ДОКУМЕНТАЦИЯ

По программе Neural Network for Tempest

г. Саров.

2020 г.

Оглавление

[Примечание: 3](#_Toc43112426)

[1. Интерфейс программы 4](#_Toc43112427)

[Вкладка MLP. 5](#_Toc43112428)

[Вкладка Synth. 9](#_Toc43112429)

[Вкладка Tempest. 13](#_Toc43112430)

[Окно построения графиков. 14](#_Toc43112431)

[2. Внутреннее устройство программы 15](#_Toc43112432)

[Алгоритм вычислений. 15](#_Toc43112433)

[Файловая иерархия программы. 16](#_Toc43112434)

[Программная реализация. 23](#_Toc43112435)

[Библиотека PyQt5 23](#_Toc43112436)

[Библиотеки numpy и scipy 25](#_Toc43112437)

[Бибилотека scikit-learn 25](#_Toc43112438)

[Дополнительная информация. 26](#_Toc43112439)

[3. Масштабируемость 27](#_Toc43112440)

[Добавление новых сигналов, записанных приёмником электромагнитных сигналов. 28](#_Toc43112441)

[Добавление новых обучающих выборок. 29](#_Toc43112442)

[4. Возможности по улучшению программы 32](#_Toc43112443)

[Улучшение оценки эффективности. 32](#_Toc43112444)

[Оптимизация работы с большими массивами. 33](#_Toc43112445)

[Изменение структуры нейронной сети. 34](#_Toc43112446)

[Изменение характера обучающей выборки. 35](#_Toc43112447)

## Примечания:

Данная работа несёт исключительно исследовательский характер. Этот проект изначально был направлен на изучение влияния методов машинного обучения на обработку и регистрацию ПЭМИ. Машинное обучение – один из перспективных средств обработки сигналов, который рано или поздно войдёт во все аспекты развития технических систем.

Первоначальный автор данной работы имеет весьма скудный профессиональный лексикон, поэтому в тексте могут встречаться некоторые размытые понятия (например: сигнал, массив) и случаи косноязычия. Данная версия документации является первой, поэтому в дальнейших редакциях всё может кардинально измениться.

Код программы, по возможности, наполнен комментариями, поэтому в документации представлены только очертания процесса работы программы.

Авторы этого проекта:

* Пелин Иван: *ivanpelin1508@yandex.ru*

# Интерфейс программы

Программа задумывалась как средство разработки нейронной сети с последующим её тестированием и использованием для обработки данных полученных приемником электромагнитного сигнала.

Вследствие этого начальное окно было разделено на три вкладки:

* MLP – используется для обучения и загрузки нейронных сетей.
* Synth – используется для генерации и просмотра искусственно синтезированных массивов данных, симулирующих зашумленные электромагнитные сигналы, а также при проведении анализа эффективности определенной нейронной сети.
* Tempest – используется для просмотра, изучения и обработки сигналов, полученных с приемников.

Подробнее остановимся на каждой вкладке.

## Вкладка MLP.

Общий вид данной вкладки представлен на рисунке 1-1:

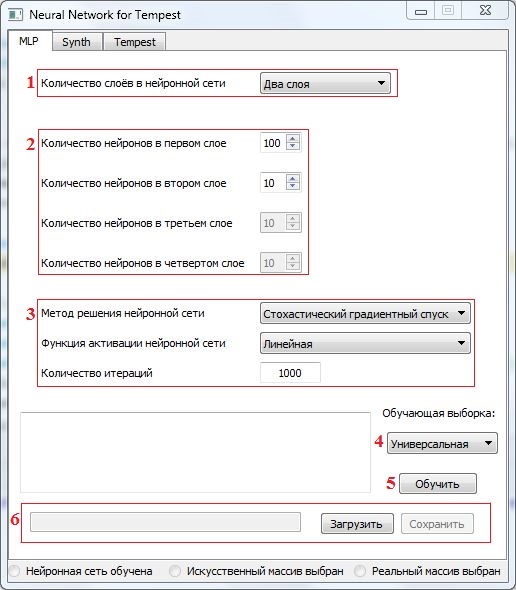


Рисунок 1‑1 Главное окно вкладки MLP

На рисунке выделены функциональные Поля, необходимые для работы с моделями нейронных сетей.

Поля 1 и 2 устанавливают *общую структуру* прямой нейронной сети, то есть указывают количество скрытых слоев в сети прямого распространения и количество нейронов в слоях. При изменении значения выдвигающегося списка в Поле 1, открывается или закрывается доступ к окошкам счётчика в Поле 2, соответственно *количеству слоёв*, выбранных в списке 1.

В Поле 3 выбираются параметры обучения нейронной сети.

Так в *методе обучения* имеются три варианта: Стохастический Градиентный Спуск (СГС, *sgd*), Квази-Ньютоновский метод (*lbfgs*) и Adam (модифицированная вариация СГС, описанная Kingma, Diederik, и Jimmy Ba). Подробнее об этих методах можно найти на сторонних ресурсах. **Рекомендации по использованию:** СГС медленнее Квази-Ньютоновского на небольших выборках; на больших выборках (понятие неточно, рекомендуется придерживаться величины в 100 000 примеров для обучения) рекомендуется использовать Adam.

*Функция обучения* выбирается из четырех вариантов, представленных в таблице 1.

*Количество итераций* представляет количество эпох, которые алгоритм обучения должен пройти. Одна эпоха продставляет полный цикл обучения на одной выборке со всеми примерами. Большое количество итераций займёт больше времени, но позволит добиться лучшей обучаемости; при этом, слишком большое количество итераций приведет к переобучению, что даст низкую степень обобщающей способности нейронной сети и модель нейронной сети не сможет справлятся должным образом с другими сигналами. Также, при сложной структуре нейронной сети, алгоритм обучения не сможет сойтись за низкое количество итераций (о чем программа предупредит) и нейронная сеть будет «пустой».

