НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО»
ІНСТИТУТ ПРИКЛАДНОГО СИСТЕМНОГО АНАЛІЗУ
КАФЕДРА МАТЕМАТИЧНИХ МЕТОДІВ СИСТЕМНОГО АНАЛІЗУ

На правах рукопису
УДК 004.853

До захисту допущено
В. о. завідувача кафедри ММСА
О.Л.Тимощук
«___» _____ 2019 р.

Магістерська дисертація
на здобуття ступеня магістра за спеціальністю 122 Комп'ютерні науки
на тему: «Методи навчання з підкріпленням в динамічних іграх»

Виконав:
студент ІІ курсу, групи КА-83 мп
Очкусь Наум Ярославович _____

Керівник: доцент кафедри ММСА,
к.ф-м.н., с.н.с. Ігнатенко О.П. _____

Рецензент: старший науковий
співробітник інституту проблем
математичних машин та систем
к.ф-м.н. Новицький Д.В. _____

Засвідчую, що у цій магістерській дисертації
немає запозичень з праць інших авторів
без відповідних посилань
Студент _____

Київ
2019

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО»
ІНСТИТУТ ПРИКЛАДНОГО СИСТЕМНОГО АНАЛІЗУ
КАФЕДРА МАТЕМАТИЧНИХ МЕТОДІВ СИСТЕМНОГО АНАЛІЗУ

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

ЗАТВЕРДЖУЮ
В. о. завідувача кафедри ММСА
О. Л. Тимощук
«___» _____ 2019 р.

ЗАВДАННЯ
на магістерську дисертацію студенту Очкусю Наум Ярославовичу

1. Тема дисертації:«Методи навчання з підкріпленням в динамічних іграх», науковий керівник дисертації Ігнатенко Олексій Петрович, доцент ф-м.н., затверджені наказом по університету від «08» листопада 2019 р. № 3862-с

2. Термін подання студентом дисертації: 13 грудня 2019

3. Об'єкт дослідження: ігри переслідування

4. Предмет дослідження: методи та алгоритми навчання з підкріпленням

5. Перелік завдань, які потрібно розробити:

1) провести аналіз методів навчання з підкріпленням для вирішення прикладних задач;
2) вибрати методи навчання з підкріпленням;
3) зробити алгоритм, що вирішує гру переслідування;
4) виконати програмну реалізацію системи пошуку оптимальних електричних двигунів;
5) виконати тестування програмної частини системи

6. Орієнтовний перелік графічного (ілюстративного) матеріалу:
1). Модель гри
2). Обмеження на систему
3). Критерій оптимальності
4). Метод TD навчання. Схема алгоритму
5). Метод DQN. Схема алгоритму
6). Метод Actor-Critic. Схема алгоритму
7). Метод Deep Deterministic Policy Gradient. Схема алгоритму
8). Інтерфейс програмного продукту
9). Результати проектування

7. Орієнтовний перелік публікацій:

(1) Ігнатенко О.П., Очкусь Н.Я. Методи машинного навчання у диференційних іграх переслідування.//Журнал “Проблеми програмування”.

8. Дата видачі завдання: _____

Календарний план

№ з/п	Назва етапів виконання магістерської дисертації	Термін виконання етапів магістерської дисертації	Примітка
1	Пошук матерілів, що пітверджують актуальнісьть та новизну теми	05.09.2019 - 11.09.2019	
2	Пошук середовищ для дослідження ігор переслідування	12.09.2019 - 21.09.2019	
3	Дослідження і реалізація коду утиліт допоміжних утиліт для навчання нейронної мережі	22.09.2019 - 05.10.2019	
4	Аніліз та дослідження існуючих архітектур навчання з підкріпленням.	06.10.2019 - 11.10.2019	
5	Реалізований код архітектур DQN та DDPG	12.10.2019 - 16.10.2019	
6	Вдосконалення точності мереж	17.10.2019 - 22-10.2019	
7	Проведення аналізу швидкості навчання та точності алгоритму	23.10.2019 - 01.11.2019	
8	Реалізція базової версії додатку	02.11.2019 - 6.11.2019	
9	Аналіз методів оптимізації алгоритму	7.11.2019 - 9.11.2019	
10	Реалізація оптимізаторів	10.11.2019 - 14.11.2019	

11	Реалізція базової версії додатку	15.11.2019 - 21.11.2019	
12	Провкдення вдосконалення додатку	22.11.2019 - 25.11.2019	

Студент Н.Я. Очкусь

Науковий керівник дисертації О.П. Ігнатенко

РЕФЕРАТ

Магістерська дисертація: 90с., 4ч., 23 табл., 14 рис., 12 джерел.

НАВЧАННЯ З ПІДКРІПЛЕННЯМ, ДИФЕРЕНЦІЙНА ГРА, ГРА ПЕРЕСЛІДУВАННЯ, Q-LEARNING, POLICY GRADIENT,ACTOR-CRITIC метод, DDPG.

Об’єкт дослідження - навчання з підкріпленням, диференційні ігри такі як ігри переслідування.

Мета роботи – даказати доцільність використання навчання з підкріпленням для розв’язання диференційних ігор.

Методи дослідження – моделювання різних форм ігор переслідування наприклад в яких є 1 переслідувач і 1 втікач та коли є один переслідувач та декілька втікачів, вирішення цих задач за допомогою теоретичних методів та методів навчання з підкріпленням.

На основі зроблених досліджень були побудовані графіки і таблиці для порівняння алгоритмів та аналізу тренування алгоритму навчання з підкріпленням.

Запропоновані автором методи можуть бути застосовані длч моделювання та вирішення описаних задач ігрової взаємодії.

ABSTRACT

Masters' thesis: 90 p., 4 p., 23 tables., 14 drawing., 12 sources.

REINFORCEMNT LEARNING, DIFFERENTIAL GAME, PURSUIT-EVASION GAME, Q-LEARNING, POLICY GRADIENT, ACTOR-CRITIC method, DDPG.

Object of study - reinforcement training, differential games such as pursuit games.

The purpose of the work is to prove the feasibility of using reinforcement learning to solve differential games.

Research Methods - Modeling various forms of pursuit games such as 1 pursuer and 1 fugitive and one persecutor and multiple fugitives, solving these problems using theoretical and reinforcement training methods.

Based on the research, graphs and tables were constructed to compare the algorithms and to analyze the training of the reinforcement learning algorithm.

The methods proposed by the author can be applied to simulate and solve the described problems of game interaction.

Зміст

ПЕРЕЛІК СКОРОЧЕНЬ

ШІ – штучний інтелект

AC – Actor-Critic

DDPG – Deep Deterministic Policy Gradient

DQN – Deep Q-Network

PG – Policy Gradient

RL – навчання з підкріпленням

TD – Temporal-Difference

ВСТУП

Ця робота досліджує доцільність використання навчання з підкріпленням для розв'язання диференційних ігор. В цій роботі досліджуються різні форми гри переслідування.

Задача переслідування давно використовується в багатьох сферах науки таких як поведінкова біологія, нейроетнологія і особливо в теорії ігор, де кілька класів задач були вивчені завдяки їхньому потенціалу. Ця задача широко застосовується в економіці, військових областях таких як повітряній та морській техніці, в ракетній техніці, робототехніці, відеоіграх, комунікаціях та мультиагентних системах.

Чим більше гравців тим складніше стає вирішити диференційне рівняння, тому є труднощі в моделюванні взаємодії з невизгаченим та невідомим середовищем і тому вирішення задачі стандартними методами можу бути занадто складним і тому для вирішення задачі використовують навчання з підкріпленням, що є одним з розділів навчання з підкріпленям. Тоді задача вирішується як Марківська гра і агент вивчає найкращу дію в кожен момент часу та пристосовується до невідомого середовища. Навчання вимагає певної кількості часу і це є важливою характеристикою для алгоритму, якщо він занадто великий, приходиться використовувати інші алгоритми.

Навчання з підкріпленням(RL) є одним з розділів машинного навчання(МН), що зараз стрімко розвивається і має багато сфер застосування, наприклад штучний інтелект і системи контролю. Навчання з підкріпленням зацікавило багато дослідників тим, що агент навчається і діє в невідомому середовищі, його мета знаючи інформацію дану йому з середовища вибрати

дію, що дасть кращу винагороду. RL відрізняється від інших сфер МН, тим що агент не знає, до чого призведе його дія і якйи буде наступний стан його середовища та яка буде отримана винагорода. Цю інформацію він дізнається після взаємодії з середовищем, тому RL не потребує підготовленого датасету.

РОЗДІЛ 1 АНАЛІЗ

1.1 Огляд

Навчання з підкріпленням(RL) прийнято відносити до методів машинного навчання, хоча останнім часом воно активно виділяється в особливий напрямок. RL відрізняється підходом до формулювання задачі навчання. На відмінну від ‘методів навчання з вчителем або без вчителя RL агент навчається в невідомому середовищі, що обумовлює важливі практичні застосування в різних галузях.

RL агент не ‘знає’ яка дія ‘вірна’. Замість цього він намагається дослідити різні можливі дії з тим, щоб вибрати найкращі, ті, що дають найбільшу ‘винагороду’. Під винагородою розуміють певну визначену числову функцію, яка відображає успішність дії агента.

Загальна структура моделі RL агента зображена на рис. 1.1.

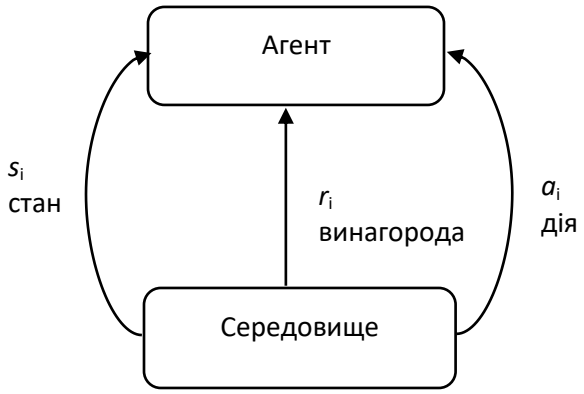


Рисунок 1.1 - Агент і середавище

Цикл взаємодії агента і середовища виглядає як послідовність

$$h_t = a_1 s_1 z_1 a_2 s_2 z_2 \dots a_t s_t z_t, \quad (1.1)$$

Можна відзначити ключові особливості навчання з підкріпленням:

- знімається проблема отримання якісних даних, оскільки дані агент отримує в результаті своїх дії(особливості роботи змінюється);
- можливо не враховувати дії іншого агента, оскільки невизначеність, оскільки невизначеність, оскільки невизначеність пов’язана з його врахована в моделі середовища.

В даній роботі RL застосовується до ігор переслідування-втечі з одним переслідувачем та різною кількістю втікачів. Диференційна гра це модель опису конфліктної ситуації, при якій динаміка гравців описується диференційним рівнянням, а керування(стратегії) є, взагалі кажучи, виміряними функціями. Іншими словами диференційна гра – це гра, що має неприривний простір дій і стану.

В рамках даної публікації будемо розглядати постановку коли гравці описуються точками в просторі R^n, а динаміка руху має вигляд:

$$\dot{x} = u, \|u\| \leq a, a > 1, \quad (1.2)$$

$$\dot{y} = v, \|v\| \leq 1, \quad (1.3)$$

Термінальна множина $M = \{z \in R^n | x = y\}$, тобто гра закінчується коли гравець x (переслідувач) дожене гравця y(втікач). При цьому переслідувач намагається мінімізувати час, задача втікача – протилежна.

Гра переслідування(1) відома під назвою може бути розв’язана різними способами в залежності від переслідувача. Відомими методами є паралельне переслідування [Чикрій], погонна крива [Кросовський], метод пропорційної навігації [Понтрегін]. В даній роботі до задачі(1) застосовуються методи навчання з підкріпленням та виконується їх порівняльний аналіз.

1.2 Огляд літератури

Було розглянута математичне вирішення нелінійної задачі переслідування на плоскості[1] та приклади реалізації підходів на дискретному полі, моделювання використовуючи value та policy iteration[5] та нейронної мережі [2].

В роботі[4] проведено порівняння п’яти різних алгоритмів машинного навчання у диференційній грі переслідування-втечі. Були проаналізовані ігри переслідування для алгоритмів:

- випадкова стратегія;
- теоретичний (оптимальний) алгоритм;
- генетичне прогнозування;
- К-середнє;
- навчання з підкріпленням.

Найкращі результати показав метод К-середніх.

В роботі[7] були порівняні алгритми для проблеми рандеву, що є дуже похожа на задачу переслідування. Проведене порівння алгоритмів нейронної мережі(NN) , мережі радільно-базисних функцій(RBF), мережа з шарами

гістограми(HistNet). В цій роботі було показані покращення від використання TRPO. Найкращі результати показала звичайна нейронна мережа.

В роботі[3] досліджуються та порівнюються алгоритми Fuzzy Actor-Critic Learning Automaton (FACLA) і Q-Learning Fuzzy Inference System(QLFIS), Residual Gradient Fuzzy Actor-Critic Learning (RGFACL), для QLFIS був використаний Fuzzy Logic Control (FLC) на основі PSO, що пришвидшує навчання мережі. Результати показали, що FACLA має меший час навчання і найменший час обчислення.

1.3 Постановка задачі

В просторі R^2 будемо розглядати два керованих об'єкта: переслідувач центр мас якого описується змінною $\bar{x}(t)$ і втікач $\bar{y}(t)$. Вважаємо, що гравці матеріальні точки. Динаміка гри задається динамікою рівнянь:

$$\begin{bmatrix} \dot{x}(t) = u(t), & \|\dot{u}\| \le a, & a > 1 \\ \dot{y}(t) = v(t), & \|\dot{u}\| \le 1 & \\ \dot{z}(t) = \dot{u(t)} - v(t) & & \\ z(t) = x(t) - y(t) & & \end{bmatrix}, \quad (1.4)$$

Переслідувач вибирає своє керування u(t) на основі інформації про стан системи:

- За умов позиційної інформації u(t) = u(z(t));
- За умов повної інформації u(t) = u(z(t), v(t)).

Метою даної роботи є застосування і порівняння методів навчання до розв’язання задач переслідування.

Задачі:

- Порівняння різних підходів теоретично відомим розв’язком;
- Оцінка швидкості і точності збіжностей;
- Висновки щодо ефективності.

1.4 Обмеження та сфера застосування навчання з підкріпленням

Більшість методів навчання для підкріплення, які ми розглядаємо в цій книзі, структуровані навколо оцінювання функцій вартості, але це не зовсім необхідно для того, щоб вирішувати проблеми навчання підкріплення. Наприклад, такі методи, як генетичні алгоритми, генетичне програмування, імітаційний відпал та інші методи оптимізації, використовувалися для підходу до проблем навчання підкріпленню, які ніколи не були привабливими для ціннісних функцій. Ці методи оцінюють поведінку багатьох людей, що не навчаються, кожен з яких використовує різні політики для взаємодії зі своїм середовищем, і обирає ті, які здатні отримати найбільшу винагороду. Ми називаємо ці еволюційні методи, оскільки їхня робота аналогічна тому, як біологічна еволюція виробляє організми з кваліфікованим поведінкою навіть тоді, коли вони не навчаються протягом свого індивідуального життя. Якщо простір політики досить невеликий або може бути структурований так, щоб хороша політика була загальною або легко знайти - або якщо для пошуку доступно багато часу, то еволюційні методи можуть бути ефективними. Крім

того, еволюційні методи мають переваги над проблемами, в яких навчальний агент не може точно визначити стан свого середовища.

Наша увага зосереджена на методах навчання підкріпленню, які включають навчання під час взаємодії з навколишнім середовищем, які не роблять еволюційні методи (якщо вони не розвивають алгоритми навчання, як у деяких досліджених підходах). Ми переконані, що методи, які можуть скористатися деталями індивідуальних поведінкових взаємодій, у багатьох випадках можуть бути набагато ефективнішими, ніж еволюційні методи. Еволюційні методи ігнорують більшу частину корисної структури проблеми навчання підкріпленню: вони не використовують той факт, що політика, яку вони шукають, є функцією від держав до дій; вони не помічають тих станів, через які людина проходить протягом свого життя, або яких дій вона вибирає. У деяких випадках ця інформація може вводити в оману (наприклад, коли держави неправильно сприймаються), але частіше вона повинна сприяти більш ефективному пошуку. Хоча еволюція та навчання мають багато особливостей і, природно, працюють разом, ми не вважаємо еволюційні методи самі по собі особливо придатними для підкріплення проблем навчання. Для простоти, в цій книзі, коли ми використовуємо термін «метод підкріплення», ми не включаємо еволюційні методи.

Проте ми включаємо деякі методи, які, як і еволюційні методи, не апелюють до ціннісних функцій. Ці методи шукають у просторах політик, визначених набором числових параметрів. Вони оцінюють напрямки, які параметри повинні бути скориговані для того, щоб найбільш швидко покращити ефективність політики. Однак, на відміну від еволюційних методів, вони виробляють ці оцінки, тоді як агент взаємодіє зі своїм середовищем і тому може скористатися деталями індивідуальних поведінкових взаємодій. Такі методи, які

називаються методами градієнта політики, виявилися корисними в багатьох проблемах, і деякі з найпростіших методів навчання підкріплення підпадають під цю категорію. Насправді, деякі з цих методів використовують переваги оціночних функцій для поліпшення їх градієнтних оцінок. Загалом, різниця між методами градієнта політики та іншими методами, які ми включаємо як методи навчання підкріплення, не є чітко визначеною.

Підключення підкріплення до методів оптимізації заслуговує додаткових коментарів, оскільки це джерело загального непорозуміння. Коли ми говоримо, що мета навчального агента з підкріплення полягає в тому, щоб максимізувати чисельний сигнал винагороди, ми, звичайно, не наполягаємо на тому, щоб агент дійсно досягав мети максимальної винагороди. Спроба максимізувати кількість не означає, що ця кількість завжди збільшується. Справа в тому, що агент з навчання підкріпленню завжди намагається збільшити суму винагороди, яку він отримує. Багато факторів можуть перешкодити їй досягти максимуму, навіть якщо він існує. Іншими словами, оптимізація - це не одна оптимальність.

1.5 Динамічні ігри

1.5.1 Динамічні ігри

Теорія ігор - це математичне вивчення взаємодії незалежних незацікавлених агентів. Він застосовувався до таких дисциплін як економіка , політологія, біологія, психологія, лінгвістика та інформатика.

Теорія ігор аналізує прийняття рішень економічними суб'єктами, яких називають, за традицією, гравцями, в ситуаціях, коли на результати цих рішень впливають дії, що вживаються іншими економічними суб'єктами. Такі ситуації

прийнято називати іграми.

