ОДЕСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ імені І.І.МЕЧНИКОВА
(повне найменування вищого навчального закладу)

Факультет математики, фізики та інформаційних технологій
(повне найменування інституту, назва факультету (відділення))

Кафедра математичного забезпечення комп'ютерних систем
(повна назва кафедри (предметної, циклової комісії))

Кваліфікаційна робота

на здобуття ступеня вищої освіти «бакалавр»
(освітньо-кваліфікаційний рівень)

Безмодельні алгоритми навчання з підкріпленням
Model-free reinforcement learning algorithms

Виконав: студент денної форми навчання
спеціальності 126 – Інформаційні системи та технології .
(шифр і назва напряму підготовки, спеціальності)
Освітня програма «Інформаційні системи та технології»
(назва освітньої програми)
Черноіваненко Сергій Миколайович
(прізвище, ім'я, по-батькові)

Керівник к. т. н., доц. Пенко В.Г.
(науковий ступінь, вчене звання, прізвище та ініціали, **підпис**)

Рецензент к.ф.-м.н. Антоненко О.С.
(науковий ступінь, вчене звання, прізвище та ініціали)

Рекомендовано до захисту:
Протокол засідання кафедри
№___ від «___»_____2023 р.
Завідувач кафедри
_____ Євгеній МАЛАХОВ
(підпис) (ім'я, прізвище)

Захищено на засіданні ЕК № ___
протокол № __ від «_ _» ____ 2023 р.
Оцінка _____/_____/_____
(за національною шкалою, шкалою ECTS, бали)
Голова ЕК
_____ Володимир ВИЧУЖАНІН
(підпис) (ім'я, прізвище)

Одеса - 2023

АНОТАЦІЯ

Метою даної кваліфікаційної роботи є апробація та модифікація алгоритму безмодельного навчання (а саме алгоритм Q-навчання).

В процесі досягнення мети роботи був виконаний огляд основних теоретичних положень про область навчання з підкріпленням, були детально розглянуті принципи роботи алгоритмів безмодельного навчання з підкріпленням, запропонована і протестована модифікація алгоритму Q-навчання для прискорення процесу навчання за рахунок використання моделі, а також вивчені результати інших досліджень щодо даної методики.

Для практичної апробації була розроблена тестувальна платформа на основі мови програмування Python, бібліотеки оточень агентів навчання з підкріпленням OpenAi Gymnasium та фреймворку машинного навчання загального призначення PyTorch.

Результатом даної роботи є бібліотека компонент агентів навчання з підкріпленням, додаток з консольним інтерфейсом для тестування розроблених агентів навчання з підкріпленням та звіт по результатах апробації із рекомендаціями та пропозиціями щодо подальших напрямків дослідження.

ABSTRACT

The purpose of this qualification work is approbation and modification of the model-free learning algorithm (namely, the Q-learning algorithm).

In the process of achieving the purpose of the work, a review of the main theoretical provisions in the field of reinforcement learning was carried out, the working principles of model-free reinforcement learning algorithms were considered in detail, a modification of the Q-learning algorithm was proposed and tested to speed up the learning process through the use of a model, and the results of other studies on similar to prosed acceleration method were also overviewed.

For practical approbation, a testing platform was developed based on the Python programming language, the OpenAi Gymnasium reinforcement learning agent environment library, and the PyTorch general-purpose machine learning framework.

The result of this work is a library of components of reinforcement learning agents, an application with a console interface for testing the developed reinforcement learning agents, and a report on the results of the approbation with recommendations and suggestions for further research directions.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ, УМОВНИХ ПОЗНАЧЕНЬ ТА ТЕРМІНІВ

НП – навчання з підкріпленням

ГНП – глибинне навчання з підкріпленням

БНП – безмодельне навчання з підкріпленням

ФС – файлова система

ВСТУП

Дослідження в сфері машинного навчання, як підгалузі штучного інтелекту, почались ще більш ніж півстоліття потому, але, за неможливістю ефективно використовувати розроблену теорію на практиці, яка була зумовлена, в першу чергу, обмеженим обчислювальним ресурсом, упродовж довгого часу після укладення теоретичних основ ця сфера була непопулярною, майже ніде не використовувалась і не розвивалась . Новий бум у дослідженнях машинного навчання і успішного використання його на практиці прийшовся на 1990-2010 рр. нашого століття, коли прогрес у апаратному забезпеченні обчислювальних машин нарешті простимулював незалежних дослідників і, згодом, комерційних інвесторів поновити роботу у цій сфері [1].

У наш час найпопулярнішими напрямками дослідження і застосування на практиці стали підгалузі глибинного навчання – навчання моделей з вчителем і без вчителя, які можуть вирішувати задачі з класифікації, прогнозування, розпізнавання образів, кластерізації, тощо – що стало наслідком розширення можливостей зі зберігання і обробки даних (задач датамайнингу) з одного боку, і розширення можливостей моделювати середовище і прикладну область різноманітних задач з іншого боку. Але існує також безліч інших менш популярних напрямків, теоретичні основи яких також були закладенні ще раніше, але великого прогресу у дослідженнях, порівняльно з глибинним навчанням, досягнуто не було. Одним з таких напрямків і є навчання з підкріпленням.

Формування напрямку НП почалось ще наприкінці 50-х рр. попереднього століття, коли дослідники почали вирішувати проблему «оптимального керування», тобто розробки регулятора, що мінімізує якусь міру поведінки динамічної системи [2]. Один з підходів до вирішення цієї задачі, за яким і було засновано НП і весь його теоретичний базис, був розроблений Річардом Беллманом. Саме він ввів поняття стану динамічної

системи, функції цінності і сформулював так зване рівняння Беллмана – найголовніші поняття в НП – а згодом і інші різні методи вирішення задач НП.

Але всі ці теоретично вірні і ефективні рішення упираються в явище, яке Беллман називав «прокляттям розмірності» – потреби в обчислювальних ресурсах зростають експоненціально при збільшенні розмірності простору параметрів стану середовища. На щастя, вже в 80-х роках були зроблені доволі вдалі спроби вирішити цю проблему за допомогою тогочасних досягнень у сфері глибинного навчання і нейронних мереж (наприклад в роботі [3]), тому не дивно, що з новою віхою розробок, досліджень і досягнень у цій сфері, НП також набуло другого дихання. Методи поєднання глибинного навчання і НП відносять до окремої категорії так званого глибинного НП, який демонструє змішанні результати, залежні від конкретної розв'язуваної задачі, що не заважає дослідникам вважати цю сферу перспективним напрямком [4, 5].

Основним вкладником у розвиток ГНП є компанія DeepMind, яка займається дослідженнями у різних галузях штучного інтелекту, зокрема і методів НП. Велика кількість доповідей у галузі ГНП, які надходять не лише від працівників DeepMind, але й різних ентузіастів, які використовують відкриту інфраструктуру для досліджень ГНП, що була розроблена цією кампанією, доповіді від інших дослідників, а також окремі випадки використання НП у комерційних галузях (наприклад автоматизація системи охолодження у датацентрах Google [6]) показують, що напрямок НП і глибинного НП користується хоч і значно меншою, аніж звичайне глибинне навчання, але все ж таки чималою популярністю [7].

