Министерство образования Республики Беларусь

Учреждение образования

«Брестский Государственный технический университет»

Кафедра ИИТ

**Лабораторная работа №2**

По дисциплине «Модели решения задач в интеллектуальных системах»

Тема: «SLP. Прогнозирование»

**Выполнил:**

Студент 3 курса

Группы ИИ-21

Карагодин Д. Л.

**Проверил:**

Туз И. С.

Брест 2023

**Цель:** Изучить обучение и функционирование ИНС при решении задач прогнозирования.

**Ход работы**

**Вариант 5**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № варианта | a | b | d | Кол-во входов ИНС |
| 5 | 1 | 9 | 0.5 | 4 |

**Код программы:**

import numpy as np

from math import sin

from matplotlib import pyplot as plt

f = lambda x: 1 \* sin(9 \* x) + 0.5

class Network:

    def \_\_init\_\_(self, weights=np.random.rand(4, 1)/5, T=0.1, a=0.0046) -> None:

        self.weights = weights

        self.T = T

        self.a = a

    def predict(self, input: list) -> float:

        return np.sum(input @ self.weights) - self.T

    def reset(self):

        self.weights = np.random.rand(4, 1)/5

        self.T = 0.1

        self.a = 0.0046

    def trainConst(self, inputs, targets,epoch = 1000) -> list:

        global E\_arr

        for i in range(epoch):

            for index, input in enumerate(inputs):

                prediction = self.predict(input)

                for j, w in enumerate(self.weights):

                    w[0] = w[0] - self.a \* (prediction - targets[index]) \* input[j]

                self.T = self.T + self.a \* (prediction - targets[index])

                #print(f"step:{index+1}, weights={self.weights}")

            E,e\_arr=0,[]

            for input, target in zip(inputs, targets):

                e\_arr.append((NN.predict(input) - target) \*\* 2)

            E = 0.5 \* np.sum(e\_arr)

            print("Эпоха:",i,"Квадратичная ошибка:", E)

            E\_arr.append(E)

    def trainAdaptive(self, inputs, targets,epoch = 1000) -> list:

        global E\_arr

        aplha = 0

        for i in range(epoch):

            for index, input in enumerate(inputs):

                prediction = self.predict(input)

                for j, w in enumerate(self.weights):

                    w[0] = w[0] - aplha \* (prediction - targets[index]) \* input[j]

                self.T = self.T + aplha \* (prediction - targets[index])

                #print(f"step:{index+1}, weights={self.weights}")

            E,e\_arr=0,[]

            for input, target in zip(inputs, targets):

                e\_arr.append((NN.predict(input) - target) \*\* 2)

            E = 0.5 \* np.sum(e\_arr)

            aplha = self.a\*E

            # Адаптивное изменение шага обучения

            print("Эпоха:",i,"Квадратичная ошибка:", E,"Альфа",aplha)

            E\_arr.append(E)

    # АЛЬФА НЕ ВЫШЕ a=0.0046

    def trainBatchConst(self,inputs,targets,epoch=1000) -> list:

        global E\_arr

        # Пакетное обучение

        for i in range(epoch):

            # Рассчитываем предсказания модели

            predictions = np.array([self.predict(i) for i in inputs])

            # Рассчитываем ошибку

            error = predictions - targets

            delta= np.zeros\_like(self.weights.transpose())

            for j in range(error.size):

                #print(delta,error[j],inputs[j])

                delta = delta + (error[j]\*inputs[j])

            E2 = 0

            for err in error:

                E2+=err\*\*2

            E2/=2

            E\_arr.append(E2)

            print(i,E2)

            self.weights = self.weights - self.a\*delta.transpose()

            self.T = self.T + self.a \* np.sum(error)

    # АЛЬФА НЕ ВЫШЕ a=0.0046

    def trainBatchAdapt(self,inputs,targets,epoch=1000) -> list:

        global E\_arr

        # Пакетное обучение

        for i in range(epoch):

            # Рассчитываем предсказания модели

            predictions = np.array([self.predict(i) for i in inputs])

            # Рассчитываем ошибку

            error = predictions - targets

            delta= np.zeros\_like(self.weights.transpose())

            for j in range(error.size):

                #print(delta,error[j],inputs[j])

                delta = delta + (error[j]\*inputs[j])

            E2 = 0

            for err in error:

                E2+=err\*\*2

            E2/=2

            alpha = min(self.a\*E2,0.0046)

            E\_arr.append(E2)

            print(i,E2,alpha)

            self.weights = self.weights - alpha \* delta.transpose()

            self.T = self.T + alpha \* np.sum(error)

# ДЛЯ 100 ТОЧЕК

x\_100 = [el for el in np.arange(0, 1, 0.01)] # Разбиение на 100 точек

data = [f(el) for el in x\_100] # Заполнение массива реальных значений функции

inputs = np.array([data[i-4:i] for i in range(4, len(data))]) # Создание массива входных значений

targets = data[4:] # Создание массива целевых значений

# ДЛЯ НЕИЗВЕСТНЫХ ТОЧЕК

x\_60 = [el for el in np.arange(0.9, 1.5, 0.01)] # Разбиение на 60 точек

data2 = [f(el) for el in x\_60] # Заполнение массива реальных значений функции

inputs2 = np.array([data2[i-4:i] for i in range(4, len(data2))])

targets2 = data2[4:] # Создание массива целевых значений

epoch=200

NN = Network()

func = {1: NN.trainConst, 2: NN.trainAdaptive, 3: NN.trainBatchConst, 4: NN.trainBatchAdapt}

for i in range(4):

    E\_arr = []

    func[i+1](inputs, targets,epoch)

    result = []

    for input in inputs2:

        result.append(NN.predict(input))

    plt.plot(range(epoch), E\_arr)

    plt.show()

    plt.plot(x\_100, data, x\_60[4:], result, '--')

    plt.show()

    NN.reset()

