Министерство образования Республики Беларусь

Учреждение образования

“Брестский государственный технический университет”

Кафедра интеллектуально-информационных технологий

Лабораторная работа №6

“ MLP. Прогнозирование”

Выполнил:

студент 3 курса

группы ИИ-22

Копанчук Е. Р.

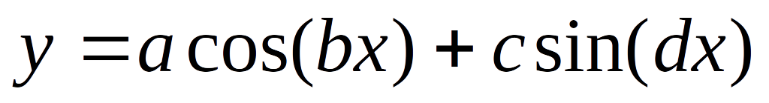
Проверил:

Туз И. С.

Брест 2023

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

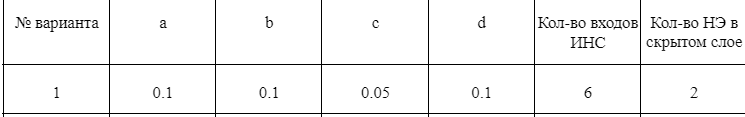
решения задачи прогнозирования функции:



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)



from matplotlib import pyplot as plt

import numpy as np

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

from sklearn.metrics import accuracy\_score, classification\_report

from scipy.special import expit

import time

E\_arr = []

class Perceptron :

def \_\_init\_\_(self, input\_size, hidden\_size, output\_size, learning\_rate = 0.05) :

self.input\_size = input\_size

self.hidden\_size = hidden\_size

self.output\_size = output\_size

self.learning\_rate = learning\_rate

self.start\_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))

# print(self.weights\_input\_hidden)

# print(self.weights\_hidden\_output)

def sigmoid(self, x) :

return expit(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.output

def backward(self, inputs, target, output) :

error = target - output

delta\_hidden = error.dot(self.weights\_hidden\_output.T) \* self.sigmoid\_derivative(self.hidden\_output)

self.weights\_hidden\_output += self.hidden\_output.T.dot(error) \* self.learning\_rate

self.bias\_output += np.sum(error, 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: int, isAdapt : bool = False) :

global E\_arr

for epoch in range(epochs) :

e\_arr = []

for i in range(len(inputs)) :

input\_data = np.array([inputs[i]])

target\_data = np.array([targets[i]])

output = self.forward(input\_data)

e\_arr.append(target\_data - output)

self.backward(input\_data, target\_data, output)

E2 = np.sum(np.array(e\_arr) \* \*2) / 2

E\_arr.append(E2)

self.learning\_rate = self.start\_rate \* (1.0 / (1.0 + epoch / 100)) if isAdapt else self.learning\_rate

#print(f"Online: Epoch: {epoch} MSE: {E2} LR: {self.learning\_rate}")

def backwardBatch(self, inputs, targets, outputs) :

error = (targets.T - outputs).T

mse\_batch = np.sum(error) / len(error)

inputs\_cut = 0

for j in range(len(error)) : inputs\_cut += error[j] \* inputs[j]

mse\_batch = np.array(mse\_batch)

inputs\_cut = np.array(inputs\_cut).reshape(1, -1)

delta\_hidden = mse\_batch.dot(self.weights\_hidden\_output.T) \* self.sigmoid\_derivative(self.hidden\_output)

self.weights\_hidden\_output += self.hidden\_output.T.dot(mse\_batch) \* self.learning\_rate

self.bias\_output += np.sum(mse\_batch, axis = 0, keepdims = True) \* self.learning\_rate

self.weights\_input\_hidden += inputs\_cut.T.dot(delta\_hidden) \* self.learning\_rate

self.bias\_hidden += np.sum(delta\_hidden, axis = 0, keepdims = True) \* self.learning\_rate

def trainBatch(self, inputs, targets, epochs: int, batchsize : int, isAdapt : bool = False) :

global E\_arr

if (len(inputs) % batchsize != 0) :

print("Плохое значение пакета")

return ValueError

inputspack = [inputs[i - batchsize:i] for i in range(batchsize, len(inputs), batchsize)]

targetspack = [targets[i - batchsize:i] for i in range(batchsize, len(targets), batchsize)]

for epoch in range(epochs) :

e\_arr = []

for i in range(len(inputspack)) :

outputs = [self.forward(batchElem).item() for batchElem in inputspack[i]]

for j in range(len(targetspack[i])) : e\_arr.append(targetspack[i][j] - outputs[j])

self.backwardBatch(inputspack[i], targetspack[i], outputs)

E2 = np.sum(np.array(e\_arr) \* \*2) / 2

E\_arr.append(E2)

self.learning\_rate = self.start\_rate \* (1.0 / (1.0 + epoch / 100)) if isAdapt else self.learning\_rate

#print(f"Batch: Epoch: {epoch} MSE: {E2} LR: {self.learning\_rate}")

def predict(self, inputs) :

output = self.forward(inputs)

return output

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

import csv

epochs = 100

X, Y = [], []

with open(r"D:\diabetes.csv") as file :

reader = csv.reader(file)

for i, row in enumerate(reader) :

X.append([float(value) for value in row[:-1]])

Y.append([float(row[-1])])

X = np.array(X)

Y = np.array(Y)

