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

Кваліфікаційна робота

на здобуття освітнього рівня бакалавра
за спеціальністю 121 Інженерія програмного забезпечення
на тему:

НАВЧАННЯ З ПІДКРІПЛЕННЯМ ДЛЯ ГЕНЕРАЦІЇ ОПТИМАЛЬНОЇ СТРАТЕГІЇ АГЕНТІВ НА ПРИКЛАДІ ЗАДАЧІ НАЙШВИДШОГО ПРОХОДЖЕННЯ МАРШРУТУ ТРАНСПОРТНИМ ЗАСОБОМ

Виконала студентка 4-го курсу
Марина ЗОЛОТАРЬОВА _____
(підпис)

Науковий керівник:
доцент, кандидат фіз.-мат. наук
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 41 сторінка, 8 рисунків, 1 таблиця, 15 використаних джерел.

PROXIMAL POLICY OPTIMIZATION, АГЕНТ, МАШИННЕ НАВЧАННЯ, НАВЧАННЯ З ПІДКРІПЛЕННЯМ, СЕРЕДОВИЩЕ, СТРАТЕГІЯ, ТРЕК.

Об’єкт розроблення програмного засобу: навчання агента для оптимального проходження дистанції у наближених до реального життя умовах.

Мета роботи:

а) розглянути способи машинного навчання і визначити доцільний для обраної задачі;

б) розглянути методи обраного способу машинного навчання і визначити доцільний для обраної задачі;

в) обрати алгоритм для реалізації програми;

г) ознайомитись з необхідними бібліотеками і компонентами, встановити їх у середовище розробки;

д) розробити програму, що створює та навчає агентів розв’язувати задану задачу у заданих умовах.

Методи та інструменти розроблення:

а) мова програмування Python;

б) IDE: VSCode v1.78.2;

в) IDLE: Python 3.11;

г) бібліотеки RLlib, OpenAI Gym, Numpy, Pygame.

Результати роботи: Обрано спосіб (навчання з підкріпленням) машинного навчання для здійснення задачі. В якості алгоритму навчання було обрано метод Proximal Policy Optimization (PPO). Встановлено необхідні модулі та здійснена реалізація програми мовою Python. Отримані результати повністю відповідають темі кваліфікаційної роботи.

Інформація щодо впровадження: дане програмне рішення можна використовувати багатьма способами:

а) як повне середовище, для тренування власних агентів для подібних задач;

б) використати лише отримані в результаті тренування ваги з власним треком та графічним інтерфейсом або без нього;

в) використати окремі частини програми для власних цілей, наприклад функція генерації треку є досить універсальною навіть для задач не пов’язаних з машинним навчанням;

Сфера застосування: розробка програмного забезпечення, теорія ігор, робототехніка.

ЗМІСТ

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАЧЕННЯ

IDE - Integrated Development Environment, інтегроване середовище розробки;

API - Application Programming Interface, прикладний програмний інтерфейс;

ML - Machine Learning, машине навчання;

IDLE - Integrated Development and Learning Environment, інтегроване середовище розробки та навчання;

LR - learning rate, крок навчання;

RL - reinforcement learning, навчання з підкріпленням.

ВСТУП

Оцінка сучасного стану об’єкта розроблення. На сьогоднішній день, машинне навчання є однією з найбільш нових та “гарячих” тем у суспільстві. Навіть люди, які не знайомі з програмуванням, використовують ці розробки у повсякденному житті. І через такий високий інтерес до цієї сфери, нові технології виникають постійно, а галузь розвивається так швидко, як ніколи. Завдяки своїй загальності, навчання з підкріпленням застосовується в багатьох дисциплінах. Серед них, такі великі розділи програмування, як теорія ігор та теорія інформації, теорія управління, дослідження операцій, оптимізація на основі моделювання та навіть статистика.

Актуальність роботи та підстави для її виконання. На сьогоднішній день спектр застосування навчання з підкріпленням надзвичайно широкий.

Серед сфер застосування - комп’ютерні та мобільні ігри, особливо ті, де немає чітко визначеного алгоритму дій для перемоги або цей алгоритм занадто складний, як, наприклад в іграх жанру MOBA, які є надзвичайно популярними сьогодні. Також, застосовується в робототехніці, розробці самокерованих автомобілей, рекомендаційних системах, детекторах спаму та в багатьох інших сферах.

Провідні сучасні компанії інвестують мільярди доларів в дослідження та розробку продуктів на основі машинного навчання. Серед них такі гіганти сучасного IT, як Google [1][2], Netflix, Apple, які успішно впроваджують системи на основі машинного навчання в свої продукти, які роками вже на ринку, тільки посилюючи їх.

У сучасному світі, хоч і існує величезна конкуренція між технічними гігантами в швидкості впровадження найновіших технологій, багато інновацій все ще випускається у відкриту, аби будь-хто міг прочитати статтю з описом реалізації та математичної основи продукту та використати його безоплатно.

Так наприклад існує ряд статей, випущених в Google AI Blog за останні 5-10 років, де провідні науковці з Google описують алгоритми на базі

навчання з підкріпленням, які компанія використовує для вирішення великої кількості задач.

Феномен останніх років, компанія, яку зараз найбільш часто обговорюють у соціальних мережах та продукти якої не спробував тільки той, хто не має смартфона, OpenAI, свої розробки не викладає в OpenSource, проте має свій ресурс з корисними статтями щодо ML, аби новачки та досвідчені інженери мали можливість здобувати актуальні знання у галузі.

Саме через велику сферу застосування методу, та наявність активних сучасних розробок, ця робота і є актуальною. Дослідження ведуться кожен день. За останні роки темп росту розробок пов’язаних із ML не тільки не спадає, він збільшується з величезною швидкістю. Лише за останній рік, важкий не тільки для Україні, а і для світу, була випущена неймовірна кількість продуктів на основі машинного навчання, про які говорять і використовують не тільки спеціалісти в сфері IT, а і звичайні люди.

Мета й завдання роботи. Мета дипломної роботи полягає в тренуванні агента в заданому середовищі, аби досягнути оптимальної поведінки. Для цього необхідно:

а) розглянути способи машинного навчання і визначити доцільний для обраної задачі;

б) розглянути методи обраного способу машинного навчання і визначити доцільний для обраної задачі;

в) обрати алгоритм для реалізації програми;

г) ознайомитись з необхідними бібліотеками і компонентами, встановити їх у середовище розробки;

д) розробити програму, що створює та навчає агентів розв’язувати задану задачу у заданих умовах.

Об’єкт, методи й засоби розроблення. Реалізація навчання з підкріпленням ділиться на дві великі групи алгоритмів:

а) навчання з підкріпленням без моделі (Model-free RL);

б) навчання з підкріпленням на основі моделі (Model-based RL).

В свою чергу начання з підкріпленням без моделі може бути реалізоване за допомогою таких методів як: Policy Optimization (включає в себе Policy Gradient, Asynchronous Advantage Actor-Critic, Trust Region Policy Optimization, Proximal Policy Optimization та інші), Q-learning (включає в себе Deep Q Neural Network, C51, Hindsight Experience Replay та інші).

Можливі сфери застосування. Навчання з підкріпленням застосовується у багатьох сферах, серед них: теорія та розробка ігор (створення штучного інтелекту, як суперника гравця), розробка апаратного забезпечення (розробка оптимального дизайну чипів), самокеровані автомобілі (динамічне планування шляху, оптимізація траєкторій, планування рухів), web-простір (система рекомендацій продуктів в інтернет-магазинах, рекомендація відео контенту на стрімінгових сайтах, чат-боти та інше), медицина (оптимізація плану лікування, персональні рекомендації), фінанси тощо.

РОЗДІЛ 1. МАШИННЕ НАВЧАННЯ

1.1 Поняття машинного навчання

Машинне навчання – це галузь штучного інтелекту, що зосереджується на розробці алгоритмів та моделей, які дозволяють комп’ютерам навчатись та робити передбачення або приймати рішення, не будучи явно програмованими для кожного можливого сценарію.

Традиційно комп'ютери програмували виконувати конкретні завдання, керуючись явними інструкціями, написаними програмістами. Для кожного сценарію виконувалась окрема гілка коду програми, розробленої людьми. Однак, у машинному навчанні головною метою є надання комп'ютерам змоги навчатися на основі прикладів або досвіду та відповідно адаптувати свою поведінку, підлаштовуючись до вже відомих сценаріїв. Така модель є найбільш близька до того, як навчаються самі люди.

