Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ *Робототехники и комплексной автоматизации*

КАФЕДРА *Системы автоматизированного проектирования (РК-6)*

ОТЧЕТ О ВЫПОЛНЕНИИ ЛАБОРАТОРНОЙ РАБОТЫ

по дисциплине: «Введение в искусственный интеллект»

Студент	Никифорова Ирина Андреевна
Группа	РК6-12М
Тип задания	лабораторная работа
Тема лабораторной работы	№3. Искусственный нейрон
Вариант	1-7. Персептрон, набор данных №7

Студент _____ **Никифорова И.А.**
подпись, дата *фамилия, и.о.*

Преподаватель _____ **Федорук В.Г.**
подпись, дата *фамилия, и.о.*

Оценка _____

Москва, 2021 г.

Оглавление

Задание на лабораторную работу

1. Лабораторная работа выполняется в среде ОС Linux с использованием компилятора gcc/g++ языка программирования C/C++. Для создания графических иллюстраций рекомендуется использовать утилиту gnuplot.
2. Разработать, используя язык C/C++, программу, моделирующую поведение искусственного трехвходового нейрона указанного преподавателем типа и обеспечивающую его обучение для решения задачи классификации. Тип нейрона 1 - персептрон.
3. Отладить модель нейрона и процедуру его обучения на произвольных двумерных данных. Рекомендуется, в тех ситуациях, когда это возможно, использовать режим обучения "оффлайн".
4. Обучить разработанный нейрон на предложенном преподавателем варианте двухмерных данных (рис. 1) и проверить его работу на ряде контрольных точек.

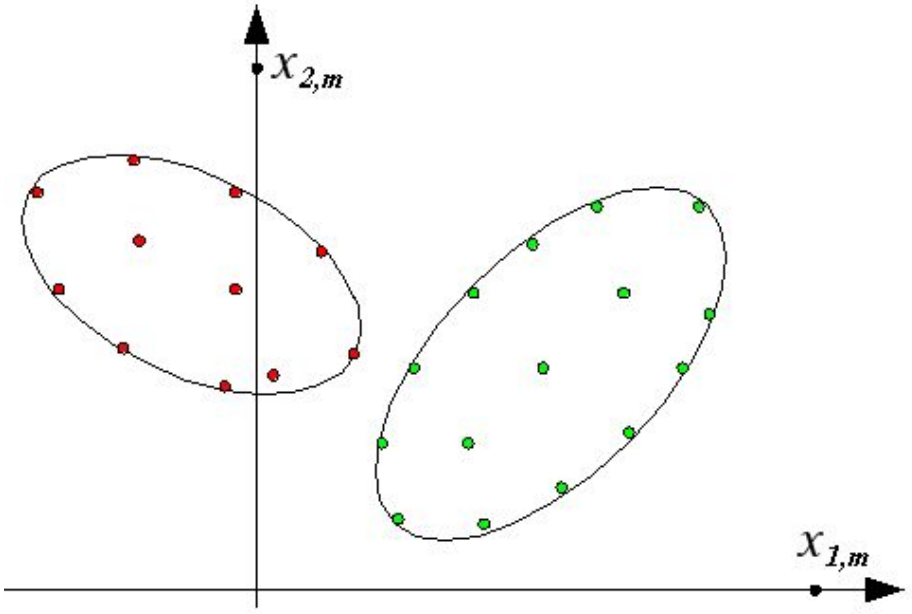

Рис. 1. Обучающие данные. Вариант №7

Цель выполнения лабораторной работы

Цель выполнения лабораторной работы – создание программы, реализующей искусственный нейрон; разработка процедуры обучения нейрона; использование полученных результатов для решения тестовых задач классификации и аппроксимации.

Содержание отчета

В отчете представлены:

1. Описание реализованной модели нейрона и процедуры его обучения.
2. Рисунок, иллюстрирующий распределение в пространстве $[x_1,x_2]^T$ обучающих данных.
3. Численные значения, характеризующие начальное состояние, ход обучения и его результат (например, начальные и итоговые значения входных весов нейрона, величина коэффициента обучения, количество циклов обучения и т.п.).
4. Графическое представление результатов обучения нейрона (например, график зависимости выходного сигнала нейрона от входных данных $[x_1,x_2]^T$).
5. Исходный текст программы.

Описание реализованной модели нейрона

В работе был реализован искусственный нейрон типа персептрон, который называется также моделью МакКаллока-Питтса. Такой нейрон имеет несколько входов, включая поляризатор. Его структурная схема представлена на рис. 2.

