**Ордена Трудового Красного Знамени**

**Федеральное государственное бюджетное образовательное учреждение**

**Высшего профессионального образования**

**Московский технический университет связи и информатики**

Факультет повышения квалификации

**Лабораторная работа №1**

**Вариант 10**

Выполнил:

магистрант гр. 3МПП1601

Климов С.А.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

подпись дата

Проверил:

к.т.н. Махров С.С.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

подпись дата

**Москва 2016**

**Обучение однослойного персептрона методом коррекции по ошибке через дельта-правило**

**Цель работы**. Изучить алгоритм обучения однослойного персептрона методом коррекции по ошибке через дельта-правило.

**Задание.** В соответствии с вариантом, необходимо обучить нейронную сеть распознавать указанные 4 символа. На каждый символ необходимо подготовить 4 обучающих образа с использованием разных шрифтов. Соответственно, всего будет 16 обучающих образов: 4 образа первым шрифтом, 4 образа, вторым шрифтом и т.д. Тестовая выборка должна содержать по 1 образу на каждый из 4-х символов. Символы должны быть написаны другим шрифтом, который не был использован в обучающей выборке.

**Теоретические сведения.** Алгоритм обучения с положительным и отрицательным подкреплением можно представить в более общей форме – в виде дельта-правила. Если за *dj* обозначить желаемый выходной сигнал *j*-го нейрона (от слов desired response, что в переводе с английского означает «желаемый отклик»), то на каждой эпохе (итерации) обучения можно рассчитывать разницу между желаемым *dj* ответом *j*-го нейрона и реальным значением *yj*, вычисляемым на его выходе. Соответственно, ошибкой выхода персептрона называется следующее выражение:

Тогда относительно алгоритма обучения персептрона с положительным и отрицательным подкреплением:

- случай *εj = 0* соответствует шагу 4а;

- случай *εj > 0* соответствует шагу 4б;

- случай *εj < 0* соответствует шагу 4в.

Идея алгоритма обучения персептронного слоя с помощью правила Хебба сохранится, если итерационный процесс корректировки весов вести по формулам:

|  |
| --- |
|  |
|  |

где *wi,j(t)* и *wi,j(t+1)* – старое и новое значения синаптических весов персептрона, *Δwi,j(t + 1)* – новое значение величины коррекции синаптических весов персептрона, *i* – номер входного сигнала, *j* – номер нейрона. Приведенные формулы называются дельта-правилом.

Кроме того, можно получить аналогичную итерационную формулу для подстройки нейронного смещения *b*, если учесть, что его можно интерпретировать как вес *w0,j* дополнительного входа *x0,j*, значение которого всегда равно 1:

|  |
| --- |
|  |
|  |

В итерационные формулы полезно ввести коэффициент скорости обучения *η*, с помощью которого можно управлять величиной коррекции синаптических весов и нейронного смещения:

|  |
| --- |
|  |
|  |

При слишком больших значениях коэффициента *η* обычно теряется устойчивость процесса обучения, тогда как при слишком малых – увеличивается время обучения. На практике коэффициент скорости обучения *η* обычно задают в пределах от 0.05 до 1.

Алгоритм обучения персептрона с использованием этих формул известен под названием обучения с коррекцией по ошибке или дельта-правило.

В общем виде, алгоритм обучения с коррекцией по ошибке через дельта-правило, представлен ниже.

***Алгоритм обучения*** ***методом коррекции по ошибке через дельта-правило***

Шаг 1. Подготовить обучающую выборку, каждый элемент которой будет состоять из пар *(X, D)m (m=1,…q)* – обучающего вектора *X = (x1,…,xn)* (*i=1,…,n)* с вектором желаемых значений *D = (d1,…,dk) (j=1,…,k*) выходов персептрона.

Шаг 2. Генератором случайных чисел всем синаптическим весам *wij* и нейронным смещениям *w0,j* (*i=1,…,n; j=1,…,k*) присваиваются некоторые малые случайные значения.

