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

Інститут прикладного системного аналізу
Кафедра математичних методів системного аналізу

На правах рукопису
УДК 004.855.5:519.876.2

До захисту допущено
В. о. завідувача кафедри ММСА
О.Л.Тимощук

«___» _____ 2020 р.

Магістерська дисертація

на здобуття ступеня магістра
зі спеціальності 122 Комп'ютерні науки та інформаційні технології
на тему: «Методи навчання з підкріпленням для гри в покер»

Виконала:
студентка ІІ курсу, групи КА-93мп
Туголукова Євгенія Валеріївна _____

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

Рецензент:
доцент кафедри СП, , к.т.н., доц.,
Безносик О. Ю. _____

Засвідчую, що в цій магістерській
Дисертації немає запозичень із праць
інших авторів без відповідних посилань
Студент _____

Київ
2020

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

Інститут прикладного системного аналізу
Кафедра математичних методів системного аналізу

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

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

ЗАВДАННЯ
на магістерську дисертацію студента Туголукової Євгенії Валеріївни

1. Тема дисертації: «Методи навчання з підкріпленням для гри в покер», науковий керівник дисертації Недашківська Надія Іванівна, доцент кафедри ММСА, д.т.н., доц., затверджені наказом по університету від 02 листопада 2020 р № 3182-с.

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

3. Об'єкт дослідження: гра з неповною інформацією "Безлімітний Техаський Холдем" для двох гравців.

4. Предмет дослідження: методи навчання з підкріпленням для розв'язування гри "Безлімітний Техаський Холдем" для двох гравців.

5. Перелік завдань, які потрібно розробити:

1) дослідити сучасний стан та особливості методів розв'язування гри
2) провести огляд сучасних підходів для побудови моделей для розв'язування гри
3) обрати та обґрунтувати вибір методу
4) дослідити алгоритми Deepstack, контрфактичної мінімізації шкодувань та Continual re–solving
5) згенерувати вхідні дані для експериментів
6) виконати експерименти, навчити модель на згенерованих даних та проаналізувати отриманий результат
7) розробити стартап-проект виведення на ринок результатів дослідження;
8) розробити концептуальні висновки за результатами наукового дослідження;
9) оформити наукову статтю.

6. Орієнтовний перелік графічного (ілюстративного) матеріалу

1. Рисунки схем роботи деяких алгоритмів
2. Таблиці порівняння результатів роботи
3. Рисунки та графіки на яких зображено результати роботи
4. Таблиці у розділі стартап-проекту

7. Дата видачі завдання: 1 вересня 2020 р.

Календарний план

№ з/п	Назва етапів виконання магістерської дисертації	Термін виконання етапів магістерської дисертації
1	Концептуальний вступ дисертації. Формулювання об’єкта, предмета, цілі, завдань, новизни, практичної значущості результатів	05.09.2020—13.09. 2020
2	Перший розділ. Огляд літературно-інформаційних джерел, формування нормативної бази. Характеристика об’єкта.	16.09.2020—27.09.2020
3	Другий розділ. Опис теоретичних основ методів дослідження.	30.09.2020—09.10.2020
4	Третій розділ. Огляд та аналіз існуючих методів.	09.10.2020—21.10.2020
5	Четвертий розділ. Реалізація та застосування обраної моделі. Збір та аналіз результатів.	21.10.2020—29.10.2020
6	П’ятий розділ. Стартап-проект	30.10.2020—13.11.2020
7	Концептуальні висновки. Перспективи розвитку отриманих рішень	14.11.2020—20.11.2020
8	Оформлення наукової статті	21.11.2020—26.11.2020

Студент Туголукова Є.В.

Науковий керівник дисертації Недашківська Н.І.

РЕФЕРАТ

Магістерська дисертація: 105 с., 16 рис., 20 табл., 1 додаток, 43 джерела.

Об’єктом дослідження є ігри з неповною інформацією, на прикладі безлімітного техаського холдему для двох гравців.

Предметом дослідження є методи навчання з підкріпленням для розв'язування ігор з неповною інформацією.

Мета та цілі роботи – розглянути теоретичне підґрунтя ігор з неповною інформацією, провести дослідження існуючих методів їх розв'язування, розробка програмного забезпечення, яке вміє оптимально розв'язувати окремі ситуації в безлімітному техаському холдемі, та ігрові партії в цілому, аналіз розробленого програмного продукту.

Дослідження ґрунтується на наукових публікаціях та інших матеріалах закордонних конференцій та архівів в галузі навчання з підкріпленням, глибокого навчання та пошуку виграшних стратегій в іграх.

Результатом роботи є клієнт для гри у безлімітний техаський холдем, що автоматично може приймати рішення у ігрових ситуаціях.

ГРА З НЕПОВНОЮ ІНФОРМАЦІЄЮ, РОЗШИРЕНА ФОРМА ГРИ, РІВНОВАГА НЕША, НАВЧАННЯ З ПІДКРІПЛЕННЯМ, СПІВСТАВЛЕННЯ ШКОДУВАНЬ, КОНТРФАКТИЧНІ ШКОДУВАННЯ

ABSTRACT

Master's thesis: 105 pages, 16 figures, 20 tables, 1 appendixes, 43 sources.

The object of the work are imperfect information games, on the example of No-limit Texas hold'em for two players.

The subject of the work is reinforcement learning methods for solving games with incomplete information.

The purpose and goals of the work – to consider the theoretical basis of games with incomplete information, to study existing methods of solving them, to develop software that can optimally solve individual situations in No-limit Texas Hold'em, and game in general, analysis of the developed software.

The research is based on scientific publications and other materials of foreign conferences and archives in the field of reinforcement learning, deep learning and finding winning strategies in games.

The result is a client for playing No-limit Texas Hold'em, which can automatically make decisions in game situations.

IMPERFECT INFORMATION GAME, EXTENSIVE-FORM GAME, NASH EQUILIBRIUM, REINFORCEMENT LEARNING, REGRET MATCHING, COUNTERFACTUAL REGRET

ЗМІСТ

ПЕРЕЛІК ПРИЙНЯТИХ ПОЗНАЧЕНЬ ТА СКОРОЧЕНЬ

CFR – Counterfactual Regret Minimization

RM – Regret Matching

КМШ – Контрфактична мінімізація шкодувань

БТХ – Безлімітний Техаський холдем

КНП – Камінь-ножиці-папір

ШІ – штучний інтелект

РОЗДІЛ 1 ДОСЛІДЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Актуальність задачі

Ще до винайдення комп’ютерів ігри використовувались для оцінки алгоритмів та методів штучного інтелекту (ШІ). У 1948 році Алан Тюрінг почав працювати з колегою над шаховою програмою, яку на той час не міг запустити жоден комп'ютер. Тим не менше, в 1952 році, не маючи життєздатного комп'ютера, на якому можна було б запускати програму, Тюрінг розіграв записаний матч від руки (шляхом емуляції комп'ютера), але, на жаль, програв [1]. Ігри мають деякі переконливі особливості, які роблять їх чудовим випробувальним тестом для прогресу в дослідженнях ШІ: кінцеві стани та простори дій, метрики для оцінки успіху та людська конкуренція різного рівня кваліфікації.

Гра з повною інформацією відповідає такій, в якій кожен гравець має повне знання про поточний ігровий стан. Як приклади, можна навести шахи, шашки та нарди. У кожній з цих ігор обидва гравці знають стан гри, а саме положення всіх фігур, що залишилися на дошці. З іншого боку, у іграх з неповною інформацією деяка інформація прихована від підмножини гравців. Прикладами недосконалих інформаційних ігор є Скрабл та Покер. У Скрабл плитки з літерами кожного гравця приховані від опонентів. Подібним чином, у покері приватні карти гравців приховані від інших гравців.

Кожна дія у детермінованій грі призводить до постійного та передбачуваного результату. Наприклад, спроба перемістити фішку шашки на вільний квадрат по правій верхній діагоналі завжди призводить до абсолютно однакового результату: фігура успішно переміщується на цей квадрат. Шашки та шахи – все це приклади детермінованих ігор. Однак стохастичні (або недетерміновані) ігри включають елемент випадковості. Приклади стохастичних ігор та їх джерело випадковості включають нарди (кидки кісток), скрабл (розіграші плиток) та покер (роздача карт).

Протягом останніх років комп’ютерні ігрові програми змогли конкурувати і перевершувати фахівців світового класу у вищезгаданих іграх:

Шашки. Проектом Chinook керував Джонатан Шеффер з Університету Альберти. У 1994 році Chinook був оголошений чемпіоном світу з шашок після того, як захисник чемпіона, доктор Маріон Тінслі, був позбавлений права через проблеми зі здоров'ям [2]. Chinook захистив свій титул у 1995 році. Команда Chinook оголосила у 2007 році, після майже двох десятиліть обчислень, що вони розв'язали Шашки [3]. Якщо припустити, що обидва гравці грають оптимально, гра завжди закінчиться внічию.

Шахи. Проектом Deep Blue керував Фен-сян Хсу з IBM. У 1997 році Deep Blue переміг чемпіона світу з шахів Гаррі Каспарова в поєдинку з двома перемогами, однією поразкою та трьома нічиїми [4].

Скрабл. Програму MAVEN написав Брайан Шеппард з IBM. У 1998 році комерційна версія MAVEN перемогла чемпіона світу з Скрабл Джоела Шермана та другого, що посів друге місце, Метта Грема із загальним рахунком 6-3 [5].

Нарди. Програму TD-Gammon розробив Джеральд Тесауро з IBM. Завдяки самостійній грі та використанню штучних нейронних мереж та вивченню часових різниць, вона досягла гри на рівні майстра, конкуруючи з одними з найкращих гравців у світі [6].

Heads-up Limit Texas Hold’em Poker. Програма Polaris була розроблена дослідницькою групою Computer Poker (CPRG) в Університеті Альберти [7]. У 2008 році Polaris перемогла деяких найкращих гравців у покер у світі. Остаточним рахунком за серією матчів було три перемоги, дві поразки та одна нічия для Polaris [8].

У цій роботі я хочу представити деякі методи створення стратегій для безлімітного Техаського холдему з двома гравцями, імплементувати один з них, та порівняти його якість з одним з топових світових онлайн ботів Slumbot. Очікується що методи, використані в цьому дослідженні, зможуть бути

застосовані також до багатокористувацького Покеру та інших його модифікацій.

1.2 Суть гри

Покер – сімейство карткових ігор, головними особливостями якої є торги між гравцями і визначення переможця як гравця з найсильнішою комбінацією карт [9]. У кожного гравця є стек, звідки гравець бере фішки для здійснення ставок. Розмір стека залежить від конкретного виду гри. Існує спеціальна фішка дилера, яка переходить від гравця до гравця за годинниковою стрілкою. Будемо називати дилером гравця, у якого є така фішка. Якщо в грі присутній круп'є, то тільки він займається роздачею карт, в іншому випадку право роздавати карти належить дилеру. Кожна карта має масть і порядок, може бути розданою у відкритому або в закритому вигляді, може бути загальною (належить всім гравцям) або приватною (належить конкретному гравцеві). Після закінчення кожної роздачі карти ретельно перемішуються. Один або кілька гравців зобов'язані робити ставки до початку роздачі карт. Обов'язкові ставки діляться на анте і блайнди. Анте – ставка невеликого розміру, яку всі гравці повинні заплатити перед початком роздачі карт. Блайнди бувають великими і малими. Великий блайнд повинен поставити гравець, наступний за годинниковою стрілкою після дилера, а малий блайнд – гравець, наступний за годинниковою стрілкою після гравця, який заплатив великий блайнд. У різних видах покеру обов'язковими ставками можуть бути тільки анте, тільки блайнди або анте і блайнди разом. Після цих ставок слідують раунди торгів, між якими певним чином змінюються карти гравців: до вже існуючих додаються нові карти або старі карти замінюються новими. Під час торгів гравець може відмовитися від подальшої гри, тоді його карти в закритому чи відкритому вигляді (за бажанням гравця, але частіше в

закритому вигляді) скидуються у відбій. В кінці кожного раунду торгів ставки збираються в загальний банк. Якщо в кінці останнього раунду торгів в грі залишається більше одного гравця, то відбувається розкриття карт гравців, що залишилися в грі. Переможцем вважається той гравець, хто має найсильнішу (сила комбінацій різна в різних видах покеру) комбінацію з п'яти карт. Переможець забирає загальний банк і додає його до свого стеку. Мета гри – виграти якомога більше фішок.

1.2.1 Техаський холдем

Техаський холдем – найпопулярніший різновид покеру. Існує велика кількість чемпіонатів по техаському холдему, найголовніший з них – світова серія покеру (World Series of Poker) с призовим фондом понад 225 млн. доларів [9]. Колода в техаському холдемі складається з 52 карт:

- 4 масті: піки (s), черви (h), бубни (d), хрести (c);
- 13 різних достоїнств: туз (A), король (K), дама (Q), валет (J), десятка (T), дев'ятка (9), вісімка (8), сімка (7), шістка (6), п'ятірка (5), четвірка (4), трійка (3), двійка (2).

Старшинство карт впорядковано за спаданням, однак туз може мати як найбільшу силу, так і найменшу. Обов'язковими ставками є великий блайнд (BB) і малий блайнд (MB), MB = BB * 0.5, їхні розміри повинні бути встановлені до початку гри в покер. Після того як блайнди поставлені, гравцям видається дві карти в закритому вигляді.

1.2.2 Раунди торгів

Існує 4 раунди торгів: префлоп, флоп, тьорн і рівер.

- Префлоп – після того, як блайнди поставлені, гравець, що сидить зліва від гравця, який заплатив великий блайнд, починає торги. Після закінчення торгів додатково відкривається три загальні карти в відкритому вигляді.
- Флоп – торги починає гравець, найближчий зліва від гравця, ставлячи великий блайнд. Після закінчення торгівлі додатково відкривається одна загальна карта в відкритому вигляді.
- Тьорн – торги починає гравець, найближчий зліва від гравця, ставлячи великий блайнд. Після закінчення торгівлі додатково відкривається одна загальна карта в відкритому вигляді.
- Рівер – торги починає гравець, найближчий зліва від гравця, ставлячи великий блайнд. Після закінчення торгівлі гравці відкривають свої карти. Перемагає той гравець, чия комбінація з двох його приватних карт і п'яти загальних карт є найсильнішою.

Право ходу передається за годинниковою стрілкою між гравцями, які беруть участь в поточній роздачі. Можливі дії гравця під час торгів:

- Бет (bet) – зробити ставку.
- Колл (call) – зрівняти ставку, тобто поставити стільки ж, скільки поставив опонент або опоненти.
- Рейз (raise) – підвищити ставку (поставити більше попереднього гравця/гравців).
- Фолд (fold) – скинути карти в пас і відмовитися від подальшої участі в роздачі.
- Чек (check) або пропустити – дія доступна гравцеві, коли він вже зробив ставку і в ситуаціях, коли ніхто перед ним не підвищив. Ця дія має на

увазі, що ви не додаєте фішки в банк. На префлопі є обов'язкові ставки (блайнди), які необхідно зрівнювати, щоб залишитися у грі, на інших же раундах, якщо всі гравці зіграють в чек, то у вас буде можливість відкрити наступну карту столу безкоштовно.

– Перейз, 3-бет (reraise, 3-bet) – підвищення ставки після рейзу.

Раунд торгівлі закінчується, якщо всі гравці, які беруть участь в роздачі, зробили свій хід як мінімум один раз, і у них зрівнялися ставки.

1.2.3 Різновиди техаського холдема

За способом підвищення ставок:

– Безлімітний – розмір максимальної ставки дорівнює розміру стеку гравця; самий поширений формат у світі.
– Лімітований – максимальний розмір ставки обмежений. Це більш безпечний вид покеру для вашого банкролу, так як ви не ризикуєте втратити відразу весь стек.

За способом гри:

– Турнірний – гравці платять бай-ін (турнірний внесок) і борються за призові місця.
– Кеш-ігри – гра ведеться на готівкові гроші (які спочатку обмінюються на фішки).

1.2.4 Сила карти

Можливі комбінації в порядку спадання, починаючи від самої сильної, показані в таблиці 1.1. Приклади комбінацій показані на рисунку 1.1.

Таблиця 1.1 – Сила карт в покері

Комбінація	Карти
Роял-Флеш	5 старших карт однієї масті
Стріт-Флеш	5 карт однієї масті по порядку
Каре	4 карти одного порядку
Фул-Хаус	3 карти одного порядку и 2 карти другого порядку
Флеш	5 карт однієї масті
Стріт	5 карт по порядку
Трійка	3 карти одного порядку
Дві пари	2 карти одного порядку и 2 карти другого порядку
Пара	2 карти одного порядку
Старша карта	карта найбільшого порядку

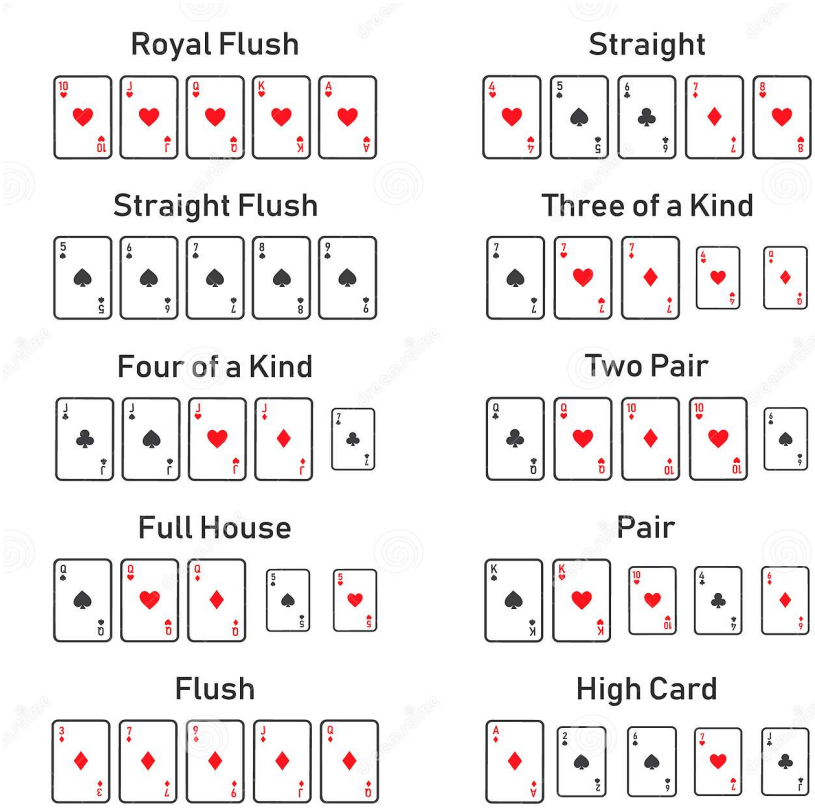

Рисунок 1.1 – Покерні комбінації

Також варто врахувати Кікер (Kicker) – це карта, яка не входить в покерну комбінацію, але визначає переможця в разі, якщо у гравців однакові за силою комбінації. Наприклад, у одного гравця AQ, у іншого A10 і на стіл поклали A-J-8-2-J. У обох гравців комбінація дві пари (AA-JJ), але у першого кікер дама, а у другого 10, тому роздачу виграє перший гравець. Якби на столі замість 8 або 2 поклали б короля, то він був би кікером для обох гравців і тоді б вони розділили би банк.

1.3 Огляд існуючих рішень

Перші серйозні спроби створити покерного бота були зроблені ще на початку 80-х. У 1984 році відомий фахівець з покеру Майк Каро представив програму Orac, яка вразила багатьох своїми можливостями [10]. Orac, наприклад, міг відносно вдало вичислити блеф противника, всього лише вимірявши час, який треба був противнику для ходу – чим довше мислила людина, тим вище була ймовірність блефу.

У 1991 році в Альбертському університеті (Канада) почалася розробка програми Polaris, призначеної для гри в холдем один на один. Після 16 років роботи над проектом, який представляв собою комбінацію декількох покерних ботів, які враховують ціле сімейство алгоритмів пошуку рівноважних стратегій, в 2007 році відбувся матч проти кількох покерних професіоналів. За умовами гри, людині і комп'ютеру роздавали одні і ті ж карти, тому вплив випадковості в грі звели до мінімуму. Спочатку Polaris відчутно тримав перевагу, але після аналізу декількох партій гравці знайшли повторювані особливості гри програми і змогли перемогти [7].

У липні 2008 року Polaris вдалося отримати перемогу на чемпіонаті між людьми і машинами. Загальний рахунок сесій склав 3 перемоги, 2 поразки, 1

нічия. Однак ця перемога не стала початком ери домінування машин і загибелі онлайн-покеру. Як уже згадувалося, «машинний покер» грається при ряді обмежень, які не дотримуються в реальних онлайн-іграх з людьми.

Перший справжній прорив стався в 2015 році, коли в університеті Альберти представили покерного бота Cepheus, заточеного для гри в лімітний Холдем. Математично бот грав практично бездоганно, тому з легкістю обігравав людей, які допускали помилки.

Але створити програму, здатну обіграти профі в безлімітний Холдем, виявилося куди більш складним завданням. За її рішення взялася група вчених з університету Карнегі-Меллона. У тому ж 2015 року вони представили покерного бота Claudico. Перевірити його ефективність зголосилася команда професіоналів, яку очолив відомий гравець хайстейкс Даг Полк. Програма перевершувала попередніх ботів, але результат виявився не на її користь – гравці в результаті здобули перемогу і отримали профіт більше $ 700 тис.

Інша покерна програма Tartanian7 університету Карнегі-Меллона в 2014 році змогла перемогти кількох новачків і комп'ютерних ботів. Таким чином, до недавнього часу штучний інтелект більш-менш впевнено почував себе в лімітних іграх з однією людиною і втрачав хватку в безлімітних і багатокористувацьких покерних іграх.

Після успіхів Tartanian7 в університеті Карнегі-Меллона приступили до створення нового, значно досконалішого покерного бота – Libratus. У той час як програма DeepStack з конкуруючого Альбертського університету показала досить непогані результати в попередніх тестах, Libratus в січні 2017 року виступив проти справжніх покерних професіоналів. Обчислення в процесі розробки Libratus зайняли 15 млн ядро-годин (Claudico знадобилось 2-3 млн) [11]. В процесі гри Libratus використовував потужності суперкомп'ютера Bridges (1.35 петафлоп / сек).

Переможний матч Libratus проти чотирьох покерних професіоналів відбувся 11-30 січня 2017 року рамках змагання "Brains vs. AI " (рис. 1.2).