За мету даної кваліфікаційної роботи ставиться апробація та модифікація одного з основних алгоритмів БНП. В процесі досягнення цієї мети було:

1. розглянуто основні варіанти алгоритмів БНП;
2. розглянуто варіанти модельного прискорення навчання безмодельних алгоритмів НП;
3. запропоновано власну модифікацію;

4. запрограмовано модуль розробки, навчання і демонстрації роботи агентів БНП на основі бібліотеки зовнішніх оточень агентів НП Gymnaisum;
5. презентовано аналіз отриманих результатів.

1 ТЕОРЕТИЧНІ ВІДОМОСТІ ПРО НП

Навчання з підкріпленням (*англ.* Reinforcement Learning, далі – НП) відноситься до різновиду метода машинного навчання, за яким агент отримує відкладену винагороду на наступному часовому русі, аби оцінити свою попередню дію. В основному цей метод використовувався в іграх (класичних настільних (шахи, шашки, нарди) або ж у популярних в сучасності відеоіграх), надаючи результати принаймні на тому ж рівні, що і люди, а, у чисельних випадках, навіть перевершують їх. Але цей метод спрямований на будь-яку задачу оптимізації чи прийняття рішень, тобто цей популярний напрямок не обмежує жодним чином ту множину задач, яку можна вирішити за допомогою НП. Останнім часом, коли алгоритми методу розвиваються в комбінації з нейронними мережами, вони становляться спроможними вирішувати більш складні завдання і проблеми не лише у теорії, а й практично. Але у той самий час, при використовуванні цього підходу, виникає низка проблем і перешкод, і ця сфера машинного навчання потребує більше досліджень для отримання дійсних і універсальних методів реалізації.

1.1. Основні поняття

Найголовнішими сутностями методу НП є агент і оточення. Оточення – це об’єкт або група об’єктів на які впливає агент, а агент, відповідно – це той алгоритм, який спостерігає за оточенням, і виконує у певний момент часу якусь дозволену дію.

Процес взаємодії агенту починається з того, що він спостерігає поточний стан оточення (або оточення надає свій стан агенту – ця різниця не є принциповою, якщо виконуються певній формальні правила, які будуть зазначені далі), агент виконує над оточенням якусь дію (множину можливих дій, яка залежить від поточного стану, агент теж дуже часто отримує від оточення), оточення змінює свій стан і надає агенту винагороду, відповідну до нового стану.

У дослідницькій літературі прийняті наступні позначення [2]:

- *A* або *a* (action) – дія;
- *S* або *s* (state) – стан (залежно від інфіксу додатково позначають порядкове положення стану у епізоді чи належність стану до часу, але просто *s* зазвичай відповідає поточному стану, а *s'* – наступному після поточного стану після виконання дії над ним);
- r (reward) – миттєва винагорода, по сутності є оцінкою оточення останньої дії, виконаної агентом;
- R – підсумкова вигода;
- π (policy) – політика або стратегія, яку використовує агент, для визначення наступної дії на підставі поточного стану;
- $V_\pi(s)$ (value) – очікувана підсумкова вигода або *цінність* стану. Це функція, яка відображає яку вигоду (середню, очікувану, сумарну, залежно від обраного метода оцінювання, які відрізняються між собою швидкістю та якістю збіжності до реального значення) отримає агент у результаті епізоду (буде розглянуте далі), якщо буде дотримуватися стратегії π, тобто виражає цінність поточного стану у довгостроковій перспективі;
- $Q_\pi(s, a)$ (quality) – якість дії у певному стані. Аналогічна до оцінки V(s), але відображає очікувану вигоду у випадку, коли агент у поточному стані *s* виконає дію *a*, після чого буде дотримуватися стратегії π. Також правильно буде сказати, що це оцінка (або думка) про цінність дії *a* над станом *s* з точки зору стратегії π;
- Епізод – послідовність переходів від одного стану до наступного, що закінчується особливим *термінальним* станом, за яким слідує або повернення до звичайного початкового стану, або ж до вибірки із стандартного розподілення початкових станів.

Мета НП – отримати певну стратегію (функцію, що відображає поточний стан оточення на певну дію агенту), яка буде максимізувати

отриману за епізод вигоду. Саме тому, коректне визначення функції винагороди для агенту, яка буде правильно відображати ціль навчання – це один з головних труднощів використання НП для вирішення прикладних задач.

Окремо слід розглянути поняття *вигоди*, тому що спосіб її обчислення також має відображати сутність задачі. У першу чергу, задачі розбиваються на ті, що складаються з епізодів та безперервні. Для епізодичних задач можна обчислювати вигоду просто як суму отриманих за епізод винагород. Для безперервних завдань буде доречніше використовувати так звану *приведену вигоду*, яка буде обчислюватись як

$$R_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k r_{t+1+k}$$

де, R_t – вигода у момент часу t,

r_i – миттєва винагорода у момент часу,

γ – коефіцієнт приведення ($0 \leq \gamma \leq 1$).

Чим ближче коефіцієнт приведення до нуля, тим більше агент орієнтується на миттєву винагороду і максимізує саме її. У протилежному випадку – якщо коефіцієнт γ наближається до одиниці – агент буде більш далекоглядним, тому що наступні винагороди набувають більшої ваги.

Насправді, для загальності, досить просто виразити суму у обчисленні підсумкової винагороди епізоду як окремий випадок приведеної нескінченної суми з коефіцієнтом приведення 1. Для цього, після термінального стану можна ввести псевдоперехід до особливого *поглинаючого стану*, який з часом не змінюється і завжди оцінюється як той, що надає винагороду нуль.

1.2. Алгоритми НП на основі моделі та безмодельні алгоритми НП

Навчання з підкріпленням відрізняється від більш вивченої проблеми навчання з вчителем декількома аспектами. Найважливіша відмінність полягає в тому, що не існує уявлення про пари цільових вхідних та вихідних

сигналів. Все, що відомо агенту – це поточний стан оточення, який він спостерігає, і винагороди за останню виконану над середовищем дію, але рішення, але не вказується , які дії були б кращими для досягнення встановленої мети. Агент повинен швидко набувати корисного досвіду про можливі стани системи, дії та винагороди, щоб діяти оптимально. Навчання з підкріпленням, у першу чергу, зосереджує свою увагу на тому, як отримати оптимальну стратегію.

Модель необхідна для імітування динаміки оточення. Тобто вона вивчає можливість переходу зі стану s_0 при виконанні дії a в наступний стан s_1. Якщо ця можливість успішно вивчена, то агент знатиме, з якою вірогідністю він зможе отримати певний стан s', якщо виконає дію a в поточному стані s. Алгоритми визначення оптимальної стратегії з урахуванням моделей оптимальної поведінки визначають, як агент повинен враховувати майбутнє у рішеннях, які він приймає, щоб вирішити, як поводитися зараз. Відмінність алгоритмів, які модель не використовують, полягає у тому, що вони спираються на метод спроб і помилок, щоб скорегувати і оновити свої знання [8].

Включення моделей і засобів планування у системи, які реалізують НП, являє собою досить нову область дослідів. Поступово стало зрозуміло, що методи НП тісно пов’язані з методами динамічного програмування, що спираються на моделі, а ті, у свою чергу, тісно пов’язані із методами планування у просторі станів. Однак алгоритми, побудовані на моделях, стають непрактичними зі зростанням простору станів і дій (квадратична асимптотика пам’яті і обчислень відносно до кількості станів і лінійна відносно до кількості можливих дій). З іншого боку, безмодельні алгоритми не потрібують великої кількості пам’яті для зберігання комбінацій стану-дії та їх оцінок.

