|  |  |
| --- | --- |
| 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 №3

Студент 2 курса группы 20221

Очного отделения

Руденко Вячеслав Сергеевич

Проверил:

Преподаватель CПбГМТУ

Кайнова Татьяна Денисовна

Санкт-Петербург

2024

Оглавление

[1. ОБЩАЯ ЧАСТЬ 3](#_Toc182820056)

1. [1.1 Цель работы 3](#_Toc182820057)
2. [1.2 Формулировка задачи 3](#_Toc182820058)

[2. ХОД РАБОТЫ 4](#_Toc182820059)

[2.1 Описание работы алгоритма 4](#_Toc182820060)

[2.3 Блок-схема работы алгоритма. 7](#_Toc182820061)

[СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ 9](#_Toc182820062)

[ПРИЛОЖЕНИЕ А 10](#_Toc182820063)

# 1. ОБЩАЯ ЧАСТЬ

## 1.1 Цель работы

Изучение и реализация алгоритма обратного распространения ошибки (backpropagation) для обучения двухслойной нейронной сети. Анализ процесса обучения сети на примере функции "исключающее ИЛИ" (XOR).

## 1.2 Формулировка задачи

Разработать двухслойную нейронную сеть с применением активационных функций tanh и сигмоиды. Реализовать процесс обучения сети на основе алгоритма обратного распространения ошибки для выполнения логической операции XOR. Провести анализ изменения весов нейронов в процессе обучения.

# 2. ХОД РАБОТЫ

### 2.1 Описание работы алгоритма

Алгоритм, представленный в работе, реализует обучение двухслойной нейронной сети для выполнения функции XOR с использованием метода обратного распространения.

В начале работы алгоритма задаются начальные параметры нейронной сети: веса, скорость обучения, обучающая выборка и её истинные значения.ия ошибки. Рассмотрим основные этапы работы.

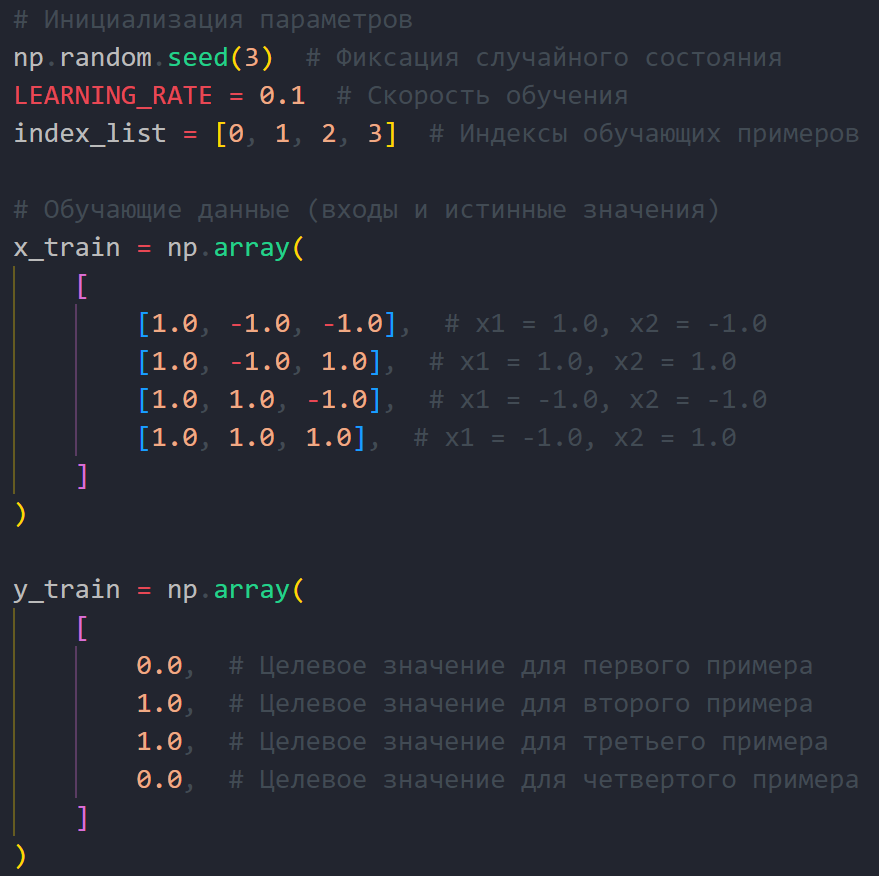
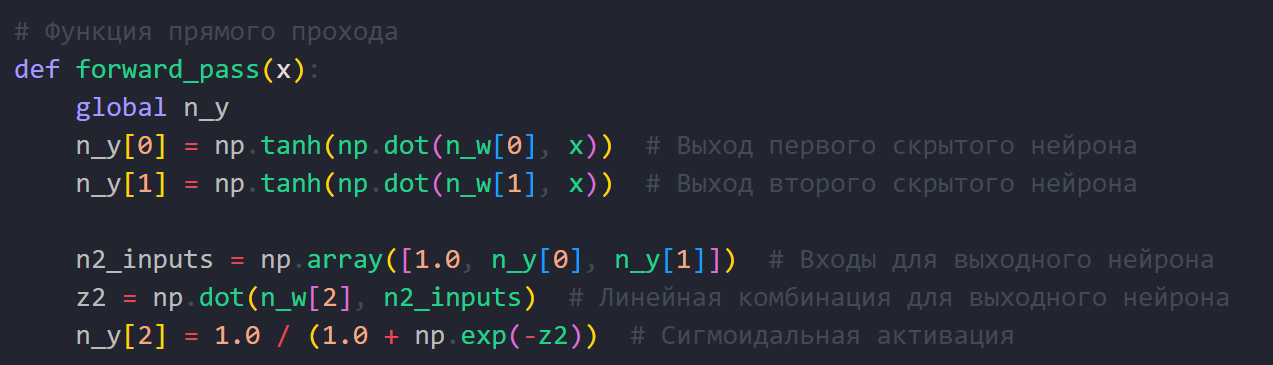


