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: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ А. А. Иванов

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***

**Описание решения**

Задача 1: Функции активации.

Для решения этой задачи было создано 2 класса: Neuron и Neural\_Network. В классе Neuron представлен конструктор, сумматор и метод изменения весов по принципу градиентного спуска. В классе Neural\_Network используются методы функции активации ReLU и функции среднеквадратичной ошибки MSE.

Алгоритм решения задачи следующий: создаётся нейросеть с одним нейроном. Этот нейрон обучается на входных примерах (выдаёт результат, сравнивает с ожидаемым, высчитывает по разнице ошибку и изменяет веса на основе ошибки). Далее, нейрон сам определяет классы точек, не ссылаясь на ожидаемый результат.

**Реализация программы на языке Python**

*# План, что и как будем делать*

*# Создаём нейронку, в которой 1 нейрон*

*# Создаём случайные веса нейронам*

*# Выбираем функцию, которую будем минимизировать в процессе обучения. Возьмём среднеквадратичную ошибку*

*# Обучаем нейронку на предоставленных точках и сравниваем с желаемым выходом (классы 1 и 0)*

*# Применяем функцию ReLU к выходу нейрона*

*# Проверяем, как наша нейронка разделяет точки*

*# p - входные точки*

*# y - верные ответы (класс 1 для 1 и 2 точки, 0 - для 3 и 4)*

*# n - скорость обучения*

*# e - кол-во эпох обучения*

from random import \*

import math

class Neuron:

    w = []

    def \_\_init\_\_(*self*):

        """Создаём нейрону веса"""

*self*.w = [uniform(-1, 1) for i in range(2)]

    def summat(*self*, *x\_in*):

        """Сумматор нейрона (включая порог)"""

        u = 1  *# не забываем про порог*

        for i in range(len(*self*.w)):

            u += *self*.w[i] \* *x\_in*[i]

        return u

    def change\_w(*self*, *n*, *y*, *gradient*):

        """Изменение весов нейрона в зависимости от выхода (градиентный спуск)"""

*self*.w[0] -= *n* \* sum([(*y*[i] + *gradient*[i]) \* (1 + *gradient*[i]) for i in range(len(*y*)) if *y*[i] == 0])

*self*.w[1] -= *n* \* sum([(*y*[i] + *gradient*[i]) \* (1 + *gradient*[i]) for i in range(len(*y*)) if *y*[i] == 1])

class Neural\_Network:

    def relu(*self*, *u*):

        """Функция активации ReLU"""

        return max(0, *u*)

    def mse\_less(*self*, *y*, *u*):

        """Функция среднеквадратичной ошибки"""

        error = 0

        for i in range(len(*y*)):

            error += math.pow(*y*[i] - *u*[i], 2)

        return error / len(*y*)

*# По сути, у нас имеется по 2 примера на каждый желаемый выходной сигнал*

p = [[0, 0], [1, 1], [1, 0], [0, 1]]

y = [1, 1, 0, 0]

n = 0.3

e = 1000

neural\_network = Neural\_Network()

neuron = Neuron()

print("Классификация точек до обучения:")

for i in range(len(p)):

    neuron\_out = neural\_network.relu(neuron.summat(p[i]))

    class\_p = 0 if neuron\_out < 0.5 else 1

    print(f"Точке {p[i]} присвоен класс {class\_p}")

for i in range(e):

    errors = []

    neuron\_outs = []

    for j in range(len(p)):

        neuron\_out = neural\_network.relu(neuron.summat(p[j]))

        neuron\_outs.append(neuron\_out)

        error = neural\_network.mse\_less([y[j]], [neuron\_out])  *# Передаём данные в виде списков*

        errors.append(error)

    gradient = []

    for k in range(len(y)):

        gradient.append(neuron\_outs[k] - y[k])

    neuron.change\_w(n, y, gradient)

    print(f"Эпоха обучения: {i + 1}/{e}, Ошибка: {errors}")  *# Мы получаем ошибки 1.0 и 0.0 из-за функции ReLU*

print("Классификация точек после обучения:")

