Министерство ОБРАЗОВАНИя и науки РФ

Федеральное государственное бюджетное образовательное учреждение

высшего образования

«Рыбинский государственный авиационный технический университет

имени П. А. Соловьева»

Кафедра математического и программного обеспеченияэлектронных вычислительных средств

Лабораторная работа

по дисциплине

«Нейрокомпьютерные системы»

на тему

«Модель однослойного перцептрона»

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

Студенты группы ИПБ-19 Абулашвили Д. А., Поникаров К. Н.

Руководитель Паламарь И. Н.

Рыбинск 2022

Оглавление

[1. Обучающая выборка 3](#_Toc121331658)

[2. Матрица весов 4](#_Toc121331659)

[3. Результаты работы сети 5](#_Toc121331660)

[4. Исходный код программы 9](#_Toc121331661)

# 1. Обучающая выборка

Обучение и тестирование сети осуществлялось на примере задачи инвертирования двоичных битов. В качестве обучающей выборки были взяты двоичные представления чисел от 0 до 15 (кроме чисел 1, 4, 9, 11, 14), и соответствующие им инвертированные значения – ожидаемый результат работы сети. Обучающая выборка представлена в таблице 1.1.

Таблица 1.1 – Обучающая выборка

|  |  |  |
| --- | --- | --- |
| № | Входной пример | Ожидаемый результат |
| 1 | {0, 0, 0, 0} | {1, 1, 1, 1} |
| 2 | {0, 0, 1, 0} | {1, 1, 0, 1} |
| 3 | {0, 0, 1, 1} | {1, 1, 0, 0} |
| 4 | {0, 1, 0, 1} | {1, 0, 1, 0} |
| 5 | {0, 1, 1, 0} | {1, 0, 0, 1} |
| 6 | {0, 1, 1, 1} | {1, 0, 0, 0} |
| 7 | {1, 0, 0, 0} | {0, 1, 1, 1} |
| 8 | {1, 0, 1, 0} | {0, 1, 0, 1} |
| 9 | {1, 1, 0, 0} | {0, 0, 1, 1} |
| 10 | {1, 1, 0, 1} | {0, 0, 1, 0} |
| 11 | {1, 1, 1, 1} | {0, 0, 0, 0} |

# 2. Матрица весов

Полученная в результате обучения матрица весов сети представлена в таблице 2.1.

Таблица 2.1 – Матрица весов сети

|  |  |  |  |
| --- | --- | --- | --- |
| -0.899 | 0.008 | 8.946E-4 | 0.006 |
| 0.003 | -1.796 | 0.005 | 0.009 |
| 0.009 | 0.006 | -0.891 | 0.001 |
| 0.904 | 0.009 | 0.009 | -1.791 |

# 3. Результаты работы сети

Для тестирования влияния различных значений смещения на процесс обучения была взята скорость обучения, равная 0.3. Значения смещения были равны -1, -0.7, -0.3, 0, 0.3 и 0.7. Условия завершения обучения – ошибка сети меньше 0.001 или отсутствие изменений в матрице весов.

На рисунке 3.1 представлен график зависимости ошибки сети от эпохи для ступенчатой функции активации.