У свою чергу, гравець - це просто термін, який зручний для проведення аналогії досліджуваної ситуації з салонної грою з чітко описаними правилами. Кожен гравець має певну свободу вибору дій. Своїми діями гравець впливає не тільки на свій результат, а й на результати всіх інших. результат оцінюється заданої для кожного гравця функцією виграшу. Вважається, що мета гравця - максимізувати свій виграш.

Гра - математична модель конфліктної ситуації.

Хід в грі - вибір і здійснення гравцем одного з передбачених правилами гри дій.

Стратегія - послідовність всіх ходів до закінчення гри.

Кожен агент має свій опис того, який стан середовища. Домінуючим підходом до моделювання інтересів агента є теорія корисності. Цей теоретичний підхід має на меті визначити ступінь переваги агента у наборі доступних альтернатив. Теорія також має на меті зрозуміти, як змінюються ці вподобання, коли агент стикається з невизначеністю щодо того, яку альтернативу він отримає.

Агенти завжди матимуть корисні функції, очікувані значення яких хочуть досягти максимально. Це говорить про те, що оптимально діяти в невизначеному оточенні концептуально прямо вперед - принаймні до тих пір, поки результати та їх вірогідність відомі агенту і можуть бути коротко представлені. Агентам просто потрібно вибрати спосіб дії, який максимізує очікувану корисність. Однак все може ускладнитися, коли світ містить два або більше агента, що збільшує кількість утиліт, дії яких можуть впливати на комунальні послуги.

Нормальна форма, є найбільш відомим представленням стратегічних взаємодій в теорії ігор. Гра, написана таким чином, являє собою уявлення про корисність кожного гравця для кожного стану у середовищі, в особливому випадку.

1.5.2 Класифікація ігор

Залежно від числа стратегій:

- кінцеві, якщо у гравця є кінцеве кількість стратегій;
- нескінченні (в іншому випадку).

За кількістю гравців:

- парні (два гравці);
- множинні (більше двох гравців).

Залежно від взаємовідносин гравців:

- кооперативні, якщо в грі заздалегідь визначені коаліції;
- коаліційні, якщо гравці можуть вступати в угоди;
- безколізаційні, якщо гравцям не можна вступати в угоди.

В іграх з нульовою сумою одні гравці виграють за рахунок інших, тобто сумарний виграш всіх гравців дорівнює нулю. Парні гри з нульовою сумою називаються антагоністичними. Кінцеві антагоністичні гри називаються матричними іграми.

1.5.3 Статичні і динамічні ігри з повною інформацією

Під статичної розуміють таку гру, в якій всі її учасники приймають рішення, не знаючи, які саме рішення приймають інші.

Під іграми з повною інформацією розуміють гри, в яких кожен з гравців точно знає характеристики інших гравців.

Динамічної називається гра, в якій кожен гравець може зробити кілька ходів, і принаймні один з гравців, роблячи хід, знає, який хід зробив інший гравець (можливо, він сам). У динамічних іграх розрізняють повну і досконалу інформацію. Якщо всі гравці мають загальну інформацію про правила гри і функціях виграшу, то інформацію вважають повною. Це поняття в рівній мірі відноситься як до статичних, так і до динамічним ігор.

Поняття досконалої інформації відноситься тільки до динамічним ігор, в яких гравці роблять ходи послідовно в різні моменти часу. Кажуть, що динамічна гра має досконалої інформацією, якщо все зроблені ходи відразу ж стають відомі всім гравцям.

Іноді динамічну гру зручно представити у вигляді дерева. Таке уявлення називається розгорнутою формою гри. Вона повинна містити:

- безліч вершин дерева гри, в тому числі одну початкову вершину;
- для кожної вершини, крім початкової - єдину вершину, яка безпосередньо їй передує; при цьому ланцюг попередніх вершин, побудована з будь-якої вершини, повинна закінчуватися в початковій вершині (це передбачає відсутність циклів);
- безліч гравців;

- для кожної вершини, крім кінцевих - єдиного гравця, якому належить хід в даній вершині;
- для кожної кінцевої вершини - вектор виграшів всіх Іроки;
- (якщо в грі є випадкові ходи «природи», то слід задати також розподіл ймовірностей на множині всіх можливих ходів «Природи»).

1.6 Висновки до розділу

В цьому розділі було розглянуте визначення навчання з підкріпленням, його ключові особливості, чому і коли його треба використовувати. Також була розглянута ігра переслідування, відомі способи розв’язання такі як паралельне переслідування при повній інформації та погонна крива при позиційній інформації.

При розгляді літератури було вияснено, що уже були спроби розв’язати задачу за допомогою навчання з підкріпленням та машинного навчання, але остались, ще багато різних алгоритмів, ефективність яких не була перевірена. Також результати не були порівняні з теоретичними методами порівняння і не було описано процес навчання мереж.

Також ми ознайомились з основною теорією про динамічні ігри.

РОЗДІЛ 2 МЕТОДИ НАВЧАННЯ З ПІДКРІПЛЕННЯМ

2.1 Введення в машинне навчання

Машинне навчання - систематичне навчання алгоритмів і систем, в результаті якого їх знання або якість роботи зростають у міру накопичення досвіду.

Існує два типи машинного навчання: навчання по прецедентах (індуктивне навчання), засноване на емпіричних даних, і дедуктивний навчання, що припускає формалізацію знань і формування бази знань. Дедуктивное навчання прийнято відносити до області експертних систем, тому в теорії і практиці машинного навчання фактично розглядається навчання по прецедентах.

Розділ машинного навчання виник в результаті поділу науки про нейронних мережах в рамках науки про штучний інтелект на методи навчання мереж і види топологій архітектури мереж, увібравши в себе деякі інші області, такі як методи математичної статистики і теорію дискретного аналізу. Цим обумовлена специфіка розглянутих способів навчання в рамках дисципліни:

- Навчання з учителем - для кожного прецеденту існує пара «ситуація, рішення»;
- навчання без вчителя - система групує об'єкти в кластери і знижує розмірність вхідної інформації, використовуючи дані про попарном схожості;

- навчання з підкріпленням - для кожного прецеденту існує пара «ситуація, реакція середовища» (навчання з підкріпленням можна вважати окремим випадком навчання з учителем, так і окремим випадком навчання без вчителя).

Існують також інші, менш поширені способи навчання, наприклад, активне навчання (той, якого навчають алгоритм має можливість призначити наступну досліджувану ситуацію), часткове залучення вчителя, трансдуктівное навчання, многозадачное і різноманітне навчання, але відмінності в цих способах несуттєві в рамках даної проблеми. Всі перераховані способи можна використовувати для класифікації традиційних методів машинного навчання і для класифікації алгоритмів навчання нейронних мереж, що реалізують будь-який з них.

Сучасне машинне навчання стикається з гострою проблемою універсальності, оскільки практично не є однорідного простору алгоритмів і методу загального вирішення проблеми індукції.

2.2 Введення в теорію навчання з підкріпленням

Ідея, що ми навчаємось через взаємодію з нашим середовищем, є, напевно, першою, що виникає у нас, коли ми думаємо про природу навчання. Коли немовля грає, махає руками або дивиться, вона не має явного вчителя, але має пряме сенсомоторне підключення до свого оточення. Здійснення цього зв'язку дає велику кількість інформації про причину і наслідки, про наслідки дій

і про те, що робити для досягнення цілей. Протягом нашого життя, такі взаємодії, безсумнівно, є основним джерелом знань про наше середовище і нас самих. Чи ми навчаємося керувати автомобілем або вести розмову, ми гостро усвідомлюємо, як наше середовище реагує на те, що ми робимо, і прагнемо впливати на те, що відбувається через нашу поведінку. Навчання від взаємодії є фундаментальною ідеєю, що лежить в основі майже всіх теорій навчання та інтелекту.

У цій дипломній роботі я досліджую обчислювальний підхід до навчання від взаємодії. Замість того, щоб безпосередньо теоретизувати, як навчаються люди або тварини, ми досліджуємо ідеалізовані ситуації навчання і оцінюємо ефективність різних методів навчання. Тобто ми приймаємо перспективу дослідника або інженера з технічної розвідки. Ми досліджуємо проекти для машин, які є ефективними у вирішенні проблем навчання з наукових або економічних інтересів, оцінюючи проекти за допомогою математичного аналізу або обчислювальних експериментів. Підхід, який ми досліджуємо, називається підкріпленням, набагато більше орієнтований на цілеспрямоване навчання від взаємодії, ніж інші підходи до машинного навчання.

Проблема навчання з підкріпленням передбачає вивчення того, що робити, як зіставити ситуації з діями, щоб максимально збільшити числовий сигнал винагороди. Істотно це проблеми замкнутого циклу, оскільки дії навчальної системи впливають на її пізніші входи. Більше того, учневі не говорять, які дії слід приймати, як у багатьох формах машинного навчання, а замість цього виявляють, які дії дають найбільшу винагороду, випробовуючи їх. У найбільш цікавих і складних випадках дії можуть впливати не тільки на безпосередню винагороду, але й на наступну ситуацію і, через це, на всі наступні нагороди. Ці три характеристики - це замкнутий цикл у суттєвому

способі, не маючи прямих вказівок щодо того, які дії слід вживати, і де наслідки дій, включаючи сигнали винагороди, відтворюються протягом тривалого періоду часу - це три найважливіші відмінні риси проблеми навчання.

Основна ідея полягає в простому захопленні найважливіших аспектів реальної проблеми, що стоїть перед агентом навчання, що взаємодіє зі своїм середовищем для досягнення мети. Зрозуміло, що такий агент повинен певною мірою відчувати стан навколишнього середовища і повинен бути в змозі вжити заходів, які впливають на стан. Агент також повинен мати мету або цілі, пов'язані зі станом навколишнього середовища. Формулювання має включити саме ці три аспекти - відчуття, дію і мету.

2.3 Відмінність навчання з підкріпленням від інших методів машинного навчання

Будь-який метод, який добре підходить для вирішення такого роду проблем, ми вважаємо методом навчання підкріплення. Навчання з підкріпленням відрізняється від навчання з учителем, вид навчання, що вивчається у найбільш актуальних дослідженнях у галузі машинного навчання. Навчання з учителем - це навчання з навчального набору позначених прикладів, наданих знаючим зовнішнім керівником. Кожен приклад - це опис ситуації разом із специфікацією - міткою - правильної дії, яку повинна прийняти система до цієї ситуації, яка часто визначає категорію, до якої належить ситуація. Мета цього виду навчання полягає в тому, щоб система екстраполювала, або узагальнювала, свої відповіді так, щоб вона діяла правильно в ситуаціях, які не є в навчальному наборі. Це важливий вид навчання, але сам по собі він не є

адекватним для навчання від взаємодії. У інтерактивних задачах часто недоцільно отримувати приклади бажаної поведінки, які є правильними і репрезентативними для всіх ситуацій, в яких агент повинен діяти. У незвіданій території, де можна очікувати, що навчання буде найбільш корисним, агент повинен вміти вчитися на власному досвіді.

Навчання з підкріпленням також відрізняється від того, що дослідники машинного навчання називають навчання без нагляду, яке, як правило, стосується находить приховані структури в нерозмічених даних. Терміни «Навчання з учителем » та «Навчання без учителя», вичерпно класифікують парадигми машинного навчання, але це не так. Хоча можна спокуситися думати про навчання підкріпленню як про нагляд без нагляду, оскільки він не спирається на приклади правильної поведінки, навчання підкріплення намагається максимізувати сигнал винагороди, а не намагатися знайти приховану структуру. Розкриття структури в досвіді агента, безумовно, може бути корисним у навчанні для підкріплення, але само по собі не стосується проблеми навчання агента з підсилення, як максимізації сигналу винагороди. Тому ми розглядаємо навчання підкріплення як третю парадигму машинного навчання, поряд з навчанням під наглядом, без нагляду та, можливо, іншими парадигмами.

2.4 Компроміс між дослідженням і використанням

Однією з проблем, що виникають саме у навчанні підкріплення, є компроміс між дослідженнями та використанням. Щоб отримати велику винагороду, агент з повинен віддавати перевагу діям, які він робив в минулому,

і виявив, що вони є ефективними у отриманні винагороди. Але щоб виявити такі дії, він повинен спробувати дії, які він не вибрав раніше. Агент повинен використовувати те, що він вже знає, щоб отримати винагороду, але він також повинен досліджувати, щоб у майбутньому зробити кращий вибір дій. Дилема полягає в тому, що ні розвідка, ні експлуатація не можуть бути здійснені виключно без відсутності завдання. Агент повинен спробувати різні дії і поступово віддавати перевагу тим, що здаються найкращими. За стохастичною задачею кожну дію потрібно багато разів намагатися отримати, щоб отримати достовірну оцінку її очікуваної винагороди. Дилема розвідки-експлуатації інтенсивно вивчалася математиками протягом багатьох десятиліть. Наразі ми просто зауважуємо, що вся проблема балансування досліджень і експлуатації навіть не виникає в контрольованому і неконтрольованому навчанні, принаймні в їх пуристських формах.

Іншою ключовою особливістю навчання підкріплення є те, що воно явно розглядає всю проблему цілеспрямованого агента, взаємодіючого з невизначеним середовищем. Це на відміну від багатьох підходів, які розглядають підзадачі, не звертаючись до того, як вони можуть увійти до більш широкої картини. Наприклад, ми згадали, що більшість досліджень з машинного навчання стосується навчання під наглядом без чіткого уточнення того, як така здатність, нарешті, буде корисною. Інші дослідники розробили теорії планування з загальними цілями, але без урахування ролі планування в процесі прийняття рішень в реальному часі, або питання про те, звідки випливають прогностичні моделі, необхідні для планування. Хоча ці підходи дали багато корисних результатів, їхня орієнтація на окремі підзадачі є значним обмеженням.

Підвищення рівня навчання здійснюється на протилежному шляху,

починаючи з повного, інтерактивного, цільового агента. Всі агенти, що вивчають підкріплення, мають чіткі цілі, можуть відчувати аспекти свого середовища і можуть вибирати дії, що впливають на їхнє середовище. Крім того, зазвичай передбачається, що агент повинен працювати, незважаючи на значну невизначеність навколишнього середовища. Коли навчання з підкріпленням передбачає планування, воно має вирішувати взаємодію між плануванням та вибором дій у реальному часі, а також питанням про те, як моделі навколишнього середовища набувають та вдосконалюються. Коли навчання підкріплення передбачає навчання під наглядом, воно робить це з конкретних причин, які визначають, які можливості є критичними і які не є. Для того, щоб наукові дослідження досягли прогресу, важливі підзадачі повинні бути виділені та вивчені, але вони повинні бути підзадачами, які відіграють чітку роль у повному, інтерактивному, цілеспрямованому агенті, навіть якщо всі деталі повного агента ще не можуть бути виконані.

Одним з найбільш захоплюючих аспектів сучасного навчання підкріплення є його суттєві та плідні взаємодії з іншими інженерними та науковими дисциплінами. Підвищення рівня навчання є частиною багаторічної тенденції у сфері мистецького інтелекту та машинного навчання з метою більшої інтеграції зі статистикою, оптимізацією та іншими математичними предметами. Наприклад, здатність деяких методів підкріплення вивчати з параметризованими апроксиматорами звертається до класичного «прокляття розмірності» в дослідженні операцій і теорії управління. Більш чітко, навчання посилення також сильно взаємоділо з психологією та неврологією, причому істотні вигоди йдуть в обох напрямках. З усіх форм машинного навчання, навчання підкріплення є найбільш близьким до виду навчання, яке роблять люди та інші тварини, і багато хто з основних алгоритмів навчання підкріплення

спочатку надихалися біологічними системами навчання. Навчання з підкріпленням також повернулося, як через психологічну модель навчання тваринам, яка краще відповідає деяким емпіричним даним, так і через потенційну модель частин системи винагороди мозку. Тіло цієї книги розвиває ідеї навчання з підкріпленням, що стосуються інженерних та технічних розвідок, з підключенням до психології та неврології, узагальнених у розділах ?

Нарешті, навчання з підкріпленням також є частиною більшої тенденції у сфері архівної розвідки до простих загальних принципів. З кінця 1960-х років багато вчених-розвідників вважали, що немає загальних принципів, які слід розкрити, що розум замість того, щоб володіти великою кількістю спеціальних цілей, процедур і евристик. Іноді говорили, що якщо ми можемо просто отримати достатню кількість фактів у машині, скажімо, мільйон або один мільярд, то вона стане розумною. Методи, засновані на загальних принципах, таких як пошук або навчання, були охарактеризовані як «слабкі методи», тоді як ті, що ґрунтувалися на конкретних знаннях, називалися «сильними методами». З нашої точки зору, це було просто передчасним: занадто мало коштів було введено в пошуки загальних принципів, щоб зробити висновок, що їх не було. Сучасний AI тепер включає в себе багато досліджень, що шукають загальні принципи навчання, пошуку і прийняття рішень, а також намагається включити величезну кількість знань в області. Незрозуміло, наскільки далеко зависає маятник, але наукові дослідження з підкріплення, безумовно, є частиною повороту назад до простіших і менших загальних принципів мистецького інтелекту.

2.5 Елементи навчання з підкріпленням

Крім агента та навколишнього середовища, можна виділити чотири основні підгрупи системи навчання підкріплення: політика, сигнал винагороди, ціннісна функція та, необов'язково, модель середовища.

Політика визначає спосіб поведінки навчального агента в певний час. Грубо кажучи, політика - це зіставлення від сприйнятих станів навколишнього середовища до дій, які мають бути вжиті в тих станах. Вона відповідає тому, що в психології будемо називати набором правил стимулювання-відповіді або асоціацій (за умови, що стимули включають ті, які можуть виходити зсередини тварини). У деяких випадках політика може бути простою функцією або пошуковою таблицею, тоді як в інших вона може включати обширні обчислення, такі як процес пошуку. Політика є ядром навчального агента з підкріплення в тому сенсі, що він є достатнім для визначення поведінки. Загалом, політики можуть бути стохастичними.