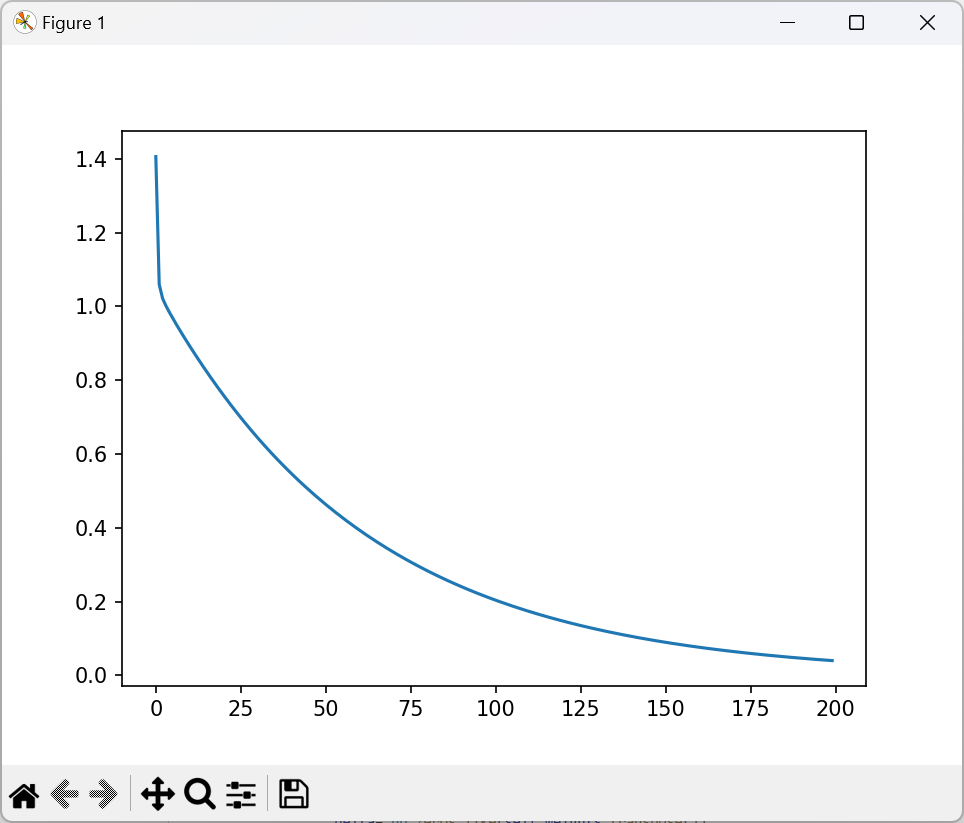
**Результат программы:**

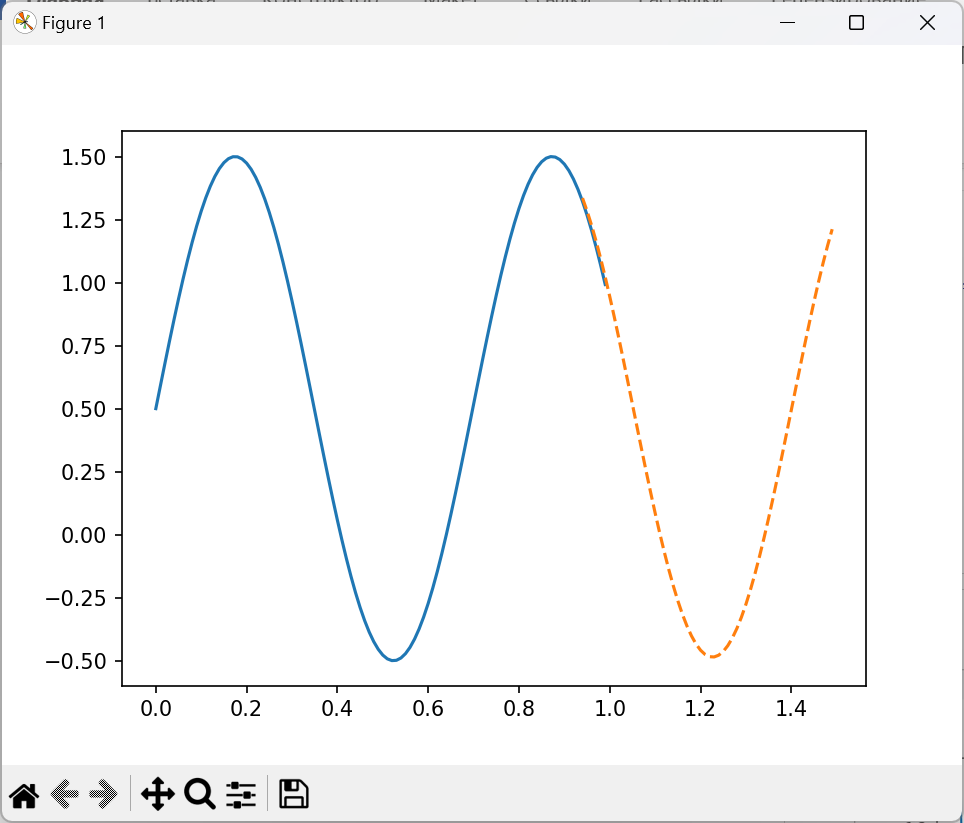
**Таблица сравнения:**

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Кол-во эпох** | **MSE** | **Время выполнения (ms)** |
| **Online-Const** | 200 | 0.0533684719124661 | 220.484 |
| **Online-Adapt** | 200 | 0.2197301818780398 | 218.533 |
| **Batch-Const** | 200 | 0.0607135029056706 | 130.461 |
| **Batch-Adapt** | 200 | 0.1700427335081447 | 166.204 |

