Міністерство освіти і науки України
Львівський національний університет імені Івана Франка
Факультет прикладної математики та інформатики
Кафедра математичного моделювання соціально-економічних процесів

Курсова робота

Застосування Reinforcement Learning у відеоіграх

Студент групи ПМа-32:
Коломієць Данііл Васильович,
спеціальність 124-системний аналіз

Науковий керівник:
кандидат фізико-математичних наук,
доцент
Філь Богдан Миколайович

Львів–2024

Зміст

Вступ

Глибоке підкріплювальне навчання (Deep Reinforcement Learning, DRL) є однією з найбільш перспективних галузей сучасного штучного інтелекту, яка знаходить широке застосування у різних сферах, включаючи відеоігри. Відеоігри є складними середовищами, де алгоритми DRL можуть демонструвати свою здатність до навчання складних стратегій та адаптації до змінних умов. Незважаючи на значні досягнення у розвитку DRL, проблема ефективного навчання агентів у складних ігрових середовищах залишається актуальною. Актуальність даної роботи полягає в дослідженні можливостей різних моделей DRL для навчання агентів у грі Space Invaders, яка є класичним прикладом складної гри з багатьма викликами для агентів. Розуміння того, яка з моделей (DQN, PPO, A2C) є найбільш ефективною для цієї гри, допоможе вдосконалити методи навчання агентів у подібних середовищах та сприятиме подальшому розвитку технологій DRL.

Метою роботи є порівняння трьох популярних моделей DRL — DQN, PPO та A2C — для визначення їхньої ефективності у навчанні агентів для гри Space Invaders. Це порівняння включає аналіз їхньої продуктивності, стабільності навчання та здатності до адаптації, що дозволить зробити висновки про їхню придатність для вирішення складних ігрових завдань.

Розділ 1

Постановка задачі штучного інтелекту

У цьому розділі приведено визначення методу Reinforcement Learning (RL), а також розглянуто проблеми, що виникають при його використанні у відеоіграх.

1.1. Навчання з підкріпленням

Навчання з підкріпленням (RL) - це метод машинного навчання (ML), який навчає модель приймати рішення для досягнення найоптимальніших результатів. Він імітує процес навчання методом спроб і помилок, який людина використовує для досягнення своїх цілей. Дії моделі, які сприяють досягненню мети, посилюються, тоді як дії, які відволікають - ігноруються.

Алгоритми RL використовують парадигму заохочення і покарання для обробки даних, навчаючись на основі зворотного зв'язку від кожної дії. Вони самостійно знаходять найкращі шляхи для досягнення кінцевого результату, іноді включаючи короткострокові жертви. RL допомагає системам штучного інтелекту досягати оптимальних результатів у непередбачуваних умовах.

Проблеми RL можна уявити як систему з агента і середовища. Середовище має стан, який агент спостерігає для вибору дії. Обрана дія виконується, середовище переходить у новий стан, і агент отримує винагороду. Цикл "стан → дія → винагорода"повторюється до завершення середовища, наприклад, до вирішення проблеми. Цей процес відбувається протягом епізоду.

Рис. 1.1: Діаграма управління

Функція, що породжує дії агента, називається *політикою(policy)*. Формально, політика - це функція, яка деякому стану співставляє дію.

1.2. Проблеми Ігрового середовища

У нас є комп'ютерна відеогра, в якій агент має навчитися приймати оптимальні рішення для досягнення заданих цілей, таких як перемога над суперниками або виконання місій. Метою є розробити методи, що дозволяють агенту, використовуючи алгоритми глибокого навчання з підкріпленням (DRL), ефективно навчатися та адаптуватися до складного і динамічного ігрового середовища.

Основні виклики, з якими ми стикаємося:

- *Візуальне сприйняття:* Агент повинен аналізувати візуальні дані з екрану гри та перетворювати їх у числові значення, придатні для алгоритмів навчання.
- *Високорозмірний простір станів:* Ігрове середовище характеризується складним і високорозмірним простором станів, що ускладнює ефективне представлення та обробку інформації.
- *Великий простір дій:* Агент має широкий набір можливих дій, які потрібно оптимально використовувати для досягнення цілей гри.
- *Нагороди:* Нагороди в грі можуть бути рідкісними та відкладеними, що ускладнює процес навчання агента.

Розділ 2

Теоретичне рішення проблеми

У цьому розділі буде описано теоретичне рішення проблеми глибокого навчання з підкріпленням (DRL), зокрема, розглянемо три основні алгоритми DRL, які широко використовуються у відеоіграх: Deep Q-Network (DQN), Advantage Actor-Critic (A2C) та Proximal Policy Optimization (PPO). Кожен з цих алгоритмів має свої особливості, переваги та обмеження, що робить їх застосування ефективним у різних сценаріях. Крім того, розглянемо проблему представлення ігрового середовища для алгоритмів DRL. Для ефективного навчання алгоритми повинні отримувати відповідні, добре структуровані дані про стан ігрового середовища. Це включає обробку зображень, що надають інформацію про поточний стан гри, та перетворення їх у форму, придатну для обробки нейронними мережами. Також важливим є вибір метрик і винагород, які б точно відображали цілі та прогрес у грі, щоб алгоритми могли ефективно вчитися і приймати оптимальні рішення.

2.1. Навчання з підкріпленням (RL)

Традиційно RL використовує математичні алгоритми для моделювання та визначення винагороди на основі невеликих наборах даних та простому навколишньому середовища. Ця проблема часто моделюється математично як *марковський процес вирішування*. Марковський процес вирішування (МПВ) - це стохастичний процес прийняття рішень, який використовує математичний апарат для моделювання прийняття рішень динамічною системою. Він використовується в сценаріях, де результати або випадкові, або контролюються особою, яка приймає послідовні рішення в часі. МПВ оцінює, які дії має здійснити особа, що приймає рішення, з огляду *лише на поточний стан середовища*. Зрозуміло, що якщо дія агента залежатиме від усіх попередніх станів та дій це призведе де неймовірної комплексності задачі. Тому функція переходу набуває наступного вигляду:

$$s_{t+1} \sim P(s_{t+1}|s_t, a_t), \tag{2.1}$$

де s_t, a_t - стан системи та дія агента на кроці t.

Тепер МПВ може бути сформульований у контексті RL. МПВ визначається кортежом з 4-х елементів, $(\mathcal{S}, \mathcal{A}, P(.), \mathcal{R}(.))$, де

- $\mathcal{S}$ - множина станів.
- $\mathcal{A}$ - множина дій.
- $P(s_{t+1}|s_t, a_t)$ - функція переходу станів середовища.
- $\mathcal{R}(s_t, a_t, s_{t+1})$ - функція винагороди середовища.

Одне з важливих припущень, що лежить в основі проблем навчання з підкріпленням, полягає в тому, що агенти не мають доступу до функції переходу $P(s_{t+1}|s_t, a_t)$ або функції винагороди $\mathcal{R}(s_t, a_t, s_{t+1})$. Єдиний спосіб, яким

агент може отримати інформацію про ці функції це через стани, дії та винагороди, які він отримує досліджуючи навколишнє середовище, тобто кортежі "досвіду"(s_t, a_t, r_t). *Траєкторією* будемо називати послідовність отриманого агентом досвіду на протязі одного епізоду, тобто $\tau = (s_0, a_0, r_0), \ldots, (s_t, a_t, r_t)$

Щоб завершити формулювання задачі, нам також потрібно формалізувати поняття цілі, яку максимізує агент. Спочатку визначимо прибуток $R(\tau)$ використовуючи траєкторію з епізоду:

$$R(\tau) = r_0 + \gamma r_1 + \gamma^2 r_2 + \cdots + \gamma^k r_k = \sum_{t=0}^{k} \gamma^t r_t. \tag{2.2}$$

Рівняння 2.2 визначає прибуток як суму винагород, де $\gamma \in [0, 1]$ - коефіцієнт знижки.

Тоді означемо так звану *ціль* $J(\tau)$ яка є просто сподіванням прибутковості за багатьма траєкторіями, як показано в рівнянні 2.3:

$$J(\tau) = \mathbb{E}_{\tau \sim \pi}[R(\tau)] = \mathbb{E}[\sum_{t=0}^{k} \gamma^t r_t]. \tag{2.3}$$

Прибуток $R(\tau)$ - це сума знижених винагород $\gamma^t r_t$ на всіх часових кроках $t = 0, \ldots, k$. Ціль $J(\tau)$ - це прибуток, усереднений за багатьма епізодами. Сподіванням враховує стохастичність у діях та середовищі - тобто, при повторних прогонах прибуток може не завжди бути однаковим. Максимізація мети - це те саме, що максимізація прибутку.

Коефіцієнт знижки $\gamma \in [0, 1]$ є важливою змінною, яка змінює спосіб оцінки майбутніх винагород. Чим менший γ, тим менша вага надається винагороді в майбутніх часових кроках, що робить ціль "недалекоглядною". У крайньому випадку, коли $\gamma = 0$, ціль враховує лише початкову винагороду.

Сформулювавши задачу RL у вигляді МПВ, ми можемо визначити чому агент має навчитися. Ми бачили, що агент може вивчати функцію, яка виробляє дії, відому як політика. Однак є й інші властивості середовища,

які можуть бути корисними для агента. Зокрема, є три основні функції, які можна використовуються агентом при навчанні з підкріпленням:

1. Політика π, яка у відповідність стану s вибирає дію a: $a \sim \pi(s)$.

2. Функція *цінності стану* $V^{\pi}(s)$, являє собою сподівання від прибутку перебуваючи у стані s дотримуючись стратегії π.

3. Функція *цінності дій* $Q^{\pi}(s, a)$, визначається як сподівання від прибутку перебуваючи у стані s, виконуючи деяку дію a та подальшого дотримання стратегії π.

Політика π - це те, як агент виробляє дії в середовищі для максимізації мети. Згідно з циклом управління рис 1.1 навчання з підкріпленням, агент повинен виконувати дію на кожному кроці після спостереження стану s. Політика є фундаментальною для цього циклу керування, оскільки вона генерує дії, які змушують його працювати. Політика може бути стохастичною. Це означає, що вона може імовірнісно виводити різні дії для одного і того ж стану. Ми можемо записати це як $\pi(a|s)$, щоб позначити ймовірність дії a для даного стану s. Дія, вибрана з політики, записується як $a \sim \pi(s)$.

Функції цінності надають інформацію про ціль. За допомогою них агент може оцінити свій поточний стан та ефективність можливих дій у контексті очікування майбутнього прибутку. Вони бувають двох видів:

$$V^{\pi}(s) = \mathbb{E}_{\pi}[R(\tau)|s_0 = s] \tag{2.4}$$

$$Q^{\pi}(s, a) = \mathbb{E}_{\pi}[R(\tau)|s_0 = s, a_0 = a] \tag{2.5}$$

Функція цінності стану $V^{\pi}(s)$ показана у рівнянні 2.4 оцінює, наскільки хороший чи поганий стан.

Функція $Q^{\pi}(s, a)$ показана в рівнянні 2.5, оцінює, наскільки хорошою чи поганою є пара стан-дія. Q^{π} вимірює сподівання прибутку від виконання дії a в стані s за умови, що агент продовжує діяти відповідно до своєї поточної

стратегії π. Так само, як і V^{π}, віддача вимірюється від поточного стану s до кінця епізоду. Це також перспективна міра, оскільки всі винагороди, отримані до стану s, ігноруються.

У сучасному світі для взаємодії зі складним навколишнім середовищем недостатньо традиційного навчання з підкріпленням. Сьогодні під RL зазвичай мають на увазі *Deep Reinforcement Learning* (DRL), сучасний підхід, який використовує *глибоку нейронну мережу (DNN)* для навчання RL моделей. DNN - це штучна нейронна мережа з багатьма шарами нейронів між вхідними та вихідними шарами. Це дозволяє виявляти складні закономірності та абстракції в даних, що забезпечує точні прогнози в нових умовах.