Сигнал за винагороду визначає мету у навчанні. На кожному кроці часу середовище надсилає агенту навчального агента єдине число, винагороду. Єдина мета агента - максимізувати загальну винагороду, яку вона отримує в довгостроковій перспективі. Таким чином, сигнал винагороди визначає, якими є хороші та погані події для агента. У біологічній системі ми можемо думати про нагороди як про аналогію досвіду задоволення або болю. Вони є безпосередніми та визначальними ознаками проблеми, з якою стикається агент. Винагорода, надіслана агенту в будь-який час, залежить від поточної дії агента та поточного стану середовища агента. Агент не може змінити цей процес. Єдиний спосіб, яким агент може впливати на сигнал винагороди, - це через його дії, які можуть

мати прямий вплив на винагороду або непрямий ефект через зміну стану навколишнього середовища. У нашому прикладі Філ, який їсть сніданок, агент з навчання підкріпленню, що керує своєю поведінкою, може отримувати різні сигнали нагороди, коли він їсть свій сніданок, залежно від того, наскільки він голодний, з яким настроєм він перебуває, та інших особливостей його тіла, яке частина його внутрішнього підкріплення середовища навчального агента. Сигнал винагороди є основною основою для зміни політики. Якщо за діями, вибраними політикою, слідує низька винагорода, політику можна змінити, щоб у майбутньому вибрати якусь іншу дію в цій ситуації. Загалом, сигнали винагороди можуть бути стохастичними функціями стану навколишнього середовища та вжитих дій.

В той час як сигнал винагороди вказує на те, що є гарним у безпосередньому сенсі, функція цінності визначає, що добре в довгостроковій перспективі. Грубо кажучи, значення держави - це загальна сума винагороди, яку агент може розраховувати накопичувати в майбутньому, починаючи з цього стану. В той час як винагороди визначають безпосередню, внутрішню бажаність стану навколишнього середовища, значення вказують на довгострокову бажаність держав після врахування держав, які, ймовірно, слідуватимуть, і нагороди, доступні в цих державах. Наприклад, держава завжди може приносити низьке безпосереднє винагороду, але все ще має високу цінність, оскільки за нею регулярно йдуть інші держави, які дають високу винагороду. Або навпаки може бути правдою. Щоб зробити людську аналогію, винагорода нагадує задоволення (якщо високе) і біль (якщо низький), тоді як значення відповідають більш чіткому і далекоглядному оцінці того, наскільки ми задоволені чи незадоволені, що наше середовище перебуває в певному стані. Висловлюючись таким чином, ми сподіваємося, що цінні функції формалізують

базову та знайоме уявлення.

Нагороди в певному сенсі первинні, тоді як значення, як передбачення нагород, є вторинними. Без винагород не може бути ніяких цінностей, і єдина мета оцінювання цінностей полягає в тому, щоб досягти більшої винагороди. Тим не менш, саме цінності, з якими ми найбільше турбуємося при прийнятті та оцінці рішень. Вибір дій здійснюється на основі оціночних суджень. Ми прагнемо до дій, які приносять держави найвищої цінності, а не найвищої нагороди, тому що ці дії отримують найбільшу нагороду за нас у довгостроковій перспективі. У процесі прийняття рішень і планування отримана величина, що називається величиною, є тією, з якою ми найбільше турбуємося. На жаль, набагато складніше визначити значення, ніж визначити винагороду. Нагороди в основному наводяться безпосередньо навколишнім середовищем, але значення повинні бути оцінені і переоцінені з послідовностей спостережень, які агент робить протягом всього свого життя. Насправді, найважливішою складовою майже всіх алгоритмів підкріплення, які ми розглядаємо, є метод ефективного оцінювання значень. Центральна роль оцінки вартості є, мабуть, найважливішою річчю, яку ми дізналися про навчання підкріплення за останні кілька десятиліть.

Четвертим і кінцевим елементом деяких навчальних систем підкріплення є модель навколишнього середовища. Це те, що імітує поведінку середовища або, загалом, дозволяє зробити висновки про те, як буде поводитися середовище. Наприклад, з урахуванням стану і дії, модель може передбачити наступний стан і наступну винагороду. Моделі використовуються для планування, під яким ми розуміємо будь-який спосіб прийняття рішення про хід дій, розглядаючи можливі майбутні ситуації до того, як вони дійсно виникнуть. Методи розв'язання проблем навчання підкріплення, які використовують моделі

та планування, називаються методами, заснованими на моделях, на відміну від більш простих методів без моделей, які явно вивчаються методами проб і помилок, які розглядаються як майже протилежні планування. У главі 9 ми досліджуємо системи навчання навчання, які одночасно вивчають методом проб і помилок, вивчають модель середовища і використовують модель планування. Сучасне навчання підкріплення охоплює спектр від низькорівневого, пробного і помилкового навчання до високого рівня, дорадчого планування.

2.6 Дисконтована винагорода

В навчанні з підкріпленням кожен агент намагається максимізувати довгострокову винагороду аніж винагороду в даний крок часу. Тобто якщо винагороди, що отримує агент після кроку часу t позначити як $\bar{r} = [r_{t+1}, r_{t+2}, r_{t+3}, \dots]$, тоді агент має максимізувати винагороду R_t, що є функцією послідовності $\bar{r}$.

Повну винагороду можна обчислити за формулою:

$$R_t = r_{t+1} + r_{t+2} + \cdots + r_\tau, \tag{2.1}$$

де τ термінальний стан.

Однак зазвичай використовують дтсконту винагороду, щоб надати більшу вагу ближчим нагородам, отриманим поблизу, ніж нагородам, отриманим у майбутньому.

$$R_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+2} + \cdots = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}, \quad (2.2)$$

де дисконтуюча змінна $\gamma \epsilon [0,1]$ зменшує майбутній очікуваний виграш.

Якщо $\gamma = 1$, то даний виграш співпадає з повною винагородою, Якщо $\gamma = 0$, то агент отримує винагороду з тепершніго кроку (жадібна стратегія). Для всіх інших значень агент враховує майбутні виграші тою чи іншою мірою.

2.7 Функція корисності

Більшість алгоритмів навчання з підкріпленням використовують функцію корисності, це може бути функція корисності стану *V(s)*, що вираховує яку винагороду може отримати агент в середовищі зі станом *s* або функція корисності дії *Q(s,a)*, що виміряє наскільки ейективно застосовувати дію *a* в стані *s*. Зазвичай функція корсності визначається стосовно певної політики.

Допустимо у нас є політика π, що відображає стани $s \epsilon S$ і дії $a \epsilon A(s)$ у ймовірне значення *π(s,a)*, тоді *V(s)* можу бути виражений як:

$$V^{\pi}(s) = E_{\pi}[R_t \mid S_t = s] = \quad (2.3)$$
$$= E_{\pi}\left\{\sum_{k=0}^{\infty} \gamma^k r_{k+t+1} \,\middle|\, s_t = s\right\},$$

де E_{π} очікуване значення якщо агент слідує політиці π. Функцію Корисності дії можна виразати як:

$$Q^{\pi}(s,a) = E_{\pi}[R_t | S_t = s, a_t = a] = \quad (2.3)$$
$$= E_{\pi}\left\{\sum_{k=0}^{\infty} \gamma^k r_{k+t+1} \middle| s_t = s, a_t = a\right\}.$$

Для безперервного простору стану та дії функції $\mathbf{V^{\pi}}(\mathbf{s})$ та $\mathbf{Q^{\pi}}(\mathbf{s},\mathbf{a})$ можна виразити за допогою методыв наближення функцій.

Оптимальною політикою π* є така, що максимізує очікувану винагороду, така політика завжди існує і вона має бути кращою за всі інші політики. Оптимальні функції корисності можна виразити як:

$$V^*(s) = \max_{\pi} V^{\pi}(s), \quad (2.4)$$

$$Q^*(s,a) = \max_{\pi} Q^{\pi}(s,a), \quad (2.5)$$

Можна переписати $Q^*(s,a)$ використовуючи V^*:

$$Q^*(s,a) = E\{r_{t+1} + \gamma V^*(s_{t+1}) | s_t = s, a_t = a\}. \quad (2.6)$$

Також можна переписати V^* у формі , що назтваєть оптимальне рівння Белмана:

$$V^*(s) = \max_{a \epsilon A(s)} Q^{\pi^*}(s,a) = \quad (2.7)$$
$$= \max_{a} E\{r_{t+1} + \gamma V^*(s_{t+1}) | s_t = s, a_t = a\}.$$

Оптимальне рівняння Беллмана для $Q^*(s, a)$:

$$Q^*(s, a) = E\left\{r_{t+1} + \gamma \max_{a'} Q^*(s_{t+1}, a') \,|s_t = s, a_t = a\right\}, \quad (2.8)$$

де *a'* дія, що обрана агентом у наступному стані і дає максимальну винагороду.

2.8 Temporal-Difference(TD) підхід

Для навчання мереж був використаний Temporal-Difference підхід. Він крмбінує деякі ідея з динамічного програмування та Монте-Карло(MC). TD похожий на MC тим, що навчається напряму з досвіду та є model free, тобто вивсчає оптимальну політику, а не намагається створити модель середовища. На відміну від MC, TD не чекає закінчення епізоду щоб обновити винагороду , а чекає тільки наступного кроку часу щоб оновити величину очікуваного значення корисності (V). Наприклад в MC:

$$V(s) = E\,[Gt \mid St = s], \quad (2.9)$$

$$Gt = Rt + 1 + \gamma Rt + 2 + \cdots, \quad (2.10)$$

а у TD:

$$V(s) = E[\mathrm{Gt} \mid \mathrm{St} = \mathrm{s}] = \tag{2.11}$$
$$= \mathrm{E}[\mathrm{R_{t+1}} + \lambda \mathrm{R_{t+2}} + \lambda^2 \mathrm{R_{t+3}} + \cdots | \mathrm{St} = \mathrm{s}] =$$
$$= \mathrm{E}[\mathrm{R_{t+1}} + \lambda(\mathrm{R_{t+2}} + \lambda \mathrm{R_{t+3}} + \cdots) | \mathrm{St} = \mathrm{s}] =$$
$$= \mathrm{E}[\mathrm{R_{t+1}} + \lambda \mathrm{G_{t+1}} | \mathrm{St} = \mathrm{s}] =$$
$$= \mathrm{E}[\mathrm{R_{t+1}} + \lambda \mathrm{v}(\mathrm{S_{t+1}}) | \mathrm{St} = \mathrm{s}],$$

Вони використовують рівняння Беллмана для оновлення оцінок. Щоб оновити значення V використовуються ці 3 формули.

$$DP \rightarrow V(S_t) \leftarrow E_\pi\big(\mathrm{R_{t+1}} + \lambda \mathrm{v}(\mathrm{S_{t+1}})\big), \tag{2.12}$$
$$MC \rightarrow V(S_t) \leftarrow V(S_t) + \alpha\big(\mathrm{G_t} - V(S_t)\big), \tag{2.13}$$
$$TD \rightarrow V(S_t) \leftarrow V(S_t) + \alpha\big(\mathrm{R_{t+1}} + \gamma V(S_{t+1}) - V(S_t)\big), \tag{2.14}$$

де $\mathrm{G_t} \rightarrow$ MC ціль,

$\mathrm{R_{t+1}} + \lambda V(S_t) \rightarrow$ TD-ціль,

$\mathrm{R_{t+1}} + \lambda V(S_t) - V(S_t) \rightarrow$ TD-похибка

У TD у кожен крок часу TD ціль використовує спостережену винагороду та $\mathrm{R_{t+1}}$ і поточну очікувану $V(S_{t+1})$,щоб зменшити TD-похибку. Повгий алгоритм можна побачити на зображенні 2.1.

Тоді звичайний метод TD, відомий як TD(0):

$$V(S_t) \leftarrow V(S_t) + \alpha\left(\mathrm{R_{t+1}} + \gamma V(S_{t+1}) - V(S_t)\right), \tag{2.15}$$

Algorithm 2.1 TD Learning algorithm.

repeat (for each episode)
  Initialize s
  repeat (for each time step t)
    For state s, choose an action a based on the policy π.
    Take action a; observe the next state s', and reward r.
    Calculate $V(s)$ from Equation (2.22).
    Set $s \leftarrow s'$.
  until (s is terminal)
until (finish all episodes)

Рисунок 2.1 - Алгоритм підходу TD

На основі TD-похибки можна функцію корисності дії Q(s,a), для цього є два підходи: SARSA(state-action-reward-state-action) і Q-learning. Ці алгоритми можна побачити на зображеннях 2.2 і 2.3. Ми будемо досліджувати алгоритми побудовані на основі другого підходу.

Використовуючи SARSA, Q(s,a) оновлюється за правилом:

$$Q(s_t, a_t) = Q(s_t, a_t) + \alpha \triangle_t, \tag{2.16}$$

де $\triangle_t$ – TD-похибка, що вираховується за формулою:

$$\triangle_t = \mathrm{r}_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t), \tag{2.17}$$

Algorithm 2.2 Sarsa Learning algorithm.

$Q(s, a)$ is initialized arbitrarily $\forall\, s \in \mathcal{S},\ a \in \mathcal{A}$
repeat (for each episode)
  Initialize s.
  For state s, choose an action a based on a certain policy (e.g., ϵ-greedy).
  repeat (for each time step t)
    Take action a; observe the next state s_{t+1}, and reward r.
    For state s_{t+1}, choose an action a_{t+1} based on a certain policy (e.g., ϵ-greedy).
    Calculate $Q(s, a)$ from Equation (2.24).
    Set $s \leftarrow s_{t+1}$.
    Set $a \leftarrow a_{t+1}$.
  until (s is terminal)
until (finish all episodes)

Рисунок 2.2 - алгоритм навчання SARSA.

Q-learning відрізняється тим, що використовує найкращу дію в наступний крок часу замість дії a_{t+1} , тоді дані включать в себе множину з $(s_t, a_t, s_{t+1}, r_{t+1})$ і TD-похибка, що вираховується:

$$\triangle_t = r_{t+1} + \gamma(\max_{a'} Q^{\pi}(s', a') - Q^{\pi}(s, a), \tag{2.18}$$

Тоді функція корисності для дії і стану для політики π:

$$Q^{\pi}(s, a) = E[R|s_t = s, a_t = a], \tag{2.19}$$

Ця Q функція може бути рекурсивно переписана як:

$$Q^{\pi}(s,a) = E_{s'}\left[r(s,a) + \gamma E_{a'\sim\pi}[Q^{\pi}(s',a')]\right], \tag{2.20}$$

що можна записати як:

$$Q^{\pi}(s,a) = Q^{\pi}(s,a) + \alpha(r_{t+1} + \gamma\left(\max_{a'} Q^{\pi}(s',a') - Q^{\pi}(s,a)\right), \tag{2.21}$$

де s - поточний стан, a - поточна дія, r - поточна винагорода, s' - наступний стан, a' - наступна найкраща дія.

Algorithm 2.3 Q**-learning algorithm.**

$Q(s,a)$ is initialized arbitrarily $\forall\, s \in \mathcal{S},\ a \in \mathcal{A}$.
repeat (for each episode)
  Initialize s.
  repeat (for each time step t)
    For state s, choose an action a based on a certain policy (e.g., ϵ-greedy).
    Take action a; observe the next state s_{t+1}, and reward r.
    Calculate $Q(s,a)$ from Equation (2.24).
    Set $s \leftarrow s_{t+1}$.
  until (s is terminal)
until (finish all episodes)

Рисунок 2.3 - Алгоритм навчання Q-learning.

2.9 Deep Q-Network (DQN)

DQN вивчає значення корисності функції Q^* відповідно до оптимальної політики мінімізуючи втрати:

$$(\theta) = E_{s,a,r,s'}\left[(Q^*(s,a|\theta) - y)^2\right], \quad (2.22)$$
$$\text{де } y = r + \gamma \max_{a'} \bar{Q}^*(s', a'),$$

де $\bar{Q}$ є ціль Q-функції параметри якої періодично обновляються останніми θ, що допомагає стабілізувати навчання. Ще один компонент, що допомагає стабілізувати DQN є використання буферу досвіду D, що містить $s\ (s, a, r, s0)$.

Q-Learning може напряму використовуватись в цій ігрі вибираючи напрямок руху(вправо, вліво, вгору або вниз). Кожен агент має свою функцію корисності Q.

2.10 Policy Gradient (PG) алгоритми

Методи градієнта політики(PG) - ще один популярний вибір для різноманітність завдань RL. Головна ідея напряму настроювати параметри θ політики щоб максимізувати $J(\theta) = Es \sim p\pi, a \sim \pi\theta\ [R]$ роблячи кроки в напрямку $\nabla\theta J(\theta)$. Використовуючи уже визначену Q функцію, градієнт політики може бути записаний як:

$$\nabla\theta J(\theta) = Es \sim p\pi\,, a \sim \pi\theta\,[\nabla\theta\, log\pi\theta(a|s) Q\,\pi\,(s,a)], \qquad (2.23)$$

де $\mathrm{p}\pi$ є розподілом стану. Теорема про градієнт політики породила кілька практичних алгоритмів, які часто відрізняються тим, як вони оцінюють $\mathrm{Q}\pi$. Наприклад, можна просто використати зразок повернення $\mathrm{Rt} = \mathrm{PT\,i} = \mathrm{t}\,\gamma\,\mathrm{i} - \mathrm{t\,ri}$, що призводить до алгоритму REINFORCE. Крім того, можна було б дізнатися наближення функції справжньої дії та значення $\mathrm{Q}\pi\,(\mathrm{s},\mathrm{a})$ наприклад temporal-difference навчання.

Відомо, що методи градієнта політики демонструють високу дисперсію оцінки градієнта. Це посилюється в при використанні багатьох агентів; оскільки винагорода агента зазвичай залежить від дій багатьох агентів, але алгоритми враховують, що вони залежні тільки від власних дій агента(коли дії інших агентів не враховуються в процесі оптимізації агента), виявляє набагато більшу мінливість, тим самим збільшуючи дисперсію його градієнтів.

2.11 Метод Actor-Critic(AC)

AC[11] складається з двох компонентів: actor і critic. Actor – це політика моделі, що вибирає дію. Critic – це функція корисності, може оцінювати корисність дії, що обрав actor $Q(a|s)$ або функцією корисності для стану $V(s)$. Actor обновляє параметри політики θ для $\pi_\theta(a|s)$, в напрямку, що запропоновав critic. Структуру можна побачити на зображенні 3.4.

