Министерство науки и высшего образования Российской Федерации

НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ

ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ (НИ ТГУ)

Институт прикладной информатики и компьютерных наук

**Кафедра прикладной информатики**

**Лабораторная работа №4**

по дисциплине «Интеллектуальные системы»

на тему «Нейронные сети»

Выполнил:

студент группы №932001

Данила Лапатин.

Проверил:

\_\_\_\_\_\_\_\_\_\_кафедры теоретических основ информатики

А. Д. Брагин

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ *зачтено /не зачтено*

Томск, 2023г.

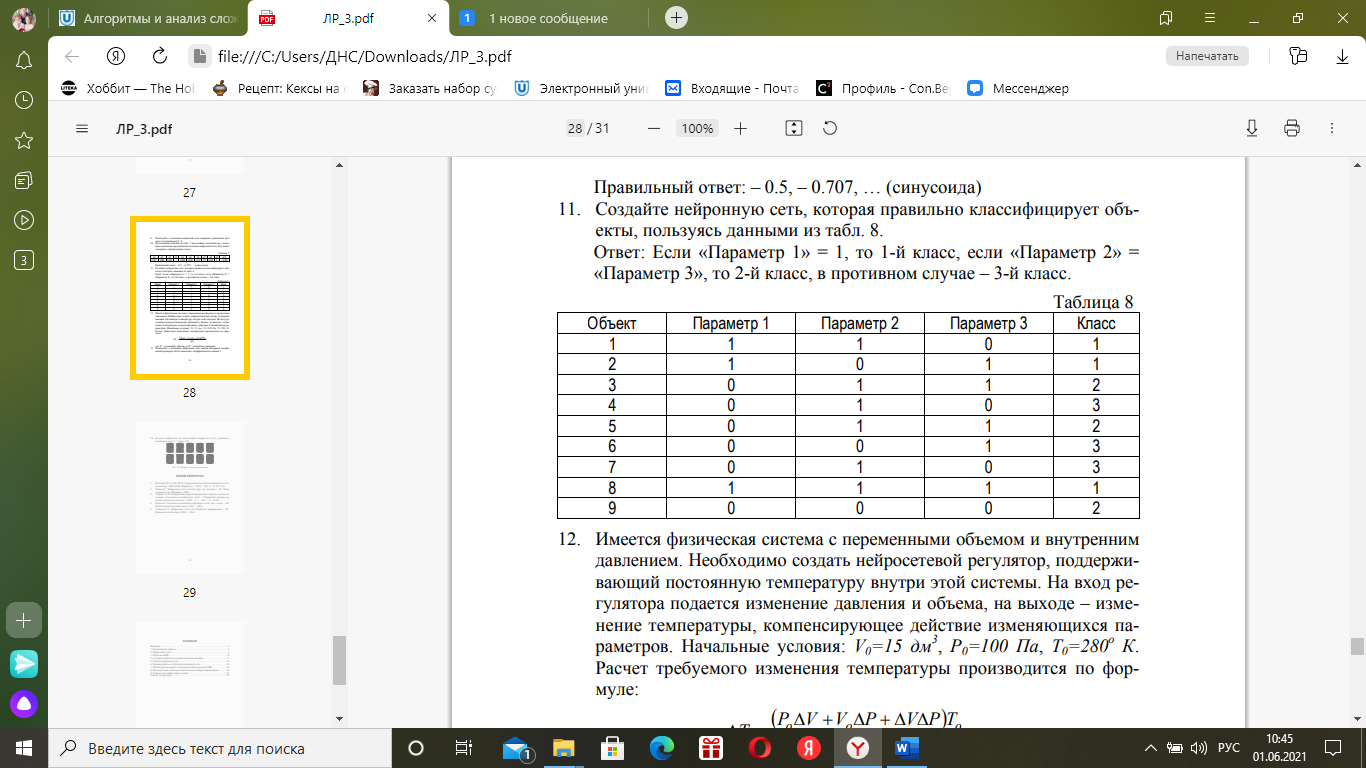
1. **Цель работы.**

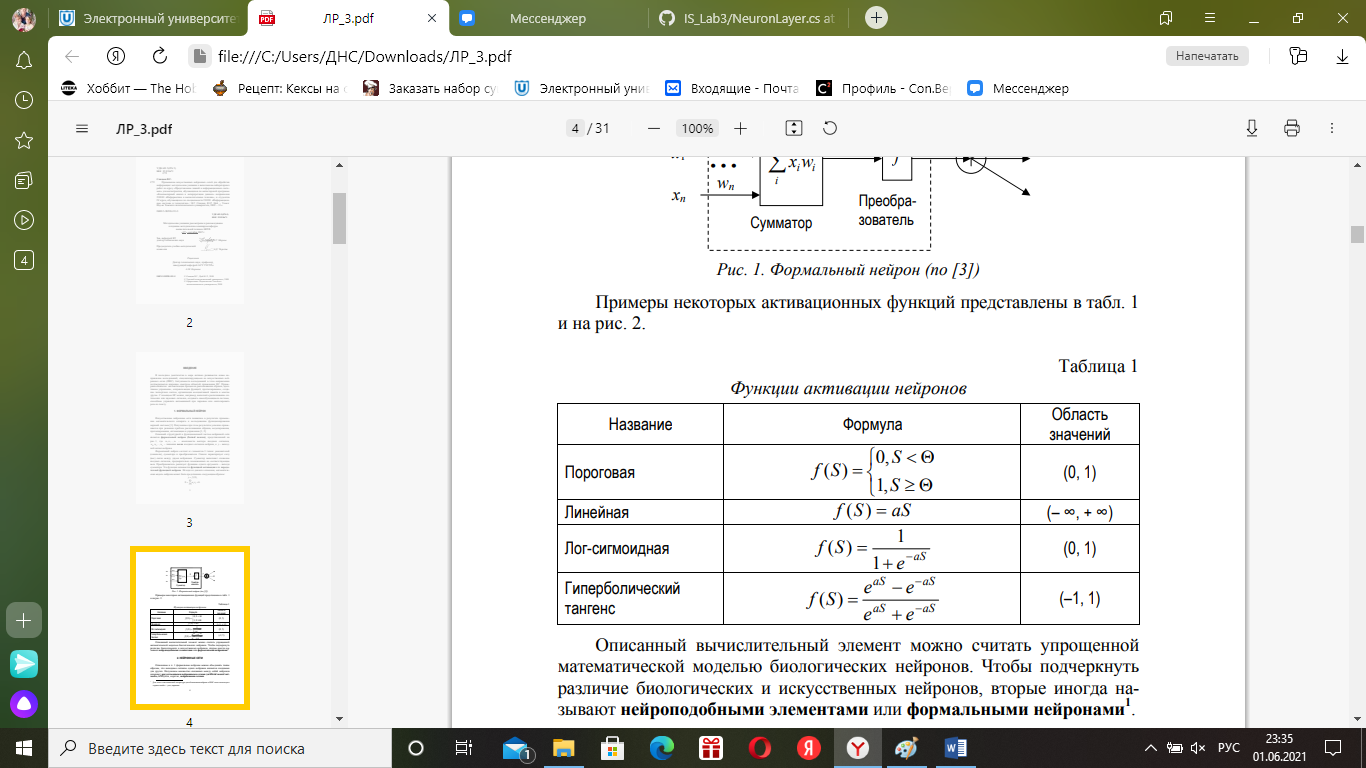
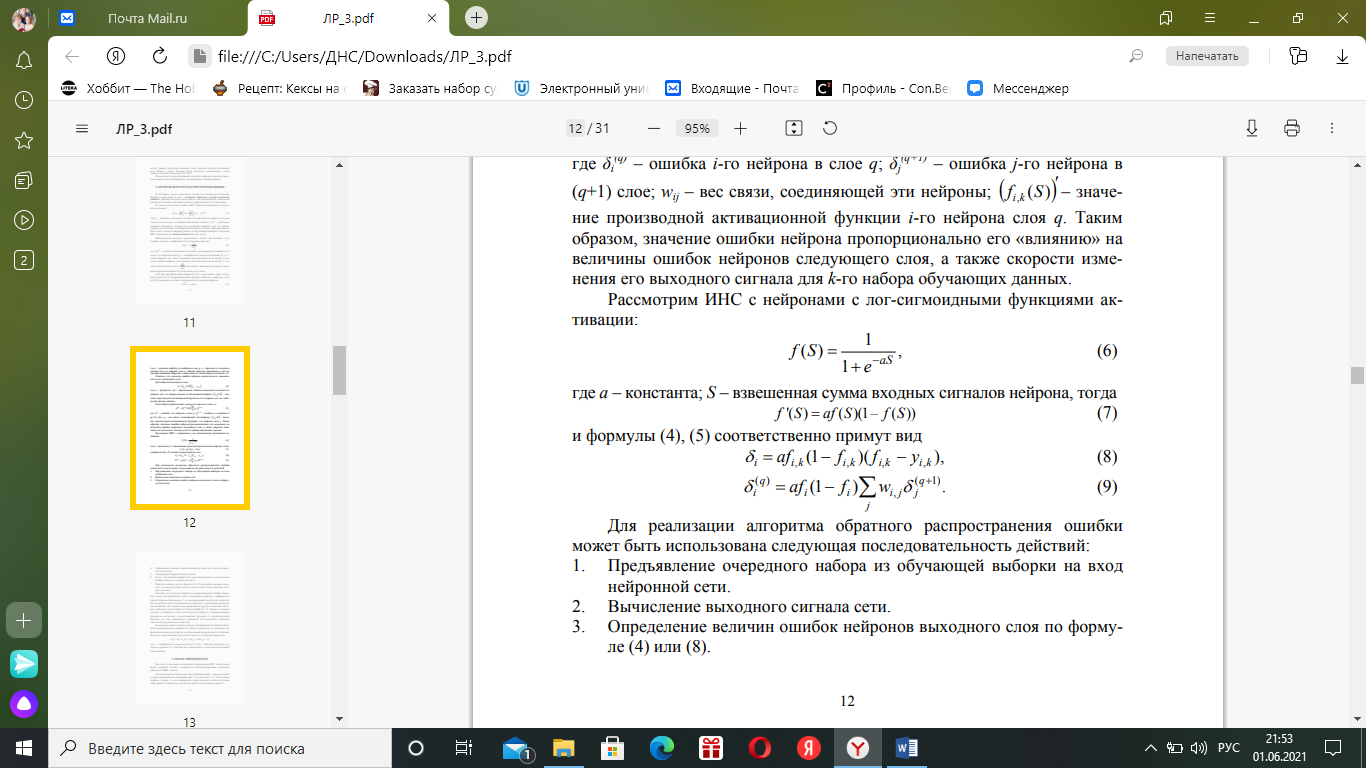
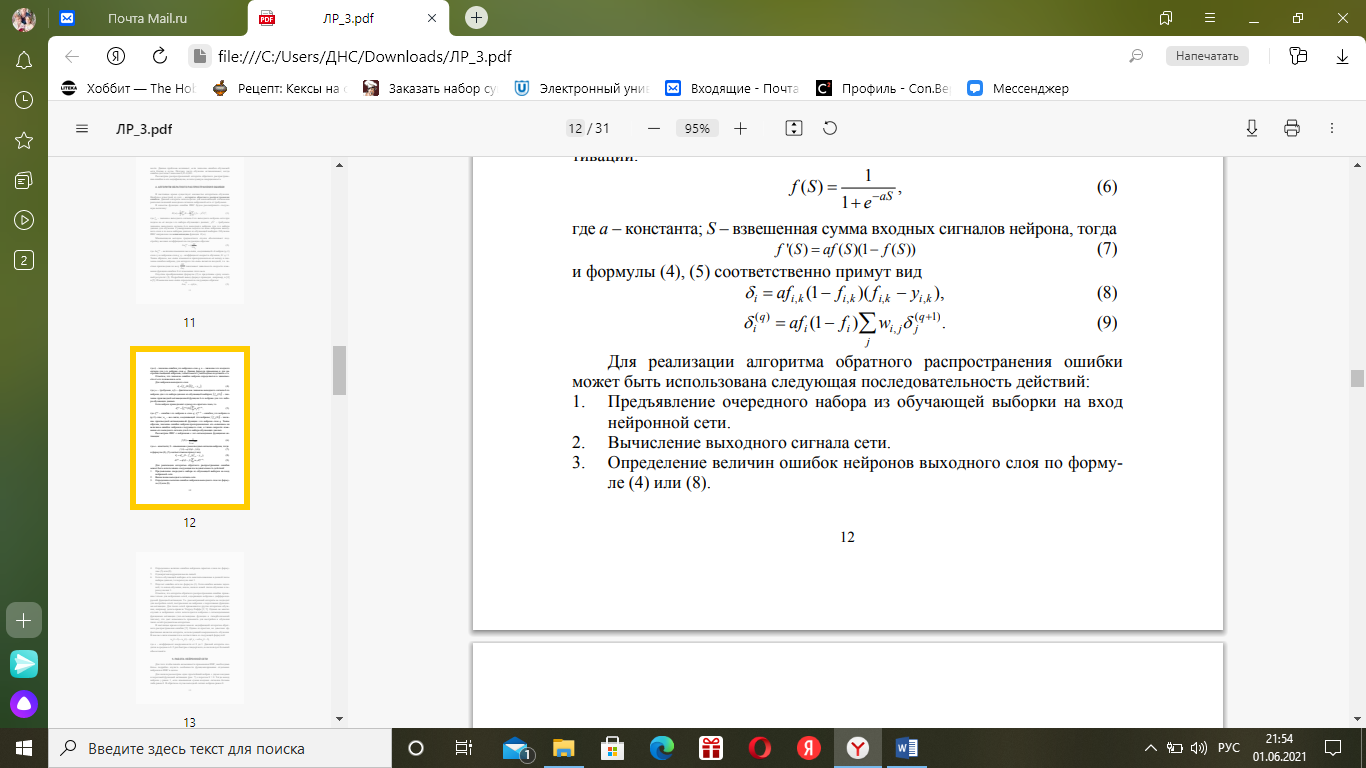