Основним недоліком є те, що модель середовища, що відповідає дійсності, зазвичай недоступна агенту. Якщо агент хоче використовувати модель у цьому випадку, він повинен вивчати модель виключно на досвіді, що

створює кілька проблем. Найбільша проблема полягає в тому, що агент може використати упередження в моделі, в результаті чого агент добре працює щодо вивченої моделі, але поводиться неоптимально (або навіть жахливо) у реальному середовищі. Наприклад, якщо агент-гравець довго вчився і досліджував гру проти гравця певного рівня, він може побудувати модель, за якою так поводяться всі гравці і буде у незнайомих для нього випадках приймати дуже погані рішення. Навчання моделі принципово важке, тому навіть напружені зусилля — бажання витратити багато часу й обчислити — можуть не окупитися.

Алгоритми, які використовують модель, називаються методами на основі моделі, а ті, які не використовують модель, називаються безмодельними. У той час як безмодельні методи відмовляються від потенційного підвищення ефективності вибірки від використання моделі, їх, як правило, легше реалізувати та налаштувати.

1.3. Методи БНП

Існує два основних підходи до представлення та навчання агентів за допомогою безмодельного НП [9].

Першим з них є оптимізація політики (*напр.* A2C, A3C, PPO). Методи в цьому сімействі представляють політику явно як $\pi_\theta(a \mid s)$. Вони оптимізують стратегію або безпосередньо, або опосередковано, максимізуючи локальні апроксимації. Ця оптимізація майже завжди виконується відповідно до політики, а це означає, що кожне оновлення використовує лише дані, зібрані під час дії відповідно до останньої версії політики. Оптимізація політики також зазвичай включає вивчення апроксиматора для внутрішньополітичної функції значення $V_\pi(s)$, яка використовується для визначення способів оновлення політики.

Другим підходом є оптимізація значення (*напр.* DQN, C51). Методи цього сімейства вивчають апроксиматор для оптимальної функції цінності дії

Q_π*(s, a)*. Зазвичай вони використовують цільову функцію, засновану на рівнянні Беллмана. Ця оптимізація майже завжди виконується поза політикою, що означає, що кожне оновлення може використовувати дані, зібрані в будь-який момент під час навчання, незалежно від того, як агент вибирав досліджувати середовище, коли дані були отримані. Відповідна політика отримується через зв'язок між Q* та π*: дії, які виконує Q-навчаючий агент, визначаються як $a(s) = \arg\max_{\mathrm{a}} Q_\theta(s, a)$.

Основна перевага методів оптимізації політики полягає в тому, що вони принципові в тому сенсі, що ви безпосередньо оптимізуєте те, що вам потрібно. Це робить їх стабільними та надійними. Навпаки, методи Q-навчання лише опосередковано оптимізують роботу агенту, навчаючи апроксиматор для задоволення рівняння самоузгодженості. Для такого типу навчання існує багато несприятливих випадків відмови, тому воно має тенденцію бути менш стабільним [9]. Але методи Q-навчання мають перевагу, оскільки вони значно ефективніші, коли вони працюють, оскільки вони можуть повторно використовувати дані більш ефективно, ніж методи оптимізації політики.

Як не дивно, оптимізація політики та Q-навчання не є несумісними (а за деяких обставин, виявляється, еквівалентними), і існує цілий ряд алгоритмів, які живуть між двома крайнощами. Алгоритми, які живуть у цьому спектрі, здатні ретельно знайти компроміс між сильними і слабкими сторонами будь-якої сторони (*напр.* DDPG, SAC).

2 ОСОБЛИВОСТІ АЛГОРИТМІВ НП БЕЗ МОДЕЛІ

2.1. TD-методи (Time Difference)

TD-методи використовують для вирішення задачі корегування функцій цінності наявний у агенту досвід. При спостереженні нетермінального стану s_t у момент часу t, агент корегує поточні оцінки відштовхуючись від наступних спостережень (переходів станів оточення та відповідних отриманих винагород). Уточнення функції цінності стану за цим методом обраховується як [11]:

$$V' \leftarrow V(s_t) + \alpha\left(\sum_{i=0}^{T}\gamma^i r_{t+i+1} + \gamma^{T+1}V(s_{t+T+1}) - V(s_t)\right)$$

де T – порядок TD методу (записується у дужках, наприклад TD(0)),

V' - наступна ітерація функції цінності,

γ – коефіцієнт приведення ($0 \le \gamma \le 1$).

Наприклад, найпростіший TD-метод, відомий як TD(0), приймає наступну форму:

$$V' \leftarrow V(s_t) + \alpha\big(r_{t+1} + \gamma V(s_{t+1}) - V(s_t)\big).$$

Безпосередньо в момент часу *t + 1* формується цільове значення оцінки і проводиться відповідне корегування функції цінності. Після цього, поточна стратегія на основі скорегованої функції може надати рішення щодо наступної дії агенту.

Найголовнішою перевагою цієї групи методів є їх безмодельність – вони не потребують знань щодо ймовірностей переходів станів оточення і відповідних отриманих винагород. Іншою – можливість навчатися «на ходу», ітераційним чином обчислюючи і корегуючи цільову функцію винагороди і знаходячи оптимальну стратегію незалежно від довжини епізоду.

2.2. Q-Learning та Deep Q-Learning

Q-Навчання – TD-метод з розмежованою оцінкою цінності стратегій. Метою моделі є знайти найкращий курс дій з урахуванням її поточного стану. Для цього він може розробити власні правила або діяти поза політикою, яку йому дано дотримуватися. Це означає, що фактичної потреби в політиці немає, тому її називають позаполітичною (off-policy). Його найпростіша форма, однокрокове Q-навчання, наближує функцію цінності дії наступним чином:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha(r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t))$$

Deep Q-Learning, або глибинне Q-навчання – це модифікація однокрокового Q-навчання, яка використовує нейронну мережу для апроксимації Q-функції [9]. Така оптимізація є логічною для вирішення проблеми «прокляття розмірності» в ситуації великої розмірності або ж безперервності простору станів оточення. За її рахунок досягається, у певному сенсі, класифікація груп станів оточення та інтерполяція значень цінності дій. Функція цінності тренується на наявному досвіді – приклади для навчання мережі формуються за наведеним правилом апроксимації – що дозволяє ефективніше опрацьовувати безперервні і дискретні параметри стану оточення.

2.3. Методи градієнту стратегії. REINFORCE

Методи градієнту стратегії корегують стратегію напряму, без використання будь-якого різновиду функції цінності – лише спостереження агенту упродовж епізоду. Формально мета методів градієнту стратегії визначається як максимізація цільової функції наступного виду [9]:

$$J(\theta) = \mathbb{E}\left[\sum_{t=0}^{T-1} r_{t+1}\right],$$

де θ – параметри стратегії π(θ),

наприклад коефіцієнти лінійного многочлену або ваги нейронних зв’язків. Упродовж навчання параметри політики уточнюються за допомогою

градієнтного підйому по параметрах цільової функції відносно параметрів політики:

$$\theta \leftarrow \theta + \frac{\partial}{\partial \theta} J(\theta)$$

Використовуючи декілька переходів і факт безмодельності агенту при диференціюванні цільової функції градієнт цільової функції приймає наступну форму:

$$\nabla_\theta J(\theta) = \sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t|s_t) R'_t \,,$$

де R'_t – приведена вигода на відрізку $[t + 1; T]$.

