Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет Комп'ютерних наук
(повна назва)
Кафедра Штучного інтелекту
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти перший (бакалаврський)

Використання асинхронних методів для підвищення ефективності Reinforcement Learning
(тема)

Виконав:
здобувач четвертого року навчання,
групи ІТШІ-21-2
Євгеній Рубанов
(власне ім'я, прізвище)

Спеціальність 122 Комп'ютерні науки
(код і повна назва спеціальності)
Тип програми освітньо-професійна
Освітня програма Штучний інтелект
(повна назва освітньої програми)
Керівник доц. Анастасія Дейнеко
(посада, власне ім'я, прізвище)

Допускається до захисту

Завідувач кафедри ШІ _____ Олег ЗОЛОТУХІН
(підпис) (власне ім'я, прізвище)

2025 р.

Харківський національний університет радіоелектроніки

Факультет Комп'ютерних наук

Кафедра Штучного інтелекту

Рівень вищої освіти перший (бакалаврський)

Спеціальність 122 Комп'ютерні науки
(код і повна назва)

Тип програми освітньо-професійна

Освітня програма Штучний інтелект
(повна назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
«_____» _____ 20 ___ р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

здобувачеві Рубанову Євгенію Олеговичу
(прізвище, ім'я, по батькові)

1. Тема роботи Використання асинхронних методів для підвищення ефективності Reinforcement Learning

затверджена наказом університету від 19 травня 20 25 р. № 378Ст
2. Термін подання студентом роботи до екзаменаційної комісії 17 червня 20 25 р.
3. Вихідні дані до роботи наукові статті з області навчання з підкріпленням, документація до алгоритмів A3C, Q-learning та SARSA, офіційна документація середовищ Arcade Learning Environment, TORCS і Labyrinth, результати емпіричних досліджень продуктивності асинхронних методів, а також вихідні коди з відкритих репозиторіїв з реалізаціями алгоритмів DRL.

4. Перелік питань, що потрібно опрацювати в роботі
1) Аналіз предметної галузі та постановка задачі
2) Теоретичні дослідження
3) Експериментальні дослідження

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Строк / терміни виконання етапів роботи	Примітка
1	Отримання завдання на кваліфікаційну роботу	19.05.2025	виконано
2	Визначення проблеми Reinforcing Learning	21.05.2025	виконано
3	Дослідження асинхронних алгоритмів	23.05.2025	виконано
4	Конфігурація експериментального середовища	25.05.2025	виконано
5	Реалізація експериментів	26.05.2025	виконано
6	Оформлення пояснювальної записки	01.06.2025	виконано
7	Захист	17.06.2025	виконано

Дата видачі завдання 19 травня 2025 р.

Здобувач _____
(підпис)

Керівник роботи _____ доц. Анастасія Дейнеко
(підпис) (посада, власне ім'я, прізвище)

РЕФЕРАТ

Пояснювальна записка: 89 с., 27 рис., 2 табл., 1 дод., 24 джерела.

АКТОР-КРИТИК, АСИНХРОННІСТЬ, ГЛИБИННЕ НАВЧАННЯ, НАВЧАННЯ З ПІДКРІПЛЕННЯМ, НЕЙРОННА МЕРЕЖА, ПАРАЛЕЛЬНІ ОБЧИСЛЕННЯ, ADVANTAGE ACTOR-CRITIC, A3C, DEEP Q-NETWORK.

Об’єктом дослідження є процес навчання агентів у задачах з підкріпленням у складних динамічних середовищах.

Предметом дослідження є асинхронні алгоритми навчання з підкріпленням, зокрема методи actor-critic з багатопотоковою реалізацією.

Метою роботи є дослідження ефективності, масштабованості та стабільності асинхронних actor-critic алгоритмів на прикладі A3C у порівнянні з іншими класичними алгоритмами навчання з підкріпленням.

Методи дослідження включають експериментальне моделювання в симуляційних середовищах таких як Atari, TORCS, Labyrinth, реалізацію багатопотокової архітектури навчання, порівняння за метриками продуктивності, стабільності та чутливості до гіперпараметрів.

У результаті дослідження доведено, що алгоритм A3C забезпечує вищу ефективність та стабільність у порівнянні з одно- та багатокроковими методами на основі Q-learning і SARSA; новизна полягає у повноцінному порівнянні асинхронних підходів при змінній кількості потоків і різних умовах середовища.

Результати роботи можуть бути використані при розробці адаптивних агентів у сфері автономного керування, симуляцій, ігрового AI та робототехніки. Запропоновані методики масштабованого асинхронного навчання можуть слугувати основою для створення ефективних рішень у ресурсно обмежених обчислювальних умовах.

ABSTRACT

Bachelor’s thesis contains: 89 pp., 27 fig., 2 tabl., 1 ann., 24 references.

A3C, ACTOR-CRITIC, ADVANTAGE ACTOR-CRITIC, ASYNCHRONY, DEEP LEARNING, DEEP Q-NETWORK, NEURAL NETWORK, PARALLEL COMPUTING, REINFORCEMENT LEARNING.

The object of the research is the process of training agents in reinforcement learning tasks within complex dynamic environments.

The subject of the research is asynchronous reinforcement learning algorithms, in particular actor-critic methods with multithreaded implementation.

The purpose of the work is to investigate the efficiency, scalability, and stability of asynchronous actor-critic algorithms using A3C as an example, in comparison with other classical reinforcement learning algorithms.

The research methods include experimental modeling in simulation environments such as Atari, TORCS, and Labyrinth, implementation of a multithreaded training architecture, and comparison based on performance, stability, and sensitivity to hyperparameters.

The study demonstrates that the A3C algorithm provides higher efficiency and stability compared to single- and multi-step Q-learning and SARSA-based methods; the novelty lies in a comprehensive comparison of asynchronous approaches under varying numbers of threads and different environmental conditions.

The results of the work can be used in the development of adaptive agents for autonomous control, simulation, game AI, and robotics. The proposed scalable asynchronous learning techniques can serve as a foundation for creating efficient solutions under resource-constrained computational conditions.

ЗМІСТ

Перелік умовних позначень, символів, одиниць, скорочень і термінів 9
Вступ 10
1 Аналіз предметної галузі та постановка задачі 12
1.1 Reinforcing Learning 12
1.1.1 Основні компоненти Reinforcing Learning 13
1.1.2 Загальна класифікація RL-алгоритмів 14
1.1.3 Приклади застосування Reinforcing Learning 16
1.2 Deep Reinforcement Learning 17
1.2.1 Класичні табличні підходи: Q-learning, Sarsa 18
1.2.2 Поява глибоких нейронних мереж у Reinforcing Learning 20
1.2.3 Deep Q-Network і проблема нестабільності 20
1.3 Проблеми та обмеження традиційного підходу 22
1.3.1 Кореляція зразків і нестабільність оновлень 23
1.3.2 Використання Experience Replay 24
1.3.3 Витрати на обчислювальні ресурси 26
1.4 Асинхронні методи Deep Reinforcing Learning 26
1.4.1 Основні принципи асинхронного оновлення 28
1.4.2 Роль паралельних потоків та декореляції 30
1.4.3 Переваги над replay-based методами 31
1.5 Огляд асинхронних алгоритмів Deep Reinforcing Learning 32
1.5.1 Asynchronous one-step Q-learning 33
1.5.2 Asynchronous one-step Sarsa 34
1.5.3 Asynchronous n-step Q-learning 34
1.5.4 Asynchronous Advantage Actor-Critic 35
1.6 Постановка задачі 36
2 Теоретичні дослідження 37
2.1 Класичні value-based та on-policy алгоритми 37
2.1.1 Математична основа Q-learning 37

2.1.2 Алгоритм Sarsa і його on-policy природа 38
2.1.3 Проблеми нестабільності 40
2.2 Класичні архітектури DRL 41
2.2.1 Архітектура Deep Q-Network 41
2.2.2 Архітектура Actor-Critic 42
2.2.3 Модифікації: Double DQN, Dueling DQN 44
2.3 N-крокові алгоритми у навчанні з підкріпленням 45
2.3.1 Мотивація використання n-step backup 46
2.3.2 Порівняння one-step і n-step схем 47
2.3.3 Роль n-step у довготривалому навчанні 49
2.4 Архітектура actor-critic та функція переваги 49
2.4.1 Основи actor-critic: розділення політики і цінності 50
2.4.2 Визначення і роль функції переваги A(s, a) 51
2.4.3 Переваги actor-critic архітектури у складних задачах 53
2.5 Асинхронна багатопотокова архітектура 53
2.5.1 Принципи паралельного навчання 54
2.5.2 Глобальна модель і локальні копії 55
2.5.3 Особливості оновлення параметрів у потоках 56
2.6 Алгоритм Asynchronous Advantage Actor-Critic 57
2.6.1 Обчислення функції втрат: політика, цінність, ентропія 58
2.6.2 Аспекти реалізації A3C: використання LSTM, ентропійна регуляризація 59
3 Експериментальні дослідження 62
3.1 План експериментів 62
3.2 Обрані середовища для проведення експериментів 63
3.2.1 Arcade Learning Environment 64
3.2.2 TORCS 64
3.2.3 Labyrinth 65
3.3 Вибір алгоритму оптимізації 66
3.4 Конфігурація експериментального середовища 68

3.5 Аналіз результатів експериментів 70
3.5.1 Ігри Atari 2600 70
3.5.2 3D-симулятор автоперегонів TORCS 73
3.5.3 3D-середовище Labyrinth 75
3.5.4 Масштабованість та ефективність використання даних 77
3.5.5 Надійність і стабільність 83
Висновки 85
Перелік джерел посилання 87
Додаток А Відомість кваліфікаційної роботи 89

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

A3C – Asynchronous Advantage Actor-Critic – асинхронний метод з перевагою в actor-critic архітектурі;

CPU – Central Processing Unit – центральний процесор;

DQN – Deep Q-Network – глибока Q-мережа;

DRL – Deep Reinforcement Learning – глибоке навчання з підкріпленням;

GPU – Graphics Processing Unit – графічний процесор;

LSTM – Long Short-Term Memory – довга короткочасна пам’ять;

RL – Reinforcement Learning – навчання з підкріпленням;

SARSA – State-Action-Reward-State-Action – алгоритм на основі політики з оновленням за дійсною дією;

SGD – Stochastic Gradient Descent – стохастичний градієнтний спуск;

TORCS – The Open Racing Car Simulator – відкритий симулятор автоперегонів.

ВСТУП

У сучасному етапі розвитку машинного навчання та штучного інтелекту особливу увагу привертає підгалузь навчання з підкріпленням, яка формує основу для розробки автономних агентів, здатних приймати рішення у складних, динамічних і лише частково відомих середовищах. Одним із важливих проривів у цій сфері стала поява алгоритмів Deep Q-Network , які вперше дозволили агентам досягати рівня людини у класичних відеоіграх Atari, оперуючи лише вхідним піксельним зображенням. Проте, попри їхню ефективність, такі методи мають низку суттєвих обмежень, серед яких найвагоміше – потреба в потужних обчислювальних ресурсах, схильність до нестабільного навчання та висока чутливість до вибору гіперпараметрів.

У відповідь на ці виклики було запропоновано новий клас підходів – асинхронні методи навчання з підкріпленням, які дозволяють уникати використання централізованого буфера досвіду, спрощують паралельне навчання декількох агентів і загалом забезпечують більш стабільну динаміку оптимізації. Найвідомішим представником цього класу є алгоритм Asynchronous Advantage Actor-Critic, який об’єднує в собі actor-critic архітектуру, використання паралельних потоків, багатокрокове оновлення градієнтів і ентропійну регуляризацію. Завдяки цим характеристикам, A3C здатен досягати кращих результатів за значно коротший час і без використання GPU, що робить його особливо привабливим для застосувань у середовищах з обмеженими обчислювальними ресурсами. Крім того, архітектура A3C відкрила нові можливості для дослідження рекурентних нейронних мереж, які здатні зберігати інформацію про попередні стани та забезпечувати агенту пам’ять у частково спостережуваних середовищах.

Актуальність цієї кваліфікаційної роботи зумовлена необхідністю пошуку більш ефективних, стабільних і обчислювально доступних методів навчання з підкріпленням, які можна було б масштабувати до реальних

сценаріїв без залежності від надмірно потужного обладнання. У багатьох практичних задачах, таких як автономне керування, адаптивне планування, медичне моделювання або управління процесами в умовах невизначеності, критично важливо, щоб агент міг навчатися ефективно, швидко адаптуватися до нових ситуацій і водночас працювати в середовищі з високою динамікою.

Ціль даної роботи полягає у дослідженні ефективності використання асинхронних алгоритмів навчання з підкріпленням, зокрема на прикладі A3C, а також порівнянні його продуктивності з базовими value-based підходами, такими як Q-learning, SARSA та n-step Q-learning, в умовах різних середовищ. Було поставлено завдання реалізувати модифікації цих алгоритмів у багатопотоковій архітектурі, оцінити їхню стабільність, швидкість збіжності, чутливість до гіперпараметрів і масштабованість при зміні кількості потоків. Для цього в експериментальному розділі використано три типи середовищ, які відрізняються за складністю: класичні ігри Atari, 3D симулятор автоперегонів TORCS і генеративне середовище Labyrinth.

Результати цієї роботи мають практичну цінність для широкого кола задач, пов’язаних з адаптивним агентним керуванням. Алгоритми, що були протестовані та проаналізовані в цій роботі, можуть бути використані для побудови автономних систем у сферах робототехніки, автономного транспорту, оптимізації в енергетичних мережах, ігрової індустрії та віртуальних середовищ для навчання та тестування. Крім того, запропоновані методи є потенційною основою для майбутніх досліджень у напрямку розподіленого навчання з підкріпленням та інтеграції асинхронних підходів з іншими глибокими архітектурами, такими як трансформери чи графові нейронні мережі.

1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Reinforcing Learning

Навчання з підкріпленням або Reinforcement Learning є одним з основних підходів у сфері машинного навчання, який ґрунтується на моделі взаємодії агента з середовищем. У цій моделі агент приймає рішення, виконуючи певні дії, які змінюють стан середовища, і на основі результатів отримує винагороду. Метою агента є побудова такої політики дій, яка максимізує очікувану сумарну винагороду за всю траєкторію взаємодії.

Основні компоненти RL-системи (рисунок 1.1) включають агента, середовище, простір станів, простір дій, функцію винагороди, а також механізм спостереження за змінами в середовищі.

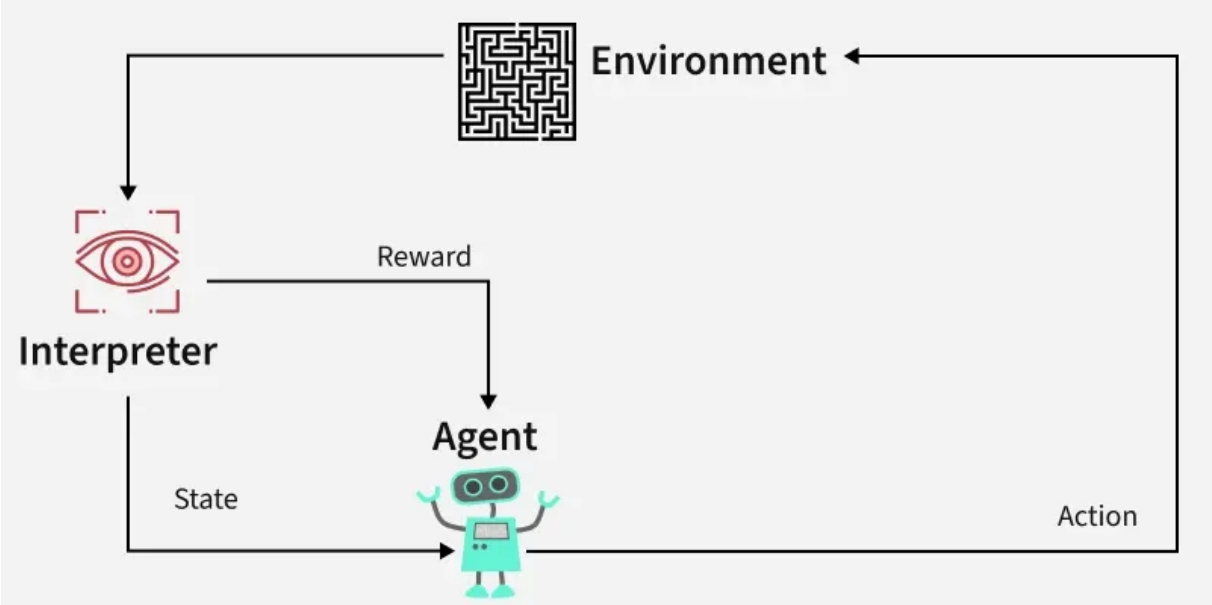

Рисунок 1.1 – Схема RL моделі

Залежно від реалізації, агент може отримувати як повну, так і часткову інформацію про стан середовища. У випадках, коли вхідна інформація є складною для безпосереднього інтерпретування (наприклад, зображення),

до архітектури додається спеціалізований інтерпретатор, який перетворює необроблені спостереження на осмислені стани.

Типова RL-модель передбачає, що агент, перебуваючи у певному стані, приймає дію, яка призводить до нового стану середовища. Середовище, у свою чергу, надає агенту сигнал винагороди, який використовується для оновлення політики дій. Цей процес повторюється багаторазово, дозволяючи агенту навчатися на основі отриманого досвіду.

1.1.1 Основні компоненти Reinforcing Learning

Основна структура моделі навчання з підкріпленням складається з шести (рисунок 1.2) ключових компонентів: агент, середовище, стан, дія, винагорода та алгоритм.

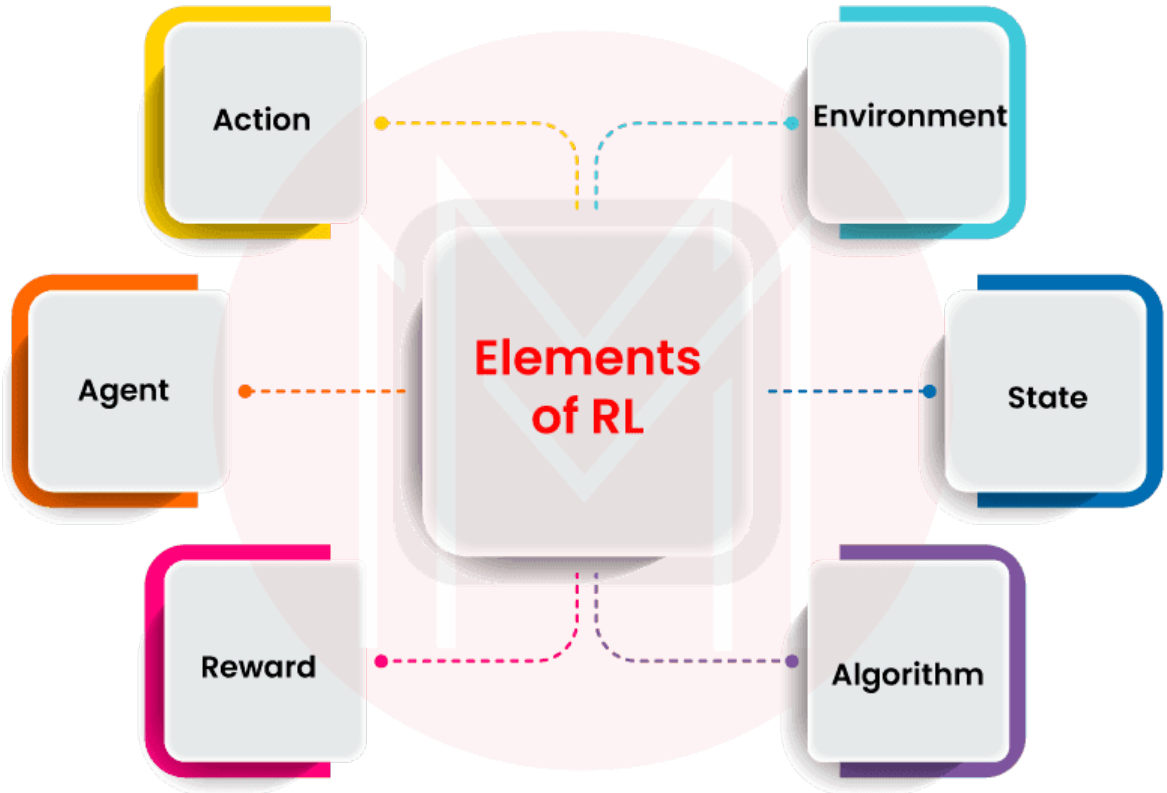

Рисунок 1.2 – Ключові елементи Reinforcing Learning

Кожен із них виконує специфічну функцію, що забезпечує повноцінну динаміку взаємодії у системі. Ці елементи формують замкнене коло, у якому агент взаємодіє з середовищем, отримуючи від нього сигнали зворотного зв’язку у вигляді винагород і спостережень.

Агент є центральною фігурою в системі RL. Це програмна сутність або модель, яка приймає рішення щодо дій у середовищі. Він має обмежене уявлення про навколишню ситуацію, яке формалізується у вигляді стану. Цей стан формується на основі спостережень, що надходять з середовища, та слугує вхідними даними для визначення наступної дії. Завдання агента полягає в тому, щоб навчитися обирати дії таким чином, аби максимізувати сумарну винагороду в довгостроковій перспективі.

Середовище представляє собою контекст, у якому функціонує агент. Воно приймає дії агента, оновлює внутрішній стан і повертає йому новий стан разом із числовим значенням винагороди. Саме середовище визначає динаміку переходів між станами та структуру функції винагороди. Агент не має прямого доступу до внутрішнього устрою середовища, що моделює реальні умови часткової або повної невизначеності.

Алгоритм навчання визначає, як агент оновлює свою політику дій на основі досвіду, отриманого через взаємодію зі середовищем. Це може бути value-based підхід (наприклад, Q-learning), policy-based (наприклад, REINFORCE) або гібридна actor-critic архітектура. Обраний алгоритм безпосередньо впливає на ефективність, швидкість та стабільність навчання агента. Усі компоненти системи навчання з підкріпленням тісно взаємопов’язані та формують загальну поведінкову модель.