Рисунок 1 - Инициализация параметров.

На этом этапе:

* Инициализируются случайные веса сети с помощью функции neuron\_w, генерирующей массив весов для каждого нейрона.
* Устанавливается фиксированный seed (np.random.seed(3)), чтобы обеспечить воспроизводимость результатов.
* Входные и выходные данные формируются для обучения.

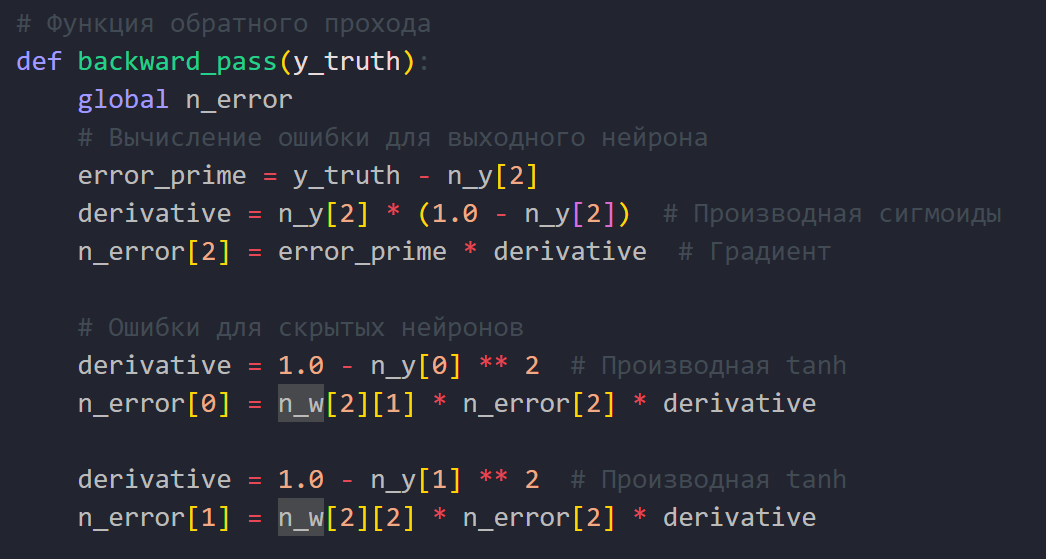
На этапе прямого прохода вычисляются выходы нейронов. Для скрытых нейронов применяется функция активации tanh, а для выходного нейрона используется логистическая сигмоида.

