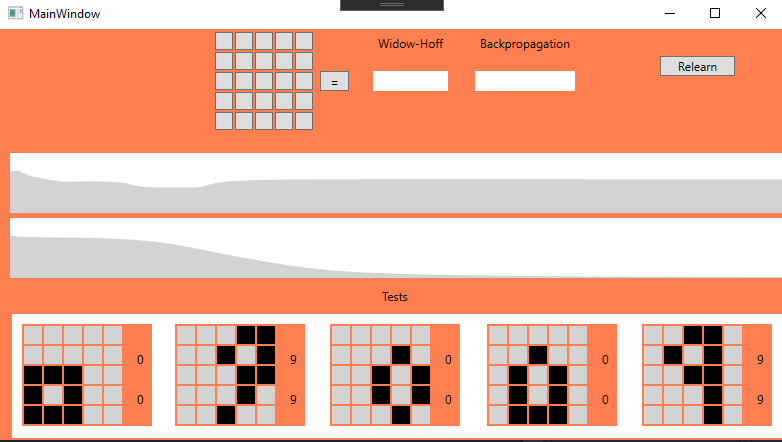


Рисунок 1 – Интерфейс приложения.

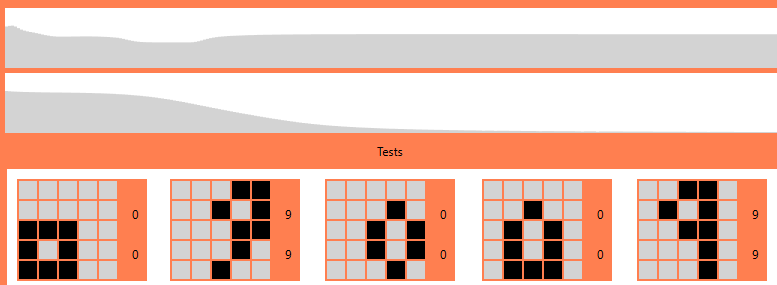


Рисунок 2 – График обучения и тестирования нейронной сети.



Рисунок 3 – Результаты тестирования на изображении.

Для проверки правильности алгоритма используем другую обучающую выборку и структуру сети из лабораторной работы 1. Результаты тестирования алгоритма обучения с коррекцией ошибки представлено на рисунке 5.

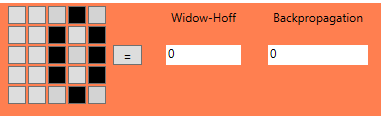


Рисунок 4 – Результаты тестирования на изображении.

Из графика видно, что сеть обучилась.

Таким образом, можно сделать вывод что предложенная заданием обучающая выборка не подходит для обучения сети по алгоритму Уидроу-Хоффа.

**Вывод:** в процессе выполнения данной лабораторной работы была изучена работа с задачами бинарной классификации с учителем с использованием простейших нейронных сетей. Полученные знания были применены для создания нейронной системы с распознаванием бинаризированного изображения.

**Приложение:**

**Листинг Neuron.cs:**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace NeuronLibrary

{

public abstract class Neuron

{

public double OutputValue { get; protected set; }

}

}

**Листинг HiddenNeuron.cs:**

using System;

using System.Linq;

namespace NeuronLibrary

{

public class HiddenNeuron : Neuron

{

public static Random rnd = new Random();

public double Error { get; protected set; }

public double[] Weights;

public Neuron[] Neurons;

public HiddenNeuron(params Neuron[] neurons)

{

Neurons = neurons;

Weights = new double[Neurons.Length];

InitWeights();

}

public void CalcOutput()

{

double summator = 0;

for (int i = 0; i < Neurons.Length; i++)

{

summator += Neurons[i].OutputValue \* Weights[i];

}

OutputValue = 1 / (1 + Math.Exp(-summator));

}

public void CalcError(params Neuron[] neurons)

{

double tempError = 0;

for (int i = 0; i < neurons.Length; i++)

{

if (((HiddenNeuron)neurons[i]).Neurons.Contains(this))

{

tempError += ((HiddenNeuron)neurons[i]).Error \* ((HiddenNeuron)neurons[i]).Weights[((HiddenNeuron)neurons[i]).Neurons.ToList().IndexOf(this)];

}

else

{

throw new ArgumentException("Wrong neuron layer");

}

}

Error = tempError;

}

private void InitWeights()

{

for (int i = 0; i < Weights.Length; i++)

{

Weights[i] = rnd.NextDouble() \* (0.5 + 0.5) - 0.5;

}

}

public void ModifyWeights(double speed)

{

for (int i = 0; i < Weights.Length; i++)

{

Weights[i] = Weights[i] + speed \* Error \* Neurons[i].OutputValue \* (1 / (1 + Math.Exp(-Neurons[i].OutputValue))) \* (1 - (1 / (1 + Math.Exp(-Neurons[i].OutputValue))));

}

}

public void HoffCorrection(double error, double speed)

{

for (int i = 0; i < Neurons.Length; i++)

{

Weights[i] = Weights[i] + Neurons[i].OutputValue \* error \* speed;

