НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

Кафедра системного програмування і спеціалізованих комп‘ютерних систем

«До захисту допущено»

Завідувач кафедри

_____ Тарасенко В.П.
(підпис) (ініціали, прізвище)

“___” червня 2018 р.

Дипломний проект

на здобуття ступеня бакалавра

з напряму підготовки **6.050102 «Комп'ютерна інженерія»**
на тему: Комп'ютерна система управління промисловим роботом

Виконав (-ла): студент (-ка) IV курсу, групи КВ-52
Ковальов Костянтин Миколайович _____

Керівник проф. каф. СПСКС, д.т.н., проф. Терейковський І.А. _____

Консультанти

- з нормоконтролю, доц.каф.СПСКС, к.т.н., доц. Клятченко Я.М. _____

Рецензент_____ _____

Засвідчую, що у цьому дипломному проекті немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ - 2019

АНОТАЦІЯ

Кваліфікаційна робота включає пояснювальну записку (56 с., 2 додатка).

Об’єкт дослідження – алгоритми навчання з підкріпленням для задачі керування промисловою роботичною рукою.

Задача непервного керування промисловою роботичною рукою для нетривіальних задач є занадто складною або навіть невирішуваною для класичних методів робототехніки. Методи навчання з підкріпленням можуть бути використані в цьому випадку. Вони є досить простими у реалізації, дозволяють узагальнюватися на небачені випадки, та вчитися на даних великої розмірності. Ми реалізуємо метод градієнту глибокої детермінованої стратегії, який підходить для складних задач непервного управління.

В ході дослідження:

- проведено аналіз існуючих класичних методів для задачі управління промисловим роботом
- проведено аналіз існуючих алгоритмів навчання з підкріпленням та їх використання в області робототехніки
- реалізовано алгоритм градієнту глибокої детермінованої стратегії
- проведено тестування реалізованого алгоритму у спрощеному середовищі
- запропоновано архітектуру нейронної мережі для вирішення поставленої задачі
- проведено тестування алгоритму на навчальній виборці
- проведено тестування алгоритму на здатність до узагальнення на тестовій виборці

Показано здатність алгоритму градієнту глибокої детермінованої стратегії з використанням нейронних мереж для представлення стратегії вирішувати поставлену задачі з зображенням в якості входу та узагальнюватися на небачені до цього об’єкти.

Ключові слова:

НАВЧАННЯ З ПІДКРІПЛЕННЯМ, РОБОТОТЕХНІКА, ГРАДІЄНТ СТРАТЕГІЇ, МАРКОВСЬКИЙ ПРОЦЕС ВИРІШУВАННЯ, НЕЙРОННА МЕРЕЖА

ABSTRACT

Qualifying work includes an explanatory note (56 p., 2 appendix).

The object of the study are reinforcement learning algorithms for the task of an industrial robotic arm control.

Continuous control of an industrial robotic arm for non-trivial tasks is too complicated or even unsolvable for classical methods of robotics. Reinforcement learning methods can be used in this case. They are quite simple to implement, allow for generalization to unseen cases, and learn from high-dimensional data. We implement deep deterministic policy gradient algorithm that is suitable for complex continuous contol tasks.

During the study:

- An analysis of existing classical methods for the problem of industrial robot control was conducted
- An analysis of existing algorithms of training with reinforcement learning and their use in the field of robotics has been conducted
- Deep deterministic policy gradient algorithm is implemented
- Implemented algorithm is tested on a simplified environment
- The architecture of the neural network is proposed for solving the problem
- Algorithm was tested on the training set of objects
- Algorithm was tested for its generalization ability on the test set

It was shown that deep deterministic policy gradient algorithm with neural network as policy approximator is able to solve the problem with the image as an input and to generalize to objects not seen before.

Keywords:

REINFORCEMENT LEARNING, ROBOTICS, POLICY GRADIENTS, MARKOV DECISION PROCESS, NEURAL NETWORK

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

Кафедра системного програмування і спеціалізованих комп‘ютерних систем

Рівень вищої освіти – перший (бакалаврський)

Напрям підготовки 6.050102 «Комп’ютерна інженерія»

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Тарасенко В.П.
(підпис) (ініціали, прізвище)
«___» червня 2019 р.

ЗАВДАННЯ
на дипломний проект студента

Ковальова Костянтина Миколайовича

1. Тема проекту: « Комп’ютерна система управління промисловим роботом», керівник проекту Терейковський Ігор Анатолійович, доц. каф. СПіСКС, к.т.н., затверджені наказом по університету від «22» квітня 2019 р. №1330-С

2. Термін подання студентом проекту «08» червня 2019 р.

3. Вихідні дані до проекту: див. технічне завдання.

4. Зміст пояснювальної записки: аналіз існуючих рішень та обґрунтування теми дипломного проекту; особливості розробки ПЗ; розробка ПЗ.

5. Перелік графічного матеріалу (із зазначенням обов’язкових креслеників, плакатів, презентацій тощо): Схема алгоритму ГГДС. Схема структурна; Взаємодія модулів програми. Схема структурна; Архітектура нейронної мережі актора алгоритму ГГДС. Схема структурна; Архітектура нейронної мережі критика алгоритму ГГДС. Схема структурна.

6. Консультанти розділів проекту*

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Нормоконтроль	Клятченко Я.М., доц. каф. СПіСКС, к.т.н.		

7. Дата видачі завдання: 15.10.2018 р.

Календарний план

№ з/п	Назва етапів роботи та питань, які мають бути розроблені відповідно до завдання	Термін виконання
1.	Видача завдання на дипломне проектування	15.10.2018
2.	Розробка технічного завдання	05.04.2019
3.	Аналіз існуючих рішень	07.04.2019
4.	Реалізація алгоритму	15.04.2019
5.	Тестування алгоритму на спрощеному середовищі	25.04.2019
6.	Створення середовища комп’ютерної симуляціїї задачі	30.04.2019
7.	Тестування алгоритму на складному середовищі	10.05.2019
8.	Підготовка пояснювальної записки	15.05.2019
9.	Оформлення матеріалів проекту	20.05.2019
10.	Попередній огляд матеріалів диплому на кафедрі	29.05.2019

Студент _____ Ковальов К.М.
(підпис)

Керівник проекту _____ Терейковський І.А.
(підпис)

* Консультантом не може бути зазначено керівника дипломного проекту.

Поз.	Формат	ПОЗНАЧЕННЯ	НАЙМЕНУВАННЯ	Кіл-ть. арк.	№ прим.	Прим.
			Документація загальна			
			Новорозроблена			
	А4	ІАЛЦ.045420.002 ТЗ	Комп’ютерна система	3		
			управління промисловим			
			роботом			
			Технічне завдання			
	А4	ІАЛЦ.045420.003 ТП	Комп’ютерна система	2		
			управління промисловим			
			роботом			
			Відомість технічного			
			проекту			
	А4	ІАЛЦ.045420.004 ПЗ	Комп’ютерна система	56		
			управління промисловим			
			роботом			
			Пояснювальна записка			
	А1	ІАЛЦ.045420.005 Д1	Комп’ютерна система	1		
			управління промисловим			
			роботом			
			Градієнт глибокої			
			детермінованної стратегії.			
			Схема алгоритму			

Змін.	Арк.	№ докум.	Підпис	Дата	ІАЛЦ.045420.001 ОА		
Розробив		Ковальов К.М.			Комп’ютерна система управління промисловим роботом Опис альбому	Літ.	Аркуш 1 / Аркушів 2
Перевірив		Терейковський І.А.					
Н. контроль		Клятченко Я.М.				КПІ ім. Ігоря Сікорського, ФПМ, КВ-52	
Затвердив		Тарасенко В.П.					

Поз.	Формат	ПОЗНАЧЕННЯ	НАЙМЕНУВАННЯ	Кіл-ть. арк.	№ прим.	Прим.
	А1	ІАЛЦ.045420.005 Д2	Комп’ютерна система	1		
			управління промисловим			
			роботом			
			Взаємодія модулів			
			програми			
	А1	ІАЛЦ.045420.005 Д3	Комп’ютерна система	1		
			управління промисловим			
			роботом			
			Архітектура нейронної			
			мережі актора алгоритму			
			ГГДС			
	А1	ІАЛЦ.045420.005 Д4	Комп’ютерна система	1		
			управління промисловим			
			роботом			
			Архітектура нейронної			
			мережі критика алгоритму			
			ГГДС			
		Диск CD-ROM	Текст ПЗ. Тексти програм.	1		
			Графічний матеріал.			
			Презентація			

Поз.	Формат	ПОЗНАЧЕННЯ	НАЙМЕНУВАННЯ	Кіл-ть. арк.	№ прим.	Прим.
			Документація загальна			
			Новорозроблена			
	А4	ІАЛЦ.045420.004 ПЗ	Комп’ютерна система	56		
			управління промисловим			
			роботом			
			Пояснювальна записка			
	А1	ІАЛЦ.045420.005 Д1	Комп’ютерна система	1		
			управління промисловим			
			роботом			
			Градієнт глибокої			
			детермінованної стратегії.			
			Схема алгоритму			
	А1	ІАЛЦ.045420.005 Д2	Комп’ютерна система	1		
			управління промисловим			
			роботом			
			Взаємодія модулів			
			програми			
	А1	ІАЛЦ.045420.005 Д3	Комп’ютерна система	1		
			управління промисловим			
			роботом			
			Архітектура нейронної			
			мережі актора алгоритму			
			ГГДС			

					ІАЛЦ.045420.003 ТП
Змін.	Арк.	№ докум.	Підпис	Дата	
Розробив		Ковальов К.М.			Комп'ютерна система управління промисловим роботом Відомість технічного проекту
Перевірив		Терейковський І.А			
Н. контроль		Клятченко Я.М.			
Затвердив		Тарасенко В.П.			

Літ.	Аркуш	Аркушів
	1	2

КПІ ім. Ігоря Сікорського, ФПМ, КВ-52

Поз.	Формат	ПОЗНАЧЕННЯ	НАЙМЕНУВАННЯ	Кіл-ть. арк.	№ прим.	Прим.
	А1	ІАЛЦ.045420.005 Д4	Комп’ютерна система	1		
			управління промисловим			
			роботом			
			Архітектура нейронної			
			мережі критика алгоритму			
			ГГДС			
		Диск CD-ROM	Текст ПЗ. Тексти програм.	1		
			Графічний матеріал.			
			Презентація			

Змін.	Арк.	№ докум.	Підпис	Дата	ІАЛЦ.045420.003 ТП	Арк 2

ЗМІСТ

СПИСОК ТЕРМІНІВ, СКОРОЧЕНЬ ТА ПОЗНАЧЕНЬ

МПВ — Марковський процес вирішування

ДП — динамічне програмування

НМ — нейронна мережа

ЗНМ — згорткова нейронна мережа

ГС — градієнт стратегії

ГДС — градієнт детермінованої стратегії

ГГДС — градієнт глибокої детермінованої стратегії

ВСТУП

Промисловий робот - автоматична машина з програмним керуванням, яка відтворює рушійні і розумові функції людини при виконанні виробничих процесів. Сучасне покоління промислових роботів використовує штучний інтелект для вирішення складних задачь керування. Одним з методів штучного інтелекту, придатного для керування промисловим роботом, є навчання з підкріпленням.

Навчання з підкріпленням — це галузь машинного навчання, що займається питанням про те, які дії повинні виконувати програмні агенти в певному середовищі задля максимізації деякого уявлення про сукупну винагороду.