За принципом накопичення досвіду такий підхід нагадує методи Монте-Карло, але зазвичай таким чином апроксимується якась з функцій цінності. Навчання моделі політики за таким правилом реалізується у методі підкріплення політики (REINFORCE у англомовній літературі). Основним недоліком такого методу є необхідність збирання траєкторії (історії, досвіду) всього епізоду для оновлення моделі політики, що може стати проблемою за умови великої довжини епізодів. Це зумовлено необхідністю знати приведену винагороду для дій і станів вже з самого початку епізоду. Але метод градієнту політики лягає у основу більшості сучасних методів групи виконавець-критик, та деяких інших, де, за рахунок заміни приведеної суми якимось іншим ваговим значенням, отримується можливість оновлення політики «на ходу» і підвищення швидкості збіжності процесу навчання до оптимальної (у переважній кількості випадків локально оптимальної – таке обмеження зумовлено можливістю методів градієнтного спуску/підйому знаходити глобальні/локальні екстремуми) стратегії.

2.4. Методи групи виконавець-критик. QAC, A2C, TDAC

Методи групи виконавець-критик – це TD-методи, що розмежовують моделі (або структури пам’яті) для явного представлення стратегії незалежно

від функції цінності. Модель, що відповідає за стратегію називається виконавцем, тому що вона використовується для вибору дії, а модель функції цінності – критиком, тому що вона критикує дії, що обирає виконавець. Після того, як виконавець обере наступну дію, критик оцінює новий стан на предмет покращення чи погіршення у порівнянні з очікуваним результатом [9].

Найбільша цінність методів цієї групи полягає у тому, що вони можуть вивчити явну стохастичну політику; тобто вони можуть дізнатися оптимальні ймовірності вибору різних дій. Ця здатність виявляється корисною в немарковських випадках. Крім того, окремий виконавець у методах виконавець-критик робить їх більш привабливими в деяких випадках як психологічні та біологічні моделі. Інколи це також може полегшити накладення обмежень, що стосуються конкретної предметної області, на набір дозволених політик.

Як вже було зазначено вище, в основу реалізації методів виконавець-критик лягає метод градієнтну стратегії. Базові алгоритми виконавець-критик засновуються на заміні приведеної вигоди якимись іншими оцінювальними значеннями, які вираховуються моделью критика, після чого обчислюється відповідний до цього значення градієнт моделі політики і вносяться відповідні зміни.

Наприклад, QAC (Q Actor-Critic) замінює приведену вигоду значенням Q-функції, а градієнт цільової функції приймає наступний вид:

$$\nabla_\theta J(\theta) = \sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t|s_t)\, Q(s_t, a_t)\,.$$

A2C (Advantage Actor-Critic) використовує так звану A-функцію (Advantage function – функція переваги), яка визначається наступним чином:

$$A(s_t, a_t) = Q(s_t, a_t) - V(s_t)\,,$$

Функція переваги надає оцінку відносної переваги дії a_t у стані s_t над іншими можливими діями у цьому стані. Відповідно, градієнт цільової функції приймає наступу форму:

$$\nabla_\theta J(\theta) = \sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t|s_t)\, A(s_t, a_t)\,.$$

TDAC (Time difference Actor-Critic) використовує як оцінювальний множник так звану різницю у часі:

$$\delta_t = r_{t+1} + \gamma V(s_{t+1}) - V(s_t).$$

Цей член використовується при уточненні функції цінності в TD-методах. А градієнт цільової функції приймає наступу форму:

$$\nabla_\theta J(\theta) = \sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t|s_t)\delta_t\,.$$

Всі ці методи відрізняються швидкістю збіжності, а також різноманітністю досліджених дій у процесі навчання.

3 ПРИСКОРЕННЯ НАВЧАННЯ АГЕНТУ БНП ЗА ДОПОМОГОЮ МОДЕЛІ

Не дивлячись на усі вище зазначені проблеми НП на основі моделі і переваги БНП, логіка походження методу НП (прототипом НП був процес навчання живої істоти) наштовхує на думку про можливість комбінації алгоритмів БНП із навчанням з моделлю оточення.

Людина може приймати рішення як стратегічно, за допомогою планування на основі внутрішньої моделі, так і швидко (або ж «інтуїтивно»). При чому, в основу стратегії навчання у стресових середовищах дуже часто лягає принцип, за яким, за допомогою попереднього стратегічного планування і дослідження моделі у навчанні, напрацьовується надійна інтуїція, яка може надавати швидкий і задовільно точний результат у реальному сценарії. Такий підхід дозволяє збільшити якість і швидкість навчання для середовища, в якому на прийняття рішення не надається достатньо для стратегічного планування часу.

В термінах НП, стратегічне планування і інтуїція відображаються на прийняття агентом рішень з урахуванням моделі оточення і без урахування (БНП) відповідно. Тоді зазначений у попередньому параграфі метод навчання для людей можна адаптувати під агента БНП. Такий підхід дозволив би використати переваги моделі у навчанні і уникнути її недоліків під час роботи агенту. Такий процес був би корисний, наприклад, для більш повноцінного навчання агента БМП, що має працювати на OEM з обмеженими ресурсами або ж у системах реального часу.

3.1. Пропозиція модифікації

В рамках даної кваліфікаційної роботи пропонується модифікація алгоритму БНП Q-Learning з використанням моделі, де обраховані за допомогою моделі оточення, яку будує агент, послідовності перетворень

станів оточення використовуються для розширення горизонту планування агенту в процесі навчання.

Якщо в класичному вигляді уточнення функції якості виконується за наступною формулою (див. Розділ 2.2):

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha(r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)),$$

то за допомогою моделі, горизонт планування Q-агенту можна розширити наступним чином (адаптуючи форму TD метода під специфіку Q-Learning; див Розділ. 2.1):

$$Q'(s_t, a_t) \leftarrow Q(s_t, a_t) +$$

$$+ \alpha \left(\sum_{i=0}^{T} \gamma^i r_{t+i+1} + \gamma^{T+1} Q \max_a Q(s_{t+T+1}, a_{t+T+1}) - Q(s_t, a_t) \right)$$

де T – глибина планування у навчанні,

Q' - наступна ітерація функції якості дії,

γ – коефіцієнт приведення ($0 \leq \gamma \leq 1$).

Для такої форми, кожна наступна після r_{t+1}винагорода отримується за допомогою виконання дії, обраної поточним агентом (з функцією якості дії поточної версії/ітерації) поза стратегією над станом, який спрогнозує його модель (визначаючи шукану винагороду а також новий стан для обраної дії). Тоді кожна винагорода r_{t+i+1} для $i > 1$ обчислюватиметься наступним чином:

$$(r_{t+i+1}, p_{t+i+1}) = M(s_{t+i}, \max_a Q[p_{t+i}, a_{t+i}]),$$

де M – функція моделі, що відображає пару поточного стану і дії на пару винагороди за дію і наступного стану,

p – спрогнозований моделлю стан (який може не відповідати реальному переходу оточення агенту.