Шаг 3. Из обучающей выборки *(X, D)1,…,(X, D)q*, взять следующий по счету вектор *Xт = (x1,…,xn)* и подать его на входы персептрона *x1,…,xn*. Сигналам нейронных входов смещения *x0*присваиваются единичные значения: *x0 = 1*.

Шаг 4. Для каждого *j*-го нейрона вычислить взвешенную сумму входных сигналов *netj* и выходной сигнал *yj* на основании функции активации *f*:

|  |
| --- |
|  |
|  |

Шаг 5. Для каждого *j*-го нейрона вычислить его ошибку:

где *dj* – вектор правильных (желаемых) ответов персептрона.

Шаг 6. Произвести коррекцию синаптических весов *j*-го нейрона и нейронных смещений:

|  |
| --- |
|  |
|  |

где t – номер итерации.

Шаги 4-6 повторяются для всех нейронов персептронного слоя при подаче конкретного образа.

Шаги 3-6 выполняются последовательно для каждого входного образа, на котором обучается персептрон.

Шаг 7. После подачи последнего обучающего вектора, проверить критерий останова обучения, если он выполняется, то завершить обучение. В противном случае – возврат к шагу 3.

Критерии останова алгоритма обучения могут быть следующими:

1. Значения синаптических весов *wi,j* перестают изменяться;
2. Перестают появляться неправильные выходные сигналы *yj*;
3. Превышен установленный лимит количества эпох (либо итераций).

Указанные критерии останова обучения используются как по отдельности, так и вместе. Важным замечанием будет то, что 3-ий критерий, при задании недостаточного количества эпох, может привести к недообучению сети, поэтому его использование в промышленном применении не рекомендуется, но в учебных целях он показателен и удобен.

**Ход выполнения работы**

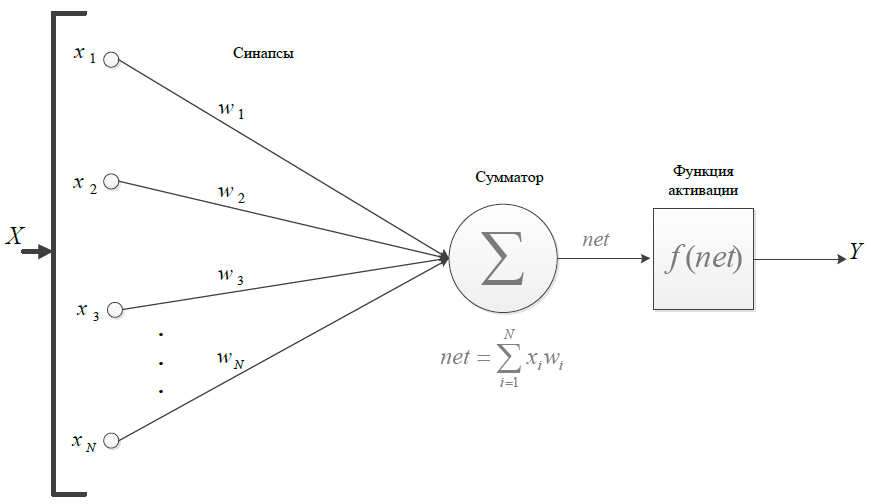
На языке программирования Java была реализована искусственная нейронная сеть, использующая алгоритм обучения однослойного персептрона методом коррекции по ошибке через дельта-правило.

Реализованы в том числе следующие классы:

1. “Neuron” – модель нейрона, являющегося простым ассоциативным элементов (А-элементом) персептрона с функцией активации вида:

C:\Users\simon\AppData\Local\Temp\ScreenClip.png

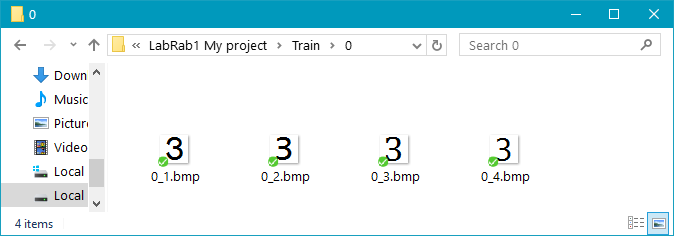
Структурная схема A-элемента:

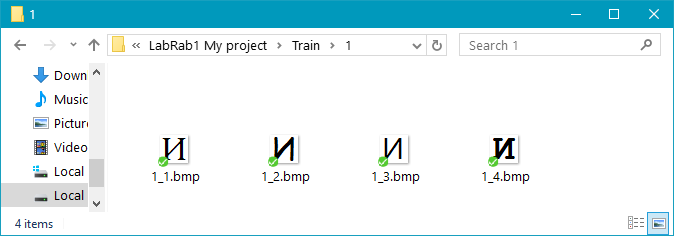


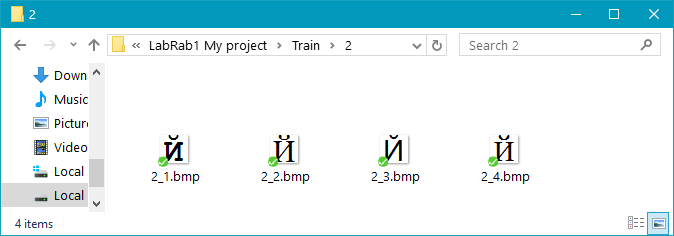
Задание начальных случайных весов производится методом randomizeWeights в диапазоне от rangeMin = -0.3 до rangeMax = 0.3. Коррекция синаптических весов производится методом correctWeights в соответствии с входным значением deltaWeight. Выходное значение нейрона рассчитывается функцией calcOut путем суммирования произведения элементов поданного на вход нейрона вектора Х на соответствующие элементы вектора синаптических весов weight и последующего применения к сумме функции активации activationFunc.

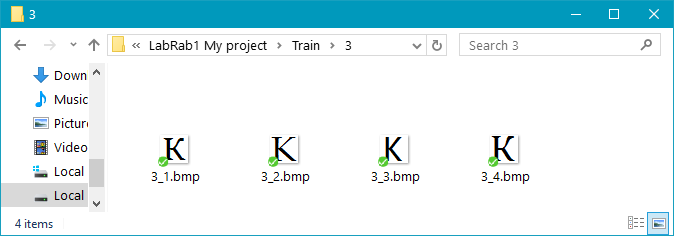
1. “NeuralNet” – модель нейронной сети – однослойного персептрона. Производит перебор обучающих векторов (шаг 3 алгоритма) и перебор нейронов (шаг 4), а затем вычисляет ошибку каждого j-го нейрона (шаг 5) и проходит цикл коррекции синаптических весов, вызывая метод correctWeights.

В соответствии с вариантом 10, на каждый символ от «З» до «К» подготовлено 4 обучающих образа с использованием разных шрифтов. Файлы обучающих образов созданы в формате .bmp с разрешением 28х28 (что соответствует 784 входным элементам) и глубиной цвета 24 бита и размещены в папке Train\[порядковый\_номер\_образа]:









Подготовлена тестовая выборка, содержащая по 1 образу на каждый из 4-х символов, набранных шрифтом, который не был использован в обучающей выборке, и размещена в папке Test\[порядковый\_номер\_образа]:

C:\Users\simon\AppData\Local\Microsoft\Windows\INetCacheContent.Word\2_0.bmp



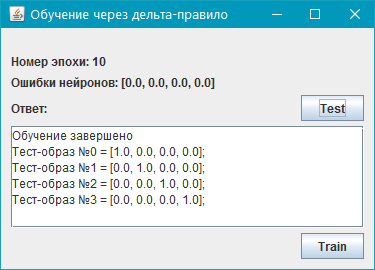
Нам необходимо обучить нейронную сеть распознавать указанные 4 символа.

Коэффициент скорости обучения η возьмем равным 0.5, т.к. при слишком больших значениях коэффициента обычно теряется устойчивость процесса обучения, тогда как при слишком малых – увеличивается время обучения, и коэффициент скорости обучения η обычно задают в пределах от 0.05 до 1.

Используем третий критерий останова алгоритма обучения: превышение количества эпох epochNumber равного 10.

Запустим программу. Осуществим настройку синаптических весов нейронной сети путем проведения 10 эпох обучения на обучающей выборке. Проведем оценку качества обучения через проверку ее работы на тестовых образах.

