НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО»
ІНСТИТУТ ПРИКЛАДНОГО СИСТЕМНОГО АНАЛІЗУ
КАФЕДРА МАТЕМАТИЧНИХ МЕТОДІВ СИСТЕМНОГО АНАЛІЗУ

На правах рукопису
УДК 004.85

До захисту допущено
В. о. завідувача кафедри ММСА

О.Л.Тимощук

«___»_____2020р.

Магістерська дисертація
на здобуття ступеня магістра за спеціальністю 122 Комп'ютерні науки
на тему: «Прогнозування фінансових ринків методами машинного навчання»

Виконав:
студент ІІ курсу, групи КА-93 мп
Фоменко Нікіта Андріович _____

Керівник: доцент кафедри ММСА,
д.т.н. доц. Недашківська Н.І. _____

Рецензент: доцент кафедри системного проектування
КПІ ім. Ігоря Сікорського
к.т.н., Безносик О.Ю. _____

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

Київ
2020

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО»
ІНСТИТУТ ПРИКЛАДНОГО СИСТЕМНОГО АНАЛІЗУ
КАФЕДРА МАТЕМАТИЧНИХ МЕТОДІВ СИСТЕМНОГО АНАЛІЗУ

Рівень вищої освіти — другий (магістерський)
Спеціальність — 122 «Комп'ютерні науки»

ЗАТВЕРДЖУЮ
В. о. завідувача кафедри ММСА
О. Л. Тимощук
«___» _____ 2020 р.

ЗАВДАННЯ

на магістерську дисертацію студенту Фоменку Нікіті Андрійовичу

1. Тема дисертації: «Прогнозування фінансових ринків методами машинного навчання», науковий керівник дисертації Недашківська Надія Іванівна, д.т.н, затверджені наказом по університету від «02» листопада 2020р. № 3182-с.

2. Термін подання студентом дисертації: 14 грудня 2020 р.

3. Об'єкт дослідження: фінансові ринки, представлені статистичними даними стосовно їх розвитку

4. Предмет дослідження: ймовірнісно-статистичні методи та глибокі нейронні мережі для моделювання і прогнозування розвитку фінансових ринків; алгоритм підбору архітектури глибоких нейронних мереж та їх гіперпараметрів

5. Перелік завдань, які потрібно розробити:

1) дослідити сучасний стан та особливості фінансових ринків;

2) проаналізувати існуючі методи прогнозування процесів на фінансових ринках та виконати огляд методів та підходів до прогнозування;

3) дослідити методи відбору архітектур та гіперпараметрів глибоких нейронних мереж; розробити власний алгоритм вибору архітектури та гіперпараметрів моделей нейронних мереж для прогнозування фінансових часових рядів;

4) виконати побудову математичних моделей на реальних статистичних даних та отримати практичні результати, виконати аналіз результатів;

5) розробити стартап-проект виведення на ринок результатів дослідження;

6) розробити концептуальні висновки за результатами наукового дослідження;

7) оформити наукову статтю.

6. Орієнтовний перелік графічного (ілюстративного) матеріалу:

1). Інформаційні графіки (рис.);

2). Приклади результатів побудованих моделей (рис.);

3). Таблиці у розділі стартап-проекту та порівняльні таблиці результатів.

7. Дата видачі завдання: 01 вересня 2020 р.

Календарний план

№ з/п	Назва етапів виконання магістерської дисертації	Термін виконання етапів магістерської дисертації
1.	Концептуальний вступ дисертації. Формулювання об’єкта, предмета, цілі, завдань, новизни, практичної значущості результатів	06.09.2020 — 13.09.2020
2.	Огляд літературно-інформаційних джерел. Характеристика існуючих проблем	14.09.2020 — 27.09.2020
3.	Огляд статистично-ймовірністних методів прогнозування нелінійних нестаціонарних процесів та глибоких нейронних мереж у задачах фінансових ринків.	28.09.2020 — 11.10.2020
4.	Формування структури моделі прогнозування та основних її етапів. Розробка алгоритму вибору архітектури та гіперпараметрів моделей нейронних мереж.	12.10.2020 — 21.10.2020
5.	Реалізація системи та отримання практичних результатів	22.10.2020 — 06.11.2020
6.	Стартап-проект	07.11.2020 — 14.11.2020
7.	Концептуальні висновки. Перспективи розвитку отриманих рішень	15.11.2020 — 20.11.2020
8.	Оформлення наукової статті	20.11.2020 — 26.11.2020

Студент Н.А. Фоменко

Науковий керівник дисертації Н.І. Недашківська

РЕФЕРАТ

Магістерська дисертація: 139 с., 35 рис., 50 табл., 1 додаток, 14 джерел.

ЧАСОВІ РЯДИ, РЕГРЕСІЙНІ МОДЕЛІ, ГЛИБОКІ НЕЙРОННІ МЕРЕЖІ, ПРОГНОЗУВАННЯ, ФІНАНСОВІ РИНКИ, ПОШУК АРХІТЕКТУРИ, ПОШУК ГІПЕРПАРАМЕТРІВ

Об’єктом дослідження є фінансові ринки, представлені у вигляді часових рядів на основі статистичних даних стосовно їхньої динаміки.

Предметом дослідження є ймовірнісно-статистичні методи та глибокі нейронні мережі для моделювання і прогнозування фінансових ринків, а також методи пошуку найкращої архітектури глибокої нейронної мережі і відповідно її найкращих гіперпараметрів.

Метою дослідження є аналіз характеру поведінки динамічних процесів фінансових ринків на основі часових рядів та прогнозування за допомогою різних видів авторегресійних моделей та глибоких нейронних мереж, а також отримання алгоритму відбору архітектури нейронної мережі та її гіперпараметрів для задачі короткострокового прогнозування фінансових часових рядів .

Методами дослідження моделювання та прогнозування поведінки фінансових ринків є методи регресійного аналізу та алгоритми глибоких нейронних мереж.

Новизною є отриманий алгоритм пошуку архітектури нейронної мережі та її гіперпараметрів.

ABSTRACT

Master’s thesis: 139 pages, 35 images, 50 tables, 1 append., 14 sources.

The theme: «Forecasting of Financial Markets using Machine Learning Methods».

TIME SERIES, REGRESSION MODELS, DEEP NEURAL NETWORK, FORECASTING, FINANCIAL MARKETS, ARCHITECTURE SEARCHING, HYPERPARAMETERS TUNING

The main research is the financial markets presented in the publication of time series on the basis of statistical data on their dynamics.

The subject of research is probabilistic-statistical methods and deep neural networks for modeling and forecasting of financial markets, as well as methods of finding the best architecture of a deep neural network and according to its best hyperparameters.

The aim of the study is to analyze the behavior of dynamic processes of financial markets based on time series and forecasting using different types of autoregressive models and deep neural networks, as well as to obtain a neural network architecture selection algorithm and hyperparameter for short-term forecasting of financial journals.

Methods of research of modeling and forecasting of behavior of financial markets are methods of regression analysis and algorithms of deep neural networks.

The novelty is the obtained algorithm for searching the neural network architecture and its hyperparameters.

Зміст

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ І ПОЗНАЧЕНЬ

AR (АР) – авторегресія

MA – ковзне середнє

ARMA (АРКС) – авторегресія з ковзним середнім

ARIMA (АРІКС) – авторегресія з інтегрованим ковзним середнім

ARCH (АРУГ) – авторегресія з умовною гетероскедастичністю

GARCH (УАРУГ) – узагальнена авторегресія з умовною гетероскедастичністю

EGARCH (ЕУАРУГ) – експоненційна узагальнена авторегресія з умовною гетероскедастичністю

DW – критерій Дарбіна-Уотсона

ACF (АКФ) – автокореляційна функція

PACF (ЧАКФ) – часткова автокореляційна функція

RMSE – середньоквадратична похбка

MAE – середня абсолютна похибка

MAPE– середня абсолютна похибка у процентах

R^2– коефіцієнт детермінації

AIC – інформаційний критерій Акайке

BIC – інформаційний критерія Байєса-Шварца

STD – стандартне відхилення

MAD – середнє абсолютне відхилення

SGD – стохастичний градієнтний спуск

GRU – Gated Reccurent Unit

LSTM – Long-Short Term Memory

ReLU – Rectified Linear Unit

ВСТУП

Часові ряди, які відображають динаміку деякого процесу, є досить складними за своєю структурою та можуть включати тренд, сезонну складову, випадковий шум. Такі часові ряди містять характеристики, що описують зміни стану системи протягом її еволюції. Наприклад, фінансові часові ряди відображають поточний економічний стан, і тому перспективними є дослідження в цій сфері для моделювання подальшого розвитку економіки. У літературі запропоновано велику кількість різноманітних моделей для вирішення проблем прогнозування, це класичні методи математичної статистики, а також методи машинного навчання. Метою даної роботи є використання методів з прогнозування фінансових часових рядів, що використовують машинне навчання, статистичні підходи та гібридні моделі. На основі аналізу підходів, що використовуються в різних публікаціях, можна їх класифікувати за двома категоріями. До першої категорії належать роботи, у яких прогнозування здійснюється на основі попередніх значень часових рядів. У публікаціях цього класу використовуються класичні статистичні підходи, такі як ARIMA, GARCH та інші варіанти цих алгоритмів. Також застосовуються прогностичні моделі на основі методів машинного навчання: методу опорних векторів, k-найближчих сусідів, дерев рішень, нейронних мереж різних типів. До другої категорії належать публікації, в яких автори намагаються об’єднати інформацію про конкретні події (фінансові або політичні новини, пошукові запити користувачів) та історію попередніх значень часових рядів. Такі моделі більш складні за рахунок великої кількості параметрів. Одним з методів розробки таких гібридних систем може бути метод побудови ансамблю різних моделей машинного навчання, які використовували б різні дані для навчання. Перевагою такого підходу є його адаптивність з точки зору використання обчислювальних ресурсів.

РОЗДІЛ 1

ПРОБЛЕМАТИКА ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Авторегресійні моделі

1.1.1 Авторегресія

Авторегресійні моделі описують змінний у часі випадковий процес, у якого цільова змінна лінійно залежить від минулих значень та випадкового члену.

Нехай ми маємо часовий ряд вигляду $\{r_t\}, t \in [1, N]$, де $N -$ кількість значень в часовому ряді. Якщо часовий ряд містить статистично значимий автокореляційний лаг першого порядку r_{t-1}, то його можна використати для прогнозування наступного значення r_t. Ця найпростіша модель авторегресії має наступний вигляд [2, 3]:

$$r_t = \varphi_0 + \varphi_1 r_{t-1} + a_t,$$

де $\{a_t\}$ – це білий шум з нульовим середнім і певною дисперсією σ_a^2.

Дана модель повторює структуру лінійної регресії, у якій r_t – це цільова змінна, а r_{t-1} – описова змінна. В літературі її називають модель авторегресії першого порядку або AR(1).

Узагальнюючі AR(1) до випадку AR(p) отримаємо наступну модель [2]:

$$r_t = \varphi_0 + \varphi_1 r_{t-1} + \cdots + \varphi_p r_{t-p} + a_t,$$

де p – це порядок авторегресії, $p \epsilon N$.

Під даним рівнянням, мається на увазі, що минулі значення r_{t-i} $(i = 1, \dots, p)$ разом визначають очікуване значення r_t.

1.1.2 Авторегресія з ковзним середнім

Для представлення авторегресії з ковзним середнім спочатку необхідно представити модель ковзного середнього. Модель ковзного середнього порядку 1 або MA(1) має наступний вигляд [3, 4]:

$$r_t = c_0 + a_t - \theta_1 a_{t-1},$$

де c_0 – це константа;

$\{a_t\}$ – це білий шум.

Якщо розширити це рівняння до загального випадку, то отримаємо наступну модель [2, 3, 4]:

$$r_t = c_0 + a_t - \theta_1 a_{t-1} - \cdots - \theta_q a_{t-q},$$

де $q > 0$ – порядок ковзного середнього.

В деяких задачах, порядок авторегресії або ковзного середнього вимагають досить високого порядку моделі, щоб адекватно описати динамічну структуру даних. Це призводить до великої кількості параметрів, які необхідно навчати і складності в обчисленнях. Щоб подолати цю проблему було представлено модель авторегресії з ковзним середнім або ARMA. Ця модель поєднує у собі ідеї авторегресії та ковзного середнього в компактній формі таким чином, щоб кількість параметрів лишалась низькою, але досягаючи гарної параметризації. Часовий ряд $\{r_t\}$ – підкорюється моделі ARMA(1, 1), якщо він задовольняє наступному рівнянню:

$$r_t - \varphi_1 r_{t-1} = \varphi_0 - \theta_1 a_{t-1} + a_t,$$

де $\{a_t\}$ – білий шум.

Ліва частина рівняння – це авторегресивна компонента моделі, а права – ковзного середнього.

Загальний вигляд ARMA(p, q) має наступну форму:

$$r_t = \varphi_0 + \sum_{i=1}^{p} \varphi_i r_{t-i} + a_t - \sum_{i=1}^{q} \theta_i a_{t-i},$$

де $\{a_t\}$ – це білий шум;

p і q – порядок авторегресії та ковзного середнього відповідно, $p, q \epsilon N$.

1.1.3 Авторегресія з інтегрованим ковзним середнім

Часовий ряд $\{y_t\}$ називають процесом ARIMA(p, 1, q), якщо ряд змін

$$c_t = y_t - y_{t-1} = (1 - B)y_t,$$

де B – оператор зворотного зсуву, підкоряється стаціонарній моделі ARMA(p, q). Трансформація нестаціонарного ряду в стаціонарний шляхом розгляду часового ряду його змін називають диференціюванням часового ряду. Ряд $c_t = y_t - y_{t-1}$ формально називають рядом перших різниць часового ряду y_t [3].

В деяких випадках диференціювання першого порядку не призводить до стаціонарного ряду. В такому випадку порядок диференціювання може бути більшим за одиницю. Таким чином приходимо до загальної моделі авторегресії з інтегрованим ковзним середнім ARIMA(p, d, q) [2, 3]:

$$\Delta^d y_t = \varphi_0 + \sum_{i=1}^{p} \varphi_i \Delta^d y_{t-i} + a_t - \sum_{i=1}^{q} \theta_i a_{t-i}$$

1.2 Гетероскедастичні моделі

1.2.1 Авторегресія з умовною гетероскедастичністю

Перша модель, яка представила повноцінний інструмент для моделювання волатильності стала модель авторегресії з умовною гетероскедастичністю або ARCH. Базовою ідеєю для цієї моделі стало те, що шум a_t не має серійної залежності, але залежний, і цю залежність можна описати простою квадратичною функцією від минулих значень. Формально, модель ARCH(m) має на увазі наступне:

$$a_t = \sigma_t \epsilon_t, \quad \sigma_t^2 = \alpha_0 + \alpha_1 a_{t-1}^2 + \cdots + \alpha_m a_{t-m}^2,$$

де $\{\epsilon_t\}$ – це послідовність незалежних і однаково розподілених випадкових величин з нульовим середнім і дисперсією рівною 1, $\alpha_0 > 0$ і $\alpha_i \geq 0$ для $i > 0$. Коефіцієнти α_i повинні задовольняти певним умовам регулярності, щоб дисперсія a_t була скінченною. На практиці зазвичай припускають, що ϵ_t розподілена за стандартним розподілом, t-розподілом Стьюдента або узагальненим нормальним розподілом.

Зі структури моделі видно, що великі значення квадратів пертурбації $\{a_{t-i}^2\}, i = 1 \ldots m$, призводять до великої умовної дисперсії a_t, яка, в свою чергу, має тенденцію приймати великі значення. Це значить, що великі пертурбації мають тенденцію слідувати за іншими великими пертурбаціями.

1.2.2 Узагальнена авторегресія з умовною гетероскедасичністю

Хоча модель ARCH досить проста, вона зазвичай вимагає багато параметрів, щоб адекватно описати процеси волатильності. Тому була необхідна якась альтернатива. Цю модель розширили і назвали узагальнена авторегресія з умовною гетероскедастичністю GARCH. Ряд a_t підпорядковується моделі GARCH(m, s), якщо:

$$a_t = \sigma_t \epsilon_t, \quad \sigma_t^2 = \alpha_0 + \sum_{i=1}^{m} \alpha_i a_{t-i}^2 + \sum_{j=1}^{s} \beta_j \sigma_{t-j}^2,$$

де $\{\epsilon_t\}$ – це послідовність незалежних і однаково розподілених випадкових величин з нульовим середнім і дисперсією рівною 1, $\alpha_0 > 0$, $\alpha_i \geq 0$, $\beta_j \geq 0$ і $\sum_{i=1}^{\max(m,s)} (\alpha_i + \beta_i) < 1$.

Останнє обмеження на $\alpha_i + \beta_i$ передбачає, що дисперсія a_t скінченна, в той час як умовна дисперсія σ_t^2 змінюється з часом. Як і в моделі ARCH, зазвичай припускають, що ϵ_t підпорядковуються стандартному розподілом, t-розподілу Стьюдента або узагальненому нормальному розподілу. Якщо параметр моделі $s = 0$, то модель зводиться до звичайної моделі ARCH(m).

1.2.3 Експоненційна узагальнена авторегресія з умовною гетероскедасичністю

Для того, щоб подолати деякі слабкості моделі GARCH у роботі з фінансовими часовими рядами, було запропоновано експоненційну узагальнену авторегресію з умовною гетероскедастичністю або EGARCH.

Зокрема, щоб забезпечити асиметричність між позитивними і негативними змінами в акціях, ввели зважену інноваційну компоненту:

$$g(\epsilon_t) = \theta\epsilon_t + \gamma[|\epsilon_t| - E(|\epsilon_t|)],$$

де θ і γ – дійсні константи;

ϵ_t і $|\epsilon_t| - E(|\epsilon_t|)$ – однаково розподілені послідовності з неперервного розподілу, з нульовим середнім.

Тому, $E[g(\epsilon_t)] = 0$. Асиметричність $g(\epsilon_t)$ можна продемонструвати, переписавши рівняння наступним чином:

$$g(\epsilon_t) = \begin{cases} (\theta + \gamma)\epsilon_t - \gamma E(|\epsilon_t|), & \epsilon_t \geq 0, \\ (\theta - \gamma)\epsilon_t - \gamma E(|\epsilon_t|), & \epsilon_t < 0. \end{cases}$$

Тоді, модель EGARCH(m, s) можна записати як:

$$a_t = \sigma_t\epsilon_t, \quad \ln(\sigma_t^2) = \alpha_0 + \frac{1+\beta_1 B+\cdots+\beta_{s-1}B^{s-1}}{1-\alpha_1 B-\cdots-\alpha_m B^m} g(\epsilon_{t-1}),$$

де α_0 – це константа;

B – це оператор зворотного зсуву.

Моделі GARCH і EGARCH мають декілька відмінностей. По-перше, EGARCH використовує затриману умовну дисперсію, щоб послабити обмеження на додатність коефіцієнтів моделі. По-друге, використання $g(\epsilon_t)$ дозволяє моделі асиметрично сприймати додатні і від'ємні затримані значення a_t.

Також існує альтернативна форма моделі EGARCH(m, s):

$$\ln(\sigma_t^2) = \alpha_0 + \sum_{i=1}^{s} \alpha_i \frac{|\alpha_{t-i} + \gamma_i a_{t-i}|}{\sigma_{t-i}} + \sum_{j=1}^{m} \beta_j \ln(\sigma_{t-j}^2).$$

У випадку з додатнім значенням a_{t-i} маємо $\alpha_i(1+\gamma_i)|\epsilon_{t-i}|$ в першій сумі, а якщо a_{t-i} – від'ємне, то маємо $\alpha_i(1-\gamma_i)|\epsilon_{t-i}|$, де $\epsilon_{t-i} = a_{t-i}/\sigma(y-i)$. В такому випадку, параметр γ_i грає роль важеля для a_{t-i}.

1.3 Методи попереднього аналізу даних

В роботі було використано ціни на акції компанії Tesla в періоді з 2013-01-01 по 2020-01-01. Дані містять 1762 запис без пропусків. Загальна статистика щодо набору даних міститься у Таблиці 1.1.

Таблиця 1.1 – Загальна статистика цін на акції компанії Tesla

Кількість	**Середнє**	**Стандартне відхилення**	**Мінімум**	**Максимум**
1762	47.77	15.54	6.58	86.19

Проведемо базовий аналіз часового ряду. Спочатку побудуємо графік оригінального ряду (Рис.1.1).

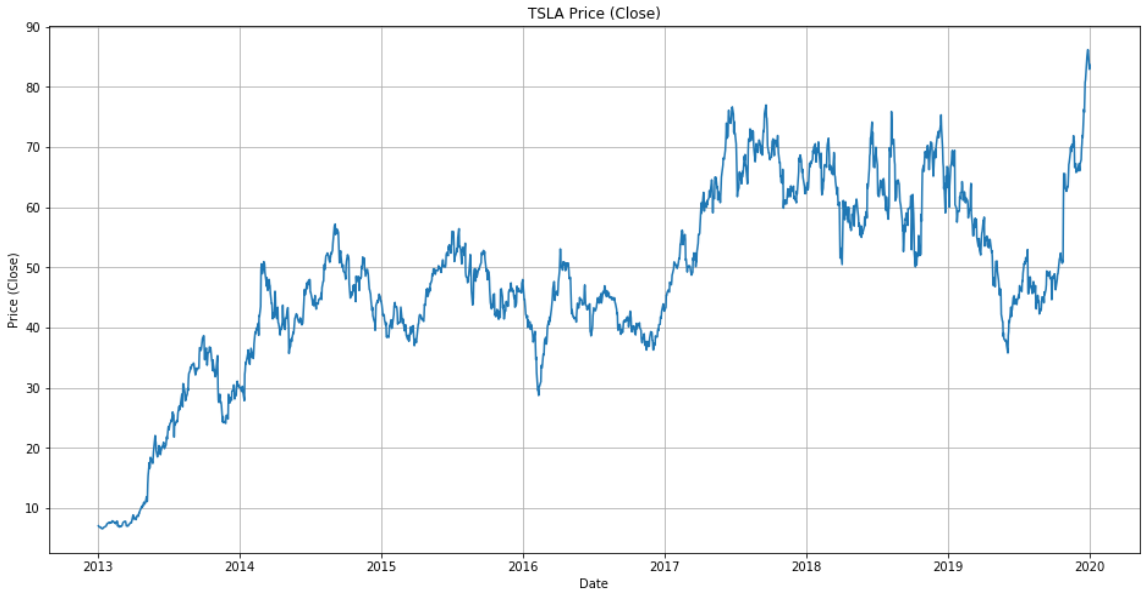


Рисунок 1.1 – Графік цін компанії Tesla на момент закриття торгового дня

Наступним етапом розглянемо графіки різниць першого порядку та відсоткових змін, оскільки вони відображають адитивну та мультиплікативну дохідність, які зображені на Рисунках 1.2-1.3 відповідно.

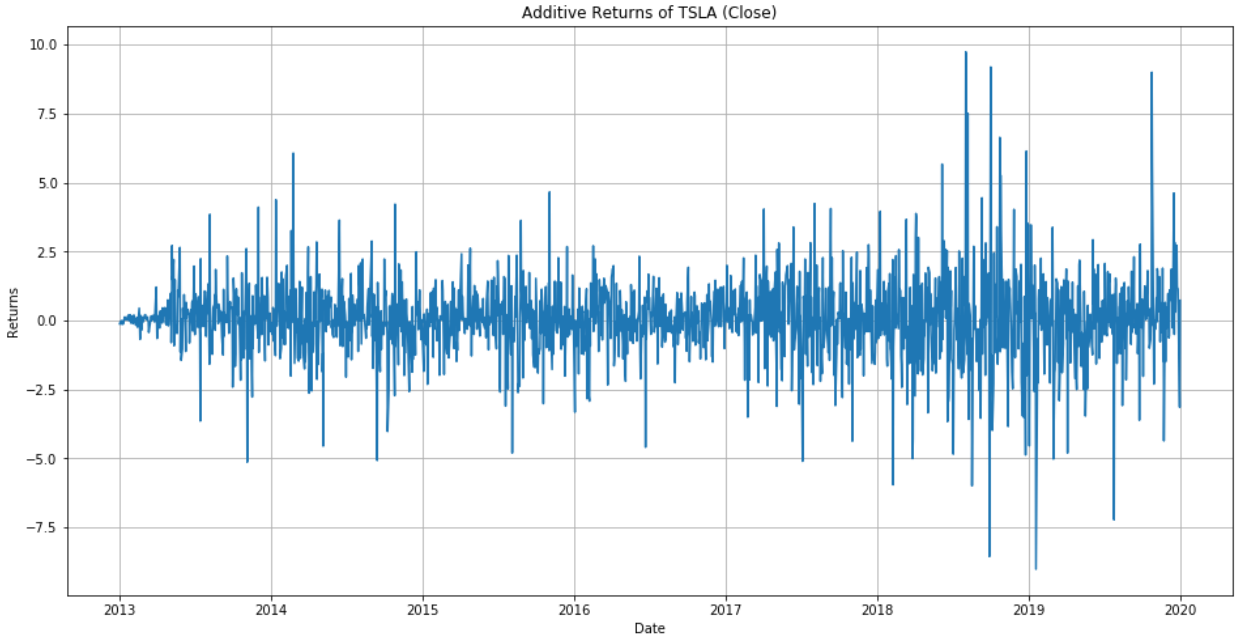


Рисунок 1.2 – Графік адитивних змін акцій TSLA

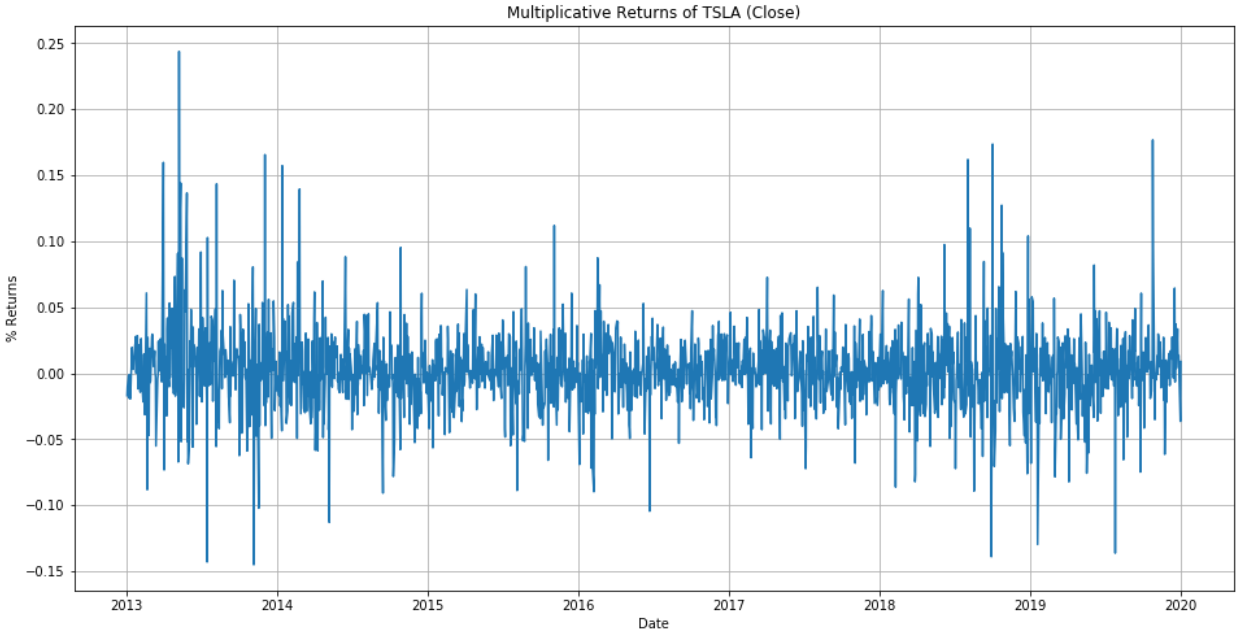


Рисунок 1.3 – Графік мультиплікативних змін акцій TSLA

Також важливим графіком є графік ковзного середнього (Рис.1.4) та ковзного стандартного відхилення (Рис.1.5) з вікном 30 днів.

Рисунок 1.4 – Графік ковзного середнього з вікном 30 днів та обраного часового ряду

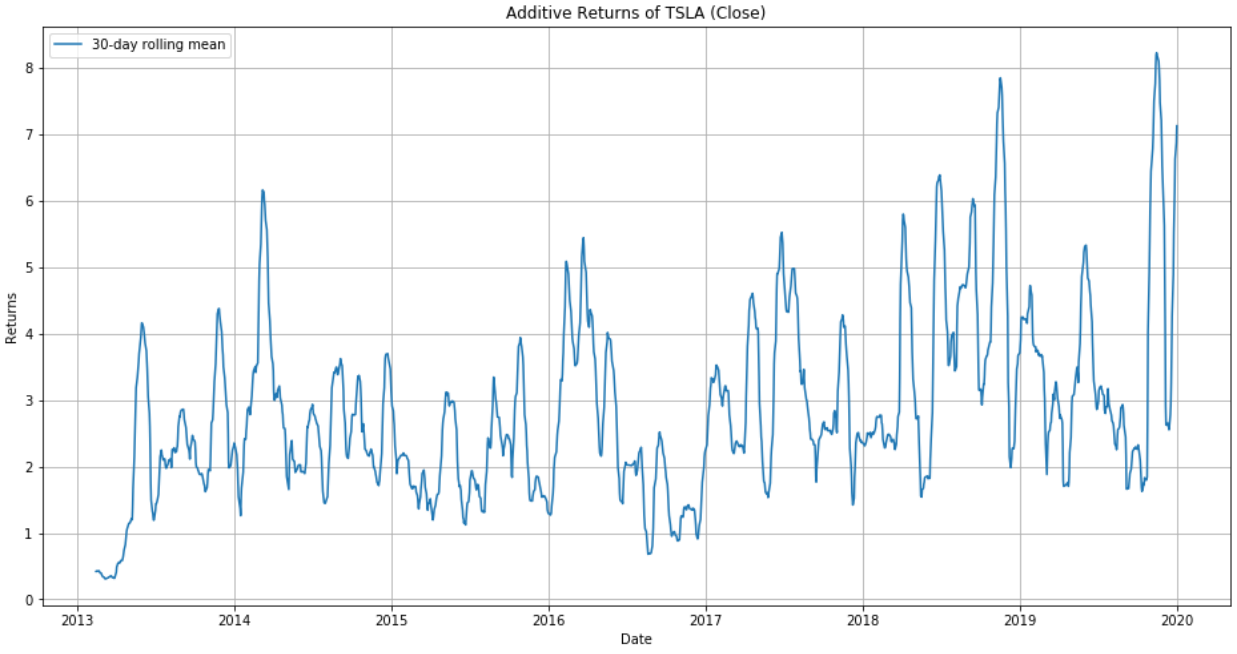


Рисунок 1.5 – Графік ковзного стандартного відхилення з вікном 30 днів

1.3.1 Графічний аналіз

Графічне відображення даних може досить сильно допомогти у вивченні поведінки даних та побачити потенціальні структури або недоліки. Але необхідно бути пильними, оскільки графіки можуть сприйматися так, ніби там є (або відсутній) деякий ефект/структура, а насправді все навпаки. Тому графіки необхідно використовувати для формулювання гіпотез, а не їх тестування. Тому в цій частині побудуємо декілька базових графіків, які нам у цьому допоможуть.

Першими, що ми побудуємо в цій секції будуть гістограми. Вони показують частоту різних значень в наборі даних. Вона дозволяє нам швидко побачити де більшість даних сконцентрована.

На Рисунку 1.6 зображено гістограму розподілу наших даних з використанням 30 «плиток».

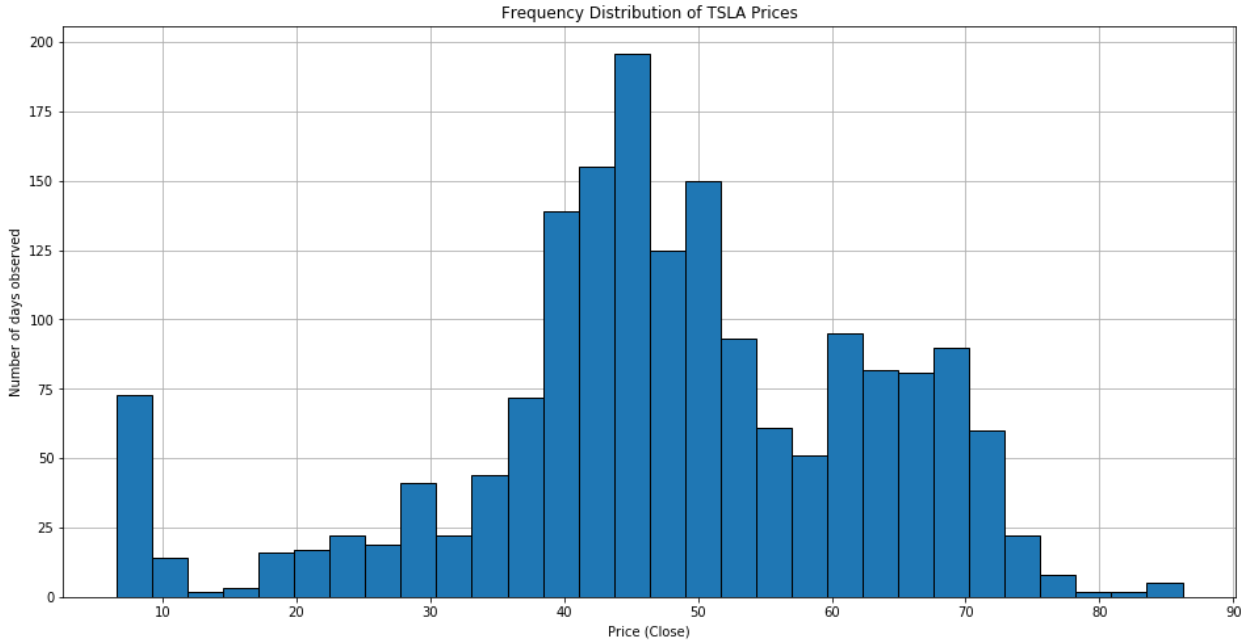


Рисунок 1.6 – Гістограма розподілу цін акцій TSLA

В фінансовій сфері не часто дивляться на розподіл цін. Причиною цього є те, що ціни не стаціонарні і досить сильно коливаються. Тому краще використовувати зміну цін, що й зображено на наступному Рисунку 1.7.

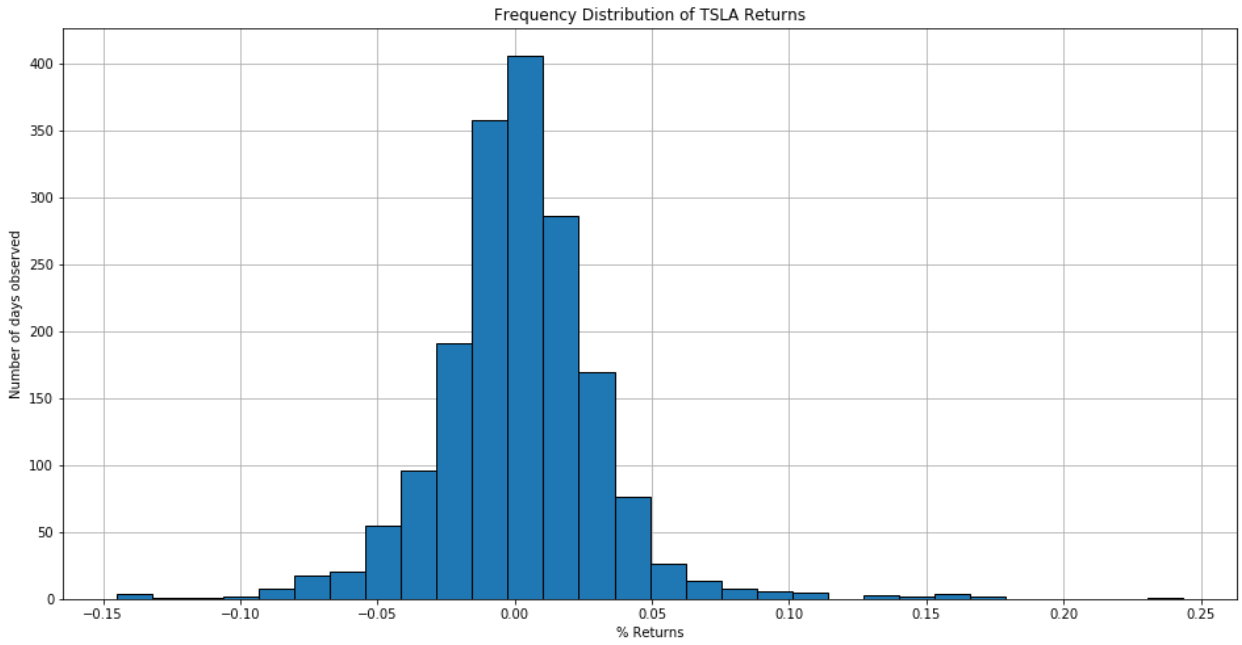


Рисунок 1.7 – Гістограма розподілу змін цін на акції TSLA

Гістограма показує, що щоденні зміни акцій Tesla були вищимм від 0.2 приблизно 10 днів. Необхідно взяти до уваги, що ми повністю відкидаємо дати, відповідні цим змінам. Також цей графік нічого не каже про те, що майбутня дохідність матиме такий самий розподіл.

