НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Факультет прикладної математики

Кафедра системного програмування

і спеціалізованих комп'ютерних систем

«На правах рукопису»
УДК ____004.75_____

До захисту допущено:
Завідувач кафедри
_____ Віталій РОМАНКЕВИЧ
«___»_____2023 р.

Магістерська дисертація

на здобуття ступеня магістра

за освітньо-науковою програмою «Системне програмування та спеціалізовані комп'ютерні системи»
зі спеціальності 123 «Комп'ютерна інженерія»

на тему: «Способи і засоби розподіленої обробки часових рядів в кластерних системах»

Виконав:
студент ІІ курсу, групи КВ-11мн
Туркін Михайло Павлович _____
Науковий керівник:
професор, д.т.н., проф.
Романкевич Віталій Олексійович
Консультант:
старший викладач
Дробязко Ірина Павлівна _____
Консультант з нормоконтролю:
доцент, к.т.н., с.н.с.
Боярінова Юлія Євгенівна _____
Рецензент:
_____
_____ _____

Засвідчую, що у цій магістерській дисертації немає запозичень з праць інших авторів без відповідних посилань.
Студент _____

Київ – 2023 року

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ

імені ІГОРЯ СІКОРСЬКОГО»

Факультет прикладної математики

Кафедра системного програмування і спеціалізованих комп‘ютерних систем

Рівень вищої освіти – другий (магістерський)

Спеціальність - 123 «Комп’ютерна інженерія»

Освітньо-наукова програма - **«Системне програмування та спеціалізовані комп’ютерні системи»**

ЗАТВЕРДЖУЮ

Завідувач кафедри

_____ Віталій РОМАНКЕВИЧ

(підпис) (ініціали, прізвище)

«___» _____2023 р.

ЗАВДАННЯ
на магістерську дисертацію студента
Туркіна Михайла Павловича

1. Тема дисертації Способи і засоби розподіленої обробки часових рядів в кластерних системах, керівник дисертації професор, д.т.н., проф. Романкевич Віталій Олексійович, затверджені наказом по університету «30» березня 2023 р. № 1359-с
2. Термін подання студентом дисертації: «16» травня 2023 р.
3. Об’єкт дослідження: обробка великих обсягів даних з використанням кластерних систем.
4. Предмет дослідження: способи та алгоритми розподіленої обробки часових рядів в кластерних системах.

5. Перелік завдань, які потрібно розробити:
   - провести дослідження існуючих способів і засобів розподіленої обробки часових рядів в кластерних системах;
   - запропонувати та обгрунтувати вибір методів та підходів для реалізації обробки часових рядів;
   - протестувати запропоновані методи та оцінити результати.
6. Орієнтований перелік графічного (ілюстрованого) матеріалу: презентація.
7. Дата видачі завдання 20.10.2021р.

Календарний план

№ з/п	Назва етапів виконання магістерської дисертації	Термін виконання етапів	Примітка
1.	Вивчення літератури за тематикою дисертації	15.12.2021	
2.	Визначення структури магістерської дисертації	01.02.2022	
3.	Робота над першим розділом магістерської дисертації; проведення наукового дослідження	21.07.2022	
4.	Підготовка матеріалів першого розділу дипломного проекту	28.08.2022	
5.	Робота над другим розділом магістерської дисертації; підготовка матеріалів доповіді на конференції ПМК-2022	08.10.2022	
6.	Проведення наукового дослідження; розроблення програмного забезпечення	11.12.2022	
7.	Проведення наукового дослідження; робота над третім розділом магістерської дисертації	17.01.2023	
8.	Підготовка матеріалів четвертого розділу магістерської дисертації	01.03.2023	
9.	Завершення роботи над основною частиною магістерської дисертації	23.04.2023	
10.	Оформлення матеріалів магістерської дисертації	30.04.2023	

Студент _____ Михайло ТУРКІН
(підпис) (ініціали, прізвище)

Науковий керівник дисертації _____ Віталій РОМАНКЕВИЧ
(підпис) (ініціали, прізвище)

Консультант _____ Ірина ДРОБЯЗКО
(підпис) (ініціали, прізвище)

РЕФЕРАТ

Актуальність теми. У сучасну цифрову епоху кількість даних, що генеруються, зростає в геометричній прогресії. Їх аналіз є дуже важливим, оскільки допомагає приймати обґрунтовані рішення, виявляти тенденції та закономірності, а також для виявлення нових бізнес-можливостей, покращення обслуговування клієнтів та розробки нових продуктів чи послуг. Це створює нагальну потребу в ефективних і масштабованих методах обробки та аналізу великих обсягів даних. Часові ряди є важливим типом даних для бізнесу, оскільки вони дають уявлення про тенденції та закономірності, які можуть бути використані для прогнозування та прийняття рішень. У роботі проводиться аналіз способів і засобів розподіленої обробки та прогнозування часових рядів на кластері. Підвищення їх ефективності є актуальним в галузі роздрібнаої торгівлі. Підприємства роздрібної торгівлі генерують величезні обсяги даних часових рядів з різних джерел, включаючи операції продажу, рівень запасів та інформацію про поведінку клієнтів. Швидка та ефективна обробка історичних даних надасть змогу робити точні прогнози продажів і попиту, що в свою чергу дозволить підприємствам оптимізувати рівень запасів і підвищити дохід.

Найпопулярнішим на сьогодні рішенням є використання інструментів та підходів, що надають хмарні сервіси. Проте, таке рішення недостатньо гнучке і потребує складної інтеграції даних. Тому існує потреба в легкомасштабованих та гнучких засобах для розподіленої обробки часових рядів.

Метою є підвищення ефективності алгоритмів обробки часових рядів шляхом використання розподілених обчислень на кластері.

Об’єктом дослідження є обробка великих обсягів даних з використанням кластерних систем.

Предметом дослідження є способи та алгоритми розподіленої обробки часових рядів в кластерних системах.

Наукова новизна полягає в наступному:

1. Запропоновано новий алгоритм для обробки часових рядів з використанням проекцій багатовимірних масивів, що дозволить суттєво зменшити час їх обробки та об’єми використовуваної для обчислень пам’яті.
2. Запропоновано модифікацію існуючих алгоритмів прогнозування часових рядів для розподіленого виконання на кластері шляхом використання синхронного навчання моделей.
3. Виконано програмну реалізацію запропонованих алгоритмів обробки та прогнозування, яка доводить їх ефективність.

Практична цінність: запропоновані в дисертації алгоритми та засоби розподіленої обробки дозволять суттєво збільшити швидкість обробки великої кількості часових рядів. Це сприятиме підвищенню ефективності прогнозування у багатьох галузях: фінансовому прогнозуванні, прогнозуванні попиту на енергію, прогнозуванні продажів в сфері роздрібної торгівлі тощо. Розроблені алгоритми можуть бути легко інтегровані в програмні засоби для розподіленої обробки і прогнозування часових рядів, надаючи користувачам зручні і масштабовані інструменти.

Апробація роботи. Основні положення і результати роботи представлено та обговорювалось на XV науковій конференції магістрантів

та аспірантів «Прикладна математика та комп'ютинг» ПМК-2022 (Київ, КПІ, ФПМ, 16-18 листопада 2022 р.) та на 89 міжнародній науковій конференції молодих учених, аспірантів і студентів «Наукові здобутки молоді – вирішенню проблем харчування людства у XXI ст.» (Київ, НУХТ, 3-7 квітня 2023 р.).

Структура та обсяг роботи. Магістерська дисертація складається з вступу, чотирьох розділів та висновків.

У вступі надано оцінку сучасного стану проблеми, наведено обґрунтування актуальності напряму дослідження, а також формулювання мети та завдань дослідження.

У першому розділі досліджено існуючі способи і засоби для розподіленої обробки даних часових рядів, виявлено основні переваги та недоліки, визначено шляхи їх вдосконалення.

У другому розділі розглянуто основні технології, способи та засоби для роботи з великими обсягами даних. Представлено сучасні підходи для розподіленого глибинного навчання.

У третьому розділі представлено опис запропонованого алгоритму для підготовки даних та модифікованих алгоритмів прогнозування часових рядів. Надано загальний підхід до програмної реалізації алгоритмів.

У четвертому розділі досліджено ефективність запропонованих алгоритмів обробки та прогнозування часових рядів на кластері.

У висновках представлені результати проведеної роботи.

Робота представлена на 90 аркушах, містить посилання на список використаних літературних джерел.

Ключові слова: великі дані, кластер, часові ряди, розподілена обробка, прогнозування.

ABSTRACT

Actuality of theme. In today's digital age, the amount of data generated is growing exponentially. Analyzing it is very important as it helps to make informed decisions, identify trends and patterns, as well as to identify new business opportunities, improve customer service, and develop new products or services. This creates an urgent need for efficient and scalable methods of processing and analyzing large amounts of data. Time series are an important type of data for business because they provide insights into trends and patterns that can be used for forecasting and decision-making. This paper analyzes ways and means of distributed processing and forecasting of time series on a cluster. Increasing their efficiency is relevant in the retail industry. Retailers generate huge amounts of time series data from various sources, including sales transactions, inventory levels, and information about customer behavior. Fast and efficient processing of historical data will enable accurate sales and demand forecasts, which in turn will allow businesses to optimize inventory levels and increase revenue.

The most popular solution today is to use tools and approaches provided by cloud services. However, this solution is not flexible enough and requires complex data integration. Therefore, there is a need for easily scalable and flexible tools for distributed time series processing.

The goal of the work is to improve the efficiency of time series processing algorithms by using distributed computing on a cluster.

The object of research is the processing of large amounts of data using cluster systems

The subject of the study are methods and algorithms for distributed time series processing in cluster systems.

The scientific novelty:

1. An effective implementation of the algorithm for processing time series is proposed.
2. A modification for distributed execution of existing time series prediction algorithms on a cluster is implemented.

Practical value of the results obtained in this work is that the modified algorithms presented in this thesis significantly increase the processing speed of a large number of time series. This will improve the efficiency of forecasting in many industries, such as financial forecasting, energy demand forecasting, and retail sales forecasting. The developed algorithms can be easily integrated into various applications, including web and classic applications, providing users with an easily accessible and scalable tool for distributed time series processing and forecasting.

Approbation of work. The main provisions and results of the work were presented and discussed at the XV scientific conference of undergraduates and graduate students "Applied Mathematics and Computing" PMC-2022 (Kyiv, November 16-18, 2022) and at the 89th International Scientific Conference of Young Scientists, Postgraduates and Students "Scientific Achievements of Youth" (Kyiv, April 3-7, 2023).

Structure and scope of work. The master's thesis consists of an introduction, four chapters and conclusions.

The *introduction* provides a general description of the work, assesses the current state of the problem, justifies the relevance of the research area, and formulates the purpose and objectives of the study.

The *first section* examines the existing methods and tools for distributed processing of time series data and identifies the main advantages and disadvantages of existing tools.

The *second section* discusses the main technologies and tools for working with large amounts of data. Approaches for distributed deep learning are described.

The *third section* describes the operation of modified algorithms for data preparation and time series forecasting. Algorithms for time series processing and distributed training of classical and deep learning models are described.

The *fourth section* investigates the efficiency and performance of the implemented algorithms.

The *conclusion* presents the results of the work.

The paper is presented on 90 pages and contains references to the list of used literature sources.

Keywords: big data, cluster, time series, distributed processing, forecasting.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ, УМОВНИХ ПОЗНАЧЕНЬ, ТЕРМІНІВ

ARIMA - Autoregressive Integrated Moving Average (авторегресійне інтегроване ковзне середнє)

CNN - Convolutional Neural Network (згорткова нейронна мережа)

DDP - Distributed Data Parallel (розподілений паралельний доступ до даних)

GPU - Graphics Processing Unit (графічний процесор)

GRU - Gated recurrent units (вентильні рекурентні вузли)

LSTM - Long Short-Term Memory (мережі довгої короткочасної пам'яті)

RNN - Recurrent Neural Network (рекурентна нейронна мережа)

SaaS - Software as a Service (програмне забезпечення як послуга)

ВСТУП

В останні роки експоненціальне зростання обсягів даних та підвищення складності процедур обробки значно збільшують попит на обчислювальні потужності. Однією зі сфер, де цей попит особливо високий, є аналіз і прогнозування часових рядів. Компанії з різних галузей, таких як роздрібна торгівля, фінанси та енергетика та ін., потребують точного прогнозування часових рядів для прийняття обґрунтованих рішень та оптимізації дій.

Однак традиційні алгоритмічні підходи до побудови системи прогнозування стикаються з проблемами, пов’язаними з використанням ресурсів однієї обчислювальної машини. В такому разі масштабування системи має обмежений характер, оскільки залежить від ресурсів однієї машини: розміру оперативної пам'яті, потужності центрального процесора або ємності сховища. Крім того, існують фізичні обмеження на можливість збільшення цих ресурсів, і, як наслідок, на зростання продуктивності за рахунок вертикального масштабування.

Для вирішення цієї проблеми дедалі більшої популярності набувають розподілені обчислення, а хмарна інфраструктура пропонує горизонтальну масштабованість. Незважаючи на це, розробка розподілених алгоритмів і додатків залишається складним завданням, оскільки вимагає управління паралельними алгоритмами, вирішення інженерних проблем комунікації між машинами та затримок передачі даних, а також забезпечення відмовостійкості.

У цьому контексті, розробка ефективних і масштабованих методів розподіленої обробки та прогнозування даних часових рядів на кластері має велике значення. Наприклад, галузь роздрібної торгівлі генерує величезні обсяги даних часових рядів, зокрема транзакції продажів, дані щодо поведінки клієнтів та рівня запасів продукції. Точне прогнозування цих даних може

дозволити ритейлерам оптимізувати рівень запасів, зменшити витрати на продукцію та збільшити прибуток.

Тому дослідження, представлені в дисертації, зосереджені на розробці ефективних алгоритмів для розподіленої обробки та прогнозування часових рядів даних на кластері. Потрібно враховувати та вирішувати проблеми, що виникають при розподілених обчисленнях, а також забезпечувати точні прогнози з високою ефективністю та масштабованістю. Таке дослідження матиме практичну цінність, зокрема в галузі роздрібної торгівлі, де є потреба в ефективних і масштабованих методах розподіленої обробки та прогнозування часових рядів для оптимізації торгових операцій та збільшення прибутків.

1. АНАЛІЗ ПРОБЛЕМИ ОБРОБКИ ЧАСОВИХ РЯДІВ ТА ФОРМУЛЮВАННЯ МЕТИ ДИСЕРТАЦІЇ

1.1. Загальний опис проблеми

Прогнозування є критично важливим аспектом сучасного бізнесу, зокрема торгівлі, а точне прогнозування користувацького попиту необхідне для підтримки оптимального рівня запасів і уникнення випадків нестачі або надлишку запасів продукції.

Прогнозування часових рядів (англ. time series forecasting) — це застосування моделі для передбачування майбутніх значень на основі значень попередньо спостережених.

Наразі традиційні методи прогнозування є недостатніми для обробки великих і складних наборів даних, що генеруються. Розподілена обробка даних стала потужною технологією, яка дозволяє обробляти і аналізувати великі масиви даних, розбиваючи їх на менші, більш керовані частини, які можна обробляти паралельно на декількох машинах. Цей підхід має низку переваг, зокрема зменшення часу обробки, покращення масштабованості та ефективне використання обчислювальних ресурсів.

У контексті прогнозування часових рядів розподілена обробка даних може бути особливо корисною через велику кількість накопичених за певний період часу послідовностей даних, які необхідно аналізувати. Розбиваючи дані на менші фрагменти і обробляючи їх паралельно, можна проводити складні статистичні аналізи, а також навчати складні моделі прогнозування за невеликий час, порівняно із традиційними методами. Крім того, розподілена обробка даних може допомогти вирішити проблему дрейфу даних, яка

виникає, коли основні статистичні розподіли вхідних характеристик моделі швидко змінюються з часом через зміни продукції, функцій користувача або інших факторів, що аналізуються [3]. Обробка та аналіз даних у режимі реального часу може допомогти виявити ці зміни та адаптуватися до них, гарантуючи, що моделі прогнозування залишатимуться точними та актуальними.

Таким чином, розподілена обробка даних є потужним методом, який можна використовувати для обробки масштабних та складних наборів даних, особливо в контексті прогнозування часових рядів. Забезпечуючи швидший час обробки, більш ефективне використання обчислювальних ресурсів і покращену масштабованість, цей метод пропонує вирішення проблем, пов’язаних із постійно зростаючими обсягами даних, які генеруються в сучасному бізнес-середовищі.

1.2. Аналіз існуючих способів і засобів розподіленої обробки часових рядів

На сьогодні існують певні рішення для масштабованого прогнозування часових рядів: платформи автоматизованого машинного навчання, хмарні сервіси для прогнозування часових рядів, бази даних часових рядів.

1.2.1. AutoML моделі

Сучасним популярним рішенням є платформи автоматизованого машинного навчання, такі як H2O.ai, DataRobot і Google AutoML. Автоматизоване машинне навчання, яке також називають AutoML, — це процес автоматизації трудомістких ітеративних завдань розробки моделі

машинного навчання. AutoML потенційно включає всі етапи від початку роботи з необробленим набором даних до побудови моделі машинного навчання, готової до розгортання. Цей підхід дозволяє спеціалістам створювати моделі машинного навчання, не заглиблюючись в деталі.