Навчання з підкріпленням має багато практичних застосувань через свою загальність і велике зростання обчислювальних ресурсів за останні десятиліття. Відомими прикладами застосування на практиці є:

- управління ресурсами в комп'ютерних кластерах [23]
- задачі неперервного керування в робототехніці, таких як переміщення робота, балансування і т. д. [3]
- системи рекомендації новин [24]
- вирішення важких ігор, таких як ГО [6], шахи, відео-ігри Atari [5] тощо
- оптимізація хімічних реакцій [22]

У даній роботі ми зосереджуємося на застосуванні алгоритмів навчання з підкріпленням до задач робототехніки, зокрема до завдання захоплення об'єктів промисловою роботичною рукою з використанням

зору як вхідного сигналу для алгоритму навчання. Робот повинен вміти захоплювати різні види об'єктів.

Класичні методи робототехніки, що використовують теорію оптимального керування, не здатні надійно вирішити це завдання, оскільки вони зазвичай розробляються для конкретної задачі і не можуть узагальнюватися на інші задачі (у нашому випадку захоплення невідомих об'єктів на основі зору). З іншого боку, алгоритми навчання з підкріпленням у поєднанні з узагальнювальною потужністю нейронних мереж здатні вирішити цю і багато інших задач робототехніки.

Для моделювання задачі використовується віртуальне середовище, яке було створено для цієї роботи. Навчання агенту виконується за допомогою алгоритму градієнту глибокої детермінованої стратегії (ГГДС), який показав себе досить надійним для багатьох завдань неперервного керування.

1. АНАЛІЗ ІСНУЮЧИХ РІШЕНЬ ТА ОБГРУНТУВАННЯ ТЕМИ ДИПЛОМНОЇ РОБОТИ

1.1. Загальні відомості про задачу керування промисловим роботом

Промисловий робот — багатоцільовий маніпуляційний робот, що складається з механічного маніпулятора і перепрограмованої системи керування, який застосовується для переміщення об'єктів в просторі трьох і більше координат та для виконання різноманітних виробничих процесів.

Промислові роботи є важливими компонентами автоматизованих гнучких виробничих систем, які дозволяють збільшити продуктивність праці. Типове застосування роботів стосується таких операцій, як зварювання, фарбування, складання, вибірка та встановлення, пакування, контроль продукції та випробування, котрі виконуються з високою надійністю, швидкістю, і точністю.

Перші роботи з'явилися в США в 60-х роках минулого століття. Промисловий робот був оснащений двопальцевим пристроєм для захоплення на пневмоприводі та «рукою» на гідроприводі з п'ятьма ступенями свободи. Його характеристики дозволяли переміщати 12-кілограмову деталь із точністю до 1,25 мм. «Розумна» машина запам'ятовувала координати точок свого маршруту й виконувала роботу згідно з програмою. Такі промислові роботи, яких можна вважати роботами першого покоління, у вартісному вираженні містили 75 % механіки і 25 % електроніки. Їх переналагодження вимагало часу й

обумовлювалося простотою устаткування. Для перепрофілювання їх з метою виконання нової роботи проводилася заміна програми керування.

Друге покоління передбачало більш тонке керування промисловими роботами — адаптивне. Роботи вимагали «впорядкування» середовища, в якому вони працювали. Тому цей етап характеризувався розробкою безлічі датчиків, за допомогою яких робот отримав так звану чутливість. Завдяки їй він отримував інформацію про зовнішнє середовище і у взаємо узгодженості із довкіллям вибирав оптимальний варіант дії. Наприклад, взявши деталь, маніпулятор робота самостійно оминав з нею перешкоди. Відбувається така дія завдяки мікропроцесорній обробці отриманої інформації. Операції при їх виконанні підлягають адаптації, тобто ініціюється багатоваріантність для покращення якості виконання будь-якої функції.

Адаптивне керування в основному здійснюється програмно й базується на багатоваріантному програмному забезпеченні. При цьому рішення про вибір типу роботи програми приймається роботом на підставі інформації про середовище, описаної детекторами.

Характерною рисою функціонування робота другого покоління є попереднє встановлення режимів роботи, кожен з яких активується при певних показниках, отриманих із зовнішнього середовища.

Роботи-автомати третього покоління здатні самостійно генерувати програму своїх дій залежно від завдання й умов зовнішнього середовища. У них немає «шпаргалок», тобто розписаних технологічних дій при певних варіантах зовнішнього середовища. Вони володіють умінням самостійно оптимально вибудовувати алгоритм своєї роботи, а також

оперативно реалізовувати його практично. Вартість електроніки такого промислового робота в десятки разів вища його механічної частини.

Новітній робот, здійснюючи захоплення деталі завдяки сенсорам, «знає», наскільки вдало він це зробив. Крім того, регулюється сама сила захоплення (завдяки зворотному зв'язку по зусиллю) у залежності від крихкості матеріалу деталі. Можливо, саме тому пристрій промислових роботів нового покоління називають інтелектуальним. «Мозком» такого приладу є система його керування. Найбільш перспективним є регулювання, яке здійснюється відповідно до методів штучного інтелекту. Інтелект цим машинам задають пакети прикладних програм, програмовані логічні контролери, інструменти моделювання. Одним з таких методів штучного інтелекту є метод навчання з підкріпленням, який буде застосовуватися у цій роботі.

1.2. Навчання з підкріпленням

Навчання з підкріпленням — це область машинного навчання, головною задачею якої є знаходження відображення подій до дій таким чином, щоб максимізувати числовий сигнал винагороди. Агенту не говориться які дії слід приймати. Замість цього він має дослідити які дії дають найбільшу винагороду, виконуючи їх. У найбільш цікавих і складних випадках дії можуть впливати не тільки на найближчу винагороду, але й на винагороду з наступної події, і всі подальші винагороди. Ці дві характеристики — пошук методом спроб і помилок та відкладена винагорода — дві найважливіші відмінні риси начавчання з підкріпленням.

Навчання з підкріпленням відрізняється від навчання з учителем тим, що не потрібно подавати позначенні пари вводу/виводу, а також, тим що неоптимальні дії не потрібно явно виправляти.

Однією з проблем, що виникають у навчанні з підкріпленням, але не в інших видах навчання, є компроміс між дослідженням та експлуатацією. Щоб отримати багато винагороди, агент повинен віддавати перевагу діям, які він виконував в минулому, і виявив, що вони є ефективними у отриманні нагороди. Але щоб виявити такі дії, він повинен спробувати дії, які він не обирав раніше. Агент повинен використовувати те, що вже пережив, щоб отримати винагороду, але він також повинен досліджувати, щоб зробити кращий вибір дій у майбутньому. Дилема полягає в тому, що ні дослідження, ні експлуатація окремо не можуть бути використані для розв’язання багатьох задач. Агент повинен спробувати різні дії і поступово віддавати перевагу тим, що здаються найкращими. У випадку стохастичної задачі, кожну дію потрібно виконати багато разів, щоб отримати достовірну оцінку очікуваної винагороди.

З усіх форм машинного навчання, навчання з підкріпленням є найбільш близьким до такого роду навчання, яке роблять люди та тварини, і багато з основних алгоритмів навчання з підкріпленням спочатку брали натхнення з біологічних систем навчання. Навчання з підкріпленням також підкріплюється, як психологічною моделлю навчання у тварин, яка краще відповідає деяким емпіричним даним, так і через впливову модель системи винагороди у мозку.

Навчання з підкріпленням успішно застосовувалося для вирішення важких ігор, таких як настільні ігри (ГО, шахи) або відео-ігри Atari, а також проблеми робототехніки, охорони здоров'я, торгівлі, тощо. У

робототехніці воно використовувалося для вирішення проблем керування великої розмірності, таких як рух, балансування.

1.3. Марковський процес вирішування (МПВ)

Ланцюг Маркова - це випадковий процес, який відбувається в наборі кроків, в кожному з яких робиться випадковий вибір серед скінченної (або зліченної) кількості станів. Так як набір індексів і простір станів дискретні, позначимо $X_n \equiv X(t_n)$; ймовірність переходу тоді може бути представлена матрицею $P = p_{ij}$, де p_{ij} - ймовірність переходу від стану і до стану j: $p_{ij} = Prob[X_{n+1} = j | X_n = i]$. Для однорідних ланцюгів ці ймовірності не залежать від t, тобто вони стаціонарні. Тоді початковий розподіл разом з матрицею переходів P визначають розподіл ймовірностей для будь-якого стану в будь-який наступний проміжок часу.

Всі елементи матриці P є невід'ємними, а суми рядків дорівнюють одиниці; такі матриці називаються стохастичними матрицями. Стани ланцюга Маркова є рекурентними, якщо ймовірність можливого повернення до цього стану дорівнює 1, або перехідна, інакше. Типи станів і властивості ланцюга відносяться до властивостей стохастичної матриці

Марковські процеси вирішування (МПВ) є узагальненням ланцюгів Маркова. МПВ забезпечують математичну систему для моделювання ухвалення рішень у ситуаціях, в яких наслідки є частково випадковими, а частково контрольованими ухвалювачем рішення. МПВ є корисними для дослідження широкого спектру задач оптимізації, розв'язуваних динамічним програмуванням та навчанням з підкріпленням.

Задача Марківського процесу вирішування - кортеж (S, A, P, R, γ), де

- S — множина станів
- A — множина дій
- $P_a(s,s')=Pr(s_{t+1}=s'|s_t=s,a_t=a)$ - ймовірність того, що дія a в стані s в момент часу t призведе до стану s' в момент часу $t+1$
- $R_a(s,s')$ є безпосередньою винагородою (або очікуваною безпосередньою винагородою), отримуваною після переходу до стану s'
- $\gamma\in[0,1]$ є коефіцієнтом знецінювання, який представляє важливість майбутніх і поточних винагород

Мета полягає в мінімізації (очікуваних) накопичених витрат або, що еквівалентно, максимізації (очікуваних) накопичених винагород. Рішення МПВ надається в термінах стратегії π, яка задає відображення стану до дії, що приймається в цьому стані.

1.4. Математична модель навчання з підкріпленням

Стандартна математична модель навчання з підкріпленням складається з агенту, який взаємодіє з середовищем E в дискретні часові проміжки. На кожному часовому проміжку t агент отримує спостереження x_t, виконує дію та отримує скалярну винагороду r_t. В усіх середовищах розглянутих у даній роботі дії є дійсними числами: $a_t\in R^N$. В загальному випадку, середовище може бути частково спостережуваним, тобто вся історія пар спостереження-дія $s_t=(x_1,a_1,\ldots,a_{t-1},x_t)$ може бути необхідно для того, щоб описати стан. В даній роботі, робиться припущення, що середовище є повністю спостережуваним, тобто $s_t=x_t$.

Поведінка агента визначається стратегією π, яка є відображення стану у розподіл ймовірності над діями $\pi: S \to P(A)$. Середовище E також может бути стохастичним. Ми моделюємо взаємодію з середовищем, як Марковський процес вирішення з множиною станів S, множиною дій $A \in R^N$, початковим розподілом станів $p(s_1)$, динамікою переходів $p(s_{t+1}|s_t, a_t)$, та функцією винагороди $r(s_t, a_t)$. Віддача стану визначається як сума знеціненної майбутньої винагороди $R_t = \prod_{i=t}^{T} \gamma^{i-t} r(s_i, a_i)$ з коефіцієнтом знецінення $\gamma \in [0,1]$. Зауважимо, що віддача залежить від обраних дій, а отже, і від стратегії π, і може бути стохастичною. Метою навчання з підкріпленням є вивчення стратегії, яка максимізує очікувану віддачу від стартового розподілу $J = E_{r_{i \geq t}, s_{i>t}\ E, a_{i>t}\ \pi}[R_t | s_t, a_t]$

Майже всі алгоритми навчання з підкріпленням передбачають оцінювання ***функцій цінностей*** - функцій станів (або пар стан-дія), які оцінюють, наскільки добре для агента бути в певному стані (або наскільки добре він виконує дану дію в даному стані). Поняття «наскільки добре» тут визначається з точки зору майбутніх винагород, які можна очікувати.

