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:

ВИКОРИСТАННЯ МЕТОДІВ НАВЧАННЯ З ПІДКРІПЛЕННЯМ ДЛЯ ГЕНЕРАЦІЇ ОПТИМАЛЬНОЇ ПОВЕДІНКИ АГЕНТІВ

Виконав студент 4-го курсу
Дмитрий БЕРНАДА _____
(підпис)

Науковий керівник:
Доцент
Ярослав ЛІНДЕР _____
(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

« __ » _____ 2023 р., протокол № __

Завідувач кафедри
Олександр ПРОВОТАР _____
(підпис)

Київ - 2023

РЕФЕРАТ

Обсяг роботи 40 сторінок, 5 рисунка, 9 використаних джерел.

НАВЧАННЯ З ПІДКРІПЛЕННЯМ, АЛГОРИТМ PPO, МАШИННЕ НАВЧАННЯ, MOUNTAIN CAR PROBLEM

Об'єкт дослідження або розроблення: Використання методів навчання з підкріпленням для генерації оптимальної поведінки агентів в грі Mountain Car Problem.

Мета роботи: Метою даної роботи є застосування методів навчання з підкріпленням для вирішення задачі Mountain Car Problem. Задача полягає в тому, щоб навчити агента (автомобіль) пройти по гірському шляху, враховуючи обмеження його руху. Головна мета полягає в тому, щоб агент навчився максимально оптимально переміщатися на цьому шляху.

Методи та інструменти розроблення: У даній роботі використовуються методи навчання з підкріпленням, зокрема можна використовувати алгоритми, такі як Q-навчання, SARSA, або Deep Q-навчання з використанням нейронних мереж. Для розробки та експериментів можуть використовуватися платформи для реалізації алгоритмів навчання з підкріпленням, наприклад, TensorFlow або PyTorch.

Результати та їх новизна: Результатом даної роботи є навчені агенти, які здатні ефективно переміщатися по гірському шляху в задачі Mountain Car Problem. Це досягається за допомогою використання методів навчання з підкріпленням, які дозволяють агентам самостійно вчитися та виробляти оптимальну стратегію. Новизна роботи полягає в застосуванні цих методів до конкретної задачі гірського автомобіля.

Інформація щодо впровадження: Розроблені методи можуть бути використані для навчання агентів у реальних середовищах, де вимагається оптимальна поведінка при подоланні подібних проблем з навігацією.

Взаємозв'язок з іншими роботами: Робота побудована на попередніх дослідженнях з використання методів навчання з підкріпленням, але вдосконалює їх застосування в контексті гри Mountain Car Problem.

Рекомендації щодо використання результатів роботи: Отримані стратегії поведінки агентів можуть бути використані для розв'язання подібних задач навігації, де потрібно здійснювати оптимальні дії для досягнення поставлених цілей.

Сфера застосування: Результати роботи можуть бути використані в галузі розробки автономних систем навігації, робототехніки та штучного інтелекту.

Значимість роботи: Робота вносить внесок у розвиток методів навчання з підкріпленням та показує їх ефективність у вирішенні складних задач навігації, таких як гра Mountain Car Problem.

Висновки та пропозиції щодо розвитку об'єкта дослідження (розроблення) й доцільності продовження досліджень або розробок: В якості алгоритму навчання було обрано Proximal Policy Optimization. Створено середовище, в якому агент може навчатися. Встановлено необхідні модулі та здійснена реалізація програми мовою Python. Отримані результати повністю відповідають темі кваліфікаційної роботи. Результатом роботи є програма для тренування ігрового агенту, що дозволяє запускати його з різними параметрами, наприклад, кількість вершин сплайну, прискорення, сила гравітації, початкове положення.

Зміст

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАЧЕННЯ

IDE (Integrated Development Environment) – система програмних засобів, що використовується для розробки програмного забезпечення (дослівний переклад: інтегроване середовище розробки).

IDLE (Integrated Development and Learning Environment) – інтегроване середовище розробки та навчання на мові Python [1].

API (Application Programming Interface) – інтерфейс взаємодії між сайтом і сторонніми програмами та серверами (дослівний переклад: прикладний програмний інтерфейс) [2].

ML (Machine Learning) – підрозділ штучного інтелекту, що застосовує методи побудови алгоритмів, здатних навчатися (дослівний переклад: машинне навчання) [3].

PPO (Proximal Policy Optimization) - це нещодавній прогрес у сфері навчання з підкріпленням, який забезпечує покращення оптимізації політики довірчого регіону (TRPO). Цей алгоритм був запропонований у 2017 році і показав чудову продуктивність, коли він був реалізований OpenAI.

ВСТУП

Оцінка сучасного стану об’єкта дослідження або розробки. Завдяки прогресу глибокого навчання в галузі reinforcement learning почали з'являтися агенти, які здатні навчатися і діяти у великих обсягах вхідних даних. Наприклад, алгоритми, такі як Deep Q-Network (DQN), досягли людського рівня в грі на багатьох класичних іграх ATARI 2600, навчаючись безпосередньо з сирих пікселів без додаткового нагляду. Однак, більшість реальних проблем пов'язані з оточеннями, де не всі можливі стани можуть бути змодельовані і спостережені, а набір доступних дій є великим. Для вирішення цих проблем агенти повинні бути здатні опрацьовувати часткові спостереження, працювати зі структурованою та складною динамікою і взаємодіяти з шумними та високорозмірними інтерфейсами управління.

Актуальність роботи та підстави для її виконання. На сьогоднішній день вже існує значна кількість ігрових агентів, які виявляються кращими за людей у виконанні своїх завдань. Це досягнення стало можливим завдяки швидкодії сучасних комп'ютерів і вдосконаленню алгоритмів машинного навчання, що дозволяють створювати більш потужних агентів. Основною метою цієї кваліфікаційної роботи було дослідження методів і принципів функціонування сучасних алгоритмів, а також виявлення можливостей для їх поліпшення та розв'язання інших проблем.

Мета й завдання роботи. Мета кваліфікаційної роботи полягає в створенні середовища для тренування ігрового агенту для проходження гри з найкращим результатом. Для цього необхідно:

1. Обрати алгоритм для реалізації програми.
2. Створити середовище для того, щоб агент міг в ньому навчатися
3. Ознайомитись з необхідними бібліотеками і компонентами, підключити їх.
4. Створити програму, що дозволить ігровому агенту тренуватися проходити маршрут за найменший час.

Об’єкт, методи й засоби дослідження або розроблення. Базова модель навчання з підкріпленням складається з:

1. множини станів середовища
2. множини дій
3. правил переходу між станами;
4. правил, які визначають *скалярну безпосередню винагороду* переходу;
5. правил, які описують, що спостерігає агент.

Ці правила часто є стохастичними. Спостереження зазвичай включає в себе скалярну безпосередню винагороду, пов'язану з крайнім переходом. У багатьох працях також вважають, що агент спостерігає поточний стан середовища, в разі чого говорять про повну спостережуваність тоді як в іншому разі говорять про часткову спостережуваність. Іноді множина доступних агентові дій є обмеженою (наприклад, ви не можете витрачати більше грошей, ніж маєте).

Агент навчання з підкріпленням взаємодіє зі своїм середовищем у дискретні моменти часу. В кожен момент часу агент отримує спостереження, яке зазвичай включає винагороду. Потім він обирає дію з множини доступних дій, яка відтак відправляється до середовища. Середовище переходить до нового стану і визначається винагорода, пов'язана з переходом. Метою агента навчання з підкріпленням є збирати якомога більше винагороди. Агент може обирати будь-яку дію як функцію історії, і може навіть робити свій вибір дії випадковим.

Коли продуктивність агента порівнюється з продуктивністю агента, який діє оптимально від початку, то різниця в продуктивності призводить до поняття смутку. Зверніть увагу, що, щоби діяти майже оптимально, агент мусить розуміти довготермінові наслідки своїх дій: щоби максимізувати свій майбутній дохід, мені краще зараз піти до школи, хоча пов'язана з цим безпосередня грошова винагорода може бути від'ємною.

