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 № 2
по дисциплине «Интеллектуальные технологии информационной безопасности»

Тема: «Применение однослойной нейронной сети с линейной функцией активации для прогнозирования временных рядов»

Вариант 1

Выполнил: Антипов И.С.,
студент группы ИУ8-63

Проверил: Волосова Н.К.,
преподаватель каф. ИУ8

г. Москва,
2021 г.

1. Цель работы

Изучить возможности однослойных НС в задачах прогнозирования временных рядов методом скользящего окна (авторегрессия)

2. Условие

Задана функция $x(t)$ на интервале $[-5; 5]$:

$$x(t) = 0.5(\cos 0.5\, t - 0.5)$$

Необходимо обучить нейронную сеть, а затем спрогнозировать поведение функции на интервале $[5; 10]$.

3. Ход работы

Первый этап – это обучение нейронной сети. Для начала необходимо научиться прогнозировать на известной выборке с помощью скользящего окна. Для этого используется авторегрессионная модель.

$$\tilde{x}_n = \sum_{k=1}^{p} w_k x_{n-p+k-1} + w_0$$

Вектор-столбцов обучающей выборки, подаваемый на вход:

$$X = \begin{pmatrix} x_1 & \cdots & x_{m-p} \\ \vdots & \ddots & \vdots \\ x_p & \cdots & x_{m-1} \end{pmatrix}$$

Прогнозируемые значения на выходе (1) сравниваются с реальными (2):

$$\tilde{x}_{p+1}, \tilde{x}_{p+2}, \dots, \tilde{x}_{m-1} \qquad (1)$$
$$x_{p+1}, x_{p+2}, \dots, x_{m-1} \qquad (2)$$

В самом начале весовые коэффициенты нулевые:

$$w_0^0 = w_1^0 = w_2^0 = w_3^0 = w_4^0 = 0$$

Коррекция весов происходит по правилу Видроу-Хоффа:

$$w_i^{l+1} = w_i^l + \Delta w_i^l$$

$$\Delta w_i^l = \eta \delta x_i^l$$

Если по достижении правого края выборки суммарная среднеквадратичная ошибка $\varepsilon = \sqrt{\sum_i [x(t_i) - \tilde{x}(t_i)]^2}$ останется достаточно большой, следует продолжить обучение, снова вернувшись к первому столбцу.

Следующий этап – прогнозирование функции на последующем отрезке. Используя полученные при обучении веса, необходимо с помощью скользящего окна дойти до конца следующего отрезка. Результат прогнозирования для количества эпох обучения = 500 представлен на рисунке 1.

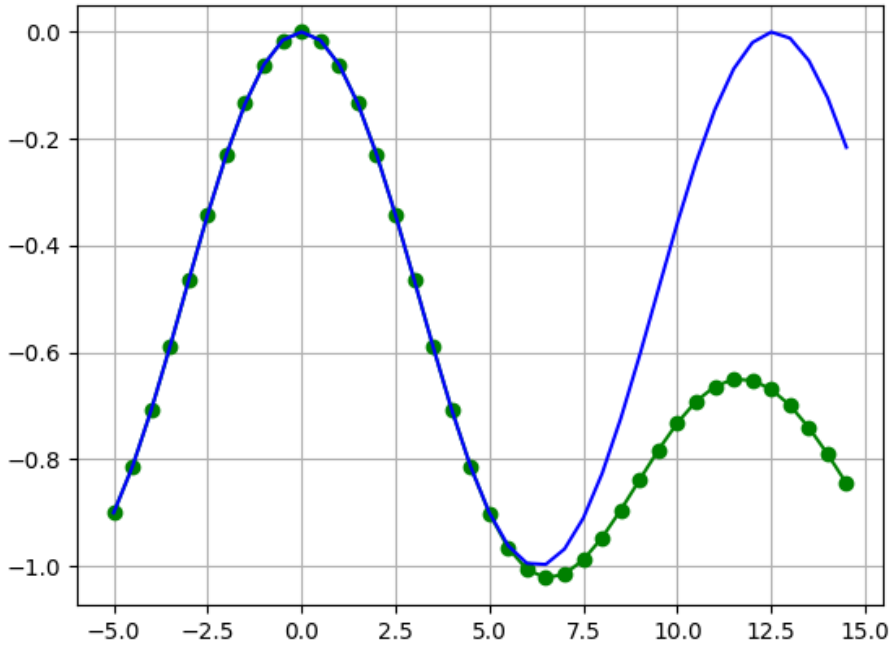

Рисунок 1 – Результат прогноза при обучении на 500 эпохах

Суммарная среднеквадратичная ошибка при 500 эпохах:

$$\varepsilon = 1{,}943$$

Результат прогнозирования для количества эпох обучения = 1500 представлен на рисунке 2.