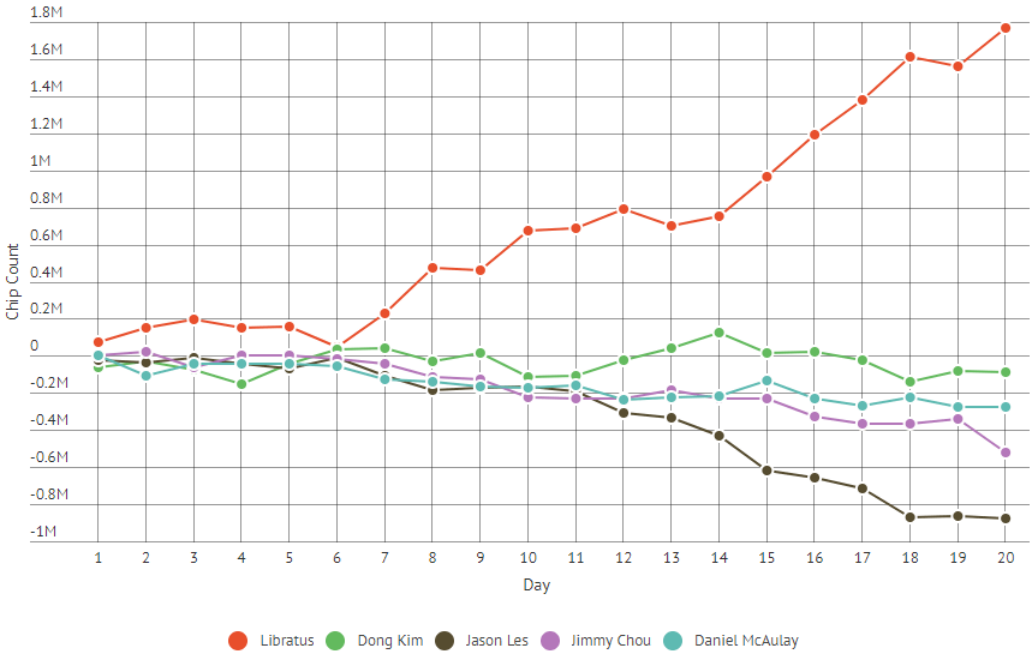

Рисунок 1.2 – Статистика змагання "Brains vs. AI"

ШІ зіграв 120 000 партій і за підсумком вийшов у плюс на $ 1 766 250 умовних доларів. Гра йшла на віртуальні гроші, але приз за перемогу – 200 000 доларів – був абсолютно реальним для чотирьох професійних гравців в покер, двоє з яких вже мали досвід гри проти бота, вигравши в 2015 році у Claudico. Щоб результат змагань не опинився надто випадковим, кожен матч дублювався так, щоб гравець А отримував карти, які отримав комп'ютер в партії з гравцем В і навпаки. Самі гравці виявилися дуже вражені грою програми, яка вміло змінювала свою стратегію кожен день, пристосовуючись до дій гравців.

У 2019 році, бот Pluribus, розроблений в університеті Карнегі – Меллона і Facebook, переміг елітних гравців за столом з шістьма гравцями. Ядро стратегії Pluribus було розраховано шляхом самостійної гри, в якій ШІ грав проти своїх копій, без будь-яких даних про людські або попередні відтворення ШІ, у якості вхідних даних. ШІ спочатку грав навмання, і поступово вдосконалював свою техніку, визначаючи, які дії та який розподіл ймовірності між цими діями призводять до кращих результатів порівняно з попередніми версіями його стратегії. Хоча досить легко побудувати ігри з більш ніж двома

гравцями, в яких широко використовувані алгоритми самогри, на практиці самогра ніколи не показала себе досить добре в деяких іграх з більш ніж двома гравцями [12].

Власна гра Pluribus виробляє стратегію для всієї гри в режимі офлайн, яка називається blueprint стратегією. Потім під час фактичної гри проти суперників, Pluribus вдосконалює стратегію проекту, шукаючи кращу стратегію в реальному часі для ситуацій, в яких він опинився під час гри.

Що цікаво, навчання програми проводилося на одному 64-ядерному сервері з 512 ГБ пам'яті протягом 8 днів і коштувало 150 доларів на хмарному сервісі [12]. Така ефективність різко контрастує з іншими недавніми попередніми проектами, де для навчання були потрібні обчислювальні потужності вартістю в мільйони доларів. Розробники повідомили, що бот не буде доступний у відкритому доступі.

Деякі покерні гравці висловлюють стурбованість, що подібні програми, можуть поставити хрест на звичайних онлайн-грі або як мінімум, сильно ускладнять гру, змусивши кожного гравця включати веб-камеру і транслювати свої дії в мережу. Але як відомо з минулого, програми не знищили шахи, а шахові турніри з величезними грошовими фондами все ще користуються популярністю. Але навіть якщо онлайн-покер поступово піде в минуле, в кінцевому рахунку все, що пов'язано з обчислювальною потужністю комп'ютерів, піде людству на користь.

1.4 Постановка задачі дослідження

1. Дослідити сучасний стан та особливості методів розв’язування гри.
2. Провести огляд сучасних підходів для побудови моделей для розв’язування гри.
3. Обрати та обґрунтувати вибір методу.

4. Дослідити алгоритми Deepstack, контрфактичної мінімізації шкодувань та Continual re–solving.
5. Згенерувати вхідні дані для експериментів.
6. Виконати експерименти, навчити модель на згенерованих даних та проаналізувати отриманий результат.
7. Розробити концептуальні висновки за результатами наукового дослідження.

Висновки до розділу 1

У даному розділі було розглянуто актуальність розв'язання гри Покер, а саме безлімітного Техаського Холдему. Протягом останніх п'яти років над цією задачею почали активно працювати багато наукових команд, щоб покращити результати попередніх підходів і нарешті розв'язати цю гру за допомогою ШІ, адже судячи по прогресу, залишилось зовсім трошки. Кожного року з'являються нові статті на цю тему щодо модифікацій існуючих методів та алгоритмів. Я також вирішила долучитись до цього процесу та спробувати реалізувати деякі ідеї у своїй роботі.

Також було приділено увагу основним правилам гри у Техаський безлімітний холдем, описала процес гри – 4 раунди, комбінації та силу карт, а також існуючі версії гри.

В кінці розділу було описано ряд спроб реалізації рішень за допомогою штучного інтелекту, починаючи ще з 80-х років, та в якому стані на даний момент знаходиться ця задача.

РОЗДІЛ 2 ТЕОРЕТИЧНЕ ПІДҐРУНТЯ МЕТОДІВ ГРИ В ПОКЕР

2.1 Основні поняття теорії ігор

Теорія ігор – це математична теорія для вирішення спірних ситуацій в житі або на ринку. Мета теорії ігор – це знаходження найліпшого для кожного гравця результату, при увазі на інших учасників гри [13].

Гра – це ідеалізована математична модель колективної поведінки кількох осіб (гравців), інтереси яких різні, що і породжує конфлікт. Конфлікт не обов'язково передбачає наявність антагоністичних протиріч сторін, але завжди пов'язаний з певного роду розбіжностями. Конфліктна ситуація буде антагоністичною, якщо збільшення виграшу однієї із сторін на деяку величину приводить до зменшення виграшу іншого боку на таку ж величину і навпаки. Антагонізм інтересів породжує конфлікт, а збіг інтересів зводить гру до координації дій (кооперації). Прикладами конфліктної ситуації є ситуації, що складаються у взаєминах покупця і продавця; в умовах конкуренції різних фірм; в ході бойових дій та інше.

У більшості ігор, що виникають з аналізу фінансово-економічних, управлінських ситуацій, інтереси гравців (сторін) не є строго антагоністичними ні абсолютно збігаються. Покупець і продавець погоджуються, що в їхніх спільних інтересах домовитися про купівлю-продаж, проте вони енергійно торгуються при виборі конкретної ціни в межах взаємної вигідності.

Від реального конфлікту гра відрізняється тим, що ведеться за певними правилами. Ці правила встановлюють послідовність ходів, обсяг інформації кожної сторони про поведінку іншої і результат гри в залежності від ситуації, що склалася. Правилами встановлюються також кінець гри, коли деяка послідовність ходів вже зроблена, і більше ходів робити не дозволяється.

2.1.1 Класифікація ігор

Кооперативні або некооперативні

Гра називається кооперативною, якщо гравці можуть об'єднуватися в групи, взявши на себе деякі зобов'язання перед іншими гравцями і координуючи свої дії [14]. Цим вона відрізняється від некооперативних ігор, в яких кожен зобов'язаний грати за себе. Некооперативні ігри описують ситуації в найменших подробицях і видають більш точні результати. Кооперативні розглядають процес гри в цілому. Гібридні ігри включають елементи кооперативних та некооперативних ігор. Наприклад, гравці можуть створювати групи, але гра буде проводитись в некооперативному стилі. Це означає, що кожен гравець буде переслідувати інтереси своєї групи, разом з тим досягти особистої вигоди.

Симетрична та антисиметрична гра

Гра буде симетричною тоді, коли відповідні стратегії у гравців будуть рівними, тобто вони матимуть однакові платежі. Інакше кажучи, якщо гравці поміняються місцями і при цьому їх виграші за ті ж самі ходи не зміняться.

З нульовою і ненульовою сумою

Ігри з нульовою сумою — це особливий різновид ігор з постійною сумою, тобто таких, де гравці не можуть збільшити або зменшити ресурси або фонд гри, що в них є. Прикладом є гра покер, де один виграє всі ставки інших. В іграх з ненульовою сумою виграш якогось гравця не обов'язково означає програш іншого, і навпаки. Результат такої гри може бути як менше, так і більше нуля.

Паралельні та послідовні

В паралельних іграх гравці ходять одночасно, або вони не знають про ходи інших гравців, поки всі не зроблять свій хід. В послідовних іграх гравці можуть робити ходи в напередодні визначеному порядку, але при цьому вони отримують деяку інформацію про ходи інших. Ця інформація може бути неповною, наприклад, гравець може дізнатися, що його опонент із десяти стратегій точно не вибрав п'яту, нічого не знаючи про інших.

З повною або неповною інформацією

В грі з повною інформацією гравці знають всі ходи, зроблені до поточного моменту, а також можливі стратегії противників, що дозволяє їм деякою мірою передбачити подальший плин гри. Більшість ігор, які вивчає математика, є іграми з неповною інформацією.

Ігри з нескінченним числом ходів

Ігри в реальному світі або ті, що вивчаються економікою, як правило, тривають в скінченну кількість ходів. Математика не так обмежена, зокрема, в теорії множин розглядаються ігри, які можуть продовжуватись нескінченно довго. При чому переможець та його виграш не визначені до завершення всіх ходів. Задача, яка зазвичай ставиться в цьому випадку, полягає не в пошуці оптимального рішення, а в пошуці хоча б виграшної стратегії. Використовуючи аксіому вибору, можна довести, що інколи навіть для ігор з повною інформацією і двома результатами — виграв або не виграв — жоден з гравців не має такої стратегії. Існування виграшних стратегій для деяких особливо сконструйованих ігор має важливу роль в дескриптивній теорії множин.

Дискретні і неперервні ігри

Більшість ігор — дискретні: в них скінчена кількість гравців, ходів, подій, результатів і т. д. Проте ці компоненти можуть бути розширеними на

множину дійсних чисел. Такі ігри часто називаються диференціальними. Вони пов'язані з прямою дійсних чисел, хоча події, що відбуваються, можуть бути дискретними по своїй природі.

2.1.2 Розширена форма гри

Гра з розширеною формою – це представлення гри у формі орієнтованого дерева (його, зазвичай, називають ігровим деревом) [15]. Вершини дерева є станами (позиціями), в яких може перебувати гра, ребра – ходи , які можуть використовувати гравці. Передбачається, що в кожній позиції може здійснювати хід не більше одного гравця. Виокремлюють три види позицій у грі:

- початкова, що є корнем дерева (вершиною, яка не має вхідних ребер);
- проміжні , що мають вхідні та вихідні ребра;
- термінальні , що мають лише вхідні ребра.

Початкова та проміжні позиції утворюють множину нетермінальних позицій.

На даний момент, опис ігор з розширеною формою є доречним для детермінованих ігор, таких як шахи та шашки. Однак, для ігор, що включають випадковість, цього опису недостатньо. У цих іграх зручно представляти випадкові події (такі як кидок кубиків у нардах або роздача карт в Покері) як вершину вибору. Направлені ребра, що виходять з таких вершин, представляють собою можливі результати вибору. Наприклад, у грі, де гравець підкидає звичайний кубик, можливими результатами будуть цифри від 1 до 6, кожна з ймовірністю 1/6.

У іграх з неповною інформацією часто буває кілька ігрових станів, які гравець не зможе відрізнити, оскільки супротивники мають приховану

інформацію (наприклад, карти на руках в Покері). Сукупність усіх невідмінних станів гри називається *інформаційною множиною* для цього гравця. Кожна вершина вибору, у грі з неповною інформацією та розширеною формою, відповідає певній інформаційній множині. У такій грі, як Покер, ігрових станів набагато більше, аніж інформаційних множин.

Гра з розширеною формою та неповною інформацією складається з наступних компонентів:

- N – кількість гравців.
- H – множина можливих послідовностей дій у грі (в т.ч. пуста множина). Підмножини дій, що слідують перед іншими діями називаються префіксами. $A(h) = \{a : (h, a) \in H\}$ – дії, що доступні після нетермінальної дії або послідовності дій. Після фінальних (термінальних) дій не може бути подальших дій.
- P – функція, яка присвоює кожній нетермінальній послідовності дій $(H\backslash Z)$ номер гравця $N \cup \{c\}$. P – функція гравця, $P(h)$ – це гравець, що виконує дію після h. Якщо $P(h) = c$, тоді випадковим чином визначається дія, що буде виконана після h.
- функція f_c – присвоює кожній історії дій h для якої $P(h) = c$ міру ймовірності $f_c(\cdot|h)$ на $A(h)$. $f_c(a|h)$ – це ймовірність, що a виникне маючи h, де кожна ймовірнісна міра незалежна одна від одної.
- для кожного гравця $i \in N$ підрозділ $\hat{I}_i$ з $\{h \in H : P(h) = i\}$ з властивістю $A(h) = A(h')$, коли h та h' знаходяться в одному підрозділі. Для $I_i \in \hat{I}_i$ позначимо через $A(I_i)$ множину $A(h)$ і через $P(I_i)$ гравця $P(h)$ для всіх $h \in I_i$. $\hat{I}_i$ це інформаційний partition (розбиття) гравця i, множина $I_i \in \hat{I}_i$ є інформаційною множиною гравця i.
- для кожного гравця $i \in N$ функція корисності $u_i : Z \to R$, де Z – кінцевий стан. Якщо $N = \{1, 2\}$ і $u_1 = -u_2$, це гра з нульовою сумою. Визначимо $\Delta u_{i,u} = \max_z u_i(z) - \min_z u_i(z)$ – range функцій корисності для гравця i.

2.1.3 Стратегія

Одним з основних понять теорії ігор є поняття стратегії. Стратегією гравця називається сукупність правил, що визначають вибір варіанта дій при кожному особистому ході в залежності від ситуації, що склалася в процесі гри [15]. У простих (одноходових) іграх, коли в кожній партії гравець може зробити лише по одному ходу, поняття стратегії і можливого варіанту дій збігаються. В цьому випадку сукупність стратегій гравця охоплює всі можливі його дії, а будь-яке можливе для гравця і дію є його стратегією. У складних (багатоходових іграх) поняття "варіант можливих дій" і "стратегія" можуть відрізнятися один від одного.

Стратегія гравця називається оптимальною, якщо вона забезпечує даному гравцю при багаторазовому повторенні гри максимально можливий середній виграш або мінімально можливий середній програш, незалежно від того, які стратегії застосовує противник. Можуть бути використані й інші критерії оптимальності. У Покері це відповідає стратегії, яка виграє найбільше грошей у всіх інших гравців, враховуючи, що всі їхні стратегії були розкриті і залишаються статичними (тобто вони не змінюються).

Можливо, що стратегія, що забезпечує максимальний виграш, не володіє іншим важливим поданням оптимальності, як стійкістю (рівноважного) рішення. Рішення гри є стійким (рівноважним), якщо відповідні цьому рішенню стратегії утворюють ситуацію, яку жоден з гравців не зацікавлений змінити.

Повторимо, що завдання теорії ігор – знаходження оптимальних стратегій.

Стратегія і–го гравця у грі з розширеною формою – це функція, яка визначає розподіл по $A(I_i)$ кожному $I_i \in \hat{I}_i$, а Σ_i – набір стратегій і–го гравця.

Комбінація стратегій σ складається із стратегії для кожного гравця, $\sigma_1, \sigma_2, \ldots$, причому σ_{-i} посилається на всі стратегії в σ, окрім σ_i.

Нехай $\pi^\sigma(h)$ – ймовірність історії h, якщо гравці обирають дії відповідно до σ. Ми можемо розкласти $\pi_i^\sigma(h) = \Pi_{i \in N \cup \{c\}} \pi_i^\sigma(h)$ на внесок кожного гравця у цю ймовірність. Отже, $\pi_i^\sigma(h)$ – це ймовірність того, що якщо гравець і грає відповідно до σ, то для всіх історій h', які є належним префіксом h з $P(h') = i$, гравець i робить відповідні дії в h. Нехай $\pi_{-i}^\sigma(h)$ є добутком внеску всіх гравців (включаючи випадковість), крім гравця i. Для $I \subseteq H$ визначте $\pi^\sigma(I) = \Sigma_{h \in I} \pi^\sigma(h)$ як ймовірність досягнення певного набору інформації, заданого σ, з $\pi_i^\sigma(I)$ та $\pi_{-i}^\sigma(h)$, визначеними аналогічно.

Тоді загальним значенням для гравця і профілю стратегії є очікувана виплата результуючого кінцевого вузла, $u_i(\sigma) = \Sigma_{h \in Z} u_i(h) \pi_i^\sigma(h)$.

2.1.4 Рівновага Неша

Неформально, набір стратегій називають рівновагою Неша, якщо жоден гравець не може здобути перевагу, односторонньо змінюючи свою стратегію. Тобто, якщо кожен гравець має статичні стратегії всіх інших гравців і все ще не може отримати виграш, змінивши свою стратегію, тоді всі вони грають в рівновагу Неша. Зверніть увагу, що кожен гравець у рівновазі Неша є найкращою відповіддю на набір усіх інших гравців. Більш формально, набір стратегій для n гравців $\sigma = (\sigma_1, \ldots, \sigma_n)$ є рівновагою Неша, якщо він задовольняє наступним обмеженням:

$$u_1(\sigma) \geq max_{\sigma' 1 \in \Sigma 1} u_1(\sigma'_1, \ldots, \sigma_n)$$

$$u_2(\sigma) \geq max_{\sigma' 2 \in \Sigma 2} u_1(\sigma_1, \sigma'_2, \ldots, \sigma_n)$$

....

$$u_n(\sigma) \geq max_{\sigma' n \in \Sigma n} u_1(\sigma_1, \ldots, \sigma'_n)$$

Таким чином, ми бачимо, що в рівновазі Неша не існує стратегії для гравця i в Σ_i, яка давала б вищу корисність проти σ_{-i}, ніж його стратегія, σ_i, в σ. Фактично, у грі з нульовою сумою для двох гравців кожен гравець, який грає стратегію рівноваги Неша, має однакове значення, незалежно від того, яку стратегію рівноваги Неша грає кожен гравець. Іншими словами, немає переваги відігравати стратегію з однієї рівноваги Неша над стратегією з іншої рівноваги Неша. Однак, на жаль, це не стосується багатокористувацьких ігор. Виграш кожного гравця в одній багатокористувацькій рівновазі Неша може відрізнятися від виграшу відповідних гравців в іншій багатокористувацькій рівновазі Неша [16].

Обчислення точної рівноваги Неша для такої великої гри, як Покер (або навіть абстракції) неможливо. Таким чином, ми часто обмежуємося, але все ще цікавимося пошуком наближень до рівноваг Неша.

Рівновага Неша – це стратегічний профіль, при якому жоден гравець не може збільшити свою корисність більш ніж шляхом односторонньої зміни своєї стратегії. Більш формально, стратегічний профіль для n гравців є рівновагою Неша, якщо він задовольняє наступним обмеженням:

$$u_1(\sigma) + e \geq max_{\sigma' 1 \in \Sigma 1} u_1(\sigma'_1, \ldots, \sigma_n)$$

$$u_2(\sigma) + e \geq max_{\sigma' 2 \in \Sigma 2} u_1(\sigma_1, \sigma'_2, \ldots, \sigma_n)$$

....

$$u_n(\sigma) + e \geq max_{\sigma' n \in \Sigma n} u_1(\sigma_1, \ldots, \sigma'_n)$$

2.1.5 Поняття Regret (шкодування)

Розберемо поняття regret та його застосування на прикладі гри “Камінь-ножиці-папір.” (КНП).

По-перше, слід формалізувати тип цієї гри у рамках термінології теорії ігор. Це некооперативна гра в нормальній формі для двох гравців з нульовою сумою [17]. Тобто:

1. Гравці одночасно та незалежно один від одного вибирають свої стратегії. Вектор стратегій *s = (s1, s2)* гравців являє собою ситуацію в грі.

2. Кожний гравець отримує виграш, який визначається значенням функції $H_i(s)$, на цьому взаємодія між ними припиняється.

Гру в нормальній формі часто називають "грою в один постріл", оскільки під час гри кожен гравець робить лише один дію. Такі ігри можна представити у вигляді n-мірної таблиці, де рядки та стовпці відповідають діям гравців. Кожному результату гри відповідає своя клітка таблиці; в цій клітинці зберігаються відповідні виграші учасників.. Таблиця виплат для КНП має такий вигляд (табл. 2.1, кожен запис має вигляд *(u1, u2)*). За домовленістю, рядок це гравець 1, а стовпчик – гравець 2.

Таблиця 2.1 – Платіжна матриця гри “Камінь-ножиці-папір”

		Гравець 2		
		камінь	ножиці	папір
Гравець 1	камінь	(0,0)	(-1,1)	(1,-1)
	ножиці	(1,-1)	(0,0)	(-1,-1)
	папір	(-1,1)	(1,-1)	(0,0)

Припустимо, ми граємо в КНП за грошову винагороду. Кожен гравець кладе на стіл долар. Якщо є переможець, переможець бере зі столу обидва долари. В іншому випадку гравці зберігають свої долари. Припустимо, що ми обираємо камінь, тоді як наш суперник вибирає папір і виграє, в результаті чого ми втрачаємо свій долар. Нехай наша корисність буде нашим чистим прибутком / збитками в доларах. Тоді наша корисність для цієї партії гри була

-1. Корисність у ситуаціях, якби ми вибрали папір чи ножиці проти паперу суперника, становила б 0 та +1 відповідно.

Ми шкодуємо, що не вибрали папір, але шкодуємо ще більше, що не обрали ножиці, оскільки наш відносний виграш був би ще більшим у ретроспективі. Тут ми визначаємо шкоду, що не вибрали дію, як різницю між корисністю цієї дії та корисністю дій, яку ми насправді вибрали, стосовно фіксованого вибору інших гравців.

Для множини дій $a \in A$ нехай s_i – дія гравця i, а s_{-i} – дії всіх інших гравців. Крім того, нехай $u(s'_i, s_{-i})$ є корисністю дії із заміщенням s'_i на s_i, тобто корисністю, якщо гравець $_i$ грав на s'_i на місці s_i. Далі, після гри, гравець і шкодує, що не зіграв s'_i, це $u(s'_i, s_{-i}) - u(a)$. Зверніть увагу, що це 0, коли $s'_i = s_i$.

У цьому прикладі ми шкодуємо, що не обрали папір: u(папір, папір) −u (камінь, папір) = 0 – (- 1) = 1, і ми шкодуємо, що обрали ножиці: u(ножиці, папір) – u (камінь , папір) = +1 – (−1) = 2.

Як це може вплинути на майбутню гру? Загалом, можна було б віддати перевагу дії, про яку шкодували найбільше, не вибравши її в минулому, але не хотілося б бути цілком передбачуваним. Одним із способів досягти цього є співставлення шкодувань, коли дії агентів вибираються випадковим чином із розподілом, пропорційним позитивним шкодуванням. Позитивні шкодування вказують на рівень відносних збитків, який зазнав гравець за те, що раніше не обрав дії. У нашому прикладі ми не шкодуємо, що вибрали камінь, але маємо шкодування 1 та 2 за те, що не обрали папір та ножиці відповідно. Співставляючи шкодування, далі ми обираємо свою наступну дію пропорційно позитивним шкодуванням, і таким чином обираємо камінь, папір та ножиці з імовірностями 0, 1/3 та 2/3 відповідно, які є нормалізованими позитивними шкодуваннями, тобто позитивними шкодуваннями, поділеними на всю суму.