1.2 Способи машинного навчання

Виділяють декілька способів ML:

а) кероване навчання (навчання з учителем);

Це спосіб навчання, коли програмі надають позначені приклади вхідних та вихідних даних. Алгоритм навчається відповідно до цих прикладів узагальнювати вхідні дані на правильний вихід. Найвідоміша задача, які вирішуються таким способом - задача класифікації. Наприклад, класифікація тварин по фото або літер написаних від руки.

б) спонтанне навчання (навчання без учителя);

Це спосіб навчання, коли програмі надають лише вхідні дані, без відповідних вихідних даних (очікуваного результату). Метою такого способу навчання є знаходження зв’язку між даними і їх впорядкування за цим зв’язком. Кластерний аналіз є типовим прикладом навчання без учителя, де алгоритм автоматично збирає схожі точки даних разом.

в) навчання з підкріпленням.

Інколи його називають частковим випадком керованого навчання – спосіб машинного навчання, у якому «вчителем» виступає середа. Агент не

має попередньої інформації про середовище, але має можливість робити в ній деякі дії. Середа реагує на ці дії і таким чином надає машині дані. Отримуючи відповідь від середи навчання, агент може реагувати на них і вчитися на їх основі. Виходить так званий метод “проб і помилок”. Фактично, машина і середовище утворюють систему зі зворотним зв'язком.

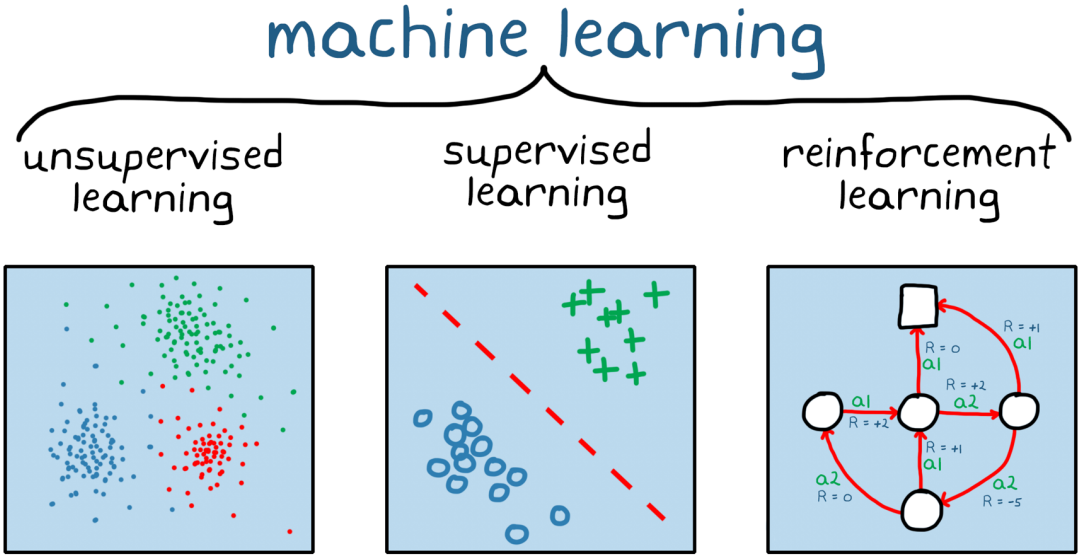

Рисунок 1 - Способи машинного навчання [3]

1.3 Методи машинного навчання

Існує величезний список різноманітних методів машинного навчання, кожен з яких може краще підходити до тієї чи іншої задачі. Можна виділити декілька основних:

а) регресія;

Використовується для прогнозування неперервних числових значені. Дозволяє знайти кореляцію між вхідними змінними, так званими незалежними ознаками, та вихідними, залежними, змінними для визначення причинно-наслідкового зв'язку. На основі визначеної кореляції може бути зроблений прогноз, побудований графік тощо. Існують різні форми та алгоритми регресії, починаючи від лінійної, найпростішої, і закінчуючи комплексною, обчисленням поліноміальних даних і представленням. Поширеними прикладами лінійної регресії є прогноз погоди, тенденції ринку, виявлення потенційних ризиків.

б) класифікація;

Метою є призначення вхідним даним попередньо визначеної категорії або класу. Цей метод, завдяки обчисленню ймовірностей, розділяє об’єкти по групах відповідно до наперед визначених ознак.. Алгоритми навчаються на основі позначених прикладів для класифікації нових, невиданих даних. Широко використовуваними алгоритмами класифікації є, наприклад, дерева рішень.

в) кластеризація;

Метою є виявлення базових структур або кластерів в даних, при цьому не маючи наперед визначених класів та маркованих даних. Цей метод, як і класифікація, розділяє об’єкти по групах (кластерах), але, на відміну від класифікації, не потребує визначення ознак. Метод кластеризації полягає в групуванні схожих точок даних на основі їх внутрішніх закономірностей або подібностей.

Найбільш відомим алгоритмом кластерізації є метод к-середніх (K-means).

г) скорочення розмірів;

Методи зменшення розмірності спрямовані на зменшення кількості вхідних змінних, зберігаючи при цьому важливу інформацію. Це дозволяє відмовитися від несуттєвих змінних. Ці методи корисні при роботі з великими даними або для зменшення обчислювальної складності. Наприклад, якщо ви хочете передбачити ризик виникнення раку у групи людей на основі багатьох даних, фактором споживання кави можна знехтувати, хоч вона і є канцерогенним продуктом.

Основний компонентний аналіз (PCA) є широко використовуваним методом зменшення розмірності.

д) метод ансамбля;

Ансамблеві методи поєднують декілька моделей для прийняття прогнозів. Вони спрямовані на покращення загальної продуктивності, використовуючи мудрість кількох моделей. Таким чином, він об'єднує різні моделі прогнозування для формування високоточних і оптимізованих

результатів. Метод ансамблю використовується для прийняття рішень при розгляданні різних факторів.

В якості простого прикладу, можна навести визначення виду тварини по фото. Якщо ми маємо одну модель, яка гарно вміє розділяти котів та собак, іншу, яка вчилась на левах та тиграх, то за допомогою методу ансамблю, ми можемо отримати модель-комбінацію двох існуючих, яка вже буде вміти розділяти не два типи, а чотири.

е) нейронні мережі.

Цей метод був натхненний структурою людського мозку, і є потужним класом алгоритмів, які використовуються для різних завдань машинного навчання. На відміну від лінійних моделей, нейронна мережа заснована на складній, дивізійній структурі даних. Ці системи складаються з взаємопов'язаних вузлів (нейронів), організованих у шари. Проте, модель все ще базується на лінійній регресії, але використовує кілька прихованих шарів. Прикладом нейронних мереж можуть бути згорткові нейронні мережі (CNN), які часто використовуються для завдань, пов'язаних з класифікацію та обробкою зображень. Також рекурентні нейронні мережі (RNNs), які добре підходять для послідовних даних.

РОЗДІЛ 2. НАВЧАННЯ З ПІДКРІПЛЕННЯМ

2.1 Поняття навчання з підкріпленням

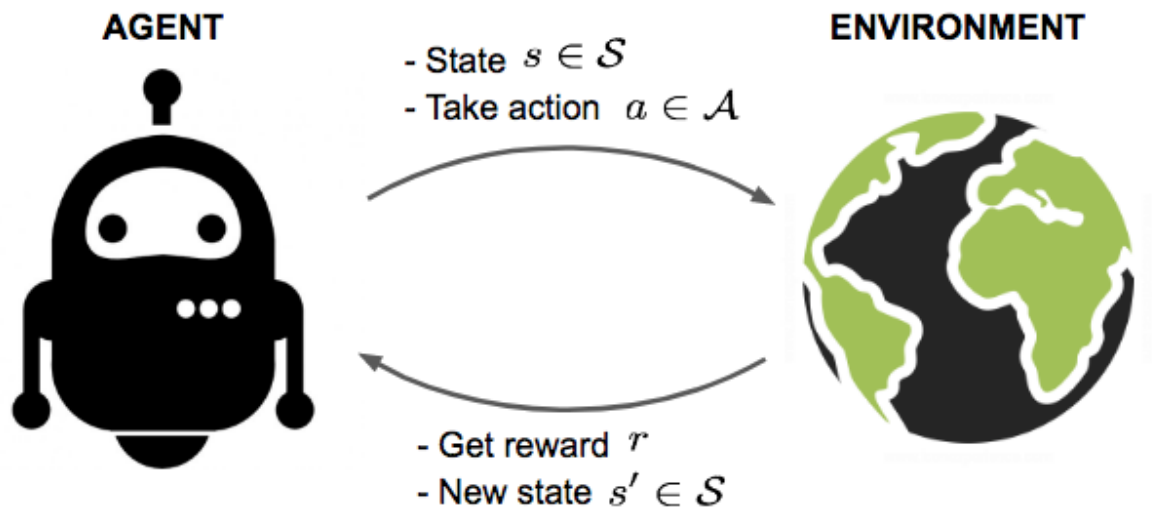

Рисунок 2 - Модель навчання з підкріпленням [7]

