МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ

РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ «САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ МОРСКОЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»  
 (СПБГМТУ)

| ФАКУЛЬТЕТ ЦИФРОВЫХ ПРОМЫШЛЕННЫХ ТЕХНОЛОГИЙ  КАФЕДРА КИБЕРФИЗИЧЕСКИХ СИСТЕМ |
| --- |

ЛАБОРАТОРНАЯ РАБОТА №2

«ПРОЕКТИРОВАНИЕ НЕЙРОННОЙ СЕТИ НА ПЕРСЕПТРОНАХ

РОЗЕНБЛАТТА»  
ВАРИАНТ 8

|  | |
| --- | --- |
|  | Выполнил  студент группы 20221 | | |
|  | Лаптев Иван Александрович | | |
|  | Проверила | | |
|  | Кайнова Татьяна Денисовна | | |
| Санкт-Петербург  2024 год | | |

**СОДЕРЖАНИЕ**

[Цели работы………………………………………………………………………. 2](#_gjdgxs)

[Результат работы………………………………………………………………….. 3](#_1fob9te)

[Заключение………………………………………………………………………... 7](#_2s8eyo1)

[Листинг кода……………………………………………………………………… 8](#_17dp8vu)

# Цели работы

Цель состоит в разработке нейронной сети на основе персептронов Розенблатта.

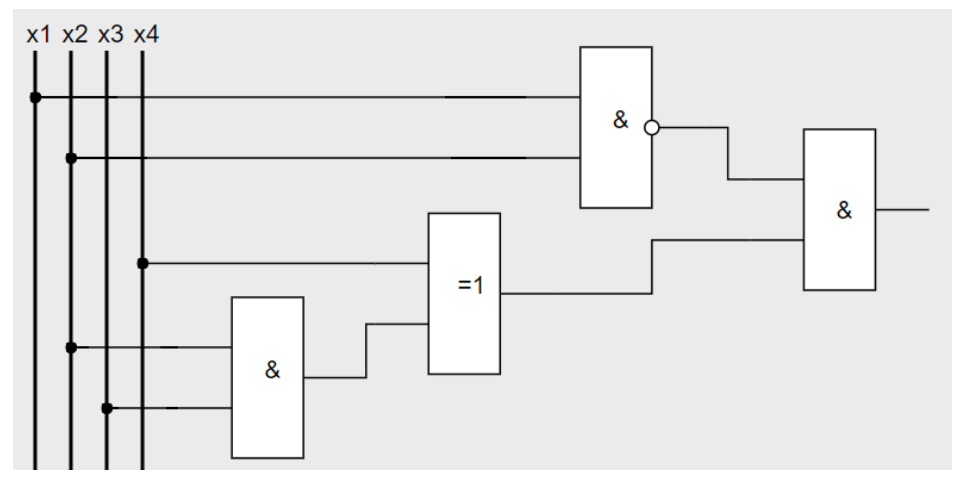
В процессе выполнения данного проекта будут затронуты следующие аспекты:

1. Создание таблицы истинности, содержащей состояния выходов промежуточных компонентов схемы.
2. Построение схемы нейронной сети, которая будет реализовывать полученную таблицу истинности с использованием персептронов.
3. Обучение отдельных модулей нейронной сети, отвечающих за выполнение функций соответствующих логических элементов схемы.
4. Создание диаграмм при помощи библиотеки matplotlib.
5. Проверка правильной работы нейронной сети на тестовых входных данных, включая все варианты исходной таблицы истинности, с выводом таблицы истинности, аналогичной структуре исходной.

# Результат работы

В результате работы, для логической схемы (Рисунок 1), были получены: таблица истинности (Таблица 1), включающая состояния выходов; схема нейронной сети (Рисунок 2), реализующая полученную таблицу истинности с помощью персептронов; алгоритм (Листинг 1), обучающий отдельные блоки нейронной сети, отвечающих за реализацию функций соответствующих логических элементов схемы и таблица истинности, выводимая нейронной сетью (Рисунок 3).

