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

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

ОПТИМІЗАЦІЯ НЕЙРОМЕРЕЖЕВОГО АЛГОРИТМУ РОЗВ'ЯЗКУ ДИФЕРЕНЦІАЛЬНИХ РІВНЯНЬ

Виконала студентка 4-го курсу
Владислава ЧЕРНОРАЙ

_____
(підпис)

Науковий керівник:
доцент, кандидат фіз.-мат. наук
Ярослав ЛІНДЕР

_____
(підпис)

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

Студентка

_____
(підпис)

Роботу розглянуто й допущено до захисту на засіданні кафедри інтелектуальних програмних систем
« 29 » травня 2023 р.,
протокол № 11
Завідувач кафедри
Олександр ПРОВОТАР

_____
(підпис)

Київ – 2023

РЕФЕРАТ

Обсяг роботи 70 сторінок, 9 рисунків, 16 використаних джерел.

АЛГОРИТМИ ОПТИМІЗАЦІЇ, ВІЗУАЛІЗАЦІЯ, ГЛИБИННІ НЕЙРОННІ МЕРЕЖІ, ДИФЕРЕНЦІАЛЬНІ РІВНЯННЯ, DEEPXDE, ЕФЕКТИВНІСТЬ, ЗВОРОТНЕ ПОШИРЕННЯ ПОМИЛКИ, КЛАСИФІКАЦІЯ, МЕТРИКИ ОЦІНКИ, МЕТОДИ РОЗВ'ЯЗАННЯ, МЕТОДИ ЧИСЕЛЬНОГО РОЗВ'ЯЗУВАННЯ, ОПТИМІЗАЦІЯ, ПРОГРАМНА РЕАЛІЗАЦІЯ, РІВНЯННЯ ШРЕДІНГЕРА, СТРУКТУРА, ФУНКЦІЯ АКТИВАЦІЇ.

Об’єкт розроблення програмної реалізації: оптимізація нейромережевого алгоритму розв’язку диференціальних рівнянь.

Мета роботи.

I. Аналіз бібліотеки DeepXDE. Вивчення функціональності, особливостей та можливостей бібліотеки DeepXDE для розв'язання диференціальних рівнянь.

II. Розроблення нейромережевого алгоритму. Розроблення та реалізація нейромережевого алгоритму для розв'язання диференціальних рівнянь з використанням бібліотеки DeepXDE.

III. Вибір оптимальної архітектури. Вивчення різних архітектур нейромереж і вибір оптимальної архітектури, яка забезпечить точність та швидкодію розв'язку диференціальних рівнянь.

IV. Оптимізація алгоритму. Дослідження та впровадження різних оптимізації нейромережевого алгоритму для покращення швидкодії та збільшення точності розв'язку.

V. Валідація результатів. Порівняння отриманих результатів з аналітичними або числовими розв'язками, валідація точності та стійкості розв'язку.

VI. Експерименти та аналіз результатів. Проведення серії експериментів з різними налаштуваннями алгоритму, аналіз отриманих результатів і висновки про ефективність оптимізованого нейромережевого алгоритму.

VII. Порівняння з іншими методами. Порівняння чисельного методу та нейромережевого алгоритму в контексті розв'язання диференціального рівняння.

Методи та інструменти розроблення.

Мова програмування: Python.

Середа розробки: Google Collaboratory (Google Colab).

Бібліотеки: DeepXDE, Numpy, Mathplot, Scipy.

Результати роботи.

Використання оптимізованих нейромережевих алгоритмів показало високу точність та ефективність у вирішенні диференціальних рівнянь. Отримані результати дозволяють швидко та надійно моделювати складні системи та здійснювати прогнозування з високою достовірністю. Оптимізація нейромережевих алгоритмів допомогла покращити швидкість збіжності та знизити витрати обчислювальних ресурсів.

Інформація щодо впровадження.

Результати досліджень із застосуванням бібліотеки DeepXDE та оптимізованих нейромережевих алгоритмів можуть бути впроваджені в різні наукові, технічні та практичні області. Це включає квантову фізику, математичне моделювання, фінансову аналітику та інші сфери, де вирішення диференціальних рівнянь є важливим завданням.

Взаємозв’язок з іншими роботами.

Результати даної роботи можуть бути використані взаємодіючи з іншими дослідженнями в галузі оптимізації нейромережевих алгоритмів та

використання бібліотеки DeepXDE. Вони сприяють удосконаленню та розвитку існуючих методів та підходів, а також можуть стати основою для подальших досліджень і нових наукових відкриттів.

Сфера застосування.

Результати даної роботи мають широку сферу застосування. Вони можуть бути використані для розв'язання диференціальних рівнянь у складних системах, прогнозування поведінки фізичних процесів, оптимізації фінансових моделей та багатьох інших областях. Впровадження цих результатів дозволить досягти більш точних та надійних розв'язків, що має велике значення для подальшого розвитку науки, технологій та практичного застосування.

ЗМІСТ

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАЧЕННЯ

AdaGrad - Адаптивний градієнтний спуск (Adaptive Gradient Descent);

Adam – Адаптивний градієнтний момент (Adaptive Gradient Estimation);

BC – Граничні умови (Boundary Conditions);

CNN – Convolutional Neural Network;

CSG – Конструктивна геометрія твердого тіла (Constructive solid geometry);

DeepONet – Deep Operator Neural Network;

FNN – Мережа прямого поширення (Forward Neural Network);

GANs – Generative Adversarial Network;

HF – Дані високої достовірності (High Fidelity);

IC - Початкові умови (Initial Conditions);

LF – Дані низької достовірності (Low Fidelity);

Loss – Функція втрат (Loss Function);

LBFGC – Limited-memory Broyden-Fletcher-Goldfarb-Shanno;

LSTM – Long short-term memory;

MAE – Mean Absolute Error;

MFNN – Multi-fidelity Neural Network;

MSE – Mean Squared Error;

PDE – Рівняння з частинними похідними (Partial differential equation);

PINN – Фізико-інформовані нейронні мережі (Physics-informed neural networks);

ReLU – Зрізаний лінійний вузол (Rectified Linear Unit);

ResNet – Залишкова нейронна мережа (Residual Network);

RMSprop – Root Mean Square Propagation;

RMSE – Root Mean Square Error;

RNN – Recurrent neural network;

SGD – Стохастичний градієнтний спуск (Stochastic Gradient Descent);

ДФ – Диференціальне рівняння;

ЗДР – Звичайне диференціальне рівняння;

РЧП – Рівняння з частинними похідними.

ВСТУП

Оцінка сучасного стану об’єкта дослідження або розробки.

Сьогодні високоточні чисельні методи для розв'язання диференціальних рівнянь є необхідним інструментом в багатьох наукових та інженерних галузях. Застосування таких методів дозволяє моделювати різноманітні фізичні явища, прогнозувати поведінку систем та вирішувати складні задачі.

Нейромережеві алгоритми здобули широку популярність в останні десятиліття як ефективний інструмент для розв'язання складних завдань в різних галузях. Використання нейромережевих алгоритмів для вирішення диференціальних рівнянь виявляється особливо перспективним підходом, оскільки ці алгоритми можуть навчатися на великій кількості даних та здатні до узагальнення на нові ситуації.

Актуальність роботи та підстави для її виконання.

Незважаючи на значний прогрес у розвитку нейромережевих алгоритмів для вирішення диференціальних рівнянь, існують виклики та проблеми, які потребують додаткового дослідження. Однією з таких проблем є оптимізація нейромережевих алгоритмів для забезпечення їх швидкості та точності. Вдосконалення цих алгоритмів дозволить ефективніше моделювати складні фізичні системи та розв'язувати диференціальні рівняння з високою точністю.

Мета й завдання роботи.

Основною метою цієї кваліфікаційної роботи є оптимізація нейромережевого алгоритму для вирішення диференціальних рівнянь на базі бібліотеки DeepXDE. Дослідження спрямоване на покращення швидкості та точності нейромережевого алгоритму з метою його більш широкого та ефективного використання у різних галузях науки та інженерії.

Для досягнення поставленої мети кваліфікаційної роботи необхідно виконати наступні завдання:

- вивчення основ роботи з бібліотекою DeepXDE та її можливостей для вирішення диференціальних рівнянь;
- розробка методів та алгоритмів оптимізації нейромережевого алгоритму на базі бібліотеки DeepXDE;
- проведення експериментів та порівняльний аналіз швидкості та точності оптимізованого алгоритму з існуючими методами;
- формулювання висновків щодо отриманих результатів та виявлення можливостей подальшого розвитку даного напрямку досліджень.

Можливі сфери застосування.

Оптимізований нейромережевий алгоритм для розв'язання диференціальних рівнянь на базі бібліотеки DeepXDE (в даній роботі буде розглянуте саме рівняння Шредінгера) може мати широке застосування в квантових дисциплінах, таких як фізика, хімія, матеріалознавство та квантові обчислення. Він може використовуватися для моделювання поведінки квантових систем, вивчення квантових явищ у різних матеріалах та структурах, розкриття квантових аспектів хімічних реакцій та спектроскопії, а також для розробки та вдосконалення квантових алгоритмів і систем обчислення. Це може сприяти кращому розумінню квантових явищ, розробці нових матеріалів та технологій, а також досягненню прогресу у галузі квантових наук.

РОЗДІЛ 1 ТЕОРЕТИЧНІ ОСНОВИ ДИФЕРЕНЦІАЛЬНИХ РІВНЯНЬ

1.1 Поняття диференціального рівняння

Диференціальне рівняння – це алгебраїчний вираз, що встановлює математичну залежність між аналітичною функцією та її похідними. Воно описує залежність між функцією та її змінами, що дозволяє аналізувати та прогнозувати допустиму динаміку системи.

В науковій нотації диференціальне рівняння може бути записане у наступному вигляді:

$$\frac{d^n y}{dx^n} = f(x, y, \frac{dx}{dy}, \frac{d^2 y}{dx^2}, ..., \frac{d^{(n-1)} y}{dx^{(n-1)}}),$$

де y – невідома функція;

x – незалежна змінна;

$\frac{d^n y}{dx^n}$ – n-та похідна функції y за змінною x;

f – функціональна залежність між похідними та змінними.

Дана узагальнена форма диференціального рівняння охоплює різноманітні типи, такі як звичайні диференціальні рівняння *(ЗДР)* і *ДР* з частинними похідними *(ЧДР)*, які залежать від кількості незалежних змінних. *ЗДР* відносяться до рівнянь з однією незалежною змінною, тоді як *ЧДР* мають більше ніж одну незалежну змінну.

Задача розв'язання диференціального рівняння полягає у визначенні функції y, яка задовольняє самому рівнянню, а також заданим початковим або крайовим умовам. Розв'язки диференціальних рівнянь широко використовуються для розуміння і прогнозування поведінки фізичних, біологічних і інженерних систем. Вони є невід'ємною складовою наукових досліджень і аналізу, оскільки дозволяють отримувати глибше уявлення про

процеси, що відбуваються в цих системах, та розробляти ефективні моделі для їх вивчення. [1]

1.2 Класифікація диференціальних рівнянь

Диференціальні рівняння можуть бути систематизовані залежно від різних особливостей, що дає змогу визначити властивості та характеристики рівнянь. Серед критеріїв, що визначають властивості диференціальних рівнянь, особливо важливо враховувати тип рівняння, що відображає його математичну форму, порядок, що визначається кількістю похідних, наявність граничних умов, часову залежність та інші фактори, які впливають на їхню динаміку та поширення.

Варто зазначити деякі з видів диференціальних рівнянь для наведення загального уявлення про класифікаційні методики.

I. ***Звичайні диференціальні рівняння (ЗДР)*** представляють собою алгебраїчні рівняння, що описують взаємозв'язок між невідомою функцією та її похідними відносно однієї незалежної змінної. Загальний математичний вигляд звичайного диференціального рівняння можна представити наступним чином:

$$F(x,\ y', y'',\ \ldots,\ y^{(n)})\ =\ 0,$$

де x – незалежна змінна;

y(x) – невідома функція;

$y', y'',\ \ldots,\ y^{(n)}$ – похідні першого, другого, ..., n-го порядку функції y(x).

II. ***Рівняння з частинними похідними (РЧП)*** – це диференціальні рівняння, в яких невідома функція залежить від кількох незалежних змінних і включає часткові похідні цієї функції по кожній з цих змінних.

III. ***Інтегро-диференціальні рівняння*** є математичними моделями, що описують складні процеси, в яких виникають як диференціальні, так і інтегральні оператори. У таких рівняннях невідома функція залежить від однієї або декількох змінних і включає як похідні, так і інтеграли від цієї функції зі специфікованими межами інтегрування.

IV. ***Дробові диференціальні рівняння*** визначаються як диференціальні рівняння, в яких похідні або їх комбінації входять у рівняння у вигляді дробових степенів. В таких рівняннях похідні можуть мати дробовий порядок, що вказує на нецілe значення або часткову похідну.

V. ***Диференціальні рівняння залежні від часу*** є математичними моделями, що описують динаміку фізичних або системних величин у залежності від незалежної змінної, якою є час. Вони виражають взаємозв'язки між функціями, підлягають диференціюванню, та їх похідні входять у рівняння з метою моделювання кінетики змін з плином часу. Це дозволяє розкрити причинно-наслідкові зв'язки між різними фізичними величинами та передбачити їх поведінку в майбутньому, враховуючи початкові умови та параметри системи.

VI. ***Диференціальні рівняння, які не залежать від часу***, є математичними моделями, в яких зміна змінних відбувається без урахування часової залежності. У таких рівняннях похідні функцій відносно незалежних змінних входять у рівняння, але сам час не фігурує в них як змінна. Це означає, що вони описують статичні або стаціонарні стани системи, де залежні змінні не змінюються з плином часу.

1.3 Класичні методи розв’язання диференціальних рівнянь

Кожен з представлених нижче методів пропонує конкретний підхід до розв'язання диференціальних рівнянь, що дозволяє отримати аналітичний розв'язок відповідно до поставлених умов задачі.

1.3.1 Метод розділення змінних

Даний метод базується на припущенні про існування функціональної залежності між змінними в диференціальному рівнянні. У даному підході передбачається декомпозиція диференціального рівняння на дві відокремлені компоненти, при цьому кожна з яких є функцією лише однієї змінної. Подальше застосування інтегрування до обох компонент дозволяє отримати загальний розв’язок. [2]

1.3.2 Метод варіації довільної сталої

Використовується для розв'язання лінійних неоднорідних диференціальних рівнянь, де розглядається пошук часткового розв'язку, представленого у вигляді лінійної комбінації загального розв'язку відповідного однорідного рівняння з коефіцієнтами, які можуть залежати від змінних. Далі використання методу варіації сталих дозволяє визначити значення цих коефіцієнтів з метою отримання часткового розв'язку, який відповідає специфічним умовам задачі. [3]

1.3.3 Метод інтегруючого множника

Метод інтегруючого множника застосовується для розв'язання диференціальних рівнянь, які є нелінійними, але можуть бути приведені до рівнянь, помножених на певний інтегруючий множник. Цей метод дозволяє спростити рівняння шляхом впровадження множника, який забезпечує можливість інтегрування. Подальше використання методу інтегруючого множника дозволяє знайти загальний розв'язок шляхом здійснення необхідних інтегралів. [4]

1.3.4 Метод невизначених коефіцієнтів

Він ґрунтується на припущенні про форму розв'язку та визначенні невідомих коефіцієнтів шляхом підстановки цієї форми розв'язку в рівняння та вирішення отриманої системи рівнянь.

1.4 Чисельні методи розв’язання диференціальних рівнянь

1.4.1 Метод Ейлера

Метод Ейлера є чисельним методом, що використовується для наближеного розв'язання звичайних диференціальних рівнянь першого порядку. Цей метод базується на апроксимації похідної розв'язку за допомогою диференціального відношення.

Даний метод використовує просту рекурсивну формулу для розв’язання диференціального рівняння виду:

$$\frac{\partial y}{\partial x} = f(x, y),$$

де $f(x, y)$ – задана функція.

Починаючи з початкової точки (x_0, y_0), метод Ейлера обраховує наближені значення розв’язку у послідовних точках (x_1, y_1), (x_2, y_2), …, (x_n, y_n) використовуючи заданий крок h.

Кроки алгоритму методу Ейлера.

I. Ініціалізувати початкову точку (x_0, y_0), де y_0 - початкове значення розв'язку.
II. Вибрати значення кроку h, яке визначає відстань між точками, на яких обчислюється розв'язок.
III. Для кожної наступної точки (x_{n+1}, y_{n+1}):
   - обчислити приблизне значення похідної: $f_n = f(x_n, y_n)$;
   - обчислити приблизне значення розв'язку: $y_{n+1} = y_n + h * f_n$;
   - обчислити нове значення x: $x_{n+1} = x_n + h$.
IV. Повторити крок 3 до досягнення бажаної точки або інтервалу.

Точність даного методу залежить від величини кроку h, а глобальна похибка оцінюється приблизно як $\Theta(h)$. В порівнянні з більш точними методами, метод Ейлера може мати відносно велику похибку, особливо для складних рівнянь або великих значень кроку h. Тому, для високої точності рекомендується використовувати більш точні чисельні методи, такі як метод Рунге-Кутта.

1.4.2 Метод Рунге-Кутта

Метод Рунге-Кутта використовується для розв'язання диференціальних рівнянь першого порядку і зазвичай має вищий порядок точності порівняно з методом Ейлера. Він базується на обчисленні проміжних кроків, що дозволяє отримати більш точний результат.

Алгоритм методу Рунге-Кутта.

I. Вибрати початкову точку (x_0, y_0).

II. Вибрати значення кроку h, який визначає відстань між точками, на яких обчислюється розв'язок.

III. Для кожного кроку (x_n, y_n):

- обчислити коефіцієнти $k_1, k_2, k_3, \ldots, k$ залежно від обраної схеми методу Рунге-Кутта; кожен коефіцієнт k_i, залежить від попереднього кроку (x_n, y_n) та попереднього коефіцієнта k_{i-1};
- обчислити нове значення розв'язку у$_{+1}$ на основі коефіцієнтів $k_1, k_2, k_3, \ldots, k$;
- обчислити нове значення x_{n+1}: $x_{n+1} = x_n + h$.

IV. Повторити крок 3 до досягнення потрібної точки або інтервалу.

1.5 Огляд використання диференціальних рівнянь у різних областях наукового та технічного дослідження.

Диференціальні рівняння являються потужним інструментом у наукових та технічних дослідженнях, а також мають широкий спектр застосування у різноманітних галузях. Вони дозволяють моделювати залежності між неперервно змінними величинами та їх похідними, які виникають у різних наукових дисциплінах, зокрема у фізиці, інженерії, економіці, біології та медицині.

У фізиці диференціальні рівняння використовуються для опису руху тіл, взаємодії електромагнітних полів, передачі тепла, квантової механіки та інших фізичних явищ. У технічних науках ці рівняння моделюють різні системи, такі як електричні ланцюги, механічні системи, системи керування та автоматика. Вони дозволяють аналізувати та прогнозувати поведінку цих систем для поліпшення їх функціональності та ефективності.

Диференціальні рівняння знаходять широке застосування у математичній біології для моделювання динаміки популяцій, поширення захворювань, фізіологічних процесів та інших біологічних систем. В економіці вони використовуються для аналізу ринкових та фінансових процесів, прогнозування тенденцій та оптимізації стратегій. Ці рівняння дозволяють формалізувати та розв'язувати складні проблеми, пов'язані зі змінними в часі та просторі, а також досліджувати вплив різних факторів на біологічні та економічні системи.

У науках про землю диференціальні рівняння широко використовуються для моделювання комплексних геофізичних явищ і процесів. Вони дозволяють аналізувати та прогнозувати кліматичні зміни, гідрологічні процеси, поширення забруднень та інші важливі аспекти земної системи.

У різних областях застосування диференціальних рівнянь сприяє формалізації складних проблем, які включають залежності від часу, простору та інших змінних. Це дозволяє моделювати поведінку систем і розв'язувати їх залежно від поставлених умов. Велика сила диференціальних рівнянь полягає в їх здатності передбачати та пояснювати різноманітні фізичні, біологічні, економічні та геофізичні явища. Вони дозволяють досліджувати вплив різних факторів на системи та розуміти їх поведінку в різних умовах, що є важливим для розвитку наукових та технічних досліджень.

РОЗДІЛ 2 ТЕОРЕТИЧНІ ОСНОВИ НЕЙРОННИХ МЕРЕЖ

2.1 Поняття нейронної мережі

Нейронні мережі є математичними моделями, які апроксимують функціональну структуру біологічних нервових систем. Вони використовуються для моделювання імітуючих обчислювальних систем здатністю до обробки інформації та виконання складних обчислень. Ці мережі складаються з набору штучних нейронів, які виконують обчислення, та зв'язків між ними, які передають сигнали та інформацію між нейронами.

Структура нейронної мережі визначається її архітектурою. Нейрони організовані у впорядковані шари, де кожен шар має свою функціональну роль. Вхідний шар отримує вхідні дані, які подаються на вхід мережі. Ці дані передаються через проміжні шари, які називаються прихованими шарами, і завершуються на вихідному шарі, де виробляється вихідний результат мережі. Кожен нейрон у шарі отримує сигнали від попереднього шару, обчислює їх шляхом застосування активаційної функції і передає оброблені значення наступному шару.

Процес навчання нейронної мережі полягає у налаштуванні вагових коефіцієнтів нейронів, щоб мережа могла пристосуватися до навчальних даних та здійснювати прогнози або класифікацію для нових вхідних даних. Навчання нейронної мережі може бути здійснене за допомогою алгоритмів зворотного поширення помилки, де помилки прогнозу мережі коригуються з метою мінімізації помилки та досягнення бажаного результату.

Нейронні мережі знаходять широке застосування в різних галузях науки і технологій, зокрема в областях штучного інтелекту, комп'ютерного зору, обробки природної мови та багатьох інших. Вони успішно використовуються для розв'язання складних завдань, таких як розпізнавання образів, прогнозування, класифікація та рекомендації. Нейронні мережі можуть ефективно впоратися з великими обсягами даних і виявляти складні

залежності, що дозволяє їм здійснювати високоякісний аналіз та обробку інформації.

2.2 Структура нейронної мережі

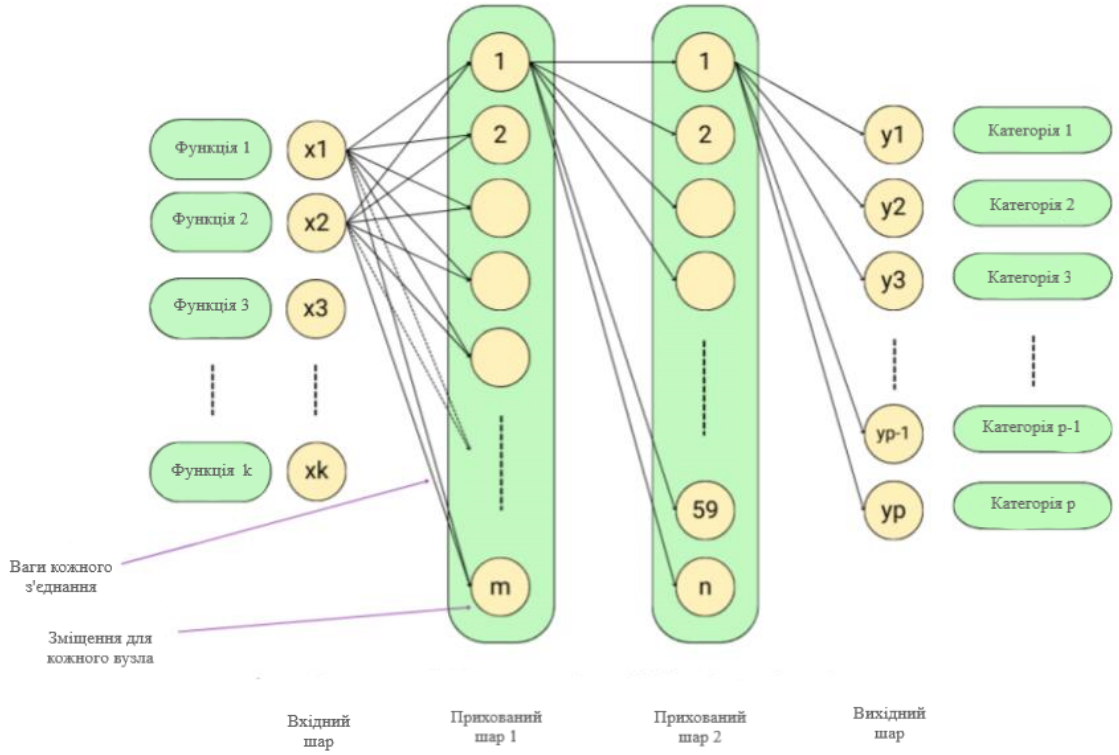

Рис.1 Схематичне зображення архітектури нейронної мережі

2.2.1 Нейрони

Нейрон є фундаментальним структурним елементом нейронної мережі, який виконує обчислювальні операції.

Ідея нейрона базується на біологічних нейронах, які присутні у людському мозку. Кожен нейрон приймає вхідні сигнали, які потім обробляються шляхом вагового коефіцієнту, виконується додавання зміщення *(bias)* і подається на функцію активації для отримання вихідного сигналу. Цей процес можна математично описати як суперпозицію вхідних сигналів, де кожен сигнал має свою вагу та зміщення. Після цього до отриманої лінійної комбінації застосовується нелінійна активаційна функція, і її результат використовується як остаточний вихідний сигнал.

2.2.2 Вхідний шар (Input Layer)

Вхідний шар нейронної мережі є ініційним шаром, що призначений для прийому та початкової обробки вхідних даних. Він складається з набору нейронів, кожен з яких відповідає одній ознаці чи характеристиці вхідного набору даних. Розмір вхідного шару визначається розміром вхідних даних, а кількість нейронів у ньому відображає кількість вхідних ознак.

Функція вхідного шару полягає в прийомі вхідних даних та їх передачі для подальшої обробки наступним шаром мережі. Кожен нейрон вхідного шару отримує значення відповідної вхідної ознаки та передає його через зв'язки до нейронів наступного шару.

Врахування розміру та структури вхідного шару є критичним при проектуванні нейронних мереж, оскільки він забезпечує початкову обробку та числове представлення вхідних даних для подальшої обробки моделлю. Вірне визначення параметрів вхідного шару є важливим для досягнення ефективної роботи мережі та досягнення поставлених цілей.