Навчання з підкріпленням (англ. reinforcement learning) — це галузь машинного навчання, що вивчає питання про те, які дії повинні виконувати програмні агенти в певному середовищі задля максимізації деякого уявлення про сукупну винагороду [4]. Цей метод був натхненний тим, як люди та тварини навчаються взаємодії з оточуючим світом, крок за кроком вивчаючи середовище. Під час навчання з підкріпленням, агенти типово вчаться за методом проб та помилок, ведучи комунікацію з середовищем, в якому знаходяться. Середовище зазвичай представляє з себе деяку математичну модель або симуляцію, описану розробником. Агенти, як частина середовища, мають вибір з деякого детермінованого списку дій на кожному кроці, на основі теперішнього стану системи. Кожен крок агент приймає рішення, яку дію він буде робити і передає її середовищу в очікуванні на відповідь. У відповіді середовища зазвичай буде два пункти: нагорода і новий стан. Нагорода, отримана агентом за попередню дію, може бути як від'ємна, так і додатна, також може дорівнювати нулю. По суті, розмір нагороди визначається тим, наскільки гарну дію зробив агент, враховуючи стан середовища на той момент. На початку навчання рішення агента - це зазвичай

щось абсолютне випадкове, адже машина ще не має жодного представлення про середовище, в якому перебуває, а отже кожній дії він надає однаковий пріоритет. З кожною новою епохою навчання, агент краще розбирається в середовищі і з більш високою вірогідністю робить дію, за яку отримає більшу винагороду.

2.2 Методи навчання з підкріпленням

Виділяють два основні підходи до реалізації навчання з підкріпленням: навчання з підкріпленням без моделі (Model-free RL) та з моделлю (Model-based RL)[4][5][6]. Ці два різних підходи, які відрізняються в першу чергу тим, як агенти вчаться та приймають рішення в умовах середовища.

Навчання з підкріпленням на основі моделі намагається вивчити середовище наперед, визначити його схему, яка відображає динаміку та систему переходів меж станами. По своїй суті, це модель є ймовірнісним представленням середовища та його відповідей на певні дії агента.

Model-based алгоритм навчання з підкріпленням складається з двох етапів:

а) визначення моделі;

Як було описано, агент просто вивчає те, як реагує середовище на деякі його дії, намагаючись отримати найбільшу можливу винагороду.

б) планування.

Після того, як агент отримує визначену модель, він може приймати свої рішення на її основі при цьому не роблячи звернення до моделі на кожному кроці з новим станом, а промоделювати та спланувати їх наперед. Для того, щоб визначити оптимальну послідовність дій на декілька кроків вперед і обрати стратегію, можуть використовуватися різні методи, такі як динамічне програмування або метод Монте-Карло. Планування включає в себе також оцінку різних вірогідних сценаріїв задля вибору найкращої стратегії дій агента.

Переваги навчання з підкріпленням на основі моделі є очевидними, адже агент має можливість планувати наперед та приймати обґрунтовані

рішення на основі тренованої на даному конкретному середовищі моделі. Також агенти, навчені таки способом, вміють підлаштовуватись під не бачені досі сценарії та невизначеності, а також узагальнювати та генералізувати вже отримані знання.

Проте, аби таке навчання було можливим, необхідно мати середовище, для якого можливо визначити модель. Далеко не для всіх систем дійсно існує така модель. Навіть, якщо її можливо визначити, вона може складатися з мільярдів розгалужень і бути достатньо тяжкою. Якщо такої моделі не існує взагалі, або вона настільки велика, що навчання агента і особливо його майбутні дії будуть займати вагомий проміжок часу, то є сенс обрати інший алгоритм для навчання, адже така система або не буде працювати взагалі, або буде неефективною.

Методи навчання з підкріпленням без моделі навчаються оптимальній стратегії та намагаються максимізувати винагороду не будуючи явно модель середовища. Це є вагомою перевагою у випадках розглянутих вище, коли використання моделі не є можливим з певних причин. Такі агенти вивчають середовище тільки методом проб та помилок, при цьому не враховуючи динаміку переходів між станами.

Model-free RL методи мають навчатися безпосередньо на основі лише власної взаємодії з середовищем. Проте, часто вони потребують значно більше ітерацій, ніж Model-based RL, адже їм необхідно більше взаємодій з середовищем задля знаходження оптимальної стратегії. Оскільки, для навчання з підкріпленням без моделі основним способом навчання є метод проб та помилок, і, в залежності від складності середовища, аби правильно виділити зв’язок між дією і винагородою, агенту без моделі може знадобиться більше епох. Через це, їх навчання вимагає більше часу. Також, для навчання такого агента може знадобитись більше тестових даних, які в деяких випадках отримуються непросто. Потрібно пам’ятати про це, обираючи метод навчання агенту у кожній задачі.

Обидва описаних методи широко використовуються в сучасній розробці і мають свої недоліки і переваги. Вибір методу залежить лише від поставленої задачі, наявності тестових даних та безпосередньої складності середовища.

2.3 Алгоритми навчання з підкріпленням без моделі

Навчання без моделі - широкий розділ навчання з підкріпленням, який досліджується і використовується не так давно, але при цьому вже існує немало алгоритмів, розроблених для різних задач. Їх можна розділити на дві основні групи: Q-learning та метод оптимізації стратегії (Policy Optimization).

а) Q-learning [8];

Основне рівняння, яке необхідне для реалізації алгоритму Q-learning - рівняння оптимальності Беллмана:

$$q_*(s,a) = E[R_{t+1} + \gamma max q_*(s'a')]$$

Це рівняння каже, що за умови оптимальної дії агента, для будь-якої заданої пари стану та дії (*s, a*) функція цінності дії (Action-Value function) або Q-value function дорівнює сумі винагороди, за умови виконання дії *a* в стані *s,* і максимального очікуємого значення з наступної пари (s, a), помноженого на коефіцієнт гамма який лежить в проміжку (0, 1]. Коефіцієнт гамма необхідний в даному рівнянні, аби найбільшу увагу на кожному кроці приділити винагороді саме на теперішньому кроці, а не на наступних. Тому досить часто цей коефіцієнт дорівнює 0.99, або іншому значенню близькому до одиниці.

В алгоритмі Q-learning ітеративно шукається оптимальна Q-функція, використовуючи рівняння оптимальності Беллмана. Для цього зберігається таблиця з усіма Q-значеннями, яка оновлюється на кожній ітерації.

$$q^{new}(s,a) = (1-\alpha)q(s,a) + \alpha(R_{t+1} + \gamma max q_*(s'a'))$$

В цьому рівнянні: α - швидкість навчання, параметр, який задається розробником для кожної моделі окремо і який часто необхідно підлаштовувати.

Також, для імплементації алгоритму, необхідно врахувати так званий

баланс дослідження та експлуатації (exploration-exploitation trade-off). Оскільки на перших епохах агент не повинен приймати якісь рішення на основі попереднього досвіду, адже він ще немає ніяких знань про середовище. Тільки з прогресом навчання, агент має “експлуатувати” свої знання, а на початку лише досліджувати. Якщо пропустити цей етап, то Q-функція з великою вірогідністю буде наближатись до локального мінімуму, а це є не оптимальним значенням для більшості середовищ.

Таким чином, алгоритм зводиться до послідовності ітерацій, перед якими ми спочатку задаємо пусту таблицю, а потім на кожному кроці її оновлюємо, намагаючись оптимізувати Q-функцію, аби отримати найбільшу винагороду.

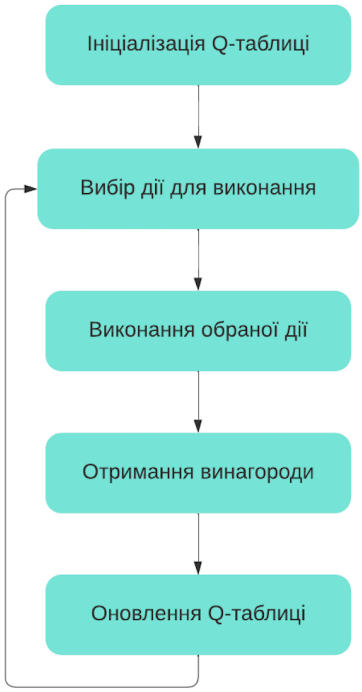

Рисунок 3 - Схема роботи алгоритму Q-learning

б) оптимізація стратегії (Policy Optimization).

