**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра ИИСТ**

отчет

**по индивидуальному домашнему заданию №2.6**

**по дисциплине «Программирование»**

|  |  |  |
| --- | --- | --- |
| Студент гр. 2586 |  | Нефёдов Е.А. |
| Преподаватель |  | Беляев Ф.Я. |

Санкт-Петербург

2024

**Цель работы.**

Нейросеть.**Задание.**

*Техническое задание*

1. Создать модель нейронной сети для распознавания образов 10\*10 точек.

2. Предусмотреть распознавание первых трех уникальных букв фамилии студента.

3. Хранение образов предусмотреть в текстовых файлах (json или строки), используя символы '0' и '1'.

4. Сеть должна выдавать результат для заданного файла примера, переданного через параметры командной строки.

5. Предусмотреть возможность обучения нейронной сети для распознавания других примеров.

6. Оценить количество необходимых циклов для обучения сети.

7. Оценить точность распознавания относительно внесенной случайной ошибки.

*Технические требования*

1) Для решения задачи использовать библиотеку numpy.

2) Предпочтительный алгоритм обучения - Обратное распространение ошибки.

*Рекомендации*

a) количество нейронов + 1 к количеству распознаваемых примеров

б) обучение можно проводить на зашумленных сигналах (numpy тут очень пригодится)

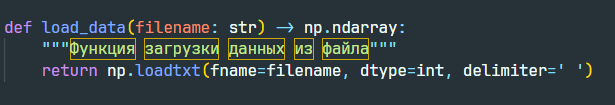
в) предел зашумления ограничить 50%

г) в качестве функции активации отлично подходит сигмоидальная функция

**Решение задачи.**

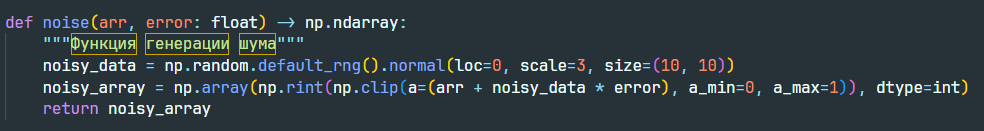
1. Функция загрузки данных из файла.

Реализуем функцию загрузки данных, которые будет обрабатывать нейронная сеть, из файла.



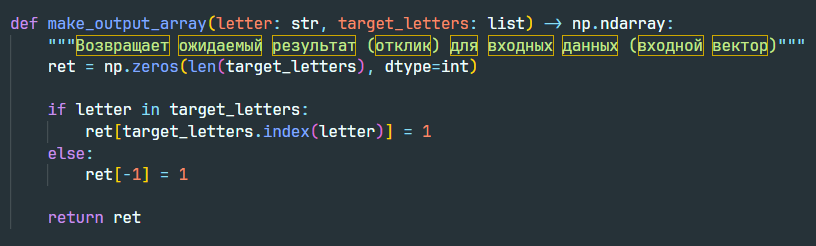
1. Функция генерации шума.

Реализуем функцию для зашумления данных. Это нужно для более качественного обучения.



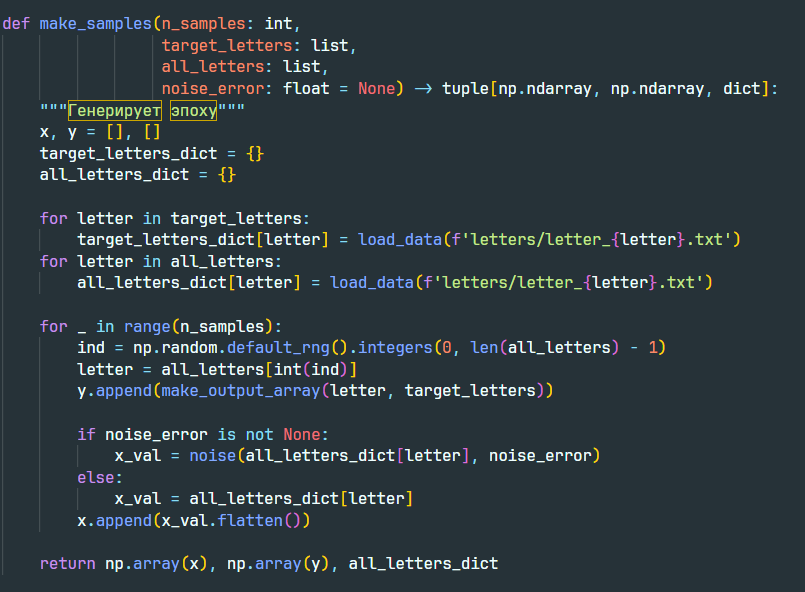
1. Функция создания отклика для входных данных.

Для того чтобы обучение было возможно нужно сопоставить входные данные с ожидаемым результатом на выходе.

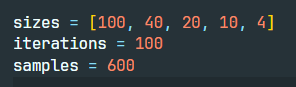


1. Функция генерации эпохи.

Эта функция возвращает 2 двумерных массива. В первом входные данные. Во втором соответствующие им ожидаемые выходные данные (отклик).



1. Подготовка настроек нейронной сети.



Здесь мы определяем количество нейронов и слоёв, количество итераций обучения и количество примеров для обучения.

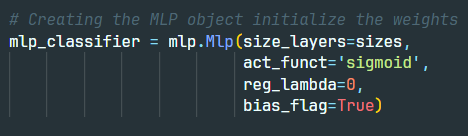
1. Разбиение эпохи на обучающую и тестовую выборку.

Для обучения всю эпоху нужно разделить минимум на две выборки — тренировочную и тестовую. Для этого будем использовать функцию train\_test\_split() из библиотеки scikit-learn.

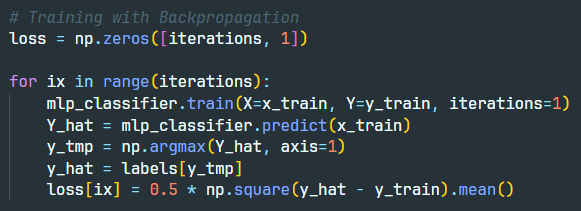


1. Создание экземпляра нейронной сети.

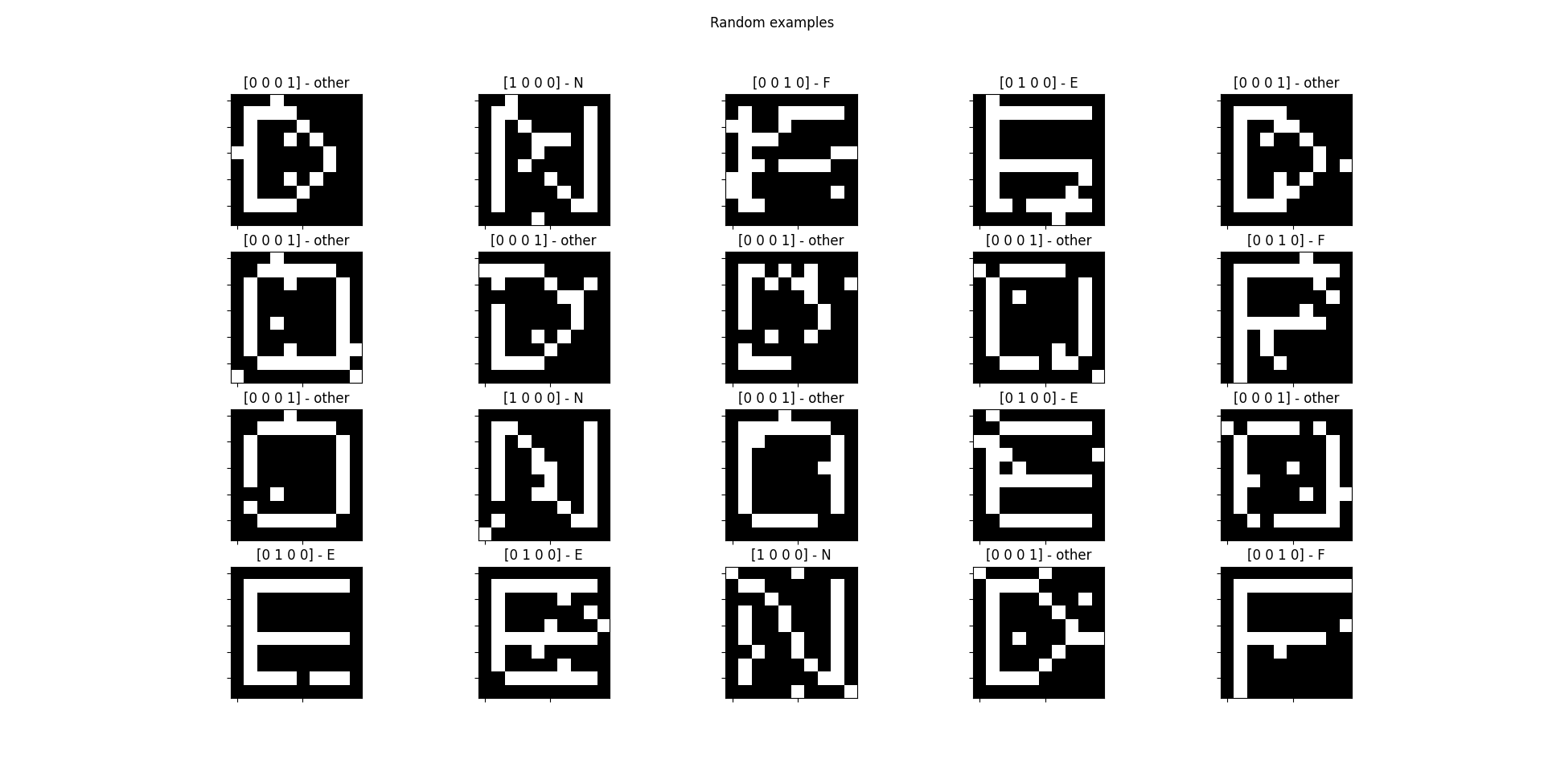
Ф качестве функции активации будем использовать сигмоид.



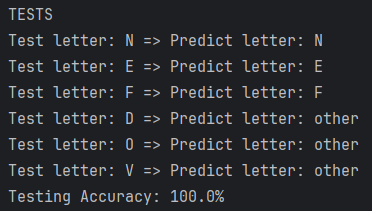
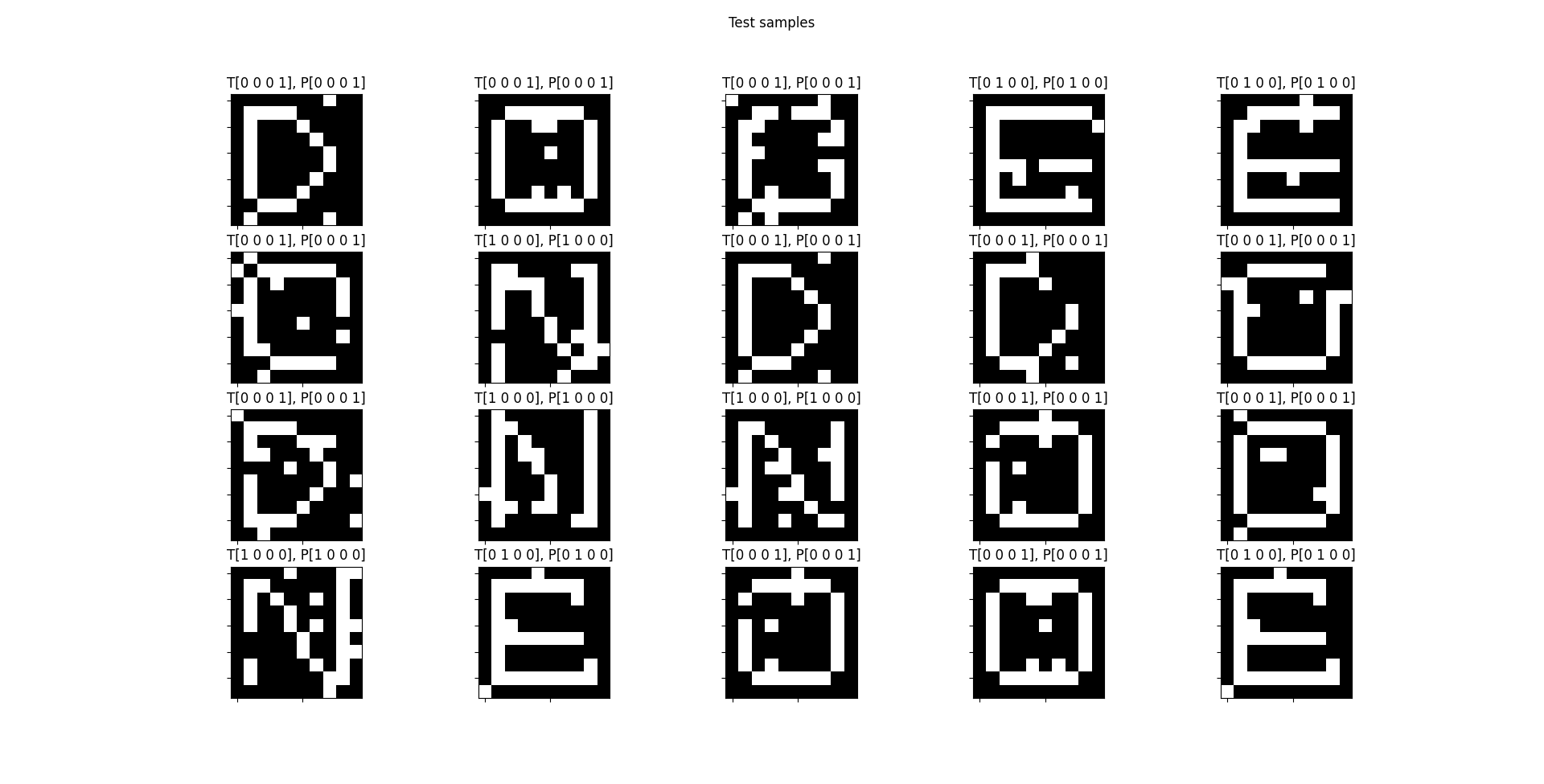
1. Обучение при помощи алгоритма Backpropagation.



