**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ**

**ФЕДЕРАЦИИ**

Федеральное государственное бюджетное образовательное

учреждение высшего образования

**«Сибирский государственный университет науки и технологий  
имени академика М.Ф. Решетнева»**

Институт информатики и телекоммуникаций

Кафедра информатики и вычислительной техники

**КУРСОВАЯ РАБОТА**

Прогнозирование временного ряда с использованием нейронной сети

тема работы

Руководитель \_\_\_\_\_\_\_\_\_\_ Тынченко В.В.

подпись, дата инициалы, фамилия

Обучающийся БПИ21-02, 211219010 \_\_\_\_\_\_\_\_\_\_ Вятскин Д.В.

номер группы, зачетной книжки подпись, дата инициалы, фамилия

Красноярск 2022

Институт информатики и телекоммуникаций

Кафедра информатики и вычислительной техники

|  |
| --- |
| **ЗАДАНИЕ**  на курсовую работу по дисциплине «Алгоритмы и структуры данных»  студенту |
| Группа БПИ21-02 Форма обучения очная |  |
| 1. Тема работы: Прогнозирование временного ряда с использованием   нейронной сети |
| 2. Срок сдачи студентом работы: 28.12.22 |
| 3. Перечень вопросов, подлежащих разработке при написании теоретической части:  Изучение понятия временного ряда, смежных понятий; изучение понятия  нейронной сети, смежных понятий; изучение основных принципов работы  нейронных сетей. |
| 4. Перечень вопросов, подлежащих разработке при написании практической части (либо указать номер варианта задания)  Реализация ПО на базе нейронной сети, результат работы которого будет  применим для прогнозирования произвольных временных рядов (анализ требо-  ваний, проектирование, кодирование, обучение нейронной сети, тестирование и  отладка). |
| 5. Дата выдачи задания: 02.09.22 |
| Руководитель: Тынченко В.В. |
| Подпись |
|  |
| Задание принял к исполнению (дата): 02.09.22 |
| (подпись студента) |

**Содержание**

[Введение 4](#_Toc122839488)

[1 Анализ предметной области 5](#_Toc122839489)

[1.1 Понятие временного ряда 5](#_Toc122839490)

[1.2 Понятие нейронной сети 5](#_Toc122839491)

[1.2.1 История глубокого обучения 5](#_Toc122839492)

[1.2.2 Понятие нейрона в глубоком обучении 6](#_Toc122839493)

[1.2.2.1 Нейрон в биологии 6](#_Toc122839494)

[1.2.2.2 Нейрон в глубоком обучении 7](#_Toc122839495)

[1.2.3 Функции активации в глубоком обучении 8](#_Toc122839496)

[1.2.3.1 Ступенчатая функция 9](#_Toc122839497)

[1.2.3.2 Линейная функция 9](#_Toc122839498)

[1.2.3.3 Сигмоидная функция 10](#_Toc122839499)

[1.2.3.4 Функция ReLU 11](#_Toc122839500)

[1.2.3.5 Функция гиперболического тангенса 12](#_Toc122839501)

[1.3 Выводы 12](#_Toc122839502)

[2 Описание программы 13](#_Toc122839503)

[2.1 Общая характеристика программы 13](#_Toc122839504)

[2.2 Структура программы и данных 13](#_Toc122839505)

[2.3 Интерфейс программы 16](#_Toc122839506)

[2.4 Тестирование программы 17](#_Toc122839507)

[2.5 Выводы 21](#_Toc122839508)

[3 Эксплуатация программы 22](#_Toc122839509)

[3.1 Определение требований к прогнозу 22](#_Toc122839510)

[3.2 Опыты 22](#_Toc122839511)

[3.3 Проверка модели 27](#_Toc122839512)

[3.4 Выводы 29](#_Toc122839513)

[Заключение 30](#_Toc122839514)

[Список использованных источников 31](#_Toc122839515)

[Приложение А - Листинг программы 32](#_Toc122839516)

[Приложение Б – Проверка на оригинальность 40](#_Toc122839517)

Введение

Нейронные сети уже долгое время являются важной частью нашей жизни. Они используются повсюду - от создания самоуправляемых автомобилей до прогнозирования будущих погодных условий. Тем не менее, сейчас их потенциал только начинает раскрываться.

В наши дни эта технология уже не является прерогативой крупных компаний и государственных учреждений. Малые предприятия тоже активно используют её в своих производственных процессах, хотя и не всегда осознают, что она лежит в основе многих используемых ими инструментов.

Нейронные сети - это невероятно сложные и децентрализованные системы. Они работают через сеть взаимосвязанных узлов, где каждый узел способен обрабатывать информацию в соответствии со своими функциями. Узлы могут общаться друг с другом с помощью определенных сигналов, что позволяет им формировать сложное поведение и решения на основе входных данных, получаемых от соседних узлов.

Данная система используется с момента ее изобретения в 1950 году Уорреном МакКаллохом и Уолтером Питтсом в Массачусетском технологическом институте как средство моделирования обработки информации об объектах или действиях нейронами в мозге. Со временем было разработано множество различных типов нейронных сетей, которые применяются различными способами в таких отраслях, как финансы, здравоохранение, маркетинг, и множестве других сфер.

Предметом исследования данной работы является применение нейронных сетей для прогнозирования временных рядов.

Цель работы – разработать программное обеспечение на основе нейронной сети, результатом работы которого будет прогнозирование заданного временного ряда в конкретный момент времени с требуемой точностью.

Задачи:

- провести анализ предметной области;

- разработать ПО для работы с произвольным временным рядом;

- провести обучение ПО для работы с частным временным рядом;

- протестировать программу.

1. Анализ предметной области
   1. Понятие временного ряда

РЯДЫ́ ДИНА́МИКИ [[1](#ВременнойРяд)] - ста­ти­стические дан­ные, ото­бра­жаю­щие раз­ви­тие во вре­ме­ни изу­чае­мо­го яв­ле­ния; так­же на­зы­ва­ют хро­но­ло­ги­че­ски­ми или вре­мен­ны́­ми ря­да­ми.

По­сле­до­ва­тель­но рас­по­ло­жен­ные во вре­ме­ни ста­ти­стические дан­ные на­зы­ва­ют­ся уров­ня­ми рядов динамики. Они долж­ны быть со­по­ста­ви­мы ме­ж­ду со­бой в тер­ри­то­ри­аль­ном раз­ре­зе, по кру­гу ох­ва­ты­вае­мых объ­ек­тов на­блю­де­ния, ме­то­ди­ке рас­чё­та, кри­тической да­те (мо­мент вре­ме­ни, на ко­то­рый про­из­во­дит­ся на­блю­де­ние ох­ва­ты­вае­мых объ­ек­тов), струк­ту­ре.

В ка­ж­дом ряде динамики име­ет­ся 2 основных эле­мен­та: по­ка­за­тель вре­ме­ни *t*; со­от­вет­ст­вую­щие им уров­ни раз­ви­тия изу­чае­мо­го яв­ле­ния  [Рисунок 1].

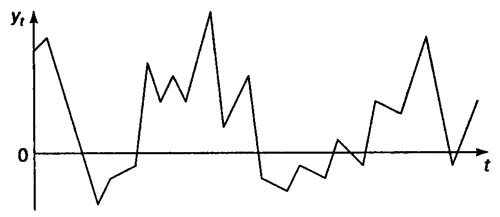


Рисунок – Пример временного ряда

В за­ви­си­мо­сти от ха­рак­те­ра изу­чае­мо­го яв­ле­ния уров­ни ряды динамики мо­гут от­но­сить­ся к оп­ре­де­лён­ным да­там (мо­мен­там) вре­ме­ни и к отд. пе­рио­дам. В со­от­вет­ст­вии с этим ряды динамики под­раз­де­ля­ют­ся на мо­мент­ные и ин­тер­валь­ные.

Мо­мент­ные ряды динамики ото­бра­жа­ют со­стоя­ние изу­чае­мых яв­ле­ний на оп­ре­де­лён­ные да­ты (мо­мен­ты) вре­ме­ни.

Ин­тер­валь­ные ряды динамики ото­бра­жа­ют ито­ги раз­ви­тия изу­чае­мых яв­ле­ний за отдельные ин­тер­ва­лы вре­ме­ни.

* 1. Понятие нейронной сети
     1. История глубокого обучения

Концепция глубокого обучения была разработана Джеффри Хинтоном в 1980-х годах. Его многие считают отцом-основателем области глубокого обучения. Он создал концепцию "нейронной сети", которая представляет собой алгоритм глубокого обучения, структурированный подобно организации нейронов в мозге. Хинтон использовал этот подход, потому что человеческий мозг, вероятно, является самым мощным вычислительным механизмом, известным на сегодняшний день.

Структура, которую создал Хинтон, называется искусственной нейронной сетью (или сокращенно искусственной нейросетью). Вот краткое описание того, как они функционируют:

* искусственные нейронные сети состоят из слоев узлов;
* каждый узел устроен так, чтобы вести себя подобно нейрону в мозге;
* первый слой нейронной сети называется входным, за ним следуют скрытые слои и, наконец, выходной слой;
* каждый узел нейронной сети выполняет определенный расчет, который передается другим узлам, расположенным глубже в нейронной сети.
  + 1. Понятие нейрона в глубоком обучении

Нейроны являются важнейшим компонентом любой модели глубокого обучения. Невозможно полностью понять глубокое обучение, не имея знаний о работе нейронов.

* + - 1. Нейрон в биологии

Нейроны в глубоком обучении были вдохновлены нейронами человеческого мозга [Рисунок 2].

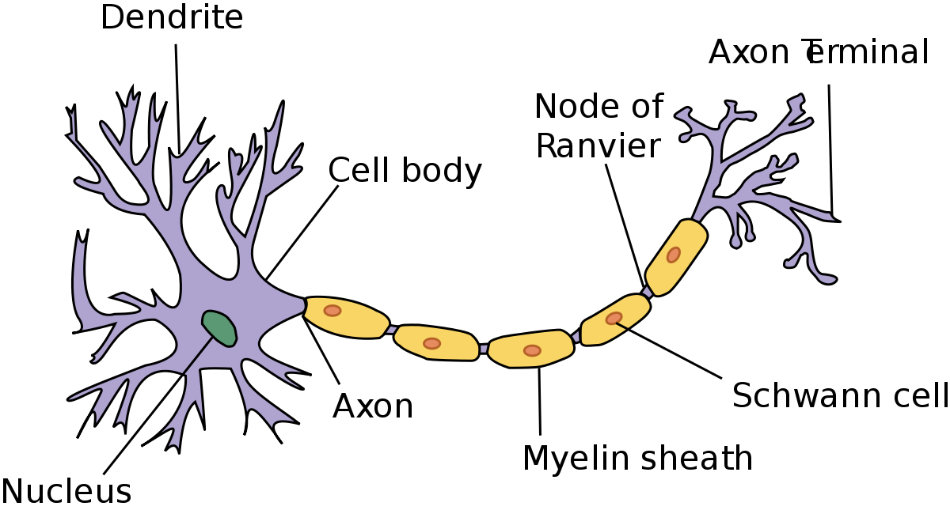


Рисунок – Нейрон человеческого мозга

Как можно видеть, нейроны имеют довольно интересную структуру. Группы нейронов работают вместе внутри человеческого мозга, чтобы выполнять функции, необходимые нам в повседневной жизни.

Вопрос, который задал Джеффри Хинтон во время своего фундаментального исследования нейронных сетей, заключался в том, сможем ли мы создать компьютерные алгоритмы, которые ведут себя аналогично нейронам в мозге. Была надежда, что, имитируя структуру мозга, мы сможем воспроизвести часть его возможностей.

Для этого исследователи изучали поведение нейронов в мозге. Одним из важных наблюдений было то, что нейрон одиночный нейрон не представляет никакой вычислительной мощности и не имеет смысла. Для создания любой значимой функциональности необходимы сети нейронов.

Это происходит потому, что нейроны функционируют, принимая и отправляя сигналы. Точнее, дендриты нейрона принимают сигналы и передают их по аксону.

Дендриты одного нейрона соединены с аксоном другого нейрона. Эти связи называются синапсами - понятие, которое было обобщено в области глубокого обучения.

* + - 1. Нейрон в глубоком обучении

Нейроны в моделях глубокого обучения - это узлы, через которые проходят данные и вычисления.

Нейроны работают следующим образом [Рисунок 3]:

* Они получают один или несколько входных сигналов (из входного набора данных или от предыдущего слоя сети).
* Они выполняют некоторые вычисления.
* Они передают выходные сигналы на следующий слой сети.

