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

На правах рукопису
УДК 519.711.2

До захисту допущено
В.о. завідувача кафедри ММСА
_____ О. Л. Тимощук
«____» _____ 2019 р.

Магістерська дисертація

на здобуття ступеня магістра за спеціальністю 124 Системний аналіз
на тему: «Система управління агентами фондового ринку із застосуванням глибинного навчання з підкріпленням»

Виконав:
студент II курсу, групи КА-82мп
Чаус Михайло Дмитрович _____

Керівник: доцент кафедри ММСА
к.ф.-м.н, доц. Яковлева А.П. _____

Рецензент: доцент кафедри математичного
аналізу та теорії ймовірностей
КПІ ім. Ігоря Сікорського
к.ф.-м.н., доц. Ільєнко А.Б. _____

Засвідчую, що в цій дипломній роботі
немає запозичень із праць інших авторів
без відповідних посилань.
Студент _____

Київ
2019

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 — 124 «Системний аналіз»

ЗАТВЕРДЖУЮ
В.о. завідувача кафедри ММСА
_____ О. Л. Тимощук
«____» _____ 2019 р.

ЗАВДАННЯ

на магістерську дисертацію студенту Чаусу Михайлу Дмитровичу

1. **Тема Дисертації**: «Система управління агентами фондового ринку із застосуванням глибинного навчання з підкріпленням», науковий керівник дисертації Яковлева Алла Петрівна, канд. фіз.-мат. наук, доцент, затверджені наказом по університету від «08» листопада 2019 р. № 3862-с.
2. **Термін подання студентом дисертації**: «13» грудня 2019 р.
3. **Об’єкт дослідження**: Агент фінансового ринку, шляхи вирішення проблеми оптимальної торгівлі.
4. **Предмет дослідження**: методи навчання з підкріпленням, які застосовуються при створенні автоматичного агента фінансового ринку.
5. **Перлік завдань, які подтрібно розробити**:
   1) дослідити сучасний стан та особливості застосування навчання з підкріпленням;
   2) розробити математичну модель агента фондового ринку;
   3) створити програмний продук на основі отриманої моделі;
   4) розробити стартап-проект виведення на ринок результатів дослідження;
   5) розробити концептуальні висновки за результатми наукового дослідження;
6. **Орієнтовний перелік графічного (ілюстративного) матеріалу**:
   1) графіки порівняння різних програмних продуктів;
   2) схеми марківських процесів прийняття рішень;

3) графікі ефективності створеного програмного продукту;
4) схеми розробленого програмного продукту;
5) таблиці у розділі стартап-проетку;

7. **Орієнтовний перелік публікацій**: Немає
8. **Дата видачі завдання**: _____

Календарний план

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи
1	Огляд літератури за тематикою роботи	01.09.2019-15.09.2019
2	Проведення аналізу існуючих підходів до застосування навчання з підкріпленням у задачах оптимального керування	01.09.2019-22.09.2019
3	Підготовка матеріалів першого розділу роботи	15.09.2019-29.09.2019
4	Розроблення математичного забезпечення для використання навчання з підкріпленням при створенні агента	01.10.2019-15.10.2019
5	Підготовка матеріалів другого розділу роботи	01.10.2019-15.10.2019
6	Підготовка матеріалів третього розділу роботи	15.10.2019-30.10.2019
7	Розроблення програмного забезпечення для вирішення задач оптимального керування методами навчання з підкріпленням	01.10.2019-15.11.2019
8	Перевірка розроблених моделей	15.11.2019-30.11.2019
9	Підготовка матеріалів четвертого розділу роботи	1.11.2019-30.11.2019
10	Оформлення пояснювальної записки	01.10.2019-30.11.2019

Студент _____ Чаус М. Д.
Науковий керівник дисертації _____ Яковлева А. П.

РЕФЕРАТ

Магістерська дисертація: 81 с., 6 ч., 22 табл., 16 рис., 1 дод., 14 джерел.

НАВЧАННЯ З ПІДКРІПЛЕННЯМ, ОПТИМАЛЬНЕ КЕРУВАННЯ, МАШИННЕ НАВЧАННЯ, АГЕНТ ФІНАНСОВОГО РИНКУ, АЛГОТРЕЙДІНГ, ГЛИБИННЕ НАВЧАННЯ, НЕЙРОННІ МЕРЕЖІ, DOCKER

Об’єкт дослідження - автономна система керування штучним агентом на фондовому ринку.

Метою даної магістерської дисертації є розробка програмного забезпечення для керування агентом фондового ринку та дослідження роботи отриманої системи в умовах реального середовища.

Методи дослідження - моделювання роботи фондового ринку з використанням потоку реальних даних з криптовалютної біржі BitMex.

Розроблена система базується на алгоритмах навчання з підкріпленням, які реалізовано за допомогою апроксимації нечіткими нейронними мережами. Глибинне навчання з підкріпленням має забезпечити довгострокову ефективність роботи агента впродовж довгого періоду часу.

Під час роботи було досліджено можливість штучного агента ефективно торгувати на фінансовому ринку. Була виконана його розробка за допомогою сучасних фреймоворків та засобів програмування. Було проведено тестування програмного продукту в реальних умовах.

На основі проведених досліджень розроблено нову технологічну модель для ведення торгів на біржі без попереднього налаштування стратегії агента.

ABSTRACT

Masters thesis: 81 p., 6 ch., 22 tbl., 16 fig., 1 addn., 14 references.

REINFORCEMENT LEARNING, OPTIMAL CONTROL, MACHINE LEARNING, FINANCIAL MARKET'S AGENT, ALGOTRADING, DEEP LEARNING, NEURAL NETWORKS, DOCKER

Object of the research - autonomous control system for managing stock market's agent.

The purpose of this thesis is a software development and research of acquired system's performance in real financial market.

Research methods - simulation of stock market operations using the flow of real data from the BitMex cryptocurrency exchange.

Developed system is based on the reinforcement learning algorythms, which were implemented by neo-fuzzy neural network approximation. Deep reinforcement learning has to maintain long run efficiency of the agent.

Potencial of the artificial agent was researched during the work. Its development was implemented using modern frameworks and programming utilities. The agent was tested under the constraints of real financial markets.

Based on the conducted research, a new technological model was developed for trading on the stock exchange without first adjusting the agent's strategy.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

НП — навчання з підкріпленням.

МППР — Марківський процес прийняття рішень.

CNFNN — cascade neo-fuzzy neural network.

ФР — фінансовий ринок.

ФВА — функціонально-вартісний аналіз.

ПП — програмний продукт.

ЕОМ – Електронна обчислювальна машина.

ВСТУП

На сьогоднішній день сукупний об’єм угод на фондовому, виконаних за рахунок алгоритмічної торгівлі складає більше 80%. Така висока популярність автоматичних агентів дала поштовх до розвитку ботів та торгових алгоритмів.

Проте існує проблема, пов’язана з тим, що агенти втрачають ефективність впродовж довгого періоду функціонування на біржі. Це явище відомо як α-розпад, за назвою коефіцієнта, що показує наскільки агент б’є ринок в середньому. Вирішенням цієї проблеми міг би стати агент, який автоматично пристосовується до ринку і постійно змінює свою стратегію.

У цій роботі буде наведено варіант реалізації такого агента. Базою для цього виступає навчання з підкріпленням. Зважаючи на високу популярність глибинного навчання у сучасному світі, навчання з підкріпленням буде реалізовано за допомогою нейронних мереж. Така система за задумом буде адекватно реагувати на зміни фінансового ринку та підтримуватиме свою ефективність.

1 ПОСТАНОВКА ЗАДАЧІ

Метою даної дипломної роботи є застосування методів навчання з підкріпленням до задачі керування агентом фондового ринку та дослідження його ефективності. Навчання з підкріпленням має бути реалізовано на основі глибинного навчання та з використанням нечітких нейронних мереж, а саме каскадних нео-фазі нейронних мереж.

Основним об’єктом дослідження виступає автоматичний торговий агент на фінансовому ринку. Він має бути реалізованим за допомогою сучасних фреймворків та засобів програмування. Після реалізації агента має бути проведено його тестування та дослідження.

Отриманий програмний продукт має влаштовувати вимоги щодо роботи з існуючими біржами та мати мінімально життєздатний функціонал.

2 НАВЧАННЯ З ПІДКРІПЛЕННЯМ

2.1 Вступ до розділу

Метою вирішення задач керування та прийняття рішень є знаходження такого розв’язку, який би був найкращим при наперед заданих критеріях. В минулому такі задачі описувалися аналітично і математики вирішували їх використовуючи класичні підходи методів оптимізації.

В результаті міг бути отриманий оптимальний розв’язок, який потім використовувався на практиці. Проте такий підхід не був ідеальним, оскільки в змінних умовах параметри теоретично описаної системи могли змінюватись і раніше отримане рішення вже не було оптимальним. Більше того раніше оптимальний розв’язок міг стати причиною аварії чи катастрофи.

З часом системи ускладнювались, їх аналітичні розв’язки вже не можна було так легко отримати. Іноді навіть важко правильно описати саму систему математично. В таких випадках виникає необхідність в створенні контроллерів, що могли б керувати автоматично, спираючись на значення деяких ключових метрик та систему керування без точного знання про внутрішню будову об’єкта керування.

Підходом, що намагається вирішити поставлену задачу є навчання з підкріпленням (англ. Reinforcement learning) [1]. Методи навчання з підкріпленням відносяться до ширшого класу задач - машинного навчання. Основною їхньою особливістю є можливість знаходження оптимального керування без знання точної динаміки керованої системи. Методи навчання з підкріпленням працюють за принципами, що були запозичені з природних систем, біології та соціології. Їх навіть можна назвати результатом спроби перенести експерименти Івана Павлова над собаками у цифровий вимір.

2.2 Елементи навчання з підкріпленням

Основною задачею навчання з підкріпленням є знаходження оптимальних дій агента у різних станах динамічної системи. Агентом можна назвати будь-яку штучну сутність, що взаємодіє усередині динамічної системи. Важливо, щоб агент не мав точної моделі системи до процесу навчання і обирав дії спираючись на свій досвід взаємодії з нею. Часто в області штучного інтелекту динамічну систему в якій діє агент називають середовищем.

Крім середовища та агента можна виділили ще чотири сутності: стратегія, нагорода, функція вартості та модель середовища.

Стратегія визначає поведінку агента всередині середовища в певний момент часу. Грубо кажучи стратегія це відображення із простору станів середовища у простір дій агента. Стратегія може бути визначена у формі функції або стратегії або навіть процесу пошуку у більш складних випадках. Стратегії достатньо для визначення дій агента, тому вона повністю його характеризує. Зазвичай стратегія є стохастичною в тому сенсі, що вона вказує на ймовірність вибору дії агента.

Сигнал нагороди визначає ціль навчання з підкріпленням. В кожну одиницю часу середовище надсилає агенту нагороду - числовий відклик. Завданням агента є максимізація суми нагород в перспективі. Зазвичай нагорода вказує на те, наскільки гарно агент діє в різних ситуаціях. Виходячи зі значення нагороди критик може змінювати стратегію актора. У середовищі нагорода агента відіграє роль досвіду болю чи насолоди у природних системах, через які закріплюється поведінка тварин.

Функція вартості, на відміну від нагороди, визначає яка дія є гарною в довгостроковій перспективі. Грубо кажучи вартість це сума нагород, які агент може отримати, почавши з поточного стану. Вартість стану не завжди корелює з нагородою, яку отримує агент, знаходячись в цьому стані. З точки зору біології вартість можна інтерпретувати як міру того, наскільки сприятливий

прогноз щодо організму в даний момент.

Вартість є центральною компонентою у навчанні з підкріпленням. Завдяки їй робляться рішення і виводяться стратегії. По факту, ключовим алгоритмом, що рухає навчання є оцінка вартості станів середовища. На основі цієї оцінки і виводиться стратегія агента.

Четвертий елемент деяких систем навчання з підкріпленням - це модель середовища. Вона зазвичай імітує динамічну систему для того, щоб оцінити поведінку системи у майбутньому. У цій роботі методи, що спираються на повну модель системи не використовуються, хоча вони є дуже цікавими з точки зору планування.

2.3 Основоположна ідея навчання з підкріпленням

Оптимальність агента визначає значення числового сигналу нагороди, яку отримує агент після кожної дії. Таким чином, для досягнення оптимального результату агент має використовувати найкращу знайдену стратегію з одного боку і досліджувати можливі покращення з іншого. Кінцева нагорода та пошук оптимальної стратегії є ключовими характеристиками, що виділяють навчання з підкріпленням від інших класів алгоритмів.

Для досягнення бажаного результату було запропоновано схему актора-критика [2], яку показано на рисунку 2.1. Ця структура показує етапи роботи алгоритмів навчання з підкріпленням, які в широкому сенсі можна назвати навчанням агента.

За цією схемою актор, знаходячись в певному стані системи виконує операцію відповідно до закладеної в нього стратегії. Після чого динамічна система переходить в наступний стан і дає певний відклик на дії агента - нагороду. Це значення оцінює критик, який встановлює ефективність дій агента і за необхідності змінює стратегію.

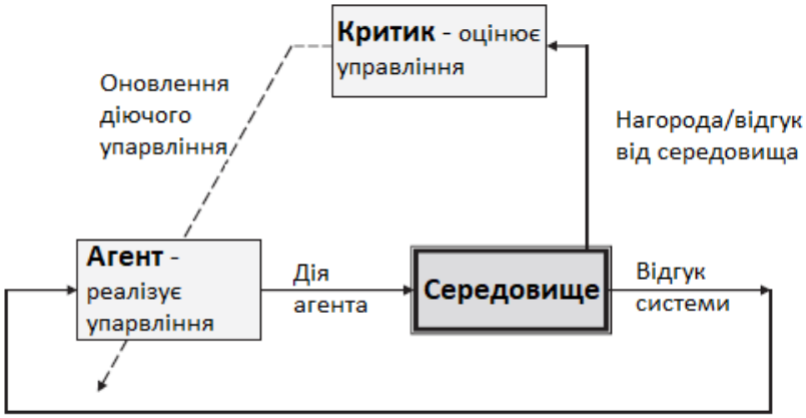

Рисунок 2.1 – Схема актор-критик, яка описує концепцію алгоритмів навчання з підкріпленням.

Агент не знає моделі динамічної системи, тому її стан він оцінює виходячи з доступної для нього інформації - це можуть бути показники датчиків, спрощена модель системи чи навіть ціле зображення навколишнього світу.

Зміна критиком стратегії після дій агента проводиться у два кроки. На першому критик переоцінює поточну стратегію, а на другому виконується її покращення. Після чого агент виконує наступну дію.

Алгоритми навчання з підкріпленням математично описуються за допомогою марківських процесів прийняття рішень. Цей математичний апарат дає змогу моделювати проблеми, а яких результат частково залежить від агента, що приймає рішення, а частково випадковий. Проблеми з таких областей як економіка, робототехніка та автоматизоване виробництво можна формулювати завдяки марківским процесам прийняття рішень.

2.4 Марківскі процеси прийняття рішень, як основа для навчання з підкріпленням

Марківский процес прийняття рішень - класичний математичний апарат для описання послідовного прийняття рішень, де дії впливають не тільки на значення нагороди на наступному кроці, а й викликають певний стохастичний процес, що змінює стан системи. У цьому розділі буде дано означення Марківським процесам прийняття рішень та наведено основні рівняння, на яких базуються методи навчання з підкріпленням.

З формальної точки зору Марківский процес прийняття рішень - це четвірка (S, A, P, R), де S - множина станів, а U - множина дій агента. Перехідні ймовірності $P : X \times A \times X \to [0,1]$ визначають для кожного стану $s \in S$ та дії $a \in A$ умовну ймовірність $P^u_{s,s'} = p(s'|s, a)$ того, що дія a виконана із стану x приведе агента у стан s'. Вартість цього переходу визначається функцією $R : S \times A \times S \to \mathbb{R}$. При цьому має виконуватись марківска властивість, що полягає в тому, що перехідні ймовірності $p(s'|s, a)$ залежать лише від поточного стану s і не залежать від того, як агент досяг цього стану. Приклад марківського процесу прийняття рішень зображено на рисунку 2.2 у вигляді графу на якому вершини це стани, а ребра - наслідки дій агента.

Агент взаємодіє з середовищем у кожен дискретний момент часу $t = 0, 1, 2, 3, ...$ В кожен момент часу агент отримує деяку репрезинтацію стану середовища $S_t \in S$, на основі якого вибирає дію $a_t \in A(s)$. В наступний момент часу він отримує нагороду $R_{t+1} \in \mathbb{R} \subset \mathbb{R}$ і переходить в новий стан S_{t+1}. Таким чином будується послідовність або траєкторія, що виглядає наступним чином:

$$S_0, A_0, R_1, S_1, A_1, R_2, S_2, A_2, R_3.... \tag{2.1}$$

Основна проблема, яку треба вирішити для марківського процесу прийняття рішень - знаходження відображення $\pi : S \times A \to [0,1]$, що для

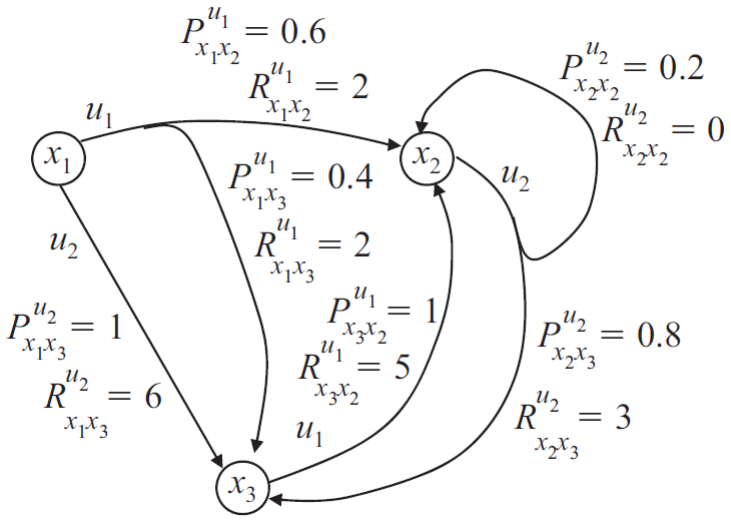

Рисунок 2.2 – Марківський процес прийняття рішень.