2.2.3 Приховані шари (Hidden Layers)

Приховані шари в нейронних мережах виконують функцію обчислювального простору, де відбувається нелінійна обробка вхідних сигналів. Вони допомагають моделі виявляти та вивчати складні неявні залежності між вхідними та вихідними даними шляхом застосування нелінійних функцій активації та зваженої суми вхідних сигналів, обчислюваних шляхом зв'язків між нейронами в прихованих шарах.

Розташування та кількість прихованих шарів є важливими параметрами, які можуть впливати на адаптованість нейронної мережі до складності проблеми. Більша кількість прихованих шарів може дозволити моделі виявити більш складні та високорівневі залежності, але це може вимагати більше обчислювальних ресурсів та складніший процес навчання. Загалом,

приховані шари є важливим компонентом нейронних мереж, які допомагають відтворити складні взаємозв'язки між вхідними та вихідними даними.

2.2.4 Вихідний шар (Output Layer)

Вихідний шар в нейронних мережах є завершальним шаром обчислень, який генерує вихідні сигнали або прогнози на основі обробки вхідної інформації. Кожен нейрон у вихідному шарі зазвичай відповідає за окремий вихід або класифікацію у відповідності до конкретної задачі машинного навчання.

Зв'язки між нейронами у прихованих шарах та вхідному шарі послідовно обробляються, і їх результати передаються до вихідного шару. Нейрони у вихідному шарі використовують активаційну функцію для створення вихідних значень, які можуть бути відповідями на конкретні запити або результатами прогнозування.

Вихідний шар має ключове значення для досягнення поставленої задачі, такої як класифікація, регресія або генерація. Його конфігурація та активаційні функції залежать від конкретної задачі та характеру оброблюваних даних у мережі.

2.2.5 З’єднання та ваги

Ваги є числовими параметрами, які налаштовуються під час процесу навчання нейронної мережі. Кожен з'єднаний сигнал між нейронами має свою вагу, яка визначає його вплив на вихідний сигнал обчислення нейрона. Формально, ваги можна представити як множники, які множаться на вхідні сигнали та сумуються для отримання вагованої суми. Далі ця вагована сума може піддаватися активаційній функції для отримання кінцевого вихідного сигналу нейрона.

2.2.6 Функція активації (Activation Function)

Функція активації визначає вихідний сигнал нейрона на основі його вхідних даних та ваг. Вона визначає, коли нейрон повинен бути активований і

передати сигнал, коли неактивний. У нейронних мережах використовуються різні типи функцій активації, такі як сигмоїдна функція, функція *ReLU (Rectified Linear Unit)*, гіперболічний тангенс та інші. Ці функції використовуються для забезпечення нелінійності та нелінійної поведінки нейрона, що дозволяє нейронній мережі моделювати складніші функції та вирішувати різноманітні завдання.

У нейронних мережах активаційні функції можна класифікувати на два основні типи:

I. ***Лінійні активаційні функції*** у нейронних мережах передають зважений вхідний сигнал без виконання додаткових перетворень. Вони встановлюють лінійну залежність між зваженим вхідним сигналом та вихідним значенням нейрона. Цей тип активаційних функцій часто використовується у вихідному шарі нейронної мережі, де важливо зберігати пропорційність між входом і виходом нейрона.

II. ***Нелінійні активаційні функції*** у нейронних мережах використовуються для створення нелінійних залежностей між зваженими вхідними сигналами та їх відповідними вихідними значеннями. Ці функції допомагають нейронам виявляти та вивчати складні шаблони, які можуть бути присутні у вхідних даних. Вони використовуються переважно у прихованих шарах нейронної мережі, оскільки дозволяють моделі виконувати нелінійну обробку та апроксимувати складні функції.

2.2.7 Функція втрат (Loss Function)

Функція втрат, також відома як функція витрат або цільова функція, є математичним інструментом для кількісного вимірювання різниці між прогнозованими і фактичними значеннями в моделі машинного навчання. Ця функція використовується для оцінки продуктивності моделі та керування процесом навчання.

Вибір конкретної функції втрат залежить від конкретної задачі, такої як класифікація, регресія або підсилення навчання. У процесі навчання моделі, головною метою є мінімізація значення функції втрат, оскільки це призводить до покращення прогностичних здібностей моделі.

2.2.8 Зворотне поширення помилки (Backpropagation)

Зворотне поширення помилки є основним алгоритмом для обчислення градієнтів функції втрат за вагами у моделі машинного навчання. Цей алгоритм дозволяє моделі вдосконалюватись, адаптуючись до даних шляхом коригування ваг відповідно до помилок, що вона робить при здійсненні прогнозів. Він передає помилку від виходу до входу нейромережі, обчислюючи градієнти і використовуючи їх для оновлення ваг з використанням методів оптимізації, таких як стохастичний градієнтний спуск. Даний процес повторюється протягом процесу навчання з метою покращення прогностичних здібностей моделі. Зворотне поширення помилки виступає важливою складовою навчання нейромережі і сприяє адаптації моделі до даних та досягненню більш точних прогнозів.

2.2.9 Алгоритм оптимізації

В контексті нейронних мереж, *алгоритм оптимізації* включає процес налаштування ваг та параметрів моделі з метою мінімізації функції втрат. Його головна мета - знайти оптимальні значення параметрів, які найкраще відповідають навчальним даним і дають найкращі результати моделі. [5]

2.3 Навчання нейромережевих моделей

Навчання нейронної мережі – ітеративний процес, в ході якого внутрішні параметри (ваги та зсуви) нейронної мережі налаштовуються за допомогою навчальних даних з метою досягнення високої продуктивності в конкретній задачі. Основна мета навчання полягає в мінімізації розбіжності між передбаченими значеннями, що генеруються нейронною мережею, та очікуваними значеннями, що відповідають цим даним. Це досягається

шляхом оптимізації функції втрат та використання методів градієнтного спуску для оновлення параметрів мережі з метою покращення її прогностичних здібностей.

Етапи навчання моделей нейронних мереж.

I. Підготовка даних:
   - поділ даних на навчальний, тестовий та, при необхідності, валідаційний набори;
   - попереднє опрацювання даних, таке як масштабування, нормалізація або перетворення ознак.

II. Встановлення архітектури мережі:
   - вибір типології шарів та їх послідовності в структурі мережі;
   - визначення оптимальної кількості шарів та нейронів у кожному з них з урахуванням розглянутих факторів;
   - вибір відповідних функцій активації для кожного шару.

III. Ініціалізація параметрів:
   - початкове встановлення вагових коефіцієнтів та зсувів мережі за допомогою випадкових значень.

IV. Пряме поширення:
   - трансляція вхідних даних через мережу в прямому напрямку від вхідного шару до вихідного шару;
   - обчислення виходу кожного шару шляхом застосування вагових коефіцієнтів, зсувів та функції активації для визначення активності нейронів у кожному шарі.

V. Виконання оцінки похибки:
   - аналіз порівняння прогнозованих значень нейронної мережі з фактичними значеннями та обчислення відхилення з використанням функції втрат.

VI. Зворотне поширення:

- обчислення градієнтів функції помилки щодо ваг та зсувів нейронної мережі;
- передача градієнтів у зворотному напрямку через мережу з метою оновлення параметрів.

VII. Актуалізація параметрів:

- використання оптимізаційних алгоритмів, зокрема градієнтного спуску, для оновлення значень ваг та зсувів у мережі.

VIII. Ітерації процесу навчання:

- повторення процесу виконання кроків 4-7 для кожного навчального зразка у наборі даних (епоха навчання);
- продовження навчання може здійснюватись протягом кількох епох з метою досягнення кращої адаптації мережі до навчальних даних.

IX. Оцінка моделі:

- виконання оцінки продуктивності моделі на незалежній тестовій вибірці для оцінки її здатності до узагальнення та здатності до виконання роботи з даними, яких раніше не було використано для навчання.

X. Ітеративний підхід:

- оптимізація гіперпараметрів моделі, таких як швидкість навчання, кількість шарів та нейронів, з метою досягнення кращої адаптації та поліпшення результатів;
- повторення послідовності кроків 2-9 з метою ітеративного удосконалення моделі та досягнення оптимальних результатів.

2.4 Основні алгоритми навчання нейронних мереж

2.4.1 Метод зворотного поширення помилки (Backpropagation)

Зворотне поширення помилки - це метод навчання нейронних мереж, що включає передачу сигналу, обчислення функції втрат, визначення градієнтів та оновлення параметрів мережі.

Передача сигналу через шари мережі здійснюється з використанням активаційних функцій, а функція втрат вимірює розбіжність між прогнозованими значеннями та очікуваними даними. Зворотне поширення обчислює градієнти для параметрів мережі, які передаються у зворотному напрямку через мережу. Параметри мережі оновлюються у напрямку, протилежному градієнту, використовуючи методи оптимізації, такі як градієнтний спуск.

Даний підхід дозволяє здійснювати оптимізацію параметрів мережі з метою поліпшення її прогнозуючих можливостей.

2.4.2 Стохастичний градієнтний спуск (Stochastic Gradient Descent, SGD)

Стохастичний градієнтний спуск *(SGD)* є методом оптимізації для навчання нейронних мереж, в якому оновлення ваг моделі відбувається після кожного навчального прикладу. Цей метод є ефективним для обробки великих обсягів даних, оскільки градієнт обчислюється на основі випадкового піднабору даних. Випадковий вибір прикладів і оновлення ваг сприяють швидшій збіжності моделі до оптимальних значень.

SGD демонструє переваги в контексті обробки великих обсягів даних, зменшує вимоги до пам'яті та забезпечує більш швидку адаптацію до змін у даних. Проте, він може проявляти менш стабільну збіжність та ризик застійного стану у локальних мінімумах. З метою поліпшення стабільності та збіжності, *SGD* може бути модифікований шляхом налаштування параметра швидкості навчання та використання моменту або адаптивної швидкості навчання.

2.4.3 Адаптивний градієнтний спуск (Adaptive Gradient Descent, AdaGrad)

Адаптивний градієнтний спуск *(AdaGrad)* є оптимізаційним алгоритмом, який динамічно змінює швидкість навчання для кожного

параметра в мережі на основі попередніх градієнтів. Цей метод ефективно навчає модель на даних з розрідженою структурою, збільшуючи швидкість навчання для параметрів з великими градієнтами та зменшуючи її для параметрів з меншими градієнтами. *AdaGrad* автоматично адаптує швидкість навчання, що спрощує вибір оптимального значення та уникає необхідності вручному налаштуванні.

AdaGrad має свої обмеження, зокрема, накопичення квадратів градієнтів, що може призвести до зменшення швидкості навчання з часом. Для подолання цього обмеження були розроблені більш розширені алгоритми оптимізації, такі як *RMSProp* та *Adam*. Ці алгоритми забезпечують більш стійку та ефективну оптимізацію глибоких моделей навчання.

2.4.4 Адаптивний момент (Adaptive Moment Estimation, Adam)

Adam є ефективним алгоритмом оптимізації для навчання нейронних мереж, який комбінує переваги адаптивного градієнтного спуску та методу адаптивного моменту. Цей метод адаптивно змінює швидкість навчання та момент градієнта, враховуючи історію градієнтів. Це сприяє згладжуванню коливань градієнта та покращенню збіжності навчання.

Алгоритм *Adam* використовує два моменти: перший момент, який представляє експоненційно згладжену оцінку градієнта, та другий момент, який представляє згладжену оцінку квадрата градієнта. Крім того, використовується корекція зміщення для оцінок моментів з метою усунення початкового спотворення під час навчання.

Під час оновлення ваг нейронної мережі, алгоритм *Adam* використовує моменти для кожного параметра та адаптивно налаштовує швидкість навчання. Зміщені оцінки першого та другого моментів використовуються для оновлення ваг шляхом їх віднімання та ділення на корінь квадратний від зміщеної оцінки другого моменту.

Алгоритм *Adam* відрізняється швидкою збіжністю та стабільністю при навчанні великих нейронних мереж та в задачах з великим обсягом даних. Він автоматично адаптує швидкість навчання для кожного параметра, усуваючи необхідність вручну налаштовувати цей гіперпараметр. Adam також ефективно працює в умовах шуму, що робить його популярним вибором для навчання нейронних мереж.

2.5 Класифікація нейромереж

2.5.1 Нейронна мережа прямого поширення (FNN)

Нейронна мережа прямого поширення *(FNN)* – це тип штучної нейронної мережі, де інформація передається лише в одному напрямку, від вхідного шару до вихідного шару, без циклічних зв'язків. Кожен нейрон обчислює лінійну комбінацію вхідних даних з вагами і застосовує нелінійну активаційну функцію до результату. Це дозволяє моделювати складні нелінійні відношення між вхідними та вихідними даними.

Архітектура нейронної мережі прямого поширення зазвичай включає вхідний шар, приховані шари і вихідний шар. Вхідний шар отримує вхідні дані, приховані шари виконують обчислення, а вихідний шар генерує остаточний результат. Мережу тренують шляхом налаштування ваг нейронів з метою мінімізації різниці між вихідними значеннями та бажаними значеннями. За допомогою алгоритмів оптимізації, таких як зворотнє поширення помилки, оновлюються ваги мережі на основі градієнта функції втрат. [6]

2.5.2 Зворотні нейромережі (RNNs)

Зворотні нейромережі *(RNNs)* – це тип нейромереж, що використовується для аналізу послідовних даних, де кожен елемент залежить від попередніх. Вони здатні враховувати контекстну інформацію з попередніх кроків, що корисно для прогнозування, класифікації та генерації послідовностей. *RNN* мають зв'язки між вузлами на різних часових кроках,

що дозволяє зберігати та використовувати інформацію з попередніх кроків обробки даних. [7]

2.5.3 Згорткові нейромережі (CNNs)

Згорткові нейромережі *(CNNs)* є потужними моделями для обробки зображень і відео. Вони використовують шари згортки для виявлення локальних ознак у вхідних даних, таких як краї, текстури і форми. Шари пулінгу допомагають зменшити розмір ознак і забезпечують інваріантність до малих зміщень у даних. Ці шари використовуються у комбінації з повнозв'язаними шарами для класифікації та розпізнавання об'єктів. [8]

2.5.4 Нейромережі з довгою короткочасною пам'яттю (LSTM)

Нейромережі з довготривалою короткочасною пам'яттю *(LSTM)* є рекурентними нейронними мережами, спеціально розробленими для роботи з послідовними даними. Вони були створені з метою вирішення проблем, які виникають у стандартних рекурентних мережах, таких як проблема зникнення та вибуху градієнту.