Тепер припустимо, що в наступній грі ми випадково обираємо ножиці (з ймовірністю 2/3), поки наш суперник вибирає камінь. У цій грі ми маємо

шкодування рівні 1, 2 та 0 щодо відповідної гри в камінь, папір та ножиці. Додаючи їх до наших попередніх шкодувань, ми маємо накопичувальне шкодування відповідно 1, 3 та 2. Таким чином, відповідність шкодувань для нашої наступної гри дає змішану стратегію (1/6, 3/6, 2/6).

2.1.6 Алгоритм співставлення шкодувань (Regret-Matching)

В ідеалі, з часом ми хотіли б звести до мінімуму наші очікувані шкодування. Однак цієї практики недостатньо, щоб звести до мінімуму наші очікувані жалі. Уявіть собі тепер, що ви суперник, і ви повністю розумієте, який підхід до збору шкодувань застосовується. Тоді ви могли б виконати ті самі обчислення, спостерігати за будь-яким упередженням, яке ми мали б щодо гри, і використовувати це упередження. До того часу, коли ми навчилися шкодувати про цю упередженість, шкода вже була б завдана, і наше нове домінуюче значення шкодувань було б використано таким же чином.

Однак існує обчислювальний контекст, в якому збіг шкодувань можна використовувати для мінімізації очікуваних шкодувань за допомогою самовідтворення. Алгоритм наступний:

- Для кожного гравця ініціалізуйте всі сукупні шкодування рівними 0.
- Виконаємо n ітерацій:
  - Обчислити профіль стратегії співпадіння зі шкодуванням. (Якщо всі шкодування гравця не є позитивними, використовуйте єдину випадкову стратегію.).
  - Додайте профіль стратегії до суми профілю стратегії.
  - Виберіть кожен профіль дії гравця відповідно до профілю стратегії.

  - Обчислення шкодування гравця.
  - Додайте шкодування гравців до сукупних шкодувань гравців.
- Обчисліть середній профіль стратегії, тобто суму профілю стратегії, поділену на кількість ітерацій.

$$R^T{}_i = \frac{1}{T}\, max_{\sigma * i \in \Sigma i} \sum_{t=1}^{T} (ui(\sigma * i, \sigma t - i) - ui(\sigma t))$$

З часом цей процес зійдеться до корельованої рівноваги [18].

Наведемо робочий приклад співставлення шкодувань для обчислення найкращої стратегії в Камінь-Ножиці-Папір (КНП). У КНП співставлення шкодувань у грі з двома гравцями збігається до рівноваги.

Ми почнемо з визначення констант та змінних, які використовуються протягом всього процесу.

```
public static final int ROCK = 0, PAPER = 1, SCISSORS = 2,
NUM_ACTIONS = 3;
public static final Random random = new Random();
double[] regretSum = new double[NUM_ACTIONS],
strategy = new double[NUM_ACTIONS],
strategySum = new double[NUM_ACTIONS],
oppStrategy = { 0.4, 0.3, 0.3 };
```

На початку ми присвоюємо ROCK, PAPER та SCISSORS 0, 1 та 2 відповідно. Такі індекси дій відповідають індексам у будь-якому масиві стратегії та шкодувань довжиною NUM ACTIONS. Ми створюємо генератор випадкових чисел, який використовується для вибору дії із змішаної стратегії. Нарешті, ми виділяємо масиви для збереження наших накопичених шкодувань про дії, стратегії, сформовані за допомогою збігу жалю, і суми всіх таких стратегій.

Співставлення шкодувань підбирає дії пропорційно позитивним шкодуванням про те, що вони не були вибрані у минулому [17]. Щоб обчислити змішану стратегію за допомогою співставлення шкодувань, ми спочатку копіюємо всі позитивні шкодування та підсумовуємо їх. Далі ми робимо другий прохід через записи стратегії. Якщо є хоча б одна дія з позитивним шкодуванням, ми нормалізуємо їй, поділивши на нашу нормовану суму позитивних шкодувань. Нормалізація в цьому контексті означає, що ми гарантуємо, що записи масиву сумарно рівні 1, і таким чином представляємо ймовірності відповідних дій у обчислюваній змішаній стратегії.

```
private double[] getStrategy() {
double normalizingSum = 0;
for (int a = 0; a < NUM_ACTIONS; a++) {
        strategy[a] = regretSum[a] > 0 ? regretSum[a] : 0;
        normalizingSum += strategy[a];
  }
for (int a = 0; a < NUM_ACTIONS; a++) {
     if (normalizingSum > 0)
        strategy[a] /= normalizingSum;
      else
       strategy[a] = 1.0 / NUM_ACTIONS;
       strategySum[a] += strategy[a];
  }
  return strategy;
}
```

Звернемо увагу, що нормалізуюча сума може бути недодатною. У таких випадках ми робимо стратегію рівномірною, надаючи кожній дії рівну ймовірність (`1 / NUM_ACTIONS`).

Після того, як обчислюється кожна ймовірність цієї змішаної стратегії, ми накопичуємо цю ймовірність до суми всіх ймовірностей, обчислених для цієї дії, у всіх навчальних ітераціях. Далі повертаємо значення стратегії.

Враховуючи будь-яку таку стратегію, можна вибрати дію відповідно до таких ймовірностей. Припустимо, у нас є змішана стратегія (.2, .5, .3). Якби поділити числовий проміжок від 0 до 1 у цих пропорціях, поділки потрапили б на .2 та .2 + .5 = .7. Далі генерація випадкового числа в діапазоні [0, 1) потрапляє в один із трьох діапазонів [0, .2), [.2, .7) або [.7, 1), що вказує на ймовірнісний вибір відповідного індексу дії.

Припустимо, що є дії $a_0, ..., a_i, ..., a_n$ із ймовірностями $p_0, ..., p_i, ..., p_n$. Нехай накопичена ймовірність $c_i = \sum_{j=0}^{i} pj$. (Зверніть увагу, що $c_n = 1$, оскільки всі ймовірності сумарно повинні дорівнювати 1). Випадкове число r, рівномірно сформоване в діапазоні (0, 1], вибере дію тоді і тільки тоді, коли для всіх $j < i$, $r \geq c_j$ і $r < c_i$.

Дія легко обчислюється наступним чином. По-перше, генерується випадкове число з плаваючою комою в діапазоні (0, 1], ініціалізується індекс дії a до 0 і ініціалізується сукупна ймовірність до 0. Якби ми досягли останнього індексу дії (NUM ACTIONS – 1), то це обов'язково була б обрана дія, тому, поки індекс дії не є останнім, ми додаємо нову ймовірність до нашої сукупної ймовірності, виходимо з циклу, якщо *r* виявляється меншим за сукупну ймовірність, інакше збільшуємо індекс дії.

```
public int getAction(double[] strategy) {
double r = random.nextDouble();
int a = 0;
double cumulativeProbability = 0;
while (a < NUM_ACTIONS - 1) {
     cumulativeProbability += strategy[a];
     if (r < cumulativeProbability)
     break;
     a++;
}
return a;
}
```

За допомогою попередніх блоків коду тепер ми можемо побудувати наш алгоритм навчання:

```
public void train(int iterations) {
double[] actionUtility = new double[NUM_ACTIONS];
for (int i = 0; i < iterations; i++) {
     ⟨Get regret-matched mixed-strategy actions⟩
     ⟨Compute action utilities⟩
     ⟨Accumulate action regrets⟩
     }
}
```

Для заданої кількості ітерацій ми обчислюємо наші дії зі змішаною стратегією, що відповідають шкодуванням, обчислюємо відповідні виграші дій та накопичуємо шкодування щодо обраної дії гравця.

Для вибору дій, обраних гравцями, ми обчислюємо поточну стратегію, що відповідає шкодуванню, і використовуємо її для вибору дій для кожного гравця. Оскільки стратегії можна змішувати, використання однієї і тієї ж стратегії не означає вибору тієї самої дії.

```
double[] strategy = getStrategy();
int myAction = getAction(strategy);
int otherAction = getAction(oppStrategy);
```

Далі ми обчислюємо виграш кожної можливої дії з точки зору гравця, який грає myAction:

```
actionUtility[otherAction] = 0;
actionUtility[otherAction == NUM_ACTIONS – 1 ? 0 : otherAction
+ 1] = 1;
actionUtility[otherAction == 0 ? NUM_ACTIONS – 1 : otherAction
– 1] = -1;
```

Нарешті, для кожної дії ми обчислюємо шкодування, тобто різницю між очікуваною виграшем дії та виграшем обраної дії, і додаємо до наших накопичувальних шкодувань.

```
for (int a = 0; a < NUM_ACTIONS; a++)
regretSum[a] += actionUtility[a] – actionUtility[myAction];
```

Для кожної окремої ітерації нашого навчання шкодування може бути тимчасово перекошеним таким чином, що прийнятна стратегія в поєднанні має негативну суму шкодувань і ніколи не буде обрана.

Суми шкодувань і, отже, одиночні ітераційні стратегії дуже нестабільні. Що сходиться до стратегії мінімального шкодування, це середня стратегія за всі ітерації. Це обчислюється таким чином, як вище getStrategy, але без необхідності мати справу з негативними шкодуваннями.

```
public double[] getAverageStrategy() {
double[] avgStrategy = new double[NUM_ACTIONS];
double normalizingSum = 0;
for (int a = 0; a < NUM_ACTIONS; a++)
     normalizingSum += strategySum[a];
for (int a = 0; a < NUM_ACTIONS; a++)
      if (normalizingSum > 0)
          avgStrategy[a] = strategySum[a] / normalizingSum;
      else
          avgStrategy[a] = 1.0 / NUM_ACTIONS;
  return avgStrategy;
}
```

Загальне обчислення складається з побудови процесу навчання, його запуску для заданої кількості ітерацій (у цьому випадку 1 000 000) та виводу отриманої середньої стратегії.

```
public static void main(String[] args) {
```

```
        RPSTrainer trainer = new RPSTrainer();
        trainer.train(1000000);

System.out.println(Arrays.toString(trainer.getAvgStrategy()));
    }
```

2.2 Основи навчання з підкріпленням

Навчання з підкріпленням (RL) – це парадигма навчання, де інтелектуальний агент вчиться робити оптимальні рішення, взаємодіючи з невідомим оточенням (рис. 2.1). У порівнянні з навчанням з вчителя, особливий виклик у RL – це вчитися без заздалегідь відомих правильних результаті. Як далі зазначено в роботі, це призведе до алгоритмічних міркувань, які часто є унікальними для RL [19].

Рисунок 2.1 – Взаємодія між агентом RL і зовнішнім середовищем.

Взаємодія агент-середовище часто моделюється як дискретний, марківський процес прийняття рішень, або MDP, описаний п’ятіркою M = (S, A, P, R, γ):

1) S є, можливо, нескінченним набором станів, в яких може перебувати середовище;

2) A – це, можливо, нескінченний набір дій, які агент може прийняти в стані;

3) P (s`| s, a) дає ймовірність переходу середовища в новий стан s` після дії a, що приймається в стані s;

4) R (s, a) – середня винагорода, що негайно отримується агентом після виконання дії a в стані s;

5) $\gamma \in (0, 1)$ – коефіцієнт дисконтування.

Процес може бути записаний як траєкторія (s1, a1, r1, ...), яка генерується на етапі t = 1, 2,. . . наступним чином:

1) агент спостерігає за поточним станом середовища $s` \in S$ і приймає дію при $a_t \in A$;

2) перехід навколишнього середовища до наступного стану s_{t+1}, розподіленого за ймовірністю переходу $P(\cdot \,|s_t, a_t)$;

3) пов'язана з переходом безпосередня винагорода $r \in R$, середнє значення якої R(s, a).

Опускаючи індекси, кожен крок призводить до появи кортежу (s, a, r, s`), що називається переходом. Метою агента RL є максимізація довгострокової винагороди шляхом прийняття оптимальних дій (які будуть визначені найближчим часом). Стратегія його вибору дій, позначена π, може бути детермінованою або стохастичною. У будь-якому випадку використовуємо π (s) для позначення вибору дії, тобто слідуємо π у стані s. З огляду на стратегію π, значення стану s є середньою дисконтованою довгостроковою винагородою від цієї стратегії:

$$V\,\pi(s) \coloneqq \mathbb{E}[r1 + \gamma r2 + \gamma^2 r3 + \cdots | s_1 = s,\, a_i \sim \pi(s_i),\, \forall i \geq 1]$$

Ми зацікавлені в оптимізації стратегії, щоб V^{π} максимізувався для всіх стратегій. Позначимо через π* оптимальну стратегію, а V* відповідне їй значення функції (також відоме як оптимальне значення функції). У багатьох

випадках більш зручно для знаходження значення винагороди використовувати іншу функцію, що називається Q-функцією:

$$Q^{\pi}(s, a) \coloneqq \mathbb{E}[r1 + \gamma r2 + \gamma^{2} r3 + \cdots | s_1 = s, a_1 = a, a_i \sim \pi(s_i), \forall i > 1]$$

яка вимірює середню дисконтовану довгострокову винагороду, вибираючи спочатку a у стані s, а потім наступну стратегію π. Оптимальна Q-функція, що відповідає оптимальній стратегії, позначається $Q*$.

Класичні алгоритми навчання з підкріпленням на кожному етапі оцінюють модель стану майбутнього повністю, привласнюють цінність кожній дії і потім вибирають дію з найвищим значенням цінності. Згодом з'явилися моделі, що використовують нові методи прийняття рішень. Розглянемо наступні сучасні алгоритми, що використовують нейронні мережі:

- Deep Q-learning (глибоке Q-навчання).
- REINFORCE (алгоритм на основі градієнтів за стратегіями).
- A3C (алгоритм асинхронного актора-критика).

2.2.1 Q-навчання

Алгоритми Q-навчання оперують функцією якості $Q(s, a)$, що позначає найбільший можливий рахунок, якого можна досягти в кінці гри після вибору дії a в стані s. Функцію $Q(s, a)$ можна визначити рекурсивно [20]:

$$Q^{*}(s, a) = R(s, a) = \gamma \sum_{s' \in S} T(s, a, s') \cdot max_{a'} Q^{*}(s', a'),$$

де $T(s, a, s')$ – ймовірність переходу в стан s' зі стану s при виборі дії a. Оскільки функції $R(s, a)$ і $T(s, a, s')$ апріорі невідомі, потрібно оцінювати $Q^{*}(s, a)$ і оновлювати оцінку на протязі дослідження гри у відповідності з

різними діями a в різних станах s, враховуючи при цьому нагороду r і наступний стан s'. Правило ітеративного поновлення оцінки Q виглядає наступним чином:

$$Q(s,a) = Q(s,a) + \alpha(r + \gamma \cdot max_{a'} Q(s',a') - Q(s,a)),$$

де γ – коефіцієнт дисконтування; α – швидкість навчання, яка визначає ступінь значущості нових спостережень при оновленні оцінки. При $\alpha = 1$, наприклад, попередні оцінки не враховуються.

На початку навчання оцінка $max_{a'} Q(s',a')$ може бути далекою від істинного значення, але в результаті налаштування $Q(s,a)$ гарантується збіжність до істинної функції [20].

Алгоритм Q-навчання у псевдокоді виглядає наступним чином:

Ініціалізувати $Q(s,a)$ довільним чином
Отримати початковий стан s
Для кожного епізоду:
Вибрати і виконати дію a
Отримати нагороду r і новий стан s'

$$Q(s,a) := Q(s,a) + \alpha(r + \gamma \cdot max_{a'} Q(s',a') - Q(s,a))$$
$$s := s'$$

Коли середовище має дискретний простір станів, а агент дискретну кількість можливих дій, модель динаміки середовища представляє собою однокрокову матрицю переходу:

$$T(s(t+1) \mid s(t), s(a)).$$

Дана стохастична матриця містить всі ймовірності переходу в бажаний стан з поточного стану при виборі кожної дії. У середовищі Atari простір станів після масштабування без втрати інформації є множиною зображень розміром 64 × 64 пікселя, а агент має 18 можливих дій. Тому розмір матриці переходу

обчислюється наступним чином: $|S \times S \times A| \approx |(66.7 \cdot 10^9) \times (66.7 \cdot 10^9) \times 18|$. При 32-бітному розмірі елемента матриці це вимагає 3.4 $\times 10^{14}$ Гб пам'яті. Через неможливість зберігати такий обсяг потрібно виконувати апроксимацію таблиці переходів.

2.2.2 Deep Q-learning (DQN)

Для вирішення описаної вище проблеми можна використовувати нелінійне представлення, яке відображає стан і дію на значення. Конкретний спосіб представлення та спосіб навчання можуть варіюватися, але один з найпопулярніших варіантів – використання глибоких нейронних мереж, особливо при роботі зі спостереженнями, представленими у вигляді екранних зображень [21]. З урахуванням цього, модифікований алгоритм Q-навчання має вигляд:

1. Ініціалізувати $Q(s, a)$ деяким початковим наближенням.
2. Взаємодіючи з оточенням, отримати кортеж $< s, a, r, s' >$.
3. Розрахувати значення втрат:

   $L = (Q(s, a) - r)^2$, якщо епізод завершився або

   $L = (Q(s, a) - (r + \gamma \cdot max_{a'} Q(s', a')))^2$ в іншому випадку.
4. Оновити $Q(s, a)$, використовуючи алгоритм стохастичного градієнтного спуску (SGD), мінімізуючи втрати по відношенню до параметрів моделі.
3. Повторювати з кроку 2 до збіжності.

Цей алгоритм виглядає простим, але, на жаль, не дуже добре працює. Перша проблема полягає у взаємодії агента з оточуючим середовищем. Якщо представлення Q вибрано гарно, то досвід, отриманий від середовища, покаже агенту релевантні дані для навчання. Однак виникають проблеми, коли наближення для Q вибрано не ідеально, як, наприклад, на початку навчання. У

такому випадку агент може застрягти в неправильних діях для деяких станів, навіть не намагаючись вести себе інакше. Це дилема розвідки і розробки (exploration versus exploitation dilemma). З одного боку, агенту необхідно досліджувати середовище, щоб скласти повну картину переходів і результатів дій. З іншого боку, потрібно ефективно використовувати взаємодію з навколишнім середовищем: нам не потрібно витрачати час на те, щоб навмання випробовувати дії, які ми вже випробували і для яких отримали результати.

Можна помітити, що випадкову поведінку краще застосувати на початку навчання, коли наближення Q погане, оскільки це дає нам більш рівномірно розподілену інформацію про стани навколишнього середовища. Після виконання декількох етапів навчання, випадкова поведінка стає неефективною, і повертаються до наближення Q, щоб вирішити як діяти.

Метод, який виконує таке поєднання двох крайніх варіантів поведінки, називається *ε-жадібним*, що означає просто перемикання між випадковою політикою і політикою Q з використанням гіперпараметра ймовірності ε. Змінюючи ε, вибирають співвідношення випадкових дій. Зазвичай починають із значення ε = 1.0, що означає 100% випадкових дій, і поступово зменшують його до деякого невеликого значення, наприклад 5% або 2% випадкових дій. Використання епсілон-жадібного методу допомагає як дослідити навколишнє середовище на початку, так і дотриматися гарної політики наприкінці навчання. Існують також інші рішення проблеми розвідки і розробки, проте ця проблема залишається одним з фундаментальних відкритих питань в RL і активною областю досліджень на сьогоднішній день.

Друга проблема полягає у невиконанні умов, необхідних для використання методу стохастичного градієнтного спуску для оптимізації. Для апроксимації складної нелінійної функції Q(s, a) за допомогою нейронної мережі потрібно обчислити цілі для цієї функції, використовуючи рівняння Беллмана, а потім розглянути задачу як проблему навчання з учителем. Проте, одна з основних вимог для оптимізації за SGD – незалежність навчальних

даних та їх однаковий розподіл. Ці вимоги не задовольняються у задачі, що розв'язується:

1. Приклади не є незалежними, вони дуже близькі один до одного, тому що належать до одного епізоду.

2. Розподіл навчальних даних не буде ідентичним до прикладів, що забезпечуються оптимальною політикою, яку хочемо вивчити. Наявні дані – результат деякої іншої політики (поточної політики, випадкової політики або їх обох в разі епсілон-жадібного методу), проте немає потреби вчитися грати випадковим чином, нам потрібна оптимальна політика з найкращою нагородою.

Для подолання вказаної проблеми зазвичай використовують великий буфер минулого досвіду і вибірки навчальних даних з нього замість використання останнього досвіду. Цей метод називається буфером відтворення (replay buffer). Найпростіша реалізація – це буфер фіксованого розміру з новими даними, що додаються в кінець буфера, щоб витіснити з нього найстаріший досвід. Буфер відтворення дозволяє проводити навчання на більш-менш незалежних даних. Існує також й інший тип буфера відтворення: буфер з пріоритетом, який забезпечує більш складний підхід до вибірки.

Третя проблема з процедурою навчання за замовчуванням пов'язана з кореляцією між кроками. Рівняння Беллмана дозволяє розрахувати значення $Q(s,a)$через $Q(s',a')$, цей процес називається самонастройкою (bootstrapping). Однак обидва стани s і s' мають тільки один крок між собою, що робить їх дуже схожими, і нейронній мережі стає дуже важко їх розрізнити. Під час оновлення параметрів мережі з метою наближення $Q(s,a)$ до бажаного результату, побічно можна змінити значення, створене для $Q(s',a')$ та інших станів поблизу. Це призводить до нестабільності процесу навчання: наприклад, має місце «переслідування власного хвоста», коли виконується оновлення Q для стану s, тоді в наступних станах виявляється, що $Q(s',a')$ стає гіршим, але спроби оновити його можуть зіпсувати наближення $Q(s,a)$ і т. д.

Щоб зробити навчання більш стабільним, існує трюк під назвою «цільова мережа» (target network), за допомогою якого зберігається копія мережі і використовується для знаходження значення Q (s ', a') в рівнянні Беллмана. Ця мережа синхронізується з основною мережею тільки періодично, наприклад, один раз за N кроків, де гіперпараметр N кількості ітерацій навчання зазвичай приймає досить велике значення рівне 10^3 або 10^4.

Підсумувавши наведене вище, алгоритм Deep Q-learning (DQN), описаний в [21], для покращення збіжності застосовує метод збереження в пам'яті попереднього досвіду, при якому в пам'яті зберігаються N попередніх переходів, а при навчанні використовується випадкова вибірка з даних переходів [22]. При виборі дії, в свою чергу, використовується ε-жадібна стратегія, яка полягає в тому, що замість вибору найбільш вигідної дії агент з ймовірністю ε вибирає випадкову дію.