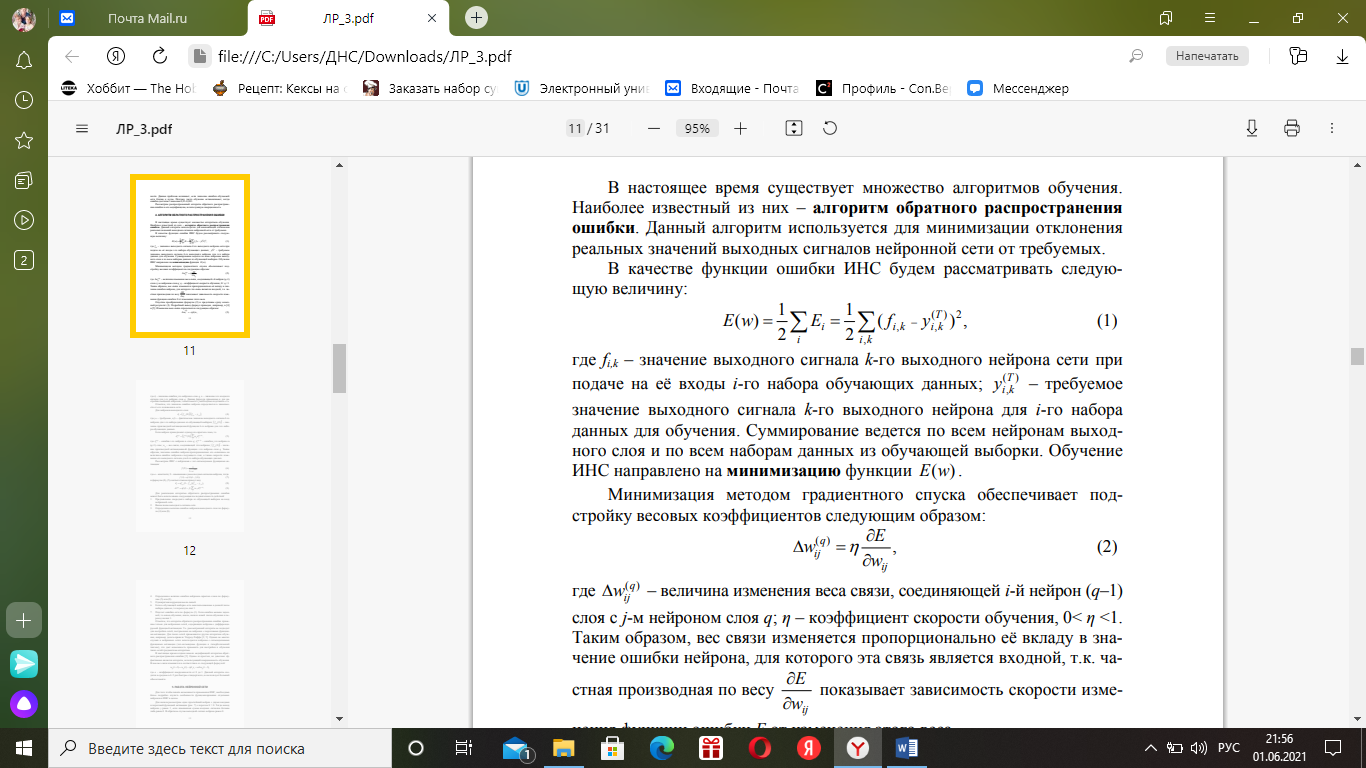
Целью лабораторной работы является создание программы, реализующей ИНС для решения поставленной задачи, в нашем случае это нейронная сеть, которая правильно классифицирует объекты, используя данные таблицы.

