**КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ  
ІМЕНІ ТАРАСА ШЕВЧЕНКА**Факультет комп’ютерних наук та кібернетики   
Кафедра інтелектуальних програмних систем

**Кваліфікаційна робота   
на здобуття ступеня бакалавра**

за спеціальністю 121 Інженерія програмного забезпечення

на тему:

**ЗАСТОСУВАННЯ МЕТОДІВ НАВЧАННЯ З ПІДКРІПЛЕННЯМ ДЛЯ НАВЧАННЯ АГЕНТІВ ІГРОВИХ СЕРЕДОВИЩ**

Виконав студент 4-го курсу   
Давидько Анатолій Євгенович \_\_\_\_\_\_\_\_\_\_\_\_

(підпис)

Науковий керівник:   
кандидат фіз.-мат. наук   
Ліндер Ярослав Миколайович \_\_\_\_\_\_\_\_\_\_\_\_

(підпис)

Засвідчую, що в цій роботі немає запозичень   
з праць інших авторів без відповідних   
посилань.

Студент \_\_\_\_\_\_\_\_\_\_\_\_

(підпис)

Роботу розглянуто й допущено до захисту   
на засіданні кафедри інтелектуальних програмних систем   
«15» червня 2020 р.,   
протокол № 13   
Завідувач кафедри   
О. І. Провотар \_\_\_\_\_\_\_\_\_\_\_\_

(підпис)

Київ – 2020

**РЕФЕРАТ**

Дипломна робота: 67 сторінок, 23 рисунка, 18 джерел та 1 додаток.

Ключові слова: навчання з підкріпленням, нагороди, оптимальна стратегія, функції корисності, TD-методи, машинне навчання, навчання з вчителем, глибоке навчання, нейронні мережі, диференційні ігри, гра переслідування.

Об’єкт дослідження: застосування методів машинного навчання з підкріпленням для пошуку оптимальної стратегії у диференційних іграх.

Мета роботи: аналіз можливостей методів машинного навчання з підкріпленням у диференційній грі переслідування, дослідження ефективності їх роботи в конкретних умовах.

Методи та інструменти розроблення: Python 3.7, бібліотека нейронних мереж Keras, машинного навчання та аналізу даних Scikit-learn, бібліотека програмування ігрових середовищ Pygame.

Рекомендації щодо використання роботи: для дослідження алгоритмів машинного навчання на середовищах диференціальних ігор.

Сфера застосування: робототехніка, комп’ютерні ігри, військові розробки, симуляція природніх процесів, системи оптимального керування ресурсами, моделювання поведінки клієнтів бізнесу.

Значимість роботи: наведено порівняння методів машинного навчання з підкріпленням у диференційній грі переслідування, запропонований підхід до реалізації параметрів середовища.

Висновки: були проаналізовані методи машинного навчання з підкріпленням у диференційній грі переслідування, їх ефективність розв’язання задачі, проведено порівняння їх властивостей, описані необхідні умови середовища для ефективного вирішення задачі.

ЗМІСТ

[СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАЧЕННЯ 5](#_Toc43556768)

[ВСТУП 6](#_Toc43556769)

[РОЗДІЛ 1 Навчання з підкріпленням 9](#_Toc43556770)

[1.1 Основи та базові поняття 9](#_Toc43556771)

[1.2 Проблема дослідження і експлуатації стратегії 11](#_Toc43556772)

[1.3 Фундаментальні методи 14](#_Toc43556773)

[1.3.1 Динамічне програмування 14](#_Toc43556774)

[1.3.2 Методи Монте-Карло 17](#_Toc43556775)

[1.3.3 TD-методи 18](#_Toc43556776)

[1.4 Випробування алгоритмів на прикладі середовища 20](#_Toc43556777)

[РОЗДІЛ 2 Основи глибокого навчання та нейронних мереж 24](#_Toc43556778)

[2.1 Визначення та постановка задачі машинного навчання 24](#_Toc43556779)

[2.2 Визначення навчання з вчителем та глибокого навчання 24](#_Toc43556780)

[2.3 Нейронна мережа як математична модель глибокого навчання 27](#_Toc43556781)

[2.3.1 Визначення нейронної мережі та її компонент 27](#_Toc43556782)

[2.3.2 Компоненти навчання нейронних мереж 32](#_Toc43556783)

[2.3.3 Метод зворотного поширення помилки 35](#_Toc43556784)

[2.3.4 Процес навчання нейронної мережі 37](#_Toc43556785)

[РОЗДІЛ 3 Навчання з підкріпленням з елементами глибокого навчання 40](#_Toc43556786)

[РОЗДІЛ 4 Диференціальні ігри та ігрові середовища 43](#_Toc43556787)

[РОЗДІЛ 5 Аналіз ефективності методів навчання з підкріпленням для гри переслідування 47](#_Toc43556788)

[5.1 Опис програмної реалізації середовища 47](#_Toc43556789)

[5.2 Опис реалізації алгоритмів навчання 51](#_Toc43556790)

[5.3 Метрики 51](#_Toc43556791)

[5.4 Визначення умов експерименту 52](#_Toc43556792)

[5.5 Результати експериментів 54](#_Toc43556793)

[1. Працездатність хижака 54](#_Toc43556794)

[2. Працездатність жертви 55](#_Toc43556795)

[3. Ефективності агентів, один алгоритм, Нагорода-, Стан-1 56](#_Toc43556796)

[4. Ефективності агентів, один алгоритм, Нагорода-, Стан-2 58](#_Toc43556797)

[5. Ефективності агентів, один алгоритм, Нагорода+, Стан-1 58](#_Toc43556798)

[6. Ефективності агентів, один алгоритм, Нагорода+, Стан-2 58](#_Toc43556799)

[7. Ефективності агентів, різні алгоритми, Нагорода+, Стан-1 59](#_Toc43556800)

[8. Ефективності агентів, різні алгоритми, Нагорода+, Стан-2 60](#_Toc43556801)

[ВИСНОВКИ 61](#_Toc43556802)

[ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ 62](#_Toc43556803)

[ДОДАТОК А. Програмний код агентів середовища 64](#_Toc43556804)

# СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАЧЕННЯ

МППР – марківський процес прийняття рішень

МК – Монте-Карло

TD-методи – методи, основані на часових різницях

API – програмний інтерфейс застосунку

ReLU – функція лінійної ректифікації

DQN – Deep Q-learning

A2C – Advantage Actor-Critic

# ВСТУП

На сьогодні галузь машинного навчання та всі її підрозділи набули небувалий розвиток за останні десять років. Науковці розвинули технології нейронних мереж при навчанні з вчителем та інших розділів машинного навчання. Отримані результати дали можливість успішно вирішувати задачі обробки зображень та розпізнання образів [6], розпізнання людської мови [5], розпізнання музичних композицій [7] та дослідження з метою створення повністю автономного транспорту. Також алгоритми машинного навчання використовуються в сфері бізнесу. Наприклад, при аналізі даних користувачів для формування таргетованої (цілеспрямованої) реклами, більш глибокої автоматизації внутрішніх бізнес-процесів [8], тощо. Відомі навіть приклади використання алгоритмів машинного навчання в політичній діяльності. Аналіз та дослідження даних соціальних мереж використовувались для визначення думки громадськості під час президентських перегонів в США (діяльність Cambridge Analytica).

Науковцями продовжуються дослідження в сфері машинного навчання, в тому числі в такому підрозділі як навчання з підкріпленням (Reinforcement Learning). На відміну від задач, розв’язуваних іншими підрозділами машинного навчання (задачі регресії, класифікації чи кластеризації), навчання з підкріпленням має на меті вирішити більш складну та широку задачу. А саме – пошук оптимальної поведінки агента в певному середовищі, що є більш близьким до пошуку рішень проблем в реальному світі [1]. Цей розділ машинного навчання має спільну рису з навчанням з учителем, яка полягає в сторонній оцінці прийнятого рішення. Його методи також використовуються в бізнес сфері – системи оптимального керування ресурсами, моделювання поведінки потенційних клієнтів. Методи навчання з підкріпленням застосовуються в більш специфічних та наукових цілях – робототехніка чи симуляція певних складних природніх процесів [8]. Проте суттєва різниця між навчанням з підкріпленням та іншими розділами машинного навчання проявляється у випадках, коли необхідно знайти оптимальну поведінку в мультиагентних середовищах [16]. Такі середовища можна виразити в термінах теорії ігор. Кажуть, що процес, у якому існує певна кількість суб’єктів, які взаємодіють з середовищем та одним з одним, називається грою. Гра може бути кооперативною чи з нульовою сумою (протистояння). Класичним прикладом мультиагентного середовища є диференціальна гра переслідування, яка зазвичай описує рух агентів диференціальними рівняннями. Але ця теорія слабо застосовна в великих масштабних середовищах, де існує велика кількість можливих станів, та у випадках, коли не існує точного алгоритму пошуку оптимальної стратегії. Саме для таких випадків застосовують методи навчання з підкріпленням. Можливість ефективно розв’язувати такі задачі була продемонстрована в 2014 році науковцями компанії Deep Mind, які реалізували алгоритми для успішного проходження ігор Atari. Згодом вони створили алгоритми, які змогли перемогти чемпіонів гри у шахи та Го (програма AlphaGo). Те саме вдалося зробити на прикладі ігор жанру стратегій реального часу (StarCraft 2), у яких як і в реальному світі складне і масштабне середовище. На сьогодні вчені продовжують дослідження галузі машинного навчання, а програмні інженери намагаються впроваджувати їх напрацювання для вирішення реальних проблем.

Ця робота присвячена застосуванню методів навчання з підкріпленням для пошуку оптимальної стратегії у диференційних іграх. Для цього будуть використані та представлені середовища Open AI Gym [18] та спеціальне середовище диференціальної гри переслідування, агенти яких названі хижаком і жертвою. Будуть розглянуті основні відомі методи, як Q-learning, Deep Q-learning (DQN), Autor-Critic Algorithm (A2C), тощо.

**Метою** цієї роботи є аналіз можливостей методів машинного навчання з підкріпленням у диференційній грі переслідування, дослідження ефективності їх роботи в конкретних умовах.

**Об’єктом** даної роботи є методи машинного навчання з підкріпленням в умовах мультиагентного середовища.

**Предмет** цієї роботи – застосування алгоритмів машинного навчання з підкріпленням.

Для досягнення визначеної мети необхідно виконати наступні **завдання**:

1. Розглянути базові поняття навчання з підкріпленням та розвиток галузі до сучасних методів.
2. Розглянути мультиагентні середовища та проаналізувати оптимальну поведінку агентів.
3. Проаналізувати очікувані результати роботи алгоритмів та необхідні їх налаштування і конфігурації.
4. Експериментально перевірити та ефективність алгоритмів на заданому середовищі.

# РОЗДІЛ 1 Навчання з підкріпленням

## Основи та базові поняття

Навчання з підкріпленням – це такий процес навчання, у якому обов’язково фігурують такі сутності як агент і середовище. В даному випадку агент – це об’єкт, задача якого в залежності від наданого середовищем стану прийняти рішення яка дія з усіх доступних призведе до отримання якомога більшої винагороди. В той же час середовище – це об’єкт, який отримує інформацію від агента щодо обраної дії та відповідно до неї визначає наступний стан агенту та оцінює скалярною величиною обрану дію в заданому стані. Ця оцінка і є винагородою, яка разом з новим станом передається агентові для подальшого отримання від нього нової дії. Зазвичай цей процес триває ітеративно до того часу, поки агент не перейде в фінішний стан, який, в залежності від апріорної інформації про середовище, може бути бажаним результатом його роботи або навпаки – ситуацією яку необхідно уникати.