3.2. Попередні дослідження

Звісно, запропонована логіка думки не є інноваційною. Використання моделі при навчанні безмодельного алгоритму є досить вивченою (в

масштабах кількості досліджень, проведених у сфері НП) методікою. Але використання моделі у навчання БМП в досліджених раніше методах зводиться до додаткової генерації даних для навчання класичного агенту.

Наприклад Р. Саттон ще у 1991 році [12] запропонував використання моделі у навчанні Q-агенту для генерації додаткових ітерацій прогонки алгоритму уточнення Q-функції (алгоритм Dyna-Q) і продемонстрував ефективність цього методу при використанні навченої моделі оточення для генерації додаткових прикладів навчання у сценаріях із скінченними епізодами. Пізніше, у 2016 році, група дослідників університету MIT запропонувала аналог Dyna-Q для сценаріїв з нескінченними епізодами [13].

Тим не менш, жоден з них не використовував модель для саме розширення горизонту планування агенту на навчанні і адаптації Q-навчання під TD-метод порядку більше нуля (що і дозволяє зробити використання моделі).

4 ОПИС РОЗРОБЛЕНОЇ ПЛАТФОРМИ ТЕСТУВАННЯ АЛГОРИТМІВ БНП

Для досягнення мети кваліфікаційної роботи за допомогою мови програмування Python була створена платформа, що забезпечує модуль абстракції агентів безмодельного навчання, модуль утиліт для автоматизації навчання із заданими параметрами, і додаток із консольним інтерфейсом, який надає користувачеві платформи доступ до розробленого функціоналу без додаткової необхідності написання власних чи змінення існуючих скриптів.

Мова Python є найчастіше використовуваною для програмної реалізації алгоритмів НП серед дослідників області. Її особливості (динамічна типізація та широкий набір різноманітних бібліотек, що доступні через менеджер пакетів pip) дозволяють дослідникам швидко і просто програмувати і перевіряти гіпотези стосовно нових алгоритмів або модифікацій НП.

Проте, широко розповсюджених модулів для програмування саме агентів НП не існує (Gymnasium надає тільки імплементації оточень агенту, але залишає реалізацію функціональності агенту на плечах розробника/дослідника). Тому дуже часто складається ситуація, коли дослідники хоч і створюють працюючу версію за допомогою усього декількох десятків рядків коду, проте часто повторюють один одного у етапах ініціалізації агентів, їх збереження і імплементації окремих розповсюджених компонентів. З точки зору інженерії програмного забезпечення, такий підхід є вразливим до поширених помилок та недоречним з точки зору написання коду.

Зазначенні вище пункти є додатковою мотивацію для побудови розробленого інструменту.

4.1. Абстракція агентів глибинного навчання

Для спрощення проектування, тестування і зберігання агентів БНП, була створена система абстрактних класів і реалізацій окремих функціональних класів, які часто використовуються при програмуванні агентів БНП.

Базовий клас агенту БНП Agent визначає наступний абстрактний інтерфейс:

- choose_action – метод обирання агентом дії;
- learn – метод навчання агенту;
- map_state – метод відображення наданого стану у форму внутрішнього представлення стану агента;
- map_action – метод відображення обраної агентом дії у його внутрішньому представлені на дію, яку приймає оточення бібліотеки оточень Gymnasium;
- load – завантаження агенту у серіалізованому вигляді з файлової системи за вказаним шляхом;
- save – збереження серіалізованого агенту у серіалізованому вигляді до файлової системи за вказаним шляхом.

Також були реалізовані класи пам’яті агенту (ReplayMemory) та клас глибинної Q мережі.

В межах цього ж модуля були реалізовані алгоритми Q-Learning та його запропонована модифікація – MQ-Learning (Modified Q-Learning) – які наслідують базовий клас і надають належні імплементації його абстрактного інтерфейсу, використовуючи надані додаткові класи пам’яті та глибинної Q-мережі.

Розглянемо, наприклад, реалізацію функції learn у класі агенту, що навчається за допомогою глибокої Q-мережі (DQN):

```
batch = self.recall()
if batch is None:
return

non_final_mask = torch.tensor(
tuple(
map(lambda s: s is not None, batch.next_state)),
device=self._device, dtype=torch.bool)
```

```
non_final_next_states = torch.cat(
[s for s in batch.next_state if s is not None])

state_batch = torch.cat(batch.state)
action_batch = torch.cat(batch.action)
reward_batch = torch.cat(batch.reward)
state_action_values = self.policy_net(state_batch).gather(1,
action_batch)

next_state_values=torch.zeros(BATCH_SIZE,device=self.device)
with torch.no_grad():
     next_state_values[non_final_mask] =
          self.target_net(non_final_next_states).max(1)[0]

expected_state_action_values = (next_state_values * GAMMA) +
reward_batch

criterion = nn.SmoothL1Loss()
loss = criterion(state_action_values,
expected_state_action_values.unsqueeze(1))

self.optimizer.zero_grad()
loss.backward()
torch.nn.utils.clip_grad_value_(
     self.policy_net.parameters(),
     100)
self.optimizer.step()

self._update_target_net()
```

У цьому фрагменті коду використовується функція recall, яку надає компонента MemoryAgent. Ця функція повертає вибірку вказаного розміру історії з усіх переходів, які запам’ятав агент. Такий функціонал часто є необхідним при реалізації алгоритмів БНП, так як без наявності моделі, єдиним практичним засобом ефективного навчання є багаторазове використання і засвоювання досвіду. Також, у цьому коді використовується мережа глибинного Q-навчання DQN (як декілька з полів класу агенту DQN, а саме – target_net та policy_net). Цей клас являє собою простий лінійний нейромережевий апроксиматор, сумісний с функціоналом пакету обчислень мови Python PyTorch. Подібні структури є кореневою складовою частиною більшості алгоритмів ГНП і можуть бути повторно використані у програмних реалізаціях різних поширених алгоритмів БНП.

4.2. Модуль тестування і демонстрації роботи агентів

Для створення, навчання, збереження, завантаження і демонстрації роботи агенту НП в модулі utility реалізовані наступні функції:

- learn – приймає на вхід об’єкт оточення, відповідний об’єкт агенту, та різноманітні параметри налаштування процесу навчання (див. далі), проводячи процес навчання агенту;
- display - приймає на вхід об’єкт оточення, відповідний об’єкт агенту, та кількість демонстраційних епізодів, після чого запускає демонстрацію роботи переданого агента у переданому оточенні.

Функція learn приймає наступні додаткові параметри:

- success_thereshold – верхня межа сумарної винагороди агенту, після якої епізод вважається успішно закінченим;
- success_count – кількість успішно завершених епізодів, яку має пройти агент для того, щоб процес навчання вважався успішно закінченим;
- num_episodes – обмеження на максимальну кількість епізодів, протягом якої необхідно навчати агента;
- report – логічний флаг, який визначає, чи необхідно виводити інформацію про перебіг кожного пройденого агентом епізоду у стандартний потік виводу додатку.

Обов’язковими для завдання на навчання є або пара «success_thereshold-success_count», або параметр episode_num, які обмежують тривалість навчання агенту.

Додатково, функція learn опитує клас агенту на предмет реалізації додаткових інтерфейсів (напр. MemoryAgent – агент з пам’яттю на основі класу ReplayMemory, якому необхідно в окремому порядку і явно передавати кортеж, що складається з інформації про перехід станів оточення на одному кроці, зробленому агентом) і адекватно використовує їх особливості для підтримання реалізованого споживачем модулю абстракцій алгоритму.

