Одеський національний університет імені І. І. Мечникова
Факультет математики, фізики та інформаційних технологій
Кафедра оптимального керування та економічної кібернетики

Кваліфікаційна робота

на здобуття ступеня вищої освіти «магістр»

«Розв'язування задач структурно-параметричної оптимізації методами машинного навчання»

«Solving the problems of structural and parametric optimization by machine learning methods»

Виконала: здобувачка денної форми навчання
спеціальності 113 Прикладна математика
Освітня програма «Прикладна математика»
Ткачова Таїсія Максимівна

Керівник: канд. фіз.-мат. наук, доц. Страхов Є.М.
Рецензент: канд. техн. наук, доц. Мороз В.В.

Рекомендовано до захисту:

Протокол засідання кафедри

№ ___ від _____ 2023 р.

Завідувач кафедри

_____ _____

Захищено на засіданні ЕК № _____

Протокол № ___ від _____ 2023 р.

Оцінка _____ / _____ / _____

Голова ЕК

_____ _____

Одеса — 2023 р.

ЗМІСТ

ВСТУП

Актуальність теми. Розв'язування задач структурно-параметричної оптимізації має вирішальне значення в процесах інженерного проєктування. Оскільки дослідження в цій галузі мають потенціал для створення кращих підходів до моделювання та вирішення задач, які забезпечуватимуть вищу ефективність та рентабельність.

Дослідження останніх років демонструють, що оптимізація є фундаментальним аспектом проєктування структур і систем у різних галузях, таких як цивільне будівництво, аерокосмічна інженерія, машинобудування тощо. Водночас традиційні методи оптимізації можуть вимагати багато часу та витрат на обчислення при вирішенні складних математичних систем керування. В свою чергу методи машинного навчання можуть допомогти прискорити процес оптимізації та потенційно знайти кращі рішення.

Практична значущість пошуку рішень в задачах оптимального керування за допомогою методів машинного навчання полягає в їх здатності забезпечувати адаптивні, ефективні та інноваційні рішення для широкого спектру застосувань, сприяючи вдосконаленню стратегій керування в складних системах.

Мета і задачі дослідження. *Метою роботи* є вивчення та дослідження підходів машинного навчання в контексті вирішення завдань оптимального керування.

Предметом дослідження є застосування методів машинного навчання для вирішення складнощів, пов'язаних із розв'язком задачі оптимального керування.

Об'єктом дослідження виступає задача оптимального керування про м'яку посадку космічного корабля на поверхню Місяця.

Методи дослідження: для досягнення поставленої мети, проводяться дослідження з побудови та оптимізації нейронних мереж за допомогою фреймворку PyTorch.

В рамках дипломної роботи було розглянуто та досліджено застосування нейронних мереж, реалізовних на основі фреймворку PyTorch, у

вирішенні здачі оптимального керування. Ключовою частиною дослідження став вибір задачі оптимального керування: задача о м'якії посадці космічного корабля на поверхню Місяця. Особлива увага приділялася моделюванні системи таким чином, щоб нейронна мережа могла ефективно розв'язувати її та з певною точністю передбачати значення двох параметрів оптимального керування.

Результати були порівняні та проаналізовані шляхом обчислення відстані Фреше між реальними та передбаченими траєкторіями. Отримані результати підкреслюють потенційні переваги використання машинного навчання в області оптимального керування та його можливості в розв'язанні складних систем керування.

РОЗДІЛ 1

ТЕОРЕТИЧНА ЧАСТИНА

1.1 Оптимальне керування

Оптимальне керування — це процес визначення закону керування або управляючу послідовності дій для заданого процесу чи об'єкта керування протягом певного періоду часу з метою мінімізації відповідної сукупності критеріїв якості системи.

Оптимальне керування та його відгалуження знайшли застосування в різноманітних сферах, включаючи аерокосмічну галузь, робототехніку, біоінженерію, управління процесами та науку про управління тощо. Через значні обчислювальні обмеження до появи цифрового комп'ютера в 1950-х роках можна було розв'язувати лише досить прості задачі оптимального керування. Поява такої потужної цифрової машини дозволила застосувати теорію та методи оптимального керування до багатьох складних проблем.

Для розв'язання задачі оптимального керування будується математична модель процесу чи об'єкта керування, що демонструє, як його поведінка змінюється з часом через зміну власного стану та керуючих факторів. Математична модель задачі оптимального керування включає в себе наступне: встановлення критерію якості управління для визначення мети керування; формулювання диференціальних або різницевих рівнянь, які описують потенційні напрямки руху керованого об'єкта; розробка рівнянь або нерівностей, що описують обмеження на використовувані ресурси. Під час розробки систем управління найбільш поширеними методами є динамічне програмування Беллмана, принцип максимуму Понтрягіна та варіаційне числення. [1, 2]

1.1.1 Загальна постановка задачі оптимального керування

Розглянемо загальну математичну постановку задачі оптимального керування. Передусім визначимо керований об'єкт, еволюція якого описується системою диференціальних рівнянь:

$$\frac{dx}{dt} = \dot{x} = f(x(t); u(t); t);$$

де:

- $x(t) = (x_1(t); : : : ; x_n(t)) \; 2 \; \mathbb{R}^n$ - вектор стану системи (або ж фазовий вектор);
- $u \; 2 \; U \quad \mathbb{R}^m$ - вектор керування, що приймає свої значення в деякій замкненій підмножині з класу кусково-неперервних функцій;
- $t \; 2 \; \mathbb{R}$ - час.

В залежності від конкретних цілей задачі цільова функція може мати різний вигляд. Зазвичай це функціонал виду:

$$J = \int_{t_0}^{t_f} L(x(t); u(t); t)\, dt + \; '(x(t_f); t_f);$$

де:

- $L : \mathbb{R}^n \quad \mathbb{R}^m \quad \mathbb{R} \; ?! \; \mathbb{R}$ - проміжна функція витрат;
- $' : \mathbb{R}^n \quad \mathbb{R} \; ?! \; \mathbb{R}$ - функція кінцевих витрат;
- t_0, t_f - початковий і кінцевий моменти часу відповідно.

Метою задачі оптимального керування є знаходження такої функції керування $u(t)$, яка задовольняє деяким обмеженням і мінімізує певний функціонал. Вигляд обмежень і функціоналів визначається умовами реальних задач. [3]

1.1.2 Задача про космічний корабель

Існує безліч варіацій постановки задачі про м'яку посадку космічного корабля на поверхню Місяця, це зумовлено складністю математичної моделі

системи. Врахування різноманітних факторів, таких як технічні характеристики апарату, змінність стану атмосфери, нерівномірність поверхні Місяця, обмеження на керування, призводить до різних математичних моделей. Розгляд різноманітних траєкторій, методів стабілізації та управління, а також врахування нечіткості чи невизначеності в системі також сприяє виникненню розмаїттю формулювань задачі. [4–8]

Розглянемо систему звичайних диференціальних рівнянь, що моделює динаміку простого космічного корабля в двовимірному просторі, де маса змінюватиметься точковою масою. Отже, маємо:

$$\dot{r} = v;$$

$$\dot{v} = c_1 \frac{u_1}{m} \hat{l} + g;$$

$$\dot{m} = -\frac{c_1}{c_2} u_1;$$

де:

- $r = (x; y)$ - вектор положення в двовимірному просторі;
- $v = (v_x; v_y)$ - відповідний вектор швидкості;
- m - маса космічного корабля;
- $g = (0; -g_m)$ - прискорення обумовлене гравітацією Місяця $g_m = 1{:}62$ (м/с^2);
- c_1 - константа, що визначає максимальне значення сили тяги (Н);
- $c_2 = I_{sp} g_e$ - константа, що задає ефективність ракетного двигуна через його питомий імпульс I_{sp} (Н), де $g_e = 9{:}81$ (м/с^2) позначає прискорення земного тяжіння;
- $u_1 \in [0; 1]$, $\in \left[-\frac{\pi}{2}; \frac{\pi}{2}\right]$ - керуючі змінні системи, що моделють амплітуду та напрямок тяги вздовж напрямку $\hat{l} = [\sin \; ; \cos \;]$.