В контексті навчання з підкріпленням, стратегія (π) - це функція, на вхід якої надається стан s, а повертається можлива дія a. В методах на основі стратегії функція визначається множиною налаштовуємих параметрів θ. Розробник може змінювати ці параметри, відслідковуючи отриманий результат, та підлаштовувати їх таким чином, щоб отримувати найкращий

можливий результат. Такий механізм лежить у основі всіх методів оптимізації стратегії, якщо не у всьому машинному навчанні.

Функція стратегії є стохастичною, тобто її параметри визначають ймовірності множини подій *a* і таким чином впливають на вірогідності деяких траєкторій.

Використовуючи стохастичну стратегію, можна відібрати різні дії і виміряти їх різницю у винагороді з відповідними ймовірностями. Обчислена винагорода в комбінації з вірогідністю дає нам сигнал винагороди. Для того, щоб обчислити напрямок зміни для покращення функції, деякі з методів оптимізації стратегії покладаються на градієнти (як, наприклад, алгоритм Policy Gradient - алгоритм градієнту стратегії).

Таким чином, ми знаємо, в якому напрямку необхідно рухати нашу функцію, аби максимізувати винагороду. Розмір кроків визначається коефіцієнтом швидкості навчання (learning rate), який, як вже було сказано, є налаштовуємим параметром. Дуже важливо підібрати LR правильно, адже він напряму впливає на ефективність навчання.

Узагальнюючи алгоритм оптимізації стратегії, можна виділити такі основні кроки:

а) агент вивчає стан середовища (*s*);

б) агент робить дію (*a*) покладаючись на свої “інстинкти”, а саме на функцію стратегії (π);

в) середовище реагує на дію агента, обробляє її, повертаючи йому свій новий стан;

г) на основі нового стану, агент знову приймає рішення і передає його середовищу;

д) покладаючись на свої спостереження, агент оновлює свою функцію стратегії, роблячи її більш точною.

РОЗДІЛ 3. PROXIMAL POLICY OPTIMIZATION

3.1 Загальний огляд групи алгоритмів

Методи оптимізації стратегії (policy optimization) за останні роки набувають домінуючої позиції у світі глибоких нейронних мереж. На сьогоднішній день ця групи методів використовується у найрізноманітніших сферах, починаючи від відеоігор і закінчуючи тривимірним пересуванням.

Тим не менш, вже згаданий метод градієнту стратегії (policy gradient або PG), який раніше був найбільш часто використовуваним серед цієї групи алгоритмів, має ряд недоліків.

В першу чергу, за допомогою цього методу дуже складно отримати гарні, робочі результати, адже алгоритм надто чутливий до вибору швидкості навчання (learning rate). Обравши занадто малий крок, дослідник ризикує зіткнутися з безнадійно повільним прогресом особливо за умови складної системи стратегії. А обравши занадто великий крок, велика вірогідність отримати перевантажений шумами сигнал або взагалі втратити продуктивність моделі.

Також варто зазначити, що, за використання методу градієнту стратегії, дуже часто необхідно витратити мільйони, або навіть мільярди епох, аби навчитись робити навіть прості задачі. І важко навіть уявити, скільки часу, людських і машинних ресурсів, доведеться витратити для розробки чогось комплексного, зі складною стратегією і системою винагород.

Саме через велику кількість недоліків PG, було розроблено ряд алгоритмів, які його оптимізують. Таким є, наприклад, є алгоритм Trust Region Policy Optimization (TRPO). В алгоритмі TRPO стратегія оновлюється роблячи найбільші можливі кроки до покращення продуктивності, при цьому накладаються деякі обмеження, щодо того, як близько одна до одної можуть бути нова та стара стратегії.

Хоч TRPO і має кращі результати ніж PG на ряді задач, цей алгоритм все ще має свої недоліки і потребує оптимізації.

3.2 Алгоритм Proximal Policy Optimization

Алгоритм Proximal Policy Optimization [9], який і було обрано для розв’язання поставленої задачі, на сьогоднішній день є найбільш популярним і вдало реалізованим серед групи алгоритмів оптимізації стратегії. Навіть у 2023 році, провідні компанії все ще вважають цей метод state-of-the-art технологією у сфері навчання з підкріпленням.

Реалізований він був ще у 2017 році командою з дуже відомої на сьогоднішній день компанії OpenAI.

Його створення було вмотивовано проблемою занадто сильної прив’язки алгоритму Policy Gradient до кроку навчання. Так само, як і вже згаданий TRPO, метод PPO намагається робити найбільший можливий крок в напрямку оптимізації функції стратегії, при цьому не зробивши цей крок завеликим.

Існує дві найбільших модифікації методу PPO:

а) PPO-Penalty, в якому так само, як і в TRPO існують обмеження щодо розбіжностей нової і старої стратегії. Проте, навідміну від TRPO, ці обмеження не є жорсткими. В цьому алгоритмі лише накладається штрафний коефіцієнт на цільову функцію, який автоматично коригується протягом всього навчання;

б) PPO-Clip. В цьому алгоритмі взагалі немає ніякого штрафу або обмежень щодо різниці між стратегіями, проте цільова функція спеціальним чином модифікується, аби усунути спроби нової функції стратегії занадто далеко відхилитися від старої.

Кожна з наявних модифікацій використовується в залежності від поставленої задачі.

Отже, порівнюючи методи навчання з підкріпленням із іншими методами машинного навчання, можна сказати, що використавши наприклад навчання з вчителем (supervised learning), досить неважко реалізувати невеликий класифікатор, який, використовуючи такі сучасні фреймоворки як Tensorflow або Pytorch, може написати навіть людина дуже поверхнево

знайома з теорією та методами машинного навчання. І навіть за таких відносно невеликих зусиль, з мінімальним налаштуванням гіперпараметрів можна отримати відносно непогані результати. Тим не менш, такі методи не є універсальними і їх неможливо застосувати до кластеру задач, які вирішуються за допомогою навчання з підкріпленням. При цьому, використовуючи reinforcement learning, в середньому потрібно витратити більше часу налаштовуючи середовище та підбираючи параметри, адже алгоритми мають багато рухомих частин і досить рідко вже існує реалізоване рішення для конкретної задачі у відкритому доступі, адже кожна проблема потребує свого унікального розв’язання.

За таких умов, алгоритм PPO є одним з найбільш використовуваних алгоритмів для навчання з підкріпленням на сьогоднішній день. І хоч, крім вибору алгоритму, існує ще багато частинок, які необхідно розробити для кожної конкретної задачі, і немає необхідності в тому, аби реалізовувати алгоритм з нуля, гарному розробнику необхідно розуміти, як працює базова складова програми, аби працювати найбільш ефективно.

РОЗДІЛ 4. ОПИС МОВИ ПРОГРАМУВАННЯ ТА БІБЛІОТЕК ВИКОРИСТАНИХ ДЛЯ ВИРІШЕННЯ ЗАДАЧІ

4.1 Мова Python

Python - це високорівнева, інтерпретована мова програмування, найбільш відома своєю простотою та універсальністю.

Створив Python голландський програміст на ім’я Гвідо ван Россум, який і досі бере активну участь в розробці нових версій мови і приймає фінальні рішення щодо дизайну.

Випущена мова на широкий загал була у 1991 році і досить швидко набула популярності. Багато рис мови запозичено з таких відомих усім мов, як C, C++ та Java.

Основа філософії мови і його творців - просте краще, ніж складне.

Мова має дуже низький поріг входу, в порівнянні з тими ж C++ та Java, а отже є легкою в засвоєнні для новачків в програмуванні.

Серед інших переваг мови можна виділити наступні риси:

а) читабельність;

Синтаксис Python розроблений для легкості читання та розуміння. При написанні коду використовують відступи та пробіли для відокремлення блоків коду замість традиційних дужок. Це сприяє читабельності коду та знижує ймовірність синтаксичних помилок.

б) універсальність;

Python - універсальна мова програмування, яку можна використовувати для широкого спектру задач. Тут немає сильної прив’язки до конкретної парадигми програмування, що дає розробникам свободу в написанні коду. Підтримуються такі парадигми, як процедурне, об'єктно-орієнтоване та функціональне програмування. Python широко використовується в веб-розробці, аналізі даних, штучному інтелекті, машинному навчанні, науковому обчисленні, автоматизації.

в) велика стандартна бібліотека;

Python має величезну стандартну бібліотеку, яка надає готові модулі та

функції для вирішення різних завдань. Стандартна бібліотека охоплює такі області, як робота з файлами, мережеву взаємодію, регулярні вирази, математику, структури даних та багато іншого.