1.1.2 Загальна класифікація RL-алгоритмів

Алгоритми навчання з підкріпленням поділяються на кілька типів залежно від того, як саме вони формують та оновлюють політику агента. Найбільш поширеною класифікацією є поділ на value-based, policy-based та

actor-critic підходи. Value-based алгоритми навчають функцію цінності, з якої потім непрямо виводиться політика. Policy-based методи напряму оптимізують політику без побудови функції цінності. Actor-critic архітектури поєднують обидва підходи, використовуючи окремі мережі для політики (actor) та оцінки стану (critic).

Окрім цього, існує інша важлива класифікація – за типом використання зібраного досвіду. Тут розрізняють on-policy та off-policy алгоритми. On-policy методи навчаються на даних, отриманих виключно від поточної політики. Це означає, що політика, яка виконує дії, і політика, яка навчається, є ідентичними. Такий підхід дозволяє чітко слідкувати за ефективністю змін політики, але вимагає стабільного і частого оновлення.

Off-policy методи, навпаки, дозволяють навчатись на досвіді, зібраному іншими або попередніми політиками. Це дає змогу використовувати буфер досвіду для багаторазового відтворення епізодів, зменшуючи витрати на збір нових даних. Візуально (рисунок 1.3) різницю між on-policy та off-policy можна проілюструвати через спосіб оновлення політики і повторне використання даних.

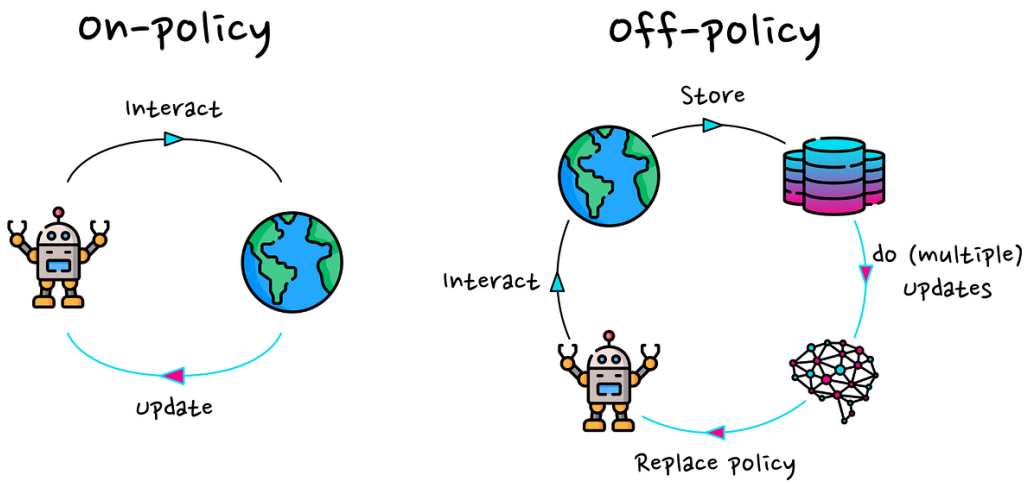

Рисунок 1.3 – Різниця між on-policy та off-policy методами

У першому випадку агент взаємодіє з середовищем і негайно оновлює політику, використовуючи лише поточний досвід. У другому – досвід зберігається в пам’яті, і політика оновлюється багаторазово на основі збережених транзакцій, навіть якщо вони були зібрані за старою політикою. Така класифікація методів навчання є фундаментальною для розуміння ефективності та обмежень різних алгоритмів у системах RL.

1.1.3 Приклади застосування Reinforcing Learning

Навчання з підкріпленням застосовується в багатьох сферах, де необхідно приймати послідовні рішення в умовах невизначеності. Одним із найбільш популярних напрямів є управління автономними агентами, зокрема у відеоіграх, симуляторах або реальних робототехнічних системах.

Наприклад, агенти RL здатні досягати надлюдської продуктивності в іграх Atari або навчатися стратегії гри в шахи. Завдяки здатності оптимізувати довгострокові нагороди, RL активно використовується в системах навігації, управління трафіком, адаптивному плануванні та виробничих процесах.

Окрему увагу отримують застосування RL у сфері охорони здоров’я. Тут алгоритми можуть допомагати у розробці персоналізованих стратегій лікування, адаптивній дозуванні ліків або оптимізації розкладу процедур. Важливою перевагою є здатність моделей враховувати як короткостроковий ефект від дій, так і довгостроковий вплив на стан пацієнта.

Завдяки гнучкості та здатності до самонавчання, RL є потужним інструментом для вирішення широкого спектру практичних задач. Він дозволяє агентам адаптуватися до нових умов, удосконалювати свою поведінку в реальному часі та знаходити оптимальні стратегії в складних динамічних середовищах.

1.2 Deep Reinforcement Learning

Глибоке навчання з підкріпленням або Deep Reinforcement Learning є поєднанням двох напрямів машинного навчання – глибокого навчання та класичного навчання з підкріпленням. Його основна ідея полягає в тому, щоб використовувати глибокі нейронні мережі як апроксиматори функцій політики або функції цінності. Це дозволяє працювати з високорозмірними та складними просторами станів, де класичні табличні методи виявляються неефективними або зовсім непридатними.

Ключовою особливістю DRL є здатність автоматично виділяти важливі ознаки із сирих даних, зокрема візуальної інформації у вигляді зображень або відео. На відміну від класичних RL-алгоритмів, які працюють з попередньо визначеними векторами станів, DRL здатен навчатися безпосередньо на основі необроблених спостережень. Це відкриває можливості для застосування в задачах комп’ютерного зору, автономного керування та медичної діагностики, де стан не подається явно, а формується зі складних сенсорних даних.

Загальна схема (рисунок 1.4) функціонування агента у DRL передбачає, що вхідний стан передається на вхід нейронної мережі, яка формує політику дій у вигляді розподілу ймовірностей. Цей розподіл визначає, яку дію обрати в поточному стані. Далі дія виконується в середовищі, результатом чого є новий стан і числова винагорода. Отримані дані використовуються для оновлення параметрів нейромережі з метою покращення політики агента. У процесі навчання мережа поступово адаптується до умов середовища, намагаючись максимізувати довгострокову вигоду. Такий підхід дозволяє агенту розвивати складні поведінкові стратегії навіть у ситуаціях з частковим або шумним зворотним зв’язком.

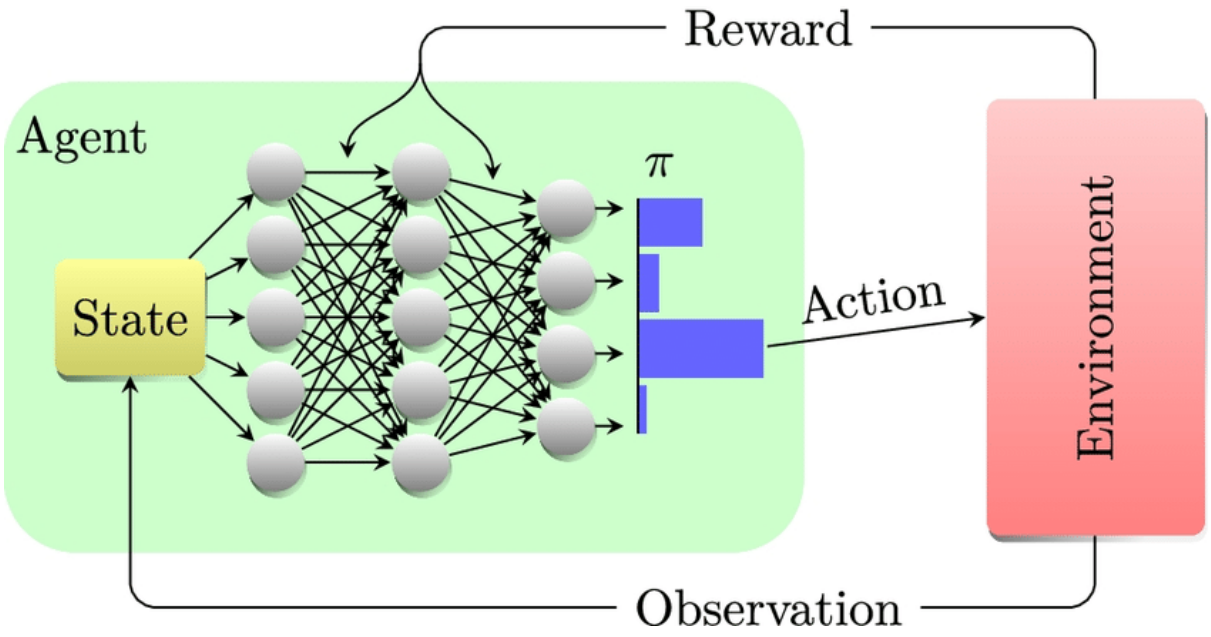

Рисунок 1.4 – Схема DRL моделі

Глибоке навчання з підкріпленням стало основою для багатьох сучасних алгоритмів, таких як Deep Q-Network (DQN), Asynchronous Advantage Actor-Critic (A3C), Deep Deterministic Policy Gradient та інші. Усі ці алгоритми мають різні підходи до оновлення політики та оцінки станів, але об’єднані використанням глибоких моделей. DRL дозволив значно розширити можливості RL, зробивши його застосовним у складних завданнях реального світу з великими просторами дій та станів.

1.2.1 Класичні табличні підходи: Q-learning, Sarsa

Класичні табличні методи навчання з підкріпленням були першими ефективними підходами до побудови агентів, які навчаються шляхом взаємодії з середовищем. Ці методи базуються на тому, що функція цінності або функція вигоди зберігається у вигляді таблиці, де кожна комірка відповідає певній парі «стан – дія». Таблиця поступово заповнюється через багаторазове оновлення значень на основі винагород, отриманих за

виконання дій. До найбільш відомих алгоритмів цього типу належать Q-learning та Sarsa.

Алгоритм Q-learning (рисунок 1.5) є прикладом off-policy методу, де агент навчається за допомогою Q-функції, яка оцінює вигідність кожної можливої дії в кожному стані. Q-функція зберігається у вигляді Q-таблиці, де кожен запис відповідає певному значенню Q(s, a). У процесі навчання ця таблиця оновлюється згідно з формулою Беллмана, яка враховує поточну винагороду і максимальне значення Q у наступному стані. Агент намагається знайти оптимальну політику шляхом вибору дій, що максимізують значення Q.

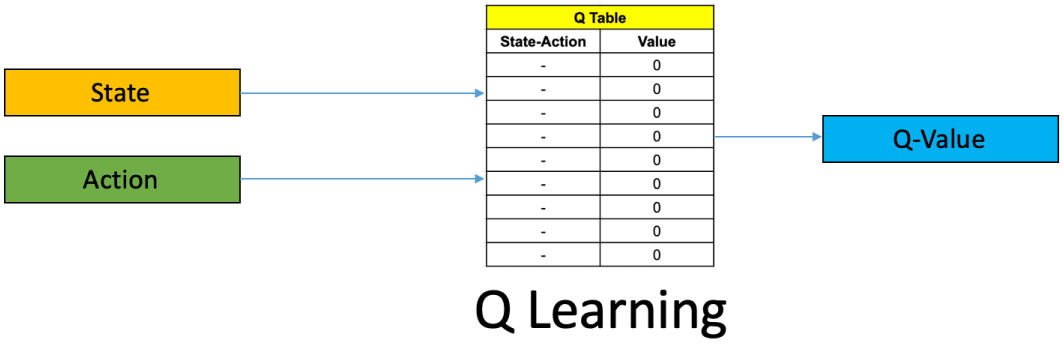

Рисунок 1.5 – Схематичне зображення Q-Learning

Алгоритм Sarsa, на відміну від Q-learning, є on-policy методом і оновлює свою Q-функцію, виходячи з тієї дії, яку фактично обрав агент. Це забезпечує більш обережну стратегію навчання, яка ґрунтується на досвіді з використанням поточної політики. У Sarsa також використовується таблиця Q(s, a), але цільове значення формується не на основі найкращої дії у наступному стані, а на основі дії, яку реально було виконано. Такий підхід дозволяє краще узгоджувати навчання з реальною поведінкою агента.

Обмеження табличних методів полягає в тому, що вони не масштабуються до середовищ із великими або неперервними просторами станів і дій. У таких випадках таблиця Q стає занадто великою або взагалі

неможливою для збереження і оновлення. Це призвело до необхідності використання апроксиматорів функції вигоди, зокрема глибоких нейронних мереж. Саме ця обмеженість табличних підходів стала рушійною силою переходу до глибокого навчання з підкріпленням.

1.2.2 Поява глибоких нейронних мереж у Reinforcing Learning

Поява глибоких нейронних мереж у навчанні з підкріпленням стала ключовим проривом, що дозволив ефективно вирішувати задачі з великими просторами станів і дій. На відміну від табличних методів, які потребують явного збереження всіх можливих комбінацій станів і дій, нейронні мережі здатні узагальнювати знання і наближати функцію вигоди або політику на основі спостережень. Це відкриває можливості для роботи з вхідними даними високої розмірності, зокрема зображеннями, звуком або сенсорними сигналами.

Глибокі моделі, такі як згорткові або рекурентні нейронні мережі, дозволяють автоматично виділяти ознаки із сирих спостережень, що особливо корисно у випадках, коли стан середовища не може бути заданий вручну. Такий підхід значно покращив результати агентів у складних задачах, таких як керування транспортом, гра у відеоігри чи обробка медичних зображень. Роль глибоких нейромереж у RL не обмежується лише функцією апроксимації – вони також виступають як основа для побудови стратегії дослідження середовища та навчання в умовах часткової спостережуваності.

1.2.3 Deep Q-Network і проблема нестабільності

Одним із перших успішних алгоритмів, що поєднав глибоке навчання з підкріпленням, став Deep Q-Network (DQN), запропонований дослідниками компанії DeepMind. Його основна ідея полягає у використанні

нейронної мережі для апроксимації Q-функції, яка приймає на вхід стан середовища і повертає значення Q для всіх можливих дій. Це дозволяє агенту оцінювати якість дій навіть у середовищах з великою кількістю станів, де табличне представлення стало б неможливим або неефективним.

Архітектура DQN передбачає подачу стану на вхід глибокої нейронної мережі, яка на виході формує вектор значень Q для кожної з можливих дій. Агент обирає дію з найбільшим значенням Q (рисунок 1.6), або досліджує нові дії з певною ймовірністю, що реалізується за допомогою стратегії epsilon-жадібності. У процесі навчання параметри мережі оновлюються таким чином, щоб значення Q наближалося до цільового значення, що обчислюється з використанням формули Беллмана.

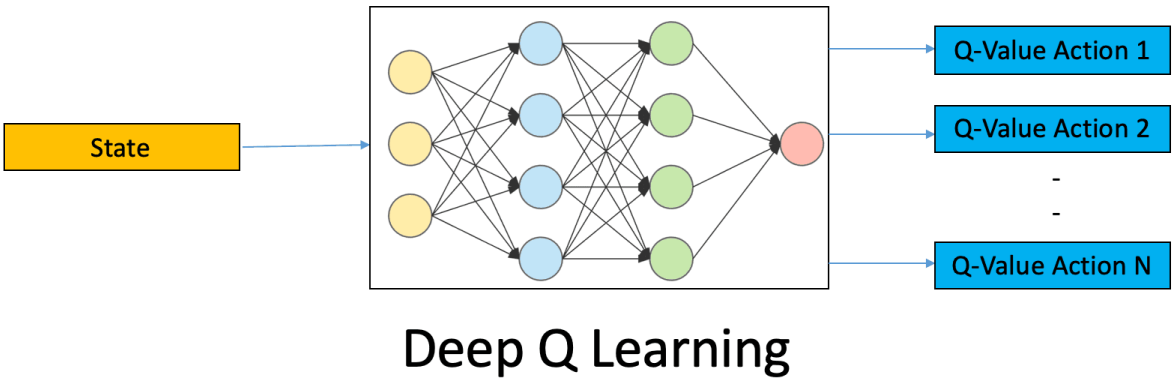

Рисунок 1.6 – Схематичне зображення Deep Q-Learning

Для стабілізації процесу навчання DQN застосовує дві ключові техніки: використання буфера досвіду та цільової мережі. Буфер дозволяє зберігати транзакції агентів та навчати мережу на випадково вибраних міні-батчах, що зменшує кореляцію між послідовними зразками. Цільова мережа є копією основної мережі з фіксованими вагами, які періодично оновлюються. Це допомагає уникнути нестабільності, викликаної швидкими змінами у функції цілі.

Незважаючи на значний успіх, DQN має низку обмежень. По-перше, він чутливий до вибору гіперпараметрів і вимагає великої кількості епізодів для досягнення стабільного результату. По-друге, алгоритм є off-policy методом, а отже не може бути безпосередньо застосований до задач, де політика повинна змінюватися адаптивно під час навчання. Ці недоліки стали поштовхом до створення нових, більш ефективних підходів, таких як Double DQN, Dueling DQN та, зокрема, асинхронні actor-critic методи, які не потребують буфера досвіду.

1.3 Проблеми та обмеження традиційного підходу

Попри значні досягнення, традиційні підходи до навчання з підкріпленням, зокрема ті, що базуються на глибоких нейронних мережах, мають низку серйозних обмежень. Однією з головних проблем є нестабільність процесу навчання, яка виникає внаслідок сильної кореляції між послідовними зразками досвіду агента. Оскільки нейронна мережа постійно оновлюється на основі даних, отриманих від поточної політики, малі зміни у параметрах можуть спричинити великі відхилення у поведінці, що ускладнює конвергенцію моделі.

Ще однією важливою проблемою є потреба у великій кількості досвіду, який необхідно зберігати, обробляти і повторно використовувати. Саме для цього в таких алгоритмах, як DQN, запроваджується буфер досвіду, experience replay, який дозволяє використовувати старі транзакції для багатократного оновлення мережі. Хоча це й покращує стабільність, але унеможливлює використання on-policy методів, які потребують оновлення лише на актуальних зразках. Крім того, досвід replay-буфера не завжди добре репрезентує реальний розподіл траєкторій, особливо у складних або динамічних середовищах.

Високі обчислювальні вимоги також є серйозним викликом для традиційних підходів. Використання глибоких моделей разом із replay-

буфером і цільовими мережами часто вимагає потужних графічних процесорів (GPU) або кластерів. Це обмежує застосування таких методів у сценаріях з обмеженими ресурсами або в реальному часі. Саме ці обмеження стали мотивацією для пошуку альтернатив, які дозволяють забезпечити стабільність, ефективність і адаптивність без потреби у складних зовнішніх механізмах, що й привело до розробки асинхронних методів навчання з підкріпленням.

1.3.1 Кореляція зразків і нестабільність оновлень

Однією з ключових причин нестабільності у процесі навчання з підкріпленням є висока кореляція між послідовними зразками, які збирає агент під час взаємодії зі середовищем. Коли агент досліджує середовище, його дії та отримані стани часто змінюються поступово, утворюючи послідовності з сильно залежними даними. Навчання нейронної мережі на таких корельованих прикладах призводить до зміщення градієнтних оцінок і знижує ефективність оновлення параметрів. Це, в свою чергу, може викликати дивергентну поведінку або повільну конвергенцію політики.

Класичні алгоритми, такі як DQN, намагаються вирішити цю проблему за допомогою буфера досвіду, з якого дані вибираються у випадковому порядку. Однак такий підхід не завжди знижує кореляцію достатньо ефективно, особливо у складних середовищах або при коротких епізодах. Крім того, накопичення і зберігання великої кількості досвіду може бути ресурсомістким. Таким чином, проблема корельованості зразків залишається актуальною і вимагає застосування альтернативних механізмів декореляції, зокрема через паралельні потоки або незалежне збирання досвіду, як це реалізовано в асинхронних методах. Як наслідок, агент отримує більш стабільний і репрезентативний потік даних для оновлення політики, що позитивно впливає на швидкість і якість навчання.

1.3.2 Використання Experience Replay

Experience replay (рисунок 1.7) є однією з найважливіших технік, що використовуються для стабілізації навчання в алгоритмах глибокого підкріплення. Її суть полягає в тому, що досвід агента зберігається у спеціальному буфері, з якого вибірки беруться випадковим чином під час оновлення параметрів нейронної мережі. Це дозволяє розірвати часову залежність між зразками та уникнути надмірної кореляції, яка шкодить стабільному навчанню. Завдяки цьому агент отримує більш різноманітні приклади для навчання, навіть якщо поточне середовище тимчасово стало менш інформативним.

