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 6

Выполнил: Калинин Д. В.,
студент группы ИУ8-31

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

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

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

Таблица 1 – Исходные данные

c	**d**	**a**	**b**	**N**	**A**
0.5	0	-2	1	20	1

3. Ход работы

Для наглядности построим график исходной зависимости $f(x) = cx + d$ (рисунок 1).

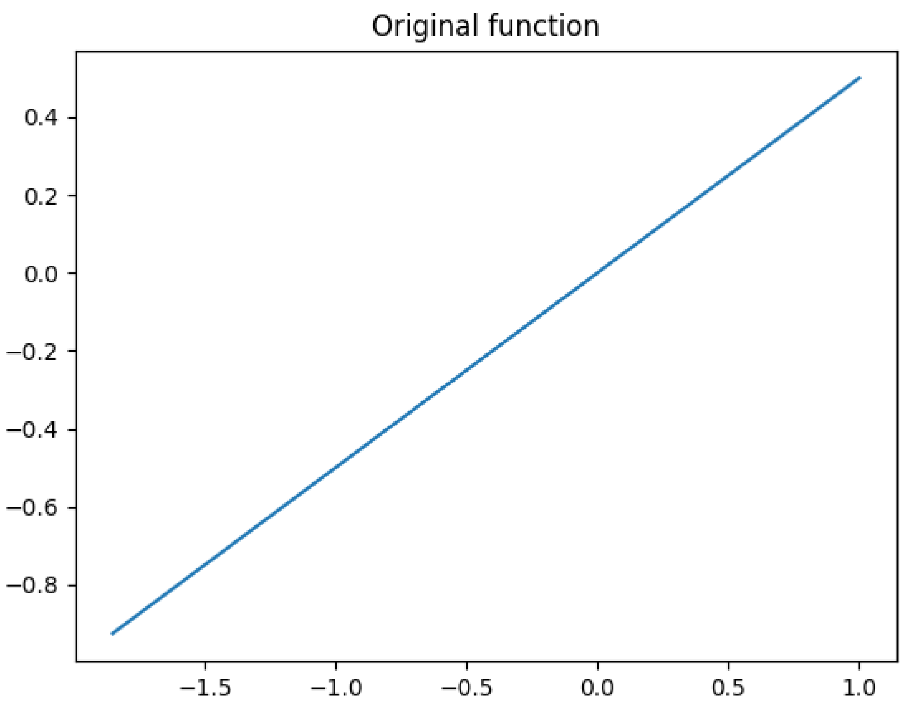

Рисунок 1 – График исходной зависимости

Найдём необходимые параметры $c_{min}, c_{max}, d_{min}, d_{max}$.

$$d_{min,max} = \pm 0.5A = \pm 0.5.$$

Значение c_{min} найдём из СЛАУ:

$$\begin{cases} f(a) + d_{max} = -2c_{min} + d \\ f(b) + d_{min} = \quad c_{min} + d \end{cases},$$

откуда $c_{min} \approx 0.17$.

Аналогично получаем $c_{max} \approx 0.83$.

На рисунке 2 приведён результат работы программы. Здесь красными точками на графике отмечены зашумленные отсчеты; синяя линия – график зависимости, найденной программой.

Код программы доступен по ссылке: https://github.com/shreddered/lab-05.

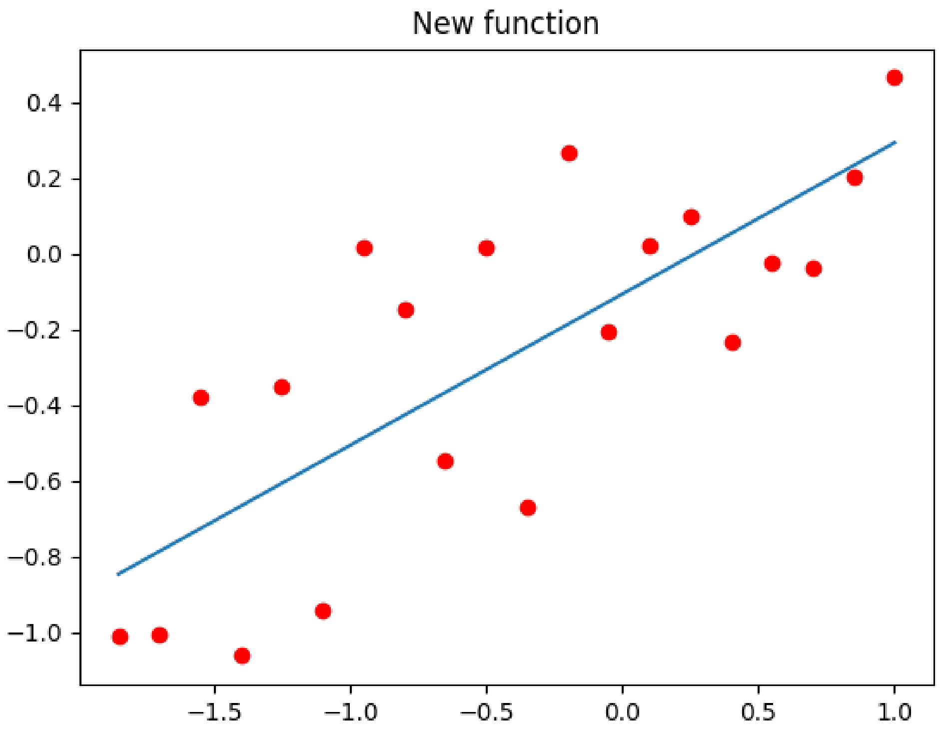

Рисунок 2 – Результаты работы программы

4. 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 А. Исходный код программы

Файл approximator.py

```
import matplotlib.pyplot as plt
import random
from datetime import datetime

random.seed(datetime.now())

class Approximator:
    verbose = True
    def __init__(self, n: int, interval: tuple, params: tuple):
        self.n = n
        self.a, self.b = interval[0], interval[1]
        self.c, self.d = params[0], params[1]
        step = (self.b - self.a) / self.n
        self.x = []
        tmp = self.a + step
        while (tmp < self.b):
            self.x.append(tmp)
            tmp += step
        self.y = [self.c * x_ + self.d for x_ in self.x]
    def search(self, amp: int):
        if Approximator.verbose:
            # plot of original function
            plt.plot(self.x, self.y)
            plt.title('Original function')
            plt.show()
        self.y = [y + amp * random.uniform(-0.5, 0.5) for y in self.y]
        c = self._dichotomy(0.17, 0.83)
        d = self._passive_search(-0.5, 0.5, c)
        if Approximator.verbose:
            # plot of new function
            plt.plot(self.x, self.y, 'ro')
            y_approx = [c * x_ + d for x_ in self.x]
            plt.plot(self.x, y_approx)
            plt.title('New function')
            plt.show()
        return (c, d)
    def _sum_of_squares(self, c: float, d: float):
        s = 0.0
        for i in range(0, self.n):
            x = self.x[i]
            t = self.y[i]
            y = c * x + d
            s += (y - t) * (y - t)
        return s
    def _passive_search(self, d_min: float, d_max: float, c: float):
        eps = 0.01
        res = 0.0
        n = 2
        while ((d_max - d_min) / n > eps):
            d = []
            tmp = d_min
            step = (self.b - self.a) / n
            while tmp < d_max:
                d.append(tmp)
                tmp += step
            squares = [self._sum_of_squares(c, d_) for d_ in d]
            res = d[squares.index(min(squares))]
            n += 1
        return res
```

```
    def _dichotomy(self, c_min: float, c_max: float):
        eps = 0.01
        delta = 0.001
        while ((c_max - c_min) > eps):
            c1, c2 = 0.5 * (c_min + c_max) - delta, 0.5 * (c_min + c_max) +
delta
            d1 = self._passive_search(-0.5, 0.5, c1)
            d2 = self._passive_search(-0.5, 0.5, c2)
            if self._sum_of_squares(c2, d2) > self._sum_of_squares(c1, d1):
                c_max = c2
            else:
                c_min = c1
        return (c_min + c_max) / 2
```

Файл approximator.py

```
#!/usr/bin/env python3

from approximator import Approximator

if __name__ == "__main__":
    approx = Approximator(n = 20, interval = (-2, 1), params = (0.5, 0))
    res = approx.search(amp = 1)
    print("c = {}, d = {}".format(res[0], res[1]))
```

Контрольные вопросы

Поясните суть метода наименьших квадратов.