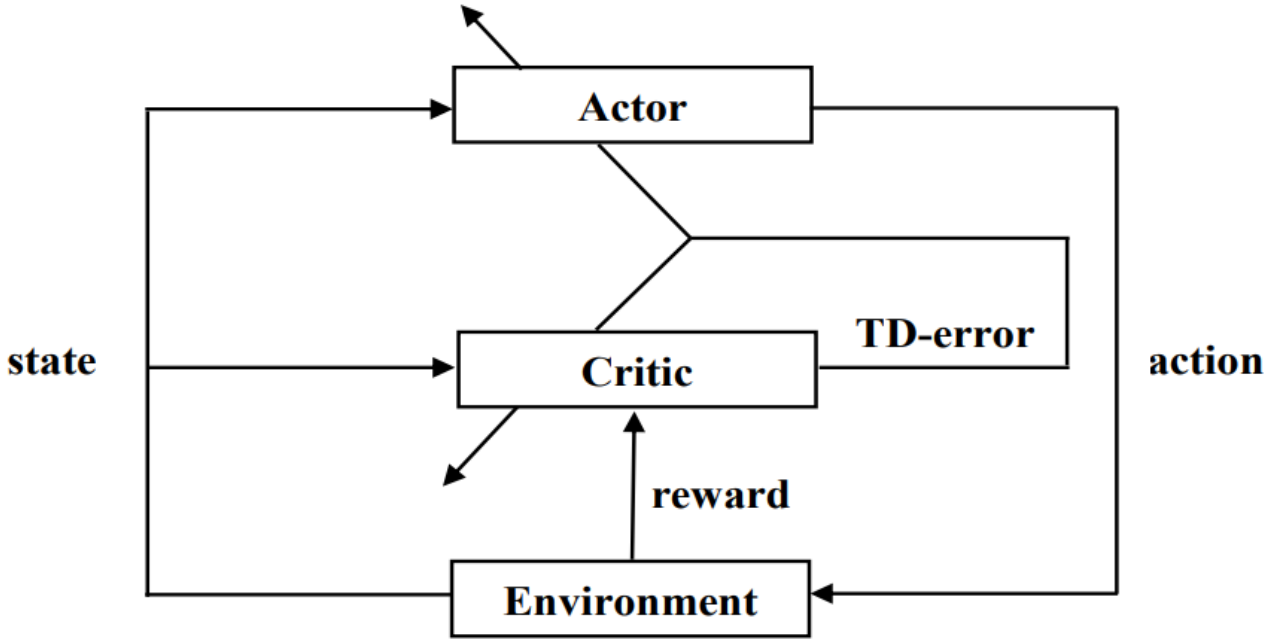


Рисунок 2.4 - Структура AC

2.12 Deep Deterministic Policy Gradient (DDPG)

DDPG[10] є model-free off-policy AC алгоритм, що комбіную DPG I DQN. Так як DQN він зберігає досвід(стан, дія, винагорода, наступний стан) у буфер перегляду, який зберігається протягом тренування та використовує цільову мережу щоб стабілізувати навчання і використовує для цього м'які оновлення за правилом :

$$\theta' \leftarrow \tau\theta + (1-\tau)\theta', \tag{2.24}$$

де θ' цільва мережа, $\tau \ll 1$.

Тоді получається, що цільва мережа обновляється повільно, раз в якусь кількість епізодів. Алгоритм зображений на рисунку 3.5.

Algorithm 1 Deep Deterministic Policy Gradient

Input: initial policy parameters θ, Q-function parameters ϕ, empty replay buffer $\mathcal{D}$
Set target parameters equal to main parameters $\theta_{\text{targ}} \leftarrow \theta$, $\phi_{\text{targ}} \leftarrow \phi$
repeat
  Observe state s and select action $a = \text{clip}(\mu_\theta(s) + \epsilon, a_{Low}, a_{High})$, where $\epsilon \sim \mathcal{N}$
  Execute a in the environment
  Observe next state s', reward r, and done signal d to indicate whether s' is terminal
  Store (s, a, r, s', d) in replay buffer $\mathcal{D}$
  If s' is terminal, reset environment state.
  if it's time to update **then**
    for however many updates **do**
      Randomly sample a batch of transitions, $B = \{(s, a, r, s', d)\}$ from $\mathcal{D}$
      Compute targets

$$y(r, s', d) = r + \gamma(1 - d)Q_{\phi_{\text{targ}}}(s', \mu_{\theta_{\text{targ}}}(s'))$$

      Update Q-function by one step of gradient descent using

$$\nabla_\phi \frac{1}{|B|} \sum_{(s,a,r,s',d) \in B} \left(Q_\phi(s, a) - y(r, s', d)\right)^2$$

      Update policy by one step of gradient ascent using

$$\nabla_\theta \frac{1}{|B|} \sum_{s \in B} Q_\phi(s, \mu_\theta(s))$$

      Update target networks with

$$\phi_{\text{targ}} \leftarrow \rho\phi_{\text{targ}} + (1 - \rho)\phi$$
$$\theta_{\text{targ}} \leftarrow \rho\theta_{\text{targ}} + (1 - \rho)\theta$$

    end for
  end if
until convergence

Рисунок 2.5 - DDPG алгоритм.

Щоб виявити найкращі дії агент не повиен зациклюватись на одній тій самій дії, а повинен находити нові. Для цього в алгоритмі DDPG використовуються шуми. Вони можуть використовуватись для дії або ваг мережі. Його потрібно вибрати відповідним до середовища.

$$\mu'(s) = \mu_\theta(s) + N, \tag{2.25}$$

де N – шум, μ_θ – детермінована політика.

Для дії були досліджені нормальний шум і шум згенерований процесом Орнштейна-Уленбека[8].

Нормальний (гауссівоский) генерує випадковий шум з нормального (гауссового) розподілу. Щільність ймовірності розподілу Гаусса вираховується за формулою:

$$p(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{\frac{-(x-m)^2}{2\sigma^2}} \tag{2.26}$$

Але, якщо генерувати декоррельований випадковий сигнал із нульовою середньою, його вплив буде усереднено з часом, і система просто коливатиметься на місці, не роблячи особливого прогресу. Тому також був дослідженний шум згенерований процессом Орнштейна-Уленбека. Він моделює швидкість частинки Брауна з тертям, що призводить до тимчасово співвідносних значень, зосереджених навколо 0. Процес є стаціонарним процесом Гаусса-Маркова. Процессом Орнштейна-Уленбека виражається формулою:

$$n_t = n_{t-1} + \theta(\mu - n_{t-1})dt + \sigma N(0,1)\sqrt{dt}, \quad (2.27)$$

де n_t шум в момент t,θ,μ та σ параметри, N(0,1) гаусівський шум,dt часовий крок.

А якщо шум утворюється в заданому часовому кроці співвідноситься з попереднім шумом, як шум з процессу Орнштейна-Уленбека , він, як правило, залишатиметься в тому ж напрямку довше, а не негайно скасовує себе, що, отже, дозволить збільшити швидкість і розморозити положення.

Також був досліджений Parameter noise[12], він вважається більш ефективнішим ніж інші. Parameter noise додає адаптивний шум до ваг мережі політики. Шум параметрів вводить випадковість безпосередньо в параметри агента, змінюючи типи рішень які він приймає так, щоб вони залежали від стану агента. Шум параметрів допомагає алгоритмам ефективніше досліджувати своє середовище, що призводить до більш високих показників та більш елегантної поведінки. Це скоріш за все тому, що навмисне додавання шуму до параметрів політики робить пошук агента послідовним у різні часові кроки, тоді як додавання шуму в простір дій призводить до більш непередбачуваного дослідження, яке не пов'язане з чим-небудь унікальним для параметрів агента.

Так як різні шари мережі мають різну чутливість до збурень, тому використовується шарова нормалізація, що забезпечує, що вихід збуреного шару (який буде входом до наступного) все ще знаходиться в аналогічному розподілі.

Алгоритм використовує адаптивну схему для регулювання розміру збурень простору параметрів. Це регулювання працює за допомогою вимірювання впливу збурень на простір дії та на те, чи більший або менший

рівень шуму в просторі дії від визначеної цілі. Ця хитраість дозволяє нам просунути проблему вибору масштабу шуму в простір дій, який є більш інтерпретаційним, ніж простір параметрів.

2.13 Висновки до розділу

В цьому розділі ми дізнались про машинне навчання, основні принципи, едемети, переваги та недоліки RL, його відмінності від інших областей ML. Також розглянули дтсконтну винагороду функцію корисності, основну проблему RL – компроміс між дослідженням.

Обрали підхід який будемо використовувати для вирішення задачі та ознаймились з описом та реалізацію алгоритмів яків підійдуть для вирішення диференційних задач з непреривнем простом дій та стану. DQN і PG не підійдуть для цієї задачі, але вони використовуються в AC і DDPG, що є розширенням для AC. Були описані методи та проблеми при навчнні алгоритмів.

РОЗДІЛ 3 ДОСЛІДЖЕННЯ МЕТОДІВ НАВЧАННЯ З ПІДКРІПЛЕННЯМ ДЛЯ ВИРІШЕННЯ ЗАДАЧІ ПЕРЕСЛІДУВАННЯ

3.1 Опис комп’ютерної симуляції

В цій задачі є непреривне поле на якому можуть бути присутні об’єкти 3 типів: гравцець, що втікає, переслідувє і нерухомі об’єкти . Всі об’єкти круглі. Рухомі об’єкти мають свою швидкість, напрям в якому вони рухаються, розмір і масу. Керувати гравцями можна вибираючи швидкість і напрям руху або напрямок в якому прискорюватись. В роботі порівнюються два підходи вирішення задачі, перший – це використовути математичні формули щоб вирахувати найортоший шлях, другий – використовуючи нейронні мережі. Фактом того, що один гравець догнав іншого буде те, що дистанція між ними буде менше ніж сума радіусів об’эктів.

Стан кожного об’єкту представляється як відстань до інших об’єктів по горизонталі та вертикалі, а також може може передавитись швидкість руху цих об’єктів.

Дія кожного агента представлена 4 числами, які позначають швидкість або прискорення у кожному напрямку. Знаходиться різниця між значеннями у протилежні напрямки і остаються 2 значення в діапазоні (-1,1) які позначають швидкості по горизонталі і вертикалі.

Якщо дія передає швидкість, тоді це значення нормалізується так щоб загальна швидкість дорівнювала максимальній.

$$v_x = \frac{v_x}{\sqrt{{v_x}^2 + {v_y}^2}} * max\_speed \quad (3.1)$$

$$v_y = \frac{v_y}{\sqrt{{v_x}^2 + {v_y}^2}} * \max\_speed \quad (3.2)$$

Якщо дія передає прискорення *f*, тоді появляються такі додаткові параметри як затухання – *damping*, масса *m* та *dt*, що позначає часовий крок, тоді швидкість буде змінюватись за формулою:

$$v = v * (1 - damping) \quad (3.3)$$

$$v = v + \frac{f}{m} * dt \quad (3.4)$$

І якщо швидкість буде перевищувати максимальну, вона нормалізується за попередньою формулою.

Винагорода залежить від дистанції до інших гравців та якщо конфліктуючі сторони зіткнулись, тоді винагорода збільшується для переслідувача і зменшується для втікача. Формула для обчислення дистанції:

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2} \quad (3.5)$$

Також додається винагорода при зіткненні втікача з переслідувачем. Переслідувачу додається фіксоване значення, а втікачу віднімається.

Як виглядає ігра можна побачити на рисунку 3.1.

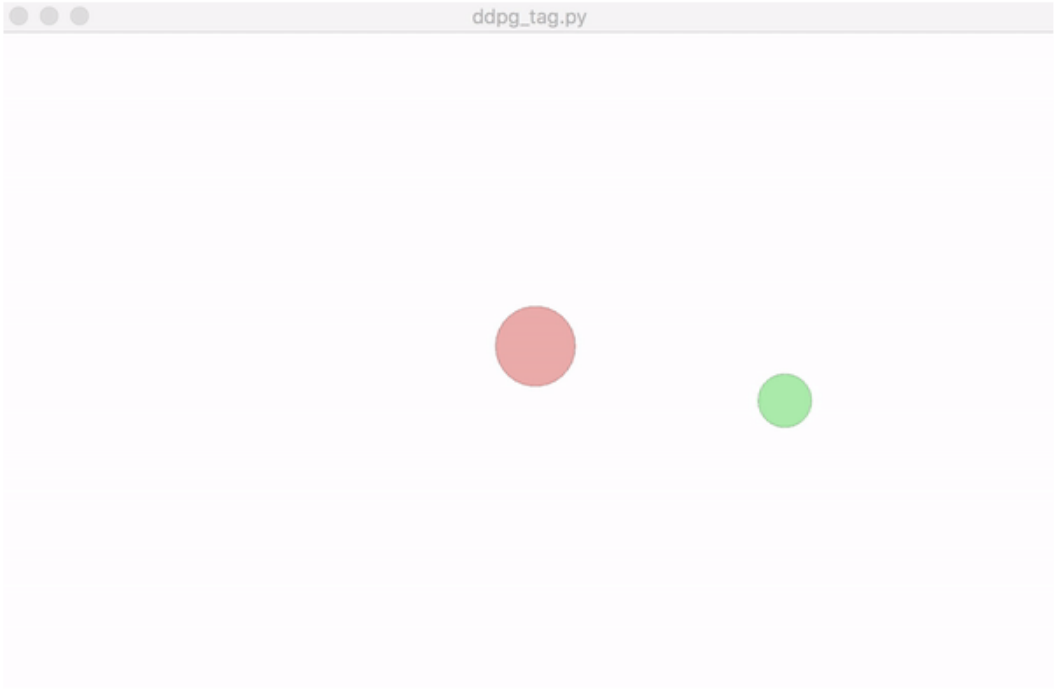


Рисунок 3.1 - Інтерфейс ігри

На зображенні 3.2 ми можемо побачити графічне зображення руху гравців, де T часом, v - швидкість втікача, a - швидкість переслідувача, z - пряма, що зєднує початкові точки гравців. β є кутом між z до напрямку руху втікача.

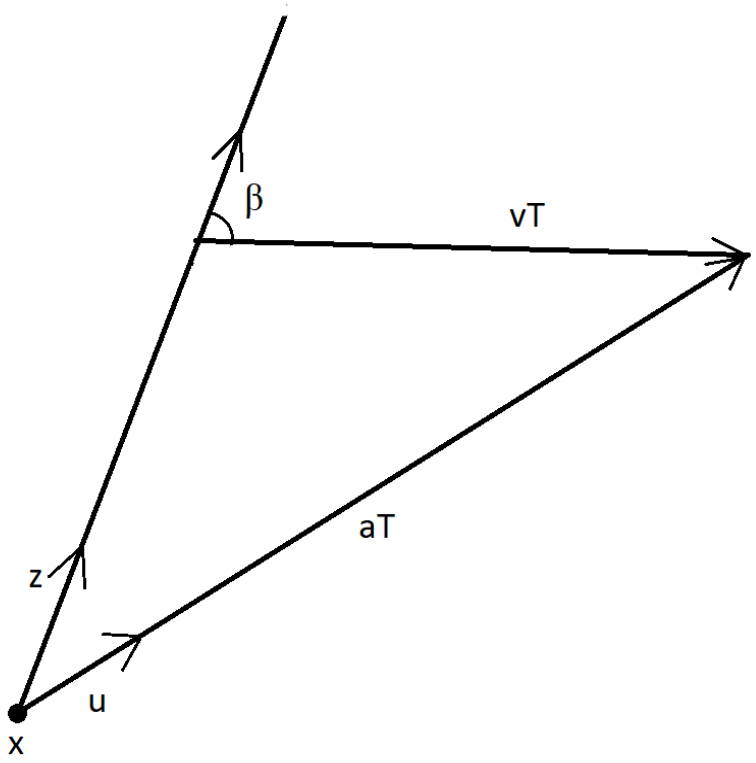


Рисунок 3.2 - Графіне зображення руху гравців

3.2 Опис теоретичних методів вирішення задач

Переслідувачу може передаватись повна або позиційна інформація про втікача. Якщо надається позиційна інформація переслідувач отримує тільки позицію втікачів: u(t) = u(z(t)). Тоді найефективнішим алгоритмом буде метод кривої погоні, тобто рухатись в напрямки поточного положення втікача. Щоб рухатись у такому напрямку треба найти дистанцію до точки по X і по У, тобто dx і dy і нормалізувати їх так щоб швидкість дорівнювала максимальній.

$$z(t) = \frac{\bar{d}}{\|\bar{d}\|}, \tag{3.6}$$

де $\bar{d}$ вектор від позиції переслідувача, до позиції втікача.

Якщо надається повна інформація: u(t) = u(z(t), v(t)), найефективніше використовувати паралельне переслідування. Тобто рухатись навипереження, знаючи в якій точці буде утікач через час t. Цю точку можна знайти:

$$p = \vec{V} * t, \tag{3.7}$$

де $\vec{V}$ вектор швидкості втікача.

Час через який вони зустрінуться можна вираховувати за формулою:

$$t = \frac{\left|\vec{d}\right|}{\sqrt{\left|\overrightarrow{V_1}\right|^2 + \left|\overrightarrow{V_2}\right|^2 - 2 * \left|\overrightarrow{V_1}\right| * \left|\overrightarrow{V_2}\right| * \cos(\varphi)}}, \tag{3.8}$$

де t через який зустрінуться гравці, $\overrightarrow{V_1}$ вектор швидкості втікача, $\overrightarrow{V_2}$ вектор швидкості переслідувача, φ кут міжвектором відстані між ігроками і вектором швидкості втікача.

Швидкість і напрям переслідувача вираховується як в першому експерименті, але до точки де вони зустрінуться.

3.3 Один втікач і один переслідувач

Так як задача де є один втікач і один переслідувач не дуже складна, для вирішення задачі використовувалась не складна мережа DDPG з 2 шарами і 64 нейронами. Крок начання для actor – 10^{-4}, для critic – 10^{-3}. Після кожної гри виконується 16 ітерацій навчання, розмір пам'яті – 10^{6}.

Так як новий, не тренований агент рухається випадковим чином, тоді шанси зловити противника дуже малі, це можна рахувати не можливим, тому функція, що вираховує винагороду складається з суми двох винагород. Перша, менша винагорода рахується як дистанція між гравцями. Для переслідувача ця дистанція зі знаком мінус, так як він намагається її зменшити. Друга частина це винагорода, за те що гравці зіткнулись. Це потрібно, для настпних дослідів, так як при зібльшені кількості гравців, дистанція не буде точно відображати потрібну винагороду, а функція, що має її замінити може бути досить складна.

Як змінювались траєктрої при навчанні DDPG, можна побачити на рисунку 3.3 при позиційгій інформації і на рис 3.4 при повній інформації.

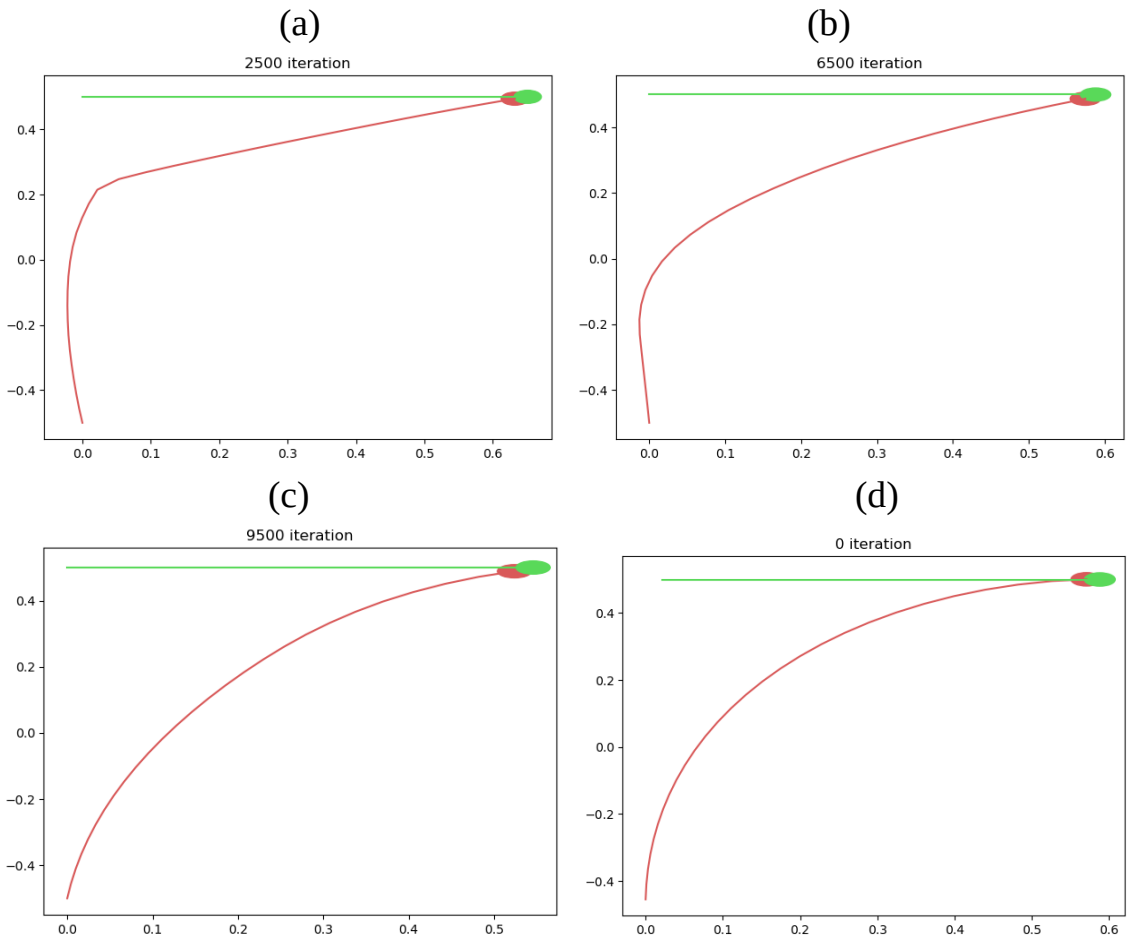


Рисунок 3.3 - На рисунках a,b,c показані результати нейронної мережі, де дії переслідувача вибирає мережа. На рисунку d показаний найефективніший шлях.

(a) (b)

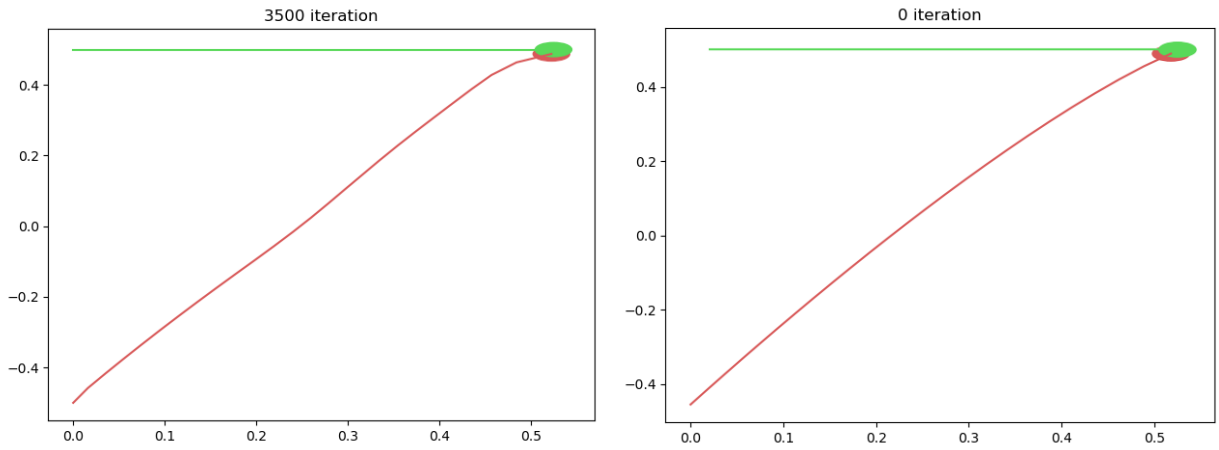


Рисунок 3.4 - На рисунку a показані результати нейронної мережі, де дії переслідувача вибирає мережа. На рисунку b показаний найефективніший шлях.

Алгоритм був протестовний з різними шумами для дослідження нових дій. Використовувалися нормальний шум, шум згенерований процесом Орнштейна-Уленбека та шум для параметрів. Всі шуми були згенеровані з цетром в 0 та середнім квадратичним відхиленням 0.1 та 0.2.

Для наступного досліду використовувався втікач, що рухається завжди вправо.

Вирішення ігри з позиційною іформацією догонною кривою заняло 28 епізодів, застосовуючи паралельне переслідування для гри з повною інформацією зайняло 25 епізоди.

Як змінювалась кількість епізодів для закінчення гри можна подивитись на рисунках 3.5, 3.6, 3.7 для гри з позиційною інформацією для 3 видів шумів та

на рисунку 3.8 для ігри з повною інформацією з шумом, що згенерований процесом Орнштейна-Уленбека та шумом для параметрів мережі.

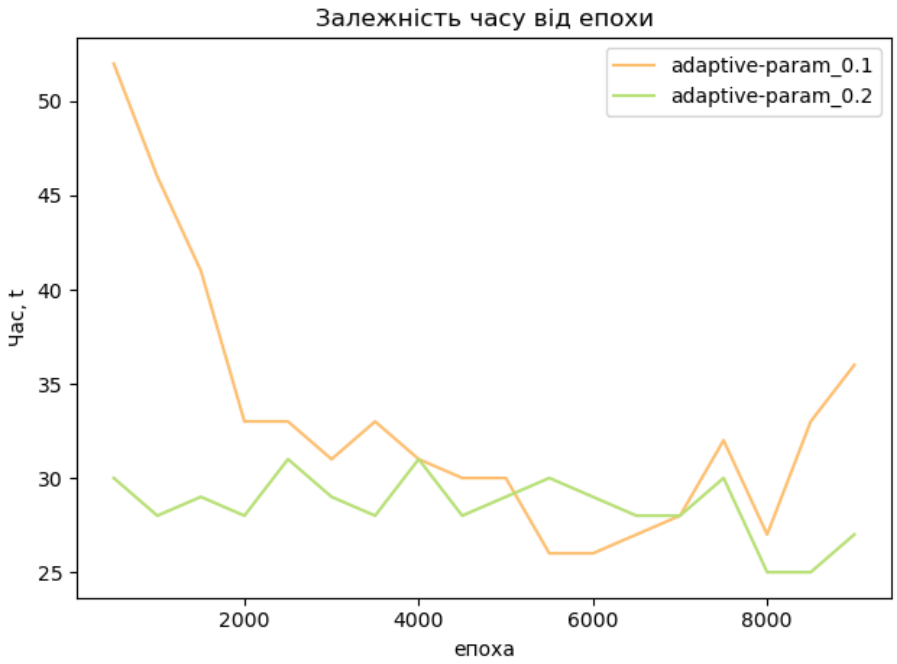


Рисунок 3.5 - залежність часу від кількості епох навчання з параметричним шумом

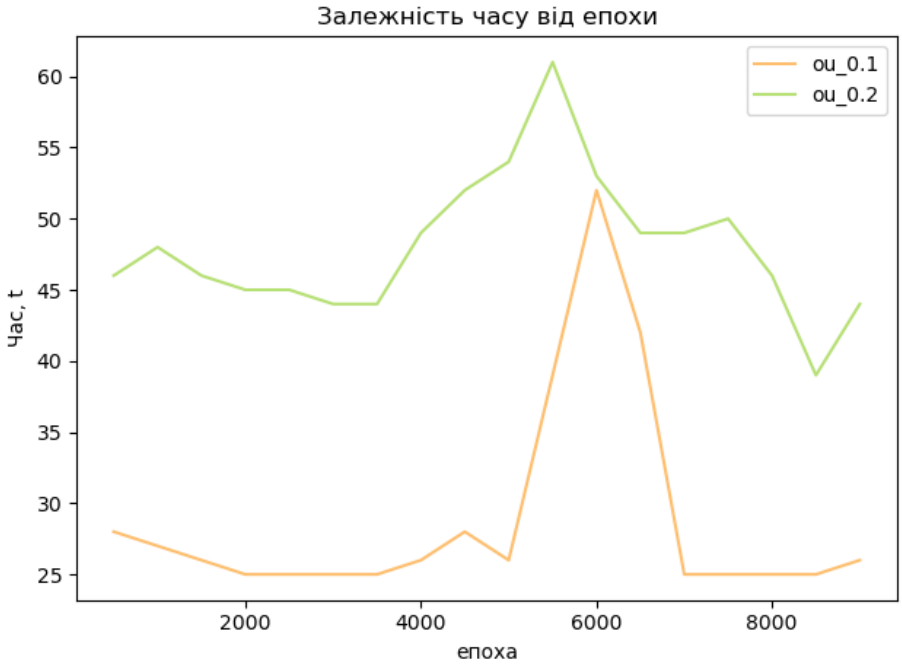


Рисунок 3.6 - залежність часу від кількості епох навчання з шум згенерований процесом Орнштейна-Уленбека

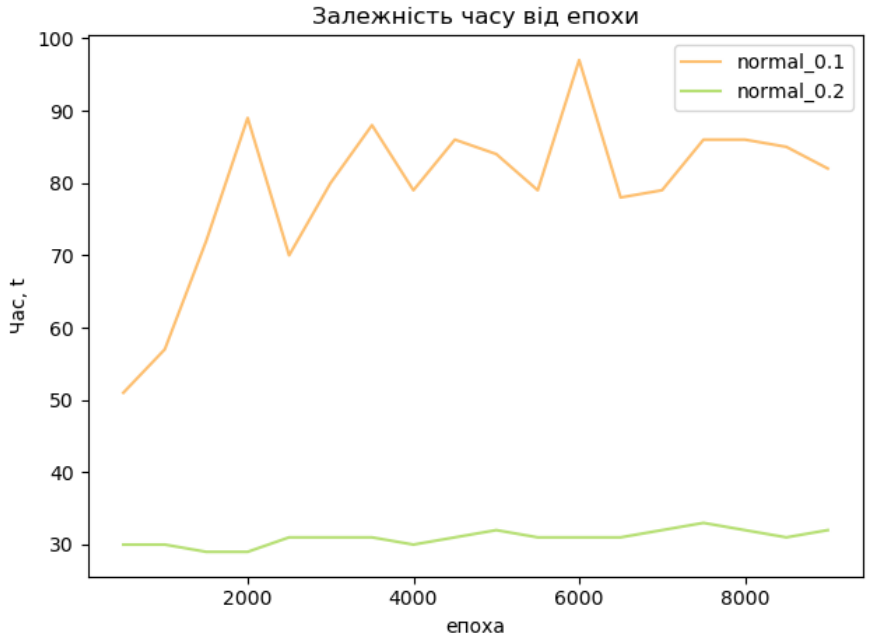


Рисунок 3.7 - залежність часу від кількості епох навчання з нормальним шумом.

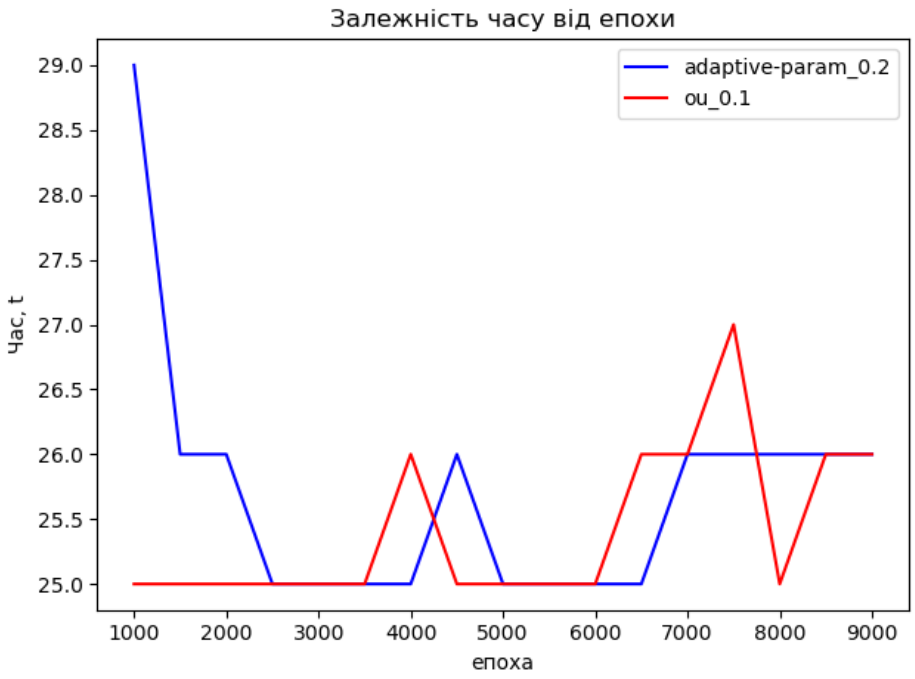


Рисунок 3.8 - залежність часу від кількості епох навчання для ігри з повною інформацією

Також в роботі був виміряний час зіткнення для декількох напрямів β в значеннях від 0 до π і порявняний з часом який затрачає переслідувач, що користується теоретичним методом. Це показано на зображенях 3.9 для ігри з позиційною інформацією і на зображенні 3.10 для ігри з повною інформацією. Для досліду були відібрені найкращі результати з попередніх графіків.

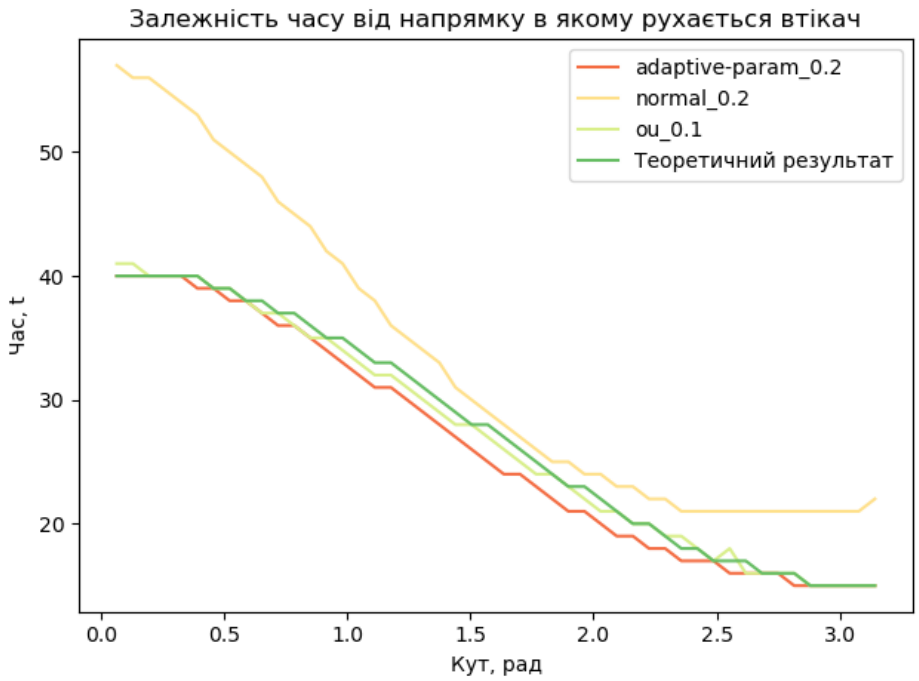


Рисунок 3.9 - Залежність часу від напрямку в якому рухається втікач

Такі ж тести буди зроблені для ігри з повною інформацією.

