КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ТАРАСА ШЕВЧЕНКА
Факультет комп'ютерних наук та кібернетики
Кафедра математичної інформатики

Кваліфікаційна робота
на здобуття ступеня бакалавра
за спеціальністю 122 Комп'ютерні науки
на тему:

РОЗВ'ЯЗАННЯ ДИФЕРЕНЦІАЛЬНИХ РІВНЯНЬ У ЧАСТИННИХ ПОХІДНИХ ГЛИБОКИМ МЕТОДОМ ГАЛЬОРКІНА

Виконав студент 4 курсу
Бендес Андрій Юрійович _____
(підпис)

Науковий керівник:
професор, доктор фіз.-мат. наук
Вергунова Ірина Миколаївна

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

Студент _____
(підпис)

Роботу розглянуто й допущено до захисту
на засіданні кафедри математичної інформатики
«____» _____ 202_ р.,
протокол № ___
Завідувач кафедри
В. М. Терещенко _____
(підпис)

Київ – 2022

РЕФЕРАТ

Обсяг роботи 40 сторінок, 14 ілюстрацій, 16 джерел посилань.

ДИФЕРЕНЦІАЛЬНЕ РІВНЯННЯ, ЧАСТКОВІ ПОХІДНІ, НЕЙРОННА МЕРЕЖА, ГЛИБОКЕ НАВЧАННЯ, ГЛИБОКИЙ МЕТОД ГАЛЬОРКІНА.

Об’єктом роботи є аналіз розв’язування диференціальних рівнянь з частковими похідними. Предметом роботи є програмний засіб для розв’язування диференціальних рівнянь з частковими похідними.

Метою роботи є створення програмного забезпечення для розв’язування задач з використанням рівнянь математичної фізики.

Методи розробки: комп’ютерне моделювання, методи побудови і навчання нейронних мереж. Інструменти розробки: середовище розробки Jupyter Notebook, мова програмування Python, фреймворк PyTorch, бібліотека Numpy.

Результати обробки: виконано огляд підходів, які використовуються для вирішення диференціальних рівнянь з частковими похідними, проаналізовано переваги кожного з цих методів, запропоновано власне рішення цієї задачі, розроблена програма, розв’язує дану проблему.

Програмний продукт може використовуватись для навчального процесу або у прикладних дослідженнях для представлення рішень модельних задач процесів, що описуються диференціальними рівняннями у часткових похідних.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ ТА ВИЗНАЧЕНЬ

PDE – Partial differential equation

LSTM – Long short-term memory

DGM – Deep Galerkin Method

DNN – Deep Neural Network

ML – Machine learning

SGD – Stochastic gradient descent

ВСТУП

Актуальність роботи та підстави для її виконання. Диференціальні рівняння в часткових похідних широко використовуються в багатьох природничих науках для моделювання природничих або соціальних проблем. Їх часто використовують для опису природних явищ і в моделюванні динамічних систем. В фізиці це, наприклад, опис коливальних процесів, процесу дифузії чи поширення тепла. В фінансах – визначення вартості складних фінансових інструментів, наприклад, рівняння Блека-Шоулза, що моделює ціну опціону. Також такі диференціальні рівняння використовують для опису поширення продукції по території країн світу, поширення захворювань тощо. Але внаслідок складності чисельної реалізації їх розв’язків питання про розробку ефективних методів для розв’язання диференціальних рівнянь з частковими похідними й на сьогодні залишається відкритим.

Оцінка сучасного стану об’єкта розробки. Загалом, класичні чисельні методи, такі як Метод скінченних різниць, вимагають використання сіток для поділу простору. Чим дрібніше сітка, тим теоретично точніше отриманий розв’язок. Але, відповідно, більш дрібні сітки вимагають більших обчислювальних витрат і більшої пам’яті для зберігання. Таким чином, ці методи можуть отримати хороші наближення до розв’язку вихідного рівняння при малих розмірах, але погано відпрацьовують на більшій розмірності, яка є дорогою з точки зору обчислень. Отже, у випадку просторових задач доцільним є використання підходу без сітки. Останнім часом все більше уваги приділяється методам розв’язування диференціальних рівнянь глибоким навчанням. Ми можемо використовувати нейронну мережу, щоб представити функцію, яку потрібно розв’язати, у диференціальному рівнянні в частинних похідних, і отримати наближений розв’язок диференціального рівняння,

налаштовуючи параметри нейронної мережі. У порівнянні з методом на основі сітки, метод глибокого навчання може значно зменшити кількість параметрів, що призведе до меншого споживання обчислень. У той же час, через популярність нинішньої системи глибокого навчання, складність програмування використання глибокого навчання для розв'язання диференціальних рівнянь у частинних похідних також буде нижчою, ніж у сіткових методів.

Мета й завдання роботи. Метою роботи є аналіз існуючих та створення нової моделі машинного навчання, яка допоможе ефективно розв'язати диференціальні рівняння в часткових похідних. Для досягнення цієї мети поставлені наступні завдання:

- дослідити існуючі алгоритми та підходи до розв'язання диференціальних задач в частинних похідних;
- дослідити застосування різних архітектур нейронних мереж;
- розробити власний чисельний алгоритм розв'язання початково-крайової задачі, реалізувати його та дослідити;
- виконати тестові розрахунки запропонованим методом.

Об'єкт, методи й засоби розроблення. Об'єктом розроблення є модель машинного навчання, що буде розв'язувати диференціальні рівняння в частинних похідних, використовуючи набір випадкових точок для свого навчання.

Методи розроблення: аналізу предметної області, нейромережні методи, методи оцінки швидкодії алгоритмів.

В якості інструментів для розроблення було обрано сервіс Google Colab – інтегроване програмне середовище (IDE) мовою Python.

Python надає дуже широкий вибір бібліотек для машинного навчання, починаючи з найпростіших навчальних, і закінчуючи дуже просунутими,

готовими до використання у реальній індустрії. Ключовою рисою цих бібліотек є висока швидкість виконання та детальна документація, підкріплена математичними формулами, що дає змогу ефективно програмувати математичні алгоритми.

Можливі сфери застосування. Розробка даного алгоритму є розвиненням методів розв’язання просторово розподілених диференціальних початково-крайових задач, що звичайно є моделями різних фізичних процесів. Програмний продукт може використовуватись для навчального процесу або у прикладних дослідженнях для представлення рішень модельних задач процесів, що описуються диференціальними рівняннями у часткових похідних.

РОЗДІЛ 1.
ОГЛЯД ІСНУЮЧИХ ПІДХОДІВ

Диференціальне рівняння з частинними похідними (також відоме як рівняння математичної фізики) — математичне рівняння, яке включає дві або більше незалежних змінних, невідому функцію (залежну від цих змінних) і часткові похідні невідомої функції відносно незалежних змінних. Порядок диференціального рівняння в частинних похідних є порядком найвищої залученої похідної. Наведемо приклад загального вигляду статичного диференціального рівняння першого порядку:

$$F\left(x_1, x_2, \dots, x_n, \omega, \frac{\partial \omega}{\partial x_1}, \frac{\partial \omega}{\partial x_2}, \dots, \frac{\partial \omega}{\partial x_n}\right) = 0,$$

де $\omega = \omega(x_1, x_2, \dots, x_n)$ – невідома функція.

Розв’язок (або окремий розв’язок) диференціального рівняння в часткових похідних — це функція, яка розв’язує рівняння або, іншими словами, перетворює його в тотожність при підстановці до рівняння. Рішення називається загальним, якщо воно містить усі окремі розв’язки відповідного рівняння.

Існують аналітичні методи вирішення рівнянь у часткових похідних такі, як метод Фур'є (метод поділу змінних). В результаті застосування таких методів рішення записується у вигляді суми нескінченного ряду досить складної структури, тому знаходження чисельного значення функції у конкретній точці є окремим математичним завданням. Часто буває так, що диференціальні рівняння не можна розв’язати аналітично, отже для їх вирішення необхідно вдатися до чисельних методів. Одним з найпопулярніших чисельних методів є метод скінченних різниць.

1.1.Чисельні методи

1.1.1. Метод скінченних різниць (Метод сіток)

Для вирішення диференціального рівняння методом скінченних різниць (сіток) [1] спочатку область, на якій ми шукаємо рішення, розбивається на сегменти, які найчастіше мають однакові розміри (наприклад, рисунок 1.1). Сукупність всіх точок називають сіткою, а величини Δ та h - кроками сітки по координатам x та t відповідною.

