Министерство образования Республики Беларусь

Учреждение образования

«Брестский Государственный технический университет»

Кафедра ИИТ

**Лабораторная работа №6**

По дисциплине «Модели решения задач в интеллектуальных системах»

Тема: «Прогнозирование»

**Выполнил:**

Студент 3 курса

Группы ИИ-21

Кирилович А. А.

**Проверил:**

Туз И. С.

Брест 2023

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

**Ход работы (Вариант 4)**

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

learning и batch-learning.

import numpy as np

class Perceptron:

def \_\_init\_\_(self, input\_size, hidden\_size, output\_size, learning\_rate="adapt"):

self.input\_size = input\_size

self.hidden\_size = hidden\_size

self.output\_size = output\_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):

if self.learning\_rate == "adapt":

error = targets - outputs

delta\_output = error \* self.sigmoid\_derivative(outputs)

self.learning\_rate\_output = (4 \* np.sum((error \*\* 2 \* outputs \* (1 - outputs)), axis=0)) / \

(1 + np.sum(self.hidden\_output \*\* 2)) \* \

(np.sum((error \* outputs \* (1 - outputs)) \*\* 2, axis=0))

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)) \* \

(np.sum((h\_error \* self.hidden\_output \* (1 - self.hidden\_output)) \*\* 2, axis=0))

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\_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:

error = targets - outputs

delta\_output = error \* self.sigmoid\_derivative(outputs)

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):

for epoch in range(epochs):

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(target\_data - output))

print(f'Epoch {epoch + 1}, Batch {i // batch\_size + 1}, Loss: {loss}')

def predict(self, inputs):

output = self.forward(inputs)

return output

if \_\_name\_\_ == "\_\_main\_\_":

from sklearn.datasets import load\_breast\_cancer

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import StandardScaler

from sklearn.preprocessing import OneHotEncoder

data = load\_breast\_cancer()

X = data.data

y = data.target.reshape(-1, 1)

scaler = StandardScaler()

X = scaler.fit\_transform(X)

encoder = OneHotEncoder(sparse=False)

y = encoder.fit\_transform(y)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

input\_size = X\_train.shape[1]

hidden\_size = 5

output\_size = y\_train.shape[1]

perceptron = Perceptron(input\_size, hidden\_size, output\_size)

perceptron.train(X\_train, y\_train, epochs=10000, batch\_size=32)

predictions = perceptron.predict(X\_test)

accuracy = np.mean(np.round(predictions) == y\_test)

print(f'Test Accuracy: {accuracy \* 100}%')

**Вывод:** написал сеть с адаптивным шагом с батч обучением.