Альтернативним підходом до відображення даних може бути використання кумулятивної функції розподілу, в якій висота «плитки» показує кількість спостережень, що лежать в цій «плитці» і в попередніх. Ця гістограма завжди неспадаюча, оскільки неможливо мати від’ємну кількість спостережень. На Рисунку 1.8 зображено кумулятивний розподіл доходів акцій Tesla.

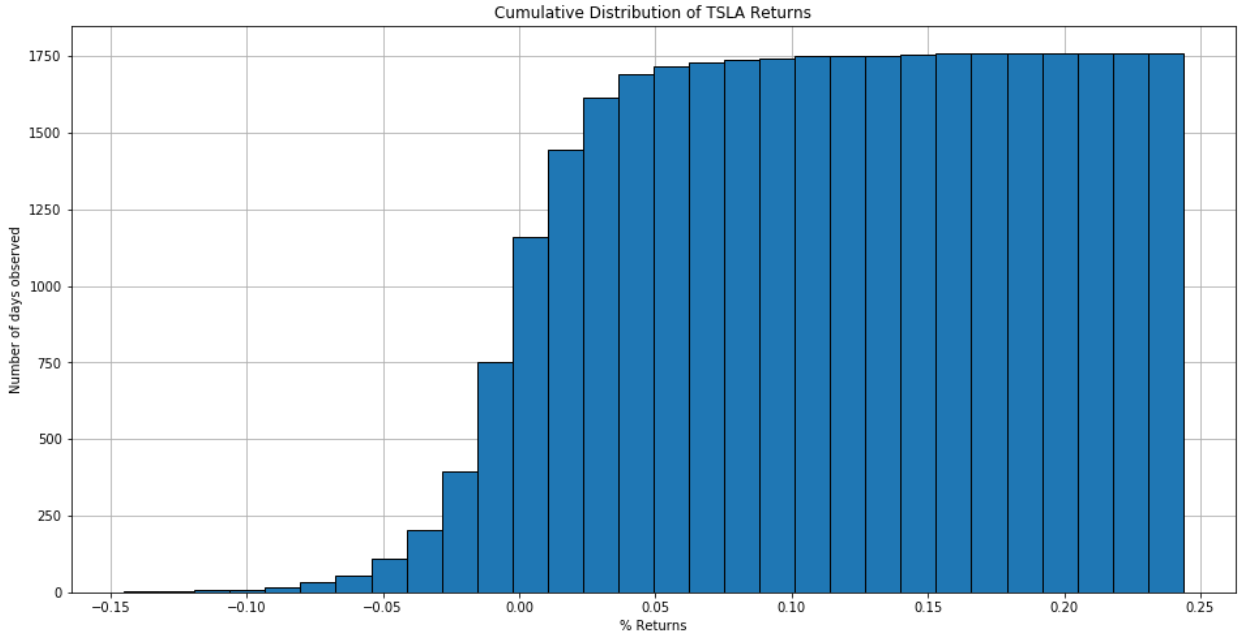


Рисунок 1.8 – Кумулятивний розподіл змін на акції компанії Tesla

1.3.2 Аналіз середніх

В цій частині розглянемо різні методи оцінки середнього. Середнє є важливим числом, оскільки воно відображає поведінку даних. Знайдемо наступні значення:

- Середнє арифметичне. Воно є досить чутливим до екстремальних значень.
- Медіана. Вона менш чутлива до екстремальних значень в даних, ніж середнє арифметичне. Вона дає нам значення, яке ділить вибірку даних навпіл, але не показує наскільки меншими або більшими інші значення є.
- Мода. Вона дає нам найбільш популярне значення датасету.

В Таблиці 1.2 наведені значення середніх для змін цін.

Таблиця 1.2 – Розраховані арифметичне середнє, медіана та мода для цін на акції TSLA.

	Арифметичне Середнє	**Медіана**	**Мода**
Ціна	47.774	47.440	(-0.0017, 0.0113)
Мультиплікативна дохідність	0.0019	0.0010	(-0.0017, 0.0113)

Середні приховують багато інформації, оскільки вони об’єднують весь розподіл в одне число. Як результат, «точкові оцінки» чи метрики, що використовують одне число, можуть приховувати великі закономірності в даних. Тому необхідно бути пильним, щоб не втратити ключову інформацію шляхом узагальнення даних одним числом. Через це, середнє необхідно розглядати разом з мірою розкиду.

Іноді, навіть коли використовуються правильні метрики середнього і розкиду, вони можуть не мати ніякого сенсу, якщо розподіл не є таким, яким ми його вважаємо. Наприклад, використання стандартного відхилення, щоб виміряти частоту події зазвичай має припущення, що дані розподілені нормально. Тому завжди необхідно ретельно перевіряти чи дані відповідають обраному розподілу.

1.3.3 Аналіз розкиду даних

Дисперсія вимірює наскільки розкиданий набір даних. Це дуже важливо в фінансах, оскільки один із шляхів вимірювання ризику полягає в тому, щоб оцінити як були розкидані зміни в минулому. Якщо зміни розкидані дуже близько до центрального значення, то тоді ризик малий, а коли він розкиданий досить сильно, то і маємо високий ризик.

Данні з малою дисперсією концентруються навколо середнього, в той час як з великою дисперсією вказують на велику кількість дуже великих і дуже малих значень.

Спочатку подивимось на діапазон доходності в нашому датасеті. Діапазон є досить сильно чутливим до викидів. Маємо:

$$Range = 0.3888$$

Наступною метрикою розкиду розглянемо середнє абсолютне відхилення (MAD). Воно показує середню відстань від спостережень до середнього значення:

$$MAD = 0.0213$$

Наступні дві метрики використовують найчастіше – дисперсія та стандартне відхилення навколо середнього. Вони використовуються частіше ніж абсолютне середнє відхилення, тому що вони диференційовані і тому можуть використовуються деякими алгоритмами оптимізації, які покладаються на диференціювання. Стандартне відхилення є коренем дисперсії, тому його легше інтерпретувати до даних:

$$Variance = 0.001$$
$$STD = 0.0311$$

Стандартне відхилення можна інтерпретувати за допомогою нерівності Чебишева. Вона показує те, що відсоток вибірки, що лежить в діапазоні від k стандартних відхилень до середнього принаймні $1 - \frac{1}{k^2}$ для всіх $k > 1$.

Хоча дисперсія та стандартне відхилення кажуть нам наскільки волатильною є величина, вони не беруть до уваги додатнім було відхилення

чи від’ємним. Зазвичай, як у випадку доходності активу, набагато важливішими є коливання вниз. Цю проблему можна вирішити за допомогою «напівдисперсії» та «напів відхилення», які враховують лише ті спостереження, що лежать нижче середнього:

$$Semivar = 0.0008$$
$$Semidevation = 0.0291$$

Всі ці обчислення надають статистику вибірки. Чи відображає це поточне відхилення не завжди очевидно і тому необхідно прикладати більше зусиль для визначення цього. Це особливо проблематично в фінансах, тому що всі дані, з якими оперують є часовими рядами і середнє та дисперсія можуть змінюватися з часом.

1.3.4 Аналіз статистичних моментів

Іноді середнього та дисперсії не вистачає, щоб достатньо описати розподіл. У випадку великих відхилень ми не знаємо чи є вони позитивними чи негативними. В такому випадку нам можуть допомогти зміщення та симетричність розподілу. Розподіл є симетричним, якщо обидві сторони від середнього значення є дзеркальними відображеннями один одного. Якщо розподіл не симетричний, то його називають зміщеним. Симетричний розподіл має зміщення 0. Розподіл з додатним зміщенням і однією модою має наступну властивість: $mean > median > mode$. В унімодального розподілу з негативним зміщенням обернена властивість: $mean < median < mode$.Розрахуємо зміщення нашого розподілу:

$$Skew = 0.6964$$

Як ми бачимо, наші дані додатньо зміщенні. Оскільки дохідність має додатне зміщення, то можна стверджувати, що її волатильність характеризується частими мали змінами в ціні з рідкими скачками угору.

Ще одним статистичним моментом, що можна розглянути є коефіцієнт ексцесу. Він дозволяє вимірювати форму відхилення від середнього. Взагалі, ексцес описує наскільки «крутим» є розподіл у порівнянні з нормальним. Всі нормальні розподіли незалежно від середнього та дисперсії мають коефіцієнт ексцесу рівний 3. «Лептокуртичний» (leptokurtic) розподіл (ексцес > 3) має високий пік і довгі хвости, в той час як «платикуртичний» (platykurtic) розподіл (ексцес < 3) – навпаки. У лептокуртичного розподілу частіші великі стрибки подалі від середнього, ніж у звичайного, тоді як у платикуртичного розподілу менше. В бібліотеках Python використовують коефіцієнт ексцесу такий, що при нормальному розподілі він дорівнює 0.

$$Kurtosis = 6.7629$$

Оскільки коефіцієнт ексцесу позитивний, то розподіл дохідності лептикуртичний. На це також вказує те, що гістограма доходності має багато спостережень, які виходять за 3 стандартних відхилення від середнього.

На Рисунку 1.9 можна переглянути візуально результати аналізу зміщення та коефіцієнту ексцесу:

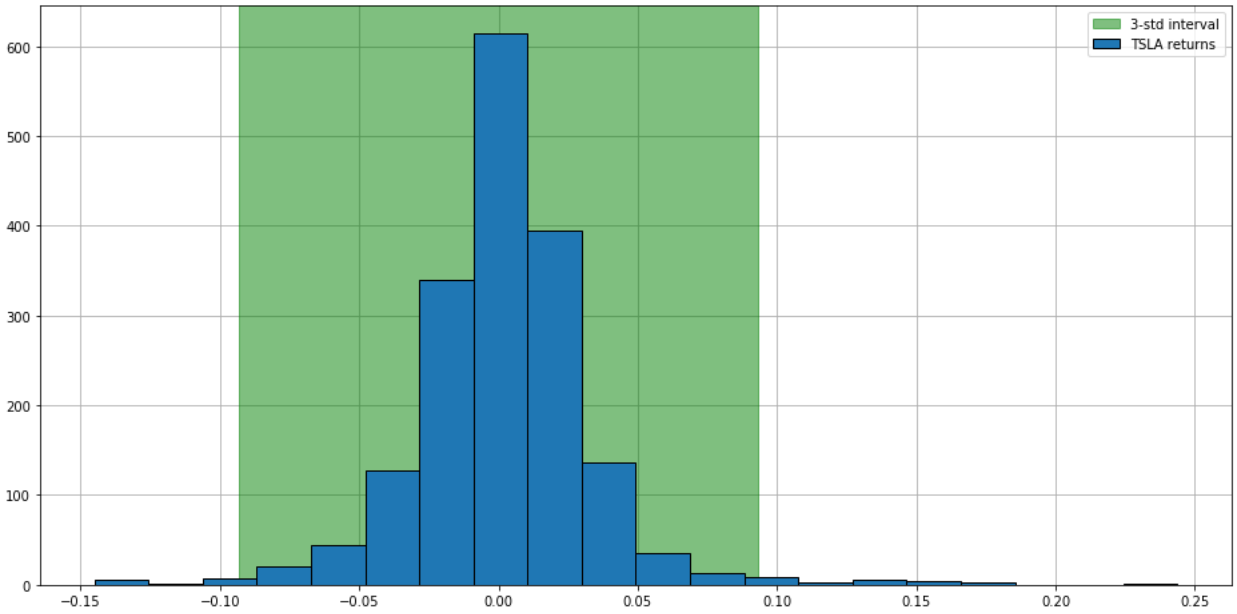


Рисунок 1.9 – Гістограма розподілу змін акцій та інтервал в -3/+3 стандартних відхилення

Проведемо тест Харке-Бера, який порівнює зміщення та коефіцієнт ексцесу з нормальним розподілом. Проведемо його на даних про зміни акцій TSLA і знайдемо p-value. Нульова гіпотеза теста Харке-Бера каже про те, що дані взяті з нормального розподілу.

$$p - value = 0.0$$

Оскільки p-value < 0.05, то наша дохідність скоріше за все не розподілена за нормальним законом. Продемонструємо це побудувавши графік щільності нормального розподілу з середнім та стандартним відхиленням аналогічним середній дохідності разом з нормованою гістограмою доходності (Рис.1.10).

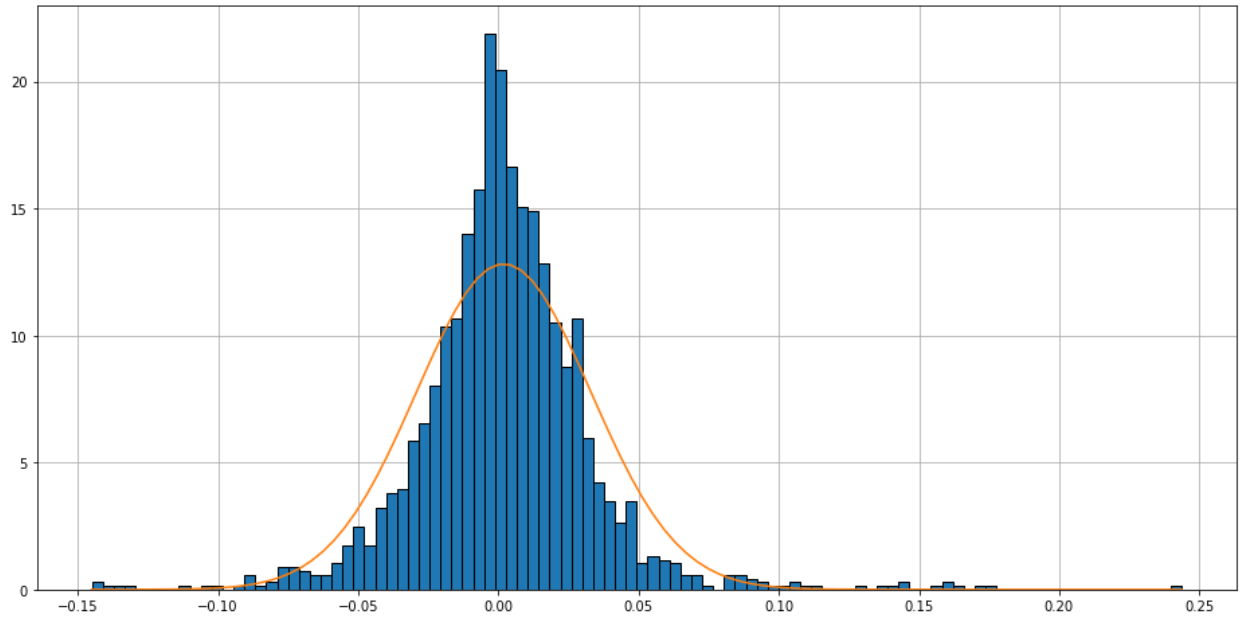


Рисунок 1.10 – Гістограма розподілу змін та графік нормального розподілу з відповідними середнім та дисперсією

Наша теоретична крива змін має істотно нижчий пік ніж реальні дані, що й доводить тест Харке-Бера. Знову ж таки, це через коефіцієнт ексцесу нормального розподілу. Дохідність має коефіцієнт ексцесу рівний 6.7154, в той час як нормальний розподіл має його рівний 3. Більший коефіцієнт ексцесу призводить до вищого піку. Це показує головну причину, чому так важко моделювати ціни на акції та їх дохідність - невідомий розподіл даних. Багато теорем та інструментів в фінансах вимагають, щоб дані були якось пов’язані з нормальним розподілом. Це головна причина чому нормальний розподіл такий розповсюджений. Тим не менш, досить важко знайти реальні дані, які ідеально підходять під припущення на нормальність.

Загалом, коли необхідно підігнати дані під якийсь ймовірнісний розподіл, завжди треба тримати певний розподіл (або розподіли) в голові. Існує багато тестів для різних розподілів, які можна застосувати до даних, щоб зрозуміти який саме розподіл підходить найкраще. Також, з отриманням нових даних, необхідно оновлювати середнє та стандартне відхилення вибірки або можна знайти інший розподіл який більш точно відображає поведінку нових даних.

Останнім кроком буде порівняння зміщення наших даних з тестовою вибіркою. Для тестової вибірки було обрано всі дані починаючи з 2020 року.

$$skew = -0.0628$$

Наше значення зміщення від'ємне, що може вказувати на те, що або тестова вибірка мала, або ж ми маємо справу з волатильним зміщенням, тому використовувати його як постійне значення – неправильно. Для перевірки побудуємо графік ковзного зміщення з вікном в 60 днів (Рис.1.11).

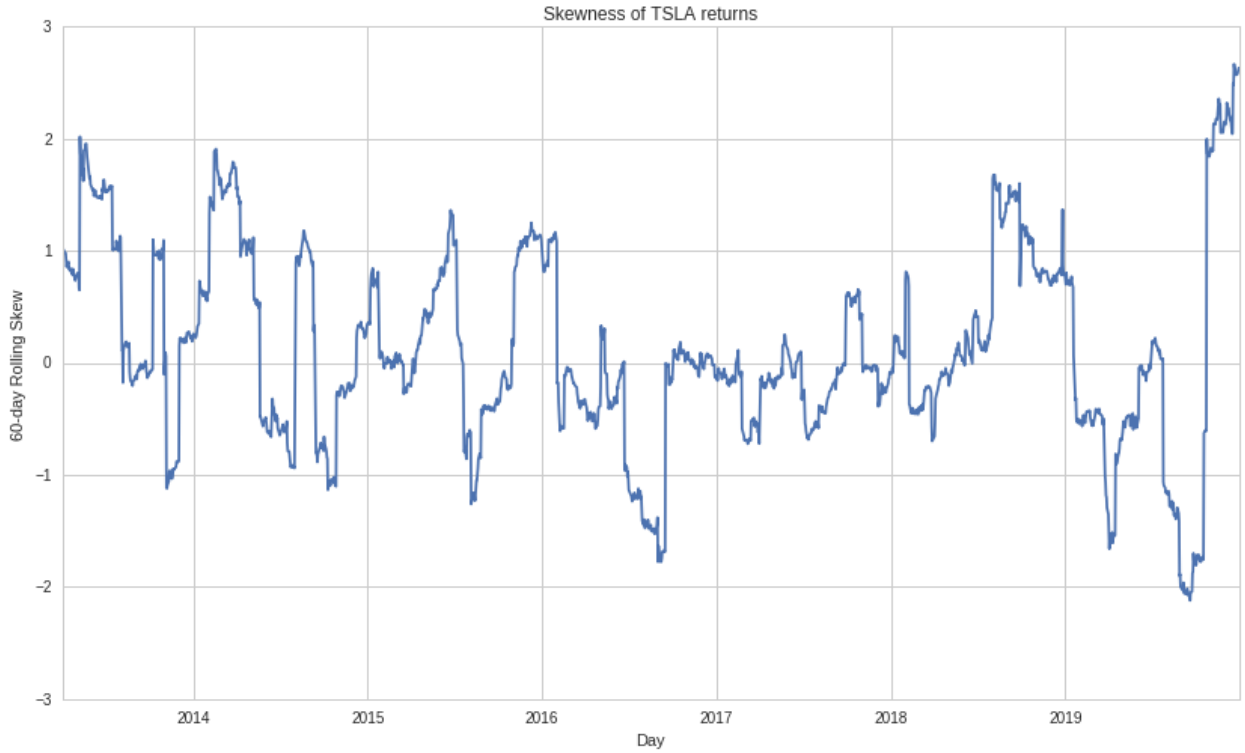


Рисунок 1.11 – Графік ковзного зміщення з вікном 60 днів

Графік підтверджує наші підозри на те, що зміщення занадто волатильне, щоб ми могли його використовувати для прогнозування.

1.3.5 Нестабільність оцінок

Будь-яка оцінка проводиться із ступенем невизначеності, але зазвичай ця невизначеність ігнорується. Це дуже ризиковано в фінансах, оскільки неправильна оцінка може стати причиною великих втрат замість стабільного доходу.

Параметри - це все, що модель використовує для своїх прогнозів. Зазвичай, параметр - це величина, що допомагає описувати набір даних чи їх розподіл. Наприклад, середнє нормального розподілу - це параметр, тому ми кажемо, що нормальний розподіл параметризований середнім та дисперсією. Якщо взяти середнє вибірки отриманої з нормального розподілу, то можна отримати оцінку середнього розподілу. Аналогічно, середнє набору спостережень - це оцінка параметру розподілу, з якого взято дані. Іншими параметрами можуть бути медіана, коефіцієнт кореляції з іншим часовим рядом, стандартне відхилення, т.д.

Тому коли беремо середнє від набору даних, то ми не знаємо середнє значення самого розподілу. Ми оцінюємо середнє як можна краще з даних, що маємо. Оцінка може бути досить неточною і це залишається справедливим для всіх параметрів. Щоб визначити наскільки добра оцінка параметру необхідно розглядати стабільність, стандартне відхилення або довірчі інтервали для неї.

Кожного разу коли ми розглядаємо набір даних, наші розрахунки параметрів можуть бути лише їх оцінкою. Оцінка може змінюватися з часом або якщо ми знайдемо більше даних. Можна визначити кількісно невизначеність наших оцінок дивлячись як параметр змінюється, якщо ми розглядаємо різні підмножини наших даних. Наприклад, стандартне відхилення описує наскільки відрізняється середнє значення датасету від значення кожного спостереження. У фінансах дані зазвичай подаються у вигляді часових рядів. В такому випадку, можна оцінювати параметри в різних часових інтервалах. Дивлячись на те, як сильно ця ковзна оцінка коливається

в залежності від часового вікна, можна розрахувати нестабільність параметру, що оцінюємо.

Однією статистикою, яку часто використовують для опису ефективності активів є коефіцієнт Шарпа [1], який вимірює додатковий дохід на одиницю додаткового ризику, відносно безризикового джерела прибутку, такого як казначейські векселі. Так само як і середнє чи стандартне відхилення можна обчислити ковзний коефіцієнт Шарпа, щоб побачити як оцінка змінюється з часом. Щоб побачити як впливає розмір вікна на коефіцієнт Шарпа, розрахуємо значення ковзного середнього для 3-х різних вікон (50, 150, 300) і розрахуємо середнє значення та стандартне відхилення для знайдених часових рядів (Табл.1.3).

Таблиця 1.3 – Середнє та стандартне відхилення ковзного середнього коефіцієнта Шарпа для 3-х вікон: 50, 150 та 300 днів

Розмір вікна	Середнє	Стандартне відхилення
50	0.057994	0.162617
150	0.048754	0.078980
300	0.043242	0.047560

Як ми бачимо з таблиці, зі збільшенням довжини вікна волатильність ковзного коефіцієнту Шарпа зменшується.

Тепер перевіримо чи можна використовувати знайдені середні та стандартні відхилення для прогнозування часового ряду на тестовій вибірці. Для цього побудуємо графіки ковзного середнього на тестовій вибірці і візьмемо середнє та стандартне відхилення з тренувальної (Рис.1.12-1.14).

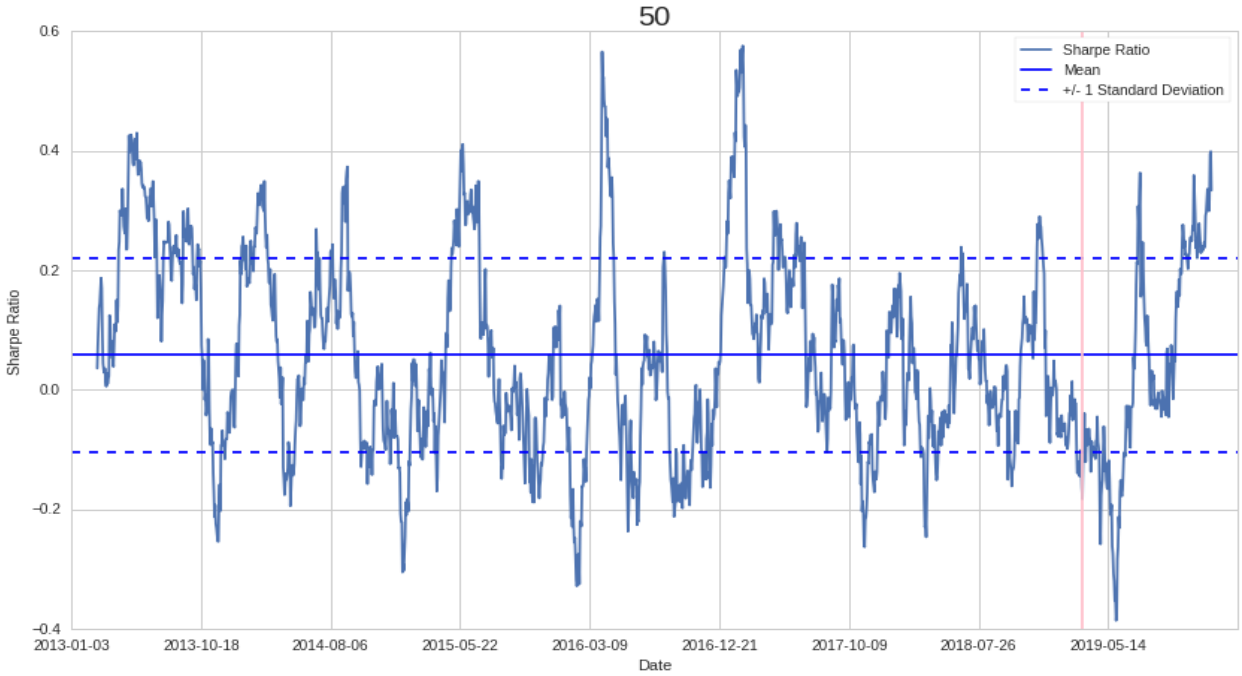


Рисунок 1.12 – Графік ковзного середнього коефіцієнта Шарпа з вікном в 50 днів обчислений на тестовій вибірці. Пунктирною лінією позначено інтервал в $+/-1$ стандартне відхилення, а пряма – середнє значення, які обчислені на тренувальній вибірці

Рисунок 1.13 – Графік ковзного середнього коефіцієнта Шарпа з вікном в 150 днів обчислений на тестовій вибірці. Пунктирною лінією позначено інтервал в $+/-1$ стандартне відхилення, а пряма – середнє значення, які обчислені на тренувальній вибірці

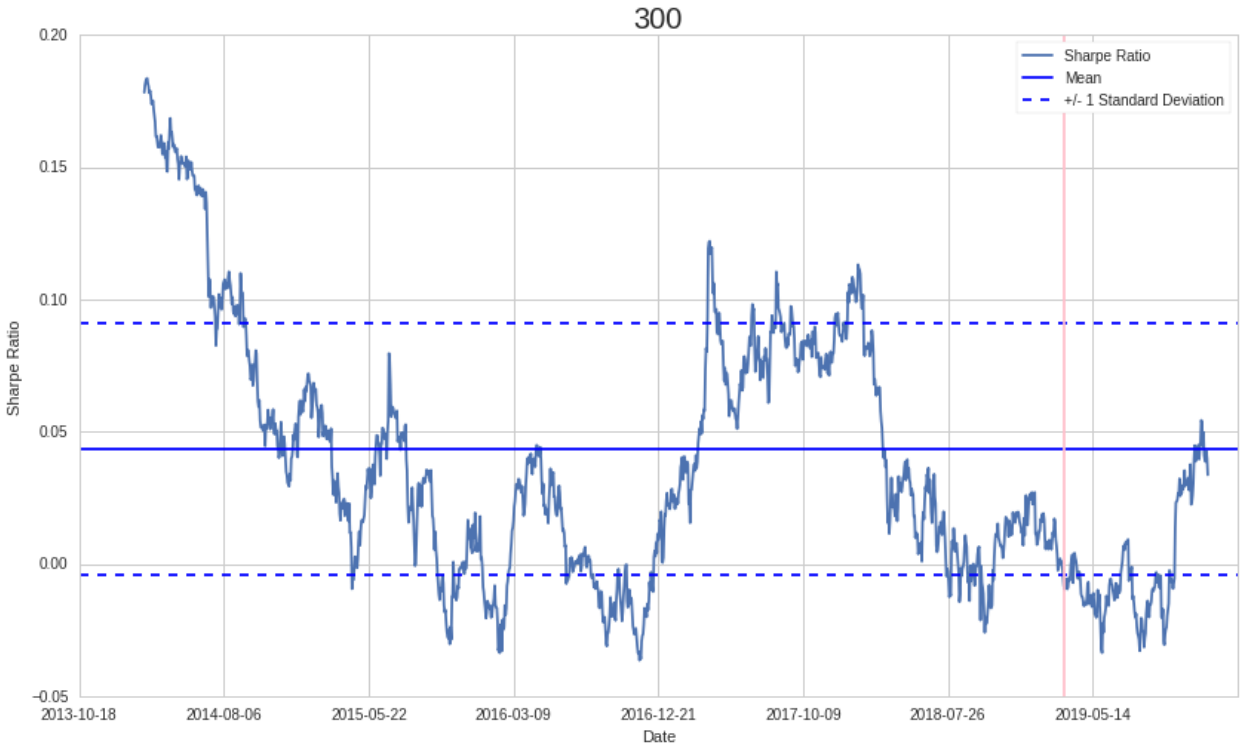


Рисунок 1.14 – Графік ковзного середнього коефіцієнта Шарпа з вікном в 300 днів обчислений на тестовій вибірці. Пунктирною лінією позначено інтервал в $+/-1$ стандартне відхилення, а пряма – середнє значення, які обчислені на тренувальній вибірці

Не дивлячись на те, що ширші вікна мають меншу волатильність, вони також досить непередбачуваними, якщо користуватися лише середнім значенням. Але разом зі стандартним відхиленням середнє набуває набагато більшого прогнозуючого значення. Також можна спробувати використати тренд, але в такому випадку все одно краще додавати стандартне відхилення.

Оскільки наш часовий ряд є досить волатильним, то можна скористатися ковзним середнім та ковзним стандартним відхиленням для більш точного прогнозування (лінії Болліндженра) (Рис.1.15).

Рисунок 1.15 – Графік цін на акції та лінії Болліндженра

Як ми бачимо з графіку, то в цьому випадку ковзне середнє набуває зовсім іншого значення і вже може використовуватися для більш точного прогнозування цін на акції. Також необхідно вказати, що відсутні результати ковзного середнього та стандартного відхилення для перших 90 днів, оскільки ми використовуємо вікно розміром в 90 днів.

Висновком до цього можна сказати, що завжди коли ми розраховуємо параметри датасету, треба також рахувати його волатильність. Інакше, не можна сказати, що нові дані будуть підходити під отримані параметри. Хорошим способом розрахунку волатильності є розбиття даних на підмножини і оцінка параметрів для кожної з підмножини, а потім знайти мінливість результатів. Також можуть бути різні зовнішні фактори, які не представлені в даних і які ми не можемо прогнозувати. Тим не менш, аналіз нестабільності і тестування стандартного відхилення залишається дуже корисним для визначення того, наскільки нам необхідно довіряти нашим оцінкам.

1.4 Постановка задачі дослідження

У попередніх пунктах були розглянуті основні методи аналізу та дослідження часових рядів. «Класичні» методи, що були розглянуті вище оперують стаціонарними рядами, тому будь-який часовий ряд перед моделюванням необхідно приводити до такого шляхом чисельного диференціювання (застосовуючи кінцеві різниці), щоб позбавити ряд тренду, відмінного від константи; масштабування, логарифмування, тощо. Також необхідно вручну обирати ступінь авторегресії та ковзного середнього, аналізуючи показники автокореляції. Тобто втручання людини у процес інжинірингу даних займає більшість часу у процесі моделювання та визначатиме адекватність моделі в цілому.

У свою чергу, методи, що використовують машинне та глибинне навчання в основному потребують лише масштабування даних (в залежності від функції активації) та перетворення самої форми подачі ряду від вигляду «час»-«значення» до «значення за попередні періоди»-«значення за поточний період», тобто подати у вигляді «ознаки-результуюче значення».

Основною проблемою такого представлення є інваріантність щодо зафіксованих значень ряду відносно часу. Простіше кажучи, таке представлення ряду припускає, що кожне значення ряду фіксується з однаковим інтервалом, хоча це не завжди так.

У даній дисертації ставляться наступні задачі:

1) дослідити сучасний стан та особливості фінансових ринків;
2) проаналізувати існуючі методи прогнозування процесів на фінансових ринках та виконати огляд методів та підходів до прогнозування;
3) дослідити методи відбору архітектур та гіперпараметрів глибоких нейронних мереж; розробити власний алгоритм вибору

архітектури та гіперпараметрів моделей нейронних мереж для прогнозування фінансових часових рядів;

4) виконати побудову математичних моделей на реальних статистичних даних та отримати практичні результати, виконати аналіз результатів;
5) розробити концептуальні висновки за результатами наукового дослідження.

Висновки за розділом 1

У даному розділі було проведено детальний аналіз часового фінансового ряду. Зокрема розраховано статистики середнього та розкиду даних. Також в даному розділі було проведено порівняння розподілу часового ряду з відповідним йому нормальним розподілом. Було проілюстровано факт того, що статистичні дані часового ряду змінюються з часом, що є індикатором нестаціонарного ряду. Також було описано деякі фінансові статистики.

Також було розглянуто математичне визначення різних економетричних моделей, а саме: авторегресія, авторегресія з ковзним середнім та авторегресія з інтегрованим ковзним середнім.

РОЗДІЛ 2

МЕТОДИ НЕЙРОННИХ МЕРЕЖ ДЛЯ ПРОГНОЗУВАННЯ ЧАСОВИХ РЯДІВ

2.1 Архітектури нейронних мереж

2.1.1 Багатошаровий перцептрон

Нейронні мережі прямого розповсюдження або **багатошаровий перцептрон** – є базовою моделлю глибоко навчання. Вони мають таку назву, оскільки вхідні дані x поступово проходять через кожен прихований шар, які апроксимують певну функцію f^*, до вихідного шару, який вже надає результат y. Цей тип нейронних мереж не має обернених зв'язків, які пов'язують вихідний шар з вхідним (як, наприклад, у рекурентних).

На Рисунку 2.1 зображена архітектура нейронної мережі [5]. Вона складається з вхідного шару, прихованих шарів та вихідного шару. Кількість прихованих шарів визначають глибину нейронної мережі. Вхідні дані x поступають у вхідний шар і потім поширюються у прихований шар під номером один. В цьому шарі виконується лінійне перетворення вхідних даних, до яких потім застосовуються певна функція активації, яка додає нелінійності у модель (хоча може використовуватись і лінійна функція активації). Аналогічним чином дані поширюються від одного до іншого прихованого шару. У вихідному шарі вже будується прогнозне значення y^*, яке має бути максимально близьким до оригінального.

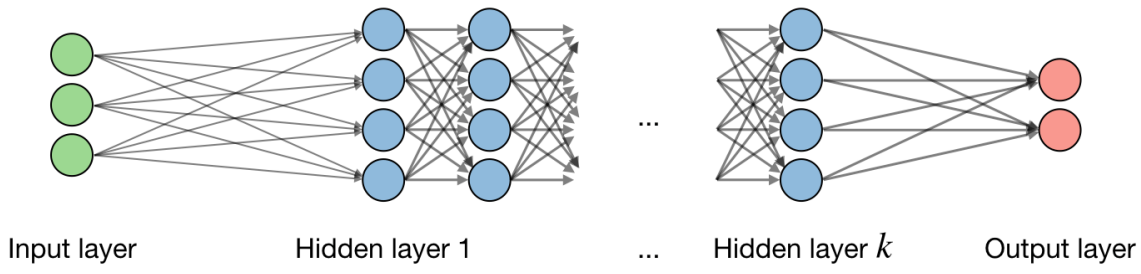


Рисунок 2.1 – Архітектура нейронної мережі прямого розповсюдження

Якщо позначити i – номер шару нейронної мережі, а j – нейрон прихованого шару, то тоді значення, яке надходить до функції активації можна представити наступним чином:

$$z_j^{[i]} = w_j^{[i]^T} x + b_j^{[i]},$$

де w, b, z – ваги, зміщення та результат лінійного перетворення відповідно.