Значення стану s за стратегії π, позначене як $v_\pi(s)$, є очікуваною віддачею при старті в стані s і слідуючи стратегії π після цього. Для МДП формально можна визначити v_π наступним чином:

$$v_\pi(s) \doteq \mathbb{E}_\pi[G_t \mid S_t = s] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \,\middle|\, S_t = s\right], \text{ for all } s \in \mathcal{S}, \quad (1.1)$$

де $E_\pi[.]$ позначає математичне сподівання випадкової величини, якщо агент дотримується стратегії π, а t - будь-який крок часу. Зауважимо, що значення термінального стану, якщо воно є, завжди дорівнює нулю. Функцію v_π називається функцією цінності стану для стратегії π.

Аналогічно визначимо цінність дії *a* в стані *s* зі стратегією π, як очікувана віддача, починаючи з *s*, виконуючи дію *a,* і таким чином слідуючи стратегії π:

$$q_\pi(s,a) \doteq \mathbb{E}_\pi[G_t \mid S_t=s, A_t=a] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty}\gamma^k R_{t+k+1} \,\middle|\, S_t=s, A_t=a\right]. \quad (1.2)$$

Функція q_π називається функцією цінності стану та дії для стратегії π

Фундаментальна властивість функцій цінності, що використовуються у навчанні з підкріпленням та динамічному програмуванні, полягає в тому, що вони задовольняють рекурсивним співвідношенням. Для будь-якої стратегії π та будь-якого стану s виконується наступна умова узгодженості між цінністю стану s та цінністю можливих наступних станів:

$$\begin{aligned} v_\pi(s) &\doteq \mathbb{E}_\pi[G_t \mid S_t=s] \\ &= \mathbb{E}_\pi[R_{t+1}+\gamma G_{t+1} \mid S_t=s] \\ &= \sum_a \pi(a|s)\sum_{s'}\sum_r p(s',r|s,a)\Big[r+\gamma\mathbb{E}_\pi[G_{t+1}|S_{t+1}=s']\Big] \\ &= \sum_a \pi(a|s)\sum_{s',r} p(s',r|s,a)\Big[r+\gamma v_\pi(s')\Big], \quad \text{for all } s\in\mathcal{S}, \end{aligned} \quad (1.3)$$

Рівняння (1.3) - це ***рівняння Белмана*** для v_π. Воно виражає зв'язок між цінністю станів і цінностями його станів-наступників.

Для візуалізації того, що відбувається у рівнянні Белмана, можна використати діаграму, як на рис. 1.1. Кожне незафарбоване коло представляє собою стан, а кожне зафарбоване коло являє собою пару стан-дія. Починаючи з стану *s*, кореневого вузла у верхній частині, агент міг би прийняти будь-який з деяких наборів дій - три показано на діаграмі - на основі своєї стратегії π. З кожного з них середовище може реагувати

одним з декількох наступних станів, s' (два на рисунку), поряд з винагородою, r, залежно від його динаміки, заданої функцією p. Рівняння Белмана (1.3) усереднює всі ймовірності, зважуючи кожну з них ймовірністю виникнення. У ньому зазначається, що цінність стартового стану повинна дорівнювати (знеціненній) цінності очікуваного наступного стану, плюс винагорода, що очікується на цьому шляху.

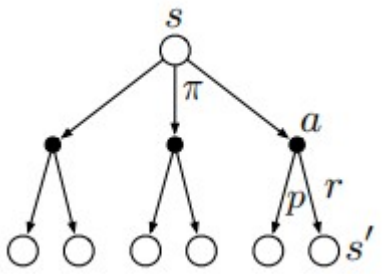

Рисунок 1.1 - Діаграмма рівняння Белмана

Вирішення задачі навчання з підкріпленням означає, грубо кажучи, знаходження стратегії, яка досягає багато винагороди в довгостроковій перспективі. Функції цінності визначає часткове упорядкування над стратегіями. Стратегія π є кращою або рівною стратегії π', якщо її очікувана віддача більша або дорівнює віддачі π' для всіх станів. Інакше кажучи, $\pi \geq \pi'$ тоді і тільки тоді, коли $v_\pi(s) \geq v_{\pi'}(s)$ для всіх $s \in S$. Завжди існує принаймні одна стратегія, яка є кращою або рівною всім інших стратегіям. Вона називається ***оптимальною стратегією***. Хоча може бути більше однієї оптимальної стратегії, позначимо всі оптимальні стратегії як π_*. Вони поділяють одну й ту саму функцію цінності стану, яку називають оптимальною функцією цінності стану, що позначається v_*, і визначають як

$$v_*(s) \doteq \max_\pi v_\pi(s), \tag{1.4}$$

для всіх s ∈ S.

Оптимальні стратегії також поділяють одні й ті самі функцію цінності стану та дії, позначеної як q∗ і визначаної як

$$q_*(s,a) \doteq \max_\pi q_\pi(s,a), \tag{1.5}$$

для всіх $s \in S$ та $a \in A(s)$. Для пари стан-дія (s, a), ця функція дає очікувану віддачу після виконання дії a в стані s і таким чином слідуючи оптимальній стратегії. Отже, можемо записати q_* через v_* наступним чином:

$$q_*(s,a) = \mathbb{E}[R_{t+1} + \gamma v_*(S_{t+1}) \mid S_t = s, A_t = a]. \tag{1.6}$$

Оскільки v_* є функцією цінності для стратегії, то вона повинна задовольняти умові самостійності, що задається рівнянням Белмана для значень станів (1.3). Оскільки v_* є оптимальною функцією цінності умова узгодженості може бути записана у спеціальній формі без посилання на будь-яку конкретну стратегію. Це рівняння Белмана для v_*, або ***рівняння оптимальності Белмана***. Інтуїтивно, рівняння оптимальності Белмана виражає той факт, що цінність стану за оптимальною стратегією повинна дорівнювати очікуваній віддачі для найкращої дії у цьому стані:

$$\begin{aligned} v_*(s) &= \max_{a \in A(s)} q_{\pi_*}(s,a) \\ &= \max_a \mathbb{E}_{\pi_*}[G_t \mid S_t = s, A_t = a] \\ &= \max_a \mathbb{E}_{\pi_*}[R_{t+1} + \gamma G_{t+1} \mid S_t = s, A_t = a] \\ &= \max_a \mathbb{E}[R_{t+1} + \gamma v_*(S_{t+1}) \mid S_t = s, A_t = a] \\ &= \max_a \sum_{s',r} p(s', r \mid s, a)\big[r + \gamma v_*(s')\big]. \end{aligned} \tag{1.7}$$

Останні два рівняння є двома формами рівняння оптимальності Белмана для v_*. Рівняння оптимальності Белмана для q_* визначається наступним чином:

$$\begin{aligned} q_*(s,a) &= \mathbb{E}\Big[R_{t+1} + \gamma \max_{a'} q_*(S_{t+1}, a') \;\Big|\; S_t = s, A_t = a\Big] \\ &= \sum_{s',r} p(s', r \,|\, s, a)\Big[r + \gamma \max_{a'} q_*(s', a')\Big]. \end{aligned} \tag{1.8}$$

Діаграми на рис. 1.2 показують графічно найближчий горизонт майбутніх станів і дії, які показані в рівняннях оптимальності Белмана для v_* і q_*. Це ті ж діаграми, що для v_π і q_π, представлені раніше, за винятком того, що дуги були додані за виборами агента, щоб представити, що максимум над цим вибором приймається, а не очікуване значення, яке надається деякою стратегією. Діаграма зліва графічно представляє рівняння оптимальності Белмана (1.7), а діаграма праворуч графічно представляє рівняння (1.8).

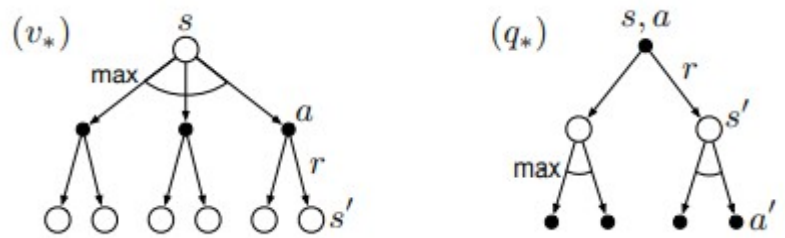

Рисунок. 1.2 - Діаграми рівняння оптимальності Белмана

Важливою складовою систем навчання з підкріпленням є модель навколишнього середовища. Це те, що імітує поведінку середовища або, загалом, дозволяє зробити висновки про те, як буде поводитися середовище. Наприклад, з урахуванням стану і дії, модель може передбачити наступний стан і наступну винагороду. Моделі

використовуються для планування, під яким ми розуміємо будь-який спосіб прийняття рішення про хід дій, розглядаючи можливі майбутні ситуації до того, як вони дійсно виникнуть. Для планування зазвичай використовувється динамічне програмування. Методи розв'язання проблем навчання з підкріпленням, які використовують моделі та планування, називаються методами, заснованими на моделях, на відміну від більш простих методів без моделей, які вчаться шляхом спроб та помилок, і які розглядаються як протилежність планування. Загальноприйняті методи без моделі - це q-навчання та градієнт стратегії. У нашій роботі ми зосереджуємося на методах градієнта стратегії.

1.5 Апроксимація функції стратегії

Системи навчання з підкріпленням повинні бути здатні узагальнюватися, якщо вони будуть застосовані до штучного інтелекту або до великих інженерних задач. Щоб досягти цього, будь-який з широкого спектру існуючих методів апроксимації функцій із області навчання з учителем можна використовувати просто розглядаючи кожне оновлення як навчальний приклад.

Наближена функція цінності представляється як параметризована функція з ваговим вектором w $\in$ R^d. Будемо писати $v(s,w) \approx v_\pi(s)$ для наближеного значення заданого вектора стану w. Наприклад, v може бути лінійною функцією ознак стану, де w - вектор ваг ознак. Загалом, v може бути функцією, обчисленою багатошаровою штучною нейронною мережею, з w - вектором ваг зв'язків у всіх шарах. Регулюючи ваги, мережа може реалізовувати будь-який з широкого спектру різних функцій. Або v може бути функцією, що обчислюється деревом рішень, де w - всі

числа, що визначають точки поділу і значення листів дерева. Як правило, кількість ваг (розмірність *w*) набагато менше числа станів (d << |S|), а зміна однієї ваги призводить до зміни цінності багатьох станів. Отже, коли єдиний стан оновлюється, зміна узагальнює вплив цього стану на значення багатьох інших станів. Таке узагальнення робить навчання потенційно більш потужним, але й потенційно складнішим для керування та розуміння.

1.5.1. Нейронна мережа

Штучні нейронні мережі (НМ) широко використовуються для апроксимації нелінійних функцій. НМ - мережа взаємопов'язаних одиниць, що мають деякі властивості нейронів, які є головним компонентом нервових систем. НМ мають довгу історію, з останніми досягненнями у навчанні багатошарових НМ, що відповідають за деякі з найбільш вражаючих можливостей систем машинного навчання, включаючи системи навчання з підкріпленням.

На рисунку 1.5 наведено НМ прямого поширення, що означає, що в мережі немає петель, тобто не існує жодних шляхів у мережі, за допомогою яких вихідні дані нейрона можуть впливати на його вхідні дані. Мережа на малюнку має вихідний шар, що складається з двох вихідних нейронів, вхідного шару з чотирма блоками введення і двох прихованих шарів: шарів, які не є ні вхідними, ні вихідними. Вага з дійсним числовим значенням пов'язана з кожним з’єднанням. Вага приблизно відповідає ефективності синаптичного з'єднання в реальній нейронній мережі. Якщо НМ має щонайменше один цикл у своїх зв'язках,

то вона називається рекурентною. Як НМ прямого поширення, так і рекурентні НМ використовуються для навчання з підкріпленням.

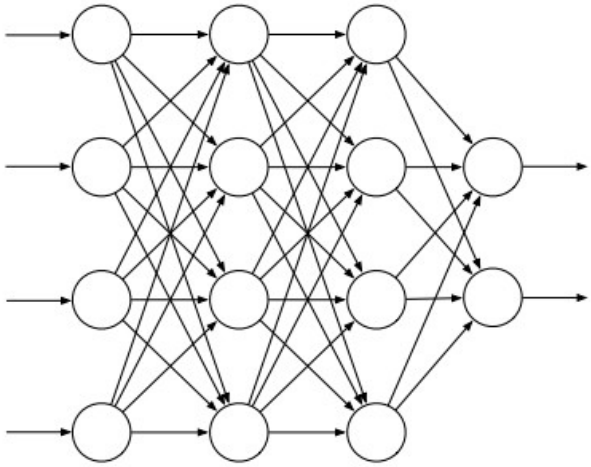

Рисунок 1.5 - НМ прямого поширення

Нейрони (кола на рис. 1.5), як правило, є напівлінійними, тобто вони обчислюють зважену суму своїх вхідних сигналів і потім застосовують до результату нелінійну функцію, яку називають функцією активації, для отримання виходу або активації нейрону. Використовуються різні функції активації, але вони, як правило, S-подібні, або сигмовидні, такі як логістична функція $f(x)=1/(1+e^{-x})$, хоча іноді використовується випрямляч (англ. ReLU) $f(x) = \max(0, x)$. Часто для нейронів різних шарів корисно використовувати різні ***функції активації***.

Активація кожного вихідного блока НМ прямого поширення є нелінійною функцією активацій вхідних блоків мережі. Функції параметризовані за вагами з'єднаннь мережі. НМ без прихованих шарів може представляти лише дуже невелику частину можливих функцій вводу-виводу. Однак НМ з єдиним прихованим шаром, що містить достатньо велике скінчене число сигмоїдних блоків, може апроксимувати

будь-яку неперервну функцію на компактній області вхідного простору мережі до будь-якого ступеня точності [25]. Це також справедливо і для інших нелінійних функцій активації, які задовольняють умови, але нелінійність є суттєвою: якщо всі блоки багатошарової НМ мають лінійні функції активації, то вся мережа еквівалентна мережі без прихованих шарів (тому що лінійні функції лінийних функцій є також лінійними).

Незважаючи на таку властивість «універсальної апроксимації» НМ з одним прихованим шаром, як досвід, так і теорія показують, що апроксимація складних функцій, необхідних для багатьох завдань штучного інтелекту, вимагає абстракцій, які є ієрархічними композиціями багатьох шарів нижчого рівня абстракції, тобто абстракції, що можна отримати за допомогою глибоких архітектур НМ, тобто НМ з багатьма прихованими шарами. Послідовні шари глибокої НМ обчислюють все більш абстрактні уявлення про «необроблений» вхід мережі, кожен з яких забезпечує ознаки, що сприяють ієрархічному поданню загальної функції вводу-виводу мережі.

Отже, тренування прихованих шарів НМ є способом автоматичного створення функцій, відповідних даній проблемі, так щоб ієрархічні подання могли бути створені, не покладаючись виключно на вручну створені ознаки. НМ зазвичай вивчають методом стохастичного градієнта. Кожна вага регулюється в напрямку, спрямованому на поліпшення загальної продуктивності мережі, що вимірюється цільовою функцією, яка буде або мінімізована, або максимізована. У найпоширенішому випадку навчання з учителем, цільова функція - це очікувана помилка, або втрата, над набором позначених прикладів навчання. При навчанні з підкріпленням НМ можуть використовувати TD помилки для вивчення функцій цінності, або вони можуть прагнути максимізувати очікувану

винагороду, як у алгоритмі градієнта стратегії. У всіх цих випадках необхідно оцінити, як зміна кожної ваги з'єднання вплине на загальну продуктивність мережі, іншими словами, оцінити часткову похідну цільової функції по відношенню до кожної ваги, враховуючи поточні значення всіх ваг мережі. ***Градієнт*** - це вектор цих часткових похідних.

Найбільш успішним способом зробити це для НМ з прихованими шарами (за умови, що блоки мають диференційовані функції активації) є алгоритм зворотного поширення помилки, який складається з чергування проходу вперед і назад через мережу. Кожен прохід вперед обчислює активацію кожного блоку з урахуванням поточних активацій вхідних блоків мережі. Після кожного проходу вперед, прохід назад обчислює часткову похідну для кожної ваги. (Як і в інших стохастичних градієнтних алгоритмах навчання, вектор цих часткових похідних є оцінкою справжнього градієнта.) Алгоритм зворотного поширення помилки може давати хороші результати для невиликих мереж, що мають 1 або 2 прихованих шару, але він може не працювати добре для більш глибоких НМ .

Насправді, навчання мережі з k + 1 прихованими шарами може призвести до погіршення продуктивності, ніж навчання мережі з k прихованими шарами, навіть якщо більш глибока мережа може представляти всі функції, які може містити більш дрібні мережі [26]. Виникають дві важливі проблеми. По-перше, велика кількість ваг в типовомій глибокій НМ ускладнює уникнення проблеми перенавчання, тобто, неможливість мережі узагальнюватися на випадки, які не зустрічалися при тренуванні. По-друге, зворотне поширення помилки не працює добре для глибоких НМ, тому що часткові похідні, обчислені зворотними проходами, швидко зменшуються доходячи до вхідної

сторони мережі, що робить навчання з глибокими шарами надзвичайно повільним, або часткові похідні швидко зростають, що робить навчання нестійким. Методи боротьби з цими проблемами багато в чому обумовлені багатьма вражаючими результатами, досягнутими системами, що використовують глибокі НМ.

Перенавчання є проблемою для будь-якого методу апроксимації функцій, який регулює функції з багатьма ступенями свободи на основі обмежених даних навчання. Це не така проблема для навчання з підкріпленням, яке не залежить від обмежених навчальних наборів, але узагальнення є дуже важливим питанням. Перенавчання є проблемою для НМ взагалі, але особливо для глибоких НМ, оскільки вони мають дуже велику кількість ваг. Для зменшення перенавчання розроблено багато методів. До них відносяться припинення тренування, коли продуктивність починає зменшуватися на перевірочній виборці (відмінній від навчальної виборки), зменення цільову функцію, що перешкоджає складності апроксимації (регуляризації), і введення залежності між вагами для зменшення кількості ступенів свободи (наприклад, розподіл ваги).

Пакетна нормалізація [20] є іншою методикою, яка полегшує навчання глибоких НМ. Відомо, що тренування НМ легше, якщо мережевий вхід нормалізується, наприклад, шляхом регулювання кожної вхідної змінної таким чином, щоб вона мала нульове середнє значення і одиничне відхилення. Пакетна нормалізація для тренування глибоких НМ нормалізує вихід глибинних шарів перед подачею в наступний шар. Статистичні дані з підмножин, або "міні-пакетів", навчальних прикладів використовуються для нормалізації міжшарових сигналів для підвищення швидкості навчання глибоких НМ. Зокрема, для і-го підсумкового входу в

l-й шар метод пакетної нормалізації зменшує підсумовані входи відповідно до їх відхилень у розподілі даних.

$$\bar{a}_i^l = \frac{g_i^l}{\sigma_i^l}\left(a_i^l - \mu_i^l\right) \qquad \mu_i^l = \mathop{\mathbb{E}}_{\mathbf{x}\sim P(\mathbf{x})}\left[a_i^l\right] \qquad \sigma_i^l = \sqrt{\mathop{\mathbb{E}}_{\mathbf{x}\sim P(\mathbf{x})}\left[\left(a_i^l - \mu_i^l\right)^2\right]} \quad (1.13)$$

де a^l_i - нормалізовані просумовані входи до і -го прихованого блоку в l-й шар і g_i - параметр підсилення, що масштабує нормалізовану активацію перед нелінійною функцією активації. Зверніть увагу, що очікування поширюється на весь розподіл даних навчання. Як правило, недоцільно обчислювати очікування у формулі (1.13) точно, оскільки для цього потрібно було б пройтись через весь набір даних з поточним набором ваг. Замість цього μ і σ оцінюються з використанням емпіричних зразків з поточного міні-пакету. Це ставить обмеження на розмір міні-партії, і ускладнює застосування методу до рекурентних нейронних мереж.

Метод нормалізації шарів [21] призначений для подолання недоліків пакетної нормалізації. Зверніть увагу на те, що зміни у виході одного шару схильні викликати сильно корельовані зміни у підсумованих входах до наступного шару, особливо з блоками ReLU, виходи яких можуть сильно змінюватися. Це говорить про те, що проблема «коваріантного зсуву» може бути зменшена шляхом фіксації середнього значення і дисперсії підсумкових входів у кожному шарі. Таким чином, ми обчислюємо статистику нормалізації шару по всіх прихованих нейронах в одному шарі наступним чином:

$$\mu^l = \frac{1}{H}\sum_{i=1}^{H} a_i^l \qquad \sigma^l = \sqrt{\frac{1}{H}\sum_{i=1}^{H}\left(a_i^l - \mu^l\right)^2} \quad (1.14)$$

де H позначає кількість прихованих нейронів у шарі. Різниця між рівнянням (1.13) і (1.14) полягає в тому, що при нормалізацією шару всі

приховані одиниці шару мають однакові нормалізаційні значення μ і σ, але різні випадки навчання мають різні вирази нормування. На відміну від пакетної нормалізації, нормалізація шарів не накладає жодних обмежень на розмір міні-пакету і може бути використана в чистому онлайн-режимі з розміром пакету 1.

1.5.2. Згорткова нейронна мережа

Типом глибоких НМ, які виявилися дуже успішними на практиці, включаючи значні застосування для навчання з підкріпленням, є глибока ***згорткова нейронна мережа*** (ЗНМ). Цей тип мережі спеціалізується на обробці багатомірних даних, розташованих у просторових масивах, таких як зображення. Натхненням для ЗНМ стали принципи візуальної обробки у мозку [27]. Через свою особливу архітектуру глибоку ЗНМ можна навчити за допомогою зворотного поширення помилки.