г) сторонні пакети;

Python має велику екосистему сторонніх пакетів та бібліотек, які розширюють його можливості. Оскільки мова програмування набула неабиякої популярності в спільноті програмістів, активна розробка сторонніх пакетів для мови продовжується досі.

Популярні пакети, такі як NumPy, Pandas, TensorFlow, Django, Tkinter та інші, розширюють функціональність Python, роблячи його ще більш універсальним та підходящим до великого спектру задач.

д) динамічна типізація;

Python є динамічно типізованою мовою, що означає, що типи змінних визначаються під час виконання програми, що дозволяє розробникам швидше писати код. Про те, саме через динамічну типізацію може бути ускладненим пошук помилок у коді.

е) автоматичне управління пам’яттю;

Python має автоматичне управління пам'яттю, збір сміття здійснюється автоматично, що також прибирає з відповідальності розробників необхідність в ручному виділенні та звільненні пам’яті.

є) крос-платформена сумісність.

Python - це крос-платформена мова, що означає, що вона працює на різних операційних системах, включаючи Windows, macOS, Linux та інші. Розробники можуть писати код на одній платформі та виконувати його на іншій без значних модифікацій.

Хоч мова і має велику кількість переваг перед своїми конкурентами, все ж таки поки що не існує мови, яка ідеально підходила б під усі задачі, а отже Python також має свої недоліки.

Головним недоліком і обмеженням мови, звісно, буде продуктивність. Оскільки Python високорівнева мова і має такі, зручні для розробника

характеристики, як колектор сміття і динамічна типізація, це все ж таки впливає на продуктивність і робить Python значно більш повільним у порівнянні все з тими ж Java і C++.

Через такий недолік, мова Python абсолютно не підходить для реалізації систем, які є залежними від продуктивності.

Також, виходячи з власного досвіду і відгуків інших програмістів, Python не є першим при виборі мови для написання серверної частини деякої великої web-орієнтованої системи. На це є ряд причин, по-перше, через відсутність статичної типізації часто можуть виникати помилки в клієнт-серверній взаємодії. Такі помилки досить важко знаходити. По-друге, через проблеми продуктивності і складність в паралельному виконанні декількох запитів в Python (все через так званий GIL - Global Interpreter Lock - який не дозволяє виконувати код, написаний на Python, в декількох потоках одночасно), сервер може занадто довго обробляти запити в системах, де є багато користувачів, а отже і багато запитів на сервер в секунду.

Також, серед недоліків мови, присутній факт, що Python, в середньому, споживає більше пам’яті, ніж його опоненти зі статичною типізацією. Звісно, це можна оптимізувати, але це також зайва робота для програміста, яку можна не робити, обравши іншу мову для написання системи.

4.2 Мова Python для машинного навчання

Беручи до уваги усі недоліки та переваги мови, на сьогоднішній день Python залишається найкращою мовою для машинного навчання.

Ця мова є першою і майже єдиною при виборі стеку технологій для задач машинного навчання не тільки для новачків, а і для професіоналів.

На це є багато причин, але перш за все варто виділити багату екосистему мови. Python має чи не найбільшу колекцію бібліотек присвячених машинному навчанню, які надають широкий спектр інструментів, вже реалізованих алгоритмів. Таке різноманіття інструментів надає розробнику можливість займатися лише високорівневою частиною розробки, не витрачаючи при цьому час на написання алгоритмів та

загальних функцій.

Також, завдяки тому, що мова активно розвивається вже не один десяток років, виникло величезне ком’юніті розробників, які готові ділитися своїм досвідом з колегами. Це дуже важливо, як для новачків, так і для професіоналів. Велике ком’юніті - запорука того, що мова буде розвиватись та покращуватись. Також це гарантує високу вірогідність того, що розробник, стикнувшись з проблемою, знайде когось, хто вже вирішив це і готовий поділитися досвідом на одному з ресурсів, створених для програмістів.

Як вже було сказано, Python - не є найкращою мовою з точки зору продуктивності, його творці розуміли це і постаралися виправити цю проблему значно спростивши інтеграцію з іншими мовами та бібліотеками, написаними не на Python. Тому мова не тільки гарно себе показує самостійно, а і також гарно інтегрується з іншими мовами, що значно покращує її продуктивність.

Серед бібліотек, використовуваних для машинного навчання варто зазначити такі, як Pytorch та Tensorflow. Обидві вони надають великий спектр інструментів саме для машинного навчання, серед яких: наявність попередньо навчених моделей, які можна інтегрувати в свої власні системи, можливість ефективного розподіленого навчання на базі графічних процесорів, що сильно прискорює етап тренування.

Отже, на сьогоднішній день Python є однією з найбільш популярних та універсальних мов. Не можна сказати, що вже залежності від задачі - Python буде кращим вибором для написання систему, адже, як і у будь-якої мови, є ряд недоліків, які в деякому сенсі обмежують доцільність використання.

Проте, можна впевнено стверджувати, що Python, завдяки своїй простоті, дійсно є мовою номер один для машинного навчання. Це підтверджується наявністю великої спільноти розробників, а також багатою різноманітністю вже реалізованих інструментів та бібліотек, які сильно полегшують та покращують швидкість розробки.

4.3 RLlib

RLlib (Reinforcement-Learning Library) - це бібліотека з відкритим вихідним кодом, яка надає великий набір різноманітних інструментів для машинного навчання, а саме - для навчання з підкріпленням.

Бібліотека надає високий рівень абстракції, дозволяючи використовувати різні методи, в залежності від поставленої задачі.

Так в RLlib підтримується мультиагентні і моноагентні середовища, різні види даних: так звані офлайн дані, або історичні дані, а також навіть зовнішні симуляції, аби збирати датасети на льоту.

Ця бібліотека використовується також на виробничому рівні, а отже підтримує роботу з високим розподіленим навантаженням, аби видавати найкращу можливу продуктивність для реальних систем.

Документація бібліотеки стверджує, що тут кожен може знайти інструмент для будь-яких прикладних задач, пов’язаних з навчанням з підкріпленням.

Про те, що бібліотека дійсно широко використовується може свідчити статистика її гітхабу. Станом на травень 2023 року, активна розробка бібліотеки продовжується: останні внесення в кодову базу були цього місяця. За весь час майже 700 розробників зробили свій вклад в розвиток бібліотеки, а остання зміна зроблена менше, ніж два дні тому.

Така активність розробників RLlib та її користувачів, свідчить про те, що технологія дійсно корисна для багатьох людей і активно використовується сьогодні.

Серед інших переваг RLlib - простота, надаючи високий рівень абстракції, розробники бібліотеки намагалися максимально зменшити час між першим знайомством з бібліотекою та першим написаним робочим кодом. Максимально просте налаштування, велика кількість робочих прикладів, які постійно оновлюються, - все це дозволяє розробникам насолодитись візуалізацією перших робочих результатів дуже швидко, що тільки підсилює інтерес та бажання розробників створити щось дійсно

потужне.

Перерахуємо усі переваги RLLib:

а) підтримка найбільш розповсюджених фреймворків для глибокого навчання: Pytorch та Tensorflow. Варто зазначити, що їх завантаження не відбувається автоматично із завантаженням RLlib, оскільки частіше за все для кожного проекту знадобиться лише один із фреймворків, а отже розробнику необхідно самому обрати, який саме, та встановити його;

б) високо розподілене навчання. В бібліотеці реалізована архітектура таким чином, що агенти можуть навчатися одночасно на сотнях процесорів, що надає можливість значно пришвидшувати етап навчання, за наявності відповідних машинних ресурсів. При цьому, кількість робітників визначається лише одним параметром налаштування, що є максимально зручно;

в) можливість створювати векторизовані та віддалені середовища;

г) мультиагентні системи. В одному середовищі можна налаштовувати одразу декілька агентів, при цьому задаючи їм однакові або різні стратегії, в залежності від поставленої задачі.

4.4 Порівняння RLlib із аналогами

Оскільки машинне навчання вцілому і навчання з підкріпленням, як частковий випадок, є одними з найбільш швидко зростаючих технологій в сучасному світі, велика кількість бібліотек і фреймворків розробляється різними конкуруючими структурами. Код таких розробок досить частко викладається у відкриту, аби усі бажаючи могли покращити його, але це надає конкурентам можливість перейняти щось, що не завжди є бажаним, особливо для великих компаній. Тому, викладати чи код у відкритий репозиторій, вирішує для себе кожна окрема команда та компанія, адже обидва підходи мають свої недоліки та переваги.

Існує великий список бібліотек, розроблених для реалізації навчання з підкріпленням, обираючи найкращу варто врахувати багато факторів виходячи із задачі та наявних ресурсів.

