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-23

Романюк А. П.

**Проверил:**

Туз И. С.

Брест 2024

**Цель:** 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)

**Ход работы****Вариант 9**

****



import matplotlib.pyplot as plt  
import numpy as np  
import pandas as pd  
import time  
class NN:  
 def \_\_init\_\_(self, alpha=0.01, neurons=5):  
 self.alpha = alpha  
 self.neurons = neurons  
 self.W = np.random.rand(neurons)  
 self.T = np.zeros(1)  
 self.errors = []  
 def create\_dataset(self, y):  
 return np.array([y[i - self.neurons : i] for i in range(self.neurons, len(y))])  
 def adaptive\_alpha(self, x):  
 return 1/(1+sum(x\*\*2))  
 def a(self,X,E,xk):  
 a = 0  
 for x, e in zip(X,E):  
 sum = 0  
 for i in range(len(x)):  
 sum+=xk[i]\*x[i]  
 a+=(self.predict(x)-e)\*(1+sum)  
 return a  
 def adaptive\_batch\_alpha(self, X, E):  
 sum\_a = 0  
 sum\_b = 0  
 for x, e in zip(X,E):  
 sum\_a+=(self.predict(x)-e)\*self.a(X,E,x)  
 sum\_b+=(self.a(X,E,x))\*\*2  
 return sum\_a/sum\_b  
 def MSE(self, x, e):  
 return (1 / 2) \* ((self.predict(x) - e) \*\* 2)  
 def predict(self, x):  
 return self.W @ x - self.T  
 def train(self, X, E):  
 for x, e in zip(X[: 31 - self.neurons], E[self.neurons : 31]):  
 y = self.predict(x)  
 self.W += -self.alpha \* (y - e) \* x  
 self.T += self.alpha \* (y - e)  
 def adapt\_train(self, X, E):  
 for x, e in zip(X[: 31 - self.neurons], E[self.neurons: 31]):  
 y = self.predict(x)  
 self.W += -self.adaptive\_alpha(x) \* (y - e) \* x  
 self.T += self.adaptive\_alpha(x) \* (y - e)  
 def batch\_train(self, X, E, batch\_size):  
 num\_batches = len(X[: 31 - self.neurons]) // batch\_size + (  
 1 if len(X[: 31 - self.neurons]) % batch\_size != 0 else 0)  
 for batch\_index in range(num\_batches):  
 start = batch\_index \* batch\_size  
 end = min((batch\_index + 1) \* batch\_size, len(X[: 31 - self.neurons]))  
 W\_update = np.zeros\_like(self.W)  
 T\_update = np.zeros(1)  
 for x, e in zip(X[start:end], E[start+self.neurons:end+self.neurons]):  
 y = self.predict(x)  
 W\_update += (y - e) \* x  
 T\_update += (y - e)  
 self.W += -self.alpha \* W\_update  
 self.T += self.alpha \* T\_update  
 def adapt\_batch\_train(self, X, E, batch\_size):  
 num\_batches = len(X[: 31 - self.neurons]) // batch\_size + (  
 1 if len(X[: 31 - self.neurons]) % batch\_size != 0 else 0)  
 for batch\_index in range(num\_batches):  
 start = batch\_index \* batch\_size  
 end = min((batch\_index + 1) \* batch\_size, len(X[: 31 - self.neurons]))  
 W\_update = np.zeros\_like(self.W)  
 T\_update = np.zeros(1)  
 for x, e in zip(X[start:end], E[start+self.neurons:end+self.neurons]):  
 y = self.predict(x)  
 W\_update += (y - e) \* x  
 T\_update += (y - e)  
 self.W += -self.adaptive\_batch\_alpha(X[start:end], E[start+self.neurons:end+self.neurons]) \* W\_update  
 self.T += self.adaptive\_batch\_alpha(X[start:end], E[start+self.neurons:end+self.neurons]) \* T\_update  
 def test(self, epoch: int, X, E):  
 values = []  
 error = np.mean([self.MSE(x, e) for x, e in zip(X[31 - self.neurons :], E[31:])])  
 for x, e in zip(X[0 : len(E) - self.neurons], E[self.neurons : len(E)]):  
 values.append(self.predict(x))  
 print(f"Epoch: {epoch}: e={error}")  
 return values  
 def set\_MSE(self, X, E):  
 self.errors.append(np.mean([self.MSE(x, e) for x, e in zip(X[31 - self.neurons :], E[31:])]))  
def collect\_results(model\_name, model, epochs, start\_time):  
 results = {  
 "Model": model\_name,  
 "Test Error": model.errors[-1], # Last error value  
 "Epochs": epochs,  
 "Training Time (s)": time.time() - start\_time # Time in seconds  
 }  
 return results  
# Create an empty list to store results  
results\_table = []  
if \_\_name\_\_ == "\_\_main\_\_":  
 # best alpha=0.2 neurons=5  
 neurons = 5  
 lin\_model = NN(alpha=0.05, neurons=neurons)  
 lin\_model\_adapt = NN(alpha=0.05, neurons=neurons)  
 batch\_model = NN(alpha=0.05, neurons=neurons)  
 batch\_model\_adapt = NN(alpha=0.05, neurons=neurons)  
 arrange = np.arange(0, 5, 0.1)  
 func = lambda a, b, d, x: a \* np.sin(b \* x) + d  
 y = func(1, 8, 0.3, np.arange(0, 5, 0.1))  
 x = lin\_model.create\_dataset(y)  
 # Linear model  
 print("\nLinear model\n")  
 start\_time = time.time()  
 last\_epoch = -1  
 for epoch in range(1, 100):  
 lin\_model.train(x, y)  
 lin\_model.set\_MSE(x, y)  
 lin\_model.test(epoch, x, y)  
 if lin\_model.errors[-1] <= 1e-5:  
 last\_epoch = epoch  
 break  
 if last\_epoch == -1:  
 last\_epoch = 99  
 print("Training results")  
 lin\_values = lin\_model.test(last\_epoch, x, y)  
 # Collect results for the linear model  
 results\_table.append(collect\_results("Linear Model", lin\_model, last\_epoch, start\_time))  
 # Linear model adaptive  
 print("\nLinear model adapt\n")  
 start\_time = time.time()  
 last\_epoch = -1  
 for epoch in range(1, 100):  
 lin\_model\_adapt.adapt\_train(x, y)  
 lin\_model\_adapt.set\_MSE(x, y)  
 lin\_model\_adapt.test(epoch, x, y)  
 if lin\_model\_adapt.errors[-1] <= 1e-5:  
 last\_epoch = epoch  
 break  
 if last\_epoch == -1:  
 last\_epoch = 99  
 print("Training results")  
 lin\_values\_adapt = lin\_model\_adapt.test(last\_epoch, x, y)  
 # Collect results for the linear model adaptive  
 results\_table.append(collect\_results("Linear Model Adaptive", lin\_model\_adapt, last\_epoch, start\_time))  
 # Batch model  
 print("\nBatch model\n")  
 start\_time = time.time()  
 last\_epoch = -1  
 for epoch in range(1, 100):  
 batch\_model.batch\_train(x, y, 10)  
 batch\_model.set\_MSE(x, y)  
 batch\_model.test(epoch, x, y)  
 if batch\_model.errors[-1] <= 1e-5:  
 last\_epoch = epoch  
 break  
 if last\_epoch == -1:  
 last\_epoch = 99  
 print("Training results")  
 batch\_values = batch\_model.test(last\_epoch, x, y)  
 # Collect results for the batch model  
 results\_table.append(collect\_results("Batch Model", batch\_model, last\_epoch, start\_time))  
 # Batch model adaptive  
 print("\nBatch model adapt\n")  
 start\_time = time.time()  
 last\_epoch = -1  
 for epoch in range(1, 100):  
 batch\_model\_adapt.adapt\_batch\_train(x, y, 10)  
 batch\_model\_adapt.set\_MSE(x, y)