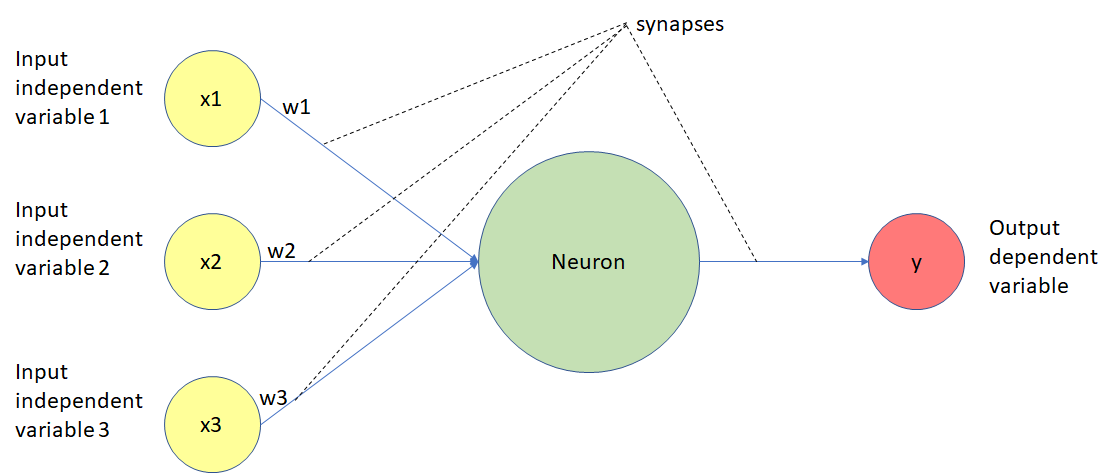


Рисунок – Диаграмма функционирования нейрона в нейронной сети глубокого обучения

Нейроны в модели глубокого обучения способны иметь синапсы, которые соединяются с более чем одним нейроном в предыдущем слое. Каждый синапс имеет соответствующий вес, который влияет на значимость предыдущего нейрона в общей нейронной сети.

Веса - очень важная тема в области глубокого обучения, поскольку регулировка весов модели является основным способом обучения моделей глубокого обучения.

Как только нейрон получает входные данные от нейронов предыдущего слоя модели, он суммирует каждый сигнал, умноженный на соответствующий вес, и передает их на функцию активации [Рисунок 4]

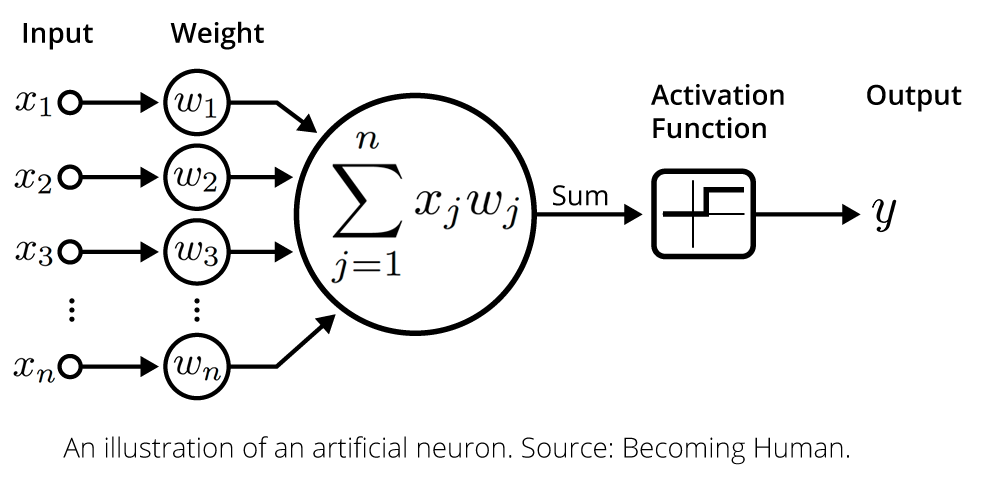


Рисунок – Суммирование произведений входных значений на их веса

Функция активации вычисляет выходное значение для нейрона. Это выходное значение затем передается на следующий слой нейронной сети через другой синапс.

* + 1. Функции активации в глубоком обучении

Нейроны получают входные сигналы от предыдущего слоя нейронной сети. Взвешенная сумма этих сигналов подается на функцию активации нейрона, а затем выход функции активации передается на следующий слой сети.

Рассмотрим нейрон:

Значение Y может быть любым в диапазоне от -**∞** до +**∞**. В действительности нейрон не знает границу, после которой следует активация. Чтобы решить, должен ли нейрон быть активирован, используется активационная функция. Она проверяет произведенное нейроном значение Y на предмет того, должны ли внешние связи рассматривать этот нейрон как активированный, или его можно игнорировать.

Существует пять основных типов функций активации:

* ступенчатые функции;
* линейные функции;
* сигмоидные функции;
* функции Rectified Linear Unit (ReLU);
* функции гиперболического тангенса.
  + - 1. Ступенчатая функция

Ступенчатые функции вычисляют выходной сигнал в зависимости от того, лежит ли его вход выше или ниже определенного порога.

Математическое определение ступенчатой функции [Рисунок 5]:

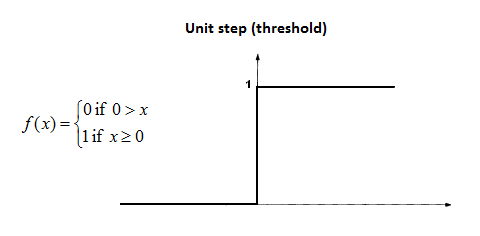


Рисунок – Ступенчатая функция

Глядя на изображение выше, легко понять, почему ступенчатую функцию иногда также называют функцией единичного шага или пороговой функцией.

Ступенчатые функции похожи на булевы переменные в программировании. Их вычисляемое значение равно либо 1 (эквивалентно True), либо 0 (эквивалентно False).

Это простой способ, однако в нём есть недостатки. Такая функция отлично работает для бинарной классификации, но она не работает, когда для классификации требуется большее число нейронов и количество возможных классов больше двух.

* + - 1. Линейная функция

Линейная функция представляет собой прямую линию и пропорциональна входу (то есть взвешенной сумме на этом нейроне).

Математическое определение линейной функции: [Рисунок 6].

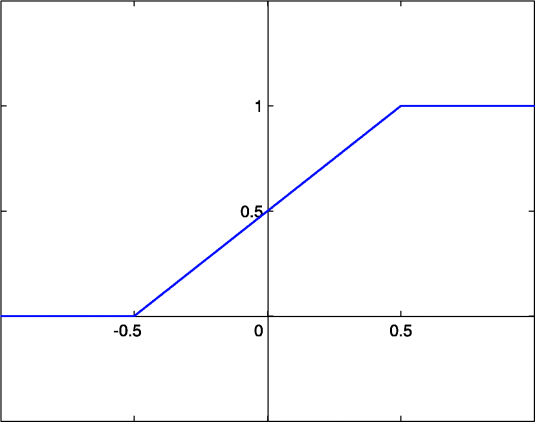


Рисунок – Линейная функция

Такой выбор активационной функции позволяет получать спектр значений, а не только бинарный ответ. Можно соединить несколько нейронов вместе и, если более одного нейрона активировано.

Не имеет значения, сколько слоев мы имеем. Если все они по своей природе линейные, то финальная функция активации в последнем слое будет просто линейной функцией от входов на первом слое.

Это означает, что два слоя (или N слоев) могут быть заменены одним слоем. Таким образом мы теряем возможность делать наборы из слоев. Не важно, в каком порядке мы будем располагать слои, вся нейронная сеть все равно будет подобна одному слою с линейной функцией активации (комбинация линейных функций линейным образом — другая линейная функция).

* + - 1. Сигмоидная функция

Функция сигмоиды хорошо известна в сообществе специалистов по обработке данных благодаря ее использованию в логистической регрессии - одной из основных методик машинного обучения, применяемых для решения задач классификации.

Функция сигмоида может принимать любое значение, но всегда вычисляет значение между 0 и 1.

Математическое определение сигмоидальной функции [Рисунок 7]:

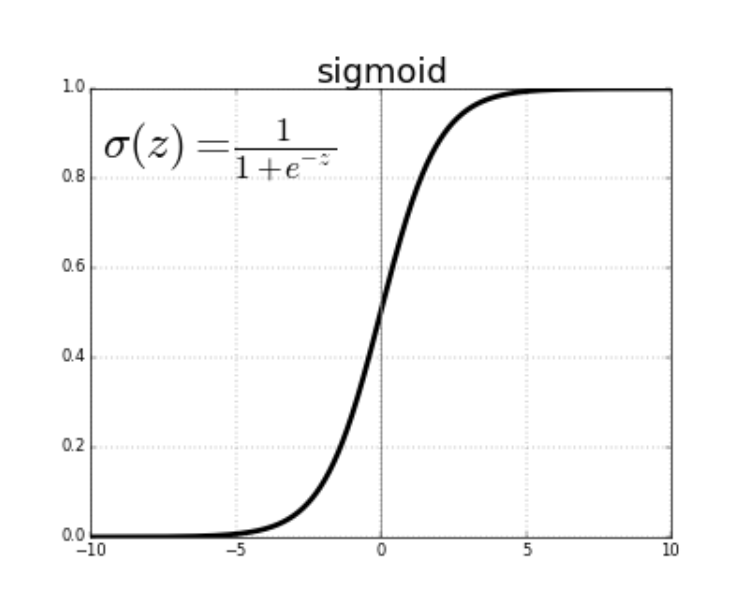


Рисунок – Сигмоидная функция

Одно из преимуществ сигмоидной функции перед пороговой функцией заключается в том, что ее кривая является гладкой. Это означает, что можно вычислить производные в любой точке кривой.

Другое преимущество сигмоиды над линейной функцией заключается в том, что она в отличие от, например, линейной функции имеет ограниченный диапазон значений — [0,1]. Такое свойство сигмоиды очень полезно, так как не приводит к ошибкам в случае больших значений активации.

Сегодня сигмоида является одной из самых частых активационных функций в нейросетях.

* + - 1. Функция ReLU

ReLU не обладает таким свойством гладкости, как сигмоидная функция из предыдущего раздела. Однако она по-прежнему очень популярна в области глубокого обучения. Функция определяется следующим образом [Рисунок 8]:

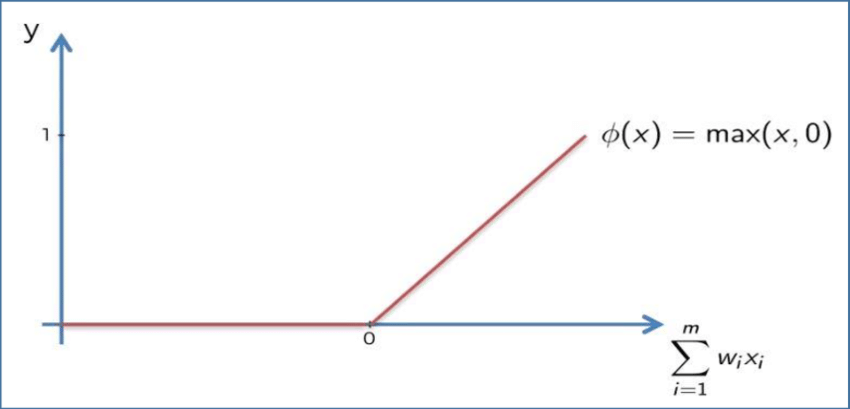


Рисунок – Функция ReLU

ReLu менее требовательно к вычислительным ресурсам, чем гиперболический тангенс или сигмоида, так как производит более простые математические операции.

* + - 1. Функция гиперболического тангенса

Функция гиперболического тангенса – наиболее распространённая из функций активации, основанных на тригонометрических тождествах. Ее математическое определение приведено ниже [Рисунок 9]:

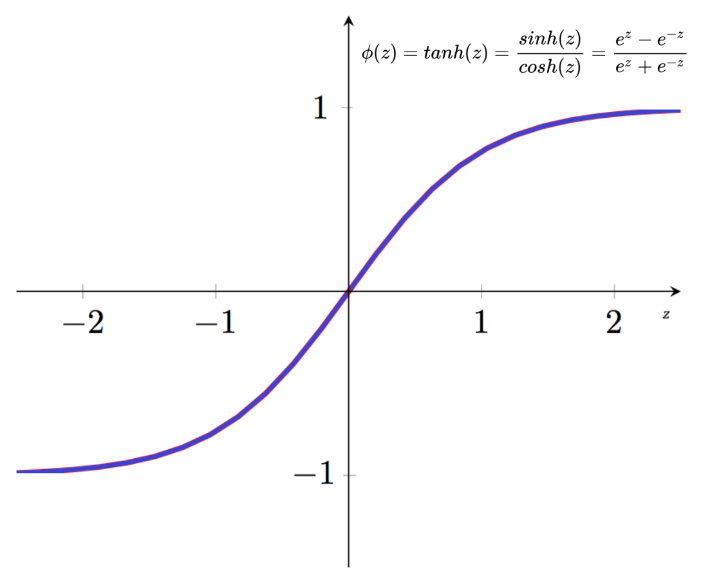


Рисунок – Функция гиперболического тангенса

Функция гиперболического тангенса похожа по внешнему виду на сигмоидную функцию, но все ее выходные значения сдвинуты вниз. И действительно, это скорректированная сигмоидная функция.

Поэтому такая функция имеет те же характеристики, что и у сигмоиды, рассмотренной ранее.

Тангенс также является очень популярной и используемой активационной функцией.

* 1. Выводы

В этом разделе были приведены понятия временного ряда, искусственной нейронной сети, искусственного нейрона, функции активации. Были рассмотрены базовые принципы работы нейронной сети, основные виды функций активации.

1. Описание программы
   1. Общая характеристика программы

Программа позволяет обучить нейронную сеть интервальному прогнозированию частного многомерного временного ряда, с помощью набора обучающих данных, представляющего собой последовательность значений величин от которых зависит прогнозируемая величина и самой прогнозируемой величины в конкретные моменты времени, представленных в виде файла .csv. После обучения на входных данных, программу можно использовать для интервального прогнозирования значений этого ряда в заданные интервалы времени.

Для реализации программы был выбран язык Python3, так как он имеет большое количество библиотек для работы с нейронными сетями.

Программа выполнена в виде консольного приложения на платформе Windows.

Минимальные системные требования к компьютеру:

* операционная система Windows 7, 8, 8.1, 10 (x64);
* процессор с тактовой частотой не ниже 1 ГГц. Рекомендуется использовать как минимум двухъядерный процессор;
* оперативная память не менее 2 ГБ;
* свободное пространство на жестком диске не менее 512 МБ.

Несмотря на небольшие минимальные требования, для получения наиболее точных вычислений требуется выполнение большого количества операций, поэтому для получения точных результатов, рекомендуется использовать систему с процессором частотой от 3ГГц и количеством ядер от 4.

* 1. Структура программы и данных

Ввиду того, что работа большей части кода программы основана на взаимодействии с внешними библиотеками и не требует создания отдельных классов для корректной работы, для решения поставленной задачи в рамках курсовой работы был выбран структурный подход.

Были разработаны функции, для обработки данных и приведения их в пригодный к использованию для обучения нейронной сети формат, а также функции интерфейса, реализующие непосредственное взаимодействие с пользователем. Их назначение, входные и выходные данные описаны в таблице 1.

Таблица 1 – Описание функций

|  |  |  |  |
| --- | --- | --- | --- |
| Название | Назначение | Входные данные | Результат работы |
| multivariate\_data | создание списка фрагментов(подинтервалов) переданного набора данных с заданными параметрами | набор данных;  прогнозируемый параметр;  индекс начала первого интервала;  индекс начала последнего интервала;  количество записей, по которым выполняется прогноз;  длина прогнозируемого интервала;  шаг, с которым будут взяты значения предыдущих записей. | возвращает наборы данных Х и У в формате, пригодном для обучения модели |
| create\_time\_steps | создание отметок оси абсцисс на графике прогноза | длина интервала | возвращает  список численных значений для оси абсцисс |
| multi\_step\_plot | построение графика прогноза | количество записей, по которым выполняется прогноз;  длина прогнозируемого интервала;  прогноз;  индекс прогнозируемого параметра в списке параметров;  шаг, с которым были взяты значения предыдущих записей. | строит график прогноза и реальных значений |
| plot\_train\_history | построение графиков потерь при обучении | список значений потерь при обучении и проверке;  название графика. | строит график потерь при обучении и проверке |

Продолжение таблицы 1

|  |  |  |  |
| --- | --- | --- | --- |
| prepare\_for\_training | подготовка данных для обучения новой модели, создание новой модели | количество записей, по которым выполняется прогноз;  длина прогнозируемого интервала;  путь к файлу с данными;  учитываемые параметры;  прогнозируемый параметр;  количество записей используемых для обучения;  шаг, с которым будут взяты значения предыдущих записей;  размер буфера для перемешивания интервалов;  размер одного пакета интервалов. | возвращает модель нейронной сети;  набор данных для обучения;  набор данных для проверки;  полные наборы значений X и Y; средние значения набора данных;  среднее отклонение величин в наборе данных |
| status | Отображение параметров текущей модели | путь к файлу с данными;  путь к файлу с моделью;  путь к файлу с весами;  количество записей, по которым выполняется прогноз;  длина прогнозируемого интервала. | выводит в консоль полученные значения |
| predict | Выполнение прогноза | модель нейронной сети;  список предыдущих значений для прогнозируемого интервала. | возвращает список спрогнозированных значений |
| save | Сохранение модели и весов нейронной сети | путь к файлам для сохранения модели и весов.  модель. | сохраняет параметры модели в заданные файлы |

Окончание таблицы 1

|  |  |  |  |
| --- | --- | --- | --- |
| load | Загрузка модели и весов нейронной сети из файлов | путь к файлу с моделью;  путь к файлу с весами. | загружает модель и веса из указанных файлов |
| train | Инициализация обучения модели нейронной сети | модель;  набор данных для обучения;  набор данных для проверки;  количество эпох обучения;  количество шагов в одной эпохе;  количество шагов при проверке. | инициализирует обучение модели нейронной сети с полученными параметрами |
| defaultLoad (не используется в финальной версии приложения) | Загрузка модели по умолчанию | - | загружает модель и параметры по умолчанию |

* 1. Интерфейс программы

Программа выполнена в виде консольного приложения.

После запуска программы, посредством открытия .exe файла, пользователь должен инициализировать обучение новой модели нейронной сети (Train - опция 3), либо загрузить существующую модель из файла (Load - опция 5), путём ввода всех запрашиваемых программой значений.

После того как модель будет обучена/загружена, пользователь может сгенерировать прогноз параметра, прогнозированию которого обучалась модель для определённого интервала времени. Для этого нужно выбрать Predict (опция 2), найти в таблице с данными время начала этого интервала, и передать программе соответствующий этому времени номер строки (не считая строки с названиями). После этого программа в новом окне отобразит график предсказания и график реальных значений величины.

Пользователь может узнать, из каких файлов загружена модель и её веса, а также значения её параметров с помощью Status (опция 1).

Для того чтобы сохранить модель и веса в файлы, пользователь должен выбрать Save (опция 4).

Для выхода из программы пользователь должен выбрать Exit (опция 6).

Примечание: при слишком больших значениях параметров обучения модели (кроме шагов), время обучения может оказаться неприемлемо большим (примерное время до окончания отображается во время обучения). В этом случае программу следует перезапустить.

Возможности работы с программой представлены в виде диаграммы [Рисунок 10].

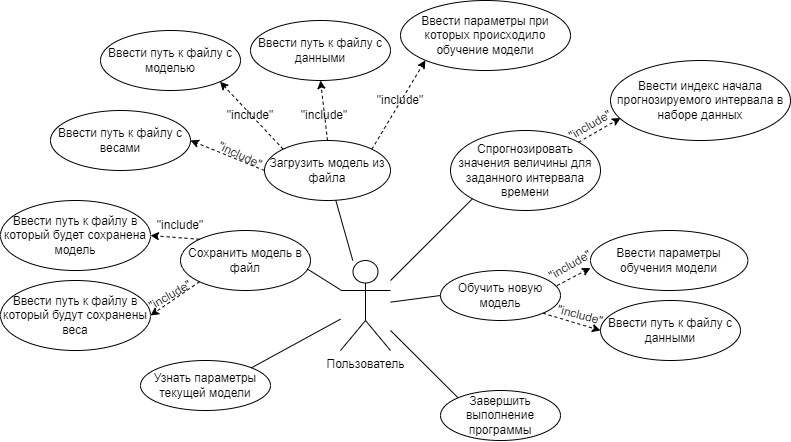


Рисунок – UML диаграмма вариантов использования программы

* 1. Тестирование программы

Ввиду того, что значительная часть кода находится в импортируемых библиотеках, в частности в TensorFlow, и проверка каждого отдельного элемента не представляется возможной, для тестирования программы был выбран метод чёрного ящика.

Будет протестирована каждая опция пользовательского меню, с разными входными данными (там, где это возможно).

Начнём с опции Status. Результат выбора опции без предварительной загрузки данных [Рисунок 11].

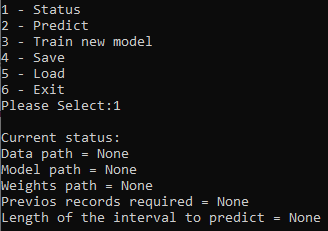


Рисунок – Status без предварительной загрузки данных

Результат выбора опции с предварительно загруженными данными [Рисунок 12].

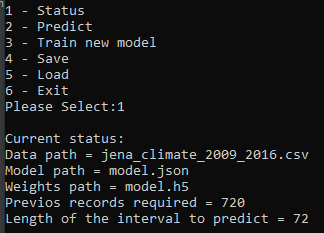


Рисунок – Status с предварительной загрузкой данных

Далее рассмотрим опцию Predict.

Выбор при отсутствии активной модели [Рисунок 13].



Рисунок – Predict при выборе в отсутствие активной модели

Загрузим заранее созданную модель.

Ввод некорректной границы интервала [Рисунок 14].

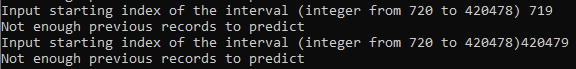


Рисунок – Predict при вводе некорректной границы интервала

Ввод корректной границы интервала [Рисунок 15], [Рисунок 16].



Рисунок – работа Predict при вводе корректной границы интервала

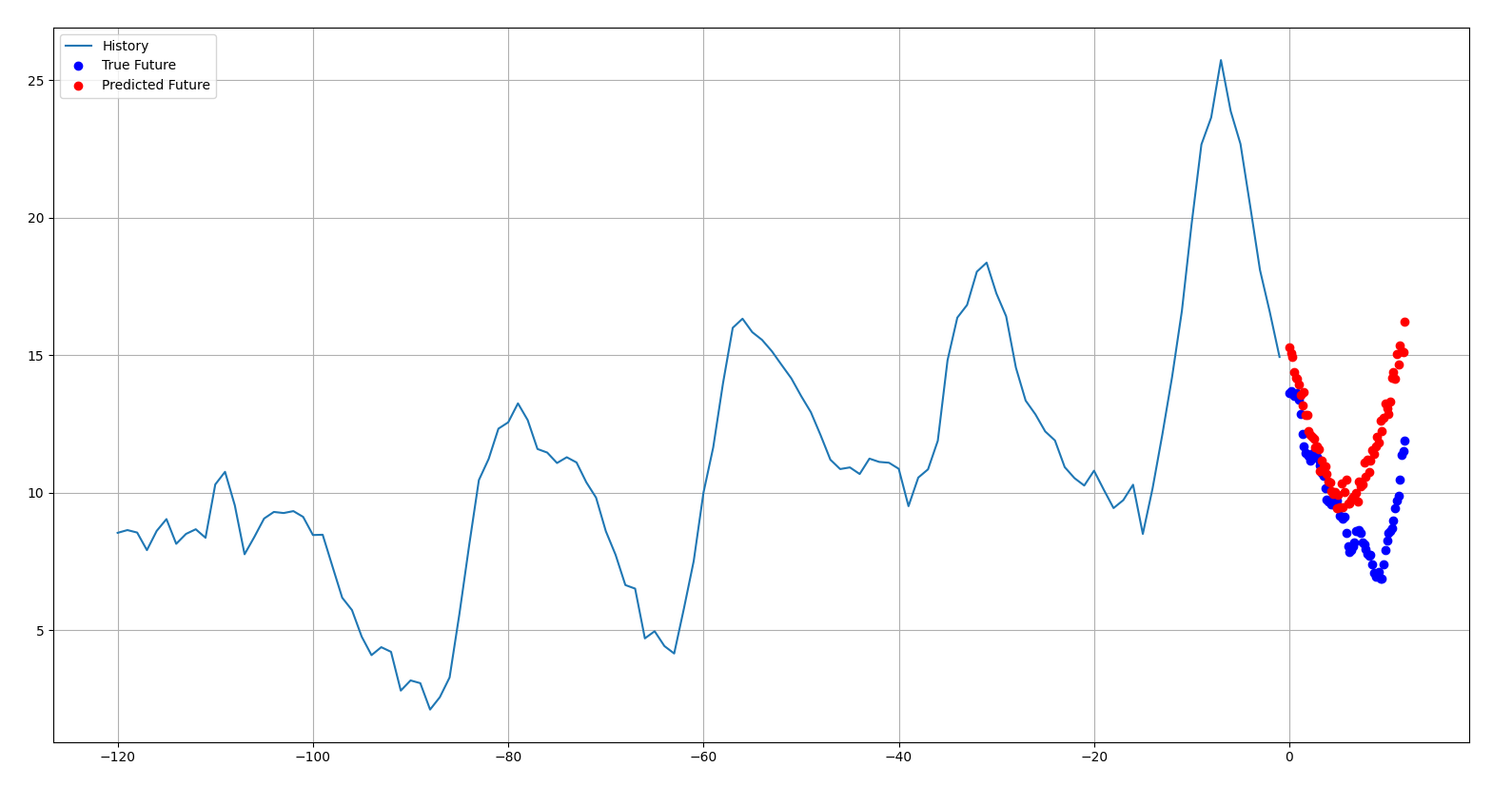


Рисунок – график Predict при вводе корректной границы интервала

Перейдём к опции *Train new model.* Чтобы проверить, что она исправно работает с различными входными данными, оубчим две модели с разными параметрами [Рисунок 17], [Рисунок 18].

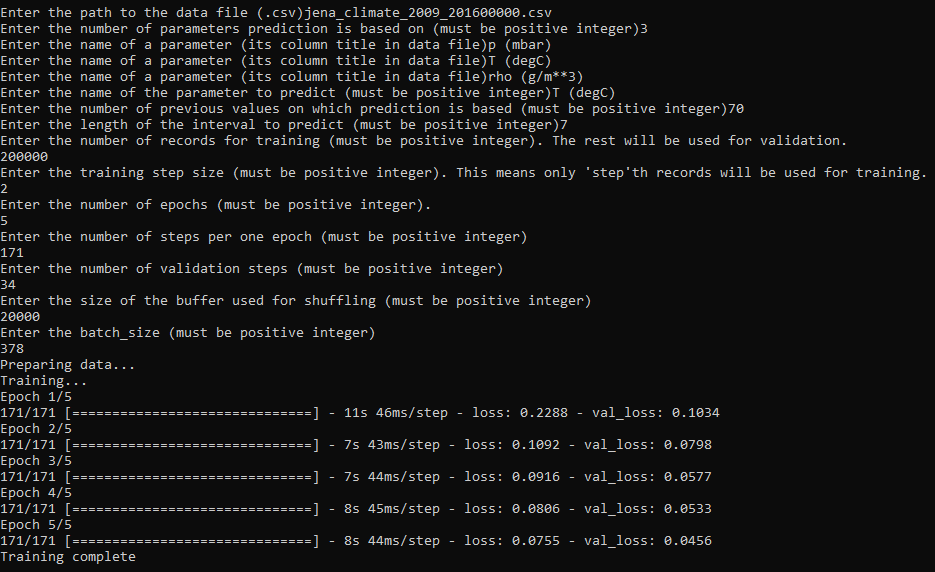


Рисунок – Тестовая модель 1

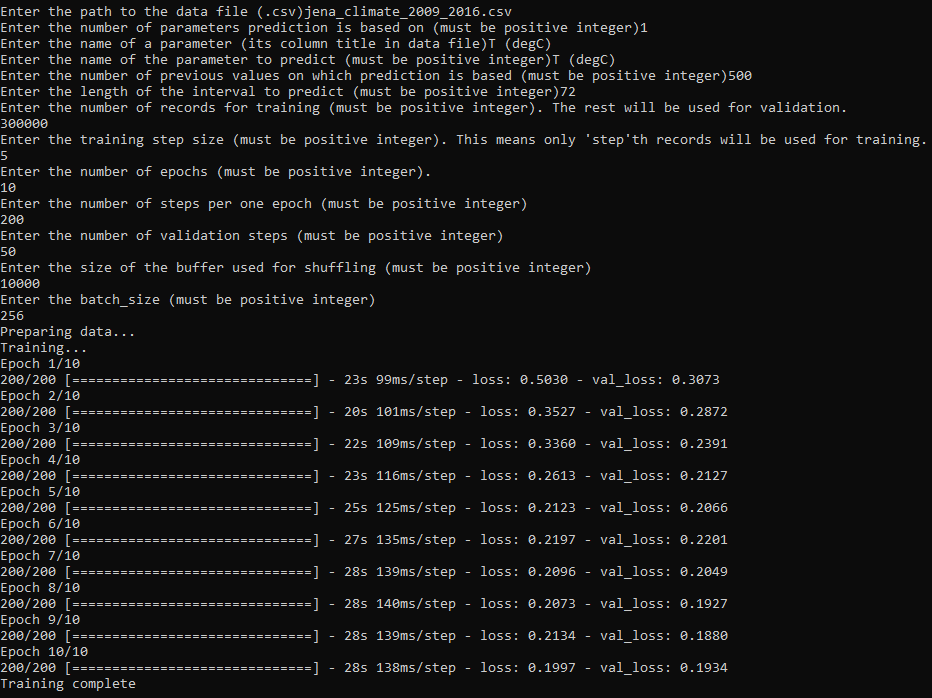


Рисунок – Тестовая модель 2

Протестируем опции Save и Load. Сохраним последнюю полученную модель [Рисунок 19],[Рисунок 20].

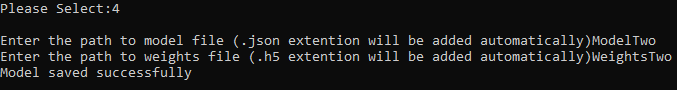


Рисунок – Работа опции Save

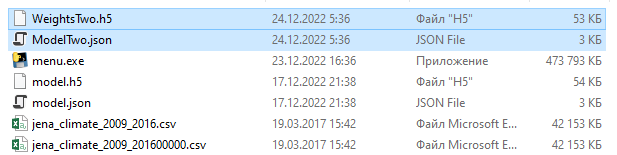


Рисунок – Файлы, созданные при выборе опции Save

Перезапустим программу и загрузим сохранённую модель с помощью опции Load [Рисунок 21].

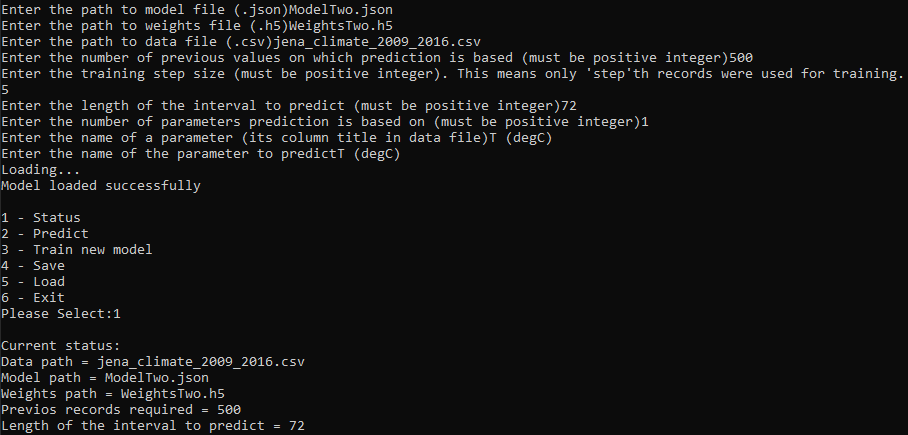


Рисунок – Работа опции Load

Проверим работу опции Exit. Программа завершает своё выполнение [Рисунок 22].

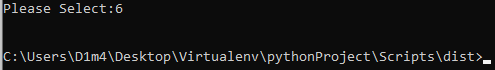


Рисунок – Работа опции Exit

* 1. Выводы

В данной главе было приведено подробное описание разработанной программы: общая характеристика, системные требования, структура, пользовательский интерфейс.

Помимо этого, были рассмотрены результаты тестирования, которые подтверждают, что элементы программы работают верно.

1. Эксплуатация программы
   1. Определение требований к прогнозу

В рамках данной работы, в качестве прогнозируемого временного ряда была выбрана температура воздуха в немецком городе Йена. В качестве источника входных данных будут использоваться временные последовательности данных о погоде, записанные на гидрометеорологической станции в Институте биогеохимии им. Макса Планка с 2009 по 2016 год.

Цель: обучить модель нейронной сети, которая сможет прогнозировать температуру воздуха на 12 часов вперёд с момента последней записи из интервала входных значений с точностью, допускающей максимальное значение функции потерь не более 0,18.

Так как интервал между записями 10 минут, для прогнозирования на 12 часов вперёд нам потребуется выполнить 72 точечных прогноза. Для удобства отображения выберем шаг обучения 6, чтобы тренировочные данные стали почасовыми.

* 1. Опыты

Опыт 1

Для начала попробуем обучить модель прогнозированию температуры только по предыдущим значениям температуры, т.е. наш ряд будет одномерным [Рисунок 23]. Количество предыдущих значений для прогноза установим равным 1000.

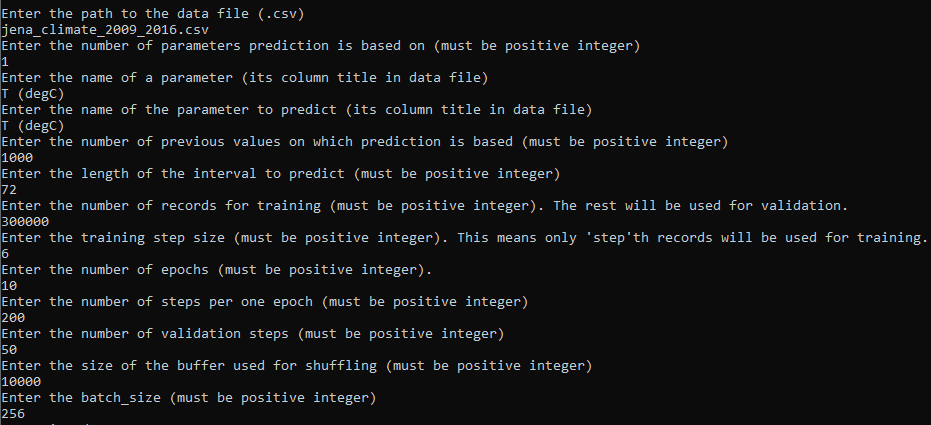


Рисунок – Входные данные обучения первой модели

Функция потерь при обучении на первых эпохах обучения была крайне велика, поэтому для сохранения информативности графика будем отображать её значения начиная с конца 3 эпохи [Рисунок 24],[Рисунок 25].

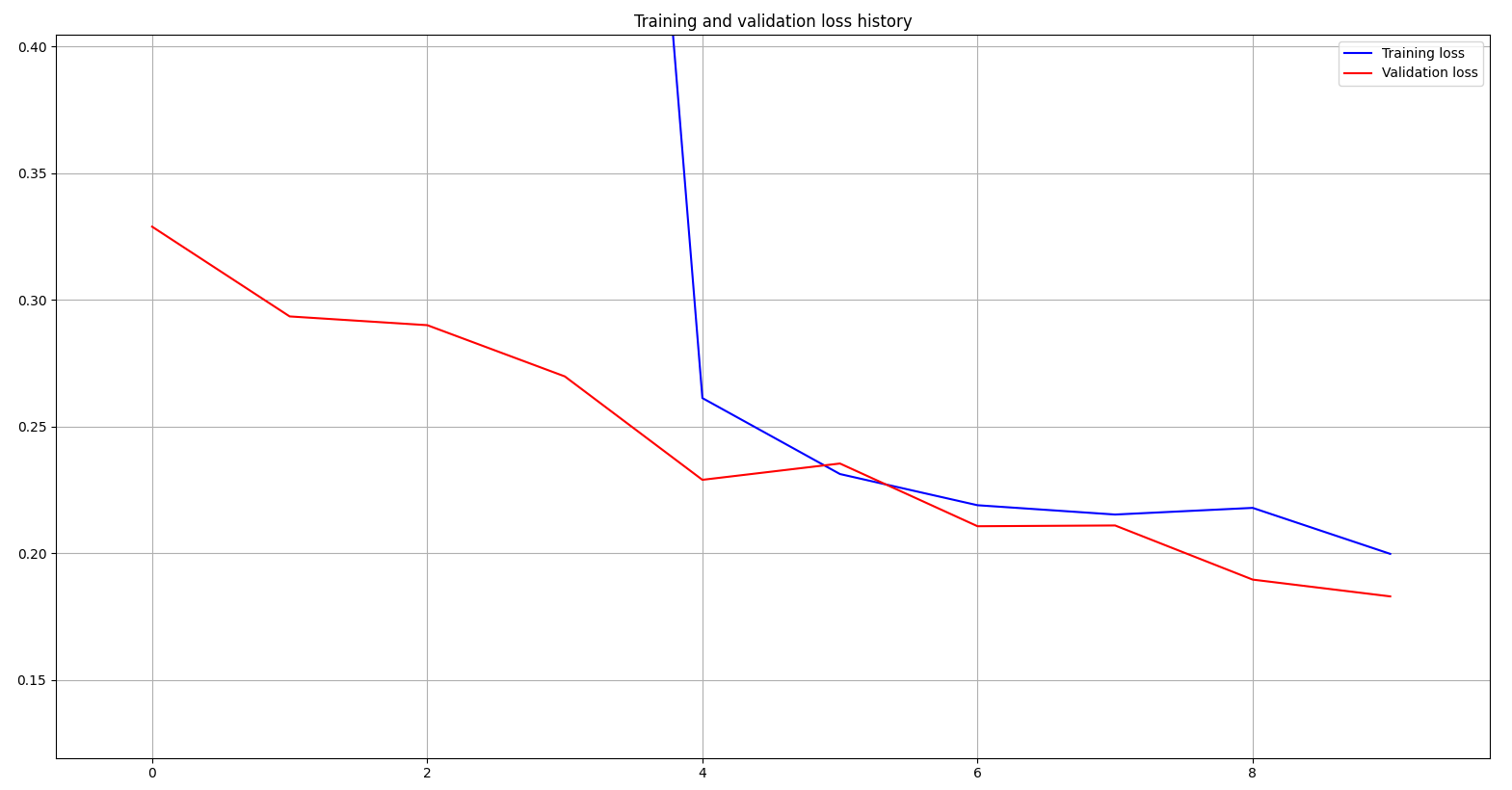


Рисунок – Графики потерь при обучении первой модели

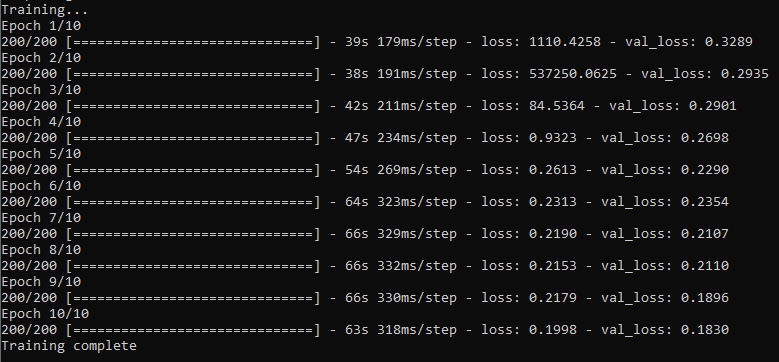


Рисунок – Числовые значения потерь при обучении первой модели

Функция потерь после обучения стала равна 0,1830. Это близкий к нужному результат, но он больше 0,18, поэтому мы продолжим опыты.