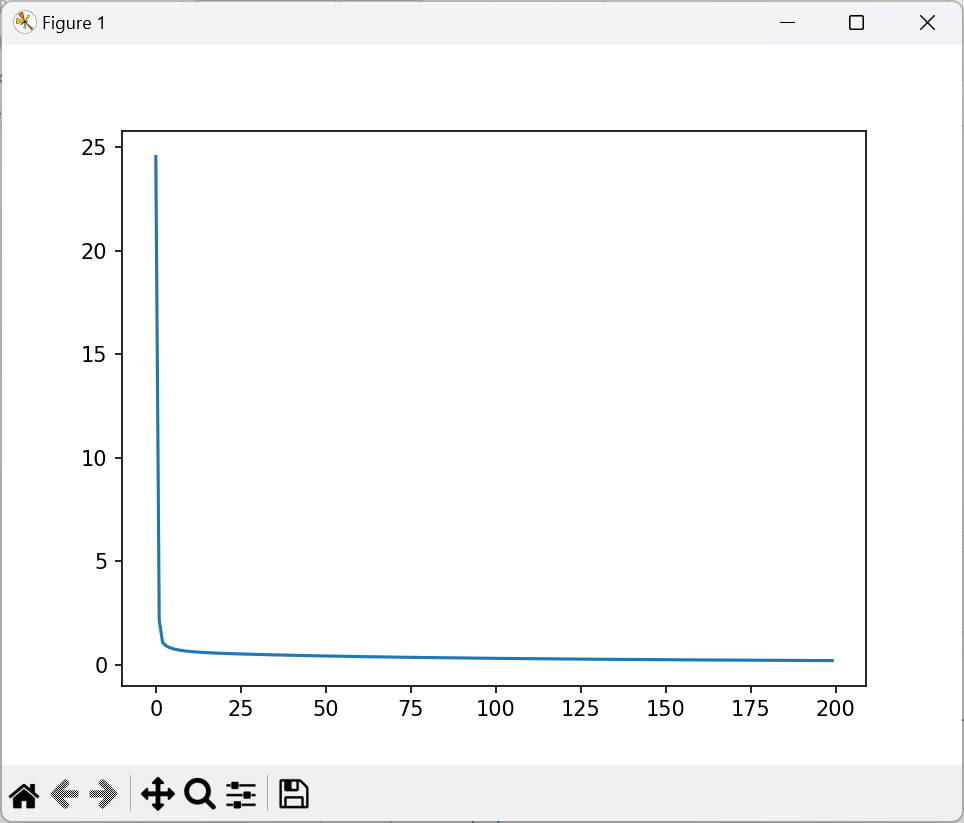
В графике прогноза прерывистая линия отвечает за прогноз дальнейших значений, плоская линия отвечает за обучающую выборку.

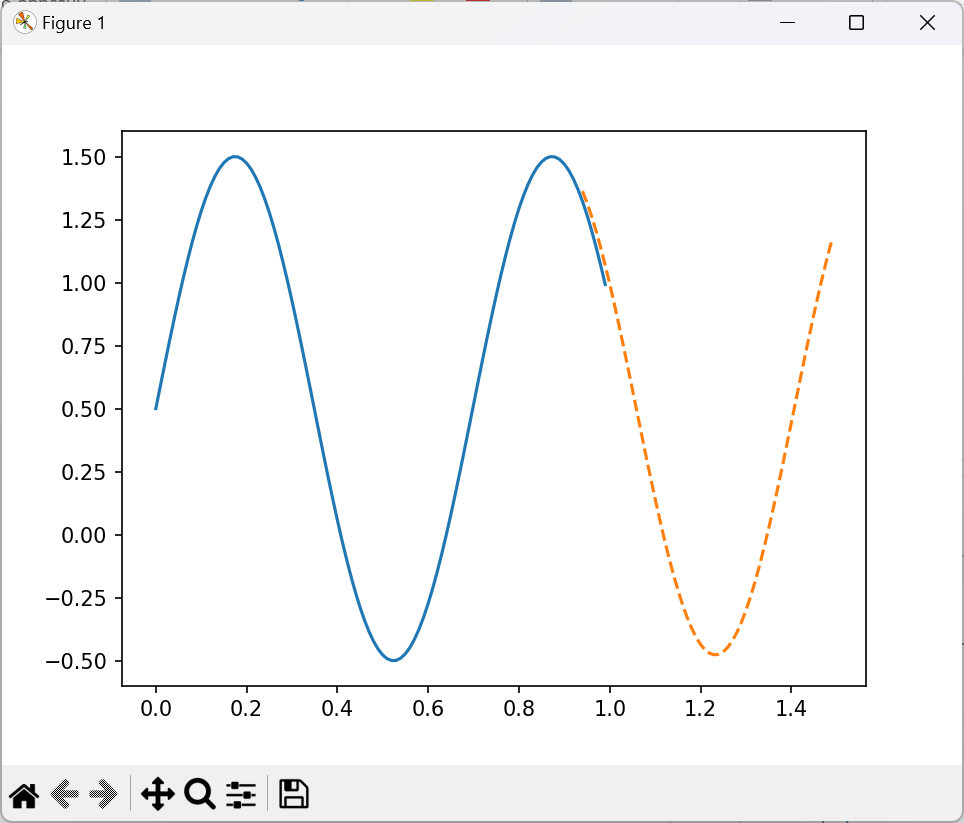
**График изменения ошибок и прогноза значений (Online-Const):**



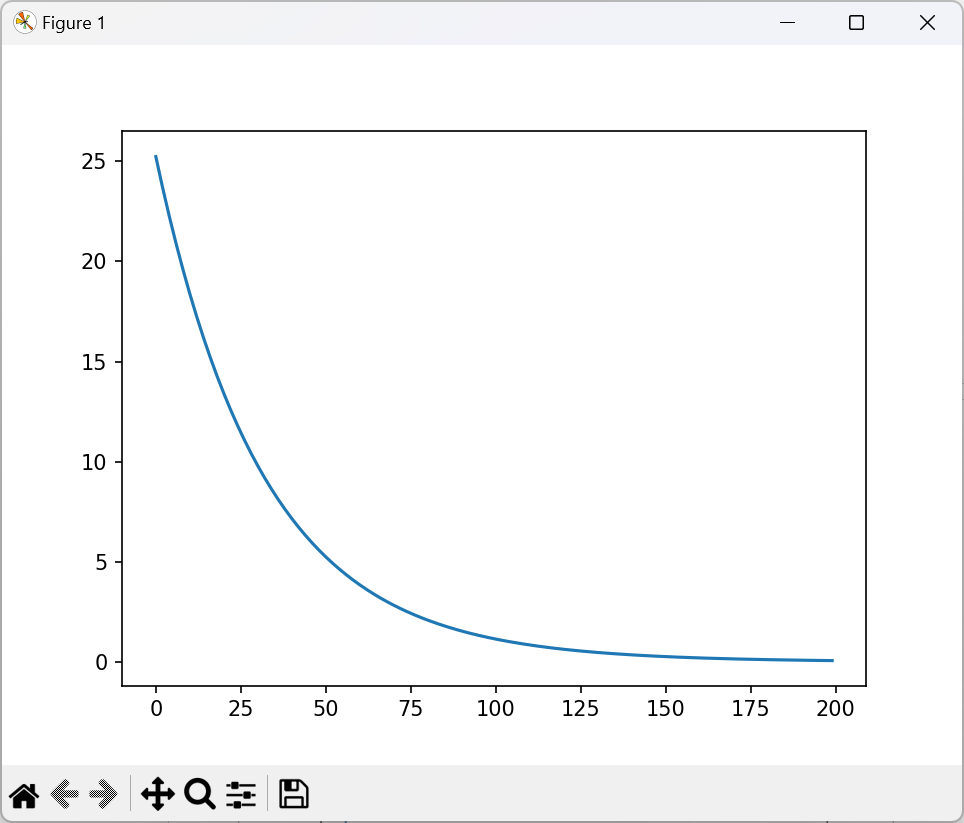


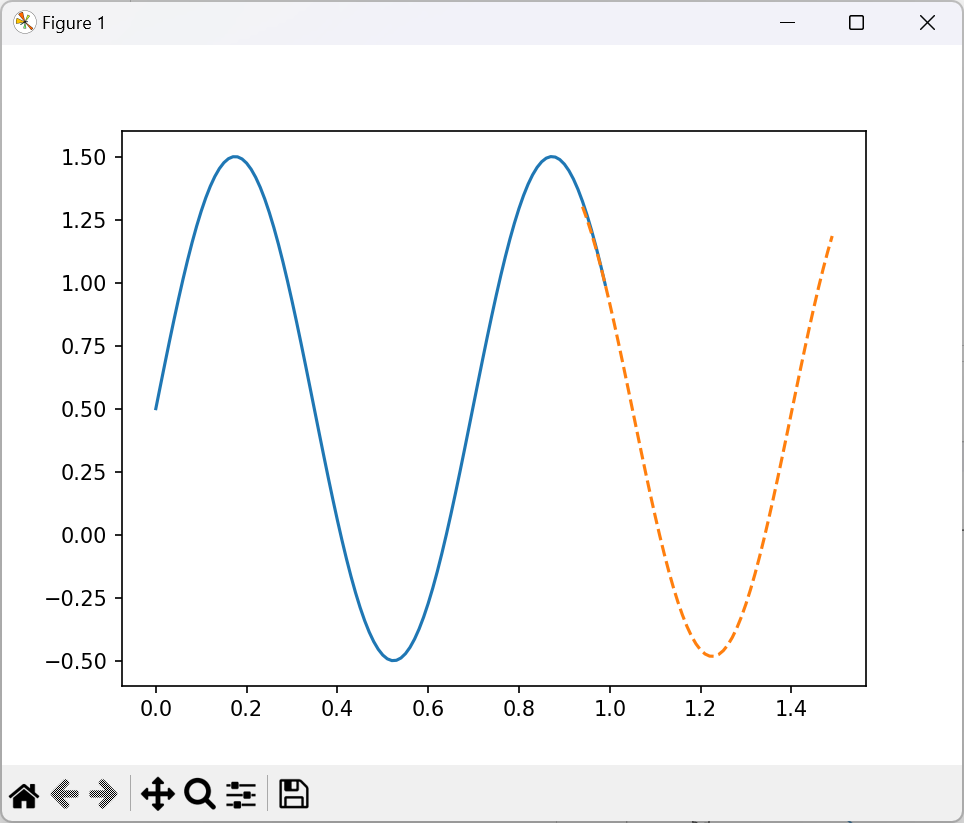
**График изменения ошибок и прогноза значений (Online-Adapt):**