* 
* Рисунок 2 - Forward-pass (прямой проход).

На этом этапе:

* Для каждого входного примера рассчитываются выходы скрытых нейронов (n\_y[0] и n\_y[1]).
* Выходы скрытых нейронов передаются на вход выходному нейрону (n2\_inputs).
* Используется сигмоидальная функция активации для получения окончательного результата сети (n\_y[2]).

Обратный проход вычисляет ошибки сети и обновляет веса на основе метода градиентного спуска (рис. 2).

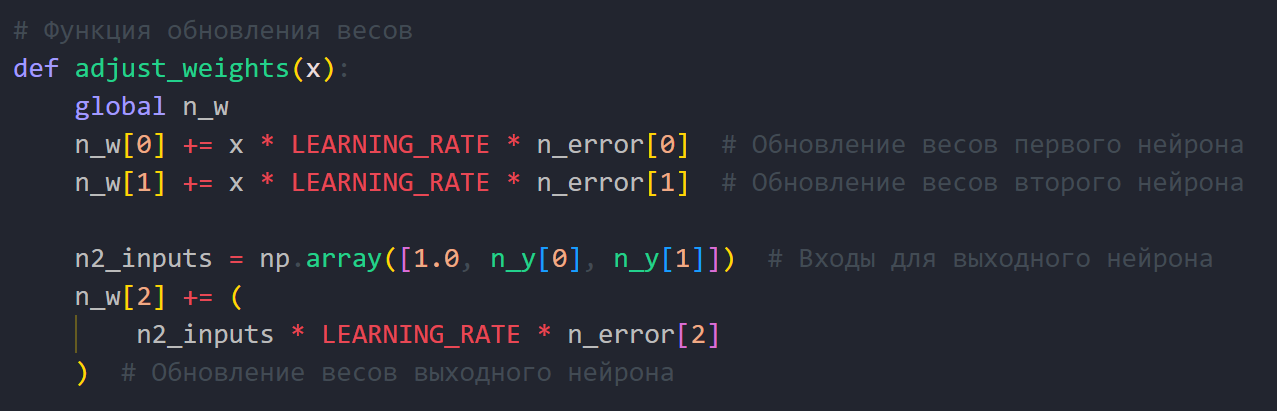


* Рисунок 2 - Backward-pass (обратный проход).

На этом этапе:

* Вычисляется ошибка выходного нейрона (n\_error[2]) с учетом разности между истинным значением (y\_truth) и выходом сети (n\_y[2]).
* Ошибки распространяются обратно через сеть для двух скрытых нейронов (n\_error[0] и n\_error[1]), с учетом их активационных функций.