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 1.1 – Основні елементи навчання з підкріпленням

Наявність оцінки дії середовищем робить цей процес схожим з навчанням з вчителем, але відмінність полягає в тому, що система не надає конкретні приклади того, як ефективно діяти агенту в конкретному випадку. Особливо це стосується проміжних ситуацій, коли нагорода ще не отримана, але необхідна послідовність дій була виконана і при цьому невідомо точно яка само поведінка є оптимальною. Задачею навчання з підкріпленням є знаходження такої поведінки агенту (стратегії), яка призведе до отримання ним максимальної кількості винагороди за гру.

Формально цю проблему представляють у вигляді *марківського процесу прийняття рішень* (МППР) [1], який описується такими ключовими компонентами як множина станів , множина дій , функція переходу – ймовірність того, що дія в стані приведе до стану – та функція нагороди .

Изображение выглядит как рисунок

Автоматически созданное описание

Рисунок 1.2 – Приклад МППР

Вирішити задачу навчання з підкріпленням у термінах МППР означає визначити таку функцію переходу (стратегію) яка буде давати сумарно найбільшу винагороду.

Стратегію агента в навчанні з підкріпленням в загальному випадку визначають як відображення множини станів в множину дій. Вона може бути реалізована різними підходами в залежності від алгоритму навчання і заданого середовища. Однак будь-який агент повинен використовувати надану нагороду для зміни (покращення) своєї стратегії, які також визначаються заданим алгоритмом навчання. Деякі алгоритми навчання безпосередньо генерують дію від заданого стану, але більшість алгоритмів, про яких мова буде далі, виконують оцінку кожній дії. Такі алгоритми називають заснованими на оцінці (Value based). Для них характерні функції корисності, які визначають наскільки дана ситуація (стан) чи дія є корисними. Функція корисності заданого стану позначають , а функцію корисності дії в стані позначають . Під словом «користь» мають на увазі наскільки багато агент може отримати винагороду в перспективі і даному випадку вона є похідним поняттям від нагороди. Алгоритми засновані на оцінці отримують оцінку кожній дії від функції корисності і згідно цієї інформації приймають рішення про те, яку дію необхідно виконати. Отримані нагороди безпосередньо визначають зміни функцій корисності.

## Проблема дослідження і експлуатації стратегії

Навчання з підкріпленням в своїй суті має базову проблему, пов’язану з тим, яким чином необхідно діяти агенту для того, щоб в один і той самий час виконувати найефективнішу дію згідно з його стратегією та досліджувати середовище для виявлення більш ефективних дій. При вже наявній ефективній стратегії для вирішення задачі агент в конкретних умовах має виконувати тільки ефективну дію. Такий підхід називається експлуатацією стратегії. Однак коли процес навчання тільки починається, то такої стратегії в нього ще немає і її необхідно сформувати. Формування стратегії відбувається при виконанні будь-яких дій, особливо найменш вживаних, та запам’ятовуванні результатів, що є дослідженням стратегії. Такий підхід не дає можливості отримати максимальну винагороду. В результаті маємо проблему, коли ми не можемо вибрати один із підходів. Це і є проблема дослідження і експлуатації стратегії. Всі існуючі рішення так чи інакше представляють компроміс в цьому виборі і ділять всі можливі ситуації вибору на ті, коли вибирається експлуатація, і ті, коли вибирається дослідження.

Канонічним прикладом гри на якій демонструється ця проблема і методи її вирішення є гра «багаторукий бандит» [4]. Розглянемо ігровий автомат у якому є рук (важелів), які можна повернути. Кожен важіль має певну ймовірність того, що якщо його повернути, то гравець отримає приз (нагорода = 1). В іншому випадку нічого не відбудеться (нагорода = 0). Формально гру можна описати так: в – множина дій, – функція ймовірності отримати винагороду від вибору дії . Поки будемо вважати, що автомат не має внутрішнього стану, тому його поведінка не буде змінюватися з часом. Для того, щоб гравець міг діяти, йому необхідна стратегія, яку для цієї гри будемо виражатися функцією , яка буде визначатися наступним чином: для кожної дії ми будем рахувати суму всіх нагород, котрі були отримані агентом після виконання поточної дії, і оптимальною дією буде та, у якої ця сума найбільша. Тобто сума є нашою функцією корисності. Із заданого визначення гри ми можемо побачити, що це такий випадок гри, у якій існує лише єдиний стан у який система повертається після виконання будь якої дії. Таке спрощення дає нам можливість краще побачити проблему дослідження і експлуатації.

Перш за все розглянемо стратегію, яка полягає у постійному виборі оптимальної дії. Така стратегія називається жадібною. Починаючи грати в багаторукого бандита агент не володіє інформацією про жодну дію і функція корисності рівня для кожної дії. Навіть якщо для почату гри стохастично сформувати початкову стратегію (надати випадкові оцінки кожній дії), то в результаті ефективною буде вважатися одна випадково вибрана дія на всю гру. Ймовірність що випадково буде вибрана саме дійсно ефективна дія низька і нижча з ростом кількості можливих дій. Якщо звернути увагу на умови середовища та функцію корисності, то можна побачити що відсутність від’ємної нагороди (покарання) та лише сумування нагород призведе до того, що с часом оцінка вибраної неефективної дії буде зростати завдяки призам отриманих від автомату [14]. З цього можна зробити висновок, що деякі умови середовища можуть створити додаткові перешкоди для навчання агента на певних алгоритмах і проблема дослідження та експлуатації може загострюватись.

Першим та достатньо ефективним кроком до вирішення такої проблеми стало використовування так званого ε-жадібний підхід. Він полягає в тому, щоб з ймовірністю в виконувати випадкову дію, інакше – виконувати дію згідно жадібній стратегії. Тобто агент більшу частину часу використовує жадібну стратегію замість всього часу. Перевагою такого підходу є те, що при рості кількості ігор до нескінченості за законом великих чисел стратегія агента збігається до істинної. На наведеному нижче графіку можна побачити порівняння ефективності наведених підходів до вирішення задачі багаторукого бандита за сумарною нагородою. Порівняння проводилось при 2000 ігор з 5 виборами зі збереженням стратегії з гри в гру та багаторукий бандитом з 10 важелями.

Изображение выглядит как текст, карта

Автоматически созданное описание

Рисунок 1.3 – Результати випробування ε-жадібного підходу

Як можна побачити ε-жадібний дійсно має перевагу над жадібною стратегією, але важливим є значення ε для ефективності на певному середовищі. Занадто високе значення ε призводить до надлишкового дослідження середовища, а значення близькі до 1 взагалі призводить до ігнорування вирішення задачі.

Інші існуючи методи є похідними від наведених вище і мають значних переваг над ними, тому цього достатньо для розгляду проблеми дослідження і експлуатації та використання їх у більш загальному випадку з множиною можливих станів середовища.

## Фундаментальні методи

Для розв’язку задачі навчання з підкріпленням було створенні фундаментальні методи [1], які мають чіткі математичні основи. Їх розділяють на три основні класи: методи динамічного програмування, Монте-Карло і методи основані на часових різницях (Temporary Difference – TD-методи). Буде розглянуто їх своєрідні властивості і певні ефективні підходи, які можуть бути поєднані в одному методі для покращення результатів.

## Динамічне програмування

Такий підхід використовують коли систему агента і середовища представляють формально через МППР як було показано раніше. Основною ідеєю динамічного програмування в навчанні з підкріпленням є оцінка стратегії на заданій ідеальній моделі середовища згідно принципу оптимальності Белмана. Цей принцип полягає в тому, що метод, який вирішує задачу пошуку оптимальної стратегії повинен шукати оптимальне рішення не в окремо взятому стані, а з врахуванням поточного стану і оптимальності всіх рішень в наступних станах. Як і МППР результат не залежить від попередніх станів та порядку їх появи.

Виконати оцінку стратегії означає визначити таку функцію корисності стану для будь-якої стратегії, яка буде задовольняти рівнянню Белмана, тобто для всіх s ∈ S

де – ймовірність виконання дії в стані при стратегії , – коефіцієнт дисконту, який гарантує збіжність суми і є умовою існування єдиної функції . Теоретично вже цього визначення достатньо щоб провести дізнатися значення функції для кожного стану і якості оптимальної дії вибирати ту, яка приведе в стан з найбільшим значенням функції корисності. Але це з точки зору обчислень це не є ідеальним та затратним методом через те, що при розрахунках значень на всіх станах функція збігається досить довго. Тому існують ітеративні методи динамічного програмування.

Особливість ітеративних методів полягає в тому, щоб для кожного стану ітеративно оновлювати значення функції корисності на основі попереднього значення. Якщо розглянути набір наближених функцій цінності , де буде визначено випадково, а всі наступні згідно формули:

Для всіх s ∈ S

то матимемо, що при , що показує еквівалентність результатів ітеративного методу з базовим. На практиці кількість ітерацій не проводять нескінченно довго і процес зупиняють коли отриманні значення функцій корисності є достатньо близькими до істинних. Умовою зупинки є , тобто ситуація, коли максимальна різниця між попередніми значеннями є незначною. Можна припустити, що ефективною дією після обчислень буде вважатися та, яка приводить до стану в найбільшим значенням функції корисності. Але зазвичай для визначення оптимальної дії розглядають окремо функцію корисності , яка безпосередньо залежить від попередньої і напряму дає оцінку доступним діям. Вона визначається наступним чином:

Ітеративний процес, у якому кожну k-ту ітерацію виконується оновлення оцінки та попарно називають ітерацією по стратегіям. Цей метод дозволяє по принципу динамічного програмування досить ефективно знаходити оптимальну стратегію на відомій моделі середовища. Крім відповідності методу принципу Белмана також можливе покращення стратегії завдяки теоремі про покращення стратегії [1]. Вона каже, що якщо існують стратегії і , для яких на всіх правдиве твердження, що

то стратегія буде не гірше ніж , тобто:

Це можна пояснити можливою наявністю стану, у якого більшість дій приводе до негативної нагороди, але тим часом існує одна оптимальна дія, яка переносить агента до більш позитивного стану. Тому можна вважати, що ітерація по стратегіям є більш ефективним підходом до розв’язку поставленої задачі.

Крім вище наведених існують ще ідеї, які розвивають тему використання динамічного програмування для розв’язку задачі навчання з підкріпленням, але в основа всіх таких методів полягає у використанні рівняння Белмана тим чи іншим шляхом. В цьому випадку не виникає проблема експлуатації та дослідження стратегії, на відміну від всіх наступних методів. Це головна суть, яку необхідно виділити для подальшого розгляду більш нових підходів до розв’язку задачі.

## Методи Монте-Карло

Методи Монте-Карло (МК) можуть бути використані, як засіб, який пропонує інший підхід до вирішення задачі навчання з підкріпленням. Для них немає необхідності мати точну модель середовища (зокрема функції переходів та нагороди) і вони не вимагають повного виконання марківської властивості. Їх особливість полягає в тому, що оцінка корисності і стратегія будується на лише на досвіді агента у взаємодії з середовищем. Досвід формується з послідовних вибірок станів, в яких знаходилась система, дії, яка була виконана та нагороди, яка була отримана внаслідок. Часто на практиці виникає ситуація, коли складно отримати модель середовища, а саме розподілення ймовірності переходів між станами та нагороди для певної дії. Проте досить легко отримати конкретні приклади даних утворені з цих розподілень. Маючи великий набір таких даних, методи МК статистично формують функції корисності та стратегію агенту, усереднюючи дані про нагороди для кожного стану чи дії. Таким чином метод використовують закон великих чисел для збіжності сформованої функції корисності до істинної. Для роботи з МК методами гру представляють у вигляді набору однакових по довжині епізодів, де кожній епізод це скінченна послідовність трійок (стан, дія, нагорода) сформовані при взаємодії з середовищем.