DRL моделі використовують систему винагород і покарань для присвоєння ваги різним нейронам під час навчання. Глибинне навчання дозволяє обробляти багатовимірні та складні дані. Наприклад, *згорткові нейронні мережі (ЗНМ)* використовуються для задач, пов'язаних з візуальними даними, таких як розпізнавання об'єктів і розуміння сцен у відеоіграх. Вони фіксують просторові ієрархії та закономірності на зображеннях, що важливо для агентів у візуально насиченому середовищі.

2.2. Глибинне Q-навчання

2.2.1. Q-навчання

Простим підходом до використання функцій цінності є прямий підхід, який застосовується в методі Q-навчання. У цьому підході ми намагаємося вивчити декілька значень для кожного стану, по одному для кожної можливої дії. Це значення називається Q-значенням дії стану і позначається як $Q^{\pi}(s, a)$. Якщо агент має доступ до істинного Q-значення кожної пари стан-дія в оточенні, то можна побудувати оптимальну політику без необхідності вивчати

її у явному вигляді. Це можна зробити, просто переглянувши всі значення стан-дія та вибравши дію, яка має найбільше значення:

$$\max_a Q^\pi(s, a). \tag{2.1}$$

Для значень стану-дії використовується рівняння сподівання Белмана для визначення значення кортежу стану-дії:

$$Q^\pi(s_t, a_t) = \mathbb{E}[r_{t+1} + \gamma Q^\pi(s_{t+1}, a_{t+1})|s_{t+1}, a_{t+1}]. \tag{2.2}$$

Оскільки наші оцінки $Q^\pi(s, a)$ на початку навчання не є точними, ми вводимо поняття швидкості навчання або так званий "крок що позначається $\alpha \in [0, 1]$ для поступового покращення оцінок:

$$Q^\pi(s_t, a_t) \leftarrow Q^\pi(s_t, a_t) + \alpha[r_{t+1} + \gamma Q^\pi(s_{t+1}, a_{t+1}) - Q^\pi(s_t, a_t)]. \tag{2.3}$$

Це частина методу *Temporal Difference(Тимчасова різниця)* навчання, де у випадку $r_{t+1} + \gamma Q^\pi(s_{t+1}, a_{t+1}) - Q^\pi(s_t, a_t)$ є незміщеною оцінкою $Q^\pi(s, a)$. Скорочено ми визначаємо $\delta_t = r_{t+1} + \gamma r_{t+1} + \gamma Q^\pi(s_{t+1}, a_{t+1}) - Q^\pi(s_t, a_t)$ як похибку часової різниці. Використовуючи значення оцінки наступного кроку для навчання поточного кроку, ми здійснюємо бутстрепінг оцінки вартості. Це дозволяє нам почати тренування оцінювача ще до того, як ми точно знаємо, як епізод буде завершенно.

У найпростішому варіанті Q-навчання використовується "Q-таблиця де кожен рядок таблиці представляє стан, а кожен стовпець - дію. Коли кортежі стан-дія вибираються з середовища, відповідні комірки матриці оновлюються за допомогою рівняння 2.3. Структура Q-таблиці показана в таблиці 2.1.

	Дії			
Стани	$a^{(0)}$	$a^{(1)}$	...	$a^{(n-1)}$
s_0	$Q(s_0, a^{(0)})$	$Q(s_0, a^{(1)})$	...	$Q(s_0, a^{(n-1)})$
s_1	$Q(s_1, a^{(0)})$	$Q(s_1, a^{(1)})$	...	$Q(s_1, a^{(n-1)})$
⋮	⋮	⋮	⋱	⋮
s_1	$Q(s_{m-1}, a^{(0)})$	$Q(s_{m-1}, a^{(1)})$	...	$Q(s_{m-1}, a^{(n-1)})$

Таблиця 2.1: Візуальне представлення Q - таблиці для середовища з n станами та m діями, де кортеж стан-дія педставляє Q-значення.

Очевидно, що побудова такої таблиці дуже дорога, якщо окремий простір дій-станів є великим, то він швидко стає нездійсненним для управління. Крім того, побудова таблиці також можлива лише тоді, коли дії є дискретними. Щоб вирішити цю проблему, ми можемо замінити Q-таблицю параметризованим апроксиматором, який можна навчити за допомогою правила оновлення, подібного до 2.3. Також постає проблема між дослідженням та експлуатацією, коли необхідно збалансувати спроби знайти нові шляхи для підвищення ефективності та використання вже відомих оптимальних стратегій. Евристики, такі як політика епсілон-жадібності, поєднують розвідку та експлуатацію, але задовільної або однозначної відповіді на цю проблему не існує.

У Q-навчанні ми обираємо дію на основі її винагороди. Агент завжди обирає оптимальну дію. Таким чином, він генерує максимальну винагороду, можливу для даного стану. При епсилон-жадібному виборі дій агент використовує як експлуатацію, щоб скористатися перевагами попередніх знань, так і розвідку, щоб шукати нові варіанти Рис. 2.1.

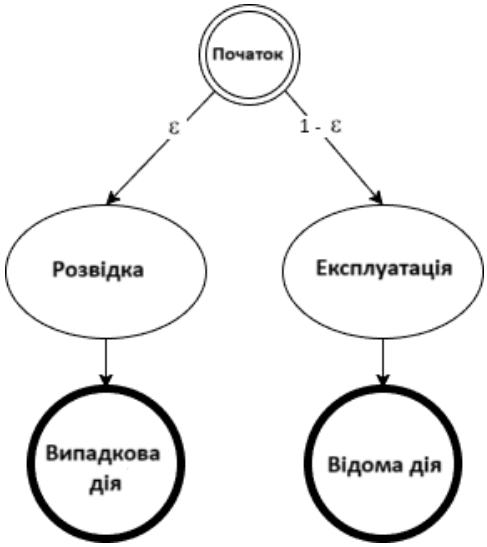

Рис. 2.1: Візуальне представлення епсилон-жадібного підходу до вибору дій.

Епсилон-жадібний підхід обирає дію з найбільшою очікуваною винагородою в більшості випадків, балансуючи між розвідкою та експлуатацією. Розвідка дозволяє спробувати щось нове, що іноді суперечить наявним знанням. З невеликою ймовірністю ε ми вирішуємо досліджувати, вибираючи дію випадково, незалежно від оцінок значення дії.

2.2.2. Deep Q-Network

Найпростіше покращення, яке глибокі нейронні мережі (DNN) можуть запропонувати для Q-навчання, - це діяти як апроксиматори Q-значення замість дорогої Q-таблиці. DNN може спостерігати за поточним станом і можливою дією та безпосередньо наближати значення Q. Це має очевидну перевагу в тому, що нейронна мережа здатна узагальнювати наявні пари стан-дія. Завдяки здатності до узагальнення, агент, який використовує глибоке Q-навчання, може точно визначати значення стану-дії для рідкісних або раніше небачених комбінацій стану-дії.

Нейронна мережа, що використовується для прогнозування Q-значень кожної дії стану, може бути використана в поєднанні з політикою ε-жадібності

для випадкового та жадібного дослідження середовища та навчання DQN з можливістю прогнозування значень дій стану.

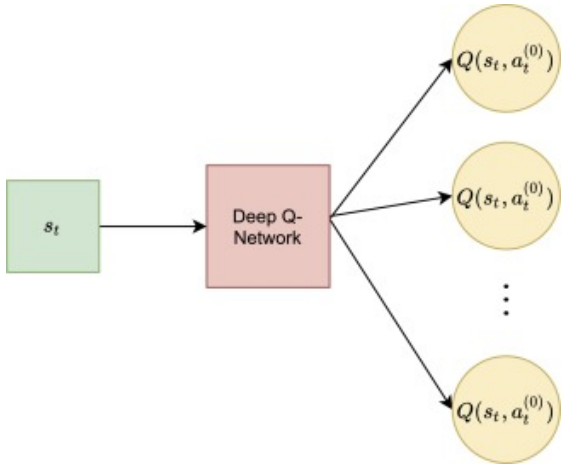

Рис. 2.2: Обчислення значень стан-дія для всіх можливих дій одного стану за один прохід.

Для тренування DQN спочатку відбирають зразки середовища, щоб зібрати "досвід". Ці вибірки досвіду називаються епізодами і додаються до пам'яті відтворення, яка черпає натхнення з психологічних досліджень. Щоб оновити оцінку Q-значення, епізоди витягуються з пам'яті відтворення і функція втрат, отримана з 2.4, обчислюється як

$$L_{s,a,r,s'} = (r + \gamma \max_{a'} Q(s', a') - Q(s, a))^2, \tag{2.4}$$

де s' стан, до якого агент потрапив після виконання дії a у стані s. У втратах ми припускаємо, що оцінка Q-значення базується на виборі оптимальних дій, тому максимальне значення стан-дія з можливих дій використовується як значення наступного стану. Градієнт функції втрат поширюється через DQN оновлюючи його ваги, поступово покращуючи прогнози значення Q. Як зазвичай для нейронних мереж, функція втрат застосовується до партії з декількох переходів, а потім усереднюється для покращення точності градієнтів.

2.3. Advantage Actor-Critic (A2C)

Алгоритм *актор-критик* - це тип алгоритму навчання з підкріпленням, який поєднує в собі аспекти методів, що базуються на політиці (актор) та цінностях (критик). Цей гібридний підхід призначений для усунення обмежень кожного методу при використанні окремо.

У рамках акторно-критичного підходу агент (*актор*) вивчає політику для прийняття рішень, а ціннісна функція (*критик*) оцінює дії, здійснені актором.

Одночасно критик оцінює ці дії, визначаючи їхню цінність або якість. Ця подвійна роль дозволяє методу досягти балансу між розвідкою та експлуатацією, використовуючи сильні сторони як політичної, так і ціннісної функцій.

2.3.1. Актор

Актори вивчають параметризовані політики π_θ, використовуючи градієнт політики, як показано у рівнянні 2.5 доведення якого описано в [4].

$$\nabla_\theta J(\theta) = \mathbb{E}[A_t^\pi \nabla_\theta \log \pi_\theta(a_t, s_t)], \tag{2.5}$$

де A_t^π - *перевага* яку генерує критик. Актор вибирає дії на основі поточної політики, взаємодіючи з середовищем і збираючи дані про свої дії та отримані винагороди. Потім актор оновлює параметри політики, щоб покращити її відповідно до обчисленого градієнта, з метою максимізації довгострокового виграшу.

2.3.2. Критик

Як вже було сказано вище, критики в свою чергу відповідають за те, щоб навчитися оцінювати пари (s; a) і використовувати це для отримання A^π

Інтуїтивно зрозуміло, що *функція переваги* $A^\pi(s_t, a_t)$ вимірює ступінь, до якого дія є кращою або гіршою за середню дію політики в конерутному стані. Перевага визначається рівнянням 2.6.

$$A^\pi(s_t, a_t) = Q^\pi(s_t, a_t) - V^\pi(s_t). \tag{2.6}$$

Загальний алгоритм A2C виглядає наступним чином:

1. *Ініціалізація*:
    - Ініціалізувати параметри актора (політики), та параметри критика (функції значення).
2. *Взаємодія з середовищем*:
    - Агент взаємодіє з середовищем, вибираючи дії відповідно до поточної політики, визначеної актором.
    - Середовище надає спостереження про стан та винагороди у відповідь на дії агента.
3. *Обчислення переваги*:
    - Обчислити функцію переваги $A^\pi(s, a)$ на основі поточної політики та оцінених значень станів.
    - Функція переваги кількісно характеризує перевагу виконання певної дії в певному стані порівняно з середнім очікуваним поверненням.
4. *Оновлення політики та значень*:
    - *Оновлення актора*: Використовуйте метод градієнта політики для оновлення параметрів актора.

- *Оновлення критика*: Використовуйте метод на основі значень для оновлення параметрів критика. Це часто включає мінімізацію помилки тимчасової різниці (TD).

2.4. Проксимальна оптимізація політики(PPO)

Проксимальна оптимізація політики (PPO) - це алгоритм навчання з підкріпленням, розроблений компанією OpenAI. Існують два основних типи PPO, які включають:

- PPO з обмеженням (clip)
- PPO з використанням Kullback-Leibler Proximity (KL-PEN)

Перш ніж заглибитись у варіанти, спростимо сурогатну ціль $J_{CPI}(\theta)$, ввівши відношення ймовірностей $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}$ і для стислості позначимо перевагу $A_{\pi_t}^{\theta_{\text{old}}}$ як A_t. Сурогатною метою стає:

$$J_{CPI}(\theta) = \mathbb{E}_{\pi_{\theta_{\text{old}}}}[r_t(\theta)A_t], \tag{2.7}$$

де сурогатна ціль - це функція, яка використовується в методах градієнта політики, щоб наблизити справжню цільову функцію, яка вимірює ефективність політики.

Перший варіант, PPO з адаптивним KL-штрафом, вводить до мети адаптивний KL-штрафний доданок, який має на меті контролювати розбіжність між новою та старою політикою. Цей штрафний член віднімається від переваги, зваженої на важливість, що призводить до наступної мети:

$$J_{KLPEN}(\theta) = \max_\theta \mathbb{E}_t[r_t(\theta)A_t - \beta \text{KL}(\pi_\theta(a_t|s_t)||\pi_{\theta_{\text{old}}}(a_t|s_t))], \tag{2.8}$$

тут β є адаптивним коефіцієнтом, який контролює розмір штрафу KL, змінюючи довірчу область оптимізації. Проблема використання постійного коефіцієнта вирішується шляхом оновлення β після кожного оновлення політики за допомогою евристичного правила оновлення.

На противагу цьому, PPO з обрізаною сурогатною метою спрощує обчислення, видаляючи обмеження KL і використовуючи обрізану версію сурогатної мети. Обрізана мета $J_{CLIP}(\theta)$ визначається наступним чином:

$$J_{CLIP}(\theta) = \mathbb{E}_t[\min(r_t(\theta)A_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)A_t)], \tag{2.9}$$

де ϵ визначає околицю відсікання, гарантуючи, що відношення ймовірностей $r_t(\theta)$ залишається у визначеному діапазоні. Відсікаючи ціль, PPO запобігає великим і ризикованим оновленням політики, роблячи процес оптимізації безпечнішим і стабільнішим.

Загальний алгоритм PPO виглядає наступним чином:

- *Ініціалізація мережі політики*: Ініціалізації нейронної мережі, яка представляє політику. Ця мережа приймає поточний стан на вході і виводить розподіл ймовірностей можливих дій.
- *Збір траєкторій*: Агент взаємодіє з навколишнім середовищем, дотримуючись поточної політики, під час якої він збирає траєкторії.
- *Обчислення оцінок переваг*: Використовуючи зібрані траєкторії, обчислюються оцінки переваг для кожної пари стан-дія.
- *Обчислення сурогатної мети*: Обчислюється сурогатна мета, яка є функцією старих і нових параметрів політики та оцінок переваг. Ця ціль апроксимує покращення політики, гарантуючи при цьому, що оновлення політики залишається в безпечному діапазоні.
- *Оптимізація мережі політик*: Використовується стохастичний градієнтний спуск (SGD) або інший варіант для оновлення параметрів мережі політик, мінімізуючи сурогатну мету.

2.5. Середовище гри

Комп'ютерні ігри надають необіхдну інформацію за допомогою екрану, тим самим забираючі прямий доступ до значення різних параметрів і представляючи їх у вигляді зображень. Люди у свою чергу можуть легко сприймати цю інформацію, а DRL агент сприймає лише числові значення параметрів гри, через це інформацію на зображенні треба обробити та представити у вигляді значень, які агент може сприймати. Варто зазначити, що ця проблема наявна лише при тренувані DRL агентів сторонім лицем, а не розробником гри. Так як розробники гри мають всю необхідну інформацію у числових значеннях, вони можуть інтегрувати DRL у їхню гру, тим самим підвищивши ефективність навчання агента.

Кожен кадр гри містить візуальне зображення поточного стану ігрового середовища. І в залежності від динаміки ігрової системи, агент має отримувати n-кількість зображень. Зібрані зображення потребують попередньої обробки, щоб виділити корисну інформацію і зменшити обчислювальні витрати. Основні методи обробки зображень включають:

- *Зменшення роздільної здатності:* Висока роздільна здатність зображень може значно збільшити обчислювальні витрати. Тому, зображення часто зменшуються до меншого розміру.
- *Перетворення в градації сірого:* Перетворення зображень у градації сірого зменшує кількість каналів, що знижує обчислювальну складність.
- *Нормалізація:* Піксельні значення зображень нормалізуються до діапазону [-1, 1]. Це підвищує швидкість навчання та впроваджує незалежність відносно розширення екрану.
- *Обтинання:* Деякі частини зображення, такі як інтерфейс користувача (UI), можуть не містити корисної інформації для алгоритмів DRL.

Після попередньої обробки зображення можуть бути піддані подальшій обробці з використанням методів комп'ютерного зору для виділення ключової інформації:

- *Виявлення об'єктів:* Використовуючи алгоритми, такі як YOLO (You Only Look Once) або Faster R-CNN, можна виявити та ідентифікувати ключові об'єкти на зображенні, наприклад, персонажів, ворогів, предмети і т.д.

- *Сегментація зображень:* Методи сегментації дозволяють розділити зображення на області, що відповідають різним об'єктам чи частинам сцени.

- *Відстеження об'єктів:* Методи відстеження об'єктів дозволяють визначити траєкторії руху об'єктів у часі, що є важливим для розуміння динаміки ігрового середовища і прогнозування майбутніх станів.

Після обробки зображень, їх необхідно перетворити у формат, зручний для алгоритмів DRL:

- *Тензорне представлення:* Зображення перетворюються у тензори, які можуть бути подані на вхід нейронних мереж. Наприклад, тензор для зображення 84x84 у градаціях сірого матиме розмір 84x84x1.