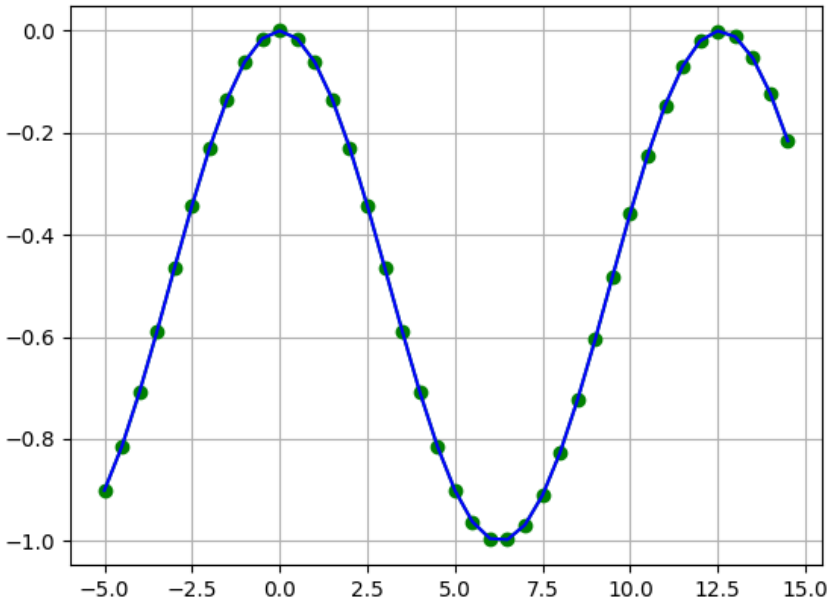

Рисунок 2 – Результат прогноза при обучении на 1500 эпохах

Суммарная среднеквадратичная ошибка при 500 эпохах:

$$\varepsilon = 1{,}943$$

Следующий этап – сравнение среднеквадратичной ошибки при разных эпохах. Эта зависимость представлена на рисунке 3.

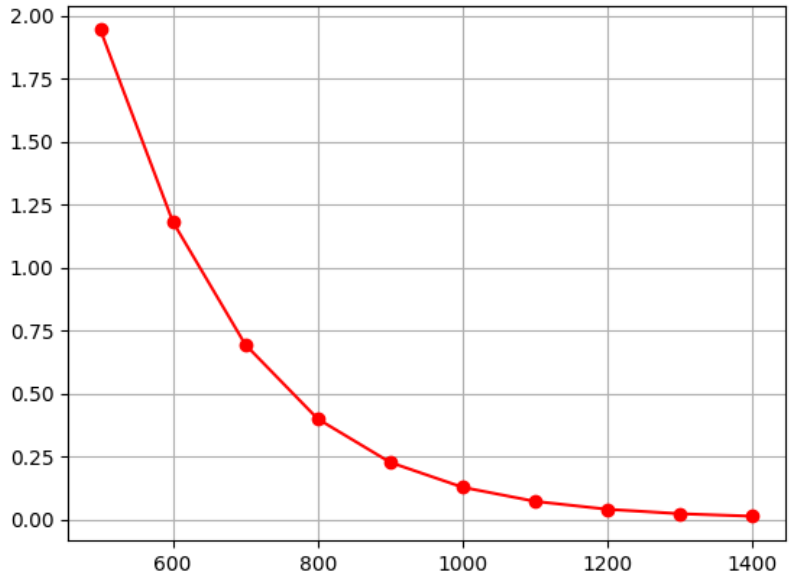

Рисунок 3 – Зависимость среднеквадратичной ошибки от количества эпох

Код программы приведен в Приложении А.