Розглянемо такий метод МК як метод всіх відвідувань і на його прикладі розглянемо формування функції корисності стану . Відвідуванням стану в цьому випадку називають кожну трійку значень в яку входить стан .

Послідовність дій наступна:

1. Спочатку створюємо випадкові і стратегію та формуємо для кожного стану пустий масив нагород .
2. Повторювати циклічно N разів:
   1. Формуємо епізод взаємодіючи з середовищем згідно стратегії.
   2. Для кожної трійки в епізоді додати до .

Оцінка кожного стану представляє собою незміщену оцінку, стандартне відхилення якої визначається як , де – число усереднених нагород [3]. Тепер маючи можна перейти до визначення стратегії . Вона зводиться до того, щоб визначити функцію корисності дій і згідно її значенням вибирати оптимальну дію. Її знаходження аналогічне пошуку функції і відрізняється тим, що нагороди асоціюються з парою (стан, дія) замість лише стану. Але в цьому випадку треба пам’ятати розглянуту проблему дослідження та експлуатації. Використовуючи жадібну стратегію при навчанні з випадковою початковою стратегією дає видає постійні детерміновані вибори дії ігноруючи дослідження інших. Тому використовуючи цей метод необхідно тим часом вирішувати і проблему дослідження та експлуатації.

Всі інші методи МК не мають якісної різниці від вище наведених, тому цього достатньо, аби описати головні їх особливості. Порівнюючи МК методи з динамічним програмуванням, необхідно звернути увагу на суттєву різницю в підходах. Головною є відсутністю впливу принципу оптимальності Белмана. З однієї сторони це великий недолік методів МК, які не можуть знайти розв’язок задачі, якщо існує лише довгострокова перспектива отримання нагороди. Хоча це можна виправити певними модифікацією алгоритма. Наприклад, знаючи послідовність станів і дій, можна відносити фінальну нагороду для кожної трійки (стан, дія, нагорода) послідовності. Самі по собі методи МК є досить гнучкі і єдина ідея, яка залишається попри всі зміни, це використання досвіду та експериментальних даних.

## TD-методи

Намагаючись створити нові методи, які будуть поєднувати переваги підходів динамічного програмування та Монте-Карло, були реалізовані методи, які основані на часових різницях (TD-методи). Цей підхід направлений на збереження ідеї відповідності принципу Белмана і розповсюдження нагороди між станами. Але при цьому не виникає необхідність знати точну модель, а як і в методах МК використовувати набір дослідних даних від середовища для формування стратегії та оцінок. Однак є різниця у реалізації цього формування. МК методи виконують оцінювання в кінці, після отримання фінального стану з відповідною отриманою нагородою. Тому існує вимога на розбиття гри на скінченні епізоди. TD-методи виконують коректування функцій корисності на кожному кроці разом з уже наявною поточними нагородою та оцінкою. Такий принцип в різні моменти часу надавати нові різні оцінки і дав назву таким методам. Типове коректування оцінки можна розглянути у простому методі як TD(0) і має наступний вигляд:

де – час формування цієї оцінки, – поточний стан, – поточна нагорода, – постійна довжина кроку навчання, – дисконт, та – наявні оцінки станів. Як видно з формули, навчання проводиться не тільки від поточної нагороди, а і від оцінки наступних станів. Саме тому воно відповідає рівнянню Белмана для методів динамічного програмування. Такий покроковий підхід до оцінювання дає можливість працювати алгоритму на середовищі з нескінченною послідовністю станів і при цьому одразу використовувати новий досвід. Це розширює можливості використання TD-методів на різних середовищах, які в своїй логікі не передбачають явного завершення гри, а, наприклад, підтримання певного процесу агентом. Незважаючи на набуті переваги і гнучкість, в значній мірі вадою цього підходу є те, що алгоритм при формуванні оцінки з самого початку опирається на випадкову функцію корисності, яка надає хибне представлення всіх станів середовища. Це провокує агента вважати певні випадково обрані дії ефективними і лише в процесі гри відбувається перевизначення хибних оцінок до наближених до істинних. Тобто виникає типова проблема жадібної стратегії визначеної у проблемі експлуатації та дослідження стратегій, рішення якої вже були розглянуті. Використання цих рішень на кожному кроці разом покроковим коригуванням оцінки дає перевагу TD-методам у швидкості апроксимації функцій корисності у порівнянні з МК методами.

Тепер розглянемо яким чином можна формувати стратегію за допомогою TD-методу. Як і в динамічному програмуванні, тут буде використаний підхід ітерації по стратегіям. Для цього необхідно визначити функцію корисності наступним чином:

Ітеративний метод, який поєднує в собі TD-метод оцінки середовища і відповідну оцінку функції корисності дії називають SARSA метод. Кожну ітерацію цього методу для оновлення та покращення стратегії використовуються дані про попередні та поточні стани і дії, які можна об’єднати в кортеж , що і дало назву методу. Тут також виконуються умови для виконання теореми про покращення стратегії, але відносно неї є ідея покращення методу. Вона полягає в тому, щоб визначати функцію корисності дії як апроксимацію безпосередньо оптимальної стратегії, незалежно від поточної. Формальний вигляд вона має наступний:

Такий метод називається Q-learning. Він схожий на SARSA і зберігає умови збіжності стратегії до оптимальної, бо стратегія продовжує визначає пари і коригувати лише їх. А завдяки визначені поточної оцінки з умовою жадібного вибору наступної дії буде формуватися більше ефективна стратегія.

## 1.4 Випробування алгоритмів на прикладі середовища

Розглянемо роботу фундаментальних методів навчання з підкріпленням за допомогою Open AI Gym – бібліотеки, в якій реалізовані приклади середовищ, на яких можна тренувати агентів. Вона надає спеціальне API, яке дозволяє на мові Python зручно взаємодіяти з середовищем, приймаючи дію від агента, надаючи інформацію про стан та нагороду агента і самостійно виводить середовище користувачу на екран.

Спочатку розглянемо середовище “MountainCar-v0”. Воно вперше було описано в роботі Andrew Moore (1990) . Середовище двовимірне і реалізує модель машинки на колесах, яка знаходиться у впадині гори (рисунок 1.4 а). Ця машинка є агентом, яка завжди має можливість виконувати одну із трьох дій – надати прискорення вліво, вправо, нічого не робити. Мета агента – зробити так, щоб машинка опинилася на вершині гори за жовтим прапорцем (рисунок 1.4 б). Але його прискорення буде недостатньо, щоб подолати силу гравітації та опинитися на вершині. Тому оптимальною стратегією буде розгойдатися, таким чином набравши достатнє прискорення, і виїхати на вершину.

Изображение выглядит как игра

Автоматически созданное описание

Рисунок 1.4 – Демонстрація гри MountainCar: а – початковий стан, б – кінцевий

Середовище надає у якості стану агента інформацію у вигляді пари , де – позиція по вісі X, а – швидкість агента по вісі X. Цього достатньо, щоб казати, що процес відповідає марківскій умові. Кожен момент часу, коли машинка координата машинки менша, ніж координата прапорця, агент отримує нагороду –1, інакше – 0 і епізод закінчується. Максимальна довжина епізоду становить 200 моментів часу.

Аналізуючи умови, яке створює середовище, можна передбачити їх вплив на навчання. В даному випадку у нас немає явно вираженої функції переходу між станами і система не надає можливості надати конкретний стан агенту для подальшого аналізу. Це унеможливлює використання методів динамічного програмування. Знаючи оптимальну стратегію розгойдування, очевидно, що стани, у яких необхідно змінювати напрямок прискорення, достатньо різноманітні, а перші кроки будуть невдалі, хоча стратегія на початку буде вірна. Це створює великі труднощі для методу Монте-Карло. Але решта умов цілком відповідають тим, на яких можуть ефективно працювати TD-методи [14].

Для експерименту було реалізовано алгоритми SARSA, Q-learning та методу Монте-Карло з такими параметрами навчання (де вони необхідні):

Результати представленні у вигляді наступного графіку:

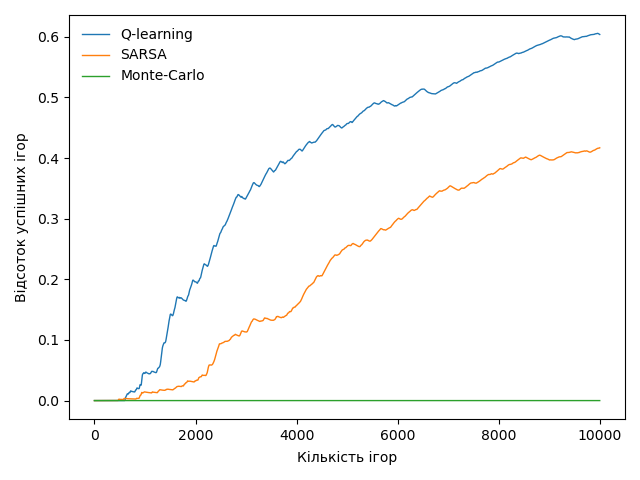


Рисунок 1.5 – графік відсотків успішних ігор проведеними алгоритмами

Як можна побачити TD-методи показали, що дійсно можуть знайти оптимальну стратегію, коли це виявилось неможливо для методу Монте-Карло. Одночасно с цим виявлено перевагу Q-методу і навчанню безпосередньо опираючись на жадібну стратегію при прогнозування оцінки, як було зазначено у визначенні.

Подальший розвиток навчання з підкріпленням у великій мірі залежить від нейронних мереж та методів глибокого машинного навчання. Завдяки ним фундаментальні методи можуть бути розвинуті до більш складних та ефективних алгоритмів, властивості яких залежать від їх реалізації. Тому спочатку буде розглянуті базові знання про глибоке навчання з вчителем та нейронні мережі.

# РОЗДІЛ 2 Основи глибокого навчання та нейронних мереж

## 2.1 Визначення та постановка задачі машинного навчання

Машинне навчання за означенням є окремим класом методів штучного інтелекту, яке базується на принципі навчання певної моделі вирішувати поставлену задачу за допомогою набору прикладів рішень подібних задач. Ключовою особливістю є те, що система не програмується і їй не надається апріорна інформація про задачу. Рішення формується лише завдяки даним представленими в загальному вигляді. Такий підхід дозволяє знаходити розв’язок задачі для якої не відомо детермінованого та ефективного алгоритму вирішення. Машинне навчання розбивається на декілька розділів, які вирішують окремі класи задач:

1. Навчання з вчителем
2. Навчання без вчителя
3. Навчання з підкріпленням

Останній був вже детально розглянутий в попередньому розділі. Причиною того, чому його можна назвати машинним навчанням є, що в процесі навчання використовуються дані сформовані функціями корисності, наданою середовищем, як приклади рішень задачі оцінювання дій агенту. Розв’язок цієї задачі є формуванням стратегії. Завдяки цієї властивості навчання з підкріпленням є подібним до навчання зі вчителем, що дає можливість використати можливості обох підходів для створення нових методів розв’язку задачі навчання з підкріпленням. Тому далі буде більш детально розглянутий цей розділ.

## 2.2 Визначення навчання з вчителем та глибокого навчання

Алгоритми навчання з вчителем містять в собі певну математичну модель, яку навчають за певним принципом вирішувати певний клас задач. З математичної точки зору її можна інтерпретувати як функцію з параметром. В якості аргумента їй надається надаються типові дані про задачу, які мають чітку структуру і числову інтерпретацію. Множину таких даних називають стимулом. Роль значення функції відіграють дані, які визначені як результати вирішення задачі на даному стимулі. Множину таких результатів називають реакцією. Стимул і реакцію для зручності також називають вхідними і вихідними даними математичної моделі [4]. В підсумку маємо пару (*стимул, реакція*), множина яких складає навчаючу вибірку, яка і визначає яка задачу і яким чином буде вирішена. Вирішити задачу означає знайти за допомогою навчаючої вибірки такі значення параметрів функції, при якій вона може для будь-якого елемента множини стимулів ставити у відповідність правильну реакцію. Принцип існування умови надання відомих розв’язків і дає назву такому підходу навчання. Вчителем називають те, що формує навчальну вибірку, або саму вибірку. Очікується, що в результаті навчання математична модель може з достатньою точністю зможе знайти та узагальнити шукану залежність між вхідними і вихідними даними. Це дозволить знаходити відповідність таким прикладам вхідних даних, на яких не відбувалося навчання.

Формально задача описується наступним чином: – множина можливих стимулів, – множина можливих реакцій, – пара «стимул-реакція», – навчаюча вибірка, – функція, яка навчається, – параметри функції, які необхідно знайти. Процес знаходження параметрів функції і є навчанням.

Прикладами задач, які розв’язуються методом навчання щ вчителем є задачі класифікації та регресії.

Задача класифікації полягає в тому, що існує множина об’єктів які для моделі визначені як її стимули, і множина класів, до яких однозначно можна віднести кожний об’єкт. Результатом функції, яка навчається, буде вектор, у якому кожний *i*-те значення – ймовірність того, що об’єкт належить *i*-тому класу. Приклад такої задачі зображений на рис. № 2.1.

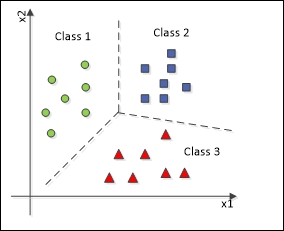


Рисунок 2.1 – Приклад задачі класифікації

В даному випадку аргументом функції, що навчається, є координати кожної точки , які можуть належати одному з трьох класів функції.

Задача регресії полягає в тому, щоб виявити та визначити залежність певної залежної величини від множини незалежної величини. Знайти таку функцію, яка буде виражати цю залежність, і яку можна використати, щоб спрогнозувати значення залежної величини від значень не представлених в навчаючій вибірці. Задачу регресії часто вирішують в статистичних задачах для виявлення певних трендів або аномальних спостережень.

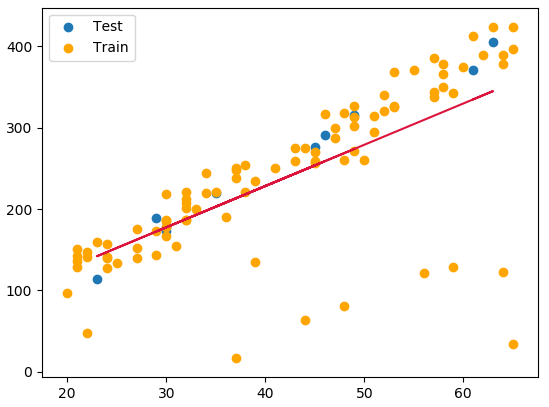


Рисунок 2.2 – Приклад задачі лінійної регресії

Приклад розв’язку задачі регресії зображено на рис. № 2.2. На вхід математичної моделі подається масив даних, який складається з кортежів значень незалежних величин, а на виході отримуємо значення залежної величини.

## 2.3 Нейронна мережа як математична модель глибокого навчання

В машинному навчанні в якості математичної моделі можуть бути використані реалізації моделей такі, як перцептрон, мережі радіально-базисних функцій, тощо. Але для використання методів навчання з вчителем у взаємодії з алгоритмами навчання з підкріпленням буде розглядатися глибоке навчання. Це підрозділ машинного навчання, який спеціалізується на нейронних мережах, які відрізняються від інших мереж, нефіксованою кількістю шарів нейронів, можливістю мати різні функції активації, тощо. Ті та інші поняття будуть описані далі.

## 2.3.1 Визначення нейронної мережі та її компонент

Нейронна мережа є складною математичною моделлю, яка може мати різні еквівалентні представлення – як з точки зору біології, так і чисто математичне визначення. Розглянемо обидва випадки і яким чином з одного формується інше.

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 2.3 – Модель нейрона

Нейрон – це елементарна одиниця з якої складаються нейронні мережі, ідея якого відповідає принципу функціювання нервових клітин. Кожний нейрон має n входів, по яким поступає сигнал до нього через ребро. Ребро має вагу – числову характеристику, яка посилює або послаблює сигнал на вході. Всі сигнали акумулюються в тілі нейрона і проходять в сумі через функцію активації, яка визначається користувачем. Результат функції передається на вихід нейрона, який може бути використаний для подальшої обробки. Тобто ми маємо вектор входів і вектор вагів . Результат роботи нейрона буде наступний:

де – функція активації. Це еквівалентно скалярному добутку векторів входів та вагів.

Шаром нейронів називається множина нейронів, які мають однакову кількість ребер і функцію активації. Кожним i-тим ребрам кожного нейрону відповідає -тий вхід сигналу. Тобто, формально кажучи, маємо вектор входів і матрицю вагів в результаті проходження сигналу через шар нейронів ми отримаємо вектор виходів (добуток вектора на матрицю). Також до шару нейронів може бути доданий умовний нейрон, який постійно зберігає значення 1. Він реалізує собою зсув (bias), який дозволяє вносити до виразу константу, значення якої визначається вагою ребра. Таким чином лінійний вираз виражений кожним нейроном має повний набір параметрів.

Изображение выглядит как текст, карта

Автоматически созданное описание

Рисунок 2.4 – Модель шару нейронів

Розглянемо послідовність шарів нейронів, ваги яких можна виразити послідовністю функцій активацій і матриць вагів таких, що , де , – кількість стовпчиків матриці , а – кількість рядків (умова добутку матриць). Таку послідовність називають нейронною мережею прямого розповсюдження. Результатом обробки такою нейронною мережею вектору входів є наступний вираз:

В якості функцій активації використовують ті, які є бієктивними відображеннями . Раніше до таких функцій висували умову диференційованості – необхідна для алгоритму зворотного поширення помилки. Але з часом на практиці почали застосовувати й інші, які не поступалися якістю і були набагато легші в обчислюванні.

Найбільш вживаними вважаються наступні функції:

1. Логістична функція (сигмоїда):

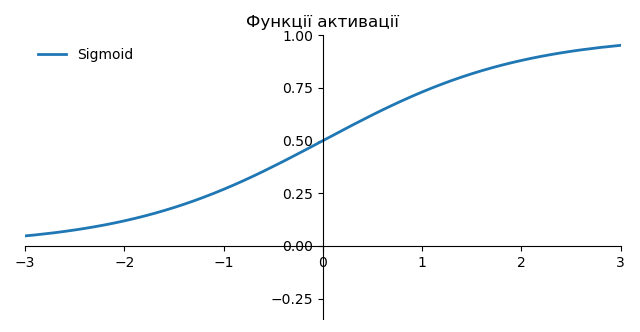


Рисунок 2.5 – Графік логістичної функції

Ця функція була одна з найперших використана для задача машинного навчання. Вона є нелінійною і цілком диференційованою, але з ростом модуля аргументу значення функції майже не змінюється, що погано впливає на процес навчання. Це явище насичення функції, що приводе до проблеми зникнення градієнту в методах оптимізації, які будуть описані пізніше. Також є проблемою обчислювальна складність у порівнянні з лінійними функціями, що стає помітно з ростом масштабу навчання. Але її перевагами є гладкий градієнт, який сприяє навчанню мережі, і нормалізація вихідного значення, що в деяких задачах ж необхідною умовою. Наприклад, його можна інтерпретувати як значення ймовірності.

1. Гіперболічний тангенс:

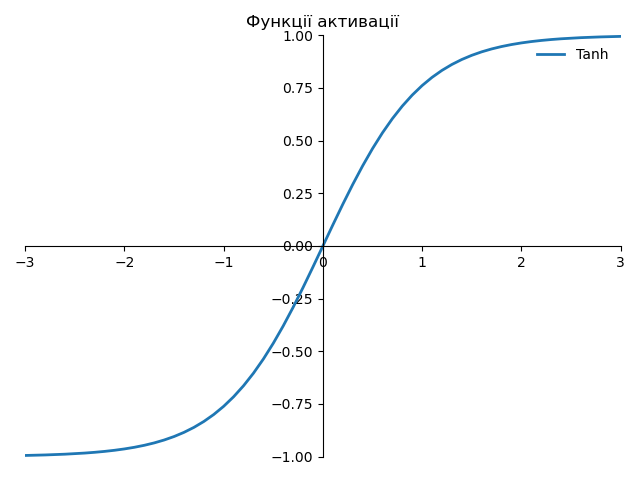


Рисунок 2.6 – Графік гіперболічного тангенсу

Ця функція своїми властивостями подібна до сигмоїди і може бути виведена через неї, через що функції мають спільні переваги і недоліки. Різниця полягає в тому, що гіперболічний тангенс має область значень , яка централізована навколо нуля. Це є причиною чому ця функція вже не може бути використана для формування нормалізованої величини. Але це знімає обмеження при розрахунку градієнту при переміщені в певному напрямку. Також має швидшу збіжність за рахунку більшого модуля похідної навколо нуля.

1. Функція лінійної ректифікації (ReLU):

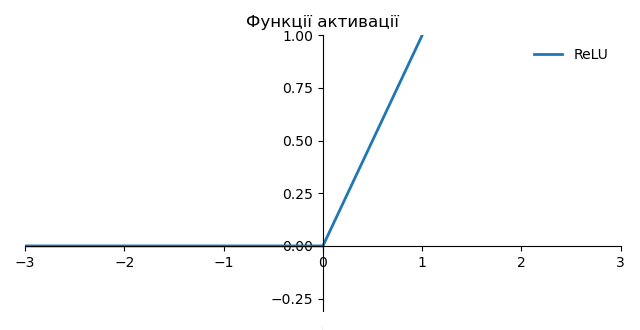


Рисунок 2.7 – Графік ReLU

Це найбільш використовувана функція активація в глибокому навчанні. Вона не є строго лінійною і не є диференційованою, бо в неї порушується умова рівномірної неперервності в точці . Але все одно вона ефективно вирішує свою задачу, а в якості похідної використовують наступну функцію:

ReLU має деякі своєрідні переваги. Вона дуже швидко обчислюється і реалізує розріджену активацію – при ініціалізації мережі близько 50% нейронів будуть видавати нульові значення (згідно формули), які зменшують кількість розрахунків. Її часткова лінійність дозволяє швидкі збігатися алгоритмам оптимізації, наприклад, стохастичному градієнтному спуску (Krizhevsky), і робить її стійкою до насичення. Головним недоліком цієї функції є явища «вмираючих» нейронів. Функція ігнорує значення менше нуля, тому є можливість, що в процесі навчання велике значення градієнта оновить ваги таким чином, що нейрон більше ніколи не буде активований. А через це градієнт також буде нульовим і тому ця частина мережі ніколи не зміниться під час навчання. Щоб вирішити цю проблему, для функції активації створили декілька модифікацій. Наприклад, Leaky ReLU, яка має формулу , яка не має нульової похідної в кожній точці області визначення і зберігає властивості оригінальної функції, окрім розрідженої активації.