Полученные результаты были обработаны путем построения нескольких графиков с помощью библиотеки matplotlib (Рисунок 4, Рисунок 5).

Рисунок 1 - Исходные данные

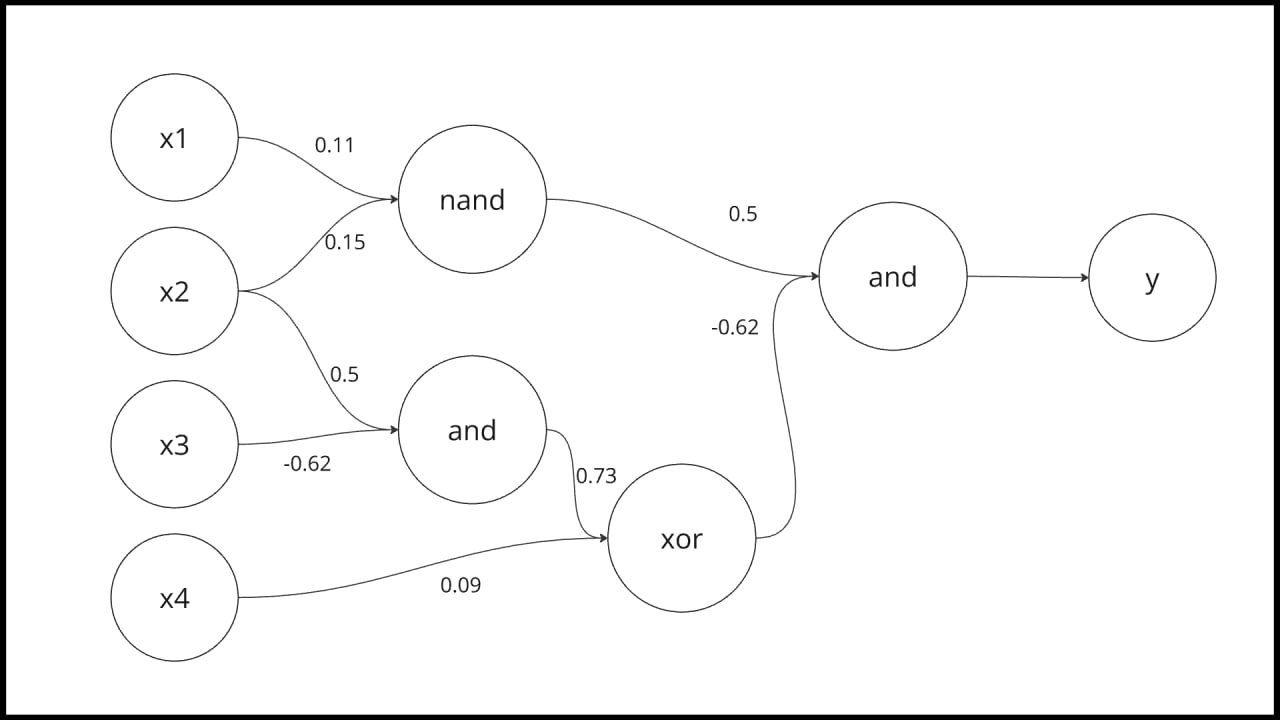


Рисунок 2 – Схема нейронной сети

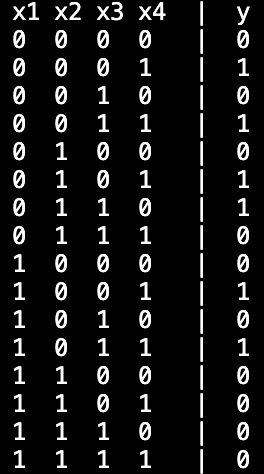
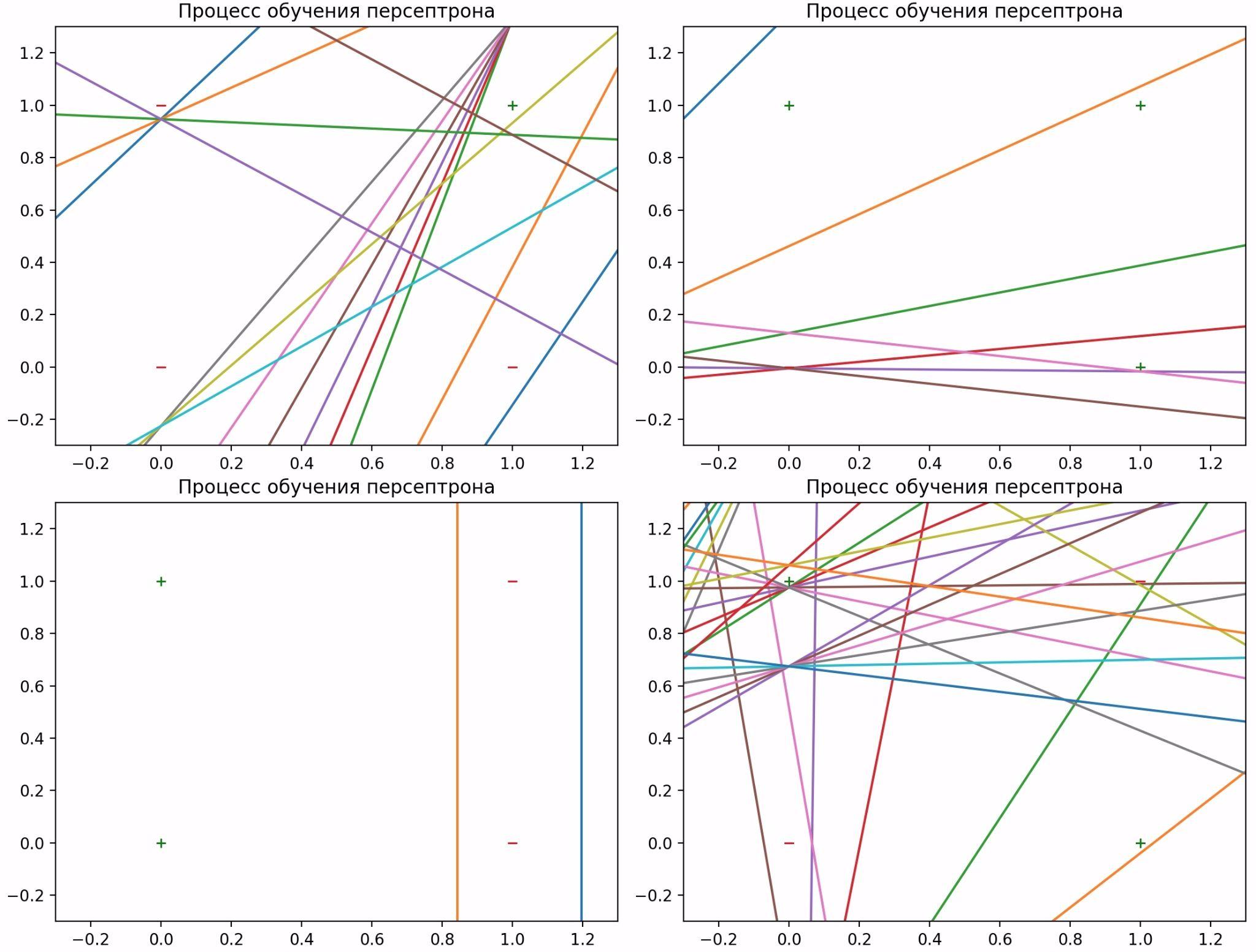