Алгоритм DQN у псевдокоді виглядає наступним чином:

Ініціалізувати пам'ять переходів D
Ініціалізувати модель обчислення $Q(s, a)$ випадковими вагами
Отримати початковий стан s
Для кожного епізоду:
    Вибрати і виконати дію a: {
        з ймовірністю ε вибрати випадкову дію
        інакше вибрати $a = arg\ max_{a'} Q(s, a')$
    }
    Отримати нагороду r і новий стан s'
    Зберегти перехід $< s, a, r, s' >$ в пам'яті D
    Отримати вибірку переходів $< ss, aa, rr, ss' >$ з пам'яті
    Обчислити tt для кожного елемента вибірки {
        якщо ss' – термінальний стан, то $tt = rr$
        інакше $tt = rr + \gamma \cdot max_{a'} Q(ss', aa')$
    }
    Навчити мережу з функцією втрат $(tt - Q(ss, aa))^2$
    $s \coloneqq s'$

2.2.3 Policy gradient

Алгоритми на основі градієнтів за стратегіями (policy gradient) – це тип алгоритмів навчання з підкріпленням, які базуються на оптимізації параметрів стратегії відповідно до очікуваної сумарної нагороди, використовуючи для цього градієнтний спуск [23]. Класичним алгоритмом даного типу вважається REINFORCE. Його особливість в тому, що в процесі оновлення стратегії, тобто коригування ваг моделі, у ньому враховується вся траєкторія агента. Під траєкторією розуміється послідовність всіх переходів – четвірок об'єктів $S \times A \to S \times R$ (стан, дія, новий стан, нагорода) протягом періоду однієї гри. Задача алгоритму – максимізувати очікувану сумарну нагороду:

$$p(s) = E\left[\sum_{t=1}^{\infty} \gamma^{t-1} R_t \mid s \right].$$

Нехай $R(\tau)$ – математичне очікування сумарної нагороди для траєкторії τ. Імовірність кожної траєкторії задається стратегією, яка визначає дії, які робить агент. Отже, формулу можна переписати таким чином:

$$p(s, \theta) = \sum_{\tau} P_\theta(\tau)\, R(\tau)\,,$$

де θ визначає параметри моделі, яка піддається навчанню. Функція $p(s, \theta)$ потребує максимізації. Для цього обчислюється градієнт $p(s, \theta)$ за θ, який можна представити у вигляді:

$$\frac{\partial p(s,\theta)}{\partial \theta} = E_\tau\left[R(\tau)\, \frac{\partial ln ln\,(P_\theta(\tau))}{\partial \theta} \right].$$

Градієнт – це вектор, направлений у бік зростання функції $p(s, \theta)$:

$$\nabla_\theta(P_\theta(\tau)) = \frac{\partial\ ln\ ln\ (P_\theta(\tau))}{\partial\theta}.$$

Далі виконується апроксимація математичного очікування E_τ за допомогою середнього арифметичного за T переходами в межах одного епізоду або гри. Отримаємо правило оновлення вектора коефіцієнтів θ, за допомогою якого максимізується функція сумарної нагороди:

$$\theta_{t+1} = \theta_t + \ \alpha \cdot R(\tau) \cdot \ \nabla_\theta\big(P_\theta(\tau)\big).$$

У псевдокоді алгоритм REINFORCE записується наступним чином:

Вхід: модель, що задає стратегію $\pi(a \mid s, \theta), \forall\ a \in A,\ s \in S.$

Ініціалізувати ваги моделі θ

Повторювати нескінченно:

Згенерувати епізод $S_0, A_0, R_1, \dots, S_{T-1}, A_{T-1}, R_T$, використовуючи стратегію π

Для кожного кроку $\tau = 0, \dots, T-1$:

Обчислити $R(\tau)$

$$\theta := \theta + \ \alpha \cdot R(\tau) \cdot \ \nabla_\theta\big(P_\theta(\tau)\big).$$

2.2.4 REINFORCE

Алгоритм REINFORCE використовує незміщену оцінку градієнта, тому має досить велику дисперсію, що на практиці суттєво сповільнює збіжність.

Основним джерелом великої дисперсії градієнту вважається розмір винагород, які входять до кумулятивної винагороди у виразі оцінки градієнту стратегії. Тому, для зменшення дисперсії, цей множник потрібно зменшувати.

Один із способів зменшення дисперсії градієнту базується на принципі причинності, який полягає в тому, що стратегія в момент часу t' не може впливати на винагороду в час $t < t'$. Оцінка винагороди в цьому випадку приймає вигляд:

$$\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\left[\sum_{t=1}^{T}\nabla_\theta \ ln\ ln \ ln\ ln\, \pi_\theta(s_{i,t}) \ \left(\sum_{t\prime=t}^{T} R(s_{i,t\prime}, a_{i,t\prime})\right)\right]$$

В результаті дисперсія градієнтів на практиці суттєво зменшується, а оцінка залишається незміщеною.

Інший спосіб полягає у зменшенні дисперсії шляхом зменшення множника із винагородами, віднімаючи від нього базову вартість – середню кумулятивну винагороду:

$$\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}[\nabla_\theta \ ln\ ln\ ln \ \pi_\theta(\tau)\,[R(\tau) - b]]$$

$$b = \frac{1}{N}\sum_{i=1}^{N} R(\tau)$$

Оцінка залишається незміщеною у випадку сталого значення b:

$$E[\nabla_\theta \ ln\ ln \ ln\ ln\, \pi_\theta(\tau) b \ \ ] = \int \ \pi_\theta(\tau)\nabla_\theta \ ln\ ln \ ln\ ln\, \pi_\theta(\tau) b d\tau \ =$$

$$= \int \ \nabla_\theta \pi_\theta(\tau) d\tau \ = b\nabla_\theta \int \ \pi_\theta(\tau) d\tau = b\nabla_\theta 1 = 0$$

Алгоритми типу **«актор-критик»** оперують стратегією, яка виступає актором, та функцією корисності, яка виступає критиком, та будують стратегію подібно до алгоритму REINFORCE. Відмінність полягає в тому, що для розрахунку градієнта використовується функція корисності, а не траєкторія. Під час переходу в новий стан система отримує значення нагороди і обчислює функцію корисності за методом часової різниці (temporal difference) за допомогою рівняння Беллмана [24]. Функція корисності потім використовується для коригування ваг моделі.

2.2.5 Алгоритм асинхронного актора-критика

У статті [25], яка описує асинхронні методи для глибокого навчання з підкріпленням, був представлений алгоритм асинхронного актора-критика (Asynchronous Advantage Actor-Critic – A3C), який використовує кілька копій нейронної мережі і кілька примірників середовища, де кожен агент взаємодіє зі своєю власною копією середовища. Результати, отримані декількома агентами, в подальшому об'єднуються в глобальну модель. Ця версія алгоритму використовує функцію переваги A, яка оцінюється, використовуючи дисконтовану суму майбутніх нагород R: $A = R - V(s)$, де $V(s)$ – функція, що позначає найбільший рахунок, якого можна досягти зі стану s, і не залежить від дій агента.

Алгоритм A3C у псевдокоді записується наступним чином:

Задати глобальні загальні вектори параметрів θ і θ_v і глобальний загальний лічильник
$T = 0$
Задати вектори параметрів θ' і θ'_v для кожного потоку
Ініціалізувати лічильник кроків потоку $t := 1$
Повторювати:

Оновити градієнти: $d\theta \coloneqq 0$, $d\theta_v \coloneqq 0$

Синхронізувати: $\theta' \coloneqq \theta$, $\theta'_v \coloneqq \theta_v$

$$t_{start} \coloneqq t$$

Отримати стан s_t

Повторювати:

Виконати дію a_t згідно зі стратегією $\pi(a_t \mid s_t, \theta')$,

Отримати нагороду r_t і новий стан s_{t+1}

$$t \coloneqq t + 1$$
$$T \coloneqq T + 1$$

до тих пір поки s_t термінал або $t - t_{start} == t_{max}$

$R = 0$, для терміналу s_t

$R = V(s_t, \theta'_v)$, для нетермінала s_t

Для $i \in \{t-1, \dots, t_{start}\}$:

$$R \coloneqq r_i + \gamma \cdot R$$
$$d\theta \coloneqq d\theta + \nabla_{\theta'} ln\, ln\, \pi(a_i \mid s_i, \theta')(R - V(s_i, \theta'))$$
$$d\theta_v \coloneqq d\theta_v + \frac{\partial (R - V(s_i, Q'_v))^2}{\partial \theta'_v}$$

Виконати асинхронне оновлення θ і θ_v

поки $T \leq T_{max}$

Висновки до розділу 2

У цьому розділі були описані теоретичні основи, на яких будуються методи, описані в наступному розділі. Було розкриті важливі поняття з теорії ігор, такі як розширена форма гри, через яку буде представлено ігрове дерево рішень та рівновага Неша, до якої ми прагнемо при знаходженні оптимальної стратегії в Покері. Було введено нове поняття – шкодування (англ. Regret) – це числове значення, що описує, наскільки гравець шкодує про прийняте рішення. Шкодування може бути позитивним, негативним або нульовим. Позитивне шкодування вказує на те, що краще було би прийняти інше

рішення. Негативне шкодування свідчить про те, що гравець задоволений своїм рішенням, а нульове – байдуже. На цьому понятті будується алгоритм Regret Matching, за допомогою якого вже можна оптимізувати стратегію у безлічі ігор, в тому числі і у Покері. Для простоти розуміння, у якості прикладу була наведена гра Камінь-Ножиці-Папір.

Далі було представлено один з напрямів машинного навчання – навчання з підкріпленням (Reinforcement Learning). Описані сучасні алгоритми, що використовують нейронні мережі: Deep Q-learning (глибоке Q-навчання), REINFORCE (алгоритм на основі градієнтів за стратегіями), A3C (алгоритм асинхронного актора-критика).

РОЗДІЛ 3 ОПИС МЕТОДІВ ДЛЯ РОЗВ'ЯЗАННЯ ГРИ В ПОКЕР

3.1 Контрфактична мінімізація шкодувань (CFR)

3.1.1 Vanilla CFR

У попередньому розділі ми застосували алгоритм regret matching для гри КНП, щоб обчислити оптимальну стратегію. Це було реалізовано з використанням обчислень шкодувань про дії, які ми вирішили не робити. У БТХ наші дії можуть не відразу призвести до виграшу, тому ми не можемо обчислити шкодування таким чином. Натомість нам доведеться розглянути нове поняття – контрфактичні шкодування.

Контрфактичне мислення – це поняття в психології, що передбачає схильність людини створювати можливі альтернативи життєвим подіям, які вже відбулися; те, що суперечить тому, що насправді сталося [26]. У даному алгоритмі розглянемо, як би виглядали альтернативні сценарії поточного ігрового раунду, якби ми вчинили іншу дію.

CFR складається з наступних основних етапів:

1. Кожна ітерація починається з перемішування та роздачі випадкових карт гравцям.
2. На кожному вузлі прийняття рішення використовується відповідність шкодувань (англ. Regret Matching) для обчислення стратегії. Далі обчислюється очікуваний виграш за цією стратегією.
3. Інформація про виграші передається назад до батьківського вузла.
4. Виграш зважується на основі того, наскільки ймовірним буде досягнення кожного з вузлів (так як ми можемо впливати на нашу гру, але не на гру наших суперників!)

Подібно зворотному розповсюдженню помилки в машинному навчанні, алгоритм виконує 2 проходи по ігровому дереву. Під час проходу вперед

обчислюється ймовірність досягнення кожного вузла дерева, тоді як при зворотному проході обчислюються виграші. Для кожного вузла ігрового дерева, який рекурсивно відвідується під час навчальної ітерації, обчислюється змішана стратегія [26].

Введемо формальні поняття, що використовуються в алгоритмі:

A – множина усіх ігрових дій.

I – набір інформації. Це множина позицій в грі, які неможливо розрізнити між собою для гравця, що здійснює в них хід, у зв'язку з не повною інформацією про дії інших учасників гри.

$A(I)$ – це сукупність усіх валідних дій відносно набору інформації I.

t, T – наші часові кроки. t стосується кожного набору інформації і збільшується з кожним відвідуванням набору інформації.

σ_i^t – стратегія і–го гравця. Усі стратегії гравців разом у момент часу t утворюють профіль стратегії σ^t.

σ_{-I} – профіль стратегії, який виключає стратегію і–го гравця.

$\sigma_{I\to A}$– профіль, еквівалентний σ, за винятком того, що дія a завжди обирається в наборі інформації I.

h – історія, послідовність дій, починаючи з кореня гри.

$\pi(\sigma, h)$ – ймовірність досягнення ігрового вузла h з урахуванням поточних стратегій σ для обох гравців.

$\pi(\sigma, I)$ – ймовірність охоплення набору інформації. Це сума ймовірностей досягнення всіх станів гри, які належать до набору інформації, тобто:

$$\pi^{\sigma}(I) = \sum_{h \in I} \pi(\sigma, h)$$

$\pi_{-i}(\sigma, h)$– ймовірність досягнення інформаційного набору I. Ми обчислюємо це значення, припускаючи, що замість того, щоб грати за звичною

стратегією, гравець завжди буде робити дії, що ведуть до інформаційного набору I, коли це можливо.

Тоді контрфактичне значення (виграш) гри h дорівнює:

$$v_i(\sigma, h) = \sum_z \pi_{-i}(\sigma, h)\pi(\sigma, h, z)u_i(z)$$

У даному рівнянні ми підсумовуємо всі листові термінальні вузли z, яких можна досягти з h, і де $\pi(\sigma, h, z)$ – це ймовірність досягнення певного листового вузла z з h, коли гравці дотримуються стратегії σ.

Контрфактичне значення заміняє виграш при обчисленні контрфактичних шкодувань у послідовних іграх, таких як покер [17]. Таким чином, ми можемо представити контрфактичне шкодування про те, що не виконали дію у стані гри h як:

$$r(h, a) = v_i(\sigma_{I \to a}, h) - v_i(\sigma, h)$$

Це означає, що ми порівнюємо значення постійної дії a в ігровому стані h зі значенням поточного ігрового стану, враховуючи нашу поточну стратегію. Контрфактичне шкодування для інформаційного набору I – це лише сума над контрфактичними шкодуваннями для всіх станів гри, що належать до інформаційного набору. Накопичувальне контрфактичне шкодування – це сума контрфактичних шкодувань за всіма навчальними ітераціями.

Оновлення стратегії для набору інформації розраховується за формулою:

$$\sigma_i^{T+1}(I, a) = \begin{cases} \dfrac{R^{T,+}(I, a)}{\sum_{a \in A(I)} R^{T,+}(I, a)}, & if \sum_{a \in A(I)} R^{T,+}(I, a) > 0 \\ \dfrac{1}{|A(I)|} & otherwise \end{cases}$$

Ми обираємо дії, пропорційні позитивним шкодуванням, тому на етапі використовуються лише позитивні накопичені шкодування:

$$R^{T,+}(I,a) = \max\left(0, \sum_{t=1}^{T} r^t(I,a)\right)$$

|A(I)| це кількість можливих дій, які може здійснити поточний гравець, тому, ми отримуємо рівномірний розподіл ймовірності за діями, коли не існує жодного позитивного накопиченого шкодування. Це завжди трапляється під час першого відвідування набору інформації.

Обговоримо способи «очищення» наших результатів таким чином, щоб вони, в більшості випадків, більш точно наближались до бажаної рівноваги. На початку кожна дія має однакову ймовірність, тому в середній стратегії завжди буде певна ймовірність дії. Може знадобитися деякий час, поки накопичувальне контрфактичне шкодування не зійдеться, тому ми часто маємо ненульову ймовірність у всіх діях, навіть у абсолютно для нас невигідних.

Одним із способів “чистки” стратегії є встановлення порогу ймовірності дії (наприклад, 0,001) та обнулення ймовірностей дії, яка має ймовірність нижче цього порогу. Після такого порогового значення потрібно перенормувати всі ймовірності дій стратегії. У гіршому випадку це очищення та порогове значення може позбавити гарантії сходження до рівноваги, але за певних умов виграш при відтворенні очищеної стратегії проти опонента може бути таким же. На практиці було показано, що даний трюк може підвищити ігрові показники в покері [17].

Іншим способом кращого сходження стратегії рівноваги є просто не включати стратегії ранніх ітерацій. Наприклад, можна обнулити всі суми стратегії до нуля до заданого числа ітерацій.

3.1.2 CFR+

CFR+ є модифікацією вищеописаного CFR, проте з невеликими змінами. По-перше, після кожної ітерації значення негативних шкодувань обнуляються. Формально у CFR+ стратегія на ітерації $T + 1$ обчислюється згідно з Regret Matching+ (RM+), який є ідентичним до (формула), але використовує шкодування $Q^T(I,a) = \max\{0, Q^{T-1}(I,a) + r^t(I,a)\}$ замість $R_+^T(I,a)$. По-друге, CFR+ використовує "лінійне" усереднення стратегії, при якому вклад ітерації t в усереднену стратегію пропорційний t, на противагу рівномірно зваженій стратегії, як було у CFR. За рахунок цих змін, CFR+ у більшості випадків збігається набагато швидше, ніж CFR [27].

3.1.3 Discounted CFR

У попередніх варіантах CFR на кожній ітерації шкодування має рівнозначну вагу. Наступна модифікація полягає у тому, щоб зменшити вплив ранніх ітерацій під час обчислення шкодувань, за рахунок присвоєння меншої ваги попереднім ітераціям.

Розглянемо просту ситуацію, коли агент має обрати одну з трьох дій. Виграш кожної дії становить 0, 1 та -1 000 000 відповідно. З (формула) ми бачимо, що CFR і CFR+ присвоюють однакову ймовірність кожній дії на першій ітерації. Це призводить до шкодувань, що дорівнюють 333 333, 333 334 та 0 відповідно. Якщо ми продовжуватимемо слідувати CFR, ймовірності обрати першу та другу дії на наступній ітерації будуть дорівнювати 50%, а шкодування будуть оновлені приблизно до 333 332,5 та 333 334,5 відповідно. Знадобиться 471 407 ітерацій, щоб агент обрав найкращу дію, тобто другу, зі 100% ймовірністю. Зменшення впливу першої ітерації з часом значно

пришвидшило би збіжність у цьому випадку. У цьому прикладі неоптимальна дія була обрана на першій ітерації, але загалом такі дії можуть обиратись протягом усього циклу, а зменшення ваги може бути корисним не тільки протягом перших ітерацій.

Існує безліч схем дисконтування, які сходяться в теорії, проте на практиці не всі з них добре працюють. Розглянемо ті варіанти, які добре показали себе з практичної точки зору. Перший алгоритм має назву лінійний CFR (LCFR). Він є ідентичним CFR, за винятком того, що на ітерації t шкодування та усереднена стратегія оновлюються з вагою t. Тобто ітерації мають лінійне зважування. Це означає, що після T ітерацій LCFR, шкодування з першої ітерації матиме вагу лише $\frac{2}{T^2+T}$ замість $\frac{1}{T}$, як було б у випадку з CFR та CFR+. У вищенаведеному прикладі з агентом та трьома діями, LCFR обере другу дію із 100% ймовірністю після 970 ітерацій, тоді як CFR+ знадобиться 471 407 ітерацій.

Оскільки модифікації, які застосовані у LCFR та CFR+, не суперечать один одному, має сенс об'єднати їх в єдиний алгоритм, де кожна ітерація t зважується відповідно її номеру, а також має обнуляються негативні шкодування. Однак провівши емпіричні дослідження виявилось, що цей новий алгоритм, під назвою LCFR+, призводить до результатів, гірших, ніж LCFR і CFR+ поокремо [28].

Тим не менше, виявляється, що використання менш агресивної схеми дисконтування призводить до стабільно високих показників. Розглянемо алгоритм, який будемо називати дисконтованими CFR з параметрами α, β та γ – DCFR(α, β, γ). У цьому алгоритмі кумулятивні позитивні шкодування будуть домножуватись на $\frac{t^\alpha}{t^\alpha+1}$, кумулятивні негативні шкодування на $\frac{t^\beta}{t^\beta+1}$, а внесок до середньої стратегії на $(\frac{t}{t+1})^\gamma$, де t – номер ітерації. У цьому випадку LCFR є еквівалентним до DCFR(1,1,1), оскільки множення шкодування на ітерації t та внесок у середню стратегію на $\frac{t'}{t'+1}$ на кожній ітерації $t \leq t' \leq T$

еквівалентно зважуванню ітерації t на $\frac{1}{T}$. CFR+ еквівалентний DCFR(∞, −∞, 2). В цілому, оптимальний вибір α, β та γ залежить від конкретної гри. Однак експериментально було виявлено, що задання параметрам значень α = 3/2, β = 0 та γ = 2 призвело до стабільно сильніших результатів, ніж CFR+ [29]. Таким чином, за замовчуванням будемо розглядати DCFR саме з вищеперечисленими параметрами.

3.1.4 MCCFR

Суть даної модифікації алгоритму полягає в тому, щоб уникнути обхід всього ігрового дерева на кожній ітерації [30]. Для цього, ми обмежимо розгляд термінальних вузлів дерева, які відвідуються на кожній ітерації. Нехай $Q = \{Q_1, \dots, Q_r\}$ - множина підмножин Z, така, що їх об'єднання покриває множину Z. Ми будемо називати одну із цих підмножин блоком. На кожній ітерації ми будемо вибирати один із цих блоків і розглядати термінальні історії гри лише у цьому блоці. Нехай $q_j > 0$ - ймовірність розгляду блоку Q_j для поточної ітерації (де $\sum_{j=1}^{r} q_j = 1$).

Нехай $q(z) = \sum_{j:z\in Q_j} q_j$, де $q(z)$ - це ймовірність врахування термінальної історії z на поточній ітерації. Тоді вибіркове контрфактичне значення значення при оновленні блоку j дорівнює:

$$\tilde{v}_i(\sigma, I|j) = \sum_{z\in Q_j \cap Z_I} \frac{1}{q(z)} u_i(z)\pi_{-i}^{\sigma}(z[I])\pi^{\sigma}(z[I], z)$$

Вибір блоку Q разом із вибірковими ймовірностями визначає суть sample-based CFR алгоритму. Замість того, щоб робити повний обхід дерева

ігор, алгоритм обирає один з цих блоків, і далі вивчає термінальні історії лише у цьому блоці.

Припустимо, що ми обрали $Q = \{Z\}$, тобто блок, який містить усі термінальні історії, і $q_1 = 1$. У цьому випадку вибіркове контрфактичне значення дорівнює звичайному контрфактичному значенню, і ми отримуємо класичний варіант CFR алгоритму.

Припустимо, що замість цього ми обираємо блок для включення всіх термінальних історій з однаковою послідовністю вузлів, де ймовірність випадкового результату не залежить від дій гравців. Тоді q_j є добутком ймовірностей у такій послідовності, і ми отримуємо chance-sampled CFR. [31].

Опишемо отриманий алгоритм MCCFR. Спочтаку обирається блок і для кожного набору інформації, що містить префікс термінальної історії у блоці, обчислюємо вибіркові контрфактичні шкодування по кожній дії. Ці шкодування накопичуються і використовуються при обчисленні стратегії гравця на наступній ітерації у regret-matching алгоритмі.

Опишемо двох представників MCCFR алгоритму.

Outcome-Sampling MCCFR

У Outcome-Sampling MCCFR алгоритмі Q обирається так, щоб кожен блок містив одну термінальну історію, тобто $\forall Q \in \mathcal{Q}, |Q| = 1$. На кожній ітерації обирається єдина термінальна історія дій і оновлюється набір інформації тільки по цій історії. Ймовірність вибірки, q_j, визначає розподіл по термінальним історіям. Позначимо цей розподіл, використовуючи профіль вибірки σ', $q(z) = \pi^{\sigma'}(z)$. Алгоритм працює, обираючи випадковим чином z з використанням профілю стратегій σ', зберігаючи $\pi^{\sigma'}(z)$. З обраною історією робиться прохід вперед (щоб обчислити ймовірність досягнення кожного префіксу історії для гравця $\pi_i^{\sigma}(h)$) і назад (щоб обчислити ймовірність зіграти необрані дії в історії, $\pi_i^{\sigma}(h, z)$). Під час зворотного проходу обчислюються (і додаються до загальних шкодувань) вибіркові контрфактичні шкодування по кожному відвіданому набору інформації [32].

$$\tilde{r}(I,a) = \begin{cases} w_I * \big(1 - \sigma(a|z[I])\big) \text{ якщо } (z[I]a) \sqsubseteq z \\ -w_I * \sigma(a|z[I]) \text{ інакше} \end{cases}, \text{ де } w_I = \frac{u_i(z)\pi_{-i}^{\sigma}(z)\pi_i^{\sigma}(z[I]a,z)}{\pi^{\sigma'}(z)}$$

Однією з переваг Outcome-Sampling MCCFR є те, що термінальна історія відбирається відповідно до стратегії опонента, $\sigma'_{-i} = \sigma_{-i}$, і далі оновлення не потребує знати про σ_{-i}. w_I стає $u_i(z)\pi_i^{\sigma}(z[I],z)/\pi_i^{\sigma'}(z)$. Таким чином, Outcome-Sampling MCCFR може бути використаний для мінімізації шкодувань.