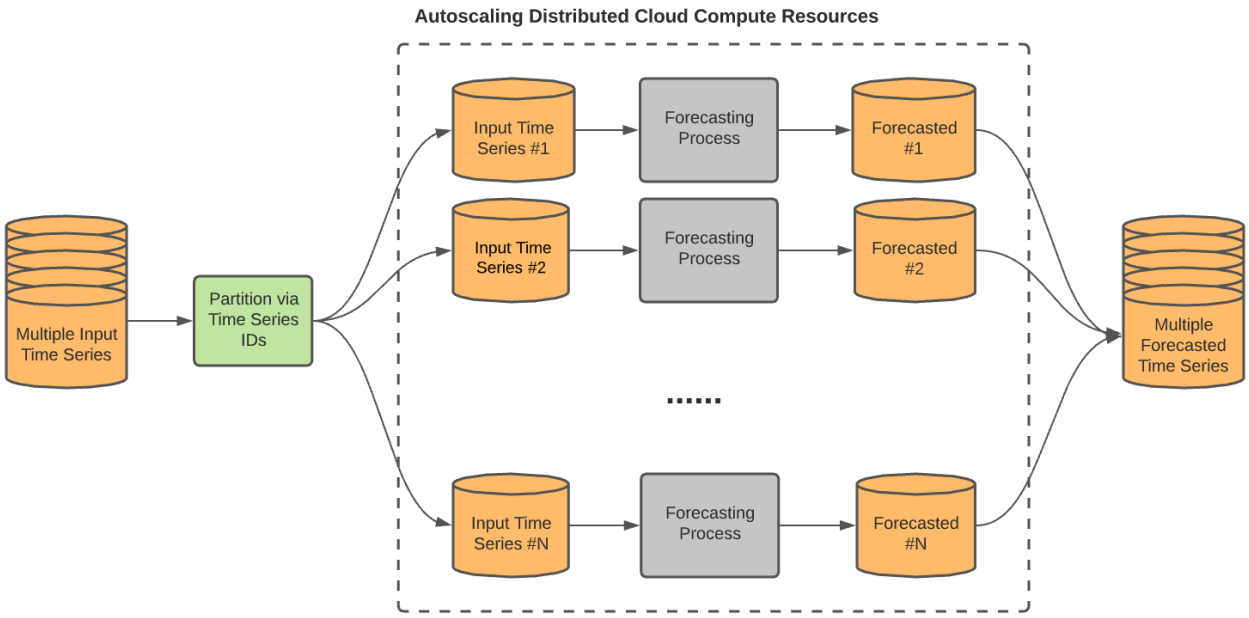

Рисунок 1.1 – Процес розподіленого прогнозування з використанням Google AutoML

Зокрема, такі платформи пропонують масштабовані рішення для прогнозування часових рядів, які використовують машинне навчання для автоматичного визначення найкращих моделей для кожного часового ряду (Рисунок 1.1). Проте, таке рішення недостатньо гнучке, оскільки згадані платформи мають за мету бути універсальним інструментом для різних задач машинного навчання. Так, наприклад, якщо дані потребують особливої

обробки, то доводиться реалізовувати виконання цієї обробки окремим кроком. До того ж, використовуючи платформу певної компанії, доведеться використовувати і сховища чи бази даних, які надає ця компанія. Це потребує додаткових коштів та часу на інтеграцію даних.

1.2.2. Хмарні сервіси

Хмарні сервіси, такі як Amazon Forecast, Microsoft Azure Time Series Insights і Google Timeseries Insights API, надають високомасштабовані рішення для прогнозування часових рядів, які можуть обробляти мільйони зразків часових рядів.

Microsoft Azure Time Series Insights - це потужна хмарна служба, яка дозволяє аналізувати та візуалізувати дані часових рядів у режимі реального часу, пропонуючи широкий спектр функцій, що полегшують дослідження даних, виявлення аномалій та предиктивну аналітику. Завдяки використанню передових алгоритмів машинного навчання та інших передових технологій, він дозволяє користувачам швидко виявляти закономірності та тенденції в даних часових рядів, отримуючи цінну інформацію, яку можна використовувати для покращення бізнес-операцій та результатів.

На противагу цьому, хоча Google Timeseries Insights API також пропонує можливості аналізу часових рядів, він є відносно новим сервісом, і тому йому бракує зрілості та стабільності деяких більш відомих конкурентів. Це означає, що в певних ситуаціях він може бути не таким надійним, як інші рішення. Тим не менш, важливо відзначити, що Google – це високоавторитетна та інноваційна компанія, а її сервіси постійно розвиваються і вдосконалюються з часом.

Обидва сервіси зосереджуються на аналізі даних та виявленні аномалій, і не мають широкого вибору інструментів для задач прогнозування.

Amazon Forecast (Рисунок 1.2) - це повністю керований сервіс, який використовує алгоритми машинного навчання для створення точних прогнозів на основі історичних даних.

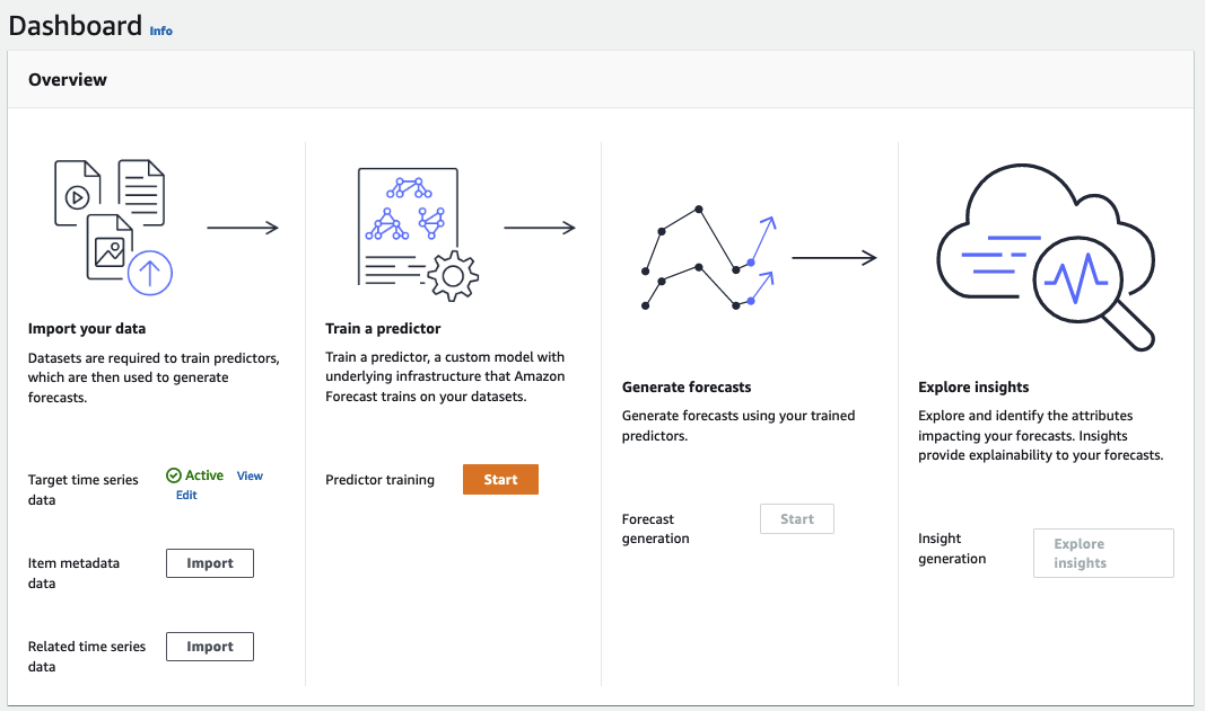

Рисунок 1.2 – Етапи обробки даних з Amazon Forecast

Він пропонує такі функції, як автоматичне очищення даних, врахування відсутніх даних та оптимізація гіперпараметрів моделі. Також цей сервіс дозволяє обирати різні алгоритми для прогнозування часових рядів:

- класичні NPTS, ARIMA, ETS;
- Deep Learning моделі CNN-QR, DeepAR+;
- модель Prophet;

Недоліком Amazon Forecast є потенційно висока вартість послуг у випадку використання для обробки масштабних об’ємів даних часових рядів. Адже модель ціноутворення Amazon базується на активності використання.

1.2.3. Бази даних часових рядів

Існують бази даних часових рядів, оптимізовані для зберігання великих обсягів даних часових рядів та оптимальних запитів до цього типу даних. Вони часто мають вбудовану функцію прогнозування.

Наприклад, TimescaleDB – це популярна реляційна база даних з відкритим вихідним кодом, спеціально розроблена для зберігання та запитів до даних часових рядів. Вона побудована на основі PostgreSQL, тому вона успадковує багато функцій і переваг PostgreSQL, таких як відповідність стандарту ACID, висока масштабованість і багата функціональність запитів.

Однак, TimescaleDB розширює PostgreSQL функціями, які спеціально розроблені для даних часових рядів: розбиття на розділи за часом, автоматичне індексування та оптимізовані запити, алгоритми стиснення, які допомагають зменшити вимоги до зберігання даних часових рядів. Також TimescaleDB підтримує розподілені гіпертаблиці, які дозволяють здійснювати горизонтальне масштабування на декількох вузлах. Однак, наявність лише класичних алгоритмів для прогнозування часових рядів робить цей засіб не оптимальним для поставленої задачі.

Порівняння рішень для масштабованого прогнозування часових рядів наведено в таблиці 1.1.

Назва	Моделі прогнозування часових рядів	Ліцензія на програмне забезпечення	Можливість кастомізації
Amazon Forecast	Класичні, Deep Learning, Prophet	SaaS	Низька
TimescaleDB	Класичні моделі	Open-source	Висока (SQL код)
Платформи AutoML	Немає вибору	SaaS	Низька або відсутня

Таблиця 1.1 – Порівняльна таблиця рішень для масштабованого прогнозування часових рядів

Висновки до розділу 1

У розділі 1 досліджено сучасні засоби для масштабованого прогнозування та обробки часових рядів: платформи автоматизованого машинного навчання, хмарні сервіси для прогнозування часових рядів та бази даних часових рядів.

Проте, ці засоби зазвичай змушують прив’язувати всі етапи роботи до однієї платформи. Тому для їх ефективного використання, компанія повинна або вже користуватися послугами даної платформи, або мати можливість легкого інтегрування до неї. В противному випадку, інтеграція даних і перехід на нову платформу може вимагати додаткових витрат часу та коштів.

Також вибір алгоритмів для прогнозування часових рядів є обмеженим.

Таким чином, наявні технічні та алгоритмічні рішення не мають достатньої гнучкості для розподіленої обробки часових рядів. Тому задача дослідження і визначення ефективних способів і засобів розподіленої обробки часових рядів на кластері є актуальною.

2. ОПИС ПІДХОДІВ ТА ЗАСОБІВ ДЛЯ РОЗПОДІЛЕНОЇ ОБРОБКИ ВЕЛИКИХ ОБ’ЄМІВ ДАНИХ

2.1. Поняття Big Data

Big Data – це термін, який використовується для опису великих і складних наборів даних, якими складно легко керувати, обробляти або аналізувати за допомогою традиційних методів обробки даних. Під великими даними зазвичай розуміють набори даних, які є занадто великими для зберігання чи обробки за допомогою звичайного апаратного та програмного забезпечення, або набори даних, які є надто різноманітними за своєю природою, наприклад, текст, зображення чи відео, щоб їх можна було проаналізувати за допомогою традиційних методів аналізу даних.

В результаті повсякденної генерації нових даних отримуємо великі колекції цінної інформації, яка є об’єктом керування, зберігання, візуалізації та аналізу для компаній.

Традиційні інструменти для роботи з даними не здатні впоратися з такою складністю, різноманіттям та обсягом даних. Це призвело до появи багатьох спеціалізованих програмних платформ для роботи з Big Data та архітектурних рішень, призначених для управління обчислювальними навантаженнями.

Виділяють три основних атрибути Big Data. Їх зазвичай називають три "V" Big Data:

- Обсяг (Volume) – великі обсяги даних, що зберігаються та обробляються.

- Швидкість (Velocity) – швидкість обробки та аналізу потоків даних повинна бути високою.
- Різноманітність (Variety) - різні джерела отримання даних, а також формати даних: числові, текстові, зображення, відео та аудіо дані.

Коли йдеться про Big Data - об’єми даних є дуже великими. Якщо традиційні дані вимірюються в таких розмірах, як мегабайти, гігабайти і терабайти, Big Data зберігаються в петабайтах і зеттабайтах. Наприклад, один гігабайт еквівалентний відео у форматі HD тривалістю 7 хвилин, тоді як один зеттабайт приблизно дорівнює 250 мільярдам DVD-дисків. Крім того, за даними Statista, виробництво даних зростає більш ніж удвічі за п'ять років, і очікується, що до 2025 року в усьому світі буде вироблено 180 зеттабайт. У сфері Big Data існують підходи, що дозволяють підтримувати архітектуру для роботи з такими даними. Без відповідних рішень для зберігання та обробки було б неможливо працювати з такою кількістю інформації, а отже, вона б втратила свою цінність.

Швидкість появи нових даних є дуже високою. Тому засоби для їх аналізу теж мають бути швидкими. Компанії та організації повинні мати можливість використовувати ці дані та добувати з них важливу інформацію в режимі реального часу, інакше дані припиняють бути корисними. Обробка даних у режимі реального часу дозволяє компаніям, діяти швидко, що дає їм перевагу над конкурентами.

Хоча деякі види даних можуть оброблятися постфактум пакетно і залишатися актуальними з часом, велика частина Big Data надходить в режимі реального часу і вимагає негайних дій для досягнення найкращих результатів. Одним із прикладів є сенсорні дані з медичних пристроїв. Можливість

миттєвої обробки даних про стан здоров'я пацієнта здатна надати користувачам і лікарям інформацію, яка потенційно може врятувати життя. Різноманіття даних зазвичай теж дуже високе. Приблизно від 80 до 90 відсотків усіх даних є неструктурованими, тобто неорганізованими і складними для аналізу за допомогою звичайних інструментів обробки даних. Все, від електронних листів і відео до наукових і метеорологічних даних, може становити потік Big Data. Кожен тип даних має свої унікальні особливості та атрибути, які потрібно враховувати.

2.2. Обробка та сфери застосування Big Data

Незважаючи на те, що масштабність Big Data може бути надзвичайно великою і складною для обробки, цей обсяг даних надає професіоналам велику кількість інформації, яку вони можуть використати для оптимізації. Великі масиви даних можна аналізувати, щоб виводити закономірності про їхні першоджерела. Це допомагає розуміти причинно-наслідкові зв’язки між подіями і різними факторами. Результати довзоляють генерувати ідеї для підвищення ефективності бізнесу або прогнозувати майбутні бізнес-результати.

До сфер застосування, в яких Big Data можуть бути корисними, можна віднести наступні:

- Оптимізація витрат
- Збільшення клієнтської бази
- Прийняття бізнес-рішень
- Автоматизація процесів

Різноманітніття даних в сфері Big Data робить їх складними для аналізу, що призводить до необхідності використання систем, здатних обробляти різні структури та формати. Неструктуровані дані часто потребують спеціалізованих NoSQL баз даних, які можуть зберігати дані у спосіб, що не вимагає суворого дотримання певної моделі. Це забезпечує гнучкість, необхідну для узгодженого аналізу різних типів інформації, та дозволяє отримати цілісне уявлення про середовище, що є джерелом цих даних.

При агрегуванні, обробці та аналізі дані та системи для їх обробки часто поділяють на оперативні або аналітичні дані. Цей поділ також впливає на спосіб зберігання даних. Оперативні системи зберігають та обробляють великі масиви даних на кількох серверах До таких даних можна віднести дані інвентаризації, дані щодо клієнтів та закупівель, повсякденні дані компаній. Аналітичні системи є більш досконалими, ніж їхні оперативні аналоги – вони здатні проводити складний аналіз даних і надавати бізнесу інформацію для прийняття рішень. Ці системи часто інтегруються в існуючі процеси та інфраструктуру, щоб максимально підвищити ефективність збору і використання даних.

Аналітика Big Data використовується майже в кожній галузі для виявлення закономірностей і тенденцій, відповідей на питання, отримання інформації про клієнтів і вирішення складних проблем. Компанії та організації використовують цю інформацію з багатьох причин: для розвитку бізнесу, розуміння рішень клієнтів, покращення досліджень, складання прогнозів та таргетування ключових аудиторій для реклами.

Фінансова та страхова галузі використовують Big Data та предиктивну аналітику для виявлення шахрайства, оцінки ризиків, кредитних рейтингів, брокерських послуг та технології блокчейн. Фінансові установи також

використовують Big Data для посилення своєї кібербезпеки та персоналізації фінансових рішень для клієнтів. Лікарні, дослідники та фармацевтичні компанії впроваджують рішення на основі Big Data для покращення та розвитку охорони здоров'я. Маючи доступ до величезних обсягів даних про пацієнтів і населення, медики вдосконалюють методи лікування, проводять більш ефективні дослідження таких захворювань, як рак і хвороба Альцгеймера, розробляють нові ліки і отримують критично важливу інформацію про закономірності здоров'я населення. Netflix, Hulu чи будь-які інші стрімінгові сервіси, що надають рекомендації своїм користувачам, використовують Big Data та алгоритми, що здатні швидко обробляти дані всіх клієнтів. Медійні компанії аналізують звички читання, перегляду та прослуховування, щоб надавати індивідуальні рекомендації кожному користувачу. Netflix навіть використовує дані про графіку, заголовки та кольори, щоб приймати рішення щодо вподобань клієнтів. Big Data може використовуватися для селекції насіння, прогнозування врожайності, оцінки стану рослин на великих територіях. Така автоматизація стрімко розвиває сільськогосподарську галузь.

Також, Big Data суттєво вплинула на сферу електронної комерції та роздрібної торгівлі. Величезна кількість даних, отриманих з різних джерел, таких як взаємодія з клієнтами, онлайн-транзакції, соціальні мережі та операції в ланцюгах поставок, відкрила для бізнесу нові можливості для отримання інсайтів та оптимізації своєї діяльності. Однією з ключових сфер використання великих даних в електронній комерції та роздрібній торгівлі є клієнтська аналітика. Аналізуючи дані про клієнтів, компанії можуть зрозуміти споживчі вподобання, поведінку та патерни, що дозволяє їм персоналізувати маркетингові кампанії, пропозиції та рекомендації. Це допомагає покращити

клієнтський досвід, підвищити лояльність та збільшити продажі. Оптимізація ланцюгів постачання – ще одна важлива сфера, де великі дані мають значення. Маючи можливість аналізувати великі обсяги даних з ланцюга постачання, компанії можуть оптимізувати управління запасами, спростити логістику та зменшити витрати. Це призводить до підвищення операційної ефективності, швидшого виконання замовлень і зменшення дефіциту або надлишку запасів. Прогнозування попиту також значно покращується завдяки великим даним. Аналізуючи історичні дані про продажі, ринкові тенденції та інші відповідні дані, компанії можуть точно спрогнозувати майбутній попит на товари чи послуги. Це дозволяє їм оптимізувати рівень запасів, мінімізувати дефіцит і знизити ризик надлишкових запасів, що в кінцевому підсумку підвищує задоволеність клієнтів і знижує витрати.