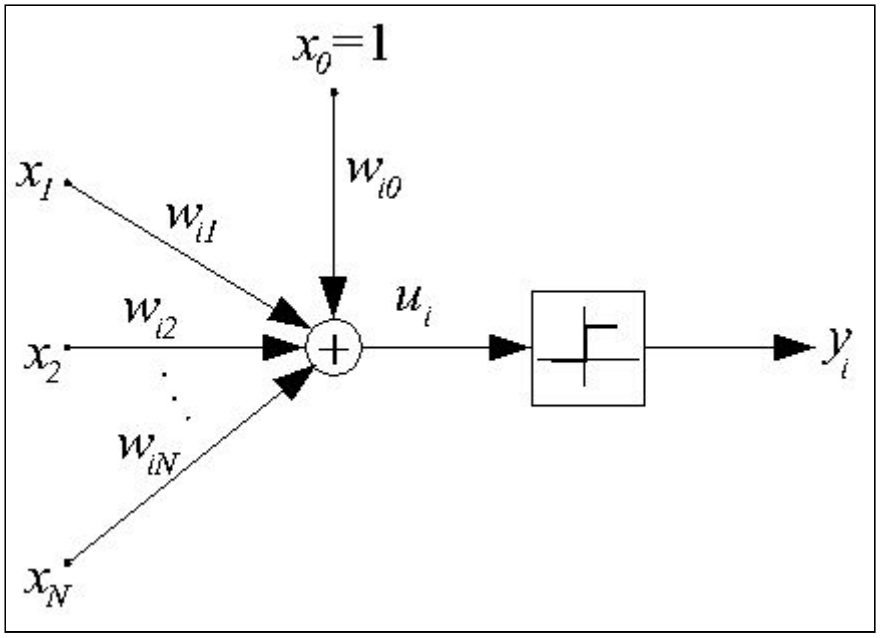

Рис. 2. Структурная модель персептрона. Обозначения: i - порядковый номер нейрона; x_0 - поляризационный сигнал; x_k, $k = 1, 2, \ldots, N$ - входные сигналы; w_{ij}, $j = 0, 1, \ldots, N$ - веса входных сигналов; u_i - взвешенная сумма входных сигналов; ⌐┘ - обозначение функции активации; y_i - выходной сигнал

В качестве функции активации для него выступает следующая функция:

$$y_i = f(u_i) = \begin{cases} 1, u_i \geq 0 \\ 0, u_i < 0 \end{cases}$$

В программе нейрон был представлен в виде класса, в закрытых полях которого хранится только количество входов (исключая поляризатор) и их веса (включая вес поляризационного входа).

Алгоритм обучения нейрона

Обучение нейрона возможно только с учителем, т.е. с представленной выборкой, состоящей из набора пар $<\boldsymbol{X_i^k}, d_i^k>$, $k = 1, 2, \ldots, p$, где p - количество пар обучающей выборки, $\boldsymbol{X_i}$ - вектор входных сигналов, d_i - соответствующий этому вектору эталонный выходной сигнал.

Целевой функцией, которую необходимо минимизировать при обучении - отыскании весовых коэффициентов нейрона - выступает следующая функция:

$$E(\mathbf{W}_i) = \frac{1}{2}\sum_{k=1}^{p}(y_i^k - d_i^k)^2,$$

где $\boldsymbol{W_i}$ - вектор весов входных сигналов, p - количество пар обучающей выборки, d_i^k- эталонный выходной сигнал для k-той пары, y_i^k- реальный выходной сигнал для k-той пары.

При минимизации данной функции происходит, фактически, минимизация отклонения реального сигнала от эталонного для всех пар в обучающей выборке.

Функция активации персептрона разрывна, поэтому обучение обычными методами оптимизации, использующими дифференцирование (старше нулевого порядка), к нему неприменимо. Тем не менее, обучение можно произвести более простым методом - используя правило персептрона (частный случай правила Видроу-Хоффа).

Правило персептрона представляет из себя следующий алгоритм:

1. Выбираются (как правило, случайно) начальные значения весов w_{ij}, $j = 0, 1, \ldots, N$ нейрона.
2. Для каждой обучающей пары $<\boldsymbol{X^k}, d_i^k>$ выполняется ряд циклов (их номера обозначим через t) уточнения значений входных весов по формуле

$$w_{ij}(t+1) = w_{ij}(t) + \delta w_{ij}(t),$$

$$\delta w_{ij}(t) = \begin{cases} 0, y_i(t) = d_i^k; \\ x_j^k, y_i(t) = 0, d_i^k = 1; \\ -x_j^k, y_i(t) = 1, d_i^k = 0. \end{cases}$$

3. Процесс обработки текущей обучающей пары завершается либо на цикле, в котором все приращения весов равны нулю, либо после достижения предельного количества циклов.

Программная реализация

Программная реализация нейрона, его обучения и использования была выполнена на языке C++.

Реализация разбита на несколько файлов (листинг 1). Полное содержание каждого файла расположено в приложении А.

Листинг 1. Файлы исходных данных программы

```
.
├── compile.sh
├── main.cpp
├── neuron.cpp
├── neuron.hpp
└── plot_results.gnu
```

В файлах *neuron.hpp* и *neuron.cpp* находятся объявление и реализация класса персептрона соответственно. В файле *main.cpp* находятся считывание и подготовка обучающих данных, обучение нейрона, проверка обученного нейрона на тестовых данных. Файл *plot_results.gnu* содержит скрипт для утилиты *gnuplot*, который создает изображение тестовой выборки и линии ее разделения, которую получает нейрон. Также был написан shell-скрипт *compile.sh*, который позволяет правильно скомпилировать программу.

Основной частью программы является реализация класса нейрона. Класс нейрона хранит текущие веса своих входов и количество этих входов, исключая поляризатор. Методы нейрона включают (приложение А, листинг А.1):

1. конструктор по количеству входов;
2. функцию активации;
3. получение взвешенной суммы входных сигналов;
4. непосредственно обучение;
5. целевую функцию (только для проверки, не используется в обучении).