}

}

}

}

**Листинг OutputNeuron.cs:**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace NeuronLibrary

{

public class OutputNeuron : HiddenNeuron

{

public OutputNeuron(params Neuron[] neurons) : base(neurons)

{

Error = 1;

}

public void CalcError(double expectedOutput)

{

Error = expectedOutput - OutputValue;

}

}

}

**Листинг InputNeuron.cs:**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace NeuronLibrary

{

public class InputNeuron : Neuron

{

public void SetStartValue(double startValue)

{

OutputValue = startValue;

}

}

}

**Листинг NN\_Hoff.cs:**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace NeuronLibrary

{

public class NN\_Hoff

{

public double speed = 0.1;

public int[] Result;

public int[][] TrainData;

public int[][] TestData;

public delegate void AccountStateHandler(string message);

AccountStateHandler \_del;

public void RegisterHandler(AccountStateHandler del)

{

\_del = del;

}

public InputNeuron[] inputLayer = new InputNeuron[25];

public HiddenNeuron[] hiddenLayer1 = new HiddenNeuron[8];

public HiddenNeuron[] hiddenLayer2 = new HiddenNeuron[8];

public OutputNeuron outputNeuron = new OutputNeuron();

public NN\_Hoff(int[][] trainData, int[][] testData, int[] result)

{

Result = result;

TrainData = trainData;

TestData = testData;

InitLayers();

}

public void InitLayers()

{

for (int i = 0; i < inputLayer.Length; i++)

{

inputLayer[i] = new InputNeuron();

}

for (int i = 0; i < hiddenLayer1.Length; i++)

{

hiddenLayer1[i] = new HiddenNeuron(inputLayer);

}

for (int i = 0; i < hiddenLayer2.Length; i++)

{

hiddenLayer2[i] = new HiddenNeuron(hiddenLayer1);

}

outputNeuron = new OutputNeuron(hiddenLayer2);

}

public double[] Learn(double accurancy)

{

List<double> result = new List<double>();

while (Math.Abs(outputNeuron.Error) > accurancy)

{

for(int iteration = 0; iteration < TrainData.Length; iteration++)

{

for (int i = 0; i < inputLayer.Length; i++)

{

inputLayer[i].SetStartValue(TrainData[iteration][i]);

}

for (int i = 0; i < hiddenLayer1.Length; i++)

{

hiddenLayer1[i].CalcOutput();

}

for (int i = 0; i < hiddenLayer2.Length; i++)

{

hiddenLayer2[i].CalcOutput();

}

outputNeuron.CalcOutput();

outputNeuron.CalcError(Result[iteration]);

if (Math.Abs(outputNeuron.Error) > accurancy) //(Math.Abs(outputNeuron.Error) > accurancy)

{

for (int i = 0; i < hiddenLayer2.Length; i++)

{

hiddenLayer2[i].CalcError(outputNeuron);

}

for (int i = 0; i < hiddenLayer1.Length; i++)

{

hiddenLayer1[i].CalcError(hiddenLayer2);

}

for (int i = 0; i < hiddenLayer1.Length; i++)

{

hiddenLayer1[i].ModifyWeights(speed);

}

for (int i = 0; i < hiddenLayer2.Length; i++)

{

hiddenLayer2[i].ModifyWeights(speed);

}

outputNeuron.ModifyWeights(speed);

}

else

break;

}

result.Add(outputNeuron.Error);

}

return result.ToArray();

}

public double[] LearnHoff()

{

List<double> result = new List<double>();

for (int f = 0; f < 500; f++)

{

for (int iteration = 0; iteration < TrainData.Length; iteration++)

{

for (int i = 0; i < inputLayer.Length; i++)

{

inputLayer[i].SetStartValue(TrainData[iteration][i]);

}

for (int i = 0; i < hiddenLayer1.Length; i++)

{

hiddenLayer1[i].CalcOutput();

}

for (int i = 0; i < hiddenLayer2.Length; i++)

{

hiddenLayer2[i].CalcOutput();

}

outputNeuron.CalcOutput();

outputNeuron.CalcError(Result[iteration]);

outputNeuron.HoffCorrection(outputNeuron.Error, speed);

for (int i = 0; i < hiddenLayer2.Length; i++)

{

hiddenLayer2[i].HoffCorrection(outputNeuron.Error, speed);

}

for (int i = 0; i < hiddenLayer1.Length; i++)

{

hiddenLayer1[i].HoffCorrection(outputNeuron.Error, speed);

}

}

result.Add(outputNeuron.Error);

}

return result.ToArray();

}

public double GetResult(int[] data)

{

for (int i = 0; i < inputLayer.Length; i++)

{

inputLayer[i].SetStartValue(data[i]);

}

for (int i = 0; i < hiddenLayer1.Length; i++)

{

hiddenLayer1[i].CalcOutput();

}

for (int i = 0; i < hiddenLayer2.Length; i++)

{

hiddenLayer2[i].CalcOutput();

}

outputNeuron.CalcOutput();

return outputNeuron.OutputValue;

}

public double[] Test()

