|  |  |
| --- | --- |
| Picture 1 | **МИНОБРНАУКИ РОССИИ**  **федеральное государственное бюджетное образовательное учреждение**  **высшего образования**  «Санкт-Петербургский государственный морской технический университет» (СПбГМТУ) |

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Факультет цифровых промышленных технологий

Направление подготовки 09.03.01

"**Интеллектуальные технологии киберфизических систем**"

Лабораторная работа №2

Вариант 14

Студент 2 курса группы 20221

Очного отделения

Руденко Вячеслав Сергеевич

Проверил:

Преподаватель CПбГМТУ

Кайнова Татьяна Денисовна

Санкт-Петербург

2024

Оглавление

[1. ОБЩАЯ ЧАСТЬ 3](#_Toc182216387)

1. [1.1 Цель работы 3](#_Toc182216388)
2. [1.2 Формулировка задачи 3](#_Toc182216389)
3. [1.3 Исходные данные 3](#_Toc182216390)

[2. ХОД РАБОТЫ 4](#_Toc182216391)

[СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ 15](#_Toc182216392)

[ПРИЛОЖЕНИЕ А 16](#_Toc182216393)

# 1. ОБЩАЯ ЧАСТЬ

## 1.1 Цель работы

Реализовать и обучить персептронную модель, способную выполнять логическую функцию, заданную на блок-схеме. Программа должна корректно реализовать логическую структуру функции и отобразить итоговые значения весов после обучения, а также визуализировать разделяющие линии для каждого логического оператора.

## 1.2 Формулировка задачи

Использовать несколько персептронов для поэтапного выполнения операций, представленных на блок-схеме, с целью получения заданной логической зависимости между переменными. Каждый персептрон должен быть обучен на наборе данных, соответствующем его логической функции, и участвовать в построении общей логической структуры, показанной на блок-схеме.

## 1.3 Исходные данные

Исходные данные 14 варианта представлены на рисунке 1.

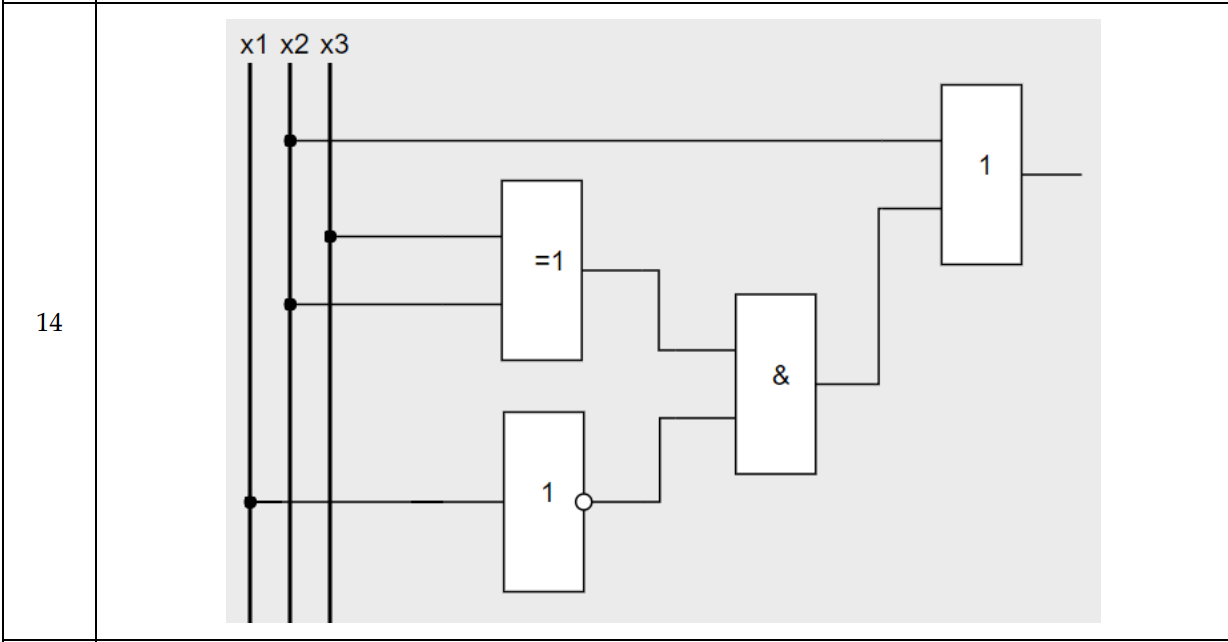


Рисунок 1 – Исходные данные

# 2. ХОД РАБОТЫ

Преобразуем блок схему (рис. 1) в логическую функцию:

(¬X1∧(X2⊕X3))∨X2=

Таким образом, мы избавились от XOR.

Далее преобразуем логическую функцию.

(¬X1∧(X2⊕X3))∨X2=

¬X1∧(X2∨X3)∧(¬X2∨¬X3)∨X2=

(¬X1∧X2∨¬X1∧X3)∧(¬X2∨¬X3)∨X2=

(¬X1∧X2∨¬X1∧X3∨X2)∧(¬X2∨¬X3∨X2) =

¬X1∧X2∨¬X1∧X3∨X2 =

¬X1∧(X2∨ ∧X3)∨X2 =

(¬X1∨X2 )∧(X2∨X3) =

¬X1∧ X3∨X2

Выполним проверку, составив таблицу истинности для исходной и для полученной логической функций:

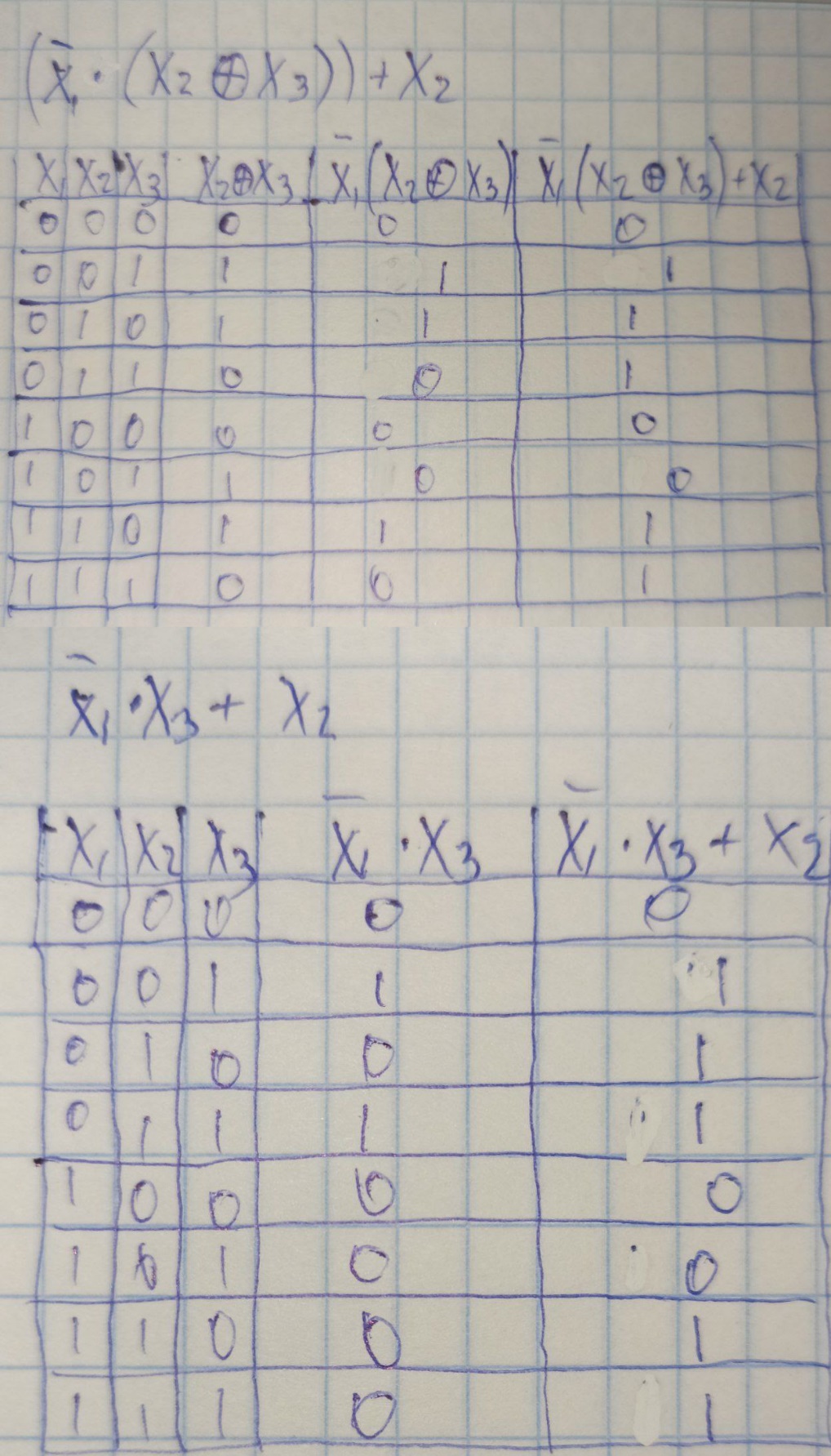


Рисунок 2 – Таблицы истинности для исходной и полученной функций.

Затем импортируем необходимых библиотеки:

NumPy для работы с массивами данных.

Matplotlib для построения графиков и визуализации разделяющих линий.

NetworkX для построения сети логических операций, визуализирующей структуру модели.

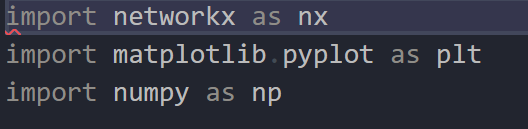


Рисунок 1 – Импорт используемых библиотек.

Создание персептронов: Созданы три персептрона для выполнения операций AND, OR и NOT. Каждый персептрон инициализируется случайными весами и обучается на соответствующих данных, что позволяет ему корректно выполнять свою логическую функцию.

Рисунок 3 – Создание класса персептрона 1 часть.

Функция предсказания и обучение (рис 3): Описана функция predict, вычисляющая предсказание на основе текущих весов, и функция train, обновляющая веса на каждом шаге обучения.

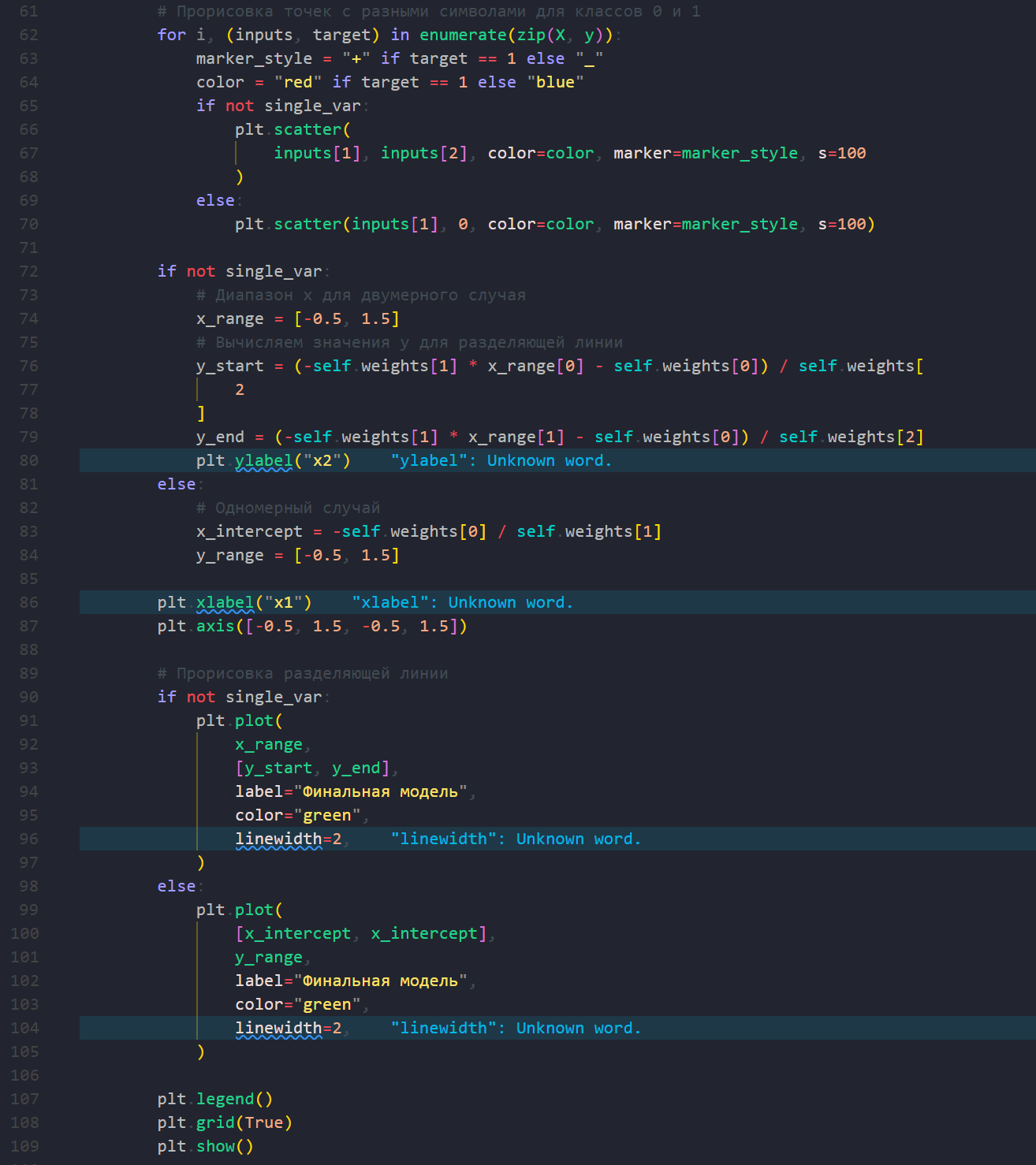


Рисунок 4 – Создание класса персептрона 2 часть.

