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 7

Выполнил: Кидинова Д.Д.,
студент группы ИУ8-31

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

г. Москва,
2020 г.

1. Цель работы

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

2. Условие задачи

Вариант № 7.

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 $e_i = A\,\mathrm{rnd}(-0.5;0.5)$. Выполнить расчет параметров c,d градиентным методом. Провести двумерный пассивный поиск оптимальных весовых коэффициентов нейронной сети (НС) регрессии.

w1 = 8, w0 = 0, a = -4, b = 2, N = 27, A = 10. Алгоритм поиска c - золотое сечение, алгоритм поиска d — пассивный.

3. Графики

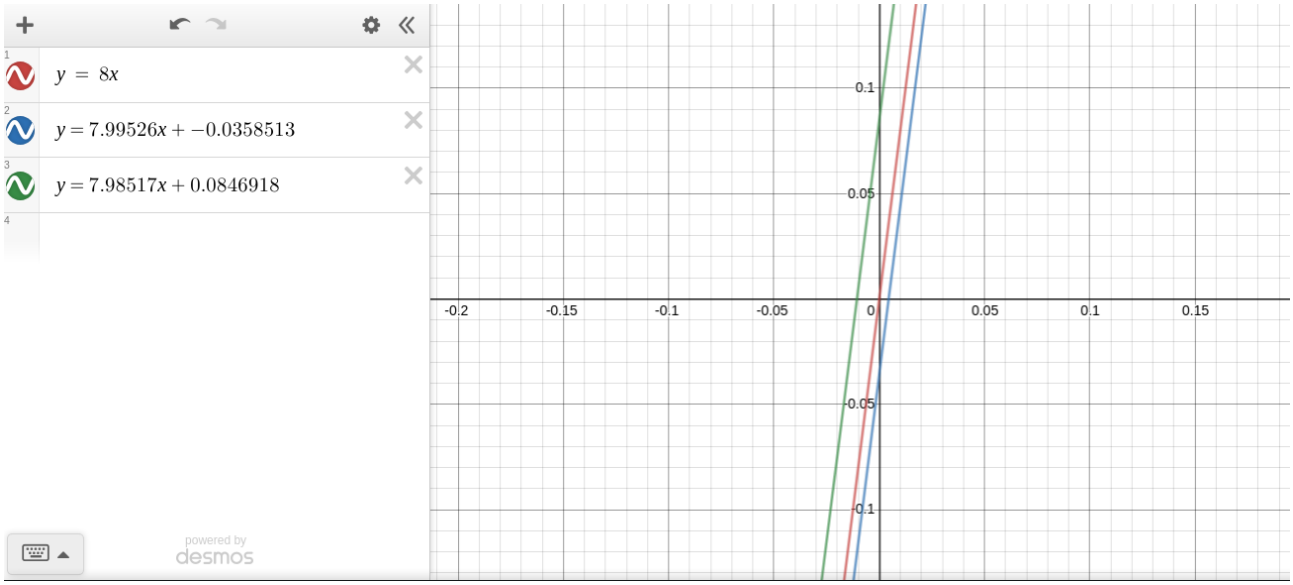

Рисунок 1: Графики, постороенные по результатам работы программы

Красный график — исходный, заданный в варианте. Синий график постороен по результатам работы при=ограммы при шуме A = 0, Зеленый — при шуме A = 10.

4. Результат работы программы

Results without noise for function y = 8 * x + 0 (w1 = 8, w0 = 0) in [-4 , 2]

Cmin = 0

Cmax = 8

Dmin = -141.833

Dmax = 80.5

w1 = 7.99526

w0 = -0.0358513

Error = 0.0249845

Results with noise A = 10 for function y = 8 * x + 0 (w1 = 8, w0 = 0) in [-4 , 2]

Cmin = 0

Cmax = 13.3138

Dmin = -139.821

Dmax = 78.4767

w1 = 7.98517

w0 = 0.0846918

Error = 115.737

5. Выводы

Результаты работы совпали с ожидаемыми: в отсутствии шума алгоритм дает точные значения параметров регрессии.

6. 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*

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

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 сводится к нахождению экстремума функции двух переменных.

Приложение 1. Исходный код программы «Задача 1»

```
#include <iostream>
#include <random>
#include <map>
#include <vector>
#include <algorithm>

using std::map;
using std::cout;
using std::vector;

const double LOWER_EDGE = -4.;
const double UPPER_EDGE = 2.;
const double C_COEFFICIENT = 8.;
const double D_COEFFICIENT = 0.;
const size_t N = 24;
const double A = 10.;
const double STEP = (UPPER_EDGE - LOWER_EDGE) / static_cast<double>(N - 1);

double LinearFunction(const double &c, const double &d, const double &x) {
    return c * x + d;
}

map<double, double> GenerateNumbers(const double &a, const double &b, const size_t N,
const double &noise) {
    map<double, double> points;
    std::random_device rd;
    std::mt19937 gen(rd());
    std::uniform_real_distribution<double> error(-0.5, 0.5);
    for (size_t i = 0; i < N; ++i) {
        double x = LOWER_EDGE + i * STEP;
        double y = LinearFunction(C_COEFFICIENT, D_COEFFICIENT, LOWER_EDGE + i * STEP) +
noise * error(gen);
        points.insert(std::make_pair(x, y));
    }
    return points;
}

void FindEdgesForMinSearch(const map<double, double> &points, double &Cmin, double
&Cmax, double &Dmin, double &Dmax) {
    Cmin = 0, Cmax = 0;
    auto y_next = ++points.begin();
    for (const auto &y_curr : points) {
        if (y_next != points.end()) {
            if (y_next->second - y_curr.second > Cmax) Cmax = (y_next->second - y_curr.second) /
STEP;
            if (y_next->second - y_curr.second < Cmin) Cmin = (y_next->second - y_curr.second) /
STEP;
        }
        ++y_next;
    }
```

```
    if (points.begin()->second < points.rbegin()->second) Cmin = 0;
    if (points.begin()->second > points.rbegin()->second) Cmax = 0;
    Dmax = (points.begin()->second < points.rbegin()->second ? points.rbegin()->second :
points.begin()->second) +
           A * 0.5;
    Dmin = (points.begin()->second > points.rbegin()->second ? points.rbegin()->second :
points.begin()->second) -
           A * 0.5;
    Dmin /= STEP;
    Dmax /= STEP;
}

double SumOfSquaredErrors(const map<double, double> &points, const double &w1, const
double &w0) {
    double sum = 0;
    for (const auto &p : points) {
        sum += (LinearFunction(w1, w0, p.first) - p.second) * (LinearFunction(w1, w0, p.first) -
p.second);
    }
    return sum;
}

double GoldenRatioMinSearch(const map<double, double> &points, double lower, double
upper) { //search for w1
    double goldenNumber = (1 + sqrt(5)) / 2;
    double x_left = lower + (1 - 1 / goldenNumber) * upper;
    double x_right = lower + upper / goldenNumber;
    double epsilon = 0.01;
    double w0 = 0;
    while (upper - lower > epsilon) {
        if (SumOfSquaredErrors(points, x_left, w0) < SumOfSquaredErrors(points, x_right, w0)) {
            upper = x_right;
            x_right = lower + upper - x_left;
        } else {
            lower = x_left;
            x_left = lower + upper - x_right;
        }
        if (x_left > x_right)
            std::swap(x_left, x_right);
    }
    return (x_left + x_right) / 2;
}

double PassiveSearch(const map<double, double> &points, const double &lower, const double
&upper,
                     const double &w1) { //search for w0
    size_t number = 1;
    double epsilon = 0.08;
    double delta = (upper - lower) / (number + 1);
    double xForMinY;
    while (delta > epsilon) {
        std::vector<double> VectorOfYk;
        delta = (upper - lower) / (number + 1);
```

```
        for (uint k = 1; k <= number; ++k) {
            VectorOfYk.push_back(SumOfSquaredErrors(points, w1, delta * k + lower));
        }
        uint kForMinY = std::min_element(VectorOfYk.begin(), VectorOfYk.end()) -
VectorOfYk.begin() + 1;
        xForMinY = (upper - lower) / (number + 1) * kForMinY + lower;
        ++number;
    }
    return xForMinY;
}

void PrintResults(const double &noise) {
    map<double, double> points = GenerateNumbers(LOWER_EDGE, UPPER_EDGE, N, noise);
    size_t i = 1;
//    for (const auto &p : points) {
//        cout << i << " ( " << p.first << " , " << p.second << " )\n";
//        ++i;
//    }
    double Cmin, Cmax, Dmin, Dmax;
    FindEdgesForMinSearch(points, Cmin, Cmax, Dmin, Dmax);
    cout << "Cmin = " << Cmin << "\nCmax = " << Cmax << "\nDmin = " << Dmin << "\
nDmax = " << Dmax << "\n";
    double w1 = GoldenRatioMinSearch(points, Cmin, Cmax);
    cout << "w1 = " << w1 << "\n";
    double w0 = PassiveSearch(points, Dmin, Dmax, w1);
    cout << "w0 = " << w0 << "\n";
    cout << "Error = " << SumOfSquaredErrors(points, w1, w0) << "\n";
}

int main() {
    cout << "Results without noise for function y = 8 * x + 0 (w1 = 8, w0 = 0) in [ " <<
LOWER_EDGE << " , "
         << UPPER_EDGE << " ]\n";
    PrintResults(0.);
    cout << "Results with noise A = " << A <<" for function y = 8 * x + 0 (w1 = 8, w0 = 0) in
[ " << LOWER_EDGE << " , "
         << UPPER_EDGE << " ]\n";
    PrintResults(A);
}
```