External-Sampling MCCFR

У External-Sampling MCCFR обираються тільки дії противника і значення у вузлі шансу (ті варіанти, на які гравець не може повпливати). Маємо блок $Q_\tau \in Q$ для кожної чистої стратегії опонента і вузла шансу, тобто для кожного детермінованого відображення τ з $I \in \mathcal{I}_c \cup \mathcal{I}_{N\setminus\{i\}}$ в $A(I)$. Ймовірності для блоків присвоюються на основі розподілів f_c і σ_{-i}, тому $q_\tau = \prod_{I\in\mathcal{I}_c} f_c(\tau(I)|I) \prod_{I\in\mathcal{I}_{N\setminus\{i\}}} \sigma_{-i}(\tau(I)|I)$. Тоді блок Q_τ містить всі термінальні історії дій z відповідно до τ, тобто якщо ha є префіксом z з $h \in I$ для деякого $I \in \mathcal{I}_{-i}$, тоді $\tau(I) = a$. На практиці ми не будемо відбирати приклади τ, а будемо відбирати окремі дії, які становлять τ тільки в разі потреби. Ключовим моментом є те, що в результаті цих блокових ймовірностей $q(z) = \pi_{-i}^{\sigma}(z)$. [32]. Алгоритм робить ітерації протягом $i \in N$ і для кожного наступного обходу дерева гри в глибину, відбирає дії при кожній історії h, де $P(h) \neq i$ (зберігаючи ці вибірки таким чином, щоб однакові дії обирались при всіх h в одному і тому ж наборі інформації). Завдяки запам'ятовуванню при обході історія з одного набору інформації буде відвідана лише один раз. Для кожного такого відвідуваного набору інформації обчислюються (і додаються до сумарних шкодувань) вибіркові контрфактичні шкодування.

$$\tilde{r}(I,a) = (1 - \sigma(a|z[I])) \sum_{z \in Q \cap Z_I} u_i(z)\pi_i^{\sigma}(z[I]a, z)$$

Зазначимо, що сума може бути легко обчислена під час проходу, завжди підтримуючи зважену суму виграшів всіх термінальних історій, що мають свої корені в поточній історії.

3.2 Методи для декомпозиції гри

3.2.1 Continual re-solving

Continual re-solving – це новий алгоритм, призначений для розв’язування ігор з неповною інформацією. Замість того, щоб заздалегідь створювати стратегію для всієї гри, що раніше було необхідно, continual re-solving дозволяє обчислити стратегію для будь-якої конкретної ситуації в грі під час її виникнення [33]. Для цього зберігається компактна статистика раніше обчислених стратегій. Розглянемо даний алгоритм на прикладі Покеру.

Введемо поняття діапазон карт, який представляє собою розподіл ймовірностей по можливим рукам гравця за умови досягнення публічного стану. Кожен гравець має свій, унікальний діапазон карт, так як одна пара карт не може бути одночасно у декількох гравців. Знаючи такий діапазон і вектор контрфактичних значень, досягнутих противником при попередньому рішенні для кожної руки, можна реконструювати стратегію розв'язання піддерева без необхідності знову розв'язувати гру повністю [33]. Кожне значення у векторі противника є контрфактичним значенням, умовним значенням "що–якщо", яке дає очікуване значення, якщо противник досягає публічного стану з певною рукою. Алгоритм CFR використовує контрфактичні значення, і застосувавши його для розв'язання, можна легко обчислити вектор контрфактичних значень опонента в будь–якому публічному стані [30].

Алгоритм Continual re–solving (послідовне перерозв’язування) складається з наступних основних етапів:

1. На початку гри діапазон карт має рівномірний розподіл, і контрфактичні значення противника ініціалізуються значеннями, що мають кожна з можливих пар приватних карт.

2. Коли настає наша черга діяти, ми перерозв'язуємо піддерево в поточному публічному стані, використовуючи збережений діапазон, а також значення супротивника, і діємо відповідно до обчисленої заново стратегії.

3. Після кожної дії гравця, або випадкової роздачі карт, ми оновлюємо наш діапазон і контрфактичні значення противника відповідно до наступних правил:

(1) власна дія: замінюємо контрфактичні значення противника на значення, обчислені в перерозв’язаній стратегії для обраної дії. Оновлюємо наш власний діапазон, використовуючи обчислену стратегію і правило Байеса.

(2) випадкова дія: замінюємо контрфактичні значення противника на ті, які були обчислені для цієї випадкової дії з останнього перерозв’язування. Оновлюємо наш власний діапазон, обнуляємо варіанти приватних пар карт в тому діапазоні, який неможливий при наявності нових відкритих карт.

(3) дія противника: не потрібно вносити ніяких змін в наш діапазон або в значення противника.

Ці оновлення гарантують, що контрфактичні значення противника задовольняють достатнім умовам, і вся процедура апроксимує рівновагу Неша [34]. Зауважимо, що послідовне перерозв’язування не відстежує діапазон противника, а тільки слідкує за його контрфактичними значеннями. Більш того, для поновлення цих значень не потрібно знати дії противника.

3.2.1 Monte Carlo Continual re-solving

Monte Carlo Continual Resolving – це модифікація CR, яка використовує Outcome Sampling MCCFR для розв’язування ігор. Розглянемо основні зміни, які відрізняють даний алгоритм від класичного.

По-перше, для зменшення використання пам'яті побудова ігрового дерева відбувається поступово, подібно до пошуку Монте-Карло по дереву (MCTS) [35]. Спочатку будується дерево, яке містить лише корінь. Якщо був досягнутий набір інформації, який відсутній у пам'яті, він додається. В збережених наборах інформації оновлюються лише шкодування.

Оскільки для обчислення точних контрфактичних значень усередненої стратегії необхідно виконати обхід всього ігрового дерева, доведеться замінити їх приближеною оцінкою. З цією метою MCCFR додатково обчислює вибіркові контрфактичні значення опонента

$$\tilde{v}_i^{\sigma^t}(I) = \frac{1}{\pi^{\sigma^t}(z)} \pi_{-i}^{\sigma^t}(h)\pi^{\sigma^t}(z|h)u_i(z)$$

Неможливо обчислити точне контрфактичне значення усередненої стратегії , знаючи лише значення поточних стратегій. Після проходження T ітерацій стандартним способом оцінки контрфактичних значень $\bar{\sigma}^T$ є використання середніх арифметичних значень:

$$\tilde{v}(I) = \frac{1}{T}\sum \tilde{v}_i^{\sigma^t}(I)$$

Однак ми спостерігали кращі результати із зваженими середніми значеннями

$$\tilde{v}(H) = \frac{\sum_t \tilde{\pi}^{\sigma^t}(h)\, v_2^{\sigma^t}(h)}{\sum_t \tilde{\pi}^{\sigma^t}(h)}$$

Для підвищення ефективності вирішення методом MCCFR використовується спеціальна схема вибірки [30]. По-перше, з імовірністю 90% обирається історія, яка належить до поточного набору інформації I. Це дозволяє зосередитись на найбільш релевантній частині гри. По-друге, щоразу, коли відвідується MCCFR, ми відбираємо обидві дії. Це підвищує прозорість алгоритму, оскільки всі ітерації виконують подібний обсяг роботи.

Для підвищення ефективності вирішення методом MCCFR використовується спеціальна схема вибірки. По-перше, з імовірністю 90% обирається історія, яка належить до поточного набору інформації I. Це дозволяє зосередитись на найбільш релевантній частині гри. По-друге, щоразу, коли $h \in S$ відвідується MCCFR, ми відбираємо обидві дії. Це підвищує прозорість алгоритму, оскільки всі ітерації виконують подібний обсяг роботи.

І під час попереднього розвязування гри, і в режимі реального часу, MCCFR працює на невеликих підмножинах ігрового дерева. Зокрема, не потрібно починати кожне розв’язування з нуля, адже використовуються попередні обчислення. Для цього зберігаються значення шкодування, середня стратегія та відповідні оцінки значень з попереднього перерозв’язування [30].

3.3 Deepstack

DeepStack – це алгоритм загального призначення для великого класу послідовних ігор з неповною інформацією [34, 36]. Розглянемо, як він може бути застосований у грі БТХ.

Стан гри в покер описується персональною інформацією гравців – двома закритими картами, і публічною інформацією (публічним станом) – п'ятьма картами, що лежать на ігровому столі, а також послідовністю дій, що виконують гравці. Можливі послідовності публічних станів в грі утворюють публічне дерево, кожен публічний стан якого має пов'язане з ним публічне піддерево (рис. 3.1).

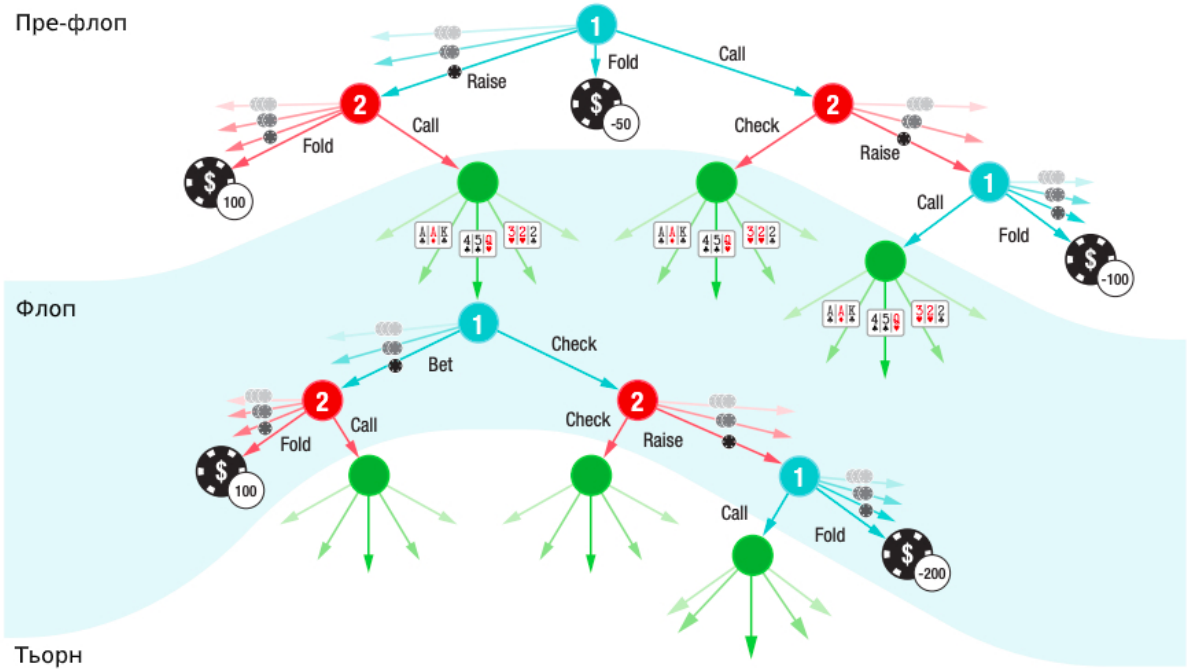

Рисунок 3.1 – Частина публічного дерева БТХ.

Вузли дерева – це публічні стани, а ребра – можливі дії в даному стані. Червоний і блакитний кольори відповідають за першого та другого гравця у грі відповідно. Зеленим кольором представлені стани, де випадковим чином роздаються публічні карти. Гра закінчується в термінальних вузлах, показаних у вигляді фішок зі значеннями очікуваних виграшів. Для термінальних вузлів, де жоден гравець не спасував свої карти, значення стану (виграш) отримує гравець, чиї приватні карти в сукупності з публічними картами склали сильнішу покерну комбінацію.

Стратегія гравця визначає розподіл ймовірностей по можливих діях для кожного стану рішення, де стан рішення – це комбінація публічного стану і приватних карт поточного гравця. З огляду на стратегію гравця, для будь–

якого публічного стану можна обчислити діапазон гравця, який представляє собою розподіл ймовірностей по можливим рукам гравця за умови досягнення публічного стану.

Виграш для конкретного гравця в термінальному публічному стані є білінійною функцією діапазонів обох гравців, що використовує матрицю виплат, яка визначається правилами гри. Очікуваний виграш для гравця в будь–якому іншому публічному стані, включаючи початковий стан, – це очікуваний виграш для досяжних термінальних станів з урахуванням фіксованих стратегій гравців. Оптимальною стратегією називається стратегія, яка при багатократному повторенні гри забезпечує гравцю максимально можливий середній виграш. В іграх з нульовою сумою для двох гравців, таких як БТХ, стратегія рівноваги Неша максимізує очікуваний виграш у грі, де опонент має оптимальну стратегію [16].

Експлуатабельність стратегії – це різниця в очікуваному виграші проти оптимальної стратегії опонента і очікуваному виграші при рівновазі Неша [37]. Це міра того, наскільки ми далеко знаходимось від рівноваги Неша. Алгоритм DeepStack прагне обчислити стратегію з низькою експлуататебльністю, тобто знайти рівновагу Неша. Він обчислює цю стратегію під час гри тільки для тих станів публічного дерева, які фактично виникають.

3.3.1 Структура DeepStack

Алгоритм DeepStack (Рис. 3.2) складається з трьох основних етапів:

1) обчислення локальної стратегії для поточного публічного стану;
2) обмежений по глибині прохід дерева з використанням навченої функції значень;
3) перегляд обмеженого набору дій.

DeepStack будує ігрове дерево, використовуючи дії: fold, call, один або декілька варіантів ставок та all-in. Це дозволяє DeepStack приймати рішення зі швидкістю, близькою до людської.

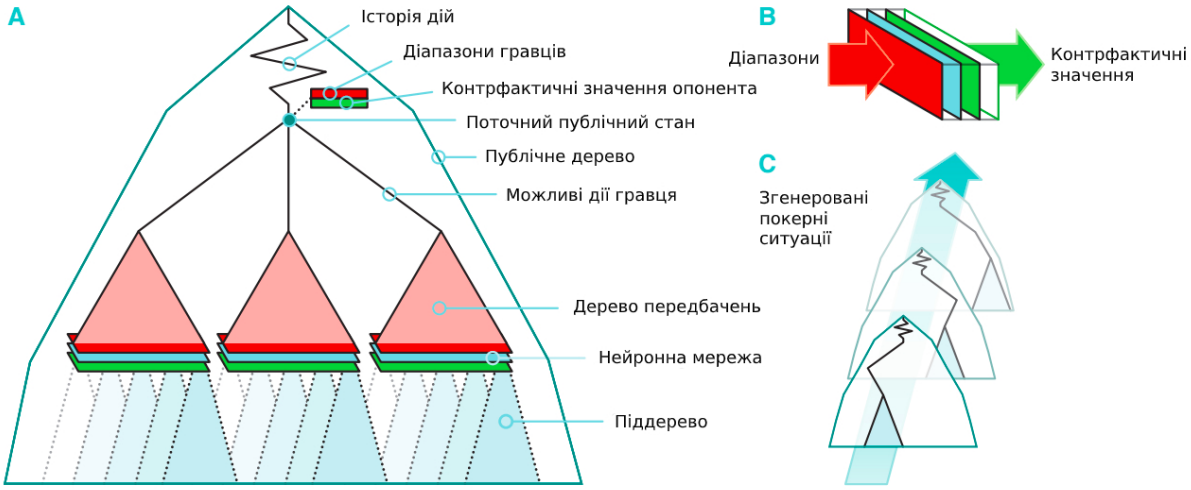

Рисунок 3.2 – Структура Deepstack

DeepStack рахує ймовірності дій для всіх карт, які можуть бути в публічному стані в дереві (Рис. 3.2, A). Під час гри зберігається інформація про два вектора – діапазони контрфактичних значень обох гравців, які оновлюються в процесі гри. Щоб обчислити ймовірності дій, виконується переоцінка, використовуючи ці діапазони. Для того, щоб спростити перерахунок ймовірності дій, дії гравців обмежуються лише доступними, а перегляд дерева відбувається тільки до кінця раунду. Під час переоцінки, контрфактичні значення для публічних станів за межами раунду апроксимуються за допомогою навченої функції оцінки. Функція оцінки представлена нейронною мережею. Ця нейронна мережа тренується перед грою, генеруючи випадкові покерні ситуації (розмір банку, публічні карти і діапазони гравців) і знаходячи для них оптимальні стратегії.

Алгоритм DeepStack побудований на основі евристичного пошуку, завдяки якому ШІ став успішним в іграх з повною інформацією [30]. Серцевиною евристичних методів пошуку є ідея "безперервного пошуку", при якому локальна стратегія пошуку викликається кожен раз, коли агент повинен діяти, не зберігаючи пам’яті про те, як і чому він діяв, щоб досягти поточного

стану. В основі DeepStack лежить безперервне повторне розв'язування (continual re–solving), обчислення локальної стратегії, яка вимагає лише мінімальної пам'яті про те, як і чому діяв гравець, щоб досягти поточного публічного стану [34].

Постійне перерозв’язування є теоретично обгрунтованим, але саме по собі непрактичне, так як покерне дерево гри надзвичайно велике і потребує багато часу на розв'язання. Щоб зробити постійне перерозв’язування практичним, необхідно обмежити глибину і ширину піддерева.

В іграх з неповною інформацією не можна замінити піддерево евристичним або попередньо розрахованим значенням [33]. DeepStack долає цю проблему, замінюючи піддерева за межами певної глибини на функцію контрфактичних значень, яка апроксимує результуючі значення. Вхідними даними для цієї функції є діапазони для обох гравців, розмір банку і публічні карти, яких достатньо для визначення публічного стану. Виходи – це вектор для кожного гравця, який містить контрфактичні значення утримання кожної руки в даній ситуації. Іншими словами, сам по собі вхід є описом покерної гри: розподіл ймовірностей роздачі окремих приватних карт, ставки на гру і будь–які відкриті карти; вихід – це оцінка того, наскільки цінним було б володіння певними картами в такій грі. Функція вартості – це свого роду інтуїція, швидка оцінка цінності перебування у довільній покерній ситуації. При обмеженні глибини чотирьох дій цей підхід зменшує розмір гри для повторного рішення з 10^{160} варіантів прийняття рішення на початку гри не більше ніж до 10^{17} способів [34]. DeepStack використовує глибинну нейронну мережу в якості навчальної функції, яка буде описана нижче.

3.3.2 Глибинні нейронні мережі контрфактичних значень

Глибинні нейронні мережі зарекомендували себе як потужні моделі і відповідають за основні досягнення в області розпізнавання образів і мови, автоматичної генерації музики і розв'язування ігор [38]. DeepStack використовує глибинні нейронні мережі як функцію оцінки для обмежених ігрових дерев (рис. 3.3). У алгоритмі використовуються три окремі мережі: одна оцінює контрфактичні значення після роздачі перших трьох публічних карт (флоп етап), інша – після роздачі четвертої публічної карти (тьорн етап), остання – коли всі публічні карти на столі (рівер). Допоміжна мережа для оцінки значень перед роздачею будь–якої публічної карти використовується для прискорення повторного рішення на початковому етапі гри.

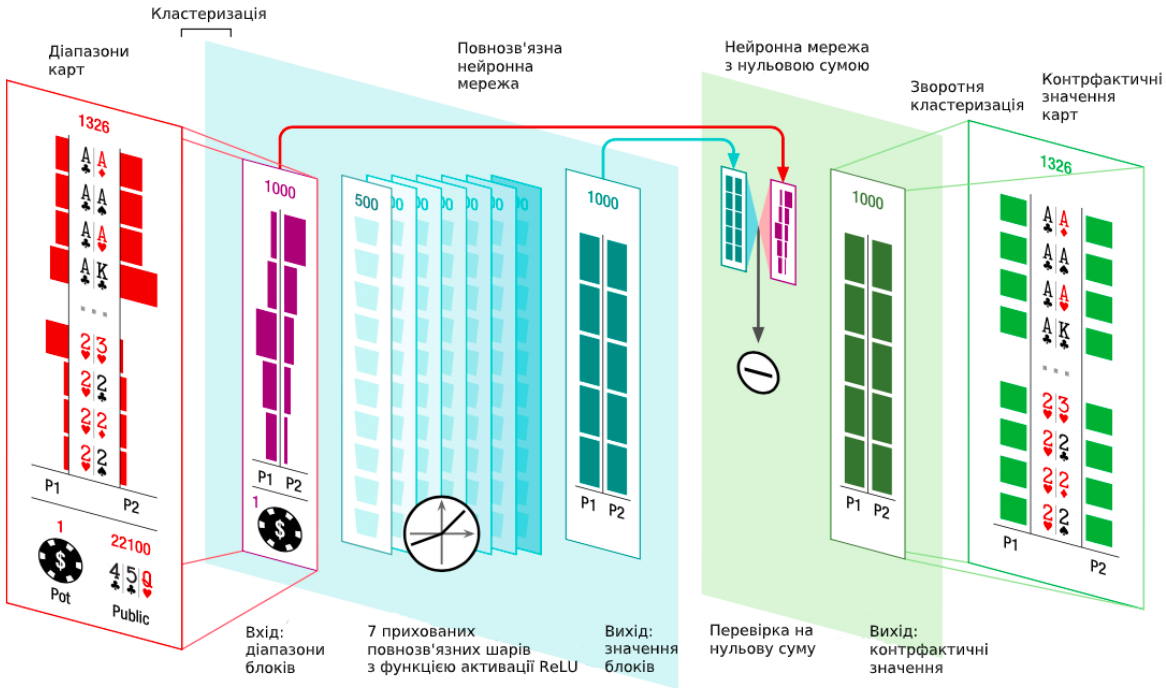

Рисунок 3.3 – Архітектура нейронної мережі Deepstack

DeepStack використовує нейронну мережу прямого розповсюдження з 7 повнозв’язними прихованими шарами, кожен з яких має 500 нейронів, і функцію активації ReLU [39]. Ця архітектура вбудована в зовнішню мережу,

яка перевіряє контрфактичні значення на нульову суму. Дана мережа може бути навчена градієнтним спуском. Вхідними даними мережі є розмір банку як частка від загальної кількості фішок гравців, діапазони і набір публічних карт. Відбувається стиснення діапазонів у 1000 кластерів, як в традиційних методах абстракції [40], і вводиться вектор ймовірностей кластерів. Вихід мережі – це вектори контрфактичних значень для кожного гравця і руки.

Висновки до розділу 3

В цьому розділі були досліджені найновіші методи, які використовуються при розв'язуванні великих ігор з неповною інформацією, в тому числі і покерної гри. Одним із найкращих та найпопулярніших, на сьогоднішній день, методів є CFR – контрфактична мінімізація шкодувань. Окрім класичної версії цього методу, були розглянуті деякі модифікації останніх років, а саме CFR+, Discounted CFR та Monte Carlo CFR. У даних модифікаціях вдається значно швидше досягнути рівноваги Неша. Саме тому одна із них – CFR+, буде використана далі у роботі.