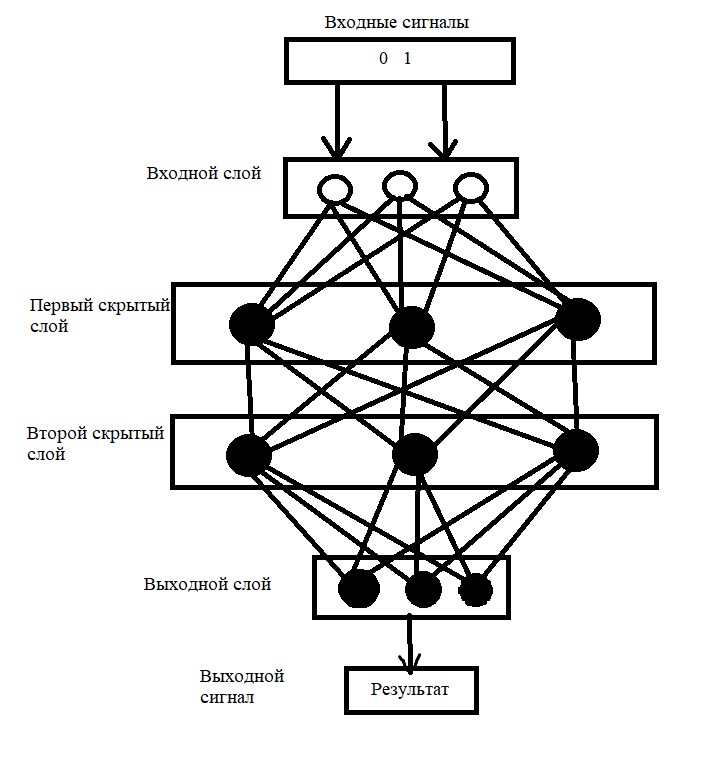
1. **Постановка задачи.**

Необходимо создать нейронную сеть, которая правильно классифицирует объекты, пользуясь данными из таблицы.

Ответ: если «Параметр 1» = 1, то 1-й класс, если «Параметр 2» = «Параметр 3», то 2-й класс, в противном случае – 3-й класс.



