1. 1. студента группы ИТ – 42  
      Курбатовой Софьи Андреевны

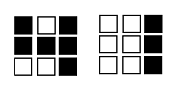
|  |  |  |  |
| --- | --- | --- | --- |
| Выполнение: |  | Защита |  |

* + 1. Работа с простейшими нейронными сетями

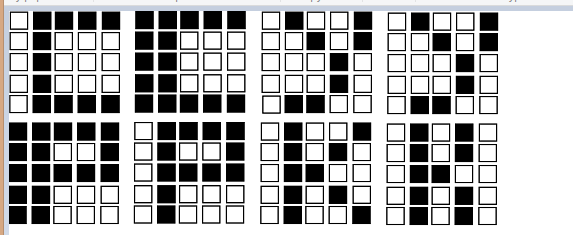
**Цель работы**: приобретение и закрепление знаний и получение практических навыков работы с простейшими нейронными сетями.

* + - 1. Содержание работы
      2. Вариант 8
      3. В соответствии с заданным вариантом:
* разработать черно-белые изображения заданных цифр;
* выбрать число входов нейрона;
* обучить нейрон с помощью алгоритма на основе правила Хебба распознаванию двух цифр;
  + - 1. В соответствии с заданным вариантом:
* разработать черно- белые изображения заданных букв;
* выбрать архитектуру относительной нейронной сети;
* обучить нейросеть с помощью алгоритма на основе правила Хебба распознаванию заданных букв.

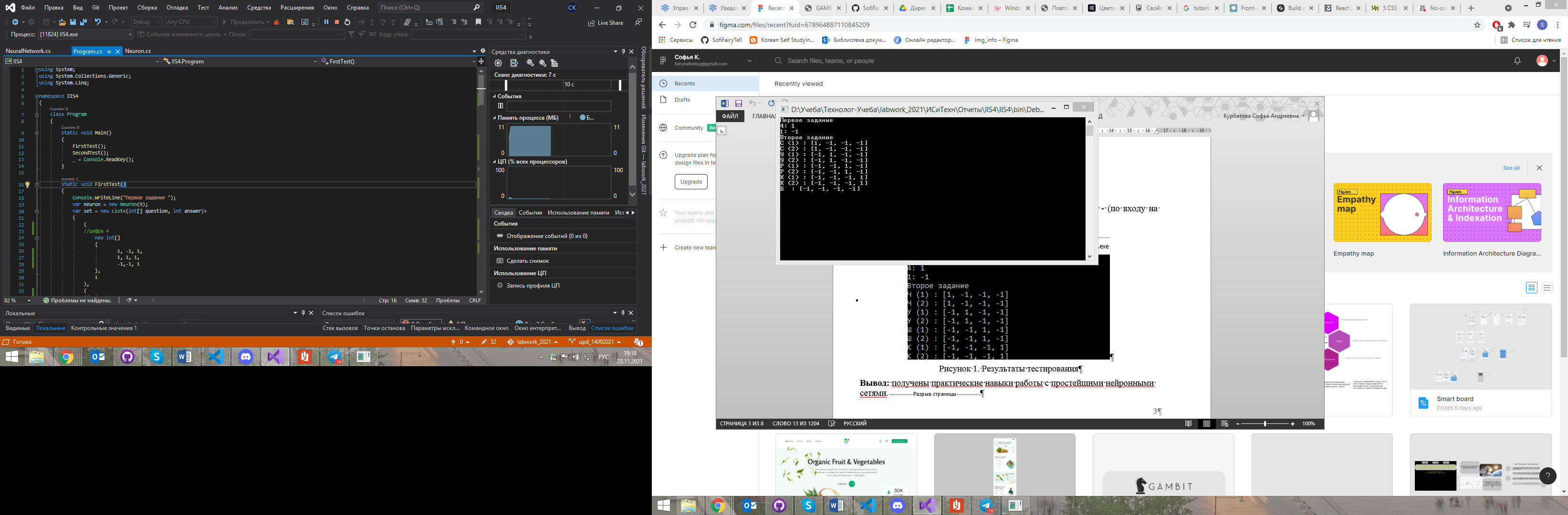
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Номер варианта | Цифры для обучения нейрона распознаванию изображений | Число изображений каждой цифры | Число *i* первых букв фамилии студента, используемых для распознавания | Число изображений каждой буквы |
| 8 | 0,8 | 1 | 4 | 2 |

* + - * 1. Ход работы
        2. 1. Созданы изображения для чисел 1 и 4. Число входов нейронов 9, по входу на пиксель. Он выдает 1, если распознает на изображении 4 и -1 если распознает 1.
      1. 