Зауваження 1.1. Для поточної задачі параметр керування u_1 виступає показником перемикання тяги, так що при значені $u_1 = 0$ двигун вимкнен, а $u_1 = 1$ вказує на роботу двигуна з допустимою величиною тяги.

Зауваження 1.2. Параметр розглядається як другий параметр керування u_2, оскільки ми припускаємо, що ми можемо самостійно керувати кроком космічного корабля, так як модель не містить обертальної інерції.

Тепер опишимо функціонал, що будемо мінімізувати, він же і задаватиме функцію витрат:

$$J = (1 \quad) \int_0^{t_f} w\frac{c_1^2}{c_2}u_1^2 dt + \int_0^{t_f} \frac{c_1}{c_2}u_1 dt;$$

де:

- $w = 1$ - коефіцієнт компромісу, який коригує функцію витрат між двома внесками;
- 2 $[0;1]$ - параметр, що визначає тип задачі; при значеннях $= 1$ ми маємо справу з задачею оптимального керування паливом, а при значеннях $= 0$ ми маємо справу з квадратичною задачею оптимального керування.

Слідуючи методу Понтрягіну, розглянемо таку функцію Гамільтона:

$$H = \ _r v + \ _v \ c_1\frac{u_1}{m}\hat{\imath} \ + g \quad \ _m\frac{c_1}{c_2}u_1 + (1 \quad) w\frac{c_1^2}{c_2}u_1^2 + \ \frac{c_1}{c_2}u_1;$$

де введено функції спільного стану $_r(t)$, $_v(t)$ та $_m(t)$. Згідно з принципом максимуму, напрямок тяги має бути протилежним до $_v$, щоб функція Гамільтона була мінімізована, отже, маємо:

$$\hat{\imath} = \ -\frac{v}{v};$$

де $_v = k \ _v k_2$.

Якщо $\not{6}\ 1$, оптимальне керування u_1 має наступний кінцевий вираз:

$$u_1 = \min \ \max \ \left[\frac{\frac{_v c_2}{m} + \ _m}{2wc_1(1 \quad)}; 0 \right]; 1 \ ;$$

і, якщо $= 1$:

$$u_1 = \begin{cases} 0; & S < 0; \\ 1; & S > 0; \end{cases}$$

де $S = \frac{_v c_2}{m} + \ _m$ - функція перемикання для випадку задачі оптималь-

ного управління масою.

У такий спосіб отримаємо спряжену систему:

$$\dot{r} = 0;$$

$$\dot{v} = r;$$

$$\dot{m} = \frac{c_1}{m} v\hat{\imath} u:$$

Зрештою, виходить така двоточкова крайова задача:

$$\underline{r} = v;$$

$$\underline{v} = c_1 \frac{u_1}{m} \hat{\imath} + g;$$

$$\underline{m} = \frac{c_1}{c_2} u_1;$$

$$\dot{m} = \frac{c_1}{m^2} u_1 \quad v_{x_0} \quad x_0 t \ \sin \ + \quad v_{y_0} \quad y_0 t \ \cos \ ;$$

з граничними умовами r_0, v_0, m_0 при $t = 0$ та r_t, v_t, m_t при $t = t_f$.

1.2 Машинне навчання

У 1959 році Артур Самуель охарактеризував машинне навчання як "область дослідження, яка дає комп'ютерам можливість навчатися без явного програмування". Іншими словами, основна мета машинного навчання полягає у створенні таких комп'ютерних програм, які можуть автоматично вдосконалюватися з часом.

Машинне навчання вивчає різні алгоритми та статистичні моделі, які дозволяють комп'ютерним системам виконувати завдання без явного програмування. Однією з ключових переваг цього підходу є те, що алгоритми можуть навчатися на основі великої кількості даних та адаптуватися до змін в середовищі.

Машинне навчання зазвичай має справу з трьома типами задач, а саме: класифікація, регресія та кластеризація. Залежно від типу задачі можна

визначити відповідний алгоритм. Алгоритмом машинного навчання називають метод за допомогою якого система штучного інтелекту передбачає вихідні значення з заданого вхідного набору даних.

У цій роботі ми зосередимось на використанні штучних нейронних мереж для розв'язку поставленої задачі оптимального керування.

1.2.1 Нейронні мережі

У сучасному науковому та технічному дискурсі штучні нейронні мережі (ШНМ) широко застосовуються для моделювання різноманітних аспектів людської діяльності у різних галузях [9–11]. Таке активне застосування ШНМ обумовлене тим, що традиційні методи обчислень, що базуються на попередньо визначених правилах та рівняннях, можуть виявитися недостатньо ефективними чи непридатними у випадках, де правила є невідомими або змінюються в ході вирішення проблеми. Однією з визначальних характеристик ШНМ є їх здатність вчитися на досвіді та прикладах, а потім адаптуватися до мінливих ситуацій. Саме тому нейронні мережі є дуже корисними інструментами для застосування в областях, де точність традиційних методів є перешкодою. Особливо це стосується програм, які вимагають розпізнавання образів або моделювання фізичних систем [12].

Використання нейронних мереж у задачах оптимального керування представляє собою перспективний підхід, який виокремлюється із традиційних методів оптимізації. Нейронні мережі можуть ефективно моделювати складні та нелінійні залежності в системах керування, що дозволяє досягти оптимальних рішень в умовах великої невизначеності та змінних умов оточення [13–15]. При використанні нейронних мереж у задачах оптимального керування, модель може адаптуватися до змін в середовищі та взаємодіяти з різноманітними факторами. Це особливо важливо в сферах, де система має складні та непередбачувані динамічні характеристики.

Нейронні мережі також можуть бути використані для розв'язання задач оптимального керування в реальному часі, забезпечуючи більш гнучке та адаптивне управління. Їхню здатність до самонавчання та адаптації робить їх потужним інструментом у вирішенні складних задач управління

в різних галузях, таких як автоматизовані системи, робототехніка та інші області, де важливо досягати оптимальних рішень в умовах невизначеності.

Алгоритми нейронної мережі для машинного навчання натхненні архітектурою та динамікою мереж нейронів у людському мозку. Хоча ці алгоритми використовують дуже ідеалізовані моделі нейронів, проте фундаментальний принцип їхньої дії лишається незмінним: штучні нейронні мережі навчаються шляхом модифікації взаємозв'язків між їхніми нейронами. Такі мережі здатні виконувати безліч завдань з обробки інформації.