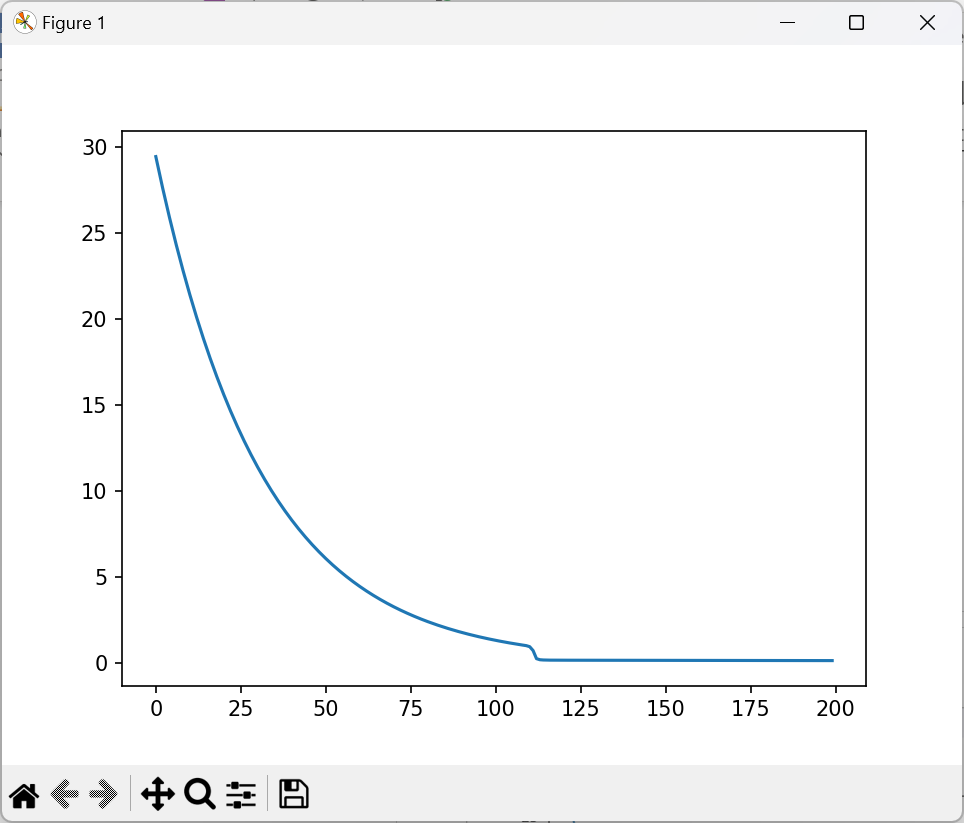


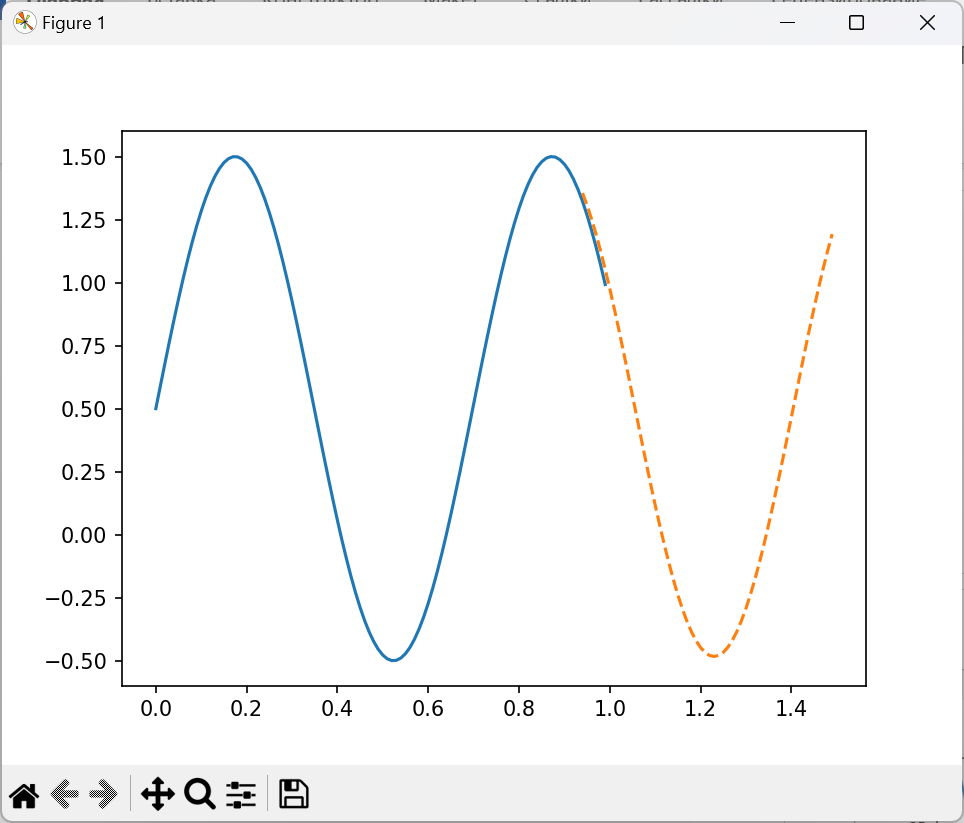
**График изменения ошибок и прогноза значений (Batch-Const):**





**График изменения ошибок и прогноза значений (Batch-Adapt):**





**Вывод:** изучил обучение и функционирование ИНС при решении задач прогнозирования.