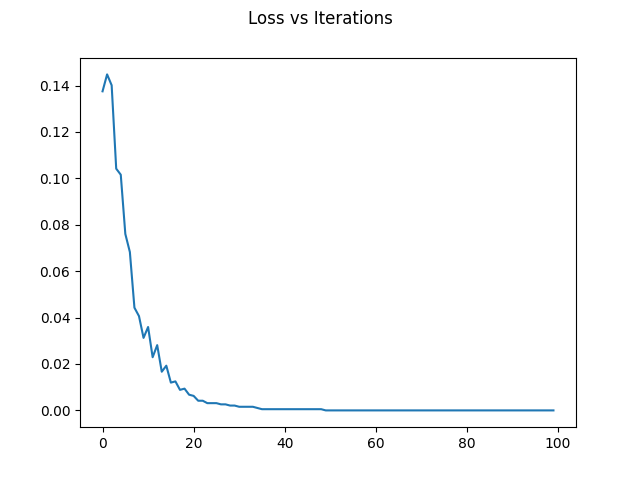
1. Примеры входных данных



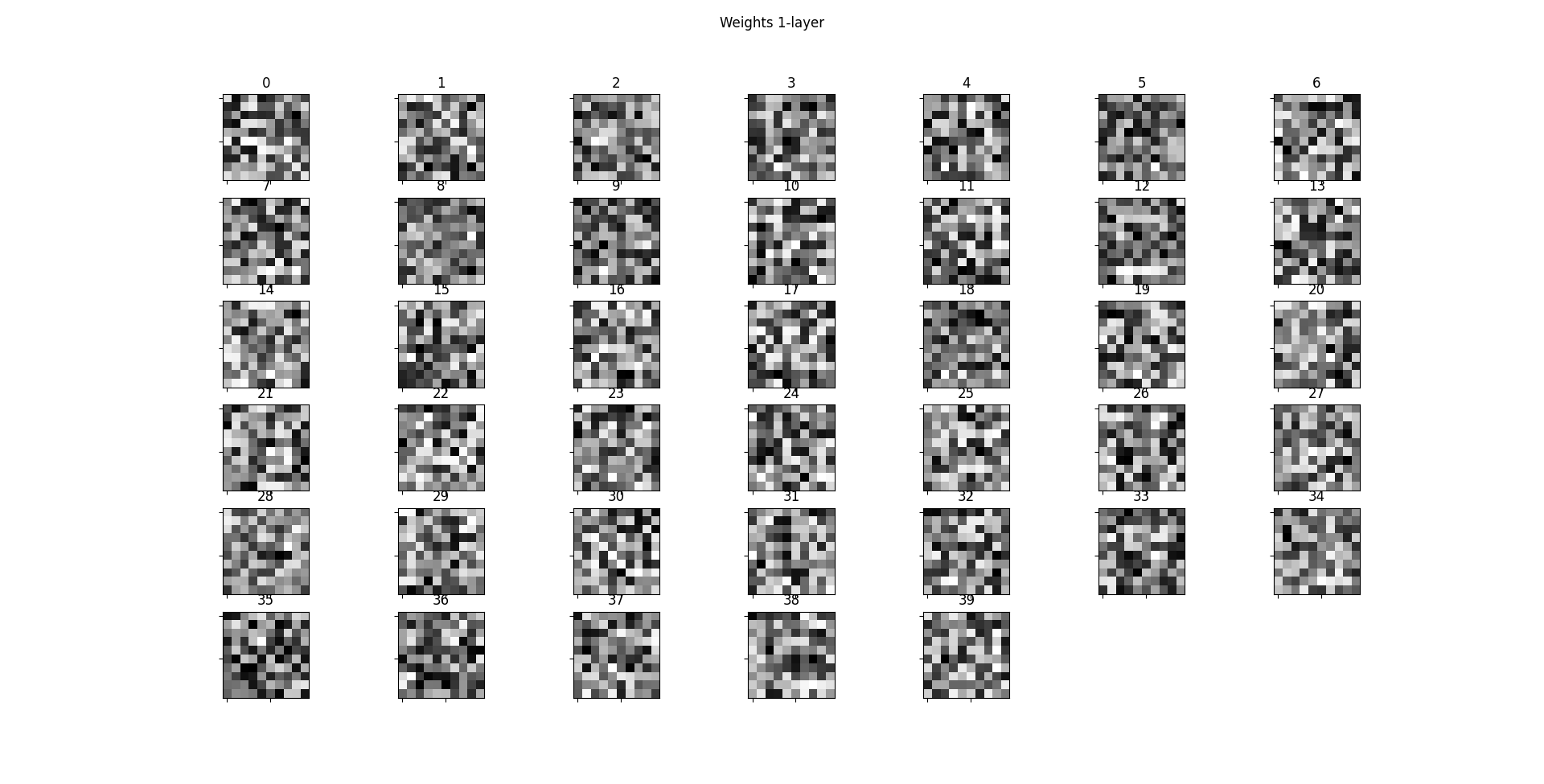
1. Тесты.

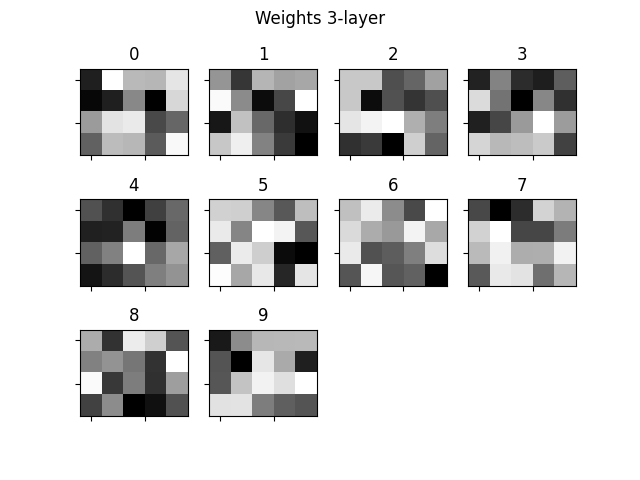
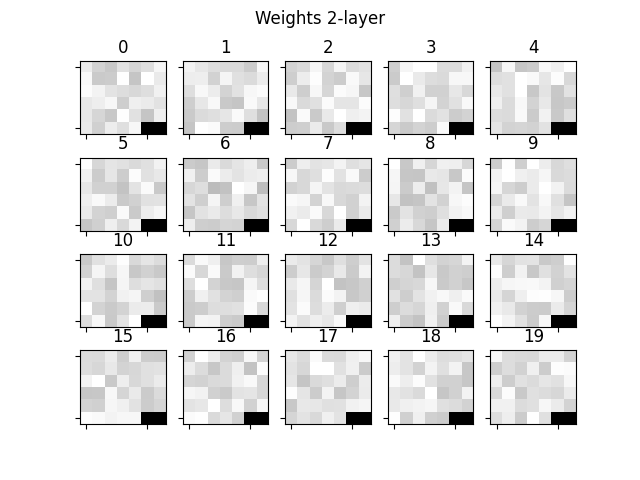
 

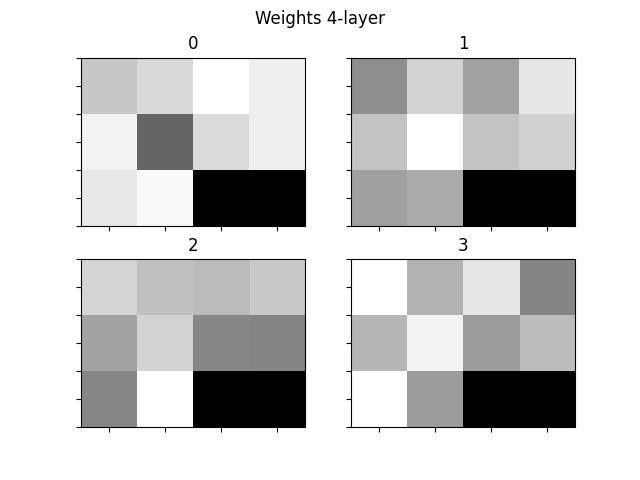
1. Точность.



1. Весовые коэффициенты на разных слоях нейронов.







**Выводы.**

1. Реализована 1 функция загрузки данных из файла load\_data().
2. Реализована 1 функция генерации шума noise().
3. Реализована 1 функция создания отклика для входных данных make\_output\_array().
4. Реализована 1 функция генерации эпохи make\_samples().
5. Создано 3 переменных, отвечающих за конфигурацию нейросети и процесс её обучения: sizes, iterations, samples.
6. Получено 4 массива с тренировочными и тестовыми данными: x\_train, x\_test, y\_train, y\_test.
7. Создан ё1 экземпляр класса, представляющего нейросеть — mlp\_classifier.
8. Произведено обучение нейросети при помощи алгоритма Backpropagation.
9. Получено 20 случайных примеров входных данных.
10. Проведены 6 тестов на исходных данных и отображены результаты 20 случайных тестов на зашумленных данных.
11. Построен график зависимости критерия качества от количества итераций.
12. Получены изображения итоговых весов на каждом нейроне. В данном случае 174 нейрона.

Приложение 1

*Исходный код*

"""ИДЗ №2.6 Нефёдов Е.А. группа 2586"""

import numpy as np

import mlp

import matplotlib.pyplot as plt

from sklearn.model\_selection import train\_test\_split

import json

def load\_data(filename: str) -> np.ndarray:

    """Функция загрузки данных из файла"""

    return np.loadtxt(fname=filename, dtype=int, delimiter=' ')

def noise(arr, error: float) -> np.ndarray:

    """Функция генерации шума"""

    noisy\_data = np.random.default\_rng().normal(loc=0, scale=3, size=(10, 10))

    noisy\_array = np.array(np.rint(np.clip(a=(arr + noisy\_data \* error), a\_min=0, a\_max=1)), dtype=int)

    return noisy\_array

def make\_output\_array(letter: str, target\_letters: list) -> np.ndarray:

    """Возвращает ожидаемый результат (отклик) для входных данных (входной вектор)"""

    ret = np.zeros(len(target\_letters), dtype=int)

    if letter in target\_letters:

        ret[target\_letters.index(letter)] = 1

    else:

        ret[-1] = 1

    return ret

def make\_samples(n\_samples: int,

                 target\_letters: list,

                 all\_letters: list,

                 noise\_error: float = None) -> tuple[np.ndarray, np.ndarray, dict]:

    """Генерирует эпоху"""

    x, y = [], []

    target\_letters\_dict = {}

    all\_letters\_dict = {}

    for letter in target\_letters:

        target\_letters\_dict[letter] = load\_data(f'letters/letter\_{letter}.txt')

    for letter in all\_letters:

        all\_letters\_dict[letter] = load\_data(f'letters/letter\_{letter}.txt')

    for \_ in range(n\_samples):

        ind = np.random.default\_rng().integers(0, len(all\_letters) - 1)

        letter = all\_letters[int(ind)]

        y.append(make\_output\_array(letter, target\_letters))

        if noise\_error is not None:

            x\_val = noise(all\_letters\_dict[letter], noise\_error)

        else:

            x\_val = all\_letters\_dict[letter]

        x.append(x\_val.flatten())

    return np.array(x), np.array(y), all\_letters\_dict

*# samples*

target\_letters = ['N', 'E', 'F', '\_']

all\_letters = ['N', 'E', 'F', 'D', 'O', 'V']

out\_letters = {0: 'N',

               1: 'E',

               2: 'F',

               3: 'other'}

sizes = [100, 40, 20, 10, 4]

iterations = 100

samples = 600

*# Подготовка эпохи*

x, y, ald = make\_samples(samples, target\_letters, all\_letters, 0.11)

*# Разбиение эпохи на обучающую и тестовую выборку*

x\_train, x\_test, y\_train, y\_test = train\_test\_split(x, y, test\_size=0.2, random\_state=20)

*# Training data*

train\_X = x\_train

train\_y = y\_train

print(f'Shape of training set: {train\_X.shape}')

*# change y [1D] to Y [2D] sparse array coding class*

n\_examples = len(y\_train)

labels = np.unique(y\_train, axis=0)[::-1]

train\_Y = np.zeros((n\_examples, len(labels)))

for ix\_label in range(len(labels)):

*# Find examples with a Label = lables(ix\_label)*

    ix\_tmp = np.where(train\_y == labels[ix\_label])[0]

    train\_Y[ix\_tmp, ix\_label] = 1

*# Test data*

test\_X = x\_test

test\_y = y\_test

print(f'Shape of test set: {test\_X.shape}')

examples = np.random.default\_rng().integers(len(test\_y), size=10)

*# change y [1D] to Y [2D] sparse array coding class*

n\_examples = len(test\_y)

labels = np.unique(test\_y, axis=0)[::-1]

test\_Y = np.zeros((n\_examples, len(labels)))

for ix\_label in range(len(labels)):

*# Find examples with a Label = lables(ix\_label)*

    ix\_tmp = np.where(test\_y == labels[ix\_label])[0]

    test\_Y[ix\_tmp, ix\_label] = 1

*# Creating the MLP object initialize the weights*

mlp\_classifier = mlp.Mlp(size\_layers=sizes,

                         act\_funct='sigmoid',

                         reg\_lambda=0,

                         bias\_flag=True)

*# Training with Backpropagation*

loss = np.zeros([iterations, 1])

for ix in range(iterations):

    mlp\_classifier.train(X=x\_train, Y=y\_train, iterations=1)

    Y\_hat = mlp\_classifier.predict(x\_train)

    y\_tmp = np.argmax(Y\_hat, axis=1)

    y\_hat = labels[y\_tmp]

    loss[ix] = 0.5 \* np.square(y\_hat - y\_train).mean()

*# Console tests*

print('TESTS')

for l in all\_letters:

    Y\_hat = mlp\_classifier.predict(np.array([ald[l].flatten()]))

    y\_tmp = np.argmax(Y\_hat, axis=1)

    print(f'Test letter: {l} => Predict letter: {out\_letters[y\_tmp[0]]}')

*# Test Accuracy*

Y\_hat = mlp\_classifier.predict(test\_X)

y\_tmp = np.argmax(Y\_hat, axis=1)

y\_hat = labels[y\_tmp]

acc = np.mean(1 \* (y\_hat == test\_y))

print(f'Testing Accuracy: {acc \* 100}%')