МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

МАТЕМАТИЧНИЙ ФАКУЛЬТЕТ
Кафедра комп'ютерних наук

КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

на тему: **«ДОСЛІДЖЕННЯ ЕФЕКТИВНОСТІ НАВЧАННЯ НЕЙРОННОЇ МЕРЕЖІ ГРІ В ШАХИ»**

Виконав: студент 4 курсу, групи 6.1220
спеціальності 122 Комп'ютерні науки
(шифр і назва спеціальності)
освітньої програми Комп'ютерні науки
(назва освітньої програми)
Н. А. Долженко
(ініціали та прізвище)
Керівник доцент кафедри комп'ютерних наук, к.т.н. Добровольський Г.А.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)
Рецензент професор кафедри програмної інженерії, к.ф.-м.н., доцент, Кудін О.В.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Запоріжжя 2024

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет математичний

Кафедра комп'ютерних наук

Рівень вищої освіти бакалавр

Спеціальність 122 Комп'ютерні науки
(шифр і назва)

Освітня програма Комп'ютерні науки

ЗАТВЕРДЖУЮ
Завідувач кафедри комп'ютерних наук,
д.т.н., доцент

_____ Шило Г.М.
(підпис)

" 22 " грудня 2023 р.

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТОВІ

Долженко Нікіті Андрійовичу
(прізвище, ім'я та по-батькові)

1. Тема роботи Дослідження ефективності навчання нейронної мережі грі в шахи

керівник роботи Добровольський Геннадій Анатолійович, к.т.н.
(прізвище, ім'я та по-батькові, науковий ступінь, вчене звання)

затверджені наказом ЗНУ від « 21 » грудня 2023 року № 2180-с

2. Строк подання студентом роботи 15.05.2024

3. Вихідні дані до роботи 1. Постановка задачі.
2. Перелік літератури.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)

1. Постановка задачі.
2. Основні теоретичні відомості.
3. Дослідження ефективності навчання нейронної мережі грі в шахи.
4. Реалізація програми, аналіз результатів.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)
презентація

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання 22.12.2023

К А Л Е Н Д А Р Н И Й П Л А Н

№	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1.	Розробка плану роботи.	05.01.2024	
2.	Збір вихідних даних.	10.01.2024	
3.	Обробка методичних та теоретичних	17.02.2024	
	джерел.		
4.	Розробка першого та другого розділу.	17.03.2024	
5.	Розробка третього розділу.	17.04.2024	
6.	Оформлення та нормоконтроль	17.05.2024	
	кваліфікаційної роботи бакалавра.		
7.	Захист кваліфікаційної роботи.	22.06.2024	

Студент _____ Н. А. Долженко
(підпис) (ініціали та прізвище)

Керівник роботи _____ Г.А. Добровольський
(підпис) (ініціали та прізвище)

Нормоконтроль пройдено

Нормоконтролер _____ О.Г.Спиця
(підпис) (ініціали та прізвище)

РЕФЕРАТ

Кваліфікаційна робота бакалавра «Дослідження ефективності навчання нейронної мережі грі в шахи»: 87 с., 31 рис., 4 табл., 47 джерел, 1 додаток.

АЛГОРИТМ, ДАНІ, ЕФЕКТИВНІСТЬ, ГРА, НАВЧАННЯ, НЕЙРОННА МЕРЕЖА, ОПТИМІЗАЦІЯ, ПОКРАЩЕННЯ, ПРОГРАМА, РЕЗУЛЬТАТИ, СТРАТЕГІЯ, ТАКТИКА, ТЕСТУВАННЯ, ТРЕНУВАННЯ, ШАХИ.

Об’єкт дослідження: процес навчання нейронних мереж грі в шахи.

Предмет дослідження: методи та алгоритми машинного навчання, що застосовуються для тренування нейронної мережі.

Мета роботи: покращення ефективності процесу навчання нейронних мереж грі в шахи.

Методи дослідження – експеримент, порівняння.

Кваліфікаційна робота містить огляд нейронних мереж, їх застосування в грі в шахи, а також розробку та тестування програми для навчання нейронних мереж. Практична частина містить етапи проектування та реалізації програми та подальше її тестування. Результати дослідження демонструють покращення ефективності нейронних мереж після оптимізації процесу навчання, а також включать в себе можливість застосування розробленої програми як інструменту для поліпшення гри в шахи.

SUMMARY

Bachelor’s Qualifying Theses «Study of the effectiveness of neural network training in chess»: 87 pages, 31 figures, 4 tables, 47 references, 1 supplement.

ALGORITHM, CHESS, DATA, EFFICIENCY, GAME, IMPROVEMENT, LEARNING, NEURAL NETWORK, OPTIMIZATION, PROGRAM, RESULTS, STRATEGY, TACTICS, TESTING, TRAINING.

Object of the study: the process of training neural networks to play chess.

Subject of research: machine learning methods and algorithms used to train a neural network.

Aim of the study: to improve the efficiency of the process of training neural networks to play chess.

Methods of research – experiment, comparison.

Bachelor’s qualifying paper contains an overview of neural networks, their application in the game of chess, as well as the development and testing of a program for training neural networks. The practical part includes the stages of designing and implementing the program and its further testing. The results of the study demonstrate the improvement of neural networks efficiency after optimization of the training process, and also include the possibility of using the developed program as a tool for improving chess play.

ЗМІСТ

Завдання на кваліфікаційну роботу 2
Реферат 4
Summary 5
Вступ 9
1 Аналіз наявних нейроних мереж, придатних для навчання гри в шахи 12
1.1 Аналіз задачі 12
1.2 Огляд видів навчання нейромереж 13
1.2.1 Навчання з підкріпленням 14
1.2.2 Навчання з учителем 15
1.2.3 Навчання за допомогою програми Stockfish 15
1.2.4 Комбіновані види навчання нейромережі для гри в шахи 16
1.2.5 Навчання з частковим залученням вчителя 16
1.2.6 Навчання без учителя 17
1.2.7 Передавання навчання 17
1.2.8 Активне навчання 18
1.2.9 Навчання з шумом 18
1.2.10 Навчання з аугментацією 19
1.2.11 Навчання з багатозадачністю 19
1.2.12 Навчання з використанням мета-даних 20
1.3 Огляд типів навчених нейромереж 20
1.3.1 Нейромережі переднього проходу 21
1.3.2 Перцептрони та багатошарові перцептрони 21
1.3.3 Радіальні базисні функції 22
1.3.4 Рекурентні нейромережі 23
1.3.5 Конволюційні нейромережі 23
1.3.6 Модулярні нейромережі 24
1.3.7 Довготривалі мережі з короткочасною пам’яттю 25
1.3.8 Генеративно-змагальні мережі 25

1.3.9 Глибокі мережі переконань 26
1.4 Аналіз технологій та інструментів 27
1.4.1 Ігровий двигун Stockfish 28
1.4.2 Бібліотека chess 29
1.4.3 Бібліотека Tensorflow 29
1.4.4 Бібліотека Torch 29
1.4.5 Бібліотека Pandas 30
1.4.6 Бібліотека Pygame 30
1.4.7 Бібліотека Tkinter 30
1.5 Вибір методів та алгоритмів 31
1.5.1 Метод градієнтного спуску 31
1.5.2 Алгоритм Q-навчання 32
1.5.3 Алгоритм Монте-Карло пошуку дерева 32
1.5.4 Буфер повторення 33
1.5.5 Усереднення моделей 34
1.5.6 Ініціалізація ваг 34
1.5.7 Регуляризація 35
1.6 Висновки 35
2 Розробка програмного продукту 37
2.1 Технічне завдання 37
2.2 Розробка архітектури програмного продукту: опис архітектурних компонентів, залежностей та взаємодії між ними 37
2.2.1 Діаграма компонентів 38
2.2.2 Діаграма розгортання 39
2.2.3 Діаграма діяльності 40
2.2.4 Діаграма прецедентів 43
2.3 Опис вхідних даних для навчання нейромережі 45
2.3.1 Вхідні дані для навчання нейромережі на основі історичних ігор 45
2.3.2 Вхідні дані для навчання нейромережі, яка грає сама проти себе 46
2.3.3 Вхідні дані для навчання нейромережі, яка грає проти Stockfish 47
3 Реалізація програми. Аналіз результатів 48

3.1 Реалізаця програми..... 48
3.2 Аналіз результатів навчання нейромереж 64
3.3 Порівняння ефективності навчання нейромереж..... 69
Перелік посилань..... 76
Додаток А Структура проєкту та повний код програми на Github..... 82

ВСТУП

У рамках цієї роботи було проведено дослідження в області штучного інтелекту (ШІ), зокрема, вивчення можливостей нейронних мереж у грі в шахи. Шахи – одна з найдавніших стратегічних ігор, що вимагає глибокого аналізу та стратегічного мислення. Ця гра стала ідеальним полем для дослідження можливостей ШІ, оскільки вона має чітко визначені правила та обмежений простір станів, що дозволяє комп'ютерам ефективно моделювати гру. Шахи також є гарним прикладом типу задачі, яку можна розв'язувати за допомогою алгоритмів пошуку та оптимізації, що робить цю гру ідеальною моделлю для вивчення ШІ.

Актуальність цього дослідження полягає в тому, що, попри значний прогрес в області штучного інтелекту, вивчення його можливостей у контексті таких складних ігор, як шахи, залишається важливим напрямком досліджень. Це дослідження має з'ясувати, чи може простий штучний інтелект навчитися стратегій і тактикам, необхідним для гри на високому рівні, а також може допомогти у розробці більш ефективних алгоритмів для ігор та інших застосувань, де потрібне складне стратегічне мислення. Одним із ключових аспектів цього дослідження є вивчення того, як штучний інтелект можна використовувати не лише для розв'язання складних проблем, але й для розробки нових, ефективніших стратегій і тактик.

Метою цієї роботи було розробити та навчити нейронні мережі грати в шахи, використовуючи різні методи навчання. Було розроблено три різні нейронні мережі, кожна з яких використовує свій власний метод навчання. Перша нейронна мережа була навчена грати в шахи, граючи сама проти себе. Цей метод, відомий як навчання з підкріпленням, дозволяє адаптуватися і вчитися з кожної гри, поступово вдосконалюючи свої стратегії. Друга нейронна мережа була навчена шляхом гри проти програми Stockfish. Це дозволило ШІ пристосуватися до стратегії та тактики алгоритмів Stockfish, який є найбільш

потужним шаховим двигуном. Третя нейронна мережа була навчена на великому наборі даних історичних ігор в шахи. Цей підхід, відомий як навчання з вчителем, дозволив ШІ вивчити досвід успішних шахістів на основі великого об’єму даних про зіграні ігри у минулому. Крім того, було проведено порівняння ефективності різних методів навчання, щоб визначити, який з них найбільш підходить для навчання нейронних мереж грі в шахи.

В результаті цього дослідження було виявлено, що прості нейронні мережі здатні ефективно набувати знань про складні та стратегічні ігри, такі як шахи. Це відкриває можливості для подальшого вдосконалення ШІ в галузі шахів та інших стратегічних ігор. Після навчання ШІ показав здатність конкурувати з досвідченими гравцями.

У цьому дослідженні було використано різні методи навчання, включаючи навчання з підкріпленням, навчання з вчителем і навчання на основі гри проти програми Stockfish. Після використання кожного з цих методів було зроблено висновок, що кожен з цих методів має свої переваги та недоліки, проте усі методи ефективно показали себе у навчанні нейронних мереж різним стратегіям і тактикам гри в шахи.

Це дослідження має велике теоретичне і практичне значення, оскільки воно допомагає краще зрозуміти, як ШІ може бути використаний для розв’язання складних проблем, таких як пошук найкращих комбінацій та ходів для гри в шахи. Результати цього дослідження мають важливе практичне значення, оскільки вони можуть бути використані для вдосконалення алгоритмів ШІ у сфері шахів та інших стратегічних ігор.

На відміну від більш традиційних методів, які зосереджуються на використанні великої кількості обчислень для аналізу всіх можливих ходів, ці методи допомогли створити ШІ, який може адаптуватися до стратегій супротивника і використовувати цю інформацію для покращення своєї гри.

Завдяки цьому дослідженню, ми можемо краще зрозуміти, як ШІ може бути використаний для розв’язання складних проблем, і як він може бути використаний для покращення наших стратегій і тактик в грі в шахи. Це дослідження є важливим кроком в напрямку розробки більш ефективних і

інтелектуальних систем ШІ, які можуть бути використані в різних галузях нашого життя. Загалом, це дослідження показує, що шахи можуть служити потужним інструментом для вивчення та розвитку штучного інтелекту, і що нейронні мережі можуть бути ефективно навчені гри в шахи, використовуючи різні методи навчання.

1 АНАЛІЗ НАЯВНИХ НЕЙРОНИХ МЕРЕЖ, ПРИДАТНИХ ДЛЯ НАВЧАННЯ ГРИ В ШАХИ

1.1 Аналіз задачі

Дослідження містить навчання з використанням трьох різних підходів: навчання з підкріпленням, навчання з вчителем та навчання на основі гри проти програми Stockfish. Кожен з цих підходів має свої особливості та вимоги до процесу навчання та тестування. Після навчання кожної нейромережі проводяться тестування для оцінки їхньої ефективності, а потім аналізуються отримані результати, що дає змогу зрозуміти, які аспекти потребують додаткового вдосконалення.

Основна задача полягає в реалізації методів, які будуть використовуватися кожною з нейромереж. Це включає створення моделей, навчання моделей, асемблювання моделей, прогнозування та оцінки ходу, конвертації до тензорної репрезентації шахової дошки та ходу, збереження моделей, завантаження вже створених моделей для подальшого навчання, створення графічного інтерфейсу для відображення процесу навчання, створення класу буфера повторення для зберігання і повторного використання даних для навчання алгоритму, створення графіків для відображення результатів навчання. Важливою частиною задачі є реалізація специфічних методів, які будуть використовуватися кожною нейромережею окремо, що і буде вливати на різний підхід до навчання нейромереж та призведе до різних результатів після закінчення навчання.

Наприклад, для створення нейромережі, навченою за допомогою методу – навчання з підкріпленням, потрібно створити функції правильної оцінки ходів, щоб нейромережа мала змогу брати висновок про успішність своїх ходів не тільки за статусом виграшу або програшу в грі, але і на основі того, як змінилася ситуація на полі після ходу, та до яких комбінацій у майбутньому може призвести цей хід.

Для створення нейромережі, яка буде навчатися грати проти програми Stockfish треба розробити метод, який буде оцінювати точність ходів за допомогою шахового двигуна, та за допомогою нього робити хід за Stockfish.

Для створення нейромережі, навченою за допомогою методу – навчання з вчителем на основі історичних партій, необхідно підготувати набори вхідних даних, створити методи, які переформатують цей набір даних в один формат, та методи, які ще раз переформатують цей набір в такий тип даних, який може бути прийнятий нейромережею.

1.2 Огляд видів навчання нейромереж

Нейромережі – це математичні моделі, які намагаються імітувати роботу людського мозку для розв’язання складних завдань. Вони здатні аналізувати великі обсяги даних та виявляти складні залежності між ними. Застосування нейромереж включають розпізнавання образів, машинний переклад, голосові системи та багато іншого, включаючи шахи. Нейромережі складаються з великої кількості штучних нейронів, які імітують роботу біологічних нейронів в мозку. Ці нейрони з’єднані між собою та передають сигнали вперед і назад мережею. Кожен нейрон обробляє вхідні сигнали, використовуючи ваги, які визначають важливість кожного вхідного сигналу. Процес навчання нейромережі полягає в тому, щоб визначити оптимальні ваги для кожного нейрона на основі набору даних. Це досягається за допомогою процесу, відомого як градієнтний спуск, який поступово зменшує помилку мережі, коригуючи ваги в напрямку, що зменшує помилку. Окрім градієнтного спуску, існують інші методи оптимізації, такі як стохастичний градієнтний спуск (SGD), RMSprop, Adam, і т.д., які можуть бути використані для тренування нейромережі. Також нейромережі можуть мати різні архітектури, включаючи повністю з’єднані мережі, конволюційні нейромережі (CNN), рекурентні нейромережі (RNN), та інші.

Нейромережі можуть бути навчені вирішувати різноманітні завдання, включаючи класифікацію, регресію, кластеризацію, а також багато інших. Це

робить нейромережі надзвичайно гнучкими та потужними інструментами для аналізу даних [1,2].

1.2.1 Навчання з підкріпленням

Метод навчання з підкріпленням базується на ідеї навчання, в якому модель (агент) вчиться, взаємодіючи з довкіллям і отримуючи винагороду або кару за свої дії. Модель намагається максимізувати очікувану винагороду, навчаючись з кожною взаємодією з оточуючим середовищем. Цей процес повторюється багато разів, і з кожною новим взаємодією агент навчається краще розуміти, які дії призводять до більш позитивних результатів. Метод широко використовується в гральних алгоритмах, де агент навчається грати, виконуючи дії та отримуючи відгук у вигляді виграшів або програшів. Він також використовується в робототехніці для навчання роботів виконувати певні завдання, такі як ходьба або маніпуляція предметами. Однією з ключових особливостей навчання з підкріпленням є його здатність до адаптації до нових ситуацій, оскільки агент навчається шляхом безпосередньої взаємодії з довкіллям та може навчитися впоратися з непередбачуваними або змінюваними умовами [3, 4].

Нейромережа, яка навчається грати в шахи за допомогою методу підкріплення, проходить через багато етапів навчання. Вона починає з випадкових ходів і поступово вивчає, які ходи призводять до перемоги. Цей процес може бути дуже повільним, але з часом нейромережа стає все кращою в грі. Важливим аспектом цього процесу є використання нагороди та покарання. Нейромережа отримує позитивну винагороду за ходи, які призводять до перемоги, і негативну винагороду за ходи, які призводять до поразки. Це стимулює нейромережу робити ходи, які збільшують шанси на перемогу, і уникати ходів, які зменшують ці шанси. Вона аналізує стан дошки після кожного ходу і використовує цю інформацію для оновлення своїх знань про гру.

1.2.2 Навчання з учителем

Навчання з учителем – основний метод машинного навчання, який використовується для розв’язання широкого спектра задач. Відмінність цього методу від інших методів полягає в тому, що він вимагає наявності розмічених даних для навчання. Розмічені дані – це дані, для яких вже відомі правильні відповіді або мітки. Модель, яка навчається з учителем, намагається вивчити відносини між вхідними даними та відповідями, щоб згодом передбачати відповіді для нових, раніше невідомих даних. Це може бути використано для різних задач, включаючи класифікацію, регресію або ранжування. Однією з ключових переваг навчання з учителем є його здатність до точного прогнозування відповідей на основі вхідних даних. Проте, це також означає, що якість навчання значною мірою залежить від якості та кількості доступних розмічених даних, тому більша кількість високоякісних даних зазвичай веде до кращих результатів [5].