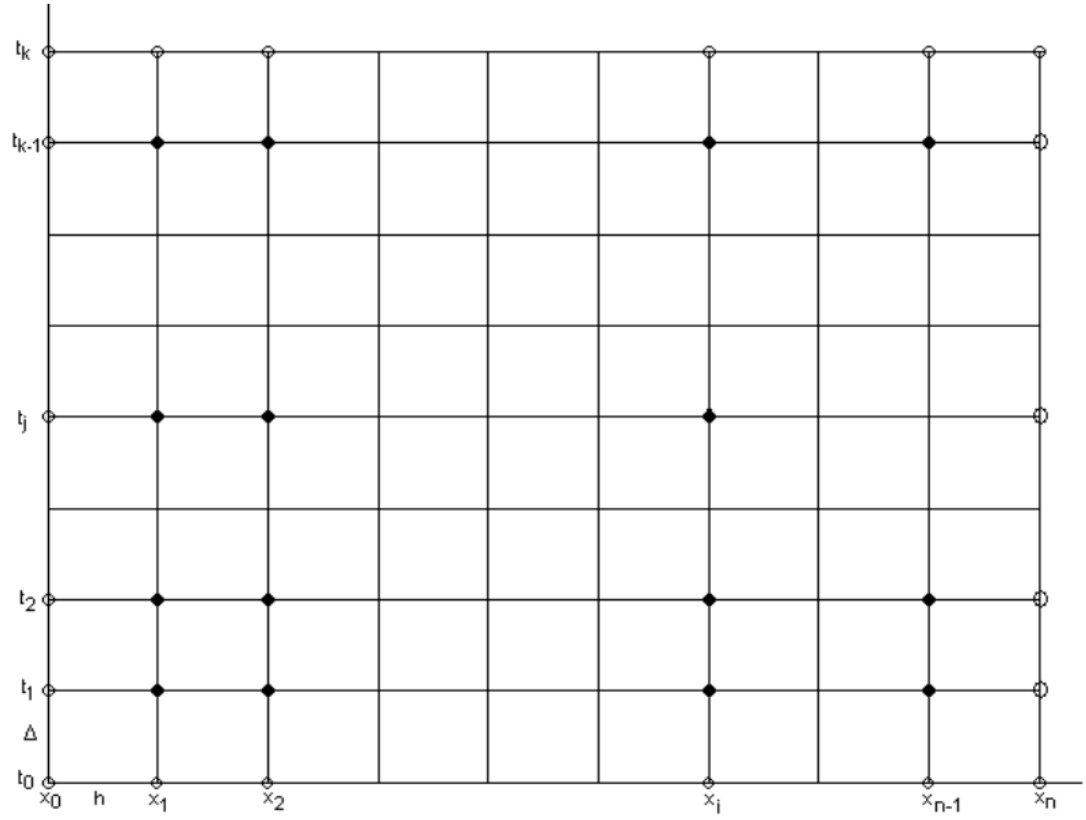

Рисунок 1.1. Приклад простої сітки для Методу скінченних різниць

Замість неперервної функції розглядаються дискретні функції, які визначені в вузлах сітки, та замість похідних будемо розглядати їх різницеві

апроксимації. Чим менші величини кроків, тим теоретично точніше алгебраїчна система моделює початкову диференціальну.

Після побудови сітки, наприклад, з порядком апроксимації $O(h^2 + \tau)$ вводимо заміни похідних їхніми різницевими аналогами на обраних шаблонах. Наприклад,

$$\frac{\partial^2 u(x_i, t_j)}{\partial x^2} = \frac{u_{i+1,j} - 2u_{i,j} + u_{i-1.j}}{h^2},$$

$$\frac{\partial u(x_i, t_j)}{\partial t} = \frac{u_{i,j+1} - u_{i,j}}{\Delta}.$$

Оскільки, на нульовому шарі значення функцій відомі, то ми після виконання заміни різницевими похідними можемо перейти від системи диференціальних рівнянь до системи алгебраїчних рівнянь (для явної або неявної схеми). Систему алгебраїчних рівнянь ми можемо розв’язати відповідними чисельними методами.

1.1.2. Методи Гальоркіна

У скінченно-різницевих методах ми апроксимуємо неперервний диференціальний оператор дискретним різницевим оператором, щоб отримати чисельну апроксимацію функції, яка задовольняє PDE, з деяким порядком апроксимації. Область визначення функції (або її частина) також повинна бути дискретизованою, щоб числові наближення значення розв’язку можна було обчислити в точках визначеної таким чином просторово-часової сітки. Крім того, значення функції в точках поза сітки також можна отримати за допомогою інтерполяційних методів.

Методи Гальоркіна використовують альтернативний підхід: заданий скінченний набір базисних функцій в одній і тій же області. Мета полягає в тому, щоб знайти лінійну комбінацію базисних функцій, яка апроксимує розв'язок PDE в області, що цікавить. Ця проблема перетворюється на варіаційну задачу, де намагаються знайти максимуми або мінімуми функції.

Точніше, нехай ми розв'язуємо рівняння $F(x) = y$, де x та y елементи просторів функцій X та Y відповідно і $F : X \to Y$ - функціонал, який може бути не лінійним. Нехай також φ_i та ψ_j формують лінійно незалежні базиси для функцій X та Y. Згідно з методом Гальоркіна, апроксимація x може бути знайдена за формулою

$$x_n = \sum_{i=1}^{n} \alpha_i \phi_i,$$

де коефіцієнти перед φ_i задовольняють рівняння:

$$\langle F\left(\sum_{i=1}^{n} \alpha_i \phi_i\right), \psi_j \rangle = \langle y, \psi_j \rangle,$$

1.1.3. Методи скінченних елементів

Методи скінченних елементів [2] можна розуміти як окремий випадок методів Гальоркіна. Зауважте, що в загальному випадку, представленому вище, апроксимація x_n може бути неправильною, у тому сенсі, що система рівнянь для α_i може не мати розв'язку або вона може мати кілька розв'язків залежно від значення n. Крім того, залежно від вибору φ_i та ψ_j, x_n може не сходитися до x при $n \to \infty$. Тим не менш, можна дискретизувати область у

досить малих областях (так званих елементах), щоб апроксимація була локально задовільною в кожній області. Додаючи обмеження узгодженості кордонів для кожного перетину області (а також для зовнішніх граничних умов, заданих визначенням проблеми) і розв’язуючи для всієї області, що цікавить, можна прийти до глобально справедливого чисельного наближення для рішення PDE.

На практиці область зазвичай поділяється на трикутники або чотирикутники (двовимірний випадок), тетраедри (тривимірний випадок) або більш загальні геометричні фігури у вищих вимірах у процесі, відомому як тріангуляція. Вибір скінченого елементу із відповідною множиною вузлів також пов'язаний з порядком апроксимації. Елементи, що використовують лише кутові вузли забезпечують лінійну інтерполяцію функцій, а елементи, що залучають вузли між кутовими точками надають квадратичну або кубічну інтерполяцію.

Типові варіанти для φ_i та ψ_j такі, що наведені вище рівняння внутрішнього добутку зводяться до системи алгебраїчних рівнянь для задач стаціонарного стану або системи ODE у випадку задач, що залежать від часу. Якщо PDE є лінійним, ці системи також будуть лінійними, і їх можна розв’язати за допомогою таких методів, як виключення Гаусса або ітераційних методів, таких як Якобі або Гаусса-Зейделя. Якщо PDE не є лінійним, може знадобитися розв’язувати системи нелінійних рівнянь, які, як правило, є більш дорогими з точки зору обчислень. Однією з основних переваг методів скінченних елементів перед методами скінченних різниць є те, що методи скінченних елементів можуть без зусиль обробляти складні геометричні кордони, які зазвичай виникають у фізичних або інженерних проблемах, тоді як цього може бути дуже важко досягти за допомогою скінченних різницевих алгоритмів.

1.2. Використання Глибокого навчання

Методи скінченних різниць стають нездійсненними при великих розмірах через зростання кількості точок сітки. Якщо простір має розмірність *d* та 1 вимір часу, сітка має розмір N^{d+1}. Коли розмірність d стає великою, рівняння стають нерозв'язними з точки зору обчислень.

Глибоке навчання зробило революцію в таких областях, як розпізнавання зображень, текстів і мовлення. Ці області вимагають статистичних підходів, які можуть моделювати нелінійні функції високорозмірних вхідних даних. Глибоке навчання, яке використовує багатошарові нейронні мережі (тобто «глибокі нейронні мережі»), виявилося дуже ефективним на практиці для таких завдань (див. рисунок 1.2). Багатошарова нейронна мережа, по суті, є «стеком» нелінійних операцій, де кожна операція прописана певними параметрами, які повинні бути оцінені з даних.

Продуктивність на практиці може сильно залежати від конкретної форми архітектури нейронної мережі та алгоритмів навчання, які використовуються. Розробка архітектур нейронних мереж і методів навчання була в центрі інтенсивних досліджень протягом останнього десятиліття. Враховуючи успіх глибокого навчання, також зростає інтерес до його застосування до низки інших галузей науки та техніки.

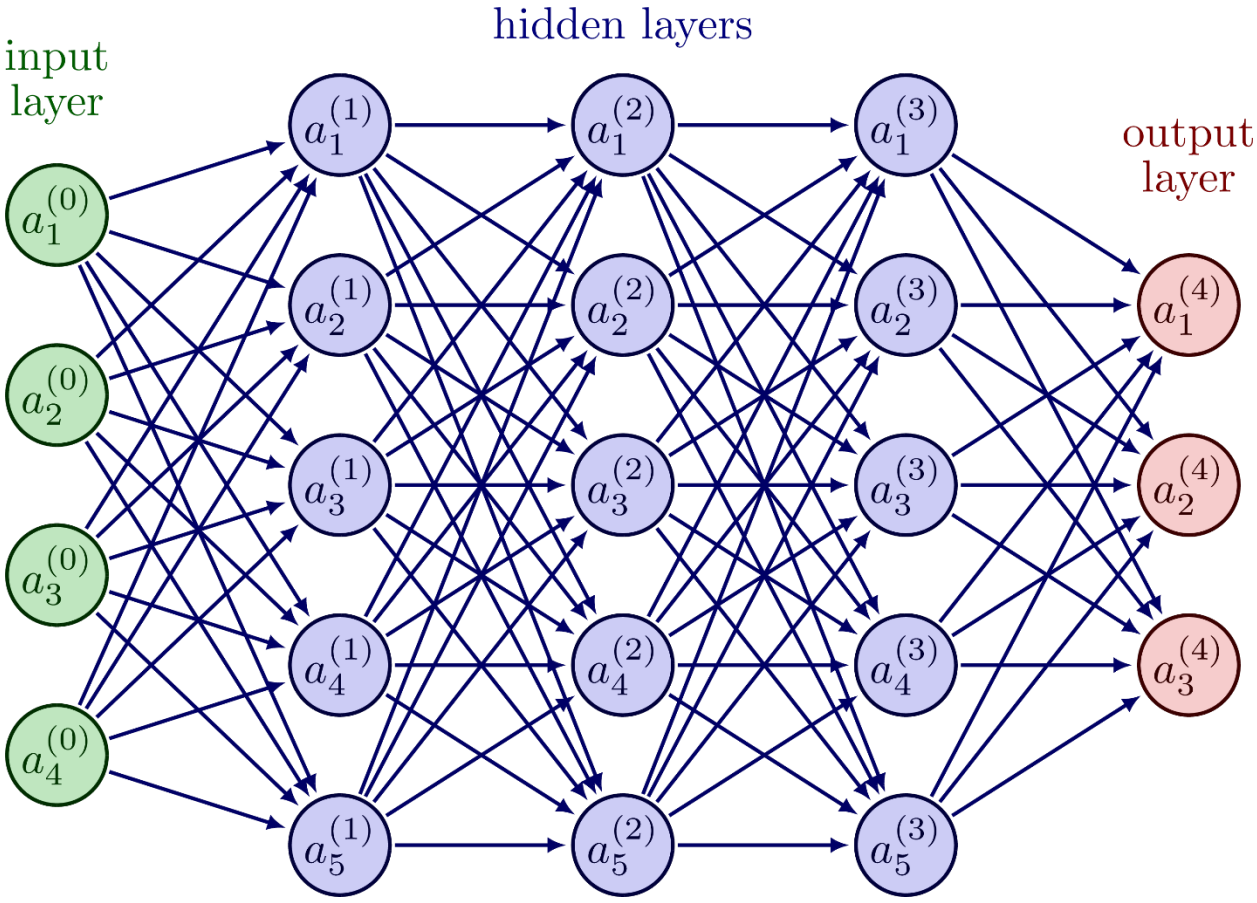

Рисунок 1.2. Приклад структури нейронної мережі

1.2.1. Нейронні мережі та глибоке навчання

Нейронні мережі – це моделі машинного навчання, яким останнім часом приділяють велику увагу завдяки їхньому успіху у вирішенні різноманітних завдань [3]. Типовий спосіб мотивації підходу, що ґрунтується на моделях нейронних мереж, — порівняти те, як вони працюють з людським мозком. Будівельні блоки мозку (і нейронних мереж) є основними обчислювальними пристроями, що називаються нейронами, які з’єднані один з одним за допомогою складної комунікаційної мережі. Зв’язки за деякої ваги та порогу

активації викликають активацію нейрона, щоб активувати інші нейрони, з якими він пов’язаний. З точки зору навчання, навчання нейронної мережі можна розглядати як визначення того, які нейрони «спрацьовують» разом.

Математично нейронну мережу можна визначити як орієнтований граф з вершинами, що представляють нейрони, і ребрами, що представляють зв’язки. Вхід для кожного нейрона є функцією зваженої суми вихідних даних усіх нейронів, які підключені до його вхідних ребер. Існує багато варіантів нейронних мереж, які відрізняються за архітектурою (як з'єднані нейрони). Найпростішою з цих форм є нейронна мережа прямого зв'язку, яку ще називають багатошаровим персептроном (MLP).

MLP можуть бути представлені орієнтованим ациклічним графом і як такий можна розглядати як подачу інформації вперед. Зазвичай мережі такого роду описуються в термінах шарів, які з’єднані разом, щоб створити вихідну функцію, де шар – це сукупність нейронів, які можна розглядати як одиницю обчислень. У найпростішому випадку є один вхідний шар і один вихідний шар. У цьому випадку вихід j (представлений j-тим нейроном у вихідному шарі) підключений до вхідного вектору x через зміщену зважену суму та функцію активації φ_j :

$$y_j = \phi_j\left(b_j + \sum_{i=1}^{d} \omega_{i,j} x_i\right).$$

Між вхідним і вихідним шарами також можна включати додаткові приховані шари. Наприклад, з одним прихованим шаром вихід буде виглядати так:

$$y_k = \phi\left[b_k^{(2)} + \sum_{i=1}^{m_2} \omega_{j,k}^{(2)} \cdot \psi\left(b_j^{(1)} + \sum_{i=1}^{m_1} \omega_{i,j}^{(1)} \cdot x_j\right)\right],$$

де $\psi\left(b_j^{(1)} + \sum_{i=1}^{m_1} \omega_{i,j}^{(1)} \cdot x_j\right)$ – перетворення вхідного шару в прихований шар, а $\phi\left[b_k^{(2)} + \sum_{i=1}^{m_2} \omega_{j,k}^{(2)} \cdot \psi\left(b_j^{(1)} + \sum_{i=1}^{m_1} \omega_{i,j}^{(1)} \cdot x_j\right)\right]$ – перетворення прихованого шару до вихідного.

Тут φ, ψ - нелінійні функції активації для кожного шару, а верхні індекси в дужках відносяться до відповідного шару. Ми можемо уявити розширення цього процесу як просте застосування правила ланцюга, наприклад:

$$f(x) = \psi_d\left(\ldots\psi_2\left(\psi_1(x)\right)\right).$$

Тут кожен рівень мережі представлений функцією ψ_i, що включає зважені суми попередніх входів і активацій підключених виходів. Кількість шарів на графіку називається глибиною нейронної мережі, а кількість нейронів у шарі представляє ширину цього конкретного шару.

Терміни «глибока» нейронна мережа та глибоке навчання відносяться до використання нейронних мереж з багатьма прихованими шарами в проблемах машинного навчання. Однією з переваг додавання прихованих шарів є те, що глибина може експоненціально зменшити обчислювальні витрати в деяких програмах і експоненціально зменшити кількість навчальних даних, необхідних для вивчення деяких функцій. Це пов’язано з тим, що деякі функції можуть бути представлені меншими глибокими мережами в порівнянні з широкими неглибокими мережами. Це зменшення розміру моделі призводить до покращення статистичної ефективності.

Легко уявити, яку величезну гнучкість і складність можна досягти, змінюючи структуру нейронної мережі. Можна змінювати глибину або ширину мережі або мати різні функції активації для кожного шару або навіть кожного нейрона. Цю гнучкість можна використовувати для досягнення дуже корисних результатів.

Далі переходимо до питання, як оцінюються параметри нейронної мережі. З цією метою ми повинні спочатку визначити функцію втрат $L(\theta; x, y)$, яка визначатиме продуктивність заданого набору параметрів θ для нейронної мережі, що складається з ваг і доданків зміщення в кожному шарі. Мета полягає в тому, щоб знайти набір параметрів, який мінімізує нашу функцію втрат. Але проблема полягає в тому, що дуже нелінійна природа нейронних мереж може призвести до неопуклості функції втрат. Проблеми неопуклої оптимізації є нетривіальними, і часто ми не можемо гарантувати, що рішення-кандидат є глобальним оптимізатором.

1.2.2. Стохастичний градієнтний спуск

Найбільш часто використовуваний підхід для оцінки параметрів нейронної мережі заснований на градієнтному спуску, який є простою методологією для оптимізації функції. Дано функцію f, ми хочемо визначити значення x, яке досягає мінімального значення f. Для цього ми починаємо з початкового припущення і обчислюємо градієнт функції у цій точці. Це визначає напрямок, у якому відбувається найбільше збільшення функції. Щоб мінімізувати функцію, ми рухаємося в протилежному напрямку, тобто виконуємо наступну ітерацію:

$$x_n = x_{n-1} - \eta \cdot \nabla_x f(x_{n-1}),$$

η — розмір кроку градієнтного спуску, що має назву швидкість навчання. Розмір кроку може бути постійним, а може зменшуватися зі збільшенням n, це призводить до більш точних результатів. Алгоритм сходить до критичної точки, коли градієнт дорівнює нулю, хоча слід зазначити, що це не обов’язково

глобальний мінімум. У контексті нейронних мереж ми обчислюємо похідні функціоналу втрат щодо набору параметрів θ.

Щоб зменшити кількість обчислювань, використовують розширення цього алгоритму, відомого як стохастичний градієнтний спуск (SGD). Коли функція втрат, яку ми мінімізуємо, є адитивною, її можна записати так:

$$\nabla L(\theta, x, y) = \frac{1}{m} \sum_{i=1}^{m} \nabla_\theta L_i\left(\theta, x^{(i)}, y^{(i)}\right),$$

де m – розмір навчального сету, а L_i – функція втрат для кожного прикладу.

Підхід у SGD полягає в тому, щоб наблизити градієнт за допомогою випадкової підмножини навчального набору, який називається міні-батчем. Тобто для фіксованої міні-батчу розміру m_0 градієнт обчислюється як:

$$\nabla_\theta L(\theta, x, y) = \frac{1}{m'} \nabla_\theta \sum_{i=1}^{m} L_i\left(\theta, x^{(i)}, y^{(i)}\right).$$

1.2.3. Адам

Алгоритм стохастичного градієнтного спуску параметризований швидкістю навчання η, яка визначає розмір кроку в напрямку найбільш крутого спуску, заданого вектором градієнта. На практиці це значення має зменшуватися під час послідовних ітерацій алгоритму SGD, щоб мережа була належним чином навчена. Щоб набір параметрів мережі був належним чином оптимізований, необхідно мати правильно вибраний графік швидкості навчання, оскільки він гарантує, що надмірна помилка зменшується на кожній

ітерації. Крім того, цей графік швидкості навчання може залежати від характеру проблеми.

З причин, розглянутих в останньому параграфі, було розроблено ряд різних алгоритмів, щоб знайти евристичні засоби, здатні керувати вибором ефективної послідовності параметрів швидкості навчання. Натхненні фізикою, багато з цих алгоритмів інтерпретують градієнт як вектор швидкості, тобто напрям і швидкість, з якою параметри рухаються в просторі параметрів. Алгоритми імпульсу, наприклад, обчислюють наступну швидкість як зважену суму градієнта останньої і знову обчисленої ітерації. Це допомагає мінімізувати нестабільність, викликану високою чутливістю функції втрат щодо деяких напрямків простору параметрів, за рахунок введення двох нових параметрів, а саме коефіцієнта затухання та параметра ініціалізації η_0. Припускаючи, що ці чутливості залежать від осі, ми можемо застосувати різні графіки швидкості навчання до кожного напряму та адаптувати їх протягом усього навчального заняття.

Робота Кінгми і Ба [4] об’єднує ідеї, обговорені в цьому розділ в єдину структуру, яку називають адаптивним імпульсом (Адам). Основна ідея полягає в тому, щоб збільшити швидкість навчання на основі попередніх величин часткових похідних для певного напрямку. Адам вважається стійким до значень гіперпараметрів.

1.2.4. Глибокий метод Гальоркіна

Нехай дано наступне диференціальне рівняння в часткових похідних на інтервалах $x \in R^n, t \in [0, T]$:

$$\partial_t u(t,x) + Lu(t,x) = 0$$

з початковою умовою

$$u(0,x) = u_0(x)$$

та крайовою умовою

$$u(t,x) = g(t,x).$$

Метою методу є апроксимація функції u нейронною мережею $f(t,x,\theta)$, де θ – множина параметрів нейронної мережі. Мережа тренується множиною точок з множини визначення диференціального рівняння, що генеруються випадково [5].

Функція втрат цієї нейронної мережі складається з трьох частин:

1. Міра, на скільки нейронна мережа задовольняє рівняння:
$$L_1(\theta) = \frac{\sum_{i=0}^{N}(\partial_t f(t_i,x_i,\theta) + Lf(t_i,x_i,\theta))}{N}.$$
2. Міра, на скільки нейронна мережа задовольняє початкову умову:
$$L_2(\theta) = \frac{\sum_{i=0}^{N}(\partial_t f(0,x_i,\theta) + u_0(x_i))}{N}.$$
3. Міра, на скільки нейронна мережа задовольняє крайову умову:
$$L_3(\theta) = \frac{\sum_{i=0}^{N}(\partial_t f(t_i,x_i,\theta) + g(t_i,x_i))}{N}.$$

Таким чином функція втрат для диференціального рівняння має вигляд:

$$L(\theta) = L_1(\theta) + L_2(\theta) + L_3(\theta).$$

Для мінімізації описаної функції втрат застосовується Стохастичний градієнтний спуск, після цього отримується нейронна мережа, яка апроксимує шукану функцію $u(t,x)$.

1.2.5. Довга короткочасна пам'ять

Програми із залежностями від часу або позиціонування, такі як розпізнавання мовлення та обробка природної мови, де кожен рівень мережі обробляє один час/позиційний крок, особливо схильні до проблеми зникаючого градієнта. Зокрема, градієнт, що зникає, може маскувати довгострокові залежності між точками спостереження, віддаленими один від одного в часі/просторі.

Простою мовою можна сказати, що нейронна мережа не здатна точно запам’ятати важливу інформацію з минулих шарів. Один із способів подолання цієї труднощі полягає в тому, щоб включити поняття пам’яті для мережі, навчаючи її вивчати, які вхідні дані з минулих шарів повинні проходити через поточний рівень і передаватися на наступний, тобто скільки інформації слід «запам’ятати» чи « забути». Це і лежить в основі мереж (рис. 1.3) довгострокової пам’яті (LSTM), представленої Хохрайтером та Шмідхубером [6].

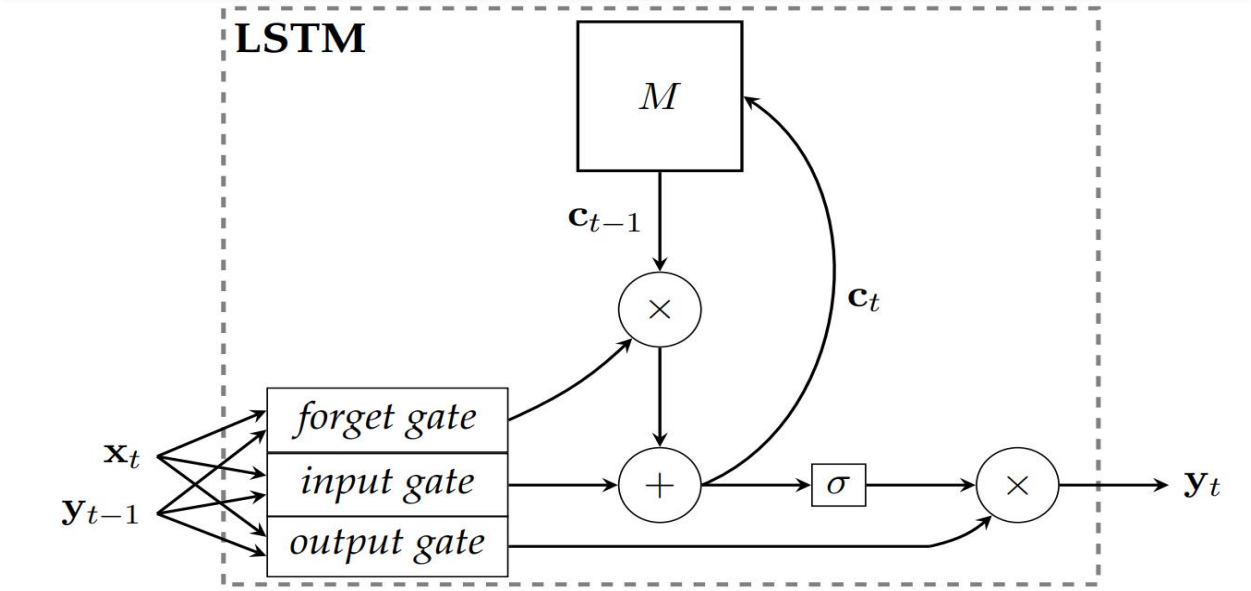

Рисунок 1.3. Архітектура частини LSTM шару

Мережі LSTM — це клас рекурентних нейронних мереж (RNN), який складається з шарів LSTM. Кожен рівень складається з комірки пам’яті, вхідного шлюзу, вихідного шлюзу та шлюзу забуття, який регулює потік інформації від одного шару до іншого і дозволяє мережі навчитися оптимальному механізму запам’ятовування/забуття. Математично деяка частина градієнтів з минулих шарів здатна проходити через поточний шар безпосередньо до наступного. Величина градієнта, який проходить через шар без змін (відносно частини, яка трансформується), а також відкинутої частини, також вивчається мережею. Це вбудовує аспект пам’яті в архітектуру LSTM, дозволяючи йому обійти проблему зникаючого градієнта та вивчати довгострокові залежності.

РОЗДІЛ 2.
ПОБУДОВА ЕФЕКТИВНОГО АЛГОРИТМУ РЕАЛІЗАЦІЇ МЕТОДУ ГЛИБОКОГО ГАЛЬОРКІНА

2.1. Постановка задачі

Нам дано диференціальне рівняння з частковими похідними, для якого потрібно знайти розв’язок Глибоким методом Гальоркіна та перевірити точність розв’язку. З метою наочності для перевірки точності результату, рівняння будуть наводитися з відомим точним аналітичним розв’язком.

Застосуємо класичний алгоритм DGM для розв’язання наступного рівняння:

$$u_t(x,t) + u_x(x,t) = 0,$$

з початковою умовою:

$$u(x,0) = sin(x),$$

та множиною визначення $x \in [-\pi, \pi]$ та $t \in [0, \pi]$.

Очевидно, що аналітичним розв’язком даного диференціального рівняння з частковими похідними є рівняння:

$$u(x,t) = sin(x - t).$$

2.2. Огляд програмного забезпечення

Як IDE будемо використовувати Google Colab – скорочено від «Colaboratory», — це продукт від Google Research. Colab дозволяє будь-кому писати та виконувати довільний код Python через браузер, і особливо добре підходить для машинного навчання, аналізу даних та освіти. Більш технічно, Colab підтримується сервісом Jupyter Notebook, який не вимагає налаштування

для використання, а також надає безкоштовний доступ до обчислювальних ресурсів, включаючи графічні процесори. GPU – досить сильно прискорює обчислення, які займають дуже багато часу під час навчанні нейронних мереж. Саме це є ключовою перевагою Google Colab.

Python є однією з найпопулярніших мов програмування для машинного навчання як в академічних колах, так і в промисловості.

Також були використані наступні бібліотеки та фрейморки:

1. PyTorch — фреймворк машинного навчання з відкритим вихідним кодом, заснований на бібліотеці Torch, який використовується для таких додатків, як комп'ютерне бачення та обробка природної мови, в першу чергу розроблений Meta AI. Це безкоштовне програмне забезпечення з відкритим вихідним кодом, випущене під ліцензією Modified BSD.
2. NumPy — популярна бібліотека для зберігання масивів чисел і виконання обчислень на них. Не тільки дозволяє часто писати більш стислий код, але й робить код швидшим, оскільки більшість підпрограм NumPy реалізовано на C для швидкості.

2.3. Навчання моделі та її архітектура

2.3.1. Створення моделі

Напишемо функцію величини кроків градієнтного спуску. Як вже зазначалось, параметр швидкості навчання — це гіперпараметр, який контролює, наскільки змінювати модель у відповідь на оцінку помилки кожного разу, коли вага моделі оновлюється.

Вибір швидкості навчання є складним, оскільки занадто мале значення може призвести до тривалого процесу навчання, який може застрягти, тоді як занадто велике значення може призвести до надто швидкого вивчення неоптимального набору ваг або нестабільного процесу навчання.

Тому було вирішено використати параметр швидкості навчання з поступовим зменшенням, що залежить від кількості ітерацій.

Створимо клас нейронної мережі. Будемо використовувати класичну нейронну мережу для DGM, яка складається з 5 лінійних шарів. На вхід до першого шару подається двохвимірний вектор, перший вимір – значення координати, другий вимір – значення часу. Кожен шар трансформує вектор наступним чином: 2 → 32 → 64 → 64 → 32 → 1 (рис. 2.1).

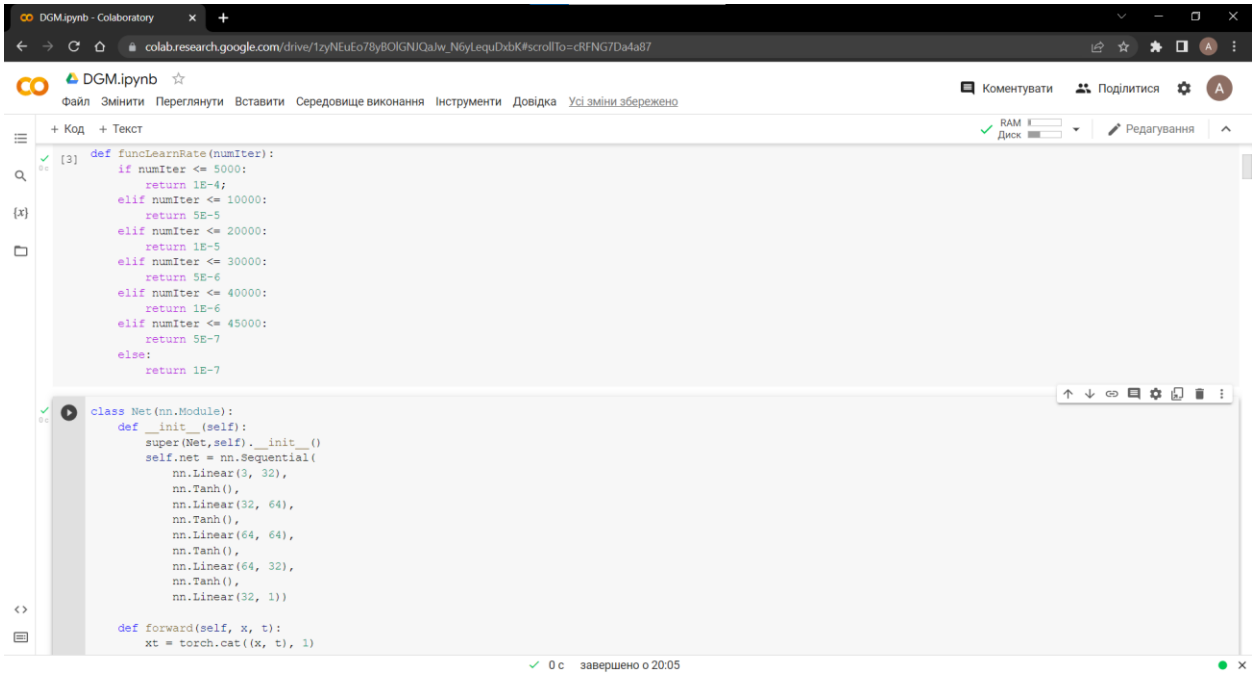

Рисунок 2.1. Демонстрація коду програми щодо задання функції величини кроків та класу нейронної мережі.

Тут nn.Sequential(*args) – це клас що здійснює реалізацію послідовного контейнера. Тобто модулі, що будуть передані в параметрах, будуть

виконуватися послідовно, в тому ж порядку в якому вони передані у конструктор.

Функція torch.cat((x, t), 1) створює матрицю додаючи вектори по розмірності 1.

Клас nn.Linear(in_features, out_features) виконує лінійну трансформацію вхідних даних за наступною формулою:

$$y = xA^T + b.$$

Після кожного шару застосовується функція гіперболічного тангенса, як функція активації:

$$Tanh(x) = \frac{\exp(x) - \exp(-x)}{\exp(x) + \exp(-x)}.$$

Кількість епох буде 20000, розмір міні-батчу 256, а для генерації тренувальної вибірки будемо використовувати клас torch.distributions.uniform.Uniform(low, high), що генерує випадкові точки в заданих межах в напіввідкритому інтервалі [low, high] за законом рівномірного розподілу. Запишемо функцію втрат, яку потрібно мінімізувати:

$$L(u) = \frac{\sum_{i=1}^{N}\left(\partial u_{t_i} + \partial u_{x_i}\right)^2}{N} + \frac{\sum_{i=1}^{N}(u(0, x_i) + \sin(x_i))^2}{N} \rightarrow min,$$

де $\frac{\sum_{i=1}^{N}\left(\partial u_{t_i} + \partial u_{x_i}\right)^2}{N}$ – втрати для диференціального рівняння в часткових похідних, а $\frac{\sum_{i=1}^{N}(u(0,x_i) + \sin(x_i))^2}{N}$ – втрати початкових умов.

Для обрахунку градієнтів, будемо використовувати torch.autograd.grad(outputs, inputs). Як алгоритм оптимізації використаємо Adam, для нього нам потрібні додаткові значення

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t$$
$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$$

де m і v — ковзні середні, g — градієнт на поточному міні-батчі, а β — нові введені гіперпараметри алгоритму. Вони мають значення за замовчуванням 0,9 і 0,999 відповідно, такі значення і використовуються в роботі. Вектори ковзаних середніх ініціалізуються нулями на першій ітерації.

2.3.2. Аналіз результатів роботи нейронної мережі

Після запуску тренування графік зміни величини функції втрат виглядає наступним чином (рис. 2.2).