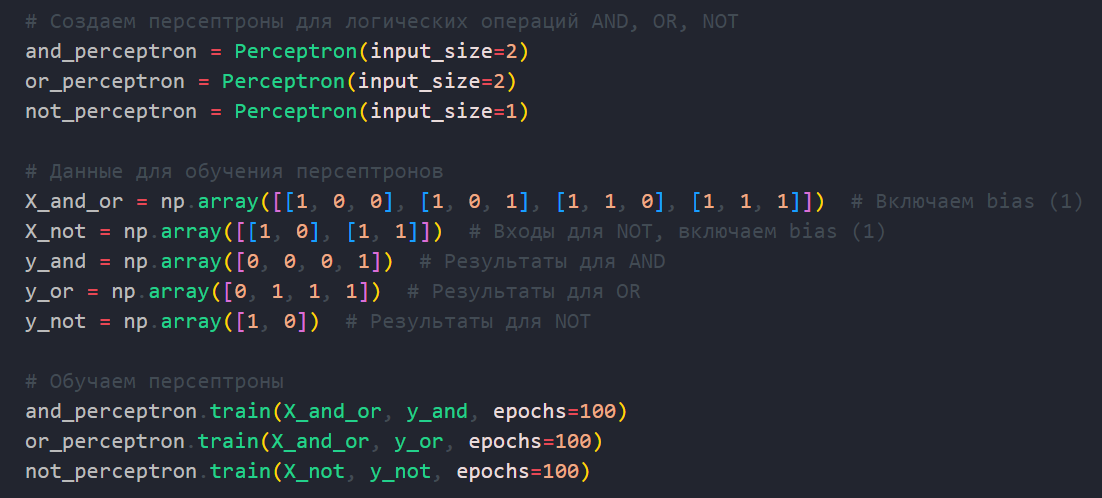


Рисунок 5 – Листинг кода функции предсказания и обучения персептрона.

Обучение логических операций (рис. 5): обучение проведено для каждого персептрона на наборе данных, соответствующем логическим операциям AND, OR и NOT. После обучения для каждой операции построены и отображены границы решений.