Можливі сфери застосування. Навчання з підкріпленням застосовується там, де потрібно порівняти відстрочену вигоду – ціль – із ситуативним прийняттям рішення. Цей вид навчання вирішує складне завдання співвіднесення негайних дій із відстроченою віддачею, що вони виробляють. Як і людям, алгоритми підкріплення навчання іноді доводиться чекати, щоб побачити плоди своїх рішень. У таких випадках часто складно зрозуміти, яка дія призводить до якого результату. Області практичного застосування reinforcement learning:

1. Постановка цілей
2. Планування
3. Системи сприйняття
4. Боти для комп'ютерних ігор
5. Трейдингові боти
6. Чат боти, які навчаються від діалогу до діалогу

Взаємозв’язок з іншими роботами. Стратегії в режимі реального часу (RTS) історично є сферою інтересів у дослідницької спільноти. Цей тип ігор має на меті імітувати контроль кількох військових одиниць у різних масштабах та рівнях складності, зазвичай на 2D карті фіксованого розміру, у дуелі або у невеликих командах. Мета гравця – збирати ресурси, які можна використовувати для розширення їхнього контролю на карту, створювати будівлі та підрозділи для боротьби з розгортанням ворога, і зрештою знищити супротивників. Одна з причин, чому ці ігри настільки важкі в тому, що високий рівень гри вимагає мислення та дії на різних рівнях абстракції. Гра вимагає управління ресурсами, складання плану будівництва, визначення пріоритетів розвитку технологій, розвідки, мікроуправління військами, а також загальна стратегія та способи протидії стратегії супротивника. Побудувати AI, який зможе зробити все це добре буде дуже тяжко. Метою цієї роботи було відтворити результати оригінальної роботи, спробувавши оптимізувати алгоритм меншого використання обчислювальних ресурсів.

РОЗДІЛ 1. МАШИННЕ НАВЧАННЯ

1.1 Поняття машинного навчання

Машинне навчання (далі говоритимемо ML) – підрозділ штучного інтелекту, що застосовує методи побудови алгоритмів, здатних навчатися [3].

ML тісно пов'язане з обчислювальною статистикою, яка також зосереджується на прогнозуванні шляхом застосування комп'ютерів. ML має тісні зв'язки з математичною оптимізацією, яка забезпечує цю галузь методами, теорією та прикладними областями.

1.2 Способи машинного навчання

Виділяють декілька способів ML:

1. Кероване навчання - підкатегорія машинного навчання, в якій агент вчиться приймати рішення на основі отриманої інформації та отримувати підкріплення або штрафи в залежності від вибраної дії.

   Зазвичай використовується модель Маркова з посиленням (Markov Decision Process, MDP), що допомагає формалізувати задачу керованого навчання. Основні компоненти MDP включають:

   Стан (**S**): Множина можливих станів системи, в якій знаходиться агент. Стани можуть бути дискретними або неперервними.

   Дії (**A**): Множина можливих дій, які агент може виконати в кожному стані.

   Функція переходу (**T**): Відображення, що визначає ймовірності переходу агента з одного стану в інший після виконання певної дії.

   Функція винагороди (**R**): Відображення, що визначає нагороду або штраф, який отримує агент після виконання певної дії в певному стані.

   Політика ($\boldsymbol{\pi}$): Стратегія агента, яка визначає, яку дію вибрати в кожному стані з метою максимізації нагороди.

   Одна з основних мет в керованому навчанні - знайти оптимальну стратегію ($\boldsymbol{\pi}^*$) - стратегію, яка максимізує очікувану нагороду в довгостроковому перспективі. Це можна знайти за допомогою

алгоритмів підкріпленого навчання, таких як Q-навчання або методи згорткового навчання, які використовують функцію цінності, що оцінює важливість дій та станів.

Таким чином, формула для керованого навчання може бути представлена наступним чином:

$$\pi^* = \arg\max \Sigma(R(s, a) + \gamma\Sigma T(s, a, s')V(s'))$$

де π^* - оптимальна стратегія, $R(s, a)$ - функція винагороди, $T(s, a, s')$ - функція переходу, $V(s')$ - функція цінності наступного стану s', γ - дисконтний фактор, який враховує майбутню винагороду.

2. Спонтанне навчання, також відоме як навчання без учителя, є підгалуззю машинного навчання, де модель аналізує дані без наявності попередньо визначених міток чи цілей. У цьому випадку модель намагається знайти приховані закономірності, структури або шаблони у вхідних даних.

   Одним з основних підходів у спонтанному навчанні є кластеризація, яка групує схожі елементи разом на основі їхньої схожості або взаємодії. Іншим підходом є виявлення асоціативних правил, яке виявляє зв'язки та залежності між різними елементами в наборі даних.
3. Навчання з підкріпленням – частковий випадок керованого навчання, але «вчителем» є середа. Машина не має інформації про середовище, але має можливість робити в ній будь-які дії. Середа реагує на ці дії і таким чином надає машині дані, які дозволяють їй реагувати на них і вчитися. Фактично машина і середовище утворюють систему зі зворотним зв'язком.

1.3 Методи машинного навчання

Виділяють шість основних методів ML [4]:

1. Регресія. Цей метод ґрунтується на базових принципах фізики, які дозволяють знайти кореляцію між двома змінними для визначення причинно-наслідкового зв'язку. На основі цих змінних можна побудувати графік і зробити прогноз. Існують різні форми регресії,

починаючи від лінійної і закінчуючи комплексною, обчисленням поліноміальних даних і представленням. Поширеними прикладами лінійної регресії є прогноз погоди, тенденції ринку, виявлення потенційних ризиків.

2. Класифікація. Цей метод, завдяки обчисленню ймовірності, розділяє об’єкти по групах відповідно до наперед визначених ознак. Тобто, якщо в якості вхідних даних ми маємо категорії «погода», «традиції», «тварини» і деякий текстовий документ, методом класифікації ми можемо або визначити, до якої з цих трьох тем відноситься поданий текст, або, якщо він на іншу тему, не віднести до жодної.
3. Кластеризація. Цей метод, як і класифікація, розділяє об’єкти по групах, але, на відміну від класифікації, не потребує визначення ознак. Метод кластеризації формує групи схожих за деякою ознакою об’єктів, але не називає цю ознаку.
4. Скорочення розмірів. Цей метод ґрунтується на зменшенні випадкових величин при категоризації даних. Це дозволяє відмовитися від несуттєвих змінних. Наприклад, якщо ви хочете передбачити ризик виникнення раку у групи людей, фактором споживання кави можна знехтувати, хоч вона і є канцерогенним продуктом. Найбільш поширеним прикладом зменшення розмірів є процес класифікації електронної пошти, який використовується для сортування спам-повідомлень.
5. Метод ансамбля. Цей метод сумує дані з впровадженням змінних прогнозування з різних моделей. Таким чином, він об'єднує різні моделі прогнозування для формування високоточних і оптимізованих результатів прогнозування. Метод ансамбля використовується для прийняття рішень при розгляданні різних факторів. В якості простого прикладу застосування можна навести приклад покупки квартири. Метод ансамблю обере найкращий варіант на основі різних факторів, таких як вартість, місцезнаходження, рівень безпеки району, відстань до метро тощо.

6. Нейронні мережі. На відміну від лінійних моделей, нейронна мережа заснована на складній, дивізійній структурі даних. Вона включає в себе кілька шарів параметра для забезпечення однакового і точного виведення. Проте, модель все ще базується на лінійній регресії, але використовує кілька прихованих шарів.

Для групування текстів за семантичними ознаками доцільно використовувати метод кластеризації, так як на вхід не поступають жодні варіанти, відповідно до яких треба розподіляти текстові документи, а на виході повинні мати декілька груп текстів, розподілених за їхніми семантичними ознаками.

РОЗДІЛ 2. НАВЧАННЯ З ПІДКРІПЛЕННЯМ

2.1 Визначення

Навчання з підкріпленням (reinforcement learning, RL) займається завданнями послідовного прийняття рішень. Багато реальних проблем, що виникають у комп'ютерних іграх, спорті, водінні автомобіля, оптимізації товарних запасів, роботизованому управлінні, тобто скрізь, де діють люди і машини, можуть бути представлені в подібному вигляді.

Вирішуючи кожну з цих завдань, ми переслідуємо якусь мету: перемогти в грі, безпечно доїхати до пункту призначення або мінімізувати вартість будівельних матеріалів. Ми робимо дії і отримуємо з навколишнього світу відповідь про те, наскільки близькі до мети: поточний рахунок, відстань до пункту призначення або ціну одного виробу. Досягнення мети, зазвичай, передбачає виконання низки дій, кожна з яких змінює навколишній світ. Ми спостерігаємо ці зміни світу, а також отримуємо зворотну інформацію, спираючись на яку приймаємо рішення про наступний крок.

Уявіть, що ви на вечірці, а ваш друг приніс флагшток і пропонує натиск якомога довше балансувати їм, поставивши на долоню. Якщо ви ніколи досі не робили цього, то початкові спроби будуть не надто вдалими. Ймовірно, перші кілька хвилин ви витратите на те, щоб методом спроби помилок відчути флагшток, адже він весь час падає. Ці помилки дозволять накопичити корисну інформацію і набути інтуїтивного розуміння того, як утримувати флагшток у рівновазі. Ви дізнаєтеся, де знаходиться його центр мас, з якою швидкістю він нахиляється, при якому куті нахилу падає, як швидко ви може підлаштуватися і т. д. Ви використовуєте цю інформацію, щоб внести корективи при наступних спробах, вдосконалюватися і знову коригувати свою поведінку. Ви навіть не помітите, як почнете утримувати рівновагу по 5, 10, 30 с, 1 хв і т. д. Цей процес наочно демонструє, як працює навчання з підкріпленням. У навчанні з підкріпленням вас можна назвати агентом, а флагшток і ваше оточення - Середовищем.

Прикладом може бути ігрова версія цього сценарію під назвою CartPole (рис. 1.1). Агент керує ковзним уздовж осі візком так, щоб утримувати стрижень у вертикальному положенні протягом заданого часу. Реальні здібності людей набагато ширші, адже ми можемо інтуїтивно розуміти фізичну сторону того, що відбувається. А можемо і застосувати навички виконання схожих завдань, таких як балансування підносом, вставленим напоями. Але, по суті, формулювання завдання залишається тим самим. Мета - утримання рівноваги стержня протягом 200 кроків за допомогою управління переміщення візка вправо і вліво.

В навчанні з підкріпленням вивчаються подібні завдання, а також методи, за допомогою яких штучні агенти вчаться їх вирішувати. Це область штучного інтелекту, яка сягає теорії оптимального управління і використовує поняття марківського процесу прийняття рішень (МППР). RL з'явився в 1950-х роках у контексті динамічного програмування та квазілінійних рівнянь завдяки Річарду Беллману. Його ім'я буде ще не раз згадано при вивченні того, хто отримав популярність у навчанні з підкріпленням рівняння Беллмана. Завдання RL можуть бути представлені як система, що складається з агента і середовища.

Середовище надає інформацію, що описує стан системи. Агент взаємодіє з середовищем, спостерігаючи стан і використовуючи цю інформацію при виборі дії. Середовище приймає дію та переходить у наступний стан, а потім повертає агенту наступний стан та винагороду. Коли цикл «стан → дія → винагорода» завершено, передбачається, що зроблено один крок. Цикл повторюється, поки середовище не завершиться, наприклад, коли завдання вирішено. Функція, відповідно до якої агент вибирає дії, називається стратегією. Формально стратегія - це функція, що відображає безліч станів у безліч дій. Дія змінює середовище і впливає на те, що агент спостерігає і робить далі. Обмін інформацією між агентом і середовищем розгортається в часі, проте його можна розглядати як процес послідовного прийняття рішень. У завданнях RL є цільова функція, яка є сумою отриманих

агентом винагород. Завдання агента - максимізувати цільову функцію, вибираючи найкращі дії. Він навчається цьому, взаємодіючи з середовищем методом пробі помилок, і використовує заохочуючі сигнали для підкріплення кращих дій. Середу можна розглядати як усе, що є не є агентом.

Наприклад, для їзди на велосипеді можливі кілька різних, але рівнозначних визначень агента і середовища. Якщо вважати агентом розумовий процес, то середовищем будуть фізичне тіло, велосипед і дорога, діями - нервові імпульси, що пересилаються від головного мозку до м'язів, а станами - що надходять в мозок сигнали від органів чуття. По суті, система навчання з підкріпленням реалізує цикл управління з зворотним зв'язком, де агент і середовище взаємодіють і обмінюються сигналами, причому агент намагається максимізувати цільову функцію. Сигнали - це трійка (st, at, rt), що відповідає стану, дії та винагороди, а індекс t вказує на номер кроку (момент часу), на якому виник сигнал. Кортеж (st, at, rt) називається прецедентом або частиною одержуваного агентом досвіду. Цикл управління може повторюватися до нескінченності або закінчитися після досягнення або кінцевого стану, або максимального значення кроку t = T. Тимчасовий горизонт t = 0 до моменту завершення середовища носить назву епізоду. Траєкторія - це послідовність прецедентів, або частина досвіду, накопиченого протягом епізоду, τ = (s0, a0, r0), (s1, a1, r1) ... Зазвичай агенту для навчання хорошої стратегії потрібно від сотень до мільйонів епізодів в залежності від складності завдання .

2.2 Навчальні функції у навчанні з підкріпленням

Існує три основні функції, які вивчаються у навчанні з підкріпленням:

1. Стратегія π, яка зіставляє стан дію: a ~ π(s).

2. Функція корисності $V^{\pi}(s)$ або $Q^{\pi}(s,a)$ для обчислення очікуваної віддачі $\mathbb{E}_{\tau}[R(\tau)]$.

3. Модель середовища P(s' | s, a).

Стратегія π - це те, яким чином агент робить дії в середовищі, щоб максимізувати цільову функцію. Згідно з циклом управління агент повинен робити дії на кожному кроці після спостереження стану s. Стратегія має фундаментальне значення для циклу управління, оскільки генерує дії, які змушують його продовжуватися. Стратегія може бути стохастичною. Це означає, що вона може з певною ймовірністю давати на виході різні дії для одного стану. Це може бути записано як π (a | s) і означає ймовірність дії a для даного стану s. Дія, обрана за стратегією, записується як a ~ π(s).

Функції корисностей подають інформацію про мету. Вони допомагають агенту зрозуміти, наскільки хороші стани і доступні дії з точки зору очікуваної віддачі. Функція корисності V^{π}, оцінює, наскільки хорошим або поганим є стан. V^{π} дає оцінку очікуваної віддачі відбуття в положенні s, припускаючи, що агент продовжує діяти відповідно до своєї поточної стратегії π. Віддача розраховується, починаючи з поточного стану s до кінця епізоду. Це прогнозна оцінка, оскільки не враховуються всі винагороди, отримані до стану s. Розглянемо простий приклад, який дозволить отримати уявлення про функцію корисності V^{π}. На рис. 1 зображено дискретне середовище з кінцевим числом станів, в якому агент може переміщатися з клітини в клітину по вертикалі горизонталі. Кожна клітина - це стан, з яким пов'язана винагорода, як показано на рис. 1, ліворуч.

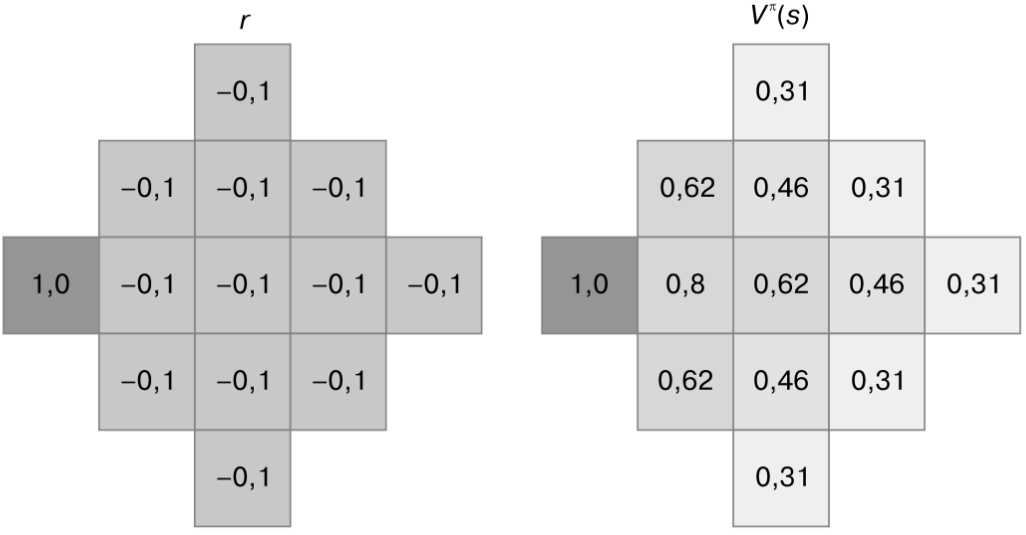

Рисунок 1 Винагороди r та корисності Vπ(s) для кожного стану s у простій клітинній середовище. Тут застосовується стратегія π, коли завжди вибирається найкоротший шлях до цільового стану з r = +1

Середовище завершується, коли агент потрапляє в цільовий стан з винагородою r = +1. Функція корисності V^{π} завжди залежить від конкретної стратегії π. У цьому прикладі взято стратегію π, за якою завжди вибирається найкоротший шлях до цільового стану. Якщо стратегія буде іншою - наприклад, переміщення тільки вправо, - то значення корисностей будуть іншими. Це показує, що функція корисності є прогнозною і допомагає агенту розрізняти стани з однаковою винагородою.

Чим ближче агент до цільового стану, тим вище корисність аналізованого стану. Функція корисності Q^{π} оцінює, наскільки хороша чи погана пара «стан - дія». Q^{π} дає оцінку очікуваної віддачі від вибору дії a в стані s, припускаючи, що агент продовжує діяти відповідно до своєї поточної стратегії π. За аналогією з V віддача розраховується, починаючи з поточного стану s і до кінця епізоду. Це теж прогнозна оцінка, оскільки не враховуються всі винагороди, отримані до стану s. а) надає інформацію про середовище. Сформувавши цю функцію, агент знаходить здатність передбачати наступний стан s', який перейде середовище після вибору дії a в стані s. Застосовуючи отриману функцію переходів, агент може уявити наслідки дій, не вступаючи в дійсну взаємодію з середовищем. Надалі він може використовувати цю інформацію для планування оптимальних дій.

2.3 Proximal Policy Optimization Algorithms

Стаття Proximal Policy Optimization Algorithms [124] була опублікована в 2017 році Шульманом та ін. PPO - легко обчислювально маловитратний алгоритм, що реалізується, без вибору δ. Завдяки цьому він став одним з найпопулярніших алгоритмів градієнта стратегії.

PPO - це сімейство алгоритмів, які вирішують задачу оптимізації стратегії,обмежену довірчою областю, за допомогою простих і ефективних евристик. Є два варіанти: перший заснований на адаптованій штрафній функції відстані Кульбака - Лейблера, а другий - на усіченій цільовій функції. Обидва вони представлені в даному розділі. .

Насамперед спростимо сурогатну цільову функцію $J^{CPI}(\theta)$, записавши $r_t(0) = \frac{\pi_{\theta(a_t|s_t))}}{\pi_{\theta_{old}(a_t|s_t))}}$.Для стислості також обозначим $A_t^{\pi_{\theta_{old}}}$ як A_t, тому що відомо, що значення переваги завжди розраховуються за допомогою більш старої стратегії $\pi_{\theta_{old}}$:

$$J^{\mathrm{CPI}}(\theta) = \mathbb{E}_t\left[\frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta_{\mathrm{old}}}(a_t \mid s_t)} A_t^{\pi_{\theta_{\mathrm{old}}}}\right] = \mathbb{E}_t\left[r_t(\theta) A_t\right].$$

Перший варіант PPO називається PPO з адаптивною штрафною функцією відстані Кульбака - Лейблера. Це перетворення обмеження відстані Кульбака - Лейблера $\mathbb{E}_t\left[\mathrm{KL}\left(\pi_\theta(a_t \mid s_t) \| \pi_{\theta_{\mathrm{old}}}(a_t \mid s_t)\right)\right] \leq \delta$ в адаптивну штрафну функцію відстані Кульбака - Лейблера, яка віднімається від значення переваги, скоригованого ваговим коефіцієнтом значущості. Математичне очікування отриманого виразу - нова цільова функція, яку потрібно максимізувати.

$$J^{\mathrm{KLpen}}(\theta) = \max_\theta \mathbb{E}_t\left[r_t(\theta) \mathrm{A}_t - \beta \mathrm{KL}\left(\pi_\theta(a_t \mid s_t) \| \pi_{\theta_{\mathrm{old}}}(a_t \mid s_t)\right)\right].$$

Данне рівняння відоме як сурогатна цільова функція зі штрафом по відстані Кульбака - Лейблера. β - адаптивний коефіцієнт, що контролює

розмір штрафу по відстані Кульбака - Лейблера. Чим менше β, тим вище схожість стратегій.

Одна з труднощів застосування постійного коефіцієнта в тому, що через відмінність між характеристиками різних завдань важко підібрати одне значення, що підходить у всіх випадках. Навіть для одного завдання при ітерації за стратегіями відбувається зміна ландшафту функції втрат, і значення β, яке раніше було робочим, може надалі стати невідповідним, тобто його потрібно адаптувати до цих змін. У статті про PPO як вирішення цієї проблеми пропонується засноване на евристиці правило оновлення β , Що дозволяє адаптувати його з плином часу. Коефіцієнт β оновлюється після кожного оновлення стратегії і на наступній ітерації береться його нове значення. Правило оновлення для наведено в алгоритмі 7.1. Воно може бути використане як процедура в алгоритмі PPO, який буде представлений пізніше в цьому розділі.

1. Встановити цільове значення δtar для маточікування відстані Кульбака — Лейблера.
2. Ініціалізувати β випадковим значенням
3. Використовувати велику кількість епох стохастичного градієнтного спуску міні-пакетами, оптимізувати сурогатну цільову функцію зі штрафом по відстані Кульбака — Лейблера:

$$J^{KLpen}(\theta) = E_t\left[r_t(\theta)\,A_t - \beta KL\left(\pi_\theta(a_t \mid s_t) \,||\, \pi_{\theta_{old}}(a_t \mid s_t)\right)\right]$$

Обрахувати $\delta = E_t\left[KL\left(\pi_\theta(a_t \mid s_t) \,||\, \pi_{\theta_{old}}(a_t \mid s_t)\right)\right]$:

if $\delta < \delta tar / 1{,}5$ then

$\beta \leftarrow \beta/2$

else if $\delta > \delta tar \times 1{,}5$ then

$\beta \leftarrow \beta \times 2$

else

pass

end if

Наприкінці кожної ітерації обчислюємо δ і порівнюємо його з бажаним цільовим δtar. Якщо δ менше, ніж δtar з деяким запасом, то знижуємо штраф на відстані Кульбака — Лейблера шляхом зменшення β. Але якщо δ більше, ніж δtar з деяким запасом, збільшуємо штраф, підвищуючи β. Конкретні значення для визначення величини запасу та правил оновлення для β підбираються досвідченим шляхом. Автори вибрали 1,5 і 2 відповідно, але виявили, що алгоритм не дуже чутливий до цих значень. Крім того, було помічено, що відстань Кульбака - Лейблера періодично сильно відхиляється від цільового значення δtar, але β швидко підлаштовується. Оптимальне значення δtar теж потрібно шукати емпірично.Перевага такого підходу – простота реалізації. Проте він не вирішує проблеми вибору цільового значення? Більш того, він може бути обчислювальним витратним через необхідність визначення відстані Кульбака - Лейблера.

