Министерство образования Республики Беларусь

Учреждение образования

«Брестский Государственный технический университет»

Кафедра ИИТ

**Лабораторная работа №3**

По дисциплине «Модели решения задач в интеллектуальных системах»

Тема: «SLP.Классификация»

**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 3**

****

import numpy as np  
import matplotlib.pyplot as plt  
import random  
import pandas as pd  
import time  
class NN:  
 def \_\_init\_\_(self,alpha,n\_samples,noise\_ratio):  
 self.noise\_ratio = noise\_ratio  
 self.n\_samples = n\_samples//2  
 self.alpha = alpha  
 #self.W = np.random.randn(2)  
 self.W = np.array([0.1,0.1])  
 self.T = np.zeros(1)  
 self.errors = []  
 self.acuracy = 0  
 def decision\_boundary(self,x):  
 return -2\*x + 12  
 def create\_dataset(self):  
 X = []  
 y = []  
 for i in range(self.n\_samples):  
 x = random.randrange(-100, 100)  
 y\_coord = random.randrange(-300, self.decision\_boundary(x))  
 X.append([x, y\_coord])  
 y.append(-1)  
 for i in range(self.n\_samples):  
 x = random.randrange(-100, 100)  
 y\_coord = random.randrange(self.decision\_boundary(x), 300)  
 X.append([x, y\_coord])  
 y.append(1)  
 n\_noise\_class0 = int(self.noise\_ratio \* self.n\_samples)  
 n\_noise\_class1 = int(self.noise\_ratio \* self.n\_samples)  
 for i in range(n\_noise\_class0):  
 x = random.randrange(-100, 100)  
 y\_coord = self.decision\_boundary(x) + random.randrange(0, 100)  
 X[i] = [x, y\_coord]  
 y[i] = 0  
 for i in range(n\_noise\_class1):  
 x = random.randrange(-100, 100)  
 y\_coord = self.decision\_boundary(x) - random.randrange(0, 100)  
 X[self.n\_samples + i] = [x, y\_coord]  
 y[self.n\_samples + i] = 1  
 return np.array(X), np.array(y)  
 def plot\_dataset(self, X, y):  
 x\_line = np.linspace(-100, 100, 400)  
 y\_line = self.decision\_boundary(x\_line)  
 plt.figure(figsize=(10, 6))  
 plt.scatter(X[y == 0][:, 0], X[y == 0][:, 1], color='blue', label='Class 0')  
 plt.scatter(X[y == 1][:, 0], X[y == 1][:, 1], color='red', label='Class 1')  
 plt.plot(x\_line, y\_line, color='black', linestyle='--', label='y = -2x + 12')  
 plt.title('Генерация датасета с шумами и разделением по линии y = -2x + 12')  
 plt.xlabel('X1')  
 plt.ylabel('X2')  
 plt.legend()  
 plt.grid(True)  
 plt.show()  
 def MSE(self, x, e):  
 return (1 / 2) \* ((self.predict(x) - e) \*\* 2)  
 def set\_MSE(self, X, E):  
 self.errors.append(np.mean([self.MSE(x, e) for x, e in zip(X, E)]))  
 def activate(self, x):  
 return 1 if x > 0 else -1  
 def predict(self, X):  
 linear\_output = self.W @ X + self.T  
 return self.activate(linear\_output)  
 def train(self, X, E):  
 for x,e in zip(X,E):  
 y = self.predict(x)  
 self.W += -self.alpha \* (y - e) \* x  
 self.T += self.alpha \* (y - e)  
 def adaptive\_alpha(self, x):  
 return 1/(1+sum(x\*\*2))  
 def adapt\_train(self, X, E):  
 for x, e in zip(X, E):  
 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) // batch\_size + (  
 1 if len(X) % batch\_size != 0 else 0)  
 for batch\_index in range(num\_batches):  
 W\_update = np.zeros\_like(self.W)  
 T\_update = np.zeros(1)  
 for x, e in zip(X, E):  
 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 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 adapt\_batch\_train(self, X, E, batch\_size):  
 num\_batches = len(X) // batch\_size + (  
 1 if len(X) % batch\_size != 0 else 0)  
 for batch\_index in range(num\_batches):  
 W\_update = np.zeros\_like(self.W)  
 T\_update = np.zeros(1)  
 for x, e in zip(X, E):  
 y = self.predict(x)  
 W\_update += (y - e) \* x  
 T\_update += (y - e)  
 self.W += -self.adaptive\_batch\_alpha(X, E) \* W\_update  
 self.T += self.adaptive\_batch\_alpha(X, E) \* T\_update  
 def test(self,epoch, X, E):  
 values = []  
 right = 0  
 error = np.mean([self.MSE(x, e) for x, e in zip(X, E)])  
 for x, e in zip(X, E):  
 if e == self.predict(x): right += 1  
 values.append(self.predict(x))  
 self.acuracy = right/(self.n\_samples\*2)\*100  
 print(f"Epoch: {epoch}: e={error}")  
 return values  
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  
 "Right Answers": model.acuracy  
 }  
 return results  
results\_table = []  
lin\_model = NN(alpha=0.05,n\_samples = 100,noise\_ratio = 0.1)  
lin\_model\_adapt = NN(alpha=0.05,n\_samples = 100,noise\_ratio = 0.1)  
batch\_model = NN(alpha=0.05,n\_samples = 100,noise\_ratio = 0.1)  
batch\_model\_adapt = NN(alpha=0.05,n\_samples = 100,noise\_ratio = 0.1)  
x,y = lin\_model.create\_dataset()  
# 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)  
results\_table.append(collect\_results("Batch Model Adaptive", batch\_model\_adapt, last\_epoch, start\_time))  
df = pd.DataFrame(results\_table)  
print("\nTraining Results Table:")  
print(df)

**Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание**

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