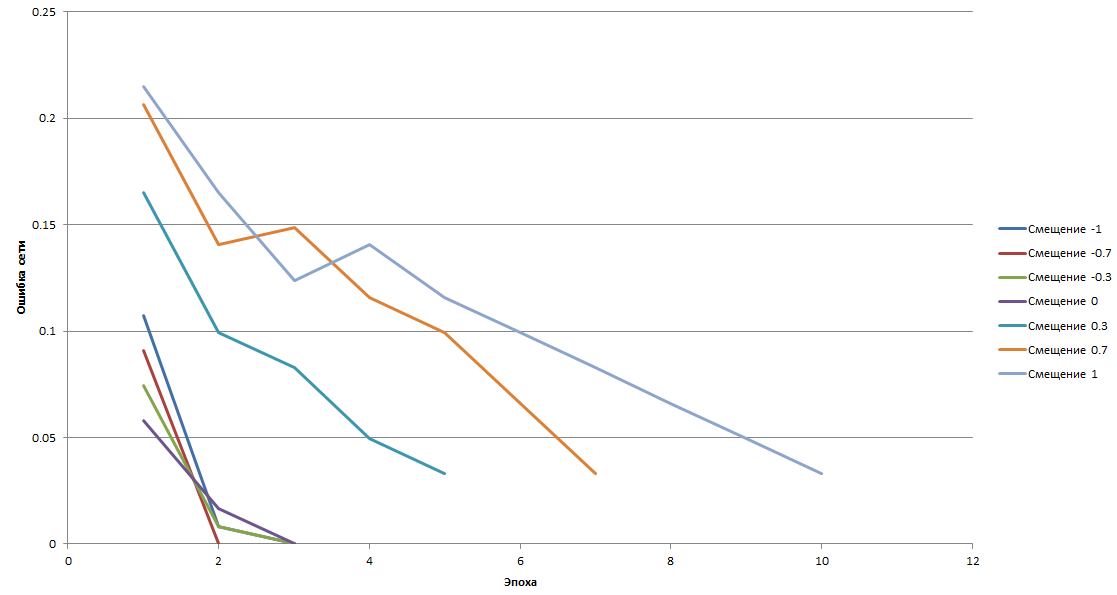


Рисунок 3.1 – Зависимость ошибки сети от эпохи

для ступенчатой функции активации с различными значениями смещения

Как видно, при использовании ступенчатой функции активации сеть обучается быстрее и с меньшей ошибкой при наименьшем смещении. При увеличении смещения увеличиваются и ошибка сети, и количество эпох обучения.

На рисунке 3.2 представлен график зависимости ошибки сети от эпохи для сигмоидной функции активации.

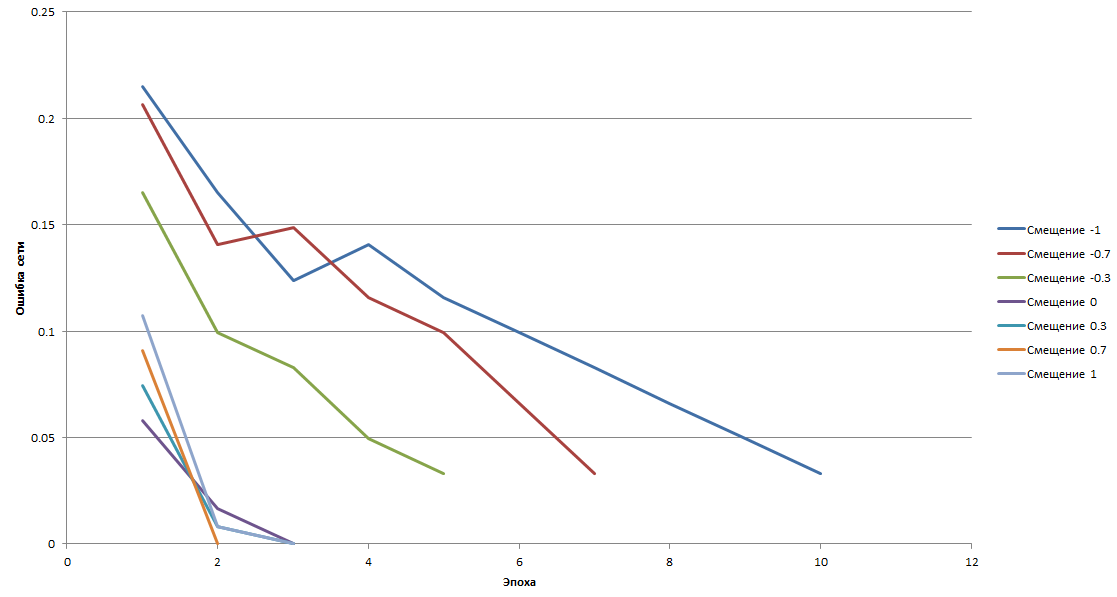


Рисунок 3.2 – Зависимость ошибки сети от эпохи

для сигмоидной функции активации с различными значениями смещения

При использовании сигмоидной функции активации ошибка сети и количество эпох увеличиваются с уменьшением смещения функции. Наилучшие результаты обучения достигаются при большем смещении.