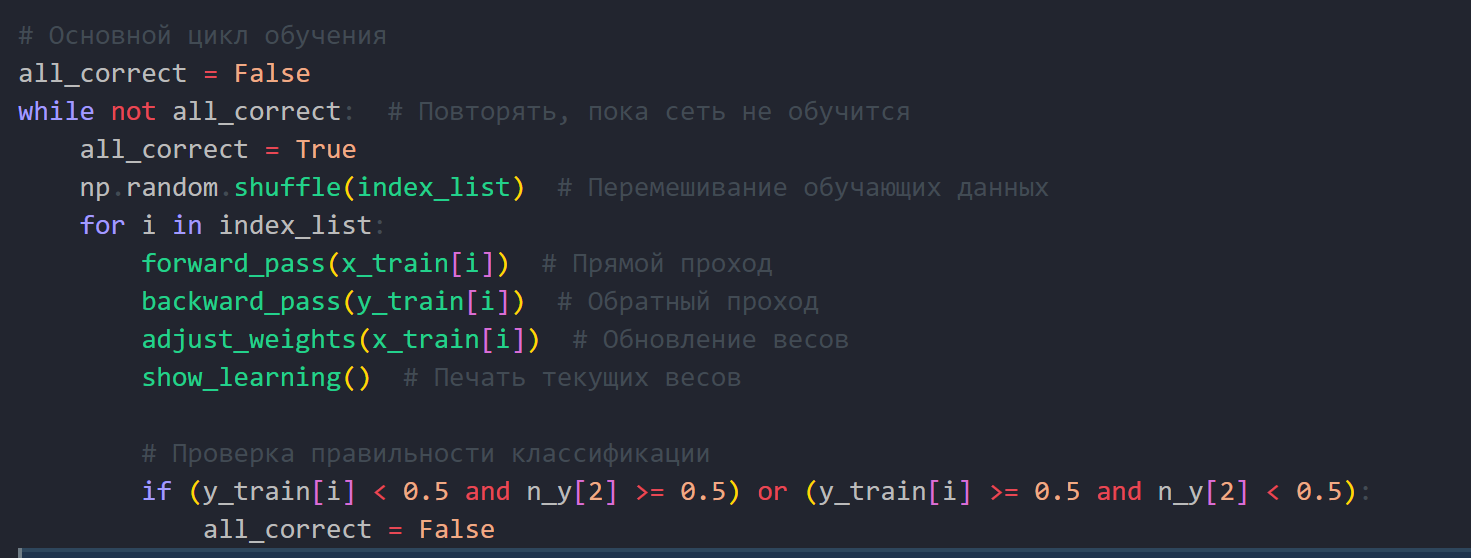
После вычисления ошибок корректируются веса нейронов (рис. 3).



* Рисунок 3 - Обновление весов.

На этом этапе веса корректируются с учетом ошибок и входных данных для каждого нейрона.

Алгоритм повторяет шаги прямого и обратного прохода до тех пор, пока сеть не классифицирует все примеры правильно (рис. 4).



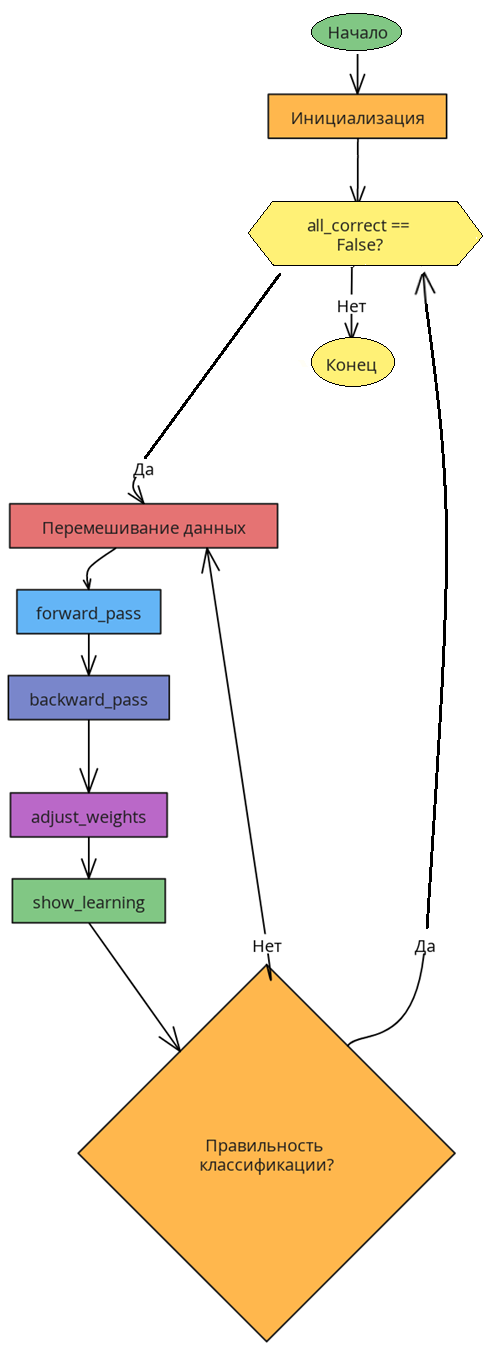
* Рисунок 4 - Цикл обучения.

