Пензенский государственный университет

Кафедра «Вычислительная техника»

**ОТЧЕТ**

по лабораторной работе № 2

по курсу «Основы глубокого обучения»

Выполнил:  
студент группы 22ВВИм1:

Емелин С.В.

Приняли:

Митрохин М.А  
Панков А.А.

Пенза 2023

# Порядок выполнения работы:

# Задание:

ЗАДАНИЕ 1: реализовать алгоритм стохастического обучения (аналогично стохастическому градиентному спуску).

ЗАДАНИЕ 2: реализовать нейронную сеть, осуществляющую классификацию цветков Ириса на 3 класса по всем 4 имеющимся признакам.

# В качестве функции потерь была выбрана MSE.

# https://blogs.brain-mentors.com/content/images/2022/06/mean-squared-error.png

Рисунок MSE

# В качестве функции активации нейрона используется сигмоида

# https://pic2.zhimg.com/d2d1335c99df1923da4f228da3da9bdd_r.jpg

Рисунок 2 Сигмоида

# Листинг программы:

import pandas as pd  
import numpy as np  
import math  
import random  
import matplotlib.pyplot as plt  
  
  
# Функция активации сигмоида.  
def sigmoid\_activation(x):  
 return 1 / (1 + math.exp(-x))  
  
  
# Производная функции активации.  
def sigmoid\_derivative\_activation(x):  
 return sigmoid\_activation(x) \* (1 - sigmoid\_activation(x))  
  
  
# Возвденние в квадрат  
def square(x):  
 return x \* x  
  
  
square = np.vectorize(square)  
  
size\_input = 4  
  
first\_layer = 6  
  
second\_layer = 3  
  
gradient\_step = 0.08  
  
  
def errors(result, expected\_class):  
 expected\_vector = np.array([0, 0, 0])  
 expected\_vector[expected\_class] = 1  
 # считаем ошибку по MSE для каждого класса и складываем  
 sum\_errors = sum(square(result - expected\_vector))  
 return (sum\_errors, np.array([result - expected\_vector]))  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 df = pd.read\_csv('data.csv')  
 df = df.iloc[np.random.permutation(len(df))]  
 y = df.iloc[0:150, 4].values  
 y = np.where(y == "Iris-setosa", 0, y)  
 y = np.where(y == "Iris-versicolor", 1, y)  
 y = np.where(y == "Iris-virginica", 2, y)  
  
 X = df.iloc[0:150, 0:4].values  
  
 # матрица весов первого слоя  
 W\_1 = np.random.rand(size\_input, first\_layer)  
 # матрица весов вторго слоя  
 W\_2 = np.random.randn(first\_layer, second\_layer)  
  
 for x\_input in X:  
 print(x\_input)  
  
 vector\_sigmoid\_activation = np.vectorize(sigmoid\_activation)  
 vector\_sigmoid\_derivative\_activation = np.vectorize(sigmoid\_derivative\_activation)  
  
 ziped = zip(X, y)  
 zipped\_list = list(ziped)  
  
 # гиперпараметры  
 epoch = 100  
 batch\_size = 25  
  
 count\_batch = (int(len(zipped\_list) / batch\_size))  
  
 errors\_epoch = []  
 for i in range(epoch):  
 error\_epoch = 0  
 random.shuffle(zipped\_list)  
 for j in range(1, count\_batch + 1):  
  
 batch = zipped\_list[batch\_size \* j - batch\_size:batch\_size \* j]  
  
 new\_W\_1 = np.zeros((size\_input, first\_layer))  
 new\_W\_2 = np.zeros((first\_layer, second\_layer))  
 sum\_errors = 0  
 for x\_input, expected in batch:  
 o1 = x\_input @ W\_1  
 o1\_activated = vector\_sigmoid\_activation(o1)  
 o2 = o1\_activated @ W\_2  
 o2\_activated = vector\_sigmoid\_activation(o2)  
 error, expected\_vector = errors(o2\_activated, expected)  
 sum\_errors = sum\_errors + error  
  
 de\_e = vector\_sigmoid\_derivative\_activation(o2) \* expected\_vector  
 de\_dw2 = np.dot(np.array([o1\_activated]).T, de\_e)  
 de\_o1\_activated = de\_e @ W\_2.T  
 de\_01 = vector\_sigmoid\_derivative\_activation(o1) \* de\_o1\_activated  
 de\_dw1 = np.array([x\_input]).T @ de\_01  
  
 new\_W\_1 = new\_W\_1 + de\_dw1  
 new\_W\_2 = new\_W\_2 + de\_dw2  
 jj = new\_W\_1 / batch\_size  
  
 W\_1 = W\_1 - gradient\_step \* (new\_W\_1 / batch\_size)  
 W\_2 = W\_2 - gradient\_step \* (new\_W\_2 / batch\_size)  
 print('error in batch', sum\_errors / batch\_size)  
 error\_epoch += sum\_errors / batch\_size  
 print('error in epoch', sum\_errors / batch\_size)  
 errors\_epoch.append(error\_epoch)  
  
 plt.plot(errors\_epoch)  
 plt.show()  
  
 correct = 0  
 incorrect = 0  
 for x\_input, expected in zipped\_list:  
 o1 = x\_input @ W\_1  
 o1\_activated = vector\_sigmoid\_activation(o1)  
 o2 = o1\_activated @ W\_2  
 o2\_activated = vector\_sigmoid\_activation(o2)  
 error, expected\_vector = errors(o2\_activated, expected)  
 sum\_errors = sum\_errors + error  
 if (np.argmax(o2\_activated, axis=0) == expected):  
 correct += 1  
 else:  
 incorrect += 1  
  
 print('correct', correct)  
 print('incorrect', incorrect)

# Результат работы программы:

# 

Рисунок 3 Процесс обучения

# 

Рисунок 4 Визуализация уменьшения ошибки

# Вывод:

# В ходе выполнения лабораторной работы был реализованы реализовать алгоритм стохастического обучения для обучения нейронной сети осуществляющую классификацию цветков Ириса на 3 класса по всем 4 имеющимся признакам.

# 