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 «Информатика и системы управления» (ИУ)

КАФЕДРА «Информационная безопасность» (ИУ8)

Отчёт

по лабораторной работе № 5
по дисциплине «Теория систем и системный анализ»

Тема: «Двумерный поиск для подбора коэффициентов простейшей нейронной сети на примере решения задачи линейной регрессии экспериментальных данных»

Вариант 15

Выполнил: Бакаев Ф. Б.,
студент группы ИУ8-31

Проверил: Коннова Н.С.,
доцент каф. ИУ8

Цель работы

Знакомство с простейшей нейронной сетью и реализация алгоритма поиска ее весовых коэффициентов на примере решения задачи регрессии экспериментальных данных.

Условие задачи

В зависимости от варианта работы найти линейную регрессию функции $y(x)$ (коэффициенты наиболее подходящей прямой c, d) по набору ее N дискретных значений, заданных равномерно на интервале $[a,b]$ со случайными ошибками $e_i \square A\,\mathrm{rnd}(\square 0.5; 0.5)$.

Выполнить расчет параметров c, d градиентным методом. Провести двумерный пассивный поиск оптимальных весовых коэффициентов нейронной сети (НС) регрессии.

c = 0	a = -4	N = 24
d = 3	b = 2	A = 0.1

График заданной функциии

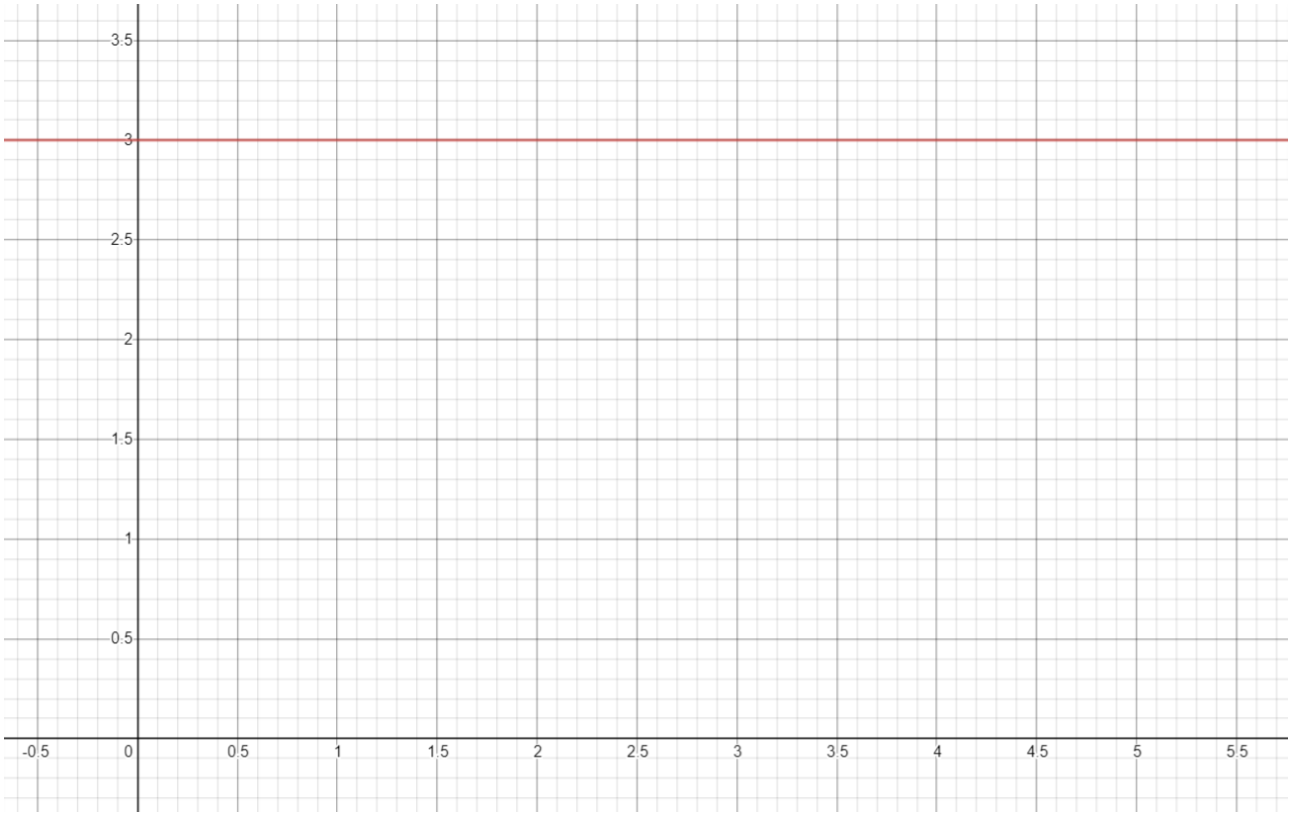

Рисунок 1 - График функции y = 0 * x + 3

Результат работы программы

c = -0.0024
d = 2.96133

Network:

x	y
-3.76	2.95013
-3.52	3.00636
-3.28	2.96933
-3.04	3.03087
-2.8	3.0085
-2.56	2.99799
-2.32	2.98503
-2.08	3.0396
-1.84	3.03228
-1.6	3.02466
-1.36	2.96741
-1.12	3.03589
-0.88	3.02105
-0.64	3.00135
-0.4	2.9804
-0.16	2.9515
0.08	2.95914
0.32	2.98645
0.56	2.96473
0.8	2.96659
1.04	3.04885
1.28	2.99457
1.52	2.96191
1.76	2.95047

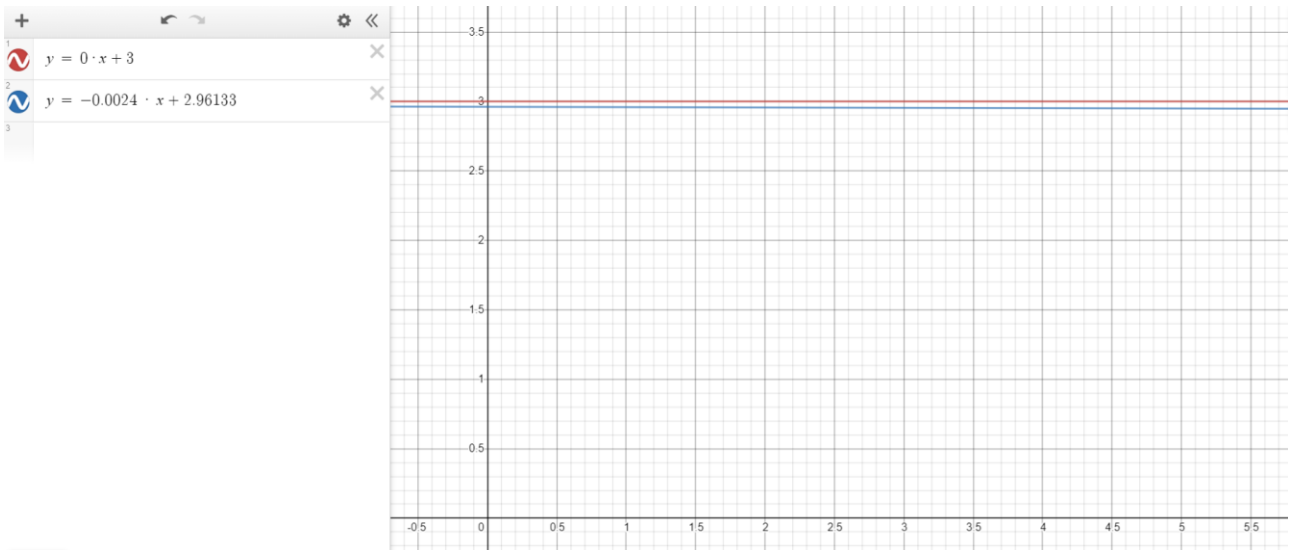

Рисунок 2 - Графики функциий y = 0 * x + 3 и y = -0.0024 * x + 2.96133

Выводы

Реализовал простейшую нейронную сеть и научился использовать метод наименьших квадратов в условиях нахождения весовых коэффициентов нейронной сети. С помощью данной нейронной сети можно показать линейную зависимость между некими переменными.

