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 №1**

**Дисциплина: Приложение нейросетевых алгоритмов**

Работу выполнила: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ В. С. Паничева

Направление подготовки: 02.03.03 Математическое обеспечение и администрирование информационных систем

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

**Тема:** задачи классификации, многослойные нейронные сети

**Цель работы:** получение знаний и практических навыков построения однослойных нейронных сетей.

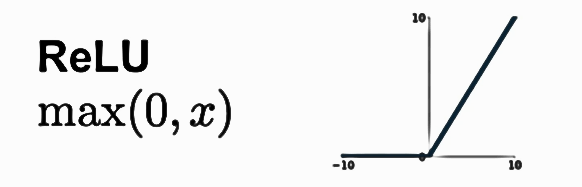
**Задание:** требуется разработать алгоритмы и программы решения задач 1–3 на языке Python без использования специализированных библиотек.

**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.

Точки можно разделить на два класса (класс 0 и класс 1) с использованием операции XOR:



Функция ReLU возвращает входное значение, если оно положительное, в нашем случае 1, иначе возвращает 0.



В данной программе реализован простая нейросеть с функцией активации ReLU, которая обучается на примере задачи XOR. В процессе обучения она пытается разделить два класса XOR, используя эту функцию активации. В функцию ReLU значения передаются в следующем виде:

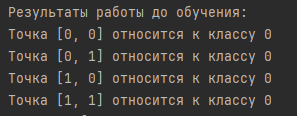
где w – веса, x – входы, b – порог

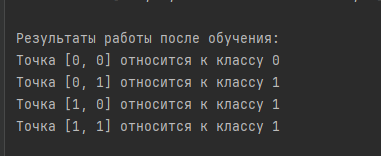
Класс Neuron содержит методы для инициализации весов, вычисления взвешенной суммы и коррекции весов нейрона.

В классе Network определена функция reLU. Далее идет определение входных данных inputs и ожидаемых ответов expected\_answ, создается экземпляр нейрона и сети. Затем происходит обучение нейрона на протяжении 1000 эпох путем вычисления выхода нейрона и коррекции весов.

Код программы:

**from** random **import** \*  
  
  
**class** Neuron:  
 **def** \_\_init\_\_(self):  
 self.w = [uniform(-1, 1) **for** \_ **in** range(2)]  
 self.learning\_rate = 0.5  
  
 **def** calculate(self, inputs): #взвешенная сумма  
 b = 1 # порог  
 **return** self.w[0] \* inputs[0] + self.w[1] \* inputs[1] + b  
  
 **def** weight\_correction(self, expected, output): #коррекция весов  
 gradient = output - expected  
 **if** expected == 1:  
 self.w[0] -= self.learning\_rate \* (expected + gradient) \* (1 + gradient)  
 **else**:  
 self.w[1] -= self.learning\_rate \* (expected + gradient) \* (1 + gradient)  
  
  
**class** Network:  
 **def** reLU(self, x):  
 **return** max(0, x)  
  
inputs = [[0, 0], [0, 1], [1, 0], [1, 1]]  
expected\_answ = [0, 1, 1, 0]  
  
neuron = Neuron()  
network = Network()  
  
print("Результаты работы до обучения:")  
**for** i **in** range(len(inputs)):  
 output = network.reLU(neuron.calculate(inputs[i]))  
 **if** output <= 0.5:  
 print("Точка " + str(inputs[i]) + " относится к классу 1")  
 **else**:  
 print("Точка " + str(inputs[i]) + " относится к классу 0")  
  
# обучение  
epochs = 1000  
**for** i **in** range(epochs):  
 **for** j **in** range(len(inputs)): #считаем взвеш сумму для каждого примера и корректируем веса  
 output = network.reLU(neuron.calculate(inputs[j]))  
 neuron.weight\_correction(expected\_answ[j], output)  
  
 print(f"Эпоха {i + 1}:", [network.reLU(neuron.calculate(input\_point)) **for** input\_point **in** inputs])  
  
print("\nРезультаты работы после обучения:")  
**for** i **in** range(len(inputs)):  
 output = network.reLU(neuron.calculate(inputs[i]))  
 **if** output <= 0.5:  
 print("Точка " + str(inputs[i]) + " относится к классу 1")  
 **else**:  
 print("Точка " + str(inputs[i]) + " относится к классу 0")





**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.

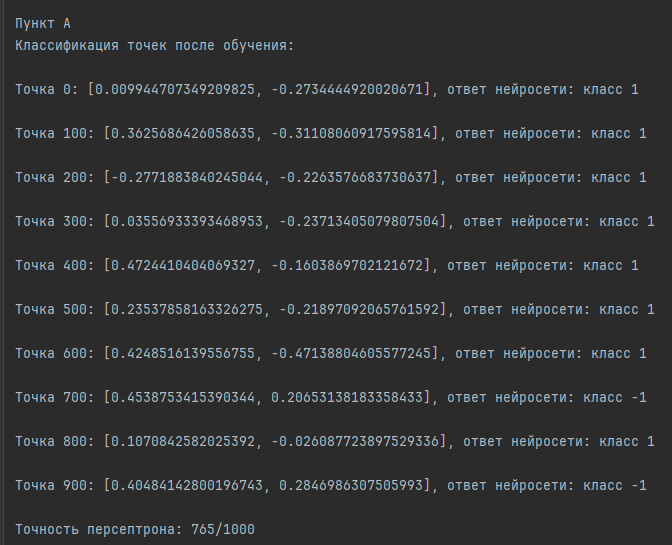
Алгоритм для решения этой задачи аналогичен алгоритму, используемому в предыдущей задаче. Однако на этот раз требуется разделить множество точек на плоскости на два класса: в первом классе первая координата больше второй, во втором классе вторая координата больше первой.

Для такой классификации все еще достаточно одного нейрона. В программе был создан класс Network и Perceptron, класс Perceptron подобен классу Neuron из прошлой задачи. Также был разработан класс Point, где каждая точка содержит информацию о ее координатах и принадлежности определенному классу.

Алгоритм обучает нейросеть на обучающих данных с помощью перцептрона, регулирует веса и затем проверяет классификацию на тестовой выборке, считая количество правильно классифицированных точек. Генерируются 1000 случайных тестовых точек в единичном квадрате и классифицируются при помощи обученного перцептрона. Результат классификации выводится в консоль.

Код программы:

**from** random **import** \*  
  
  
**class** Neuron:  
 **def** \_\_init\_\_(self):  
 self.w = [uniform(-1, 1) **for** \_ **in** range(2)]  
 self.learning\_rate = 0.5  
  
 **def** calculate(self, inputs): #взвешенная сумма  
 b = 1 # порог  
 **return** self.w[0] \* inputs[0] + self.w[1] \* inputs[1] + b  
  
 **def** weight\_correction(self, expected, output): #коррекция весов  
 gradient = output - expected  
 **if** expected == 1:  
 self.w[0] -= self.learning\_rate \* (expected + gradient) \* (1 + gradient)  
 **else**:  
 self.w[1] -= self.learning\_rate \* (expected + gradient) \* (1 + gradient)  
  
  
**class** Network:  
 **def** reLU(self, x):  
 **return** max(0, x)  
  
inputs = [[0, 0], [0, 1], [1, 0], [1, 1]]  
expected\_answ = [0, 1, 1, 0]  
  
neuron = Neuron()  
network = Network()  
  
print("Результаты работы до обучения:")  
**for** i **in** range(len(inputs)):  
 output = network.reLU(neuron.calculate(inputs[i]))  
 **if** output <= 0.5:  
 print("Точка " + str(inputs[i]) + " относится к классу 1")  
 **else**:  
 print("Точка " + str(inputs[i]) + " относится к классу 0")  
  
# обучение  
epochs = 1000  
**for** i **in** range(epochs):  
 **for** j **in** range(len(inputs)): #считаем взвеш сумму для каждого примера и корректируем веса  
 output = network.reLU(neuron.calculate(inputs[j]))  
 neuron.weight\_correction(expected\_answ[j], output)  
  
 print(f"Эпоха {i + 1}:", [network.reLU(neuron.calculate(input\_point)) **for** input\_point **in** inputs])  
  
print("\nРезультаты работы после обучения:")  
**for** i **in** range(len(inputs)):  
 output = network.reLU(neuron.calculate(inputs[i]))  
 **if** output <= 0.5:  
 print("Точка " + str(inputs[i]) + " относится к классу 1")  
 **else**:  
 print("Точка " + str(inputs[i]) + " относится к классу 0")



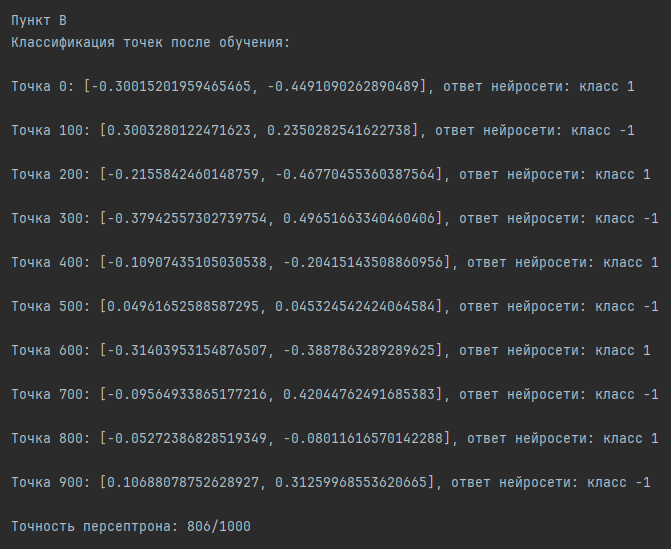
Б. Замените критерий перцептрона на нейрон типа адалайн при реализации тренировки и повторите определение точности вычислений на тех же тестовых точках, которые использовали перед этим. Удалось ли вам в каком-то из способов получить лучшую точность? Как вы считаете, в каком случае классификация тех же 1000 тестовых точек не изменится значительно, если использовать другой набор из 20 тренировочных точек?