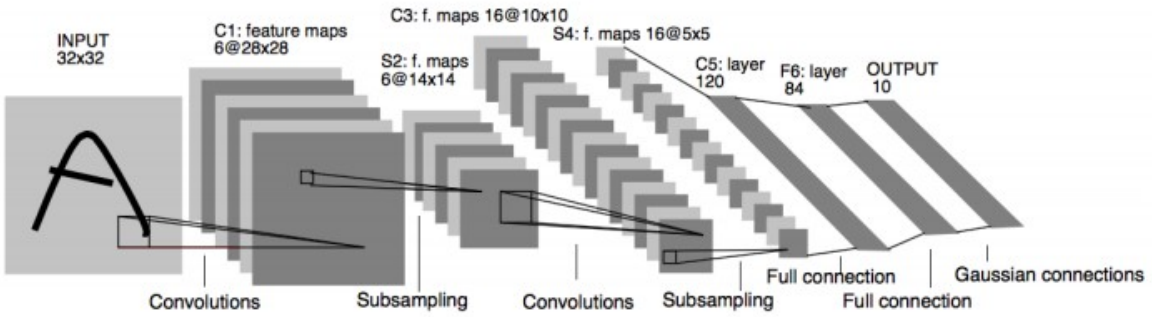

Рис 1.6 - Глибока згорткова нейронна мережа

Рис. 1.6 ілюструє архітектуру глибокої ЗНМ. Цей приклад, від [27] був розроблений для розпізнавання рукописних символів. Вона складається зі згорткових шарів, за якими слідують кілька повнозв'язних кінцевих шарів. Кожен згортковий шар видає ряд карт властивостей. Карта ознак є шаблоном активності над масивом блоків, де кожен блок виконує

ту ж саму операцію над даними у своєму рецептивному полі, яке є частиною даних, які цей блок "бачить" з попереднього шару (або з зовнішнього входу у випадку першого згорткового шару). Блоки карти ознак ідентичні один одному, за винятком того, що їх рецептивні поля, які мають однаковий розмір і форму, зміщені на різні місця на масивах вхідних даних. Блоки на тій самій карті поділяють однакові ваги. Це означає, що карта ознак виявляє ті самі ознаки незалежно від того, де вона знаходиться у вхідному масиві. У мережі на рис. 1.6, наприклад, перший згортковий шар виробляє 6 карт, кожна з яких складається з 28 × 28 блоків. Кожен блок в кожній карті ознак має рецептивне поле розміру 5 × 5, і ці рецептивні поля перекриваються (в даному випадку чотирма стовпцями і чотирма рядками). Отже, кожна з 6 карт ознак визначається лише 25 вагами, які регулюються. Успіхи в розробці ЗНМ роблять великий вклад у навчання з підкріпленням.

1.6. Методи градієнту стратегії

1.6.1. Теорема градієнту стратегії

Градієнт стратегії [8] - це метод, який навчається параметризованій стратегії, яка може вибирати дії, не звертаючись до функції цінності. Функція цінності може все ще використовуватися для налаштування параметрів стратегії, але вона не потрібна для вибору дій. Використаємо позначення $\theta \in R^d$ для вектора параметрів стратегії. Таким чином, ми пишемо $\pi(a|s,\theta)=Pr(A_t=a|S_t=s,\theta_t=\theta)$ для ймовірності того, що дія *a* виконується в момент часу *t,* враховуючи, що середовище знаходиться в стані *s* в момент часу *t* з параметром θ. Якщо метод

використовує вивчену функцію цінності, то ваговий вектор функції цінності позначається w $\in R^d$ як у *v (s, w)*.

Ми розглядаємо методи для знаходження параметра стратегії на основі градієнта деякого показника продуктивності J(θ) щодо параметра стратегії. Ці методи прагнуть максимізувати продуктивність, яка представлена як наближення градієнтного підйому в J:

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha \widehat{\nabla J(\boldsymbol{\theta}_t)}, \tag{1.15}$$

де ∇J(θt) є стохастичною оцінкою, очікування якої апроксимує градієнт продуктивності по відношенню до її аргументу θ_t.

Для оцінки градієнта продуктивності відносно параметрів стратегії, коли градієнт залежить від невідомого ефекту змін стратегії на розподіл стану, ми використовуємо теорему градієнта стратегії. Вона дає нам аналітичне вираження для градієнта продуктивності відносно параметрів стратегії (те, що нам потрібно апроксимувати для градієнтного підйому (1.15)), що не пов'язане з похідною розподілу стану. ***Теорема градієнта стратегії*** встановлює, що:

$$\nabla J(\boldsymbol{\theta}) \propto \sum_s \mu(s) \sum_a q_\pi(s,a) \nabla_{\boldsymbol{\theta}} \pi(a|s,\boldsymbol{\theta}), \tag{1.16}$$

де градієнти - вектори стовпців часткових похідних по компонентах θ, а π - стратегія, що відповідає вектору параметрів θ. Символ ∝ тут означає “пропорційний”. У випадку епізодичних середовищ постійна пропорційності є середньою довжиною епізоду, а в випадку неперервних середовищ вона дорівнює 1, так що співвідношення фактично є рівнянням.

1.6.2. Параметризація стратегії для неперервних дій

Методи на основі градієнту стратегії пропонують практичні способи боротьби з великими розмірностями множини дій, навіть неперервними множинами з нескінченною кількістю дій. Замість того, щоб обчислювати вивчені ймовірності для кожної з багатьох дій, ми будемо налаштовувати статистику розподілу ймовірностей. Наприклад, набір дій може бути дійсними числами, з діями, вибраними з нормального розподілу (розподілу Гауса). Функція густини ймовірності для нормального розподілу умовно записується як:

$$p(x) \doteq \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right), \tag{1.17}$$

де μ та σ — це середнє значення та стандартне відхилення нормального розподілу.

Значення p(x) — це щільність ймовірності у точці x. Воно може бути більше 1; сумарна площа під p(x) має в суммі давати 1. Загалом, можна взяти інтеграл під p (x) для будь-якого діапазону значень x, щоб отримати ймовірність x, що потрапляє в цей діапазон.

Щоб отримати параметризацію стратегії, стратегія може бути визначена як густина нормального розподілу над дійсною скалярною дією, з середнім значенням і стандартним відхиленням, заданими параметричними апроксиматорами функцій, які залежать від стану:

$$\pi(a|s,\boldsymbol{\theta}) \doteq \frac{1}{\sigma(s,\boldsymbol{\theta})\sqrt{2\pi}} \exp\left(-\frac{(a-\mu(s,\boldsymbol{\theta}))^2}{2\sigma(s,\boldsymbol{\theta})^2}\right), \tag{1.18}$$

де $\mu : S \times R^d \to R$ та $\sigma : S \times R^d \to R^+$ - два параметризовані апроксиматора фукнії.

1.6.3. Методи актора та критика

Методи, які навчаються апроксимації як стратегії, так і функції цінностей, часто називають методами актора та критика, де «актор» означає вивчену стратегію, а «критик» - вивчену функцію цінності, зазвичай функцію цінності станів та дій.

1.7 Обгрунтування теми дипломної роботи

1.7.1. Постановка задачі

Оскільки методи навчання з підкріпленням є дуже загальними, вони можуть використовуватися для багатьох задач керування без модифікації алгоритму (з можливим налаштування гіперпараметрів).

В цій роботі ми сфокусуємося на задачі захоплення об’єктів, використовуючи для цього промислову роботичну руку. Моделювання задачі потребує також створення комп’ютерної симуляції, яка є близькою до реального середовища, адже головною метою є використання даних методів на реальних роботах у майбутньому. Незважаючи на те, що компьютерна симуляція дозволяє отримувати інформацію про розташування і форму об’єктів (що полегшує навчання), у реальному світі цю інформація так просто недоступна. Тому ми використовуємо віртуальну камеру розташовану над столом та направлену на об’єкти для того, щоб агент розумів яким чином він впливає на середовище.

Агент має бути взмозі піднімати об’єкти різних розмірів та форм (об’єкти підібрані таким чином, що в теорії їх можна підняти). Більш того,

він має бути взмозі піднімати об’єкти, яких він не бачив при навчанні, тобто бути взмозі узагальнюватися.

1.7.2. Класичні методи вирішення поставленої задачі

Для вирішення задач керування у класичних методах зазвичай використовується теорія оптимального керування. ***Теорія оптимального керування*** розглядає проблему знаходження закону керування для даної системи таким чином, що досягається певний критерій оптимальності. Проблема керування включає функціональну вартість, яка є функцією змінних стану та керування. Оптимальне керування — це набір диференціальних рівнянь, що описують контури управляючих змінних, які мінімізують функцію витрат.

Проблеми оптимального керування, як правило, є нелінійними і тому не мають аналітичних рішень (наприклад, подібно до задачі лінійно-квадратичного оптимального керування). Як результат, необхідно використовувати чисельні методи для вирішення задач оптимального керування

Щоб вирішити проблеми керування у робототехніці, можна використовувати ***зворотну кінематику***. У робототехніці зворотна кінематика використовує кінематичні рівняння для визначення параметрів з’єднань, які забезпечують бажане положення для кожного з кінцевих ефекторів робота. Специфікація руху робота таким чином, щоб її кінцеві ефектори досягли бажаних завдань, відома як планування руху. Інверсна кінематика перетворює план руху в траєкторії моторів з’єднань для робота.

Кінематичний аналіз є одним з перших кроків у розробці більшості промислових роботів. Кінематичний аналіз дозволяє конструктору отримати інформацію про положення кожного компонента в межах механічної системи. Ця інформація необхідна для подальшого динамічного аналізу разом з шляхами керування.

Зворотна кінематика є прикладом кінематичного аналізу обмеженої системи твердих тіл, або кінематичного ланцюга. Кінематичні рівняння робота можуть бути використані для визначення рівнянь циклу складної з'єднаної системи. Ці циклічні рівняння є нелінійними обмеженнями на конфігураційні параметри системи. Незалежні параметри в цих рівняннях відомі як ступені свободи системи.

Існує багато методів моделювання та розв'язання задач зворотної кінематики. Найбільш гнучкі з цих методів зазвичай покладаються на ітераційну оптимізацію для пошуку наближеного рішення, через труднощі інвертування рівняння прямої кінематики і можливості порожнього простору рішень. Основна ідея деяких з цих методів полягає в моделюванні рівняння прямої кінематики, використовуючи розширення ряду Тейлора, яке може бути простіше інвертувати і вирішити, ніж вихідна система.

Використання камери для керування роботом потребує додаткових зусиль по локалізації об'єктів на робочій поверхні та визначення їх розмірів.

1.7.3. Переваги методів навчання з підкріпленням

Як вже зазначалося, методи навчання з підкріпленням, можуть виконувати керування у випадку неперервної множини станів та дій, що

необхідно для задач робототехніки. Для цього можна використовувати нейронну мережу в якості апроксиматора стратегії або функції цінності.

Великою перевагою використання нейронної мережі для подання стратегії є те, що ми можемо подавати зображення в якості стану системи, замість явної інформації про розташування об’єктів. Використання згорткових нейронних мереж дає нам можливість отримувати високорівневі ознаки з зображень, тобто інформацію про об’єкти, яку агент може використовувати для покращення стратегії. При цьому немає необхідності робити окремий модуль для розпізнавання та локалізації об’єктів на зображенні: нейронна мережа буде одночасно і розпізнавати об’єкти, і керувати агентом.

Також винятковою особливістю нейронних мереж в якості апроксиматорів є здатність узагальнюватися до нових прикладів (тих що не були представлені у навчальній виборці). Таким чином, після навчання на певному наборі об’єктів, агент зможе також успішно працювати з новими об’єктами.

2. МЕТОД ГРАДІЄНТУ ГЛИБОКОЇ ДЕТЕРМІНОВАНОЇ СТРАТЕГІЇ

2.1. Алгоритм градієнту детермінованої стратегії

Алгоритм градієнту глибокої детермінованої стратегії (ГГДС) [3] є покращеннням алгоритму градієнту детермінованої стратегії (ГДС). ***Алгоритм градієнту детермінованої стратегії*** [4] використовує параметризовану функцію актора $\mu(s|\theta^{\mu})$, яка визначає поточну стратегію шляхом детермінованого відображення станів на конкретну дію. Критик Q(s,a) оновлюється за допомогою рівняння Белмана. Актор оновлюється подальшим застосуванням ланцюгового правила до очікуваного повернення від початкового розподілу J відносно параметрів актора

$$\begin{aligned}\nabla_{\theta^{\mu}} J &\approx \mathbb{E}_{s_t \sim \rho^{\beta}}\left[\nabla_{\theta^{\mu}} Q(s, a|\theta^{Q})|_{s=s_t, a=\mu(s_t|\theta^{\mu})}\right] \\ &= \mathbb{E}_{s_t \sim \rho^{\beta}}\left[\nabla_{a} Q(s, a|\theta^{Q})|_{s=s_t, a=\mu(s_t)} \nabla_{\theta_{\mu}} \mu(s|\theta^{\mu})|_{s=s_t}\right]\end{aligned} \tag{2.1}$$

З практичної точки зору існує принципова різниця між стохастичними та детермінованими градієнтами стратегії. У стохастичному випадку градієнт стратегії інтегрується як у просторі станів, так і в просторі дій, тоді як в детермінованому випадку він інтегрується тільки над простором стану. Як результат, обчислення градієнта стохастичної стратегії може вимагати більше зразків, особливо якщо простір дій є багатомірним. Для вивчення повного простору станів і простору дій часто необхідна стохастична стратегія. Щоб гарантувати, що алгоритм градієнта детермінованої стратегії продовжує тренуватися задовільно, використовується алгоритм навчання поза стратегією. Основна ідея полягає у виборі дій відповідно до стохастичної стратегії (для забезпечення адекватного дослідження), але оновлювати

детерміновану стратегію (користуючись ефективністю детермінованого градієнта стратегії).

2.2. Використання нейронних мереж

Однією з проблем при використанні нейронних мереж у навчанні з підкріпленням є те, що більшість алгоритмів оптимізації припускають, що спостереження незалежно і однаково розподілені. Очевидно, що коли спостереження генеруються при тренуванні послідовно з середовища, це припущення більше не виконується. Крім того, для ефективного використання оптимізації апаратних засобів необхідно тренувати використовуючи міні-пакети, а не онлайн.

Як і в алгоритмі глибоких Q-мереж [5], будемо використовувати ***буфер спостержень*** для вирішення цих питань. Буфер спотережень - кеш обмеженого розміру R. Спостеження отримуються з середовища згідно зі стратегією дослідження, а кортеж (s_t, a_t, r_t, s_{t+1}) зберігається в буфері спостережень. Коли буфер спостережень заповняється, старі спостреження відкидаються. Після кожного кроку актор і критик оновлюються шляхом вибірки міні-пакету рівномірно з буфера. Оскільки ГГДС є алгоритмом поза стратегією, буфер спостережень може бути великим, що дозволяє алгоритму використовувати навчання на наборі некорельованих спостережень.

Q-навчання з нейронними мережами виявилася нестабільною у багатьох середовищах. Оскільки мережа $Q(s, a| \theta^Q)$, що оновлюється, також використовується при обчисленні цільового значення оновлення Q схильне до розходження. Запропоноване рішення схоже на ***цільову мережу***, яка використовується в [5], але модифікована для архітектури

актор-критик і використовує «м'які» цільові оновлення, а не безпосереднє копіювання ваг. Створюється копія НМ актора та критика, $\mu'(s|\theta^{\mu'})$ і $Q'(s,a|\theta^{Q'})$ відповідно, які використовуються для обчислення цільових значень. Ваги цих цільових мереж потім оновлюються шляхом повільного відстеження основних мереж актора та критика: $\theta' \leftarrow \tau\theta+(1-\tau)\theta'$, де $\tau<<1$. Це означає, що цільові значення повільно змінюються, значно покращуючи стабільність навчання. Ця проста зміна переносить відносно нестабільну проблему вивчення функції цінності дії-значення ближче до випадку навчання з учителем, проблема, для якої існують надійні рішення. Наявність цільової мережі як для актора μ', так і для критика Q' необхідно для тренування без розбіжності. Це може сповільнити навчання, оскільки цільова мережа затримує поширення оцінок цінності. Проте, на практиці стабільність тренування є більш важливою.

2.3. Вирішення проблеми дослідження

Великою проблемою навчання з підкріпленням у випадку неперервного простору дій є проблема дослідження. Перевагою таких алгоритмів, як ГГДС, є те, що ми можемо розглядати проблему дослідження незалежно від алгоритму навчання. Ми використовуємо дослідницьку стратегію μ', додавши шум, відібраний з процесу *N,* до стратегії актора:

$$\mu'(s_t) = \mu(s_t|\theta_t^{\mu}) + \mathcal{N} \qquad (2.2)$$

N може бути обрано відповідно до середовища. Для цього використовується ***процес Орнштейна-Уленбека*** [16], щоб сформувати часово корельовану стратегію дослідження. Процес Орнштейна–Уленбека є стаціонарним процесом Гаусса–Маркова, тобто поєднання гаусівського

процесу і марковського процесу, і він часово однорідним. З часом процес має тенденцію до зміщення до свого довгострокового середнього значення. Процес може розглядатися як модифікація випадкового блукання в безперервному часі.

На рис 2.1 представлений псевдокод алгоритму градієнту глибокої детермінованої стратегії:

Algorithm 1 DDPG algorithm

Randomly initialize critic network $Q(s, a|\theta^Q)$ and actor $\mu(s|\theta^\mu)$ with weights θ^Q and θ^μ.
Initialize target network Q' and μ' with weights $\theta^{Q'} \leftarrow \theta^Q$, $\theta^{\mu'} \leftarrow \theta^\mu$
Initialize replay buffer R
for episode = 1, M **do**
  Initialize a random process $\mathcal{N}$ for action exploration
  Receive initial observation state s_1
  for t = 1, T **do**
    Select action $a_t = \mu(s_t|\theta^\mu) + \mathcal{N}_t$ according to the current policy and exploration noise
    Execute action a_t and observe reward r_t and observe new state s_{t+1}
    Store transition (s_t, a_t, r_t, s_{t+1}) in R
    Sample a random minibatch of N transitions (s_i, a_i, r_i, s_{i+1}) from R
    Set $y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'})$
    Update critic by minimizing the loss: $L = \frac{1}{N}\sum_i (y_i - Q(s_i, a_i|\theta^Q))^2$
    Update the actor policy using the sampled policy gradient:

$$\nabla_{\theta^\mu} J \approx \frac{1}{N} \sum_i \nabla_a Q(s, a|\theta^Q)|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^\mu} \mu(s|\theta^\mu)|_{s_i}$$

    Update the target networks:

$$\theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'}$$
$$\theta^{\mu'} \leftarrow \tau\theta^\mu + (1-\tau)\theta^{\mu'}$$

  end for
end for

Рис 2.1 - Псевдокод алгоритму

3. ОПИС ПРОГРАМНИХ ЗАСОБІВ

Оскільки алгоритми навчанння з підкріпленням є дуже загальними, тестування алгоритму ГГДС здійснювалося на спрощеному середовищі, очікуючи, що він також буде надійно працювати при переході на основне середовище (середовище з промисловою роботиною рукою). Єдина зміна, яка була необхідна при переході зі спрощеного до основного середовища, це архітектура нейронних мереж актора та критика, оскільки простір станів є різним для цих двух задач (детальніше буде описано нижче).

3.1. Опис спрощеного середовища

В якості спрощеного середовища, ми використали середовище Lunar Lander з бібліотеки OpenAI Gym (скріншот середовища можна побачити на рис. 3.1). В цьому середовищі агенту необхідно посадити віртуальний космічний корабель на обидві опори якомога повільніше, щоб нічого не пошкодити. При цьому керування здійснюється заданням двох дійсних чисел, які відповідають за дії.

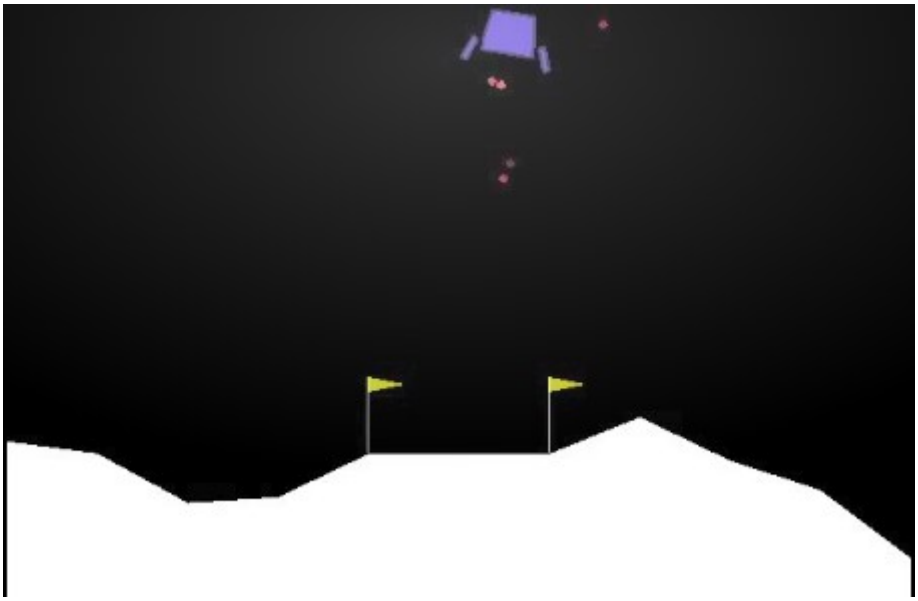

Рисунок 3.1. - Зображення спрощеного середовища

Перше значення керує головним двигуном: від -1 до 0 означає, що двигун вимкнений, від 0 до 1 відподає від 50% до 100% потужності. Двигун не може працювати з потужністю менше 50%. Друге значення в діапазоні від -1.0 до - 0.5 залучає лівий двигун, від +0.5 до +1.0 залучає правий двигун, -0.5 до 0.5 не залучає бокових двигунів.

Посадочний майданчик завжди знаходиться на координатах (0,0). Координати корабля - це перші два числа у векторі стану. Нагорода за перехід від вершини екрану до посадкового майданчику і нульової швидкості становить близько 100..140 одиниць. Якщо корабель віддаляється від посадкового майданчику, він втрачає винагороду. Епізод закінчується, якщо апарат припиняє роботу або сідає, виключаючи двигуни, отримуючи додаткові -100 або +100 одиниць. Кожний контакт опорою із землею — додаткові +10 одиниць. Користування основним двигуном коштує -0,3 одиниці за кожен крок. Середовище вважається вирішеним, якщо агент зможе набрати більше 200 одиниць винагороди. Можлива посадка за межами посадкового майданчику. Паливо нескінченне, тому агент може навчитися літати, а потім приземлитися з першої спроби.

Таким чином множина станів є вектором з 8 дійсних чисел, а множина дій — вектором з 2 дійсних чисел.

