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 4

Выполнила: Бояркина Е.Р.,
студент группы ИУ8-31

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

г. Москва, 2020 г.

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 (табл. 1) 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 = Arnd(-0.5, 0.5). Выполнить расчет параметров *c*, *d* градиентным методом. Провести двумерный пассивный поиск оптимальных весовых коэффициентов нейронной сети (НС) регрессии.

Условие варианта: *c = -0.5; d = 0; a = -2; b = 2; N = 16; A = 2*; алгоритм поиска *c* – пассивный, алгоритм поиска *d* – случайный.

Графики

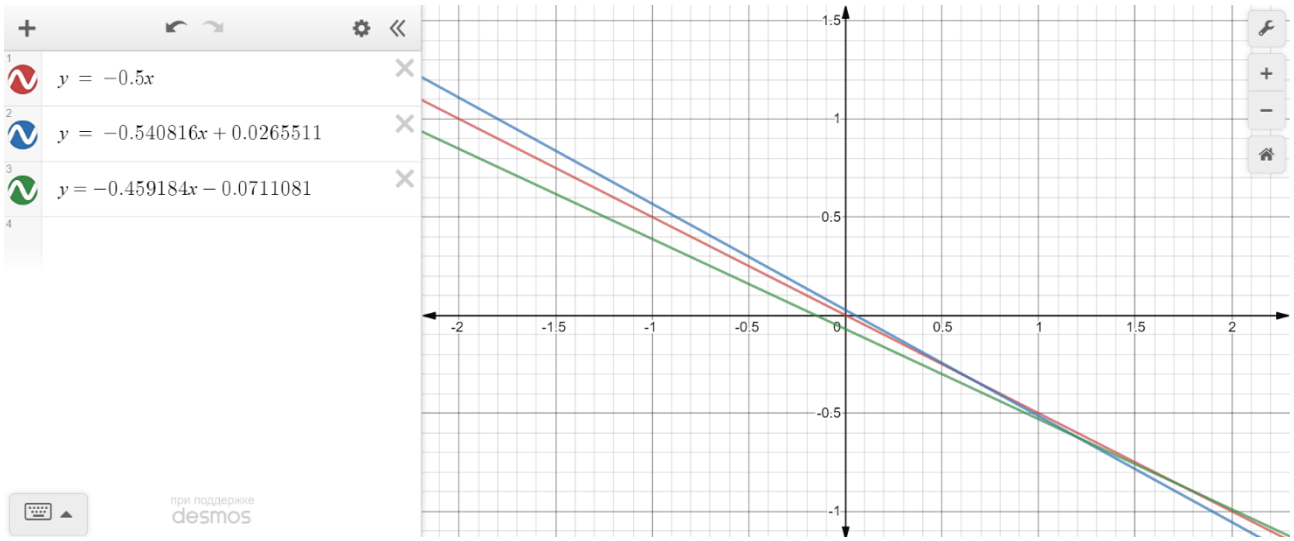

Рисунок 1 – Графики, построенные по результатам работы программы

Красный график – исходный график, заданный в варианте. Синий график построен по результатам работы программы без зашумления, зеленый график построен при зашумлении A = 2.

Зашумленная функция:

x	f(x)
-2	1.27519
-1.73333	0.239054
-1.46667	1.40935
-1.2	0.632807
-0.933333	0.464927
-0.666667	-0.188808

```
| -0.4        | -0.338926  |
| -0.133333   | -0.337551  |
| 0.133333    | -0.180043  |
| 0.4         | -1.1154    |
| 0.666667    | -0.520015  |
| 0.933333    | -0.410055  |
| 1.2         | 0.246553   |
| 1.46667     | -0.238842  |
| 1.73333     | -0.100744  |
| 2           | -1.49998   |
---------------------------
```

Результат работы программы:

1. Без шумов: y = -0.540816x + 0.0265511
2. С зашумлением: y = -0.459184x - 0.0711081

Выводы

По результатам работы программы видно, что алгоритм успешно аппроксимирует как не зашумленные, так и зашумленные функции. Отклонения от исходного графика на заданном интервале очень малы.

Приложение. Исходный код программы

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

struct Point {
    double x;
    double y;
};

struct Coefficients {
    double c;
    double d;
};

double linearFunction(const double _x) {
    return -0.5*_x + 0;
}

double error(const std::vector<Point>& right, const double c, const double d) {
    double sum = 0.;
    for (auto point : right) {
        sum += pow(point.y - (c * point.x + d), 2);
    }
    return sum;
}

double randomInRange(const double lower, const double upper) {
    return lower + rand() * 1./RAND_MAX * (upper - lower);
}
```

```
std::vector<Point> divideInterval(const double lower, const double upper,
                                  const size_t pointsNumber, const double fault) {
    std::vector<Point> points(pointsNumber);
    const double step = (upper - lower) / static_cast<double>(pointsNumber - 1);
    for (size_t i = 0; i < pointsNumber; ++i) {
        points[i].x = lower + i * step;
        points[i].y = linearFunction(points[i].x) + randomInRange(- fault / 2, fault /
2);
    }
    return points;
}

void findBestDCoefficientForC(const std::vector<Point>& points, Coefficients& current)
{
    const double D_MIN = -2;
    const double D_MAX = 2;
    const size_t iterations = 50;

    current.d = D_MIN;
    for (size_t i = 0; i < iterations; ++i) {
        double new_d = randomInRange(D_MIN, D_MAX);
        if (error(points, current.c, new_d) < error(points, current.c, current.d)) {
            current.d = new_d;
        }
    }
}

Coefficients findBestCoefficients(const std::vector<Point>& points) {
    const double C_MIN = -2.5;
    const double C_MAX = 1.5;
    const size_t iterations = 50;
    const double step = (C_MAX - C_MIN) / static_cast<double>(iterations - 1);

    std::vector<Coefficients> allVariants;
    for (size_t i = 0; i < iterations; ++i) {
        Coefficients newCoefficients{};
        newCoefficients.c = C_MIN + i * step;
        findBestDCoefficientForC(points, newCoefficients);
        allVariants.push_back(newCoefficients);
    }

    Coefficients bestCoefficients = allVariants[0];
    for (const Coefficients& item : allVariants) {
        if (error(points, item.c, item.d) < error(points, bestCoefficients.c,
bestCoefficients.d)) {
            bestCoefficients = item;
        }
    }
    return bestCoefficients;
}

void printTable(const std::vector<Point>& points) {
    std::cout << std::string(27, '-') << std::endl;
    std::cout << "| " << std::left << std::setw(10) << 'x'
              << " | " << std::setw(10) << "f(x)" << " |\n";
    std::cout << std::string(27, '-') << std::endl;
    for (const auto& item : points) {
        std::cout << "| " << std::setw(10) << item.x
                  << " | " << std::setw(10) << item.y << " |\n";
    }
    std::cout << std::string(27, '-') << std::endl;
}
```

```cpp
int main() {
    const double MINIMUM = -2.;
    const double MAXIMUM = 2.;
    const size_t POINTS = 16;
    const double ERROR_LIMIT = 2.;

    // Part 1. Function without noise
    srand(time(nullptr));
    auto points_1 = divideInterval(MINIMUM, MAXIMUM, POINTS, 0.);
    auto coefficients_1 = findBestCoefficients(points_1);
    std::cout << "Part 1. Without noise\n"
              << "Right function: y = -0.5x\nTable of values:\n";
    printTable(points_1);
    std::cout << "Found function: y = " << coefficients_1.c
              << "x " << (coefficients_1.d >= 0 ? "+ ": "- ")
              << (coefficients_1.d >= 0 ? coefficients_1.d : (coefficients_1.d * -1.))
<< std::endl;

    // Part 2. Function with noise
    auto points_2 = divideInterval(MINIMUM, MAXIMUM, POINTS, ERROR_LIMIT);
    auto coefficients_2 = findBestCoefficients(points_2);
    std::cout << "Part 2. With noise\n"
              << "Right function: y = -0.5x + random(A)\nTable of values:\n";
    printTable(points_2);
    std::cout << "Found function: y = " << coefficients_2.c
              << "x " << (coefficients_2.d >= 0 ? "+ ": "- ")
              << (coefficients_2.d >= 0 ? coefficients_2.d : (coefficients_2.d * -1.))
<< std::endl;

    return 0;
}
```

Ответ на контрольный вопрос

1. Поясните суть метода наименьших квадратов.

Задача заключается в нахождении коэффициентов линейной зависимости, при которой функция двух переменных w_1 и w_0:

$$E^2(w_1, w_0) = \sum_{i=1}^{N} [y(x_i) - t_i]^2 \rightarrow \min_{c,d}$$

Принимает наименьшее значение. То есть, при данных w_1 и w_0 сумма квадратов отклонений экспериментальных данных от исходной прямой будет наименьшей. В этом суть метода наименьших квадратов.