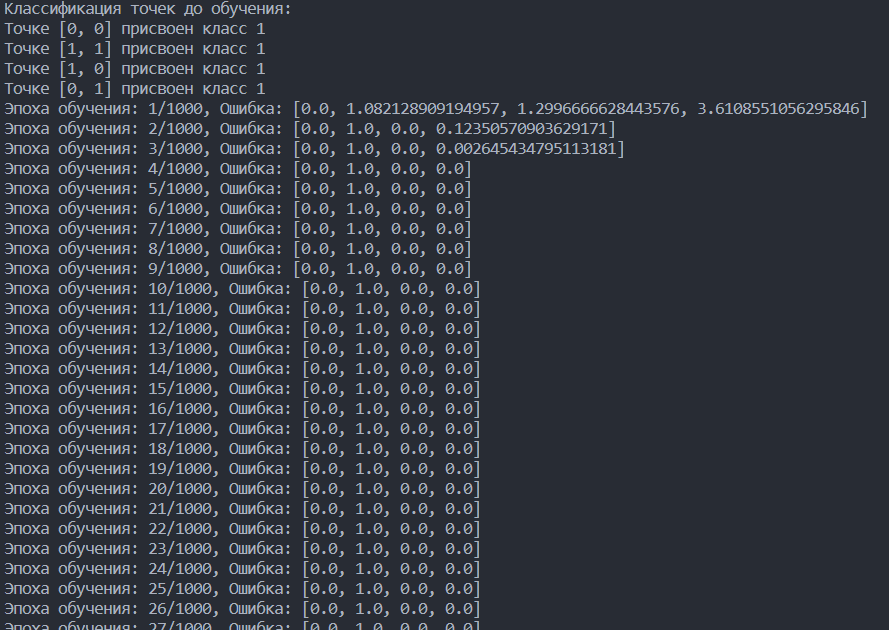
for i in range(len(p)):

    neuron\_out = neural\_network.relu(neuron.summat(p[i]))

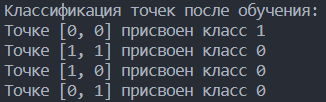
    class\_p = 0 if neuron\_out < 0.5 else 1

    print(f"Точке {p[i]} присвоен класс {class\_p}")

**Примеры запусков**



**Выводы программы**



***Задача 1***

**Описание решения**

Задача 2: Классификация точек.

Для решения задачи созданы вышеуказанные 2 класса, Neuron и Neural\_Network. Однако, появились ещё классы Point и Neuron\_Adaline. Класс Point отвечает за формирование точек в пространстве, а класс Neuron\_Adaline предназначен для реализации иной логики изменения весов (не основе другой функции активации).

Алгоритм следующий: на стартовых точках нейросеть обучается, а затем тестируется на новых точках, созданных по тому же алгоритму, что и стартовые, но в более широкой области. Рассмотри отдельно пункты А) и Б).

А) Нейросеть создана по модели обратного распространения ошибки, обучаема и показывает приемлемые результаты.

Б) Из-за изменения логики работы нейрона меняется и поведение самой нейросети: из-за линейной функции активации сеть стала гибче классифицировать точки и показывает более лучшую точность.

Можно сделать следующий вывод: функция ReLU является куда более универсальной функцией, но она не подходит для использования в однослойных нейросетях из-за отсутсвия дополнительных слоёв.

**Реализация программы на языке Python**

*# Имеется двухмерный набор данных, то есть как и раньше, 2 веса для двухкомпонентных точек*

*# Определяем принадлежность к классу так: x1 > x2 -> 1, x1 < x2 -> -1*

*# Есть какая-то гиперплоскость (линия, уравнение которой: x1 - x2 = 0)*

*# Единичный квадрат - квадрат со стороной = 1, центрированный относительно начала координат ->*

*# -> положительный квадрант такого квадрата - область по Х и по У [0, 0.5]*

*# Для точек сделать метку о принадлежности к классу*

*# А) Обучаем на 20 точках -> тестируем на 1000 точках (по всему квадрату)*

*# Б) Сделать то же самое, но с нейроном типа адалайна (адаптивный линейный элемент) (какой-то дискретный случай)*

*# points\_base - обучающие точки*

*# array\_points\_base - список, который содержит только пары координат точек*

*# array\_answer - массив желаемых выходов по обучаемым точкам*

*# n - скорость обучения*

*# e - кол-во эпох обучения*

*# points\_new - тестируемые точки*

*# Краткий вывод: нейрон типа адалайна показал лучшую точность. Возможно, из-за линейной функции активации*

from random import \*

import math

class Point:

    x = []

    class\_p = 0

    def \_\_init\_\_(*self*, *digit1*, *digit2*, *flag*):

        """Создаём точку в заданной области"""