PPO з усіченою сурогатною цільовою функцією вирішує це відмовою від обмеження відстані Кульбака - Лейблера і застосуванням більш простого перетворення сурогатної цільової функції з рівняння:

$$J^{\text{clip}}(\theta) = \mathbb{E}_t\left[\min\left(r_t(\theta)A_t, \text{clip}\left(r_t(\theta), 1-\varepsilon, 1+\varepsilon\right)A_t\right)\right].$$

Рівняння відоме як усічена сурогатна цільова функція. Значення ε обмежує околиця, через яку проводиться усічення, $|r_t(\theta)1| \leq \varepsilon$. Це налаштовуваний гіперпараметр, який може зменшуватися в процесі навчання. Перший член виразу під знаком мінімуму, min (·), - Це просто сурогатна цільова функція J^{CPI}. Другий член, $clip(r_t(\theta), 1-\varepsilon, 1+\varepsilon)A_t$, обмежує область значень J^{CPI} між $(1-\varepsilon)A_t$ і $(1+\varepsilon)At$. Коли rt(θ) знаходиться в межах інтервалу [1 - ε, 1 + ε], обидва члени під знаком мінімуму рівні.

Така цільова функція запобігає оновленню параметрів, які можуть викликати великі та ризиковані зміни стратегії πθ. Як кількісної міри великих змін стратегії застосовується імовірнісне відношення $r_t(\theta)$. Коли нова стратегія рівноцінна старої $r_t(\theta) = r_t(\theta_{old})\frac{\pi_{\theta(a_t|s_t))}}{\pi_{\theta_{old}(a_t|s_t))}} = 1$. Якщо нова стратегія відрізняється від старої, значення $r_t(\theta)$ відхиляється від 1.Сутність тут у тому, щоб обмежити $r_t(\theta)$ до ε-околиці [1 – ε, 1 + ε]. Зазвичай максимізація сурогатної цільової функції J^{CPI} без обмежень може сприяти більшим оновленням стратегії. Це викликано тим, що одним із механізмів, за допомогою якого може бути покращена продуктивність поцільової функції, є великі зміни $r_t(\theta)$. Усіченням цільової функції ми усуваємо причини, що породжують великі оновлення стратегії, які викликали б вихід $r_t(\theta)$ з ε-околиці. Щоб зрозуміти, чому це відбувається, розглянемо випадок, коли $r_t(\theta)A_t$ приймає великі позитивні значення при A_t> 0, $r_t(\theta) > 0$ або A_t <0, $r_t(\theta)$ <0.

При A_t> 0, $r_t(\theta) > 0$, якщо $r_t(\theta)$ стає набагато більше 1, верхня межа усічення 1 – ε застосовується для обмеження зверху $r_t(\theta) \leq 1 + \varepsilon$, отже, $J^{clip} \leq (1 + \varepsilon)A_t$. З іншого боку, при $A_t < 0$, $r_t(\theta) < 0$, якщо $r_t(\theta)$ стає набагато менше 1, нижня межа усічення 1 - ε знову застосовується для обмеження зверху $J^{clip} \leq (1 - \varepsilon)$ At. При взятті мінімуму в рівнянні Jclip завжди обмежена зверху одним із двох способів. Крім того, взяття мінімуму також має на увазі, що Jclip - песимістична нижня межа початкової сурогатної цільової функції J^{CPI}. Таким чином, вплив $r_t(\theta)$ ігнорується при спробі поліпшити цільову функцію більш ніж на εA_t, але завжди враховується, коли з-за нього цільова функція погіршується. виходу rt(θ) із околиці [1 – ε, 1 + ε]. Отже, оновлення стратегії безпечні. Це дає нам підстави для багаторазового повторного використання вибраних траєкторій для оновлення параметрів. Таким чином підвищується ефективність вибірки алгоритму. Цільова функція залежить від , яка оновлюється після кожного кроку навчання. Значить, PPO — алгоритм навчання за актуальним досвідом, тому старі траєкторії повинні відкидатися після кожного кроку навчання. Усічена цільова функція J^{clip} обчислювально

маловитратна, дуже проста для розуміння, і для її реалізації потрібно лише кілька тривіальних перетворень початкової сурогатної цільової функції J^{CPI}. є кроки обчислення імовірнісного відношення $r_t(\theta)$ і значення переваги At. Однак це мінімальні розрахунки, необхідні для будь-яких алгоритмів оптимізації сурогатної цільової функції. Інші обчислення - це усічення і мінімізація, які виконуються за постійний час. Лейблер. Версія PPO з усіченням краще, тому що вона і простіше, і продуктивніше. [7]

РОЗДІЛ 3. СЕРЕДОВИЩЕ RLLIB

RLlib є потужною бібліотекою з підтримкою посиленого навчання (reinforcement learning), розробленою OpenAI для зручного використання у мові програмування Python. Вона надає набір інструментів, алгоритмів та моделей для побудови та тренування агентів з використанням методів підсилення.

RLlib має багато переваг, які сприяють швидкому розробленню і ефективному навчанню агентів в різних середовищах. Деякі з цих переваг:

1. Масштабованість: RLlib підтримує розподілене навчання, що дозволяє розподіляти обчислення між багатьма процесами та машинами. Це дозволяє прискорити тренування шляхом використання більшої обчислювальної потужності.

2. Висока продуктивність: Бібліотека оптимізована для швидкості виконання і надає високоефективні реалізації алгоритмів, зокрема алгоритми згорткових нейронних мереж, глибокого Q-навчання, а також еволюційного та генетичного навчання.

3. Гнучкість: RLlib надає широкий вибір алгоритмів та моделей, що дозволяє розробляти агентів для різних завдань. Вона включає в себе класичні алгоритми, такі як Q-навчання, Sarsa і A2C, а також сучасні алгоритми, такі як PPO, DDPG, SAC і TD3.

4. Мультиагентне навчання: RLlib підтримує навчання декількох агентів, що взаємодіють у спільному середовищі. Це дозволяє вирішувати завдання співпраці, конкуренції та комунікації між агентами.

5. Підтримка середовищ: RLlib може працювати з різними середовищами, включаючи OpenAI Gym, RoboSchool, PyBullet, Unity ML-Agents та багато інших. Вона надає простий інтерфейс для інтеграції з різними середовищами та дозволяє легко налаштовувати параметри навчання.

6. Інструменти для оцінки: RLlib надає інструменти для збору статистики, візуалізації результатів і зберігання прогресу навчання. Вона дозволяє аналізувати результати тренування і приймати рішення про подальші кроки для покращення агента.

7. Документація та підтримка: RLlib має докладну документацію, яка пояснює використання бібліотеки і надає приклади коду. Вона також має активну спільноту користувачів, яка може допомогти вирішити питання та проблеми.

3.2 Приклади використання бібліотеки

Тестування агента на середовищі після тренування:

```
trained_agent = trainer.get_policy()
obs = env.reset()
done = False
total_reward = 0

while not done:
   action = trained_agent.compute_single_action(obs)
   obs, reward, done, _ = env.step(action)
   total_reward += reward

print("Total reward:", total_reward)
```

Конфігурація більш складних параметрів навчання:

```
config = {
   "env": "CartPole-v1",
   "framework": "torch",
   "num_workers": 2,
   "model": {
       "fcnet_hiddens": [64, 64],
   },
   "optimizer": {
       "grad_clip": 0.5,
   },
   "lr": 0.001,
   "gamma": 0.99,
   "lambda": 0.95,
```

```
    "sgd_minibatch_size": 64,
}


trainer = PPOTrainer(config=config)
```

Параметри, що вказані в config, використовуються для налаштування агента, який навчатиметься з використанням алгоритму PPO (Proximal Policy Optimization). Ось пояснення деяких ключових параметрів:

- "env": Вказує середовище, на якому буде тренуватися агент. У даному випадку використовується середовище "CartPole-v1", що є класичним завданням у контексті посиленого навчання.
- "framework": Вказує фреймворк, який використовуватиметься для навчання моделі агента. У даному випадку використовується фреймворк "torch", що означає використання бібліотеки PyTorch.
- "num_workers": Вказує кількість робочих процесів, які будуть використовуватись для збору даних і обчислення градієнтів. У даному випадку використовується 2 робочих процеси.
- "model": Вказує параметри моделі агента. У даному випадку використовується модель з двома повнозв'язними шари (64 нейрони в кожному шарі).
- "optimizer": Вказує параметри оптимізатора, який використовується для навчання моделі. У даному випадку використовується градієнтний зріз з обмеженням (grad_clip) на величину градієнту, що регулює швидкість навчання.
- "lr": Вказує швидкість навчання (learning rate) для оптимізатора. У даному випадку використовується значення 0.001.
- "gamma": Вказує дисконтний фактор (discount factor), який враховує майбутні винагороди при прийнятті рішень. У даному випадку використовується значення 0.99, що вказує на високу вагу майбутніх винагород.

- "lambda": Вказує параметр λ, який використовується при обчисленні ваги між старим і новим логарифмом ймовірностей дій. У даному випадку використовується значення 0.95.
- "sgd_minibatch_size": Вказує розмір міні-пакета (mini-batch) для стохастичного градієнтного спуску. У даному випадку використовується розмір 64.

3.3 Аналоги

[8]Існує кілька аналогів бібліотеки RLlib, які також надають функціональність для розробки і тренування агентів з використанням посиленого навчання. Ось декілька з них, разом з їх перевагами та обмеженнями:

1. OpenAI Gym: OpenAI Gym є однією з найпопулярніших бібліотек для розв'язання завдань посиленого навчання. Вона надає широкий вибір середовищ, на яких можна тренувати агентів, і має простий інтерфейс для взаємодії з цими середовищами. Однак, OpenAI Gym не надає повних функціональних можливостей для алгоритмів навчання, тому вам доведеться власноручно реалізувати алгоритми навчання.

2. Stable Baselines: Stable Baselines є іншою популярною бібліотекою, яка забезпечує реалізацію різних алгоритмів посиленого навчання. Вона надає готові реалізації алгоритмів, таких як PPO, A2C, DQN та інші. Stable Baselines має простий інтерфейс, докладну документацію та приклади коду. Однак, Stable Baselines не має розподіленого навчання, тому вона не підтримує автоматичне масштабування тренування на кластерах.

3. DeepMind's Dopamine: Dopamine є фреймворком, розробленим DeepMind, який забезпечує реалізацію різних алгоритмів посиленого навчання. Він спроектований для дослідження алгоритмів і має простий інтерфейс. Dopamine також надає інструменти для оцінки результатів навчання та аналізує ефективність алгоритмів. Однак, Dopamine не має такого рівня розширюваності і гнучкості, як RLlib.

4. TensorForce: TensorForce є бібліотекою, яка пропонує реалізацію алгоритмів посиленого навчання на основі TensorFlow. Вона надає гнучкий інтерфейс для налаштування агентів і середовищ, а також підтримку розподіленого навчання. Однак, TensorForce може бути складним для використання, особливо для початківців.

Переваги використання бібліотек, які спеціалізуються на посиленому навчанні, включають наступне:

- Готові реалізації алгоритмів: Ці бібліотеки надають готові реалізації популярних алгоритмів посиленого навчання, що зменшує необхідність ручної реалізації алгоритмів.

- Висока розширюваність: Багато з цих бібліотек надають гнучкість та розширюваність для налаштування агентів, середовищ та алгоритмів.

- Інструменти оцінки та аналізу: Деякі бібліотеки надають інструменти для оцінки результатів навчання та аналізування ефективності алгоритмів.

Однак, існують певні обмеження, пов'язані з використанням таких бібліотек:

- Обмеження функціональності: Деякі бібліотеки можуть мати обмежену функціональність порівняно зі спеціалізованою бібліотекою, такою як RLlib.

- Вимоги до навчання: Використання цих бібліотек вимагає глибокого розуміння алгоритмів посиленого навчання та налаштування відповідних параметрів для досягнення найкращих результатів.

- Обмеження розподіленого навчання: Деякі бібліотеки можуть не підтримувати автоматичне масштабування тренування на кластерах, що може бути обмеженням для складних завдань.

Вибір бібліотеки залежить від потреб вашого проекту, рівня експертизи та специфічних вимог до алгоритмів посиленого навчання.[9]

РОЗДІЛ 4. РЕАЛІЗАЦІЯ ОБ'ЄКТА ДОСЛІДЖЕННЯ

4.1 Опис класичної задачі Mountain Car Problem

У двовимірному світі машині потрібно піднятися із западини між двома пагорбами на вершину правого пагорба. Все ускладнює той факт, що їй не вистачає потужності двигуна, щоб подолати силу гравітації та виїхати туди з першої спроби. Нам пропонується навчити агента який зможе, керуючи нею, піднятися на правий пагорб якнайшвидше. Управління машиною здійснюється за допомогою взаємодії із середовищем. Воно поділяється на незалежні епізоди, а кожен епізод здійснюється крок за кроком. На кожному кроці агент отримує з оточення стан s та нагороду r у відповідь на вчинену ним дію a. Крім того, іноді середовище може додатково повідомити, що епізод закінчився.

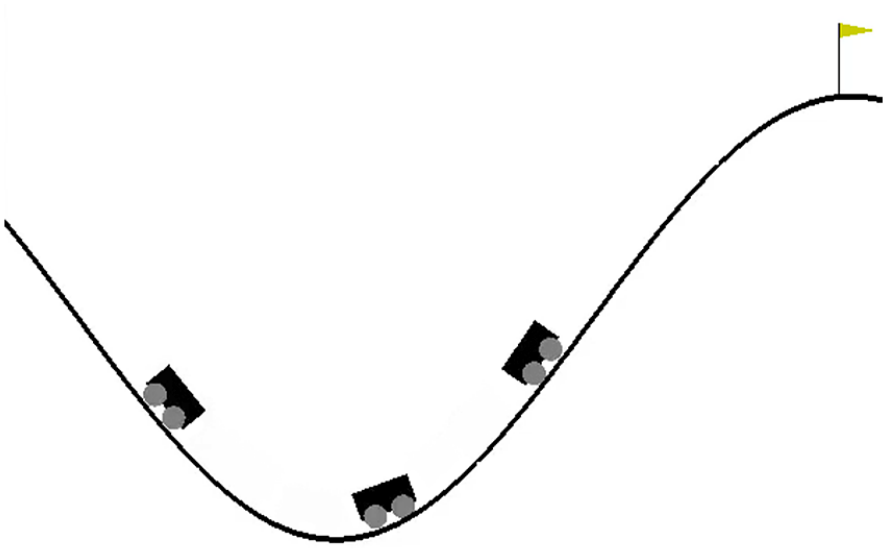

Рисунок 2 Початкова задача

У даній задачі s - пара чисел, перше з яких - положення автомобіля на кривій (однієї координати достатньо, тому що ми не можемо відірватися від поверхні), а друге - його швидкість на поверхні (зі знаком). Нагорода r - число, яке дорівнює:

- “-1” - якщо машинка не доїхала до фінішу
- “100” - якщо машинка доїхала до фінішу.

Було проаналізовано декілька підходів. Один із них - це збереження максимальної координати x, якої досягла машинка (max_reached_position). Якщо під час одного з епізодів машинка долає max_reached_position, то вона отримує додаткові бали, що повинно було мотивувати агента рухатися до фінішу. Але після сотень ітерацій навчання було виявлено, що агент не спішить розганятися до фінішу і закінчувати епізод, а повільно рухається, щоб набрати якомога більше очок.

Можливих дій “a” лише три: штовхати машину ліворуч, нічого не робити і штовхати машину праворуч. Цим діям відповідають числа від 0 до 2. Епізод може завершитися у випадку, якщо автомобіль досяг вершини правого пагорба або якщо агент зробив 200 кроків.

