Министерство образования Республики Беларусь

Учреждение образования

«Брестский Государственный технический университет»

Кафедра ИИТ

**Лабораторная работа №6**

По дисциплине «Модели решения задач в интеллектуальных системах»

Тема: «MLP. Классификация»

**Выполнил:**

Студент 3 курса

Группы ИИ-23

Романюк А. П.

**Проверил:**

Туз И. С.

Брест 2024

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

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

решения задачи классификации:

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, метрики и тд)

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

**Изображение выглядит как текст, Шрифт, белый, логотип

Автоматически созданное описаниеВариант: 9**

**Ход работы**

import numpy as np  
import pandas as pd  
import time  
def create\_dataset():  
 data = pd.read\_csv('mushrooms.csv')  
 encoded\_data = pd.get\_dummies(data)  
 X = encoded\_data.drop(columns=['class\_e', 'class\_p'])  
 y = encoded\_data['class\_p']  
 return np.array(X), np.array(y)  
def train\_test\_split(X, y, test\_size=0.2, random\_state=None):  
 if random\_state is not None:  
 np.random.seed(random\_state)  
 test\_size = int(len(X) \* test\_size)  
 indices = np.arange(len(X))  
 np.random.shuffle(indices)  
 test\_indices = indices[:test\_size]  
 train\_indices = indices[test\_size:]  
 X\_train = X[train\_indices]  
 X\_test = X[test\_indices]  
 y\_train = y[train\_indices]  
 y\_test = y[test\_indices]  
 return X\_train, X\_test, y\_train, y\_test  
class MLP:  
 def \_\_init\_\_(self, input\_size, output\_size, hidden\_size, learning\_rate="adapt"):  
 self.input\_size = input\_size  
 self.output\_size = output\_size  
 self.hidden\_size = hidden\_size  
 self.learning\_rate = learning\_rate  
 self.weights\_input\_hidden = np.random.randn(self.input\_size, self.hidden\_size)  
 self.bias\_hidden = np.zeros((1, self.hidden\_size))  
 self.weights\_hidden\_output = np.random.randn(self.hidden\_size, self.output\_size)  
 self.bias\_output = np.zeros((1, self.output\_size))  
 def sigmoid(self, x):  
 return 1 / (1 + np.exp(-x))  
 def sigmoid\_derivative(self, x):  
 return x \* (1 - x)  
 def forward(self, inputs):  
 self.hidden\_input = np.dot(inputs, self.weights\_input\_hidden) + self.bias\_hidden  
 self.hidden\_output = self.sigmoid(self.hidden\_input)  
 self.output = np.dot(self.hidden\_output, self.weights\_hidden\_output) + self.bias\_output  
 return self.sigmoid(self.output)  
 def backward(self, inputs, targets, outputs):  
 targets = targets.reshape(-1, 1)  
 error = targets - outputs  
 delta\_output = error \* self.sigmoid\_derivative(outputs)  
 if self.learning\_rate == "adapt":  
 self.learning\_rate\_output = (4 \* np.sum((error \*\* 2 \* outputs \* (1 - outputs)), axis=0)) / \  
 (1 + np.sum(self.hidden\_output \*\* 2) + 1e-10) \* \  
 (np.sum((error \* outputs \* (1 - outputs)) \*\* 2, axis=0) + 1e-10)  
 h\_error = delta\_output.dot(self.weights\_hidden\_output.T)  
 self.learning\_rate\_hidden = (4 \* np.sum((h\_error \*\* 2 \* self.hidden\_output \* (1 - self.hidden\_output)), axis=0)) / \  
 (1 + np.sum(self.hidden\_input \*\* 2) + 1e-10) \* \  
 (np.sum((h\_error \* self.hidden\_output \* (1 - self.hidden\_output)) \*\* 2, axis=0) + 1e-10)  
 self.learning\_rate\_hidden = self.learning\_rate\_output  
 delta\_hidden = h\_error \* self.sigmoid\_derivative(self.hidden\_output)  