{

double[] result = new double[TestData.Length];

for (int iteration = 0; iteration < TestData.Length; iteration++)

{

for (int i = 0; i < inputLayer.Length; i++)

{

inputLayer[i].SetStartValue(TestData[iteration][i]);

}

for (int i = 0; i < hiddenLayer1.Length; i++)

{

hiddenLayer1[i].CalcOutput();

}

for (int i = 0; i < hiddenLayer2.Length; i++)

{

hiddenLayer2[i].CalcOutput();

}

outputNeuron.CalcOutput();

outputNeuron.CalcError(Result[iteration]);

result[iteration] = outputNeuron.OutputValue;

}

return result;

}

}

}

**Листинг MainWindow.cs:**

using NeuronLibrary;

using System;

using System.Collections.Generic;

using System.IO;

using System.Linq;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Media;

using System.Windows.Shapes;

namespace NN\_WPF

{

/// <summary>

/// Логика взаимодействия для MainWindow.xaml

/// </summary>

public partial class MainWindow : Window

{

int[][] trainData;

int[][] testData;

int[] result;

NN\_Hoff backProp;

NN\_Hoff hoff;

public MainWindow()

{

InitializeComponent();

trainData = ReadFile(@"C:\Users\saymo\source\repos\NN\_train.txt");

testData = ReadFile(@"C:\Users\saymo\source\repos\NN\_test.txt");

result = ReadFile(@"C:\Users\saymo\source\repos\NN\_results.txt")[0];

backProp = new NN\_Hoff(trainData, testData, result);

DisplayGraphic(backProp.Learn(0.01), graphicCanvas);

hoff = new NN\_Hoff(trainData, testData, result);

DisplayGraphic(hoff.LearnHoff(), graphicCanvas1);

//hoff.RegisterHandler(DisplayData);

DisplayTests();

}

private void DisplayTests()

{

double[] hoffTestData = hoff.Test();

double[] propTestData = backProp.Test();

for(int i = 0; i < testsArea.Children.Count; i++)

{

for(int j = 0; j < ((Grid)testsArea.Children[i]).Children.Count; j++)

{

if(j < 25)

{

if(testData[i][j] == 1)

{

((Rectangle)((Grid)testsArea.Children[i]).Children[j]).Fill = new SolidColorBrush(Colors.Black);

}

}

else

{

if(j == 25)

{

if(propTestData[i] > 0.9)

{

((Label)((Grid)testsArea.Children[i]).Children[j]).Content = "9";

}

else

{

((Label)((Grid)testsArea.Children[i]).Children[j]).Content = "0";

}

}

else

{

if (hoffTestData[i] > 0.6)

{

((Label)((Grid)testsArea.Children[i]).Children[j]).Content = "9";

}

else

{

((Label)((Grid)testsArea.Children[i]).Children[j]).Content = "0";

}

}

}

}

}

}

private int[][] ReadFile(string path)

{

List<int[]> res = new List<int[]>();

using (StreamReader sr = new StreamReader(path, System.Text.Encoding.Default))

{

string line;

while ((line = sr.ReadLine()) != null)

{

res.Add(line.Split(',').Select(n => Convert.ToInt32(n)).ToArray());

}

}

return res.ToArray();

}

private void DisplayGraphic(double[] data, Canvas canvas)

{

int i = 0;

canvas.Children.Clear();

foreach (var item in data)

{

Rectangle rectangle = new Rectangle();

rectangle.Fill = new SolidColorBrush(Colors.LightGray);

rectangle.Height = Math.Abs(item) \* 80;

rectangle.Width = 3;

rectangle.Margin = new Thickness(rectangle.Width \* i, canvas.Height - rectangle.Height, 0, 0);

canvas.Children.Add(rectangle);

i++;

}

}

private void Result\_Button(object sender, RoutedEventArgs e)

{

int[] startData = new int[25];

for(int i = 0; i < userInput.Children.Count; i++)

{

if(((Button)userInput.Children[i]).Background == Brushes.Black)

{

startData[i] = 1;

}

else

{

startData[i] = 0;

}

}

if (hoff.GetResult(startData) > 0.6)

{

hoffResult.Text = "9";

}

else

{

hoffResult.Text = "0";

}

if (backProp.GetResult(startData) > 0.9)

{

backpropResult.Text = "9";

}

else

{

backpropResult.Text = "0";

}

//hoffResult.Text = hoff.GetResult(startData).ToString();

//backpropResult.Text = backProp.GetResult(startData).ToString();

}

private void ChangeCollor\_Click(object sender, RoutedEventArgs e)

{

if(((Button)sender).Background == Brushes.Black)

{

((Button)sender).Background = Brushes.LightGray;

}

else

{

((Button)sender).Background = Brushes.Black;

}

}

private void Relearch\_Click(object sender, RoutedEventArgs e)

{

backProp = new NN\_Hoff(trainData, testData, result);

DisplayGraphic(backProp.Learn(0.01), graphicCanvas);

hoff = new NN\_Hoff(trainData, testData, result);

DisplayGraphic(hoff.LearnHoff(), graphicCanvas1);

DisplayTests();

}

}

}