Процесс обучения и тестирования

Для обучения нейрона было использовано правило персептрона. Обучение производилось на точках, представленных на рис. 1. Координаты для точек были найдены посредством наложения сетки с шагом 0.1 (рис. 3).

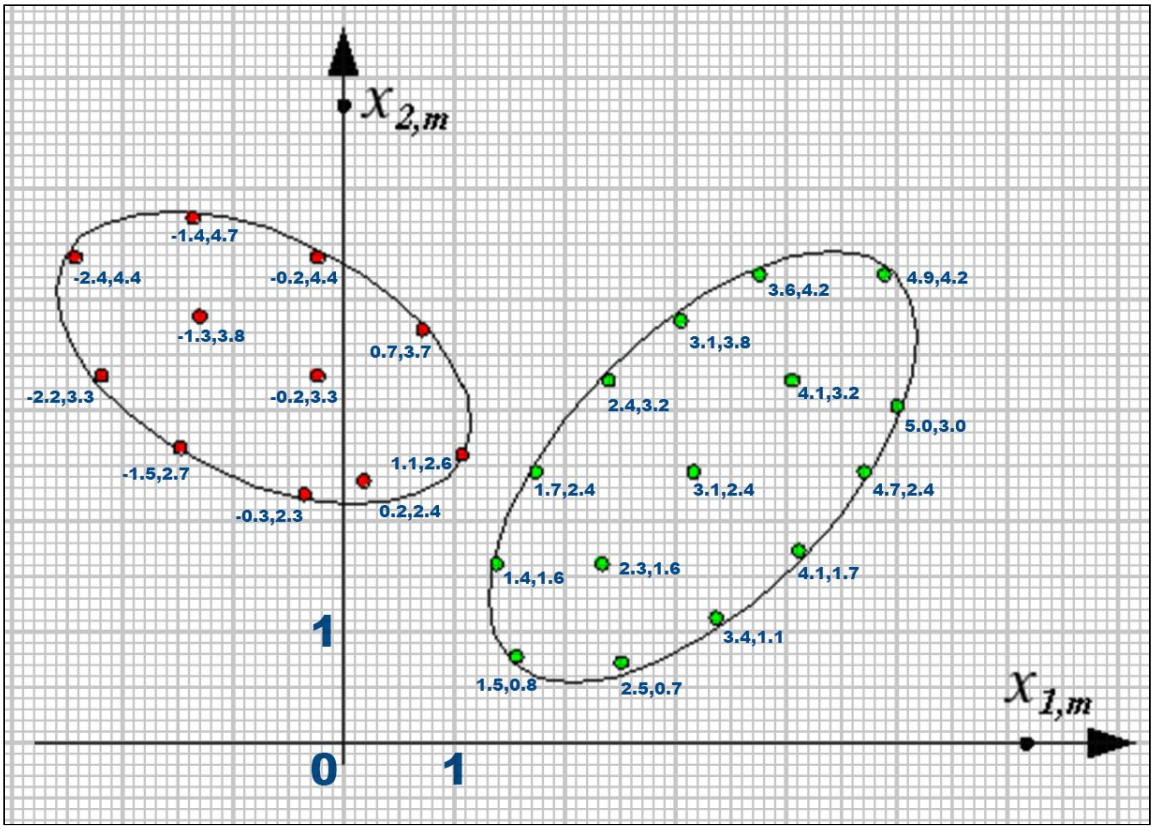

Рис. 3. Получение координат для обучающей выборки

Далее полученные данные были представлены в текстовой форме и размечены, т.к. обучение должно протекать с учителем. Красные точки были отнесены к классу 0, а зеленые - к классу 1. В результате был получен следующий файл исходных данных *learn_data.dat* (листинг 2), где первые две колонки отвечают за координаты, а третья - за номер класса (столбцы разделены пробелами).

Листинг 2. Файл обучающих данных

-1.4 4.7 0	-0.3 2.3 0	1.7 2.4 1	1.5 0.8 1

-2.4 4.4 0 -0.2 4.4 0 -1.3 3.8 0 0.7 3.7 0 -2.2 3.3 0 -1.5 2.7 0	0.2 2.4 0 1.1 2.6 0 -0.2 3.3 0 1.4 1.6 1 2.3 1.6 1 4.1 1.7 1	3.1 2.4 1 4.7 2.4 1 5.0 3.0 1 4.9 4.2 1 4.1 3.2 1	2.5 0.7 1 3.4 1.1 1 3.6 4.2 1 3.1 3.8 1 2.4 3.2 1

Данные в программе считываются с помощью простого *CSV*-парсера, реализованного в рамках предыдущих работ. После считывания из данных формируются следующие вектора, которые передаются в функцию обучения нейрона (приложение А, листинг А.2):

1. *vector< vector<double> > X_learn* - вектор входных векторов нейрона;
2. *vector<bool> D* - вектор соответствующих им эталонных выходных сигналов нейрона.

Кроме обучения для нейрона необходимо проводить тестирование. Оно позволит узнать, как нейрон будет отвечать на данные, которые он ранее не встречал. Для этого аналогичным образом был создан, считан и использован файл тестовых данных (листинг 3).