Здесь был добавлен класс Adaline. Этот тип нейрона отличается только подходом к изменению весов. Благодаря такому изменению точность нейросети возросла.

Классификация 1000 тестовых точек не изменится значительно, если использовать другой набор из 20 тренировочных точек, если новые тренировочные точки представляют собой качественную выборку данных и включают разнообразные характеристики, позволяющие модели адекватно обучиться на них.

Код программы:

**from** random **import** \*  
**class** Point:  
 classification = 0  
  
 **def** \_\_init\_\_(self, a, b):  
 self.x = [uniform(a, b) **for** \_ **in** range(2)]  
  
 **if** self.x[0] > self.x[1]:  
 self.classification = 1  
 **else**:  
 self.classification = -1  
**class** Perceptron:  
 **def** \_\_init\_\_(self):  
 self.w = [uniform(-1, 1) **for** \_ **in** range(2)]  
 self.learning\_rate = 0.5  
 **def** calculate(self, inputs): #взвешенная сумма  
 b = 1 # порог  
 **return** self.w[0] \* inputs[0] + self.w[1] \* inputs[1] + b  
 **def** weight\_correction(self, expected, output): #коррекция весов  
 gradient = output - expected  
 **if** expected == 0:  
 self.w[0] -= self.learning\_rate \* (expected + gradient) \* (1 + gradient)  
 **else**:  
 self.w[1] -= self.learning\_rate \* (expected + gradient) \* (1 + gradient)  
  
**class** Adaline:  
 **def** \_\_init\_\_(self):  
 self.w = [uniform(-1, 1) **for** \_ **in** range(2)]  
 self.learning\_rate = 0.5  
  
 **def** calculate(self, inputs): #взвешенная сумма  
 b = 1 # порог  
 **return** self.w[0] \* inputs[0] + self.w[1] \* inputs[1] + b  
  
 **def** weight\_correction(self, y, u, x): #коррекция весов дискретным способом  
 **for** i **in** range(len(self.w)):  
 self.w[i] -= self.learning\_rate \* (u - y) \* x[i]  
**class** Network:  
 **def** reLU(self, summ):  
 **return** max(0, summ)  
  
  
#генерация обучающих примеров  
points\_init = [Point(0, 0.5) **for** i **in** range(20)]  
points\_coords = [[points\_init[i].x[0], points\_init[i].x[1]] **for** i **in** range(len(points\_init))]  
points\_result\_class = [points\_init[i].classification **for** i **in** range(len(points\_init))]  
print()  
  
#задание 1  
print("Пункт A")  
neuron = Perceptron()  
network = Network()  
  
# обучаем нейросеть, изменяем веса  
epochs = 1000  
**for** i **in** range(epochs):  
 errors = []  
 **for** j **in** range(len(points\_init)): #считаем взвеш сумму для каждого примера и корректируем веса  
 output = network.reLU(neuron.calculate(points\_coords[j]))  
 neuron.weight\_correction(points\_result\_class[j], output)  
  
points\_random = [Point((-0.5), 0.5) **for** i **in** range(1000)]  
points\_random\_coords = [[points\_random[i].x[0], points\_random[i].x[1]] **for** i **in** range(len(points\_random))]  
points\_random\_result = [points\_random[i].classification **for** i **in** range(len(points\_random))]  
  
print("Классификация точек после обучения:")  
correctly = 0  
**for** i **in** range(len(points\_random)):  
 output = network.reLU(neuron.calculate(points\_random\_coords[i]))  
  
 classification = (-1) **if** output < 0.5 **else** 1  
  
 **if** i % 100 == 0:  
 print()  
 print("Точка " + str(i) + ": " +str(points\_random\_coords[i]) + ", ответ нейросети: класс " + str(classification))  
  
 **if** points\_random[i].classification == classification:  
 correctly += 1  
print()  
print("Точность персептрона: " + str(correctly) + "/1000")  
  
print()  
print("Пункт В")  
neuron = Adaline()  
network = Network()  
  