Нейромережа, навчена методом навчання з вчителем, використовує велику базу даних історичних шахових партій для вивчення стратегій гри. Кожна партія в цій базі даних містить послідовність ходів та відповідний результат гри. Нейромережа вивчає ці дані та намагається вивчити залежності між ходами та їх результатами. Це дозволяє нейромережі передбачати результат гри на основі поточного стану дошки та можливих ходів.

1.2.3 Навчання за допомогою програми Stockfish

Нейромережа, навчена методом гри проти програми Stockfish, використовує шаховий двигун Stockfish як “вчителя”. Stockfish – це один з найсильніших відкритих шахових двигунів, і він використовується для генерації ходів проти нейромережі під час тренування. Нейромережа аналізує ходи, згенеровані Stockfish, і намагається вивчити та вдосконалити свої власні стратегії гри. Це дозволяє нейромережі адаптуватися до високорівневих стратегій гри, вивчених Stockfish, та розвивати власні унікальні стратегії [6].

1.2.4 Комбіновані види навчання нейромережі для гри в шахи

Нейромережа, яка включає декілька типів навчання, використовує комбінований підхід, що може об’єднувати, наприклад, елементи навчання з учителем і навчання з підкріпленням. У цьому випадку, на початковому етапі вона використовує метод навчання з учителем, де “вчитель” – це програма Stockfish. Кожен хід, згенерований Stockfish, асоціюється з оцінкою шахового двигуна. Нейромережа аналізує ці ходи та їх оцінки, намагаючись вивчити та вдосконалити свої власні стратегії гри.

Після цього, нейромережа переходить до методу навчання з підкріпленням. Використовуючи знання, отримані від Stockfish, нейромережа сама приймає рішення в грі, отримуючи відгук у вигляді винагороди або покарання за кожен зроблений хід. Це дозволяє нейромережі адаптуватися до нових ситуацій в грі та покращувати свої власні стратегії на основі отриманого досвіду.

Комбінований підхід дозволяє нейромережі ефективно вивчати та адаптуватися до високорівневих стратегій гри, вивчених в іграх проти Stockfish, а також розвивати та вдосконалювати свої власні унікальні стратегії в процесі гри.

1.2.5 Навчання з частковим залученням вчителя

Метод поєднує елементи навчання з учителем і навчання без учителя. Метод використовується, коли доступний лише обмежений набір розмічених даних, але є великий набір нерозмічених даних. У цьому випадку, модель початку навчається на розмічених даних за допомогою методу навчання з учителем, а потім використовується для прогнозування міток для нерозмічених даних, які разом з оригінальними розміченими даними, потім використовуються для подальшого навчання моделі. Цей процес може бути повторений кілька разів, доки модель не досягне прийнятного рівня точності. Цей підхід дозволяє використовувати великі обсяги нерозмічених даних, що можуть бути доступні,

та покращує загальну ефективність моделі, особливо в ситуаціях, коли розмічені дані є обмеженими [7].

1.2.6 Навчання без учителя

Метод навчання без учителя передбачає наявність повного набору розмічених даних для тренування моделі на всіх етапах її побудови. Модель намагається знайти структуру в наборі даних без відомих відповідей. Вона виявляє закономірності або структури, які відображаються у вхідних даних, без заздалегідь відомої моделі відповідей. Основна мета навчання без учителя – виявити внутрішню структуру даних. Це може бути використано для різних задач, таких як кластеризація, виявлення аномалій, зменшення розмірності, і т.д. Кластеризація – процес групування подібних прикладів в одну групу або кластер. Кластеризація може бути корисною для сегментації ринку, групування новин або виявлення аномалій. Виявлення аномалій – процес виявлення незвичайних або підозрілих прикладів в даних. Використовується для виявлення шахрайства або вірусів, моніторингу здоров’я. Зменшення розмірності – процес перетворення даних з високої розмірності до низької розмірності, зберігаючи при цьому важливу інформацію. Використовується для візуалізації даних, зменшення шуму або покращення ефективності інших алгоритмів машинного навчання. Однією з ключових переваг навчання без учителя є його здатність працювати з великими обсягами нерозмічених даних [8].

1.2.7 Передавання навчання

Передавання навчання є важливим напрямком в галузі машинного навчання, який дозволяє моделі використовувати знання, отримані під час розв’язання однієї задачі, для розв’язання іншої пов’язаної задачі. Цей метод є особливо корисним, коли доступні дані для нової задачі є обмеженими, але існує велика кількість даних для подібної задачі. Процес передавання навчання включає два основних основних етапи. На першому етапі, модель навчається на

основній задачі, яка має багато даних. На другому етапі, модель адаптується до нової задачі, використовуючи меншу кількість даних. Це може включати в себе “заморожування” деяких шарів моделі та навчання інших шарів на нових даних, або просто використання моделі як вихідної точки та навчання всієї моделі на нових даних. Важливо зазначити, що передавання навчання найкраще працює, коли основна та нова задачі є пов’язаними, тобто якщо задачі занадто різні, модель може не змогти ефективно перенести своє навчання [9].

1.2.8 Активне навчання

Метод активного навчання полягає в оптимізації процесу навчання шляхом вибору найбільш інформативних прикладів для навчання. Цей метод використовується при обмежених ресурсах анотації даних (процесу додавання інформаційних міток до даних). При активному навчанні, модель вибирає, які приклади з набору нерозмічених даних їй слід навчити наступними, щоб найефективніше покращити свою продуктивність. Активне навчання може бути особливо корисним у ситуаціях, коли розмітка даних є дорогою або часом затратною, наприклад, коли розмітка вимагає експертного знання. Використовуючи активне навчання, можна зменшити кількість прикладів, які потребують розмітки, і все ж досягти високої продуктивності моделі [10].

1.2.9 Навчання з шумом

Метод навчання з шумом полягає в додаванні шуму до даних під час навчання моделі. Цей метод використовується для покращення стійкості моделі до варіацій в вхідних даних та запобігання перенавчанню. Шум може бути доданий до вхідних даних або до вихідних міток. Наприклад, при обробці зображень можна додати шум до пікселів зображення, або при класифікації можна додати шум до міток класу. Додавання шуму до даних під час навчання може допомогти моделі краще узагальнювати, оскільки вона вчиться ігнорувати незначні варіації в даних. Метод використовується у тих випадках, коли модель

навчається для прогнозування на нових даних, які можуть мати невеликі відмінності від даних, на яких вона вже була навчена [11].

1.2.10 Навчання з аугментацією

Аугментація даних – це процес збільшення обсягу тренувальних даних шляхом створення модифікованих версій існуючих прикладів. Метод використовується для покращення загальної продуктивності моделі та запобігання перенавчанню, особливо коли доступні дані є обмеженими. Покращення досягається шляхом застосування доменно-специфічних технік до прикладів з тренувальних даних, що створюють нові та різні тренувальні приклади. Аугментація даних є важливим методом у навчанні нейронних мереж, який допомагає збільшити обсяг та різноманітність тренувальних даних. Метод включає в себе створення модифікованих версій існуючих прикладів в наборі даних, що дозволяє штучно збільшити розмір тренувального набору даних [12].

1.2.11 Навчання з багатозадачністю

Навчання з багатозадачністю – метод в якому одна модель навчається вирішувати кілька пов’язаних завдань одночасно. Підхід базується на ідеї, що вирішення кількох завдань одночасно може допомогти моделі краще узагальнити, ніж коли вона навчається вирішувати кожне завдання окремо. Використання MTL (Multi-Task Learning) може привести до покращення точності прогнозування, збільшення ефективності даних та скорочення часу навчання, тому що модель здатна використовувати ту інформацію, яку вже вивчила з попередніх завдань. Завдяки спільному використанню даних модель може використовувати доступні дані більш ефективно, а також потребувати менше часу, необхідного для навчання [13].

1.2.12 Навчання з використанням мета-даних

Навчання з використанням мета-даних – напрямок в машинному навчанні, де моделі навчаються не просто вирішувати задачу, але й вчитися, використовуючи досвід з попередніх задач. Мета-навчання відноситься до алгоритмів навчання, які вчаться від інших алгоритмів навчання, тобто алгоритм, який вчиться якнайкраще комбінувати прогнози від інших алгоритмів машинного навчання. Мета-навчання зазвичай включає навчання моделі на різноманітних завданнях з метою навчання загальнодоступних знань, які можна передати на нові завдання. Мета-навчання, також відоме як алгоритми “навчання навчатися”, це гілка машинного навчання, яка зосереджена на навчанні моделей самостійно адаптуватися і вирішувати нові проблеми з мінімальним або відсутнім людським втручанням. Мета-навчання також відноситься до алгоритмів, які вчаться, як навчатися на ряді пов’язаних завдань прогнозування, що називається багатозадачним навчанням [14].

1.3 Огляд типів навчених нейромереж

Враховуючи різноманітність завдань, які можуть вирішувати нейромережі, важливо зазначити, що існує багато різних типів нейромереж, кожна з яких має свої унікальні характеристики та застосування. Ці типи нейромереж відрізняються за архітектурою, методами навчання, типами вхідних даних, які вони обробляють, та завданнями, які вони вирішують. Хоча всі нейромережі мають загальну мету – виявлення шаблонів в даних та використання цих шаблонів для вирішення завдань – спосіб, яким вони досягають цієї мети, може значно відрізнятися. Розглянемо декілька типів нейромереж, та яке застосування вони можуть мати безпосередньо при навчанні гри в шахи.

1.3.1 Нейромережі переднього проходу

Нейромережі переднього проходу є одним з найпростіших типів нейромереж. Вони були першим типом штучних нейромереж, що були створені, і вони є основою для більшості архітектур нейромереж. В переднього проходу нейромережі, інформація рухається лише в одному напрямку, тобто сигнали проходять через мережу від вхідного шару до вихідного шару без циклів або петель. Ці мережі складаються з трьох основних типів шарів: вхідного, який приймає вхідні дані, прихованого, який обробляє вхідні дані та вихідного, який виводить результат. Кожен шар складається з вузлів або нейронів, і кожен вузол в одному шарі з'єднаний з кожним вузлом в наступному шарі. Ці з'єднання не є симетричними, тобто вони мають напрямок, що рухається від вхідного шару до вихідного. Нейромережі переднього проходу охоплює два етапи: прямий прохід, який обчислює прогнози та відповідну вартість цих прогнозів, та зворотний прохід, який оновлює ваги та зміщення на основі градієнтів, обчислених під час прямого проходу [15]. У контексті шахів, ці мережі можуть бути використані для оцінки позицій на шаховій дошці, використовуючи вхідні дані про розташування фігур.

1.3.2 Перцептрони та багатошарові перцептрони

Перцептрон – це математична або комп'ютерна модель, яка намагається імітувати роботу біологічних нейронів в мозку. Він був запропонований Френком Розенблатом в 1957 році та став однією з перших моделей нейромереж. Перцептрон складається з трьох типів елементів: сигнали, що надходять від давачів, передаються до асоціативних елементів та до елементів, що реагують. Основна математична задача, з якою він здатний впоратися – це лінійне розділення довільних нелінійних множин. Багатошаровий перцептрон є окремим випадком перцептрона Розенблата, в якому один алгоритм зворотного поширення помилки навчає всі шари. Він використовує нелінійну функцію активації, зазвичай сигмоїдальну. Число шарів, які навчають, більше одного.

Сигнали, що надходять на вхід, та одержувані з виходу не бінарні, а можуть кодуватися десятковими числами. Багатошаровий перцептрон буде володіти функціональними перевагами в порівнянні з перцептроном Розенблата лише в тому випадку, якщо у відповідь на стимули буде виконана не просто якась реакція, а виразиться у підвищенні ефективності вироблення таких реакцій [16]. Перцептрони можуть бути використані для оцінки позицій на шаховій дошці. Вони можуть навчитися асоціювати певні позиції з певними оцінками, використовуючи вхідні дані про розташування фігур. Багатошарові перцептрони можуть виявляти складні шаблони та закономірності в даних, що може бути корисним для виявлення стратегій та тактик в шахах. Також, вони можуть бути навчені з учителем, що означає, що нейромережа може навчитися на основі вже відомих ходів та стратегій.

1.3.3 Радіальні базисні функції

Мережа радіальних базисних функцій є штучною нейронною мережею, яка використовує радіальні базисні функції як функцію активації. Виходом мережі є лінійна комбінація радіальних базисних функцій входу та параметрів нейрона. Мережі радіальних базисних функцій застосовуються в апроксимації функцій, прогнозуванні часових рядів, задачах класифікації. Завдяки гнучким умовам на форму функції активації, RBF мережі є універсальними апроксиматорами на компактному просторі, тобто мережа RBF з достатньою кількістю прихованих нейронів може апроксимувати будь-яку неперервну функцію на замкненій обмеженій множині з довільною точністю. У контексті шахів, така нейромережа здатна використовувати функції апроксимації, які оцінюють позиції на шаховій дошці. Нейромережа здатна до інтерполяції, що може використовуватися для моделювання переходів між різними стратегіями гри. Внаслідок того, що радіально базисні функції використовуються для управління нелінійними системами, навчання грі в шахи може бути ефективним, оскільки саме шахи можна розглядати як нелінійну систему, яка має велику кількість можливих станів та переходів [17].

1.3.4 Рекурентні нейромережі

Рекурентні нейронні мережі (RNN) – це клас штучних нейронних мереж, у якому з’єднання між вузлами утворюють граф орієнтований у часі, що дозволяє нейромережі проявляти динамічну поведінку в часі. На відміну від нейронних мереж прямого поширення, РНМ можуть використовувати свою внутрішню пам’ять для обробки довільних послідовностей входів. Рекурентні нейронні мережі відрізняються тим, що вони мають пам’ять, яка зберігає інформацію про те, що було оброблено в попередніх кроках. Рекурентні нейронні мережі, призначені для роботи з послідовними даних, зберігають інформацію про попередні кроки та використовують її для складнішого аналізу. Ця мережа ефективна для обробки послідовності ходів у шахах, оскільки вона здатна зберігати інформацію про передні ходи [18, 19].

1.3.5 Конволюційні нейромережі

Конволюційні нейромережі (CNN) – це клас глибоких штучних нейронних мереж, які в основному використовуються для обробки зображень. Конволюційні нейромережі містять три основні типи шарів: конволюційний, шар субдискретизації та повнозв'язний. Завданнями цих шарів є виявлення патернів на зображенні, їх стиснення для підвищення обчислювальної ефективності та віднесення виявлених характеристик до певних класів. Конволюційний шар – це основа CNN. Його завдання – виявлення локальних патернів на зображенні за допомогою операції коеволюції. На відміну від повнозв'язних шарів, де кожен нейрон пов'язаний з кожним вхідним пікселем, у конволюційних шарах кожен нейрон пов'язаний тільки з невеликою областю вхідного зображення. Шар субдискретизації (pooling) використовується для зменшення розмірності вхідного зображення, що підвищує обчислювальну ефективність мережі та контролює перенавчання. Повнозв'язний шар використовується в кінці CNN для висновку, тобто для аналізу на попередніх етапах особливостей та відносить їх до певних класів [20, 21]. У контексті шахів,

ця нейромережа може бути використана для аналізу структури шахової дошки, враховуючи просторові відносини між фігурами.

1.3.6 Модулярні нейромережі

Модулярні нейронні мережі – це група нейронних мереж (що керуються певним посередником. Кожна нейронна мережа слугує модулем і оперує окремими входами для вирішення певних підзавдань із групи завдань, які повинна виконати модулярна нейронна мережа. Посередник приймає вихідні сигнали кожного модуля нейронної мережі, виконує певну їх обробку і створює вихідний сигнал усієї модулярної мережі. Посередник не отримує жодних інших сигналів, окрім сигналів з виходів модулів нейронної мережі. Одна з основних переваг модулярних нейронних мереж полягає в здатності розбивати великі нейронні мережі на менші компоненти, якими легше оперувати. Кількість зв’язків в нейронній мережі надзвичайно швидко зростає при додаванні нових нейронів. Розділення завдання між підмережами у модулярній мережі призводить до того, що кожен модуль працює над частиною усього завдання, а не вся мережа намагається виконати глобальне завдання відразу, що є ефективнішим. В модулярній нейронній мережі кожна підмережа прив’язана до певного типу завдання, що не призводить до спантеличення усієї мережі при надходженні нових даних. Крім того, навчальні дані, що використовуються для кожної підмережі, можуть бути унікальними, що призводить до більшої ефективності навчання. Модулярні нейронні мережі дозволяють ізолювати та управляти окремими частинами мережі [22]. Для навчання гри в шахи, такі нейромережі можуть бути дуже ефективними, оскільки можуть розподілити завдання між різними модулями, такими як оцінка позиції, вибір стратегії та визначення оптимального ходу. Різні аспекти гри в шахи можуть бути навчені одночасно завдяки, тому що кожен модуль може бути навчений незалежно від інших модулів. Завдяки високої адаптації до змін, якщо стратегія або тактика, яку використовує модуль, стає менш ефективною, модуль може бути замінений або перенавчений без впливу на інші модулі.

1.3.7 Довготривалі мережі з короткочасною пам’яттю

Довготривалі мережі з короткочасною пам’яттю (LSTM) – це вид рекурентних нейронних мереж, які спеціально розроблені для роботи з довгими послідовностями. Основна ідея LSTM полягає в тому, щоб запобігти проблемі зникання градієнта, яка є типовою для традиційних рекурентних нейронних мереж. LSTM це досягає за допомогою використання так званих “воріт”, які дозволяють моделі вирішувати, яку інформацію слід зберегти або відкинути. Вхідні ворота – визначають, яку нову інформацію слід зберегти в стані пам’яті. Ворота забування – визначають, яку інформацію з поточного стану пам’яті слід відкинути. Вихідні ворота – визначають, яка частина поточного стану пам’яті слід використовувати для вихідного стану. Шахи є грою, що базується на послідовностях, де кожен хід залежить від попередніх ходів. LSTM володіють унікальною здатністю враховувати контекст інформації, що надходить у вигляді послідовностей, що означає що вони можуть використовувати інформацію про попередні ходи для прийняття рішень про поточний та майбутні ходи. В шахах стратегія часто вимагає планування на кілька ходів вперед. LSTM мають “вбудовану пам’ять”, яка дозволяє їм запам’ятовувати важливу інформацію протягом тривалого часу. Також ці нейромережі добре підходять для навчання з досвіду з метою класифікації, обробки або передбачення часових рядів в умовах, коли між важливими подіями існують часові затримки невідомої тривалості [23, 24, 25]. У контексті шахів, це може означати врахування важливих ходів, які відбулися декілька ходів назад.