Программа выдает результат тестирования:



Нейронная сеть классифицирует тест-образ №0 как образ №0 обучающей выборки, тест-образ №1 как образ №1 обучающей выборки и т.д., т.е. *все тестовые образы определены правильно*.

Однако при повторном обучении сети и последующей проверке ее работы на тестовых образах можем заметить частые ошибки в распознавании тест-образа №2 (буквы «Й»):

1. Тест-образ №2 не распознан:

Тест-образ №0 = [**1.0**, 0.0, 0.0, 0.0];

Тест-образ №1 = [0.0, **1.0**, 0.0, 0.0];

Тест-образ №2 = [0.0, 0.0, **0.0**, 0.0];

Тест-образ №3 = [0.0, 0.0, 0.0, **1.0**];

1. Тест-образ №2 классифицирован одновременно как образ №1 (буква «И») и как образ №2 (буква «Й»):

Тест-образ №0 = [**1.0**, 0.0, 0.0, 0.0];

Тест-образ №1 = [0.0, **1.0**, 0.0, 0.0];

Тест-образ №2 = [0.0, **1.0, 1.0**, 0.0];

Тест-образ №3 = [0.0, 0.0, 0.0, **1.0**];

3) Тест-образ №2 классифицирован как образ №1 (буква «И»):

Тест-образ №0 = [**1.0**, 0.0, 0.0, 0.0];

Тест-образ №1 = [0.0, **1.0**, 0.0, 0.0];

Тест-образ №2 = [0.0, **1.0**, **0.0**, 0.0];

Тест-образ №3 = [0.0, 0.0, 0.0, **1.0**];

Попробуем скорректировать параметры нейронной сети для исключения вышеописанных ошибок распознавания образа №2:

1. Увеличим количество эпох с 10 до 50.

Результат: неудовлетворительный.

1. Уменьшим коэффициент скорости обучения η с 0.5 до 0.1 и затем до 0.05.

Результат: количество ошибок в распознавании тест-образа №2 существенно снизилось (до 40%), но по-прежнему сеть иногда распознает образ №2 как образ №1.

1. Будем задавать начальные синаптические веса не из диапазона [-0.3, 0.3], а из [-0.003, 0.003] и затем из [-0.0003, 0.0003].

Результат: количество ошибок в распознавании тест-образа №2 *увеличилось*, а также стали эпизодически появляться ошибки в распознавании образа №1. В целом, участились «ложные» срабатывания, когда сеть классифицирует тестовый образ как соответствующий правильному классу, а также еще какому-либо классу.

**Вывод.**

Путем создания модели однослойного персептрона на языке программирования Java и его обучения методом коррекции по ошибке через дельта-правило была решена классическая задача классификации, т.е. создание группировки множества объектов на основе учета общих признаков этих объектов и закономерных связей между ними.

После настройки синаптических весов искусственной нейронной сети путем проведения 10 эпох обучения на обучающей выборке, сеть могла уверенно классифицировать каждый из тестовых образов. Однако наблюдались ошибки в определении образа №2 (буквы «Й»), который часто распознавался как образ №1 (буква «И»), что легко объясняется их похожим начертанием.

Для улучшения качества обучения было увеличено количество эпох, уменьшен коэффициент скорости обучения, а также уменьшен диапазон случайного задания начальных синаптических весов. Ни одна из этих мер не привела к улучшению качества обучения; напротив, участились случаи ложного срабатывания нейронной сети.

Вероятно, используемый простейший метод обучения персептрона с одним обучаемым слоем методом коррекции по ошибке через дельта-правило (подкрепление с коррекцией ошибок) не позволяет достичь желаемого качества обучения, при котором искусственная нейронная сеть сможет безошибочно классифицировать образы в виде графического изображения символов.

