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 выполнила: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Д. В. Яценко

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 – Функция активации ReLU***

1. **Описание решения**

Для решения задача созданы 2 класса – Neuron (класс нейрона) и Network (класс нейросети).

С помощью класса Neuron создаются объекты нейрона. У каждого нейрона есть поле входные сигналы и веса. В классе также есть методы сумматора и метод изменения весов.

С помощью класса Network создается нейросеть, которая инициализируется нейронами. Этот класс также содержит вспомогательные методы: функцию активации и функцию среднеквадратической ошибки mse.

Алгоритм решения задачи можно описать так:

1. Создаем нейросеть с одним нейроном
2. Обучаем нейрон на входных примерах (нейрон выдает результат, сравнивает с ожидаемым, считает ошибку и изменяет эти веса на основе ошибки)
3. Дальше обученный нейрон сам в состоянии определять классы точке, не обращая внимания на желаемый результат
4. **Реализация на языке Python**

from random import \*  
  
class Neuron:  
 w = []  
  
 def \_\_init\_\_(self):  
 """Create random weights"""  
 self.w = [uniform(-1, 1) for i in range(2)]  
  
 def summing(self, x):  
 summ = 1 # порог  
  
 for i in range(len(self.w)):  
 summ += self.w[i] \* x[i]  
  
 return summ  
  
 def change\_weights(self, n, answer, gradient):  
 """Меняем веса по правилу градиентного спуска"""  
 if answer == 1:  
 self.w[0] -= n \* (answer + gradient) \* (1 + gradient)  
 else:  
 self.w[1] -= n \* (answer + gradient) \* (1 + gradient)  
  
class Network:  
 def relu(self, summ):  
 return max(0, summ)  
  
 def mse(self, answer, real\_output):  
 """Функция среднеквадратичной ошибки, которую будем минимизировать"""  
 error = (answer - real\_output) \* (answer - real\_output)  
 return error  
  
  
  
  
sites = [[0, 0], [1, 1], [1, 0], [0, 1]]  
answers = [1, 1, 0, 0]  
n = 0.3  
  
neuron = Neuron()  
network = Network()  
  
# результаты нейросети до обучения  
print("Результаты работы до обучения:")  
for i in range(len(sites)):  
 output = network.relu(neuron.summing(sites[i]))  
  
 if output < 0.5:  
 print("Точка " + str(sites[i]) + " относится к классу 0")  
 else:  
 print("Точка " + str(sites[i]) + " относится к классу 1")  
  
# обучаем нейросеть, изменяем веса  
print()  
print("Обучение нейросети")  
for i in range(1000):  
 real\_outputs = []  
 errors = []  
  
 for j in range(len(sites)):  
 real\_output = network.relu(neuron.summing(sites[j]))  
 real\_outputs.append(real\_output)  
 error = network.mse(answers[j], real\_output)  
 errors.append(error)  
  
 gradient = []  
 for k in range(len(answers)):  
 gradient.append(real\_outputs[k] - answers[k])  
 # меняем веса  
 neuron.change\_weights(n, answers[k], gradient[k])  
  
 print("Эпоха обучения: " + str(i + 1) + " | Ошибки: " + str(errors))  
  
  
# результаты нейросети после обучения  
print()  
print("Результаты работы после обучения:")  
for i in range(len(sites)):  
 output = network.relu(neuron.summing(sites[i]))  
  
 if output < 0.5:  
 print("Точка " + str(sites[i]) + " относится к классу 0")  
 else:  
 print("Точка " + str(sites[i]) + " относится к классу 1")

1. **Запуск программы**

Для каждого входного вектора было несколько прогонок и изменений весов. В качестве примера приведен вывод для одного из векторов.

Изображение выглядит как текст, Шрифт, снимок экрана

Автоматически созданное описание

Изображение выглядит как текст, снимок экрана, Шрифт, дизайн

Автоматически созданное описание

Изображение выглядит как текст, Шрифт, снимок экрана

Автоматически созданное описание

***Задача 2 – Классификация точек***

1. **Описание решения**

Алгоритм решения этой задачи очень похож на алгоритм решения первой задачи. Только теперь нужно разделять множество точек на плоскости на 2 класса: первый класс – первая координата больше второй, второй класс – вторая координата больше первой.