чб изображения для чисел

* + - 1. 2. Были разработаны изображения для букв. Всего нейронов будет 4 по количеству букв, а входов 25 при учете по пикселю на вход.
      2. 

чб изображения для букв

* + - 1. 3. Получены следующие результаты тестирования.
      2. 

Результаты тестирования

* + - 1. **Вывод**: Таким образом в ходе выполнения лабораторной работы были получены практические навыки работы с простейшими нейронными сетями.

Листинг 1.

using System;

using System.Collections.Generic;

using System.Linq;

namespace IIS4

{

class Program

{

static void Main()

{

FirstTest();

SecondTest();

\_ = Console.ReadKey();

}

static void FirstTest()

{

Console.WriteLine("Первое задание ");

var neuron = new Neuron(9);

var set = new List<(int[] question, int answer)>

{

(

//цифра 4

new int[]

{

1, -1, 1,

1, 1, 1,

-1,-1, 1

},

1

),

(

//цифра 1

new int[]

{

-1, -1, 1,

-1, -1, 1,

-1, -1, 1,

},

-1

)

};

neuron.Training(set);

var res = neuron.AskQuestion(new int[]

{

1, -1, 1,

1, 1, 1,

-1, -1, 1

});

Console.WriteLine($"4: { res }");

res = neuron.AskQuestion(new int[]

{

-1, -1, 1,

-1, -1, 1,

-1, -1, 1

});

Console.WriteLine($"1: { res }");

}

static void SecondTest()

{

Console.WriteLine("Второе задание ");

var neuralNetwork = new NeuralNetwork(25, 4);

var set = new List<(int[] question, int[] answer)>

{

(

//буква С

new int[]

{

-1, 1, 1, 1, 1,

-1, 1, -1, -1, -1,

-1, 1, -1, -1, -1,

-1, 1, -1, -1, -1,

-1, 1, 1, 1, 1

},

new int[]{ 1, -1, -1, -1}

),

(

//буква С другой вариант

new int[]

{

1, 1, 1, 1, 1,

1, 1, -1, -1, -1,

1, 1, -1, -1, -1,

1, 1, -1, -1, -1,

1, 1, 1, 1, 1

},

new int[]{ 1, -1, -1, -1}

),

(//буква у

new int[]

{

-1, 1, -1, -1, 1,

-1, -1, 1, -1, 1,

-1, -1, -1, 1, -1,

-1, -1, -1, 1, -1,

-1, 1, 1, -1, -1,

},

new int[]{ -1, 1, -1, -1}

),

(//буква у другой вариант

new int[]

{

-1, 1, -1, -1, 1,

-1, -1, 1, -1, 1,

-1, -1, -1, 1, -1,

-1, -1, -1, 1, -1,

-1, 1, 1, -1, -1,

},

new int[]{ -1, 1, -1, -1}

),

(//буква Р первый вариант

new int[]

{

-1, 1, 1, 1, 1,

-1, 1, -1, -1, 1,

-1, 1, 1, 1, 1,

-1, 1, -1, -1, -1,

-1, 1, -1, -1, -1,

},

new int[]{ -1, -1, 1, -1}

),

(//буква Р второй вариант

new int[]

{

1, 1, 1, 1, 1,

1, 1, -1, -1, 1,

1, 1, 1, 1, 1,

1, 1, -1, -1, -1,

1, 1, -1, -1, -1,

},

new int[]{ -1, -1, 1, -1}

),

(

new int[]

{

-1, 1, -1, -1, 1,

-1, 1, -1, 1, -1,

-1, 1, 1, -1, -1,

-1, 1, -1, 1, -1,

-1, 1, -1, -1, 1,

},

new int[]{ -1, -1, -1, 1}

),

(

new int[]