Наведений невеликий огляд є в деяких частинах суб’єктивний, адже дві останні колонки щодо оцінки якості документації та наявності туторіалів - особисто моя думка.

Проте, обираючи ту чи іншу бібліотеку серед аналогів, крім перевірки її відповідності поставленій задачі, дуже важливо сконцентруватися на тому, чи використовують її інші люди. Якщо у гітхабі велика активність, запити користувачів отримують відповідь від розробників - значить з великою вірогідністю навіть знайшовши проблему з бібліотекою, можна розраховувати на швидку відповідь від підтримки з подальшим виправленням.

Таблиця 4.1. Порівняльна характеристика бібліотек для навчання із підкріпленням [10][11][12][13][14][15]

	Відкритий код	Зірки на GitHub	Форки на GitHub	Дата останнього коміту в головну гілку	Якість документації	Наявність туторіалів
RLlib	Так	25800	4500	Травень 2023	10/10	Так, вичерпна кількість
Dopamine	Так	10100	1400	20 березня 2023	8/10	Так
Mushroom-RL	Так	683	126	Травень 2023	6/10	Ні
RL_Coach	Так	2200	449	11 грудня 2022, репозиторій архівовано	9/10	Так, вичерпна кількість
Tensorforce	Так	3200	540	15 січня 2023	9/10	Так

Не існує одразу ідеальних продуктів, кожен інструмент потребує оновлень на основі відгуків користувачів, тому важливо брати до уваги, чи дійсно ведеться активна розробка над бібліотекою досі, аби не було ситуації, коли посеред розробки, необхідно змінювати увесь стек використовуваних технологій, через знайдену помилку в бібліотеці.

Роблячи висновки з проведеного порівняння, можна з впевненістю сказати, що робота на RLlib ведеться активна починаючи з 2016 року і закінчуючи сьогоднішнім днем, люди активно нею користуються, а розробники відповідають на запити користувачів досить швидко.

Крім того, документація бібліотеки вичерпна, а різноманіття реалізованих алгоритмів більш ніж вичерпне.

РОЗДІЛ 5. РЕАЛІЗАЦІЯ ОБ'ЄКТА ДОСЛІДЖЕННЯ

5.1 Реалізація моноагентного середовища з єдиним керуванням

Використовуючи сучасні бібліотеки для навчання з підкріпленням, такі, як RLlib, основна частина реалізації, про яку варто думати програмісту - розробка середовища та системи винагород агента. Додатково ще має сенс реалізувати графічний інтерфейс, звісно, це не є необхідним для самого тренування, проте важливо, аби людським оком можна було простіше оцінити результати роботи.

Поділимо задачу розробки середовища на менші частини:

а) генерація треку, по якому будуть їздити машинки;

Задача стоїть в тому, аби агенти вміли проходити будь-який, обраний випадковим чином, трек. Ми не можемо вчитись на одному треку, адже тоді агенти вивчать лише його і навчаться діяти лише в заданих умовах, не генералізуючись при цьому на інші задачі. Саме тому, нам необхідна функція, яка автоматично може генерувати треки різної довжини та характеристик.

Алгоритм генерації досить простий - генеруємо ламану випадково обираючи довжину кожної компоненти в заданих обмеженнях, потім робимо паралельний перенос ламаної, отримуючи дорогу з двома паралельними сторонами.

б) реалізація “зору”;

В кожний момент часу, аби приймати рішення, машинка має отримувати деякий стан середовища. Намагаючись приблизити модельовану ситуацію, до ситуації реального водіння, була обрана схема зорових променів. Машинка бачить попереду себе на деяку відстань, в першій реалізації - на 450 пікселей, кут зору - 90 градусів, від -45 до 45 градусів, виходячи від переду агента. Рівномірно розподілені промені зору (25 штук) дають інформацію про те, наскільки близько до кожного променя є перепона. Ця інформація передається нормалізованою на максимальну довжину променя, аби інформація, передана агенту щодо кожного променя лежала в проміжку від 0

до 1.

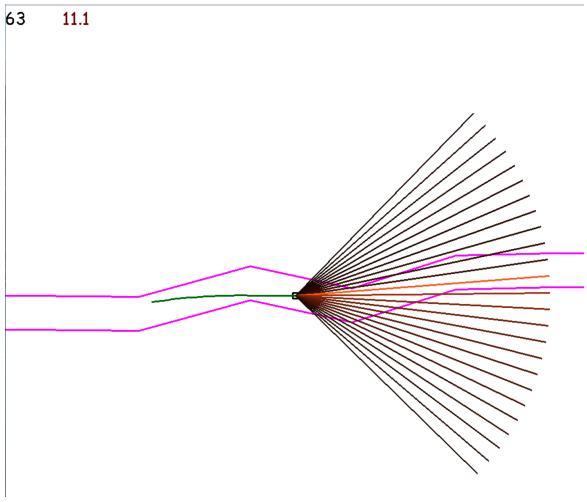

Рисунок 4 - Візуалізація зорових променів агента

в) реалізація керування;

У першій версії машинка завжди їхала з константою швидкістю і могла контролювати лише свій кут повороту. Саме через це було накладено декілька обмежень на трек, оскільки, маючи константу швидкість, без можливості гальмувати, проходити різкі повороти у машинки не було можливості.

Таким чином, в кожен момент часу у машинки є лише три варіанти дії: їхати в тому ж напрямку, звернути вліво, звернути вправо. При цьому також накладається обмеження щодо максимального куту повороту в кожен момент часу.

г) умови виходу та винагороди;

Є два варіанти завершення епізоду для машинки: дійти до фінішу або врізатися в стіну треку.

На кожному моменті часу ми перевіряємо умови завершення.

Якщо машинка врізалася в стіну - її винагорода буде дорівнювати нулю, якщо ж перетнула фінішну пряму - винагорода буде дорівнювати максимальній довжині епізоду мінус кількість кроків, які знадобились машинці аби доїхати до фінішу.

Таким чином ми мотивуємо агента перетнути фініш як можна швидше.

Реалізувавши всі необхідні компоненти середовища, можемо

переходити до тренування. Завершивши близько ста епох, машинка навчилася коректно проходити весь трек до самого фінішу, не зачіпаючи при цьому огородження треку.

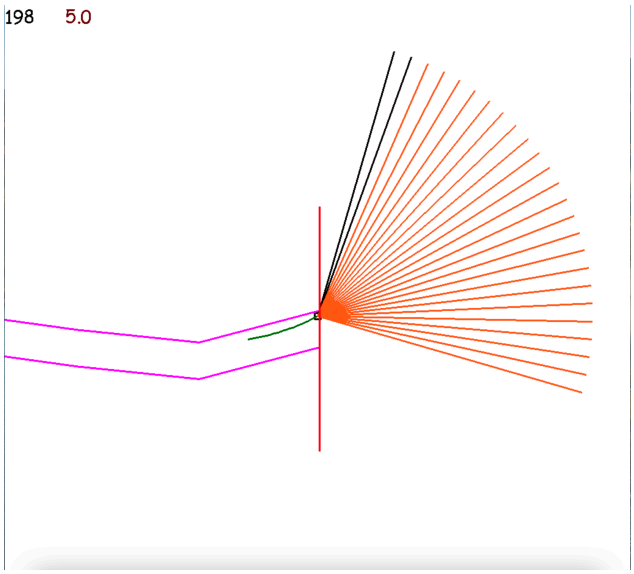

Рисунок 5 - Візуалізація проходження агентом фінішної прямої

5.2 Реалізація моноагентного середовища з двома керуваннями

Отримавши гарні результати тренувань першої версії, можна переходити до більш складної задачі - передачі агенту другого керування.

Очевидно, системі необхідно також керування швидкістю, аби мати можливість прискорюватись на рівних ділянках та скидати швидкість перед поворотами.

Наявність другого керування дозволяє нам також ускладнити трек, зробивши в ньому різкі повороти, які агент минулої версії не міг проходити.

Провівши тренування, додаємо ще один параметр до спостережень агента - кут наступного повороту. Умовно, тепер агент має деякий “дорожній знак”, який дає йому інформацію про наступний поворот, дозволяючи повернутись у правильному напрямку.

Також, зменшуємо максимальну видимість променів, адже тепер поворотів стало більше і вони є більш різкими. Тепер немає сенсу таких довгих променів, адже вони все частіше натикаються на перепону ближче, ніж за 100 пікселів від початку.

Також змінемо зоровий кут машинки. Тепер агент бачить на 180 градусів навколо себе, а не на 90, як це було в минулій версії.

Остання зміна, яку необхідно зробити в цій версії - переосмислити систему винагород. Аби агент простіше розумів, що йому необхідно розвивати вищу швидкість, де це можливо, будемо заохочувати його на кожному кроці виходячи зі швидкості.