1. Softmax

Ця функція є узагальненням логістичної функції для багатовимірного випадку, коли в якості аргументу надається вектор , а результатом є вектор, кожен -тий елемент якого обчислюється за наступною формулою:

Кожний елемент вихідного вектору належить множині , а сума всіх елементів дорівнює одиниці. Це можна розуміти як розподіл ймовірностей, яке складається з значень ймовірностей. Ця функція активації в такій інтерпретації використовується в задачах класифікації з небінарною кількістю класів в останньому шарі нейронної мережі.

## 2.3.2 Компоненти навчання нейронних мереж

Як уже було зазначено, нейронні мережі є прикладом математичної моделі для навчання з вчителем. Розглянувши її побудову і принцип функціювання, можна розглянути яким чином нейронна мережа навчається. Для реалізації цього процесу існує метод зворотного поширення помилки, який був вперше описаний P.J. Werbos в 1974 році. Метод використовує навчаючу вибірку для навчання і потребує визначені функція помилки і алгоритм оптимізації.

Функція помилки (функція втрат) – це функція, яка чисельно описує наскільки даний нейронною мережею вихід відрізняється від істинного . Тобто, це функція , яка приймає аргументом значення та . Класичним прикладом такої функції застосовується в методі найменших квадратів, де вона визначена наступним чином:

Маючи таку або подібні функції, задача навчання нейронної мережі еквівалентна задача оптимізації функції втрат. Тобто, ми маємо мережу, представлену як функцію , та функцію втрат , де і необхідно знайти такі параметри , при яких для всіх .

Для вирішення цієї задачі метод зворотного поширення помилки використовує алгоритми оптимізації. Метод буде розглянутий далі і алгоритми оптимізації необхідні для визначення певної зміни параметрів мережі з метою наближення функції втрат до мінімуму. Класичним алгоритмом оптимізації є метод стохастичного градієнтного спуску.

Метод стохастичного градієнтного спуску вирішує задачу пошуку нижнього локального екстремума функції. Принцип його роботи полягає в наступному. Нехай – диференційована функція з областю визначення та похідною .Візьмемо випадкову точку з множини та обчислимо значення . Якщо розглянути його як одновимірний вектор і обернути його напрямок (зі знаком мінус), то напрямок буде вказувати на один із локальних мінімумів функцій. Ми можемо це використати для визначення точки , яка буде ближча до мінімуму за допомогою наступної більш узагальненої формули:

де – коефіцієнт швидкості навчання, необхідний для масштабування зміщення значення наступної точки. Таким чином ми можемо визначати все більше точок до тих пір, поки точка не буде достатньо близько до шуканого мінімуму. Процес можна спостерігати на рис. № 2.8.

Изображение выглядит как карта

Автоматически созданное описание

Рисунок 2.8 – Приклад процесу градієнтного спуску

Як можна помітити, чим менше значення похідної в точці, тим менший вибирається зсув, який таким чином регулюються від надмірних значень. Саме тому присутній коефіцієнт швидкості навчання, який виконує ту саму задачу. Умовою зупинки генерації нових точок є достатньо мала відстань між сусідніми точками, тобто . Як можна побачити цей метод не гарантує знаходження глобального мінімуму, який є найкращим розв’язком задачі оптимізації. Але у випадку нейронних мереж у нас функція має дуже високу розмірність і ми маємо справу з градієнтом функції втрат. Це означає, що ймовірність того, що потрапимо у локальний мінімум спадає з ростом кількості вимірів, тому на практиці це не є проблемою.

Набагато частішим випадком є потрапляння у сідлову точку, що є проблемою для методу градієнтного спуску. При її досяганні значення градієнту близькі до нуля, що сильно затримує процес виходу з сідлової точки та пошук мінімуму функції. Для вирішення цієї проблеми були створені інші методи оптимізації, які інтерпретують спуск як рух точки донизу і використовують для оптимізації величини швидкості та прискорення руху точки. Серед таких методів можна виділити метод пришвидшених градієнтів Нестерова та Adam. Вони – перший і найкращий методи відповідно, які опираються на накопиченій швидкості спуску, яка зміщує точку з сідла у напрямку пройденого спуску. Детально розглядатися вони не будуть, а згадані вони тому, що будуть використані далі для вирішення задачі.

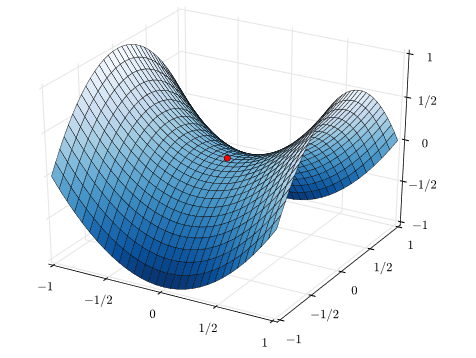
****

Рисунок 2.9 – Приклад сідлової точки

## 2.3.3 Метод зворотного поширення помилки

Тепер, визначивши поняття функція втрат та алгоритму оптимізації, можемо розглянути принцип роботи методу зворотного поширення помилки для навчання нейронних мереж. В якості функції втрат буде використовуватися квадрат різниці методу найменших квадратів. Метод градієнтного спуску буде розглядатись в якості алгоритму оптимізації.

Нехай маємо елемент вибірки та мережу , яка складається з одного шару нейронів з функцією активації . Мережа прийняла на вхід значення та сформувала відповідь . Маємо значення функції втрат . Відтепер будемо розглядаємо функцію , яка має стале значення та змінну . В результаті маємо функцію втрат, мінімум якої необхідно знайти:

Для методу стохастичного градієнтного спуску необхідно знати похідну функції. В нашому випадку похідну функції втрат від змінної .

Тепер застосуємо її для методу стохастичного градієнтного спуску:

Таким чином ми знайшли необхідний зсув параметру функції нейронної мережі з одним шаром для елементу вибірки .

Цей підхід можна узагальнити для мереж з нетривіальною кількістю шарів. Для цього необхідно виконувати описану процедуру для кожного шару нейронів, починаючи з останнього, крім першого. Замість використовувати вихідне значення попереднього шару, яке в ньому виникає при розрахунку мережею своєї функції від аргументу . Формально для мережі, як послідовності шарів нейронів з функціями активацій і матриць вагів , зсув вагів кожного шару на елементів вибірки можна визначити наступним чином:

де – значення нових вагів для -того шару.

## 2.3.4 Процес навчання нейронної мережі

Було розглянуто яким чином нейронна мережа може навчатися апроксимувати функцію відповідності стимулу до реакції на одному елементі вибірки . Для повноцінного процесу навчання на всій навчальній вибірці необхідно визначити принцип використання вибірки згідно властивостей нейронної мережі до узагальнення отриманих знань. Перш за все це необхідно для уникання явища недонавчання та перенавчання мереж.

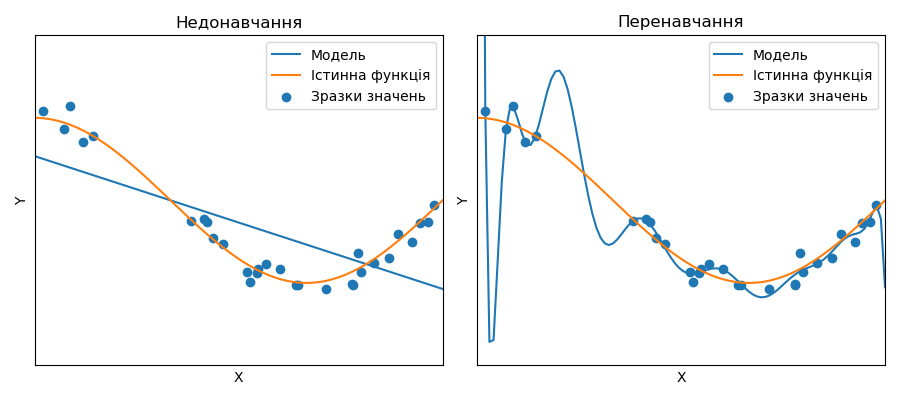


Рисунок 2.10 – Приклад недонавчання та перенавчання моделі

Явище недонавчання характеризується недостатньо низьким значення функції втрат для будь-якого елементу вибірки та за межами неї. В таких умовах можна сказати що мережа не може вирішити задачу і не виявила закономірності між даними. Причиною цього як правило називають недостатню складність моделі, яка навчається.

При перенавчанні моделі виникає ситуація, коли модель точно описує залежність між даними вибірки, але при цьому порушаються принцип узагальнення залежності. Модель більш запам’ятовує поточну шумову інформацію вибірки, яка виникає при її формуванні. Тому функція втрат обчислювана на вибірці дає низьке значення помилки і водночас високу на будь-яких інших даних множини можливих даних.

Для виявлення і вирішення вказаних проблем використовують підхід валідації результатів моделі за допомогою підмножини вибірки, які називають тестовими даними. Тобто перед процесом навчання вся вибірка ділиться на дві множини, які не перетинаються, у приблизному відношенні 1:10. Більша множина називається тренувальними даними, а менша – тестовими. В процесі навчання для моделі використовують тренувальні дані для методу зворотного поширення помилки, а за допомогою тестових даних відслідковують значення функції втрат в поточний момент навчання. Використання цих двох множин повинно чергуватися, тому для визначення таких ітерацій всю множину тренувальних даних ділять на частини певного фіксованого розміру. Такі частини називаються батчами і як правило їм задають розмір 32/64/128 елементів тренувальної множини. Кількість цих батчів визначає кількість ітерацій навчання моделі. Кожну ітерацію модель навчають на -тому батчі і обчислюють середнє значення функції втрат на тестових даних. Таким чином можна відслідковувати процес навчання та його точність.

Недонавчання можна виявити, помітивши нестабільні зміни значень функції втрат. Замість того, що постійно йти на спад і зупинитися у певної асимптоти коло нуля, значення функції в середньому залишаються в тому ж інтервалі, в якому були в момент почату навчання. Також для недонавчання характерно, що одночасно з описаною проблемою значення функції втрат стабільно зменшується, що мало би казати про успішність процесу навчання. Вирішенням цієї проблеми є ускладнення нейронної мережі, додавши нейрони до існуючих шарів або додавши новий шар.

Перенавчання виникає в той момент, коли після певного проміжку часу успішного навчання моделі, значення функції втрат на тренувальних даних продовжують знижатися, а на тестових даних спостерігається поступовий ріст функції. Це означає, що саме в цей момент модель вивчає шум тренувальних даних, який не відповідає тестовим. Достатньо завершити процес навчання на цьому моменті часу і сказати, що процес був успішний.

Використовуючи такий підхід до використання навчальної вибірки з методом зворотного поширення помилки з визначеними алгоритмом оптимізації, функції втрат для навчання нейронної мережі з визначеною топологією було повністю визначено процес глибокого навчання.