Основною особливістю *LSTM* є використання спеціальних воріт для керування потоком інформації всередині мережі. Ці ворота визначають, яка інформація повинна бути збережена, яка повинна бути забута, а також яка частина поточного стану має бути виведена. Це дозволяє *LSTM* ефективно працювати з довготривалими залежностями в послідовних даних. [9]

2.5.5 Мережі з увагою (Attention Networks)

Мережі з увагою – це архітектура моделей, яка дозволяє їм активно фокусуватись на важливих або релевантних частинах вхідних даних під час обробки і аналізу. Механізм уваги зазвичай використовується разом з рекурентними або трансформерними мережами, особливо в задачах обробки природної мови, наприклад, у машинному перекладі або генерації тексту. Цей механізм дозволяє моделі активно приділяти увагу певним словам, фразам або контекстуальним залежностям, що сприяє кращій якості результатів. [10]

РОЗДІЛ 3 ЗАСТОСУВАННЯ БІБЛІОТЕКИ DeepXDE ДЛЯ ВИРІШЕННЯ ДИФЕРЕНЦІАЛЬНИХ РІВНЯНЬ

3.1 DeepXDE: Використання глибокого навчання для розв'язання диференціальних рівнянь.

DeepXDE є потужною бібліотекою глибокого навчання, спеціально призначеною для вирішення диференціальних рівнянь. Вона використовує фізично-засновані нейронні мережі *(PINNs)*, щоб ефективно вбудовувати *ДР* безпосередньо в функцію втрат нейронної мережі з використанням автоматичної диференціації. Такий підхід дозволяє моделювати фізичні процеси з високою точністю та швидкістю, спрощуючи процес вирішення складних ДР шляхом використання глибокого навчання.

Однією з важливих переваг *DeepXDE* є його здатність вирішувати як прямі, так і обернені задачі в контексті диференціальних рівнянь. У випадку прямих задач, де потрібно знайти розв'язок *ДР* на основі вихідних умов, бібліотека використовує початкові і крайові умови для розв'язання *ДР*. Загалом, *DeepXDE* може моделювати фізичні процеси шляхом знаходження точного розв'язку *ДР* з використанням глибокого навчання.

Крім того, *DeepXDE* може вирішувати обернені задачі, коли відомі додаткові вимірювання. Це означає, що на основі набору вимірювань можна відновити параметри *ДР* або визначити початкові/крайові умови. Ця здатність дозволяє використовувати бібліотеку для широкого спектру застосувань у наукових дослідженнях та інженерних розрахунках, де обернені задачі є важливим етапом в аналізі та моделюванні фізичних систем.

DeepXDE вміє ефективно моделювати проблеми зі складною геометрією шляхом підтримки складних геометричних областей на основі конструктивної міцності. Цей підхід дозволяє бібліотеці вирішувати задачі, пов'язані зі складними формами тіл або областями з отворами. Використання конструктивної міцності дозволяє зручно та гнучко моделювати реальні

завдання з різних галузей, де геометрична складність є суттєвим аспектом. Такий підхід розширює можливості *DeepXDE* і дозволяє вирішувати більш реалістичні й складні задачі, забезпечуючи точні та адаптивні моделі для аналізу та прогнозування фізичних явищ.

Загалом, *DeepXDE* представляє собою цінний інструмент для дослідників, викладачів та інженерів, які займаються розв'язанням диференціальних рівнянь. Ця бібліотека допомагає прискорити розвиток наукового машинного навчання і сприяє вирішенню складних завдань у галузі обчислювальної науки та інженерії.

3.2 Аналіз існуючих методів застосування бібліотеки DeepXDE у розв'язанні диференціальних рівнянь.

Використання нейромереж у галузі обчислювальної математики та наукового обчислення відкриває перспективний підхід до розв'язання диференціальних рівнянь, які є ключовим інструментом для моделювання поведінки різноманітних систем, включаючи фізичні, біологічні та інженерні. Традиційні чисельні методи, які були використовувані протягом тривалого часу, мають свої обмеження, особливо при роботі зі складними геометріями, високовимірними задачами та нелінійною динамікою. Однак, останні прориви в галузі глибокого навчання та розвитку нейромереж відкрили нові перспективи для ефективного розв'язання складних обчислювальних проблем, що виникають у контексті диференціальних рівнянь.

Далі будуть наведені основні стратегії, що застосовуються бібліотекою *DeepXDE* у розв'язанні диференціальних рівнянь.

I. ***Фізико-інформовані нейронні мережі (PINN)*** є універсальними функціональними апроксиматорами, які інтегрують фізичні закони у процес навчання. Вони виявляються корисними у випадках обмеженої доступності даних, коли стандартні методи машинного навчання можуть бути неефективними. Використання апріорних фізичних знань

як обмежень під час навчання покращує точність функціональної апроксимації, що дає змогу нейронним мережам знаходити правильні розв'язки та добре узагальнювати навіть за обмеженої кількості тренувальних прикладів.

Механізм роботи. Ідея даного методу базується на включенні фізичних рівнянь або граничних умов безпосередньо в функцію втрат під час навчання мережі. В процесі тренування мережа намагається знайти оптимальні параметри, що задовольняють як дані, так і фізичні обмеження системи. Застосування автодиференціації дозволяє обчислити градієнти та інтегрувати фізичні рівняння в функцію втрат для забезпечення виконання фізичних законів. Цей підхід дозволяє мережі "навчитися" фізичним законам системи, яку вона моделює, і забезпечує більш точні та надійні прогнози. (Рис. 2)

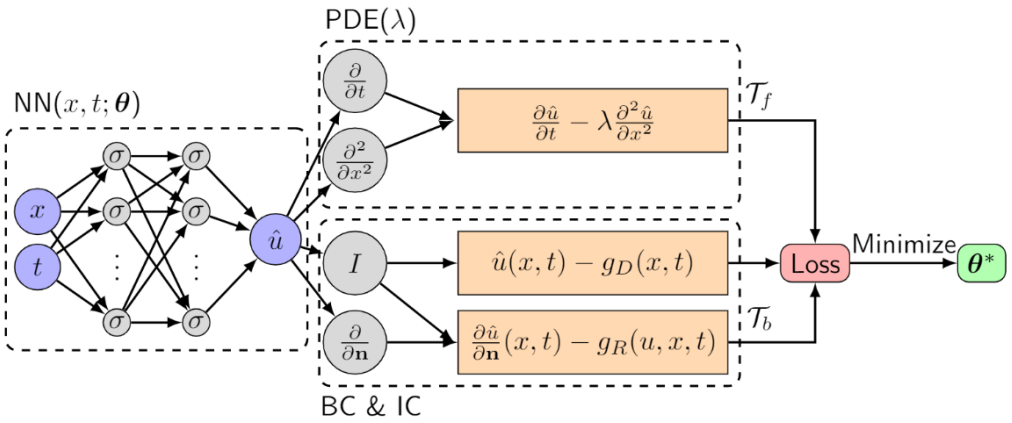

Рис. 2 Схематичне зображення архітектури фізико-інформованої нейронної мережі (*PINN*) для вирішення диференціальних рівнянь для розв'язання рівняння дифузії $\frac{\partial u}{\partial t} = \lambda \frac{\partial^2 u}{\partial x^2}$ з комбінованими граничними умовами (*BC*) $u(x, t) = g_D(x, t)$ в $\Gamma_D \subset \partial\Omega$ та $\frac{\partial u}{\partial n}(x, t) = g_R(u, x, t)$ в $\Gamma_R \subset \partial\Omega$. Початкова умова (IC) розглядається у контексті як особлива форма граничних умов. T_f та T_b – дві множини точок, які представляють залишкові значення для самого рівняння та для граничних/початкових умов.

Алгоритм PINN для вирішення диференціальних рівнянь.

I. Реалізовуємо нейронну мережу $\widehat{u}(x, \theta)$ з параметром θ.

II. Позначимо два набори для навчання: T_f для самого рівняння та T_b для граничних та початкових умов.

III. Встановимо функцію втрат, яка складатиметься з залишкових значень рівняння *PDE* і граничних та початкових умов.

IV. Навчаємо нейронну мережу з метою знаходження оптимальних параметрів θ^* шляхом мінімізації функції втрат $Loss(\theta, T)$.

Функція втрат (loss function):

$$Loss(\theta, T) = \omega_f * Loss_f(\theta, T) + \omega_b * Loss_b(\theta, T),$$

У цій формулі $Loss(\theta, T)$, де θ - параметри нейронної мережі, а T – навчальні дані, використовується для обчислення загальної функції втрат. Ця функція втрат складається з двох компонентів: $Loss_f(\theta, T)$ та $Loss_b(\theta, T)$, які відповідають за оцінку втрат, пов'язаних з рівнянням та граничними/початковими умовами відповідно.

Фактор ω_f представляє ваговий коефіцієнт, який визначає важливість компоненти $Loss_f(\theta, T)$ у загальній функції втрат. Аналогічно, фактор ω_b визначає ваговий коефіцієнт для компоненти $Loss_b(\theta, T)$. Змінюючи значення ω_f та ω_b, можна контролювати внесок кожної компоненти у загальну функцію втрат та налаштовувати, як нейронна мережа буде враховувати рівняння та граничні/початкові умови під час процесу навчання. [11]

II. ***Deep Operator Network (DeepONet)*** представляє собою нейромережевий алгоритмом, який комбінує глибокі нейронні мережі з операторними методами для моделювання фізичних процесів. *DeepONet* використовує операторну мережу для апроксимації фізичних операторів і

функцію-вузол для апроксимації розв'язків рівнянь, що дозволяє моделювати складні фізичні системи. В процесі навчання враховуються фізичні обмеження, що сприяє покращенню точності та універсальності моделі. Використовуючи операторну мережу, яка апроксимує фізичні оператори, *DeepONet* може вирішувати різні типи рівнянь, такі як диференціальні рівняння чи інтегральні рівняння. [12]

Механізм роботи. *DeepONet* в контексті вирішення диференціальних рівнянь працює наступним чином. Спочатку використовується операторна мережа, яка навчається апроксимувати фізичний оператор, що з'являється в диференціальному рівнянні. Ця мережа вміє обчислювати значення оператора для будь-яких вхідних даних. Потім використовуються функції-вузли, які апроксимують розв'язок диференціального рівняння. Ці функції-вузли навчаються моделювати розв'язок шляхом наближення функції, яка задовольняє рівнянню і початковим умовам. (Рис. 3)

Під час навчання *DeepONet* використовується метод оптимізації для зменшення розбіжності між значеннями оператора, обчисленими мережею, і фактичними значеннями оператора відомих розв'язків. Це покращує точність моделі та дозволяє враховувати фізичні обмеження.

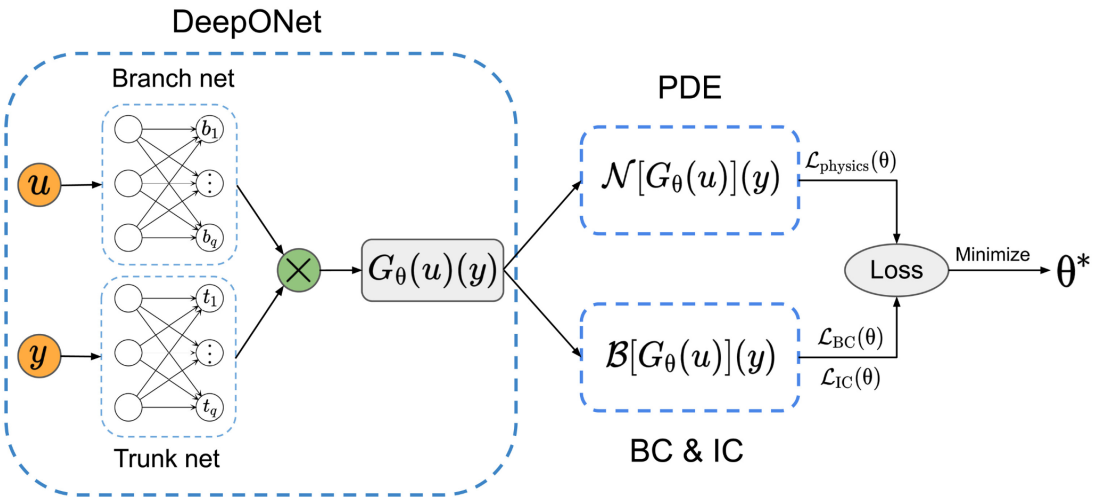

Рис. 3 Схематичне зображення архітектури Deep Operator Network (DeepONet). Дана топологія складається з гілкової і стовпчастої мереж, які витягують латентні

представлення вхідних функцій і координат. Потім ці представлення об'єднуються, отримуючи неперервні і диференційовані вихідні функції. Застосовується автоматичне диференціювання для регуляризації вихідних значень DeepONet, щоб задовольнити систему рівнянь частинних похідних. *BC* & *IC* - граничні та початкові умови відповідно.