Для такой классификации все еще достаточно одного нейрона. Поэтому в программе были созданы те же самые классы Neuron и Network. Но также был добавлен класс Neuron\_Adaline – для решения пункта б этой задачи. Этот тип нейрона отличается только функцией активации, а также подходом к изменению весов.

Также для большей простоты был создан класс точки Site, где у каждой точки хранились координаты, а также метка о том, к какому классу эта точка принадлежит.

Алгоритм можно описать так: веса нейрона инициализируются случайными величинами, затем нейрон обучается на множестве точек (это множество первой четверти координатной плоскости), затем нейрон тестируется на более широком множестве точек.

Так как задача разбита на 2 пункта, результаты решения нужно рассмотреть отдельно:

А) В первом пункте нейросеть обучается по методу обратного распространения ошибки. В результате обучения она показывает неплохие результаты.

Б) Во втором случае нейросеть обучалась по-другому и с другой функцией активации – сигнум. Благодаря такому изменению точность нейросети значительно возросла, она редко ошибается.

Таким образом, можно сделать вывод о том, что функция ReLU более универсальна, но в однослойных нейросетях лучше применять другие функции активации, так как они более гибкие.

1. **Реализация на языке Python**

from random import \*  
  
class Site:  
 x = []  
 clas = 0  
  
 def \_\_init\_\_(self, a, b, fl):  
 self.x = [uniform(a, b) for i in range(2)]  
  
 if fl:  
 self.clas = 1 if self.x[0] > self.x[1] else (-1)  
 else:  
 pass  
  
  
class Neuron\_Adaline:  
 w = []  
  
 def \_\_init\_\_(self):  
 """Create random weights"""  
 self.w = [uniform(-1, 1) for i in range(2)]  
  
 def summing(self, x):  
 summ = 1 # порог  
  
 for i in range(len(self.w)):  
 summ += self.w[i] \* x[i]  
  
 return summ  
  
  
 def change\_weights(self, n, y, u, x):  
 """Изменяем веса дискретным способом, по формуле"""  
 # тут уже не градиентный спуск  
 for i in range(len(self.w)):  
 self.w[i] -= n \* (u - y) \* x[i]  
  
class Neuron:  
 w = []  
  
 def \_\_init\_\_(self):  
 """Create random weights"""  
 self.w = [uniform(-1, 1) for i in range(2)]  
  
 def summing(self, x):  
 summ = 1 # порог  
  
 for i in range(len(self.w)):  
 summ += self.w[i] \* x[i]  
  
 return summ  
  
  
 def change\_weights(self, n, answer, gradient):  
 """Меняем веса по правилу градиентного спуска"""  
 if answer == 0:  
 self.w[0] -= n \* (answer + gradient) \* (1 + gradient)  
 else:  
 self.w[1] -= n \* (answer + gradient) \* (1 + gradient)  
  
class Network:  
 def relu(self, summ):  
 return max(0, summ)  
  
 def signum(self, x):  
 """Функция активации сигнум"""  
 if x > 0:  
 return 1  
 else:  
 return -1  
  
 def mse(self, answer, real\_output):  
 """Функция среднеквадратичной ошибки, которую будем минимизировать"""  
 error = (answer - real\_output) \* (answer - real\_output)  
 return error  
  
  
sites\_train = [Site(0, 0.5, True) for i in range(20)]  
sites\_train\_cords = [[sites\_train[i].x[0], sites\_train[i].x[1]] for i in range(len(sites\_train))]  
sites\_train\_answer = [sites\_train[i].clas for i in range(len(sites\_train))]  
n = 0.3  
  
print()  
print("<<<<<<<<<<<<<<<<< Task A >>>>>>>>>>>>>>>>>>>")  
neuron = Neuron()  
network = Network()  
  
# обучаем нейросеть, изменяем веса  
print()  
print("Обучение нейросети")  
  
for i in range(1000):  
 real\_outputs = []  
 errors = []  
  
 for j in range(len(sites\_train)):  
 real\_output = network.relu(neuron.summing(sites\_train\_cords[j]))  
 real\_outputs.append(real\_output)  
 error = network.mse(sites\_train\_answer[j], real\_output)  
 errors.append(error)  
  
 gradient = []  
 for k in range(len(sites\_train\_answer)):  
 gradient.append(real\_outputs[k] - sites\_train\_answer[k])  
 # меняем веса  
 neuron.change\_weights(n, sites\_train\_answer[k], gradient[k])  
  
