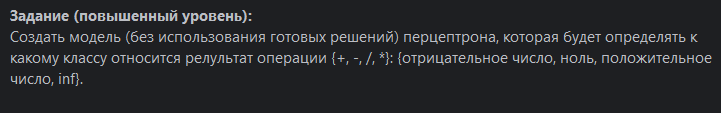
# Группа 5 Панфилов Валерий Александрович Лабораторная работа №2

«Создание нейронной сети, выполняющей мультиклассовую классификацию»

Задание: 

Код:

import random  
import math  
  
dataset = list()  
  
actions = ['+','-', "/", "\*"]  
  
  
for i in range(1000):  
 temp = list()  
 temp.append(random.randint(-100, 100) \* 0.01)  
 temp.append(random.randint(-100, 100) \* 0.01)  
 temp.append(random.randint(0, 3))  
 if temp[2] == 1:  
 if temp[0] - temp[1] < 0:  
 temp.append(0)  
 elif temp[0] - temp[1] == 0:  
 temp.append(1)  
 else:  
 temp.append(2)  
 elif temp[2] == 0:  
 if temp[0] + temp[1] < 0:  
 temp.append(0)  
 elif temp[0] + temp[1] == 0:  
 temp.append(1)  
 else:  
 temp.append(2)  
 elif temp[2] == 2:  
 if temp[1] == 0:  
 temp.append(3)  
 elif temp[0] / temp[1] < 0:  
 temp.append(0)  
 elif temp[0] / temp[1] == 0:  
 temp.append(1)  
 else:  
 temp.append(2)  
 elif temp[2] == 3:  
 if temp[0] \* temp[1] < 0:  
 temp.append(0)  
 elif temp[0] \* temp[1] > 0:  
 temp.append(2)  
 else:  
 temp.append(1)  
  
 dataset.append(temp)  
  
  
class Perceptron():  
 def \_\_init\_\_(self, training\_dataset, test\_dataset):  
 self.training\_dataset = training\_dataset  
 self.test\_dataset = test\_dataset  
 self.train()  
  
 def sigmoid(self,x):  
 sig = (1 / (1 + math.exp(-x)))  
 return sig  
  
 def multiclass\_logistic\_regression(self,weights, inputs):  
 scores = []  
 for i in range(4): # Четыре класса  
 z = weights[i][0] # Смещение (bias) для i-го класса  
 for j in range(len(weights[i]) - 1):  
  
 z += weights[i][j + 1] \* inputs[j]  
 scores.append(self.sigmoid(z))  
 return scores  
  
 def update\_weights(self,weights, inputs, target, learning\_rate):  
 new\_weights = []  
 scores = self.multiclass\_logistic\_regression(weights, inputs)  
 for i in range(4): # Четыре класса  
 error = 1 if i == target else 0 # Целевой класс  
 error -= scores[i] # Градиент  
  
 new\_weights.append([weights[i][0] + learning\_rate \* error])  
 for j in range(len(weights[i]) - 1):  
 new\_weights[i].append(weights[i][j + 1] + learning\_rate \* error \* inputs[j])  
 return new\_weights  
  
 def train(self):  
 num\_features = 3 # Количество признаков  
 weights = [[0.05 for \_ in range(num\_features)] for \_ in range(4)] # Веса для каждого класса  
 learning\_rate = 0.00001 # Скорость обучения  
 num\_epochs = 1000 # Количество эпох обучения  
 for epoch in range(num\_epochs):  
 for i in range(len(self.training\_dataset)):  
 inputs = self.training\_dataset[i][:3]  
 target = self.training\_dataset[i][3]  
 weights = self.update\_weights(weights, inputs, target, learning\_rate)  
 self.test\_training\_dataset(weights)  
 self.test(weights)  
  
 def test\_training\_dataset(self,weights):  
 correct\_predictions = 0  
 for i in range(len(self.training\_dataset)):  
 inputs = self.training\_dataset[i][:3]  
 target = self.training\_dataset[i][3]  
 scores = self.multiclass\_logistic\_regression(weights, inputs)  
 predicted\_class = scores.index(max(scores))  
 if predicted\_class == target:  
 correct\_predictions += 1  
  
 accuracy = correct\_predictions / len(self.training\_dataset)  
 print(f"Точность модели на обучающем датасете: {accuracy \* 100}%")  
  
 def test(self,weights):  
 correct\_predictions = 0  
 for i in range(len(self.test\_dataset)):  
 inputs = self.test\_dataset[i][:3]  
 target = self.test\_dataset[i][3]  
 scores = self.multiclass\_logistic\_regression(weights, inputs)  
 predicted\_class = scores.index(max(scores))  
 if predicted\_class == target:  
 correct\_predictions += 1  
  
 accuracy = correct\_predictions / len(self.test\_dataset)  
 print(f"Точность модели на тестовом датасете: {accuracy \* 100}%")  
  
  
  
perceptron = Perceptron(dataset[:800],dataset[800:])

Результат выполнения:

