|  |  |
| --- | --- |
| 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 №4

Студент 2 курса группы 20221

Очного отделения

Руденко Вячеслав Сергеевич

Проверил:

Преподаватель CПбГМТУ

Кайнова Татьяна Денисовна

Санкт-Петербург

2024

Оглавление

[ОБЩАЯ ЧАСТЬ 3](#_Toc183613088)

1. [1.1 Цель работы 3](#_Toc183613089)
2. [1.2 Формулировка задачи 3](#_Toc183613090)

[ХОД РАБОТЫ 4](#_Toc183613091)

[ПРИЛОЖЕНИЕ А 10](#_Toc183613092)

# ОБЩАЯ ЧАСТЬ

## 1.1 Цель работы

Изучить метод обратного распространения ошибки для настройки параметров многослойной нейронной сети, реализовать алгоритм обучения нейронной сети с использованием двух скрытых слоев и проверить ее работу на примере логической функции.

## 1.2 Формулировка задачи

Доработать программу из второй лабораторной работы, необходимо сделать теперь её автоматическое обучение с помощью алгоритма обратного распространения ошибки.

Обязательное условие: в сети должно быть не менее двух скрытых слоёв + выходной слой, а в каждом скрытом слое более двух нейронов.

# ХОД РАБОТЫ

Для начала реализовываем функции: sigmoid – сигмоида, sigmoid\_derivative – производная сигмоиды для массива, tanh – гиперболический тангенс и tanh\_derivative – проиводная гиперболического тангенса (рис. 1)

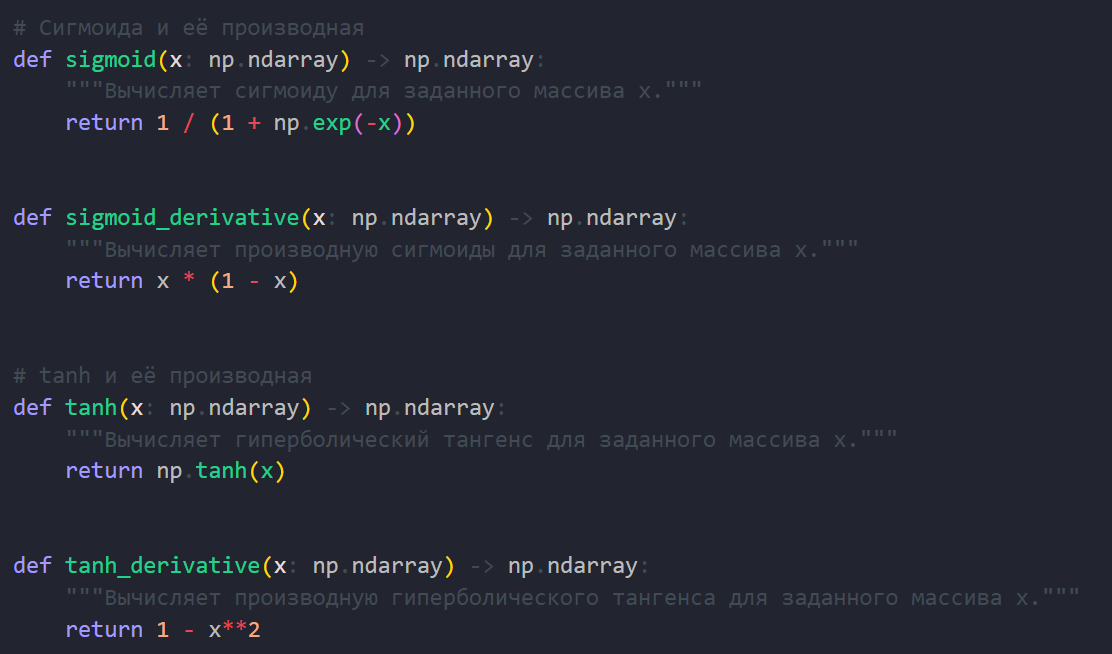


Рисунок 1 – Инициализация функций: sigmoid, sigmoid\_derivative, tanh, tanh\_derivative

Класс NeuralNetwork реализует нейронную сеть с одним скрытым слоем и поддержкой обратного распространения ошибки.

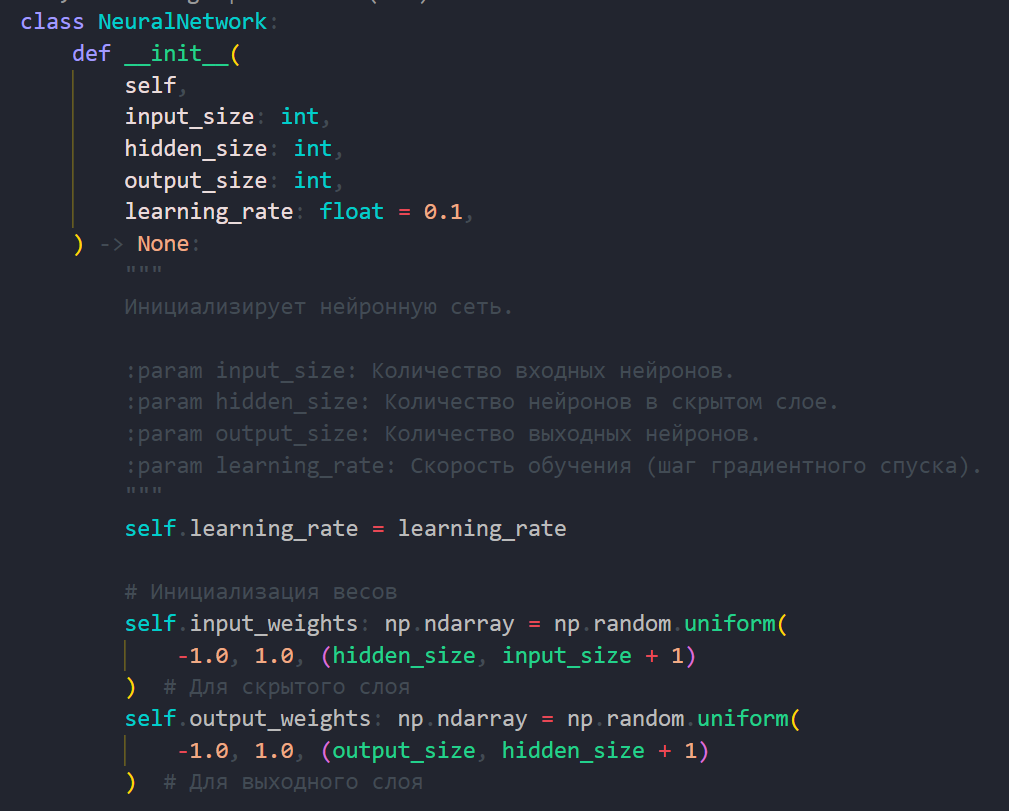


Рисунок 2 – Инициализация класса NeuralNetwork

При инициализации нейронной сети (рис. 2) задаются размеры входного, скрытого и выходного слоёв. Также задаётся скорость обучения. Весовые коэффициенты инициализируются случайными значениями из диапазона [−1,1]. Для каждого слоя предусмотрены отдельные матрицы весов: одна для преобразования входных данных в скрытые активации, а другая — для преобразования скрытых активаций в выходные значения.

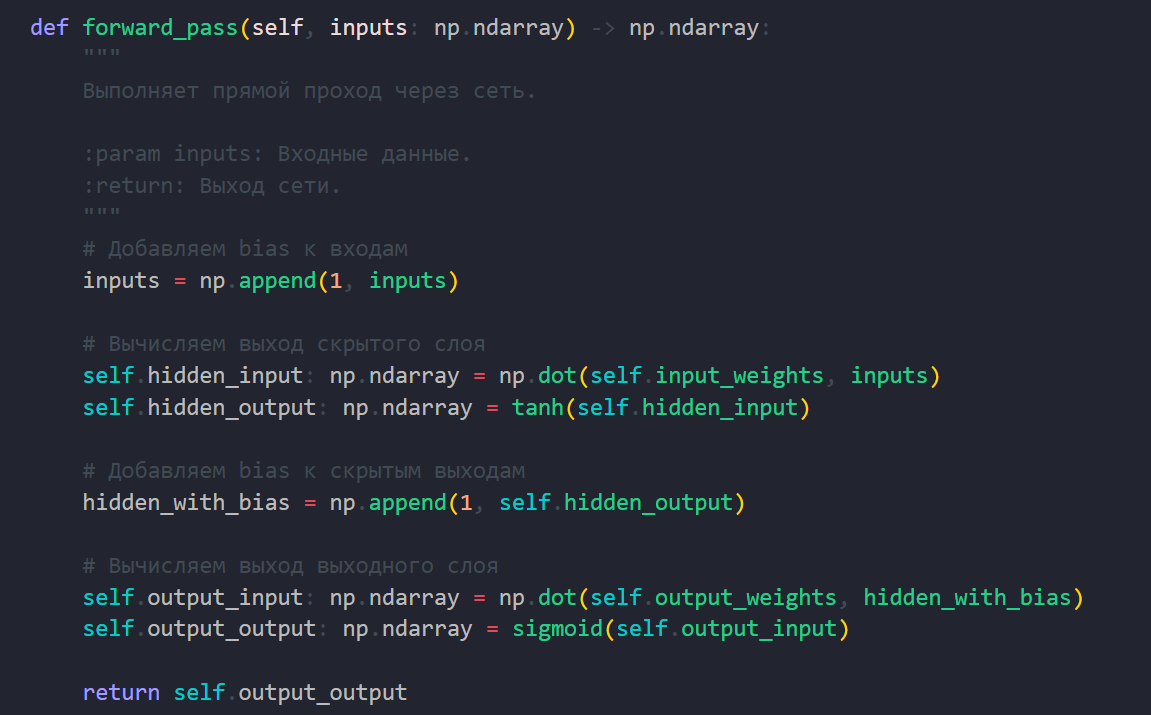


Рисунок 3 – Реализация метода forward\_pass

Прямой проход реализован в методе forward\_pass (рис. 3). Здесь входные данные, к которым добавляется bias, преобразуются сначала в скрытые активации с помощью матрицы весов скрытого слоя. Для активации используется функция tanh, которая нормализует значения в диапазоне [−1,1]. Затем, к результату скрытого слоя также добавляется bias, и вычисляются выходные значения с использованием весов выходного слоя и функции активации sigmoid. Функция sigmoid преобразует значения в диапазон [0,1], что особенно удобно для задач классификации.

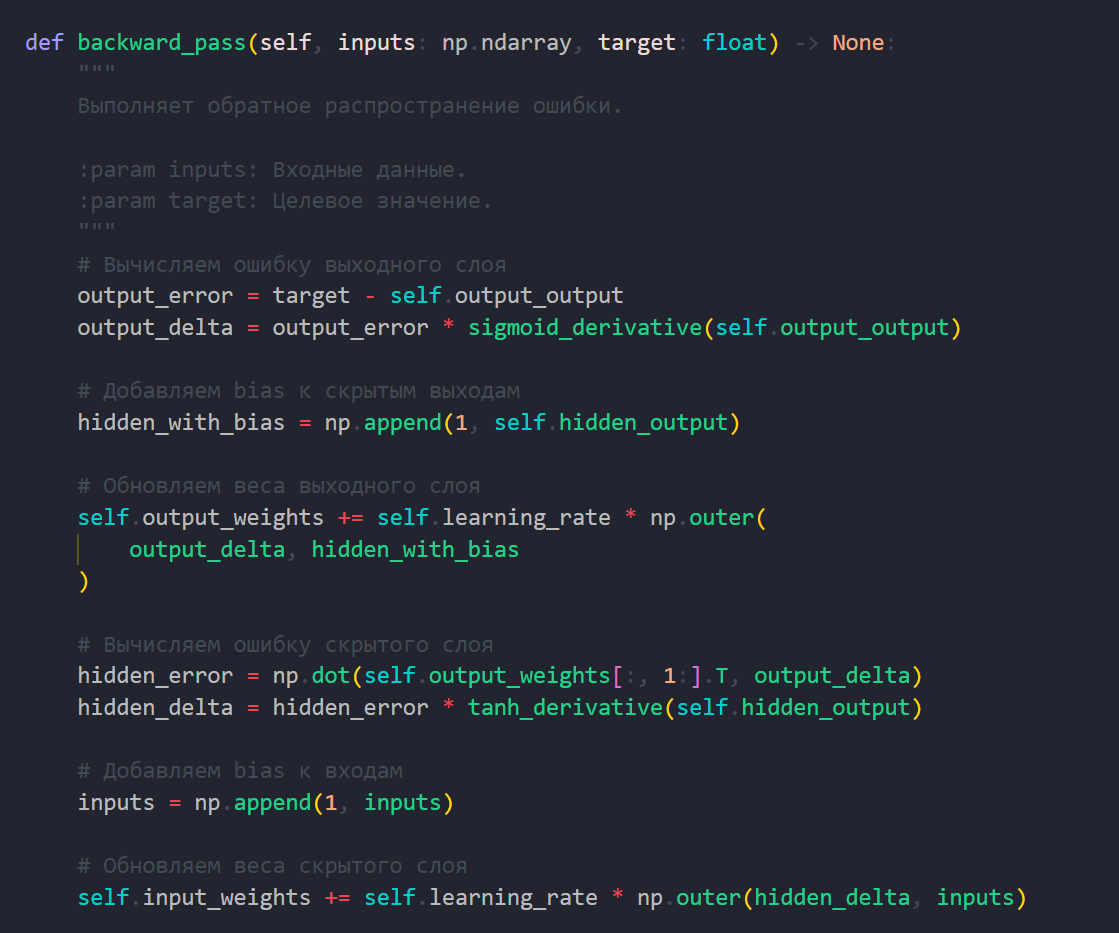


Рисунок 4 – Реализация метода backward\_pass

Обучение сети реализовано через метод обратного распространения ошибки, описанный в функции backward\_pass (рис 4.). На этом этапе сначала рассчитывается ошибка выходного слоя, то есть разница между ожидаемым значением (целевым) и предсказанием сети. Затем для выхода вычисляется градиент ошибки с помощью производной функции активации sigmoid, что позволяет оценить, насколько сильно нужно корректировать веса. Обновление весов выходного слоя происходит на основе этого градиента.

Далее ошибка распространяется обратно к скрытому слою. Вычисляется её влияние на скрытые активации через производную функции tanh. Таким образом, корректируются и веса скрытого слоя. Для этого этапа bias также играет важную роль, позволяя модели учиться лучше.

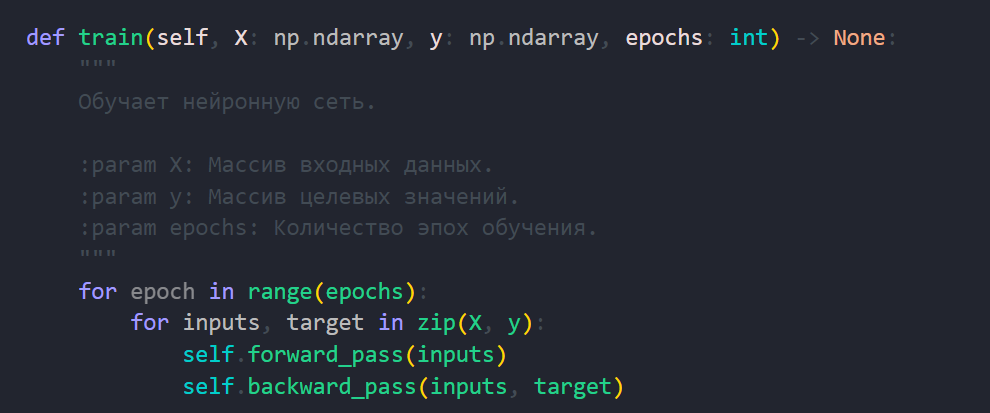


Рисунок 5 – Реализация метода train

Метод train (рис. 5) отвечает за полный цикл обучения сети. Он повторяет прямой и обратный проходы для каждого примера входных данных на протяжении заданного числа эпох. Этот процесс обеспечивает постепенное снижение ошибки и улучшение предсказательной способности модели.

