**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**

Выполнил: студент гр. ИТИ-21

Шеметков К.Д.

Принял: преподаватель

Карабчикова Е.А.

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 14/2**

Вычисление значения функции «XOR». сеть Хебба (правила Хебба).

**Ход работы**

В ходе выполнения лабораторной работы была создана обучающая выборка и тестовая выборка, которые будут использоваться для обучения и проверки нейронной сети. Обучающая выборка показана на рисунке 1.



Рисунок 1 – Обучающая выборка

Был разработан графический интерфейс, позволяющий загружать обучающую выборку и преобразовывать его в массив входных сигналов, задавать топологию и на ее основе создавать нейронные системы с разными методами обучения, производить обучение нейронной системы с отслеживанием процесса обучения с помощью графика и тестировать обученную нейронную сеть на правильность результатов обучения.

Для нейросети было взято 25 нейронов для работы с изображениями размера 5х5 пикселей и 2 скрытых слоя.

Интерфейс приложения представлен на рисунке 4. График нейросети показан на рисунке 3. Результаты тестирования нейронной сети изображениях показаны на рисунках 4 и 5.

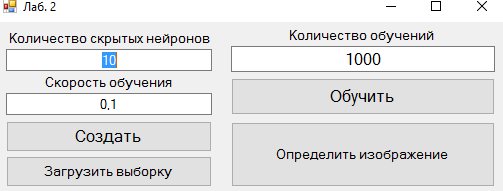


Рисунок 4 – Интерфейс приложения

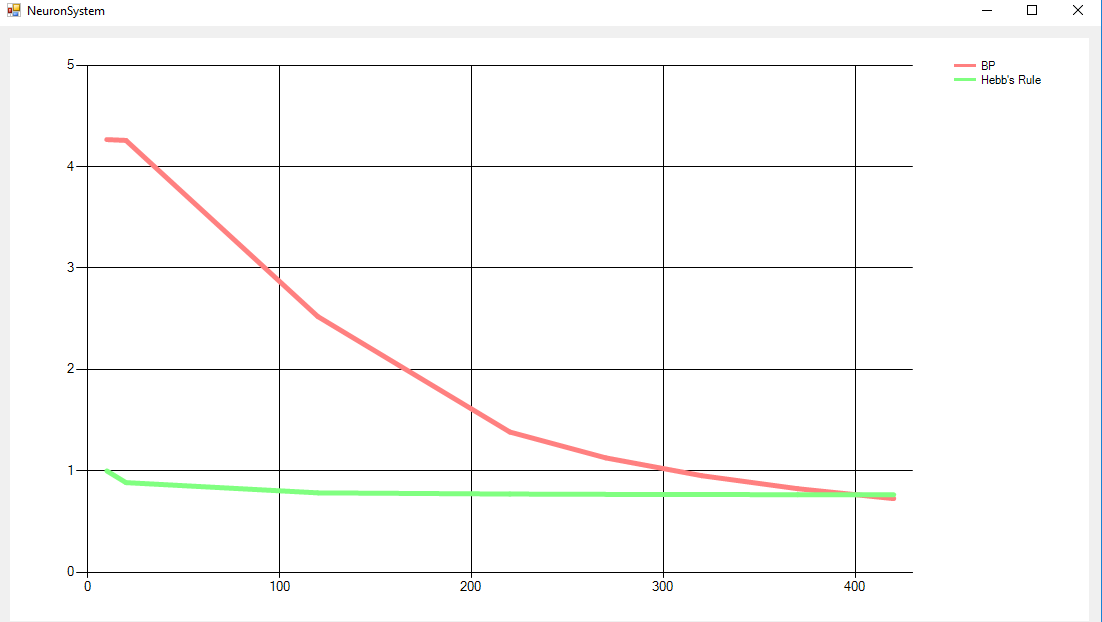


Рисунок 5 – График обучения нейронной сети

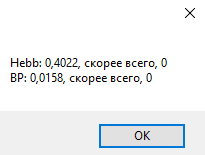


Рисунок 6 – Результат тестовой выборки №1

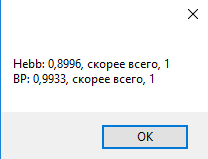


Рисунок 7 – Результат тестовой выборки №2

**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.

**Листинг HebbRule.cs:**

namespace NeuroLib

{

public class HebbRule : Learning

{

public HebbRule(Network network)

: base(network)

{

}

/// <summary>

/// Коррекция весов нейрона

/// </summary>

/// <param name="neuron">Нейрон</param>

/// <param name="error">Ошибка</param>

private void UpdateWeights(Neuron neuron, double error)

{

if (neuron.NeuronType == TypeOfNeuron.Input) return;

if (error != 0)

neuron.Delta = error \* neuron.ActivationFunc(neuron.Output);

else

neuron.Delta = 0;

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

{

var weight = neuron.Weights[i];

var input = neuron.Inputs[i];

var newWeigth = weight + input \* neuron.Delta;

neuron.Weights[i] = newWeigth;

}

}

/// <summary>

/// Обучение по правилу Хебба

/// </summary>

/// <param name="expectedOutput">Ожидаемое значение</param>

/// <param name="inputSignals">Входные сигналы</param>

/// <returns></returns>

/// <summary>

public override double Learn(double expectedOutput, params double[] inputSignals)

{

double actualOutput = Network.FeedForwardSignals(inputSignals).Output;

double difference = actualOutput - expectedOutput;

for (int j = Network.Layers.Count - 1; j >= 0; j--)

{

Layer currentLayer = Network.Layers[j];

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

{

Neuron neuron = currentLayer.Neurons[i];

double error = expectedOutput - neuron.Output;

UpdateWeights(neuron, error);

}

}

return difference \* difference / 2;

}

}

}

**Листинг ActFuncSigm.cs:**

using System;

namespace NeuroLib

{

public class ActFuncSigm : Neuron

{

public ActFuncSigm() : base() { }

/// <summary>

/// Функция активации логистическая

/// </summary>

/// <param name="x">Сумма</param>

/// <returns>Значение после активации</returns>

public override double ActivationFunction(double x)

{

return 1.0 / (1.0 + Math.Exp(-x));

}

/// <summary>

/// Производная функция активации сигмоида

/// </summary>

/// <param name="x">Сумма</param>

/// <returns>Значение после активации</returns>

public override double ActivationFunc(double x)

{

double sigmoid = ActivationFunction(x);

return sigmoid / (1 - sigmoid);

}

}

}

**Листинг BP.cs:**

using System.Linq;

namespace NeuroLib

{

public class BP : Learning

{

public BP(Network network)

: base(network)

{

}

/// <summary>

/// Метод обратного распростарнения ошибки

/// </summary>

/// <param name="expectedOutput">Ожидаемое значение</param>

/// <param name="inputSignals">Входные сигналы</param>

/// <returns>Квадрат ошибки</returns>

public override double Learn(double expectedOutput, params double[] inputSignals)

{

double actualOutput = Network.FeedForwardSignals(inputSignals).Output;

double difference = expectedOutput - actualOutput;

foreach (Neuron neuron in Network.Layers.Last().Neurons)

UpdateWeights(neuron, difference);

for (int j = Network.Layers.Count - 2; j >= 0; j--)

{

Layer currentLayer = Network.Layers[j];

Layer previousLayer = Network.Layers[j + 1];

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

{

Neuron neuron = currentLayer.Neurons[i];

for (int k = 0; k < previousLayer.Count; k++)

{

Neuron previousNeuron = previousLayer.Neurons[k];

double error = previousNeuron.Weights[i] \* previousNeuron.Delta \* Network.Topology.LearningRate;

UpdateWeights(neuron, error);

}

}

}

return difference \* difference;

}

/// <summary>

/// Изменение весов нейрона.

/// </summary>

/// <param name="neuron">Нейрон, веса которого будут изменяться.</param>

/// <param name="error">Значение оЗначениешибки.</param>

private void UpdateWeights(Neuron neuron, double error)

{

if (neuron.NeuronType == TypeOfNeuron.Input) return;

neuron.Delta = error \* neuron.ActivationFunc(neuron.Output);

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

{

var weight = neuron.Weights[i];

var input = neuron.Inputs[i];

var newWeigth = weight + input \* neuron.Delta \* Network.Topology.LearningRate;

neuron.Weights[i] = newWeigth;

}

}

}

}

**Листинг Layer.cs:**

using System;

using System.Collections.Generic;

namespace NeuroLib

{

/// <summary>

/// Класс слоя нейроной сети

/// </summary>

public class Layer

{

/// <summary>

/// Нейроны на данном слое

/// </summary>

public List<Neuron> Neurons { get; private set; }

/// <summary>

/// Количество нейроннов на данном слое

/// </summary>

public int Count => Neurons?.Count ?? 0;

/// <summary>

/// Конструктор

/// </summary>

/// <param name="neurons">Нейроны на данном слое</param>

/// <param name="type">Тип нейронов на данном слое</param>

public Layer(List<Neuron> neurons, TypeOfNeuron type = TypeOfNeuron.Intermediate)

{

if (neurons == null)

throw new Exception("Нет нейронов.");

foreach (Neuron neuron in neurons)

if (neuron.NeuronType != type)

throw new Exception("Ошибка типа нейрона.");