4.2 Ускладнення задачі

Для ускладнення задачі було вирішено внести деякі зміни в умову задачі:

- Замість звичайної синусоїди генерувався випадковий поліном. Задається конкретна кількість точок N, всі точки рівномірно розподіляються по осі X, а далі їм рандомно генеруються координати на осі Y. Далі, по заданим точкам будується поліном 10 степені. Таким чином ми генеруємо випадкову криву для руху машинки

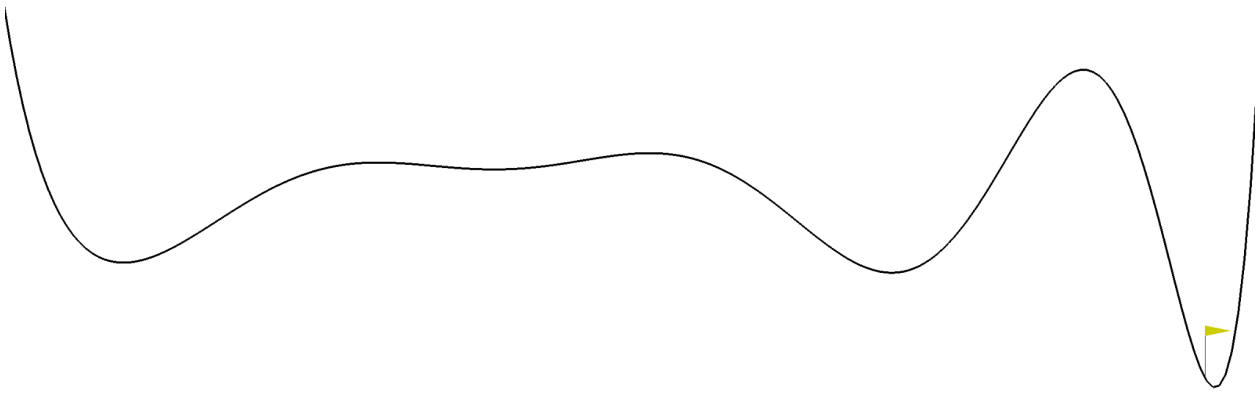

Рисунок 3 Приклад випадкового треку

- Значення змінної s було змінене, був доданий новий інструмент для спостереженнями за середовищем. Промені, які знаходяться в нижній півкулі відносно об’єкта руху і вимірюють відстань від об’єкта до треку. Це дозволяє ігровому агенту відштовхуватися від рельєфу місцевості і оптимальніше навчатися.

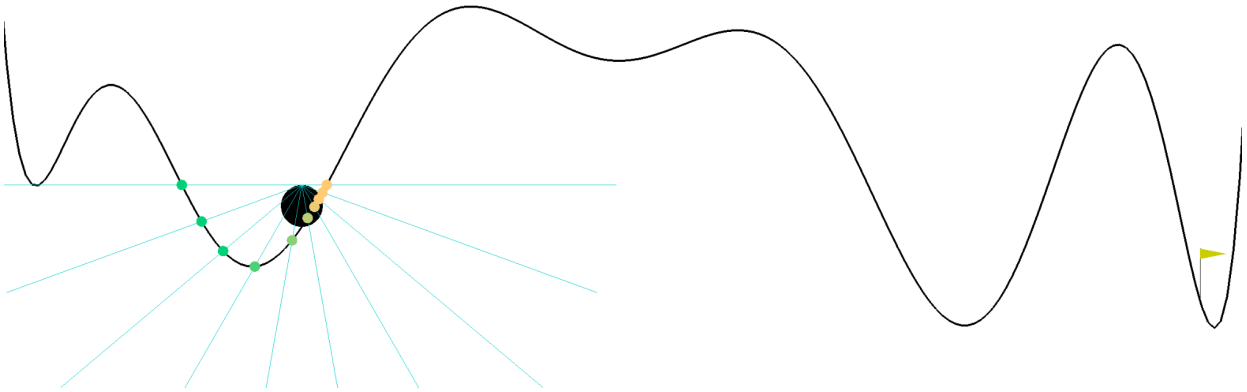

Рисунок 4 Приклад променів для параметру s

Також, було вирішено при генерації поліному першу точку робити максимально високо, щоб об’єкт мав можливість розганятися для подолання висот, які можуть утворитися під час генерації випадкового треку. Початкове положення об’єкта також задається рандомно для того, щоб агент постійно потрапляв в різне середовище.

4.3 Реалізація

4.3.1 Конструктор

Перейдемо до коду реалізації ігрового середовища.

```
def __init__(self, config=None):
```

```
    config = config or {}
    self.steps_count = 0
    self.screen_width = 1200
    self.screen_height = 800
    self.circle_radius = 20

    self.min_position = 0
    self.max_position = self.screen_width
    self.count_vertexes = config.get("count_vertexes", 10)

    self.velocity = 0

    self.goal_position = self.max_position - 50
    self.position = self.min_position + random.randint(50, 150)

    self.poly, self.track_points = generate_track_np(self.min_position,
self.max_position, self.count_vertexes,
                                                     self.screen_height)
    self.derivative = self.poly.deriv()      # for finding tangent

    self.force = 1                           # for action
    self.gravity = 1

    self.rays_count = 10
    self.rays = []
    self.intersections = [None] * self.rays_count
    self.vision_range = [180, 360]
    self.ray_max_distance = 300

    self.viewer = None

    self.action_space = spaces.Discrete(3)
    self.observation_space = spaces.Dict(
        {
            "vision": spaces.Box(0, 1, shape=(self.rays_count,), dtype=float),
            "velocity": spaces.Box(-10000, 10000, shape=(1,), dtype=float),
        }
    )
    self.obs = {}
```

Під час створення середовища варто відмітити деякі налаштування:

- self.force - сила, з якою машинку штовхає дія “a”

- self.gravity - сила, яка діє на машинку, під час руху по кривій і яка не дозволяє одразу виїхати під гірку
- self.ray_max_distance - довжина променів огляду. Дозволяє агенту бачити далі.
- self.poly - функція поліному, для задання треку

4.3.2 Обрахунок швидкості

Розглянемо деякі функції, які дуже важливі для правильного функціонування середовища.

```
def _calculate_new_velocity(self, old_position, old_velocity, action):
    return old_velocity + (action - 1) * self.force +
self.derivative(old_position) * (-self.gravity)
```

Для підрахунку швидкості використовується формула:

$$\upsilon_{new} = \upsilon_{old} + (action - 1) * f + tan(\alpha) * (-g)$$

де

- υ_{old} - швидкість в минулому епізоді
- $action$ - вибрана агентом дія
- f - self.force
- g - self.gravity
- α - кут нахилу кривої в точці, в якій знаходиться машинка.

Тангенс кута нахилу кривої розраховується за допомогою похідної від функції поліному за допомогою функції `self.poly.deriv()` з бібліотеки numpy.

3.3.3 Обрахунок колізій променів огляду з треком

Одна з самих складних функцій. В основу покладений принцип генерації лінійної функції по двум точкам променя (початку та кінця) та розвязання рівняння

$$f(x) - g(x) = 0, x \in [a, b]$$

де $f(x)$ - функція поліному для генерації треку, $g(x)$ - функція променя, $[a, b]$ - кінцеві значення X координат променя. Для цього використовується

вбудована функція `np.roots()`, яка повертає всі можливі корені. Далі, ми обираємо найближчий корінь до початку променя, який не є комплексним числом. І зберігаємо точку перетину для відображення під час рендеру.

```
def _get_ray_collision(self):
    vision = []
    for index, ray in enumerate(self.rays):
        func = np.polyfit([ray[0], ray[2]], [ray[1], ray[3]], 1)
        ray_poly = np.poly1d(func)
        roots = np.roots(self.poly - ray_poly)
        if roots.any():
            closest = roots[0]
            ray_start = [ray[0], ray_poly(ray[0])]
            for root in roots:
                new_point = [root, ray_poly(root)]
                if root.imag == 0.0 and \
                        min(ray[0], ray[2]) <= root <= max(ray[0], ray[2]) and \
                        self._dist_2_points(new_point, ray_start) <
self._dist_2_points(ray_start, [closest, ray_poly(closest)]):
                    closest = root.real

            if min(ray[0], ray[2]) <= closest <= max(ray[0], ray[2]):
                self.intersections[index] = [closest, ray_poly(closest)]
                point1 = [closest, ray_poly(closest)]
                point2 = [ray[0], ray_poly(ray[0])]
                vision.append(self._dist_2_points(point1, point2) /
self.ray_max_distance)
                continue

        self.intersections[index] = [None, None]
        vision.append(1.0)
    return vision
```

