Министерство науки и высшего образования Российской Федерации

Федеральное государственное автономное образовательное учреждение высшего образования

ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
СИСТЕМ УПРАВЛЕНИЯ И РАДИОЭЛЕКТРОНИКИ (ТУСУР)

Кафедра КСУП

Визуальное проектирование нейронных сетей в конструкторе вычислительного эксперимента СМ МАРС

ОТЧЕТ

ПО РЕЗУЛЬТАТАМ

Производственной (вид практики) практики: Преддипломной (тип практики.)

Обучающийся гр. 583-М

(подпись) Карабатов П.В (И.О. Фамилия)

17.05.25
(дата)

Руководитель практики от профильной организации:
Доцент кафедры КСУП, к.т.н.
(должность, ученая степень, звание)

Отлично (оценка) (подпись) Кочергин М.И (И.О. Фамилия)

М.П. 17.05.25
(дата)

Руководитель практики от Университета:
Профессор кафедры КСУП, д.т.н, профессор
(должность, ученая степень, звание)

(оценка) (подпись) Шурыгин Ю.А (И.О. Фамилия)

(дата)

Томск 2025

Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное учреждение высшего образования

ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
СИСТЕМ УПРАВЛЕНИЯ И РАДИОЭЛЕКТРОНИКИ (ТУСУР)

Кафедра КСУП

УТВЕРЖДАЮ
Зав.
кафедрой КСУП
Шурыгин Ю.А
(Ф.И.О.)

(подпись)

«___» _____ 2025 г.

ИНДИВИДУАЛЬНОЕ ЗАДАНИЕ

на Производственную практику: Преддипломную
(вид практики) (тип практики)

студенту гр.583-М факультета вычислительных систем
Карабатову П.В
(Ф.И.О студента)

1. Тема практики: Визуальное проектирование нейронных сетей в конструкторе вычислительного эксперимента СМ МАРС
2. Цель практики: Получение теоретических и практических результатов, достаточных для успешного выполнения и защиты выпускной квалификационной работы
3. Сроки прохождения практики:24.03.2025-17.05.2025

Совместный рабочий график (план) проведения практики

№ п/п	Перечень заданий	Сроки выполнения
1	Изучение обновленного конструктора	24.03.2025-05.04.2025
2	Проектирование модулей	07.04.2025-19.04.2025
3	Реализация модулей	21.04.2025-10.05.2025
4	Написание отчета и дневника по практике	12.05.2025-17.05.2025

Дата выдачи: «_24_» _Марта___ 2025 г.

Руководитель практики от университета

Профессор кафедры КСУП _____ Шурыгин Ю.А
(должность) (Подпись) (Ф.И.О.)

Согласовано:

Руководитель практики от профильной организации

Доцент кафедры КСУП _____ Кочергин М.И
(должность) (Подпись) (Ф.И.О.)

М.П.

Задание принял к исполнению «24» Марта 2025г.

Студент гр. 583-М _____ Карабатов П.В
(Подпись) (Ф.И.О.)

Оглавление

Введение

Использование нейронных сетей позволяет автоматизировать множество рутинных процессов, часто связанных с распознаванием или прогнозированием, что в свою очередь позволяет использовать освободившееся время с большей пользой. Однако для каждого отдельного случая необходимо создание и обучение отдельной нейронной сети, поскольку эффективность машинного обучения напрямую зависит от выбранной модели и составленной обучающей выборки.

Проектирование нейронных сетей представляет собой непростой итерационный процесс выбора архитектуры сети, количества скрытых слоёв, нейронов в этих слоях, функций активации нейронов, методов оптимизации для обучения сети, метрик оценки качества и пр. Поэтому актуальным является разработка средств автоматизации проектирования нейронных сетей. Создание таких инструментов на базе систем компьютерного моделирования, позволит не только создавать и тренировать нейронные сети без необходимости программирования, но и интегрировать их в модели систем управления объектами, чьи модели построены в этих средах моделирования. Часто подобные программы работают по принципу визуального проектирования, поскольку слоённая организация современных нейронных сетей прекрасно перекладывается на подобную систему.

Среда моделирования MAPC предоставляет возможности визуального моделирования объектов и их систем управления, поэтому включение возможности разработки и использования нейронных сетей в СМ MAPC открывает множество возможностей по расширению сфер применения среды моделирования.

Целью данной работы является исследование возможностей интеграции методов машинного обучения в визуальный конструктор вычислительного эксперимента СМ MAPC.

В рамках текущей работы предполагается создание основного инструментария по работе с данными перед их использованием в рамках работы с нейронными сетями и моноблоки нейронной сети прямого распространения, рекуррентной нейронной сети и сверточной нейронной сети.

Для реализации данной задачи необходимо:

1) Изучить визуальный конструктор вычислительного эксперимента СМ МАРС;
2) Реализовать инструментарий данных;
3) Реализовать моноблоки нейронных сетей.

1 Анализ предметной области

1.1 Нейронные сети и машинное обучение

Нейронные сети [2] – это метод машинного обучения, при котором компьютерная программа работает по принципу человеческого мозга, используя различные нейронные связи. Если очень сильно упрощать, это человеческий мозг в миниатюре, только нейроны в нем искусственные и представляют собой вычислительные элементы, созданные по образу и подобию биологических нейронов.

Нейрон – это вычислительная единица, которая получает информацию, производит над ней простые вычисления и передает ее дальше. Они делятся на три основных типа: входной, скрытый и выходной. Также есть нейрон смещения и контекстный нейрон. В том случае, когда нейросеть состоит из большого количества нейронов, вводят термин слоя. Соответственно, есть входной слой, который получает информацию, n скрытых слоев (обычно их не больше 3), которые ее обрабатывают и выходной слой, который выводит результат.

У каждого из нейронов есть 2 основных параметра: входные данные (input data) и выходные данные (output data).

Синапс – это связь между двумя нейронами. У синапсов есть 1 параметр – вес. Благодаря ему, входная информация изменяется, когда передается от одного нейрона к другому. Допустим, есть 3 нейрона, которые передают информацию следующему. Тогда у нас есть 3 веса, соответствующие каждому из этих нейронов. У того нейрона, у которого вес будет больше, та информация и будет доминирующей в следующем нейроне (пример – смешение цветов).

Нейросеть является обучаемой системой и может быть самообучаемой. Она может обучаться как с помощью заданных человеком алгоритмов распознавания или команд, так и на основе прошлого опыта – то есть самостоятельно, используя ранее полученные данные.

Нейронные сети используются для решения сложных задач, которые требуют аналитических вычислений подобных тем, что делает человеческий мозг. Самыми распространенными задачами, успешно решаемыми с применением нейронных сетей, являются:

1) Классификация – распределение объектов по классам. Например, на вход дается список пассажиров Титаника и нужно построить модель, предсказывающую спасётся пассажир или нет. Эту работу может сделать нейронная сеть, анализируя такую информацию как: возраст, класс, количество членов семьи на борту и т.д;

2) Предсказание – возможность предсказывать будущие значения показателя на основе прошлых. Например, рост или падение акций, основываясь на ситуации на фондовом рынке;

3) Распознавание – в настоящее время, самое широкое применение нейронных сетей. Используются для нахождения необходимого образа среди вводных данных.

Для каждой цели требуется свой подход. На сегодняшний день существует множество архитектур и подходов, которые максимизируют производительность нейронных сетей под конкретную задачу.

Наиболее распространённые типы нейронных сетей:

1) Перцептрон – это самый простой вид нейронной сети, который был разработан в 1957 году Фрэнком Розенблаттом. Он состоит из одного или нескольких нейронов, связанных между собой. Перцептрон применяется для решения задач бинарной классификации и может использоваться для создания логических операций;

2) Сверточные нейронные сети были разработаны для анализа и обработки изображений. Они используют сверточные слои для автоматического извлечения признаков из входных данных. CNN обычно применяются для задач распознавания образов, классификации изображений, детекции объектов и даже анализа видео;

3) Рекуррентные нейронные сети предназначены для работы с последовательными данными, такими как тексты, речь и временные ряды. Они имеют обратные связи между нейронами, что позволяет учитывать контекст и зависимости в данных. Однако классические RNN имеют проблему затухающего или взрывающегося градиента. Для решения этой проблемы были разработаны модификации, такие как LSTM (долгая краткосрочная память) и GRU (единицы с воротами);

4) LSTM – это вид рекуррентных нейронных сетей, специально разработанный для работы с долгосрочными зависимостями в данных. Они позволяют моделям учиться на длительных последовательностях и успешно применяются в задачах генерации текста, машинного перевода и анализа временных рядов;

5) Сети с архитектурой внимания позволяют моделировать взаимосвязи между элементами входных данных, уделяя особое внимание определенным частям. Они успешно применяются в машинном переводе, генерации текста и других задачах, где важно учесть контекст и взаимосвязи между элементами данных.

1.2 Среда моделирования МАРС

Среда моделирования МАРС (**М**оделирование и **А**втоматический **Р**асчёт **С**истем) [3] – универсальная программная система компьютерного моделирования, которая позволяет частично или полностью заменить физический эксперимент вычислительным, исследовать и оптимизировать характеристики создаваемых устройств или подсистем в поисках наилучшего варианта.

Основанная на предложенном профессором Томского государственного университета систем управления и радиоэлектроники (ТУСУР) Дмитриевым Вячеславом Михайловичем методе компонентных цепей, система обладает широким набором режимов анализа, а адаптация к моделированию нового класса устройств осуществляется оперативно путем использования и расширения библиотеки моделей компонентов. При этом имеется возможность автоматизированной генерации новых моделей компонентов, что обеспечивает лёгкую и быструю расширяемость областей применения данной системы.

К возможностям среды можно отнести следующие:

1) Графический набор схемы;
2) многооконный пользовательский интерфейс;
3) просмотр и обработка результатов;
4) режим анализа цепи по постоянному току;
5) режим анализа цепи во временной области;
6) режим анализа цепи в частотной области;
7) моделирование как линейных, так и нелинейных цепей;
8) печать схемы и выходной информации, такой как графики, таблицы и т.д.;

9) возможность изменения параметров (номиналов) компонентов во время моделирования;

10) обширный набор компонентов от аналоговой до цифровой электротехники и электроники.

Интерфейс программы СМ МАРС представлен на рисунке 1.1.

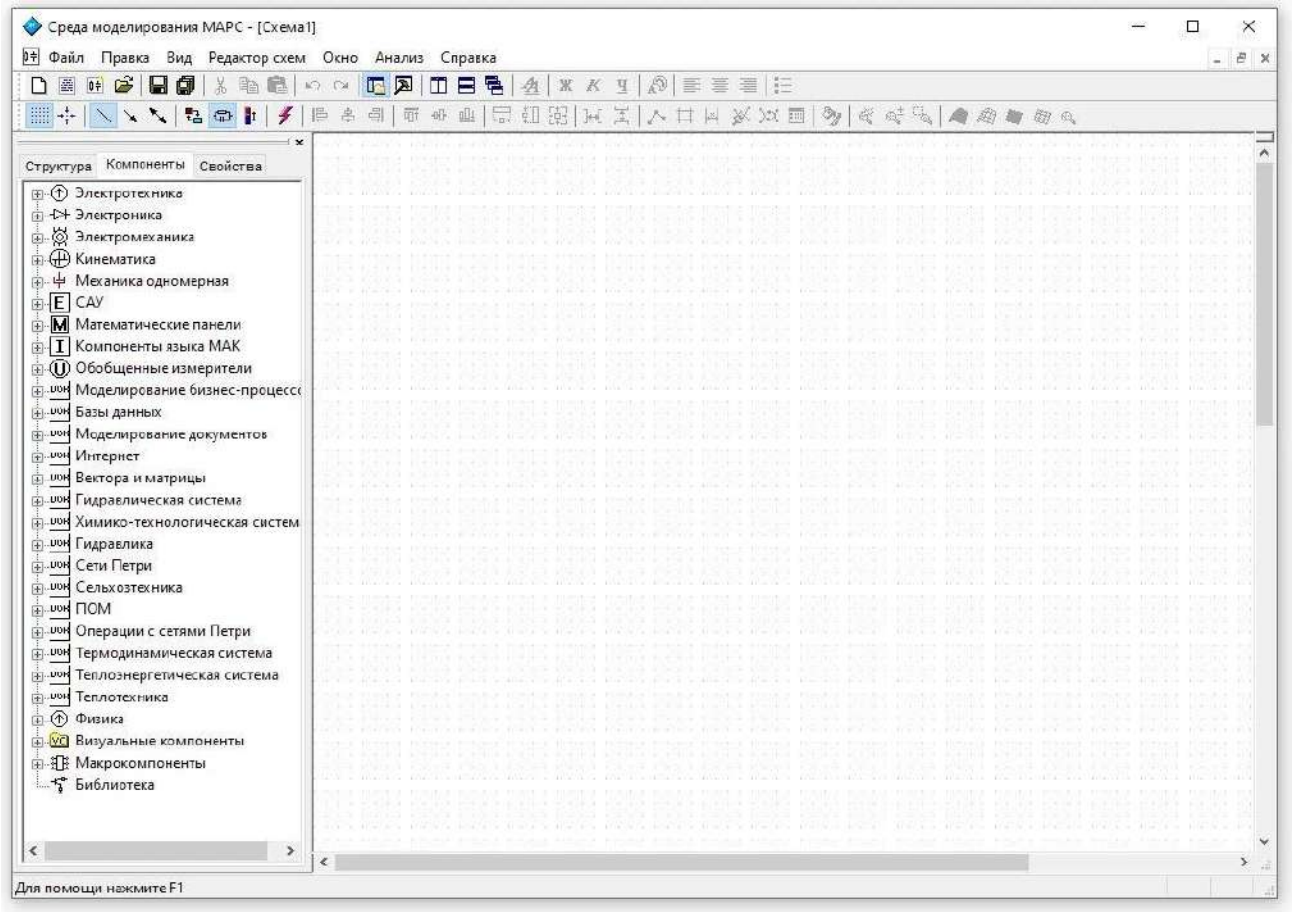

Рисунок 1.1 – Основное окно СМ МАРС

Пример визуальных компонентов СМ МАРС представлен на рисунке 1.2.

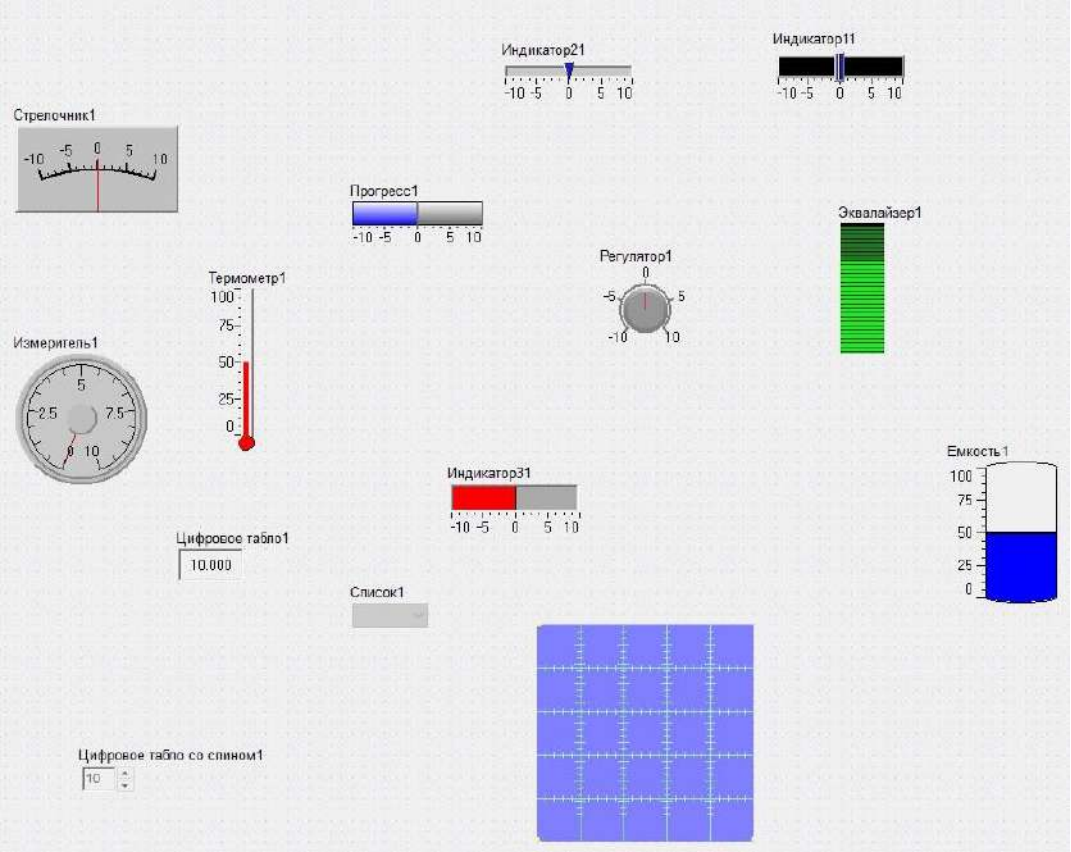

Рисунок 1.2 – Пример визуальных компонентов СМ МАРС

На базе среды моделирования МАРС реализован ряд виртуальных лабораторий. Универсальный аппарат моделирования среды МАРС позволяет создавать виртуальные учебно-исследовательские лаборатории практически по любой технической дисциплине. Лаборатории оснащены генераторами различных сигналов, полным набором измерительных приборов, включая осциллографы и анализаторы спектров, и обладают широкими возможностями в части обработки результатов вычислительного эксперимента.

Сферы применения среды моделирования МАРС:

1) Полупроводниковые приборы (более 10 различных полупроводниковых приборов);

2) Источники воздействий (открытый для пополнения набор источников, задаваемых математическим выражением);

3) Вольтамперные характеристики;

4) Моделирование электрических машин;

5) Моделирование электромагнитных элементов с разветвленной магнитной системой;

6) Электронное и функциональное моделирование систем автоматического управления;

7) Моделирование звеньев нелинейных систем автоматического управления;

8) Цифровые устройства;

9) Структурное моделирование электропривода.

2 Обзор Аналогов

На рынке программ по созданию нейронных сетей существует огромное множество готовых решений от крупных информационных компаний. Благодаря разделению нейронных сетей на слои большая их часть использует блочную структуру, однако это не является единственно возможной архитектурой.

2.1 Deep Learning Toolbox

Deep Learning Toolbox [4] – сборник дополнений для среды математического моделирования Matlab, который обеспечивает основу для разработки и реализации глубоких нейронных сетей с алгоритмами, предварительно обученными моделями и приложениями.

Интерфейс программы представлен на рисунке 2.1.

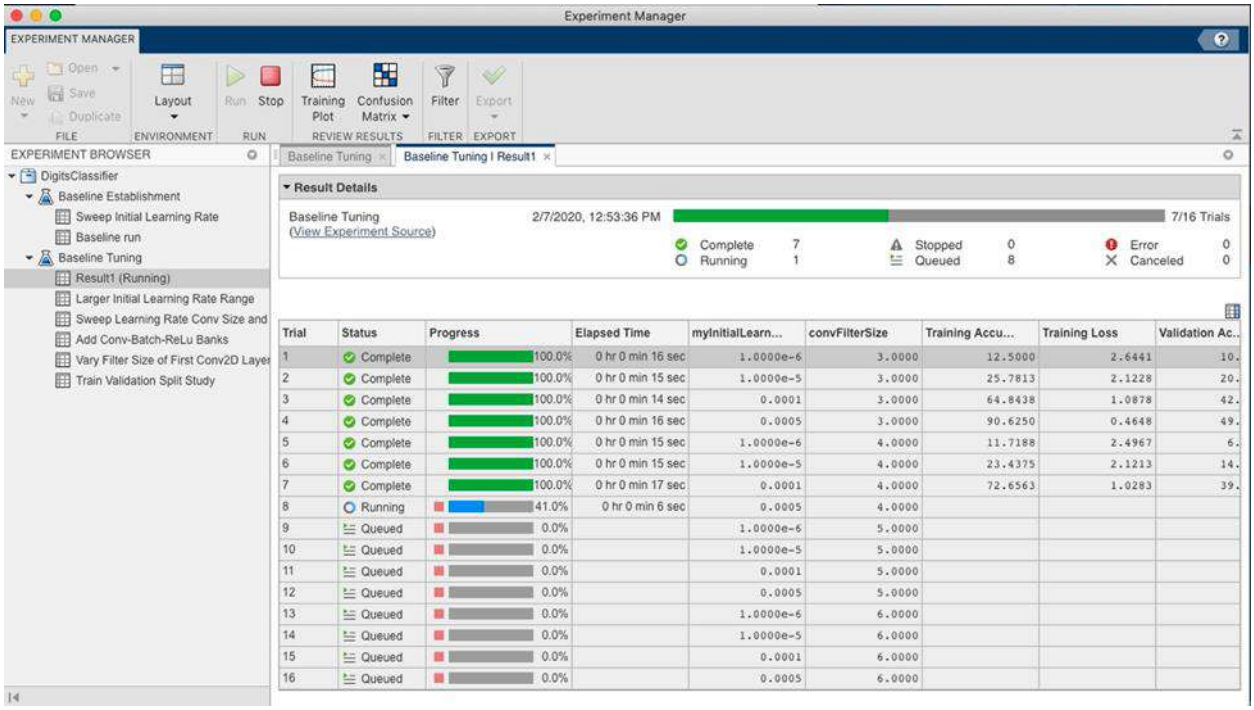

Рисунок 2.1 – Интерфейс Deep Learning Toolbox

2.1.1 Deep Network Designer

Элемент Deep Network Designer [5] сборника Deep Learning Toolbox позволяет графически создавать, анализировать и тренировать модели нейронных сетей.

Интерфейс создания нейронных сетей в Deep Network Designer разделён на две части: Layer Library, библиотека слоёв из которых мы выбираем нужные нам и переносим в поле Designer, где на их базе уже создаётся нужная нам нейронная сеть.

Интерфейс программы представлен на рисунках 2.2, 2.3 и 2.4.

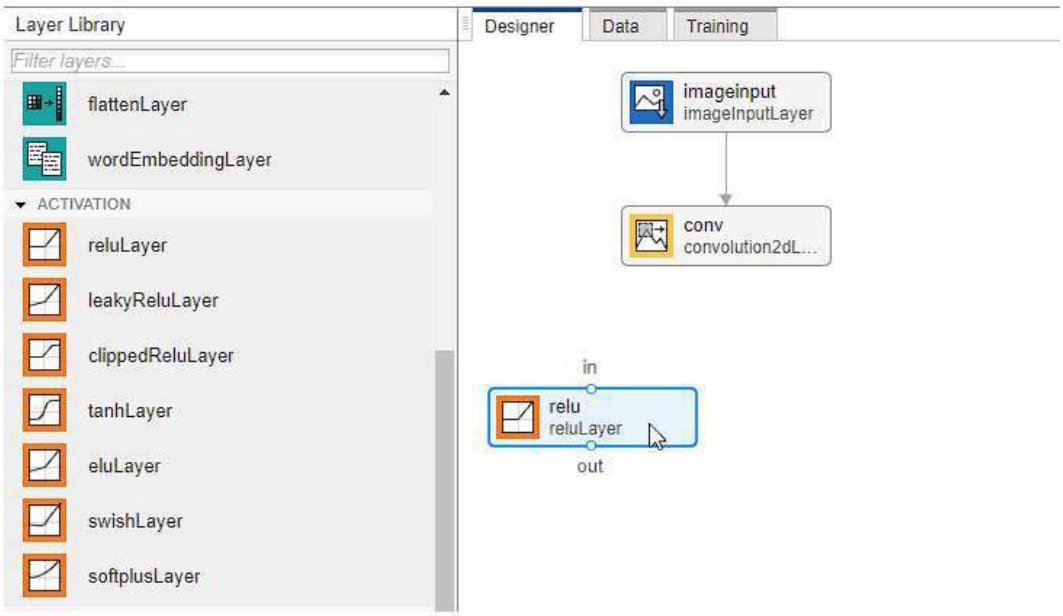

Рисунок 2.2 – Интерфейс Deep Learning Designer

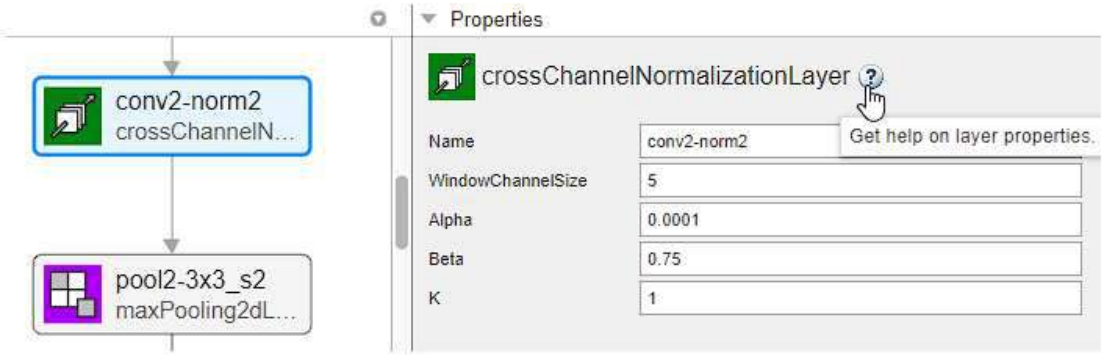

Рисунок 2.3 – Интерфейс настройки слоёв

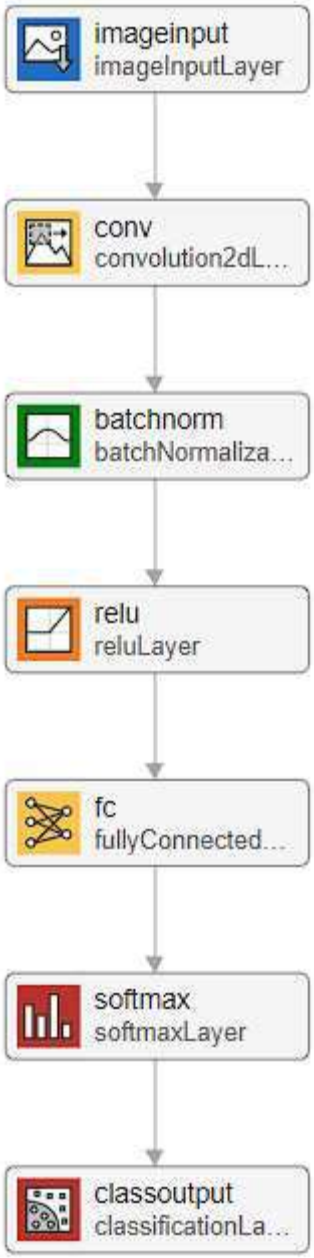

Рисунок 2.4 – Сверточная нейронная сеть

Кроме того, возможен вариант задания нейронной сети через код Matlab. Реализация сверточной нейронной сети аналогичной созданной при помощи модулей слоёв представлена на листинге 2.1.

Листинг 2.1 – Код для создания сверточной нейронной сети

```
inputSize = [28 28 1];
numClasses = 10;
layers = [
imageInputLayer (inputSize)
convolution2dLayer(5,20)
batchNormalizationLayer
reluLayer
```

```
fullyConnectedLayer(numClasses)
softmaxLayer
classificationLayer];
deepNetworkDesigner (layers)
```

Из всего вышеперечисленного можно сделать вывод, что создание большинства видов нейронных сетей возможно в компоненте Deep Network Designer и многих других компонентах сборки Deep Learning Toolbox.

2.1.2 Neural Fitting App

Приложение Neural Net Fitting [6] позволяет создавать, визуализировать и обучать двухслойную сеть с прямолинейным движением для решения задач подгонки данных.

Это условие мешает нам создавать рекуррентные и сверточные нейронные сети с использованием подгонки данных, что оставляет возможным лишь использованием нейронных сетей прямого распространения.

Интерфейс программы представлен на рисунке 2.5.

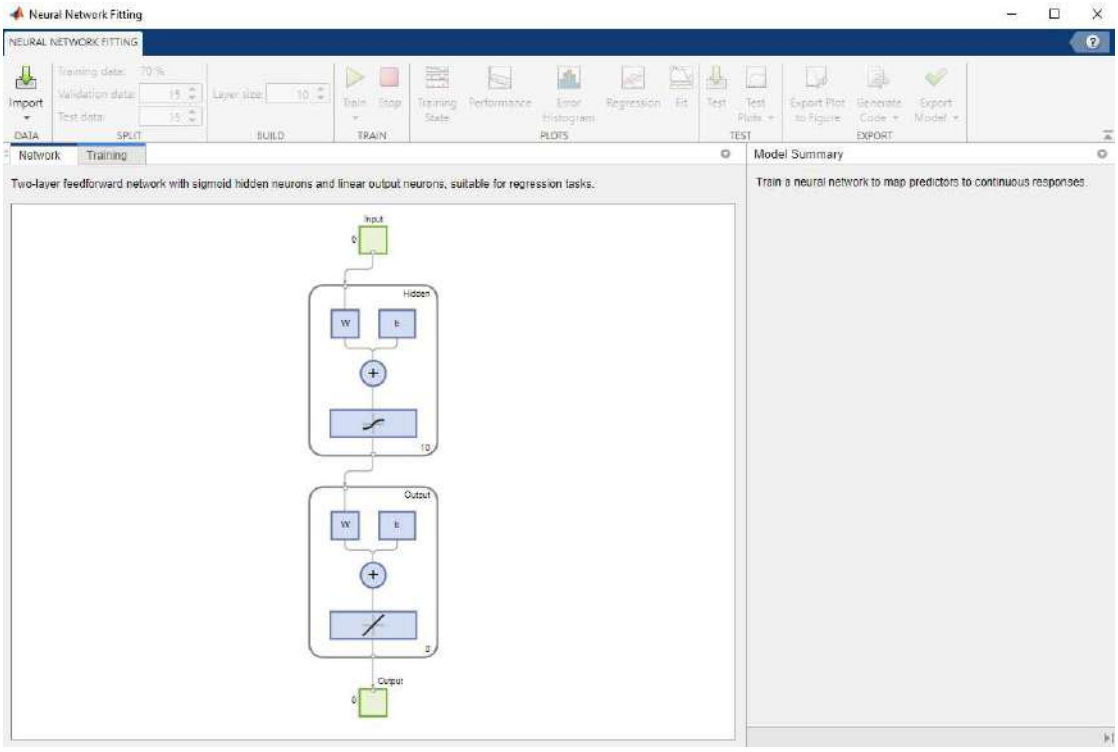

Рисунок 2.5 – Интерфейс программы Neural Fitting App

2.2 NeuroGenetic Optimizer

NeuroGenetic Optimizer [6] – это программа, созданная компанией BioComp Systems, которая автоматизирует проектирование и обучение нейронных сетей, осуществляя перебор комбинаций входных переменных, архитектур нейронных моделей, а также их внутренних структур для создания групп полностью обученных высокоэффективных надежных моделей, которые прогнозируют то, что требуется. По заверениям разработчиков эта программа специализирована на моделировании временных рядов.

В процессе поиска NeuroGenetic Optimizer разбивает данные на 3 массива данных;

1) Моделирование: используется для выполнения регрессии;
2) Оптимизация: используется для оценки альтернативных моделей и определения того, когда следует прекратить обучение модели;
3) Отбор: для выбора моделей, которые следует сохранить и представить в "ансамбле" или наборе.

Эти наборы данных извлекаются последовательно, случайным образом или случайно с проверкой на сходство, чтобы обеспечить соответствующую выборку для построения модели. Использование регрессии с "ранней остановкой" и двойной валидацией, а также поиск согласованности по всем трем наборам данных позволяет создавать модели, обладающие повышенной надежностью.

Интерфейс программы представлен на рисунке 2.6.

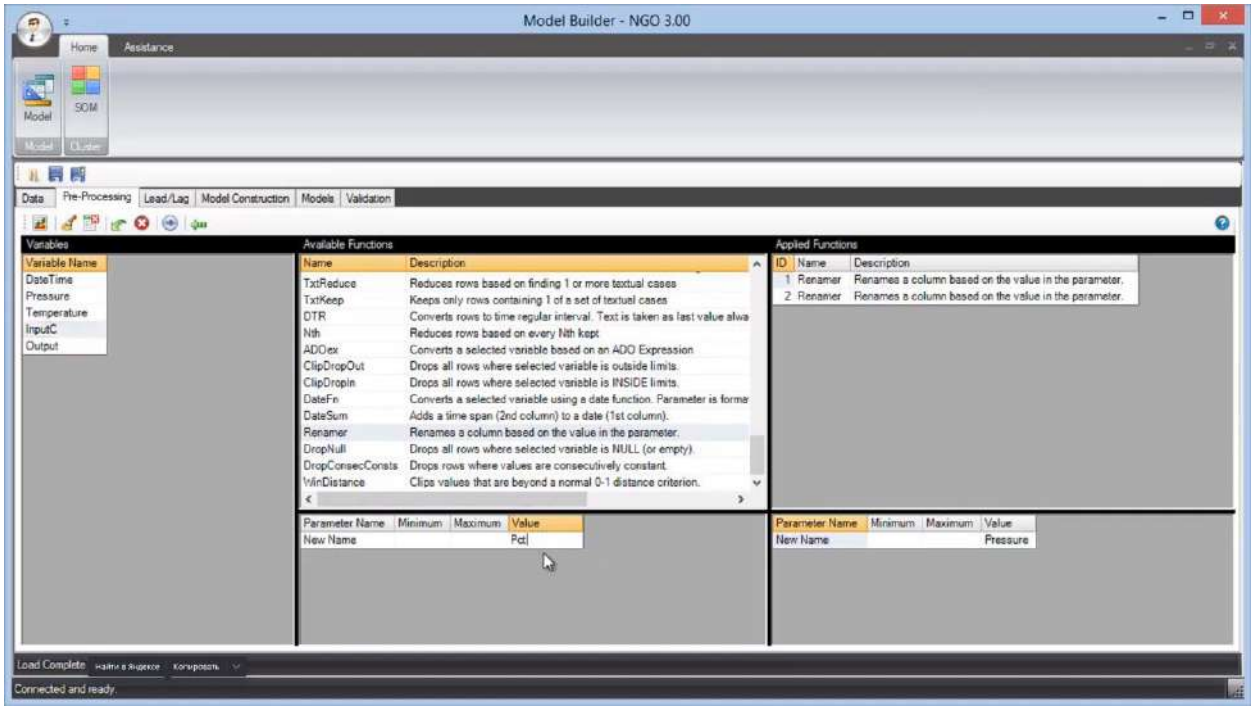

Рисунок 2.6 – Интерфейс программы Neurogenetic Optimizer

2.3 SimInTech

SimInTech(от Simulation in Tech) [8] **-** среда разработки математических моделей, алгоритмов управления, интерфейсов управления и автоматической генерации кода для контроллеров управления и графических дисплеев .

Программа SimInTech предназначена для детального исследования и анализа нестационарных процессов в ядерных и тепловых энергоустановках, в системах автоматического управления, в следящих приводах и роботах, и в любых технических системах, описание динамики которых может быть представлено в виде системы дифференциально-алгебраических уравнений и/или реализовано методами структурного моделирования. Основными направлениями использования SimInTech являются создание моделей, проектирование алгоритмов управления, их отладка на модели объекта, генерация исходного кода на языке Си для программируемых контроллеров.

Создание нейронных сетей в среде SimInTech работает по принципу модульного строительства. Пользователь выбирает доступные модули и

соединяет их при помощи графического интерфейса без взаимодействия с кодом самих модулей.

