Министерство образования Республики Беларусь

Учреждение образования

«Брестский Государственный технический университет»

Кафедра ИИТ

**Лабораторная работа №5**

По дисциплине «Модели решения задач в интеллектуальных системах»

Тема: «MLP. Прогнозирование»

**Выполнил:**

Студент 3 курса

Группы ИИ-23

Романюк А. П.

**Проверил:**

Туз И. С.

Брест 2024

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

Написать нейронную сеть(multilayer perceptron c одним скрытым слоем) для

решения задачи прогнозирования функции:

Изображение выглядит как снимок экрана, черный

Автоматически созданное описание

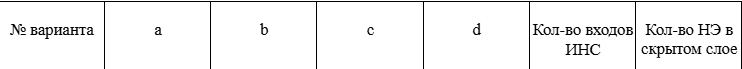
Обучить сеть с использованием константного и адаптивного шага обучения, online-

learning и batch-learning. Результаты для каждого варианта сети занести в таблицу(

test error, количество эпох, время обучения и тд).

В обучении использовать алгоритм обратного распространения ошибки и batch\_size > 1

**Вариант: 9**

****

****

**Ход работы**

import numpy as np  
import matplotlib.pyplot as plt  
import pandas as pd  
import time  
class MLP:  
 def \_\_init\_\_(self, input\_size, hide\_size, output\_size, alpha):  
 self.input\_size = input\_size  
 self.output\_size = output\_size  
 self.hide\_size = hide\_size  
 self.Wh = np.random.uniform(size=(input\_size, hide\_size))  
 self.Wo = np.random.uniform(size=(hide\_size, output\_size))  
 self.Th = np.random.uniform(size=(1, hide\_size))  
 self.To = np.random.uniform(size=(1, output\_size))  
 self.alpha = alpha  
 def tanh(self, x):  
 return np.tanh(x)  
 def tanh\_derivative(self, x):  
 return 1 - np.tanh(x) \*\* 2  
 def create\_dataset(self, noise\_percentage=0.1):  
 func = lambda a, b, c, d, x: a \* np.cos(b \* x) + c \* np.sin(d \* x)  
 x = np.arange(0, 15, 0.1)  
 y = func(0.1, 0.3, 0.08, 3, x)  
 y = np.array([y[i - self.input\_size: i] for i in range(self.input\_size, len(y))])  
 e = np.array([y[i + self.input\_size][0] for i in range(len(y) - self.input\_size)])  
 # Calculate the number of noisy points based on the noise\_percentage  
 num\_noisy\_points = int(len(e) \* (noise\_percentage))  
 noisy\_indices = np.random.choice(len(e), num\_noisy\_points, replace=False)  
 # Apply noise to the selected indices of e  
 noise = np.random.uniform(-0.1, 0.1, size=num\_noisy\_points) # Adjust the range of noise  
 e[noisy\_indices] += noise # Add noise to the selected indices  
 return y[:len(y) - self.input\_size], e, x[:len(y) - self.input\_size]  
 def train(self, x, e, batch\_size,mode="ba"):  
 if mode == "ba":  
 num\_samples = x.shape[0]  
 for start in range(0, num\_samples, batch\_size):  
 end = start + batch\_size  
 x\_batch = x[start:min(end, num\_samples)]  
 e\_batch = e[start:min(end, num\_samples)].reshape(-1, self.output\_size)  
 yi = np.dot(x\_batch, self.Wh) + self.Th  
 # Calculate the hidden layer output  
 yh = self.tanh(np.dot(x\_batch, self.Wh) + self.Th)  
 # Output layer calculation  
 yo = np.dot(yh, self.Wo) + self.To  
  
 error = e\_batch - yo  
 deltao = error \* self.tanh\_derivative(yo)  
 self.learning\_rate\_output = np.sum(error\*\*2\*self.tanh\_derivative(yo)) / \  
 (1 + np.sum(yh\*\*2) \* np.sum(error\*\*2\*self.tanh\_derivative(yo)\*\*2))  
 hidden\_error = deltao.dot(self.Wo.T)  
 self.learning\_rate\_hidden = np.sum(hidden\_error\*\*2\*self.tanh\_derivative(yh)) / \  
 (1 + (np.sum(yi\*\*2)) \* np.sum(hidden\_error\*\*2\*self.tanh\_derivative(yh)\*\*2))  
 self.learning\_rate\_hidden = self.alpha  
 deltah = hidden\_error \* self.tanh\_derivative(yh)  
 self.Wo += yh.T.dot(deltao) \* self.learning\_rate\_output / x\_batch.shape[0]  
 self.To += np.sum(deltao, axis=0, keepdims=True) \* self.learning\_rate\_output / x\_batch.shape[0]  
 self.Wh += x\_batch.T.dot(deltah) \* self.learning\_rate\_hidden / x\_batch.shape[0]  
 self.Th += np.sum(deltah, axis=0, keepdims=True) \* self.learning\_rate\_hidden / x\_batch.shape[0]  
 elif mode == "b":  
 num\_samples = x.shape[0]  
 for start in range(0, num\_samples, batch\_size):  
 end = start + batch\_size  
 x\_batch = x[start:min(end, num\_samples)]  
 e\_batch = e[start:min(end, num\_samples)].reshape(-1, self.output\_size)  
 yh = self.tanh(np.dot(x\_batch, self.Wh) + self.Th)  
 # Output layer calculation  
 yo = np.dot(yh, self.Wo) + self.To  
 error = e\_batch - yo  
 deltao = error \* self.tanh\_derivative(yo)  
 hidden\_error = deltao.dot(self.Wo.T)  
 deltah = hidden\_error \* self.tanh\_derivative(yh)  
 self.Wo += yh.T.dot(deltao) \* self.alpha / x\_batch.shape[0]  
 self.To += np.sum(deltao, axis=0, keepdims=True) \* self.alpha / x\_batch.shape[0]  
 self.Wh += x\_batch.T.dot(deltah) \* self.alpha / x\_batch.shape[0]  
 self.Th += np.sum(deltah, axis=0, keepdims=True) \* self.alpha / x\_batch.shape[0]  
 def test(self, x):  
 yh = self.tanh(np.dot(x, self.Wh) + self.Th)  
 yo = np.dot(yh, self.Wo) + self.To  
 return yo.flatten()  
Изображение выглядит как текст, линия, График, диаграмма

Автоматически созданное описание def mean\_squared\_error(self, true\_values, predictions):  
 return np.mean((true\_values - predictions) \*\* 2)  
 def draw\_test\_on\_interval(self, start, end,e, step=0.1):  
 x\_new = np.arange(start, end, step)  
 func = lambda a, b, c, d, x: a \* np.cos(b \* x) + c \* np.sin(d \* x)  
 y\_new = func(0.1, 0.3, 0.08, 3, x\_new)  
 y\_input = np.array([y\_new[i - self.input\_size: i] for i in range(self.input\_size, len(y\_new))])  
 predictions = self.test(y\_input)  
 x\_new\_cropped = x\_new[self.input\_size:]  
 print(e[:10])  
 print(y\_new[self.input\_size:20])  
 plt.figure(figsize=(12, 6))  
 plt.plot(x\_new\_cropped, y\_new[self.input\_size:], label='Original Function', color='blue')  
 plt.plot(x\_new\_cropped, predictions, label='Predictions', color='red', linestyle='dashed')  
 plt.plot(x\_new\_cropped[:len(e)], e, label='noise', color='green', linestyle='dashed')  
 plt.xlabel('X values')  
 plt.ylabel('Function value')  
 plt.title('Original Function vs. Predictions')  
 plt.legend()  
 plt.grid()  
 plt.show()  
def run\_model(mode, input\_size=10, hide\_size=4, output\_size=1, alpha=0.2, batch\_size=10, epochs=10000):  
 model = MLP(input\_size=input\_size, hide\_size=hide\_size, output\_size=output\_size, alpha=alpha)  
 y, e, x\_values = model.create\_dataset()  
 start\_time = time.time()  
 for epoch in range(epochs):  
 model.train(y, e, batch\_size, mode=mode)  
 if (epoch + 1) % 100 == 0:  
 predictions = model.test(y)  
 mse = model.mean\_squared\_error(e, predictions)  
 print(f'Epoch {epoch + 1}, Mean Squared Error: {mse}')  
 training\_time = time.time() - start\_time  
 predictions = model.test(y)  
 mse = model.mean\_squared\_error(e, predictions)  
 model.draw\_test\_on\_interval(0,50,e)  
 return mse, training\_time, epochs  
def show\_results\_table(results):   
 df = pd.DataFrame(results, columns=['Model Type', 'Test Error (MSE)', 'Epochs', 'Training Time (s)'])  
 print(df)  
Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание# Run and collect results for different models  
results = []  
mse, training\_time, epochs = run\_model(mode="ba")  
results.append(["Batch Adapt", mse, epochs, training\_time])  
mse, training\_time, epochs = run\_model(mode="b", batch\_size=10)

results.append(["Batch Static", mse, epochs, training\_time])  
mse, training\_time, epochs = run\_model(mode="b", batch\_size=1)  
results.append(["Linear Static", mse, epochs, training\_time])  
mse, training\_time, epochs = run\_model(mode="ba", batch\_size=1)  
results.append(["Linear Adapt", mse, epochs, training\_time])  
# Show the table of results  
show\_results\_table(results)

**Вывод:** в ходе лабораторной работы я научился реализовывать однослойную нейронную сеть.