*self*.x = [uniform(*digit1*, *digit2*) for i in range(2)]

        if *flag*: *self*.class\_p = 1 if *self*.x[0] > *self*.x[1] else (-1)

        else: pass

    def show(*self*):

        """Вспомогательная функция"""

        print(f"x1 = {*self*.x[0]}, x2 = {*self*.x[1]}, class\_p = {*self*.class\_p}")

class Neuron:

    w = []

    def \_\_init\_\_(*self*):

        """Создаём нейрону веса и передаём точку"""

*self*.w = [uniform(-1, 1) for i in range(2)]

    def summat(*self*, *x\_in*):

        """Сумматор нейрона (включая порог)"""

        u = 1  *# не забываем про порог*

        for i in range(len(*self*.w)):

            u += *self*.w[i] \* *x\_in*[i]

        return u

    def change\_w(*self*, *n*, *y*, *gradient*):

        """Изменение весов нейрона в зависимости от выхода (градиентный спуск)"""

*self*.w[0] -= *n* \* sum([(*y*[i] + *gradient*[i]) \* (1 + *gradient*[i]) for i in range(len(*y*)) if *y*[i] == 0])

*self*.w[1] -= *n* \* sum([(*y*[i] + *gradient*[i]) \* (1 + *gradient*[i]) for i in range(len(*y*)) if *y*[i] == 1])

class Neuron\_Adaline:

    w = []

    def \_\_init\_\_(*self*):

        """Создаём нейрону веса и передаём точку"""

*self*.w = [uniform(-1, 1) for i in range(2)]

    def summat(*self*, *x\_in*):

        """Сумматор нейрона (включая порог)"""

        u = 1  *# не забываем про порог*

        for i in range(len(*self*.w)):

            u += *self*.w[i] \* *x\_in*[i]

        return u

    def activation(*self*, *u*):

        """Функция активации - линейная функция"""

        return *u*

    def change\_w(*self*, *n*, *y*, *u*, *x\_in*):

        """Изменение весов нейрона в зависимости от выхода (не градиентный спуск)"""

        for i in range(len(*self*.w)):

*self*.w[i] -= *n* \* (*u* - *y*) \* *x\_in*[i]

class Neural\_Network:

    def relu(*self*, *u*):

        """Функция активации ReLU"""

        return max(0, *u*)

    def mse\_less(*self*, *y*, *u*):

        """Функция среднеквадратичной ошибки"""

        error = 0

        for i in range(len(*y*)):

            error += math.pow(*y*[i] - *u*[i], 2)

        return error / len(*y*)

points\_base = [Point(0, 0.5, True) for i in range(20)]

array\_points\_base = [[points\_base[i].x[0], points\_base[i].x[1]] for i in range(len(points\_base))]

array\_answer = [points\_base[i].class\_p for i in range(len(points\_base))]

n = 0.3

e = 1000

print("--------------------Пункт А--------------------")

neural\_network = Neural\_Network()

neuron = Neuron()

for i in range(e):

    errors = []

    neuron\_outs = []

    for j in range(len(array\_points\_base)):

        neuron\_out = neural\_network.relu(neuron.summat(array\_points\_base[j]))

        neuron\_outs.append(neuron\_out)

        error = neural\_network.mse\_less([array\_answer[j]], [neuron\_out])  *# Передаём данные в виде списков*

        errors.append(error)

    gradient = []

    for k in range(len(array\_answer)):

        gradient.append(neuron\_outs[k] - array\_answer[k])

    neuron.change\_w(n, array\_answer, gradient)

    if i % 50 == 0:

        print(f"Эпоха обучения: {i}/{e}, Ошибка: {errors}")  *# Мы получаем ошибки 1.0 и 0.0 из-за функции ReLU*

print()

points\_new = [Point((-0.5), 0.5, True) for i in range(1000)]

array\_points\_new = [[points\_new[i].x[0], points\_new[i].x[1]] for i in range(len(points\_new))]

print("Классификация точек после обучения:")

kol = 0

for i in range(len(points\_new)):

    neuron\_out = neural\_network.relu(neuron.summat(array\_points\_new[i]))

    class\_p = (-1) if neuron\_out < 0.5 else 1

    if i % 50 == 0:

        print(f"Точке {i} присвоен класс {class\_p}")

        print(f"Информация самой точки: x1 = {points\_new[i].x[0]}, x2 = {points\_new[i].x[1]}, class\_p = {points\_new[i].class\_p}")