На данный момент в SimInTech доступны следующие модули по построению нейронных сетей:

1) Входной слой: Компонент представляет собой входной слой нейронной сети (НС). Этот компонент создает НС и производит первичную обработку входных данных;

2) Полносвязный слой: Компонент представляет собой полносвязный слой нейронной сети (НС);

3) Сверточный слой: Компонент представляет собой сверточный слой нейронной сети (НС);

4) Антисверточный слой: Компонент представляет собой антисверточный слой нейронной сети (НС);

5) Слой объединения: Компонент представляет собой субдискретизирующий слой нейронной сети (НС);

6) Слой сцепления: Компонент представляет собой слой нейронной сети (НС), сцепляющий несколько веток;

7) Слой обрезки: Компонент представляет собой слой обрезки нейронной сети (НС);

8) Слой активации: Компонент представляет собой слой с функцией активации нейронной сети (НС);

9) Слой изменения размера: Компонент представляет собой слой нейронной сети (НС), служащий для изменения размера данных;

10) Слой суммирования: Компонент представляет собой слой суммирования нейронной сети (НС);

11) Выходной слой: Компонент представляет собой выходной слой нейронной сети (НС). Он содержит в себе последний полносвязный слой и функцию потерь. Для корректной работы нейронной сети необходимо подключить к нему блок НС - Режим работы;

12) Режим работы: Данный блок задает режим работы нейронной сети (НС). Для корректной работы схему нейронной сети всегда необходимо подключать этом блоку. В зависимости от выбранного режима работы у блока появляются соответствующие порты и свойства;

13) Исходные данные: Компонент представляет собой набор базовых тестовых примеров, которые необходимы как для проведения проверок работоспособности нейронной сети (НС), так и для обучения персонала работе с НС. Компонент работает совместно с файлами наборов данных (такими как база данных MNIST). Компонент предоставляет «чистые» данные и метки, если имеются для данного набора;

14) MNIST to PX: Данный блок преобразует данные MNIST в RGB.

В среде SimInTech возможно создание сверточных нейронных сетей благодаря модулю сверточный слой и рекуррентных нейронных сетей благодаря модулю полносвязный слой.

Примеры создания нейронной сети в среде SimInTech представлены на рисунках 2.7 и 2.8.

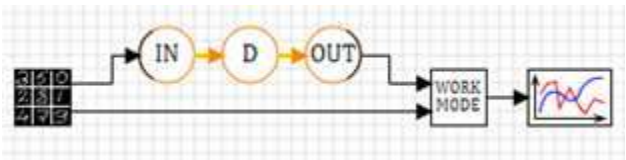

Рисунок 2.7 – Создание нейронной сети прямого распространения

Рисунок 2.8 – Создание сверточной нейронной сети

Сопоставительный анализ функционала приложений с функционалом создания машинного обучения представлен в таблице 2.1.

Таблица 2.1 – Функции приложений по проектированию нейронных сетей

		Персептроны	Сверточные нейросети	Рекуррентные нейросети	Необходимые программы	Шаблонные нейронные сети
Deep Learning Toolbox	Deep Network Designer	Да	Да	Да	Matlab	Да
	Neural Fitting App	Да	Нет	Нет	Matlab	
NeuroGenetic Optimizer		Да	Нет	Да	Нет	Нет
SimInTech		Да	Нет	Да	Нет	Нет

3 Синтез портрета проектируемой системы

3.1 Формулирование требований к проектируемой системе

Исходя из проведенного обзора, можно сказать что наиболее полным функционалом обладает программа Deep Learning Toolbox, что предоставляет доступ к наиболее обширному арсеналу заготовленных слоёв и функций, а также выверенному синтаксису среды Matlab и полноценной документацией всех функций.

SimInTech предоставляет доступ к меньшему числу встроенных функций, однако обладает более интуитивно понятным интерфейсом и не требуют предустановки каких-либо программ.

Neural Fitting App и NeuroGenetic Optimizer предоставляют чрезвычайно полезный, но в тоже время крайне ситуативный функционал, который может пригодиться в процессе обучения нейронных сетей, но построение моделей на их основе если не невозможно, то крайне осложнено программными ограничениями.

Можно сделать вывод, что при проектировании среды визуального проектирования нейронных сетей на базе среды моделирования МАРС необходимо включить в её состав все базовые функции Deep Network Designer, а именно:

1) Блочную структуру построения нейронных сетей;

2) Возможность создания персептронов, рекуррентных и сверточных нейросетей;

3) Возможность тонкой настройки слоёв и синапсов;

4) Набор шаблонных нейронных сетей для решения тривиальных задач;

5) Полноценную документацию, проясняющую работу каждого отдельного модуля.

3.2 Обзор средств реализации

Для целей создания среды визуального моделирования нейронных сетей на базе среды моделирования МАРС актуальным является обзор существующих библиотек машинного обучения для их интеграции в разрабатываемое приложение. Ввиду наличия опыта интеграции кода СМ МАРС и программ на языке Python [9] обзор библиотек будет производиться для данного языка.

Библиотеки машинного обучения предоставляют средства для проектирования нейросетей следующих архитектур:

1) Сети прямого распространения (многослойные персептроны, внутри которых информация поэтапно проходит от входного слоя до выходного), применяемые для решения задач классификации, регрессии и пр.;

2) Нейронные сети свёрточного типа, применяемые для задач обработки и анализа изображений (их особенность –использование свёрточных слоёв для поэтапного извлечения заданных признаков из входного изображения);

3) Нейронные сети рекуррентного типа, применяемые для задач, связанных с последовательными данными, такими как текст (их особенность – возможность обратного прохода сигналов между слоями и наличие внутренней памяти).

Кратко охарактеризуем анализируемые библиотеки.

TensorFlow [10] предоставляет реализации нейросетей на языках: Python, C++, C#, R, Java и др. Работа TensorFlow основана на составления графа вычисления, который вычисляет «тензоры» – многомерные массивы, через которые могут быть представлены вектора признаков, изображение или массив изображений. Поскольку TensorFlow даёт лишь базовые компоненты для создания нейронных сетей, то для создания даже самой базовой сети прямого распространения необходимо полностью прописать её структуру и поведение на каждом из уровней.

Theano [11] – библиотека численного вычисления с открытым исходным кодом, созданная группой машинного обучения Монреальского Университета. По аналогии с TensorFlow Theano может быть использована в качестве ядра для моделей нейронных сетей, поэтому для создания нейронных сетей на базе Theano необходимо ручное создание всех необходимых конструктов, что во много раз усложняет создание даже самых простых сетей, при этом позволяет точно задать все необходимые параметры.

Keras [12] – библиотека с открытым исходным кодом высокого уровня, построенная на основе TensorFlow, но предлагающая более компактный и упрощенный синтаксис вкупе с возможностью простого создания рекуррентных и сверточных нейронных сетей путём использования готовых модулей. Модели нейронных сетей в Keras состоят из отдельных шаблонных слоёв, которые по очереди добавляются в объект типа модель.

Создание сверточной нейронной сети при помощи Keras представлено на листинге 3.1.

Листинг 3.1 – Модель сверточной нейронной сети в Keras

```
model = models.Sequential()
model.add(layers.Flatten())
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(10))
model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(32, 32,
3)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
```

```
model.add(layers.Conv2D(64,        (3,        3),        activation='relu'))
```

Создание рекуррентной нейронной сети при помощи Keras представлено на листинге 3.2.

Листинг 3.2 – Модель рекуррентной нейронной сети в Keras

```
model = keras.Sequential()
model.add(layers.Embedding(input_dim=1000, output_dim=64))
model.add(layers.GRU(256, return_sequences=True))
model.add(layers.SimpleRNN(128))
model.add(layers.Dense(10))
model.summary()
```

PyTorch [13] – фреймворк машинного обучения с открытым исходным кодом, созданный на базе библиотеки Torch языка Lua. Фреймворк адаптирован для использования графических ускорителей одновременно с процессором в процессе тензорных вычислений, что во много раз увеличивает производительность. Кроме того, с помощью фреймворка возможно быстрое создание глубоких нейронных сетей. Модели нейронных сетей в PyTorch строятся на основе построения пользовательских классов с заданными слоями и порядком действий при переходе между слоями.

Создание сверточной нейронной сети при помощи PyTorch представлено на листинге 4.3.

Листинг 3.3 – Модель сверточной нейронной сети в PyTorch

```
class ConvNet(nn.Module):
     def __init__(self):
         super(ConvNet, self).__init__()
        self.layer1 = nn.Sequential( nn.Conv2d(1, 32, kernel_size=5, stride=1,
padding=2),
            nn.ReLU(), nn.MaxPool2d(kernel_size=2, stride=2))
             self.layer2  =  nn.Sequential(  nn.Conv2d(32,  64,  kernel_size=5,
stride=1, padding=2),
            nn.ReLU(), nn.MaxPool2d(kernel_size=2, stride=2))
         self.drop_out = nn.Dropout()
         self.fc1 = nn.Linear(7 * 7 * 64, 1000)
         self.fc2 = nn.Linear(1000, 10)
    def forward(self, x):
     out = self.layer1(x)
     out = self.layer2(out)
     out = out.reshape(out.size(0), -1)
     out = self.drop_out(out)
     out = self.fc1(out)
     out = self.fc2(out)
     return out
```

Создание рекуррентной нейронной сети при помощи PyTorch представлено на листинге 4.4.

Листинг 3.4 – Модель рекуррентной нейронной сети в PyTorch

```
class MyRNN(nn.Module):
    def __init__(self, input_size, hidden_size, output_size):
        super(MyRNN, self).__init__()
        self.hidden_size = hidden_size
        self.in2hidden = nn.Linear(input_size + hidden_size, hidden_size)
        self.in2output = nn.Linear(input_size + hidden_size, output_size)

    def forward(self, x, hidden_state):
        combined = torch.cat((x, hidden_state), 1)
        hidden = torch.sigmoid(self.in2hidden(combined))
        output = self.in2output(combined)
        return output, hidden

    def init_hidden(self):
        return nn.init.kaiming_uniform_(torch.empty(1, self.hidden_size))
```

PyBrain [14] – модульная библиотека машинного обучения. Модульность системы и встроенные методы обучения позволяют снизить порог вхождения при разработке с использованием данной библиотеки. Создание модели нейронной сети в PyBrain похоже на Keras так как слои задаются при помощи команд обьекта типа модели, но в отличии от Keras PyBrain требует ручного ввода связей между слоями. Библиотека PyBrain не поддерживает сверточные нейронные сети, однако полностью поддерживает рекуррентные сети.

Создание рекуррентной нейронной сети при помощи PyBrain представлено на листинге 3.5.

Листинг 3.5 – Модель рекуррентной нейронной сети в PyBrain

```
n2 = RecurrentNetwork(name='net2')
n2.addInputModule (LinearLayer (2, name='in'))
n2.addModule (SigmoidLayer (3, name='h'))
n2 addOutputModule (LinearLayer(1, name='out'))
n2.addConnection (FullConnection (n2 ['in'], n2['h'], name='c1'))
n2.addConnection (FullConnection (n2 ['h'], n2 ['out'], name='c2')) n2.sortModules ()
```

Сопоставительный анализ функционала различных библиотек машинного обучения представлен в таблице 3.1.

Таблица 3.1 – Функции библиотек машинного обучения

	Персептроны	Сверточные нейросети	Рекуррентные нейросети	Необходимые библиотеки	Поддержка GPU	Тип Лицензии
TensorFlow	Да	Пользовательские	Пользовательские	NumPy	Да	Apache 2.0
Theano	Да	Пользовательские	Пользовательские	NumPy	Да	BSD
Keras	Да	Да	Да	NumPy, TensorFlow	Да (многопоточная)	MIT Licence
PyTorch	Да	Да	Да	Нет	Да (многопоточная)	BSD
PyBrain	Да	Нет	Да	SciPy	Нет	BSD

Из собранных данных можно сделать вывод, что каждая библиотека обладает своими преимуществами и недостатками, в основном связанными с быстродействием и возможностью развёртывания определенных видов нейронных сетей. Поэтому выбор фреймворка для работы в первую очередь должен быть основан на требованиях проекта.

Из разобранных библиотек связка TensorFlow и Keras позволяет использовать легко создавать большую часть стандартных видов нейронных сетей. PyTorch обладает несколько более громоздким синтаксисом и требует больше внимательности при составлении модели нейронной сети, но позволяет проводить обучение нейронной сети во много раз быстрее благодаря поддержке многопоточных вычислений. Модульность PyBrain делает использование библиотеки выигрышным в условиях, когда необходимо создать нестандартную нейронную сеть, однако невозможность создание сверточных нейронных сетей сильно ограничивают эффективность библиотеки. Использование Theano на практике ограниченно из-за отсутствия каких-либо стандартных функций по работе с нейронными сетями.

Для разработки проекта наиболее подходящей библиотекой является PyTorch. Благодаря оптимизации для многопоточных вычислений PyTorch будет являться предпочтительным вариантом при наличии достаточно мощного аппаратного обеспечения, а возможность установки пользовательских связей между слоями делает эту библиотеку более гибкой для разработки для разработки, при этом более требовательной к математическим способностям пользователя. Keras предлагает более простой синтаксис и меньше возможностей по модификации стандартных элементов, что сильно упрощает разработку сетей где нет необходимости создания сложных пользовательских алгоритмов, однако для реализации настроек соединения слоёв-блоков нейронной сети такая организация будет менее удобной из-за ограниченных возможностей по настройки связи слоёв внутри создаваемой нейронной сети.

3.3 Архитектура системы

На данный момент можно синтезировать два варианта архитектуры средств по созданию нейронных сетей, которые позволят выполнить поставленные выше требования:

1) Построение нейронной сети в виде моноблока на логическом слое, где параметры нейронной сети задаются при помощи параметров блока;
2) Построение нейронной сети в более традиционном представлении множества соединённых компонентов, представляющие различные слои нейронной сети.

Плюсом первого варианта является простота использования для конечного пользователя: ему понадобиться лишь добавить нужный компонент на логический слой, а также возможность изменения параметров нейронной сети во время выполнения каких-либо действий. Минусом такого варианта является строгая типизация - если создать блок основываясь на механизме прямого распространения, то для использования другого типа нейронной сети придётся создавать совершенно другой блок.

Основным плюсом второго варианта является полная свобода действий со стороны пользователя: благодаря многоблочной архитектуре он волен создавать любую систему, что будет ему необходима, а также позволяет более гибкий инструментарий по оптимизации нейронной сети. Минусом подобной организации является более высокий порог вхождения: конечному пользователю необходимо иметь более глубокое представление о принципах работы и компоновки нейронных сетей.

3.3.1 Одноблочная система

Для реализации одноблочной системы необходимо разместить единственный блок на логическом слое, на начало которого необходимо послать входные данные в виде тензора, в параметрах которого необходимо задать параметры нейронной сети и на выходе мы должны получить результат путём выполнения обучения.

На вход In подаются входные данные, выход Out выдаёт полученные данные.

Схема блока нейронной сети представлена на рисунке 3.1.

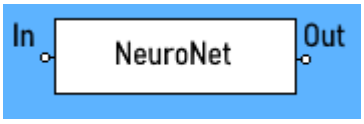

Рисунок 3.1 – Схема блока «Нейронная Сеть»

Параметры блока «Нейронная сеть»:

1) Поле «Количество слоёв» - целочисленное число, определяющее количество скрытых слоёв нейронной сети;

2) Поле «Количество нейронов» - целочисленное число, определяющее количество нейронов на каждом скрытом слое нейронной сети;

3) Поле «Функция активация» - перечисление значений, определяющее тип функции активации для скрытых слоёв нейронной сети.

3.3.2 Многоблочная система

Для реализации многоблочной системы необходимо разместить несколько блоков-модулей на логическом слое, представляющих собой все необходимые компоненты для построения и работы нейронной сети.

Блок разделения на выборки: необходим для разбиения входных данных на обучающую и тестовые выборки. На вход In подаются входные данные, на выходе Out_T мы получаем тестовую выборку, на выходе Out_O – обучающую выборку.

Схема блока разделения на выборки представлена на рисунке 3.2.

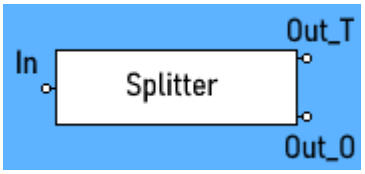

Рисунок 3.2 – Схема блока «Разделение на выборки»

Параметры блока «Разделение на выборки»:

1) Поле «Доля обучающей выборки» - дробное число в диапазоне [0;1], определяющее долю обучающей выборки;

2) Поле «Константа воспроизведения» - целочисленное число, определяющее сид генератора случайных значений, используемого для разделения исходных данных.

Блок обучения: необходим для обучения нейронной сети на основе выборки данных. На вход In_O подаётся обучающая выборка, на вход In_T – тестовая выборка, на вход M – сигнал, управляющий режимом обучения, на вход In_R – реальный отклик нейронной сети. С выхода Out_R выдаётся ошибка, с выхода Out_W – рассчитанные весы нейронной сети.

Схема блока обучения представлена на рисунке 3.3.

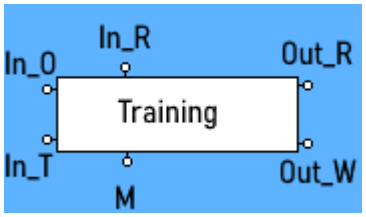

Рисунок 3.3 – Схема блока «Обучение»

Параметры блока «Обучение»:

1) Поле «Метрика точности» - перечисление значений, определяющие используемую метрику оценки ошибок;

2) Поле «Метод оптимизации» - перечисление значений, определяющие используемый метод оптимизации;

3) Поле «Количество эпох обучения» - целочисленное число, определяющие количество эпох обучения;

4) Поле «Размер пакета данных» - целочисленное число, определяющие размер пакета данных;

5) Поле «Коэффициент скорости обучения» - дробное число в диапазоне [0;1], определяющее коэффициент скорости обучения;

Входной слой: на вход In подаются входные данные, на вход Upd подаются обновленные в процессе обучения коэффициенты, выход Out передаёт их на скрытие слои.

Схема блока входного слоя представлена на рисунке 3.4.

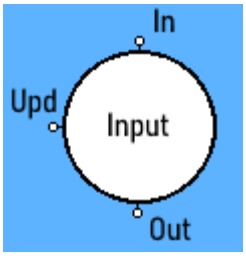

Рисунок 3.4 – Схема блока «Входной слой»

Параметры блока «Входной слой»

1) Поле «Количество нейронов» - целочисленное число, определяющее количество нейронов во входном слое;

2) Поле «Функция активация» - перечисление значений,

определяющее тип функции активации слоя.

Скрытые слои: основные строительные блоки нейронной сети, в которых происходит преобразование входных данных. Существует огромное множество возможных вариантов скрытых слоёв для самых различных вариантов нейронных сетей. В рамках, составленных требования нам необходимо реализовать слои для работы сверточных, реккурентных и сетей прямого распространения. Для реализации сверточной нейронной сети нам нужны слои свертки и слои подвыборки, для реализации реккурентых сетей нужен слой с реализацией долгой краткосрочной памяти, а для реализации сетей прямого распространения необходим плотносвязанный слой.

Несмотря на разницу в работе блоков скрытых слоёв, их представление на логическом слое будет функционально идентичным и главным их отличием друг от друга будут параметры.

Схема блока скрытого слоя представлена на рисунке 3.5.

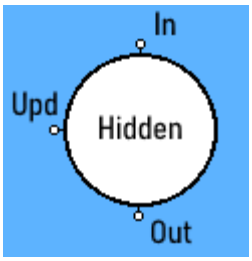

Рисунок 3.5 – Схема блока «Скрытый слой»

На вход In подаются входные данные, на вход Upd подаются обновленные в процессе обучения коэффициенты, выход Out выдаёт полученные данные.

Для каждого вида скрытых слоёв задаётся поле «Количество нейронов» - целочисленное число, определяющее количество нейронов в скрытом слое;

Каждый вид блока скрытых слоёв также обладает своими собственными параметрами, зависящие от предназначения блока.

Параметры блока «Полносвязный слой»:

1) Поле «Количество входных записей» - целочисленное значение, определяющие количество входных записей.

2) Поле «Количество выходных откликов» - целочисленное значение, определяющие количество выходных записей.

Параметры блока «Слой свертки»:

1) Поле «Количество входных каналов» - целочисленное значение, определяющие количество входных каналов;

2) Поле «Количество выходных каналов» - целочисленное значение, определяющие количество выходных каналов;

3) Поле «Шаг свертки» - целочисленное значение, определяющие шаг свертки;

4) Поле «Размер сверточного ядра» - целочисленное значение, определяющие размер сверточного ядра;

5) Поле «Размер подбивки» - целочисленное значение, определяющие размер подбивки;

6) Поле «Расстояние между элементов ядра» - целочисленное значение, определяющие расстояние между элементов ядра;

7) Поле «Количество заблокированных соединений» - целочисленное значение, определяющие количество заблокированных соединение между входными и выходными каналами;

8) Поле «Обучаемое смещение» - логическое значение, определяющие наличие обучаемого смещения на слое;

9) Поле «Функция активация»: перечисление значений, определяющее тип функции активации слоя.

Параметры блока «Слой подвыборки»:

1) Поле «Шаг свертки» - целочисленное значение, определяющие шаг свертки;

2) Поле «Размер сверточного ядра» - целочисленное значение, определяющие размер сверточного ядра;

3) Поле «Подбивка» - логическое значение, определяющие наличие подбивки боковых значений отрицательными бесконечностями в случае

несоответствия размеров;

4) Поле «Ширина шага свертки» - целочисленное значение, определяющие ширину шага свертки;

5) Поле «Возращение индексов» - логическое значение, определяющие возвращаются ли максимальные индексы вместе с выходными данными;

6) Поле «Режим Ceil» - логическое значение, определяющие используется ли режим Ceil для вычисления размера выходной формы. Если значение верно, используется режим Ceil, если неверно – режим Floor.

Параметры блока «Слой долгой краткосрочной памяти»:

1) Поле «Ожидаемое количество входных данных» - целочисленное значение, определяющие ожидаемое количество признаков входных данных;

2) Поле «Количество признаков в скрытом состоянии» - целочисленное значение, определяющие количество признаков в скрытом состоянии;

3) Поле «Количество рекуррентных слоёв» - целочисленное значение, определяющие количество рекуррентных слоёв;

4) Поле «Использование весов» - логическое значение, определяющие использование весов b_ih и b_h. Если значение верно, то весы используются;

5) Поле «Формат данных» - логическое значение, определяющие формат данных. Если значение верно, то формат данных представляется в виде (batch, seq, feature), если неверно, то в формате (seq, batch, feature);

6) Поле «Слой выпадения» - целочисленное значение, определяющие наличие слоя выпадения и шанса на выпадение. Если значение равно 0, то слой выпадения отсутствует, при любом другом значении слой выпадения присутствует и шанс на выпадение равен

значению поля;

7) Поле «Размер проекции» - целочисленное значение, определяющие размер проекции;

8) Поле «Двунаправленный слой» - логическое значение, определяющие является ли свои двунаправленным или нет.

Выходной слой: этот блок передаёт полученный результат работы нейронной сети далее по схеме.

На вход In подаются входные данные, выход Out выдаёт полученные данные.

Схема блока выходного слоя представлена на рисунке 3.6.

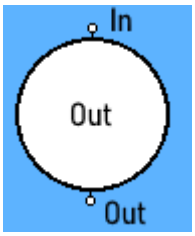

Рисунок 3.6 – Схема блока «Выходной слой»

Параметры блока «Выходной слой»

1) Поле «Количество нейронов» - целочисленное число, определяющее количество нейронов во выходном слое;
2) Поле «Функция активация» - перечисление значений, определяющее тип функции активации слоя.

4 Работа с нейронными сетями в PyTorch

Создание нейронных сетей в PyTorch основывается на базовом классе nn.Module. В состав класса входит 2 функции: конструктор __init__ и функция forward, которая определяет вычисления, выполняемые при каждом вызове. nn.Module также может содержать в себе другие элементы типа nn.Module, благодаря чему PyTorch позволяет создавать многослойные нейронные сети.

Структура нейронной сети определяется путём добавления модулей слоёв в конструктор __init__ через конструкцию self.*название модуля*. На

листинге 4.1 представлено определение структуры стандартной сверточной нейронной сети состоящей из двух сверточных слоёв в функции __init__.

Листинг 4.1 – Определение структуры нейронной сети

```
def __init__(self) -> None:
    super().__init__()
    self.conv1 = nn.Conv2d(1, 20, 5)
    self.conv2 = nn.Conv2d(20, 20, 5)
```

Определение функции прохода forward для той же нейронной сети можно увидеть на листинге 4.2.

Листинг 4.2 – Определение функции прохода нейронной сети

```
def forward(self, x):
x = F.relu(self.conv1(x)) return F.relu(self.conv2(x))
```

Подобная организация позволяет создавать на её основе гибкие многофункциональные системы и узконаправленные сети, кроме того, такая структура отлично масштабируется благодаря вложенности.

4.1 Работа с нейронными сетями прямого распространения

Для создания нейронных сетей прямого распространения в фреймворке PyTorch используется модуль nn.Linear и его ответвление nn.Bilinear.

Модуль nn.Linear представляет собой слой, который применяет линейное преобразование к входным данным. Он называется линейным преобразованием, потому что применяет линейное уравнение $y = xA^T + b$ к передаваемым данным.

Модуль nn.Linear ожидает, что входные данные будут представлены в виде одномерного, двухмерного или трехмерного тензора, последняя размерность которого - размер входных данных, а остальные размерности - размер партии и другие размерности.

Параметры модуля:

1) in_features – размер входной выборки;
2) out_features – размер выходной выборки;
3) bias – если False, то слой не использует весы смещения b_ih или b_hh. Значение по умолчанию: True.

Модуль nn.Bilinear представляет собой ответвление от модуля nn.Linear которое применяет билинейное преобразование $y = x_1^T A x_2 + b$ к входящим данным.

Параметры модуля:

1) in1_features – размер первой входной выборки;
2) in2_features – размер второй входной выборки;
3) out_features – размер выходной выборки;
4) bias – если False, то слой не использует весы смещения b_ih или b_hh. Значение по умолчанию: True.

4.2 Работа с рекуррентными нейронными сетями

Для создания рекуррентных нейронных сетей в фреймворке PyTorch существуют три модуля, представляющих различные имплементации данной концепции: nn.RNN, nn.LSTM и nn.GRU.

Модуль nn.RNN представляет собой модель нейронной сети Элмана. Нейронная сеть Элмана представляет собой рекуррентную нейронную сеть, полученную в результате добавления обратных связей в многослойный персептрон.

Схема сети Элмана можно увидеть на рисунке 4.1.

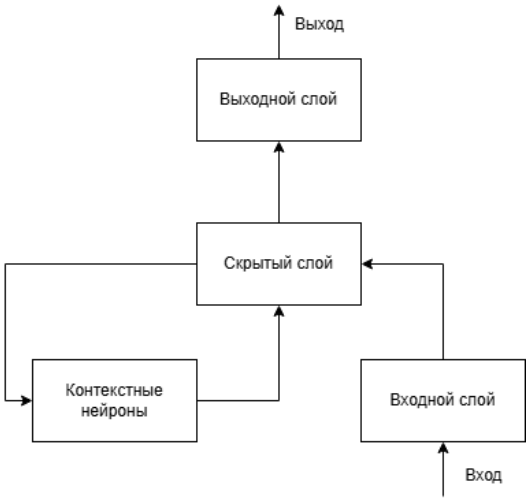

Рисунок 4.1 – Схема рекуррентной нейронной сети Элмана

Сеть Элмана представляет собой самый простой вариант создания рекуррентной нейронной сети из представленных в PyTorch и обладает рядом достоинств, в первую очередь высокой скоростью, простотой тренировки и низкими требованиями к мощности. Однако сети Элмана намного хуже справляются с большими объемами данных.

Параметры модуля:

1) input_size – количество ожидаемых признаков во входных данных;
2) hidden_size – количество признаков в скрытом состоянии;
3) num_layers – количество рекуррентных слоёв. Например, num_layers=2 означает, что модуль будет представлять собой соединение 2 моделей Элмана, вторая из которых при этом будет принимать вывод данных из первой на свой входной слой. Значение по умолчанию: 1;
4) nonlinearity – используемая модель нелинейности. Может принимать значения 'tanh' или 'relu'. Значение по умолчанию: 'tanh';
5) bias – если False, то слой не использует весы смещения b_ih или b_hh. Значение по умолчанию: True;
6) batch_first – Если True, то входные и выходные тензоры будут представлены как (batch, seq, feature) вместо (seq, batch, feature). При этом, это не относится к скрытым и ячейковым состояниям. Значение по умолчанию: False;
7) dropout – если значение отличается от нуля, то добавляет слой Отключения/Dropout на выход каждого слоя RNN кроме последнего. Шанс на исключения равен заданному значению. Значение по умолчанию: 0;
8) bidirectional – если True, то модель становится двунаправленной RNN. Значение по умолчанию: False.

Данные на вход:

1) Input – тензор выбранного параметром batch_first формата или формата (seq,input_size) для непакетных данных;

2) h_0 – тензор, содержащий начальное скрытое состояние для входной последовательности. Значение по умолчанию: нули. Формат тензора: (D*num_layers,hidden_size) или (D*num_layers,batch,hidden_size), где D = 1 или 2, в зависимости от того, является ли сеть двунаправленной.

Данные на выход:

1) output – тензор выбранного параметром batch_first формата или формата (seq,hidden_size) для непакетных данных

2) h_n – тензор, содержащий финальное скрытое состояние. Значение по умолчанию: нули. Формат тензора: (D*num_layers,hidden_size) или (D*num_layers,batch,hidden_size), где D = 1 или 2, в зависимости от того, является ли сеть двунаправленной.

Модуль nn.LSTM представляет собой модель Долгосрочной Короткой Памяти (англ. Long Short Term Memory, LSTM). Особенностью LSTM является возможность запоминания значений на короткие и длинные промежутки путём использования механизмов фильтров/врат, которые позволяют «доставать» из памяти необходимые данные в нужный момент. Схема Долгосрочной Короткой Памяти представлена на рисунке 4.2.

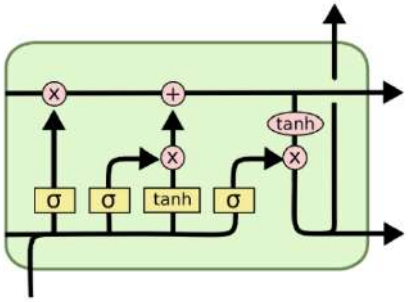

Рисунок 4.2 – Схема Долгосрочной Короткой Памяти

LSTM-сети лучше всего подходят для решения задач с большим объемом данных и необходимостью наиболее точных результатов, однако из-за механизма фильтров они также являются более ресурсозатратными.

Параметры модуля:

1) input_size – количество ожидаемых признаков во входных данных;
2) hidden_size – количество признаков в скрытом состоянии;
3) num_layers – количество рекуррентных слоёв. Например, num_layers=2 означает, что модуль будет представлять собой соединение 2 LSTM-модулей, второй из которых при этом будет принимать вывод данных из первого на свой входной слой. Значение по умолчанию: 1;
4) bias – если False, то слой не использует весы смещения b_ih или b_hh. Значение по умолчанию: True;
5) batch_first – Если True, то входные и выходные тензоры будут представлены как (batch, seq, feature) вместо (seq, batch, feature). При этом, это не относится к скрытым и ячейковым состояниям. Значение по умолчанию: False;
6) dropout – если значение отличается от нуля, то добавляет слой Отключения/Dropout на выход каждого слоя RNN кроме последнего. Шанс на исключения равен заданному значению. Значение по умолчанию: 0;
7) bidirectional – если True, то модель становится двунаправленной RNN. Значение по умолчанию: False;
8) proj_size – если > 0, будет использоваться LSTM с проекциями соответствующего размера. Должен быть меньше чем hidden_size. Значение по умолчанию: 0.

Данные на вход:

1) Input – тензор выбранного параметром batch_first формата или формата (seq,input_size) для непакетных данных;

2) h_0 – тензор, содержащий начальное скрытое состояние для каждого элемента входной последовательности. Формат тензора: (D*num_layers,hidden_size) или (D*num_layers,batch,hidden_size), где D = 1 или 2, в зависимости от того, является ли сеть двунаправленной. hidden_size заменяется на proj_size если его значение >0. Значение по умолчанию: нули;

3) c_0 – тензор, содержащий начальное состояние ячейки для каждого элемента входной последовательности. Формат тензора: (D*num_layers,hidden_size) или (D*num_layers,batch,hidden_size), где D = 1 или 2, в зависимости от того, является ли сеть двунаправленной. Значение по умолчанию: нули;

Данные на выход:

1) output – тензор формата (seq, batch, D*hidden_size)/(batch,seq,D*hidden_size) или формата (seq,D* hidden_size) для непакетных данных. hidden_size заменяется на proj_size если его значение >0.

2) h_n – тензор, содержащий финальное скрытое состояние. Значение по умолчанию: нули. Формат тензора: (D*num_layers,hidden_size) или (D*num_layers,batch,hidden_size), где D = 1 или 2, в зависимости от того, является ли сеть двунаправленной.

3) c_n – тензор, содержащий финальное состояние ячейки для каждого элемента входной последовательности. Формат тензора: (D*num_layers,hidden_size) или (D*num_layers,batch,hidden_size), где D = 1 или 2, в зависимости от того, является ли сеть двунаправленной. Если bidirectional = True, c_n будет содержать объединение финальных состояний ячеек прямого и обратного направления, соответственно.;

Модуль nn.RNN представляет собой модель Управляемого Рекуррентного Блока (англ. Gated Recurrent Unit, GRU). GRU представляет

собой упрощенную модель сети LSTM, где в отличии от неё нет постоянной ячейки памяти. Схема Долгосрочной Короткой Памяти представлена на рисунке 4.3.

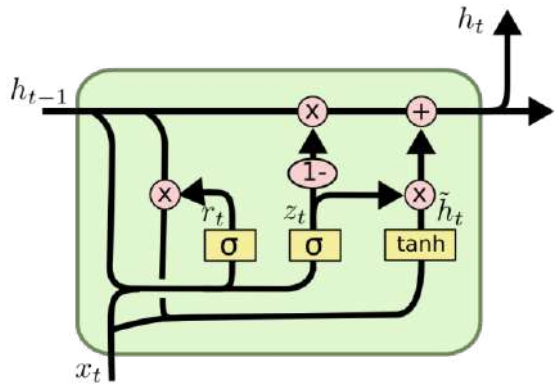

Рисунок 4.3 – Схема Управляемого Рекуррентного Блока

По сравнению с LSTM GRU имеет меньшее количество параметров и меньшую ресурсозатратность из-за отсутствия выходного фильтра и состояния ячеек, что позволяет использовать GRU там, где LSTM является избыточной, предоставляя при этом сравнимые по точности результаты. Однако из-за отсутствия постоянной ячейки памяти GRU проигрывает LSTM при работе с большими объемами данных.

Параметры модуля:

1) input_size – количество ожидаемых признаков во входных данных;

2) hidden_size – количество признаков в скрытом состоянии;

3) num_layers – количество рекуррентных слоёв. Например, num_layers=2 означает, что модуль будет представлять собой соединение 2 моделей Элмана, вторая из которых при этом будет принимать вывод данных из первой на свой входной слой. Значение по умолчанию: 1;

4) bias – если False, то слой не использует весы смещения b_ih или b_hh. Значение по умолчанию: True;

5) batch_first – Если True, то входные и выходные тензоры будут представлены как (batch, seq, feature) вместо (seq, batch, feature). При этом,

это не относится к скрытым и ячейковым состояниям. Значение по умолчанию: False;

6) dropout – если значение отличается от нуля, то добавляет слой Отключения/Dropout на выход каждого слоя RNN кроме последнего. Шанс на исключения равен заданному значению. Значение по умолчанию: 0;

7) bidirectional – если True, то модель становится двунаправленной RNN. Значение по умолчанию: False.

Данные на вход:

1) Input – тензор выбранного параметром batch_first формата или формата (seq,input_size) для непакетных данных;

2) h_0 – тензор, содержащий начальное скрытое состояние для входной последовательности. Значение по умолчанию: нули. Формат тензора: (D*num_layers,hidden_size) или (D*num_layers,batch,hidden_size), где D = 1 или 2, в зависимости от того, является ли сеть двунаправленной.

Данные на выход:

1) output – тензор формата (seq, batch, D*hidden_size)/(batch,seq,D*hidden_size)или формата (seq,D*hidden_size) для непакетных данных

2) h_n – тензор, содержащий финальное скрытое состояние. Значение по умолчанию: нули. Формат тензора: (D*num_layers,hidden_size) или (D*num_layers,batch,hidden_size), где D = 1 или 2, в зависимости от того, является ли сеть двунаправленной.

В результате исследования была составлена таблица 4.1, внутри которой приведено сравнение различных характеристик модулей, используемых для построения рекуррентных нейронных сетей. Таблица 4.1 представлена ниже.

Таблица 4.1 – Сравнение модулей рекуррентных нейронных сетей

Свойство	RNN	LSTM	GRU
Структура	Три связанных слоя	Система из трёх фильтров и ячеек памяти	Система из двух фильтров
Время тренировки	Быстрое, но неточное	Медленнее из-за большего количества фильтров и операций с памятью.	Быстрее, чем LSTM, но медленнее, чем RNN, из-за системы фильтров.
Требования к памяти	Низкие требования к памяти.	Высокое потребление памяти из-за сложной архитектуры	Меньшее потребление памяти по сравнению с LSTM, но большее по сравнению с RNN
Работа с длинными последовательностями	Проблемы с поддержанием долгосрочных зависимостей из-за затухания градиентов.	Превосходно справляется с отображением долгосрочных зависимостей благодаря ячейкам памяти.	Эффективнее, чем RNN, но менее эффективен, по сравнению с LSTM, для долгосрочных зависимостей.
Сложность тренировки	Склонность к затуханию градиентам, что затрудняет обучение на длинных последовательностях.	Легче тренировать длинные последовательности благодаря ячейкам памяти	Проще, чем LSTM, и легче в обучении, чем RNN, сохраняя при этом эффективность

Основываясь на полученных данных, можно сказать, что для выбора нужного блока для создания рекуррентной нейронной сети нужно отталкиваться от целей применения и доступных мощностей. Если в приоритете точность или предполагается использование длинных последовательностей, то LSTM является наиболее предпочтительным вариантом, однако она также является наиболее ресурсоёмкой. Если ресурсы ограничены, GRU предоставляет схожую с LSTM точность, при значительно меньшем потреблением памяти и вычислительных мощностей. На данный

момент RNN предпочтительны к использованию если в приоритете скорость и к использованию предполагаются короткие последовательности.

В целях проверки теории было решено измерить производительность различных модулей путём создания трёх нейронных сетей, идентичных по своему составу кроме модуля рекуррентного слоя, и использовать их для решения задачи прогнозирования временных рядов и экстраполяции данных цен на бирже.

В качестве датасета для прогнозирования использовались данные о стоимости акций AAPL, доступные через обращение к библиотеке YFinance.

В качестве метрик для оценки качества были взяты: скорость тренировки, средняя абсолютная ошибка (MAE), коэффициент детерминации R^2. Также субъективно оценивалась сложность составления сети. Для каждой задачи было проведено 10 измерений и в итоге были записаны их средние значения. Сравнение скорости тренировки, средней абсолютной ошибки и производительности различных модулей представлены в таблицах 4.2–4.6.

Таблица 4.2 – Скорость тренировки нейронных сетей с различным модулем рекуррентного слоя

№	Скорость обучения LSTM, с	Скорость обучения RNN, с	Скорость обучения GRU, с
1	343,7605	397,376	399,2546
2	411,7926	390,0983	335,5638
3	378,8344	350,1509	341,0333
4	407,7142	392,0773	391,9251
5	404,607	353,9301	397,6111
6	397,5382	404,3469	390,0731
7	403,9959	284,7281	396,8586
8	398,1452	319,8039	374,2337
9	404,1109	308,0997	335,1703
10	406,1499	353,8356	357,3788
Среднее значение	395,66488	355,44468	371,91024

Как видно из полученных данных, LSTM медленнее RNN и GRU в скорости тренировки на 11% и 6% соответственно.

Таблица 4.3 – Сравнение точности нейронных сетей с различным модулем рекуррентного слоя при прогнозировании временных рядов

№	R^2 LSTM	R^2 RNN	R^2 GRU
1	0,9956	0,9962	0,9901
2	0,9953	0,996	0,9913
3	0,9969	0,9968	0,9941
4	0,9964	-0,1743	0,9953
5	0,9975	0,9582	0,9863
6	0,9971	0,9944	-0,0175
7	0,9963	0,9967	0,9832
8	0,9975	0,9973	0,9763
9	0,9977	0,9914	0,9932
10	0,9973	0,9451	0,9921
Среднее значение	0,99676	0,86978	0,8884

Таблица 4.4 – Сравнение точности нейронных сетей с различным модулем рекуррентного слоя при решении задачи экстраполяции

№	R^2 LSTM	R^2 RNN	R^2 GRU
1	0,3064	0,1896	0,2286
2	0,1777	0,4687	0,0415
3	0,3714	0,4061	0,1777
4	0,1844	0,0132	0,4388
5	0,3322	-0,2869	0,0682
6	0,3725	0,4859	0,444
7	0,204	0,5584	0,4282
8	0,5652	-0,0404	0,4295
9	0,458	0,0322	0,1672
10	0,2734	0,0584	0,291
Среднее значение	0,32452	0,18852	0,27147

В плане точности экстраполяции составленные нейронные сети не проявляют особых результатов, однако сеть на базе LSTM вновь оказывается самой точной.

Таблица 4.5 – Сравнение средней абсолютной ошибка нейронных сетей с различным модулем рекуррентного слоя при прогнозировании временных рядов

№	MAE LSTM	MAE RNN	MAE GRU
1	0,014	0,0114	0,0185
2	0,0139	0,013	0,0247
3	0,0112	0,0112	0,00134
4	0,0118	0,541	0,0759
5	0,0095	0,0483	0,06732
6	0,0107	0,016	0,0804
7	0,0129	0,0122	0,0978
8	0,0099	0,0998	0,113
9	0,0093	0,1025	0,1256
10	0,01	0,1385	0,142
Среднее значение	0,01132	0,09939	0,074656

Таблица 4.6 – Сравнение средней абсолютной ошибка нейронных сетей с различным модулем рекуррентного слоя при решении задачи экстраполяции

№	MAE LSTM	MAE RNN	MAE GRU
1	0,0257	0,0315	0,0268
2	0,0349	0,0222	0,3782
3	0,0229	0,0272	0,2375
4	0,028	0,475	0,228
5	0,0254	0,077	0,1535
6	0,0237	0,0212	0,0695
7	0,0291	0,0213	0,2178
8	0,0204	0,228	0,1446
9	0,0218	0,2338	0,3638
10	0,026	0,367	0,1586
Среднее значение	0,02579	0,15042	0,19783

Сравнение сложности составления сети представлена в таблице 4.7.

Таблица 4.7 – Сравнение сложности составления и работы с различными модулями рекуррентных слоёв

Свойства	LSTM	RNN	GRU
Передача скрытого состояния на выход	Да	Да	Да
Передача состояния ячейки на выход	Да	Нет	Нет
Количество параметров	8	8	7
Возможность установки количества слоёв	Да	Да	Да
Возможность использования в Sequential	Нет	Нет	Нет

С точки зрения разработки сетей модули RNN и GRU полностью взаимозаменяемы. Главной проблемой с интеграцией всех представленных модулей является невозможность использования их в модуле Sequential, на котором основывается работа динамического количества слоёв в системе. Для решения этой проблемы можно использовать функцию, которая будет извлекать из выходных данных рекуррентного слоя скрытое состояния и/или состояние ячейки.

Изучив теоретические основания [15], техническую документацию и проверив различные модули рекуррентных слоёв в PyTorch можно сделать вывод, что наиболее перспективным вариантом для разработки является рекуррентные нейронные сети на базе модуля LSTM. Обладая наибольшей точностью среди других модулей рекуррентных слоёв LSTM компенсирует самое долгое время обучения нейронной сети путём большей оптимизации под длинные последовательности данных, сохраняя при это относительную простую структуру, схожую почти по всем параметрам с другими модулями рекуррентных слоёв.

4.3 Работа со сверточным нейронными сетями

Для создания сверточных нейронных сетей фреймворк PyTorch предоставляет широкий ассортимент модулей, главным образом отличающихся друг от друга ожидаемыми входными данными.

Модули nn.Conv1d, nn.Conv2d и nn.Conv3d представляют собой модули, применяющие преобразование $out(N_i, C_{out}) = bias(C_{out}) + \sum_{k=0}^{C_{in}-1} weight(C_{out}, k) * input(N_i, k)$ к входным данным. Главной отличие между ними заключается именно в форме входных данных. По названию nn.Conv1d принимает одномерные тензоры, nn.Conv2d – двумерные, а nn.Conv3d – трехмерные.

У всех трёх слоёв одинаковые параметры:

1) in_channels - Количество каналов во входном изображении;

2) out_channels - Количество каналов, полученных в результате свертки
3) kernel_size - Размер сверточного ядра;
4) stride - ширина полосы свертки. По умолчанию: 1;
5) padding - Добавление разметки к обеим сторонам входного изображения. По умолчанию: 0;
6) dilation - Расстояние между элементами ядра. По умолчанию: 1;
7) groups - Количество блокированных соединений от входных каналов к выходным. По умолчанию: 1;
8) bias - Если True, добавляет обучаемое искажение к выходу. По умолчанию: True;
9) padding_mode – Режим «набивки». По умолчанию: нули.

Слой субдискретизации (иначе подвыборки или «пулинга») представляет собой нелинейное уплотнение карты признаков, при этом группа пикселей (обычно размера 2×2) уплотняется до одного пикселя, проходя нелинейное преобразование. Наиболее употребительна при этом функция максимума. Преобразования затрагивают непересекающиеся прямоугольники или квадраты, каждый из которых ужимается в один пиксель, при этом выбирается пиксель, имеющий максимальное значение. Операция субдискретизации позволяет существенно уменьшить пространственный объём изображения. Субдискретизация интерпретируется так: если на предыдущей операции свёртки уже были выявлены некоторые признаки, то для дальнейшей обработки настолько подробное изображение уже не нужно, и оно уплотняется до менее подробного.

К тому же фильтрация уже ненужных деталей помогает не переобучаться. Слой субдискретизации, как правило, вставляется после слоя свёртки перед слоем следующей свёртки

Слои субдискретизации в PyTorch представлены вариантами функции максимума MaxPool1d, MaxPool2d и MaxPool3d для одномерных, двумерных

и трехмерных тензоров соответственно и AvgPool1d, AvgPool2d и AvgPool3d для функции среднего значения.

Параметры модулей функции максимума;

1) kernel_size - Размер скользящего окна, должен быть больше 0;
2) stride - Ширина скользящего окна, должна быть больше 0. Значение по умолчанию - kernel_size;
3) padding - Неявное отрицательное бесконечное добавление с обеих сторон, должно быть >= 0 и <= kernel_size / 2;
4) dilation - Размах между элементами в скользящем окне, должно быть > 0;
5) return_indices - Если True, возвращает argmax вместе с максимальными значениями;
6) ceil_mode - Если True, то для вычисления выходной формы будет использоваться ceil вместо floor. Это гарантирует, что каждый элемент входного тензора будет покрыт скользящим окном;

Параметры модулей функции среднего значения:

1) kernel_size - Размер скользящего окна, должен быть больше 0;
2) stride - Ширина скользящего окна, должна быть больше 0. Значение по умолчанию - kernel_size;
3) padding - Неявное отрицательное бесконечное добавление с обеих сторон, должно быть >= 0 и <= kernel_size / 2;
4) ceil_mode - Если True, то для вычисления выходной формы будет использоваться ceil вместо floor. Это гарантирует, что каждый элемент входного тензора будет покрыт скользящим окном;
5) count_include_pad - Если значение True, то нулевая прокладка будет включена в расчет усреднения;
6) divisor_override - Если указано, то будет использоваться в качестве делителя, иначе будет использоваться размер области объединения.

5 Конструктор вычислительного эксперимента для СМ МАРС

5.1 Внутренняя логика конструктора

5.1.1 DearPyGUI

DearPyGUI – графический фреймворк [16] для языка Python созданный на основе Dear ImGUI и других расширений с целью создание простого и оптимизированного фреймворка для создания интерфейса. Главным отличием DearPyGUI от Dear ImGUI является использование парадигмы сохраненного режима при создании API, что позволяет сохранять схему рендеринга самой библиотекой вместо клиента, в то время как Dear ImGUI использует схему немедленного режима. Это позволяет создавать намного более динамичные интерфейсы.

Подобно PyQt, DearPyGUI не использует встроенные виджеты, а рисует их с помощью видеокарты персонального компьютера с использованием API рендеринга, а именно Directx11, Metal и Vulkan в зависимости от системы.

Аналогично тому, как Dear ImGUI предоставляет разработчикам игр возможность создавать простые и эффективные графические интерфейсы приложений, DPG предоставляет разработчикам Python возможность создавать быстрые и мощные графические интерфейсы для скриптов.

Цикл рендеринга (или цикл событий) работает непрерывно и отвечает за обработку пользовательского ввода и отрисовку виджетов.

Отрисовка элементов — это то, как DearPyGUI обновляет элементы. DearPyGUI делает это со скоростью обновления монитора, если для параметра set_viewport_vsync установлено значение True. Если значение vsync установлено False, цикл рендеринга будет выполняться максимально быстро. Если внутри цикла рендеринга единовременно будет выполнятся слишком много операций, требующих больших вычислительных затрат, то это может снизить частоту кадров разрабатываемого приложения.

Пример интерфейса созданного на основе фреймворка DearPyGUI представлен на рисунке 5.1.

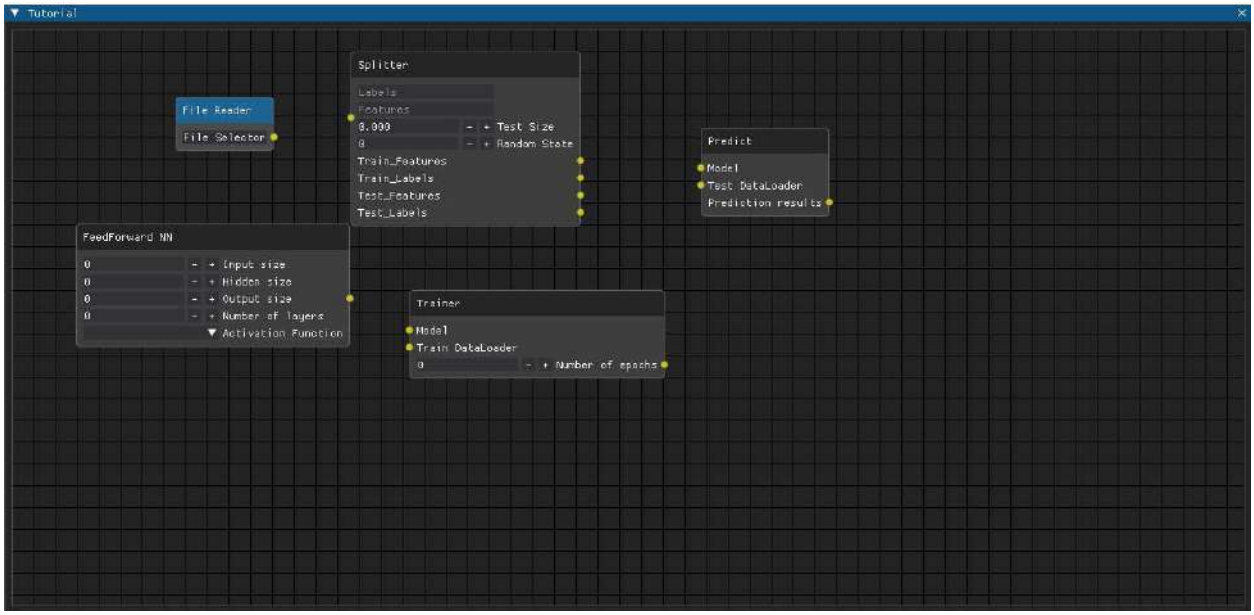

Рисунок 5.1 – Пример интерфейса на DearPyGUI

5.1.2 Архитектура конструктора

Архитектура конструктора завязана на использование множества модулей, которые могут быть связаны друг с другом. Основываясь на технологии NodeEditor из DearPyGUI было решено создать модульную систему, которая будет функционировать за счёт связей между модулями и возможностью глубокой кастомизации.

Основой модуля конструктора является класс BaseNode, который включает в себя логику отрисовки модуля на интерфейсе и построение модуля в зависимости от заложенного разработчиком функционала путём наследования от базового класс при создании собственных классов модулей.

Поля класса BaseNode:

1) param_Id: словарь параметров;
2) node_Id: словарь компонентов;
3) param_widget_dict: словарь параметров виджетов;
4) nodes_dict: словарь связанных модулей;
5) layer: слой на котором находится модуль;

6) name: имя модуля;
7) count: количество компонентов модуля;

params: компоненты модуля. По умолчанию содержит только имя компонента, но при наследовании разработчики могут добавлять различные компоненты и виджеты по своему усмотрению.

Методы класса BaseNode:

1) Draw: отвечает за прорисовку модуля в редакторе;
2) SetParam_dpgID: установка связи между id объекта (пина) в dpg и параметром;
3) SetNode_dpgID: установка связи между id объекта (пина) в dpg и компонентом;
4) RefreshFromGUI_to_Model: обновление параметров компонента - передача данных из графического интерфейса в модель;
5) RefreshFromModel_to_GUI: обновление параметров компонента - передача данных из модели в графический интерфейс;
6) SendValue: отправка данных из одного компонента в другой. Передается название параметра, который необходимо отправить
7) Action: основная функция класса;
8) GetValue: получение данных компонента.

У каждого элемента также могут существовать атрибуты, описывающие какое-либо свойство модуля и позволяющего пользователю взаимодействовать с ним при построении модели, а также контролировать взаимодействие модулей на уровне входных и выходных данных для каждого отдельного элемента, или входных и выходных пинов. Работа атрибутов завязана на классе NodeAttribute.

Поля класса NodeAttribute:

1) FieldType: Тип поля. Для создания пинов выбранного типа;
2) WidgetType: тип виджета. Для создания поля выбранного типа;
3) Value: значение поля;

4) Range: диапазон значений поля (например, для Combobox);

5) Callback: вызываемая функция поля.

Пины описаны в классе Pin.

Поля класса Pin:

1) Name: имя атрибута;

2) DpgID: идентификационный номер атрибута;

3) No: номер атрибута.

Кроме того, для правильной работы связанных модулей необходимо хранить внутри кода данные о их связях. Для этого используется класс Wire.

Поля класса Wire:

1) Id: идентификационный номер связи;

2) Component1: первый соединяемый компонент;

3) nPinComponent1: номера узлов (пинов) первого компонента;

4) Component2: второй соединяемый компонент;

5) nPinComponent2: номера узлов (пинов) второго компонента;

Диаграмма классов, составляющих базовую структуру модуля конструктора представлена на рисунке 5.2.

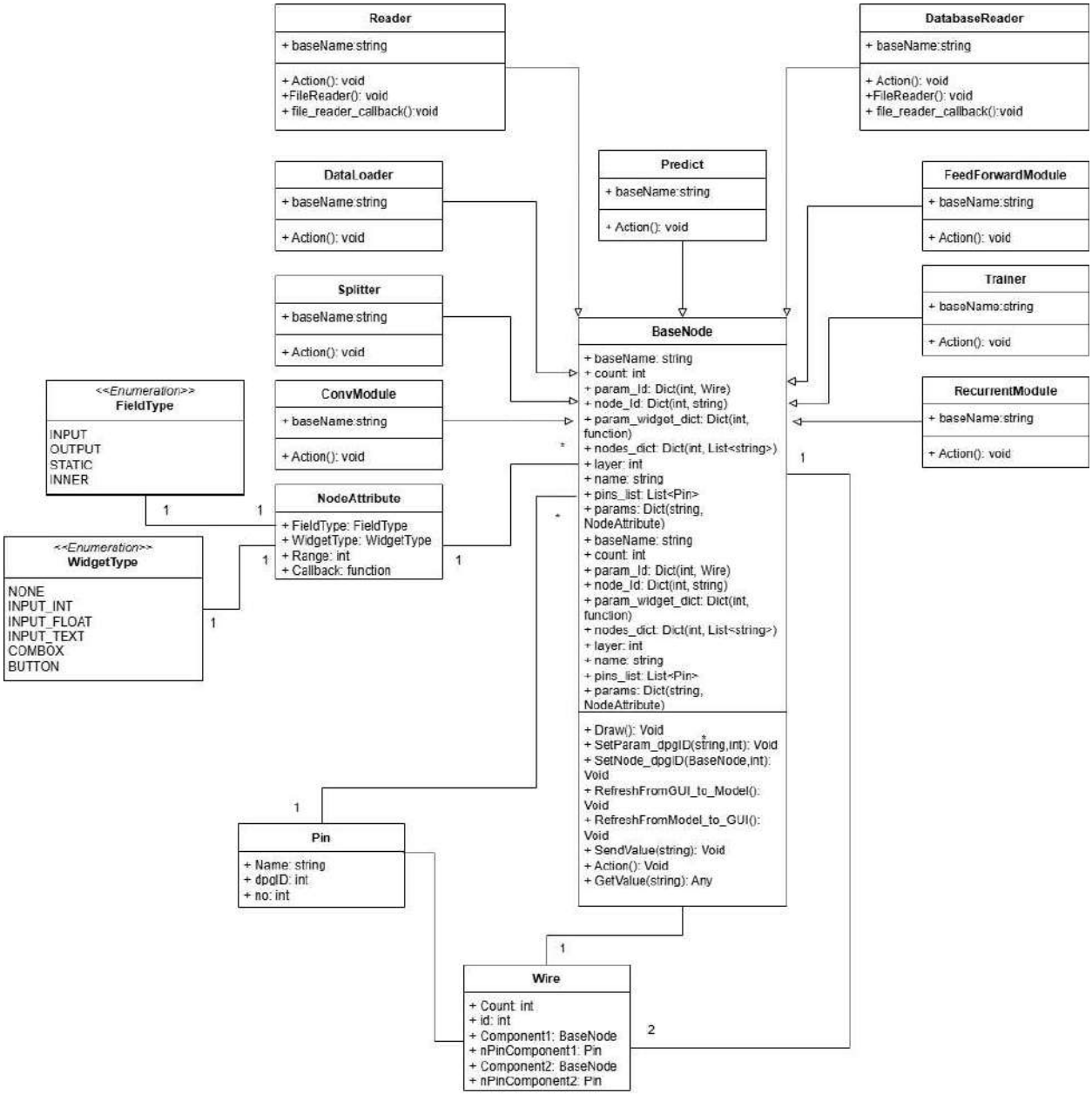

Рисунок 5.2 – Диаграмма классов модуля конструктора

5.2 Интерфейс конструктора

Интерфейс конструктора можно разделить на три части: редактор схемы, библиотека моделей и панель управления.

Интерфейс программы представлен на рисунке 5.3.

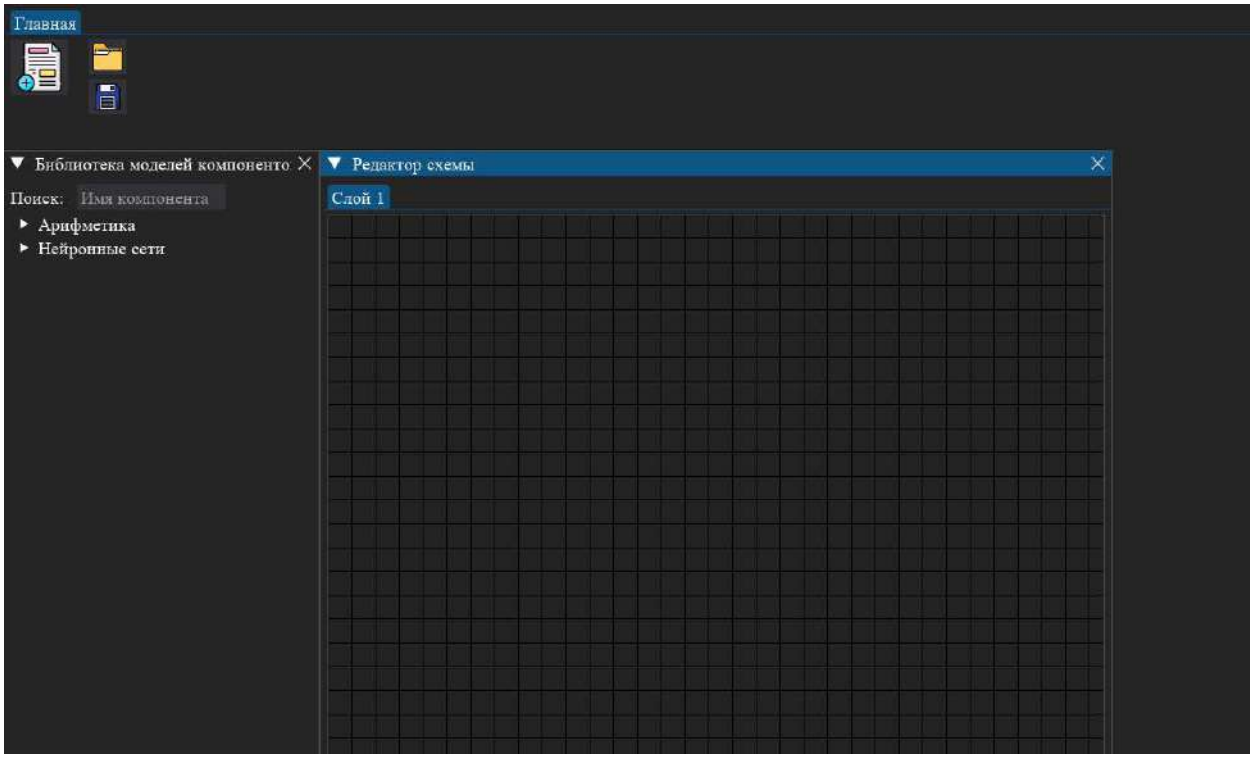

Рисунок 5.3 – Интерфейс конструктора

Редактор схемы представляет собой основную часть интерфейса, которая предназначена для использования конечным пользователем для составления схем и моделей различных процессов. Конструктор повторяет многослойную архитектуру среды моделирования МАРС и позволяет пользователю работать одновременно с несколькими взаимосвязанными слоями.

Для создания модуля внутри редактора пользователь должен обратиться ко второй части интерфейса, а именно библиотеке моделей.

Библиотека моделей представляет собой список доступных пользователю модулей, разделённых по своему предназначению по папкам. Кроме того, вместо папочного доступа пользователь может использовать строку поиска чтобы найти модуль, наиболее подходящий под его запрос.

Для добавления модуля на экран редактора пользователь должен нажать левой кнопкой мыши на строку с выбранным модулем и держа кнопку зажатой перенести стрелку с библиотеки на слой редактора конструктора.

Состояние редактора при переносе модуля с зажатой кнопкой мыши представлен на рисунке 5.4.

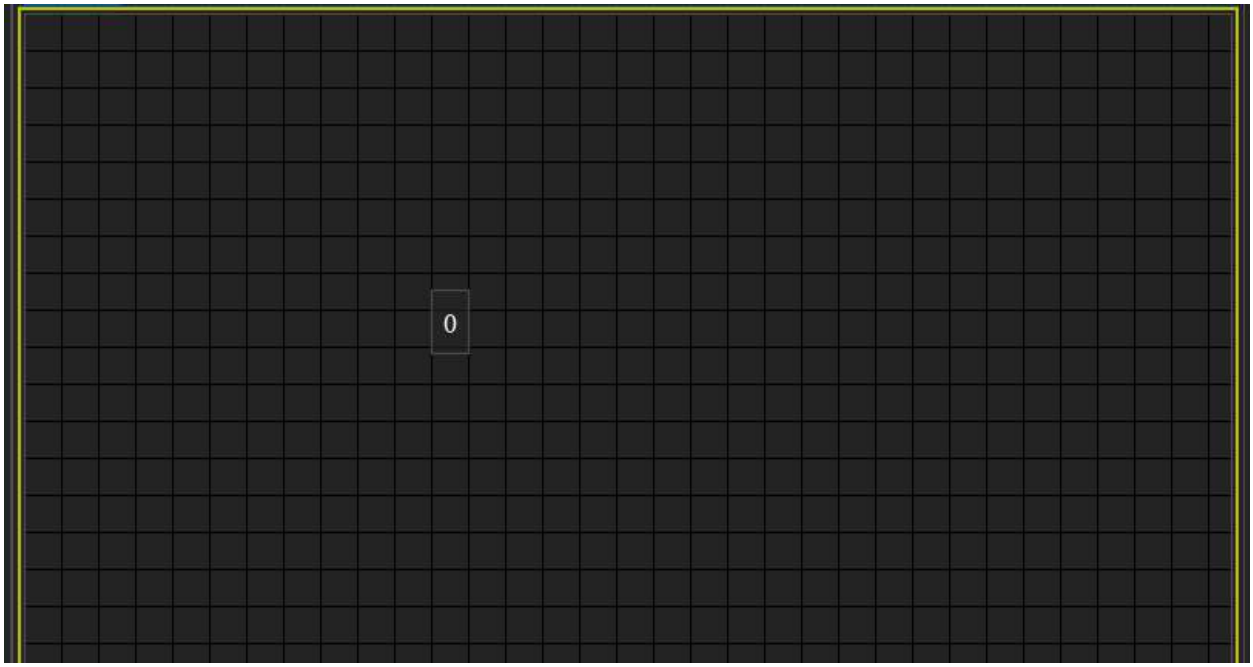

Рисунок 5.4 –Перенос модуля на слой редактора

Результат данной операции можно увидеть на рисунке 5.5.

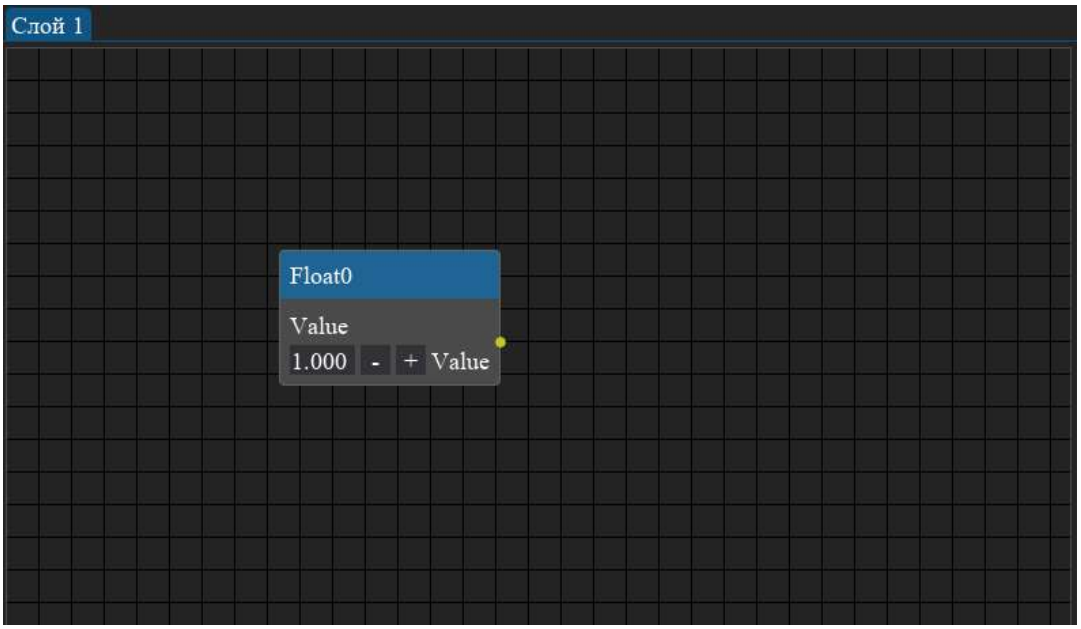

Рисунок 5.5 – Пример создаваемого модуля

Для соединения модулей необходимо соединить при помощи мышки вход и выход двух раздельных модулей, что создаст соответствующую пару в словаре и передаст данные со входа на выход и в зависимости от режима работы системы позволит выполнить действия модуля.

Пример создания связи между двумя модулями можно увидеть на рисунке 5.6.

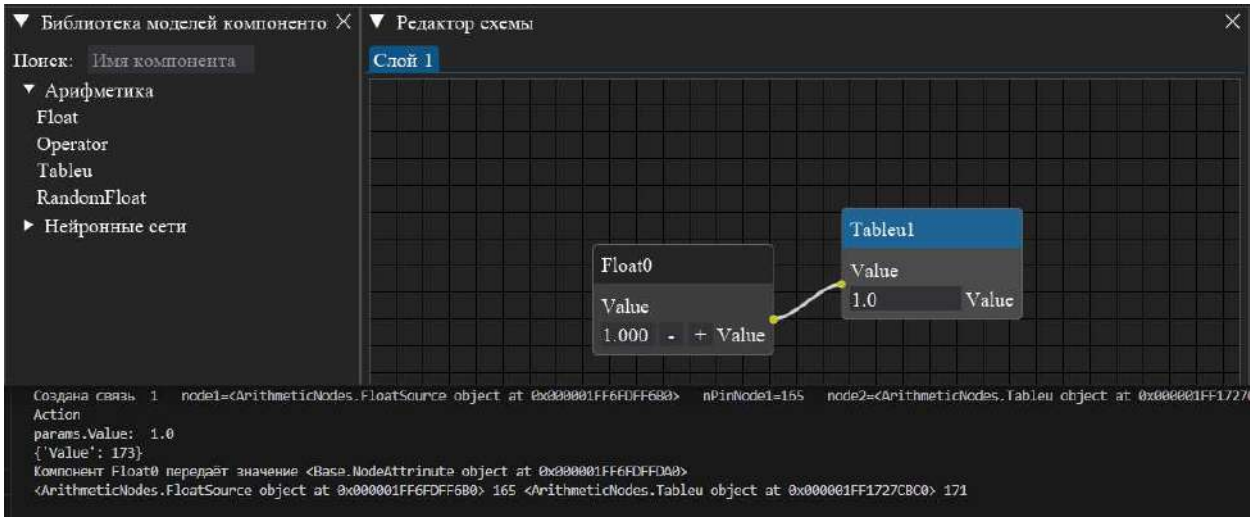

Рисунок 5.6 – Создание связи между модулями

На панели управления находятся функции работы с файлом созданной модели. Пользователь может сохранять, загружать и открывать все файлы нужного формата внутри конструктора.

6 Реализация модулей для конструктора вычислительного эксперимента

6.1 Реализация нейронных сетей для модулей системы

6.1.1 Реализация моноблочных сетей

Для реализации моноблочных сетей было решено создать класс-оболочку, включающею в себя саму нейронную сеть, а также функции тренировки и валидации.

Поскольку планируется создание множества блоков для различных видов нейронных сетей было решено создать абстрактный класс NeuralNetworkBlock от которого будут наследоваться отдельные моноблоки. Данный класс содержит в себе определения для функции train(), которая описывает собой процесс тренировки модели, а также evaluate(), описывающую процесс валидации результатов работы нейронной сети. Далее, сама нейронная сеть будет представлена в виде отдельного класса, наследованного от nn.Module в который при создании экземпляра будут

передаваться различные параметры сети, различные в зависимости от создаваемого блока. Для класса, описывающего моноблок нейронной сети прямого распространения — это будет значения количества входных признаков, количества нейронов в скрытых слоях и количества выходных признаков, а также тип функции активации и количество скрытых слоёв.

Для реализации динамического количества скрытых слоёв с настраиваемым количеством нейронов в них было решено использовать класс-оболочку nn. Sequential и передавать динамически-созданные слои в неё, что позволяет создать единообразную функцию forward, которая будет считывать каждый класс из Sequential в порядке добавления его в оболочку, что решает проблему чёткого определения слоёв в модели.

Диаграмма классов моноблока на примере блока нейронной сети прямого распространения представлена на рисунке 6.1.

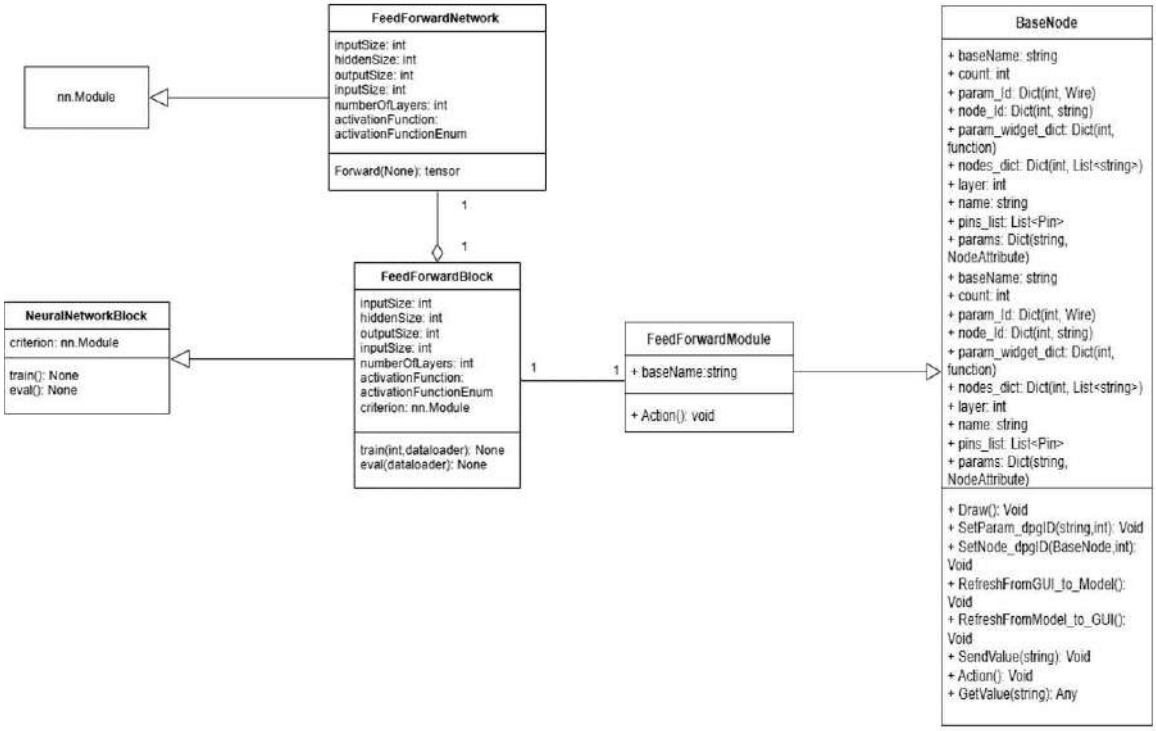

Рисунок 6.1 – Диаграмма составных классов для моноблока нейронной сети прямого распространения

Пример создания моноблока нейронной сети и его использование со стороны кода представлен на листинге 6.1.

Листинг 6.1 – Пример создания нейронной сети моноблочным способом

```
# Загрузка данных
train_loader = DataLoader(train_dataset,batch_size=10, shuffle=True)
block                                                                    =
rb.RecurrentBlock(inputSize=2,hiddenSize=128,outputSize=1,numberOfLayers=1,
criterion=torch.nn.MSELoss())
block.optimizer = torch.optim.Adam(params=block.model.parameters(),lr=0.001)
# #Тренировка
block.train(100,train_loader)
# Предугадывание
predict_y = block.model(torch.from_numpy(x_test).type(torch.float))
```

6.1.2 Реализация многоблочных сетей

Для реализации многоблочной системы было решено использовать функцию exec() языка Python, которая выполняет передаваемый в нее в виде строки код. Это позволяет динамически составлять модель нейронной сети путём добавления различных модулей слоёв в собираемую пользователем строку путём соединения различных блоков в интерфейсе среды моделирования MAPC.

Каждый такой блок представляет собой функцию, принимающую в себя строку и добавляющую к ней строку-определение модуля с различными установленными пользователем параметрами для PyTorch. Поскольку предполагается создание множества вариантов конечной сети было решено использовать шаблон проектирования «Строитель» для упрощения работы с объектом. В конечном итоге был создан класс CustomModelScript, в котором находится строковое поле, содержащее саму строку, а также методы, которые добавляют к строке необходимые строчки кода по пользовательской спецификации.

Сам код исполняется в классе CustomNN, куда передается составленная строка и она выполняется путём её передачи в exec() через блок-класс CustomBlock, в котором определены функции для тренировки и валидации сети..

Пример многоблочного создания нейронной сети со стороны кода представлен на листинге 6.2.

Листинг 6.2 – Пример создания нейронной сети многоблочным способом

```
# Загрузка данных
train_loader = DataLoader(train_dataset,batch_size=10, shuffle=True)
#Определение нейронной сети
code = CustomModelScript()
code.addLinearLayer(2,128)
code.addLinearLayer(128,64)
code.addActivationLayer(activationFunction.ReLU)
code.addLinearLayer(64,1)
block = CB(code_script=code.script, criterion=torch.nn.MSELoss())
block.optimizer = torch.optim.Adam(params=block.model.parameters(),lr=0.001)
#Тренировка
block.train(100,train_loader)
#Предугадывание
predict_y = block.model(torch.from_numpy(x_test).type(torch.float))
```

Структура созданной таким образом нейронной сети представлена на рисунке 6.2.

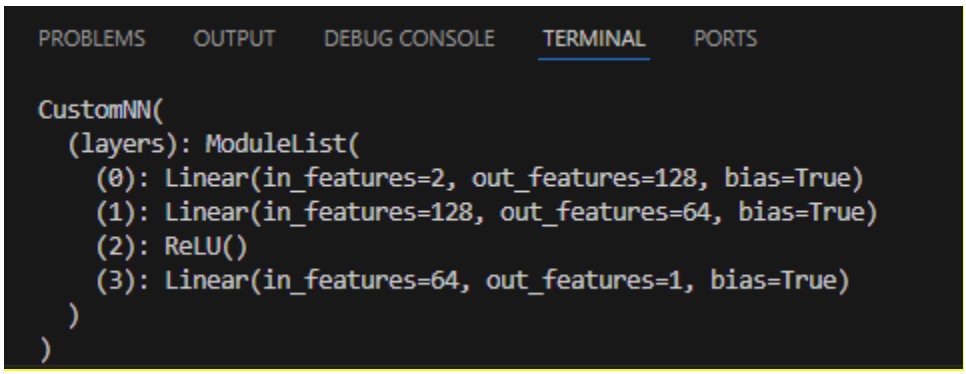

Рисунок 6.2 Образец созданной нейронной сети

Как видно из полученной в результате работы строителя сети, алгоритм добавления блоков создаёт валидную для фреймворка PyTorch нейронную сеть.

6.2 Реализация модулей конструктора нейронных сетей

Reader (от англ. Чтец) – блок, основной задачей которого является считывание данных из файла, их преобразование в объект данных и передача далее по цепочке блоков. Благодаря библиотеке Pandas предоставляется возможность использования в качестве датасетов файлов текстового формата

.CSV и файлов формата Excel .xlsx, .xlsm, .xlsb, .odf, .ods и .odt. На выходе блок отдаёт значение формата DataFrame, содержащие выбранный пользователем датасет.

Входные данные блока:

1) В качестве входных данных блок принимает путь к файлу на персональном компьютере пользователя. Пользователь может прописать путь вручную или же воспользоваться интерфейсом выбора файла, вызываемого через нажатие кнопки «Choose file».

Выходные данные блока:

1) В качестве выходных данных блок предоставляет датасет формата DataFrame через порт Dataset.

Блок Reader в интерфейсе конструктора представлен на рисунке 6.3.

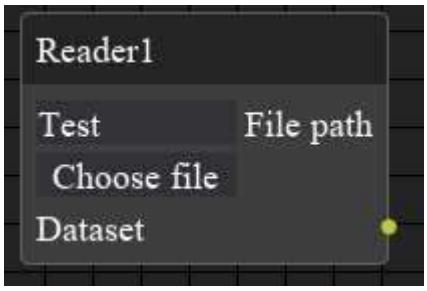

Рисунок 6.3 – Блок Reader

Интерфейс выбора файла представлен на рисунке 6.4.

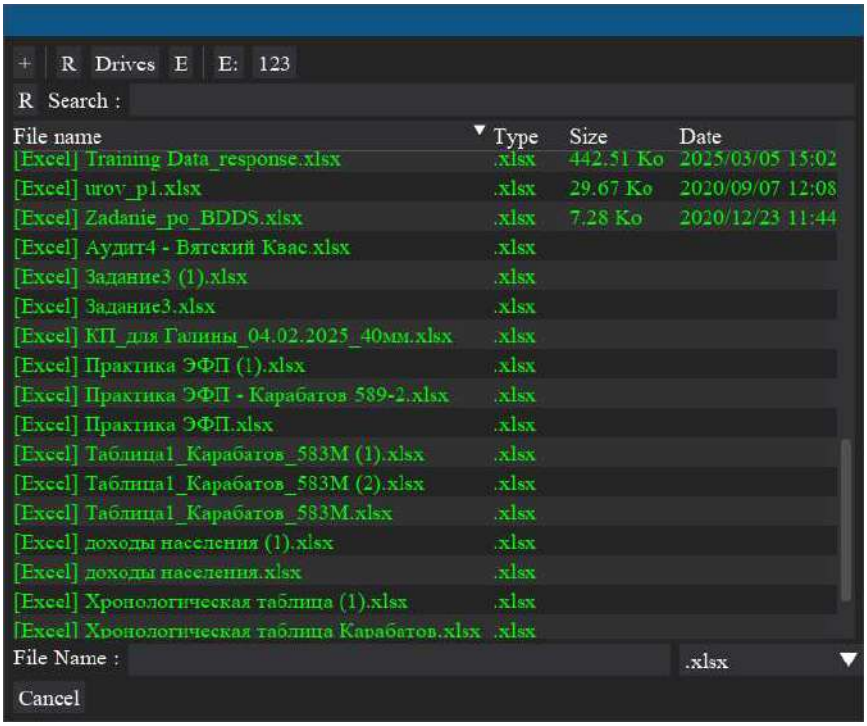

Рисунок 6.4 – Интерфейс выбора файла

Database Reader (от англ. Чтец Базы Данных) – разновидность блока Reader, главной задачей которого является чтение данные, хранящихся в встроенной базе данных на основе технологии SQLite. Для этого используется метод read_sql фреймворка Pandas, на вход которого подаются путь к встроенной базе данных и название таблицы, которую необходимо превратить в датасет. На выходе блок отдаёт значение формата DataFrame, содержащий данные из указанной пользователем таблицы.

Входные данные блока:

1) В качестве входных данных блок принимает путь к файлу на персональном компьютере пользователя. Пользователь может прописать путь вручную или же воспользоваться интерфейсом выбора файла, вызываемого через нажатие кнопки «Choose file».

Выходные данные блока:

1) В качестве выходных данных блок предоставляет датасет формата DataFrame через порт Dataset.

Блок Reader в интерфейсе конструктора представлен на рисунке 6.5.

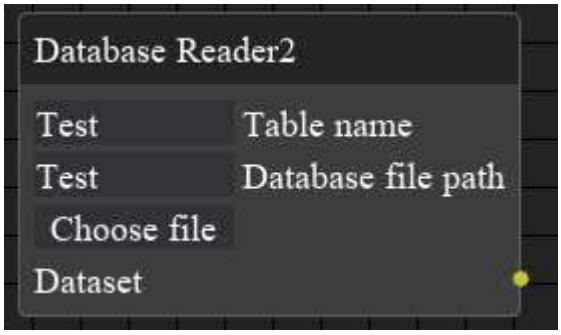

Рисунок 6.5 – Блок Database Reader

Splitter (от англ. Разделитель) – блок, основной задачей которого является разделение датасета на тренировочную и тестовую выборку и получение набора признаков и меток для дальнешего использования в процессе создания и тренировки нейронной сети. Для выполнения этих задач используется модуль train_test_split из библиотеки scikit-learn.

Входные данные блока:

1) Dataset: передаваемый датасет в формате DataFrame;
2) Features: набор интересующих пользователя признаков, передаваемых как строка с названиями признаков, перечисленными через запятую;
3) Labels: набор интересующих пользователя меток, передаваемых как строка с названиями признаков, перечисленными через запятую;
4) Test Size: размер тренировочной выборки, выраженный в виде числа с плавающей запятой со значением от 0 до 1, где остаток от введенного числа будет использован как размер тренировочной выборки.
5) Random State: число для генерации случайного порядка подтасовки в выборках. Целочисленное число.

Входные данные блока:

1) Training Subset: признаки и метки тренировочной выборки;
2) Testing Subset: признаки и метки тестовой выборки.

Блок Splitter в интерфейсе конструктора представлен на рисунке 6.6.

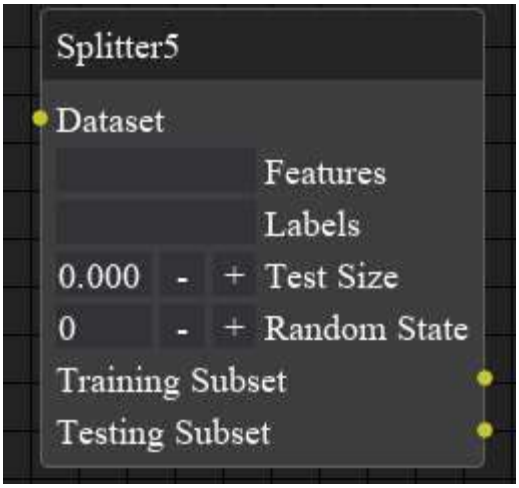

Рисунок 6.6 – Блок Splitter

DataLoader (от англ. Загрузчик Информации) – блок, основной задачей которого является запаковка полученных признаков и меток в формат Pytorch.DataLoader, который можно легко использовать для тренировки и использования нейронной сети. Сначала данные группируют вместе с помощью объекта TensorDataset и далее с его помощью создают объект DataLoader с добавление дополнительных пользовательских параметров, передаваемых в блок.

Входные данные блока:

1) Subset: признаки и метки нужной выборки;
2) Batch Size: размер одной партии данных из разделенного датасета;
3) Shuffle: булевое значение, отвечает за включение функции перетасовки данных. По умолчанию True.

Выходные данные блока:

1) DataLoader: полученный объект загрузчика данных.

Блок DataLoader в интерфейсе конструктора представлен на рисунке 7.7.

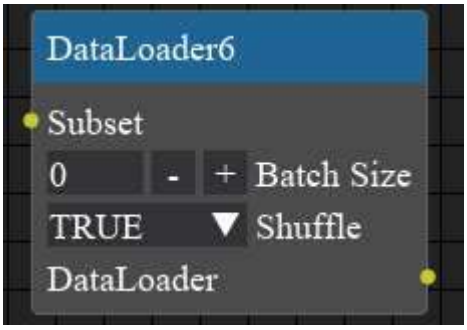

Рисунок 6.7 – Блок DataLoader

Feed Forward Block (от англ. Блок Прямого Распространения) – блок, основной задачей которого является создание модели нейронной сети прямого распространения. Для инициализации модели используется спроектированный ранее моноблок нейронной сети прямого распространения. С помощью блока все необходимые данные передаются в конструктор и на выходе мы получаем сформированную модель, состоящею из самой сети и функций тренировки и валидации.

Входные данные блока:

1) Input Size: количество ожидаемых признаков во входных данных. Целочисленное число;
2) Hidden Size: количество нейронов на скрытых слоях нейронной сети. Целочисленное число;
3) Output Size: ожидаемый размер выходных данных. Целочисленное число;
4) Number of Layers: количество скрытых слоёв в модели. Целочисленное число;
5) Activation Function: пользователь выбирает одну из предложенных функций активаций для использования в создаваемой сети. Значение по умолчанию: ReLU;
6) Criterion: пользователь выбирает одну из предложенных функций потери для использования в создаваемом блоке. Значение по умолчанию: MSE.

Выходные данные блока:

1) Model: сконструированный и инициализированный моноблок с моделью нейронной сети, созданной по пользовательским параметрам.

Блок Feed Forward Block в интерфейсе конструктора представлен на рисунке 7.8.

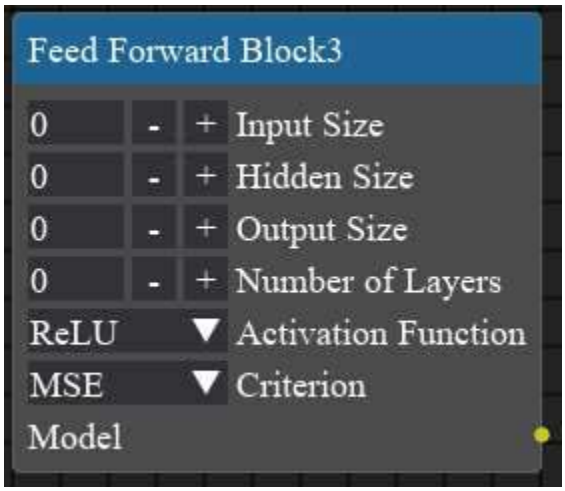

Рисунок 6.8 – Блок FeedForwardBlock

Recurrent Block (от англ. Рекуррентный Блок) – блок, основной задачей которого является создание модели рекуррентной нейронной сети. Для инициализации модели используется спроектированный ранее моноблок рекуррентной нейронной сети. С помощью блока все необходимые данные передаются в конструктор и на выходе мы получаем сформированную модель на основе технологии долгой краткосрочной памяти, состоящею из самой сети и функций тренировки и валидации.