1.3.8 Генеративно-змагальні мережі

Генеративно-змагальні мережі (Generative Adversarial Networks, GANs) – це клас алгоритмів штучного інтелекту, що використовуються в некерованому навчанні. Вони реалізовані системою двох штучних нейронних мереж, які змагаються одна з одною в рамках гри з нульовою сумою. Одна мережа генерує кандидатів (генератор), а інша оцінює їх (дискримінатор). Як правило,

генеративна мережа навчається будувати відповідності з латентного простору до певного розподілу даних, тоді як дискримінаційна мережа розрізняє представників справжнього розподілу даних та кандидатів, вироблених генератором. Метою тренувальної мережі є збільшення частоти помилок дискримінаційної мережі, шляхом створення нових синтезованих екземплярів, які повинні походити на представників справжнього розподілу даних [26, 27, 28]. Генератор в GAN може бути використаний для генерації нових стратегій гри в шахи, які можуть бути використані для тренування нейромережі. GANs використовують змагальний процес для навчання, тому у контексті шахів одна мережа може генерувати ходи які виглядають реалістично, але насправді є новими та непередбачуваними. GANs можуть адаптуватися до нових ситуацій, оскільки вони постійно навчаються від своїх помилок, тому це може бути корисним, коли ситуація на дошці постійно змінюється. Так генеративно-змагальні мережі відомі тим, що можуть генерувати високоякісні дані, які важко відрізнити від реальних, тому згенеровані ходи можуть виглядати так, ніби вони були зроблені досвідченим гравцем.

1.3.9 Глибокі мережі переконань

Глибокі мережі переконань – (ГМП, англ. deep belief network, DBN) – це породжувальна графова модель або, інакше, клас глибоких нейронних мереж, що складено з кількох шарів латентних змінних («прихованих вузлів»), зі з’єднаннями між шарами, але не між вузлами всередині кожного шару. При тренуванні на наборі прикладів без керування ГМП може навчатися ймовірнісно відбудовувати свої входи, де шари діють як виявлячі ознак. Після цього етапу навчання ГМП можливо тренувати далі з керуванням для виконання класифікування. Нейромережу можливо розглядати як композицію простих некерованих мереж, таких як обмежені машини Больцмана (ОМБ) або автокодувальники. Така композиція веде до швидкої пошарової процедури некерованого тренування, де контрастове розходження застосовують по черзі до кожної підмережі, починаючи з найнижчої пари шарів. ГМП можуть виявляти

важливі ознаки в даних, що можуть бути корисними для визначення стратегій гри в шахи. Наприклад, вони можуть виявляти патерни в розташуванні фігур на дошці, які вказують на певні стратегії. ГМП є породжувальними моделями, що можуть генерувати нові дані на основі навчання, тобто генерувати потенційно нові ходи або стратегії, можуть використовувати змагальний процес для навчання та адаптуватися до нових ситуацій, оскільки вони постійно навчаються від своїх помилок, тому вони будуть корисні для тих випадків, коли ситуація на дошці постійно змінюється [29].

1.3.10 Самоорганізовані карти

Самоорганізовані карти (Self-Organizing Maps, SOM) – це тип нейронних мереж з некерованим навчанням, які використовуються для конструювання багатовимірного простору в простір з нижчою розмірністю (найчастіше, двовимірний). Вони створюють дискретне представлення вхідних просторів навчальних вибірок, які називаються картою [30]. Самоорганізовані карти можуть виявляти важливі ознаки та шаблони в даних, що можуть бути корисними для визначення стратегій гри в шахи. SOM можуть використовуватися для візуалізації даних, що допомагає зрозуміти структуру та взаємозв’язки між різними елементами даних, що у контексті шахів означає візуалізацію різних стратегій гри або взаємозв’язків між різними ходами. Також SOM можуть використовуватися для кластеризації даних. У шахах це може бути використано для групування подібних стратегій або ходів разом, що може допомогти в ідентифікації та вивченні ефективних стратегій.

1.4 Аналіз технологій та інструментів

Виконуючи аналіз технологій та інструментів, серед сучасних бібліотек, наявних рішень зі створення нейромереж, а також інструментів для роботи з шахами та графічним інтерфейсом, треба звернути увагу на різні аспекти, в тому

числі варіативність використання, щоб мати змогу створити декілька нейромереж, які будуть ефективно навчатися, потребуючи якнайменше ресурсів для обчислення. Не менш важливо представити результати дослідження у наочному форматі, саме тому необхідно обрати інструмент для збору статистичних даних та їх форматуванні в графічний спосіб представлення.

1.4.1 Ігровий двигун Stockfish

Двигун Stockfish був розроблений Марко Костальбою, Джуно Кійскі, Гарі Ліндскоттом і Тордом Ромстадом. Зараз він розвивається і підтримується спільнотою Stockfish.

Проєкт Stockfish розпочався з рушія з відкритим вихідним кодом Glaurung, автором якого є Торд Ромстад. У листопаді 2008 року Марко Костальба форкнув (тобто скопіював репозиторій проєкт зі своїми змінами) код Glaurung 2.1 і представив Stockfish 1.0. Торд і Юна Кііскі приєдналися до проєкту Stockfish, а проєкт Glaurung поступово згас. Тим часом Stockfish швидко став найсильнішим шаховим рушієм з відкритим вихідним кодом, з частими оновленнями кожні кілька місяців. Сьогодні він залишається одним з найсильніших рушіїв у світі. Stockfish є безплатною програмою і поширюється на умовах ліцензії GNU General Public License версії 3 (GPL v3), тобто ця програма є доступною для поширення, продажу, а також використанню її як відправної точки для власних програмних проєктів [31]. Stockfish надає ряд методів, які можуть бути корисними при використанні його як вчителя для нейронних мереж. Насамперед Stockfish використовується для оцінки шахових позицій. Програма надає числову оцінку, яка відображає перевагу білих або чорних. Також Stockfish на основі своїх оцінок може робити власні ходи, що може використовуватися, коли нейромережа грає проти Stockfish. Завдяки тому, що Stockfish використовує свою власну нейромережу під назвою NNUE, оцінка позицій є швидкою, що позитивно впливатиме на швидкість навчання інших нейромереж.

1.4.2 Бібліотека chess

Бібліотека chess в Python – це потужна бібліотека, яка надає широкий спектр функцій для роботи з шахами. Вона є дуже корисною при навчанні нейромереж, адже має можливість створювати дошку з фігурами, та відстежувати положення всіх фігур. Методи цієї бібліотеки дозволяють робити ходи на дошці, перевіряти чи є можливим хід у відповідній позиції, надавати списки усіх можливих ходів, серед яких нейромережа може вибирати найкращий, а також використовувати формати PGN та FEN для запису ігор та позицій [32]. Завдяки протоколу UCI (Universal Chess Interface), який підтримує як Stockfish, так і бібліотека Chess, можна використовувати їх разом для гри в шахи та навчанню нейромереж.

1.4.3 Бібліотека Tensorflow

TensorFlow – це потужна бібліотека Python для машинного навчання, яка широко використовується для створення та тренування нейронних мереж. Ця бібліотека може бути легко інтегрована з іншими бібліотеками Python, такими як numpy та chess, що дозволяє легко обробляти дані та взаємодіяти з шаховими двигунами, у тому числі зі Stockfish. Саме ці дані можуть бути використані для генерації ходів на основі поточної позиції на дошці. TensorFlow має потужні інструменти для оптимізації, такі як автоматичне диференціювання та оптимізатори градієнтного спуску, які можуть допомогти покращити продуктивність та точність нейромережі. TensorFlow підтримує багато типів нейронних мереж, включаючи повнозв’язні, згорткові, рекурентні нейронні мережі та багато інших [33].

1.4.4 Бібліотека Torch

Torch також є бібліотекою Python для машинного навчання. Головними відмінностями від TensorFlow є динамічний граф обчислень, що дозволяє змінювати його в режимі реального часу під час виконання коду [34]. Torch має

більшу гнучкість та інтуїтивність, що часто використовується розробниками в академічних дослідженнях нейромереж. Загалом, обидві бібліотеки є одними з найкращих інструментів для навчання нейромереж та можуть використовуватися разом, тому було обрано використовувати обидві бібліотеки.

1.4.5 Бібліотека Pandas

Pandas – бібліотека Python, яка надає широкий спектр функцій для обробки та аналізу даних. Однією з ключових особливостей Pandas є її здатність створювати графічне представлення даних. Pandas полегшує створення графіків, включаючи гістограми, діаграми розмаху, графіки щільності, діаграми розсіювання, графіки площі, стовпчасті діаграми та багато інших [35]. Ця бібліотека може бути використана, коли великий набір даних, незрозумілий користувачеві, потребує переформатування у більш наочний для користувача, графічний формат.

1.4.6 Бібліотека Pygame

PyGame – це бібліотека Python, яка надає потужні інструменти для створення графічного інтерфейсу та обробки вхідних даних від користувача. У ході дослідження, цю бібліотеку було використано для відображення дошок з шаховими фігурами, а також стрілок, які показують останній хід. Це дозволяє користувачу більш продуктивно спостерігати за станом ігор та якістю навчання нейромереж [36].

1.4.7 Бібліотека Tkinter

Tkinter – це стандартна бібліотека Python для створення графічного інтерфейсу користувача (GUI). Вона надає інструменти для створення вікон, кнопок, текстових полів та інших елементів інтерфейсу, що дозволяє легко створювати додатки з візуальним інтерфейсом. У дослідженні, ця бібліотека була використана для створення графічного інтерфейсу для оцінки ефективності

навчання моделей [37].

1.5 Вибір методів та алгоритмів

Для створення нейромереж, які навчаються грати в шахи, можна використовувати різні методи та алгоритми. Ці методи направлені на покращення ефективності навчання, запобіганню перенавчанню, або зменшеню часу, необхідного для навчання моделей. В цьому дослідженні було проаналізовано декілька алгоритмів, які можуть бути використані при створенні нейромереж.

1.5.1 Метод градієнтного спуску

Метод градієнтного спуску є основним алгоритмом оптимізації, який використовується для мінімізації функції втрат при тренуванні нейронних мереж. Цей метод працює шляхом ітеративного оновлення параметрів моделі (ваги та зсуви) в напрямку, що мінімізує функцію втрат. Градієнт функції втрат вказує напрямок найшвидшого зростання значення функції, тому при відніманні градієнт від поточних параметрів, починається рух в напрямку найшвидшого зменшення функції втрат [38].

У цьому досліджені, було використано цей метод завдяки бібліотеці TensorFlow. Завдяки тому, що при компіляції моделі було передано параметр “adam”, модель використовує метод градієнтного спуску як оптимізатор для тренування: model.compile(optimizer='adam', loss='mean_squared_error'). Оптимізатор ‘adam’ є варіацією методу градієнтного спуску, який адаптивно налаштовує швидкість навчання для кожного параметра моделі. Під час процесу навчання, TensorFlow автоматично обчислює градієнти функції втрат щодо параметрів моделі, а потім використовує оптимізатор для оновлення цих параметрів. Цей процес повторюється для кожного батчу даних в навчальному наборі даних до тих пір, поки модель не буде навчена [39].

1.5.2 Алгоритм Q-навчання

Алгоритм Q-навчання є одним з основних методів навчання з підкріпленням, який дозволяє агенту вивчити оптимальну стратегію поведінки в заданому середовищі. Алгоритм базується на ідеї оцінки якості дій, які виконує агент, з метою максимізації сумарної винагороди. Основні кроки алгоритму Q-навчання полягають в ініціалізації Q-таблиці нулями для кожної пари стан-дія, спостереженні за поточним станом та виборі дії випадковим чином або на основі деякої стратегії дослідження, виконанні дії та отриманні негайної винагороди та наступного стану. Потім відбувається оновлення запису Q-таблиці для поточного стану та дії, використовуючи формулу:

$$Q(s, a) = r + \gamma * \max Q(s', a'),$$

де r – це негайна винагорода, γ – це коефіцієнт дисконтування, s’ – це наступний стан, a’ – будь-яка можлива дія з s’.

Потім призначається наступний стан як поточний стан та процес повторюється до досягнення цільового стану або до завершення епізоду. Цей процес повторюється для багатьох епізодів до тих пір, поки Q-таблиця не збіжиться до оптимальної стратегії [40, 41].

1.5.3 Алгоритм Монте-Карло пошуку дерева

Алгоритм Монте-Карло пошуку дерева (MCTS) є важливим методом в галузі штучного інтелекту, який використовується для розв’язання задач, що включають велику кількість можливих ходів, таких як шахи. Він працює шляхом випадкового вибору ходів і оцінки результатів цих ходів. Основна ідея MCTS полягає в інкрементній побудові дерева пошуку, використовуючи симуляції багатьох випадкових графів. Ці симуляції виконуються до тих пір, поки не буде досягнуто термінального стану або попередньо визначеної глибини. Результати цих симуляцій потім пропагуються вгору дерева, оновлюючи статистику вузлів,

відвіданих під час гри, включаючи кількість візитів та відношення перемог. Під час пошуку MCTS динамічно збалансовує дослідження та експлуатацію. Алгоритм вибирає дії, враховуючи як експлуатацію дій з високими відношеннями перемог, так і дослідження невивчених або менш досліджених дій [42].

MCTS був успішно впроваджений в різних областях, включаючи шахи, покер та відеоігри та досяг дивовижної продуктивності в багатьох складних сценаріях гри, часто перевершуючи людське розуміння. MCTS також був розширений та адаптований для вирішення інших проблемних областей, таких як планування, планування та оптимізація.

Однією з видатних переваг MCTS є його здатність обробляти ігри з невідомими або недосконалими даними, оскільки він покладається на статистичне вибіркове дослідження, а не на повне знання стану гри. Крім того, MCTS масштабується та може бути ефективно паралелізований, тому він підходить для розподіленого обчислення та архітектур з багатьма ядрами.

1.5.4 Буфер повторення

Буфер повторення є важливим компонентом в алгоритмах навчання з підкріпленням. Під час тренування буфери повторення запитують підмножину траєкторій (або послідовну підмножину, або вибірку), щоб “повторити” досвід агента. Основна ідея буфера повторення полягає в тому, що він допомагає стабілізувати процес навчання, зменшуючи кореляцію між вибірками та вирішуючи нестаціонарну природу цільових значень. Це досягається шляхом рандомізації порядку, в якому досвіди використовуються для навчання, ефективно декорелюючи вибірки та забезпечуючи більш стабільне середовище навчання [43, 44]. У досліджені було використано цю технологію шляхом реалізування буфера як простого масиву, де кожен елемент становить собою певний досвід агента. Досвід включав поточний стан (позиції фігур на дошці), виконану дію (хід), отриману винагороду (або штраф за хороший чи поганий хід), наступний стан (як змінились позиції фігур після ходу) та інформацію про

те, чи завершено партію гри в шахи. Цей метод дозволяє нейромережі вчитися з більшого обсягу досвіду, ніж тільки з останнього досвіду, що призводить до більш ефективного навчання.

1.5.5 Усереднення моделей

Усереднення моделей є методом, який дозволяє поєднати прогнози з декількох моделей, створюючи ансамбль моделей. Цей метод використовується для покращення стабільності та точності прогнозування, зменшуючи варіативність та усереднюючи помилки. Метод найчастіше використовується, коли декілька моделей навчаються водночас. Основна ідея усереднення моделей полягає в тому, що використання декількох моделей замість однієї може допомогти зменшити ризик вибору неправильної моделі та покращити загальну продуктивність [45]. У дослідженні цей метод були використано при навчанні декількох моделей водночас, щоб повисити швідкість навчання. Одна група моделей тренувалася, починаючи з ходу за білих проти Stockfish, а інша – з ходу чорних. Завдяки усередненню моделей, було отримано одну модель шляхом приділення моделі ваг на основі продуктивності інших моделей.

1.5.6 Ініціалізація ваг

Ініціалізація ваг є критично важливим етапом при розробці нейронних мереж. Вона використовується для визначення початкових значень параметрів в моделях нейронних мереж перед тренуванням моделей на наборі даних. Якщо ваги не ініціалізовані правильно, це може призвести до проблеми зникного градієнта або проблеми вибухового градієнта [46, 47]. У цьому дослідженні було використано метод ініціалізації ваг Глорота. Цей метод ініціалізації ваг використовується для вузлів, які використовують функції активації Sigmoid або Tanh. За замовчуванням, бібліотеки глибокого навчання, такі як TensorFlow та Torch, використовують ініціалізацію Глорота для слоїв Dense та Conv2d. Після створення кожного слою, можна застосувати ініціалізацію Глорота до його ваг,

що дозволяє забезпечити оптимальні початкові значення для подальшого ефективного навчання моделі.

1.5.7 Регуляризація

Регуляризація в машинному навчанні є методом, який допомагає запобігти перенавчанню моделі. Це досягається шляхом додавання додаткової інформації до моделі, що обмежує складність моделі. Основна мета регуляризації полягає в тому, щоб знайти функцію, сумарна похибка передбачень якої для всіх можливих значень була б мінімальною. Регуляризація вводить штраф за включення зайвих областей функціонального простору, що використовується для побудови моделі та може покращити узагальнення [48]. Існує декілька загальних методів регуляризації:

- L1-регуляризація (Lasso): Цей метод регуляризації вводить штраф, що дорівнює абсолютному значенню величини ваг. Це може призвести до того, що деякі ваги стануть нулем, що робить L1-регуляризацію корисною для вибору ознак.
- L2-регуляризація (Ridge): Цей метод вводить штраф, що дорівнює квадрату величини ваг. Метод зазвичай не зводить ваги до нуля, але робить їх дуже малими.

Дослідження використовує метод L2-регуляризації. Функція utils.clip_grad_norm_(neural_net.parameters(), max_norm=1.0) з бібліотеки Torch обмежує L2-норму градієнтів до вказаного максимального значення, що ефективно впроваджує L2-регуляризацію.

1.6 Висновки

Для дослідження були обрані нейронні мережі, які навчені гри в шахи на основі досвіду гри:

- проти двигуна Stockfish

– проти самої себе
– на основі багатої кількості історичних ігор

Джерелами навчання для нейромереж стали вхідні дані, які потрапляють у буфер, а саме:

– позиції фігур на дошці
– останній хід
– отримана винагорода або штраф
– позиція фігур після ходу
– інформацію, чи завершено поточну гру

Для навчання нейромережі на основі історичних ігор, вся ця інформація міститься в .csv файлі, яка була зібрана за допомогою відкритого доступу до історії ігор найпопулярніших ресурсів для гри в шахи. Інформація для навчання інших нейромереж генерується по ходу гри на основі прогнозувань нейромережі та двигуна Stockfish.