Суть метода наименьших квадратов заключается в том, чтобы минимизировать сумму квадратов отклонений некоторых значений от эталонных. Таким образом, с использованием различных методов поиска в данной лабораторной работе были найдены коэффициенты линейной регрессии.

```python
import matplotlib.pyplot as plt
import random
from datetime import datetime

random.seed(datetime.now())

class Approximator:
    verbose = True
    def __init__(self, n: int, interval: tuple, params: tuple):
        self.n = n
        self.a, self.b = interval[0], interval[1]
        self.c, self.d = params[0], params[1]
        step = (self.b - self.a) / self.n
        self.x = []
        tmp = self.a + step
        while (tmp < self.b):
            self.x.append(tmp)
            tmp += step
        self.y = [self.c * x_ + self.d for x_ in self.x]
    def search(self, amp: int):
        if Approximator.verbose:
            # plot of original function
            plt.plot(self.x, self.y)
            plt.title('Original function')
            plt.show()
        self.y = [y + amp * random.uniform(-0.5, 0.5) for y in self.y]
        c = self._dichotomy(0.17, 0.83)
        d = self._passive_search(-0.5, 0.5, c)
        if Approximator.verbose:
            # plot of new function
            plt.plot(self.x, self.y, 'ro')
            y_approx = [c * x_ + d for x_ in self.x]
            plt.plot(self.x, y_approx)
            plt.title('New function')
            plt.show()
        return (c, d)
    def _sum_of_squares(self, c: float, d: float):
        s = 0.0
        for i in range(0, self.n):
            x = self.x[i]
            t = self.y[i]
            y = c * x + d
            s += (y - t) * (y - t)
        return s
    def _passive_search(self, d_min: float, d_max: float, c: float):
        eps = 0.01
        res = 0.0
        n = 2
        while ((d_max - d_min) / n > eps):
            d = []
            tmp = d_min
            step = (self.b - self.a) / n
            while tmp < d_max:
                d.append(tmp)
                tmp += step
            squares = [self._sum_of_squares(c, d_) for d_ in d]
            res = d[squares.index(min(squares))]
            n += 1
        return res
    def _dichotomy(self, c_min: float, c_max: float):
        eps = 0.01
        delta = 0.001
        while ((c_max - c_min) > eps):
            c1, c2 = 0.5 * (c_min + c_max) - delta, 0.5 * (c_min + c_max) + delta
            d1 = self._passive_search(-0.5, 0.5, c1)
```

```python
import matplotlib.pyplot as plt
import random
from datetime import datetime

random.seed(datetime.now())

class Approximator:
    verbose = True
    def __init__(self, n: int, interval: tuple, params: tuple):
        self.n = n
        self.a, self.b = interval[0], interval[1]
        self.c, self.d = params[0], params[1]
        step = (self.b - self.a) / self.n
        self.x = []
        tmp = self.a + step
        while (tmp < self.b):
            self.x.append(tmp)
            tmp += step
        self.y = [self.c * x_ + self.d for x_ in self.x]
    def search(self, amp: int):
        if Approximator.verbose:
            # plot of original function
            plt.plot(self.x, self.y)
            plt.title('Original function')
            plt.show()
        self.y = [y + amp * random.uniform(-0.5, 0.5) for y in self.y]
        c = self._dichotomy(0.17, 0.83)
        d = self._passive_search(-0.5, 0.5, c)
        if Approximator.verbose:
            # plot of new function
            plt.plot(self.x, self.y, 'ro')
            y_approx = [c * x_ + d for x_ in self.x]
            plt.plot(self.x, y_approx)
            plt.title('New function')
            plt.show()
        return (c, d)
    def _sum_of_squares(self, c: float, d: float):
        s = 0.0
        for i in range(0, self.n):
            x = self.x[i]
            t = self.y[i]
            y = c * x + d
            s += (y - t) * (y - t)
        return s
    def _passive_search(self, d_min: float, d_max: float, c: float):
        eps = 0.01
        res = 0.0
        n = 2
        while ((d_max - d_min) / n > eps):
            d = []
            tmp = d_min
            step = (self.b - self.a) / n
            while tmp < d_max:
                d.append(tmp)
                tmp += step
            squares = [self._sum_of_squares(c, d_) for d_ in d]
            res = d[squares.index(min(squares))]
            n += 1
        return res
    def _dichotomy(self, c_min: float, c_max: float):
        eps = 0.01
        delta = 0.001
        while ((c_max - c_min) > eps):
            c1, c2 = 0.5 * (c_min + c_max) - delta, 0.5 * (c_min + c_max) + delta
            d1 = self._passive_search(-0.5, 0.5, c1)
```