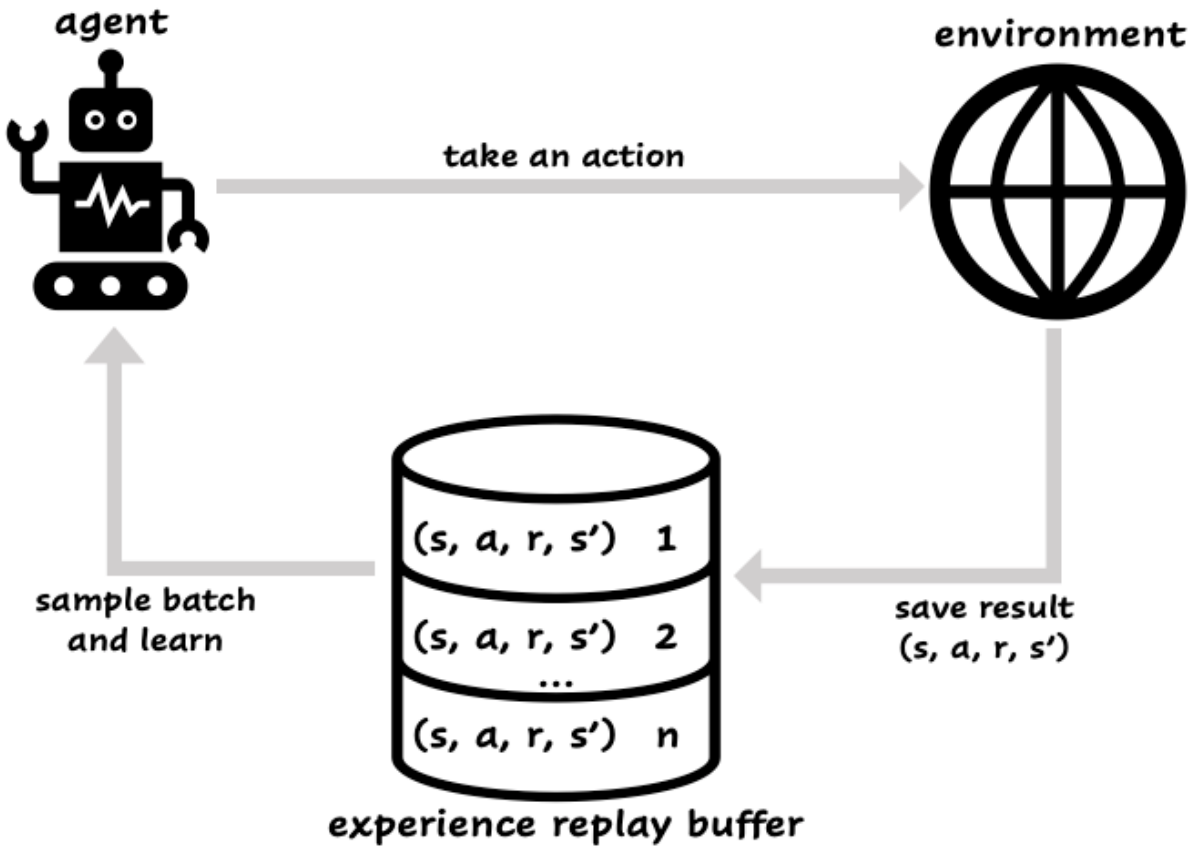

Рисунок 1.7 – Схематичне зображення Experience Replay

Experience replay є однією з найважливіших технік, що використовуються для стабілізації навчання в алгоритмах глибокого підкріплення. Її суть полягає в тому, що досвід агента зберігається у спеціальному буфері, з якого вибірки беруться випадковим чином під час оновлення параметрів нейронної мережі. Це дозволяє розірвати часову залежність між зразками та уникнути надмірної кореляції, яка шкодить стабільному навчанню. Завдяки цьому агент отримує більш різноманітні приклади для навчання, навіть якщо поточне середовище тимчасово стало менш інформативним.

У типовій реалізації experience replay буфер зберігає кортежі виду (s, a, r, s′), що відповідають стану, виконаній дії, отриманій винагороді та новому стану після дії. Під час оновлення агент вибирає випадкову підмножину таких прикладів (mini-batch), на основі яких обчислюються градієнти втрати і оновлюються параметри мережі. Це забезпечує ефективніше використання наявного досвіду і дозволяє зменшити потребу в надмірній кількості нових епізодів. Даний підхід особливо корисний у середовищах з повільною динамікою або у випадках, коли збирання нових даних є дорогим або ризикованим.

Разом із тим, використання буфера досвіду має низку обмежень. По-перше, replay buffer вимагає додаткової пам’яті для зберігання великої кількості транзакцій, що може бути критичним у великих або тривалих задачах. По-друге, старі зразки можуть поступово втрачати актуальність, що знижує ефективність навчання в динамічно змінному середовищі. Крім того, використання застарілих даних суперечить оновленню політики на основі поточного досвіду, що унеможливлює використання on-policy алгоритмів.

Незважаючи на ці недоліки, experience replay широко використовується в сучасних алгоритмах, таких як DQN та його модифікаціях. Він демонструє свою ефективність у середовищах зі статичною або слабо динамічною структурою, де повторне використання

досвіду значно пришвидшує навчання. Однак у сучасних підходах, зокрема в асинхронних методах, робиться спроба уникнути replay buffer, натомість забезпечивши декореляцію за рахунок паралельного збирання досвіду.

1.3.3 Витрати на обчислювальні ресурси

Глибокі алгоритми навчання з підкріпленням зазвичай мають високі обчислювальні вимоги, особливо при використанні складних нейронних мереж і буфера досвіду. Процес навчання передбачає багаторазові проходження через велику кількість епізодів, обробку великих обсягів даних, обчислення градієнтів та оновлення параметрів моделі. У більшості випадків для ефективного виконання таких операцій необхідне застосування графічних процесорів (GPU) або навіть обчислювальних кластерів. Це створює бар’єри для використання DRL у середовищах із обмеженими ресурсами, таких як мобільні пристрої, вбудовані системи чи реальний час.

Крім того, потреба у тривалому навчанні та значних обчислювальних витратах ускладнює повторюваність експериментів та адаптацію моделей до нових задач. Традиційні методи часто не враховують змінність середовища і вимагають повного перенавчання при найменших змінах у структурі даних або функції винагороди. У відповідь на ці виклики сучасні дослідження фокусуються на пошуку більш ефективних алгоритмів, зокрема асинхронних методів, які дозволяють зменшити залежність від обчислювальних ресурсів без втрати якості навчання.

1.4 Асинхронні методи Deep Reinforcing Learning

Асинхронні методи навчання з підкріпленням стали відповіддю на ключові обмеження традиційних алгоритмів DRL, пов’язані з нестабільністю навчання, кореляцією зразків і високими обчислювальними

витратами. Основна ідея таких методів (рисунок 1.8) полягає у паралельному запуску кількох незалежних агентів, які взаємодіють із окремими копіями середовища та навчають спільну глобальну модель. Це дозволяє отримувати некорельовані приклади досвіду без використання буфера пам’яті та значно підвищує стабільність і швидкість збіжності. Кожен агент накопичує градієнти протягом певної кількості кроків і передає їх до глобальної мережі, яка оновлюється асинхронно.

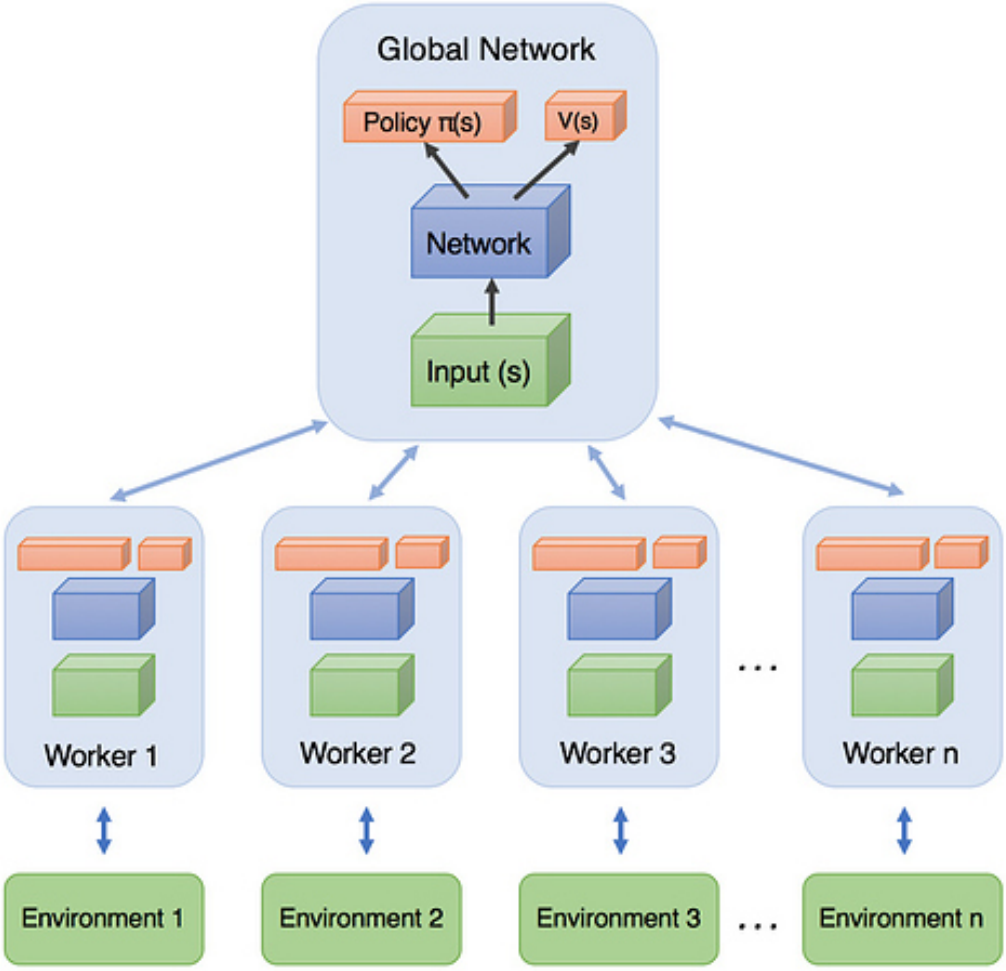

Рисунок 1.8 – Візуалізація роботи асинхронних методів DRL

У такій архітектурі використовується центральна глобальна модель, яка зберігає параметри політики і, за потреби, функції цінності. Кожен потік

має свою локальну копію цієї моделі та власне середовище, у якому він самостійно генерує досвід. Після завершення локального етапу навчання потік обчислює градієнти втрати на основі свого досвіду і оновлює глобальну модель. Паралельне оновлення з різних потоків забезпечує декореляцію та природне усереднення градієнтів, що підвищує стабільність навчання.

Асинхронні методи дозволяють позбутися необхідності у replay buffer, оскільки паралельне збирання досвіду з різних середовищ забезпечує достатню різноманітність навчальних прикладів. Крім того, вони є ефективними з погляду обчислювальних ресурсів, оскільки можуть бути реалізовані виключно на CPU, без потреби у потужному графічному процесорі. Це відкриває можливість використання таких алгоритмів на широкому колі пристроїв та у реальному часі, зокрема в мобільних роботах або вбудованих системах.

У порівнянні з традиційними методами, асинхронні підходи демонструють кращу масштабованість та прискорення процесу навчання. При цьому вони не вимагають централізованого сервера параметрів або жорсткої синхронізації між потоками, що значно спрощує їх реалізацію. Результати численних експериментів показують, що такі методи, зокрема A3C, здатні перевершувати DQN як за швидкістю навчання, так і за якістю отриманої політики. Саме завдяки поєднанню ефективності, стабільності та адаптивності асинхронні алгоритми стали важливою віхою у розвитку сучасного підходу до навчання з підкріпленням.

1.4.1 Основні принципи асинхронного оновлення

Асинхронне оновлення є центральною концепцією в асинхронних методах навчання з підкріпленням, яка дозволяє ефективно масштабувати процес навчання без втрати стабільності. Основна ідея полягає в тому, що кілька незалежних агентів (потоків) паралельно взаємодіють з окремими

екземплярами середовища, збирають досвід і обчислюють градієнти локально. Ці градієнти потім асинхронно передаються до глобальної моделі, яка оновлює свої параметри без потреби в централізованій синхронізації. Такий підхід усуває необхідність чекати завершення всіх потоків, як це відбувається в синхронному навчанні, що значно пришвидшує загальний процес оптимізації.

Під час роботи кожен агент копіює поточні параметри глобальної моделі і використовує їх для взаємодії з середовищем. Протягом певної кількості кроків агент виконує дії, накопичує стани, винагороди і оцінки, після чого обчислює градієнти втрати. Замість негайного оновлення локальної копії, агент надсилає градієнти на центральний сервер або безпосередньо застосовує їх до глобальної моделі. Оскільки оновлення відбуваються у довільному порядку та з різних джерел, параметри глобальної мережі змінюються постійно, але завдяки частоті і випадковості оновлень досягається природне згладжування впливу окремих потоків.

Такий принцип асинхронного оновлення дозволяє уникнути проблеми корельованості зразків, оскільки кожен потік отримує досвід у власному середовищі, незалежно від інших агентів. Як наслідок, глобальна модель навчається на більш різноманітному і менш залежному наборі даних. Крім того, асинхронність сприяє уникненню затримок, пов’язаних із координацією між потоками, і забезпечує більш стабільну динаміку навчання. Це особливо важливо для on-policy алгоритмів, таких як A3C, які чутливі до точності і актуальності зібраного досвіду.

Варто також зазначити, що асинхронне оновлення не потребує складної інфраструктури або централізованого буфера даних, як у випадку з experience replay. Завдяки цьому метод зберігає гнучкість і простоту реалізації, при цьому демонструючи високі показники продуктивності на широкому спектрі задач.

1.4.2 Роль паралельних потоків та декореляції

Паралельні потоки є ключовим інструментом забезпечення декореляції зразків у асинхронних методах навчання з підкріпленням. На відміну від традиційних алгоритмів, де агент послідовно взаємодіє зі середовищем і накопичує досвід у буфері, в асинхронній архітектурі багато незалежних агентів одночасно працюють із різними екземплярами середовища. Кожен агент отримує унікальний досвід, зумовлений випадковістю початкових станів, стратегій вибору дій та динамікою середовища. Це дозволяє значно зменшити кореляцію між навчальними прикладами без використання спеціальних механізмів, таких як experience replay.

Роль паралельних потоків полягає не лише у підвищенні продуктивності, а й у формуванні більш репрезентативного та різноманітного потоку даних, який використовується для навчання глобальної моделі. Завдяки цьому агент отримує стабільніші та загальніші градієнти, що сприяє швидшій і надійнішій збіжності. Кожен потік незалежно виконує серію дій, накопичує траєкторії, обчислює градієнти та оновлює загальну модель, вносячи свій внесок у колективне навчання. У поєднанні з асинхронним оновленням це створює стійкий до шуму і нестабільності процес навчання, що не потребує жорсткої синхронізації або зберігання великих обсягів історичних даних.

Декореляція, досягнута завдяки паралелізму, є критично важливою для on-policy алгоритмів, де застарілі або повторювані зразки можуть значно знижувати якість навчання. Паралельні потоки забезпечують постійне оновлення даних у реальному часі, що дозволяє агенту ефективно адаптуватися до змін у середовищі. Це особливо актуально в динамічних умовах або під час навчання на складних задачах з великою кількістю можливих дій, де традиційні методи втрачають ефективність. Завдяки

паралельному збиранню досвіду асинхронні методи демонструють високу продуктивність навіть без доступу до потужного апаратного забезпечення.

1.4.3 Переваги над replay-based методами

Асинхронні методи навчання з підкріпленням мають низку важливих переваг порівняно з підходами, що базуються на використанні буфера досвіду. Насамперед, вони дозволяють повністю відмовитися від централізованого зберігання попередніх зразків, оскільки декореляція даних досягається завдяки паралельному збиранню досвіду в різних потоках. Це усуває необхідність у додатковій пам’яті для зберігання великої кількості транзакцій і дозволяє агенту навчатися без затримок, пов’язаних із вибіркою з буфера.

Іншою важливою перевагою є можливість використання on-policy алгоритмів, які за своєю природою не сумісні з застарілими або повторними зразками. У replay-based підходах агент навчається на попередньо зібраних даних, що порушує актуальність політики під час оновлення. Асинхронні методи забезпечують використання свіжих, поточних траєкторій кожного потоку, що дозволяє ефективно реалізовувати on-policy навчання без втрати стабільності. Це відкриває шлях до використання більш чутливих до змін алгоритмів, таких як actor-critic, у складних динамічних середовищах.

Також важливою перевагою є підвищення обчислювальної ефективності. Асинхронні алгоритми, як правило, працюють на CPU і не потребують потужних GPU для обробки великих батчів з буфера досвіду. Це дозволяє масштабувати систему на велику кількість потоків без значного ускладнення інфраструктури, що робить їх привабливими для розгортання в розподілених або вбудованих системах. Крім того, така архітектура забезпечує природне пришвидшення процесу навчання за рахунок розпаралелювання.

У сукупності, асинхронні методи забезпечують простішу реалізацію, кращу адаптивність, менше залежностей від гіперпараметрів та підвищену стійкість до нестабільностей, які характерні для replay-based підходів. Вони довели свою ефективність у широкому спектрі задач, зокрема в тих, де класичні методи потребують значних обчислювальних ресурсів або зазнають труднощів через корельованість даних. Саме ці переваги зробили асинхронні методи, такі як A3C, новим стандартом у багатьох сучасних дослідженнях з підкріплювального навчання.

1.5 Огляд асинхронних алгоритмів Deep Reinforcing Learning

Асинхронні алгоритми глибокого навчання з підкріпленням стали важливим етапом у розвитку сучасних адаптивних систем, що навчаються шляхом взаємодії з середовищем. Вони поєднують у собі переваги глибоких нейронних мереж та ефективність асинхронного паралельного навчання, забезпечуючи високу швидкість збіжності та стабільність без потреби у складних допоміжних механізмах, таких як буфер досвіду. Ключова ідея полягає в тому, що кілька потоків (або агентів) паралельно досліджують середовище, обчислюють градієнти втрат та оновлюють спільну глобальну модель, яка навчається на їхньому колективному досвіді.

На основі цієї концепції було розроблено кілька варіантів алгоритмів, які відрізняються способом оновлення функції цінності, оцінкою винагороди та стратегією побудови політики. До таких алгоритмів належать асинхронні модифікації класичних підходів, зокрема one-step Q-learning, one-step Sarsa, n-step Q-learning, а також actor-critic архітектури, представлені алгоритмом A3C. Кожен із цих методів має свої особливості, але всі вони базуються на одній спільній архітектурі, де потоки навчаються незалежно та асинхронно, сприяючи ефективному декорелюванню даних та швидкій адаптації.

Асинхронні алгоритми демонструють високу продуктивність у багатьох складних задачах, зокрема в середовищах, де традиційні алгоритми не можуть забезпечити стабільне навчання. Їх гнучкість, здатність працювати на CPU та відсутність потреби в централізованих буферах роблять їх універсальним інструментом у широкому спектрі прикладних сценаріїв. У цьому підрозділі буде розглянуто чотири основні варіанти асинхронних алгоритмів DRL, які були реалізовані та протестовані у статті, що стала базовою для цієї роботи. В кожному випадку буде проаналізовано архітектурні особливості, механізм оновлення, а також сильні та слабкі сторони відповідного підходу.

1.5.1 Asynchronous one-step Q-learning

Asynchronous one-step Q-learning є модифікацією класичного алгоритму Q-learning, адаптованою до асинхронної багатопотокової архітектури. Його реалізація передбачає запуск кількох агентів, кожен з яких взаємодіє з окремим середовищем, збирає досвід та оновлює спільну модель на основі локальних обчислень. Завдяки цьому досягається ефективне декорелювання даних, що усуває потребу у зберіганні та повторному використанні транзакцій з буфера досвіду. Глобальна модель оновлюється асинхронно, що дозволяє уникнути блокувань і пришвидшує процес оптимізації.

Такий підхід забезпечує стабільність і дозволяє використовувати off-policy природу Q-learning без обмежень, характерних для replay-based реалізацій. Асинхронність також сприяє підвищенню різноманітності досвіду агентів, що позитивно впливає на збіжність політики. Хоча базовий принцип оновлення Q-функції зберігається, паралельна структура взаємодії та навчання суттєво змінює динаміку алгоритму, роблячи його більш придатним для реалізації в сучасних обчислювальних середовищах.

1.5.2 Asynchronous one-step Sarsa

Asynchronous one-step Sarsa є асинхронною реалізацією класичного on-policy алгоритму Sarsa, адаптованою до багатопотокового навчання без використання буфера досвіду. У цій архітектурі декілька потоків незалежно досліджують середовище, збирають епізодичний досвід і локально оновлюють функцію дій на основі фактично виконаних дій у наступних станах. Це дозволяє алгоритму зберігати узгодженість між політикою, яка генерує досвід, та політикою, що навчається, що є критичним для on-policy методів.

Асинхронне оновлення глобальної моделі дає змогу одночасно використовувати переваги паралельної обробки та адаптивної побудови політики. Завдяки незалежності потоків досягається природне декорелювання навчальних прикладів, що дозволяє забезпечити стабільніше навчання, порівняно з послідовними реалізаціями. Такий підхід є придатним для задач, де важливо підтримувати тісний зв’язок між поточною поведінкою агента та процесом оновлення політики.

1.5.3 Asynchronous n-step Q-learning

Asynchronous n-step Q-learning є розширенням асинхронного one-step підходу, яке дозволяє враховувати послідовності декількох кроків при обчисленні цільового значення функції дій. Така стратегія дозволяє агенту ефективніше використовувати інформацію про майбутні винагороди, покращуючи стабільність і швидкість навчання. Замість оновлення функції дій лише на основі негайної винагороди, як у one-step варіантах, n-step методи враховують кумулятивну винагороду за кілька наступних кроків, що забезпечує глибше навчання на кожному оновленні.

У межах асинхронної архітектури кілька агентів паралельно взаємодіють із середовищем, формують n-крокові траєкторії та

використовують їх для обчислення градієнтів. Такий підхід дозволяє моделі ефективніше реагувати на довгострокові наслідки дій, що є важливим для задач із відкладеними винагородами. Асинхронне оновлення параметрів на основі багатокрокових послідовностей додатково знижує залежність від окремих епізодів та підвищує загальну узагальнювальну здатність моделі. Це робить n-step Q-learning перспективним варіантом для навчання агентів у складних середовищах.

1.5.4 Asynchronous Advantage Actor-Critic

Asynchronous Advantage Actor-Critic (A3C) є найбільш комплексним і потужним представником асинхронних алгоритмів глибокого навчання з підкріпленням. Він поєднує в собі принципи actor-critic підходу, де одна нейронна мережа (actor) відповідає за побудову політики, а інша (critic) – за оцінку функції цінності станів. Головною відмінністю A3C є використання асинхронної багатопотокової архітектури, в якій декілька агентів паралельно навчаються в окремих середовищах, кожен із власною копією моделі. Це дозволяє значно покращити декореляцію зразків і забезпечити стабільне оновлення глобальної політики.

Ключовим елементом алгоритму є використання переваги (advantage), яка визначає, наскільки поточна дія краща за середнє очікуване значення у даному стані. Це дозволяє зменшити дисперсію градієнтів і пришвидшити процес навчання. Завдяки своїй on-policy природі, A3C відмовляється від буфера досвіду, покладаючись виключно на актуальні траєкторії кожного агента. Паралельне навчання забезпечує природну регуляризацію, тоді як окремі агенти можуть бути запущені з різними параметрами дослідження, що сприяє кращому покриттю простору дій.