1. **Метод решения задачи.**
2. Была реализована **лог-сигмоидная** функция активации и её производная
3. Коэффицент скорости обучения ƞ = 0.01
4. 4 слоя, 2 скрытых, 1 входной, 1 выходной, в каждом слое по 3 нейрона
5. В работе была реализована многослойная ИНС прямого распространения. В многослойных сетях нейроны объединяются в слои таким образом, что нейроны одного слоя имеют одинаковые входные сигналы. Число нейронов в слое может быть произвольным. Входные сигналы подаются на нейроны входного слоя, а выходами сети являются выходные сигналы нейронов последнего слоя. Кроме входного и выходного слоев в многослойной нейронной сети были использованы скрытые слои.
6. В программе было реализовано **обучение с учителем** (**supervised learning**) подразумевает использование заранее сформированного множества обучающих примеров. Каждый пример содержит вектор входных сигналов и соответствующий вектор эталонных выходных сигналов, которые зависят от поставленной задачи. Обучение нейронной сети направлено на такое изменение весов связей ИНС, при котором значение выходных сигналов ИНС как можно меньше отличаются от требуемых значений выходных сигналов для данного вектора входных сигналов.
7. В процессе обучения на вход нейронной сети предъявляются данные (в данной программе это 0 и отличное от нуля значение) из обучающей выборки и, в соответствии со значениями выходных сигналов, определяющих ошибку функционирования сети, производится коррекция весов связей. В результате обучения получаем нейронную сеть, которая без перенастройки весов связей формирует с требуемой погрешностью выходные сигналы Y при подаче на вход сети любого набора входных сигналов из обучающего множества. (принадлежность к классу в target)
8. Был реализован алгоритм обратного распространения ошибки, реализован следующим образом:
9. Предъявление очередного набора из обучающей выборки на вход нейронной сети.
10. Вычисление выходного сигнала сети.
11. Определение величин ошибок нейронов выходного слоя по формуле:.
12. Определение величин ошибок нейронов скрытых слоев по формуле:
13. Однократная коррекция весов связей.
14. Если в обучающей выборке есть неиспользованные в данной эпохе наборы данных, то переход на шаг 1.
15. Подсчет ошибки сети по формуле:.  
     Если ошибка меньше заданной, то конец обучения, иначе, начало новой эпохи обучения и переход на шаг 1.
16. Структурная схема алгоритма



Листинг программы:

public class Neuron

{

//Выходное значение

public double Output;

//Базис

public double Base;

}

public class Layer

{

//Размер слоя

public int Size;

//Нейроны

public Neuron[] Neurons;

//Веса

public double[][] Weights;

public Layer() { }

//Инициализируем слой, задаем кол-во нейронов

public void Init(int size, int next)

{

Size = size;

Neurons = new Neuron[size];

Weights = new double[size][];

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

{

Neurons[i] = new Neuron();

Weights[i] = new double[next];

}

}

}

//Класс нейронной сети

public class NeuralNetwork

{

// Слои

public Layer[] Layers;

// Входной слой

public Layer Input => Layers.First();

// Выходной слой

public Layer Output => Layers.Last();

// Коэфф обучения

public double LearningRate = 0.01;

public double Activation(double x) { return 1 / (1 + Math.Exp(-x)); } // Активационная функция

public double Derivative(double x) { return x \* (1 - x); } // Производная активационной функции

public NeuralNetwork(int[] size)

{

// Инициализируем слои

Layers = new Layer[size.Length];

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

{

Layers[i] = new Layer(); //инициализация слоя

int next\_size = 0;

if (i + 1 < size.Length) next\_size = size[i + 1]; //закос о будущем слое

Layers[i].Init(size[i], next\_size); //инициализация слоёв

}

}

public void Work()

{ //newLrs-новый слой lstLrs-предыдущий слой curNeur-нейроны нового слоя

// Для каждого нейрона в каждом слое рассчитываем выходное значение

for (int i = 1; i < Layers.Length; i++)

{

var curLrs = Layers[i]; //новый слой

var prevLrs = Layers[i - 1]; //предыдущий слой

for (int k = 0; k < curLrs.Size; k++)

{

var curNeur = curLrs.Neurons[k]; //нейроны нынешнего слоя

for (int j = 0; j < prevLrs.Size; j++)

{

curNeur.Output += prevLrs.Weights[j][k] \* prevLrs.Neurons[j].Output; //(w\*x) получаем сумму

}

curNeur.Output += curNeur.Base; //+b

curNeur.Output = Activation(curNeur.Output); //преобразователь выхода сумматора

}

}

}

// Очистка выходных значений

public void Clear()

{

foreach (var l in Layers)

{

foreach (var n in l.Neurons)

{

n.Output = 0;

}

}

}

// Обучение на основе алгоритма обратного распространения ошибки

public void BackPropagation(double[] targets)

{

double[] errors = new double[Output.Size];

for (int i = 0; i < Output.Size; i++)

{

errors[i] = targets[i] - Output.Neurons[i].Output; //отклонение от реальных значений

}

for (int k = Layers.Length - 2; k >= 0; k--)

{

Layer prevLrs = Layers[k];

Layer curLrs = Layers[k + 1];

double[] errorsNext = new double[prevLrs.Size]; //для следующего слоя

double[] gradients = new double[curLrs.Size];

for (int i = 0; i < curLrs.Size; i++) //вычисление градиента

{

gradients[i] = errors[i] \* Derivative(curLrs.Neurons[i].Output);

gradients[i] \*= LearningRate;

}

double[][] deltas = new double[curLrs.Size][];

for (int i = 0; i < curLrs.Size; i++)

{

deltas[i] = new double[prevLrs.Size];

for (int j = 0; j < prevLrs.Size; j++)

{

deltas[i][j] = gradients[i] \* prevLrs.Neurons[j].Output; //значение ошибки нейрона

}

}

for (int i = 0; i < prevLrs.Size; i++)

{

errorsNext[i] = 0;

for (int j = 0; j < curLrs.Size; j++)

{

errorsNext[i] += prevLrs.Weights[i][j] \* errors[j];

}

}

errors = new double[prevLrs.Size];

errorsNext.CopyTo(errors, 0);

double[][] weightsNew = new double[prevLrs.Size][];

for (int i = 0; i < prevLrs.Size; i++)

weightsNew[i] = new double[curLrs.Size];

for (int i = 0; i < curLrs.Size; i++)

{

for (int j = 0; j < prevLrs.Size; j++)

{

weightsNew[j][i] = prevLrs.Weights[j][i] + deltas[i][j]; //коррекция весов связей ИНС

}

}

prevLrs.Weights = weightsNew; //Замена весов

for (int i = 0; i < curLrs.Size; i++)

{

curLrs.Neurons[i].Base += gradients[i]; //Настройка смещения нейронов

}

}

}

}

class Program

{

static NeuralNetwork MyNetwork;

static readonly Random rnd = new Random((int)DateTime.Now.Ticks);

static void Main()

{

//Задаем кол-во слоев и нейронов

var size = new int[] { 3, 3, 3, 3 }; //4 слоя с 3 нейронами

MyNetwork = new NeuralNetwork(size); //Создаем нейронную сеть

//Заполняем нейроны начальными данными

foreach (var l in MyNetwork.Layers) //в каждом нейроне каждого слоя заполняем base

{

foreach (var n in l.Neurons)

{

n.Base = rnd.NextDouble() \* 2 - 1; //base от -1 до 1

}

}

for (int i = 1; i < MyNetwork.Layers.Length; i++) //weight от -1 до 1

{

for (int j = 0; j < MyNetwork.Layers[i].Weights.Length; j++)

{

for (int k = 0; k < MyNetwork.Layers[i].Weights[j].Length; k++)

MyNetwork.Layers[i].Weights[j][k] = rnd.NextDouble() \* 2 - 1;

}

}

//Производим 100,000 итераций обучения

for (int i = 0; i < 100000; i++) Learning();

while (true) //Работаем с нейронной сетью

{

//Вводим три параметра

Console.WriteLine("Input 3 parametrs(0, 1): ");

for (int i = 0; i < MyNetwork.Input.Size; i++)

{

//Задаем значения входных

MyNetwork.Input.Neurons[i].Output = int.Parse(Console.ReadLine());

}

Console.WriteLine($"Results: {MyNetwork.Input.Neurons[0].Output} {MyNetwork.Input.Neurons[1].Output} {MyNetwork.Input.Neurons[2].Output}");

//Начинаем работу

MyNetwork.Work();

//Выводим результат из выходных нейронов

int max = 0;

for (int i = 1; i < MyNetwork.Output.Size; i++)

{

if (MyNetwork.Output.Neurons[max].Output < MyNetwork.Output.Neurons[i].Output) max = i;