Для тестирования влияния скорости обучения на процесс обучения было взято значение смещение, равное 5, для большего количества эпох обучения. Тестирование проводилось для значений скорости обучения от 0.1 до 0.9.

На рисунке 3.3 представлен график зависимости ошибки сети от эпохи для ступенчатой функции активации.

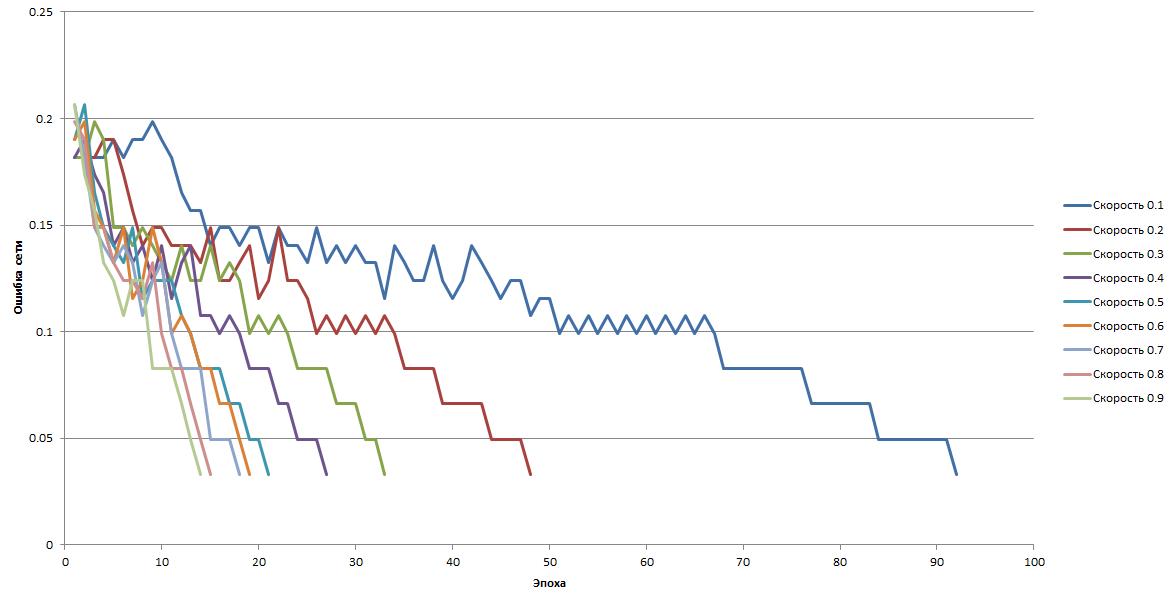


Рисунок 3.3 – Зависимость ошибки сети от эпохи для ступенчатой функции активации с различными значениями скорости обучения

На рисунке 3.4 представлен график зависимости ошибки сети от эпохи для ступенчатой функции активации.