# РОЗДІЛ 3 Навчання з підкріпленням з елементами глибокого навчання

Розглянуті розділи машинного навчання, як вже було визначено, розв’язують різні задачі. Але існує можливість використати методи глибокого навчання для розв’язку певних підзадач навчання з підкріпленням. Такою задачею є пошук функцій корисності станів та дій і , які визначають роботу алгоритмів TD-методів. В процесі взаємодії агента і середовища перший в кожен момент часу отримує інформацію про те, в якому він стані і яку нагороду він отримав за виконання визначеної дії. Накопичуючи цю інформацію впродовж всього часу взаємодії, можна сформувати навчаючу вибірку для навчання математичної моделі глибокого навчання. Така ідея вперше було реалізована в алгоритмі Deep Q-learning (DQN).

Алгоритм DQN опирається на ті самі принципи, що Q-learning, і також має на меті пошук оптимальної стратегії за принципом Белмана. Його особливість полягає в реалізації апроксимації та обчислення функції корисності за допомогою математичної моделі глибокого навчання. Водночас функція зберігає своє визначення для алгоритму :

Використання нейронних мереж для апроксимації функцій корисності має враховувати особливості їх навчання. Крім змінних параметрів мережі, як топологія і швидкість навчання, TD-методи мають свої параметри як дисконт на наступне значення функції корисності. Це ускладнює процес навчання і збільшує ймовірність створити для нього несприятливі умови. Ці умови можуть призвести до недонавчання або до виродження процесу навчання, коли оптимізація функції втрат не виконується через великі зсуви градієнту.

Також при навчанні нейронної мережі в навчанні з підкріпленням ми не можемо опиратися на значення функції втрат. Оптимальна стратегія агента досліджується і відповідність між стимулом і реакцією не є сталою. Однак потенційно нейронна мережа здатна формувати узагальнену відповідність між даними, що допомагає їй розглядати схожі стимули, які раніше вона не розглядала, і формувати подібні висновки. Тобто дійсно відбувається відтворення залежності на всій множини вхідних даних. Класичні TD-методи, як наприклад Q-learning, в своїй реалізації не мають властивості «розуміти» схожість певних вхідних даних. Для них будь-які аргументи функцій корисності, які не дорівнюють один одному, представляються як апріорі різні і подібність між ними лише досліджується в процесі пошуку та формування стратегії.

Як і в типових задачах глибокого навчання існує певний алгоритм до навчання моделі на елементах навчаючої вибірки для запобігання явища перенавчання. Цим підходом метод подібний до методів Монте-Карло. Гра ділиться на епізоди, впродовж яких агент діє в межах його стратегії, яка не змінюється, і накопичує дані надані середовищем. Ці дані зберігаються разом з іншими тренувальними даними, отриманими в попередніх епізодах. Єдиною вимогою до структури, в яких вони зберігаються, це наявність максимального розміру, після набуття якого починають видалятися найстарші елементи вибірки. Така умова необхідна для того, щоб з часом навчання опиралося на більш актуальні ситуації, не розглядаючи дії, які відомі як неефективні. Після кожного епізоду з набору накопичених тренувальних даних вибирається випадковим чином у випадковому порядку певна кількість елементів над якими виконують зворотне поширення помилки моделі. Таким чином повністю визначається робота алгоритму для вирішення поставленої задачі.

В цілому враховуючі всі особливості даного підходу можна сказати, що в порівнянні з базовими TD-методами DQN має більше можливостей для пошуку оптимальної стратегії витрачаючи для цього більше часу. Так як це перший метод який поєднував в собі елементи навчання з підкріпленням та навчання зі вчителем, після нього було створено багато модифікацій до подібних TD-методів. Серед них буде розглянутий метод Advantage Actor-Critic (A2C). Ідея методу була визначена науковцями в контексті фундаментальних алгоритмів навчання з підкріпленням, але ефективний результат він показав використовуючи елементи глибокого навчання.

В основі його ідеї полягає визначення нового поняття «перспективи» (advantage) і використання двох моделей для оцінки та формування стратегії. Модель, яка відповідає за оцінку наявної стратегії, називають критиком. Вона обчислює значення функції . Критик на кожному кроці дає оцінку ситуації в якій опинився агент і ця оцінка є складовою перспективи. Перспектива, яку визначає актор – друга модель алгоритму, визначається як оцінка оптимальності дії відносно інших дій та поточного стану:

Тобто в той час, коли DQN визначає абсолютну оптимальність дії, незалежно від оцінки стану, A2C опирається на середнє значення ефективності всіх дій, формульоване . Головною перевагою цього підходу є те, що алгоритм ефективно оброблює випадки, коли абсолютне значення оптимальності дій є негативним і виникає зміщення середнього значення оцінок від нуля. Тобто при наявності лише негативних дій алгоритм вибере найменш негативне, при наявності лише позитивних дій – найбільш позитивне. Таким чином процес навчання агенту стає більш гнучким і стабільним, ніж використання алгоритму DQN.

Методи, які були описані в цьому розділі, будуть застосовані для пошуку оптимальної стратегії в диференціальних іграх.

# РОЗДІЛ 4 Диференціальні ігри та ігрові середовища

Диференціальні ігри є розділом більш широкої математичної теорії, яка називається теорією керування. Це такий розділ математики, який розглядає задачу, в яких присутні такі елементи системи як об’єкт керування, пристрій керування. Розв’язком задачі знаходження оптимального керування системою керування відповідно до умов та обмежень, встановлених до об’єкту і його внутрішніх характеристик. Ці характеристики описуються певними залежностями різної природи – алгебраїчними, диференціальними, інтегральними та іншими. Більш формально всю систему можна описати наступним чином.

Изображение выглядит как экран, общедоступный, белый, знак

Автоматически созданное описание

Рисунок 4.1 – Основні елементи системи керування

Існують об’єкт керування і пристрій керування. Об’єкт генерує – вектор фазових координат, який описує поточний стан системи. Множина всіх можливих фазових координат утворює множину всіх можливих станів системи і називається фазовим простором. Пристрій керування, який отримує цю інформацію від об’єкта, на її основі формує керування, визначене векторною функцією керування . Пару векторних функцій називають процесом керування. Задачею оптимального керування називають знаходження такого , яке за фіксований час змінить стан системи з фази до і при цьому забезпечить мінімум цільової функції:

Простіше кажучи, керування повинно виконати задачу з мінімальним втручанням.

В якості прикладу такої задачі можна навести задачу про рух математичної точки. Нехай існує математична точка, рух якої можна описати через систему диференційних рівнянь:

Також відома інформація початковий стан і необхідний кінцевий стан і про обмеженість керування:

Для даної системи ставиться задача пошуку керування на відрізку , яка переведе її з початкових фазових координат до кінцевих за той же фіксований час і забезпечувати мінімум функції:

В термінах теорії керування диференціальні ігри розглядаються як керуючу систему, у якій існує нетривіальна кількість керувань, яким поставлені протилежні задачі. Для визначення керування використовується лише поточна інформація про кожного з гравців, яка цілком виражається фазовими координатами системи. Ці керування впливають на окремі компоненти фазових координат, які відповідають керованим ними об’єктами.

Найбільш дослідженим випадком диференціальної гри є задача переслідування з двома керуваннями. Зазвичай це інтерпретується як задача хижака і жертви , у яких абсолютно протилежні задачі. Хижак повинен за найкоротший час наздогнати жертву, а друга повинна уникати зустрічі якомога довше. Вони знаходяться у двовимірному середовищі з випадковим положенням і нульовою швидкістю. Обоє мають максимальну швидкість, котра більша у хижака, і прискорення, яке більше у жертви. Формально це виражається наступною розгорнутою системою рівнянь:

Саме ця задача буде розглянута для дослідження ефективності роботи методів навчання з підкріпленням для диференціальних ігор.

Задачі теорії керування за постановкою досить подібними до задач машинного навчання з підкріпленням. В той час, коли перша оперує поняттями пристрою і об’єктом керування, друга має еквівалентні поняття середовища і агенту. Фазовим координатам відповідають стани, а керуванням – дії агента в середовищі. Різниця полягає у виражені поставленої задачі для агента – диференційні рівняння і нагорода від середовища – та методу її вирішення, яка проводе чітку межу між цими галузями науки. Але це не виключає можливості використання методів навчання з підкріпленням для вирішення задач оптимального керування шляхом спеціального визначення ігрового середовища. Таке визначення полягає у відтворення відповідності функції нагород і умов задачі оптимального керування. Наприклад, система може видавати певне значення нагороди в момент досягнення цілі і відносно мале значення покарання за кожен момент затраченого часу. Тому знаходження оптимальної стратегії буде еквівалентним пошуку оптимального керування. Далі буде розглянуто конкретну реалізацію такого середовища і застосування описаних алгоритмів глибокого навчання з підкріпленням для дослідження ефективності цих методів пошуку розв’язку задачі диференційної гри.

# РОЗДІЛ 5 Аналіз ефективності методів навчання з підкріпленням для гри переслідування

## 5.1 Опис програмної реалізації середовища

Для того, щоб дослідити ефективність роботи алгоритмів навчання з підкріпленням, було розроблено програмну реалізацію гри переслідування. Для цього була використана мова програмування Python разом з бібліотеками Pygame, яка реалізовувала графічне зображення середовища і агентів, і TKinter, яка реалізує керуючий інтерфейс програми. Завдяки ньому є можливість керувати середовищем і змінювати певні характеристики процесу навчання.

Изображение выглядит как снимок экрана, компьютер, ноутбук

Автоматически созданное описание

Рисунок 5.1 – Головне вікно програмної реалізації середовища

Як і в стандартному визначені гри, існують два агенти – хижак, зображений червоним кругом, і жертва, зображена зеленим кругом. Вони мають координати положення у двовимірному просторі, вектори швидкості, які разом формують стан системи (фазові координати), на які будуть опиратися агенти під час навчання. Швидкості агентів обмежені певними максимальними значенням у відповідності умовам гри переслідування. Ці значення можна спостерігати у спеціально відведеній таблиці програми, яка зображена на рисунку № 2.5. Слід зазначити, що фазові координати попередньо нормовані системою відповідно до максимальних значень.

Изображение выглядит как снимок экрана

Автоматически созданное описание

Рисунок 5.2– Вивід поточного стану середовища та нагород

Тобто центр координатної площини розміщений в центрі ігрової площини і координати положення агентів обчислюються як відношення вектору положення до половини довжини ігрового простору. Швидкість агентів виражається у відношенні фактичної швидкості до максимальної швидкості хижака. Таким чином кожна координата системи є числом з проміжку .

Проте в контексті даної задачі можна зробити певне спрощення задачі для агентів. Пам’ятаючи, що саме явище руху є відносним, ми можемо представити рух агентів відносно один одного і виражати положення вектором напрямку хижака до жертви. Одночасно с цим залишимо в складі вектору стану середовища нормовані вектори швидкості. Таким чином агенти зможуть достатньо швидко знайти достатньо ефективну, але не оптимальну стратегію, яка буде фіксованою точкою для пошуку оптимальної. Тобто буде досліджена необхідна послідовність перших дій для подальшого пошуку стратегії. Очевидно, що оптимальна стратегія жертви буде передбачає використання різниці прискорень і максимальних значень швидкостей для «спритного ухиляння» від атак хижака. А оптимальна стратегія хижака передбачає не просто слідуванню вектору напрямку, що є тією самою ефективною, але неоптимальною стратегією. Вона передбачає рух жертви і вимірює певне упередження руху – зміщення траєкторії, згідно «спритним» можливим діям жертви згідно даним векторам швидкості. Таким чином в процесі навчання не витрачається час на пошук з повної множини можливих станів середовища. Досліджується підмножина станів, яка в будь-якому випадку має ефективну стратегію і в тому числі оптимальну.