Neurons = neurons;

}

/// <summary>

/// Возращает выходные сигналы всех нейронов на данном слое

/// </summary>

/// <returns>Список выходных сигналов слоя</returns>

public List<double> GetSignals()

{

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

foreach (Neuron neuron in Neurons)

results.Add(neuron.Output);

return results;

}

}

}

**Листинг Neuron.cs:**

using System;

using System.Collections.Generic;

namespace NeuroLib

{

/// <summary>

/// Класс нейрона

/// </summary>

public abstract class Neuron

{

/// <summary>

/// Веса приходящих синапсов

/// </summary>

public List<double> Weights { get; protected set; }

/// <summary>

/// Входные сигналы нейронов

/// </summary>

public List<double> Inputs { get; protected set; }

/// <summary>

/// Тип нейрона

/// </summary>

public TypeOfNeuron NeuronType { get; protected set; }

/// <summary>

/// Значение выходного синапса нейрона

/// </summary>

public double Output { get; protected set; }

/// <summary>

/// Значение дельты

/// </summary>

public double Delta { get; set; }

/// <summary>

/// Конструктор

/// </summary>

public Neuron() { }

/// <summary>

/// Инициализация нейрона

/// </summary>

/// <param name="inputCount">Число входных сигналов</param>

/// <param name="type">Тип нейрона</param>

public void InitializeNeuron(int inputCount, TypeOfNeuron type = TypeOfNeuron.Intermediate)

{

NeuronType = type;

Weights = new List<double>();

Inputs = new List<double>();

InitializeRandomWeights(inputCount);

}

/// <summary>

/// Инициализация весов рандомными значениями

/// </summary>

/// <param name="inputCount">Количество входных сигналов</param>

private void InitializeRandomWeights(int inputCount)

{

Random random = new Random();

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

{

if (NeuronType == TypeOfNeuron.Input)

{

Weights.Add(1);

}

else

{

Weights.Add(random.NextDouble());

}

Inputs.Add(0);

}

}

/// <summary>

/// Подаются сигналы и активируется функция

/// </summary>

/// <param name="inputSignals">Входные сигналы</param>

/// <returns>Выходное значение нейрона нейрона</returns>

public double FeedForwardSignals(List<double> inputSignals)

{

if (Weights.Count != inputSignals.Count)

throw new Exception("Неверное количество входящих сигналов");

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

{

Inputs[i] = inputSignals[i];

}

double sum = 0;

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

{

sum += inputSignals[i] \* Weights[i];

}

if (NeuronType != TypeOfNeuron.Input)

Output = ActivationFunction(sum);

else

Output = sum;

return Output;

}

/// <summary>

/// функция активации

/// </summary>

/// <param name="x"> сумма </param>

/// <returns></returns>

public abstract double ActivationFunction(double x);

public abstract double ActivationFunc(double x);

/// <summary>

/// Переопределение метода ToString которые повзращает строку с выходным значением синапса нейрона

/// </summary>

/// <returns>Выходное значением синапса нейрона</returns>

public override string ToString()

{

return Output.ToString();

}

}

public enum TypeOfNeuron

{

Input = 0,

Intermediate,

Output

}

}

**Листинг Topology.cs:**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace NeuroLib

{

/// <summary>

/// Класс топологии сети

/// </summary>

public class Topology

{

/// <summary>

/// Количество входных слоев

/// </summary>

public int InputCount { get; private set; }

/// <summary>

/// Количество выходных слоев

/// </summary>

public int OutputCount { get; private set; }

/// <summary>

/// Коэфициент скорости обучения

/// </summary>

public double LearningRate { get; private set; }

/// <summary>

/// Скрытые слои

/// </summary>

public List<int> HiddenLayers { get; private set; }

/// <summary>

/// Конструктор

/// </summary>

/// <param name="inputCount">Количество входных слоев</param>

/// <param name="outputCount">Количество выходных слоев</param>

/// <param name="learningRate">Коэфициент обучения</param>

/// <param name="hiddenLayers">Скрытые слои</param>

public Topology(int inputCount, int outputCount, double learningRate, params int[] hiddenLayers)

{

InputCount = inputCount;

OutputCount = outputCount;

LearningRate = learningRate;

HiddenLayers = new List<int>();

HiddenLayers.AddRange(hiddenLayers);

}

}

}

**Листинг Data.cs:**

namespace NeuroLib

{

public class Data

{

public double Expected { get; private set; }

public double[] InputsSignals { get; private set; }

public Data(double expected, double[] inputsSignals)

{

Expected = expected;

InputsSignals = inputsSignals;

}

}

}