4. Выводы

В ходе выполнения лабораторной работы было произведено обучение нейронной сети с помощью метода скользящего окна. Далее был проведен прогноз относительно поведения функции на последующем отрезке.

Результаты прогноза с увеличением количества эпох приближаются к реальным, что говорит о корректности обучения. С увеличением количества эпох обучения, ошибка постепенно снижается.

Приложение А. Исходный код программы

Файл main.py

```
import math
import matplotlib.pyplot as plot


class NeuralNetwork:
    def __init__(self, eta, N, a, b, p, M):
        self.__eta = eta
        self.__N = N
        self.__a = a
        self.__b = b
        self.__p = p
        self.__M = M
        self.__epoch = 0
        self.__weight = [0] * (p + 1)
        self.__x = []
        self.__y = []
        self.__realY = []
        self.__error = 0

    def generateX(self):
        deltaX = (abs(self.__b) + abs(self.__a)) / self.__N
        for i in range(self.__p):
            self.__x.append(self.__a + i * deltaX)

    def generateY(self):
        for i in self.__x:
            self.__y.append(self.func(i))
        self.__realY = self.__y.copy()

    def func(self, x):
        return 0.5 * math.cos(0.5 * x) - 0.5

    def countNet(self, i):
        net = 0
        for j in range(0, self.__p):
            net += self.__y[i + j] * self.__weight[j]
        net += self.__weight[self.__p]
        return net

    def weightCorrect(self, sigma, i):
        for j in range(0, self.__p):
            self.__weight[j] = self.__weight[j] + self.countDeltaWeight(sigma, self.__y[i + j])

    def countDeltaWeight(self, sigma, y_j):
        return sigma * self.__eta * y_j

    def countError(self):
        for j in range(self.__N, 2 * self.__N):
            self.__error += (self.__realY[j] - self.__y[j]) ** 2
        self.__error = math.sqrt(self.__error)

    def study(self):
        while self.__epoch < self.__M:
            self.__x.clear()
            self.__y.clear()
            self.__realY.clear()

            self.generateX()
            self.generateY()

            for i in range(self.__p, self.__N):
```

```
                y = self.countNet(i - self.__p)
                self.__x.append(self.__x[-1] + (abs(self.__b) + abs(self.__a)) / self.__N)
                self.__realY.append(self.func(self.__x[i]))

                self.weightCorrect(self.__realY[i] - y, i - self.__p)
                self.__y.append(self.__realY[i])

            self.__epoch += 1


    def forecast(self):
        for i in range(self.__N, 2 * self.__N):
            self.__y.append(self.countNet(i - self.__p))
            self.__x.append(self.__x[-1] + (abs(self.__b) + abs(self.__a)) / self.__N)
            self.__realY.append(self.func(self.__x[i]))

        self.countError()


    def getX(self):
        return self.__x

    def getY(self):
        return self.__y

    def getRealY(self):
        return self.__realY

    def getError(self):
        return self.__error


def printGraph(x, y, realY):
    plot.plot(x, y, 'go-')
    plot.plot(x, realY, 'b-')
    plot.grid(True)
    plot.show()

def printErrors(m, errors):
    plot.plot(m, errors, 'ro-')
    plot.grid(True)
    plot.show()


def start():
    nw1 = NeuralNetwork(1, 20, -5, 5, 4, 500)
    nw1.study()
    nw1.forecast()
    printGraph(nw1.getX(), nw1.getY(), nw1.getRealY())

    print("Amount of epochs 500\n")
    print("Error = ", nw1.getError(), "\n\n")

    nw2 = NeuralNetwork(1, 20, -5, 5, 4, 2000)
    nw2.study()
    nw2.forecast()
    printGraph(nw2.getX(), nw2.getY(), nw2.getRealY())

    print("Amount of epochs 1500\n")
    print("Error = ", nw2.getError(), "\n\n")

    errors = []
    m = [i for i in range(500, 1500, 100)]

    for i in range(len(m)):
        nwE = NeuralNetwork(1, 20, -5, 5, 4, m[i])
```

```
            nwE.study()
            nwE.forecast()
            errors.append(nwE.getError())

        printErrors(m, errors)


if __name__ == '__main__':
    start()
```