Поряд з вищезазначеними сферами, аналітика Big Data охоплює майже всі галузі, змінюючи способи ведення бізнесу в сучасному масштабі. Також Big Data використовується у сферах реклами та маркетингу, бізнесу, освіти, технологій Інтернету речей та спорту.

Залежно від складності структури даних, вони можуть зберігатися в різних сховищах, таких як хмарні сховища даних або озера даних, звідки аналітики можуть отримати до них доступ у разі потреби. Існує досить багато сучасних хмарних рішень, які, як правило, включають компоненти зберігання, обчислень і клієнтської інфраструктури.

Серед найпопулярніших платформ – Google Cloud, AWS та Microsoft Azure, проте значно зросла кількість платформ, що надають більш спеціалізовані послуги. Зокрема, зростає популярність платформ, що спеціалізуються на машинному навчанні та науці про дані, таких як Databricks та Anyscale.

Однією з головних відмінностей між цими спеціалізованими платформами та більш загальними хмарними рішеннями є те, що ці платформи пропонують ряд потужних інструментів і функцій, які спрощують процес розробки і розгортання моделей машинного навчання, дозволяючи аналітикам даних і розробникам зосередитися на аналізі та інтерпретації результатів.

Databricks, зокрема, побудована на основі Apache Spark, потужного фреймворку для розподілених обчислень, і пропонує набір функцій, таких як спільні блокноти, оптимізовані бібліотеки машинного навчання та кластери з автоматичним масштабуванням. Anyscale, з іншого боку, - це платформа, яка фокусується саме на розподіленому машинному навчанні, надаючи ряд інструментів і сервісів для розробки і розгортання моделей машинного навчання в масштабі. Вона побудована на основі Ray, фреймворку розподілених обчислень з відкритим вихідним кодом, і включає такі функції, як розподілене налаштування гіперпараметрів і розподілена обробка даних.

Після того, як дані вже зібрані, їх потрібно перетворити в найбільш зручні для сприйняття форми, щоб отримати відповіді на запитання бізнесу. Для цього існують різні варіанти обробки даних. Вибір правильного підходу може залежати від обчислювальних можливостей та аналітичних завдань компанії, а також від наявних ресурсів. В залежності від використання однієї або декількох машин, середовища обробки поділяються на централізовані та розподілені.

Централізована обробка передбачає, що вся обробка відбувається однією комп'ютерною системою (виділеному сервері, на якому зберігаються всі дані). Таким чином, кілька користувачів одночасно користуються одними і тими ж ресурсами і даними. Разом з єдиною точкою контролю з'являється і єдина

точка відмови. Якщо відбудеться збій в системі, то з ладу вийде вся система, а не лише її частина.

Розподілена обробка використовується, коли набори даних занадто великі, щоб їх можна було обробити на одній машині. Такий підхід дозволяє розбивати великі набори даних на менші частини і зберігати їх на декількох серверах. Це, в свою чергу, дає можливість обробляти дані паралельно. Перевагою розподіленого підходу є те, що завдання з обробки даних можуть бути перенесені на інші доступні сервери у випадку, якщо один сервер у мережі вийде з ладу.

Залежно від часу обробки виділяють пакетну та реальну обробку даних.

Пакетна обробка - це метод, при якому фрагменти даних, накопичені протягом певного періоду часу, обробляються пакетами. Це відбувається в той час, коли обчислювальні ресурси легко доступні, що дозволяє економити на них, але вимагає певного часу для виконання пакетних завдань. Пакетна обробка може бути обрана замість обробки в реальному часі, коли точність значно важливіша за швидкість.

Обробка в режимі реального часу гарантує, що дані завжди актуальні завдяки безперервному введенню, перетворенню та виведенню елементів даних. Цей тип обробки передбачає, що всі обчислювальні операції виконуються протягом короткого проміжку часу, зазвичай за лічені секунди або мілісекунди. Прикладом можуть бути рекомендаційні системи, що обробляють дії клієнта в реальному часі, щоб надати нові рекомендації. Більш складна з точки зору реалізації, обробка в реальному часі є хорошим варіантом для швидшого прийняття рішень.

2.3. Аналіз технологій та засобів для роботи з Big Data

Як зазначалось вище, масштабні об’єми даних зазвичай важко обробити чи проаналізувати за допомогою традиційних методів аналізу даних. Для проведення складного аналізу за прийнятний час використовуються засоби, створені саме для роботи з Big Data. Засоби Big Data здатні контролювати великі масиви даних і виявляти закономірності в розподіленому масштабі і в режимі реального часу, заощаджуючи велику кількість часу і грошей.

Проаналізуємо декілька популярних інструментів для роботи з Big Data, які сьогодні використовуються в різних галузях.

2.3.1. Apache Hadoop

Apache Hadoop є фреймворком на базі Java з відкритим вихідним кодом, який надає засоби пакетної обробки та зберігання даних на групі апаратних машин, об'єднаних у кластери. Hadoop створений для надійних, масштабованих та розподілених обчислень. В той же час, він може використовуватися як файлове сховище загального призначення. Цей фреймворк здатний зберігати і обробляти петабайти інформації.

2.3.2. Apache Spark

Apache Spark – це фреймворк пакетної обробки з покращеною обробкою потоків даних. Завдяки виконанню операцій в пам'яті (на відміну від Hadoop), а також наявності засобів оптимізації порядку обчислень, він є надзвичайно швидкою кластерною обчислювальною системою [6].

Spark може працювати самостійно, підтримуючи роботу кластеру, або забезпечити послідовну інтеграцію з Hadoop. Фреймворк підтримує такі популярні мови програмування, як Python, R, Java та Scala.

Перевагами Spark є швидкодія, розширений набір операцій для аналізу даних порівняно із Hadoop, наявність додаткових пакетів з реалізацією алгоритмів машинного навчання, роботи із графами. Також, це достатньо поширена та стабільна технологія, тому багато компаній в сфері обробки даних та розподілених обчислень підтримують її та забезпечують розвиток.

2.3.3. Dask

Dask – фреймворк для паралельних обчислень з відкритим вихідним кодом, створений в 2015 році, тому є відносно новим в порівнянні зі Spark. Також, на відміну від Spark, одним з принципів проектування, прийнятих при розробці Dask, було зробити бібліотеку максимально зручною для розробників, які використовують Pandas в задачах аналізу даних. Зважаючи на це, додано підтримку паралельного виконання операцій над Pandas DataFrames. Також, є можливість запускати на кластері алгоритми популярної бібліотеки для машинного навчання scikit-learn. Dask дозволяє зручно працювати в багатомашинному кластері з обсягами даних в декілька терабайт.

2.3.4. Ray

Ray – ще один проект для спрощення розподілених обчислень. Ray складається з двох основних компонентів: Ray Core, який є фреймворком розподілених обчислень, та Ray Ecosystem, який фактично є низкою бібліотек для конкретних завдань, які постачаються разом з Ray.

Ray схожий на Dask тим, що дозволяє користувачеві використовувати код на Python для паралельної обробки даних на всіх машинах кластера. Однак, на відміну від Dask, Ray не намагається імітувати API Pandas. Він лише надає інтерфейс для виконання коду на Python паралельно. Це робить його універсальним фреймворком, який можна використовувати для створення та запуску розподілених додатків будь-якого типу. Стандартний спосіб завантаження та обміну даними у бібліотеках та додатках Ray – це Ray Datasets. Вони надають інструменти для базових розподілених перетворень даних, такі як maps (map_batches), агрегації на згрупованих даних (GroupedDataset), операції перемішування (random_shuffle, sort, repartition). Ray Datasets сумісні з різними форматами файлів, джерелами даних і розподіленими фреймворками.

Ray Datasets призначені для завантаження та попередньої обробки даних для розподілених конвеєрів навчання ML. Порівняно з іншими рішеннями для завантаження, Ray Datasets є більш гнучкими і забезпечують вищу загальну продуктивність.

Datasets складаються зі списку посилань об'єктів Ray на блоки. Кожен блок містить набір елементів у форматі таблиці Arrow або списку Python (для нетабличних даних). Ray Datasets також підтримує змішування тензорних і табличних даних, що може бути корисним при роботі з методами глибинного навчання. Наявність декількох блоків у наборі даних дозволяє проводити їх паралельне перетворення та використання.

На рисунку 2.1 візуалізовано табличний набір даних з трьома блоками, кожен з яких містить 1000 рядків:

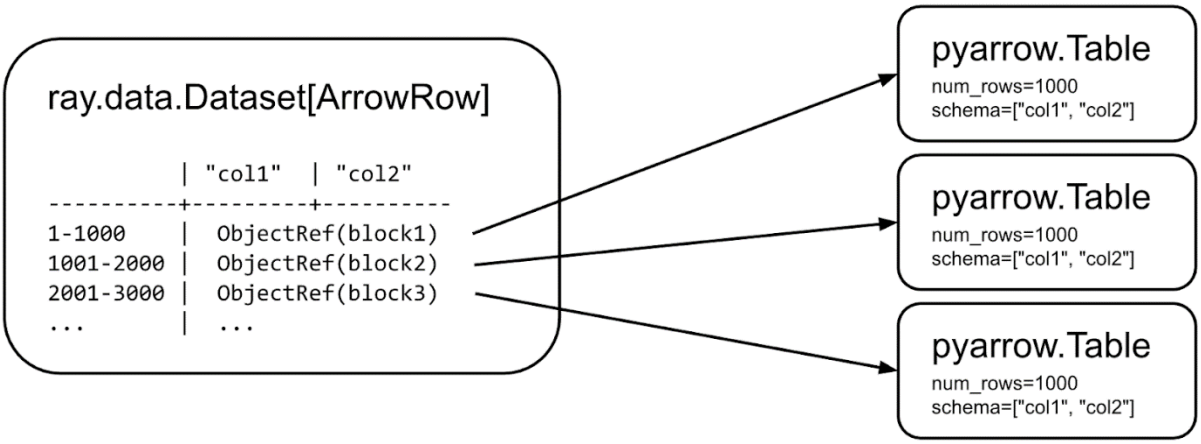

Рисунок 2.1 – Внутрішній формат Ray Datasets

Оскільки набір даних – це просто список посилань на об'єкти Ray, його можна вільно передавати між завданнями, акторами та бібліотеками Ray, як будь-яке інше посилання на об'єкт. Ця гнучкість є унікальною характеристикою наборів даних Ray. Ray Datasets використовує Ray-завдання для читання даних з віддаленого сховища. При читанні з файлового джерела даних (наприклад, S3, GCS) він створює таку кількість Ray-завдань для читання даних, яка пропорційна кількості процесорів у кластері.

Кожне Ray-завдання для читання даних читає призначені їй файли і створює блок виводу (рисунок 2.2).

Рисунок 2.2 – Розподілене читання даних Ray Datasets

Набори даних можуть використовувати або Ray-завдання, або Ray-акторів для перетворення наборів даних. За замовчуванням використовуються завдання. Акторів можна вказати за допомогою compute=ActorPoolStrategy(), яка створює пул Ray-акторів з автоматичним масштабуванням для обробки перетворень.

Використання акторів дозволяє кешувати операції ініціалізації станів, які зазвичай є обчислювально важкими (рисунок 2.3).

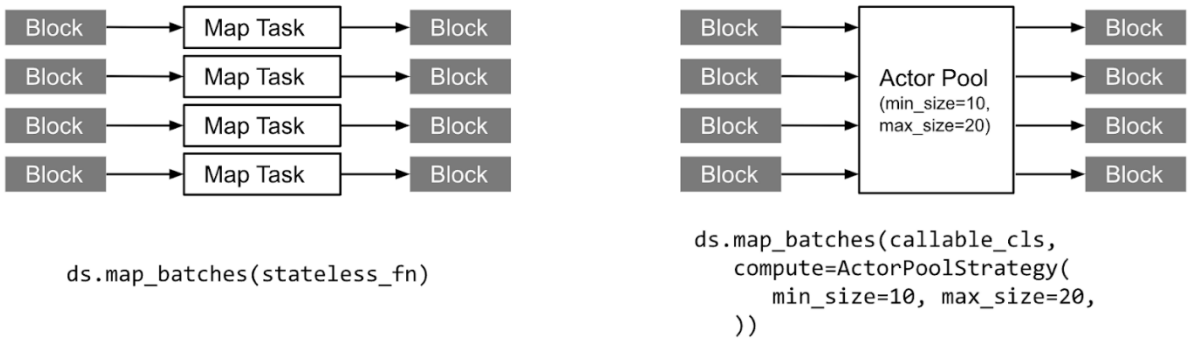

Рисунок 2.3 – Розподілене застосування операцій до Ray Datasets

Деякі операції, такі як сортування або групування, вимагають розділення блоків даних за значенням або перемішування. Datasets використовує Ray-завдання для реалізації розподіленого перемішування у стилі map-reduce, використовуючи Ray-завдання map для розділення блоків за значенням, а потім Ray-завдання reduce для об'єднання розділених блоків разом.

Перемішування наборів даних може масштабуватися до обробки сотень терабайт даних. Набори даних покладаються на відмовостійкість на основі завдань в ядрі Ray. Зокрема, набір даних буде автоматично відновлений Ray у разі збоїв. Це працює завдяки реконструкції лінії: набір даних - це колекція об'єктів Ray, що зберігаються у спільній пам'яті, і якщо будь-який з цих об'єктів буде втрачено, то Ray відтворить його, повторно виконавши завдання, які його створили.

Конвеєри даних дозволяють виконувати перетворення набору даних інкрементно над вікнами базових даних, а не над усіма даними одразу (рисунок 2.4). Це може бути використано для потокового завантаження даних у навчання ML або для виконання пакетних перетворень над великими наборами даних без необхідності завантажувати весь набір даних у пам'ять кластера.

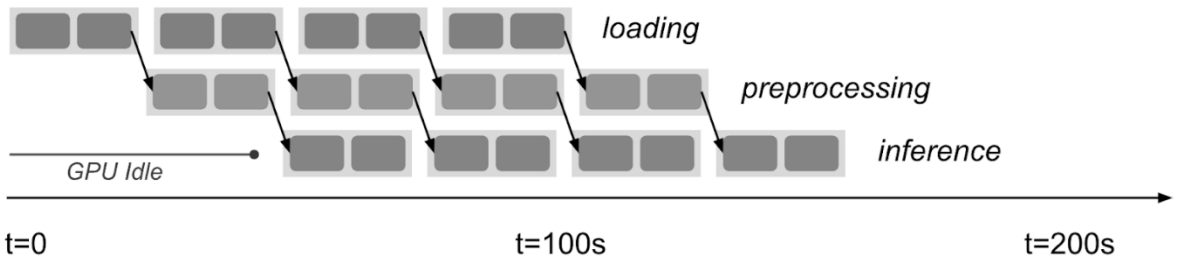

Рисунок 2.4 – Візуалізація роботи конвеєрів даних в Ray Datasets

Конвеєри наборів даних можуть зчитуватися в потоковому режимі одним споживачем або розділятися на кілька підконвеєрів і зчитуватися паралельно кількома споживачами для розподіленого навчання.

2.3.5. Google Cloud Storage

Google Cloud Storage – це хмарний сервіс зберігання об'єктів, який надає користувачам гнучку та масштабовану платформу для зберігання та доступу до їхніх даних. Цей сервіс є частиною хмарної платформи Google Cloud Platform, яка пропонує набір послуг хмарних обчислень, включаючи зберігання, обчислення та аналітику даних. Google Cloud Storage дозволяє користувачам зберігати об'єкти будь-якого розміру, від декількох байт до декількох терабайт, і надає простий API для доступу до даних та управління ними.

Платформа розроблена таким чином, щоб бути високодоступною та відмовостійкою, з вбудованим багаторівневим резервуванням для забезпечення довговічності даних. Це гарантує, що дані завжди доступні та доступні для користувачів, незалежно від того, де вони знаходяться. Крім того, Google Cloud Storage підтримує широкий спектр типів даних і методів доступу, включаючи структуровані та неструктуровані дані, пакетну обробку та обробку в реальному часі, а також різні класи сховищ. Це надає користувачам гнучкість в оптимізації витрат на зберігання даних відповідно до їхніх конкретних потреб.

2.3.6. Apache Parquet

Apache Parquet – широко використовуваний, з відкритим вихідним кодом, стовпцево-орієнтований формат файлів даних, розроблений для ефективного зберігання та пошуку даних. Він призначений для забезпечення високопродуктивної обробки складних даних у великих обсягах і пропонує різні ефективні схеми стиснення та кодування даних. Формат не залежить від мови і зазвичай використовується для аналітики (OLAP) у поєднанні з традиційними базами даних OLTP.

Однією з ключових характеристик Parquet є те, що це формат на основі стовпців, тобто файли організовані за стовпцями, а не за рядками. Такий підхід економить місце в сховищі і прискорює аналітичні запити, особливо ті, які потребують зчитування певних стовпців з великої таблиці. Parquet підтримує складні типи даних і розширені вкладені структури даних, що робить його універсальним варіантом для зберігання великих даних будь-якого типу, включаючи структуровані таблиці даних, зображення, відео та документи.

Також Parquet дозволяє робити так зване партиціонування, тобто, розбиття на розділи. Це дозволяє поділити великий файл на менші, більш керовані частини. Зокрема, розбиття на розділи передбачає поділ даних на менші фрагменти на основі певних критеріїв, таких як дата, час або конкретні значення в стовпчику. Основна мета розбиття на розділи – зменшити обсяг даних, які потрібно обробити для отримання бажаних результатів, і таким чином підвищити продуктивність запитів.

Розбиття на розділи у файлах Parquet працює шляхом створення окремих файлів для кожного розділу на основі заданих критеріїв. Наприклад, користувач може розділити свої дані на основі стовпця дати, в результаті чого будуть створені окремі файли для кожного дня. При виконанні запиту система сканує лише відповідні розділи, а не весь набір даних, мінімізуючи обсяг

даних, які необхідно обробити. Такий підхід значно підвищує продуктивність запитів і особливо корисний при роботі з великими масивами даних.

2.4. Глибинне навчання і Big Data

У випадку складних задач, прийнято розбивати його на менші підзадачі та виконувати їх паралельно. Такий підхід не лише економить час, але й робить складні завдання більш зрозумілими і простими. У контексті глибокого навчання використовується схожа стратегія, відома як розподілене навчання.

Розподілене навчання передбачає розподіл обчислень між декількома процесорами, які часто називають робочими вузлами або просто робітниками, з метою навчання великомасштабної моделі глибокого навчання. Таке розпаралелювання навчання дозволяє прискорити час навчання. Загалом, існує два основних підходи до паралелізму в розподіленому навчанні: паралелізм даних і паралелізм моделі.

Паралелізм даних – це підхід, при якому дані поділяються на n розділів, де n - загальна кількість доступних робочих вузлів в обчислювальному кластері. Кожен робочий вузол оснащений копією моделі і відповідає за її навчання на власній підмножині даних. Цикли навчання виконуються синхронно або асинхронно, залежно від конкретної реалізації.

Такий підхід дозволяє паралельно обробляти навчальне навантаження, коли кожен робочий вузол працює незалежно над своєю частиною даних. Синхронні або асинхронні навчальні цикли сприяють ефективній координації між працівниками, гарантуючи, що процес навчання моделі виконується організовано та ефективно.

Паралелізм даних є широко використовуваною технікою в розподіленому глибокому навчанні, оскільки він дозволяє швидше навчати великі моделі на великих наборах даних. Розподіляючи дані і модель між кількома працівниками, процес навчання можна прискорити, що робить більш реальним вирішення складних завдань глибокого навчання, які в іншому випадку були б трудомісткими або нездійсненними при одновузловому навчанні.

2.4.1. Синхронне навчання

Як було описано вище, при паралельній обробці даних датасет ділиться на частини. Кожна частина надсилається окремому працівнику. Кожен працівник має повну копію моделі і навчається лише на частині даних. При синхронному навчанні прохід вперед починається одночасно у всіх робітників, і вони обчислюють різні вихідні дані та градієнти. Тут кожен працівник чекає, поки всі інші працівники завершать свої цикли навчання і обчислять свої відповідні градієнти. Після того, як всі працівники завершили обчислення градієнтів, вони починають спілкуватися один з одним і об'єднують градієнти, використовуючи алгоритм повної редукції. Після того, як всі градієнти об'єднано, копія оновлених градієнтів надсилається всім працівникам. Тепер, отримавши оновлені градієнти за допомогою алгоритму all-reduce, кожен працівник продовжує зворотній прохід і оновлює локальну копію ваг у звичайному режимі. Поки всі робітники не оновлять свої ваги, наступний прохід вперед не почнеться, тому він називається синхронним.

Важливо зазначити, що всі працівники створюють різні градієнти, оскільки вони навчаються на різних підмножинах даних, але в будь-який момент часу всі працівники мають абсолютно однакові ваги. Використовуючи алгоритм

повного зведення, всі робітники повинні розділити навантаження зі зберігання та підтримки глобальних параметрів. Згідно алгоритму кожен робітник ділиться своїми градієнтами з усіма іншими робітниками і застосовує операцію редукції. Тобто, алгоритм зводить цільові масиви у всіх робітників до одного масиву і повертає отриманий масив всім робітникам.

Існують різні реалізації алгоритмів повного зведення, які визначають, як обчислюються і передаються ці параметри. В одній реалізації всі робітники надсилають свої градієнти одному робітнику, відомому як водій, який відповідає за зведення градієнтів і надсилання оновлених градієнтів усім робітникам. Але проблема такого підходу полягає в тому, що водій стає точкою затримки, оскільки його комунікація та застосування операції зменшення зростають зі збільшенням кількості процесів, а отже, такий підхідпогано масштабується.

Тому може бути використано інший підхід, який називається "кільцеве скорочення", в якому працівники об'єднані в кільце. Кожен працівник відповідає за певну підмножину параметрів, які доступні лише наступному працівнику в кільці. Цей алгоритм значно зменшує накладні витрати на синхронізацію.

2.4.2. Асинхронне навчання

Асинхронне навчанн ефективніше використовує ресурси, дозволяючи працівникам працювати незалежно, не чекаючи на інших працівників у кластері. Це особливо корисно, коли існує значна різниця в обчислювальних потужностях між працівниками, оскільки синхронний підхід може бути сповільнений найповільнішим працівником у кластері.

Одним з поширених підходів для досягнення асинхронного навчання є використання сервера параметрів. У цьому випадку існує окрема сутність, яка називається сервером параметрів, що зберігає та керує глобальними параметрами моделі. Робітники виконують проходження вперед і назад незалежно, не чекаючи на інших робітників. Замість того, щоб синхронізувати градієнти, як при синхронному навчанні, працівники безпосередньо оновлюють глобальні параметри на сервері параметрів.

Кожен працівник обчислює свої градієнти на основі локальних даних і надсилає їх на сервер параметрів. Сервер параметрів потім оновлює глобальні параметри, використовуючи ці градієнти. Інші працівники можуть продовжувати виконувати свої навчальні цикли і не чекати, поки сервер параметрів завершить оновлення глобальних параметрів. Це дозволяє працівникам прогресувати незалежно, що робить асинхронне навчання потенційно швидшим порівняно з синхронним, оскільки працівникам не потрібно чекати один на одного.

Однак асинхронне навчання може також створювати проблеми, такі як застарілість параметрів, коли працівники можуть оновлювати сервер параметрів із застарілими градієнтами, і неузгодженість між працівниками через відсутність синхронізації. Для забезпечення збіжності та стабільності асинхронного навчання необхідне ретельне проектування та управління сервером параметрів і оновленнями на боці кожного працівника.

2.4.3. Сервер параметрів

У розподіленому навчанні працівникам можуть бути призначені різні ролі для оптимізації процесу навчання. Одним з поширених підходів є

використання деяких працівників як серверів параметрів, а решти – як працівників, що навчають.

Сервери параметрів відповідають за зберігання глобальних параметрів моделі та оновлення глобального стану. Вони отримують градієнти від працівників, що навчають, і відповідно оновлюють глобальні параметри. Працівники, що навчають, з іншого боку, виконують власне цикл навчання на своїх локальних даних, обчислюють градієнти і надсилають їх на сервери параметрів для оновлення глобальних параметрів.

Призначаючи окремі ролі працівникам, можемо досягти паралелізму в процесі навчання. Працівники, які навчають, можуть виконувати обчислення незалежно на своїх локальних даних, в той час як сервери параметрів обробляють агрегацію та оновлення глобальних параметрів. Це потенційно може прискорити процес навчання і підвищити загальну ефективність розподіленого навчання.

У контексті розподіленого навчання цей процес можна описати наступним чином:

1. Реплікація моделі. Модель реплікується для всіх працівників у кластері, і кожному працівнику призначається підмножина даних для навчання.
2. Отримання параметрів. Кожен працівник, що навчається, отримує параметри з серверів параметрів, які відповідають за зберігання глобального стану моделі.
3. Цикл навчання. Кожен навчальний робот виконує ітеративний цикл навчання, обчислюючи градієнти та значення втрат на основі заданих даних, і надсилає обчислені градієнти назад на всі сервери параметрів.

4. Оновлення параметрів моделі. Сервери параметрів отримують градієнти від декількох працівників, що навчаються, і відповідно оновлюють глобальні параметри моделі.

Такий підхід до розподіленого навчання дозволяє кожному працівнику навчати модель незалежно, не чекаючи, поки інші працівники виконають свої завдання.

Однак важливо зазначити, що цей підхід також має деякі недоліки в контексті розподіленого навчання:

1. Застаріла версія моделі. У будь-який момент часу лише один працівник використовує оновлену версію моделі, в той час як інші можуть використовувати застарілі версії моделі, оскільки сервери параметрів оновлюють параметри моделі асинхронно.
2. Так зване вузьке місце та єдина точка відмови. Якщо єдиним сервером параметрів є одна робоча машина, вона може стати вузьким місцем і єдиною точкою відмови у великих кластерах. Однак цю проблему вузького місця можна вирішити, створивши кілька паралельних серверів параметрів.

2.4.4. Паралелізм моделі

У контексті розподіленого навчання паралелізм моделі – це метод, який використовується, коли розмір моделі занадто великий, щоб з нею міг впоратися один працівник. При паралелізмі моделі, також відомому як мережевий паралелізм, модель ділиться горизонтально або вертикально на

різні частини, які можуть виконуватися одночасно різними працівниками, при цьому кожен працівник обробляє одні й ті ж дані (рисунок 2.5). Спільні параметри моделі зазвичай синхронізуються один раз для кожної ітерації прямого або зворотного поширення.

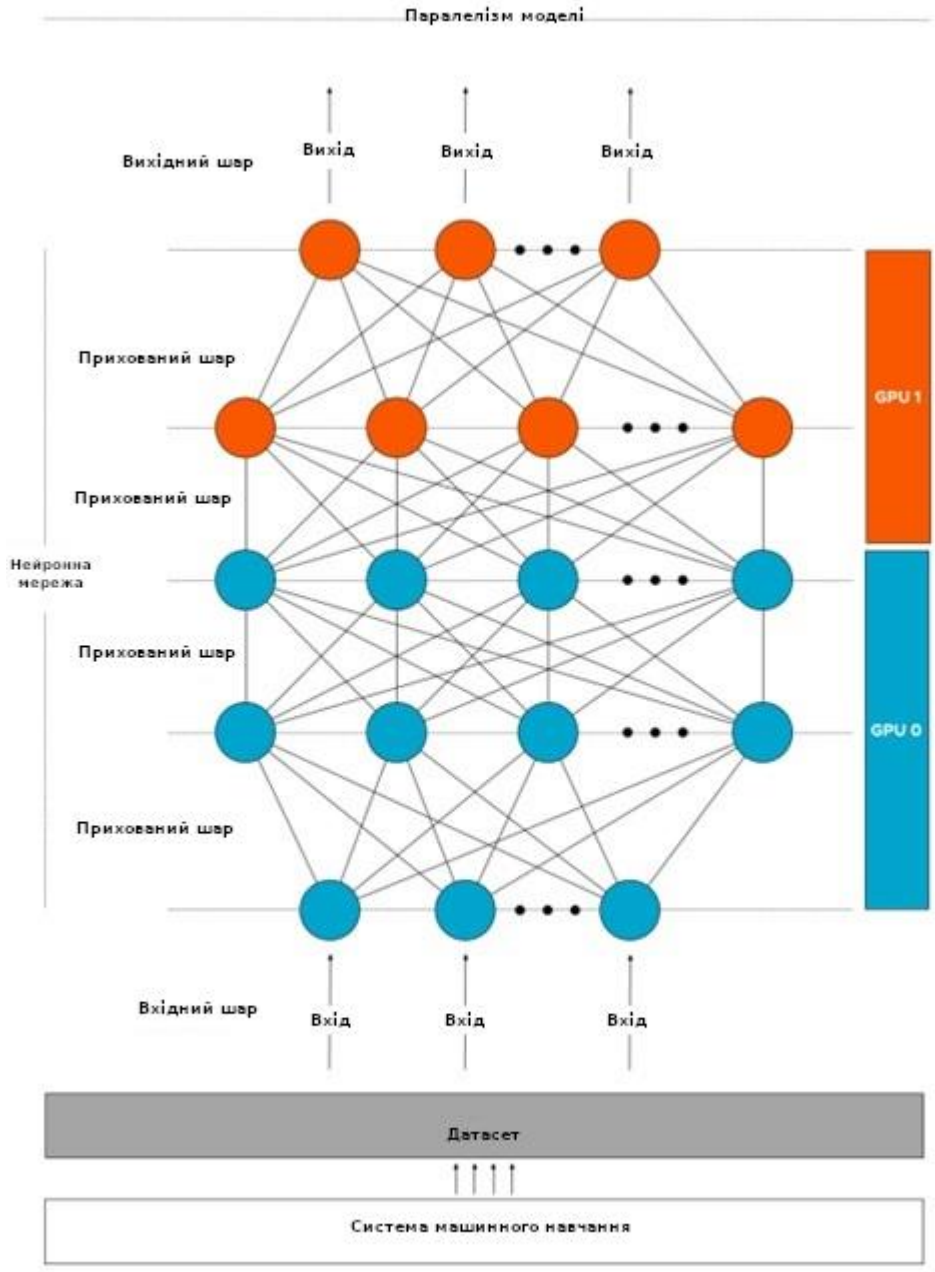

Рисунок 2.5. – Вертикальне розбиття моделі

Вертикальне розбиття не впливає на шари моделі, що робить його застосовним до будь-якої моделі глибокого навчання. Вертикальному розбиттю часто надають перевагу, тоді як горизонтальне розбиття розглядається як крайній засіб, коли немає інших можливостей вмістити шар у пам'ять однієї машини. У деяких випадках паралелізм моделі може бути застосований у більш простий спосіб, наприклад, навчання кодера і декодера окремо в архітектурі кодер-декодер.

Моделі NLP, такі як Transformers, GPT-3, BERT тощо, зазвичай використовують паралелізм моделей через їхній великий розмір і складність. Розділяючи модель на менші частини і розподіляючи їх між кількома працівниками, паралелізм моделей дозволяє ефективно навчати ці великі моделі, долаючи при цьому обмеження обсягу пам'яті однієї машини.

Важливо ретельно продумати дизайн і реалізацію паралелізму моделей у розподіленому навчанні, оскільки він може спричинити додаткові проблеми, такі як накладні витрати на синхронізацію, балансування навантаження та ефективність комунікації. Правильне управління цими проблемами має вирішальне значення для ефективного використання паралелізму моделей і досягнення ефективного і масштабованого розподіленого навчання великих моделей глибокого навчання.

Висновки до розділу 2

У розділі 2 представлено загальні концепції великих даних та особливості їх застосування в різних галузях.

Відзначено, що Big Data є найважливішим аспектом сучасного управління даними, яке сприяло створенню і розвитку різноманітних технологій, способів і засобів для роботи з великими даними.

Розглянуто та проаналізовано сучасні технології та засоби для роботи з великими даними, зокрема розподілені файлові системи, бази даних та фреймворки для обробки даних. Ці засоби дозволяють зберігати, аналізувати та керувати великими обсягами даних, що раніше було складно або навіть неможливо аналізувати.

Виконано порівняння основних підходів до розподіленого навчання моделей на багатьох машинах у кластері. Розподілене навчання моделей стало важливою технікою для навчання моделей машинного навчання на великих наборах даних. Визначено основні недоліки та переваги розподіленого навчання моделей, а також різні підходи для навчання моделей на великих наборах даних.

В той же час залишається проблема складності застосування описаних засобів та підходів на практиці. Адже для забезпечння їх роботи потрібно забезпечити правильне керування кластером та працівниками, а також синхронізація між усіма задачами, що виконуються. Необхідно враховувати накладні витрати на синхронізацію, балансування навантаження та ефективність комунікації.

3. АЛГОРИТМИ ПРОГНОЗУВАННЯ ЧАСОВИХ РЯДІВ

3.1. Методи і цілі прогнозування

Прогнозування потрібне в багатьох ситуаціях: рішення про будівництво нової електростанції протягом наступних п'яти років вимагає прогнозів майбутнього попиту; планування роботи персоналу колл-центру на наступний тиждень вимагає прогнозів обсягів дзвінків; формування запасів продуктів на складі вимагає прогнозів потреб у запасах. Прогнози можуть бути потрібні на кілька років наперед (у випадку капітальних інвестицій) або лише на кілька хвилин (для телекомунікаційної маршрутизації). Незалежно від обставин чи часових горизонтів, прогнозування є дуже важливим для ефективного та результативного планування.

Передбачуваність події або величини залежить від кількох факторів, а саме:

- наскільки добре ми розуміємо фактори, що впливають на неї;
- скільки даних є в наявності;
- чи можуть прогнози вплинути на те, що ми намагаємося передбачити.

Наприклад, прогнози попиту на електроенергію можуть бути дуже точними, оскільки всі три умови зазвичай виконуються. Ми добре знаємо фактори, що впливають на попит: попит на електроенергію значною мірою визначається температурою, впливом календарних подій, таких як свята, та економічних умов. За умови наявності достатньої кількості даних про попит на електроенергію та погодні умови, а також за наявності навичок розробки якісної моделі, що пов'язує попит на електроенергію та ключові змінні, прогнози можуть бути напрочуд точними.

З іншого боку, при прогнозуванні валютних курсів виконується лише одна з умов: є багато доступних даних. Однак ми маємо обмежене розуміння факторів, які впливають на обмінні курси, а прогнози обмінного курсу мають безпосередній вплив на самі курси. Якщо є широко розрекламовані прогнози про те, що обмінний курс зросте, то люди негайно скоригують ціну, яку вони готові платити, і таким чином прогнози самоздійснюються. У певному сенсі, обмінні курси стають своїми власними прогнозами. Це приклад "гіпотези ефективного ринку". Отже, прогнозування того, зросте чи знизиться обмінний курс завтра, є приблизно таким же передбачуваним, як і прогнозування того, чи впаде підкинута монета - орлом чи решкою. В обох ситуаціях ви будете праві приблизно в 50% випадків, незалежно від того, що ви прогнозуєте. У таких ситуаціях прогнозисти повинні усвідомлювати власні обмеження і не претендувати на більше, ніж можливо.

Часто в прогнозуванні ключовим кроком є розуміння того, коли щось можна передбачити точно, а коли прогнози будуть не кращими за підкидання монети. Хороші прогнози відображають справжні закономірності та взаємозв'язки, які існують в історичних даних, але не повторюють минулі події, які більше не повторяться.

Багато людей помилково вважають, що прогнозування неможливе у мінливому середовищі. Будь-яке середовище змінюється, і хороша модель прогнозування відображає спосіб, у який ці зміни відбуваються. Прогнози рідко припускають, що середовище є незмінним. Зазвичай припускають, що спосіб, у який середовище змінюється, продовжуватиметься і в майбутньому. Тобто, дуже нестабільне середовище і надалі залишатиметься дуже нестабільним; бізнес з нестабільними продажами і надалі матиме нестабільні

продажі; а економіка, яка переживала буми і спади, і надалі переживатиме буми і спади.

Ситуації, що прогнозуються, дуже різняться за своїми часовими горизонтами, факторами, що визначають фактичні результати, типами даних та багатьма іншими аспектами. Методи прогнозування можуть бути простими, наприклад, використання останнього спостереження в якості прогнозу (наївний метод), або дуже складними, такими як нейронні мережі. Іноді дані можуть бути відсутні взагалі. Наприклад, ми хочемо спрогнозувати продажі нового продукту в перший рік його існування, але очевидно, що немає даних, з якими можна було б працювати. Вибір методу залежить від наявних даних і передбачуваності прогнозованої величини.

Відповідні методи прогнозування значною мірою залежать від того, які дані є в наявності. Якщо даних немає, або якщо наявні дані не є релевантними для прогнозів, то необхідно використовувати якісні методи прогнозування. Ці методи не є просто здогадками - існують добре розроблені структуровані підходи до отримання хороших прогнозів без використання історичних даних. Але набагато більш поширеним є кількісне прогнозування. Його можна застосовувати, коли виконуються дві умови:

- наявна числова інформація про минуле;
- є підстави припускати, що деякі аспекти минулих тенденцій збережуться і в майбутньому.

Існує широкий спектр методів кількісного прогнозування, часто розроблених в рамках конкретних дисциплін для конкретних цілей. Кожен

метод має свої властивості, точність і особливості, які необхідно враховувати при виборі конкретного методу.

Зазвичай для кількісного прогнозування використовують або дані часових рядів (дані зібрані через регулярні проміжки часу).

Все, що спостерігається послідовно протягом певного часу, є часовим рядом. При прогнозуванні даних часових рядів метою є оцінка того, як послідовність спостережень буде продовжуватися в майбутньому.

Найпростіші методи прогнозування часових рядів використовують лише інформацію про змінну, що прогнозується, і не намагаються виявити фактори, які впливають на її поведінку. Тому вони екстраполюють тенденції та сезонні закономірності, але ігнорують всю іншу інформацію, таку як маркетингові ініціативи, діяльність конкурентів, зміни в економічних умовах тощо.

Моделі часових рядів, що використовуються для прогнозування, включають моделі декомпозиції, моделі експоненціального згладжування та моделі ARIMA.

Змінні-предиктори часто корисні при прогнозуванні часових рядів. Наприклад, припустимо, що ми хочемо спрогнозувати погодинний попит на електроенергію (ED) у спекотному регіоні протягом літнього періоду. Модель зі змінними предикторами може мати вигляд:

$$ED = f\begin{pmatrix}\text{current temperature, strength of economy,}\\ \text{population, time of day, day of week, error}\end{pmatrix}$$

Зв'язок не є точним - завжди будуть відбуватися зміни в попиті на електроенергію, які не можуть бути враховані за допомогою змінних-предикторів. Термін "error" враховує випадкові коливання та вплив відповідних змінних, які не включені в модель. Таку модель називають

пояснювальною, оскільки вона допомагає пояснити, що спричиняє коливання попиту на електроенергію.

Оскільки дані про попит на електроенергію формують часовий ряд, ми також можемо використовувати модель часового ряду для прогнозування. У цьому випадку відповідне рівняння прогнозування часового ряду має вигляд:

$$ED_{t+1} = f(ED_t, ED_{t-1}, ED_{t-2}, \dots, error),$$

де t - поточна година, t-1 - минула година, t-2 - дві години тому, і так далі. Тут передбачення майбутнього базується на минулих значеннях змінної, а не на зовнішніх змінних, які можуть впливати на систему. Знову ж таки, термін "error" праворуч враховує випадкові коливання та вплив відповідних змінних, які не включені в модель.

Існує також третій тип моделей, який поєднує в собі риси двох вищезгаданих моделей. Такі моделі можна подати у вигляді:

$$ED_{t+1} = f\begin{pmatrix} ED, \text{current temperature, strength of economy,} \\ \text{population, time of day, day of week, error} \end{pmatrix}$$

Ці типи змішаних моделей отримали різні назви в різних дисциплінах. Вони відомі як динамічні регресійні моделі, моделі панельних даних, поздовжні моделі, моделі передавальних функцій та лінійні системні моделі (припускаючи, що функція f є лінійною).

Пояснювальна модель є корисною, оскільки вона включає інформацію про інші змінні, а не лише історичні значення змінної, що прогнозується. Однак є кілька причин, чому прогнозист може обрати модель часового ряду, а не пояснювальну чи змішану модель. По-перше, система може бути незрозумілою, і навіть якщо вона зрозуміла, може бути надзвичайно важко

виміряти взаємозв'язки, які, як передбачається, керують її поведінкою. По-друге, необхідно знати або передбачити майбутні значення різних предикторів, щоб мати можливість прогнозувати змінну, яка нас цікавить, а це може бути надто складно. По-третє, головне завдання може полягати лише в тому, щоб передбачити, що станеться, а не в тому, щоб знати, чому це станеться. Нарешті, модель часового ряду може дати більш точні прогнози, ніж пояснювальна або змішана модель [8].

Вибір моделі для прогнозування залежить від наявних ресурсів і даних, точності конкуруючих моделей, а також від того, як саме буде використовуватися модель прогнозування.

3.2. Глибинне навчання для прогнозування часових рядів

Використання глибинного навчання для прогнозування часових рядів долає недоліки підходів традиційного машинного навчання за допомогою багатьох підходів. Однією з ключових переваг є те, що моделі глибокого навчання можуть врахувати складні нелінійні зв'язки між змінними, тоді як класичні методи зазвичай моделюють лінійні залежності. Моделі глибинного навчання також можуть автоматично витягувати відповідні ознаки з необроблених даних часових рядів, зменшуючи потребу в ручній інженерії ознак. Крім того, моделі глибокого навчання часто більш стійкі до зашумлених даних і викидів, оскільки вони можуть навчитися виявляти закономірності та ігнорувати нерелевантну інформацію [7].

3.2.1. Рекурентні нейронні мережі (RNN)

Рекурентні нейронні мережі - це мережі нейроподібних вузлів, організованих у послідовні шари, з архітектурою, подібною до архітектури стандартних нейронних мереж. Фактично, як і в стандартних нейронних мережах, нейрони поділяються на вхідний шар, приховані шари та вихідний шар. Кожен зв'язок між нейронами має відповідну навчальну вагу [9].

Різниця полягає в тому, що в RNN кожному нейрону присвоюється фіксований часовий крок. Нейрони в прихованому шарі також пересилають дані в залежному від часу напрямку, тобто кожен з них повністю пов'язаний тільки з нейронами в прихованому шарі з тим же призначеним часовим кроком, і з'єднаний одностороннім зв'язком з кожним нейроном, призначеним на наступний часовий крок. Вхідні та вихідні нейрони з'єднані лише з нейронами прихованих шарів з однаковим кроком за часом.

Оскільки вихід прихованого шару одного часового кроку є частиною входу наступного часового кроку, активація нейронів обчислюється в часовому порядку: на кожному часовому кроці обчислюється активація лише тих нейронів, які призначені для цього кроку.

У RNN вхідний вектор в момент часу t пов'язаний з нейронами прихованого шару в момент часу t ваговою матрицею U, нейрони прихованого шару пов'язані з нейронами в моменти часу t-1 і t+1 ваговою матрицею W, а нейрони прихованого шару пов'язані з вихідним вектором в момент часу t ваговою матрицею V; всі вагові матриці є постійними для кожного часового кроку, тобто, використовуються під час всіх епох тренування (рисунок 3.1).

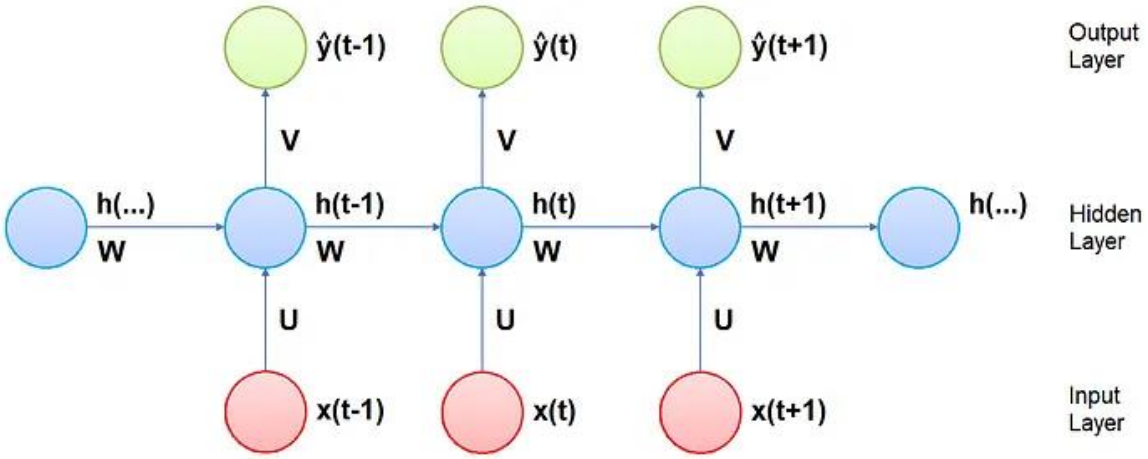

Рисунок 3.1 – Візуалізація роботи моделі RNN

Вектор x(t) є вхідними даними мережі на часовому кроці t.

Вектор h(t) є прихованим станом у момент часу t і є своєрідною пам'яттю мережі. Він обчислюється на основі поточного вхідного значення та прихованого стану на попередньому кроці:

$$h(t) = \tanh\left(Wh(t-1) + Ux(t)\right.$$

Вектор $\hat{y}(t)$ є виходом мережі в момент часу t:

$$\hat{y}(t) = softmax(V_s(t))$$

Метою процесу навчання є пошук найкращих вагових матриць U, V та W. Тобто таких, які дають найкраще передбачення y(t) реального значення y(t).

Для того, щоб контролювати якість передбачення ми визначаємо цільову функцію, яка називається функцією втрат і позначається J. Вона кількісно визначає відстань між реальним і передбаченим значеннями на загальній навчальній множині. Вона задається формулою:

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m}\sum_{t=1}^{N_i} L\big(\hat{y}(t), y(t)\big), \text{ де}$$

m - розмір навчальної вибірки;

θ - вектор параметрів моделі.

Функція витрат L оцінює відстань між реальним та прогнозованим значеннями на одному часовому кроці. Є велика кількість загальноприйнятих функцій втрат, що можуть бути корисні в тих чи інших випадках. Наприклад, MAE (Mean Absolute Error) визначається як середнє значення абсолютної різниці між прогнозованими та фактичними значеннями:

$$MAE = \frac{1}{N}\sum_{i=1}^{N}(\hat{y}_i - y_i)$$

MAE показує нам, яку середню похибку слід очікувати від прогнозу. MAE = 0 означає, що очікувані значення є правильними, а статистика помилок подається у вихідних одиницях прогнозованих значень.

Однак, оскільки MAE не показує пропорційний масштаб помилки, може бути важко розрізнити значні та незначні помилки.

Щоб вирішити цю проблему часто застосовують MAPE (Mean Absolute Percentage Error). Ця функція дуже схожа на MAE. Проте, в даному випадку, береться модуль різниці між прогнозованим і фактичним значеннями і ділиться на фактичне значення.

$$MAPE = \frac{1}{N}\sum_{i=1}^{N}\frac{|\widehat{y_i} - y_i|}{y_i}$$

MAPE краще працює з даними, які не містять нулів та екстремальних значень через наявність дії ділення. Проте, MAPE, як і MAE, недооцінює вплив великих, але рідкісних помилок, спричинених екстремальними значеннями.

Для вирішення цієї проблеми можна використати середньоквадратичну похибку. MSE (Mean Squared Error) визначається як середнє квадратів помилок.

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(\hat{y}_i - y_i)^2$$

Ця функція втрат також набагато сильніше карає великі помилки завдяки наявності операції піднесення до квадрату.

Будь-яка із цих функцій втрат J мінімізується за допомогою двох основних кроків: прямого та зворотного розповсюдження в часі. Ці кроки повторюються багато разів, а ітерація називається епохою.

Під час прямого поширення дані проходять через мережу з фіксованими параметрами U, W і V, і в кожен момент часу t ми обчислюємо $\hat{y}(t)$,

використовуючи попередньо визначені формули. В кінці кожного проходу обчислюється функція втрат. Після цього починається зворотне поширення в часі - градієнти функції витрат обчислюються відносно різних параметрів. Потім застосовується алгоритм градієнтного спуску для їх оновлення. Градієнти на кожному виході залежать як від елементів того самого часового кроку, так і від стану пам'яті на попередньому часовому кроці. Нас цікавить, як саме можна змінити матриці параметрів так, щоб прогноз став більш точним.

Переваги рекурентних нейронних мереж полягають в тому, що RNN вирішують багато проблем традиційних моделей машинного навчання для прогнозування часових рядів, а саме:

- На продуктивність RNN несуттєво впливають пропущені значення за умови їх правильної обробки;
- RNN можуть знаходити складні закономірності у вхідних часових рядах;
- RNN можуть прогнозувати на кілька кроків вперед;
- RNN можуть моделювати послідовність даних таким чином, що кожна наступна послідовність може враховувати залежності і патерни попередніх.

До недоліків рекурентних нейронних мереж можна віднести наступне:

- при навчанні на довгих часових рядах RNN зазвичай страждають від проблеми зникаючого градієнта або вибухаючого градієнта, що означає, що параметри в прихованих шарах або майже не змінюються, або призводять до числової нестабільності і хаотичної поведінки;

- RNN страждають від недостатньо довготривалої пам'яті - вони не здатні враховувати багато елементів минулого при прогнозуванні майбутнього, зокрема тому, що не вміють виділяти корисні спостереження;
- навчання рекурентної нейронної мережі важко розпаралелити, на відміну від класичних методів машинного навчання, а також воно вимагає значних обчислювальних витрат.

Враховуючи ці недоліки, були розроблені різні розширення RNN, що дозволяють вирішити частину проблем та збільшити “пам’ять” мережі: двонаправлені нейронні мережі, LSTM, GRU, механізм уваги. Розширення “пам'яті” може мати вирішальне значення в певних галузях, таких як фінанси, де важливо запам'ятати якомога більше історії, щоб передбачити наступні кроки.

3.2.2. Довга короткочасна пам'ять (LSTM)

Мережі з довгою короткочасною пам'яттю (LSTM) були розроблені для подолання проблеми зникаючого градієнта в стандартних RNN шляхом покращення градієнтного потоку всередині мережі [9]. Це досягається за допомогою LSTM замість прихованого шару. Як показано на рисунку нижче, блок LSTM складається з

- стану комірки, що несе інформацію вздовж всієї послідовності і представляє пам'ять мережі;
- вентиля забування, який вирішує, що важливо зберегти з попередніх часових кроків, а що забути;

- вхідного вентиля, який вирішує, яку інформацію потрібно додати з поточного кроку;
- вихідного вентиля, який визначає значення виходу на поточному кроці.

Подібно до RNN, вхідний вектор в момент часу t пов'язаний з коміркою LSTM в момент часу t ваговою матрицею U. Комірка LSTM пов'язана з комірками LSTM в моменти часу t-1 і t+1 ваговою матрицею W, а комірка LSTM пов'язана з вихідним вектором в момент часу t ваговою матрицею V. Матриці W і U розділені на підматриці (Wf, Wi, Wg, Wo; Uf, Ui, Ug, Uo), які підключені до різних елементів комірки LSTM (рисунок 3.2). Всі вагові матриці є спільними в часі.

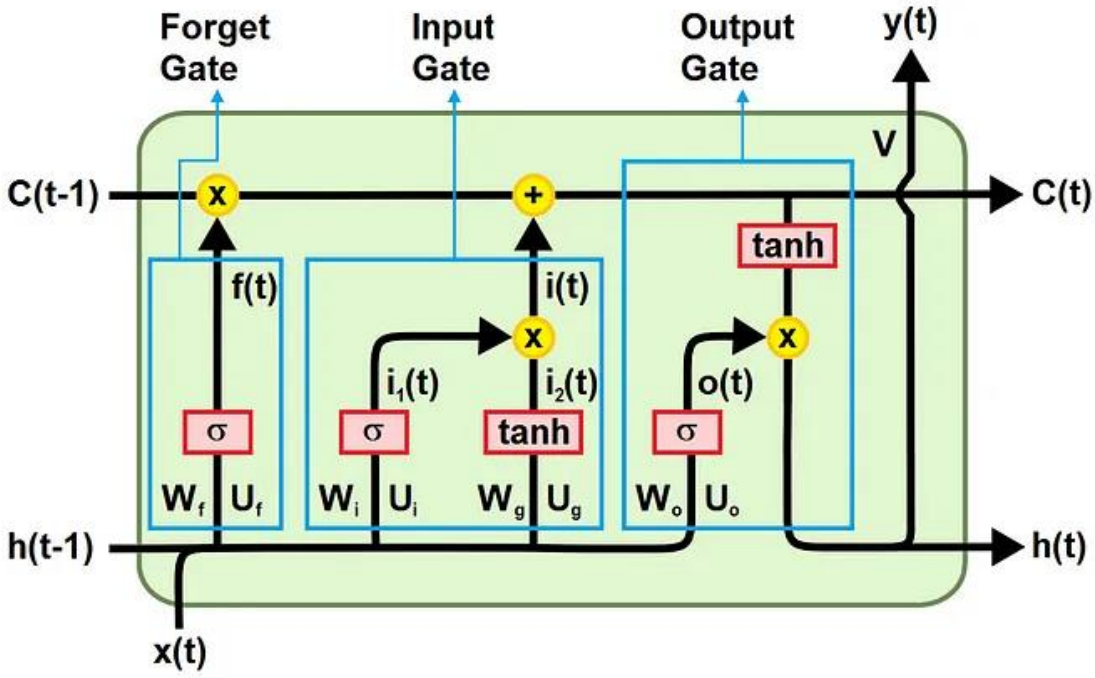

Рисунок 3.2 – Комірка LSTM

Стан комірки передає відповідну інформацію під час обробки, так що інформація з попередніх часових кроків також надходить на кожному часовому кроці, зменшуючи ефект короткої пам'яті. Під час навчання на всіх часових кроках вентилі дізнаються, яку інформацію важливо зберегти, а яку забути, і додають її до стану комірки або видаляють з неї.

Таким чином LSTM дозволяє відновлювати дані, передані в пам'яті, вирішуючи проблему зникаючого градієнта. LSTM корисні для класифікації, обробки та прогнозування часових рядів з часовими лагами невідомої тривалості [9].

Перший вентиль - це вентиль забуття. Цей вентиль вирішує, яку інформацію слід видалити, а яку зберегти. Інформація з попереднього прихованого стану та інформація з поточного входу пропускається через сигмоїдну функцію. Вихід, близький до 0, означає, що інформація може бути забута, тоді як вихід, близький до 1, означає, що інформація повинна бути збережена.

$$f(t) = \sigma(x(t)U_f + h(t-1)W_f)$$

Другий вентиль - це вхідний вентиль. Він використовується для оновлення стану комірки. Спочатку попередній прихований стан і поточний вхід подаються на вхід сигмоїдної функції (чим ближче вихід до 1, тим важливіша інформація). Прихований стан і поточні вхідні дані також передаються на тангенціальну функцію для переведення значень в шкалу між -1 і 1, щоб покращити налаштування мережі. Потім виходи тангенса і сигмоїди перемножуються елемент за елементом. Результат потрапляє в сигмоїдну функцію, що вирішує, яку вхідну інформацію важливо зберегти.

$$i_1(t) = \sigma(x(t)U_i + h(t-1)W_i)$$

$$i_2(t) = \tanh\left(x(t)U_g + h(t-1)W_g\right)$$

$$i(t) = i_1(t) * i_2(t)$$

Після активації вхідного вентилю можна обчислити стан комірки. Спочатку стан комірки на попередньому часовому кроці поелементно множиться на вихід вентиля забування. Це дає можливість ігнорувати значення у стані комірки, коли вони помножені на значення, близькі до 0. Потім вихід вхідного вентиля поелементно додається до стану комірки. На виході отримуємо новий стан комірки.

$$C(t) = \sigma(f(t) * C(t-1) + i(t))$$

Третій і останній вентиль - це вихідний вентиль, який визначає значення наступного прихованого стану, що містить інформацію про попередні входи. Спочатку попередній прихований стан і поточний вхід підсумовуються і передаються сигмоїдній функції. Потім новий стан комірки передається до функції гіперболічного тангенса. В кінці вихід гіперболічного тангенса і вихід сигмоїди перемножуються, щоб вирішити, яку інформацію повинен містити прихований стан. Результатом є новий прихований стан. Новий стан комірки і новий прихований стан переносяться на наступний часовий крок.

$$o(t) = \sigma(x(t)U_o + h(t-1)W_o)$$

$$h(t) = \tanh(C_t) * o(t)$$

3.2.3. Gated Recurrent Unit (GRU)

GRU - це нове покоління рекурентних нейронних мереж. Підхід дуже схожий на LSTM. Для вирішення проблеми зникаючого градієнта стандартного RNN, GRU використовує вентиль оновлення та вентиль скидання. Ці два вентилі вирішують, яку інформацію слід передавати на вихід. Ці два вентилі можна навчити зберігати інформацію з багатьох часових кроків до поточного часового кроку, не розмиваючи її в часі, або видаляти інформацію, яка не має відношення до прогнозу [9]. Як і LSTM, GRU може працювати зі складними часовими залежностями.

Як показано на рисунку 3.3, блок GRU складається з

- вентиля скидання, що вирішує, яку частину інформації з попередніх часових кроків можна забути;
- вентиля оновлення, який вирішує, яку частину інформації з попередніх часових кроків необхідно зберегти;
- пам'яті, яка зберігає інформацію на протязі обробки всіх послідовностей і являє собою пам'ять мережі.

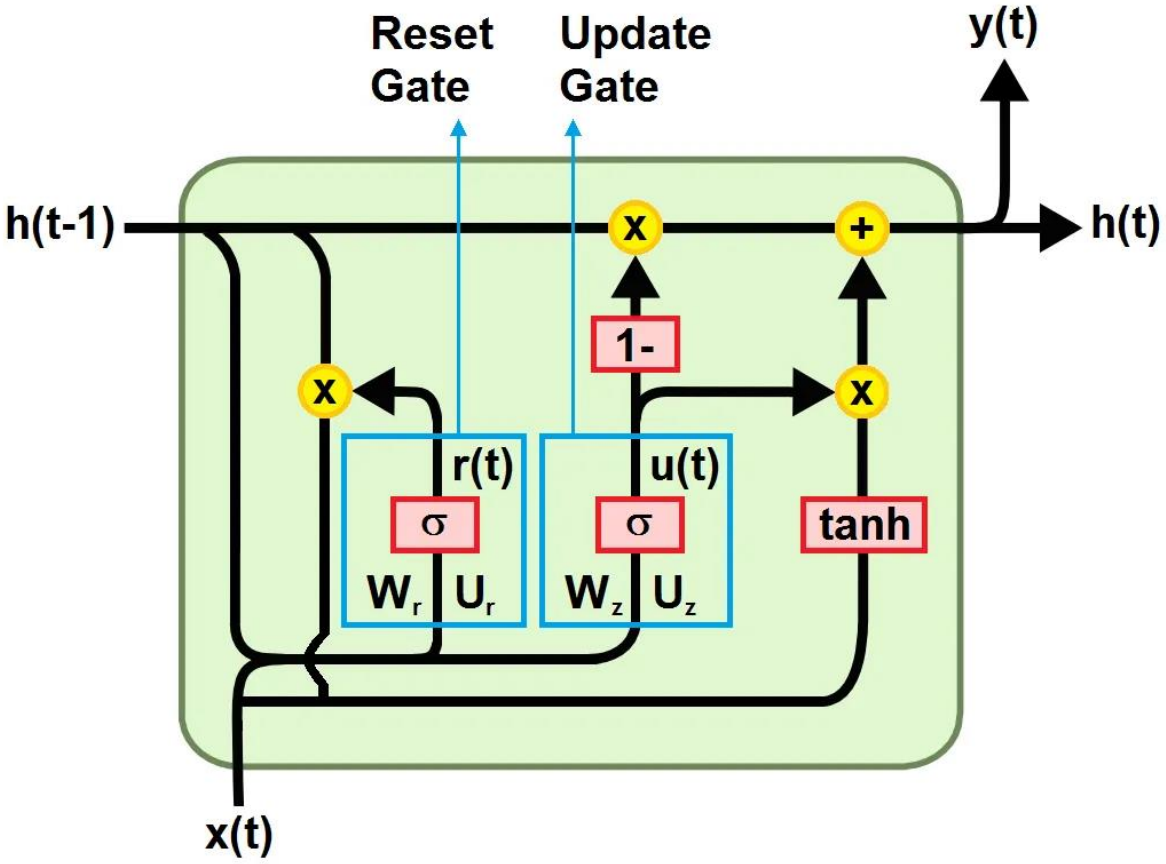

Рисунок 3.3 – Комірка GRU

Перший вентиль - це вентиль скидання. Він визначає, як поєднати новий вхід з попередньою пам'яттю, вирішуючи, скільки інформації з попередніх часових кроків можна забути. Спочатку виконується зважена сума між входом x(t) і пам'яттю h(t-1), яка містить інформацію за попередні t-1 кроків. Потім застосовується сигмоїдна функція активації, щоб перевести результат в шкалу між 0 та 1.

$$r(t) = \sigma(x(t)U_r + h(t-1)W_r)$$

Другий вентиль - це вентиль оновлення. Він допомагає моделі визначити, скільки інформації з попередніх часових кроків потрібно передати в майбутнє. Це дуже важливо, оскільки дає моделі можливість вирішити, яку частину інформації зберегти з минулого. Таким чином ми усуваємо ризик проблеми зникнення градієнта. Формула для його розрахунку аналогічна формулі для вентиля скидання, але різниця полягає у вагах та використанні вентиля (буде пояснено під час розрахунку пам'яті).

$$z(t) = \sigma(x(t)U_z + h(t-1)W_z)$$

Поточну пам’ять використовує вентиль скидання для зберігання релевантної інформації з минулого. Для її отримання спочатку обчислюється поелементне множення між виходом вентилю скидання r(t) та кінцевою пам'яттю на попередньому часовому кроці h(t-1), потім виконується зважена сума між результатом та входом x(t). Нарешті, застосовується нелінійна функція активації tanh.

$$\bar{h}(t) = \tanh\left(x(t)U_h + \big(r(t) * h(t-1)\big)W_h\right)$$

На останньому кроці мережа повинна обчислити h(t), тобто вектор, який містить інформацію для поточної одиниці і передає її на наступний часовий крок. Він визначає, що взяти з поточного вмісту пам'яті $\bar{h}(t)$, а що з попередніх кроків h(t-1). Він обчислюється шляхом поелементного множення між вентилем оновлення z(t) і $\bar{h}(t)$, а також між (1-z(t)) і h(t-1), і, нарешті, виконанням зваженої суми між цими двома результатами.

$$h(t) = \big(1 - z(t)\big) * h(t-1) + z(t) * \bar{h}(t)$$

3.2.4. Модель кодера-декодера

У RNN, LSTM, GRU кожному входу відповідає вихід на одному часовому кроці. Однак у багатьох реальних випадках ми хочемо передбачити вихідну послідовність за вхідною послідовністю різної довжини, без відповідності між кожним входом і кожним виходом. Така ситуація називається моделлю відображення послідовності в послідовність (seq2seq) і лежить в основі багатьох поширених додатків, таких як, наприклад, мовні перекладачі, пристрої з підтримкою голосу та онлайн чат-боти.

Модель кодера-декодера для рекурентних нейронних мереж була створена для того, щоб зробити можливим відображення послідовності в послідовність. Кодер-декодер приймає послідовність на вхід і генерує найбільш ймовірну наступну послідовність на виході. Як показано на рисунку 3.4, модель складається з двох підмоделей:

- кодер, який відповідає за проходження вхідних часових кроків і кодування всієї послідовності у вектор фіксованої довжини, який називається вектором контексту;
- декодер, який відповідає за формування вихідних часових кроків, враховуючи вектор контексту.

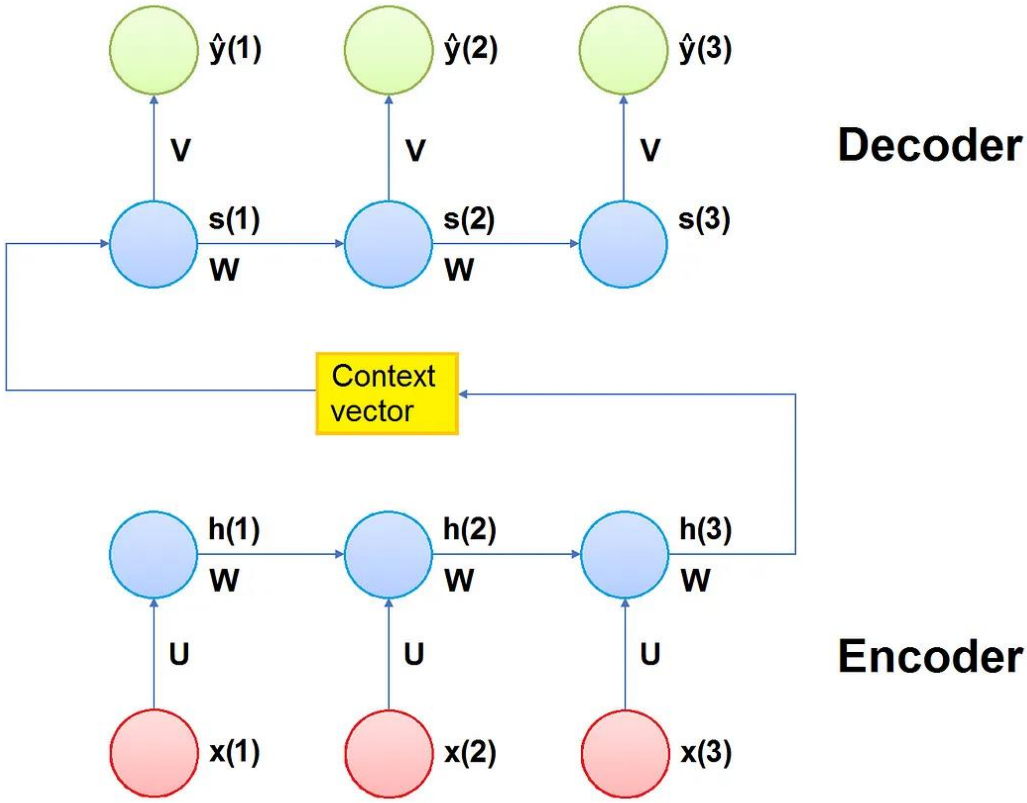

Рисунок 3.4 – Модель кодер-декодер

Кодер - це стек з декількох рекурентних блоків, які можуть бути простими RNN, LSTM комірками або GRU комірками. Кожен блок приймає один елемент вхідної послідовності, збирає інформацію з цього елемента і передає її далі.

Прихований вектор стану h(t) обчислюється за допомогою функції обраного рекурентного елемента. Функція застосовується з відповідними вагами до попереднього прихованого стану h(t-1) та вхідного вектора x(t):

$$h(t) = f(Wh(t-1) + Ux(t))$$

Кінцевий вектор прихованого стану h(t) містить всю закодовану інформацію з попередніх прихованих представлень та попередніх входів.

Вектор контексту - це кінцевий прихований стан, отриманий з кодуючої частини моделі, і являє собою початковий прихований стан для декодера. Він інкапсулює інформацію для всіх вхідних елементів, щоб допомогти декодеру робити точні прогнози.

Декодер складається зі стеку з декількох рекурентних блоків. Кожен рекурентний елемент приймає прихований стан s(t-1) від попереднього елемента і видає $\hat{y}(t)$, а також свій власний прихований стан s(t).

Прихований стан s(t) обчислюється відповідно до функції обраного рекурентного блоку:

$$s(t) = f(Wh(t-1))$$

Вихід $\hat{y}(t)$ обчислюється за допомогою функції softmax з використанням прихованого стану на поточному кроці часу s(t) разом з відповідною вагою, щоб створити вектор ймовірності:

$$\hat{y}(t) = softmax(Vs(t))$$

Особливість цієї моделі полягає в тому, що вона може зіставляти послідовності різної довжини одна з одною, оскільки входи і виходи не є корельованими і можуть мати різну довжину. Це відкриває цілу низку нових задач, які тепер можна розв'язувати за допомогою такої архітектури.

Проте, хоча ця техніка і працює добре для невеликих послідовностей, але при збільшенні довжини послідовності стає важко сформувати один вектор,

який буде добре описувати всю довгу послідовність. І тоді модель часто забуває попередні частини вхідної послідовності при обробці останніх частин. Багато експериментів показують, що продуктивність цієї моделі зменшується зі збільшенням розміру послідовності.

3.2.5. Оцінка невизначеності моделі

Іще одним популярним методом, що застосовується в задачах прогнозування часових рядів, є оцінка невизначеності моделі. Оскільки, на основі прогнозів, які дала модель, можуть прийматися важливі рішення, було б корисно знати, наскільки модель впевнена в своєму прогнозові. Так, наприклад, передбачити попит на популярні товари, які купуються щодня зазвичай легше, ніж на ті, що продаються рідко і без жодних патернів. Відповідно прогноз буде значно сильніше варіюватися від зовнішніх чинників і буде менш стабільним для другої групи товарів.

Щоб оцінити невизначеність моделі можна використати методи марковських ланцюгів Монте-Карло (MCMC). Методи MCMC дозволяють робити вибірку з апостеріорного розподілу вагових коефіцієнтів моделі. Такі вибірки можна використовувати для оцінки невизначеності в прогнозах моделі. Однак методи MCMC можуть бути обчислювально дорогими, що робить їх непрактичними для великих наборів даних. Для вирішення цієї проблеми використовують MCMC dropout як обчислювально ефективну альтернативу методам MCMC [10]. MCMC dropout - це варіант традиційної техніки dropout, що використовується в нейронних мережах, де під час навчання підмножина нейронів випадковим чином відсіюється. У методі MCMC dropout процес відсіву повторюється багато разів, щоб згенерувати

набір вагових вибірок з апостеріорного розподілу. Ці вагові вибірки можуть бути використані для оцінки невизначеності в прогнозах моделі, які можуть бути корисними в різних сферах застосування, включаючи прийняття рішень, оцінку ризиків і вибір моделі.

3.3. Опис модифікованих способів і засобів для розподіленої обробки часових рядів

3.3.1. Підготовка даних

Перед тим як подавати дані в будь-яку модель глибинного навчання їх потрібно певним чином підготувати. Під час цього процесу забезпечується цілісність даних шляхом виявлення та вирішення таких проблем, як пропущені значення, викиди або помилки вимірювання. Також забезпечується узгодженість даних шляхом повторної вибірки або інтерполяції даних для отримання послідовних часових інтервалів, що є важливим для багатьох методів аналізу часових рядів і моделей. Попередня обробка може включати зменшення шуму в даних або фільтрування нерелевантної інформацію. Крім того, може бути створено нові ознаки, які будуть корисні для моделі та полегшать її навчання.

Так, дуже важливим методом є виконання обчислень за допомогою рухомих або ковзних вікон. Цей підхід передбачає створення вікна певного розміру, яке переміщується або "прокручується" крізь дані, дозволяючи виконувати обчислення над даними в межах вікна. Цей метод особливо корисний для підготовки даних для мереж, що працюють з послідовностями

даних (RNN, LSTM, GRU), які зазвичай використовуються для аналізу часових рядів.

Одним із поширених підходів є використання фіксованого вікна, коли до даних часового ряду застосовується вікно фіксованого розміру. Це може бути корисно при роботі з дуже довгими окремими послідовностями. Вікно переміщується по даних, і обчислення виконуються в межах вікна, щоб виділити релевантні ознаки або нарізати послідовність на менші семпли потрібного розміру.

Інший підхід полягає у використанні комбінації згорткових нейронних мереж CNN і RNN-мереж. CNN використовується для інтерпретації кожної наступної послідовності часових кроків, створюючи часовий ряд інтерпретацій послідовностей, що стануть вхідними даними для RNN-моделі. Це може допомогти виявити локальні закономірності або особливості в даних, які потім можуть бути оброблені для аналізу або прогнозування на більш високому рівні.

Важливо ретельно розробляти та впроваджувати розрахунки з ковзним або рухомим вікном, оскільки вони можуть суттєво вплинути на результати аналізу часових рядів. Правильний вибір розміру вікна, обробка розривів в ряді та управління вікнами, що перекриваються або не перекриваються, є дуже важливим.

У Python є різні реалізації нарізання часового ряду ковзним вікном. Найбільш простим рішенням є використання бібліотеки NumPy або Pandas.

Виконання ковзних обчислень у Pandas є дійсно простим і зручним, особливо для даних часових рядів. Pandas надає вбудовану функціональність для обробки ковзних вікон, що дозволяє легко обчислювати різні статистичні

дані, застосовувати користувацькі функції та маніпулювати даними в межах ковзного вікна.

З іншого боку, в NumPy утиліта для виконання ковзних обчислень не настільки проста і може вимагати більше ручної реалізації. Однак, при роботі з великими наборами даних або критичними до продуктивності додатками, операції нижнього рівня NumPy та оптимізовані операції з масивами можуть забезпечити більшу продуктивність та ефективність у порівнянні з Pandas. NumPy забезпечує більш тонкий контроль над маніпуляціями з масивами і може бути більш придатним для складних задач обробки та маніпулювання даними. Tensorflow в своїх навчальних матеріалах теж використовує NumPy для нарізання часового ряду [11].

Нарізка часового ряду вікнами, також відома як "ковзне вікно", або "ковзаюче вікно", - це метод, який використовується в аналізі часових рядів для обробки даних у вигляді фрагментів або вікон фіксованого розміру, які послідовно переміщуються над даними часового ряду. Це дозволяє аналізувати дані протягом певних періодів часу і може бути корисним для таких завдань, як прогнозування, аналіз трендів і вилучення особливостей.

Зазвичай розбиття часового ряду на вікна виконується наступним чином:

1. Першим кроком у процесі розбиття на вікна є визначення бажаного розміру вікна або кількості точок даних, які будуть включені в кожне вікно. Це може ґрунтуватися на конкретних вимогах аналізу або моделювання і може змінюватися залежно від характеристик даних часового ряду.

2. Потім потрібно вказати початкову точку або початкову позицію вікна в даних часового ряду. Це може бути перша точка даних або будь-який інший допустимий показник у часовому ряді.
3. Після того, як визначено розмір вікна та початкову точку, ми отримуємо дані з вікна - послідовні точки даних у часовому ряді, які підпадають під розмір вікна, починаючи з вказаної початкової точки.
4. Після того, як вікно з даними витягнуто, ми можемо обробити його потрібним нам чином та подати на вхід моделі.
5. Після завершення аналізу або моделювання, необхідно перемістити вікно до наступної позиції в часовому ряді. Зазвичай це робиться шляхом просування вікна на фіксований розмір кроку, який часто називають "зсувом". Зсув визначає, наскільки вікно просувається вперед у кожній ітерації.
6. Кроки з 3 по 5 повторюються ітеративно, поки вікно не переміститься по всьому часовому ряду даних. Це дозволяє виконувати аналіз або моделювання на декількох вікнах, що перекриваються, захоплюючи різні сегменти даних часового ряду.

Рисунок 3.5 ілюструє етапи нарізання часового ряду ковзним вікном.

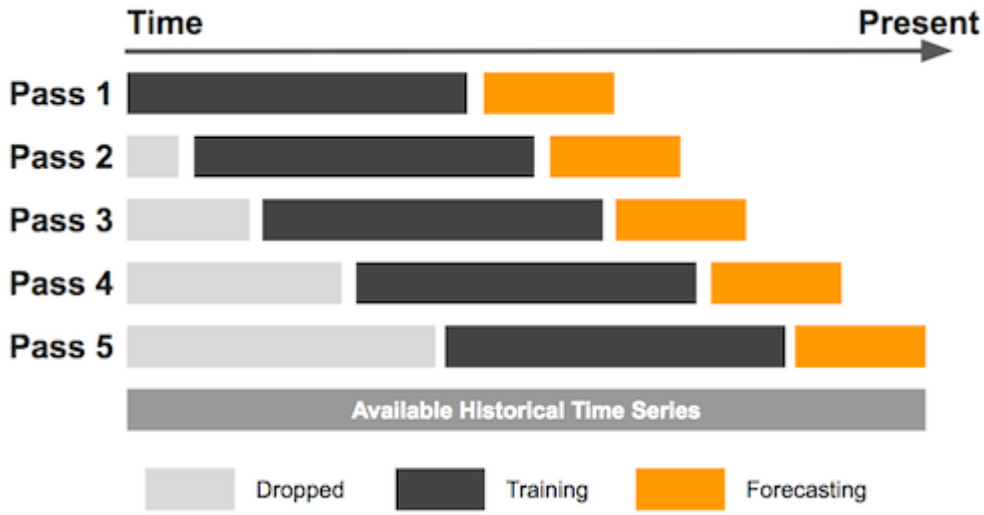

Рисунок 3.5 – Нарізання часового ряду ковзним вікном

Цю операцію можна реалізувати ітеративно або ж функціонал stride_tricks з бібліотеки NumPy. Основна ідея stride_tricks полягає в тому, щоб маніпулювати базовою структурою пам'яті масиву, вказуючи нові зсуви для масиву. Зсув - це кількість байт, на яку потрібно зробити крок у кожному вимірі масиву, щоб отримати доступ до наступних елементів. Так, наприклад, для звичайного двовимірного масиву розміру N на M з типом даних int зсуви будуть мати значення 1*sizeof(int) і M*sizeof(int). Тобто, щоб перейти від першого елементу до другого, необхідно пересунутися в пам’яті на розмір одного елемента. А для переходу від елемента в першому рядку до елемента в наступному рядку треба зробити зсув на розмір M елементів. Маніпулюючи зсувами масиву, ми можемо створювати нові представлення масиву, які можна використовувати для широкого спектру розширених операцій. Цей функціонал рекомендується використовувати з великою обережністю, адже у випадку використання ми маніпулюємо внутрішньою структурою даних

масиву і, якщо це зроблено неправильно, елементи масиву можуть вказувати на неправильну область пам'яті, що може призвести до спотворення результатів або аварійного завершення роботи програми. Проте, даний функціонал дуже корисний для розбиття масиву на вікна. Псевдо-код, що описує логіку функції для нарізання даних (в реальності не містить циклів):

```
function sliding_window_view(data, window_size, stride):
  # Define a helper function to apply the sliding window view to a single column
  function sliding_window_view_column(x):
    # Calculate the number of windows for the given window size and stride
    num_windows = (len(x) - window_size) // stride + 1

    # Calculate the strides for the sliding window view
    strides = (stride * len(x), len(x))

    # Apply the sliding window view using stride tricks
    result = array(shape=(num_windows, window_size))
    for i in range(num_windows):
      for j in range(window_size):
        result[i, j] = x.at(i * strides[0], j * strides[1])
    return result

  # Apply the sliding window view to each column of the dataset
  for column in data:
    windowed_data[column] = sliding_window_view_column(data[column])

  # Return the resulting windowed data
return windowed_data
```

3.3.2. Розподілене тренування моделі

Для тренування моделі використаємо синхронний підхід до розподіленого тренування – DDP (Distributed Data Parallel). Це рішення обґрунтовано тим, що кластер для тестування буде складатися з машин одного типу, з однаковими обчислювальними потужностями. Як результат час обробки даних однакового розміру буде близький, і ситуації, коли багато працівників чекатимуть на завершення роботи одного виникати не має.

При використанні цього підходу модель і дані розподіляються між декількома графічними процесорами або машинами. Потім кожен працівник

обчислює прямий прохід на підмножині даних і надсилає результати іншим працівникам. На наступному кроці результати усереднюються між усіма працівниками, щоб отримати глобальний градієнт, який використовується для оновлення ваг моделі. Цей процес повторюється протягом декількох ітерацій, причому кожен працівник обробляє окрему підгрупу даних.

Наступні кроки детально описують, як працює цей підхід:

- Ініціалізація. Ініціалізується модель, яку потрібно навчити, а також задається кількість графічних процесорів або машин, які потрібно використати для навчання. Потім кожному працівнику присвоюється унікальний ідентифікатор.
- Розбиття даних на розділи: Набір даних розбивається на підмножини, щоб кожен працівник міг обробляти свою підмножину даних. Це дозволяє виконувати паралельну обробку даних кількома працівниками.
- Реплікація моделі: Модель реплікується між усіма працівниками, при цьому кожен працівник має копію параметрів моделі. Кожна копія моделі ініціалізується однаковими вагами.
- Прохід вперед: Кожен працівник обчислює прохід вперед на своїй підмножині даних, використовуючи свою копію моделі. Результати зберігаються локально на робочому місці.
- Прохід назад: Кожен працівник обчислює градієнти параметрів моделі відносно функції втрат, використовуючи локальні дані. Потім градієнти надсилаються всім іншим працівникам.
- Усереднення градієнтів: Градієнти від усіх робітників усереднюються для отримання глобального градієнта. Цей

глобальний градієнт потім використовується для оновлення параметрів моделі для кожного робітника.

- Синхронізація: Всі робітники синхронізують свої параметри моделі один з одним після кожної ітерації. Це гарантує, що кожен робітник має останню версію параметрів моделі.
- Повторення: кроки 4-7 повторюються для декількох ітерацій, при цьому кожен працівник обробляє окрему підмножину даних.

Розподіляючи дані та обчислення між кількома працівниками, можливо ефективно паралельно обробляти великі набори даних та використовувати складні моделі. Це може значно скоротити час навчання. Однак реалізація розподіленого навчання вимагає ретельного підходу до апаратного забезпечення, топології мережі та розподілу даних. Тому для налаштування кластера ми використаємо Ray кластер (рисунок 3.6).

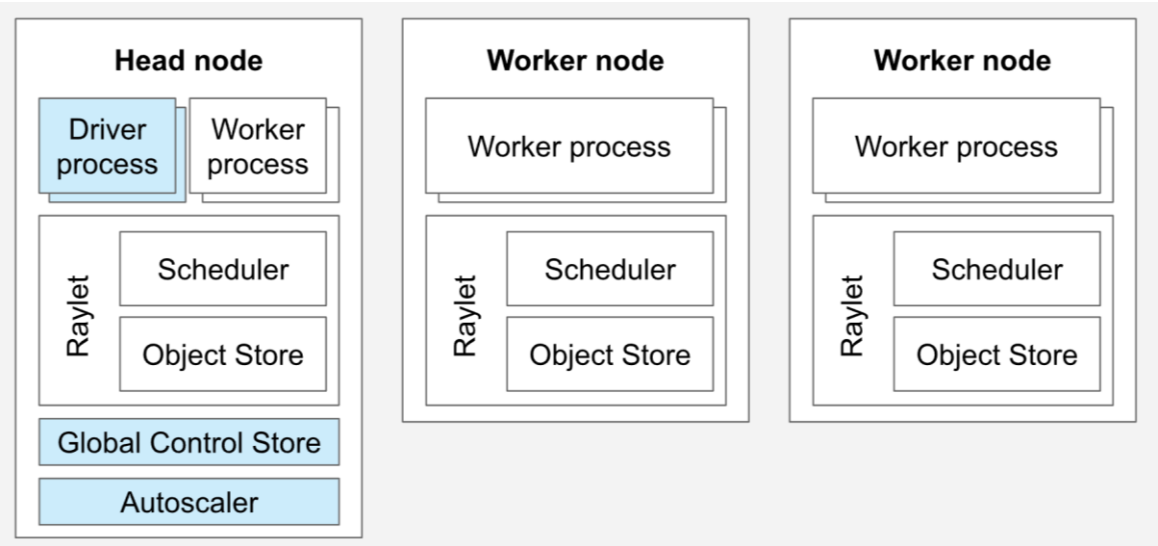

Рисунок 3.6 - Візуалізація компонентів Ray кластера з одним головним і двома робочими вузлами.

Висновки до розділу 3

В розділі 3 на основі аналізу основних способів і засобів обробки запропоновано модифікації алгоритмів підготовки даних та тренування моделей шляхом розпроділених обчислень

Запропоновано виконання попередньої підготовки даних часових рядів для тренування моделей за допомогою ефективного алгоритму створення проекцій масиву через зсуви. Такий підхід дозволяє обробляти великі масиви даних швидше, ніж ітеративний підхід. Для реалізації алгоритму запропоновано використання можливостей бібліотеки NumPy, яка надає функціонал для створення проекцій багатовимірних масивів. Як результат немає необхідності створювати копії даних в пам’яті, що дозволяє зменшити об’єм необхідної для обчислень пам’яті.

Для забезпечення можливості здійснення розподіленого навчання запропоновано використання синхронного навчання моделі. Такий підхід розбиває дані на менші частини, які обробляються паралельно. І тому має низку переваг, зокрема скорочення часу навчання, покращення масштабованості та більш ефективне використання обчислювальних ресурсів.

Розроблено програмну реалізацію запропонованих алгоритмів для тестування їх ефективності.

4. ТЕСТУВАННЯ МОДИФІКОВАНИХ АЛГОРИТМІВ ТА АНАЛІЗ ОТРИМАНИХ РЕЗУЛЬТАТІВ

4.1. Обхід даних вікном

Для нарізання даних часових рядів на послідовності для навчання було реалізовано логіку, що працює завдяки створенню нового представлення масиву. Для порівняння швидкодії реалізованого функціоналу із звичайним ітеративним підходом обидві реалізації було запущено на послідовностях даних різної довжини. В даному випадку тестування відбувалося на одному ядрі процесора AMD Ryzen 5 5600H з частотою 3,3 ГГц.

З рисунка 4.1 можемо бачити, що хоча обидві реалізації мають лінійну часову складність O(n), ітеративна реалізація набагато більш повільна, ніж реалізація з використанням stride_tricks. Крім того, у випадку обробки двовимірного масиву звичайна реалізація потребуватиме ще одного цикла, що зробить її ще повільнішою. В той же час реалізація з stride_tricks може обробляти двовимірні масиви без застосування додаткових циклів.

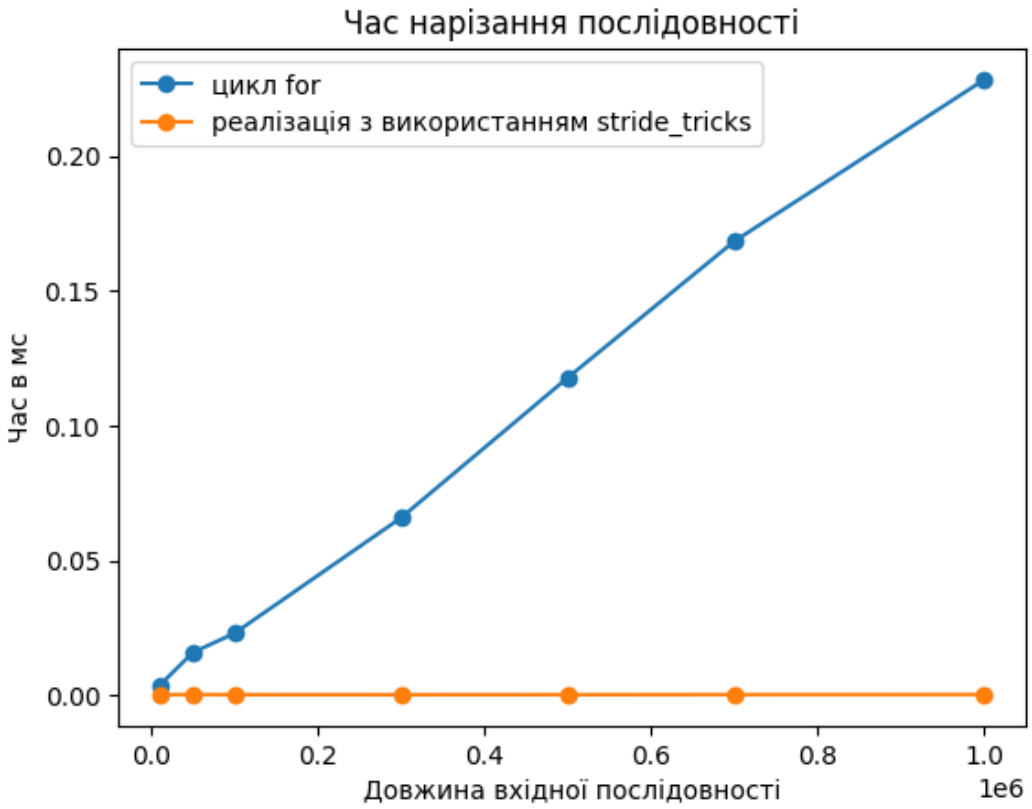

Рисунок 4.1 – Порівняння часу нарізання вхідної послідовності

4.2. Тренування

Моделі, що використовуються для прогнозування часових рядів можуть бути незалежними і навчатися на кожному часовому ряді окремо (класичні підходи). Або ж це може бути одна модель, що вчиться на всьому наборі даних (моделі глибинного навчання). Щоб протестувати обидва варіанти тренування було обрано класичну модель AutoARIMA та модель глибинного навчання Bayesian

4.2.1. Класична модель AutoARIMA

Для тестування можливостей реалізованого підходу до розподіленого навчання використаємо бібліотеку StatsForecast від Nixtla, адже вона має інтеграцію з Ray для розподілених обчислень.

AutoARIMA є однією з найбільш широко використовуваних моделей для прогнозування часових рядів. Вона була розроблена Хайндманом та Хандакаром і з моменту своєї появи стала галузевим стандартом швидкості та точності. ARIMA моделює часовий ряд за допомогою трьох параметрів: порядок авторегресії p, порядок інтегрування d та кількість параметрів ковзного середнього q. AutoARIMA знаходить найкращу модель ARIMA, підбираючи гіперпараметри. Налаштування гіперпараметрів (правильний набір p, d, q) відбувається всередині моделі за допомогою алгоритму Хайндмана і Хандакара, тому користувачеві не потрібно думати про це завдання. Модель фокусується на використанні автокореляцій, різниць та середніх для отримання точних прогнозів і є добре перевіреною моделлю, яка чудово працює для багатьох типів даних часових рядів.

До впровадження AutoARIMA компанією Nixtla, модель була доступна лише у мові R (яка була розроблена Хайндманом). Крім того, існувала версія на Python, pmdarima, заснована на StatsModels. Як показує цей експеримент, pmdarima є набагато повільнішою та менш продуктивною, особливо для часових рядів, що мають сезонність.

StatsForecast компілює AutoARIMA just in time (JIT) за допомогою Numba, що робить алгоритм надзвичайно швидким. За допомогою Ray ми можемо розподілити обчислення між різними ядрами і масштабувати систему горизонтально стільки, скільки нам потрібно.

Щоб створити кластер Ray, необхідно підготувати yaml-файл і хмарну платформу. У цьому конкретному експерименті використовується хмарна платформа Google Cloud. Перед запуском кластера необхідно налаштувати облікові дані Google Cloud. Для цього потрібно створити спеціальний json файл з обліковими даними користувача. Цей файл містить інформацію про проект, в якому був створений, та ключ, що надає можливість автентифікації додатку в хмарних сервісах та API Google. Після того, як файл буде створено, ми можемо вказати шлях до нього як змінну середовища GOOGLE_APPLICATION_CREDENTIALS.

Тепер можемо створювати кластер. Для цього необхідно вказати всі необхідні нам параметри в yaml файлі.

Створимо великий кластер на 30 машин типу n1-standard-8. Кожна машина має 8 vCPU та 30 Гб оперативної пам’яті. Сумарно це дає нам 240 vCPU та 900 Гб оперативної пам’яті розподілених між усіма машинами кластера. Частину конфігураційного файлу наведено нижче.

```
cluster_name: default
max_workers: 30
upscaling_speed: 1.0
docker:
 image: "rayproject/ray-ml:latest-cpu" # You can change this to latest-cpu if you don't need GPU support and want a faster startup
 container_name: "ray_container"
 pull_before_run: True
 run_options:  # Extra options to pass into "docker run"
   - --ulimit nofile=65536:65536
idle_timeout_minutes: 5
provider:
   type: gcp
   region: us-central1
   availability_zone: us-central1-a
   project_id: coherent-flame-379720 # Globally unique project id
auth:
   ssh_user: ubuntu
available_node_types:
   ray_head_default:
      resources: {"CPU": 8}
      node_config:
         machineType: n1-standard-8
         disks:
           - boot: true
             autoDelete: true
             type: PERSISTENT
             initializeParams:
               diskSizeGb: 50
```

```
            # See https://cloud.google.com/compute/docs/images for more images
            sourceImage: projects/deeplearning-platform-release/global/images/family/common-cpu
ray_worker_small:
    min_workers: 30
    max_workers: 30
    resources: {"CPU": 8}
    node_config:
        machineType: n1-standard-8
        disks:
          - boot: true
            autoDelete: true
            type: PERSISTENT
            initializeParams:
              diskSizeGb: 50
              sourceImage: projects/deeplearning-platform-release/global/images/family/common-cpu
        scheduling:
          - preemptible: true
```

Експеримент було проведено наступним чином - модель було навчено на різних наборах часових рядів, використовуючи створений кластер. Для тренування використовуючи різні розміри часових рядів, а саме: 5 тисяч часових рядів, 10 тисяч часових рядів, 50 тисяч, 100 тисяч. Ці набори даних часових рядів генеруються випадковим чином за допомогою функції "generate_series" бібліотеки StatsForecast. Кожен набір даних часових рядів генерується з розміром від 50 до 100 спостережень випадково, а періодичність часових рядів встановлюється як щоденна. Після завершення процесу навчання моделі для всіх згенерованих наборів даних часових рядів, прогнози будуть згенеровані на основі навченої моделі (рисунок 4.2).

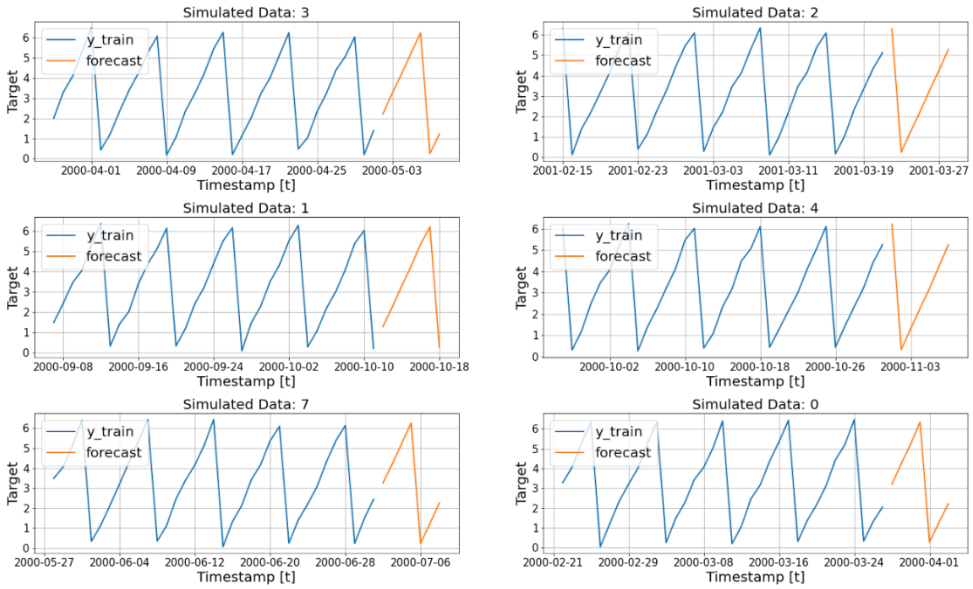

Рисунок 4.2 - Приклади отриманих зразків прогнозів

Результати експерименту наведено нижче. Як показано на рисунку 4.3, використовуючи реалізовані методи для тренування моделі на кластері можна навчити AutoARIMA на 100 тисячах часових рядів менше ніж за годину.

Рисунок 4.3 – Час навчання моделі в залежності від кількості часових рядів

Також було проведено експеримент без використання розподілених обчислень на кластері. Як і очікувалося, час тренування з Ray набагато швидший, ніж без нього (рисунок 4.4).

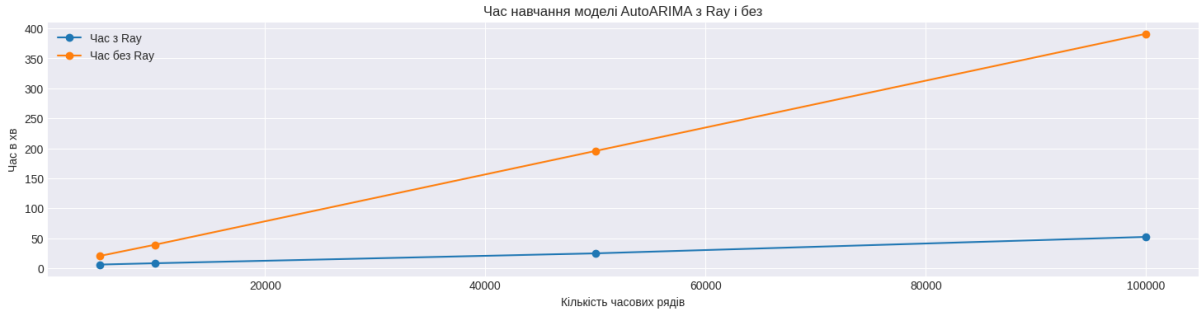

Рисунок 4.4 – Порівняння часу навчання моделі з використанням розробленого алгоритму і без

Слід зазначити, що модель ARIMA навчається незалежно на кожному окремому наборі даних часових рядів. Отже, закономірності та висновки, отримані з одного конкретного часового ряду, жодним чином не використовуються для передбачення або прогнозування інших наборів даних часових рядів. Модель ARIMA навчається окремо для кожного часового ряду, використовуючи його власні унікальні характеристики та історичні дані. Такий підхід гарантує, що процес прогнозування залишається окремим і незалежним для кожного окремого набору даних часових рядів, без включення будь-якої спільної інформації або закономірностей з інших наборів даних. Це дозволяє легко навчити алгоритм паралельно на багатьох часових рядах одночасно.

4.2.2. Модель глибинного навчання байєсівський кодер-декодер

На відміну від моделей ARIMA, моделі глибокого навчання, як правило, використовують всю доступну інформацію і здатні виявляти спільні закономірності та зв'язки між окремими наборами даних часових рядів. Тому замість окремих моделей для кожного часового ряду ми навчаємо одну загальну модель на всіх доступних даних. Отже, для прискорення навчання такого алгоритму глибокого навчання потрібен альтернативний підхід - розподілене навчання.

Для цього експерименту було використано набір даних, що складається з історичних даних про надані послуги за 2022 рік, отриманих від нью-йоркської служби "Жовте таксі". Дані містять поля, що фіксують дати/час посадки та висадки, місця посадки та висадки, відстані поїздок, деталізовані тарифи, типи тарифів, типи оплати та кількість пасажирів, про яку повідомив водій. Всього таблиця містить близько 40 мільйонів рядків, має розмір 544 Мб у форматі Apache Parquet. В таблиці існує 260 унікальних точок старту поїздки. Ми їх вважатимемо за ідентифікатор часового ряду. Таким чином, маємо 260 унікальних часових рядів. Щоб отримати ще більший датасет ми продублювали наявний. Для цього завантажили оригінальний датасет на вебслужбу хостингу файлів Google Cloud Storage. Після цього зробили 50 копій датасету. Для кожної копії вказали нові унікальні точки старту поїздки. Також кожна копія була збережена як окремий файл (Apache Parquet дозволяє зберігати одну таблицю як набір файлів). В результаті ми отримали датасет розміром близько 50 Гб, що містить близько 2 мільярдів записів та 13000 унікальних часових рядів.

Для обробки цього датасету було створено кластер на платформі Google Cloud з використанням 6 машин типу n1-standard-4. Машини серії N1 - це перше покоління універсальних машин Compute Engine, доступних на різних процесорних платформах, включаючи Skylake, Broadwell, Haswell, Sandy Bridge та Ivy Bridge. Конкретний тип обраної машини - n1-standard-4 - має 4 віртуальні процесори (vCPU) та 15 ГБ пам'яті.

Кластер складається з одного головного вузла та п'яти робочих вузлів. Головний вузол відповідає за управління всім кластером, включаючи розподіл і координацію завдань, а також виконання обчислень. З іншого боку, робочі вузли призначені виключно для виконання обчислень.

Загальні ресурси кластеру складають 24 vCPU та приблизно 60 ГБ вільної оперативної пам'яті (рисунок 4.5). Ці ресурси будуть використовуватися для обробки створеного датасету всередині кластера, забезпечуючи ефективне використання наявних обчислювальних потужностей.

Рисунок 4.5 – Панель керування Ray кластером

Для того, щоб обробити дані дистрибутивно використовується Ray Dataset. Він надає API вищого рівня для паралельних обчислень. Датасет розподіляється між вузлами кластера і кожна частина датасету оброблюється окремо.

Для того, щоб прибрати складову тренду з даних використовується груповий нормалізатор PyTorch Forecasting, який застосовує пакетну нормалізацію для кожної групи часових рядів. Дані поділяються на вхідні дані для навчання тривалістю 63 години та дані для прогнозування тривалістю 168 годин, що відповідає тижневому інтервалу. Такий підхід забезпечує збереження часової впорядкованості даних, оскільки дані для навчання/перевірки відбираються з використанням вікон довжиною 63/168 годин. Тобто, модель не має жодної інформації про майбутнє, яке їй доведеться прогнозувати.

В якості моделі використовувалась Байєсівська LSTM з MCMC dropout. Ця модель проводить досить багато обчислень для обробки одного пакету даних. Розмір одного пакету - 256 послідовностей. Тренування моделі впродовж 10 епох зайняло 2 години і 4 хвилини (рисунок 4.6).

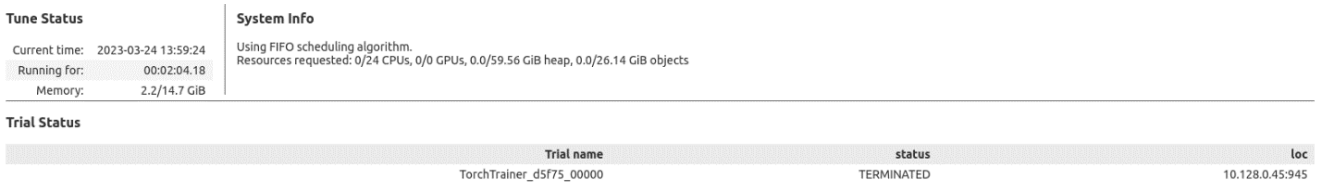

Рисунок 4.6 – Час тренування впродовж 10 епох

Для того щоб провести подібне тренування на одній машині - потрібні досить великі потужності та об’єм оперативної пам’яті. Крім того, якщо

об’єми обчислень зростуть і стануть завеликими, доведеться оновлювати апаратну частину. Використовуючи ж Ray, можна протестувати код на невеликому об’ємі даних локально, після чого запустити тренування на кластері. При цьому не доведеться змінювати код чи логіку програми. Якщо у майбутньому кількість даних зросте - кластер може бути збільшеним, щоб пришвидшити тренування. Зробити це дуже просто - достатньо змінити налаштування кластеру через конфігураційний файл.

Висновки до розділу 4

У даному розділі проведено дослідження ефективності реалізованих алгоритмів. Для цього створено набори даних часових рядів великого обсягу – 50 Гб даних у форматі Apache Parquet та штучно згенерований набір зі ста тисяч часових рядів. Продемонстровано, що реалізовані способи та підходи до обробки великих об’ємів часових рядів та розподіленого тренування моделей дають значний приріст швидкості як у випадку використання моделей ARIMA, так і у випадку використання моделей глибинного навчання. До того ж показано, що алгоритми легко масштабуються. Тому у випадку збільшення об’ємів даних, достатньо збільшити кластер, щоб отримати стабільну швидкість роботи. Така можливість є дуже важливою при застосуванні алгоритмів у реальному середовищі, коли є чіткі обмеження по часу обчислень.

ВИСНОВКИ

Метою магістерської дисертації є підвищення ефективності алгоритмів обробки часових рядів шляхом використання розподілених обчислень на кластері. Розробка нових і вдосконалення існуючих способів і алгоритмів обробки великих об’ємів даних є актуальною для вирішення задач прогнозування часових рядів.

Проведені дослідження сучасних способів і засобів розподіленої обробки часових рядів в кластерних системах показали, що існуючі рішення не є гнучкими та пропонують обмежену кількість алгоритмів для обробки даних.

В роботі запропоновано алгоритми та засоби для розподіленого навчання незалежних моделей та для тренування моделей глибинного навчання, що мають спільні параметри для всіх наявних даних. Реалізовано пришвидшені методи підготовки даних, що можуть бути виконані паралельно на декількох процесорах або декількох машинах кластера.

Розроблені алгоритми та підходи протестовано на великих об’ємах даних. Показано, що вони дають суттєвий приріст швидкодії порівняно із стандартними підходами та значно скорочують час, необхідний для підготовки даних. Це дозволяє швидше навчати моделі машинного навчання.

Проведені дослідження доводять, що запропоновані алгоритми та засоби розподіленої обробки дозволяють суттєво збільшити швидкість обробки великої кількості часових рядів. Вони можуть бути застосовані для рішення багатьох практичних задач в різних галузях: фінансовому прогнозуванні, прогнозуванні користувацького попиту на, прогнозуванні продажів в сфері роздрібної торгівлі тощо. Це допоможе компаніям швидше виявляти та реагувати на зміни в бізнес-середовищі, а отже бути більш ефективними.

Розроблені алгоритми легко інтегруються у вже існуючі системи та засоби, надаючи користувачам зручні і масштабовані інструменти.

Для подальшого вдосконалення доцільно реалізувати алгоритм, що використовує метод паралелізму моделі. Це дозволить навчати дуже великі моделі, поділяючи між працівниками саму модель, а не лише набори даних.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Holden Karau, Boris Lublinsky. Scaling Python with Ray, 2022, с. 45–50.
2. Philipp Moritz. Ray: A Distributed Execution Engine for the Machine Learning Ecosystem, EECS Department, University of California, Berkeley, 2019, с. 33-47.
3. Ilya Katsov. The theory and practice of enterprise AI: Recipes and Reference Implementations for Marketing, Supply Chain, and Production Operations, 2022, с. 213–230
4. Stream Processing Platforms: URL: https://scramjet.org/stream-processing-platforms (дата звернення: 01.11.2022)
5. Spark, Dask, and Ray: Choosing the Right Framework: URL: https://www.dominodatalab.com/blog/spark-dask-ray-choosing-the-right-framework (дата звернення: 09.10.2022)
6. Jules S. Damji, Brooke Wenig, Tathagata Das, and Denny Lee. Learning Spark Lightning-Fast Data Analytics, 2020, с. 336-337
7. John Wolohan. Mastering Large Datasets with Python, 2020, 312 с.
8. Rob J. Hyndman and George Athanasopoulos. Forecasting: Principles and Practice (2nd ed), Monash University, Australia, 2018. URL: https://otexts.com/fpp2/ (дата звернення: 09.04.2023)
9. Advances in Deep Learning for Time Series Forecasting and Classification: URL: https://towardsdatascience.com/advances-in-deep-learning-for-time-series-forecasting-and-classification-winter-2023-edition-6617c203c1d1 (дата звернення: 12.02.2023)
10. Monte Carlo Dropout: URL: https://towardsdatascience.com/monte-carlo-dropout-7fd52f8b6571 (дата звернення: 19.01.2022)

11. TensorFlow Core. Time series forecasting: URL: https://www.tensorflow.org/tutorials/structured_data/time_series#1_indexes_and_offsets (дата звернення: 19.04.2023)