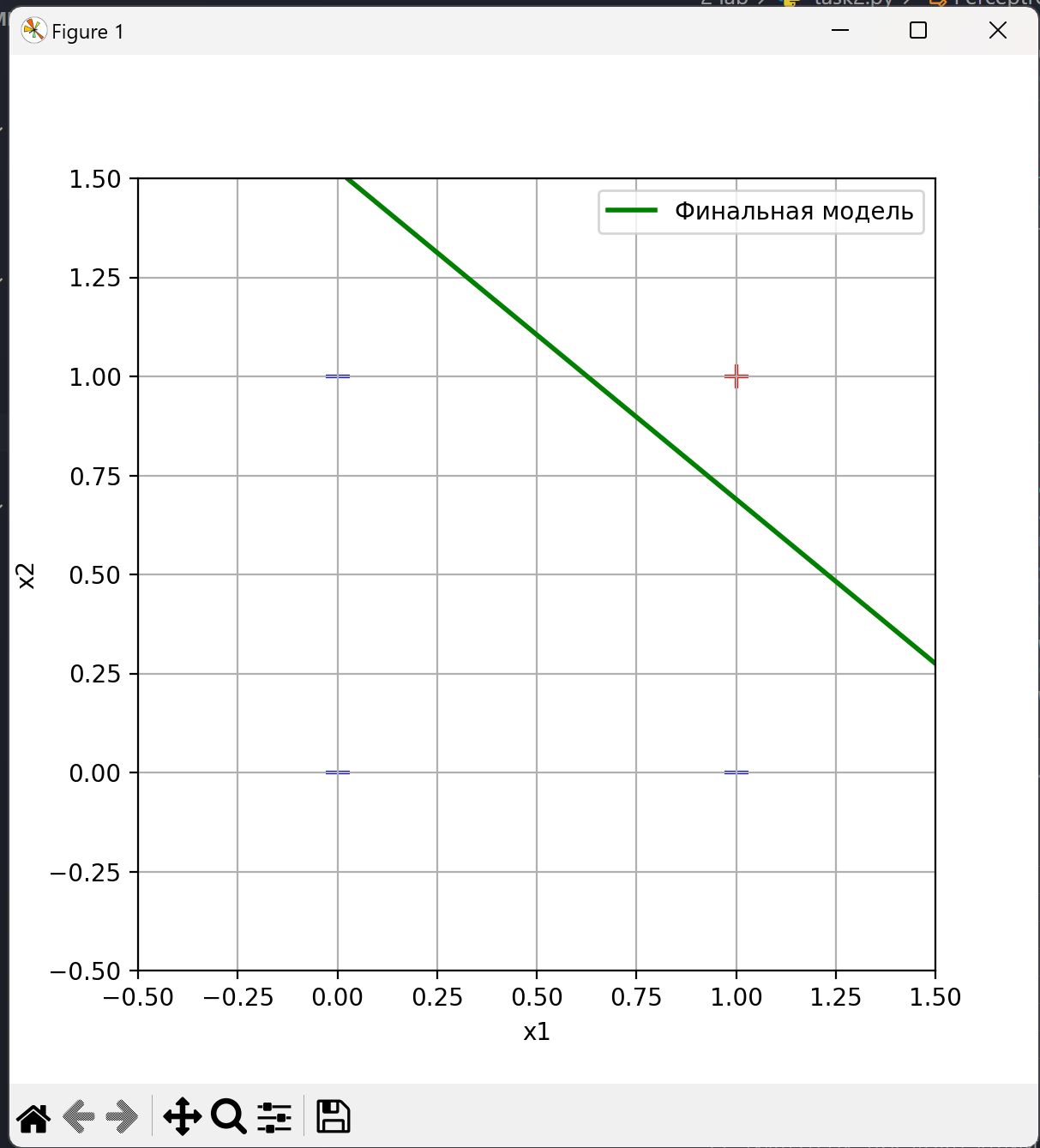


Рисунок 6 – Графики разделяющих линий для логических операций AND

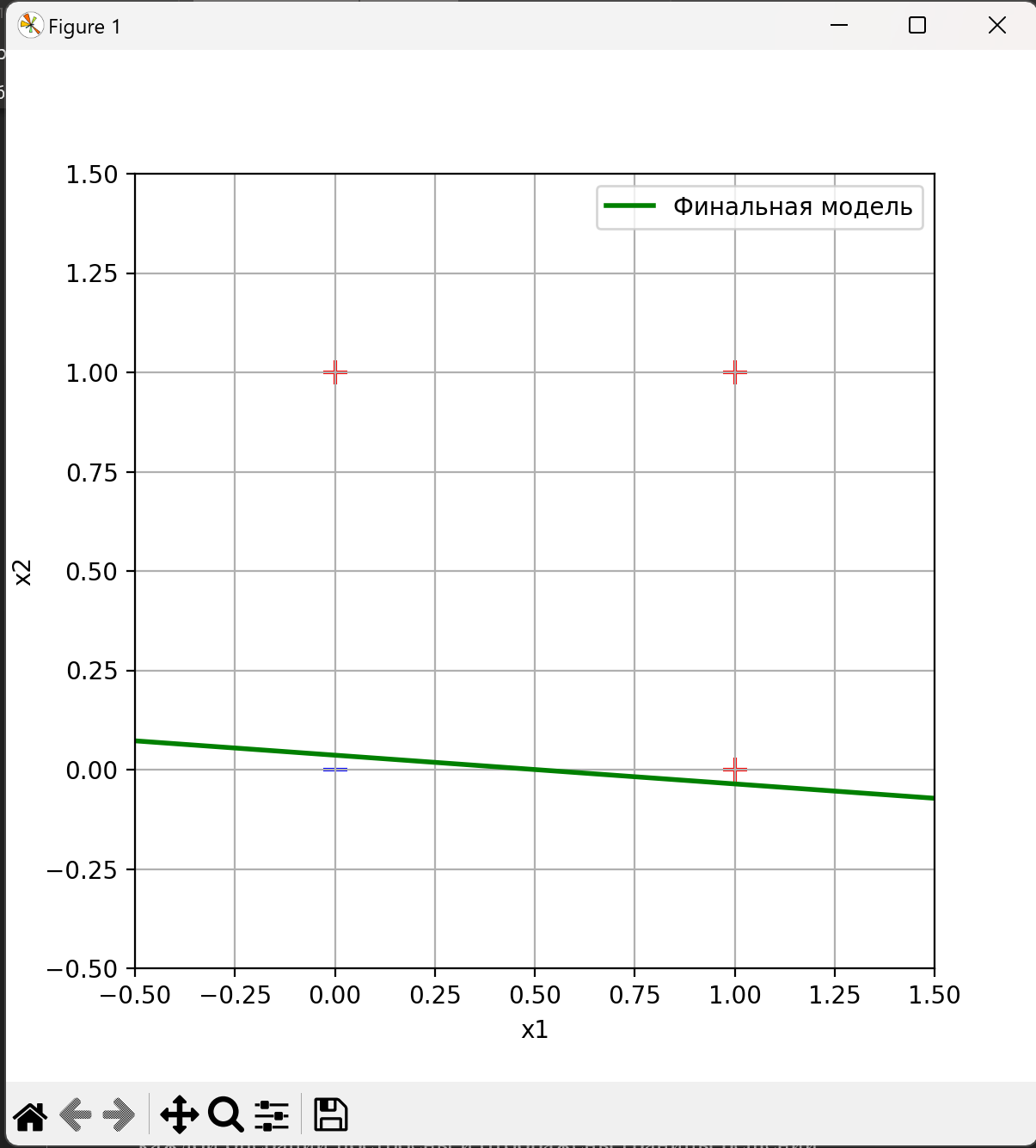


Рисунок 7 – Графики разделяющих линий для логических операций OR

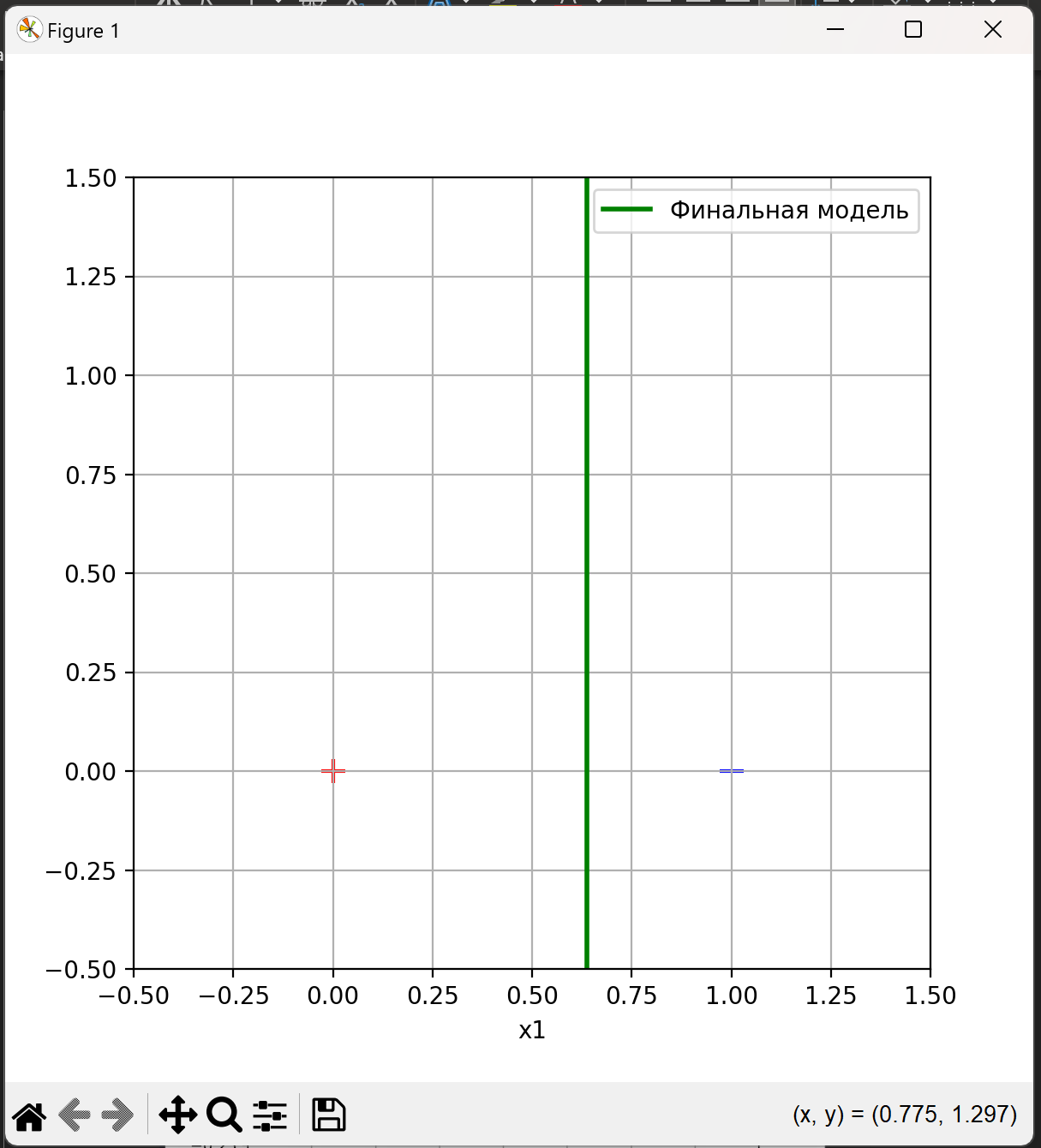


Рисунок 8 – Графики разделяющих линий для логических операций NOT

