|  |  |
| --- | --- |
| 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 № 2

по дисциплине «Интеллектуальные технологии информационной безопасности»

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

Вариант 3

Выполнил: Горбачев А.А.,

студент группы ИУ8-63

Проверил: Волосова Н.К.,

аспирант каф. ИУ8

г. Москва,

2021 г.

# Цель работы

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

# Условия

Условия согласно варианту 3:

Функция:

Норма обучения

# Аналитическая часть

# Ход работы

Рассмотрим прогноз функции из условия варианта по 20 равноотстоящим исходным значениям x. Размер окна будет равным 4ем, норма обучения . Вес смещения

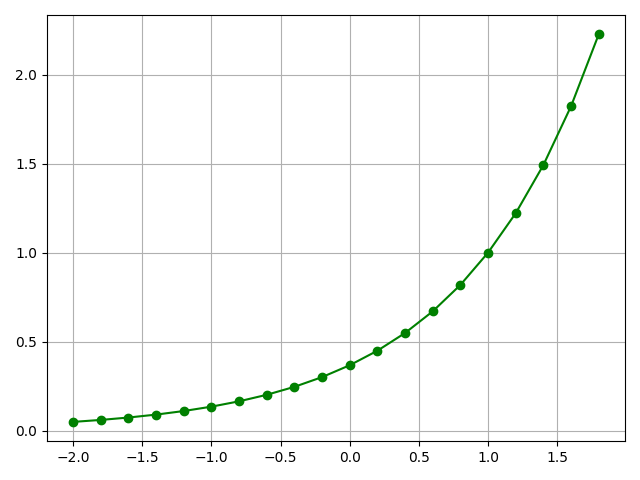


Рисунок 1 – Исходная функция:

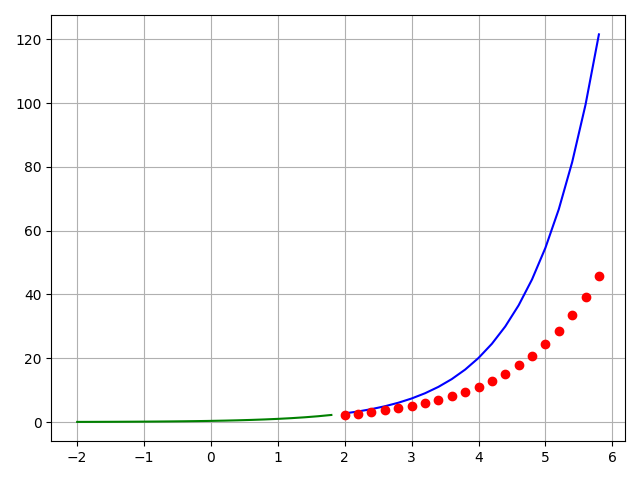


Рисунок 2 – Исходная функция и ее прогноз при M = 10, ():

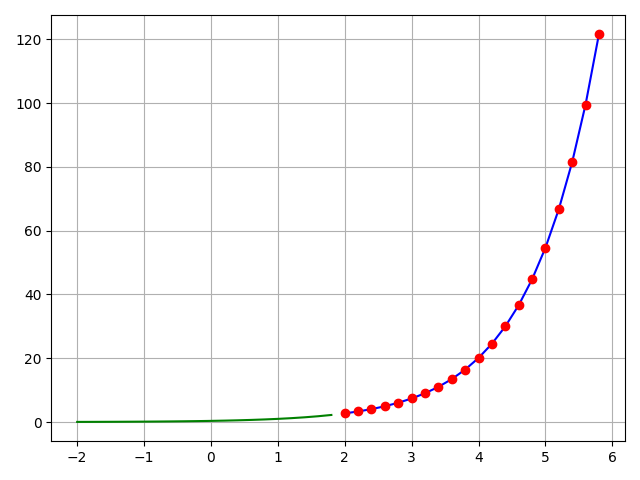


Рисунок 3 – Исходная функция и ее прогноз при M = 100, ():

Вектор весовых коэффициентов при M = 100 равен

Сравнивая результаты прогноза при различном количестве эпох, следует отметить, что его качество неудовлетворительно примерно до M = 20, а затем быстро улучшается, и при M > 35 прогнозные значения практически совпадают с точными в пределах графичекого изображения.

# Выводы

В ходе выполнения лабораторной работы, мною было исследованно функционирование простейшей нейронной сети (НС) на базе нейрона с нелинейной функцией активации. Также она была обучена по правилу Видроу-Хоффа. Результаты совпали с ожидаемыми, что говорит о корректности работы программы.