кожного стану s та дії a надає умовну ймовірність $\pi(s,a)$ вибору дії a у стані s. Якщо для кожного стану передбачена лише одна дія, то така стратегія називається дитерміністичною.

Стратегія вважається оптимальною, якщо її середній сумарний відклик є найбільшим. В найпростішому випадку сумарний відклик це просто сума нагород агента:

$$G_t = R_{t+1} + R_{t+2} + R_{t+3} + R_{t+4} + R_{t+5} + ... + R_T, \tag{2.2}$$

де T - кінцевий момент часу. Стан середовища в момент T називається термінільним станом. Якщо $T = \infty$, то такий відгук не є дуже зручним через необмеженість. У такому випадку доцільно використовувати дисконтування:

$$G_t = R_{t+1} + \gamma^1 R_{t+2} + \gamma^2 R_{t+3} + ... = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}, \tag{2.3}$$

де γ - параметр, $0 \leq \gamma \leq 1$, що називається норма дисконтуванням.

Норма дисконтування визначає цінність майбутніх нагород зараз: нагорода, отримана через k моментів часу вартує в γ^{k-1} разів менше, ніж якби вона була отримана зараз. З практичної точки зору дисконтування вказує на той факт, що капітал зараз є ціннішим ніж через деякий час.

Функцію вартості в стані s для стратегією π, позначимо $v_\pi(s)$. Для марківського процесу прийняття рішень формально вона визначається наступним чином:

$$v_\pi(s) = \mathbb{E}_\pi[G_t|S_t = s] = E_\pi[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1}|S_t = s], \text{ для всіх } s \in S. \tag{2.4}$$

При цьому значення термінального стану завжди рівне нулю. Функція v_π називається функцією вартості стану для стратегії π.

Так само визначимо вартість виконання дії a у стані s для стратегії π, позначимо $q_\pi(s, a)$, як середній сумарний відклик:

$$q_\pi(s, a) = \mathbb{E}_\pi[G_t|S_t = s, A_t = a] = E_\pi[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1}|S_t = s, A_t = a]. \tag{2.5}$$

Функція q_π називається функцією вартості дій для стратегії π. Часто в літературі можна зустріти просту назву - Q-функція від англійського слова Quality.

Фундаментальною властивістю функцій вартості є можливість вираження їх у вигляді рекурентних формул. Для будь-якої стратегії π та стану s виконується наступне співвідношення між вартістю стану s та вартістю його можливих наступників:

$$\begin{aligned} v_\pi(s) &= \mathbb{E}_\pi[G_t|S_t = s] = E_\pi[R_{t+1} + \gamma G_{t+1}|S_t = s] = \\ &= \sum_a \pi(a|s) \sum_{s'} \sum_r p(s', r|s,a)[r + \gamma \mathbb{E}_\pi[G_{t+1}|S_{t+1} = s']] = \\ &= \sum_a \pi(a|s) \sum_{s',r} p(s', r|s,a)[r + \gamma v_\pi(s')], \text{ для всіх } s \in S. \end{aligned} \tag{2.6}$$

Рівняння 2.6 називається рівнянням Белмана [3] для v_π. Аналогічну формулу можна записати для Q-функції:

$$
\begin{aligned}
q_\pi(s,a) = \mathbb{E}_\pi[G_t|S_t = s, A_t = a] = \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1}|S_t = s, A_t = a] = \\
= \sum_{r,s'} p(s', r|s, a)(r + \gamma \mathbb{E}_\pi[G_{t+1}|S_{t+1} = s']) = \\
= \sum_{r,s'} p(s', r|s,a)(r + \gamma \sum_{a'} \pi(a'|s')q(s',a')),
\end{aligned}
\tag{2.7}
$$

Ці два рівняння є фундаментальними для побудови на їх основі методів для визначення, апроксимації та вивчення функцій вартості.

2.5 Оптимальні стратегії та оптимальні функції вартості

При розв'язанні задачі навчання з підкріпленням головною метою є знаходження стратегії, що заробляє велику нагороду в перспективі. Для кінцевих марківських процесів прийняття рушень, можна точно визначити оптимальну стратегію, оскільки функція вартості дає змогу встановити відношення порядку між стратегіями.

Так стратегія π вважається кращою за стратегію π', тоді і тільки тоді, коли $v_\pi(s) \geq v_{\pi'}(s)$ для всіх $s \in S$. Серед усіх стратегії існує завжди така, яка не гірша за всі інші, це і є оптимальна стратегія. У оптимальна стратегія $\pi_\star$ може бути не єдиною. Проте всі такі стратегії матимуть одну й ту ж функцію вартості $v_\star$, що визначається наступним чином:

$$
v_\star(s) = \max_\pi v_\pi(s) \text{ для всіх } s \in S. \tag{2.8}
$$

У всіх оптимальних стратегій також одна й та ж сама оптимальна функція вартості станів - Q-функція:

$$
q_\star(s, a) = \max_\pi q_\pi(s, a) \text{ для всіх } s \in S \text{ та } a \in A(s). \tag{2.9}
$$

Для пар (s,a) ця функція дає математичне сподівання нагороди за виконання дії a у стані s та нагороду за слідування оптимальній стратегії. Таким чином можна записати $q_\star$ через $v_\star$ наступним чином:

$$q_\star(s,a) = \mathbb{E}[R_{t+1} + \gamma v_\star(S_{t+1}) | S_t = s, A_t = a]. \tag{2.10}$$

Оскільки $v_\star$ є функція вартості стану деякої оптимальної стратегії, то для неї можна записати рівняння Белмана для вартості станів 2.6. Оскільки $v_\star$ оптимальна функція вартості, то її можна виразити без ймовірностей оптимальної стратегії. Таку форму запису прийнято називати рівнянням оптимальності Белмана. Грубо кажучи рівняння оптимальності Белмана виражає той факт, що вартість довільного стану оптимальної стратегії рівна очікуваній вартості найкращої дії з цього стану. Цей принцип виражає наступне рівняння:

$$\begin{aligned} v_\star(s) &= \max_{a \in A(s)} q_{\pi_\star}(s,a) \\ &= \max_a \mathbb{E}_{\pi_\star}[G_t | S_t = s, A_t = a] \\ &= \max_a \mathbb{E}_{\pi_\star}[R_{t+1} + \gamma G_{t+1} | S_t = s, A_t = a] \\ &= \max_a \mathbb{E}[R_{t+1} + \gamma v_\star(S_{t+1}) | S_t = s] \\ &= \max_a \sum_{s',r} p(s',r|s,a)[r + \gamma v_\star(s')]. \end{aligned} \tag{2.11}$$

Останні два рівняння визначають дві форми запису рівняння оптимальності Белмана для $v_\star$. Рівняння оптимальності Белмана для $q_\star$ має наступний вигляд:

$$\begin{aligned} q_\star(s,a) =& \mathbb{E}[R_{t+1} + \gamma \max_{a'} q_\star(S_{t+1}, a') | S_t = s, A_t = a] \\ =& \sum_{s',r} p(s',r|s,a)[r + \gamma \max_{a'} q_\star(s',a')] \end{aligned} \tag{2.12}$$

Для скінчених марківських процесів прийняття рішень рівняння оптимальності Белмана має єдиний розв'язок для $v_\star$. Для кожного стану s існує

одне рівняння оптимальності, тому насправді рівняння оптимальності Белмана це система рівнянь з n невідомими при n станах. Цю систему рівнянь можна розв'язати чисельними методами для розв'язку нелінійних рівнянь.

Отримавши значення $v_\star$ можна з легкістю відновити оптимальну стратегію, діючи жадібно. А маючи $q_\star$ для цього навіть не треба знати динаміку середовища.

На практиці визначення точної функції вартості досить рідко трапляється, оскільки таке рішення потребує величезних затрат по пам'яті та часу через необмежений простір станів. Тому такі функції вартості часто оцінюються різними методами апроксимації.

2.6 Динамічне програмування

Основою для всіх алгоритмів навчання з підкріпленням є динамічне програмування, що об'єднує декілька алгоритмів для знаходження оптимальних стратегій при заданій повній моделі марківського процесу прийняття рішень. По факту методи навчання з підкріпленням досягають тих же цілей тільки з меншою кількістю розрахунків та без точної моделі середовища.

2.6.1 Оцінка стратегій

Першим важливим алгоритмом є алгоритм оцінення довільної дитерміністичної стратегії π. Як і більшість алгоритмів динамічного програмування цей алгоритм можна отримати шляхом перетворення рівняння Белмана в правило для оновлення функції вартості станів при заданій стратегії.

Із рівняння 2.6 маємо:

$$v_\pi(s) = \sum_a \pi(a|s) \sum_{s',r} p(s', r|s,a)[r + \gamma v_\pi(s')], \text{ для всіх } s \in S, \tag{2.13}$$

де $\pi(a|s)$ - ймовірність виконання дії a в стані s при заданій стратегії π. Існування та єдиність функції v_π гарантується, якщо $\gamma < 1$ або відомо, що агент досягає термінального стану з будь-якого іншого за скінчений проміжок часу при заданій стратегії π.

Якщо динаміка системи точно відома, то рівняння 2.13 є системою з $|S|$ лінійних рівнянь з $|S|$ невідомими. Найбільш відповідним методом для вирішення цієї системи з точки зору навчання з підкріпленням є метод ітерації.

Розглянемо послідовність наближень функції вартості $v_0, v_1, v_2, v_3 \ldots$. Якщо початкове наближення v_0 обрано довільно (крім випадку термінального стану, коли значення функції вартості має бути 0), то кожне наступне наближення буде отримано з рівняння Белмана для v_π:

$$\begin{aligned} v_{k+1}(s) =& \mathbb{E}[R_{t+1} + \gamma v_k(S_{t+1})|S_t = s] \\ =& \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a)[r + \gamma v_k(s')] \text{ для всіх } s \in S. \end{aligned} \tag{2.14}$$

Очевидно, що $v_k = v_\pi$ - фіксована точка для цього ітеративного методу, тому що в цьому випадку ми отримуємо рівняння Белмана 2.6. А послідовність v_k сходиться до v_π, при $k \to \infty$, при тих же умовах, що забезпечують існування v_π. Цей алгоритм називається ітеративна оцінка стратегії.

2.6.2 Алгоритм покращення стратегій

Причина по якій ми обчислюємо значення функції вартості - знаходження кращих стратегій. Припустимо, що відоме значення функції вартості v_π для довільної дитермінованої стратегії π. Для довільного стану s цікаво знати, змінювати наші дії чи продовжувати діяти відносно стратегії $\pi(s)$. Відповідь на це питання можна отримати, розглянувши дію a в стані s, а потім продовжити слідувати стратегії π. В такому випадку функція вартості матиме наступний вигляд:

$$\begin{aligned} q_\pi(s,a) =& \mathbb{E}[R_{t+1} + \gamma v_\pi(S_{t+1}) | S_t = s, A_t = a] \\ =& \sum_{s',r} p(s',r|s,a)[r + \gamma v_\pi(s')]. \end{aligned} \tag{2.15}$$

Якщо значення $q_\pi(s,a)$ більше за $v_\pi(s)$, то по факту нова стратегія, що полягає в обранні дії a в стані s є кращою за π. Тобто у формі двох стратегій має виконуватись наступна нерівність:

$$v'_\pi(s) \geq v_\pi(s). \tag{2.16}$$

Цей результат випливає з теореми про покращення стратегії, яка стверджує, що виконується, якщо:

$$q_\pi(s, \pi'(s)) \geq v_\pi(s). \tag{2.17}$$

Дійсно, розписуючи ми отримуємо наступний ланцюжок рівностей:

$$
\begin{aligned}
v_\pi(s) \leq & q_\pi(s, \pi'(s)) \\
= & \mathbb{E}[R_{t+1} + \gamma v_\pi(S_{t+1}) | S_t = s, A_t = \pi'(s)] \\
= & \mathbb{E}_{\pi'}[R_{t+1} + \gamma v_\pi(S_{t+1}) | S_t = s] \\
\leq & \mathbb{E}_{\pi'}[R_{t+1} + \gamma q_\pi(S_{t+1}, \pi'(S_{t+1})) | S_t = s] \\
= & \mathbb{E}_{\pi'}[R_{t+1} + \gamma \mathbb{E}[R_{t+2} + \gamma v_\pi(S_{t+2}) | S_{t+1}, A_{t+1} = \pi'(S_{t+1})] | S_t = s] \\
& ... \\
\leq & \mathbb{E}_{\pi'}[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 R_{t+4} + ... | S_t = s] \\
= & v_{\pi'}(s).
\end{aligned}
\tag{2.18}
$$

Ці нерівності доводять, що вибираючи кращу дію в довільному стані можна покращити поточну стратегію. Логічно вибирати найкращу дію з точки зору поточної стратегії, тобто для кожного стану запропонувати альтернативну стратегію:

$$
\begin{aligned}
\pi'(s) &= \arg\max_a q_\pi(s,a) \\
&= \arg\max_a \mathbb{E}[R_{t+1} + \gamma v_\pi(S_{t+1}) | S_t = s, A_t = a] \\
&= \arg\max_a \sum_{s',r} p(s',r|s,a)[r + \gamma v_\pi(s')].
\end{aligned}
\tag{2.19}
$$

В такому випадку π' є жадібною і кращою за π. Для знаходження стратегії π'' необхідно оцінити $v_{\pi'}$ і знов обрати дії жадібно відповідно до $v_{\pi'}$. Так треба робити до сходження, тобто коли вартість стратегії не зміниться, в цьому випадку ми отримаємо наступну рівність:

$$
\begin{aligned}
v_{\pi'}(s) &= \max_a \mathbb{E}[R_{t+1} + \gamma v_{\pi'}(S_{t+1}) | S_t = s, A_t = a] \\
&= \max_a \sum_{s',r} p(s',r|s,a)[r + \gamma v_{\pi'}(s')],
\end{aligned}
\tag{2.20}
$$

яка є рівняння оптимальності Белмана 2.6, з чого випливає, що π' - оптимальна стратегія.

Таким чином було отримано алгоритм покращення стратегій, який в літературі називається перебором стратегій [4]. Оскільки скінчений марківський процес прийняття рішень має кінцеву кількість дитерміністичних стратегій, то процес перебору стратегій буде скінченим:

$$\pi_0 \xrightarrow{\text{О}} v_{\pi_0} \xrightarrow{\text{П}} \pi_1 \xrightarrow{\text{О}} v_{\pi_1} \xrightarrow{\text{П}} \pi_2 \xrightarrow{\text{О}} v_{\pi_2} \xrightarrow{\text{П}} \pi_2 \xrightarrow{\text{О}} \ldots \xrightarrow{\text{П}} \pi_\star \xrightarrow{\text{О}} v_\star, \tag{2.21}$$

де О означає оцінку, а П - покращення.

Одним з недоліків такого алгоритму є те, що на кожній ітерації ми маємо оцінювати поточну стратегію, що є досить затратним з точки зору обчислень. На практиці не обов'язково знаходити вартість стратегії, можна оцінити її з певною точністю, зробивши менше ітерацій. Один важливий випадок, коли робиться лише одна ітерація, називається ітерація функції вартості [5]. Цей алгоритм отримано шляхом перетворення 2.6 в правило оновлення:

$$\begin{aligned} v_{k+1}(s) &= \max_a \mathbb{E}[R_{t+1} + \gamma v_k(S_{t+1}) | S_t = s, A_t = a] \\ &= \max_a \sum_{s',r} p(s',r|s,a)[r + \gamma v_k(s')] \text{ для всіх } s \in S. \end{aligned} \tag{2.22}$$

Схематично сходження вище описаних методів до оптимальної стратегії можна описати геометрично, як показано на рисунку 2.3. На кожному кроці ми робимо два види обчислень: оцінюємо поточну стратегію та оновлюємо її відповідно до нашої оцінки.

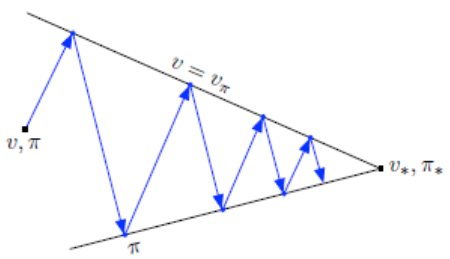

Рисунок 2.3 – Схема роботи алгоритмів покращення стратегій.

2.7 Вивчення стратегій

Алгоритми, що розглядались в минулому розділі є підгрунтям для безпосередньо алгоритмів навчання з підкріпленням, проте вони мають той недолік, що вимагають знання про структуру досліджуваного середовища. У цьому розділі будуть розглядатись алгоритми, побудовані на основі методів динамічного програмування. Вони будуть використовувати досвід агента для оцінки функцій вартості та знаходження оптимального керування.

2.7.1 Методи Монте Карло

Методи, що розглядаються у цьому підрозділі називаються методами Монте Карло, оскільки використовують усереднення багаторазового проходження агента через середовище. Кожен раз, пройшовши певну траєкторію, агент оновлює своє представлення про середовище через зміну оцінки функцій вартості.

Так, наприклад функція вартості станів для деякої стратегії π вираховується для кожного стану, як середній відклик, отриманий агентом після цього стану. Пройшовши середовище достатньо велику кількість разів, агент може досить точно оцінити функція вартості станів. При цьому є два типи методів. Один при якому відклик розглядається лише після першого потрапляння в стан. Другий, в якому відклик зберігається після всіх потраплянь. Алгоритм для першого методу описано нижче. Алгоритм для другого методу майже не буде відрізнятись, за виключенням перевірки на потрапляння в попередні кроки.

Вхід: визначити стратегію π, що буде оцінюватись

Ініціалізація: $V(s) \in \mathbb{R}$ для всіх $s \in S$

$R(s) -$ пустий список для всіх $s \in S$

$N \in \mathbb{N} -$ к-сть проходів середовища

Цикл N разів :

Сгенерувати траєкторію $\pi : S_0, A_0, R_1, S_1, A_1, R_2, ...,S_{T-1}, A_{T-1}, R_T$

$G \leftarrow 0$

Цикл для кожного кроку агента, $t = T-1, T-2, ...0$:

$G \leftarrow \gamma G + R_{t+1}$

Якщо S_t не в $S_0, S_1, ..., S_{t-1}$:

Додати G до$R(S_t)$

$V(s) \leftarrow$ середнє$(R(s))$ для всіх $s \in S$.

Обидва алгоритми сходяться до значення $v_\pi(s)$ зі збільшенням кількості потрапляння до стану s. Вищеописаний алгоритм, в якому значення нагороди після кожного потрапляння - незалежна випадкова величина, за законом великих чисел прямує до мат. сподівання $v_\pi(s)$. З кожним новим потраплянням стандартне відхилення середнього від $v_\pi(s)$ прямує до нуля.

Оцінка вартості дій працює за тим самим принципом. Єдина відмінність полягає в тому, що оцінюються не стани, а пари стан, дія (s, a). В алгоритмах оцінки вартості дій є проблема, пов'язана з тим, що агент може не відвідати всі пари (s, a) через його стратегію. Це може бути великою проблемою при вирішенні питання оптимального керування, оскільки заважає вибору серед невідомих пар.

Є два варіанти вирішення цієї проблеми. Перший полягає в тому, щоб починати траєкторію з довільної пари (s, a). Таким чином за досить великий проміжок часу будуть оцінені всі пари. Другий полягає в тому, щоб слідувати не

оптимальній стратегії, а субоптимальній, при якій оптимальні дії виконуються більшість часу, а неоптимальні обираються з малою ймовірністю.

На основі оцінення вартості визначається алгоритм оновлення стратегій, який забезпечує досягнення оптимального керування. Суть його полягає в тому, що після проходження кожної чергової траєкторії стратегія оновлюється жадібно зі збереженням можливості потрапити в будь-який стан:

Ініціалізація: $\epsilon > 0$

$Q(s, a) \in \mathbb{R} -$ для всіх $s \in S, a \in A(s)$

$R(s, a) -$ пустий список для всіх $s \in S, a \in A(s)$

$N \in \mathbb{N} -$ к-сть проходів середовища

Цикл N разів :

Сгенерувати траєкторію $\pi : S_0, A_0, R_1, S_1, A_1, R_2, ...,S_{T-1}, A_{T-1}, R_T$

$G \leftarrow 0$

Цикл для кожного кроку агента, $t = T - 1, T - 2, ...0$:

$G \leftarrow \gamma G + R_{t+1}$

Якщо (S_t, A_t) не в $S_0, A_0, S_1, A_1, ..., S_{t-1}, A_{t-1}$:

Додати G до$R(S_t, A_t)$

$Q(S_t, A_t) \leftarrow$ середнє$(Returns(S_t, A_t))$

$A^\star \leftarrow \arg\max_a Q(S_t, a)$

Оновити стратегію $\pi_\epsilon(a|S_t)$.

Використовуючи цей алгоритм вже можна знаходити оптимальні стратегії для багатьох прикладних задач. Даний алгоритм вимагає збереження значень вартості всіх пар (s, a), що може бути також досить затратним з точки зору ресурсів ЕОМ. Також для оновлення інформації про середовище необхідно повністю пройти всю траєкторію. З практичної точки зору це не дуже

ефективно, оскільки довжина окремих траєкторій може бути досить великою.

2.7.2 Різницеві методи

Методи різницевого навчання (РН) поєднують ідеї методів Монте Карло та динамічного програмування. Так, оцінка вартості стратегій робиться на основі досвіду агентів але без очікування термінального стану. Таким чином нові методи біль універсальні та гнучкі при застосуванні.

Обидва методи РН та Монте Карло використовують досвід агента для вирішення проблеми оцінки вартості стратегії. Отримавши нову порцію інформації після слідування стратегії π, обидва методи оновлюють свій прогноз щодо функції вартості v_π для всіх нетермінальних станів S_t, які вони відвідали. Але метод Монте карло очікує, поки агент досягне термінального стану:

$$V(S_t) \to V(S_t) + \alpha[G_t - V(S_t)], \tag{2.23}$$

де G - повний відклик середовища, отриманий агентом після кроку t. $alpha$ - параметр швидкості навчання.

На відміну від методів Монте Карло, методи РН не мають чекати зупинки агента, а можуть оновити оцінку функції вартості вже на наступному кроці:

$$V(S_t) \to V(S_t) + \alpha[R_{t+1} + \gamma V(S_{t+1}) - V(s_t)]. \tag{2.24}$$

Це правило оновлює значення функції вартості v_π, використовуючи як еталон значення очікування $R_{t+1} + \gamma V(S_{t+1})$ замість G_t. Методи РН як правило сходяться швидше за метод Монте Карло, проте вони схильні до упередженого відхилення, через те, що початкове значення вартості задається наперед і відрізняється від справжнього.

Методи РН сходяться до справжнього значення функції вартості, якщо

наступні умови накладені на параметр α:

$$\sum_{n=1}^{\infty} \alpha_n(a) = \infty \quad \text{та} \quad \sum_{n=1}^{\infty} \alpha_n^2(a) < \infty \tag{2.25}$$

Методи РН можна також використовувати для визначення оптимального керування. Як і у випадку методів Монте Карло необхідно робити компроміс між використанням та дослідженням стратегій. Тому визначення стратегій робиться у два кроки.

На першому кроці необхідно оцінити функцію вартості дій $q_\pi(s,a)$ для поточної стратегії π. Це робиться так само, як і з $v_\pi(s)$:

$$Q(S_t, A_t) \to Q(S_t, A_t) + \alpha[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)]. \tag{2.26}$$

Ця процедура виконується після кожного нетермінального стану S_t, якщо S_{t+1} - термінальний, то його значення $Q(S_{t+1}, A_{t+1})$ прирівнюється до 0. Це правило оцінки стратегії використовує набір з 5 елементів $(S_t, A_t, R_{t+1}, S_{t+1}, A_{t+1})$, через що в іноземній літературі цей метод називається SARSA.

На другому кроці кожен раз ми обираємо стратегію відповідно до значень Q. Ця стратегія має бути $\epsilon-$ гладкою, для того, щоб забезпечити сходження до оптимальної.

Одним з найуспішніших алгоритмів РН став алгоритм Q-learning [6], який визначається наступним правилом:

$$Q(S_t, A_t) \to Q(S_t, A_t) + \alpha[R_{t+1} + \gamma \max_a Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)]. \tag{2.27}$$

У цьому випадку $Q(S_t, A_t)$ відразу апроксимує $q_\star$ - функцію вартості оптимальної стратегії. При виконанні умови 2.25 та припущенні про проходження всіх пар (s, a), значення Q збігається до $q_\star$. Наступна процедура

реалізує алгоритм Q-learning:

Ініціалізація: $\epsilon > 0, \alpha \in (0,1]$

$N \in \mathbb{N}$ – к-сть проходів середовища

Цикл N разів :

Визначити S_0

Поки $S_t \neq$ термінальний стан:

Обрати A з $S,$ використовуючи ϵ – гладку стратегію при Q

Виконати дію $A,$ та визначити R, S' :

$Q(S,A) \leftarrow Q(S,A) + \alpha[R + \gamma \max_a Q(S',a) - Q(S,A)]$

$S \leftarrow S'.$

2.7.3 Методи апроксимації

Всі методи, що розглядались до цього в цьому розділі мають один значний недолік, що полягає в використані таблиць для збереження оцінених значень функцій вартості. З практичної точки зору це дуже неефективне використання пам'яті.

Для вирішення цієї проблеми можна використовувати різні методи апроксимації функцій вартості. Апроксимуюча функція має набір вагів $w \in \mathbb{R}^d$, що дозволяють зробити наближення $\hat{v}(s,w) \approx v_\pi(s)$. Оскільки розмірність w набагато менша ніж S, необхідно значно менше пам'яті для оцінки функції вартості.

В області машинного навчання розроблено багато алгоритмів, що виконують апроксимацію деякої функції при заданому певному наборі даних.

Такі алгоритми об'єднують у групу, що називається навчання з вчителем. Застосовуючи цей підхід до задачі оцінки функції вартості, можна було б записати задачу оптимізації:

$$J(w) = E_\pi[(v_\pi(s) - \hat{v}(s,w))^2] \to min. \tag{2.28}$$

Таку задачу можна спробувати вирішити за допомогою метода градієнтного спуску, в якому оновлення ваг відбувається до сходження на наступну величину:

$$\begin{aligned} \Delta w = &- \frac{1}{2}\nabla_w J(w) = \\ &= \alpha \mathbb{E}_\pi[(v_\pi(s) - \hat{v}(s,w))\nabla_w \hat{v}(s,w)]. \end{aligned} \tag{2.29}$$

Але на практиці такий підхід звичайно неможливий, через відсутність $v_\pi(s)$. Тому виникає необхідність у заміні функції вартості $v_\pi(s)$ іншою величиною.

Для методу Монте Карло правило оновлення ваг буде мати наступний вигляд:

$$\Delta w = \alpha(G_t - \hat{v}(S_t,w))\nabla_w \hat{v}(S_t, w). \tag{2.30}$$

Це оновлення вагів відбувається для кожного кроку після проходження всієї траєкторії. Аналогічне правило оновлення вагів можна записати для методів РН. При цьому оновлення вагів буде відбуватись на кожному кроці агента:

$$\Delta w = \alpha(R_{t+1} + \gamma\hat{v}(S_{t+1}, w) - \hat{v}(S_t,w))\nabla_w \hat{v}(S_t, w). \tag{2.31}$$

Таким чином можна вважати, що ми застосували методи навчання з вчителем для множини пар даних. У випадку методу Монте Карло це будуть пари: $((S_1, G_1), (S_2, G_2), ...(S_T, G_T))$. Для РН це будуть пари: $((S_1, R_2 + \gamma\hat{v}(S_2, w), (S_2, R_3 + \gamma\hat{v}(S_3, w), (S_3, R_4 + \gamma\hat{v}(S_4, w), ..., (S_{T-1}, R_T))$

За ти самим принципом можна оцінювати функцію вартості дій. Нижче

наведено приклад для методу Монте Карло:

$$\Delta w = \alpha(G_t - \hat{q}(S_t,A_t, w))\nabla_w \hat{q}(S_t, A_t, w). \tag{2.32}$$

Для Q-learning за тим же принципом правило оновлення вагів наступне:

$$\Delta w = \alpha(R_{t+1} + \gamma \max_a \hat{q}(S_{t+1}, a, w) - \hat{v}(S_t, A_t, w))\nabla_w \hat{q}(S_t, A_t, w) \tag{2.33}$$

У якості апроксиматора також може виступати нейронна мережа [7], яка є центральним об'єктом при створенні агента у даній роботі. Нейронна мережа буде виступати апроксиматором для алгоритму Q-learning. Ефективність цього підходу була доведена на практиці [8].

Для реалізації ефективного навчання агента використаємо декілька прийомів. По-перше, будемо зберігати досвід агента у пам'яті шляхом запам'ятовування переходів $(s_t, a_t, r_{t+1}, s_{t+1})$. По-друге, при тренуванні будемо використовувати дві нейронні мережі. Одну будемо використовувати як предиктор функції вартості, а іншу будемо безпосередньо навчати.

Це необхідно для стабілізації нейронної мережі, оскільки без цих дій її ваги починають розходитись. Основними стабілізаторами виступають дві сутності. По-перше, випадковий вибір даних з вибірки при тренування зменшує кореляцію між діями і сприяє кращому узагальненню. По-друге, дві мережі не дають коефіцієнтам вибухнути за короткий проміжок часу і стримують їх зміну. Нейронна мережа предиктор оновлює свої параметри періодично впродовж навчання.

Ваги другої мережі оновлюються так, щоб зменшити середньоквадратичну похибку:

$$L_t(w_t) = E_{D_t \sim (s,a,r,s')}[(r + \gamma \max_a Q_b(s',a, w') - Q_t(s, a, w))^2], \tag{2.34}$$

де D_t - набір даних для навчання, (s,a,r,s') - перехід, Q_b - мережа предиктор, Q_t - мережа, яку навчаємо.

Повний алгоритм глибинного навчання з підкріпленням на основі Q-learning виглядає наступним чином:

Ініціалізація: $Q_t, Q_b -$ нейронні мережі

$D -$ Пам'ять переходів

$N \in \mathbb{N} -$ к-сть проходів середовища

$n \in \mathbb{N} -$ к-сть переходів для навчання

$k \in \mathbb{N} -$ час оновлення

Цикл N разів :

Визначити S_0

$I \leftarrow 0$

Поки $S_t \neq$ термінальний стан:

$I \leftarrow I + 1$

Обрати A з $S,$ використовуючи $\epsilon -$ гладуку стратегію при Q_b

Виконати дію $A,$ та визначити R, S' :

$D \leftarrow S_t, A, R, S'$:

$D \rightarrow D_t -$ Обрати n випадкових переходи

Оптимізувати Q_t на даних D_t

Якщо $I \ mod \ k = 0 : Q_b \leftarrow Q_t.$

Для вирішення задачі розробки системи керування агентами фондового ринку було обрано каскадні нео-фазі нейронні мережі, що використовуються як апроксиматори функції вартості.

2.8 Висновки до розділу

У цьому розділі було наведено теоретичні основи для використання глибинного навчання при вирішенні задач навчання з підкріпленням. Розглянутий математичний апарат повністю описує весь шлях розробки методів від динамічного програмування до апроксимації штучними нейронними мережами. Цей розділ закладає фундамент для продовження дослідження і переходу до вирішення практичної проблеми створення системи керування агентами фондового ринку.

3 НЕЙРОННІ МЕРЕЖІ

3.1 Вступ до розділу

Штучні нейронні мережі - це математична модель, яка широко використовується в машинному навчанні. Мотеди, які використовують нейронні мережі відносять до глибинного навчання (англ. Deep learning). Глибинне навчання має довгу історію і розвивалось під впливом багатьох ідей з взятих із навколишнього світу, зокрема будови людського мозку.

На сьогоднішній день глибинне навчання знаходиться у фазі швидкого розвитку і застосовується для вирішення багатьох прикладних проблем. Підходи, що використовують штучні нейронні мережі стають найкращими в індустрії останні декілька років. Це штовхає інженерну спільноту на створення нових технологій в областях апаратного забезпечення та програмних продуктів.

Швидке нарощення потужностей сучасних відеокарт зробило можливим використання та навчання надглибоких та важких нейронних мереж. Зокрема такий гігант індустрії, як Nvidia в останніх своїх розробках пропонує чіпи, що ефективно виконують тензорні операції на своїх ядрах. Їх вклад в розвиток глибинного навчання важко переоцінити, оскільки вони дали можливість прискорити роботу штучних нейронних мереж в сотні разів. На рисунку 3.1 показано порівняння ефективності роботи процесора та відеокарт при тренуванні нейронних мереж.

Серед програмного забезпечення також відбувся бум за останні роки. Найпопулярнішими фреймворками для розробки штучних нейронних мереж [9] є tensorflow, pytorch, keras, theano, caffe. Вони дозволяють швидко налаштувати нейронні мережі завдяки зручним інтерфейсам написаним на Python. Також ці фреймворки використовують розширення мови C - CUDA для використання потенціалу паралельних обчислень на відеокартах Nvidia.

Зростаючу потребу в обчислювальних потужностях використали великі компанії для побудови хмарних сервісів, що могли б обслуговувати клієнтів. Такі

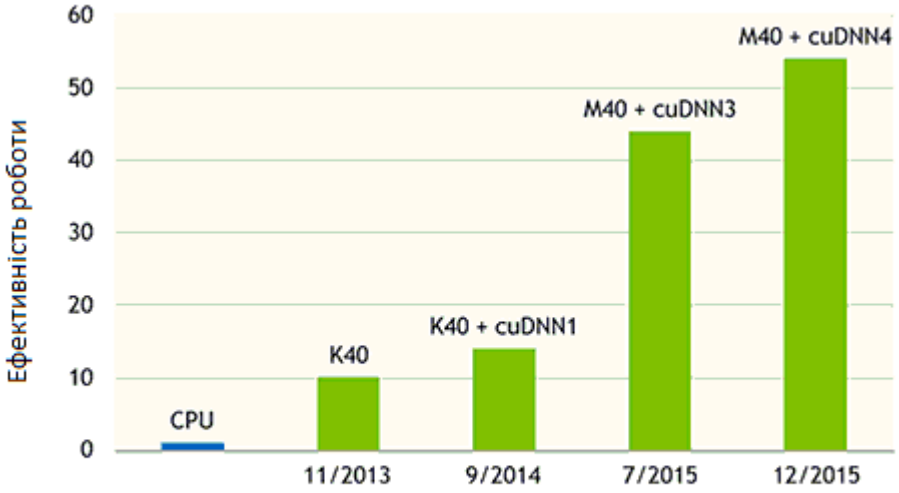

Рисунок 3.1 – Порівняння ефективності роботи процесора та відеокарт .

платформи, як Microsof Azure, Google Cloud, Amason Web Service(AWS) зараз широко використовуються, щоб перекласти всі ризики, пов'язані з апаратним забезпеченням на контрагентів. Така модель дає змогу швидко розгортати та тестувати нейронні мережі на практиці.

Завдяки всім вище описаним досягненням інженерів, сьогодні ми можемо створювати дійсно потужні нейронні мережі. Додаючи додаткові шари та вузли всередині шарів, можна апроксимувати складні функції, не втрачаючи ефективності навчання. Цю властивість ми і використаємо для наближення функції вартості дій для агента, що діє в середовищі фінансового ринку.