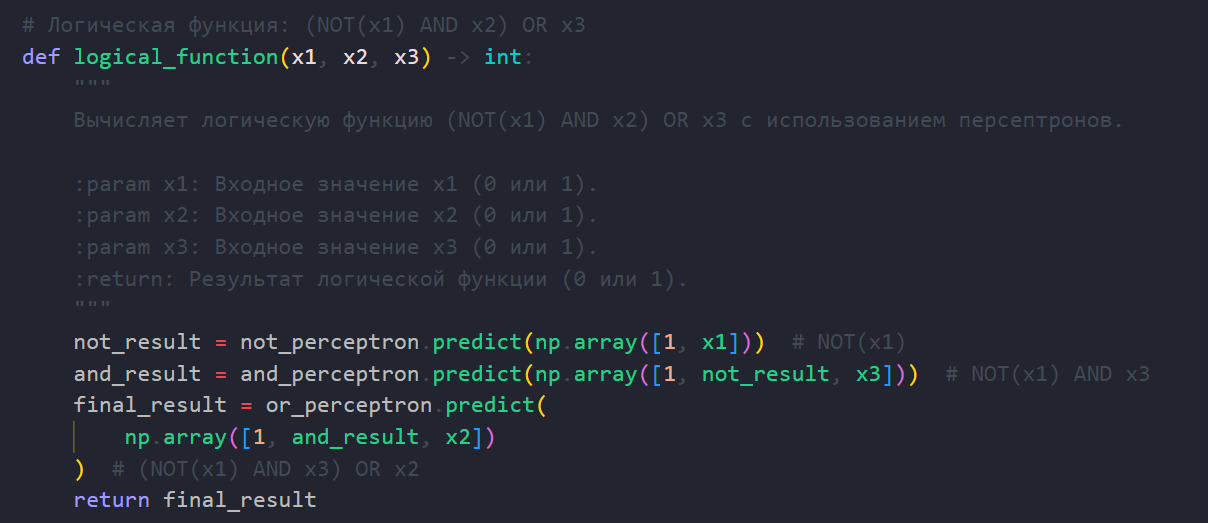
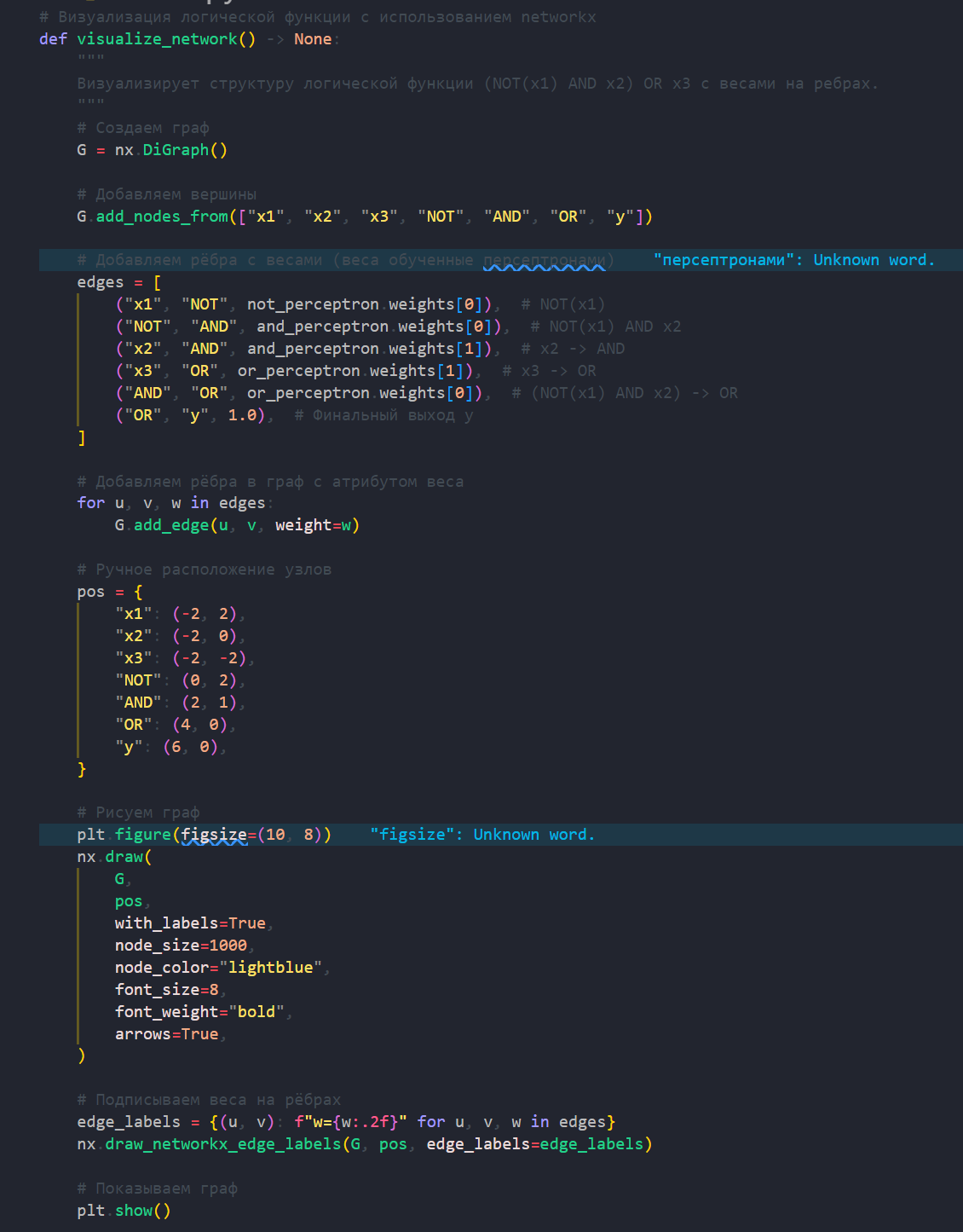


Рисунок 9 – Код логической функции (NOT(x1) AND x3) OR x2 с использованием персептронов.

Функция visualize\_network (рис. 10) визуализирует веса и схему со слоями нейронной сети, как показано на рис. 11.

  
Рисунок 10 – Код функции visualize\_network.