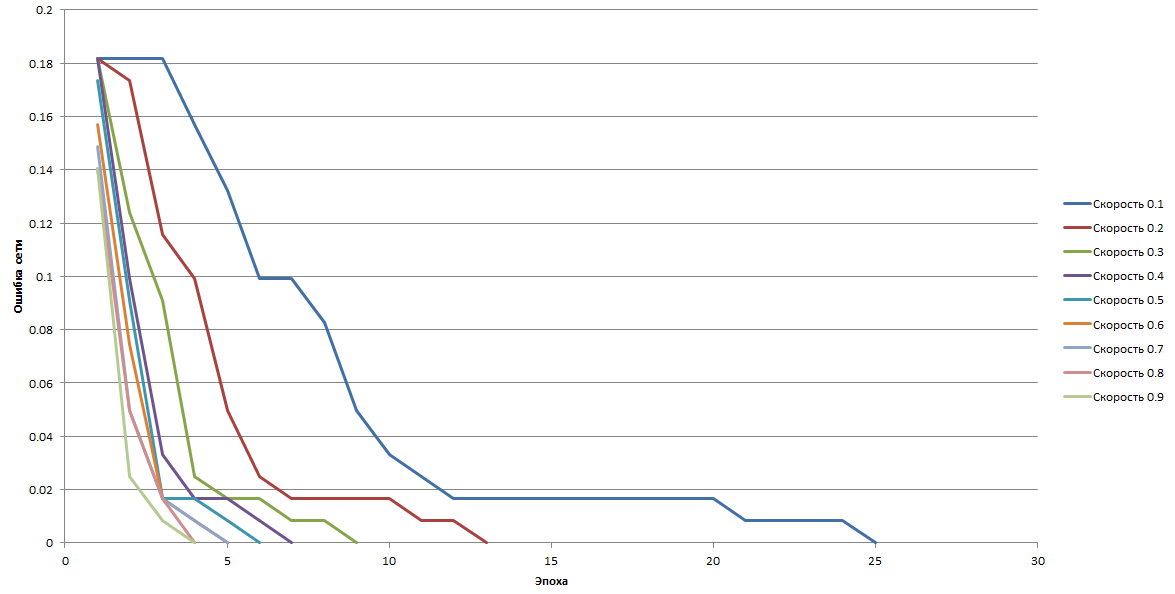


Рисунок 3.3 – Зависимость ошибки сети от эпохи для сигмоидной функции активации с различными значениями скорости обучения

Для обеих функций большая скорость обучения дает меньшее количество эпох обучения и обеспечивает более быстрое снижение ошибки сети.

В таблице 3.1 приведен результат инвертирования сетью двоичных чисел от 0 до 15 с обеими функциями активации. Значение смещения – 0, скорость обучения 0.5.

Таблица 3.1 – Результат работы сети

|  |  |  |  |
| --- | --- | --- | --- |
| № | Входной пример | Ступенчатая функция | Сигмоидная функция |
| 1 | {0, 0, 0, 0} | [1.0, 1.0, 1.0, 1.0] | [1.0, 1.0, 1.0, 1.0] |
| 2 | {0, 0, 0, 1} | [1.0, 1.0, 1.0, 0.0] | [1.0, 1.0, 1.0, 0.0] |
| 3 | {0, 0, 1, 0} | [1.0, 1.0, 0.0, 1.0] | [1.0, 1.0, 0.0, 1.0] |
| 4 | {0, 0, 1, 1} | [1.0, 1.0, 0.0, 0.0] | [1.0, 1.0, 0.0, 0.0] |
| 5 | {0, 1, 0, 0} | [1.0, 0.0, 1.0, 1.0] | [1.0, 0.0, 1.0, 1.0] |
| 6 | {0, 1, 0, 1} | [1.0, 0.0, 1.0, 0.0] | [1.0, 0.0, 1.0, 0.0] |
| 7 | {0, 1, 1, 0} | [1.0, 0.0, 0.0, 1.0] | [1.0, 0.0, 0.0, 1.0] |
| 8 | {0, 1, 1, 1} | [1.0, 0.0, 0.0, 0.0] | [1.0, 0.0, 0.0, 0.0] |
| 9 | {1, 0, 0, 0} | [0.0, 1.0, 1.0, 1.0] | [0.0, 1.0, 1.0, 1.0] |
| 10 | {1, 0, 0, 1} | [0.0, 1.0, 1.0, 0.0] | [0.0, 1.0, 1.0, 0.0] |
| 11 | {1, 0, 1, 0} | [0.0, 1.0, 0.0, 1.0] | [0.0, 1.0, 0.0, 1.0] |
| 12 | {1, 0, 1, 1} | [0.0, 1.0, 0.0, 0.0] | [0.0, 1.0, 0.0, 0.0] |
| 13 | {1, 1, 0, 0} | [0.0, 0.0, 1.0, 1.0] | [0.0, 0.0, 1.0, 1.0] |
| 14 | {1, 1, 0, 1} | [0.0, 0.0, 1.0, 0.0] | [0.0, 0.0, 1.0, 0.0] |
| 15 | {1, 1, 1, 0} | [0.0, 0.0, 0.0, 1.0] | [0.0, 0.0, 0.0, 1.0] |
| 16 | {1, 1, 1, 1} | [0.0, 0.0, 0.0, 0.0] | [0.0, 0.0, 0.0, 0.0] |

# 4. Исходный код программы

Исходный код класса Neuron представлен в листинге 4.1.