Изображение выглядит как часы

Автоматически созданное описание

Рисунок 5.3 – Множина дій агентів в середовищі

В даному середовищі агенти мають постійне скалярне значення прискорення відповідно до гри переслідування, і керування агентів зводиться до вибору вектору напрямку прискорення. Але вектор вибирається не з всього простору можливих векторів, а з визначеної множини, яка має 8 елементів, які представлені на рисунку № 5.3. Таке спрощення дозволяє спросити процес пошуку оптимальної стратегії, а представлення у вигляді зліченної множини дозволяє застосовувати на них функції корисності виражені кожною компонентою вихідних даних нейронної мережі.

Для визначення задачі агентів згідно виражених вище гіпотез нагороди видаються наступним чином.

Жертві необхідно уникати зустрічі і залишатися «живою» якомога довше. Тому при зустрічі вона отримуватиме покарання, допустимо, в 10 одиниць. В інших випадках вона буде отримувати нагороду в 1 одиницю. Це дозволить їй ухилятися від негативних сценаріїв, які буде створювати хижак

Мета хижака зустріти жертву і бажано за мінімальний час. При цьому бездіяльність не є задовільною поведінкою і саме на ньому повинна лежати ініціатива. Це ускладнює визначення необхідних нагород. Однозначно, що при зустрічі хижак повинен отримати значну нагороду, нехай в 20 одиниць. Проте складно відповісти яка повинна бути нагорода в будь-який інший момент часу. А саме – в проміжкові моменти між початком гри і шуканою зустріччю. Розглянемо чотири можливі випадки нагород:

1. Наявність нагороди.

Не має сенсу давати хижаку таку нагороду в процесі переслідування і це може змінити його мету на бездіяльність, що не відповідає вимогам задачі.

1. Відсутність нагороди і покарань.

З одного боку це є чесною оцінкою, проте хижак апріорі не знає про можливі нагороди. В той час, коли момент зустрічі є дуже малою частиною множини всіх можливих станів. Тому агент, діючи спочатку стохастично, може ніколи і не отримати нагороди, а тому не навчатися. В такому випадку не допоможе компроміс дослідження і експлуатації, бо немає чого досліджувати.

1. Наявність покарання

З одного боку агент під час дослідження дій почне відсікати негативні вибори і з часом може зупинитися на оптимальній. Але виникає проблема в збережені інформації при довгій послідовності дій, необхідній для нагороди. Якщо нагорода буде отримана лише через виконання деякої довгої послідовності дій, то постійне покарання буде розмивати отримане позитивне значення. Це призводить до того, що його поведінка залишиться стохастичною.

1. Параметризована нагорода

Це нагорода, яка буде формуватися в залежності від певних величин. Тобто це буде нестаціонарна числова функція, яка буде обчислюватися на кожному кроці. Вимагається зустріч хижака та жертви, що означає мінімізацію відстані між ними. Тому можна виразити нагороду як зменшення відстані між ними. Тобто це буде різниця відстані на попередньому стані від відстані на поточному стані. Таке визначення нагороди відповідає поставленій задачі і уникає проблем, які виникають в інших підходах. Також це відповідає зробленим висновкам по станам середовища. Такий підхід буде обрано для описаного середовища.

## 5.2 Опис реалізації алгоритмів навчання

На описаному середовищі будуть розглянуті такі методи навчання з підкріпленням як DQN та A2C. Вони будуть попарно застосовані для навчання в якості хижака і жертви. Також для незалежного аналізу ефективності один від одного вони будуть випробувані самостійно без керування другим агентом. Тому ми матимемо шість комбінацій застосувань, які будуть досліджені. Кожен із них реалізований на мові програмування Python, використовуючи бібліотеку для нейронних мереж Keras, яка побудована на базі бібліотеки для ефективної обробки матриць TensorFlow. В якості моделі для навчання використані нейроні мережі.

Для алгоритму A2C застосовано дві моделі: для актора по п’ять шарів нейронів з кількостями нейронів 24, 24, 18, 18 і 8 відповідно (8 на вихідному шарі відповідно кількості дій), а для критика – те саме, тільки один нейрон в останньому шарі. Модель критика, як вже було означено раніше, виконує задачу по обчисленню функції , тобто повертає одне скалярне значення. Іншими словами можна сказати, що модель критика вирішує задачу регресії. До значень, які повертають дані математичні моделі, буде застосований ε-жадібний підхід.

Для алгоритму DQN застосована модель з чотирма шарами нейронів з 90, 90, 72 і 8 нейронів відповідно.

Програмний код цих агентів буде наведений в додатку А.

## 5.3 Метрики

Для проведення експерименту необхідно визначити метрики - величини, по яким будуть зроблені висновки щодо ефективності алгоритмів. Ці метрики повинні відображати те, наскільки успішною є стратегія з точки зору користувача, який проводить експеримент, та точки зору самого середовища, для виявлення «розуміння» задачі системою.

В якості користувацьких метрик буде використовуватись відсоткове співвідношення пройдених епізодів до епізодів, які закінчилися зустріччю хижака та жертви. Високі значення цього співвідношення будуть означати високу ефективність роботи алгоритму. Також в якості метрики буде використана тривалість кожного епізоду, яка в нашому випадку матиме певне максимальне значення через обмеженість задачі у часі. Низька тривалість епізоду буде означати використання ефективної стратегії хижаком.

В якості метрики системи агента і середовища будуть використані значення сумарної нагороди агентів за епізод. Таким чином у нас будуть дві величини, які в принципі також неявно обмежені максимальним значенням, яке виникає через обмеженість часового проміжку гри. Очевидно, що значення цих показників буде вказувати на успішність агентів з точки зору середовища.

## 5.4 Визначення умов експерименту

Для дослідження ефективності за допомогою експерименту необхідно визначити певні різні умови, які надає система агенту, та властивості самого агенту. Різноманітність умов середовища полягають у наданні стану в різних його представленнях, які вже були описані вище, та у різних підходах до оцінювання дій. Серед властивостей, які змінюють якість роботи агенту є деякі характеристики як значення максимальної швидкості та скалярне значення прискорення.

Серед умов середовища будуть розглянуті стани виражені координатами положень агентів (далі «Стан-1») та векторами напрямку і відносної швидкості (далі «Стан-2») та негативні (далі «Нагорода-») і параметризовані нагороди (далі «Нагорода+»). Також будуть зконфігуровані деякі властивості агентів при необхідності.

Поєднуючи ці умови очікується отримання різних результатів на основі метрик, які визначають смисл проведення інших експериментів та висновки, які можна зробити щодо ефективності агентів. Перш за все необхідно переконатися в тому, що алгоритми здатні взагалі вирішувати задачу пошуку деякої стратегії на сприятливих умовах. А далі ускладнювати завдання і робити висновки опираючись на отримані метрики.

Тому буде виконана наступна послідовність експериментів:

1. Працездатність хижака

Хижак буде використовувати кожний алгоритм для розв’язку задачі, жертва буде бездіяльна. Обмеження на швидкість та прискорення буде стандартним. Здатність хижака рухатись до нерухомої жертви є необхідною умовою для продовження експерименту.

1. Працездатність жертви

Хижак і жертва будуть використовувати кожний алгоритм для формування стратегії. Обмеження на швидкість та прискорення будуть стандартними за винятком швидкості жертви – вона буде дорівнювати швидкості хижака. Здатність знайти ефективну стратегію при таких сприятливих умовах є необхідною умовою для продовження експерименту.

В усіх наступних експериментах хижак і жертва використовують однаковий алгоритм, обмеження на швидкість та прискорення стандартні:

1. Ефективності агентів, один алгоритм, Нагорода-, Стан-1
2. Ефективності агентів, один алгоритм, Нагорода-, Стан-2
3. Ефективності агентів, один алгоритм, Нагорода+, Стан-1
4. Ефективності агентів, один алгоритм, Нагорода+, Стан-2
5. Ефективності агентів, різні алгоритми, Нагорода+, Стан-1
6. Ефективності агентів, різні алгоритми, Нагорода+, Стан-2

Кожний експеримент триває 300 епізодів по 500 одиниць часу в кожному. Протягом його реалізації будуть зібрані дані метрик для побудови графіків, які ілюструють результати експерименту.

## 5.5 Результати експериментів

Розглянемо окремо результат кожного експерименту:

### 1. Працездатність хижака

Був проведений експеримент з двома алгоритмами, отримані метрики виражені графіком на рисунку №5.4.

Изображение выглядит как карта, текст

Автоматически созданное описание

Рисунок 5.4 – Результат експерименту №1

Як можна побачити, алгоритм A2C досить швидко знайшов ефективну стратегію і наблизився до 100% успішних епізодів, що означає здатність алгоритму вирішувати цю задачу. Тривалість епізоду з часом зменшувалась і це означає, що хижак все швидше досягав мети і його стратегія з часом позбавлялася неефективних проміжкових дій. С точки зору середовища агент відібрав таку стратегію, яка рівномірно усереднює значення нагороди впродовж всього епізоду і є стабільною.

Алгоритм DQN не знайшов ефективної стратегії для вирішення задачі, що зрозуміло по відсотку успішних ігор. Мала тривалість епізодів на початку експерименту пояснюється проблемою зі згладженням графіку. В наступних епізодах можемо спостерігати високу тривалість епізоду, значення якою різко змінювалися. Також графік сумарної нагороди не є рівномірним. Це свідчить про нестабільність процесу пошуку оптимальної стратегії алгоритмом.

Отримавши такі результати, можна сказати, що алгоритм A2C є ефективним для пошуку стратегії в даних умов і тому буде застосований в наступних експериментах в якості хижака. Ситуація з DQN протилежна, тому цей алгоритм не буде застосований в ролі хижака.

### 2. Працездатність жертви

Був проведений експеримент з двома алгоритмами, отримані метрики виражені графіком на рисунку №5.4. Хижак керується алгоритмом A2C.

Изображение выглядит как текст, карта

Автоматически созданное описание

Рисунок 5.5 – Результат експерименту №2

Незважаючи на умови, які надають цілковиту перевагу жертві для уникнення зустрічі, алгоритм DQN не зміг знайти стратегію для досягнення поставленої мети. Частота зустрічей хижака і жертви швидко збігається до 100% і тривалість епізоду достатньо низька, що приводить до наведених висновків.

Алгоритм A2C зміг знайти достатньо ефективну стратегію, але її не можна назвати оптимальною. Частота зустрічі хижака і жертви впродовж експерименту залишалася на рівні 30%, що є непоганим результатом. Враховуючи, що жертва мала тільки вектори напрямку в якості наданих станів, вона не може оцінити дистанцію між агентами, тому це є достатньо ефективною стратегією. Також необхідно звернути увагу на проміжок між 200 та 250 епізодом. В цей момент спостерігається скачок на графіку тривалості епізодів і на графіку сумарних нагород. Вони означають що в саме на початку цього проміжку жертва виявила ефективну стратегію уникнення, яка стала дуже ефективною для уникнення хижака з його поточною стратегією. Але хижак також розвивається в продовж експерименту, тому на 250 епізоді ним була виявлена своя стратегія, яка показала ефективність проти нової стратегії жертви.