X /= 1000

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

perceptron = Perceptron(8, 1, 1)

start = time.time()

perceptron.train(X\_train, y\_train, epochs, False)

end = time.time()

temp = perceptron.predict(X\_test)

y\_pred = (temp > 0.5).astype(int)

accuracy = accuracy\_score(y\_test, y\_pred)

print(f"Time taken: {(end-start):.03f}s")

print(f"Точность модели : {accuracy:.2f}")

print(f"Точность модели (MSE) : {min(E\_arr):.10f}")

plt.plot(range(epochs), E\_arr, '-')

E\_arr.clear()

print(classification\_report(y\_test, y\_pred, zero\_division = 1))

perceptron = Perceptron(8, 1, 1)

start = time.time()

perceptron.train(X\_train, y\_train, epochs, True)

end = time.time()

temp = perceptron.predict(X\_test)

y\_pred = (temp > 0.5).astype(int)

accuracy = accuracy\_score(y\_test, y\_pred)

print(f"Time taken: {(end-start):.03f}s")

print(f"Точность модели: {accuracy:.2f}")

print(f"Точность модели (MSE) : {min(E\_arr):.10f}")

plt.plot(range(epochs), E\_arr, '--')

E\_arr.clear()

print(classification\_report(y\_test, y\_pred, zero\_division = 1))

perceptron = Perceptron(8, 1, 1)

start = time.time()

perceptron.trainBatch(X\_train, y\_train, epochs, 2, False)

end = time.time()

temp = perceptron.predict(X\_test)

y\_pred = (temp > 0.5).astype(int)

accuracy = accuracy\_score(y\_test, y\_pred)

print(f"Time taken: {(end-start):.03f}s")

print(f"Точность модели: {accuracy:.2f}")

print(f"Точность модели (MSE) : {min(E\_arr):.10f}")

plt.plot(range(epochs), E\_arr, '-.')

E\_arr.clear()

print(classification\_report(y\_test, y\_pred, zero\_division = 1))

perceptron = Perceptron(8, 1, 1)

start = time.time()

perceptron.trainBatch(X\_train, y\_train, epochs, 2, True)

end = time.time()

temp = perceptron.predict(X\_test)

y\_pred = (temp > 0.5).astype(int)

accuracy = accuracy\_score(y\_test, y\_pred)

print(f"Time taken: {(end-start):.03f}s")

print(f"Точность модели: {accuracy:.2f}")

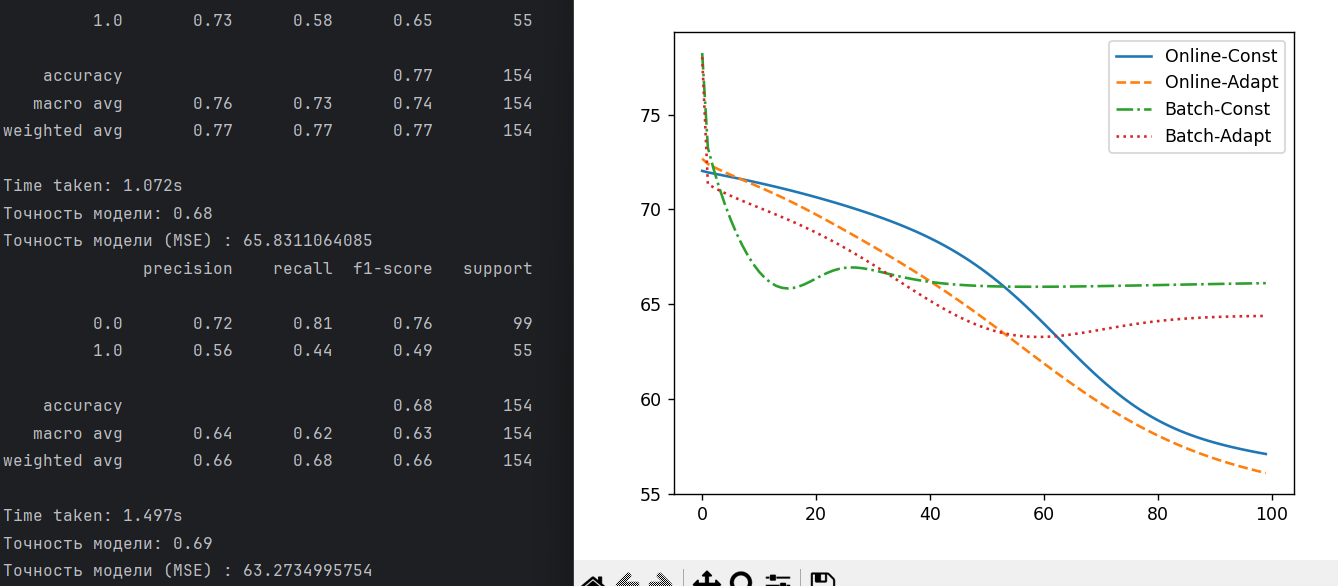
print(f"Точность модели (MSE) : {min(E\_arr):.10f}")

plt.plot(range(epochs), E\_arr, ':')

plt.legend(['Online-Const', 'Online-Adapt', 'Batch-Const', 'Batch-Adapt'], loc = "upper right")

plt.show()

print(classification\_report(y\_test, y\_pred, zero\_division = 1))



Вывод: изучил обучение и функционирование ИНС при решении задач прогнозирование.