Опыт 2

Теперь попробуем, строить наши прогнозы основываясь на трёх параметрах: температуре, давлении и абсолютной влажности воздуха. Увеличим количество шагов в одной эпохе до 300, чтобы сделать прогнозы более точными, но чтобы время обучения не стало слишком долгим, уменьшим количество предыдущих записей для прогноза до 360 [Рисунок 26].

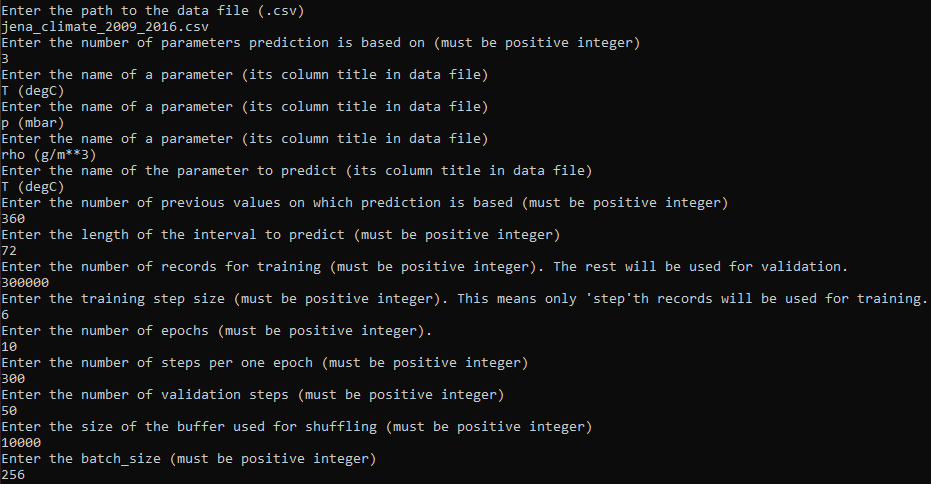


Рисунок - Входные данные обучения второй модели

Потери при обучении [Рисунок 27], [Рисунок 28]

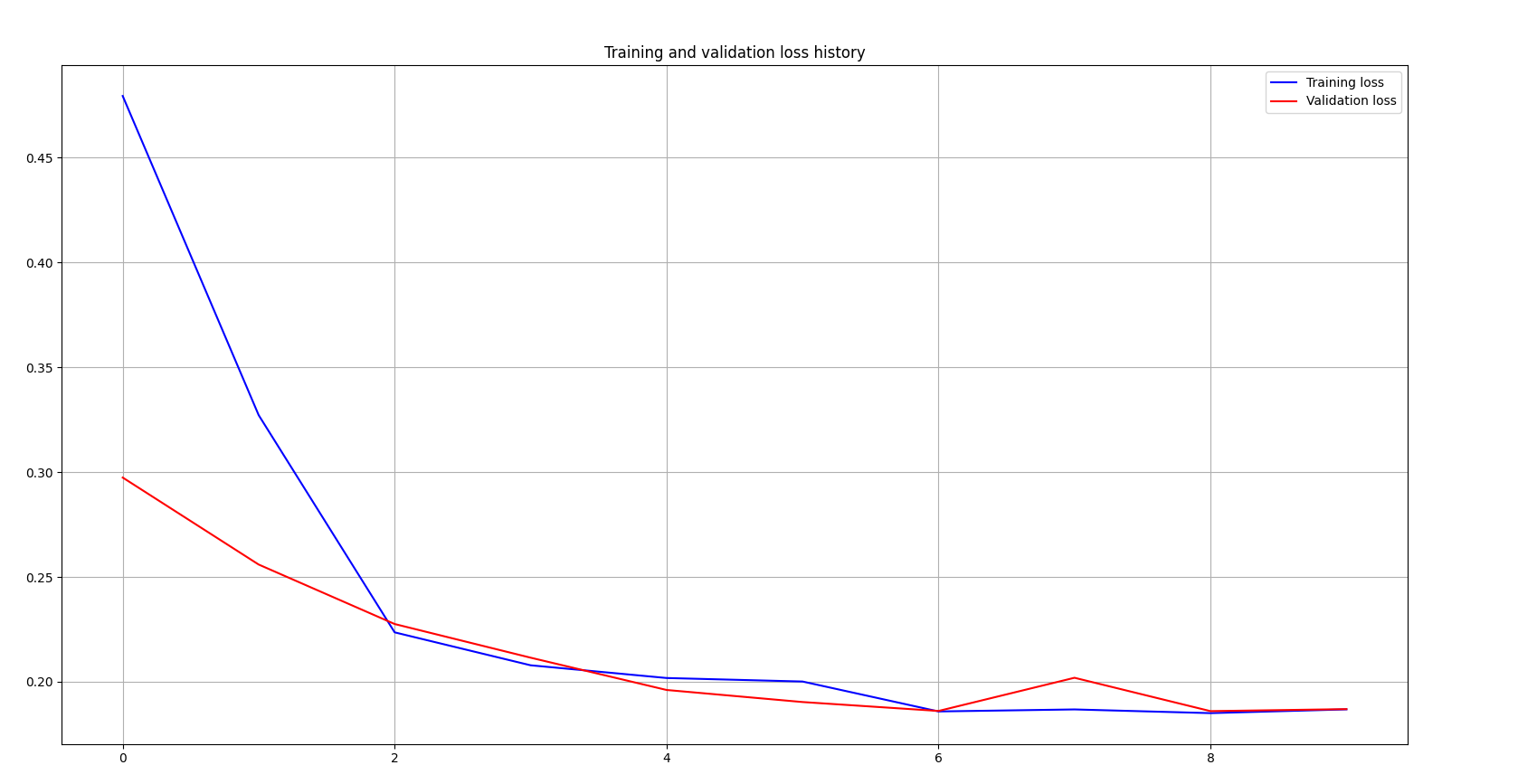


Рисунок - Графики потерь при обучении второй модели

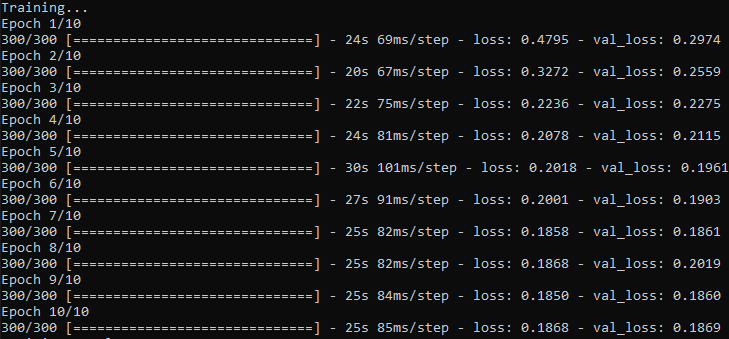


Рисунок – Числовые значения потерь при обучении второй модели

Функция потерь после обучения стала равна 0,1869. Этот результат хуже предыдущего, но следует учитывать, что время обучения модели уменьшилось примерно в 2,5 раза, а количество требуемых предыдущих записей уменьшилось почти в 3 раза.

Опыт 3

Прошлые опыт показали, что входные параметры влияют на время обучения сильно, но не критически, поэтому мы установим большие значения для достижения требуемой точности. Увеличим количество предыдущих записей до 1080, количество шагов в эпохе - до 500, размер пакетов с интервалами значений, используемых при обучении - до 512 [Рисунок 29], [Рисунок 30], [Рисунок 31].

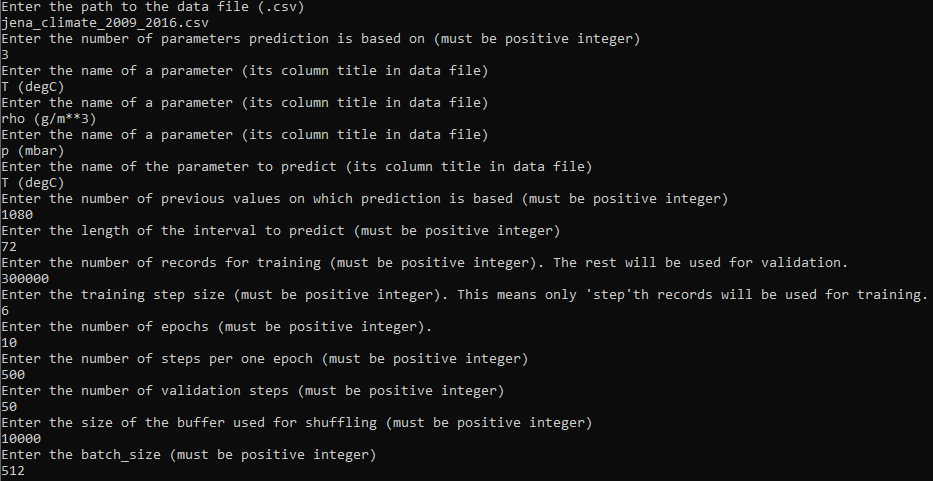


Рисунок - Входные данные обучения третьей модели

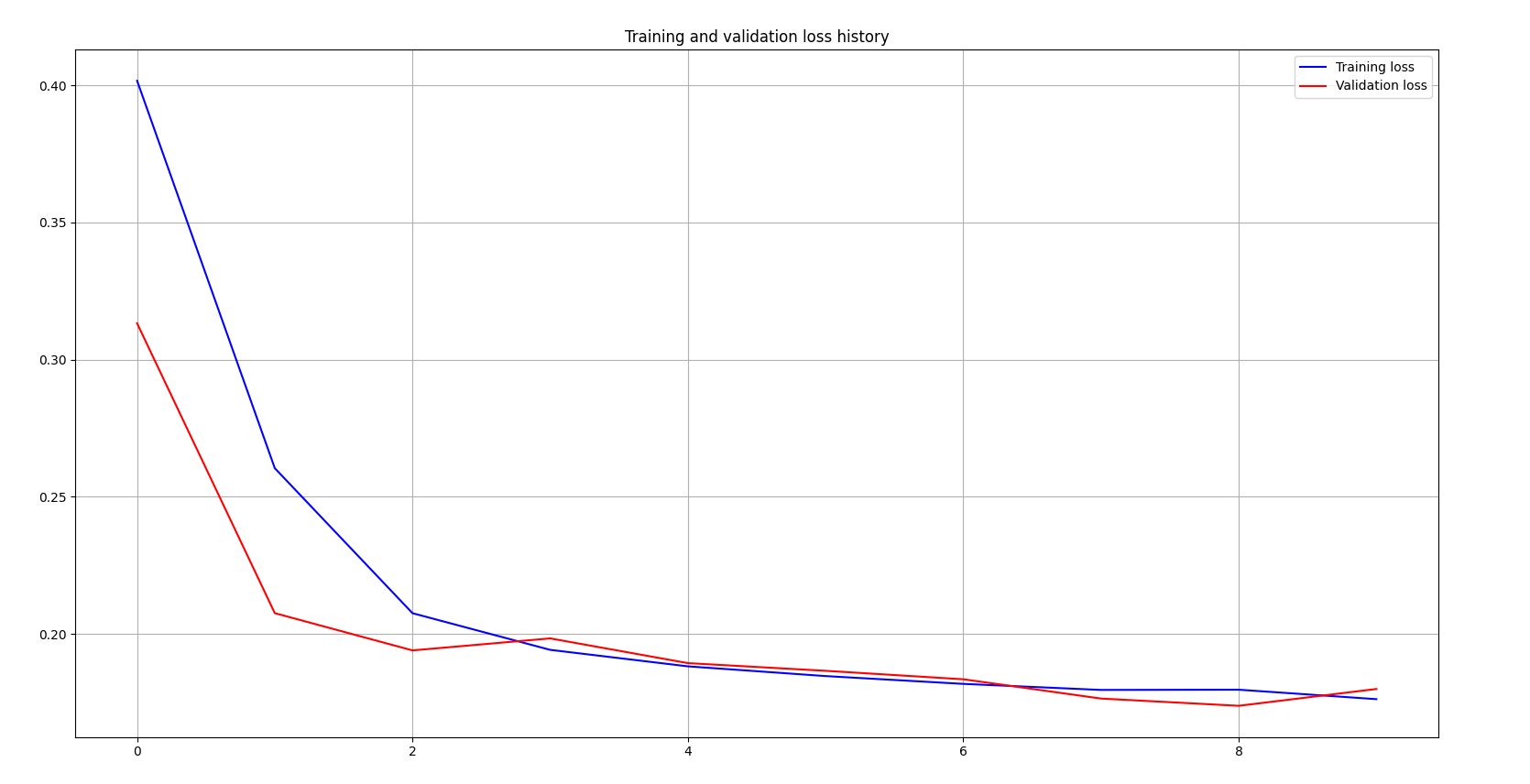


Рисунок - Графики потерь при обучении третьей модели

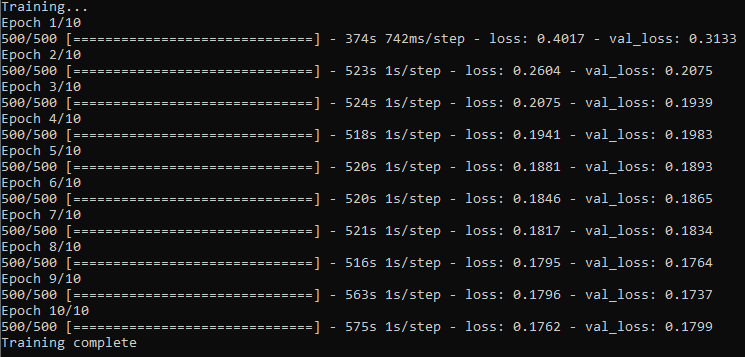


Рисунок – Числовые значения потерь при обучении третьей модели

Функция потерь после обучения стала равна 0,1799. Это значение удовлетворяет нашим требованиям. Следует отметить, что время обучения модели возросло примерно в 20 раз по сравнению с предыдущим опытом и, вероятно, существует иной набор входных параметров, который позволил бы достичь заданной точности более быстро.