На этом этапе:

* Для всех примеров выполняются forward-pass, backward-pass и обновление весов.
* Перемешивание данных (np.random.shuffle) предотвращает переобучение на одном порядке данных.
* Проверяется, правильно ли сеть классифицирует каждый пример.

### Блок-схема работы алгоритма.

Блок-схема работы алгоритма представлена на рисунке 5.



* Рисунок 5 - Блок-схема работы алгоритма.

1. Инициализация параметров

* Устанавливается фиксированное начальное состояние генератора случайных чисел np.random.seed(3) для воспроизводимости результатов.
* Определяется скорость обучения LEARNING\_RATE = 0.1.
* Задаются индексы обучающих данных index\_list = [0, 1, 2, 3].
* Формируются входные данные x\_train и истинные значения y\_train.

2. Инициализация весов нейронов

* Каждый нейрон получает случайные начальные веса с помощью функции neuron\_w().
* Весы включают смещение и рандомизируются в пределах от -1.0 до 1.0.
* Веса нейронов хранятся в массиве n\_w.
* Прямой проход (forward\_pass)
* Рассчитываются выходы скрытых нейронов с использованием функции активации tanh.
* Выход скрытых нейронов используется для формирования входа выходного нейрона.
* Для выходного нейрона применяется логистическая сигмоида для получения финального результата.

4. Обратный проход (backward\_pass)

* Вычисляется ошибка выходного нейрона (n\_error[2]) как разница между целевым значением и предсказанием.
* Ошибка распространяется на скрытые нейроны через веса выходного нейрона и производные функций активации.

5. Обновление весов (adjust\_weights)

* Веса каждого нейрона корректируются на основе ошибок, скорости обучения и соответствующих входных значений.

6. Основной цикл обучения

* Устанавливается флаг all\_correct = False, чтобы обучение продолжалось, пока сеть не классифицирует все данные правильно.
* Выполняется перемешивание данных для улучшения качества обучения (np.random.shuffle(index\_list)).
* Для каждого примера из обучающего набора выполняются:
* forward\_pass для расчета предсказаний.
* backward\_pass для вычисления ошибок.
* adjust\_weights для обновления весов.
* После каждой итерации проверяется корректность предсказаний. Если сеть классифицирует пример неверно, флаг all\_correct сбрасывается, и обучение продолжается.

7. Вывод текущих весов

* В конце каждой итерации вызывается функция show\_learning, которая отображает обновленные веса каждого нейрона.

ЗАКЛЮЧЕНИЕ

В ходе выполнения лабораторной работы была разработана и протестирована двухслойная нейронная сеть для решения задачи реализации логической функции "исключающее ИЛИ" (XOR).

Основной целью было изучение алгоритма обратного распространения ошибки (backpropagation), который позволяет корректировать веса нейронной сети для достижения точных результатов. Алгоритм был реализован в Python с использованием библиотеки NumPy.

В процессе работы:

1. Реализован этап инициализации весов и параметров сети.
2. Проведен анализ работы функций прямого и обратного прохода.
3. Выполнено обучение сети с использованием циклического процесса обновления весов до достижения заданной точности классификации.
4. Исследована зависимость корректировки весов от ошибок предсказания и скорости обучения.

Сеть успешно обучилась классифицировать все входные данные, что подтверждает правильность реализации алгоритма. Полученные результаты и визуализация этапов обучения продемонстрировали эффективность алгоритма обратного распространения ошибки.

Лабораторная работа выполнена успешно, поставленные задачи достигнуты. Алгоритм может быть использован в дальнейшем для реализации более сложных задач.

# 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 numpy as np

# Инициализация параметров

np.random.seed(3)  # Фиксация случайного состояния

LEARNING\_RATE = 0.1  # Скорость обучения

index\_list = [0, 1, 2, 3]  # Индексы обучающих примеров

# Обучающие данные (входы и истинные значения)

x\_train = np.array(

    [

        [1.0, -1.0, -1.0],  # x1 = 1.0, x2 = -1.0

        [1.0, -1.0, 1.0],  # x1 = 1.0, x2 = 1.0

        [1.0, 1.0, -1.0],  # x1 = -1.0, x2 = -1.0

        [1.0, 1.0, 1.0],  # x1 = -1.0, x2 = 1.0

    ]

)

y\_train = np.array(

    [

        0.0,  # Целевое значение для первого примера

        1.0,  # Целевое значение для второго примера

        1.0,  # Целевое значение для третьего примера

        0.0,  # Целевое значение для четвертого примера

    ]

)

# Инициализация весов нейронов

def neuron\_w(input\_count):

    weights = np.zeros(input\_count + 1)  # Включая вес смещения

    for i in range(1, input\_count + 1):

        weights[i] = np.random.uniform(-1.0, 1.0)  # Рандомизация весов

    return weights

n\_w = [neuron\_w(2), neuron\_w(2), neuron\_w(2)]  # Веса всех нейронов

n\_y = [0, 0, 0]  # Выходы нейронов

n\_error = [0, 0, 0]  # Ошибки нейронов

# Функция прямого прохода

def forward\_pass(x):

    global n\_y

    n\_y[0] = np.tanh(np.dot(n\_w[0], x))  # Выход первого скрытого нейрона

    n\_y[1] = np.tanh(np.dot(n\_w[1], x))  # Выход второго скрытого нейрона

    n2\_inputs = np.array([1.0, n\_y[0], n\_y[1]])  # Входы для выходного нейрона

    z2 = np.dot(n\_w[2], n2\_inputs)  # Линейная комбинация для выходного нейрона

    n\_y[2] = 1.0 / (1.0 + np.exp(-z2))  # Сигмоидальная активация

# Функция обратного прохода

def backward\_pass(y\_truth):

    global n\_error

    # Вычисление ошибки для выходного нейрона

    error\_prime = y\_truth - n\_y[2]

    derivative = n\_y[2] \* (1.0 - n\_y[2])  # Производная сигмоиды

    n\_error[2] = error\_prime \* derivative  # Градиент

    # Ошибки для скрытых нейронов

    derivative = 1.0 - n\_y[0] \*\* 2  # Производная tanh

    n\_error[0] = n\_w[2][1] \* n\_error[2] \* derivative

    derivative = 1.0 - n\_y[1] \*\* 2  # Производная tanh

    n\_error[1] = n\_w[2][2] \* n\_error[2] \* derivative

# Функция обновления весов

def adjust\_weights(x):

    global n\_w

    n\_w[0] += x \* LEARNING\_RATE \* n\_error[0]  # Обновление весов первого нейрона

    n\_w[1] += x \* LEARNING\_RATE \* n\_error[1]  # Обновление весов второго нейрона

    n2\_inputs = np.array([1.0, n\_y[0], n\_y[1]])  # Входы для выходного нейрона

    n\_w[2] += (

        n2\_inputs \* LEARNING\_RATE \* n\_error[2]

    )  # Обновление весов выходного нейрона

# Функция для отображения текущих весов каждого нейрона

def show\_learning():

    print("Current weights:")

    for i, w in enumerate(n\_w):

        print(

            "neuron ",

            i,

            ": w0 =",

            "%.2f" % w[0],

            ", w1 =",

            "%.2f" % w[1],

            ", w2 =",

            "%.2f" % w[2],

        )

    print("----------------")

# Основной цикл обучения

all\_correct = False

while not all\_correct:  # Повторять, пока сеть не обучится

    all\_correct = True

    np.random.shuffle(index\_list)  # Перемешивание обучающих данных

    for i in index\_list:

        forward\_pass(x\_train[i])  # Прямой проход

        backward\_pass(y\_train[i])  # Обратный проход

        adjust\_weights(x\_train[i])  # Обновление весов

        show\_learning()  # Печать текущих весов

        # Проверка правильности классификации

        if (y\_train[i] < 0.5 and n\_y[2] >= 0.5) or (y\_train[i] >= 0.5 and n\_y[2] < 0.5):

            all\_correct = False