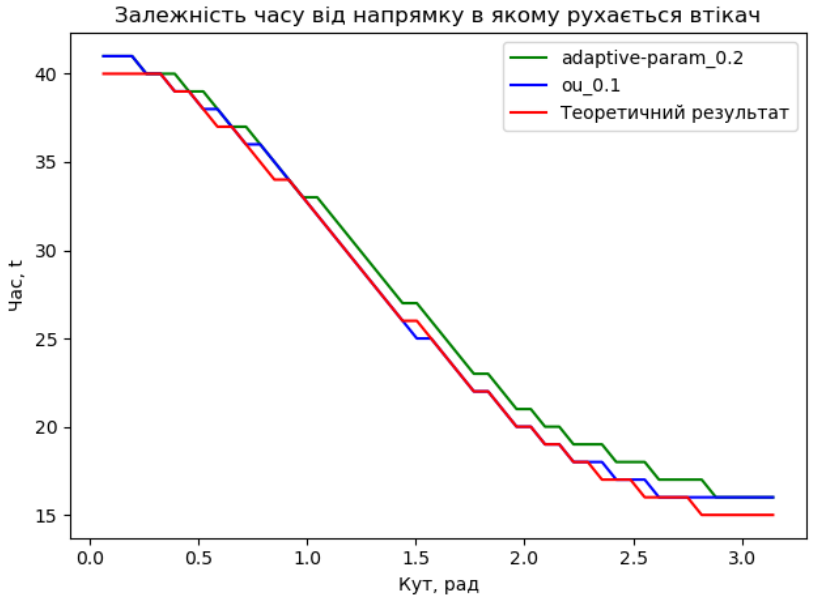


Рисунок 3.10 - Залежність часу від напрямку в якому рухається втікач у грі з повною інформацією

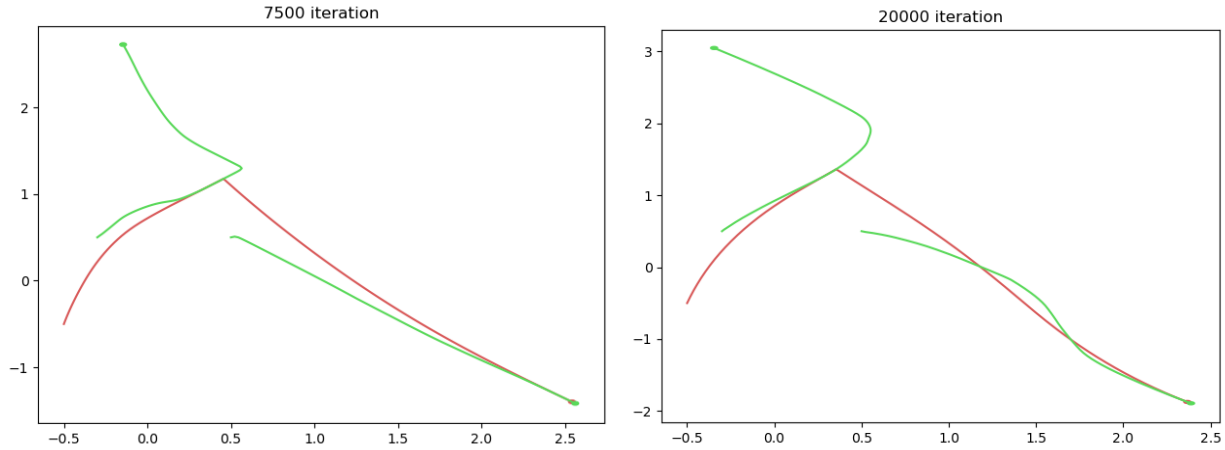
3.4 Декілька втікачів і один переслідувач

Переслідувач повинен догнати одного втікача, а потім другого, агенти знають тільки положення один відносно одного. В цьому випадку переслідувач має переслідувати ближчого втікача, а потім іншого. Оптимальними діями для втікачів буде те, що ближчий втікач повинен рухатись в напрямку від переслідувача, а дальній має вирахувати точку в якій доженуть першого втікача і рухатись від неї.

Для того щоб втікач знав чи дружнього гравця уже догнали, ця інформація була закодована, тобто до інформації про позицію інших гравців була додана 1 змінна, що приймає значення ‘1’ якщо союзного гравця уже було піймано, інакше присвоюється ‘0’.

На рисунках 3.11 можна побачити як діє мережа і порівняти з тим як діє теоретичний алгоритм.

(a) (b)



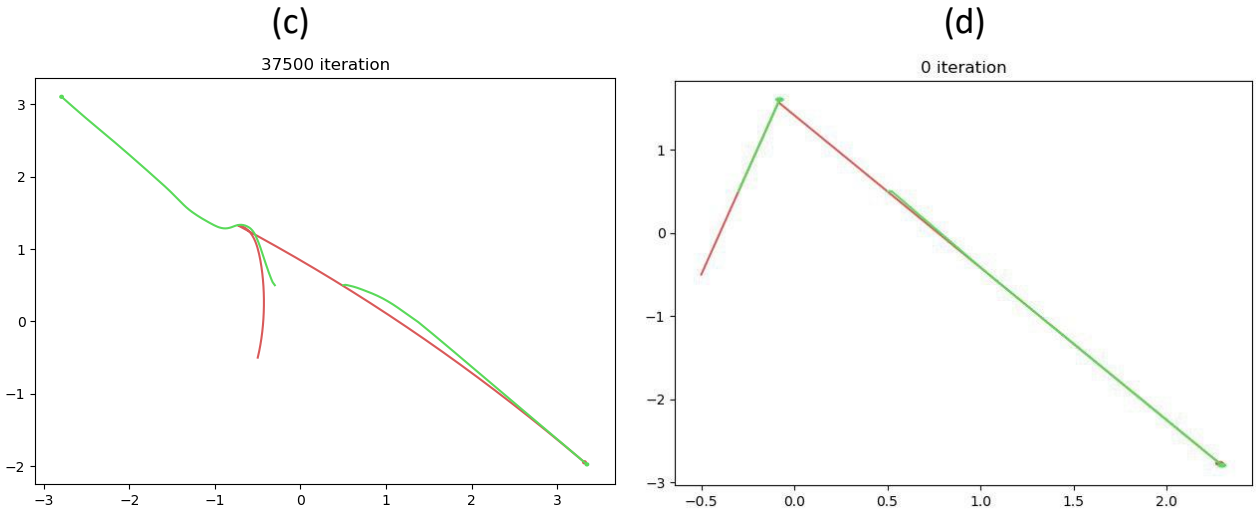


Рисунок 3.11 - На рисунках a,b,c показані результати нейронної мережі, де дії втікача вибирає мережа. На рисунку d показаний найефективніший шлях.

На рисунку 3.12 показані результати навчання двох мереж, одна з 2 шарами і 128 нейронами, а друга з 3 шарами і 128 нейронами.

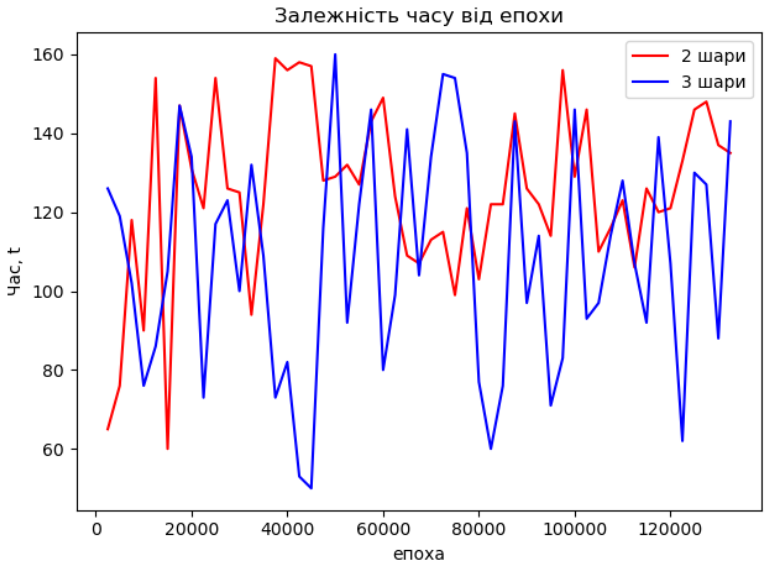


Рисунок 3.12 – Результати навчання мереж

Хоть і збільшення мережі не помогло далі використовуючи мережу з 3 шарами і 128 нейронами, використовуємо 1 масив епізодів, для двох агентів та збільшуємо його розмір в 10 разів. Результати можна побачити на рисунку 3.13.

Рисунок 3.13 Залежність часу закінчення від кількості епох навчання при збільшенні розміру пам‘яті в декілька разів.

Також була начена мережа яка вибирає дію переслідувачу, а дії втікачів виконує теоретичний алгоритм. На рисунку 3.14 можна побачити як діє переслідувач і на рисунку 3.15 зображена зміна результатів взалежності від кількості епох навчання.

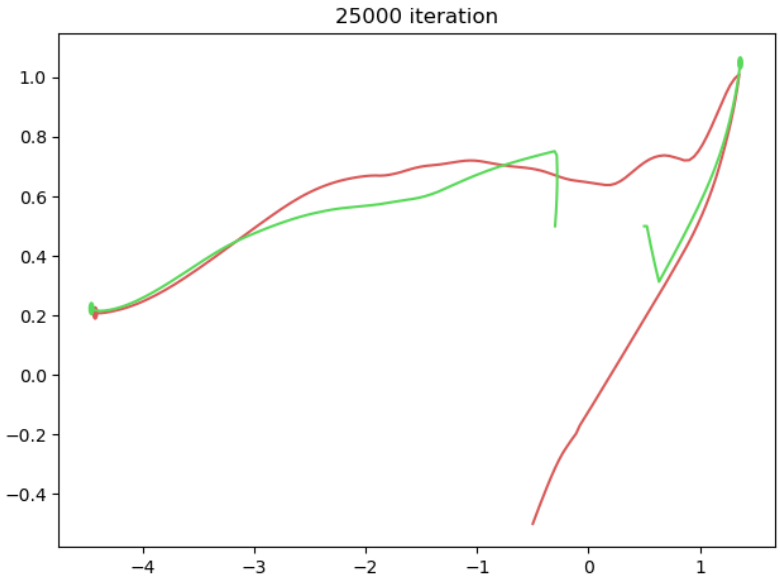


Рисунок 3.14 – дії переслідувача

Рисунок 3.15 – Зміна часу в залежності від кількості епох

3.5 Висновки до розділу

З дослідів ігри з позиційною інформацією, де є один втікач і один переслідувач можна замітити, що результати, що надала мережа, дуже близькі до результатів від теоретичних методів. Але вони не дуже стабільні, часто змінюються і деколи є навіть кращими ніж очікується, тобто якщо вирішувати задачу теоретичним методом, ігра завершиться за 28 епізодів, а за допомогою навчання з підкріпленням, може закінчитись в діапазоні від 25 до 31. Ігра з повною інформаціє в ідеальному випадку має закінчитись за 25 епізодів, результати показали, що при роботі алгоритму, вона закінчувалась за 25-26 ходів, що є дуже хорошим результатом.

Для цих дослідів використовувались такі шуми як нормальний шум, шум згенерований процесом Орнштейна-Уленбека та шум для параметрів. Нормальний шум виявився зовсім неефективний. Результати від шуму Орнштейна-Уленбека та параметричного шуму є схожими, але результати від параметричного шум трішки точніші і стабільніші.

Гра де є однин переслідувач і два втікача складніша, особливо коли треба навчати двох агентів одночасно. І ми можемо побачити, що результаті не такі хороші як для попередньої задачі, вони є дуже не стабільними. Вони є дуже не стабільні і збільшення мережі чи зміна кроку навчання не вирішувала цю проблему. Відчутні покразення були після використання спільного масиву де зберігаються попередні епізоди для всіх агентів і збільшенні його розміру в декілька разів, для чого потрібно багато пам’яті.

РОЗДІЛ 4 РОЗРОБЛЕННЯ СТАРТАП-ПРОЄКТУ

У даному розділі буде розглянуто ключові особливості розробленої системи як майбутнього стартап-проекту. Проект розглядатиметься як система голосового розпізнавання людини.

4.1 Опис ідеї проєкту

Спочатку проаналізуємо та подамо у вигляді таблиці зміст ідеї стартап-проекту, можливі напрямки застосування та основні вигоди, які може отримати користувач товару. Ці характеристики стартап-проекту зображено в таблиці 4.1.

Таблиця 4.1 - Опис ідеї стартап-проекту

Зміст ідеї	Напрямки застосування	Вигоди для користувача
Програмний додаток для дослідження нейронних мереж в задачі переслідування.	1. Застосування як середовище для дослідження задачі переслідування.	Можливість швидко отримати доступ до додатку Мінімальні витрати на апаратне забезпечення (сервер).
	2. Застосування як середовище для дослідження алгоритмів нейронних мереж для систем з неперервними простором дій та мультиагентних систем.	Легкий перехід між алгоритмами, використання декількох одночасно та вивід статистики.

Тепер зробимо аналіз потенційних техніко-економічних переваг ідеї порівняно із пропозиціями конкурентів. Результати аналізу зображено в таблиці 4.2.

Таблиця 4.2 – Визначення сильних, слабких та нейтральних характеристик ідеї проекту

№ п/п	Техніко-економічні характеристики ідеї	Товари/концепції конкурентів			W (слабка сторона)	N (нейтральна сторона)	S (сильна сторона)
		Мій проект	Конкурент 1	Конкурент 2			
1.	Ціна	200$/ рік	400$/ рік	50$/ рік		+	
2.	Прибутки	30000$/ рік	40000$/ рік	200$/ рік		+	
3.	Контроль якості	Аналітики та програмісти	Аналітики, програмісти та деякі клієнти	Програмісти			+
4.	Динаміка галузі	Швидка	Швидка	Швидка		+	
5.	Постійні витрати	10000$/ рік	20000$/ рік	5000$/ рік		+	

Продовження таблиці 4.2

№ п/п	Техніко-економічні характеристики ідеї	Товари/концепції конкурентів			W (слабка сторона)	N (нейтральна сторона)	S (сильна сторона)
		Мій проект	Конкурент 1	Конкурент 2			
6.	Змінні витрати	500$ - 1000$/ рік	1000$ - 2000$/ рік	2000$ - 5000$/ рік			+
7.	Патенти на продукти	Немає	Патент на кожний проект	Декілька патентів на винахід	+		
8.	Гнучкі ціни	Ціна єдина	Ціна варіюється з року в рік	Ціна єдина		+	
9.	Законодавчі обмеження	Обмеження на використання коду в комерційних цілях	Обмеження на використання коду в комерційних цілях	Обмеження на використання коду в комерційних цілях		+	

4.2 Технологічний аудит ідеї проєкту

Визначимо технологічну здійсненність ідеї проекту за допомогою аналізу таких складових, як технології, за якою буде виготовлено товар згідно ідеї проекту, існування таких технологій, чи їх необхідно розробити / доробити, доступність таких технологій авторам проекту. Результати даного аналізу зображено в таблиці 4.3.

Таблиця 4.3 – Технологічна здійсненність ідеї проекту

Ідея проекту	Технології її реалізації	Наявність технологій	Доступність технологій
Програмний додаток для дослідження нейронних мереж в задачі переслідування.	Технологія написання нейронних мереж Tensorflow	Так	Дані технології доступні.
	Технологія написання нейронних мереж PyTorch	Так	Дані технології доступні, однак, штатного функціоналу все ще не вистачає для вирішення поставлених задач.
Обрана технологія реалізації ідеї проекту: технологія написання нейронних мереж Tensorflow			

4.3 Аналіз ринкових можливостей запуску стартап-проєкту

Проведемо аналіз попиту: наявність попиту, обсяг, динаміка розвитку ринку. Результати даного аналізу зображено в таблиці 4.4.

Таблиця 4.4 – Попередня характеристика потенційного ринку стартап-проекту

№ п/п	*Показники стану ринку (найменування)*	*Характеристика*
1	Кількість головних гравців, од	3
2	Загальний обсяг продаж, грн/ум.од	300 000
3	Динаміка ринку (якісна оцінка)	Зростає
4	Наявність обмежень для входу (вказати характер обмежень)	Висока точність розпізнавання, швидкодія, невибагливість до ресурсів
5	Специфічні вимоги до стандартизації та сертифікації	Немає, GDRP
6	Середня норма рентабельності в галузі (або по ринку), %	80

Таким чином, за попереднім оцінюванням, ринок є привабливим для входження.

Надалі визначимо потенційні групи клієнтів, їх характеристики, та сформуємо орієнтовний перелік вимог до товару для кожної групи. Ці дані зображено в таблиці 4.5.

Таблиця 4.5 – Характеристика потенційних клієнтів стартап-проекту

№ п/п	*Потреба, що формує ринок*	*Цільова аудиторія (цільові сегменти ринку)*	*Відмінності у поведінці різних потенційних цільових груп клієнтів*	*Вимоги споживачів до товару*
1	Наведення ракет у воєнній промисловості	Великий бізнес	У великому бізнесу може виникнути потреба у інтеграції даної системи з іншими системами наводження та протидії.	Клієнти прагнуть мати технології кращі ніж у конкурентів. Також прагнуть точності та швидкодії від системи.

Після визначення потенційних груп клієнтів проведемо аналіз ринкового середовища: складемо таблиці факторів, що сприяють ринковому впровадженню проекту (таблиця 4.6), та факторів, що йому перешкоджають (таблиця 4.7).

Таблиця 4.6 – Фактори загроз

№ п/п	*Фактор*	*Зміст загрози*	*Можлива реакція компанії*
1	Відсутність попиту	Бізнес може не оцінити переваги продукту, або ж у цілому відмовитися від ведення нарад	Акцентувати увагу на клієнтах, що вже скористалися продуктом, якщо такі є, навести інфографіку результативності (очікувану), запропонувати знижку потенційному клієнту в рамках тендеру.

Таблиця 4.7 – Фактори можливостей

№ п/п	*Фактор*	*Зміст можливості*	*Можлива реакція компанії*
1	Кобрендінг	Пропозиція від певної компанії, що спеціалізується на дослідженні задачі переслідування або мультиагентних системах, розробити спільний продукт	Виділення частини штату на реалізацію проекту, підготовка акційних пропозицій по переходу на новий продукт існуючим клієнтам.

Надалі проведемо аналіз пропозиції: визначимо загальні риси конкуренції на ринку. Результати даного аналізу зображені в таблиці 4.8.

Таблиця 4.8 – Ступеневий аналіз конкуренції на ринку

Особливості конкурентного середовища	*В чому проявляється дана характеристика*	*Вплив на діяльність підприємства (можливі дії компанії, щоб бути конкурентоспроможною)*
1. Чиста конкуренція	Гравці ринку не мають явних переваг один над одним	Більш вигідні умови на тендерах, агресивний маркетинг
2. Регіональна конкуренція	Гравці ринку – інтернаціональні підприємства	Вихід на ті ринки, які ще не зайняті конкурентами
3. Внутрішньогалузева конкуренція	Гравці ринку знаходяться в одній галузі – розробці ПЗ	
4. Товарно-видова конкуренція	Усі продукти гравців ринку мають одне призначення	Розробка найбільш інтуїтивного інтерфейсу, розробка унікальних мовленнєвих пакетів, оптимізація алгоритмів (щоб аналіз проходив швидше, ніж у конкурентів)
5. Конкурентні переваги нецінові	Продукти відрізняються гнучкістю, функціоналом (незначно) і надійністю.	У маркетингу неявно порівнювати власний продукт з іншими, робити вигідні цінові пропозиції
6. Марочна конкуренція	Значна увага приділяється бренду, що розробив продукт	Кобрендінг

Тепер визначимо та обґрунтуємо фактори конкурентоспроможності, які зображені в таблиці 4.9.

