Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение

высшего образования

**«КУБАНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»**

**(ФГБОУ ВО «КубГУ»)**

**Факультет компьютерных технологий и прикладной математики**

**Кафедра вычислительных технологий**

**ЛАБОРАТОРНАЯ РАБОТА №2**

**Дисциплина: Приложение нейросетевых алгоритмов**

Работу выполнил: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Д.М. Вакуленко

Направление подготовки: 02.03.03 Математическое обеспечение и администрирование информационных систем

Преподаватель: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ В. И. Шиян

**Цель работы**

Получение знаний и практических навыков построения

однослойных нейронных сетей.

***Задача 1***

Пусть имеется функция XOR, в которой две точки {(0, 0), (1, 1)}

принадлежат к одному классу, а две другие точки {(1, 0), (0, 1)} – к другому.

Покажите, как разделить два этих класса, используя функцию активации

ReLU.

**Решение:**

import random  
  
class Dot:  
 def \_\_init\_\_(self): #Инициализируем точку  
 self.coord = [  
 random.randint(0,1),  
 random.randint(0,1)  
 ]  
  
 def calculate(self, w: list): #Считаем индуцированное локальное поле  
 return max(0, self.coord[0] \* w[0] + self.coord[1] \* w[1])  
  
 def out(self): #Выводим точку на экран  
 return self.coord  
  
  
class NeuroNet:  
  
 def \_\_init\_\_(self):  
 self.w = [1, 1]  
  
 self.dots = [Dot() for i in range(10)]  
  
 def start(self): #Запускаем нейросеть  
 for i in range(10):  
 if self.dots[i].calculate(self.w) == 1:  
 print(self.dots[i].out())  
 print("Точка принадлежит к 2ому классу\n")  
 else:  
 print(self.dots[i].out())  
 print("Точка принадлежит к 1ому классу\n")  
  
  
def main():  
 n = NeuroNet()  
 n.start()  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 main()

**Описание алгоритма:**

В классе Dot определена точка с двумя координатами, которые инициализируются случайным образом при создании экземпляра класса. Есть метод calculate(), который вычисляет индуцированное локальное поле для этой точки на основе весов w1, w2. Метод out() выводит координаты точки на экран.

В классе NeuroNet определен нейронная сеть с весами w1, w2 и списком из 10 точек, представленных экземплярами класса Dot. Метод start() запускает нейросеть и для каждой точки выводит ее координаты и класс, к которому она принадлежит (1ый или 2ой).

Функция main создает экземпляр класса NeuroNet и вызывает метод start().

Таким образом, программа создает нейронную сеть, классифицирует случайно сгенерированные точки на два класса и выводит результаты на экран.

**Вывод алгоритма:**

**Изображение выглядит как текст, снимок экрана, программное обеспечение, Мультимедийное программное обеспечение

Автоматически созданное описание**

***Задача 2***

Задача 2 Пусть имеется двухмерный набор данных, в котором все точки

с x1 > x2 принадлежат к положительному классу, а все точки с x1 < x2 к

отрицательному. Разделителем для этих двух классов является линейная

гиперплоскость (прямая линия), определяемая уравнением x1 − x2 = 0 .

Создайте набор тренировочных данных с 20 точками, сгенерированными случайным образом в положительном квадранте единичного квадрата. Снабдите каждую точку меткой, указывающей на то, превышает или не превышает её первая координата x1 вторую координату x2.

А. Реализуйте алгоритм перцептрона, обучите его на полученных выше

20 точках и протестируйте его точность на 1000 точках, случайно

сгенерированных в единичном квадрате. Используйте для генерирования

тестовых точек ту же процедуру, что и для тренировочных.

**Решение:**

**A:**

import numpy as np  
  
  
class Perceptron:  
 def \_\_init\_\_(self, num\_features):  
 self.weights = np.zeros(num\_features)  
 self.bias = 0.0  
  
 def predict(self, x):  
 return np.sign(np.dot(self.weights, x) + self.bias)  
  
 def train(self, X\_train, y\_train, epochs=100, learning\_rate=0.1):  
 for \_ in range(epochs):  
 for i in range(len(X\_train)):  
 prediction = self.predict(X\_train[i])  
 if prediction != y\_train[i]:  
 # print(type(y\_train[i]), type(X\_train[i]))  
 self.weights += learning\_rate \* y\_train[i] \* X\_train[i]  
 self.bias += learning\_rate \* y\_train[i]  
  
 def \_\_str\_\_(self):  
 return f"Weights: {self.weights}, Bias: {self.bias}"  
  
  
# Генерация данных  
X\_train = np.random.uniform(0, 1, (20, 2))  
y\_train = np.array([1 if x[0] > x[1] else 0 for x in X\_train])  
  