У наступних розділах буде показано основні ідеї та методи, що були використані при застосуванні нейронних мереж для реалізації навчання агента.

3.2 Принцип роботи нейронної мережі

В класичному варіанті штучна нейронна мережа виконує функціональне перетворення вхідного вектора $X = x_1, x_2, x_3, x_4 ... x_N$ у вихідний

$Y = y_1, y_2, y_3, y_4, ...y_M$. Кожна з координат представляє собою вхідний або вихідний нейрон відповідно. Ці нейрони формують два шари нейронної мережі - вхідний та вихідний. Між цими шарами розташовуються приховані шари, нейрони яких є результатом перетворень нейронів попередніх шарів. Наприклад, нейрони другого шару розміром H формуються наступним чином:

$$h_j = \sigma(\sum_{i=1}^{N} W_{ij}^1 x_i + W_j^T), j = 1..H, \tag{3.1}$$

де W_{ij} - ваги, x_i - сигнал з вхідного шару, а σ - функція сигмоїд, що задається наступною формулою:

$$\sigma = \frac{1}{1 + e^{-x}}. \tag{3.2}$$

Решта шарів формується аналогічно. За сучасною методологією нейронна мережа розглядається як набір шарів, і значенням окремих нейронів часто нехтують. Так різні конфігурації вагів дають змогу створювати не просто повнозв'язні шари, а розроблювати структури набагато складніші, наприклад, згортки [10]. Приклад простої тришарової нейронної мережі зображено на рисунку 3.2.

Вибір функції σ не єдиноможливим. Так замість сигмоїда часто застосовують функцію гіперболічного тангенса, ReLu, RBF. Ці функції дають змогу нейронній мережі використовувати нелінійні перетворення при апроксимації. Графіки основних функцій активації зображено на малюнку 3.3.

3.3 Каскадна neo-fuzzy нейронна мережа (CNFNN)

3.3.1 Neo-fuzzy нейрон

Розглянемо neo-fuzzy нейрон [11] [12] - нелінійну систему з декількома входами і єдиним виходом, яка зображена на рисунку 3.4. Вона реалізується

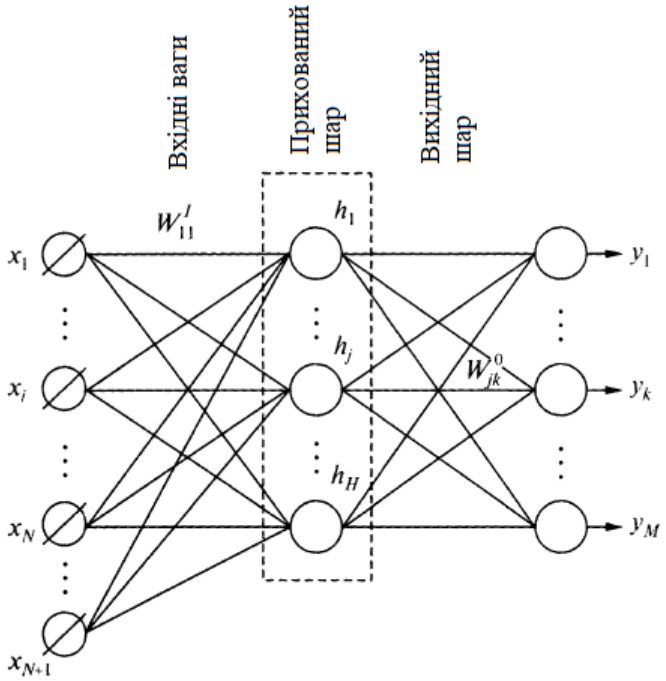

Рисунок 3.2 – Граф, що описує структуру тришарової нейронної мережі.

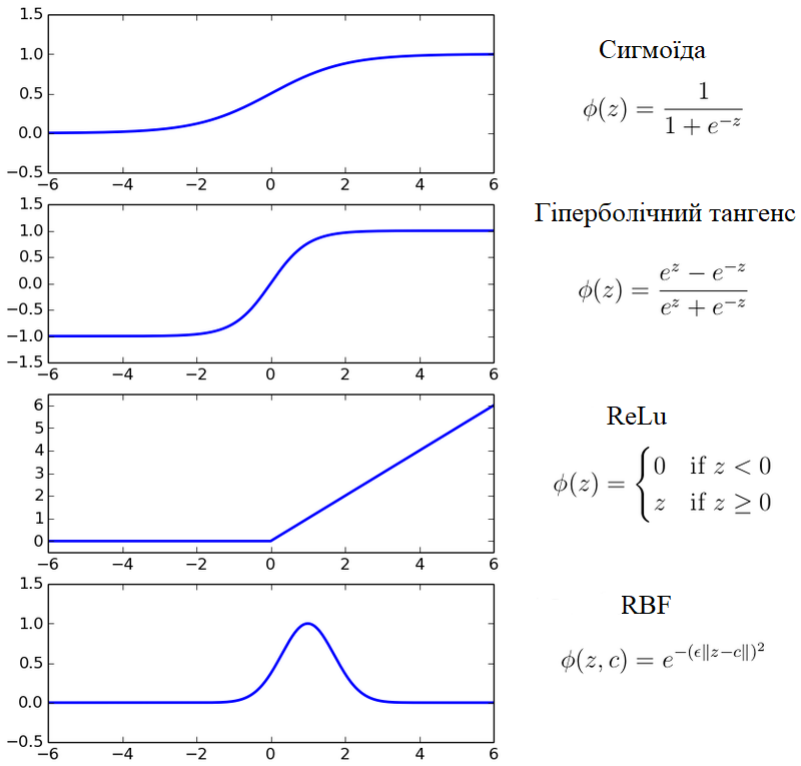

Рисунок 3.3 – Різні функції активації.

наступним відображенням:

$$\hat{y} = \sum_{i=1}^{N} f_i(x_i), \tag{3.3}$$

де x_i - і-й вхід (i = 1, 2,..., n), $\hat{y}$ - вихід системи. Структурні блоки neo-fuzzy нейрона є нелінійним синапсом, який переводить і-й вхідний сигнал в форму:

$$f_i(x_i) = \sum_{j=1}^{h} w_{ij}\mu_{ij}(x_i). \tag{3.4}$$

Neo-fuzzy нейрон виконує нечіткий висновок: $x_i x_{ij} i w_{ji}$, де x_{ij} - нечітке число, функція приналежності якого μ_{ij} , w_{ji} - синаптична вага. Очевидно, що нелінійний синапс фактично реалізує нечіткий висновок Такагі-Сугено нульового порядку.

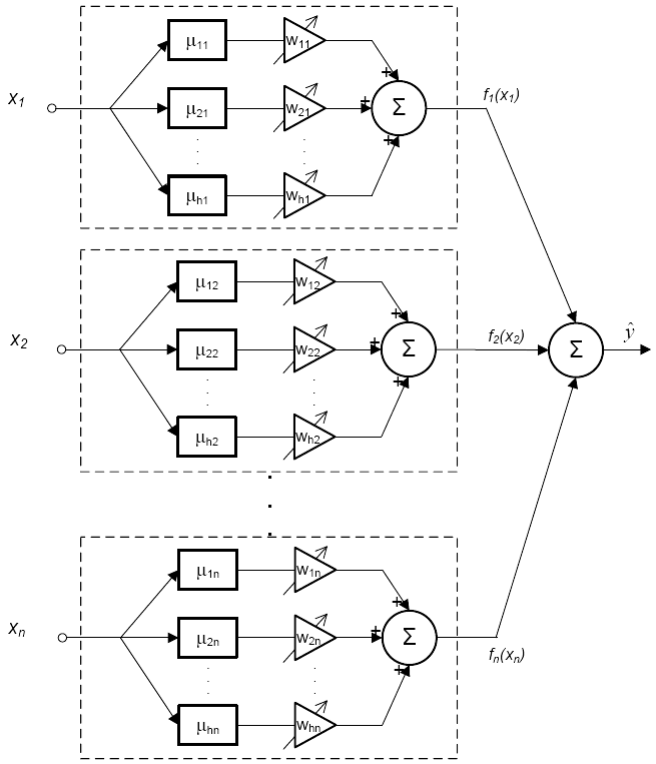

Рисунок 3.4 – Neo-fuzzy нейрон.

Традиційно функція приналежності $\mu_{ij}(x_i)$ є трикутною і показана на рисунку 3.5. Для попередніх нормалізованих вхідних змінних x_i (зазвичай $0 \leq x_i \leq 1$ — нормалізована зміна), функція приналежності має вигляд:

$$\mu_{ij}(x_i) = \begin{cases} \frac{x_i - c_{j-1,i}}{c_{j,i} - c_{j-1,i}}, & x_i \in [c_{j-1,i}, c_{j,i}], \\ \frac{c_{j+1,i} - x_i}{c_{j+1,i} - c_{j,i}}, & x_i \in [c_{j,i}, c_{j+1,i}], \\ 0, & \textit{в іншому разі} \end{cases}, \tag{3.5}$$

де $c_j i$ довільно відібрані центри відповідних функцій приналежності. Зазвичай вони рівномірно розташовані в інтервалі $[0, 1]$. Це необхідно, щоб спростити нечіткий процес виведення. Таким чином, вхідний сигнал x_i активізує тільки дві сусідніх функції приналежності і суми ступенів цих двох функцій приналежності рівні 1, тобто $\mu_{ji}(x_i) + \mu_{j+1,i}(x_i) = 1$.

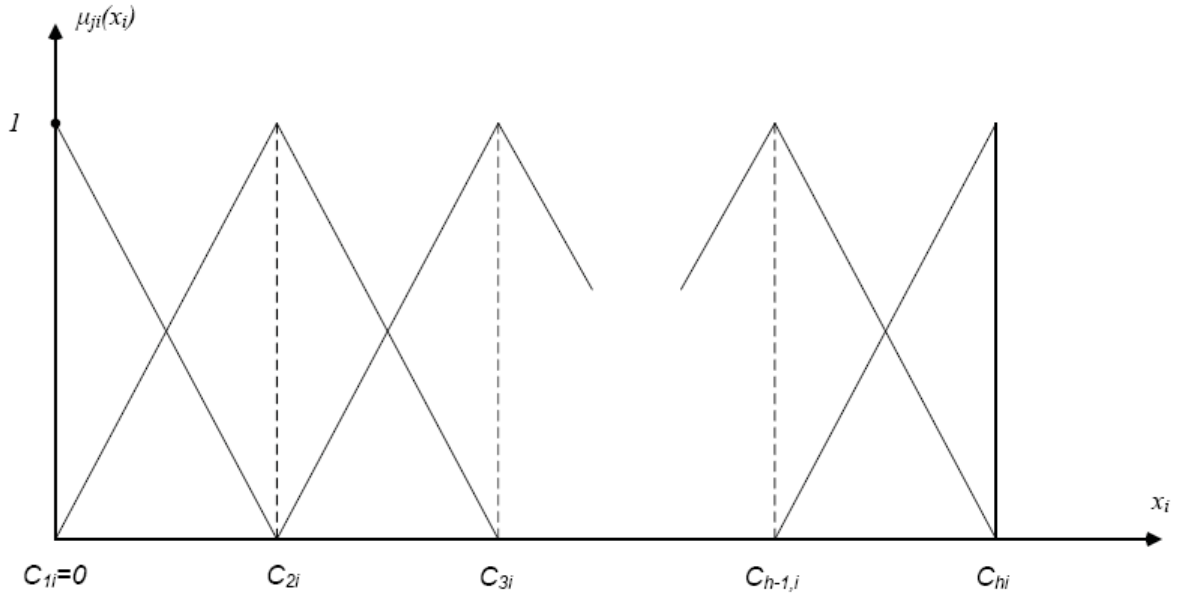

Рисунок 3.5 – Трикутна функція активації.

Таким чином, результат нечіткого виведення, використовуючи метод дифазіфікації - метод центру мас, можна представити в дуже простій формі: $f_i(x_i) = w_{ji}\mu_{ji}(x_i) + w_{j+1,i}\mu_{j+1,i}(x_i)$. Сумуючи $f_i(x_i)$ з формули 3.3 отримуємо вивід $\hat{y}$.

Коли векторний сигнал $x(k) = (x_1(k), x_2(k), ..., x_n(k))^T (k = 1, 2...)$ подається на вхід neo-fuzzy нейрона, вивід цього нейрона визначається обома функціями належності $\mu_{ji}(x_i(k))$ і налаштованим синаптичними вагами

$w_{ji}(k-1)$, які були отримані під час попередньої епохи навчання:

$$\hat{y} = \sum_{i=1}^{N} f_i(x_i(k)) = \sum_{i=1}^{N} \sum_{j=1}^{H} w_{ij}(k-1)\mu_{ij}(x_i(k)). \tag{3.6}$$

Таким чином neo-fuzzy нейрон містить $h \times n$ синаптичні ваги, які необхідно визначити.

Автори neo-fuzzy нейрона серед його найбільших переваг виділяють високу здатність до навчання, обчислювальну простоту, можливість знаходження глобального мінімуму критерію навчання в режимі реального часу. Критерій навчання (цільова функція) має вигляд стандартної квадратичної похибки:

$$E(k) = \frac{1}{2}(y(k) - \hat{y})^2 = \frac{1}{2}e(k)^2 = \frac{1}{2}(y(k) - \sum_{i=1}^{N} \sum_{j=1}^{H} w_{ij}\mu_{ij}(x_i(k)))^2. \tag{3.7}$$

Даний критерій можливо мінімізувати звичайним методом градієнтного спуску.

На основі neo-fuzzy нейронів була синтезовано дворівневу feedforward neo-fuzzy мережу. Вона має покращені здібності апроксимації в порівнянні зі звичайним багатошаровим персептроном. Дана ANN використовує backpropagation для адаптації вагів, і очевидно це призводить до зменшення показника збіжності на прихованому шарі. Ця обставина також стала причиною розвитку каскадної neo-fuzzy нейронної мережі (CNFNN), описаної в наступному підрозділі.

3.3.2 Архітектура каскадної neo-fuzzy нейронної мережі (CNFNN)

Архітектура каскадної neo-fuzzy нейронної мережі [13], яка показана на рисунку 3.6 і відображення, що її характеризує має наступну форму:

-neo-fuzzy нейрон першого каскаду:

$$\hat{y} = \sum_{i=1}^{N} \sum_{j=1}^{H} w_{ij}^{[1]} \mu_{ij}(x_i). \tag{3.8}$$

-neo-fuzzy нейрон другого каскаду:

$$\hat{y} = \sum_{i=1}^{N} \sum_{j=1}^{H} w_{ij}^{[2]} \mu_{ij}(x_i) + \sum_{j=1}^{H} w_{i,n+1}^{[2]} \mu_{i,n+1}(\hat{y}^{[1]}). \tag{3.9}$$

-neo-fuzzy нейрон третього каскаду:

$$\hat{y} = \sum_{i=1}^{N} \sum_{j=1}^{H} w_{ij}^{[3]} \mu_{ij}(x_i) + \sum_{j=1}^{H} w_{i,n+1}^{[3]} \mu_{i,n+1}(\hat{y}^{[1]}) + \sum_{j=1}^{H} w_{i,n+2}^{[3]} \mu_{i,n+2}(\hat{y}^{[2]}). \tag{3.10}$$

-neo-fuzzy нейрон m-го каскаду:

$$\hat{y} = \sum_{i=1}^{N} \sum_{j=1}^{H} w_{ij}^{[m]} \mu_{ij}(x_i) + \sum_{l=n+1}^{n+m-1} \sum_{j=1}^{H} w_{i,l}^{[m]} \mu_{i,l}(\hat{y}^{[l-n]}). \tag{3.11}$$

Отже, каскадна neo-fuzzy нейронна мережа містить $h \times (n + \sum_{l=1}^{m-1} l)$ параметрів, що настроюються і, що важливо, всі вони лінійно включені в формулу 3.11.

Нехай $h \times (n + m - 1) \times 1$ вектор функцій приналежності m-го neo-fuzzy нейрону має наступний вигляд:

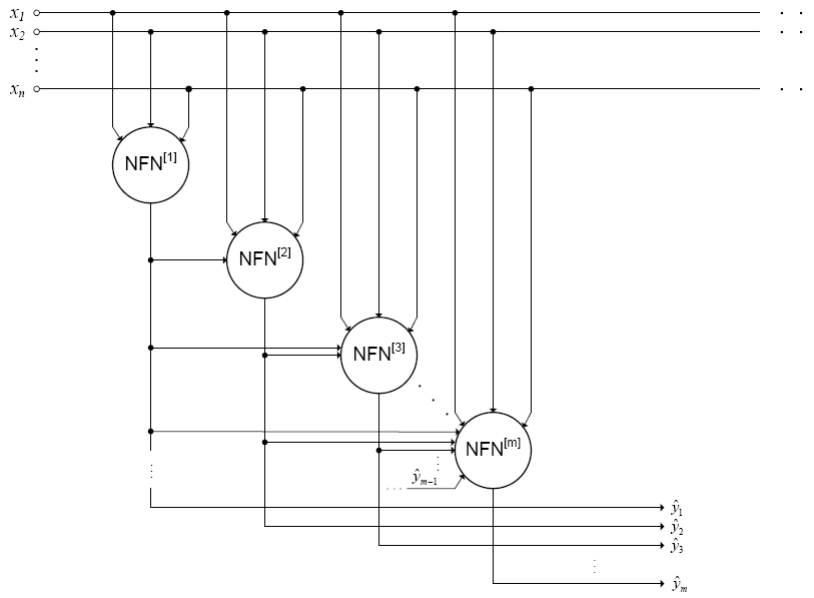

Рисунок 3.6 – Архітектура каскадної neo-fuzzy нейронної мережі.

$$\mu^{[m]} = (\mu_{11}(x_1), ..., \mu_{h1}(x_1), \mu_{12}(x_2), ..., \mu_{h2}(x_2), ..., \mu_{1i}(x_i), ..., \mu_{hn}(x_n), \mu_{1,n+1}(\hat{y}^{[1]}), ..., \mu_{h,n+1}(\hat{y}^{[1]}), ..., \mu_{h,n+m-1}(\hat{y}^{[m-1]}))^T, \tag{3.12}$$

та відповідний вектор синаптичних ваг:

$$w^{[m]} = (w_{11}^{[m]}, w_{21}^{[m]}, ..., w_{h1}^{[m]}, w_{12}^{[m]}, ..., w_{h2}^{[m]}, ..., w_{ji}^{[m]}, ..., w_{hh}^{[m]}, w_{1,n+1}^{[m]}, ..., w_{h,n_1}^{[m]}, ..., w_{h,n+m-1}^{[m]})^T. \tag{3.13}$$

Цей вектор має ту ж саму розмірність, тому ми можемо представити рівняння 3.11 в матричній формі:

$$\hat{y}^{[m]} = \hat{w}^{[m]T}\hat{\mu}^{[m]}. \tag{3.14}$$

3.3.3 Навчання каскадної neo-fuzzy нейронної мережі

Навчання каскадної neo-fuzzy нейронної мережі, може бути виконано як в пакетному режимі, так і в режимі послідовної обробки інформації (адаптивні настройки ваг).