**Исходный код программы**

1. Main.java:

**public class** Main {  
 **public static void** main(String[] args) {  
 GUI gui = **new** GUI();  
 }  
}

1. Neuron.java:

**import** java.util.Random;  
  
**public class** Neuron {  
 **private double**[] **weight**;  
 **private double out**;  
 **public static** Random *random* = **new** Random();  
 **public double rangeMin** = -0.0003;  
 **public double rangeMax** = 0.0003;  
  
 **public** Neuron(**int** weightCount) {  
 **this**.**weight** = **new double**[weightCount];  
 **this**.**out** = 0.0;  
 randomizeWeights();  
 }  
  
 **public double** getOut() {  
 **return out**;  
 }  
  
 **public void** correctWeights(**double**[] deltaWeight) {  
 **for** (**int** i = 0; i < **weight**.**length**; i++) {  
 **weight**[i] += deltaWeight[i];  
 }  
 }  
  
 **public double**[] getWeight() {  
 **return weight**;  
 }  
  
 **public void** randomizeWeights() {  
 **for** (**int** i = 0; i < **weight**.**length**; i++) {  
 **weight**[i] = **rangeMin** + (**rangeMax** - **rangeMin**) \* *random*.nextDouble();  
 }  
 }  
  
 **private double** activationFunc(**double** val) {  
 **if** (val >= 0)  
 **return** 1.0;  
 **else  
 return** 0.0;  
 }  
  
 **public void** calcOut (**double**[] x) {  
 **double** sum = 0.0;  
 **for** (**int** i = 0; i < x.**length**; i++) {  
 sum += x[i] \* **weight**[i];  
 }  
 **this**.**out** = activationFunc(sum);  
 }  
}

1. Vector.java:

**public class** Vector {  
 **private double**[] **x**; *// массив входных весов* **private double**[] **desireOutputs**; *// массив желаемых значений  
  
 // конструктор вектора* **public** Vector(**double**[] x, **double**[] desireOutputs) {  
 **this**.**x** = **new double**[x.**length** +1];  
 x[0] = 1;  
 **for** (**int** i = 1; i < x.**length**; i++) {  
 **this**.**x**[i] = x[i - 1];  
 }  
 **this**.**desireOutputs** = desireOutputs;  
 }  
  
 **public double**[] getX() {  
 **return x**;  
 }  
  
 **public double**[] getDesireOutputs() {  
 **return desireOutputs**;  
 }  
}

1. NeuralNet.java:

**public class** NeuralNet {  
 **private int inputVectorSize**;  
 **private** Neuron[] **layer**;  
 **private int epochNumber**;  
 **private boolean complete**;  
 **private double**[] **error**;  
 **private double eta** = 0.05;  
  
 **public** NeuralNet(**int** inputVectorSize, **int** outputNeuronsCount) {  
 **this**.**inputVectorSize** = inputVectorSize;  
 **layer** = **new** Neuron[outputNeuronsCount];  
 **for** (**int** j = 0; j < outputNeuronsCount; j++) {  
 **layer**[j] = **new** Neuron(inputVectorSize);  
 }  
 **error** = **new double**[**layer**.**length**];  
 }  
  
 **public double**[] getError() {  
 **return error**;  
 }  
  
 **public int** getEpochNumber() {  
 **return epochNumber**;  
 }  
  
 **public boolean** isComplete() {  
 **return complete**;  
 }  
  
 **public void** setComplete(**boolean** complete) {  
 **this**.**complete** = complete;  
 }  
  
 **public void** train(Vector[] vectorSet) {  
 **epochNumber** = 0;  
 **do** {  
  
 *// Шаг 3  
 // перебор обучающих векторов* **for** (**int** m = 0; m < vectorSet.**length**; m++) {  
  
 *// Шаг 4  
 // перебор нейронов* **for** (**int** j = 0; j < **layer**.**length**; j++) {  
 **layer**[j].calcOut(vectorSet[m].getX());  
 }  
   
 *// создаем массив для хранения ошибки каждого нейрона* **error** = **new double**[**layer**.**length**];  
 **double** sumError = 0.0;  
   
 *// Шаг 5 алгоритма* **for** (**int** j = 0; j < **layer**.**length**; j++) {  
 *// считаем ошибку каждого j-го нейрона* **error**[j] = vectorSet[m].getDesireOutputs()[j] - **layer**[j].getOut();  
 sumError += **error**[j];  
 }  
  