2 РОЗРОБКА ПРОГРАМНОГО ПРОДУКТУ

2.1 Технічне завдання

Основа задача полягає в реалізації методів для успішного створення та навчання трьох нейромереж, за допомогою існуючих бібліотек Python та бази даних історичних ігр в шахи.

Потрібно:

- реалізувати методи для створення, збереження, завантаження, навчання, асамблювання моделей
- реалізувати методи для навчання нейромережі, створеною для оцінки позицій
- реалізувати методи оцінки позицій нейромережею та Stockfish
- реалізувати методи для конвертації до тензорної репрезентації шахової дошки та ходу
- реалізувати метод, який дозволить об'єднати декілька баз історичних ігор в одну, виключивши непотрібні ігри
- реалізувати графічний інтерфейс, а саме: відображення декількох шахових дошок з фігурами, відображення останнього ходу за допомогою стрілок, відображення оцінки позицій для обох сторін
- реалізувати відображення результатів навчання за допомогою графіків
- порівняти результати навчання і зробити висновок про ефективність навчання кожної нейромережі

2.2 Розробка архітектури програмного продукту: опис архітектурних компонентів, залежностей та взаємодії між ними

Для кращого розуміння структури проекту та взаємодії усіх його частин, в ході дослідження було розроблено діаграми компонентів, розгортання,

діяльності та прецедентів. Такі діаграми допомогають візуалізувати складні концепції та процеси при навчанні нейромереж.

2.2.1 Діаграма компонентів

Діаграма компонентів – це візуальне представлення архітектури програмного продукту. Вона відображає основні компоненти системи та їх взаємодію. Компоненти можуть бути будь-якими частинами системи, які виконують певні функції. Залежності між компонентами відображаються за допомогою ліній або стрілок, які показують, як дані або контроль передаються від одного компонента до іншого.

Діаграма компонентів для навчених нейромереж може містити наступні компоненти: “Самонавчання”, “Історія ігор”, “Навчання за допомогою Stockfish” – що відображають назву трьох нейромереж, які були навчені, а також “Навчання”, “Дані для навчання”, “Сховище для моделей”, “Гра в шахи”, “Збережені ігри”, “Обчислення результатів навчання” та “Сховище для результатів навчання”.

Компонент “Історія ігор” отримує дані для навчання з файлу, який містить інформацію про партії та ходи досвідчених шахістів у минулому, а компоненти “Самонавчання” та “Навчання за допомогою Stockfish” генерують дані для навчання на основі прогнозувань нейромережі та ходу партії. Усі нейромережі проходять процес навчання, де функція отримує дані для навчання на вході та повертає навчену модель, яка зберігається у сховищі для моделей. Модель, яка ще не завершила навчання, може проходити навчання знову на іншому наборі даних, поки навчання не завершиться. Діаграма показує, що навчені моделі можуть використовуватися для гри в шахи з користувачем, а збережені ігри можуть бути додані у вигляді даних для навчання нейромережі, яка використовує історію ігор. Для аналізу ефективності навчання, проводиться обчислення результатів навчання, та ці дані зберігаються в сховищі у графічному вигляді.

Описана діаграма компонентів показана на рис. 2.1.

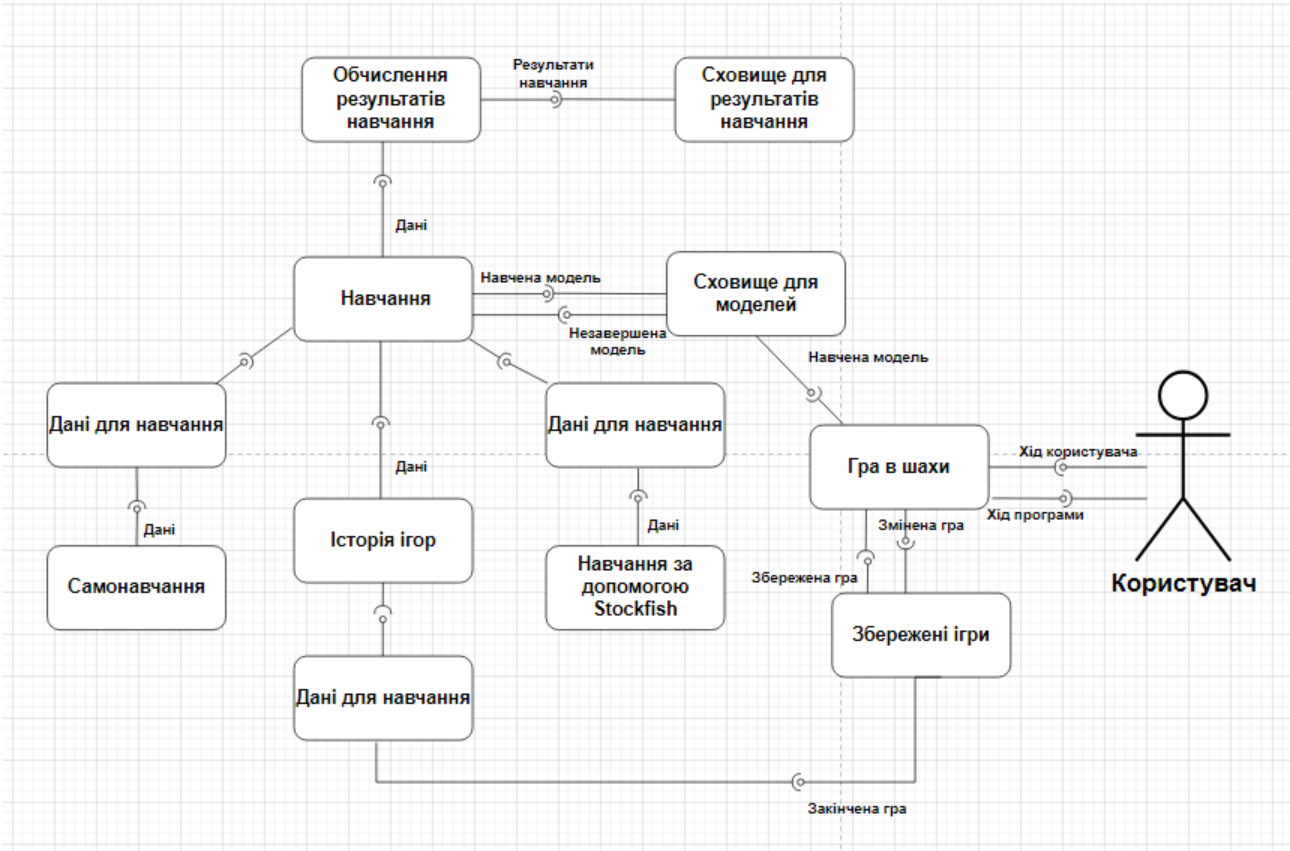

Рисунок 2.1 – Діаграма компонентів

2.2.2 Діаграма розгортання

Діаграма розгортання – це тип діаграми, яка використовується для відображення архітектури розгортання програмного продукту. Вона відображає фізичну конфігурацію програмного забезпечення та апаратного забезпечення, що включає вузли (наприклад, сервери, робочі станції, мобільні пристрої тощо), їх взаємозв’язки та використовується для визначення того, як програмне забезпечення розподіляється та встановлюється в різних частинах системи.

Діаграма містить основні модулі, а також компоненти, які ці модулі містять. Модулі “віртуальна машина для обчислень” містять компоненти “Програма Stockfish”, “Гра в шахи” та “Навчання”. Ця структура означає, що завдання, які потребують великого навантаження, такі як навчання нейромереж, доцільніше усього розмістити на віртуальних машинах, які набагато більш продуктивні та зможуть збільшити швидкість навчання нейромереж. Модуль

“Вебсервер” містить компонент “Клієнтська програма”, яка може використовуватися для гри користувача з нейромережею. Модуль “Сховище даних” містить компоненти “Сховище для моделей”, “Збережені ігри” та “Результати навчання”. Цей модуль відповідає для збереження усіх даних, які потрібні для подальшого навчання моделей, або аналізу їх ефективності.

Описана діаграма розгортання показана на рис. 2.2.

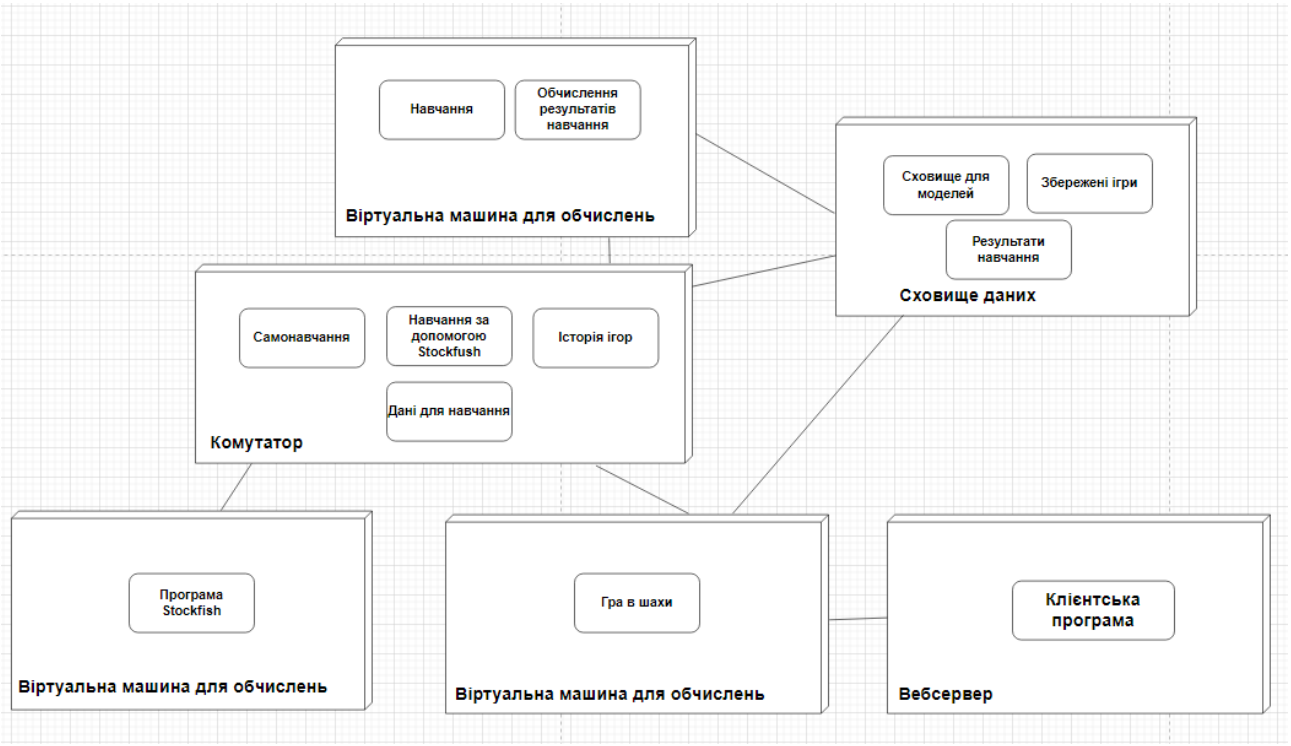

Рисунок 2.2 – Діаграма розгортання

2.2.3 Діаграма діяльності

Діаграма діяльності – це тип діаграми, який візуалізує процес використання та ілюструє потік повідомлень від однієї дії до іншої, тобто показує цілісну роботу системи. Діаграма містить в собі одну або декілька відправних точок, дії, та кінцевий вузол потоку, може містити вузли прийняття рішень, які є розгалуженнями з умовою та кількома варіантами дій та вузли злиття, які є об’єднанням потоків створених вузлом прийняття рішень.

Хоча усі 3 нейромережі мають різний підхід до навчання, вони використовують схожі методи, які можна описати за допомогою діаграми

діяльності. Відправною точкою в діаграмі є нейромережі, під назвами “Самонавчана модель”, “Модель на основі історичних ігор” та “Модель на основі гри проти Stockfish. Спочатку діаграма містить вузол прийняття рішень, який при умові “збережені моделі не знайдені” створює нову модель, а при умові «збережену модель знайдено”, завантажує попередню модель, яка вже проходила навчання. Це необхідно для того, щоб програмний код почав навчання коректно, коли моделей ще немає, і мав змогу продовжити навчання будь-якої моделі, незалежно від того, чи було навчання перерване.

Далі діаграма містить ще один вузол прийняття рішень, який при умові навчання моделі на основі історичних ігор завантажує дані про ходи та партії успішних шахістів з файлу. Наступним етапом є відтворення графічного інтерфейсу для візуалізації процесу навчання. Після цього нейромережа робить хід, відображає його в графічному інтерфейсі та робить оцінку цього ходу. Далі діаграма містить вузол прийняття рішень, який залежно від нейромережі, яка проходить навчання, робить хід за StockFish або також за мережу. Цей хід також має свою оцінку і дані про хід додаються до буфера. Якщо кількості даних в буфері достатньо, проводиться навчання та навчена модель зберігається, а якщо ні – процес повторюється для наступного ходу, допоки буфер не заповниться даними. Процес ходів повторюється до моменту досягнення заданих циклів навчання.

Описана діаграма діяльності показана на рис. 2.3. та рис. 2.4.

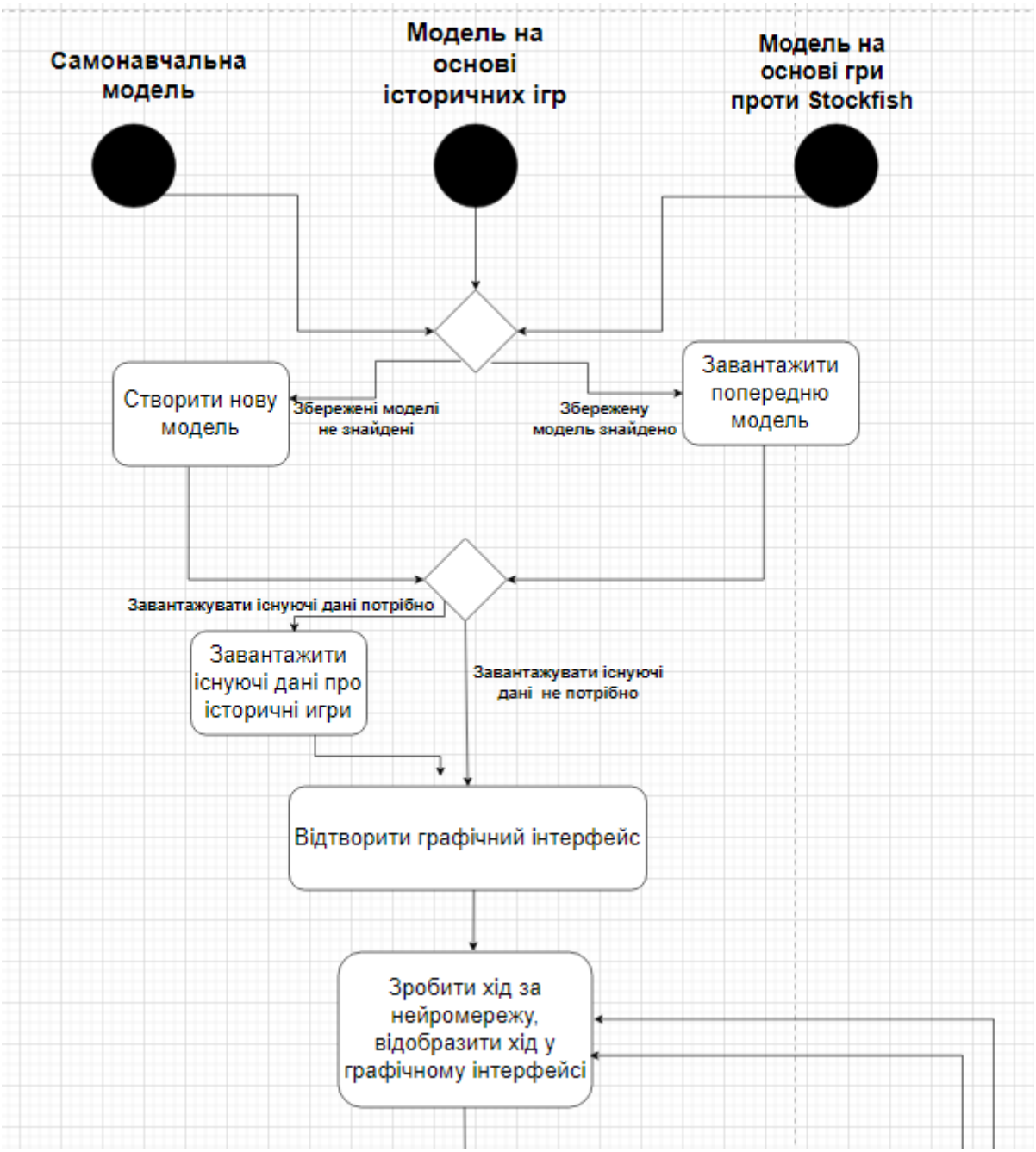

Рисунок 2.3 – Діаграма діяльності (частина 1)

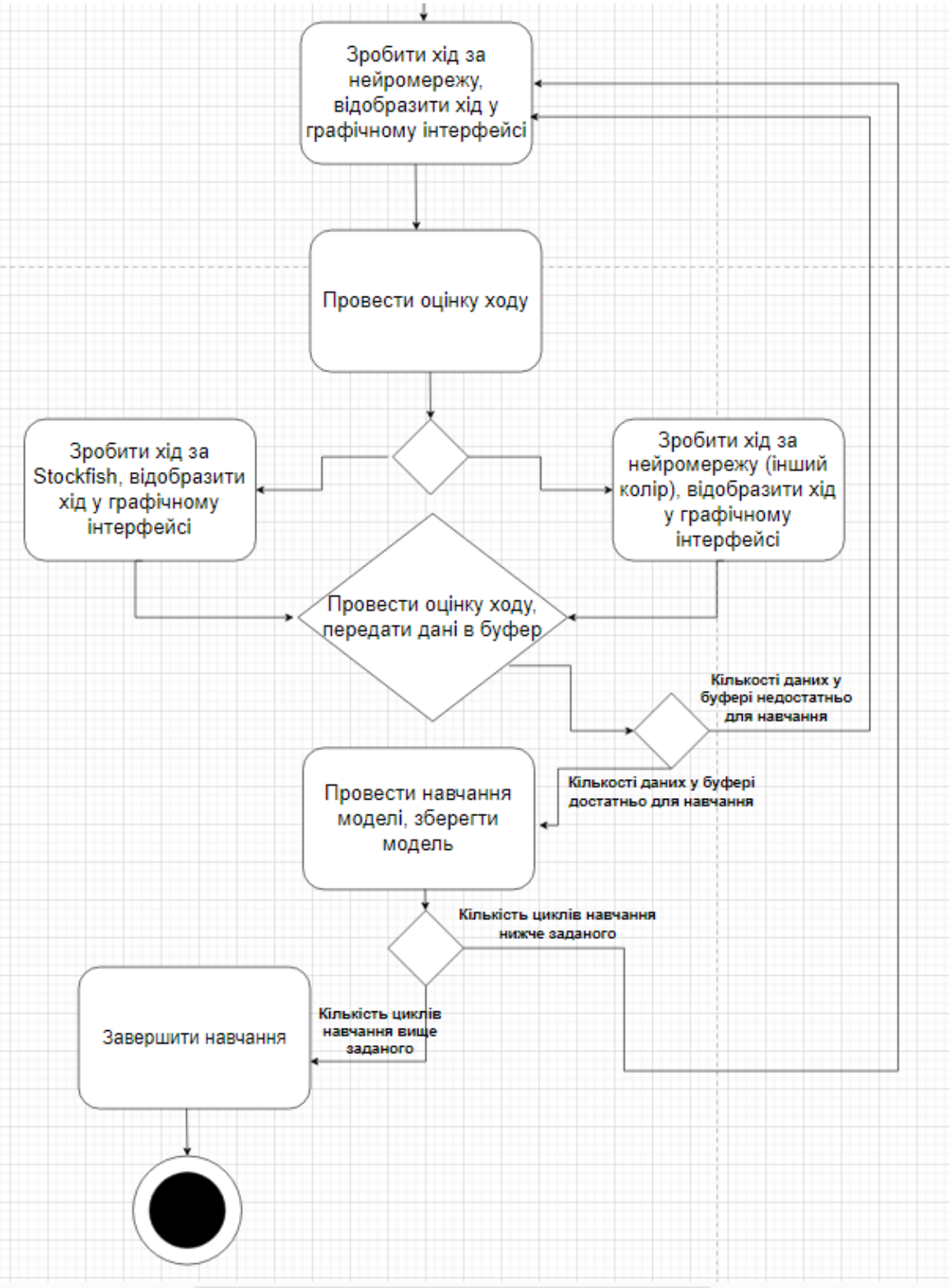

Рисунок 2.4 – Діаграма діяльності (частина 2)

2.2.4 Діаграма прецедентів

Діаграма прецедентів – це тип діаграми, на якій зображено відношення між акторами та прецедентами в системі. Діаграми прецедентів використовуються для відображення елементів моделі варіантів використання.