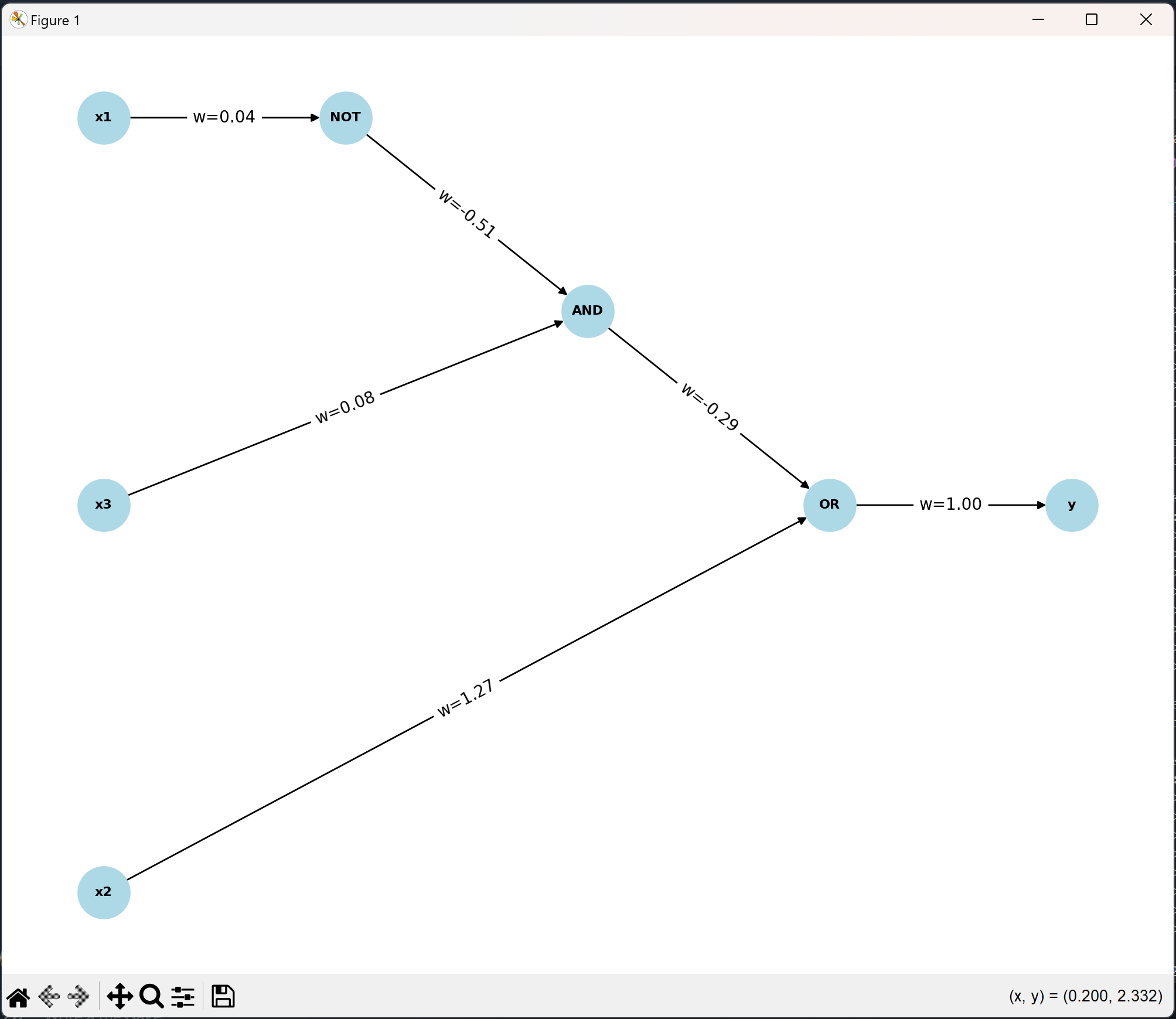


Рисунок 11 – Пример работы функции visualize\_network.

Структура нейронной сети, представленная на рис. 11 представляет собой логическую схему, состоящую из трех входных нейронов (x1, x2, x3), нескольких логических элементов (NOT, AND, OR) и одного выходного нейрона (y). Описание структуры:

Входной нейрон x1 подключен к логическому элементу NOT с весом 0.04. Выход NOT передается на элемент AND с весом 0.51.

* Входной нейрон x3 напрямую соединен с элементом AND с весом 0.08.
* Элемент AND соединен с элементом OR с весом 0.29.
* Входной нейрон x2 напрямую соединен с элементом OR с весом 1.27.
* Элемент OR соединен с выходным нейроном y с весом 1.00.

В результате эта структура моделирует комбинацию логических операций, которые зависят от весов связей между элементами, и её можно использовать для вычисления результата логических условий, основанных на значениях входных нейронов x1, x2 и x3.

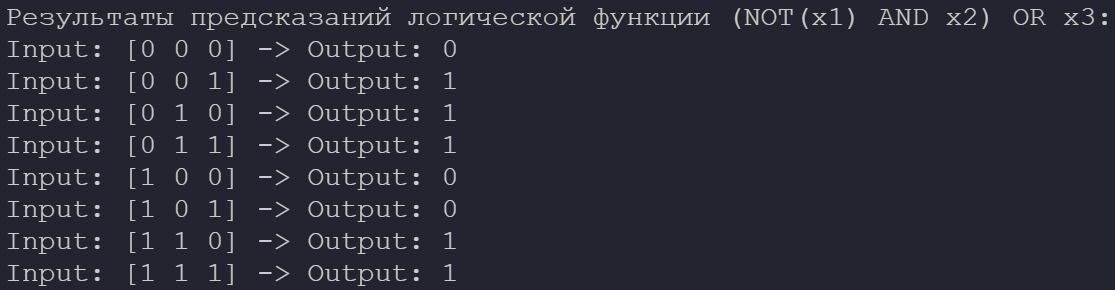


Рисунок 12 – Вывод нейронной сети логической функции.

Если сравнить вывод функции нейронной сети (рис. 12) с исходным выводом функции (рис. 2), то можно придти к выводу, что нейронная сеть правильно обучилась, и теперь она умеет предсказывать вывод для логической функции.

ЗАКЛЮЧЕНИЕ

В ходе выполнения лабораторной работы была разработана персептронная модель, состоящая из нескольких нейронов, реализующих последовательные логические операции по блок-схеме. Программа успешно выполняет заданную функцию, отображает финальные значения весов для каждой операции и визуализирует разделяющие линии на графиках, что позволяет наглядно продемонстрировать, как модель обрабатывает различные комбинации входных значений. Таким образом, цель работы достигнута, и построенная модель может быть использована для визуализации и анализа логической зависимости, заданной на блок-схеме.

# СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

1. ГОСТ Р 7.0.97-2016. Национальный стандарт Российской Федерации. Система стандартов по информации, библиотечному и издательскому делу. Организационно-распорядительная документация. Требования к оформлению документов: утвержден и введен в действие Приказом Федерального агентства по техническому регулированию и метрологии от 14.05.2018 N 244-ст: Дата введения 2018-07-01. - URL: <https://docs.cntd.ru/document/1200159234> (дата обращения: 02.10.2024). - Текст: электронный.

# ПРИЛОЖЕНИЕ А

Листинг кода

Листинг 1 - perseptron.py

import networkx as nx

import matplotlib.pyplot as plt

import numpy as np

# Класс для одного персептрона

class Perceptron:

    def \_\_init\_\_(self, input\_size: int, learning\_rate: float = 0.1) -> None:

        """

        Инициализирует персептрон с заданным количеством входов и скоростью обучения.

        :param input\_size: Количество входов (без учета bias).

        :param learning\_rate: Скорость обучения для обновления весов.

        """

        self.weights = np.random.randn(input\_size + 1)  # Включаем bias (доп. вес)

        self.learning\_rate = learning\_rate

    def predict(self, inputs: np.ndarray) -> int:

        """

        Выполняет предсказание для входных данных на основе текущих весов.

        :param inputs: Входные данные (включая bias).

        :return: Результат предсказания (1 или 0).

        """

        # Вычисляем взвешенную сумму

        summation = np.dot(inputs, self.weights)

        # Применяем пороговую активацию: если сумма >= 0, возвращаем 1, иначе 0

        return 1 if summation >= 0 else 0

    def train(self, X: np.ndarray, y: np.ndarray, epochs: int = 100) -> None:

        """

        Обучает персептрон на основе входных данных и целевых значений.

        :param X: Массив входных данных (включает bias в качестве первого элемента).

        :param y: Массив целевых значений (результатов).

        :param epochs: Количество эпох обучения.

        """

        for epoch in range(epochs):

            for inputs, target in zip(X, y):

                prediction = self.predict(inputs)  # Делаем предсказание

                error = target - prediction  # Вычисляем ошибку

                if error != 0:

                    # Обновляем веса для всех входов, кроме bias

                    self.weights[1:] += self.learning\_rate \* error \* inputs[1:]

                    # Обновляем bias (первый вес)

                    self.weights[0] += self.learning\_rate \* error

    def draw\_decision\_boundary(

        self, X: np.ndarray, y: np.ndarray, single\_var: bool = False

    ) -> None:

        """

        Визуализирует разделяющую линию (границу решения) для текущих весов персептрона.

        Также отображает входные данные и их классы (с символами "+" для 1 и "-" для 0).

        :param X: Входные данные (включая bias).

        :param y: Целевые значения (0 или 1).

        :param single\_var: Флаг, если у персептрона один входной параметр (для NOT-функции).