3.2. Опис основного середовища

Для моделювання задачі захоплення об’єктів промисловою роботичною рукою було взято комп’ютерну симуляцію, що була створена за допомогою фізичного симулятора з відкритим вихідним кодом Bullet Physics. Цей симулятор дозволяє наближено відворювати фізику реальних

об’єктів, використовуючи їх 3D моделі та фізичні властивості. Також у даному симуляторі підримується візуалізація симуляції в окремому вікні. В якості середовища для симуляції нашої задачі було обрано середовище KukaDiverseObjectGrasping, яке в точності відповідає поставленій задачі.

Загальна кількість типів об’єктів — 100. Об’єкти розділені випадковим чином на навчальну та тестову виборку у співвідношенні 90/10 відповідно. Об’єкти обиралися таким чином, щоб у теорії роботична рука була взмозі їх схопити та підняти.

Як вже зазначалося, в якості стану, будемо використовувати кольорове зображення з розширенням 84 на 84, отримане з віртуальної камери. На рисунку 3.2 можна побачити зображення середовища. У лівому верхньому куту відображено типове зображення, що подається в якості стану агента:

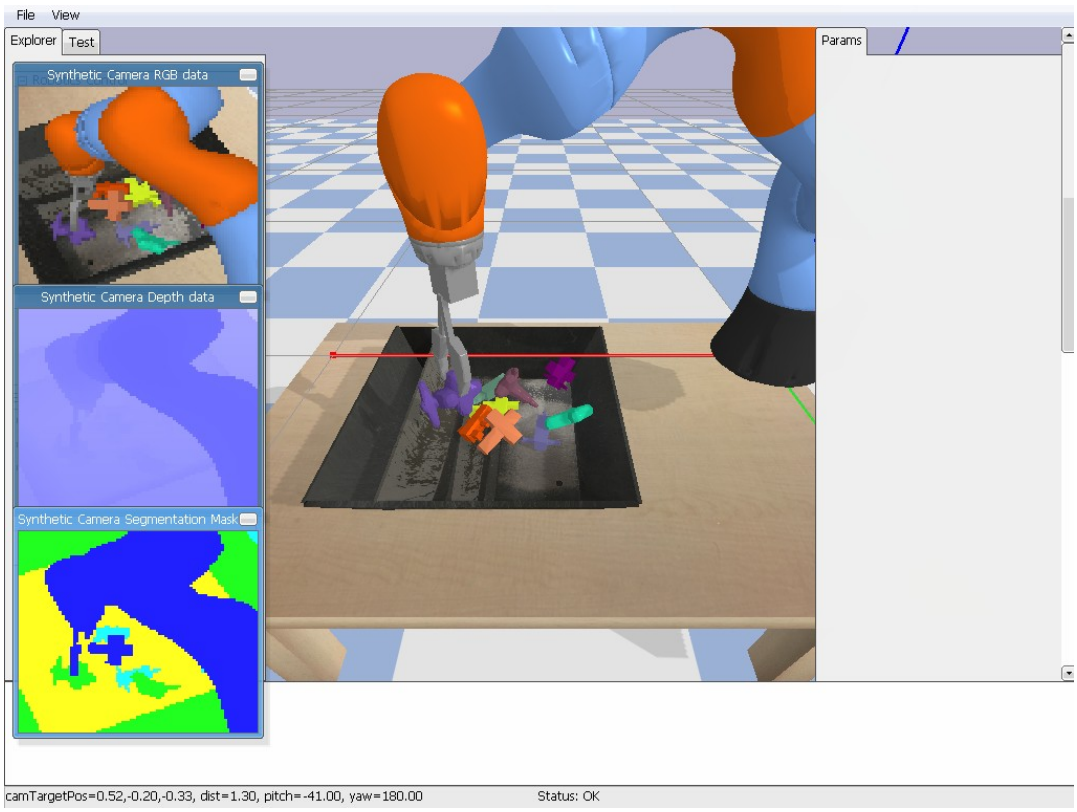

Рисунок 3.2 - Зображення промислового робота у комп’ютерній симуляції

Простір дій складається з 4 дійсних чисел у діапазоні [-1, 1]. Перші три значення задають зміщення захоплючої частини роботичної руки відносно її поточного положення. Останнє значення задає наскільки треба повернути захоплюючу частину в площині паралельній землі. Нове рішення приймається кожні 80 кроків симуляції, а в проміжних кроках повторюється попереднє. Один крок симуляції еквівалентний 4 мілісекундам. Максимальна довжина епізоду — 1200 кроків (приблизно 5 секунд).

3.3. Програмне середовище та компоненти розробки

Програма була написана з використанням мови програмування Python. Комп'ютерна симуляція поставленої задачі була здійснена засобами фізичного симулятора Bullet Physics.

В процесі розробки були використані наступні бібліотеки:

- **OpenAI Gym** — інструментарій для розробки та порівняння алгоритмів навчання з підкріпленням.
- **Numpy** — бібліотека, призначена для швидкого виконання різних математичних операцій з векторами та матрицями.
- **PyTorch** — бібліотека, призначена для задач машинного навчання
- **Pybullet** — Python інтерфейс для фізичного симулятора Bullet Physics

3.4. Опис основної програми

Програма складається з наступних файлів:

- **main.py** — зчитує конфігураційний файл, створює середовище, передає конфігурацію та створене середовище агенту, запускає навчання
- **ddpg.py** — містить основний алгоритм, виконує взаємодію із середовищем, тренування стратегії, оцінку стратегії
- **replay_memory.py** — містить визначення класу ReplayMemory, який представляє собою буфер спостережень, з якого можна вибирати випадковий міні-пакет спостережень під час тренування
- **log.py** — дозволяє запусувати програмні логи як на екран, так і у файл для подальшого аналізу
- **stats.py** — дозволяє записувати корисну статистику під час навчання для подальшого аналізу

4. АНАЛІЗ ОТРИМАНИХ РЕЗУЛЬТАТІВ

4.1. Архітектура нейронних мереж та гіперпараметри для спрощеного середовища

На рисунку 4.1 представлена архітектура нейронної мережі для актора алгоритму ГГДС для спрощеного середовища. На вхід отримуємо стан агента (представлений вектором дійсних чисел) і після обробки набором шарів нейронної мережі нам повертається вектор дії. На рисунку використовувалися наступні позначення:

- **ПЗ** — повнозв'язний шар НМ
- **НШ** — нормалізація шару
- **ReLU** — активаційна функція випрямляч
- **tanh** — активаційна функція гіперболічного тантенсу

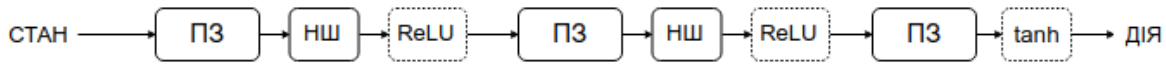

Рисунок 4.1 - Архітектура нейронної мережі актора для спрощеного середовища

На рисунку 4.2 представлена архітектура нейронної мережі для критика алгоритму ГГДС для спрощеного середовища. На вхід отримуємо стан і дію агента і після обробки набором шарів нейронної мережі нам повертається значення функції цінності стану та дії. Позначення такі ж, як

і на попередньому рисунку. Коло зі знаком плюс всередині означає операцію конкатенації.

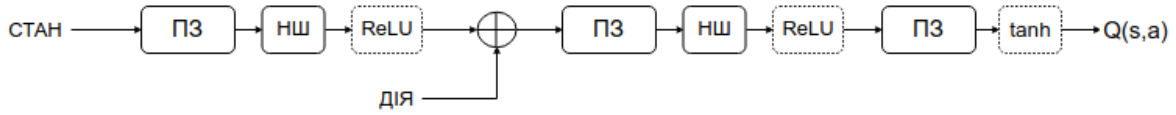

Рисунок 4.2 - Архітектура нейронної мережі критика для спрощеного середовища

Гіперепараметри алгоритму ГГДС для спрощеного середовища представлені на таблиці 4.1:

Назва гіперпараметру	**Значення**
Частота оновлення НМ	2
Випадкове початкове значення	1
Коефіцієнт знецінення	0.99
Розмір повнозв’язних шарів	128
Швидкість навчання актора	0.0001
Швидкість навчання критика	0.001
Розмір буфера спостережень	50000
Розмір міні-пакету	128
Коефіцієнт зміни цільових НМ	0.001

Таблиця 4.1 - Гіперпараметри для спрощеного середовища

4.2. Аналіз результатів для спрощеного середовища

На рисунку 4.3 представлений графік залежності середньої винагороди від кількості пройдених кроків у середовищі. Як бачимо після приблизно 150 тисяч кроків агент навчився успішно виконувати поставлену задачу (як було зазначено в описі середовища, задача вважається вирішеною, якщо винагорода є більшою за 200 одиниць). На початку навчання агент здійснює випадкові дії згідно процесу Орнштейна-Уленбека, тому винарода є дуже маленькою, проте з часом стратегія стає все більш детермінованою і агент починає справлятися із поставленої задачею.

Рисунок 4.3 — Графік залежності середньої винагороди від кількості пройдених кроків у спрощеному середовищі

Функція витрат для актора монотонно спадає (окрім початку, коли агент виконує повністю стохастичні дії), як можна побачити на рисунку 4.4. Метою градієнтних методів оптимізації є мінімазація функції витрат, що ми і бачимо на графіку.

Рисунок 4.4 — Графік залежності значення функції витрат актора від кількості пройдених кроків у спрощеному середовищі

На рисунку 4.5 можна побачити графік залежності значення функції витрат критика. від кількості пройдених кроків у спрощеному середовищі. Функція витрат для критика є менш передбачуваною, оскільки в ході навчання оцінювана цінність станів та дій постійно змінюється в залежності від того, як добре в агента виходить виконання завдань.

Рисунок 4.5 — Графік залежності значення функції витрат критика від кількості пройдених кроків у спрощеному середовищі

4.3. Архітектура нейронних мереж та гіперпараметри для основного середовища

Основне середовище потребує дещо іншої архітектури нейронної мережі, адже в якості стану ми використовуємо зображення. Тому на вході ми маємо три додаткових згорткових шари, які, на концептуальному рівні, знаходять у зображені високорівневі ознаки (такі як об’єкти, наприклад).

На рисунку 4.6 представлена архітектура нейронної мережі для актора алгоритму ГГДС для основного середовища. На вхід ми отримуємо стан агента (представлений кольоровим зображенням, отриманим з

віртуальної камери) і після обробки набором шарів нейронної мережі нам повертається вектор дії. На рисунку використовувалися наступні позначення:

- **ЗГ (N, K, S)** — згортковий шар, де N — кількість фільтрів, K — розмір фільтрів, S — крок операції згортки
- **ПЗ** — повнозв'язний шар НМ
- **НШ** — нормалізація шару
- **ReLU** — активаційна функція випрямляч
- **tanh** — активаційна функція гіперболічного тантенсу

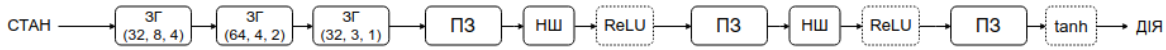

Рисунок 4.6 - Архітектура нейронної мережі актора для основного середовища

На рисунку 4.7 представлена архітектура нейронної мережі для критика алгоритму ГГДС для основного середовища. На вхід ми отримуємо стан і дію агента і після обробки набором шарів нейронної мережі нам повертається значення функції цінності стану та дії. Позначення такі ж, як і на попередньому рисунку. Коло зі знаком плюс всередині означає операцію конкатенації.