Приложение А. Исходный код программы

*Файл main.py*

import numpy as np  
import math  
import matplotlib.pyplot as plt  
  
  
class NNPredictor*()*:  
 def \_\_init\_\_*(*self, M, a, b, eta*)*:  
 self.\_M = M  
 self.\_a = a  
 self.\_b = b  
 self.\_eta = eta  
  
 self.\_learnFunction = *[] # Список для обучающейся функции* self.\_mainFunction = *[] # Список для функции по условию* self.\_N = 20  
  
 self.\_windowSize = abs*(*self.\_a*)* + abs*(*self.\_b*)* self.\_w = np.zeros*(*self.\_windowSize + 1*)* self.\_epsilon = 1  
 self.\_delta = 0  
 self.\_k = 0  
  
 self.\_era = *[]* self.\_error = *[]  
  
  
 # Считаем net* def net*(*self, windowSize, x, w*)*:  
 net = *(*sum*(*w*[*i + 1*]* \* x*[*i*]* for i in range*(*windowSize*)))* return net  
  
 *# Получаем интервал с шагом* def getX*(*self, a, b*)*:  
 vectorX = *[]* dx = *(*b - a*)* / self.\_N  
  
 for i in range*(*self.\_N*)*:  
 vectorX.append*(*a + i \* dx*)* return vectorX  
  
 *# Получаем значения функции на интервале* def getY*(*self, a, b*)*:  
 vectorY = *[]* vectorX = self.getX*(*a, b*)* for i in range*(*self.\_N*)*:  
 vectorY.append*(*math.exp*(*vectorX*[*i*]* - 1*))  
 #vectorY.append(0.5 \* math.sin(0.5 \* vectorX[i]) - 0.5)* return vectorY  
  
 *# Алгоритм обучения* def training\_mode*(*self*)*:  
 self.\_learnFunction = *[*0*]* \* self.\_N  
 self.\_mainFunction = self.getY*(*self.\_a, self.\_b*)* plt.plot*(*self.getX*(*self.\_a, self.\_b*)*, self.getY*(*self.\_a, self.\_b*)*, **'go-'***)* plt.grid*(*True*)* plt.show*()* self.\_w = np.zeros*(*self.\_windowSize + 1*)* while self.\_k < self.\_M:  
 for q in range*(*self.\_windowSize*)*:  
 self.\_learnFunction*[*q*]* = self.\_mainFunction*[*q*]* for i in range*(*self.\_windowSize, self.\_N*)*:  
 self.\_learnFunction*[*i*]* = self.net*(*self.\_windowSize, self.\_mainFunction*[*i - self.\_windowSize: i*]*,  
 self.\_w*)* self.\_delta = self.\_mainFunction*[*i*]* - self.\_learnFunction*[*i*]* for j in range*(*self.\_windowSize*)*:  
 self.\_w*[*j + 1*]* += self.\_eta \* self.\_delta \* self.\_mainFunction*[*i - self.\_windowSize + j*]* self.\_epsilon = sum*(  
 (*self.\_mainFunction*[*index*]* - self.\_learnFunction*[*index*])* \*\* 2 for index in range*(*self.\_N*))* self.\_epsilon = math.sqrt*(*self.\_epsilon*)* self.\_era.append*(*self.\_k*)* self.\_error.append*(*self.\_epsilon*)* self.\_k += 1  
  
  
 *# Функция для прогнозирования* def test\_func*(*self*)*:  
 vecFX = self.getX*(*self.\_a, self.\_b*)* vecFT = self.getY*(*self.\_a, self.\_b*)* vectorX = self.getX*(*self.\_b, 2 \* self.\_b - self.\_a*)* vectorY = self.getY*(*self.\_b, 2 \* self.\_b - self.\_a*)* testFunction = *[*0*]* \* *(*self.\_N + self.\_windowSize*)* TF = *[*0*]* \* self.\_N  
  
 for i in range*(*self.\_N - self.\_windowSize, self.\_N*)*:  
 testFunction*[*i - *(*self.\_N - self.\_windowSize*)]* = self.\_learnFunction*[*i*]* for j in range*(*self.\_windowSize, len*(*vectorY*)* + self.\_windowSize*)*:  
 vectorTestFunction = *[*0*]* \* self.\_windowSize  
 for k in range*(*self.\_windowSize*)*:  
 vectorTestFunction*[*k*]* = testFunction*[*k + j - self.\_windowSize*]* testFunction*[*j*]* = self.net*(*self.\_windowSize, vectorTestFunction, self.\_w*)* for i in range*(*self.\_windowSize, len*(*testFunction*))*:  
 TF*[*i - self.\_windowSize*]* = testFunction*[*i*]* plt.plot*(*vecFX, vecFT, **'g-'***)* plt.plot*(*vectorX, vectorY, **'b-'***)* plt.plot*(*vectorX, TF, **'ro'***)* plt.grid*(*True*)* plt.show*()* def results*(*self*)*:  
 return self.\_w, self.\_epsilon  
  
  
  
if \_\_name\_\_ == **'\_\_main\_\_'**:  
 obj = NNPredictor*(*25, -2, 2, 0.01*)* obj.training\_mode*()* obj.test\_func*()* w, e = obj.results*()* print*(*w*)* print*(*e*)* obj2 = NNPredictor*(*100, -2, 2, 0.01*)* obj2.training\_mode*()* obj2.test\_func*()* w2, e2 = obj2.results*()* print*(*w2*)* print*(*e2*)*