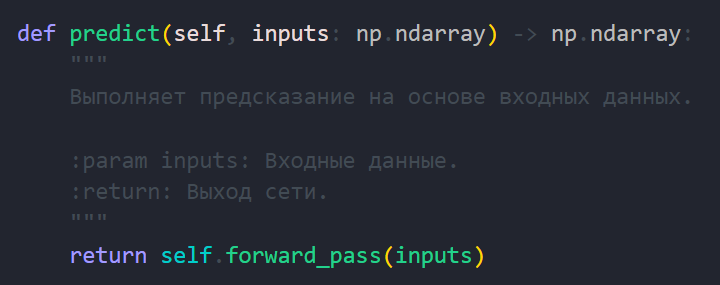


Рисунок 6 – Реализация метода predict

Наконец, метод predict (рис.6) используется для получения выходных значений на основе новых входных данных. Он выполняет только прямой проход, что делает предсказание быстрым и эффективным.



Рисунок 7 – Инициализация входов и выходов.

На рисунке 7 показана инициализация данных для обучения

Массив X\_train — примеры входных данных для логической функции, а массив y\_train — соответствующие целевые значения.

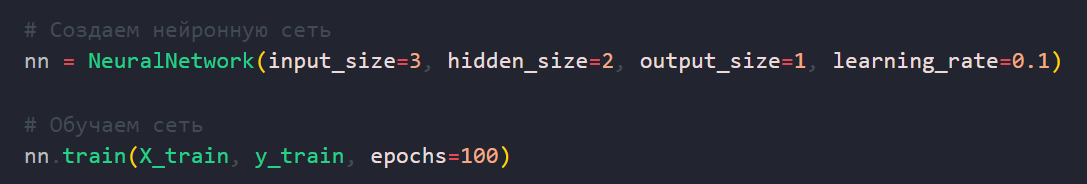


Рисунок 8 – Создание нейронной сети и её обучение.

На этапе создания нейронной сети (рис. 8) задаются её основные параметры:

Размер входного слоя (input\_size) определяет количество входных нейронов, которые принимают исходные данные.

Размер скрытого слоя (hidden\_size) определяет количество нейронов, обрабатывающих данные внутри сети.

Скорость обучения (learning\_rate) управляет величиной изменения весов на каждом шаге. Более высокий learning rate ускоряет обучение, но может привести к нестабильности, тогда как низкий делает обучение более плавным, но медленным.

После определения параметров случайным образом инициализируются веса. Это включает:

Веса, связывающие входной слой со скрытым (матрица input\_weights).

Веса, связывающие скрытый слой с выходным (матрица output\_weights).

Обучение сети происходит в методе train (рис. 8 и рис.5), который принимает:

X — массив входных данных.

y — массив целевых значений, соответствующих каждому примеру в X.

epochs — количество циклов обучения, которое задаёт, сколько раз сеть обработает весь набор данных.

Во время обучения для каждого примера данных прямой и обратный проход:

Прямой проход (forward\_pass):

Входные данные подаются на сеть, и для них вычисляются выходы скрытого и выходного слоёв.

На этом этапе сеть выдаёт предсказание, которое сравнивается с целевым значением.

Обратный проход (backward\_pass):

Сначала вычисляется ошибка на выходе, то есть разница между предсказанным и целевым значением. Затем ошибка распространяется обратно: сначала обновляются веса выходного слоя, а затем — скрытого.

Для обновления весов используются производные функций активации sigmoid и tanh, чтобы учитывать нелинейные зависимости в данных.

Процесс повторяется для всех примеров в наборе данных и для всех эпох. С каждым циклом веса корректируются.

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.

# ПРИЛОЖЕНИЕ А

Листинг кода

Листинг 1 - perseptron.py

import numpy as np

# Сигмоида и её производная

def sigmoid(x: np.ndarray) -> np.ndarray:

    """Вычисляет сигмоиду для заданного массива x."""

    return 1 / (1 + np.exp(-x))

def sigmoid\_derivative(x: np.ndarray) -> np.ndarray:

    """Вычисляет производную сигмоиды для заданного массива x."""

    return x \* (1 - x)

# tanh и её производная

def tanh(x: np.ndarray) -> np.ndarray:

    """Вычисляет гиперболический тангенс для заданного массива x."""

    return np.tanh(x)

def tanh\_derivative(x: np.ndarray) -> np.ndarray:

    """Вычисляет производную гиперболического тангенса для заданного массива x."""

    return 1 - x\*\*2

# Нейронная сеть с обратным распространением ошибки

class NeuralNetwork:

    def \_\_init\_\_(

        self,

        input\_size: int,

        hidden\_size: int,

        output\_size: int,

        learning\_rate: float = 0.1,

    ) -> None:

        """

        Инициализирует нейронную сеть.