Приложение. Исходный код программы

```
#include <iostream>
#include <vector>
#include <cmath>
#include <algorithm>
#include <iomanip>

using std::cout;
using std::endl;

class neuron {
private:
    double x;
    double y;

public:
    neuron() : x(0), y(0) {};

    void set_x(double buf_x) {
        x = buf_x;
    }

    void set_y(double buf_y) {
        y = buf_y;
    }

    auto get_x() const noexcept -> double {
        return x;
    }

    auto get_y() const noexcept -> double {
        return y;
    }
};

auto random(double min, double max) -> double {
    return (double)(rand()) / RAND_MAX * (max - min) + min;
}

auto linear_function(double c, double d, double x) -> double {
    return  c * x + d;
}

auto random_error(double a, double b, double A) -> double {
    return A * random(a, b);
}

auto square_func(const std::vector<neuron>& neurons, double c, double d, double N) -> double {
    double sum = 0;
    for (auto i = 0; i < N; i++) {
        auto x = neurons[i].get_x();
        auto y = neurons[i].get_y();
        double new_y = linear_function(c, d, x);
        sum += pow(new_y - y, 2);
    }
    return sum;
}
```

```
auto fill_network(double c, double d, double A, double a, double b, size_t neurons_number) -> std::vector<neuron> {
    auto neuron_distance = (b - a) / static_cast<double>(neurons_number + 1);
    std::vector<neuron> neurons(neurons_number);
    auto current_x = a;
    for (auto& neuron : neurons) {
        current_x += neuron_distance;
        neuron.set_x(current_x);
        neuron.set_y(linear_function(c, d, neuron.get_x()) + random_error(-0.5, 0.5, A));
    }
    return neurons;
}

auto passive_search(const std::vector<neuron>& neurons, double a, double b, double N) -> double {
    auto min_d = neurons[0].get_y();
    auto max_d = neurons[0].get_y();
    for (const auto& neuron : neurons) {
        if (neuron.get_y() < min_d) {
            min_d = neuron.get_y();
        }
        if (neuron.get_y() > max_d) {
            max_d = neuron.get_y();
        }
    }
    auto d = random(min_d, max_d);

    auto max_c = 1.26;
    auto min_c = -1.37;
    std::vector<double> vec_c;
    const int number_of_iterations = 99;
    std::vector<double> sum;
    for (size_t k = 0; k < number_of_iterations; k++) {
        vec_c.push_back(min_c + ((max_c - min_c) / static_cast<double>(number_of_iterations + 1)) * (k + 1));
        sum.push_back(square_func(neurons, vec_c[k], d, N));
    }

    auto min_sum = std::min_element(sum.begin(), sum.end());
    auto num = std::distance(sum.begin(), min_sum);
    return vec_c[num];
}

auto dihotomia(const std::vector<neuron>& neurons, double a, double b, double c, double N) -> double {
    const double eps = 0.1;
    const double delta = 0.01;
    do {
        auto d_left = 0.5 * (b + a) - delta;
        auto d_right = 0.5 * (b + a) + delta;
        auto f_left = square_func(neurons, c, d_left, N);
        auto f_right = square_func(neurons, c, d_right, N);
        if (f_right > f_left) {
            b = d_right;
        } else {
            a = d_left;
        }
    } while ((b - a) > eps);
    return ((b + a) / 2);
}

void print(const std::vector<neuron>& neurons) {
    cout << "--------------------" << endl;
    cout << "| " << std::setw(5) << std::left << "x" << " | " << std::setw(8) << std::left << "y" << "|" << endl;
    cout << "--------------------" << endl;
    for (const auto& neuron : neurons) {
        cout << "| " << std::setw(5) << std::left << neuron.get_x() << " | " << std::setw(8) << std::left << neuron.get_y() << "|"
<< endl;
    }
    cout << "--------------------" << endl;
}

int main() {
```

```
    const double a = -4;
    const double b = 2;
    const double c = 0;
    const double d = 3;
    const size_t N = 24;
    const double A = 0.1;

    auto network = fill_network(c, d, A, a, b, N);

    auto c_find = passive_search(network, a, b, N);

    cout << "c = " << c_find << endl;
    cout << "d = " << dihotomia(network, a, 4, c_find, N) << endl;

    cout << endl << "Network: " << endl;

    print(network);

    return 0;
}
```

Ответ на контрольный вопрос

1. Поясните суть метода наименьших квадратов.

Задача заключается в нахождении коэффициентов линейной зависимости, при которых функция двух переменных a и b:

$$F(a,b) = \sum_{i=1}^{n} (y_i - (a\,x_i + b)^2)$$

принимает наименьшее значение. То есть, при данных a и b сумма квадратов отклонений экспериментальных данных от найденной прямой будет наименьшей. В этом вся суть метода наименьших квадратов.

Таким образом, решение примера сводится к нахождению экстремума функции двух переменных.