Функція активації, в свою чергу обчислюється як:

$$o_j^{[i]} = g\left(z_j^{[i]}\right),$$

де $o_j^{[i]}$ – результат застовування функції активації;

g – функція активації.

2.1.2 Згорткові мережі

Наступним типом нейронних мереж, які будуть розглянуто – **згорткові нейронні мережі.** Вони найбільш широко використовується в задачах комп'ютерного зору. Проте, вони спеціалізовані для роботи з даними, що мають певну сітчасту структуру, в які також входять зображення з двумірної сіткою. Наприклад, часові ряди, які можна представити у вигляді одномірної сітки беручі навчальні приклади за певні інтервали [6].

Свою назву вони отримали завдяки математичній операції «згортки». Згортка – це особливий тип лінійної операції. Тому згорткові нейронні мережі можна визначити як звичайні нейронні мережі, які використовують операцію згортки замість перемноження матриць хоча б в одному зі своїх шарів.

Операція згортки зазвичай позначають зірочкою $(*)$ [6]:

$$s(t) = (x * w)(t)$$

Якщо розглядати дану операцію з точки зору нейронних мереж, то функція x – є вхідним значенням, а w – ядром. Результат операції іноді називають картою характеристик.

Треба зауважити, що насправді, в багатьох практичних задачах використовують не згортку в чистому вигляді, а пов’язану з нею функцію – крос-кореляцію. Крос-кореляція являє собою ту саму згортку, проте з невеликою особливістю відносно ядра, що дозволяє прискорити обчислення, які в глибокому навчанні займають досить багато часу.

Операції згортки має в собі обґрунтування, чому саме її так часто використовують, а саме: розріджені зв’язки, обмін параметрами та еквіваріентне відображення. Також, згортка надає можливість роботи з даними змінних розмірів.

Щоб пояснити розрідженні зв’язки, то повернемося до звичайних нейронних мереж прямого розповсюдження. Вони використовують множення матриці параметрів, в якій лежать значення, що пояснюють взаємодію кожного нейрону з кожним входом. В свою чергу, згорткові нейронні мережі мають розрідженні зв’язки. Це досягається тим, що використовують ядро менше за розмірі вхідних даних. Наприклад, зображення може мати мільйони пікселів, проте за допомогою згортки можна виявляти, певні особливості, такі як конктури на певних частинах зображення. Це значить, що нам вже треба менше пам’яті для зберігання параметрів і менше операцій для обчислення результату.

Обмін параметрами значить що один і той же параметр може використовуватися більше ніж в одній функції моделі. В традиційних нейронних мережах кожен елемент матриці ваг використовується лише один

для обчислення вихідного шару. В згорткових нейронних мережах кожен елемент ядра використовується на кожній позиції вхідних даних (крім крайових, що залежить від конкретної архітектури згорткового шару). Це значить, що ми замість того, щоб окремо навчати параметри для кожної локалії, ми робимо це тільки для певного набору параметрів, який визначається ядром. Це ніяк не впливає на час прямого проходу, проте зменшує кількість необхідної пам'яті для параметрів моделі.

Останньою особливістю згорткової нейронної мережі є еквіваріантність до переміщення. Функція є еквіваріантною в тому випадку, якщо при зміні вхідних даних, вихідні дані змінюються таким же чином. Наприклад, нехай I – функція, що робить зображення темніше на певних координатах. g – функція, що відображає одну функцію для зображення в іншу, таким чином, що $I' = g(I)$ – функція зображення з $I'(x, y) = I(x - 1, y)$ (зміщення кожного пікселі зображення вправо на одиницю). Тоді якщо ми застосуємо цю трансформацію до I, потім виконаємо згортку, то результат залишиться таким самим якби ми застосували згортку до I', потім застосували трансформацію g до виходу. Для обробки часових рядів це значить, що згортка створює своєрідну шкалу часу, яка показує, коли з'являються певні особливості у вхідних даних. Якщо ми змістимо подію пізніше у вхідних даних, то таке саме її представлення отримаємо на виході, але трохи пізніше [6].

Типовим шар згорткової нейронної мережі складається з трьох кроків. На першому кроці шар виконує декілька паралельних згорток, щоб побудувати набір лінійних активацій (Рис.2.2). На другому кроці кожна лінійна активація проходить через функцію активації. На цьому етапі зазвичай виявляються пені особливості. І останній крок використовує операцію субдискретизації, щоб модифікувати результат шару певним чином.

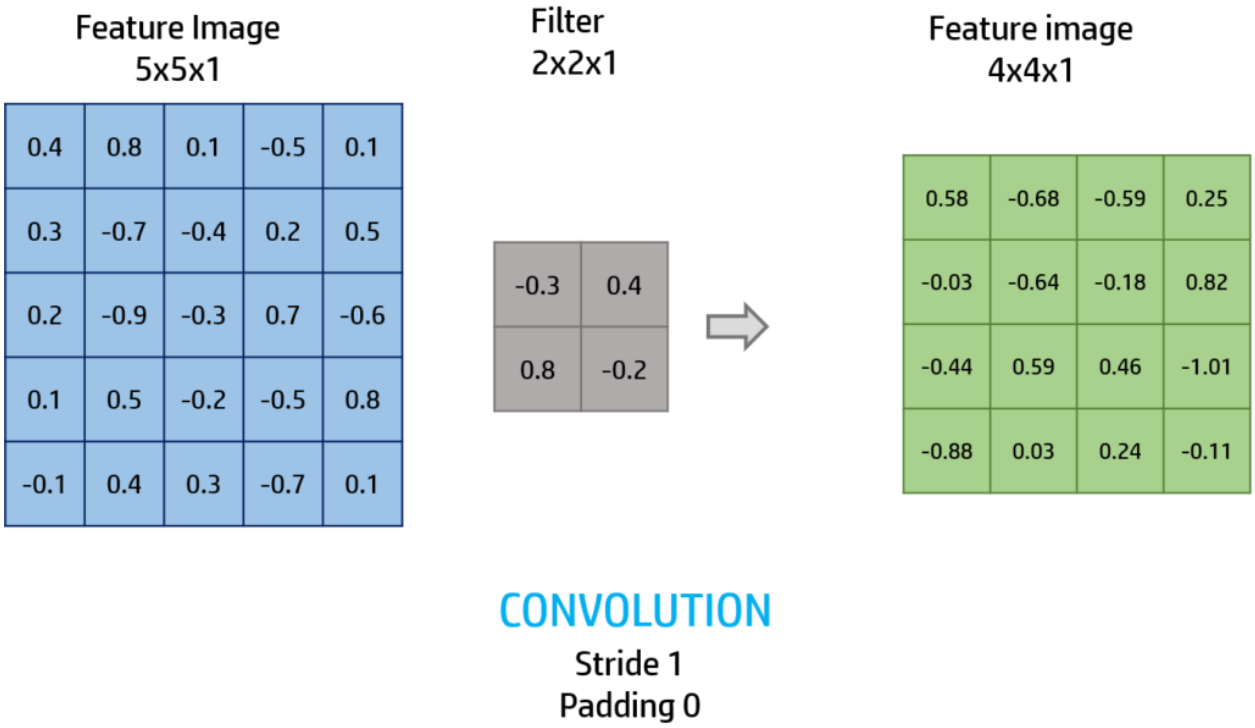


Рисунок 2.2 – Процес згортки зображення одним фільтро розміром 2 на 2, кроком рівним 1 і нульовим відступом

Функція субдискретизації замінює результат певного шару якось сумарної статистикою сусідніх виходів. Найбільш популярними статистиками є субдискретизація максимальних (Рис.2.3) та середніх значень.

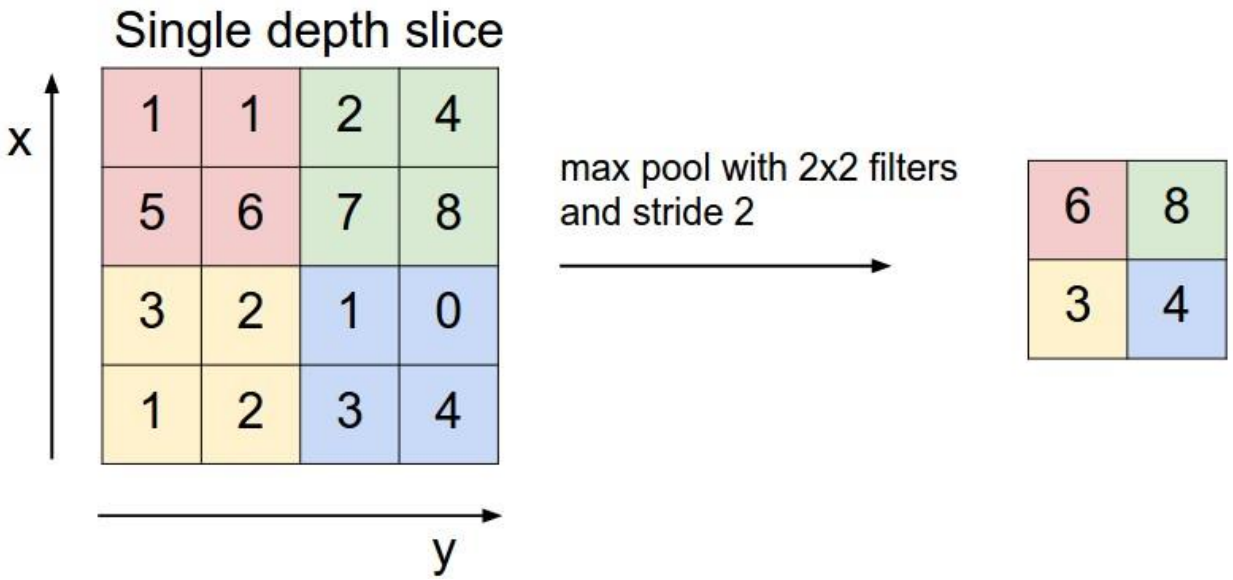


Рисунок 2.3 – Субдискретизація максимальних значень з фільтром 2 на 2 пікселі і кроком 2

Субдискретизація допомагає зробити відображення частково інваріантним відносно малих зміщень вхідних даних. Оскільки субдискретизація узагальнює результат всіх сусідніх значень, то можна використовувати ядро субдискретизації меншим за ядро згортки. Цей крок покращує швидкість обчислень мережі, тому що результатом цього кроку стануть дані меншої розмірності [6, 7].

2.1.3 Рекурентні мережі

Останнім типом нейромереж, які будуть розглянуті у цій роботі стануть **рекурентні нейронні мережі**. Це цілий клас нейронних мереж для обробки послідовних даних $x^{(1)}, \dots, x^{(t)}$. Так само як і згорткові нейронні мережі можуть легко масштабуватися до великих зображень та деякі з них працювати зі змінною розмірністю даних, так і рекурентні можуть масштабуватися до набагато більших послідовностей ніж звичайні нейронні мережі прямого розповсюдження. Також більшість з них можуть працювати з послідовностями різної довжини.

Рекурентні мережа також обмінюються параметрами як і згорткові мережі, проте механізм дещо інший. Кожен елемент результату обробки є функцією від минулого елемента. Також кожен елемент результату роботи прихованого шару рекурентної мережі створюється такими самими правилами як і попередній. Таким чином і виконується обмін параметрами у дуже глибокому обчислювальному графі.

На Рисунку 2.4 зображено структуру рекурентної нейронної мережі без вихідного шару [6]. Зазвичай в рекурентних мережах також присутній вихідний шар, який на основі значення обчисленого в прихованого шару робить певний прогноз.

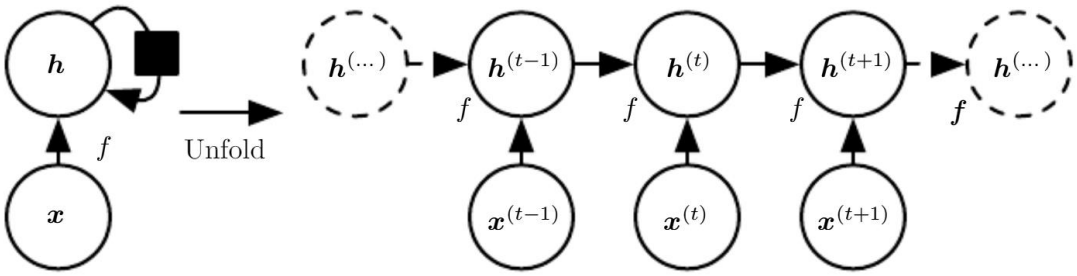


Рисунок 2.4 – Граф рекурентної нейронної мережі

Результати обчислень прихованого шару можна визначити наступним чином:

$$h^{(t)} = f\left(h^{(t-1)}, x^{(t)}, \theta\right),$$

де $h^{(t)}$ – результат обчислень елементу t;

$h^{(t-1)}$ – результат попереднього входу;

$x^{(t)}$ – елемент t вхідних даних;

θ – ваги прихованого шару.

Найбільш популярними видами рекурентних мереж можна назвати наступні:

Рекурентні мережі, що дають вихідне значення на кожному кроці і мають рекурентні зв’язки між прихованими елементами.

Рекурентні мережі, що дають вихідне значення на кожному кроці і мають рекурентні зв’язки лише в виходом на одному кроці з прихованим елементом наступного кроку.

Рекурентні мережі з рекурентними зв’язками між прихованими елементами, що зчитують повну послідовність і потім обчислюють єдине результуюче значення для всієї послідовності.

Пряме поширення даної нейронно мережі можна описати наступними рівняннями:

$$a^{(t)} = b + Wh^{(t-1)} + Ux^{(t)},$$
$$h^{(t)} = \sigma\left(a^{(t)}\right),$$
$$o^{(t)} = c + Vh^{(t)},$$
$$\hat{y}^{(t)} = softmax\left(o^{(t)}\right),$$

де $a^{(t)}$ – результат лінійної комбінації прихованого елементу на кроці $(t-1)$ та вхідного значення t;

$h^{(t)}$ – результат активації в прихованого шарі на кроці t;

$o^{(t)}$ – обчислення вихідного значення;

$\hat{y}^{(t)}$ – вихідне значення нейронної мережі;

b, c – параметри зміження;

U, V, W – матриці вагів;

σ – функція активації.

Серед недоліків даного класу нейронних мереж можна навести наступне:

– Досить довге обчислення.
– Складність доступу до старої інформації;
– Не беруться до уваги майбутні значення.

Також рекурентні нейронні мережі страждають від феномену, що носить назву вибухаючого та затухаючого градієнту. Причиною цього є те, що досить важко зрозуміти довготривалу залежність через перемноження градієнтів, які можуть збільшуватися та зменшуватися з експоненційною швидкістю відповідно кількості шарів.

Аби подолати проблему вибухаючого градієнту використовують спеціальну техніку – відсікання градієнту. Шляхом обмеження максимального значення градієнту, цю проблему можна контролювати.

Для другої проблеми пов’язаною з затухаючим градієнтом вводять різні типи воріт. Вони зазвичай позначаються як Г і мають наступний вигляд:

$$\Gamma = \sigma\left(Wx^{(t)} + Ua^{(t-1)} + b\right),$$

де W, U, b – коефіцієнти конретних «воріт»;

σ – сигмоїда.

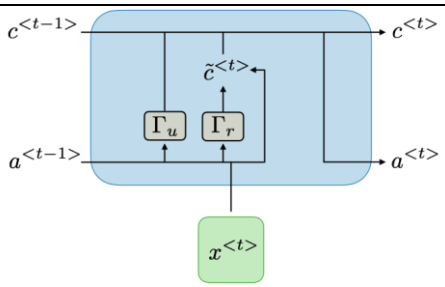
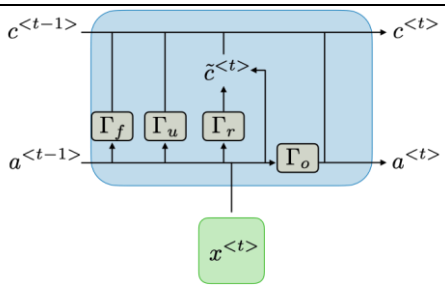
В Таблиці 2.1 один наведені основні типи «воріт» [5].

Таблиця 2.1 – Різні типи воріт з їх роллю та місцем використання

Тип	**Роль**	**Використання**
Ворота оновлення Γ_u	Наскільки сильно минулі значення повинні мати вплив зараз?	GRU, LSTM
Ворота актуальності Γ_r	Чи відкинути минулу інформацію?	GRU, LSTM
Ворота забуття Γ_f	Видаляти секцію чи ні?	LSTM
Вихідні ворота Γ_o	Наскільки розкривати значення?	LSTM

Найбільш популярними блоками в рекурентних нейронних мереж є блок GRU та LSTM. Вони борються з проблемами зникаючих градієнтів, що виникають в традиційних рекурентних мережах. Також LSTM – це більш узагальнена версія GRU. В Таблиці 2.2 наведені основні рівняння, що описують кожну архітектуру [5].

Таблица 2.2 – Основні блоки рекурентних мереж. ⋆ - позначає по елементне множення векторів

Характеристика	GRU	LSTM
$\tilde{c}^{(t)}$	$\tanh\left(W_c\left[\Gamma_r \star a^{(t-1)}, x^{(t)}\right] + b_c\right)$	$\tanh\left(W_c\left[\Gamma_r \star a^{(t-1)}, x^{(t)}\right] + b_c\right)$
$c^{(t)}$	$\Gamma_u \star \tilde{c}^{(t)} + (1 - \Gamma_u) \star c^{(t-1)}$	$\Gamma_u \star \tilde{c}^{(t)} + \Gamma_f \star c^{(t-1)}$
$a^{(t)}$	$c^{(t)}$	$\Gamma_o \star c^{(t)}$
Структура		

2.2 Функції активації

Вибір функції активації нейронної мережі має величезний вплив на динаміку навчання і продуктивність задачі. Довгий час ReLU була по замовчуванню найкраща серед функцій активації в глибокому навчанні. Але нещодавно було представлено нову – Swish – яка показує досить конкурентні результати.

Функції активації мають довгу історію. На початку, використовувалася сигмоїда через її похідну, діапазон значень від 0 до 1 та згладжену вірогіднісну форму. Її вигляд був наступний:

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

Її альтернативою був гіперболічний тангенс, значення якого лежали в інтервалі в -1 до 1:

$$\tanh(x) = \frac{sh\ x}{ch\ x} = \frac{e^x - e^{-x}}{e^x + e^{-x}} = \frac{e^{2x} + 1}{e^{2x} - 1}$$

Але класичні функції активації замінила ReLU (Рис. 2.5). Її простота і ефективність підштовхнула її розвиток і пов’язаних з нею Leaky ReLU та параметризованою ReLU.

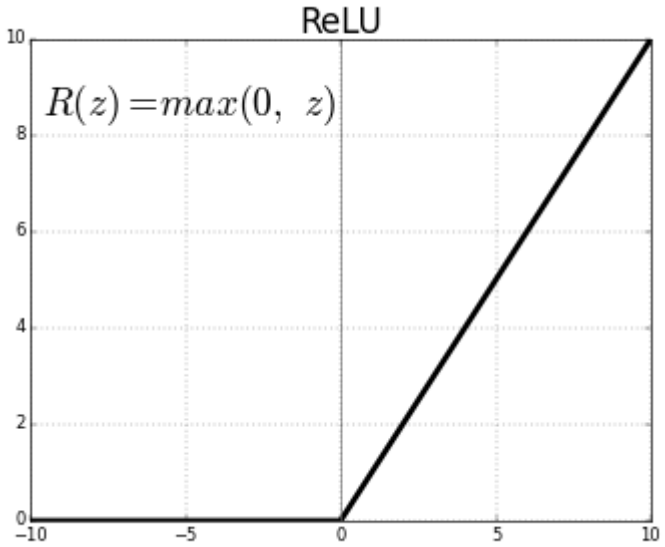


Рисунок 2.5 – Графік функції ReLU

Нещодавно представлена Swish [8] показала досить гарні результати. На Рисунку 2.6 зображено її графік.

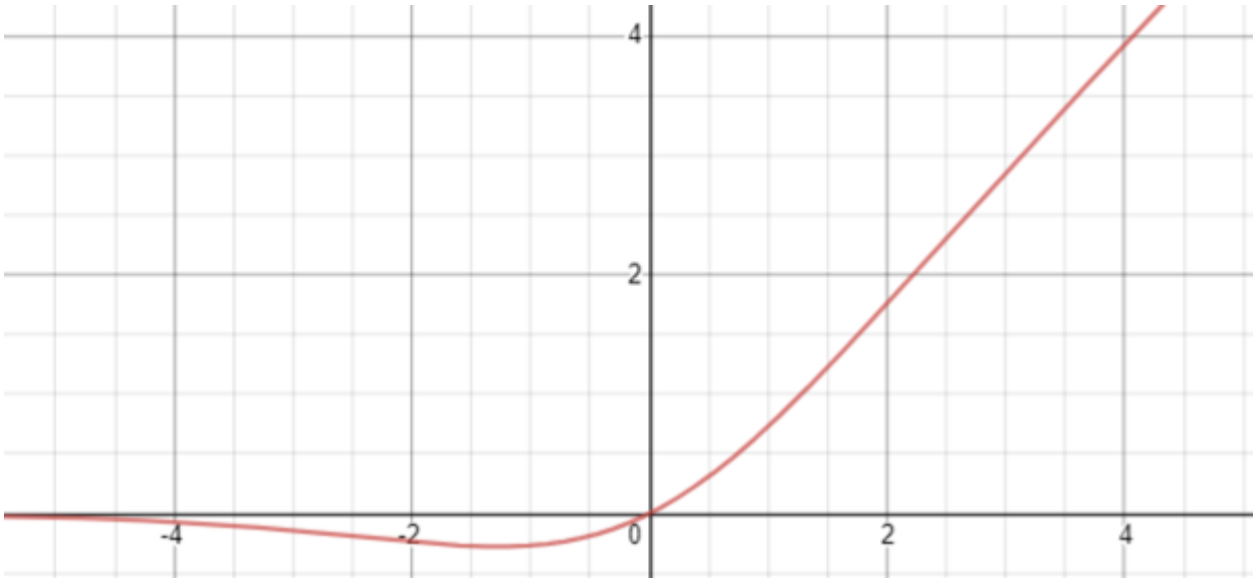


Рисунок 2.6 – Графік функції Swish

Swish можна представити наступним рівнянням:

$$f(x) = x \cdot \sigma(x) = \frac{x}{1 + e^{-x}}$$

Так само як і ReLU, Swish обмежена знизу, але необмежена зверху. Тим не менш вона згладжена в порівнянні з ReLU. Ще однією її особливістю є те, що вона немонотонна. Це значить, що вона має від'ємну похідну в одних точках, та позитивну в інших, навідміну від сигмоїду, у якій всі похідні додатні.

Похідна Swish має наступний вигляд:

$$f'(x) = f(x) + \sigma(x)\big(1 - f(x)\big),$$

де $f(x)$ – функція активації Swish;

$\sigma(x)$ – сигмоїда.

На Рисунку 2.7 зображено графіки першої та другої похідних Swish.

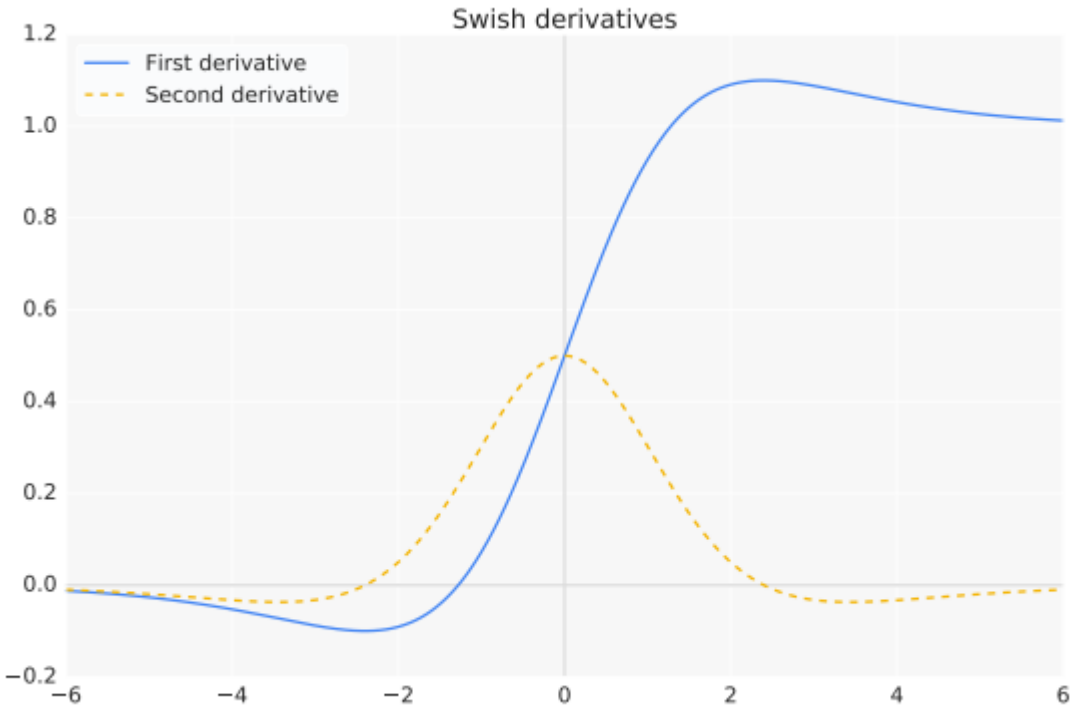


Рисунок 2.7 – Графіки першої та другої похідних функції Swish

Необмеженість – це одна з бажаних властивостей для функцій активації тому що вона дозволяє уникати повільного навчання для градієнтів зі значенням близьким до нуля. Такі функції як сигмоїда чи гіперболічний тангенс обмежені знизу та зверху, тому нейронні мережі з ними необхідно уважно ініціалізувати, щоб лишитися в рамках цих функцій активацій. В даному випадку ReLU є покращенням гіперболічного тангенсу, який обмежений зверху.

Обмеження знизу може бути корисним через сильну регуляризацію, оскільки функції, у яких $\lim_{x\to-\infty} f(x) = 0$, виконують певну регуляризацію шляхом відкидання дуже великих негативних значень. Це є досить важливим на момент початку тренування, коли великі від'ємні активації є досить частими.

Ці межі задовольняються такими функціями активації як Softplus, ReLU, Swish, проте не монотонність останньої збільшує «виразність» входу і покращує поведінку градієнтів.

Також, згладженність допомагає оптимізувати та узагальнити нейронну мережу. На Рисунку 2.8 зображено вихідні ландшафти випадкової нейронної мережі з різними функціями активації. Ландшафт нейронної мережі з ReLU має багато «гострих» регіонів, потенційно через її незгладжену природу. У порівнянні з цим, результуючий ландшафт неронної мережі з Swish набагато гладший. Цей ландшафт напряму впливає на згладженість ландшафту функції втрат. Інтуітивно, більше згладжену функція втрат легше оптимізувати через те, що по ній легше проходити, зменшуючи чутливість до ініціалізації вагів і швидкості навчання [8].

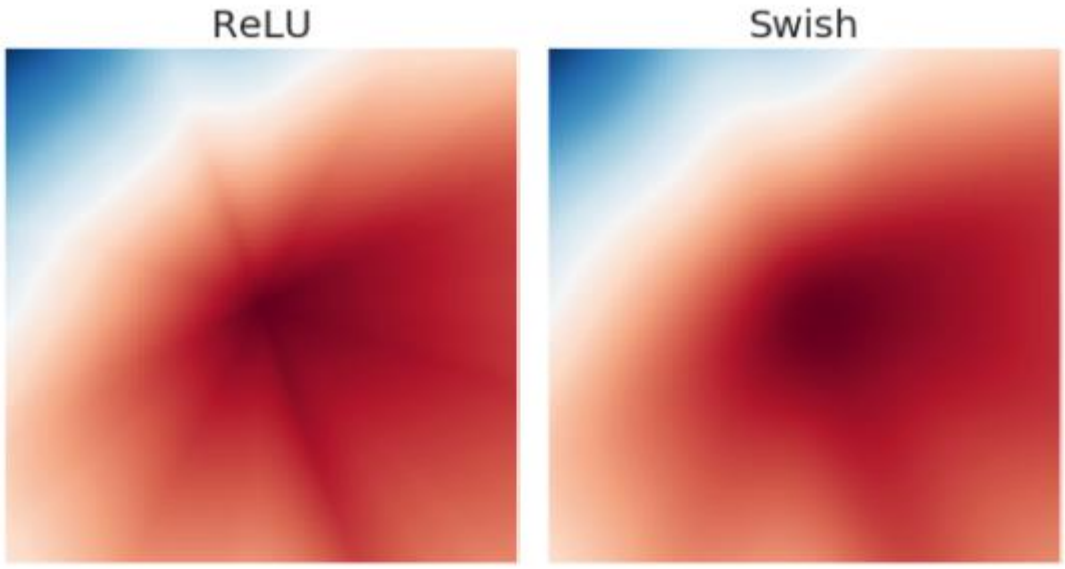


Рисунок 2.8 – Ландшафти функцій активації ReLU та Swish

2.3 Методи оптимізації нейронних мереж

Серед багатьох проблем оптимізації, однією з найскладніших є навчання глибоких нейронних мереж. Досить часто витрачається від декількох днів до місяців на десятках машинах, щоб навчити одну глибоку нейронну мережу.

Оскільки ця проблема залишається досить актуальною і дорогою, то були розроблені спеціальні техніки оптимізації для цього.

2.3.1 Градієнтний спуск

Градієнтний спуск – базовий, але найпоширеніший алгоритм оптимізації. Він часто використовується в лінійній регресії та алгоритмах класифікації. Зворотне поширення помилок в нейронних мережах також застосовує цей алгоритм.

Градієнтний спуск є алгоритмом оптимізації першого порядку, тобто він залежить від першої похідної функції втрат. Він обчислює те в якому напрямку ваги повинні змінюватися, щоб функція досягнула мінімуму. Під час зворотного поширення, втрати поширюються з одного шару в інший і параметри моделі змінюються відповідно таким чином, щоб функця втрат зменшувалася.

Правило оновлення виглядає досить просто [5, 6, 9]:

$$\theta^{(t)} = \theta^{(t-1)} - \alpha \nabla J\left(\theta^{(t-1)}\right),$$

де $\theta^{(t)}$ – ваги нейронної мережі;

α – гіперпараметр швидкості навчання;

∇J – градієнт функції втрат.

Переваги:

1. Прості обчислення.
2. Просто імплементувати.
3. Легкий для розуміння.

Недоліки:

1. Може попадати в пастку локального мінімуму.
2. Ваги змінюються після обчислення градієнту на всьому наборі даних. Тому, якщо дані дуже великі, то це може займати дуже багато часу, поки алгоритм знайде мінімум.

Великі вимоги до пам’яті під час обрахунку градієнту на всьому наборі даних.

2.3.2 Стохастичний градієнтний спуск

Першим варіантом градієнтного спуску є **стохастичний градієнтний спуск (SGD)**. Він оновлює параметри моделі частіше. Таким чином, ваги змінюються після обчислення функції втрат на кожному тренувальному прикладі.

Правило оновлення виглядає наступним чином [6, 7]:

$$\theta^{(t)} = \theta^{(t-1)} - \alpha \nabla J\left(\theta^{(t-1)}; x_i; y_i\right),$$

де x_i, y_i – тренувальні приклади.

Оскільки параметри моделі оновлюються частіше, то вони мають більшу дисперсію і присутнє коливання значення функції втрат з різною інтенсивністю.

Переваги:

1. Частіші оновлення ваг призводять до швидшого сходження алгоритму.
2. Необхідно менше пам’яті і немає необхідності зберігати значення функції втрат.
3. Може справлятися з пасткою локального мінімуму.

Недоліки:

1. Висока дисперсія параметрів моделі.
2. Після досягнення глобального мінімуму може зійтись до іншого значення.
3. Для того, аби отримати таку саму збіжність як і градієнтний спуск, треба повільно зменшувати швидкість навчання.

2.3.3 Міні-пакетний градієнтний спуск

Серед алгоритмів градієнтного спуску, найкращим є **міні-пакетний градієнтний спуск**. Він є покращенням стохастичного і звичайного градієнтного спуску. Оновлення вагів відбувається після кожного пакету даних. Таким чином, набір даних ділиться на пакети і після проходу кожного пакету параметри оновлюються.

Правило оновлення виглядає наступним чином [9]:

$$\theta^{(t)} = \theta^{(t-1)} - \alpha \nabla J\left(\theta^{(t-1)}; B_i\right),$$

де B_i – пакет тренувальних прикладів.

Переваги:

1. Часте оновлення параметрів моделі і менша їх дисперсія.
2. Середні вимоги до пам’яті.

Серед усіх алгоритмів градієнтного спуску можна визначити наступні складності:

– Визначення оптимального значення швидкості навчання. Якщо швидкість навчання дуже мала, то навчання буде дуже повільним,

в той час як при великій швидкості навчання алгоритм буде розходитись.

– Однакова швидкість навчання для всіх параметрів моделі. Може бути ситуація, коли деякі з параметрів вже не треба оновлювати.
– Проблема локального мінімуму.

2.3.4 Градієнтний спуск з моментом

Для того аби зменшити велику дисперсію і пом’якшити збіжність стохастичного градієнтного спуску було розроблено **момент**. Він прискорює збіжність у правильному напрямку і зменшує коливання в сторону неправильного. В цьому методі додається ще один гіперпараметр під назвою «момент»:

$$V^{(t)} = \gamma V^{(t-1)} + \alpha \nabla J\left(\theta^{(t-1)}\right),$$

де γ – гіперпараметр «момент».

Тоді правило оновлення приймає наступного вигляду [5, 9]:

$$\theta^{(t)} = \theta^{(t-1)} - V^{(t)}$$

Зазвичай гіперпараметр моменту встановлюють рівним до 0.9 або близьким до нього.

Переваги:

1. Зменшує коливання і дисперсію параметрів моделі.
2. Швидше сходиться ніж градієнтний спуск.

Недоліки:

1. Додається ще один гіперпараметр, який треба вручну і дуже уважно підбирати.

2.3.5. Прискорений градієнт Нестерова

Моментум може бути гарним методом для оптимізації, але якщо гіперпараметр моменту дуже великий, то алгоритм проскочить локальний мінімум і продовжить пошук. Щоб подолати цю проблему було розроблено **прискорений градієнт Нестерова**. Він думає наперед. Оскільки відомо, що алгоритм буде використовувати значення $\gamma V^{(t-1)}$ для оновлення ваг, то ми можемо апроксимувати майбутнє значення за допомогою $\theta^{(t-1)} - \gamma V^{(t-1)}$. Таким чином, функція втрат розраховується на основі цього майбутнього параметру, а не теперішнього:

$$V^{(t)} = \gamma V^{(t-1)} + \alpha \nabla\left(\theta^{(t-1)} - \gamma V^{(t-1)}\right)$$

Потім оновлюємо ваги використовуючи наступне правило:

$$\theta^{(t)} = \theta^{(t-1)} - V^{(t)}$$

Переваги:

1. Не пропускає локальний мінімум.
2. Сповільнюється, якщо мають місце мінімум.

Недоліки:

1. Гіперпараметри необхідно обирати вручну.

2.3.6 AdaGrad

Один з недоліків розглянутих алгоритмів є те, що швидкість навчання однакова для всіх параметрів моделі. **Оптимізатор AdaGrad** змінює швидкість навчання. Він змінює швидкість навчання η для кожного параметру і на кожному кроці t. Це алгоритм оптимізації другого порядку. Він працює з похідною функції помилок [9]:

$$g_i^{(t-1)} = \nabla J(\theta_i^{(t-1)})$$

Тоді оновлення параметрів виконується за наступним правилом:

$$\theta_i^{(t)} = \theta_i^{(t-1)} - \frac{\eta}{\sqrt{G_{ii}^{(t-1)} + \epsilon}} g_i^{(t-1)},$$

де η – швидкість навчання, яка змінюється в залежності від параметру $\theta_i^{(t-1)}$ на основі минулих градієнтів;

$G_{ii}^{(t-1)}$ – сума квадратів градієнтів $\theta_i^{(t-1)}$;

ϵ – елемент, який дозволяє уникати ділення ну нуль (зазвичай близький до $1e-8$).

Алгоритм робить великі оновлення для менш частих параметрів і маленькі – для частіших.

Переваги:

1. Швидкість навчання окремо змінюється для кожного параметра.
2. Немає необхідності в ручному регулюванні швидкості навчання.
3. Здатний тренуватися на розріджених даних.