}

//Console.WriteLine($"Class: {max + 1}= {nn.Output.Neurons[max].Output}\n");

Console.WriteLine($"Class: {max + 1}\n");

MyNetwork.Clear();

}

}

static void Learning() //обучение на входных данных

{ //задаем входные нейроны

MyNetwork.Input.Neurons[0].Output = 1; MyNetwork.Input.Neurons[1].Output =1; MyNetwork.Input.Neurons[2].Output = 0;

var target = new double[] { 1, 0, 0 }; //цель обучения

MyNetwork.Work(); //Начало обучения

MyNetwork.BackPropagation(target); //Расчет новых весов нейронов используя алгоритм обратного распространения ошибки

MyNetwork.Input.Neurons[0].Output = 1; MyNetwork.Input.Neurons[1].Output =0; MyNetwork.Input.Neurons[2].Output = 1;

target = new double[] { 1, 0, 0 };

MyNetwork.Work();

MyNetwork.BackPropagation(target);

MyNetwork.Input.Neurons[0].Output = 0; MyNetwork.Input.Neurons[1].Output =1; MyNetwork.Input.Neurons[2].Output = 1;

target = new double[] { 0, 1, 0 };

MyNetwork.Work();

MyNetwork.BackPropagation(target);

MyNetwork.Input.Neurons[0].Output = 0; MyNetwork.Input.Neurons[1].Output =1; MyNetwork.Input.Neurons[2].Output = 0;

target = new double[] { 0, 0, 1 };

MyNetwork.Work();

MyNetwork.BackPropagation(target);

MyNetwork.Input.Neurons[0].Output = 0; MyNetwork.Input.Neurons[1].Output =0; MyNetwork.Input.Neurons[2].Output = 1;

target = new double[] { 0, 0, 1 };

MyNetwork.Work();

MyNetwork.BackPropagation(target);

MyNetwork.Input.Neurons[0].Output = 0; MyNetwork.Input.Neurons[1].Output =1; MyNetwork.Input.Neurons[2].Output = 0;

target = new double[] { 0, 0, 1 };

MyNetwork.Work();

MyNetwork.BackPropagation(target);

MyNetwork.Input.Neurons[0].Output = 1; MyNetwork.Input.Neurons[1].Output =1; MyNetwork.Input.Neurons[2].Output = 1;

target = new double[] { 1, 0, 0 };

MyNetwork.Work();

MyNetwork.BackPropagation(target);

MyNetwork.Input.Neurons[0].Output = 0; MyNetwork.Input.Neurons[1].Output =0; MyNetwork.Input.Neurons[2].Output = 0;

target = new double[] { 0, 1, 0 };

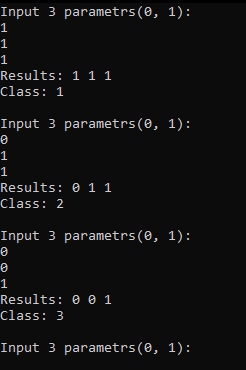
MyNetwork.Work();

MyNetwork.BackPropagation(target);

}

}

1. Вывод программы:

  
ссылка на GitHub:

https://github.com/DanilaLapatin/IntSistLab.git

1. Выводы:

* Цель работы является достигнутой.
* Использование ИНС обладает следующими преимуществами:
* Возможность решения трудно формализуемых задач, для которых трудно найти точный алгоритм решения (распознавание речи, рукописного текста).
* Массовый параллелизм в обработке информации. Данное преимущество позволяет реализовать нейросетевые алгоритмы и методы на параллельных вычислительных структурах, что особенно актуально в настоящее время в связи с распространением распределенных вычислений и массовым внедрением многоядерных центральных и графических процессоров для ПК, а также в связи с унификацией разнородных вычислений (научных, физических, графических и др.) на персональных компьютерах.
* ИНС представляют единую концепцию для решения разнообразных задач, таких как задачи классификации, аппроксимации, моделирования, распознавания образов, принятия решений, обработки информации, кластеризации и др.
* Возможность нестандартного решения известных задач, что расширяет и обогащает арсенал существующих средств и подходов, поскольку позволяет посмотреть на проблему и ее решение под «нестандартным» углом.