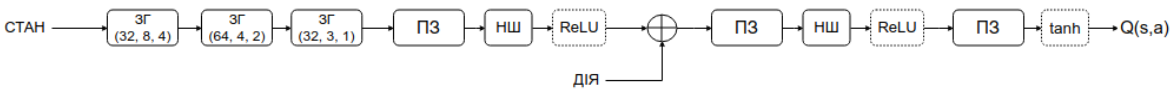

Рисунок 4.7 - Архітектура нейронної мережі критика для основного середовища

Гіперепараметри алгоритму ГГДС для основного середовища представлені на таблиці 4.2:

Назва гіперпараметру	Значення
Частота оновлення НМ	1
Випадкове початкове значення	1
Коефіцієнт знецінення	0.99
Розмір повнозв’язних шарів	300
Швидкість навчання актора	0.001
Швидкість навчання критика	0.001
Розмір буфера спостережень	100000
Розмір міні-пакету	100
Коефіцієнт зміни цільових НМ	0.005

Таблиця 4.2. Гіперпараметри для основного середовища

4.4. Аналіз результатів для основного середовища

Як вже зазначалось, навчання агента проводилось з використанням навчальної та тестувальної вибірки предметів. Загальна кількість типів предметів — 100, таким чином для навчальної вибірки використовувалося 90 випадково обраних типів предметів, а для тестувальної — 10 з тих, що

залишились. Під час тренування випадковим чином обиралося 10 предметів, які було розкидано на віртуальному столі.

Під час навчання кожні 50 тисяч кроків виконувалася оцінювання продуктивності агента на навчальній та тестовій виборці, виконуючи додаткові експерименти 100 разів, та обчислюючи відсоток успішних експериментів. На рисунку 4.8 можна побачити результати навчання агента з використанням алгоритму ГГДС.

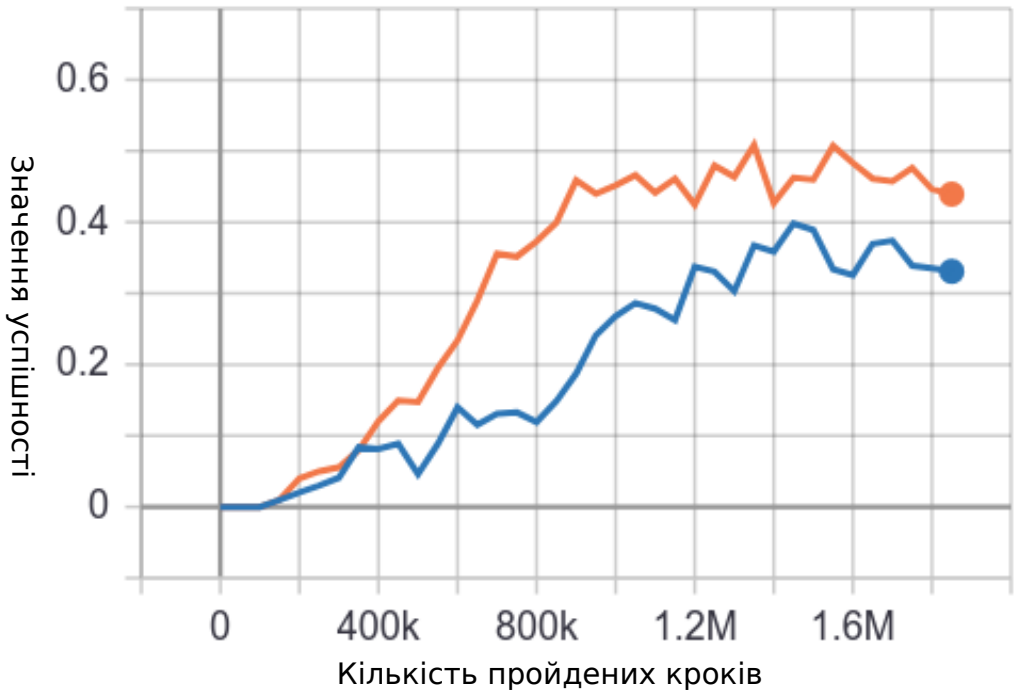

Рисунок 4.8 - Графік залежності значення успішності від кількості пройдених кроків в основному середовищі

Таким чином, на навчальній виборці було досягнуто відсоток успішності приблизно 50%, а на тестовій приблизно 40%. Ці дані свідчать про те, що алгоритм ГГДС з використанням нейронних мереж має високу здатність до узагальнення, оскільки різниця між значенням успішності на навчальній і тестовій виборці є невеликою. Проте досягнуті значення успішності є досить невеликими і потребують подальшого покращення.

Недоліком алгоритмів з підкріпленням є неефективність використання даних, адже алгоритм потребує величезної кількості спостережень для того, щоб навчитися надійно виконувати поставлену задачу. Алгоритми, що використовують буфер спостережень дещо допомагають справлятися з цією проблемою, адже застосовуючи буфер ми більш ефективно використовуємо інформацію з попередніх кроків. Ця проблема особливо важлива, у випадку якщо навчання проводиться на реальних роботах.

Перенос стратегії натренованої на комп'ютерній симуляції на реальних роботів є можливим, проте потребує дуже високу якість симуляції, що зазвичай дуже нелегко досягнути, та може бути дуже ресурсовитратним.

Також для прискорення навчання та покращення результатів можна використовувати поступове ускладнення задачі в ході навчання (наприклад підбирати складність предметів згідно їх форми в нашій задачі). Це дозволило б агенту швидко навчитися виконувати задачу на легких задачах і після цього узагальнювати отриману інформацію на більш складні випадки.

Ще одним методом прискорення навчання є використання експертної інформації, тобто використання знання про те, як задача вирішується людиною. Для цього створюється набір демонстрацій, в яких людина (експерт) виконує завдання, і агент після цього намається відтворити ці дії. Таким чином, зменшується кількість дослідницьких кроків агента, адже він вже матиме приблизне уявлення про те, як треба вирішувати поставлену задачу.

Симуляція та навчання нейронних мереж потребує великої кількості обчислювальних ресурсів та займає тривалий проміжок часу. Для зменшення часу тренування та більш ефективного використання наявних обчислювальних ресурсів можна застосовувати навчання з використанням багатьох агентів, які виконують одне і те ж завдання (кожен у своєму середовищі) і одночасно збирають спостереження, та синхронно або асинхронно оновлюють ваги НМ.

ВИСНОВКИ

Дипломна робота присвячена розв’язанню задачі розробки комп’ютерної системи управління промисловим роботом. В процесі розв’язання отримано наступні результати:

1. Показано, що перспективним шляхом керування роботами нового покоління є використання методів штучного інтелекту, а саме методів навчання з підкріпленням.
2. Реалізовано алгоритм градієнту глибокої детермінованої стратегії для задачі неперервного керування та простестовано його коректність на спрощеному середовищі
3. Запропоновано архітектури нейронних мережі актора та критика алгоритму ГГДС для спрощеного та основного середовища
4. Проведено експериментальні дослідження на середовищі комп’ютерної симуляції поставленої задачі, в ході яких було перевірено здатність алгоритму навчитися виконувати завдання на навчальній виборці об’єктів та узагальнюватися на тестову виборку.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Sutton, R. and Barto, A. (2018). Reinforcement Learning: An Introduction, Second edition. MIT Press.
2. Dmitry Kalashnikov, Alex Irpan, Peter Pastor, Julian Ibarz, Alexander Herzog, Eric Jang, Deirdre Quillen, Ethan Holly, Mrinal Kalakrishnan, Vincent Vanhoucke, and Sergey Levine. Qt-opt: Scalable deep reinforcement learning for vision-based robotic manipulation. In Conference on Robot Learning, 2018
3. T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, “Continuous control with deep reinforcement learning,” in International Conference on Learning Representations (ICLR), 2016.
4. Silver, David, Lever, Guy, Heess, Nicolas, Degris, Thomas, Wierstra, Daan, and Riedmiller, Martin. Deterministic policy gradient algorithms. In ICML, 2014
5. V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, “Playing Atari with deep reinforcement learning,” in NIPS Deep Learning Workshop, 2013.
6. D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis, “Mastering the game of Go with deep neural networks and tree search,” Nature, vol. 529, pp. 484–489, 2016.

7. P. Dhariwal, C. Hesse, O. Klimov, A. Nichol, M. Plappert, A. Radford, J. Schulman, S. Sidor, and Y. Wu, “OpenAI baselines,” https://github.com/openai/baselines, 2017.

8. Sutton, R. S., Mcallester, D., Singh, S., and Mansour, Y. (2000). Policy gradient methods for reinforcement learning with function approximation. In NIPS’1999 , pages 1057—1063. MIT Press

9. Goodfellow, I., Bengio, Y., and Courville, A. (2016). Deep Learning. MIT Press

10. LeCun, Y., Kavukcuoglu, K., and Farabet, C. (2010). Convolutional networks andapplications in vision. In Circuits and Systems (ISCAS), Proceedings of 2010 IEEEInternational Symposium on, pages 253–256. IEEE

11. S. R. Buss, “Introduction to inverse kinematics with jacobian transpose, pseudoinverse and damped least squares methods,” IEEE Journal of Robotics and Automation, vol. 17, no. 1-19, p. 16, 2004.

12. Kingma, Diederik and Ba, Jimmy. Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980, 2014.

13. Koutnık, Jan, Schmidhuber, Jurgen, and Gomez, Faustino. Evolving deep unsupervised convolu- ¨ tional networks for vision-based reinforcement learning. In Proceedings of the 2014 conference on Genetic and evolutionary computation, pp. 541–548. ACM, 2014a

14. Krizhevsky, Alex, Sutskever, Ilya, and Hinton, Geoffrey E. Imagenet classification with deep convolutional neural networks. In Advances in neural information processing systems, pp. 1097–1105, 2012.

15. Levine, Sergey, Finn, Chelsea, Darrell, Trevor, and Abbeel, Pieter. End-to-end training of deep visuomotor policies. arXiv preprint arXiv:1504.00702, 2015.

16. Uhlenbeck, George E and Ornstein, Leonard S. On the theory of the brownian motion. Physical review, 36(5):823, 1930.

17. J. M. McCarthy, 1990, Introduction to Theoretical Kinematics, MIT Press, Cambridge, MA.

18. Bertsekas, D. P. (1995). Dynamic Programming and Optimal Control. Belmont: Athena. ISBN 1-886529-11-6.

19. Bellman, R. (1957). "A Markovian Decision Process". Journal of Mathematics and Mechanics

20. Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. ICML, 2015.

21. Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. 2016. Layer normalization. arXiv preprint arXiv:1607.06450.

22. Z. Zhou, X. Li, and R. N. Zare. Optimizing Chemical Reactions with Deep Reinforcement Learning. ACSCentral Science3, 2017.

23. H.Mao, Alizadeh, M. Alizadeh, Menache, I.Menache, and S.Kandula. Resource Management With deep Reinforcement Learning. In ACM Workshop on Hot Topics in Networks, 2016.

24. G. Zheng, F. Zhang, Z. Zheng, Y. Xiang, Ni. J. Yuan, X. Xie, and Z. Li. DRN: A Deep Reinforcement Learning Frameworkfor News Recommendation. 2018.

25. Cybenko, G. (1989). Approximation by superpositions of a sigmoidal function. Mathematics of control, signals and systems, 2(4):303–314.

26. Bengio, Y. (2009). Learning deep architectures for AI. Foundations and Trends in Machine Learning, 2(1):1–27.

27. LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. Proceedings of the IEEE, 86(11):2278–2324.