Выдвигающийся список 4 даёт возможность выбрать *обучающую выборку*. По умолчанию там стоит Универсальная обучающая выборка, которая представляет из себя комбинацию из 4 имеющихся обучающих выборок, созданных вручную. Принцип создания и добавления обучающих выборок представлены в части 3 «Масштабируемость».

Кнопка 5 инициирует *обучение модели* нейронной сети по заданным в блоках 1-4 параметрам. Текстовое поле в текущем окне показывает Лог-данные за текущую сессию.

Блок 6 управляет всеми имеющимися моделями нейронных сетей. Нажав кнопку «Сохранить», можно сохранить непосредственно обученную модель нейронной сети. Нажав кнопку «Загрузить», откроется небольшое окно (Рис.1-2), в котором будут все обученные до этого модели.

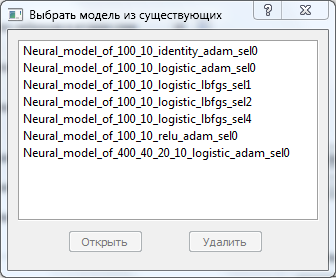
В этом окне можно либо кликнуть два раза на название необходимой модели, либо выбрать модель и на кнопку «Открыть», чтобы загрузить модель в текущую сессию. Кнопка «Удалить» удаляет выбранную модель с компьютера.

Рисунок 1‑2. Окно загрузки модели НС.

Вернёмся к главному окну. После обучения модели или загрузки из существующих, текстовое поле в Блоке 6 измениться и будет содержать название текущей модели.

Название модели формируется из:



Заголовок

Структура

НС

Функция

активации

Метод

обучения

Выборка

Рисунок 1‑3. Строка названия модели.

При этом номер выборки берётся прямо из выдвигающегося списка 4, то есть нулевая выбока – Универсальная, следующая за ней в списке – sel1, и т.д.

Таблица 1. Список функций активации нейрона

|  |  |  |
| --- | --- | --- |
| Название | Формула | График функции |
| Линейная функция  (*identity*) |  |  |
| Сигмоидальная функция  (*logistic*) |  |  |
| Положительная линейная функция  (*relu*) |  |  |
| Тангенциальная функция  (*tanh*) |  |  |

## Вкладка Synth.

Общий вид вкладки показан на рисунке 1-4:

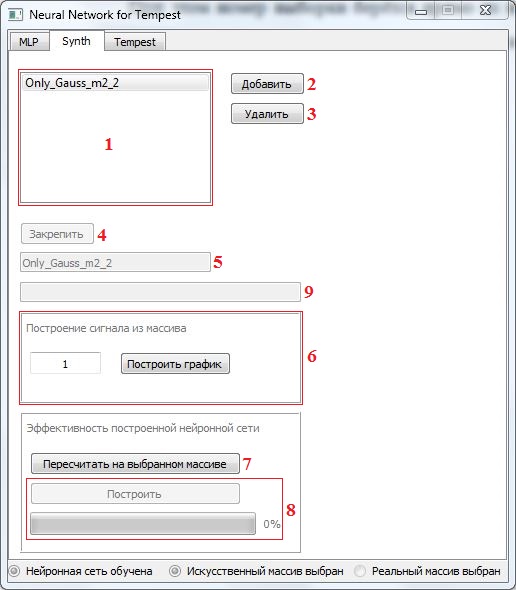


Рисунок 1‑4. Главное окно вкладки Synth.

В Поле 1 расположен список. В этом списке содержатся все массивы, созданные и хранящиеся на компьютере. При двойном клике на определенный массив открывается окно описания массива (Рис 1-5).

В открывшемся окне можно увидеть все свойства выбранного массива. Количество выборок указывает, сколько в указанном диапазоне шумов взято отсчётов, на которых пересчитывается отношение сигнал/шум и строится на этом отношении пример сигнала.

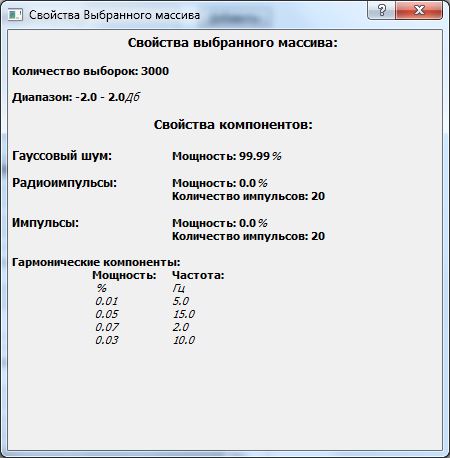


Рисунок 1‑5 Окно описания массивов.

То есть: Возьмем массив, выбранный на рисунке 1-5. Интервал отношений сигнал/шум для данного массива: (-2 дБ; 2 дБ). При этом количество выборок равно 3000. Это значит, что первый искусственно построенный сигнал будет строится на отношении сигнал/шум равном -2 дБ. А уже следующий за ним в массиве сигнал будет строится на ОСШ равном дБ. И так для каждого следующего сигнала вплоть до 2 дБ.

Для всех компонент имеется такой параметр как «Мощность». Этот параметр указан в процентах. Проценты показывают отношение абсолютной мощности данной компоненты к мощности шума. Это было сделано с целью максимально тонко определять зашумленный массив. Для импульсов имеется такой параметр как «Количество импульсов». Этот параметр указывает, сколько импульсов будет расположено в сформированном сигнале на всей области определения. И «Частота» для гармонических шумов, соответственно, означает частоту гармонической компоненты с указанной мощностью.

Вернёмся в основное окно. Кнопка 2 «Добавить» позволяет создать новый массив. При нажатии на данную кнопку открывается новое окно создания массива (Рис. 1-6).

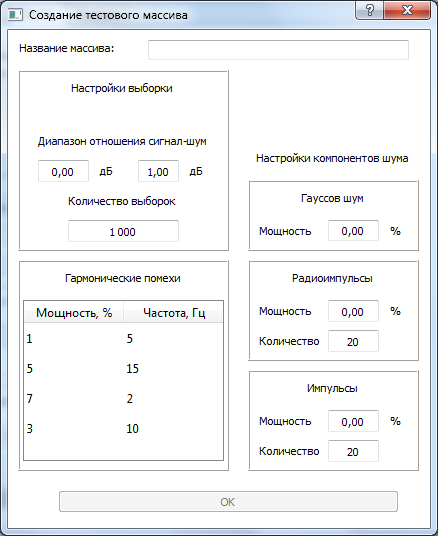


Рисунок 1‑6. Окно создания массива.

В самом верхнем текстовом поле можно указать пользовательское название массива. Все остальные поля являются настраиваемыми параметрами массива, которые были описаны ранее. И кнопка «ОК» закрывает окно и начинает создание массива. После завершения данной операции название массива появляется в Списке 1.

Кнопка 3 «Удалить» удаляет массив из списка и стирает его данные с компьютера.

При выборе массива и нажатии на Кнопку 4 «Закрепить» начинается процесс загрузки выбранного массива в оперативную память. После завершения данного процесса в Текстовом поле 5 закрепляется название выбранного массива, и после этого с данным массивом можно работать.

Работа с массивом на данном этапе разработки заключается только в просмотре определенных примеров сигнала, т.е. построение их на плоскости, и в проверке эффективности нейронной сети на выбранной в первой вкладке модели нейронной сети.

Так, в Поле 6 изменение значения в Окошке счетчика устанавливает значение ОСШ определенного примера из массива, а кнопка «Построить» строит его график.

Когда определенный массив загружен, а также загружена модель нейронной сети, становится активной Кнопка 7, которая запускает процесс пересчёта эффективности модели нейронной сети на всех примерах массива, таким образом, можно получить характеристику эффективности нейронной сети от отношения сигнал/шум. При нажатии на кнопку в Текстовом поле 9 появляется название текущей модели нейронной сети и становится активным Индикатор выполнения в Поле 8, и он показывает степень выполнения пересчёта (пересчёт не является критически важным процессом, так что закрытие программы просто остановит его). После выполнения пересчёта становится активной кнопка «Построить» в Поле 8. Нажатие на данную кнопку позволяет построить график характеристики.

## Вкладка Tempest.

Общий вид вкладки показан на рисунке 1-7.

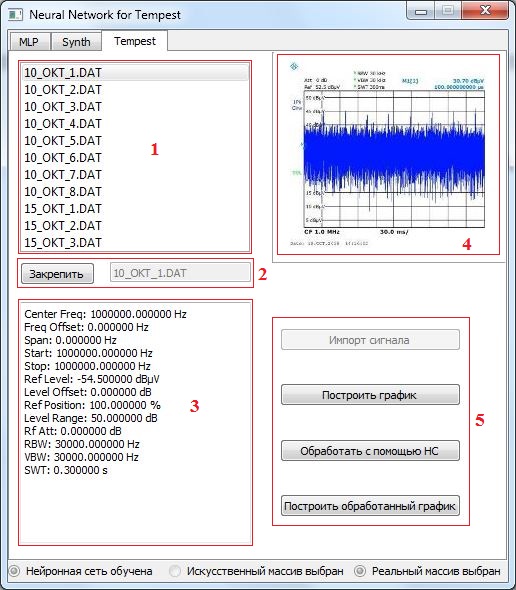


Рисунок 1‑7. Вкладка Tempest.

Данная вкладка позволяет работать с сигналами ПЭМИ, полученными с приемников. Список 1 содержит все имеющиеся на данный момент файлы с сигналами. При выборе определенного названия в Поле 4 появляется миниатюра, изображающая данный сигнал. Изображение должно быть получено с приёмника, как и файл формата .DAT. После чего они могут быть добавлены кнопкой «Импорт сигнала» в Поле 5, но об этом будет подробнее указано в Главе 3 «Масштабируемость».

При выборе определенного сигнала, двойным щелчком мыши по названию или нажатием кнопки «Закрепить» в Поле 2, данный сигнал загружается в оперативную память и его название появляется в Текстовом поле Поля 2. После этого в Поле 3 закрепляется общая информация о выбранном сигнале. Обычно эта информация формируется приемником в качестве шапки файла формата .DAT.

## Окно построения графиков.

Для построения графиков используется универсальное окно, представленное на рисунке 1-8.

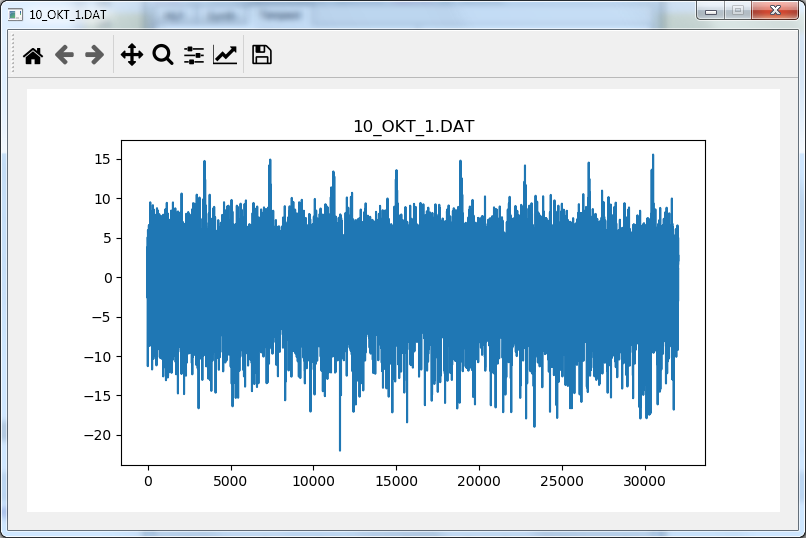


Рисунок 1‑8. Окно построения графиков.

# Внутреннее устройство программы

## Алгоритм вычислений.

Алгоритм внутренних вычислений подробно описан в отчёте на конкурс работ № 195-77/40754 от 25.05.2020. После данной работы были также введены несколько совершенствований, направленных на улучшение эффективности обучаемой модели нейронной сети и универсальность обработки принимаемого реального сигнала.

В качестве мер по улучшению эффективности обучаемой модели нейронной сети добавлена возможность расширять обучающую выборку. Подробный механизм будет рассматриваться в части 3 «Масштабируемость». Процесс расширения заключается в объединении всех обучающих массивов, находящихся заархивированными в папке data\_for\_train.

Универсальность обработки принимаемого сигнала обусловлена возможностью обрабатывать запись сигнала с приемника, которая имеет разлчное количество отсчётов (изначально, программа была ориентирована на сигналы с размерностью 32000 отсчётов). Теперь же в класс обработки файлов \*.DAT, которые записывает приёмник, добавлена переменная, которая хранит в себе общее количество отсчётов сигнала и используется для контроля размерности.

## Файловая иерархия программы.

В корневой папке программы расположены 6 функционально-важных директорий проекта:

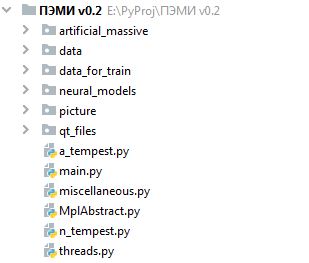


Рисунок 2‑1. Файловое устройство неоткомпилированной программы.

В папке artificial\_massive расположены директории, включающие в себя файлы, связанные с определенным ранее созданным искусственным массивом. В каждой, имеющейся директории расположены строго определённые категории файлов:

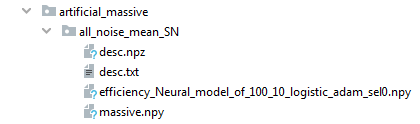


Рисунок 2‑2. Директория artificial\_massive.

Файл desc.npz хранит в себе все заданные пользователем характеристики создания массива, используется для быстрого вывода этих характеристик, носит вспомогательную функцию. Содержит в себе параметры:

* min\_SN – нижняя граница диапазона отношений сигнал/шум;
* max\_SN – верхняя граница диапазона отношений сигнал/шум;
* iteration – количество итераций;
* param – словарь, содержащий все параметры шумов.

Пример кода для работы с данным файлом:

file = np.load(f'artificial\_massive/{'''Название массива'''}/desc.npz')

min\_SN = file["min\_SN"].item() # Записываем в min\_SN соответствующее знач.

gaus = file["param"].item()["gaus"] # Коэффициент гауссового шума

Следующий файл – desc.txt. Содержит всё то же самое, что и предыдущий файл, но в формате текстового файла, и в понятном оформлении. Создан для быстрого ознакомления с параметрами массива вне программы. Имеет следующий формат:

Минимальное отношение сигнал/шум: -5.0

Максимальное отношение сигнал/шум: 5.0

Количество выборок: 2000

Параметры выборки: {'gaus': 1.0, 'impuls': 1.0, 'num\_impuls': 20, 'radio\_impuls': 1.0, 'num\_radio': 20, 'amp\_freq': [(0.01, 5.0), (0.05, 15.0), (0.07, 2.0), (0.03, 10.0)]}

Файл massive.npy непосредственно содержит созданный массив. Массив является двумерным, по первому индексу можно получить выборки сигнала с отношением сигнал/шум равным: , где i – номер индекса.

Файлы формата efficiency\_{?}.npy содержат характеристику эффективности модели нейронной сети, название которой записывается в {?}. Данная характеристика записывается в одномерном массиве и необходима для быстрого доступа к готовым характеристикам. Так как процесс оценки обычно занимает достаточно много времени.

В папке data хранятся все файлы формата \*.DAT и они формируют список во вкладке Tempest.

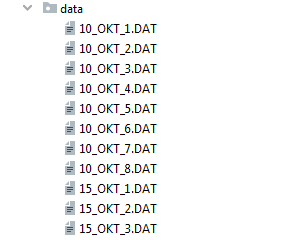


Рисунок 2‑3. Директория data.

Папка data\_for\_train содержит два типа файлов. Это файлы sgnl\_{i}.npy и trn\_{i}.npy, где {i} – это номер обучающей выборки, в первом типе файлов содержатся примеры реальных сигналов, принятых из файлов \*.DAT, обработанных и заархивированных для быстрого доступа к ним. Второй тип файлов – тренировочные выборки, то есть сигналы, вручную избавленные от шумов и поэтому пригодные для использования в качестве требуемого сигнала для соответствующего зашумленного.

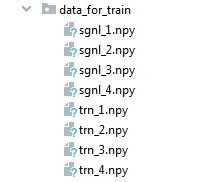


Рисунок 2‑4. Директория data\_for\_train.

Папка neural\_models хранит все модели нейронных сетей, обученные до этого и названные по образцу, представленному на рисунке 1-3. В каждой папке содержаться два файла: model.npy и size.npy. В model.npy заархивирован объект класса MLPRegressor, который содержит все данные об обученной нейронной сети. В файле size.npy содержится массив со значениями количества нейронов в каждом слое (структурой), этот файл несет чисто вспомогательный характер.

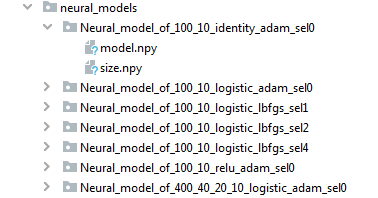


Рисунок 2‑5. Директория neural\_models.

Папка picture предназначена для хранения разного рода изображений, связанных с программой. На данный момент она содержит всего лишь папку mini\_tempest с миниатюрами моделей электромагнитных сигналов, которые отображаются в Поле 4 во вкладке Tempest. Также рекомендую в данную папку сохранять графики, полученные сохранением.

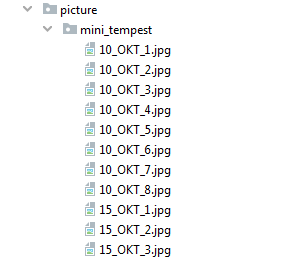


Рисунок 2‑6. Директория picture.

Директория qt\_files предназначена для хранения файлов описания графических окон. Она содержит два типа файлов: \*.ui и \*.py.

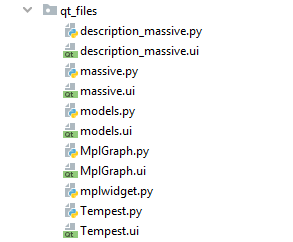


Рисунок 2‑7. Директория qt\_files.

Первые являются промежуточными файлами для графического изменения окон в программе Qt Designer. Данная программа проста, интуитивно понятна, и при любом косметическом редактировании окон рекомендую начинать с неё.

Конвертирование из \*.ui в \*.py исходит через коммандную строку Windows. Для этого необходимо открыть командную строку в корневой папке, перейти в папку qt\_files командой:

cd qt\_files

и следующей командой конвертировать файлы (для примера конвертирование файла Tempest.ui):

pyuic5 Tempest.ui –o Tempest.py

Файлы \*.py в папке qt\_files непосредственно связаны с кодом программы и передают в основной код всю необходимую информацию о графическом наполнении окон.

Перечислю функции всех файлов \*.py в данной папке:

* description\_massive.py – окно описания массива, представленное на рисунке 1-5;
* massive.py – окно создания массива, рисунок 1-6;
* models.py – окно загрузки и удаления моделей нейронных сетей, рисунок 1-2;
* MPLGraph.py – окно отображения графиков, рисунок 1-8;
* mplwidget.py – вспомогательный инструмент для рисования графиков;
* Tempest.py – главное окно программы, рисунок 1-1.

Перейдём в корневую папку. Здесь расположены ключевые файлы \*.py, необходимые для работы программы. Кратко опишу каждый из них:

a\_tempest.py – содержит основные функции обработки и создания сигналов;

main.py – ключевой файл, содержащий описание всех окон;

miscellaneous.py – содержит вспомогательные функции;

MplAbstract.py – содержит абстрактный класс создания окна рисования графика;

n\_tempest.py – содержит класс для работы с файлами \*.DAT;

threads.py – содержит описания потоков для осуществления многопоточного программирования.

## Программная реализация.

В программе, помимо стандартных библиотек Python, используются также библиотеки:

PyQt5 – библиотека для создания графического интерфейса;

numpy – библиотека для расширениz возможностей python в плане работы с массивами;

scipy – библиотека для научных расчётов, в частности в программе применён модуль signal из данной библиотеки, направленный на цифровую обработку сигналов;

scikit-learn – библиотека, направленная на машинное обучение, используется для создания и обучения нейронных сетей.

Остановимся подробнее на каждой библиотеке.

### Библиотека PyQt5

Подробнее PyQt5 можно почитать в книге: Python 3 и PyQt5. Разработка приложений. / Н.А. Прохорёнок, В.А. Дронов. – СПб.: БХВ-Петербург, 2016; - либо на иностранном ресурсе: riverbankcomputing.com.

Не буду вдаваться в подробности функционирования этой библиотеки, Опишу основные моменты, используемые мной при разработке.

Начало редактирования (создания) графического интерфейса, как было указано ранее, рекомендуется начинать с обработки \*.ui файлов в программе QtDesigner. Это достаточно простая программа, и достаточно эффективна для визуального редактирования. После окончания редактирования и сохранения соответствующего файла, необходимо воспользоваться терминалом Windows и кодом со страницы 19 для конвертирования сохранённого файла в код Python.

Привязка полученного кода к основному файлу происходит через импорт сответствующего класса в основной файл и внесением в объект ui рабочего класса имеющегося класса из полученного ранее кода. Например, привязка ui-модуля основного окна к рабочему классу Tempest происходит в следующем фрагменте кода (main.py, строки 14, 107-112):

from qt\_files.Tempest import Ui\_MainWindow

…

class Tempest(QtWidgets.QMainWindow):

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

super(Tempest, self).\_\_init\_\_()

self.massive\_window = None

self.ui = Ui\_MainWindow()

self.ui.setupUi(self)

В этом примере Ui\_MainWindow() – это класс из файла ./qt\_files/Tempest.py, в котором описаны все графические компоненты главного окна. Функция setupUi(self) переносит все графические компоненты в главный класс, что облегчает процесс верстки интерфейса.

Со статичной картинкой на этом всё, дальше в основном классе начинаем разработку логики окна. Для этого описываем необходимое действие определенного элемента и через метод connect() привязываем к действию выполнение определенной программы. Например, так в программе реализуется вызов функции preset\_mlp() из основного класса (через преффикс self.) по нажатию на кнопку «Обучить» (main.py, строка 123):

self.ui.button\_learning.clicked.connect(self.preset\_mlp)

Уже в адресованной функции прописывается реакция на нажатие кнопки.

Следующим важным пунктом, связанным с PyQt5 следует отметить использование многопоточности. Так как некоторые функции требуют очень много времени (например, обучение нейронной сети), то с целью предотвращения зависания основного окна на время выполения функции создаётся дополнительный поток, который производит обработку «долгой» функции и по окончанию обработки выдаёт полученные результаты в основное окно (то есть в основной класс). Для создания и проведения обработки дополнительным классом был добавлен файл ./threads.py. В нём содержатся все необходимые потоки, вызов которых осуществляется в основном классе. То есть логика проста: в основном классе создаётся объект класса потока из ./threads.py, затем дополнительный поток связывается через сигнал с основным, и просто передаёт через сигнал необходимые данные.

Подробная информация о всех перечисленных возможностях PyQt5 есть в книге Python 3 и PyQt5. Разработка приложений. / Н.А. Прохорёнок, В.А. Дронов. – СПб.: БХВ-Петербург, 2016.

### Библиотеки numpy и scipy

Данные библиотеки несут вспомогательную роль в функционировании программы, поэтому, если возникнут вопросы по их реализации, можно посмотреть документацию, которую можно скачать с сайта numpy.org и scipy.org.

### Бибилотека scikit-learn

Данная библиотека предоставляет необхомые функции для реализации машинного обучения. В данной программе эта библиотека отвечает за создание нейронной сети и её обучение.

Нейронная сеть создаётся как объект класса MLPRegressor. Этот класс формирует все свойства прямолинейной полносвязной нейронной сети, и при обучении изменяет их так, как если бы нейронная сеть стала обученной. Таким образом, можно сохранить любой объект данного класса после опрtделенного процесса обучения, и мы будем иметь готовую нейронную сеть.

Подробнее об этой библиотеке можно посмотреть в Mastering Machine Learning with scikit-learn. / Gavin Hackeling. – BIRMINGHAM - MUMBAI, 2017.

## Дополнительная информация.

При проектировании программы, можно пользоваться следующими приёмами:

1. Если разработка проходит в IDLE PyCharm, то можно легко перейти к реализации определенной функции или класса, так как разработчики добавляют информацию о своём коде, к тому же можно посмотреть, как функция работает. Для этого надо нажать правой кнопкой мыши по функции, информацию о которой необходимо узнать: Go To –> Implementations.
2. В IDLE присутствуют отладчики кода. Полезно применять отладку, когда программа выдаёт неверный результат и необходимо посмотреть, где закралась ошибка.
3. Код, по возможности, закомментирован для более легкого понимания.

# Масштабируемость

Программа расcчитана на дальнейшее применение в качестве исследовательского инструмента, поэтому основной целью в процессе разработки было создание универсальности при использовании программы. Под универсальностью понимается возможность добавления новых данных, которые необходимо обработать, при гарантии корректной работы программы. А так же создание и использование различных структур нейронных сетей и тестовых массивов с различными шумами.

Изначально, данный исследовательский проект был направлен на обнаружение утечки сигнала. Начальным материалом для исследования были сигналы с приёмника R&S ESCI 1166.5950.03, записывающий сигналы ПЭМИ с клавиатуры компьютера. Потенциально, данную программу можно использовать для обнаружения различных информационных сигналов. Но для этого программу необходимо расширить не программно, а с добавлением и изменением файлов в папке программы.

Необходимо сохранять файловую структуру программы во избежание критических ошибок. При добавлении файлов в соответствующие папки, о которых будет написано позже, в программе автоматически применятся соответствующие изменения. Это позволяет минимизировать усилия в изменении программы.

Желательно сохранять положительную корреляцию между формой сигнала, на котором была обучена нейронная сеть, и формой сигнала, который обрабатывается. Нейронная сеть направлена на обработку конкретного сигнала, поэтому обработка другого по форме сигнала может не дать необходимого результата.

## Добавление новых сигналов, записанных приёмником электромагнитных сигналов.

При получении новых измерений возможно без программных изменений внести их в программу. Для этого потребуется иметь файл формата \*.DAT и желательно иметь изображение, полученное с приёмника. Имена файлов должны быть идентичными. Файл \*.DAT должен быть перемещён в папке ./data, а файл с изображением формата \*.jpg должен быть перемещён в папку ./picture/mini\_tempest.

Дальше перечислены требования к файлу \*.DAT:

* Должен содержать шапку и информационную часть;
* Шапка формируется приемником, содержит параметры приёма, каждый отдельный элемент описания должен ограничиваться точкой с запятой (обязательно проверить перед добавлением), имеет структуру, изображенную на рисунке 3-1;
* Для корректного вывода в Поле 3 вкладки Tempest, необходимо иметь три значения в строке, находящейся после строки ‘Mode’, разделенные точкой с запятой, где первое – название величины, второе – значение величины, третье – единицы измерения (если единиц измерения нет, то оставлять пустое значение, обрамлённое точкой с запятой);заканчиваться массив значений должен значением ‘SWT’;
* Шапка должна заканчиваться строчкой: ‘Values;{i};’, где {i} – это количество пар значений, которые даны в файле, увеличенное на единицу;
* После строки, указанной в предыдущем пункте, идут строки с парами значений, где первое значение в паре – это время, а второе – амплитуда сигнала.

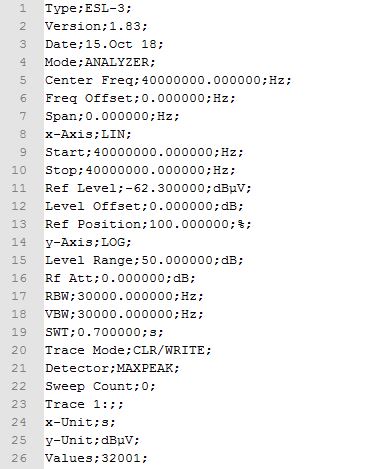


Рисунок 3‑1. Структура шапки в файлах \*.DAT

Требование к миниатюре \*.jpg:

* Должна содержать изображение сигнала;
* Размер изображения 195х190;
* Если нужного изображения нет, необходимо внести пустое изображение.

## Добавление новых обучающих выборок.

Также в программу добавлена возможность расширения обучающеq выборки «извне», т.е. не изменяя код программы. Для этого необходимо провести две процедуры. Первая процедура – это подготовка файлов обучающей выборки, а вторая – добавление файлов в директорию программы.

Для обучения нейронной сети необходимо иметь два массива данных: входной и целевой. Входной массив представляет собой реальный сигнал, из тех, что может быть обработан во вкладке Tempest. Целевой должен создаваться путём ручной обработки входного.

Алгоритм получения целевого сигнала:

1. Выбор такого реального сигнала, в котором визуально хорошо видно все информативные импульсы;
2. Установить границу разделения шумового сигнала и информативного;
3. Всё, что ниже полученной границы, приравнять к нулю.

Покажу на примере:

Пусть есть сигнал из файла 10\_OKT\_5.DAT, он представлен на рисунке 3-2:

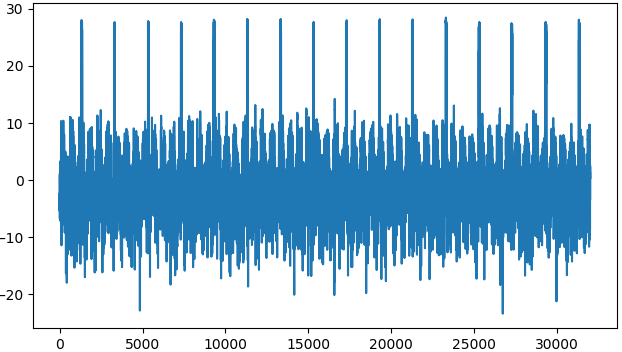


Рисунок 3‑2. График сигнала из файла 10\_OKT\_5.DAT.

Так как необходимо работать с типами данных языка Python, то необходимо получить этот сигнал в виде массива Python. Для этого, откроем консоль Python в корневой папке программы (это можно сделать следующим образом: при открытом проекте в PyCharm в нижней строке выбрать вкладку ), и введём следующий код:

from n\_tempest import tempest

t = tempest('data/10\_OKT\_5.DAT')

sgnl = t.signal

В итоге получаем массив sgnl, который необходимо обработать. По рисунку 3-2 видно, что чётко можно выделить границу в 15 dBμV. Поэтому можно слудеющим кодом приравнять всё, что ниже этой границы, к нулю:

trn = sgnl.copy()

trn[trn < 15] = 0

В итоге, у нас есть два массива sgnl и trn, которые получены из файла 10\_OKT\_5.DAT, и содержат входной и целевой сигналы для обучения нейронной сети. Дальше их надо должным образом сохранить. Для этого выполним следующий кусок кода, если примем, что в папке data\_for\_train максимальный индекс файлов sgnl\_?.npy и trn\_?.npy равен i:

import numpy as np  
np.save(f'data\_for\_train/sgnl\_{i+1}.npy', sgnl)  
np.save(f'data\_for\_train/trn\_{i+1}.npy', trn)

При этом необходимо также создать в папке data\_for\_train файл desc\_{i+1}.txt с кодировкой UTF-8, в котором написать название реального сигнала, из которого мы получили данные примеры.

Примечание: алгоритм, направленный на автоматизацию объединения обучающих выборок, принимает не больше 9 примеров. Удалять примеры необходимо с сохранением общей прогрессии индексов, иначе программа выдаст ошибку.

# Возможности по улучшению программы

Дальше перечислены доработки, которые в силу сроков и недостаточного уровня знаний первоначального автора были либо отложены, либо упущены из вида. При должном внимании к этим поправкам и их осуществлении, программа может стать более результативной.

## Улучшение оценки эффективности.

На данном этапе, оценка эффективности происходит посредством вычисления среднеквадратичного отклонения между обработанным сигналом и входным сигналом. Но это слишком слабая оценка эффективности. Необходимо разработать комплексную оценку, которая сможет более точно описывать модель.

Под комплексной оценкой понимается добавление визуальных, статистических и корреляционных методов оценки.

Визуальные методы хороши при оценке очистки сигнала от импульсных шумов, под ними подразумевается нахождение в автоматическом режиме пиков рассматриваемых сигналов, и их сравнение.

Статистический метод заключаются в рассмотрении параметров рассматриваемых сигналов на предмет их различия. Например: сравнения мод.

При должном, взвешенном объединении всех этих методов реально найти комплексный метод, который может быть полезен не только в данной работе, но и в других задачах, связанных с фильтрацией сигналов.

## Оптимизация работы с большими массивами.

Существующий метод создания и работы с массивами искусственных сигналов слишком громоздкий, что сильно нагружает память компьютера. Были предусмотрены различные средства для промежуточной оптимизации (сохранение синтезированных данных). Но необходимо добавление более стройных оптимизационных инструментов.

1. Разбиение большого массива не более мелкие и очередное или выборочное их использование. Это значительно сократит нагрузку на оперативную память.
2. Запись общих результатов и их экстраполяция. Изначально, метод создания большого количества синтезированных массивов был введён в качестве статистически более выгодного средства, но можно найти способ для экстраполяции общих данных, что также сократит нагрузку на оперативную память.

## Изменение структуры нейронной сети.

Это, на самом деле, самая важная проблема данной работы, так как качество обработки зависит напрямую от структуры нейронной сети. В качестве примера можно рассмотреть сложные структуры нейронных сетей, использующихся для машинного перевода, или обработки визуальных и звуковых сигналов. Рисунок 4-1 иллюстрирует некоторые разработанные модели.

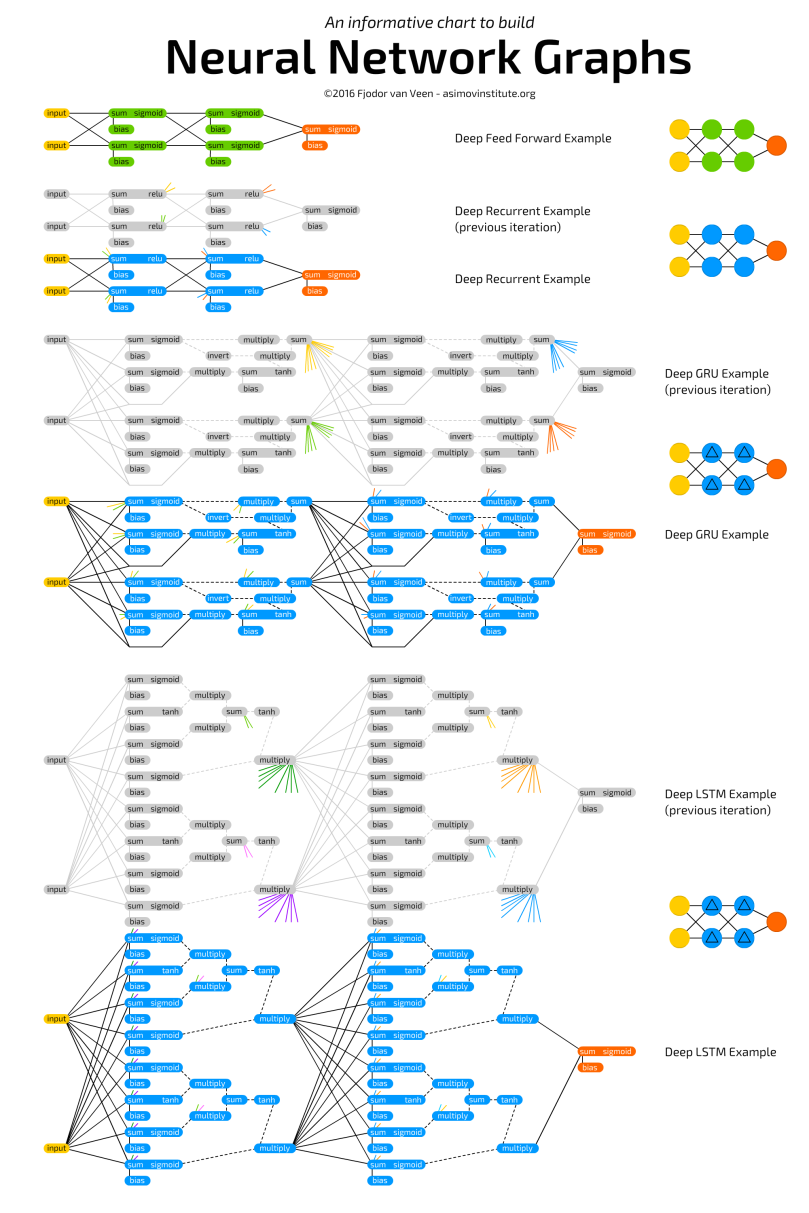


Рисунок 4‑1. Некоторые архитектуры нейронных сетей, использующихся для конкретных задач.

Как видно, сеть прямого распространения, использующаяся в данной работе, является простейшей, это напрямую не говорит о её слабости. Но стоит учесть, что электромагнитные сигналы и ПЭМИ обладают крайне сложной структурой, и обработка с помощью простейшей нейронной сети не сможет дать должных результатов. Стоит пересмотреть структуру, возможно с упором на уже разработанную архитектуру.

Но имеющаяся библиотека sk-learn не подходит для целей расширения, так как она заточена только на сети прямого распространения. Более мощным инструментом на момент написания этой работы является Keras/TensorFlow, позволяющий создавать гибкие структуры, и обладающий более мощным инструментарием для обучения и оптимизации.

## Изменение характера обучающей выборки.

В силу недостаточного уровня знаний автора этой работы, возможно, были допущены ошибки при подготовке обучающих выборок. Так как изначально сигнал, получаемый из файла \*.DAT приводится к среднему в районе нуля. А потом к этому нулю приводятся значения сигналов не принадлежащих к информативным. Это может искажать данные и вызывать ненормальные результаты. Поэтому необходимо пересмотреть данный аспект работы программы и привести метод подготовки целевого сигнала к подобающему виду.

Это же касается и создания искусственно синтезированных сигналов. Для них был выбран метод наполнения шумами обратный процессу ручной подготовки обучающей выборки. Ещё необходимо проработать расчёт ОСШ в дБ, так как в настоящей версии данный расчёт весьма условный.