A3C демонструє високу ефективність у задачах з візуальним входом, складною динамікою середовища та відкладеними винагородами. Його гнучка архітектура дозволяє легко адаптувати алгоритм до різних типів

задач, включаючи як дискретні, так і неперервні простори дій. Завдяки використанню згорткових і рекурентних мереж, A3C здатен працювати з неструктурованими вхідними даними та середовищами з частковою спостережуваністю. Це робить його одним з найпоширеніших і найуспішніших алгоритмів у сучасних реалізаціях навчання з підкріпленням.

1.6 Постановка задачі

Метою цієї роботи є дослідження ефективності асинхронних методів глибокого навчання з підкріпленням, зокрема алгоритмів Asynchronous one-step Q-learning, Asynchronous one-step Sarsa, Asynchronous n-step Q-learning та Asynchronous Advantage Actor-Critic. У фокусі дослідження знаходиться оцінка їх здатності забезпечувати стабільне і швидке навчання без використання буфера досвіду, а також аналіз того, наскільки асинхронне оновлення параметрів і паралельне збирання досвіду компенсують традиційні обмеження, притаманні класичним підходам. Робота також має на меті визначити, які з досліджуваних методів демонструють найкращу збіжність і якість отриманої політики в умовах складних середовищ.

Для досягнення цієї мети передбачається реалізація експериментального середовища, в якому обрані алгоритми будуть тренуватися на однакових умовах із фіксованими гіперпараметрами. На основі результатів порівняння буде зроблено висновки щодо доцільності застосування асинхронних методів у практичних задачах, зокрема тих, де використання experience replay є недоцільним або ресурсозалежним. Також буде проаналізовано потенціал масштабування таких методів на багатоядерних процесорах без потреби у GPU, що є актуальним для задач реального часу та автономних систем.

2 ТЕОРЕТИЧНІ ДОСЛІДЖЕННЯ

2.1 Класичні value-based та on-policy алгоритми

Цей підрозділ присвячено огляду класичних алгоритмів навчання з підкріпленням, які історично стали основою для подальшого розвитку більш складних і ефективних методів глибокого навчання. Тут буде розглянуто два фундаментальні підходи – off-policy алгоритм Q-learning та on-policy алгоритм Sarsa, які репрезентують два принципово різні способи оновлення політики агента. Окрема увага буде приділена формальним засадам цих алгоритмів, їхній поведінці під час навчання, а також обмеженням, з якими стикаються реалізації на основі табличних подань. Такий аналіз необхідний для розуміння того, як класичні схеми вплинули на архітектуру сучасних алгоритмів і чому виникла потреба в переході до глибоких моделей.

У підрозділі також буде проаналізовано характерні проблеми, що виникають у класичних value-based і on-policy підходах, зокрема нестабільність оновлень, проблема переоцінювання Q-функції та розрив між поведінковою і цільовою політиками. Ці аспекти важливі з огляду на подальший перехід до асинхронних і n-крокових модифікацій, які були розроблені як відповідь на зазначені недоліки. Узагальнення базових алгоритмів дозволить чітко окреслити їхню роль у формуванні архітектур DRL та підготувати теоретичний ґрунт для подальшого опису більш просунутих стратегій, таких як actor-critic або A3C.

2.1.1 Математична основа Q-learning

Q-learning є одним із базових алгоритмів навчання з підкріпленням, що реалізує off-policy стратегію оновлення політики. Його мета полягає в апроксимації оптимальної функції вигоди $Q*(s,a)$, яка вказує на очікувану

сумарну винагороду за вибір дії a в стані s з подальшим дотриманням найкращої політики. Алгоритм не зберігає політику явно, а навчається через ітеративне оновлення табличних або параметричних оцінок функції Q.

Основне оновлення у Q-learning базується на рівнянні Беллмана, яке дозволяє покроково коригувати значення функції вигоди на основі поточного досвіду. Оновлення здійснюється після кожного переходу агента в середовищі, при цьому враховується отримана винагорода та максимальна вигода з наступного стану. Це дозволяє алгоритму поступово наближатися до оптимального значення функції Q навіть без повного знання динаміки середовища. Формула має наступний вигляд:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t) \right], \qquad (2.1)$$

де $Q(s_t, a_t)$ – поточна оцінка вигоди дії a в стані s;

α – коефіцієнт навчання;

r_{t+1} – негайна винагорода;

γ – коефіцієнт дисконтування;

$\max_a Q(s_{t+1}, a)$ – прогнозована вигода найкращої дії в наступному стані.

Таким чином, Q-learning забезпечує ефективне навчання політики шляхом поступового оновлення оцінок вигоди, що робить його універсальним інструментом для задач із дискретним простором дій. Незважаючи на простоту, саме цей алгоритм став фундаментом для більш складних моделей, включаючи його глибокі та асинхронні модифікації.

2.1.2 Алгоритм Sarsa і його on-policy природа

Sarsa є представником on-policy алгоритмів, тобто таких, що оновлюють оцінку функції вигоди на основі дій, які реально були виконані агентом у відповідності до поточної політики. Це означає, що політика, яка

використовується для вибору дій, одночасно є й політикою, за якою відбувається навчання. Такий підхід дозволяє краще враховувати фактичну поведінку агента і зменшує ризик некоректних оцінок, які можуть виникнути у випадках, коли політика постійно змінюється.

На відміну від Q-learning (рисунок 2.1), де оцінка функції вигоди відбувається на основі максимальної вигоди в наступному стані, Sarsa враховує конкретну дію, яка була обрана агентом у цьому стані. Завдяки цьому оновлення є більш консервативним і стабільним, особливо на ранніх етапах навчання. Така особливість робить алгоритм менш чутливим до флуктуацій у середовищі, хоча іноді це може уповільнити збіжність до оптимальної політики.

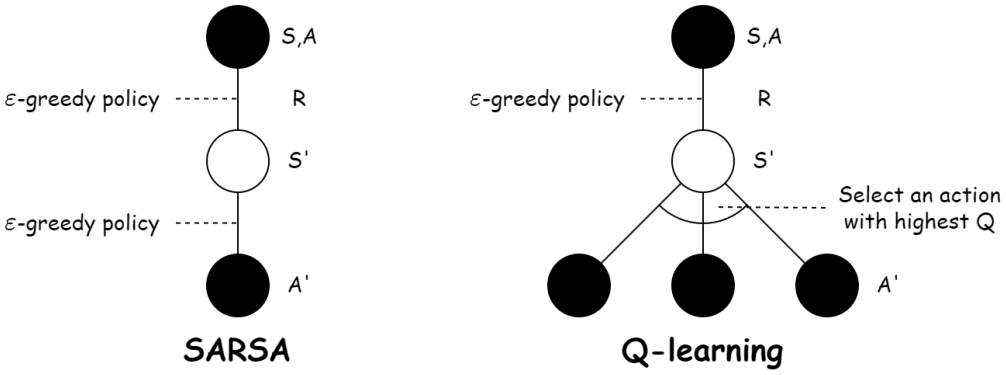

Рисунок 2.1 – Різниця між Sarsa та Q-Learning

На рисунку 2.1 представлено схематичне порівняння між двома підходами. Ліворуч показано, як у Sarsa дія в наступному стані обирається згідно з тією ж політикою, що й дія в поточному. Праворуч ілюструється механізм Q-learning, у якому наступна дія не залежить від поведінки агента, а вибирається як найкраща згідно з поточними оцінками. Це чітко демонструє ключову різницю між on-policy та off-policy навчанням.

2.1.3 Проблеми нестабільності

Хоча алгоритми Q-learning та Sarsa є фундаментальними для навчання з підкріпленням, обидва підходи мають ряд обмежень, що суттєво впливають на стабільність та ефективність навчання. Однією з основних проблем є так званий «розрив між політиками» – ситуація, коли політика, яка генерує досвід, не збігається з тією, яка оновлюється. У off-policy алгоритмах, таких як Q-learning, це призводить до того, що навчання базується на гіпотетичних діях, а не на реальних діях агента, що може викликати нестабільність і уповільнити процес збіжності.

Ще однією відомою проблемою є переоцінювання значень Q-функції. Оскільки оновлення базується на максимальному значенні серед можливих дій, навіть незначний шум або помилки в оцінках можуть призвести до систематичного завищення очікуваних винагород. У складних середовищах з великою кількістю можливих дій це призводить до некоректного пріоритету дій, що уповільнює процес навчання або навіть повністю зупиняє його. Ця проблема особливо актуальна для алгоритмів, що не використовують додаткові стабілізаційні техніки.

Також важливо враховувати обмеження, які будуть пов’язані з однокроковим (one-step) оновленням, яке обмежує агент у здатності враховувати довготривалі наслідки своїх дій. У випадках, коли винагорода настає лише через декілька кроків, агент не здатен ефективно передати сигнал зворотного зв’язку назад до початкового стану. Це призводить до повільного накопичення корисної інформації про середовище і створює додаткові виклики у навчанні. Усі ці обмеження мотивують до розробки покращених методів, таких як Double DQN, Dueling DQN, n-step підходи та, зокрема, асинхронні архітектури, які будуть розглянуті у наступних підрозділах.

2.2 Класичні архітектури DRL

Даний підрозділ зосереджений на аналізі базових архітектур глибокого навчання з підкріпленням, які стали основою для побудови сучасних алгоритмів DRL. Тут буде розглянуто принципову будову Deep Q-Network, що адаптує класичний Q-learning до високорозмірних просторових ознак за допомогою глибоких нейронних мереж. Також буде описано архітектуру actor-critic, яка поєднує компоненти політики та цінності в єдиній системі, забезпечуючи більшу гнучкість у задачах з частковою спостережуваністю або неперервними просторами дій. Розуміння цих архітектур є критично важливим для обґрунтування вибору тих чи інших алгоритмів у подальших експериментах.

Окремий акцент буде зроблено на модифікаціях, які були розроблені з метою подолання слабких сторін базових архітектур. Зокрема, йдеться про алгоритми Double DQN і Dueling DQN, які покращують стабільність та точність оцінювання дій, зменшуючи переоцінювання і розділяючи обчислення цінності стану та переваги дії. Вивчення цих архітектур дозволить не лише глибше зрозуміти еволюцію DRL, але й створить передумови для переходу до асинхронних і багатопотокових підходів, у яких закладені ті ж принципи, але реалізовані в більш ефективний спосіб.

2.2.1 Архітектура Deep Q-Network

Архітектура Deep Q-Network (DQN) була запропонована як розв'язання проблеми неможливості зберігати значення Q-функції у вигляді таблиці при великій розмірності простору станів. Замість табличного подання, DQN використовує глибоку нейронну мережу, яка наближає функцію вигоди. На вхід мережі подається представлення поточного стану, а на виході формуються значення вигоди для всіх можливих дій. Це

дозволяє агенту ефективно приймати рішення навіть у складних середовищах з візуальними або неперервними даними.

Однією з ключових особливостей DQN є розділення навчання та вибору дій. Мережа оновлюється за допомогою алгоритму Q-learning, проте значення майбутньої вигоди береться з копії мережі – цільової (target network), яка оновлюється із затримкою. Це зменшує нестабільність, спричинену швидкими змінами функції цілі. Також DQN використовує буфер досвіду (experience replay), з якого випадковим чином вибираються приклади для навчання, що розриває часову кореляцію між зразками.

2.2.2 Архітектура Actor-Critic

Архітектура Actor-Critic поєднує два підходи навчання з підкріпленням – policy-based і value-based – у єдину систему, що забезпечує ефективне оновлення політики агента. Основна ідея полягає в тому, що модель складається з двох взаємопов’язаних частин: actor, який формує політику, тобто визначає, яку дію обрати у даному стані, та critic, який оцінює якість цієї дії шляхом обчислення цінності стану або пари стан-дія. Ця співпраця дозволяє агенту не тільки приймати рішення, але й коригувати їх, враховуючи довгострокові наслідки.

Actor безпосередньо бере участь у виборі дії, яка буде виконана у поточному стані, базуючись на ймовірнісній або детермінованій політиці. Водночас critic оцінює результат цієї дії, визначаючи наскільки вона була вдалою у контексті очікуваної винагороди. Оцінка критика використовується для побудови сигналу навчання, який коригує параметри політики. Таким чином, actor поступово вдосконалює свою стратегію на основі оцінок, що надходять від critic, що значно знижує дисперсію градієнтів порівняно з чисто політико-орієнтованими методами.

Нижче зображено (рисунок 2.2) схему взаємодії між компонентами системи. Agent, представлений actor-critic архітектурою, отримує стан з

середовища (system) і генерує відповідну дію. Після виконання дії агент отримує новий стан і винагороду. Actor приймає рішення, тоді як critic, аналізуючи зміну стану та отриману винагороду, формує зворотний зв'язок для оновлення параметрів обох компонентів. Така структура дозволяє створити цикл навчання, в якому дія, оцінка й оновлення відбуваються у злагодженому режимі.

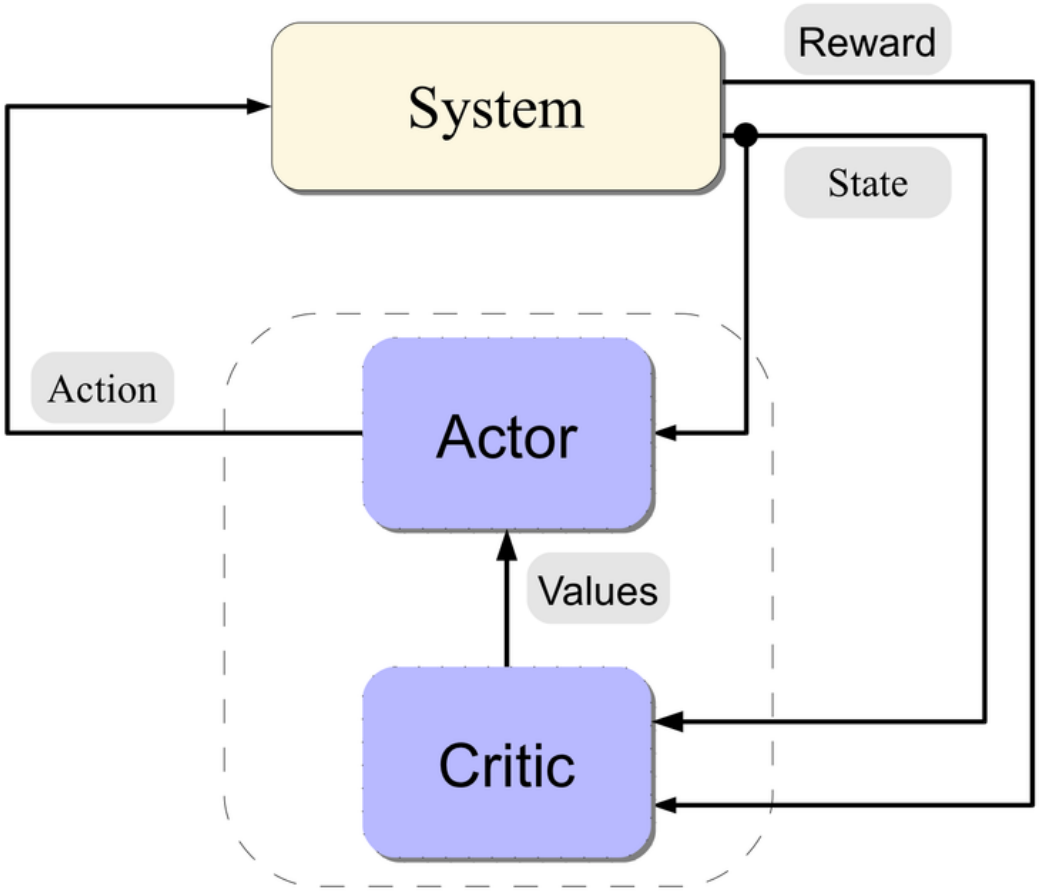

Рисунок 2.2 – Схема архітектури Actor-Critic

Архітектура actor-critic має високу гнучкість і добре масштабується для задач із неперервним простором дій. Вона також дозволяє ефективно працювати у середовищах з частковою спостережуваністю, якщо використовується разом із рекурентними шарами, такими як LSTM. Крім того, можливість оптимізувати політику без необхідності зберігати всі значення вигоди для кожної дії робить actor-critic зручним для складних

середовищ. Завдяки цим перевагам, дана архітектура стала основою для побудови багатьох сучасних алгоритмів, включаючи A2C, A3C та інші.

2.2.3 Модифікації: Double DQN, Dueling DQN

Одним із найбільш відомих удосконалень архітектури DQN є алгоритм Double DQN, який усуває проблему переоцінювання Q-значень. У класичному DQN одна й та сама нейронна мережа використовується як для вибору дії, так і для її оцінки, що призводить до систематичного завищення очікуваних винагород. У Double DQN ці два процеси розділені: основна мережа відповідає за вибір дії, тоді як окрема цільова мережа використовується для оцінювання значення цієї дії. Такий підхід забезпечує більш об’єктивне оновлення Q-функції, що сприяє стабільнішому навчанню.

Іншим важливим покращенням стала поява архітектури Dueling DQN. Її ключова ідея полягає в розділенні оцінювання цінності стану та переваги конкретної дії. Замість того щоб напряму обчислювати Q-функцію для кожної дії, нейронна мережа Dueling DQN має дві окремі гілки: одна обчислює функцію цінності стану, а інша – функцію переваги дії. Потім ці значення комбінуються для побудови фінального Q-значення. Це дозволяє моделі ефективніше навчатися у ситуаціях, де конкретна дія не має суттєвого впливу на результат, але важливо оцінити сам стан.

Нижче показано порівняння (рисунок 2.3) між класичною архітектурою Q-мережі та її модифікацією у вигляді Dueling DQN. У верхній частині зображення мережа напряму формує Q-значення для кожної дії. У нижній частині видно, як потік обробки розділяється на дві гілки – оцінка стану та переваги – які згодом поєднуються для отримання кінцевого результату. Це дає можливість нейронній мережі краще структурувати простір ознак і прискорити навчання.

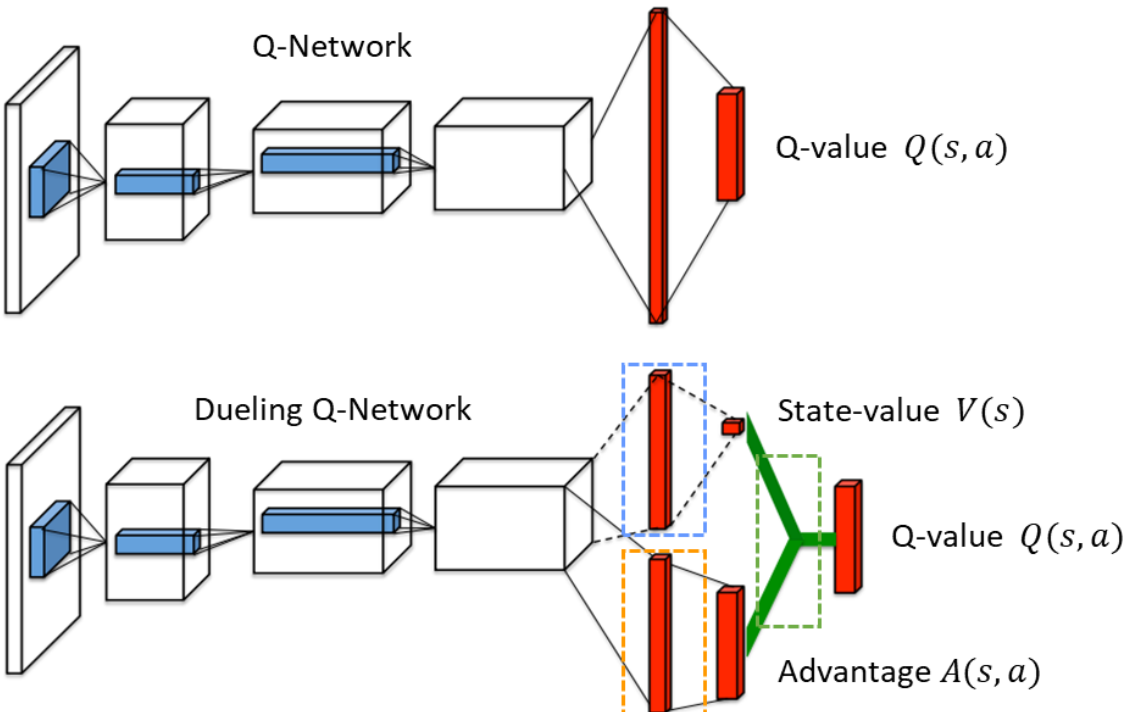

Рисунок 2.3 – Порівняння архітектури DQN та Dueling DQN

Обидві модифікації – Double DQN і Dueling DQN – не змінюють базову логіку навчання з підкріпленням, але істотно впливають на стабільність і якість результатів. Вони дозволяють уникнути помилкових оцінок, швидше досягати збіжності та краще узагальнювати політику агента. Саме тому ці архітектурні рішення активно використовуються в сучасних DRL-системах і можуть бути інтегровані як у класичні, так і в асинхронні або багатокрокові алгоритми. Їх застосування є одним з ключових кроків на шляху до підвищення надійності глибокого навчання з підкріпленням.

2.3 N-крокові алгоритми у навчанні з підкріпленням

У наступному теоретичному блоці розглядаються n-крокові (n-step) алгоритми як узагальнення базових схем оновлення функції цінності. На відміну від однокрокових підходів, які враховують лише негайну винагороду та наступний стан, n-step алгоритми дозволяють

використовувати кумулятивну винагороду за декілька кроків уперед. Це забезпечує кращу передачу сигналу зворотного зв’язку, особливо у випадках, коли винагороди значущі, але відкладені у часі. Такий підхід має потенціал покращення як швидкості збіжності, так і загальної стабільності навчання, особливо у великих або складних середовищах.

Розгляд n-step схем є важливим не лише з математичної точки зору, а й у контексті подальшої реалізації асинхронних алгоритмів, зокрема A3C. Саме багатокрокові оновлення дозволяють кожному потоку агента накопичувати значущу інформацію перед тим, як передати її до глобальної моделі. У межах цього теоретичного аналізу буде пояснено, чому n-step підходи мають перевагу над one-step варіантами в задачах з довготривалим впливом дій, а також буде порівняно вплив глибини оновлення на точність і швидкість навчання. Це дозволить чітко побачити, як зростає ефективність системи за рахунок врахування більшої кількості інформації при кожному оновленні.

2.3.1 Мотивація використання n-step backup

Однокрокові оновлення, які використовуються в базових алгоритмах Q-learning і Sarsa, мають суттєве обмеження – вони враховують лише негайну винагороду та короткострокову інформацію про стан середовища. Це ускладнює навчання у випадках, коли агенту необхідно приймати рішення з урахуванням наслідків, що проявляються лише через кілька кроків. У таких умовах однокроковий backup недостатньо ефективний для передачі сигналу зворотного зв’язку назад до початкових дій. Саме тому виникла потреба у використанні n-крокових оновлень, які дозволяють враховувати послідовність переходів і кумулятивну винагороду за кілька дій уперед.

Застосування n-step backup забезпечує гнучкий баланс між точністю прогнозу та швидкістю поширення навчального сигналу. З одного боку, це

дає змогу моделі краще оцінювати довгострокові наслідки дій, з іншого – зменшує затримку зворотного зв’язку, яка виникає у повністю епізодичних оновленнях. У результаті агент отримує більше контексту про свою поведінку в середовищі, що покращує як стабільність навчання, так і збіжність до ефективної політики. Саме тому n-step backup став важливою складовою сучасних алгоритмів, зокрема у поєднанні з асинхронною архітектурою.

2.3.2 Порівняння one-step і n-step схем

Однокрокові (one-step) та багатокрокові (n-step) схеми оновлення в навчанні з підкріпленням відрізняються підходом до обчислення цільового значення для оновлення функції вигоди. У one-step схемі оновлення базується лише на одній негайній винагороді та прогнозі з наступного стану. Це забезпечує швидке оновлення, але має обмеження у здатності передавати інформацію про довготривалі наслідки дій, особливо у складних середовищах з відкладеною винагородою. Такі оновлення підходять для стабільного, але короткозорого навчання.

На відміну від цього, n-step схеми враховують кумулятивну винагороду за кілька кроків уперед, що дозволяє краще передавати сигнал зворотного зв’язку до попередніх дій. Замість фокусування лише на наступному стані, n-step backup дозволяє враховувати послідовність переходів, накопичену винагороду та глибшу залежність між діями та їхніми наслідками. Це покращує здатність агента навчатись у складних середовищах і швидше формувати узгоджену політику.

На рисунку зображено (рисунок 2.4) порівняння між однокроковою та n-кроковою схемами. У верхній частині видно, як one-step оновлення використовує інформацію лише з одного переходу між станами, тобто лише безпосередню винагороду за дію у конкретному стані. Це робить його більш швидким з точки зору обчислень, однак воно має суттєве обмеження –

обмежений контекст. У таких умовах агент оновлює свою політику на основі лише негайного зворотного зв’язку, що може призвести до недооцінки або переоцінки дій, які мають затриманий ефект.

У нижній частині видно, що n-step backup охоплює більшу ділянку епізоду та агрегує інформацію з кількох послідовних кроків, враховуючи сукупність проміжних винагород. Це дозволяє більш точно оцінити довгострокову вигоду, особливо у випадках, коли винагорода розподілена у часі, наприклад, якщо позитивний зворотний зв’язок настає лише після низки правильних дій. Таке агрегування сприяє пришвидшеному навчанню агента та дозволяє краще коригувати оцінки цінності.

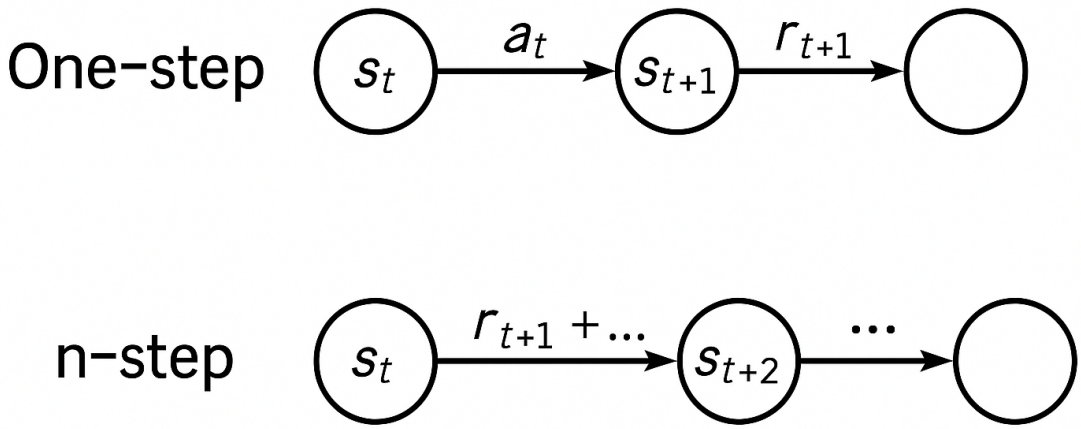

Рисунок 2.4 – Різниця між one-step та n-step

Попри очевидні переваги, використання n-step схем вимагає більшої обчислювальної потужності та кращої організації буфера досвіду або локального зберігання переходів. Крім того, занадто велике значення n може призвести до переобтяження моделі шумом, якщо дані з глибини епізоду є менш релевантними. Тому у практичних реалізаціях часто застосовуються компромісні значення n, які забезпечують баланс між глибиною навчального сигналу та стабільністю оновлень.

2.3.3 Роль n-step у довготривалому навчанні

Застосування n-step підходу суттєво впливає на збіжність алгоритму навчання з підкріпленням, особливо в середовищах із довготривалими залежностями між діями та винагородами. Оскільки агент при n-крововому оновленні враховує не лише негайну винагороду, а й накопичену за кілька кроків вперед, це дозволяє швидше поширювати інформацію про доцільність дій у попередні стани. Таким чином, сигнали зворотного зв'язку мають ширший радіус впливу, що прискорює процес навчання, особливо у складних або розріджених середовищах, де винагорода надається епізодично або з затримкою.

Ще одним важливим аспектом є покращення довготривалого планування. Завдяки тому, що агент аналізує послідовність кроків, він навчається обирати дії, які не приносять миттєвої вигоди, але ведуть до вигідних ситуацій у майбутньому. Це створює умови для формування більш стратегічної політики, яка орієнтована не лише на короткочасний прибуток, а й на досягнення глобальних цілей. У поєднанні з асинхронною багатопотоковою архітектурою, яка постачає різноманітні траєкторії, n-step backup сприяє стабільній та швидкій збіжності до оптимальної поведінки.

2.4 Архітектура actor-critic та функція переваги

У цьому теоретичному сегменті розглядається архітектура actor-critic, яка є фундаментом для багатьох сучасних алгоритмів глибокого навчання з підкріпленням. Вона поєднує два підходи: оцінювання політики (actor) та оцінювання станів або дій (critic). На відміну від чисто policy-based або value-based алгоритмів, actor-critic дозволяє ефективно оновлювати політику агента на основі поточних оцінок якості дій, водночас коригуючи ці оцінки у реальному часі. Така гібридна структура забезпечує баланс між стабільністю і швидкістю навчання, а також дозволяє агенту працювати у

складних середовищах, де інші архітектури виявляються менш ефективними.

Особливу роль у цій архітектурі відіграє функція переваги (advantage function), яка вимірює, наскільки конкретна дія у даному стані є кращою за середню дію в тому ж стані. Використання переваги замість повної функції вигоди дозволяє зменшити дисперсію градієнтів при оновленні політики, що сприяє більш стабільному процесу навчання. У цьому блоці буде розглянуто математичні основи actor-critic, інтерпретацію переваги, а також ситуації, в яких ця архітектура є найбільш ефективною. Це створює підґрунтя для подальшого аналізу алгоритму A3C, що базується саме на actor-critic підході з використанням асинхронного навчання.

2.4.1 Основи actor-critic: розділення політики і цінності

Actor-critic підхід є гібридною формою навчання з підкріпленням, що об’єднує два основні напрями: policy-based і value-based алгоритми. Основна ідея полягає у розділенні функціоналу агента на дві частини – actor і critic. Actor відповідає за побудову політики, яка визначає ймовірність вибору певної дії в конкретному стані, тоді як critic оцінює якість дій за допомогою функції цінності. Це дозволяє моделі не лише здійснювати вибір, а й аналізувати його ефективність, формуючи сигнал навчання для оновлення політики.

Розділення політики та цінності дає змогу обійти деякі обмеження чисто політико-орієнтованих підходів, таких як високий рівень дисперсії градієнтів. Використання оцінки стану або дії дозволяє побудувати більш точний і спрямований сигнал для оновлення параметрів політики. З іншого боку, actor-critic усуває необхідність перебирати всі можливі дії в кожному кроці, як це властиво деяким value-based методам. Таким чином, ця архітектура забезпечує як ефективність навчання, так і зниження обчислювальних витрат.

Ще однією перевагою actor-critic підходу є можливість застосування в умовах неперервного простору дій. Оскільки actor формує безпосередньо політику, яка може бути параметризована неперервними функціями, це відкриває шлях до ефективного використання у складних динамічних системах. Critic у такому випадку виступає як джерело інформації про якість вибору дій, що дозволяє адаптувати політику до нових умов і підтримувати стабільність у процесі навчання. Саме ця гнучкість і адаптивність зробили actor-critic фундаментом для багатьох сучасних алгоритмів.

2.4.2 Визначення і роль функції переваги A(s, a)

Функція переваги є важливим елементом у сучасних actor-critic алгоритмах, оскільки дозволяє точніше оцінювати внесок конкретної дії у загальну вигоду агента. Замість того щоб покладатися лише на абсолютне значення функції вигоди, агент може орієнтуватися на те, наскільки кращою або гіршою є обрана дія порівняно з очікуваною середньою дією в даному стані. Це допомагає зосередити процес навчання на тих діях, які справді впливають на результат, і зменшує вплив флуктуацій у значеннях винагород, характерних для багатьох середовищ.

Математично функція переваги визначається як різниця між функцією вигоди і функцією цінності стану. Формула має наступний вигляд:

$$A(s,a) = Q(s,a) - V(s), \tag{2.2}$$

де $A(s,a)$ – функція переваги для дії a в стані s;

$Q(s,a)$ – функція вигоди при виконанні дії a в стані s;

$V(s)$ – очікувана загальна вигода від стану s, незалежно від дії.

Тут $Q(s,a)$ визначає очікувану сумарну винагороду при виконанні дії в стані, а $V(s)$ – середню вигоду від усіх можливих дій у цьому стані згідно з поточною політикою. Такий підхід дозволяє фокусуватися саме на

корисності конкретних дій, а не на загальній ситуації, що є ключовим для побудови ефективного градієнта політики.

Нижче зображено (рисунок 2.5), як функція переваги інтегрується в архітектуру actor-critic. Компонент actor приймає стан із середовища та вибирає дію згідно з поточною політикою. Паралельно critic оцінює, наскільки вдалою була ця дія в контексті ситуації. Результат цієї оцінки використовується для формування сигналу, який повідомляє actor, наскільки обрана дія була кращою або гіршою за середню. Такий сигнал допомагає коригувати політику таким чином, щоб посилювати ті дії, що давали вищу результативність у конкретних станах.

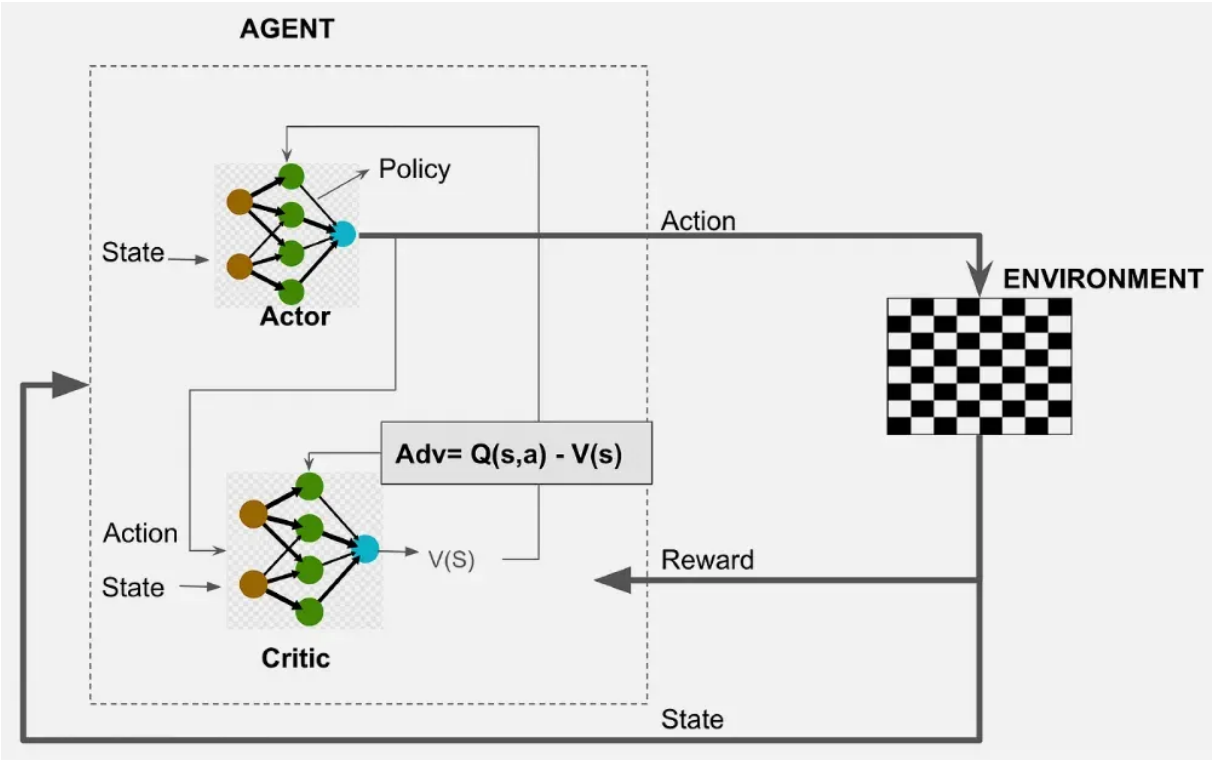

Рисунок 2.5 – Схема архітектури Actor-Critic з функцією переваги

Функція переваги виступає інструментом фокусування навчання на справді релевантних діях. Вона дозволяє уникнути ситуацій, коли навіть неефективні дії здаються вигідними лише тому, що трапилися у сприятливому стані. Завдяки цьому навчання стає стабільнішим, а сам агент

поступово формує стратегію, орієнтовану на довгостроковий успіх. Саме тому перевага активно використовується в більшості сучасних алгоритмів навчання з підкріпленням, які поєднують стратегічне прийняття рішень із гнучкою оцінкою досвіду.

2.4.3 Переваги actor-critic архітектури у складних задачах

Actor-critic архітектура демонструє високу ефективність у задачах зі складною структурою середовища, великим простором станів або неперервними діями. Її ключова перевага полягає в тому, що модель не потребує перебору всіх можливих дій для оцінки політики, як це відбувається у value-based підходах. Actor безпосередньо генерує дію, а critic дає швидкий сигнал оцінки, що значно знижує обчислювальні витрати та дозволяє працювати в реальному часі. Це робить actor-critic підходящим вибором для робототехніки, ігор з неперервним контролем або систем прийняття рішень зі складною динамікою.

Ще однією перевагою є здатність actor-critic адаптуватися до частково спостережуваних середовищ, особливо при використанні рекурентних шарів. Навіть якщо агент не має повної інформації про поточний стан, він може на основі послідовності спостережень формувати ефективну політику. Крім того, гнучкість у побудові actor і critic окремо дозволяє легко адаптувати архітектуру до конкретної задачі, оптимізуючи її під конкретні характеристики середовища. У підсумку actor-critic забезпечує як високу якість рішень, так і стабільне навчання навіть у складних умовах.

2.5 Асинхронна багатопотокова архітектура

Окрема частина теоретичного аналізу присвячена асинхронній багатопотоковій архітектурі, яка стала ключовим проривом у подоланні обмежень класичних алгоритмів глибокого навчання з підкріпленням. На

відміну від традиційних підходів, де досвід збирається послідовно одним агентом і використовується через буфер пам’яті, асинхронна архітектура дозволяє запускати кілька агентів одночасно. Кожен потік незалежно взаємодіє з власним середовищем, накопичує локальний досвід і оновлює глобальну модель на основі локально обчислених градієнтів. Це не лише знижує кореляцію між зразками, а й дозволяє уникнути потреби в досвід-буфері, що є суттєвою перевагою для on-policy навчання.

Розгляд цієї архітектури охоплює ключові компоненти: паралельне збирання досвіду, використання локальних копій моделі, асинхронне оновлення параметрів та їх синхронізацію із глобальною мережею. Такий підхід демонструє високу стабільність, навіть без ускладнених механізмів стабілізації, характерних для DQN. Теоретичний аналіз буде зосереджено на тому, як така архітектура забезпечує масштабованість, дозволяє ефективне використання CPU-ресурсів та сприяє пришвидшенню збіжності. Це створює логічне підґрунтя для розгляду конкретного алгоритму A3C, який реалізує всі ці принципи на практиці.

2.5.1 Принципи паралельного навчання

Паралельне навчання є ключовим принципом, що лежить в основі асинхронних алгоритмів навчання з підкріпленням, таких як A3C. Його суть полягає у запуску декількох агентів або потоків, які одночасно взаємодіють із окремими копіями середовища. Кожен з цих агентів досліджує середовище незалежно, збирає досвід та локально обчислює оновлення для моделі. Такий підхід дозволяє суттєво підвищити різноманітність зібраних траєкторій, що позитивно впливає на узагальнювальну здатність глобальної моделі.

Один з головних плюсів паралельного навчання – зменшення кореляції між послідовними станами, що є типовою проблемою для однопотокових алгоритмів. Оскільки кожен агент діє автономно, він

відвідує різні частини простору станів і генерує унікальні послідовності дій. Це створює більш якісний навчальний сигнал, уникаючи надмірного повторення одних і тих самих ситуацій. Крім того, такий підхід дозволяє краще охопити можливі варіанти поведінки в складному або стохастичному середовищі.

Ще однією важливою перевагою є можливість використання багатоядерних систем для розподілу обчислювального навантаження. Паралельне навчання дозволяє виконувати обчислення в потоках незалежно, що робить систему масштабованою і зручною для реалізації на сучасному обладнанні. Завдяки цьому досягається не лише збільшення швидкості навчання, а й підвищення стабільності – адже глобальна модель оновлюється з урахуванням середнього досвіду з багатьох різних джерел. Це робить паралельне навчання особливо актуальним для задач із високими обчислювальними вимогами та великою кількістю станів.

2.5.2 Глобальна модель і локальні копії

У парадигмі асинхронного навчання ключовим механізмом є наявність глобальної моделі, яка виступає центральним сховищем знань, накопичених усіма агентами. Ця модель не взаємодіє безпосередньо з середовищем, а лише отримує оновлення у вигляді градієнтів від локальних копій, що працюють у паралельних потоках. Глобальна модель синхронізує свої параметри з локальними копіями, надаючи їм оновлені значення перед початком нового циклу навчання. Завдяки цьому вся система зберігає узгодженість політики, навіть при незалежному дослідженні середовищ кожним потоком.

Локальні копії є повноцінними агентами, які отримують стан середовища, генерують дії та обчислюють втрати на основі власного досвіду. Вони періодично надсилають обчислені градієнти до глобальної моделі, яка оновлює свої параметри на основі цих внесків. Такий підхід

дозволяє уникнути централізованого зберігання буфера досвіду, оскільки кожен агент опрацьовує власну послідовність переходів у реальному часі. Це особливо важливо для алгоритмів on-policy типу, які повинні оновлюватися лише за поточним розподілом політики.

Використання глобальної моделі та локальних копій забезпечує високу ефективність, гнучкість та масштабованість системи. Усі потоки працюють незалежно, тому немає потреби в синхронному доступі до загальних ресурсів, що зменшує затримки й ризик блокування. Навіть якщо окремий агент працює повільніше або виявляється менш ефективним, він не впливає критично на загальний хід навчання. Така архітектура дозволяє забезпечити баланс між індивідуальним дослідженням і централізованим оновленням, сприяючи стабільній і швидкій збіжності глобальної політики.

2.5.3 Особливості оновлення параметрів у потоках

Процес оновлення параметрів у потоках асинхронної архітектури відрізняється від традиційних синхронних методів тим, що кожен агент виконує локальне навчання незалежно від інших. Кожен потік взаємодіє зі своєю копією середовища, накопичує досвід і після фіксованої кількості кроків обчислює градієнти втрати на основі локальних даних. Ці градієнти не використовуються безпосередньо для оновлення локальної моделі, а надсилаються до глобальної мережі, яка оновлюється централізовано. Це дозволяє уникнути конфліктів між потоками та сприяє ефективному агрегуванню знань із різних джерел.

Оновлення глобальної моделі виконується за допомогою стохастичного градієнтного спуску або його модифікацій, де параметри змінюються на основі накопичених змін із локальних потоків. Оскільки ці оновлення відбуваються асинхронно і незалежно, у системі немає жорсткої потреби у блокуванні ресурсів або черговості оновлень. Це дозволяє масштабувати систему на десятки потоків без втрати продуктивності. При

цьому кожен потік оновлюється за принципом policy gradient, n-step backup або actor-critic, залежно від обраного алгоритму.

Важливою особливістю є також частота оновлення. Занадто рідке оновлення може призвести до застарівання політики, що погіршує якість взаємодії агента з середовищем, тоді як надто часте оновлення може зменшити стабільність через надмірну чутливість до локальних флуктуацій. Тому в практичних реалізаціях використовуються механізми контролю, такі як фіксована кількість кроків між оновленнями або динамічне регулювання частоти синхронізації. Це дозволяє досягти балансу між адаптивністю та стабільністю глобальної моделі.

Ще однією важливою деталлю є те, що параметри глобальної моделі не оновлюються миттєво після кожного кроку агента, а лише після завершення певного локального етапу навчання. Така буферизація градієнтів дозволяє зменшити коливання параметрів та зберегти сталість у процесі оптимізації. Водночас, це дозволяє забезпечити незалежність потоків, зменшити взаємне блокування й забезпечити високу продуктивність системи на багатоядерних процесорах. Саме завдяки цим особливостям асинхронна модель оновлення є не лише ефективною, а й стійкою до локальних нестабільностей у поведінці окремих агентів.

2.6 Алгоритм Asynchronous Advantage Actor-Critic

Завершальний теоретичний розділ присвячений детальному розгляду алгоритму Asynchronous Advantage Actor-Critic (A3C), який став однією з найефективніших реалізацій асинхронного навчання з підкріпленням. Цей алгоритм поєднує ключові принципи actor-critic підходу, використання функції переваги та багатопотокової архітектури, що дозволяє досягати високої продуктивності без потреби в буфері досвіду або складній синхронізації. У A3C декілька агентів паралельно навчають одну глобальну

модель, використовуючи n-крокові траєкторії, перевагу як сигнал навчання та ентропійне регуляризування для збереження різноманітності у політиці.

У межах цього блоку буде розглянуто архітектуру A3C на рівні окремих компонентів: обчислення функції втрат для політики та цінності, інтеграція переваги, механізм ентропійної регуляризації та принцип асинхронного оновлення ваг. Також буде проаналізовано можливість використання рекурентних шарів, зокрема LSTM, для обробки частково спостережуваних станів. Така деталізація дозволить чітко зрозуміти, як A3C об’єднує у собі теоретичні ідеї і чому саме цей алгоритм було обрано для подальших експериментальних досліджень.

2.6.1 Обчислення функції втрат: політика, цінність, ентропія

У алгоритмі Asynchronous Advantage Actor-Critic (A3C) обчислення функції втрат є центральним компонентом, що об’єднує кілька напрямів оптимізації – оновлення політики агента, уточнення оцінки цінності стану та регуляризацію політики для збереження її дослідницьких властивостей. Кожен з цих елементів відіграє окрему роль у навчанні та разом формує загальну функцію втрат, яка використовується для оновлення параметрів як actor, так і critic частини моделі. Формула має наступний вигляд:

$$L = L_{policy} + c_v \times L_{value} - c_e \times H(\pi), \qquad (2.3)$$

де L_{policy} – втрата для політики, що ґрунтується на градієнті політики і функції переваги;

c_v – коефіцієнт масштабування;

L_{value} – похибка у передбаченні цінності;

c_e – коефіцієнт масштабування;

$H(\pi)$ – ентропія політики.

Компонент втрат для політики заснований на градієнті виграшу та функції переваги. Він коригує параметри actor у напрямі, який збільшує ймовірність вибору дій з високим значенням переваги. Іншими словами, чим кориснішою була дія у певному стані, тим більше підсилюється ймовірність її повторного вибору в майбутньому. Це дозволяє моделі швидко фокусуватись на діях, що дають високі результати, залишаючи при цьому простір для адаптації.

Другий компонент – це втрата для ціннісної функції. Вона обчислюється як різниця між прогнозованою оцінкою стану та фактично отриманою сумарною винагородою. Такий підхід дозволяє critic зменшувати похибку у прогнозах і покращувати якість сигналу навчання для actor. Третій елемент – ентропійна регуляризація – додається для того, щоб уникнути передчасної детермінізації політики. Вона збільшує ентропію розподілу дій, тобто стимулює агента до дослідження нових стратегій. Це особливо важливо на ранніх етапах навчання, коли середовище ще недостатньо вивчене.

У підсумку функція втрат в A3C є сумою трьох складових – втрати політики, втрати цінності та ентропійного штрафу. Кожен із цих компонентів масштабовано з власним коефіцієнтом, що дозволяє точно налаштувати вплив кожного елементу на загальний процес навчання. Такий підхід забезпечує стабільність, адаптивність і ефективність, що й зробило A3C одним із найуспішніших асинхронних алгоритмів у сфері навчання з підкріпленням.

2.6.2 Аспекти реалізації A3C: використання LSTM, ентропійна регуляризація

У сучасних реалізаціях алгоритму A3C важливою особливістю є розширення архітектури агентів за допомогою рекурентних нейронних мереж, зокрема блоків типу LSTM. Їх використання дозволяє моделі

враховувати послідовність попередніх станів, що є критично важливим у задачах з частково спостережуваним середовищем. На відміну від класичних підходів, які покладаються лише на поточне спостереження, агент із LSTM має можливість формувати внутрішнє уявлення про історію взаємодії, зберігаючи та обробляючи контекст попередніх дій. Це підвищує здатність моделі до стратегічного планування в умовах неповної інформації.

Нижче зображено детальну структуру (рисунок 2.6) одного кроку LSTM-блоку, що використовується в агенті A3C. Механізм LSTM складається з комірки пам'яті, вхідного, вихідного та забуваючого елементів керування, кожен з яких виконує специфічну роль. Наприклад, забуваючий блок визначає, яка частина попередньої інформації повинна бути збережена, а яка – відкинута. Вхідний блок додає нову інформацію, а вихідний – формує нове приховане представлення стану. Цей механізм дозволяє ефективно управляти як короткостроковою, так і довгостроковою пам'яттю.

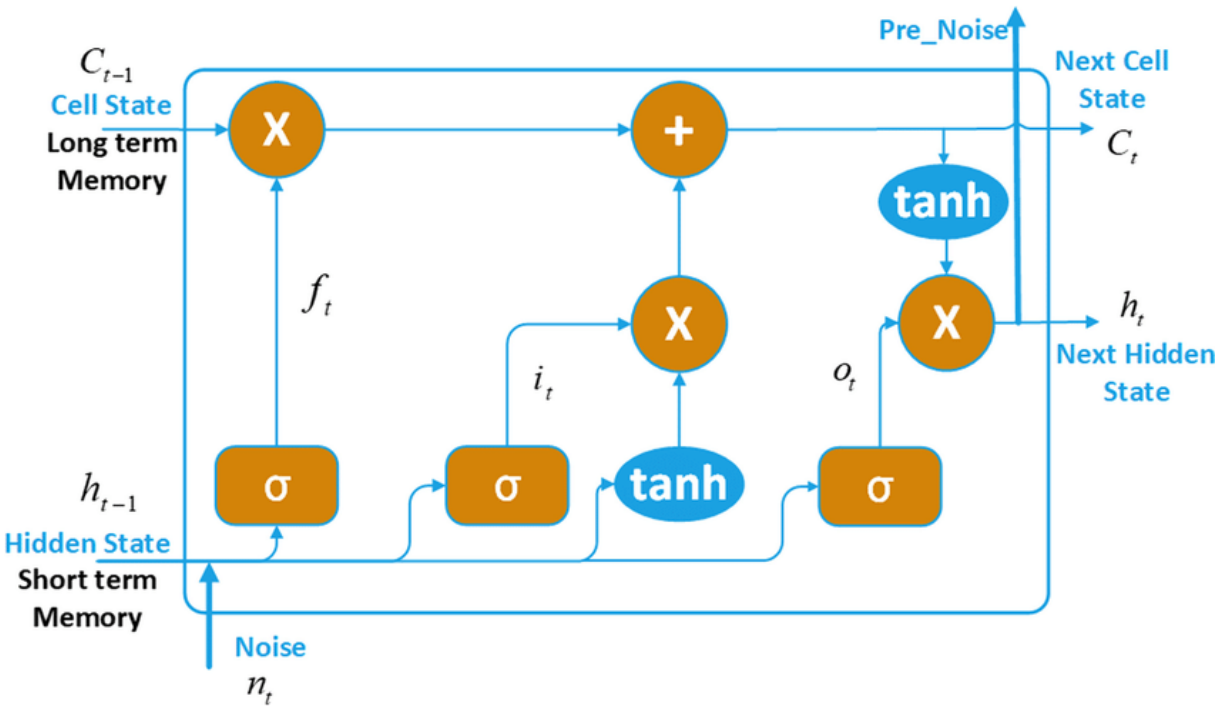

Рисунок 2.6 – Схема LSTM

Окрім використання LSTM, алгоритм A3C включає важливий елемент під назвою ентропійна регуляризація. Цей механізм відіграє роль у збереженні балансу між експлуатацією відомих стратегій та дослідженням нових. У процесі навчання агент часто може переходити до жорстко детермінованої політики, що знижує здатність досліджувати нові дії. Для запобігання цьому до функції втрат додається спеціальний член, який стимулює зростання ентропії політики, тобто її невизначеність.

Ентропія політики є мірою того, наскільки рівномірно агент розподіляє ймовірність між доступними діями. Висока ентропія свідчить про те, що всі дії мають приблизно однакову ймовірність, що дозволяє агенту досліджувати нові стратегії. Натомість низька ентропія сигналізує про фіксацію агента на кількох обраних діях, навіть якщо вони не є оптимальними. Регуляризаційний ентропійний термін у функції втрат дозволяє штучно підтримувати високу ентропію на ранніх етапах навчання, зменшуючи її поступово в міру стабілізації політики.

Математично ентропія політики обчислюється як сума добутків ймовірностей дій на логарифми від цих ймовірностей зі знаком мінус. Цей термін масштабується на певний коефіцієнт і додається до основної функції втрат із протилежним знаком, щоб максимізувати ентропію. Таким чином, агент заохочується до того, щоб розподіляти свою політику не надто впевнено на початку навчання. Такий підхід сприяє кращому покриттю простору станів і зменшує ризик потрапляння в локальні мінімуми політики.

У поєднанні використання LSTM для запам’ятовування послідовностей і ентропійної регуляризації для підтримки дослідження, реалізація A3C досягає високої гнучкості та адаптивності. Агенти стають здатними не лише оптимізувати поведінку в конкретних ситуаціях, але й шукати нові шляхи розв’язання задач, що змінюються з часом. Це особливо важливо для складних, стохастичних середовищ, де довгострокове планування, пам’ять і стратегічне дослідження є необхідними для досягнення високих результатів.

3 ЕКСПЕРИМЕНТАЛЬНІ ДОСЛІДЖЕННЯ

3.1 План експериментів

У межах даного дослідження планується реалізувати серію експериментів для перевірки ефективності асинхронних багатопотокових алгоритмів навчання з підкріпленням. Передбачається протестувати чотири модифікації: одно-кроковий Sarsa, одно-кроковий Q-learning, n-кроковий Q-learning та actor-critic із функцією переваги. Вибір цих алгоритмів зумовлений їхньою популярністю у класичному навчанні з підкріпленням, а також потенціалом для асинхронного паралельного виконання без потреби у складних ресурсозатратних механізмах стабілізації, таких як буфер досвіду. Головна мета експериментів полягає в оцінці здатності цих методів стабільно тренувати політики, представлені глибокими нейронними мережами, із мінімальними вимогами до апаратного забезпечення.

Планується протестувати алгоритми на трьох різних типах середовищ. Основне місце в експериментах відводиться середовищу Arcade Learning Environment, яке надає доступ до широкого набору ігор Atari 2600. Це середовище є загальновизнаним стандартом для тестування алгоритмів підкріплення, оскільки дозволяє оцінити здатність агентів справлятись із широким спектром завдань. Результати, отримані в Atari, дадуть змогу не лише проаналізувати швидкість навчання і збіжність алгоритмів, а й провести пряме порівняння з відомими підходами, такими як DQN, Gorila чи A2C.

Крім Atari, планується проведення експериментів у 3D симуляторі TORCS – середовищі, яке суттєво відрізняється за динамікою, типом керування та вимогами до агента. На відміну від дискретного управління в Atari, у TORCS агенту доведеться навчатися в умовах неперервних дій і працювати з симульованою фізикою руху автомобіля. Це середовище

дозволить перевірити масштабованість і здатність алгоритмів адаптуватися до більш складних сценаріїв управління.

Останнє середовище, заплановане до використання, – це Labyrinth. Воно представляє складну навігаційну задачу у частково спостережуваному просторі, де агенту необхідно досліджувати невідомі простори, приймати рішення в умовах невизначеності та узагальнювати досвід. Labyrinth буде використано переважно для тестування алгоритму A3C, оскільки він об’єднує властивості актор-критик моделі, багатокрокового оновлення та асинхронного паралелізму, що робить його придатним для таких динамічних задач.

Таким чином, заплановані експерименти охоплюють широкий діапазон середовищ, що відрізняються за природою даних, складністю динаміки та типом завдань. Це дозволить провести комплексну оцінку запропонованих асинхронних методів, визначити їх переваги й недоліки в різних контекстах, а також встановити області найбільш ефективного застосування.

3.2 Обрані середовища для проведення експериментів

У цьому підрозділі розглядаються середовища, обрані для проведення експериментального дослідження ефективності асинхронних алгоритмів навчання з підкріпленням. Основна увага приділяється трьом типам середовищ, які суттєво відрізняються за структурою простору станів, динамікою середовища та характером завдань. Таке різноманіття дає змогу всебічно оцінити здатність агентів адаптуватися до задач різної складності. Кожне з середовищ виконує свою функцію в дослідженні: перевірка узагальнювальної здатності, здатності до довготривалого планування та стійкості до непередбачуваних змін.

3.2.1 Arcade Learning Environment

Arcade Learning Environment є класичним тестовим середовищем для оцінювання алгоритмів навчання з підкріпленням у задачах із візуальним входом та дискретними діями. Воно ґрунтується на емуляції ігор Atari 2600, які пропонують широкий спектр динамічних завдань, що вимагають як швидкої реакції, так і довготривалого планування. Усі ігри в цьому середовищі мають однаковий формат вхідних даних – піксельне зображення з екрану гри – що забезпечує стандартизовану оцінку здатності агентів до узагальнення політики візуального аналізу. Саме завдяки цьому є загальновизнаним еталоном для порівняння нових алгоритмів з наявними state-of-the-art підходами.

У планованому дослідженні середовище Atari використовується як основна експериментальна платформа. Завдання, які будуть протестовані, охоплюють ігри з різним ступенем складності: від простих реактивних сценаріїв до складних стратегічних ігор з відкладеною винагородою. Це дозволить дослідити, наскільки добре асинхронні алгоритми здатні масштабуватися до задач із різною глибиною залежностей та наскільки стійкими вони залишаються при тривалому навчанні. Додатково, використання дозволяє повторно перевірити результати попередніх досліджень, таких як DQN і Gorila, на однакових умовах.

3.2.2 TORCS

TORCS або The Open Racing Car Simulator є реалістичним тривимірним симулятором автомобільних перегонів, що широко використовується у дослідженнях навчання з підкріпленням для задач із неперервним простором дій. У порівнянні з іграми Atari, це середовище має суттєво складнішу динаміку, зокрема моделювання фізики руху, поворотів, зіткнень та інерції. Агенти в TORCS повинні приймати рішення не на основі

дискретних дій, а шляхом безперервного регулювання параметрів, таких як прискорення, гальмування та кут повороту керма. Це робить його корисним для перевірки алгоритмів, здатних ефективно функціонувати в умовах високої розмірності та плавної динаміки.

Планується використовувати TORCS для порівняльного тестування алгоритмів, зокрема actor-critic та A3C, у сценаріях з різним ступенем складності – як без суперників, так і з іншими автомобілями на трасі. Середовище дозволить оцінити, наскільки агенти, навчені в асинхронному багатопотоковому режимі, здатні виробити стратегії для ефективного управління транспортом у змінних умовах. Зокрема, буде досліджено здатність агентів до довгострокового планування, узгодження швидкості та керування при уникненні зіткнень. Завдяки високій варіативності поведінки в TORCS, це середовище слугує важливим етапом для перевірки загальної придатності алгоритмів до задач реального світу.

3.2.3 Labyrinth

Labyrinth є візуально насиченим середовищем із тривимірною перспективою, створеним спеціально для тестування агентів у завданнях просторової навігації, дослідження та навчання в умовах часткової спостережуваності. Це середовище поєднує у собі риси класичних задач навчання з підкріпленням та особливості, характерні для когнітивних сценаріїв – зокрема, необхідність побудови внутрішнього уявлення про простір, запам’ятовування раніше відвіданих ділянок та адаптації до динамічно змінюваної структури лабіринту. Агенти отримують на вхід лише зображення з першої особи, що імітує умови роботи автономних систем у невідомих середовищах.

У рамках експериментального дослідження планується використати Labyrinth переважно для оцінки алгоритму A3C, оскільки саме ця архітектура поєднує рекурентні механізми пам’яті з паралельним

асинхронним навчанням, що є критично важливим для задач навігації. Labyrinth дозволяє перевірити здатність агента вчитись на основі неповної інформації, приймати обґрунтовані рішення у складному середовищі та адаптувати свою політику в умовах високої невизначеності. Вивчення поведінки агентів у цьому середовищі дозволить отримати глибше розуміння здатності асинхронних методів до генералізації та стратегічного планування у просторах, що змінюються.

3.3 Вибір алгоритму оптимізації

У рамках дослідження ефективності асинхронних алгоритмів навчання було заплановано порівняння різних варіантів оптимізації параметрів моделей. Для цього розглядалися два основні алгоритми – стохастичний градієнтний спуск (SGD) та RMSProp. Основною вимогою до оптимізаторів було забезпечення високої пропускної здатності при багатопотоковій обробці без використання блокувань. Це дозволяє максимізувати швидкість оновлень глобальної моделі при великій кількості паралельних агентів.

Реалізація SGD у асинхронному контексті є відносно прямолінійною. Параметри мережі спільно використовуються всіма потоками, де кожен потік обчислює власні градієнти та окремо застосовує оновлення зі своїм власним вектором моментуму. Оновлення відбуваються незалежно для кожного потоку, що забезпечує мінімальні витрати на синхронізацію, але водночас може призводити до нестабільності при великій кількості потоків через неконтрольовані коливання градієнтів.

Для алгоритму RMSProp було досліджено два варіанти реалізації в асинхронному режимі. Перший варіант, окремий RMSProp, передбачає, що кожен потік підтримує власну статистику середнього квадрата градієнтів. Це дозволяє кожному агенту адаптувати свої кроки навчання індивідуально, але збільшує вимоги до пам’яті, оскільки потрібно зберігати окремі копії

параметрів для кожного потоку. Другий варіант, спільний RMSProp, передбачає, що всі потоки використовують спільну статистику, яка оновлюється асинхронно без блокувань. Такий підхід значно економить пам’ять і знижує ризик розходження статистик у різних потоках.

Щоб оцінити ефективність і стійкість кожного з варіантів оптимізації, було проведено серію експериментів (рисунок 3.1) із двома різними типами алгоритмів навчання – асинхронним n-кроковим Q-learning і асинхронним actor-critic з функцією переваги. Тестування проводилось на чотирьох різних іграх середовища Atari: Breakout, Beamrider, Seaquest та Space Invaders. Для кожного методу було проведено 50 незалежних експериментів із різними випадковими початковими умовами та швидкостями навчання, що дозволяє оцінити чутливість алгоритмів до ініціалізації і налаштувань.

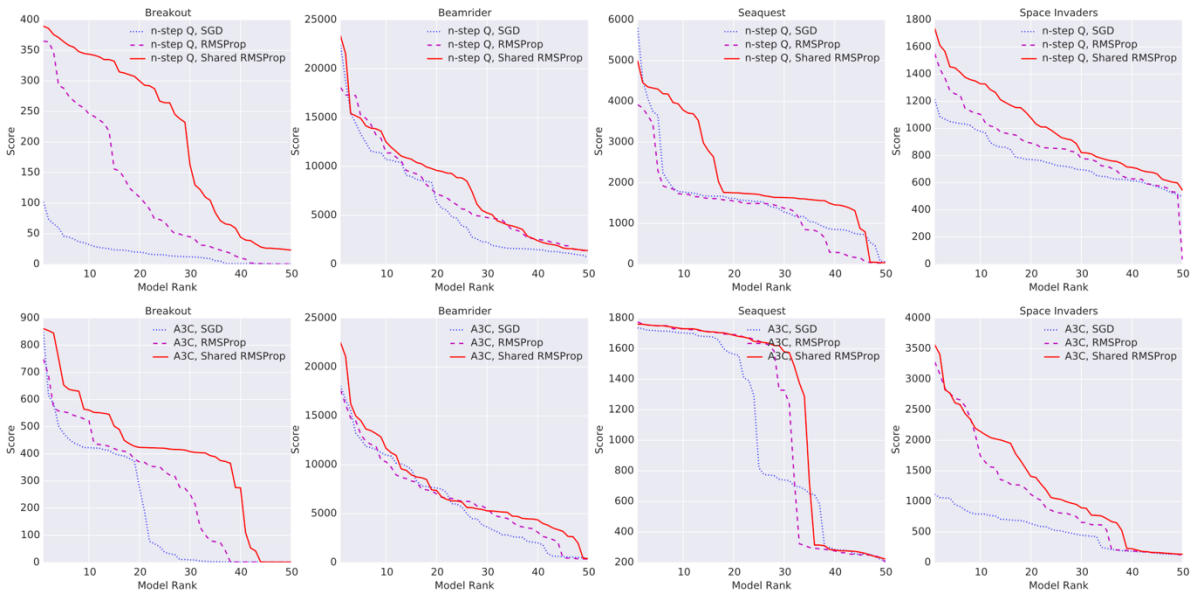

Рисунок 3.1 – Порівняння трьох алгоритмів оптимізації

На рисунку наведено графіки ранжування результатів моделей за фінальним середнім балом. Вісь x відображає ранг моделі після сортування, а вісь y – фінальний середній результат для кожної моделі. Спостерігається чітка перевага варіанту RMSProp зі спільною статистикою над іншими

підходами як для n-крокового Q-learning, так і для actor-critic. Криві для цього варіанту демонструють меншу розбіжність між моделями й вищі середні показники у всіх експериментальних умовах.

Результати показали, що використання RMSProp зі спільною статистикою дозволяє значно підвищити стійкість навчання, зменшуючи залежність від вибору початкової швидкості навчання. На відміну від окремого RMSProp та класичного SGD, спільний RMSProp забезпечує більш прогнозоване зростання якості політик навіть за умови варіативних параметрів. Тому для всіх подальших експериментів у дослідженні обирається саме цей варіант оптимізації як основний.

3.4 Конфігурація експериментального середовища

У рамках експериментального дослідження було визначено єдину конфігурацію, яка використовувалась як для задач на іграх Atari, так і для симуляцій у середовищі TORCS. Усі експерименти проводились на одному комп’ютері без застосування графічних прискорювачів, що підкреслює ефективність обраних асинхронних методів. Для кожного тестового запуску залучалося 16 потоків акторів-учнів, які діяли незалежно, забезпечуючи паралельне навчання на окремих копіях середовища. Таке налаштування дозволяє пришвидшити збір даних та оновлення політики без потреби в централізованому буфері досвіду.