Листинг 3. Файл тестовых данных

-1 3 0 -2 4 0 -1.2 3.2 0	2 1 1 3 3 1 4.2 2.2 1

Для тестовых данных в нейрон были отправлены только значения координат. Далее, полученный от нейрона (от функции *activation_func*) ответ сравнивался с эталонным ответом из файла. После этого информация о том, совпали ответы или нет, была выведена в консоль.

Результаты работы программы

Процесс обучения нейрона зависит от случайных начальных весов, которые генерируются в пределах [0;10) с двумя знаками после запятой. Поэтому была проведена проверка для разных начальных значений. Результаты работы программы представлены в листингах 4 - 6, где *N* - номер итерации изменения весов для одной обучающей пары. Веса выводятся только при изменении, функция активации *Aim func* выводится только для справки.

Листинг 4. Результат запуска программы без использования *srand*

```
Learning:
Start W: 7.77 3.83 8.86

N = 0. 6.77 5.23 4.16
Aim func = 5.5

N = 1. 5.77 6.63 -0.54
Aim func = 3

N = 0. 4.77 6.83 -4.94
Aim func = 0

Testing:
Neuron res = 0, D = 0. Test passed!
Neuron res = 0, D = 0. Test passed!
Neuron res = 0, D = 0. Test passed!
Neuron res = 1, D = 1. Test passed!
Neuron res = 1, D = 1. Test passed!
Neuron res = 1, D = 1. Test passed!
```

Листинг 5. Результат запуска программы с использованием *srand* (1)

```
Learning:
```

```
Start W: 8.83 1.69 8.66

N = 0. 7.83 3.09 3.96
Aim func = 5.5

N = 1. 6.83 4.49 -0.74
Aim func = 3

N = 0. 5.83 4.69 -5.14
Aim func = 0

Testing:
Neuron res = 0, D = 0. Test passed!
Neuron res = 0, D = 0. Test passed!
Neuron res = 0, D = 0. Test passed!
Neuron res = 1, D = 1. Test passed!
Neuron res = 1, D = 1. Test passed!
Neuron res = 1, D = 1. Test passed!
```

Листинг 6. Результат запуска программы с использованием *srand* (2)

```
Learning:
Start W: 5.06 6.34 8.5

N = 0. 4.06 7.74 3.8
Aim func = 5

N = 1. 3.06 9.14 -0.9
Aim func = 1.5

N = 0. 2.06 8.44 -4.6
Aim func = 0
```

```
Testing:
Neuron res = 0, D = 0. Test passed!
Neuron res = 0, D = 0. Test passed!
Neuron res = 0, D = 0. Test passed!
Neuron res = 1, D = 1. Test passed!
Neuron res = 1, D = 1. Test passed!
Neuron res = 1, D = 1. Test passed!
```

Нейрон типа персептрон разделяет данные с помощью гиперплоскости. Иначе говоря, если представить двумерные данные на плоскости, то нейрон будет отвечать за линию их разделения по двум классам. Благодаря этой особенности можно посмотреть как именно нейрон делит данные. Для этого нужно визуализировать линию раздела, построенную по весам нейрона, на той же плоскости, что обучающие и тестовые данные. Для этого был написан скрипт для утилиты *gnuplot* под названием *plot_results.gnu* (листинг А.4). Для результирующих весов (они вносятся в скрипт вручную) каждого из описанных запусков (листинг 4 - 6) были получены изображения (рис. 4 - 6 соответсвенно). На изображениях черным цветом обозначены точки, принадлежащие к классу 0, красным - к классу 1. Круглые маркеры обозначают обучающую выборку, маркеры-звездочки - тестовую. Линией обозначена линия пересечения гиперплоскости, которую создает нейрон, с координатной плоскостью Ox_1x_2.

Для дополнительного сравнения был также создан рисунок 7, где все три линии, соответствующие разным вариантам весов нейрона, представлены совместно. На рисунке 7 линиями представлены варианты линий раздела нейрона, черным цветом обозначены точки, принадлежащие к классу 0, красным - к классу 1. Незакрашенные маркеры обозначают обучающую выборку, закрашенные - тестовую.

Как видно из рисунков 4 - 7, все полученные варианты весов нейрона являются корректными и поставленная задача была выполнена.