Недоліки:

1. Обчислювально важкий, оскільки треба обчислювати похідну другого порядку.
2. Швидкість навчання постійно зменшується, що приводить до повільного навчання.

2.3.7 AdaDelta

Як розширення алгоритму AdaGrad було розроблено **AdaDelta**. Він боровся з проблемою затухаючої швидкості навчання. Замість того, щоб акумулювати квадрати минулих градінтів, AdaDelta обмежує вікно минулих градієнтів до фіксованого значення w. В такому випадку використовується експоненційне зважене середнє замість сума всіх градієнтів:

$$E[g^2]^{(t)} = \gamma E[g^2]^{(t-1)} + (1-\gamma)[g^2]^{(t)},$$
$$\theta^{(t+1)} = \theta^{(t)} - \frac{\eta}{\sqrt{E[g^2]^{(t)} + \epsilon}} g^{(t)}$$

Переваги:

1. Гіперпараметр швидкості навчання не затухає і тренування не зупиняється.

Недоліки:

1. Вимагає великих обчислень.

2.3.8 Adam

Останнім алгоритмом оптимізації, який буде розглянуто стане **Adam** [5, 6, 9]. Він працює з моментами першого і другого порядку. Його ідея полягає в наступному: ми не хочемо дуже швидко рухатися, оскільки можемо перестрибнути мінімум, ми хочемо зменшувати швидкість трохи, щоб проводити більш уважний пошук. В додаток до того, щоб зберігати експоненційно зважене середнє квадратів минулих градієнтів як це робить AdaDelta, Adam також зберігає експоненційно зважене середнє минулих значень градієнтів моменту.

$$\widehat{m}^{(t)} = \frac{m^{(t)}}{1 - \beta_1^{(t)}},$$

$$\hat{v}^{(t)} = \frac{v^{(t)}}{1 - \beta_2^{(t)}},$$

де β_1, β_2 – гіперпараметри методу;

$m^{(t)}$ і $Vv^{(t)}$ – це значення першого і другого моментів, які ,в свою чергу, є відповідно середнім і невідцентрованою дисперсією градієнтів.

Оновлення параметрів тепер виглядає таким чином:

$$\theta^{(t+1)} = \theta^{(t)} - \frac{\eta}{\sqrt{\hat{v}^{(t)}} + \epsilon} \widehat{m}^{(t)}$$

Переваги:

1. Метод дуже швидкий і дуже швидко збігається.
2. Долає проблему затухаючої швидкості навчання і високої дисперсії.

Недоліки:

1. Вимагає багато обчислень.

2.3 Пошук гіперпараметрів нейронних мереж

Більшість алгоритмів глибоко навчання мають декілька гіперпараметрів, які контролюють поведінку алгоритму. Деякі з них впливають на час і вимоги до пам’яті алгоритму, а деякі – на якість моделі під час навчання і їх здібність до виведення правильних результатів на нових даних.

Існує два загальних принципа підбору гіперпараметрів [6]: обирати їх вручну чи автоматично. Для того аби підбирати їх вручну необхідно розуміти на що саме гіперпараметр впливає і як модель машинного навчання досягає висого рівня генералізації. Автоматичний підбір сильно зменшує необхідність в розумінні цих ідей, проте зазвичай вони вимагають набагато більше обчислень.

Ідеальний алгоритм навчання повинен просто брати набір даних і давати функцію, без необхідності підбору гіперпараметрів [6]. Популярність багатьох алгоритмів навчання таких як логістична регресія і метод опорних векторів йде від того, що для них досить підбору одного чи двох гіперпараметрів. Іноді нейронні мережі також можуть показувати гарні результати з малою кількістю змінних гіперпараметрів, проте зазвичай гарні результати досягаються шляхом варіювання великою кількості таких.

Якщо розглядати логіку того, як користувач навчального алгоритму шукає гарні значення гіперпараметрів, то стає зрозуміло, що тут має місце оптимізація: шукаються значення гіперпараметрів, які оптимізуються певну цільову функцію, іноді з якимись обмеженнями на час, пам’ять та ін. Тому в теорії можна розробити алгоритм оптимізації гіперпараметрів, який обирає

алгоритм навчання і його гіперпараметри, таким чином приховуючи їх пошук від користувача. Проте, такі алгоритми зазвичай мають свої власні гіперпараметри, такі як діапазон значень, що треба перевірити. Вибір цих гіперпараметрів, як правило, простіший в тому сенсі, що обранні вторинні гіперпараметри можна застосовувати до широкого спектру задач.

Коли в задачі невелика кількість гіперпараметрів, то на практиці зазвичай використовують пошук по сітці. Для кожного гіперпараметру, користувач обирає невеликий скінченний набір значень, які треба дослідити. Потім алгоритм пошуку по сітці навчає модель для кожного набору значень із декартового добутку обраних наборів. Експеримент, який показав найкраще значення помилки на валідаційній вибірці відбирає модель з відповідним набором найкращих гіперпараметрів. На Рисунку 2.9 проілюстровано сітка значень гіперпараметрів моделі [6, 10].

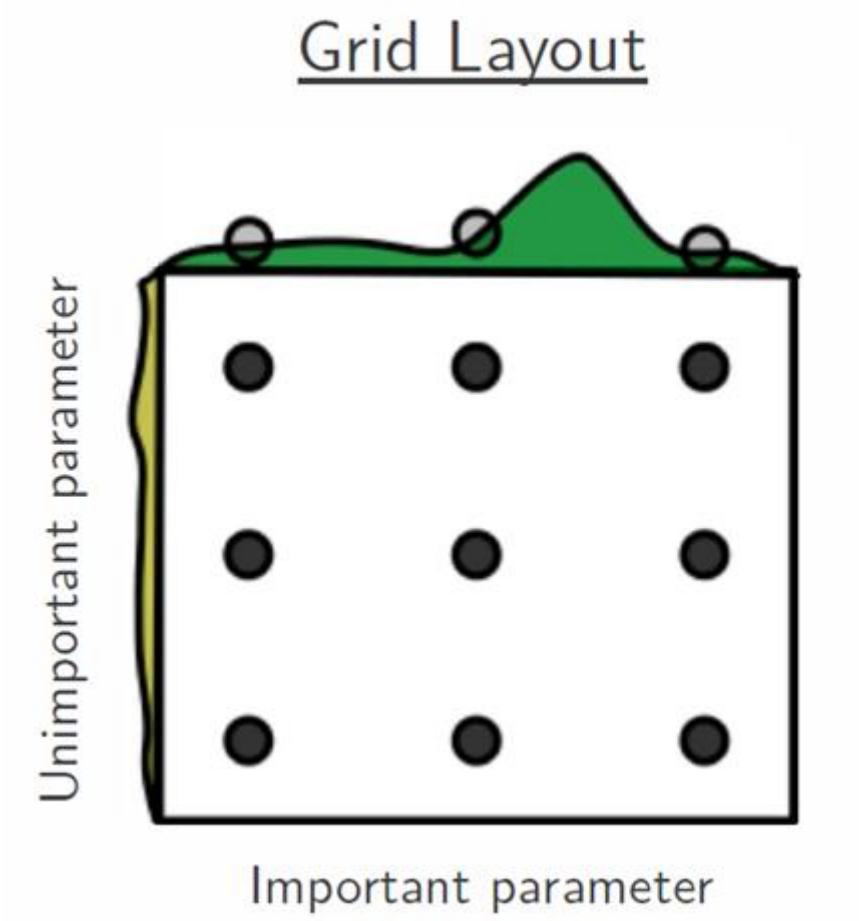


Рисунок 2.9 – Пошук по сітці

У випадку з чисельними (порядковими) гіперпараметрами, обирається найбільше і найменше значення зі списку, на основі досвіду зі схожими експериментами, щоб бути впевненими, що оптимальне значення лежить в обраному діапазоні. Зазвичай, у пошуці по сітці використовують значення логарифмічного масштабу.

Пошук по сітці показує найкращі результати, коли він застосовується повторно. Наприклад, нехай було зроблено пошук по сітці для гіперпараметру α використовуючи значення $\{-1, 0, 1\}$. Якщо найкраще знайдене значення було 1, тоді ми недооцінили діапазон, в якому лежить найкраще значення α. Тоді ми зміщуємо сітку і робимо новий пошук по сітці для α, наприклад для значень $\{1, 2, 3\}$. Якщо б найкраще значення α було 0, тоді можна були покращити нашу оцінку діапазона шляхом його масштабування до пошуку в $\{-0.1, 0, 0.1\}$.

Очевидною проблемою даного алгоритму є його обчислювальні вимоги. Вони ростуть експоненційно від кількості параметрів. Для пришвидшення алгоритму можна проводу пошук паралельно на декількох комп’ютерах.

Висновки за розділом 2

Даний розділ розкриває основні архітектури глибоких нейронних мереж, що використовуються для прогнозування фінансових даних. Було запропоновано для використання відносно нову функцію активації Swish, що показує гарні результати. Серед алгоритмів оптимізації було проведено детальний опис їх роботи та наведено недоліки з перевагами.

РОЗДІЛ 3

МЕТОДИКА ПІДБОРУ АРХІТЕКТУРИ ТА ГІПЕРПАРАМЕТРІВ НЕЙРОННОЇ МЕРЕЖІ ДЛЯ ПРОГНОЗУВАННЯ ФІНАНСОВИХ РИНКІВ

3.1 Етапи алгоритму

У роботі пропонується алгоритм для підбору архітектури та гіперпараметрів нейронної мережі з метою досягнення найвищої точності прогнозування часового ряду, який включає наступні етапи:

1. Економетричний аналіз фінансових даних. Побудова гіпотези про розмір вікна, який приймає нейронна мережа на вхід на основі знайденого порядку авторегресії.
2. Відбір найкращих трьох наборів гіперпараметрів для різних типів нейронних мереж методом Grid Search, модифікованим під особливості часових рядів.
3. Навчання обраних моделей на більшій кількості епох, для досягнення найкращих результатів.
4. Відбір найкращої моделі кожного типу.
5. Максимально звужений і чутливий підбір гіперпараметрів нейронних мереж та їх оптимізаторів для досягнення максимальної точності на тестовій вибірці.
6. Відбір найкращої моделі на валідаційній вибірці, яка не була включена у навчання та підбір гіперпараметрів, на основі різних метрик, що мають найбільше значення в контексті обраної задачі.

Детально розглянемо кожен з цих етапів.

На першому етапі алгоритму виконується економетричний аналіз даних, будуються авторегресійні моделі, перевіряються гетероскедастичність рядів, та інші тести. Цей етап переслідує наступні цілі:

Якщо авторегресійні чи інші економетричні моделі показують чудовий результат, то вже немає необхідності в побудові більш складних як за архітектурою, так і за обчисленнями нейронних мереж.

Ми отримуємо деякі проміжні результати, які в майбутньому можна використати при побудові нейронних мереж.

При першому результаті подій наш алгоритм закінчується і ми отримуємо економетричну модель для прогнозування часового ряду, яка набагато легша в обчисленнях і краща для розуміння, ніж нейронні мережі. При другому варіанті, результати побудованих авторегресійних моделей пропонується використовувати для підбору архітектури мережі та її гіперпараметрів. Для цього результати отриманих моделей можна використати як базовий (таким же чином як ковзне середнє використовують як саму базову модель) підхід, який є індикатором того, що підібрана архітектура або гіперпараметри є актуальними до обраної задачі. Також, такий параметр як порядок авторегресії можна інтерпретувати як кількість вхідних нейронів у нейронній мережі.

Для побудови авторегресії необхідно визначити її порядок. Для цього існує два різних підходи. Перший підхід полягає у тому, щоб побудувати графік частково автокореляційної функції (ЧАКФ) і визначити лаг, який суттєво не відрізняється від нуля. Номер цього лагу і буде порядком авторегресії. Другий підхід покладається на різні критерії, такі як Акайке або Байєса-Шварца. Для цього визначається певний максимальний порядок лагу, який теоретично може не відрізнятися сильно від нуля. Далі будуються моделі авторегресії з порядком від першого до максимально обраного лагу. Потім за допомогою обраного критерію обирається найкраща модель. В даному випадку використовуються саме критерії Акайке, Байєса-Шварца або подібні до них, а не звичайні метрики якості, оскільки вони крім результатів прогнозування враховують також складність моделі.

На основі автокореляційного аналізу робиться висновок про те, чи залежать певним чином наші майбутні значення часового ряду від своїх

попередніх значень. Оскільки на вхід нейронна мережа отримує минулі значення для прогнозування майбутнього значення, то є досить логічним, що якраз порядок авторегресії можна використати як оптимальний розмір вхідного вікна.

На другому етапі алгоритму виконується відбір найкращих наборів гіперпараметрів для різних типів нейронних мереж методом Grid Search.

Grid Search – один із найбільш популярних методів підбору гіперпараметрів, які використовуються в машинному навчанні [5, 6, 10]. Він полягає в тому, що задаються різні значення гіперапарметрів і потім будуються всі можливі їх комбінації. Потім кожен набір цих гіперпараметрів використовується у моделі. Для навчання моделі оригінальний ряд ділиться на певну кількість частин і по черзі кожна частина використовується як тестова, а всі інші виступають у ролі тренувальної. Таким чином зменшується вплив стохастичних процесів на моделі і досягається мінімальна дисперсія метрик якості.

Цей підхід є досить відомим у світі машинного навчання, але напряму його неможливо використовувати з часовими рядами. Оскільки часовий ряд має залежність в часі, то його не можна розбити на певну кількість частин, а потім почергово брати кожну з них як тестову, оскільки в такому випадку збивається часова змінна і такі особливості ряду, як циклічність і сезонність не враховуються. Тому його модифікують певним чином [10, 11].

Перша модифікація покладається на датасет. Якщо в даних присутня сезонність або циклічність, то тоді його розбивають таким чином, щоб всі ці особливості врахувати. Тобто в одній частині повинна враховуватися і сезонність і циклічність. Іншими словами, дані, що містяться у кожній з підмножин оригінального ряду повинні мати однаковий розподіл. Ще однією проблемою, яка може виникнути в даному підході є тренд. Якщо часовий ряд має тренд, то перед тим як використовувати дану модифікацію Grid Search, необхідно його позбутися. Оскільки це напряму впливає на магнітуду даних, а отже і на розподіл.

Друга модифікація методу Grid Search змінює сам процес розбиття ряду. Замість того, щоб ділити дані на N частин і потім по черзі брати одну з них як тестову, а інші – тренувальну, то роблять дещо по-іншому. Нехай в наборі даних міститься N записів. Тоді визначають максимальний розмір тестової вибірки M (або мінімальний розмір тренувальної вибірки). Після цього роблять наступне [10]:

1. Призначають тренувальній вибірці N – k точок.
2. Для тестової вибірки обирають всі інші дані.
3. Тренують модель на тренувальній вибірці.
4. Розраховують метрику якості на тестовій вибірці.
5. Заносять результати метрики в масив.

k є [1, M] – номер ітерації.

Таким чином ми навчаємо модель M раз поступово збільшуючи (або зменшуючи) розмір тестової вибірки на 1 і враховуємо порядок елементів навчального набору, що дуже важливо в часових рядах.

Отже, досягається та сама ціль, яку переслідує оригінальний Grid Search і враховується особливість часових рядів. Можна помітити, що даний підхід вимагає досить багато ітерацій навчання для оцінки якості моделі. Обмеженням методу є те, що він займає досить багато часу. Для подолання цього обмеження, для зменшення кількості тренувань можна, по-перше, збільшувати розмір тестової вибірки не на одиницю, а на більше число. Це зменшить час, необхідний на оцінку моделі, не сильно пошкоджуючи оцінку якості моделі. По-друге, приймається до уваги розмір аналізованого часового ряду. По-третє, при простій структурі даних, як в часових рядах, моделі нейронних мереж з дуже великою кількістю шарів буду перенавчатися, що буде призводити до поганих узагальнюючих властивостей побудованих моделей. Крім того, моделі, з великою кількістю параметрів, вимагають великої обчислювальної потужності. В нашому випадку при аналізі часового ряду немає сенсу їх використовувати.

До розгляду варто обрати декілька різних архітектур нейронних мереж: багатошарові прямого розповсюдження, згорткові, рекурентні мережі, та до кожної з них свою множину гіперпараметрів, які будемо варіювати. Всі моделі будемо тренувати на 100 епохах. Тричі будемо повторювати навчання для того, щоб мінімізувати вплив випадкових елементів.

В результаті цього кроку було протестовано велику кількість гіперпараметрів, виконано «широкий» підбір параметрів. Для кожної архітектури звужено множину наборів гіперпараметрів до трьох наборів, що показують найкращий результат, для їх використання на наступному етапі.

На третьому етапі виконується навчання обраних моделей на більшій кількості епох, для досягнення більш кращих значень метрик якості. Також, щоб зменшити вплив випадкових факторів збільшимо кількість експериментів. Таким чином отримаємо максимально точні оцінки результатів. Пропонується збільшити кількість епох до 300, а кількість експериментів до 10.

На четвертому етапі виконується відбір найкращої моделі кожного типу. Для цього можуть використовуватися наступні метрики [3, 10, 12]:

$$\mathrm{RMSE} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(\hat{y}_i - y_i)^2},$$

$$\mathrm{MAE} = \frac{1}{N}\sum_{i=1}^{N}|\hat{y}_i - y_i|,\ \mathrm{MAPE} = \frac{1}{N}\sum_{i=1}^{N}\frac{|\hat{y}_i - y_i|}{y_i} * 100\ (\%),$$

де вектор $\hat{y}$ містить значення прогнозу на основі моделі, y – вектор реальних значень вихідної змінної, а N – розмірність перевірочного набору даних.

Вибір метрики залежить від конкретної задачі і може відрізнятися від задачі до задачі. Також можна агрегувати значення за декількома метриками, використовуючи методи підтримки прийняття рішень [13] .

Після того як було відібрано найкращі моделі різних типів, на п’ятому етапі виконується звужене коригування гіперпараметрів. Оскільки основні

гіперпараметри вже обрано, на цьому етапі можна додати додаткові гіперпараметри, такі як функція активації нейронів, метод оптимізації, гіперпараметри, що відносяться до оптимізатору та інші. На цьому кроці проводиться більш ретельний підбір гіперпараметрів, який вже буде мати фінальна модель, для досягнення максимальної точності моделі на тестовій вибірці

На останньому шостому етапі проводиться відбір найкращої моделі на валідаційній вибірці, яка не була включена у навчання та ~~у~~—підбір гіперпараметрів на основі різних метрик, що мають найбільше значення в контексті обраної задачі.

3.2 Опис розробленого програмного продукту

Для виконнання дисертаційної роботи були написані програмі модулі на мові Python. Такий вибір мови пояснюється наступними факторами:

- велика спільнота;
- підтримка різних парадигм програмування;
- простота;
- великий стек технологій для побудови нейронних мереж.

Для побудови модулів були використані наступни бібліотеки:

- tensorflow, Keras – для побудови і навчання нейронних мереж;
- matplotlib – для побудови графічних елементів;
- statsmodels – для роботи з економетричними моделями.

Написані модулі реалізують описаний у роботі алгоритм пошуку архітектури глибокої нейронної мережі та її гіперпараметрів. В реалізації алгоритму пошуку по сітці реалізоване коригування самої сітки, що надає гнучкості алгоритму. Також, по-замовчуванню існують вже готові, побудовані

глибокі нейронні мережі (одношарова та двошарова мережа прямого розповсюдження, згорткова мережа і рекурентна мережа), на яких можна проводити досліди на різних типах даних, коригуючи при цьому сітку гіперпараметрів.

Також створено програмний модуль, який проводить економетричний аналіз даних, а саме: проводить різні статистичні тести (на стаціонарність, на належність до нормального розподілу, тощо), робить графічний аналіз, будуючи графіки та гістограми різних типів, для того, аби користувачу було інтуітивно зрозуміло результати статистичних тестів.

Для того, аби можна було порівнювати результати глибоких нейронних мереж з більш простими статистичними моделями, було розроблено програмний модуль, який будує різні авторегресійні моделі: AR, ARMA, ARIMA, ARCH, GARCH, EGARCH та ін.

3.3 Результати роботи алгоритму на вибраних часових рядах

На **першому етапі** виконується економетричний аналіз ряду. Він є важливим, тому що може вказати на особливості датасету, які можна в майбутньому використати для моделювання прогнозу.

З Рисунку 3.1 видно, що дані мають певний тренд, що може вказувати на те, що дані не стаціонарні.–Графіки адитивних і мультиплікативних змін свідчать про можливу стаціонарність заданого ряду.

Гістограма розподілу дозволяє виявити аномальні значення у наборі даних, а також оцінити розподіл вхідних даних. На Рисунку 3.2 зображено гістограму змін першого порядку.

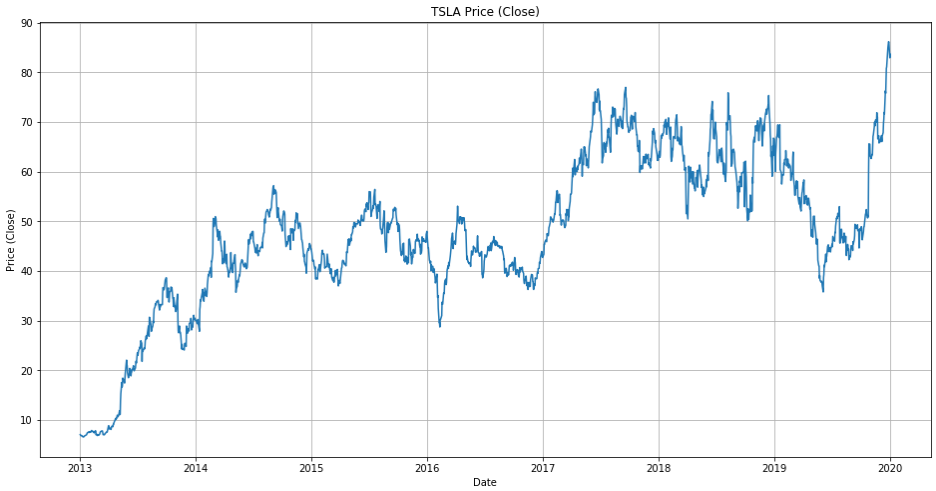


Рисунок 3.1 – Графік акцій компанії Tesla

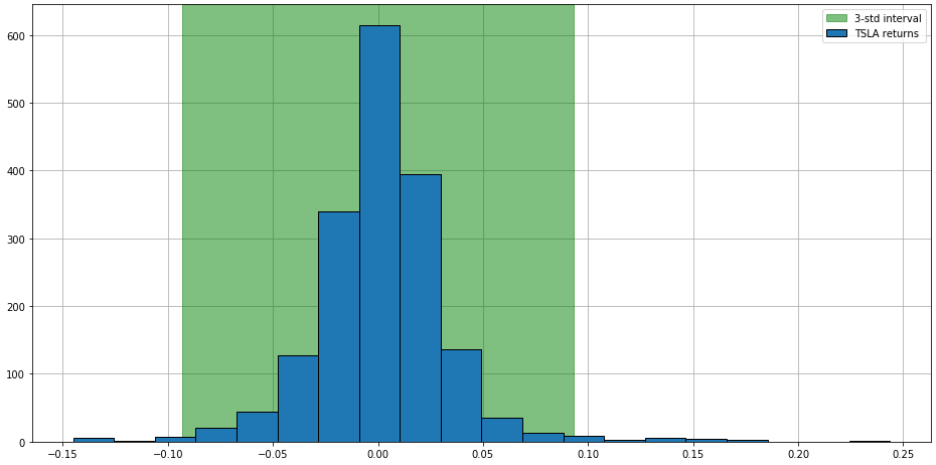


Рисунок 3.2 – Гістограма розподілу змін першого порядку і інтервал 3 стандартних відхилення

За допомогою гістрограми можна оцінити належність даних до нормального розподілу. На Рисунку 3.3 зображено гістограму наших даних поруч з графіком нормального розподілу, у якого середнє та дисперсія відповідають аналогічним значенням нашого ряду. З рисунку 3 видно, що дані не відповідають нормальному розподілу, тому проведемо статистичний тест Харке-Бера, який може свідчити, що дані взяті з нормального розподілу:

$$p - value = 0.0$$

Оскільки $p - value < 0.05$, то зміни цін на акції компанії Tesla скоріше за все не розподілені за нормальним законом.

Також важливо розуміти чи наш ряд є гетероскедастичними. Для цього скористаємося тестом Бройша-Пагана:

$$p - value = 0.03283$$

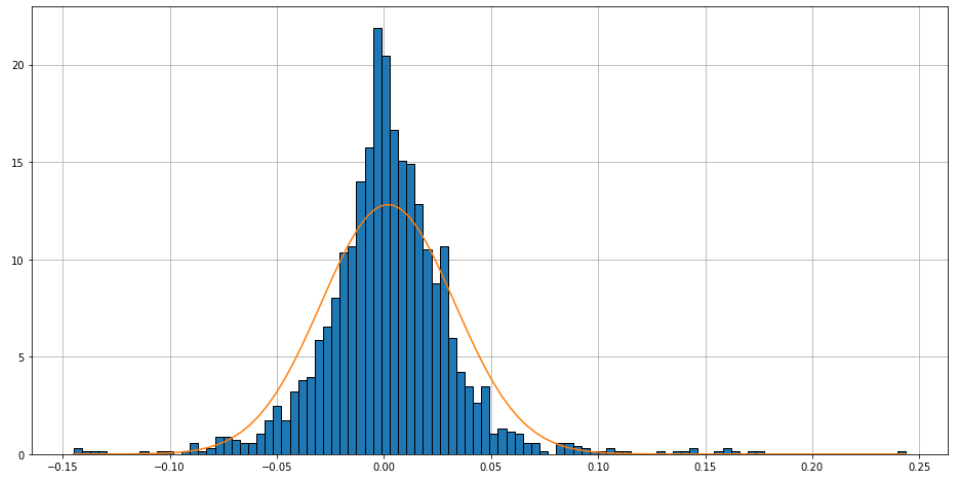


Рисунок 3.3 – Графік розподілу акцій TSLA разом з нормальним розподілом з аналогічним середнім та дисперсією

Таке p-value вказує на те, що дані ймовірніше за все є гетероскедастичними. Для того, аби позбутися гетероскедастичності, можна застосувати спеціальні трансформаці, такі як різницевий аналіз, логарифмічне перетворення або перетворення Бокса-Кокса.

Тепер перевіримо стаціонарність наших даних за допомогою тесту Адфулера:

$$p - value = 0.215639,$$

що означає нестаціонарність даних. Для того, аби зробити дані стаціонарними часто використовують диференціювання. І якщо ми

продиференціюємо наш ряд, а потім проведемо цей же тест, то побачимо наступне:

$$p - value = 0.0$$

Перед побудовою авторегресійних моделей побудуємо для нашого ряду графік ЧАКФ (Рис. 3.4).

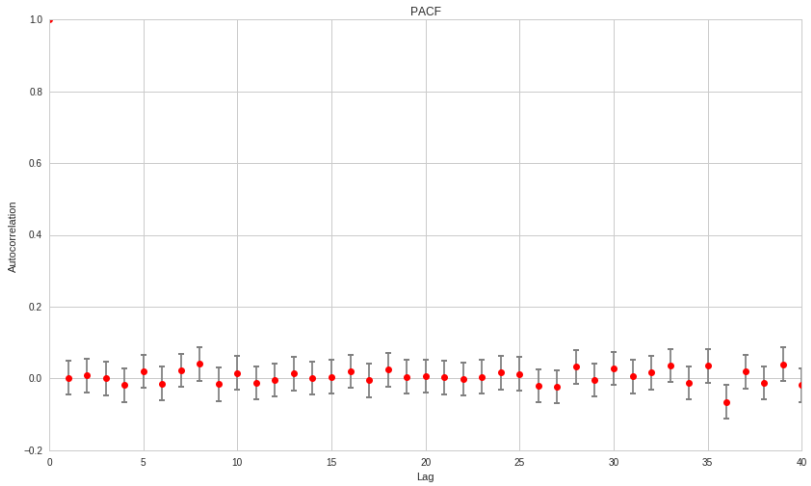


Рисунок 3.4 – Графік ЧАКФ для змін цін на акції TSLA

На Рисунку 3.4 видно, що значення автокореляції лежать в довірчих інтервалах і значно не відрізняються від нуля. Можна зауважити, що лаг під номером 8 близький до того, аби показувати певну автокореляцію, тому візьмемо його до увагу і повернемося до нього пізніше. Порядок авторегресії дорівнює нулю. В таблиці 3.1 та 3.2 продемонстровано результати моделювання.

Таблиця 3.1 – Результати моделювання

AIC	BIC	Durbin-Watson	R2
-7214.076	-7203.130	1.9959354088	0

Таблиця 3.2 – Параметри моделі

	coef	std err	[95% Conf. Int.]
const	0.0019	0.001	[0.000, 0.003]

Критерій Дурбіна-Уотсона близький до 2, що означає, що в даному ряді немає автокореляції першого порядку і тому це підтверджує наші початкові припущення. Значення коефіцієнту детермінації рівне 0 також досить легко пояснюється. Оскільки в нашому ряді відсутня автокореляція, то ми обрали модель, яка не бере до уваги минулі значення. Таким чином, навчання моделі зводиться до знаходження горизонтальної лінії сума відстаней від якої до кожної точки буде найменшою. Цією лінією буде середнє значення і тому наша константа і є середнім значенням. Оскільки наша модель завжди прогнозує середнє, а коефіцієнт детермінації порівнює якість моделі якраз відносно прогнозування простим середнім, то ці величини однакові і при діленні будуть рівні 1. І саме тому результат рівний 0.

Останнім кроком буде перевірка адекватності моделі. Модель адекватно працює, якщо її залишки поводять себе як білий шум. Для перевірки використаємо АКФ та статистику Льюнга-Бокса.

З графіка (Рис.3.5) видно, що наші залишки не автокорельовані. Підтвердимо це статистичними тестами Льюнга-Бокса та Бокса-Пірса.

p-values (Ljung-Box):

[0.93404204, 0.93978466, 0.98878085, 0.94571276, 0.9221237 ,
0.93677406, 0.90986584, 0.68368103, 0.72686949, 0.7585834 ,
0.80050874, 0.85625655, 0.87943405, 0.91736987, 0.94105545,
0.94105242, 0.95951661, 0.94791914, 0.96477885, 0.97539565,
0.98364152, 0.9894332 , 0.9928338 , 0.99383021, 0.99516186,
0.99558708, 0.99422771, 0.98740771, 0.99136843, 0.98708054,
0.99064132, 0.99168772, 0.97999314, 0.9828336 , 0.96894627,
0.83038249, 0.84119457, 0.86666844, 0.8213764 , 0.84423755]

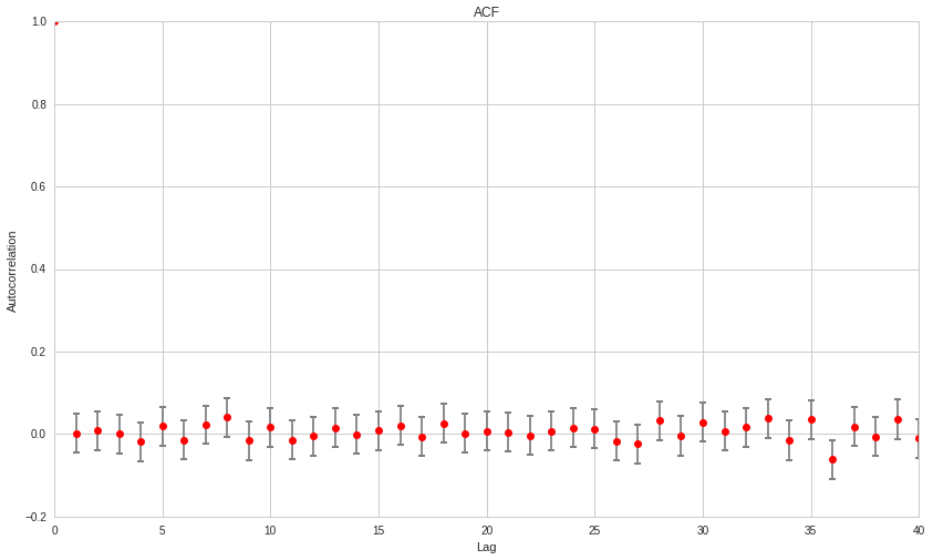


Рисунок 3.5 – АКФ для залишків

p-values(Box-Pierce):

[0.93409809, 0.93991534, 0.98881758, 0.94601916, 0.92268415,
0.93733588, 0.91081881, 0.68683128, 0.73001063, 0.76174596,
0.80349802, 0.85872965, 0.88176753, 0.91916612, 0.94248183,
0.94264336, 0.96071735, 0.94963256, 0.96604081, 0.9763519 ,
0.98432707, 0.9899083 , 0.99317975, 0.99415474, 0.99543631,
0.99585875, 0.9946175 , 0.98833224, 0.99203869, 0.98814456,
0.99145437, 0.99244883, 0.98189449, 0.98454405, 0.97209574,
0.8450087 , 0.85547317, 0.87940667, 0.83795979, 0.85951929]

Бачимо, що всі значення більші 0.05. Тому ми приймаємо нульова гіпотезу про те, що залишки не мають автокореляції.

Наступними кроками може бути побудова більш складних моделей авторегресії, таких як АРКС, АРІКС і т.д. Це покращить результати базового методу прогнозування і дасть змогу більше звузити множину гіперпарамтерів і архітектур нейронних мереж, що будемо будувати у майбутньому.

В нашому прикладі було отримано порядок авторегресії рівний нулю. Цей порядок було отримано на часовому ряді змін акцій, але оригінальний ряд напряму матиме залежність в 1 крок від попередніх значень. Також було виявлено 8 лаг, який можливо і має якусь значимість, тому його також використаємо для експерименту. За допомогою **цього** кроку **було зроблено**

висновок про розмір вхідного вікна для нейронної мережі (або кількість вхідних нейронів). Будемо розглядати два вікна: 1, 8.

На другому етапі алгоритму проводиться відбір найкращих трьох наборів гіперпараметрів для різних типів нейронних мереж методом Grid Search, модифікованим під особливості часових рядів. Заданий часовий ряд невеликий за розміром, містить всього 1762 записи, тому будемо використовувати для нього другу модифікацію Grid Search: вона буде ефективною, оскільки навчання пройде досить швидко.

Для аналізу заданого ряду розглянемо 3 різних архітектури нейромережі і 1 частковий випадок багатошарової мережі прямого розповсюдження, в якій лише один прихований шар та до кожної з них своя множина гіперпараметрів, які будемо варіювати. Всі моделі будемо тренувати на 100 епохах. Тричі повторюючи для мінімізації впливу випадкових елементів.

Перша архітектура, результати якої буде продемонстровано – одношарова нейронна мережа (Рис.3.6). Для цієї моделі обрано гіперпараметри:

1. Розмір вхідного вікна: 1, 8.
2. Кількість нейронів у прихованому шарі: 64, 128.
3. Розмір батчів: 1, 32.
4. Порядок диференціювання: 0, 1, 8.

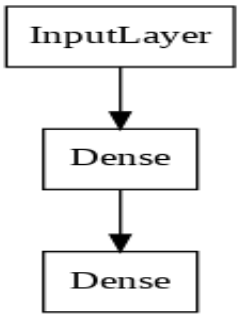


Рисунок 3.6 – Одношарова нейронна мережа

Всього 24 параметри. В Таблиці 3.3 продемонстровано результати тренування трьох найкращих моделей, які проходять до наступного кроку.

Таблиця 3.3 – Результати підбору гіперпараметрів за допомогою GridSearch у випадку одношарової нейронної мережі

Набір параметрів	Перший експеримент	Другий експеримент	Третій експеримент	Середній результат
[1, 64, 100, 32, 0]	1.731	1.719	1.720	1.7230
[1, 128, 100, 32, 0]	1.723	1.727	1.771	1.7405
[1, 64, 100, 32, 1]	1.750	1.742	1.747	1.7463

В даному випадку ми отримали наступні результати:

1. Моделі з одним вхідним значення показують найкращі результати.
2. Кількість нейронів в прихованому шарі може варіюватися.
3. Розмір батчу найкращий рівний 32.
4. Порядок диференціювання може бути або першим або без диференціювання взагалі.

Ускладнимо архітектуру додатковим прихованим шаром (Рис.3.7). Визначимо кількість нейронів у цьому шарі рівною половині попереднього прихованого шару. Це зменшить кількість гіперпараметрів, які необхідно варіювати, тому тепер, коли змінюється кількість нейронів у першому шарі, у другому автоматично теж.

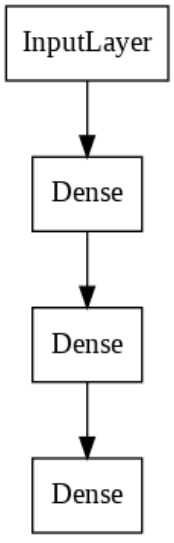


Рисунок 3.7 – Двошарова нейронна мережа

Для цієї моделі було обрано наступні гіперпараметри:

1. Розмір вхідного вікна: 1, 8.
2. Кількість нейронів у прихованому шарі: 64 (32), 128 (64).
3. Розмір батчів: 1, 32.
4. Порядок диференціювання: 0, 1, 8.

Всього 24 параметри аналогічно до першої архітектури. В Таблиці 3.4 продемонстровано результати тренування трьох найкращих моделей, які проходять до наступного кроку:

Таблиця 3.4 – Результати підбору гіперпараметрів за допомогою GridSearch у випадку двошарової нейронної мережі

Набір параметрів	Перший експеримент	Другий експеримент	Третій експеримент	Середній результат
[1, 128, 100, 32, 0]	1.740	1.718	1.753	1.7373
[1, 64, 100, 32, 1]	1.744	1.752	1.748	1.7483
[1, 128, 100, 32, 1]	1.756	1.757	1.754	1.7555

Результати даної архітектури схожі з одношаровою нейронною мережею, проте найкращий результат все таки показує перша. Це може пояснювати або малою кількістю епох, що буде виправлено в наступному кроці, або ж тим, що в датасеті мала кількість даних, тому більш складній моделі треба більше інформації для розуміння структури в даних.

Наступною архітектурою, яку було обрано є згорткова нейронна мережа (Рис.3.8). Її досить часто використовують в задачах комп’ютерного зору. Проте існують модифікації згорткового шару і шару субдискретизації, що дозволяють процювати з одномірними даними.

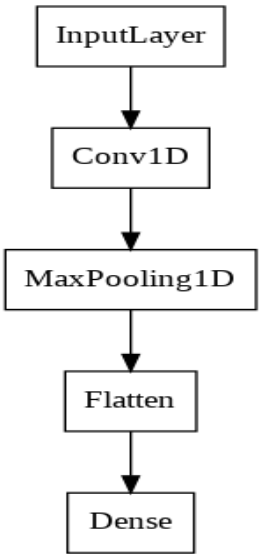


Рисунок 3.8 – Згорткова нейронна мережа

В цій моделі варіювали наступні гіперпараметри:

1. Розмір вхідного вікна: 4, 8, 12.
2. Кількість фільтрів згорткового шару: 32, 64.
3. Розмір ядра згорткового шару: 3, 5.
4. Розмір батчу: 32, 64.
5. Порядок диференціювання: 0, 1, 8.

Можна відмітити (Табл.3.5), що не дивлячись на те, що згорткові мережі знаходять своє застосування в більшості випадках у задачах комп’ютерного зору, в задачі прогнозування часових рядів вони показують також непоганий результат. Також цікавий той факт, що кожен експеримент незалежно від моделі (якщо розглядати обрані три), показують близький один до одного результат, що вказує на малий вплив випадкових факторів.

Таблиця 3.5 – Результати підбору гіперпараметрів за допомогою GridSearch у випадку згорткової нейронної мережі

Набір параметрів	Перший експеримент	Другий експеримент	Третій експеримент	Середній результат
[4, 32, 3, 100, 64, 1]	1.744	1.744	1.747	1.7448
[8, 32, 3, 100, 32, 1]	1.734	1.745	1.759	1.7458
[4, 64, 3, 100, 64, 1]	1.742	1.752	1.746	1.7466

І останньою архітектурою, яку буде розглянуто в задачі прогнозування часових рядів буде рекурентна нейронна мережа (Рис.3.9).

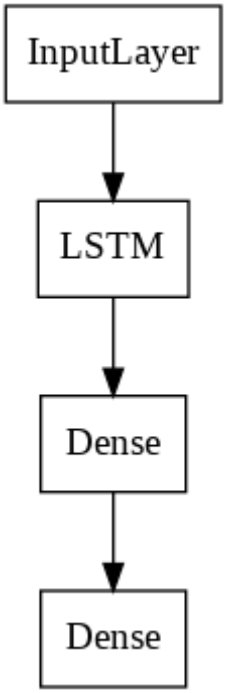


Рисунок 3.9 – Рекурентна нейронна мережа

Дана модель має наступні гіперпараметри, які було обрано для варіації:

1. Розмір вхідного вікна: 4, 8, 12.
2. Кількість нейронів в прихованому шарі: 64, 128.
3. Розмір батчу: 32, 64.
4. Порядок диференціювання: 0, 1, 8.

Результати рекурентних мереж (Табл.3.6) для даного часового ряду виявилися найгіршими, хоча саме вони домінують серед архітектур, які зазвичай використовують у роботі с послідовностями. Це може бути пов’язано з особливостями фінансових часових рядів

Таблиця 3.6 – Результати підбору гіперпараметрів за допомогою GridSearch у випадку рекурентних мереж

Набір параметрів	Перший експеримент	Другий експеримент	Третій експеримент	Середній результат
[8, 64, 100, 32, 0]	1.798	1.726	1.755	1.7595
[12, 128, 100, 64, 0]	1.760	1.759	1.780	1.7662
[4, 64, 100, 32, 0]	1.826	1.774	1.791	1.7969

Після цього кроку було протестовано велику кількість гіперпараметрів. Це був «широкий» підбір параметрів, більш вузький буде виконано в наступних кроках. Було обрано по три набори гіперпараметрів для кожної архітектури, що показують найкращий результат.

На третьому етапі алгоритму виконується навчання обраних моделей на більшій кількості епох, для досягнення найкращих результатів.

Збільшимо кількість епох до 300, а кількість експериментів до 10. В Таблицях 3.7-3.10 показані метрики RMSE для експериментів кожного типу нейромереж, а також їх середній результат.

Таблиця 3.7 – Значення RMSE та їх усереднене значення під час донавчання обраних моделей одношарових нейронних мереж

Номер експерименту	[1, 64, 300, 32, 0]	[1, 128, 300, 32, 0]	1, 64, 300, 32, 1]
1	1.733	1.720	1.764
2	1.866	1.763	1.756
3	1.762	1.761	1.764
4	1.718	1.718	1.763
5	1.726	1.836	1.755
6	1.718	1.731	1.766
7	1.727	1.723	1.763
8	1.758	1.719	1.760
9	1.729	1.807	1.756
10	1.722	1.816	1.753
Середнє значення	**1.746**	**1.759**	**1.760**

Таблиця 3.8 – Значення RMSE та їх усереднене значення під час донавчання обраних моделей двошарових нейронних мереж

Номер експерименту	[1, 128, 300, 32, 0]	[1, 64, 300, 32, 1]	[1, 128, 300, 32, 1]
1	1.719	1.880	1.893
2	1.733	1.849	1.858
3	1.731	1.780	1.967
4	1.804	1.824	1.890
5	1.731	1.835	1.895
6	1.720	1.821	1.896
7	1.820	1.782	1.901
8	1.928	1.820	1.846
9	1.738	1.879	1.908
10	1.727	1.761	1.895
Середнє значення	**1.765**	**1.823**	**1.895**

Таблиця 3.9 – Значення RMSE та їх усереднене значення під час донавчання обраних моделей згорткових нейронних мереж

Номер експерименту	[4, 32, 3, 300, 64, 1]	[8, 32, 3, 300, 32, 1]	[4, 64, 3, 300, 64, 1]
1	1.751	1.778	1.769
2	1.744	1.811	1.766
3	1.760	1.822	1.762
4	1.745	1.850	1.769
5	1.759	1.776	1.760
6	1.760	1.757	1.764
7	1.767	1.804	1.757
8	1.764	1.751	1.777
9	1.766	1.766	1.764
10	1.758	1.870	1.775
Середнє значення	**1.757**	**1.798**	**1.766**

Таблиця 3.10 – Значення RMSE та їх усереднене значення під час донавчання обраних моделей рекурентних нейронних мереж

Номер експерименту	[8, 64, 300, 32, 0]	[12, 128, 300, 64, 0]	[4, 64, 300, 32, 0]
1	1.856	1.889	1.887
2	1.834	1.762	1.767
3	1.740	1.806	1.769
4	1.815	1.763	2.015
5	1.775	1.764	1.827
6	1.779	1.802	1.819
7	1.742	1.813	1.804
8	1.855	1.800	1.765
9	1.769	1.880	2.019
10	1.820	1.762	1.985
Середнє значення	**1.799**	**1.804**	**1.866**

На четвертому кроці алгоритму було відібрано наступні моделі:

1. Одношарова нейронна мережа з одним вхідним нейроном, 64 нейронами в прихованому шарі, розміром батчу рівним 32 та без диференціювання даних.
2. Двошарова нейронна мережа з одним вхідним нейроном, 128 нейронами в першому прихованому шарі та 64 – в другому, розміром батчу рівним 32 та без диференціювання даних.
3. Згорткова нейронна мережа з 4 вхідними нейронами, 32 фільтрами, розміром ядра рівним 3, розміром батчу рівним 64 та з першим порядком диференціювання.
4. Рекурентна нейронна мережа з 8 вхідними нейронами, 64 нейронами в прихованому шарі, розміром батчу рівним 32 та без диференціювання.

Відбір моделей проводився на основі метрики кореня середньоквадратичної похибки.

На п’ятому етапі алгоритму виконано звужене коригування гіперпараметрів обраних на попередньому етапі моделей. Додатково було

обрано двоє гіперпараметрів, пошук серед яких було проведено – це функція активації та оптимізатор. Будемо використовувати наступні варіанти:

1. Функція активації: Exponential Linear Unit (elu), Linear (linear), Rectified Linear Unit (relu), Scaled Exponential Linear Unit (selu), Sigmoid (sigmoid), Swish (swish).
2. Оптимізатор: AdaDelta, AdaGrad, Adam, NAdam, RMSProp, SGD.

Оскільки використання GPU для рекурентних нейронних мереж не підтримує CUDA, то це значно сповільнює їх навчання, тому для рекурентних нейронних мереж звузимо множини гіперпараметрів до наступних:

1. Функція активації: Linear (linear), Rectified Linear Unit (relu), Scaled Exponential Linear Unit (selu), Swish (swish).
2. Оптимізатор: AdaGrad, Adam, RMSProp, SGD.

Всі інші гіперпараметри залишаються незмінними. В Таблицях 3.11 – 3.14 наведені значення RMSE та усереднене значення цієї метрики для трьох найкращих наборів гіперпараметрів для кожного типу мереж.

Таблиця 3.11 – Значення RMSE трьох найкращих наборів функції активації та оптимізатора для обраної моделі одношарової нейронної мережі

Номер експерименту	['linear', 'adagrad']	['swish', 'adam']	['selu', 'adam']
1	1.720	1.723	1.731
2	1.724	1.738	1.743
3	1.725	1.742	1.727
4	1.724	1.730	1.759
5	1.724	1.724	1.728
6	1.722	1.722	1.771
7	1.723	1.722	1.731
8	1.723	1.747	1.722
9	1.721	1.724	1.823
10	1.720	1.722	1.731
Середнє	**1.7226**	**1.7294**	**1.7468**
Мінімальне	**1.720**	**1.722**	**1.722**

Таблиця 3.12 – Значення RMSE трьох найкращих наборів функції активації та оптимізатора для обраної моделі двошарової нейронної мережі

Номер експерименту	[‘linear’, ‘adadelta’]	[‘relu’, ‘adadelta’]	[‘relu’, ‘adagrad’]
1	1.718	1.719	1.720
2	1.718	1.719	1.719
3	1.719	1.719	1.720
4	1.718	1.720	1.721
5	1.719	1.721	1.719
6	1.719	1.719	1.720
7	1.719	1.719	1.720
8	1.719	1.722	1.720
9	1.720	1.720	1.720
10	1.719	1.720	1.720
Середнє	**1.7188**	**1.7197**	**1.7199**
Мінімальне	**1.718**	**1.719**	**1.719**

Таблиця 3.13 – Значення RMSE трьох найкращих наборів функції активації та оптимізатора для обраної моделі згорткової нейронної мережі

Номер експерименту	[‘sigmoid’, ‘adagrad’]	[‘linear’, ‘nadam’]	[‘linear’, ‘sgd’]
1	1.719	1.727	1.729
2	1.727	1.727	1.717
3	1.725	1.730	1.729
4	1.714	1.729	1.728
5	1.723	1.728	1.724
6	1.712	1.726	1.727
7	1.724	1.724	1.725
8	1.724	1.728	1.727
9	1.726	1.724	1.726
10	1.730	1.727	1.731
Середнє	**1.7225**	**1.7261**	**1.7264**
Мінімальне	**1.712**	**1.724**	**1.717**

Таблиця 3.14 – Значення RMSE трьох найкращих наборів функції активації та оптимізатора для обраної моделі рекурентної нейронної мережі

Номер експерименту	[‘swish’, ‘adagrad’]	[‘relu’, ‘adagrad’]	[‘linear’, ‘adagrad’]
1	1.741	1.756	1.791
2	1.779	1.755	1.764
3	1.749	1.754	1.749
4	1.751	1.723	1.737
5	1.776	1.779	1.738
6	1.729	1.776	1.730
7	1.722	1.750	1.750
8	1.756	1.731	1.791
9	1.803	1.730	1.787
10	1.772	1.856	1.817
Середнє	**1.758**	**1.761**	**1.766**
Мінімальне	**1.722**	**1.723**	**1.730**

На основі середнього значення RMSE підібрано функції активації та оптимізатори (Табл.3.15).

Таблиця 3.15 – Найкращі за метрикою RMSE функції активації та оптимізатор відповідно до кожного типу нейронної мережі

Тип	**Активація**	**Оптимізатор**	**RMSE**
Одношарова	Linear	AdaGrad	1.7226
Двошарова	Linear	AdaDelta	1.7188
Згорткова	Sigmoid	AdaGrad	1.7225
Рекурентна	Swish	AdaGrad	1.7220

На останньому шостому етапі проведено відбір найкращої моделі на валідаційній вибірці, яка не була включена у навчання та підбір гіперпараметрів обраних моделей. Вона містить записи за перших три місяці 2020 року – 62 точки. Для вибору архітектури використаємо наступні метрики:

– максимальне значення похибки;
– середню абсолютну відсоткову похибку;
– корінь з середньоквадратичної похибки;
– коефіцієнт детермінації.

На Рисунках 3.10-3.11 зображені графіки прогнозів і реальних валідаційних даних відповідно згорткової та рекурентної мереж.

У Таблиці 3.16 наведені значення різних метрик відповідно до кожної моделі.

Таблиця 3.16 – Значення різних метрик якості для різних типів моделей

Модель	Max. Error	MAPE	RMSE	R2
Одношарова	28.8191	5.2124%	9.4117	0.8956
Двошарова	30.0523	5.1916%	9.3914	0.8961
Згорткова	30.6004	5.2072%	9.4089	0.8957
Рекурентна	34.5901	8.0506%	12.9874	0.8013

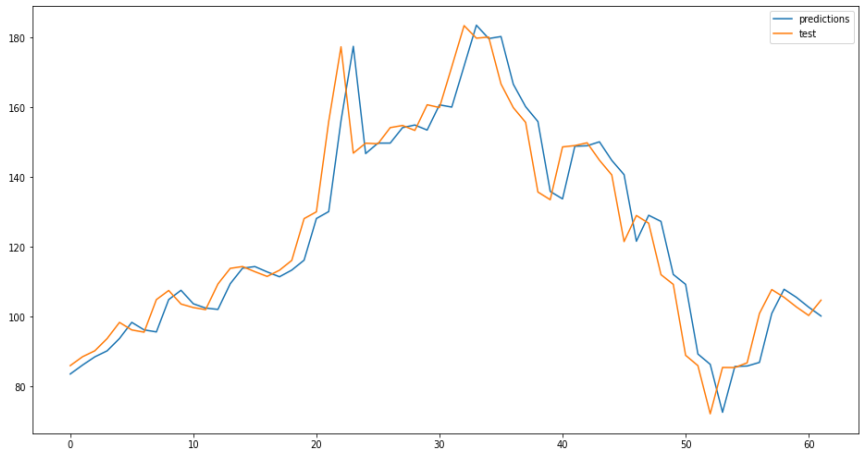


Рисунок 3.10 – Графік прогнозування згорткової нейронної мережі з реальними даними

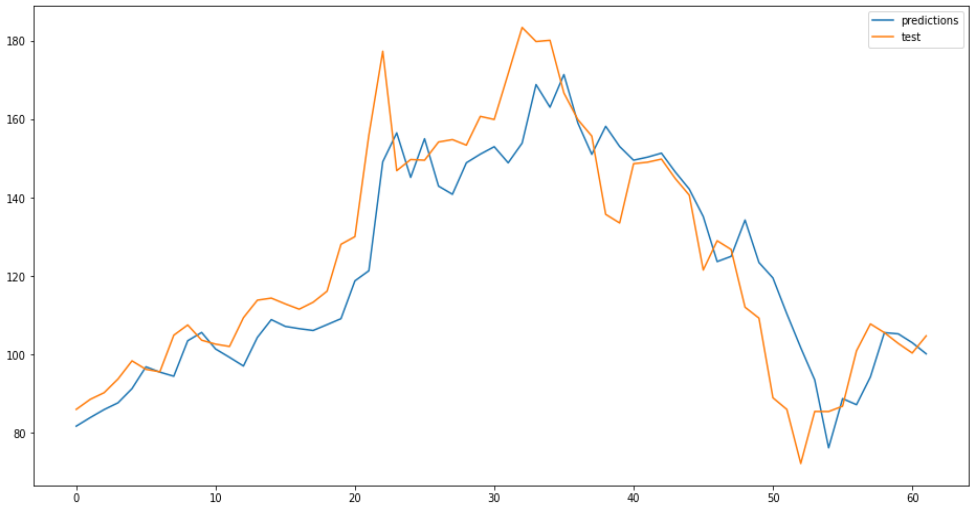


Рисунок 3.11 – Графік прогнозування рекурентної нейронної мережі з реальними даними

3.4 Аналіз результатів прогнозування фінансових ринків

Розрахунки для вибраного ряду показали, що одношарова, двошарова і згорткова мережі прийшли до схожого рішення, а саме прогнозування наступного значення таким самим як і попереднє з мінімальними змінами один від одного. І єдина модель, яка до цього не прийшла – це рекурентна нейронна мережа.

Рекурентна мережа показала для заданого ряду найгірші результати по всім розглянутим метрикам. Тому робимо наступний висновок:

Якщо нас цікавить мінімізація ризику, тоді для заданого ряду краще обрати одношарову нейронну мережу з підібраними раніше гіперпараметрами.

Якщо ж задачі стоїть в тому, щоб для заданого ряду знайти модель, яка в середньому буде показувати найбільш точні короткострокові прогнози, тоді краще обрати двошарову нейронну мережу з відповідними гіперпараметрами, оскільки по всім розглянутим показникам крім коефіцієнту детермінації вона краща за згорткову.

Висновки за роздіом 3

З використанням методу пошуку посітці модифікованим під часові ряди було розроблено і описано алгоритм пошуку архітектури глибокої нейронної мережі та її гіперпараметрів з найкращою якістю короткострокового прогнозування.

В цьому розділі було детально проведено експерименти із запропонованим у роботі методом пошуку архітектури нейронної мережі та її гіперпараметрів.

Для демонстрації тривіальних моделей побудована лінійна регресія та авторегресія. Також було проведено детальний пошук по сітці для віднаходження оптимальних гіперпараметрів різних типів нейронних мереж.

За допомогою таких метрик як коефіцієнт детермінації, корінь середньоквадратичної похибки, середньої абсолютної похибки та середньої абсолютної відсоткової похибки було віднайдено найкращу модель для даного типу даних.

Було продемонстровано доцільність наведено у роботі алгоритму, оскільки на кожному його етапі дані метрики RMSE покращувалися.

РОЗДІЛ 4

РОЗРОБКА ВЛАСНОГО СТАРТАП ПРОЄКТУ

4.1 Сутність та особливості стартапу

Мій варіант виробу повинен виконувати функції інтуїтивно зрозумілого для кінцевого користувача інтерфейсного програмного забезпечення для моделювання та прогнозування фінансових процесів з використанням авторегресійних методів, методів машинного навчання та різних архітектур нейроннах мереж. В Таблиці 4.1 наведена морфологічна карта, а в Таблиці 4.2 – план оновлення.

Таблиця 4.1 – Морфологічна карта

Основні параметри	**Проміжні рішення**				
	1-ше	**2-ге**	**3-е**	**4-те**	**5-те**
Сфера застосування даних	Демографічні процеси	Фінансові процеси (курси валют, значення ВВП і т.д.)	Страхові випадки та кредити (ймовірні ризики та історія клієнта)	Медична галузь	Інше
Формат вхідних даних	Excel таблиця	Csv файл	База даних	Хмарне сховище	Інше
Методи прогнозування	Нейронні мережі	Байєсівська мережа	Регресійний аналіз	Методи Інтелектуального Аналізу Даних	Інше

Продовження таблиці 4.1

Тип програми	Веб додаток	Локальна версія для ПК	Мобільний додаток	Програма у форматі VR	Інше
Мова програмування	Python	R	SAS	Matlab	Інше
Тип ліцензій	Безкоштовна	Комерційна	Випробувальна версія	Студентська	Інше

Таблиця 4.2 – План оновлення

№ з/п	**Запитання**	**Відповідь**
1	**2**	**3**
1	Частиною яких систем є продукт?	Аналітичні пакети програмних засобів
2	Які функції надсистеми може виконувати продукт? Як їх з ним пов’язати?	Прогноз макроекономічних показників у світі, прогнозуання фінансових ринків, кредитоспроможності клієнта банку
3	Чи можна розділити продукт на частини?	Базу даних можна зберігати локально або на загальному сервері, коли обчислювальні процеси виконуються у хмарі, а інтерфейс на девайсі користувача
4	Чи можна об’єднати (агрегувати) кілька елементів продукту в один?	Безперечно для комфортного та ефективного користування кінцевим продуктом буде краще зберігати всі елементи продукту в одному місці, для ефективного виконання обчислювальних процесів

Продовження таблиці 4.2

5	Яким має бути ідеальний продукт?	Продукт має приймати на вхід файли будь-якого типу, а також враховувати специфікацію та особливості даних, зручний та інтуїтивно зрозумілий користувацький інтерфейс, низьковартісні обчислювальні ресурси
6	Що відбудеться, якщо вилучити цей продукт? Чим його можна замінити?	Готовий продукт на ринку аналітики такий, як Eviews, Deductor, ghjlerwsz SAS
7	Яким цей продукт був у минулому?	Eviews – незручний користувацький інтерфейс , відсутня можливість завантаження вхідних файлів певний типів, низька обчислювальна швидкість, не врахування особливостей нестаціонарних, нелінійних процесів та інші особливості
8	На розвиток яких функцій було спрямоване удосконалення продукту?	Сучасний користувацький інтерфейс, інші платформи використання, а також врахування особливостей процесів
9	Які функції залишилися «недорозвиненими»?	Додати композицію методів для більш точного прогнозування
10	Як можна зараз розвинути ці функції?	Доступне послідовне використання методів прогнозування, що веде за собою додаткові часові витрати та матеріальні

ForecastPad – веб додаток для завантаження, аналізу та прогнозування нестаціонарних, нелінійних, динамічних процесів у різних сферах застосування. Використовуючи статистичні дані, які нам необхідно проаналізувати користувач їх підвантажує та застосовує різноманітні

математичні операції для аналізу та прогнозування. В Таблиці 4.3 наведено синхронізацію завдань.

Таблиця 4.3 – Синхронізація завдань

Етапи	**Продукти (послідовність заміщення)**	
Назва	**Збереження даних**	**Прогнозування на N кроків**
Вчора	Локально файли	Обчислення з використанням регресійних моделей
Сьогодні	Бази даних, csv, Excel	Нейронні мережі
Завтра	Хмарне сховище	Штучний інтелект
Післязавтра	Персональне хмарне сховище із VR візуалізацією даних	Future View

4.2 Формування команди стартапу

В Таблиці 4.4 наведено важливість факторів та їх вкладу у створення стартапу, а в Таблиці 4.5 вклад кожного за партнеріву зі школою оцінки 0-10. В Таблиці 4.6 зображено дольову участь кожного з партнерів по шкалі 0-10.

Таблиця 4.4 – Визначення важливості факторів щодо їх вкладу у створення та реалізацію стартапу

Фактор	**Вага (важливість)**
Ідея	6
Підготовка бізнес плану	7
Компетентність	8
Залученість і ризики	3
Обов’язки	5

Таблиця 4.5 – Оцінювання особистого внеску кожного партнера у створення та реалізацію стартапу

Фактор	**Фоменко Нікіта**	**Котирло Віталій**
Ідея	6	6
Підготовка бізнес плану	7	9
Компетентність	9	6
Залученість і ризики	2	4
Обов’язки	8	8

Таблиця 4.6 – Визначення дольової участі у стартап проекті кожного учасника

Фактор	**Вага (важливість)**	**Фоменко Нікіта**	**Котирло Віталій**
Ідея	6	6	6
Підготовка бізнес плану	7	7	9
Компетентність	8	9	6
Залученість і ризики	3	2	4
Обов’язки	5	8	8
Разом	402	203	199
Процент	100%	50,5%	49,5%

4.3 Розробка продукту для стартапу

1-й MVP: Меню прогнозування, де користувач завантажує статистичні дані у форматі .csv, за необхідністю перевіряє їх на стаціонарність за допомогою кнопки «Тест Діккі-Фуллера», за допомогою радіокнопки можна обрати модель і її порядок, та за потреби побудувати АКФ та ЧАКФ, вибравши

власноруч кількість лагів. І, власне, зробити прогноз на визначену кількість років, натиснувши кнопку ОК.

2-й MVP: Візуалізації побудови моделей для порівняння їхньої ефективності на реальних статистичних даних кількості населення України на часовому проміжку. Яку в подальшому можна використовувати для визначення оптимального порядку моделі авторегресійного прогнозування.

3-й MVP: Візуалізація методів Автокореляційної функції, а також частокової автокореляційної функції необхідної для обробки та вибору порядку моделей при натисканні однієї з кнопок програмного продукту.

4.4 Розроблення бізнес-моделі стартапу

У Таблицях 4.7-4.8 наведено бізнес модель «Canvas» та «Lean Canvas» відповідно.

Таблиця 4.7 – Бізнес-модель «Canvas»

Ключові партнери	**Ключові види діяльності**	**Ціннісна пропозиція**	**Взаємовідносини з клієнтами**	**Споживчі сегменти**
- державні служби статистики; - фондові біржі (NYSE, NASDAQ, D-J, ASX); - Финам финансовийй портал; -спеціалізовані пакети	- Розробка ПЗ. - Вирішення проблеми актуальності статистичних даних та їхня повнота. - Робота відділу підтримки.	- Актуальні статистичні дані та високоточний прогноз на будь-який проміжок часу. - Можливість безкоштовного випробувального терміну продукту.	- Щоденна робота відділу Support 24/7 в телефонному режимі, а також за допомогою чат-ботів. - Створення спеціалізованих спільнот. -Автоматизоване обслуговування	- Державні департаменти різних сфер діяльності (фінансова, економічна, соціальна); - Міжнародні організації; - Підприємства, що

Продовження таблиці 4.7

<table>
<tr>
<td>статистичнх даних(Yahoo Finance;Google Finance; Federal Reserve Bank of St. Louis (FOREX))</td>
<td>Ключові ресурси
-MySQL/MongoDB.
- Android SDK.
- Microsoft.</td>
<td colspan="2">- Підтримка продукту протягом всього етапу його використання.
- Зручний та інтуїтивно зрозумілий користувацький інтерфейс.</td>
<td>Канали збуту
- Реклама в соціальних мережах.
- Реклама на спеціалізованих платформах для ведення таргетного бізнесу.
- Особистий зв’язок та розсилка промо з державними установами.</td>
<td>потребують статистичні дані та прогноз зміни тенденцій, від яких залежить їхній бізнес</td>
</tr>
<tr>
<td colspan="3">Структура витрат
- з переважною увагою до цінності
- експлуатація серверів БД
- ефект диверсифікації
- Залучення користувачів</td>
<td colspan="3">Потоки надходження доходу
- Ліцензії програмного забезпечення;
- Продаж актуальних статистичних даних;
- Технічна підтримка та обслуговування продукту протягом всього циклу</td>
</tr>
</table>

Таблиця 4.8 – Бізнес-модель «Lean Canvas»

Проблема **1** 1) Відсутність безкоштовних платформ прогнозування 2) Відсутність підтримки протягом всього циклу співпраці 3) Відсутність надання гарантії точності спрогнозованих процесів	**Рішення** **3** - Використання альтернативних безкоштовних платформ обслуговування процесів - Довгострокові договори та відділ підтримки - Запатентовані метрики оцінки прогнозу	**Унікальна торгова пропозиція** **2** - Підтримка продукту протягом всього етапу його використання - Наявна гарантія точності прогнозу та актуальності даних, що базується на запатентованих аналітичних оцінках - Зручний та інтуїтивно зрозумілий користувацький інтерфейс	**Прихована перевага** **7** - Сфокусованість на специфіку бізнесу - Унікальність методів вирішення проблеми - Регулярні бонуси за успішну співпрацю, що проявлятимуться згідно особливостей кожного партнера	**Споживчі** **1** - Малий та середній бізнес в різних сферах - Державні та міжнародні установи - Дослідницькі та наукові центри

Продовження таблиці 4.8

<table>
<tr>
<td></td>
<td>Ключові метрики

6

- Оцінка повноти наданих статистичних даних та їх актуальність

- Оцінка точності побудови моделі та її прогнозу, як короткострокового так і довгострокового

-Стійкість моделі до впливу зовнішніх чинників в процесі</td>
<td>- Зручний та інтуїтивно зрозумілий користувацький інтерфейс</td>
<td>Канали збуту

4

Реклама в соціальних мережах

- Реклама на спеціалізованих платформах для ведення таргетного бізнесу

- Особистий зв’язок та розсилка промо з державними установами</td>
<td></td>
</tr>
<tr>
<td colspan="3">Структура витрат

5

- з переважною увагою до цінності

- експлуатація серверів БД

- ефект диверсифікації

- Залучення користувачів</td>
<td colspan="2">Потоки надходження доходу

5

- Ліцензії програмного забезпечення;

- Продаж актуальних статистичних даних;

- Технічна підтримка та обслуговування продукту протягом всього циклу</td>
</tr>
</table>

4.5 Маркетингове планування стартапу

У таблиці 4.9 було проведено аналіз характеристик потенційних клієнтів стартпу. А в таблиці 4.10 визначено ринкові загрози та можливості.

Таблиця 4.9 – Характеристика потенційних клієнтів стартап-проєкту

№ п/п	**Потреба, що формує ринок**	**Цільова аудиторія (цільові сегменти ринку)**	**Відмінності у поведінці різних потенційних цільових груп клієнтів**	**Вимоги споживачів до товару**
1	ПЗ для швидкого та точного прогнозування як короткострокового так і довгострокового	- Державні департаменти; - Міжнародні організації; - Підприємства	Сфера діяльності та відповідно сфера застосування статистичних даних	-актуальність ститистичних даних; -точність та якість прогнозу;
2	Зручний та інтуїтивно зрозумілий користувацький інтерфейс	- Державні департаменти; - Міжнародні організації; - Підприємства	1) Простота використання ПЗ 2)Високоточність результату	- Акцент на зручності та простоті інтерфейсу ПЗ - Якість та точність прогнозу

Таблиця 4.10 – Визначення ринкових можливостей і загроз

Параметри оцінки	**Можливості**	**Загрози**
1. Конкуренція	Кращі результати прогнозування за короткий проміжок часу та низьку ринкову ціну	Можлива робота у від'ємну маржинальну дохідність та з кастомними клієнтами
2. Якість та точність прогнозу	Отримання високоточного результату прогнозу	Можлива велика ресурсозатратність в обчислювальних можливостях
3. Робота з клієнтами вузької сфери діяльності	Унікальність власної розробки, яка є дефіцитною на ринку	Перепрофілювання суто на кастомного клієнта, з чого слідує відсутність універсальності та довготривалість проекту

Ступеневий аналіз конкуренції на ринку представлений у вигляді згрупованої Таблиці 4.11 всіх необхідних показників, що впливають на ті чи інші процеси, що нас цікавлять в процесі створення стартап проекту.

Таблиця 4.11 – ступеневий аналіз конкуренції на ринку

Особливості конкурентного середовища	**В чому проявляється дана характеристика**	**Вплив на діяльність підприємства (можливі дії компанії, щоб бути конкурентоспроможною)**
1. Вказати тип конкуренції - Чиста	Можливість незалежної конкуренції між компаніями та встановлення самостійної ціни, що не контролюється ринковими цінами	Можливість прояву унікальності продукту та його інновація без встановлених стандартів

Продовження таблиці 4.11

2. За рівнем конкурентної боротьби - Національна	Конкуренція на міжнародному ринку	Врахування особливостей локалізації клієнта, а не диктування своїх методів
3. За галузевою ознакою - Міжгалузева	Застосування ПЗ у різних сферах діяльності	Можливість легко інтегруватись у такі сфери як економіка, екологія, медицина, політика
4. Конкуренція за видами товарів: - Товарно-родова	Конкуренція між різноманітними вида-ми товарів, що можуть виконувати схожі функції	Більш багатофактурна сфера застосування
5. За характером конкурентних переваг - Цінова/нецінова	Конкурентна перевага і цінова і нецінова. Якість продукту та її цінова категорія	Цікаві пропозиції для заохочення менш бюджетних компаній
6. За інтенсивністю - Не марочна	Спеціальної марки програмний продукт не має	Адаптація під інші виробництва призводить іноді до зміни функцій програми

Відомий маркетолог М.Пропер наводив канонічні параметри, за якими аналізувались галузі конкуренції, для обраного стартапу наведено та проаналізовано наступні, що зображені в Таблиці 4.12.

Таблиця 4.12 – Аналіз конкуренції в галузі за М.Портером

Складові аналізу	**Прямі конкуренти в галузі**	**Потенційні конкуренти**	**Постачальники**	**Клієнти**	**Товари замінники**
	- HYDE; - NYSE; - Google Finance	- Статистика ООН; - Finam	HYDE LTD Corp., MADA ltd Comm	- Державні департамент; - Міжнародні організації; - Підприємства	Програмні продукти створені дослідницькими центрами
Висновки:	HYDE – вузькопрофільне середовище збереження даних; NYSE – не розглядає можливість прогнозування екологічних процесів Google Finance – високозатрат	Висока конкуренція на ринку, що має різноманітні шляхи розвитку	Фінансова залежність від спонсорів, що має сезонний характер та може впливати на роботоспроможність компанії	Зацікавлені в простоті використання та отриманні високоякісного результату	Дуже обмежений функціонал, що може впливати на якість результату

Далі в таблиці 4.13 наведено обгрунтування факторів конкурентоспроможності.

Таблиця 4.13 – обґрунтування факторів конурентноспроможності

№ п/п	Фактор конкурентоспроможності	Обґрунтування (наведення чинників, що роблять фактор для порівняння конкурентних проектів значущим)
1	Унікальність користувацького інтерфейсу	На ринку дуже мало компаній випускають зручний та зрозумілий у використанні інтерфейс, щоб персонал без додаткових туторіалів та навчальних лекцій не інтегрував інтерфейс в робочий процес
2	Математична та статистична база	Використання найновіших методик розрахунку прогнозу з урахуванням особливостей різноманітних процесів
3	Актуальність даних	Використання актуальних статистичних даних, що оновлюються якнайшвидше, залежно від характеру процесів та специфіки бажаного результату

Нарешті можна продемонструвати SWOT- аналіз стартап-проекту, що зображений в Таблиці 4.14.

Таблиця 4.14 – SWOT-аналіз стартап-проєкту

Сильні сторони:	**Слабкі сторони:**
- Унікальні підходи до математичного розрахунку; - Високоякісні технології; - Великий спектр застосування; - Інтуїтивно зрозумілий інтерфейс	- Молода компанія на ринку; - Відсутність налагоджених зв’язків з клієнтом; - Відсутність початкового фінансування;

Продовження таблиці 4.14

Можливості:	**Загрози:**
- Заохочення малого та середнього бізнесу; - Робота висококваліфікованих спеціалістів; - Інтегрування процесів прогнозування в інші сфери діяльності; - Іноваційні підходи та цікаві пропозиції	- Можливість поглинання конкурентами; - Неактуальність на локальному ринку; - Втрата стартової компенсації

4.6 Бізнес-план стартап проекту

Для хорошого бізнес- плану стартап проекту дуже важливо мати визначення базової стратегії розвитку, що підготована та сформована в таблиці 4.15.