- *Векторні ознаки:* Крім зображень, важливо представити інші ознаки, такі як позиції об'єктів, стан здоров'я персонажа, наявні ресурси тощо. Ці дані можуть бути представлені у вигляді векторів і об'єднані з тензорами зображень.

Таке подання інформації про стан гри відповідає вимогам і можливостям алгоритмів глибокого навчання з підкріпленням, забезпечуючи їм доступ до повної та релевантної інформації для ефективного тренування або прийняття рішень у грі.

Розділ 3

Чисельні експерименти

У цьому розділі приведено чисельні експерименти, в яких буде розглянуто три моделі глибокого підкріплювального навчання (DRL): Proximal Policy Optimization (PPO), Deep Q-Network (DQN) та Advantage Actor-Critic (A2C). Ці моделі будуть застосовані до гри Space Invaders для порівняння їх ефективності та продуктивності.

3.1. Середовище розробки та обладнання

Усі числові експерименти були виконані на ноутбуці Asus ROG Strix G15. Характеристики цього пристрою наведені нижче:

- *GPU:* LAPTOP NVIDIA RTX 3060 з 6GB відеопам'яті
- *CPU:* AMD Ryzen 7 6800H, 8 ядер, 16 потоків, з тактовою частотою до 4.7GHz
- *Оперативна пам'ять:* 16GB DDR5
- *SSD:* Kingston KC3000 з швидкістю читання/запису до 7GB/сек

3.1.1. Середовище розробки

Для розробки та проведення числових експериментів використовувалися наступні середовища та інструменти:

- *Visual Studio Code*
  Visual Studio Code (VS Code) — це потужний редактор коду, який підтримує безліч мов програмування і має великий набір плагінів для розширення функціональності. Він був використаний для написання та налагодження коду.

- *Jupyter Notebook*
  Jupyter Notebook — це інтерактивне середовище для виконання коду, яке дозволяє створювати та ділитися документами з живим кодом, рівняннями, візуалізаціями та текстом. Вона була використана для експериментів, аналізу результатів та візуалізації даних.

3.1.2. Використані бібліотеки

Для реалізації та тестування моделей глибокого підкріплювального навчання були використані наступні бібліотеки:

- *TensorFlow*
  TensorFlow — це потужна бібліотека для чисельних обчислень, що спеціалізується на створенні та тренуванні моделей машинного навчання, включаючи глибоке навчання. Вона забезпечує ефективне виконання на GPU, що значно пришвидшує процес тренування моделей.

- *PyTorch*
  PyTorch — це популярна бібліотека для глибокого навчання, відома своєю гнучкістю і легкістю у використанні. Вона забезпечує динамічну

обчислювальну графіку, що дозволяє легко змінювати моделі під час виконання, а також підтримує ефективне виконання на GPU.

- *Gymnasium*
  Gymnasium (раніше відомий як OpenAI Gym) — це бібліотека для створення і тестування середовищ для підкріплювального навчання. Вона забезпечує стандартизований інтерфейс для взаємодії з різноманітними середовищами, такими як класичні ігри, робототехніка та інші симуляції.

- *Stable-Baselines3*
  Stable-Baselines3 — це набір високоякісних реалізацій алгоритмів підкріплювального навчання на основі TensorFlow і PyTorch. Вона забезпечує простий у використанні інтерфейс для тренування та оцінки моделей, що дозволяє швидко і ефективно проводити експерименти з різними алгоритмами.

Ці інструменти та бібліотеки були вибрані завдяки їхній ефективності, потужності та зручності використання, що дозволило швидко розробити, тренувати та оцінити моделі глибокого підкріплювального навчання у грі Space Invaders.

Програмна реалізація глибинного підкріплювального навчання у відеоіграх передбачає вибір відповідного середовища, такого як Gymnasium або Unity ML-Agents, та його налаштування для створення необхідних умов навчання агента. Використовуються популярні алгоритми DRL, як-от DQN, PPO або A2C, і порівнюється їх ефективність у конкретному ігровому середовищі. Для обробки ігрових зображень проектується архітектура нейронної мережі, зазвичай з використанням згорткових нейронних мереж (CNN). Навчальний процес включає налаштування гіперпараметрів, вибір стратегії дослідження та експлуатації (наприклад, епсілон-жадібний) , а також

застосування технік стабілізації, таких як фіксовані цільові мережі або пріоритетний досвід відтворення. Для аналізу і візуалізації навчання Використовуються інструменти, такі як TensorBoard, що дозволяє відстежувати і оцінювати продуктивність моделі на основі показників середньої нагороди. Після завершення початкового навчання проводиться оптимізація параметрів моделі, а також вивчається вплив різних факторів на ефективність навчання.

3.2. Приклади агента на грі Space Invaders

Space Invaders — це класична аркадна гра, вперше випущена в 1978 році, яка стала популярною завдяки своїй простій, але захоплюючій механіці. Гравець керує космічним кораблем в нижній частині екрану, який може рухатися вліво та вправо і стріляти по інопланетних загарбниках, що рухаються вниз з верхньої частини екрану. Мета гри — знищити всіх інопланетян, перш ніж вони досягнуть нижньої частини екрану, уникати ворожих пострілів і заробляти якомога більше очок.

3.2.1. DQN агент

Модель Deep Q-Network (DQN) була тренована для гри Space Invaders з метою оцінки її ефективності у вирішенні складних ігрових задач. Процес тренування тривав 4 години 51 хвилину, під час якого було виконано загалом 1.5 мільйона кроків. Протягом тренування модель навчалася оптимальним діям, аналізуючи різні ігрові стани та винагороди, що дозволило їй покращити свої стратегії для досягнення вищих результатів у грі.

Рис. 3.1: Графік зміни винагороди для DQN агента

Аналізуючи рис 3.1, можна побачити, що модель DQN є здатною до ефективного навчання. Найбільші винагороди були отримані на 3 годині 36 хвилині тренування(1.118 мільйоний крок), де максимальна винагорода становила 354.4. Після цього моменту винагорода впала до 300.