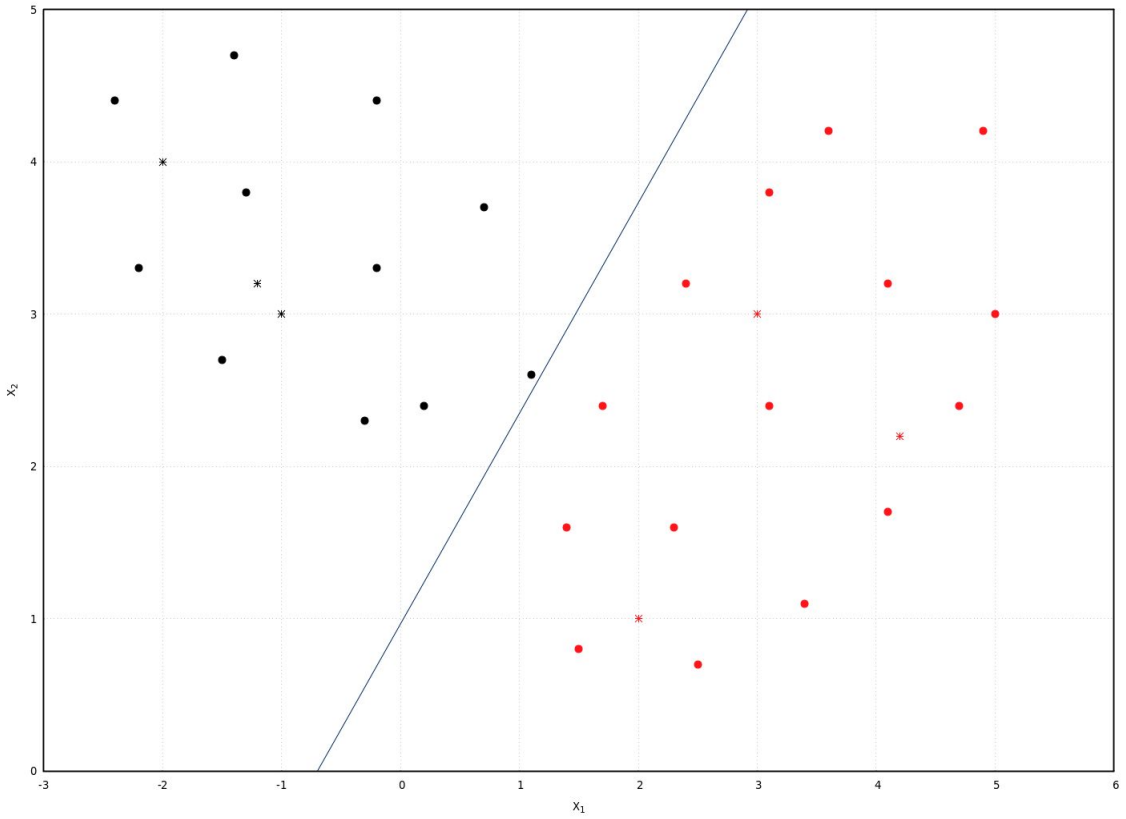

Рис. 4. Визуализация для весов нейрона $w_0 = 4.77$, $w_1 = 6.83$, $w_2 = -4.94$

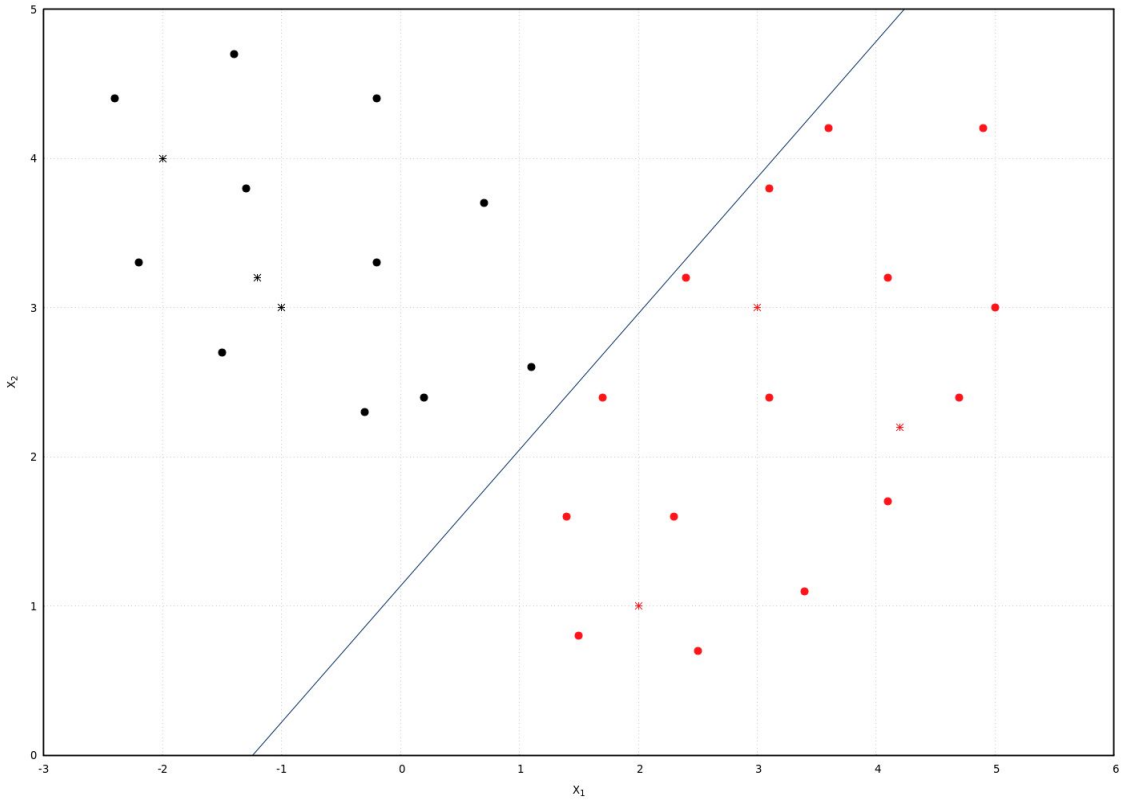

Рис. 5. Визуализация для весов нейрона $w_0 = 5.83$, $w_1 = 4.69$, $w_2 = -5.14$

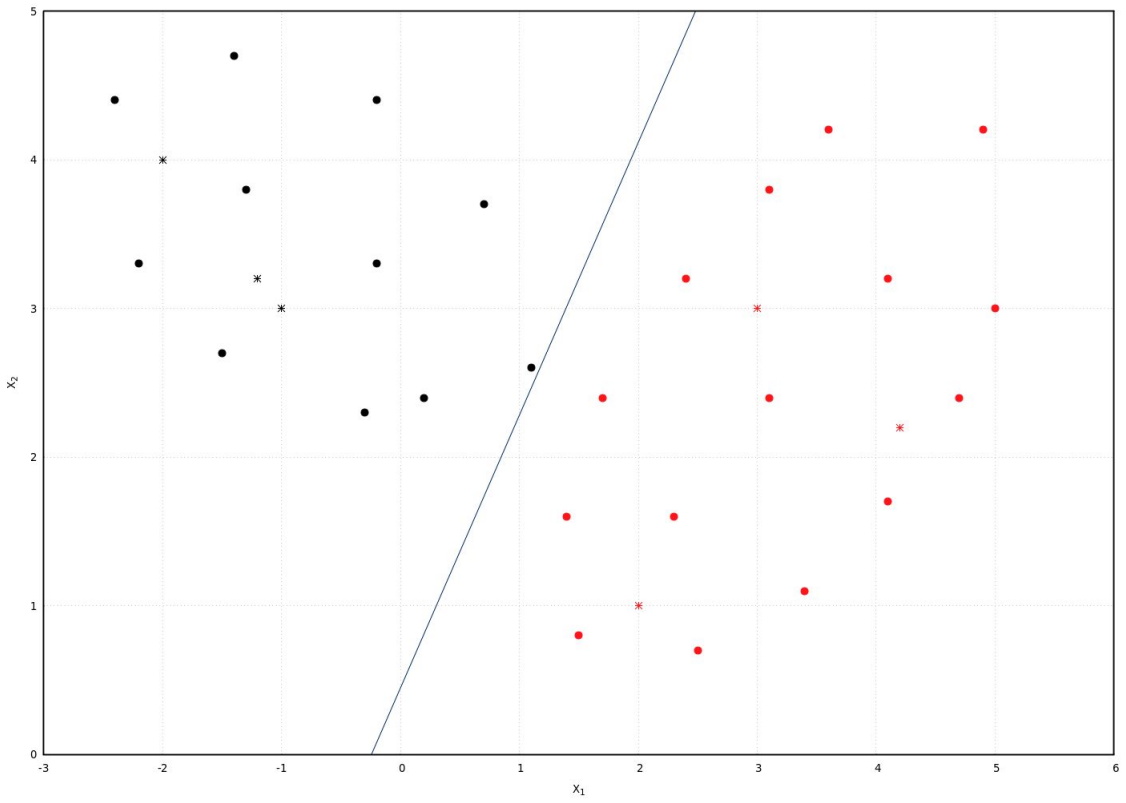

Рис. 6. Визуализация для весов нейрона $w_0 = 2.06$, $w_1 = 8.44$, $w_2 = -4.6$

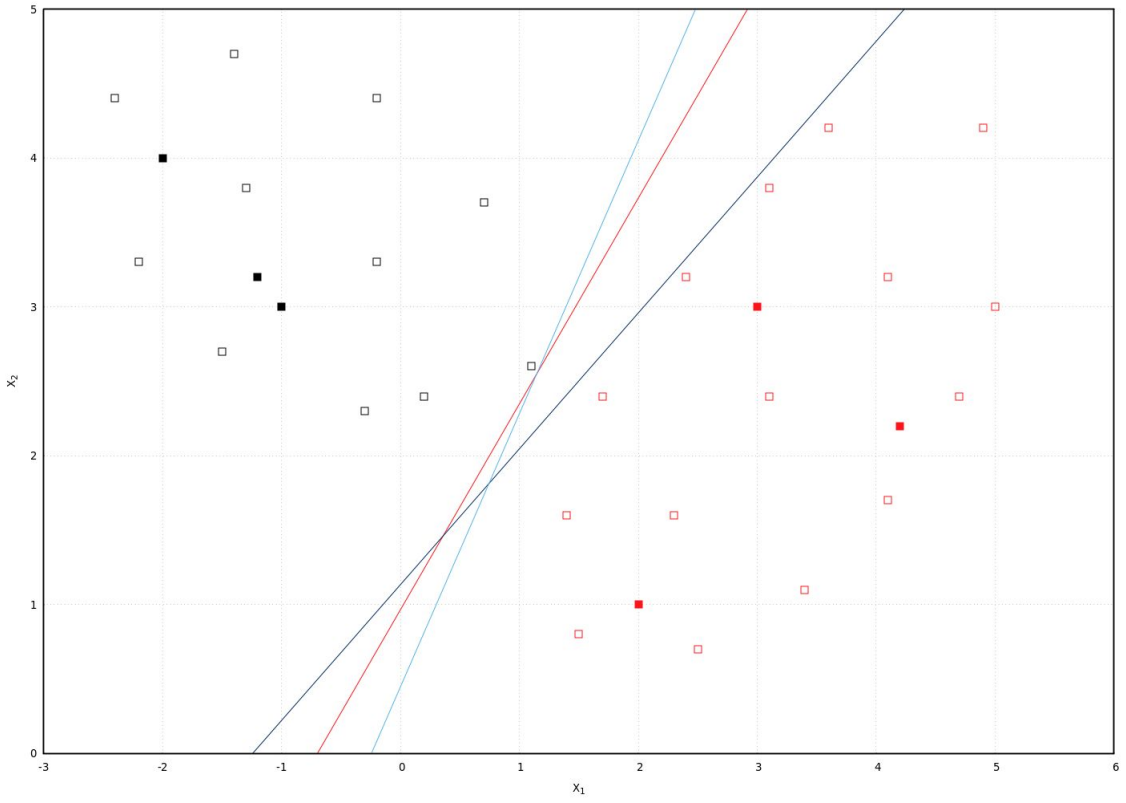

Рис. 7. Совместная визуализация различных весов нейрона

Источники

1. Федорук, В. Г. Персептрон / В. Г. Федорук. — Текст : электронный // БиГОР : [сайт]. — URL: http://bigor.bmstu.ru/?cnt/?doc=NN/011-neurons.mod/?cou=NN/base.cou.
2. Федорук, В. Г. Методические указания по проведению лабораторной работы N1 "Программирование искусственного нейрона" по курсу "Искусственные нейронные сети" / В. Г. Федорук. — Текст : электронный // fedoruk.comcor.ru : [сайт]. — URL: http://fedoruk.comcor.ru/AI_mag/NNlab/lab1.html.

Приложение А. Текст программы

Листинг А.1. Файл *neuron.hpp*

```
#pragma once
#include <vector>
#include <iostream>

using namespace std;

class Perceptron {
private:
   int inputs_num;
   vector<double> W;
public:
   Perceptron(int _inputs_num);
   bool activation_func(vector<double>& X);
   double u_summ(vector<double>& X);
   void learn(vector< vector<double> >& X_learn, vector<bool>& D, int
N_max);
   double aim_func(vector<bool>& D, vector<bool>& Y);
};
```

Листинг А.2. Файл *neuron.cpp*

```
#include "neuron.hpp"
#include <ctime>

// конструктор нейрона по _inputs_num - количеству входов
Perceptron::Perceptron(int _inputs_num) {
   inputs_num = _inputs_num;
   W.resize(inputs_num + 1);
}

// функция активации, X - вектор входных сигналов размера inputs_num
bool Perceptron::activation_func(vector<double>& X) {
   if (X.size() != inputs_num) {
       cout << "ERROR: X.size() != inputs_num" << endl;
       return 0;
   }

   if (u_summ(X) >= 0) {
       return true;
   } else {
```

```
        return false;
    }
}

// сумма сигналов * на веса u, X - вектор входных сигналов размера
inputs_num
double Perceptron::u_summ(vector<double>& X) {
    if (X.size() != inputs_num) {
        cout << "ERROR: X.size() != inputs_num" << endl;
        return 0;
    }

    double u = 0;

    for (size_t i = 0; i < inputs_num; i++) {
        u += X[i] * W[i];
    }
    u += W[inputs_num]; // поляризатор

    return u;
}

// обучение, X_learn - вектор входных сигналов размера обучающей выборки
(каждый подвектор размера inputs_num),
// D - вектор эталонных выходных сигналов размера обучающей выборки, N_max
- предельное число циклов обучения
void Perceptron::learn(vector< vector<double> >& X_learn, vector<bool>& D,
int N_max) {
    if (D.size() != X_learn.size()) {
        cout << "ERROR: D.size() != X_learn.size()" << endl;
        return;
    }

    srand(time(NULL));
    for (size_t i = 0; i < inputs_num + 1; i++) {
        W[i] = rand() % 1000;
        W[i] /= 100;
    }

    cout << "Start W: ";
    cout << W[inputs_num] << " ";
    for (size_t i = 0; i < inputs_num; i++) {
        cout << W[i] << " ";
    }
```

```
    cout << endl << endl;

    // для подсчета функции активации
    vector<bool> Y(D.size());

    for (size_t k = 0; k < X_learn.size(); k++) {
        bool stop = false;

        for (size_t i = 0; i < N_max && !stop; i++) {