 if (i+1) % 100 == 0:  
 print("Эпоха обучения: " + str(i + 1) + " | Ошибки: " + str(errors))  
  
print()  
sites\_random = [Site((-0.5), 0.5, True) for i in range(1000)]  
sites\_random\_cords = [[sites\_random[i].x[0], sites\_random[i].x[1]] for i in range(len(sites\_random))]  
sites\_random\_answer = [sites\_random[i].clas for i in range(len(sites\_random))]  
  
print("Классификация точек после обучения")  
  
rights = 0  
  
for i in range(len(sites\_random)):  
 output = network.relu(neuron.summing(sites\_random\_cords[i]))  
  
 clas = (-1) if output < 0.5 else 1  
  
 if i % 200 == 0:  
 print()  
 print("Точка " + str(sites\_random\_cords[i]) + ", класс: " + str(sites\_random\_answer[i]))  
 print("Ответ нейросети: " + str(clas))  
  
 if sites\_random[i].clas == clas:  
 rights += 1  
  
print("Точность обычного нейрона: " + str(rights) + "/1000")  
  
print()  
print()  
print("<<<<<<<<<<<<<<<<< Task B >>>>>>>>>>>>>>>>>>>")  
neuron = Neuron\_Adaline()  
network = Network()  
  
# обучаем наш нейрон и считаем ошибки  
for i in range(1000):  
 real\_outputs = []  
 errors = []  
  
 for j in range(len(sites\_train)):  
 real\_output = network.signum(neuron.summing(sites\_train\_cords[j]))  
 real\_outputs.append(real\_output)  
 error = network.mse(sites\_train\_answer[j], real\_output)  
 errors.append(error)  
 neuron.change\_weights(n, sites\_train\_answer[j], real\_output, sites\_train\_cords[j])  
  
  
 if (i + 1) % 100 == 0:  
 print("Эпоха обучения: " + str(i+1) + " | Ошибки: " + str(errors))  
  
print("Классификация точек после обучения")  
  
rights = 0  
  
for i in range(len(sites\_random)):  
 output = network.signum(neuron.summing(sites\_random\_cords[i]))  
  
 clas = (-1) if output < 0.5 else 1  
  
 if i % 200 == 0:  
 print()  
 print("Точка " + str(sites\_random\_cords[i]) + ", класс: " + str(sites\_random\_answer[i]))  
 print("Ответ нейросети: " + str(clas))  
  
 if sites\_random[i].clas == clas:  
 rights += 1  
  
print("Точность нейрона Адалайна: " + str(rights) + "/1000")

1. **Запуск программы**

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Изображение выглядит как текст, снимок экрана, Шрифт, дизайн

Автоматически созданное описание

***Задача 3 – Распознавание букв***

1. **Описание решения**

Структура программы похожа на предыдущие. Но отличие этой задачи в том, что в этой нейросети будет уже 4 нейрона. Это необходимо, так как в задаче требуется классификация на 4 группы (то есть 4 типа букв).

Каждая буква задается в виде последовательности из 9 символов (ноликов и единичек). Соответственно у каждого нейрона будет 9 весов.