    if class\_p == points\_new[i].class\_p:

        kol += 1

print(f"Итоговая точность: {kol}/1000")

print()

print("--------------------Пункт Б--------------------")

neural\_network = Neural\_Network()

neuron = Neuron\_Adaline()

for i in range(e):

    errors = []

    neuron\_outs = []

    for j in range(len(array\_points\_base)):

        neuron\_out = neuron.activation(neuron.summat(array\_points\_base[j]))

        neuron\_outs.append(neuron\_out)

        error = neural\_network.mse\_less([array\_answer[j]], [neuron\_out])  *# Передаём данные в виде списков*

        errors.append(error)

        neuron.change\_w(n, array\_answer[j], neuron\_out, array\_points\_base[j])

    if i % 50 == 0:

        print(f"Эпоха обучения: {i}/{e}, Ошибка: {errors}")

print()

print("Классификация точек после обучения:")

kol = 0

for i in range(len(points\_new)):

    neuron\_out = neuron.activation(neuron.summat(array\_points\_new[i]))

    class\_p = (-1) if neuron\_out < 0.5 else 1

    if i % 50 == 0:

        print(f"Точке {i} присвоен класс {class\_p}")

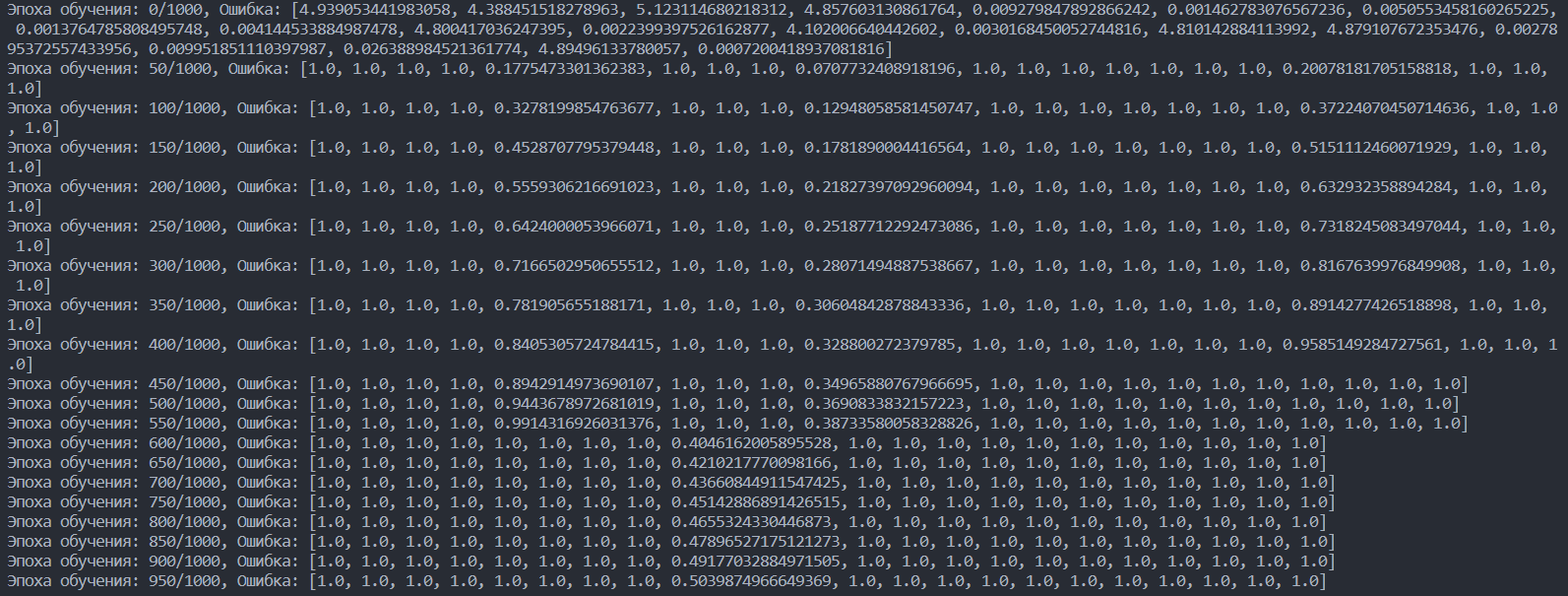
        print(f"Информация самой точки: x1 = {points\_new[i].x[0]}, x2 = {points\_new[i].x[1]}, class\_p = {points\_new[i].class\_p}")

    if class\_p == points\_new[i].class\_p:

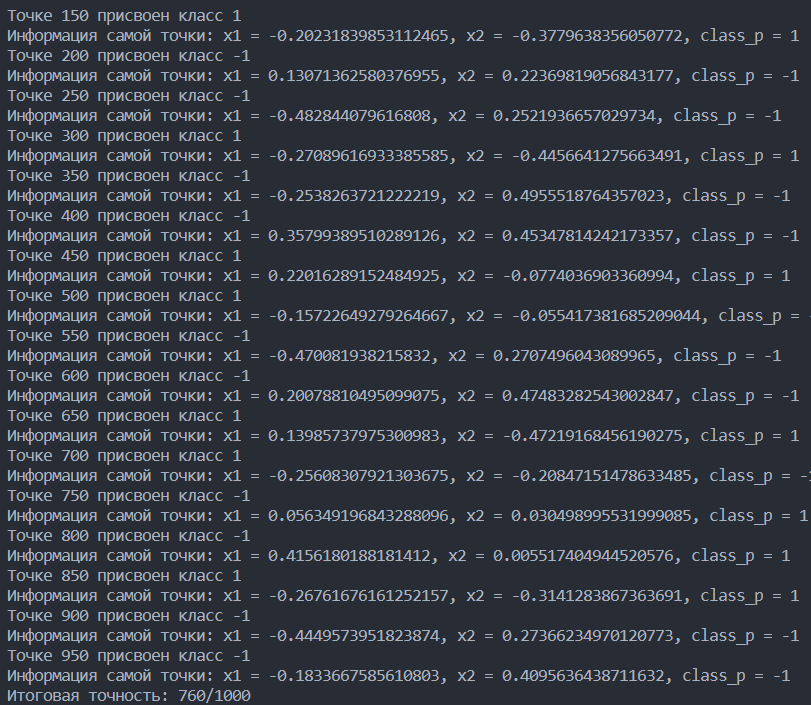
        kol += 1

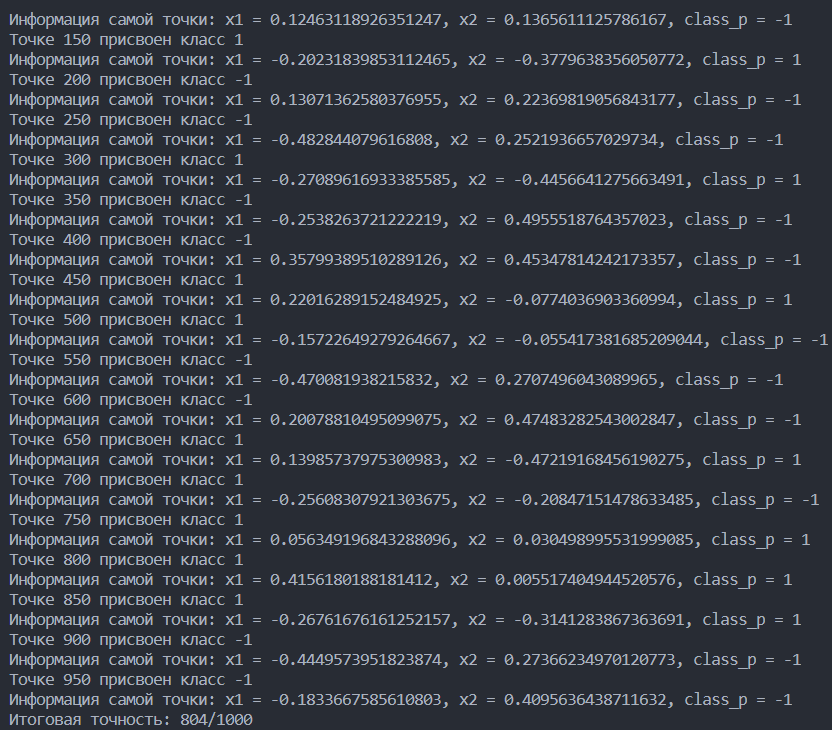
print(f"Итоговая точность: {kol}/1000")

**Примеры запусков**



**Выводы программы**





***Задача 1***

**Описание решения**

Задача 3: Распознавание букв.

Для решения этой задачи используются те же классы Neuron и Neural\_Network.