Таблиця 4.15 – Визначення базової стратегії розвитку

п/п	**Стратегія охоплення ринку**	**Ключові конкурентоспроможні позиції відповідно до обраної альтернативи**	**Базова стратегія розвитку**
1	Промо-розсилка в зацікавлені державні департамент та дослідні центри, а також пропозиції малому та середньому бізнесу.	Випробувальні та безкоштовні версії ПЗ на обмежений час та для кастомних клієнтів. Цікаві пропозиції в напрямку підтримки власного продукту на стороні клієнта	Максимальна рекламна компанія та охват цільової аудиторії. Запровадження ПЗ та демонстрація роботи. Підтримка та оновлення ПЗ протягом всього життєвого циклу.

А також не менш важливо мати вже сформовану таблицю 4.16, що містить визначення базової стратегії конкурентної поведінки.

Таблиця 4.16 – Визначення базової стратегії конкурентної поведінки

п/п	**Чи є проект «першопрохідцем» на ринку?**	**Чи буде компанія шукати нових споживачів, або забирати існуючих у конкурентів?**	**Чи буде компанія копіювати основні характеристики товару конкурента, і які?**	**Стратегія конкурентної поведінки***
1	Ні, але має унікальні ідеї	Орієнтована на пошук нових партнерів, шляхом зацікавлення в новому підході до ведення їхнього бізнесу	Основні характеристики будуть незмінні, оскільки вони базуються на математичних законах, а підходи до рішення будуть змінюватись	Стратегією будуть особливі пропозиції та інноваційні підходи

Визначення ключових переваг концепції потенційного товару займає також велике значення в аналізі розвитку стартапу, тому для ефективного виконання цього завдання створена таблиця 4.17.

Таблиця 4.17 – Ключові переваги концепції товару

п/п	Потреба	Вигода, яку пропонує товар	Ключові переваги перед конкурентами (існуючі або такі, що потрібно створити)
1	Отримання актуальних даних	Високоточні статистичні дані в режимі реального часу	Оновлення даних якнайшвидше в режимі реального часу залежно від специфіки даних та поставлених бізнес-цілей

Продовження таблиці 4.17

2	Високоякісний довгостроковий/ короткостроковий прогноз	Побудований прогноз, який відповідає усім критерія оцінки	Використання найновіших методик розрахунку прогнозу з урахуванням особливостей різноманітних процесів та поставленого бізнес-плану користувача
3	Інтуїтивно зрозумілий інтерфейс	Простота та лаконічність зовнішньої оболонки ПЗ, яку бачить та використовує користувач	На ринку дуже мало компаній випускають зручний та зрозумілий у використанні інтерфейс, щоб персонал без додаткових туторіалів та навчальних лекцій не інтегрував інтерфейс в робочий процес

Опис всіх рівнів моделі товару, який буде результатом стартапу наведений в таблиці 4.18.

Таблица 4.18 – Опис трьох рівнів моделі товару

Рівні товару	**Сутність та складові**
I. Товар за задумом	Сервіс прогнозування
II. Товар у реальному виконанні	Сервіс надання актуальної статистичної інформації та побудова на основі неї високоякісного короткострокового/довгострокового прогнозу враховуючи особливості обраних динамічних процесів; Сучасний зрозумілий інтерфейс для візуалізації та запуску усіх обраних процесів
III. Товар із підкріпленням	Можливість автономного виконання прогнозування на вказаний період часу та тригерні запуски функціоналу залежно від особливостей поведінки нестаціонарних нелінійних процесів у різноманітних сферах діяльності

Формування системи збуту наведено в таблиці 4.19.

Таблиця 4.19 – Формування системи збуту

п/п	**Специфіка закупівельної поведінки цільових клієнтів**	**Функції збуту, які має виконувати постачальник товару**	**Глибина каналу збуту**	**Оптимальна система збуту**
1	Державні організації міжнаціонального рівня	Виняткові права на постачання товару з відсутністю посередників та дотримання норм чинного законодавства	канал нульового рівня	Розсилка промо в держоргани відповідальним особам
2	Малий та середній бізнес	Виняткові права на постачання товару з відсутністю посередників та дотримання комерційної таємниці про співробітництво	канал нульового рівня	Розсилка промо на офіційну робочу електрону пошту

Концепція маркетингових комунікацій можна зобразити у форматі таблиці 4.20, що представлена нижче.

Таблиця 4.20 – Концепція маркетингових комунікацій

п/п	**Специфіка поведінки цільових клієнтів**	**Канали комунікацій, якими користуються цільові клієнти**	**Ключові позиції, обрані для позиціонування**	**Завдання рекламного повідомлення**	**Концепція рекламного звернення**
	Пошук безкоштовних або низьковартісних варіантів продукту	Спеціалізовані групи та канали зв’язку в мережі інтернет	Дешевизна	Запропонувати вигідну та цікаву пропозицію з подальшою співпрацею	Реклама, що містить демонстративний характер особливостей нашого продукту

Продоження таблиці 4.20

Шляхом укладання взаємовигідних угод/договорів з двостороннім обміном	Державні канали зв’язку з потенційними партнерами	Надійна та довготривала співпраця	Представити основні переваги над конкурентами	Офіційне звернення у вигляді листа, що містить основні пункти співпраці

4.7 Правові аспекти реалізації стартапів, інтелектуальна власність та патентування

Резюме

Найменування проекту: ForecastPad;

Характеристика організації:

- ТОВ “Форкастпад”;
- Товариство з обмеженою відповідальністю;
- кількість співробітників: 2;
- контактні дані: ForecastPad_ltd@gmail.com, +380671112233;
- Фоменко Нікіта Андрійович, 22, Head of Department Analytics.

ForecastPad - “start-up” проект, реалізований як мобільний додаток для Android та IOS, а також веб-сайт для створення власних короткострокових або ж довгострокових динамічних процесів базованих на обраних статистичних даних.

Персонал:

1. Фоменко Нікіта Андрійович - Data Scientist;
2. Котирло Віталій Володимирович - Web Developer.

– На сьогоднішній день існують сервіси для обробки статистичних даних та створення короткострокових або ж довгострокових динамічних процесів, проте вони є або вузьконаправленими тільки

на певної сфери застосування або ж їх використання є багатозатратним та немає підтримки сервісу на постійній основі.

- Використання сучасних технологій для обробки актуальних статистичних даних в будь-якій сфері діяльності.
- Власні ресурси відсутні.
- Вихід на міжнародний рівень після двох років роботи, вихід на перше місце серед конкурентів.
- Потреба інвестиції у серверну частину проекту, інвестиції будуть повертатися за допомогою преміум версії підписки на сервіс, а також реклами.
- Використання ліцензій для випробувальних термінів для поширення технологій в наукових центрах та лабораторіях.
- Застосування хмарних середовищ Google для зберігання та обробки статистичної інформації з подальшим впровадженням в технологію.
- Половина середнього та великого бізнесу використовують статистичні дані для планування своєї стратегії та розподілення ресурсів.
- Самостійне створення необхідних сервісів, які є аналогами Google Analytics та поступовий перехід виключно на власний сайт.

Опис підприємства

Команда складається з двох осіб розробників, та має комерційний досвід у веб-технологіях, а саме PHP, Angular JS, React JS; розробці мобільних додатків (Kotlin, Swift), і роботи з базами даних Oracle, MySQL.

Опис продукту

ForecastPad – веб додаток для завантаження, аналізу та прогнозування нестаціонарних, нелінійних, динамічних процесів у різних сферах застосування. Використовуючи статистичні дані, які нам необхідно проаналізувати користувач їх підвантажує та застосовує різноманітні математичні операції для аналізу та прогнозування.

Розкажіть про асортимент вашої продукції, перерахуйте основні найменування ваших товарів і послуг, якщо їх занадто багато, об'єднайте в групи

Наш продукт - це комплект веб-сервіс (futuresviews.com) + мобільний додаток Android + мобільний додаток iOS.

Опишіть функціональне призначення продукції, для яких цілей вона призначена. Які потреби задовольняє, які проблеми допомагає вирішити.

Оскільки наший сайт має два глобальні підходи до використання даних та отримання бажаного результату з мінімальними затратами, користувач має свободу вибору. Перше - це зручний веб-сервіс, який поєднує в собі хмарне середовище, де можна знайти безліч корисного(найактуальніші статистичні дані структуризовані на вибір користувача відповідно його вимог з дуже зручною панеллю пошуку та голосовим асистентом), а також додати власноруч власні дані, які адаптуються до необхідної моделі. Перевага сервісу в тому, що користувачеві не потрібно зберігати всі етапи обробки та проміжні результати на власному пристрої, для цього створюється власний кабінет, де його дані будуть надійно захищені відповідно до всесвітньої технології Security Personal Data Storage. Друге – це можливість швидкого доступу до Ваших даних через мобільні додатки, що реалізовані як на Android та і на iOS. Інтуїтивно зрозумілий інтерфейс та можливість прив’язки до власного кабінету користувача спростить Вашу роботу та допоможе зберегти дорогоцінні хвилини в доступу до актуальної інформації по бізнесу.

Уявіть основні технічні характеристики вашого продукту (Не перевантажуйте читача спеціалізованими термінами).

Отже, наш веб-сервіс створений за допомогою Angular JS та в основі має одну з найпопулярніших та найнадійніших нерелятивних баз даних Oracle, основні методи та моделі високоякісного прогнозу реалізовані на Python. Та мають онлайн доступ до швидкісного оновлення даних, що прив’язані до Загальної Світової Статистичної Бази (ЗССБ).

Наведіть приклади використання продукції.

– вибір статистичних даних із категорій запропонованих веб-сервісом;
– завантаження власних статистичних даних;
– вибір адаптації статистичних даних;
– вибір параметрів для виконання прогнозу;
– вибір типу оцінки якості моделювання та прогнозування.

Перерахуйте основні етапи виробництва вашого продукту.

– побудова бази даних;
– розробка веб-сервісу;
– розробка додатку Android/ iOS;
– розробка хмарного сховища;
– підтримка та розробка нового функціоналу на всіх платформах.

Наскільки ваш продукт є універсальним або унікальним для кожного клієнта, в чому це проявляється.

Будь-який користувач може налаштувати інтерфейс на свій лад. Вибір платформи для роботи з програмою, яка інтегрується з усіма пристроями за допомогою власного кабінету. Кожен користувач може налаштувати продукт відповідно до свого типу бізнесу та особливостями ведення його відповідно географічного розташування.

Наскільки ваші товари або послуги відповідають прийнятим стандартам

Дана продукція повністю відповідає вимогам законодавства, менталітету та особливостям ведення тамтешнього бізнесу країни, де застосовується наший продукт. Торгової марки наразі немає. Дизайн сайту та додатків створений для надшвидкого та інтуїтивного використання. Продукт є унікальним не лише в Україні, а також у решті світу. Він є інтуїтивно простим у використанні, людина будь-якого сфери діяльності користуватися всіма функціями без особливої підготовки.

Товар та послуги повністю відповідають стандартам.

На якій стадії знаходиться продукт в даний час (ідея, робочий проект, дослідний зразок, серійне виробництво і т.д.)

Продукт знаходиться на стадії розробки додатку.

Вкажіть, чи є можливості для подальшого розвитку продукту

Планується перевести продукт на вищий рівень реалізації в плані покращення хмарного середовища з елементами автоматизації за допомогою Штучного інтелекту для полегшення роботи.

Проаналізуйте продукцію конкурентів, яка існує на ринку, відповідно до наступних критеріїв

На даний момент, конкурентами на нашому ринку організації, що займаються прогнозуванням вузьких сфер діяльності, проте їх достатня кількість та вони мають налагоджені зв’язки збуту продукції та вже вкладені договори про співпрацю.

Що відзначають споживачі в своїх відгуках про ваш продукт. Якщо є письмові відгуки, або рекомендації, приведіть їх в додатку.

Сьогодні продукт має схвальні відгуки на рівні університетських установ та викладачів, які допомагали з рецензією.

Аналіз ринку

Аналіз ринку вимагає табличного представлення pest – аналізу, що наведено в Таблиці 4.21.

Таблиця 4.21 – Pest - аналіз

Групи чинників	**Події/ чинники**	**Небезпека/можливість**	**Вірогідність події або прояву чинника**	**Важливість чинника або події**	**Вплив на підприємство**	**Програма дій**
Політичні	1 Державний вплив у галузь	2	1	1	1	Робота з державними сервісами
	2 Зміна законів України	1	1	2	2	Переобладнання роботи
Економічні	1 Ринок і торговельні цикли	3	2	3	3	Дослідження ринку

Продовження таблиці 4.21

	2 Витрати на підтримку	1	1	2	3	Дослідження фінансової частини
Соціальні	1 Вплив ЗМІ	1	1	1	1	Робота з ЗМІ
	2 Демографічні зміни	2	2	3	3	Реклама, та робота з ЗМІ
Технологічні	1 Нові продукти	2	3	2	3	Впровадження нових технологій, цінова політика
	2 Розвиток технологій	2	3	3	2	Впровадження нових технологій

Сильні та слабкі сторони проекту наведені в Таблиці 4.22 нижче.

Таблиця 4.22 – SWOT-аналіз стартап-проєкту

Сильні сторони:	**Слабкі сторони:**
- Унікальні підходи до математичного розрахунку; - Високоякісні технології; - Великий спектр застосування; - Інтуїтивно зрозумілий інтерфейс	- Молода компанія на ринку; - Відсутність налагоджених зв’язків з клієнтом; - Відсутність початкового фінансування;

Продовження таблиці 4.22

Можливості:	**Загрози:**
- Заохочення малого та середнього бізнесу; - Робота висококваліфікованих спеціалістів; - Інтегрування процесів прогнозування в інші сфери діяльності; - Іноваційні підходи та цікаві пропозиції	- Можливість поглинання конкурентами; - Неактуальність на локальному ринку; - Втрата стартової компенсації

Матриця оцінки загальноекономічних факторів впливу зовнішнього середовища на реалізацію стартап-проекту наведена в Таблиці 4.23.

Таблиця 4.23 Матриця оцінки загальноекономічних факторів впливу зовнішнього середовища на реалізацію стартап-проекту

Групи та окремі фактори		**Оцінка залежності бізнесу від впливу зовнішніх факторів***					**Сприятливі можливості**	**Загрози та небезпеки**
Макроекономічні фактори	Посилення інфляції		2				Немає	Немає
	Зростання доходів населення та підвищення купівельної спроможності			3			Збільшення цін	Конкуренція
	Посилення диференціації доходів населення		2				Немає	Немає
	Зниження процентних ставок на банківський кредит		2				Немає	Немає

Продовження таблиці 4.23

	Нестабільність макроекономічної ситуації			3			Немає	Немає
	Зростання економіки і розширення місткості ринку				4		Збільшення клієнтів і цін	Конкуренція
Елементи конкурентного середовища	Ринкові фактори				4		Збільшення цін	Конкуренція
	Вплив конкурентів					5	Впровадження нових технологій	Втрата клієнтів
	Вплив споживачів				4		Немає	Конкуренція
	Вплив постачальників		2				Немає	Немає
Соціально-демографічні характеристики	Зміни смаків споживачів, посилення рівня диференціації споживчих запитів та поява вільних ринкових сегментів, ін.			3			Отримання нових користувачів	Втрата користувачів та партнерів

Продовження таблиці 4.23

Зміни в системі державного регулювання	Зміни в оподаткуванні		2				Немає	Немає
	Запровадження ліцензування та обов'язкової стандартизації продукції і послуг			3			Втрата деяких конкурентів	Втрата партнерів
	Посилення державного контролю та вимог			3			Немає	Втрата партнерів
	Посилення корупції	1					Немає	Немає
	Регулювання ціноутворення				4		Немає	Втрата користувачів
	Зміни в державних замовленнях	1					Немає	Немає
Галузеві фактори	Посилення конкуренції					5	Немає	Втрата клієнтів
	Ускладнення доступу до сировини та ресурсів				4		Немає	Втрата партнерів і користувачів
	Ускладнення доступу до споживачів				4		Немає	Втрата клієнтів
	Поява товарів-замінників					5	Немає	Втрата клієнтів
	Зміна рівня цін			3			Кількість користувачів	Конкуренція

Продовження таблиці 4.23

Політичні			2				Немає	Немає
Науковотехнічні					4		Упровадження нових технологій	Конкуренція
Природні		1					Немає	Немає
Ментальні			2				Немає	Немає

План збуту та маркетингу

Планується два види отримання прибутку від створеного проекту. Перший з них це отримання коштів за куплені ліцензії різних типів, серед яких є й безкоштовні ліцензії, що покликані на заохочення до співпраці науково-дослідницьких центрів та перспективних студентів технічних вишів, а також безпосередньо категорія Преміум ліцензії, які і становитимуть вагому нішу нашого доходу. І другим видом доходу, буде монетизації кількості скачувань додатків на пристрої через Google Play та AppStore.

Вартість продукції буде розраховано з урахуванням витрат на підтримку серверної частини проекту, зарплати за розробку, оплати доставки та реклами.

Види стратегій залежно від ціни та якості продукції продемонстровані в Таблиці 4.24.

Таблиця 4.24 – Види стратегій залежно від ціни та якості продукції

Якість	**Ціна**		
	Висока	**Середня**	**Низька**
Висока	Висока ціна та якість товару	Зацікавлення покупця завдяки високій якості товару і середній ціні	Завоювання ринку, збільшення частку ринку
Середня	Мінімізуємо витрати на етапі впровадження	Встановлення середніх цін на товар середньої якості	Завоювання ринку, збільшення частку ринку
Низька	Несе загрозу втратити в майбутньому покупців	Несе загрозу втратити в майбутньому покупців	Встановлення низької ціни на товари низької якості

Наразі є план продавати понад 1000 преміум версій додатка у місяць в Таблиці 4.25.

Таблиця 4.25 – План-продажів

Найменування показників	**Періоди (по місяцях)**												**Всього за рік**
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	
Преміум версія додатку													
* обсяг ліцензій	10	20	30	40	50	70	90	100	120	150	150	170	1000
* ціна	$ 500 - 6,83	$ 1000	$ 1500	$ 2000 1,11	$ 2500 1,32	$ 3500 1,43	$ 4500	$ 5000 1,58	$ 6000 1,59	$ 7500	$ 7500	$ 8500	$50000
* виручка	$ 400 - 6,83	$ 800	$ 1200	$ 1600 1,11	$ 2000 1,32	$ 2800 1,43	$ 3600	$ 4000 1,58	$ 4800 1,59	$ 6000	$ 6000	$ 6800	$ 40000

Наразі цільовою аудиторією є малий та середній бізнес, проте в перспективі є заохочення великого бізнесу в різних галузях.

Для початку потрібно $8000 для створення та підтримку сайту, мобільних додатків, юридичне оформлення.

Буде організована таргетна реклама в Інтернеті.

Виробничий план

- Купівля ліцензії на користування бази даних Oracle;
- побудова інтерфейсу додатків та веб-сервісу на Angular JS;
- додаткові функції для покращення роботи з користувачем за допомогою штучного інтелекту.

Організаційний план

Профіль посади є дуже важливими при наймі працівників, зображені в Таблиці 4.26.

Таблица 4.26 – Профіль посади

Критерій	**Зміст**
Основна освіта	Вища освіта
Додаткова освіта, спеціалізація	Комп'ютерна спеціалізація
Необхідний досвід роботи	Щонайменше 2 рік
Завдання	Підтримка веб сайту та додатків
Знання	Oracle, Python, PHP, C#, Java Script, Android dev, Swift dev
Навички, вміння, ділові якості	Робота у команді
Особистісні якості	Вміння працювати з різними ПО
Мотивація (що можемо запропонувати)	Робота з дому

Стимулами будуть:

– матеріальні: заробітна плата; постійний оклад, премії за досягнуті показники ефективності;
– нематеріальні: можливість роботи, похвала, позиція в колективі та компанії, авторитет.

Фінансово-економічний план

На разі залучення інвесторів та власні кошти є орієнтованими джерелами фінансування нашого проекту Таблиця 4.27.

Таблиця 4.27 – Планування загальних інвестицій по проєкту

Група та вид інвестицій	**Сума**
Першопочаткові інвестиції, в т.ч.:	$2000
Розробка веб-сервісу	$10000
Розробка додатку під Android	$4000
Розробка додатку під IOS	$5000
Юридичні витрати	$1000
Витрати на підтримку сайту та програм	$5000
Витрати на рекламу	$2000
ВСЬОГО ІНВЕСТИЦІЇ ПО ПРОЕКТУ:	**$29000**

Заходи щодо упередження та реагування на ризики є в Таблиці 4.28.

Таблиця 4.28 – Заходи щодо упередження та реагування на ризики

№	РИЗИК	ІНДИКАТОРИ НАСТАННЯ	ЗАХОДИ ЩОДО УПЕРЕДЖЕННЯ	ЗАХОДИ ЩОДО РЕАГУВАННЯ	ВІДПОВІДАЛЬНИЙ
1	НЕКОРЕКТНІСТЬ РОБОТИ	НЕСПРИЙНЯТНІСТЬ РЕЗУЛЬТАТІВ	ВІДПОВІДНІ ОЦІНКИ МОДЕЛЮВАННЯ	ВИБІР ІНШИХ ПАРАМЕТРІВ	Фоменко Нікіта
2	ЗАВИСАННЯ САЙТУ	ПРИПИНЕННЯ РОБОТИСЕРВІСУ	ТЕСТУВАННЯ	ВИХІД НОВИХ ВЕРСІЙ	Котирло Віталій

Розрахунок ефективності проекту, що аналізується є важливою складовою в інвестиціюванні та наведено в таблиці 4.29.

Таблиця 4.29 – Розрахунок ефективності проєкту, що аналізується

ПОКАЗНИК	2020	2021	2022	2023
1. СУМА ІНВЕСТИЦІЙ	$29000	-	-	-
2. ВИРУЧКА ВІД РЕАЛІЗАЦІЇ	-	$18500	$16000	$25400
3. ВИТРАТИ НА ЕКСПЛУАТАЦІЮ ПРОЕКТУ	-	$3200	$2400	$2900
4. АМОРТИЗАЦІЙНІ ВІДРАХУВАННЯ	-	$320	$320	$520
5. СТАВКА ДИСКОНТУ	-	18%	18%	18%
6. ГРОШОВІ ПОТОКИ	-	$16000	$14000	$24000

Продовження таблиці 4.29

7. ДИСКОНТОВАНІ ГРОШОВІ ПОТОКИ	-	$13500	$10000	$14000
8. ДИСКОНТОВАНІ ГРОШОВИЙ ПОТІК З ПОЧАТКУ ЕКСПЛУАТАЦІЇ ПРОЕКТУ	-	$13500	$24000	$38000
9. ДИСКОНТОВАНІ ГРОШОВІ ПОТОКИ ЗА СТАВКОЮ ДИСКОНТУ 40%	-	$12000	$7000	$8500
10. ДИСКОНТОВАНІ ВИГОДИ, ТИС. ГРН.	-	$11500	$12000	$16000
11. ДИСКОНТОВАНІ ВИТРАТИ, ТИС. ГРН.	-	$3000	$2000	$19000

4.8 Розрахунок ефективності проєкту

1. Сума інвестицій у проект становить $29000.

2. Дисконтовані грошові потоки в результаті реалізації проекту становитимуть за 2020-2022рр. $38000.

3. Чиста теперішня вартість проекту $\$38000 - \$29000 = \$9000$. Оскільки, $NPV > 0$, інвестиційний проект є вигідним для підприємства інвестора. За 3 роки функціювання проекту грошовий потік не лише задовольняє очікування інвестора у відношенні щодо одержання доходу, а й перевищують очікувані доходи на $9000.

4. Термін окупності інвестицій. Застосуємо алгоритм розрахунку інвестицій:

$$\text{ТО} = (3-1) + (29000 - 24000) / 24000 = 2{,}208 \text{ роки}$$

5. Внутрішня норма рентабельності. При ставці дисконту 40%, NPV дорівнює $26000. Отже, $IRR = 0{,}18 + (38000 * (0.4 - 0.18) / (38000 - 26000)) = 0{,}89$, або при ставці 89% сумарні дисконтовані вигоди дорівнюють сумарним дисконтованим витратам. Тобто IRR є ставкою дисконту, при якій NPV проекту дорівнює нулю.

6. Коефіцієнт вигід/витрат. $\$40000/\$7000 = 5{,}71$. За $1 теперішньої вартості вкладених коштів у проект підприємство отримає $5,71 теперішньої вартості доходу.

7. Індекс прибутковості $\$38000/\$29000 = 1{,}31$ Отже, $PI > 0$ і проект є ефективним.

4.9 Інвестиційна та фінансове забезпечення стартап-проекту

Назва проекту та відомості про організацію

Назва стартап-проекту: ForecastPad.

Галузева приналежність стартап-проекту: IT.

Форма участі інвестора в стартап-проекті: надання інвестицій за договором простого товариства (договору про спільну діяльність).

Місце реалізації стартап-проекту: Gorizont1, м. Київ.

Передбачувана дата початку реалізації стартап-проекту: 01.07.2020.

Повне найменування організації / підприємства (партнера по проекту / ініціатора проекту): ТОВ “Форкастпад”.

Форма власності: приватна.

Дата реєстрації: 01.04.2020.

Поточні фінансові результати: -29000$.

Опис проекту (ідея/передумови, стан та етапи проекту, очікувані результати):

Ідея, передумови

ForecastPad – веб додаток для завантаження, аналізу та прогнозування нестаціонарних, нелінійних, динамічних процесів у різних сферах застосування. Використовуючи статистичні дані, які нам необхідно проаналізувати користувач їх підвантажує та застосовує різноманітні математичні операції для аналізу та прогнозування.

Етапи:

1. Створення сайту, запуск маркетингової кампанії.
2. Створення мобільних додатків.
3. Розвиток сайту та додатків, розробка нового функціоналу.
4. Вихід на міжнародний ринок.

Результат - міжнародний проект масового форкасту, статистична платформа по набору даних

Характеристика ринку планованої до випуску продукції

1. Характеристика запланованої до випуску продукції:

– найменування та опис продукції: веб-сервіс ForecastPad.com, мобільний додаток ForecastPad;
– основні споживачі: малий та середній бізнес у різних сферах діяльності та без кордонних меж;
– зацікавлені державні та науково-дослідницькі інститути у роботі з талановитою молоддю.

2. Опис поточного стану внутрішнього ринку планованої до випуску продукції:

– молодь України, від 15 до 35 років - 11 млн;
– кількість зацікавлених видів бізнесів більше 100 типів.

3. Основні показники внутрішнього ринку планованої до випуску продукції:

- річний обсяг промислового виробництва продукції, в поточних цінах: у 2021 році;
- рентабельність продажів продукції (в%): 45.7 (у 2021 році);
- ступінь концентрації (приблизна сумарна частка ринку п'яти найбільших гравців, в%): 100.

Перспективність і конкурентні переваги проекту

1. Опис наявної інфраструктури:

- організації доставки Нова пошта, Укрпошта

2. Стратегічні переваги:

- ненасичений ринок пропозиції
- легкий вихід на міжнародну арену (організація доставки за межами України, підтримка серверів та трафіку сервісу)

Потреба у фінансуванні

Загальні інвестиційні витрати за стартап-проектом, всього: 29000$, в тому числі капітальні витрати: 2200$.

Потреба у фінансуванні проекту за рахунок інвестора: є, адже на розкрутку сервісу знадобиться багато часу, і потрібні будуть кошти на підтримку, доки сайт не почне приносити прибуток.

Попередні фінансові показники проекту

Виручка, без ПДВ (після виходу на проектну потужність): $24000.

Простий /динамічний термін окупності: 2.2 роки.

Внутрішня норма прибутковості (IRR): 89 %.

Чиста поточна вартість (NPV): $29000.

Ставка дисконтування: 18 %.

Висновки за розділом 4

Результати даного розділу демонструють, ідею створення стартапу на дану тему. Задача короткострокового прогнозування фінансових рядів актуальна і цікава для сучасного бізнесу та науки.

Рентабельність усіх процесів під час створення та реалізації даного проекта повністю окупиться та зможе приносити прибутки, адже це актуальна проблема для ведення сучасної управлінської діяльності будь-якого виду керівництва. Функціонально-вартісний аналіз демонструє, що застосування вказаних інструментів має повністю прокрити витрати на забезпечення якості програмного засобу та його реалізації.

Аналіз програмних середовищ та обчислювальні потужності кожного з ним тільки підтверджують те, що найкраще з такими задачами буде справлятись саме мова програмування Python.

ВИСНОВКИ ПО РОБОТІ ТА ПЕРСПЕКТИВИ ПОДАЛЬШИХ ДОСЛІДЖЕНЬ

Магістерська дисертація є результатом дослідження, аналізу, моделювання та прогнозування фінансових часових рядів, для яких характерні нелінійність та нестаціонарність. Також розроблено алгоритм за допомогою якого можна обрати архітектуру глибокої нейронної мережі та її гіперапараметрів для задачі короткострокового прогнозування. У даній роботі було продемонстровані економетричні моделі та моделі глибоких нейронних мереж, що побудовані на фінансових даних фондових ринків.

У ході виконання дисертації було створено набір різноманітних глибоких нейронних мереж, які включали в себе нейронні мережі прямого розповсюдження, згорткові нейронні мережі та рекурентні. Виконано аналіз випробувальних даних на наявність нелінійності, нестаціонарності, належності до нормального розподілу за допомогою графічних та статистичних методів. Зроблено графічну візуалізуцію прогнозування різних типів моделей. Перевірено адекватність побудованих моделей на основі критеріїв адекватності та зроблено висновки щодо якості оцінювання прогнозу. Отримані результати короктострокового прогнозування імплементовано у графіки, що демонструють майбутню тенденцію характеру поведінки розглянутих часових рядів.

Що стосується результатів запропоноваго у роботі алгоритму, то Він дозволяє швидко звузити множину гіперпараметрів і віднайти ту модель, що показує найкращий результат за різними метриками, наприклад: RMSE, MAPE, Max. Error та R^2. На прикладі акцій компанії Tesla було знайдено дві моделі, які показують досить гарні показники різних метрик – це моделі одношарової та двошарової нейронної мереж. На кожному етапі алгоритму значення метрики RMSE у даному прикладі зменшувалося, що вказує на доцільність пропонованого алгоритму.

Для покращення майбутніх досліджень фінансових процесів необхідно спробувати застосувати інші моделі регресійного аналізу та їх модифіковані типи, а також спробувати ансамбль цих методів. Також варто дослідити представлений у роботі алгоритм для довгострокового прогнозування фінансових рядів. Ще одним напрямком руху, який можна відзначити є прогнозування багатовимірних часових рядів.

ПЕРЕЛІК ПОСИЛАНЬ

1. William F. S. The Sharpe Ratio. *The Journal of Portfolio Management*. 1994, January *21. P. 49-59.*

2. Ruey S. T. Analysis of Financial Time Series. 3rd ed. Hoboken, NJ: John Wiley & Sons, Inc, 2010. 720 p.

3. William H. G. Econometric Analysis. 7th ed. London: Pearson Education, 2011, 1232 p.

4. Dominick S., Derrick R. Schaum's Outline of Statistics and Econometrics, New York: McGraw-Hill Education, 2011, 336 p.

5. Afshine A., Shervine A. Deep Learning cheatsheet. URL: https://stanford.edu/~shervine/teaching/cs-229/cheatsheet-deep-learning.

6. Гудфеллоу Я., Бенджио И., Курвилль А.. Глубокое обучение. пер. с анг. А. А. Слинкина. 2-е изд., испр. М.: ДМК Пресс, 2018, 652 с.

7. Николенко С., Кадурин А., Архангельская Е. Глубокое обучение. СПб.: Питер, 2018, 480 с.

8. Prajit R., Barret Z., Quoc V. L. Searching for activation function. URL: https://arxiv.org/pdf/1710.05941.pdf.

9. Aarthi K. Optimization Techinques popularly used in Deep Learning. URL: https://medium.com/@minions.k/optimization-techniques-popularly-used-in-deep-learning-3c219ec8e0cc.

10. Jason B. How to Grid Search Deep Learning Models for Time Series Forecasting. URL: https://machinelearningmastery.com/how-to-grid-search-deep-learning-models-for-time-series-forecasting/.

11. Cristophe P. How to use Deep Learning for Time Series Forecasting. URL: https://towardsdatascience.com/how-to-use-deep-learning-for-time-series-forecasting-3f8a399cf205.

12. Bryan L., Stefan Z. Time Series Forecasting With Deep Learning: A Survey. URL: https://arxiv.org/abs/2004.13408.

13. Nedashkovskaya N.I. Investigation of methods for improving consistency of a pairwise comparison matrix. *Journal of the Operational Research Society*. 2018. Vol.69, No.12, P.1947 – 1956.

14. Бідюк П.І., Романенко В.Д., Тимощук О.Л. Аналіз часових рядів. Київ: Політехніка, 2012. 360 с.