III. ***Multi-fidelity нейронна мережа (MFNN)*** використовується для поєднання даних низької достовірності ***(LF)*** та даних високої достовірності ***(HF)*** з метою поліпшення точності та оптимальності мережі. Поєднання даних *LF* і *HF* здійснюється за допомогою моделі *(MFNN)*, яка використовується для апроксимації та моделювання залежностей між цими даними. (Рис. 4)

У багатофакторному моделюванні *LF* та *HF* дані взаємозв'язані наступним чином:

$$y_H = \omega(x) y_L + b(x),$$

де y_H – дані низької достовірності *(LF)*;

y_L – дані високої достовірності *(HF)*;

$\omega(x)$ та $b(x)$ – оператори, що апроксимують кореляцію множення та додавання відповідно.

Комбінація операцій множення та додавання може апроксимувати будь-яку лінійну функцію, але в реальності зустрічаються багато нелінійних кореляцій, які виникають внаслідок складних фізичних механізмів. У контексті взаємозв'язків між *LF* і *HF* даними, можна надати більш загальний математичний вираз:

$$y_H = \mathcal{M}(x, y_L),$$

де $\mathcal{M}$ – оператор, який включає як лінійне, так і нелінійне відображення від *LF* до *HF*, може бути розкладений на дві складові: лінійний оператор і нелінійний оператор, як представлено нижче:

$$\mathcal{M} = \mathcal{M}_1 + \mathcal{M}_2$$

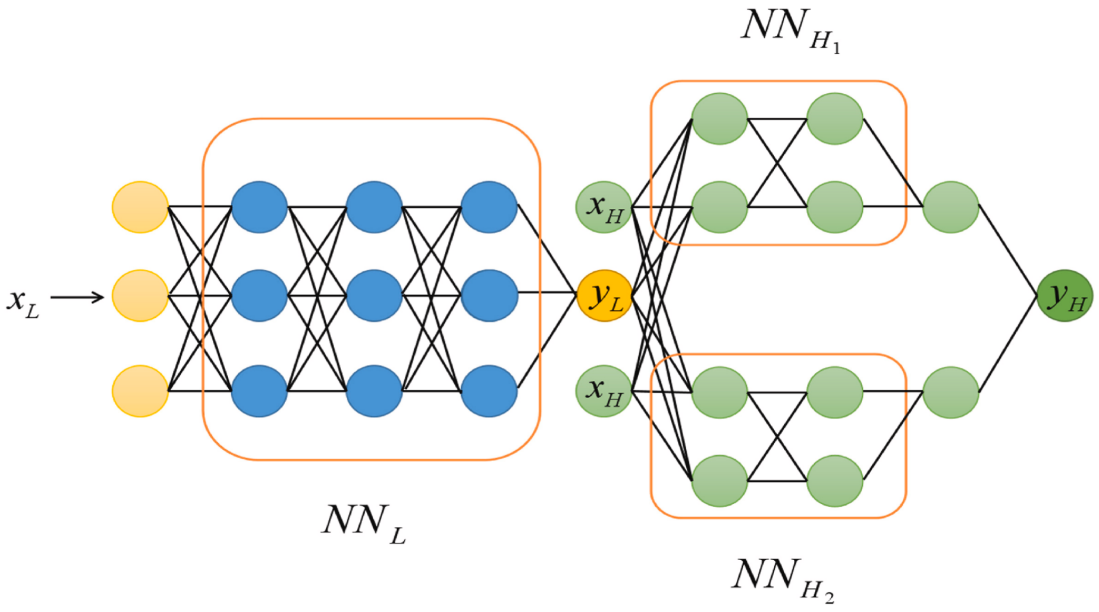

Рис. 4 Схематичне зображення архітектури Multi-fidelity neural network (MFNN), що побудована з урахуванням попереднього рівняння. Ця мережа моделює взаємозв'язки між вхідними і вихідними даними за допомогою різних шарів нейронів, які з'єднані ваговими зв'язками. Вхідні дані проходять через шари нейронів, де вони обробляються та перетворюються до вихідного шару, де формується результат роботи мережі. Зображення показує вхідний шар, приховані шари та вихідний шар, а також зв'язки між нейронами, які вказують напрямок передачі даних.

Один з підходів до використання *MFNN* в контексті диференціальних рівнянь полягає в поєднанні моделей з різними рівнями точності або обчислювальною складністю. Для побудови нейронних мереж можна використовувати різні конфігурації, залежно від потреб задачі. Наприклад, можна створити дві або більше мережі з різною глибиною (кількістю шарів) або шириною (кількістю нейронів у кожному шарі). Одна модель може бути менш складною, але менш точною, тоді як інша модель може бути більш складною, але точнішою.

Ці нейронні мережі можуть бути навчені на різних наборах даних або параметрів. Кожна модель може мати вищу достовірність у своєму відповідному діапазоні параметрів. Залежно від задачі та доступних

ресурсів, можна вибрати оптимальну конфігурацію мережі, яка найкраще задовольнятиме потреби щодо точності та ефективності. [13]

3.3 Специфіка DeepXDE

Особливості бібліотеки *DeepXDE* включають:

I. *Вбудовані примітивні геометрії*. *DeepXDE* має набір основних геометричних примітивів, таких як інтервал, трикутник, прямокутник, полігон, диск, кубоїд і сфера. Вони використовуються для визначення областей, в яких вирішуються диференціальні рівняння.

II. *Конструктивна геометрія твердого тіла (CSG)*. *DeepXDE* підтримує конструктивну геометрію твердого тіла *(CSG)*, що дозволяє створювати складні геометричні форми, поєднуючи примітивні геометрії за допомогою булевих операцій, таких як об'єднання, різниця та перетин. Це дозволяє моделювати складні області для вирішення диференціальних рівнянь.

III. *Підтримка чотирьох типів граничних умов*. умови Діріхле, Неймана, Робіна і періодичні. Крім того, ви можете визначити власні граничні умови за допомогою операторів. Це надає гнучкість при встановленні поведінки системи на межі області.

IV. *Підтримка нейронних мереж*. *DeepXDE* надає два типи нейронних мереж - мережу прямого поширення *(FNN)* і мережу зі зворотним зв'язком *(ResNet)*. Ви можете використовувати ці нейронні мережі для побудови моделей, які апроксимують розв'язки диференціальних рівнянь.

V. *Гнучкість налаштування навчання*. *DeepXDE* надає широкі можливості для налаштування процесу навчання шляхом варіювання різних гіперпараметрів, таких як функції втрат, метрики, оптимізатори, графіки швидкості навчання, ініціалізація та регуляризація. Це дозволяє вам ефективно настроїти процес навчання з метою досягнення оптимальних результатів.

VI. *Розширені можливості*. У *DeepXDE* є додаткові функціональності, які розширюють його можливості. Крім розв'язання диференціальних рівнянь,

він підтримує апроксимацію функцій з використанням багаторівневої точності та вивчення нелінійних операторів. Це дозволяє застосовувати бібліотеку для широкого спектра завдань, що включають не лише диференціальні рівняння.

3.4 Застосування DeepXDE

Алгоритм використання бібліотеки DeepXDE в контексті розв’язання диференціальних рівнянь.

I. За допомогою модуля *geometry* визначається геометрична область обчислень.

II. Зазначується рівняння з частинними похідними *(PDE)* відповідно до синтаксису *TensorFlow*.

III. Встановлюються граничні та початкові умови.

IV. Поєднується геометрію, рівняння з частинними похідними *(PDE)* та граничні/початкові умови в *PDE*-дані або часові дані відповідно. Для визначення навчальних даних вказуються конкретні точки розташування або лише кількість точок, і *DeepXDE* обере потрібну кількість точок на сітці або випадковим чином.

V. Створюється нейронна мережа, використовуючи функціонал модуля *maps*.

VI. Створюється модель, яка комбінує розв'язання рівняння з частинними похідними *(PDE)* на четвертому кроці з використанням нейронної мережі на п'ятому кроці.

VII. Використовується *Model.compile* для налаштування гіперпараметрів оптимізації, таких як вибір оптимізатора та швидкості навчання. Ваги можна встановити за допомогою аргументу *loss_weights*.

VIII. Застосовується *Model.train* для тренування мережі з випадковою ініціалізацією або використанням попередньо навченої моделі за допомогою аргументу *model_restore_path*. За допомогою зворотних викликів *(callbacks)* можна гнучко контролювати та змінювати поведінку навчання.

IX. Використовується *Model.predict* для отримання прогнозу розв'язку рівняння з частинними похідними *(PDE)* в різних точках. (Рис. 5) [14]

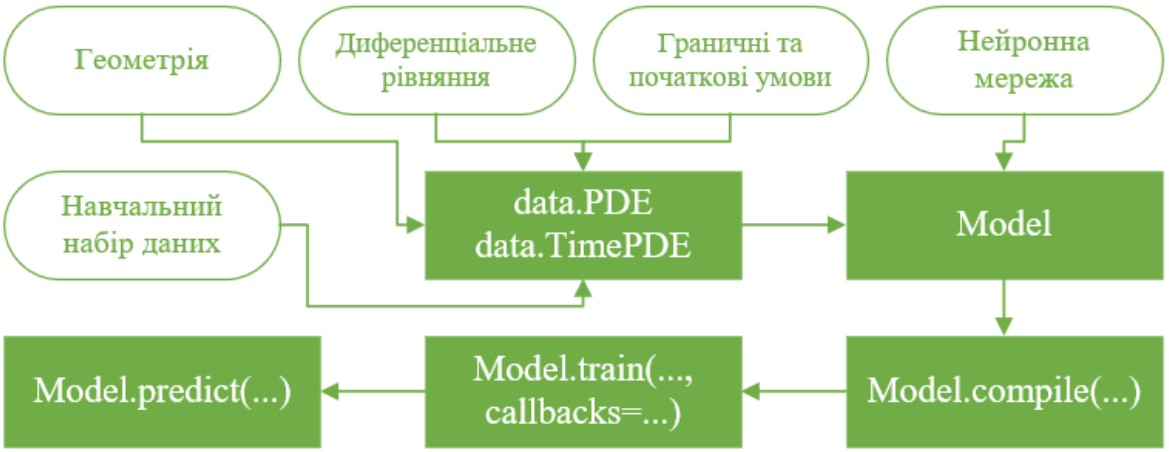

Рис. 5 Візуалізація алгоритму використання бібліотеки *DeepXDE* (Білі прямокутники у .фловчарті представляють *PDE*-проблему (диференціальне рівняння) та гіперпараметри навчання (настройки моделі та оптимізації). Сині прямокутники поєднують *PDE*-проблему з гіперпараметрами навчання, щоб визначити конкретний навчальний сценарій. Помаранчеві прямокутники показують три кроки, які виконуються у порядку зправа наліво для вирішення *PDE* (побудова моделі, тренування моделі та оцінка результатів.)).

РОЗДІЛ 4 МЕТОДИ ОПТИМІЗАЦІЇ НЕЙРОННИХ МЕРЕЖ

4.1 Значення оптимізації нейромережевих алгоритмів

Оптимізація нейронних мереж є фундаментальним і важливим завданням у галузі глибокого навчання. Нейронні мережі - це потужні моделі машинного навчання, здатні адаптуватися і виявляти складні закономірності у даних. Однак, для досягнення оптимальної продуктивності і точності необхідно належним чином налаштувати параметри та ваги нейронної мережі.

Методи оптимізації нейронних мереж надають набір алгоритмів і технік, які допомагають у пошуку оптимальних значень параметрів моделі. Вони спрямовані на вирішення задачі мінімізації функції втрат, яка вимірює різницю між передбаченими значеннями моделі та фактичними цільовими значеннями. Чим менше функція втрат, тим краще працює нейронна мережа.

Складність оптимізації нейронних мереж обумовлена кількома факторами, включаючи велику кількість параметрів, високу розмірність простору пошуку та нелінійність функції втрат. Тому розробка ефективних методів оптимізації є критичним аспектом для забезпечення успішного навчання та застосування нейронних мереж.

Протягом останніх років було запропоновано багато методів оптимізації нейронних мереж. Вони відрізняються своєю структурою, використовуваними алгоритмами та принципами роботи. Деякі з найпоширеніших методів включають стохастичний градієнтний спуск *(SGD)*, адаптивні методи оптимізації, такі як *Adam*, *RMSprop* і *Adagrad*, а також методи на основі другого порядку, такі як методи Левенберга-Марквардта та квазіньютонівські методи.

Кожен метод оптимізації має свої переваги та обмеження, і вибір відповідного методу залежить від конкретної задачі та характеристик даних. Комбінація методів оптимізації з іншими стратегіями, такими як

регуляризація та ініціалізація ваг, може дати кращі результати та підвищити продуктивність нейронних мереж.

4.2 Критерії оптимальності нейромережевих алгоритмів

Оптимальність роботи нейронної мережі залежить від наступних критеріїв:

I. *Висока точність*. Важливо, щоб нейронна мережа демонструвала високу точність у вирішенні завдань класифікації, регресії або сегментації. Точність передбачень в цих завданнях має велике значення, і саме її досягнення є ключовим результатом для нейронної мережі.

II. *Узагальнююча здатність*. Важливою характеристикою для нейронної мережі є її здатність узагальнювати набуті знання на нові, невідомі дані. Це означає, що модель може використовувати отриману інформацію для вирішення реальних ситуацій, з якими вона раніше не зустрічалася. Узагальнююча здатність є ключовим критерієм, який дозволяє нейронній мережі бути ефективною у різних практичних застосуваннях.

III. *Швидкодія*. Висока швидкодія є важливою характеристикою для нейронної мережі, оскільки вона повинна здатися виконувати передбачення та обробку даних з високою швидкістю. Це особливо важливо в ситуаціях, коли потрібна обробка в реальному часі або взаємодія з великим обсягом даних. Швидкодія є критичною для забезпечення ефективності та придатності нейронної мережі у різних практичних сценаріях.

IV. *Стабільність*. Нейронна мережа повинна бути стабільною і надійною при різних умовах та змінах у даних. Вона не повинна бути занадто чутливою до шуму або незначних змін у вхідних даних.

V. *Ефективне використання ресурсів*. Нейронна мережа повинна ефективно використовувати ресурси, такі як пам'ять та обчислювальна потужність. Це особливо важливо для масштабованості та застосування

нейронних мереж на пристроях з обмеженими ресурсами, наприклад, мобільних пристроях або вбудованих системах.

VI. *Розумні вимоги до навчальних даних*. Нейронна мережа повинна мати розумні вимоги до навчальних даних, щоб забезпечити ефективне навчання та передбачення.

Кожен з цих критеріїв є суттєвим для оптимальної роботи нейронної мережі, і їх досягнення може бути складним завданням, яке вимагає вибору відповідної архітектури, методів оптимізації та ретельного налаштування моделі.

4.3 Методи оптимізації нейромережевих алгоритмів

Оптимізація нейромережевих алгоритмів, включаючи використання їх для розв'язання диференціальних рівнянь базованих на бібліотеці *DeepXDE*, передбачає використання декількох методів. Нижче наведено кілька часто використовуваних методів для оптимізації нейронних мереж:

I. Проектування архітектури мережі:

- *зміна глибини та ширини мережі*: варіювання кількість шарів і нейронів на кожному шарі. Глибші мережі з більшою кількістю шарів можуть краще моделювати складні закономірності, тоді як ширші мережі з більшою кількістю нейронів на шару можуть збільшити потужність моделі.
- *варіювання функцій активації*: використання різних функцій активації, для визначення тої, що найкраще підходить для конкретної проблеми.
- *використання методів полегшення потоку градієнтів*: застосування методів, такі як залишкові з'єднання або пропускні

з'єднання, за для полегшення потоку градієнтів під час тренування, що допоможе уникнути проблем з втратою градієнтів та покращить стійкість навчання мережі.

II. Техніки регуляризації:

- *L1 та L2 регуляризація*: додавання штрафу до функції втрат на основі величини ваг допомагає уникнути перенавчання. *L1* регуляризація додає штраф до функції втрат, рівний сумі абсолютних значень ваг моделі, тоді як *L2* регуляризація додає штраф, рівний сумі квадратів ваг. Ці штрафи змушують модель уникати великих значень ваг і сприяють створенню більш узагальнюючих моделей.
- *випадкове відключення (dropout)*: ця техніка випадково деактивує частину нейронів під час кожного кроку тренування. Вона допомагає забезпечити стійкість моделі до варіацій та перенавчання. Під час тренування, кожний нейрон має певну ймовірність бути відключеним, тим самим змушуючи мережу використовувати різні комбінації нейронів і зменшуючи залежність ваг один від одного.
- *термінація за раннім зупиненням (early stopping)*: ця техніка використовує моніторинг продуктивності моделі на валідаційному наборі. Тренування зупиняється, коли продуктивність моделі на валідаційному наборі починає погіршуватися, що допомагає уникнути перенавчання. Застосування раннього зупинення дозволяє зберегти модель, яка

досягла найкращих результатів на валідаційному наборі до цього моменту.

III. Використання алгоритмів оптимізації:

- *стохастичний градієнтний спуск (SGD)*: оновлення ваг моделі на основі обчислених градієнтів на невеликих пакетах тренувальних даних.
- *адаптивні алгоритми*: методи, такі як *Adam*, *AdaGrad* або *RMSprop*, адаптують швидкість навчання для кожного параметра на основі попередніх градієнтів, щоб прискорити збіжність.

IV. Керування швидкістю навчання:

- *зменшення швидкості навчання з часом*: цей підхід полягає в поступовому зменшенні швидкості навчання протягом тренування. Наприклад, можна використовувати змінну швидкість навчання, яка поступово зменшується з кожним епохою або пакетом тренувальних даних. Це допомагає моделі збігатися до більш стабільного розв'язку.
- *циклічні швидкості навчання*: цей підхід включає зміну швидкості навчання в циклічний спосіб. Замість поступового зменшення, швидкість навчання періодично змінюється між межами, що дозволяє ефективніше досліджувати простір розв'язків. Наприклад, можна використовувати техніку "циклічного розкладу швидкості навчання", де швидкість навчання змінюється між мінімальним і максимальним значеннями протягом певного періоду.

- *розгортання швидкості навчання*: цей підхід використовує різні значення швидкості навчання для різних шарів або параметрів моделі. Наприклад, можна використовувати більш високу швидкість навчання для шарів ближче до входу моделі, а меншу швидкість для шарів ближче до виходу. Це допомагає краще налаштувати різні рівні абстракції у моделі.

V. ***Нормалізація за пакетами (Batch normalization).*** Нормалізація функцій активації кожного шару всередині пакета, для поліпшення стабільності тренування та прискорення збіжності моделі.

VI. ***Аугментація даних.*** Збільшення розміру тренувального набору, застосовуючи випадкові перетворення до вхідних даних (наприклад, обертання, масштабування або відображення), щоб допомогти моделі краще узагальнювати.

VII. ***Оптимізація гіперпараметрів.*** Експерименти з різними значеннями гіперпараметрів, такими як швидкість навчання, розмір пакету, сила регуляризації та архітектура мережі. Оптимізація гіперпараметрів відіграє важливу роль у покращенні продуктивності моделі та її здатності до узагальнення.

РОЗДІЛ 5 ПРОГРАМНА РЕАЛІЗАЦІЯ

5.1 Постановка задачі

У даній кваліфікаційній роботі було обрано кубічне нелінійне рівняння Шредінгера як основний об'єкт дослідження та оптимізації нейромережевого алгоритму. Рівняння Шредінгера є фундаментальним рівнянням квантової механіки, яке використовується для опису поведінки квантових систем, таких як електрони, атоми та молекули.

Застосування нейромережевих алгоритмів для оптимізації розв'язку рівняння Шредінгера є актуальним завданням, оскільки це відкриває нові можливості для покращення точності та швидкості розв'язання. Нейромережі проявляють потужність у роботі зі складними даними та можуть автоматично виявляти складні залежності, що виникають у процесі розв'язання рівняння Шредінгера. Використання нейромережевих алгоритмів у поєднанні з бібліотекою *DeepXDE* надає можливість покращити ефективність розв'язання рівняння Шредінгера та розширити його застосування на практиці.

5.1.1 Опис рівняння Шредінгера та його використання в різних фізичних задачах

Рівняння Шредінгера є фундаментальним математичним виразом квантової механіки, що дозволяє описувати поведінку квантових систем на мікроскопічному рівні, таких як електрони, атоми, молекули та інші частинки.

Математичну форму даного рівняння, а саме його кубічно-нелінійну варіацію, можна подати наступним чином:

$$iu_t + u_{xx} + q|u|^2 u = 0;\ L_0 \le x \le L_1\ ,$$

де $i = \sqrt{-1}$,

L_0 та L_1 – граничні умови для змінної x;

$q \geq 0 -$ дійсний параметр;

$$u = u(x,\ t) = u(x,\ t)\ + i\omega(x,\ t),$$

з початковою умовою:

$$u(x,\ t\ =\ 0)\ =\ g(x)\ =\ g_R(x)\ +\ ig_I(x);\ L_0\ \leq\ x\ \leq\ L_1\ ,$$

в якому $g_R(x)$ та $g_I(x)$ – визначені дійсні неперервні функції з граничними умовами:

$$\frac{\partial u(L_0,\ t)}{\partial x}\ = \frac{\partial u(L_1,\ t)}{\partial x}\ = 0;\ t \geq t_0$$

Рівняння Шредінгера виражає залежність хвильової функції, яка математично описує квантовий стан системи, від часу і тривимірних просторових координат. Це рівняння дає можливість прогнозувати ймовірнісні результати вимірювань квантових величин та розуміти розподіл енергій та інших властивостей системи. Шляхом розв'язання рівняння Шредінгера, можна отримати хвильові функції, які описують еволюцію системи у часі та просторі, і тим самим забезпечують фундаментальне розуміння квантової природи матерії. [15]

5.2 Розробка нейромережевого алгоритму для вирішення рівняння Шредінгера на базі бібліотеки DeepXDE

Для вирішення кубічного нелінійного рівняння Шредінгера (з параметром 0.6) в даній кваліфікаційній роботі було використано бібліотеку *DeepXDE*, яка надає потужні інструменти для розв'язування рівнянь з використанням глибокого навчання.

5.2.1 Розробка архітектури нейромережі

В даній програмній реалізації було використано повнозв’язну архітектуру нейронної мережі *(FNN)*. Ця архітектура передбачає послідовне розміщення шарів, в яких кожен нейрон пов'язаний з кожним нейроном попереднього шару.

5.2.2 Структура обраної нейромережі

Вищезазначена нейромережа має наступну структуру:

I. Вхідний шар: складається з двох нейронів, які представляють вхідні змінні *x* та *t*.
II. Приховані шари: ця нейромережа має чотири повнозв'язаних шари, кожен з яких містить *100* нейронів. У цих шарах використовується гіперболічна тангенс (*tanh*) як функція активації. Також встановлюється функція ініціалізації ваг моделі (*Glorot normal*).
III. Вихідний шар: складається з двох нейронів, що представляють вихідні змінні *u* та *v*.

5.2.3 Встановлення гіперпараметрів моделі

У цьому коді гіперпараметри моделі встановлені наступним чином:

I. Швидкість навчання (learning rate): встановлена на значення *1e-3* у функції *model.compile()*.
II. Кількість ітерацій навчання: встановлена на значення *10000* у функції *model.train()*.

5.2.4 Процес реалізації

Процес реалізації даного алгоритму можна поділити на такі основні етапи:

I. Визначення області та сітки: були задані границі області та створена сітка значень для просторових і часових точок, що дозволить в подальшому візуалізувати отримані результати на даній сітці.
II. Визначення геометрії: в реалізації був використаний геометричний об'єкт, який дозволив описати просторову і часову області, в яких розв'язувалося рівняння рівняння Шредінгера.

Цей геометричний об'єкт використовується для встановлення граничних умов і обмежень на розв'язок.

III. Визначення рівняння: створена функція, яка описувала праву частину рівняння Шредінгера. В цій функції використовуються оператори диференціювання для обчислення похідних першого і другого порядку.

IV. Визначення граничних та початкових умов: були задані граничні та початкові умови для компонентів, за допомогою яких враховуються фізичні обмеження на систему.

V. Створення об'єкта даних: використовується клас *TimePDE*, що надається бібліотекою *DeepXDE*, для створення об'єкта даних. Цей об'єкт містить інформацію про геометрію, рівняння та умови, і використовується для тренування моделі.

VI. Визначення моделі: застосовується нейронна мережа з використанням бібліотеки *DeepXDE*, що складається з шару *FNN* та декількома прихованими шарами.

VII. Тренування моделі: модель тренувалась з використанням оптимізатора *Adam* і налаштуваннями параметрів навчання, на базі даних, які містили інформацію про граничні, початкові умови та рівняння.

VIII. Прогнозування: прогнозування значень розв'язку на всій заданій області було візуалізовано за допомогою графіків.

5.3 Оцінка та аналіз отриманих результатів

5.3.1 Використання метрики оцінки моделі

Для оцінки ефективності та якості моделі розв’язання рівняння Шредінгера використовувалася функція втрат (*loss function*). Ця функція слугує метрикою для вимірювання рівня помилок, здійснюваних моделлю під час передбачення на тренувальних та тестових даних. Втрати можуть бути

визначені різними способами, враховуючи поставлену задачу та властивості вхідних даних.

Шляхом порівняльного аналізу значень функції втрат, проводилося дослідження різних параметрів та конфігурацій моделі з метою визначення оптимальних налаштувань. Це дозволило здійснювати обґрунтовані рішення стосовно подальшого поліпшення моделі та досягнення більш точних передбачень.