4.3.4 Запуск середовища

Код для запуску випадкового агенту (не навченого)

```
import time

from mountain_car_spline import MountainCarEnvSpline

env = MountainCarEnvSpline(config={'count_vertexes': 20})
state = env.reset()
done = False
```

```
episode_reward = 0

while not done:
    observation, reward, done, info = env.step(env.action_space.sample())
    env.render()
    episode_reward += reward
    time.sleep(0.1)
print(f"{episode_reward=}")

env.close()
```

В якості параметру ми можемо явно передати 'count_vertexes' (за замовчуванням дорівнює 10). Це коефіцієнт який використовується для побудови синусоїди. Чим він більше, тим більше пагорбів ми будемо мати під час навчання ігрового агента їзди по цих пагорбах. В такому випадку маємо кінцеву винагороду після завершення заїзду:

episode_reward=-1241

Код для навчання агенту:

```
from ray.rllib.agents.ppo import PPOTrainer
from mountain_car_spline import MountainCarEnvSpline

config = {
    "framework": "torch",
    "env_config": {"count_vertexes": 20}
}
trainer = PPOTrainer(env=MountainCarEnvSpline, config=config)

for i in range(100):
    info = trainer.train()
    path = trainer.save(f'checkpoints_spline_new_reward/')
    print(i, info['episode_reward_mean'], path, info)

env = MountainCarEnvSpline(config={"count_vertexes": 20})
episode_reward = 0
done = False
obs = env.reset()
while not done:
    action = trainer.compute_single_action(obs)
    obs, reward, done, info = env.step(action)
```

```
    env.render()
    episode_reward += reward
print(f"{episode_reward=}")

while not done:
    action = trainer.compute_single_action(obs)
    obs, reward, done, info = env.step(action)
    env.render()
    episode_reward += reward
print(f"{episode_reward=}")
```

Тут ми створюємо PPO тренера, який 100 ітерацій навчає агента і в кінці зберігає налаштування агента в папку checkpoints в директорії проекта, щоб потім мати можливість запустити агента на будь яких трасах.

Натренований агент показує набагато кращий результат:

```
(RolloutWorker pid=34995) 2022-06-13 17:31:19,311   WARNING rollout_worker
(RolloutWorker pid=34995) 2022-06-13 17:31:19,311   WARNING env.py:120 --
2022-06-13 17:31:19,348 WARNING util.py:60 -- Install gputil for GPU syste
2022-06-13 17:31:19,360 INFO trainable.py:534 -- Restored on 127.0.0.1 fro
2022-06-13 17:31:19,360 INFO trainable.py:543 -- Current state after resto
(RolloutWorker pid=34996) 2022-06-13 17:31:19,348   WARNING rollout_worker
(RolloutWorker pid=34996) 2022-06-13 17:31:19,348   WARNING env.py:120 --
2022-06-13 17:31:21.491 Python[34969:1653594] ApplePersistenceIgnoreState:
episode_reward=-33

Process finished with exit code 0
```

Рисунок 5 Результат після тренування

Робимо висновок що агент успішно пройшов навчання та здатен проходити маршрут за найкоротший проміжок часу.

ВИСНОВКИ

Розроблена програма для тренування ігрового агента вийшла досить гнучкою завдяки можливості змінити конфігурацію запуску. Це дозволило провести ряд досліджень для перевірки впливу різних параметрів на ефективність тренування. Програмний застосунок реалізує можливість роботи агента як з відображенням самої гри зі встановленою швидкістю зміни кадрів, що дозволяє наочно побачити поведінку агента, так і без візуальної частини, що пришвидшує виконання. Такий підхід може дати кращий результат, однак тренування триватиме набагато довше і вимагатиме більше ресурсів

Експериментально підтверджено, що обрана модель продемонструвала свою працездатність і достатньо високий рівень близькості до очікуваних результатів.

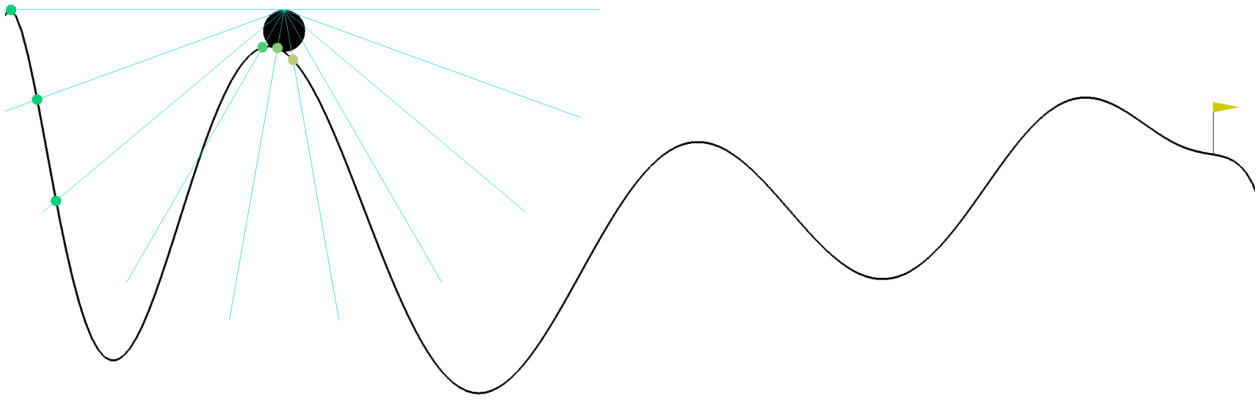

Рисунок 6 Натренована модель проходить випадковий маршрут.

Важко не помітити, що модель відмінно справляється з будь яким ландшафтом. Не без допомоги відмінно налаштованої системи огляду, яка завжди точно передає потрібні координати та точки перетину. Завдяки даній дипломній роботі, було досліджено різні алгоритми штучного навчання,

проаналізовані аналоги бібліотек по машиному навчанню та проведенно вдале моделювання середовища, яке дозволило створити ігрового агента, який справляється з будь яким ландшафтом.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

[1] IDLE [Електронний ресурс] – Режим доступу до ресурсу: https://ru.wikipedia.org/wiki/IDLE.

[2] API [Електронний ресурс] – Режим доступу до ресурсу: https://te-st.ru/2014/08/15/what-is-api/.

[3] Machine Learning [Електронний ресурс] – Режим доступу до ресурсу: https://www.it.ua/ru/knowledge-base/technology-innovation/machine-learning.

[4] Топ 6 Методов Машинного Обучения [Електронний ресурс] – Режим доступу до ресурсу: https://datascience.eu/ru/машинное-обучение/топ-6-методов-машинного-обучения/.

[5] Застосування навчання з підкріпленням для управління ігровим агентом, Мельник Євгеній Іванович: http://naukam.triada.in.ua/index.php/konferentsiji/49-dev-yatnadtsyata-vseukrajinska-praktichno-piznavalna-internet-konferentsiya/436-zastosuvannya-navchannya-z-pidkriplennyam-dlya-upravlinnya-igrovim-agentom

[6] Сравнительный анализ алгоритмов proximal policy optimization и soft-actor-critic, Чачанидзе Елизавета Романовна, студент Московского Государственного Технического Университета им. Н.Э. Баумана, Россия, г. Москва https://cyberleninka.ru/article/n/sravnitelnyy-analiz-algoritmov-proximal-policy-optimization-i-soft-actor-critic/viewer

[7] Глубокое обучение с подкреплением, Лаура Грессер, Ван Лун Кенг. https://drive.google.com/drive/folders/1uO5eF8QsDx7ZaXzb89lvpZdaJ0lur_M2

[8] RLLib documentation. https://docs.ray.io/en/latest/rllib/rllib-training.html

[9] ПОРІВНЯЛЬНИЙ АНАЛІЗ ПРОГРАМНИХ БІБЛІОТЕК ДЛЯ КЛАСИФІКАЦІЇ ТЕКСТОВИХ ДАНИХ ІЗ ВИКОРИСТАННЯМ

ШТУЧНИХ НЕЙРОННИХ МЕРЕЖ. Яременко В.С, Тарасенко М.В., Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського», https://www.tech.vernadskyjournals.in.ua/journals/2019/3_2019/part_1/40.pdf