{

-1, 1, -1, -1, 1,

-1, 1, -1, 1, -1,

-1, 1, 1, 1, -1,

-1, 1, -1, 1, -1,

-1, 1, -1, -1, 1,

},

new int[]{ -1, -1, -1, 1}

),

(

new int[]

{

-1, -1, -1, -1, -1,

-1, -1, -1, -1, -1,

-1, -1, -1, -1, -1,

-1, -1, -1, -1, -1,

-1, -1, -1, -1, -1,

},

new int[]{ -1, -1, -1, -1}

),

};

neuralNetwork.Training(set);

var values = set.Select(x => x.question).ToList();

var res = neuralNetwork.AskQuestion(values[0]);

Console.WriteLine($"С (1) : [{ string.Join(", ", res) }]");

res = neuralNetwork.AskQuestion(values[1]);

Console.WriteLine($"С (2) : [{ string.Join(", ", res) }]");

res = neuralNetwork.AskQuestion(values[2]);

Console.WriteLine($"У (1) : [{ string.Join(", ", res) }]");

res = neuralNetwork.AskQuestion(values[3]);

Console.WriteLine($"У (2) : [{ string.Join(", ", res) }]");

res = neuralNetwork.AskQuestion(values[4]);

Console.WriteLine($"Р (1) : [{ string.Join(", ", res) }]");

res = neuralNetwork.AskQuestion(values[5]);

Console.WriteLine($"Р (2) : [{ string.Join(", ", res) }]");

res = neuralNetwork.AskQuestion(values[6]);

Console.WriteLine($"К (1) : [{ string.Join(", ", res) }]");

res = neuralNetwork.AskQuestion(values[7]);

Console.WriteLine($"К (2) : [{ string.Join(", ", res) }]");

res = neuralNetwork.AskQuestion(values[8]);

Console.WriteLine($"Б : [{ string.Join(", ", res) }]");

}

}

}

namespace IIS4

{

public class Neuron

{

private double[] EntranceWeights;

private int Exit;

public Neuron(int entranceCount)

{

EntranceWeights = new double[entranceCount];

}

public void Training(List<(int[] question, int answer)> tuples)

{

var countGoodAnswer = 0;

while (countGoodAnswer < tuples.Count())

{

countGoodAnswer = 0;

for (var i = 0; i < tuples.Count(); i++)

{

var tuple = tuples[i];

Recount(tuple.question);

if (Exit != tuple.answer)

{

RecountEntrancesWeights(tuple);

}

else

{

countGoodAnswer++;

}

}

}

}

public int AskQuestion(int[] question)

{

Recount(question);

return Exit;

}

public void RecountEntrancesWeights((int[] question, int answer) tuple)

{

for (var i = 0; i < tuple.question.Length; i++)

{

EntranceWeights[i] = EntranceWeights[i] + tuple.question[i] \* tuple.answer;

}

}

private void Recount(int[] question)

{

var sum = Enumerable.Range(0, EntranceWeights.Length)

.Select(i => question[i] \* EntranceWeights[i])

.Sum();

Exit = sum > 0 ? 1 : -1;

}

}

}

public class NeuralNetwork

{

private Neuron[] Neurons;

public NeuralNetwork(int entranceCount, int neuronCount)

{

Neurons = Enumerable.Range(0, neuronCount)

.Select(\_ => new Neuron(entranceCount))

.ToArray();

}

public int[] AskQuestion(int[] question)

=> Neurons.Select(x => x.AskQuestion(question)).ToArray();

public void Training(List<(int[] question, int[] answers)> tuples)

{

var countGoodAnswer = 0;

while (countGoodAnswer < tuples.Count())

{

countGoodAnswer = 0;

for (var i = 0; i < tuples.Count(); ++i)

{

var tuple = tuples[i];

var countGoodAnswerForNeuron = 0;

var answers = AskQuestion(tuple.question);

for (var j = 0; j < Neurons.Count(); ++j)

{

if (answers[j] == tuple.answers[j])

{

countGoodAnswerForNeuron++;

}

else

{

Neurons[j].RecountEntrancesWeights((tuple.question, tuple.answers[j]));

}

}

if (countGoodAnswerForNeuron == Neurons.Count())

{

++countGoodAnswer;

}

}

}

}

}