КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ТАРАСА ШЕВЧЕНКА
ЕКОНОМІЧНИЙ ФАКУЛЬТЕТ
КАФЕДРА ЕКОНОМІЧНОЇ КІБЕРНЕТИКИ

КВАЛІФІКАЦІЙНА РОБОТА МАГІСТРА

МОДЕЛЮВАННЯ ФІНАНСОВИХ ІНСТРУМЕНТІВ БІРЖІ МЕТОДАМИ НЕЙРОННИХ МЕРЕЖ

Студента 2 курсу магістратури
спеціальності 051 «Економіка»
освітньо-професійної програми
«Економічна кібернетика»
Атояна Андрія Гарійовича

Науковий керівник
к.е.н., доцент
Подскребко Олександр Сергійович

Засвідчую, що в цій роботі немає запозичень із праць інших авторів без відповідних посилань

Студент _____
(підпис)

Робота допущена до захисту в ЕК
рішенням кафедри економічної кібернетики
від 4 травня 2022 р., протокол № 13

Завідувач кафедри економічної кібернетики,
доктор економічних наук, професор
Ляшенко Олена Ігорівна _____
(підпис)

Київ - 2022

РЕФЕРАТ

Кваліфікаційна робота магістра містить: 120 ст., 53 рис., 3 табл., 41 джерело, 4 додатки.

Ключові слова: фондовий ринок, фінансові інструменти, часові ряди, рекурентні нейронні мережі, глибинне навчання.

Об'єкт дослідження: біржові індекси акцій компаній.

Мета дослідження: дослідження фундаментальних принципів функціонування рекурентних нейронних мереж та оцінка їх ефективності в моделюванні фінансових часових рядів.

Методи дослідження: рекурентній нейронні мережі RNN, LSTM, GRU, класичні моделі Arima, Arch, Garch (її різновиди).

Наукова новизна, теоретична значимість дослідження: полягає у формуванні фундаментального розуміння принципів роботи нейронних мереж, їх побудови, процесу навчання та його запроваджені на прикладі відомого алгоритму машинного навчання.

Практична цінність: полягає у застосуванні отриманих знань на практиці, освоєнні інструментів реалізації аналізу, а також процесу тренування та моделювання методами рекурентних нейронних мереж.

RESUME

Kyiv National Taras Shevchenko University,

Faculty of Economics, Department of Economic Cybernetics

Key words: stock market, financial instruments, time series, recurrent neural networks, deep learning.

The graduation research of student: «Modeling of exchange financial instruments using neural network methods».

Deals with: formation of a fundamental understanding of the principles of neural networks, their construction, learning process and identification of key factors influencing the results of modeling.

The practical significance is in applying the acquired knowledge in practice, mastering the tools for analysis implementation, as well as the process of training and modeling using recurrent neural networks.

Pages 120, pictures 53., tables 3, bibliog. 41, append 4.

ЗМІСТ

ВСТУП

На сьогоднішній день існує безліч методів машинного навчання, кожен з яких має власну структуру і принцип роботи. Найчастіше, чинником розподілу алгоритмів на умовні підгрупи є область застосування. У свою чергу, нейронні мережі формують власну, окрему групу, визначальним чинником якої є їх архітектура.

Наявність прихованих прошарків, різних функцій активації і великої кількості параметрів, роблять нейронні мережі досить дорогими у використанні. Але платою за це виступає висока ефективність і здатність моделювати дані краще за своїх конкурентів. Цей факт зіграв важливу роль у поширенні та вивченні цієї групи моделей.

Особливої популярності набули рекурентні нейронні мережі, чий функціонал активно використовується для моделювання часових рядів на фондовій біржі. Складна природа фінансових інструментів, поведінка яких відображає вплив відразу безлічі факторів, виражається в таких проявах як волатильність, що негативно позначається на роботі базових алгоритмів як ARIMA і GARCH моделей.

Таким чином, **метою даної роботи** є дослідження фундаментальних принципів функціонування рекурентних нейронних мереж та оцінка їх ефективності у моделюванні фінансових часових рядів, що дасть змогу синтезувати рекомендації щодо застосування та ефективного використання даного класу моделей, виявити основні переваги та недоліки.

Методи дослідження складаються з базових архітектур рекурентних нейронних мереж: RNN, LSTM, GRU та традиційних моделей машинного навчання, як: ARIMA, ARCH, GARCH та їх різновидів.

Джерелом даних є платформа «Yahoo Finance!», яка публікує безкоштовну фінансову інформацію біржових індексів. Основним інструментом реалізації виступає мова Python та R. Серед ключових пакетів слід виділити: «pytorch», «numpy», «sklearn».

Відповідно до мети, **завдання роботи** полягає в:

- ознайомленні з науковою літературою та навчальними матеріалами провідних установ, стосовно особливостей рекурентних нейронних мереж, аспектів їх побудови та факторів впливу на результати моделювання.
- дослідженні принципу навчання нейронних мереж та проектування цього підходу на прикладі логістичної регресії з метою формуванню математичного підґрунтя та розкриття ключових факторів цього процесу.
- формуванні глибинного розуміння архітектур рекурентних нейронних мереж, їх ключових компонентів, принципів обрахунку, виявленні основних переваг та недоліків відносно один одного.
- оцінці та визначенні ефективної методології прогнозування часових рядів, порівнянні результатів нейромережевого моделювання з класичними методами аналізу фінансових часових рядів.

Об'єктом дослідження є біржові індекси акцій компаній.

Предметом дослідження є алгоритми глибинного навчання, їх архітектура та фактори впливу на ефективність моделювання фінансових часових рядів. Порівняння результатів аналізу з класичними методами машинного навчання.

Теоретична цінність полягає у формуванні фундаментального розуміння принципів роботи нейронних мереж. Використанні отриманих знань як основу для вивчення більш складних архітектур.

Практична цінність полягає у застосуванні отриманих знань на практиці, освоєнні інструментів реалізації аналізу, а також процесу тренування та моделювання методами рекурентних нейронних мереж.

Наукова та практична новизна роботи, пов’язана з детальним розкриттям процесів оцінки та навчання рекурентних нейронних мереж, проектуванні даного принципу на прикладі відомого алгоритиму машинного навчання.

Актуальність теми обумовлена популярністю рекурентних нейронних мереж, враховуючи особливості їх будови та гнучкості стосовно форматів даних, що

дозволяє значно розширити область їх застосування, що виходить за рамки класичних моделей.

Структура дипломної роботи складається з вступу, змісту, трьох розділів та дев’яти підпунктів, списку використаної літератури та додатків. Загальний обсяг роботи становить 120 сторінок.

РОЗДІЛ 1. DEEP LEARNING ЯК ВАЖЛИВА СКЛАДОВА МАШИННОГО НАВЧАННЯ

1.1 Місце нейронних мереж серед алгоритмів машинного навчання.

Машинне навчання є важливим компонентом зростаючої галузі науки про дані. Завдяки використанню статистичних методів, ми можемо автоматизувати процес аналізу для вирішення ряду завдань, як класифікація, кластеризація, прогнозування, тощо. Проте більшість алгоритмів, які наразі є загально відомими, є доволі обмеженими, щодо області застосування. Саме тому, виникла потреба у формуванні нової гілки алгоритмів, що матимуть складнішу архітектору, а разом з тим, будуть здатні задовольняти нові потреби в аналізі даних.

Оскільки глибинне навчання та машинне навчання, як правило, використовуються взаємозамінно, варто відзначити різницю між ними. Машинне навчання, глибинне навчання та нейронні мережі – все це підобласті штучного інтелекту. Проте глибинне навчання насправді є підобластю машинного навчання, а нейронні мережі – глибинного.

У рамках останнього, ми можемо використовувати позначені набори даних, також відомі як навчання з учителем, але це не обов'язковою вимогою. Можливим також є використання неструктурованих даних у необробленому вигляді, як текст, зображення – NLP та CNN мережі, відповідно. У контексті трансформації даних та принципів навчання моделей, глибинне навчання, має певну перевагу. Головним фактором цього, є гнучкість алгоритмів, що входять до цієї області ML, а саме – нейронні мережі, що включають в себе велике різноманіття складних архітектур.

Штучні нейронні мережі складаються з шарів вузлів, що містять вхідний шар, один або кілька прихованих шарів та вихідний шар. Кожен вузол або штучний нейрон з'єднаний з іншими нейронами попереднього шару. Цей зв’язок корегується вагами, а саме значення нейрона, обумовлене типом функції активації та величиною константи – зміщення (рис 1.1).

Якщо вихід будь-якого окремого вузла перевищує вказане граничне значення, цей вузол активується, відправляючи дані на наступний рівень мережі. В іншому випадку дані не передаються. "Deep" в глибокому навчанні просто відноситься до

глибини шарів у нейронній мережі. Нейронна мережа, що складається з більш ніж трьох шарів, включаючи вхідні та вихідні дані, може вважатися алгоритмом глибинного навчання. Нейронна мережа, яка має лише два або три шари, це просто базова нейронна мережа.

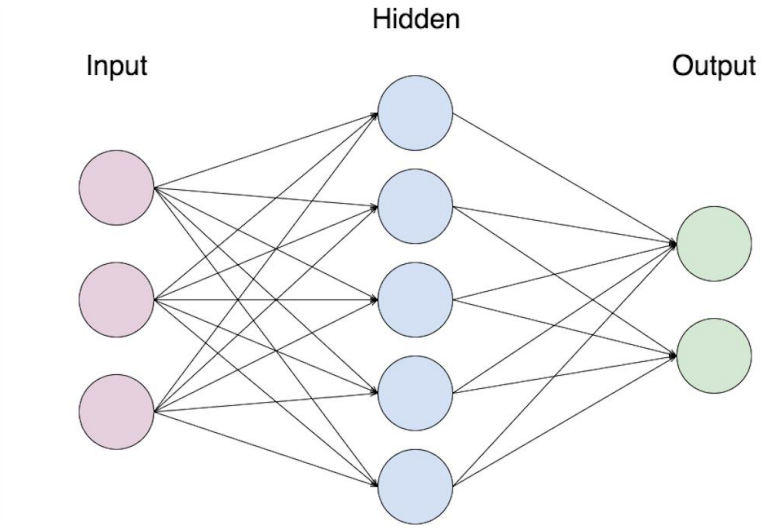


Рис. 1.1. Feed Forward Neural Network.

Джерело: Розроблено автором.

Фактично, спосіб обрахунку окремого нейрону можна описати за допомогою наступного рівняння (1.1):

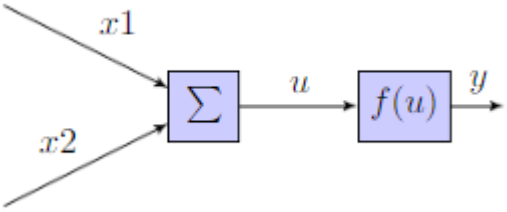
$$n_i^L = \sigma(\omega_i^L \cdot n^{[L-1]} + b_i^L) \qquad (1.1)$$

де n_i^L - це значення нейрону поточного шару, ω_i^L - вагові коефіцієнти для значень нейронів минулого шару, b_i^L - зміщення, σ - відповідна функція. Використання функції активації - це своєрідний спосіб шифрування та передачі даних до наступних прошарків.

У даному розділі ми умисно уникнемо як математичного підґрунтя самої моделі, так і процесів пов’язаних з її оцінкою та тренуванням. Лише зазначмо, що архітектура моделі, визначається предметом аналізу. І хоча перевага даного типу моделей, наразі, не виглядає очевидною, це стане більш прозорим у наступних розділах роботи.

Великій популярності та широкому застосуванню нейронних мереж, передує складний та тривалий шлях, розвитку цього типу моделей. Переважно, це пов’язано з особливістю побудови та великій кількості параметрів, необхідності їх оптимізації.

Сама рання публікація в області нейронних мереж датована 1943 роком, коли Уоррен МакКаллох і Уолтер Піттс заклали першу цеглу в фундамент передового майбутнього штучних нейронних мереж [1]. Вони розробили математичну модель штучної нейронної мережі, використовуючи порогову логіку, щоб імітувати роботу нейрона людського мозку (рис 1.2). Принцип її роботи доволі простий: якщо сума станів x_1 та x_2, більша або дорівнює шкалі θ, тоді вихідне значення нейрону дорівнює нулю, в іншому випадку – одиниці.



$$f(u) = \begin{Bmatrix} 0 : якщо\ \mu >= \theta \\ 1 : якщо\ \mu < \theta \end{Bmatrix} \quad (1.2)$$

Рис. 1.2. Структура нейрону Маккалоха і Піттса.

Джерело: A logical calculus of the ideas immanent in nervous activity [1].

Через шість років Дональд Хебб [2] у своїй книзі «The Organization of Behavior» розробив власну концепцію навчання нейронів. В її основі закладений факт того, що коли людський мозок дізнається про щось нове, нейрони активуються і з'єднуються з іншими нейронами, утворюючи нейронну мережу. Ці зв'язки спочатку слабкі, але кожного разу, коли стимул повторюється, зв'язки поступово стають сильнішими [3].

У міру того, як у 1950-х роках комп'ютери ставали дедалі досконалішими, нарешті стало можливим змоделювати гіпотетичну нейронну мережу. Перший крок до цього зробив Натаніель Рочестер [4]. Згодом, Френк Розенблатт створив модель «Perceptron», яка була першою у своєму роді для розпізнавання образів у 1958 році [5]. Але Марвін Мінськ та Сеймур Пейперт у своїй книзі «Perceptrons: an introduction to computational geometry» [6] виявили безліч проблем з моделлю, які пізніше були вирішені методом зворотного розповсюдження помилки, за допомогою градієнтного спуску, яке вперше було винайдено Полом Вербосом [7] у 1974, і популяризовано Девідом Румельхартом і ін., в 1986 роках [8].

До 1980-х років застосування нейронних мереж обмежувалося логічними та математичними операціями. Проте вчені вірили в їх здатність вирішувати складні завдання. Проте, через певний час, внаслідок збільшення обчислювальних потужностей машин та розвитку даного підходу, нейронні мережі почали застосовуватися для розпізнавання мови [9], вирішення проблеми класифікації [10], інтелектуального аналізу даних, прогнозування часових рядів та інших напрямків.

Сьогодні, нейронні мережі – це рушійна сила, що значно розширює можливості, щодо аналізу даних та в значній мірі визначне, яким буде наше майбутнє.

1.2 Види нейронних мереж та сфери їх застосування.

Ще однією перевагою нейронних мереж на фоні традиційних методів машинного навчання полягає в гнучкості їх застосування. Фактично, вони є мультизадачні, що дозволяє використовувати одну архітектуру для вирішення цілого ряду завдань.

У розділі 1.1 був розглянутий найпростіший, базовий тип архітектури нейронних мереж, а саме «Feed Forward NN». У цій мережі інформація рухається безпосередньо від вхідного шару до вихідного, долаючи при цьому вузли прихованих шарів. В залежності від типу завдання, кількість вузлів може змінюватися.

Найбільш популярною мережею цього типу є Convolutional neural network, яка широко використовується для розпізнавання образів. Як ми можемо бачити з рис 1.3, даний тип моделей має доволі складну архітектуру. В першу чергу, це пояснюється великою кількість даних, що представляють собою зображення.

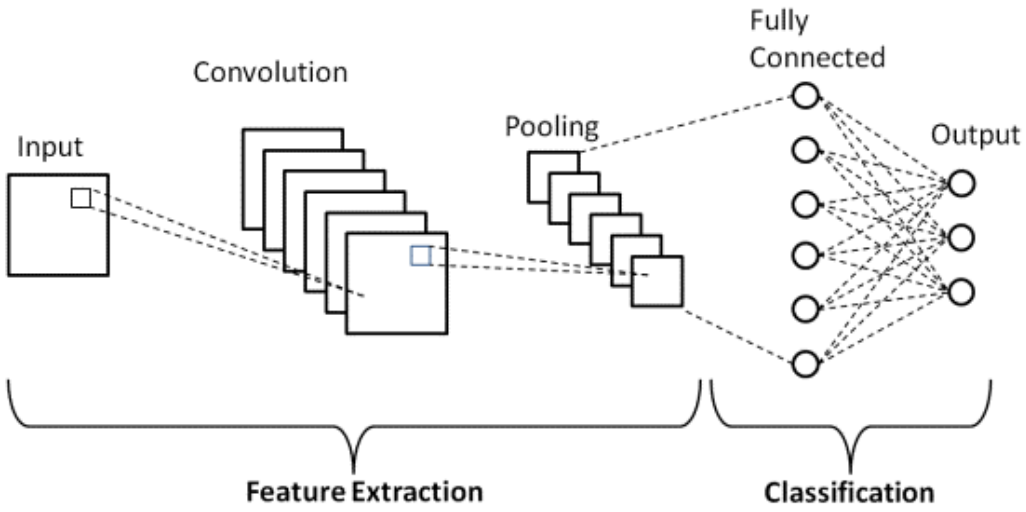


Рис. 1.3. Архітектура CNN моделі.

Джерело: Vanilla CNN architecture [12].

Під розпізнаванням образів, слід розуміти процес знаходження та певних, складових, таких як грані об’єктів, їх форми [11]. У урахуваннями того, що сучасні зображення мають високе розширення, аналіз такого масиву пікселів – є дуже є дорогим у контексті обчислення та оцінки моделі. Саме тому, CNN містять ряд структурних блоків, метою яких є:

1) Знаходженні відповідних патернів;
2) Зменшення кількості параметрів;

У найпростішій формі CNN складається з трьох ланок: згорткових, об’єднаних і повнозв’язаних шарів [12].

Згортковий шар, використовується для вилучення різних функцій із вхідних зображень. У цьому шарі виконується математична операція згортки між вхідним зображенням та фільтром певного розміру $M \times M$. При переміщенні фільтра по вхідному зображенню відбувається скалярний добуток між частинами вхідного зображення та фільтром, з урахуванням розміру останнього.

Вихідні дані даного шару, називаються картою об'єктів, яка дає нам інформацію про зображення, у вигляді певних граней. Після цього, карта об'єктів передається наступним шарам для вивчення інших особливостей вхідного зображення.

У більшості випадків за згортковим шаром йде шар об’єднання. Його мета – зменшення розміру згорнутої карти об’єктів, а разом з тим і витрат на обчислення. У контексті роботи з зображеннями, роботу цього шару важко недооцінити, з урахуванням аспекту, описаному на початку. Залежно від методу, існує кілька типів операцій об'єднання. Проте їх розгляд знаходиться за рамками даного розділу.

Останній шар – «Fully connected», є проміжними шаром і використовується для з'єднання нейронів між двома вузлами. В архітектурі CNN зазвичай розміщуються наприкінці. Метою даного шару є аналіз отриманої інформації та її передачі до вихідного вузла.

На даному етапі, слід приділити особливу увагу функціям активації. Їх призначення полягає в шифруванні інформації у певному діапазоні значень для її передача до наступних прошарків. Як правило, при виборі функції необхідно спиратися на ряд факторів, такі як: тип проблеми, що вирішує мережа; місце вузла в архітектурі; її вплив на ефективність моделювання. Найбільш поширеними функціями активаціями є: sigmoid, relu, tanh, за межами $[0; 1]$, $[0; +\infty]$, $[-1; 1]$.

У випадку CNN для класифікації зображень, на передостанньому вузлі, нерідко використовується функція «softmax» (1.3). Головна мета - нормалізація вихідних даних мережі до розподілу ймовірностей за прогнозованими вихідними класами [13].

$$softmax(z_i) = \frac{e^{z_i}}{\sum_{i=1}^{m} e^{z_i}} \qquad (1.3)$$

Застосовується у випадках, коли зображення містить тільки один з можливих класів. У нашому випадку рис. 1.3, мережа не має такого обмеження. А тому, може ідентифікувати декілька класів за їх наявності на малюнку.

На рис 1.4 проілюстрований принцип роботи CNN моделі [14]. Як було зазначено на початку, метою є знаходження певних граней та форм. В залежності від складності завдання, архітектура моделі може змінюватися, шляхом збільшення відповідних вузлів. Проте концепція залишається незмінною.

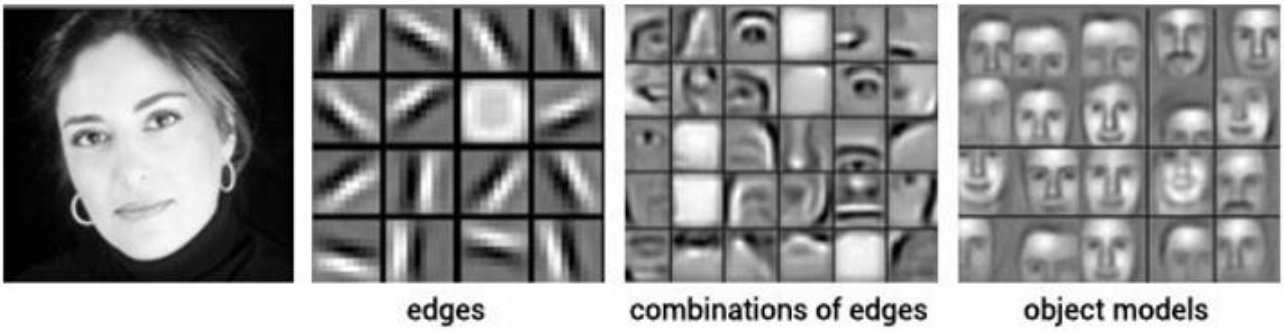


Рис 1.4. Принцип роботи CNN моделі.

Джерело: Understanding Neural Networks Through Deep Visualization [14]

Для прикладу, дана нейронна мережа може виконувати завдання класифікації щодо статі людини, віку. Даний вид аналізу є недосяжним для традиційної методології, в той час як для нейронних мереж, це лише одна з можливих сфер [15]. Серед них, слід виділити наступні: розпізнавання зображень і відео (особливо в медицині), комп'ютерний зір, обробка природної мови [16].

В свою чергу, об'єктом аналізу рекурентних нейронних мереж (RNN) є послідовності, або дані часових рядів. У даному розділі ми уникнемо архітектурних особливостей цього типу мереж, оскілки йому присвячені подальші розділи цієї роботи. Замість цього, детальніше запнимося на області застосування.

Цей алгоритм глибинного навчання зазвичай використовуються для таких завдань, як: мовний переклад, аналіз тональності тексту, обробка природної мови, моделювання фінансових часових рядів [17], а у деяких випадках – для аналізу зображень, що є основним напрямком CNN. Цей факт - ще одне підтвердженням тезису, про мультизадачність нейронних мереж [18].

RNN є складовими таких відомих програм як «Siri», голосовий пошук та «Google Translate». Їх архітектура відрізняються властивістю побудови блоку «пам'яті», оскільки вони враховують інформацію з попередніх ітерації. Після обрахунку рекурентного шару, результат зберігається та використовується в якості вхідного значення, наряду із новими даними. У той час як традиційні нейронні мережі припускають, що вхідні та вихідні дані незалежні один від одного.

Як ми можемо бачити, алгоритми deep learning, вирізняються особливостями власної, складної архітектури та мають широку сферу застосування. Їх гнучкість відносно форматів даних та будови моделей, дозволяють вирішувати завдання недосяжні для традиційних методів машинного навчання. Наразі, ми розглянули лише три базові конструкції нейронних мереж. Проте в дійсності, їх значно більше, як і сфер аналізу, які стали досяжними за вдяки їм.

1.3 Моделювання принципу роботи НМ на прикладі логістичної регресії.

Метою цього розділу є демонстрація принципу навчання нейронних мереж, на прикладі відомого алгоритму машинного навчання. В даному випадку ми вирішили використати логістичну регресію, що дозволяє вирішити задачу класифікації. Такий вибір обґрунтований наступними факторами:

1. сигмоїда логістичної регресії є однією з можливих функцій активації, що використовуються для звуження значень в середині нейронів у певному діапазоні (у разі сигмовидної функції це [0, 1]). У світлі проблеми класифікації це дає нам можливість отримати значення вірогідності подій. У нейронних мережах вона використовується для стиснення інформації та її передачі у наступні прошароки.
2. функція витрат логістичної регресії – показник, що демонструє, наскільки добре модель навчена на різних етапах (чим нижче значення – тим краще); Її узагальнена версія також відома як Cross entropy, де вибір не є біномним. Остання, доволі часто використовується у нейронних мережах, тому доцільним буде вивчення її попередню версії для формування кращого розуміння.

Перш за все, нам потрібно завантажити наші дані. Для того, щоб охопити всі аспекти, описані в цьому розділі, було створено спеціальний набір даних, що складається з 4 колонок. Нас цікавлять змінні "x1", "x2", "y" (рис. 1.5.).

	x1	x2	y	X1plusX2
0	89.946963	72.091781	1	162.038744
1	77.111425	75.206589	1	152.318014
2	71.854053	80.549965	1	152.404018
3	67.182350	97.026782	1	164.209131
4	71.087755	71.098405	1	142.186160

Рис. 1.5. Вміст файлу «data_logit».

Джерело: розроблено автором.

Поширеним підходом є виконання деяких перетворень даних перед навчанням моделі. Алгоритми оптимізації, такі як градієнтний спуск, вимагають масштабування

даних, щоб забезпечити більш ефективне обчислення для пошуку оптимального рішення.

Оскільки величини незалежних змінних можуть відрізняється один від одного, це може істотно вплинути на розмір кроку, навіть якщо ми використовуємо learning rate в 0,01. Різниця в діапазонах спричиняє непропорційні кроки під час ітерацій. Це, у свою чергу, може збільшити час навчання і зробити його більш витратним у обчислювальному плані.

Існує два поширені підходи, які ми можемо застосувати, щоб запобігти цьому:

- **Нормалізація** – передбачає зведення діапазону даних у межі [0; 1]. Є доволі поширеною технікою перетворення у сферах машинного та глибинного навчаннях. Для прикладу: CNN моделі, що застосовуються для розпізнання образів візуальних зображень, використовують значення пікселів у якості вхідних даних. Останні, знаходяться у чітких межах [0, 255]. Застосування нормалізації дозволяє пришвидшити процес навчання та знаходження оптимального рішення. Відповідна формула розрахунку представлена нижче:

$$x_i = \frac{x_i - x_{\min}}{x_{\max} - x_{\min}},\ i = \overline{1,n} \quad (1.4)$$

- **Стандартизація** - техніка в основі трансформації якої лежать дві характеристики ряду: середнє значення μ та стандартне відхилення σ. На відміну від нормалізації, не має чітких меж.

$$x_i = \frac{x_i - \mu}{\sigma},\ i = \overline{1,n} \quad (1.5)$$

Для того, щоб здійснити вибір на користь тієї чи іншої трансформації, необхідно детально ознайомитися зі структурними елементами самої моделі. Для цього розглянемо взаємозв'язок сигмоїди із функцією витрат моделі (1.6).

$$Loss = \frac{1}{n}\sum_{i=1}^{n}(y_i \log_e \hat{y}_i + (1-y_i)\log_e(1-\hat{y}_i)) \quad (1.6)$$

Бінарна класифікація передбачається два можливі значення змінної y_i. Рівняння побудоване таким чином, що в залежності від того, чим є y_i (1 або 0), ми потрапляймо

у першу, або в другу частину рівняння, відповідно. В обох випадках ми маємо справу з логарифмом, мета якого полягає в нарахування штрафів. Для його уникнення, ми прагнемо, щоб кінцеве значення аргументу було наближене до одиниці, оскільки $\log_e 1 = 0$. Для цього, оцінки моделі $\hat{y}_i$ мають бути максимально близькі до реальних значень y_i.

У свою чергу, підрахунок $\hat{y}_i$ відбувається за формулою (1.7). Саме тут і знаходиться відповідь на наше питання. У разі, якщо ми застосуємо нормалізацію, то наші значення будуть знаходитися у межах [0; 1]. Підставивши їх у рівняння сигмоїди, ми матимемо справу лише з однією половиною графіку, оскільки $sigmoid(0) = 0.5$. Якою буде ця половина, залежить від знаку коефіцієнтів моделі. Це, в свою чергу призведе до труднощів із навчанням.

$$sigmoid = \frac{1}{1+e^{-\beta x}} \quad (1.7)$$

Застосувавши стандартизацію, ми уникнемо цієї проблеми, оскільки початковий ряд буде трансформований відповідно до його характеристик та включатиме як негативні так і позитивні значення. Це забезпечить розміщення оцінок в вздовж усієї синусоїди та пришвидшить процес навчання.

Але перш ніж ми застосуємо трансформацію, необхідно розділити наші дані на дві групи: навчальну та тестову. Це дозволить нам пізніше протестувати продуктивність моделі. Частка навчальних/тестових груп становить 80/20 відповідно.

У цьому розділі ми відтворимо процес оптимізації вручну. Причина цього в тому, що ми хочемо показати передумови навчання моделі, математику, що стоїть за цим, щоб створити базове розуміння цих процесів. Саме тому, для цієї секції був написаніта іплементовані блоки коду (додаток А).

Кожен алгоритм має свої етапи, які нам потрібно пройти, щоб досягти оптимізації параметрів. Через те, що ми маємо справу з величезною кількістю параметрів, знайти градієнт для кожного параметру щодо функції Loss є доволі складним завданням. Саме тому в світі нейронних мереж етапи обчислення моделі представлені за допомогою «computational graphs» [19], де кожен вузол відповідає

певній математичної операції. Такий підхід усуває складність обчислень та забезпечує інтуїтивність навчання моделі (forward та backward propagations), спрощує математику.

Великою перевагою є те, що ми можемо застосувати ту саму техніку до інших алгоритмів із меншою кількістю параметрів. В нашому випадку ми проілюструємо це за допомогою логістичної регресії, розрахунковий граф якої показаний нижче.

В моделі логістичної регресії ми прагнемо модифікувати параметри W_x та b таким чином, щоб мінімізувати функцію втрат. Оскільки ми маємо 2 незалежні змінні, розрахунковий граф має наступний вигляд (рис 1.6).

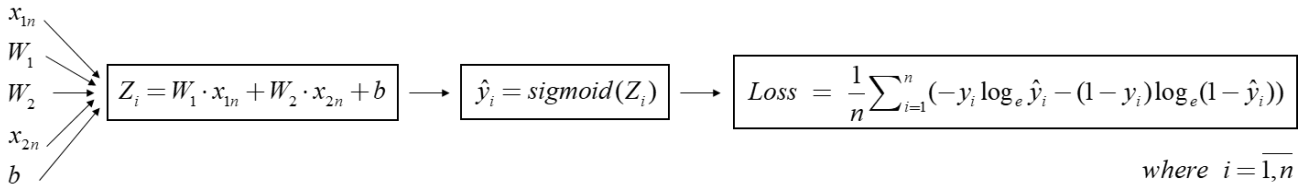


Рис. 1.6 Розрахунковий граф логістичної регресії.

Джерело: розроблено автором.

Для початку, розглянемо знаходження часткової похідної функції витрат відносно W_x для окремого спостереження (1.8):

$$\frac{\partial Loss}{\partial W_j} = \frac{\partial Loss}{\partial \hat{y}_i} \cdot \frac{\partial \hat{y}_i}{\partial z_i} \cdot \frac{\partial z_i}{\partial W_j} \tag{1.8}$$

$$\frac{\partial Loss}{\partial \hat{y}_i} = \left(-\frac{y_i}{\hat{y}_i} - \frac{(1-y_i)}{(1-\hat{y}_i)} \cdot (-1) \right) = \left(\frac{\hat{y}_i - y_i\hat{y}_i - y_i + y_i\hat{y}_i}{\hat{y}_i(1-\hat{y}_i)} \right) = \frac{\hat{y}_i - y_i}{\hat{y}_i(1-\hat{y}_i)}$$

$$\frac{\partial \hat{y}_i}{\partial z_i} = (-1) \cdot (1+e^{-z_i})^{-2} \cdot e^{-z_i}(-1) = \frac{e^{-z_i}}{(1+e^{-z_i})^2}$$

$$\frac{\partial z_i}{\partial W_j} = x_{jn} \ \ , where \ j = \overline{1,2}$$

Використовуючи автоматизоване програмне забезпечення, ми отримаємо точно такий же результат для $\frac{\partial \hat{y}_i}{\partial z_i}$. Проте для зручності подальших обчислень ми можемо перезаписати отримане рівняння як (1.9):

$$\frac{e^{-z_i}}{(1+e^{-z_i})^2} = \frac{1}{(1+e^{-z_i})} \cdot \frac{(1-1+e^{-z_i})}{(1+e^{-z_i})} = \frac{1}{(1+e^{-z_i})} \cdot \left(1 - \frac{1}{(1+e^{-z_i})} \right) = \hat{y}_i \cdot (1-\hat{y}_i) \tag{1.9}$$

Наразі, ми маємо все необхідне для знаходження часткової похідної. Слід також узяти до уваги те, що на практиці, ми працюємо з вибіркою даних, а тому, маємо провести аналогічні розрахунки для всіх спостережень.

З урахування даного аспекту та результатів отриманих вище, градієнт градієнт функції витрат відносно W_x дорівнює (1.10):

$$\frac{\partial Loss}{\partial W_j} = \frac{1}{n} \cdot \sum_{i=1}^{n} \frac{\hat{y}_i - y_i}{\hat{y}_i(1-\hat{y}_i)} \cdot \hat{y}_i(1-\hat{y}_i) \cdot x_{jn} = \frac{1}{n} \cdot \sum_{i=1}^{n} (\hat{y}_i - y_i) \cdot x_{ji} \qquad (1.10)$$

У випадку b градієнт дорівнює (1.11):

$$\frac{\partial Loss}{\partial b} = \frac{1}{n} \sum_{i=1}^{n} \frac{\hat{y}_i - y_i}{\hat{y}_i(1-\hat{y}_i)} \cdot \hat{y}_i(1-\hat{y}_i) \cdot 1 = \frac{1}{n} \sum_{i=1}^{n} (\hat{y}_i - y_i) \qquad (1.11)$$

Слід також зазначити, що використовуючи Gradient descent у якості алгоритму оптимізації, ми маємо поставити мінус перед функцією втрат. Наша мета, мінімізувати значення функції, діставшись глобального мінімуму – точки, де параметри моделі оптимізовані. Оновлення параметрів відбувається шляхом віднімання від їх фактичного значення градієнтів, помножених на learning rate. В іншому випадку, ми маємо використовувати Gradient Ascent. В обох випадках ми отримаємо однакові результати, проте перший підхід є загально прийнятою практикою.

Фактично, **backpropagation** - це не що інше, як **chain rule** у зворотному напрямку. Причина його використання в тому, що він дозволяє нам знаходити часткові похідні для всіх ваг одразу. В іншому випадку нам довелося б обчислювати їх окремо для кожної параметру (починаючи спочатку).

У світлі логістичної регресії пошук параметрів не є витратним з точки зору обчислення. А як щодо інших алгоритмів? На практиці нам потрібно знайти оптимальний підхід, щоб отримати ті ж результати, проте більш ефективним способом.

Рішенням є застосування матриць Якобі [20]. Ідея полягає в тому, що ми знаходимо похідні відносно параметрів кожного вузла для всіх спостережень, збираючи їх у матриці, а потім перемножуємо їх у зворотному порядку. Це дозволяє

нам векторизувати дані та досягти бажаного результату за рахунок множення матриць.

Те, що ми маємо взяти до уваги — це форма об'єкта, для якого хочемо взяти градієнт. У наведеному нижче прикладі ми провели множення матриць Якобі (1.12):

1) $z_i = b + W_1 \cdot x_{1i} + W_2 \cdot x_{2i}$, *where* $i = \overline{1,n}$ (1.12)

$$f\left(\begin{bmatrix} b \\ W_1 \\ W_2 \end{bmatrix}\right) = \begin{bmatrix} \frac{\partial z_1}{\partial b} & \frac{\partial z_1}{\partial W_1} & \frac{\partial z_1}{\partial W_2} \\ \frac{\partial z_2}{\partial b} & \frac{\partial z_2}{\partial W_1} & \frac{\partial z_2}{\partial W_2} \\ & .... & \\ \frac{\partial z_n}{\partial b} & \frac{\partial z_n}{\partial W_1} & \frac{\partial z_n}{\partial W_2} \end{bmatrix}; \qquad J^{(1)} = \begin{bmatrix} 1 & x_{11} & x_{21} \\ 1 & x_{12} & x_{22} \\ & .... & \\ 1 & x_{1n} & x_{2n} \end{bmatrix};$$

2) $y_i = \sigma(z_i) = \frac{1}{1+e^{-z_i}}$, *where* $i = \overline{1,n}$

$$f\left(\begin{bmatrix} z_1 \\ z_2 \\ .... \\ z_n \end{bmatrix}\right) = \begin{bmatrix} \frac{\partial y_1}{\partial z_1} & \frac{\partial y_1}{\partial z_2} & .... & \frac{\partial y_1}{\partial z_n} \\ \frac{\partial y_2}{\partial z_1} & \frac{\partial y_2}{\partial z_2} & .... & \frac{\partial y_2}{\partial z_n} \\ & .... & & \\ \frac{\partial y_n}{\partial z_1} & \frac{\partial y_n}{\partial z_2} & .... & \frac{\partial y_n}{\partial z_n} \end{bmatrix}; \qquad J^{(2)} = \begin{bmatrix} \frac{e^{-z_1}}{(1+e^{-z_1})^2} & 0 & .... & 0 \\ 0 & \frac{e^{-z_2}}{(1+e^{-z_2})^2} & .... & 0 \\ & & .... & \\ 0 & 0 & .... & \frac{e^{-z_n}}{(1+e^{-z_n})^2} \end{bmatrix};$$

3) $Loss = \frac{1}{n}\sum_{i=1}^{n}(-y_i \log_e \hat{y}_i - (1-y_i)\log_e(1-\hat{y}_i))$, *where* $i = \overline{1,n}$

$$f\left(\begin{bmatrix} \hat{y}_1 \\ \hat{y}_2 \\ .... \\ \hat{y}_n \end{bmatrix}\right) = \begin{bmatrix} \frac{\partial Loss}{\partial \hat{y}_1} & \frac{\partial Loss}{\partial \hat{y}_2} & .... & \frac{\partial Loss}{\partial \hat{y}_n} \end{bmatrix};$$

$$J^{(3)} = \begin{bmatrix} \frac{1}{n} \cdot \frac{\hat{y}_1 - y_1}{\hat{y}_1(1-\hat{y}_1)} & \frac{1}{n} \cdot \frac{\hat{y}_2 - y_2}{\hat{y}_2(1-\hat{y}_2)} & .... & \frac{1}{n} \cdot \frac{\hat{y}_n - y_n}{\hat{y}_n(1-\hat{y}_n)} \end{bmatrix};$$

Для того, що отримати градієнти параметрів моделі, необхідно здійснити перемноження матриць у зворотному напрямку: $J = J^{(3)}(1\times n)\, J^{(2)}(\mathrm{n}\times n)\, J^{(1)}(\mathrm{n}\times 3)$. В результаті, отримаємо матрицю розмірністю в $J(1\times 3)$.

Таким чином, використовуючи матриці Якобі, ми можемо векторизувати знаходження градієнтів параметрів моделі, що значною мірою спрощує обрахунок та економить ресурси.

Для того, щоб оновити значення параметрів (1.13), нам потрібно відняти відповідні градієнти. Але, перш ніж ми це зробимо, нам потрібно встановити так звану швидкість навчання (learning rate). Ідея цього параметра у тому, що він визначає величину кроку кожної ітерації.

$$W_1 = W_1 - lr \cdot \frac{\partial Loss}{\partial W_1}\ ; \quad W_2 = W_2 - lr \cdot \frac{\partial Loss}{\partial W_2}\ ; \quad b = b - lr \cdot \frac{\partial Loss}{\partial b}; \qquad (1.13)$$

Високе значення цього параметру дозволяє моделі вчитися швидше. Теоретично ми можемо досягти глобального мінімуму за меншу кількість ітерацій. Але це не завжди так. Коли кроки надто великі, це може завадити збіжності в оптимальній точці.

У той час, як низький learning rate передбачає більш оптимальний спосіб навчання моделі, але значно збільшує кількість ітерацій. Проблема, з якою ми можемо зіткнутися тут, полягає в тому, що у світлі попередніх перетворень даних може знадобитися надмірна кількість ітерацій через крихітні оновлення параметрів. У деяких випадках 20.000 – 30.000 ітерацій буде недостатньо.

Таким чином, ми повинні враховувати масштаб даних, протестувати кілька значень, щоб вибрати оптимальну швидкість навчання, використовуючи функцію втрат як індикатор навчання моделі. Пізніше, ми розглянемо, деякі з згаданих тут очікувань.

У ході виконання даного розділу, мною був написаний код, для оцінки моделей та реалізації зазначених концепцій. Для підтвердження описаної математики, була побудована власна логістична регресія за допомогою інструментів пакету «pytorch»,

а отримані результати пізніше використані для повіряння з обрахунком автоматизованих версій моделі (рис. 1.7).

	Custom Logit	Automated
0	-0.0688	-0.068787
1	1.1434	1.143450
2	1.5835	1.583526

Рис. 1.7. Значення оцінок коефіцієнтів логістичної регресії.

Джерело: розроблено автором.

Як ми можемо бачити, ми отримали ті ж самі результати. Слід зауважити, що алгоритм, який використовується в автоматизованій функції LogisticRegression з пакету «sklearn», відрізняється від нашого. Причина в тому, що існує безліч алгоритмів оптимізації, які можна використовувати для пошуку параметрів. Більшість з них засновані на градієнті, як і метод: Newton-CG. Вони можуть ефективнішими в порівнянні зі стохастичним градієнтним спуском SGD, який ми представили у нашій моделі, але результат має бути таким самим. Таким чином, можна сказати, що ми зробили повторну перевірку і ще раз впевнились в оцінках.

Графічним відображенням процесу навчання моделі є графік значень функції втрат. Слід зазначити, що в ході оптимізації параметрів загальна кількість ітерацій складала 4000, learning rate становив 0.1, що є нетиповим значенням (рис 1.8).

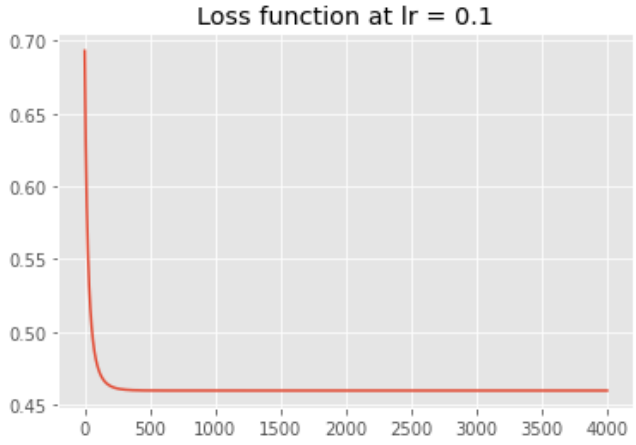


Рис. 1.8. Динаміка зміни функції витрат моделі, при learning rate в 0.1.

Джерело: розроблено автором.

Як видно з графіку, втрати значно зменшилися після перших 200 ітерацій. Потім почалося скорочення, але масштаби змін стали дуже малими. Тому решта значень нагадує витягнуту горизонтальну лінію, що насправді не так. Оновлення буде виконуватися до того моменту, поки ми не досягнемо глобального мінімуму або будемо настільки близькими до нього, що градієнт, помножений на швидкість навчання, майже не дасть результату.

Для прикладу, змінимо значення швидкості навчання на 0,01 і проведемо ту саму операцію (рис 1.9).

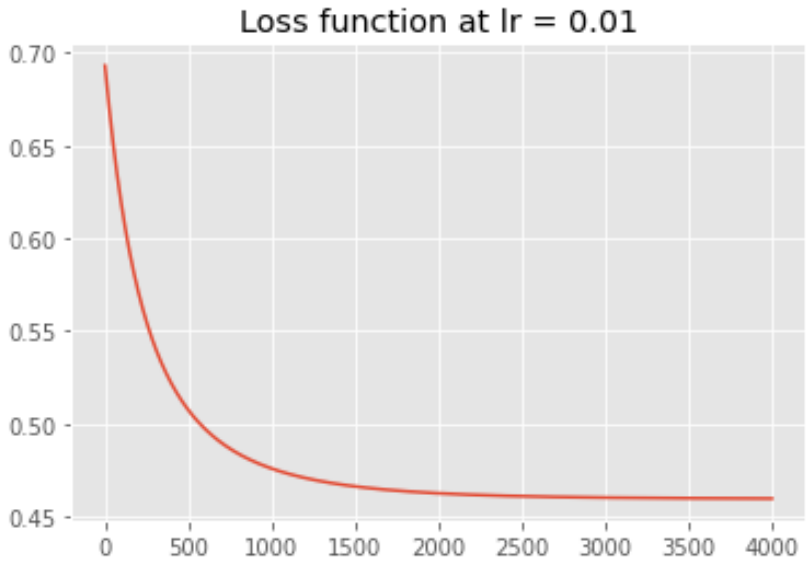


Рис. 1.9. Динаміка зміни функції витрат моделі, при learning rate в 0.01.

Джерело: розроблено автором.

Схил більш плавний, що пояснюється розміром кроку. На горизонтальній осі більше немає прямої лінії. Більше того, судячи з оцінок коефіцієнтів (рис. 1.10), вони повністю оптимізовані. Це той випадок, який ми описували у теоретичній частині, на початку.

	Logit, lr=0.01
0	-0.0637
1	1.1011
2	1.5250

Рис. 1.10. Значення коефіцієнтів моделі, при learning rate в 0.01.

Джерело: розроблено автором.

Незважаючи на те, що ми змінили швидкість навчання з 0,1 на 0,01, ми повинні пам'ятати про масштаб самих даних, змінених шляхом регуляризації. Якщо вплив цього фактора не є неочевидним, згадаймо формули, які ми отримали, провівши розрахунки раніше.Тому ми також маємо враховувати масштаб даних, з якими ми працюємо.

Ми вже закінчили основну частину оптимізації логістичної регресії, використовуючи підхід нейронних мереж. Після завершення тренування моделі на навчальному наборі даних, ми можемо використовувати отримані коефіцієнти, щоб знайти значення прогнозів для тестового набору, який ми підготували раніше.

Для порівняння продуктивності моделі на різних сетах даних ми будемо використовувати точність в якості основного показника. Для цього нам потрібно знайти криву ROC, яка визначає правильний поріг класифікації ймовірностей.

Ми не будемо вдаватися до деталей confusion matrices, кривих ROC і пов'язаних з ними термінології — це виходить за межі даного розділу. Але надзвичайно корисно розуміти ідею, яка стоїть за цим, щоб забезпечити якість аналізу.

Немає єдиного способу визначення оптимального порогу. Залежно від того, який параметр ми хочемо максимізувати, ми можемо вибрати конкретний метод серед багатьох інших. Але найбільш загальний підхід полягає у використанні Youden's J-statistics. Згідно результатів обчислення, оптимальний поріг складає 0,4146.

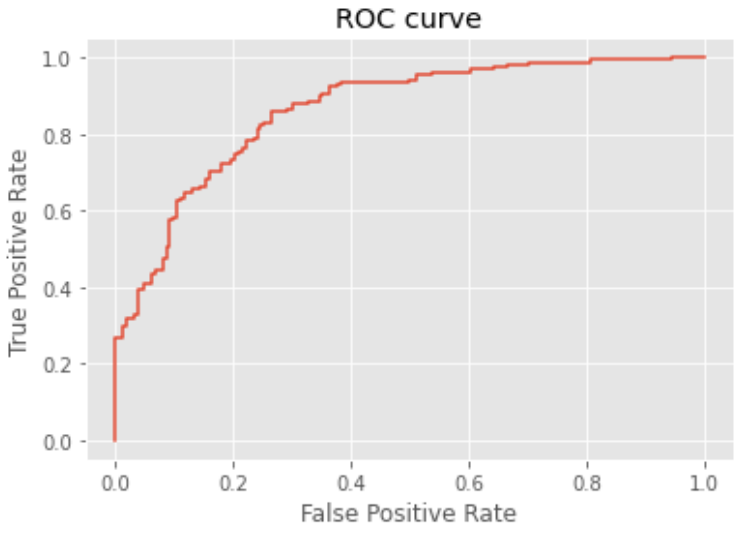


Рис. 1.11. ROC крива для логістичної регресії.

Джерело: розроблено автором.

Точність моделі при використанні тестового зразка становить 81% (правильно ідентифіковано 65 міток з 80). Тоді як у випадку навчального набору точність дещо нижча (79% міток визначено правильно).

Як зазначалося на початку, мета цієї глави - розкрити принцип навчання нейронних мереж. Ми розпочали з блоку теорії, який формував базове розуміння наведених процесів. На його основі, ми побудували модель логістичної регресії, по можливості уникаючи автоматизованого програмного забезпечення. У міру поглиблення теми, ми поліпшили код і визначили основні чинники, які можуть вплинути на ефективність оптимізації стохастичним градієнтним спуском. Ми показали на практиці важливість трансформації даних та значення швидкості навчання, розкрили поняття обчислювального графа та векторизації обчислень шляхом добутку матриць Якобі. Результати наших моделей співпали із результатами запропонованого автоматизованого рішення.

Незважаючи на те, що подальший аналіз виходить за межі цього розділу, було прийняте рішення його завершити. Знайдені ROC-крива, оптимальний поріг та матриці плутанини. Основним параметром якості моделі була точність, значення якої складає 79% та 81% для навчальної та тестової вибірок відповідно.

РОЗДІЛ 2. АРХІТЕКТУРИ РЕКУРЕНТНИХ НЕЙРОННИХ МЕРЕЖ.

2.1 Загальні положення RNN моделі.

У розділі 1.3 було розглянуто ідею навчання нейронних мереж на прикладі логістичної регресії. На цей раз, увага буде акцентована на самих нейронних мережах. У зв'язку з тим, що предметом аналізу є дані фінансових часових рядів, ми розглянемо рекурентні нейронні мережі. У даному розділі ми розглянемо наступні моделі:

- RNN (Recurrent neural network);
- LSTM (Long short-term memory);
- GRU (Gated recurrent unit);

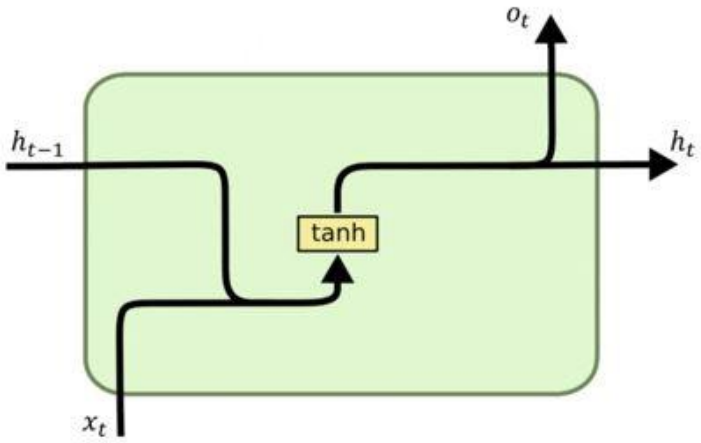
Метою теоретичної частини є дослідження модульної архітектури цих моделей, виявлення їх сильних і слабких сторін відносно один одного. Практична частина включатиме ручну реконструкцію даних архітектур, що дозволить нам глибше зрозуміти деталі обчислювального процесу та порівняти результати з автоматизованими функціями пакету «pytorch».

Рекурентні нейронні мережі – це особливий вид алгоритмів глибокого навчання. У попередніх розділах ми розглянули різні архітектури, принципи їх навчання. На цей раз ми розглянемо самі рекурентні мережі та конструкції, що широко використовуються при аналізі часових рядів.

RNN, показана нижче (рис 2.1), є найпростішою архітектурою цього класу моделей. Вона представлена одним рівнянням, яке розраховується на основі попереднього значення активації нейрона та вхідних даних. Функція активації, зазначена на рис _, не є суворим правилом. Для прикладу, у пакеті «pytorch» можливими є застосування двох варіантів: tanh з діапазоном значень $[-1;1]$ та relu $[0;+\infty]$. В результаті проведення відповідних розрахунків, ми отримуємо вихідне значення нейрону поточної ітерацій, що також використовується у якості вхідного значення для наступної.

Звичайно, цей тип моделі можна модифікувати, збільшивши кількість вхідних даних або прихованих станів, які необхідно обчислити. А ось як це буде реалізовано,

може бути не так очевидно, як здається на перший погляд. Саме тому теоретична частина кожної моделі супроводжується практичною реалізацією (додаток В).



$$h_t = \tanh(W_{ih}x_t + b_{ih} + W_{hh}h_{(t-1)} + b_{hh})$$

Рис. 2.1. Архітектура блоку пам’яті RNN моделі.

Джерело: розроблено автором на основі [21]

Доволі часто, у ході дослідження даної сфери я зіткався з проблемою сумісності теоретичної та практичних частин. Це у свою чергу, створює певні перешкоди у пізнанні обраної теми.

input_size = 1, hidden_size = 1

$$h_t = tanh(W_{x_{11}}x_{t1} + b_{x_1} + W_{h_1}h_{t-1} + b_{h_1})$$

input_size = 2, hidden_size = 1

$$h_t = tanh(W_{x_{11}}x_{t1} + W_{x_{12}}x_{t2} + b_{x_1} + W_{h_1}h_{t-1} + b_{h_1})$$

input_size = 1, hidden_size = 2

$$\begin{bmatrix} h_t \\ c_t \end{bmatrix} = tanh\begin{pmatrix} W_{x_{11}}x_{t1} + b_{x_1} + W_{h_1}h_{t-1} + W_{c_1}c_{t-1} + b_{h_1} \\ W_{x_{21}}x_{t1} + b_{x_2} + W_{h_2}h_{t-1} + W_{c_2}c_{t-1} + b_{c_1} \end{pmatrix}$$

input_size = 2, hidden_size = 2

$$\begin{bmatrix} h_t \\ c_t \end{bmatrix} = tanh\begin{pmatrix} W_{x_{11}}x_{t1} + W_{x_{12}}x_{t2} + b_{x_1} + W_{h_1}h_{t-1} + W_{c_1}c_{t-1} + b_{h_1} \\ W_{x_{21}}x_{t1} + W_{x_{22}}x_{t2} + b_{x_2} + W_{h_2}h_{t-1} + W_{c_2}c_{t-1} + b_{c_1} \end{pmatrix}$$

$$where\ t = \overline{1,n}\ in\ each\ case$$

Рис. 2.2. Приклади обрахунку RNN в залежності від заданих параметрів.

Джерело: розроблено автором основі [21].

Тому, для забезпечення базового розуміння реалізації RNN відомими пакетами, як «pytorch» [21], були написані відповідні блоки для кожної із моделей та розглянуто декілька випадків (рис 2.2).

Як бачимо, враховуючи структуру даних та значення гіперпараметрів, які ми встановили, обчислення всередині відрізняються. Наприклад: урахування значень інших прихованих станів для розрахунку кожного стану окремо – не є очевидним. Ось чому нам слід уважно вивчити роботу автоматизованих функцій. Забігаючи наперед, слід зазначити, що спосіб зберігання вагових матриць є однаковим для всіх трьох моделей.

Розуміння того, як виконується **forward propagation** - дуже важливе, але також необхідно охопити й інший бік питання - **backward propagation**.

Щоб оновити наші початкові ваги, нам потрібно застосувати техніку, з якою ми вже знайомі з розділу 1.3. У випадку рекурентних нейронних мереж це називається «BPTT» - зворотне поширення в часі. Щоб проілюструвати, як обчислюються градієнти, ми спростимо наші розрахунки, використовуючи традиційне представлення моделі RNN. Але перш ніж почати, ми також повинні згадати різні типи RNN.

Працюючи з даними часових рядів, рекурентні нейронні мережі стали надзвичайно популярними в ряді областей, як-от: розпізнавання мовлення, класифікація настроїв, прогнозування фінансових часових рядів тощо. Залежно від проблеми, яку ми збираємося вирішити, можуть застосовуватися різні типи RNN.

У статті, опублікованій Afshine Amidi та Shervine Amidi зі Стендфордського університету, надається список відповідних типів [22]. У нашому випадку ми будемо використовувати конструкцію «many-to-one». Розгорнуте представлення моделі показано нижче (рис. 2.3).

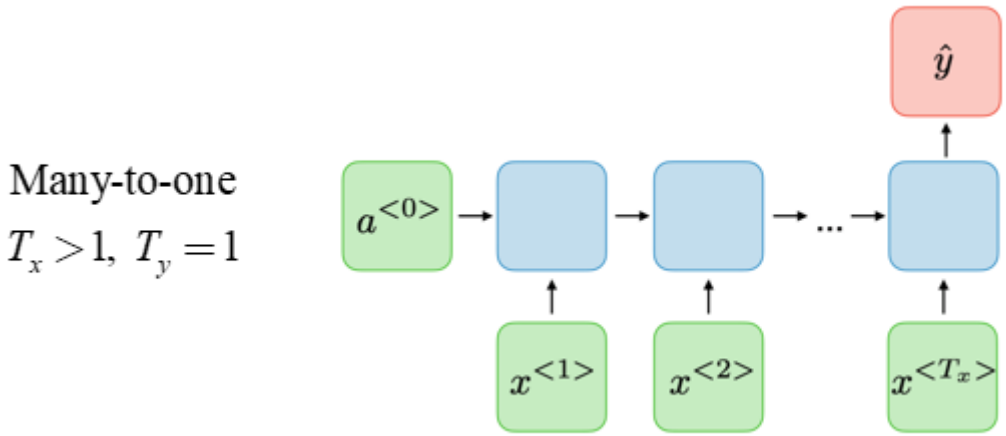


Рис. 2.3. Графічне зображення моделі типу «many-to-one»

Джерело: розроблено автором на основі [22].

Робота з нейронними мережами передбачає використання величезної кількості даних. Для ефективного навчання моделі ми збираємо спостереження в «batches». Технічно це тривимірний тензор, що складається з групи екземплярів із набору даних.

Фінансові часові ряди, як і будь-які інші дані, повинні бути спочатку перетворені для використання в подальших обчисленнях. У нашому випадку ми хочемо визначити часові рамки для попередніх періодів, на які наша мережа буде покладатися у ході навчання. Потім ми розбиваємо ряд відносно цього фрейму, зміщуючи його на одне спостереження за раз. В результаті отримуємо групу послідовностей. Набір таких послідовностей є не що інше, як самі «batches».

Використовуючи RNN типу «many-to-one», ми припускаємо, що прогноз моделі (на один період вперед) буде заснований на інформації з попередніх n-періодів. Для кращого розуміння того, як обчислюються градієнти, розглянемо випадок, коли розмір «batches» відносно невеликий [23]. Це дозволить нам охопити зворотне поширення в часі моделі RNN і уникнути складних обчислень, що стає очевидним з рівнянь нижче (рис. 2.4).

Розглянемо RNN модель, типу:

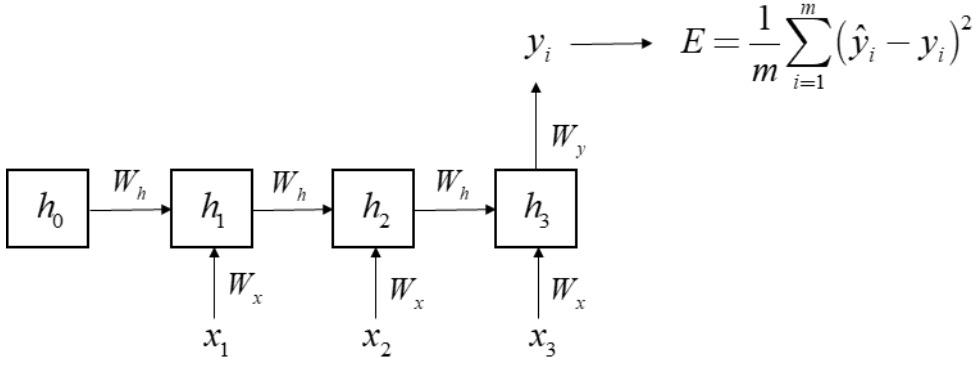


Рис. 2.4. Практичний приклад RNN моделі типу «many-to-one».

Джерело: розроблено автором.

Відповідні рівняння моделі (2.1):

$$E = \frac{1}{m}\sum_{i=1}^{m}(\hat{y}_i - y_i)^2 \text{ , } \textit{де "m" це кількість batches;} \qquad (2.1)$$

$$\hat{y}_i = W_y \cdot h_t$$

$$h_t = \tanh(z_t)$$

$$z_t = W_x \cdot x_t + W_h \cdot h_{t-1} \text{ , } t = \overline{1,n}$$

Ми повинні сказати кілька слів про побудову моделі, яка складається з рекурентного і лінійного прошарків. Коли призначення першого обумовлене його архітектурою, потреба в лінійному шарі може бути не такою очевидною на перший погляд.

Тут варто нагадати, що залежно від типу функції активації, яку ми використовуємо, значення стискаються в певному діапазоні. У випадку функції tanh це [-1, 1]. Цілком очевидно, що реальні серії, які ми збираємося прогнозувати, не лежать у цьому діапазоні. Тому нам потрібно додати лінійний прошарок, щоб перетворити вихідне значення рекурентного.

Слід також зазначити, що обидва прошарки побудовані без здвигів. Таким чином, мета зворотного поширення — знайти градієнти щодо ваг: W_y, W_x, W_h.

Знаходження градієнту для W_y (2.2):

$$\frac{\partial E}{\partial W_y} = \frac{\partial E}{\partial \hat{y}_i} \cdot \frac{\partial \hat{y}_i}{\partial W_y} = \frac{1}{m} 2 \ \hat{y}_i - y_i \ \cdot h_t \qquad (2.2)$$

Градієнт функції витрат відносно W_x, розраховується за формулою (2.3):

$$\frac{\partial E}{\partial W_x} = \frac{\partial E}{\partial \hat{y}_i} \frac{\partial \hat{y}_i}{\partial h_t} \frac{\partial h_t}{\partial z_t} \frac{\partial z_t}{\partial W_x} + \frac{\partial E}{\partial \hat{y}_i} \frac{\partial \hat{y}_i}{\partial h_t} \frac{\partial h_t}{\partial z_t} \frac{\partial z_t}{\partial h_{t-1}} \frac{\partial h_{t-1}}{\partial z_{t-1}} \frac{\partial z_{t-1}}{\partial W_x} + \frac{\partial E}{\partial \hat{y}_i} \frac{\partial \hat{y}_i}{\partial h_t} \frac{\partial h_t}{\partial z_t} \frac{\partial z_t}{\partial h_{t-1}} \frac{\partial h_{t-1}}{\partial z_{t-1}} \frac{\partial z_{t-1}}{\partial h_{t-2}} \frac{\partial h_{t-2}}{\partial z_{t-2}} \frac{\partial z_{t-2}}{\partial W_x} =$$

$$= \frac{1}{m} 2(\hat{y}_i - y_i) \cdot W_y \cdot (1 - \tanh^2(z_t)) \cdot x_t + \frac{1}{m} 2(\hat{y}_i - y_i) \cdot W_y \cdot (1 - \tanh^2(z_t)) \cdot W_h \cdot (1 - \tanh^2(z_{t-1})) \cdot x_{t-1} +$$

$$+ \frac{1}{m} 2(\hat{y}_i - y_i) \cdot W_y \cdot (1 - \tanh^2(z_t)) \cdot W_h \cdot (1 - \tanh^2(z_{t-1})) \cdot W_h \cdot (1 - \tanh^2(z_{t-2})) \cdot x_{t-2}$$

У випадку W_h градієнт дорівнює (2.4):

$$\frac{\partial E}{\partial W_h} = \frac{\partial E}{\partial \hat{y}_i} \frac{\partial \hat{y}_i}{\partial h_t} \frac{\partial h_t}{\partial z_t} \frac{\partial z_t}{\partial W_h} + \frac{\partial E}{\partial \hat{y}_i} \frac{\partial \hat{y}_i}{\partial h_t} \frac{\partial h_t}{\partial z_t} \frac{\partial z_t}{\partial h_{t-1}} \frac{\partial h_{t-1}}{\partial z_{t-1}} \frac{\partial z_{t-1}}{\partial W_h} + \frac{\partial E}{\partial \hat{y}_i} \frac{\partial \hat{y}_i}{\partial h_t} \frac{\partial h_t}{\partial z_t} \frac{\partial z_t}{\partial h_{t-1}} \frac{\partial h_{t-1}}{\partial z_{t-1}} \frac{\partial z_{t-1}}{\partial h_{t-2}} \frac{\partial h_{t-2}}{\partial z_{t-2}} \frac{\partial z_{t-2}}{\partial W_h} =$$

$$= \frac{1}{m} 2(\hat{y}_i - y_i) \cdot W_y \cdot (1 - \tanh^2(z_t)) \cdot h_{t-1} + \frac{1}{m} 2(\hat{y}_i - y_i) \cdot W_y \cdot (1 - \tanh^2(z_t)) \cdot W_h \cdot (1 - \tanh^2(z_{t-1})) \cdot h_{t-2} +$$

$$+ \frac{1}{m} 2(\hat{y}_i - y_i) \cdot W_y \cdot (1 - \tanh^2(z_t)) \cdot W_h \cdot (1 - \tanh^2(z_{t-1})) \cdot W_h \cdot (1 - \tanh^2(z_{t-2})) \cdot h_{t-3}$$

Принцип такий самий, як ми розглянули в розділі 1.3. Єдина відмінність полягає в тому, що цього разу $\hat{y}_i$ залежить як від значень прихованого стану всіх періодів.

Таким чином, щоб обчислюючи градієнт функції втрат щодо W_x, ми повинні взяти часткові похідні для всіх кроків і підсумувати їх. Те ж саме стосується W_x. Єдиним винятком є ваг W_y, складність обчислення якого не залежить від довжини послідовності і пояснюється типом RNN, який ми тут використовуємо.

Використовуючи дані обрахунки був створений блок коду, мета якого полягала в порівняння результатів пошуку градієнтів із автоматизованим підходом в пакеті «pytorch». Згідно отриманих оцінок, результати повністю співпали.

Однак цей тип архітектури не здатний зберігати довгострокові залежності. Це явище також відоме як проблема зникаючого градієнта [24]. Після виконання деяких обчислень ми бачимо, як градієнти стають меншими, по мірі того, як ми заглиблюємося в модель [25]. Дане явище частково пояснюється функціями активації та великою кількістю прошарків. Градієнти стають настільки малими, що параметри на ранніх прошарках не оновлюються [26].

Це легше зрозуміти на прикладі інших типів мереж, таких як CNN. Але ідея полягає в наступному: ми не можемо ефективно зберігати довгострокові залежності, щоб використовувати їх у наших прогнозах. Саме тому виникла необхідність у появі нових, більш складних архітектур, здатних впоратися з цим аспектом. Ними стали LSTM і GRU.

2.2 Особливості будови блоку пам’яті LSTM моделі

Запропонована в 1997 році Зеппом Хохрайтером і Юргеном Шмідхубером, LSTM, також відома як модель довгої-короткочасної пам’яті, була першою архітектурою, яка здобула високу популярність у науковому співтоваристві після RNN і зарекомендувала себе в ряді областей [27].

В основному, коли ми працюємо з даними часових рядів, нам потрібна лише частина відомої інформації для прогнозування. Припустимо, що для прогнозування періоду «t» нам потрібно спиратися на значення попередніх періодів (не обов’язково послідовних), і те саме стосується періодів «t+1», «t+2» тощо. Щоб забезпечити високу ефективність моделювання мережі, нам потрібно визначити набір відповідних даних, які будуть використовуватися та оновлюватися з часом. Для забезпечення цього процесу вбудований спеціальний механізм: «gates», кожен з яких має своє призначення.

На рис 2.5 наведена архітектура блоку пам’яті LSTM моделі:

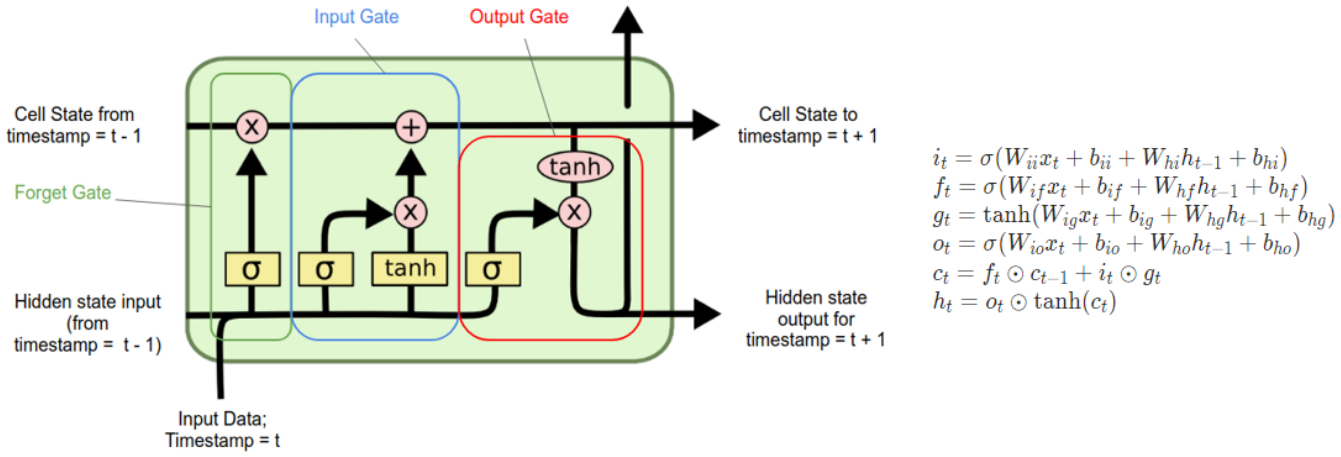


Рис. 2.5. Архітектура блоку пам’яті LSTM моделі.

Джерело: розроблено автором на основі [28]

Стан у LSTM - це пара векторів: c_t та h_t:

- c_t - осередок пам'яті, що містить інформацію попередніх періодів;
- h_t - прихований стан у час "t"; зашифрована інформація осередку пам'яті c_t;

Для того, щоб зрозуміти принцип їх обрахунку, необхідно детально зупинитися на ролі кожного з вентилів та їх взаємодії [28].

Forget gate f_t - визначає, що нам потрібно стерти з комірки пам'яті, на основі поточного входу x_t і прихованого стану h_{t-1}, оціненого на попередньому кроці. Функція активації, що використовується в цьому рівнянні - сигмоїда, з якою ми вже ознайомилися у розділі 1.3. Якщо значення вентилю в якомусь випадку дорівнює нулю, то відповідне значення комірки пам’яті стирається (2.5).

$$f_t = \sigma(W_{if} x_t + b_{if} + W_{hf} h_{t-1} + b_{hf}) \quad (2.5)$$

Input gate i_t - визначає величину нової інформації, яку ми додаємо в комірку пам'яті. Таким чином, для початку необхідно здійснити розрахунок відповідного кандидата g_t (2.6). Отриманий результат потім фільтрується вхідним вентилем i_t (2.7), принцип роботи якого схожий на той, що був описаний раніше.

$$g_t = \tanh(W_{ig} x_t + b_{ig} + W_{hg} h_{t-1} + b_{hg}) \quad (2.6)$$

$$i_t = \sigma(W_{ii} x_t + b_{ii} + W_{hi} h_{t-1} + b_{hi}) \quad (2.7)$$

На цьому етапі у нас є все для оновлення значення комірки пам'яті c_t. Використовуючи добуток Адамара, описаний у рівнянні (2.8), ми отримуємо нову комірку пам’яті, яка буде використовуватися в подальших ітераціях.

$$c_t = f_t \odot c_{t-1} + i_t \odot g_t \quad (2.8)$$

Але перш ніж закінчити ітерацію, нам потрібно знайти вихідне значення поточного шару ($y_t = h_t$). Для цього, ми використовуємо останній, вихідний вентиль.

Output gate o_t - фільтр для осередоку пам'яті, що визначає вихідне значення поточного шару. На початку опису ми згадували, що стан в LSTM є парою векторів. Ми також зазначили, що h_t - це зашифрована інформація комірки пам'яті c_t. На даний момент ми можемо легко переконатися в цьому, виходячи з рівняння (2.9):

$$h_t = o_t \odot \tanh\ c_t \quad (2.9)$$

Як ми бачимо, осередок пам’яті, отриманий на поточному кроці, обертається у функцію tanh, а потім фільтрується вихідним вентилем. Саме це ми мали на увазі під «зашифрованою» інформація осередку пам'яті c_t.

Результатом розрахунку є два стани LSTM: нова комірка пам’яті c_t та її зашифрована версія h_t. Останнє також є вихідним значенням поточного шару. Обидва будуть використовуватися та оновлюватися в подальших ітераціях [29].

Після закінчення теоретичної частини, корисним буде закріпити нові знання на практиці. Тому була створена спеціальна функція: «LSTM_cell», для порівняння її результатів із автоматизованою версією. Слід зазначити, що основна ідея практичної частини полягає в тому, щоб надати чітке розуміння архітектур моделей та принципів обрахунку. Код містить детальний опис, щоб забезпечити інтуїтивне розуміння його побудови.

Останнє, на що ми хочемо звернути увагу, це деякі архітектурні проблеми LSTM:

Коли ми оцінюємо вентилі, ми враховуємо їх не на основі повної інформації c_t, а на основі відфільтрованої версії пам’яті h_t. Тут є концептуальна проблема [30]. Навіть якщо прийняти це як належне, стає дивним, що у випадку вихідного вентиля o_t ми не враховуємо нову комірку пам’яті c_t, розраховану на поточному кроці.

Ця проблема була помічена науковою спільнотою. Тому через кілька років після публікації основної статті була запропонована нова, оновлена версія LSTM: «LSTM with peephole connections».

У новій моделі осередок пам'яті c_{t-1} враховується для обрахунки описаних нами структурних елементів. Також слід звернути увагу, що вихідний елемент o_t тепер містить осередок пам'яті c_t, розрахований на поточному кроці, що фактично вирішує описану вище проблему.

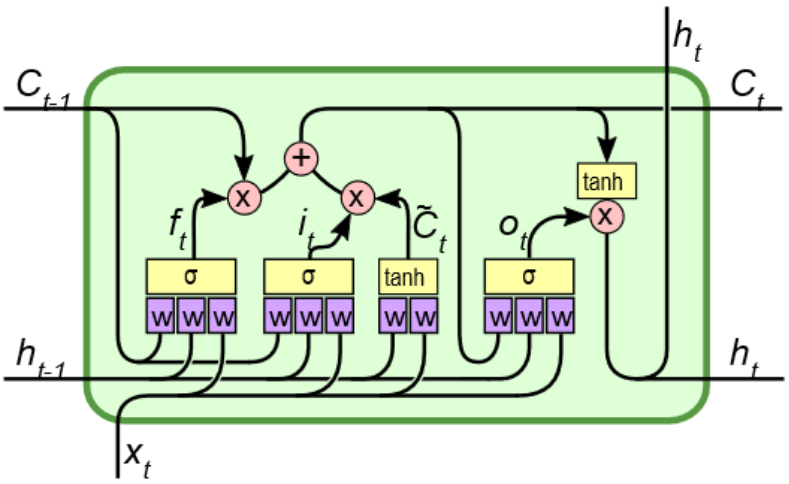


Рис. 2.6. Архітектура блоку пам’яті Peephole LSTM.

Джерело: розроблено автором на основі [30]

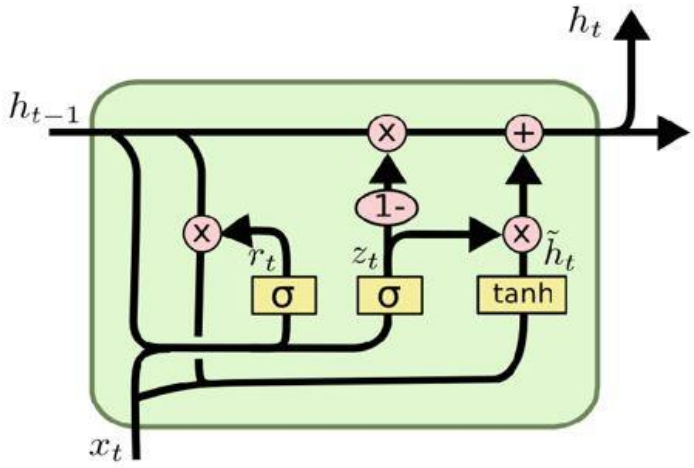
Нова модель продемонструвала непогані результати та стала певним узагальненням початкової версії. Проте базова LSTM залишається актуальною й досі, адже потребує меншої кількості параметрів. І хоча, сучасні пакети нейронних мереж пропонують обидві версії, у більшості випадків, результати є доволі схожими.

2.3 Збереження пам'яті в моделі GRU.

Довгий час LSTM залишався популярним вибором у контексті моделювання часових рядів. Неодноразово наукове співтовариство намагалося перевершити результати цієї моделі, шукаючи нові архітектури. Знайти потрібну конструкцію – надзвичайно складне завдання, оскільки нова модель повинна добре працювати не в окремому прикладі, а в різних сферах.

Лише в 2014 році нова модель була представлена. "GRU", або "Gated Reccurent Unit" - алгоритм, який дуже схожий на той, який ми використовували в попередньому розділі. Даний вид архітектури був мотивований самою LSTM, що зазначено в оригінальній статті [31].

Явними перевагами GRU є зменшення кількості вентилів, спрощення математики обчислень моделі та злиття комірки пам’яті та її зашифрованої форми – тепер вони є одним цілим. Розглянемо її архітектуру дещо детальніше (рис 2.7)



$$r_t = \sigma(W_{ir}x_t + b_{ir} + W_{hr}h_{(t-1)} + b_{hr})$$
$$z_t = \sigma(W_{iz}x_t + b_{iz} + W_{hz}h_{(t-1)} + b_{hz})$$
$$n_t = \tanh(W_{in}x_t + b_{in} + r_t * (W_{hn}h_{(t-1)} + b_{hn}))$$
$$h_t = (1 - z_t) * n_t + z_t * h_{(t-1)}$$

Рис. 2.7. Архітектура блоку пам’яті GRU моделі.

Джерело: розроблено автором на основі [31]

Reset gate r_t - вибирає релевантну інформацію з попереднього стану h_{t-1}, враховуючи при цьому нову інформацію на вході - x_t. Через те, що LSTM і GRU здатні зберігати довготривалі залежності, не вся інформація, що зберігається в h_{t-1}, необхідна на певному кроці (2.10).

$$r_t = \sigma(W_{ir}x_t + b_{ir} + W_{hr}h_{t-1} + b_{hr}) \quad (2.10)$$

Для кращого розуміння, можна привести аналогію з сезонною складовою моделі Sarima. Це означає, що ми спираємося на попередні спостереження, які не обов'язково є послідовними. Ідея цього вентилю полягає у визначенні частки інформації щодо минулих значень ряду, яка будете корисною для поточного кроку.

Update gate z_t - контролює, скільки інформації з попереднього стану h_{t-1} буде перенесено в новий h_t (2.11). Як ми бачимо з наведених нижче рівнянь, для обчислення кандидата поточного стану (2.12), ми використовуємо: вхідне значення x_t, попередній стан h_{t-1} і вентиль r_t, який приймає значення [0,1].

$$z_t = \sigma(W_{iz}x_t + b_{iz} + W_{hz}h_{t-1} + b_{hz}) \quad (2.11)$$

$$n_t = \tanh(W_{in}x_t + b_{in} + r_t \cdot (W_{hn}h_{t-1} + b_{hn})) \quad (2.12)$$

Очевидно, що кандидат містить як нову, так і стару інформацію. Таким чином, наша мета полягає в тому, щоб знайти компроміс, частину h_{t-1} і n_t, яка буде вбудована в кінцевий стан h_t. Це власне те, що насправді робить шлюз оновлення (2.13).

$$h_t = (1 - z_t) \cdot \mathrm{n}_t + \mathrm{z}_t \cdot \mathrm{h}_{t-1} \quad (2.13)$$

Ця архітектура дозволяє нам містити та оновлювати відповідну інформацію, фіксувати короткострокові та довгострокові залежності та забезпечувати бажаний результат [32]. Як і LSTM, вона також вирішує проблему зникаючого градієнта, але має менше параметрів, що робить її обчислення більш ефективним. На сьогоднішній день GRU модель стала справжнім конкурентом LSTM [33]. На практиці аналітики вважають за краще використовувати обидві моделі, оскільки в залежності від предмета аналізу продуктивність моделей може відрізнятися. Незважаючи на це, вони вважаються стабільними і однозначно заслуговують на увагу наукової спільноти.

Прогнозування даних часових рядів є складним завданням. Незважаючи на те, що спостереження збираються через регулярні проміжки часу, у випадку прогнозування фондового ринку, де все може змінитися за частку секунди. Основні алгоритми, такі як моделі Arima, Sarima, Arch і Garch, не можуть ефективно працювати у випадку волатильності, яка є невідмінною складовою біржі.

Ось чому зростає інтерес до потужніших і складніших з точки зору математики алгоритмів. Особлива увага приділяється області глибинного навчання, зокрема

рекурентним нейронним мережам. На відміну від своїх попередників, вони багатозадачні і легко справляються з тим, що виходить за межі можливостей традиційних моделей.

Vanilla RNN не відноситься до тих моделей, які зазвичай використовуються для цілей прогнозування. Основною причиною цього є неможливість зберігати довгострокові залежності та проблема зникнення градієнтів — випадок, коли параметри не тренуються, або їх оновлення не є значущим. Але це чудовий навчальний матеріал і відправна точка для більш складних рекурентних нейронних мереж.

Сучасні архітектури, такі як GRU та LSTM, чудово підходять для прогнозування фінансових часових рядів, зокрема акцій компаній та інших цінних паперів. Обидві моделі виявилися ефективними і таким чином виправдали свої обчислювальні витрати та завоювали увагу спільності.

РОЗДІЛ 3. МОДЕЛЮВАННЯ ЧАСОВОГО РЯДУ ТА АНАЛІЗ РЕЗУЛЬТАТІВ

3.1 Аналіз часового ряду методом нейронних мереж.

Цей розділ присвячений практичній реалізації рекурентних нейронних мереж, які ми розглянули у другій главі. Беручи до уваги архітектуру моделей та аспекти обробки даних, які необхідно виконати перед передачею даних у модель, було створено ряд класів і кастомних функцій.

Перш за все, нам потрібно завантажити наші дані. Для цього ми використали такий популярний ресурс, як Yahoo Finance, який надає актуальну інформацію про більшість тикерів на фінансовому ринку. Часовий ряд, який нас цікавить, — це акції компанії MSFT з період з 17.03.2018 по 25.01.2020. Причина, чому ми обрали ці спостереження замість того, щоб взяти щось нове, пов'язана з відомою пандемією, яка спричинила рецесію ринку. Таким чином, моделювання таких нестабільних рядів було б малоефективним.

Date	Open	High	Low	Close	Adj Close	Volume
2018-03-19	93.739998	93.900002	92.110001	92.889999	88.527260	33344100
2018-03-20	93.050003	93.769997	93.000000	93.129997	88.755997	23075200
2018-03-21	92.930000	94.050003	92.209999	92.480003	88.136536	24457100
2018-03-22	91.269997	91.750000	89.660004	89.790001	85.572868	38604700
2018-03-23	89.500000	90.459999	87.080002	87.180000	83.085457	44068900

Рис. 3.1. Часовий ряд компанії MSFT з 2018-03-17 до 2020-01-25.

Джерело: розроблено автором

Серія, яку ми хочемо проаналізувати, міститься в стовпці "Close". Загальновідомо, що масштаб може значно погіршити чи, навпаки, покращити збіжність. Тому нам потрібно стиснути наш ряд до якогось певного діапазону. Але перш ніж ми це зробимо, візьмемо логарифм ряду, для того щоб згладити значення.

Існує багато способів стиснути наші дані до певного діапазону, але найпоширенішими підходами є нормалізація та стандартизація [34]. На мою думку, другий варіант є єдиним правильним, у випадку моделювання часових рядів.

Коли ми працюємо із зображеннями, пікселі яких знаходяться в діапазоні від 0 до 255, ми можемо застосувати нормалізацію, будучи впевненими, що значення не вийдуть за ці межі. Тоді як у випадку моделювання часових рядів ми не можемо це стверджувати. Причина, по якій ми використовуємо нормалізацію замість інших методів, як стандартизація, полягає в тому, що результати для таких рядів були дещо гіршими.

Незважаючи на це, я все ще вважаю, що стандартизація, яка не передбачає наявності чітких меж, є єдиним правильним вибором тут. Можливо, потрібно трохи більше часу, щоб знайти пояснення цим результатам.

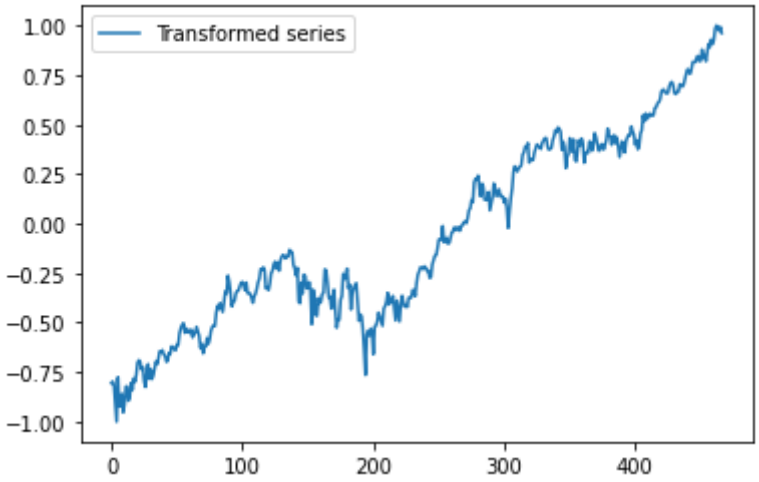


Рис. 3.2. Графічне відображення ряду «Close» після трансформації.
Джерело: розроблено автором

Як видно з малюнку вище, наша серія була трансформована та лежить у діапазоні [-1;1]. Слід також зазначити, що нормалізація передбачає стиснення ряду в межах від [0;1], проте використовуючи узагальнену версію (3.1), ми можемо задати власний проміжок, відносно якого будуть відбуватися перетворення.

$$X_{norm\_adj} = \frac{x - x_{\min}}{x_{\max} - x_{\min}} \cdot (max - min) + min \qquad (3.1)$$

де $x_{\min}$, $x_{\max}$ - найменше та найбільше значення вибірки, а «max», «min» - межі трансформації.

Наступний етап пов’язаний із форматом зберігання даних. У розділі 2 ми частково висвітлили ідею «batches» та їх структуру. Проте, процес їх формування був

описаний доволі абстрактно. Є три основні характеристики, які ми повинні враховувати:

1) кількість спостережень часового ряду;
2) кількість спостережень, які ми щоразу передаємо в рекурентний шар (input_size);
3) кількість послідовностей (2), на основі яких, ми хочемо, щоб наша модель навчалася (sequence_length);

Метою функції «data_split», є створення відповідних груп послідовностей із урахування зазначених вище параметрів. (Додаток С). У нашому дослідженні, ми бажаємо навчити моделі, на основі 10 сетів послідовностей (sequence_length), зо містять 2 спостереження в кожний момент часу (input_size).

Сформувавши новий формат даних, ми можемо перейти до поділу «batches» на тренувальну та тестову вибірки. Ця процедура необхідна для перевірки ефективності моделювання ряду, на основі нових даних, після оцінки моделі.

Виконавши всі необхідні етапи, ми можемо перейти до оцінки моделей.

Нейронні мережі — це той клас моделей, які потребують величезної кількості даних, для навчання. У нашому дослідженні ми також хочемо порівняти наші результати з традиційними моделями, такими як Arima, Arch, Garch, тощо. Цим, частково пояснюється часовий проміжок обраних даних – 467 спостереження. Тому було б несправедливо надавати нейронним мережам перевагу у вигляді більшої кількості даних. Щоб вирішити цю проблему, ми можемо збільшити кількість ітерацій у ході тренування.

При цьому потрібно пам'ятати про перенавчання - випадок, коли мережі запам'ятовують зайву інформацію. Вони все ще прагнуть знайти і запам’ятати закономірності в даних, але мета поступово змінюється: замість пошуку патернів для моделювання часового ряду, модель намагатиметься запам’ятати саму серію. Це стає очевидним, якщо порівняти точність на основі навчальної й тестової наборів.

У нашому випадку, всі 3 моделі мають однакові характеристики:

- «input_size», довжина послідовності в кожний момент часу - 2;

- «hidden_size», кількість прихованих станів - 20;
- «num_layers», кількість прошарків - 2;
- «output_size» , кількість значень, що повертає мережа (many-to-one) - 1;
- «num_epochs», або кількість ітерацій - 200;

У якості функції витрат був використаний «MSELoss», а методом оптимізації був обраний алгоритм Адама з рівнем навчання в 0.01.

Рис. 3.3. Відображення ряду «Close» та оцінок моделі RNN на тренувальній вибірці. Джерело: розроблено автором

Провівши необхідні розрахунки для RNN моделі, ми можемо візуалізувати наші результати. На рис 3.3, ми можемо оцінити ефективність моделювання RNN для тренувальної вибірки. Значення прогнозів досить близькі до реальних. Навіть на проміжках зі значною волатильністю точність залишається високою.

Для дослідження динаміки тренування моделі, доречним буде відобразить зміну значень функції втрат (рис 3.4). Цей графік забезпечує інтуїтивне розуміння етапів навчання моделі і може допомогти нам визначити оптимальну кількість ітерацій. Основна причина, чому ми робимо відобразили ці результати, полягає в тому, щоб переконатися, що останні спостереження нагадують пряму горизонтальну лінію. В

іншому випадку, наявність помітного схилу, може означати, що тренування було незавершене.

Незважаючи на результати навчання, говорити про продуктивність моделі ще зарано. Щоб розвіяти будь-які сумніви, нам потрібно обчислити прогнози для тестового набору даних, здійснити порівняння за визначеним критерієм (RMSE) і візуалізувати результати.

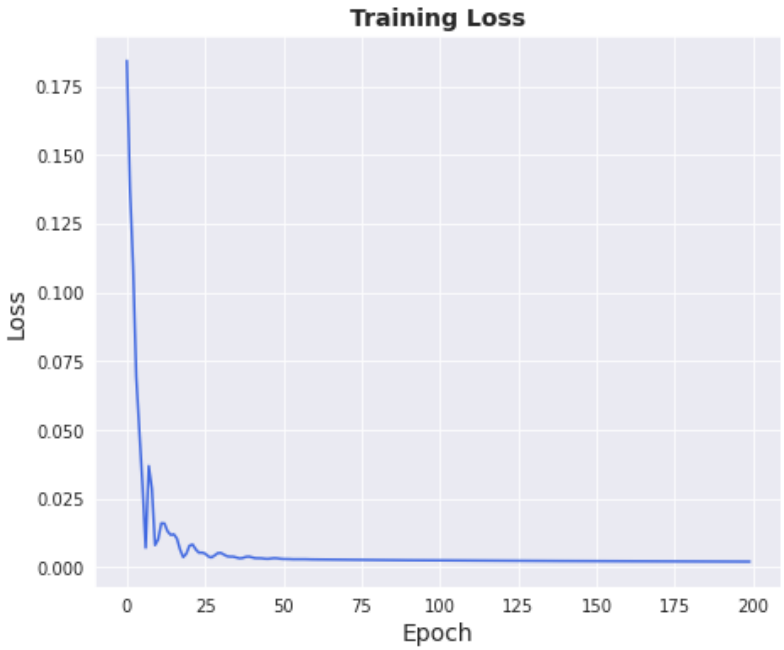


Рис. 3.4. Динаміка зміни значення функції втрат RNN під час тренування.
Джерело: розроблено автором

Відповідного до отриманих результатів, можна констатувати, що продуктивність моделі досить висока. Ми також можемо помітити, що прогнози віддаляються від реальних значень, починаючи з точки, де ряд стає зростаючим. Ця концепція також відома як екстраполяція. Проте, відповідь на питання, чому це слід вважати проблемою, досить проста: очікуємо від моделі прогнозів для даних, що виходять за межі діапазону значень, на яких вона була навчена.

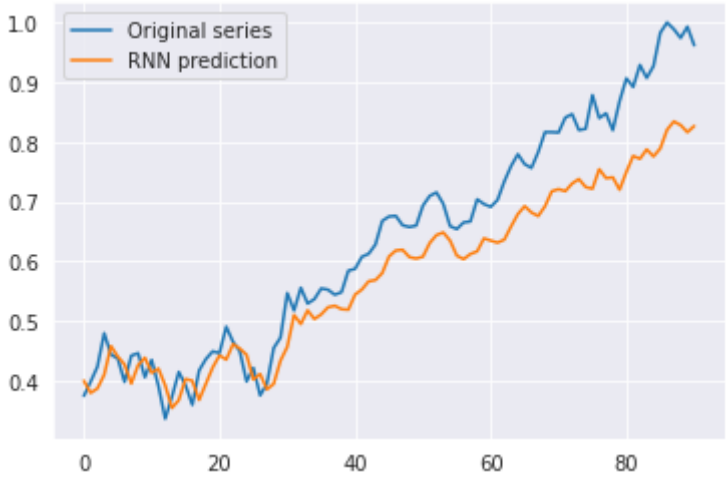


Рис. 3.5. Порівняння оцінок RNN для тестової вибірки.

Джерело: розроблено автором

Якщо порівняти серії за місцем поділу на тренувальну та тестову вибірки, то помітимо, що їх значення доволі схожі. Цим обумовлена точність для перших 20 спостережень у тестовому наборі. Ось чому в моделюванні часових рядів ми вважаємо за краще проводити перетворення, як диференціювання. Той факт, що RNN здатна так добре прогнозувати дані, без подібних перетворень, дійсно вражає.

Використовуючи RMSE як основний критерій, ми обчислили оцінки як для навчального, так і для тестового набору. Відповідні значення 0,046 і 0,081 відповідно. Таким чином, можна стверджувати, що результати для навчального набору вдвічі кращі за результати тестового. Беручи до уваги викладену вище концепцію, можна відзначити, що прогнози все ще досить непогані.

У цьому розділі ми також збираємося оцінити модель LSTM, використовуючи ті ж дані та параметри, визначені раніше. Критерій та алгоритм оптимізації не змінюються. Те ж саме стосується моделі GRU. Таким чином, ми не будемо зупинятися на описі процесів навчання моделі, а представимо результати аналізу та порівняємо їх між собою.

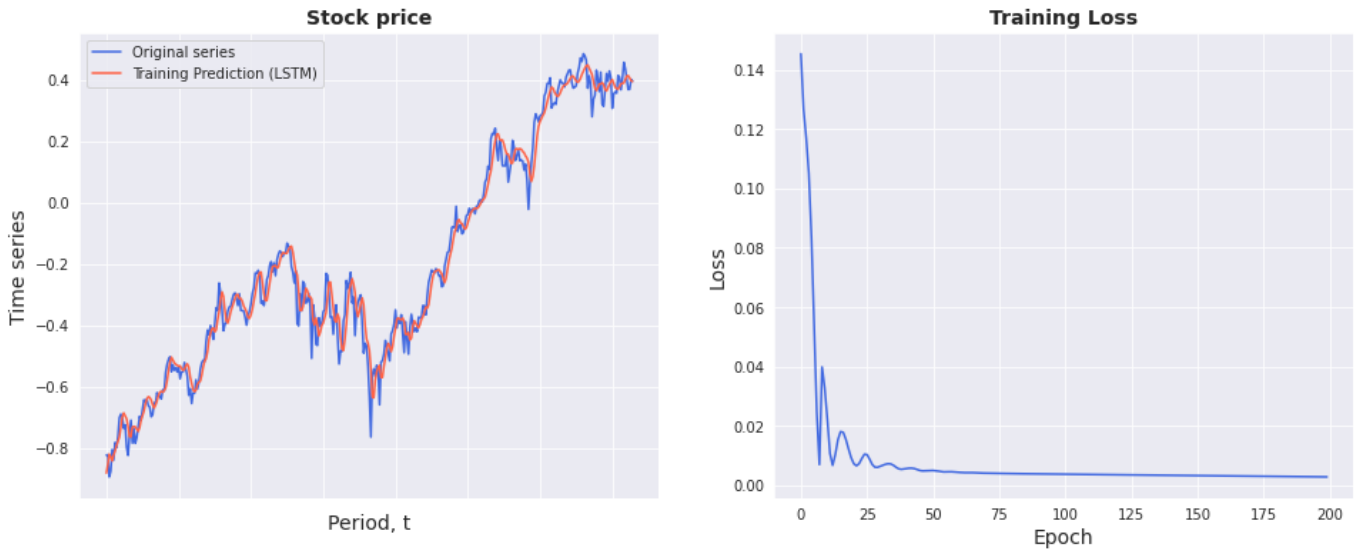


Рис. 3.6. оцінки та функція витрат LSTM моделі на тренувальній вибірці.

Джерело: розроблено автором

Як видно з першого графіка, у випадку моделі LSTM лінія прогнозу є більш плавною порівняно з моделлю RNN. Точність явно нижча, особливо в кінці серії. Щоб знайти відповідь на те, що стало причиною цього, нам потрібно вникнути в проблематику навчання мереж.

Розуміння архітектури нейронних мереж, процесів прямого і зворотного поширення є великою перевагою. У той же час дослідження, що стосуються взаємозв’язку між параметрами, які ми встановлюємо, як : кількість вхідних даних серії, прихованих станів і осередків пам'яті (у випадку LSTM), форма «batches», кількість ітерацій - є не менш важливими.

Вчені, які висвітлюють цю область аналізу, намагаються дослідити цей взаємозв'язок і його вплив на продуктивність моделі. Крім того, існує пара фреймворків, які містять коефіцієнти, отримані в результаті таких досліджень для різних моделей. На жаль, статті, опубліковані на відомих ресурсах як «arxiv.org», представлені без коду. Єдине, на що ми можемо покладатися, це на наше розуміння того, як проводився аналіз.

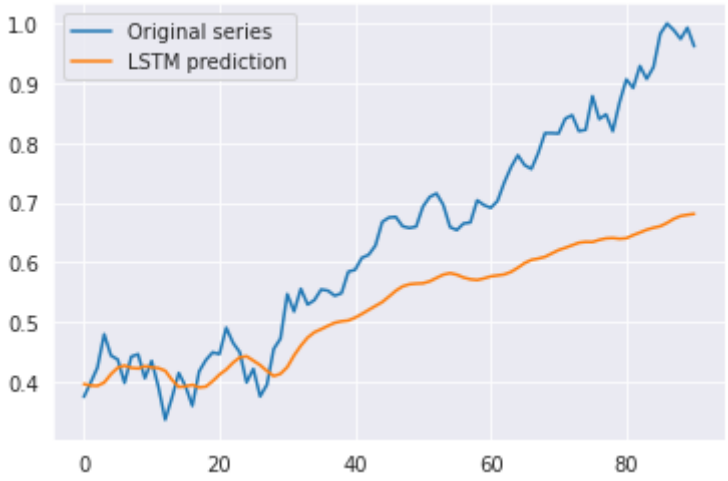


Рис. 3.7. оцінки та функція витрат LSTM моделі на тренувальній вибірці.

Джерело: розроблено автором

Відповідно до графіка вище, продуктивність моделі на тестовому наборі значно гірша в порівнянні з RNN. Такий же висновок можна зробити з результатів оцінок RMSE: 0.0528 та 0.1477 для тренувальної та тестової вибірки, відповідно.

Результати для моделі GRU (рис 3.8) подібні до lstm. Прогнозована лінія є більш гладкою порівняно з моделлю RNN, але ближчою до дійсних значень, ніж у випадку lstm моделі (рис 3.9). Те саме стосується тестового набору та оцінки RMSE.

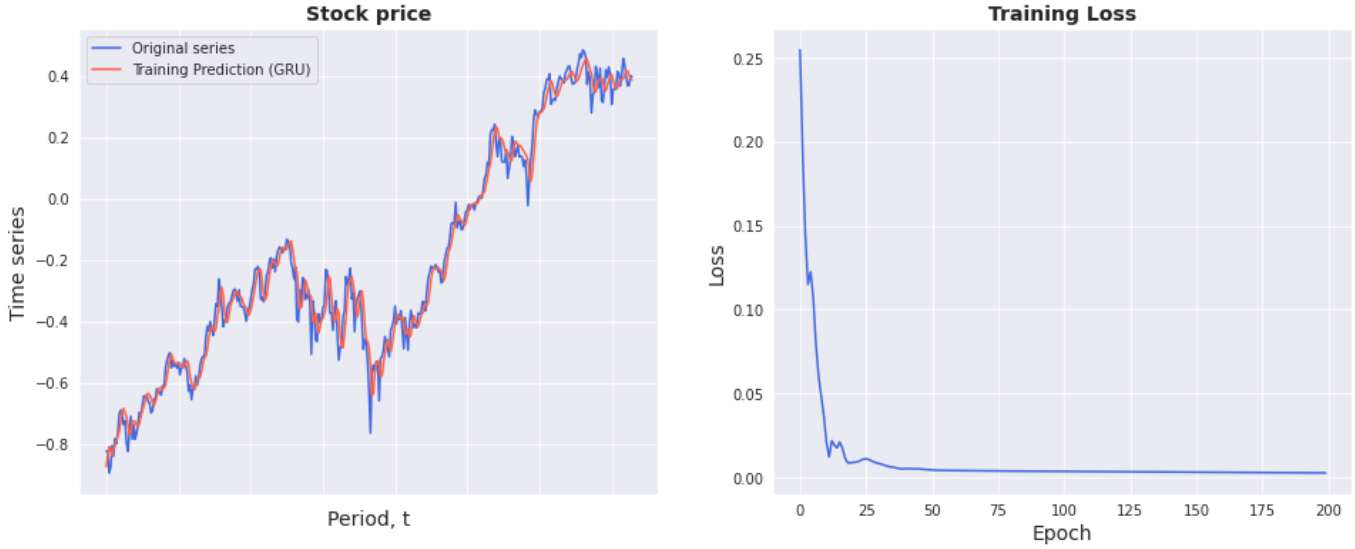


Рис. 3.8. оцінки та функція витрат GRU моделі на тренувальній вибірці.

Джерело: розроблено автором.

Обидві моделі: GRU і LSTM є потужнішими архітектурами рекурентних нейронних мереж. У цьому аналізі я спробував безліч перетворень даних у поєднанні з різними параметрами, пов'язаними зі структурою даних і самими моделями. Але результати не були настільки значущими, щоб покращити продуктивність моделі. Даний аспект буде докладніше висвітлений у висновках для цього розділу.

Оцінки RMSE для GRU моделі склали: 0.0528 та 0.1278 для тренувальної та тестової вибірки, відповідно.

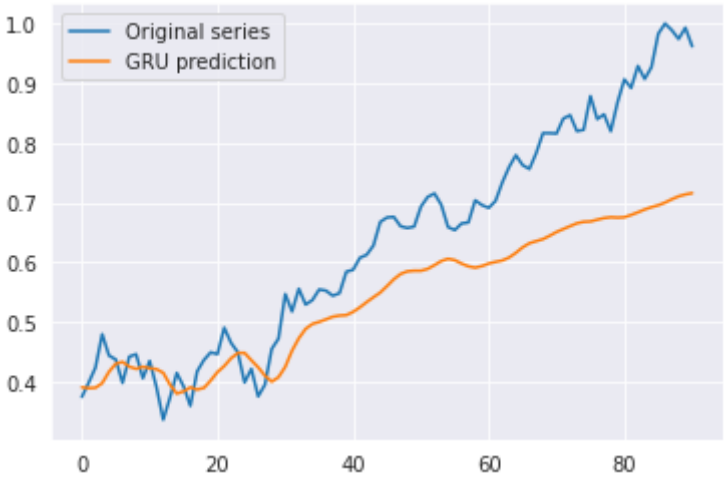


Рис. 3.9. оцінки та функція витрат GRU моделі на тренувальній вибірці.
Джерело: розроблено автором

Точний прогноз — це те, чого ми прагнемо, оцінюючи моделі. Зі змісту попередніх розділів стає цілком очевидним, що прогноз у випадку рекурентних нейронних мереж, буде поступатися в точності порівняно з традиційними моделями з розділу 3.2. Незважаючи на це, нам все одно необхідно провести передбачення ряду на один період вперед (t+1).

Для цього була створена функція «batch_for_pred». Ідея та ж, що й у функції, яку ми використовували для поділу даних на «batches», описаної на початку. Але цього разу віна повертає групу послідовностей, необхідну для здійснення прогнозу.

Після передачі даних у моделі, описанні вище, ми отримали наступні прогнози для RNN, LSTM та GRU. Щоб порівняти їх із значенням вихідного ряду, нам потрібно виконати перетворення, зворотні до тих, які ми робили в розділі обробки даних (табл. 3.1).

Табл.3.1 Порівняння значень прогнозу оцінених моделей на наступний період.

	Прогнози моделей		Фактичне значення
	До зворотного перетворення	Після	
RNN	0.8136	157.2667	167.54
LSTM	0.6812	150.6385	
GRU	0.7169	152.3969	

Джерело: розрублено автором.

Отож, слід підсумувати результати блоку.

Прогнозування часових рядів – надзвичайно складна річ. Нині є багато алгоритмів машинного навчання, які можуть виявляти закономірності та фіксувати ці залежності всередині серії. Вони відрізняються принципом дії, складністю архітектур, а також вимагають різних видів обробки даних. Для прогнозування часових рядів також широко використовуються такі алгоритми глибокого навчання, як RNN, LSTM і GRU. Вони потужніші в порівнянні з традиційними моделями, такими як Arima, Arch і Garch, але є дещо складнішими.

У рамках цього дослідження ми провели основні операції обробки даних та оцінили три рекурентні нейронні мережі. Детально розглянули отримані результати та фактори, які на них безпосередньо впливають. Також було проведено багато роботи, яка залишається в тіні цього дослідження.

Готуючись до цього аналізу, я випробував різні методи перетворення даних на різних серіях, включаючи ті, що використовуються для традиційних моделей. Один з таких експериментів представлений нижче.

Для тестування була обрана більш волатильна серія, а саме ряд тикеру «AABA» з 03.01.2006 по 29.12.2017. Пов'язані з ним перетворення включають:

1) логарифмування початкового ряду;

2) розрахунок прибутку за формулою: $x_t - x_{t-1} \ / \ x_{t-1}$ на основі (1);

Рис. 3.10. Оцінки LSTM для трансформованого ряду «AABA».

Джерело: розроблено автором.

Диференціювання також можна використовувати на 2-му кроці. Але тут я хотів уникнути проблеми масштабу. Таким чином, кінцевий ряд — це логарифмований прибуток, виражений у відсотках. На перший погляд здавалося, що модель зовсім не тренується. Результати прогнозу нагадували пряму горизонтальну лінію. Але якщо ми помножимо цей ряд на деяке значення, щоб збільшити його величину, ми отримаємо наступне (рис 3.11).

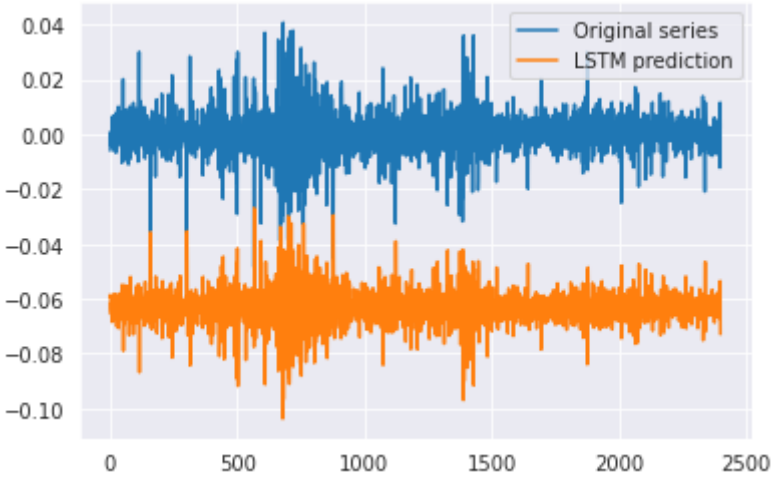


Рис. 3.11. Збільшений масштаб оцінок моделі.

Джерело: розроблено автором.

Збільшуючи значення прогнозу, ми бачимо, що він набуває сенсу. Насправді це не пряма лінія, а серія, яка дуже близька до оригінальної. Мені не вдалося знайти пояснення цьому явищу. Принаймні зараз.

Повертаючись до аналізу, представленого в цьому розділі, ми повинні зазначити, що моделі навчалися на оригінальній серії, яка була певним чином трансформована. У випадку з традиційними моделями ми хочемо ряд стаціонарним. Ці ряди мають такі характеристики, як: постійне середнє значення, дисперсію та коваріація не залежать від часу. Якщо ми спробуємо оцінити їх на основі ряду, що ми використовували для нейронних мереж, результати будуть жахливими. Тому я схильний вважати, що отримані в рамках розділу результати, є досить непогані.

3.2 Використання інших алгоритмів машинного навчання.

У попередньому розділі, ми провели оцінку та моделювання часового ряду методами нейронних мереж. В даному ми продовжимо наш аналіз, використовуючи традиційні методи прогнозування, для порівняння результатів. Слід зазначити, що часовий ряд, в обох випадках є однаковим.

Першим кроком у побудові ARMA-GARCH/ARCH моделі є визначення оптимальної ARIMA (додаток D).

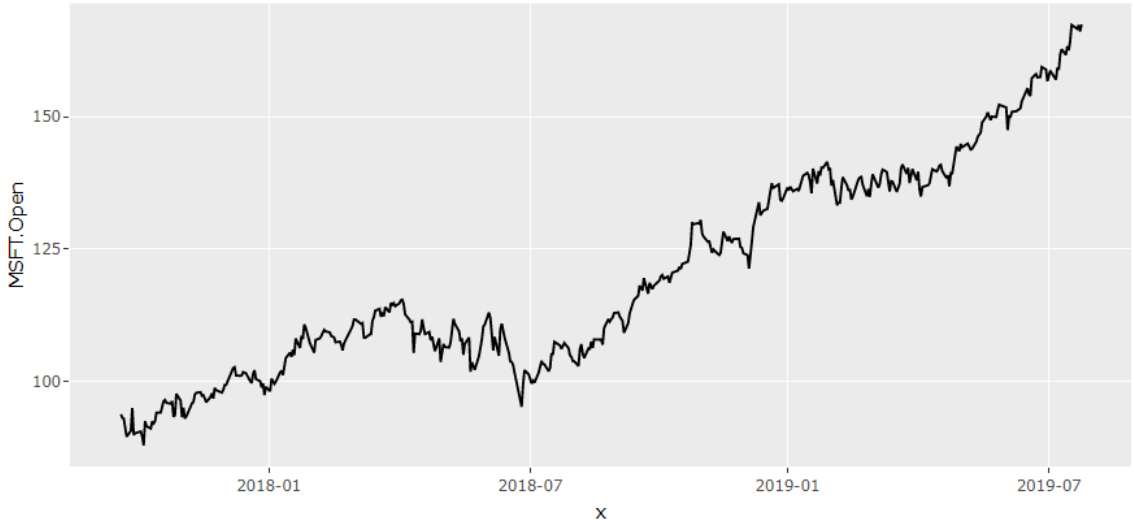


Рис. 3.12. Часовий ряд «Close» компанії MSFT з 2018-03-17 до 2020-01-25.

Джерело: розроблено автором.

Доволі поширеною є практика перетворення ряду цін шляхом логарифмування. Логарифмування дозволяє зменшити дисперсію ряду, що позитивно відобразиться на оцінці моделей. Таким чином, ми отримуємо ряд відсоткових змін акцій [35].

Як ми можемо бачити, ряд не є стаціонарними. Для впевненості перевіримо це за допомогою тесту Діккі-Фуллера [36] (рис. 3.13.):

```
        Augmented Dickey-Fuller Test

data:  TSeries
Dickey-Fuller = -1.811, Lag order = 7, p-value = 0.6579
alternative hypothesis: stationary
```

Рис. 3.13. Тест Діккі-Фуллера для логарифмованого ряду.

Джерело: розроблено автором.

Значення p-value більше за 0.05 , що свідчить про присутність нестаціонарності. Спробуємо позбутися її шляхом диференціювання. Запровадивши диференціювання першого порядку, ми знову перевіряємо ряд на стаціонарність (рис. 3.14.).

```
        Augmented Dickey-Fuller Test

data:  mData$Tseries
Dickey-Fuller = -8.5678, Lag order = 7, p-value = 0.01
alternative hypothesis: stationary
```

Рис. 3.14. Тест Діккі-Фуллера для диференційованого ряду.

Джерело: розроблено автором.

Виходячи з зазначених результатів (p-value<0,05), ряд став стаціонарним. Графічне зображення ряду відображене нижче (рис. 3.15.).

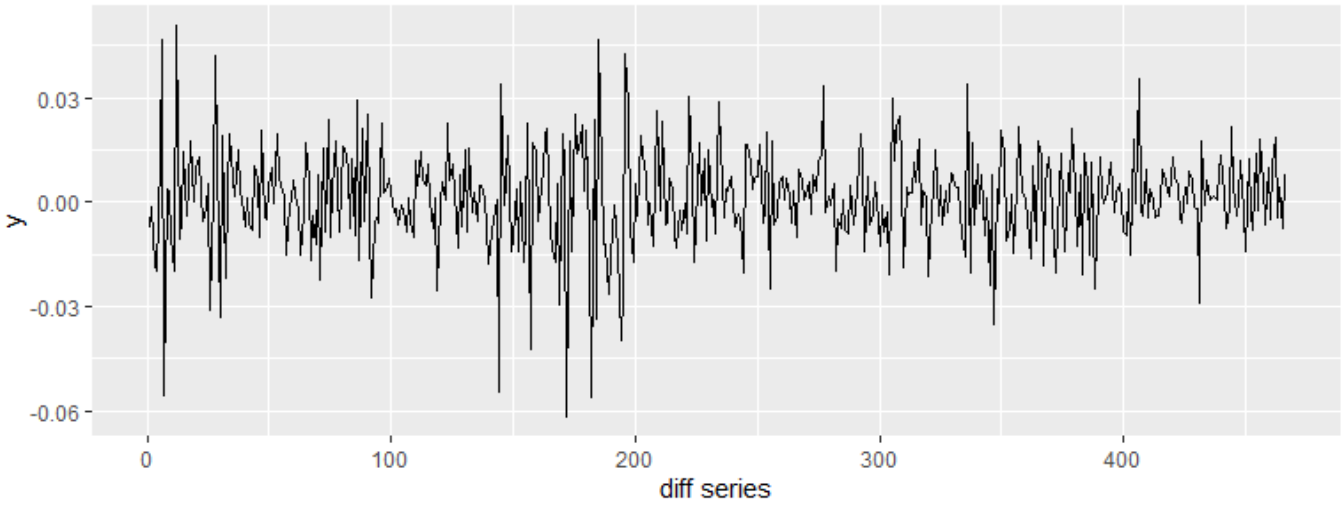


Рис. 3. 15. Вигляд ряду після набуття стаціонарності.

Джерело: розроблено автором.

Наступним кроком є знаходження параметрів ARMA(p, q). Для цього розглянемо графіки ACF та PACF (рис. 3.16) . Як ми можемо бачити, ми маємо значущі лаги 1, 2, 9 на ACF, та 1,9,18 на графіку PACF. Зважаючи на отримані результати доцільно буде припустити наступну специфіку моделі - ARMA(1,2).

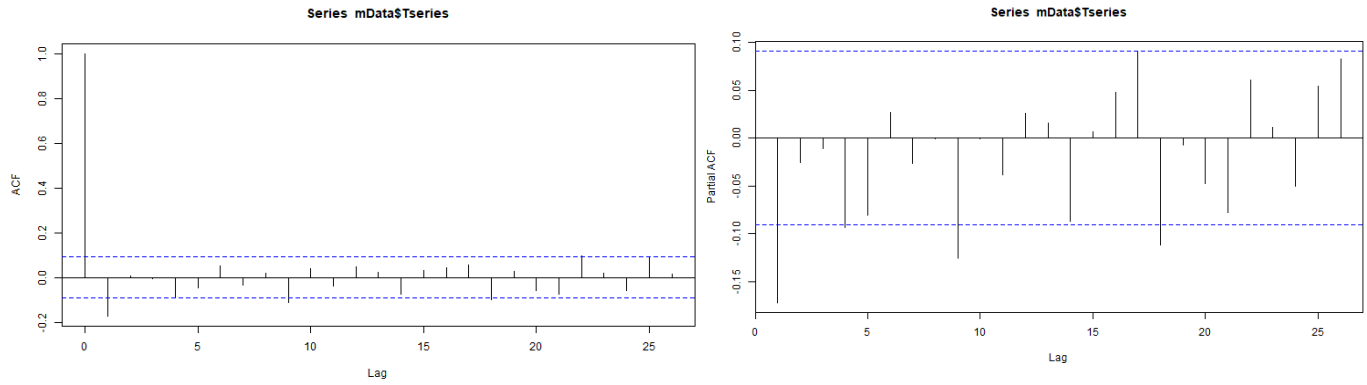


Рис. 3.16. Графіки автокореляційної та частково автокореляційної функцій.

Джерело: розроблено автором.

Проте незважаючи на весь потенціал цього методу, доволі складно вручну вибрати порядок параметрів ARMA(p ,q), яка буде добре працювати при прогнозуванні набору даних [37]. Для вирішення цієї проблеми чудово підійде функція auto.arima з пакету forecast.

Дана функція передбачає можливість знаходження оптимальної ARIMA спираючись на відповідний критерій. У нашому випадку це критерій AIC (Інформаційний критерій Акаіке).

```
Series: mData$Tseries
ARIMA(1,1,2) with drift

Coefficients:
         ar1      ma1     ma2   drift
      0.8131  -0.9964  0.1007  0.0013
s.e.  0.0956   0.1051  0.0560  0.0004

sigma^2 estimated as 0.0001975:  log likelihood=1328.26
AIC=-2646.51   AICc=-2646.38   BIC=-2625.79
```

Рис. 3.17. Результати функції auto.arima для логарифмованого ряду.

Джерело: розроблено автором.

Спираючись на отримані результати ми можемо стверджувати, що модель ARIMA(1, 1, 2) – э оптимальною (рис. 3.17.).

Для остаточної впевненості у правильності вибраної моделі перевіримо її на адекватність. Модель є адекватною, якщо ряд залишків є випадковою компонентою, тобто ACF залишків не повинна істотно відрізнятися від нуля.

Для цього скористаємося критерієм Льюнга-Бокса (*Ljung–Box test*) [38]:

```
        Box-Ljung test

data:  residuals(mData$arima)
X-squared = 11.445, df = 12, p-value = 0.4912
```

Рис. 3.18. Результати тесту Льюнга-Бокса для ARIMA(1, 1, 2).

Джерело: розроблено автором.

Як видно з результатів тесту (рис. 3.18.) , ми не відхиляємо нульову гіпотезу, що данні є білим шумом (p-value > 0,05). Аналогічного висновку можна дійти, подивившись на графік залишків, ACF та PACF (рис. 3.19.)

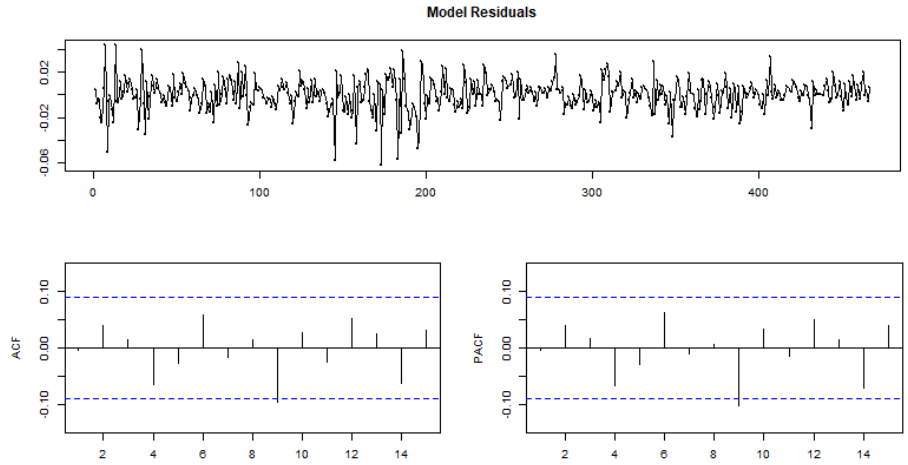


Рис. 3.19. Графік залишків, ACF та PACF для ARIMA(1, 1, 1).

Джерело: розроблено автором.

Останнім кроком у ході аналізу моделі Arima є проведення прогнозу. Оскільки побудована Arima(1, 1, 2) для прогнозування базується на минулих двох значеннях ряду, то доцільним буде здійснити прогноз на 2 періоди вперед, для забезпечення його точності та якості. Відповідні результати зазначені нижче:

	Point Forecast	Lo 80	Hi 80	Lo 95	Hi 95
	All	All	All	All	All
468	167.376351050212	164.389038105182	170.417950088369	162.829293750673	172.050386300764
469	167.422580483018	163.574632819688	171.361047690636	161.573582634677	173.483313289959

Рис. 3.20. Прогноз Arima(1,1,2) на наступні 2 періоди

Джерело: розроблено автором.

Наступним кроком у побудові гібридної моделі ARMA-GARCH/ARCH є визначення найкращої Arch моделі. Для цього необхідно перевірити ряд на наявність Arch ефекту, присутність якого, є обов'язковою умовою їх застосування [39].

Строгий білий шум не може бути передбачений ні лінійно, ні нелінійно, в той час як загальний білий шум не може бути передбачений лише лінійно. Якщо залишки представляють собою строгий білий шум, вони незалежні з нульовим середнім, нормально розподілені, і ACF & PACF квадратів залишків не покажуть значних лагів (рис. 3.21.).

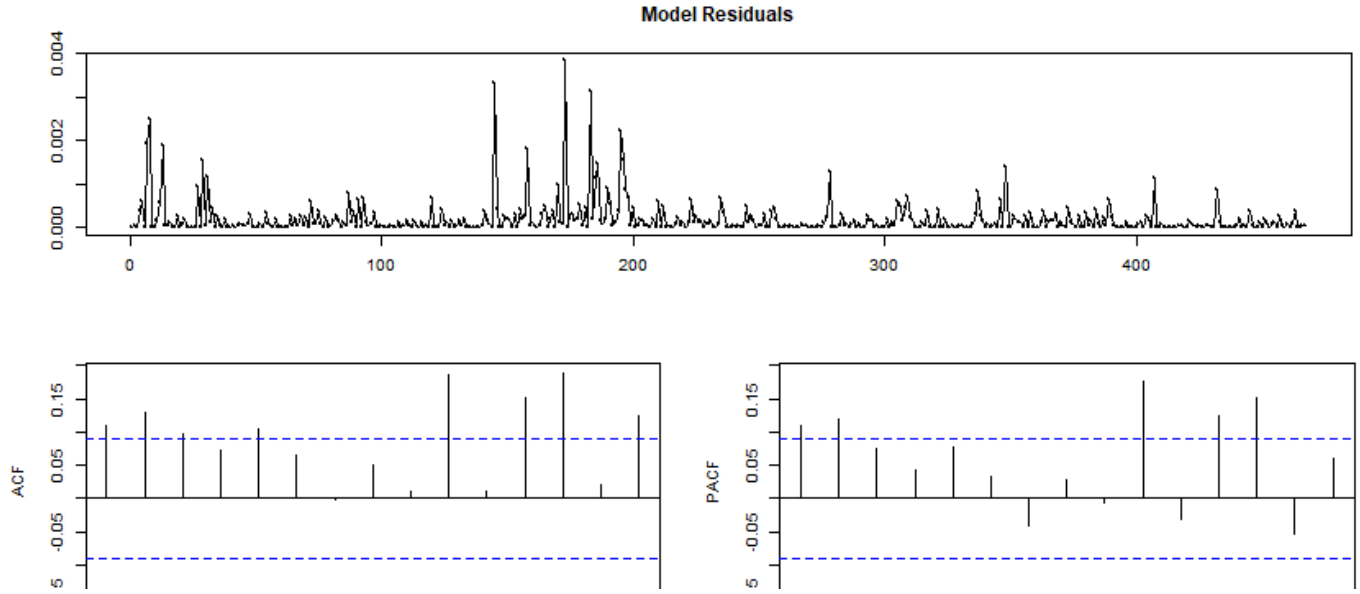


Рис. 3.21. ACF та PACF для квадратів залишків.

Джерело: розроблено автором.

В якості альтернативи тесту Енгеля для ARCH (LM test), ми можемо перевірити послідовну залежність (наявність ефекту ARCH) у ряді залишків, виконавши Q-тест Льюнга-Боксу для перших m лагів квадратів залишків.

Кількість лагів m = P + Q, де GARCH(P, Q), ARCH(P * Q). Оскільки ми ще не знаємо значення параметрів моделей, ми виконаємо перевірку для деякого ряду випадків (рис. 3.22.):

	lag 1	lag 2	lag 3	lag 4	lag 5
1	0.0174560737114994	0.00113572959277564	0.000421821932154898	0.000386964956790803	0.000100342857325963

Рис. 3.22 Q-тест Льюнга-Боксу для перших m лагів квадратів залишків

Джерело: розроблено автором.

Оскільки значення p-value в усіх 5 випадках менше за 5%, ми можемо стверджувати про наявність ARCH ефекту.

Наступним кроком є знаходження найкращої ARMA-ARCH моделі. Нижче зображені результати регресії для ARCH(1)-ARCH(4) відповідно (рис. 3.23):

	AIC	BIC	SIC	HQIC
ARIMA-Arch(1)	-5.69713646824028	-5.64377785492536	-5.69746244283585	-5.67613630834475
ARIMA-Arch(2)	-5.75134593211469	-5.68909421658061	-5.75178838079333	-5.7268457455699
ARIMA-Arch(3)	-5.76039709879406	-5.68925228104083	-5.7609733828176	-5.73239688560001
ARIMA-Arch(4)	-5.76786318545435	-5.68782526548196	-5.76859052230542	-5.73636294561105

Рис. 3.23. Результати інформаційних критеріїв для моделей ARIMA-ARCH

Джерело: розроблено автором.

Виходячи з наведених результатів, найкращою є модель Arima-Arch(4), оскільки їй відповідає найменше значення критерію AIC -5.7678. Проте, з урахуванням значимості оцінених коефіцієнтів, ми робимо вибір у сторону моделі Arima-Arch(2). Відповідні результати зображені нижче:

```
Error Analysis:
         Estimate  Std. Error  t value Pr(>|t|)
mu      4.176e-04   1.495e-04    2.793 0.005224 **
ar1     7.523e-01   7.293e-02   10.316  < 2e-16 ***
ma1    -8.900e-01   8.757e-02  -10.162  < 2e-16 ***
ma2     2.136e-02   5.682e-02    0.376 0.707041
omega   1.079e-04   1.434e-05    7.527 5.17e-14 ***
alpha1  1.043e-01   4.944e-02    2.110 0.034817 *
alpha2  4.441e-01   1.269e-01    3.499 0.000466 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Рис. 3.24. Результати оцінки моделі Arima-Arch(2)

Джерело: розроблено автором.

```
Error Analysis:
         Estimate  Std. Error  t value Pr(>|t|)
mu      4.575e-04   1.519e-04    3.012 0.002591 **
ar1     7.420e-01   7.187e-02   10.324  < 2e-16 ***
ma1    -9.314e-01   8.696e-02  -10.711  < 2e-16 ***
ma2     4.878e-02   5.874e-02    0.831 0.406244
omega   8.480e-05   1.305e-05    6.496 8.24e-11 ***
alpha1  6.612e-02   4.659e-02    1.419 0.155848
alpha2  3.703e-01   1.078e-01    3.434 0.000595 ***
alpha3  1.244e-01   5.922e-02    2.101 0.035678 *
alpha4  8.707e-02   6.043e-02    1.441 0.149652
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Рис. 3.25. Результати оцінки моделі Arima-Arch(4)

Джерело: розроблено автором.

Після вибору найкращої моделі ARMA-ARCH, наступним кроком є побудова прогнозу. Відповідні результати зазначені нижче:

	ARMA-ARCH(2)
T+1	167.518311477315
T+2	167.619385111179

Рис. 3.26. Прогноз Arima-Arch(2) моделі на наступні 2 періоди.

Джерело: розроблено автором.

Наступним етапом є знаходження оптимальної ARMA-GARCH моделі. Значною перевагою GARCH моделі перед ARCH є те, що умовна дисперсія залежить також від минулих значень самої дисперсії [40]. Таким чином частково нівелюється проблема умови коефіцієнтів, оскільки GARCH(P, Q) може дати не гірші результати за ARCH (P + Q) [41]. Виходячи з цього припущення розглянемо 4 основні моделі GARCH: (1, 1); (1, 2); (2,1); (2,2). Виходячи із результатів, зображених на рис. 3.27. найкращою моделлю є Arima-Garch(2,2).

	ARIMA-Garch(1,1)	ARIMA-Garch(1,2)	ARIMA-Garch(2,1)	ARIMA-Garch(2,2)
Akaike	-5.81474700750352	-5.81045523644269	-5.81751580724442	-5.81755533576004
Bayes	-5.74360218975029	-5.7304173164703	-5.73747788727204	-5.7286243135685
Shibata	-5.81532329152707	-5.81118257329376	-5.8182431440955	-5.81845080092081
Hannan-Quinn	-5.78674679430948	-5.77895499659939	-5.78601556740112	-5.78255506926748

Рис. 3.27. Результати інформаційних критеріїв для моделей ARIMA-GARCH

Джерело: розроблено автором.

Проте, зважаючи на значимість коефіцієнтів, модель Arima-Garch(1,1) є найкращою з представлених (рис. 3.28).

```
GARCH Model     : sGARCH(1,1)
Mean Model      : ARFIMA(1,0,2)
Distribution    : std

Optimal Parameters
------------------------------------
        Estimate  Std. Error  t value Pr(>|t|)
mu      0.001705    0.000278   6.1321 0.000000
ar1     0.819439    0.067199  12.1941 0.000000
ma1    -0.975701    0.060783 -16.0523 0.000000
ma2     0.067113    0.035641   1.8830 0.059695
omega   0.000009    0.000001   6.5283 0.000000
alpha1  0.074133    0.013044   5.6831 0.000000
beta1   0.876857    0.024487  35.8098 0.000000
shape   5.590065    1.265308   4.4179 0.000010
```

Рис. 3.28. Найкраща Arima-Garch модель

Джерело: розроблено автором.

Після знаходження оптимальної Arma-Garch, наступним кроком є побудова прогнозу. Відповідні результати зазначені нижче:

	ARMA-sGARCH(1,1)
T+1	167.480954968141
T+2	167.583778857824

Рис. 3.29. Прогноз Arima-Garch(1,1) моделі на наступні 2 періоди.

Джерело: розроблено автором.

Одним з розширенням стандартної моделі Garch є модель TGarch, основним припущенням якої є наявність асиметричного ефекту на волатильність, обумовленого значенням збурення e_{t-1}.

В цілому, коли негативні новини потрапляють на фінансовий ринок, ціни на активи, як правило, вступають в бурхливу фазу, і волатильність збільшується, але при позитивних новинах волатильність, як правило, невелика, і ринок вступає в період стабільності.

Нище представлені результати аналізу моделей Arma-TGarch для раніше вказаних порядків (Рис) На основі наведених результатів, найкращою моделлю є Arma-TGarch(1,1) із значенням AIC -5.8322.

	ARIMA-TGarch(1,1)	ARIMA-TGarch(1,2)	ARIMA-TGarch(2,1)	ARIMA-TGarch(2,2)
Akaike	-5.83226483550039	-5.82797295892301	-5.82388292620953	-5.81945874677401
Bayes	-5.75222691552801	-5.73904193673147	-5.72605880179884	-5.71274152014416
Shibata	-5.83299217235147	-5.82886842408379	-5.82496345491121	-5.82074113571773
Hannan-Quinn	-5.80076459565709	-5.79297269243046	-5.78538263306772	-5.77745842698294

Рис. 3.30. Результати інформаційних критеріїв для моделей ARIMA-TGARCH

Джерело: розроблено автором.

На етапі оцінки параметрів, дана модель також показала найкращі результати (рис. 3.31).

```
GARCH Model     : fGARCH(1,1)
fGARCH Sub-Model        : TGARCH
Mean Model      : ARFIMA(1,0,2)
Distribution    : std

Optimal Parameters
------------------------------------
        Estimate  Std. Error  t value Pr(>|t|)
mu      0.001538    0.000306   5.0311 0.000000
ar1     0.850697    0.077411  10.9894 0.000000
ma1    -0.988048    0.066624 -14.8302 0.000000
ma2     0.066807    0.031016   2.1540 0.031243
omega   0.001378    0.000851   1.6199 0.105252
alpha1  0.093899    0.051802   1.8126 0.069886
beta1   0.825076    0.092485   8.9212 0.000000
eta11   0.995803    0.497676   2.0009 0.045402
shape   5.484563    1.335159   4.1078 0.000040
```

Рис. 3.31. Найкраща Arima-TGarch модель

Джерело: розроблено автором.

Останнім етапом у роботі з Arma-TGarch є побудова прогнозу. Відповідні результати для наступних двох періодів зазначені нижче:

	ARMA-TGARCH(1,1)
T+1	167.457241198459
T+2	167.551841222922

Рис. 3.32. Прогноз Arima-TGarch(1,1) моделі на наступні 2 періоди.

Джерело: розроблено автором.

Ще одним популярним розширенням моделі GARCH є модель «GARCH-in-mean», де акцент робиться на позитивному взаємозв'язку між ризиком (який часто вимірюється волатильністю) і прибутковістю фінансових інструментів.

Однак, хоча ми оцінили середнє рівняння для моделювання прибутковості і оцінили модель GARCH для визначення мінливості, що змінюється в часі, ми не використали ризик для пояснення прибутковості. Саме це і покликана зробити модель «GARCH-in-mean».

У ході дослідження були отримані результати оцінки критеріїв гібридних моделей, де найкращою стала модель Arima-MGarch(1,1), із значенням AIC -5.8010.

	ARIMA-MGarch(1,1)	ARIMA-MGarch(1,2)	ARIMA-MGarch(2,1)	ARIMA-MGarch(2,2)
Akaike	-5.80104550600618	-5.79658413469207	-5.79226307332246	-5.78846125680901
Bayes	-5.70322138159549	-5.68986690806223	-5.67665274447346	-5.66395782574086
Shibata	-5.80212603470786	-5.79786652363579	-5.79376398237211	-5.79019721065661
Hannan-Quinn	-5.76254521286437	-5.75458381490101	-5.74676272688214	-5.73946088371943

Рис. 3.33. Результати інформаційних критеріїв для моделей ARIMA-MGARCH

Джерело: розроблено автором.

Такий же висновок був отриманий на основі оцінок параметрів моделей. Результат найкращої моделі зазначений нижче (рис.3.34.):

```
GARCH Model     : fGARCH(1,1)
fGARCH Sub-Model        : APARCH
Mean Model      : ARFIMA(1,0,2)
Distribution    : std

Optimal Parameters
------------------------------------
        Estimate  Std. Error   t value Pr(>|t|)
mu      0.001472    0.001340  1.098796 0.271857
ar1    -0.305621    0.877974 -0.348098 0.727767
ma1     0.161584    0.875653  0.184530 0.853598
ma2    -0.014941    0.145308 -0.102825 0.918102
archm   0.343703    7.709999  0.044579 0.964443
omega   0.000000    0.000001  0.105437 0.916029
alpha1  0.014001    0.011789  1.187684 0.234958
beta1   0.922739    0.075545 12.214361 0.000000
eta11   0.546782    0.407789  1.340844 0.179971
lambda  2.826618    0.270276 10.458271 0.000000
shape   5.965129    1.680047  3.550573 0.000384
```

Рис. 3.34. Найкраща Arima-MGarch модель

Джерело: розроблено автором.

Після знаходження оптимальної Arma-MGarch, наступним кроком є побудова прогнозу. Відповідні результати зазначені нижче:

	ARMA-MGARCH(1,1)
T+1	167.593286377338
T+2	167.885555097156

Рис. 3.35. Прогноз Arima-MGarch(1,1) моделі на наступні 2 періоди.

Джерело: розроблено автором.

На даному етапі аналізу ми використали вже доволі знайомі та поширені методи моделювання часових рядів. Кожен з них має свої особливості будови та умови, дотримання яких є необхідним для їх ефективного застосування.

У ході аналізу основу методології склали наступні моделі: лінійна модель Arima й її гібридне поєднання з моделями авторегресивної умовної гетероскедастичності Arch та Garch (й їх різновидів). Виходячи із результатів проведеного дослідження ми можемо порівняти їх ефективність моделювання обраного ряду на основі спільного критерію – AIC. Відповідні значення відображені в таблиці нижче:

Таблиця 3.2. Інформаційний критерій AIC для оцінених моделей

	AIC	AIC (rugarch)
Arima(1, 2)	-2646.51	-5.6792
Arma(1, 2) - Arch(2)	-2684,30	-5.7603
Arma(1, 2) - Garch(1, 1)	-2709,65	-5,8147
Arma(1, 2)-TGarch(1, 1)	**-2717,81**	**-5,8322**
Arma(1, 2)-MGarch(1, 1)	-2703,27	-5,8010

Джерело: розроблено автором.

Слід зазначити, що для побудови моделей Garch використовувався пакет rugarch, де формула AIC дещо видозмінена (фінальний результат поділений на кількість періодів у вибірці). Цим обумовлена необхідність створення двох колонок AIC, для можливості інтерпретації результатів та їх порівняння.

Згідно наведених даних, модель Arma(1,2)-TGarch(1,1) є найкращою з представлених. Їй належить найменше значення AIC у -2717,81 одиниць.

Альтернативним способом перевірки ефективності наведених моделей є порівняння якості їх прогнозу із реальними даними. Дані результати наведені нижче:

Таблиця 3.3. Інформаційний критерій AIC для оцінених моделей

	T+1	T+2
Arima(1, 2)	167,38	167,42
Arma(1, 2) - Arch(2)	167,52	167,62
Arma(1, 2) - Garch(1, 1)	167,48	167,58
Arma(1, 2)-TGarch(1, 1)	167,46	167,55
Arma(1, 2)-MGarch(1, 1)	**167,59**	**167,88**
Фактичне значення ряду	167.54	167,91

Джерело: розроблено автором.

Головною метою будь-якого алгоритму моделювання даних є знаходження максимально наближених до фактичних результатів, шляхом визначення ряду прихованих закономірностей всередині самих даних. Виходячи з наведених значень, ми можемо констатувати, що для даного часового проміжку, найкраще з прогнозом впоралась модель Arma(1, 2)-MGarch(1, 1). Проте, в цілому усі моделі показали доволі близькі до реальних результати. Виходячи із розрахунків критеріїв AIC, модель Arma(1, 2)-TGarch(1, 1) здатна краще виявити дані закономірності у довгостроковій перспективі. А тому, для більш якісного моделювання даного часового ряду є доцільним спиратися на результати декількох моделей.

3.3 Порівняльний аналіз отриманих результатів та синтез рекомендацій по моделюванню часових рядів.

Отже, виходячи з інформації зазначеної в розділі, ми можемо стверджувати, що традиційні моделі, які широко використовуються для моделювання фінансових часових рядів – це потужна, надійна методологія, яка враховує особливості часового ряду та дозволяє отримати надійні результати. І хоча у ході проведеного аналізу, нейронні мережі поступаються точністю прогнозів, при належному тренуванні вони здатні перевершити оцінки зазначених моделей.

Базуючись на результати класичних моделей (табл. 3.3), ми можемо констатувати, що для даного часового проміжку, найкраще з прогнозом впоралась модель Arma(1, 2)-MGarch(1, 1). Проте, в цілому усі моделі показали доволі близькі до реальних результати. Виходячи із розрахунків критеріїв AIC (табл. 3.2), модель Arma(1, 2)-TGarch(1, 1) здатна краще виявити дані закономірності у довгостроковій перспективі. Той факт, що подібні комбінації моделей здані показати кращу ефективність моделювання, пояснюється нівелюванням головної проблеми arima. Дисперсія у представлених моделях не є константою, а тому вона здатна краще враховувати аспект мінливої волатильності ринку. Таким чином, об’єднавши потужності двох типів моделей, нам вдалося врахувати особливості наведеного часового ряду.

Серед нейронних мереж, найкращі результати продемонструвала RNN модель (табл. 3.1). Значення оцінок для тренувальної та тестових вибірок є найкращими з усіх представлених. Навіть на проміжках зі значною волатильністю ефективність моделювання залишається високою. Аналогічного висновку можна дійти, спираючись на критерій RMSE. Для мережі RNN, це значення 0.046 і 0.081, відповідно І хоча LSTM та GRU є більш складним та ефективними, з точки зору моделювання архітекторами, в даному аналізі вони показали гірші результати.

У ході реалізації було виконано безліч тестів, щодо трансформацій даних та тренувань нейронних мереж. Метою даних досліджень було розуміння принців навчання моделей та відповідних факторів впливу на результати дослідження. Глибинне розуміння архітектур, відображене у секції 2 цього диплому, є безумовною

перевагою в розумінні цих процесів. Але у той же час, існує низка інших аспектів, які слід брати до уваги. Саме тому, був складений і описаний ряд проблем моделювання методами нейронних мереж та можливі рішення, щодо їх усунення.

В першу чергу, справа стосуються взаємозв’язку між параметрами, які ми встановлюємо, як: кількість вхідних даних серії (input_size), прихованих станів (hidden_size), форма «batches», кількість ітерацій навчання, тощо. Вчені, які висвітлюють цю область аналізу, намагаються дослідити цей взаємозв'язок і його вплив на продуктивність моделювання.

У мережі, існує безліч публікацій, де цим аспектам не приділяється належна увага. Переважно їх автори, схильні задавати велику кількість прихованих станів моделі, припускаючи, що це допоможе їй знайти закономірності в даних та підвищити ефективність моделювання часового ряду. За таких умов, існує досить висока вірогідність, що процес пошуку закономірностей перетвориться на процес запам’ятовування значень тренувальної вибірки. Результатом цього є тотальний провал продуктивності моделі на тестовому сеті даних.

Ще одна розповсюджена помилка стосується етапу трансформації даних, а саме збереження їх у формі «batches» - груп послідовностей. Для їх формування ми маємо завчасно вирішити, яку кількість параметрів часового ряду ми надаємо нейронній мережі в кожний момент часу (input_size). Доволі дивним є той факт, що в більшості публікацій значення цього параметру дорівнює одиниці.

За таких умов, проблема ефективності моделювання полягає в тому, якщо для рекурентних меж, метою яких є визначення закономірностей в ряді послідовностей, данні на вході є окремими значеннями в кожний момент часу. В такому разі, рекурентна мережа нічим не відрізняється від звичайної AR(1) моделі.

У нашому випадку, ми уникнули цієї помилки, написавши відповідну функцію «data_split», метою якої є створення відповідних груп послідовностей із урахуванням ряду параметрів. Завдяки їй, ми можемо задати бажану структуру даних, беручи до уваги структуру мережі.

Проте, відкритим залишилося питання взаємодії параметрів, пов’язаних зі налаштуванням самої моделі. Існує ряд фреймворків, які містять коефіцієнти,

отримані в результаті відповідних досліджень, та структурних параметрів самої моделі. Проте, на жаль, статті, опубліковані на відомих ресурсах як «arxiv.org», представлені без коду. Тому, ми можемо покладатися лише на власне розуміння того, як проводився аналіз.

Ще одне дуже важливе уточнення, це вибір алгоритму оптимізації та функції витрат. У нашому аналізі ними стали алгоритм Адама та «MSE Loss», відповідно. На моє переконання, функція витрат не повністю відповідає поставленому завданню. Оскільки така річ, як урахування напрямку руху часового ряду не враховується зовсім.

Розглянемо приклад: якщо вартість активу для періодів «t-1», «t» становить 1000.00 та 999.99 одиниць відповідно, а значення прогнозу 1000.01, то з урахуванням побудови функції витрат, дана похибка не є значущою, а проноз точним. В цьому і полягає концептуальна проблема.

В даному випадку, враховується лише близькість значень, тоді як хибність напрямку прогнозу не береться до уваги. Через це, ми не враховуємо частину інформації, яка може значно допомогти нам підвищити ефективність моделювання. Тому важливим, є розгляд альтернативних функцій витрат та їх тестування.

Також, питання трансформації початкових даних є вкрай важливим, що було зазначено та детально описано у висновках пункту 3.1. Незважаючи на те, що нормалізація у ряді тестів показала кращі результати, її використання, з урахуванням специфіки часових рядів, на моє переконання - є хибним. Дослідження відносно традиційних та інших форм, включаючи стандартизацію та підходи описані наприкінці розділу, заслуговують більшої уваги та подальшого тестування.

Отож, згідно отриманих результатів, кращим виявилися традиційні моделі. Їм вдалося врахувати специфіку ряду та його закономірностей. Проте нейронні мережі також показали непогані результати. Враховуючи кількість факторів описаних вище, я можу стверджувати, що прогнозування методами нейронних мереж є значно складнішим, проте у разі знаходження взаємозалежності параметрів формування даних та структурних складових моделі, вони здані суттєво покращити та перевершити результати традиційних підходів.

ВИСНОВКИ

В умовах глобалізації та стрімкого розвитку комп'ютерних технологій виникає потреба швидкого реагування на зміни, які несуть в собі фінансові ринки. Поява нових інструментів та збільшення об'єму інформації змушують учасників торгів шукати нові методи дослідження біржової кон'юнктури.

Складна природа фінансових інструментів, поведінка яких відображає вплив відразу безлічі факторів, виражатися в таких проявах як волатильність, що негативно позначається на роботі традиційних алгоритмів. Саме тому, великої популярності набули нові, потужніші архітектури, як рекурентні нейронні мережі.

Згідно **завдань дипломної роботи**:

- проаналізована низка наукових публікацій, стосовно особливостей рекурентних нейронних мереж, аспектів їх побудови та факторів впливу на результати моделювання.
- був проілюстрований принцип навчання нейронних мереж на прикладі логістичної регресії, з метою формування математичного підґрунтя цього процесу. Детально розкриті етапи знаходження оцінок моделі, градієнтів параметрів та шляху їх оптимізації. Описані ролі структурних складових процесу, як: розрахунковий граф та матриці Якобі. З метою усунення проблеми сумісності теоретичної та практичної частин, були реалізовані відповідні блоки коду (додаток А).
- розглянуті базові архітектури рекурентних нейронних мереж, їх ключові компоненти, принципи обрахунку, а також переваги та недоліки відносно один одного. Особлива увага приділялася процесам прямого та зворотного поширення в часі. Даний розділ також супроводжується практичною реалізацією (додаток В).
- поведений аналіз та відповідне порівняння результатів мережевого моделювання та класичних методів машинного навчання. У якості даних використовувався часовий ряд акцій компанії MSFT в період з 2018-03-17 по 2020-01-25. Згідно отриманих результатів, найкраще з прогнозом впиралась

модель Arma(1, 2)-MGarch(1, 1). Проте, в цілому усі моделі показали доволі близькі до реальних результати. Згідно розрахунків критеріїв AIC, модель Arma(1,2)-TGarch(1,1) здатна краще виявити приховані в даних закономірності у довгостроковій перспективі. Щодо нейронних мереж, найкращі результати продемонструвала RNN модель. Значення прогнозів та оцінок RMSE для тренувальної та тестових вибірок, є найкращими серед представлених мереж. І хоча в розрізі даного аналізу, нейронні мережі поступаються точністю прогнозів, при належному тренуванні вони здатні перевершити оцінки класичних моделей. Саме тому, був складений і описаний ряд проблем поточного моделювання методами нейронних мереж та можливі рішення, щодо їх усунення.

Теоретична цінність полягає в формуванні фундаментального розуміння принципів роботи нейронних мереж, їх побудови, процесу навчання та виявленні ключових факторів впливу на результати моделювання. Отриманні знання, можуть слугувати основою для вивчення більш складних архітектур глибинного навчання.

Практична цінність полягає у застосуванні отриманих знань на практиці, освоєнні сучасних інструментів реалізації аналізу, а також процесу тренування та моделювання методами рекурентних нейронних мереж.

Головною перевагою роботи є детально викладений матеріал, щодо видів рекурентних мереж, їх архітектур та математично обґрунтування процесів прямого та зворотного поширення. Даний аспект, дозволяє проектувати принцип навчання НМ на інші алгоритми машинного навчання, для дослідження їх будови та принципів роботи. Чудовим прикладом цього, є розділ 1.3.

Актуальність теми пов’язана з виском інтересом до алгоритмів глибинного навчання. Особливості їх будови та гнучкість стосовно форматів даних, дозволяє значно розширити області їх застосування. У контексті моделювання фінансових інструментів біржі, рекурентні нейронні мережі набули великої популярності. За належного тренування, а краще, поєднання з іншими мережами, як NLP (урахування фактору новин), вони здатні значною мірою перевершити класичні методи аналізу фінансових часових рядів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. McCulloch W. S. and Pitts W., “A logical calculus of the ideas immanent in nervous activity,” The bulletin of mathematical biophysics, vol. 5, no. 4, pp. 115–133, 1943.
2. Hebb D. O., The organization of behavior: A neuropsychological theory. Psychology Press, 2005.
3. Löwel, S. and Singer, W., "Selection of Intrinsic Horizontal Connections in the Visual Cortex by Correlated Neuronal Activity", pp. 209–212, 1992.
4. Crevier D., AI: “The Tumultuous Search for Artificial Intelligence”, p. 39, 1993.
5. Rosenblatt F., “The perceptron: A probabilistic model for information storage and organization in the brain.” Psychological review, vol. 65, no. 6, p. 386, 1958.
6. Minsky M., and Papert S., “Perceptrons: An Introduction to Computational Geometry” MIT Press, Cambridge, MA, USA, 1969.
7. Werbos P. Beyond regression:" new tools for prediction and analysis in the behavioral sciences. Ph. D. dissertation, Harvard University, 1974.
8. Rumelhart D.E., Hinton G.E., and Williams R.J.. Learning representations by back-propagating errors. nature, 323(6088):533, 1986.
9. Martin T.B. and Talavage J.J., “Application of neural logic to speech analysis and recognition,” IEEE Transactions on Military Electronics,vol. MIL-7, no. 2 & 3, pp. 189–196, April 1963.
10. Smith P., Bull D., and Wykes C., “Target classification with artificial neural networks using ultrasonic phased arrays,” WIT Transactions on Information and Communication Technologies, vol. 1, 1970.
11. Wang Z., Turko R., Shaikh O. CNN Explainer: Learning Convolutional Neural Networks with Interactive Visualization. arXiv:2004.15004v3, 2020, URL: https://arxiv.org/pdf/2004.15004.pdf
12. O'Shea K., Nash R., An Introduction to Convolutional Neural Networks. Lancaster University. pp. 4–8, 2015. URL: https://arxiv.org/pdf/1511.08458.pdf
13. Qin Z., Kim D., Gedeon T., Rethinking Softmax with Cross-Entropy: Neural Network Classifier as Mutual Information Estimator. arXiv:1911.10688v4. 2020. URL: https://arxiv.org/pdf/1911.10688.pdf

14. Yosinski J., Clune J., Nguyen A., Fuchs T., and Lipson H., Understanding Neural Networks Through Deep Visualization. In ICML Deep Learning Workshop, 2015.
15. Fouh E., Akbar M., and Shaffer A., The Role of Visualization in Computer Science Education. Computers in the Schools, 29(1-2):95–117, Jan. 2012.
16. Li Z., Yang W., Peng S., Liu F., A Survey of Convolutional Neural Networks: Analysis, Applications, and Prospects. arXiv:2004.02806. 2020. URL: https://bit.ly/3sGnoqs
17. Gajamannage K., Park Y., Real-time Forecasting of Time Series in Financial Markets Using Sequentially Trained Many-to-one LSTMs. arXiv:2205.04678v1. 2022. URL: https://arxiv.org/pdf/2205.04678.pdf
18. Wang J., Yang Y., Mao J., CNN-RNN: A Unified Framework for Multi-label Image Classification. arXiv:1604.04573. 2016. URL: https://bit.ly/3sMUwgp
19. Ng A., Neural Networks Basics: Computation Graph. Coursera. 2017. URL: https://bit.ly/3Nt6zaB
20. Clark K., Computing Neural Network Gradient. Stanford university. 2019. URL: https://stanford.io/3sLc4cK
21. Pytorch official documentation, URL: https://bit.ly/3l6XIzc
22. Amidi A., Amidi S., “Recurrent Neural Networks”, Stanford University. 2020. URL: https://stanford.io/3l9bVvl
23. Chen G., “A Gentle Tutorial of Recurrent Neural Network with Error Backpropagation”, SUNY at Buffalo. 2018. URL: https://bit.ly/3G3IHIc
24. Hochreiter J., Institut fur Informatik, «Untersuchungen zu dynamischen neuronalen Netzen», Technische Universit¨at Munchen, 1991. URL: https://bit.ly/3szzRMF
25. Abraham D., "Retention Length and Memory Capacity of Recurrent Neural Network". University of Pretoria, University of Pretoria, 2020. URL: https://bit.ly/3oSOmcH
26. Bengio Y., Simard P., Frasconi P., «Learning long-term dependencies with gradient descent is difficult» IEEE Transactions on Neural Networks Volume: 5, Issue: 2, March 1994.

27. Hochreiter S., Schmidhuber J., Long short-term memory. // Neural Computation journal. — 1997. — Vol. 9, no. 8. — P. 1735—1780. URL: https://bit.ly/3sK1rpM
28. Elsayed N., ElSayed Z., LSTM Architecture for Deep Recurrent Neural Networks. University of Cincinnati. 2020. ULR: https://arxiv.org/pdf/2201.11624.pdf
29. Ling Q., Matloob K., Josiah P.. Event-Driven LSTM For Forex Price Prediction. The University of Sydney, Australia. 2021. URL: https://bit.ly/3rUuKa0
30. Landia F., Baraldi L. “Working Memory Connections for LSTM”, University of Modena and Reggio Emilia, Modena, Italy, URL: https://bit.ly/3Mx3VR4
31. Chung J., Gulcehre C., Cho K., Bengio Y., "Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling". 2014. URL: https://bit.ly/3sKkSiU
32. Dey R., Salem F., Gate-Variants of Gated Recurrent Unit (GRU) Neural Networks. Michigan State University. 2017. URL: https://bit.ly/3Ly01WG
33. Lawi A., Mesra H., Amir S., “Implementation of LSTM and GRU on Grouped Time-Series Data to Predict Stock Prices Accurately. 2021. URL: https://bit.ly/3yA5aL9
34. Bao, Wei, Jun Y., and Yulei R., A deep learning framework for financial time series using stacked autoencoders and long-short term memory. 2017.
35. Shunrong S., Jiang H., Zhang T., Stock Market Forecasting Using Machine Learning Algorithms. Stanford: Department of Electrical Engineering, Stanford University, pp. 1–5, 2012.
36. Nau K., Hansen B. “Stationary and Differencing.” Duke University. Durham, NC. 2000.
37. Identifying the numbers of AR or MA terms in an ARIMA model URL: https://bit.ly/3yN1s0Q
38. Box G. E. P. and Pierce D. A. (1970). "Distribution of Residual Autocorrelations in Autoregressive-Integrated Moving Average Time Series Models". Journal of the American Statistical Association, Vol. 65, No. 332 (Dec., 1970), pp. 1509-1526.
39. Engle R. Autoregressive Conditional Heteroscedasticity with Estamates of the Variance of United Kingdom Inflation/ R. Engle // Econometrica. – 1982. - Vol. 50, Is. 4. – P. 987-1008.

40. Pedersen J. H. ARMA(1,1)-GARCH(1,1) Estimation and forecast using rugarch 1.2-2 URL: https://bit.ly/2wjxgch
41. Bollerslev T. Generalized autoregressive conditional heteroskedasticity. Journal of Econometrics, 31:307–327, 1986

ДОДАТКИ

Додаток «А»

Logistic regression - Section 1.3

```
from google.colab import files
import pandas as pd
import numpy as np
import torch
import torch.nn as nn
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn import metrics
from matplotlib import pyplot as plt
from matplotlib import style


#import data in google colab; file: "datalog.csv"
files.upload()

#read it as a dataframe;
df = pd.read_csv("data_logit.csv")
df.head(5)


x_np = df[["x1", "x2"]].to_numpy()
y_np = df['y'].to_numpy()


#The proportion of training/test groups is 80/20 respectively.
x_train, x_test, y_train, y_test = train_test_split(x_np, y_np, test_size=0.2, r
andom_state=120)


sc = StandardScaler()
x_train = sc.fit_transform(x_train)
x_test = sc.transform(x_test)


#Here we will use our initial "x", that contains all the data about predictors;
sc_example= StandardScaler()
x_empl = sc_example.fit_transform(x_np)

print(f"mean of sc_example: {sc_example.mean_[0]}; manually: {np.mean(x_np[:,0])
}")
print(f"sd of sc_example: {sc_example.var_[0]**0.5}; manually: {np.std(x_np[:,0]
)}") #sd is calculated for the population, to fix that, we need set the degrees
of freedom;
```

```
x_train = np.insert(x_train, 0, 1, axis=1)
x_test = np.insert(x_test, 0, 1, axis=1)


x_train_t = torch.from_numpy(x_train.astype(np.float32))
x_test_t = torch.from_numpy(x_test.astype(np.float32))
y_train_t = torch.from_numpy(y_train.astype(np.float32))
y_test_t = torch.from_numpy(y_test.astype(np.float32))


#Note that we set requires_grad=True in the case of the weight tensor so that it
 can track the phases of the graph;
w = torch.tensor([[0.0], [0.0], [0.0]], dtype=torch.float32, requires_grad=True)
x = x_train_t
y = y_train_t.reshape(-1,1)
print(f"shape of w: {w.shape}; x: {x.shape}; y: {y.shape}")


#The numbers next to the code correspond to the node operations mentioned in the
 computational graph;
z = x @ w #(1)
y_hat = torch.sigmoid(z) #(2)
loss = (- y * torch.log(y_hat) - (1 - y) * torch.log(1-
y_hat)).mean()  #(3) - custom version of nn.BCELoss()


#calculation of gradients with respect to graph leaves.
loss.backward()
w.grad

with torch.no_grad():
  dw0 = (x[:,0].reshape(-1, 1) * (y_hat-y)).mean()
  dw1 = (x[:,1].reshape(-1, 1) * (y_hat-y)).mean()
  dw2 = (x[:,2].reshape(-1, 1) * (y_hat-y)).mean()
print(f"dw0: {dw0}; dw1: {dw1}; dw2: {dw2};")

#Note that regularization is applied by default; To remove it, we need to set: p
enalty='none'
model = LogisticRegression(penalty='none', max_iter = 4000, solver='newton-
cg').fit(x_train[:,[1,2]], y_train)

print(f"Estimated coefficients: intercept {model.intercept_}; w1 {model.coef_[:,
0]}; w2 {model.coef_[:,1]}")

w = torch.tensor([[0.0], [0.0], [0.0]], dtype=torch.float32, requires_grad=True)
x = x_train_t
y = y_train_t.reshape(-1,1)

lr = 0.1 #learning rate
cost_f = np.array([]) #array to collect cost function values
```

```
def model_training(w, x, y, lr, cost_f, num_iter = 4000):

  for epoch in range(num_iter):
    z = x @ w
    y_hat = torch.sigmoid(z)
    loss = (- y * torch.log(y_hat) - (1 - y) * torch.log(1-y_hat)).mean()

    loss.backward()

    with torch.no_grad():
      #update weights
      w -=  lr * w.grad
      #collect the value of cost function
      cost_f = np.append(cost_f, loss)

    #empty the gradients to provide the next iterantion step
    w.grad.zero_()

  return cost_f, w

cost_f, w = model_training(w, x, y, lr, cost_f)
w

# using ggplot style is a habit from R;
style.use('ggplot')

plt.title("Loss function at lr = 0.1")
plt.plot(cost_f)

w = torch.tensor([[0.0], [0.0], [0.0]], dtype=torch.float32, requires_grad=True)
x = x_train_t
y = y_train_t.reshape(-1,1)

lr = 0.01 #learning rate
cost_f = np.array([]) #array to collect cost function values

cost_f, w = model_training(w, x, y, lr, cost_f)
#estimated coefficients
W

plt.title("Loss function at lr = 0.01")
plt.plot(cost_f)

x = x_train_t[:,[1,2]]
y = y_train_t.reshape(-1,1)

class LogisticReg(nn.Module): #(1)
  def __init__(self, n_features_f):
    super(LogisticReg, self).__init__()
    self.linear = nn.Linear(n_features_f, 1)
```

```
    with torch.no_grad():
        self.linear.weight.copy_(torch.zeros(1))
        self.linear.bias.copy_(torch.zeros(1))

  def forward(self, x):
    y_pred = torch.sigmoid(self.linear(x))
    return(y_pred)

n_features = x.shape[1]


#creating an instance of a class "LogisticReg";
model = LogisticReg(n_features)

#create loss and optimizer:
loss = nn.BCELoss()
optimizer = torch.optim.SGD(model.parameters(), lr=0.1)


for epoch in range(4000):
  #forward pass;
  y_hat = model.forward(x)

  #compute the loss and perform backpropagation;
  Loss = loss(y_hat, y).backward()

  #updates;
  optimizer.step()

  #empty gradients;
  optimizer.zero_grad()


#let`s derive the estimated coefficients and represent them in a more convenient
 way;
torch.hstack((model.linear.bias.reshape(-1,1), model.linear.weight)).detach()


#predicted values of the training set;
y_hat = model.forward(x).detach()
#components of the ROC curve;
fpr, tpr, threshold = metrics.roc_curve(y,  y_hat)


plt.title("ROC curve")
plt.plot(fpr,tpr)
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()
```

```
idx = np.argmax(tpr - fpr)
opt_p = threshold[idx]
print(f'The optimal threshold is {opt_p}' )


#getting the predicted values;
y_hat = model.forward(x).detach()
#round them with respect to optimal threshold;
y_hat = torch.round((y_hat - opt_p + 0.5))
#create the confusion matrix;
metrics.confusion_matrix(y, y_hat)
#The prediction accuracy
metrics.accuracy_score(y, y_hat)


x_2 = x_test_t[:,[1,2]]
y_2 = y_test_t.reshape(-1,1)

#getting the predicted values;
y_hat2 = model.forward(x_2).detach()
#round them with respect to optimal threshold;
y_hat2 = torch.round((y_hat2 - opt_p + 0.5))
#create the confusion matrix;
metrics.confusion_matrix(y_2, y_hat2)
#The prediction accuracy
metrics.accuracy_score(y_2, y_hat2)
```

Додаток «В»

Architectures of recurrent neural networks - Section 2

```
import torch
import torch.nn as nn


#Section 2.1 RNN unit architecture

def RNN_cell(x_f, wx_f, wh_f, bx_f = None, bh_f = None):

  #dimensionality check
  if len(x_f.size()) == 3:

    #creating a tensor of hidden states relative to the dimension of the coeffic
ient matrix;
    h_data = h_lag = torch.zeros((wh_f.shape[1],1))

    #setting the correct shape for biases;
    if None in {bx_f, bh_f}:
      bx_f = bh_f = torch.zeros((wh_f.shape[1],1))
    else:
      bx_f, bh_f = bx_f.reshape(-1, 1), bh_f.reshape(-1, 1)

    for i in range(x_f.shape[1]):
      #estimation of the hidden states inside the neuron;
      h_lag = torch.tanh(wx_f @ torch.t(x_f[0][[i],:]) + bx_f + wh_f @ h_lag + b
h_f)
      #data collection
      h_data = torch.cat((h_data, h_lag),1)

    print(torch.t(h_data))

  else:
    print("The input must be a 3D tensor. Whereas its shape is: ", x_f.shape)


seq = torch.FloatTensor([[3, 4], [4, 5], [5, 6]]).unsqueeze(0)
seq = torch.FloatTensor([[3], [4], [5]]).unsqueeze(0)


#This time for each hidden layer we would compute two hidden states;
rnn = nn.RNN(input_size=seq.shape[2], hidden_size=2, num_layers = 1, bias = True
, batch_first=True)

#here we get the hidden states, estimated for each value in the sequence;
out_all, out_last = rnn(seq)
print(out_all)

RNN_cell(seq, rnn.weight_ih_l0.detach(), rnn.weight_hh_l0.detach())
```

```
RNN_cell(seq, rnn.weight_ih_l0.detach(), rnn.weight_hh_l0.detach(), rnn.bias_ih_
l0.detach(),  rnn.bias_hh_l0.detach())


torch.set_printoptions(sci_mode=False)


# Gradients computation

# Automated software:

#creating sequences and storing them as batches;
seq1 = torch.FloatTensor([[2], [4], [6], [8]]).unsqueeze(0)
seq2 = torch.FloatTensor([[4], [6], [8], [10]]).unsqueeze(0)

batches = torch.cat([seq1, seq2], dim=0)

#spliting data
x_train = batches[:, :-1, :]
y_train = batches[:, -1, :]


#model parameters
input_size = x_train.size(2)
hidden_size = 1
num_layers = 1
output_size = 1


class RNN_m(nn.Module):
  def __init__(self, input_size, hidden_size, num_layers, output_size):
    super(RNN_m, self).__init__()
    self.num_layers = num_layers
    self.hidden_size = hidden_size

    self.rnn = nn.RNN(input_size, hidden_size, num_layers, bias = False, batch_f
irst=True)
    self.linear = nn.Linear(hidden_size, output_size, bias = False)

  def forward(self, x):
    #Setting the initial values of hidden states;
    h0 = torch.zeros(self.num_layers, x.size(0), self.hidden_size)

    #Computation of rnn layer;
    out, _ = self.rnn(x, h0)

    #Extracting the last hidden state for each batch and feeding them to the lin
ear layer;
    #This is where we actually define the type of neural network (many-to-one);
    out = self.linear(out[:, -1, :])
```

```
    return(out)
#setting up the Loss function
criterion = nn.MSELoss(reduction='mean')


#creating an instance of a class "RNN_m";
model = RNN_m(input_size, hidden_size, num_layers, output_size)

#forward pass;
y_hat = model.forward(x_train)

#loss calculation and backpropagation performing;
Loss = criterion(y_hat, y_train).backward()

#Gradients for "Wx", "Wh", "Wy" repectively.
print(model.rnn.weight_ih_l0.grad)
print(model.rnn.weight_hh_l0.grad)
print(model.linear.weight.grad)

#Manually:

#Extraction of weights from the created model;
wx = model.rnn.weight_ih_l0.detach()
wh = model.rnn.weight_hh_l0.detach()
wy = model.linear.weight.detach()


list_grad = torch.empty(0,3)

m = x_train.size(0) #number of batches;

for i in range(2):
  h0 = 0 #default value for initial hidden state in pytorch;
  h1 = torch.tanh(wx * x_train[i,0] + wh*h0)
  h2 = torch.tanh(wx * x_train[i,1] + wh*h1)
  h3 = torch.tanh(wx * x_train[i,2] + wh*h2)

  yhat = wy*h3

  #This type of loss computation looks strange, but explained by the type of RNN
 we are using;
  #We have two predicted values, one for each batch, so the fraction in front of
 the loss function will be 1/2;
  #To avoid redundancy, we could take some part of the equation out of the brack
ets. But the main idea of this code is to perform the exactly the same operation
s, as in the picture above;

  #The only thing, we took out of the brackets is the fraction from the loss fun
ction;
  #I believe, that this representation provides better intuition and blurs the l
ine between theory and practice;
```

```
  #gradinet with respect to "Wy";
  grad_wy = (2*(yhat-y_train[i,0])*h3)/m
  #gradinet with respect to "Wx";
  grad_wx = (2*(yhat-y_train[i,0])*wy*(1-h3**2)*x_train[i,2] + 2*(yhat-
y_train[i,0])*wy*(1-h3**2)*wh*(1-h2**2)*x_train[i,1] + 2*(yhat-
y_train[i,0])*wy*(1-h3**2)*wh*(1-h2**2)*wh*(1-h1**2)*x_train[i,0])/m
  #gradinet with respect to "Wh";
  grad_wh = (2*(yhat-y_train[i,0])*wy*(1-h3**2)*h2 + 2*(yhat-
y_train[i,0])*wy*(1-h3**2)*wh*(1-h2**2)*h1 + 2*(yhat-y_train[i,0])*wy*(1-
h3**2)*wh*(1-h2**2)*wh*(1-h1**2)*h0)/m

  #collecting the gradients
  grad = torch.cat((grad_wx, grad_wh, grad_wy),1)
  list_grad = torch.cat((list_grad, grad), 0)


torch.sum(list_grad, dim = 0).reshape(-1, 1)

#Section 2.2 LSTM layer construction

def LSTM_cell(x_f, wx_f, wh_f, bx_f = None, bh_f = None):

  #dimensionality check
  if len(x_f.size()) == 3:

    #determine the matrices of weights;
    W_ii, W_if, W_ig, W_io = torch.vsplit(wx_f, 4)
    W_hi, W_hf, W_hg, W_ho = torch.vsplit(wh_f, 4)

    #creating a tensor of hidden states and memory cell with respect to the dime
nsion of the coefficient matrix;
    h_data = h_lag = c_lag = torch.zeros((W_hi.size(1), 1))

    #setting the correct shape for biases;
    if None in {bx_f, bh_f}:
      bx_f = bh_f = torch.zeros((wh_f.shape[0], 1))
    else:
      bx_f, bh_f = bx_f.reshape(-1, 1), bh_f.reshape(-1, 1)

    #determine the appropriate matrices of biases;
    b_ii, b_if, b_ig, b_io = torch.vsplit(bx_f.reshape(-1, 1), 4)
    b_hi, b_hf, b_hg, b_ho = torch.vsplit(bh_f.reshape(-1, 1), 4)

    for i in range(x_f.size(1)):
      fi = torch.sigmoid(W_ii @ torch.t(x_f[0][[i], :]) + b_ii + W_hi @ h_lag +
b_hi) #input gate
      ft = torch.sigmoid(W_if @ torch.t(x_f[0][[i], :]) + b_if + W_hf @ h_lag +
b_hf) #forget gate
```

```
      gt = torch.tanh(W_ig @ torch.t(x_f[0][[i], :]) + b_ig + W_hg @ h_lag + b_h
g) #the candidate

      ot = torch.sigmoid(W_io @ torch.t(x_f[0][[i], :]) + b_io + W_ho @ h_lag +
b_ho) #output gate

      c_lag = ft * c_lag + fi * gt #new memory cell
      h_lag = ot * torch.tanh(c_lag) #hidden state estimate
      #data collection
      h_data = torch.cat((h_data, h_lag), 1)

    print(torch.t(h_data))

  else:
    print("The input must be a 3D tensor. Whereas its shape is: ", x_f.shape)


seq = torch.FloatTensor([[3, 4], [4, 5], [5, 6]]).unsqueeze(0)
seq = torch.FloatTensor([[3], [4], [5]]).unsqueeze(0)


#This time for each hidden layer we would compute two hidden states;
lstm = nn.LSTM(input_size=seq.shape[2], hidden_size=1, num_layers = 1, bias = Tr
ue, batch_first=True)


#here we get the hidden states, estimated for each value in the sequence;
out_all, out_last = lstm(seq)
print(out_all)


LSTM_cell(seq, lstm.weight_ih_l0.detach(), lstm.weight_hh_l0.detach())
LSTM_cell(seq, lstm.weight_ih_l0.detach(), lstm.weight_hh_l0.detach(), lstm.bias
_ih_l0.detach(), lstm.bias_hh_l0.detach())


#Section 2.3 LSTM layer construction

def GRU_cell(x_f, wx_f, wh_f, bx_f = None, bh_f = None):

  #dimensionality check
  if len(x_f.size()) == 3:

    #determine the appropriate matrices for the equations;
    wx1 , wx2, wx3 = torch.vsplit(wx_f, 3)
    wr, wz, wn = torch.vsplit(wh_f, 3)

    #creating a tensor of hidden states with respect to the dimension of the coe
fficient matrix;
    h_data = h_lag = torch.zeros((wr.size(1), 1))
```

```
    #setting the correct shape for biases;
    if None in {bx_f, bh_f}:
      bx_f = bh_f = torch.zeros((wh_f.shape[0], 1))
    else:
      bx_f, bh_f = bx_f.reshape(-1, 1), bh_f.reshape(-1, 1)

    #split biases according to the equations;
    bx1, bx2, bx3 = torch.vsplit(bx_f, 3)
    bh1, bh2, bh3 = torch.vsplit(bh_f, 3)

    for i in range(x_f.size(1)):
      rt = torch.sigmoid(wx1 @ torch.t(x_f[0][[i],:]) + bx1 + wr @ h_lag + bh1)
#reset gate;
      zt = torch.sigmoid(wx2 @ torch.t(x_f[0][[i],:]) + bx2 + wz @ h_lag + bh2)
#update gate;

      nt = torch.tanh(wx3 @ torch.t(x_f[0][[i],:]) + bx3 + rt * (wn @ h_lag  + b
h3)) #hidden state candidate;
      h_lag = (1 - zt) * nt + zt * h_lag #hidden state itself;
      #data collection
      h_data = torch.cat((h_data, h_lag), 1)

    print(torch.t(h_data))

  else:
    print("The input must be a 3D tensor. Whereas its shape is: ", x_f.shape)


seq = torch.FloatTensor([[3, 4], [4, 5], [5, 6]]).unsqueeze(0)
seq = torch.FloatTensor([[3], [4], [5]]).unsqueeze(0)


#This time for each hidden layer we would compute two hidden states;
gru = nn.GRU(input_size=seq.shape[2], hidden_size=1, num_layers = 1, bias = True
, batch_first=True)


#here we get the hidden states, estimated for each value in the sequence;
out_all, out_last = gru(seq)
print(out_all)


GRU_cell(seq, gru.weight_ih_l0.detach(), gru.weight_hh_l0.detach())
GRU_cell(seq, gru.weight_ih_l0.detach(), gru.weight_hh_l0.detach(), gru.bias_ih_
l0.detach(), gru.bias_hh_l0.detach())
```

Додаток «С»

Section 1.3 Time series modeling using neural networks

```
 import torch
import torch.nn as nn
import yfinance as yf #If not installed: !pip install yfinance
import pandas as pd
import numpy as np
import time
import math
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler
import seaborn as sns


data = yf.download('MSFT', start='2018-03-17', end='2020-01-25', progress=False)
data.head()


price = data[['Close']].values
print("shape: ", price.shape)

plt.plot(range(price.shape[0]), price[:, 0], label = "line 2")
plt.legend()
plt.show()


#Taking a log of initial series;
log_p = np.log(price)


#Normalize the data with respect to the given range;
scaler = MinMaxScaler(feature_range=(-1,1))
stock_adj = scaler.fit_transform(log_p)


plt.plot(range(len(stock_adj)), stock_adj, label = "Transformed series")
plt.legend()
plt.show()


def data_split(stock, input_size, seq_len): #data itself | input_size | number o
f sequences in batches (sequence_length);

  #step 1: creating an array of lagged values;
  data = stock

  for i in np.arange(-1, -(input_size + 1), -1):
    data = np.c_[data, np.roll(stock, i)]

  data = data[:-input_size, :]
```

```
  #step 2: spliting this array on batches;
  batches = []
  num_batches = len(data) - seq_len + 1

  for epoch in range(num_batches):
    batches.append(data[epoch:epoch + seq_len, :])

  batches = np.array(batches).astype(np.float32)

  #step 3: splitting batches on train and test sets;
  test_size = int(np.round(0.2 * batches.shape[0]))
  train_size = batches.shape[0] - test_size

  x_train = batches[:train_size, :, :-1]
  y_train = batches[:train_size, -1, -1].reshape(-1, 1)

  x_test = batches[train_size:, :, :-1]
  y_test = batches[train_size:, -1, -1].reshape(-1, 1)

  return x_train, y_train, x_test, y_test


#This list of features is needed to set up the construction of our networks;
sequence_length = 10 #related only to the creation of batches;
input_size = 2
hidden_size = 20
num_layers = 2
output_size = 1
num_epochs = 200


x_train, y_train, x_test, y_test = data_split(stock_adj, input_size, sequence_le
ngth)


x_train.shape
y_train.shape


#converting numpy arrays into torch tensors;
xt_train = torch.from_numpy(x_train)
yt_train = torch.from_numpy(y_train)

xt_test = torch.from_numpy(x_test)
yt_test = torch.from_numpy(y_test)
```

```
#RNN model

#converting numpy arrays into torch tensors;
class RNN_m(nn.Module):
  def __init__(self, input_size, hidden_size, num_layers, output_size):
    super(RNN_m, self).__init__()
    self.num_layers = num_layers
    self.hidden_size = hidden_size

    self.rnn = nn.RNN(input_size, hidden_size, num_layers, batch_first=True)
    self.linear = nn.Linear(hidden_size, output_size)

  def forward(self, x):
    #Setting the initial values of hidden states;
    h0 = torch.zeros(self.num_layers, x.size(0), self.hidden_size)

    #Computation of rnn layer;
    out, _ = self.rnn(x, h0)

    #Extracting the last hidden state for each batch and feeding them to the lin
ear layer;
    #This is where we actually define the type of neural network (many-to-one);
    out = self.linear(out[:, -1, :])

    return(out)


#Creating an instance of a class "RNN_m" - our RNN model;
model_0 = RNN_m(input_size, hidden_size, num_layers, output_size)
#setting the loss function;
criterion = nn.MSELoss(reduction='mean')
#optimization algorithm;
optimiser = torch.optim.Adam(model_0.parameters(), lr = 0.01)


#array for loss collection;
rnn_loss = np.zeros(num_epochs)

start_time = time.time()

for t in range(num_epochs):
  #forward path;
  y_hat = model_0.forward(xt_train)
  #MSE computation;
  loss = criterion(y_hat, yt_train)

  #collecting the loss;
  with torch.no_grad():
    #print("Epoch: ", t, "MSE: ", loss.item())
    rnn_loss[t] = loss.item()
```

```
  #cleaning the gradients;
  optimiser.zero_grad()
  #backward path;
  loss.backward()
  #weights update;
  optimiser.step()

#time of training;
training_time = time.time() - start_time
print("Training time: {}".format(training_time))


#Estimates for training set: original | prediction ;
rnn_train = torch.cat((yt_train, y_hat.detach()), 1)


sns.set_style("darkgrid")


fig = plt.figure()
fig.subplots_adjust(hspace=0.2, wspace=0.2)

#
plt.subplot(1, 2, 1)
ax = sns.lineplot(x = range(rnn_train.size(0)), y = rnn_train[:, 0], label="Orig
inal series", color='royalblue')
ax = sns.lineplot(x = range(rnn_train.size(0)), y = rnn_train[:, 1], label="Trai
ning Prediction (RNN)", color='tomato')
ax.set_title('Stock price', size = 14, fontweight='bold')
ax.set_xlabel("Period, t", size = 14)
ax.set_ylabel("Time series", size = 14)
ax.set_xticklabels('', size=10)

#
plt.subplot(1, 2, 2)
ax2 = sns.lineplot(data=rnn_loss, color='royalblue')
ax2.set_xlabel("Epoch", size = 14)
ax2.set_ylabel("Loss", size = 14)
ax2.set_title("Training Loss", size = 14, fontweight='bold')
fig.set_figheight(6)
fig.set_figwidth(16)


#Test set prediction computation;
y_hat_test = model_0(xt_test).detach()
#Estimates for testing set: original | prediction ;
rnn_test = torch.cat((yt_test, y_hat_test), 1)


plt.plot(range(len(yt_test)), yt_test, label = "Original series")
```

```
plt.plot(range(len(y_hat_test)), y_hat_test, label = "RNN prediction")
plt.legend()
plt.show()


lstm_rmse_train = math.sqrt(criterion(rnn_train[:, 1], rnn_train[:, 0]))
print("RNN, train set: ", lstm_rmse_train, "RMSE")
lstm_rmse_test = math.sqrt(criterion(rnn_test[:, 1], rnn_test[:, 0]))
print("RNN, test set: ",lstm_rmse_test, "RMSE")


#LSTM model
class LSTM_m(nn.Module):
  def __init__(self, input_size, hidden_size, num_layers, output_size):
    super(LSTM_m, self).__init__()
    self.num_layers = num_layers
    self.hidden_size = hidden_size

    self.lstm = nn.LSTM(input_size, hidden_size, num_layers, batch_first=True)
    self.linear = nn.Linear(hidden_size, output_size)

  def forward(self, x):
    h0 = torch.zeros(self.num_layers, x.size(0), self.hidden_size)
    c0 = torch.zeros(self.num_layers, x.size(0), self.hidden_size)

    out, (hn, cn) = self.lstm(x, (h0, c0))

    out = self.linear(out[:, -1, :])

    return out


model_1 = LSTM_m(input_size, hidden_size, num_layers, output_size)
criterion = nn.MSELoss(reduction='mean')
optimiser = torch.optim.Adam(model_1.parameters(), lr = 0.01)


lstm_loss = np.zeros(num_epochs)
#
start_time = time.time()

for t in range(num_epochs):
  #forward path;
  y_hat = model_1.forward(xt_train)
  #MSE computation;
  loss = criterion(y_hat, yt_train)

  #collecting the loss;
  with torch.no_grad():
    #print("Epoch: ", t, "MSE: ", loss.item())
    lstm_loss[t] = loss.item()
```

```
  #cleaning the gradients;
  optimiser.zero_grad()
  #backward path;
  loss.backward()
  #weights update;
  optimiser.step()

training_time = time.time() - start_time
print("Training time: {}".format(training_time))


#Estimates for training set: original | prediction ;
lstm_train = torch.cat((yt_train, y_hat.detach()), 1)


fig = plt.figure()
fig.subplots_adjust(hspace=0.2, wspace=0.2)

#
plt.subplot(1, 2, 1)
ax = sns.lineplot(x = range(lstm_train.size(0)), y = lstm_train[:, 0], label="Or
iginal series", color='royalblue')
ax = sns.lineplot(x = range(lstm_train.size(0)), y = lstm_train[:, 1], label="Tr
aining Prediction (LSTM)", color='tomato')
ax.set_title('Stock price', size = 14, fontweight='bold')
ax.set_xlabel("Period, t", size = 14)
ax.set_ylabel("Time series", size = 14)
ax.set_xticklabels('', size=10)

#
plt.subplot(1, 2, 2)
ax2 = sns.lineplot(data=lstm_loss, color='royalblue')
ax2.set_xlabel("Epoch", size = 14)
ax2.set_ylabel("Loss", size = 14)
ax2.set_title("Training Loss", size = 14, fontweight='bold')
fig.set_figheight(6)
fig.set_figwidth(16)


#Test set prediction computation;
y_hat_test = model_1(xt_test).detach()
#Estimates for testing set: original | prediction ;
lstm_test = torch.cat((yt_test, y_hat_test), 1)


plt.plot(range(len(yt_test)), yt_test, label = "Original series")
plt.plot(range(len(y_hat_test)), y_hat_test, label = "LSTM prediction")
plt.legend()
plt.show()
```

```
lstm_rmse_train = math.sqrt(criterion(lstm_train[:, 1], lstm_train[:, 0]))
print("LSTM, train set: ", lstm_rmse_train, "RMSE")
lstm_rmse_test = math.sqrt(criterion(lstm_test[:, 1], lstm_test[:, 0]))
print("LSTM, test set: ",lstm_rmse_test, "RMSE")


#GRU model
class GRU_m(nn.Module):
  def __init__(self, input_size, hidden_size, num_layers, output_size):
    super(GRU_m, self).__init__()
    self.num_layers = num_layers
    self.hidden_size = hidden_size

    self.gru = nn.GRU(input_size, hidden_size, num_layers, batch_first=True)
    self.linear = nn.Linear(hidden_size, output_size)

  def forward(self, x):
    h0 = torch.zeros(self.num_layers, x.size(0), self.hidden_size)

    out, hn = self.gru(x, h0)

    out = self.linear(out[:, -1, :])

    return out

model_2 = GRU_m(input_size, hidden_size, num_layers, output_size)
criterion = nn.MSELoss(reduction='mean')
optimiser = torch.optim.Adam(model_2.parameters(), lr = 0.01)

gru_loss = np.zeros(num_epochs)
#
start_time = time.time()

for t in range(num_epochs):
  #forward path;
  y_hat = model_2.forward(xt_train)
  #MSE computation;
  loss = criterion(y_hat, yt_train)

  #collecting the loss;
  with torch.no_grad():
    #print("Epoch: ", t, "MSE: ", loss.item())
    gru_loss[t] = loss.item()

  #cleaning the gradients;
  optimiser.zero_grad()
  #backward path;
  loss.backward()
  #weights update;
  optimiser.step()
```

```
training_time = time.time() - start_time
print("Training time: {}".format(training_time))

#Estimates for training set: original | prediction ;
gru_train = torch.cat((yt_train, y_hat.detach()), 1)

fig = plt.figure()
fig.subplots_adjust(hspace=0.2, wspace=0.2)

#
plt.subplot(1, 2, 1)
ax = sns.lineplot(x = range(gru_train.size(0)), y = gru_train[:, 0], label="Orig
inal series", color='royalblue')
ax = sns.lineplot(x = range(gru_train.size(0)), y = gru_train[:, 1], label="Trai
ning Prediction (GRU)", color='tomato')
ax.set_title('Stock price', size = 14, fontweight='bold')
ax.set_xlabel("Period, t", size = 14)
ax.set_ylabel("Time series", size = 14)
ax.set_xticklabels('', size=10)

#
plt.subplot(1, 2, 2)
ax2 = sns.lineplot(data=gru_loss, color='royalblue')
ax2.set_xlabel("Epoch", size = 14)
ax2.set_ylabel("Loss", size = 14)
ax2.set_title("Training Loss", size = 14, fontweight='bold')
fig.set_figheight(6)
fig.set_figwidth(16)

#Test set prediction computation;
y_hat_test = model_2(xt_test).detach()
#Estimates for testing set: original | prediction ;
gru_test = torch.cat((yt_test, y_hat_test), 1)

plt.plot(range(len(yt_test)), yt_test, label = "Original series")
plt.plot(range(len(y_hat_test)), y_hat_test, label = "GRU prediction")
plt.legend()
plt.show()

gru_rmse_train = math.sqrt(criterion(gru_train[:, 1], gru_train[:, 0]))
print("LSTM, train set: ", lstm_rmse_train, "RMSE")
lstm_rmse_test = math.sqrt(criterion(gru_test[:, 1], gru_test[:, 0]))
print("LSTM, test set: ",lstm_rmse_test, "RMSE")

#Predictions

def batch_for_pred(stock_f, input_size_f , seq_len_f):
  data = stock_f[-seq_len_f:]
```

```
  for i in range(1):
    data = np.c_[stock_f[-(seq_len_f+i+1):][:-(i+1)], data]

  batch_pred = torch.from_numpy(data.astype(np.float32)).unsqueeze(0)

  return batch_pred


#using batch_for_pred() we form the batch need to make a forecast. In our case i
t is t+1;
X = batch_for_pred(stock_adj, 2, 10)
X


#forecast one period ahead, each model;
t1_rnn = model_0(X).detach()
t1_lstm = model_1(X).detach()
t1_gru = model_2(X).detach()

print(t1_rnn, t1_lstm, t1_gru)


t1_rnn = np.exp(scaler.inverse_transform(t1_rnn))
t1_lstm = np.exp(scaler.inverse_transform(t1_lstm))
t1_gru = np.exp(scaler.inverse_transform(t1_gru))

print(t1_rnn, t1_lstm, t1_gru)
```

Додаток «D»

Classical machine learning models - Section 3.2

```
---
title: "Untitled"
runtime: shiny
output:
 flexdashboard::flex_dashboard:
  orientation: columns
  vertical_layout: scroll
  source_code: embed
---

```{r setup, include=FALSE}
library(flexdashboard)
library(shiny)
library(shinyjs)
library(shinyalert)
library(lobstr)
library(htmltools)
library(readr)
library(magrittr)
library(DT)
library(ggplot2)
library(plotly)
library(quantmod)
library(forecast)
library(tseries)
library(fGarch)
library(rugarch)
```

PAGE1 {data-icon="fa-table"}

Column_1 {.sidebar}
```

```{r}
useShinyjs(rmd = TRUE)


fileInput(inputId="Upload_1", label="Choose the file", multiple = FALSE, accept = ".csv")


textInput(inputId="Upload_2", label="Symbol", value="")
dateRangeInput("Dates_set", label = h3("Date range"), start = Sys.Date()-800, end = Sys.Date())


actionButton("reset", label = "Reset", width="80px")
actionButton("submit", label = "Submit", class="btn btn-primary")
```


```{r}
observeEvent(input$submit11,{
  shinyjs::reset("Dates_plot")
  value$Date_min<-NULL
  value$Date_max<-NULL
})
```


```{r}
#evaluation featres of isTruthy() - https://cran.r-project.org/web/packages/shiny/shiny.pdf
observeEvent(input$submit,{


  if (fileCheck() && input$Upload_2=="") {


      Import1<-read_csv(input$Upload_1$datapath)
    if(ncol(Import1)==1){
      Import1<-read_csv2(input$Upload_1$datapath)
    }
      value$data<-as.data.frame(Import1)
  }
```

```
  else if (isFALSE(fileCheck()) && isTruthy(input$Upload_2)==TRUE){

    tryCatch({
      if(is.na(input$Dates_set[1]) || is.na(input$Dates_set[2])){
        Import2<-getSymbols(input$Upload_2, src="yahoo", auto.assign = FALSE)
      }
      else{
        Import2<-getSymbols(input$Upload_2, from=input$Dates_set[1],
to=input$Dates_set[2]+1,  src="yahoo", auto.assign = FALSE)
      }

      value$data<-data.frame(Date=index(Import2), coredata(Import2)) },

    error = function(c) {runjs('alert("Error: symbol does not exist.");')},
    finally={gc()}
    )
  }
  else {
    erase()
  }
})
```

```{r}
observeEvent(input$reset,{erase()})
```

```{r}
erase<-function() {
    reset("Upload_1")
    reset("Upload_2")
    value$addr_old <- value$addr_new
    runjs('alert("Controls reset!");')
}
```

```{r}
#Data storage
value <- reactiveValues(
  data = NULL,
  addr_old=NULL,
  addr_new=NULL,
  DateColInd=NULL,
  Date_min=NULL,
  Date_max=NULL,
  choise=NULL,
  series=NULL
)
```

```{r}
#block of code, that activates once. Needed for fileInput control reset
observeEvent(handlerExpr=NULL,{
  value$addr_old<-obj_addr(input$Upload_1)}, once = TRUE)
```

```{r}
#Here we check, whether fileInput has been updated.
fileCheck<-eventReactive(c(input$Upload_1, value$addr_old),{

  value$addr_new <- obj_addr(input$Upload_1)

  if(value$addr_new == value$addr_old) {FALSE}
  else {
    value$addr_old == value$addr_new
    TRUE
  }
})
```

```r
#Initialization block
observeEvent(value$data,{
  req(is.data.frame(value$data)==TRUE)

  value$data<-na.omit(value$data) #modify a bit

  is.date <- function(x) inherits(x, 'Date')
  index <- sapply(value$data, is.date) %>% which

  if(length(index) == 1) {
    value$Date_min <- value$data[1,index]
    value$Date_max <- value$data[nrow(value$data),index]
    value$DateColInd<-index
  }
  else {

    is.convertible.to.date <- function(x) !anyNA(as.Date(as.character(x), tz =
'UTC', format = '%Y-%m-%d'))
    index <- sapply(value$data, is.convertible.to.date) %>% which

    if(length(index) != 1) {

      ### make an alert
    }
    else {
      value$Date_min <- value$data[1,index]
      value$Date_max <- value$data[nrow(value$data),index]
      value$DateColInd<-index
    }
  }

  updateDateRangeInput(session, "Dates_plot",
    start= value$Date_min,
    end= value$Date_max
```

```
   )


  value$choise <- as.list(colnames(value$data)[unlist(lapply(value$data,
is.numeric), use.names = FALSE)])


  updateSelectizeInput(session, "select", choices = value$choise, selected =
NULL, options = list(), server = FALSE)


  updateSelectizeInput(session, "finalSelect", choices = value$choise, selected
= NULL, options = list(), server = FALSE)


})


```


```{r}
observe({
  req(is.null(value$data)==FALSE) #TryCatch - не работает, необходио дописать!


    if (input$Finaldates[1]<input$Finaldates[2]){
      colIndex <- colnames(value$data) %>% grepl(input$finalSelect,.) %>% which
      value$series <-
value$data[(value$data[,value$DateColInd]>=input$Finaldates[1] &
value$data[,value$DateColInd]<=input$Finaldates[2]), colIndex]
    } else {
      colIndex <- colnames(value$data) %>% grepl(input$finalSelect,.) %>% which
      value$series <- value$data[ , colIndex]
    }
})
```


Column_2

Time series plot

```r
fillCol(flex=c(8.5,1.5),
  plotlyOutput("MainPlot", height="100%"),

  fillRow(
    dateRangeInput("Dates_plot", label = "Date range"),
    selectizeInput("select", label = "Select box",
    choices = NULL,
    selected = 1, options=list(dropdownParent="body"))
  )
)
```

```r
output$MainPlot <-renderPlotly({

  req(is.data.frame(value$data), isTruthy(value$choise))
  p<-ggplot(value$data)+aes_string(x=value$data[,value$DateColInd], y=
input$select) + coord_cartesian(xlim =
c(input$Dates_plot[1],input$Dates_plot[2])) + geom_line()
  plotly::ggplotly(p)

})
```

Data transformation panel

```r
fillRow(
  selectizeInput("finalSelect", label = h3("Series to analyze"),
      choices = NULL,
      selected = 1, options=list(dropdownParent="body")),

  dateRangeInput("Finaldates", label = h3("Range")), height = "100px"
)
```

```
checkboxGroupInput("checkGroup", label = h3("Checkbox group"),
    choices = list("Outlier" = 1, "Round" = 2, "Log" = 3),
    selected = 1)

```
```{r}
observe({
   updateDateRangeInput(session, "Finaldates",
    start= value$Date_min,
    end= value$Date_max
 )
})

```

Datatable
```{r}
output$DataTable<- renderDataTable(

  datatable(value$data, filter="top") #%>%
    #formatRound(colnames(value$data)[unlist(lapply(value$data, is.numeric),
use.names = FALSE)], 2)
)
```

```{r}
dataTableOutput('DataTable')
```

```{r}
output$a111<- renderPrint({mData$Tseries})
verbatimTextOutput("a111")
```

```
# ARIMA/ARCH/GARCH


```{r}
mData <- reactiveValues(
  Tseries=NULL,
  arima=NULL,
  p=NULL,
  q=NULL,
  d=NULL,
  adf=NULL,
  diffSeries=NULL,
  LMtestTable=NULL,
  ArchModels=NULL,
  crARCH=NULL,
  GarchModels=NULL,
  crGARCH=NULL,
  TGarchModels=NULL,
  crTGARCH=NULL,
  MGarchModels=NULL,
  crMGARCH=NULL
)
```


```{r}
actionButton("button1", "Analyze")
```

Foreword

In financial time series, it is often that the series is transformed by logging and then the differencing is performed.People often look at the returns of the stock rather than the its prices. Differences of log prices represent the returns and are similar to percentage changes of stock prices.

Estimated Arima model


```{r}
```
```

```
observeEvent(input$button1,{
  req(is.null(value$series)==FALSE)
  mData$Tseries <- ts(value$series)

  if(any(input$checkGroup %>% grepl('^1$',.))) mData$Tseries <-
tsclean(mData$Tseries)
  if(any(input$checkGroup %>% grepl('^2$',.))) mData$Tseries <-
round(mData$Tseries,2)
  if(any(input$checkGroup %>% grepl('^3$',.))) mData$Tseries <-
log(mData$Tseries)

  mData$arima<-auto.arima(mData$Tseries, ic="aic", trace=TRUE)
})
```

```{r}
output$EstimatedArima<- renderPrint({mData$arima})
verbatimTextOutput("EstimatedArima")
```

Stationarity and differencing of time series data set:

Check of stationarity:
Ho: series is non-stationary (p-value > 0.05)
H1: series is stationary   (p-value < 0.05)

```{r}
#calculations block !!!!!

observeEvent(mData$arima,{

#Getting the orders of arima model
mData$p <- mData$arima$arma[1]
mData$q <- mData$arima$arma[2]
mData$d <- mData$arima$arma[length(mData$arima$arma)-1]
```

```
if(mData$d ==0){
  mData$adf<-adf.test(mData$Tseries, alternative = "stationary")
}
else {

  i=0
  for(i in mData$d-1){
  mData$Tseries<-diff(mData$Tseries, differences = 1)
  mData$adf<-adf.test(mData$Tseries, alternative = "stationary")
  }
}
mData$diffSeries<-mData$Tseries
})
```

```{r}
output$AdfTest<- renderPrint({ mData$adf })
#output$Tseries<- renderPrint({ mData$Tseries }) #nigde net
```

```{r}
verbatimTextOutput("AdfTest")
```

Series after differencing

```{r}
output$diffplot <-renderPlotly({
  req(is.null(mData$diffSeries)==FALSE)
  diff<-ggplot()+aes_string(x=1:length(mData$diffSeries), y= mData$diffSeries) +
xlab("diff series") + geom_line()
  plotly::ggplotly(diff)
})
```

```{r}
```

```
plotlyOutput("diffplot", height="100%")
```
```

### Residuals plot: ACF & PACF

Observing residual plot and it`s ACF & PACF diagram

If ACF & PACF of the model residuals show no significant lags, the selected model is appropriate.

### ACF plot

```{r}
plotOutput(outputId = "ACF")

output$ACF<- renderPlot({
  req(is.null(mData$diffSeries)==FALSE)
  acf(mData$diffSeries)
})

```

PACF plot

```{r}
plotOutput(outputId = "PACF")

output$PACF<- renderPlot({
  req(is.null(mData$diffSeries)==FALSE)
  pacf(mData$diffSeries)
})
```

Ljung-Box test

Ljung-Box is a test of autocorrelation in which it verifies whether the autocorrelations of a time series are different from 0.

More formally, the Ljung-Box test can be defined as follows:
```

```
H0:   The data are random. - independent and uncorrelated;<br>

H1:   The data are not random. - remains serial correlation;<br>


Note: If the p-value is greater than 0.05 then the residuals are independent
which we want for the model to be correct.<br>


```{r}

output$LBtest <- renderPrint({

  req(is.null(mData$arima)==FALSE)

  Box.test(residuals(mData$arima),lag=12, type="Ljung-Box")

})

verbatimTextOutput("LBtest")

```


Diagnostic checking

1.1) Observing residual plot and it`s ACF & PACF diagram

If ACF & PACF of the model residuals show no significant lags, the selected
model is appropriate.

```{r}

plotOutput(outputId = "ModelResiduals")


output$ModelResiduals<- renderPlot({

  req(is.null(mData$arima)==FALSE)

  tsdisplay(residuals(mData$arima), lag.max=15, main='Model Residuals')

})

```


ARIMA forecast


```{r}

observeEvent(mData$arima,{


  ArimaPred<-as.data.frame(forecast(mData$arima, h=4))


  if (any(input$checkGroup %>% grepl('^3$',.))){
```
```

```
    mData$ArimaPred<-exp(ArimaPred)
  } else {
    mData$ArimaPred<-ArimaPred
  }
})

```

```{r}
output$ArimaPrediction <- renderDataTable(

  datatable(mData$ArimaPred, filter="top")
)
```

```{r}
dataTableOutput('ArimaPrediction')
```

Remark

Although ACF and PACF of residuals have no significant lags, the time series plot of residuals shows some cluster of volatility.

It is important to note that ARIMA is a method to @linear model the data and the forecast width remains constant because the model does not reflect recent changes or incorporate new information.

In other words, it provides best linear forecasts for the series, and thus plays little role in forecasting model nonlinearly.

In order to model volatility, ARCH/GARCH method is used.

Testing for ARCH effect

1) Firstly, check if residual plot displays any cluster of volatility.

2) Observe the squared residual plot; Are there any volatility clusters?

3) Observe ACF & PACF of squared residuals.

```{r}
plotOutput(outputId = "ArchEffect")


output$ArchEffect<- renderPlot({
  req(is.null(mData$arima)==FALSE)
  tsdisplay(residuals(mData$arima)^2, lag.max=15, main='Model Residuals')
})
```

Remark

A strict white noise cannot be predicted either linearly or nonlinearly while general white noise might not be predicted linearly yet done so nonlinearly. If the residuals are strict white noise, they are independent with zero mean, normally distributed, and ACF & PACF of squared residuals displays no significant lags.

LM test

As an alternative to Engle`s ARCH test, you can check for serial dependence (ARCH effects) in a residual series by conducting a Ljung-Box Q-test on the first m lags of the squared residual series, where m = P + Q, ARCH(P*Q), GARCH(P,Q).

```{r}
observe({
  req(is.null(mData$arima)==FALSE)


  LMtestTable <- data.frame(matrix(ncol = 5, nrow = 1))
  for(i in 1:5){
    u <-Box.test(residuals(mData$arima)^2,lag=i, type="Ljung-Box")
    LMtestTable[1,i]<-u$p.value
  }


  colnames(LMtestTable) <- c("lag 1","lag 2","lag 3","lag 4", "lag 5")


  mData$LMtestTable <- LMtestTable
})
```

```
```{r}
output$TestResultTable <- renderDataTable(
  datatable(mData$LMtestTable)
)
```


```{r}
dataTableOutput('TestResultTable')
```


ARCH notes
The general rules for ARCH varience cofficients are following:
ht = a0 + a1 e^2 + a2 e^2 .., a0>0, 0< (sum of ai) <=1

ARCH model selection


```{r}
observe({
req(is.null(mData$LMtestTable)==FALSE)


list1<-list()
list2<-list()
  for(i in 1:4){
    t<-garchFit(substitute(~arma(p,q)+garch(i,0),list(p=mData$p, q=mData$q,
i=i)), data=mData$Tseries)
    t1<-attributes(t)$fit$ics
    list1[[i]]<-t
    list2[[i]]<-t1
  }


  mData$ArchModels<-list1
  #mData$crARCH<-list2
  ctr<-as.data.frame(list2)
  ctr<-as.data.frame(t(ctr))
```
```

```
  rownames(ctr)<-c("ARIMA-Arch(1)", "ARIMA-Arch(2)", "ARIMA-Arch(3)", "ARIMA-
Arch(4)")
  mData$crARCH<-ctr
})
```

```{r}
output$ArchModelslist<- renderPrint({ mData$ArchModels
})
verbatimTextOutput("ArchModelslist")
```

ARCH models estimated criteria

```{r}
output$ARCH_CR <- renderDataTable(
  datatable(mData$crARCH)
)
```

```{r}
dataTableOutput('ARCH_CR')
```

GARCH model selection

```{r}
observe({
req(is.null(mData$LMtestTable)==FALSE)

list3<-list()
list4<-list()
z=1

for(i in 1:2){
  for(j in 1:2){
```

```
    garchSpec <- ugarchspec(
                variance.model=list(model="sGARCH",
                                    garchOrder=c(i,j)),
                mean.model=list(armaOrder=c(mData$p,mData$q)),
                distribution.model="std")
    garchFit <- ugarchfit(spec=garchSpec, data=mData$Tseries)

    t3<-garchFit
    t4<-infocriteria(garchFit)

    list3[[z]]<-t3
    list4[[z]]<-t4

    z=z+1
    }
}

mData$GarchModels<-list3

ctrr<-as.data.frame(list4)
colnames(ctrr)<-c("ARIMA-Garch(1,1)", "ARIMA-Garch(1,2)", "ARIMA-Garch(2,1)",
"ARIMA-Garch(2,2)")
mData$crGARCH<-ctrr

})
```

```{r}
output$GarchModelslist<- renderPrint({ mData$GarchModels
})
verbatimTextOutput("GarchModelslist")
```

GARCH models estimated criteria

```
```{r}
output$GARCH_CR <- renderDataTable(
  datatable(mData$crGARCH)
)
```


```{r}
dataTableOutput('GARCH_CR')
```


TGARCH model selection


```{r}
observe({
req(is.null(mData$LMtestTable)==FALSE)

list3<-list()
list4<-list()
z=1

for(i in 1:2){
  for(j in 1:2){
    garchSpec <- ugarchspec(
                variance.model=list(model="fGARCH",
                                    garchOrder=c(i,j),
                                    submodel="TGARCH"),
                mean.model=list(armaOrder=c(mData$p,mData$q)),
                distribution.model="std")
    garchFit <- ugarchfit(spec=garchSpec, data=mData$Tseries)

    t3<-garchFit
    t4<-infocriteria(garchFit)

    list3[[z]]<-t3
```
```

```
    list4[[z]]<-t4


    z=z+1
    }
}


mData$TGarchModels<-list3


ctrr<-as.data.frame(list4)
colnames(ctrr)<-c("ARIMA-TGarch(1,1)", "ARIMA-TGarch(1,2)", "ARIMA-TGarch(2,1)",
"ARIMA-TGarch(2,2)")
mData$crTGARCH<-ctrr


})
```


```{r}
output$TGarchModelslist<- renderPrint({ mData$TGarchModels
})
verbatimTextOutput("TGarchModelslist")
```


TGARCH models estimated criteria


```{r}
output$TGARCH_CR <- renderDataTable(
  datatable(mData$crTGARCH)
)
```


```{r}
dataTableOutput('TGARCH_CR')
```


GARCH-in-Mean model selection

```{r}
observe({
req(is.null(mData$LMtestTable)==FALSE)

list3<-list()
list4<-list()
z=1

for(i in 1:2){
  for(j in 1:2){
    garchSpec <- ugarchspec(
                variance.model=list(model="fGARCH",
                                    garchOrder=c(i,j),
                                    submodel="APARCH"),
                mean.model=list(armaOrder=c(mData$p,mData$q),
                                include.mean=TRUE,
                                archm=TRUE,
                                archpow=2),
                distribution.model="std")
    garchFit <- ugarchfit(spec=garchSpec, data=mData$Tseries)

    t3<-garchFit
    t4<-infocriteria(garchFit)

    list3[[z]]<-t3
    list4[[z]]<-t4

    z=z+1
    }
}

mData$MGarchModels<-list3
```

```
ctrr<-as.data.frame(list4)
colnames(ctrr)<-c("ARIMA-MGarch(1,1)", "ARIMA-MGarch(1,2)", "ARIMA-MGarch(2,1)",
"ARIMA-MGarch(2,2)")
mData$crMGARCH<-ctrr

})
```

```{r}
output$MGarchModelslist<- renderPrint({ mData$MGarchModels
})
verbatimTextOutput("MGarchModelslist")
```

GARCH-In-Mean models estimated criteria

```{r}
output$MGARCH_CR <- renderDataTable(
  datatable(mData$crMGARCH)
)
```

```{r}
dataTableOutput('MGARCH_CR')
```

ARNN

```{r}
netData <- reactiveValues(
  Seriess=NULL,
  modelfit=NULL,
  NNF=NULL,
  fcast=NULL
)
```

```{r}
actionButton("button2", "Analyze")
```

```{r}
observeEvent(input$button2,{
  req(is.null(value$series)==FALSE)
  netData$Seriess <- ts(value$series)

  if(any(input$checkGroup %>% grepl('^1$',.))) netData$Seriess <- tsclean(netData$Seriess)
  if(any(input$checkGroup %>% grepl('^2$',.))) netData$Seriess <- round(netData$Seriess,2)
  if(any(input$checkGroup %>% grepl('^3$',.))) netData$Seriess <- log(netData$Seriess)

  netData$modelfit <- nnetar(netData$Seriess, repeats = 20, lambda=0.5)
})
```

Foreword

Feed-forward neural networks with a single hidden layer and lagged inputs for forecasting univariate time series.The nnetar function in the forecast package for R fits a neural network model to a time series with lagged values of the time series as inputs (and possibly some other exogenous inputs). So it is a nonlinear autogressive model, and it is not possible to analytically derive prediction intervals. Therefore we use simulation.The neural networks is fit by the function:

nnetar(y, p, P = 1, size, repeats = 20, xreg = NULL, lambda = NULL, model = NULL, subset = NULL, scale.inputs = TRUE, x = y, …)

```{r}
output$ResultNet<- renderPrint({ netData$modelfit
})
verbatimTextOutput("ResultNet")
```

```
```

ARNN Forecast

```{r}
observeEvent(netData$modelfit,{
  req((is.null(netData$modelfit)==FALSE))

  if (any(input$checkGroup %>% grepl('^3$',.))){
    netData$NNF <- forecast(netData$modelfit, PI=TRUE, h=4)
    netData$fcast <- exp(as.data.frame(netData$NNF))
  } else {
    netData$NNF <- forecast(netData$modelfit, PI=TRUE, h=4)
    netData$fcast <- as.data.frame(netData$NNF)
  }

})

```
```{r}
output$ARNNForecast <- renderDataTable(
  datatable(netData$fcast)
)
```
```{r}
dataTableOutput('ARNNForecast')
```

Forecast plot

```{r}
plotOutput(outputId = "netPlot")
```

```{r}
```

```
output$netPlot <- renderPlot({
  req(is.null(netData$NNF)==FALSE)
  forecast::autoplot(netData$NNF)
})
```

Київський національний університет імені Тараса Шевченка
Економічний факультет
Кафедра економічної кібернетики

ЗАВДАННЯ
на кваліфікаційну роботу магістра

студента 2 курсу спеціальності 051 «Економіка»,
ОНП «Економічна кібернетика»
Атояна Андрія Гарійовича

1. **Тема роботи**: «Моделювання фінансових інструментів біржі методами нейронних мереж»
2. **Термін завершення роботи**: 10.05.2022
3. **Попередній захист роботи**: 12.05.2022
4. **Об'єкт дослідження**: біржові індекси акцій компаній
5. **Предмет дослідження:** алгоритми глибинного навчання, їх архітектура та фактори впливу на ефективність моделювання фінансових часових рядів. Порівняння результатів аналізу з класичними методами машинного навчання.
6. **Мета дослідження**: дослідження фундаментальних принципів функціонування рекурентних нейронних мереж та оцінка їх ефективності у моделюванні фінансових часових рядів, що дасть змогу синтезувати рекомендації щодо застосування та ефективного використання даного класу моделей, виявити основні переваги та недоліки.
7. **Завдання дослідження**:
   - **7.1.** ознайомлення з науковою літературою та навчальними матеріалами провідних установ, стосовно особливостей рекурентних нейронних мереж, аспектів їх побудови та факторів впливу на результати моделювання.
   - **7.2.** дослідження принципів навчання нейронних мереж та проектування цього підходу на прикладі логістичної регресії з метою формуванню математичного підґрунтя та розкриття ключових факторів цього процесу.
   - **7.3.** формування глибинного розуміння архітектур рекурентних нейронних мереж, їх ключових компонентів, принципів обрахунку, виявлення основних переваг та недоліків відносно один одного.

7.4. оцінка та визначення ефективної методології прогнозування часових рядів, порівняння результатів нейромережевого моделювання з класичними методами аналізу фінансових часових рядів.

Науковий керівник: к.е.н., доцент Подскребко Олександр Сергійович

Студент: Атоян Андрій Гарійович

Затверджено на засіданні кафедри економічної кібернетики

протокол № 3 від 12 жовтня 2021 р

Календарний план виконання кваліфікаційної роботи магістра

№	Етапи роботи	Терміни виконання	Відмітка керівництва про виконання
1	Вибір теми кваліфікаційної роботи магістра	01.08.2021 – 01.09.2021	
2	Розробка та затвердження завдання кваліфікаційної роботи магістра	01.09.2021 – 20.09.2021	
3	Ознайомлення з науковою літературою та навчальними матеріалами провідних установ, з урахуванням обраної тематики. Освоєння інструментів реалізації аналізу.	20.09.2021 – 31.12.2021	
4	Написання розділу 1	01.01.2022 – 01.02.2022	
5	Написання розділу 2	01.02.2022 – 01.03.2022	
6	Написання розділу 3	01.03.2022 – 20.04.2022	
7	Написання вступу та висновків, оформлення документу.	20.04.2022 – 10.05.2022	
8	Подання роботи до попереднього захисту	12.05.2022	
9	Захист магістерської роботи	24.05.2022	

Науковий керівник: к.е.н., доцент Подскребко Олександр Сергійович

Студент: Атоян Андрій Гарійович