4.3. Додаток із консольним інтерфейсом

Для взаємодії з розробленими інструментами користувачеві інструменту надається додаток із інтерфейсом командного рядку, так само, як і попередні модулі, реалізований мовою програмування Python.

На вибір користувачеві пропонується 3 команди. Перша команда – команда agent – створює новий неспеціалізований і ненавчений агент зазначеного позиційним аргументом agent_type типу для оточення з ідентифікатором env_id. Також ця команда приймає на вхід шлях, до локації у ФС, у якій має бути збережено агенту у серіалізованому вигляді (відповідно до імплементації цього агенту).

Наприклад (рис. 4.1), для того щоб створити нового агента Deeep Q-Leraning для оточення Gymnasium «CartPole-v1» і зберегти його у поточній папці з ім’ям test_dqn_cartpole необхідно викликати модуль chamber, за допомогою інтерпретатора python, передати на першій позиції назву команди – agent – після чого вказати у такій послідовності ідентифікатор алгоритму навчання, ідентифікатор оточення Gymnasium відповідно до документації, та власне назву цільового файлу у ФС (додаток здатний самостійно визначати розширення для цього файлу відповідне до типу агенту).

```
PS F:\Study\ONU Bachelour\Diploma\LonelyKeter\Gymnasium> python.exe chamber.py agent dqn CartPole-v1 test_dqn_cartpole
Created agent of type dqn at location F:\Study\ONU Bachelour\Diploma\LonelyKeter\Gymnasium\test_dqn_cartpole.dqn
```

Рисунок 4.1 – Приклад використання команди agent і результатів її успішного виконання

Якщо необхідно створити агента для іншого оточення (наприклад «LunarLander-v2»), то достатньо буде змінити тільки другий позиційний арумент (рис. 4.2).

```
PS F:\Study\ONU Bachelour\Diploma\LonelyKeter\Gymnasium> python.exe chamber.py agent dqn LunarLander-v2 test_dqn_lunar_lander
Created agent of type dqn at location F:\Study\ONU Bachelour\Diploma\LonelyKeter\Gymnasium\test_dqn_lunar_lander.dqn
PS F:\Study\ONU Bachelour\Diploma\LonelyKeter\Gymnasium>
```

Рисунок 4.2 – Приклад створення агенту для іншого оточення

Команда display приймає на вхід 3 позиційних аргументи:

1. agent_source – задає джерело для завантаження у додаток реалізації класу агенту;
2. env_id – задає назву оточення Gymnasium, для якого слід виконати демонстрацію роботи завантаженого агенту;
3. num_episodes – задає кількість демонстраційних епізодів взаємодії агенту із середовищем, після відображення яких додаток успішно завершиться.

Наприклад (рис. 4.3), для того, щоб передивитись 10 епізодів роботи агенту dqn у середовищі «CartPole-v1», необхідно виконати модуль chamber за допомогою інтерпретатора Python, передавши як аргумент командного рядку назву команди – display – після чого шлях до попередньо створеного файлу агенту, ідентифікатор необхідного оточення Gymnasium та кількість епізодів.

Рисунок 4.3 – Приклад використання команди display і результатів її виконання

Для того ж, наприклад, щоб передивитись кілька епізодів для іншого агенту з іншого оточення, достатньо буде лише змінити перші два позиційних

аргументи (ті, що йдуть за назвою команди display). З таким прикладом можна ознайомитись на Рисунку 4.4.

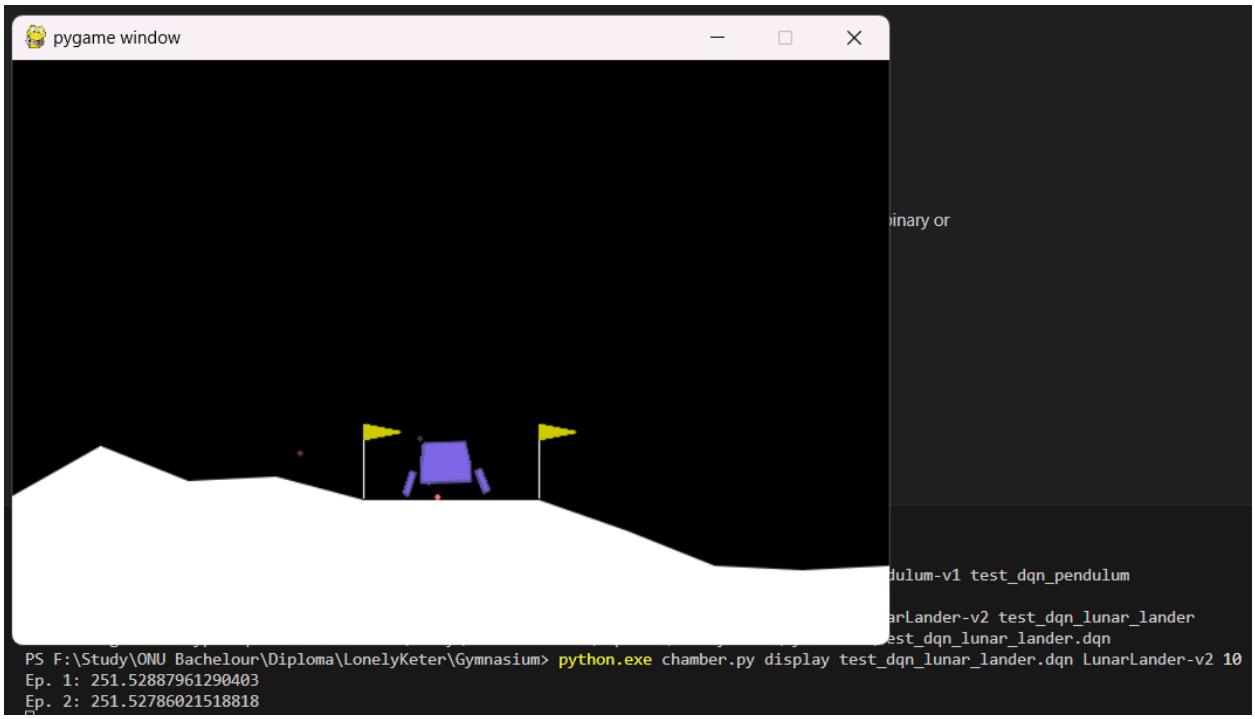

Рисунок 4.4 – Приклад використання тестувальної платформи для демонстрації роботи навченого у середовищі «LunarLander-v2» агенту DQN протягом 10 епізодів

Джерелом агенту може бути або вірний шлях до файлу, в якому зберігається серіалізований агент, підтримуваний розробленою системою, або ж спеціальний унікальний ідентифікатор. Зазвичай, значення цього ідентифікатору збігається із відповідним до цього агенту типом серіалізованого файлу. Також, ця команда приймає необов’язковий аргумент – max_steps – який обмежує максимальну кількість кроків у епізоді, який демонструється.

Аргумент оточення може приймати значення будь якої точної назви оточення агенту, що зареєстровано у корені пакету Gymnasium, наприклад класичний «CartPole-v1». За незбіжностей у типі вказаного оточення і у типі оточення, на якому здійснювалося тренування агенту, додаток не буде виконувати демонстрацію роботи агенту чи жодним чином змінювати

серіалізоване представлення, а повідомить про помилку користувачеві і завершить роботу.