Листинг 4.1. – Исходный код класса Neuron

import activationfunc.ActivationFunction;

public class Neuron {

private final int inputNumber;

private double[] weights;

public Neuron(int inputNumber) {

this.inputNumber = inputNumber;

initializeWeights();

}

public double getWeight(int index) {

return weights[index];

}

public void setWeight(int index, double weight) {

weights[index] = weight;

}

public double[] getWeights() {

return weights;

}

private void initializeWeights() {

weights = new double[inputNumber];

for (int index = 0; index < inputNumber; ++index) {

weights[index] = Math.random() / 1000;

}

}

}

Исходный код класса Perceptron представлен в листинге 4.2.

Листинг 4.1. – Исходный код класса Perceptron

import activationfunc.ActivationFunction;

import activationfunc.SigmoidFunction;

import activationfunc.StepFunction;

import java.util.Arrays;

public class Perceptron {

private Neuron[] neurons;

private double[][] neuronInputWeights;

private final ActivationFunction activationFunction;

public Perceptron(int inputNumber, int outputNumber, ActivationFunction activationFunction) {

this.activationFunction = activationFunction;

initializeNeurons(inputNumber, outputNumber);

}

public void learn(double[][] inputSamples, double[][] outputSamples, double learningRate, double eps) {

int epochNumber = 0;

double rmsError;

do {

rmsError = 0;

saveNeuronInputWeights();

for (int index = 0; index < inputSamples.length; ++index) {

double[] neuronsOutputValues = getNeuronsOutputValues(getNeuronsPotentials(inputSamples[index]));

double[] neuronsErrors = getNeuronsErrors(neuronsOutputValues, outputSamples[index]);

setNeuronsWeights(inputSamples[index], neuronsErrors, learningRate);

rmsError += getRmsError(neuronsOutputValues, outputSamples[index], inputSamples.length);

}

rmsError /= inputSamples.length;

System.out.println(rmsError);

++epochNumber;

} while (rmsError >= eps && isWeightsChange());

System.out.println("Epoch number: " + epochNumber);

}

public double[] getResult(double[] inputSample) {

return getNeuronsOutputValues(getNeuronsPotentials(inputSample));

}

private boolean isWeightsChange() {

for (int index1 = 0; index1 < neurons.length; ++index1) {

for (int index2 = 0; index2 < neurons[index1].getWeights().length; ++index2) {

if (Double.compare(neurons[index1].getWeight(index2), neuronInputWeights[index1][index2]) != 0) {

return true;

}

}

}

return false;

}

private void saveNeuronInputWeights() {

neuronInputWeights = new double[neurons.length][];

for (int index = 0; index < neurons.length; ++index) {

neuronInputWeights[index] = neurons[index].getWeights().clone();

//System.out.println(Arrays.toString(neuronInputWeights[index]));

}

}

private double getRmsError(double[] neuronsOutputValues, double[] expectedValues, int samplesNumber) {

double rmsError = 0;

for (int index = 0; index < neuronsOutputValues.length; ++index) {

rmsError += Math.pow(expectedValues[index] - neuronsOutputValues[index], 2);

}

return rmsError / samplesNumber;

}

private void setNeuronsWeights(double[] inputSample, double[] neuronsErrors, double learningRate) {

for (int index1 = 0; index1 < neurons.length; ++index1) {

for (int index2 = 0; index2 < neurons[index1].getWeights().length; ++index2) {

neurons[index1].setWeight(

index2,

neurons[index1].getWeight(index2) + learningRate \* inputSample[index2] \* neuronsErrors[index1]

);