По-перше, нехай розглядається ситуація, коли навчальна вибірка визначена апріорно, тобто ми маємо вибірку значень:

$$x(1), y(1); x(2), y(2); x(3), y(3); ...; x(k), y(k); ...; x(N), y(N). \tag{3.15}$$

Для neo-fuzzy нейрона першого каскаду вибірка значень функцій приналежності $\mu^{[1]}(1), \mu^{[1]}(2), ..., \mu^{[1]}(k), ..., \mu^{[1]}(N)$ визначається наступним чином:

$$\begin{gathered}\mu^{[1]}(k) = (\mu_{11}(x_1(k)), ..., \mu_{h1}(x_1(k)), \mu_{12}(x_2(k)), ..., \mu_{h2}(x_2(k)), ..., \\ \mu_{ji}(x_i(k)), ..., \mu_{hn}(x_n(k)))^T.\end{gathered} \tag{3.16}$$

Потім, мінімізуючи критерій навчання:

$$E_N^{[1]}(k) = \frac{1}{2}\sum_{i=1}^{n} e^{[1]}(k)^2 = \frac{1}{2}\sum_{i=1}^{n}(y(k) - \hat{y}^{[1]})(k))^2, \tag{3.17}$$

знаходимо вектор синаптичних вагів для першого каскаду.

Після навчання першого каскаду, синаптичні ваги neo-fuzzy нейрона $NFN^{[1]}$ стають "замороженими", всі значення $\hat{y}^{[1]}(1), \hat{y}^{[1]}(2), ..., \hat{y}^{[1]}(k), ..., \hat{y}^{[1]}(N)$, оцінені і ми переходимо до визначення другого каскаду мережі, який складається з єдиного neo-fuzzy нейрона $NFN^{[2]}$.Він має один додатковий вхід для сигналу виведення першого каскаду. Потім знову використовуємо алгоритм оптимізації похибки 3.17 для настройки

вектора вагових коефіцієнтів $w^{[2]}(x)$, розмірність якого $h \times (n+1) \times 1$.

Процес росту нейронної мережі (збільшення кількості каскадів) продовжується до того часу, поки ми не отримаємо необхідну точність.

3.3.4 Метод оптимізації ADAM

В процесі розробки програмного забезпечення в якості алгоритму оптимізації функції похибки 3.17 було обрано модифікацію стохастичного градієнтного спуска — алгоритм ADAM (розшифровується як adaptive moment estimation) [14]. Розглянемо принцип роботи даного алгоритму.

Нехай $E(w)$ - критерій якості вагів w. Звичайний алгоритм градієнтного спуску мав би такий вигляд:

$$\begin{gathered} \Delta = -\eta \nabla_w E(w), \\ w = w + \Delta w = w - \eta \nabla_w E(w), \end{gathered} \tag{3.18}$$

де ∇ - швидкість навчання.

Однак цей метод навчання не є ефективним на нейромережах з великою кількістю вагів. Він може збігатись до локальних мінімумів чи некоректно вести себе на складному ландшафті цільової функції, де плато часто змінюються з раптовими обривами. Тому доцільно використовувати стохастичні методи, такі як ADAM.

Основна ідея алгоритму полягає в тому, щоб напрям спуску обирати не тільки на основі теперішнього стану мережі, а й попередніх. Так, пропонується зберігати значення градієнтів, як експоненційне ковзне середнє:

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) \nabla_w E(w), \tag{3.19}$$

де β_1 - параметр моделі, що відповідає за ”довжину” пам’яті.

Наочною інтерпретацією цієї ідеї є рух кульки по поверхні. Так при

досяганні плато, кулька почне рухатись по інерції, а не зупиниться в точці, де градієнт рівний 0.

Крім того, пропонується також зберігати історію зміни градієнта у вигляді його середньої не центрованої дисперсії:

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2)(\nabla_w E(w))^2, \tag{3.20}$$

де β_2 - параметр моделі, що відповідає за "довжину" пам'яті, коефіцієнт v_t необхідний для того, щоб відслідковувати параметри, які дуже часто коливаються і зменшувати їх вплив на рух навчання.

В кінцевому випадку остаточне правило оновлення вагів виглядає наступним чином:

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t + \epsilon}} \hat{m}_t, \tag{3.21}$$

де ϵ - згладжуючий параметр, необхідний, щоб уникнути поділу на 0. Зазвичай його обирають відносно малим. η - параметр швидкості навчання.

Автори Adam пропонують в якості значень за замовчуванням $\beta_1 = 0.9, \beta_2 = 0.999, \epsilon = 10^{-8}$ і стверджують, що алгоритм виступає краще або приблизно так само, як і інші модифіковані алгоритми (Nesterov Accelerated Gradient чи Adagrad) на широкому наборі даних.

3.4 Висновки до розділу

У цьому розділі було розглянуто конкретну архітектуру нейронної мережі, що буде використовуватись для апроксимації функції вартості. Було наведено алгоритм навчання такої нейронної мережі та основні переваги нейронних мереж в індустрії на сьогоднішній день.

4 РЕАЛІЗАЦІЯ ПРОГРАМНОГО ПРОДУКТУ

4.1 Вступ до розділу

На сьогоднішній день більшість програмного забезпечення для додатків тестується та запускається на серверах. Зазвичай операційні системи що підтримують додаток є види Linux. Для забезпечення обмежень часто такі системи створюються як віртуальні машини на інших ”хостових” ОС. Незважаючи на переваги з точки зору безпеки, керованості та обмеженості віртуальних машин такі рішення мають і суттєві недоліки. До яких належать досить громіздке налаштування та погане масштабування.

Хоча зараз є досить багато рішень для подолання вище описаних проблем з віртуальними машинами, сьогодні набирає популярності рішення, що використовує контейнеризацію. Це рішення називається Docker. Його було обрано для створення додатку та підтримання сервісів, що будуть його обслуговувати.

У цьому розділі буде розглянуто основні переваги застосування Docker та показано на прикладі архітектури проекту як його правильно застосувати.

4.2 Принцип роботи Docker

На відмінну від технології віртуалізації, контейнеризація не створює окрему ОС, а обмежує доступ до ресурсів хостової ОС та реалізує ізоляцію процесів. Це робиться завдяки механізму cgroups та namespaces, що реалізовані в ядрі Linux.

До Docker входять декілька програмних засобів таких як Docker-deamon, Docker-client та засіб оркестрації такий як, наприклад Docker swarm. Docker-deamon це звичайний процес що обмежує доступ інших процесів до

ресурсів ОС. Docker-client дозволяє керувати користувачу шляхом використання командного рядка. Docker swarm - це програмне забезпечення, що дозволяє розгортати сервіси на декількох хостах з docker.

Процеси, якими керує Docker називаються контейнерами. По суті це ті ж самі процеси що ми побачимо при виклику команди pid, проте вони ізольовані від всієї системи. Схематично влаштування Docker можна побачити на рисунку 4.1.

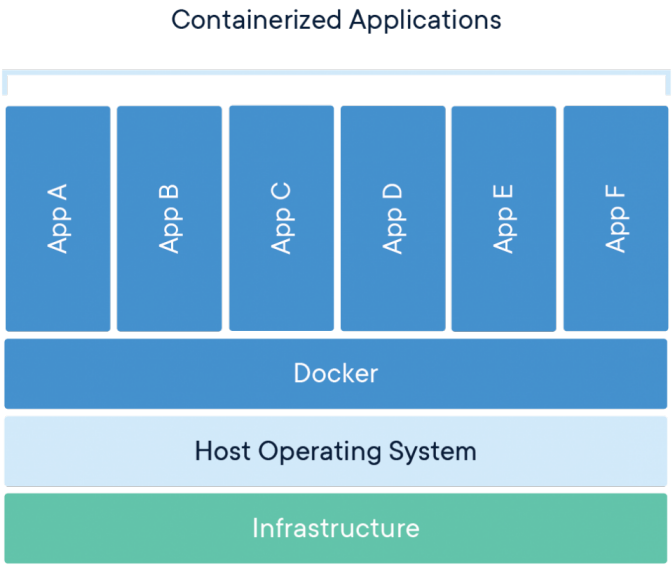

Рисунок 4.1 – Схема влаштування Docker.

Кожен контейнер в свою чергу запускається з пакету, який називається image. Image - це послідовність змін файлової системи з точки зору контейнера та відповідні команди що виконуються під час запуску контейнера. Image складається з шарів. Кожен шар - це одна модифікація Image, що на практиці відповідає одній стрічці Dockerfile, який є файлом налаштувань. Схематично всю цю структуру показана на 4.2.

Для об’єднання декількох хостів та масштабування використовується Docker swarm. Це засіб, що дає змогу організувати роботу декількох демонів docker. Для організації роботи обираються декілька менеджерів, зазвичай

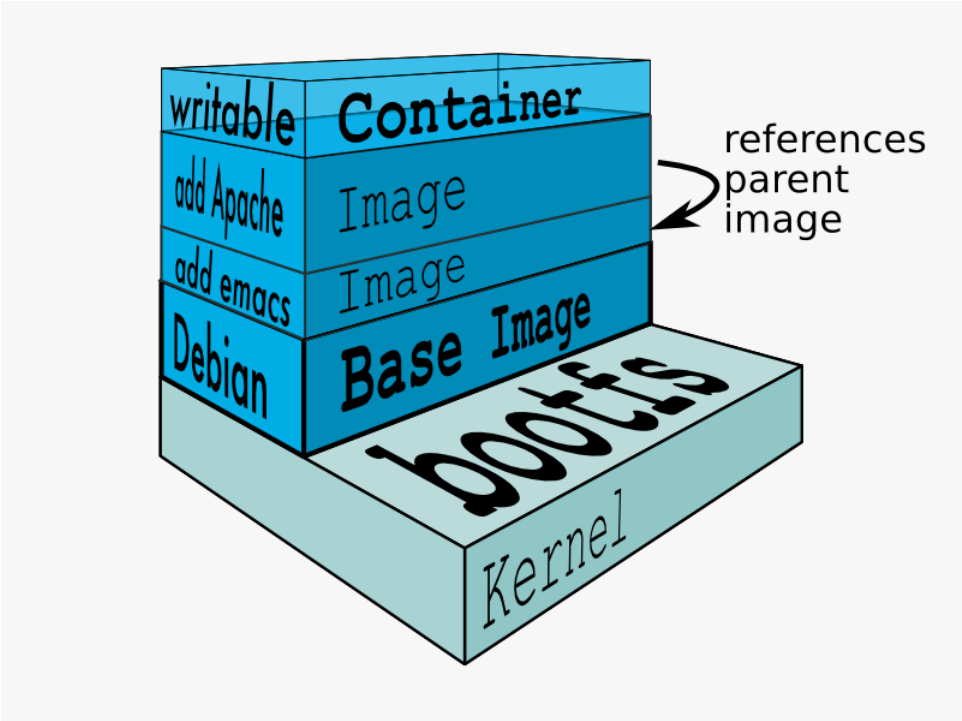

Рисунок 4.2 – Схема влаштування контейнеру.

непарна кількість, що пов'язана з механізмом синхронізації Raft та декілька працівників, що об'єднуються в єдиний кластер, що називається роєм. Схематично принцип роботи роя видно на рисунку 4.3.

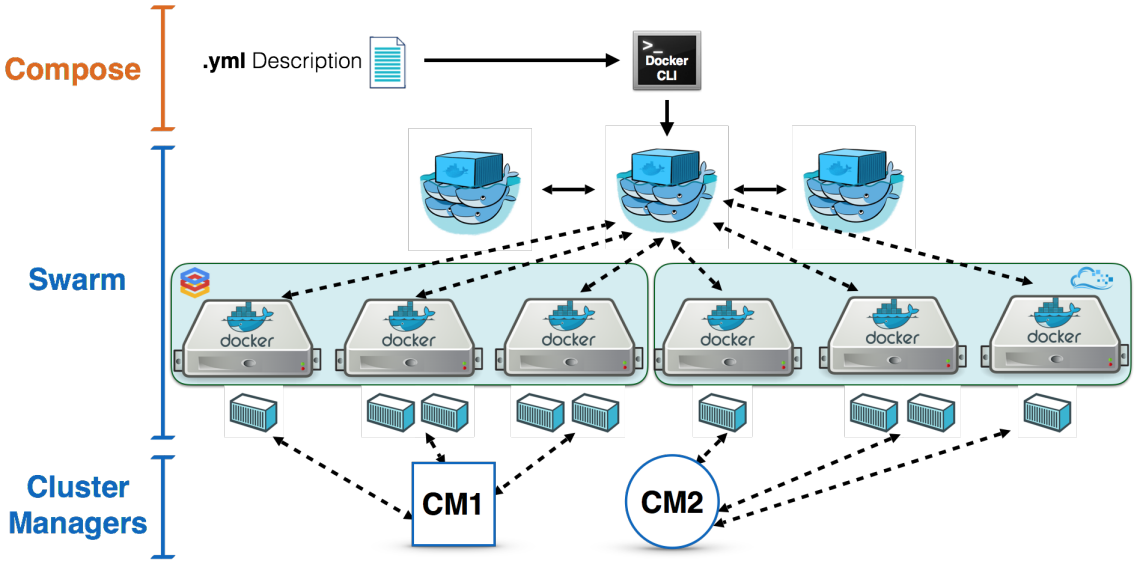

Рисунок 4.3 – Схема влаштування рою.

У роботі я використовував не декілька фізично окремих хостів, а

віртуальні машини, що розгорнуті засобом віртуалізації VirtualBox. Зазвичай docker працює як з серверами так і з хмарними рішеннями, що допомагає розподіляти задачі.

На практиці найбільш зручною та корисною властивістю Docker є управління контейнерами та налаштування, що робляться в одному файлі. Це дає можливість швидко розгорнути сервіси та запустити проект навіть не дуже досвідченому системному адміністратору чи розробнику.

Всі налаштування прописуються в dockerfile або docker-compose файлах типу yaml. Такий механізм роботи дає змогу компактно описати роботу сервісів та запустити додаток.

4.3 Побудова додатку

За планом додаток має підтримувати роботу агента, щоб той міг навчатись та торгувати на фондовій біржі. Перший пункт - навчання, покладається на нейронні мережі. Як було описано в частині 3, тренування нейронних мереж вимагає великих обчислювальних потужностей. Тому цю частину роботи було вирішено покласти на хмарний сервіс, що надається компанією Google - GoogleCloud. Таке рішення дасть змогу не витрачати гроші на побудову власної інфраструктури і перекласти всі технічні питання на постачальника послуг.

Для тренування нейронної мережі необхідні дані, які має збирати локальний сервіс з біржі. Тут ми переходимо до другого пункту, що полягає у створенні системи локальних сервісів для запиту, збору та збереження інформації, торгівлі на біржі та обміном даними з хмарним сервісом.

Першим в нашому додатку буде сервіс, що забирає дані з біржі про останні торги. Цей сервіс написано на мові програмування Python, та в ньому було створено можливість, використовуючи API біржі BitMEX, отримувати

пакети з інформацією про торги в режимі реального часу. Біржу BitMEX було обрано через те, що вона пропонує зручне API для тестування додатку. В подальшому на основі створених інтерфейсів можна буде реалізувати зв'язок з будь-якою іншою біржею.

Отримані дані потім зберігаються в нереляційній MongoDB. Дане рішення зручно використовувати для роботи з великою к-сть логованих даних, таких як інформація про торги. З даної бази наступний сервіс періодично отримує дані у хмарі, щоб тренувався нейронні мережі.

Після кожного тренування модель скидається на наступний локальний сервіс, що перезавантажує бота, який є останнім сервісом в ланцюжку. Таким чином ми нарахували як мінімум шість сервісів та два хости, на яких буде виконуватись вся робота. Схема потоку даних показана на малюнку 4.4

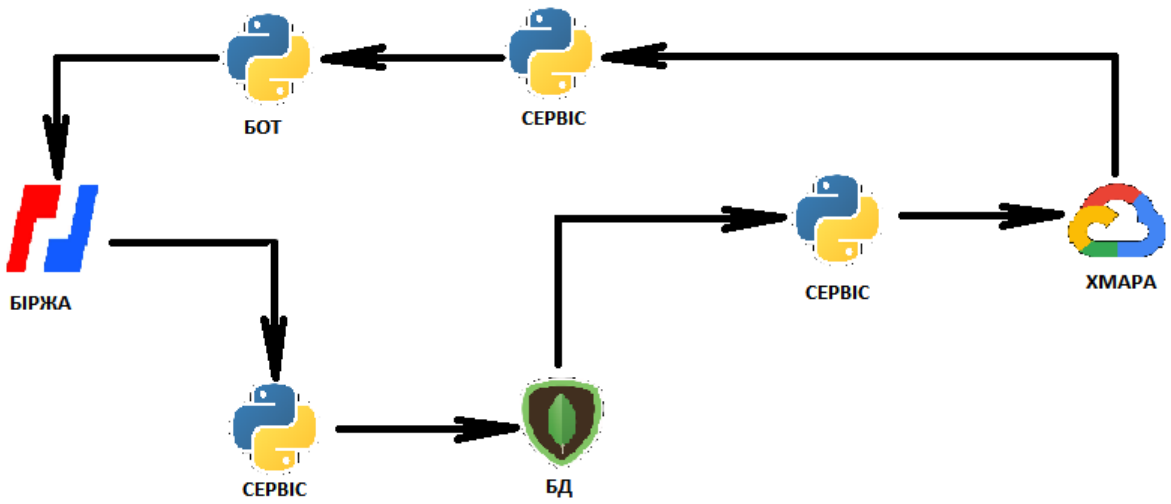

Рисунок 4.4 – Схема проходження даних через додаток.