 *// Thread.sleep(100);  
  
 // Шаг 6. Цикл коррекции синаптических весов* **for** (**int** j = 0; j < **layer**.**length**; j++) {  
 **int** n = **layer**[j].getWeight().**length**; *// кол-во синаптических весов у j-го нейрона* **double**[] deltaWeight = **new double**[n];  
  
 **for** (**int** i = 0; i < n; i++) {  
 deltaWeight[i] += **eta** \* **error**[j] \* vectorSet[m].getX()[i];  
 }  
 **layer**[j].correctWeights(deltaWeight);  
 }  
  
 }  
  
 **epochNumber**++;  
 } **while** (**epochNumber** <= 50); *// критерий останова обучения* **complete** = **true**;  
 }  
  
 **public double**[] test(**double**[] vector) {  
 **double**[] outVector = **new double**[**layer**.**length**];  
 **for** (**int** j = 0; j < **layer**.**length**; j++) {  
 **layer**[j].calcOut(vector);  
 outVector[j] = **layer**[j].getOut();  
 }  
 **return** outVector;  
 }  
  
}

1. GUI.java:

**import** javax.imageio.ImageIO;  
**import** javax.swing.\*;  
**import** java.awt.\*;  
**import** java.awt.event.ActionEvent;  
**import** java.awt.event.ActionListener;  
**import** java.awt.image.BufferedImage;  
**import** java.io.File;  
**import** java.io.IOException;  
**import** java.util.ArrayList;  
**import** java.util.Arrays;  
**import** java.util.List;  
**import static** javax.swing.JOptionPane.showMessageDialog;  
  
  
**public class** GUI **extends** JFrame {  
 **private** JPanel **rootPanel**;  
 **private** JButton **buttonTrain**;  
 **private** JPanel **imagePanel**;  
 **private** JLabel **imageLabel**;  
 **private** JLabel **labelError**;  
 **private** JLabel **labelAnswer**;  
 **private** JLabel **labelEpoch**;  
 **private** JTextArea **textAreaAnswer**;  
 **private** JButton **buttonTest**;  
 **private** NeuralNet **neuralNet**;  
  
 **public** GUI() {  
 setContentPane(**rootPanel**);  
 pack();  
 setTitle(**"Обучение через дельта-правило"**);  
 setDefaultCloseOperation(JFrame.***EXIT\_ON\_CLOSE***);  
 setLocationRelativeTo(**null**);  
 setVisible(**true**);  
 **buttonTest**.setEnabled(**false**);  
  
 **buttonTrain**.addActionListener(**new** ActionListener() {  
 **public void** actionPerformed(ActionEvent e) {  
 train();  
 }  
 });  
 **buttonTest**.addActionListener(**new** ActionListener() {  
 @Override  
 **public void** actionPerformed(ActionEvent e) {  
 test();  
 }  
 });  
 }  
  
 **public void** train() {  
 **try** {  
 **buttonTest**.setEnabled(**false**);  
 Vector[] trainVectorSet = readTrainVectors(**"Train"**);  
 **neuralNet** = **new** NeuralNet(trainVectorSet[0].getX().**length**, trainVectorSet[0].getDesireOutputs().**length**);  
 **neuralNet**.setComplete(**false**);  
  
 Runnable task1 = () -> { **neuralNet**.train(trainVectorSet); };  
 Thread thread1 = **new** Thread(task1);  
 Runnable task2 = () -> {  
 **while** (!**neuralNet**.isComplete()) {  
 **labelEpoch**.setText(**"Номер эпохи: "** + **neuralNet**.getEpochNumber());  
 **labelError**.setText(**"Ошибки нейронов: "** + Arrays.*toString*(**neuralNet**.getError()));  
 }  
 **textAreaAnswer**.append(**"Обучение завершено\n"**);  
 **buttonTest**.setEnabled(**true**);  
 };  
 Thread thread2 = **new** Thread(task2);  
  
 thread1.start();  
 thread2.start();  
 }  
 **catch** (IOException e) {  
 e.printStackTrace();  
 *showMessageDialog*(**null**, **"Файл не найден"**);  
 }  
 **catch** (Exception e) {  
 e.printStackTrace();  
 *showMessageDialog*(**null**, e.toString());  
  