Тепер функція винагороди виглядає таким чином:

Якщо агент врізався у стінку трека - винагорода дорівнює мінус 10.

Якщо агент дійшов до фінішу - винагорода дорівнює максимальній кількості кроків в середовищі (константа, вирахувана експерементальним шляхом, рівна 1000) мінус кількість кроків, яка знадобилась агенту, аби дійти до фінішу, поділена на 10.

В іншому випадку - швидкість агенту, розділена на 10.

Таким чином, агент вчиться, що найбільша можлива винагорода отримується за умови перетину фінішної лінії, розвиваючи найбільшу безпечну швидкість на шляху.

Усі параметри були підібрані експерементально, виходячи з результатів тренувань.

5.3 Реалізація мультиагентного середовища

Отримавши вичерпні результати при тренування одного агенту, має сенс розширити систему до декількох агентів, аби декілька машинок могли одночасно тренуватися в середовищі.

Для першої версії оберемо однакову стратегію для кожного з чотирьох агентів, таким чином, кожен агент, вивчаючи середовище, буде оновлювати одну й ту ж саму функцію. Маючи однакову стратегію, агенти будуть робити дії в однакових станах з однаковою вірогідністю. Це не означає, що вони будуть себе вести ідентично, адже стратегія є стохастичною функцією, проте

це дає можливість агентам використати спільну “мудрість”.

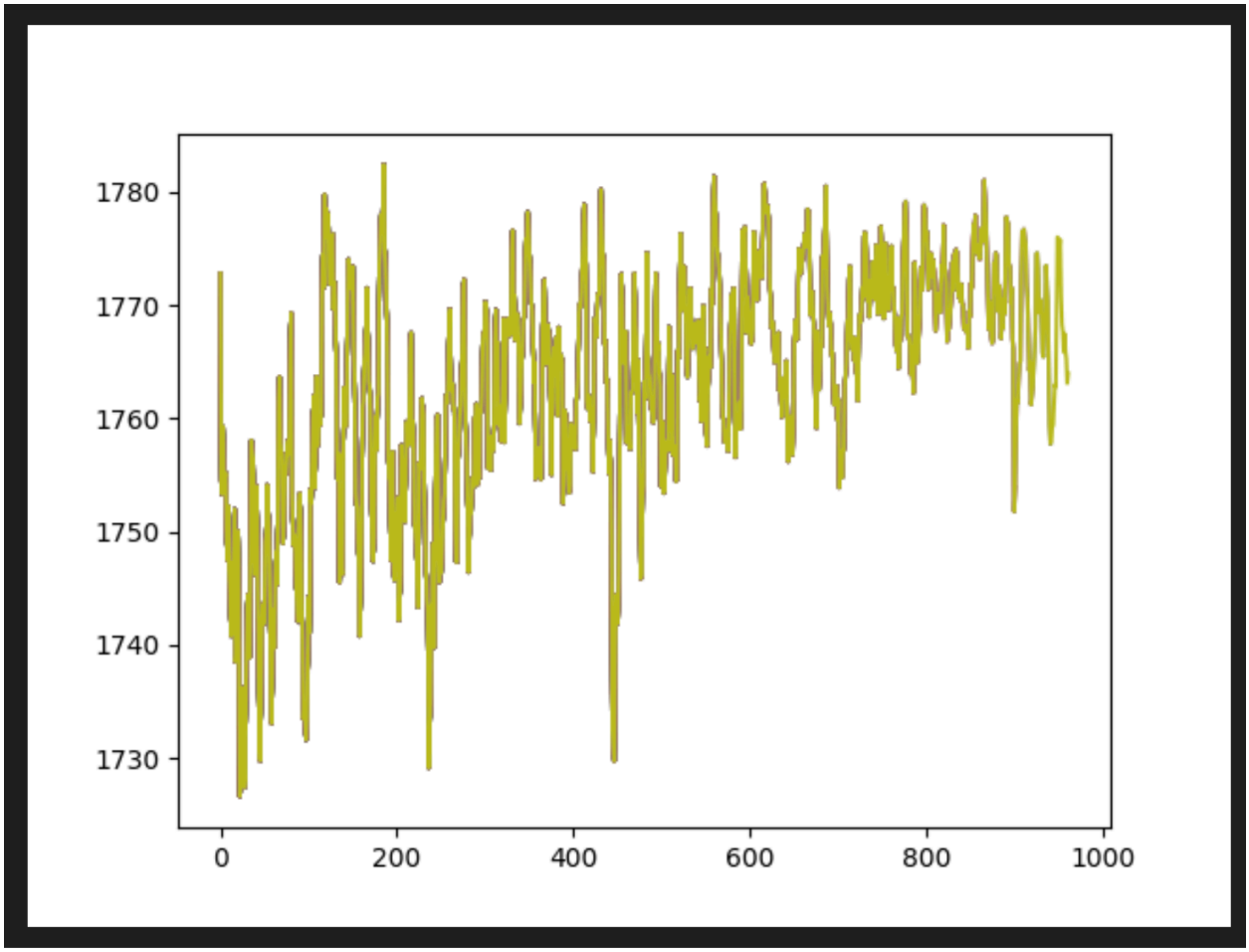

Рисунок 6 - Графік залежності отриманої винагороди від епохи тренування

Маємо графік залежності сумарно отриманої агентами винагороди від епохи тренування. Перед цим тренуванням, 400 епох тренування пройшли зі зростанням винагороди майже лінійно до 1700-1750 в середньому.

Аналізуючи наведений графік, бачимо, що з епохи номер 400 по номер 1400 ніяких суттєвих покращень не відбулося, винагорода лише коливається, в проміжку 1700-1800 одиниць.

Такий графік може свідчити про недосконалу систему винагороди, тобто, що агенти більше не знають, як покращити свою винагороду, через неправильно задані співвідношення між винагородами в різних сценаріях поведінки агента.

Також це може свідчити про те, що агенти досягли дійсно оптимальної схеми дій в заданому середовищі і ці коливання залежать лише від випадково генерованого треку.

Ми відкидаємо можливість пристосування агентами до конкретного треку, адже кожен епізод ми генеруємо його випадковим чином.

Рисунок 7 - Результат тренування декількох агентів

Переглянувши результати тренувань, бачимо, що усі агенти успішно доходять до фінішу, маючи при цьому на майже фінальному кроці однакову винагороду. Це результат того, що ми маємо однакову стратегію для всіх агентів.

В даній реалізації, агенти ніяк не взаємодіють один з одним. Немає сценарію “аварії” між декількома агентами, машинки нічого не знають одна про одну, по суті ми маємо просто декілька шарів треку, окремий шар - для кожного окремого агента.

Проте, ми маємо успішно натреноване середовище, де одночасно можуть перебувати декілька агентів, що є значним прогресом у порівнянні з попередньою версією.

5.4 Реалізація мультиагентного середовища із взаємодією

Ми провели тренування минулої версії, звідки зрозуміло, що маючи однакову стратегію, агенти будуть себе вести ідентично в однакових умовах. Така концепція не дуже схоже на те, що відбувається в реальному житті, адже всі розумні створіння в природі - мають свій власний досвід і можливість взаємодіяти.

Таким чином, необхідно реалізувати дві речі - надати всім агентам свою особисту стратегію, аби вони не розділяли знання один між одним; а також передати кожному агенту інформацію щодо решти агентів та внести корективи в систему винагород.

Зміни в реалізації:

а) вводимо окрему стратегію для кожного агента;

б) в перелік спостережень агента тепер передаємо відстань до решти машинок, а також кут між заданим агентом і кожним із суперників;

в) в перелік можливих ситуацій завершення епізоду для агенту, додаємо опцію “аварії” із іншою машинкою. Таким чином, машинки, які врізались одна в одну або в стіну треку, вибувають з перегонів і перестають бути видимими для інших агентів. Також, агенти, які перетнули фінішну лінію, також стають невидимими для інших.

Перед запуском тренування, робимо довжину треку трохи меншою, аби епізод проходив швидше і агенти тренувались не так повільно.

Аналізуючи результати тренування перших 160 епох бачимо, що винагорода зростає стабільно, а до епохи номер 60, взагалі можна було спостерігати монотонне зростання.

Таким чином, можна зробити висновки, що тренування проходить впевнено, агенти вчаться взаємодіяти один з одним та уникати аварій.
Враховуючи те, що ми зменшили розміри треку вдвічі, а отже середня винагорода від проходження агентом треку має також сильно зменшитися, приблизно у 1.5 - 1.7 рази. Під час минулого тренування, отриманий

результат був близько 1700-1800, отже на цьому тренуванні очікуємо результат близько 900-1100. Такий результат будемо вважати, як гарний.