Входные данные блока:

1) Input Size: количество ожидаемых признаков во входных данных. Целочисленное число;
2) Hidden Size: количество нейронов на скрытых слоях нейронной сети. Целочисленное число;
3) Output Size: ожидаемый размер выходных данных. Целочисленное число;
4) Number of Layers: количество скрытых слоёв в модели.

Целочисленное число;

5) Dropout: добавляет слой отключения на выход каждого слоя нейронной сети кроме последнего. Шанс на исключения равен заданному значению;
6) Criterion: пользователь выбирает одну из предложенных функций потери для использования в создаваемом блоке. Значение по умолчанию: MSE.

Выходные данные блока:

1) Model: сконструированный и инициализированный моноблок с моделью нейронной сети, созданной по пользовательским параметрам.

Блок Recurrent Block в интерфейсе конструктора представлен на рисунке 6.9.

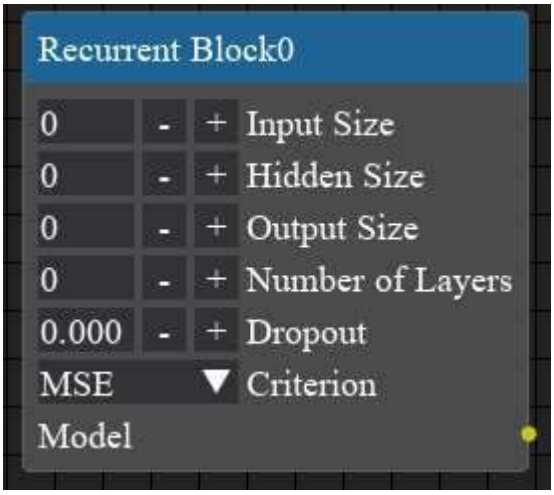

Рисунок 6.9 – Блок RecurrentBlock

Convolution Block (от англ. Сверточный Блок) – блок, основной задачей которого является создание модели сверточной нейронной сети. Для инициализации модели используется спроектированный ранее моноблок сверточной нейронной сети. С помощью блока все необходимые данные передаются в конструктор и на выходе мы получаем сформированную модель, состоящею из самой сети и функций тренировки и валидации.

Входные данные блока:

1) Input Size: количество ожидаемых признаков во входных данных. Целочисленное число;
2) Hidden Size: количество нейронов на скрытых слоях нейронной сети. Целочисленное число;
3) Output Size: ожидаемый размер выходных данных. Целочисленное число;
4) Number of Layers: количество сверточных слоёв в модели. Целочисленное число;
5) Kernel Size: размер сверточного ядра. Целочисленное число;
6) Stride: ширина полосы свертки. Целочисленное число;
7) Dimension: размерность передаваемых данных. Возможные значения: 1D, 2D, 3D;
8) Padding: неявное отрицательное бесконечное добавление с обеих сторон, должно быть >= 0 и <= kernel_size / 2. Целочисленное число;
9) Use Pooling: использование слоёв подвыборки. Булевое значение, значение по умолчанию: True;
10) Activation Function: пользователь выбирает одну из предложенных функций активаций для использования в создаваемой сети. Значение по умолчанию: ReLU;
11) Criterion: пользователь выбирает одну из предложенных функций потери для использования в создаваемом блоке. Значение по умолчанию: MSE.

Выходные данные блока:

1) Model: сконструированный и инициализированный сверточный моноблок с моделью нейронной сети, созданной по пользовательским параметрам.

Блок Convolution Block в интерфейсе конструктора представлен на рисунке 6.10.

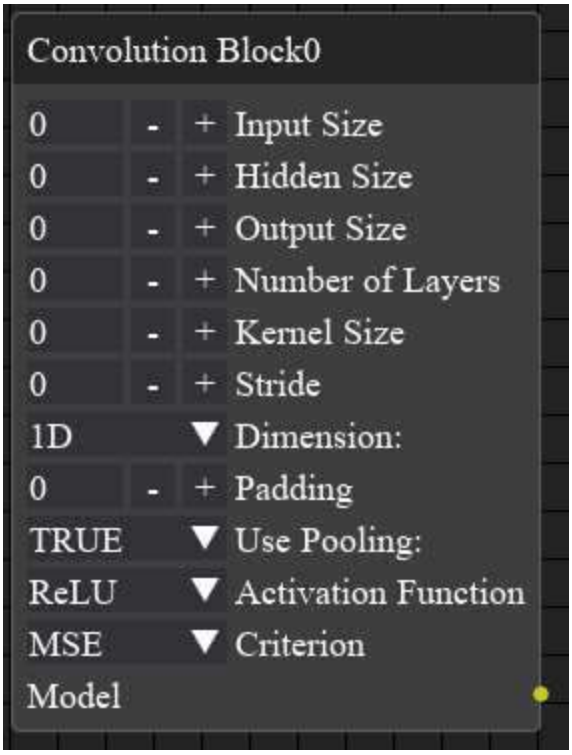

Рисунок 6.10 – Блок Convolution Block

Trainer (от Англ. Тренер) – блок, основной задачей которого является обучение нейронной сети. В блок передаётся сама модель, тренировочные данные и дополнительная информация о процессе, благодаря чему становится возможно провести процесс тренировки. После завершения блок передаёт натренированную модель далее по цепи.

Входные данные блока:

1) Model: сформированная нейронная сеть. В зависимости от типа передаваемой сети внутренняя логика тренировки может изменяться, однако для пользователя этого не будет заметно;
2) Number of Epoch: количество итераций тренировки. Целочисленное число;
3) Training Dataloader: DataLoader с тренировочной выборкой данных.

Выходные данные блока:

1) Trained Model: натренированная нейронная сеть.

Блок Trainer в интерфейсе конструктора представлен на рисунке 6.11.

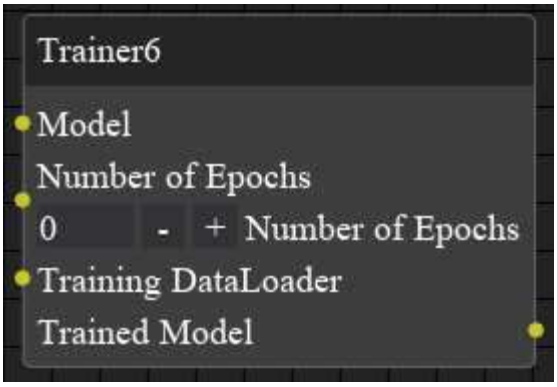

Рисунок 6.11 – Блок Trainer

Блок Predict (от Англ. Предсказать) – блок, основной задачей которого является прогнозирование на основе тестовой выборки данных. В блок передаётся натренированная нейронная сеть и тестовые данные, а на выход отправляются результаты прогнозирования.

Входные данные блока:

1) Model: натренированная модель нейронной сети;
2) Test Data: данные тестовой выборки в формате DataLoader;

Выходные данные блока:

1) Prediction Result: результаты прогнозирования.

Блок Predict в интерфейсе конструктора представлен на рисунке 6.12.

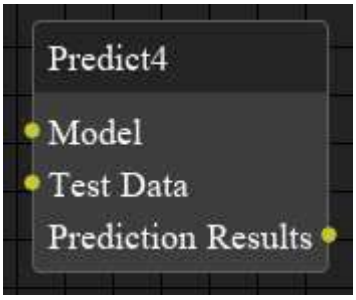

Рисунок 6.12 – Блок Predictor

Краткое описание всех разработанных модулей представлено в таблице 6.1.

Таблица 6.1 – Краткое изложение разработанных модулей

Название модуля	Функционал Модуля	Входы модуля	Выходы модуля
Reader	Получение данных из файла	Нет	Датасет

Окончание таблицы 6.1.

Database Reader	Получение данных из базы данных	Нет	Датасет
Splitter	Разделение данных на выборки	Датасет	Тренировочная выборка, тестовая выборка
DataLoader	Подготовка выборки к тренировке	Тренировочная выборка	Подготовленные данные
Feed Forward Block	Моноблок нейронной сети прямого распространения	Нет	Модель нейронной сети
Recurrent Block	Моноблок рекуррентной нейронной сети	Нет	Модель нейронной сети
Convolutional Block	Моноблок сверточной нейронной сети	Нет	Модель нейронной сети
Trainer	Тренировка нейронной сети	Модель нейронной сети, Dataloader	Натренированная модель нейронной сети
Predict	Предугадывание данных	Натренированная модель нейронной сети	Полученные нейросетью данные
Convolutional Block	Моноблок сверточной нейронной сети	Нет	Модель нейронной сети

6.3 Сравнение работоспособности модулей

Для проверки работоспособностей модулей было принято решение провести серию испытаний двух вариантов запуска моноблоков нейронной сети: непосредственно из конструктора вычислительного эксперимента и напрямую из кода. В качестве рассматриваемой задачи будет использована задача регрессии для датасета цен на недвижимость в Бостоне. В качестве основного инструмента будет использоваться моноблок нейронной сети прямого распространения.

Код для запуска решения задачи из кода представлен на листинге 6.2.

Листинг 6.2 – Создание нейронной сети для решения задачи регрессии

```
# Проверка устройства
device = 'cuda' if torch.cuda.is_available() else 'cpu'
# Загрузка данных
data = pd.read_csv('boston.csv')
# Нормализация данных
scaler = MinMaxScaler()
data_scaled = pd.DataFrame(scaler.fit_transform(data), columns=data.columns)
feature_list = ['LSTAT','RM'] # ---Сюда можно дописывать входные признаки
inputs_count = len(feature_list) #это число определяет количество входов в
нейронной сети
x = data_scaled[feature_list].to_numpy()
y = data_scaled['MEDV'].to_numpy()
# Разделение данных на обучающую и тестовую выборки
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2,
random_state=42)
#train_dataset                                                          =
TensorDataset(torch.from_numpy(x_train).type(torch.float),torch.from_numpy(y_
train.values).type(torch.float))
train_dataset = BostonDataset(x_train, y_train)
# Загрузка данных
train_loader = DataLoader(train_dataset,batch_size=10, shuffle=True)
block                                                                  =
ffb.FeedForwardBlock(inputSize=2,hiddenSize=128,outputSize=1,numberOfLayers=1
, criterion=torch.nn.MSELoss())
block.optimizer = torch.optim.Adam(params=block.model.parameters(),lr=0.001)
# #Тренировка
block.train(100,train_loader)
# Предугадывание
predict_y = block.model(torch.from_numpy(x_test).type(torch.float))
accuracy = r2_score(y_test,predict_y.detach().numpy())
print(accuracy)
```

Код для запуска решения задачи из конструктора вычислительного эксперимента представлен на рисунке 6.13.

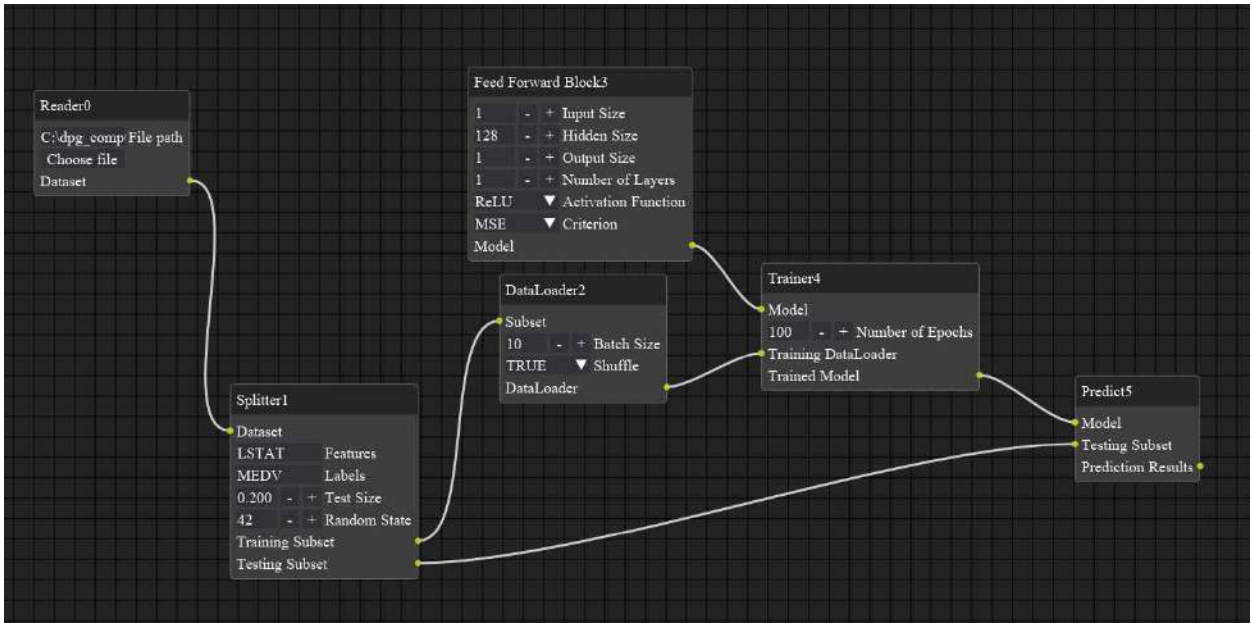

Рисунок 6.13 – Создание нейронной сети для решения задачи регрессии в конструкторе вычислительного эксперимента

Результаты замеров представлены в таблице 6.2.

Таблица 6.2 – Сравнение точности решения задачи регрессии при решении внутри конструктора вычислительного эксперимента

№	R^2, внутри конструктора	R^2, в коде
1	0.662575	0.690188
2	0.697060	0.702941
3	0.687588	0.680013
4	0.677293	0.692351
5	0.695979	0.698372
6	0.697700	0.676089
7	0.687228	0.684607
8	0.693805	0.694148
9	0.683546	0.645039
10	0.690312	0.697325
Среднее значение	0,676444	0,693757

Из полученных результатов можно сделать вывод, что созданные в конструкторе вычислительного эксперимента модули полностью повторяют функционал решений, созданных в коде.

Заключение

В рамках данной научно-исследовательской работы был исследован конструктор вычислительного эксперимента СМ МАРС, а также были спроектированы и реализованы основные модули необходимые для построения нейронных сетей в СМ МАРС.

В результате проектирования было составлено описание элементов, которые необходимо реализовать для выполнения поставленных задач.

В результате выполнения работы были созданы модули, позволяющие конечному пользователю работать с данными с целью подготовки их к обработке нейронными сетями и создавать рабочие нейронные сети с возможностью изменения выбранных параметров.

Список разработанных модулей:

1) Блок чтения файла датасета из файла табличного формата;
2) Блок чтения файла датасета из базы данных;
3) Блок разделения датасета на выборки;
4) Блок создания загрузчика данных;
5) Моноблок нейронной сети прямого распространения;
6) Моноблок рекуррентной нейронной сети;
7) Моноблок сверточной нейронной сети;
8) Блок тренировки нейронной сети;
9) Блок предсказания.

Оценка приобретенных компетенций:

ПК-1: в рамках прохождения учебной практики были разработаны технические требования к разрабатываемой системе и её модулям.

ПК-2: в рамках прохождения учебной практики был разработан модуль, позволяющий пользователям взаимодействовать с базами данных на основе технологии SQLite.

ПК-3: в рамках учебной практики был составлен проект и план реализации модулей средств по созданию нейронных сетей.

ПК-4: в рамках учебной практики были реализованы модули по созданию нейронных сетей для конструктора вычислительного эксперимента.

Список использованных источников

1) Образовательный стандарт вуза ОС ТУСУР 01-2021. Работы студенческие по направлениям подготовки и специальностям технического профиля. Общие требования и правила оформления [Электронный ресурс]. URL: https://regulations.tusur.ru/#/documents/87 (дата обращения:15.05.2025).

2) Николенко С.И., Кадурин А.А., Архангельская Е.В. Глубокое Обучение. Погружение в мир нейронных сетей. СПб: Питер, 2018. 480 с.

3) Дмитриев В.М., Шутенков А.В., Зайченко Т.Н., Ганджа Т.В. МАРС – среда моделирования технических устройств и систем – Томск: В-Спектр, 2011. 277 с.

4) Deep Learning Toolbox [Электронный ресурс]. URL: https://www.mathworks.com/products/deep-learning.html (дата обращения: 15.05.2025).

5) Deep Network Desiner [Электронный ресурс]. URL: https://www.mathworks.com/help/deeplearning/ref/deepnetworkdesigner-app.html (дата обращения: 15.05.2025).

6) Neural Net Fitting [Электронный ресурс]. URL: https://www.mathworks.com/help/deeplearning/ref/neuralnetfitting-app.html (дата обращения: 15.05.2025).

7) NeuroGenetic Optimizer [Электронный ресурс]. URL: https://www.biocompsystems.com/content/products/analytics/ngo/ (дата обращения: 15.05.2025).

8) SimInTech [Электронный ресурс]. URL: https://simintech.ru/ (дата обращения: 15.05.2025).

9) Петрова Е.С., Коновалов Н.А. Обзор библиотек для интеграции кода python в приложение на C++ // Сборник избранных статей научной сессии ТУСУР. Томск: В-Спектр, 2023. Ч. 2. С. 121–123.

10) Библиотека TensorFlow [Электронный ресурс]. URL: https://www.tensorflow.org/?hl=ru (дата обращения: 15.05.2025).

11) Библиотека Theano [Электронный ресурс]. URL: https://github.com/Theano/Theano (дата обращения: 10.05.2024).

12) Библиотека Keras [Электронный ресурс]. URL: https://keras.io/ (дата обращения: 15.05.2025).

13) Библиотека PyTorch [Электронный ресурс]. URL: https://pytorch.org/ (дата обращения: 15.05.2025).

14) Библиотека PyBrain [Электронный ресурс]. URL: https://github.com/pybrain/pybrain (дата обращения: 15.05.2025).

15) Картер Д. Нейросети. Обработка естественного языка. – М: «Автор», 2023. – 156 с.

16) Документация DearPyGUI [Электронный ресурс]. URL: https://dearpygui.readthedocs.io/en/latest/ (дата обращения: 15.05.2025).