 }  
  
 **int** f = 0;  
 }  
  
 **public void** test() {  
 **try** {  
 String path = **"Test"**;  
 **for** (**int** i = 0; i < 4; i++) {  
 File[] files = **new** File(path + **"//"** + i).listFiles();  
 **for** (File file : files) {  
 **double**[] testVector = readVector(file.getPath());  
 **double**[] answer = **neuralNet**.test(testVector);  
 **textAreaAnswer**.append(String.*format*(**"Тест-образ №%d = %s;%n"**, i, Arrays.*toString*(answer)));  
 *//textAreaAnswer.setCaretPosition(textAreaAnswer.getDocument().getLength());* }  
 }  
 }  
 **catch** (IOException e) {  
 e.printStackTrace();  
 *showMessageDialog*(**null**, **"Файл не найден"**);  
 }  
 **catch** (Exception e) {  
 e.printStackTrace();  
 *showMessageDialog*(**null**, e.toString());  
  
 }  
 }  
  
 **public double**[] readVector(String path) **throws** IOException {  
 BufferedImage image = ImageIO.*read*(**new** File(path));  
 **int**[][] grayImage = imageToGrayScale(image);  
 **double**[] imageVector = imageToVector(grayImage);  
 **return** imageVector;  
 }  
  
  
 **public** Vector[] readTrainVectors(String rootDir) **throws** IOException {  
 List<Vector> trainVectorSet = **new** ArrayList();  
  
 **for** (**int** i = 0; i < 4; i++) {  
 File[] files = **new** File(rootDir + **"//"** + i).listFiles();  
 **for** (File file : files) {  
 BufferedImage image = ImageIO.*read*(file);  
  
 **int**[][] grayImage = imageToGrayScale(image);  
 **double**[] imageVector = imageToVector(grayImage);  
  
 **double**[] desireOutputs = **new double**[4];  
 **for** (**int** k = 0; k < desireOutputs.**length**; k++) {  
 desireOutputs[k] = i == k ? 1 : 0;  
 }  
  
 trainVectorSet.add(**new** Vector(imageVector, desireOutputs));  
 *//imageLabel.setIcon(new ImageIcon(image));* }  
 }  
 **return** (Vector[])trainVectorSet.toArray(**new** Vector[trainVectorSet.size()]);  
 }  
  
 **public int**[][] imageToGrayScale(BufferedImage image) {  
 **int**[][] resultImage = **new int**[image.getWidth()][image.getHeight()];  
 **for**(**int** x = 0; x < image.getWidth(); x++) {  
 **for** (**int** y = 0; y < image.getHeight(); y++) {  
 Color c = **new** Color(image.getRGB(x, y));  
 resultImage[x][y] = (c.getRed() + c.getGreen() + c.getBlue()) / 3;  
 }  
 }  
 **return** resultImage;  
 }  
  
 **public** BufferedImage grayScaleToImage(**int**[][] grayImage)  
 {  
 **int** height = grayImage[0].**length**;  
 **int** width = grayImage[1].**length**;  
 BufferedImage image = **new** BufferedImage(width, height, BufferedImage.***TYPE\_INT\_RGB***);  
 **for** (**int** x = 0; x < width; x++) {  
 **for** (**int** y = 0; y < height; y++) {  
 Color c = **new** Color(grayImage[x][y], grayImage[x][y], grayImage[x][y], 0);  
 image.setRGB(x, y, c.getRGB());  
 }  
 }  
 **return** image;  
 }  
  
  
 **public double**[] imageToVector(**int**[][] image) {  
 **double**[] resultVector = **new double**[image[0].**length** \* image[1].**length**];  
 **int** i = 0;  
 **for**(**int** x = 0; x < image.**length**; x++) {  
 **for** (**int** y = 0; y < image.**length**; y++) {  
 resultVector[i++] = image[x][y];  
 }  
 }  
 **return** resultVector;  
 }  
  
}