ДОДАТОК А

ЛІСТИНГ ПРОГРАМИ

```
# -*- coding: utf-8 -*-
"""magistr.ipynb

Automatically generated by Colaboratory.

Original file is located at

https://colab.research.google.com/drive/1NoZPx_BID_1UrTTrchgxahCCL6JiZH10
"""



!pip install yfinance

import tensorflow as tf
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import os
import yfinance as yf

plt.rcParams['figure.figsize'] = (16, 8)

tsla = yf.Ticker("TSLA")
hist = tsla.history('max')
series = hist['Close'][(hist.index >= '2013-01-01') & (hist.index < '2020-01-01')]

series.plot()

"""___

# 1. Perceptron
"""

# grid search mlps
from math import sqrt
from numpy import array
from numpy import mean
from pandas import DataFrame
from pandas import concat
from pandas import read_csv
from sklearn.metrics import mean_squared_error
from keras.models import Sequential
from keras.layers import Dense, Dropout
from time import time

# split a univariate dataset into train/test sets
def train_test_split(data, n_test):
	return data[:n_test], data[n_test:]

# transform list into supervised learning format
def series_to_supervised(data, n_in=1, n_out=1):
	df = DataFrame(data)
	cols = list()
	# input sequence (t-n, ... t-1)
	for i in range(n_in, 0, -1):

	cols.append(df.shift(i))
	# forecast sequence (t, t+1, ... t+n)
	for i in range(0, n_out):

	cols.append(df.shift(-i))
	# put it all together
	agg = concat(cols, axis=1)
	# drop rows with NaN values
	agg.dropna(inplace=True)
	return agg.values

# root mean squared error or rmse
def measure_rmse(actual, predicted):
	return sqrt(mean_squared_error(actual, predicted))

# difference dataset
def difference(data, order):
	return [data[i] - data[i - order] for i in range(order, len(data))]

# fit a model
def model_fit(train, config):
  # unpack config
  n_input, n_nodes, n_epochs, n_batch, n_diff = config
  # prepare data
  if n_diff > 0:
    train = difference(train, n_diff)
  # transform series into supervised format
  data = series_to_supervised(train, n_in=n_input)
  # separate inputs and outputs
  train_x, train_y = data[:, :-1], data[:, -1]
  # define model
  model = Sequential()
  model.add(Dense(n_nodes, activation='relu', input_dim=n_input))
  model.add(Dense(1))
  model.compile(loss='mse', optimizer='adam')
  # fit model
  model.fit(train_x, train_y, epochs=n_epochs, batch_size=n_batch, verbose=0)
  return model

# forecast with the fit model
def model_predict(model, history, config):
	# unpack config
	n_input, _, _, _, n_diff = config
	# prepare data
	correction = 0.0
	if n_diff > 0:
		correction = history[-n_diff]
		history = difference(history, n_diff)
	# shape input for model
	x_input = array(history[-n_input:]).reshape((1, n_input))
	# make forecast
	yhat = model.predict(x_input, verbose=0)
	# correct forecast if it was differenced
	return correction + yhat[0]

# walk-forward validation for univariate data
def walk_forward_validation(data, n_test, cfg):
  start = time()
  predictions = list()
  # split dataset
  train, test = train_test_split(data, n_test)
  # fit model
  model = model_fit(train, cfg)
  # seed history with training dataset
  history = [x for x in train]
  # step over each time-step in the test set
  for i in range(len(test)):
    # fit model and make forecast for history
    yhat = model_predict(model, history, cfg)
    # store forecast in list of predictions
    predictions.append(yhat)
    # add actual observation to history for the next loop
    history.append(test[i])
  # estimate prediction error
  error = measure_rmse(test, predictions)
  end = time()
  print(' > %.3f | %.3fs' % (error, end - start))
  return error

# score a model, return None on failure
def repeat_evaluate(data, config, n_test, n_repeats=3):
	# convert config to a key
	key = str(config)
	# fit and evaluate the model n times
	scores = [walk_forward_validation(data, n_test, config) for _ in range(n_repeats)]
	# summarize score
	result = mean(scores)
	print('> Model[%s] %.3f' % (key, result))
	return (key, result)

# grid search configs
def grid_search(data, cfg_list, n_test):
	# evaluate configs
	scores = [repeat_evaluate(data, cfg, n_test) for cfg in cfg_list]
	# sort configs by error, asc
	scores.sort(key=lambda tup: tup[1])
	return scores

# create a list of configs to try
def model_configs():
	# define scope of configs
	n_input = [1, 8]
	n_nodes = [64, 128]
	n_epochs = [100]
	n_batch = [1, 32]
	n_diff = [0, 1, 8]
```

```
        # create configs
        configs = list()
        for i in n_input:
                for j in n_nodes:
                        for k in n_epochs:
                for l in n_batch:
                        for m in n_diff:
                                cfg = [i, j, k, l, m]
                configs.append(cfg)
        print('Total configs: %d' % len(configs))
        return configs

data = series.values
# data split
n_test = 1509
# model configs
cfg_list = model_configs()
# grid search
scores = grid_search(data, cfg_list, n_test)
print('done')
# list top 3 configs
for cfg, error in scores[:3]:
        print(cfg, error)

from tensorflow.keras.layers import LSTM

model = Sequential()
model.add(LSTM(64, activation='relu', input_shape=(1, 1)))
model.add(Dense(32, activation='relu'))
model.add(Dense(1))
model.compile(loss='mse', optimizer='adam')

tf.keras.utils.plot_model(model, show_layer_names=False)

# grid search mlps
from math import sqrt
from numpy import array
from numpy import mean
from pandas import DataFrame
from pandas import concat
from pandas import read_csv
from sklearn.metrics import mean_squared_error
from keras.models import Sequential
from keras.layers import Dense, Dropout
from time import time

# split a univariate dataset into train/test sets
def train_test_split(data, n_test):
        return data[:n_test], data[n_test:]

# transform list into supervised learning format
def series_to_supervised(data, n_in=1, n_out=1):
        df = DataFrame(data)
        cols = list()
        # input sequence (t-n, ... t-1)
        for i in range(n_in, 0, -1):
                cols.append(df.shift(i))
        # forecast sequence (t, t+1, ... t+n)
        for i in range(0, n_out):
                cols.append(df.shift(-i))
        # put it all together
        agg = concat(cols, axis=1)
        # drop rows with NaN values
        agg.dropna(inplace=True)
        return agg.values

# root mean squared error or rmse
def measure_rmse(actual, predicted):
        return sqrt(mean_squared_error(actual, predicted))

# difference dataset
def difference(data, order):
        return [data[i] - data[i - order] for i in range(order, len(data))]

# fit a model
def model_fit(train, config):
  # unpack config
  n_input, n_nodes, n_epochs, n_batch, n_diff = config
  # prepare data
  if n_diff > 0:
    train = difference(train, n_diff)
  # transform series into supervised format
  data = series_to_supervised(train, n_in=n_input)
  # separate inputs and outputs
  train_x, train_y = data[:, :-1], data[:, -1]
  # define model
  model = Sequential()
  model.add(Dense(n_nodes, activation='relu', input_dim=n_input))
  model.add(Dense(1))
  model.compile(loss='mse', optimizer='adam')
  # fit model
  model.fit(train_x, train_y, epochs=n_epochs, batch_size=n_batch, verbose=0)
  return model

# forecast with the fit model
def model_predict(model, history, config):
        # unpack config
        n_input, _, _, _, n_diff = config
        # prepare data
        correction = 0.0
        if n_diff > 0:
                correction = history[-n_diff]
                history = difference(history, n_diff)
        # shape input for model
        x_input = array(history[-n_input:]).reshape((1, n_input))
        # make forecast
        yhat = model.predict(x_input, verbose=0)
        # correct forecast if it was differenced
        return correction + yhat[0]

# walk-forward validation for univariate data
def walk_forward_validation(data, n_test, cfg):
  start = time()
  predictions = list()
  # split dataset
  train, test = train_test_split(data, n_test)
  # fit model
  model = model_fit(train, cfg)
  # seed history with training dataset
  history = [x for x in train]
  # step over each time-step in the test set
  for i in range(len(test)):
    # fit model and make forecast for history
    yhat = model_predict(model, history, cfg)
    # store forecast in list of predictions
    predictions.append(yhat)
    # add actual observation to history for the next loop
    history.append(test[i])
  # estimate prediction error
  error = measure_rmse(test, predictions)
  end = time()
  print(' > %.3f | %.3fs' % (error, end - start))
  return error

# score a model, return None on failure
def repeat_evaluate(data, config, n_test, n_repeats=10):
        # convert config to a key
        key = str(config)
        # fit and evaluate the model n times
        scores = [walk_forward_validation(data, n_test, config) for _ in range(n_repeats)]
        # summarize score
        result = mean(scores)
        print('> Model[%s] %.3f' % (key, result))
        return (key, result)

# grid search configs
def grid_search(data, cfg_list, n_test):
        # evaluate configs
        scores = [repeat_evaluate(data, cfg, n_test) for cfg in cfg_list]
        # sort configs by error, asc
        scores.sort(key=lambda tup: tup[1])
        return scores

# create a list of configs to try
def model_configs():
        # define scope of configs
        n_input = [1]
        n_nodes = [64, 128]
        n_epochs = [300]
        n_batch = [32]
        n_diff = [0, 1]
        # create configs
        configs = list()
        for i in n_input:
                for j in n_nodes:
                        for k in n_epochs:
                for l in n_batch:
                        for m in n_diff:
                                cfg = [i, j, k, l, m]
                configs.append(cfg)
        print('Total configs: %d' % len(configs))
        return configs

data = series.values
```

```
# data split
n_test = 1509
# model configs
cfg_list = model_configs()
# grid search
scores = grid_search(data, cfg_list, n_test)
print('done')
# list top 3 configs
for cfg, error in scores[:3]:
	print(cfg, error)

# grid search mlps
from math import sqrt
from numpy import array
from numpy import mean
from pandas import DataFrame
from pandas import concat
from pandas import read_csv
from sklearn.metrics import mean_squared_error
from keras.models import Sequential
from keras.layers import Dense, Dropout
from time import time

# split a univariate dataset into train/test sets
def train_test_split(data, n_test):
	return data[:n_test], data[n_test:]

# transform list into supervised learning format
def series_to_supervised(data, n_in=1, n_out=1):
	df = DataFrame(data)
	cols = list()
	# input sequence (t-n, ... t-1)
	for i in range(n_in, 0, -1):

	cols.append(df.shift(i))
	# forecast sequence (t, t+1, ... t+n)
	for i in range(0, n_out):

	cols.append(df.shift(-i))
	# put it all together
	agg = concat(cols, axis=1)
	# drop rows with NaN values
	agg.dropna(inplace=True)
	return agg.values

# root mean squared error or rmse
def measure_rmse(actual, predicted):
	return sqrt(mean_squared_error(actual, predicted))

# difference dataset
def difference(data, order):
	return [data[i] - data[i - order] for i in range(order, len(data))]

# fit a model
def model_fit(train, config):
  # unpack config
  n_input, n_nodes, n_epochs, n_batch, n_diff, activation, optimizer = config
  # prepare data
  if n_diff > 0:
    train = difference(train, n_diff)
  # transform series into supervised format
  data = series_to_supervised(train, n_in=n_input)
  # separate inputs and outputs
  train_x, train_y = data[:, :-1], data[:, -1]
  # define model
  model = Sequential()
  model.add(Dense(n_nodes, activation=activation, input_dim=n_input))
  model.add(Dense(1))
  model.compile(loss='mse', optimizer=optimizer)
  # fit model
  model.fit(train_x, train_y, epochs=n_epochs, batch_size=n_batch, verbose=0)
  return model

# forecast with the fit model
def model_predict(model, history, config):
	# unpack config
	n_input, _, _, _, n_diff, _, _ = config
	# prepare data
	correction = 0.0
	if n_diff > 0:
		correction = history[-n_diff]
		history = difference(history, n_diff)
	# shape input for model
	x_input = array(history[-n_input:]).reshape((1, n_input))
	# make forecast
	yhat = model.predict(x_input, verbose=0)
	# correct forecast if it was differenced
	return correction + yhat[0]

# walk-forward validation for univariate data
def walk_forward_validation(data, n_test, cfg):
  start = time()
  predictions = list()
  # split dataset
  train, test = train_test_split(data, n_test)
  # fit model
  model = model_fit(train, cfg)
  # seed history with training dataset
  history = [x for x in train]
  # step over each time-step in the test set
  for i in range(len(test)):
    # fit model and make forecast for history
    yhat = model_predict(model, history, cfg)
    if pd.isna(yhat) or (yhat > 999999):
      yhat = 999999
    # store forecast in list of predictions
    predictions.append(yhat)
    # add actual observation to history for the next loop
    history.append(test[i])
  # estimate prediction error
  error = measure_rmse(test, predictions)
  end = time()
  print(' > %.3f | %.3fs' % (error, end - start))
  return error

# score a model, return None on failure
def repeat_evaluate(data, config, n_test, n_repeats=10):
	# convert config to a key
	key = str(config)
	# fit and evaluate the model n times
	scores = [walk_forward_validation(data, n_test, config) for _ in range(n_repeats)]
	# summarize score
	result = mean(scores)
	print('> Model[%s] %.3f' % (key, result))
	return (key, result)

# grid search configs
def grid_search(data, cfg_list, n_test):
	# evaluate configs
	scores = [repeat_evaluate(data, cfg, n_test) for cfg in cfg_list]
	# sort configs by error, asc
	scores.sort(key=lambda tup: tup[1])
	return scores

# create a list of configs to try
def model_configs():
  # define scope of configs
  n_input = [1]
  n_nodes = [64]
  n_epochs = [300]
  n_batch = [32]
  n_diff = [0]
  activations = ['elu', 'linear', 'relu', 'selu', 'sigmoid', 'swish']
  optimizers = ['adadelta', 'adagrad', 'adam', 'nadam', 'rmsprop', 'sgd']
	# create configs
  configs = list()
  for i in n_input:
    for j in n_nodes:
      for k in n_epochs:
        for l in n_batch:
          for m in n_diff:
            for n in activations:
              for o in optimizers:
                cfg = [i, j, k, l, m, n, o]
                configs.append(cfg)
  print('Total configs: %d' % len(configs))
  return configs

data = series.values
# data split
n_test = 1509
# model configs
cfg_list = model_configs()
# grid search
scores = grid_search(data, cfg_list, n_test)
print('done')
# list top 3 configs
for cfg, error in scores[:3]:
	print(cfg, error)

"""# 2. MLP"""

# grid search mlps
from math import sqrt
from numpy import array
from numpy import mean
from pandas import DataFrame
from pandas import concat
from pandas import read_csv
from sklearn.metrics import mean_squared_error
from keras.models import Sequential
from keras.layers import Dense, Dropout
from time import time

# split a univariate dataset into train/test sets
def train_test_split(data, n_test):
	return data[:n_test], data[n_test:]
```

```
# transform list into supervised learning format
def series_to_supervised(data, n_in=1, n_out=1):
	df = DataFrame(data)
	cols = list()
	# input sequence (t-n, ... t-1)
	for i in range(n_in, 0, -1):

	cols.append(df.shift(i))
	# forecast sequence (t, t+1, ... t+n)
	for i in range(0, n_out):

	cols.append(df.shift(-i))
	# put it all together
	agg = concat(cols, axis=1)
	# drop rows with NaN values
	agg.dropna(inplace=True)
	return agg.values

# root mean squared error or rmse
def measure_rmse(actual, predicted):
	return sqrt(mean_squared_error(actual, predicted))

# difference dataset
def difference(data, order):
	return [data[i] - data[i - order] for i in range(order, len(data))]

# fit a model
def model_fit(train, config):
 # unpack config
 n_input, n_nodes, n_epochs, n_batch, n_diff = config
 # prepare data
 if n_diff > 0:
  train = difference(train, n_diff)
 # transform series into supervised format
 data = series_to_supervised(train, n_in=n_input)
 # separate inputs and outputs
 train_x, train_y = data[:, :-1], data[:, -1]
 # define model
 model = Sequential()
 model.add(Dense(n_nodes, activation='relu', input_dim=n_input)),
 model.add(Dense(n_nodes // 2, activation='relu'))
 model.add(Dense(1))
 model.compile(loss='mse', optimizer='adam')
 # fit model
 model.fit(train_x, train_y, epochs=n_epochs, batch_size=n_batch, verbose=0)
 return model

# forecast with the fit model
def model_predict(model, history, config):
	# unpack config
	n_input, _, _, _, n_diff = config
	# prepare data
	correction = 0.0
	if n_diff > 0:
		correction = history[-n_diff]
		history = difference(history, n_diff)
	# shape input for model
	x_input = array(history[-n_input:]).reshape((1, n_input))
	# make forecast
	yhat = model.predict(x_input, verbose=0)
	# correct forecast if it was differenced
	return correction + yhat[0]

# walk-forward validation for univariate data
def walk_forward_validation(data, n_test, cfg):
 start = time()
 predictions = list()
 # split dataset
 train, test = train_test_split(data, n_test)
 # fit model
 model = model_fit(train, cfg)
 # seed history with training dataset
 history = [x for x in train]
 # step over each time-step in the test set
 for i in range(len(test)):
  # fit model and make forecast for history
  yhat = model_predict(model, history, cfg)
  # store forecast in list of predictions
  predictions.append(yhat)
  # add actual observation to history for the next loop
  history.append(test[i])
 # estimate prediction error
 error = measure_rmse(test, predictions)
 end = time()
 print(' > %.3f | %.3fs' % (error, end - start))
 return error

# score a model, return None on failure
def repeat_evaluate(data, config, n_test, n_repeats=3):
	# convert config to a key
	key = str(config)
	# fit and evaluate the model n times
	scores = [walk_forward_validation(data, n_test, config) for _ in range(n_repeats)]
	# summarize score
	result = mean(scores)
	print('> Model[%s] %.3f' % (key, result))
	return (key, result)

# grid search configs
def grid_search(data, cfg_list, n_test):
	# evaluate configs
	scores = [repeat_evaluate(data, cfg, n_test) for cfg in cfg_list]
	# sort configs by error, asc
	scores.sort(key=lambda tup: tup[1])
	return scores

# create a list of configs to try
def model_configs():
	# define scope of configs
	n_input = [1, 8]
	n_nodes = [64, 128]
	n_epochs = [100]
	n_batch = [1, 32]
	n_diff = [0, 1, 8]
	# create configs
	configs = list()
	for i in n_input:
		for j in n_nodes:
			for k in n_epochs:
				for l in n_batch:
					for m in n_diff:
						cfg = [i, j, k, l, m]
	configs.append(cfg)
	print('Total configs: %d' % len(configs))
	return configs

data = series.values
# data split
n_test = 1509
# model configs
cfg_list = model_configs()
# grid search
scores = grid_search(data, cfg_list, n_test)
print('done')
# list top 3 configs
for cfg, error in scores[:3]:
	print(cfg, error)

# grid search mlps
from math import sqrt
from numpy import array
from numpy import mean
from pandas import DataFrame
from pandas import concat
from pandas import read_csv
from sklearn.metrics import mean_squared_error
from keras.models import Sequential
from keras.layers import Dense, Dropout
from time import time

# split a univariate dataset into train/test sets
def train_test_split(data, n_test):
	return data[:n_test], data[n_test:]

# transform list into supervised learning format
def series_to_supervised(data, n_in=1, n_out=1):
	df = DataFrame(data)
	cols = list()
	# input sequence (t-n, ... t-1)
	for i in range(n_in, 0, -1):

	cols.append(df.shift(i))
	# forecast sequence (t, t+1, ... t+n)
	for i in range(0, n_out):

	cols.append(df.shift(-i))
	# put it all together
	agg = concat(cols, axis=1)
	# drop rows with NaN values
	agg.dropna(inplace=True)
	return agg.values

# root mean squared error or rmse
def measure_rmse(actual, predicted):
	return sqrt(mean_squared_error(actual, predicted))

# difference dataset
def difference(data, order):
	return [data[i] - data[i - order] for i in range(order, len(data))]
```

```
# fit a model
def model_fit(train, config):
  # unpack config
  n_input, n_nodes, n_epochs, n_batch, n_diff = config
  # prepare data
  if n_diff > 0:
    train = difference(train, n_diff)
  # transform series into supervised format
  data = series_to_supervised(train, n_in=n_input)
  # separate inputs and outputs
  train_x, train_y = data[:, :-1], data[:, -1]
  # define model
  model = Sequential()
  model.add(Dense(n_nodes, activation='relu', input_dim=n_input)),
  model.add(Dense(n_nodes // 2, activation='relu'))
  model.add(Dense(1))
  model.compile(loss='mse', optimizer='adam')
  # fit model
  model.fit(train_x, train_y, epochs=n_epochs, batch_size=n_batch, verbose=0)
  return model

# forecast with the fit model
def model_predict(model, history, config):
	# unpack config
	n_input, _, _, _, n_diff = config
	# prepare data
	correction = 0.0
	if n_diff > 0:
		correction = history[-n_diff]
		history = difference(history, n_diff)
	# shape input for model
	x_input = array(history[-n_input:]).reshape((1, n_input))
	# make forecast
	yhat = model.predict(x_input, verbose=0)
	# correct forecast if it was differenced
	return correction + yhat[0]

# walk-forward validation for univariate data
def walk_forward_validation(data, n_test, cfg):
  start = time()
  predictions = list()
  # split dataset
  train, test = train_test_split(data, n_test)
  # fit model
  model = model_fit(train, cfg)
  # seed history with training dataset
  history = [x for x in train]
  # step over each time-step in the test set
  for i in range(len(test)):
    # fit model and make forecast for history
    yhat = model_predict(model, history, cfg)
    # store forecast in list of predictions
    predictions.append(yhat)
    # add actual observation to history for the next loop
    history.append(test[i])
  # estimate prediction error
  error = measure_rmse(test, predictions)
  end = time()
  print(' > %.3f | %.3fs' % (error, end - start))
  return error

# score a model, return None on failure
def repeat_evaluate(data, config, n_test, n_repeats=10):
	# convert config to a key
	key = str(config)
	# fit and evaluate the model n times
	scores = [walk_forward_validation(data, n_test, config) for _ in range(n_repeats)]
	# summarize score
	result = mean(scores)
	print('> Model[%s] %.3f' % (key, result))
	return (key, result)

# grid search configs
def grid_search(data, cfg_list, n_test):
	# evaluate configs
	scores = [repeat_evaluate(data, cfg, n_test) for cfg in cfg_list]
	# sort configs by error, asc
	scores.sort(key=lambda tup: tup[1])
	return scores

# create a list of configs to try
def model_configs():
	# define scope of configs
	n_input = [1]
	n_nodes = [64, 128]
	n_epochs = [300]
	n_batch = [32]
	n_diff = [0, 1]
	# create configs
	configs = list()
	for i in n_input:
		for j in n_nodes:
			for k in n_epochs:

	for l in n_batch:

		for m in n_diff:

			cfg = [i, j, k, l, m]



	configs.append(cfg)
	print('Total configs: %d' % len(configs))
	return configs


data = series.values
# data split
n_test = 1509
# model configs
cfg_list = model_configs()
# grid search
scores = grid_search(data, cfg_list, n_test)
print('done')
# list top 3 configs
for cfg, error in scores[:3]:
	print(cfg, error)

# grid search mlps
from math import sqrt
from numpy import array
from numpy import mean
from pandas import DataFrame
from pandas import concat
from pandas import read_csv
from sklearn.metrics import mean_squared_error
from keras.models import Sequential
from keras.layers import Dense, Dropout
from time import time

# split a univariate dataset into train/test sets
def train_test_split(data, n_test):
	return data[:n_test], data[n_test:]

# transform list into supervised learning format
def series_to_supervised(data, n_in=1, n_out=1):
	df = DataFrame(data)
	cols = list()
	# input sequence (t-n, ... t-1)
	for i in range(n_in, 0, -1):

	cols.append(df.shift(i))
	# forecast sequence (t, t+1, ... t+n)
	for i in range(0, n_out):

	cols.append(df.shift(-i))
	# put it all together
	agg = concat(cols, axis=1)
	# drop rows with NaN values
	agg.dropna(inplace=True)
	return agg.values

# root mean squared error or rmse
def measure_rmse(actual, predicted):
	return sqrt(mean_squared_error(actual, predicted))

# difference dataset
def difference(data, order):
	return [data[i] - data[i - order] for i in range(order, len(data))]

# fit a model
def model_fit(train, config):
  # unpack config
  n_input, n_nodes, n_epochs, n_batch, n_diff, activation, optimizer = config
  # prepare data
  if n_diff > 0:
    train = difference(train, n_diff)
  # transform series into supervised format
  data = series_to_supervised(train, n_in=n_input)
  # separate inputs and outputs
  train_x, train_y = data[:, :-1], data[:, -1]
  # define model
  model = Sequential()
  model.add(Dense(n_nodes, activation=activation, input_dim=n_input)),
  model.add(Dense(n_nodes // 2, activation=activation))
  model.add(Dense(1))
  model.compile(loss='mse', optimizer=optimizer)
  # fit model
  model.fit(train_x, train_y, epochs=n_epochs, batch_size=n_batch, verbose=0)
  return model

# forecast with the fit model
```

```
def model_predict(model, history, config):
	# unpack config
	n_input, _, _, _, n_diff, _, _ = config
	# prepare data
	correction = 0.0
	if n_diff > 0:
		correction = history[-n_diff]
		history = difference(history, n_diff)
	# shape input for model
	x_input = array(history[-n_input:]).reshape((1, n_input))
	# make forecast
	yhat = model.predict(x_input, verbose=0)
	# correct forecast if it was differenced
	return correction + yhat[0]

# walk-forward validation for univariate data
def walk_forward_validation(data, n_test, cfg):
 start = time()
 predictions = list()
 # split dataset
 train, test = train_test_split(data, n_test)
 # fit model
 model = model_fit(train, cfg)
 # seed history with training dataset
 history = [x for x in train]
 # step over each time-step in the test set
 for i in range(len(test)):
  # fit model and make forecast for history
  yhat = model_predict(model, history, cfg)
  if pd.isna(yhat) or (yhat > 999999):
   yhat = 999999
  # store forecast in list of predictions
  predictions.append(yhat)
  # add actual observation to history for the next loop
  history.append(test[i])
 # estimate prediction error
 error = measure_rmse(test, predictions)
 end = time()
 print(' > %.3f | %.3fs' % (error, end - start))
 return error

# score a model, return None on failure
def repeat_evaluate(data, config, n_test, n_repeats=10):
	# convert config to a key
	key = str(config)
	# fit and evaluate the model n times
	scores = [walk_forward_validation(data, n_test, config) for _ in range(n_repeats)]
	# summarize score
	result = mean(scores)
	print('> Model[%s] %.3f' % (key, result))
	return (key, result)

# grid search configs
def grid_search(data, cfg_list, n_test):
	# evaluate configs
	scores = [repeat_evaluate(data, cfg, n_test) for cfg in cfg_list]
	# sort configs by error, asc
	scores.sort(key=lambda tup: tup[1])
	return scores

# create a list of configs to try
def model_configs():
	# define scope of configs
 n_input = [1]
 n_nodes = [128]
 n_epochs = [300]
 n_batch = [32]
 n_diff = [0]
 activations = ['elu', 'linear', 'relu', 'selu', 'sigmoid', 'swish']
 optimizers = ['adadelta', 'adagrad', 'adam', 'nadam', 'rmsprop', 'sgd']
	# create configs
 configs = list()
 for i in n_input:
  for j in n_nodes:
   for k in n_epochs:
    for l in n_batch:
     for m in n_diff:
      for n in activations:
       for o in optimizers:
        cfg = [i, j, k, l, m, n, o]
        configs.append(cfg)
 print('Total configs: %d' % len(configs))
 return configs


data = series.values
# data split
n_test = 1509
# model configs
cfg_list = model_configs()
# grid search
scores = grid_search(data, cfg_list, n_test)
print('done')
# list top 3 configs
for cfg, error in scores[:3]:
	print(cfg, error)

"""# 3. CNN"""

# grid search cnn for airline passengers
from math import sqrt
from numpy import array
from numpy import mean
from pandas import DataFrame
from pandas import concat
from pandas import read_csv
from sklearn.metrics import mean_squared_error
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import Flatten
from keras.layers.convolutional import Conv1D
from keras.layers.convolutional import MaxPooling1D

# split a univariate dataset into train/test sets
def train_test_split(data, n_test):
	return data[:n_test], data[n_test:]

# transform list into supervised learning format
def series_to_supervised(data, n_in=1, n_out=1):
	df = DataFrame(data)
	cols = list()
	# input sequence (t-n, ... t-1)
	for i in range(n_in, 0, -1):
		cols.append(df.shift(i))
	# forecast sequence (t, t+1, ... t+n)
	for i in range(0, n_out):

		cols.append(df.shift(-i))
	# put it all together
	agg = concat(cols, axis=1)
	# drop rows with NaN values
	agg.dropna(inplace=True)
	return agg.values

# root mean squared error or rmse
def measure_rmse(actual, predicted):
	return sqrt(mean_squared_error(actual, predicted))

# difference dataset
def difference(data, order):
	return [data[i] - data[i - order] for i in range(order, len(data))]

# fit a model
def model_fit(train, config):
	# unpack config
	n_input, n_filters, n_kernel, n_epochs, n_batch, n_diff = config
	# prepare data
	if n_diff > 0:
		train = difference(train, n_diff)
	# transform series into supervised format
	data = series_to_supervised(train, n_in=n_input)
	# separate inputs and outputs
	train_x, train_y = data[:, :-1], data[:, -1]
	# reshape input data into [samples, timesteps, features]
	n_features = 1
	train_x = train_x.reshape((train_x.shape[0], train_x.shape[1], n_features))
	# define model
	model = Sequential()
	model.add(Conv1D(filters=n_filters, kernel_size=n_kernel, activation='relu', input_shape=(n_input, n_features)))
	model.add(MaxPooling1D(pool_size=2))
	model.add(Flatten())
	model.add(Dense(1))
	model.compile(loss='mse', optimizer='adam')
	# fit
	model.fit(train_x, train_y, epochs=n_epochs, batch_size=n_batch, verbose=0)
	return model

# forecast with the fit model
def model_predict(model, history, config):
	# unpack config
	n_input, _, _, _, _, n_diff = config
	# prepare data
	correction = 0.0
	if n_diff > 0:
		correction = history[-n_diff]
		history = difference(history, n_diff)
```

```
            x_input = array(history[-n_input:]).reshape((1, n_input, 1))
            # forecast
            yhat = model.predict(x_input, verbose=0)
            return correction + yhat[0]

# walk-forward validation for univariate data
def walk_forward_validation(data, n_test, cfg):
  start = time()
  predictions = list()
  # split dataset
  train, test = train_test_split(data, n_test)
  # fit model
  model = model_fit(train, cfg)
  # seed history with training dataset
  history = [x for x in train]
  # step over each time-step in the test set
  for i in range(len(test)):
            # fit model and make forecast for history
    yhat = model_predict(model, history, cfg)
                        # store forecast in list of predictions
    predictions.append(yhat)
                        # add actual observation to history for the next loop
    history.append(test[i])
            # estimate prediction error
  error = measure_rmse(test, predictions)
  end = time()
  print(' > %.3f | %.3f' % (error, end - start))
  return error

# score a model, return None on failure
def repeat_evaluate(data, config, n_test, n_repeats=3):
            # convert config to a key
  key = str(config)
            # fit and evaluate the model n times
  try:
    scores = [walk_forward_validation(data, n_test, config) for _ in range(n_repeats)]
    # summarize score
    result = mean(scores)
  except ValueError:
    result = 999.9
  print('> Model[%s] %.3f' % (key, result))
  return (key, result)

# grid search configs
def grid_search(data, cfg_list, n_test):
            # evaluate configs
            scores = [repeat_evaluate(data, cfg, n_test) for cfg in cfg_list]
            # sort configs by error, asc
            scores.sort(key=lambda tup: tup[1])
            return scores

# create a list of configs to try
def model_configs():
            # define scope of configs
            n_input = [4, 8, 12]
            n_filters = [32, 64]
            n_kernels = [3, 5]
            n_epochs = [100]
            n_batch = [32, 64]
            n_diff = [0, 1, 8]
            # create configs
            configs = list()
            for a in n_input:
                        for b in n_filters:
                                    for c in n_kernels:

                                                for d in n_epochs:

                                                            for e in n_batch:

                                                                        for f in n_diff:



                                                                                    cfg = [a,b,c,d,e,f]



                                                                                    configs.append(cfg)
            print('Total configs: %d' % len(configs))
            return configs

# define dataset
data = series.values
# data split
n_test = 1509
# model configs
cfg_list = model_configs()
# grid search
scores = grid_search(data, cfg_list, n_test)
print('done')
# list top 10 configs
for cfg, error in scores[:3]:
            print(cfg, error)

# grid search cnn for airline passengers
from math import sqrt
from numpy import array
from numpy import mean
from pandas import DataFrame
from pandas import concat
from pandas import read_csv
from sklearn.metrics import mean_squared_error
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import Flatten
from keras.layers.convolutional import Conv1D
from keras.layers.convolutional import MaxPooling1D

# split a univariate dataset into train/test sets
def train_test_split(data, n_test):
            return data[:n_test], data[n_test:]

# transform list into supervised learning format
def series_to_supervised(data, n_in=1, n_out=1):
            df = DataFrame(data)
            cols = list()
            # input sequence (t-n, ... t-1)
            for i in range(n_in, 0, -1):

                        cols.append(df.shift(i))
            # forecast sequence (t, t+1, ... t+n)
            for i in range(0, n_out):

                        cols.append(df.shift(-i))
            # put it all together
            agg = concat(cols, axis=1)
            # drop rows with NaN values
            agg.dropna(inplace=True)
            return agg.values

# root mean squared error or rmse
def measure_rmse(actual, predicted):
            return sqrt(mean_squared_error(actual, predicted))

# difference dataset
def difference(data, order):
            return [data[i] - data[i - order] for i in range(order, len(data))]

# fit a model
def model_fit(train, config):
            # unpack config
            n_input, n_filters, n_kernel, n_epochs, n_batch, n_diff = config
            # prepare data
            if n_diff > 0:
                        train = difference(train, n_diff)
            # transform series into supervised format
            data = series_to_supervised(train, n_in=n_input)
            # separate inputs and outputs
            train_x, train_y = data[:, :-1], data[:, -1]
            # reshape input data into [samples, timesteps, features]
            n_features = 1
            train_x = train_x.reshape((train_x.shape[0], train_x.shape[1], n_features))
            # define model
            model = Sequential()
            model.add(Conv1D(filters=n_filters, kernel_size=n_kernel, activation='relu', input_shape=(n_input, n_features)))
            model.add(MaxPooling1D(pool_size=2))
            model.add(Flatten())
            model.add(Dense(1))
            model.compile(loss='mse', optimizer='adam')
            # fit
            model.fit(train_x, train_y, epochs=n_epochs, batch_size=n_batch, verbose=0)
            return model

# forecast with the fit model
def model_predict(model, history, config):
            # unpack config
            n_input, _, _, _, _, n_diff = config
            # prepare data
            correction = 0.0
            if n_diff > 0:
                        correction = history[-n_diff]
                        history = difference(history, n_diff)
            x_input = array(history[-n_input:]).reshape((1, n_input, 1))
            # forecast
            yhat = model.predict(x_input, verbose=0)
            return correction + yhat[0]

# walk-forward validation for univariate data
def walk_forward_validation(data, n_test, cfg):
```

```
 start = time()
 predictions = list()
 # split dataset
 train, test = train_test_split(data, n_test)
 # fit model
 model = model_fit(train, cfg)
 # seed history with training dataset
 history = [x for x in train]
 # step over each time-step in the test set
 for i in range(len(test)):
        # fit model and make forecast for history
   yhat = model_predict(model, history, cfg)
        # store forecast in list of predictions
   predictions.append(yhat)
        # add actual observation to history for the next loop
   history.append(test[i])
        # estimate prediction error
 error = measure_rmse(test, predictions)
 end = time()
 print(' > %.3f | %.3f' % (error, end - start))
 return error

# score a model, return None on failure
def repeat_evaluate(data, config, n_test, n_repeats=10):
        # convert config to a key
 key = str(config)
        # fit and evaluate the model n times
 try:
   scores = [walk_forward_validation(data, n_test, config) for _ in range(n_repeats)]
   # summarize score
   result = mean(scores)
 except ValueError:
   result = 999.9
 print('> Model[%s] %.3f' % (key, result))
 return (key, result)

# grid search configs
def grid_search(data, cfg_list, n_test):
        # evaluate configs
        scores = [repeat_evaluate(data, cfg, n_test) for cfg in cfg_list]
        # sort configs by error, asc
        scores.sort(key=lambda tup: tup[1])
        return scores

# create a list of configs to try
def model_configs():
        # define scope of configs
        n_input = [4, 8]
        n_filters = [32, 64]
        n_kernels = [3]
        n_epochs = [300]
        n_batch = [32, 64]
        n_diff = [1]
        # create configs
        configs = list()
        for a in n_input:
                for b in n_filters:
                        for c in n_kernels:

        for d in n_epochs:

                for e in n_batch:
                        for f in n_diff:


        cfg = [a,b,c,d,e,f]


        configs.append(cfg)
        print('Total configs: %d' % len(configs))
        return configs


# define dataset
data = series.values
# data split
n_test = 1509
# model configs
cfg_list = model_configs()
# grid search
scores = grid_search(data, cfg_list, n_test)
print('done')
# list top 10 configs
for cfg, error in scores[:3]:
        print(cfg, error)

# grid search cnn for airline passengers
from math import sqrt
from numpy import array
from numpy import mean
from pandas import DataFrame
from pandas import concat
from pandas import read_csv
from sklearn.metrics import mean_squared_error
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import Flatten
from keras.layers.convolutional import Conv1D
from keras.layers.convolutional import MaxPooling1D

# split a univariate dataset into train/test sets
def train_test_split(data, n_test):
        return data[:n_test], data[n_test:]

# transform list into supervised learning format
def series_to_supervised(data, n_in=1, n_out=1):
        df = DataFrame(data)
        cols = list()
        # input sequence (t-n, ... t-1)
        for i in range(n_in, 0, -1):

        cols.append(df.shift(i))
        # forecast sequence (t, t+1, ... t+n)
        for i in range(0, n_out):

        cols.append(df.shift(-i))
        # put it all together
        agg = concat(cols, axis=1)
        # drop rows with NaN values
        agg.dropna(inplace=True)
        return agg.values