Така різка зміна даних дозволяють розуміти на якому проміжку всього експерименту який агент має перевагу над іншим на поточний момент. Слід зауважити, що це є нормальною поведінкою у випадку диференційних ігор і не є аномальним спостереженням.

### 3. Ефективності агентів, один алгоритм, Нагорода-, Стан-1

Був проведений експеримент, використовуючи лише A2C. Отримані метрики виражені графіком на рисунку №5.6 *синіми* лініями.

Результати пошуку стратегії хижаком є незадовільними з точки зору будь-яких наведених показників. Єдиною причиною цього стало використання негативної нагороди в будь-який момент часу для стимуляції оцінки кожної дії. Агент не здатен зрозуміти свою мету на основі лише негативних оцінок дій. Йому необхідно вказувати збалансовану оцінку поточної ситуації для розуміння того, наскільки умови змінилися в кращу сторону.

Изображение выглядит как текст, карта

Автоматически созданное описание

Рисунок 5.6 – Результат експериментів №3,4,5,6

У випадку жертви також справедливі наведені висновки, але на відміну від хижака у неї будь-яке положення чи стан, який не є термінальним, є задовільним для вирішення задачі. Враховуємо, що для зустрічі агента і жертви достатньо знаходитися на певній мінімальній відстані. Тому, хоч і потужність множини всіх термінальних станів дорівнює потужності множини всіх станів, геометрично одна є нескінченно меншою, ніж друга. Тому ймовірність випадково потрапити в термінальний стан є дуже малою. Саме тому для хижака стає неможливою задача пошуку оптимальної стратегії зустрічі в таких умовах. А для жертви стає неможливою задача пошуку оптимальної стратегії при наявності такої у хижака.

Залишається питання про тривалість епізоду і чого вона є такою низькою при таких складних умовах хижака, на яке складно надати обґрунтовану відповідь.

### 4. Ефективності агентів, один алгоритм, Нагорода-, Стан-2

Був проведений експеримент, використовуючи лише A2C. Отримані метрики виражені графіком на рисунку №5.6 *помаранчевими* лініями.

Як можна побачити, ситуація не набагато краща, ніж в попередньому випадку. Значення нагород відповідає наведеним висновкам по використанню негативної нагороди. Але спостерігається деяка перевага хижака над жертвою на початку експерименту. Скоріш за все це є випадковим результатом який виник від експлуатації початкової випадкової стратегії хижака. Але вище описані проблеми інтерпретації нагороди агентів не зберегли цю стратегію і відновили проблему попереднього експерименту.

### Ефективності агентів, один алгоритм, Нагорода+, Стан-1

Був проведений експеримент, використовуючи лише A2C. Отримані метрики виражені графіком на рисунку №5.6 *зеленими* лініями.

За даними метриками можна спостерігати успішний пошук стратегії хижаком в продовж всього експерименту, але при цьому жертва продемонструвала задовільний результат. Дані про тривалість епізоду є досить нерівномірними, що свідчить про активний і успішний пошук стратегій агентами. Цьому сприяло використання параметризованої нагороди. Також важливу роль відігравала інтерпретація станів у вигляді координат положень, що буде детальніше розглянуто в наступному експерименті.

### Ефективності агентів, один алгоритм, Нагорода+, Стан-2

Був проведений експеримент, використовуючи лише A2C. Отримані метрики виражені графіком на рисунку №5.6 *червоними* лініями.

Результати цього експерименту свідчать про вплив параметризованої нагороди для пошуку оптимальної стратегії агентами. Але надання станів у вигляді векторів напрямку та відносної швидкості позбавила жертву можливості оцінювати дистанцію між агентами в обмін на легше сприйняття відносного положення. Таким чином процес пошуку оптимальної стратегії жертвою погіршився. Хижак використовує цю ваду і отримує кращі результати.

### Ефективності агентів, різні алгоритми, Нагорода+, Стан-1

Був проведений експеримент, використовуючи алгоритм A2C в якості хижака і DQN в якості жертви. Отримані метрики виражені графіком на рисунку №5.8.

Изображение выглядит как текст, карта

Автоматически созданное описание

Рисунок 5.7 – Результат експериментів №7,8

Незважаючи на погані результати алгоритму DQN в експерименті перевірки здатності вирішувати задачу, ці результати не були нульовими. До того ж використовуючи інший підхід до пошуку стратегії процес рішення задачі агентами може значно відрізнятися. Тому виникає необхідність виконати подібні експерименти, щоб виявити нові залежності або підтвердити попередні.

Показник відсотку успішних для хижака епізодів показує різницю між результатами наступного експерименту. Ця різниця підтверджує висновки щодо станів виражених координатами, але у випадку ефективності використовуваних алгоритмів. Ці дані показали, що DQN може показувати певні задовільні результати у ролі жертви. Також про це свідчать дані про тривалість епізодів. Вони дуже нерівномірні і виражають часту зміну переваги одного агента над іншим. Дані про сумарну нагороду демонструють часті стрибки значень нагороди. Це зумовлено явищем знаходженням деякої ефективної стратегії жертвою, але хижак швидко вирішує цю проблему.

### Ефективності агентів, різні алгоритми, Нагорода+, Стан-2

Був проведений експеримент, використовуючи алгоритм A2C в якості хижака і DQN в якості жертви. Отримані метрики виражені графіком на рисунку №5.8.

Результати цього експерименту підтверджують все висновки наведені вище про вплив вираження станів і нагород для процесу навчання агентів і не наводять ніяких нових випадків функціювання агентів. Відсоток успішних епізодів хижака прямує до 100%, вектори напрямків ослаблюють процес навчання жертви, сумарна нагорода є рівномірною.

# ВИСНОВКИ

У ході виконання дипломної роботи були реалізовані та застосовані алгоритми машинного навчання з підкріпленням у взаємодії з елементами глибокого машинного навчання. А саме – алгоритми Deep Q-Learning та Advantage Actor-Critic. Описано принцип їх роботи та теоретичні основи, які в них закладені. Було проаналізовано роль математичних моделей глибокого навчання для застосування в описаних алгоритмах навчання з підкріпленням.

Для випробування алгоритмів було реалізовано середовище диференціальної гри переслідування та визначено необхідні деталі задачі. Були запропоновані необхідні характеристики середовища такі, як визначення умов отримання нагороди, і надано обґрунтування їх вибору.

Обидва алгоритми продемонстрували деяку здатність вирішувати поставлену задачу переслідування. Було проаналізовано їх роботу і порівняно ефективність пошуку ефективної стратегії в заданому середовищі. Дослідження показало перевагу алгоритму A2C над DQN у швидкості пошуку оптимальної стратегії у всіх випадках його застосування. Проте спостереження показали, що алгоритмам необхідно досить багато часу, щоб їх поточна стратегія почала сходитися до оптимальної. Достатньо висока розмірність простору можливих станів і відтермінована нагорода є досить сильною перешкодою для алгоритмів навчання з підкріпленням, яка була подолана шляхом формування параметризованої нагороди.

Результати цієї роботи можуть бути застосовані для подальшого дослідження ефективності методів машинного навчання з підкріпленням для інших класів ігор та в задачах оптимального керування.

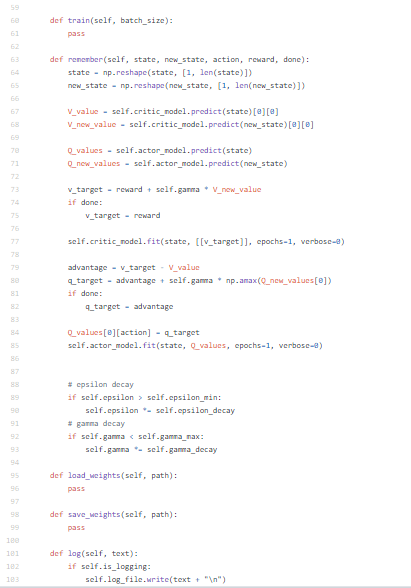
# ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Reinforcement Learning: An Introduction / Sutton Richard S., Barto Andrew G.– MIT Press, Cambridge, Massachusetts, 2018 – 552 p. – ISBN 978-026- 203-924-6.
2. An Introduction to Deep Reinforcement Learning / Francois-Lavet Vincent, Islam Riashat, Bellemare Marc G., Joelle Pineau – Now Foundations and Trends, 2018 – 156 p. – ISBN 978-168-083-538-0.
3. Rupesh Kumar Srivastava, Pranav Shyam, Filipe Mutz. Wojciech Jaśkowski, Jürgen Schmidhuber. Training Agents using Upside-Down Reinforcement Learning. The Swiss AI Lab IDSIA (2019).
4. On the Gittins Index for Multi-Armed Bandits / Weber R. / The Annals of Applied Probability. Vol. 2, No. 4. – Institute of Mathematical Statistics, 1992. – P. 1024-1033. – ISSN 1050-5164
5. Yonghui Wu, Mike Schuster, Zhifeng Chen. Google's Neural Machine Translation System. (2016)
6. Sankar, Swami & Jain, Arpit & Chellappa, Rama & Lim, Ser-Nam. Regularizing deep networks using efficient layerwise adversarial training. (2017).
7. An Industrial Strength Audio Search Algorithm (2003). Shazam Entertainment, Ltd.
8. Applications of Reinforcement Learning in Real World. [Електронний ресурс] Режим доступу до статті: <https://towardsdatascience.com/applications-of-reinforcement-learning-in-real-world-1a94955bcd12>
9. Deep Learning in Neural Networks: An Overview / Schmidhuber J. / Neural Networks. Vol. 61 – Elsevier, January 2015. – P. 85-117. – ISSN 0893-6080.
10. Reinforcement Learning: An Introduction - Richard S. Sutton and Andrew G. Barto [Електронний ресурс] Режим доступу до матеріалів: <http://incompleteideas.net/book/bookdraft2018jan1.pdf>
11. Ian Goodfellow, Yoshua Bengio, Aaron Courville. Deep Learning Book. MIT Press, Adaptive Computation and Machine Learning series (2016)
12. Robert H. Crites, Andrew G. Barto. “An Actor/Critic Algorithm that is Equivalent to Q-Learning”. [Електронний ресурс] Режим доступу до статті: <https://papers.nips.cc/paper/916-an-actorcritic-algorithm-that-is-equivalent-to-q-learning.pdf>
13. Practical Machine Learning Tutorial with Python Introduction. [Електронний ресурс] Режим доступу до матеріалів: <https://pythonprogramming.net/machine-learning-tutorial-python-introduction/>
14. Q-Learning introduction and Q Table - Reinforcement Learning [Електронний ресурс] Режим доступу до матеріалів: <https://pythonprogramming.net/q-learning-reinforcement-learning-python-tutorial/>
15. UCL Course on RL - David Silver. [Електронний ресурс] Режим доступу до матеріалів: <https://www.davidsilver.uk/teaching/>
16. Scaling Multi-Agent Reinforcement Learning - The BAIR Blog [Електронний ресурс] Режим доступу до статті: <https://bair.berkeley.edu/blog/2018/12/12/rllib/>
17. Understanding Actor Critic Methods and A2C. [Електронний ресурс] Режим доступу до матеріалів: <https://towardsdatascience.com/understanding-actor-critic-methods-931b97b6df3f>
18. OpenAI Baselines: ACKTR & A2C – Open AI. [Електронний ресурс] Режим доступу до матеріалів: <https://openai.com/blog/baselines-acktr-a2c/>

# ДОДАТОК А. Програмний код агентів середовища

A2C:





DQN:



