|  |  |
| --- | --- |
| Gerb-BMSTU_01 | **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 «№3»

Выполнил: Быкова П. Е.,

студентка группы ИУ8-32

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. Постановка задачи

В зависимости от варианта работы (табл. 4.1) 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\*rnd(-0.5;0.5). Выполнить расчет параметров c, d градиентным методом. Провести двумерный пассивный поиск оптимальных весовых коэффициентов нейронной сети (НС) регрессии.

# 3. Ход работы

1. Рассчитать равномерно распределенные на вашем интервале точки (N дискретных значений, N вводит пользователь программы),

2. Построить график заданной по варианту функций f (x) ;

3. При помощи генератора псевдослучайных чисел с указанными параметрами осуществить зашумление рассчитанных точек, далее использовать их при регрессии;

4. Отобразить эти зашумленные точки на графике;

5. Реализовать программно нейросетевую модель,

6. Используя указанные по варианту методы, осуществить поиск параметров c\*, d\*;

7. Построить график полученной приближенной регрессии;

8. Рассчитайте и выведите получившуюся погрешность приближения, используя рассчитанные при помощи найденных параметров c\*, d \* точки и исходные отсчеты.

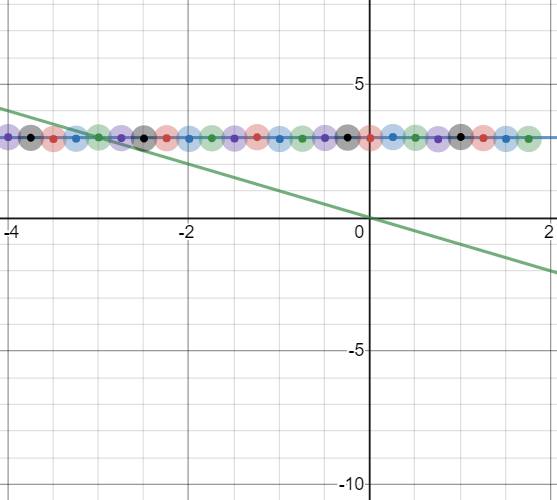


Рисунок 5.1 Графики функций

Сплошная голубая линия - график функции y = 3;

Сплошная зелёная линия – график полученной приближённой регрессии;

Разноцветные точки – зашумлённые точки.

**4. Выводы**

По результатам лабораторной работы я познакомилась с устройством простейшей нейронной сети и реализовала алгоритм поиска ее коэффициентов.

**5. Контрольные вопросы**

*Вопрос:* *Поясните суть метода наименьших квадратов.*

*Ответ: В принципе наименьших квадратов параметры должны быть выбраны таким образом, чтобы минимизировать сумму квадратов ошибок для всех точек:*

*.*

*То есть, при различных значениях a и b сумма квадратов отклонений представленных данных от получившейся прямой (учитываемая ошибка экспериментальных данных) будет иметь минимальное значение. В этом и состоит суть метода наименьших квадратов.*

**Приложение 1. Исходный код программы «Нейронные сети»**

*Файл* ***code\_of\_the\_laboratory\_work.cpp***

// вариант №3

#include <iostream>

#include <random>

#include <vector>

#include <algorithm>

const int N = 24; // ккол-во дискретных значений в наборе (из варианта)

// создаём вектор пар, в который будем записывать набор экспериментальных данных вида {x,t}

std::vector<std::pair<double, double>> function(24);

// метод, возвращающий сумму квадратов ошибок всех N точек

double recurssion\_function(double C, double D) {

double sum\_E = 0;

for (int i = 0; i < N; i++) { // для вектора пар от 0 до 23 значений

double x = function[i].first; // устанавливаем в паре "х" первой координатой

double t = function[i].second; // устанавливаем в паре "t" второй координатой

double y = C \* x + D; // считаем значение нашей y(x) нашей линейной зависимости

sum\_E = sum\_E + pow((y - t), 2); // считаем сумму квадратов ошибок для всех точек

}

return sum\_E;

}

// метод, определяющий коэффициент "С" с помощью прямого пассивного поиска

double Method\_Passive\_Search() {

// задаём длинну интервала неопределённости, по достижению котрой коэффициент "С" считается найденным

const double eps = 0.01;

double e = 1.; // текущая ошибка кнкретной точки

double now\_C; // текущий коэффициент "С"

double C = 0.; // переменная для записи коэффициента "С"

double min\_possible\_C = pow(10, 200); // заведомо большое число, с которым будет сравниваться искомый коэффициент "С"

double iterator = 1.; // счётчик

const double C\_max = 1.; // значение определённое самим, исходя из варианта

const double C\_min = -1.; // значение определённое самим, исходя из варианта

while (e > eps) { // пока заведомо большая ошибка данной точки не достигла отметки 0,01, то делаем:

for (int m = 0; m < (iterator + 1.); m++) {

// вычисляем коэффициент "С" на текущей итерации (формула из методички)

now\_C = (((C\_max - C\_min) \* m) / (N - 1.)) + C\_min;

// если суммарная ошибка квадратов всех точек с таким коэффициентом "С" меньше заведомо огромной ошибки, то:

if (recurssion\_function(now\_C, 0.) < min\_possible\_C) {

// меняем заведомо огромную ошибку на суммарную ошибку квадратов всех точек для дальнейшего сравнения

min\_possible\_C = recurssion\_function(now\_C, 0.);

// "удачный" коэффициент "С" записываем

C = now\_C;

}

}

e = (C\_max - C\_min) / (iterator + 1.); // меняем величину текущей ошибки

iterator++; // увеличиваем счётчик

}

return C;

}

// метод, определяющий коэффициент "D" с помощью метода дихотомии

double Method\_dihotomia(double a, double b, double C) { // подаём на вход 3 параметра: длинну отрезка [a,b] и коэффициент "С"

// задаём длинну интервала неопределённости, по достижению котрой коэффициент "D" считается найденным

const double eps = 0.1;

// задаём некоторое малое число "дельта", которое будем прибавлять слева и справа

const double delta = 0.01;

// правое и левое значение коэффициента "D" (аналог "х" в стандартном методе дихотомии)

double D\_left, D\_right;

// правое и левое значение функции

double y\_left, y\_right;

do { // выполняем алгоритм до тех пор, пока наш интервал не достигнет отметки 0,1

D\_left = 0.5 \* (b + a) - delta; // вычисляем левое значение по формуле из стандартного метода дихотомии

D\_right = 0.5 \* (b + a) + delta; // вычисляем правое значение по формуле из стандартного метода дихотомии

y\_left = recurssion\_function(C, D\_left); // вычисляем левое значение функции при левом коэффициенте "D"

y\_right = recurssion\_function(C, D\_right); // вычисляем правое значение функции при правом коэффициенте "D"

// если правое значение функции больше левого значения, то:

if (y\_right > y\_left) {

b = D\_right; // меняем правую границу интервала на правый коэффициент "D"

}

else {

a = D\_left; // меняем левую границу интервала на левый коэффициент "D"

}

} while ((b - a) > eps);

return ((b + a) / 2); // возвращаем коэффициент "D" (формула из стандартного метода дихотомии)

}

int main() {

double a = -4.;

double b = 2.;

double A = 0.1; // значения из варианта

double c = 0.;

double d = 3.;

double iterator\_x = (b - a) / N; // длинна шага

int i = 0; // счётчик итераций

std::random\_device rand; //...

std::mt19937 generation(rand()); //... эти три строчки позволяют выбрать случайным образом одно число типа double из интервала (-0.5, 0.5)

std::uniform\_real\_distribution<double> interval(-0.5, 0.5); //...

double C = Method\_Passive\_Search(); // находим коэффициент "С"

double D = Method\_dihotomia(-4, 2, C); // находим коэффициент "D"

std::cout << "Found coefficients: " << "C = " << C << "; D = " << D << std::endl;

for (double x2 = a; x2 < b; x2 += iterator\_x) {

// делаем зашумление рассчитанных точек (формула из методички)

double t2 = c \* x2 + d + A \* interval(generation);

// записываем набор полученных пар в наш вектор значений

function[i] = { x2, t2 };

// переход на следующую итерацию

i++;

}

// выводим полученный набор на экран

std::cout << "Points with noise:" << std::endl;

for (int i = 0; i < N; i++) {

std::cout << "(" << function[i].first << ";" << function[i].second << ")" << std::endl;

}

return 0;

}

**Приложение 2. Ссылка на git-репозиторий**

https://github.com/PolkaBBB/TSISA-lab05