batch\_model\_adapt.test(epoch, x, y)  
 if batch\_model\_adapt.errors[-1] <= 1e-5:  
 last\_epoch = epoch  
 break  
 if last\_epoch == -1:  
 last\_epoch = 99  
 print("Training results")  
 batch\_values\_adapt = batch\_model\_adapt.test(last\_epoch, x, y)  
 # Collect results for the batch model adaptive  
 results\_table.append(collect\_results("Batch Model Adaptive", batch\_model\_adapt, last\_epoch, start\_time))  
 # Convert results to a pandas DataFrame  
 df = pd.DataFrame(results\_table)  
 # Print the table  
 print("\nTraining Results Table:")  
 print(df)  
 # Optionally, save the table to a CSV file  
 df.to\_csv("training\_results.csv", index=False)  
 plt.plot(range(1, len(lin\_model.errors) + 1), lin\_model.errors, label="Lin Model")  
 plt.plot(range(1, len(lin\_model\_adapt.errors) + 1), lin\_model\_adapt.errors, label="Lin Model Adapt")  
 plt.plot(range(1, len(batch\_model.errors) + 1), batch\_model.errors, label="Batch Model")  
 plt.plot(range(1, len(batch\_model\_adapt.errors) + 1), batch\_model\_adapt.errors, label="Batch Model Adapt")  
 plt.xlabel("Epoch")  
 plt.ylabel("Error")  
 plt.legend() # Optional, to add labels for each line  
 plt.savefig("NN.png")  
 plt.clf()  
 plt.plot(np.arange(0, 5, 0.1), func(1, 8, 0.3, np.arange(0, 5, 0.1)), label="Original Function")  
 plt.plot(np.arange(neurons / 10, 5, 0.1), lin\_values, label="Prediction (Linear)")  
 plt.plot(np.arange(neurons / 10, 5, 0.1), lin\_values\_adapt, label="Prediction (Linear, Adaptive)")  
 plt.plot(np.arange(neurons / 10, 5, 0.1), batch\_values, label="Prediction (Batch)")  
 plt.plot(np.arange(neurons / 10, 5, 0.1), batch\_values\_adapt, label="Prediction (Batch, Adaptive)")  
 plt.xlabel("X (Input Value)")  
 plt.ylabel("Y (Output Value)")  
 plt.title("Comparison of Original Function and Neural Network Predictions")  
 plt.legend(loc="upper center")  
 plt.savefig("graph.png")

**Изображение выглядит как текст, снимок экрана, линия, График

Автоматически созданное описание**Изображение выглядит как текст, линия, снимок экрана, диаграмма

Автоматически созданное описание

**Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание**

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