Далі були описані методи для декомпозиції ігор великого розміру, що дозволяють зменшити час виконання та використання пам’яті при розв'язуванні – continual re-solving та його модифікація Monte Carlo continual re-solving. Використання подібних методів однозначно необхідно в Покері, адже розв'язати повну гру з кореня дерева до кінця практично неможливо, занадто багато ресурсів на це потрібно.

Наприкінці розділу був розглянутий алгоритм DeepStack. Його особливість полягає в тому, що він не обчислює і не зберігає повну стратегію до початку гри. Замість цього, він аналізує кожну конкретну ситуацію, яка виникає під час гри, але не ізольовано. Він дозволяє уникнути міркувань про подальшу гру, замінюючи приблизною оцінкою обчислення за межами певної

глибини ігрового дерева. Використовуючи даний алгоритм, ми зможемо реалізувати покерного бота, щоб ефективно розв'язувати покерні партії.

РОЗДІЛ 4 РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ РОЗВ'ЯЗАННЯ ПОКЕРНОЇ ГРИ

4.1 Постановка задачі

Основним завданням даного розділу є розробка бота, який навчається грати у покер БТХ. Для розв'язання даної задачі було вирішено застосувати алгоритм Deepstack, описаний у розділі 3.3, що будується на основі рекурсивного мислення – кожен стан гри розглядається як похідний від попередніх. В основі такого мислення лежить контрфактична мінімізація шкодувань, що була згадана в розділі 3.1. Було вирішено взяти модифікацію алгоритму CFR+ для прискорення пошуку рівноваги Неша. Крім того, для апроксимації оцінок станів у ігровому дереві та зменшення часу розв'язання гри були застосовані глибинні нейронні мережі.

4.2 Обґрунтування вибору платформи та мови реалізації

Для розробки покерного бота була обрана мова програмування Lua та бібліотека для ефективних математичних розрахунків та глибинного навчання Torch. Lua – швидка і компактна скриптова мова програмування. Lua комбінує простий процедурний синтаксис з потужними можливостями опису даних через використання асоціативних масивів і розширюваної семантики мови [41]. Разом з бібліотекою Torch, яка ефективно зберігає дані у вигляді тензорів та підтримує розпаралелювання обчислень засобами CUDA, бот отримає значну перевагу у швидкості.

Так як Torch бібліотека підтримується тільки на Unix-подібних системах, нашого бота може запускати на операційних системах Linux та Mac OS.

4.3 Аналіз архітектури

Програмний продукт можна представити у вигляді наступних модулів:

Tree – побудова ігрового дерева, що представляє всю гру або її частину.

Lookahead – структура, яка використовує ігрове дерево для розв'язання гри. Lookahead ефективно зберігає дані про контрфактичні значення, шкодування, стратегії на кожному вузлі дерева за допомогою torch тензорів.

Resolving – continual re-solving алгоритм

DataGeneration – розв'язання випадково згенерованих покерних ситуацій за допомогою Lookahead та Resolving, зберігання результатів цих ігор у файли.

Training – тренування нейронної мережі на основі даних, згенерованих у DataGeneration модулі.

Nn – реалізація глибинних нейронних мереж контрфактичних значень

Game – обробка приватних, публічних карт та їх можливих комбінацій

TerminalEquity – оцінювання термінальних станів гри.

Player – модуль, який реалізує клієнтську сторону програми (підключення до сервера та отримання/відправлення повідомлень про гру)

Data – модуль для зберігання даних, таких як навчені нейронні мережі, та згенеровані ігри

4.4 Деталі реалізації

Для розробленого бота було згенеровано 1 500 000 покерних ситуацій для рівер етапу і 1 000 000 для трьорну та флопу. У цих іграх використовувались випадково створені діапазони, публічні карти та випадковий розмір банку. Цільові контрфактичні значення для кожної тренувальної гри були сформовані шляхом розв'язання гри з діями гравців,

такими як фолд (пас), колл (нульова ставка або зрівняна з опонентом ставка), ставка в розмірі банку та олл–ін ставка (всі доступні фішки). Додавання нових дій, таких як інакші розміри ставок, значно збільшує розмір ігрового дерева та час роздумів бота, але не сильно впливає на якість стратегії.

Розв'язання гри відбувалось з використанням модифікації алгоритму контрфактичної мінімізації шкодувань, де негативні шкодування обнулялись (CFR+). Для кожної ігрової ситуації обчислювалось 1000 ітерацій CFR+, а до розрахунку усередненої стратегії брались в увагу останні 500. Спочатку відбувалось навчання рівер моделі, далі тьорн, який вже використовував у своїх розрахунках модель ріверу, і аналогічно флоп брав результати тьорн.

Пре-флоп етап значно впливає на швидкодію програми, оскільки вимагає обчислення 22 100 можливих відкритих карт на флопі та оцінки кожної на мережі флопу. Щоб пришвидшити гру перед флопом, була навчена допоміжна нейронна мережа, щоб оцінити очікуване значення мережі флопа на всіх можливих варіантах флопу. Крім того, результат continual re-solving для кожної спостережуваної ситуації перед флопом зберігається у кеш. Коли та сама послідовність ставок повторюється, тоді просто використовуються кешовані результати, без перерахунку.

Зазначені мережі були навчені за допомогою процедури оптимізації градієнтного спуску Адама з функцією втрат Губера [42]. Генерація даних та навчання проводилось на локальному комп'ютері з двома GPU NVIDIA GeForce GTX 1070, та на одному інстансі Google Cloud Platform з NVIDIA Tesla V100 GPU. Весь процес зайняв близько місяця.

4.5 Опис програмного продукту

Покерний бот може бути запущений у двох режимах – гра з реальною людиною та ботом Slumbot на сайті slumbot.com (рис. 4.1).

Рисунок 4.1 – Головна сторінка сайту slumbot.com

Гра з людиною відбувається у консольному режимі. Для цього запускається три вікна – сервер, клієнт-бот та клієнт-людина.

Сервер за допомогою сокет з'єднання отримує та відправляє клієнтам повідомлення про стан гри (рис. 4.2). Повідомлення мають наступний формат: MATCHSTATE:[номер гравця]:[номер гри]:[послідовність дій розділена / між раундами]:[приватні карти]:[публічні карти]

```
FROM 1 at 1607084818.531660 MATCHSTATE:0:0:cc/c:5d5c|/8dAs8s:c
TO 1 at 1607084818.531691 MATCHSTATE:0:0:cc/cc/:5d5c|/8dAs8s/4h
TO 2 at 1607084818.531706 MATCHSTATE:1:0:cc/cc/:|9hQd/8dAs8s/4h
FROM 2 at 1607084826.482350 MATCHSTATE:1:0:cc/cc/:|9hQd/8dAs8s/4h:c
TO 1 at 1607084826.482382 MATCHSTATE:0:0:cc/cc/c:5d5c|/8dAs8s/4h
TO 2 at 1607084826.482392 MATCHSTATE:1:0:cc/cc/c:|9hQd/8dAs8s/4h
FROM 1 at 1607084975.955046 MATCHSTATE:0:0:cc/cc/c:5d5c|/8dAs8s/4h:c
TO 1 at 1607084975.955075 MATCHSTATE:0:0:cc/cc/cc/:5d5c|/8dAs8s/4h/6d
TO 2 at 1607084975.955091 MATCHSTATE:1:0:cc/cc/cc/:|9hQd/8dAs8s/4h/6d
FROM 2 at 1607084979.969437 MATCHSTATE:1:0:cc/cc/cc/:|9hQd/8dAs8s/4h/6d:c
TO 1 at 1607084979.969471 MATCHSTATE:0:0:cc/cc/cc/c:5d5c|/8dAs8s/4h/6d
TO 2 at 1607084979.969484 MATCHSTATE:1:0:cc/cc/cc/c:|9hQd/8dAs8s/4h/6d
```

Рис 4.2 – Консольне вікно серверної частини програми

Клієнт-людина отримує повідомлення з сервера та вводить у консоль обрану дію – фолд (f), колл (c), рейз (число, розмір ставки), олл-ін (число, розмір сумарної кількості фішок гравця, за замовчуванням 20000) (рис 4.3).

```
Received acpc dealer message:
MATCHSTATE:0:0:cc/cc/c:5d5c|/8dAs8s/4h
Our turn
input action:
c
Sending a message to the acpc dealer:
MATCHSTATE:0:0:cc/cc/c:5d5c|/8dAs8s/4h:c
```

Рисунок 4.3 – Консольне вікно клієнта-людини

Клієнт-бот отримує аналогічне повідомлення, запускається наш розроблений бот, на виході він видає розподіл ймовірностей по 4 діям (фолд, колл, рейз, олл-ін) та відповідно до цього розподілу генерує дію (рис. 4.4).

```
strategy:
 0.0000
 1.0000
 0.0000
 0.0000
[torch.CudaTensor of size 4]

playing action that has prob: 1
Sending a message to the acpc dealer:
MATCHSTATE:1:0:cc/cc/cc/:|9hQd/8dAs8s/4h/6d:c
Received acpc dealer message:
MATCHSTATE:1:0:cc/cc/cc/c:|9hQd/8dAs8s/4h/6d
Not our turn
```

Рисунок 4.4 – Консольне вікно клієнта-бота

Для реалізації даного режиму гри був використаний готовий ACPC (англ. Annual Computer Poker Competition) сервер, що є у відкритому доступі на Github.

У режимі гри з ботом Slumbot ігри відбуваються автоматично, також у консольному режимі (рис. 4.5, рис. 4.6), а статистика записується на акаунт, який вказаний у початкових конфігураціях. Для автоматизації процесу гри на сайті Slumbot, було використано Selenium WebDriver та мову програмування Python.

```
strategy:
 0.0000
 0.1787
 0.8213
 0.0000
[torch.CudaTensor of size 4]

playing action that has prob: 0.82130867242813
raise
900
MATCHSTATE:0:4:cr300c/cr1200r3000:7h6s|/8h5d4d
Our turn
```

Рисунок 4.5 – Консольне вікно клієнта

```
sent MATCHSTATE:0:2::Tc2d|
:c
sent MATCHSTATE:0:2:cr300:Tc2d|
:f
sent MATCHSTATE:1:3:r200:|9s4d
:f
sent MATCHSTATE:0:4::7h6s|
:c
sent MATCHSTATE:0:4:cr300:7h6s|
:c
sent MATCHSTATE:0:4:cr300c/c:7h6s|/8h5d4d
:900
```

Рисунок 4.6 – Консольне вікно сервера

4.6 Результати

Щоб оцінити, наскільки добре грає розроблений бот, було прийнято рішення порівняти його з уже існуючим покерним ботом під назвою Slumbot, а саме зіграти з ним 20000 партій та оцінити сумарну статистику. На сьогодні, Slumbot є одним з топових ботів в БТХ, що був створений у 2017 році, та має власний сайт, де кожен може спробувати свої сили та зіграти з ним – slumbot.com [43]. Під капотом він використовує схожий з Libratus [11] алгоритм, та є сильним суперником, тож його часто використовують для тестування написаних моделей, чим ми і скористаємось.

Досліджувалися два показники якості: Raw BB/100 (рис. 4.7) та Baseline BB/100 (рис. 4.8). В кожній грі розміри малого та великого блайнду

дорівнювали 50 та 100 фішок відповідно, а сумарна кількість фішок – 20000 для кожного гравця.

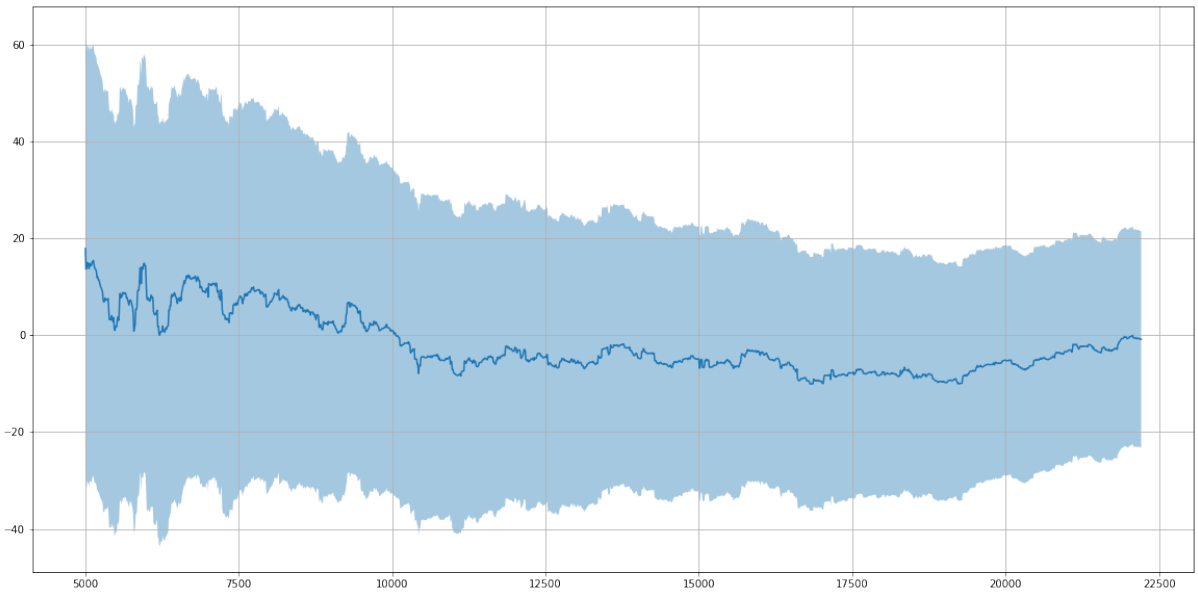

Рисунок 4.7 – Raw BB/100 по кількості ігор для розробленого бота

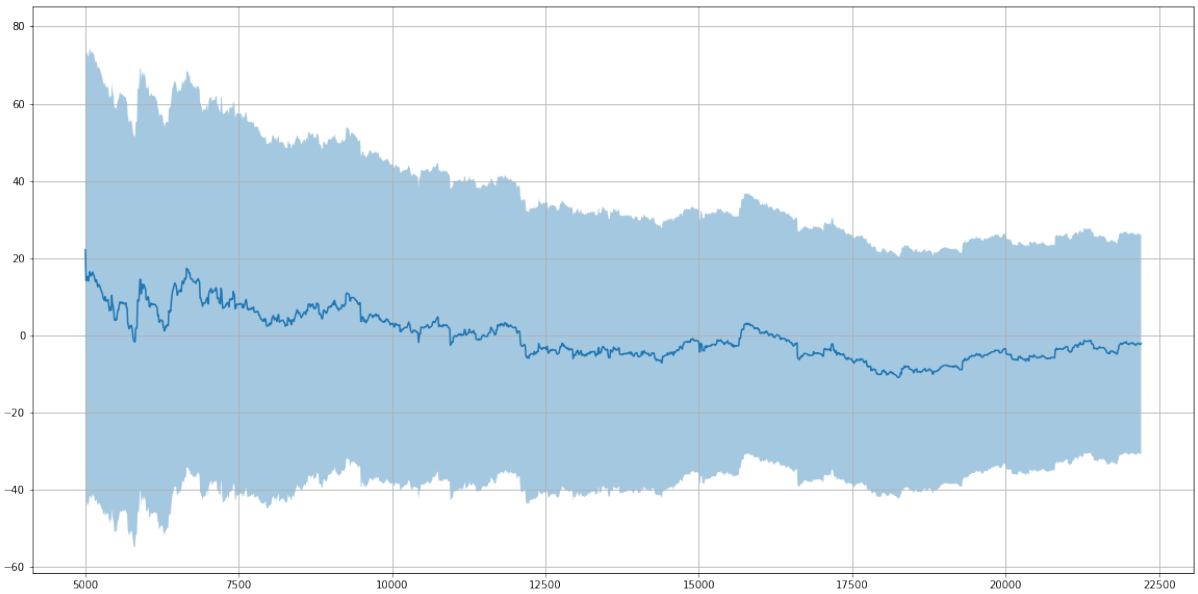

Рисунок 4.8 – Baseline BB/100 по кількості ігор для розробленого бота

Raw BB/100 каже про те, скільки великих блайндів було виграно чи програно сумарно за 100 партій гри:

$$raw\_bb = \frac{total\_reward}{100 * bb\_size}$$

Ця статистика напряму залежить від розданих карт, можлива ситуація, коли буде перевага у картах у одного чи іншого гравця, адже карти роздаються випадковим чином. Тому нас більше цікавить наступна статистика – Baseline BB/100 (рис. 4.8):

$$raw\_bb = \frac{total\_reward - slumbot\_reward}{100 * bb\_size}$$

Показник Baseline BB/100 відповідає за різницю між справжнім прибутком і тим, скільки б міг виграти Slumbot, якби він виявився в аналогічній ситуації на нашому місці, тобто грав би з нашими картами на точно такій же дошці з публічними картами. Тут вже можна сказати, чи краще грає розроблений бот за відомий Slumbot, чи ні, і наскільки краще чи гірше.

Результати гри для розробленого бота під назвою ytuh_diploma наведені на рис. 4.9 і 4.10. Було проведено 20804 партії гри. Значення –0.79 на рис. 6 свідчить про те, що бот за 100 ігор в середньому програє 0.79 біг блайндів або 79 фішок (так як ми задали розмір біг блайнду 100 фішок). Значення –2.15 біг блайндів на 100 ігор означає, що якби Slumbot грав на нашому місці, то він би вигравав в середньому на 215 фішок більше сумарно за 100 ігор, ніж розроблений у роботі бот.

Earnings:	-0.79 BB/100 +/- 22.36
Baseline Earnings:	-2.15 BB/100 +/- 28.37
Num Hands:	20804

Рисунок 4.9 – Фінальна статистика Deepstack

Minimum 20,000 hands	
marcus	142.59
MM_GV_RP_ackr_128	142.20
justIN	16.45
PaperChampion	13.65
bd0918	8.04
hitsz-open-135-0	7.77
ytuh_diploma	-2.15
HorstGonzales	-3.53
x.h.v3m.1	-7.25
Radford76	-7.43

Рисунок 4.10 – Рейтингова таблиця на slumbot.com

Одним з головних критеріїв для оцінки алгоритму є час його роботи.

У табл. 4.1 порівнюється середній час роздумів (у секундах) покерного бота та людей на кожному етапі гри. Як видно, розроблений бот робить хід значно швидше за людину на всіх етапах, окрім флопу.

Таблиця 4.1 – Час роздумів на кожному етапі гри

Етап гри	Алгоритм Deepstack	Люди
Pre–flop	1.348	10.3
Flop	11.26	9.1
Turn	4.63	8.0
River	1.622	9.5

4.7 Аналіз отриманих результатів

Як видно з графіків (рис. 4.7 і 4.8) та кінцевої статистики (рис. 4.9 і 4.10), розроблений бот зайшов у невеликий мінус порівняно із Slumbot, але його можна вважати незначним – за 100 партій гри в середньому бот програє всього 215 фішок з 20000. Важливим є те, що за результатами 20000 ігор розроблений бот потрапив у рейтингову таблицю і посів достойне 7 місце серед усіх інших ботів, що конкурували зі Slumbot (рис. 4.10). Окрім цього, бот показав гарну швидкодію – час його роздумів над ходом у декілька разів менше за людський на всіх етапах гри, за винятком флопу. Таким чином, бот, розроблений у роботі на основі алгоритму Deepstack, залишається одним з найсильніших і може сміливо конкурувати з людьми.

Особливістю розробленого бота є те, що він активно чинить опір аналізу своєї стратегії з боку опонента. З точки зору гри в покер основним завданням DeepStack є пошук рівноваги Неша, тобто мінімізація можливості викриття своєї стратегії іншим гравцем для отримання ним прибутку. Проблема існуючих алгоритмів полягала в тому, що їх стратегії легко викривались за допомогою техніки LBR (local best–response). Стратегія DeepStack абсолютно не викривається за допомогою LBR, що є додатковим плюсом у виборі саме цього алгоритму.

Висновки до розділу 4

У четвертому розділі був детально описаний та проаналізований розроблений програмний продукт – бот, що вміє розв'язувати ігрові ситуації в Безлімітному Техаському Холдемі. Для реалізації програми була обрана мова

програмування Lua з використанням бібліотеки для машинного навчання Torch на базі операційної системи Linux.

Були представлені основні складові програмного продукту та як саме вони застосовуються. Значну увагу було приділено нюансам реалізації, опису використаних методів, а також яким чином та скільки часу відбувалось навчання нейронних мереж.

Далі було показано, яким чином може бути запущений покерний бот, з поясненнями та відповідними скріншотами.

Останнім і найголовнішим завданням цього розділу був опис отриманих результатів роботи програми та проведення їх аналізу. Для цього було проведено більше 20000 партій покеру з уже існуючим ботом під назвою Slumbot та зібрано сумарну статистику. За результатами цих партій було показано, що розробленому боту вдалося навчитись грати в Безлімітний Техаський Холдем та вийти практично в нуль зі своїм сильним суперником.

РОЗДІЛ 5 РОЗРОБКА СТАРТАП ПРОЕКТУ

5.1 Опис ідеї проекту (товару, послуги, технології)

В межах розділу послідовно проаналізовані та подані у вигляді таблиць:

1. Зміст ідеї (що пропонується).
2. Можливі напрямки застосування.
3. Основні вигоди, що може отримати користувач товару (за кожним напрямком застосування).
4. Чим відрізняється від існуючих аналогів та замінників.

Пункти подані у вигляді таблиці (табл. 5.1) і дають цілісне уявлення про зміст ідеї та можливі базові потенційні ринки, в межах яких потрібно шукати групи потенційних клієнтів.

Таблиця 5.1 – Опис ідеї стартап-проекту

Зміст ідеї	Напрямки застосування	Вигоди для користувача
Сервіс для гри в покер pokerludus.com (сайт), де гравець зможе тренуватись та покращувати свої навички покеру, граючи з ботом. Можна регулювати рівень гри бота. Також буде доступний функціонал гри з іншими гравцями.	Комп'ютерні ігри	Можливість обрати різний рівень складності гри, так що гравцю будь-якого рівня, що новачку, що професіоналу знайдеться достойний суперник. Навчитись будувати стратегії гри, аналізуючи гру бота.

Аналіз потенційних техніко-економічних переваг ідеї (чим відрізняється від існуючих аналогів та замінників) порівняно із пропозиціями конкурентів

передбачає: визначення переліку техніко-економічних властивостей та характеристик ідеї; визначення попереднього кола конкурентів (проектів-конкурентів) або товарів-замінників чи товарів-аналогів, що вже існують на ринку, проведення збору інформації щодо значень техніко-економічних показників для ідеї власного проекту та проектів-конкурентів відповідно до визначеного вище переліку; порівняльний аналіз показників: для власної ідеї визначаються показники, що мають а) гірші значення (W, слабкі); б) аналогічні (N, нейтральні) значення; в) кращі значення (S, сильні) (табл. 5.2).

Таблиця 5.2 – Визначення сильних, слабких та нейтральних характеристик ідеї проекту

Техніко-економічні характеристики ідеї	(потенційні) товари/концепції конкурентів			W (слабка сторона)	N (нейтральна сторона)	S (сильна сторона)
	PokerLudus (мій проект)	Pokers Stars	Worldpokerclub			
Доступ з будь-якого девайсу	+	-	+			+
Задання рівня суперника	+	-	-			+
Можливість грати на гроші	-	+	+		+	
Складність розробки	+	+-	+-	+		

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

5.2 Технологічний аудит ідеї проекту

В межах даного підрозділу необхідно провести аудит технології, за допомогою якої можна реалізувати ідею проекту (технології створення товару). Визначення технологічної здійсненності ідеї проекту передбачає аналіз таких складових (табл. 5.3):

1. За якою технологією буде виготовлено товар згідно ідеї проекту.
2. Чи існують такі технології, чи їх потрібно розробити/доробити.
3. Чи доступні такі технології авторам проекту.