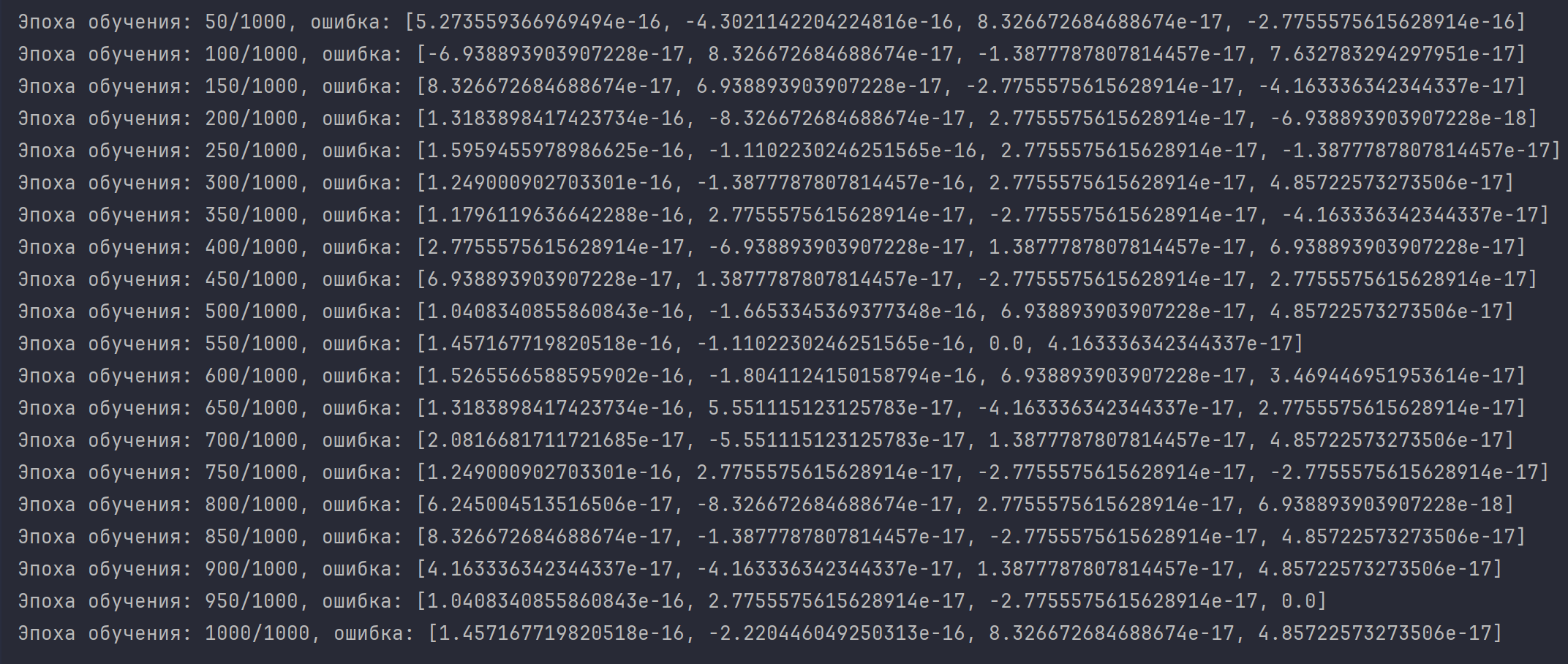
Для обучения такой нейросети был выбран принцип из задачи 2 пункта б, с помощью функции активации сингума.

Изображение выглядит как диаграмма, линия, круг, рисунок

Автоматически созданное описание

Сначала нейросеть обучается на 4 тестовых примерах с заранее известным ответом, проходя несколько эпох обучения. А затем мы тестируем ее на исходных буквах, и на нечетких (шумных) буквах.

1. **Реализация на языке Python**
2. # делаем сеть из 4 нейронов  
   import random  
   import math  
     
   class Neuron:  
    w = []  
     
    def \_\_init\_\_(self):  
    self.w = [random.uniform(-1, 1) for i in range(9)]  
     
    def summing(self, x):  
    summ = 1  
     
    for i in range(9):  
    summ += x[i] \* self.w[i]  
     
    return summ  
     
    def change\_weights(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 x  
     
    def mse(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].summing(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  
   x = [[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(x)): # обучающие примеры  
    outputs = []  
     
    for k in range(len(neurons)): # нейроны  
    output = network.relu(neurons[k].summing(x[j]))  
    outputs.append(output)  
     
    error = network.mse(answers[j], outputs)  
    errors.append(error)  
     
    # меняем веса у нейронов  
    for k in range(len(neurons)):  
    neurons[k].change\_weights(n, answers[j][k], outputs[k], x[j])  
     
    if (i+1) % 50 == 0:  
    print(f"Эпоха обучения: {i+1}/1000, ошибка: {errors}")  
     
     
   # проверка после обучения на обычных буквах  
   print()  
   print("Проверка на обыкновенных буквах:")  
   testing(x)  
     
   # 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)
3. **Запуск программы**

****

**Изображение выглядит как текст, снимок экрана, Шрифт, дизайн

Автоматически созданное описание**

**Вывод**

В ходе работы были выполнены 3 задания, касающиеся построения однослойных нейронных сетей. В первом и втором случае были созданы нейросети, состоящие из одного нейрона для классификации точек. Во втором задании было выявлено, что нейрон типа Адалайна, значительно повысил точность нейросети. А в третьем задании была написана нейросеть из 4 нейронов, которая достаточно точно распознает одну из 4 букв.