Архітектура нейронної мережі включає у себе три компоненти, відомі як шари (рис. 1.1):

1) Вхідний шар (input layer): цей рівень відповідає за отримання інформації (даних), сигналів, функцій або вимірювань. Кількість нейронів у цьому шарі визначається розміром вхідних даних.
2) Приховані шари (hidden layers): це один або кілька шарів нейронів, які знаходяться між вхідним та вихідним шарами. Кожен нейрон у прихованому шарі пов'язаний з кожним нейроном попереднього та наступного шарів.
3) Вихідний шар (output layer): це останній шар нейронів, який відповідає за створення та представлення кінцевих мережевих виходів, які є результатом обробки, виконаної нейронами на попередніх рівнях. Кількість нейронів у вихідному шарі зазвичай визначається кількістю класів (у випадку задачі класифікації) або кількістю вихідних змінних (у випадку регресії).

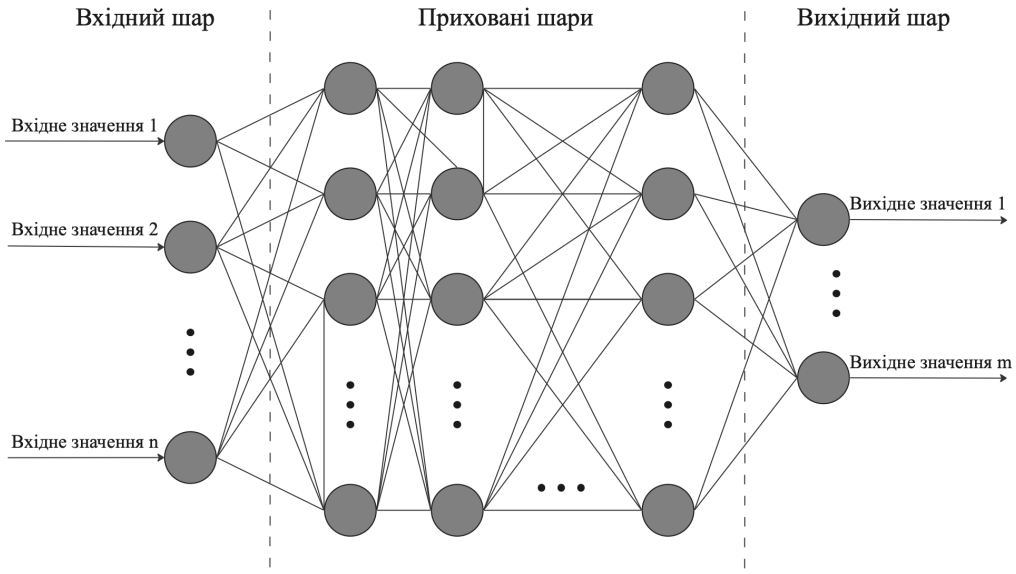

Рис. 1.1. Архітектура нейронної мережі.

У структурі нейронної мережі кожен нейрон зв'язаний з іншими нейронами в сусідніх шарах за допомогою вагових зв'язків. Вага кожного зв'язку визначає його вплив та важливість передачі сигналу між нейронами. Під час навчання ці ваги є об'єктом налаштування для оптимізації ефективності мережі.

Кожен нейрон у структурі мережі використовує функцію активації для розрахунку виходу на основі ваг та вхідних сигналів. Найбільш розповсюдженими функціями активації вважаються сигмоїда, гіперболічний тангенс (tanh) та функція випрямленого лінійного вузла (ReLU).

Метою нейронної мережі є оптимізація деякої функції витрат. Функція втрат (loss function) є мірою помилки та використовується оптимізаційним алгоритмом для корекції ваг та підтримки збалансованого навчання мережі в процесі навчання.

РОЗДІЛ 2

ЕКСПЕРИМЕНТИ ТА РЕЗУЛЬТАТИ

2.1 Опис тренувальних даних

Для поточної задачі використовується набір даних, що містить оптимальні траєкторії. Кожна оптимальна траєкторія складається зі списку ознак $(x; y; v_x; v_y; m)$ та пари цільових змінних $(u_1; u_2)$, де x та y задають вектор положення в двовимірному просторі, v_x та v_y - швидкості по горизонтальній та вертикальній осям відповідно, m - маса космічного корабля, а u_1, u_2 – оптимальні керуючі змінні.

Дані створені синтетично шляхом моделювання траєкторій, які входять до області ініціалізації, наведеної в таблиці 2.1.

x	y	v_x	v_y	m
$[\ 250; 250]$	$[0; 2000]$	$[\ 10; 10]$	$[\ 30; 30]$	$[8000; 12000]$

Табл. 2.1. Область ініціалізації траєкторій.

Модель навчається на 460 згенерованих траєкторій, кожна з яких складається від 90 до 890 значень. Загалом, навчальний набір даних містить 106396 рядків.

2.2 Опис реалізації експерименту

У процесі реалізації ШНМ я широко застосовувала фреймворк PyTorch. PyTorch — це оптимізована тензорна бібліотека з відкритим кодом для машинного та глибинного навчання, що заснована на мові програмування Python [16]. Цей фреймворк дає змогу створювати та активно використовувати нейронні мережі за допомогою пакета torch.nn [17].

Використовуючи вищезгадану бібліотеку моделюється клас ‘Net’, який конструює архітектуру нейронної мережі, використовуючи повністю з’єднані

шари, та призначений для вирішення завдань моделювання і симуляції руху космічного корабля. Також варто відзначити, що метод ‘forward_pass’ класу ‘Net’ визначає проходження сигналу через мережу, із застосуванням функцій активації ReLU та tanh. Це визначення забезпечує отримання вихідних значень мережі для конкретного вхідного сигналу. Крім того, вихідний код містить метод ‘forward_simulate’, який моделює рух об’єкта за допомогою нейронної мережі. Він використовує вихідні значення мережі як керування для обчислення зміни стану об’єкта у просторі. Такий підхід необхідний для прийняття рішень у режимі реального часу.

Нейронні мережі піддаваються навчанню до досягнення стану збіжності в рамках стохастичного градієнтного спуску (stochastic gradient descent), при цьому використовується концепція розміру пакета (batch size), що визначається кількістю тренувальних екземплярів, обраних для одного циклу проходження вперед та назад нейронною мережею, і яка становить 10.

З метою оцінки розриву між оптимальними діями та прогнозами, здійсненими нейронною мережею, використовується середньоквадратичне відхилення (mean squared error). MSE обчислюється шляхом підсумовування різниць між прогнозованими та цільовими значеннями моделі, зведенням результатів у квадрат та подальшим поділом на загальну кількість спостережень в наборі даних. Цей метричний показник сприяє аналізу ефективності нейронної мережі, однак не забезпечує повноцінного вимірювання того, наскільки ефективно виконується завдання посадки.

Основні аспекти програмної реалізації зазначені в розділі Додаток А. Результат виконання першої спроби нейронної мережі (моделі №1) занесені в таблицю 2.3.

2.3 Налаштування гіперпараметрів

Оптимізація моделі є однією з найскладніших проблем у пошуку розв’язання задач машинного навчання. Етапом розв’язку задач машинного навчання, завдяки якому досягається оптимальність моделі називають налаштуванням гіперпараметрів (або оптимізацією гіперпараметрів).

Ãïïåðïàðàìåòðè öå ïàðàìåòðè, ùî âñòàíîâëþþòüñÿ ïåðåä ïî÷àòêîì ïðîöåñó íàâ÷àííÿ òà âèçíà÷àþòü àðõiòåêòóðó ìîäåëi. Ïðåòiêñ ¾ãiãåð¿ ãîâîðèòü ïðî òå, ùî öå ïàðàìåòðè ¾âåðõíüîãî ðiâíÿ¿, òîáòî êîíòðîëþþòü ïðîöåñ íàâ÷àííÿ òà ïàðàìåòðè ìîäåëi, ùî âèïëèâàþòü iç íå¨.

Íàëàøòóâàííÿ ãiïåðïàðàìåòðiâ ¹ âàæëèâîþ ÷àñòèíîþ êåðóâàííÿ ïîâåäiíêè ìîäåëi ìàøèíîãî íàâ÷àííÿ. Öåé ïðîöåñ ïîëÿãà¹ ó çíàõîäæåíi òàêî¨ êîìáiíàöi¨ çíà÷åíü ãiïåðïàðàìåòðiâ, ÿêà íàéêðàùå ìàêñèìiçó¹ ïðîäóêòèâíiñòü ìîäåëi, ìiíiìiçóþ÷è ïîïåðåäíüî âèçíà÷åíó ôóíêöiþ âòðàò.

2.3.1 Ïîøóê ó ñiòöi

Êëàñè÷íèì òà íàéïðîñòiøèì ïiäõîäîì äëÿ íàëàøòóâàííÿ ãiïåðïàðàìåòðiâ ¹ âè÷åðïíèé ïîøóê ó ñiòöi (grid search). Ñiòêà ïîøóêó áàçó¹òüñÿ íà îñíîâi äåêàðòîâîãî äîáóòêó óñiõ ìîæëèâèõ êîìáiíàöié ãiïåðïàðàìåòðiâ. Ïîòiì áóäóþòüñÿ ìîäåëi äëÿ öèõ êîìáiíàöié, i âèáèðàþòüñÿ íàéêðàùi ãiïåðïàðàìåòðè çàâäÿêè ìåòîäèêàì ïåðåõðåñíî¨ ïåðåâiðêè. [18]

Ïåðåâàãîþ òàêîãî ïîøóêó ¹ éîãî ðåòåëüíiñòü. Îñêiëüêè îòiíþ¹òüñÿ êîæíà ìîæëèâà êîìáiíàöiÿ ãiïåðïàðàìåòðiâ, ïðîóñòèòè íàéêðàùó ìàéæå íåìîæëèâî. Íåäîëiêîì ¹ òå, ùî ÷àñ îáðîáêè öiëèõ íàáîðiâ ïàðàìåòðiâ ìîæå áóòè âåëè÷åçíèì, à îòæå, êiëüêiñòü ïàðàìåòðiâ äëÿ äîñëiäæåíîÿ ìà¹ ïðàêòè÷íi îáìåæåíîÿ.

2.3.2 Ðåçóëüòàò ïîøóêà ó ñiòöi

Äëÿ ïîêðàùåíîÿ ïåðääáà÷åíîÿ ìîäåëi áóëî îáðàíî íàñòóïíi ãiïåðïàðàìåòðè äëÿ îïòèìiçàöi¨:

1) êiëüêiñòü ïðèõîâàíèõ øàðiâ íåéðîíí¨ ìåðåæi;
2) êiëüêiñòü íåéðîíiâ ó êîæíîìó øàði;
3) êiëüêiñòü åïîõ: åïîõà - öå îäèí öèêë íàâ÷àííÿ, à êiëüêiñòü åïîõ âèçíà÷à¹ ñêiëüêè ðàçiâ âåñü íàáið äàíèõ áóäå âèêîðèñòîâóâàòèñÿ äëÿ íàâ÷àííÿ ìåðåæi.
4) øâèäêiñòü íàâ÷àííÿ (learning rate) - ïàðàìåòð, ùî êîíòðîëþ¹ ðîçìið êðîêiâ, âèêîíàíèõ ïiä ÷àñ ïðîöåñó îïòèìiçàöi¨. Âií âèçíà÷à¹, íàñêiëüêè

ïàðàìåòðè ìîäåëi ïîâèííi áóòè ñêîðèãîâàíi ùîäî ãðàäi¹íòà ôóíêöiї âèòðàò.

5) iìïóëüñ (momentum) - ïàðàìåòð, ÿêèé âèêîðèñòîâó¹òüñÿ äëÿ ãàñiííÿ êîëèâàíü i ïðèñêîðåííÿ ñòîõàñòè÷íîãî ãðàäi¹íòíîãî ñïóñêó ó âiäïîâiäíîìó íàïðÿìêó.

Ó òàáëèöi 2.2 ïîäàíi îáðàíi çíà÷åííÿ ãiïåðïàðàìåòðiâ, ñåðåä ÿêèõ ïðîâîäèòèìåòüñÿ ïîøóê ïîòèìàëüíîї êîìáiíàöiї.

Ãiïåðïàðàìåòð	Ìàñèâ çíà÷åíü
Êiëüêiñòü ïðèõîâàíèõ øàðiâ	1; 2; 3
Êiëüêiñòü íåéðîíiâ	16; 32; 64
Êiëüêiñòü åïîõ	5; 10; 15; 20; 25
Øâèäêiñòü íàâ÷àííÿ	0:001; 0:005; 0:01
Iìïóëüñ	0:9; 0:95; 0:99

Òàáë. 2.2. Âèáiðêà çíà÷åíü ãiïåðïàðàìåòðiâ.

Îïòèìàëüíi ãiïåðïàðàìåòðè äëÿ âèáðàíîãî íàáîðó çíà÷åíü (ìîäåëü •2) òà ïîðiâíÿëüíèé àíàëiç ñðåäíüîêâàäðàòè÷íî ïîõèáêè ìiæ ìîäåëÿìè ïðåäñòàâëåíi ó òàáëèöi 2.3.

Ìîäåëü	Ãiïåðïàðàìåòðè	MSE
• 1.	êiëüêiñòü ïðèõîâàíèõ øàðiâ: 3, êiëüêiñòü íåéðîíiâ: 16, êiëüêiñòü åïîõ: 10, øâèäêiñòü íàâ÷àííÿ: 0.005, iìïóëüñ: 0.9	0:000382
• 2.	êiëüêiñòü ïðèõîâàíèõ øàðiâ: 1, êiëüêiñòü íåéðîíiâ: 64, êiëüêiñòü åïîõ: 25, øâèäêiñòü íàâ÷àííÿ: 0.01, iìïóëüñ: 0.99	0:000031

Òàáë. 2.3. Ïîðiâíÿííÿ ìîäåëåé.

Âàæëèâî âiäçíà÷èòè, ùî îïòèìiçàöiÿ ãiïåðïàðàìåòðiâ âíåñëà çíà÷íèé âíåñîê ó ïiäâèùåííÿ åôåêòèâíîñòi ìîäåëi.

2.4 Àíàëiç ðåçóëüòàòiâ

Äëÿ îòðèìàíîÿ êðàùîãî ðîçóìíîÿ òî÷íîñòi ïåðåääà÷åíîÿ ïîòèìàëüíèõ çíà÷åíü êåðóâàííÿ íåéðîííîþ ìåðåæåþ íåîáõiäíî âðàõîâóâàòè íå ëèøå ôóíêöiþ âòðàò ìîäåëi. Öå ïîÿñíþ¹òüñÿ òèì, ùî íàâiòü íåâåëèêi çìiíè â óïðàâëiííi ìîæóòü ïðèçâåñòè äî àáñîëþòíî iíøîãî ðiøåííÿ, òîìó ùî íåçíà÷íi ïîìèëêè ìîæóòü ïîøèðþâàòèñÿ ïî òðà¹êòîði¨, ùî ïðèçâåäå äî íåâäàëèõ àáî íåïîòèàëüíèõ ïîñäîê. Îòæå, ïiñëÿ îòðèìàííÿ îïòèìàëüíèõ óïðàâëiíü íà òåñòîâié âèáiðöi, ìè âiäíîâèìî çíà÷åííÿ òðà¹êòîðié òà ïîðiâíÿ¹ìî êðèâi, ùî âèçíà÷àþòüñÿ òî÷êàìè (x; y).

Ç ìåòîþ îöiíêè ïîäiáíîñòi òðà¹êòîðié ìè âèêîðèñòîâó¹ìî âiäñòàíü Ôðåøå - öå ìiðà ïîäiáíîñòi ìiæ êðèâèìè, ÿêà âðàõîâó¹ ðîçòàøóâàííÿ òà ïîðÿäîê òî÷îê óçäîâæ êðèâèõ [19, 20]. Ïîÿñíþþ÷è êîíöåïöiþ âiäñòàíi Ôðåøå, äîñèòü ÷àñòî âèêîðèñòîâóþòü àíàëîãiþ òîãî, ÿê ãîñïîäàð âèãóëþ¹ ñîáàêó íà ïîâiäöi. Ëþäèíà ðóõà¹òüñÿ ïî ïàíîìó øëÿõó, ñîáàêà ïî iíøîìó, îáèäâà çäàòíi çìiíþâàòè ñâîþ øâèäêiñòü, ïðîòå íå ìîæóòü ðóõàòèñÿ ó çâîðîòíîìó íàïðÿìêó. Âiäñòàíü Ôðåøå, ó öüîìó êîíòåêñòi, âèçíà÷à¹òüñÿ ÿê äîâæèíà íàéêîðîòøîãî ìîæëèâîãî ïîâiäöÿ, äîñòàòíüîãî äëÿ ïðîïóñêàííÿ îáîõ êðèâèõ, i âiäïîâiäíî ñëóãó¹ ìåòðèêîþ ñõîæîñòi ìiæ íèìè.

Îòðèìàíî íàñòóïíi ðåçóëüòàòè íà îäíîì ç îá'¹êòiâ òåñòîâî¨ âèáiðêè (òàáë. 2.4).

Ìîäåëü	MSE	Âiäñòàíü Ôðåøå
• 1.	0:000382	859:0664
• 2.	0:000031	855:78592

Òàáë. 2.4. Òåñò 1. Ïîðiâíÿííÿ çíà÷åííÿ âiäñòàíi Ôðåøå.

Çäiéñíèìî ãðàôi÷íå çîáðàæåííÿ ôàêòè÷íèõ òà ïðîãíîçîâàíèõ òðà¹êòîðié äëÿ äîñÿãíåííÿ áiëüøî¨ íàî÷íîñòi (ðèñ. 2.1, 2.1).

Ðèñ. 2.1. Òåñò 1. Ìîäåëü •1. Ïîðîâíÿííÿ ðåàëüíèõ òà ïåðåäáà÷åíèõ òðà¹-êòîðié.

Ðèñ. 2.2. Òåñò 1. Ìîäåëü •2. Ïîðîâíÿííÿ ðåàëüíèõ òà ïåðåäáà÷åíèõ òðà¹-êòîðié.

Ïîðiâíÿëüíèé àíàëiç ôàêòè÷íèõ òà ïðîãíîçîâàíèõ òðà¹êòîðié, à òàêîæ îá÷èñëåííÿ âiäñòàíi Ôðåøå íà ùå äâåiëüêîõ ïðèêëàäàõ, íàâåäåíî ó ðîçäiëi Äîäàòîê Á.

Õî÷à ìîäåëi ïðîäåìîíñòðóâàëè íåâåëèêi çíà÷åííÿ ñåðåäíüîêâàäðàòè÷íî¨ ïîõèáêè, âiäñòàíü Ôðåøå âèÿâèëàñÿ çíà÷íîþ. Öå âêàçó¹ íà òå, ùî íàâiòü ïðè çíàõîäæåííi òðà¹êòîði¨, â äàíîìó ïðèêëàäi âîíà çäèøà¹òüñÿ íåïîèìàëüíîþ. Óñóíåðå÷ çàçíà÷åíîìó ïîêàçíèêó ìiðè ïîäiáíîñòi äâîõ òðà¹êòîðié, âàðòî ïiäêðåñëèòè, ùî îòðèìàíi ðåçóëüòàòè äåìîíñòðóþòü ïåðñïåêòèâè âèêîðèñòàííÿ ìåòîäè ìàøèííîãî íàâ÷àííÿ ó âèðiøåííi çäàà÷ îïòèìàëüíîãî êåðóâàííÿ. Òîìó ìàéáóòi äîñëiäæåííÿ ìàþòü ñïðÿìîâóâàòèñÿ íà ïîëiïøåííÿ ìîäåëåé òà âðàõóâàííÿ äîäàòêîâèõ ôàêòîðiâ äëÿ äîñÿãíåííÿ áiëüø òî÷íèõ i ïîòìàëüíèõ ðåçóëüòàòiâ.

ÂÈÑÍÎÂÊÈ

Ïiä ÷àñ âèêîíàíïÿ äèïëîìíî ðîáîòè áóëî âèâ÷åíî çàñòîñóâàííÿ íåéðîííèõ ìåðåæ äëÿ âèðiøåíïÿ ïðîáëåìè îïòèìàëüíîãî êåðóâàííÿ ó êîíòåêñòi ì'ÿêî ïîñàäêè êîñìi÷íîãî êîðàáëÿ íà ïîâåðõíi Ìiñÿöÿ.

Ðåçóëüòàòè äîñëiäæåíïÿ ïiäòâåðäæóþòü ïîòåíöiéíi ïåðåâàãè âèêîðèñòàííÿ ìàòiìàiâ ìàøèííîãî íàâ÷àííÿ ó ñôåði îïòèìàëüíîãî êåðóâàííÿ òà ¨õ çäàòíiñòü åôåêòèâíî âèðiøóâàòè ñêëàäíi çàâäàííÿ êåðóâàííÿ. Âîäíî÷àñ àíàëiç âiäñòàíi Ôðåøå ñâiä÷àòü ïðî íåîáõiäíiñòü ïîäàëüøîãî óäîñêîíàëåííÿ ìîäåëi òà óðàõóâàííÿ äîäàòêîâèõ ôàêòîðiâ äëÿ äîñÿãíåííÿ áiëüø òî÷íèõ òà ïîâíiøèõ ðåçóëüòàòiâ. Çîêðåìà, ìîæëèâiñòü ðîçøèðåííÿ îáñÿãó íàâ÷àëüíî¨ âèáiðêè âèçíà÷à¹òüñÿ ÿê îäèí iç ïåðñïåêòèâíèõ àñïåêòiâ. Ïîñëåäíié êiëüêîñòi íàâ÷àëüíèõ ïðèêëàäiâ ìîæå âèÿâèòè çíà÷íèé âïëèâ íà çäàòíiñòü ìîäåi äî áiëüø òî÷íîãî àäàïòóâàííÿ äî ðiçíîìàíiòíèõ ñöåíàði¨â ì'ÿêî¨ ïîñàäêè. Äîäàòêîâî, ðîçãëÿä âåäåííÿ áiëüøîãî ÷èñëà ïàðàìåòðiâ, ùî õàðàêòåðèçóþòü êîñìi÷íèé êîðàáåëü, ïðåäñòàâëÿ¹ ñîáîþ âàæëèâèé àñïåêò äëÿ ïîäàëüøî¨ äåòàëiçàöi¨ ìîäåi äëÿ áiëüø ïîâíîãî âðàõóâàííÿ ôiçè÷íèõ òà òåõi÷íèõ îñîáëèâîñòåé ñèñòåìè. Òàêîæ, óäîñêîíàëåííÿ åôåêòèâíîñòi ìîæå âèìàãàòè óñëàäíåííÿ àðõiòåêòóðè íåéðîííî¨ ìåðåæi. Òàêi íàïðÿìêè ïîäàëüøèõ äîñëiäæåíü ðîçãëÿäàþòüñÿ ÿê ïîòåíöiéíî ïåðñïåêòèâíi äëÿ äîñÿãíåííÿ òî÷íiøèõ ðåçóëüòàòiâ òà ðîçøèðåííÿ ìîæëèâîñòåé çàñòîñóâàííÿ ìåòîäiâ ìàøèííîãî íàâ÷àííÿ ó âèðiøåíi çäää÷ ïîèìàëüíîãî êåðóâàííÿ.

ÑÏÈÑÎÊ ËIÒÅÐÀÒÓÐÈ

1. Bellman R. E. Dynamic programming. New Jersey: Princeton University Press, Sixth Printing, 1972. 342 pp.
2. The Mathematical Teory of Optimal Processes, Division of John Wiley and Sons. / Pontryagin L. S., Boltyanskii V. G., Gamkreidze R. V., Mishchenko E. F. New York, 1962. 362 pp.
3. Lee E. B., Markus L. Foundations of optimal control theory. New York, 1967. 576 pp.
4. Âàðiàöiéíå ÷èñëåíiÿ òà ìåòîäè îïòèìiçàöiï / Ïåðåñòþê Ì. Î., Ñòàíæèöüêèé Î. Ì., Êàïóñòyí Î. Â., Ëîâåéêií Þ. Â. Êè¨â: ÊÍÓ iìåíi Òàðàñà Øåâ÷åíêà, 2010. 121 c.
5. Wang J., Cui N., Wei C. Optimal Rocket Landing Guidance Using Convex Optimization and Model Predictive Control. Journal of Guidance, Control, and Dynamics, 2019. 15 pp.
6. Liu X. L., Duan G. R., Teo K. L. Optimal soft landing control for moon lander. Automatica, 2008. 1097 1103 pp.
7. Cheng L., Wang Z., Jiang F. Real-time control for fuel-optimal Moon landing based on an interactive deep reinforcement learning algorithm. Tsinghua University Press, 2019. 12 pp.
8. Capolupo F., Rinalducci A. Descent & Landing Trajectory and Guidance Algorithms with Divert Capabilities for Moon Landing. The Netherlands, 2023. 20 pp.
9. Ra q M. Y., Bugmann G., Easterbrook D. J. Neural network design for engineering applications. UK: Computers and Structures journal, 2001. 1541-1552 pp.
10. Adeli H. Neural Networks in Civil Engineering: 1989-2000. USA: The Ohio State University, Department of Civil and Environmental Engineering and Geodetic Science, 2001. 126-142 pp.
11. Caterini A. L., Chang D. E. Deep Neural Networks in a Mathematical Framework. Springer Briefs in Computer Science, 2018. 84 pp.
12. Swingler K. Applying neural networks: a practical guide. New York:

Academic Press, 1996. 268 pp.

13. Antsaklis P. J. Neural Networks in Control Systems. IEEE Control Systems Magazine, 1990. 3 pp.
14. Parisini T., Zoppoli R. Neural networks for feedback feedforward nonlinear control systems. IEEE Transactions on Neural Networks, 1994. 436-449 pp.
15. Kooi1 J. E., Babuska R. Inclined Quadrotor Landing using Deep Reinforcement Learning. Czech Republic: IEEE/RSJ International Conference on Intelligent Robots and Systems, 2021. 8 pp.
16. PyTorch documentation. URL:https://pytorch.org/docs/stable/index.html
17. PyTorch documentation: Neural Networks URL: https://pytorch.org/tutorials/beginner/blitz/neural_networks_tutorial.html
18. Swamynathan M. Mastering Machine Learning with Python in Six Steps: A Practical Implementation Guide to Predictive Data Analytics Using Python. Apress, 2017. 358 pp.
19. Optimized Discrete Frechet Distance between trajectories / Devogele T., Etienne, L., Esnault M., Lardy F. France: University of Tours, 2017. 9 pp.
20. Heiter T., Manila H. Computing Discrete Frechet Distance. Austria: Technical University Vienna, Christian Doppler Laboratory for Expert Systems, 1994. 7 pp.

ÄÎÄÀÒÎÊ A

Ó äàíîì ðîçäiëi íàâåäåíî îñíîâíi ñêëàäîâi ïðîãðàìíîãî êîäó, ÿêi âèêîðèñòîâóþòüñÿ äëÿ ðåàëiçàöiї åêñïåðèìåíòàëüíî÷àñòèíè.

```
# Çàäà¹îî êëàñ äëÿ îïèñó êîñìi÷íîãî àïàðàòó. Ïàðàìåòðè: x_0: Ïî÷àòêîâà êîîðäèíàòà x. y_0: Ïî÷àòêîâà êîîðäèíàòà y.
vx_0: Ïî÷àòêîâà øâèäêiñòü ïî x. vy_0: Ïî÷àòêîâà øâèäêiñòü ïî y. m_0: Ïî÷àòêîâà ìàñà.
class Spacecraft:
    def __init__(self, x_0 = 0, y_0 = 1000, vx_0 = 0, vy_0 = -20, m_0 = 10000, disperse=True, alpha=0, w=1):
        self.alpha = alpha
        self.w = w
        self.x_0, self.y_0, self.vx_0, self.vy_0, self.m_0  = x_0, y_0, vx_0, vy_0, m_0
        # Îïèñ âåêòîðà ñòàíó.
        self.s_0 = (self.x_0, self.y_0, self.vx_0, self.vy_0, self.m_0)
        self.g = 1.62 # Ãðàâiòàöiÿ Ìiñÿöÿ.
        self.c1 = 50000 # Îáìåæåííÿ íà òÿãó.
        self.c2 = 450 * 9.81 # Ïèòîìèé ìîìóëüñ.
        self.vec_dynamics = np.vectorize(self.dynamics)
        self.t_char = (self.vy_0 + (self.vy_0**2 + 2 * self.g * self.y_0)**0.5) / self.g
        # Îáìåæåííÿ âåêòîðà ñòàíó
        self.x_range  =  -500,  500
        self.y_range  =     0, 2100
        self.vx_range =  -100,  100
        self.vy_range =  -100,   50
        self.m_range  =     0, self.m_0
        self.ranges = [self.x_range, self.y_range, self.vx_range, self.vy_range, self.m_range]

# Îá÷èñëþ¹îî äèíàìiêó ñèñòåìè.
# Ïàðàìåòðè: s - âåêòîð ñòàíó (x, y, vx, vy, m), u - âåêòîð êåðóâàííÿ (u1, u2).
# Ïîâåðòà¹: êîðòåæ çíà÷åíü, ùî ïðåäſòàâëÿþòü äèíàìiêó ñèſòåìè.
    def dynamics(self, s, u):
        (x, y, vx, vy, m) = s
        (u1, u2) = u
        ds = [None]*5
        ds[0] = vx, ds[1] = vy
        ds[2] = self.c1*(u1/m)*sin(u2)
        ds[3] = self.c1*(u1/m)*cos(u2) - self.g
        ds[4] = -(self.c1/self.c2)*u1
        return tuple(ds)
    def propagate(self, s, u, dt=0.1): # Ïðîïàãó¹ ſèſòåìó ó ÷àſi íà êðîê dt.
    # s - âåêòîð ſòàíó (x, y, vx, vy, m), u - âåêòîð êåðóâàííÿ (u1, u2), dt - êðîê ÷àſó â ſåêóíäàõ
        ds = self.dynamics(s , u)
        return tuple([s[i] + ds[i]*dt for i in range(len(s))])
    def lunar_guidance(self):
        # àäàïòîâàíèé äâîâèìiðíèé âèïàäîê
        af1, af2 = 0, 0
        rf1, rf2 = 0, 0
        vf1, vf2 = 0, 0
        tgo = self.tgo = abs(3*(rf2-self.y_0)/(2*vf2 + self.vy_0))
        C01 = af1 -(6/tgo)*(vf1 + self.vx_0) + (12/tgo**2)*(rf1-self.x_0)
        C02 = af2 -(6/tgo)*(vf2 + self.vy_0) + (12/tgo**2)*(rf2-self.y_0)
        C11 = -6*af1/tgo + (6/tgo**2)*(5*vf1 + 3*self.vx_0) - (48/tgo**3)*(rf1 - self.x_0)
        C12 = -6*af2/tgo + (6/tgo**2)*(5*vf2 + 3*self.vy_0) - (48/tgo**3)*(rf2 - self.y_0)
        C21 = (6*af1/tgo**2) - (12/tgo**3)*(2*vf1 + self.vx_0) + (36/tgo**4)*(rf1-self.x_0)
        C22 = (6*af2/tgo**2) - (12/tgo**3)*(2*vf2 + self.vy_0) + (36/tgo**4)*(rf2-self.y_0)
        self.Cs = ((C01, C11, C21), (C02, C12, C22))

        # Âiäíîâëþ¹îî ïðèſîðåíÿ òðà¹êòîði¨
```

```
            t = self.t_span = np.linspace(0, tgo)
            self.net_accel_x = net_accel_1 = C01 + C11*t + C21*t**2
            self.net_accel_y = net_accel_2 = C02 + C12*t + C22*t**2
            self.vx_sol = C01*t + 0.5*C11*t**2 + (1/3)*C21*t**3 + self.vx_0
            self.vy_sol = C02*t + 0.5*C12*t**2 + (1/3)*C22*t**3 + self.vy_0
            self.x_sol = (1/2)*C01*t**2 + (1/6)*C11*t**3 + (1/12)*C21*t**4 + self.vx_0*t + self.x_0
            self.y_sol = (1/2)*C02*t**2 + (1/6)*C12*t**3 + (1/12)*C22*t**4 + self.vy_0*t + self.y_0
            return
        def lunar_guidance_live(self, h_ratio=0.5, tmax = 1000):
            # Àäàïòîâàíèé äâîâèìåíèé âèäåîê
            # Ïàðàìåòðè:
            # h_ratio: Âiäíîøåííÿ âèñîòè, íà ÿêó ñïóñêàòèìåòüñÿ àïàðàò, äî çàãàëüíî¨ âèñîòè (çà çàìîâ÷óâàííÿì 0.5).
            # tmax: Ìàêñèìàëüíèé ÷àñ ïîäåþâàííÿ (çà çàìîâ÷óâàííÿì 1000).
            af1, af2 = 0, 0
            rf1, rf2 = 0, 0
            vf1, vf2 = 0, 0
            t = self.t = [0,]
            dt = 0.1
            x, y, vx, vy, m = ([s,] for s in self.s_0)
            self.x, self.y, self.vx, self.vy, self.m = x, y, vx, vy, m
            u1_mag = self.u1_mag = []
            u2_angle = self.u2_angle = []
            while y[-1] > 0 and t[-1]<tmax:
                tgo = self.tgo = abs(3 * (rf2 - y[-1])/(2 * vf2 + vy[-1]))
                C01 = af1 -(6/tgo)*(vf1 + vx[-1]) + (12/tgo**2)*(rf1 - x[-1])
                C02 = af2 -(6/tgo)*(vf2 + vy[-1]) + (12/tgo**2)*(rf2 - y[-1])
                # Âiäíîâëåííÿ ïðèñêîðåííÿ òðà¹êòîði¨.
                a_x = C01
                a_y = C02 + self.g
                # Îá÷èñëþ¹ìî âåêòîð êåðóâàííÿ.
                if y[-1] > h_ratio*y[0]: # Ïðèíóñîâå âiëüíå ïàäiííÿ.
                    u1_mag.append(0)
                    u2_angle.append(0)
                else:
                    u1_mag.append((1/self.c1)*m[-1]*(a_x**2 + a_y**2)**0.5)
                    u2_angle.append(arctan2(a_x, a_y))
                # Çáåðåæåííÿ ïàðàìåòðè ñòàíó òà êåðóâàííÿ.
                sf = (x[-1], y[-1], vx[-1], vy[-1], m[-1])
                uf = (u1_mag[-1], u2_angle[-1])
                new_state = self.propagate(sf, uf, dt=dt)
                x.append(new_state[0])
                y.append(new_state[1])
                vx.append(new_state[2])
                vy.append(new_state[3])
                m.append(new_state[4])
                t.append(t[-1]+dt)
            u1_mag.append(u1_mag[-1])
            u2_angle.append(u2_angle[-1])
            if t[-1] >= tmax or abs(x[-1]) > 0.1 or abs(y[-1]) > 0.1 or abs(vx[-1]) > 0.1 or abs(vy[-1]) > 0.1
            or m[-1] < 100:
                self.lunar_success = False
            else:
                self.lunar_success = True
            return

    # Êëàñ Net âèçíà÷à¹ íåéðîííó ìåðåæó çà äîïîìîãîþ áiáëiîòåêè PyTorch.
    class Net(nn.Module):
        def __init__(self, num_layers=3, num_neurons=16):
            super(Net, self).__init__()
            # Âèçíà÷åíiÿ àðõiòåêòóðè ìåðåæi ç ïîâíiñòþ çâ     ' ÿçíèìè øàðàìè.
            self.layers = nn.ModuleList([nn.Linear(5, num_neurons)])
```

```
            for _ in range(num_layers - 1):
                self.layers.append(nn.Linear(num_neurons, num_neurons))
            self.fc_out = nn.Linear(num_neurons, 2)
            # Ííïðìàëèçàöèÿ øàðiâ çà äîïîìîãîþ ðiâíîìiðíîãî ðîçïîäiëó.
            for layer in self.layers:
                a = (12 / (5 + num_neurons)) ** 0.5, nn.init.uniform_(layer.weight, a=-a, b=a)
            a = (6 / (num_neurons + 2)) ** 0.5, nn.init.uniform_(self.fc_out.weight, a=-a, b=a)
        # Ïðîõîäæåíiÿ âïåðåä (forward pass) ÷åðåç íåéðîííó ìåðåæó ç ôóíêöiÿìè àêòèâàöi¨ ReLU òà tanh.
        def forward_pass(self, x):
            for layer in self.layers:
                x = F.relu(layer(x))
            x = torch.tanh(self.fc_out(x))
            return x
        # Ñèìóëÿöiÿ ðóõó îá ' ¹êòà çà äîïîìîãîþ íåéðîííî¨ ìåðåæi.
        def forward_simulate(self, x, z, vx, vz, m, dt = 0.1, g = 1.62, c1 = 50000, c2 = 450*9.81):
            # Ïåðåòâîðåíiÿ âõiäíèõ äàíèõ íà òåíçîð PyTorch ïåðåä ïåðåäà÷åþ äî ôóíêöi¨ forward_pass.
            u1, u2 = self.forward_pass(torch.tensor([x, z, vx, vz, m], dtype=torch.float32))
            # Îá÷èñëåíiÿ çìiíè ñòàíó îá ' ¹êòà çà äîïîìîãîþ óïðàâëiíiÿ.
            ds[0] = vx, ds[1] = vz, ds[2] = self.c1*(u1/m)*sin(u2)
            ds[3] = self.c1*(u1/m)*cos(u2) - self.g

    # Åòàï ïiäáîðó ãiïåðïàðàìåòðiâ.
    # Âèçíà÷åíiÿ çíà÷åíiü ãiïåðïàðàìåòðiâ äëÿ ïîøóêó.
    lr_values = [0.001, 0.005, 0.01], momentum_values = [0.9, 0.95, 0.99], n_epochs_values = [5, 10, 15, 20, 25],
    num_layers_values = [1, 2, 3], num_neurons_values = [16, 32, 64]
    # Ñòâîðþ¹îî ôóíêöiþ äëÿ íàâ÷àííÿ ìîäåëi iç çàäàíèìè ãiïåðïàðàìåòðàìè.
    def train_model(lr, momentum, n_epochs, num_layers, num_neurons):
        net = Net(num_layers=num_layers, num_neurons=num_neurons)
        optimizer = optim.SGD(net.parameters(), lr=lr, momentum=momentum), criterion = nn.MSELoss()
        train_step = make_train_step(model=net, loss_fn=criterion, optimizer=optimizer)
        val_losses = []
        train_size = int(0.7 * len(full_data)), val_size = len(full_data) - train_size
        train_dataset, validate_dataset = torch.utils.data.random_split(full_data, [train_size, val_size])
        train_dl = data.DataLoader(train_dataset, batch_size=10, shuffle=True)
        validate_dl = data.DataLoader(validate_dataset, batch_size=10, shuffle=True)
        print(f"Training with lr={lr}, momentum={momentum}, n_epochs={n_epochs}, num_layers={num_layers},
        num_neurons={num_neurons}")
        for epoch in range(n_epochs):
            episode_loss_val = 0
            with torch.no_grad():
                for batch in validate_dl:
                    net.eval()
                    state = batch[ ' state ' ], control = batch[ ' control ' ]
                    c_hat = net(state), val_loss = criterion(control, c_hat)
                    val_losses.append(val_loss.item()), episode_loss_val += val_loss.item()
            print(f"Epoch: {epoch}, Val_loss: {sum(val_losses)/len(val_losses):.6f},
            Epoch Val Loss: {episode_loss_val/len(val_losses):.6f}")
        return sum(val_losses) / len(val_losses)
    # Âèêîíàííÿ ïîøóêó ó ñiòöi.
    best_val_loss = float( ' inf ' ), best_hyperparameters = {}
    for lr in lr_values:
        for momentum in momentum_values:
            for n_epochs in n_epochs_values:
                for num_layers in num_layers_values:
                    for num_neurons in num_neurons_values:
                        val_loss = train_model(lr, momentum, n_epochs, num_layers, num_neurons)
                        if val_loss < best_val_loss:
                            best_val_loss = val_loss
                            best_hyperparameters = { ' lr ' : lr, ' momentum' momentum, ' n_epochs' : n_epochs,
                            ' num_layers ' : num_layers, ' num_neurons' : num_neurons}
    print(f"\nBest Hyperparameters: {best_hyperparameters}, Best Validation Loss: {best_val_loss:.6f}")
```

ÄÎÄÀÒÎÊ Á

Ó öüîìó äîäàòêó ïðåäñòàâëåíi ïîðiâíÿëüíi ãðàôiêè òà òàáëèöi, ÿêi äîïîâíþþòü îñíîâíèé âìiñò äîñëiäæåííÿ, ðîçêðèâàþ÷è äåòàëi òà âàðiàöi¨ â îá÷èñëþâàëüíèõ äàíèõ.

Ìîäåëü	MSE	Âiäñòàíü Ôðåøå
• 1.	0:000382	837:07934
• 2.	0:000031	823:54272

Òàáë. 2.5. Òåñò 2. Ïîðiâíÿííÿ çíà÷åííÿ âiäñòàíi Ôðåøå.

Ðèñ. 2.3. Òåñò 2. Ìîäåëü •1. Ïîðiâíÿííÿ ðåàëüíèõ òà ïðîãíîçîâàíèõ òðà¹-êòîðié.

Ðèñ. 2.4.Òåñò 2. Ìîäåëü •2. Ïîðîâíÿííÿ ðåàëüíèõ òà ïðåääà÷åíèõ òðà¹-êòîðié.

Ìîäåëü	MSE	Âiäñòàíü Ôðåøå
• 1.	0:000382	183256941
• 2.	0:000031	183256634

Òàáë. 2.6. Òåñò 3. Ïîðîâíÿííÿ çíà÷åíîÿ âiäñòàíi Ôðåøå.

Ðèñ. 2.5. Òåñò 3. Ìîäåëü •1. Ïîðîâíÿíèÿ ðåàëüíèõ òà ïåðåäáà÷åíèõ òðà¹-êòîðié.

Ðèñ. 2.6. Òåñò 3. Ìîäåëü •2. Ïîðîâíÿíèÿ ðåàëüíèõ òà ïåðåäáà÷åíèõ òðà¹-êòîðié.

Ìîäåëü	MSE	Âiäñòàíü Ôðåøå
• 1.	0:000382	110131608
• 2.	0:000031	103430964

Òàáë. 2.7. Òåñò 4. Ïîðiâíÿííÿ çíà÷åíü âiäñòàíi Ôðåøå.

Ðèñ. 2.7. Òåñò 4. Ìîäåëü •1. Ïîðiâíÿííÿ ðåàëüíèõ òà ïåðåäáà÷åíèõ òðà¹-êòîðié.