Перевагою програмного забезпечення Docker є те, що всі налаштування можна зробити лише завдяки одному фалу, що буде запускати додаток та розгортати сервіси на хостах. Оркестрація всіх сервісів робиться завдяки іншому програмному забезпеченню docker swarm. Таке рішення є елегантним та безпечним.

4.4 Висновки до розділу

У цьому розділі було розглянуто основні концепції технології Docker. Було представлено рішення за допомогою якої було реалізовано систему, яку обслуговують декілька сервісів, два з яких знаходяться на хмарі. Дане рішення являє собою високоефективний додаток що вирішує проблеми адміністрування та розгортання системи на практиці.

5 АНАЛІЗ ТА ТЕСТУВАННЯ АГЕНТА

5.1 Вступ до розділу

Створення агента полягає не лише у розробці програмного забезпечення для реалізації навчання з підкріпленням. Дуже важливим етапом є аналіз та попередня обробка даних для найкращого використання їх моделлю.

У цьому розділі будуть розглянуті вхідні дані з біржі BitMEX. Дані цієї біржі було обрано для навчання агента через наявність API, що дозволяє швидко протестувати основні інтерфейси системи керування агентом. Незважаючи на те, що ця біржа торгує криптовалютою, дані мають ту ж саму природу, що й дані з фондових бірж. В подальшому агента можна буде використовувати для торгівлі на будь-якій біржі, написавши реалізації класів для відповідних інтерфейсів.

5.2 Огляд даних

Дані приходять з біржі кожну хвилину. У даних міститься 8 значень: час, ціна відкриття, ціна закриття, найвища ціна, найнижча ціна, об’єм валютних операцій, об’єм грошових операцій, зважена ціна. Таким чином у нас накопичується 8 часових рядів. Для тренування агента було вирішено використовувати ціну відкриття, закриття, найвищу ціну, найнижчі ціни та об’єм валютних операцій. Час зберігається для відбору даних при тренуванні.

Дані у часових рядах мають нестаціонарну природу, що може вплинути на якість тренування агента. Дані мають очевидний тренд та сезонність. Сезонність долається шляхом логарифмування ряду. Для видалення тренду було використано диференціювання, що полягає у заміні ряду на ряд різниць теперішніх даних до попередніх. Стаціонарність даних, отриманих таким способом можна перевірити за допомогою розширеного тесту Дікей-Фюлера.

Приклад стаціонарного ряду показано на рисунку 5.1.

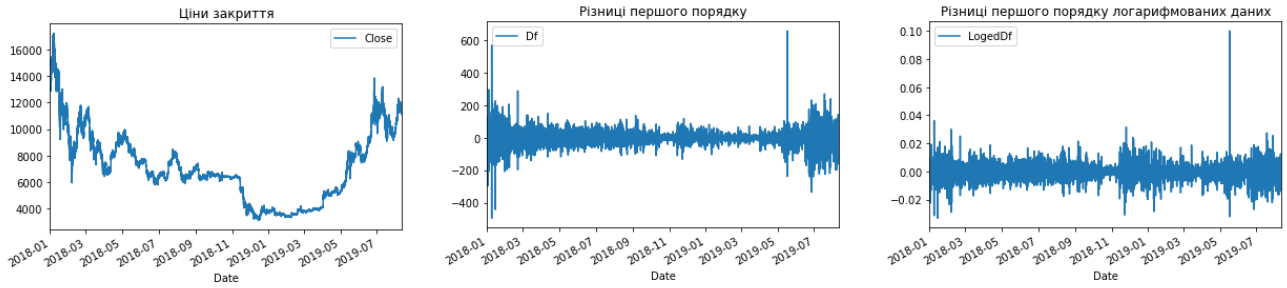

Рисунок 5.1 – Процес отримання стаціонарного ряду.

Для тренування агентів буде доцільно використовувати також різні додаткові ознаки, які можна отримати з даних. Для цього можна використати множину стандартних технічних індикаторів, що наявні, наприклад, в бібліотеці ta.

Використовувати всі індикатори не має сенсу, проте можна знайти ті, які не корелюють в межах однієї групи. Індикатори розділені на групи: моментум, індикатори об’єму, індикатори тренду та індикатори волатильності. І потім використати їх, як додаткові вхідні поля для агента. З 58 доступних ознаки було обрано 20. Кореляції ряду цін закриття показано на рисунку 5.2.

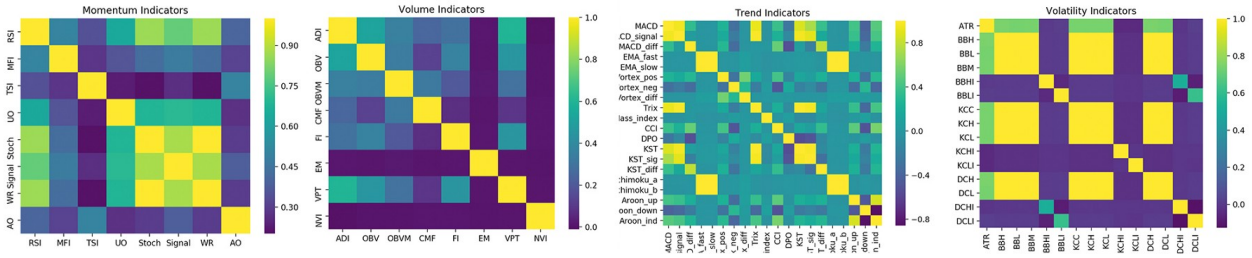

Рисунок 5.2 – Кореляційна матриця індикаторів.

Також було вирішено додати прогноз до вхідних даних. Було використано модель SARIMA для прогнозування наступного значення ряду. Ця ознака може допомогти агенту точніше приймати рішення.

5.3 Огляд агента

Після того, як ми описали вхідні дані для агента, зробимо аналіз його внутрішнього влаштування. Для цього опишемо поведінку агента та розглянемо процес його тренування.

Відповідно до постановки задачі, наш агент має автоматично торгувати на біржі. На сьогоднішній день агент може робити декілька типів ордерів на ринку. Це може бути ринковий ордер, лімітний ордер, ордер з умовою тощо. Кожна з цих заявок може вважатись окремим видом дій агента. Для спрощення ситуації будемо вважати, що агент виконує маркет ордер по ціні закриття.

Другим важливим параметром є об’єм заявки. Для нашого агента встановимо об’єм в долях від його поточного балансу. Агент матиме змогу торгувати в частинах 1/10, 2/10 і тд. за одиницю часу. Також агент може нічого не робити, в такому випадку він буде просто утримувати свій актив.

Таким чином агент може виконати 21 різну дію. А саме 10 варіантів продаж активу та 10 варіантів купівель на різний об’єм. 1 дія унікальна - тримати.

З точки зору навчання з підкріпленням, ми маємо наступне середовище: станами виступають вхідні дані, які ми отримуємо з біржі та поточний баланс агента. Дані беруться не тільки поточні, а й за 50 минулих кроків. Дії - це 20 можливих ордерів агента та бездіяльність.

Нагорода за дії агента може виступати сума, яку заробив агент за свої транзакції, проте така метрика не буде кращою. Доречно використовувати показники ефективності інвестиційної стратегії, такі як наприклад коефіцієнт Шарпа чи коефіцієнт Сортіно. В даній роботі використовувався коефіцієнт Сортіно. Цей показник розраховується, як відношення середньої премії за ризик до так званої "волатильності вниз".

Для апроксимації функції вартості агента було обрано каскадну нео-фазі нейронну мережу. Оскільки агент має постійно навчатись, то було вирішено

використовувати дві моделі. Перша буде торгувати, в той час коли друга періодично навчатись на найновіших даних, отриманих з біржі і після навчання заміняти першу. Така стратегія за задумом дає можливість відловлювати інсайти ринку за останній час і використовувати їх при торгівлі на нетривалий період.

5.4 Оцінка ефективності агента

Перед тим, як оцінювати результати ми маємо визначити еталон, з яким можна було б порівняти нашого агента. Для цього ми збираємось використовувати розповсюджені, проте ефективні стратегії для торгівлі криптовалютою. Одна з них - стратегія "Купив-тримай при якій інвестор сподівається на ріст ціни активу. Дві інші, які будуть розглядатись, використовують прості, проте ефективні методи технічного аналізу.

Стратегія "Купив-тримай"здається найбільш простою з усіх можливих - купи як можна більше і тримай до кінця життя. Незважаючи на те, що вона не складно, ця стратегія довела свою ефективність в минулому.

Друга стратегія - розходження індексу відносної сили (RSI divergence). Вона полягає в тому, що при послідовному рості цін закриття та падінні індексу RSI дає сигнал на продаж, оскільки тренд піде вниз. Сигнал на покупку виникає, коли відбувається ріст цін закриття при падінні RSI.

Третя стратегія базується на перетині простих ковзних середніх. Коли довге ковзне середнє перетинає коротке ковзне середнє зверху - виникає сигнал на продаж. Сигнал на купівлю виникає, коли перетин відбувається навпаки - коротке середня над довгим.

Сенс цих тестувань полягає в тому, щоб продемонструвати, що наш агент створює альфу на ринку і таким чином працює краще ніж в середньому ринок. За початковий стан балансу було обрано 10000 умовних одиниць. Цього має

вистачити для того, щоб вижити при початкових кроках агента, коли він може робити контрефективні ордери.

Порівняння різних стратегій зображено на рисунку 5.3

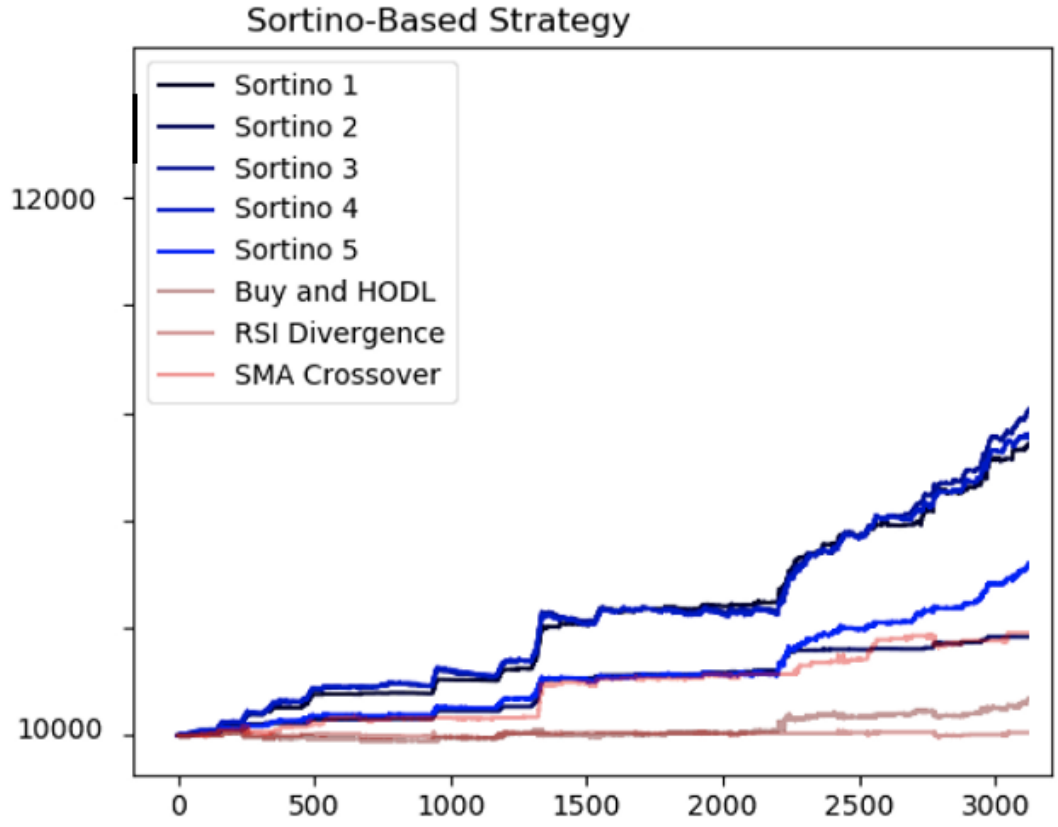

Рисунок 5.3 – Результати тестування агентів.

За результатами роботи можна сказати, що агент веде продуктивну торгівлю на ринку, що вища за найпростіші алгоритми, хоча й не показує надприбутків. На цьому моменті варто зазначити, що дана система створюється з метою навчання агентів а не їх тестування, що вимагає більш тонкого моделювання середовища. В реальних умовах ми б мали додаткові збитки за рахунок комісій за транзакції. Також моделювання по ціні закриття не дуже точне, оскільки ми не знаємо точного розподілу стакану. В реальності маркет ордер може бути не таким прибутковим.

5.5 Висновки до розділу

У даному розділі було розкрито деталі обробки вхідних даних та розкрито задум тренування агента на найновіших даних з ринку, що має забезпечити його підвищену ефективність. Було розкрито основні метрики за якими оцінюється агент та порівняно його роботу з стандартними трейдинговими алгоритмами. Отриманий результат свідчить про ефективність алгоритму в змодельованих умовах. Проте використання алгоритму на біржі вимагає більш детального моделювання та дослідження.

6 РОЗРОБКА СТАРТАП ПРОЕКТУ

6.1 Опис ідеї стартап-проекту

В межах даного підрозділу послідовно проаналізовано та подано у вигляді таблиць наступні пункти:

- зміст ідеї;
- можливі напрямки застосування;
- основні вигоди, що може отримати користувач товару (за кожним напрямом застосування);
- чим відрізняється від існуючих аналогів та замінників;

Перші три пункти подано у вигляді таблиці 6.1 і дають цілісне уявлення про зміст ідеї та можливі базові потенційні ринки, в межах яких потрібно шукати групи потенційних клієнтів.

Таблиця 6.1 – Опис ідеї стартап проекту

Зміст ідеї	**Напрямки застосування**	**Вигоди для користувача**
Створення додатку який керує діяльністю агента на фондовому ринку	Торгівля активами на фондовому ринку	Створення алгоритму, що з часом не втрачає рівня дохідності

Аналіз потенційних техніко-економічних переваг ідеї порівняно із пропозиціями конкурентів передбачає:

- визначення переліку техніко-економічних властивостей та характеристик ідеї;
- визначення попереднього кола конкурентів, проектів-конкурентів, товарів-замінників чи товарів-аналогів, що вже існують на ринку;
- збір інформації щодо значень техніко-економічних показників для ідеї

власного проекту та проектів-конкурентів.

Відповідно до визначеного вище переліку проводиться порівняльний аналіз показників: гірші значення (W, слабкі); аналогічні (N, нейтральні) значення; кращі значення (S, сильні).

Визначення сильних, слабких та нейтральних характеристик ідеї стартап-проекту ”Додатку для торгівлі на фондовому ринку” наведено у таблиці 6.2

Таблиця 6.2 – Сильні, слабкі та нейтральні характеристики ідеї проекту

№	Техніко-економічні характеристики ідеї	Значення параметру			Х-ка
		Мій проект	Звичайний бот	Інвест. фонд	
1	Форма виконання	Інтелектуаль-ний агент	Алгоритміч-ний агент	Фінансова установа	N
2	Собівартість	Відносно низька	Відносно низька	Дуже висока	S
3	Складність використання	Невисока	Невисока	Дуже висока	S
4	Автономність	Автономний	Відносно автономний	Не є автономним	S
5	Тип аналізу	Технічний	Технічний	Фунда-ментальний	N

Визначений перелік слабких, сильних та нейтральних характеристик та властивостей ідеї потенційного товару є підґрунтям для формування його конкурентоспроможності.

6.2 Технологічний аудит ідеї стартап-проекту

В межах даного підрозділу необхідно провести аудит технології, за допомогою якої можливо реалізувати ідею проекту (технології створення товару).

Визначення технологічної здійсненності ідеї проекту передбачає аналіз таких складових:

- за якою технологією буде виготовлено товар згідно ідеї проекту;
- чи існують такі технології, чи їх потрібно розробити/доробити;
- чи доступні такі технології авторам проекту? .

Технологічну здійсненність ідеї стартап-проекту ”Додаток для торгівлі на фондовому ринку” наведено у таблиці 6.3.

Таблиця 6.3 – Технологічна здійсненність ідеї стартап-проекту

№	Ідея проекту	Технології її реалізації	Наявність технологій	Доступність технологій
1	Створення додатку для керування агентами фондового ринку	Python	Наявна	Безкоштовна, доступна
		Docker	Наявна	Безкоштовна, доступна
		Tensorflow	Наявна	Безкоштовна, доступна
		Keras	Наявна	Безкоштовна, доступна

Обрана технологія реалізації ідеї проекту: для створення додатку буди обрані технології Docker, Pyhton, Tensorflow, Keras, які є безкоштовні та якими володіють розробники.

6.3 Аналіз ринкових можливостей запуску стартап-проекту

Визначення ринкових можливостей, які можна використати під час ринкового впровадження проекту, та ринкових загроз, які можуть перешкодити реалізації проекту, дозволяє спланувати напрями розвитку проекту із урахуванням стану ринкового середовища, потреб потенційних клієнтів та пропозицій проектів-конкурентів. Спочатку проводиться аналіз попиту: наявність попиту, обсяг, динаміка розвитку ринку (таблиця).