Рисунок 3 - Таблица, выводимая в консоль

Рисунок 4 – Процесс обучения

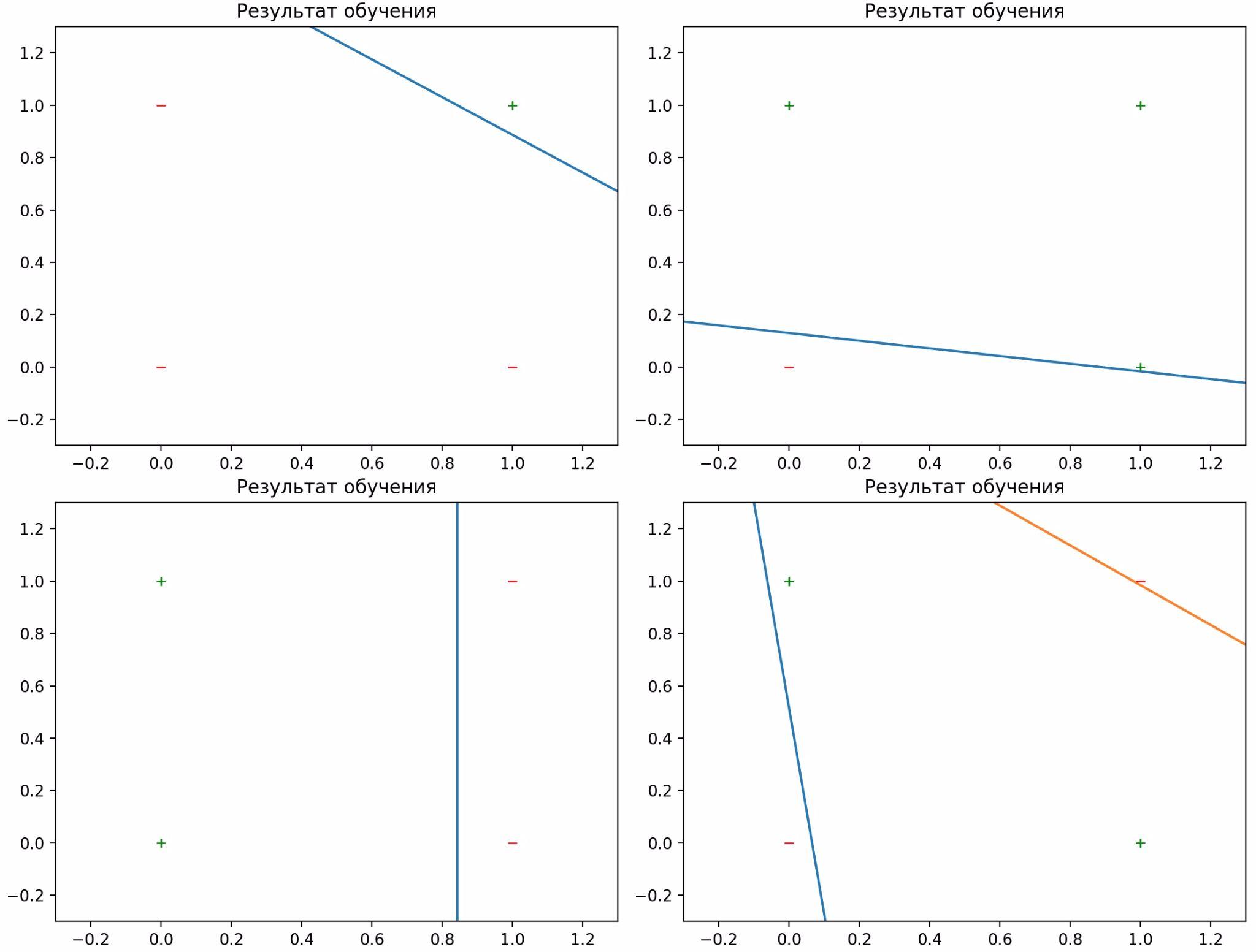
**Рисунок 5 - Конечные значения весов после обучения

Таблица 1 - Таблица истинности

| X1 | X2 | X3 | X4 | X1∧X2 | ¬X1∧X2 | X2∧X3 | X4⊕X2∧X3 | ¬X1∧X2∧(X4⊕X2∧X3) |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 1 |
| 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 0 | 1 | 0 | 1 | 1 |
| 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 1 |
| 0 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 1 |
| 0 | 1 | 1 | 1 | 0 | 1 | 1 | 0 | 0 |
| 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| 1 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 1 |
| 1 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 1 | 0 | 1 | 0 | 1 | 1 |
| 1 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| 1 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 0 |
| 1 | 1 | 1 | 0 | 1 | 0 | 1 | 1 | 0 |
| 1 | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 0 |

# Заключение

В ходе лабораторной работы был получен алгоритм, решающий поставленную задачу, к тому же были получен опыт работы со следующими вопросами:

1. Составление таблицы истинности, включающую состояния выходов промежуточных элементов схемы.

2. Составление схемы нейронной сети, реализующую полученную таблицу истинности с помощью персептронов.

3. Обучение отдельных блоков нейронной сети, отвечающих за реализацию функций соответствующих логических элементов схемы.

4. Создание диаграмм при помощи библиотеки matplotlib.

5. Проверка корректной работы нейронной сети на тестовых входных данных — всех вариантах исходной таблицы истинности, с выводом таблицы истинности, аналогичной по структуре исходной таблице.

# Листинг кода

Листинг 1 – Алгоритм

import numpy as np

import random

import matplotlib.pyplot as plt

random.seed(52)

LEARNING\_RATE = 0.1

# Класс для реализации перцептрона

class SimplePerceptron:

def \_\_init\_\_(self, input\_count: int):

self.weights = [random.uniform(-1, 1) for \_ in range(input\_count)]

def evaluate(self, inputs: list):

weighted\_sum = np.dot(inputs, self.weights)

return 0 if np.sign(weighted\_sum) == -1 else 1

def train(self, input\_set: list, expected\_output: list):

index\_list = list(range(len(input\_set)))

correctly\_classified = False

self.weight\_history = list()

while not correctly\_classified:

self.weight\_history.append(self.weights.copy())

correctly\_classified = True

random.shuffle(index\_list)

for idx in index\_list:

x = input\_set[idx]

y = expected\_output[idx]

output = self.evaluate(x)

if output != y:

correctly\_classified = False

for j in range(0, len(self.weights)):

weight\_adjustment = x[j] \* LEARNING\_RATE

if y == 0:

weight\_adjustment \*= -1

self.weights[j] += weight\_adjustment

class XORGate:

def \_\_init\_\_(self):

self.perceptrons = [SimplePerceptron(3) for \_ in range(3)]

self.training\_history = [list() for \_ in range(3)]

def evaluate(self, inputs: list):

return self.perceptrons[2].evaluate([1, self.perceptrons[1].evaluate(inputs), self.perceptrons[0].evaluate(inputs)])

def train(self):

self.perceptrons[0].train([(1, 0, 0), (1, 0, 1), (1, 1, 0), (1, 1, 1)], [0, 1, 1, 1]) # OR

self.perceptrons[1].train([(1, 0, 0), (1, 0, 1), (1, 1, 0), (1, 1, 1)], [1, 1, 1, 0]) # NOT AND

self.perceptrons[2].train([(1, 0, 0), (1, 0, 1), (1, 1, 0), (1, 1, 1)], [0, 0, 0, 1]) # AND

self.training\_history[0] = self.perceptrons[0].weight\_history

self.training\_history[1] = self.perceptrons[1].weight\_history

self.training\_history[2] = self.perceptrons[2].weight\_history

# Создание перцептронов и обучение для различных логических операций

and\_perceptron = SimplePerceptron(3)

and\_perceptron.train([(1, 0, 0), (1, 0, 1), (1, 1, 0), (1, 1, 1)], [0, 0, 0, 1])

or\_perceptron = SimplePerceptron(3)

or\_perceptron.train([(1, 0, 0), (1, 0, 1), (1, 1, 0), (1, 1, 1)], [0, 1, 1, 1])

not\_perceptron = SimplePerceptron(2)

not\_perceptron.train([(1, 0), (1, 1)], [1, 0])

xor\_perceptron = XORGate()

xor\_perceptron.train()

# Функция для отрисовки графика для логических операций

def plot\_logic\_function(weights, logic\_func=None):

x\_max = 1 + 0.3

x\_min = 0 - 0.3

y\_min = -weights[1] \* x\_min / weights[2] - weights[0] / weights[2]

y\_max = -weights[1] \* x\_max / weights[2] - weights[0] / weights[2]

plt.axis([-0.3, 1.3, -0.3, 1.3])

if logic\_func is not None:

for y in (0, 1):

for x in (0, 1):

if logic\_func([1, x, y]) == 1:

plt.plot(x, y, "g+")

else:

plt.plot(x, y, "r\_")

plt.plot((x\_min, x\_max), (y\_min, y\_max))

# Функция для отрисовки графика для логической операции NOT

def plot\_not\_logic\_function(weights, logic\_func=None):

x\_boundary = -weights[0] / weights[1]

y\_min = -0.3

y\_max = 1.3

plt.axis([-0.3, 1.3, -0.3, 1.3])

if logic\_func is not None:

for y in (0, 1):

for x in (0, 1):

if logic\_func([1, x]) == 1:

plt.plot(x, y, "g+")

else:

plt.plot(x, y, "r\_")

plt.plot((x\_boundary, x\_boundary), (y\_min, y\_max))

# Отрисовка графиков для каждой логической операции

for weights in and\_perceptron.weight\_history:

plot\_logic\_function(weights, and\_perceptron.evaluate)

plt.title ("Процесс обучения персептрона")

plt.show()

plot\_logic\_function(and\_perceptron.weights, and\_perceptron.evaluate)

plt.title ("Результат обучения")

plt.show()

for weights in or\_perceptron.weight\_history:

plot\_logic\_function(weights, or\_perceptron.evaluate)

plt.title ("Процесс обучения персептрона")

plt.show()

plot\_logic\_function(or\_perceptron.weights, or\_perceptron.evaluate)

plt.title ("Результат обучения")

plt.show()

for weights in not\_perceptron.weight\_history:

plot\_not\_logic\_function(weights, not\_perceptron.evaluate)

plt.title ("Процесс обучения персептрона")

plt.show()

plot\_not\_logic\_function(not\_perceptron.weights, not\_perceptron.evaluate)

plt.title ("Результат обучения")

plt.show()

for perceptron\_weights in xor\_perceptron.training\_history:

for weights in perceptron\_weights:

plot\_logic\_function(weights, xor\_perceptron.evaluate)

plt.title ("Процесс обучения персептрона")

plt.show()

plot\_logic\_function(xor\_perceptron.perceptrons[0].weights, xor\_perceptron.evaluate)

plot\_logic\_function(xor\_perceptron.perceptrons[1].weights, xor\_perceptron.evaluate)

plt.title ("Результат обучения")

plt.show()

# Новая сложная логическая функция ¬X1∧X2∧(X4⊕X2∧X3)

def logic\_function(x1, x2, x3, x4):

and\_result = and\_perceptron.evaluate([1, x2, x3])

xor\_result = xor\_perceptron.evaluate([1, x4, and\_result])

and\_result2 = and\_perceptron.evaluate([1, x1, x2])

not\_result = not\_perceptron.evaluate([1, and\_result2])

and\_final\_result = and\_perceptron.evaluate([1, not\_result, xor\_result])

return and\_final\_result

# Проверка работы

print("x1 x2 x3 x4 | y")

for x1 in (0, 1):

for x2 in (0, 1):

for x3 in (0, 1):

for x4 in (0,1):

print(f"{x1} {x2} {x3} {x4} ", " | ", logic\_function(x1, x2, x3, x4))