Дочекавшись доки агенти завершать тренування та будуть отримувати в середньому близько 900 одиниць винагороди, можемо бачити результат.

Усі агенти тримають дистанцію один від одного, аби встигнути зреагувати на дії агента попереду і не допустити аварійної ситуації.

Доходять агенти до фінішу по черзі, не намагаючись обігнати один одного, проте завжди тримаючи гарну швидкість.

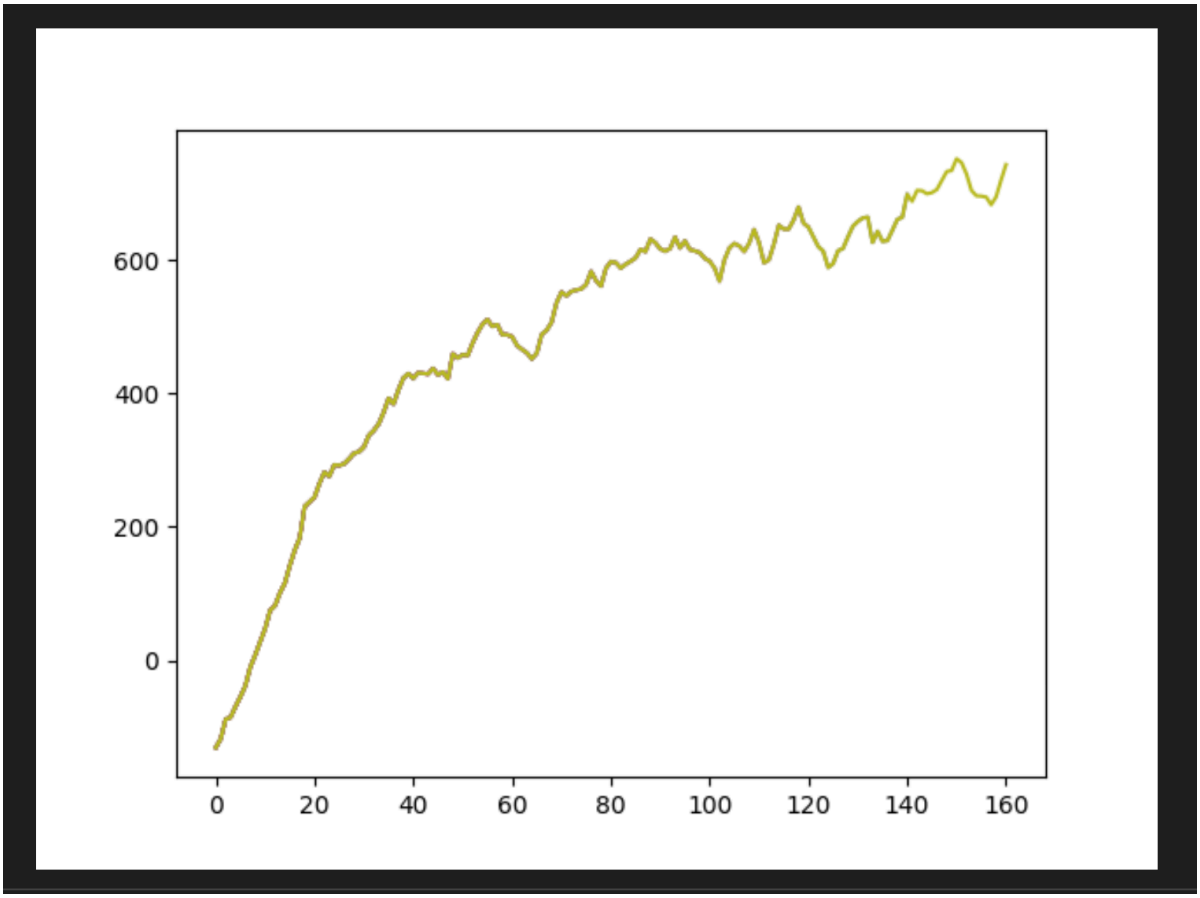

Рисунок 8 - Графік залежності отриманої винагороди від епохи тренування

Можемо вважати, що тренування пройшло успішно, адже поставлена задача була виконана, тепер агенти вміють взаємодіяти одним з одним в середовищі та знають про позицію один одного, при цьому вміють запобігати аварійним ситуаціям.

ВИСНОВКИ

Навчання з підкріпленням є потужним методом машинного навчання, за допомоги якого можливо навчити машину розв’язувати велику кількість нетривіальних завдань, рішення яких суттєво можуть спростити та покращити людське життя.

За допомогою reinforcement learning можливо вирішувати надзвичайно велику кількість не схожих одну на одну задач, починаючи від розробки ботів для відеоігор, які кіберспортсмени та просто геймери можуть використовувати для власних тренувань та пошуку оптимальних стратегій у іграх, та закінчуючи розробкою штучного інтелекту для самокерованих автомобілей та інтернету речей.

В ході виконання завдання, було визначено, що для правильної роботи алгоритму та досягнення найкращого результату необхідно приділити багато часу створенню середовища та вибору підходящої під задачу стратегії.

Варто розуміти, що обравши навчання з підкріпленням, як метод розв’язання поставленої задачі, необхідно розробляти власне середовище та підбирати стратегію конкретно під проблему, адже, на відміну від навчання з вчителем, тут майже неможливо повторно використати вже написану модель.

При виборі технологій для виконання поставленої задачі, було обрано state-of-the-art технології, які активно використовуються сучасними розробниками, як для власних приватних проектів, так і для великих прикладних систем.

В обраній бібліотеці для реалізації навчання з підкріпленням - RLlib, чудово реалізовані можливості тренування агентів, як в моноагентному середовищі, так і в мультиагентному. При цьому, перехід від одного до другого здійснюється досить просто, враховуючи факт того, ще тренування можна проводити розподілено.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Chip Design with Deep Reinforcement Learning [Електронний ресурс] – Режим доступу до ресурсу: https://ai.googleblog.com/2020/04/chip-design-with-deep-reinforcement.html.

2. AlphaGo: Mastering the ancient game of Go with Machine Learning [Електронний ресурс] – Режим доступу до ресурсу: https://ai.googleblog.com/2016/01/alphago-mastering-ancient-game-of-go.html.

3. Мал. 1. Машинне навчання [Електронний ресурс] – Режим доступу до ресурсу: https://se.mathworks.com/discovery/reinforcement-learning/_jcr_content/mainParsys3/discoverysubsection/mainParsys/image.adapt.full.medium.png/1647932644734.png.

4. Reinforcement learning: What Is, Algorithms, Types & Examples [Електронний ресурс] – Режим доступу до ресурсу: https://www.guru99.com/reinforcement-learning-tutorial.html.

5. Reinforcement learning algorithms - an intuitive overview [Електронний ресурс] – Режим доступу до ресурсу: https://smartlabai.medium.com/reinforcement-learning-algorithms-an-intuitive-overview-904e2dff5bbc.

6. A Gentle Introduction to Reinforcement Learning [Електронний ресурс] – Режим доступу до ресурсу: https://aamrani1999.medium.com/a-gentle-introduction-to-reinforcement-learning-d26cba6455f7.

7. Мал. 2. Навчання з підкріпленням [Електронний ресурс] – Режим доступу до ресурсу: https://lilianweng.github.io/posts/2018-02-19-rl-overview/RL_illustration.png.

8. Jang, Beakcheol & Kim, Myeonghwi & Harerimana, Gaspard & Kim, Jong. (2019). Q-Learning Algorithms: A Comprehensive Classification and Applications. IEEE Access. PP. 1-1. 10.1109/ACCESS.2019.2941229.

9. John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, Oleg Klimov. (2017). Proximal Policy Optimization Algorithms.

10. The Best Tools For Reinforcement Learning In Python You Actually Want To Try [Електронний ресурс] – Режим доступу до ресурсу: https://neptune.ai/blog/the-best-tools-for-reinforcement-learning-in-python.

11. Ray project - Github [Електронний ресурс] – Режим доступу до ресурсу: https://github.com/ray-project/ray.

12. Dopamine - Github [Електронний ресурс] – Режим доступу до ресурсу: https://github.com/google/dopamine.

13. Tensorforce - Github [Електронний ресурс] – Режим доступу до ресурсу: https://github.com/tensorforce/tensorforce.

14. MushroomRl - Github [Електронний ресурс] – Режим доступу до ресурсу: https://github.com/MushroomRL/mushroom-rl.

15. RL_Coach - Github [Електронний ресурс] – Режим доступу до ресурсу: https://github.com/IntelLabs/coach.