        """

        plt.figure(figsize=(6, 6))

        # Прорисовка точек с разными символами для классов 0 и 1

        for i, (inputs, target) in enumerate(zip(X, y)):

            marker\_style = "+" if target == 1 else "\_"

            color = "red" if target == 1 else "blue"

            if not single\_var:

                plt.scatter(

                    inputs[1], inputs[2], color=color, marker=marker\_style, s=100

                )

            else:

                plt.scatter(inputs[1], 0, color=color, marker=marker\_style, s=100)

        if not single\_var:

            # Диапазон x для двумерного случая

            x\_range = [-0.5, 1.5]

            # Вычисляем значения y для разделяющей линии

            y\_start = (-self.weights[1] \* x\_range[0] - self.weights[0]) / self.weights[

                2

            ]

            y\_end = (-self.weights[1] \* x\_range[1] - self.weights[0]) / self.weights[2]

            plt.ylabel("x2")

        else:

            # Одномерный случай

            x\_intercept = -self.weights[0] / self.weights[1]

            y\_range = [-0.5, 1.5]

        plt.xlabel("x1")

        plt.axis([-0.5, 1.5, -0.5, 1.5])

        # Прорисовка разделяющей линии

        if not single\_var:

            plt.plot(

                x\_range,

                [y\_start, y\_end],

                label="Финальная модель",

                color="green",

                linewidth=2,

            )

        else:

            plt.plot(

                [x\_intercept, x\_intercept],

                y\_range,

                label="Финальная модель",

                color="green",

                linewidth=2,

            )

        plt.legend()

        plt.grid(True)

        plt.show()

# Создаем персептроны для логических операций AND, OR, NOT

and\_perceptron = Perceptron(input\_size=2)

or\_perceptron = Perceptron(input\_size=2)

not\_perceptron = Perceptron(input\_size=1)

# Данные для обучения персептронов

X\_and\_or = np.array([[1, 0, 0], [1, 0, 1], [1, 1, 0], [1, 1, 1]])  # Включаем bias (1)

X\_not = np.array([[1, 0], [1, 1]])  # Входы для NOT, включаем bias (1)

y\_and = np.array([0, 0, 0, 1])  # Результаты для AND

y\_or = np.array([0, 1, 1, 1])  # Результаты для OR

y\_not = np.array([1, 0])  # Результаты для NOT

# Обучаем персептроны

and\_perceptron.train(X\_and\_or, y\_and, epochs=100)

or\_perceptron.train(X\_and\_or, y\_or, epochs=100)

not\_perceptron.train(X\_not, y\_not, epochs=100)

# Логическая функция: (NOT(x1) AND x2) OR x3

def logical\_function(x1, x2, x3) -> int:

    """

    Вычисляет логическую функцию (NOT(x1) AND x2) OR x3 с использованием персептронов.

    :param x1: Входное значение x1 (0 или 1).

    :param x2: Входное значение x2 (0 или 1).

    :param x3: Входное значение x3 (0 или 1).

    :return: Результат логической функции (0 или 1).

    """

    not\_result = not\_perceptron.predict(np.array([1, x1]))  # NOT(x1)

    and\_result = and\_perceptron.predict(np.array([1, not\_result, x3]))  # NOT(x1) AND x3

    final\_result = or\_perceptron.predict(

        np.array([1, and\_result, x2])

    )  # (NOT(x1) AND x3) OR x2

    return final\_result

# Тестирование на всех возможных входах

X\_test = np.array(

    [

        [0, 0, 0],

        [0, 0, 1],

        [0, 1, 0],

        [0, 1, 1],

        [1, 0, 0],

        [1, 0, 1],

        [1, 1, 0],

        [1, 1, 1],

    ]

)

print("Результаты предсказаний логической функции NOT(x1) and (x2 XOR x3) OR x2")

for inputs in X\_test:

    x1, x2, x3 = inputs

    result = logical\_function(x1, x2, x3)

    print(f"Input: {inputs} -> Output: {result}")

print("Результаты предсказаний логической функции (NOT(x1) AND x3) OR x3")

for inputs in X\_test:

    x1, x2, x3 = inputs

    result = logical\_function(x1, x2, x3)

    print(f"Input: {inputs} -> Output: {result}")

# Визуализация логической функции с использованием networkx

def visualize\_network() -> None:

    """

    Визуализирует структуру логической функции (NOT(x1) AND x2) OR x3 с весами на ребрах.

    """

    # Создаем граф

    G = nx.DiGraph()

    # Добавляем вершины

    G.add\_nodes\_from(["x1", "x2", "x3", "NOT", "AND", "OR", "y"])

    # Добавляем рёбра с весами (веса обученные персептронами)

    edges = [

        ("x1", "NOT", not\_perceptron.weights[0]),  # NOT(x1)

        ("NOT", "AND", and\_perceptron.weights[0]),  # NOT(x1) AND x2

        ("x3", "AND", and\_perceptron.weights[1]),  # x2 -> AND

        ("x2", "OR", or\_perceptron.weights[1]),  # x3 -> OR

        ("AND", "OR", or\_perceptron.weights[0]),  # (NOT(x1) AND x2) -> OR

        ("OR", "y", 1.0),  # Финальный выход y

    ]

    # Добавляем рёбра в граф с атрибутом веса

    for u, v, w in edges:

        G.add\_edge(u, v, weight=w)

    # Ручное расположение узлов

    pos = {

        "x1": (-2, 2),

        "x2": (-2, -2),

        "x3": (-2, 0),

        "NOT": (0, 2),

        "AND": (2, 1),

        "OR": (4, 0),

        "y": (6, 0),

    }

    # Рисуем граф

    plt.figure(figsize=(10, 8))

    nx.draw(

        G,

        pos,

        with\_labels=True,

        node\_size=1000,

        node\_color="lightblue",

        font\_size=8,

        font\_weight="bold",

        arrows=True,

    )

    # Подписываем веса на рёбрах

    edge\_labels = {(u, v): f"w={w:.2f}" for u, v, w in edges}

    nx.draw\_networkx\_edge\_labels(G, pos, edge\_labels=edge\_labels)

    # Показываем граф

    plt.show()

# Визуализация сети

visualize\_network()

and\_perceptron.draw\_decision\_boundary(X\_and\_or, y\_and)

or\_perceptron.draw\_decision\_boundary(X\_and\_or, y\_or)

not\_perceptron.draw\_decision\_boundary(X\_not, y\_not, single\_var=True)

import networkx as nx

import matplotlib.pyplot as plt

import numpy as np

# Класс для одного персептрона

class Perceptron:

    def \_\_init\_\_(self, input\_size: int, learning\_rate: float = 0.1) -> None:

        """

        Инициализирует персептрон с заданным количеством входов и скоростью обучения.

        :param input\_size: Количество входов (без учета bias).

        :param learning\_rate: Скорость обучения для обновления весов.

        """

        self.weights = np.random.randn(input\_size + 1)  # Включаем bias (доп. вес)

        self.learning\_rate = learning\_rate

    def predict(self, inputs: np.ndarray) -> int:

        """

        Выполняет предсказание для входных данных на основе текущих весов.

        :param inputs: Входные данные (включая bias).

        :return: Результат предсказания (1 или 0).

        """

        # Вычисляем взвешенную сумму

        summation = np.dot(inputs, self.weights)

        # Применяем пороговую активацию: если сумма >= 0, возвращаем 1, иначе 0

        return 1 if summation >= 0 else 0

    def train(self, X: np.ndarray, y: np.ndarray, epochs: int = 100) -> None:

        """

        Обучает персептрон на основе входных данных и целевых значений.

        :param X: Массив входных данных (включает bias в качестве первого элемента).

        :param y: Массив целевых значений (результатов).

        :param epochs: Количество эпох обучения.

        """

        for epoch in range(epochs):

            for inputs, target in zip(X, y):

                prediction = self.predict(inputs)  # Делаем предсказание

                error = target - prediction  # Вычисляем ошибку

                if error != 0:

                    # Обновляем веса для всех входов, кроме bias

                    self.weights[1:] += self.learning\_rate \* error \* inputs[1:]

                    # Обновляем bias (первый вес)

                    self.weights[0] += self.learning\_rate \* error

    def draw\_decision\_boundary(

        self, X: np.ndarray, y: np.ndarray, single\_var: bool = False

    ) -> None:

        """

        Визуализирует разделяющую линию (границу решения) для текущих весов персептрона.

        Также отображает входные данные и их классы (с символами "+" для 1 и "-" для 0).

        :param X: Входные данные (включая bias).

        :param y: Целевые значения (0 или 1).

        :param single\_var: Флаг, если у персептрона один входной параметр (для NOT-функции).

        """

        plt.figure(figsize=(6, 6))

        # Прорисовка точек с разными символами для классов 0 и 1

        for i, (inputs, target) in enumerate(zip(X, y)):

            marker\_style = "+" if target == 1 else "\_"

            color = "red" if target == 1 else "blue"

            if not single\_var:

                plt.scatter(

                    inputs[1], inputs[2], color=color, marker=marker\_style, s=100

                )

            else:

                plt.scatter(inputs[1], 0, color=color, marker=marker\_style, s=100)

        if not single\_var:

            # Диапазон x для двумерного случая

            x\_range = [-0.5, 1.5]

            # Вычисляем значения y для разделяющей линии

            y\_start = (-self.weights[1] \* x\_range[0] - self.weights[0]) / self.weights[

                2

            ]

            y\_end = (-self.weights[1] \* x\_range[1] - self.weights[0]) / self.weights[2]

            plt.ylabel("x2")

        else:

            # Одномерный случай

            x\_intercept = -self.weights[0] / self.weights[1]

            y\_range = [-0.5, 1.5]

        plt.xlabel("x1")

        plt.axis([-0.5, 1.5, -0.5, 1.5])

        # Прорисовка разделяющей линии

        if not single\_var:

            plt.plot(

                x\_range,

                [y\_start, y\_end],

                label="Финальная модель",

                color="green",

                linewidth=2,

            )

        else:

            plt.plot(

                [x\_intercept, x\_intercept],

                y\_range,

                label="Финальная модель",

                color="green",

                linewidth=2,

            )

        plt.legend()

        plt.grid(True)

        plt.show()

# Создаем персептроны для логических операций AND, OR, NOT

and\_perceptron = Perceptron(input\_size=2)

or\_perceptron = Perceptron(input\_size=2)

not\_perceptron = Perceptron(input\_size=1)

# Данные для обучения персептронов

X\_and\_or = np.array([[1, 0, 0], [1, 0, 1], [1, 1, 0], [1, 1, 1]])  # Включаем bias (1)

X\_not = np.array([[1, 0], [1, 1]])  # Входы для NOT, включаем bias (1)

y\_and = np.array([0, 0, 0, 1])  # Результаты для AND

y\_or = np.array([0, 1, 1, 1])  # Результаты для OR

y\_not = np.array([1, 0])  # Результаты для NOT

# Обучаем персептроны

and\_perceptron.train(X\_and\_or, y\_and, epochs=100)

or\_perceptron.train(X\_and\_or, y\_or, epochs=100)

not\_perceptron.train(X\_not, y\_not, epochs=100)

# Логическая функция: (NOT(x1) AND x2) OR x3

def logical\_function(x1, x2, x3) -> int:

    """

    Вычисляет логическую функцию (NOT(x1) AND x2) OR x3 с использованием персептронов.

    :param x1: Входное значение x1 (0 или 1).

    :param x2: Входное значение x2 (0 или 1).

    :param x3: Входное значение x3 (0 или 1).

    :return: Результат логической функции (0 или 1).

    """

    not\_result = not\_perceptron.predict(np.array([1, x1]))  # NOT(x1)

    and\_result = and\_perceptron.predict(np.array([1, not\_result, x3]))  # NOT(x1) AND x3

    final\_result = or\_perceptron.predict(

        np.array([1, and\_result, x2])

    )  # (NOT(x1) AND x3) OR x2

    return final\_result

# Тестирование на всех возможных входах

X\_test = np.array(

    [

        [0, 0, 0],

        [0, 0, 1],

        [0, 1, 0],

        [0, 1, 1],

        [1, 0, 0],

        [1, 0, 1],

        [1, 1, 0],

        [1, 1, 1],

    ]

)

print("Результаты предсказаний логической функции NOT(x1) and (x2 XOR x3) OR x2")

for inputs in X\_test:

    x1, x2, x3 = inputs

    result = logical\_function(x1, x2, x3)

    print(f"Input: {inputs} -> Output: {result}")

print("Результаты предсказаний логической функции (NOT(x1) AND x3) OR x3")

for inputs in X\_test:

    x1, x2, x3 = inputs

    result = logical\_function(x1, x2, x3)

    print(f"Input: {inputs} -> Output: {result}")

# Визуализация логической функции с использованием networkx

def visualize\_network() -> None:

    """

    Визуализирует структуру логической функции (NOT(x1) AND x2) OR x3 с весами на ребрах.

    """

    # Создаем граф

    G = nx.DiGraph()

    # Добавляем вершины

    G.add\_nodes\_from(["x1", "x2", "x3", "NOT", "AND", "OR", "y"])

    # Добавляем рёбра с весами (веса обученные персептронами)

    edges = [

        ("x1", "NOT", not\_perceptron.weights[0]),  # NOT(x1)

        ("NOT", "AND", and\_perceptron.weights[0]),  # NOT(x1) AND x2

        ("x3", "AND", and\_perceptron.weights[1]),  # x2 -> AND

        ("x2", "OR", or\_perceptron.weights[1]),  # x3 -> OR

        ("AND", "OR", or\_perceptron.weights[0]),  # (NOT(x1) AND x2) -> OR

        ("OR", "y", 1.0),  # Финальный выход y

    ]

    # Добавляем рёбра в граф с атрибутом веса

    for u, v, w in edges:

        G.add\_edge(u, v, weight=w)

    # Ручное расположение узлов

    pos = {

        "x1": (-2, 2),

        "x2": (-2, -2),

        "x3": (-2, 0),

        "NOT": (0, 2),

        "AND": (2, 1),

        "OR": (4, 0),

        "y": (6, 0),

    }

    # Рисуем граф

    plt.figure(figsize=(10, 8))

    nx.draw(

        G,

        pos,

        with\_labels=True,

        node\_size=1000,

        node\_color="lightblue",

        font\_size=8,

        font\_weight="bold",

        arrows=True,

    )

    # Подписываем веса на рёбрах

    edge\_labels = {(u, v): f"w={w:.2f}" for u, v, w in edges}

    nx.draw\_networkx\_edge\_labels(G, pos, edge\_labels=edge\_labels)

    # Показываем граф

    plt.show()

# Визуализация сети

visualize\_network()

and\_perceptron.draw\_decision\_boundary(X\_and\_or, y\_and)

or\_perceptron.draw\_decision\_boundary(X\_and\_or, y\_or)

not\_perceptron.draw\_decision\_boundary(X\_not, y\_not, single\_var=True)