Вони допомагають візуалізувати, які послуги система надає актору, та як актори взаємодіють з системою.

Система під назвою “Система навчання” надає послуги двом акторам – “Нейромережа” та “Користувач”. Актор “Нейромережа” виконує такі завдання, як збір даних, навчання моделі, збереження моделі, створення графічного інтерфейсу відображення діяльності моделі. Актор “Користувач” може виконувати такі завдання як запуск навчання моделі, перевірка результатів моделі, використання навчаної моделі, спостереження за графічним інтерфейсом моделі.

Описана діаграма прецедентів показана на рис. 2.5.

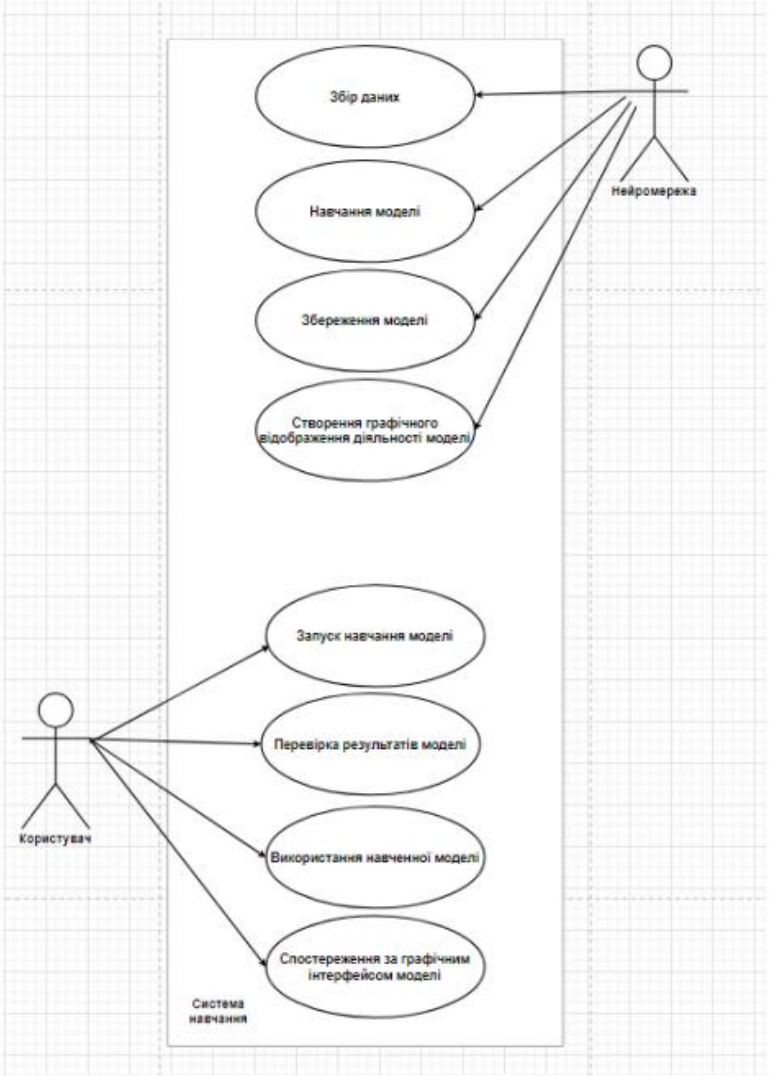

Рисунок 2.5 – Діаграма прецедентів

2.3 Опис вхідних даних для навчання нейромережі

При навчанні нейронних мереж використовуються вхідні дані, які представлені у вигляді числових наборів. Завдяки аналізу великої кількості таких наборів даних, нейронна мережа здатна виявляти складні закономірності та використовувати їх для прогнозування оптимальних рішень. Цей процес навчання містить використання алгоритмів оптимізації, які поступово налаштовують ваги та зміщення в мережі з метою мінімізації помилки прогнозування. Чим більше даних використовується для навчання, тим краще мережа зможе узагальнювати свої прогнози на нових, невідомих їй даних.

2.3.1 Вхідні дані для навчання нейромережі на основі історичних ігор

Нейромережа, навчена завдяки великому набору даних на основі історичних ігор використовує csv файл, тобто файл текстового формату для представлення табличних даних. Кожен рядок таблиці відповідає рядку тексту, який містить поля, розділені комами. Файл містить такі поля, як: “White Elo”, “Black Elo”, “Result”, “Result-Winner”, “Moves”. “White Elo” та “Black Elo” – рейтинг гравців, які зіграли партію. “Result” – це результат гри, представлений у форматі x – y, де x – сторона білих, y – сторона чорних. “Result-Winner” – поле, що містить інформацію про переможця партії (білі або чорні). “Moves” – це рядок, що містить послідовність ходів у партії. Кожен хід розділений пробілом, а після кожної пари ходів вставляється номер пари. Цей файл був отриманий з декількох великих баз даних реальних партій, завдяки інтернет-ресурсу Kaggle – найбільшої у світі спільноти наукових співробітників. Ці бази даних містять набори великої кількості партій реальних гравців з таких сайтів для гри в шахи як chess.com та lichess.org. В ході дослідження за допомогою програмного коду було об’єднано декілька баз даних в один файл, видалено партії, які закінчилися швидко, партії, де гравець здався, не догравши партію до кінця, а також партії, де рейтинг гравців був маленьким, або сильно відрізнявся. Таким чином, було отримано понад 45000 закінчених партії гравців лише високого рейтингу (понад

2000 рейтингу).

2.3.2 Вхідні дані для навчання нейромережі, яка грає сама проти себе

У ході дослідження, спочатку було створено та навчено нейромережу коректно оцінювати ходи за допомогою оцінок двигуна StockFish, а вже потім використовувати цю нейромережу для оцінок ходів нейромережі, яка грає сама проти себе.

В першій нейромережі спочатку свтворується тензор розміром 12x8x8, заповнений нулями. Цей тензор представляє шахову дошку, де 12 шарів відповідають 12 можливим станам кожної клітинки на дошці (6 типів фігур для кожного з двох кольорів), а 8x8 відповідає розміру шахової дошки. Далі нейромережа визначає, який с 12 шарів тензора буде оновлено. Шари 0-5 відповідають білим фігурам (пішак, конь, слон, тура, ферзь, король), а шари 6-11 – чорним фігурам. Також визначаються координати клітинки на дошці, де стоїть ця фігура. Таким чином, тензор заповняється даними, які відображають позиції всіх фігур на дошці. Далі обчислюються втрати, та використовуються алгоритми отпимизіції для оновлення параметрів моделі для зменьшення цих втрат.

Для другої нейромережі сторюється тензор розміру 8x8x13, де вже 13 шарів відповідають 12 можливим станам на дошці та додатковому шару для вказівки, кто зробить наступний хід. Моделі начаються, використовуючи буфер повтору, в який додаються такі дані, як стан позицій на дошці до ходу, останній хід, винагороду за цей хід, стан позицій на дошці після цього ходу, та інформацію про те, чи завершилася партія. Останній хід отримується завдяки інормації про те, якою була клітинка, яку займала фігура до ходу на після нього. Потім у тензорі, який представляє шахову дошку, відбувається пошук цих клітинок та відповідни значення оновлюються на 1, що відповідають вихідній та кінцевій клітинкам ходу. Завдяки бібілотеці chess після кожного ходу можно перевіряти статус завершення партії, тому відповідні дані також оновлюються на значення 1, якщо партія завершена.

2.3.3 Вхідні дані для навчання нейромережі, яка грає проти Stockfish

Ця нейромережа також використовує буфер повтору, який використовувся у нейромережі, яка грає сама проти себе. Головною відмінністю цієї моделі є те, що вона не використовує додаткову нейромережу для оцінки ходів, а оцінку отримає одразу на основі рішення шахового двигуна StockFish. Ходи двигуна StockFish також впливають на навчання, тому модель отримує різні набори даних, включачи накращі від StockFish, та ходи різної сили від самої нейромережі, навідміну від нейронної мережі, яка грає сама проти себе, де на початку навчання майже усі ходи можуть бути слабкими.

3 РЕАЛІЗАЦІЯ ПРОГРАМИ. АНАЛІЗ РЕЗУЛЬТАТІВ

3.1 Реалізаця програми

Програма використовує 3 основних файла, які можуть бути запущені окремо для навчання трьох нейромереж та імпортують основні методи та функції з інших файлів. Ці методи були винесені в окремі файли, тому що використовуються одночасно для навчання декількох нейромереж.

Реалізаця програми у вигляді скороченого псевдокоду з описом для виділення логіки та наочності має вигляд як на рис. 3.1 – 3.11.

Імпорт необхідних бібліотек та модулів

Функція обробки історичних даних ігор

Завантаження даних з CSV файлу

Ініціалізація списку для зберігання даних для навчання моделі

Цикл: для кожного рядку з csv файлу:

– Отримання рядка з ходами гри
– Розбиття рядка на список ходів
– Створення нової шахової дошки
– Встановлення черги ходу
– Обробка кожного ходу в грі
– Перевірка, чи не вийшли за межі списку
– Отримання наступного ходу в форматі SAN
– Парсинг ходу в форматі SAN
– Перевірка, чи є хід допустимим
– Отримання поточного стану дошки
– Отримання дії в форматі тензора
– Отримання наступного стану дошки

Рисунок 3.1 – Перша частина псевдокоду основного файлу для навчання нейромережі на основі історичних ігор

- Перевірка, чи гра закінчена
- Оцінка до ходу
- Робимо хід
- Зміна черги ходу
- Оцінка після ходу
- Використання різниці оцінок як винагороди
- Додавання даних для навчання
- Виведення кількості зібраних даних для навчання
- Навчання моделі на зібраних даних
- Очищення списку з даними після навчання
- Повернення оновленої моделі

Кінець циклу

Вказання файлу з історичними даними ігор

Вказання шляху до моделі нейронної мережі

Спроба завантажити або створити модель нейронної мережі

Якщо модель ще не скомпільована, то компіляція моделі

Завантаження історичних даних ігор з CSV та навчання моделі

Збереження навченої моделі

Обробка перервання навчання та виходу з програми

Рисунок 3.2 – Друга частина псевдокоду основного файлу для навчання нейромережі на основі історичних ігор

Імпорт необхідних бібліотек та модулів

Встановлення початкового номера моделі

Встановлення кількості навчальних циклів.

Визначення шлях до файлу, де зберігаються результати ігор.

Оголошення функції – гра між нейронною мережею та Stockfish (параметри: модель, колір сторони нейронної мережі, чергу для відображення змін дошки, номер гри, буфер для навчання нейронної мережі)

– Створення нової шахової дошки
– Початок циклу (допоки гра не завершиться):

1. Обрахування оцінки до ходу

2. Якщо хід нейронної мережі:

Обробка ходу нейронної мережі

2. Якщо хід шахового двигуна Stockfish:

Обробка ходу двигуна Stockfish

3. Обрахування оцінки після ходу

4. Відображення ходу в графічному інтерфейсі

5. Переформатування стану дошки у формат для навчання нейромережі

6. Перевірка, чи не завершена гра

7. Обчислення винагороди

8. Додання дані до буфера для навчання

– Кінець циклу
– Тренування нейромережі
– Додання даних про ходи у файл для збору статистики
– Повернення моделі

Рисунок 3.3 – Перша частина псевдокоду основного файлу для навчання нейромережі, яка грає проти програми Stockfish

Оголошення функції – запуск гри для нейромережі для гри за певний колір (параметри: колір сторони гри, черга для результатів гри, черга для відображення змін дошки, номер гри, буфер для навчання нейронної мережі):

- Знаходження усіх моделей в теці
- Умова: якщо модель існує – завантажити модель, якщо модель не існує – створити нову модель, зберегти модель
- Виклик функції для гри між нейронною мережею та Stockfish
- Обробка виходу в функції

Створення черг та буфери для управління процесами

Запуск потоку для оновлення дисплея

Знаходження усіх моделей в вказаній теці

Умова: якщо немає моделей – створити нову та зберегти її

Початок циклу навчання:

- Цикл запуску процесів для гри за обидва кольори
- Створення обробнику сигналів
- Очікування завершення процесів
- Умова: якщо є результати в черзі:

  Умова: якщо отримано достатньо моделей:

  1. Запуск функції асемблювання моделей
  2. Збереження моделей
  3. Умова: чи модель є скомпільованою, якщо ні – скомпілювати
  4. Тренування моделі
  5. Видалення старих моделей
  6. Збереження нової моделі
  7. Отримання даних про ігри
  8. Створення графіку процесу навчання

Очищення списку процесів та моделей

Кінець циклу

Обробка перервання навчання та виходу з програми

Рисунок 3.4 – Друга частина псевдокоду основного файлу для навчання нейромережі, яка грає проти програми Stockfish

Імпорт необхідних бібліотек та модулів

Встановлення останнього номеру моделі для зберігання результатів гри

Встановлення кількості циклів навчання

Встановлення шляху до файлу, де зберігаються результати гри

Встановлення шляху до виконуваного файлу Stockfish для використання у ролі шахового двигуна

Запускається шаховий двигун Stockfish

Встановлення шляху до збереженої моделі нейронної мережі

Умова: якщо модель існує – завантажити модель, якщо модель не існує – створити нову модель, зберегти модель

Використання алгоритму оптимізації Adam для навчання моделі

Оголошення функції: отримання оцінки навченої моделі (параметри: дошка, колір нейронної мережі):

– Переформатування позицій фігур на дошці у формат, який підходить до навчання нейромережі
– Перетворення вхідних даних у формат float
– Додання додаткового вимірювання
– Використання моделі для оцінки позиції
– Перетворення результату винагороди у тип даних float
– Повертання даних з функції

Рисунок 3.5 – Перша частина псевдокоду основного файлу для навчання нейромережі, яка грає сама проти себе

Оголошення функції: перетворення шахової дошки в тензор (параметри: дошка, колір нейронної мережі):

– Створення тензора, який представляє стан шахової дошки
– Початок циклу: ітерація по кожній клітинці на дошці

1. Умова: чи є фігура на поточній клітинці

Якщо фігура є, то:

– Визначення кольору фігури
– Конвертація результату в ціле число
– Визначення типу фігури
– Заповнення відповідної клітинки у тензорі значенням 1, що відповідає позиції фігури на дошці.

Якщо фігури немає, то:

– Пропустити цю клітинку

2. Додання додаткової розмірності на початку тензора
3. Повернення тензора, який представляє шахову позицію

Оголошення функції: розрахунок точності оцінки нейронної мережі у порівнянні з оцінкою Stockfish (параметри: оцінка нейронної мережі, оцінка шахового двигуна Stockfish):

– Обчислення різниці між оцінками, наданими нейронною мережею та Stockfish
– Обчислення точності та повернення значення

Рисунок 3.6 – Друга частина псевдокоду основного файлу для навчання нейромережі, яка грає сама проти себе

Оголошення функції: навчання нейронної мережі (параметри: модель нейронної мережі, оптимізатор для моделі, поточна шахова дошка з позицією, колір нейронній мережі, список для збереження точностей):

- Перетворення шахової позиції в формат тензора для моделі
- Аналіз позиції за допомогою Stockfish та отримання оцінки
- Перетворення тензора у формат float і додаємо додатковий вимір
- Використання моделі для оцінки позиції
- Обчислення втрат як квадрат різниці між оцінками
- Очищення градієнтів оптимізатора
- Обчислення градієнтів та оновлення параметрів моделі
- Розрахунок точності та додавання її до списку accuracies.