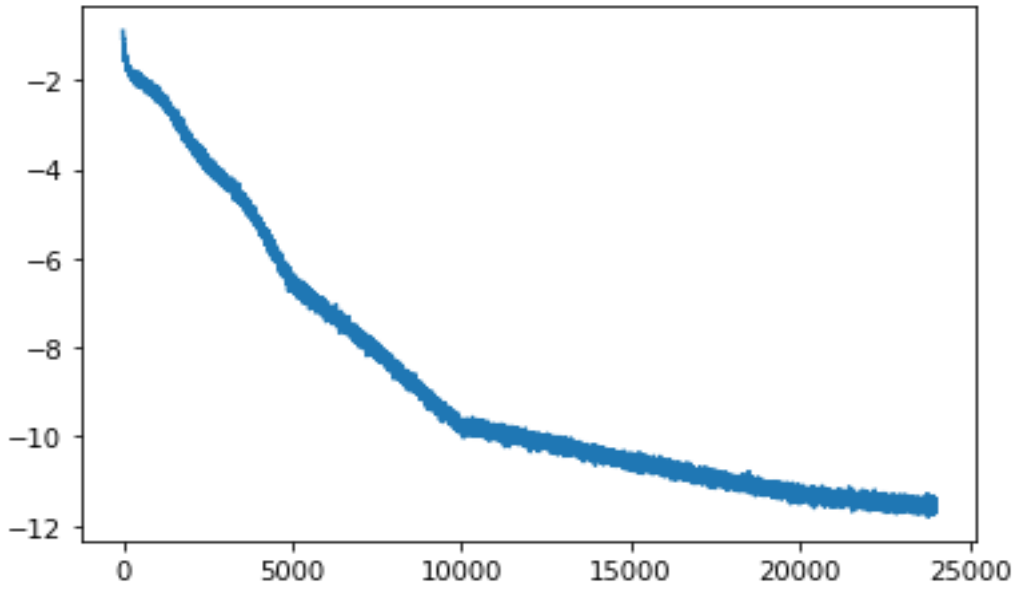

Рисунок 2.2. Графік зміни функції втрат з кількістю епох.

Також виведемо графіки порівняння розв’язку, що отриманий методом DGM (позначений на рисунку синім) з аналітичним розв’язком (позначений червоним) на рис. 2.3 – 2.5.

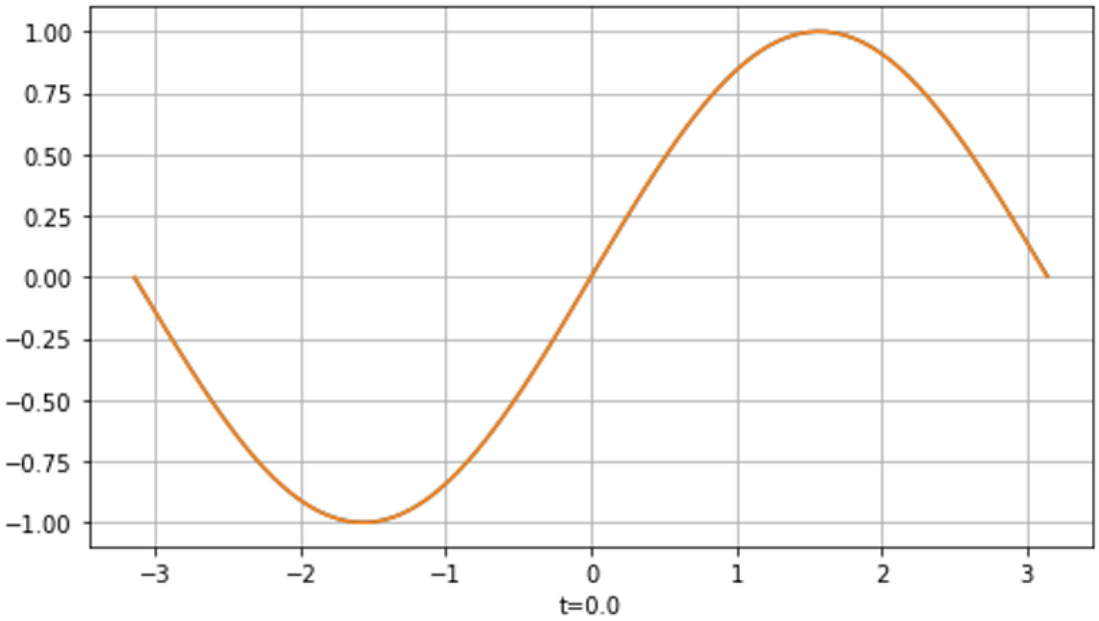

Рисунок 2.3. Порівняння аналітичного розв’язку з DGM розв’язком при значенні t = 0

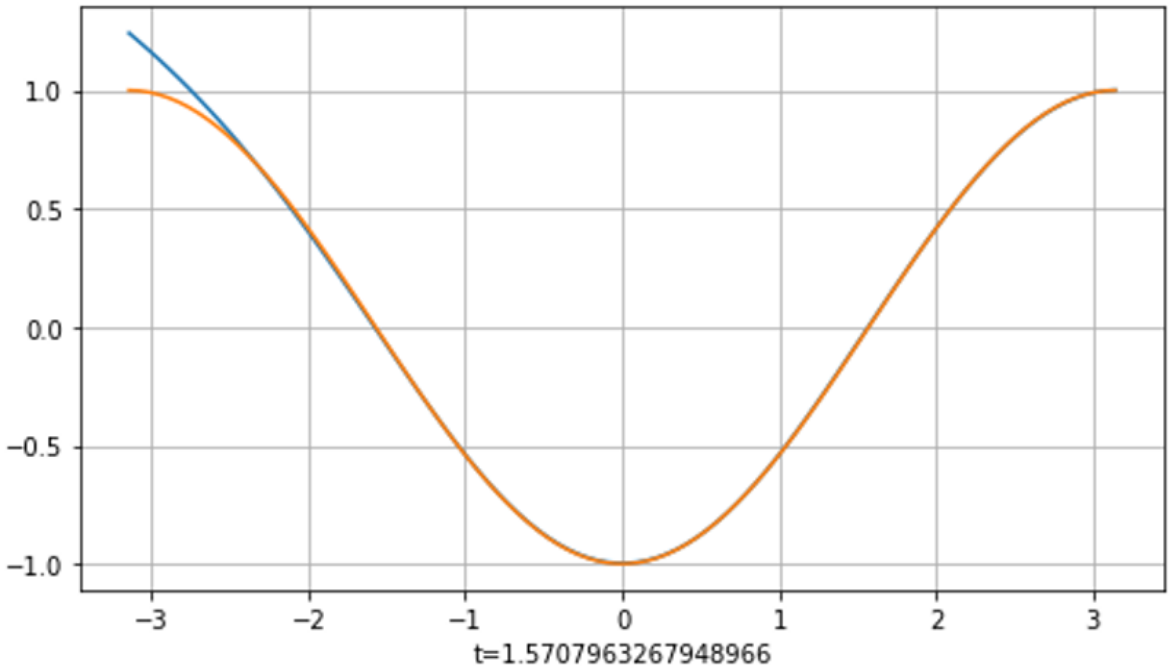

Рисунок 2.4. Порівняння аналітичного розв’язку з DGM розв’язком при значенні $t = \frac{\pi}{2}$

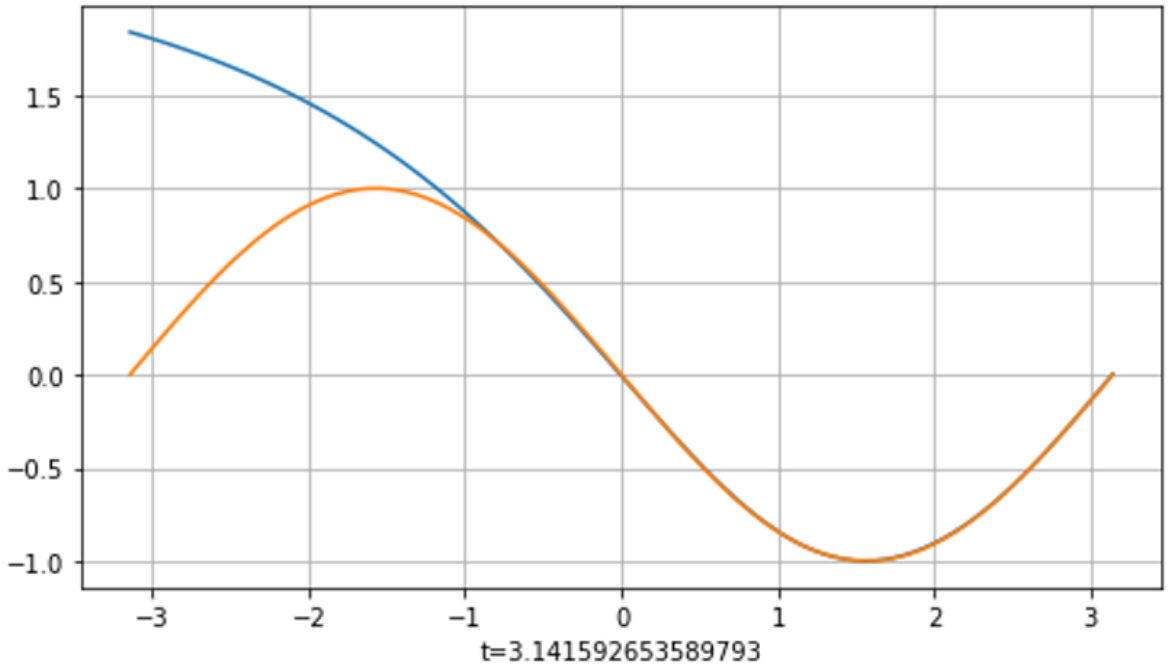

Рисунок 2.5. Порівняння аналітичного розв’язку з DGM розв’язком при значенні t = π

Можемо спостерігати, що при t = 0 розв’язок отриманий Глибоким методом Гальоркіна майже повністю збігся з аналітичним (рис. 2.3), чого немає при t = $\frac{\pi}{2}$ та при t = π. Отже маємо розбіжність, коли *t* набувають більших значень (рис. 2.4, 2.5).

Для вишення цього спробуємо збільшити межі тренувальниго сету від $x \in [-\pi, \pi], t \in [0, \pi]$ до $x \in [-2\pi, 2\pi]$ $t \in [0,2\pi]$. Після цього також виведемо функцію втрат та результати при тих самих значеннях t. Також збільшимо кількість епох до 35000, оскільки очевидно що мінімізація функції втрат зі збільшенням меж погіршиться.

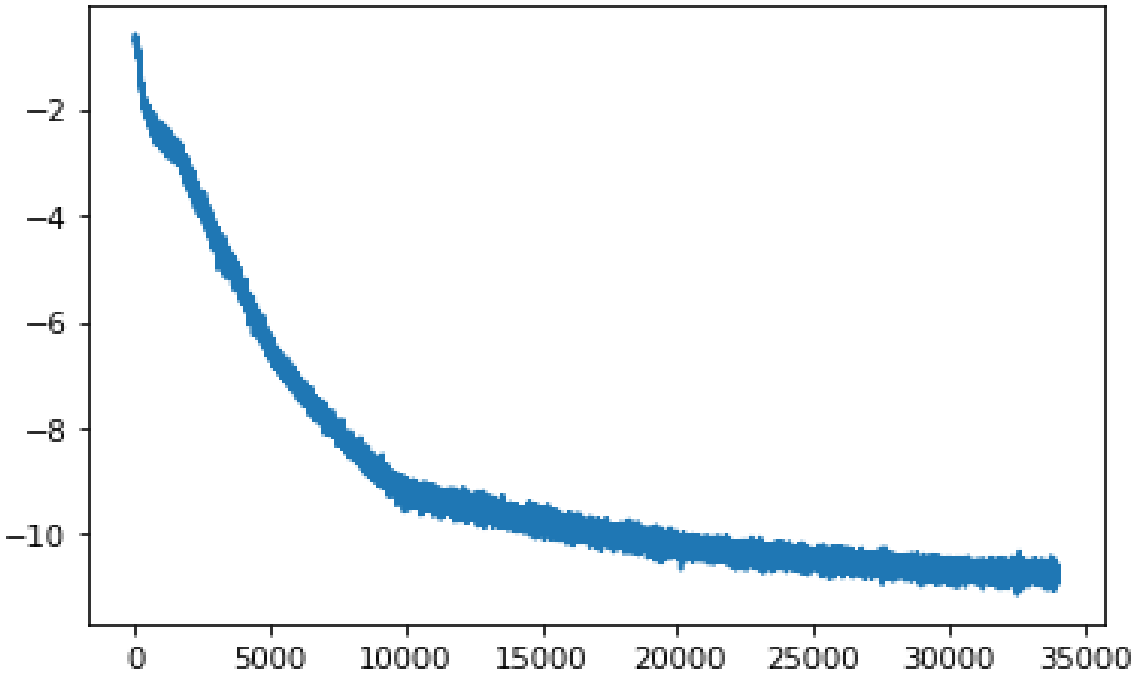

Рисунок 2.6. Графік зміни функції втрат з кількістю епох при збільшенні меж в тренувальному сеті

Ми бачимо, що мінімізація функція втрат погіршилася не зважаючи на збільшення кількості епох (рис. 2.6). Попри це результат збіжності з аналітичним розв'язком сильно покращився у кінцевих точках, але погіршився в деяких середніх точках (рис. 2.7 – 2.9).

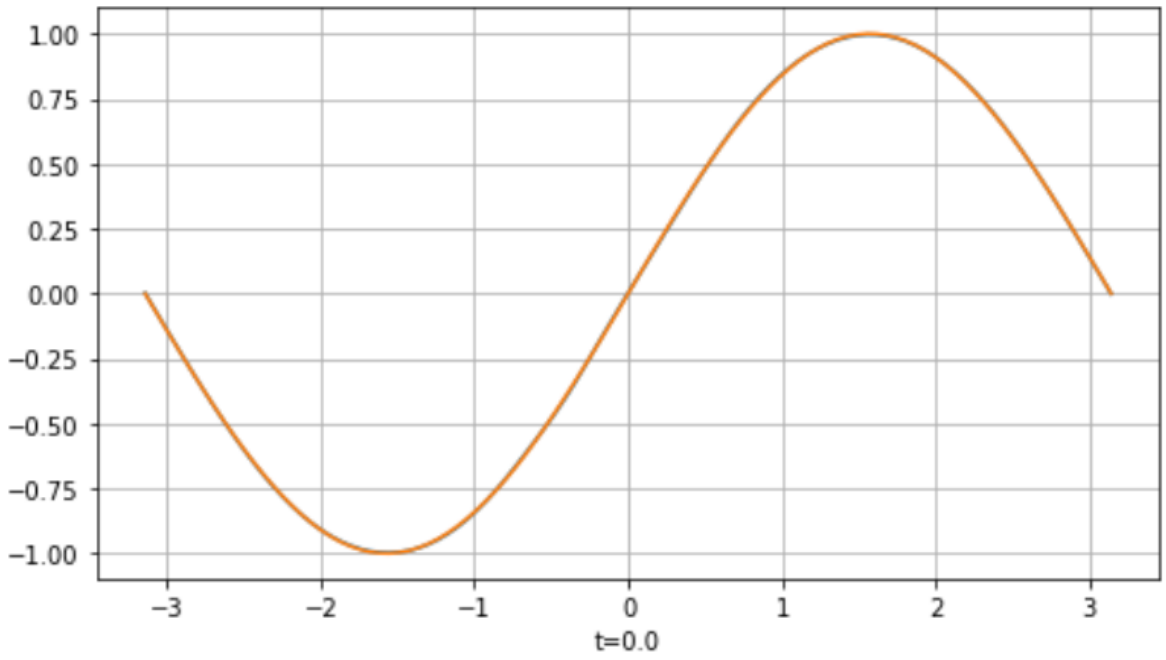

Рисунок 2.7. Порівняння аналітичного розв’язку з DGM розв’язком при значенні t = 0 при збільшених межах

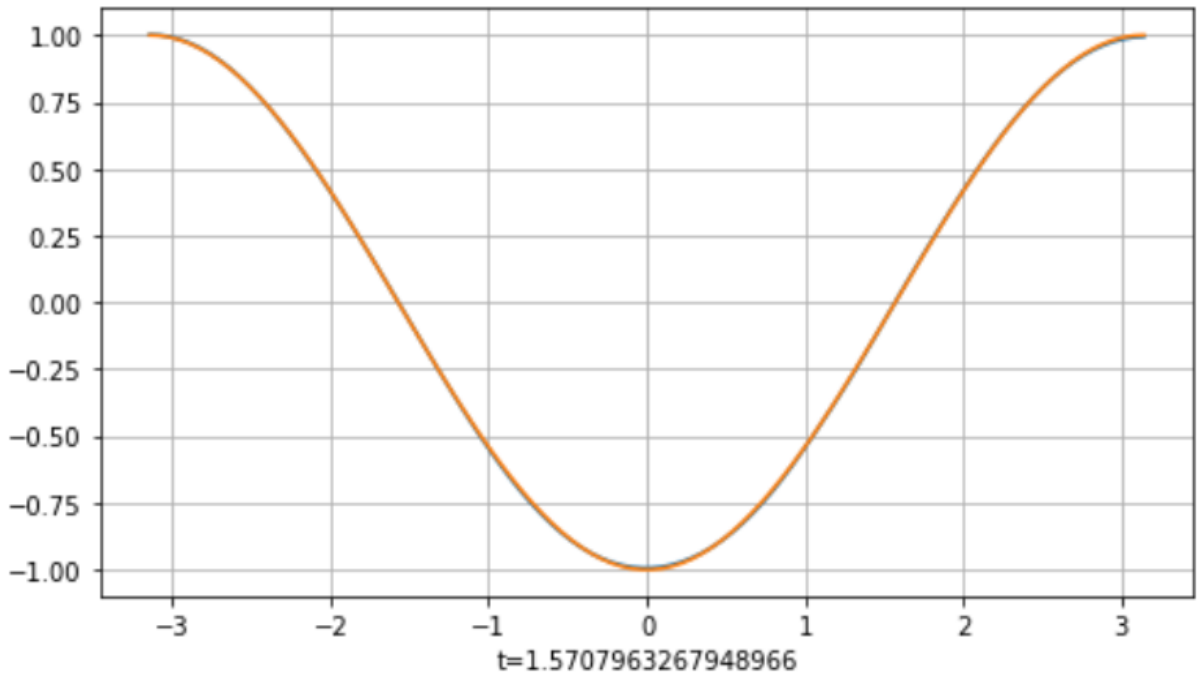

Рисунок 2.8. Порівняння аналітичного розв’язку з DGM розв’язком при значенні t = $\frac{\pi}{2}$ при збільшених межах