Оновлення параметрів нейронної мережі в усіх протестованих алгоритмах здійснювалося після кожних п’яти виконаних дій, що відповідає значенням $t_{max} = 5$ та $I_{update} = 5$. Як оптимізатор було обрано RMSProp зі спільною статистикою між потоками. Цей варіант довів свою ефективність у попередньому аналізі завдяки стабільній динаміці збіжності та низькій чутливості до параметрів початкової ініціалізації. У реалізації алгоритму було використано коефіцієнт згладжування $\alpha = 0{,}99$, що забезпечувало поступове оновлення середніх значень градієнтів у кожному кроці.

В алгоритмах на основі ціннісних функцій використовувалась глобальна цільова мережа, яка оновлювалась кожні 40 000 кадрів. Для actor-critic архітектури A3C структура нейронної мережі включала два виходи: один із softmax-активацією для політики та інший із лінійним виходом для функції цінності. Архітектура складалась із двох згорткових шарів (16 фільтрів розміром 8×8 зі кроком 4 та 32 фільтри 4×4 зі кроком 2), а також повнозв'язаного шару з 256 нейронів і активацією ReLU. Вихідний шар для value-based методів був лінійним, відповідно до кількості дій у середовищі.

Попередня обробка вхідних зображень для ігор Atari була стандартизована. Зокрема, застосовувалося приведення зображення до сірого кольору, зміна розміру до 84×84 пікселів та повторення дій (action repeat) з коефіцієнтом 4. Це дозволяє суттєво зменшити кількість кадрів для обробки, не втрачаючи при цьому критичної інформації. Знижка майбутніх нагород в усіх експериментах фіксувалася на рівні $\gamma = 0{,}99$, що відповідає стандартній практиці глибокого підкріплення.

Поведінка агентів у value-based методах додатково регулювалася параметром ε, що визначав ступінь дослідження. Його значення обиралося з трьох фіксованих варіантів за ймовірностями 0.4, 0.3 та 0.3 відповідно. Для кожного запуску ε анелювалося від 1 до мінімального значення (0.1, 0.01 або 0.5) протягом перших 4 мільйонів кадрів. У методі A3C до функції втрат було включено ентропійний доданок з коефіцієнтом $\beta = 0{,}01$, що сприяло підтриманню рівня стохастичності в політиці на ранніх етапах навчання.

Для забезпечення статистичної надійності результатів у кожному середовищі проводилося по 50 незалежних експериментів із різними початковими ініціалізаціями та випадковими значеннями швидкості навчання. Початкове значення learning rate вибиралось з логарифмічно рівномірного розподілу LogUniform у межах від 10^{-4} до 10^{-2} і поступово зменшувалося в процесі навчання. Всі порівняння з попередніми підходами здійснювалися з використанням фіксованих гіперпараметрів, що були підібрані згідно з опублікованими протоколами оцінки результатів. Це

дозволяє забезпечити об’єктивність та відтворюваність експериментального процесу.

3.5 Аналіз результатів експериментів

У цьому підрозділі здійснюється систематичний аналіз результатів, отриманих у ході проведення експериментів із застосуванням асинхронних методів навчання з підкріпленням. Основна увага приділяється порівнянню ефективності різних алгоритмів на основі середніх і медіанних показників, нормалізованих відносно результатів людини. Враховуються різні архітектури нейронних мереж, способи оптимізації, типи середовищ і тривалість навчання. У таблицях і графіках відображено, як саме змінюються підсумкові бали залежно від конфігурації, що дозволяє зробити висновки про чутливість моделей до гіперпараметрів і здатність до узагальнення.

Крім кількісних метрик, у підрозділі також аналізується стабільність навчання, розкиди результатів між різними запусками, вплив багатопотокового оновлення та загальна динаміка збіжності. Особлива увага приділяється порівнянню з попередніми підходами, зокрема DQN, Gorila та їх модифікаціями. Аналіз результатів дозволяє підтвердити гіпотезу про перевагу асинхронних методів A3C над класичними value-based підходами як за якістю отриманої політики, так і за стабільністю та обчислювальною ефективністю в умовах обмежених ресурсів.

3.5.1 Ігри Atari 2600

Основним середовищем для оцінки ефективності запропонованих асинхронних методів навчання з підкріпленням було Arcade Learning Environment, що емулює класичні ігри для Atari 2600. Це середовище широко використовується у дослідженнях глибокого RL завдяки наявності

великого набору задач із дискретними станами, чітко визначеними правилами та візуальним представленням, яке легко обробляється згортковими мережами. Для первинного тестування було обрано п’ять ігор – Beamrider, Breakout, Pong, Q*bert та Space Invaders. Метою експериментів було порівняння швидкості та якості навчання між базовим методом DQN та представленими асинхронними підходами – одно-кроковим Sarsa, одно-кроковим Q-learning, n-кроковим Q-learning та actor-critic з перевагою.

На графіках (рисунок 3.2) представлено залежність результативності агентів від часу тренування, виміряного в годинах. Результати показують, що всі асинхронні методи значно переважають DQN за швидкістю навчання. Наприклад, у грі Pong actor-critic досягає стабільно високих балів за значно коротший час, ніж інші методи, демонструючи стрімкий ріст продуктивності вже у перші години тренування. У грі Q*bert A3C демонструє перевагу над іншими підходами як у швидкості сходження, так і в максимальних досягнутих балах. Варто також відзначити, що n-крокові методи мають кращу динаміку навчання, ніж одно-крокові, завдяки ефективнішій передачі інформації про довготривалі залежності між станами і нагородами.

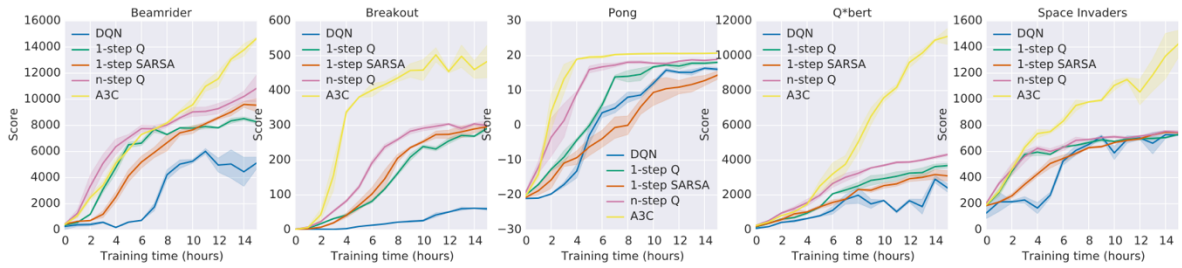

Рисунок 3.2 – Порівняння швидкості навчання алгоритмів

Детальні результати, зібрані по 57 іграх Atari, наведено в таблиці 3.1. Для кожного з методів обчислено середнє та медіанне значення, нормалізовані відносно досягнень людини. Алгоритми на основі DQN,

навіть у модифікованих версіях (Dueling DQN, Prioritized DQN), показують гірші результати порівняно з A3C. Зокрема, A3C з рекурентною архітектурою (LSTM) досягає найвищого середнього значення – 623%, демонструючи здатність моделі накопичувати інформацію про середовище та діяти в умовах часткової спостережуваності. Feedforward-версія A3C (FF) також значно перевищує DQN – 496.8% проти 121.9%.

Таблиця 3.1 – Середні та медіанні значення навчання

Алгоритм	Час навчання	Середнє значення	Медіанне значення
DQN	8 днів на GPU	121,9%	47,5%
D-DQN	8 днів на GPU	332,9%	110,9%
Dueling D-DQN	8 днів на GPU	343,8%	117,1%
Prioritized DQN	8 днів на GPU	463,6%	127,6%
A3C, FF	4 дня на GPU	496,8%	116,6%
A3C, LSTM	4 дня на GPU	623,0%	112,6%

Одним із ключових досягнень експериментів є те, що A3C-агенти навчаються на CPU за чотири дні, тоді як інші методи потребують вісім днів на GPU. Це доводить не лише перевагу A3C з точки зору якості, але й її обчислювальну ефективність – відсутність необхідності в дорогому обладнанні. Варто також зазначити, що навіть після одного дня тренування, A3C FF досягає середнього результату на рівні Dueling DQN, який тренується на GPU. Таке порівняння підкреслює потужність запропонованої архітектури та асинхронного підходу в цілому.

Ще однією особливістю експериментів є стабільність результатів. Було проведено 50 незалежних запусків для кожного алгоритму, кожен з яких мав свою початкову ініціалізацію та значення learning rate. Це дозволило продемонструвати, що A3C є менш чутливим до вибору гіперпараметрів, порівняно з іншими методами. Завдяки регуляризації

ентропії, агент A3C зберігає стохастичність на ранніх етапах тренування, що сприяє ефективному дослідженню середовища. Крім того, архітектура A3C дозволяє інтегрувати переваги таких підходів як Dueling DQN та Double Q-learning, що потенційно може дати ще вищі результати.

Таким чином, експерименти з іграми Atari 2600 підтвердили високу ефективність actor-critic архітектури з асинхронним навчанням. Ці результати показали, що навіть без спеціалізованого обладнання, агент A3C здатен досягати і перевершувати найкращі відомі результати, що робить його одним із найбільш перспективних рішень для задач підкріплення в складних середовищах.

3.5.2 3D-симулятор автоперегонів TORCS

Для глибшої перевірки ефективності асинхронних методів навчання з підкріпленням було проведено серію експериментів у 3D-симуляторі автоперегонів TORCS. Це середовище є значно складнішим, ніж ігри Atari, оскільки передбачає неперервний простір дій, фізичну динаміку та часткову спостережуваність. Крім того, завдання передбачає навігацію на високих швидкостях у просторі з перешкодами, що створює додаткове навантаження на здатність агентів до довготривалого планування, самокорекції та узагальнення. Експерименти проводилися в чотирьох різних конфігураціях: повільний автомобіль без суперників, повільний автомобіль з суперниками, швидкий автомобіль без суперників та швидкий автомобіль з суперниками.

Результати, представлені на графіках (рисунок 3.3), демонструють, як змінювалася результативність агентів протягом 48 годин тренування. У всіх чотирьох конфігураціях найвищі результати стабільно демонстрував алгоритм асинхронного actor-critic, який швидко досягав рівня гри тестувальника-людини та утримував його з незначними коливаннями. Важливо підкреслити, що агент A3C показував високу ефективність уже через 5–10 годин тренування, тоді як інші алгоритми досягали лише частини

його показників. Зокрема, в умовах відсутності суперників (верхні графіки), actor-critic агент швидко стабілізував стратегію руху по треку з мінімізацією помилок і втрат швидкості.

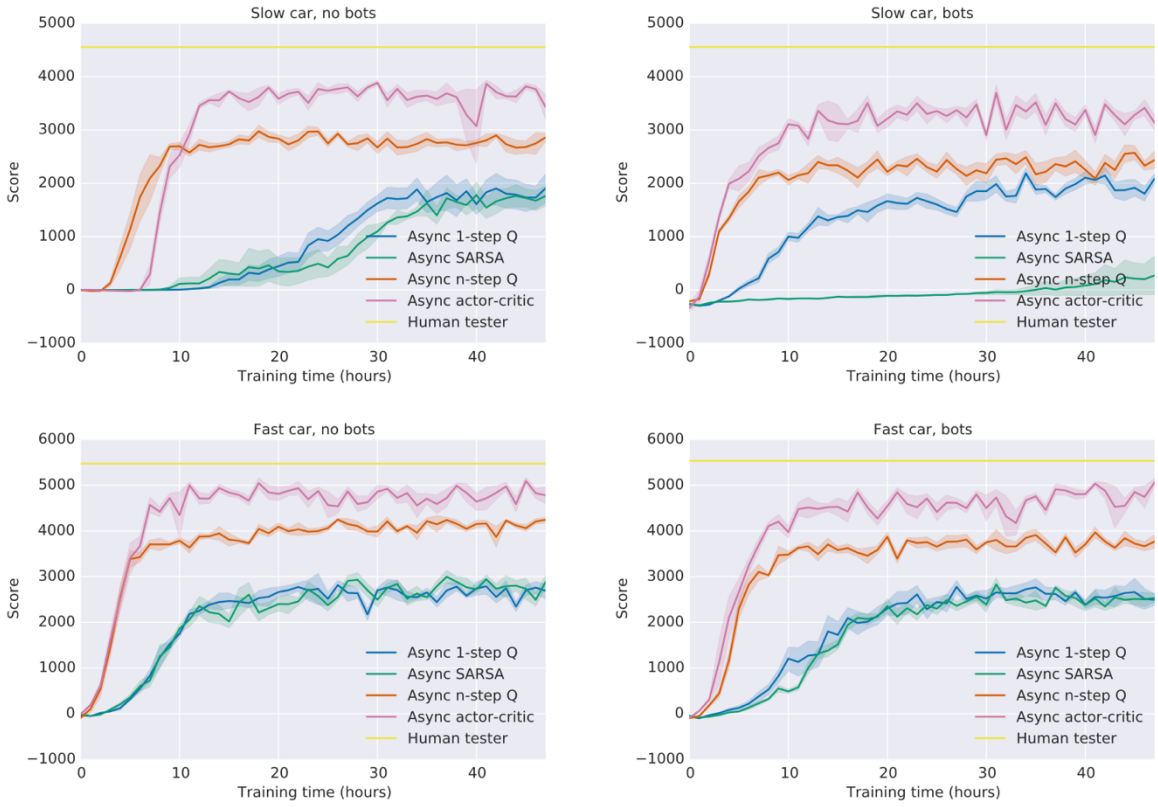

Рисунок 3.3 – Порівняння алгоритмів на симуляторі TORCS

Найбільш критичними стали конфігурації з наявністю суперників, графіки праворуч, де агент повинен не лише проходити трасу, а й уникати зіткнень, вибудовувати оптимальні траєкторії з урахуванням поведінки інших учасників. Тут actor-critic знову виявився найефективнішим – він не тільки швидше навчився, але й показав вищу стабільність. Інші методи, зокрема одно-кроковий Q-learning та SARSA, демонстрували поступове покращення, але не могли перевищити позначки 2000–3000 балів, тоді як A3C стабільно утримував показники понад 4000.

У випадках із швидким автомобілем без суперників (нижній лівий графік) видно, що A3C та n-кроковий Q-learning навчаються швидше, ніж

інші алгоритми, що свідчить про здатність цих методів адаптуватися до задач з високим ризиком помилок. Водночас SARSA показує нижчу продуктивність в усіх конфігураціях, що може бути пов’язано з її on-policy природою та чутливістю до вибору дії під час тренування. Особливо значущою є різниця між n-кроковим та одно-кроковим Q-learning: додавання декількох кроків назад дозволяє агенту ефективніше враховувати довгострокові наслідки дій.

Ще один важливий аспект, який ілюструє рисунок, полягає в узагальнювальній здатності алгоритмів. Незалежно від конфігурації – повільна чи швидка машина, присутність суперників чи ні – actor-critic стабільно забезпечує високий рівень адаптації. Це свідчить про сильну інтеграцію між блоками оцінки та прийняття рішень, що дозволяє A3C реагувати на складні зміни в середовищі без необхідності повного переобучення.

Таким чином, експерименти у TORCS підтверджують загальні висновки, зроблені на іграх Atari: actor-critic є найефективнішим методом серед усіх протестованих підходів. Завдяки паралельному навчанню, механізмам переваги та використанню ентропійної регуляризації, цей підхід забезпечує високу продуктивність навіть у середовищах із неперервною динамікою та обмеженою спостережуваністю. Результати свідчать про те, що A3C є придатним для застосування у складних задачах навігації, планування та керування у реальному часі.

3.5.3 3D-середовище Labyrinth

На завершальному етапі експериментального дослідження було проведено додаткову серію тестувань у новому середовищі Labyrinth, яке є візуально складним тривимірним середовищем з частковою спостережуваністю. Це середовище моделює задачу орієнтування в лабіринтах, які щоразу генеруються випадково, тим самим ставлячи перед

агентом виклик загального дослідження і побудови внутрішнього уявлення про простір. На відміну від попередніх середовищ, таких як Atari чи TORCS, Labyrinth не надає агенту жодної попередньої інформації про структуру простору, тому успішне навчання вимагає не лише локальної реакції на сигнали, але й формування стратегій навігації та планування.

У межах експерименту агент на початку кожного епізоду розміщувався у новому лабіринті, сформованому з кімнат і з'єднувальних коридорів. Його мета полягала в тому, щоб зібрати якомога більше балів за 60 секунд. У середовищі існували два типи об'єктів, за які агент отримував винагороду: яблука з винагородою 1 бал та портали з винагородою 10 балів. Після активації порталу агент телепортувався в іншу випадкову точку лабіринту, а всі яблука автоматично з'являлися заново. Оптимальна стратегія передбачала знайти портал якомога раніше, а потім багаторазово повертатися до нього після збору яблук для повторного збору винагород.

Для навчання було обрано модифікацію алгоритму A3C з використанням рекурентної нейронної мережі LSTM. Агент отримував на вхід лише візуальні спостереження – тривимірні зображення у форматі RGB розміром 84×84 пікселі. Такий формат імітує реальні умови роботи роботизованих систем, які мають обмежену спостережуваність і використовують лише зображення з камери без доступу до внутрішніх параметрів середовища. Рекурентність LSTM дозволила агенту ефективно зберігати послідовну інформацію, в тому числі про структуру простору, розташування об'єктів та місцезнаходження порталу, навіть після телепортації.

Це завдання виявилось суттєво складнішим порівняно з керуванням у TORCS або Atari, оскільки агент не може покладатися на стабільну структуру середовища. Він повинен щоразу формувати нову стратегію дослідження, адаптовану до нових умов, що вимагає високого ступеня узагальнення. Успішне проходження цього тесту демонструє здатність алгоритму не просто запам'ятовувати послідовність дій, а навчатися гнучкій

поведінці, заснованій на аналізі візуального простору та розпізнаванні шаблонів.

Середній фінальний результат агента A3C, натренованого у середовищі Labyrinth, склав близько 50 балів. Це свідчить про те, що розроблений агент зміг виробити ефективну стратегію навігації у випадкових тривимірних лабіринтах. Результати доводять, що навіть за відсутності спеціалізованої сенсорики та зі спрощеним вхідним вектором, агент здатен навчитися узагальненій політиці пошуку винагород, що є надзвичайно важливим у завданнях автономної навігації, дослідження нових просторів та роботи в умовах обмеженої інформації.

3.5.4 Масштабованість та ефективність використання даних

Одним із ключових аспектів асинхронних методів навчання з підкріпленням є здатність до масштабування з використанням більшої кількості паралельних потоків агентів. З цією метою було проведено експериментальне дослідження, яке порівнює швидкість навчання та ефективність використання даних для різних алгоритмів при варіативній кількості паралельних агентів (1, 2, 4, 8, 16 потоків). Усі агенти навчаються одночасно, використовуючи спільну нейронну мережу, що оновлюється асинхронно.

Загальна картина масштабованості відображена у таблиці 3.2, яка містить середнє прискорення навчання для чотирьох основних алгоритмів – 1-step Q, 1-step SARSA, n-step Q та A3C. Очевидно, що всі алгоритми демонструють чітку позитивну динаміку зі зростанням кількості потоків. Наприклад, 1-step Q при переході з одного потоку до 16-ти забезпечує приріст у швидкості навчання майже в 24 рази, що є найвищим показником серед усіх протестованих підходів. Для n-step Q та SARSA приріст становить відповідно 17.2 та 22.1, тоді як у A3C він дещо нижчий – 12.5. Це пояснюється більш складною взаємодією між політикою та критиком, що

ускладнює стабільне навчання при великій кількості оновлень у паралельному режимі.

Таблиця 3.2 – Середнє прискорення навчання кожного алгоритму при збільшенні кількості паралельних агентів

Алгоритм	Кількість потоків				
	1	2	4	8	16
1-step Q	1.0	3.0	6.3	13.3	24.1
1-step SARSA	1.0	2.8	5.9	13.1	22.1
n-step Q	1.0	2.7	5.9	10.7	17.2
A3C	1.0	2.1	3.7	6.9	12.5

Детальні графіки (рисунок 3.4 – 3.7) демонструють, як змінюється ефективність тренування для кожного з алгоритмів на п’яти класичних іграх Atari (Beamrider, Breakout, Pong, Q*bert, Space Invaders) при зміні кількості потоків. Графіки (рисунок 3.4) відображає навчання для 1-step Q-learning. Видно, що збільшення кількості потоків з 1 до 16 значно прискорює досягнення високих результатів. При цьому криві з більшою кількістю потоків демонструють вищі фінальні результати й меншу дисперсію.

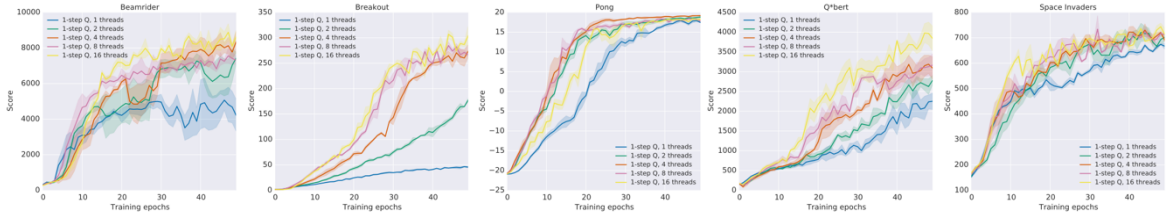

Рисунок 3.4 – Порівняння ефективності використання різної кількості акторів для 1-step Q

Графіки (рисунок 3.5) присвячені n-step Q-learning. Аналогічно як до 1-step Q, видно, що приріст за кількістю потоків значно пришвидшує

навчання. Особливо яскраво це проявляється у грі Beamrider, де результати зростають майже в три рази у фінальній точці. Зазначимо також, що n-step Q більш чутливий до кількості потоків у грі Q*bert, де використання 16 потоків дає найвищі результати серед усіх конфігурацій.

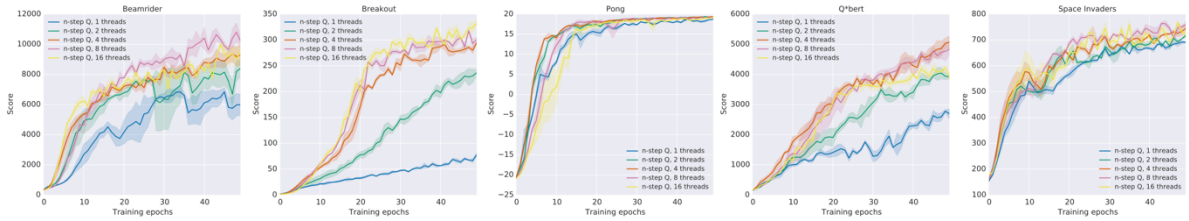

Рисунок 3.5 – Порівняння ефективності використання різної кількості акторів для n-step Q

Зображення (рисунок 3.6) проаналізовано алгоритм A3C. Хоча його прискорення дещо поступається value-based методам, A3C демонструє вищу стабільність у більшості ігор. Наприклад, у грі Pong навчання відбувається стрімко навіть при малій кількості потоків, а при 8 і 16 потоках результати вирівнюються рано і залишаються високими до кінця експерименту. Також варто відзначити, що A3C демонструє значну стійкість до флуктуацій.

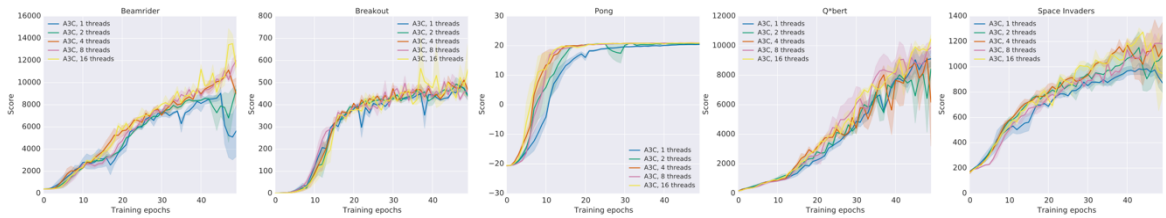

Рисунок 3.6 – Порівняння ефективності використання різної кількості акторів для A3C

Нижче ілюстрація результатів для 1-step SARSA (рисунок 3.7). Хоча цей метод загалом демонструє нижчі результати у порівнянні з іншими

підходами, збільшення кількості потоків все ж таки покращує його продуктивність. Найбільший приріст видно в іграх Breakout та Q*bert, що може свідчити про здатність SARSA ефективно працювати в умовах, де політика часто змінюється в процесі навчання.

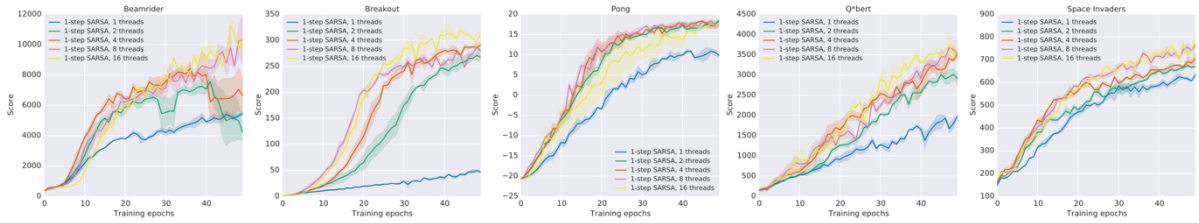

Рисунок 3.7 – Порівняння ефективності використання різної кількості акторів для SARSA

Загалом, результати показують, що всі методи виграють від масштабування – як у плані швидкості досягнення високих балів, так і в плані фінальної продуктивності. Найбільш масштабованими виявилися value-based методи, зокрема 1-step Q, який демонструє майже лінійне прискорення. Натомість A3C проявляє найвищу стабільність, зокрема у складних ігрових сценаріях. Це робить асинхронні підходи надзвичайно перспективними для завдань, де обчислювальні ресурси можуть масштабуватися паралельно.

Детальний аналіз показує не лише підвищення ефективності за кількістю даних, але й прискорення процесу навчання при збільшенні кількості паралельних акторів. Згідно з результатами, асинхронні методи демонструють значне скорочення часу досягнення високих результатів, що особливо помітно при використанні 8 і 16 потоків.

Представлені графіки (рисунок 3.8 – 3.11) залежності середнього балу від часу навчання для різної кількості акторів при застосуванні алгоритмів 1-step Q, n-step Q, A3C та 1-step SARSA. Кожна лінія на графіках відповідає

певній кількості потоків, що дозволяє безпосередньо оцінити ефект масштабування на швидкість досягнення високої якості політики.

Наприклад, у випадку алгоритму 1-step Q (рисунок 3.8) застосування 16 потоків дозволяє досягти рівня продуктивності, недосяжного при одному потоці навіть за вдвічі більший час.

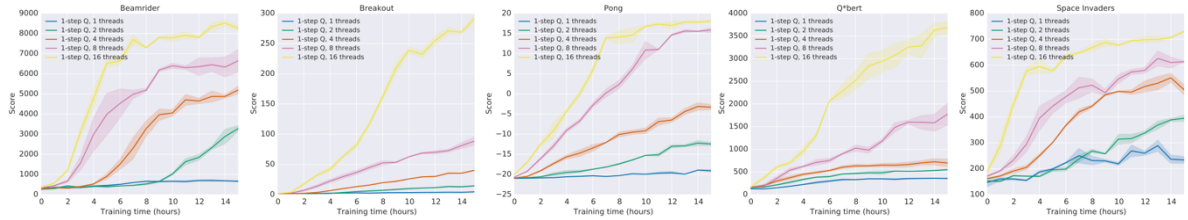

Рисунок 3.8 – Порівняння швидкості навчання різної кількості акторів для 1-step Q

Подібна тенденція спостерігається для алгоритму n-step Q, де збільшення потоків приводить до швидшого виходу на плато стабільної продуктивності (рисунок 3.9).

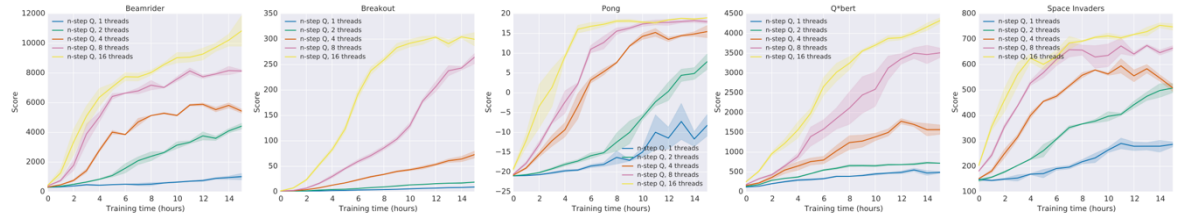

Рисунок 3.9 – Порівняння швидкості навчання різної кількості акторів для n-step Q

У випадку алгоритму A3C (рисунок 3.10) переваги паралельного навчання також очевидні: навіть за умов високої складності середовищ агент навчається ефективніше, використовуючи більшу кількість паралельних акторів. Особливо це помітно у грі Beamrider, де час

досягнення високих балів скоротився майже удвічі при переході від 1 до 16 потоків.

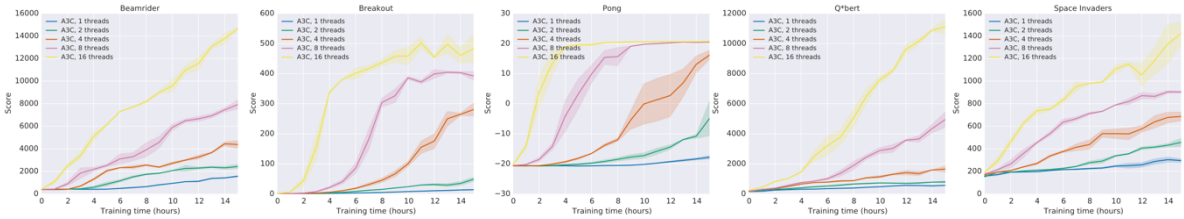

Рисунок 3.10 – Порівняння швидкості навчання різної кількості акторів для A3C

Алгоритм 1-step SARSA (рисунок 3.11) демонструє аналогічний характер зростання швидкості навчання при збільшенні кількості потоків, однак із дещо меншою інтенсивністю порівняно з іншими розглянутими методами. Це свідчить про загальну властивість асинхронних методів покращувати як ефективність використання даних, так і швидкість набуття навичок при розподіленні навантаження між паралельними агентами.

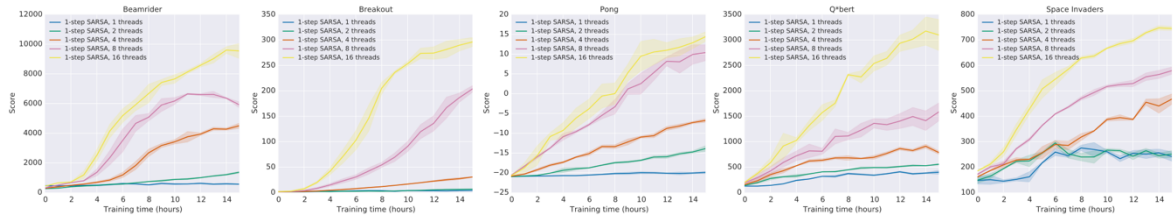

Рисунок 3.11 – Порівняння швидкості навчання різної кількості акторів для SARSA

Таким чином, проведені експерименти підтвердили, що масштабування кількості потоків є критичним чинником для прискорення процесу навчання без погіршення якості кінцевих політик. Це робить

асинхронні підходи вельми привабливими для практичного застосування у складних завданнях навчання з підкріпленням.

3.5.5 Надійність і стабільність

Цей підрозділ присвячено аналізу того, наскільки результати навчання асинхронних алгоритмів з підкріпленням залишаються стабільними при зміні ключових гіперпараметрів, зокрема швидкості навчання. Однією з найважливіших вимог до практичного застосування алгоритмів DRL є здатність демонструвати стабільні результати при неідеальних умовах, наприклад, при варіації початкових умов, різних конфігураціях середовища або відмінностях у гіперпараметрах. Основна мета аналізу – визначити, які з розглянутих методів мають найвищу надійність при зміні цих параметрів.

На графіках нижче (рисунок 3.12 – 3.13) показано залежність фінального балу від значення швидкості навчання для кожного з алгоритмів у п’яти іграх Atari. Графіки ілюструють, що навіть незначні зміни гіперпараметра можуть мати суттєвий вплив на результати тренування. Особливо вразливими виявилися алгоритми one-step Q та one-step SARSA, для яких невдало обраний коефіцієнт навчання призводив до значного зниження продуктивності. Це видно, зокрема, на прикладах ігор Breakout та Beamrider, де за високих або низьких значень learning rate алгоритми або взагалі не навчалися, або швидко переходили в розвалену політику.

На відміну від них, алгоритми n-step Q та A3C демонструють вищу толерантність до вибору learning rate. У більшості випадків вони зберігають прийнятну продуктивність у широкому діапазоні параметрів. Зокрема, на графіках A3C (рисунок 3.12) можна побачити, що його продуктивність знижується значно повільніше поза оптимальним діапазоном learning rate, ніж у конкурентів. Це робить його більш придатним для практичного

використання, оскільки зменшує потребу в тривалому процесі налаштування гіперпараметрів.

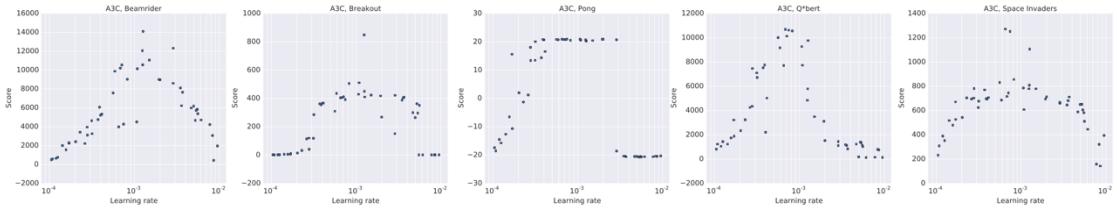

Рисунок 3.12 – Графіки розсіювання для A3C

У цілому, результати свідчать про те, що алгоритми з багатокроковою актуалізацією та побудовані на actor-critic підходах забезпечують не лише кращу ефективність, а й вищу стабільність навчання.

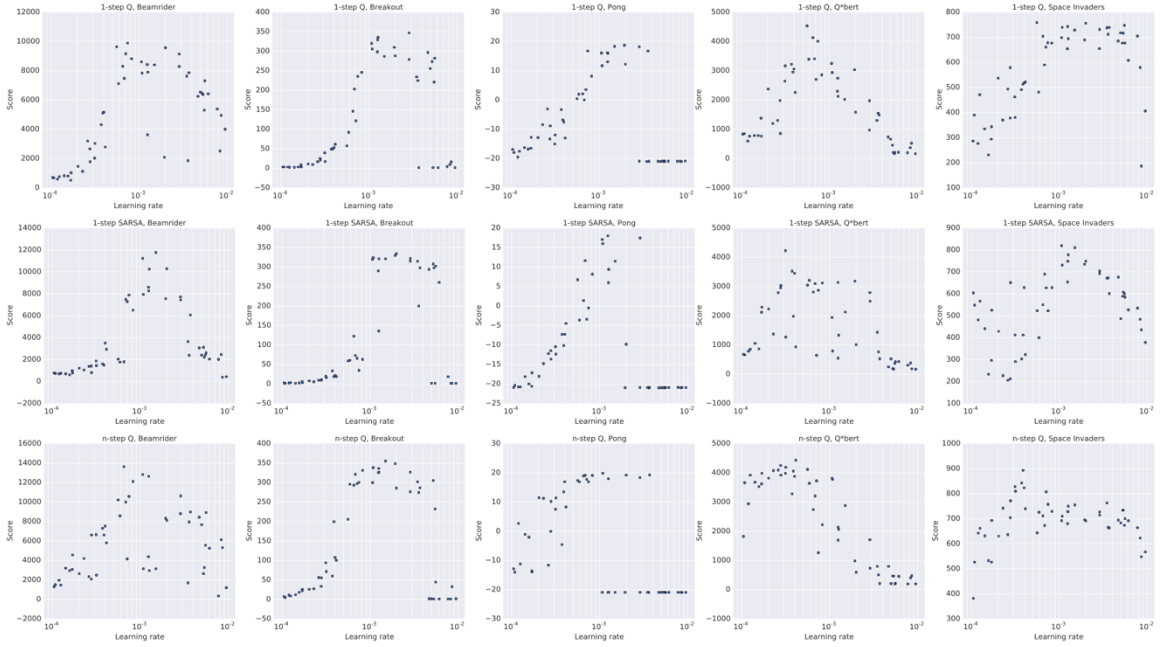

Рисунок 3.13 – Графіки розсіювання для трьох інших алгоритмів

Вони менш чутливі до випадкової ініціалізації та варіацій швидкості навчання.

ВИСНОВКИ

У результаті виконання кваліфікаційної роботи було повністю реалізовано поставлену мету – розробити та дослідити асинхронні алгоритми навчання з підкріпленням, здатні ефективно працювати в умовах обмежених обчислювальних ресурсів, забезпечуючи при цьому надійне і стабільне навчання політик на основі глибоких нейронних мереж. У процесі дослідження було реалізовано та порівняно кілька варіантів алгоритмів: одно-кроковий Sarsa, одно-кроковий Q-learning, n-кроковий Q-learning та A3C. Для всіх методів було здійснено адаптацію до асинхронного багатопотокового середовища, що дозволило оцінити масштабованість, стійкість до варіативності параметрів та здатність до узагальнення.

Завдяки великій серії експериментів у трьох різних середовищах – Atari 2600, 3D TORCS і Labyrinth – було отримано вичерпну кількісну та якісну характеристику досліджуваних підходів. Найбільший приріст стабільності та продуктивності показав метод A3C, який, працюючи на CPU, зміг перевершити інші методи навіть при меншій кількості ресурсів. Наприклад, середнє значення для A3C з LSTM склало 623.0 %, тоді як для DQN – лише 121.9 %. Показники на TORCS також продемонстрували перевагу actor-critic архітектур у динамічних та складних задачах з безперервним потоком вхідних даних. Дослідження у Labyrinth підтвердили здатність агента до ефективного вивчення стратегій у рандомізованих 3D-середовищах, де він досяг середнього балу близько 50, що вказує на високий рівень узагальнення при мінімумі інформації на вході.

Усі реалізовані алгоритми демонстрували масштабованість при збільшенні кількості акторів. Експерименти засвідчили, що зростання кількості паралельних потоків позитивно впливає на швидкість та якість навчання – у випадку 1-step Q приріст становив до 24.1 разів при переході від 1 до 16 потоків. Аналогічні результати були зафіксовані для n-step Q, SARSA та A3C. Водночас, було доведено, що методи зі спільною

статистикою (наприклад, RMSProp Shared) забезпечують більш стабільну оптимізацію, зменшуючи залежність від початкових параметрів, зокрема швидкості навчання.

Аналіз чутливості алгоритмів до вибору гіперпараметрів продемонстрував, що actor-critic методи є більш стійкими до варіацій коефіцієнтів навчання, забезпечуючи ширший діапазон стабільної роботи, на відміну від value-based алгоритмів, у яких спостерігається різке падіння продуктивності за межами оптимального діапазону. Це дозволяє зробити висновок про доцільність застосування actor-critic архітектур у практичних системах зі змінними умовами середовища.

Отримані результати дозволяють рекомендувати використання асинхронних actor-critic методів як найбільш універсальних та ефективних у задачах навчання з підкріпленням у складних динамічних середовищах. Завдяки поєднанню стабільного процесу оновлення параметрів, можливості паралельного навчання та гнучкості архітектурної реалізації, ці методи демонструють високу адаптивність до широкого спектра завдань – від класичних ігрових сценаріїв до 3D-симуляторів з частковою спостережуваністю. Асинхронні actor-critic підходи не лише зменшують залежність від обчислювальних ресурсів, але й забезпечують прискорене навчання без погіршення якості політики, що є критично важливим у задачах реального часу.

У майбутньому доцільним є розширення цієї роботи шляхом використання покращених архітектур, зокрема агентів на основі трансформерів, які дозволяють моделювати довготривалі залежності у послідовностях та ефективно працювати з великомасштабними входами.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Aggarwal C. C. Neural Networks and Deep Learning: A Textbook. Springer, 2018.

2. Asynchronous methods for deep reinforcement learning / V. Mnih et al. *International Conference on Machine Learning*. 2016. P. 1928–1937. URL: https://arxiv.org/abs/1602.01783 (date of access: 30.01.2025).

3. Brownlee J. Mastering Machine Learning Algorithms. Machine Learning Mastery, 2016.

4. Chollet F. Deep Learning with Python. Manning Publications, 2021.

5. Continuous control with deep reinforcement learning / T. P. Lillicrap et al. *International Conference on Learning Representations (ICLR)*. 2016. URL: https://arxiv.org/abs/1509.02971 (date of access: 25.02.2025).

6. DeepMind Lab / C. Beattie et al. 2016. URL: https://arxiv.org/abs/1612.03801 (date of access: 30.03.2025).

7. Dueling network architectures for deep reinforcement learning / Z. Wang et al. *International Conference on Machine Learning*. 2016. P. 1995–2003. URL: https://arxiv.org/abs/1511.06581 (date of access: 30.03.2025).

8. Géron A. Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow. O'Reilly Media, 2019.

9. Goodfellow I., Bengio Y., Courville A. Deep Learning. MIT Press, 2016.

10. Hausknecht M., Stone P. Deep recurrent Q-learning for partially observable MDPs. *AAAI Fall Symposium Series*. 2015. URL: https://arxiv.org/abs/1507.06527 (date of access: 14.04.2025).

11. Human-level control through deep reinforcement learning / V. Mnih et al. *Nature*. 2015. Vol. 518, no. 7540. P. 529–533. URL: https://arxiv.org/abs/1507.06527 (date of access: 15.04.2025).

12. Kaelbling L. P., Littman M. L., Moore A. W. Reinforcement Learning: A Survey. *Morgan Kaufmann*. 1996.

13. Kelleher J. D., Mac Carthy M. Deep Learning. MIT Press, 2020.

14. Lample G., Chaplot D. S. Playing FPS games with deep reinforcement learning. *AAAI Conference on Artificial Intelligence*. 2017. URL: https://arxiv.org/abs/1609.05521 (date of access: 17.04.2025).

15. Lapan M. Deep Reinforcement Learning Hands-On: Apply Modern RL Methods, with Deep Q-networks, Value Iteration, Policy Gradients, TRPO, AlphaGo Zero and More. Packt Publishing, 2020.

16. Nielsen M. A. Neural Networks and Deep Learning: A Free Online Book. Determination Press, 2015.

17. Patterson J., Gibson A. Deep Learning: A Practitioner's Approach. O'Reilly Media, 2017.

18. Rainbow: Combining improvements in deep reinforcement learning / M. Hessel et al. *AAAI Conference on Artificial Intelligence*. 2018. URL: https://arxiv.org/abs/1710.02298 (date of access: 30.03.2025).

19. Reinforcement Learning: State-of-the-Art / ed. by M. Wiering, M. Van Otterlo. Springer, 2012.

20. Russell S. J., Norvig P. Artificial Intelligence: A Modern Approach. Pearson, 2020.

21. Silver D., Sutton R., Müller M. Temporal-Difference Learning and TD-Gammon. *MIT Compilation Series on Machine Intelligence*. 2010.

22. Sutton R. S., Barto A. G. Reinforcement Learning: An Introduction. MIT Press, 2018.

23. The Arcade Learning Environment: An evaluation platform for general agents / M. G. Bellemare et al. *Journal of Artificial Intelligence Research*. 2013. Vol. 47. P. 253–279. URL: https://jair.org/index.php/jair/article/view/10819 (date of access: 30.04.2025).

24. Zhang C., Ma Y. Ensemble Machine Learning: Methods and Applications. Springer, 2021.