        :param input\_size: Количество входных нейронов.

        :param hidden\_size: Количество нейронов в скрытом слое.

        :param output\_size: Количество выходных нейронов.

        :param learning\_rate: Скорость обучения (шаг градиентного спуска).

        """

        self.learning\_rate = learning\_rate

        # Инициализация весов

        self.input\_weights: np.ndarray = np.random.uniform(

            -1.0, 1.0, (hidden\_size, input\_size + 1)

        )  # Для скрытого слоя

        self.output\_weights: np.ndarray = np.random.uniform(

            -1.0, 1.0, (output\_size, hidden\_size + 1)

        )  # Для выходного слоя

    def forward\_pass(self, inputs: np.ndarray) -> np.ndarray:

        """

        Выполняет прямой проход через сеть.

        :param inputs: Входные данные.

        :return: Выход сети.

        """

        # Добавляем bias к входам

        inputs = np.append(1, inputs)

        # Вычисляем выход скрытого слоя

        self.hidden\_input: np.ndarray = np.dot(self.input\_weights, inputs)

        self.hidden\_output: np.ndarray = tanh(self.hidden\_input)

        # Добавляем bias к скрытым выходам

        hidden\_with\_bias = np.append(1, self.hidden\_output)

        # Вычисляем выход выходного слоя

        self.output\_input: np.ndarray = np.dot(self.output\_weights, hidden\_with\_bias)

        self.output\_output: np.ndarray = sigmoid(self.output\_input)

        return self.output\_output

    def backward\_pass(self, inputs: np.ndarray, target: float) -> None:

        """

        Выполняет обратное распространение ошибки.

        :param inputs: Входные данные.

        :param target: Целевое значение.

        """

        # Вычисляем ошибку выходного слоя

        output\_error = target - self.output\_output

        output\_delta = output\_error \* sigmoid\_derivative(self.output\_output)

        # Добавляем bias к скрытым выходам

        hidden\_with\_bias = np.append(1, self.hidden\_output)

        # Обновляем веса выходного слоя

        self.output\_weights += self.learning\_rate \* np.outer(

            output\_delta, hidden\_with\_bias

        )

        # Вычисляем ошибку скрытого слоя

        hidden\_error = np.dot(self.output\_weights[:, 1:].T, output\_delta)

        hidden\_delta = hidden\_error \* tanh\_derivative(self.hidden\_output)

        # Добавляем bias к входам

        inputs = np.append(1, inputs)

        # Обновляем веса скрытого слоя

        self.input\_weights += self.learning\_rate \* np.outer(hidden\_delta, inputs)

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

        """

        Обучает нейронную сеть.

        :param X: Массив входных данных.

        :param y: Массив целевых значений.

        :param epochs: Количество эпох обучения.

        """

        for epoch in range(epochs):

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

                self.forward\_pass(inputs)

                self.backward\_pass(inputs, target)

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

        """

        Выполняет предсказание на основе входных данных.

        :param inputs: Входные данные.

        :return: Выход сети.

        """

        return self.forward\_pass(inputs)

# Данные для обучения логической функции

X\_train: np.ndarray = 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],

    ]

)

y\_train: np.ndarray = np.array([0, 1, 1, 1, 0, 0, 1, 1])  # Пример целевых значений

# Создаем нейронную сеть

nn = NeuralNetwork(input\_size=3, hidden\_size=2, output\_size=1, learning\_rate=0.1)

# Обучаем сеть

nn.train(X\_train, y\_train, epochs=10000)

# Тестирование сети

print("Результаты предсказаний логической функции:")

for inputs in X\_train:

    output = nn.predict(inputs)

    print(f"Input: {inputs} -> Output: {output.round()}")