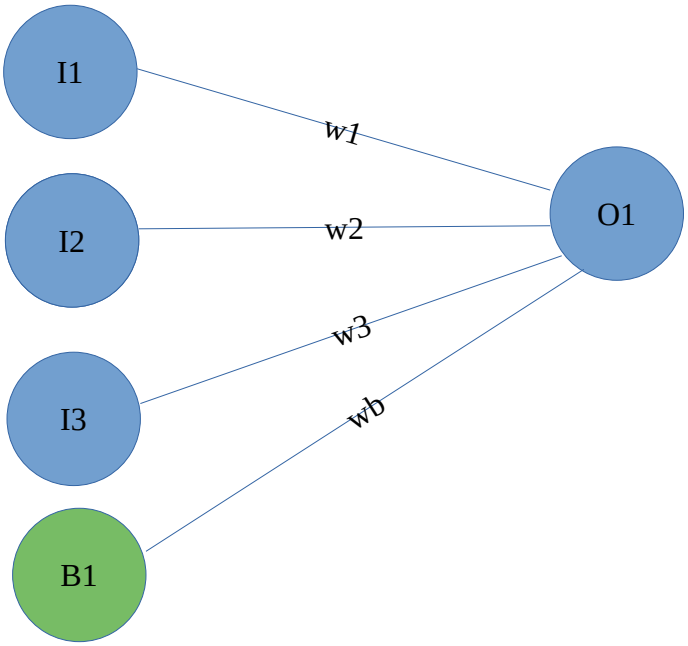
Задание: синтезировать нейронную сеть, решающую задачу «Тернарной логической функции «ЗИ»»

1) Таблица истинности задачи ЗИ

x	**y**	**z**	**x&y&z**
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

2) Т.к. в задаче есть 3 входных параметра и 1 выходной структура НС, для ее решения будет выглядеть следующим образом



3) Принято решение использовать в качестве функции активации — ступенчатую функцию вида:

$$y = 1, x > 0$$
$$y = 0, x <= 0$$

Такая функция отлично работает для бинарной классификации.

4) Начальные веса синапсов имеют вес:
w1=0.19655392355925017; w2=-0.4506764929921737; w3=-0.1083241250429996

wb = -0.893310190784654 — константа

5) Реализация решения поставленной задачи тернарной функции ЗИ проводилась на языке java с применением библиотеки машинного обучения Neuroph ver. 2.98. В данной библиотеке в выходном слое по умолчанию используется класс ThresholdNeuron, который на модели НС

можно представить в виде bias нейрона. Величина thresh в ThresholdNeuron задается случайным образом при инициализации нейрона и является константой. Соотношение между весом wb и thresh определяется как

$$wb = -thresh$$

Neuroph ver. 2.98

Начальные веса: [0.19655392355925017, -0.4506764929921737, -0.1083241250429996]
Thresh: 0.893310190784654

Обучаем нейросеть...
Эпоха: 1, Ошибка: 0,0625000000
Веса после 1го прохода: [-0.18205682911387475, 0.08613058343431582, 0.29046179083986867]
Thresh: 0.893310190784654
Input: [0.0, 0.0, 0.0] Output: [0,0000000000]
Input: [0.0, 0.0, 1.0] Output: [0,0000000000]
Input: [0.0, 1.0, 0.0] Output: [0,0000000000]
Input: [0.0, 1.0, 1.0] Output: [0,0000000000]
Input: [1.0, 0.0, 0.0] Output: [0,0000000000]
Input: [1.0, 0.0, 1.0] Output: [0,0000000000]
Input: [1.0, 1.0, 0.0] Output: [0,0000000000]
Input: [1.0, 1.0, 1.0] Output: [0,0000000000]

Эпоха: 2, Ошибка: 0,0625000000
Эпоха: 3, Ошибка: 0,0625000000
Эпоха: 4, Ошибка: 0,0625000000
Эпоха: 5, Ошибка: 0,0625000000
Эпоха: 6, Ошибка: 0,0000000000
Обучение завершено за 9 миллисекунд.

Сохраняем результат обучения в learn_result.nnet.
Веса обученной нейросети: [0.11794317088612527, 0.28613058343431585, 0.4904617908398686]
Thresh: 0.893310190784654
Ошибка обученной нейросети: 0,0000000000
Количество эпох: 6
Проверка:
Input: [0.0, 0.0, 0.0] Output: [0,0000000000]
Input: [0.0, 0.0, 1.0] Output: [0,0000000000]
Input: [0.0, 1.0, 0.0] Output: [0,0000000000]
Input: [0.0, 1.0, 1.0] Output: [0,0000000000]
Input: [1.0, 0.0, 0.0] Output: [0,0000000000]
Input: [1.0, 0.0, 1.0] Output: [0,0000000000]
Input: [1.0, 1.0, 0.0] Output: [0,0000000000]
Input: [1.0, 1.0, 1.0] Output: [1,0000000000]

6) Реализация процесса обучения НС, решающей задачу тернарной функции ЗИ, имеет вид:

тренировочный сет dataset.csv
x;y;z;x&y&z
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

```
package ru.sergst;

import org.neuroph.core.NeuralNetwork;
import org.neuroph.core.data.DataSet;
import org.neuroph.core.data.DataSetRow;
import org.neuroph.nnet.Perceptron;
import org.neuroph.nnet.learning.BackPropagation;
import org.neuroph.util.Neuroph;
import org.neuroph.util.TransferFunctionType;

import java.net.URISyntaxException;
import java.time.Duration;
import java.time.Instant;
import java.util.Arrays;

import static org.neuroph.core.events.LearningEvent.Type.EPOCH_ENDED;

public class Main {

    private static final String nnetFileName = "learn_result.nnet";

    public static void main(String[] args) {
        System.out.printf("%s ver. %s%n%n", Neuroph.class.getSimpleName(),
Neuroph.getVersion());
        var dataset = DataSet.createFromFile("dataset.csv", 3, 1, ";", true);
        var currentDataSet = dataset;

        var perceptron = new Perceptron
                (3, 1, TransferFunctionType.STEP);
        perceptron.setLearningRule(new BackPropagation());
        var learningRule = (BackPropagation) perceptron.getLearningRule();
        learningRule.setMaxIterations(100_000);
        learningRule.addListener(event -> {
            if (event.getEventType() == EPOCH_ENDED) {
                System.out.printf(
                        "Эпоха: %d, Ошибка: %.10f\n",
                        learningRule.getCurrentIteration(),
                        learningRule.getTotalNetworkError()
                );
                if (learningRule.getCurrentIteration() == 1) {
                    System.out.printf("Веса после 1го прохода: %s%n",
Arrays.toString(perceptron.getWeights()));
                    System.out.printf("Thresh: %.10f%n%n", ((ThresholdNeuron)
perceptron.getOutputNeurons().get(0)).getThresh());
                    testNeuralNetwork(perceptron, dataset);
                }
            }
        });
        Runtime.getRuntime().addShutdownHook(new Thread(() ->
perceptron.save("learn_result.nnet")));

        System.out.printf("Начальные веса: %s%n", Arrays.toString(perceptron.getWeights()));
        System.out.printf("Thresh: %.10f%n%n", ((ThresholdNeuron)
perceptron.getOutputNeurons().get(0)).getThresh());
        System.out.println("Обучаем нейросеть...");
        var start = Instant.now();
        do {
            perceptron.randomizeWeights();
            perceptron.learn(currentDataSet);
        } while (learningRule.getTotalNetworkError() > learningRule.getMaxError());
        System.out.printf("Обучение завершено за %s миллисекунд.\n\n",
Duration.between(start, Instant.now()).toMillis());
```

```
        System.out.printf("Сохраняем результат обучения в %s.%n", nnetFileName);
        perceptron.save(nnetFileName);
        System.out.printf("Веса обученной нейросети: %s%n",
Arrays.toString(perceptron.getWeights()));
        System.out.printf("Thresh: %.10f%n", ((ThresholdNeuron)
perceptron.getOutputNeurons().get(0)).getThresh());
System.out.printf("Ошибка обученной нейросети: %.10f%n",
learningRule.getTotalNetworkError());
        System.out.printf("Количество эпох: %d%n", learningRule.getCurrentIteration());
        System.out.println("Проверка:");
        testNeuralNetwork(perceptron, currentDataSet);
    }

    public static void testNeuralNetwork(NeuralNetwork nnet, DataSet tset) {

        for (DataSetRow dataRow : tset.getRows()) {

            nnet.setInput(dataRow.getInput());
            nnet.calculate();
            double[ ] networkOutput = nnet.getOutput();
            System.out.print("Input: " + Arrays.toString(dataRow.getInput()) );
            System.out.println(" Output: " +
Arrays.toString(Arrays.stream(networkOutput).mapToObj("%.10f"::formatted).toArray()));
        }
        System.out.println();
    }
}
```

В результате обучения сети методом обратного распространения ошибки были получены следующие веса синапсов
w1=0.11794317088612527; w2=0.28613058343431585; w3= 0.4904617908398686
wb = -0.893310190784654
Алгоритму понадобилось 6 итераций.
Отклик натренированной сети:
0 AND 0 AND 0 = 0,0000;
0 AND 0 AND 1 = 0,0000;
0 AND 1 AND 0 = 0,0000;
0 AND 1 AND 1 = 0,0000;
1 AND 0 AND 0 = 0,0000;
1 AND 0 AND 1 = 0,0000;
1 AND 1 AND 0 = 0,0000;
1 AND 1 AND 1 = 1,0000.
Ошибка модели составила 0,0000.

Проверка проводилась при помощи такого кода на java:

```
var bias = -0.893310190784654;
var weights = new double[] {0.11794317088612527, 0.28613058343431585,
0.4904617908398686};
var inputSet = new double[][] {
        {0, 0, 0},
        {0, 0, 1},
        {0, 1, 0},
        {0, 1, 1},
        {1, 0, 0},
        {1, 0, 1},
        {1, 1, 0},
        {1, 1, 1},
```

```
};
var transferFunction = (Function<Double, Double>) n -> n > 0d ? 1.0 : 0.0;
for (double[] input : inputSet) {
    var output = 0d;
    for (int i = 0; i < input.length; i++) {
        output += input[i] * weights[i];
    }
    output += bias;
    var result = transferFunction.apply(output);

    System.out.printf("input %s, output %.1f%n", Arrays.toString(input), result);
}
```

Результат проверки:

```
input [0.0, 0.0, 0.0], output 0,0000000000
input [0.0, 0.0, 1.0], output 0,0000000000
input [0.0, 1.0, 0.0], output 0,0000000000
input [0.0, 1.0, 1.0], output 0,0000000000
input [1.0, 0.0, 0.0], output 0,0000000000
input [1.0, 0.0, 1.0], output 0,0000000000
input [1.0, 1.0, 0.0], output 0,0000000000
input [1.0, 1.0, 1.0], output 1,0000000000
```