Падіння винагороди після піку може бути пов'язане з кількома факторами:

- *Переобучення:* Модель могла почати переобучатися на специфічних станах, втрачаючи здатність узагальнювати стратегії на інші стани гри.
- *Зміна стратегій:* Під час тренування агент міг перейти до нових стратегій, які тимчасово менш ефективні, але згодом можуть призвести до кращих результатів.
- *Стохастичні зміни:* Природа алгоритмів підкріплювального навчання може призводити до випадкових коливань у винагородах, особливо під час експлоративних кроків.

Таким чином, незважаючи на тимчасове зниження винагороди, модель DQN демонструє здатність до ефективного навчання і покращення своїх стратегій в процесі тренування.

3.2.2. PPO агент

Модель Proximal Policy Optimization (PPO) була тренована на протязі 2 днів та виконала загалом 12 мільйонів кроків. Процес тренування був тривалим, але результативним, що дозволило моделі покращити свої стратегії та досягти високих результатів у грі.

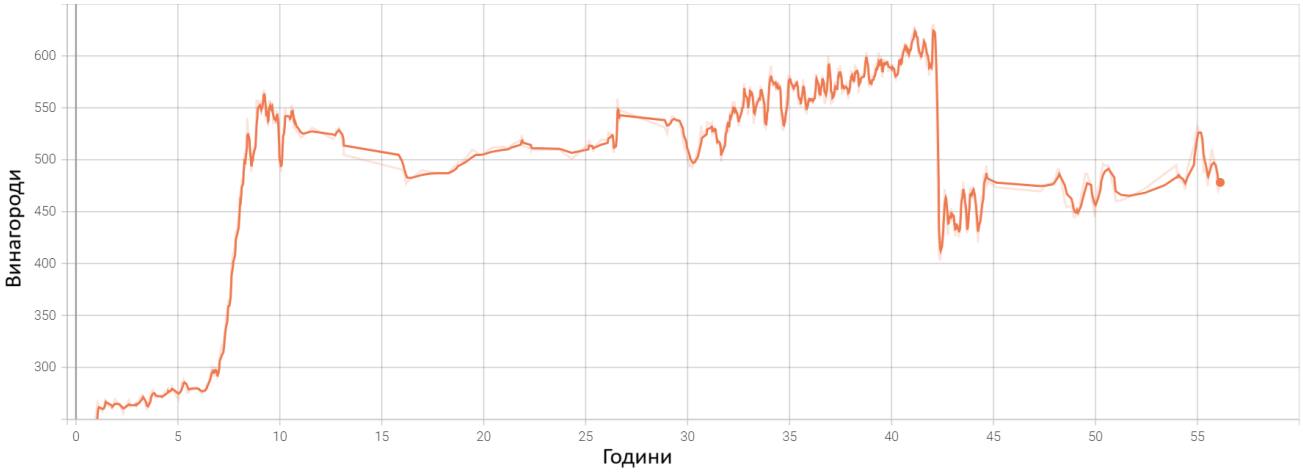

Рис. 3.2: Графік зміни винагороди для PPO агента

Аналізуючи рис 3.2 моделі PPO, можна побачити наступні закономірності. До 7 години навчання середня винагорода моделі становила 250-300 очків. Однак, починаючи з 7 години (1.5 мільйона кроків) і до 8.5 години (2.3 мільйона кроків), модель неймовірно вистрілила, і її середнє значення винагороди становило 550 очків.

На 1 дні та 18 годині (9.28 мільйона кроків) навчання модель досягла свого максимального значення винагороди — 624 очки. Після цього винагорода моделі різко впала до 400 очків, і на останній годині тренування її винагорода становила 484 очки.

Падіння винагороди пов'язане зі зміною стратегії моделі. На своєму піку модель завжди переходила до правого краю екрану і, стоячи там, знищувала ворогів. Починаючи з 1 дня 18 години 30 хвилин тренування (10 мільйонів кроків), модель змінила свою стратегію і почала навчатися тримати прямий контакт з ворогами. Ця нова стратегія могла бути менш ефективною в короткостроковій перспективі, що призвело до тимчасового зниження винагороди.

3.2.3. A2C агент

Модель A2C була піддана тренуванню на протязі 15.5 годин, що відповідає приблизно 14 мільйонам кроків. Під час тренування модель вдосконалювала свої стратегії та оцінювала цінність своїх дій, використовуючи ці дані для максимізації винагороди в грі.

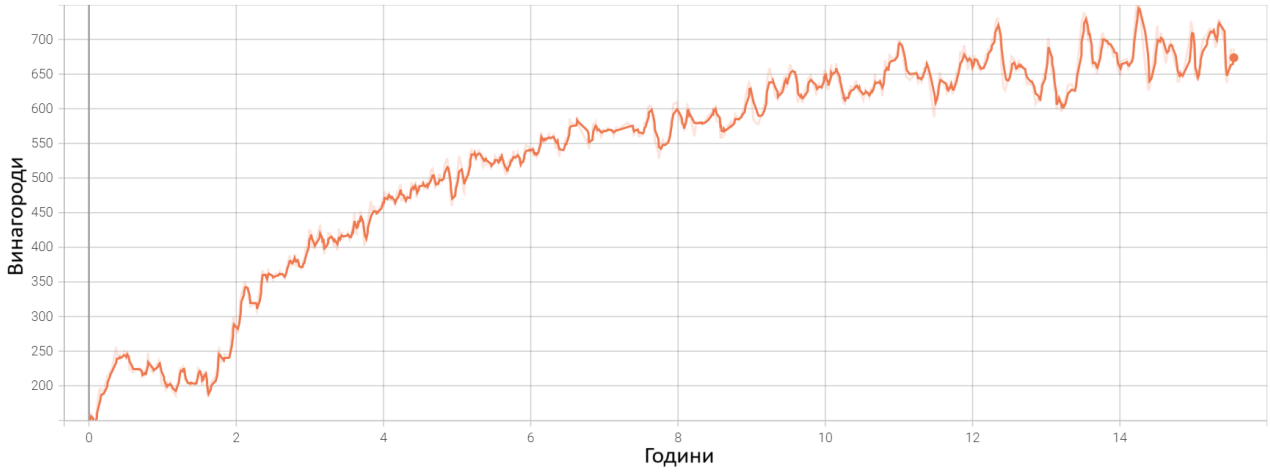

Рис. 3.3: Графік зміни винагороди для A2C агента

Під час аналізу рис 3.3 моделі A2C ,було виявлено, що до 1 години 40 хвилин тренування модель демонструвала невисоку ефективність, з середньою винагородою близько 200 очок. Проте після цього періоду спостерігалося помітне покращення її продуктивності.

Відзначається, що після першого періоду модель почала демонструвати стабільний логарифмічний зріст. Наприкінці п'ятої години тренування вона досягла значення середньої винагороди у 531 очко, що є помітним покращенням порівняно з початковими показниками. Це свідчить про здатність моделі до поступового та послідовного вдосконалення в процесі тренування, що дає підстави вважати, що вона може бути успішно навченою та досягнути високих результатів у майбутньому.

Після аналізу графіків тренування для моделей DQN, PPO і A2C на грі Space Invaders можна зробити наступний висновок: до п'ятої години тренування найкраще себе проявила модель A2C, після чого йшла модель DQN, а потім модель PPO. Проте після п'ятої години тренування модель PPO неймовірно вистрілила, перегнавши DQN. У той час, модель A2C продовжувала навчатися стабільно, залишаючись всі інші поделі позаду.

Аналізуючи ці дані, можна зробити висновок, що найкращою моделлю для гри Space Invaders є A2C. Це не дивно, оскільки модель A2C виявилася стабільною та ефективною протягом усього тренування, показавши найкращі результати до п'ятої години навчання та не втративши свою ефективність пізніше. Отже, модель A2C виявилася кращою за інші моделі, такі як DQN та PPO, у грі Space Invaders.

Висновки

У цій курсовій роботі ми вивчили та порівняли декілька моделей глибокого підкріплювального навчання (DRL) на прикладі гри Space Invaders. Дослідження показало, що кожна з розглянутих моделей - DQN, PPO і A2C - має свої переваги та недоліки.

Модель DQN виявилася досить ефективною у початкових стадіях тренування, проте стійкість до подальшого вдосконалення може бути обмеженою.

Модель PPO проявила найекспресивніші результати, здатність до яких розкрилася вже після початкової фази навчання. Проте виникає питання щодо стійкості цього підходу та його можливості до подальшого вдосконалення.

Нарешті, модель A2C показала стабільний та послідовний зріст у винагороді протягом усього тренування. Це свідчить про її здатність до поступового вдосконалення стратегій та великий потенціал для подальшого розвитку.

Отже, в контексті гри Space Invaders, модель A2C виглядає найбільш перспективною. Проте для конкретної гри або завдання важливо розглянути всі можливі аспекти та особливості кожної моделі перед прийняттям рішення щодо її використання.

Список використаних джерел

[1] Сеньо П. С. Теорія ймовірностей та математична статистика / П. С. Сеньо - Київ : Знання, 2007. - 556 с.

[2] Sutton R.S. Reinforcement Learning: An Introduction / Sutton R.S. , Barto A.G. - London: The MIT Press, 2014. - 337 с.

[3] Laura Graesser Foundations of Deep Reinforcement Learning / Laura Graesser, Wah Loon Keng - U.S: Addison-Wesley, 2020. - 384 с.

[4] Iosifidis A. Deep Learning for Robot Perception / A. Iosifidis, A. Tefas - U.S: Academic Press, 2022. - 634 с.

[5] V7 Labs. Deep Reinforcement Learning Guide [Електронний ресурс]. – Режим доступу: https://www.v7labs.com/blog/deep-reinforcement-learning-guide#h1.

[6] Neptune AI. Model-Based and Model-Free Reinforcement Learning: PyTennis Case Study [Електронний ресурс]. – Режим доступу: https://neptune.ai/blog/model-based-and-model-free-reinforcement-learning-pytennis-case-study~:text=Think%20of%20it%20this%20way,%2C%20it%20is%20model-free.

[7] Wikipedia. Partially Observable Markov Decision Process [Електронний ресурс]. – Режим доступу: https://en.wikipedia.org/wiki/Partially_observable_Markov_decision_process.

[8] Tutorialspoint. Python Deep Learning - Deep Neural Networks [Електронний ресурс]. – Режим доступу: https://www.tutorialspoint.com/python_deep_learning/python_deep_learning_deep_neural_networks.htm.

[9] Deepchecks. Reinforcement Learning Applications: From Gaming to Real World [Електронний ресурс]. – Режим доступу: https://deepchecks.com/reinforcement-learning-applications-from-gaming-to-real-world/.

[10] Pacific Northwest National Laboratory (PNNL). Explainer: Deep Reinforcement Learning [Електронний ресурс]. – Режим доступу: https://www.pnnl.gov/explainer-articles/deep-reinforcement-learning.

[11] Pathmind. Deep Reinforcement Learning [Електронний ресурс]. – Режим доступу: https://wiki.pathmind.com/deep-reinforcement-learning.

[12] Neptune AI. Markov Decision Process in Reinforcement Learning [Електронний ресурс]. – Режим доступу: https://neptune.ai/blog/markov-decision-process-in-reinforcement-learning.

[13] Built In. Markov Decision Process [Електронний ресурс]. – Режим доступу: https://builtin.com/machine-learning/markov-decision-process.

[14] Towards Data Science. Understanding Actor-Critic Methods [Електронний ресурс]. – Режим доступу: https://towardsdatascience.com/understanding-actor-critic-methods-931b97b6df3f.

[15] Scholarpedia. Policy Gradient Methods [Електронний ресурс]. – Режим доступу: http://www.scholarpedia.org/article/Policy_gradient_methods#:~:text=Policy%20gradient%20methods%20are%20a,cumulative%20reward)%20by%20gradient%20descent.

[16] LessWrong. Deep Q-Networks Explained [Електронний ресурс]. – Режим доступу: https://www.lesswrong.com/posts/kyvCNgx9oAwJCuevo/deep-q-networks-explained.