Найбільшу вагу несе у собі команда learn. Вона приймає наступні аргументи та параметри (рис. 4.5-4.6):

– за аналогією з командою display, для цієї команди я обов'язковими аргументи agent_source та env_id;
– save_agent_path (у скороченій формі «-s») – опціональний параметр, який задає шлях для зберігання серіалізованої версії навченого агенту після тестування;
– report_progress (у скороченій формі -r) – бульовий параметр, який вказує, чи необхідно виводити у потік стандартного виводу відомості про перебіг кожного окремого епізоду;
– по одному з аргументів на вхід для кожного відповідного аргументу методу learn модулю utility (див. Розділ 4.2).

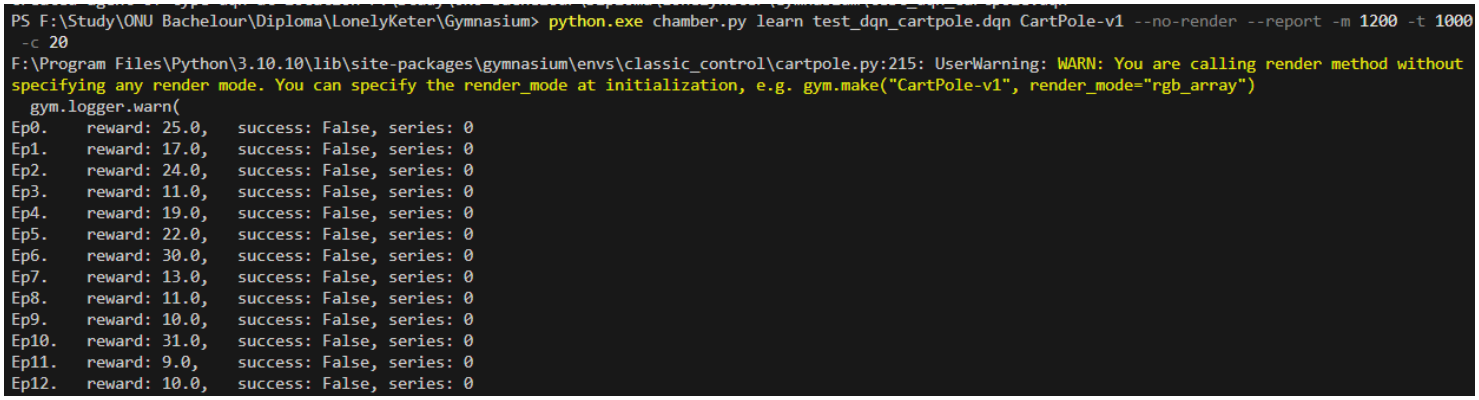

Рисунок 4.5 – Приклад використання команди learn

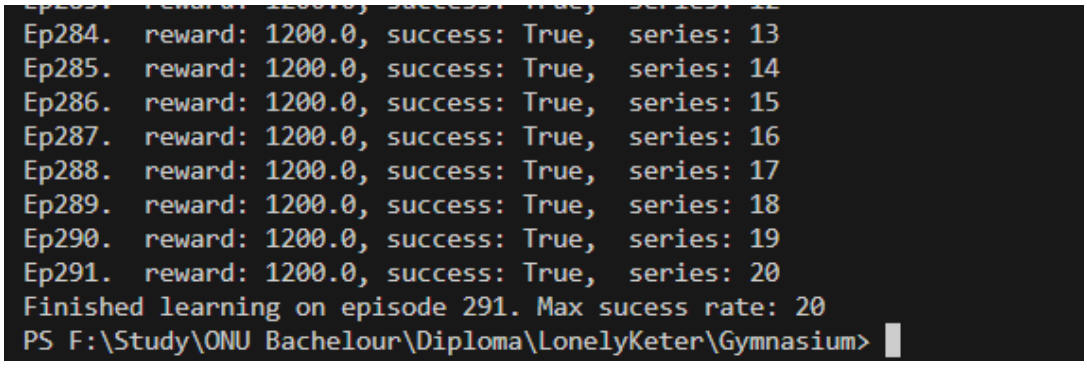

Рисунок 4.6 – Приклад кінцевого результату роботи команди learn

5 ОГЛЯД РЕЗУЛЬТАТІВ АПРОБАЦІЇ ЗАПРОПОНОВАНОЇ МОДИФІКАЦІЇ

Для оцінки ефективності навчання агенту НП за допомогою запропонованої модифікації алгоритму в рамках побудованої платформи були реалізовані алгоритми DQN та, власне, його запропонована модифікація. В імплементації були додатково використані класи глибинної мережі DeepQNetwork та агенту з пам’яттю MemoryAgent.

Апробація проводилася на різноманітних оточеннях, наданих бібліотекою Gymnasium, для кожного з цих алгоритмів. Через принципову одноманітність отриманих результатів, далі буде детально розглянуто лише процес апробації на оточенні «CartPole-v1». Із таблицею повних результатів випробування можна ознайомитись у Додатку А.

5.1. Порівняння результатів навчання у середовищі CartPole-v1

Середовище «CartPole-v1» є імплементацією класичного середовища для навчання агенту НП, запропонованого Р. Саттоном [2]. В ньому, ціллю агента є пересування платформи уздовж горизонтальної осі координат для балансування жердини, яка закріплена на цій платформі шарніром з одним ступенем свободи таким чином, щоби рухатися у відповідній до осі зміщення платформи вертикальній площині. Чим довше агент зможе балансувати жердину для запобігання її відхиленню від суворо вертикального положення на 12 градусів (не виходячи при цьому за певні встановлені обмеження по зміщенню уздовж осі пересування платформи), тим більшу винагороду він отримає.

Для навчання DQN-агенту на цьому середовищі з параметрами, зазначеними на Рисунку 4.3, в середньому було необхідно від 290 до 1200 епізодів (стохастична ε-жадібна стратегія могла як швидко знайти

оптимальний засіб поводження, так і довго шукати рішення у заздалегідь невірному просторі), що є досить непоганим результатом.

Алгоритм Dyna-Q з попередньо навченою моделлю у такій ситуації надавав би результат ще швидше і надійніше (відповідно до результатів, отриманих самим Р. Саттоном [12]). Але, на жаль, запропонована модифікація (з ненавченою моделлю) не змогла значним чином навчитися виконанню поставленої задачі навіть за проміжок навчання, більший у двічі за максимальний для класичного Q-навчання. Подібна картина спостерігалася для всіх протестованих середовищ.

5.2. Пропозиції щодо покращення запропонованої модифікації

Перша недосконалість запропонованої модифікації полягає у необхідності навчати модель паралельно із агентом. Навчання моделі – складний процес, який зазвичай триває набагато довше, аніж навчання агенту БНП. Навіть якщо в теорії використання моделі і допомагає у навчанні, розширюючи горизонт планування агенту, на практиці, для простих задач із простою динамікою оточуючої системи, побудова «безмодельної інтуїції» агенту відбувається набагато ефективніше і швидше. Більше того, внеполітничний алгоритм Q-Learning має тенденцію до високої упередженості щодо власного досвіду. Якщо досвід сформований на неправильній моделі поведінки системи (яку надає ненавчена модель оточення), то з часом навчання зменшується швидкість збігання стратегії агенту, що навчається, до оптимальної.

Виправити це можна, по-перше, за допомогою заздалегідь навченої моделі. Проте, така модель не завжди є доступною – таке оточення є окремим випадком, що потребує додаткового дослідження і порівняння із існуючими засобами прискорення процесу навчання безмодельного агенту.

Другий спосіб, у який потенційно можна покращити отриманий результат – це введеня додаткової метрики для визначення, чи придатна

модель на поточний момент навчання до передбачення переходів стану оточення із заданою глибиною прогнозування. Такий метод також потребує додаткових досліджень із визначенням суворих або еврістичних методів оцінки продуктивності моделі оточення агентів.

По-третє, необхідно додатково провести дослідження поведінки побудованої модифікації у більш складних системах оточення агенту, в яких базовий алгоритм не надає оптимального результату. Таку систему можна побудувати, працюючи у тандемі зі спеціалістами предметної області, з якої обрана складна для базового алгоритму задача.

По-четверте, можна розширити запропоновану модифікацію на інші алгоритми БНП і додатково дослідити їх ефективність.

Слід зазначити, що для дослідження будь-якого запропонованого варіанту покращення можна буде повторно використати побудовану в межах цієї кваліфікаційної роботи тестувальну платформу.

ВИСНОВКИ

В процесі виконання даної кваліфікаційної роботи було розглянуто основи НП, створено порівняльну характеристику алгоритмів БНП і модельного НП, детально розглянуто декілька різних алгоритмів БНП і запропоновано модифікацію для прискорення навчання агенту БНП за допомогою моделі. Додатково були розглянуті попередні дослідження на тему запропонованої модифікації та обґрунтовано новизну запропонованої модифікації.

Для апробації алгоритмів, що досліджувались, була побудована тестова платформа, що складається з модулю загальних абстракцій для побудови імплементацій алгоритмів БНП, модуль утиліт для створення і навчання агентів, а також тестування їх роботи.

В процесі апробації було виявлено неефективність запропонованої модифікації у оточеннях з простою динамікою, був проведений аналіз можливих причин невдачі, запропоновані потенційні рішення та напрямки подальшого дослідження.

Результатом цієї роботи є готова тестувальна платформа, що може бути використана у майбутніх випробуваннях і дослідженнях у зазначених напрямках, а також результати експериментального дослідження запропонованої модифікації для найпростішого випадку її імплементації та використання.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. L.P. Kaelbling, Reinforcement Learning: A Survey / L.P. Kaelbling, M.L. Littman, A.W. Moore // Journal of Artificial Intelligence Research. vol. 4, issue 1 – California, USA: December 15, 1996. Association for the Advancement of Artificial Intelligence. – 49p.
2. R.S. Sutton, Reinforcement Learning: An Introduction (Second Edition) / Richard S. Sutton, Andrew G. Barto – Cambridge: MIT Press, 2018. – 548p.
3. Gerald Tesauro, Temporal Difference Learning and TD-Gammon // Communications of the ACM, vol. 48 – March 1, 1995. New York, USA: Association for computing machinery. – 15p.
4. Yan Duan, Fast Reinforcement Learning via Slow Reinforcement Learning / Yan Duan, John Schulman, Xi Chen and oth. // Conference paper for ICLR 2017 – March 1, 1995. California: OpenAI Department of Electrical Engineering and Computer Science. – 14p.
5. Peter Henderson, Deep reinforcement learning that matters / Peter Henderson, Riashat Islam, Philip Bachman and oth. // Thirthy-Second AAAI Conference On Artificial Intelligence (AAAI) – September 19, 2017. Monreal, USA: Association for the Advancement of Artificial Intelligence. – 8p.
6. Lazic Nevena, Data center cooling using model-predictive control [Electronic book] / Nevena Lazic, Tyler Lu, Craig Boutilier, Moonkyung Ryu // Proceedings of the Thirty-second Conference on Neural Information Processing Systems (NeurIPS-18). December 2-8, 2018. Montreal, USA. – pp. 3818-3827. – Access Mode: https://storage.googleapis.com/pub-tools-public-publication-data/pdf/bb67802995f7af4c6ba948ede1acfc8756be7134.pdf
7. Yuxi Li, Reinforcement Learning in Practice: Opportunities and Challenges – February 23, 2022. New York: Cornell University. – 56p.
8. Phillip Swazinna, Comparing Model-free and Model-based Algorithms for Offline Reinforcement Learning / Phillip Swazinna, Steffen Udluft, Daniel Hein, Thomas Runkler // 6th IFAC Conference on Intelligent Control and

Automation Sciences ICONS 2022 – 13-15 July, 2022. Cluj-Napoca, Romania. – 8p.

9. Vincent François-Lavet, An Introduction to Deep Reinforcement Learning / Vincent François-Lavet, Peter Henderson, Riashat Islam and oth. // Foundations and Trends in Machine Learning: vol. 11, no. 3-4 – March 11, 2019. Boston, USA: Now Publishers Inc. – 140p.
10. J. N. Tsitsiklis, An Analysis of Temporal-Difference Learning with Function Approximation / J. N. Tsitsiklis, B.V. Roy // Ieee Transactions On Automatic Control, vol. 42, no. 5 – May, 1997. Cambridge, USA: MIT Press. – 17p.
11. R.S. Sutton, Learning to Predict by the Methods of Temporal Differences // Machine Learning, vol. 3 – August 1988. Boston, USA: Kluwer Academic Publishers. – pp. 9-44
12. R.S. Sutton, Dyna, an integrated architecture for learning, planning, and reacting // ACM SIGART Bulletin, vol 2, issue 4 – July 1, 1999. New York, USA: Association for Computing Machinery. – pp 160–163
13. Shixiang Gu, Continuous deep Q-learning with model-based acceleration / Shixiang Gu, Timothy Lillicrap, Ilya Sutskever, Sergey Levine [Electronic publication] // ICML'16: Proceedings of the 33rd International Conference on International Conference on Machine Learning, vol. 48 – Jun 19, 2016. USA: JMLR.org. – pp. 2829–2838. – Access mode: https://arxiv.org/abs/1603.00748

ДОДАТОК А

Результати тестування базового алгоритму та його запропонованої модифікації

Таблиця А.1 – Результати тестування базового алгоритму та запропонованої модифікації за допомогою побудованої тестувальної платформи

ID оточення Gymnasium	Порогове значення сумарної винагороди	Пояснення цілі навчання агенту у середовищі	Обмеження по кількості кроків на епізод	Кількість епізодів тренування базового алгоритму	Кількість епізодів тренування модифікованого алгоритму
CartPole-v1	1200	Балансування жердини на візку	1200	290-1200	Занадто велика
MountainCar-v0	-150	Розгойдування машини у прірві для виїзджання з неї	200	900-1900	Занадто велика
Acrobot-v0	-65	Розгойдування дволанкового маятника для перетину кінцем його вільної ланки заданої висоти	100	1000 - 1600	Занадто велика
LunarLander-v2	200	Керування модулем шатлу для здійснення посадки в умовах горизонтального вітру	500	500-900	Занадто велика

Digitally signed by Олександр Антоненко
Date: 2023.06.18 15:23:18 +03'00'