Алгоритм решения следующий: обучаем сеть на входных буквах, сравнивая выход нейронов с желаемым, изменяем им веса. Тут особое внимание нужно обратить на формирование выхода нейронов, так как тут мы сравниваем совокупность ответов, которую может испортить всего лишь 1 нейрон. Это сильно усложняет задачу классификации, так как для однослойной сети, да ещё и с функцией ReLU возникает сильная погрешность вычислений, которая влияет на оценку итоговой точности.

После обучения тестируем сеть и на исходных буквах, не ссылаясь на желаемый отклик, и на «шумных» буквах.

**Реализация программы на языке Python**

*# Будем делать сеть из 4 нейронов, то есть буквам будут следующие желаемые выходы: ->*

*# -> X - 0001, Y - 0010, I - 0100, L - 1000, то есть комбинация ответов нейронов должна выдавать одну букву*

*# Обучаем на данных 4 буквах, затем тестируем с добавлением шума и смотрим, что по чём*

*# Возьмём нейроны как из прошлой задачи, функцию ReLU, подгонку весов через градиент*

*# n - скорость обучения*

*# e - кол-во эпох обучения*

*# x - массив букв*

*# y - желаемые выходы по буквам*

import random

import math

class Neuron:

    w = []

    def \_\_init\_\_(*self*):

        """Создание весов нейрона"""

*self*.w = [random.uniform(-1, 1) for i in range(9)]

    def summat(*self*, *x\_in*):

        """Сумматор нейрона"""

        return 1 + sum([(*self*.w[i] \* *x\_in*[i]) for i in range(len(*self*.w))])

    def change\_w(*self*, *n*, *x\_in*, *error*):

        """Изменение весов методом градиентного спуска"""

*# самая большая сложность - оказалась тут*

*# суть в том, что веса изменяются на приращение, где приращение - n\_in \* p(w), ->*

*# -> где p(w) - частная производная e/w\_ij со знаком минус, где e - 1/2 \* sum((y\_in - u\_in)^2)*

*# P.S. Взято из книги Солдатовой О.П. "Нейроинформатика"*

*self*.w = [w + *n* \* *error* \* x for w, x in zip(*self*.w, *x\_in*)]

class Neural\_Network:

    def relu(*self*, *u\_in*):

        """Функция активации ReLU"""

        return max(0, *u\_in*)

*#return 1 / (1 + math.exp(-u\_in))*

    def mse\_less(*self*, *y\_in*, *u\_in*):

        """Функция среднеквадратичной ошибки"""

*#  тут возникает ошибка слишком большого числа, возможно, из-за функции ReLU*

*#return (1 / len(y\_in)) \* sum([math.pow((y\_in[i] - u\_in[i]), 2) for i in range(len(u\_in))])*

        return (1 / len(*y\_in*)) \* sum([(*y\_in*[i] - *u\_in*[i]) for i in range(len(*u\_in*))])

def test(*x\_in*):

    for i in range(len(*x\_in*)):

        neurons\_out = []

        for j in range(len(neurons)):

            neuron\_out = neural\_network.relu(neurons[j].summat(*x\_in*[i]))

            answer = 0 if neuron\_out < 0.5 else 1

            neurons\_out.append(answer)

        print(f"Для {i + 1}-ой буквы ответ: {neurons\_out}")

neural\_network = Neural\_Network()

neurons = [Neuron() for i in range(4)]

n = 0.3

e = 1000

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]]

y = [[0, 0, 0, 1], [0, 0, 1, 0], [0, 1, 0, 0], [1, 0, 0, 0]]

*# --------------------Обучение--------------------*

for i in range(e):  *# цикл по эпохам*

    for j in range(len(x)):  *# цикл по примерам*

        neurons\_out = []

        for k in range(len(neurons)):  *# цикл по нейронам*

            neuron\_out = neural\_network.relu(neurons[k].summat(x[j]))

            neurons\_out.append(neuron\_out)

        error = neural\_network.mse\_less(y[j], neurons\_out)

        for k in range(len(neurons)):

            neurons[k].change\_w(n, x[j], error)

    if i % 50 == 0:

        print(f"Эпоха обучения: {i}/{e}, ошибка: {error}")

*# --------------------Проверка на обычных буквах--------------------*

print()

print("--------------------Проверка на обычных буквах--------------------")

test(x)

*# --------------------Проверка на <шумных> буквах--------------------*

print()

print("--------------------Проверка на <шумных> буквах--------------------")

x\_new = [[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]]

test(x\_new)

**Примеры запусков**



**Выводы программы**