# root mean squared error or rmse
def measure_rmse(actual, predicted):
        return sqrt(mean_squared_error(actual, predicted))

# difference dataset
def difference(data, order):
        return [data[i] - data[i - order] for i in range(order, len(data))]

# fit a model
def model_fit(train, config):
        # unpack config
        n_input, n_filters, n_kernel, n_epochs, n_batch, n_diff, activation, optimizer = config
        # prepare data
        if n_diff > 0:
                train = difference(train, n_diff)
        # transform series into supervised format
        data = series_to_supervised(train, n_in=n_input)
        # separate inputs and outputs
        train_x, train_y = data[:, :-1], data[:, -1]
        # reshape input data into [samples, timesteps, features]
        n_features = 1
        train_x = train_x.reshape((train_x.shape[0], train_x.shape[1], n_features))
        # define model
        model = Sequential()
        model.add(Conv1D(filters=n_filters, kernel_size=n_kernel, activation=activation, input_shape=(n_input, n_features)))
        model.add(MaxPooling1D(pool_size=2))
        model.add(Flatten())
        model.add(Dense(1))
        model.compile(loss='mse', optimizer=optimizer)
        # fit
        model.fit(train_x, train_y, epochs=n_epochs, batch_size=n_batch, verbose=0)
        return model

# forecast with the fit model
def model_predict(model, history, config):
        # unpack config
        n_input, _, _, _, _, n_diff, _, _ = config
        # prepare data
        correction = 0.0
        if n_diff > 0:
                correction = history[-n_diff]
                history = difference(history, n_diff)
        x_input = array(history[-n_input:]).reshape((1, n_input, 1))
        # forecast
        yhat = model.predict(x_input, verbose=0)
        return correction + yhat[0]

# walk-forward validation for univariate data
def walk_forward_validation(data, n_test, cfg):
 start = time()
 predictions = list()
 # split dataset
 train, test = train_test_split(data, n_test)
 # fit model
 model = model_fit(train, cfg)
 # seed history with training dataset
```

```
 history = [x for x in train]
 # step over each time-step in the test set
 for i in range(len(test)):
		# fit model and make forecast for history
   yhat = model_predict(model, history, cfg)
   if pd.isna(yhat) or (yhat > 999999):
    yhat = 999999
			# store forecast in list of predictions
   predictions.append(yhat)
			# add actual observation to history for the next loop
   history.append(test[i])
		# estimate prediction error
 error = measure_rmse(test, predictions)
 end = time()
 print(' > %.3f | %.3f' % (error, end - start))
 return error

# score a model, return None on failure
def repeat_evaluate(data, config, n_test, n_repeats=10):
		# convert config to a key
 key = str(config)
		# fit and evaluate the model n times
 try:
  scores = [walk_forward_validation(data, n_test, config) for _ in range(n_repeats)]
  # summarize score
  result = mean(scores)
 except ValueError:
  result = 999.9
 print('> Model[%s] %.3f' % (key, result))
 return (key, result)

# grid search configs
def grid_search(data, cfg_list, n_test):
		# evaluate configs
		scores = [repeat_evaluate(data, cfg, n_test) for cfg in cfg_list]
		# sort configs by error, asc
		scores.sort(key=lambda tup: tup[1])
		return scores

# create a list of configs to try
def model_configs():
		# define scope of configs
 n_input = [4]
 n_filters = [32]
 n_kernels = [3]
 n_epochs = [300]
 n_batch = [64]
 n_diff = [1]
 activations = ['elu', 'linear', 'relu', 'selu', 'sigmoid', 'swish']
 optimizers = ['adadelta', 'adagrad', 'adam', 'nadam', 'rmsprop', 'sgd']
		# create configs
 configs = list()
 for a in n_input:
  for b in n_filters:
   for c in n_kernels:
    for d in n_epochs:
     for e in n_batch:
      for f in n_diff:
       for g in activations:
        for l in optimizers:
         cfg = [a,b,c,d,e,f,g,l]
         configs.append(cfg)
 print('Total configs: %d' % len(configs))
 return configs


# define dataset
data = series.values
# data split
n_test = 1509
# model configs
cfg_list = model_configs()
# grid search
scores = grid_search(data, cfg_list, n_test)
print('done')
# list top 10 configs
for cfg, error in scores[:3]:
		print(cfg, error)

"""# 4. RNN"""

# grid search lstm for airline passengers
from math import sqrt
from numpy import array
from numpy import mean
from pandas import DataFrame
from pandas import concat
from pandas import read_csv
from sklearn.metrics import mean_squared_error
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
from time import time

# split a univariate dataset into train/test sets
def train_test_split(data, n_test):
		return data[:n_test], data[n_test:]

# transform list into supervised learning format
def series_to_supervised(data, n_in=1, n_out=1):
		df = DataFrame(data)
		cols = list()
		# input sequence (t-n, ... t-1)
		for i in range(n_in, 0, -1):

		cols.append(df.shift(i))
		# forecast sequence (t, t+1, ... t+n)
		for i in range(0, n_out):

		cols.append(df.shift(-i))
		# put it all together
		agg = concat(cols, axis=1)
		# drop rows with NaN values
		agg.dropna(inplace=True)
		return agg.values

# root mean squared error or rmse
def measure_rmse(actual, predicted):
		return sqrt(mean_squared_error(actual, predicted))

# difference dataset
def difference(data, order):
		return [data[i] - data[i - order] for i in range(order, len(data))]

# fit a model
def model_fit(train, config):
		# unpack config
		n_input, n_nodes, n_epochs, n_batch, n_diff = config
		# prepare data
		if n_diff > 0:
			train = difference(train, n_diff)
		# transform series into supervised format
		data = series_to_supervised(train, n_in=n_input)
		# separate inputs and outputs
		train_x, train_y = data[:, :-1], data[:, -1]
		# reshape input data into [samples, timesteps, features]
		n_features = 1
		train_x = train_x.reshape((train_x.shape[0], train_x.shape[1], n_features))
		# define model
		model = Sequential()
		model.add(LSTM(n_nodes, activation='relu', input_shape=(n_input, n_features)))
		model.add(Dense(n_nodes, activation='relu'))
		model.add(Dense(1))
		model.compile(loss='mse', optimizer='adam')
		# fit model
		model.fit(train_x, train_y, epochs=n_epochs, batch_size=n_batch, verbose=0)
		return model

# forecast with the fit model
def model_predict(model, history, config):
		# unpack config
		n_input, _, _, _, n_diff = config
		# prepare data
		correction = 0.0
		if n_diff > 0:
			correction = history[-n_diff]
			history = difference(history, n_diff)
		# reshape sample into [samples, timesteps, features]
		x_input = array(history[-n_input:]).reshape((1, n_input, 1))
		# forecast
		yhat = model.predict(x_input, verbose=0)
		return correction + yhat[0]

# walk-forward validation for univariate data
def walk_forward_validation(data, n_test, cfg):
 start = time()
 predictions = list()
		# split dataset
 train, test = train_test_split(data, n_test)
		# fit model
 model = model_fit(train, cfg)
		# seed history with training dataset
 history = [x for x in train]
		# step over each time-step in the test set
 for i in range(len(test)):
			# fit model and make forecast for history
   yhat = model_predict(model, history, cfg)
			# store forecast in list of predictions
   predictions.append(yhat)
```

```
                # add actual observation to history for the next loop
    history.append(test[i])
            # estimate prediction error
  error = measure_rmse(test, predictions)
  end = time()
  print(' > %.3f | %.3f' % (error, end - start))
  return error

# score a model, return None on failure
def repeat_evaluate(data, config, n_test, n_repeats=3):
            # convert config to a key
            key = str(config)
            # fit and evaluate the model n times
            scores = [walk_forward_validation(data, n_test, config) for _ in range(n_repeats)]
            # summarize score
            result = mean(scores)
            print('> Model[%s] %.3f' % (key, result))
            return (key, result)

# grid search configs
def grid_search(data, cfg_list, n_test):
            # evaluate configs
            scores = [repeat_evaluate(data, cfg, n_test) for cfg in cfg_list]
            # sort configs by error, asc
            scores.sort(key=lambda tup: tup[1])
            return scores

# create a list of configs to try
def model_configs():
            # define scope of configs
            n_input = [4, 8, 12]
            n_nodes = [64, 128]
            n_epochs = [100]
            n_batch = [32, 64]
            n_diff = [0, 1, 8]
            # create configs
            configs = list()
            for i in n_input:
                        for j in n_nodes:
                                    for k in n_epochs:

            for l in n_batch:

                        for m in n_diff:

                                    cfg = [i, j, k, l, m]



            configs.append(cfg)
            print('Total configs: %d' % len(configs))
            return configs

# define dataset
data = series.values
# data split
n_test = 1509
# model configs
cfg_list = model_configs()
# grid search
scores = grid_search(data, cfg_list, n_test)
print('done')
# list top 10 configs
for cfg, error in scores[:3]:
            print(cfg, error)
```

```
# grid search lstm for airline passengers
from math import sqrt
from numpy import array
from numpy import mean
from pandas import DataFrame
from pandas import concat
from pandas import read_csv
from sklearn.metrics import mean_squared_error
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
from time import time

# split a univariate dataset into train/test sets
def train_test_split(data, n_test):
            return data[:n_test], data[n_test:]

# transform list into supervised learning format
def series_to_supervised(data, n_in=1, n_out=1):
            df = DataFrame(data)
            cols = list()
            # input sequence (t-n, ... t-1)
            for i in range(n_in, 0, -1):

            cols.append(df.shift(i))
            # forecast sequence (t, t+1, ... t+n)
            for i in range(0, n_out):

            cols.append(df.shift(-i))
            # put it all together
            agg = concat(cols, axis=1)
            # drop rows with NaN values
            agg.dropna(inplace=True)
            return agg.values

# root mean squared error or rmse
def measure_rmse(actual, predicted):
            return sqrt(mean_squared_error(actual, predicted))

# difference dataset
def difference(data, order):
            return [data[i] - data[i - order] for i in range(order, len(data))]

# fit a model
def model_fit(train, config):
            # unpack config
            n_input, n_nodes, n_epochs, n_batch, n_diff = config
            # prepare data
            if n_diff > 0:
                        train = difference(train, n_diff)
            # transform series into supervised format
            data = series_to_supervised(train, n_in=n_input)
            # separate inputs and outputs
            train_x, train_y = data[:, :-1], data[:, -1]
            # reshape input data into [samples, timesteps, features]
            n_features = 1
            train_x = train_x.reshape((train_x.shape[0], train_x.shape[1], n_features))
            # define model
            model = Sequential()
            model.add(LSTM(n_nodes, activation='relu', input_shape=(n_input, n_features)))
            model.add(Dense(n_nodes, activation='relu'))
            model.add(Dense(1))
            model.compile(loss='mse', optimizer='adam')
            # fit model
            model.fit(train_x, train_y, epochs=n_epochs, batch_size=n_batch, verbose=0)
            return model

# forecast with the fit model
def model_predict(model, history, config):
            # unpack config
            n_input, _, _, _, n_diff = config
            # prepare data
            correction = 0.0
            if n_diff > 0:
                        correction = history[-n_diff]
                        history = difference(history, n_diff)
            # reshape sample into [samples, timesteps, features]
            x_input = array(history[-n_input:]).reshape((1, n_input, 1))
            # forecast
            yhat = model.predict(x_input, verbose=0)
            return correction + yhat[0]

# walk-forward validation for univariate data
def walk_forward_validation(data, n_test, cfg):
  start = time()
  predictions = list()
            # split dataset
  train, test = train_test_split(data, n_test)
            # fit model
  model = model_fit(train, cfg)
            # seed history with training dataset
  history = [x for x in train]
            # step over each time-step in the test set
  for i in range(len(test)):
                        # fit model and make forecast for history
    yhat = model_predict(model, history, cfg)
                        # store forecast in list of predictions
    predictions.append(yhat)
                        # add actual observation to history for the next loop
    history.append(test[i])
            # estimate prediction error
  error = measure_rmse(test, predictions)
  end = time()
  print(' > %.3f | %.3f' % (error, end - start))
  return error

# score a model, return None on failure
def repeat_evaluate(data, config, n_test, n_repeats=10):
            # convert config to a key
            key = str(config)
            # fit and evaluate the model n times
```

```
        scores = [walk_forward_validation(data, n_test, config) for _ in range(n_repeats)]
        # summarize score
        result = mean(scores)
        print('> Model[%s] %.3f' % (key, result))
        return (key, result)

# grid search configs
def grid_search(data, cfg_list, n_test):
        # evaluate configs
        scores = [repeat_evaluate(data, cfg, n_test) for cfg in cfg_list]
        # sort configs by error, asc
        scores.sort(key=lambda tup: tup[1])
        return scores

# create a list of configs to try
def model_configs():
        # define scope of configs
        n_input = [4, 8, 12]
        n_nodes = [64, 128]
        n_epochs = [300]
        n_batch = [32, 64]
        n_diff = [0]
        # create configs
        configs = list()
        for i in n_input:
                for j in n_nodes:
                        for k in n_epochs:

        for l in n_batch:

                for m in n_diff:

                        cfg = [i, j, k, l, m]



        configs.append(cfg)
        print('Total configs: %d' % len(configs))
        return [[8, 64, 100, 32, 0], [12, 128, 100, 64, 0], [4, 64, 100, 32, 0]]



# define dataset
data = series.values
# data split
n_test = 1509
# model configs
cfg_list = model_configs()
# grid search
scores = grid_search(data, cfg_list, n_test)
print('done')
# list top 10 configs
for cfg, error in scores[:3]:
        print(cfg, error)

# grid search lstm for airline passengers
from math import sqrt
from numpy import array
from numpy import mean
from pandas import DataFrame
from pandas import concat
from pandas import read_csv
from sklearn.metrics import mean_squared_error
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
from time import time

# split a univariate dataset into train/test sets
def train_test_split(data, n_test):
        return data[:n_test], data[n_test:]

# transform list into supervised learning format
def series_to_supervised(data, n_in=1, n_out=1):
        df = DataFrame(data)
        cols = list()
        # input sequence (t-n, ... t-1)
        for i in range(n_in, 0, -1):

        cols.append(df.shift(i))
        # forecast sequence (t, t+1, ... t+n)
        for i in range(0, n_out):

        cols.append(df.shift(-i))
        # put it all together
        agg = concat(cols, axis=1)
        # drop rows with NaN values
        agg.dropna(inplace=True)
        return agg.values

# root mean squared error or rmse
def measure_rmse(actual, predicted):
        return sqrt(mean_squared_error(actual, predicted))

# difference dataset
def difference(data, order):
        return [data[i] - data[i - order] for i in range(order, len(data))]

# fit a model
def model_fit(train, config):
        # unpack config
        n_input, n_nodes, n_epochs, n_batch, n_diff, activation, optimizer = config
        # prepare data
        if n_diff > 0:
                train = difference(train, n_diff)
        # transform series into supervised format
        data = series_to_supervised(train, n_in=n_input)
        # separate inputs and outputs
        train_x, train_y = data[:, :-1], data[:, -1]
        # reshape input data into [samples, timesteps, features]
        n_features = 1
        train_x = train_x.reshape((train_x.shape[0], train_x.shape[1], n_features))
        # define model
        model = Sequential()
        model.add(LSTM(n_nodes, activation=activation, input_shape=(n_input, n_features)))
        model.add(Dense(n_nodes, activation=activation))
        model.add(Dense(1))
        model.compile(loss='mse', optimizer=optimizer)
        # fit model
        model.fit(train_x, train_y, epochs=n_epochs, batch_size=n_batch, verbose=0)
        return model

# forecast with the fit model
def model_predict(model, history, config):
        # unpack config
        n_input, _, _, _, n_diff, _, _ = config
        # prepare data
        correction = 0.0
        if n_diff > 0:
                correction = history[-n_diff]
                history = difference(history, n_diff)
        # reshape sample into [samples, timesteps, features]
        x_input = array(history[-n_input:]).reshape((1, n_input, 1))
        # forecast
        yhat = model.predict(x_input, verbose=0)
        return correction + yhat[0]

# walk-forward validation for univariate data
def walk_forward_validation(data, n_test, cfg):
  start = time()
  predictions = list()
        # split dataset
  train, test = train_test_split(data, n_test)
        # fit model
  model = model_fit(train, cfg)
        # seed history with training dataset
  history = [x for x in train]
        # step over each time-step in the test set
  for i in range(len(test)):
                # fit model and make forecast for history
    yhat = model_predict(model, history, cfg)
    if pd.isna(yhat) or (yhat > 999999):
      yhat = 999999
                # store forecast in list of predictions
    predictions.append(yhat)
                # add actual observation to history for the next loop
    history.append(test[i])
        # estimate prediction error
  error = measure_rmse(test, predictions)
  end = time()
  print(' > %.3f | %.3f' % (error, end - start))
  return error

# score a model, return None on failure
def repeat_evaluate(data, config, n_test, n_repeats=10):
        # convert config to a key
        key = str(config)
        # fit and evaluate the model n times
        scores = [walk_forward_validation(data, n_test, config) for _ in range(n_repeats)]
        # summarize score
        result = mean(scores)
        print('> Model[%s] %.3f' % (key, result))
        return (key, result)

# grid search configs
def grid_search(data, cfg_list, n_test):
        # evaluate configs
```

```
        scores = [repeat_evaluate(data, cfg, n_test) for cfg in cfg_list]
        # sort configs by error, asc
        scores.sort(key=lambda tup: tup[1])
        return scores

# create a list of configs to try
def model_configs():
        # define scope of configs
  n_input = [8]
  n_nodes = [64]
  n_epochs = [300]
  n_batch = [32]
  n_diff = [0]
  activations = ['linear', 'relu', 'selu', 'swish']
  optimizers = ['adagrad', 'adam', 'rmsprop', 'sgd']

        # create configs
  configs = list()
  for i in n_input:
    for j in n_nodes:
      for k in n_epochs:
        for l in n_batch:
          for m in n_diff:
            for n in activations:
              for o in optimizers:
                cfg = [i, j, k, l, m, n, o]
                configs.append(cfg)
  print('Total configs: %d' % len(configs))
  return configs



# define dataset
data = series.values
# data split
n_test = 1509
# model configs
cfg_list = model_configs()
# grid search
scores = grid_search(data, cfg_list, n_test)
print('done')
# list top 10 configs
for cfg, error in scores[:3]:
        print(cfg, error)

import logging
import tensorflow as tf

tf.get_logger().setLevel(logging.ERROR)

# grid search lstm for airline passengers
from math import sqrt
from numpy import array
from numpy import mean
from pandas import DataFrame
from pandas import concat
from pandas import read_csv
from sklearn.metrics import mean_squared_error
from keras.models import Sequential
from keras.layers import Dense
from keras.layers.convolutional import Conv1D
from keras.layers import LSTM
from time import time

# split a univariate dataset into train/test sets
def train_test_split(data, n_test):
        return data[:n_test], data[n_test:]

# transform list into supervised learning format
def series_to_supervised(data, n_in=1, n_out=1):
        df = DataFrame(data)
        cols = list()
        # input sequence (t-n, ... t-1)
        for i in range(n_in, 0, -1):

        cols.append(df.shift(i))
        # forecast sequence (t, t+1, ... t+n)
        for i in range(0, n_out):

        cols.append(df.shift(-i))
        # put it all together
        agg = concat(cols, axis=1)
        # drop rows with NaN values
        agg.dropna(inplace=True)
        return agg.values

# root mean squared error or rmse
def measure_rmse(actual, predicted):
        return sqrt(mean_squared_error(actual, predicted))

# difference dataset
def difference(data, order):
        return [data[i] - data[i - order] for i in range(order, len(data))]

# fit a model
def model_fit(train, config):
        # unpack config
  n_input, n_nodes, n_epochs, n_batch, n_diff, n_filters, n_kernel, activation, optimizer = config
        # prepare data
  if n_diff > 0:
    train = difference(train, n_diff)
        # transform series into supervised format
  data = series_to_supervised(train, n_in=n_input)
        # separate inputs and outputs
  train_x, train_y = data[:, :-1], data[:, -1]
        # reshape input data into [samples, timesteps, features]
  n_features = 1
  train_x = train_x.reshape((train_x.shape[0], train_x.shape[1], n_features))
        # define model
  model = Sequential()
  model.add(Conv1D(filters=n_filters, kernel_size=n_kernel, input_shape=(n_input, n_features)))
  model.add(LSTM(n_nodes, activation=activation, return_sequences=True))
  model.add(LSTM(n_nodes, activation=activation))
  model.add(Dense(n_nodes // 2, activation=activation)),
  model.add(Dense(n_nodes // 4, activation=activation)),
  model.add(Dense(1))

  model.compile(loss='mse', optimizer='adam')
        # fit model
  model.fit(train_x, train_y, epochs=n_epochs, batch_size=n_batch, verbose=0)
  return model

# forecast with the fit model
def model_predict(model, history, config):
  #print(config)
        # unpack config
  n_input, _, _, _, n_diff, _, _, _, _ = config
        # prepare data
  correction = 0.0
  if n_diff > 0:
    correction = history[-n_diff]
    history = difference(history, n_diff)
        # reshape sample into [samples, timesteps, features]
  x_input = array(history[-n_input:]).reshape((1, n_input, 1))
        # forecast
  yhat = model.predict(x_input, verbose=0)

  return correction + yhat[0]

# walk-forward validation for univariate data
def walk_forward_validation(data, n_test, cfg):
  start = time()
  predictions = list()
        # split dataset
  train, test = train_test_split(data, n_test)
        # fit model
  model = model_fit(train, cfg)
        # seed history with training dataset
  history = [x for x in train]
        # step over each time-step in the test set
  for i in range(len(test)):
                # fit model and make forecast for history
    yhat = model_predict(model, history, cfg)
                # store forecast in list of predictions
    predictions.append(yhat)
                # add actual observation to history for the next loop
    history.append(test[i])
        # estimate prediction error
  error = measure_rmse(test, predictions)
  end = time()
  print(' > %.3f | %.3f' % (error, end - start))
  return error

# score a model, return None on failure
def repeat_evaluate(data, config, n_test, n_repeats=3):
        # convert config to a key
  key = str(config)
        # fit and evaluate the model n times
  try:
    scores = [walk_forward_validation(data, n_test, config) for _ in range(n_repeats)]
    # summarize score
    result = mean(scores)
  except ValueError:
    result = 999.9
  print('> Model[%s] %.3f' % (key, result))
  return (key, result)

# grid search configs
def grid_search(data, cfg_list, n_test):
        # evaluate configs
```

```
		scores = [repeat_evaluate(data, cfg, n_test) for cfg in cfg_list]
		# sort configs by error, asc
		scores.sort(key=lambda tup: tup[1])
		return scores

# create a list of configs to try
def model_configs():
		# define scope of configs
  n_input = [4, 8, 30]
  n_nodes = [64, 128]
  n_epochs = [100]
  n_batch = [32]
  n_diff = [0, 1, 8]
  n_filters = [32, 64]
  n_kernel = [3, 5]
  activations = ['relu', 'linear', 'swish']
  optimizers = ['adam']
		# create configs
  configs = list()
  for i in n_input:
    for j in n_nodes:
      for k in n_epochs:
        for l in n_batch:
          for m in n_diff:
            for n in n_filters:
              for o in n_kernel:
                for p in activations:
                  for q in optimizers:
                    cfg = [i, j, k, l, m, n, o, p, q]
                    configs.append(cfg)
  print('Total configs: %d' % len(configs))
  return configs

# define dataset
data = series.values
# data split
n_test = 1509
# model configs
cfg_list = model_configs()
# grid search
scores = grid_search(data, cfg_list, n_test)
print('done')
# list top 10 configs
for cfg, error in scores[:3]:
  print(cfg, error)

# grid search lstm for airline passengers
from math import sqrt
from numpy import array
from numpy import mean
from pandas import DataFrame
from pandas import concat
from pandas import read_csv
from sklearn.metrics import mean_squared_error
from keras.models import Sequential
from keras.layers import Dense
from keras.layers.convolutional import Conv1D
from keras.layers import LSTM
from time import time

# split a univariate dataset into train/test sets
def train_test_split(data, n_test):
		return data[:n_test], data[n_test:]

# transform list into supervised learning format
def series_to_supervised(data, n_in=1, n_out=1):
		df = DataFrame(data)
		cols = list()
		# input sequence (t-n, ... t-1)
		for i in range(n_in, 0, -1):

		cols.append(df.shift(i))
		# forecast sequence (t, t+1, ... t+n)
		for i in range(0, n_out):

		cols.append(df.shift(-i))
		# put it all together
		agg = concat(cols, axis=1)
		# drop rows with NaN values
		agg.dropna(inplace=True)
		return agg.values

# root mean squared error or rmse
def measure_rmse(actual, predicted):
		return sqrt(mean_squared_error(actual, predicted))

# difference dataset
def difference(data, order):
		return [data[i] - data[i - order] for i in range(order, len(data))]

# fit a model
def model_fit(train, config):
		# unpack config
  n_input, n_nodes, n_epochs, n_batch, n_diff, n_filters, n_kernel, activation, optimizer = config
		# prepare data
  if n_diff > 0:
    train = difference(train, n_diff)
		# transform series into supervised format
  data = series_to_supervised(train, n_in=n_input)
		# separate inputs and outputs
  train_x, train_y = data[:, :-1], data[:, -1]
		# reshape input data into [samples, timesteps, features]
  n_features = 1
  train_x = train_x.reshape((train_x.shape[0], train_x.shape[1], n_features))
		# define model
  model = Sequential()
  model.add(Conv1D(filters=n_filters, kernel_size=n_kernel, input_shape=(n_input, n_features)))
  model.add(LSTM(n_nodes, activation=activation, return_sequences=True))
  model.add(LSTM(n_nodes, activation=activation))
  model.add(Dense(n_nodes // 2, activation=activation)),
  model.add(Dense(n_nodes // 4, activation=activation)),
  model.add(Dense(1))

  model.compile(loss='mse', optimizer='adam')
		# fit model
  model.fit(train_x, train_y, epochs=n_epochs, batch_size=n_batch, verbose=0)
  return model

# forecast with the fit model
def model_predict(model, history, config):
  #print(config)
		# unpack config
  n_input, _, _, _, n_diff, _, _, _, _ = config
		# prepare data
  correction = 0.0
  if n_diff > 0:
    correction = history[-n_diff]
    history = difference(history, n_diff)
		# reshape sample into [samples, timesteps, features]
  x_input = array(history[-n_input:]).reshape((1, n_input, 1))
		# forecast
  yhat = model.predict(x_input, verbose=0)

  return correction + yhat[0]

# walk-forward validation for univariate data
def walk_forward_validation(data, n_test, cfg):
  start = time()
  predictions = list()
		# split dataset
  train, test = train_test_split(data, n_test)
		# fit model
  model = model_fit(train, cfg)
		# seed history with training dataset
  history = [x for x in train]
		# step over each time-step in the test set
  for i in range(len(test)):
				# fit model and make forecast for history
    yhat = model_predict(model, history, cfg)
				# store forecast in list of predictions
    predictions.append(yhat)
				# add actual observation to history for the next loop
    history.append(test[i])
		# estimate prediction error
  error = measure_rmse(test, predictions)
  end = time()
  print(' > %.3f | %.3f' % (error, end - start))
  return error

# score a model, return None on failure
def repeat_evaluate(data, config, n_test, n_repeats=3):
		# convert config to a key
  key = str(config)
		# fit and evaluate the model n times
  try:
    scores = [walk_forward_validation(data, n_test, config) for _ in range(n_repeats)]
    # summarize score
    result = mean(scores)
  except ValueError:
    result = 999.9
  print('> Model[%s] %.3f' % (key, result))
  return (key, result)

# grid search configs
def grid_search(data, cfg_list, n_test):
		# evaluate configs
		scores = [repeat_evaluate(data, cfg, n_test) for cfg in cfg_list]
		# sort configs by error, asc
		scores.sort(key=lambda tup: tup[1])
		return scores

# create a list of configs to try
```

```
def model_configs():
            # define scope of configs
  n_input = [4, 8, 30]
  n_nodes = [64, 128]
  n_epochs = [100]
  n_batch = [32]
  n_diff = [0, 1, 8]
  n_filters = [32, 64]
  n_kernel = [3, 5]
  activations = ['relu']
  optimizers = ['adam']
            # create configs
  configs = list()
  for i in n_input:
    for j in n_nodes:
      for k in n_epochs:
        for l in n_batch:
          for m in n_diff:
            for n in n_filters:
              for o in n_kernel:
                for p in activations:
                  for q in optimizers:
                    cfg = [i, j, k, l, m, n, o, p, q]
                    configs.append(cfg)
  print('Total configs: %d' % len(configs))
  return configs

# define dataset
data = series.values
# data split
n_test = 1509
# model configs
cfg_list = model_configs()
# grid search
scores = grid_search(data, cfg_list, n_test)
print('done')
# list top 10 configs
for cfg, error in scores[:3]:
  print(cfg, error)

"""___"""

model

"""### 1. Deep Neural Networks"""

def plot_series(time, series, format="-", start=0, end=None):
    plt.plot(time[start:end], series[start:end], format)
    plt.xlabel("Time")
    plt.ylabel("Value")
    plt.grid(False)

def windowed_dataset(series, window_size, batch_size, shuffle_buffer):
  dataset = tf.data.Dataset.from_tensor_slices(series)
  dataset = dataset.window(window_size + 1, shift=1, drop_remainder=True)
  dataset = dataset.flat_map(lambda window: window.batch(window_size + 1))
  dataset = dataset.shuffle(shuffle_buffer).map(lambda window: (window[:-1], window[-1]))
  dataset = dataset.batch(batch_size).prefetch(1)
  return dataset

time = np.array(series.index)
series = np.array(series)

split_time = 1509
time_train = time[:split_time]
x_train = series[:split_time]
time_valid = time[split_time:]
x_valid = series[split_time:]

window_size = 8
batch_size = 32
shuffle_buffer_size = 500

plot_series(time, series)

# Seed value
# Apparently you may use different seed values at each stage
seed_value= 0

# 1. Set `PYTHONHASHSEED` environment variable at a fixed value
import os
os.environ['PYTHONHASHSEED']=str(seed_value)

# 2. Set `python` built-in pseudo-random generator at a fixed value
import random
random.seed(seed_value)

# 3. Set `numpy` pseudo-random generator at a fixed value
import numpy as np
np.random.seed(seed_value)

# 4. Set the `tensorflow` pseudo-random generator at a fixed value
import tensorflow as tf
tf.random.set_seed(seed_value)

dataset = windowed_dataset(x_train, window_size, batch_size, shuffle_buffer_size)
val_dataset = windowed_dataset(x_valid, window_size, batch_size, shuffle_buffer_size)
tf.random.set_seed(7)
batch_size = 128
model = tf.keras.models.Sequential([
   tf.keras.layers.Dense(100, input_shape=[window_size], activation="relu"),
   tf.keras.layers.Dense(10, activation="relu"),
   tf.keras.layers.Dense(1)
])

model.compile(loss="mse", optimizer=tf.keras.optimizers.SGD(lr=1e-6, momentum=0.9))
model.fit(dataset, epochs=200, verbose=1, validation_data=val_dataset)

loss = model.history.history['loss']
val_loss = model.history.history['val_loss']

plt.plot(loss[10:], label='loss')
plt.plot(val_loss[10:], label='val_loss')

forecast = []
for time in range(len(series) - window_size):

forecast.append(model.predict(series[time:time + window_size][np.newaxis]))

forecast = forecast[split_time-window_size:]
results = np.array(forecast)[:, 0, 0]

plt.figure(figsize=(16, 8))
plot_series(time_valid, x_valid)
plot_series(time_valid, results)

tf.keras.metrics.mean_absolute_error(x_valid, results).numpy()

tf.keras.metrics.mean_squared_error(x_valid, results).numpy()

"""___

###2. Recurrent Neural Network
"""

tf.keras.backend.clear_session()
tf.random.set_seed(51)
np.random.seed(51)

tf.keras.backend.clear_session()
dataset = windowed_dataset(x_train, window_size, batch_size, shuffle_buffer_size)

model = tf.keras.models.Sequential([
  tf.keras.layers.Lambda(lambda x: tf.expand_dims(x, axis=-1),
                      input_shape=[None]),

tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(32, return_sequences=True)),

tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(32)),
  tf.keras.layers.Dense(1),
  tf.keras.layers.Lambda(lambda x: x * 100.0)
])

lr_schedule = tf.keras.callbacks.LearningRateScheduler(
    lambda epoch: 1e-8 * 10**(epoch / 20))
optimizer = tf.keras.optimizers.SGD(lr=1e-8, momentum=0.9)
model.compile(loss=tf.keras.losses.Huber(),
              optimizer=optimizer,
              metrics=["mae"])
history = model.fit(dataset, epochs=100, callbacks=[lr_schedule])

plt.semilogx(history.history["lr"], history.history["loss"])

tf.keras.backend.clear_session()
tf.random.set_seed(51)
np.random.seed(51)

tf.keras.backend.clear_session()
dataset = windowed_dataset(x_train, window_size, batch_size, shuffle_buffer_size)

model = tf.keras.models.Sequential([
  tf.keras.layers.Lambda(lambda x: tf.expand_dims(x, axis=-1),
                      input_shape=[None]),

tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(32, return_sequences=True)),

tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(32)),
  tf.keras.layers.Dense(1),
```

```
  tf.keras.layers.Lambda(lambda x: x * 100.0)
])


model.compile(loss=tf.keras.losses.Huber(),
optimizer=tf.keras.optimizers.SGD(lr=1e-6, momentum=0.9),metrics=["mae"])
history = model.fit(dataset,epochs=500,verbose=1)

forecast = []
results = []
for time in range(len(series) - window_size):

forecast.append(model.predict(series[time:time + window_size][np.newaxis]))

forecast = forecast[split_time-window_size:]
results = np.array(forecast)[:, 0, 0]


plt.figure(figsize=(16, 8))

plot_series(time_valid, x_valid)
plot_series(time_valid, results)

tf.keras.metrics.mean_absolute_error(x_valid, results).numpy()

import matplotlib.image  as mpimg
import matplotlib.pyplot as plt

#-----------------------------------------------------------
# Retrieve a list of list results on training and test data
# sets for each training epoch
#-----------------------------------------------------------
mae=history.history['mae']
loss=history.history['loss']

epochs=range(len(loss)) # Get number of epochs

#------------------------------------------------
# Plot MAE and Loss
#------------------------------------------------
plt.plot(epochs, mae, 'r')
plt.plot(epochs, loss, 'b')
plt.title('MAE and Loss')
plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.legend(["MAE", "Loss"])

plt.figure()

epochs_zoom = epochs[200:]
mae_zoom = mae[200:]
loss_zoom = loss[200:]

#------------------------------------------------
# Plot Zoomed MAE and Loss
#------------------------------------------------
plt.plot(epochs_zoom, mae_zoom, 'r')
plt.plot(epochs_zoom, loss_zoom, 'b')
plt.title('MAE and Loss')
plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.legend(["MAE", "Loss"])

plt.figure()

"""___

###3. Hybrid Neural Network
"""

def model_forecast(model, series, window_size):
   ds = tf.data.Dataset.from_tensor_slices(series)
   ds = ds.window(window_size, shift=1, drop_remainder=True)
   ds = ds.flat_map(lambda w: w.batch(window_size))
   ds = ds.batch(32).prefetch(1)
   forecast = model.predict(ds)
   return forecast

def windowed_dataset(series, window_size, batch_size, shuffle_buffer):
   series = tf.expand_dims(series, axis=-1)
   ds = tf.data.Dataset.from_tensor_slices(series)
   ds = ds.window(window_size + 1, shift=1, drop_remainder=True)
   ds = ds.flat_map(lambda w: w.batch(window_size + 1))
   ds = ds.shuffle(shuffle_buffer)
   ds = ds.map(lambda w: (w[:-1], w[1:]))
   return ds.batch(batch_size).prefetch(1)

tf.keras.backend.clear_session()
tf.random.set_seed(51)
np.random.seed(51)
window_size = 8
batch_size = 256
train_set = windowed_dataset(x_train, window_size, batch_size, shuffle_buffer_size)
print(train_set)
print(x_train.shape)

model = tf.keras.models.Sequential([
  tf.keras.layers.Conv1D(filters=32, kernel_size=5,
                         strides=1, padding="causal",
                         activation="relu",
                         input_shape=[None, 1]),
  tf.keras.layers.LSTM(64, return_sequences=True),
  tf.keras.layers.LSTM(64, return_sequences=True),
  tf.keras.layers.Dense(30, activation="relu"),
  tf.keras.layers.Dense(10, activation="relu"),
  tf.keras.layers.Dense(1),
  tf.keras.layers.Lambda(lambda x: x * 400)
])

lr_schedule = tf.keras.callbacks.LearningRateScheduler(
   lambda epoch: 1e-8 * 10**(epoch / 20))
optimizer = tf.keras.optimizers.SGD(lr=1e-8, momentum=0.9)
model.compile(loss=tf.keras.losses.Huber(),
              optimizer=optimizer,
              metrics=["mae"])
history = model.fit(train_set, epochs=100, callbacks=[lr_schedule])

plt.semilogx(history.history["lr"], history.history["loss"])

tf.keras.backend.clear_session()
tf.random.set_seed(51)
np.random.seed(51)
window_size = 8
train_set = windowed_dataset(x_train, window_size=window_size, batch_size=100, shuffle_buffer=shuffle_buffer_size)
model = tf.keras.models.Sequential([
  tf.keras.layers.Conv1D(filters=8, kernel_size=5,
                         strides=1, padding="causal",
                         activation="relu",
                         input_shape=[None, 1]),
  tf.keras.layers.LSTM(60, return_sequences=True),
  tf.keras.layers.LSTM(60, return_sequences=True),
  tf.keras.layers.Dense(30, activation="relu"),
  tf.keras.layers.Dense(10, activation="relu"),
  tf.keras.layers.Dense(1),
  tf.keras.layers.Lambda(lambda x: x * 200)
])


optimizer = tf.keras.optimizers.SGD(lr=1e-5, momentum=0.9)
model.compile(loss=tf.keras.losses.Huber(),
              optimizer=optimizer,
              metrics=["mae"])
history = model.fit(train_set,epochs=150)

rnn_forecast = model_forecast(model, series[..., np.newaxis], window_size)
rnn_forecast = rnn_forecast[split_time - window_size:-1, -1, 0]

plt.figure(figsize=(16, 8))
plot_series(time_valid, x_valid)
plot_series(time_valid, rnn_forecast)

tf.keras.metrics.mean_absolute_error(x_valid, rnn_forecast).numpy()
```