В ході таких досліджень вивчаються особливості і перспективи розвитку попиту на конкретні товари, позиції конкурентів на ринку, їх сильні і слабкі сторони, динаміку цін і т.д. Стартап-проекту важливо знати, чи буде обсяг продажів його товарів достатнім для компенсації зусиль щодо виходу на ринок, тому важливою характеристикою ринку є його ємність, під якою розуміють максимально можливий обсяг продажу певного товару протягом року, виражений в натуральних і вартісних одиницях.

Попит на більшість товару, який визначає місткість ринку, характеризується нестабільністю. Тому кожне підприємство прагне мати достовірний прогноз попиту на свій товар. З метою стимулювання збільшення попиту на товар необхідно вивчити і проаналізувати думки і потреби споживачів певного товару.

Попередню характеристику потенційного ринку стартап-проекту ”Додатку для торгівлі на фондовому ринку” наведено у таблиці 6.4.

Таблиця 6.4 – Технологічна здійсненність ідеї стартап-проекту

Показники стану ринку	**Характеристика**
Кількість головних гравців, од	2
Загальний обсяг продаж, грн/ум.од	1500000000$

Продовження таблиці 6.4

Показники стану ринку	**Характеристика**
Динаміка ринку (якісна оцінка)	Зростає
Наявність обмежень для входу (вказати характер обмежень)	Немає
Специфічні вимоги до стандартизації та сертифікації	Немає
Середня норма рентабельності в галузі (або по ринку), %	К=20%

Отже, проаналізовано наявність попиту, обсяг, динаміку розвитку ринку. Обмеження для входу на ринок відсутні, динаміка ринку зростає, галузь є рентабельною.

Характеристику потенційних клієнтів стартап-проекту ”Додатку для торгівлі на фондовому ринку” наведено у таблиці 6.5 .

Таблиця 6.5 – Технологічна здійсненність ідеї стартап-проекту

№ п/п	**Потреба, що формує ринок**	**Цільова аудиторія**	**Відмінності у поведінці різних потенційних цільових груп клієнтів**	**Вимоги споживачів до товару**
1	Ефективно торгувати на фондовому ринку	Аудиторія: індивідуальні користувачі, інвест. фонди	Для сегменту дрібних користувачів більш характерні додатки з базовим набором функцій.	Усім споживачам важлива можливість швидко заробити на ринку

Продовження таблиці 6.5

№ п/п	Потреба, що формує ринок	Цільова аудиторія	Відмінності у поведінці різних потенційних цільових груп клієнтів	Вимоги споживачів до товару
			Великі підприємства зацікавлені у додатках з широким спектром функцій та високою якістю результатів, у постійному оновленні та підтримці.	

Після визначення потенційних груп клієнтів проводиться аналіз ринкового середовища: складаються таблиці факторів, що сприяють ринковому впровадженню проекту, та факторів, що йому перешкоджають. Фактори в таблицях подають в порядку зменшення значущості.

Ринкові можливості – це сприятливі обставини, які підприємство може використовувати для отримання переваг. Слід зазначити, що можливостями з погляду SWOT-аналізу є не всі можливості, які існують на ринку, а тільки ті, які можна використовувати.

Ринкові загрози – події, настання яких може несприятливо вплинути на підприємство.

Після визначення загроз та переваг маємо виробити адекватну реакцію компанії для захисту та успішного розвитку, враховуючи особливості нашої компанії та ринку в цілому.

Фактори загроз стартап-проекту ”Додатку для торгівлі на фондовому ринку” наведено у таблиці 6.6. Фактори можливостей стартап-проекту ”Додатку для торгівлі на фондовому ринку” наведено у таблиці 6.7.

Таблиця 6.6 – Фактори загроз

№ п/п	Фактор	Зміст загрози	Можлива реакція компанії
1	Зростаюча конкуренція	Зі зростанням попиту на розробку додатків для торгівлі на фондовому ринку зросла і пропозиція.	Розробляти додаток високої якості та з додатковими унікальними функціями.
2	Зміна потреб користувачів	Користувачам необхідне програмне забезпечення з іншим функціоналом	Передбачити можливість додавання нового функціоналу до створюваного ПЗ

Таблиця 6.7 – Фактори можливостей

№ п/п	Фактор	Зміст можливості	Можлива реакція компанії
1	Зростаючий попит	Збільшення попиту на додаток	Надавати високоякісні рішення, займати нішу ринку
2	Оптимізація швидкості передачі	Оптимізація швидкості передачі даних	Оптимізація швидкості передачі за рахунок апаратної оптимізації засобів зв’язку
3	Зниження довіри до конкурентів	Конкуренти часто надають неефективні чи недовгострокові рішення	При виході на ринок звертати увагу покупців на іноваційність нашого ПЗ

Ступеневий аналіз конкуренції на ринку стартап-проекту ”Додатку для торгівлі на фондовому ринку” наведено у таблиці 6.8.

Таблиця 6.8 – Ступеневий аналіз конкуренції на ринку

№ п/п	**Особливості конкурентного середовища**	**В чому проявляється дана характеристика**	**Вплив на діяльність підприємства**
1	Тип конкуренції - досконала	Існує 2 конкурентні моделі на ринку	Врахувати відсталість конкурентних компаній на початкових етапах створення бізнесу, реклама (вказати на конкретні переваги перед конкурентами)
2	За рівнем конкурентної боротьби - міжнародний	Існують компанії з інших країн	Використовувати мови міжнародного користування
3	За галузевою ознакою - внутрішньогалузева	Конкуренти мають ПЗ, яке використовується лише всередині даної галузі	Створити основу ПЗ таким чином, щоб можна було легко переробити дане ПЗ для використання у інших галузях та додавати нові модулі в існуюче
4	Конкуренція за видами товарів: - товарно-видова	Види товарів є різні	Створити ПЗ, враховуючи недоліки конкурентів
5	За характером конкурентних переваг - неціновa	Вдосконалення технології створення ПЗ, щоб собівартість була нижчою	Використання менш дорогих технологій для розробки, ніж використовують конкуренти

Продовження таблиці 6.8

№ п/п	Особливості конкурентного середовища	В чому проявляється дана характеристика	Вплив на діяльність підприємства
6	За інтенсивністю - марочна	Бренди присутні	-

Після аналізу конкуренції проведено більш детальний аналіз умов конкуренції в галузі (таблиця 6.9).

Таблиця 6.9 – Аналіз конкуренції в галузі за М. Портером

Складові аналізу		Висновки
Прямі конкуренти в галузі	Навести перелік прямих конкурентів	Існує 3 конкуренти на ринку. Найбільш схожим за виконанням є конкурент 3, так як його рішення також представлене у вигляді ПЗ.
Постача-льники	Визначити фактори сили постачальників	Постачальники відсутні
Клієнти	Визначити фактори сили споживачів	Важливим для користувача є кросплатформеність ПЗ та якість його роботи
Потенційні конкуренти	Визначити бар’єри входження в ринок	Так, можливості для входу на ринок є, бо наше рішення покращує роботу
Товари -замінники	Фактори загроз з боку замінників	Товари-замінники можуть використати більш дешеву технологію створення ПЗ та зменшити собівартість товару

За результатами аналізу таблиці зроблено висновок щодо принципової можливості роботи на ринку з огляду на конкурентну ситуацію. Також зроблено висновок щодо характеристик (сильних сторін), які повинен мати проект, щоб бути конкурентоспроможним на ринку. Другий висновок враховується при формулюванні переліку факторів конкурентоспроможності. На основі аналізу конкуренції, проведеного в таблиці 6.9, а також із урахуванням характеристик ідеї проекту 6.2, вимог споживачів до товару (таблиця 6.7) та факторів маркетингового середовища (таблиця 6.8 – таблиця 6.9) визначено та обґрунтовано перелік факторів конкурентоспроможності. Аналіз оформляється за таблиця 6.10.

Таблиця 6.10 – Обґрунтування факторів конкурентоспроможності

№	**Фактор конкуренто-спроможності**	**Обґрунтування (наведення чинників, що роблять фактор для порівняння конкурентних проектів значущим)**
1	Виконання ПЗ у вигляді додатоку	Це рішення дозволяє швидко встановлювати та використовувати ПЗ
2	Простота інтерфейсу користувача	Інтерфейс користувача зроблений таким чином, що користувачу необхідно лише виконати декілька каманд.
3	Наявність моделей ШІ	Це дозволить надати користувачеві інформацію, яка може спростити його работу

За визначеними факторами конкурентоспроможності (таблиця 6.10) проведено аналіз сильних та слабких сторін стартап-проекту (таблиця 6.11).

Таблиця 6.11 – Порівняльний аналіз сильних та слабких сторін проекту

№	**Фактор конкуренто -спроможності**	**Бали 1-20**	**Рейтинг товарів-конкурентів у порівнянні з нашим підприємством**						
			-3	**-2**	**-1**	**0**	**1**	**2**	**3**

Продовження таблиці 6.11

1	Автоматична ідентифікація ризиків	20			+				
2	Простота інтерфейсу користувача	15	+						

Фінальним етапом ринкового аналізу можливостей впровадження проекту є складання SWOT-аналізу (матриці аналізу сильних (Strength) та слабких (Weak) сторін, загроз (Troubles) та можливостей (Opportunities) (таблиця 6.12) на основі виділених ринкових загроз та можливостей, та сильних і слабких сторін (таблиця 6.11). Перелік ринкових загроз та ринкових можливостей складено на основі аналізу факторів загроз та факторів можливостей маркетингового середовища. Ринкові загрози та ринкові можливості є наслідками (прогнозованими результатами) впливу факторів, і, на відміну від них, ще не є реалізованими на ринку та мають певну ймовірність здійснення. Наприклад: зниження доходів потенційних споживачів – фактор загрози, на основі якого можна зробити прогноз щодо посилення значущості цінового фактору при виборі товару та відповідно, – цінової конкуренції (а це вже – ринкова загроза).

Таблиця 6.12 – SWOT-аналіз стартап-проекту

Сильні сторони: простий інтерфейс користувача, інтеграція с jira, наявність списку можливих ризиків, аналіз великих даних	**Слабкі сторони:** доступно тільки англійською мовою, список можливих ризиків неспеціалізований, неточності у визначенні ключових слів

Продовження таблиці 6.12

Можливості: зростання популярності торгівлі на біржі у широкої аудиторії	**Загрози:** конкуренція, зміна потреб користувачів, поява подібного функціоналу в конкурентів

На основі SWOT-аналізу розроблено альтернативи ринкової поведінки (перелік заходів) для виведення стартап-проекту на ринок та орієнтовний оптимальний час їх ринкової реалізації з огляду на потенційні проекти конкурентів, що можуть бути виведені на ринок (див. таблицю 6.9, аналіз потенційних конкурентів). Визначені альтернативи проаналізовано з точки зору строків та ймовірності отримання ресурсів (таблиця 6.13).

Таблиця 6.13 – Альтернативи ринкового впровадження стартап-проекту

№	**Альтернатива (орієнтовний комплекс заходів) ринкової поведінки**	**Ймовірність отримання ресурсів**	**Строки реалізації**
1	Створення додатку з використання технології Tensorflow	75%	6 місяців
2	Створення додатку з використання технології Pytorch	40%	5 місяців

Обираємо альтернативу 1.

З означених альтернатив обирається та, для якої: а) отримання ресурсів є більш простим та ймовірним; б) строки реалізації – не набагато більші. Враховуючи, що наявність веб-застосунку збільшить ймовірність отримання ресурсів, то обираємо перший варіант.

6.4 Розроблення ринкової стратегії проекту

Розроблення ринкової стратегії першим кроком передбачає визначення стратегії охоплення ринку: опис цільових груп потенційних споживачів (таблиця 6.14).

Таблиця 6.14 – Вибір цільових груп потенційних споживачів

№	**Опис профілю цільової групи потенційних клієнтів**	**Готовність споживачів сприйняти продукт**	**Орієнтовний попит в межах цільової групи (сегменту)**	**Інтенсивність конкуренції в сегменті**	**Простота входу у сегмент**
1	Приватні особи	Середня	Високий	Велика	Легка
2	Фонди	Висока	Середній	Середня	Важко

За результатами аналізу потенційних груп споживачів (сегментів) автори ідеї обирають цільові групи, для яких вони пропонуватимуть свій товар, та визначають стратегію охоплення ринку:

- якщо компанія зосереджується на одному сегменті – вона обирає стратегію концентрованого маркетингу;

- можливі напрямки застосування;

- якщо працює із кількома сегментами, розробляючи для них окремо програми ринкового впливу – вона використовує стратегію диференційованого маркетингу;

- кщо компанія працює з усім ринком, пропонуючи стандартизовану програму (включно із характеристиками товару/послуги) – вона використовує масовий маркетинг. Для роботи в обраних сегментах ринку сформовано базову

стратегію розвитку (таблиця 6.15)

Таблиця 6.15 – Визначення базової стратегії розвитку

№	**Обрана альтернатива розвитку проекту**	**Стратегія охоплення ринку**	**Ключові конкурентоспроможні позиції відповідно до обраної альтернативи**	**Базова стратегія розвитку**
1	Створення додатку для торгівлі на фондовому ринку	Ринкове по-зиціювання	Простота інтерфейсу, пришвидшення роботи, легкість вбудовуємості, можливість налаштування параметрів	Диферен-нціації

Наступним кроком є вибір стратегії конкурентної поведінки (таблиця 6.16).

Таблиця 6.16 – Визначення базової стратегії конкурентної поведінки

№	**Чи є проект «першо-прохідцем» 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	Так	Так: базові функції керування	Зайняття конкурентної ніші

На основі вимог споживачів з обраних сегментів до постачальника

(стартап-компанії) та до продукту (див. таблицю 6.5), а також в залежності від обраної базової стратегії розвитку (таблицю 6.15) та стратегії конкурентної поведінки (таблиця 6.16) розробляється стратегія позиціонування (таблиця 6.17), що полягає у формуванні ринкової позиції (комплексу асоціацій), за яким споживачі мають ідентифікувати торгівельну марку/проект.

Таблиця 6.17 – Визначення базової стратегії конкурентної поведінки

№	**Вимоги до товару цільової аудиторії**	**Базова стратегія розвитку**	**Ключові конкурентоспромож-ні позиції власного стартап-проекту**	**Вибір асоціацій, які мають сформувати комплексну позицію власного проекту (три ключових)**
1	Швидкість роботи, відповідність результатів	Диферен-ціації	Повністю автоматична торгівля	Швидкість легкість, точність, великі дані, аналітика

Результатом виконання підрозділу стала узгоджена система рішень щодо ринкової поведінки стартап-компанії, яка визначає напрями роботи стартап-компанії на ринку.

6.5 Розроблення маркетингової програми стартап-проекту

Першим кроком є формування маркетингової концепції товару, який отримує споживач. Для цього у таблтці 6.18 підсумовано результати попереднього аналізу конкурентоспроможності товару. Концепція товару - письмовий опис фізичних та інших характеристик товару, які сприймаються споживачем, і набору вигод, які він обіцяє певній групі споживачів.

Таблиця 6.18 – Визначення базової стратегії конкурентної поведінки

№	Потреба	Вигода, яку пропонує товар	Ключові переваги перед конкурентами (існуючі або такі, що потрібно створити
1	Витрачати менше часу на торгівлю	Автоматична торгівля	Економія часу та зусиль
2	Можливість більше заробляти	Наявність оптимального агента	Користувач обирає агента для тогрівлі

Розроблена трирівнева маркетингова модель товару: уточнюється ідея продукту та/або послуги, його фізичні складові, особливості процесу його надання (таблиця 6.19).

Таблиця 6.19 – Опис трьох рівнів моделі товару

Рівні товару	Сутність та складові		
I. Товар за задумом	Товар допомагає користувачам автоматично торгувати на біржі без здійснення додаткових налаштувань штучного агента.		
I. Товар у реальному виконанні	Характеристики	М/Нм	Вр/Тх/Тл/Е/Ор
	1) додаток у смарфоні; 2) Простота у використанні; 3) Можливість розширення	-	-
	Якість: згідно до стандарту ISO 4444 буде проведено тестування		
	Маркування відсутнє		

Продовження таблиці 6.19

Рівні товару	**Сутність та складові**
III. Товар із підкріплен-ням	Безкоштовна версія з урізаним функціоналом
	Постійна підтримка для користувачів

1-й рівень - При формуванні задуму товару вирішується питання щодо того, засобом вирішення якої потреби і / або проблеми буде даний товар, яка його основна вигода. Дане питання безпосередньо пов’язаний з формуванням технічного завдання в процесі розробки конструкторської документації на виріб.

2-й рівень - Цей рівень являє рішення того, як буде реалізований товар в реальному/ включає в себе якість, властивості, дизайн, упаковку, ціну.

3-й рівень - Товар з підкріпленням (супроводом) - додаткові послуги та переваги для споживача, що створюються на основі товару за задумом і товару в реальному виконанні (гарантії якості , доставка, умови оплати та ін)

За рахунок чого потенційний товар буде захищено від копіювання: ноу-хау.

Після формування маркетингової моделі товару слід особливо відмітити – чим саме проект буде захищено від копіювання. Захист може бути організовано за рахунок захисту ідеї товару (захист інтелектуальної власності), або ноу-хау, чи комплексне поєднання властивостей і характеристик, закладене на другому та третьому рівнях товару. Наступним кроком визначено цінові межі, якими необхідно керуватись при встановленні ціни на потенційний товар, яке передбачає аналіз ціни на товари-аналоги або товари субститути, а також аналіз рівня доходів цільової групи споживачів (таблиця 6.20). Аналіз проводився експертним методом.

