Міністерство освіти і науки України
Національний технічний університет
«Дніпровська політехніка»

Інститут електроенергетики
(інститут)

Факультет інформаційних технологій
(факультет)

Кафедра Програмного забезпечення комп'ютерних систем
(повна назва)

ПОЯСНЮВАЛЬНА ЗАПИСКА
кваліфікаційної роботи ступеня

бакалавра
(назва освітньо-кваліфікаційного рівня)

студента *Гуліної Ольги Олександрівни*
(ПІБ)

академічної групи *122-18-1*
(шифр)

спеціальності *122 Комп'ютерні науки*
(код і назва спеціальності)

освітньої програми *Комп'ютерні науки*
(назва освітньої програми)

на тему: *Розробка нейронної мережі для візуалізації поверхні земної кулі у заданий час на основі Python/Tensorflow*

Development of a neural network to visualize the surface of the globe at a given time based on Python / Tensorflow

Керівники	Прізвище, ініціали	Оцінка за шкалою		Підпис
		рейтинговою	інституційною	
кваліфікаційної роботи	*доц. Реута О.В.*			
розділів:				
спеціальний	*доц. Реута О.В.*			
економічний	*доц. Касьяненко Л.В.*			
Рецензент				
Нормоконтролер	*доц. Гуліна І.Г.*			

Дніпро
2022

Міністерство освіти і науки України
НТУ «Дніпровська політехніка»

ЗАТВЕРДЖЕНО:
завідувач кафедри
програмного забезпечення комп'ютерних систем
(повна назва)

_____ І.М. Удовик
(підпис) (прізвище, ініціали)

« » _____ 2022 року

ЗАВДАННЯ
на кваліфікаційну роботу

бакалавра
(назва освітньо-кваліфікаційного рівня)

студента *122-18-1* (група) *Гуліна О.О.* (прізвище та ініціали)

тема кваліфікаційної роботи *Розробка нейронної мережі для візуалізації поверхні земної кулі у заданий час на основі Python/Tensorflow*

Development of a neural network to visualize the surface of the globe at a given time based on Python / Tensorflow

затверджена наказом ректора НТУ «ДП» від *18 травня 2022* №

Розділ	Зміст виконання	Термін виконання
Спеціальний	***На основі матеріалів проєктно-технологічної практики та інших науково-технічних джерел провести аналіз стану рішення проблеми та постановку задачі. Обґрунтувати вибір та здійснити реалізацію методів вирішення проблеми***	***13.05.2022 р.***
Економічний	***Провести розрахунок трудомісткості розробки програмного забезпечення, витрат на створення ПЗ й тривалості його розробки***	***27.05.2022 р.***

Завдання видав _____ *доц. Реута О.В.*
(підпис) (посада, прізвище, ініціали)

Завдання прийняв до виконання _____ *Гуліна О.О.*
(підпис) (прізвище, ініціали)

Дата видачі завдання: *14.01.2022 р.*

Термін подання кваліфікаційної роботи до ЕК: *13.06.2022 р.*

РЕФЕРАТ

Пояснювальна записка: 76 с., 21 рис., 1 табл., 21 джерел.

Об'єкт розробки: нейронна мережа для побудови зображення земної кулі.

Мета кваліфікаційної роботи: розробка нейронної мережі яка зможе будувати зображення нашої планети, опираючись тільки на вхідні дані. У результаті, мережа зможе будувати зображення планети у різний час, тобто робити власне передбачення вигляду земної кулі у майбутньому.

У вступі виконується аналіз сучасного стану проблеми, уточнюється постановка завдання, мета кваліфікаційної роботи та галузь її застосування, обґрунтовується актуальність теми.

У першому розділі проводиться дослідження предметної області та існуючих рішень, визначається актуальність завдання та призначення розробки, розроблюється постановка завдання.

У другому розділі обирається платформа для розробки, виконується проектування програми і її розробка, наводиться опис алгоритму і структури функціонування системи, визначаються вхідні і вихідні дані, наводяться характеристики складу параметрів технічних засобів, описується робота програми.

В економічному розділі визначається трудомісткість розробленого програмного продукту, проводиться підрахунок вартості роботи по створенню застосунку та розраховується час на його створення.

Практичне значення полягає у створенні веб-сервісу, що надає можливість переглядати результат виконання програми та взаємодіяти з додатком у реальному часі. Тобто подивитись на вигляд Землі, власноруч задавши час у веб-додатку.

Список ключових слів: НЕЙРОННА МЕРЕЖА, МАШИННЕ НАВЧАННЯ, АЛГОРИТМИ, СИСТЕМА, ДАНІ, БАЗА ДАНИХ, ІНФОРМАЦІЯ, ВІЗУАЛІЗАЦІЯ, НЕЙРОННА МЕРЕЖА ЗГОРТКИ, РЕКУРЕНТНА НЕЙРОННА МЕРЕЖА, ГЕОЛОГІЯ.

ABSTRACT

Explanatory note: 76 pp., 21 figs., 1 tabs., 21 sources.

Object of development: a neural network for constructing an image of the globe.

The purpose of the qualification work: the development of a neural network that can build images of our planet, based only on input data. As a result, the network will be able to build images of the planet at different times, that is, to make their own predictions of the shape of the globe in the future.

The introduction analyzes the current state of the problem, specifies the task, the purpose of the qualification work and the field of its application, substantiates the relevance of the topic.

In the first section the research of the subject area and existing decisions is carried out, the urgency of the task and the purpose of development is determined, the statement of the task is developed.

The second section selects the platform for development, performs program design and development, describes the algorithm and structure of the system, determines the input and output data, provides characteristics of the parameters of technical means, describes the program.

The economic section determines the complexity of the developed software product, calculates the cost of work to create an application and calculates the time to create it.

The practical value is to create a web service that allows you to view the results of the program and interact with the application in real time. That is, look at the appearance of the Earth, setting your own time in the web application.

List of keywords: NEURAL NETWORK, MACHINE LEARNING, ALGORITHMS, SYSTEM, DATA, DATABASE, INFORMATION, VISUALIZATION, CONVOLUTIONAL NEURAL NETWORK, RECCURENT NEURAL NETWORK, GEOLOGY.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

БД – база даних;

ОС – операційна система;

ЕОМ – електронно-обчислювальна машина;

API – це Application Programming Interface (з англ. інтерфейс для програмування застосунків);

Ч.Р. – часовий ряд

CNN – convolutional neural network, нейронна мережа згортки

RNN – recurrent neural network, рекурентна нейронна мережа

ВСТУП

Завдання даної кваліфікаційної роботи та об'єкт її діяльності безпосередньо пов'язані з напрямом підготовки та відповідає узагальненій тематиці кваліфікаційних робіт і переліку зазначених виробничих функцій, типових задач діяльності, умінню та компетенціям, якими повинні володіти бакалаври напряму 122 «Комп’ютерні науки та інформаційні технології».

Тематикою кваліфікаційної роботи є розробка нейронної мережі для створення зображення земної кулі на основі певних початкових даних. Метою кваліфікаційної роботи є створення алгоритму мережі який зможе, опираюсь на вхідні дані, будувати власне уявлення щодо вигляду земної кулі. Кваліфікаційна робота також передбачає розробку базового інтерфейсу, де можна було б задати вхідні дані та переглянути результат роботи мережі.

Після аналізу аналогічних сервісів та додатків були виявлені деякі технології, які стали у нагоді при створюванні цієї роботи. Історичні та геологічні дані також були влучені до роботи як теоретичні знання та частина логіки нейронної мережі.

Причиною виникнення необхідності розробки програмного забезпечення є безперечна цінність даної роботи у перспективі. З її допомогою

та різними вхідними даними можна буде будувати різні моделі планет, що може стати у нагоді у різних сферах наукової діяльності.

Виходячи з вищеописаної інформації, було прийнято рішення розробити складну нейронну мережу для вдоволення заданих цілей. Дане програмне забезпечення є актуальним через те, що воно унікальне та може застосовуватися у більш серйозних наукових роботах.

РОЗДІЛ 1

АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1. Загальні відомості з предметної галузі

Шту́чні нейро́нні мере́жі (ШНМ, англ. *artificial neural networks, ANN*), або конективістські системи (англ. *connectionist systems*) — це обчислювальні системи, натхнені біологічними нейронними мережами, що складають мозок тварин. Такі системи навчаються задач (поступально покращують свою продуктивність на них), розглядаючи приклади, загалом без спеціального програмування під задачу. Наприклад, у розпізнаванні зображень вони можуть навчатися ідентифікувати зображення, які містять котів, аналізуючи приклади зображень, мічені як «кіт» і «не кіт», і використовуючи результати для ідентифікування котів в інших зображеннях. Вони роблять це без жодного апріорного знання про котів, наприклад, що вони мають хутро, хвости, вуса та котоподібні писки. Натомість, вони розвивають свій власний набір доречних характеристик з навчального матеріалу, який вони оброблюють.

Первинною метою підходу ШНМ було розв'язання задач таким же способом, як це робив би людський мозок. З часом увага зосередилася на відповідності певним розумовим здібностям, ведучи до відхилень від біології.

ШНМ використовували в ряді різноманітних задач, включно з комп'ютерним баченням, розпізнаванням мовлення, машинним перекладом, соціально-мережевим фільтруванням, грою в настільні та відеоігри, та медичним діагностуванням.

1.2. Призначення розробки та галузь застосування

Повна назва розробленої системи для кваліфікаційної роботи – «Розробка нейронної мережі для візуалізації земної кулі у заданий час на основі Python/Tensorflow». Основна термінологія та ключові слова:

- **Нейронна мережа** - один із напрямків штучного інтелекту, мета якого змоделювати аналітичні механізми, що здійснюються людським мозком.

- **Python** - це мова програмування високого рівня, інтерактивна, об'єктно-орієнтована інтерпретаційна. Python проста у використанні, та водночас повноцінна мова програмування, що надає багато засобів для структурування і підтримки великих програм. Python дозволяє розбивати програми на модулі, що потім можуть бути використані в інших програмах. Python поставляється з великою бібліотекою стандартних модулів, які можна використовувати як основу для нових програм або як приклади при вивченні мови. Стандартні модулі надають засоби для роботи з файлами, системними викликами, мережними з'єднаннями і навіть інтерфейсами до різних графічних бібліотек.

- **Tensorflow** - це бібліотека з відкритим вихідним кодом для цифрових обчислень і великомасштабного машинного навчання. TensorFlow поєднує

багато моделей і алгоритмів для машинного навчання та глибокого навчання (також відомих як нейронні мережі) і робить їх корисними за допомогою загальної метафори. Він використовує Python, щоб забезпечити зручний інтерфейс для створення додатків на основі фреймворків, запускаючи ці програми на високопродуктивному C ++.

Ця технологія створена у першу чергу з метою наукового підходу до побудови нашої планети. Згідно геологічних даних, планета Земля зазнавала значних впливів та змін з моменту її створення, які були викликані багатьма факторами.

Зрушення тектонічних плит найбільш за все посприяло зсуву континентів на нашій планеті у порівнянні з першими даними припущення вигляду Землі (близько 750 мільйонів років тому).

Великий вплив також принесли і зовнішні фактори такі як метеоритний дощ (близько 65 мільйонів років тому) у епоху життя динозаврів.

До уваги також можна взяти такі фактори як глобальне потепління, що сприяє і до сьогодні постійній зміні рівня CO2 та кисню, рівню поверхні води на планеті, кількість льодовиків. Важливим фактором також є приріст кількості людей, що живуть на планеті. Природний приріст становить 10.8% на даний момент, що являється доволі високим показником. Приріст населення впливає на кількість CO2, результати праці швидко збільшуюсь рівень забруднень на планеті, погіршується екологія, температура глобально росте.

Головною метою роботи є розробка такої мережі, яка, опираючись на фактори зазначені вище, зможе не тільки зображати планету у різні моменти часу у минулому, а і зможе будувати власні припущення щодо вигляду Землі у майбутньому.

Ми маємо зображення планети у різні моменти часу. Не складно знайти як виглядала наша домівка 10, 100 тисяч, мільйон років тому. Але, невже не буде цікаво поглянути у майбутнє? З нашими сучасними технологіями ми не можемо зробити це у реальному житті та подивитися крізь роки уперед, але ми можемо зробити припущення. Звичайній людині буде важно проаналізувати усі фактори, щоб хоча б намалювати приблизну модель планети через 100 або 1000 років. Але, на відміну від людини, у нейронної мережи є невичерпний ресурс, користуючись яким вона може об’єктивно та цільно оброблювати інформацію і робити найбільш вирогідне припущення.

Саме тому я обрала цю технологію для своєї ідеї. Технологія зможе застосовуватись у першу чергу у науковій діяльності. Результат мережі зможе сильно вплинути на наші уявлення про планету, побачити найбільш вирогідний результат, проаналізувати наші дії як населення у глобальному масштабі. У 2016 році Стівен Хокінг предрік гибель людства не більш ніж через 1000 років. На його думку, це станеться не через діяльність людства на планеті (глобальне потепління та приріст населення як наслідки). Найбільш очікуваними причинами він назвав ядерну війну, розповсюдження смертельного вірусу, або діяльність штучного інтелекту.

Тим не менш, людська діяльність дуже сильно руйнує екологію, що з часом зробить життя на планеті не комфортним та важким в плані ресурсів.

Якщо температура сильно підвиситься, це також повпливає на вимирання деяких видів тварин та комах та зробить життя для людей тут фізично тяжким.

Моя робота зможе допомогти у екологічній, геологічній, антропологічній, астрономічній та технологічній сферах. Зазирнувши вперед, побачивши результат наших діянь, можна буде зрозуміти причини деяких катаклізмів, ставшихся у минулому. Аналізуючи результат моєї роботи, можна вдосконалити екологічні методи боротьби, які використовуються наразі. Подивившись на результат системи, безперечно можна продумати плани на майбутнє освоєння космосу, взявши до уваги припущення щодо життя та ув'ядання Землі. З іншого боку, результат може буде використаний для технологічних засобів для збереження та покрашення нашої планети.

Система може бути застосована людьми як інструмент ознайомлення та покращення знань у предметній галузі. Вона також може посприяти заохоченню молодих умів до науки, адже завдяки своїй унікальності вона дає неповторний функціонал для вивчення планети та побудові припущень.

1.3. Підстави для розробки

Чим більше ми знаємо, тим легше планувати наступні дії. Історія дає нам спосіб та можливість робити припущення, основані на подіях минулого. Це можна порівняти з даром погляду у майбутнє, але, на відміну від недоказаних теорій та непідтверджених пророцтв, опираючись на історію, ми можемо чітко назвати причини наших думок та рішень.

Я вважаю це підхід кращим у багатьох сферах нашого життя. Аналізуючи інформацію, ми робимо найбільш корисні та логічні, за нашою думкою, вибори. Без сумніву, ця система не бездоганна адже, як я казала, нам дуже складко та ресурсо-затратно аналізувати всі дані при кожному рішенні, що ми маємо прийняти.

Підставою для планування ідеї моєї розробки стала дуже далека від айті подія. У 2014 році на території Карелії стався вибух захворювання сибірською язвою. Для тих місць це була дуже нетипова подія, адже захворювання до цього не було помічене у тому регіоні упродовж століть. Від раптового вибуху померло близько 800 голів рогатого скота та декілька людей. Ніхто не вакцинувався від цього захворювання, адже ніхто не міг подумати, що взагалі існує ризик захворіти. А ризику і справді не було так як хвороба мирно перезимовувала у товщах грунту тієї місцевості. Через аномально спекотний сезон того року, спричинений глобальним потеплінням, хвороба несподівано вийшла із сплячого стану та почала активно діяти.

Це натовкнуло мене на думку, що знай населення про ризики, цієї події скоріш за все не сталося би.

Хоча айті и вважається далекою сферою під подібного типу подій, я вважаю необхідним залучитися та допомогти розвитку деяких питань у інших сферах за допомогою технологій. Айті вже приносить безцінний вклад у розвиток медицини, будівництва, екології, освіти, але багато питань ще досі лишені неосвіченими.

Відповідно до освітньої програми, згідно навчального плану та графіків навчального процесу, в кінці навчання студент виконує кваліфікаційну роботу. Тема роботи узгоджується з керівником проекту, випускаючою кафедрою, та затверджується з наказом ректора.

Таким чином підставами для розробки (виконанням кваліфікаційної роботи) є:

− освітня програма спеціальності 122 «Комп’ютерні науки та інформаційні технології»;
− навчальний план та графік навчального процесу;
− наказ ректора Національного технічного університету «Дніпровська політехніка» №317-с від 07.06.2022 р;
− завдання на кваліфікаційну роботу на тему «Розробка нейронної мережі для візуалізації земної кулі у заданий час на основі Python/Tensorflow».

1.4. Постановка завдання

Метою роботи є створення нейронної мережі, яка, опираючись на вхідні дані, зможе робити власні припущення щодо вигляду нашої планети у минулому та майбутньому.

Модель повинна:

- Вміти оброблювати вхідні дані
- Виконувати математичні обчислення над вхідними даними
- Перетворювати вхідні значення у вихідні на основі математичних операцій
- Видавати вихідні дані
- Розраховувати ваги при обчисленнях, рахувати відносну похибку
- Згідно похибці, корегувати ваги та змінювати їх до більш точних

1.5. Вимоги до програми або програмного виробу

1.5.1. Вимоги до функціональних характеристик

Розроблюваний кінцевий веб-сервіс повинен відповідати наступним функціональним характеристикам:

- Давати змогу користувачу вводити необхідний час на заданому часовому проміжку

- Давати змогу побачити результат виконання системи на екрані, згідно введених даних

- Давати можливість онулити результат та задати нову дату на часовому проміжку

1.5.2. Вимоги до інформаційної безпеки

Для забезпечення надійного функціонування фінальній версії веб-застосунку необхідно реалізувати наступні вимоги:

− контроль та верифікація вхідних даних при відправці запиту на сервер та перевірка на відповідність типів та на введення обов’язкових полів;

− контроль вихідних даних, виключення випадків відправки важливої / приватної інформації з серверної частини на клієнтську;

− обробка виняткових ситуацій та виведення повідомлень у разі виникнення помилок.

1.5.3. Вимоги до складу та параметрів технічних засобів

Для нормального функціонування веб-сервісу необхідно, щоб обчислювальна машина, на якій буде функціонувати система, відповідала наступним вимогам:

− чотирьох ядерний процесор з тактовою частотою не менш 2.4 ГГц;

− оперативна пам'ять обсягом не менше 8 GB;

− SSD-накопичувач 120 GB;

− Відеокарта для швидшого розрахунку та виводу результату.

Наведені вище технічні характеристики є рекомендованими, тобто при наявності технічних засобів не нижче зазначених, розроблений програмний виріб буде функціонувати відповідно до вимог щодо надійності, швидкості обробки даних і безпеки, висунутими замовником.

1.5.4. Вимоги до інформаційної та програмної сумісності

Для нормального функціонування програми необхідно, щоб програмне забезпечення обчислювальної машини, на якій буде функціонувати веборієнтована система, відповідало наступним вимогам:

− Операційні системи: Windows (7, 8, 10), Mac OS, Linux.

− Браузери: Chrome, Safari, Mozilla Firefox, Edge, Opera та інші.

− Будь-яка середа для запуску коду на Python: PyCharm, IDLE, Spyder, Atom, Sublime Text + console. Як альтернативу, можна застосовувати Google Colaboratory - онлайн середу, яка дозволяє корегувати та запускати код у браузері.

Рекомендована швидкість інтернет-з'єднання: від 3-5 Мбіт / сек.

Основні мови програмування, які були використані: Python, Tensorflow, Keras API, бібліотеки Python такі як Pandas, Numpy, Matplotlib, Seaborn.

РОЗДІЛ 2

ПРОЄКТУВАННЯ ТА РОЗРОБКА ІНФОРМАЦІЙНОЇ СИСТЕМИ

2.1. Функціональне призначення системи

2.2. Опис застосованих математичних методів

У системі за основу взяті принципи диференціювання, принцип градієнтного спуску, середньоквадратичні рівняння та інші обчислювані методи, які виконуються оптимізаторами, вбудованими у мову та Keras API.

Градієнтний спуск - це алгоритм, який допомагає нам знайти мінімальну похибку або де значення втрат менше. Метод градієнтного спуску є найбільш раннім і найпоширенішим методом оптимізації. Ідея методу градієнтного спуску полягає в тому, що змінні ітеративно оновлюються у (протилежному) напрямку градієнтів цільової функції. Оновлення виконується для поступового зближення до оптимального значення цільової функції. Швидкість навчання η визначає розмір кроку в кожній ітерації, таким чином, впливає на кількість ітерацій, щоб досягти оптимального значення

2.3. Опис використаних технологій та мов програмування

Цей веб-сервіс був розроблений з використанням таких технологій та мов програмування:

- Python
- Tensorflow
- Keras API
- Бібліотеки Python: Pandas, Numpy, Matplotlib, Seaborn
- Google Collaboratory

Keras API є одним із провідних API нейронних мереж високого рівня. Він написаний на Python і підтримує багато механізмів для обчислення нейронних мереж із задньої частини.

Keras розроблено, щоб бути простим у використанні, модульним, легко розширюваним і працювати з Python. API «призначений для людей, а не для машин» і «дотримується найкращих практик для зменшення когнітивного навантаження».

Нейронні шари, функції вартості, оптимізатори, схеми ініціалізації, функції активації та схеми керування — це окремі модулі, які можна об'єднати для створення нових моделей. Нові модулі легко додавати як нові класи та функції. Моделі визначаються в коді Python, а не в окремих файлах конфігурації моделі.

Google Collaboratory — це безкоштовне інтерактивне хмарне середовище для роботи з кодом від Google. Він дає змогу створювати й виконувати код Python у веб-переглядачі за допомогою:

- без попередніх налаштувань
- з безкоштовним доступом до графічних процесорів
- з легким спільним доступом

2.4. Опис структури системи та алгоритмів її функціонування

2.4.1. Опис архітектурного підходу

При обранні найкращої архітектури для моєї системи я зіткнулася із питанням вибору типу мережі, яка має бути застосована. Сучасні варіанти існуючих нейронних мереж, типологія яких вже розроблена та використовується на практиці у різних галузях, вражає своєю різноманітністю.

Серед основних можна виділити класичну систему перцептрона, нейронну мережу згортки (convolutional neural network – CNN), рекурентні нейронні мережі (RNN).

У своїй роботі я використовувала всі ці типи та кілька більш унікальних видів для дослідження кращої системи для розрахунку результату. Моєю метою було побачити, яка система найкраще впорається із заданою задачею.

Хоча може здаватись, що чим складніша типологія використовується у мережі, тим точніше вона має показати результат, на практиці це не так. У процесі навчання, нейронна мережа автоматично генерує ваги, які впливають на вихідні значення. Ці ваги напочатку генеруються рандомно (або майже рандомно, якщо не було задано ніяких обмежень та стартових параметрів). Навіть у однієї системи ваги можуть відрізнятись в залежності від кількості заданих вхідних параметрів. Для деяких типів мереж, ваги генеруються для

кожного шару, а для деяких (перцептрон) – для кожного нейрона у кожному шарі окремо. Даремно і казати, що ваги у різних моделях нейронної мережі також будуть різними.

При побудові системи та створенні пакету вхідних значень дуже важко сказати, яка саме архітектура нейронної мережі буде працювати найкраще. Саме через незнання значень вагів напочатку роботи, неможливо передбачити результат виконання кожної моделі мережі без практичного її створення і застосування. Тобто, лише на практиці можливо з'ясувати, яка саме архітектура буде підходити для виконання конкретної задачі.

Саме тому під час роботи я додала усі основні типи нейронних мереж до проекту та створила по одному екземпляру. Кожний тип мережі виконував незалежні обчислення похибок та вагів, навчався під час кожної ітерації, та виводив свій результат. Таким чином, я мала змогу порівняти результати та виділити найбільш точну мережу серед інших.

Першочергово, було необхідно завантажити дані, згідно з якими мережа мала навчатися та робити прогнозування. Завантажені дані було перетворено на датасет за допомогою підкласів Keras API.

При створенні мережі, не всі дані мають йти саме на її навчання. Дані було розділено на валідаційні, тренувальні та тестові у відношенні: 20% /70% /10%. При цьому дані не перемішуються випадковим чином перед розподілом. Це з двох причин:

- Це гарантує, що поділ даних на вікна послідовних вибірок, можливий.
- Це гарантує, що результати перевірки/тестування будуть більш реалістичними, оскільки оцінюються на основі даних, зібраних після навчання моделі.

При створенні моделей для обробки підготовлених даних, я розраховувала на архітектуру рекурентної нейронної мережі як на найкращу систему для моєї моделі.

Рекурентна нейронна мережа (RNN) - це тип нейронної мережі, що добре підходить для даних часових рядів. RNN обробляють тимчасовий ряд крок за кроком, зберігаючи внутрішній стан від кроку до кроку. Сигнал з вихідних нейронів або нейронів прихованого шару частково передається на входи нейронів вхідного шару (зворотний зв'язок).

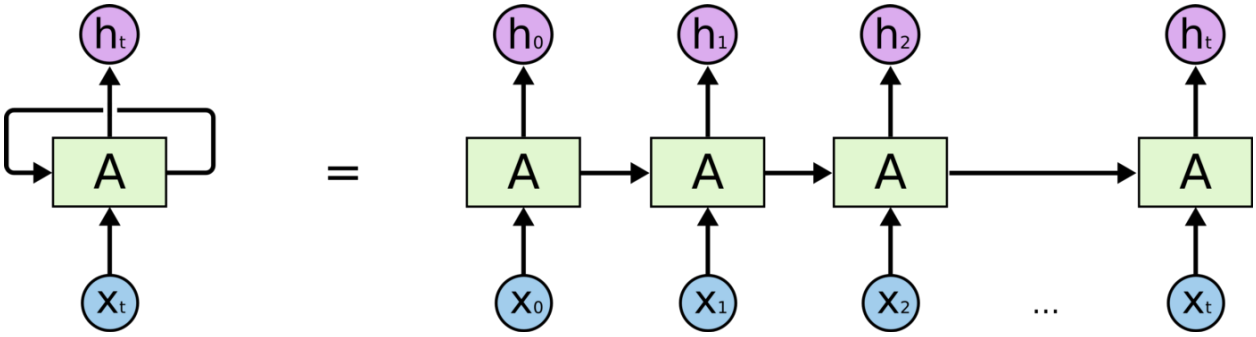


Рис. 1.1 Схема рекурентної нейронної мережі

Одним із різновидів рекурентних нейронних мереж є *до́вга коротко́часна па́м'ять* (англ. *long short-term memory, LSTM*). На відміну від традиційних RNN, мережа LSTM добре підходить для навчання з досвіду з метою обробки, класифікації, або передбачення часових рядів в умовах, коли між важливими подіями існують часові затримки невідомої тривалості. Відносна нечутливість до довжини прогалин дає LSTM перевагу в численних застосуваннях над альтернативними RNN.

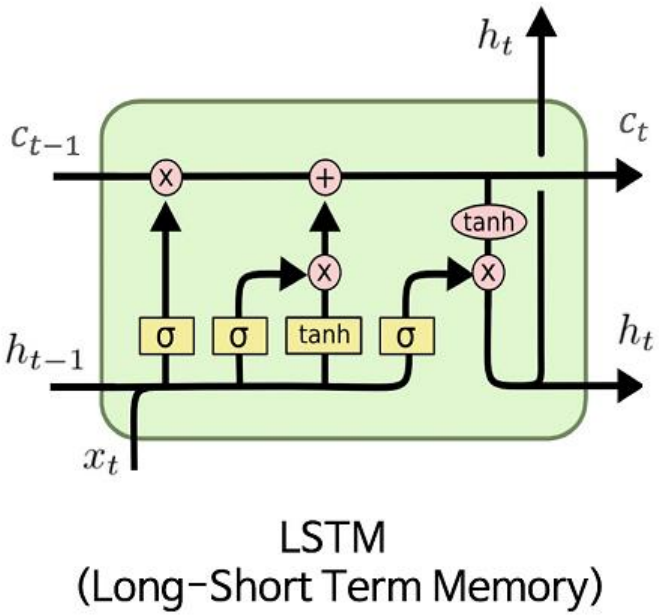


Рис 1.2 Схема LSTM мережі

Ще одним популярним типом нейронних мереж є нейронна мережа згортки (англ. *convolutional neural network,* CNN). Найбільшого застосування вона набула при роботі задач класифікації зображень. Цей тип мережі однонаправлений та багатошаровий. Принцип роботи у тому, що розпізнавання образів проходить у багато кроків, починаючи із розпізнавання простих елементів та поступово перетворюючи їх у більш складні об’єкти. Таким чином, CNN може використовуватись для розпізнавання різних наборів даних, різних груп об’єктів: архітектура використовується для наборів рукописних цифр, класифікацій предметів, тварин, та розпізнавання облич людей.

Ця архітектура була також використана у моїй моделі через свою розповсюдженість. Незважаючи на свою оригінально іншу задачу, вона показала досить непогані результати у прогнозуванні часових рядів.

Шар згортки CNN – це основний блок цього типу мережі. Шар згортки включає в себе унікальний фільтр для кожного каналу, ядро згортки якого обробляє попередній шар за фрагментами (підсумовуючи результати

поелементного результату множення для кожного фрагмента). Вагові коефіцієнти ядра згортки (невеликої матриці) невідомі та встановлюються у процесі навчання.

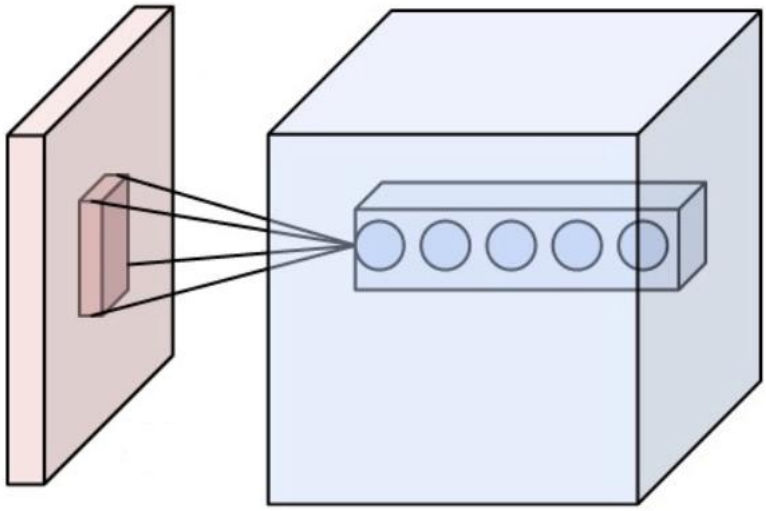


Рис 1.3 Шар згортки CNN

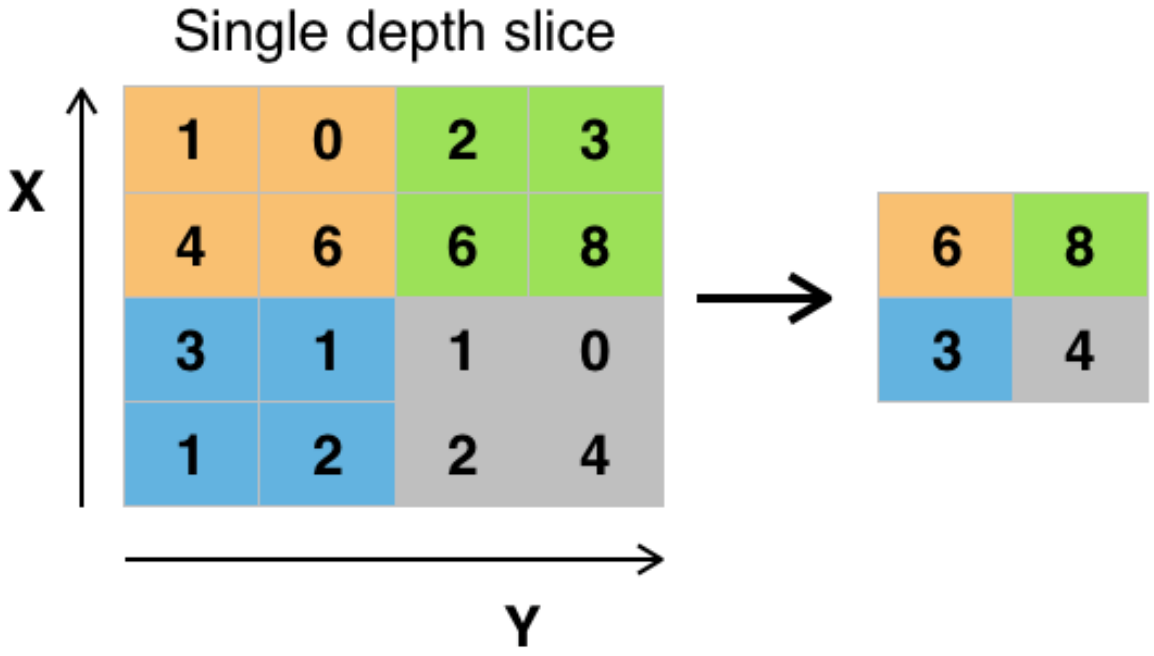


Рис. 1.4 Схема роботи пулінгу CNN

Пулінг – важливий принцип роботи CNN мереж. Завдяки йому здійснюється поступове розпізнавання об’єктів на зображенні, починаючи із примітивних рис, поступово збільшуючись до більш складних та детальних. Шар пулінгу також іноді використовується для зменшення зображення у інших задачах, так як він відкидує вже існуючи оброблені риси зображення, переходячи до нового шару. Таким чином, розмір зображення також зменшується.

2.4.2. Опис структури бази даних

Головною метою задачі було завантаження даних до системи, їх обробка та нормалізація, математичні обчислення над обробленими даними, та їх імпорт до окремого файлу.

Для функціонування кінцевого веб-застосунку також повинен бути розроблений функціонал для обробки записаних даних, правильний їх вивід до системи веб-застосунку, відображення даних інтерфейсом застосунку для кінцевих користувачів.

На даний момент, у системі використовується тільки функціонал локального збереження даних у вигляді CSV, імпорт даних у систему здійснюється за допомогою виклику Keras API також у вигляді архівованого CSV файлу.

Джерелом даних у роботі став портал EarthByte (earthbyte.org). Дані щодо деяких характеристик планети зібрано у окремі файли в зручній CSV таблиці. Дані зібрані за часом: період 1000-0 Ма (мільйон років).

У роботі використовувалась вибірка даних про серединно-океанічний хребет планети. Це глобальна система лінійно-довгастих піднять на дні океанів, протяжністю в десятки та сотні тисяч кілометрів.

Вибірка включає в себе дані про довжину загального хребта, його протяжність та середні значення за весь проміжок часу.

2.4.3. Короткий опис структури проекту

Проект побуваний у вигляді пов’язаного між собою коду, блоків класів, функцій, та окремо створених змінних. Оскільки на даний момент реалізація візуального інтерфейсу не запущена у веб-застосунок, проект не містить окремих модулів для реалізації серверної та клієнтських частин, що мають бути відокремлені від функціоналу нейронної мережі.

При створенні веб-застосунку, клієнт-серверна частини будуть відокремлені у окремі файли щоб забезпечити зручний доступ до файлів проекту, можливість корегувати/додавати функціонал до необхідних частин коду, не порушуючи цільний каркас програми та встановлені зв’язки між її складовими.

У результаті, модулі розподіляться наступним чином:

- модуль для завантаження вхідних даних, їх нормалізації та базової обробки;

- модуль створення класів нейронної мережі та загальних функцій;

- модуль створення екземплярів нейронної мережі, їх навчання та запис результату роботи до окремих файлів;

- модуль клієнт-частини для описання інтерфейсу користувача, описання логіки клієнт-частини;

- модуль серверної частини проекту, його взаємодія з клієнтом.

2.4.4. Детальний опис модулів розроблюваної системи

У модулі для завантаження вхідних даних має бути функціонал для виклику файлів із EarthByte ресурсу, перетворення даних у масив для подальшої обробки системою. Масив перетворюється у датасет за допомогою Keras API підкласів, який у свою чергу може бути використаний нейронною мережею для тренування.

Модуль створення класів нейронної мережі має містити усі типи мереж, які можуть реалізувати механізм поставленої задачі. У своїй роботі я використала шість типів нейронних мереж за допомогою функціоналу Keras та Tensorflow. Було створено власний тип Baseline, який наслідувався від стандартної моделі Keras.Model. Цей модуль має містити каркас основних типів мереж та їх архітектуру.

Модуль створення екземплярів нейронних мереж має містити реалізацію класів мереж, створених у попередньому модулі. Змінним присвоюється тип нейронної мережі, створюються масиви для запису даних, екземпляри класів навчаються за допомогою написаного попередньо каркасу. Результат навчання (ваги, середня похибка, точність) можуть виводитись у консоль. Запис результату фіксується у окремому файлі для подальшого використання у клієнт-серверній частині. Також в цьому модулі відбувається порівняння успішності кожного типу архітектури мережі згідно результуючих даних. Для зручності оцінки, створено візуальну діаграму успішності кожної мережі.

Модуль клієнт-частини має містити функціонал взаємодії системи і кінцевих користувачів. Користувач матиме змогу вводу часового проміжку або точного часу на екрані. Обробивши запит, користувач побачить на екрані вигляд планети у заданий час. На даному етапі, інформація про вигляд планети основана лише на даних про серединно-океанічний хребет та тектонічні плити,

але у майбутньому система буде доповнена іншими параметрами для більш вдосконаленого виводу результату.

Модуль серверної частини проекту повинен бути пов’язаний із сервером, де будуть проходити всі обчислення, прийняття і відсилання запитів кінцевим користувачам.

2.4.5. Опис можливості збереження даних

Оскільки результат системи це онлайн веб-застосунок, який не потребує аутентифікації, створення власного кабінету, облікового запису, або вводу будь-яких персональних даних, власник продукту не має запитувати та зберігати будь-яку персональну інформацію користувачів та виділяти місце під неї на серверах. Єдине що може фіксуватися при використанні сервісом це базова інформація про браузер, версію, cipher suits для встановлення зв’язку браузера з сервером. Для цього достатньо стандартних сервер-клієнтських API викликів.

Для побудови веб-застосунку та зберігання даних програми на сервері, обробки даних та їх відсилання на запит користувача, я можу обрати декілька варіантів.

Перша опція це сховище S3 від компанії Amazon. Воно надає можливість зберігання та отримання будь-якого обсягу даних в будь-який час з будь-якої точки мережі. Ця веб-служба є комерційною публічною хмарою Amazon Web Services, яка надає передплатникам послуги як по інфраструктурної моделі (віртуальні сервери, ресурси зберігання), так і платформного рівня (хмарні бази даних, хмарні обчислення, хмарне програмне забезпечення, засоби розробки). Amazon S3 пропонує зручні у використанні інструменти

адміністрування, які дозволяють організувати дані і точно налаштувати обмеження доступу відповідно до потреб.

Безкоштовний сервіс Amazon S3 надає клієнтам 5 гігабайт для зберігання файлів, 20000 запитів на отримання і 2000 запитів на оновлення даних.

Інша опція це хостинг пропонований компанією Namecheap. Хоча це і платна послуга, найдешевший можливий план надасть 20 GB SSD пам'яті, безкоштовний CDN сервіс та можливість завантажувати оновлення, файли, редагувати сайт через зручну інтерфейс-панель замість консолі. Також, Namecheap хостинг не має обмежень на передачу даних по мережі, у той час як у Amazon встановлений ліміт – 2000 запитів.

2.4.6. Опис аутентифікації користувача у веб-сервісі

Сервіс не верифікує кінцевих користувачів, запитуючи логін деталі або будь-яку персональну інформацію. Веб-додаток є універсальним та відкритим, він не колекціонує персональні дані, не передає їх по мережі на сервер.

2.5. Обґрунтування та організація вхідних та вихідних даних програми

Вхідні дані приходять у систему у вигляді цілих та дробових чисел, зібраних у таблицю CSV файлу. Ці дані можна порівняти із масивом. Ці дані сортуються до масиву для тренування, валідації та тесту та нормалізуються перед використанням мережею. Після обробки дані записуються у окремий CSV файл у вигляді масиву.

Для реалізації трансферу даних між сервером та клієнт-браузером найпевніший варіант це передача даних у форматі JSON.

2.6. Опис розробленої системи

Розроблений продукт являє собою нейронну мережу, яка робить передбачення деяких параметрів земної кулі у майбутньому на основі даних, зібраних у попередні роки. Кінцевий продукт – це веб-застосунок, який зможе відображати результат із UI інтерфейсом для кінцевих користувачів, беручи за основу розрахунки нейронної мережі.

Сервіс не робить ніяких запитів для ідентифікації користувача. Дані, розраховані нейронною мережею, будуть віддаватися користувачу згідно його запиту через інтерфейс застосунку.

2.6.1. Використані технічні засоби

Для правильного функціонування розробленої системи, необхідно мати ЕОМ з певними характеристиками, а саме потрібен сервер, на якому буде запущений проект даної кваліфікаційної роботи. Згідно припущень, проект рекомендовано бути запущеним на віддаленому сервері з наступними мінімальними характеристиками:

− ЦП [CPU]: Чотирьохядерний процесор.

− Накопичувач [HDD]: 1 ГБ.

− Оперативна пам’ять [RAM]: 1 ГБ.

Безпосередньо розробка і тестування системи проводилось на ЕОМ з наступними технічними характеристиками:

− ЦП [CPU]: Inter Core i3.

− Відеопам'ять [VRAM]: 8 ГБ.

− Накопичувач [SDD]: 120 ГБ.

− Оперативна пам'ять [RAM]: 16 ГБ

2.6.2. Використані програмні засоби

Під час розробки даної кваліфікаційної роботи були використані такі програмні засоби, за допомогою яких забезпечується функціонування розробленої системи:

- Google Colaboratory
- PyCharm

Colaboratory, або скорочено «Colab», — це продукт від Google Research. Colab дозволяє будь-кому писати та виконувати довільний код Python через браузер, і особливо добре підходить для машинного навчання, аналізу даних та освіти. Більш технічно, Colab — це розміщена служба ноутбуків Jupyter, яка не вимагає налаштування для використання, а також надає безкоштовний доступ до обчислювальних ресурсів, включаючи графічні процесори.

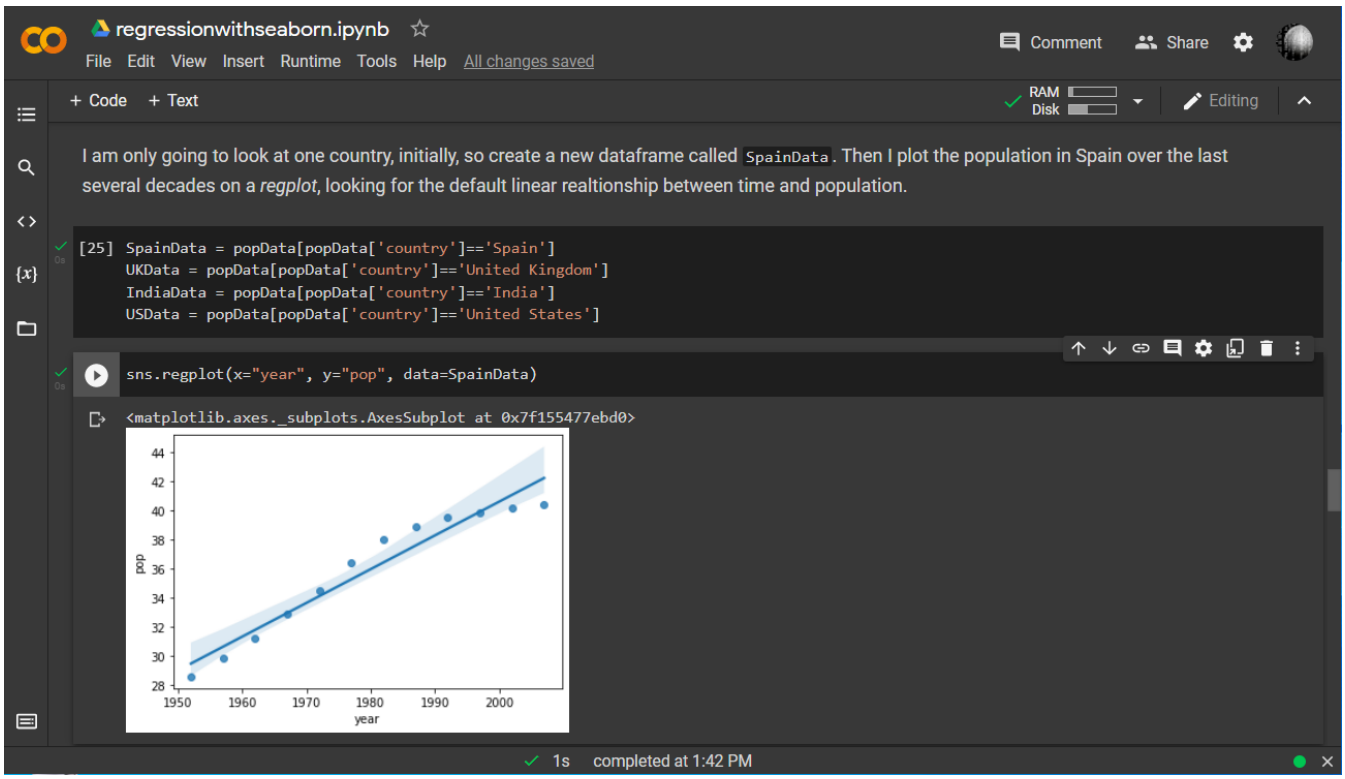


Рис. 2.1 Рабочий екран Colab

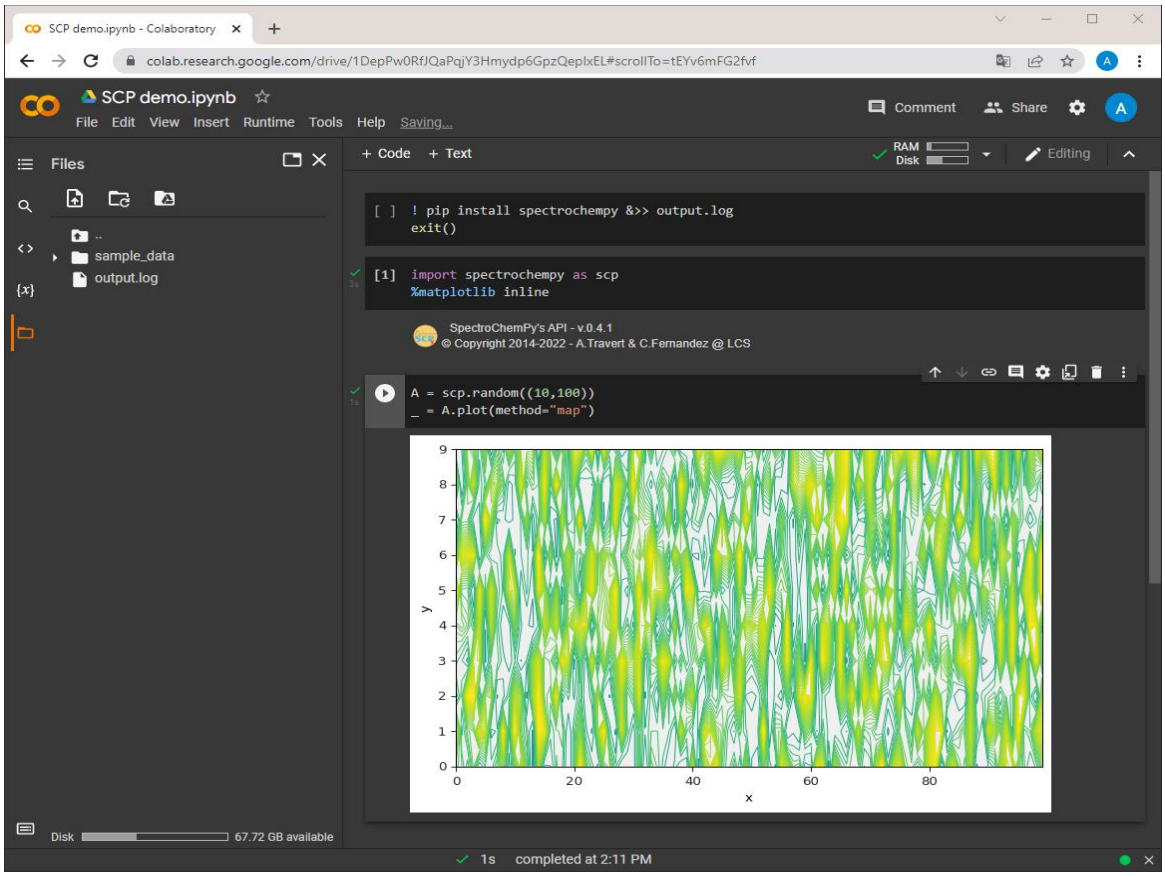


Рис. 2. 2

PyCharm — це IDE Python для науки про дані та веб-розробки з інтелектуальним завершенням коду, швидкою онлайн перевіркою помилок у реальному часі та швидкими виправленнями.

Воно надає широкий спектр необхідних інструментів для розробників Python, тісно інтегрованих для створення зручного середовища для продуктивної розробки Python, веб- та наукових досліджень.

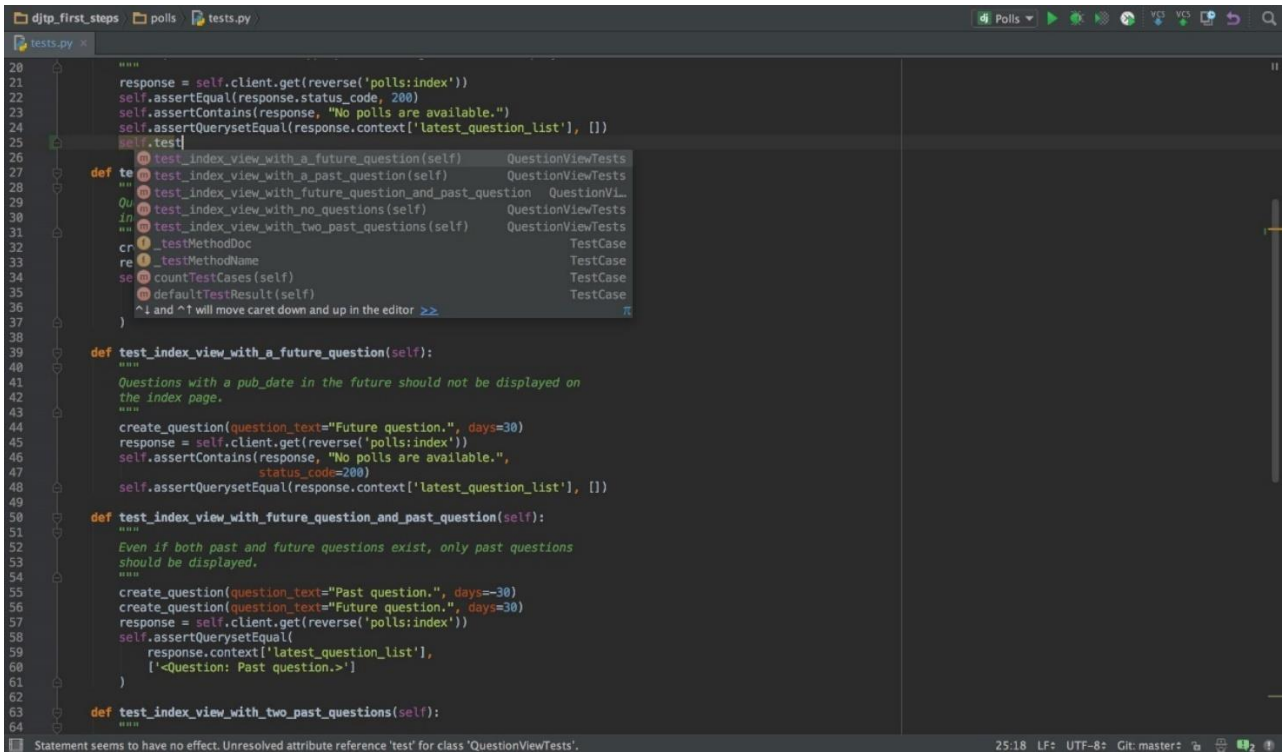


Рис. 2.3 Приклад вікна PyCharm

Перевагою цієї IDE є те, що робота над проектом відбувається на локальному комп’ютері, що дає змогу не переживати на цілісність та доцільне збереження файлів. Також це дає змогу швидко взаємодіяти із файловою системою комп’ютера.

2.6.3. Використана методологія розробки

Під час розробки даної кваліфікаційної роботи була використана гнучка методологія розробки Scrum. При розробці проекту, перевага віддалася саме цій технології, а не Agile. Scrum розбивається на коротші спринти та менші результати, тоді як у Agile результат показується наприкінці проекту.

Agile залучає членів різних міжфункціональних команд, тоді як команда проекту Scrum включає в себе певні ролі, наприклад, Scrum Master і Product Owner. Цей нюанс не був ключовим у обранні методології, але те, що Scrum вимагає показ некінцевих результатів швидше у часі став вирішальним у виборі саме цього підходу.

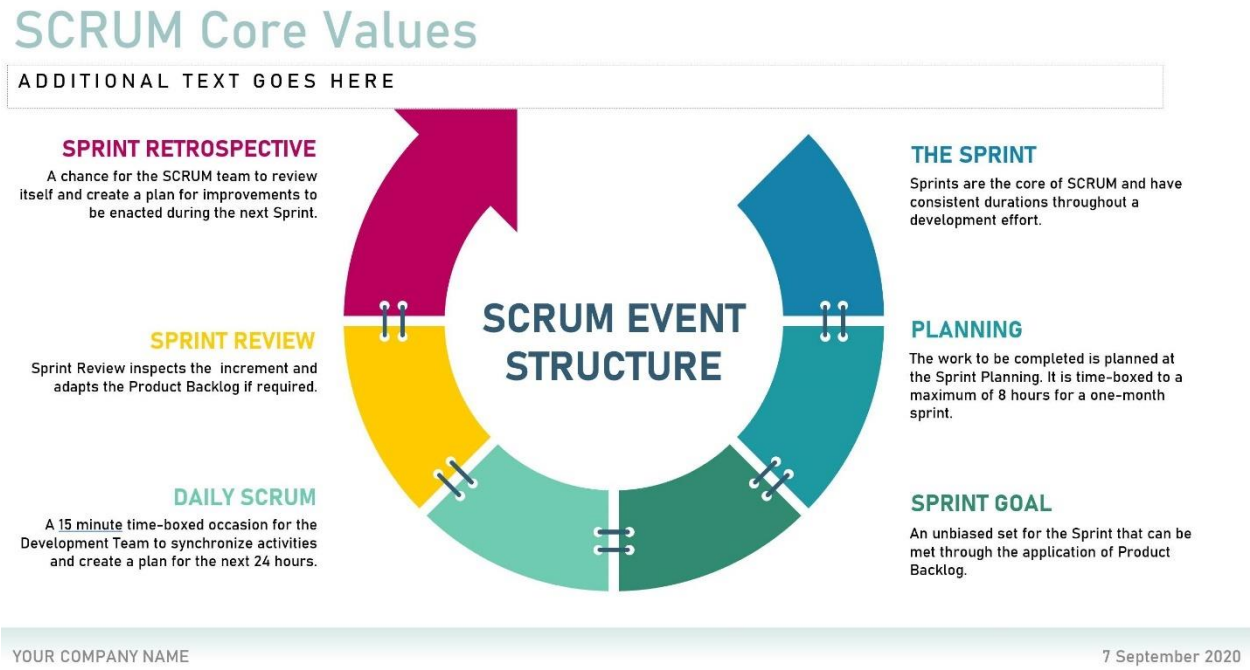


Рис. 2.4 Методологія Scrum

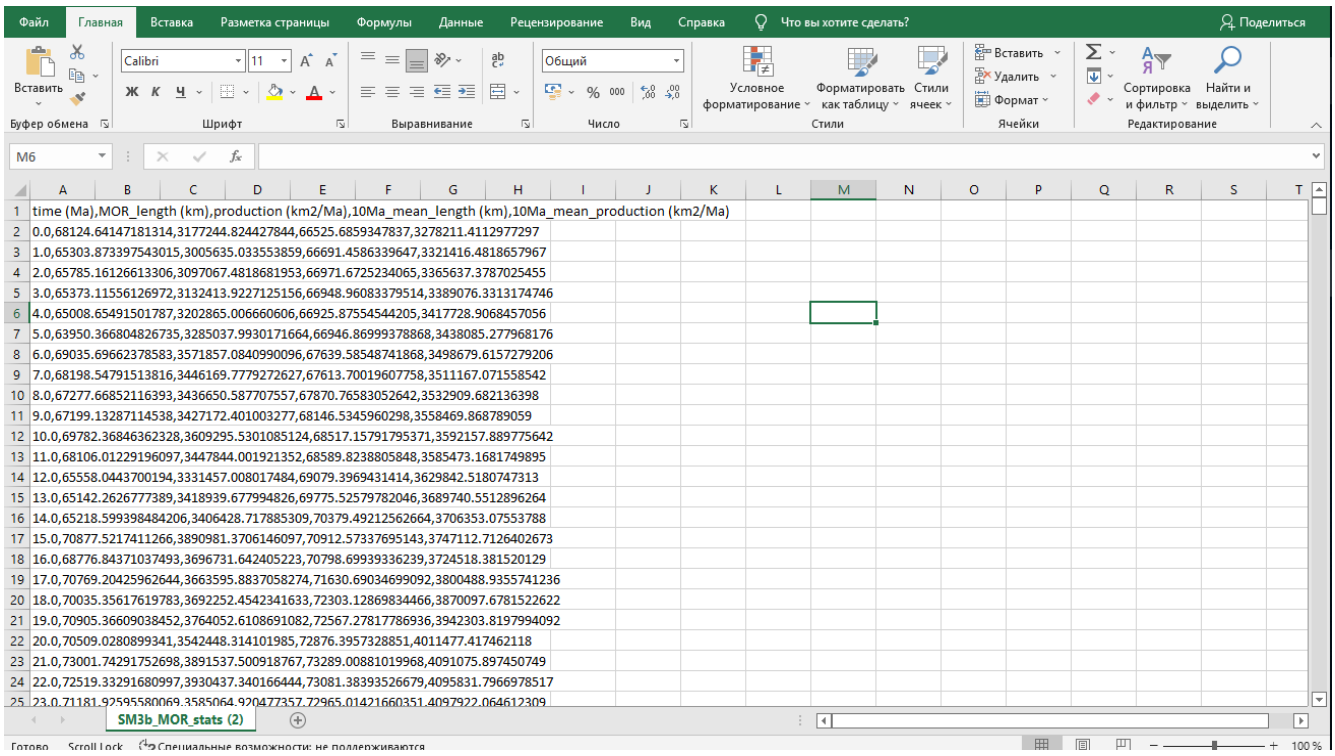
Основні принципи методології:

- Можливість оцінки результату завдання та ідеї для покращення результату наступного завдання;
- Оцінка результату завдання підвищує та корегує попередні дані проекту, якщо необхідно;
- Обговорення результату із командою, створення плану наступних завдань
- Зустріч та обговорення проекту з усіма членами команди є основою методології;
- Завдання розподілені по часу. У основі розділення на невеликі задачі, що займуть не дуже багато часу. Так, є можливість швидко переходити від однієї задачі до іншої, відстежуючи прогрес;
- Мета кожної зустрічі полягає у обговоренні наступних та виконаних задач, плануванні розподілення обов'язків, формування логу виконаних завдань.

2.6.4. Виклик та завантаження програми

Виклик програми відбувається у середовищі Google Colaboratory.

Напочатку, завантажується файл із даними, що мають використовуватися для навчання нейронної мережі:



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
1	time (Ma),MOR_length (km),production (km2/Ma),10Ma_mean_length (km),10Ma_mean_production (km2/Ma)																			
2	0.0,68124.64147181314,3177244.824427844,66525.6859347837,3278211.4112977297																			
3	1.0,65303.873397543015,3005635.033553859,66691.4586339647,3321416.4818657967																			
4	2.0,65785.16126613306,3097067.4818681953,66971.6725234065,3365637.3787025455																			
5	3.0,65373.11556126972,3132413.9227125156,66948.96083379514,3389076.3313174746																			
6	4.0,65008.65491501787,3202865.006660606,66925.87554544205,3417728.9068457056																			
7	5.0,63950.366804826735,3285037.9930171664,66946.86999378868,3438085.277968176																			
8	6.0,69035.69662378583,3571857.0840990096,67639.58548741868,3498679.6157279206																			
9	7.0,68198.54791513816,3446169.7779272627,67613.70019607758,3511167.071558542																			
10	8.0,67277.66852116393,3436650.587707557,67870.76583052642,3532909.682136398																			
11	9.0,67199.13287114538,3427172.401003277,68146.5345960298,3558469.868789059																			
12	10.0,69782.36846362328,3609295.5301085124,68517.15791795371,3592157.889775642																			
13	11.0,68106.01229196097,3447844.001921352,68589.8238805848,3585473.1681749895																			
14	12.0,65558.0443700194,3331457.008017484,69079.3969431414,3629842.5180747313																			
15	13.0,65142.2626777389,3418939.677994826,69775.52579782046,3689740.5512896264																			
16	14.0,65218.599398484206,3406428.717885309,70379.49212562664,3706353.07553788																			
17	15.0,70877.5217411266,3890981.3706146097,70912.57337695143,3747112.7126402673																			
18	16.0,68776.84371037493,3696731.642405223,70798.69939336239,3724518.381520129																			
19	17.0,70769.20425962644,3663595.8837058274,71630.69034699092,3800488.9355741236																			
20	18.0,70035.35617619783,3692252.4542341633,72303.12869834466,3870097.6781522622																			
21	19.0,70905.36609038452,3764052.6108691082,72567.27817786936,3942303.8197994092																			
22	20.0,70509.0280899341,3542448.314101985,72876.3957328851,4011477.417462118																			
23	21.0,73001.74291752698,3891537.500918767,73289.00881019968,4091075.897450749																			
24	22.0,72519.33291680997,3930437.340166444,73081.38393526679,4095831.7966978517																			
25	23.0,71181.92595580069,3585064.920477357,72965.01421660351,4097922.064612309																			

Рис. 2.5 Файл CSV із вхідними даними

Цей набір даних завантажується у систему за допомогою Keras API та імпортується у датасет для подальшої обробки системою.

2.6.5. Опис інтерфейсу користувача

Після того як дані завантажено, розподілено та оброблено, створюється каркас нейронних мереж.

Перша модель Baseline цілком наслідується від підкласу ‘Model’ Keras API:

```
class Baseline(tf.keras.Model):
```

Оскільки при розробці моделі було створено декілька нейронних архітектур, було вирішено створити загальну функцію для навчання моделі, яку буде використовувати кожний тип моделі, щоб не прописувати кожній

функцію окремо. Як функцію підрахунку похибки, функція тренування використовує

tf.losses.MeanSquaredError()

Ця утиліта вираховує квадрат помилки між заданими значеннями та припущеними значеннями.

Для оптимізації система використовує алгоритм Adam:

tf.keras.optimizers.Adam

Оптимізація за допомогою цього алгоритму — це метод стохастичного градієнтного спуску, який базується на адаптивній оцінці моментів першого та другого порядку.

Після створення екземпляру класу та його тренування, відображаємо результат на екрані за допомогою бібліотеки Python Matplotlib:

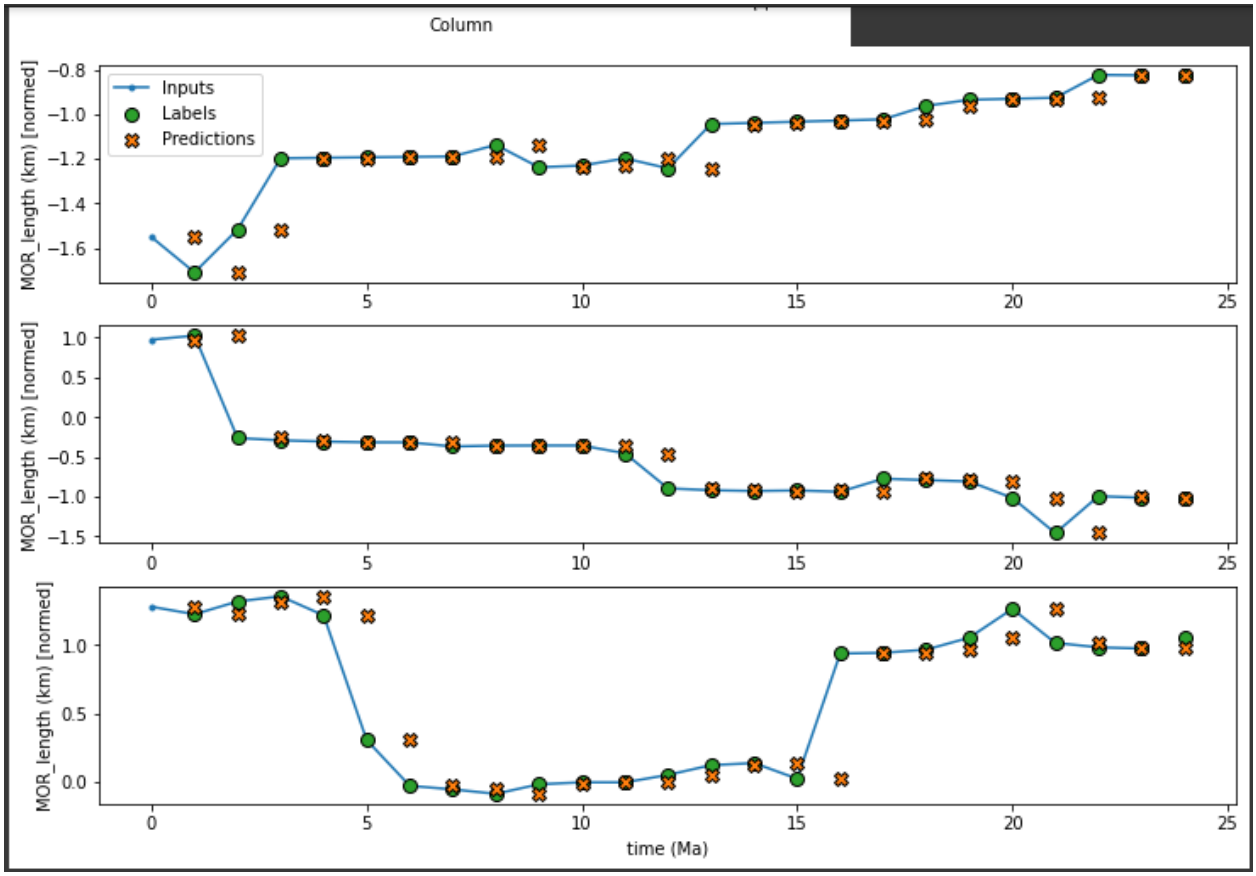


Рис. 2.6 Результат моделі Baseline

У цьому прикладі модель ще не робить ніяких припущень щодо даних майбутнього, адже вона ще не може будувати часові ряди. У даному випадку, модель лише навчається на заданих даних та робить своє припущення щодо даних, які вона ще не бачила, але які є у системі. Тобто, нейронна мережа тренується та видає результат на тестових даних, які ми не давали їй для навчання.

Слід зазначити, що результат моделі (припущення значень) є досить точним. Іноді, модель не дуже точно попадає у точку, або будує припущення зовсім далеке, але у загальному вигляді архітектура показала себе добре.

Наступна модель Linear наслідується від об’єкту Keras API ‘Sequential’. Це більш складна система, що може мати декілька шарів усередині. Але у нашому Linear блоці поки що тільки один шар. Свого роду це одношаровий перцептрон, який не дуже добре вміє оброблювати дані. Ось такий результат вийшов при виконанні цього методу:

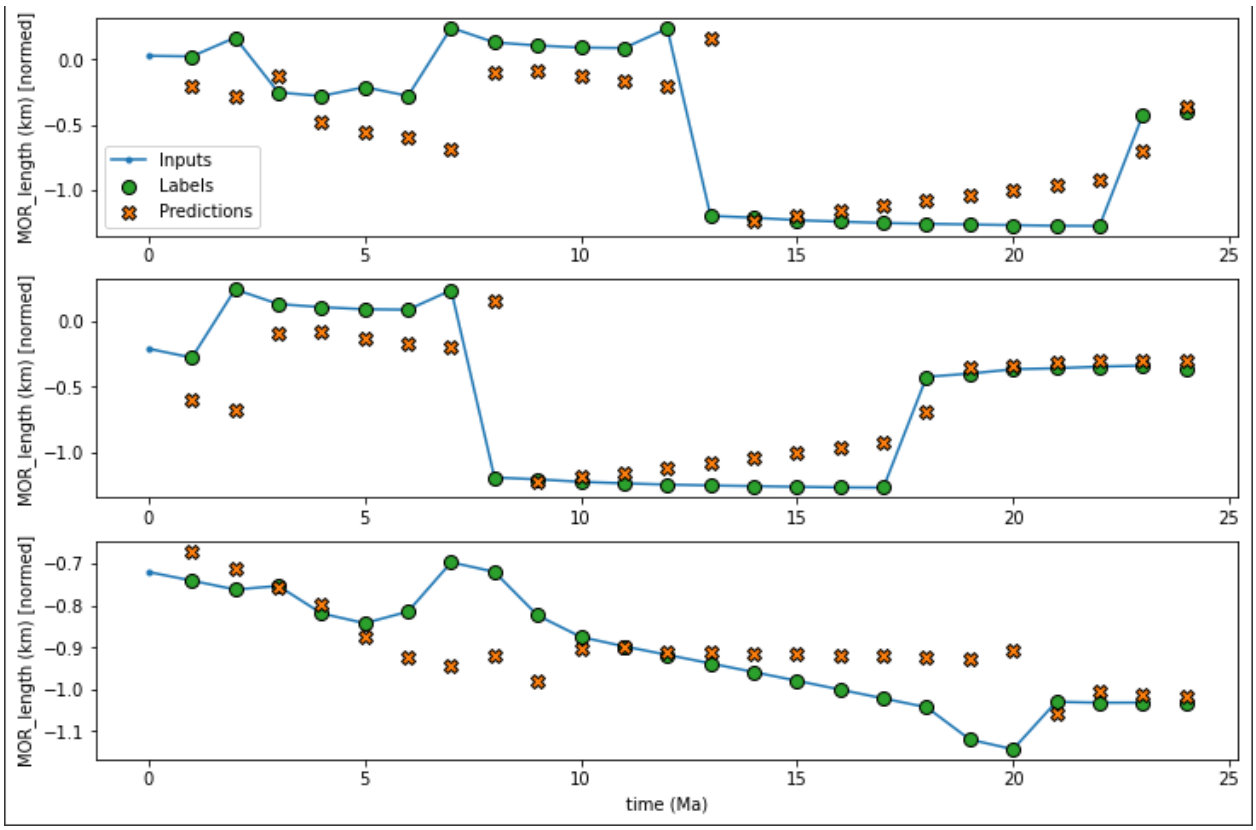


Рис. 2.7 Результат Linear моделі

Можна подивитись на навчання моделі та побачити, що результат доволі непоганий:

```
Epoch 1/20
22/22 [==============================] - 2s 19ms/step - loss: 0.1452 -
mean_absolute_error: 0.3054 - val_loss: 0.2341 - val_mean_absolute_error:
0.4108
Epoch 2/20
22/22 [==============================] - 0s 8ms/step - loss: 0.1284 -
mean_absolute_error: 0.2893 - val_loss: 0.1730 - val_mean_absolute_error:
0.3382
Epoch 3/20
22/22 [==============================] - 0s 8ms/step - loss: 0.1196 -
mean_absolute_error: 0.2793 - val_loss: 0.1455 - val_mean_absolute_error:
0.2990
Epoch 4/20
22/22 [==============================] - 0s 7ms/step - loss: 0.1133 -
mean_absolute_error: 0.2706 - val_loss: 0.1299 - val_mean_absolute_error:
0.2752
Epoch 5/20
22/22 [==============================] - 0s 6ms/step - loss: 0.1078 -
mean_absolute_error: 0.2627 - val_loss: 0.1185 - val_mean_absolute_error:
0.2563
Epoch 6/20
```

```
22/22 [==============================] - 0s 6ms/step - loss: 0.1026 -
mean_absolute_error: 0.2551 - val_loss: 0.1101 - val_mean_absolute_error:
0.2423
Epoch 7/20
22/22 [==============================] - 0s 6ms/step - loss: 0.0979 -
mean_absolute_error: 0.2479 - val_loss: 0.1038 - val_mean_absolute_error:
0.2316
Epoch 8/20
22/22 [==============================] - 0s 8ms/step - loss: 0.0937 -
mean_absolute_error: 0.2412 - val_loss: 0.0963 - val_mean_absolute_error:
0.2177
Epoch 9/20
22/22 [==============================] - 0s 6ms/step - loss: 0.0896 -
mean_absolute_error: 0.2347 - val_loss: 0.0902 - val_mean_absolute_error:
0.2061
Epoch 10/20
22/22 [==============================] - 0s 6ms/step - loss: 0.0860 -
mean_absolute_error: 0.2285 - val_loss: 0.0859 - val_mean_absolute_error:
0.1984
Epoch 11/20
22/22 [==============================] - 0s 7ms/step - loss: 0.0826 -
mean_absolute_error: 0.2226 - val_loss: 0.0813 - val_mean_absolute_error:
0.1892
Epoch 12/20
22/22 [==============================] - 0s 8ms/step - loss: 0.0795 -
mean_absolute_error: 0.2171 - val_loss: 0.0780 - val_mean_absolute_error:
0.1829
Epoch 13/20
22/22 [==============================] - 0s 7ms/step - loss: 0.0767 -
mean_absolute_error: 0.2119 - val_loss: 0.0747 - val_mean_absolute_error:
0.1758
Epoch 14/20
22/22 [==============================] - 0s 7ms/step - loss: 0.0742 -
mean_absolute_error: 0.2068 - val_loss: 0.0722 - val_mean_absolute_error:
0.1710
Epoch 15/20
22/22 [==============================] - 0s 8ms/step - loss: 0.0717 -
mean_absolute_error: 0.2022 - val_loss: 0.0684 - val_mean_absolute_error:
0.1611
Epoch 16/20
22/22 [==============================] - 0s 8ms/step - loss: 0.0694 -
mean_absolute_error: 0.1977 - val_loss: 0.0669 - val_mean_absolute_error:
0.1587
Epoch 17/20
22/22 [==============================] - 0s 7ms/step - loss: 0.0674 -
mean_absolute_error: 0.1934 - val_loss: 0.0643 - val_mean_absolute_error:
0.1516
Epoch 18/20
22/22 [==============================] - 0s 8ms/step - loss: 0.0654 -
mean_absolute_error: 0.1893 - val_loss: 0.0627 - val_mean_absolute_error:
0.1478
Epoch 19/20
22/22 [==============================] - 0s 8ms/step - loss: 0.0636 -
mean_absolute_error: 0.1856 - val_loss: 0.0611 - val_mean_absolute_error:
0.1441
Epoch 20/20
22/22 [==============================] - 0s 6ms/step - loss: 0.0619 -
mean_absolute_error: 0.1819 - val_loss: 0.0596 - val_mean_absolute_error:
0.1404
```

При виконанні тренування у циклі з 20 повтореннями можна побачити тенденцію зменшення середньої та стандартної похибок. У результаті, система може допускати похибку 0.0619, що у два рази менше у порівнянні із похибкою напочатку циклу (0.1452).

Наступна модель також наслідується від Sequential утиліти Keras. Ця модель на відміну від попередніх має 3 шари, кожен з яких містить функцію активації ReLu.

Подивимось на результат обчислень:

```
Epoch 1/20
22/22 [==============================] - 1s 23ms/step - loss: 0.4603 -
mean_absolute_error: 0.5166 - val_loss: 0.2059 - val_mean_absolute_error:
0.3805
Epoch 2/20
22/22 [==============================] - 0s 8ms/step - loss: 0.0681 -
mean_absolute_error: 0.2044 - val_loss: 0.0336 - val_mean_absolute_error:
0.1111
Epoch 3/20
22/22 [==============================] - 0s 8ms/step - loss: 0.0344 -
mean_absolute_error: 0.1331 - val_loss: 0.0347 - val_mean_absolute_error:
0.1129
Epoch 4/20
22/22 [==============================] - 0s 8ms/step - loss: 0.0297 -
mean_absolute_error: 0.1156 - val_loss: 0.0325 - val_mean_absolute_error:
0.1050
Epoch 5/20
22/22 [==============================] - 0s 7ms/step - loss: 0.0282 -
mean_absolute_error: 0.1082 - val_loss: 0.0288 - val_mean_absolute_error:
0.0915
Epoch 6/20
22/22 [==============================] - 0s 6ms/step - loss: 0.0277 -
mean_absolute_error: 0.1060 - val_loss: 0.0279 - val_mean_absolute_error:
0.0926
Epoch 7/20
22/22 [==============================] - 0s 9ms/step - loss: 0.0269 -
mean_absolute_error: 0.1041 - val_loss: 0.0291 - val_mean_absolute_error:
0.0946
Epoch 8/20
22/22 [==============================] - 0s 6ms/step - loss: 0.0266 -
mean_absolute_error: 0.1027 - val_loss: 0.0294 - val_mean_absolute_error:
0.0972
```

Тут похибка досягає дуже низького значення вже на 8 циклі. Зменшившись у два рази за цей час мі можемо побачити такі передбачення:

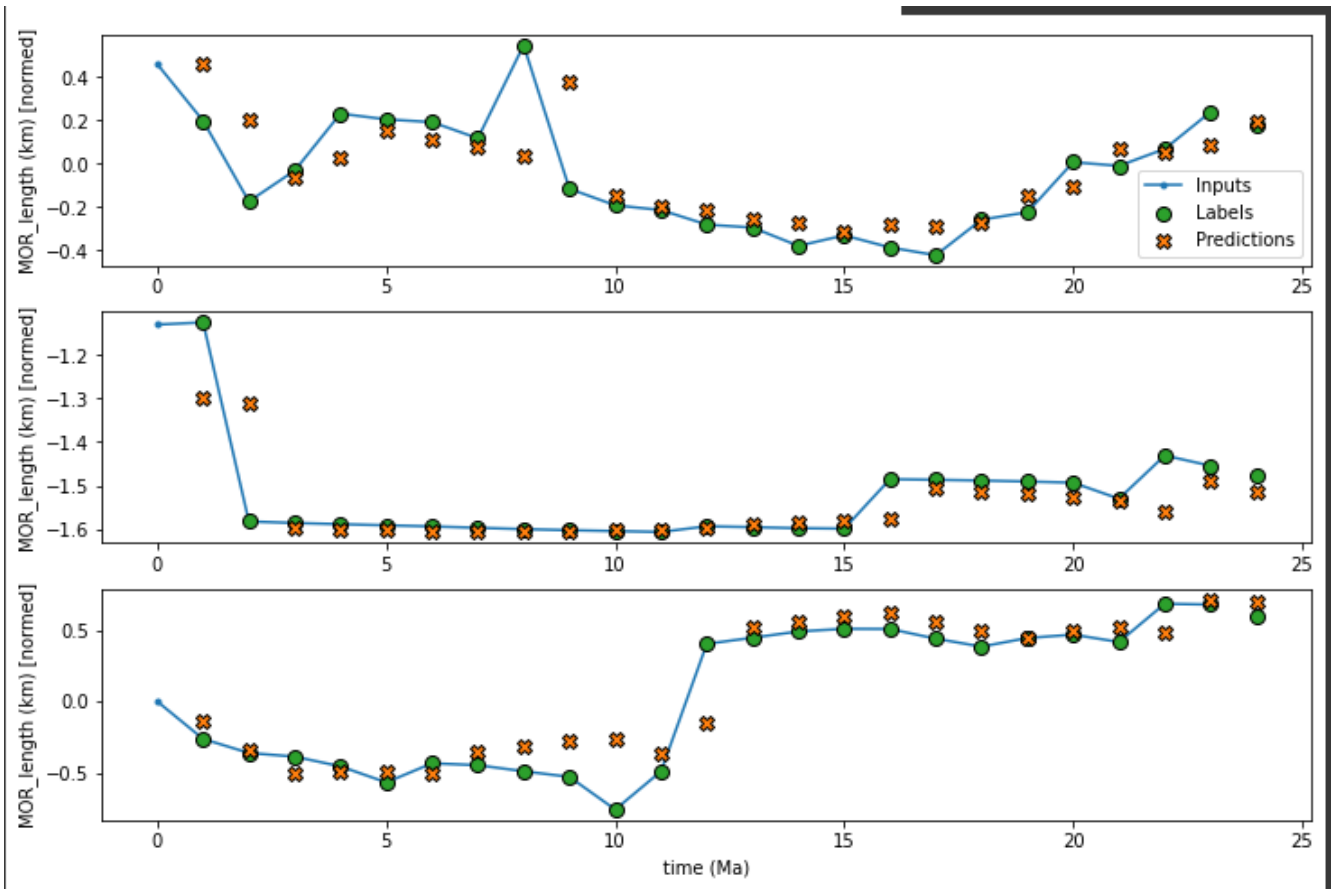


Рис. 2.8 Результат 3-шарової моделі

Дані отримані цією мережею поки що найточніші.

Модель CNN показує гірші результати:

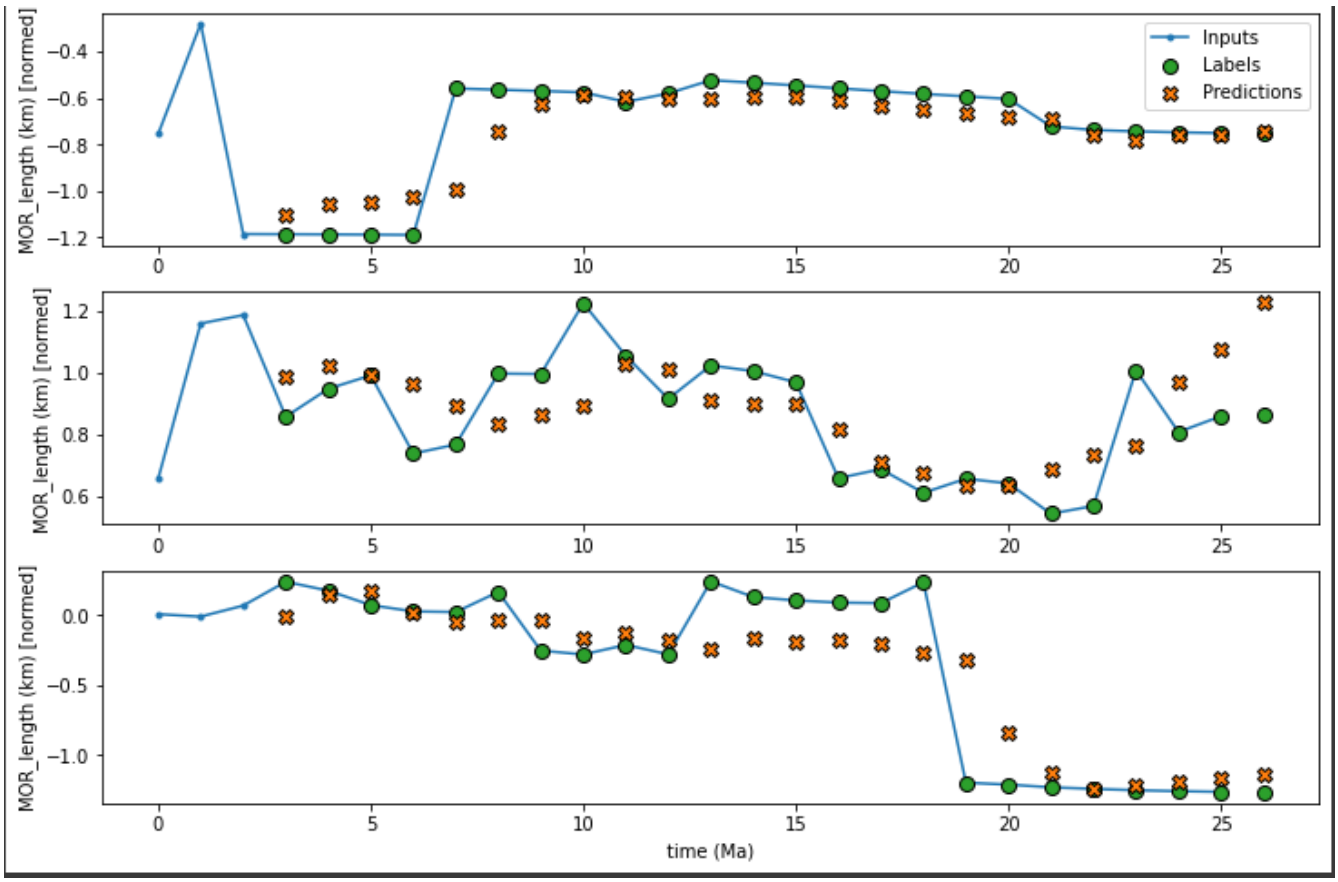


Рис. 2.9 Результат CNN моделі

Наступною для прогнозування була побудована RNN LSTM модель.

Фінальне значення похибки після проходження 20 циклів досягло 0.0791. При цьому дані на графіку досить чіткі:

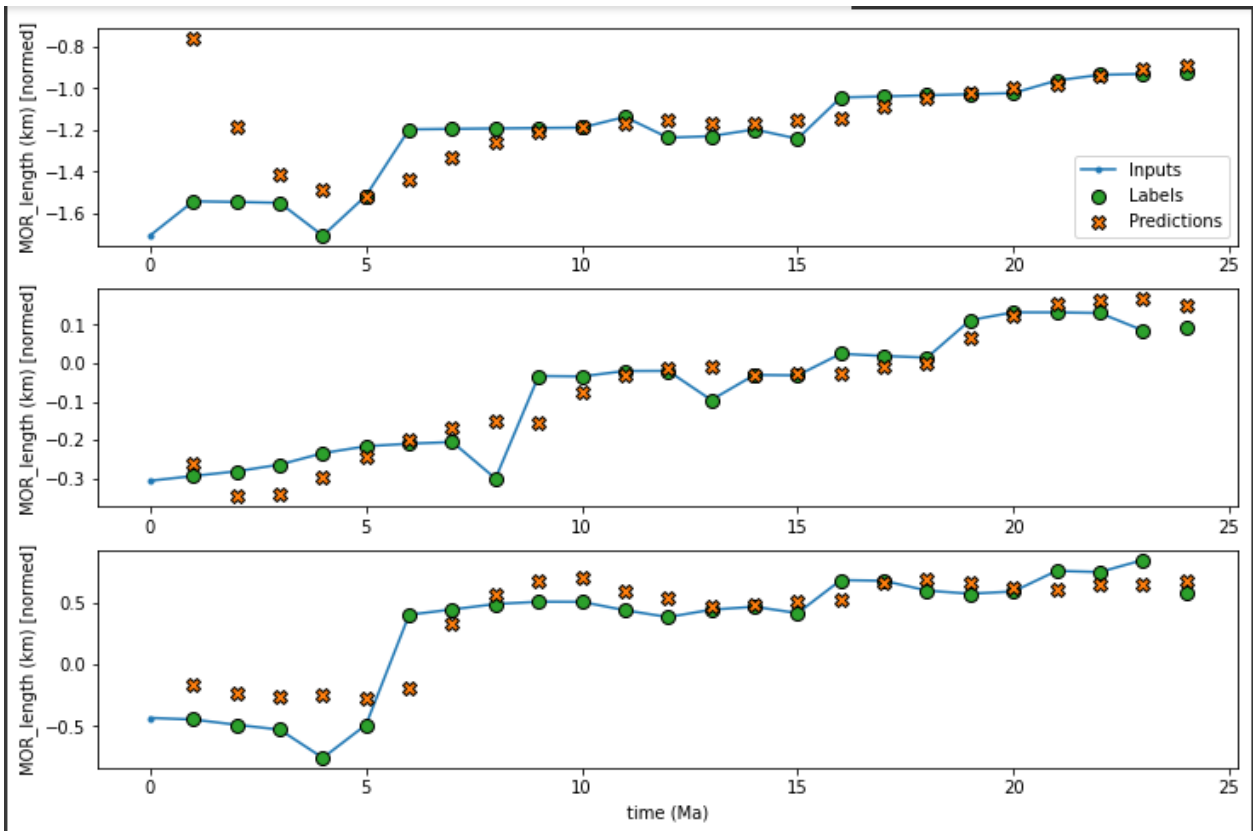


Рис. 2.10 Результат моделі LSTM

Так як задачею проекту є побудова передбачень деяких характеристик планети у майбутньому, система має вміти будувати часові ряди та самостійно без відомих даних робити прогнозування значень.

У багатоетапному прогнозуванні модель має навчитися прогнозувати діапазон майбутніх значень. Таким чином, на відміну від одноступінчастої моделі, в якій передбачається лише одна точка майбутнього, багатоступінчаста модель передбачає послідовність майбутніх значень.

Є два грубі підходи до цього:

- Прогнози одиночного пострілу, коли весь часовий ряд прогнозується одразу.
- Прогнози авторегрессии, у яких модель робить лише однокрокові прогнози, та її вихідні дані повертаються як вхідних даних.

Вдосконалена модель Baseline лише повторює результат даних, беручи за основу існуючі дані, до яких вона має доступ:

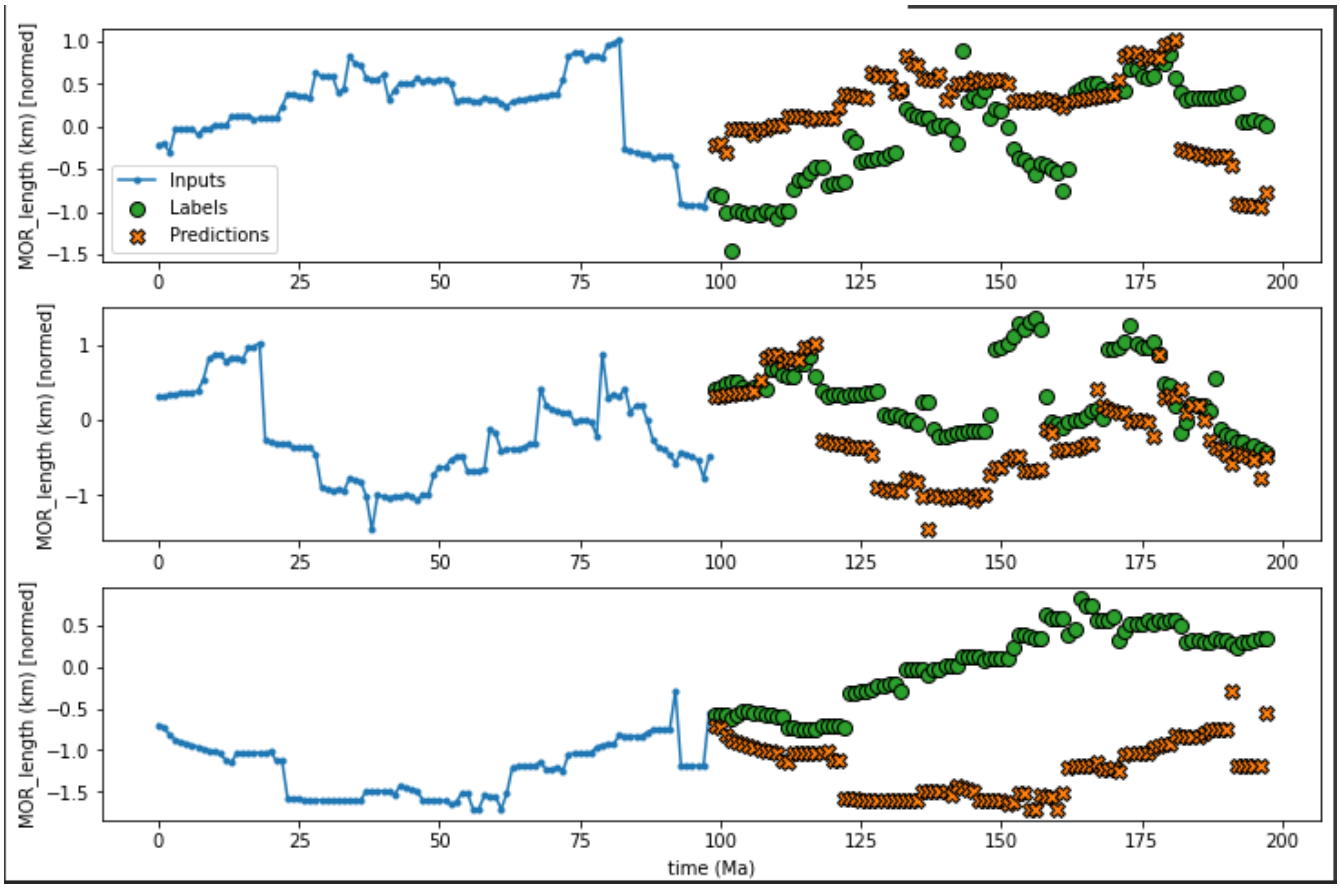


Рис. 2.11 Результат багатошарової моделі Baseline

Якщо подивитися на результат (predictions), то можна помітити що значення повторюють графік вхідних значень (inputs). Задавши менший часовий діапазон це стає очевидним:

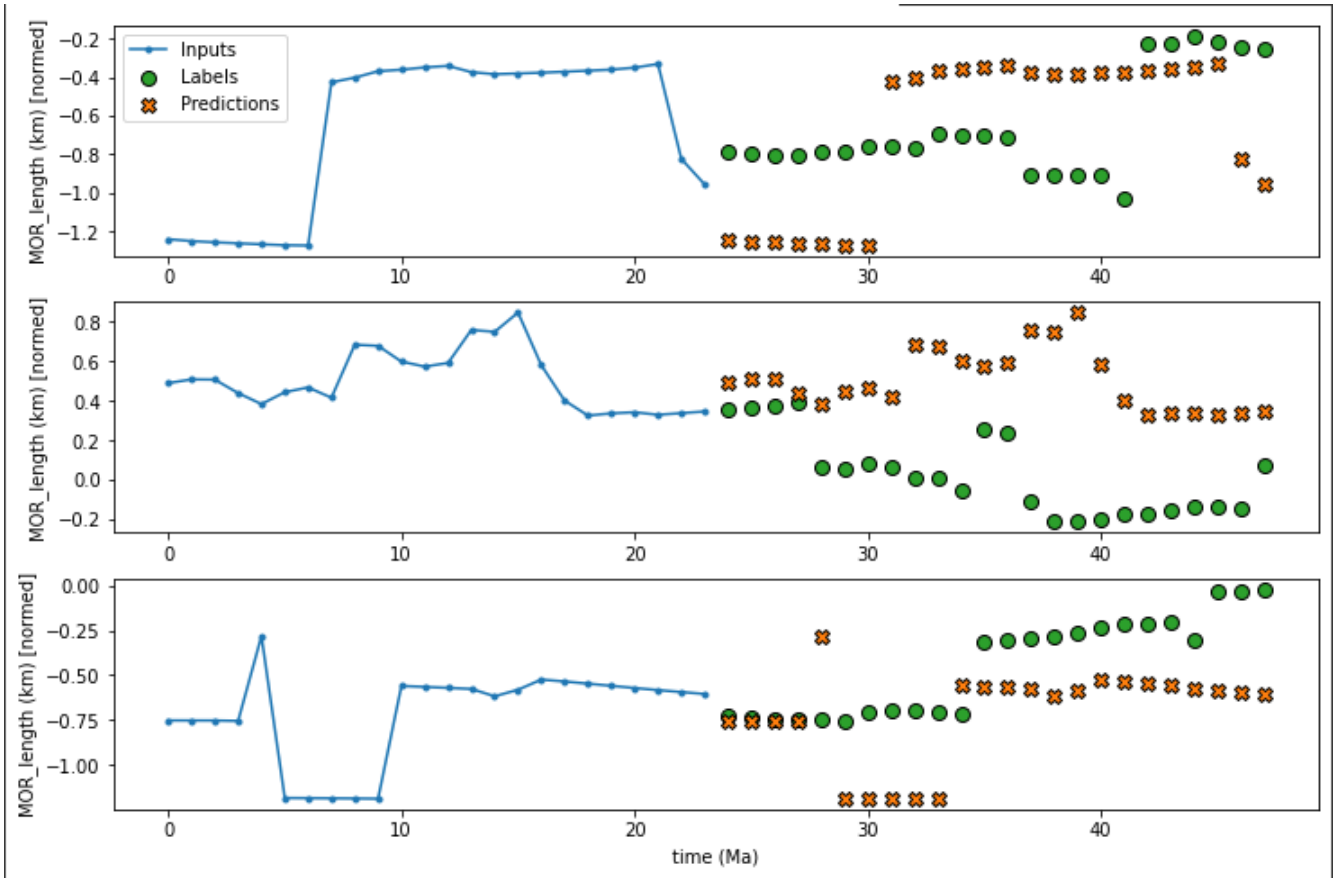


Рис. 2.12 Менший діапазон значень для багатошар. Baseline

Модель робить такий прогноз, беручи за істину доволі примітивну логіку: якщо так було у минулому, то так само має повторитися у майбутньому.

Багатоступінчаста модель Linear не повторює графік існуючих значень, але її показники все ще неточні:

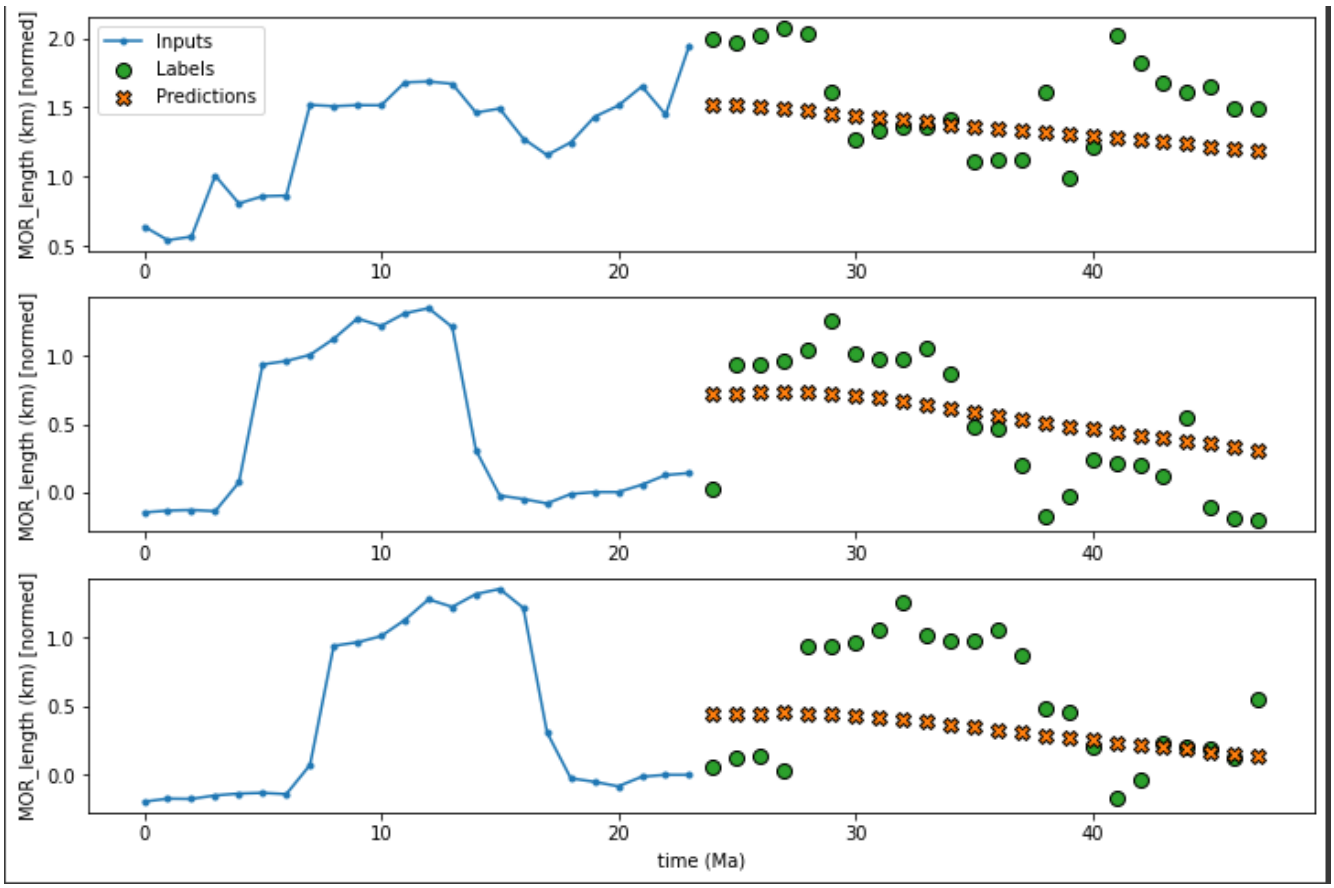


Рис. 2. 13 Результат багатоступ. Моделі Linear

Модель просто неспроможна адаптуватися під гнучкі зміни даних та змоделювати підходящий графік.

Багатоступінчаста модель Dense показує непоганий результат похибки після обчислень: 0.2450.

Результат не можна назвати ідеальним, модель може відстежувати тенденцію, але не вказує точних значень:

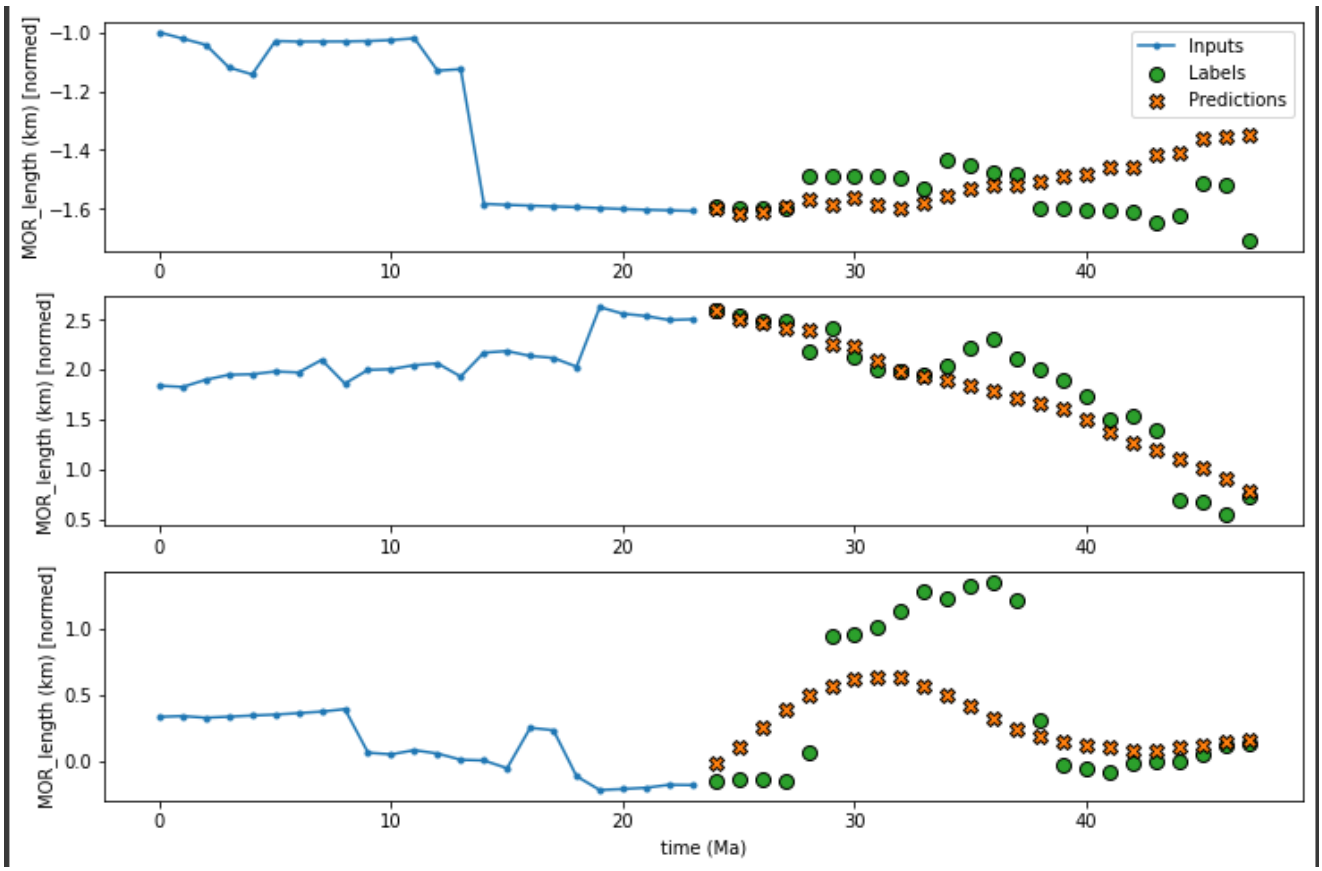


Рис. 2.14 Результат багатоступ. Моделі Dense

Якщо подивитись на більш широкому часовому діапазоні результат стане більш зрозумілим:

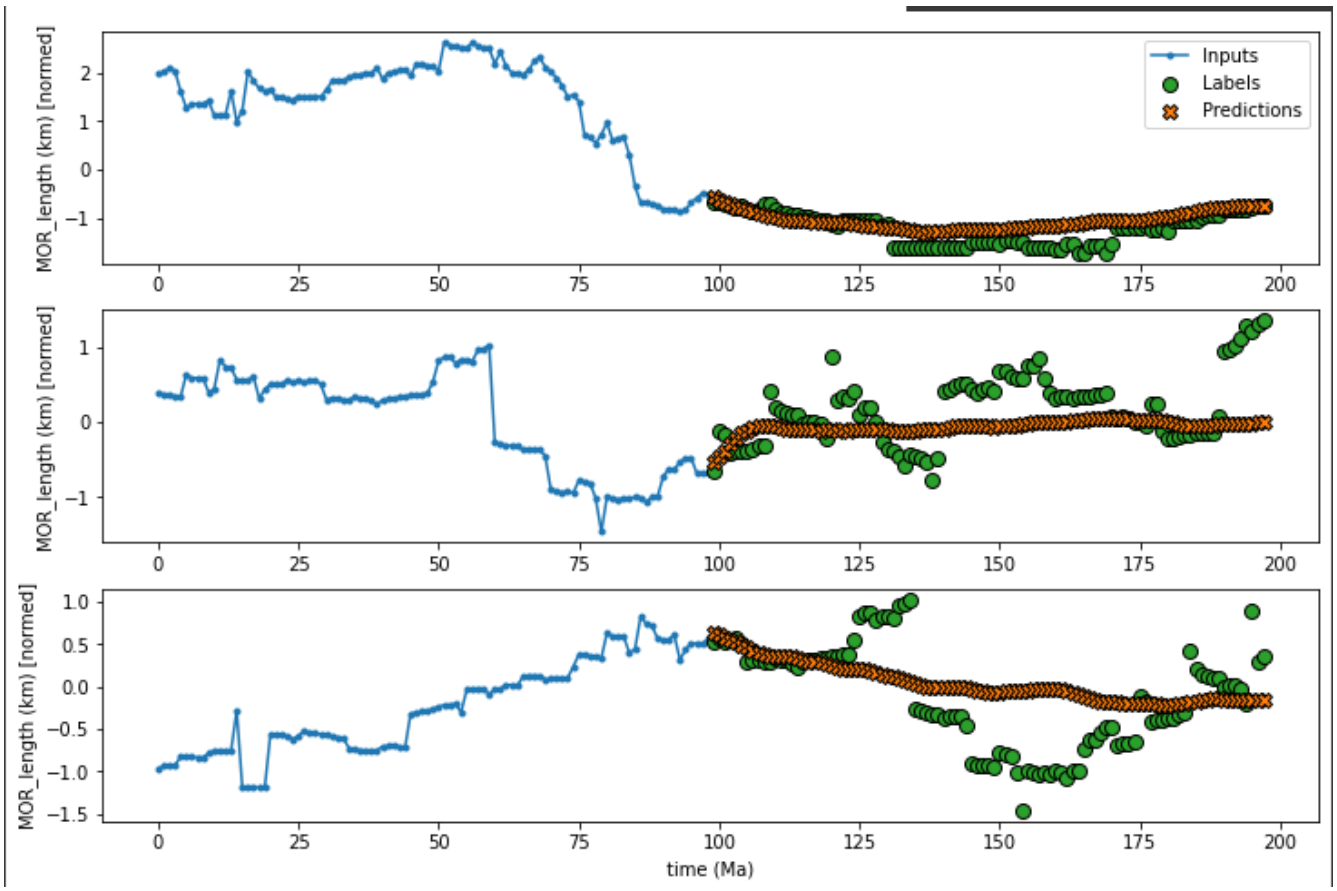


Рис. 2.15 Результат багатоступ. Моделі Dense на широкому ч.д.

CNN модель також буда змінена до багатоступінчастої. Результат похибки після обчислень дуже близький до значення похибки багатоступінчастої Dense моделі. Значення-прогноз також дуже схожий до графіку Dense:

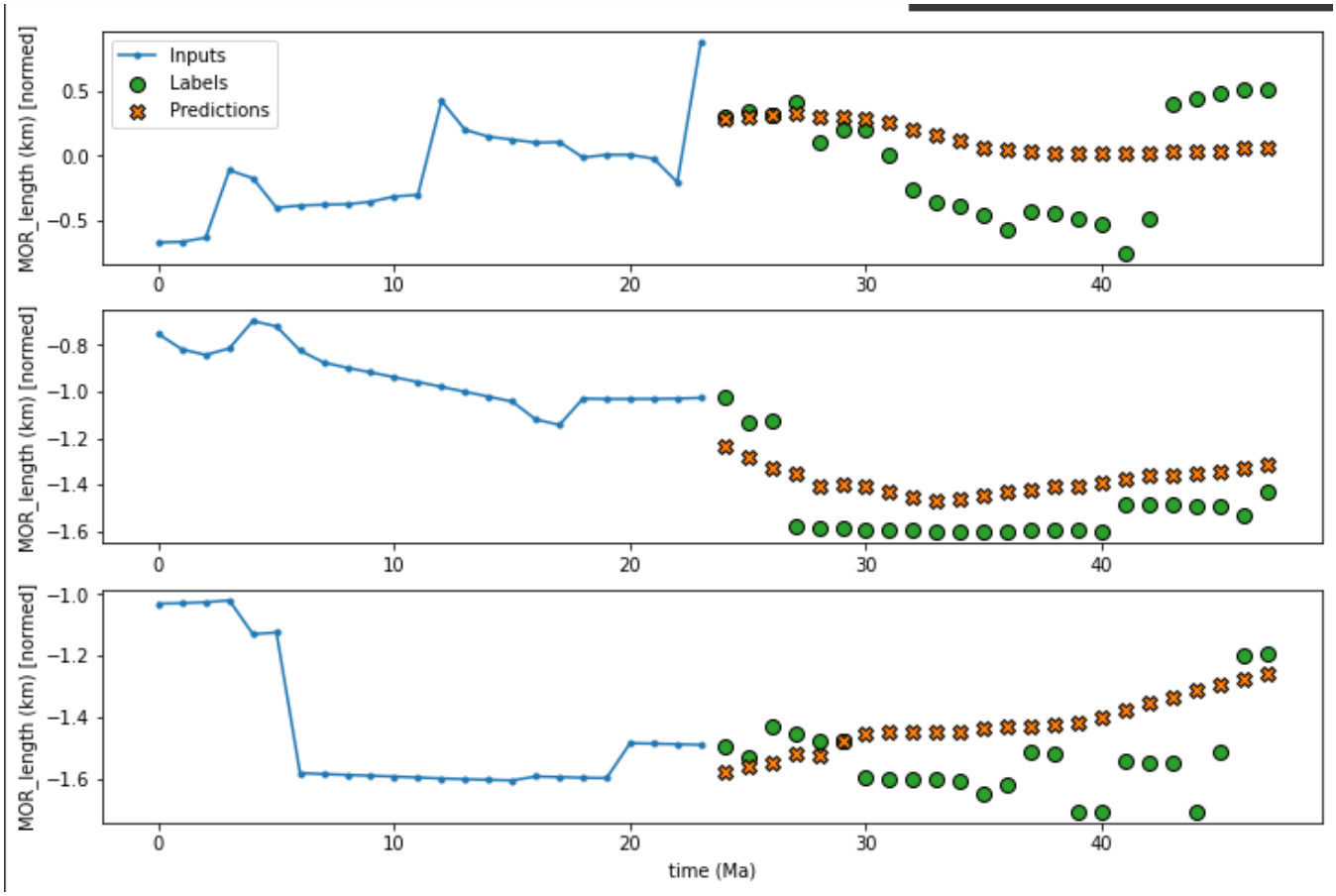


Рис. 2.16 Результат багатоступ. Моделі CNN

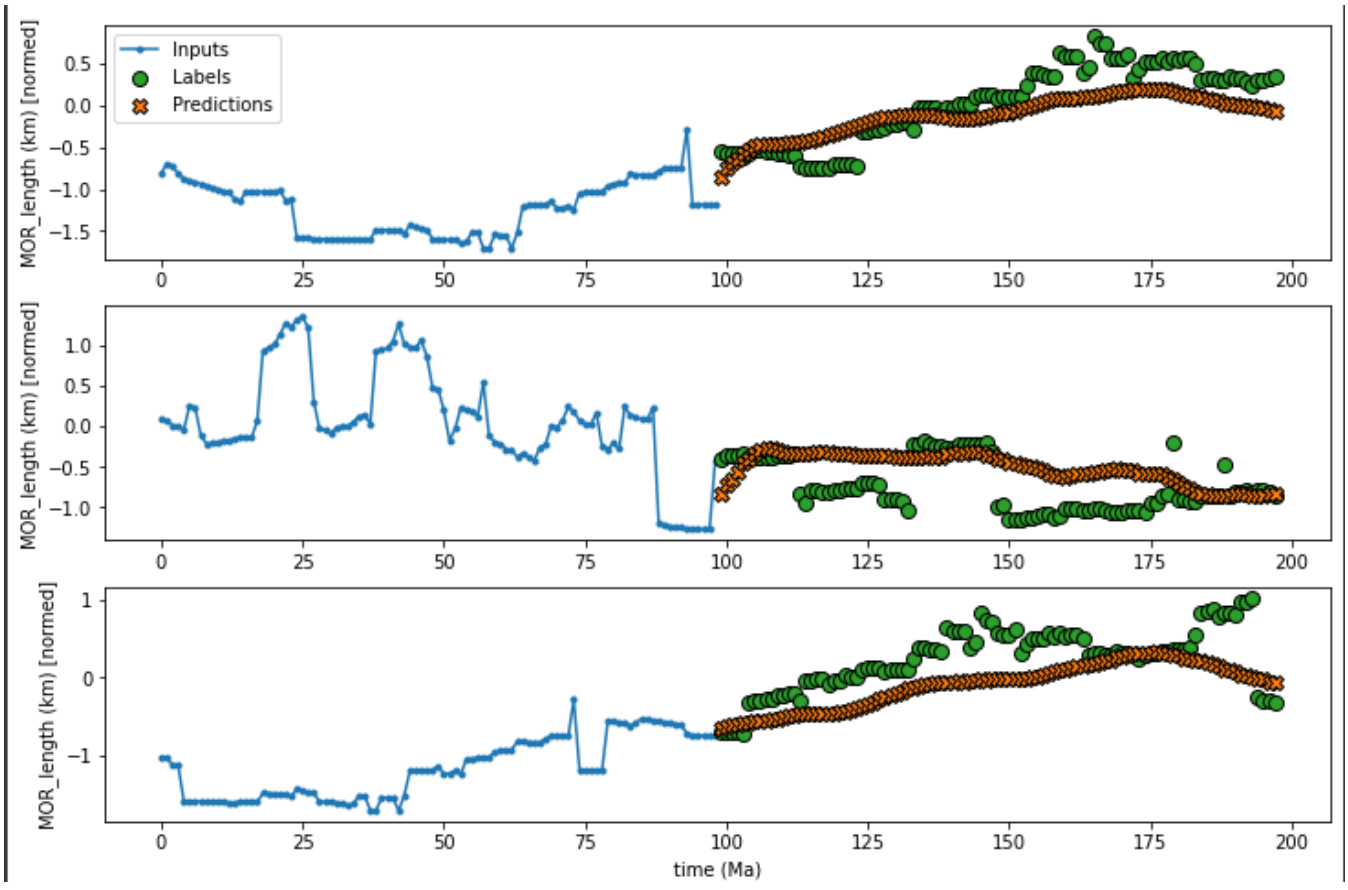


Рис. 2.17 Багатоступ. CNN на більшому ч.д

LSTM модель показала такі результати:

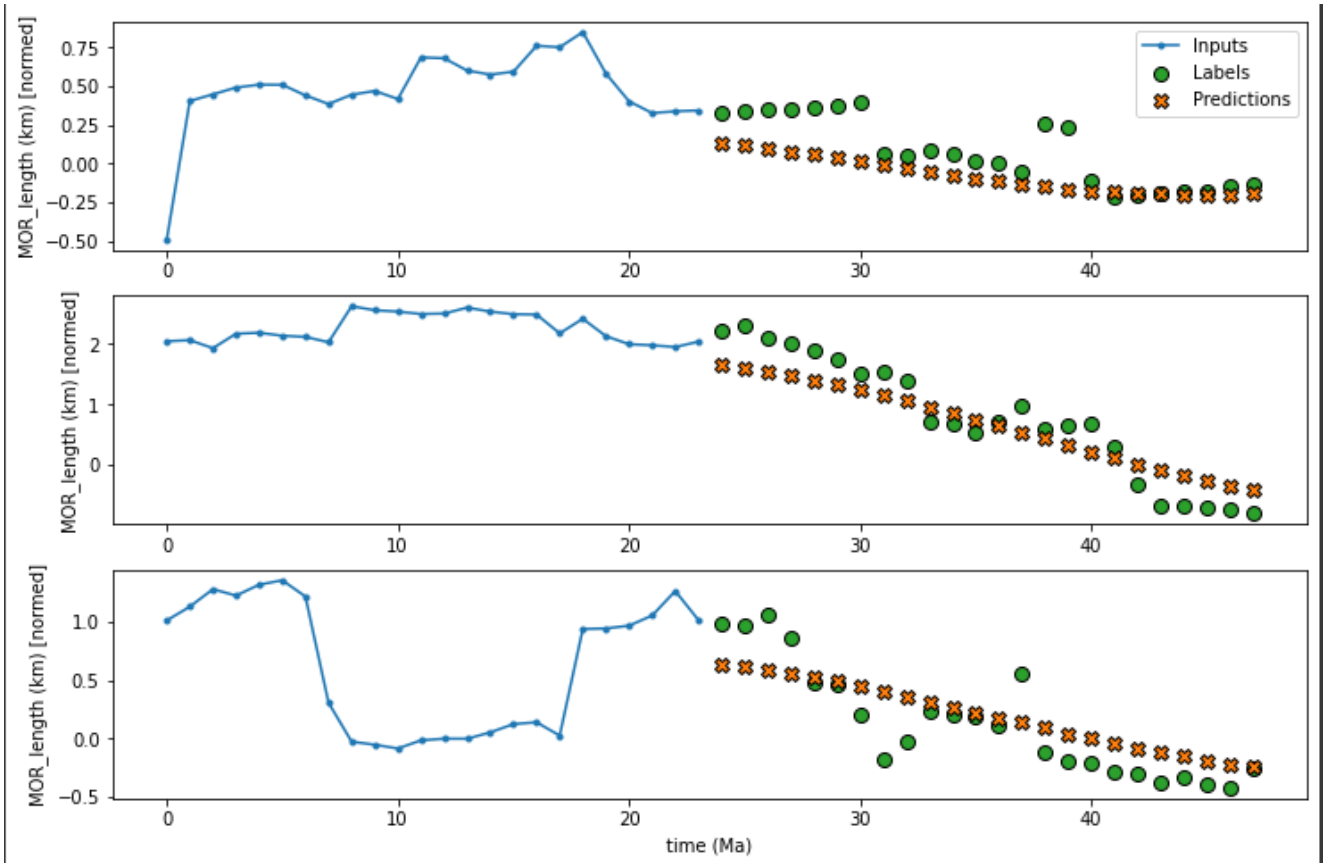


Рис. 2.18 Результат багатоступ. Моделі LSTM

Усі перелічені моделі пророкують всю вихідну послідовність за один крок.

У деяких випадках для моделі може бути корисно розкласти цей прогноз на окремі кроки. Потім вихідні дані кожної моделі можуть бути повернені в себе на кожному кроці, і прогнози можуть бути зроблені в залежності від попереднього кроку.

Для побудови авторегресійної моделі за основу взята RNN. Після обчислень і розрахунків була вирахувана похибка 0.4860. Це не найменший досягнутий результат, на графіку дані виглядають так:

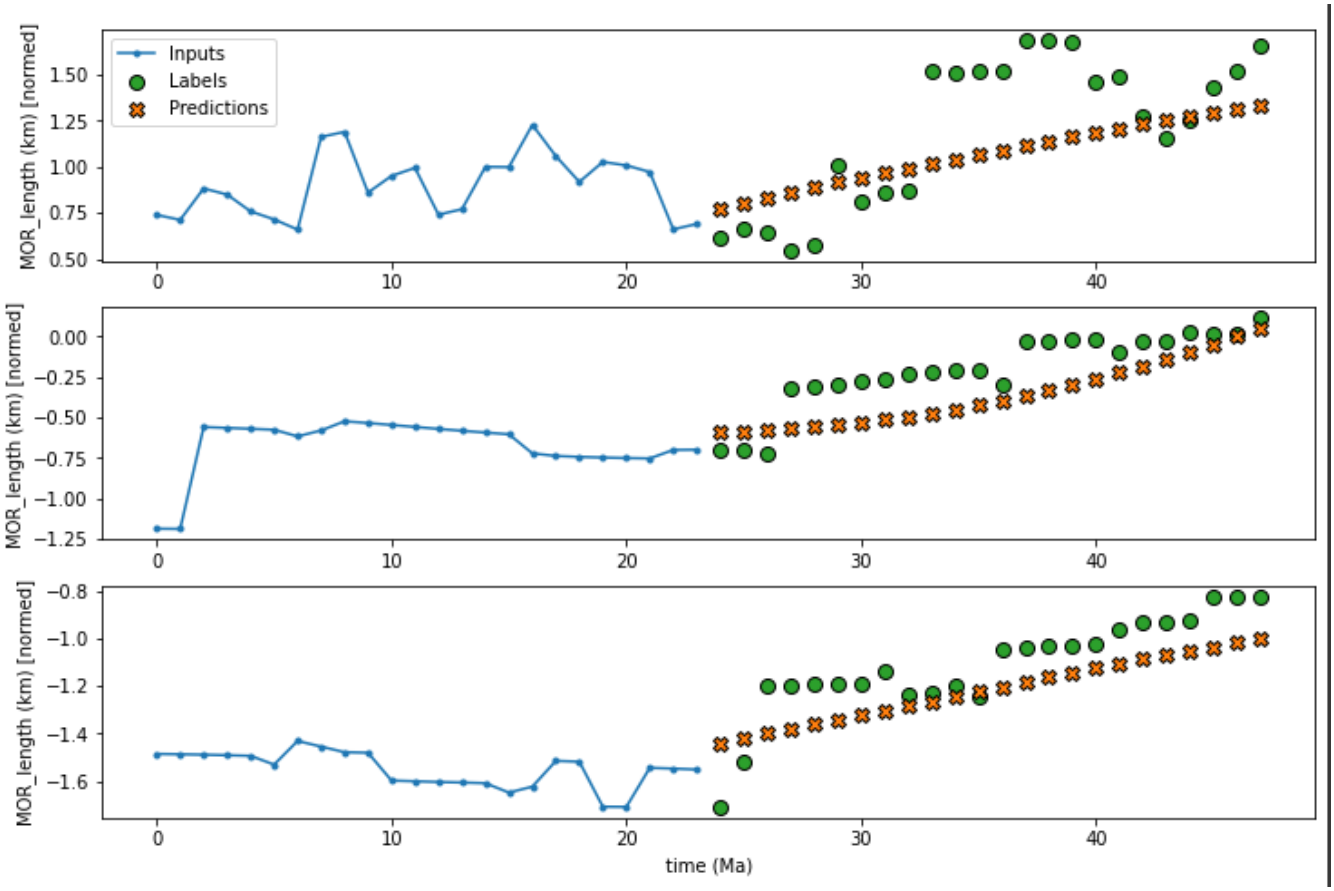


Рис. 2.19 Результат авторегрес. Моделі RNN

Таким чином, найменшої похибки довелось досягнути мережі із архітектурою Dense. Це мережа із двома слоями, із функцією активації ReLU.

Також непогані результати у обчисленні значень за створенні припущень показали багатоступінчаста LSTM та авторегресійна RNN мережі.

Для створення веб-застосунку я буду використовувати результати саме одної з цих архітектур, адже вони найкраще підходять саме для мети цієї кваліфікаційної роботи.

2.6.6. Прогнозні припущення про розвиток об’єкту розробки

У перспективі у систему будуть додано різновид інших параметрів як вхідних даних для більш точного та розширеного прогнозування у майбутньому. На основі заданих даних, система зможе прогнозувати не лише значення для параметрів планети Земля, а і для інших планет нашої сонячної системи. Оскільки ми маємо інформацію із супутників інших планет про склад атмосфери, поверхні планети, складу повітря та факту наявності чи відсутності води та інших природних речовин, вона може бути оброблена системою. У результаті, можна буде отримати дані про стан будь-якої сонячної планети, про яку у нас є дані, у майбутньому. Звичайно, інформації тільки про планету буде замало. Щоб отримати точний прогноз, треба мати на увазі також фактори загальної сонячної системи такі як викиди радіації, розмір сонця, стан супутників планети.

Ці параметри можуть бути додані до системи, вони можуть бути оброблені нею та використані для навчання нової нейронної мережі, яка за допомогою них буде здатна зробити власний прогноз.

РОЗДІЛ 3

ЕКОНОМІЧНИЙ РОЗДІЛ

3.1. Розрахунок трудомісткості та вартості розробки програмного продукту

Початкові дані:

1. передбачуване число операторів програми – 3853;
2. коефіцієнт складності програми – 1,3;
3. коефіцієнт корекції програми в ході її розробки – 0,07;
4. годинна заробітна плата програміста – 233 грн/год;

Середня годинна зарплата програміста була вирахувати основуючись на даних порталу «Української спільноти програмістів (DOU)». Середньоукраїнська заробітна плата програміста, який пише програми на мові програмування Python, стандартних бібліотек та фреймворків та має досвід роботи близько року дорівнює 1500 американських доларів в місяць. При курсі валют НБУ на початок червня 2022 року один американський долар дорівнює 27,34 грн, тому середня зарплата в гривнях дорівнює 41010 грн. При восьмигодинному робочому дні (в середньому, 176 годин в місяць) середня зарплата за годину буде дорівнювати 233 грн.

Джерела:

− Веб-сайт DOU «Українська спільнота програмістів»,

URL:
https://jobs.dou.ua/salaries/#period=dec2020&city=all&title=Software%20Eng ineer&language=Java&spec=&exp1=0&exp2=1

− Веб-сайт НБУ,

URL: https://bank.gov.ua/ua/markets/exchangerate-chart?cn%5B%5D=USD

5. коефіцієнт збільшення витрат праці внаслідок недостатнього опису задачі – 1,2;

6. коефіцієнт кваліфікації програміста, в залежності від стажу роботи з даної спеціальності – 1,3;

7. вартість машино-години ЕОМ – 13,93 грн/год.

Звичайний комп'ютер споживає 180 Ватт / год, що дорівнює 0,18 кВт / год. Вартість одного кВт на годину від постачальника Yasno на стан середини 2022 року дорівнює 1,68 грн в незалежності від обсягу споживання. Тобто вартість електроенергії споживаної комп'ютером за годину є 0,30 грн.

Ціна довгострокової оренди комп'ютера дорівнює 2400 грн на місяць, з чого можна вирахувати, що ціна оренди на годину буде дорівнювати 13,63 грн.

Остаточна вартість машино-години ЕОМ дорівнює 13,93 грн за годину. Джерела:

− Веб-сайт постачальника електроенергії Yasno,

URL: https://yasno.com.ua/b2c-tariffs

− Веб-сайт компанії орендаря комп'ютерів «ChipChip»,

URL: https://komputers.com.ua/arenda-kompyutera/

Нормування праці в процесі створення ПЗ істотно ускладнено в силу творчого характеру праці програміста. Тому трудомісткість розробки ПЗ може бути розрахована на основі системи моделей з різною точністю оцінки. Трудомісткість розробки ПЗ можна розрахувати за формулою:

$$t = t_o + t_u + t_a + t_n + t_{отл} + t_д \text{, людино-годин,}$$

(3.1)

де t_o – витрати праці на підготовку й опис поставленої задачі (приймається 50);

t_u – витрати праці на дослідження алгоритму рішення задачі;

t_a – витрати праці на розробку блок-схеми алгоритму;

t_n – витрати праці на програмування по готовій блок-схемі;

$t_{отл}$– витрати праці на налагодження програми на ЕОМ;

$t_д$ – витрати праці на підготовку документації.

Складові витрати праці визначаються через умовне число операторів у ПЗ, яке розробляється.

Умовне число операторів (підпрограм):

$$Q = q \cdot C \cdot (1 + p) \text{ , де}$$

q – передбачуване число операторів;

С – коефіцієнт складності програми;

р – коефіцієнт корекції програми в ході її розробки.

$$Q = 3853 \cdot 1{,}3 \cdot (1 + 0{,}07) = 5359{,}52;$$

Витрати праці на вивчення опису задачі ti визначається з урахуванням уточнення опису і кваліфікації програміста:

$$tu = Q \cdot B\ (75\ldots85) \cdot K \text{ , людино-годин, де}$$

В – коефіцієнт збільшення витрат праці внаслідок недостатнього опису задачі;

К – коефіцієнт кваліфікації програміста, обумовлений стажем роботи з даної спеціальності;

tu = 5359,52 · 1,2 80 · 1,3 = 61,84, людино-годин.

Витрати праці на розробку алгоритму рішення задачі:

$ta = Q$ (20…25) · K ;

ta = 5359,52 24 · 1,3 = 171,77, людино-годин.

Витрати на складання програми по готовій блок-схемі:

$tn = Q$ (20…25)·K ;

tn = 5359,52 25 · 1,3 = 164,9, людино-годин.

Витрати праці на налагодження програми на ЕОМ: − за умови автономного налагодження одного завдання:

tотл = Q (4…5) · K ;

tотл = 5359,52 5 · 1,3 = 824,54, людино-годин, − за умови комплексного налагодження завдання:

tотл к = 1,2 · tотл;

tотл к = 1,2 · 824,54 = 1025,44, людино-годин

Витрати праці на підготовку документації:

$t\partial = t\partial p + t\partial o$; де

tдр – трудомісткість підготовки матеріалів і рукопису

$t\partial$р = Q (15…20)·K ;

$t\partial$р = 5359,52 17 · 1,3 = 242,51, людино-годин.

tдо– трудомісткість редагування, печатки й оформлення документації

tдо = 0,75·tдр ;

$t\partial o$ = 0,75 · 242,51 = 181,88, людино-годин.

$t\partial$ = 242,51 + 181,88 = 424,39, людино-годин.

У результаті, отримаємо трудомісткість розробки програмного забезпечення:

t = 50 + 61,84 + 171,77 + 164,9 + 824,54 + 424,39 = 1697,44, людино-годин.

Отже, розрахувавши, в загальній складності необхідно 1697,44 людино-годин для розробки даного програмного забезпечення.

3.2. Розрахунок витрат на створення програми

Витрати на створення ПЗ Кпо включають витрати на заробітну плату виконавця програми Зз/п і витрат машинного часу, необхідного на налагодження програми на ЕОМ.

КПО = ЗЗП + ЗМВ , грн,

де ЗЗП – заробітна плата виконавців, яка визначається за формулою:

ЗЗП = t·СПР , грн, де

t – загальна трудомісткість, людино-годин;

СПР – середня годинна заробітна плата програміста, грн/година

ЗЗП = 1697,44 · 233 = 395503,52, грн.

ЗМВ – Вартість машинного часу, необхідного для налагодження програми на ЕОМ:

ЗМВ = tотл ·СМ , грн, де

tотл – трудомісткість налагодження програми на ЕОМ, год.

СМЧ – вартість машино-години ЕОМ, грн/год.

ЗМВ = 1697,44 · 13,93 = 23645,33 грн.

КПО = 395503,52 + 23645,33 = 419148,85 грн.

Очікуваний період створення ПЗ:

Bk Fp t T · = , мес. де

Bk- число виконавців;

Fp – місячний фонд робочого часу (при 40 годинному робочому тижні Fp=176 годин).

T = 1697,44 1 · 176 = 9,6 міс.

Висновки: На розробку даного програмного забезпечення піде 1697,44 людино-годин. Тобто, ймовірна очікувана тривалість розробки складатиме 9,6 місяців при стандартному 40-годинному робочому тижні і 176-годинному 70 робочому місяці. Очікувані витрати на створення програмного забезпечення складатимуть 419148,85 грн.

ВИСНОВКИ

У даній кваліфікаційній роботі було створено нейронну мережу, яка вміє оброблювати вхідні дані, вчитися на них, робити прогнозування майбутніх даних. Результат роботи можна візуалізувати, щоб побачити прогнози на графіку.

Функціонал розроблено у вигляді декількох найбільш розповсюджених моделях нейронних мереж. Кожна з них обчислює і прогнозує дані за допомогою Tensorflow, Keras API, бібліотек мови програмування Python.

Для побудови різних архітектур нейронних мереж було переглянуто інформацію та проаналізовано різні види нейронних мереж, особливості побудови, переваги, цільові області використання, можливості кожного типу мережі для певної задачі.

Під час роботи було створено по екземпляру кожної нейронної мережі, проаналізовано результати кожної з використаних архітектур нейронних мереж та виявлено найбільш влучні та вдалі саме для завдання цієї кваліфікаційної роботи.

За допомогою отриманих у результаті даних буде побудовано інтерфейс для веб-застосунку для візуалізації результату для кінцевих користувачів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Коротенко Л. М., Шевцова О. С. Методичні рекомендації до виконання кваліфікаційних робіт бакалаврів напряму підготовки 122 "Комп'ютерні науки" галузі знань 12 Інформаційні технології. Нац. тех. ун-т. - Д : ДВНЗ НТУДП, 2021. - 65 с.
2. Вагонова О. Г., Волотковська Ю. А., Романюк Н.Н. Методичні вказівки з виконання економічного розділу в дипломних проектах студентів спеціальності “Програмне забезпечення”. Дніпропетровськ: Національний гірничий університет, 2013. – 11 с.
3. Хайкін С. «Нейронні Мережі» 1104 с.
4. Єлбон Кріс «Машинне навчання з використанням Python». Збірник. (БХВ-Петербург)
5. Ian Goodfellow, Yoshua Bengio, Aaron Courville «Deep Learning»
6. Christopher Bishop «Pattern Recognition and Machine Learning»
7. Луїс Педро Коельо, Віллі Річард «Побудова систем машинного навчання мовою Python», 2014.
8. arXiv.org, URL: https://arxiv.org/
9. Pedro Domingos ‘The Master Algorithm: How the Quest for the Ultimate Learning Machine Will Remake Our World’
10. Richard Sutton ‘Reinforcement Learning, second edition: An Introduction’
11. EarthByte datasets, URL: https://www.earthbyte.org/category/resources/data-models/global-regional-plate-motion-models/
12. EarthByte Mid-ocean Ridge dataset and schemes, URL: https://www.earthbyte.org/webdav/ftp/Data_Collections/Merdith_etal_2021_ESR/
13. Tensorflow documentation, datasets, URL: https://www.tensorflow.org/datasets

14. Tensorflow Keras API module and documentation, URL: https://www.tensorflow.org/api_docs/python/tf/keras
15. Ancient Earth visualization model, URL: https://dinosaurpictures.org/ancient-earth#750
16. Neural Network explanation and history, URL: https://www.sciencedirect.com/topics/social-sciences/neural-network
17. The machine learning articles, URL: https://www.sciencedirect.com/search?qs=machine%20learning
18. Python dictionaries tutorials, URL: https://www.w3schools.com/python/python_dictionaries.asp
19. Python Matplotlib, URL: https://matplotlib.org/
20. Matplotlib tutorials, URL: https://matplotlib.org/stable/tutorials/introductory/pyplot.html
URL: https://www.w3schools.com/python/matplotlib_pyplot.asp
21. Python Numpy, URL: https://www.w3schools.com/python/numpy/numpy_intro.asp

ДОДАТОК А

КОД ПРОГРАМИ

У додатку наведено код програми нейронної мережі, включаючи побудовані класи, екземпляри різних типів мереж, загальні функції.

```
import os
import datetime
import csv
import IPython
import IPython.display
import matplotlib as mpl
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import seaborn as sns
import tensorflow as tf

mpl.rcParams['figure.figsize'] = (8, 6)
mpl.rcParams['axes.grid'] = False

zip_path = tf.keras.utils.get_file(
    origin='https://www.earthbyte.org/webdav/ftp/Data_Collections/Merdith_
etal_2021_ESR/SM3b_MOR_stats.csv.zip',
    fname='SM3b_MOR_stats.csv.zip',
    extract=True)

csv_path, _ = os.path.splitext(zip_path)



df = pd.read_csv(csv_path)
# Slice [start:stop:step], starting from index 5 take every 6th record.
df = df[8::1]
#date_time = pd.to_datetime(df.pop('time (Ma)'), format='%Y')
df.head()
df.describe().transpose()
#timestamp_s = date_time.map(pd.Timestamp.timestamp)


column_indices = {name: i for i, name in enumerate(df.columns)}
```

```
n = len(df)
train_df = df[0:int(n*0.7)]
val_df = df[int(n*0.7):int(n*0.9)]
test_df = df[int(n*0.9):]


num_features = df.shape[1]


train_mean = train_df.mean()
train_std = train_df.std()


train_df = (train_df - train_mean) / train_std
val_df = (val_df - train_mean) / train_std
test_df = (test_df - train_mean) / train_std


df_std = (df - train_mean) / train_std
df_std = df_std.melt(var_name='Column', value_name='Normalized')
#plt.figure(figsize=(12, 6))
ax = sns.violinplot(x='Column', y='Normalized', data=df_std)
_ = ax.set_xticklabels(df.keys(), rotation=90)


class WindowGenerator():
  def __init__(self, input_width, label_width, shift,
               train_df=train_df, val_df=val_df, test_df=test_df,
               label_columns=None):
    # Store the raw data.
    self.train_df = train_df
    self.val_df = val_df
    self.test_df = test_df

    # Work out the label column indices.
    self.label_columns = label_columns
    if label_columns is not None:
      self.label_columns_indices = {name: i for i, name in
                                    enumerate(label_columns)}
    self.column_indices = {name: i for i, name in
                           enumerate(train_df.columns)}

    # Work out the window parameters.
    self.input_width = input_width
    self.label_width = label_width
    self.shift = shift

    self.total_window_size = input_width + shift

    self.input_slice = slice(0, input_width)
```

```
    self.input_indices = np.arange(self.total_window_size)[self.input_slic
e]

    self.label_start = self.total_window_size - self.label_width
    self.labels_slice = slice(self.label_start, None)
    self.label_indices = np.arange(self.total_window_size)[self.labels_sli
ce]

  def __repr__(self):
    return '\n'.join([
        f'Total window size: {self.total_window_size}',
        f'Input indices: {self.input_indices}',
        f'Label indices: {self.label_indices}',
        f'Label column name(s): {self.label_columns}'])

w1 = WindowGenerator(input_width=24, label_width=1, shift=24,
                     label_columns=['MOR_length (km)'])
w1

w2 = WindowGenerator(input_width=7, label_width=1, shift=1,
                     label_columns=['MOR_length (km)'])
w2
def split_window(self, features):
  inputs = features[:, self.input_slice, :]
  labels = features[:, self.labels_slice, :]
  if self.label_columns is not None:
    labels = tf.stack(
        [labels[:, :, self.column_indices[name]] for name in self.label_co
lumns],
        axis=-1)

  # Slicing doesn't preserve static shape information, so set the shapes
  # manually. This way the `tf.data.Datasets` are easier to inspect.
  inputs.set_shape([None, self.input_width, None])
  labels.set_shape([None, self.label_width, None])

  return inputs, labels


WindowGenerator.split_window = split_window

example_window = tf.stack([np.array(train_df[:w2.total_window_size]),
                           np.array(train_df[100:100+w2.total_window_size]
),
                           np.array(train_df[200:200+w2.total_window_size]
)])

example_inputs, example_labels = w2.split_window(example_window)
w2.example = example_inputs, example_labels
def make_dataset(self, data):
```

```
  data = np.array(data, dtype=np.float32)
  ds = tf.keras.utils.timeseries_dataset_from_array(
      data=data,
      targets=None,
      sequence_length=self.total_window_size,
      sequence_stride=1,
      shuffle=True,
      batch_size=32,)

  ds = ds.map(self.split_window)

  return ds

WindowGenerator.make_dataset = make_dataset
@property
def train(self):
  return self.make_dataset(self.train_df)

@property
def val(self):
  return self.make_dataset(self.val_df)

@property
def test(self):
  return self.make_dataset(self.test_df)

@property
def example(self):
  """Get and cache an example batch of `inputs, labels` for plotting."""
  result = getattr(self, '_example', None)
  if result is None:
    # No example batch was found, so get one from the `.train` dataset
    result = next(iter(self.train))
    # And cache it for next time
    self._example = result
  return result

WindowGenerator.train = train
WindowGenerator.val = val
WindowGenerator.test = test
WindowGenerator.example = example


def plot(self, model=None, plot_col='MOR_length (km)', max_subplots=3):
  inputs, labels = self.example
  plt.figure(figsize=(12, 8))
  plot_col_index = self.column_indices[plot_col]
  max_n = min(max_subplots, len(inputs))
  for n in range(max_n):
    plt.subplot(max_n, 1, n+1)
```

```
    plt.ylabel(f'{plot_col} [normed]')
    plt.plot(self.input_indices, inputs[n, :, plot_col_index],
             label='Inputs', marker='.', zorder=-10)

    if self.label_columns:
      label_col_index = self.label_columns_indices.get(plot_col, None)
    else:
      label_col_index = plot_col_index

    if label_col_index is None:
      continue

    plt.scatter(self.label_indices, labels[n, :, label_col_index],
                edgecolors='k', label='Labels', c='#2ca02c', s=64)
    if model is not None:
      predictions = model(inputs)
      plt.scatter(self.label_indices, predictions[n, :, label_col_index],
                  marker='X', edgecolors='k', label='Predictions',
                  c='#ff7f0e', s=64)
    if n == 0:
      plt.legend()

  plt.xlabel('time (Ma)')
WindowGenerator.plot = plot


single_step_window = WindowGenerator(
    input_width=1, label_width=1, shift=1,
    label_columns=['MOR_length (km)'])
single_step_window


class Baseline(tf.keras.Model):
  def __init__(self, label_index=None):
    super().__init__()
    self.label_index = label_index

  def call(self, inputs):
    if self.label_index is None:
      return inputs
    result = inputs[:, :, self.label_index]
    return result[:, :, tf.newaxis]



val_performance = {}
performance = {}


wide_window = WindowGenerator(
    input_width=24, label_width=24, shift=1,
```

```
    label_columns=['MOR_length (km)'])

wide_window



MAX_EPOCHS = 20

def compile_and_fit(model, window, patience=2):
  early_stopping = tf.keras.callbacks.EarlyStopping(monitor='val_loss',
                                                    patience=patience,
                                                    mode='min')

  model.compile(loss=tf.losses.MeanSquaredError(),
                optimizer=tf.optimizers.Adam(),
                metrics=[tf.metrics.MeanAbsoluteError()])

  history = model.fit(window.train, epochs=MAX_EPOCHS,
                      validation_data=window.val,
                      callbacks=[early_stopping])
  return history

#creation of Basic exemplar
baseline = Baseline(label_index = column_indices['MOR_length (km)'])

baseline.compile(loss=tf.losses.MeanSquaredError(),
                 metrics=[tf.metrics.MeanAbsoluteError()])
val_performance['Baseline'] = baseline.evaluate(single_step_window.val)
performance['Baseline'] = baseline.evaluate(single_step_window.test, verbo
se=0)
#wide_window.plot(baseline)

#creation of Linear exemplar
linear = tf.keras.Sequential([
    tf.keras.layers.Dense(units=1)
])

history = compile_and_fit(linear, single_step_window)

val_performance['Linear'] = linear.evaluate(single_step_window.val)
performance['Linear'] = linear.evaluate(single_step_window.test, verbose=0
)

wide_window.plot(linear)


#creation of Dense exemplar
dense = tf.keras.Sequential([
    tf.keras.layers.Dense(units=64, activation='relu'),
    tf.keras.layers.Dense(units=64, activation='relu'),
    tf.keras.layers.Dense(units=1)
```

```
])

history = compile_and_fit(dense, single_step_window)

val_performance['Dense'] = dense.evaluate(single_step_window.val)
performance['Dense'] = dense.evaluate(single_step_window.test, verbose=0)
wide_window.plot(dense)

CONV_WIDTH = 3
conv_window = WindowGenerator(
    input_width=CONV_WIDTH,
    label_width=1,
    shift=1,
    label_columns=['MOR_length (km)'])

conv_window


#creation of MULTI_STEP_DENSE exemplar
multi_step_dense = tf.keras.Sequential([
    # Shape: (time, features) => (time*features)
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(units=32, activation='relu'),
    tf.keras.layers.Dense(units=32, activation='relu'),
    tf.keras.layers.Dense(units=1),
    # Add back the time dimension.
    # Shape: (outputs) => (1, outputs)
    tf.keras.layers.Reshape([1, -1]),
])
history = compile_and_fit(multi_step_dense, conv_window)

IPython.display.clear_output()
val_performance['Multi step dense'] = multi_step_dense.evaluate(conv_windo
w.val)
performance['Multi step dense'] = multi_step_dense.evaluate(conv_window.te
st, verbose=0)


#creation of Conv exemplar
conv_model = tf.keras.Sequential([
    tf.keras.layers.Conv1D(filters=32,
                           kernel_size=(CONV_WIDTH,),
                           activation='relu'),
    tf.keras.layers.Dense(units=32, activation='relu'),
    tf.keras.layers.Dense(units=1),
])

history = compile_and_fit(conv_model, conv_window)

IPython.display.clear_output()
val_performance['Conv'] = conv_model.evaluate(conv_window.val)
```

```
performance['Conv'] = conv_model.evaluate(conv_window.test, verbose=0)



LABEL_WIDTH = 24
INPUT_WIDTH = LABEL_WIDTH + (CONV_WIDTH - 1)
wide_conv_window = WindowGenerator(
    input_width=INPUT_WIDTH,
    label_width=LABEL_WIDTH,
    shift=1,
    label_columns=['MOR_length (km)'])

wide_conv_window
wide_conv_window.plot(conv_model)


#creation of RNN LSTM exemplar
lstm_model = tf.keras.models.Sequential([
    # Shape [batch, time, features] => [batch, time, lstm_units]
    tf.keras.layers.LSTM(32, return_sequences=True),
    # Shape => [batch, time, features]
    tf.keras.layers.Dense(units=1)
])

history = compile_and_fit(lstm_model, wide_window)

IPython.display.clear_output()
val_performance['LSTM'] = lstm_model.evaluate(wide_window.val)
performance['LSTM'] = lstm_model.evaluate(wide_window.test, verbose=0)
wide_window.plot(lstm_model)



single_step_window = WindowGenerator(
    # `WindowGenerator` returns all features as labels if you
    # don't set the `label_columns` argument.
    input_width=1, label_width=1, shift=1)

wide_window = WindowGenerator(
    input_width=24, label_width=24, shift=1)

for example_inputs, example_labels in wide_window.train.take(1):
  print(f'Inputs shape (batch, time, features): {example_inputs.shape}')
  print(f'Labels shape (batch, time, features): {example_labels.shape}')



OUT_STEPS = 24
multi_window = WindowGenerator(input_width=24,
                               label_width=OUT_STEPS,
                               shift=OUT_STEPS)
```

```
multi_window.plot()
multi_window


class MultiStepLastBaseline(tf.keras.Model):
  def call(self, inputs):
    return tf.tile(inputs[:, -1:, :], [1, OUT_STEPS, 1])



#BASELINE MODEL
last_baseline = MultiStepLastBaseline()
last_baseline.compile(loss=tf.losses.MeanSquaredError(),
                      metrics=[tf.metrics.MeanAbsoluteError()])

multi_val_performance = {}
multi_performance = {}

multi_val_performance['Last'] = last_baseline.evaluate(multi_window.val)
multi_performance['Last'] = last_baseline.evaluate(multi_window.test, verb
ose=0)
multi_window.plot(last_baseline)

class RepeatBaseline(tf.keras.Model):
  def call(self, inputs):
    return inputs

repeat_baseline = RepeatBaseline()
repeat_baseline.compile(loss=tf.losses.MeanSquaredError(),
                        metrics=[tf.metrics.MeanAbsoluteError()])

multi_val_performance['Repeat'] = repeat_baseline.evaluate(multi_window.va
l)
multi_performance['Repeat'] = repeat_baseline.evaluate(multi_window.test,
verbose=0)
myData = [multi_window.test]
multi_window.plot(repeat_baseline)

multi_linear_model = tf.keras.Sequential([
    # Take the last time-step.
    # Shape [batch, time, features] => [batch, 1, features]
    tf.keras.layers.Lambda(lambda x: x[:, -1:, :]),
    # Shape => [batch, 1, out_steps*features]
    tf.keras.layers.Dense(OUT_STEPS*num_features,
                          kernel_initializer=tf.initializers.zeros()),
    # Shape => [batch, out_steps, features]
    tf.keras.layers.Reshape([OUT_STEPS, num_features])
])

history = compile_and_fit(multi_linear_model, multi_window)
```

```
IPython.display.clear_output()
multi_val_performance['Linear'] = multi_linear_model.evaluate(multi_window
.val)
multi_performance['Linear'] = multi_linear_model.evaluate(multi_window.tes
t, verbose=0)
multi_window.plot(multi_linear_model)



multi_dense_model = tf.keras.Sequential([
    # Take the last time step.
    # Shape [batch, time, features] => [batch, 1, features]
    tf.keras.layers.Lambda(lambda x: x[:, -1:, :]),
    # Shape => [batch, 1, dense_units]
    tf.keras.layers.Dense(512, activation='relu'),
    # Shape => [batch, out_steps*features]
    tf.keras.layers.Dense(OUT_STEPS*num_features,
                          kernel_initializer=tf.initializers.zeros()),
    # Shape => [batch, out_steps, features]
    tf.keras.layers.Reshape([OUT_STEPS, num_features])
])

history = compile_and_fit(multi_dense_model, multi_window)

IPython.display.clear_output()
multi_val_performance['Dense'] = multi_dense_model.evaluate(multi_window.v
al)
multi_performance['Dense'] = multi_dense_model.evaluate(multi_window.test,
 verbose=0)
myData_dense = [multi_window.test]
myFile_dense = open('predictions_dense.csv', 'w')
with myFile_dense:
      writer = csv.writer(myFile_dense)
      writer.writerows(myData_dense)
      print("Writing complete")
multi_window.plot(multi_dense_model)


#MULTI_CONVOLUTIONAL MODEL
CONV_WIDTH = 3
multi_conv_model = tf.keras.Sequential([
    # Shape [batch, time, features] => [batch, CONV_WIDTH, features]
    tf.keras.layers.Lambda(lambda x: x[:, -CONV_WIDTH:, :]),
    # Shape => [batch, 1, conv_units]
    tf.keras.layers.Conv1D(256, activation='relu', kernel_size=(CONV_WIDTH
)),
    # Shape => [batch, 1,  out_steps*features]
    tf.keras.layers.Dense(OUT_STEPS*num_features,
                          kernel_initializer=tf.initializers.zeros()),
    # Shape => [batch, out_steps, features]
```

```
    tf.keras.layers.Reshape([OUT_STEPS, num_features])
])

history = compile_and_fit(multi_conv_model, multi_window)

IPython.display.clear_output()

multi_val_performance['Conv'] = multi_conv_model.evaluate(multi_window.val
)
multi_performance['Conv'] = multi_conv_model.evaluate(multi_window.test, v
erbose=0)
myData_conv = [multi_window.test]
myFile_conv = open('predictions_conv.csv', 'w')
with myFile_conv:
      writer = csv.writer(myFile_conv)
      writer.writerows(myData_conv)
      print("Writing complete")
multi_window.plot(multi_conv_model)



multi_lstm_model = tf.keras.Sequential([
    # Shape [batch, time, features] => [batch, lstm_units].
    # Adding more `lstm_units` just overfits more quickly.
    tf.keras.layers.LSTM(32, return_sequences=False),
    # Shape => [batch, out_steps*features].
    tf.keras.layers.Dense(OUT_STEPS*num_features,
                          kernel_initializer=tf.initializers.zeros()),
    # Shape => [batch, out_steps, features].
    tf.keras.layers.Reshape([OUT_STEPS, num_features])
])

history = compile_and_fit(multi_lstm_model, multi_window)

IPython.display.clear_output()

multi_val_performance['LSTM'] = multi_lstm_model.evaluate(multi_window.val
)
multi_performance['LSTM'] = multi_lstm_model.evaluate(multi_window.test, v
erbose=0)
myData_lstm = [multi_window.test]
myFile_lstm = open('predictions_lstm.csv', 'w')
with myFile_lstm:
      writer = csv.writer(myFile_lstm)
      writer.writerows(myData_lstm)
      print("Writing complete")
multi_window.plot(multi_lstm_model)



class FeedBack(tf.keras.Model):
```

```
  def __init__(self, units, out_steps):
    super().__init__()
    self.out_steps = out_steps
    self.units = units
    self.lstm_cell = tf.keras.layers.LSTMCell(units)
    # Also wrap the LSTMCell in an RNN to simplify the `warmup` method.
    self.lstm_rnn = tf.keras.layers.RNN(self.lstm_cell, return_state=True)
    self.dense = tf.keras.layers.Dense(num_features)

feedback_model = FeedBack(units=32, out_steps=OUT_STEPS)

def warmup(self, inputs):
  # inputs.shape => (batch, time, features)
  # x.shape => (batch, lstm_units)
  x, *state = self.lstm_rnn(inputs)

  # predictions.shape => (batch, features)
  prediction = self.dense(x)
  return prediction, state

FeedBack.warmup = warmup
prediction, state = feedback_model.warmup(multi_window.example[0])
prediction.shape

def call(self, inputs, training=None):
  # Use a TensorArray to capture dynamically unrolled outputs.
  predictions = []
  # Initialize the LSTM state.
  prediction, state = self.warmup(inputs)

  # Insert the first prediction.
  predictions.append(prediction)

  # Run the rest of the prediction steps.
  for n in range(1, self.out_steps):
    # Use the last prediction as input.
    x = prediction
    # Execute one lstm step.
    x, state = self.lstm_cell(x, states=state,
                              training=training)
    # Convert the lstm output to a prediction.
    prediction = self.dense(x)
    # Add the prediction to the output.
    predictions.append(prediction)

  # predictions.shape => (time, batch, features)
  predictions = tf.stack(predictions)
  # predictions.shape => (batch, time, features)
  predictions = tf.transpose(predictions, [1, 0, 2])
  return predictions
```

```
FeedBack.call = call

history = compile_and_fit(feedback_model, multi_window)

IPython.display.clear_output()

multi_val_performance['AR LSTM'] = feedback_model.evaluate(multi_window.va
l)
multi_performance['AR LSTM'] = feedback_model.evaluate(multi_window.test,
verbose=0)
multi_window.plot(feedback_model)
```

ДОДАТОК Б

ВІДГУК

керівника економічного розділу на кваліфікаційну роботу бакалавра

на тему:

«Розробка нейронної мережі для візуалізації поверхні земної кулі у заданий час на основі Python/Tensorflow»

студенки групи 122-18-1

Гуліної Ольги Олександрівни

Керівник економічного розділу **Л. В. Касьяненко**

доцент каф. ПЕП та ПУ, к.е.н

ДОДАТОК В

ПЕРЕЛІК ДОКУМЕНТІВ НА ОПТИЧНОМУ НОСІЇ

Ім’я файлу	Опис
Пояснювальні документи	
Кваліфікаційна_робота_Гуліна.docx	Пояснювальна записка до кваліфікаційної роботи. Документ Word
Кваліфікаційна_робота_Гуліна.pdf	Пояснювальна записка до кваліфікаційної роботи в форматі PDF
Програма	
Гуліна.zip	Архів. Містить коди програми і скомпільовану програму
Презентація	
Гуліна.pptx	Презентація кваліфікаційної роботи