Рисунок 3.7 – Третя частина псевдокоду основного файлу для навчання нейромережі, яка грає сама проти себе

Оголошення функції: навчання нейромережі позиціям на дошці (немає параметрів):

- Встановлення кількості ігор на навчання
- Ініціалізація списку accuracies для збереження точностей
- Ініціалізація списку average_accuracies для збереження середніх точностей за кожну гру
- Цикл: для кожної гри:
    1. Створення нової шахової дошки
    2. Цикл, доки гра не завершена:
        - Навчання нейронної мережі випадковими ходами
    3. Обчислення середньої точності за поточну гру
    4. Додавання середньої точності в список average_accuracies
    5. Вивід середньої точності та перевірка, чи досягнута кінцева гра
    6. Побудова графіку середньої точності
    7. Збереження графіку
    8. Додавання точностей поточної гри в загальний список
    9. Збереження моделі

    Кінець циклу

Завершення роботи двигуна Stockfish

Рисунок 3.8 – Четверта частина псевдокоду основного файлу для навчання нейромережі, яка грає сама проти себе

Оголошення функції: гра в шахи (параметри: модель нейронної мережі, черга для передачі візуалізації гри, індекс гри, буфер для збереження станів гри для подальшого навчання):

- Створення нової шахової дошки та початкового значення останнього ходу
- Відстеження статистики ходів для кожного кольору
- Цикл: доки гра не завершена:
  1. Отримання ходу від нейронної мережі.
  2. Додавання ходу нейронної мережі в історію ходів для відповідного кольору
  3. Отримання оцінки позиції перед ходом
  4. Оновлення дошки та відображення ходу
  5. Перевірка, чи гра закінчена
  6. Обчислення винагороди як різницю між оцінками до і після ходу
  7. Додавання винагороди в списки відповідного кольору
  8. Перетворення ходу в тензор
  9. Додавання стану, дії, винагороди, наступного стану та ознаки закінчення гри в буфер повтору
  10. Відстеження кількості та правильності ходів для кожного кольору
  11. Вивід повідомлення про завершення гри

  Кінець циклу
- Навчання моделі на основі буфера повтору
- Зчитування результатів гри з файлу
- Запис оновлених результатів гри в файл
- Повернення оновленої моделі

Рисунок 3.9 – П’ята частина псевдокоду основного файлу для навчання нейромережі, яка грає сама проти себе

Оголошення функції: запуск гри для певного кольору (параметри: черга для передачі результатів гри, черга для передачі візуалізації гри, індекс гри, буфер для збереження станів гри для подальшого навчання):

- Перевірка наявності файлів з моделями нейронних мереж.
- Завантаження останньої доступної моделі
- Умова: чи модель є скомпільованою, якщо ні – скомпілювати модель
- Запуск гри
- Передача результату гри в чергу
- Обробка клавіатурного переривання

Оголошення функції: процес навчання гри в шахи (немає параметрів):

- Перевірка наявності файлів з моделями нейронних мереж
- Завантаження останньої доступної моделі
- Умова: чи модель є скомпільованою, якщо ні – скомпілювати модель
- Запуск гри за допомогою завантаженої моделі
- Використання черги візуалізації та буферу повтору
- Передача результатів гри через чергу результатів
- Обробка можливого клавіатурного переривання
- Визначення нової моделі шляхом асемблювання
- Тренування нової моделі.
- Збереження нової моделі.
- Створення графіка суми штрафів та нагород за гру.

Запуск головної програми

Умова: якщо користувач обрав перший режим – запустити навчання моделі гри в шахи, якщо користувач обрав другим режим – запустити навчання моделі оцінки шахових позицій.

Обробка перервання навчання та виходу з програми

Рисунок 3.10 – Шоста частина псевдокоду основного файлу для навчання нейромережі, яка грає сама проти себе

Усі нейромережі використовують функцію, яка прогнозує ходи для нейромережі. Псевдокод цієї функції представлений на рис. 3.11.

Оголошення функції: отримання ходу нейромережі (параметри: модель нейронної мережі, стан дошки, фактор дослідження)

- Отримання списку усіх можливих ходів на поточній дошці
- Створення копії поточного стану дошки
- Ініціалізація порожнього списку для зберігання передбачень моделі для кожного можливого ходу
- Цикл: для кожного можливого ходу:
  1. Застосування поточного ходу до копії дошки
  2. Перетворення стану дошки в тензор, який можна подати на вхід моделі
  3. Перетворення поточного ходу в тензор
  4. Подання тензори стану дошки та ходу на вхід моделі – отримання передбачення
  5. Додання пари (хід, передбачення) до списку передбачень
  6. Скасування останнього хід уна копії дошки.
  7. Сортування ходів за спаданням передбачень
  8. Вибір трьох ходів з найбільшими передбаченнями
  9. Умова: якщо випадкове число менше за фактор дослідження:
     - повертання випадкового ходу із трьох найкращих

     В іншому разі:
     - повертання ходу із найбільшим передбаченням

Рисунок 3.11 – Псевдокод функції отримання ходу нейронної мережі

Важливим аспектом програми було створення графічного інтерфейсу за допомогою бібліотеки Pygame. Цей графічний інтерфейс використовується для візуалізації процесу навчання нейромережі, яка грає проти Stockfish та нейромережі, яка грає сама проти себе. Графічний інтерфейс відображає 8

дошок, тобто 8 моделей, які проходять навчання одночасно, щоб потім завдяки методу асемблювання отримати одну модель. Псевдокод файлу, який містить функції для роботи графічного інтерфейсу, описано на рис. 3.12 – 3.15.

Імпорт необхідних бібліотек та модулів

Оголошення функції: оновлення відображення шахової дошки (параметри: черга для передачі даних для відображення, параметр для визначення – кто бере участь в грі):

– Ініціалізація Pygame
– Встановлення назви вікна
– Встановлення розміру вікна
– Встановлення кольору світлих та темних клітин на дошці
– Створення словника для зберігання зображень фігур
– Створення список для тексту над першим рядком дошок
– Створення списку для тексту під першим рядком дошок
– Сторення списків для фону
– Сторення годиннка для контролю часу

Рисунок 3.12 – Перша частина псевдокоду файлу, який містить функії для роботи графічного інтерфейсу

– Цикл:
  – Оновлення подій Pygame
  – Умова: якщо черга дисплея не порожня:

Цикл: для кожної фігури на дошці:

1. Отримання шляху до зображення та позицію
2. Завантажуємо та масштабуємо зображення

– Визначаємо зміщення по осі X
– Визначаємо зміщеня по осі Y
– Цикл: для кожного рядка на дошці:

1. Цикл: для кожної колонки на дошці:

– Визначення кольору клітинки
– Малювання клітинки
– Цикл: для кожної фігури на дошці:

1. Отримання шляху до зображення та позицію
2. Отримання початкової та кінцевої позиції
3. Умова: якщо хід зроблений конем:
   – Логіка для малювання углової стрілки, яка відображує ход конем
3. Якщо хід зроблений не конем:
   – Намалювати звичайну пряму стрілку

– Додання тексту над та під кожною доскою
– Встановлення шрифту
– Встановлення кольору фону
– Умова: якщо нейромережа грає сама проти себе:

1. Додати відповідний текст

– Якщо нейромережа грає проти Stockfish:

1. Додати відповідний текст

– Оновлення дісплею
– Обмеження частоти кадрів

Рисунок 3.13 – Друга частина псевдокоду файлу, який містить функії для роботи графічного інтерфейсу

Оголошення функції: створення стрілок:

– Створення лінії від початку до кінця
– Визначення вектору напрямку
– Визначення довжини вектора
– Нормалізація вектору напрямку
– Визначення розміру стрілки
– Розрахунок точок для стрілки
– Створення стрілки

Оголошення функції: створення тексту:

– Умова: якщо текстова поверхня не порожня:

1. Створення прямокутнику фону

– Рендер тексту на поверхню
– Отримання прямокутнику тексту
– Копіювання промокутника тексту для фону
– Відображення тексту на екрані
– Повертання текстову поверхню та прямокутник фону

Рисунок 3.14 – Третя частина псевдокоду файлу, який містить функії для роботи графічного інтерфейсу

Оголошення функції: сторення дошки:

– Ініціалізація списку для зберігання інформації про дошку
– Цикл: для всіх клітин на дошці:
    1. Отримаємо фігуру, яка знаходиться на даній клітинці
    2. Умова: якщо на клітині знайшлась фігура:
        – Визначення префіксу кольору фігури
        – Визначення назви фігури
        – Визначення шляху до зображення фігури
        – Визначення позиції фігури на дошці
        – Додання інформції про фігуру до списку
– Кінець циклу
– Умова: якщо був зроблений останній хід
    1. Визначення початкової позиції ходу
    2. Визначення кінцевої позиції ходу
    3. Додання інформації про хід до списку

Повертання з функції інформції про дошку, номер процесу, колір нейронної мережі, оцінку до ходу, дошку, останній хід

Рисунок 3.15 – Четверта частина псевдокоду файлу, який містить функії для роботи графічного інтерфейсу

Повна структура проєкту була описана за допомогою розширення “Draw Folder Structure” для Visual Code та повний код програми був викладений на веб-сервіс для хостингу IT-проектів Github (див. додаток А).

Приклади вигляду графічного інтерфесу у ході процесу навчання показані на рис. 3.16 та рис. 3.17.

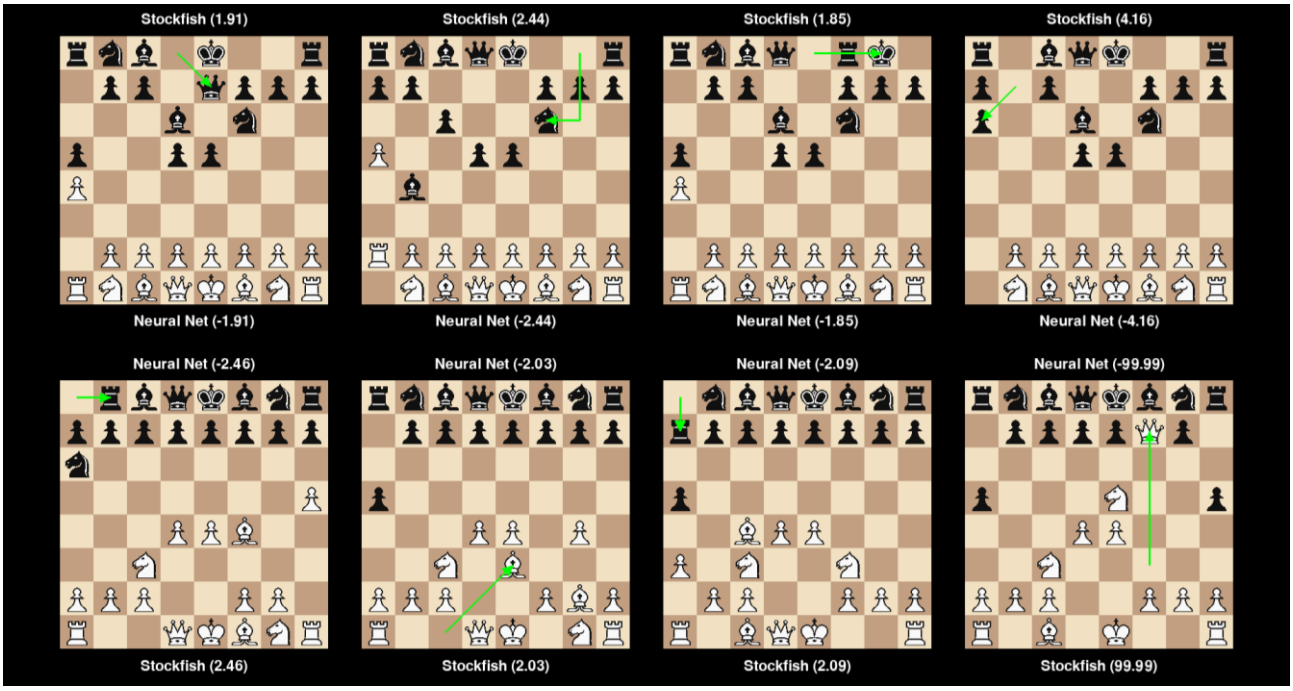

Рисунок 3.16 – Графічний інтерфейс навчання нейромережі у грі проти шахового двигуна Stockfish

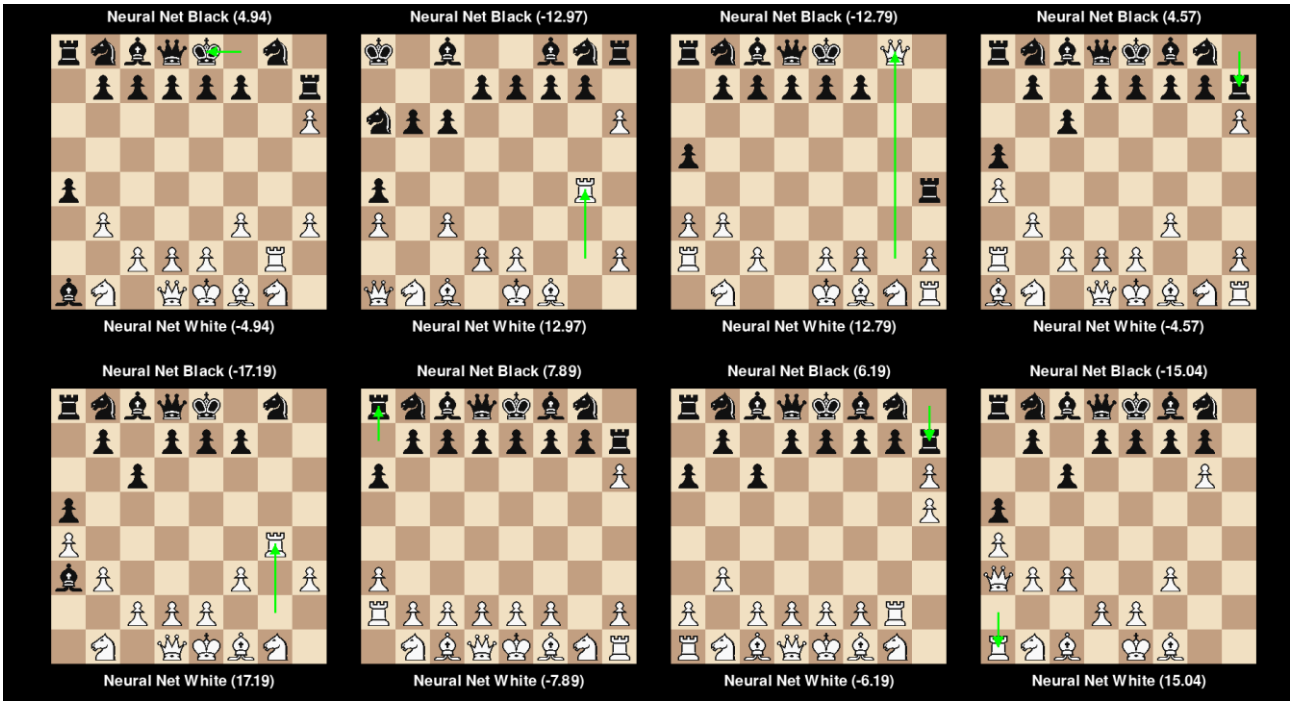

Рисунок 3.17 – Графічний інтерфейс навчання нейромережі у грі проти самої себе

3.2 Аналіз результатів навчання нейромереж

В ході дослідження, був розроблений інтерфейс, який допомагає перевірити ефективність навчаних моделей, шляхом порівняння якості ходів нейромережі з ходами Stockfish графічним методом. Для розробки графічного інтерфейсу програми оцінки моделі було використано бібліотеку Tkinter. За допомогою цієї бібліотеки було створено вікно з кнопками для завантаження моделі, вибору директорії для збереження, запуску та зупинки процесу оцінки, а також текстовими полями для відображення шляху до моделі та директорії збереження. Це забезпечило легку взаємодію користувача з програмою, дозволяючи легко контролювати процес оцінки нейромережі та зберігання результатів. Програма обчислює оцінку позицій на дошці за допомогою шахового двигуна після ходу нейромережі та Stockfish, та виводить результат у вигляді графіку. Графік відображає зміну між оцінкою до ходів Stockfish та нейромережі та після, а також виводить середнє значення та кореляцію між цими змінами. Для обчислення кореляції програма використовує формулу для коефіцієнта кореляції Пірсона. Для покращення точності оцінки, графік показує середні значення для декількох ігор, а кількість ходів для кожної гри є обмеженою. Середня зміна оцінки при ідеальних умовах очікується близько 0, тобто ситуація на полі залишається рівною. Значення близько до 0 буде свідчити про те, що ход моделі був такої ж сили, як і хід найпотужнішого шахового двигуна.

Реалізацію інтерфейсу для оцінки ефективності навчаних моделей описано на рис. 3.18 – 3.21.

Імпорт необхідних бібліотек та модулів

Оголошення функції: отримання ходу нейронної мережі (параметри: модель, шахова дошка):

– Створення списку легальних ходів
– Копіювання дошки
– Створення списку для збереження передбачень
– Цикл для кожного ходу:

1. Зробити хід на копії дошки
2. Перетворення дошки на тензор
3. Перетворення ходу на тензор
4. Отримання передбачення моделі
5. Збереження ходу та передбачення
6. Відміна ходу на копії дошки

– Сортування передбачень
– Повернення ходу з найвищим передбаченням

Рисунок 3.18 – Перша частина псевдокоду файлу, якій містить функції для оцінки ефективності навчання нейромереж

Оголошення функції: оцінка моделі (параметри: модель, шлях до моделі, номер тесту, директорія для збереження, прапорець зупинки):

- Ініціалізація списків покращень моделі та Stockfish
- Створення шахової дошки
- Ініціалізація першого ходу
- Ініціалізація лічильника ходів, списків оцінок моделі та Stockfish
- Виведення номера тесту
- Цикл до завершення гри або зупинки:
    1. Отримання оцінки Stockfish до ходу
    2. Отримання ходу нейронної мережі
    3. Хід моделі на дошці
    4. Отримання оцінки після ходу моделі
    5. Відміна ходу моделі
    6. Обчислення покращення моделі
    7. Додавання покращення до списку
    8. Отримання ходу Stockfish
    9. Хід Stockfish на дошці
    10. Отримання оцінки після ходу Stockfish
    11. Обчислення покращення Stockfish
    12. Додавання покращення до списку
    13. Виведення оцінок до та після ходів
    14. Оновлення середніх покращень та кореляції
    15. Оновлення графіку
- Виведення середніх покращень та кореляції
- Збереження графіку

Рисунок 3.19 – Друга частина псевдокоду файлу, якій містить функції для оцінки ефективності навчання нейромереж

Оголошення функції: оновлення графіку (параметри: лічильник ходів, оцінки моделі, оцінки Stockfish, шлях до моделі, номер тесту, середнє покращення моделі, середнє покращення Stockfish, кореляція):

– Очищення графіку
– Побудова графіку для покращень моделі та Stockfish
– Додавання підписів та легенди
– Встановлення меж Y
– Додавання тексту до графіку
– Оновлення графіку

Оголошення функції: збереження графіку (параметри: шлях до моделі, номер тесту, директорія для збереження):

– Збереження графіку у вказаній директорії

Оголошення функції: завантаження моделі:

– Вибір директорії з моделлю
– Завантаження моделі
– Компіляція моделі за потреби
– Повернення моделі та шляху

Оголошення функції: вибір директорії для збереження:

– Вибір директорії для збереження

Рисунок 3.20 – Третя частина псевдокоду файлу, якій містить функції для оцінки ефективності навчання нейромереж

Оголошення функції: головна функція:

– Внутрішня функція: встановлення моделі
    1. Завантаження моделі та оновлення змінної
    2. Оновлення тексту про шлях до моделі
    3. Оновлення стану кнопки старту
– Внутрішня функція: встановлення директорії збереження
    1. Вибір директорії та оновлення змінної
    2. Оновлення тексту про шлях до директорії
    3. Оновлення стану кнопки старту
– Внутрішня функція: запуск оцінки
    1. Деактивація кнопки старту
    2. Активування кнопки зупинки
    3. Запуск потоку для оцінки моделі
– Внутрішня функція: зупинка оцінки
    1. Встановлення прапорця зупинки
    2. Очікування завершення потоку
    3. Активація кнопки старту
    4. Деактивація кнопки зупинки
– Внутрішня функція: оновлення стану кнопки старту
    1. Перевірка наявності моделі та директорії для збереження
    2. Активація або деактивація кнопки старту

Ініціалізація GUI з використанням Tkinter

Створення змінних для збереження моделі, шляху, директорії, потоку та прапорця

Створення кнопок та елементів GUI для взаємодії користувача

Запуск головного циклу Tkinter

Запуск функції main

Рисунок 3.21 – Четверта частина псевдокоду файлу, якій містить функції для оцінки ефективності навчання нейромереж

Вигляд графічного інтерфесу представлений на рис. 3.22.