Таблиця 6.20 – Визначення меж встановлення ціни

№	**Рівень цін на товари-замінники**	**Рівень цін на товари-аналоги**	**Рівень доходів цільової групи споживачів**	**Верхня та нижня межі встановлення ціни на товар/послугу**
1	1000	1500	200000	500

Наступним кроком є визначення оптимальної системи збуту, в межах якого приймається рішення (таблиця 6.21):

- проводити збут власними силами або залучати сторонніх посередників (власна або залучена система збуту);
- вибір та обґрунтування оптимальної глибини каналу збуту;
- вибір та обґрунтування виду посередників.

Таблиця 6.21 – Формування системи збуту

№	**Специфіка закупівельної поведінки цільових клієнтів**	**Функції збуту, які має виконувати постачальник товару**	**Глибина каналу збуту**	**Оптимальна система збуту**
1	Купують ПЗ та роблять щорічні внески для подовження ліцензії	Продаж	1(через посередника)	Власна та через посередників

Останньою складової маркетингової програми є розроблення концепції маркетингових комунікацій, що спирається на попередньо обрану основу для позиціонування, визначену специфіку поведінки клієнтів (таблиця 6.22).

Таблиця 6.22 – Концепція маркетингових комунікацій

№	**Специфіка поведінки цільових клієнтів**	**Канали комунікацій, якими користуються цільові клієнти**	**Ключові позиції, обрані для позиціонування**	**Завдання рекламного повідомлення**	**Концепція рекламного звернення**
1	Купівля ліцензій на використання через інтернет повної версії	Інтернет	автоматична торгівля на біржі	Показати переваги ПЗ, у тому числі і перед конкурентами	Демо-ролик із використанням

Результатом пункту 5 є ринкова (маркетингова) програма, що включає в себе концепції товару, збуту, просування та попередній аналіз можливостей ціноутворення, спирається на цінності та потреби потенційних клієнтів, конкурентні переваги ідеї, стан та динаміку ринкового середовища, в межах якого впроваджено проект, та відповідну обрану альтернативу ринкової поведінки

6.6 Висновки до розділу

Згідно до проведених досліджень:

- існує можливість ринкової комерціалізації проекту;

- існують перспективи впровадження з огляду на потенційні групи клієнтів, бар’єри входження високі, але проект має одну значну перевагу перед конкурентами;

- необхідно реалізувати додаток штучного торгового агента за технологією Python Tensorflow для керування торгами автоматично;
- подальша імплементація є доцільною.

ВИСНОВКИ

У даній роботі розглянуто новий підхід для ведення торгів на фондовій біржі. Цей підхід базується на глибинному навчанні з підкріпленням. Впродовж другого розділу було розглянуто математичний апарат, що є основою для створення системи керування агентом фондового ринку.

В третьому розділі описується архітектура нейронних мереж, на основі яких робиться апроксимація функцій вартості дій агента в середовищі фондового ринку. Нео-фазі нейронні мережі є ключовим новим елементом при вирішенні заданої задачі.

В четвертому розділі було розглянуто технології, що дозволяють швидко розробити додаток, як систему взаємопов’язаних сервісів. Було описано архітектуру розробленого ПЗ та розкрито схему взаємодії окремих сервісів.

В п’ятому розділі було розглянуто основні методи аналізу та обробки вхідних даних. Розкрито ідею ітеративного перенавчання агента впродовж всього його існування для підвищення його ефективності. Було продемонстровано результати тестування та порівняно його з іншими стандартними методами торгівлі на біржі BitMEX. В результаті можна зробити висновок про створення життєздатної системи керування агентами фондового ринку.

Наприкінці було проведено дослідження можливості ринкової комерціалізації продукту. В результаті було визначено доцільним проводити подальший розвиток та впровадження розробленого програмного забезпечення.

ПЕРЕЛІК ПОСИЛАНЬ

1. Sutton R. S., Barto A. G. Reinforcement Learning–An Introduction. Cambridge, MT: MIT Press, 1998. 548 p.

2. Werbos P. J. Approximate dynamic programming for real-time control and neural modeling. *Handbook of Intelligent Control*. New York: Van Nostrand Reinhold, 1992. P. 493–525.

3. Bellman R. E. Dynamic Programming. Princeton: Princeton University Press, 1957. 65 p.

4. Hewer G. An iterative technique for the computation of the steady state gains for the discrete optimal regulator. *IEEE Transactions on Automatic Control*. 1971. Vol.16, No.6. P. 382–384.

5. Lancaster P., Rodman L. Algebraic Riccati Equations. Oxford: Oxford University Press, 1995. 504 p.

6. Watkins C. J. C. H., Dayan P. Q-learning. *Machine Learning*. 1992. Vol.8, No.1. P.279–292.

7. Bertsekas D. P., Tsitsiklis J. N. Neuro-Dynamic Programming. Atheba: Athena Scientific, 1996. 504 p.

8. Mnih V., Kavukcuoglu K., Silver D., Graves A., Antonoglou I., Wierstra D., Riedmiller M. Playing Atari with Deep Reinforcement Learning. *NIPS Deep Learning Workshop*. 2013.

9. Nick McClure, TensorFlow Machine Learning Cookbook. New Yourk: Packt Publishing, 2017. 370 p.

10. LeCun Y., Jackel L. D., Boser B., Denker J. S., Graf H. P., Guyon I., Henderson D., Howard R. E., Hubbard W. Handwritten digit recognition: Applications of neural network chips and automatic learning. *IEEE Communications Magazine*. 1989. Vol.27, No.11. P. 41–46.

11. Yamakawa T., Uchino E., Miki T., Kusanagi H. A neo-fuzzy neuron and its applications to system identification and prediction of the system begavior. *Proc. 2nd Intern. Conf. Fuzzy Logic and Neural Networks ”LIZUKA-92”*. Lizuka, Japan,

July 17-22, 1992. P.477-483.

12. Bodyanskiy Ye., Zaychenko Yu., Pavlikovskaya E. et al. Neo-fuzzy neural network structure optimization using GMDH for solving forecasting and classsification problems. *Proc. Int. Workshop on Inductive Modkeling*. Lviv, Ukraine, September 17-20, 2009. - P.77-89

13. М.З. Згуровский, Ю.П. Зайченко. Основы вычислительного интеллекта. Киев: Изд. ”Наукова Думка”, 2013. 406 с.

14. Diederik P. Kingma, Jimmy Ba. Adam. A Method for Stochastic Optimization. *3rd International Conference for Learning Representations*. San Diego, California, USA, May 7-9, 2015. Conference Track Proceedings. P.113-128.

Додаток А

Лістинг програми

Лістинг файлу actor-critic.py — Реалізація нейронних мереж

```
import tensorflow as tf
import numpy as np
from config import get_config


class Actor:
    """The actor class"""

    def __init__(self, sess, num_actions, observation_shape, config):
        self._sess = sess

        self._state = tf.placeholder(dtype=tf.float32, shape=observation_shape, name='state')
        self._action = tf.placeholder(dtype=tf.int32, name='action')
        self._target = tf.placeholder(dtype=tf.float32, name='target')

        self._hidden_layer = tf.layers.dense(inputs=tf.expand_dims(self._state, 0), units=32, activation=tf.nn.relu,
             kernel_initializer=tf.zeros_initializer())
        self._output_layer = tf.layers.dense(inputs=self._hidden_layer, units=num_actions, kernel_initializer=tf.zeros_initializer())
        self._action_probs = tf.squeeze(tf.nn.softmax(self._output_layer))
        self._picked_action_prob = tf.gather(self._action_probs, self._action)

        self._loss = -tf.log(self._picked_action_prob) * self._target

        self._optimizer = tf.train.AdamOptimizer(learning_rate=config.learning_rate)
        self._train_op = self._optimizer.minimize(self._loss)

    def predict(self, s):
        return self._sess.run(self._action_probs, {self._state: s})

    def update(self, s, a, target):
        self._sess.run(self._train_op, {self._state: s, self._action: a, self._target: target})


class Critic:
    """The critic class"""

    def __init__(self, sess, observation_shape, config):
        self._sess = sess
        self._config = config
        self._name = config.critic_name
        self._observation_shape = observation_shape
        self._build_model()

    def _build_model(self):
        with tf.variable_scope(self._name):
            self._state = tf.placeholder(dtype=tf.float32, shape=self._observation_shape, name='state')
            self._target = tf.placeholder(dtype=tf.float32, name='target')

            self._hidden_layer = tf.layers.dense(inputs=tf.expand_dims(self._state, 0), units=32, activation=tf.nn.relu,
                 kernel_initializer=tf.zeros_initializer())
            self._out = tf.layers.dense(inputs=self._hidden_layer, units=1, kernel_initializer=tf.zeros_initializer())

            self._value_estimate = tf.squeeze(self._out)
            self._loss = tf.squared_difference(self._out, self._target)

            self._optimizer = tf.train.AdamOptimizer(learning_rate=self._config.learning_rate)
            self._update_step = self._optimizer.minimize(self._loss)

    def predict(self, s):
        return self._sess.run(self._value_estimate, feed_dict={self._state: s})

    def update(self, s, target):
        self._sess.run(self._update_step, feed_dict={self._state: s, self._target: target})


def test():
    """Function to test the model Implementation"""
    state = np.random.rand(4)
    target = np.random.rand(1)
    action = 1

    tf.reset_default_graph()
    sess = tf.Session()

    config = get_config()

    actor = Actor(sess, 4, [4], config)
    critic = Critic(sess, [4], config)

    sess.run(tf.group(tf.global_variables_initializer(), tf.local_variables_initializer()))

    actor.predict(state)
    critic.predict(state)
    actor.update(state, action, target)
```

```
    critic.update(state, target)
    pass


if __name__ == "__main__":
    test()
```

Лістинг файлу agent.py — Реалізація агента

```
import numpy as np
import tensorflow as tf
import itertools
import os
from model import Actor, Critic


class Agent:
    def __init__(self, sess, config, environment):
        # Get the session, config, environment, and create a replaymemory
        self.sess = sess
        self.config = config
        self.environment = environment

        self.init_dirs()
        self.init_cur_epsiode()
        self.init_global_step()
        self.init_summaries()

        # Intialize the graph which contain 2 Networks Actor and Critic
        self.actor = Actor(sess, self.environment.n_actions, self.environment.state_shape, config)
        self.critic = Critic(sess, self.environment.state_shape, config)

        # To initialize all variables
        self.init = tf.group(tf.global_variables_initializer(), tf.local_variables_initializer())
        self.sess.run(self.init)

        self.saver = tf.train.Saver(max_to_keep=10)
        self.summary_writer = tf.summary.FileWriter(self.summary_dir, self.sess.graph)

        if config.is_train and not config.cont_training:
            pass
        elif config.is_train and config.cont_training:
            self.load()
        elif config.is_play:
            self.load()
        else:
            raise Exception("Please␣Set␣proper␣mode␣for␣training␣or␣playing")

    def load(self):
        latest_checkpoint = tf.train.latest_checkpoint(self.checkpoint_dir)
        if latest_checkpoint:
            print("Loading␣model␣checkpoint␣{}...\n".format(latest_checkpoint))
            self.saver.restore(self.sess, latest_checkpoint)

    def save(self):
        self.saver.save(self.sess, self.checkpoint_dir, self.global_step_tensor)

    def init_dirs(self):
        # Create directories for checkpoints and summaries
        self.checkpoint_dir = os.path.join(self.config.experiment_dir, "checkpoints/")
        self.summary_dir = os.path.join(self.config.experiment_dir, "summaries/")

    def init_cur_epsiode(self):
        """Create␣cur␣episode␣tensor␣to␣totally␣save␣the␣process␣of␣the␣training"""
        with tf.variable_scope('cur_episode'):
            self.cur_episode_tensor = tf.Variable(-1, trainable=False, name='cur_epsiode')
            self.cur_epsiode_input = tf.placeholder('int32', None, name='cur_episode_input')
            self.cur_episode_assign_op = self.cur_episode_tensor.assign(self.cur_epsiode_input)

    def init_global_step(self):
        """Create␣a␣global␣step␣variable␣to␣be␣a␣reference␣to␣the␣number␣of␣iterations"""
        with tf.variable_scope('step'):
            self.global_step_tensor = tf.Variable(0, trainable=False, name='global_step')
            self.global_step_input = tf.placeholder('int32', None, name='global_step_input')
            self.global_step_assign_op = self.global_step_tensor.assign(self.global_step_input)

    def init_summaries(self):
        """Create␣the␣summary␣part␣of␣the␣graph"""
        with tf.variable_scope('summary'):
            self.summary_placeholders = {}
            self.summary_ops = {}
            self.scalar_summary_tags = ['episode.total_reward', 'episode.length', 'evaluation.total_reward', 'evaluation.length', '
                epsilon']
            for tag in self.scalar_summary_tags:
                self.summary_placeholders[tag] = tf.placeholder('float32', None, name=tag)
                self.summary_ops[tag] = tf.summary.scalar(tag, self.summary_placeholders[tag])

    def add_summary(self, summaries_dict, step):
        """Add␣the␣summaries␣to␣tensorboard"""
        summary_list = self.sess.run([self.summary_ops[tag] for tag in summaries_dict.keys()],
                                     {self.summary_placeholders[tag]: value for tag, value in summaries_dict.items()})
        for summary in summary_list:
            self.summary_writer.add_summary(summary, step)
        self.summary_writer.flush()

    def take_action(self, state):
```

```
        """Take the action"""
        action_probs = self.actor.predict(state)
        action = np.random.choice(np.arange(len(action_probs)), p=action_probs)
        return action

    def observe(self, action):
        """Function that observe the new state , reward"""
        return self.environment.step(action)

    def train_episodic(self):
        """Train the agent in episodic techniques"""

        for cur_episode in range(self.cur_episode_tensor.eval(self.sess) + 1, self.config.num_episodes, 1):

            # Save the current checkpoint
            self.save()

            # Update the Cur Episode tensor
            self.cur_episode_assign_op.eval(session=self.sess, feed_dict={self.cur_epsiode_input: self.cur_episode_tensor.eval(self.
                sess) + 1})

            state = self.environment.reset()
            total_reward = 0

            # Take steps in the environment untill terminal state of epsiode
            for t in itertools.count():

                # Update the Global step
                self.global_step_assign_op.eval(session=self.sess, feed_dict={self.global_step_input: self.global_step_tensor.eval(
                    self.sess) + 1})

                # Take an action
                action = self.take_action(state)
                next_state, reward, done = self.observe(self.environment.valid_actions[action])

                # Calculate the TD Target
                value_next = self.critic.predict(next_state)
                td_target = reward + self.config.discount_factor * value_next
                td_error = td_target - self.critic.predict(state)

                # Update the Critic
                self.critic.update(state, td_target)

                # Update the Actor
                # using the td error as our advantage estimate
                # TODO Research about the best advantage estimate
                self.actor.update(state, action, td_error)

                total_reward += reward

                if done:  # IF terminal state so exit the episode
                    # Add summaries to tensorboard
                    summaries_dict = {'episode.total_reward': total_reward,
                                      'episode.length': t}
                    self.add_summary(summaries_dict, self.global_step_tensor.eval(self.sess))
                    break

                state = next_state

        print("Training Finished")
```

Лістинг файлу wavelet.py — Реалізація нео-фазі шару нейронної мережі

```
import tensorflow as tf
import keras.backend as K
from keras.engine.topology import Layer
from keras.callbacks import Callback
from keras import regularizers

def triang_wavelet(x):
    pass

class FuzzyLayer(Layer):
    def __init__(self,
                 triangle_num,
                 output_dim = 1,
                 kernel_regularizer=None,
                 **kwargs):
        if 'input_shape' not in kwargs and 'input_dim' in kwargs:
            kwargs['input_shape'] = (kwargs.pop('input_dim'),)
        self.triangle_num = triangle_num
        self.output_dim = output_dim
        self.kernel_regularizer = regularizers.get(kernel_regularizer)
        super(FuzzyLayer, self).__init__(**kwargs)

    def build(self, input_shape):
        self.input_dimensions = list(input_shape)[:-1:-1]
        self.centers = self.add_weight(name='centers',
                                       shape=((self.triangle_num, input_shape[-1])),
                                       initializer=my_init,
                                       trainable=False)
        self.kernel = self.add_weight(name='weights',
                                      shape=((self.triangle_num, input_shape[-1])),
                                      initializer='ones',
                                      regularizer=self.kernel_regularizer,
```

```
                                    trainable=True)
        super(FuzzyLayer, self).build(input_shape)

    def call(self, x):
        centers = self.centers[None, :, :]          # Shape: 1 triangle_num, x_width
        kernel = self.kernel[None, :, :]            # Shape: 1 triangle_num, x_width
        x = x[:, None, :]                           # Shape: batch_size, 1, x_width
        dist = tf.math.subtract(centers, x)         # Shape: batch_size, triangle_num, x_width
        dist = tf.math.abs(dist)
        width = tf.constant(1 / (self.triangle_num - 1))
        val = tf.math.subtract(dist, width)
        val = tf.clip_by_value(val, -1, 0)
        val = tf.math.abs(val)
        scores = tf.math.multiply(val, tf.constant(self.triangle_num - 1.0))
        result = tf.reduce_sum(scores * self.kernel, axis=1)
        return tf.reduce_sum(result, axis=-1)

    def compute_output_shape(self, input_shape):
        return tuple(input_shape[:-1]) + (self.output_dim,)

def my_init(shape, dtype=None):
    '''Create␣centers␣of␣the␣fuzzy␣membership␣functions'''
    column = tf.range(0, 1 + K.epsilon(), delta=1 / (shape[0] - 1), dtype=None, name='range')
    column = tf.reshape(column, (shape[0], 1))
    result = K.repeat_elements(column, shape[1], axis=1)
    return result

class LossHistory(Callback):
    def on_train_begin(self, logs={}):
        self.losses = []

    def on_epoch_begin(self, epoch, logs={}):
        print('␣Epoch:␣{}'.format(logs.get('epoch')), end='')

    def on_epoch_end(self, epoch, logs={}):
        loss = logs.get('loss')
        self.losses.append(loss)
        print('␣Loss:␣{}'.format(loss))
```