  
# Обучение перцептрона  
perceptron = Perceptron(2)  
perceptron.train(X\_train, y\_train)  
print(perceptron)  
  
  
# Тестирование на 1000 точках  
X\_test = np.random.uniform(0, 1, (1000, 2))  
y\_test = np.array([1 if x[0] > x[1] else 0 for x in X\_test])  
  
  
correct\_predictions = 0  
for i in range(len(X\_test)):  
 if perceptron.predict(X\_test[i]) == y\_test[i]:  
 correct\_predictions += 1  
  
accuracy = correct\_predictions / len(X\_test)  
print(f"Точность на 1000 точках: {accuracy}")

**Описание алгоритма:**

В начале программы создается класс Perceptron, который реализует простой персептрон. В конструкторе класса инициализируются веса и смещение (bias) весами равными нулю.

Метод predict принимает входные данные x и возвращает предсказание модели (1 или -1) на основе текущих весов и смещения.

Метод train используется для обучения персептрона. Персептрон обновляет веса и смещение на каждом примере тренировочного набора данных, если предсказание не соответствует метке y\_train.

После обучения персептрона на случайно сгенерированных данных X\_train и метках y\_train, создается экземпляр класса Perceptron и вызывается метод train для обучения.

Затем программа тестирует обученный персептрон на 1000 случайных точек X\_test и их метках y\_test.

На выходе программа выводит значения обученных весов и смещения, а также точность модели на 1000 случайных точек.

Б. Замените критерий перцептрона на нейрон типа адалайн.

**Решение:**

import numpy as np  
  
  
class Perceptron:  
 def \_\_init\_\_(self, num\_features):  
 self.weights = np.zeros(num\_features)  
 self.bias = 0.0  
  
 def predict(self, x):  
 return np.sign(np.dot(self.weights, x) + self.bias)  
  
 def train(self, X\_train, y\_train, epochs=100, learning\_rate=0.1):  
 for \_ in range(epochs):  
 for i in range(len(X\_train)):  
 prediction = self.predict(X\_train[i])  
 if prediction != y\_train[i]:  
 self.weights += learning\_rate \* y\_train[i] \* X\_train[i]  
 self.bias += learning\_rate \* y\_train[i]  
  
 def \_\_str\_\_(self):  
 return f"Weights: {self.weights}, Bias: {self.bias}"  
  
  
# Генерация данных  
X\_train = np.random.uniform(0, 1, (20, 2))  
y\_train = np.array([1 if x[0] > x[1] else -1 for x in X\_train])  
  
  
# Обучение перцептрона  
perceptron = Perceptron(2)  
perceptron.train(X\_train, y\_train)  
print(perceptron)  
  
  
# Тестирование на 1000 точках  
X\_test = np.random.uniform(0, 1, (1000, 2))  
y\_test = np.array([1 if x[0] > x[1] else -1 for x in X\_test])  
  
  
correct\_predictions = 0  
for i in range(len(X\_test)):  
 if perceptron.predict(X\_test[i]) == y\_test[i]:  
 correct\_predictions += 1  
  
accuracy = correct\_predictions / len(X\_test)  
print(f"Точность на 1000 точках: {accuracy}")

Алгоритм такой же, как и в пункте А, за исключением того, что на выходе нейрона может быть значение 1 и -1, а не 1 и 0, как в предыдущем пункте.

**Вывод программы:**

A:

**Изображение выглядит как текст, Шрифт, снимок экрана

Автоматически созданное описание**

B:

Изображение выглядит как текст, Мультимедийное программное обеспечение, программное обеспечение, снимок экрана

Автоматически созданное описание

***Задача 3***

Требуется разработать и исследовать нейронную сеть обратного распространения, предназначенную для распознавания образов.

Даны в виде матрицы 3×3 (см. таблицу) 4 латинские буквы X , Y , L, I .

**Решение:**

import random  
  
import numpy as np  
  
letters = ['X', 'Y', 'I', 'L']  
# Задаем входной вектор X, Y, L или I  
examples = np.array([[1, 0, 1, 0, 1, 0, 1, 0, 1],  
 [1, 0, 1, 0, 1, 0, 0, 1, 0],  
 [0, 1, 0, 0, 1, 0, 0, 1, 0],  
 [1, 0, 0, 1, 0, 0, 1, 1, 1]])  
  
# Задаем выходные метки для каждой из букв X, Y, I, L  
# X: [0, 0, 0, 1], Y: [0, 0, 1, 0], I: [0, 1, 0, 0], L: [1, 0, 0, 0]  
need\_outputs = np.array(  
 [[0, 0, 0, 1],  
 [0, 0, 1, 0],  
 [0, 1, 0, 0],  
 [1, 0, 0, 0]])  
  
input\_neurons = 9  
hidden\_neurons = 5  
output\_neurons = 4  
  
weights\_input\_hidden = np.random.uniform(size=(input\_neurons, hidden\_neurons))  
weights\_hidden\_output = np.random.uniform(size=(hidden\_neurons, output\_neurons))  
bias\_hidden = np.random.uniform(size=(1, hidden\_neurons))  
bias\_output = np.random.uniform(size=(1, output\_neurons))  
  
# Задаем скорость обучения и количество эпох  
learning\_rate = 0.01  
epochs = 100  
  
  
# Функция активации (сигмоида)  
def sigmoid(x):  
 return 1 / (1 + np.exp(-x))  
  
# Производная функции активации  
def sigmoid\_derivative(x):  
 return x \* (1 - x)  
  
# Обучение нейронной сети  
for i in range(len(examples)):  
 for epoch in range(epochs):  
 # Forward pass  
 hidden\_layer\_input = np.dot(examples[i], weights\_input\_hidden) + bias\_hidden  
 hidden\_layer\_output = sigmoid(hidden\_layer\_input)  
  
 output\_layer\_input = np.dot(hidden\_layer\_output, weights\_hidden\_output) + bias\_output  
 output = sigmoid(output\_layer\_input)  
  
 # Backpropagation  
 error = need\_outputs[i] - output  
 d\_output = error \* sigmoid\_derivative(output)  
  
 error\_hidden = d\_output.dot(weights\_hidden\_output.T)  
 d\_hidden = error\_hidden \* sigmoid\_derivative(hidden\_layer\_output)  
  
 # Обновление весов и смещений  
 weights\_hidden\_output += hidden\_layer\_output.T.dot(d\_output) \* learning\_rate  
 bias\_output += np.sum(d\_output, axis=0, keepdims=True) \* learning\_rate  
  
 weights\_input\_hidden += examples[i].reshape(9, 1).dot(d\_hidden) \* learning\_rate  
 bias\_hidden += np.sum(d\_hidden, axis=0, keepdims=True) \* learning\_rate  
  
 test\_input = np.array(examples[i])  
 hidden\_layer\_input = np.dot(test\_input, weights\_input\_hidden) + bias\_hidden  
 hidden\_layer\_output = sigmoid(hidden\_layer\_input)  
  
 output\_layer\_input = np.dot(hidden\_layer\_output, weights\_hidden\_output) + bias\_output  
 predicted\_output = sigmoid(output\_layer\_input)  
  
 # Выводим предсказанный выход  
 print(predicted\_output)  
  
print("////////////////////////////////////////////\n")  
  
correct\_prediction = 0  
  
for i in range(1000):  
 n = random.randint(0,3)  
 test\_input = np.array(examples[n])  
 hidden\_layer\_input = np.dot(test\_input, weights\_input\_hidden) + bias\_hidden  
 hidden\_layer\_output = sigmoid(hidden\_layer\_input)  
  
 output\_layer\_input = np.dot(hidden\_layer\_output, weights\_hidden\_output) + bias\_output  
 predicted\_output = sigmoid(output\_layer\_input)  
 arr = []  
 for elem in predicted\_output:  
 arr.append(elem)  
 if arr.index(max(arr)) == n:  
 correct\_prediction += 1  
  
print("Accuracy: " + str(float(correct\_prediction/1000)\*100) + str("%"))

**Описание алгоритма:**

Этот код реализует нейронную сеть для распознавания букв X, Y, I, L. Нейронная сеть имеет один скрытый слой с пятью нейронами и четыре выходных нейрона, соответствующих каждой из букв.

Вначале мы задаем входные векторы для каждой из букв и соответствующие им выходные метки. Затем задаем веса и смещения для связей между входным слоем и скрытым слоем, а также между скрытым и выходным слоями.

Затем мы начинаем обучение нейронной сети с помощью алгоритма обратного распространения ошибки. Для каждого обучающего примера мы делаем прямой проход через сеть, вычисляем ошибку и обновляем веса и смещения, чтобы минимизировать эту ошибку.

После завершения обучения мы проверяем результат предсказания для каждой из букв. Затем мы проводим тестирование сети на случайно выбранных образцах и подсчитываем точность предсказания.

Итоговый вывод показывает предсказанный выход для каждого обучающего образца и точность предсказания после тестирования на случайных данных.

**Вывод программы:**

**Изображение выглядит как текст, программное обеспечение, Мультимедийное программное обеспечение, Графическое программное обеспечение

Автоматически созданное описание**