# обучаем нейросеть, изменяем веса  
**for** i **in** range(epochs):  
 errors = []  
 **for** j **in** range(len(points\_init)): #считаем взвеш сумму для каждого примера и корректируем веса  
 output = network.reLU(neuron.calculate(points\_coords[j]))  
 neuron.weight\_correction(points\_result\_class[j], output, points\_coords[j])  
  
points\_random = [Point((-0.5), 0.5) **for** i **in** range(1000)]  
points\_random\_coords = [[points\_random[i].x[0], points\_random[i].x[1]] **for** i **in** range(len(points\_random))]  
points\_random\_result = [points\_random[i].classification **for** i **in** range(len(points\_random))]  
  
print("Классификация точек после обучения:")  
  
correctly = 0  
**for** i **in** range(len(points\_random)):  
 output = network.reLU(neuron.calculate(points\_random\_coords[i]))  
  
 classification = (-1) **if** output < 0.5 **else** 1  
  
 **if** i % 100 == 0:  
 print()  
 print("Точка " + str(i) + ": " +str(points\_random\_coords[i]) + ", ответ нейросети: класс " + str(classification))  
  
 **if** points\_random[i].classification == classification:  
 correctly += 1  
print()  
print("Точность персептрона: " + str(correctly) + "/1000")



**Задача 3.** Требуется разработать и исследовать нейронную сеть обратного распространения, предназначенную для распознавания образов.

Даны в виде матрицы 3 х 3 4 латинские буквы X, Y, L, I.

Требуется:

1. Построить и обучить нейронную сеть, которая могла бы решать задачу распознавания символов.

2. Произвести тестирование нейронной сети при добавлении шума.

Программа имеет схожую структуру с предыдущими, но в этой задаче используется нейросеть с 4 нейронами, поскольку требуется классификация букв в 4 группы. Для этого каждая буква представлена в виде последовательности из 9 символов (ноликов и единичек), и у каждого нейрона есть 9 весов.

Нейросеть обучается на 4 тестовых примерах с заранее известным ответом, проходя несколько эпох обучения. После чего оценивается на исходных буквах и на буквах с шумом.

Код программы:

**import** random  
**import** math  
  
**class** Neuron:  
 w = []  
  
 **def** \_\_init\_\_(self):  
 self.w = [random.uniform(-1, 1) **for** i **in** range(9)]  
  
 **def** calculate(self, inputs):  
 b = 1  
 **for** i **in** range(9):  
 b += inputs[i] \* self.w[i]  
 **return** b  
  
 **def** weight\_correction(self, n, y, u, x\_in):  
 **for** i **in** range(len(self.w)):  
 self.w[i] -= n \* (u - y) \* x\_in[i]  
  
**class** Network:  
  
 **def** reLU(self, x):  
 **return** max(0, x)  
  
 **def** err(self, answer, output):  
 summ = 0  
  
 **for** i **in** range(len(answer)):  
 summ += answer[i] - output[i]  
  
 **return** summ / len(answer)  
  
**def** testing(x):  
 **for** i **in** range(len(x)):  
 outputs = []  
  
 **for** j **in** range(len(x)):  
 output = network.reLU(neurons[j].calculate(x[i]))  
 answer = 0 **if** output < 0.5 **else** 1  
 outputs.append(answer)  
  
 print(f"Для {i + 1}-ой буквы ответ: {outputs}")  
  
  
network = Network()  
neurons = [Neuron() **for** i **in** range(4)]  
  
n = 0.3  
inputs = [[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]]  
  
answers = [[0, 0, 0, 1],  
 [0, 0, 1, 0],  
 [0, 1, 0, 0],  
 [1, 0, 0, 0]]  
  
# обучение нейросети  
**for** i **in** range(1000): # эпохи  
 errors = []  
 **for** j **in** range(len(inputs)): # обучающие примеры  
 outputs = []  
  
 **for** k **in** range(len(neurons)): # нейроны  
 output = network.reLU(neurons[k].calculate(inputs[j]))  
 outputs.append(output)  
  
 error = network.err(answers[j], outputs)  
 errors.append(error)  
  
 # меняем веса у нейронов  
 **for** k **in** range(len(neurons)):  
 neurons[k].weight\_correction(n, answers[j][k], outputs[k], inputs[j])  
  
 **if** (i+1) % 100 == 0:  
 print(f"Эпоха обучения: {i+1}/1000, ошибка: {errors}")  
  
  
# проверка после обучения на обычных буквах  
print()  
print("Проверка на обыкновенных буквах:")  
testing(inputs)  
  
# проверка после обучения на буквах с шумами  
print()  
print("Проверка на буквах с шумами:")  
x\_loud = [[1, 1, 1, 0, 1, 0, 1, 0, 1], [1, 0, 1, 0, 1, 1, 0, 1, 0],  
 [0, 1, 0, 0, 1, 1, 0, 1, 0], [1, 0, 0, 1, 0, 0, 1, 1, 0]]  
testing(x\_loud)