            bool y = activation_func(X_learn[k]);

            if (y != D[k]) {
                if (y == false) {
                    for (size_t j = 0; j < inputs_num; j++) {
                        W[j] += X_learn[k][j];
                    }
                    W[inputs_num] += 1;
                } else {
                    for (size_t j = 0; j < inputs_num; j++) {
                        W[j] -= X_learn[k][j];
                    }
                    W[inputs_num] -= 1;
                }

                cout << "N = " << i << ". ";
                cout << W[inputs_num] << " ";
                for (size_t i = 0; i < inputs_num; i++) {
                    cout << W[i] << " ";
                }
                cout << endl;

                // Подсчет функции активации для наблюдения
                // за ее изменением
                for (size_t i = 0; i < D.size(); i++) {
                    Y[i] = activation_func(X_learn[i]);
                }
                cout << "Aim func = " << aim_func(D, Y);
                cout << endl << endl;
            } else {
                stop = true;
            }
        }
    }
```

```
    return;
}

// целевая функция, D - вектор эталонных выходных сигналов размера
обучающей выборки,
// Y - вектор реальных выходных сигналов размера обучающей выборки
double Perceptron::aim_func(vector<bool>& D, vector<bool>& Y) {
    if (D.size() != Y.size()) {
        cout << "ERROR: D.size() != Y.size()" << endl;
        return 0;
    }

    double E = 0;

    for (size_t k = 0; k < D.size(); k++) {
        E += (Y[k] - D[k]) * (Y[k] - D[k]);
    }

    return 0.5 * E;
}
```

Листинг А.3. Файл *main.cpp*

```
#include <iostream>
#include <vector>
#include <string>
#include <fstream>
#include "neuron.hpp"

using namespace std;

// Убирает пробелы с концов переданной строки
string trim_spaces(string line) {
    string trimmed;
    string space_like_characters = " \t\n";

    size_t first_non_space = line.find_first_not_of(space_like_characters);
    size_t last_non_space = line.find_last_not_of(space_like_characters);

    trimmed = line.substr(first_non_space, last_non_space - first_non_space
+ 1);

    return trimmed;
```

```
}

// Разделяет строку на массив строк по переданному разделителю, сокращая
пробелы на концах разделенных строк
vector<double> csv_parser(string line, string delimiter) {
   vector<double> splited_line;

   size_t prev_pos = 0, delimiter_pos = line.find(delimiter);
   string sub_line;
   while (delimiter_pos != string::npos) {
       sub_line = line.substr(prev_pos, delimiter_pos - prev_pos);
       splited_line.push_back(stod(trim_spaces(sub_line)));

       prev_pos = delimiter_pos + delimiter.size();
       delimiter_pos = line.find(delimiter, prev_pos);
   }
   splited_line.push_back(stod(trim_spaces(line.substr(prev_pos))));

   return splited_line;
}

int main() {
   // чтение из файла обучающей выборки
   ifstream learn_data_file;
   learn_data_file.open("learn_data.dat");
   if (!learn_data_file) {
       cout << "ERROR: can't open file 'learn_data.dat'" << endl;
       return 1;
   }

   vector<bool> D_learn;
   vector< vector<double> > X_learn;
   vector<double> x_local(2);
   string line;
   while(getline(learn_data_file, line)) {
       vector<double> values = csv_parser(line, " ");
       x_local[0] = values[0];
       x_local[1] = values[1];
       X_learn.push_back(x_local);
       D_learn.push_back(values[2]);
   }

   learn_data_file.close();
```

```
    cout << "Learning:" << endl;
    // создание и обучение персептрона
    Perceptron P(2);
    P.learn(X_learn, D_learn, 100);

    // чтение из файла тестовой выборки
    ifstream test_data_file;
    test_data_file.open("test_data.dat");
    if (!test_data_file) {
        cout << "ERROR: can't open file 'test_data.dat'" << endl;
        return 1;
    }

    vector<bool> D_test;
    vector< vector<double> > X_test;
    while(getline(test_data_file, line)) {
        vector<double> values = csv_parser(line, " ");
        x_local[0] = values[0];
        x_local[1] = values[1];
        X_test.push_back(x_local);
        D_test.push_back(values[2]);
    }

    cout << "Testing:" << endl;
    // тестирование нейрона
    for (size_t i = 0; i < D_test.size(); i++) {
        bool result = P.activation_func(X_test[i]);
        cout << "Neuron res = " << result << ", ";
        cout << "D = " << D_test[i] << ". ";
        if (result == D_test[i]) {
            cout << "Test passed!";
        } else {
            cout << "Test not passed!";
        }
        cout << endl;
    }


    return 0;
}
```

Листинг А.4. Файл *plot_results.gnu*

```
set terminal pngcairo size 1418,1017 enhanced font 'Verdana,10'
set output 'results.png'

# Styling
set border linewidth 1.5
set pointsize 1.5
set style line 1 lc rgb '#FD151B' pt 7   # circle

set border linewidth 1.5
set pointsize 1.5
set style line 2 lc rgb '#01295F' pt 7   # circle

unset key

set tics scale 0.1
set xtics 1
set ytics 1
set yrange[0:5]
set xrange[-3:6]
set xlabel 'X_1'
set ylabel 'X_2'


set grid

# эти данные заменяются вручную из результатов программы
w_00 = 4.77
w_01 = 6.83
w_02 = -4.94

w_10 = 5.83
w_11 = 4.69
w_12 = -5.14

w_20 = 2.06
w_21 = 8.44
w_22 = -4.6

f0(x) = -1/(w_02) * (w_01 * x + w_00)
f1(x) = -1/(w_12) * (w_11 * x + w_10)
f2(x) = -1/(w_22) * (w_21 * x + w_20)
```

```
set style increment user
plot f0(x) w l, f1(x) w l, f2(x) w l, 'learn_data.dat' using 1:2:3 w p lc
var, 'test_data.dat' using 1:2:3 w p lc var
```

Листинг А.5. Файл *compile.sh*

```
c++ main.cpp neuron.cpp
```