Таблиця 4.9 – Обґрунтування факторів конкурентоспроможності

№ п/п	*Фактор конкурентоспроможності*	*Обґрунтування (наведення чинників, що роблять фактор для порівняння конкурентних проектів значущим)*
1	Невибагливість до апаратних ресурсів (серверів). А отже дешевизна апаратних ресурсів, потрібних для нашої системи	В продукті використане поєднання класичних методів голосового розпізнавання та методів Машинного навчання
2	Швидкодія	В продукті використане поєднання класичних методів голосового розпізнавання та методів Машинного навчання
3	Інтеграція	Продукт може бути використаний в будь-якому веб-сайті захищеному протоколом ssl. Продукт не потребує придбання спеціалізованого апаратного забезпечення
4	Модульність	Замовник може обирати тип продукту (веб-додаток, або мобільний-додаток)
5	Гнучкість	Кожен замовник має можливість замовити розширення функціоналу продукту під його конкретні задачі
6	Голосова аутентифікація	Клієнти замовника зможуть швидше увійти до системи

Таблиця 4.10 – Аналіз конкуренції в галузі за М. Портером

Складові аналізу	Прямі конкуренти в галузі	Потенційні конкуренти	Постачальни-ки	Клієнти	Товари-замінники
	Динаміка галузі, продуктова лінія, бар'єри проникнен-ня	Наявність товарних знаків, доступ до ресурсів, патенти на продукти	Концентрація постачальни-ків, диференціація витрат	Рівень чутливо-сті до зміни цін, прибутки, контроль якості	Ціна, лояльність спожива-чів
Виснов-ки:	Конкурен-ція не є інтенсив-ною, адже даний ринок ще ніким не зайнятий.	Для входу на ринок необхідно створити товарний знак та написати бета-версію програм-ного продукту. На даний момент потенцій-них конкурентів немає.	Постачаль-ники не диктують умови роботи на ринку, бо програмному продукту не потрібно постачання.	Клієнти диктують умови роботи на ринку, бо вони є єдиним джерелом прибутку компанії.	При наявності товарів замінників необхідно буде зменшува-ти ціну програм-ного продукту чи створю-вати ПЗ для інших технічних систем.

За визначеними факторами конкурентоспроможності проведемо аналіз сильних та слабких сторін стартап-проекту. Результати даного аналізу зображено в таблиці 4.11.

Таблиця 4.11 – Порівняльний аналіз сильних та слабких сторін системи «РЕ»

№ п/п	*Фактор конкуренто-спроможності*	*Бали 1-20*	*Рейтинг товарів-конкурентів у порівнянні з* РЕ						
			–3	*–2*	*–1*	*0*	*+1*	*+2*	*+3*
1	Інтеграція	15			*				
2	Модульність	16						*	
3	Гнучкість	18						*	

Тепер проведемо SWOT-аналіз на основі виділених загроз і можливостей, та сильних і слабких сторін проекту. SWOT-матриця зображено в таблиці 4.12.

Таблиця 4.12 – SWOT-аналіз стартап-проекту

Сильні сторони: Гнучкість, Модульність	Слабкі сторони: Інтеграція має пройти із залученням розробників на стороні замовника
Можливості: Кобрендінг	Загрози: Неточність прогнозування, відсутність попиту

На основі SWOT-аналізу розробимо альтернативи ринкової поведінки для виведення стартап-проекту на ринок та орієнтований оптимальний час їх

ринкової реалізації з огляду на потенційні проекти конкурентів, що можуть бути виведені на ринок. Дані альтернативи зображено в таблиці 4.13.

Таблиця 4.13 – Альтернативи ринкового впровадження стартап-проекту

№ п/п	*Альтернатива (орієнтовний комплекс заходів) ринкової поведінки*	*Ймовірність отримання ресурсів*	*Строки реалізації*
1	Реалізація можливості використання системи не тільки на веб-сайтах, а й телефонних додатках	Середня	18 місяців
2	Створення системи емоційного розпізнавання людини	Висока	24 місяці
3	Розробка MVP	Висока	12 місяців

Серед даних альтернатив було обрано третю альтернативу, адже строки її реалізації найменші та є ймовірність отримання ресурсів.

4.4 Розроблення ринкової стратегії проєкту

Для розроблення ринкової стратегії першим кроком необхідно описати цільові груп потенційних споживачів, які можна побачити в таблиці 4.14.

Таблиця 4.14 – Вибір цільових груп потенційних споживачів

№ п/п	Опис профілю цільової групи потенційних клієнтів	Готовність споживачів сприйняти продукт	Орієнтовний попит в межах цільової групи (сегменту)	Інтенсивність конкуренції в сегменті	Простота входу у сегмент
1.	Малий бізнес	Мала	5-10 підприємств в рік	Слабка	Середня
2.	Середній бізнес	Готові	5-10 підприємств в рік	Висока	Висока
3.	Великий бізнес	Готові	3-5 закладів в рік	Висока	Висока
Було обрано цільову групу підприємств групи малого бізнеса.					

Для роботи в обраних сегментах ринку необхідно сформувати базову стратегію розвитку, яку ображено в таблиці 4.15.

Таблиця 4.15 – Визначення базової стратегії розвитку

Стратегія охоплення ринку	Ключові конкурентоспроможні позиції відповідно до обраної альтернативи	Базова стратегія розвитку
Концентрація на потребах одного цільового сегменту – веб-сайтах.	Створений продукт є дешевим у використанні та іноваційним	Стратегія спеціалізації.

Наступним кроком є вибір стратегії конкурентної поведінки, яку зображено в таблиці 4.16.

Таблиця 4.16 – Визначення базової стратегії конкурентної поведінки

Чи є проект «першопрохід-цем» на ринку?	Чи буде компанія шукати нових споживачів, або забирати існуючих у конкурентів?	Чи буде компанія копіювати основні характеристики товару конкурента, і які?	Стратегія конкурентної поведінки
Так.	Компанія буде шукати нових споживачів, але і, за потреби, буде намагатися забирати існуючих у конкурентів.	Компанія, за потреби, буде копіювати характеристики конкурентів.	Стратегія заняття конкурентної ніші.

Тепер розробимо стратегію позиціонування, що полягає у формуванні ринкової позиції (комплексу асоціацій), за яким споживачі мають ідентифікувати торгівельну марку/проект. Її зображено в таблиці 4.17

Таблиця 4.17 – Визначення стратегії позиціонування

Вимоги до товару цільової аудиторії	Базова стратегія розвитку	Ключові конкурентоспро-можні позиції власного стартап-проекту	Вибір асоціацій, які мають сформувати комплексну позицію власного проекту (три ключових)
Розпізнавання особи за голосом має бути точним. Система є невибагливою до ресурсів Система є щвидкою	Проведен-ня крупних оновлень (оптимі-зація розрахун-ків), створення додатков-ого функціона-лу.	Товар є іноваційним (в тренді) та дешевим у використанні порівняно з альтернативами	Швидкий, невибагливий до ресурсів та зручний доступ до приватного контенту з використанням голосу, додаткова система аутентифікації, програма працює в режимі онлайн.

4.5 Розроблення маркетингової програми стартап-проєкту

Сформуємо маркетингову концепцію товару, який отримає споживач. В таблиці 4.18 зображено результати попереднього аналізу конкурентоспроможності товару.

Таблиця 4.18 – Визначення ключових переваг концепції потенційного товару

№ п/п	Потреба	Вигода, яку пропонує товар	Ключові переваги перед конкурентами (існуючі або такі, що потрібно створити)
1.	Точне розпізнавання особи.	Точне розпізнавання особи за її голосом.	Доступність для компаній з невеликим капіталом.
2.	Невибагливість до апаратних ресурсів	Невибагливість до апаратних ресурсів клієнта	Доступність для компаній з невеликим капіталом.
3.	Швидкодія	Швидкодія системи	Більша швидкість

Надалі розробимо трирівневу маркетингову модель товару: уточнимо ідею продукту, його фізичні складові, особливості процесу його надання. Дана модель зображена в таблиці 4.19.

Таблиця 4.19 – Опис трьох рівнів моделі товару

Рівні товару	Сутність та складові
I. Товар за задумом	Програмний продукт – система голосової біометрії, яка дозволяє користувачу проходити аутентифікацію жо приватного контенту вебсайту за допомогою голосу.
II. Товар у реальному виконанні	Властивості / характеристики: 1. Можливість зареєструватися в систему за допомогою голосу 2. Можливість пройти аутентифікацію за допомогою голосу особи 3. Можливість пройти аутентифікацію іншим, альтернативним, спомобом
	Якість: програмний продукт пройшов всі етапі тестування та готовий до використання.
	Файл з розширенням “.ру”, віртуальне середовище.
	Марка: назва організації-розробника «YG», назва товару «BioM».

Продовження таблиці 4.19

Рівні товару	Сутність та складові
III. Товар із підкріпленням	Спеціаліст із впровадження встановлює ПЗ.
	Відділ розробки підтримує життєдіяльність ПЗ.
Захист програмного продукту буде організовано за допомогою ноу-хау.	

Тепер визначимо цінові межі, якими необхідно керуватись при встановленні ціни на потенційний товар, яке передбачає аналіз ціни на товари-аналоги або товари субститути, а також аналіз рівня доходів цільової групи споживачів. Аналіз проводився експертним методом і його результати зображено в таблиці 4.20.

Таблиця 4.20 – Визначення меж встановлення цін

Рівень цін на товари-замінники	Рівень цін на товари-аналоги	Рівень доходів цільової групи споживачів	Верхня та нижня межі встановлення ціни на товар/послугу
3000-5000 $/рік	5000-6000 $/рік	12000-50000 $/рік	Нижня межа – 3000 $/рік, верхня межа - 5000 $/рік

Надалі визначимо оптимальну систему збуту, в межах якого приймається рішення. Дану систему зображено в таблиці 4.21.

Таблиця 4.21 – Формування системи збуту

Специфіка закупівельної поведінки цільових клієнтів	Функції збуту, які має виконувати постачальник товару	Глибина каналу збуту	Оптимальна система збуту
Клієнт виплачує гроші на рік, тоді до нього приходить спеціаліст із впровадження інформаційних систем і встановлює ПЗ на комп'ютер клієнта.	Встановити програмний продукт на комп'ютери клієнтів.	Один посередник – спеціаліст по впровадженню інформаційних систем.	Канал збуту одного рівня.

Тепер розробимо концепцію маркетингових комунікацій, що спирається на попередньо обрану основу для позиціонування, визначену специфіку поведінки клієнтів. Дану концепцію зображено в таблиці 4.22.

Таблиця 4.22 – Концепція маркетингових комунікацій

Специфіка поведінки цільових клієнтів	Канали комунікацій, якими користуютьс я цільові клієнти	Ключові позиції, обрані для позиціонува-ння	Завдання рекламного повідомлення	Концепція рекламного звернення
Клієнт намагається знайти нові методи переслідування.	Мережа Інтернет, соціальні мережі, відео-портали.	Швидкодія, невибагливіст ь до апаратних ресурсів, іноваційність ПЗ, відносно невелика вартість ПЗ.	Продемонструва -ти швидкодію, дешевизну експлуатації, іноваційність та відносну невелику вартість ПЗ.	Показати можливість за невелику ціну зацікавити користувачі в свого вебсайту.

4.6 Висновки до розділу

В даному розділі було повністю виконано перший етап розроблення стартап-проекту, а саме, виконано маркетинговий аналіз стартап-проекту.

За допомогою нього можна сказати, що існує можливість ринкової комерціалізації проекту, адже на ринку програм систем біометрії наявний попит на системи голосової біометрії, до того ж рентабельність роботи є досить високою.

З огляду на потенційну групу клієнтів, а саме, малий бізнес, що має вебсайт з приватним контентом для своїх клієнтів, та іноваційність технології є великі перспективи впровадження даного програмного забезпечення.

Для ринкової реалізації проекту доцільно обрати таку альтернативу впровадження: створення MVP та впровадження його в невелику кількість веб-сайтів малих бізнесів.

ВИСНОВКИ

В цій роботі ми ознайомились з методами навчання з підкріпленням, розбрали те як вони діють, як їй застосовувати та перевірили на практиці їхню ефективність, вони майже не відсають від теоретичних методів, але є набагато гнучкішими, хоть і потребують багато часу і є складними у навчання.

Ми можемо побічити на рисунку 3.9, що результат мережі коливався біля результатів погонної кривої, деколи він був кращий, а деколи гірший. На рисунку 3.10 ми можемо побачити, що результат нашого алгоритму такий самий або гірший на 1-2 кроки ніж при паралельному переслідуванні. З рисунків 3.3 і 3.4 ми можемо зробити висовки, що гравець яким керує агоритм навчання з підкріпленням, діє дуже подібно до гравця, що діє найкращим способом.

При збільшенні кількості гравців ми зіткнулися з багатьма проблемами, гравці перестали діяти так точно, для навчання гравцям треба було набагато більше ітерацій та більша мережа, також з риунків 3.12 і 3.13 можна зробити висновок, що дуже важливо, щоб був великий масив різноманітних данних для навчання.

Також перевірили ефективність різних шумів для мережі DDPG, найефективнішим є параметричний шум.

Про методи навчання з підкріпленням в динамічних іграх можна зробити такі висновки:

- При докладанні зусиль, перебиранні різниз алгоритмів і параметрів для них, можна отримати результ близький до оптимального
- Вони потребують багато часу для навчання

- Можуть застосовуватись до невідомих середовищ і не потребують значних змін при зміні конфігурацій цього середовища

В роботі було досліджено як пришвидшити процес навчання гравця, зменшити затрати при досягнені оптимальної поведінки гравця. Було проведено дослідження використання алгоритмів в мультиагентних системах.

ПЕРЕЛІК ПОСИЛАНЬ

1. Желнин Ю.Н. Нелинейная задача преследования на плоскости. 1974. С. 85-94. URL: https://cyberleninka.ru/article/v/nelineynaya-igrovaya-zadacha-presledovaniya-na-ploskosti (дата звернення: 7.14.19).
2. Романников Д. ПРИМЕР РЕШЕНИЯ МИНИМАКСНОЙ ЗАДАЧИ ПРЕСЛЕДОВАНИЯ С ИСПОЛЬЗОВАНИЕМ НЕЙРОННЫХ СЕТЕЙ. 2018. URL: https://journals.nstu.ru/sbornik/download_article?id=17406 (дата звернення: 7.14.19).
3. Al-Talabi A. Learning in the Multi-Robot Pursuit Evasion Game. 2019. URL: https://curve.carleton.ca/system/files/etd/fdc82b11-054a-4132-94e0-65c92418e0eb/etd_pdf/b7f3b1aa6469622f9dfc93eac3c2a28f/al-talabi-learninginthemultirobotpursuitevasiongame.pdf (Last accessed: 7.14.19).
4. Baltes J., Park Y. Comparison of Several Machine Learning Techniques in Pursuit-Evasion Games. URL: https://www.researchgate.net/publication/220797707_Comparison_of_Several_Machine_Learning_Techniques_in_Pursuit-Evasion_Games/link/02bfe5142869e573bc000000/download (Last accessed: 7.14.19).
5. García M. Comparison of two methods for solving Markov Decision Processes in the persecution-evasion problem. URL: https://www.researchgate.net/profile/Michel_Garcia3/publication/268000728_Comparison_of_two_methods_for_solving_Markov_Decision_Processes_in_the_persecution-evasion_problem/links/5488938c0cf289302e30b5a6/Comparison-of-two-methods-for-solving-Markov-Decision-Processes-in-the-persecution-evasion-problem.pdf (Last accessed: 7.14.19).

6. Hado van Hasselt Reinforcement Learning in Continuous Action Spaces. 2007. URL: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.75.7658&rep=rep1&type=pdf (Last accessed: 7.14.19).
7. Hüttenrauch M. Deep Reinforcement Learning for Swarm Systems. 2019. URL: https://arxiv.org/pdf/1807.06613.pdf (Last accessed: 7.14.19).
8. Implementing Ornstein–Uhlenbeck. URL: https://math.stackexchange.com/questions/1287634/implementing-ornstein-uhlenbeck-in-matlab (Last accessed: 7.14.19).
9. Kyowoon L. Deep Reinforcement Learning in Continuous Action Spaces: a Case Study in the Game of Simulated Curling. URL: http://proceedings.mlr.press/v80/lee18b/lee18b.pdf (Last accessed: 7.14.19).
10. Lillicrap T.P. CONTINUOUS CONTROL WITH DEEP REINFORCEMENT LEARNING. 2016. URL: https://arxiv.org/pdf/1509.02971.pdf (Last accessed: 7.14.19).
11. Mnih V. Asynchronous Methods for Deep Reinforcement Learning. 2016. URL: https://arxiv.org/pdf/1602.01783.pdf (Last accessed: 7.14.19).
12. Plappert M. PARAMETER SPACE NOISE FOR EXPLORATION. 2018. URL: https://arxiv.org/pdf/1706.01905.pdf. (Last accessed: 7.14.19).

ДОДАТОК А

```
import numpy as np
import random
from multiagent.policy import Policy

class StayAgent(Policy):
    def __init__(self, env, agent_index):
        super(StayAgent, self).__init__(agent_index)
        self.env = env
        self.reset()

    def action(self, obs):
        return np.concatenate([self.u, np.zeros(self.env.world.dim_c)])

    def reset(self):
        if self.env.discrete_action_input:
            self.u = random.randint(1, 4)
        else:
            self.u = np.zeros(2)  # 5-d because of no-move action
            self.u[0] = random.random() * 2 - 1
            self.u[1] = random.random() * 2 - 1

class VectorAgent(Policy):
    def __init__(self, env, agent_index, advertisal_agent_index):
        super(VectorAgent, self).__init__(agent_index)
        self.advertisal_agent_index = advertisal_agent_index
        self.env = env
        self.agent_index = agent_index

    "action return speed the fastest to get to the point if we know velocity"
    def action(self, obs):
        # ignore observation and just act based on keyboard events
        u = np.zeros(2)
        pos1 = self.env.agents[self.agent_index].state.p_pos
        pos2 = self.env.agents[self.advertisal_agent_index].state.p_pos
        xd = pos2[1] - pos1[1]
        yd = pos2[0] - pos1[0]
        distance = np.sqrt(xd * xd + yd * yd)
        if distance == 0:
            return u
        v1 = self.env.agents[self.advertisal_agent_index].max_speed
        v2 = self.env.agents[self.agent_index].max_speed
        scalar = xd * self.env.agents[self.advertisal_agent_index].state.p_vel[1] + yd * \
                 self.env.agents[self.advertisal_agent_index].state.p_vel[0]
        t = distance / (np.sqrt(v1 * v1 + v2 * v2 - 2 * v2 * scalar / distance))
        pointX = pos2[1] + t * self.env.agents[self.advertisal_agent_index].state.p_vel[1]
        pointY = pos2[0] + t * self.env.agents[self.advertisal_agent_index].state.p_vel[0]
        pointXD = pointX - pos1[1]
        pointYD = pointY - pos1[0]
        distanceToPoint = np.sqrt(pointXD * pointXD + pointYD * pointYD)
        xd = pointXD / distanceToPoint
        yd = pointYD / distanceToPoint
        u[1] = xd
```

```
        u[0] = yd
        return np.concatenate([u, np.zeros(self.env.world.dim_c)])

class ToPointAgent(Policy):
    def __init__(self, env, agent_index, advertisal_agent_index):
        super(ToPointAgent, self).__init__(agent_index)
        self.advertisal_agent_index = advertisal_agent_index
        self.env = env
        self.agent_index = agent_index

    "action return speed the fastest to get to the point if we don't know velocity"
    def action(self, obs):
        # ignore observation and just act based on keyboard events
        u = np.zeros(2)
        pos1 = self.env.agents[self.agent_index].state.p_pos
        pos2 = self.env.agents[self.advertisal_agent_index].state.p_pos
        xd = pos2[1] - pos1[1]
        yd = pos2[0] - pos1[0]
        k = np.sqrt(xd * xd + yd * yd)
        xd = xd / k
        yd = yd / k
        u[1] = xd
        u[0] = yd
        return np.concatenate([u, np.zeros(self.env.world.dim_c)])

class ToPointsAgent(Policy):
    def __init__(self, env, agent_index, advertisal_agent_indexex):
        super(ToPointsAgent, self).__init__(agent_index)
        self.advertisal_agent_indexex = advertisal_agent_indexex
        self.env = env
        self.caught_agents = set()

    def action(self, obs):
        agent = self.env.agents[self.agent_index]
        for advertisal_agent_index in self.advertisal_agent_indexex:
            if advertisal_agent_index in self.caught_agents:
                continue
            if self.env._is_collision(self.env.agents[self.agent_index], self.env.agents[advertisal_agent_index]):
                self.caught_agents.add(advertisal_agent_index)
        min_distance = 1e10
        closest_agent = self.advertisal_agent_indexex[0]
        for advertisal_agent_index in self.advertisal_agent_indexex:
            if advertisal_agent_index in self.caught_agents:
                continue
            if self.env._is_collision(self.env.agents[self.agent_index], self.env.agents[advertisal_agent_index]):
                self.caught_agents.add(advertisal_agent_index)
            distance = np.sqrt(
                np.sum(np.square(agent.state.p_pos - self.env.agents[advertisal_agent_index].state.p_pos)))
            if distance < min_distance:
                min_distance = distance
                closest_agent = advertisal_agent_index
        u = np.zeros(2)
        pos1 = agent.state.p_pos
        pos2 = self.env.agents[closest_agent].state.p_pos
        xd = pos2[1] - pos1[1]
        yd = pos2[0] - pos1[0]
```

```
        k = np.sqrt(xd * xd + yd * yd)
        xd = xd / k
        yd = yd / k
        u[1] = xd
        u[0] = yd
        return np.concatenate([u, np.zeros(self.env.world.dim_c)])

    def reset(self):
        self.caught_agents = set()

class ToPointsAgentWithVelocity(Policy):
    def __init__(self, env, agent_index, advertisal_agent_indexex):
        super(ToPointsAgentWithVelocity, self).__init__(agent_index)
        self.advertisal_agent_indexex = advertisal_agent_indexex
        self.env = env
        self.caught_agents = set()

    def action(self, obs):
        # ignore observation and just act based on keyboard events
        agent = self.env.agents[self.agent_index]
        # check if catch
        for advertisal_agent_index in self.advertisal_agent_indexex:
            if advertisal_agent_index in self.caught_agents:
                continue
            if self.env._is_collision(self.env.agents[self.agent_index], self.env.agents[advertisal_agent_index]):
                self.caught_agents.add(advertisal_agent_index)
        min_distance = 1e10
        closest_agent = self.advertisal_agent_indexex[0]
        for advertisal_agent_index in self.advertisal_agent_indexex:
            if advertisal_agent_index in self.caught_agents:
                continue
            if self.env._is_collision(self.env.agents[self.agent_index], self.env.agents[advertisal_agent_index]):
                self.caught_agents.add(advertisal_agent_index)
            distance = np.sqrt(
                np.sum(np.square(agent.state.p_pos - self.env.agents[advertisal_agent_index].state.p_pos)))
            if distance < min_distance:
                min_distance = distance
                closest_agent = advertisal_agent_index
        u = np.zeros(2)
        pos1 = agent.state.p_pos
        pos2 = self.env.agents[closest_agent].state.p_pos
        xd = pos2[1] - pos1[1]
        yd = pos2[0] - pos1[0]
        k = np.sqrt(xd * xd + yd * yd)
        xd = xd / k
        yd = yd / k
        u[1] = xd
        u[0] = yd
        pos1 = self.env.agents[self.agent_index].state.p_pos
        pos2 = self.env.agents[closest_agent].state.p_pos
        xd = pos2[1] - pos1[1]
        yd = pos2[0] - pos1[0]
        distance = np.sqrt(xd * xd + yd * yd)
        if distance == 0:
            return u
        v1 = self.env.agents[closest_agent].max_speed
```

```
                v2 = self.env.agents[self.agent_index].max_speed
                scalar = xd * self.env.agents[closest_agent].state.p_vel[1] + yd * \
                        self.env.agents[closest_agent].state.p_vel[0]
                t = distance / (np.sqrt(v1 * v1 + v2 * v2 - 2 * v2 * scalar / distance))
                pointX = pos2[1] + t * self.env.agents[closest_agent].state.p_vel[1]
                pointY = pos2[0] + t * self.env.agents[closest_agent].state.p_vel[0]
                pointXD = pointX - pos1[1]
                pointYD = pointY - pos1[0]
                distanceToPoint = np.sqrt(pointXD * pointXD + pointYD * pointYD)
                xd = pointXD / distanceToPoint
                yd = pointYD / distanceToPoint
                u[1] = xd
                u[0] = yd
                return np.concatenate([u, np.zeros(self.env.world.dim_c)])

            def reset(self):
                self.caught_agents = set()

        class EvasionAgentWithVelocity(Policy):
            def __init__(self, env, agent_index, adversarial_agent_index, good_agent_index):
                super(EvasionAgentWithVelocity, self).__init__(agent_index)
                self.good_agent_index = good_agent_index
                self.adversarial_agent_index = adversarial_agent_index
                self.env = env
                self.is_caught = False
                self.is_caught_another_agent = False

            def action(self, obs):
                u = np.zeros(2)
                agent = self.env.agents[self.agent_index]
                if not self.is_caught:
                    if self.env._is_collision(self.env.agents[self.agent_index],
self.env.agents[self.adversarial_agent_index]):
                        self.is_caught = True
                if not self.is_caught_another_agent:
                    if self.env._is_collision(self.env.agents[self.good_agent_index],
                                              self.env.agents[self.adversarial_agent_index]):
                        self.is_caught_another_agent = True
                if self.is_caught:
                    u = np.zeros(2)
                    return np.concatenate([u, np.zeros(self.env.world.dim_c)])
                agent_is_closest = True
                import utilities
                distance = utilities.distance(self.env.agents[self.agent_index].state.p_pos,
                                              self.env.agents[self.adversarial_agent_index].state.p_pos)
                distance_another = utilities.distance(self.env.agents[self.good_agent_index].state.p_pos,
                                                      self.env.agents[self.adversarial_agent_index].state.p_pos)
                if distance_another < distance and not self.is_caught_another_agent:
                    agent_is_closest = False
                if agent_is_closest:
                    pos1 = agent.state.p_pos
                    pos2 = self.env.agents[self.adversarial_agent_index].state.p_pos
                    xd = pos2[1] - pos1[1]
                    yd = pos2[0] - pos1[0]
                    k = np.sqrt(xd * xd + yd * yd)
                    xd = -xd / k
```

```
                    yd = -yd / k
                else:
                    pos1 = self.env.agents[self.adversarial_agent_index].state.p_pos
                    pos2 = self.env.agents[self.good_agent_index].state.p_pos
                    xd = pos2[1] - pos1[1]
                    yd = pos2[0] - pos1[0]
                    distance = np.sqrt(xd * xd + yd * yd)
                    if distance == 0:
                        return u
                    v1 = self.env.agents[self.good_agent_index].max_speed
                    v2 = self.env.agents[self.adversarial_agent_index].max_speed
                    scalar = xd * self.env.agents[self.good_agent_index].state.p_vel[1] + yd * \
                            self.env.agents[self.good_agent_index].state.p_vel[0]
                                t = distance / (np.sqrt(v1 * v1 + v2 * v2 - 2 * v2 * scalar / distance))
                    pointX = pos2[1] + t * self.env.agents[self.good_agent_index].state.p_vel[1]
                    pointY = pos2[0] + t * self.env.agents[self.good_agent_index].state.p_vel[0]
                    pointXD = pointX - self.env.agents[self.agent_index].state.p_pos[1]
                    pointYD = pointY - self.env.agents[self.agent_index].state.p_pos[0]
                    distanceToPoint = np.sqrt(pointXD * pointXD + pointYD * pointYD)
                    xd = -pointXD / distanceToPoint
                    yd = -pointYD / distanceToPoint
                u[1] = xd
                u[0] = yd
                return np.concatenate([u, np.zeros(self.env.world.dim_c)])

            def reset(self):
                self.is_caught = False
                self.is_caught_another_agent = False

import numpy as np
from baselines.ddpg.memory import Memory
from models import Actor, Critic
from baselines.ddpg.ddpg import DDPG
from baselines.ddpg.noise import AdaptiveParamNoiseSpec, NormalActionNoise, OrnsteinUhlenbeckActionNoise
from baselines import logger
import tensorflow as tf
from multiagent.policy import Policy

class DDPGAgent(Policy):
    def __init__(self, env, agent_index, sess, action_range=(-1., 1.), reward_scale=0.1, critic_l2_reg=1e-2,
                 actor_lr=1e-4, critic_lr=1e-3, popart=False, gamma=0.975, clip_norm=10, batch_size=64,
                 memory_size=1e6, tau=0.01,
                 normalize_returns=False, normalize_observations=False,
                 noise_type="adaptive-param_0.1",  layer_norm=True,  nb_layers=2,  nb_neurons=64,activation='tanh',
**network_kwargs):
        super(DDPGAgent, self).__init__(agent_index)
        # self.sess = sess
        self.nb_actions = env.action_space[agent_index].n
        print('agent action_space ' + str(env.action_space[agent_index].n))
        self.state_size = env.observation_space[agent_index].shape
        self.action_range = action_range
        with tf.variable_scope('ddpg_' + str(agent_index)):
            critic = Critic(name='critic_' + str(agent_index), layer_norm=layer_norm,
                            nb_layers=nb_layers, nb_neurons=nb_neurons)
            actor = Actor(self.nb_actions, name='actor_' + str(agent_index),
                          layer_norm=layer_norm, nb_neurons=nb_neurons, activation=activation)
```

```
            memory = Memory(limit=int(memory_size), action_shape=(self.nb_actions,),
                        observation_shape=self.state_size)
            action_noise = None
            param_noise = None
            if noise_type is not None:
                for current_noise_type in noise_type.split(','):
                    current_noise_type = current_noise_type.strip()
                    if current_noise_type == 'none':
                        pass
                    elif 'adaptive-param' in current_noise_type:
                        _, stddev = current_noise_type.split('_')
                        param_noise = AdaptiveParamNoiseSpec(initial_stddev=float(stddev),
                                                             desired_action_stddev=float(stddev))
                    elif 'normal' in current_noise_type:
                        _, stddev = current_noise_type.split('_')
                        action_noise = NormalActionNoise(mu=np.zeros(self.nb_actions),
                                                         sigma=float(stddev) * np.ones(self.nb_actions))
                    elif 'ou' in current_noise_type:
                        _, stddev = current_noise_type.split('_')
                        action_noise = OrnsteinUhlenbeckActionNoise(mu=np.zeros(self.nb_actions),
                                                                    sigma=float(stddev) * np.ones(self.nb_actions), dt=env.world.dt, theta=0.1)
                    else:
                        raise RuntimeError('unknown noise type "{}"'.format(current_noise_type))

            self.agent = DDPG(actor, critic, memory, self.state_size, (self.nb_actions,),
                              action_range=self.action_range,
                              gamma=gamma, tau=tau, normalize_returns=normalize_returns,
                              normalize_observations=normalize_observations,
                              batch_size=batch_size, action_noise=action_noise, param_noise=param_noise,
                              critic_l2_reg=critic_l2_reg,
                              actor_lr=actor_lr, critic_lr=critic_lr, enable_popart=popart, clip_norm=clip_norm,
                              reward_scale=reward_scale)
            logger.info('Using agent with the following configuration:')
            logger.info(str(self.agent.__dict__.items()))
        self.agent.initialize(sess)
        self.agent.reset()

    def action(self, obs, apply_noise=False, compute_Q=False):
        if compute_Q:
            return self.agent.pi(obs, apply_noise=apply_noise, compute_Q=compute_Q)
        else:
            return self.agent.pi(obs, apply_noise=apply_noise, compute_Q=compute_Q)[0]

    def reset(self):
        return self.agent.reset()

import argparse
import numpy as np
import os
import utilities
from multiagent.environment import MultiAgentEnv
from multiagent.scenarios.open_1_1_pursuit_know_vel import Scenario
from multiagent.theoretical_agents import StayAgent, ToPointAgent,StayForceAgent, VectorAgent
from multiagent.ddpg_agent import DDPGAgent
import baselines.common.tf_util as U
import tensorflow as tf
```

```
import evaluate_models_1_1 as evaluate_models
parser = argparse.ArgumentParser()
parser.add_argument('--actor_lr', default=1e-5, type=float)
parser.add_argument('--critic_lr', default=1e-4, type=float)
parser.add_argument('--episodes', default=10000, type=int)
parser.add_argument('--evaluate_every_n_episodes', default=500, type=int)
parser.add_argument('--max_steps', default=200, type=int)
parser.add_argument('--batch_size', default=1024, type=int)
parser.add_argument('--nb_train_steps', default=16, type=int)
parser.add_argument('--param_noise_adaption_interval', default=8, type=int)
parser.add_argument('--experiment_prefix', default='/statistics/', type=str)
parser.add_argument('--noise_type', default='ou_0.2', type=str)
parser.add_argument('--nb_layers', default=2, type=str)
parser.add_argument('--nb_neurons', default=64, type=str)
parser.add_argument('--save_every_n_episodes', default=500, type=int)
parser.add_argument('--load_weights', default=False)
parser.add_argument('--render', default=False, action="store_true")
parser.add_argument('--memory_size', default=2e5, type=int)
args = parser.parse_args()

def train(scenario):
    path_to_save = 'models/' + scenario.__module__.split('.')[-1] + '/ddpg'
    train_n = 4
    if not os.path.exists(path_to_save):
        os.makedirs(path_to_save)
    world = scenario.make_world()
    env = MultiAgentEnv(world, reset_callback=scenario.reset_world, reward_callback=scenario.reward,
                        observation_callback=scenario.observation, info_callback=None,
                        done_callback=scenario.done, collision_callback=scenario.is_collision,
                        shared_viewer=True, )
    evaluator = evaluate_models.Evaluator(args, scenario, save=scenario.name + '/' + str(train_n))

    with U.single_threaded_session() as sess:
        simple_agents = [StayAgent(env, 1)] #good agent
        agents_with_nn = [
            DDPGAgent(env, 0, sess, batch_size=args.batch_size, memory_size=args.memory_size,
                      noise_type=args.noise_type,
                      # good agent
                      actor_lr=args.actor_lr, critic_lr=args.critic_lr, layer_norm=True,
                      nb_layers=args.nb_layers, nb_neurons=args.nb_neurons)
        ]
        policies = [agents_with_nn[0], simple_agents[0]]
        saver = tf.train.Saver()
        if args.load_weights:
            saver.restore(sess,
                          'models/' + scenario.name + '/ddpg/model')
        sess.graph.finalize()
        statistics_header = ["episode"]
        statistics_header.append("steps")
        statistics_header.extend(["reward_{}".format(i) for i in range(env.n)])
        statistics_header.extend(["q_{}".format(i) for i in range(env.n)])
        statistics = utilities.Time_Series_Statistics_Store(
            statistics_header)

        for episode in range(args.episodes):
            if episode % 500 == 0:
```

```
            print('episode ' + str(episode))
        for agent in policies:
            agent.reset()
        states = env.reset()
        step = 0
        while True:
            episode_q = np.zeros(env.n)
            episode_rewards = np.zeros(env.n)
            step += 1
            env_done = False
            if args.render:
                env.render()
            actions = [None for _ in range(len(world.policy_agents))]
            for agent in simple_agents:
                actions[agent.agent_index] = (agent.action(states[agent.agent_index]))
                episode_q[0] += 0
            for agent in agents_with_nn:
                action, q = agent.action(states[agent.agent_index], apply_noise=True, compute_Q=True)
                actions[agent.agent_index] = action
                episode_q[agent.agent_index] += q
            states_next, rewards, done, info = env.step(actions)
            episode_rewards += rewards
            for agent in agents_with_nn:
                agent.agent.store_transition(states[agent.agent_index], actions[agent.agent_index],
                                             rewards[agent.agent_index], states_next[agent.agent_index],
                                             done[agent.agent_index])
            if step >= args.max_steps:
                env_done = True
            for agent in agents_with_nn:
                if done[agent.agent_index]:
                    env_done = True
            states = states_next
            if env_done:
                episode_rewards = episode_rewards / step
                episode_losses = episode_q / step
                statistic = [episode]
                statistic.append(step)
                statistic.extend([episode_rewards[i] for i in range(env.n)])
                statistic.extend([episode_q[i] for i in range(env.n)])
                statistics.add_statistics(statistic)
                break
        for t_train in range(args.nb_train_steps):
            for agent in agents_with_nn:
                if agent.agent.memory.nb_entries >= args.batch_size:
                    if episode % args.param_noise_adaption_interval == 0:
                        distance = agent.agent.adapt_param_noise()
                    cl, al = agent.agent.train()
                    agent.agent.update_target_net()
        if episode % args.save_every_n_episodes == 0:
            saver.save(sess, 'models/' + scenario.__module__.split('.')[-1] + '/ddpg/model')
        if args.evaluate_every_n_episodes != 0 and episode % args.evaluate_every_n_episodes == 0:
            statistics.dump("{}_{}.csv".format(
                args.experiment_prefix + scenario.__module__.split('.')[-1], episode))
            evaluator.evaluate(env, policies, episode)
    saver.save(sess, 'models/' + scenario.__module__.split('.')[-1] + '/ddpg/model')
```