 self.weights\_hidden\_output += self.hidden\_output.T.dot(delta\_output) \* self.learning\_rate\_output  
 self.bias\_output += np.sum(delta\_output, axis=0, keepdims=True) \* self.learning\_rate\_output  
 self.weights\_input\_hidden += inputs.T.dot(delta\_hidden) \* self.learning\_rate\_hidden  
 self.bias\_hidden += np.sum(delta\_hidden, axis=0, keepdims=True) \* self.learning\_rate\_hidden  
 else:  
 delta\_hidden = delta\_output.dot(self.weights\_hidden\_output.T) \* self.sigmoid\_derivative(self.hidden\_output)  
 self.weights\_hidden\_output += self.hidden\_output.T.dot(delta\_output) \* self.learning\_rate  
 self.bias\_output += np.sum(delta\_output, axis=0, keepdims=True) \* self.learning\_rate  
 self.weights\_input\_hidden += inputs.T.dot(delta\_hidden) \* self.learning\_rate  
 self.bias\_hidden += np.sum(delta\_hidden, axis=0, keepdims=True) \* self.learning\_rate  
 def train(self, inputs, targets, epochs, batch\_size):  
 training\_losses = []  
 for epoch in range(epochs):  
 loss = 0  
 for i in range(0, len(inputs), batch\_size):  
 input\_data = inputs[i:i + batch\_size]  
 target\_data = targets[i:i + batch\_size]  
 output = self.forward(input\_data)  
 self.backward(input\_data, target\_data, output)  
 loss += np.mean(np.square(output - target\_data))  
 avg\_loss = loss / (len(inputs) // batch\_size)  
 training\_losses.append(avg\_loss)  
 print(f'Epoch {epoch + 1}, Loss: {avg\_loss}')  
 return training\_losses  
 def predict(self, inputs):  
 output = self.forward(inputs)  
 return np.round(output)  
# Create and prepare dataset  
X, y = create\_dataset()  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, 0.3, random\_state=1)  
input\_size = X\_train.shape[1]  
# List of configurations to try  
configurations = [  
 {'batch\_size': 1, 'learning\_rate': 0.1},  
 {'batch\_size': 256, 'learning\_rate': 0.1},  
 {'batch\_size': 1, 'learning\_rate': 'adapt'},  
 {'batch\_size': 256, 'learning\_rate': 'adapt'},  
]  
# List to store results  
results = []  
for config in configurations:  
 model = MLP(input\_size, 1, hidden\_size=5, learning\_rate=config['learning\_rate'])  
 # Record training time  
 start\_time = time.time()  
 model.train(X\_train, y\_train, epochs=200, batch\_size=config['batch\_size'])  
 training\_time = time.time() - start\_time  
 # Test the model  
 predictions = model.predict(X\_test)  
 accuracy\_test = np.mean(predictions.flatten() == y\_test)  
 test\_error = 1 - accuracy\_test  
 predictions\_train = model.predict(X\_train)  
 accuracy\_train = np.mean(predictions\_train.flatten() == y\_train)  
 # Store results  
 results.append({  
 'Batch Size': config['batch\_size'],  
 'Learning Rate': config['learning\_rate'],  
 'Test Error': test\_error,  
 'Train Accuracy (%)': accuracy\_train \* 100,  
 'Test Accuracy (%)': accuracy\_test \* 100,  
 'Training Time (s)': training\_time,  
 'Epochs': 200,  
 })  
# Create a summary DataFrame  
pd.options.display.max\_columns = None # Show all columns  
pd.options.display.max\_rows = None # Show all rows  
pd.options.display.width = 1000  
results\_df = pd.DataFrame(results)  
results\_df.to\_csv('mlp\_training\_results.csv', index=False)  
print("Results saved to 'mlp\_training\_results.csv'")  
Изображение выглядит как снимок экрана, текст, Шрифт

Автоматически созданное описаниеprint(results\_df)

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