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

Направление подготовки: 02.03.03 Математическое обеспечение и администрирование информационных систем

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

1. Реализовать нейросеть, состоящую из 4ёх нейронов типа WTA, для классификации входных двухкомпонентных векторов. Вывести веса нейронов после обучения для коэффициента обучения 0,5.

from random import random  
from math import sqrt  
  
  
class Neuron:  
 def \_\_init\_\_(self):  
 self.nu = 0.5  
 self.w = [random(), random()] # Получаем веса  
 divider = sqrt(self.w[0] \*\* 2 + self.w[1] \*\* 2) # Нормируем веса  
 self.w[0] /= divider  
 self.w[1] /= divider  
  
 def calculate(self, x: list):  
 return self.w[0] \* x[0] + self.w[1] \* x[1]  
  
 def recalculate(self, x: list):  
 self.w[0] += self.nu \* (x[0] - self.w[0])  
 self.w[1] += self.nu \* (x[1] - self.w[1])  
  
  
class NeuralNetwork:  
 def \_\_init\_\_(self):  
 self.x = [ # Массив входных значений  
 [0.97, 0.2],  
 [1, 0],  
 [-0.72, 0.7],  
 [-0.67, 0.74],  
 [-0.8, 0.6],  
 [0, -1],  
 [0.2, -0.97],  
 [-0.3, -0.95]  
 ]  
  
 self.neurons = [Neuron() for i in range(4)]  
  
 def \_\_str\_\_(self):  
 s = ''  
 for neuron in self.neurons:  
 s += str(neuron.w) + '\n'  
 return s  
  
 def start(self, threshold\_number):  
 u = [0 for i in range(4)]  
 number\_wins = [0, 0, 0, 0]  
  
 for i in range(len(self.x)):  
 for j in range(4):  
 if number\_wins[j] < threshold\_number:  
 u[j] = self.neurons[j].calculate(self.x[i])  
 else:  
 u[j] -= 100  
 j = u.index(max(u))  
 self.neurons[j].recalculate(self.x[i])  
 number\_wins[j] += 1  
 print(j + 1)  
  
  
nn = NeuralNetwork()  
print(nn)  
nn.start(3)  
print(nn)

В классе нейрон у нас есть 3 метода:

В методе \_\_init\_\_() указываем коэффициент обучения n, берём случайные веса w и нормируем полученные веса.

В методе calculate() высчитываем индуцированное поле нейрона.

В методе recalute() пересчитываем значения весов в том случае, если данный нейрон победил в одном из раундов.

В классе NeuralNetwork есть 3 метода:

В методе \_\_init\_\_() объявляем 8 векторов с входными данными и создаём 4 нейрона.

В методе str() выводим значения весов нейронов.

В методе start() высчитываем победителей в каждом из 8 раундов. Если нейрон побеждает, то позволяем ему пересчитать значения своих весов и выводим номер победившего нейрона на экран.

1. Добавить к предыдущему заданию ограничение на количество побед нейронов, чтобы не создавалась проблема мёртвых нейронов.

Т.к у нас уже это было реализовано, что можно просто прокомментировать, как оно работает:

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

При вызове функции start мы передаём значение порога(т.е максимально допустимое количество побед для нейрона) и если данное условие нарушается, то мы записываем в массив u[] очень маленькое значение индуцированного поля для данного нейрона, чтобы этот нейрон не участвовал в сравнении с другими нейронами.

1. Для предыдущей задачи реализовать обучение НС с двумя нейронами по правилу Хебба.
2. import math  
   from random import random  
   from math import sqrt  
     
     
   class Neuron:  
    def \_\_init\_\_(self):  
    self.nu = 1  
    self.w = [random(), random()] # Получаем веса  
    divider = sqrt(self.w[0] \*\* 2 + self.w[1] \*\* 2) # Нормируем веса  
    self.w[0] /= divider  
    self.w[1] /= divider  
    self.v = 0 # Индуц. локальное поле нейрона  
     
    def calculate(self, x: list):  
    self.v = self.w[0] \* x[0] + self.w[1] \* x[1]  
    return self.v  
     
    def recalculate(self, x: list):  
    self.w[0] += self.nu \* x[0] \* self.activation\_func()  
    self.w[1] += self.nu \* x[1] \* self.activation\_func()  
     
    def activation\_func(self):  
    return 1/(1 + math.exp(-self.v))  
     
     
   class NeuralNetwork:  
    def \_\_init\_\_(self):  
    self.x = [ # Массив входных значений  
    [0.97, 0.2],  
    [1, 0],  
    [-0.72, 0.7],  
    [-0.67, 0.74],  
    [-0.8, 0.6],  
    [0, -1],  
    [0.2, -0.97],  
    [-0.3, -0.95]  
    ]  
     
    self.neurons = [Neuron() for i in range(2)]  
     
    def \_\_str\_\_(self):  
    s = ''  
    for neuron in self.neurons:  
    s += str(neuron.w) + '\n'  
    return s  
     
    def start(self, threshold\_number):  
    u = [0 for i in range(2)]  
    number\_wins = [0, 0]  
     
    for i in range(len(self.x)):  
    for j in range(2):  
    self.neurons[j].recalculate(self.x[i])  
    if number\_wins[j] < threshold\_number:  
    u[j] = self.neurons[j].calculate(self.x[i])  
    else:  
    u[j] -= 100  
    j = u.index(max(u))  
    self.neurons[j].recalculate(self.x[i])  
    number\_wins[j] += 1  
    print(j + 1)  
     
     
   nn = NeuralNetwork()  
   print(nn)  
   nn.start(8)  
   print(nn)

В 3-ем задании изменил функцию recalculate, в котором просчитывалось изменение веса нейронов, согласно правилу Хебба. Была добавлена функция активации (взял сигмоидальную), а также изменил кол-во нейронов в НС до 2ух.

Пример работы программы:

