МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДОНЕЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ВАСИЛЯ СТУСА

ІВАНЕНКО АНДРІЙ ВАДИМОВИЧ

Допускається до захисту:
в. о. завідувача кафедри
інформаційних технологій,
канд. техн. наук, доцент
_____О.В. Зелінська
«____»_____ 2024 р.

ВИКОРИСТАННЯ МЕТОДІВ МАШИННОГО НАВЧАННЯ ДЛЯ ПРОГНОЗУВАННЯ СПОЖИВЧОГО ПОПИТУ В ЕЛЕКТРОННОЇ КОМЕРЦІЇ

Спеціальність 122 «Комп’ютерні науки»

Кваліфікаційна (магістерська) робота

Науковий керівник:
Т. В. Січко, доцент кафедри
інформаційних технологій,
канд. техн. наук, доцент

Оцінка: ___/___/_____
(бали за шкалою ЄКТС/за національною шкалою)

Голова ЕК: _____
(підпис)

Вінниця 2024

АНОТАЦІЯ

Іваненко А. В. Використання методів машинного навчання для прогнозування споживчого попиту в електронної комерції. Спеціальність 122 «Комп'ютерні науки», освітня програма «Комп'ютерні технології обробки даних (Data science)», Донецький національний університет імені Василя Стуса, Вінниця, 2024.

У кваліфікаційній (магістерській) роботі досліджені методи машинного навчання для вирішення задачі прогнозування споживчого попиту. Результатом роботи є розробка гібридної моделі нейронної мережі для прогнозування попиту, яка б допомагала у плануванні логістичної діяльності підприємств у сфері електронної комерції.

Ключові слова: машинне навчання, штучна нейронна мережа, глибоке навчання, прогнозування часових рядів, прогнозування споживчого попиту, електронна комерція.

101 с., 19 табл., 30 рис., 54 джерела.

ABSTRACT

Ivanenko A. V. Analysis and Forecasting of Consumer Demand in E-Commerce Based on Machine Learning Methods. Specialty 122 "Computer Science", educational program "Data Science", Vasyl' Stus Donetsk National University, Vinnytsia, 2024.

In the qualification (master's) thesis, machine learning methods were investigated to solve the problem of consumer demand forecasting. The result of the work is the development of a hybrid neural network model for demand forecasting, which would help in planning the logistics activities of e-commerce enterprises.

Keywords: machine learning, artificial neural network, deep learning, time series forecasting, consumer demand forecasting, e-commerce.

101 p., 19 tables, 30 figures, 54 sources.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

APE	absolute percentage error, абсолютна відсоткова помилка
AR	autoregressive, авторегресія
ARIMA	autoregressive integrated moving-average, авторегресійне інтегроване ковзне середнє
ARMA	autoregressive moving-average, авторегресійне ковзне середнє
CNN	convolutional neural network, згорткова нейронна мережа
ES	exponential smoothing, експоненційне згладжування
GRU	gated recurrent unit, вентильний рекурентний вузол
LSTM	long short-term memory, довга короткочасна пам’ять
MA	moving-average, ковзне середнє
MAE	mean absolute error, середня абсолютна похибка
MAPE	mean absolute percentage error, середня абсолютна відсоткова помилка
MLP	multilayer perceptron, багатошаровий перцептрон
RMSE	root mean square error, середньоквадратична помилка
RNN	recurrent neural network, рекурентна нейронна мережа
xLSTM	extended long short-term memory, розширена довга короткочасна пам’ять

ВСТУП

Актуальність теми дослідження. Прогнозування попиту є важливим завданням для управління ланцюгом постачання, особливо в електронній комерції, де попит клієнтів є менш передбачуваним і швидко змінюваним. Традиційні методи аналізу часто виявляються недостатніми, тоді як машинне навчання дозволяє враховувати складні взаємозв'язки між різними факторами та створювати точні моделі прогнозування. Ці методи працюють з різноманітними типами даних і автоматично адаптуються до нових споживчих трендів, що робить їх незамінними в динамічному онлайн-середовищі. Завдяки цьому компанії можуть краще задовольняти потреби клієнтів і залишатися конкурентоспроможними.

Об'єктом дослідження є процес розробки нових більш точних способів прогнозування попиту в сфері електронної комерції задля оптимізації логістичної діяльності.

Предметом дослідження є методи машинного навчання, які спрямовані на вирішення задачі прогнозування часових рядів, а саме прогнозування споживчого попиту.

Метою дослідження є розробка спеціалізованого алгоритму для прогнозування споживчого попиту, який би використовував інструменти машинного навчання.

Задачами дослідження є:

– розглянути теоретичні основи стосовно проблематики прогнозування споживчого попиту;

– дослідити еволюцію розвитку методів машинного навчання, використовуваних для прогнозування часових рядів;

– спроектувати спеціалізований алгоритм машинного навчання для прогнозування часових рядів;

– обрати інструменти для реалізації алгоритму прогнозування часових рядів;

– реалізувати розроблений алгоритм прогнозування;

– порівняти точність прогнозу реалізованого алгоритму з іншими алгоритмами машинного навчання.

Методами дослідження є: методи машинного навчання; статистичний аналіз, за яким оцінюється точність результатів розробленої моделі за допомогою числових метрик; метод візуалізації даних, застосовуваний для графічного представлення результатів роботи моделей прогнозування; метод порівняння значень метрик розробленої моделі зі значеннями результатів від інших методів прогнозування для кращого розуміння ефективності моделі.

Наукова новизна дослідження полягає в розробці нової гібридної моделі нейромережі ES-xLSTM – подальшої модифікації архітектури нейронних мереж xLSTM шляхом інтеграції в неї статистичного алгоритму машинного навчання – експоненційного згладжування, застосованої для більш ефективного рішення задачі прогнозування споживчого попиту.

Структура роботи. Магістерська робота складається зі вступу, трьох розділів, висновків та списку використаних джерел.

У першому розділі розглянуті теоретичні засади стосовно задачі прогнозування споживчого попиту в сфері електронної комерції.

В другому розділі досліджується розвиток методів прогнозування часових рядів та пропонується свій алгоритм прогнозу.

Третій розділ присвячений розробці моделі запропонованого алгоритму для прогнозування часових рядів, також реалізації інших алгоритмів, з ефективністю яких порівнюється точність прогнозу спроектованої моделі.

Практичне значення роботи полягає в застосуванні розробленого алгоритму для вирішення з більшою точністю задачі прогнозування споживчого попиту або інших видів часових рядів.

Результати дослідження апробовано на III Міжнародній науково-практичній конференції «Прикладні аспекти сучасних міждисциплінарних досліджень» (м. Вінниця, 1 листопада 2024 року).

РОЗДІЛ 1
ПРОБЛЕМАТИКА ПРОГНОЗУВАННЯ ПОПИТУ

1.1 Цілі прогнозування попиту

Прогнозування попиту — це процес оцінки майбутнього попиту на товар або послугу, який дозволяє підприємствам визначити приблизну кількість продукту або послуги, яку споживачі можуть бажати придбати у певний період часу [10]. Точність прогнозування є ключовою для підприємств, оскільки вона дозволяє ефективно планувати виробництво та задовольняти потреби клієнтів. Невірна оцінка попиту може призвести до надлишкових запасів або дефіциту продукції.

Прогнозування попиту та продажів є однією з ключових функцій для виробників, дистриб'юторів і торгових фірм, особливо у сфері електронної комерції, де споживчий попит дуже динамічний і менш передбачуваний. Підтримуючи баланс між попитом і пропозицією, підприємства можуть зменшити надлишок і дефіцит запасів, що підвищує їх прибутковість. Якщо виробник прагне задовольнити завищений попит, надлишок виробництва призводить до накопичення зайвих запасів. З іншого боку, недооцінка попиту призводить до невиконаних замовлень, втрати продажів і зниження рівня обслуговування. Обидві ситуації спричиняють неефективність у ланцюгу поставок. Тому точне прогнозування попиту є важливим викликом для учасників ланцюга поставок.

З точки зору реальних бізнес-проблем, прогнозування попиту зазвичай використовується в роздрібній торгівлі та електронній комерції для прогнозування рівня продажів і управління запасами. Наприклад, продуктовий магазин може використовувати прогнозування попиту, щоб передбачити, скільки кожного продукту замовляти у своїх постачальників, щоб переконатися, що вони мають достатньо запасів для задоволення попиту клієнтів. Іншим прикладом є роздрібний онлайн-продавець, який може використовувати прогнозування попиту, щоб передбачити, скільки продуктів

зберігати на своєму складі, щоб переконатися, що у нього достатньо запасів для задоволення попиту клієнтів.

Переваги від прогнозування попиту можна підсумувати таким чином [10]:

– прогнозування попиту приводить до кращого управління запасами. Оскільки чим точнішими будуть прогнози попиту, тим краще підприємство зможете керувати рівнем запасів. Це означає більшу ефективність, менше відходів, нижчі витрати та вищі прибутки;

– виконується покращене планування виробництва. Добре планування попиту може значно збільшити здатність підприємства узгоджувати виробництво з попитом. Це дасть більше можливостей керувати часом і ресурсами, а також мінімізувати відходи;

– прогноз попиту сприяє спільному плануванню. Так як, надаючи уніфіковану картину майбутнього попиту, компанія зможе переконатися, що її відділи продажів, маркетингу, фінансів і операцій працюють з однаковими числами;

– покращене управління ланцюгом поставок. Без чіткого уявлення про те, куди рухається бізнес, важко приймати правильні рішення щодо управління ланцюгом поставок щодо того, які матеріали йому потрібні, з якими постачальниками працювати та навіть куди слід інвестувати свій час. Хороший прогноз попиту надає ясність і напрямок, необхідний для вжиття найкращих дій у ланцюзі поставок;

– зміцнення матеріального становища. Чим кращі прогнози попиту підприємства, тим краще його здатність прогнозувати грошові потоки, доходи та загальний фінансовий стан.

Досконале прогнозування попиту здійснюється у наступні чотири кроки.

Перший крок – це підбір правильної сфери застосування. В залежності від умов можуть застосовуватись різні типи прогнозування, воно може здійснюватися у різних вимірах та часових горизонтах. Для ефективного прогнозування важливо розуміти, які типи прогнозів потрібні у конкретній

ситуації, що залежить від типу товарів чи послуг, попит на які потрібно спрогнозувати, та умов на ринку.

Другий крок – це використання для прогнозу даних, які коректно відображають ситуацію. Дані є основою будь-якого прогнозу. Проте важливо не просто зібрати інформацію, а забезпечити її якість та релевантність. Самий простий шлях для прогнозування попиту - це використання даних за продажі. Проте продажі відображають лише той попит, який вдалося задовольнити, але існує й прихований попит, обумовлений відсутністю товарів, обмеженою доступністю чи іншими бар'єрами. Визначення цього потенційного попиту дозволяє отримати реальнішу картину. Також попит залежить не лише від властивостей продукту, але й від: акцій чи змін у ціноутворенні; дій конкурентів, таких як запуск нових продуктів; сезонних коливань або макроекономічних змін. Крім того важливо забезпечити якість даних: відсутність пропусків, дублювань, та провести очищення даних від шуму, щоб модель працювала ефективно.

Третій крок – це вибір відповідних метрик для оцінки точності прогнозу.

Четвертий крок – це підбір точної моделі для створення прогнозу. Важливо вибрати модель, що відповідає вашим умовам. Краще всього використовувати якомога потужнішу модель для виконання більш точних прогнозів.

Далі будуть розглянуті теоретичні засади щодо зазначених кроків.

1.2 Види прогнозування споживчого попиту

Існують різні галузі прогнозування попиту, кожна з яких пропонує унікальну інформацію та підходить для різних потреб бізнесу та ринкових умов. Використання кількох типів прогнозів дасть повніше уявлення про майбутні продажі компанії, а також може підкреслити відмінності в прогнозах. Ці відмінності можуть вказувати на необхідність додаткових досліджень або кращого введення даних.

Виділяють такі типи прогнозування попиту [28]:

– пасивне прогнозування попиту – ідеально підходить для підприємств зі стабільними, передбачуваними ринками. Воно покладається на історичні дані та тенденції для прогнозування майбутнього попиту, припускаючи, що минулі закономірності збережуться. Цей спосіб менш підходить для швидкозмінних або висококонкурентних ринків, де інновації та зриви є звичайним явищем;

– активне прогнозування попиту – використовує більш динамічний підхід, що включає поточні тенденції ринку, зміни в поведінці споживачів і майбутні зміни в галузі. Воно призначене для підприємств у швидкоплинних середовищах, де гнучкість і оперативність ланцюжка поставок є вирішальними для того, щоб залишатися попереду;

– короткострокове прогнозування попиту – зосереджуючись на найближчому майбутньому, як правило, на рік вперед, короткострокове прогнозування попиту допомагає компаніям ефективно керувати повсякденними операціями та сезонними коливаннями. Це надзвичайно важливо для управління запасами та досягнення короткострокових фінансових цілей;

– довгострокове прогнозування попиту – заглядаючи на кілька років у майбутнє, довгострокове прогнозування попиту використовується для стратегічного планування, наприклад, для розширення потужностей, виходу на нові ринки або запуску нових продуктів. Хоча за своєю суттю воно більш невизначене, воно життєво важливе для спрямування довгострокового розвитку бізнесу та інвестиційних рішень;

– прогнозування зовнішнього попиту – цей підхід виходить за межі внутрішніх даних компанії, враховуючи макроекономічні показники, галузеві тенденції та конкурентний ландшафт. Прогнозування зовнішнього попиту має важливе значення для розуміння ширших ринкових сил і потенційних вузьких місць у ланцюжку поставок, щоб позиціонувати бізнес для сталого зростання;

– прогнозування внутрішнього попиту – зосереджуючись на власних даних про продажі та продуктивність компанії, даний підхід надає інформацію на основі історичних тенденцій продажів, виробничих потужностей і

внутрішніх ресурсів. Це особливо корисно для оптимізації операційної ефективності та розподілу ресурсів.

Також прогноз попиту визначається за трьома вимірами (або ієрархіями): за продуктами, за географією та за періодами часу. [49]

При прогнозуванні за продуктами можна прогнозувати попит, використовуючи різні матеріальні рівні: за SKU (ідентифікатором товарної позиції), продуктом, сегментом, брендом тощо. А також з різними показниками вимірювання: одиницями, значенням, вагою, типом необхідної сировини, тощо.

При прогнозуванні за географією можна прогнозувати попит за країною, регіоном, ринком, каналом, сегментом клієнтів, складом, магазином, поштовим індексом, тощо.

При прогнозуванні за періодами часу можна використовувати різні часові відрізки (щодня, щотижня, щомісяця, щоквартально чи навіть щорічно).

Крім того при прогнозі треба мати на увазі різні моделі поведінки попиту, наприклад, такі як: горизонтальна модель, модель трендів, сезонна модель, акційна модель [45].

Горизонтальна модель – припускає, що попит залишається відносно постійним з часом. Немає значних тенденцій або сезонних закономірностей, і попит коливається навколо постійного середнього значення. Ця модель часто використовується для продуктів або послуг зі стабільним попитом, де зовнішні фактори не викликають значних коливань. Приклади включають основні побутові товари або продукти харчування.

Модель трендів – фіксує систематичні зміни попиту з часом у бік зростання або зниження. Ця модель визначає довгострокове зростання попиту, яке може бути спричинене такими факторами, як економічне зростання, технологічний прогрес або зміни в споживчих уподобаннях. Ця модель підходить для продуктів або послуг, які переживають зростання або занепад, наприклад технологічних продуктів або галузей на фазах розширення або скорочення.

Сезонна модель – враховує регулярні коливання попиту, які відбуваються в певний час року, наприклад щомісяця, кварталу чи року. Ці моделі повторюються протягом фіксованого періоду часу. Ця модель застосовна для продуктів або послуг із прогнозованими сезонними коливаннями, наприклад сезонний одяг, товари пов'язані зі святами, тощо.

Акційна модель – досліджує вплив рекламних заходів на попит. Ця модель включає такі фактори, як знижки, рекламні кампанії та спеціальні пропозиції, які можуть тимчасово підвищити попит. Роздрібні продавці та компанії, які проводять часті рекламні акції, щоб стимулювати продажі, наприклад під час святкових сезонів, розпродажів або запуску продуктів. Акція зазвичай має дату початку та дату завершення, і попит у ці дні є відносно вищим, ніж у звичайні дні, коли не проводяться акції. Ця ситуація викликає різкі коливання в історії попиту та порушує модель прогнозування, що використовується. Стандартне відхилення збільшується, і коефіцієнти прогнозу виходять з-під нормального контролю. Тому для подолання коливань у прогнозних моделях необхідні спеціальні коригування.

На практиці поведінка попиту часто демонструє комбінацію факторів моделей перелічених вище. А тому досить часто використовуються гібридні моделі, які комбінують елементи цих моделей. Кожна модель забезпечує структуру для охоплення різних аспектів попиту, і їх поєднання може вирішити проблеми, пов'язані з хаотичною природою попиту.

Розглянуті вище різновиди прогнозування попиту є важливими при продумуванні логістичної діяльності підприємств та плануванні здійснення прогнозування попиту. Проте у нашому дослідженні для демонстрації методів прогнозування попиту зосередимося саме на типі довгострокового прогнозування певного товару, використовуючи дані з одного датасету.

Методи прогнозування можна поділити на три групи: якісні методи, причинно-наслідкові методи та методи часових рядів [23].

Якісні методи прогнозування суб’єктивні, ґрунтуються на думці і судженні споживачів та експертів; вони підходять лише тоді, коли попередні

дані недоступні. Прикладами якісних методів прогнозування є, наприклад, метод експертних оцінювань, метод Дельфі, дослідження ринку, метод опитувань та метод сценаріїв.

Причинно-наслідкові методи базуються на припущеннях, що існують причинно-наслідкові зв’язки між попитом і різними факторами, такими як економічні показники, поведінка споживачів і ринкові тенденції; прогнозування попиту за допомогою цих методів ґрунтується на визначенні кореляцій між цими факторами та попитом. Вони корисні, коли є чітке розуміння факторів попиту, таких як реклама, рекламні акції чи ціноутворення. Однак створення причинно-наслідкових моделей може бути складним і вимагає значних даних і досвіду. Прикладами причинно-наслідкових методів є лінійна регресія та економетричні моделі.

Лінійна регресія використовується для оцінки впливу однієї або декількох незалежних змінних на залежну змінну, що може бути споживчим попитом. Вона може бути одноразовою (одна незалежна змінна) або багаторазовою (декілька незалежних змінних). Економетричні моделі, у свою чергу, глибше аналізують економічні зв'язки, використовуючи теоретичні концепції та статистичні методи для прогнозування споживчого попиту. Вони можуть враховувати різні фактори, такі як ціни, доходи, рекламні витрати тощо, для побудови складних моделей, які краще відображають реальність. Такі моделі можуть бути статичними або динамічними, в залежності від того, які часові рамки вони охоплюють та які фактори враховуються.

Методи часових рядів аналізують історичні дані про попит для виявлення закономірностей і тенденцій. Припускається, що майбутній попит буде слідувати аналогічним шаблонам, які спостерігаються в минулому. Він ефективний для продуктів з відносно стабільним попитом. Однак він потребує додаткової підтримки, щоб зафіксувати раптові зміни попиту, спричинені непередбачуваними подіями або ринковими збоями. Аналіз часових рядів включає такі методи як: наївний метод прогнозу, ковзне середнє,

експоненційне згладжування, аналіз кривої тренду, авторегресійне лінійне прогнозування, авторегресійне інтегроване ковзне середнє та багато інших.

Серед представлених способів прогнозування попиту надамо перевагу методам часових рядів. Дані методи є більш ефективними для прогнозування попиту, оскільки споживчий попит може бути досить спонтанним, що робить складним його моделювання іншими способами, а тому його краще прогнозувати на попередніх вже існуючих даних. Крім того, так як методи часових рядів працюють на основі історичних даних, це робить їх зручними для сфери електронної комерції, в якій можна легко збирати дані користувачів.

1.3 Особливості обробки даних в електронній комерції для прогнозу попиту

При цьому потрібно приймати до уваги викривлення інформації про попит, які виникають у ланцюзі поставок, таких як ефект «бичачого батога» (Bullwhip effect) [52]. Ефект «бичачого батога» – це явище в ланцюгу поставок, коли замовлення постачальникам мають тенденцію до більшої мінливості, ніж продажі покупцям, що призводить до посилення мінливості попиту на вищих рівнях ланцюга. Частково це призводить до збільшення коливань запасів у відповідь на зміну споживчого попиту в міру просування вгору по ланцюгу поставок.

Ефект «бичачого батога» є основним джерелом труднощів, пов’язаних з управлінням ланцюгом поставок, особливо з операційним управлінням. Даний ефект призводить до появи багатьох проблем, серед яких вирізняють: проблеми у сфері планування виробництва, труднощі в управлінні людьми залежно від флуктуацій попиту, проблеми з управлінням запасами, проблеми з управлінням складами та транспортом в ситуації частих і значущих змін рівня запасів, видовження часу реалізації замовлень, тощо.

Використання деяких методів прогнозування, наприклад, ковзної середньої, наївного прогнозування або обробки сигналів попиту призводить до появи ефекту «бичачого батога», натомість авторегресійні моделі

прогнозування показали, що вони здатні зменшувати даний ефект, а тому потрібно приділяти більшу увагу при виборі правильних методик прогнозу.

Крім того при зборі даних для аналізу попиту важливо підкреслити, що ланцюгам постачання необхідно прогнозувати попит, а не самі продажі. Тобто попит не вимірюється просто кількістю фактичних продажів, які обмежені наявними запасами. Звичайно, доки запаси достатні, всі запити призводять до продажів. Тому відстеження продажів, а не прямого попиту, вважається нормою, поки не виникає дефіцит. Проте проблеми з аналізу попиту виникають, коли запасів стає не вистачати. Таким чином виявлення реального обсягу попиту може бути складною задачею.

Для вирішення даної проблеми в електронній комерції є можливість використання системи управління замовленнями для обробки вхідних замовлень та запитів клієнтів. Таким чином можна збирати дані про попит навіть у випадку дефіциту товарів [49].

Для цього потрібно відстежувати різні типи замовлень, такі як:

– відкриті замовлення – це замовлення, які ще не доставлені, найімовірніше тому, що наразі немає необхідних товарів. Потрібно зберігати резервні копії цих відкритих замовлень і інформувати своїх клієнтів про них;

– дубльовані замовлення – якщо не зберігати резерв відкритих замовлень, клієнти можуть повторно замовляти те саме замовлення кілька разів, доки воно не буде виконано. Коли записувати кожне подібне вхідне замовлення як нове, збирається надмірно завищений попит;

– скасовані замовлення – деякі клієнти скасовують свої замовлення, тому що компанії не можуть їх обслужити вчасно. Є необхідність відстежувати ці замовлення, особливо їхні початково запитані дати доставки, і причину їх скасування;

– замовлення на заміну – деякі клієнти вирішать придбати інший продукт замість свого початкового вибору, якщо його немає в наявності. Потрібно буде відстежувати ці замовлення на заміну окремо, оскільки попит слід віднести до початкового продукту, а не до проданого продукту.

Нам також потрібно враховувати нездійснені замовлення, коли клієнти не роблять покупку через відсутність товару. Для збору даних з них можна використовувати онлайн-кошики та списки бажаного, в які споживачі можуть записувати товари, які вони бажають замовити, але не можуть через відсутність продуктів. Хоча відстеження цього попиту є складним, тим не менш, такі замовлення також повинні бути включені до загального необмеженого попиту.

Висновки до розділу 1

У розділі розглядається задача прогнозування споживчого попиту, у контексті сфери електронної комерції.

Прогнозування попиту – це процес оцінювання попиту споживачів на товар або послугу протягом певного майбутнього періоду часу.

При цьому діяльність з прогнозування попиту стає є складнішою, коли мова йде про галузь електронної комерції, оскільки вона дуже динамічна, а попит клієнтів є менш передбачуваним і може сильно коливатися щодня.

Виділяють такі типи прогнозування попиту як: пасивне, активне, короткострокове, довгострокове, зовнішнє, внутрішнє.

Також прогноз попиту визначається за трьома вимірами: за продуктами, за географією та за періодами часу.

Методи прогнозування попиту поділяють на три групи: якісні методи, які ґрунтуються на судженні споживачів та експертів; причинно-наслідкові методи, які призначені для побудови математичних моделей, що відображають причинно-наслідкові зв’язки між попитом і різними факторами; методи часових рядів, які аналізують історичні дані про попит для виявлення закономірностей і тенденцій.

У розділі також досліджено які проблеми можуть виникати при зборі даних для з'ясування справжнього попиту на товари або послуги.

РОЗДІЛ 2
ЗАГАЛЬНІ ВІДОМОСТІ ПРО МЕТОДИ МАШИННОГО НАВЧАННЯ

2.1 Статистичні методи прогнозування часових рядів

Прогнозування є невід'ємною частиною управління ланцюгом поставок. Але традиційні методи прогнозування мають серйозні обмеження, які впливають на точність прогнозування. Алгоритми машинного навчання виявилися корисними методами для прогнозування попиту завдяки їхній здатності враховувати нелінійні дані, фіксувати тонкі функціональні зв'язки між емпіричними даними, навіть якщо базові зв'язки невідомі або їх важко описати.

Машинне навчання – це клас методів штучного інтелекту, характерною рисою яких є не пряме розв'язання завдання, а навчання за рахунок застосування рішень безлічі подібних завдань [54]. Машинне навчання є підрозділом штучного інтелекту, що скерований на самонавчання комп'ютерів, щоб частково або навіть повністю автоматизувати рішення різних складних аналітичних задач. Замість написання чітких інструкцій, алгоритми отримують великі обсяги даних і самостійно формують висновки на їх основі. Такий підхід дозволяє знаходити приховані закономірності в даних, створювати моделі та отримувати результати без необхідності задавати наперед прописані правила.

Основна мета машинного навчання – забезпечити максимально точні прогнози на основі вхідної інформації, допомагаючи користувачам приймати обґрунтовані рішення у своїй сфері. Завдяки цьому алгоритми можуть прогнозувати результати, запам'ятовувати їх, відтворювати за потреби і вибирати найкращі варіанти з кількох можливих.

Для машинного навчання, потрібні три складові: дані, ознаки та алгоритм.

Дані є основою будь-якої системи машинного навчання. Без достатньої кількості якісних даних неможливо створити модель, яка б давала коректні

прогнози або класифікації. Для різних задач можуть використовуватись дані в різних форматах, таких як текст, зображення, відео, аудіо, табличні дані тощо.

Ознаки є характеристиками або властивостями, які використовуються для побудови моделі машинного навчання. Вибір та інженерія ознак є одним з найважливіших етапів у процесі машинного навчання, оскільки від якості ознак залежить здатність моделі навчатися і робити точні прогнози. При алгоритмах машинного навчання, які навчаються з вчителем, ознаки задаються в самому наборі даних, який повинен використовувати алгоритм, а при навчанні без вчителя алгоритм сам повинен з усієї множини даних виділити значущі ознаки, за якими можна бачити закономірності в даних.

Алгоритм машинного навчання — це математична або статистична модель, яка навчається на даних для того, щоб робити прогнози або класифікації на нових даних. Існує багато різних алгоритмів машинного навчання, таких як лінійна регресія, логістична регресія, дерева рішень, випадковий ліс, метод опорних векторів та багато інших. Вибір алгоритму залежить від конкретної задачі, типу даних і вимог до моделі. Наприклад, для задачі прогнозування попиту можуть використовуватись алгоритми, які були зазначені раніше, такі як алгоритми з групи причинно-наслідкових методів та методів часових рядів.

Розрізняють такі види машинного навчання як: навчання з вчителем, навчання без вчителя та навчання з підкріпленням.

– Навчання з вчителем передбачає наявність навчальної множини, що містить правильні приклади (пари «вхід» – «вихід»). Алгоритм навчається на деякій множині таких навчальних пар: пред’являються входи, обчислюється вихід алгоритму і порівнюється з відповідним правильним виходом. Для мінімізації похибки ваги змінюються відповідно до методики навчання. Приклади навчальної множини пред’являються послідовно, обчислюються похибки і ваги підлаштовуються для всіх прикладів використаної навчальної множини. Такий період називається епохою. Алгоритм ітеративно повторює

кроки навчання, доки похибка по всій навчальній множині не досягне прийнятного рівня.

– Навчання без вчителя використовує навчальну множину, що складається лише з вхідних векторів. Навчальний алгоритм підлаштовує ваги мережі так, щоб пред'явлення достатньо близьких вхідних векторів надавало однакові виходи. Процес навчання виділяє статистичні властивості навчальної множини і групує подібні вхідні вектори у класи чи кластери.

– Навчання з підкріпленням – один із способів машинного навчання, в ході якого система навчається, взаємодіючи з певним середовищем. Відгуком середовища на прийняті рішення є сигнали підкріплення, тому таке навчання є окремим випадком навчання з вчителем, але вчителем є середовище або його модель. Деякі правила підкріплення базуються на неявних вчителів, через що їх можна віднести до навчання без вчителя.

Далі розглянемо статистичні алгоритмів машинного навчання для прогнозування часових рядів. Розглянемо еволюцію розвитку методів для прогнозування часових рядів.

— Наївний прогноз (Naïve Forecasting) – один із найпростіших способів прогнозування часових рядів, за яким для значень прогнозу наступного періоду використовуються фактичні дані за попередній період, без їх коригувань чи спроби встановити причинні фактори. Він у більшості своєму використовується для порівняння з прогнозами кращими складнішими алгоритмами.

Формула наївного прогнозу має такий вигляд [13]:

$$f_t = d_{t-1} \tag{2.1}$$

де f_t – прогноз за період t;

d_{t-1} – фактичні дані за попередній період.

— Сезонний наївний прогноз (Seasonal Naïve Forecast) – це вид наївного прогнозу, при якому часовий ряд поділяється на сезони (наприклад роки,

місяці, тижні) і передбачається, що значення прогнозу наступного періоду поточного сезону дорівнюватиме значенню в аналогічному періоді попереднього сезону. Наприклад, якщо зпрогнозувати місячний попит за наступний рік, то, використовуючи сезонний наївний прогноз, будемо вважати, що попит за кожний місяць відповідатиме попиту за аналогічний місяць попереднього року.

Формула сезонного наївного прогнозу має наступний вигляд [13]:

$$f_t = d_{t-s} \tag{2.2}$$

де s – це кількість періодів t, що визначають сезон;

d_{t-s} – фактичні дані за аналогічний період в попередньому сезоні.

— Ковзне середнє (Moving-Average) – це алгоритм, за яким значення прогнозу наступного періоду визначається як середнє арифметичне фактичних значень n-кількості останніх періодів. n-кількість останніх періодів називають «вікном», за яким відбувається прогноз. Основна ідея алгоритму полягає в тому, що, розглядаючи не один попередній період, а середнє значення декількох останніх періодів, ми зменшуємо вплив випадкових коливань або «шуму» на прогноз, таким чином згладжуємо короткострокові коливання і виділяємо загальну тенденцію.

Формула ковзного середнього виглядає так [13]:

$$f_t = \frac{1}{n}\sum_{i=1}^{n} d_{t-i} \tag{2.3}$$

де n – кількість періодів, для яких береться середнє значення;

d_t – попит протягом періоду t.

— Зважене ковзне середнє (Weighted Moving-Average) – це вид прогнозування ковзного середнього, в якому значення прогнозу наступного періоду визначається як середнє зважене арифметичне фактичних значень n-кількості останніх періодів. Тобто, кожному періоду з минулих n-значень

призначається своя вага w, на яку вони множаться, причому новіші значення, як правило, мають більшу вагу. Результати множення підсумовуються і діляться на суму всіх ваг.

Одним із недоліків звичайного ковзного середнього є присвоєння при його розрахунку всім цінам однакових ваг при усередненні незалежно від того, ближче чи далі вони від поточного моменту. Цей недолік усунуто у зваженому ковзному середньому. Таким чином, алгоритм дозволяє враховувати значущість більш свіжих даних для точнішого прогнозування.

Формула зваженого ковзного середнього має такий вигляд [13]:

$$f_t = \frac{\sum_{i=1}^{n} w_i d_{t-i}}{\sum_{i=1}^{n} w_i} \tag{2.4}$$

де w_i – вага, яку призначають для і-того періоду.

— Експоненційне згладжування або експоненційне ковзне середнє (Exponential Smoothing, Exponential Moving-Average) — це алгоритм прогнозування часових рядів, різновид зваженого ковзного середнього, який базується на використанні зваженого середнього попередніх значень ряду, де ваги експоненційно зменшуються в часі. Цей підхід дозволяє ще більше уваги приділяти останнім спостереженням, зменшуючи вплив старіших значень, що робить його особливо ефективним для даних з поступовими змінами або деякою ступенем волатильності.

Основна ідея алгоритму полягає в тому, що новий прогноз є лінійною комбінацією попереднього прогнозу і фактичного спостереження. Кожен новий прогноз обчислюється як сума двох компонентів: першого — це попередній прогноз, помножений на параметр згладжування α, і другого — фактичного значення попереднього періоду, також помноженого на (1 - α). Параметр α (від 0 до 1) визначає ступінь впливу нових даних: чим більше значення α, тим більше значення надається новим даним, і тим швидше прогноз адаптується до змін у ряді.

Експоненційне згладжування можна сформулювати таким чином [13]:

$$f_t = f_{t-1} + \alpha(d_{t-1} - f_{t-1}) \quad (2.5)$$

або

$$f_t = \alpha d_{t-1} + (1-\alpha) f_{t-1} \quad (2.6)$$

де α – параметр згладжування.

— Авторегресійне лінійне прогнозування (Autoregressive Linear Forecast) – є статистичним підходом, який використовується для прогнозування майбутніх значень часових рядів на основі їхніх власних попередніх значень [38]. Ця модель передбачає, що значення в даний момент часу залежить від певної кількості попередніх значень та випадкової похибки.

У моделях авторегресії застосовується лінійна регресія на основі вихідних значень змінних, отриманих на попередніх кроках. На відміну від звичайної лінійної регресії, модель авторегресії не використовує інші незалежні змінні, крім передбачуваних раніше результатів.

Формула авторегресійної моделі має такий вигляд [13]:

$$f_t = \sum_{i=1}^{n} \phi_i f_{t-i} + \varepsilon_t \quad (2.7)$$

де ϕ_i – коефіцієнти моделі, що визначають вплив попередніх значень на прогноз;

ε_t – випадкова похибка для періоду t.

— Авторегресійне ковзне середнє (Autoregressive Moving-Average, ARMA) – це техніка часових рядів, яка припускає, що прогноз за наступний період можна представити як лінійну функцію спостережень і залишкових помилок від фактичних даних попередніх періодів [38]. Ця модель прогнозування часових рядів комбінує дві складові: авторегресійну частину (AR) та складову ковзного середнього (MA).

Для кожного обчислюваного прогнозованого періоду модель ARMA використовує попередні фактичні дані періодів часового ряду та похибки для обчислення прогнозованого значення. AR частина прогнозує на основі попередніх значень ряду, а MA частина враховує минулі помилки для коригування прогнозу.

Дану модель можна визначити як функцію ARMA(p, q), де p – це кількість «запізнень» – попередніх значень, у авторегресійній частині, а q – кількість попередніх значень (запізнень) у частині ковзного середнього. Формула моделі ARMA є об'єднанням формул авторегресійної моделі та моделі ковзного середнього (статистична модель ковзного середнього відрізняється від звичайного алгоритму прогнозування ковзного середнього) [13]:

$$f_t = \varepsilon_t + \sum_{i=1}^{p} \phi_i f_{t-i} + \sum_{i=1}^{q} \theta_i \varepsilon_{t-i} \qquad (2.8)$$

де ε_t – випадкова похибка або шум в момент часу t;

ϕ_i – коефіцієнт авторегресійної частини (AR);

θ_i – коефіцієнт частини ковзного середнього (MA);

ε_{t-i} – минулі похибки прогнозу.

— Авторегресійне інтегроване ковзне середнє (Autoregressive Integrated Moving-Average, ARIMA) – статистична модель, яка є покращенням моделі ARMA. У порівнянні з ARMA, модель ARIMA поєднує в собі три компоненти: авторегресію (AR), інтеграцію (I) і ковзне середнє (MA) [38]. Як і у моделі ARMA, частина авторегресії AR відповідає за прогноз значення наступного періоду залежно від попередніх значень фактичних даних, а частина ковзного середнього MA – визначає, що прогнозоване значення також залежить від попередніх помилок моделі. Частина інтеграції I вказує на кількість різниць, які необхідні для того, щоб зробити ряд стаціонарним (виправити тренд або змінити дисперсію). Іншими словами, це етап, коли різниця між послідовними значеннями використовується для перетворення ряду прогнозованих даних.

Основні відмінності між моделями ARMA та ARIMA полягають у поняттях інтегрування та диференціювання. ARMA є стаціонарною моделлю, і вона дуже добре працює зі стаціонарними часовими рядами (статистичні властивості яких, такі як середнє, автокореляція та сезонність, не залежать від часу, в який спостерігався ряд). При цьому можна стаціонаризувати часовий ряд за допомогою різницевих методів (наприклад, шляхом віднімання значення, що спостерігається в момент часу t, від значення, що спостерігається в момент часу t-1). Процес оцінки того, скільки несезонних різниць потрібно для стаціонарності часового ряду, називається інтегруванням (I) або інтегрованим методом.

Дану модель можна визначити як функцію ARIMA(p, d, q), де p — порядок авторегресії (кількість попередніх значень ряду, які використовуються в прогнозі), d — ступінь різниці (кількість разів скільки повинно відбутись різниць фактичних даних часового ряду під час прогнозу), а q — порядок ковзного середнього (кількість попередніх похибок, що впливають на прогноз). Формула моделі ARIMA основана на формулі моделі ARMA [13]:

$$\Delta^d f_t = \varepsilon_t + \sum_{i=1}^{p} \phi_i \Delta^d f_{t-i} + \sum_{i=1}^{q} \theta_i \varepsilon_{t-i} \qquad (2.9)$$

де $\Delta^d f_t$ – позначає різницю значень прогнозів.

2.2 Теоретичні основи нейронних мереж, використовуваних для прогнозу часових рядів

Названі алгоритми є традиційними інструментами для прогнозування часових рядів. Проте, останнім часом дедалі більшої популярності набувають методи, засновані на нейронних мережах.

Глибоке навчання – це підрозділ машинного навчання, який зосереджується на використанні штучних нейронних мереж для вирішення складних завдань аналізу даних.

Штучна нейронна мережа – це обчислювальна модель, натхненна структурою та функціональними аспектами біологічних нейронних мереж. Нейромережа складається з взаємопов'язаної групи штучних нейронів, і вона обробляє інформацію, використовуючи зв'язковий підхід до обчислень [53].

Нейрон – це обчислювальна одиниця, яка отримує інформацію, здійснює над нею прості обчислення і передає її далі. Вони поділяються на три основні типи: вхідний, прихований та вихідний.

Штучна нейронна мережа є адаптивною, найчастіше нелінійною системою, яка вчиться виконувати функцію на основі даних. Адаптивний означає, що параметри системи змінюються під час роботи, яка зазвичай називається фазою навчання. Після фази навчання параметри мережі фіксуються, а система розгортається для вирішення поставленої проблеми – це фаза тестування. Штучна нейромережа будується за допомогою систематичної покрокової процедури для оптимізації критерію ефективності або дотримання деяких неявних внутрішніх обмежень, які зазвичай називають правилом навчання. Навчальні дані введення/виведення є основоположними в технології нейронних мереж, оскільки вони передають необхідну інформацію для «виявлення» оптимальної робочої точки. Нелінійний характер елементів обробки нейромереж надає системі велику гнучкість для досягнення практично будь-якої бажаної карти введення/виведення, тобто деякі штучні нейронні мережі є універсальними відображувачами.

Вхід подається до нейромережі, а на виході встановлюється відповідна бажана відповідь або ціль (у цьому випадку навчання називається контрольованим). Похибка визначається як різниця між бажаною відповіддю та виходом системи. Ця інформація про похибку повертається в систему та систематично коригує системні параметри. Процес повторюється, поки продуктивність не стане прийнятною. З цього опису зрозуміло, що продуктивність значною мірою залежить від даних.

Глибокому навчанню почали надавати перевагу з поміж інших галузей машинного навчання через низку різних переваг, які мають штучні нейронні

мережі. Перевагами нейронних мереж перед традиційними обчислювальними методами є такими:

– рішення задач в умовах невизначеності. Завдяки здатності до навчання нейронна мережа дозволяє вирішувати завдання з невідомими закономірностями і залежностями між вхідними та вихідними даними, що дозволяє працювати з неповними даними;

– стійкість до шумів у вхідних даних. Нейронна мережа може самостійно виявляти неінформативні для аналізу параметри і видаляти їх, в зв'язку з чим відпадає необхідність у попередньому аналізі вхідних даних;

– гнучкість структури нейронних мереж. Компоненти нейрокомп'ютерів – нейрони і зв'язки між ними – можна комбінувати в різний спосіб. За рахунок цього один нейрокомп'ютер можна застосовувати для вирішення різних завдань, часто не пов'язаних між собою;

– висока швидкодія. Вхідні дані обробляються багатьма нейронами одночасно, завдяки чому нейронні мережі вирішують завдання швидше, ніж більшість інших алгоритмів;

– адаптація до змін навколишнього середовища. Нейронні мережі, навчаючись на даних, здатні підлаштовуватися під змінне навколишнє середовище (наприклад, зміни ситуації на ринку). Якщо необхідно вирішувати якесь завдання в умовах нестаціонарного середовища, то можуть бути створені нейронні мережі, що перенавчаються в режимі реального часу. Чим вище адаптивні здібності системи, тим більш стійкою буде її робота в нестаціонарному середовищі;

– відмовостійкість нейронних мереж. На несприятливу зміну умов нейромережа реагує лише незначним зниженням продуктивності. Ця особливість пояснюється розподіленим характером зберігання інформації в нейронної мережі, тому істотно вплинути на працездатність нейромережі можуть лише серйозні пошкодження структури.

На сьогоднішній день глибоке навчання це один з напрямків, що швидко розвиваються, в машинному навчанні за останні роки. Це пов’язано з низкою факторів [15]:

– зростанням обчислювальної потужності, що дозволяє навчати складніші нейронні мережі;

– появою великих наборів даних, які необхідні навчання нейронних мереж;

– розвитком нових алгоритмів навчання, які дозволяють нейронним мережам навчатися швидше та точніше.

Найважливішими властивостями штучних нейронних мереж є наступні [9]:

– здатність до навчання. Завдяки використанню методу навчання мережа може витягти існуючий зв’язок між декількома змінними програми;

– здатність до узагальнення. Після завершення процесу навчання мережа може узагальнити отримані знання, дозволяючи оцінювати рішення, які поки що невідомі;

– організація даних. На основі вродженої інформації про певний процес мережа може організовувати цю інформацію, таким чином дозволяючи кластеризувати шаблони зі спільними характеристиками;

– розподілене зберігання. Знання про поведінку певного процесу, отримані нейронною мережею, зберігаються в кожному з кількох синапсів між штучними нейронами, тому покращення надійності архітектури у випадку втрати деяких нейронів;

– спрощене створення прототипів. Залежно від особливостей застосування більшість нейронних архітектур можна легко прототипувати на апаратному або програмному забезпеченні, оскільки її результати після процесу навчання зазвичай отримують за допомогою деяких фундаментальних математичних операцій.

При розгляді теми нейронних мереж важливим є розгляд терміну «активаційної функції». У штучних нейронних мережах функція активації

нейрона визначає вихідний сигнал, що визначається вхідним сигналом чи набором вхідних сигналів. Функція активації – це спосіб нормалізації вхідних даних. Тобто, якщо на вході у вас буде велика кількість, пропустивши його через функцію активації, ви отримаєте вихід у потрібному діапазоні. Функція активації є ключовим компонентом нейронної мережі, що дозволяє вивчати складні закономірності та приймати нелінійні рішення. Обрання відповідної активаційної функції є важливою частиною дизайну нейронних мереж і значно впливає на їх продуктивність.

Існує безліч різних видів функцій активації, але з них основними є: лінійна функція, сигмоїд (логістична функція), гіперболічний тангенс, зрізаний лінійний вузол (Rectified Linear Unit, ReLU) та нормована експоненційна функція (Softmax).

Лінійна функція – це найпростіша активаційна функція, де вихід є просто пропорційним входу. Вона легка у використанні та має нескінченний діапазон значень, але також вона не дозволяє моделювати нелінійності, що обмежує здатність мережі навчатися складним функціям [53].

$$f(x) = x \tag{2.10}$$

де x – це вхідні дані, які проходять через активаційну функцію f(x) і повинні бути оброблені нейронною мережею.

Сигмоїд – нелінійна функція, яка обмежує вхідні значення в діапазон (0, 1). Вона добре підходить для моделювання ймовірностей, але схильна до проблеми «зникаючого градієнта», де градієнти стають дуже малими для великих або малих вхідних значень, що ускладнює навчання глибоких мереж [53].

$$f(x) = \frac{1}{1+e^{-x}} \tag{2.11}$$

Гіперболічний тангенс – це нелінійна функція, яка обмежує вхідні значення в діапазон (-1, 1). Вона є масштабованою версією сигмоїдної функції. Дана функція симетрична відносно нуля, що допомагає центрувати дані. При цьому дана функція також схильна до проблеми «зникаючого градієнта» [53].

$$f(x) = \frac{e^{2x}-1}{e^{2x}+1} \tag{2.12}$$

Зрізаний лінійний вузол (Rectified Linear Unit, ReLU) – нелінійна функція, яка обмежує вхідні значення в діапазон $[0, \infty)$, а всі від'ємні значення перетворюються на нуль. Перевагою даної функції є легкість обчислення та вирішення проблеми «зникаючого градієнта» для більшості позитивних значень, проте може виникнути інша проблема коли нейрони назавжди залишаються в неактивному стані (значення стає постійно нульовим) [53].

$$f(x) = \max(0, x) \tag{2.13}$$

Нормована експоненційна функція (Softmax) – використовується переважно в останньому шарі нейронних мереж для задач класифікації. Мапує вхідні значення у вектор ймовірностей. Функція перетворює виходи в ймовірності, які сумуються до 1, що зручно для інтерпретації виходу як ймовірностей класів. Але вона може бути вразлива до перенасичення, де одна з ймовірностей стає домінуючою [53].

$$f(x) = \frac{e^{x}}{\sum_{j=1}^{K} e^{x_j}} \tag{2.14}$$

Кожна з цих активаційних функцій має свої унікальні властивості та підходить для різних типів задач і архітектур нейронних мереж.

Існує множина різних архітектур нейромереж, які можна використовувати для прогнозування попиту, кожна із яких має свої переваги

та недоліки. Розглянемо лінію розвитку нейронних мереж, використовуваних для прогнозування часових рядів.

— Багатошаровий перцептрон (Multilayer Perceptron, MLP) – це один з класичних типів штучних нейронних мереж, який використовується для вирішення різноманітних завдань, включаючи класифікацію, регресію, а також прогнозування часових рядів. Архітектура MLP складається з кількох шарів нейронів: вхідного шару, одного або більше прихованих шарів та вихідного шару. Шари нейронів багатошарового перцептрону організовані за принципом feed-forward, тобто дані передаються вперед, без зворотних зв’язків. На кожному шарі відбувається лінійна трансформація даних за допомогою ваг, а потім застосовуються нелінійні функції активації.

Багатошаровий перцептрон є доволі універсальною архітектурою, що дозволяє використовувати її для вирішення багатьох задач, включаючи прогнозування часових рядів; він простий у реалізації і має більшу швидкість навчання, але при цьому, оскільки багатошаровий перцептрон не спеціалізований саме для вирішення проблем прогнозування, його точність може бути нижчою за більш складніші архітектури. Архітектура багатошарового перцептрона має такий вигляд (див. рис. 2.1):

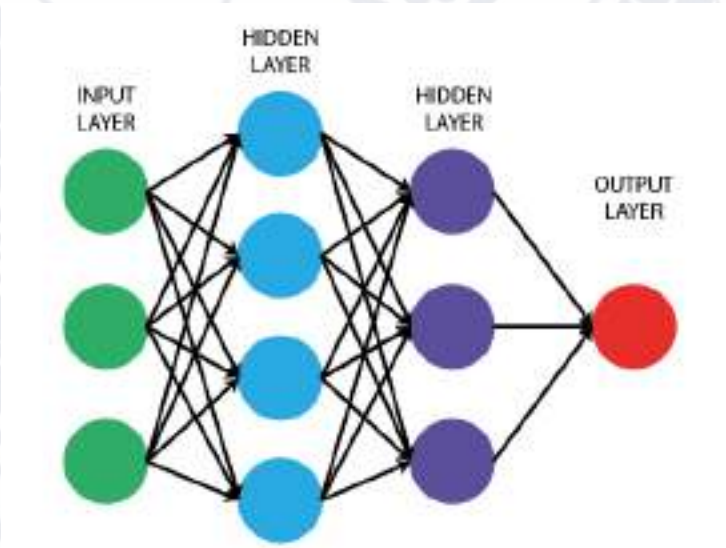

Рис. 2.1. Структура багатошарового перцептрона [50]

— Згорткова нейронна мережа (Convolutional Neural Network, CNN) – архітектура нейронних мереж, що була початково розроблена для обробки зображень, але її успішно адаптували для різних завдань, таких як обробка сигналів і навіть прогнозування часових рядів [50]. Згорткові фільтри даної нейромережі можуть вивчати короткі часові залежності, виявляючи локальні тренди, цикли або інші патерни в даних.

Згорткова нейронна мережа складається з таких шарів нейронів як: вхідний шар, шари згортки (convolutional layers), шар нелінійної активації, шар агрегування (pooling layers), повнозв'язні шари (fully connected layers) та вихідний шар. На вхідний шар який подаються початкові дані. Шари згортки є основними шарами даної мережі, у яких до вхідних даних застосовуються фільтри (kernels), що дозволяє виділяти локальні ознаки (features). Кожен фільтр зчитує невелике вікно даних і застосовує операцію згортки для виявлення патернів у цьому локальному вікні. У випадку часових рядів використовуються одновимірні згортки (1D Convolution), де фільтри переміщаються уздовж часової осі, що дозволяє моделі вивчати послідовні залежності. Після кожної згорткової операції у шарі активації використовується нелінійна активаційна функція, що додає нелінійність у модель і допомагає вивчати складні патерни. Агрегувальний шар здійснює на вхідних даних функцію агрегування (pooling). Pooling зменшує розмірність вхідних даних, що дозволяє знизити обчислювальні витрати та виділяти найбільш важливі ознаки. Після кількох згорткових і pooling шарів, отримані ознаки передаються до одного або кількох повнозв'язних шарів. Ці шари об'єднують інформацію з усіх попередніх шарів і приймають рішення (наприклад, прогнозують наступне значення часового ряду). В кінці йде вихідний шар, який виводить нові дані. Архітектура згорткової нейронної мережі виглядає таким чином (див. рис. 2.2):

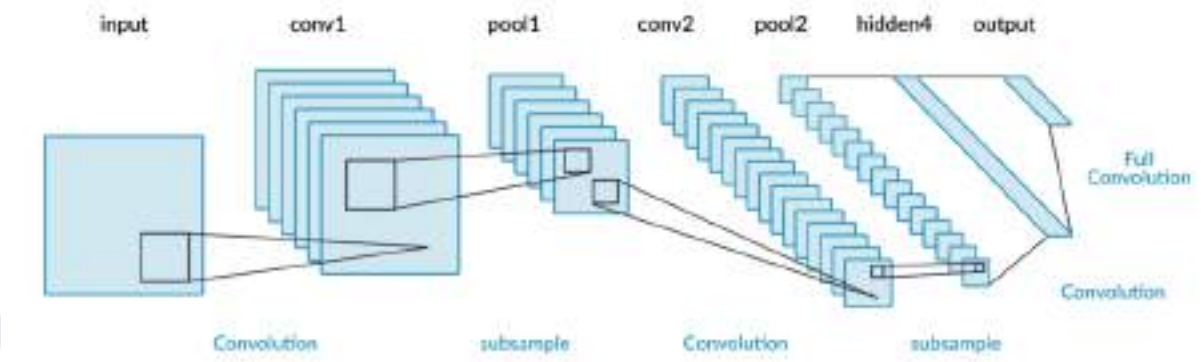

Рис. 2.2. Структура згорткової нейронної мережі [50]

— Архітектура рекурентної нейронної мережі (Recurrent Neural Network, RNN) є спеціальною формою нейронних мереж, яка ефективно працює з послідовними даними, такими як часові ряди, текст або звукові сигнали. Ключовою особливістю RNN є петлі (рекуренція), які дозволяють зберігати інформацію з попередніх кроків обробки даних, що робить цю архітектуру здатною до аналізу залежностей між елементами послідовності.

Як і в звичайних нейронних мережах, RNN містить нейрони, які з’єднані між собою і які формують вхідний шар, вихідний шар та декілька прихованих шарів. Проте у RNN вони також мають зворотні зв’язки з самим собою (recurrent connection), що дозволяє їм використовувати інформацію з попередніх кроків (див. рис. 2.3).

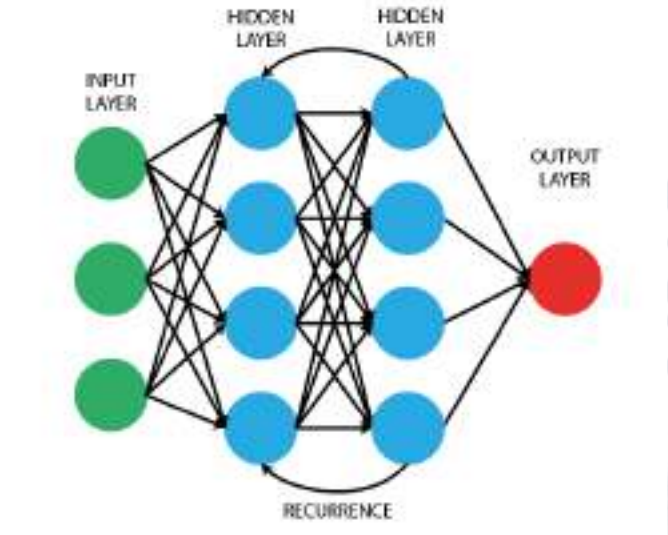

Рис. 2.3. Структура рекурентної нейронної мережі [53]

— Вентильний рекурентний вузол (Gated Recurrent Unit, GRU) – це тип рекурентної нейронної мережі, який розроблений для ефективного вирішення проблем, пов'язаних з довготривалою залежністю в послідовностях. Основна відмінність GRU полягає в тому, що вона використовує два «гейти» (вентилі) для управління потоком інформації, які допомагають контролювати, скільки інформації з попереднього стану мережа повинна зберігати або ігнорувати. Це дозволяє більш ефективно працювати з послідовними даними, наприклад, при прогнозуванні часових рядів.

Окрім стандартних вхідних, прихованих та вихідних шарів, мережа GRU також має дві затворні структури: затвор оновлення (update gate) та затвор скидання (reset gate). Ці затвори дозволяють мережі вибирати, яка інформація з попередніх часових моментів повинна передаватися далі, а яка — ігноруватися. Затвор оновлення визначає, скільки нової інформації з поточного часу потрібно зберегти в стані прихованого шару, а скільки інформації зі старого стану варто передати далі. Він працює на рівні контролю того, як багато «пам’яті» залишати на довгострокову перспективу. Затвор скидання контролює, скільки інформації з минулого стану потрібно «забути».

Архітектурою вентильного рекурентного вузла є послідовністю структурних блоків, які об’єднують у собі затвори оновлення та скидання. Дані блоки мають наступний вигляд (див. рис. 2.4):

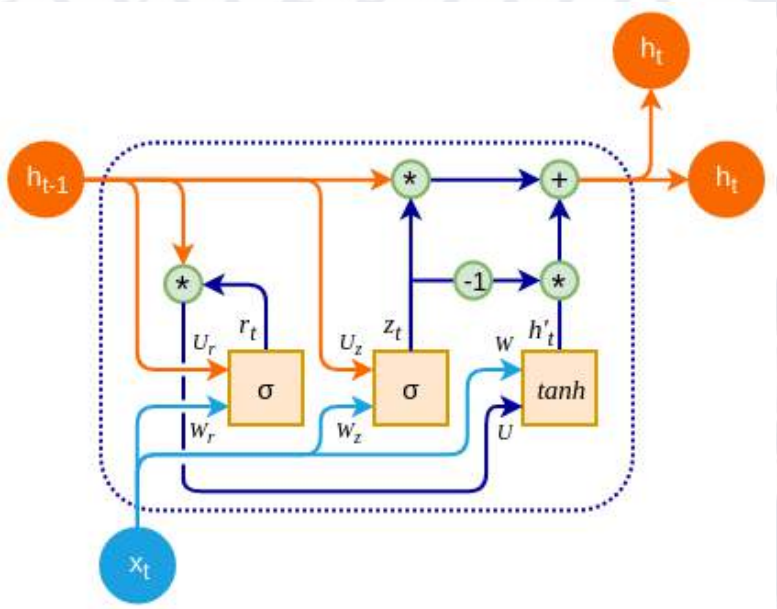

Рис. 2.4. Структура блоку вентильного рекурентного вузла [53]

— Архітектура нейронної мережі «Довга короткочасна пам’ять» (Long Short-Term Memory, LSTM) також була створена для вирішення проблеми довготривалої залежності в послідовних даних, з якою стикалися традиційні рекурентні нейронні мережі (RNN). LSTM використовують спеціальні механізми для збереження важливої інформації на тривалий період часу, що робить їх ефективними у задачах обробки послідовностей, таких як прогнозування часових рядів.

LSTM складається з набору осередків пам’яті (memory cells), де кожен осередок містить три основні елементи управління: вхідний, вихідний і забувальний шлюзи (gates). Вони відповідають за передачу та управління інформацією через мережу. Вхідний шар контролює, яка частина нової інформації буде додана до стану пам’яті (cell state). Він відповідає за модифікацію поточного стану осередку пам'яті новими даними з вхідної послідовності. Шлюз забування вирішує, яка частина інформації, що

зберігається в пам'яті, повинна бути забута. Цей шлюз дозволяє мережі очищати стан пам’яті від старих або нерелевантних даних. Вихідний шлюз вирішує, яка частина стану осередку пам’яті буде використана для генерації вихідного значення на поточному кроці. Вихідний шлюз також впливає на стан прихованих нейронів, які передаються далі через часові кроки.

Структура осередків пам’яті мережі «довга короткочасна пам’ять» має наступний вигляд (див. рис. 2.5):

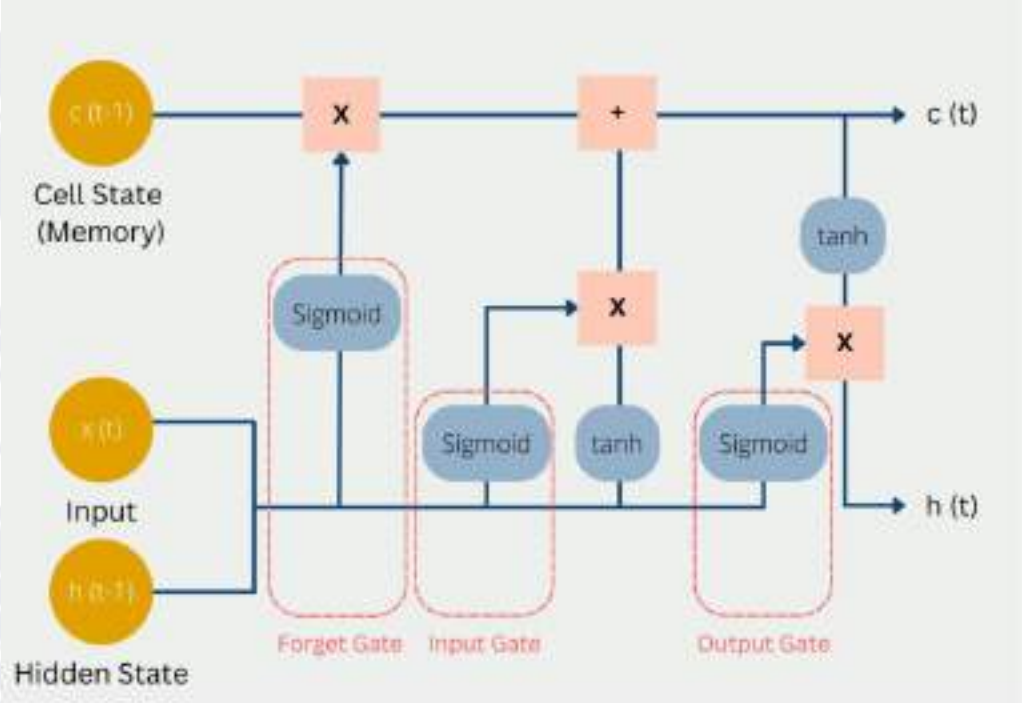

Рис. 2.5. Структура осередку пам’яті нейромережі «довга короткочасна пам’ять» [53]

— Розширена довга короткочасна пам’ять (Extended Long Short-Term Memory, xLSTM) – це передова розробка в гілці розвитку рекурентних мереж, яка покращує обробку інформації архітектури LSTM шляхом додання нових типів вузлів [3, 5]. xLSTM збільшує свою ефективність через інтеграцію двох інших модифікацій довгої короткочасної пам’яті: скалярної LSTM (Scalar LSTM, sLSTM) та матричної LSTM (Matrix LSTM, mLSTM), які використовують у своїй структурі відповідні блоки sLSTM та mLSTM, що є покращеннями оригінальної клітини пам’яті мережі LSTM. Основою

розширеної довгої короткочасної пам’яті є блоки xLSTM, які містять в собі компоненти вузлів sLSTM і mLSTM, що дозволяє їм приймати форму того чи іншого вузла. В залежності від атрибутів набору даних ми можемо перетворювати блоки xLSTM або у sLSTM, або у mLSTM. Загальна архітектура xLSTM – це послідовність блоків xLSTM, які приймають форми mLSTM або sLSTM у різних пропорціях в залежності від вхідних даних (див. рис. 2.5). Таким чином xLSTM оптимізує управління інформаційними потоками, інтегруючи різні типи вузлів від інших модифікованих версій LSTM.

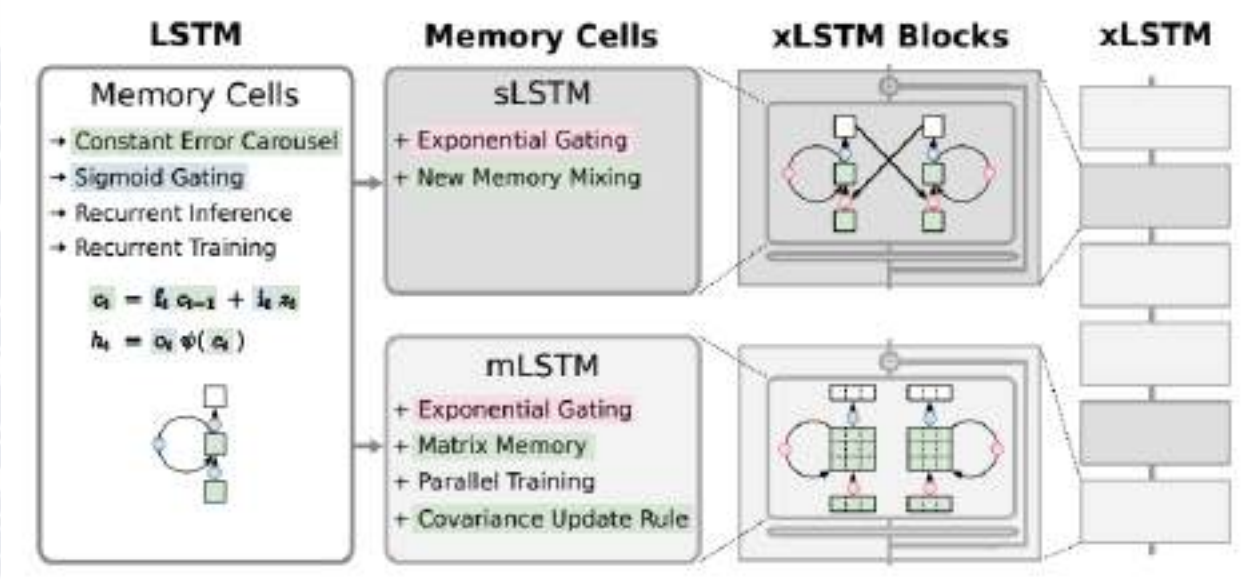

Рис. 2.6. Структура нейромережі xLSTM разом з блоками sLSTM і mLSTM, із яких вона складається [5]

Версія sLSTM використовує скалярну пам’ять і експоненціальне стробування для керування довгостроковими залежностями та керування відповідною пам’яттю для історичної інформації. sLSTM покращує спосіб змішування та оновлення пам’яті, дозволяючи точніше зберігати та обробляти дані. Версія mLSTM шляхом перетворення комірок пам'яті в матричну структуру підвищує здатність мережі обробляти операції паралельно, що значно прискорює обробку. Матрична структура в mLSTM розширює ємність пам'яті, а також й підвищує ефективність пошуку та зберігання інформації. Ця

структурна зміна дозволяє краще обробляти завдання зі складними структурами даних або довгостроковими залежностями.

Одним із перспективних напрямків розвитку у сфері машинного навчання є розробка гібридних нейронних мереж. Гібридними називають мережі, які поєднують у собі ознаки декількох типів нейромереж або нейронної мережі з іншими статистичними методами машинного навчання. Така інтеграція різних моделей дозволяє використовувати їхні сильні сторони для обробки різноманітних типів даних і врахування складних взаємозв’язків, що робить прогнозування попиту більш ефективним і точним. У сфері прогнозування попиту гібридні моделі здатні більш точно адаптуватися до мінливих умов ринку, що робить їх особливо перспективними в умовах високої невизначеності та швидких змін у споживчих уподобаннях.

Гібридні нейронні мережі мають величезний потенціал для підвищення точності прогнозування, оскільки вони можуть одночасно враховувати різні аспекти даних, такі як сезонність, тренди, раптові зміни в поведінці споживачів тощо. Окрім цього, поєднання з іншими підходами, наприклад, методами оптимізації або традиційними економетричними моделями, може додатково збільшити точність і стійкість прогнозів, особливо у випадках, коли кількість вхідних даних є обмеженою або вони містять велику кількість шуму.

Розглянемо приклади таких гібридних моделей нейронних мереж.

— LSTM-CNN – гібридна нейронна мережа, яка поєднує в собі переваги двох архітектур: згорткових нейронних мереж (CNN) і довготривалої короткочасної пам'яті (LSTM) [22]. Ця архітектура використовує CNN для виявлення локальних, короткотермінових патернів у даних, які є критично важливими для часових рядів з високою мінливістю або сезонністю. CNN працює шляхом застосування фільтрів до послідовностей даних, виявляючи важливі характеристики (наприклад, різкі зміни чи локальні піки). Після цього вихід CNN передається до LSTM, яка відома своєю здатністю запам'ятовувати довготривалі залежності у послідовностях. Це особливо важливо для часових рядів, де поточні значення залежать від попередніх значень у довгому

часовому контексті. LSTM має механізми збереження та забування інформації, що дозволяє їй зберігати релевантні патерни у довгостроковій перспективі і одночасно позбавлятися зайвих.

Комбінація CNN і LSTM дозволяє моделі бути чутливою до як короткострокових, так і довгострокових характеристик у часових рядах. У задачах прогнозування CNN може виявити неочевидні локальні патерни, а LSTM допоможе визначити загальні тенденції і тривалі залежності. Це забезпечує більшу точність прогнозів, оскільки модель не лише розуміє локальні зміни, але й може враховувати важливі зміни, що траплялися раніше.

Структура гібридної моделі LSTM-CNN виглядає таким чином (див. рис. 2.7):

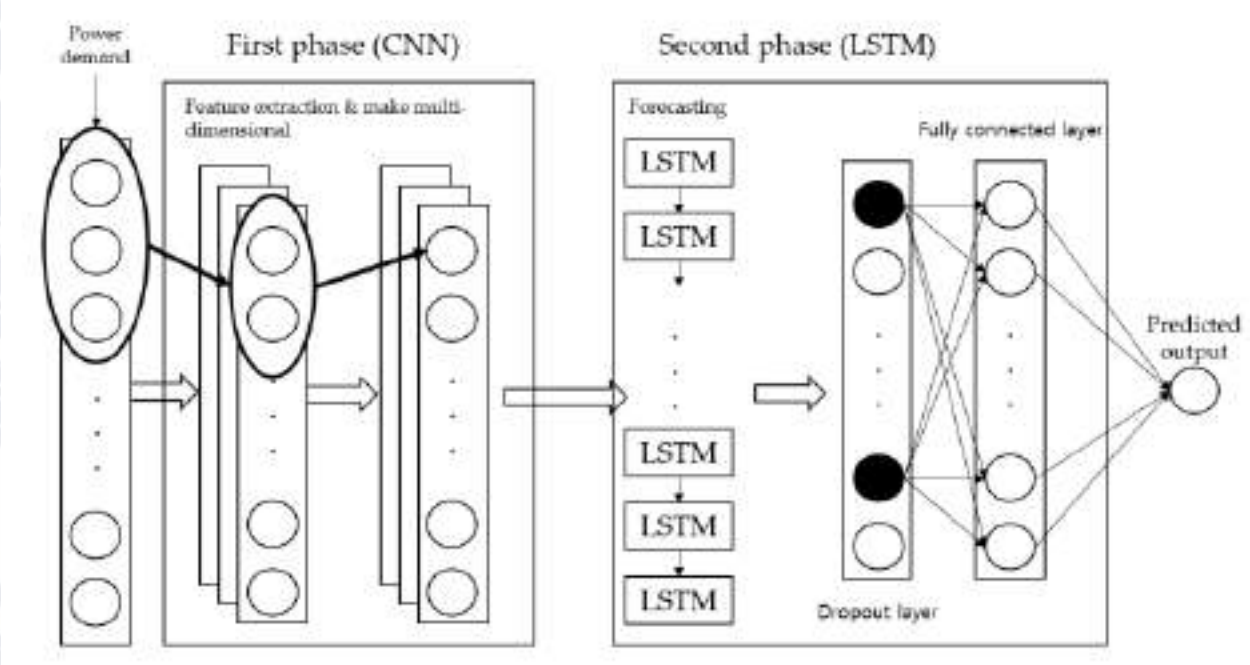

Рис. 2.7. Структура одного з варіантів реалізації гібридної нейронної мережі LSTM-CNN [22]

— ES-RNN (Exponential Smoothing – Recurrent Neural Network) – це гібридна модель, яка поєднує в собі моделі експоненційного згладжування (ES) та рекурентної нейронної мережі (RNN) [40, 41]. Модель розроблена для того, щоб обробляти як короткострокові, так і довгострокові залежності в часі,

що є ключовим фактором для точного передбачення тенденцій у часових рядах.

Експоненційне згладжування використовується для виділення основних трендів і сезонних компонентів у даних. Завдяки своїм властивостям, експоненційне згладжування здатне адаптивно враховувати зміни в трендах і циклах, зменшуючи вплив короткострокових коливань. З іншого боку, рекурентна нейронна мережа добре підходить для обробки складних часових залежностей і взаємозв'язків між попередніми та поточними значеннями в ряді. RNN здатна запам'ятовувати інформацію про минулі стани завдяки своїй рекурентній структурі, що дозволяє їй краще моделювати послідовні процеси. Гібридизація ES і RNN у моделі ES-RNN дозволяє використовувати експоненційне згладжування для відбору та перетворення вхідних даних, які надалі обробляються рекурентною нейронною мережею для точного прогнозування. ES ефективно фільтрує дані, усуваючи шум і залишкові коливання, після чого RNN працює з більш чистими й структурованими даними, що покращує здатність мережі передбачати складні нелінійні залежності.

Структура гібридної моделі ES-RNN виглядає таким чином (див. рис. 2.8):

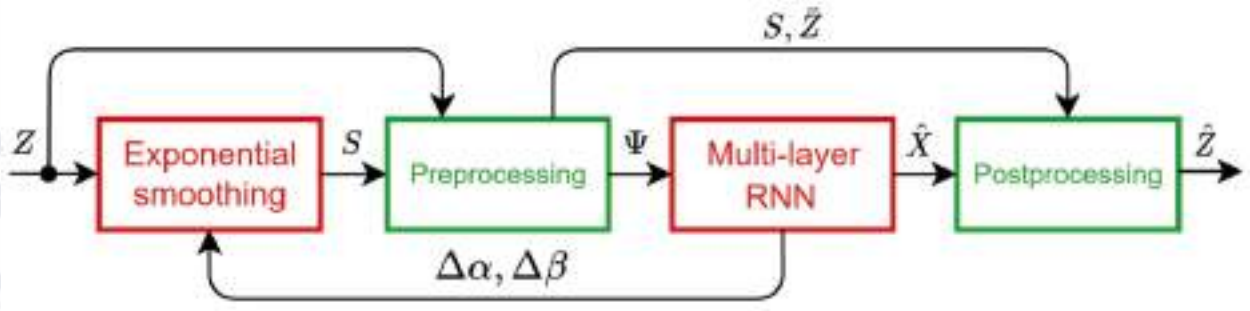

Рис. 2.8. Структура реалізації гібридної нейронної мережі ES-RNN

Ціллю нашого дослідження є розробка гібридної моделі нейронної мережі ES-xLSTM (Exponential Smoothing – Extended Long Short-Term Memory), використовуючи розглянутий раніше процес гібридизації на нейромережі розширеної довгої короткочасної пам’яті (xLSTM) як одної з новітній

архітектур, яка показує хороші результати, і яку ми повинні адаптувати для задачі прогнозування часових рядів, для поєдання її разом з більш традиційним методом прогнозування – експоненційним згладжуванням (ES). Аналогічно до гібридної моделі ES-RNN, яка покращує результати рекурентної мережі шляхом інтеграції її з експоненційним згладжуванням, ми можемо зробити теж саме виконавши інтеграцію ES з мережею xLSTM.

Мережа ES-xLSTM може забезпечити синергію між сильними сторонами статистичних моделей і глибоких нейронних мереж, що робить її особливо ефективною для аналізу часових рядів. Експоненційне згладжування добре справляється із короткостроковими тенденціями та сезонними коливаннями, адже воно фокусується на останніх спостереженнях, забезпечуючи просте і швидке прогнозування. У свою чергу, нейромережа xLSTM здатна захоплювати довгострокові залежності та складні нелінійні зв’язки в даних завдяки своїй розширеній архітектурі пам’яті, яка перевершує стандартні LSTM у збереженні довготривалої інформації.

Об’єднання цих підходів дозволяє моделі зберегти адаптивність і чутливість до останніх змін у даних через компонент ES, водночас компенсуючи його слабкість у врахуванні складних взаємозв’язків, які розкриваються через xLSTM. Таким чином, ES забезпечує базову стабільність прогнозів і знижує ризик перенавчання нейромережі, виступаючи як фільтр для шуму в даних, тоді як xLSTM додає глибину та гнучкість, враховуючи нелінійні патерни. Ця інтеграція також сприяє кращій узагальнюваності моделі, оскільки ES може виконувати функцію попереднього вирівнювання або нормалізації даних, спрощуючи завдання для xLSTM. У результаті модель ES-xLSTM стає не лише більш точною, але й надійною до змінних умов.

2.3 Метрики для визначення точності прогнозування

Отже, було розглянута множина різних методик та алгоритмів прогнозування. Більшість із них може мати багато різних варіантів, залежно від прогнозованої моделі попиту. Тому виникає питання, як вирішити, чи

справляється краще один алгоритм з прогнозуванням конкретного попиту за певний період часу, ніж інший, тобто нам потрібно порівняти методи та знайти ті, які найкраще працюють у нашому контексті. Таким чином, нам потрібні ключові показники ефективності (Key Performance Indicators, KPI) якості прогнозу або вимірювання помилок. Вимірювання ефективності є дуже важливим для управління: якщо її не вимірювати, то буде важко встановлювати цілі та координувати діяльність для виконання мети. Це твердження справедливе майже для будь-якого аспекту бізнесу, але особливо для прогнозування.

Говорячи про якість прогнозу, перше, що ми повинні розрізнити, це різниця між точністю прогнозу та його зміщенням. Точність прогнозу вимірює різницю між прогнозом і фактичними даними. Точність дає уявлення про величину помилок, але не про їх загальний напрям. Зміщення представляє загальний напрямок помилок. Він визначає, чи прогнози в середньому зависокі – прогнози перевищують попит, чи занадто низькі – вони занижують попит. Щоб відстежити якість прогнозу, ми повинні виміряти як його зміщення, так і точність.

Найпростішою метрикою, яка використовується для визначення точності та зміщення, є метрика помилки прогнозу, яка окремо обчислюється для кожної точки прогнозу. Будемо обчислювати помилку прогнозу як різницю значення справжнього попиту за певну одиницю часу від значення прогнозу для даного періоду часу [49]:

$$Error = e_t = f_t - d_t \tag{2.15}$$

де e_t – помилка прогнозу за період t;

f_t – прогноз за період t;

d_t – фактичні дані за період t.

При цьому визначенні помилка буде позитивною, коли прогноз перевищує попит; помилка буде від'ємною, якщо прогноз буде меншим за попит.

Значення загального зміщення прогнозу обчислюється шляхом усереднення помилки прогнозу за кілька періодів. Розгляд зміщення за кілька періодів зазвичай більш релевантний, ніж для одного періоду. Постійне зміщення може вказувати на те, що щось не так із системою прогнозування або процесом прогнозування. Зміщення можна розрахувати наступним чином, поділивши суму помилок часового ряду на кількість точок в цьому ряді [49]:

$$Bias = \frac{1}{n}\sum e_t \tag{2.16}$$

Якщо обчислювати зміщення на основі даної формули, ми отримуємо абсолютне значення зміщення. Але при такому визначенні зміщення доводиться порівнювати дане значення з середнім значенням попиту для кожного прогнозу, щоб розуміти наскільки великим є зміщення. Тому замість цього можна обчислювати метрику проценту зміщення, яка показує наскільки прогноз зміщений від середнього значення попиту в процентах. Можна обчислити його, поділивши загальну похибку на загальний попит [49]:

$$Bias\% = \frac{\sum e_t}{\sum d_t} \tag{2.17}$$

Дивитись лише на зміщення ще недостатньо для того, щоб оцінити якість прогнозу. Позитивна помилка в одному періоді може компенсувати негативну помилку в іншому періоді. Таким чином, модель прогнозу може досягти дуже низького зміщення і водночас не бути точною. Дуже зміщений прогноз вказує на те, що в моделі чи процесі прогнозування щось не так.

Середня абсолютна похибка (Mean Absolute Error, MAE) — це прямий показник для вимірювання точності прогнозу. Як випливає з назви, це середнє значення абсолютної похибки [49]:

$$MAE = \frac{1}{n}\sum|e_t| \tag{2.18}$$

Подібно до метрики зміщення можна обчислити метрику процентної середньої абсолютної похибки [49]:

$$MAE\% = \frac{\sum|e_t|}{\sum d_t} \tag{2.19}$$

Середня абсолютна похибка це дуже проста для обчислення метрика при визначенні точності прогнозу. Але при цьому мінімізація даного показника часто призводить до заниження прогнозу. Це пояснюється тим, що мінімізація абсолютної похибки подібна до націлювання на медіанний попит, який зазвичай нижчий за середній попит.

Середня абсолютна відсоткова похибка (Mean Absolute Percentage Error, MAPE) є одним із найбільш часто використовуваних показників для вимірювання точності прогнозу, незважаючи на те, що це не кращий показник точності. Дана метрика обчислюється як середнє значення індивідуальних абсолютних похибок, поділене на попит (кожен період ділиться окремо). Простіше кажучи, це середнє значення абсолютних відсоткових помилок (APE) [49]:

$$APE = \frac{|e_t|}{d_t}, \tag{2.20}$$

$$MAPE = \frac{1}{n}\sum\frac{|e_t|}{d_t} \tag{2.21}$$

MAPE ділить кожну помилку окремо на відповідний попит – ось чому цей показник такий недосконалий. Ці окремі поділки призводять до спотвореної ваги помилки: великі похибки в періоди низького попиту значно впливають на MAPE (оскільки помилка ділиться на низьке значення). Як наслідок, якщо необхідно мінімізувати MAPE, слід дотримуватися консервативних прогнозованих значень, щоб уникнути значних похибок у періоди низького попиту.

Середньоквадратична похибка (Root Mean Square Error, RMSE) обчислюється як квадратний корінь середньої квадратичної похибки прогнозу [49]:

$$RMSE = \sqrt{\frac{1}{n}\sum e_t^2} \tag{2.22}$$

Процентна середньоквадратична помилка:

$$RMSE\% = \frac{\sqrt{\frac{1}{n}\sum e_t^2}}{\frac{1}{n}\sum d_t} \tag{2.23}$$

Середньоквадратична похибка може бути дуже корисною з трьох основних причин. По-перше, вона приділяє більше значення найбільш значним помилкам прогнозу. Призначення більшої ваги найбільшим помилкам корисно, оскільки вплив помилок прогнозу на ланцюги поставок не є лінійним: великі помилки мають величезний вплив (втрата продажів, мертві запаси), тоді як невеликі помилки майже не впливають. По-друге, RMSE фактично пов’язана з більшістю формул оптимізації запасів, коли справа доходить до обчислення необхідної кількості страхових запасів. Крім того, RMSE є показником незміщеності: хороша оцінка RMSE часто корелює з незміщеним прогнозом.

При цьому на практиці, замість того, щоб оцінювати прогноз за допомогою одної метрики, обчислюється комбінований показник, підсумовуючи декілька метрик, наприклад, суму середньої абсолютної похибки та абсолютного зміщення:

$$Score = MAE + |Bias| \tag{2.24}$$

Використання цього показника буде практичним для легкого порівняння кількісно різних прогнозів. Використання цього показника є найкращою практикою під час оцінювання точності окремого продукту.

Висновки до розділу 2

У розділі розглянуте машинне навчання як сфера штучного інтелекту, що фокусується на розробці алгоритмів і моделей, здатних навчатися на основі даних і покращувати свою роботу без явного програмування. Також була розібрана окрема галузь машинного навчання - глибоке навчання, яке зосереджене на використанні нейронних мереж. Нейромережі – це моделі, натхненні біологічними нейронними мережами, що складаються з штучних нейронів – структурних одиниць, які приймають дані, виконують над ними прості обчислення та передають цю інформацію далі.

Була досліджена еволюція розвитку методів машинного навчання для вирішення задачі часових рядів, а саме були розглянуті такі засоби прогнозу як: наївний прогноз, ковзне середнє, експоненційне згладжування, авторегресійна лінійна модель, ARMA, ARIMA; а також моделі нейронних мереж адаптованих для прогнозу часових рядів: MLP, CNN, RNN, GRU, LSTM, xLSTM.

Сучасними досягненнями в цій галузі є гібридні нейронні мережі, які об’єднують характеристики кількох видів нейромереж або інтегрують нейронні мережі з традиційними методами машинного навчання. Така комбінація моделей дозволяє максимально ефективно використовувати їхні

переваги для аналізу різноманітних типів даних і врахування складних взаємозв’язків, що забезпечує більш точне та ефективне прогнозування попиту. Було розглянуто приклади таких гібридних моделей: LSTM-CNN, ES-RNN.

Крім того було запропоновано власну гібридну модель ES-xLSTM, яка б поєднувала передову нейронну архітектуру xLSTM з алгоритмом експоненційного згладжування, який потенційно може покращити обробку даних даної мережі.

Для визначення точності прогнозів часових рядів традиційно використовують такі метрики як: зміщення, середня абсолютна похибка (MAE), середня абсолютна відсоткова похибка (MAPE), та середньоквадратична похибка (RMSE).

РОЗДІЛ 3
РЕАЛІЗАЦІЯ МЕТОДІВ МАШИННОГО НАВЧАННЯ ДЛЯ ПРОГНОЗУВАННЯ ПОПИТУ

3.1 Підбір інструментів для реалізації алгоритмів прогнозування

Далі зосередимося на практичній реалізації алгоритмів прогнозування часових рядів. Спочатку розглянемо імплементацію більш простих алгоритмів прогнозу, які використаємо для порівняння точності прогнозу з розроблюваною моделлю, а також щоб краще розуміти наскільки покращилась ефективність прогнозування з новими спеціалізованими методами. Після цього спробуємо адаптувати нейронну мережу xLSTM для задачі прогнозування часових рядів. А потім почнемо розробку моделі ES-xLSTM, код якої буде використовувати наробки з програмної реалізації xLSTM.

Для реалізації різних методів машинного навчання з ціллю прогнозу попиту будемо використовувати мову програмування Python.

Python є однією з найбільш популярних мов для реалізації алгоритмів машинного навчання та створення нейронних мереж завдяки своїй простоті, гнучкості та широкій екосистемі бібліотек.

Одна з головних переваг Python полягає в наявності великих кількостей високоякісних бібліотек, таких як TensorFlow, Keras, PyTorch та Scikit-learn, що дозволяють швидко реалізувати найсучасніші підходи в машинному навчанні, включаючи глибокі нейронні мережі. Ці бібліотеки забезпечують доступ до потужних функцій, алгоритмів та інструментів, що робить процес розробки моделей значно швидшим і ефективнішим. Важливу роль грає його здатність до швидкої обробки та аналізу даних завдяки бібліотекам, таких як Pandas і NumPy, які спрощують маніпуляції з великими обсягами даних, що є невід'ємною частиною машинного навчання. Крім того існує активна спільнота розробників, яка постійно оновлює і вдосконалює ці бібліотеки. Це означає, що будь-який новий алгоритм або техніка машинного навчання швидко стають доступними для використання в Python.

Інтеграція з іншими інструментами також є важливою перевагою. Наприклад, Python легко інтегрується з системами для візуалізації даних, такими як Matplotlib та Seaborn, що дозволяє не тільки будувати й навчати моделі, а й наочно аналізувати результати. Використовуючи ці бібліотеки, можна швидко створювати графіки та діаграми, які допомагають краще зрозуміти поведінку моделі та прийняти рішення щодо її покращення.

При розробці будемо користуватися наступними сторонніми бібліотеками:

– numpy – це основна бібліотека для роботи з багатовимірними масивами та обчисленнями на них. NumPy забезпечує високу продуктивність завдяки оптимізованим математичним операціям над масивами. Дану бібліотеку будемо використовувати для швидких обчислень при роботі з масивами даних [32];

– pandas – це бібліотека для аналізу та маніпуляцій з даними, яка надає потужні інструменти для роботи з табличними даними та часом (DataFrame). Дана бібліотека буде використовуватись для роботи з нашим набором даних, до якого будуть застосовуватись методи прогнозування часових рядів. Будемо також використовувати вбудовані функції даної бібліотеки для реалізації деяких простих алгоритмів прогнозування таких як: moving-average forecast, weighted moving average та exponential smoothing [33];

– statsmodels – це бібліотека для статистичного аналізу та економетрики. Вона дозволяє користувачам створювати статистичні моделі, проводити тестування гіпотез та аналізувати результати за допомогою статистичних моделей. Будемо її використовувати для реалізації більш комплексних класичних методів машинного навчання таких як autoregressive linear forecasting, autoregressive moving-average та autoregressive integrated moving-average [47];

– sklearn – це бібліотека для машинного навчання з простим і ефективним інтерфейсом для вирішення завдань класифікації, регресії, кластеризації та

зменшення розмірності. Будемо її застосовувати при моделюванні багатошарового перцептрона (MLP) [37];

– tensorflow – це одна із самих популярних бібліотек Python для глибокого навчання. Вона дозволяє створювати та тренувати нейронні мережі для різноманітних завдань штучного інтелекту. Відповідно для реалізації класичних нейронних мереж, включаючи мережі CNN, RNN, GRU, LSTM, будемо використовувати дану бібліотеку [43];

– torch – це інша відома бібліотека для реалізації нейромереж, що спеціалізується для роботи з тензорами – багатовимірними масивами, що підтримують обчислення з автоматичним диференціюванням. Також torch надає можливість роботи з глибокими нейромережами, побудованими на системі автоградації на основі стрічки. Цю бібліотеку будемо використовувати при розробці моделей xLSTM та ES-xLSTM [35];

– matplotlib – це основна бібліотека для візуалізації даних у Python. Вона дозволяє створювати різноманітні графіки та діаграми для аналізу даних. І відповідно будемо її використовувати для відображення обчислених прогнозів, порівнюючи з оригінальними даними [30];

– ESRNN – стороння бібліотека, створена суцільно для реалізації гібридної нейронної мережі ES-RNN [12].

В якості набору даних, над яким будемо застосовувати методи машинного навчання для прогнозування, будемо використовувати наступний простий часовий ряд: це синтетичний набір даних місячного попиту на певний товар з 2009 по 2020 роки (див. рис. 3.1 та рис. 3.2). Загалом він зберігає в собі 144 значення.

	Month	Demand
0	2009-01	112
1	2009-02	118
2	2009-03	132
3	2009-04	129
4	2009-05	121
..	...	...
139	2020-08	606
140	2020-09	508
141	2020-10	461
142	2020-11	390
143	2020-12	432

Рис. 3.1. Загальний вигляд використовуваного набору даних

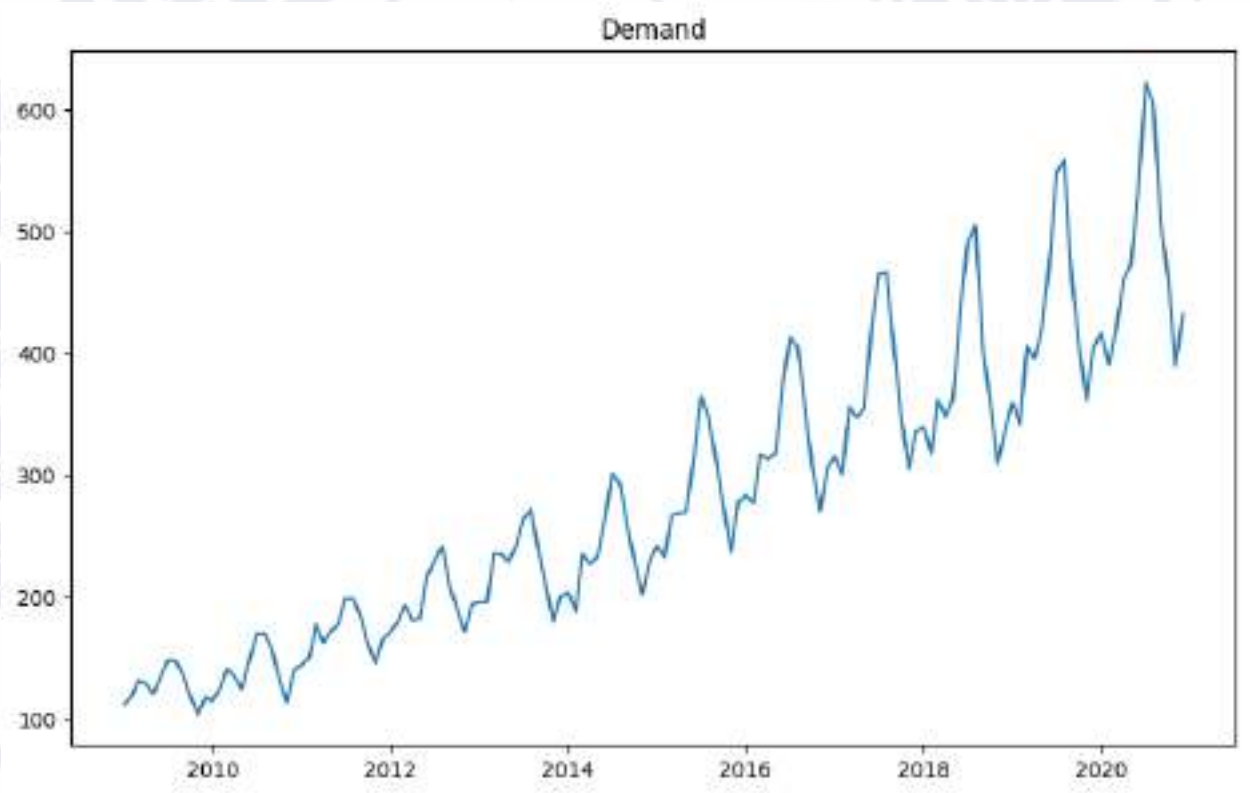

Рис. 3.2. Графік використовуваного набору даних

При аналізі часових рядів в них виділяють чотири компоненти [27]:

– тренди – це явні патерни змін в часових рядах на протязі довгого часу. Наприклад, коли спостерігається довгострокове зростання або спадання в даних;

– сезони – це регулярні сезонні зміни. Наприклад, зміни протягом різних днів тижня чи протягом різних пор року;

– цикли – це довгострокові патерни змін, які можуть повторюватись, подібно до сезонів, але які можуть мати різну протяжність і не прив'язані до певних періодів часу. Прикладами таких патернів є бізнес-цикли, які мають повторювані елементи росту, рецесії та відновлення;

– нерегулярності – це хаотичні зміни часових рядів, які можуть бути викликані неочікуваними подіями або які представляють простий шум в даних.

Обраний набір даних чітко відображає компоненти тренду, сезонності та нерегулярностей, що робить його добре підходящим для демонстрації на ньому методів прогнозування часових рядів.

Для визначення точності прогнозу будемо використовувати наступні метрики, які вже зазначались раніше:

– Bias% – процентне зміщення;

– MAE% – процентна середня абсолютна похибка;

– MAPE – середня абсолютна відсоткова похибка;

– RMSE% – процентна середньоквадратична похибка;

– комбінований показник, який є середнім значенням попередніх чотирьох метрик.

Функція kpi() для обчислення цих метрик має наступний вигляд. Вона приймає в себе частину оригінального набору даних за період часу, який ми хочемо спрогнозувати, та сам прогноз за цей період. Функція обчислює різницю між отриманими послідовностями даних та обраховує метрики відповідно до розглянутих раніше формул:

```
def kpi(original, forecast):
  length = len(original)
  original_sum = sum(original)
  error = original - forecast
  error_sum = sum(error)
  absolute_error = np.abs(error)

  bias = round(error_sum/original_sum * 100, 2)
  mae = round(sum(absolute_error)/original_sum * 100, 2)
  mape = round(sum(absolute_error/original)/length * 100, 2)
```

```
  rmse = round(np.sqrt(sum(pow(absolute_error, 2))/length) /
(original_sum/length) * 100, 2)
  score = round((bias+mae+mape+rmse)/4, 2)

  print(f"Bias%: {bias}%")
  print(f" MAE%: {mae}%")
  print(f" MAPE: {mape}%")
  print(f"RMSE%: {rmse}%" )
  print(f"Score: {score}%")
```

3.2 Реалізація класичних алгоритмів прогнозування часових рядів

Отже, розглянувши інструменти, які будемо використовувати, перейдемо до спроб реалізації окремих алгоритмів машинного навчання для прогнозування часових рядів.

— Наївний прогноз (Naïve Forecasting):

Даний алгоритм дуже простий у реалізації. Для нього можна використати тільки інструменти бібліотеки NumPy для швидкого копіювання попередніх фактичних даних у якості прогнозу на наступний період.

Проте потрібно розуміти, що є різниця у використанні даного та інших подібних простих алгоритмів для прогнозування на коротких строк та на довгий строк. При прогнозуванні на короткий строк, наприклад, як в нашому випадку, коли нам потрібно спрогнозувати попит на наступний місяць, маючи значення попиту на поточний місяць, подібний тип прогнозу може не сильно відрізнятись від фактичних даних. Але на довгий строк подібний алгоритм втрачає будь-яку прогнозуючу спроможність. Те ж саме відноситься і до інших подібних простих алгоритмів прогнозування, таких як ковзне середнє, експоненційне згладжування, тощо.

Нижче можна побачити результати реалізації алгоритму наївного прогнозу на короткий (див. рис. 3.3 і таб. 3.1) та на довгий строки (рис. 3.4 і таб. 3.2). У першому варіанті будемо прогнозувати попит на наступний місяць, знаючи фактичні дані за поточний місяць, таким чином виконуючи прогноз всього часового ряду. У другому варіанті з певної точки у часі будемо намагатися спрогнозувати попит на всі наступні місяці, використовуючи

фактичне значення попиту на поточний місяць. Якщо у першій ситуації прогноз просто відображає криву фактичного попиту із запізненням, то у другій через недостачу додаткових даних, які б могли покращити результат, результатом роботи алгоритму є проста пряма, яка ніяким чином не співвідноситься з коливаннями попиту.

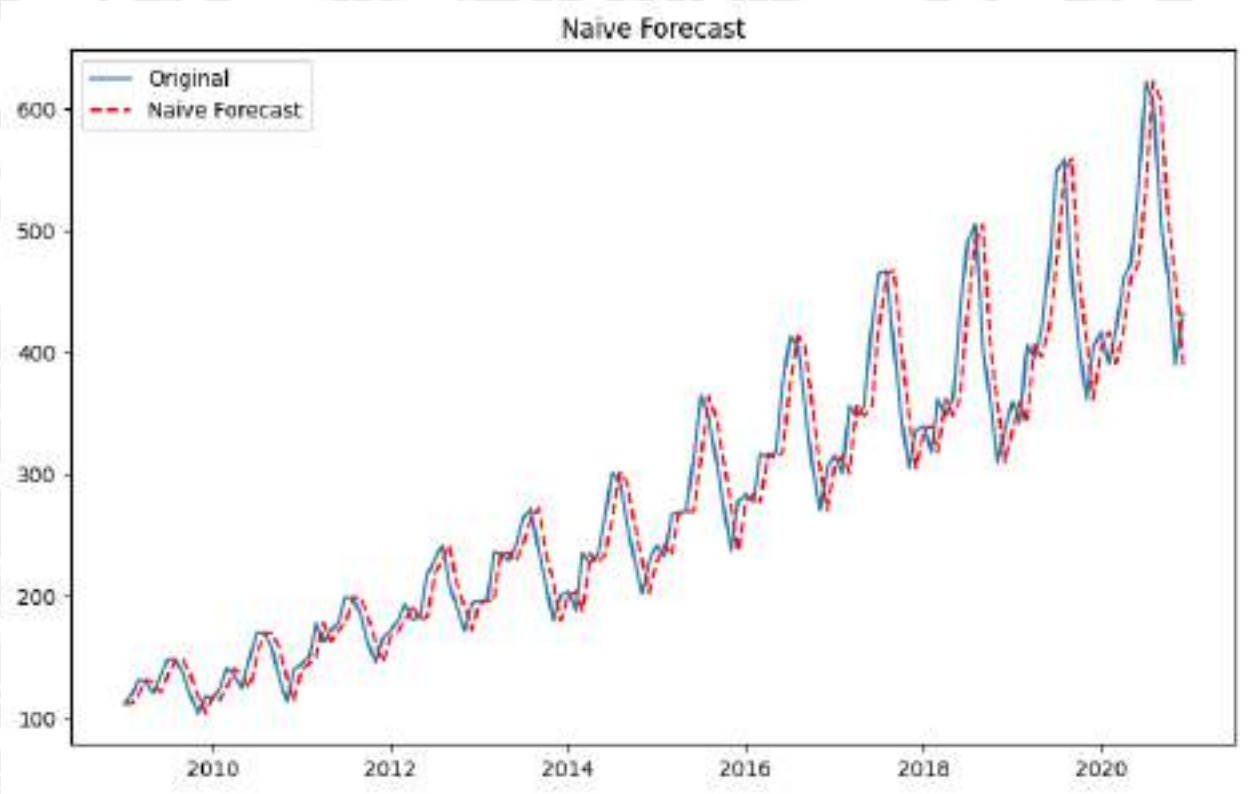

Рис. 3.3. Графік короткострокового наївного прогнозу

Таблиця 3.1 – Показники рівня похибки короткострокового наївного прогнозу

Метрика	**Значення**
Bias%	0.7%
MAE%	9.18%
MAPE	9.01%
RMSE%	12%
Середнє значення	7.72%

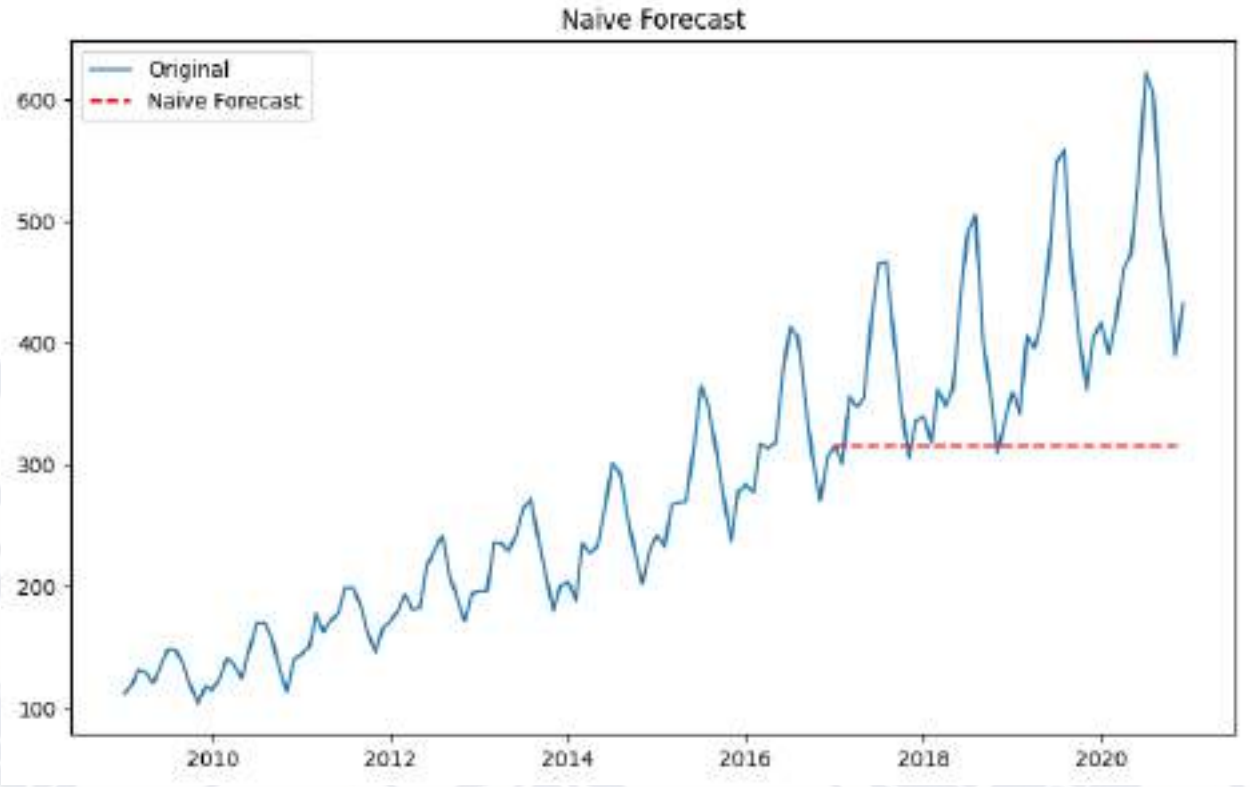

Рис. 3.4. Графік довгострокового наївного прогнозу

Таблиця 3.2 – Показники рівня похибки довгострокового наївного прогнозу

Метрика	**Значення**
Bias%	23.33%
MAE%	23.72%
MAPE	21.18%
RMSE%	30.25%
Середнє значення	24.62%

Підсумовуючи, можна сказати, що в середньому алгоритм наївного прогнозу є доволі мало ефективним через неспроможності до довгострокового прогнозування та проблеми при прогнозуванні стрімких змін у часовому ряді. Даний алгоритм застосовується в основному в якості порівняння при розгляді інших алгоритмів прогнозу.

Надалі при розгляді подібних простих алгоритмів будуть розглядатись прогнозування тільки на короткий строк, оскільки використання їх на довгий строк не має сенсу через вкрай низьку точність.

— Сезонний наївний прогноз (Seasonal Naïve Forecast):

Подібно до звичайного наївного прогнозу даний алгоритм легкий у реалізації простим копіюванням попередніх фактичних даних в якості прогнозу на наступний період, і єдиним додатковими інструментами, якими ми можемо скористатися, є модулі такі як NumPy, які просто дозволяють пришвидшити всі обчислення.

Графік сезонного наївного прогнозу, а також його показники похибки, мають такий вигляд (див. рис. 3.5 і таб. 3.3):

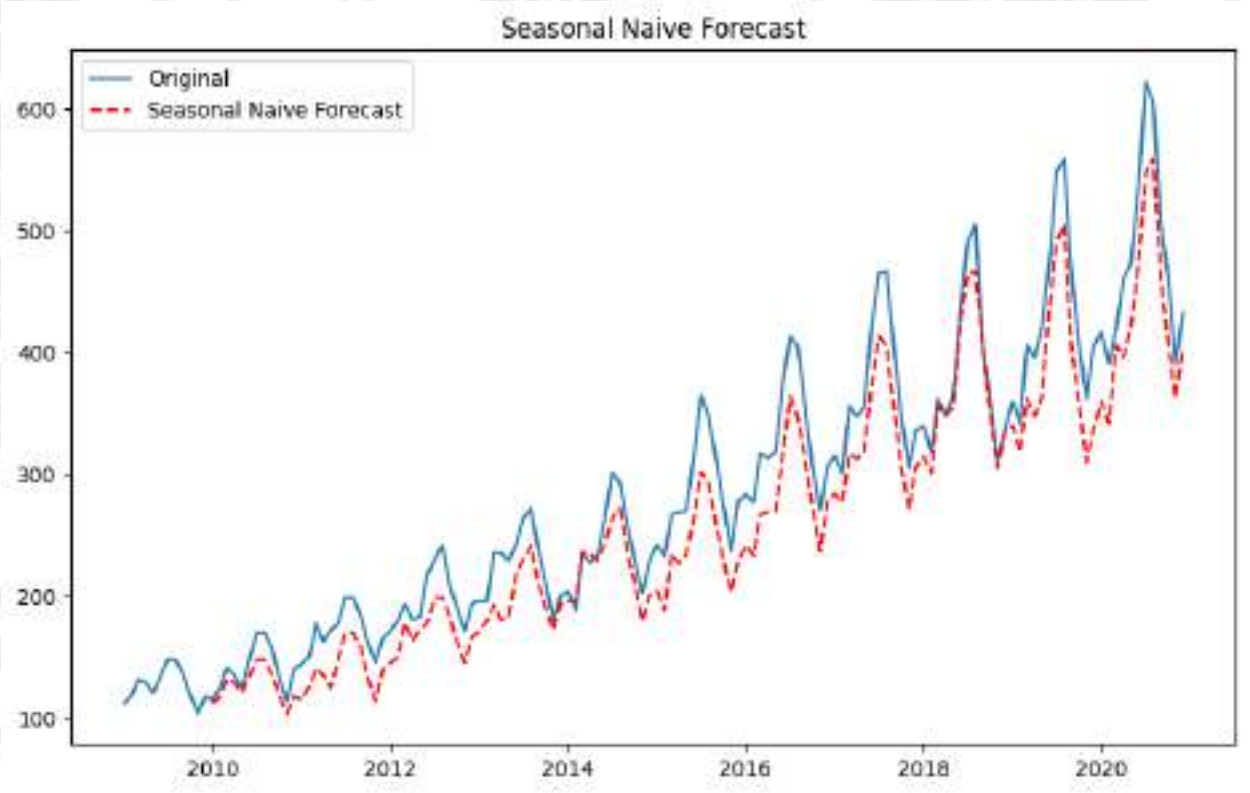

Рис. 3.5. Графік сезонного наївного прогнозу

Таблиця 3.3 – Показники рівня похибки сезонного наївного прогнозу

Метрика	**Значення**
Bias%	10.8%
MAE%	10.88%
MAPE	11.25%
RMSE%	12.34%
Середнє значення	11.32%

— Ковзне середнє (Moving-Average Forecast):

Для реалізації даного алгоритму прогнозування можна застосувати функції бібліотеки pandas, а саме функцію rolling(), який дозволяє виконувати розрахунки ковзного вікна, та функцію mean(), який обчислює середнє арифметичне значення для даних створених вікон. Основний принцип роботи функції rolling() полягає в тому, що він бере набір значень у поточному «вікні» (або вибірці) з даних і дозволяє застосувати до них певну агрегуючу функцію, наприклад, як в даному випадку, функцію mean(), яка обчислює середнє значення для кожного складеного вікна з n елементів заданого набору даних. Функція rolling() приймає в себе одне значення – довжину вікна, тобто кількість елементів n, яких буде складатись кожне вікно. У нашому випадку будемо використовувати довжину вікна, що дорівнює 4.

Код застосування функції ковзного вікна буде мати такий вигляд:

```
sma = dataset.rolling(window = 4).mean()
```

де dataset – це наш набір даних, по якому робиться прогноз, а sma – вже є результуючим прогнозом простим алгоритмом ковзного середнього.

Графік прогнозу, а також його показники похибки, можна побачити нижче (див. рис. 3.6 і таб. 3.4).

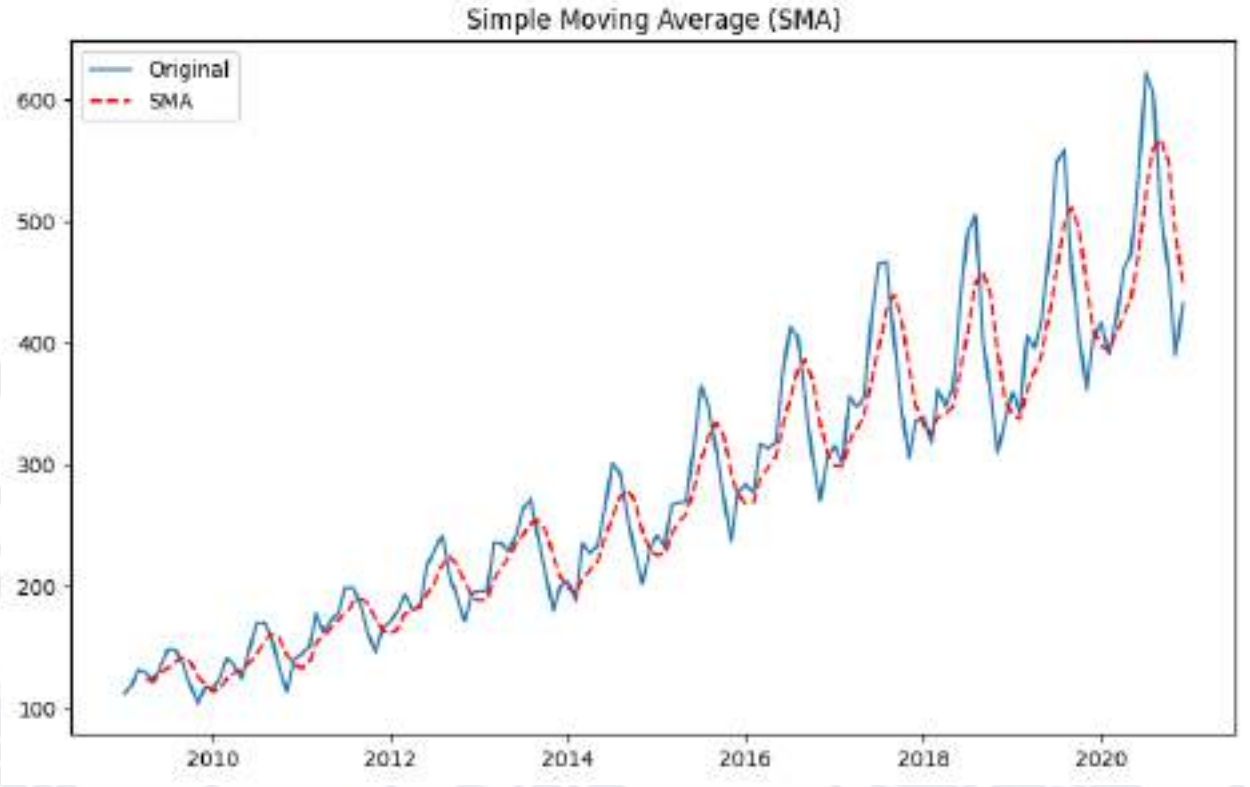

Рис. 3.6. Графік прогнозу ковзного середнього

Таблиця 3.4 – Показники рівня похибки прогнозу ковзного середнього

Метрика	**Значення**
Bias%	1.04%
MAE%	10.36%
MAPE	10.12%
RMSE%	12.68%
Середнє значення	8.55%

— Зважене ковзне середнє (Weighted Moving-Average):

Для реалізації даного алгоритму прогнозування будемо застосовувати функції бібліотеки pandas, а саме функцію rolling(), для обчислень ковзного вікна, як і для попереднього алгоритму прогнозу, та функцію apply(), який дозволяє застосувати на набір даних надану функцію. В даному випадку спочатку створюється масив ваг, які будемо застосовувати до елементів ковзного вікна, від 1 до 4, відповідно до заданої довжини вікна, що дорівнює

4. Потім використовуючи функцію apply(), задаємо в нього функцію, яка перемножує елементи ковзного вікна на відповідну вагу, а потім ділить на суму всіх ваг.

Код застосування зваженого ковзного середнього буде мати такий вигляд:

```
weights = numpy.arange(1, 5)
wma = dataset.rolling(window = 4).apply(lambda demand:
numpy.dot(demand, weights)/weights.sum(), raw = True)
```

де dataset – це набір даних, по якому робимо прогноз, weights – масив ваг, які ми застосовуємо до ковзного вікна, а wma – вже є результуючим прогнозом алгоритму зваженого ковзного середнього.

Графік прогнозу, а також його показники похибки, мають такий вигляд (див. рис. 3.7 і таб. 3.5):

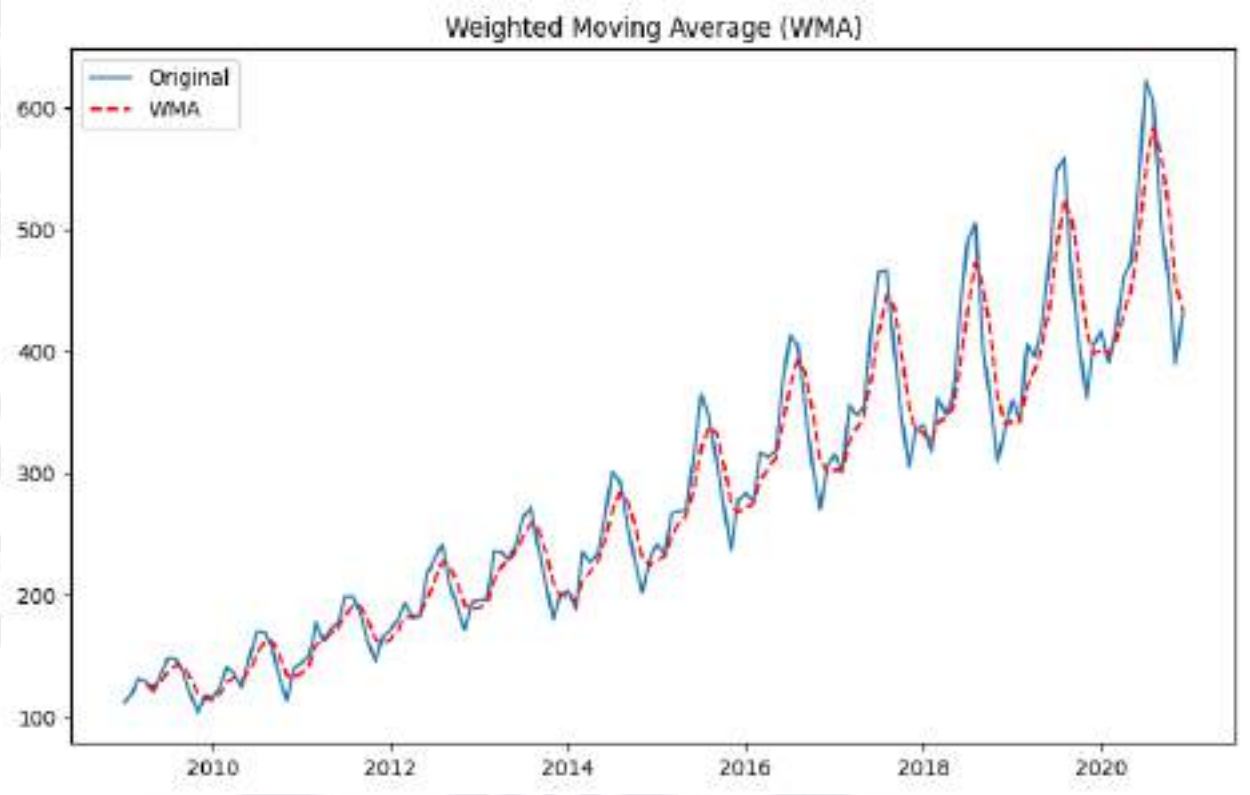

Рис. 3.7. Графік прогнозу зваженого ковзного середнього

Таблиця 3.5 – Показники рівня похибки прогнозу зваженого ковзного середнього

Метрика	Значення
Bias%	0.68%
MAE%	7.36%
MAPE	7.18%
RMSE%	8.98%
Середнє значення	6.05%

— Експоненційне згладжування або експоненційне ковзне середнє (Exponential Smoothing, Exponential Moving-Average):

Для реалізації даного алгоритму прогнозування застосовуємо вбудовані функції бібліотеки pandas, а саме функцію ewm(), який, подібно до функції rolling(), створює ковзні вікна і надає до них експоненціально зважені обчислення, та функцію mean(), який знаходить середнє арифметичне створених ковзних вікон, до яких були застосовані експоненційні ваги.

Код застосування експоненційного згладжування буде мати такий вигляд:

```
ema = dataset.ewm(span = 4, adjust = False).mean()
```

де dataset – це наш набір даних, а ema – вже є результуючим прогнозом алгоритму експоненційного ковзного середнього. Графік прогнозу, а також його показники похибки, можна побачити нижче (див. рис. 3.8 і таб. 3.6).

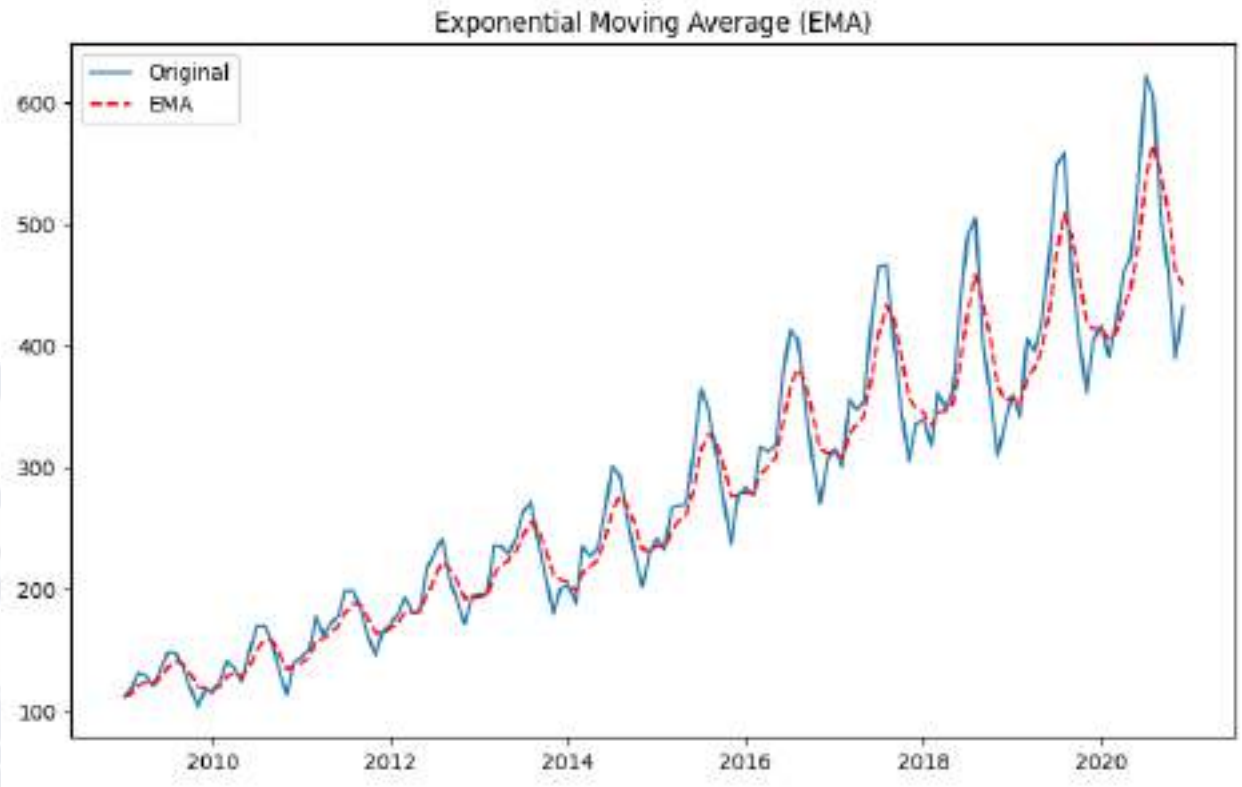

Рис. 3.8. Графік прогнозу експоненційного згладжування

Таблиця 3.6 – Показники рівня похибки прогнозу експоненційного згладжування

Метрика	**Значення**
Bias%	1.11%
MAE%	7.59%
MAPE	7.35%
RMSE%	9.23%
Середнє значення	6.32%

— Авторегресійне лінійне прогнозування (Autoregressive Linear Forecast):

Для реалізації даного алгоритму прогнозу будемо використовувати функцію AutoReg бібліотеки statsmodels. Функція AutoReg дозволяє користувачеві побудувати модель з різним порядком авторегресії, який визначає кількість попередніх значень, що будуть використовуватися для прогнозування. AutoReg приймає такі основні параметр як: сам часовий ряд

(endog), порядок моделі (lags), що вказує кількість попередніх спостережень, які будуть враховані при прогнозуванні, та додаткові параметри, наприклад, включення константи (trend), що дозволяє моделі мати сталий термін. Також можна вказати параметр сезонності та інші, що дозволяє робити прогнози з урахуванням певних циклічних властивостей часового ряду. Після створення моделі з допомогою AutoReg, її можна підігнати до даних (за допомогою функції fit()), що дозволяє оцінити параметри моделі на основі переданого часового ряду. Після цього модель надає змогу робити прогнози (через функцію predict()) на майбутні періоди, використовуючи як вхідні дані вже оцінені параметри.

Код застосування авторегресійної лінійної моделі буде мати такий вигляд:

```
model = AutoReg(train, lags = [12, 24])
model_fit = model.fit()
pred = model_fit.predict(model_fit.params,
start=int(len(dataset)*0.66), end=len(dataset))
```

де train – це частина (дві треті) нашого набору даних, на якій відбувається навчання авторегресійної лінійної моделі, model – це створювана авторегресійна лінійна модель, а pred – вже є результуючим авторегресійним лінійним прогнозом.

Графік авторегресійного лінійного прогнозу, а також його показники похибки, мають такий вигляд (див. рис. 3.9 і таб. 3.7):

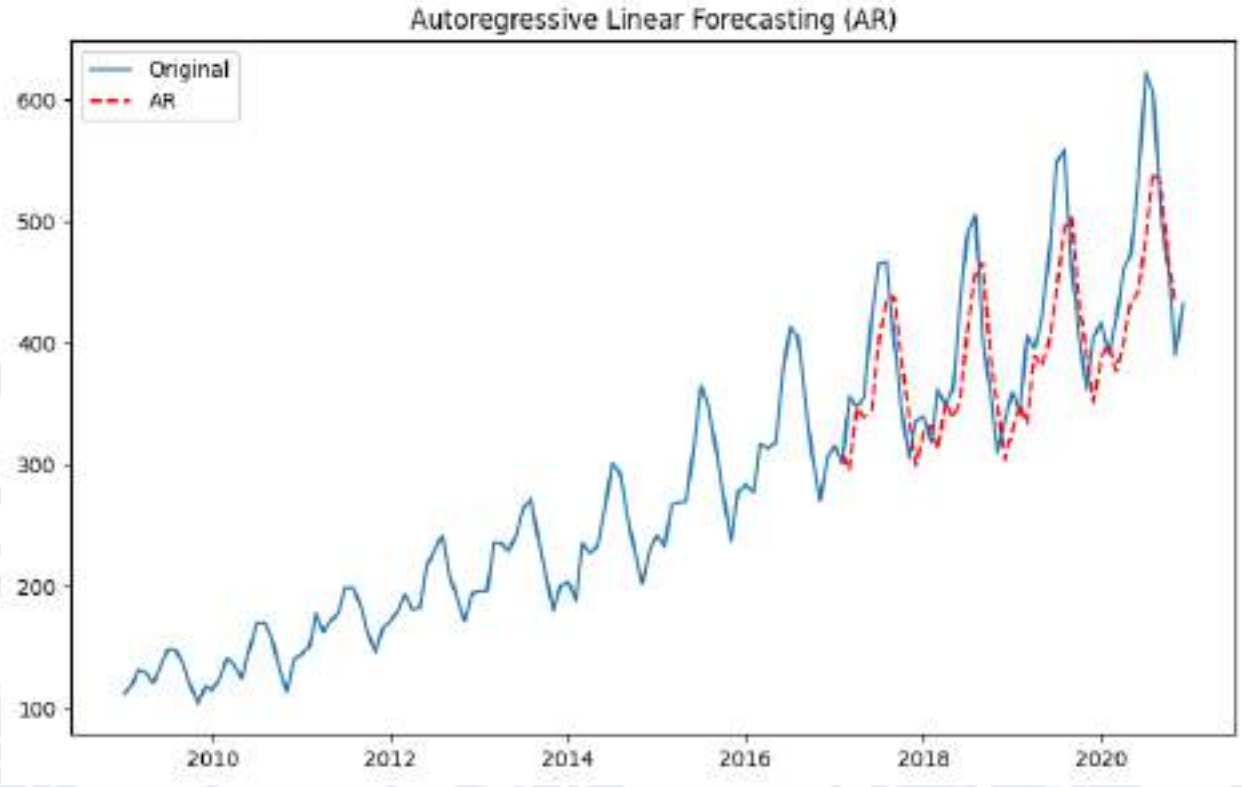

Рис. 3.9. Графік авторегресійного лінійного прогнозу

Таблиця 3.7 – Показники рівня похибки авторегресійного лінійного прогнозу

Метрика	**Значення**
Bias%	7.73%
MAE%	10.91%
MAPE	10.39%
RMSE%	13.36%
Середнє значення	10.60%

— Авторегресійне ковзне середнє (Autoregressive Moving-Average, ARMA):

Реалізовуємо модель авторегресійного ковзного середнього за допомогою функції бібліотеки statsmodels – ARMA(). Використовуючи дану функцію користувач може створити і налаштувати модель ARMA, вказавши порядок авторегресії (p) та ковзного середнього (q). Основними етапами роботи з цією

функцією є побудова моделі на основі переданих даних, оцінка параметрів моделі за допомогою функції максимальної правдоподібності, а також побудова прогнозів. Користувач може використовувати функцію fit для отримання найкращих параметрів моделі на основі історичних даних, після чого доступні функції для здійснення прогнозування та аналізу якості моделі.

Код застосування авторегресійного ковзного середнього буде мати такий вигляд:

```
model = ARMA(train, order=(2,2))
model_fit = model.fit()
forecast_steps = 48
predict = model_fit.forecast(steps=forecast_steps)
```

де train – це частина (дві треті) нашого набору даних, на якій відбувається навчання моделі ARMA, model – це створювана модель ARMA, forecast_steps – кількість кроків, на яку будується прогноз, в даному випадку це третина нашого набору даних – 48, а predict – вже є результуючим прогнозом алгоритмом ARMA.

Графік прогнозу авторегресійного ковзного середнього, а також його показники похибки, мають такий вигляд (див. рис. 3.10 і таб. 3.8):

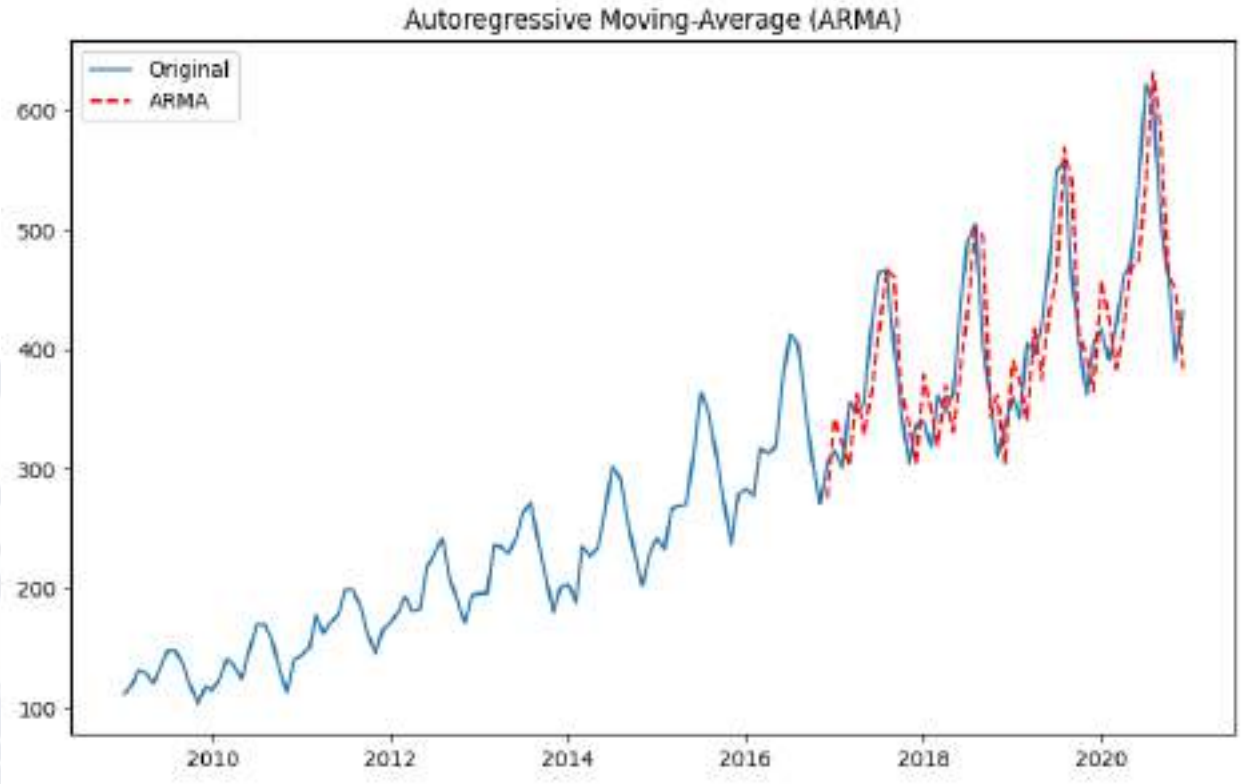

Рис. 3.10. Графік прогнозу авторегресійного ковзного середнього

Таблиця 3.8 – Показники рівня похибки прогнозу авторегресійного ковзного середнього

Метрика	**Значення**
Bias%	0.95%
MAE%	9.37%
MAPE	9.43%
RMSE%	10.81%
Середнє значення	7.64%

— Авторегресійне інтегроване ковзне середнє (Autoregressive Integrated Moving-Average, ARIMA):

Модель ARIMA, як і ARMA, також може бути реалізована за допомогою однойменної функції бібліотеки statsmodels. Відповідно у даній функції користувач задає три параметри: p — порядок авторегресії, d — ступінь різниці та q — порядок ковзного середнього. Після налаштування параметрів

модель ARIMA виконує оцінку за допомогою функції максимальної правдоподібності, після чого можна отримати прогноз, який включає як точкові прогнози, так і довірчі інтервали для майбутніх значень.

Код застосування авторегресійного інтегрованого ковзного середнього буде мати такий вигляд:

```
model = ARIMA(train, order=(2,1,2))
model_fit = model.fit()
forecast_steps = 48
predict = model_fit.forecast(steps=forecast_steps)
```

де train – це частина (дві треті) нашого набору даних, на якій відбувається навчання моделі ARIMA, model – це створювана модель ARIMA, forecast_steps – кількість кроків, на яку будується прогноз, в даному випадку це третина нашого набору даних – 48, а predict – вже є результуючим прогнозом алгоритмом ARIMA.

Графік прогнозу авторегресійного інтегрованого ковзного середнього, а також його показники похибки, мають такий вигляд (див. рис. 3.11 і таб. 3.9):

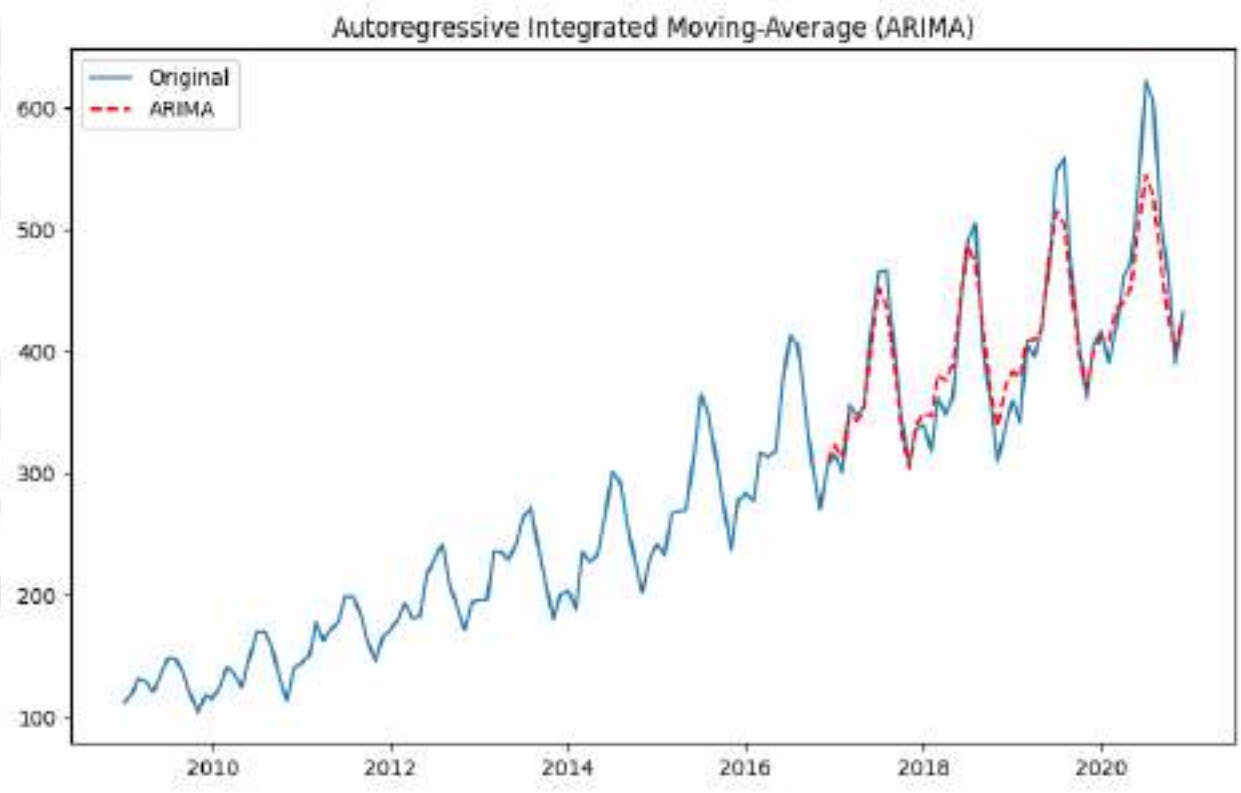

Рис. 3.11. Графік прогнозу авторегресійного інтегрованого ковзного середнього

Таблиця 3.9 – Показники рівня похибки прогнозу авторегресійного інтегрованого ковзного середнього

Метрика	Значення
Bias%	3.99%
MAE%	6.01%
MAPE	5.39%
RMSE%	8.65%
Середнє значення	6.01%

— Багатошаровий перцептрон (Multilayer Perceptron, MLP):

Для створення багатошарового перцептрона будемо використовувати клас бібліотеки sklearn – MLPRegressor. Він реалізує штучну нейронну мережу, що складається з одного або декількох шарів нейронів, кожен з яких пов'язаний з попереднім і наступним шаром, утворюючи з'єднання з певною вагою. Основними параметрами MLPRegressor є кількість прихованих шарів і кількість нейронів у кожному шарі, які визначають структуру моделі. В нашому випадку налаштовуємо один прихований шар з 50 нейронами. Також зазначаємо кількість ітерацій, за яку буде навчатися модель. Навчання для всіх нейромереж будемо проводити у кількості 250 ітерацій. Далі до створеної моделі нейромережі можна підігнати наші дані за допомогою функції fit() та отримати по ним прогноз, використовуючи функцію predict().

Код реалізації багатошарового перцептрона буде мати такий вигляд:

```
mlp = MLPRegressor(hidden_layer_sizes=(50,), max_iter=250,
random_state=1)
mlp.fit(x_train, y_train)
pred = mlp.predict(x_test)
```

Графік прогнозу багатошарового перцептрона та його показники похибки виглядають таким чином (див. рис. 3.12 і таб. 3.10):

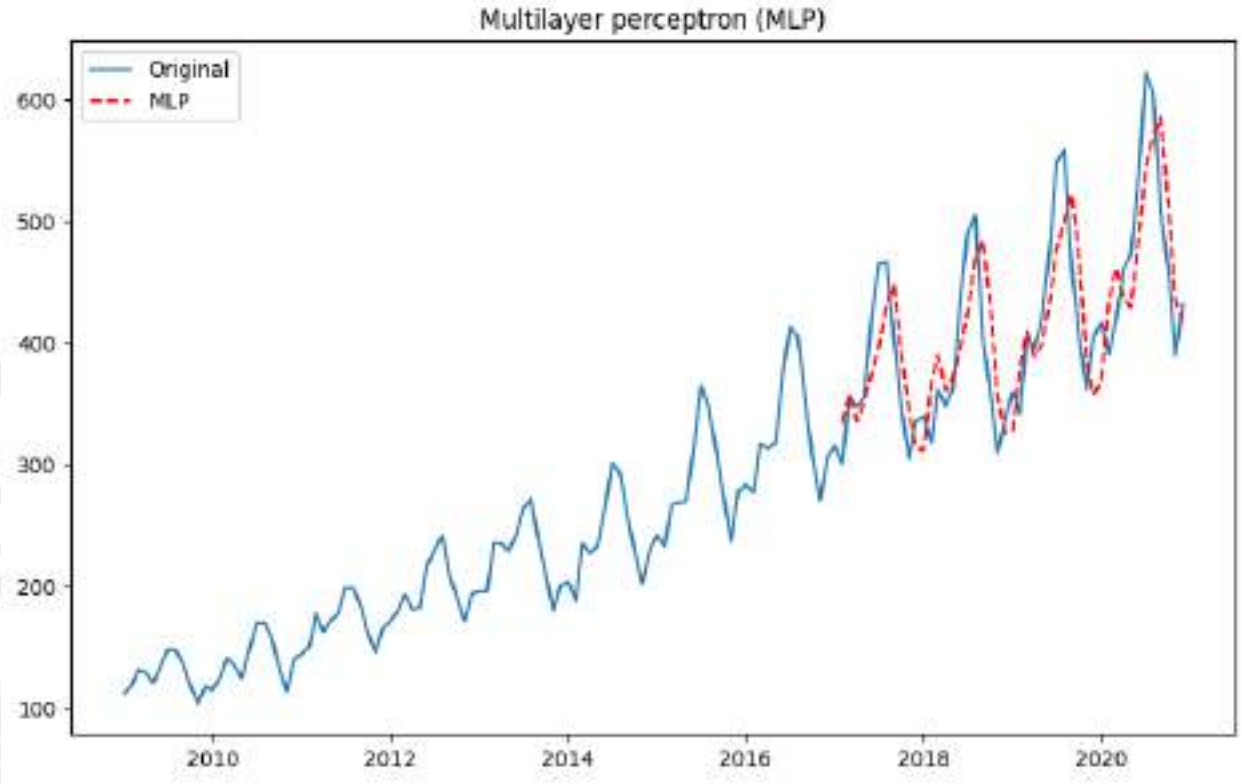

Рис. 3.12. Графік прогнозу багатошарового перцептрона

Таблиця 3.10 – Показники рівня похибки прогнозу багатошарового перцептрона

Метрика	**Значення**
Bias%	4.17%
MAE%	10.55%
MAPE	10.84%
RMSE%	12.41%
Середнє значення	9.49%

— Згорткова нейронна мережа (Convolutional Neural Network, CNN):

Для створення згорткової нейронної мережі будемо використовувати клас Sequential() бібліотеки TensorFlow, який дозволяє налаштовувати нейромережу шляхом послідовного додавання шарів до неї. Спочатку ініціалізуємо новий об’єкт – нашу модель, а потім за допомогою функції add() додаємо та налаштовуємо шари до неї відповідно до зазначених раніше шарів,

з яких повинна складатись згорткова нейронна мережа, та активаційну функцію (в даному випадку – ReLU). В кінці компілюємо дану модель, підганяємо до неї наш набір даних і робимо прогноз.

Код реалізації згорткової нейронної мережі наступний:

```
model = Sequential()
model.add(Conv1D(filters=64, kernel_size=3, activation='relu',
input_shape=(seq_length, 1)))
model.add(MaxPooling1D(pool_size=2))
model.add(Flatten())
model.add(Dense(50, activation='relu'))
model.add(Dense(1))

model.compile(optimizer='adam', loss='mse')
model.fit(x_train, y_train, epochs=250, batch_size=16,
validation_data=(x_test, y_test), verbose=2)
y_pred = model.predict(X_test)
```

Графік прогнозу згорткової нейронної мережі, а також його показники похибки, мають такий вигляд (див. рис. 3.13 і таб. 3.11):

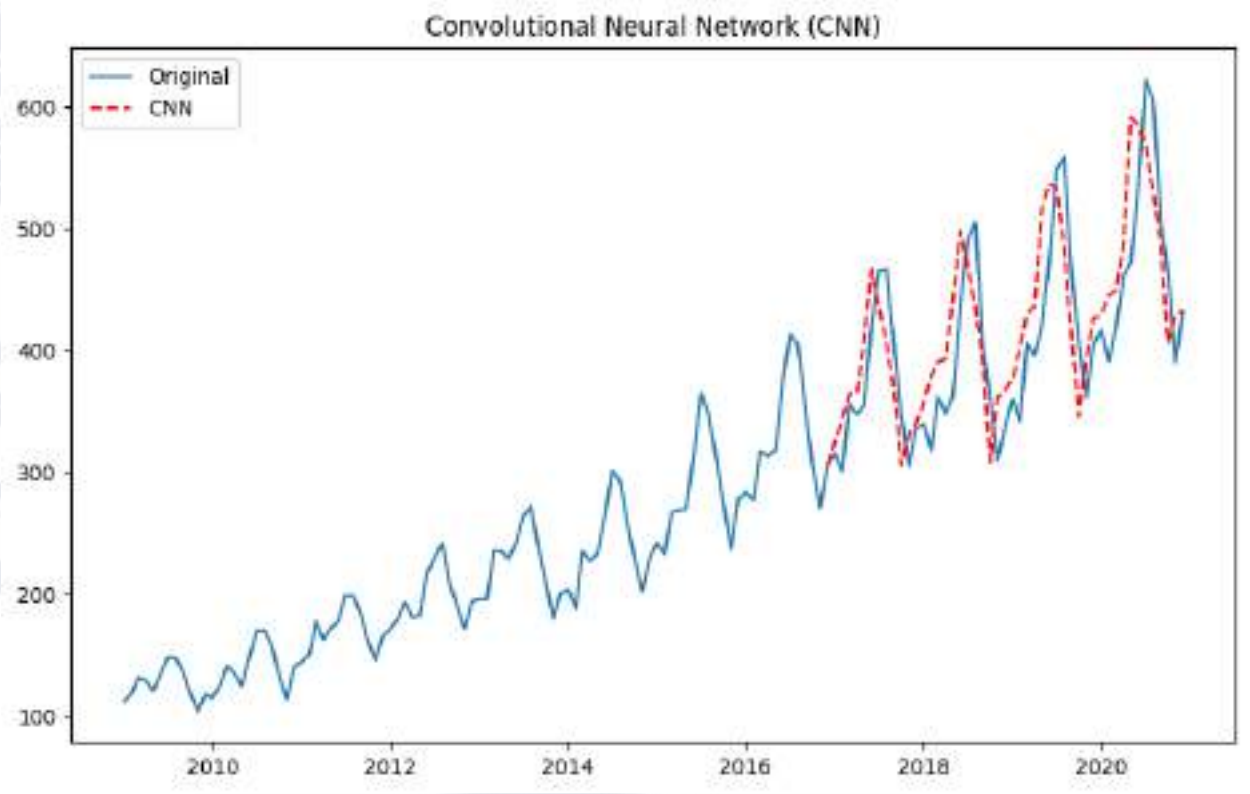

Рис. 3.13. Графік прогнозу згорткової нейронної мережі

Таблиця 3.11 – Показники рівня похибки прогнозу згорткової нейронної мережі.

Метрика	Значення
Bias%	1.83%
MAE%	6.84%
MAPE	6.67%
RMSE%	7.88%
Середнє значення	5.81%

— Рекурентна нейронна мережа (Recurrent Neural Network, RNN):

Як і для згорткової нейронної мережі, при створенні рекурентної нейронної мережі будемо використовувати клас Sequential() бібліотеки TensorFlow. Будуємо модель, додаючи до неї необхідні шари штучних нейронів, застосовуємо активаційні функції (в даному випадку це гіперболічний тангенс та сигмоїд), підганяємо до неї дані та виконуємо прогноз.

Код реалізації рекурентної нейронної мережі буде мати такий вигляд:

```
model = Sequential()
model.add(SimpleRNN(units = 50, activation = "tanh", return_sequences
= True, input_shape = (x_train.shape[1],1)))
model.add(Dropout(0.2))
model.add(SimpleRNN(units = 50, activation = "tanh", return_sequences
= True))
model.add(SimpleRNN(units = 50, activation = "tanh", return_sequences
= True))
model.add(SimpleRNN(units = 50))
model.add(Dense(units = 1,activation='sigmoid'))

model.compile(optimizer = SGD(learning_rate=0.01, decay=1e-6,
momentum=0.9, nesterov=True), loss = "mean_squared_error")
model.fit(x_train, y_train, epochs = 250, batch_size = 2)
y_pred = model.predict(x_test)
```

Графік прогнозу рекурентної нейронної мережі та його показники похибки виглядають таким чином (див. рис. 3.14 і таб. 3.12):

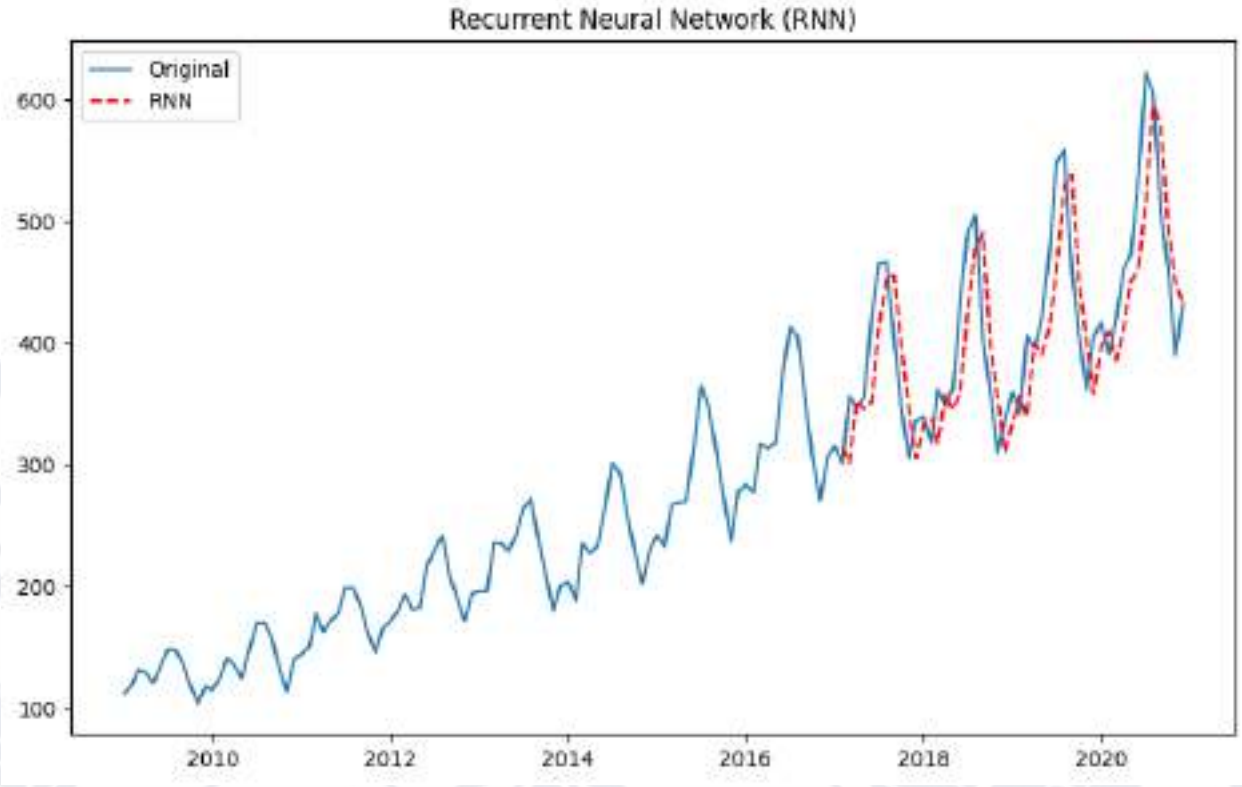

Рис. 3.14. Графік прогнозу рекурентної нейронної мережі

Таблиця 3.12 – Показники рівня похибки прогнозу рекурентної нейронної мережі

Метрика	**Значення**
Bias%	2.75%
MAE%	9.77%
MAPE	9.63%
RMSE%	11.46%
Середнє значення	8.40%

— Вентильний рекурентний вузол (Gated Recurrent Unit, GRU):

Використовуючи клас Sequential() бібліотеки TensorFlow, формулюємо наступний код для реалізації вентильного рекурентного вузла:

```
model = Sequential()
model.add(GRU(units=50, return_sequences=True,
input_shape=(x_train.shape[1],1), activation='tanh'))
model.add(Dropout(0.2))
```

```
model.add(GRU(units=50, return_sequences=True, activation='tanh'))
model.add(GRU(units=50, return_sequences=True, activation='tanh'))
model.add(GRU(units=50, activation='tanh'))
model.add(Dense(units=1, activation='relu'))

model.compile(optimizer=SGD(learning_rate=0.01, decay=1e-7,
momentum=0.9, nesterov=False), loss='mean_squared_error')
model.fit(x_train, y_train, epochs=250, batch_size=1)
y_pred = model.predict(x_test)
```

Графік прогнозу вентильного рекурентного вузла, а також його показники похибки, мають такий вигляд (див. рис. 3.15 і таб. 3.13):

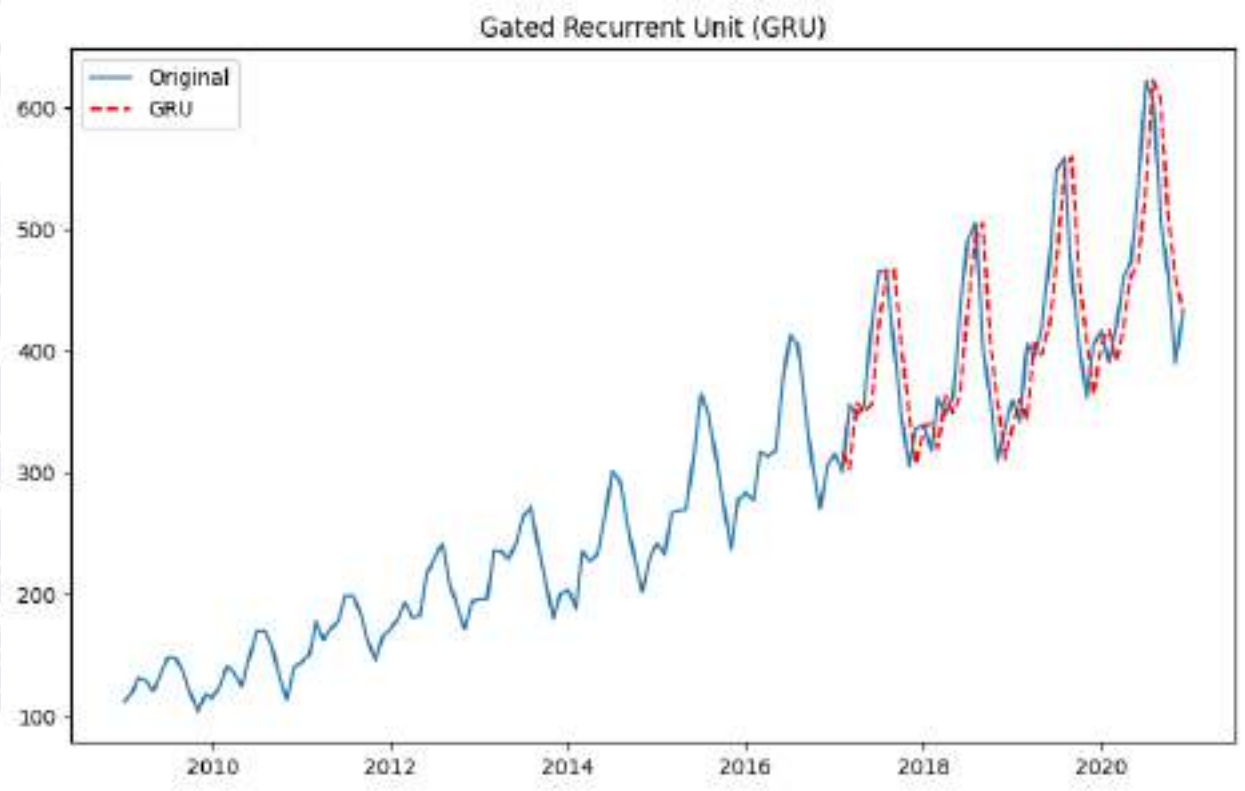

Рис. 3.15. Графік прогнозу вентильного рекурентного вузла

Таблиця 3.13 – Показники рівня похибки прогнозу вентильного рекурентного вузла

Метрика	**Значення**
Bias%	0.68%
MAE%	9.63%
MAPE	9.62%
RMSE%	11.55%
Середнє значення	7.87%

— Довга короткочасна пам’ять (Long Short-Term Memory, LSTM):

Код реалізації мережі «довга короткочасна пам’ять» буде мати такий вигляд:

```
model = Sequential()
model.add(LSTM(50, return_sequences = True, input_shape =
(x_train.shape[1],1)))
model.add(LSTM(50, return_sequences = False))
model.add(Dense(25))
model.add(Dense(1))

model.compile(optimizer = 'adam', loss = 'mean_squared_error', metrics
= ["accuracy"])
model.fit(x_train, y_train, batch_size = 1, epochs = 250, verbose=2)
y_pred = model.predict(x_test)
```

Графік прогнозу мережі «довга короткочасна пам’ять» та його показники похибки виглядають таким чином (див. рис. 3.16 і таб. 3.14):

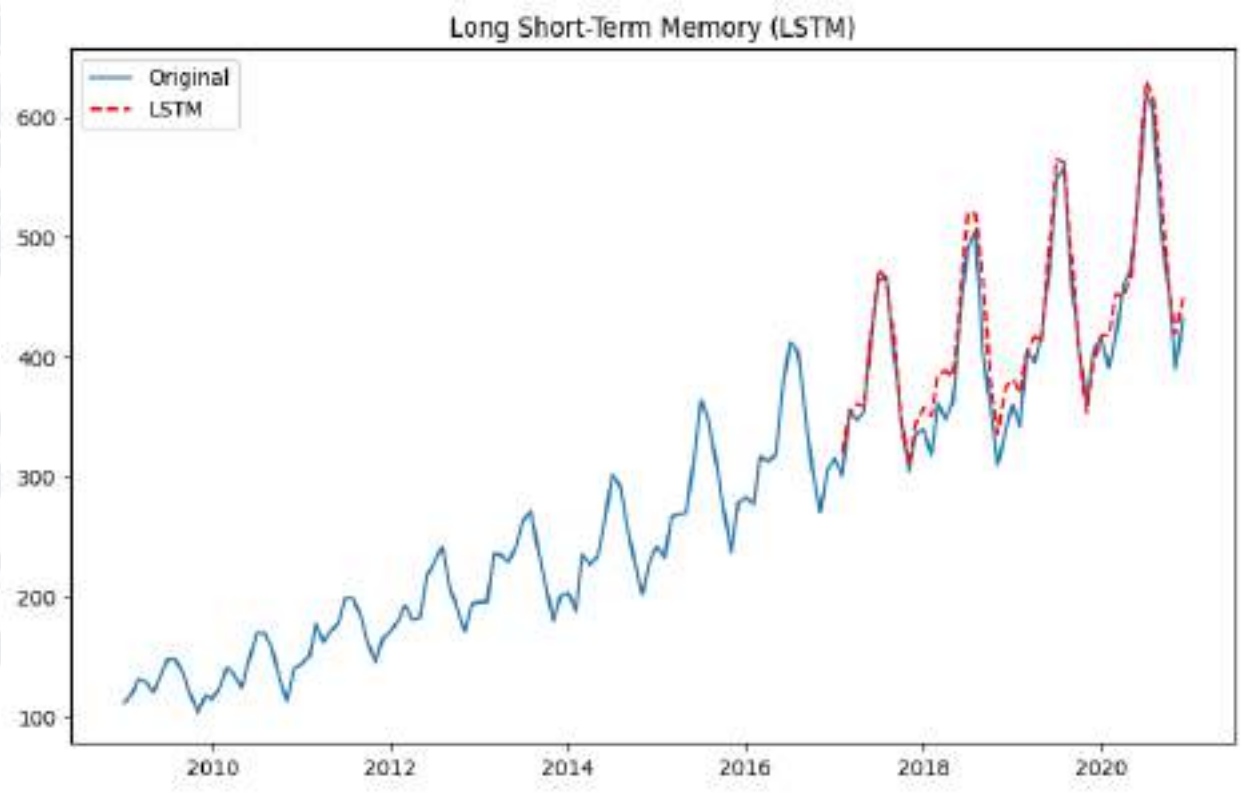

Рис. 3.16. Графік прогнозу «довгої короткочасної пам’яті»

Таблиця 3.14 – Показники рівня похибки прогнозу «довгої короткочасної пам’яті»

Метрика	Значення
Bias%	3.33%
MAE%	3.87%
MAPE	4.09%
RMSE%	4.91%
Середнє значення	4.05%

— LSTM-CNN:

Для побудови гібридної мережі LSTM-CNN можна використати клас Sequential() бібліотеки tensorflow, додаючи до створеної моделі різні шари штучних нейронів, які відповідають архітектурам мереж LSTM та CNN.

Код реалізації гібридної мережі LSTM-CNN буде мати такий вигляд:

```
model = Sequential()
model.add(tf.keras.Input(shape=(x_train.shape[1], x_train.shape[2])))
model.add(tf.keras.layers.Conv1D(filters=6, kernel_size=5,
activation='relu'))
model.add(LSTM(6, return_sequences=True, activation='relu'))
model.add(LSTM(6, return_sequences=False, activation='relu'))
model.add(Dense(future))

model.compile(loss='mean_squared_error', optimizer='adam')
model.fit(x_train, y_train, epochs=250, batch_size=16, verbose=0,
validation_data=(x_test, y_test), shuffle=True)
y_pred = model.predict(x_test)
```

Графік прогнозу гібридної мережі LSTM-CNN та його показники похибки виглядають таким чином (див. рис. 3.17 і таб. 3.15):

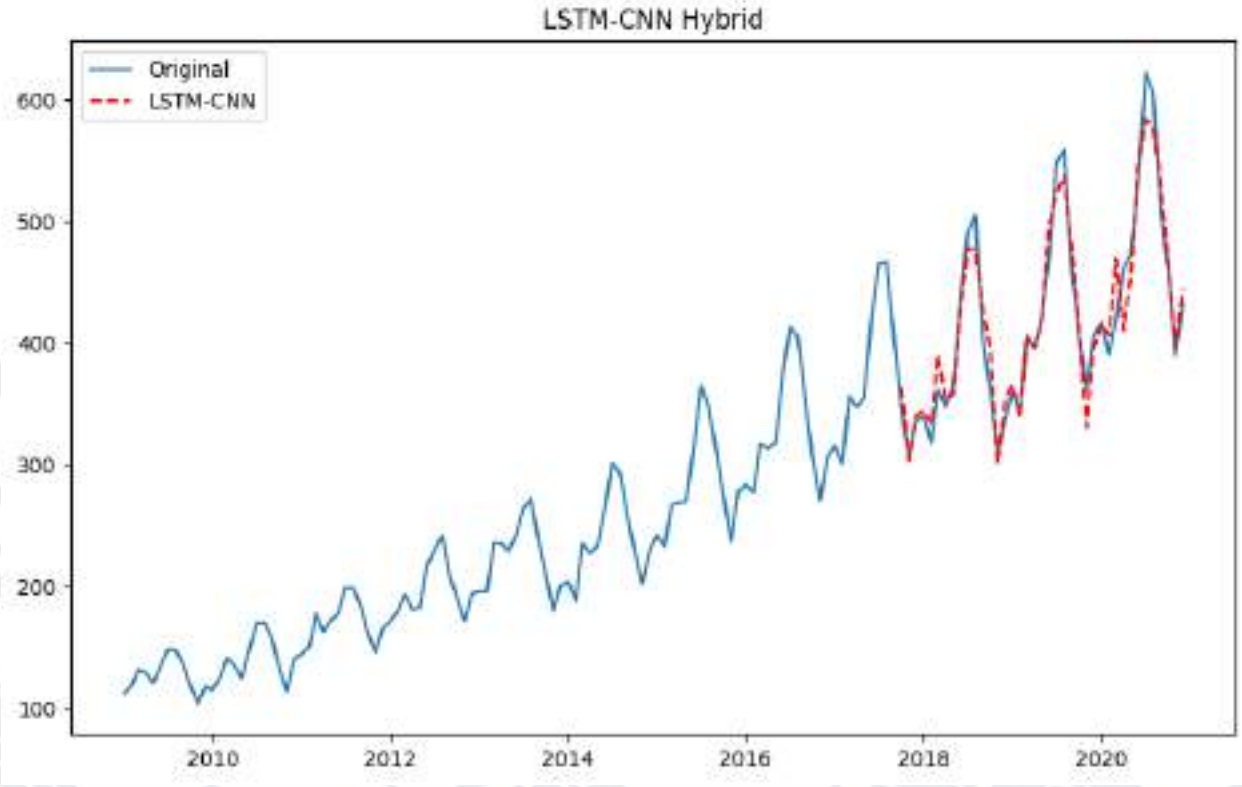

Рис. 3.17. Графік прогнозу гібридної нейромережі LSTM-CNN

Таблиця 3.15 – Показники рівня похибки прогнозу гібридної нейромережі LSTM-CNN

Метрика	**Значення**
Bias%	0.85%
MAE%	3.46%
MAPE	3.6%
RMSE%	4.43%
Середнє значення	3.09%

— ES-RNN (Exponential Smoothing – Recurrent Neural Network):

Для побудови гібридної мережі ES-RNN будемо використовувати бібліотеку ESRNN [12], яка надає доступ до однойменного класу, що дозволяє налаштувати гібридну нейронну мережу даного типу.

Код реалізації гібридної мережі ES-RNN наступний:

```
model = ESRNN(max_epochs=250, freq_of_test=5, batch_size=4,
              learning_rate=1e-4,
              per_series_lr_multip=0.8, lr_scheduler_step_size=10,
              lr_decay=0.1, gradient_clipping_threshold=50,
              rnn_weight_decay=0.0, level_variability_penalty=100,
              testing_percentile=50, training_percentile=50,
              ensemble=False, max_periods=25, seasonality=[],
              input_size=4, output_size=6,
              cell_type='LSTM', state_hsize=40,
              dilations=[[1], [6]], add_nl_layer=False,
              random_seed=1, device='cpu')

model.fit(x_train, y_train, x_test, y_test)

y_pred = model.predict(x_test)
```

Графік прогнозу гібридної мережі ES-RNN, а також його показники похибки, мають такий вигляд (див. рис. 3.18 і таб. 3.16):

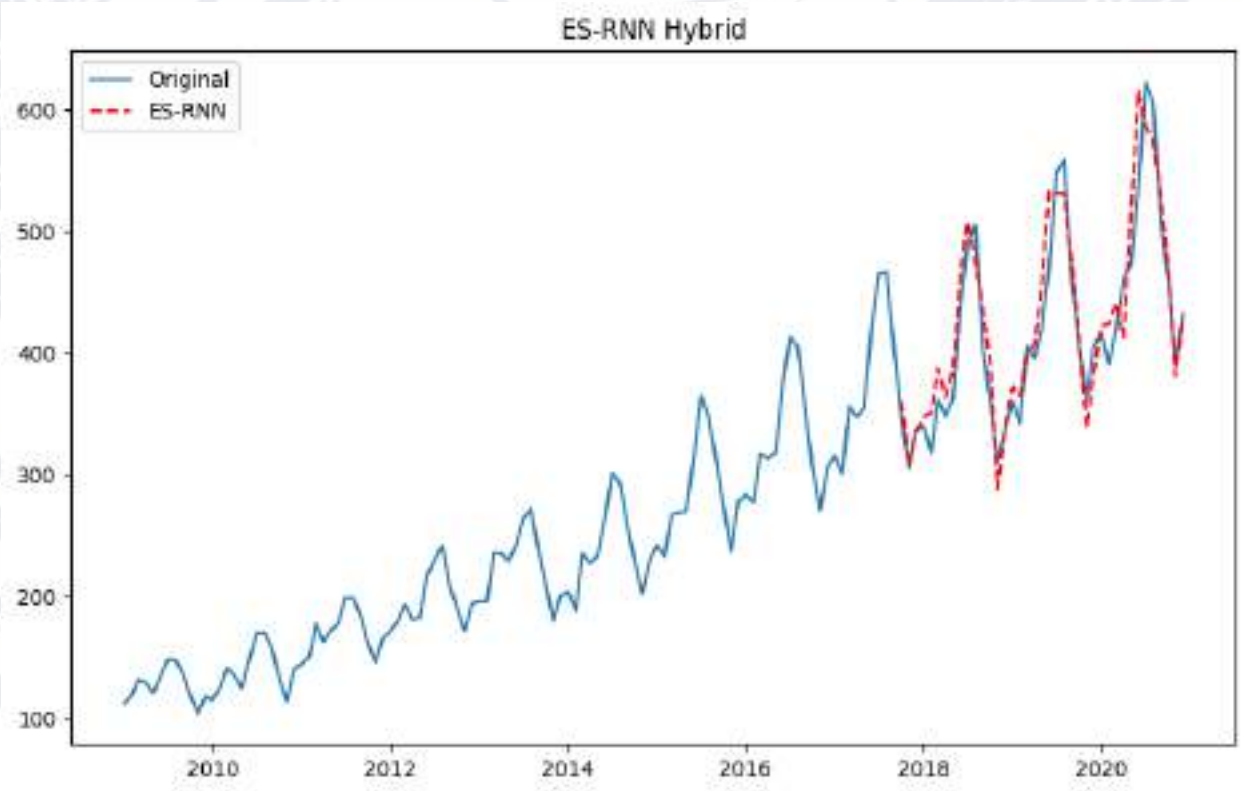

Рис. 3.18. Графік прогнозу гібридної нейромережі ES-RNN

Таблиця 3.16 – Показники рівня похибки прогнозу гібридної нейромережі ES-RNN

Метрика	Значення
Bias%	1.14%
MAE%	3.39%
MAPE	3.53%
RMSE%	4.45%
Середнє значення	3.13%

3.3 Розробка гібридної моделі нейронної мережі

Тепер спробуємо реалізувати модель розширеної довгої короткочасної пам’яті xLSTM. Розглянемо основні частини коду, які дозволять побудувати дану нейронну мережу і допоможуть у створенні гібридної моделі ES-xLSTM.

При побудові xLSTM будемо використовувати бібліотеку torch, а саме написаний код класів для мережі xLSTM буде наслідувати класу базової нейронної мережі з бібліотеки torch, до якого будуть надбудовані свої властивості та функції.

Спочатку реалізуємо клас вузлів sLSTM – sLSTMCell. Він має в собі три функції: __init__, reset_parameters, forward.

Конструктор __init__(self, input_size, hidden_size) відповідає за ініціалізацію об'єкта класу. Він приймає в себе два значення: input_size – кількість вхідних властивостей, та hidden_size – розмір прихованої частини, тобто кількість прихованих вузлів;

```
class sLSTMCell(torch.nn.Module):

    def __init__(self, input_size, hidden_size):
        super(sLSTMCell, self).__init__()
        self.input_size = input_size
        self.hidden_size = hidden_size

        self.weight_ih = torch.nn.Parameter(torch.randn(4 * hidden_size,
input_size))
```

```
        self.weight_hh = torch.nn.Parameter(torch.randn(4 * hidden_size,
hidden_size))
        self.bias = torch.nn.Parameter(torch.randn(4 * hidden_size))

        self.reset_parameters()
```

Функція reset_parameters(self) задає ваги вузла за допомогою уніфікованої ініціалізації Ксав'є. Ініціалізації Ксав'є – задає значення вагам, взявши їх із випадкового рівномірного розподілу в певному діапазоні;

```
    def reset_parameters(self):
        torch.nn.init.xavier_uniform_(self.weight_ih)
        torch.nn.init.xavier_uniform_(self.weight_hh)
        torch.nn.init.zeros_(self.bias)
```

forward(self, input, hx) – функція, яка виконує прямий прохід через вузол sLSTM. Функція приймає в себе: input – вхідний тензор параметрів вузла, hx – стан попереднього прихованого вузла. Дана функція як результат своєї роботи повертає новий прихований вузол.

```
    def forward(self, input, hx):
        h, c = hx
        gates = torch.nn.functional.linear(input, self.weight_ih,
self.bias) + F.linear(h, self.weight_hh)

        i, f, g, o = gates.chunk(4, 1)

        i = torch.exp(i)
        f = torch.exp(f)
        g = torch.tanh(g)
        o = torch.sigmoid(o)

        c = f * c + i * g
        h = o * torch.tanh(c)

        return h, c
```

Далі задаємо клас sLSTM, який відповідає за створення шару вузлів sLSTM. При ініціалізації об’єкту даного класу створюється та об’єднує послідовність вузлів sLSTM. Він має в собі наступні функції.

Конструктор __init__(self, input_size, hidden_size, num_layers, dropout=0.0) відповідає за ініціалізацію об'єкта класу. Він приймає в себе чотири значення: input_size – кількість вхідних властивостей; hidden_size – розмір прихованої частини; num_layers – кількість шарів вузлів sLSTM; dropout – імовірність виключення між шарами, яка має значення за замовчуванням 0,0;

```
class sLSTM(torch.nn.Module):

    def __init__(self, input_size, hidden_size, num_layers, dropout=0.0):
        super(sLSTM, self).__init__()
        self.input_size = input_size
        self.hidden_size = hidden_size
        self.num_layers = num_layers
        self.dropout = dropout

        self.layers = torch.nn.ModuleList([sLSTMCell(input_size if i == 0
else hidden_size, hidden_size)
                                           for i in range(num_layers)])
        self.dropout_layer = torch.nn.Dropout(dropout)
```

forward(self, input_seq, hidden_state=None) – функція, яка виконує прямий прохід через шар вузлів sLSTM. Функція приймає в себе: input_seq – вхідна послідовність зі значень параметрів вузлів та кількості самих вузлів, hidden_state – початковий прихований стан, який за замовчуванням дорівнює None. Дана функція як результат своєї роботи повертає послідовність вузлів;

```
    def forward(self, input_seq, hidden_state=None):
        batch_size, seq_length, _ = input_seq.size()

        if hidden_state is None:
            hidden_state = self.init_hidden(batch_size)

        outputs = []
        for t in range(seq_length):
            x = input_seq[:, t, :]
            for layer_idx, layer in enumerate(self.layers):
                h, c = hidden_state[layer_idx]
                h, c = layer(x, (h, c))
                hidden_state[layer_idx] = (h, c)
                x = self.dropout_layer(h) if layer_idx < self.num_layers -
1 else h
            outputs.append(x)
```

```
        return torch.stack(outputs, dim=1), hidden_state
```

Функція init_hidden(self, batch_size) ініціалізує приховані стани та стани пам'яті вузлів. Функція приймає в себе кількість послідовностей batch_size у пакеті. А повертає список довжини num_layers, в якому кожен елемент — це пара тензорів (h, c), де h — прихований стан розмірності (batch_size, hidden_size), c — стан пам'яті розмірності (batch_size, hidden_size).

```
    def init_hidden(self, batch_size):
        return [(torch.zeros(batch_size, self.hidden_size,
device=self.layers[0].weight_ih.device),
                 torch.zeros(batch_size, self.hidden_size,
device=self.layers[0].weight_ih.device))
                for _ in range(self.num_layers)]
```

Аналогічно до sLSTMCell, задаємо клас вузлів mLSTM – mLSTMCell. Він має в собі подібні до sLSTMCell функції (__init__, reset_parameters, forward), але які пристосовані для створення матричних вузлів mLSTM:

```
class mLSTMCell(torch.nn.Module):

    def __init__(self, input_size, hidden_size):
        super(mLSTMCell, self).__init__()
        self.input_size = input_size
        self.hidden_size = hidden_size

        self.weight_ih = torch.nn.Parameter(torch.randn(3 * hidden_size,
input_size))
        self.weight_hh = torch.nn.Parameter(torch.randn(3 * hidden_size,
hidden_size))
        self.bias = torch.nn.Parameter(torch.randn(3 * hidden_size))

        self.W_q = torch.nn.Linear(input_size, hidden_size)
        self.W_k = torch.nn.Linear(input_size, hidden_size)
        self.W_v = torch.nn.Linear(input_size, hidden_size)

        self.reset_parameters()

    def reset_parameters(self):
        torch.nn.init.xavier_uniform_(self.weight_ih)
        torch.nn.init.xavier_uniform_(self.weight_hh)
        torch.nn.init.zeros_(self.bias)
        torch.nn.init.xavier_uniform_(self.W_q.weight)
```

```
        torch.nn.init.xavier_uniform_(self.W_k.weight)
        torch.nn.init.xavier_uniform_(self.W_v.weight)
        torch.nn.init.zeros_(self.W_q.bias)
        torch.nn.init.zeros_(self.W_k.bias)
        torch.nn.init.zeros_(self.W_v.bias)

    def forward(self, input, hx):
        h, C = hx
        gates = torch.nn.functional.linear(input, self.weight_ih,
self.bias) + F.linear(h, self.weight_hh)

        i, f, o = gates.chunk(3, 1)

        i = torch.exp(i)
        f = torch.exp(f)
        o = torch.sigmoid(o)

        q = self.W_q(input)
        k = self.W_k(input)
        v = self.W_v(input)

        C = f.unsqueeze(2) * C + i.unsqueeze(2) *
torch.bmm(v.unsqueeze(2), k.unsqueeze(1))
        h = o * torch.bmm(q.unsqueeze(1), C).squeeze(1)

        return h, C
```

Те ж саме з класом шарів вузлів mLSTM, який має в собі функції __init__, forward та init_hidden, аналогічні до таких в класі sLSTM, які дозволяють йому створювати послідовність вузлів mLSTM:

```
class mLSTM(torch.nn.Module):

    def __init__(self, input_size, hidden_size, num_layers, dropout=0.0):
        super(mLSTM, self).__init__()
        self.input_size = input_size
        self.hidden_size = hidden_size
        self.num_layers = num_layers
        self.dropout = dropout

        self.layers = torch.nn.ModuleList([mLSTMCell(input_size if i == 0
else hidden_size, hidden_size)
                                           for i in range(num_layers)])
        self.dropout_layer = torch.nn.Dropout(dropout)

    def forward(self, input_seq, hidden_state=None):
        batch_size, seq_length, _ = input_seq.size()
```

```
        if hidden_state is None:
            hidden_state = self.init_hidden(batch_size)

        outputs = []
        for t in range(seq_length):
            x = input_seq[:, t, :]
            for layer_idx, layer in enumerate(self.layers):
                h, C = hidden_state[layer_idx]
                h, C = layer(x, (h, C))
                hidden_state[layer_idx] = (h, C)
                x = self.dropout_layer(h) if layer_idx < self.num_layers -
1 else h
            outputs.append(x)

        return torch.stack(outputs, dim=1), hidden_state

    def init_hidden(self, batch_size):
        return [(torch.zeros(batch_size, self.hidden_size,
device=self.layers[0].weight_ih.device),
                 torch.zeros(batch_size, self.hidden_size,
self.hidden_size, device=self.layers[0].weight_ih.device))
                for _ in range(self.num_layers)]
```

Далі реалізовуємо клас вузла xLSTM – xLSTMBlock. Блок мережі xLSTM може приймати форму або вузла sLSTM, або вузла mLSTM, при цьому поєднуючи його з функціями нормалізації шару, залишкових зв'язків та додаткових лінійних проекцій. xLSTMBlock має в собі дві функції: __init__() та forward.

__init__(self, input_size, hidden_size, num_layers, dropout=0.0, lstm_type="slstm") – це конструктор для ініціалізації вузла xLSTM, який може мати у своїй основі як вузол sLSTM, так і вузол mLSTM, які оточуються шарами нормалізації, активації та проекції. Дана функція приймає в себе такі значення: input_size – кількість вхідних властивостей, hidden_size – розмір прихованого стану LSTM, num_layers – кількість шарів LSTM, dropout – імовірність виключення (за замовчуванням 0,0), lstm_type – тип вузла, який береться за основу, може приймати значення 'slstm' або 'mlstm', хоча за замовчуванням він дорівнює 'slstm';

```
class xLSTMBlock(torch.nn.Module):
```

```
    def __init__(self, input_size, hidden_size, num_layers, dropout=0.0,
lstm_type="slstm"):
        super(xLSTMBlock, self).__init__()
        self.input_size = input_size
        self.hidden_size = hidden_size
        self.num_layers = num_layers
        self.dropout = dropout
        self.lstm_type = lstm_type

        if lstm_type == "slstm":
            self.lstm = sLSTM(input_size, hidden_size, num_layers,
dropout)
        elif lstm_type == "mlstm":
            self.lstm = mLSTM(input_size, hidden_size, num_layers,
dropout)
        else:
            raise ValueError(f"Invalid LSTM type: {lstm_type}")

        self.norm = torch.nn.LayerNorm(hidden_size)
        self.activation = torch.nn.GELU()
        self.dropout_layer = torch.nn.Dropout(dropout)
        self.proj = torch.nn.Linear(hidden_size, input_size)
```

forward(self, input_seq, hidden_state=None) – функція, яка виконує прямий прохід через вузол xLSTM. Функція приймає в себе: input_seq – вхідна послідовність зі значень параметрів вузлів та кількості самих вузлів, hidden_state – початковий прихований стан, який за замовчуванням дорівнює None. Дана функція як результат своєї роботи повертає послідовність вузлів.

```
    def forward(self, input_seq, hidden_state=None):
        lstm_output, hidden_state = self.lstm(input_seq, hidden_state)
        output = self.activation(lstm_output)
        output = self.norm(output)
        output = self.proj(output)
        output = self.dropout_layer(output + input_seq)
        return output, hidden_state
```

Тепер реалізовуємо клас самої моделі xLSTM. Даний клас має в собі функції __init__() та forward().

__init__(self, vocab_size, embedding_size, hidden_size, num_layers, num_blocks, dropout=0.0, lstm_type="slstm") – конструктор для ініціалізації моделі xLSTM. Дана функція приймає в себе такі значення: vocab_size – розмір

словника мережі, embedding_size – розмір вбудованих токенів, hidden_size – розмір прихованого стану в блоках LSTM, num_layers – кількість шарів в кожному блоці LSTM, num_blocks – кількість блоків xLSTM, dropout – імовірність виключення (за замовчуванням 0,0), lstm_type – тип LSTM, який використовується, може приймати значення 'slstm' або 'mlstm', хоча за замовчуванням він дорівнює 'slstm';

```
class xLSTM(torch.nn.Module):

    def __init__(self, vocab_size, embedding_size, hidden_size,
num_layers, num_blocks,
                 dropout=0.0, lstm_type="slstm"):
        super(xLSTM, self).__init__()
        self.vocab_size = vocab_size
        self.embedding_size = embedding_size
        self.hidden_size = hidden_size
        self.num_layers = num_layers
        self.num_blocks = num_blocks
        self.dropout = dropout
        self.lstm_type = lstm_type

        self.embedding = torch.nn.Embedding(vocab_size, embedding_size)
        self.blocks = torch.nn.ModuleList([
            xLSTMBlock(embedding_size, hidden_size, num_layers, dropout,
lstm_type)
            for _ in range(num_blocks)
        ])
        self.output_layer = torch.nn.Linear(embedding_size, vocab_size)
```

forward(self, input_seq, hidden_state=None) – функція, яка виконує прямий прохід через модель xLSTM. Функція приймає в себе: input_seq – вхідна послідовність індексів токенів, hidden_state – початковий прихований стан для кожного блоку, який за замовчуванням дорівнює None.

```
    def forward(self, input_seq, hidden_states=None):
        embedded_seq = self.embedding(input_seq)

        if hidden_states is None:
            hidden_states = [None] * self.num_blocks

        output_seq = embedded_seq
        for i, block in enumerate(self.blocks):
```

```
            output_seq, hidden_states[i] = block(output_seq,
hidden_states[i])

        output_seq = self.output_layer(output_seq)
        return output_seq, hidden_states
```

Тут визначаємо функцію, яка допоможе нам навчити створену модель нейромережі xLSTM:

```
def train_model(model, epochs=250, learning_rate=0.01):
    criterion = nn.MSELoss()
    optimizer = torch.optim.Adam(model.parameters(), lr=learning_rate)

    for epoch in tqdm(range(epochs), desc=f'Training model xLSTM'):
        model.train()
        epoch_loss = 0
        for i, (inputs, targets) in enumerate(train_loader):
            optimizer.zero_grad()
            outputs, _ = model(inputs)
            outputs = outputs[:, -1, :]
            loss = criterion(outputs, targets)
            loss.backward()
            optimizer.step()

            epoch_loss += loss.item()

            if (i + 1) % 100 == 0:
              print(f"Epoch [{epoch+1}/{epochs}], Batch
[{i+1}/{len(train_loader)}], Loss: {loss.item():.4f}")

        print(f"Epoch [{epoch+1}/{epochs}], Validation Loss:
{epoch_loss/len(train_loader):.4f}")

    return model
```

А далі визначаємо функцію для виконання прогнозу на основі вхідних даних:

```
def evaluate_model(model, data_loader):
    model.eval()
    predictions = []
    with torch.no_grad():
        for inputs, _ in data_loader:
            outputs, _ = model(inputs)
            predictions.extend(outputs[:, -1, :].numpy())
    return predictions
```

Тепер можна навчити саму модель та використати для виконання прогнозу. Графік прогнозу xLSTM та його показники похибки виглядають таким чином (див. рис. 3.19 і таб. 3.17):

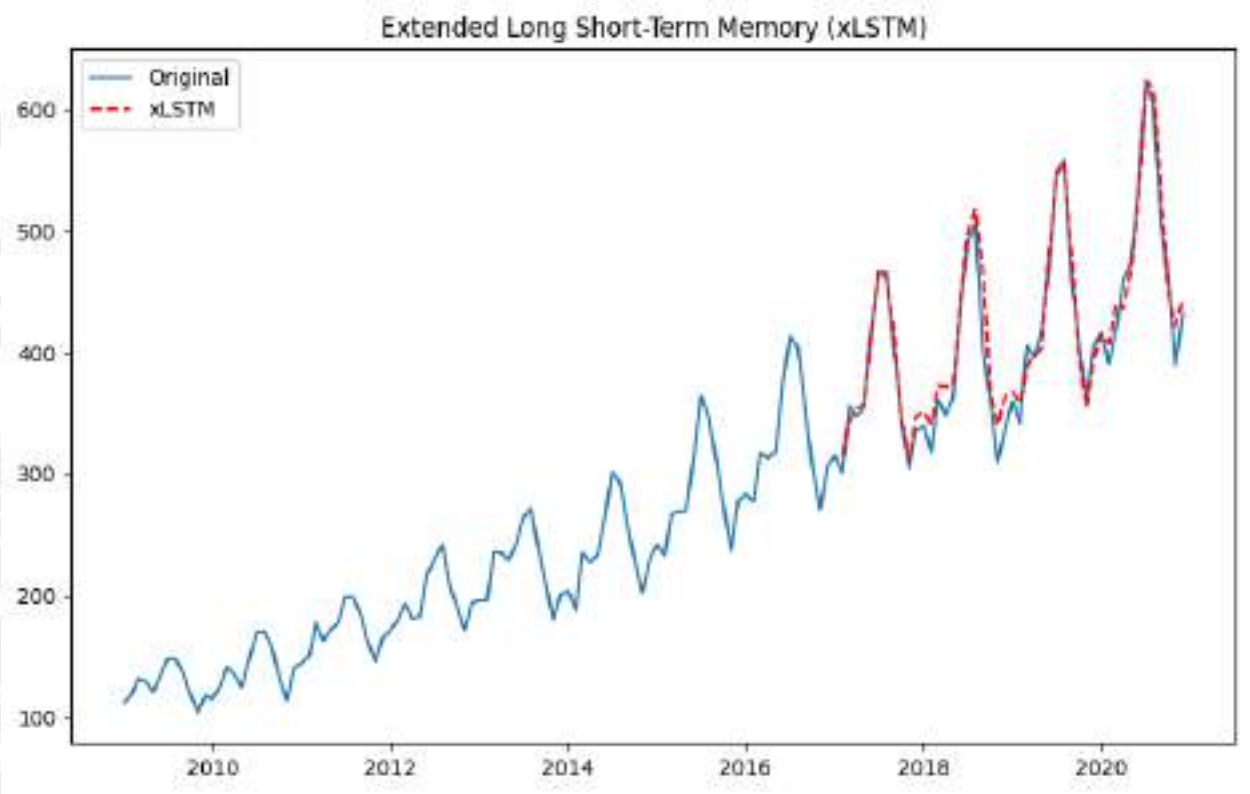

Рис. 3.19. Графік прогнозу мережі xLSTM

Таблиця 3.17 – Показники рівня похибки прогнозу xLSTM

Метрика	**Значення**
Bias%	1.03%
MAE%	3.18%
MAPE	3.32%
RMSE%	4.38%
Середнє значення	2.98%

Далі розробимо гібридну мережу ES-xLSTM (Exponential Smoothing – Extended Long Short-Term Memory), ґрунтуючись на коді вже описаної мережі

xLSTM. Як і з xLSTM, при реалізації мережі ES-xLSTM будемо використовувати бібліотеку torch.

Спочатку реалізовуємо клас _ES, який відповідає за частину експоненційного згладжування в нашій гібридній моделі. Клас _ESM має в собі наступні функції.

Конструктор __init__ ініціалізує властивості класу _ES.

```
class _ES(torch.nn.Module):
  def __init__(self, mc):
    super(_ES, self).__init__()
    self.mc = mc
    self.n_series = self.mc.n_series
    self.output_size = self.mc.output_size
    assert len(self.mc.seasonality) in [0, 1, 2]
```

Функція gaussian_noise призначений для додавання до набору даних шуму.

```
  def gaussian_noise(self, input_data, std=0.2):
    size = input_data.size()
    noise = torch.autograd.Variable(input_data.data.new(size).normal_(0,
std))
    return input_data + noise
```

Функція forward спочатку виконує розрахунок параметрів вікна. У залежності від режиму (навчання або оцінка) визначаються межі вікон для вхідних (windows_y_hat) і вихідних даних (windows_y).

```
  def forward(self, ts_object):
    input_size = self.mc.input_size
    output_size = self.mc.output_size
    exogenous_size = self.mc.exogenous_size
    noise_std = self.mc.noise_std
    seasonality = self.mc.seasonality
    batch_size = len(ts_object.idxs)

    y = ts_object.y
    idxs = ts_object.idxs
    n_series, n_time = y.shape
    if self.training:
      windows_end = n_time-input_size-output_size+1
```

```
      windows_range = range(windows_end)
    else:
      windows_start = n_time-input_size-output_size+1
      windows_end = n_time-input_size+1

      windows_range = range(windows_start, windows_end)
    n_windows = len(windows_range)
    assert n_windows>0
```

Потім функція здійснює обчислення рівнів і сезонностей. Викликається функція compute_levels_seasons(), який розраховує рівень levels для нормалізації та коефіцієнти сезонності seasonalities.

```
    levels, seasonalities = self.compute_levels_seasons(y, idxs)
    windows_y_hat = torch.zeros((n_windows, batch_size,
input_size+exogenous_size), device=self.mc.device)
    windows_y = torch.zeros((n_windows, batch_size, output_size),
device=self.mc.device)
```

Далі функція виконує нормалізацію вікон даних. Використовується функція normalize() для приведення даних до єдиного масштабу, враховуючи рівень і сезонність. У режимі навчання також розраховуються і нормалізуються вихідні значення (windows_y).

Функція здійснює додавання екзогенних змінних. Якщо розмір екзогенних даних (exogenous_size) більше 0, ці змінні додаються до нормалізованого вікна.

Також у режимі навчання функція регуляризує вхідні дані шумом: до нормалізованих даних додається шум через функцію gaussian_noise.

```
    for i, window in enumerate(windows_range):
      y_hat_start = window
      y_hat_end = input_size + window

      window_y_hat = self.normalize(y=y[:, y_hat_start:y_hat_end],
                                    level=levels[:, [y_hat_end-1]],
                                    seasonalities=seasonalities,
                                    start=y_hat_start, end=y_hat_end)

      if self.training:
        window_y_hat = self.gaussian_noise(window_y_hat, std=noise_std)
```

```
      if exogenous_size>0:
        window_y_hat = torch.cat((window_y_hat, ts_object.categories), 1)

      windows_y_hat[i, :, :] += window_y_hat

      if self.training:
        y_start = y_hat_end
        y_end = y_start+output_size
        window_y = self.normalize(y=y[:, y_start:y_end],
                                  level=levels[:, [y_start]],
                                  seasonalities=seasonalities,
                                  start=y_start, end=y_end)
        windows_y[i, :, :] += window_y

    return windows_y_hat, windows_y, levels, seasonalities
```

Далі описуємо клас _ESM, який базується на _ES і який реалізовує покращену версію експоненційного згладжування – модель Холт-Уінтерса. Клас _ESM має в собі такі функції: __init__(), compute_levels_seasons(), normalize(), predict().

Конструктор __init__ ініціалізує наступні властивості класу. mc – об’єкт конфігурації або мета-параметри моделі. Очікується, що mc.seasonality – це список, який вказує кількість сезонних компонентів. embeds – вектор ваг (torch.nn.Embedding), який містить початкові значення параметрів для рівнів і сезонних компонентів. seasonality – постійний буфер для збереження інформації про сезонність.

```
class _ESM(_ES):
  def __init__(self, mc):
    super(_ESM, self).__init__(mc)
    embeds_size = 1 + len(self.mc.seasonality) + sum(self.mc.seasonality)
    init_embeds = torch.ones((self.n_series, embeds_size)) * 0.5
    self.embeds = torch.nn.Embedding(self.n_series, embeds_size)
    self.embeds.weight.data.copy_(init_embeds)
    self.register_buffer('seasonality',
torch.LongTensor(self.mc.seasonality))
```

Функція compute_levels_seasons обчислює рівні та сезонності на основі вхідних значень y і індексів idxs.

Спочатку функція витягує ваги для відповідних індексів із self.embeds; визначає коефіцієнт для оновлення рівня (lev_sms) через torch.sigmoid, а також ініціалізує множники для сезонності, якщо такі є:

```
  def compute_levels_seasons(self, y, idxs):
    embeds = self.embeds(idxs)
    lev_sms = torch.sigmoid(embeds[:, 0])

    seas_prod = torch.ones(len(y[:,0])).to(y.device)
    seasonalities1 = []
    seasonalities2 = []
    seas_sms1 = torch.ones(1).to(y.device)
    seas_sms2 = torch.ones(1).to(y.device)
```

Далі виконується обчислення сезонностей. Якщо є одна сезонність, функція ініціалізує сезонні множники (init_seas1) із вектору ваг та додає ці множники в список seasonalities1. Якщо є дві сезонності, функція аналогічно обчислює init_seas2 для другої сезонності.

```
    if len(self.seasonality)>0:
      seas_sms1 = torch.sigmoid(embeds[:, 1])
      init_seas1 = torch.exp(embeds[:,
2:(2+self.seasonality[0])]).unbind(1)
      assert len(init_seas1) == self.seasonality[0]

      for i in range(len(init_seas1)):
        seasonalities1 += [init_seas1[i]]
      seasonalities1 += [init_seas1[0]]
      seas_prod = seas_prod * init_seas1[0]

    if len(self.seasonality)==2:
      seas_sms2 = torch.sigmoid(embeds[:, 2+self.seasonality[0]])
      init_seas2 = torch.exp(embeds[:, 3+self.seasonality[0]:]).unbind(1)
      assert len(init_seas2) == self.seasonality[1]

      for i in range(len(init_seas2)):
        seasonalities2 += [init_seas2[i]]
      seasonalities2 += [init_seas2[0]]
      seas_prod = seas_prod * init_seas2[0]

    levels = []
    levels += [y[:,0]/seas_prod]

    ys = y.unbind(1)
    n_time = len(ys)
```

```
    for t in range(1, n_time):

      seas_prod_t = torch.ones(len(y[:,t])).to(y.device)
      if len(self.seasonality)>0:
        seas_prod_t = seas_prod_t * seasonalities1[t]
      if len(self.seasonality)==2:
        seas_prod_t = seas_prod_t * seasonalities2[t]

      newlev = lev_sms * (ys[t] / seas_prod_t) + (1-lev_sms) * levels[t-1]
      levels += [newlev]
```

Потім функція здійснює оновлення рівнів і сезонностей. Ітеративно для кожного моменту часу t функція оновлює рівень, а також оновлює сезонності залежно від кількості сезонних компонентів (1 або 2), використовуючи схожі формули.

В кінці функція повертає матриці рівнів (levels) і сезонностей (seasonalities), кожна з яких містить відповідні значення для всіх моментів часу.

```
      if len(self.seasonality)==1:
        newseason1 = seas_sms1 * (ys[t] / newlev) + (1-seas_sms1) *
seasonalities1[t]
        seasonalities1 += [newseason1]

      if len(self.seasonality)==2:
        newseason1 = seas_sms1 * (ys[t] / (newlev * seasonalities2[t])) +
(1-seas_sms1) * seasonalities1[t]
        seasonalities1 += [newseason1]
        newseason2 = seas_sms2 * (ys[t] / (newlev * seasonalities1[t])) +
(1-seas_sms2) * seasonalities2[t]
        seasonalities2 += [newseason2]

    levels = torch.stack(levels).transpose(1,0)

    seasonalities = []

    if len(self.seasonality)>0:
      seasonalities += [torch.stack(seasonalities1).transpose(1,0)]

    if len(self.seasonality)==2:
      seasonalities += [torch.stack(seasonalities2).transpose(1,0)]

    return levels, seasonalities
```

Функція normalize, який виконує нормалізацію вхідних даних y, враховуючи тренд level і сезонні компоненти.

```
  def normalize(self, y, level, seasonalities, start, end):
    y_n = y / level
    for s in range(len(self.seasonality)):
      y_n /= seasonalities[s][:, start:end]
    y_n = torch.log(y_n)
    return y_n
```

Функція predict, який виконує передбачення значень часових рядів, використовуючи тренд, рівні levels і сезонності. Дана функція перетворює логарифмічний тренд у експоненційний масштаб. Потім, якщо розмір виходу output_size перевищує період сезонності, створює додаткові сезонні компоненти шляхом повторення останнього періоду. Далі обчислює передбачення y_hat через послідовне множення на сезонні компоненти для відповідного діапазону часу. Функція повертає вихідний тензор y_hat, що містить передбачені значення.

```
  def predict(self, trend, levels, seasonalities):
    output_size = self.mc.output_size
    seasonality = self.mc.seasonality
    n_time = levels.shape[1]

    trend = torch.exp(trend)

    for s in range(len(seasonality)):
      if output_size > seasonality[s]:
        repetitions = int(np.ceil(output_size/seasonality[s]))-1
        last_season = seasonalities[s][:, -seasonality[s]:]
        extra_seasonality = last_season.repeat((1, repetitions))
        seasonalities[s] = torch.cat((seasonalities[s],
extra_seasonality), 1)

    y_hat = trend * levels[:,[n_time-1]]
    for s in range(len(seasonality)):
      y_hat *= seasonalities[s][:, n_time:(n_time+output_size)]

    return y_hat
```

Наступний клас _xLSTM призначений для підготовки мережі xLSTM для інтеграції з експоненційним згладжуванням. Він будує послідовність шарів xLSTM, використовуючи вже раніше визначений клас xLSTM даної нейронної мережі.

Конструктор __init__ ініціалізує параметри моделі, задаючи об'єкт конфігурації mc, що містить в собі наступні параметри: mc.input_size - розмірність вхідних даних; mc.exogenous_size - розмірність екзогенних змінних (якщо є); mc.state_hsize - розмір прихованого стану (hidden state) в xLSTM; mc.dilations - структура стеку (глибина і кількість шарів для кожної групи); mc.cell_type - тип вузлів в xLSTM (sLSTM або mLSTM); mc.add_nl_layer - чи додавати нелінійний шар після стеку; mc.output_size - розмірність виходу моделі.

```
class _xLSTM(torch.nn.Module):
  def __init__(self, mc):
    super(_xLSTM, self).__init__()
    self.mc = mc
    self.layers = len(mc.dilations)

    layers = []
    for grp_num in range(len(mc.dilations)):
      if grp_num == 0:
        input_size = mc.input_size + mc.exogenous_size
      else:
        input_size = mc.state_hsize
      layer = xLSTM(input_size,
                    mc.state_hsize,
                    num_layers=len(mc.dilations[grp_num]),
                    lstm_type=mc.cell_type)
      layers.append(layer)

    self.lstm_stack = torch.nn.Sequential(*layers)

    if self.mc.add_nl_layer:
      self.MLPW  = torch.nn.Linear(mc.state_hsize, mc.state_hsize)

    self.adapterW  = torch.nn.Linear(mc.state_hsize, mc.output_size)
```

Функція forward виконує прямий прохід через модель.

```
  def forward(self, input_data):
    for layer_num in range(len(self.lstm_stack)):
```

```
      residual = input_data
      output, _ = self.lstm_stack[layer_num](input_data)
      if layer_num > 0:
        output += residual
      input_data = output

    if self.mc.add_nl_layer:
      input_data = self.MLPW(input_data)
      input_data = torch.tanh(input_data)

    input_data = self.adapterW(input_data)
    return input_data
```

Клас _ESxLSTM призначений для інтеграції двох частин гібридної моделі ES-xLSTM: експоненційного згладжування та розширеної довгої короткочасної пам'яті. Для цього він використовує попередньо визначені класи _ESM та _xLSTM. Клас _ESxLSTM має наступні функції:

Конструктор __init__, що ініціалізує об’єкт _ESM, який виконує експоненційне згладжування, та об'єкт _xLSTM, який є моделлю мережі xLSTM.

Функція forward, що реалізує прямий прохід через модель.

Функція predict, призначений для прогнозування.

```
class _ESxLSTM(torch.nn.Module):
  def __init__(self, mc):
    super(_ESxLSTM, self).__init__()
    self.mc = mc
    self.es = _ESM(mc).to(self.mc.device)
    self.lstm = _xLSTM(mc).to(self.mc.device)

  def forward(self, ts_object):
    windows_y_hat, windows_y, levels, seasonalities = self.es(ts_object)

    windows_y_hat = self.lstm(windows_y_hat)

    return windows_y, windows_y_hat, levels

  def predict(self, ts_object):
    windows_y_hat, _, levels, seasonalities = self.es(ts_object)

    windows_y_hat = self.lstm(windows_y_hat)
    trend = windows_y_hat[-1,:,:]

    y_hat = self.es.predict(trend, levels, seasonalities)
```

```
    return y_hat
```

Тепер, маючи код реалізації нейромережі xLSTM та код інтеграції даної мережі з експоненційним згладжуванням, можемо навчити модель мережі ES-xLSTM та застосувати її для прогнозу часового ряду.

Графік прогнозу гібридної мережі ES-xLSTM, а також його показники похибки, мають такий вигляд (див. рис. 3.20 і таб. 3.18):

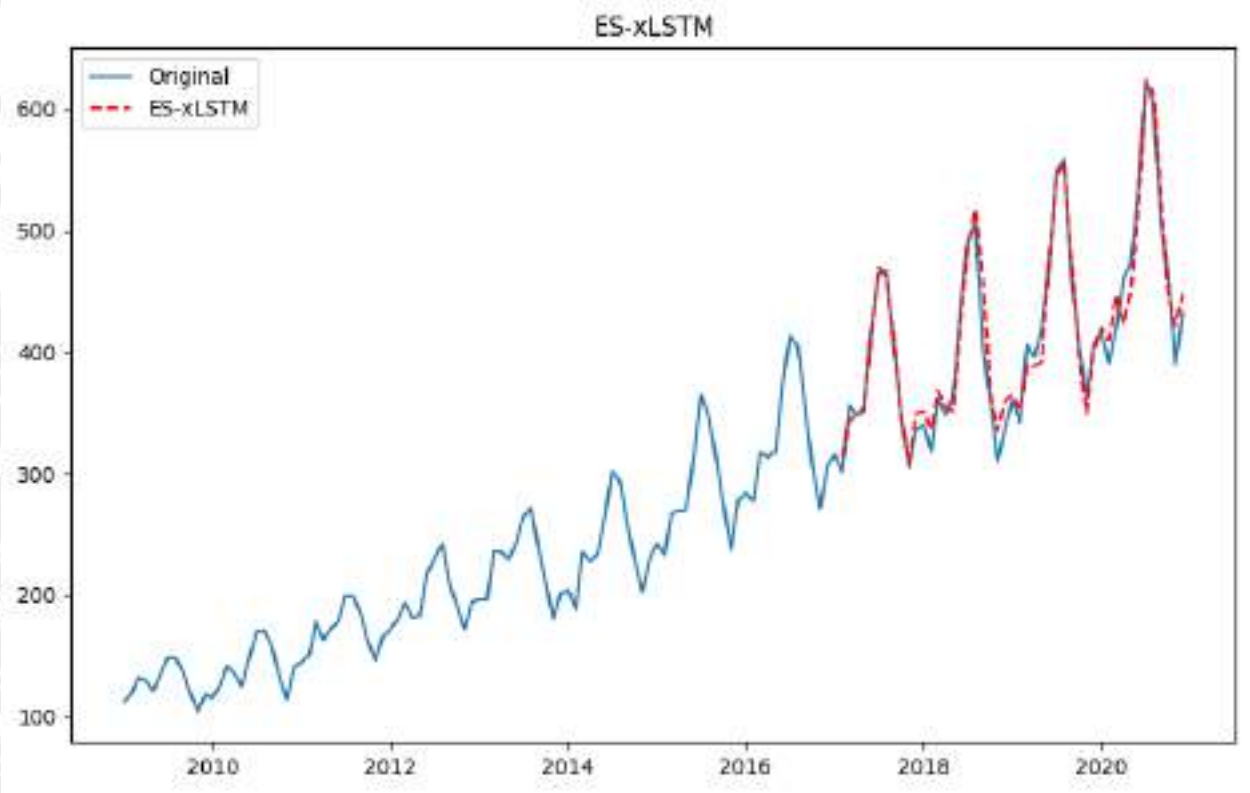

Рис. 3.20. Графік прогнозу гібридної нейромережі ES-xLSTM

Таблиця 3.18 – Показники рівня похибки прогнозу гібридної нейромережі ES-RNN

Метрика	**Значення**
Bias%	0.65%
MAE%	3.08%
MAPE	3.19%
RMSE%	4.05%
Середнє значення	2.74%

Далі спробуємо порівняти точність прогнозу розробленої моделі ES-xLSTM з іншими попередньо визначиними алгоритмами прогнозування. Відобразимо графіки порівняння прогнозу нашої розробленої моделі з трьома іншими найбільш ефективними мережами з попередньо розглянутих: ES-RNN, LSTM-CNN, xLSTM та ES-xLSTM.

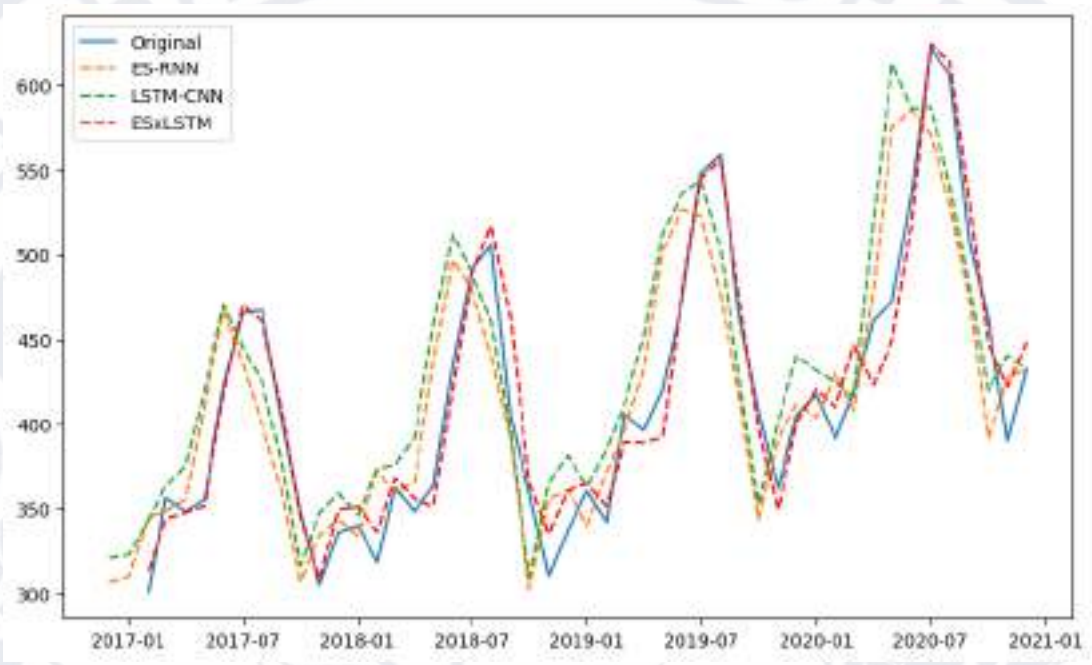

Рис. 3.21. Графік порівняння прогнозу моделей ES-RNN, LSTM-CNN та ES-xLSTM

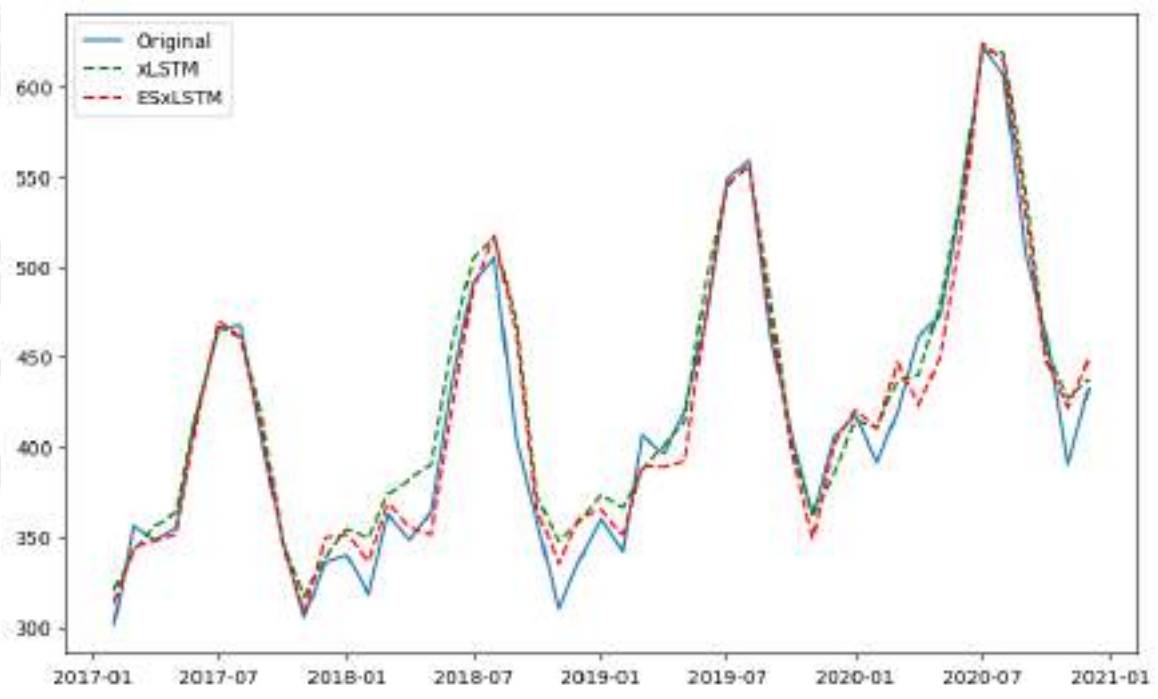

Рис. 3.22. Графік порівняння прогнозу моделей xLSTM та ES-xLSTM

Підсумовуючи дані щодо ефективності різних алгоритмів машинного навчання для прогнозування попиту, отримуємо наступний результат (див. таб. 3.19):

Таблиця 3.19 – Показники рівнів похибки всіх зазначених раніше алгоритмів прогнозу

Алгоритми прогнозу	Метрики				
	Bias%	MAE%	MAPE	RMSE%	Середнє значення
Simple Naïve Method	0.7%	9.18%	9.01%	12%	7.72%
Seasonal Naïve Method	10.8%	10.88%	11.25%	12.34%	11.32%
Simple Moving-Average	1.04%	10.36%	10.12%	12.68%	8.55%
Weighted Moving Average	0.68%	7.36%	7.18%	8.98%	6.05%
Exponential Smoothing	1.11%	7.59%	7.35%	9.23%	6.32%
Autoregressive Linear Model	7.73%	10.91%	10.39%	13.36%	10.60%
Autoregressive Moving-Average Model	0.95%	9.37%	9.43%	10.81%	7.64%
Autoregressive Integrated Moving-Average Model	3.99%	6.01%	5.39%	8.65%	6.01%
Multilayer Perceptron	4.17%	10.55%	10.84%	12.41%	9.49%
Convolutional Neural Network	1.83%	6.84%	6.67%	7.88%	5.81%
Recurrent Neural Network	2.75%	9.77%	9.63%	11.46%	8.40%
Gated Recurrent Unit	0.68%	9.63%	9.62%	11.55%	7.87%
Long Short-Term Memory	3.33%	3.87%	4.09%	4.91%	4.05%
LSTM-CNN	0.85%	3.46%	3.6%	4.43%	3.09%
ES-RNN	1.14%	3.39%	3.53%	4.45%	3.13%
xLSTM	1.03%	3.18%	3.32%	4.38%	2.98%
ES-xLSTM	0.65%	3.08%	3.19%	4.05%	2.74%

Аналізуючи отримані дані приходимо до наступних висновків. З поміж традиційних алгоритмів машинного навчання найбільшу точність має авторегресійне інтегроване ковзне середнє як самий комплексний серед них. Деякі прості алгоритми машинного навчання, такі к зважене ковзне середнє та експоненційне згладжування, показують результати кращі за деякі більш складніші алгоритми та нейронні мережі, але потрібно розуміти, що вони мають таку велику ефективність лише на короткому строці. При довгостроковому прогнозі їх точність значно спадає і їх використання стає більш проблематичним. З поміж нейронних мереж мають доволі високу точність мережі CNN та LSTM, проте найбільшу ефективність зі всіх алгоритмів мають чотири самі комплексні моделі, включаючи і нашу розробку: ES-RNN LSTM-CNN, xLSTM, ES-xLSTM. Розроблена модель хоч і не сильно, але все ж таки має показники вище ніж у оригінальної архітектури xLSTM. Але і ці показники можуть бути перевершені все новішими більш комплексними архітектурами нейронних мереж або іншими варіантами реалізації попередніх моделей.

Висновки до розділу 3

У розділі розроблюється модель нейронної мережі ES-xLSTM, код якої базується на коді мережі xLSTM, що ми адаптували для рішення задачі прогнозування часових рядів та до якої інтегрували алгоритм експоненційного згладжування. При розробці використовувалась бібліотека мови Python – torch.

Також було реалізовано інші алгоритми-попередники для прогнозування часових рядів, щоб порівняти точність прогнозу розробленої моделі з ними.

ВИСНОВКИ

В результаті проведеного дослідження була розроблена гібридна модель нейронної мережі ES-xLSTM для більш точного вирішення задачі прогнозування часових рядів. За результатами дослідження можна зробити такі висновки:

Розглянуто проблематику прогнозування попиту для сфери електронної комерції, а також висвітлені різноманітні типи та методи прогнозу попиту для різних моделей поведінки попиту.

Досліджено розвиток алгоритмів машинного навчання, використовуваних для прогнозування часових рядів, включаючи і споживчого попиту.

Спроектовано гібридну модель ES-xLSTM шляхом інтеграції передової архітектури нейромережі – розширеної довгої короткочасної пам’яті, зі статистичним алгоритмом машинного навчання – експоненційним згладжуванням, яка повинна покращити обчислення даних у роботі з мережею xLSTM.

Підібравши належні інструменти, реалізовано спроектовану модель для виконання прогнозів та навчено на тестовому наборі даних.

Також реалізовано перелік алгоритмів прогнозування часових рядів, включаючи як класичні статистичні моделі, так і засоби засновані на нейронних мережах, на тестовому наборі даних для порівняння точності розробленої моделі.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Abbate R. та ін. Demand forecasting for delivery platforms by using neural network. IFAC-PapersOnLine. 2022. Т. 55. №. 10. С. 607–612.

2. Aguiar-Pérez J. M., Pérez-Juárez M. Á. An insight of deep learning based demand forecasting in smart grids. Sensors. 2023. Т. 23. №. 3. 30 с. URL: https://doi.org/10.3390/s23031467

3. Alharthi M., Mahmood A. XLSTMTime: Long-Term Time Series Forecasting with xLSTM. Ai. 2024. Т. 5, №. 3. URL: https://doi.org/10.3390/ai5030071

4. Bandara K. та ін. Sales demand forecast in e-commerce using a long short-term memory neural network methodology //Neural Information Processing: 26th International Conference, ICONIP 2019, Sydney, NSW, Australia, December 12–15, 2019, Proceedings, Part III 26. Springer International Publishing, 2019. P. 462-474.

5. Beck M. та ін. xLSTM: Extended Long Short-Term Memory. arXiv preprint arXiv:2405.04517. 2024. 21 с. URL: http://arxiv.org/abs/2405.04517

6. Boylan J. E., Syntetos A. A. Intermittent demand forecasting: context, methods and applications. John Wiley & Sons. 2021. 383 p.

7. Brownlee J. Deep learning for time series forecasting: predict the future with MLPs, CNNs and LSTMs in Python. Machine Learning Mastery, 2018.

8. Brownlee J. Introduction to time series forecasting with python: how to prepare data and develop models to predict the future. Machine Learning Mastery, 2017. 365 p.

9. Charu C A. Neural networks and deep learning: a textbook. Springer. 2018. 506 p.

10. Chase C. W. Demand-driven forecasting: a structured approach to forecasting. John Wiley & Sons. 2013. 384 p.

11. Chawla A. та ін. Demand forecasting using artificial neural networks—a case study of American retail corporation. Applications of Artificial Intelligence Techniques in Engineering: SIGMA 2018, T. 2. Springer Singapore, 2019. 79-89 p.

12. GitHub – kdgutier/esrnn_torch. ES-RNN hybrid neural network implementation. GitHub. URL: https://github.com/kdgutier/esrnn_torch

13. Gridin I. Time series forecasting using deep learning: combining PyTorch, RNN, TCN, and deep neural network models to provide production-ready prediction solutions. BPB Publications, 2021.

14. Haben S., Voss M., Holderbaum W. Core Concepts and Methods in Load Forecasting: With Applications in Distribution Networks. Springer Nature, 2023. 331 p.

15. Hamoudia M., Makridakis S., Spiliotis E. (ed.). Forecasting with Artificial Intelligence: Theory and Applications. Springer Nature, 2023.

16. Huang C., Petukhina A. Applied time series analysis and forecasting with python. Springer International Publishing AG, 2022.

17. Huang L. та ін. Regional logistics demand forecasting: a BP neural network approach. Complex & Intelligent Systems. 2021. 1-16 p.

18. Ivan Nunes da Silva, Danilo Hernane Spatti, Rogerio Andrade Flauzino, Luisa Helena Bartocci Liboni, Silas Franco dos Reis Alves. Artificial neural networks a practical course, Springer International Publishing Switzerland 2017, Publisher Springer Cham, DOI.

19. Jafari N., Lewison M. Forecasting air passenger traffic and market share using deep neural networks with multiple inputs and outputs. Frontiers in artificial intelligence. 2024. T. 7.

20. Joseph M. Modern time series forecasting with python: explore industry-ready time series forecasting using modern machine learning and deep learning. Packt Publishing. 2022. 522 p.

21. Kilimci Z. H. та ін. An improved demand forecasting model using deep learning approach and proposed decision integration strategy for supply chain. Complexity. 2019. T. 2019. 1–15 p.

22. Kim M. та ін. A hybrid neural network model for power demand forecasting. Energies. 2019. Т. 12. №. 5. 931 p.

23. Kochak A., Sharma S. Demand forecasting using neural network for supply chain management. International journal of mechanical engineering and robotics research. 2015. Т. 4. №. 1. 96-104 p.

24. Kolassa S., Siemsen E. Demand forecasting for managers. Business Expert Press, 2016.

25. Korstanje J. Advanced forecasting with Python. United States: Apress, 2021. 243-251 p.

26. Kozodoi N. та ін. Probabilistic demand forecasting with graph neural networks. ECML PKDD 2023 ML4ITS Workshop. 2023.

27. Lazzeri F. Machine learning for time series forecasting with python. Wiley & Sons, Limited, John, 2021. 250 p.

28. Lewis C. Demand forecasting and inventory control. Taylor & Francis Group, 2012. 176 p.

29. Mahbub N., Paul S. K., Azeem A. A neural approach to product demand forecasting. International Journal of Industrial and systems engineering. 2013. Т. 15. №. 1. P. 1-18.

30. Matplotlib documentation – Matplotlib 3.9.2 documentation. Matplotlib – Visualization with Python. URL: https://matplotlib.org/stable/index.html

31. Montgomery D. C., Jennings C. L., Kulahci M. Introduction to time series analysis and forecasting. Wiley & Sons, Incorporated, John, 2015. 672 p.

32. NumPy documentation – numpy v2.1 manual. URL: https://numpy.org/doc/stable/index.html

33. Pandas documentation – pandas 2.2.3 documentation. pandas - Python Data Analysis Library. URL: https://pandas.pydata.org/docs/

34. Peixeiro M. Time series forecasting in python. Manning Publications Co. LLC, 2022.

35. PyTorch Guide – PyTorch 2.5 documentation. URL: https://pytorch.org/docs/stable/index.html

36. Rathipriya R. та ін. Demand forecasting model for time-series pharmaceutical data using shallow and deep neural network model. Neural Computing and Applications. 2023. T. 35.

37. Scikit-learn: machine learning in Python – scikit-learn 1.5.2 documentation. URL: https://scikit-learn.org/stable/

38. Shell K., Granger C. W. J., Newbold P. Forecasting economic time series. Elsevier Science & Technology Books, 2014.

39. Slimani I., El Farissi I., Achchab S. Artificial neural networks for demand forecasting: Application using Moroccan supermarket data. 2015 15th International Conference on Intelligent Systems Design and Applications (ISDA). IEEE, 2015. P. 266-271.

40. Smyl S. A hybrid method of exponential smoothing and recurrent neural networks for time series forecasting. International journal of forecasting. 2020. T. 36. №. 1. P. 75-85.

41. Smyl S., Dudek G., Pełka P. ES-dRNN: a hybrid exponential smoothing and dilated recurrent neural network model for short-term load forecasting. IEEE transactions on neural networks and learning systems. 2023. 1–13 с. URL: https://doi.org/10.1109/tnnls.2023.3259149

42. Taghizadeh E. Utilizing artificial neural networks to predict demand for weather-sensitive products at retail stores. arXiv preprint arXiv:1711.08325. 2017.

43. TensorFlow Core | Guide. TensorFlow library documentation. TensorFlow. URL: https://www.tensorflow.org/guide?hl=en

44. Thakur A., Konde A. Fundamentals of neural networks. International Journal for Research in Applied Science and Engineering Technology. 2021. T. 9. 407-26 с.

45. Thomopoulos N. T. T. Demand forecasting for inventory control. Springer, 2016. 200 p.

46. Torres J. F., Martínez-Álvarez F., Troncoso A. A deep LSTM network for the Spanish electricity consumption forecasting. Neural computing and applications. 2022.

47. User guide - statsmodels 0.14.4. Statsmodels library documentation. URL: https://www.statsmodels.org/stable/user-guide.html

48. Vandeput N. Data science for supply chain forecasting. de Gruyter GmbH, Walter, 2021. 280 p.

49. Vandeput N. Demand Forecasting Best Practices. Simon and Schuster, 2023. 218 c.

50. Vasilev I. Advanced deep learning with python: design and implement advanced next-generation AI solutions using TensorFlow and PyTorch. Packt Publishing Ltd, 2019.

51. Іваненко А. В., Січко Т. В. Методи машинного навчання в аналізі та прогнозуванні споживчого попиту. Прикладні аспекти сучасних міждисциплінарних досліджень. 2024.

52. Майборода О.Є., Майборода О.В., Палах І.С. Ефект «бичачого батога» в логістичній діяльності підприємства. Науковий журнал «Науковий погляд: економіка та управління» № 4 (74), 2021 Р. 5-12.

53. Субботін С. О. Нейронні мережі : теорія та практика: навч. посіб. Житомир : Вид. О. О. Євенок, 2020. 184 с.

54. Юрчак І.Ю. Системи з самоорганізацією та самонавчанням. Курс лекцій. / І.Ю. Юрчак – Львів : Кафедра САП, НУ "Львівська політехніка". URL: https://www.victoria.lviv.ua/library/students/sss/lecture.html

ДЕКЛАРАЦІЯ

про дотримання академічної доброчесності

Я, _____

_____

_____

_____

_____

Повністю вказується ПІБ та статус (освітня (освітньо-наукова) програма – для здобувачів вищої освіти, назва кваліфікаційної роботи)

що нижче підписався, розуміючи та підтримуючи загальновизнані засади справедливості, доброчесності та законності,

ЗОБОВ'ЯЗУЮСЬ:

дотримуватися принципів та правил академічної доброчесності, що визначені законодавством України, локальними нормативними актами Донецького національного університету імені Василя Стуса, положеннями, правилами, умовами, визначеними іншими суб'єктами, та не допускати їх порушення.

ПІДТВЕРДЖУЮ:

що мені відомі положення статті 42 Закону України «Про освіту»;

що у даній роботі не представляв чиїсь роботи повністю або частково як свої власні. Там, де я скористався працею інших, я зробив відповідні посилання на джерела інформації;

що дана робота не передавалась іншим особам і подається вперше, не порушує авторських та суміжних прав закріплених статтями 21-25 Закону України «Про авторське право та суміжні права», а дані та інформація не отримувались в недозволений спосіб.

УСВІДОМЛЮЮ:

що ця робота може бути перевірена університетом на плагіат або інші порушення академічної доброчесності, в тому числі з використанням спеціалізованих сервісів;

що у разі порушення академічної доброчесності, до мене можуть бути застосовані процедури, передбачені законодавством України та Кодексом академічної доброчесності та корпоративної етики Донецького національного університету імені Василя Стуса, іншими локальними нормативними актами університету, та я можу бути притягнутий до академічної відповідальності.

_____ _____

(дата) (підпис)