5.3.2 Візуалізація результатів

Візуальне представлення рівняння Шредінгера. (Рис. 6)

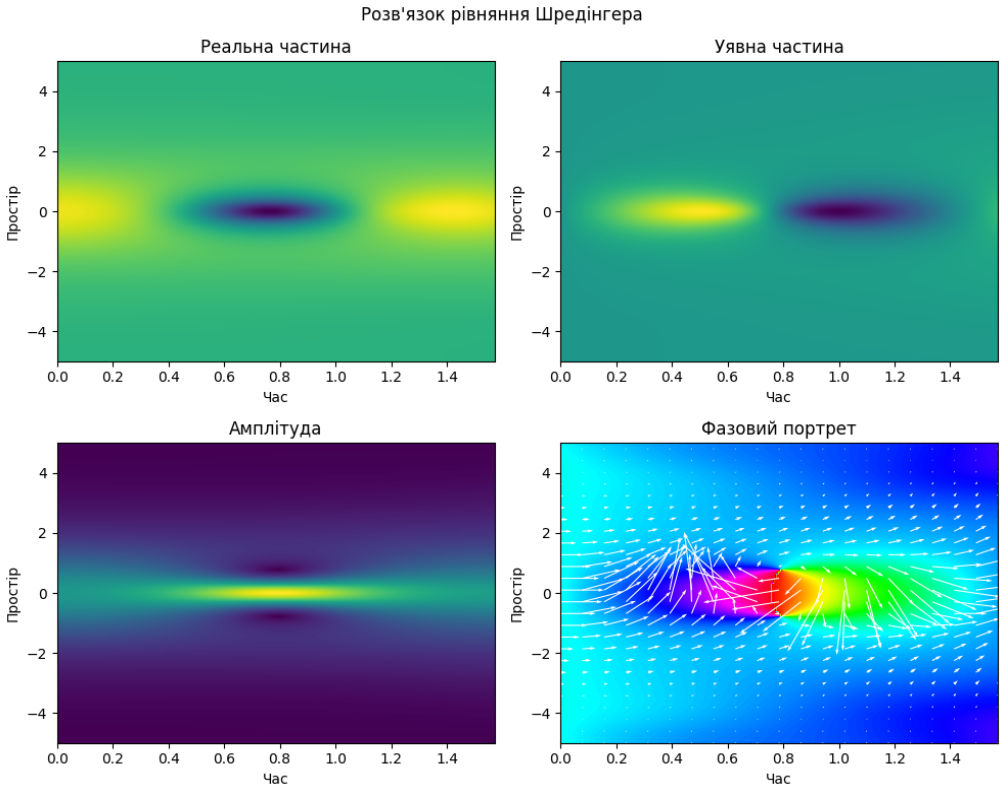

Рис. 6

Опис зображених графіків:

I. Графік *"Real Part"* (Реальна частина). Цей графік відображає розподіл реальної складової хвильової функції (фізична амплітуда коливання) в залежності від просторових і часових змін.

II. Графік *"Imaginary Part"* (Уявна частина). Цей графік відображає розподіл уявної складової хвильової функції (фази хвилі, що вказують на зміщення або зсув коливання вгору або вниз) в залежності від просторових і часових змін.

III. Графік *"Amplitude"* (Амплітуда). Цей графік відображає розподіл амплітуди хвильової функції (ймовірність знаходження частинки чи системи в певному стані) в залежності від просторових і часових змін. Він показує величину амплітуди хвильової функції в кожній точці простору і часу.

IV. Графік *"Phase"* (Фаза). Цей графік відображає розподіл фазового кута хвильової функції (положення хвилі у своєму коливанні відносно початкового моменту в часі) в залежності від просторових і часових змін.

У даному випадку буде доцільним термін “флуктуація”, що відображається як зміна значень хвильової функції в різних точках простору та часу. За допомогою кольорового кодування фази, графік може відображати ці флуктуації, демонструючи зміни кольорів або відтінків на різних ділянках графіку. (Рис. 7)

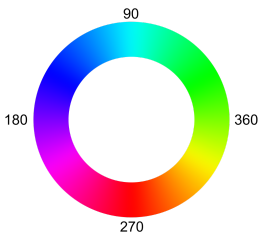

Рис. 7

5.4 Оптимізація моделі

5.4.1 Застосування алгоритмів оптимізації

На початку були проведені попередні експерименти з застосуванням різних методів оптимізації, таких як: *Adam*, *AdaGrad*, *SGD*, *RMSprop*, – з метою отримання базового рівня продуктивності та якості моделі.

Далі наведена таблиця з втратами моделі з різними методами оптимізації, на базі яких було проведене порівняння. (Таблиця 1)

Таблиця 1.

Метод оптимізації	Adam	AdaGrad	RMSprop	SGD
Втрати	0,00239	0,0667	0,0373	0,0858

Алгоритм *Adam*, на основі наведених даних, показав найнижчі втрати *(0,00268)*, що свідчить про його ефективність у зменшенні помилок моделі. Таким чином, використання *Adam* дозволить досягти кращої точності прогнозування.

5.4.2 Застосування оптимізатора LBFGC

Був застосований оптимізатор *LBFGS* до нейромережі з метою оптимізації моделі і покращення її результатів. Перед застосуванням *LBFGS*, втрати моделі становили *(0,00239)*. Проте, після застосування оптимізатора *LBFGS*, втрати зменшилися до значення *(0,00000742)*. Це покращення втрат є значним і свідчить про успішну роботу оптимізатора *LBFGS* у пошуку оптимальних параметрів моделі.

Отримані результати свідчать про те, що *LBFGS* успішно знайшов локальний мінімум цільової функції, що відповідає найменшим втратам. Це підтверджує його здатність досягати високої точності і поліпшувати результати моделі.

З урахуванням цих результатів, можна зробити висновок, що використання оптимізатора *LBFGS* є обґрунтованим і раціональним вибором для подальшої оптимізації та покращення результатів моделі.

5.4.3 Вибір оптимальної швидкості навчання

Було проведено дослідження з метою вибору оптимальної швидкості навчання *(learning rate)* для нейронної мережі. Для цього було проведено експеримент з трьома різними значеннями *learning rate*: 10^{-3}, 10^{-2} та 10^{-4}.

Далі наведена порівняльна таблиця втрат для різних швидкостей навчання моделі. (Таблиця 2)

Таблиця 2.

Швидкість навчання	10^{-3}	10^{-2}	10^{-4}
Втрати	0,00000742	0,00000731	0,00000965

Отже, на основі результатів, значення *learning rate* 10^{-2} показало найкращі результати втрат на тестовому наборі даних, що свідчить про його ефективність у покращенні якості моделі.

5.4.4 Варіювання функцій активації

Було проведено порівняння різних варіацій функцій активації, таких як: *tanh*, *RELU*, *sigmoid*, *SELU*. Далі наведена таблиця з втратами, на основі яких було проведене порівняння застосування функцій активації. (Таблиця 3)

Таблиця 3.

Функції активації	tanh	sigmoid	ReLU	SeLU
Втрати	0,00000731	0,0000195	0,124	0,0567

За отриманими результатами можна зробити наступний висновок щодо варіацій функцій активації:

- функції активації *tanh* та *sigmoid* показали найкращі результати, де втрати на тестовому наборі даних складали *(0.00000731)* та (*0.0000195)* відповідно. Ці функції активації забезпечили найнижчі значення втрат, що свідчить про їх ефективність у моделюванні складних залежностей між вхідними та вихідними даними.
- функція активації *ReLU* показала гірші результати, де втрати становили *(0.124)*. Це може свідчити про проблему виникнення "мертвих" нейронів у моделі, де деякі нейрони неактивні на більшій частині даних.
- функція активації *SeLU* також показала досить високі значення втрат, де вони складали *(0.0567)*. Це може вказувати на проблему з розподілом значень у нейронній мережі, де відсутній необхідний баланс між активаціями.

Отже, на основі отриманих результатів можна зробити висновок, що використання функції активації *tanh* є найкращим варіантом для досягнення оптимальних результатів у моделюванні нейронної мережі в даному контексті.

5.4.5 Архітектурна модифікація

Було проведено порівняльний аналіз моделей з різною кількістю шарів та нейронів. Проведено серію експериментів з різними конфігураціями моделей, з метою визначення оптимальної архітектури, яка забезпечує найкращу точність прогнозування та ефективність моделі. Далі представлені результати. (Таблиця 4)

Таблиця 4.

Нейрони /Шари	100/3	100/4	100/5	150/3	150/4	150/5	200/3	200/4
Втрати	0,00000731	0,00000772	0,00000611	0,0000207	0,0000864	0,0000114	0,0000101	0,0000982

На основі порівняльного аналізу моделей з різною кількістю шарів та нейронів найкращий результат досягнуто моделлю з *100* нейронами та *5* шарами, де втрати складають *(0,00000611)*. Ця конфігурація моделі демонструє найкращу точність прогнозування та ефективність серед усіх розглянутих варіантів.

5.4.6 Вибір методу ініціалізації шарів

Було проведено порівняльний аналіз різних методів ініціалізації шарів для заданої моделі з метою вибору оптимального підходу. Нижче наведена таблиця з результатами експерименту. (Таблиця 5)

Таблиця 5.

Метод ініціалізації шарів	Glorot normal	Glorot uniform	He normal	Le Cun normal
Втрати	0,00000611	0,00000517	0,0000119	0,0000107

На основі результатів порівняльного аналізу різних методів ініціалізації шарів для заданої моделі, можна зробити наступний висновок. Метод ініціалізації *"Glorot uniform"* показав найкращі результати з найнижчим значенням втрат *(0,00000517)*. Це може бути пов'язано з тим, що *"Glorot uniform"* розподіляє ваги шарів з урахуванням розмірів вхідних та вихідних шарів, сприяючи стабільному навчанню моделі. Інші методи ініціалізації, такі як *"Glorot normal"*, *"He normal"* та *"Le Cun normal"*, показали вищі значення втрат, що може вказувати на меншу ефективність при навчанні моделі.

5.4.7 Підбір оптимальної кількості ітерацій

Було проведено дослідження з метою оцінки впливу кількості ітерацій на ефективність нейромережевого алгоритму. Для цього були виконані

експерименти з різними значеннями кількості ітерацій, і були зібрані дані про показники втрат моделі.

В результаті аналізу експериментальних даних встановлено, що зі збільшенням кількості ітерацій алгоритм показує кращі результати. Це означає, що більша кількість ітерацій дозволяє нейромережевому алгоритму краще виявляти та апроксимувати складні залежності в даних. Зменшення значення функції втрат та підвищення точності прогнозування є показниками покращення роботи моделі при більшій кількості ітерацій.

Отже, на основі отриманих результатів можна зробити висновок, що збільшення кількості ітерацій є важливим фактором для поліпшення ефективності нейромережевого алгоритму, дозволяючи досягти кращих результатів у прогнозуванні та апроксимації даних.

5.4.8 Інші методи оптимізації

У ході дослідження були використані додаткові підходи, такі як *L1*-регуляризація та рання зупинка *(early stopping)*, з метою покращення ефективності моделі. Проте, після проведення експериментів було виявлено, що застосування даних методів не принесло позитивних результатів, а навіть призвело до зниження точності моделі. Далі представлені результати. (Таблиця 5, Таблиця 6)

Таблиця 5.

L1-регуляризація (параметр)	0,1	0,2	0,3
Втрати	0,00000495	0,00000966	0,0000154

Таблиця 6.

Рання зупинка (ітерації)	1000	800	600

Втрати	0,0000095	0,00000638	0,0000395

У даному контексті, застосування даних методів оптимізації не виявилося доцільним для моделі та поставленої задачі.

Враховуючи результати експериментів, було вирішено відмовитися від використання цих методів оптимізації у подальшому навчанні моделі. Це дозволило зосередитися на інших аспектах оптимізації та підвищенні ефективності моделі, що в результаті призвело до отримання кращих результатів в цілому.

5.5 Аналіз результатів

На основі проведених експериментів та порівняння різних методів оптимізації, функцій активації та регуляризації можна зробити наступні висновки:

I. Алгоритм оптимізації *Adam* показав найкращі результати у порівнянні з *SGD*, *AdaGrad* та *RMSprop* для даної нейромережевої моделі. Він мав найменші значення втрат, що свідчить про його оптимальність для покращення продуктивності та точності моделі.

II. Використання оптимізатора *LBFGS* у даному дослідженні дало значні результати щодо покращення моделі нейромережі. Початкові втрати моделі становили *(0,00239)*, проте після застосування оптимізатора *LBFGS* вони зменшилися до надзвичайно низького значення *(0,00000742)*. Це доволі значне покращення втрат, що свідчить про успішну роботу оптимізатора у пошуку оптимальних параметрів моделі.

III. На основі порівняння втрат для різних значень швидкості навчання *(learning rate)* можна зробити висновок, що значення 10^{-2} є оптимальним для даної нейронної мережі. При

використанні цього значення *learning rate*, модель досягла найнижчих втрат, що свідчить про ефективність цього значення у покращенні якості моделі.

IV. На основі отриманих результатів, найкращу функцію активації для даного контексту можна визначити як *tanh*. Ця функція активації показала найменші значення втрат на тестовому наборі даних *(0.00000731)*, що свідчить про її ефективність у моделюванні складних залежностей між вхідними та вихідними даними.

V. На основі результатів порівняльного аналізу моделей з різною кількістю шарів та нейронів, можна зробити висновок, що оптимальна архітектура моделі полягає у використанні *5* шарів із *100* нейронами кожен. Ця конфігурація моделі показала найкращі результати з найменшими втратами *(0,00000611)*.

VI. Метод ініціалізації ваг *"Glorot uniform"* показав найкращі результати з найнижчими втратами *(0,00000517)*. В порівнянні з методами *"Glorot normal"*, *"He normal"* та *"Le Cun normal"*, *"Glorot uniform"* виявився більш ефективним і призвів до кращої точності моделі. Його перевага полягає у розподілі ваг шарів, що забезпечує стабільне навчання та поліпшення результатів прогнозування.

VII. Збільшення кількості ітерацій є важливим фактором для поліпшення ефективності нейромережевого алгоритму, дозволяючи досягти кращих результатів у прогнозуванні та апроксимації даних.

VIII. У ході дослідження *L1*-регуляризації та ранньої зупинки *(early stopping)* було виявлено, що ці методи не покращують ефективність моделі, а навіть можуть призвести до зниження точності. *L1*-регуляризація з різними значеннями параметра не змогла ефективно контролювати перенавчання моделі, тоді як

рання зупинка з різними значеннями кількості ітерацій призвела до вищих втрат порівняно зі стандартною моделлю без ранньої зупинки.

5.6 Оцінка ефективності нейромережевого та чисельного методів у розв'язанні диференціального рівняння

5.6.1 Реалізація чисельного методу

Для порівняння ефективності нейромережевого підходу з чисельним методом для вирішення рівняння Шредінгера було реалізовано чисельний алгоритм на базі потужного інструменту *Wolfram*.

В наведеному нижче коді ми використовуємо функцію *NDSolveValue* для чисельного розв'язання диференціального рівняння Шредінгера з заданими граничними умовами. Рівняння моделює динаміку хвилі в квантовій системі з певним потенціалом.

```
In[88]:= Clear["Global`*"]
T = Pi / 2;
BCS = PeriodicBoundaryCondition[u[t, x], x == -5, TranslationTransform[{10}]];
fun = Check[NDSolveValue[
    {I * D[u[t, x], t] + D[u[t, x], {x, 2}] + 0.6 * Abs[u[t, x]]^2 * u[t, x] == 0,
      BCS, u[0, x] == 2 * Sech[x]}, u, {t, 0, T}, {x, -5, 5},
    AccuracyGoal → 10, PrecisionGoal → 10, MaxStepSize → 0.01
  ], $Failed]

If[fun === $Failed,
  Print["Error: ", $Failed],
  initialNorm = NIntegrate[Abs[fun[0, x]]^2, {x, -5, 5}];
  finalNorm = NIntegrate[Abs[fun[T, x]]^2, {x, -5, 5}];
  loss = initialNorm - finalNorm;
  Print["Втрати: ", loss];
]
```

Рис.8

У даній реалізації на початку визначається часовий проміжок *T* та граничну умову *BCS* для періодичної системи. Далі, за допомогою функції

NDSolveValue, ми розв'язуємо рівняння Шредінгера з використанням чисельного методу. Результат розв'язку зберігається в змінній *fun*.

Після чисельного розв'язку ми обчислюємо норми хвилі на початку та в кінці часового проміжку, використовуючи функцію *NIntegrate*. Ці значення дозволяють нам порівняти початкову та кінцеву норми, що характеризують зміну в розв'язку хвилі протягом заданого часу. Крім того, обчислюється втрата як різниця між початковою та кінцевою нормами. В даному випадку втрати становлять *(0,0000000917)*.

Далі представлена візуалізація розв’язку рівняння Шредінгера з використанням *Wolfram*. (Рис. 9)

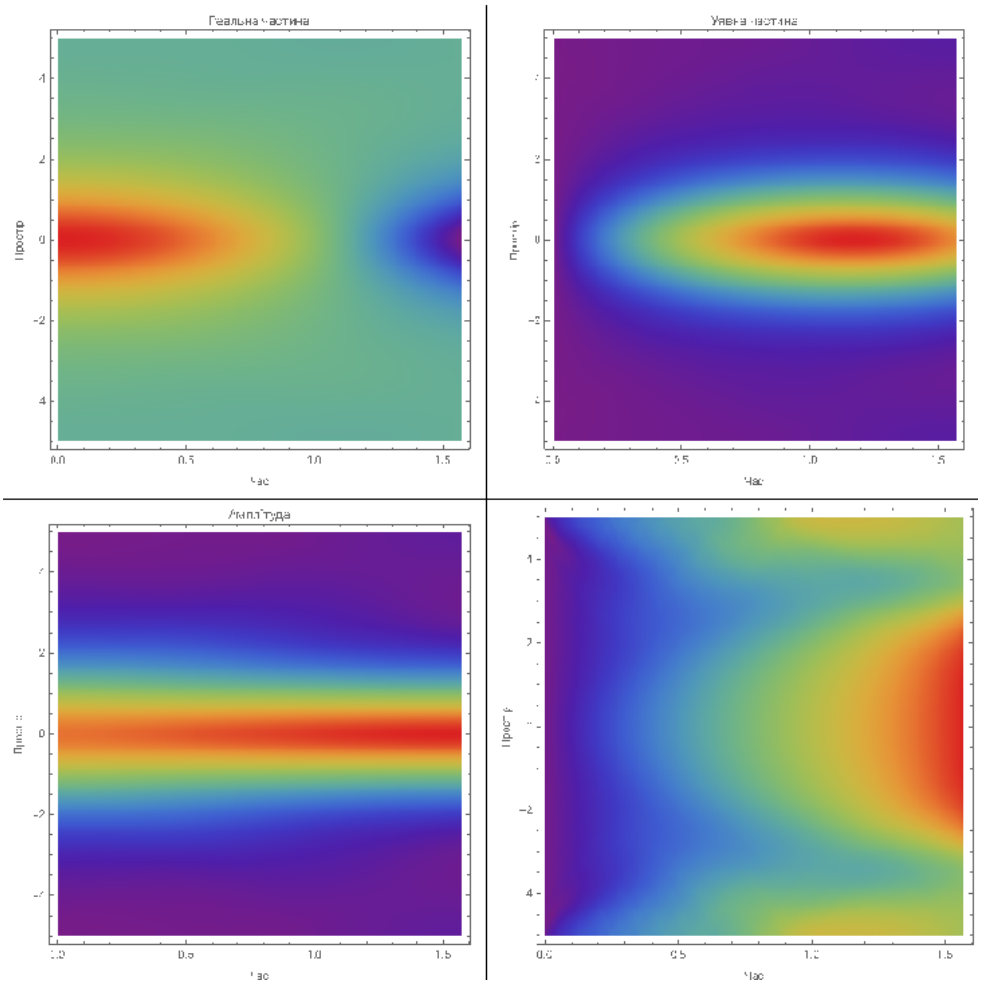

Рис. 9

5.6.2 Порівняння отриманих результатів

На основі отриманих даних, в яких втрата норми для нейромережевого алгоритму на базі бібліотеки *DeepXDE* складає *(0,00000517)*, а для чисельного методу на базі *Wolfram – (0,0000000917)*, можна зробити наступний висновок на користь нейромережевого підходу:

I. Менші значення втрат. Втрата представляє різницю між початковою та кінцевою нормами розв'язку диференціального рівняння. За отриманими даними, нейромережевий алгоритм на базі *DeepXDE* має втрату *(0,00000517)*, що значно більше ніж втрата чисельного методу на базі *Wolfram (0,0000000917)*. Це свідчить про більшу точність та стабільність розв'язку, що отримано за допомогою чисельного методу.

II. Гнучкість та адаптивність. Нейромережеві алгоритми здатні адаптуватися до різноманітних видів диференціальних рівнянь та задач, завдяки своїй гнучкості та здатності до навчання. Вони можуть розв'язувати складні диференціальні рівняння з великою кількістю змінних та невідомих параметрів. Таким чином, нейромережевий алгоритм може бути більш ефективним у вирішенні складних диференціальних рівнянь, де чисельні методи можуть зіштовхнутися з обмеженнями.

III. Швидкість розв'язання є важливим фактором у порівнянні нейромережевого алгоритму на базі *DeepXDE* та чисельного методу на базі *Wolfram*. Нейромережеві алгоритми, при належній конфігурації та оптимізації, можуть досягати високої швидкості розв'язання диференціальних рівнянь. Це особливо корисно в ситуаціях, де необхідно обробляти великий обсяг даних або виконувати багато ітерацій.

У перспективі, зі збільшенням кількості ітерацій, нейромережевий алгоритм може досягати майже ідеальних результатів. Проте, для досягнення цієї високої точності та ефективності, необхідна потужна

обчислювальна здатність. Розрахункові ресурси, такі як обчислювальна потужність процесорів та доступ до великих обчислювальних кластерів або хмар, можуть бути вирішальними факторами для успішного застосування нейромережевого підходу в складних задачах диференціальних рівнянь.

Загалом, на основі вищезазначених факторів, можна зробити висновок, що нейромережевий алгоритм на базі бібліотеки *DeepXDE* може бути ефективним та потенційно кращим у вирішенні диференціальних рівнянь порівняно з чисельним методом на базі *Wolfram*.

ВИСНОВКИ

В ході дослідження було розглянуто теоретичні основи диференціальних рівнянь, нейронних мереж та бібліотеки *DeepXDE* для їх вирішення. Було проведений аналіз застосування бібліотеки *DeepXDE* у розв'язанні диференціальних рівнянь і виявлено її потенціал для вирішення складних задач.

Оптимізація нейромережевих алгоритмів стала ключовим аспектом дослідження. Було розглянуто різні методи оптимізації з метою поліпшення ефективності та точності розв'язання диференціальних рівнянь.

Результати роботи були втілені у програмну реалізацію нейромережевого алгоритму для вирішення рівняння Шредінгера на базі бібліотеки *DeepXDE*. Отримані результати були піддані оцінці та аналізу, що підтвердило ефективність та точність розробленої моделі. Застосування оптимізаційних підходів дало змогу покращити продуктивність та швидкість розв'язання рівняння Шредінгера.

У ході роботи було проведено порівняння чисельного методу, реалізованого за допомогою інструменту *WolfrDeepXDEam*, та нейромережевого алгоритму, на базі, для вирішення рівняння Шредінгера, і проведено їх аналіз.

Узагальнюючи, оптимізація нейромережевих алгоритмів є важливим напрямком досліджень, оскільки вона дозволяє покращити швидкість збіжності моделей, зменшити кількість параметрів та ефективно використовувати обчислювальні ресурси. Це дає змогу зробити розв'язання диференціальних рівнянь більш доступним та придатним для реалізації в різних практичних задачах.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

[1]. Leech J. S. Differential Equations. [Електронний ресурс] / J. S. Leech, R. C. Yates // The american mathematical monthly. – 1953. – Т. 60, № 3. – С. 202. – Режим доступу до ресурсу: https://doi.org/10.2307/2307594

[2]. Wanxie Z. Method of separation of variables and Hamiltonian system [Електронний ресурс] / Zhong Wanxie, Zhong Xiangxiang // Numerical methods for partial differential equations. – 1993. – Т. 9, № 1. – С. 63–75. – Режим доступу до ресурсу: https://doi.org/10.1002/num.1690090107

[3]. Mejlbro L. Solution of linear ordinary differential equations by means of the method of variation of arbitrary constants [Електронний ресурс] / Leif Mejlbro // International journal of mathematical education in science and technology. – 1997. – Т. 28, № 3. – С. 321–331. – Режим доступу до ресурсу: https://doi.org/10.1080/0020739970280302

[4]. Recurrent neural networks [Електронний ресурс] / ред.: L. Medsker, L. C. Jain. – [Б. м.] : CRC Press, 1999. – Режим доступу: https://doi.org/10.1201/9781420049176

[5]. Components of an Artificial Neural Network [Електронний ресурс] – Режим доступу до ресурсу: https://www.enjoyalgorithms.com/blog/components-of-ann

[6]. Feedforward neural network [Електронний ресурс] – Режим доступу до ресурсу: https://deepai.org/machine-learning-glossary-and-terms/feed-forward-neural-network

[7]. Recurrent neural networks [Електронний ресурс] / ред.: L. Medsker, L. C. Jain. – [Б. м.] : CRC Press, 1999. – Режим доступу до ресурсу: https://doi.org/10.1201/9781420049176

[8]. Recurrent neural networks [Електронний ресурс] / ред.: L. Medsker, L. C. Jain. – [Б. м.] : CRC Press, 1999. – Режим доступу до ресурсу: https://doi.org/10.1201/9781420049176

[9]. Recurrent neural networks [Електронний ресурс] / ред.: L. Medsker, L. C. Jain. – [Б. м.] : CRC Press, 1999. – Режим доступу до ресурсу: https://doi.org/10.1201/9781420049176

[10]. A Tour of Attention-Based Architectures [Електронний ресурс] – Режим доступу до ресурсу: https://machinelearningmastery.com/a-tour-of-attention-based-architectures/

[11]. Physics-informed neural networks [Електронний ресурс] – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Physics-informed_neural_networks

[12]. Learning nonlinear operators via DeepONet based on the universal approximation theorem of operators [Електронний ресурс] / Lu Lu [та ін.] // Nature machine intelligence. – 2021. – Т. 3, № 3. – С. 218–229. – Режим доступу: https://doi.org/10.1038/s42256-021-00302-5

[13]. Chakraborty S. Transfer learning based multi-fidelity physics informed deep neural network [Електронний ресурс] / Souvik Chakraborty // Journal of computational physics. – 2021. – Т. 426. – С. 109942. – Режим доступу: https://doi.org/10.1016/j.jcp.2020.109942

[14]. Li S. Efficient regional seismic risk assessment via deep generative learning of surrogate models [Електронний ресурс] / Shanwu Li, Charles Farrar, Yongchao Yang // Earthquake Engineering & Structural Dynamics. – 2023. – Режим доступу: https://doi.org/10.1002/eqe.3849

[15]. The cubic nonlinear Schrödinger equation [Електронний ресурс] – Режим доступу до ресурсу:

https://www.sciencedirect.com/science/article/pii/S0377042706004717

[16]. Посилання на Github [Електронний ресурс] – Режим доступу до ресурсу: https://github.com/VladaChernoray/diploma_schredinger/blob/main/schredinger.ipynb