Таблиця 5.3 – Технологічна здійсненність ідеї проекту

	Ідея проекту	Технології її реалізації	Наявність технологій	Доступність технологій
1	Розробка повноцінної та автономної системи для гри	Python	+	+
2	Бібліотека для побудови нейронних мереж	PyTorch	+	+
3	Фреймворк для розробки веб-застосунків	Flask	+	+
4	Сервер для хмарних обчислень	Amazon Web Services EC2	+	-(дорого)
5	Сервер для хмарних обчислень	Google Cloud Platform	+	+- (дають 300$ кредиту на 90 днів)

Обрані технології Python+PyTorch+Flask+GCP.

За результатами аналізу таблиці робиться висновок щодо можливості технологічної реалізації проекту: так чи ні, а також технологічного шляху, яким це доцільно зробити (з поміж названих технологій обираються такі, що доступні авторам проекту та є наявними на ринку).

5.3 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 з урахуванням стану ринкового середовища, потреб потенційних клієнтів та пропозицій проектів-конкурентів.

Спочатку проводиться аналіз попиту: наявність попиту, обсяг, динаміка розвитку ринку (табл. 5.4).

Рентабельність — поняття, що характеризує економічну ефективність виробництва, за якої за рахунок грошової виручки від реалізації продукції (робіт, послуг) повністю відшкодовує витрати на її виробництво й одержується прибуток як головне джерело розширеного відтворення.

Таблиця 5.4 – Попередня характеристика потенційного ринку стартап-проекту

	Показники стану ринку (найменування)	Характеристика
1	Кількість головних гравців, од	15
2	Загальний обсяг продаж, грн/ум.од	5$/ум.од
3	Динаміка ринку (якісна оцінка)	Стабільна

Продовження таблиці 5.4

4	Наявність обмежень для входу (вказати характер обмежень)	Немає
5	Специфічні вимоги до стандартизації та сертифікації	Немає
6	Середня норма рентабельності в галузі (або по ринку), %	48%

Суть одного із найважливіших методів оцінки економічної ефективності інвестицій полягає у розрахунку їх середньої рентабельності.

Інвестувати грошові засоби доцільно тоді, коли від цього можна отримати більший прибуток, ніж від їх зберігання. Порівнюючи середньорічну рентабельність інвестицій зі ставкою банківського відсотка, можна дійти висновку, що вигідніше.

Середня норма рентабельності в галузі (або по ринку) порівнюється із банківським відсотком на вкладення. За умови, що останній є вищим, можливо, має сенс вкласти кошти в інший проект.

За результатами аналізу таблиці робиться висновок щодо того, чи є ринок привабливим для входження за попереднім оцінюванням.

Надалі визначаються потенційні групи клієнтів, їх характеристики, та формується орієнтовний перелік вимог до товару для кожної групи (табл. 5.5).

Таблиця 5.5 – Характеристика потенційних клієнтів стартап-проекту

Потреба, що формує ринок	Цільова аудиторія (цільові сегменти ринку)	Відмінності у поведінці різних потенційних цільових груп клієнтів	Вимоги споживачів до товару

Продовження таблиці 5.5

Бажання грати в інтелектуальні ігри, заробляти на цьому гроші	Звичайні користувачі мережі Інтернет	Деякі користувачі будуть грати для лише задоволення, інші будуть навчатись грі, щоб надалі брати участь у професійних турнірах та отримувати грошові винагороди.	Швидкий та простий доступ до сайту, зрозумілий та візуально приємний інтерфейс.

Після визначення потенційних груп клієнтів проводиться аналіз ринкового середовища: складаються таблиці факторів, що сприяють ринковому впровадженню проекту, та факторів, що йому перешкоджають (табл. 5.6 – 5.7). Фактори в таблиці подаються в порядку зменшення значущості.

Таблиця 5.6 – Фактори загроз

Фактор	Зміст загрози	Можлива реакція компанії
Конкуренція	Збільшення зацікавленості основних гравців в цій галузі	Розробка унікального функціоналу, покращення візуальних рішень (UI/UX) та клієнтської підтримки
Зміна потреб користувачів	Зміна пріоритетів користувачів, у зв’язку з якими, програмне забезпечення перестає задовольняти клієнта	Розширення функціональних можливостей сервісу.
Низькі продажі	Сервіс не затребуваний, погано поширюється	Збільшити витрати на маркетинг

Таблиця 5.7 – Фактори можливостей

Фактор	Зміст можливості	Можлива реакція компанії
Увага компаній	Привернення уваги основних гравців ринку до нашого продукту	Розширення команди та початок роботи над покращенням сервісу
Висока популярність веб-застосунку	Увага аудиторії до продукту дозволяє пропонувати їм більше можливостей	Пропозиції укласти угоду із зацікавленими партнерами для розширенні аудиторії за рахунок реклами

Надалі проводиться аналіз пропозиції: визначаються загальні риси конкуренції на ринку (табл. 5.8, табл. 5.9)

Таблиця 5.8 – Ступеневий аналіз конкуренції на ринку

Особливості конкурентного середовища	В чому проявляється дана характеристика	Вплив на діяльність підприємства (можливі дії компанії, щоб бути конкурентоспроможною)
1. Вказати тип конкуренції - монополія/олігополія/ монополістична/чиста	Чиста	Розробка унікального функціоналу, покращення візуальних рішень (UI/UX) та клієнтської підтримки. Більшість функціоналу сервісу надається безкоштовно на відміну від конкурентів.

Продовження таблиці 5.8

2. За рівнем конкурентної боротьби локальний/національний/	Світовий	Сама природа нашого сервісу дозволяє пропонувати продукт всьому світові без затрат
3. За галузевою ознакою - міжгалузева/ внутрішньогалузева	Внутрішньогалузева	Велика кількість інших розроблених сервісів, в яких наш може просто загубитися, тому необхідно зосередити увагу клієнта на унікальності нашого продукту
4. Конкуренція за видами товарів: - товарно-родова - товарно-видова - між бажаннями	Між бажаннями	Вкладення в різні види реклами, щоб переконати клієнта, що йому потрібен наш продукт
3. За характером конкурентних переваг - цінова / нецінова	Нецінова	Конкурентні переваги — функції та можливості додатку
6. За інтенсивністю - марочна/не марочна	Марочна	Необхідно зосередити увагу клієнта на бренді та унікальності нашого продукту

Таблиця 5.9 – Аналіз конкуренції в галузі за М. Портером

Складові аналізу	Прямі конкуренти в галузі	Потенційні конкуренти	Клієнти	Товари-замінники
	PokerStars, WorldPokerclub	Потрібен велика кількість часу і ресурсів	Очікування щодо якості обслуговування ,вплив покупців на ринкову ціну	Поява відносно дешевих або більш зручних у використанні замінників
Висновки	Наш сервіс буде надавати унікальні пропозиції, яких у немає конкурентів	Є можливості виходу на ринок в найближчі строки	Ні	Обмежень немає

За результатами аналізу таблиці робиться висновок щодо принципової можливості роботи на ринку з огляду на конкурентну ситуацію. Також робиться висновок щодо характеристик (сильних сторін), які повинен мати проект, щоб бути конкурентоспроможним на ринку.

На основі аналізу конкуренції, проведеного в табл. 5.9, а також із урахуванням характеристик ідеї проекту (табл. 5.2), вимог споживачів до товару (табл. 5.5) та факторів маркетингового середовища (табл. 5.6-5.7) визначається та обґрунтовується перелік факторів конкурентоспроможності (табл. 5.10).

Таблиця 5.10 – Обґрунтування факторів конкурентоспроможності

Фактор конкурентоспроможності	Обґрунтування (наведення чинників, що роблять фактор для порівняння конкурентних проектів значущим)
Якість продукту	Один із факторів для вибору продукту клієнтом.
Кількість видів соціальної взаємодії	Більша зацікавленість клієнта продуктом.
Ціна	Один із факторів для вибору продукту клієнтом.
Простота експлуатації	Один із факторів для зберігання зацікавленості клієнта.

За визначеними факторами конкурентоспроможності (табл. 5.10) проводиться аналіз сильних та слабких сторін стартап-проекту (табл. 5.11).

Таблиця 5.11 – Порівняльний аналіз сильних та слабких сторін PokerLumus

	Фактор конкурентоспроможності	Бали 1-20	Рейтинг товарів-конкурентів у порівнянні						
			-3	-2	-1	0	1	2	3
1	Якість продукту	15			+				
2	Різноманіття функціоналу	10				+			
3	Ціна	10						+	
4.	Простота експлуатації	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) (табл.

5.12) на основі виділених ринкових загроз та можливостей, та сильних і слабких сторін (табл. 5.11).

Перелік ринкових загроз та ринкових можливостей складається на основі аналізу факторів загроз та факторів можливостей маркетингового середовища. Ринкові загрози та ринкові можливості є наслідками 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 5.12 – SWOT- аналіз стартап-проекту

Сильні сторони: ціна, простота використання	Слабкі сторони: досі невідома компанія
Можливості: наявність безкоштовного функціоналу, унікальні рішення в продукті	Загрози: усування з ринку більшими конкурентами, вузький сегмент користувачів

На основі SWOT-аналізу розробляються 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 (табл. 5.9, аналіз потенційних конкурентів).

Визначені альтернативи аналізуються з точки зору строків та ймовірності отримання ресурсів (табл. 5.11).

З означених альтернатив (табл. 5.13) обирається та, для якої:

1) отримання ресурсів є більш простим та ймовірним;
2) строки реалізації – більш стислими.

Таблиця 5.13 Альтернативи ринкового впровадження стартап-проекту

Альтернатива (орієнтовний комплекс заходів) ринкової поведінки	Ймовірність отримання ресурсів	Строки реалізації
PR, просування бренду	50%	5
Перехід на безкоштовне розповсюдження	65%	3
Партнерство для об’єднання продукції	75%	3

5.4 Розробка ринкової стратегії продукту

Розробка ринкової стратегії першим кроком передбачає визначення стратегії охоплення ринку: опис цільових груп потенційних споживачів (табл. 5.14)

Таблиця 5.14 – 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	Поодинокі користувачі	Середня	Низький	Висока	Висока

Продовження таблиці 5.14

2	Люди, що заробляють на покері	Висока	Високий	Середня	Низька
3	Професіональні гравці у покер	Висока	Високий	Низька	Низька
Які цільові групи обрано: 2,3					

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;

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

- якщо компанія працює зі всім ринком, пропонуючи стандартизовану програму (включно із характеристиками товару/послуги) – вона використовує масовий маркетинг.

Для роботи в обраних сегментах ринку необхідно сформувати базову стратегію розвитку (табл. 5.15).

Стратегія диференціації передбачає надання товару важливих з точки зору споживача відмітних властивостей, які роблять товар відмінним від товарів конкурентів. Така відмінність може базуватися на об’єктивних або суб’єктивних, відчутних і невідчутних властивостях товару(у ширшому розумінні – комплексі маркетингу), бути реальною або уявною. Інструментом реалізації стратегії диференціації є ринкове позиціонування.

Реалізація цієї стратегії вимагає, як правило, більш високих витрат. Проте успішна диференціація дозволяє компанії домогтись більшої рентабельності за рахунок того, що ринок готовий прийняти більш високу ціну (цінову премію бренду).

При веденні конкурентної боротьби з використанням цієї стратегії на ринку в першу чергу терплять фіаско фірми, що не здатні визначати потреби цільових ринків, оперативно реагувати на зміни в ринковому попиті, проводити ефективну політику маркетингових комунікацій, не мають необхідних навичок в області брендингу. Найважливішими здібностями, які повинна мати компанія, що приймає цю стратегію, є з генерування маркетингових ноу-хау, здійснення продуктових інновацій.

5.5 Розробка маркетингової програми стартап-проекту

Таблиця 5.15 – Визначення базової стратегії розвитку

Обрана альтернатива розвитку проекту	Стратегія охоплення ринку	Ключові конкурентоспроможні позиції відповідно до обраної альтернативи	Базова стратегія розвитку
Надання товару важливих з точки зору споживача відмітних властивостей	Стратегія диференціації	Більша кількість способів соціальної взаємодії, можливість переходу до частково безкоштовного типу розповсюдження	Стратегія диференціації

Наступним кроком є вибір стратегії конкурентної поведінки.

Компанії, що приймають слідування за лідером – це підприємства з невеликою часткою ринку, які вибирають адаптивну лінію поведінки на ринку, усвідомлюють своє місце на нім і йдуть у фарватері фірм-лідерів. Головна перевага такої стратегії – економія фінансових ресурсів, пов'язаних з необхідністю розширення товарного(галузевого) ринку, постійними інноваціями, витратами на утримання домінуючого положення

Стратегія наслідування лідеру найчастіше має місце у випадку олігополії, коли кожен конкурент прагне уникнути боротьби, особливо цінової, а також у випадку, коли слабо виражений ефект масштабу, що не дозволяє отримати переваги від об'ємів продажів або ж він не грає істотної ролі. Стратегію наслідування лідеру приймають також фірми, які не змогли реалізувати стратегію виклику лідерові.

Компанії, що приймають таку стратегію, зазвичай випускають товари-імітатори, займаючи ринкову частку, яку з різних причин не можуть охопити фірми лідери. Вибір такої стратегії може також бути обумовлений також перевагою локалізації (краще знання ринку, налагоджені зв'язки з клієнтами тощо).

Для ефективної реалізації цієї стратегії компанії повинні задовольняти наступним основним умовам:

- систематичний аналіз сегментації ринку з метою виділення нових ринкових сегментів або таких, що незадовільно обслуговуються;

- ефективне використання НДДКР з метою вдосконалення технологічних процесів і незначних продуктових інновацій;

- концентрація на прибутковості, а не на простому зростанні об'ємів продажів;

- залишатися досить малим, щоб не бути досить цікавим для фірм-лідерів;

- сильний керівник, здатний не лише формулювати стратегію, але і тримати усю діяльність компанії під власним контролем.

Якщо врахувати, що лідерами ринку можуть бути лише декілька компаній, то ця стратегія наймасовішою.

На основі вимог споживачів з обраних сегментів до постачальника (стартап-компанії) та до продукту (див. табл. 5.5), а також в залежності від обраної базової стратегії розвитку (табл. 5.15) та стратегії конкурентної поведінки (табл. 5.16) розробляється стратегія позиціонування (табл. 5.17). що полягає у формуванні ринкової позиції (комплексу асоціацій), за яким споживачі мають ідентифікувати торгівельну марку/проект.

Таблиця 5.16 – Визначення базової стратегії конкурентної поведінки

Чи є проект «першопрохідцем» на ринку?	Чи буде компанія шукати нових споживачів, або забирати існуючих у конкурентів?	Чи буде компанія копіювати основні характеристики товару конкурента, і які?	Стратегія конкурентної поведінки
Ні	Забирати існуючих	Ні, буде їх пов’язувати та розширювати, створюючи новий функціонал	Стратегія наслідування лідеру

Таблиця 5.17 – Визначення стратегії позиціонування

Вимоги до товару цільової аудиторії	Базова стратегія розвитку	Ключові конкурентоспроможні позиції власного стартап-проекту	Вибір асоціацій, які мають сформувати комплексну позицію власного проекту (три ключових)

Продовження таблиці 5.17

Якість	Позиціювання за показниками якості	Тестування розробленого продукту та виправлення всіх багів	Стабільність роботи, якість роботи
Більша кількість функціональних можливостей	На основі специфічних відчутних характеристик	Розробка більшої кількості оригінальних можливостей	Велика можливостей

Результатом виконання підрозділу має стати узгоджена система рішень щодо ринкової поведінки стартап-компанії, яка визначатиме напрями роботи стартап-компанії на ринку.

Висновки до розділу 5

В даному розділі було проведено аналіз програмного продукту у якості стартап проекту. Можна зазначити, що у проекта є можливість комерціалізації, адже у нашого продукту є попит у користувачів, а також деякий його функціонал є унікальним, що створює цінність нашому сервісу.

Проект є доволі конкурентоспроможним, проте для його реалізації необхідно буде залучати певні інвестиції для розширення штату робітників, оренди серверів, оренди офісу, купівлі потужного обладнання, яке необхідне для коректної роботи сервісу.

Для впровадження ринкової реалізації проекту слід обрати альтернативу, яка передбачає розробку програмного продукту, а потім якісну рекламу та PR, сконцентровану навколо позитивних характеристиках даного

програмного продукту, таких як низька ціна, доступність, широкий функціонал тощо.

ВИСНОВКИ

Основним завдання даної роботи було дослідження найновіших методів навчання з підкріпленням для великих ігор з неповною інформацією. В якості прикладу такої гри була взяти версія покеру під назвою Безлімітний Техаський Холдем. Результатом даної роботи стала розробка програмного забезпечення, яке вміє оптимально розв'язувати окремі ситуації в покері, та ігрові партії в цілому.

На першому етапі було описано актуальність даної задачі, та проведено історичний огляд існуючих рішень. Були розглянуті основи гри безлімітного Техаського холдему, описано процес гри, її правила, а також існуючі версії гри.

У другому розділі були описані теоретичні основи, на яких будуються методи для розв'язання покерної гри. Було розкрито важливі поняття з теорії ігор та представлено теоретичне підґрунтя одного з напрямів машинного навчання – навчання з підкріпленням (Reinforcement Learning).

Третій розділ був присвячений дослідженню методів, які використовуються при розв'язуванні великих ігор з неповною інформацією, в тому числі і покерної гри.

У четвертому розділі був описаний процес та деталі розробки покерного клієнта для Безлімітного Техаського Холдему. Для розв'язання даної задачі було вирішено застосувати алгоритм Deepstack, описаний у третьому розділі. Було проведено опис отриманих результатів роботи програми та їх аналіз. Розроблений покерний клієнт показав гідний результат – вийшов практично в нуль зі Slumbot, одним з найсильніших сучасних покерних ботів на даний момент, та посів 7 місце у рейтинговій таблиці. Експериментально було встановлено ще одну перевагу розробленого боту: він робить хід значно швидше за людину на всіх етапах, окрім флопу.

Задача розв'язування покерної гри є дуже перспективною, існує ще багато можливостей покращень описаних методів, такі як модифікації CFR алгоритму, застосування різних архітектур нейронних мереж, імплементація програми на C++ з CUDA оптимізацією для збільшення швидкодії і т.д.

ПЕРЕЛІК ПОСИЛАНЬ

1. A. Hodges. Alan Turing: the enigma. Princeton, N.J: Princeton University Press, 2012, 587 p.
2. J. Schaeffer, R. Lake, P. Lu, and M. Bryant. Chinook. The world man-machine checkers champion. *AI Magazine*, 1996, Vol. 17, No. 21. P. 21–29.
3. J. Schaeffer, N. Burch, Y. Bjornsson, A. Kishimoto, M. Muller, R.Lake. Lu, S.Sutphen. Checkers is solved. *Science*. 2007. Vol. 317, No. 5844. P. 1518-1522.
4. F.H. Hsu. Behind Deep Blue: Building the computer that defeated the world chess champion. Princeton, Princeton University Press, 2002, 320 p.
5. B. Sheppard. World-championship-caliber Scrabble. *Artificial Intelligence*. 2002. Vol. 134. No. 1-2. P. 241–276.
6. G. Tesauro. Temporal difference learning and TD-Gammon. *Communications of the ACM*, 1995, P.58–68.
7. Rehmeyer, J.; Fox, N.; and Rico. Ante up, human: The adventures of polaris the poker-playing robot. *Wired*, 2008, Vol. 16. No. 1. P.186–191.
8. The Second Man-Machine Poker Competition. URL: http://poker.cs.ualberta.ca/man-machine/.
9. Правила покеру: як навчитися грати в покер. URL: https://ua.cardmates.net/pravila_pokeru
10. Развитие искусственного интеллекта в покере за 20 лет. URL: https://pekarstas.com/blog/razvitie-iskusstvennogo-intellekta-v-pokere-za-20-let/
11. Brown, N. and T. Sandholm. Libratus: The Superhuman AI for No-Limit Poker. *International Joint Conference on Artificial Intelligence*, 2017. P. 5226-5228.
12. Brown, N. and Sandholm, T. Superhuman AI for multiplayer poker. *Science*. 2019. Vol. 365. No. 6456. P. 885–890.

13. Game Theory. URL:https://www.investopedia.com/terms/g/gametheory.asp
14. Теорія ігор. URL: https://uk.wikipedia.org/wiki/Теорія_ігор
15. Marc Lanctot, Kevin Waugh, Martin Zinkevich, and Michael Bowling. Monte Carlo sampling for regret minimization in extensive games. *Advances in Neural Information Processing Systems 22*. 2009. P. 1078-1086.
16. J.F. Nash. Equilibrium points in n-person games. *Proceedings of the National Academy of Sciences of the United States of America*. 1950. Vol. 36, No. 1. P.48–49.
17. Todd W. Neller and Marc Lanctot. An introduction to counterfactual regret minimization. *In Proceedings of Model AI Assignments, The Fourth Symposium on Educational Advances in Artificial Intelligence*, 2013.
18. Sergiu Hart and Andreu Mas-Colell. A simple adaptive procedure leading to correlated equilibrium. *Econometrica*. 2000. Vol.68. No.1128.P.1127–1150.
19. Саттон Р.С, Э. Г. Барто Обучение с подкреплением. Москва Лаборатория знаний. 2011. 399с.
20. Melo F.S. Convergence of Q-learning: A simple proof. 2001. Institute Of Systems and Robotics, Tech. Rep. 1–4.
21. Mnih V. Human-level control through deep reinforcement learning. *Nature*, 2015. Vol. 518. No. 7540. P. 529–533.
22. Schaul T., Quan J., Antonoglou I., Silver D. Prioritized experience replay. URL:http://www0.cs.ucl.ac.uk/staff/d.silver/web/Publications_files/prioritized-replay.pdf
23. Silver D., Lever G., Heess N., Degris T., Wierstra D., Riedmiller M. Deterministic policy gradient algorithms. URL: http://www0.cs.ucl.ac.uk/staff/d.silver/web/Publications_files/deterministic-policy-gradients.pdf
24. Николенко С., Кадурин А., Архангельская Е. Глубокое обучение. СПб: Питер, 2017. 480 с.
25. Mnih V., Badia A.P., Mirza M., Graves A., Harley T., Lillicrap T.P., Silver D., Kavukcuoglu K. Asynchronous methods for deep reinforcement

learning. Proc. *International Conference on Machine Learning 16*, 2016. P. 1928-1937.