* 1. Проверка модели

Для того, чтобы эмпирически убедиться в верности прогнозов модели, сделаем несколько прогнозов для разных интервалов данных [Рисунок 32], [Рисунок 33], [Рисунок 34], [Рисунок 35], [Рисунок 36]

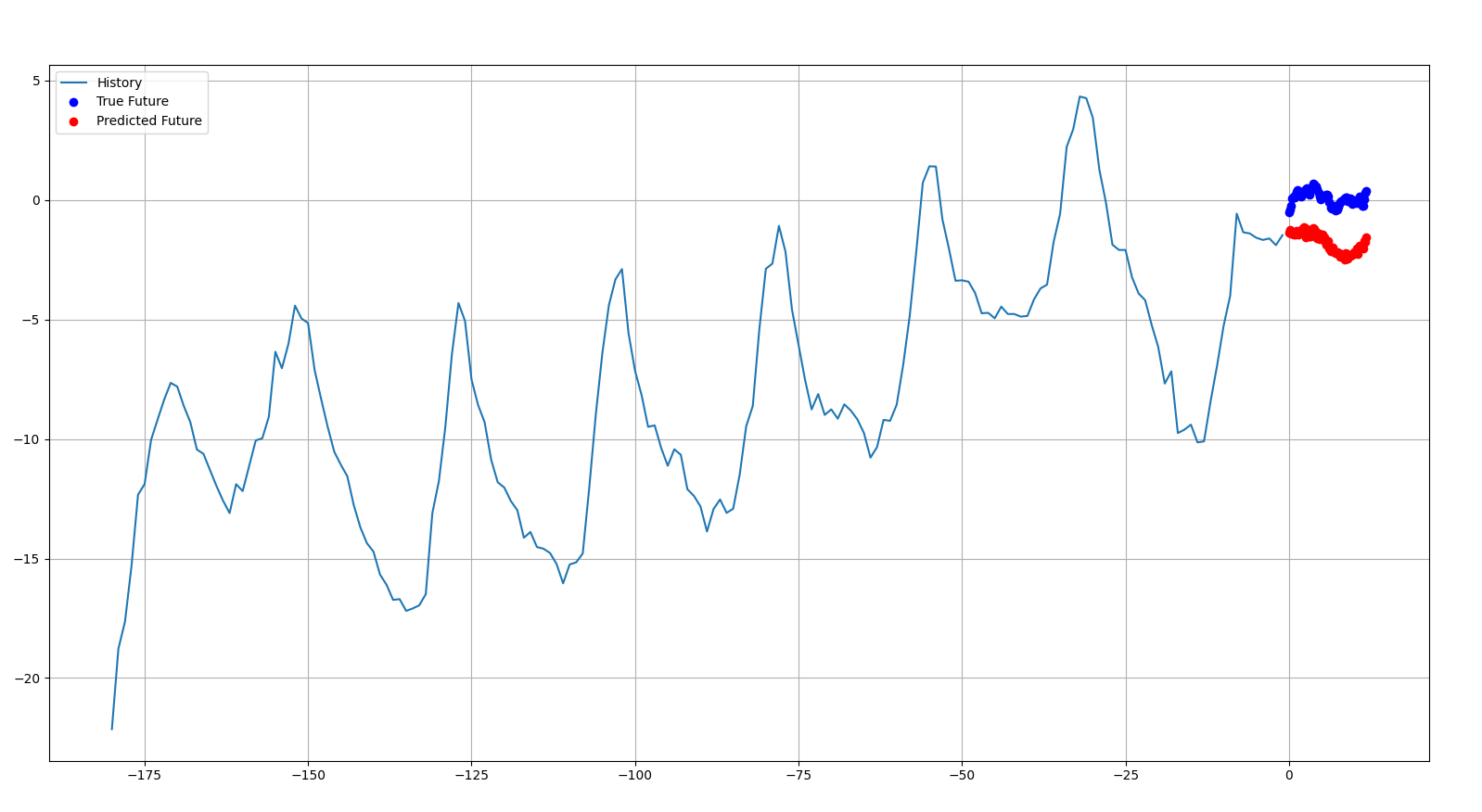


Рисунок – График реальных значений температуры (синий и голубой) и прогноза модели (красный) для 12-часового интервала начиная с записи 2000

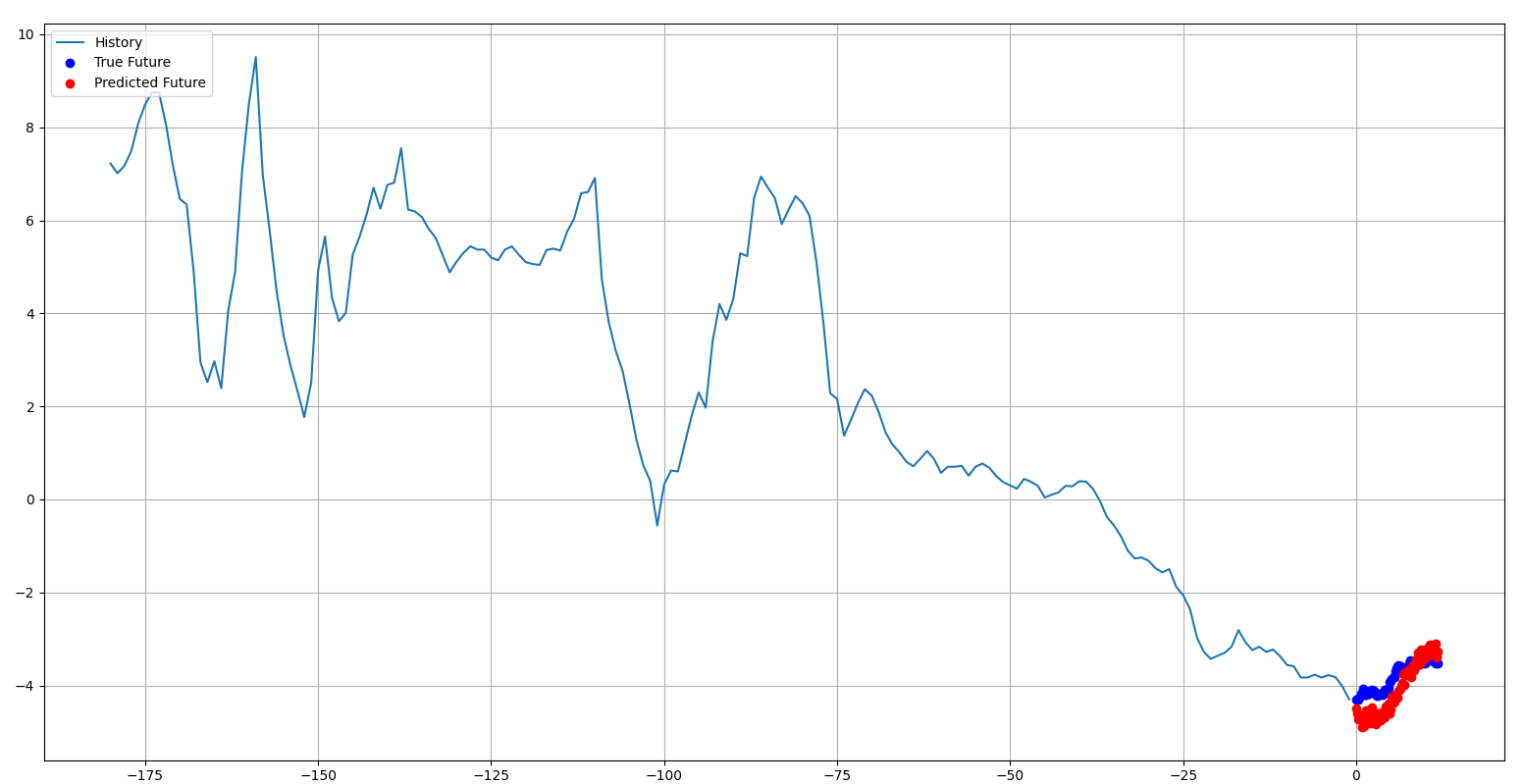


Рисунок – График реальных значений температуры (синий и голубой) и прогноза модели (красный) для 12-часового интервала начиная с записи 50000

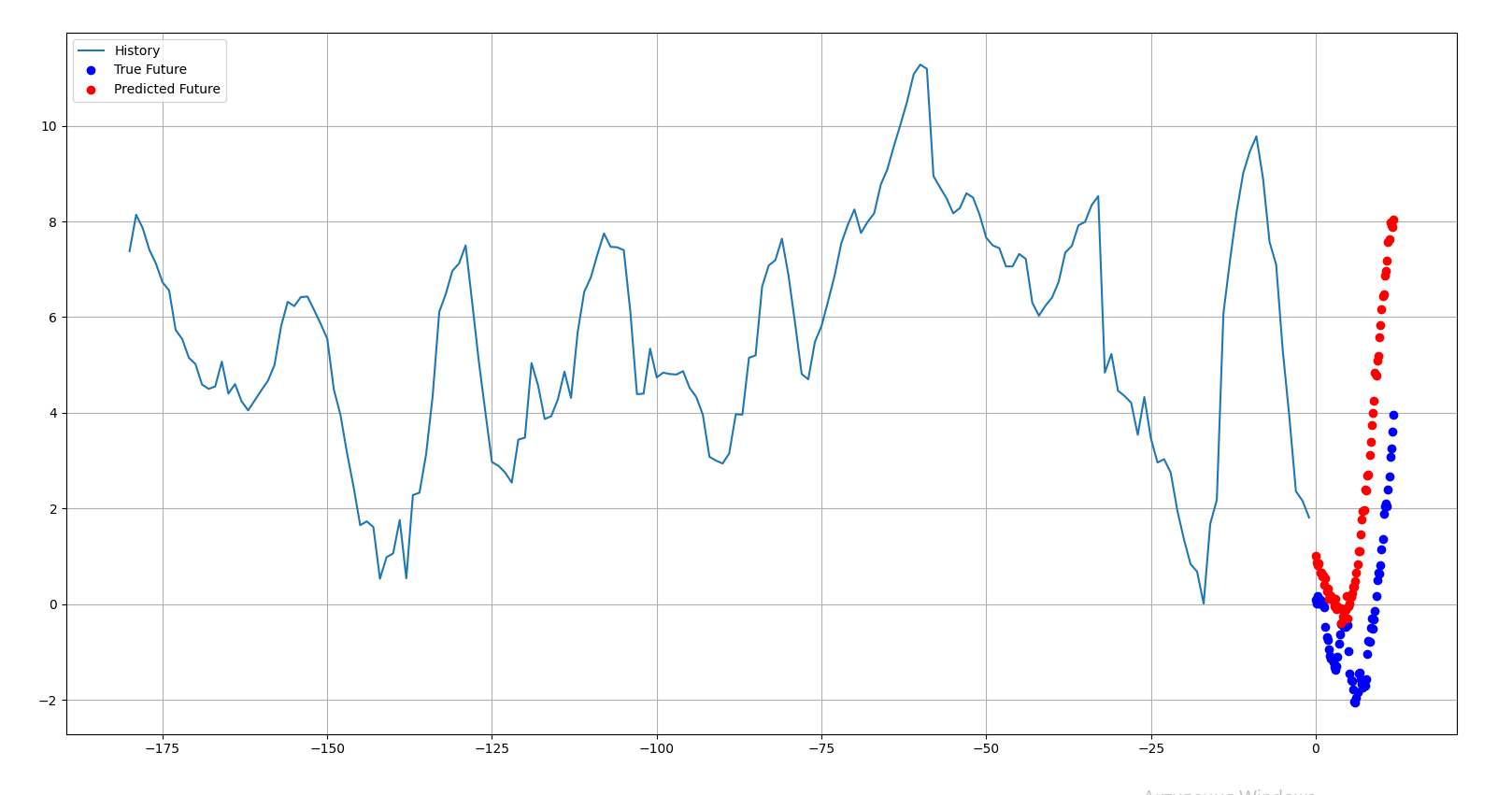


Рисунок – График реальных значений температуры (синий и голубой) и прогноза модели (красный) для 12-часового интервала начиная с записи 270000

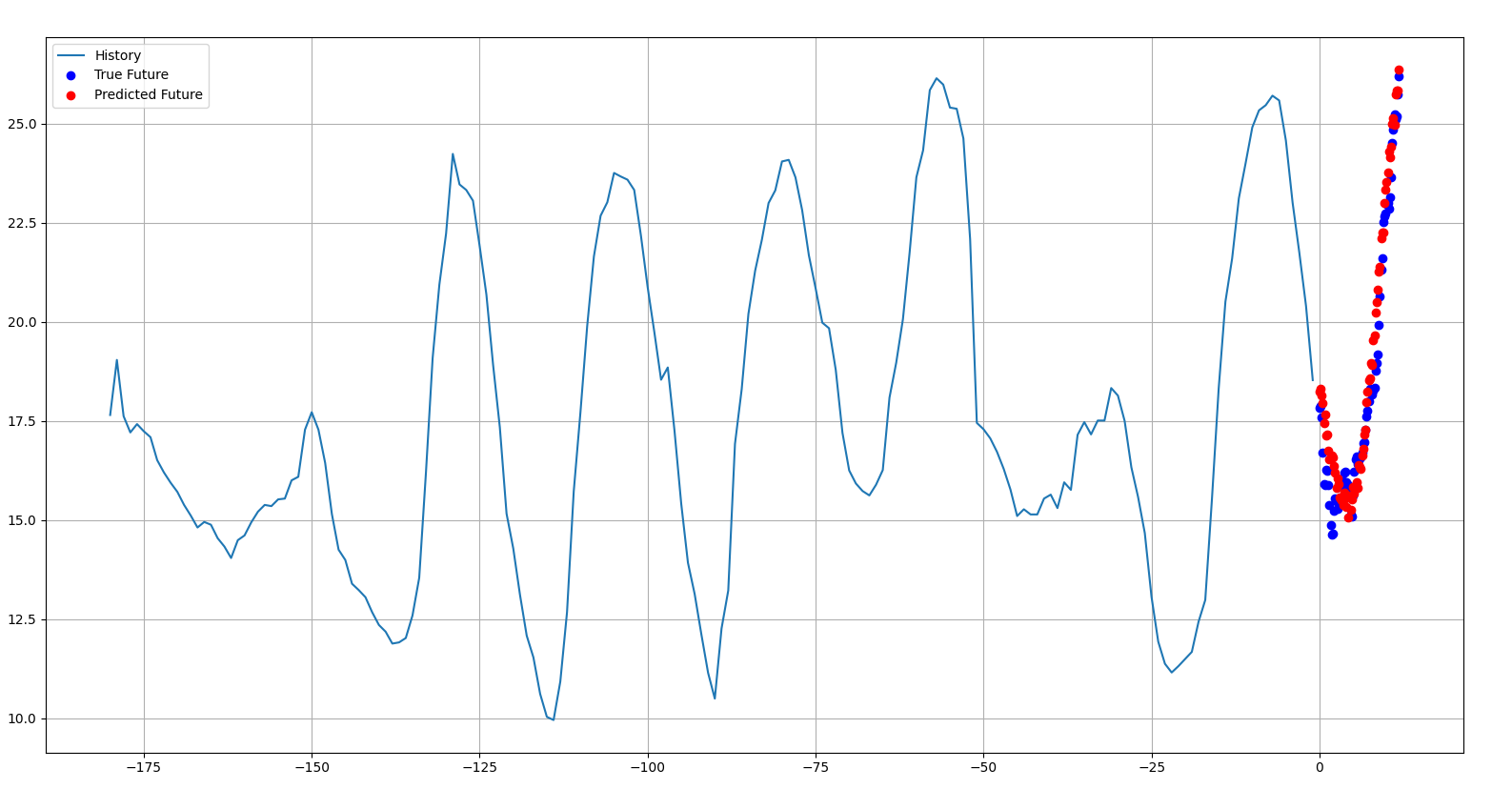


Рисунок – График реальных значений температуры (синий и голубой) и прогноза модели (красный) для 12-часового интервала начиная с записи 350000

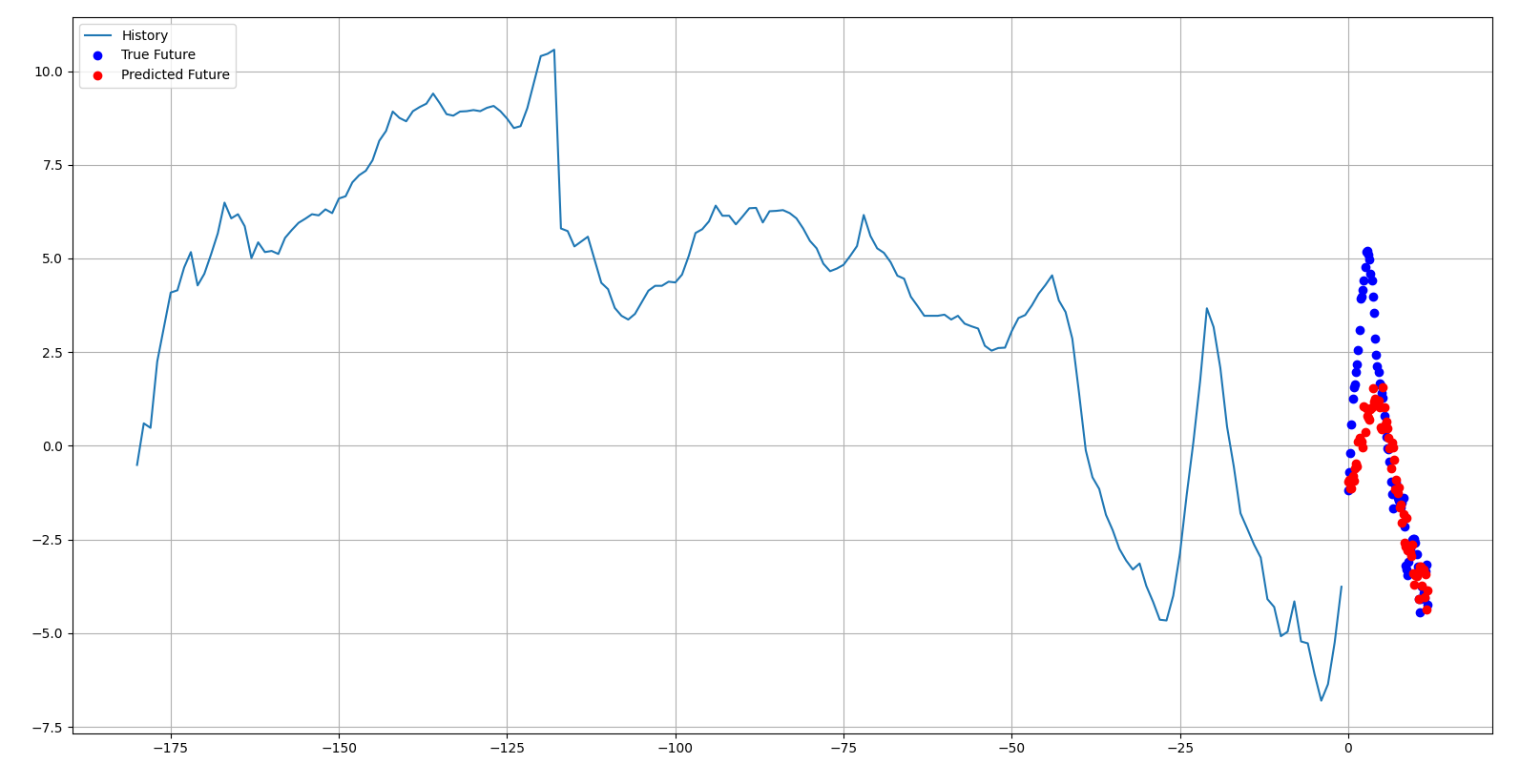


Рисунок – График реальных значений температуры (синий и голубой) и прогноза модели (красный) для 12-часового интервала начиная с записи 420478

Как можно видеть, предсказания модели не всегда оказываются точными, но, в большинстве случаев, достаточно близки к истине, что позволяет сделать вывод о том, что модель работает верно.

* 1. Выводы

В данном разделе был на рассмотрен порядок эксплуатации программы на примере обучения модели нейронной сети для прогнозирования температуры воздуха на 12 часов вперёд, по метеоданным предыдущих дней с заданной точностью. Были сформулированы требования к модели, проведён ряд опытов и проверка полученного результата. Цель, поставленная в начале данного раздела, была достигнута.

Заключение

В ходе написания курсовой работы мы познакомились с понятиями нейронной сети и временного ряда, их основными разновидностями.

С использованием библиотек TensorFlow была разработана и протестирована программа, выполняющая обучение моделей нейронных сетей с заданными параметрами для интервального прогнозирования произвольных многомерных временных рядов.

Была составлена пояснительная записка с общей характеристикой, системными требованиями и подробным описанием структуры, вариантов использования и порядка эксплуатации программы.

Разработанная программа была использована для прогнозирования частного временного ряда, на примере прогнозирования температуры воздуха в немецком городе Йена по метеоданным записанным на гидрометеорологической станции в Институте биогеохимии им. Макса Планка с 2009 по 2016 год. Были построены графики прогнозов обученной модели для визуализации результата.

Все поставленные теоретические и практические вопросы были проработаны, все поставленные цели достигнуты.

Список использованных источников

1. FREECODECAMP.ORG: сайт – Deep Learning Neural Networks Explained in Plain English / freeCodeCamp – URL: <https://www.freecodecamp.org/news/deep-learning-neural-networks-explained-in-plain-english/> (дата обращения: 20.09.2022) – Текст и изображения: электронные.

2. NEUROHIVE.IO: сайт – Функции активации нейросети: сигмоида, линейная, ступенчатая, ReLu, than / Neurohive – URL: <https://neurohive.io/ru/osnovy-data-science/activation-functions/> (дата обращения: 21.09.2022) – Текст и изображения: электронные.

3. HABR.COM: сайт – Прогнозирование временных рядов с помощью рекуррентных неройнных сетей / Хабр – URL: <https://habr.com/ru/post/495884/>

(дата обращения: 4.11.2022) – Текст и изображения: электронные.

4. HABR.COM: сайт – Использование диаграммы вариантов использования UML при проектировании программного обеспечения / Хабр – URL: <https://habr.com/ru/post/566218/> (дата обращения: 6.11.2022) – Текст и изображения: электронные.

Приложение А - Листинг программы

Содержимое файла ***funcs.py:***

**import** os *# Библиотека для работы с операционной системой*os.environ[**'TF\_CPP\_MIN\_LOG\_LEVEL'**] = **'2'** *# Отключаем лишние информационные сообщения tensorflow***import** tensorflow **as** tf *# Самая важная библиотека с инструментами для создания нейронных сетей***import** matplotlib **as** mpl *# Библиотека для построения графиков***import** matplotlib.pyplot **as** plt  
**import** numpy **as** np *# Библиотека для работы с многомерными массивами***import** pandas **as** pd *# Библиотека для работы с таблицами*mpl.rcParams[**'figure.figsize'**] = (8, 6) *# Задаём параметры окна с графиками*mpl.rcParams[**'axes.grid'**] = **True***#tf.random.set\_seed(13) # Для обеспечения воспроизводимости результатов используется функция set\_seed  
# Задаём переменным значения по умолчанию*model = **None**model\_path = **None**data\_path = **None**weights\_path = **None**x = **None**y = **None***# STEP = 6*past\_history = **None**future\_target = **None  
  
  
def** multivariate\_data(dataset, target, start\_index, end\_index, history\_size,  
 *# Функция для создания списка фрагментов(подинтервалов) переданного набора данных с заданными параметрами* target\_size, step):  
 data = []  
 labels = []  
  
 start\_index = start\_index + history\_size  
 **if** end\_index **is None**:  
 end\_index = len(dataset) - target\_size  
  
 **for** i **in** range(start\_index, end\_index):  
 indices = range(i - history\_size, i, step)  
 data.append(dataset[indices])  
 labels.append(target[i:i + target\_size])  
  
 **return** np.array(data), np.array(labels) *# Фактически мы получили x и y, где x - вектор, y - одиночное значение***def** create\_time\_steps(length): *# Функция для создания отметок оси абсцисс на графике прогноза* **return** list(range(-length, 0))  
  
  
**def** multi\_step\_plot(history, true\_future, prediction, index\_to\_plot, STEP): *# Функция для построения графика прогноза* plt.figure(figsize=(12, 6)) *# Задаём параметры окна с графиком* num\_in = create\_time\_steps(len(history)) *# Абсциссы предыдущих моментов времени* num\_out = len(true\_future) *# Абсциссы предсказываемых моментов времени* plt.plot(num\_in, np.array(history[:, index\_to\_plot]),  
 label=**'History'**) *# Строим график значений по которым генерируется прогноз значения величины* plt.plot(np.arange(num\_out) / STEP, np.array(true\_future), **'bo'**, *# Строим график реальных значений величины* label=**'True Future'**)  
 **if** prediction.any():  
 plt.plot(np.arange(num\_out) / STEP, np.array(prediction), **'ro'**,  
 label=**'Predicted Future'**) *# Строим график спрогнозированных значений* plt.legend(loc=**'upper left'**)  
 plt.show()  
  
  
**def** plot\_train\_history(history, title): *# Функция для построения графиков потерь при обучении* loss = history.history[**'loss'**]  
 val\_loss = history.history[**'val\_loss'**]  
  
 epochs = range(len(loss))  
  
 plt.figure()  
  
 plt.plot(epochs, loss, **'b'**, label=**'Training loss'**)  
 plt.plot(epochs, val\_loss, **'r'**, label=**'Validation loss'**)  
 plt.title(title)  
 plt.legend()  
  
 plt.show()  
  
  
**def** prepare\_for\_training(past\_history, future\_target, data\_path, features\_considered, feature\_to\_predict,  
 *# Функция для подготовки данных к обучению новой модели* TRAIN\_SPLIT, STEP, BUFFER\_SIZE, BATCH\_SIZE):  
 df = pd.read\_csv(data\_path) *# Загружаем данные в таблицу* features = df[features\_considered] *# Выбираем нужные нам столбцы* features.index = df[**'Date Time'**] *# Устанавливаем индексацию по времени* dataset = features.values *# Создаём набор данных с которыми будем работать  
 # Проводим стандартизацию данных* data\_mean = dataset[:TRAIN\_SPLIT].mean(axis=0)  
 data\_std = dataset[:TRAIN\_SPLIT].std(axis=0)  
 print(data\_mean, data\_std)  
 dataset = (dataset - data\_mean) / data\_std  
 print(dataset.mean(axis=0))  
 *# Создаём набор всех возможных последовательных интервалов значений данных всего набора данных с заданными параметрами* x, y = multivariate\_data(dataset, dataset[:, features\_considered.index(feature\_to\_predict)], 0,  
 **None**, past\_history,  
 future\_target, STEP)  
 *# Создаём набор всех возможных последовательных интервалов с начала набора данных до отметки конца тренировочных данных (TRAIN\_SPLIT) с заданными параметрами. Эти данные будут использоваться для обучения.* x\_train, y\_train = multivariate\_data(dataset, dataset[:, features\_considered.index(feature\_to\_predict)], 0,  
 TRAIN\_SPLIT, past\_history,  
 future\_target, STEP)  
 *# Создаём набор всех возможных последовательных интервалов c отметки тренировочных данных (TRAIN\_SPLIT) и до конца набора данных с заданными параметрами. Эти данные будут использоваться для проверки.* x\_validation, y\_validation = multivariate\_data(dataset, dataset[:, features\_considered.index(feature\_to\_predict)],  
 TRAIN\_SPLIT, **None**, past\_history,  
 future\_target, STEP)  
  
 *# Собираем наши наборы интервалов в новые наборы данных, перемешиваем их с помощью shuffle() и объединяем их в пакеты с помощью batch()* train\_data = tf.data.Dataset.from\_tensor\_slices((x\_train, y\_train))  
 train\_data = train\_data.cache().shuffle(BUFFER\_SIZE).batch(BATCH\_SIZE).repeat()  
  
 validation\_data = tf.data.Dataset.from\_tensor\_slices((x\_validation, y\_validation))  
 validation\_data = validation\_data.batch(BATCH\_SIZE).repeat()  
  
 *# Создаём новую модель нейросети с заданными параметрами. В нашем случае она состоит из двух рекуррентных слоёв c долгой краткосрочной памятью (LSTM) и одного выходного слоя.* model = tf.keras.models.Sequential()  
 model.add(tf.keras.layers.LSTM(32,  
 return\_sequences=**True**,  
 input\_shape=x\_train.shape[-2:])) *# Задаём количество нейронов и форму входных данных для первого слоя* model.add(tf.keras.layers.LSTM(16, activation=**'relu'**)) *# Устанавливаем количество нейронов и функцию активации второго слоя* model.add(tf.keras.layers.Dense(  
 future\_target)) *# Устанавливаем количество нейронов выходного слоя равным длине интервала предсказания, т.е. один нейрон отвечает за одну точку* model.compile(optimizer=tf.keras.optimizers.RMSprop(clipvalue=1.0), loss=**'mae'**) *# Компилируем новую модель* **return** model, train\_data, validation\_data, x, y, data\_mean, data\_std  
  
  
**def** status(data\_path, model\_path, weights\_path, past\_history,  
 future\_target): *# Функция для отображения текущих параметров модели* print(**"Data path ="**, data\_path)  
 print(**"Model path ="**, model\_path)  
 print(**"Weights path ="**, weights\_path)  
 print(**"Previos records required ="**, past\_history)  
 print(**"Length of the interval to predict ="**, future\_target)  
  
  
**def** predict(model, data): *# Функция для выполнения прогноза* **return** model.predict(data)[0]  
  
  
**def** save(model\_path, weights\_path, model): *# Функция для сохранения модели и весов нейронной сети* model\_json = model.to\_json()  
 **with** open(model\_path + **".json"**, **"w"**) **as** json\_file:  
 json\_file.write(model\_json)  
 model.save\_weights(weights\_path + **".h5"**)  
  
  
**def** load(model\_path, weights\_path): *# Функция для загрузки модели и весов нейронной сети из файлов* json\_file = open(model\_path, **'r'**)  
 loaded\_model\_json = json\_file.read()  
 json\_file.close()  
 model = tf.keras.models.model\_from\_json(loaded\_model\_json)  
 model.load\_weights(weights\_path)  
 **return** model  
  
  
**def** train(model, train\_data, validation\_data, epochs=10, steps\_per\_epoch=200,  
 validation\_steps=50): *# Функция для инициализации обучения нейронной сети* **return** model.fit(train\_data, epochs=epochs, steps\_per\_epoch=steps\_per\_epoch, validation\_data=validation\_data,  
 validation\_steps=validation\_steps)  
  
  
**def** defaultLoad(): *# Функция для загрузки модели по умолчанию (использовалась при тестировании, чтобы не вводить данные вручную)* print(**"Loading default configuration..."**)  
 model = load(**"model.json"**, **"model.h5"**)  
 df = pd.read\_csv(**"jena\_climate\_2009\_2016.csv"**)  
 features\_considered = [**'p (mbar)'**, **'T (degC)'**, **'rho (g/m\*\*3)'**]  
 features = df[features\_considered]  
 features.index = df[**'Date Time'**]  
 real\_dataset = features.values  
 data\_mean = real\_dataset.mean(axis=0)  
 data\_std = real\_dataset.std(axis=0)  
 dataset = (real\_dataset - data\_mean) / data\_std  
 STEP = 6  
 past\_history = 720  
 future\_target = 72  
 x, y = multivariate\_data(dataset, dataset[:, 1], 0,  
 **None**, past\_history,  
 future\_target, STEP)  
 print(**"Loaded successfully"**)  
 **return** model, x, y, data\_mean, data\_std, **"jena\_climate\_2009\_2016.csv"**, **"model.json"**, **"model.h5"**, 720, 72, [**'p (mbar)'**, **'T (degC)'**, **'rho (g/m\*\*3)'**],**'T (degC)'**, 6

Содержимое файла ***menu.py:***

**from** funcs **import** \*  
  
menu = {} *# Создаём меню для взаимодействия с программой*menu[**'1'**] = **"Status"**menu[**'2'**] = **"Predict"**menu[**'3'**] = **"Train new model"**menu[**'4'**] = **"Save"**menu[**'5'**] = **"Load"**menu[**'6'**] = **"Exit"***# model, x, y, data\_mean, data\_std, data\_path, model\_path, weights\_path, past\_history, future\_target, features\_considered, feature\_to\_predict, STEP = defaultLoad() # Производим загрузку конфигурации по умолчанию***if** \_\_name\_\_ == **'\_\_main\_\_'**:  
 **try**:  
 **while True**:  
 print(**""**)  
 **for** entry **in** menu.items(): *# Выводим все опции меню* print(entry[0], **"-"**, entry[1])  
 selection = input(**"Please Select: "**) *# Считываем выбор пользователя* print(**""**)  
  
 **if** selection == **'1'**: *# Вывод текущего статуса модели* print(**"Current status:"**)  
 status(data\_path, model\_path, weights\_path, past\_history, future\_target)  
  
  
 **elif** selection == **'2'**: *# Прогнозирование заданного интервала* **if** model **is not None**:  
 start = -1  
 **while** start < 0 **or** start >= len(  
 x): *# Проверяем, что нам хватает предыдущих значений для предсказания заданного интервала* start = int(input(  
 **"Input starting index of the interval (integer from "** + str(past\_history) + **" to "** + str(  
 len(x) + past\_history - 1) + **")\n"**)) - past\_history  
 **if** start < 0 **or** start >= len(x):  
 print(**"Not enough previous records to predict"**)  
  
 prediction = predict(model, x[start:start + 1]) *# Выполняем предсказание* multi\_step\_plot(  
 x[start:start + 1][0] \* data\_std[features\_considered.index(feature\_to\_predict)] + data\_mean[  
 features\_considered.index(feature\_to\_predict)],  
 y[start:start + 1][0] \* data\_std[features\_considered.index(feature\_to\_predict)] + data\_mean[  
 features\_considered.index(feature\_to\_predict)],  
 prediction \* data\_std[features\_considered.index(feature\_to\_predict)] + data\_mean[  
 features\_considered.index(feature\_to\_predict)],  
 features\_considered.index(feature\_to\_predict),  
 STEP) *# Строим график предсказания и реальных значений* **else**:  
 print(**"Train or load a model first"**)  
  
  
 **elif** selection == **'3'**: *# Выполняем обучение новой модели  
 # Вводим все необходимые параметры* data\_path = input(**"Enter the path to the data file (.csv)\n"**)  
 n = int(input(**"Enter the number of parameters prediction is based on (must be positive integer)\n"**))  
 features\_considered = []  
 **for** i **in** range(n):  
 features\_considered.append(input(**"Enter the name of a parameter (its column title in data file)\n"**))  
 feature\_to\_predict = input(  
 **"Enter the name of the parameter to predict (its column title in data file)\n"**)  
 past\_history = int(input(  
 **"Enter the number of previous values on which prediction is based (must be positive integer)\n"**))  
 future\_target = int(input(**"Enter the length of the interval to predict (must be positive integer)\n"**))  
 train\_split = int(input(  
 **"Enter the number of records for training (must be positive integer). The rest will be used for validation.\n"**))  
 STEP = int(input(  
 **"Enter the training step size (must be positive integer). This means only 'step'th records will be used for training.\n"**))  
 epochs = int(input(**"Enter the number of epochs (must be positive integer).\n"**))  
 steps\_per\_epoch = int(  
 input(**"Enter the number of steps per one epoch (must be positive integer)\n"**))  
 validation\_steps = int(  
 input(**"Enter the number of validation steps (must be positive integer)\n"**))  
 buffer\_size = int(  
 input(**"Enter the size of the buffer used for shuffling (must be positive integer)\n"**))  
 batch\_size = int(input(**"Enter the batch\_size (must be positive integer)\n"**))  
  
 print(**"Preparing data..."**)  
 *# Подготавиливаем данные, создаём модель* model, train\_data, validation\_data, x, y, data\_mean, data\_std = prepare\_for\_training(past\_history,  
 future\_target,  
 data\_path,  
 features\_considered,  
 feature\_to\_predict,  
 train\_split, STEP,  
 buffer\_size,  
 batch\_size)  
 print(**"Training..."**)  
 *# Тренируем модель* train\_history = train(model, train\_data, validation\_data, epochs, steps\_per\_epoch, validation\_steps)  
 print(**"Training complete"**)  
 *# Выводим график потерь при обучении* plot\_train\_history(train\_history, **"Training and validation loss history"**)  
  
  
 **elif** selection == **'4'**: *# Сохраняем модель и веса в заданные файлы* **if** model **is not None**:  
 model\_path = input(**"Enter the path to model file (.json extention will be added automatically)\n"**)  
 weights\_path = input(**"Enter the path to weights file (.h5 extention will be added automatically)\n"**)  
 save(model\_path, weights\_path, model)  
 print(**"Model saved successfully"**)  
 **else**:  
 print(**"Train or load a model first"**)  
  
  
 **elif** selection == **'5'**: *# Загружаем модель из файла и данные для прогноза.* model\_path = input(**"Enter the path to model file (.json)\n"**)  
 weights\_path = input(**"Enter the path to weights file (.h5)\n"**)  
 data\_path = input(**"Enter the path to data file (.csv)\n"**)  
 past\_history = int(input(  
 **"Enter the number of previous values on which prediction is based (must be positive integer)\n"**))  
 STEP = int(  
 input(  
 **"Enter the training step size (must be positive integer). This means only 'step'th records were used for training.\n"**))  
 future\_target = int(input(**"Enter the length of the interval to predict (must be positive integer)\n"**))  
 n = int(input(**"Enter the number of parameters prediction is based on (must be positive integer)\n"**))  
 features\_considered = []  
 **for** i **in** range(n):  
 features\_considered.append(input(**"Enter the name of a parameter (its column title in data file)\n"**))  
 feature\_to\_predict = input(**"Enter the name of the parameter to predict\n"**)  
  
 model = **None** model = load(model\_path, weights\_path)  
 **if** model **is not None**:  
 print(**"Loading..."**)  
 df = pd.read\_csv(data\_path)  
 features = df[features\_considered]  
 features.index = df[**'Date Time'**]  
 dataset = features.values  
 data\_mean = dataset.mean(axis=0)  
 data\_std = dataset.std(axis=0)  
 dataset = (dataset - data\_mean) / data\_std  
 x, y = multivariate\_data(dataset, dataset[:, features\_considered.index(feature\_to\_predict)], 0,  
 **None**, past\_history,  
 future\_target, STEP)  
 print(**"Model loaded successfully"**)  
  
  
 **elif** selection == **'6'**: *# Выход из программы* **break  
  
  
 else**: *# В случае неверного ввода* print(**"Unknown Option Selected!"**)  
 **except**:  
 print(**"Something went wrong!"**)  
 input()

Приложение Б – Проверка на оригинальность

Проверка осуществлялась с помощью веб-ресурса: https://www.antiplagiat.ru/

Результат проверки: 94,76% оригинальности [Рисунок 37]

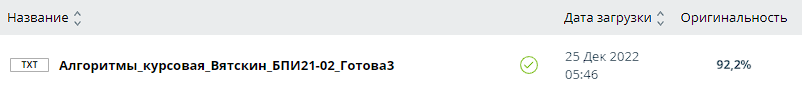


Рисунок – Проверка на оригинальность