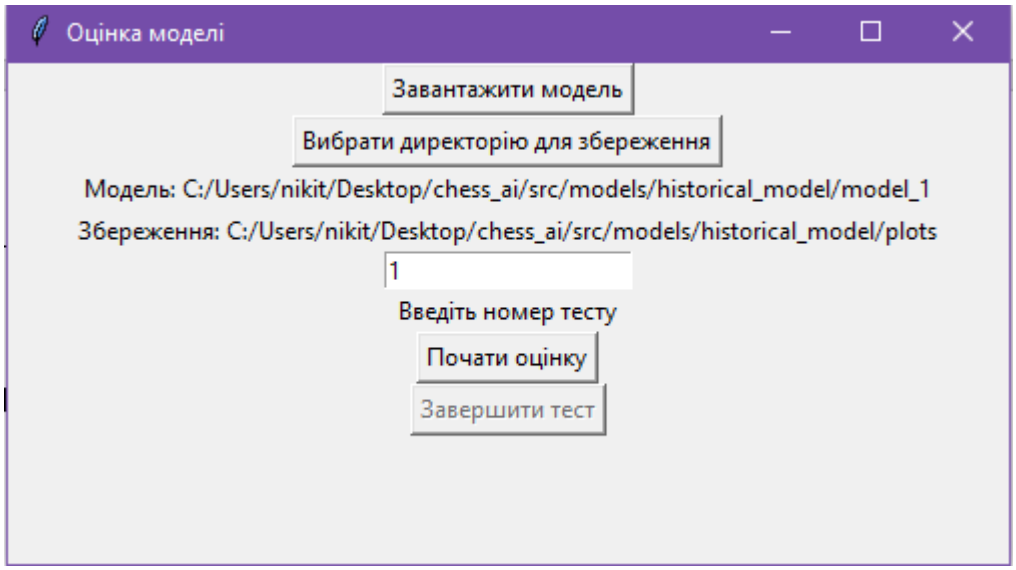

Рисунок 3.22 – Графічний інтерфейс для оцінки ефективності моделей

Для відстеження часу, затраченого на навчання, а також кількості ігор, усі нейромережі зберігали цю інформацію в файл формату txt. Файл містить 4 строки, кожна з яких містить дані про час та кількість ігор у таких способів навчання нейромереж як: навчання за допомогою історичних ігор, навчання за допомогою шахового двигуна Stockfish, навчання моделі, що грає сама проти себе оцінювати шахові позиції та навчання цієї моделі грі в шахи відповідно.

3.3 Порівняння ефективності навчання нейромереж

Одним з важливих етапів навчання нейромережі, яка грає сама проти себе, було використання додаткової нейромережі, яка була навчана оцінювати ходи за допомогою оцінок від двигуна Stockfish. Точність цієї нейромережі була обчислена на основі різниці між оцінками нейромережі та Stockfish шляхом визначення відхилення, де одиниця відхилення дорівнює 20% точності. Результати навчання цієї нейромережі були занесені в табличку (табл. 3.1).

Таблиця 3.1 – Середня точність нейромережі для оцінки ходів

Час навчання (год)	Кількість ігор	Середня точність оцінки (%)
5	215	23.12
15	650	43.49
30	1320	58.62

Останній етап навчання показано у вигляді графіку (рис. 3.23).

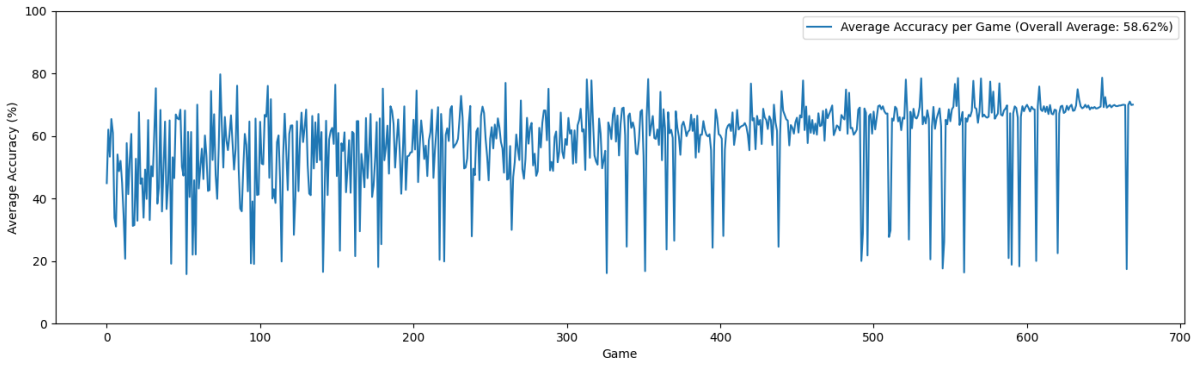

Рисунок 3.23 – Графік навчання нейромережі для оцінки ходів (гра 650 – 1320)

На основі наведених даних можна побачити, що нейромережа ефективно проходить навчання та з часом показує кращі результати. Графік не є рівномірним, і інколи точність нейромережі суттєво знижується в деяких іграх. Це можна пояснити тим, що така гра, як шахи, має надзвичайно велику кількість комбінацій, і оцінка ходу може сильно відрізнятися навіть при дуже схожих позиціях фігур. З цього можна зробити висновок, що нейромережа навчилася доволі точно оцінювати більшість типових позицій, тому може бути використана як допоміжний інструмент для нейромережі, яка грає сама проти себе.

Результати навчання нейромереж, а саме середня зміна оцінки (покращення) після ходу моделі та Stockfish, а також кореляція між змінами, були занесені в таблиці, дані в яких були записані на різних етапах навчання (табл. 3.2, 3.3, 3.4).

Таблиця 3.2 – Результати на початковому етапі навчання

Назва нейромережі	Нейромережа, навчена на основі історичних ігор	Нейромережа, навчена шляхом гри проти Stockfish	Нейромережа, навчена шляхом гри проти самої себе
Час навчання (год)	10	10	5
Кількість зіграних ігор	515	2184	1320
Середня зміна оцінки після ходу моделі	-2.68	-2.91	-3.21
Середня зміна оцінки оцінки після ходу Stockfish	0.01	-0.01	0.01
Кореляція між змінами оцінки після ходу моделі та Stockfish	-0.05	-0.02	-0.07

Таблиця 3.3 – Результати навчання на середньому етапі навчання

Назва нейромережі	Нейромережа, навчена на основі історичних ігор	Нейромережа, навчена шляхом гри проти Stockfish	Нейромережа, навчена шляхом гри проти самої себе
Час навчання (год)	30	30	15
Кількість зіграних ігор	1540	6608	3968
Середня зміна оцінки після ходу моделі	-1.89	-2.15	-2.55
Середня зміна оцінки оцінки після ходу Stockfish	0.02	-0.01	-0.01
Кореляція між змінами оцінки після ходу моделі та Stockfish	0.23	0.27	0.12

Таблиця 3.4 – Результати навчання на завершальному етапі навчання

Назва нейромережі	Нейромережа, навчена на основі історичних ігор	Нейромережа, навчена шляхом гри проти Stockfish	Нейромережа, навчена шляхом гри проти самої себе
Час навчання (год)	60	60	30
Кількість зіграних ігор	3220	13224	7984
Середня зміна оцінки після ходу моделі	-1.23	-1.67	-1.98
Середня зміна оцінки оцінки після ходу Stockfish	0	-0.01	0
Кореляція між змінами оцінки після ходу моделі та Stockfish	0.44	0.38	0.25

Фінальні результати навчання у вигляді графіків представлені на рис. 3.24 – 3.26.

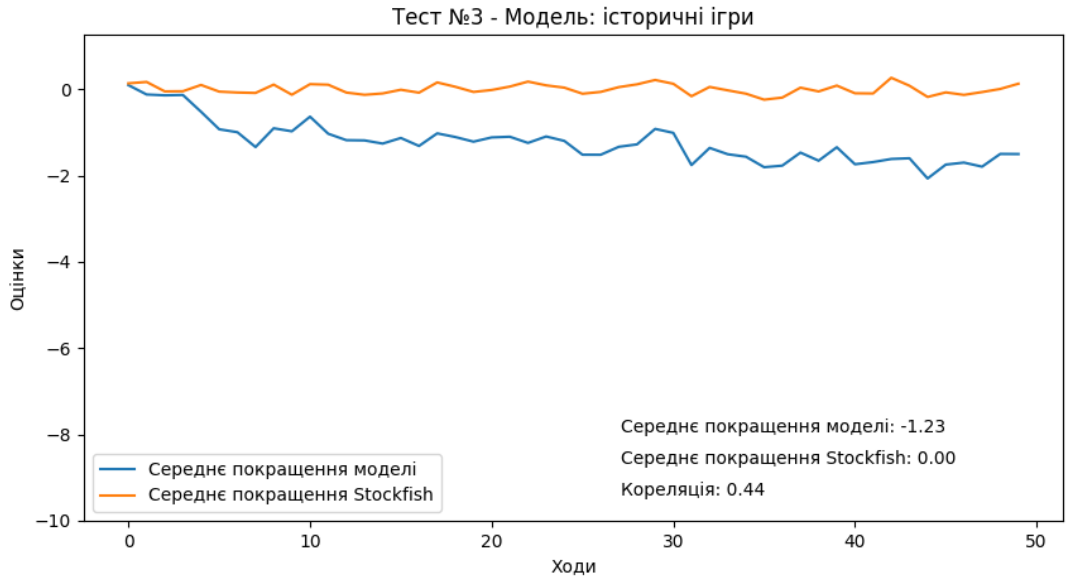

Рисунок 3.24 – Графік середнього покращення (середньої зміни оцінки) моделі та Stockfish для нейромережі, навченої на основі історичних ігор

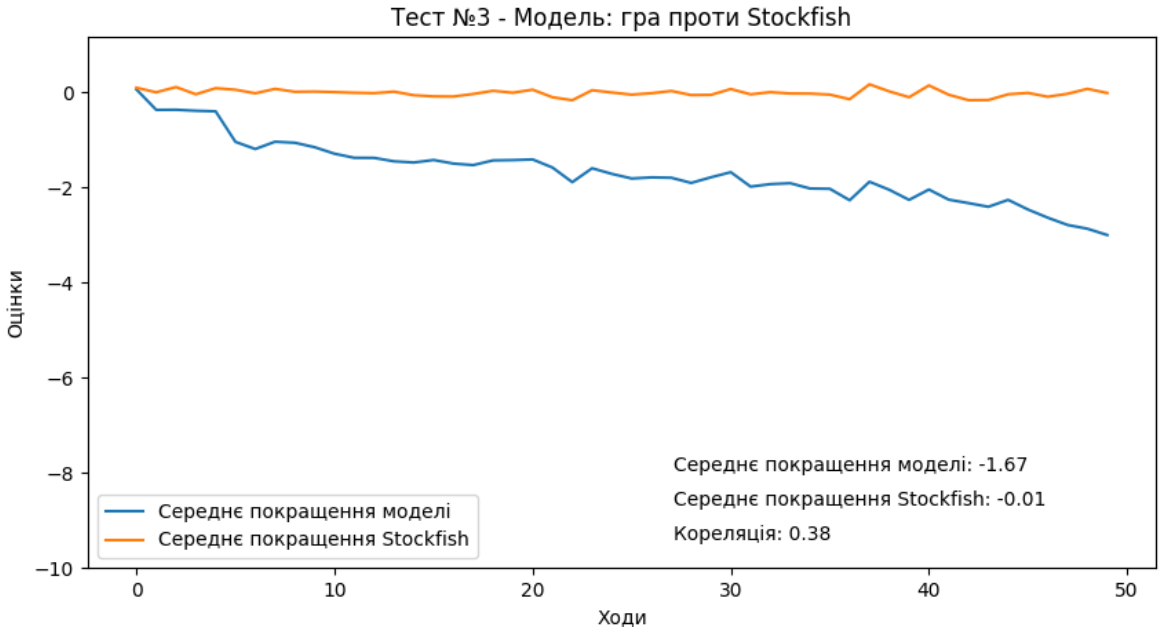

Рисунок 3.25 – Графік середнього покращення (середньої зміни оцінки) моделі та Stockfish для нейромережі, навченої шляхом гри проти Stockfish

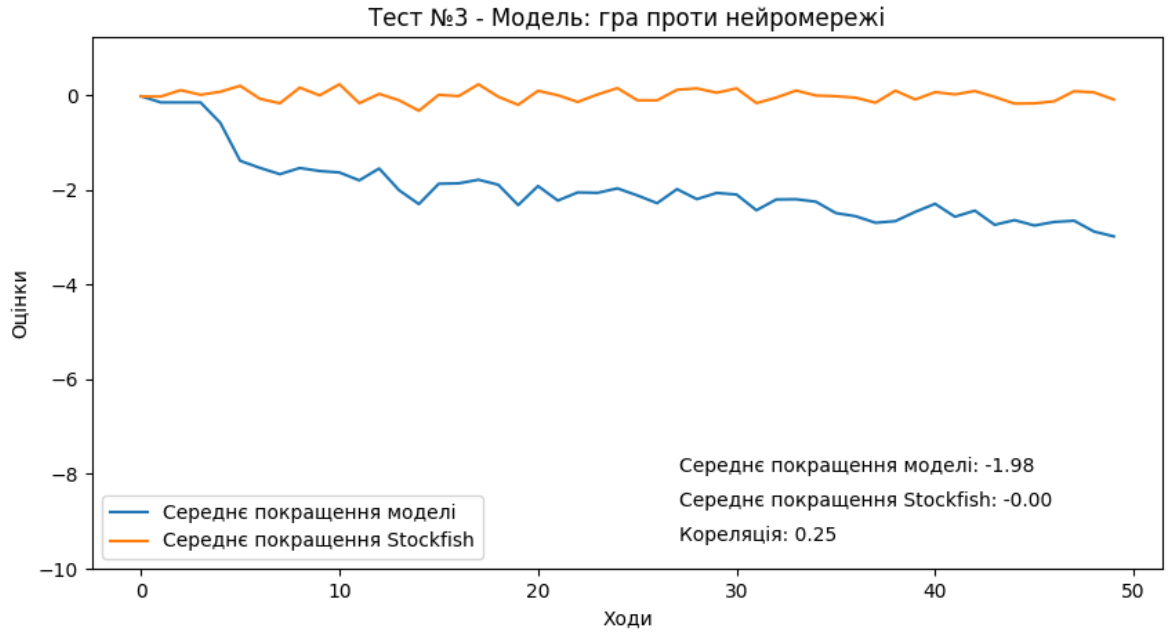

Рисунок 3.26 – графік середнього покращення (середньої зміни оцінки) моделі та Stockfish для нейромережі, навченої шляхом гри проти самої себе

На основі наведених даних можна зробити висновок, що усі нейромережі ефективно навчалися, та з кожним новим тестом показували кращі результати. Кореляція між покращеннями підвищилася з оцінки, близької до 0, до вищої, що

свідчить про позитивну лінійну залежність між покращеннями моделей та шахового двигуна Stockfish.

Найбільш ефективним виявилося навчання нейромережі, що базувалася на аналізі історичних шахових партій. Попри те, що швидкість навчання цієї нейромережі була найнижчою серед усіх, різноманітність даних про партії та ходи досвідчених гравців дозволила їй досягти найкращих результатів. Завдяки цьому, нейромережа могла навчитися складних стратегій і тактик, які використовувалися у реальних іграх.

Нейромережа, яка тренувалася, граючи проти шахового двигуна Stockfish, також продемонструвала хороші результати, провівши значно більше партій, ніж інші нейромережі. Проте, оскільки партії проти Stockfish переважно закінчувалися на користь двигуна, нейромережа не змогла отримати достатньо даних про рівні ситуації на дошці у пізніх стадіях гри. Це чітко видно на графіку, де погіршення якості ходів наприкінці партій було більш вираженим, ніж в інших випадках.

Найменш ефективним виявився процес навчання нейромережі, яка грала сама проти себе. Це пояснюється тим, що така нейромережа потребує значно більше часу для навчання, оскільки спочатку вона повинна навчитися оцінювати ходи, а вже потім використовувати ці оцінки для вибору оптимальних ходів. Відсутність зовнішніх орієнтирів у вигляді даних від реальних ігор чи сильного суперника призвела до більш повільного прогресу навчання нейромережі.

ВИСНОВКИ

Мета роботи, а саме розробка та навчання нейронних мереж грі в шахи, використовуючи різні методи навчання, була досягнена.

Загалом були розроблені три нейромережі: нейромережа, яка навчана на основі історичних ігор; нейромережа, яка навчана шляхом гри проти шахового двигуна Stockfish та нейромережа, навчана шляхом гри проти самої себе. Усі нейромережі пройшли процес навчання, який контролювався за допомогою розробленого графічного інтерфейсу. Навчені нейромережі пройшли процес оцінки якості навчання за допомогою розробленої програми, та були порівняні між собою.

У ході дослідження був використаний стек бібліотек Python для роботи з нейромережами, графіками та графічним інтерфейсом, а саме: chess, Tensorflow, Torch, Pandas, Pygame, Tkinter. Завдяки своєму великому функціоналу, шаховий двигун Stockfish став важливою частиною при навчанні та оцінки якості навчених нейромереж.

Для навчання нейромереж були використані такі методи як: буфер повторення, метод градієнтного спуску, метод усереднення моделей, метод ініціалізації ваг та метод L2 – регуляризації.

Розроблені нейромережі використовували такі види навчання як: навчання з підкріпленням, навчання з вчителем та комбінований тип навчання, що поєднує навчання з підкріпленням та навчання з вчителем.

У ході дослідження були розроблені діаграми компонентів, розгортання, діяльності та прецедентів для кращого розуміння взаємодії компонентів нейромережі між собою та користувачем.

Навчені нейромережі доступні для подальшого навчання для посилення їхньої ефективності.

ПЕРЕЛІК ПОСИЛАНЬ

1. Scikit-learn: machine learning in Python – scikit-learn 1.4.2 documentation. URL : https://scikit-learn.org/stable/ (дата звернення: 17.02.2024).

2. Juergen Schmidhuber. Deep Learning in Neural Networks: An Overview. University of Lugano & SUPSI. Switzerland, 2014. 88 с. URL : https://arxiv.org/pdf/1404.7828 (дата звернення: 18.02.2024).

3. Part 1: Key Concepts in RL – Spinning Up documentation. OpenAI Spinning Up. URL : https://spinningup.openai.com/en/latest/spinningup/rl_intro.html#id2 (дата звернення: 21.02.2024).

4. Kumar Chandrakant. Reinforcement Learning with Neural Network. Baeldung. 2024. URL : https://www.baeldung.com/cs/reinforcement-learning-neural-network (дата звернення: 27.02.2024).

5. Jie Gui, Tuo Chen, Jing Zhang, Qiong Cao, Zhenan Sun, Hao Luo, Dacheng Tao. A Survey on Self-supervised Learning: Algorithms, Applications, and Future Trends. 2015. 20 с. URL : https://arxiv.org/pdf/2301.05712 (дата звернення: 28.02.2024).

6. David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, Timothy Lillicrap, Karen Simonyan, Demis Hassabis. Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm. DeepMind, 6 Pancras Square, London N1C 4AG. 2017. 19 с. URL : https://arxiv.org/pdf/1712.01815 (дата звернення: 28.02.2024).

7. Weak supervision. Wikipedia: веб-сайт. URL : https://en.wikipedia.org/wiki/Weak_supervision (дата звернення: 28.02.2024).

8. Afreen Khalfe. Unsupervised machine learning: Clustering, dimensionality reduction, and anomaly detection techniques. Talent500: веб-сайт. 2023. URL : https://talent500.co/blog/unsupervised-machine-learning-clustering-dimensionality-reduction-and-anomaly-detection-techniques (дата звернення: 28.02.2024).

9. Fuzhen Zhuang, Zhiyuan Qi, Keyu Duan, Dongbo Xi, Yongchun Zhu,

Hengshu Zhu, Hui Xiong, Qing He. A Comprehensive Survey on Transfer Learning. 2019. 31 с. URL : https://arxiv.org/pdf/1911.02685 (дата звернення: 29.02.2024).

10. Burr Settles. Active Learning Literature Survey. University of Wisconsin–Madison. 2010. 67 с. URL : https://burrsettles.com/pub/settles.activelearning.pdf (дата звернення: 29.02.2024).

11. Yang Lu, Yiliang Zhang, Bo Han, Yiu-ming Cheung, Hanzi Wang. Label-Noise Learning with Intrinsically Long-Tailed Data. Fujian Key Laboratory of Sensing and Computing for Smart City, School of Informatics, Xiamen University, Xiamen, China. 2022. 10 с. URL : https://arxiv.org/pdf/2208.09833 (дата звернення: 29.02.2024).

12. Alhassan Mumuni, Fuseini Mumun. Data augmentation: A comprehensive survey of modern approaches. Cape Coast Technical University, P. O. Box DL 50, Cape Coast, Ghana. University of Mines and Technology, P.O. Box 237, Tarkwa, Ghana. 2022. 27 с. URL : https://www.researchgate.net/publication/365378179_Data_augmentation_A_comprehensive_survey_of_modern_approaches (дата звернення: 01.03.2024).

13. Michael Crawshaw. Multi-Task Learning with Deep Neural Networks: A Survey. Department of Computer Science, George Mason University. 2020. 43 с. URL : https://arxiv.org/pdf/2009.09796 (дата звернення: 01.03.2024).

14. Kaivan Kamali. Deep Learning (Part 1) – Feedforward neural networks (FNN). Galaxy Training. 2021. URL : https://training.galaxyproject.org/training-material/topics/statistics/tutorials/FNN/tutorial.html (дата звернення: 01.03.2024).

15. Що таке багатошаровий персептрон (Multilayer Perceptron, MLP) у машинному навчанні? The Transmitted. 2023. URL : https://thetransmitted.com/adlucem/shho-take-mlp-u-mashynnomu-navchanni. (дата звернення: 02.03.2024).

16. Dimitrios Kagkas, Despina Karamichailidou, Alex Alexandridis. Chess Position Evaluation Using Radial Basis Function Neural Networks. 2023. URL : https://www.hindawi.com/journals/complexity/2023/7143943/ (дата звернення: 03.03.2024).

17. Nesma M. Rezk, Madhura Purnaprajna, Tomas Nordström, Zain Ul-

Abdin. Recurrent Neural Networks: An Embedded Computing Perspective. School of information technology, Halmstad University, Sweden. 2020. 33 с. URL : https://arxiv.org/pdf/1908.07062 (дата звернення: 05.03.2024).

18. Robin M. Schmidt. Recurrent Neural Networks (RNNs): A gentle Introduction and Overview. Department of Computer Science, Eberhard-Karls-University Tübingen, Tübingen, Germany. 16 с. URL : https://arxiv.org/pdf/1912.05911 (дата звернення: 06.03.2024).

19. Jiuxiang Gu, Zhenhua Wang, Jason Kuen, Lianyang Ma, Amir Shahroudy, Bing Shuai, Ting Liu, Xingxing Wang, Li Wang, Gang Wang, Jianfei Cai, Tsuhan Chen. Recent Advances in Convolutional Neural Networks. ROSE Lab, Interdisciplinary Graduate School, Nanyang Technological University, Singapore. 38 с. URL : https://arxiv.org/pdf/1512.07108 (дата звернення: 07.03.2024).

20. Keiron O'Shea, Ryan Nash. An Introduction to Convolutional Neural Networks. Department of Computer Science, Aberystwyth University, Ceredigion, SY23 3DB. 11 с. URL : https://arxiv.org/pdf/1511.08458 (дата звернення: 07.03.2024).

21. Louis Kirsch, Julius Kunze, David Barber. Modular Networks: Learning to Decompose Neural Computation. Department of Computer Science, University College London. 13 с. URL : https://arxiv.org/pdf/1811.05249 (дата звернення: 07.03.2024).

22. Alex Sherstinsky. Fundamentals of Recurrent Neural Network (RNN) and Long Short-Term Memory (LSTM) Network. 2018. 43 с. URL : https://arxiv.org/pdf/1808.03314 (дата звернення: 08.03.2024).

23. Ralf C. Staudemeyer, Eric Rothstein Morris. Understanding LSTM – a tutorial into Long Short-Term Memory Recurrent Neural Networks. Faculty of Computer Science, Schmalkalden University of Applied Sciences, Germany. 2019. 42 с. URL : https://arxiv.org/pdf/1909.09586 (дата звернення: 09.03.2024).

24. Benyamin Ghojogh, Ali Ghodsi. Recurrent Neural Networks and Long Short-Term Memory Networks: Tutorial and Survey. Department of Statistics and Actuarial Science & David R. Cheriton School of Computer Science, Data Analytics Laboratory, University of Waterloo, Waterloo, ON, Canada. 2023. 15 с. URL :

https://arxiv.org/pdf/2304.11461 (дата звернення: 10.03.2024).

25. Generative Adversarial Network (GAN). GeeksForGeeks. 2024. URL : https://www.geeksforgeeks.org/generative-adversarial-network-gan/ (дата звернення: 11.03.2024).

26. Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, Yoshua Bengio. Generative Adversarial Networks. Departement d'informatique et de recherche operationnelle, Universite de Montreal, Montreal, QC H3C 3J7. 2014. 9 с. URL : https://arxiv.org/pdf/1406.2661 (дата звернення: 12.03.2024).

27. Antonia Creswell, Tom White, Vincent Dumoulin, Kai Arulkumaran, Biswa Sengupta, Anil A Bharath. Generative Adversarial Networks: An Overview. Cortexica Vision Systems Ltd., London, United Kingdom. 2017. 14 с. URL : https://arxiv.org/pdf/1710.07035v1 (дата звернення: 13.03.2024).

28. Gaudenz Boesch. Deep Belief Networks (DBNs) Explained. Viso.ai. URL : https://viso.ai/deep-learning/deep-belief-networks/ (дата звернення: 14.03.2024).

29. Abhinav Ralhan. Self Organizing Maps. Medium. 2018. URL : https://medium.com/@abhinavr8/self-organizing-maps-ff5853a118d4 (дата звернення: 15.03.2024).

30. Ilya Zhelyabuzhsky. Stockfish chess engine with Python. Stockfish 3.28.0: веб-сайт. URL : https://pypi.org/project/stockfish/ (дата звернення: 17.03.2024).

31. Niklas Fiekas. Chess library with move generation and validation. Chess 1.10.0: веб-сайт. URL : https://pypi.org/project/chess/ (дата звернення: 18.03.2024).

32. TensorFlow. Wikipedia: веб-сайт. URL : https://uk.wikipedia.org/wiki/TensorFlow (дата звернення: 19.03.2024).

33. PyTorch Team. Tensors and Dynamic neural networks in Python with strong GPU acceleration. Torch 2.3.0: веб-сайт. URL : https://pypi.org/project/torch/ (дата звернення: 19.03.2024).

34. The Pandas Development Team. Pandas: powerful Python data analysis toolkit. Pandas 2.2.2: веб-сайт. URL : https://pypi.org/project/pandas/ (дата звернення: 19.03.2024).

35. A community project. Pygame: Python Game Development. Pygame 2.5.2: веб-сайт. URL : https://pypi.org/project/pygame/ (дата звернення: 19.03.2024).

36. tkinter – Python interface to Tcl/Tk. Стандартна бібліотека Python: веб-сайт. URL : https://docs.python.org/uk/3.12/library/tkinter.html (дата звернення: 19.03.2024).

37. Sebastian Ruder. An overview of gradient descent optimization algorithms. Insight Centre for Data Analytics, NUI Galway, Aylien Ltd., Dublin. 14 с. 2016. URL : https://arxiv.org/pdf/1609.04747 (дата звернення: 19.03.2024).

38. Adam. Pytorch: веб-сайт. URL : https://pytorch.org/docs/stable/generated/torch.optim.Adam.html (дата звернення: 20.03.2024).

39. An Introduction to Q-Learning: A Tutorial For Beginners. Datacamp: веб-сайт. URL : https://www.datacamp.com/tutorial/introduction-q-learning-beginner-tutorial (дата звернення: 21.03.2024).

40. Q-learning. Wikipedia: веб-сайт. URL : https://en.wikipedia.org/wiki/Q-learning (дата звернення: 22.03.2024).

41. Maciej Świechowski, Konrad Godlewski, Bartosz Sawicki, Jacek Mańdziuk. Monte Carlo Tree Search: A Review of Recent Modifications and Applications. Faculty of Mathematics and Information Science, Warsaw University of Technology, Warsaw, Poland. 99 с. 2021. URL : https://arxiv.org/pdf/2103.04931 (дата звернення: 22.03.2024).

42. Replay Buffers. TensorFlow: веб-сайт. URL : https://www.tensorflow.org/agents/tutorials/5_replay_buffers_tutorial (дата звернення: 22.03.2024).

43. Thibault Lahire, Matthieu Geist, Emmanuel Rachelson. Large Batch Experience Replay. International Conference on Machine Learning, Baltimore, Maryland, USA, PMLR 162, 2022. 24 с. URL : https://arxiv.org/pdf/2110.01528 (дата звернення: 23.03.2024).

44. Alexia Jolicoeur-Martineau, Emy Gervais, Kilian Fatras, Yan Zhang, Simon Lacoste-Julien. PopulAtion Parameter Averaging. Samsung SAIT AI Lab, Montreal. 2023. URL : https://ajolicoeur.ca/papa/#:~:text=Ensemble%20methods

%20combine%20the%20predictions,their%20weights%20(model%20soups) (дата звернення: 24.03.2024).

45. Jason Brownlee. Weight Initialization for Deep Learning Neural Networks. Machine Learning Mastery. 2021. URL : https://machinelearningmastery.com/weight-initialization-for-deep-learning-neural-networks/#:~:text=Weight%20initialization%20is%20a%20procedure,of%20the%20neural%20network%20model. (дата звернення: 25.03.2024).

46. Hyunwoo Lee, Yunho Kim, Seung Yeop Yang, Hayoung Choi. Improved weight initialization for deep and narrow feedforward neural network. Department of Mathematics, Kyungpook National University, Daegu 41566, Republic of Korea. 2023. 13 с. URL : https://arxiv.org/pdf/2311.03733 (дата звернення: 26.03.2024).

47. Regularization in Machine Learning (with Code Examples). Dataquest: веб-сайт. 2022. URL : https://www.dataquest.io/blog/regularization-in-machine-learning/ (дата звернення: 27.03.2024).

ДОДАТОК А

Стуктура проєкту та повний код програми на Github

```
└── src
    ├── chess_net.py
    ├── evaluate_model.py
    ├── gui
    │   └── gui.py
    ├── historical_games
    │   ├── games_clean.csv
    │   └── games_orig.csv
    ├── history.py
    ├── images
    │   ├── bB.png
    │   ├── bK.png
    │   ├── bN.png
    │   ├── bP.png
    │   ├── bQ.png
    │   ├── bR.png
    │   ├── wB.png
    │   ├── wK.png
    │   ├── wN.png
    │   ├── wP.png
    │   ├── wQ.png
    │   └── wR.png
    └── models
        └── historical_model
            └── model
```

```
        └── assets
        └── fingerprint.pb
        └── keras_metadata.pb
        └── saved_model.pb
        └── variables
            └── variables.data-00000-of-00001
            └── variables.index
    ├── plots
└── learn_evaluate
    └── model
└── reinforcement_learning_stockfish_model
    └── game_results.json
    └── model
        └── assets
        └── fingerprint.pb
        └── keras_metadata.pb
        └── saved_model.pb
        └── variables
            └── variables.data-00000-of-00001
            └── variables.index
    └── plot.png
    ├── plots
└── self_learning_model
    └── game_results.json
    └── model
        └── assets
        └── fingerprint.pb
        └── keras_metadata.pb
        └── saved_model.pb
        └── variables
            └── variables.data-00000-of-00001
```

```
            └── variables.index
        ├── plots
└── self.py
└── stockfish
    └── AUTHORS
    └── CITATION.cff
    └── Copying.txt
    └── README.md
    └── src
        └── benchmark.cpp
        └── benchmark.h
        └── bitbase.cpp
        └── bitboard.cpp
        └── bitboard.h
        └── endgame.cpp
        └── endgame.h
        └── evaluate.cpp
        └── evaluate.h
        └── incbin
            └── incbin.h
            └── UNLICENCE
        └── main.cpp
        └── Makefile
        └── material.cpp
        └── material.h
        └── misc.cpp
        └── misc.h
        └── movegen.cpp
        └── movegen.h
        └── movepick.cpp
        └── movepick.h
```

```
└── nnue
    └── evaluate_nnue.cpp
    └── evaluate_nnue.h
    └── features
        └── half_ka_v2_hm.cpp
        └── half_ka_v2_hm.h
    └── layers
        └── affine_transform.h
        └── affine_transform_sparse_input.h
        └── clipped_relu.h
        └── simd.h
        └── sqr_clipped_relu.h
    └── nnue_accumulator.h
    └── nnue_architecture.h
    └── nnue_common.h
    └── nnue_feature_transformer.h
└── pawns.cpp
└── pawns.h
└── position.cpp
└── position.h
└── psqt.cpp
└── psqt.h
└── search.cpp
└── search.h
└── syzygy
    └── tbprobe.cpp
    └── tbprobe.h
└── thread.cpp
└── thread.h
└── thread_win32_osx.h
└── timeman.cpp
```

```
        └── timeman.h
        └── tt.cpp
        └── tt.h
        └── tune.cpp
        └── tune.h
        └── types.h
        └── uci.cpp
        └── uci.h
        └── ucioption.cpp
    └── stockfish.exe
    └── Top CPU Contributors.txt
    └── wiki
        └── Advanced-topics.md
        └── Commands.md
        └── Compiling-from-source.md
        └── Developers.md
        └── Download-and-usage.md
        └── Governance-and-responsibilities.md
        └── Home.md
        └── Regression-Tests.md
        └── Stockfish-FAQ.md
        └── Terminology.md
        └── Useful-data.md
        └── _Footer.md
└── stockfish.py
└── time.txt
└── utils
    └── buffer.py
    └── clean.py
    └── models.py
    └── moves.py
```

└── stockfish_evaluation.py

└── tensors.py

└── train.py

Посилання на повний код проєкту на Github: https://github.com/ppleaser/chessAI

Декларація

академічної доброчесності

здобувача ступеня вищої освіти ЗНУ

Я, Долженко Нікіта Андрійович, студент 4 курсу, денної форми навчання, математичного факультету, спеціальності 122 комп'ютері науки, адреса електронної пошти nikitadolzhenko03@gmail.com, – підтверджую, що написана мною кваліфікаційна робота магістра на тему «Дослідження ефективності навчання нейронної мережі грі в шахи» відповідає вимогам академічної доброчесності та не містить порушень, що визначені у ст. 42 Закону України «Про освіту», зі змістом яких ознайомлений;

– заявляю, що надана мною для перевірки електронна версія роботи є ідентичною її друкованій версії;

– згоден на перевірку моєї роботи на відповідність критеріям академічної доброчесності у будь-який спосіб, у тому числі за допомогою інтернет-системи, а також на архівування моєї роботи в базі даних цієї системи.

Студент

15.05.2024		Долженко Н.А.
(дата)	(підпис)	(прізвище, ініціали)

Науковий керівник

15.05.2024		Добровольський Г.А.
(дата)	(підпис)	(прізвище, ініціали)