26. Steps to building a Poker AI — Sequential Games, Kuhn Poker and Counterfactual Regrets. URL: https://medium.com/ai–in–plain–english/building–a–poker–ai–part–5–sequential–games–kuhn–poker–and–counterfactual–regrets–311f533f786e
27. Oskari Tammelin. CFR+. *CoRR*. Vol.1407. No.5042. 2014.
28. Brown, N. and Sandholm, T. Solving imperfect-information games via discounted regret minimization. *AAAI Conference on Artificial Intelligence 33*, 2019. P.1829-1836.
29. Noam Brown, Christian Kroer, and Tuomas Sandholm. Dynamic thresholding and pruning for regret minimization. *AAAI Conference on Artificial Intelligence*. 2017. P.421–429.
30. Michal Sustr, Vojtech Kovarík, and Viliam Lisy. Monte carlo continual resolving for online strategy computation in imperfect information games. *International Foundation for Autonomous Agents and Multiagent Systems*. 2019. P.224–232.
31. Martin Zinkevich, Michael Johanson, Michael Bowling, and Carmelo Piccione. Regret minimization in games with incomplete information. *In Advances in neural information processing systems 20*, 2008. P.1729–1736.
32. M. Johanson, N. Bard, M. Lanctot, R. Gibson, and M. Bowling. Efficient Nash equilibrium approximation through Monte Carlo counterfactual regret minimization. *In Proceedings of the Eleventh International Conference on Autonomous Agents and Multi-Agent Systems*, 2012. P.837-746.
33. N. Burch, M. Johanson, M. Bowling, *Proceedings of the Twenty–Eighth Conference on Artificial Intelligence*. 2014. P. 602–608.
34. Moravcík, M. et al. DeepStack: expert–level artificial intelligence in heads–up no–limit poker. *Science*. Vol. 356. No.6337. P.508–513.
35. P. Ciancarini, G.P. Favini. Monte Carlo tree search in Kriegspiel. *Artificial Intelligence*. 2010. Vol. 174. No. 11. P.670-684.

36. N. Yakovenko, L. Cao, C. Raffel, J. Fan. *Proceedings of the Thirtieth Conference on Artificial Intelligence*. 2016. P.360–367.
37. Vanilla Counterfactual Regret Minimization for Engineers. URL: https://justinsermeno.com/posts/cfr/
38. A. Krizhevsky, I. Sutskever, G. E. Hinton. Imagenet classification with deep Convolutional neural networks. *Advances in Neural Information Processing Systems*. 2012. Vol. 25. P.1106–1114.
39. K. He, X. Zhang, S. Ren, J. Sun. Delving deep into rectifiers: surpassing human-level performance on imagenet classification. *Proceedings of the 2015 IEEE International Conference on Computer Vision*. 2015. P.1026–1034.
40. A. Gilpin, T. Sandholm, T. B. Sørensen. Potential-aware automated abstraction of sequential games, and holistic equilibrium analysis of Texas hold'em poker. *Proceedings of the Twenty–Second Conference on Artificial Intelligence.* 2007. P.50–57.
41. Lua. URL: https://uk.wikipedia.org/wiki/Lua
42. P. J. Huber. Robust Estimation of a Location Parameter. *Annals of Mathematical Statistics*. 1964. Vol. 35. No.1. P.73-101.
43. 9 различий в игре против бота и против человека. URL: https://pekarstas.com/blog/9–razlichiy–v–igre–protiv–bota–protiv–cheloveka/

ДОДАТОК А. ЛІСТИНГ ПРОГРАМИ

Модуль Lookahead

```
require 'Lookahead.lookahead_builder'
require 'TerminalEquity.terminal_equity'
require 'Lookahead.cfrd_gadget'
local arguments = require 'Settings.arguments'
local constants = require 'Settings.constants'
local game_settings = require 'Settings.game_settings'
local tools = require 'tools'
local card_tools = require 'Game.card_tools'
local card_to_string = require 'Game.card_to_string_conversion'

local Lookahead = torch.class('Lookahead')
local timings = {}
--- Constructor
function Lookahead:__init(terminal_equity, batch_size)
  self.builder = LookaheadBuilder(self)
  self.terminal_equity = terminal_equity
  self.batch_size = batch_size
end

--- Constructs the lookahead from a game's public tree.
--
-- Must be called to initialize the lookahead.
-- @param tree a public tree
function Lookahead:build_lookahead(tree)
  self.builder:build_from_tree(tree)
end

function Lookahead:reset()
  self.builder:reset()
end
--- Re-solves the lookahead using input ranges.
--
-- Uses the input range for the opponent instead of a gadget range, so only
-- appropriate for re-solving the root node of the game tree (where ranges
-- are fixed).
--
-- @{build_lookahead} must be called first.
--
-- @param player_range a range vector for the re-solving player
-- @param opponent_range a range vector for the opponent
function Lookahead:resolve_first_node(player_range, opponent_range)
  self.ranges_data[1][{{}, {}, {}, {}, 1, {}}]:copy(player_range)
  self.ranges_data[1][{{}, {}, {}, {}, 2, {}}]:copy(opponent_range)
  self:_compute()
end

--- Re-solves the lookahead using an input range for the player and
-- the @{cfrd_gadget|CFRDGadget} to generate ranges for the opponent.
--
-- @{build_lookahead} must be called first.
--
-- @param player_range a range vector for the re-solving player
-- @param opponent_cfvs a vector of cfvs achieved by the opponent
```

```
-- before re-solving
function Lookahead:resolve(player_range, opponent_cfvs)
  assert(player_range)
  assert(opponent_cfvs)

  self.reconstruction_gadget = CFRDGadget(self.tree.board, player_range, opponent_cfvs)

  self.ranges_data[1][{{}, {}, {}, {}, 1, {}}]:copy(player_range)
  self.reconstruction_opponent_cfvs = opponent_cfvs
  self:_compute()
end

--- Re-solves the lookahead.
-- @local
function Lookahead:_compute()
  --1.0 main loop

  for i=1,8 do
    timings[i] = 0
  end
  for iter=1,arguments.cfr_iters do

    local timer = torch.Timer()
    timer:reset()
    self:_set_opponent_starting_range(iter)
    timings[1] = timings[1] + timer:time().real
    timer:reset()
    self:_compute_current_strategies()
    timings[2] = timings[2] + timer:time().real
    timer:reset()
    self:_compute_ranges()
    timings[3] = timings[3] + timer:time().real
    timer:reset()
    self:_compute_update_average_strategies(iter)
    timings[4] = timings[4] + timer:time().real
    timer:reset()
    self:_compute_terminal_equities()
    timings[5] = timings[5] + timer:time().real
    timer:reset()
    self:_compute_cfvs()
    timings[6] = timings[6] + timer:time().real
    timer:reset()
    self:_compute_regrets()
    timings[7] = timings[7] + timer:time().real
    timer:reset()
    self:_compute_cumulate_average_cfvs(iter)
    timings[8] = timings[8] + timer:time().real
    timer:reset()
  end

  for i=1,8 do
    print(' ' .. i .. ': ' .. timings[i])
  end
  --2.0 at the end normalize average strategy
  self:_compute_normalize_average_strategies()
  --2.1 normalize root's CFVs
  self:_compute_normalize_average_cfvs()
end

--- Uses regret matching to generate the players' current strategies.
-- @local
function Lookahead:_compute_current_strategies()
  for d=2,self.depth do
```

```
    self.positive_regrets_data[d]:copy(self.regrets_data[d])
    self.positive_regrets_data[d]:clamp(self.regret_epsilon, tools:max_number())

    --1.0 set regret of empty actions to 0
    self.positive_regrets_data[d]:cmul(self.empty_action_mask[d])

    --1.1  regret matching
    --note that the regrets as well as the CFVs have switched player indexing
    torch.sum(self.regrets_sum[d], self.positive_regrets_data[d], 1)
    local player_current_strategy = self.current_strategy_data[d]
    local player_regrets = self.positive_regrets_data[d]
    local player_regrets_sum = self.regrets_sum[d]

    player_current_strategy:cdiv(player_regrets, player_regrets_sum:expandAs(player_regrets))
  end
end

--- Using the players' current strategies, computes their probabilities of
-- reaching each state of the lookahead.
-- @local
function Lookahead:_compute_ranges()

  for d=1,self.depth-1 do
    local current_level_ranges = self.ranges_data[d]
    local next_level_ranges = self.ranges_data[d+1]

    local prev_layer_terminal_actions_count = self.terminal_actions_count[d-1]
    local prev_layer_actions_count = self.actions_count[d-1]
    local prev_layer_bets_count = self.bets_count[d-1]
    local gp_layer_nonallin_bets_count = self.nonallinbets_count[d-2]
    local gp_layer_terminal_actions_count = self.terminal_actions_count[d-2]


    --copy the ranges of inner nodes and transpose
    self.inner_nodes[d]:copy(current_level_ranges[{{prev_layer_terminal_actions_count+1, -1}, {1,
gp_layer_nonallin_bets_count}, {}, {}, {}, {}}]:transpose(2,3))

    local super_view = self.inner_nodes[d]
    super_view = super_view:view(1, prev_layer_bets_count, -1, self.batch_size, constants.players_count,
game_settings.hand_count)

    super_view = super_view:expandAs(next_level_ranges)
    local next_level_strategies = self.current_strategy_data[d+1]

    next_level_ranges:copy(super_view)

    --multiply the ranges of the acting player by his strategy
    next_level_ranges[{{}, {}, {}, {}, self.acting_player[d], {}}]:cmul(next_level_strategies)
  end
end

--- Updates the players' average strategies with their current strategies.
-- @param iter the current iteration number of re-solving
-- @local
function Lookahead:_compute_update_average_strategies(iter)
  if iter > arguments.cfr_skip_iters then
    --no need to go through layers since we care for the average strategy only in the first node anyway
    --note that if you wanted to average strategy on lower layers, you would need to weight the current strategy by the
current reach probability
    self.average_strategies_data[2]:add(self.current_strategy_data[2])
  end
end
```

```
--- Using the players' reach probabilities, computes their counterfactual
-- values at each lookahead state which is a terminal state of the game.
-- @local
function Lookahead:_compute_terminal_equities_terminal_equity()

  -- copy in range data
  for d=2,self.depth do
    if d > 2 or self.first_call_terminal then
      if self.tree.street ~= constants.streets_count then
        self.ranges_data_call[{self.term_call_indices[d]}]:copy(self.ranges_data[d][2][-1])
      else
        self.ranges_data_call[{self.term_call_indices[d]}]:copy(self.ranges_data[d][2])
      end
    end
    self.ranges_data_fold[{self.term_fold_indices[d]}]:copy(self.ranges_data[d][1])
  end

  self.terminal_equity:call_value(self.ranges_data_call:view(-1, game_settings.hand_count),
self.cfvs_data_call:view(-1, game_settings.hand_count))
  self.terminal_equity:fold_value(self.ranges_data_fold:view(-1, game_settings.hand_count),
self.cfvs_data_fold:view(-1, game_settings.hand_count))

  for d=2,self.depth do
    if self.tree.street ~= constants.streets_count then
      if game_settings.nl and (d>2 or self.first_call_terminal) then
        self.cfvs_data[d][2][-1]:copy(self.cfvs_data_call[{self.term_call_indices[d]}])
      end
    else
      if d>2 or self.first_call_terminal then
        self.cfvs_data[d][2]:copy(self.cfvs_data_call[{self.term_call_indices[d]}])
      end
    end
    self.cfvs_data[d][1]:copy(self.cfvs_data_fold[{self.term_fold_indices[d]}])

    --correctly set the folded player by mutliplying by -1
    local fold_mutliplier = (self.acting_player[d]*2 – 3)
    self.cfvs_data[d][{1, {}, {}, {}, 1, {}}]:mul(fold_mutliplier)
    self.cfvs_data[d][{1, {}, {}, {}, 2, {}}]:mul(-fold_mutliplier)
  end
end

--- Using the players' reach probabilities, calls the neural net to compute the
-- players' counterfactual values at the depth-limited states of the lookahead.
-- @local
function Lookahead:_compute_terminal_equities_next_street_box()

  assert(self.tree.street ~= constants.streets_count)

  if self.num_pot_sizes == 0 then
    return
  end

  for d=2, self.depth do
    if d > 2 or self.first_call_transition then

      -- if there's only 1 parent, then it should've been an all in, so skip this next_street_box calculation
      if self.ranges_data[d][2]:size(1) > 1 or (d == 2 and self.first_call_transition) or not game_settings.nl then
        local parent_indices = {1, -2}
        if d == 2 then
          parent_indices = {1,1}
        elseif not game_settings.nl then
          parent_indices = {}
        end
```

```
      self.next_street_boxes_outputs[{self.indices[d], {}, {}, {}}]:copy(self.ranges_data[d][{2, parent_indices, {},
{}, {}, {}}])
    end
  end
 end

 if self.tree.current_player == 2 then
  self.next_street_boxes_inputs:copy(self.next_street_boxes_outputs)
 else
  self.next_street_boxes_inputs[{{}, {}, 1, {}}]:copy(self.next_street_boxes_outputs[{{}, {}, 2, {}}])
  self.next_street_boxes_inputs[{{}, {}, 2, {}}]:copy(self.next_street_boxes_outputs[{{}, {}, 1, {}}])
 end

 if self.tree.street == 1 then
  self.next_street_boxes:get_value_aux(
   self.next_street_boxes_inputs:view(-1, constants.players_count, game_settings.hand_count),
   self.next_street_boxes_outputs:view(-1, constants.players_count, game_settings.hand_count),
   self.next_board_idx)
 else
  self.next_street_boxes:get_value(
   self.next_street_boxes_inputs:view(-1, constants.players_count, game_settings.hand_count),
   self.next_street_boxes_outputs:view(-1, constants.players_count, game_settings.hand_count))
 end

 --now the neural net outputs for P1 and P2 respectively, so we need to swap the output values if necessary
 if self.tree.current_player == 2 then
  self.next_street_boxes_inputs:copy(self.next_street_boxes_outputs)

  self.next_street_boxes_outputs[{{}, {}, 1, {}}]:copy(self.next_street_boxes_inputs[{{}, {}, 2, {}}])
  self.next_street_boxes_outputs[{{}, {}, 2, {}}]:copy(self.next_street_boxes_inputs[{{}, {}, 1, {}}])
 end

 for d=2, self.depth do
  if d > 2 or self.first_call_transition then
   if self.ranges_data[d][2]:size(1) > 1 or (d == 2 and self.first_call_transition) or not game_settings.nl then
    local parent_indices = {1, -2}
    if d == 2 then
     parent_indices = {1,1}
    elseif not game_settings.nl then
     parent_indices = {}
    end
    self.cfvs_data[d][{2, parent_indices, {}, {}, {}, {}}]:copy(self.next_street_boxes_outputs[{self.indices[d], {},
{}, {}}])
    end
  end
 end
end

--- Gives the average counterfactual values for the opponent during re-solving
-- after a chance event (the betting round changes and more cards are dealt).
--
-- Used during continual re-solving to track opponent cfvs. The lookahead must
-- first be re-solved with @{resolve} or @{resolve_first_node}.
--
-- @param action_index the action taken by the re-solving player at the start
-- of the lookahead
-- @param board a tensor of board cards, updated by the chance event
-- @return a vector of cfvs
function Lookahead:get_chance_action_cfv(action, board)

 local box_outputs = self.next_street_boxes_outputs:view(-1, constants.players_count, game_settings.hand_count)
 local next_street_box = self.next_street_boxes
 local batch_index = self.action_to_index[action]
```

```
  assert(batch_index ~= nil)
  local pot_mult = self.next_round_pot_sizes[batch_index]

  if box_outputs == nil then
    assert(false)
  end
  next_street_box:get_value_on_board(board, box_outputs)

  local out = box_outputs[batch_index][self.tree.current_player]
  out:mul(pot_mult)

  return out
end

--- Using the players' reach probabilities, computes their counterfactual
-- values at all terminal states of the lookahead.
--
-- These include terminal states of the game and depth-limited states.
-- @local
function Lookahead:_compute_terminal_equities()
  if self.tree.street ~= constants.streets_count then
    self:_compute_terminal_equities_next_street_box()
  end

  self:_compute_terminal_equities_terminal_equity()
  --multiply by pot scale factor
  for d=2,self.depth do
    self.cfvs_data[d]:cmul(self.pot_size[d])
  end
end

--- Using the players' reach probabilities and terminal counterfactual
-- values, computes their cfvs at all states of the lookahead.
-- @local
function Lookahead:_compute_cfvs()
  for d=self.depth,2,-1 do
    local gp_layer_terminal_actions_count = self.terminal_actions_count[d-2]
    local ggp_layer_nonallin_bets_count = self.nonallinbets_count[d-3]

    self.cfvs_data[d][{{}, {}, {}, {}, {1}, {}}]:cmul(self.empty_action_mask[d])
    self.cfvs_data[d][{{}, {}, {}, {}, {2}, {}}]:cmul(self.empty_action_mask[d])

    self.placeholder_data[d]:copy(self.cfvs_data[d])

    --player indexing is swapped for cfvs
    self.placeholder_data[d][{{}, {}, {}, {}, self.acting_player[d], {}}]:cmul(self.current_strategy_data[d])

    torch.sum(self.regrets_sum[d], self.placeholder_data[d], 1)

    --use a swap placeholder to change {{1,2,3}, {4,5,6}} into {{1,2}, {3,4}, {5,6}}
    local swap = self.swap_data[d-1]
    swap:copy(self.regrets_sum[d])

    self.cfvs_data[d-1][{{gp_layer_terminal_actions_count+1, -1}, {1, ggp_layer_nonallin_bets_count}, {}, {}, {},
{}}]:copy(swap:transpose(2,3))
  end

end

--- Updates the players' average counterfactual values with their cfvs from the
-- current iteration.
-- @param iter the current iteration number of re-solving
-- @local
```

```
function Lookahead:_compute_cumulate_average_cfvs(iter)
  if iter > arguments.cfr_skip_iters then
    self.average_cfvs_data[1]:add(self.cfvs_data[1])

    self.average_cfvs_data[2]:add(self.cfvs_data[2])
  end
end

--- Normalizes the players' average strategies.
--
-- Used at the end of re-solving so that we can track un-normalized average
-- strategies, which are simpler to compute.
-- @local
function Lookahead:_compute_normalize_average_strategies()

  --using regrets_sum as a placeholder container
  local player_avg_strategy = self.average_strategies_data[2]
  local player_avg_strategy_sum = self.regrets_sum[2]


  torch.sum(player_avg_strategy_sum, player_avg_strategy, 1)
  player_avg_strategy:cdiv(player_avg_strategy_sum:expandAs(player_avg_strategy))

  --if the strategy is 'empty' (zero reach), strategy does not matter but we need to make sure
  --it sums to one -> now we set to always fold
  player_avg_strategy[1][player_avg_strategy[1]:ne(player_avg_strategy[1])] = 1
  player_avg_strategy[player_avg_strategy:ne(player_avg_strategy)] = 0
end

--- Normalizes the players' average counterfactual values.
--
-- Used at the end of re-solving so that we can track un-normalized average
-- cfvs, which are simpler to compute.
-- @local
function Lookahead:_compute_normalize_average_cfvs()
  self.average_cfvs_data[1]:div(arguments.cfr_iters – arguments.cfr_skip_iters)
end

--- Using the players' counterfactual values, updates their total regrets
-- for every state in the lookahead.
-- @local
function Lookahead:_compute_regrets()
  for d=self.depth,2,-1 do
    local gp_layer_terminal_actions_count = self.terminal_actions_count[d-2]
    local gp_layer_bets_count = self.bets_count[d-2]
    local ggp_layer_nonallin_bets_count = self.nonallinbets_count[d-3]

    local current_regrets = self.current_regrets_data[d]
    current_regrets:copy(self.cfvs_data[d][{{}, {}, {}, {}, self.acting_player[d], {}}])

    local next_level_cfvs = self.cfvs_data[d-1]

    local parent_inner_nodes = self.inner_nodes_p1[d-1]
    parent_inner_nodes:copy(next_level_cfvs[{{gp_layer_terminal_actions_count+1, -1}, {1,
ggp_layer_nonallin_bets_count}, {}, {}, self.acting_player[d], {}}]:transpose(2,3))
    parent_inner_nodes = parent_inner_nodes:view(1, gp_layer_bets_count, -1, self.batch_size,
game_settings.hand_count)
    parent_inner_nodes = parent_inner_nodes:expandAs(current_regrets)

    current_regrets:csub(parent_inner_nodes)

    self.regrets_data[d]:add(self.regrets_data[d], current_regrets)
```

```
    --(CFR+)
    self.regrets_data[d]:clamp(0,  tools:max_number())
  end
end


--- Gets the results of re-solving the lookahead.
--
-- The lookahead must first be re-solved with @{resolve} or
-- @{resolve_first_node}.
--
-- @return a table containing the fields:
--
-- * `strategy`: an AxK tensor containing the re-solve player's strategy at the
-- root of the lookahead, where A is the number of actions and K is the range size
--
-- * `achieved_cfvs`: a vector of the opponent's average counterfactual values at the
-- root of the lookahead
--
-- * `children_cfvs`: an AxK tensor of opponent average counterfactual values after
-- each action that the re-solve player can take at the root of the lookahead
function Lookahead:get_results()
  local out = {}

  local actions_count = self.average_strategies_data[2]:size(1)

  --1.0 average strategy
  --[actions x range]
  --lookahead already computes the averate strategy we just convert the dimensions
  out.strategy = self.average_strategies_data[2]:view(-1, self.batch_size, game_settings.hand_count):clone()

  --2.0 achieved opponent's CFVs at the starting node
  out.achieved_cfvs = self.average_cfvs_data[1]:view(self.batch_size, constants.players_count,
game_settings.hand_count)[{{},1,{}}]:clone()

  --3.0 CFVs for the acting player only when resolving first node
  if self.reconstruction_opponent_cfvs then
    out.root_cfvs = nil
  else
    out.root_cfvs = self.average_cfvs_data[1]:view(self.batch_size, constants.players_count,
game_settings.hand_count)[{{},2,{}}]:clone()

    --swap cfvs indexing
    out.root_cfvs_both_players = self.average_cfvs_data[1]:view(self.batch_size, constants.players_count,
game_settings.hand_count):clone()
    out.root_cfvs_both_players[{{},2,{}}]:copy(self.average_cfvs_data[1]:view(self.batch_size,
constants.players_count, game_settings.hand_count)[{{},1,{}}])
    out.root_cfvs_both_players[{{},1,{}}]:copy(self.average_cfvs_data[1]:view(self.batch_size,
constants.players_count, game_settings.hand_count)[{{},2,{}}])
  end

  --4.0 children CFVs
  --[actions x range]
  out.children_cfvs = self.average_cfvs_data[2][{{}, {}, {}, {}, 1, {}}]:clone():view(-1, game_settings.hand_count)

  --IMPORTANT divide average CFVs by average strategy in here
  local scaler = self.average_strategies_data[2]:view(-1, self.batch_size, game_settings.hand_count):clone()


  local range_mul = self.ranges_data[1][{{}, {}, {}, {}, 1, {}}]:clone():view(1, self.batch_size,
game_settings.hand_count):clone()
  range_mul = range_mul:expandAs(scaler)
```

```
  scaler = scaler:cmul(range_mul)
  scaler = scaler:sum(3):expandAs(range_mul):clone()
  scaler = scaler:mul(arguments.cfr_iters – arguments.cfr_skip_iters)

  out.children_cfvs:cdiv(scaler)

  assert(out.children_cfvs)
  assert(out.strategy)
  assert(out.achieved_cfvs)

  return out
end

--- Generates the opponent's range for the current re-solve iteration using
-- the @{cfrd_gadget|CFRDGadget}.
-- @param iteration the current iteration number of re-solving
-- @local
function Lookahead:_set_opponent_starting_range(iteration)
  if self.reconstruction_opponent_cfvs then
    --note that CFVs indexing is swapped, thus the CFVs for the reconstruction player are for player '1'
    local opponent_range = self.reconstruction_gadget:compute_opponent_range(self.cfvs_data[1][{{}, {}, {}, {}, 1,
{}}], iteration)
    self.ranges_data[1][{{}, {}, {}, {}, 2, {}}]:copy(opponent_range)
  end
end
```