}

}

}

private double[] getNeuronsErrors(double[] neuronsOutputValues, double[] expectedValues) {

double[] neuronsErrors = new double[neuronsOutputValues.length];

for (int index = 0; index < neuronsOutputValues.length; ++index) {

neuronsErrors[index] = expectedValues[index] - neuronsOutputValues[index];

}

//System.out.println(Arrays.toString(neuronsErrors));

return neuronsErrors;

}

private double[] getNeuronsOutputValues(double[] potentials) {

double[] neuronsOutputValues = new double[potentials.length];

for (int index = 0; index < potentials.length; ++index) {

neuronsOutputValues[index] = activationFunction.getY(potentials[index], 0);

}

return neuronsOutputValues;

}

private double[] getNeuronsPotentials(double[] inputSample) {

double[] neuronsPotentials = new double[neurons.length];

for (int index = 0; index < neurons.length; ++index) {

neuronsPotentials[index] = getNeuronPotential(neurons[index], inputSample);

}

return neuronsPotentials;

}

private double getNeuronPotential(Neuron neuron, double[] inputSample) {

double potential = 0;

for (int index = 0; index < neuron.getWeights().length; ++index) {

potential += neuron.getWeight(index) \* inputSample[index];

}

return potential;

}

private void initializeNeurons(int inputNumber, int outputNumber) {

neurons = new Neuron[outputNumber];

for (int index = 0; index < outputNumber; ++index) {

neurons[index] = new Neuron(inputNumber);

}

}

public static void main(String[] args) {

double[][] inputSamples = {

{0, 0, 0, 0}, {0, 0, 1, 0}, {0, 0, 1, 1},

{0, 1, 0, 1}, {0, 1, 1, 0}, {0, 1, 1, 1},

{1, 0, 0, 0}, {1, 0, 1, 0}, {1, 1, 0, 0},

{1, 1, 0, 1}, {1, 1, 1, 1}

};

double[][] outputSamples = {

{1, 1, 1, 1}, {1, 1, 0, 1}, {1, 1, 0, 0},

{1, 0, 1, 0}, {1, 0, 0, 1}, {1, 0, 0, 0},

{0, 1, 1, 1}, {0, 1, 0, 1}, {0, 0, 1, 1},

{0, 0, 1, 0}, {0, 0, 0, 0}

};

double[][] testSamples = {

{0, 0, 0, 0}, {0, 0, 0, 1}, {0, 0, 1, 0}, {0, 0, 1, 1},

{0, 1, 0, 0}, {0, 1, 0, 1}, {0, 1, 1, 0}, {0, 1, 1, 1},

{1, 0, 0, 0}, {1, 0, 0, 1}, {1, 0, 1, 0}, {1, 0, 1, 1},

{1, 1, 0, 0}, {1, 1, 0, 1}, {1, 1, 1, 0}, {1, 1, 1, 1}

};

Perceptron perceptron = new Perceptron(inputSamples[0].length, outputSamples[0].length, new StepFunction());

perceptron.learn(inputSamples, outputSamples, 0.1, 0.001);

for (double[] inputSample : testSamples) {

System.out.println(Arrays.toString(perceptron.getResult(inputSample)));

perceptron.getResult(inputSample);

}

}

}

Исходный код интерфейса ActivationFunction представлен в листинге 4.3.

Листинг 4.1. – Исходный код интерфейса ActivationFunction

package activationfunc;

public interface ActivationFunction {

double getY(double potential, double offset);

}

Исходный код класса SigmoidFunction представлен в листинге 4.4.

Листинг 4.4. – Исходный код класса SigmoidFunction

package activationfunc;

public class SigmoidFunction implements ActivationFunction {

@Override

public double getY(double potential, double offset) {

return Math.round(1 / (1 + Math.exp(-(potential + offset))));

}

}

Исходный код класса StepFunction представлен в листинге 4.5.

Листинг 4.5. – Исходный код класса StepFunction

package activationfunc;

public class StepFunction implements ActivationFunction {

@Override

public double getY(double potential, double offset) {

return Double.compare(potential, offset) < 0 ? 0 : 1;

}

}