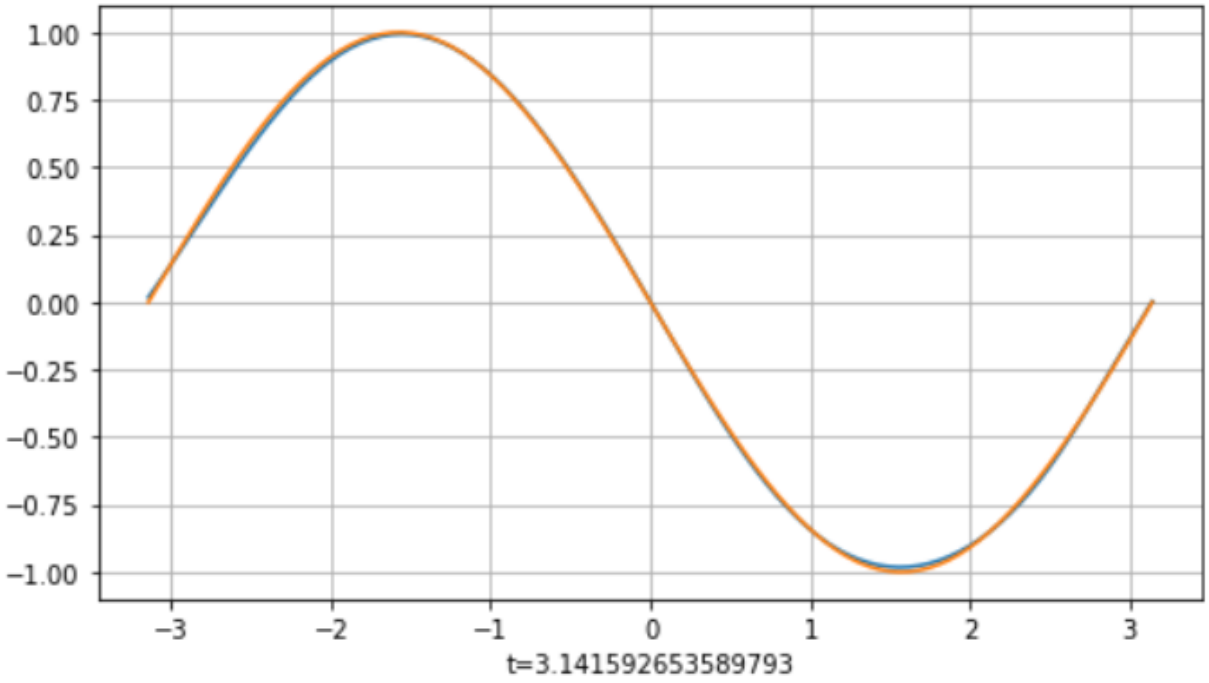

Рисунок 2.9. Порівняння аналітичного розв’язку з DGM розв’язком при значенні t = π при збільшених межах

2.3.3. Розв’язання рівняння теплопровідності

Рівняння теплопровідності — диференціальне рівняння в часткових похідних другого порядку, яке визначає розподіл температури у заданій області простору та її зміна у часі.

Розв’яжемо наступне рівняння теплопровідності:

$$u_t - \Delta u = 0,$$

$$u(x_1, x_2, 0) = \sin(\pi x_1)\sin(\pi x_2).$$

Дане рівняння визначене на множині $x_1 \epsilon [0,1]$, $x_2 \epsilon [0,1]$, $t \epsilon [0,1]$.

В рівнянні Δ – оператор Лапласа, що означає:

$$\Delta F = \frac{\partial^2 F}{\partial x_1^2} + \frac{\partial^2 F}{\partial x_2^2} + \cdots + \frac{\partial^2 F}{\partial x_n^2}$$

При розв’язанні рівняння теплопровідності мережею, що використовувалася вище, отримано величини функції втрат, показані на рисунку 2.10.

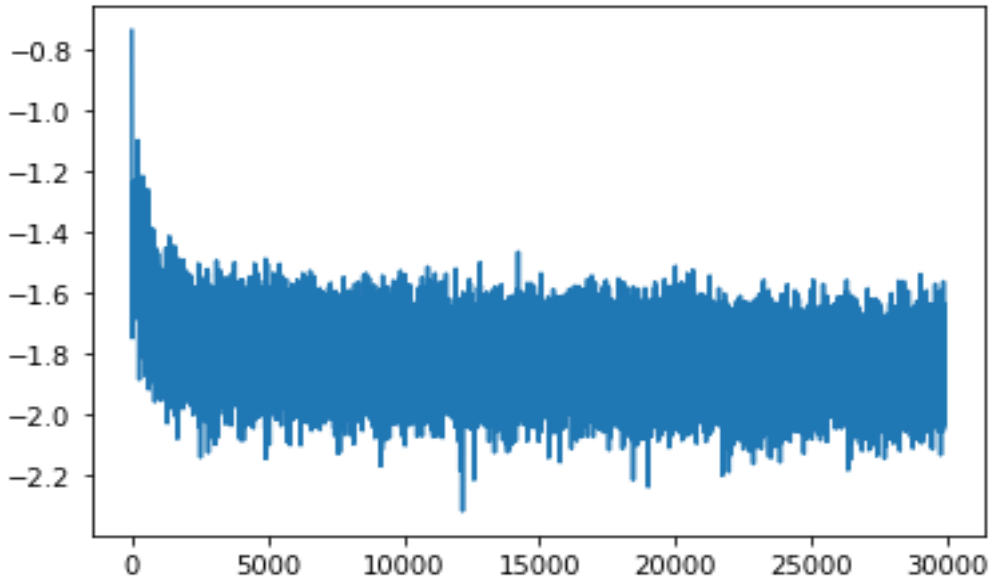

Рисунок 2.10. Графік зміни функції втрат з кількістю епох для рівняння теплопровідності розв’язаним методом DGM

Очевидно, що ця нейромережа потребує покращень, оскільки рішення не може бути ефективним. Для покращення нейромережі можна запропонувати збільшити кількість її шарів, але тоді можливі проблеми пов’язані зі зникненням градієнту. Цю проблему допомагає вирішити LSTM нейромережа.

2.3.4. Покращення архітектури

У якості покращеної моделі було обрано модель, що базується на LSTM шарах. Як вже згадувалося у цій роботі, ці нейромережі чудово підходять для виявлення якихось послідовних патернів, а саме тому використовуються у ряді задач машинного навчання, таких як обробка природної мови та аналіз часових рядів. Так як LSTM cell є одним із основних елементів нейронної мережі і суттєво відрізняє її від аналогів, то варто зупинитися на принципі роботи LSTM шару.

Як вже зазначалося, LSTM – це окремий випадок рекурентних нейромереж. Порівняно зі звичайними рекурентними нейромережами, LSTM мережі є більш робастними для визначення довгих часових трендів (як от наприклад рух автомобіля протягом певного часу).

LSTM розшифровується як Long-Short-Term-Memory, а отже в такому шарі є так звана “пам’ять” (позначення на рис. 1.3).

Базуючись на нових вхідних даних, попередньому стані та попередньому виході, LSTM шар проводить операції, використовуючи так звані “гейти”. Вони бувають трьох типів:

- забуття (forget): використовує інпути для того, щоб вирішити які дані “забути” з попереднього стану;
- вхід (input) вирішує які нові дані зберегти, базуючись на нових вхідних даний та попередньому виході;
- вивід (output) комбінує новий вивід шару, базуючись на попередніх станах та на виході input гейту.

Таким чином, нейромережа має можливість визначати довгострокові тренди у вхідних даних і робити це з адекватною обчислювальною ефективністю, адже шар не накопичує всі дані, а тримає в пам’яті тільки ті, які

є актуальними. Через свою рекурсивну природу, один LSTM шар може бути розглянутий як окрема глибока нейронна мережа (див. рис. 2.11). Теоретично, такі шари можна нашаровувати один на інший, схоже з тим, як працюють згорткові нейронні мережі. проте попередні досліди показують, що в задачах, подібних на цю, нашарування цих шарів не має переваги над одним єдиним шаром нейромережі.

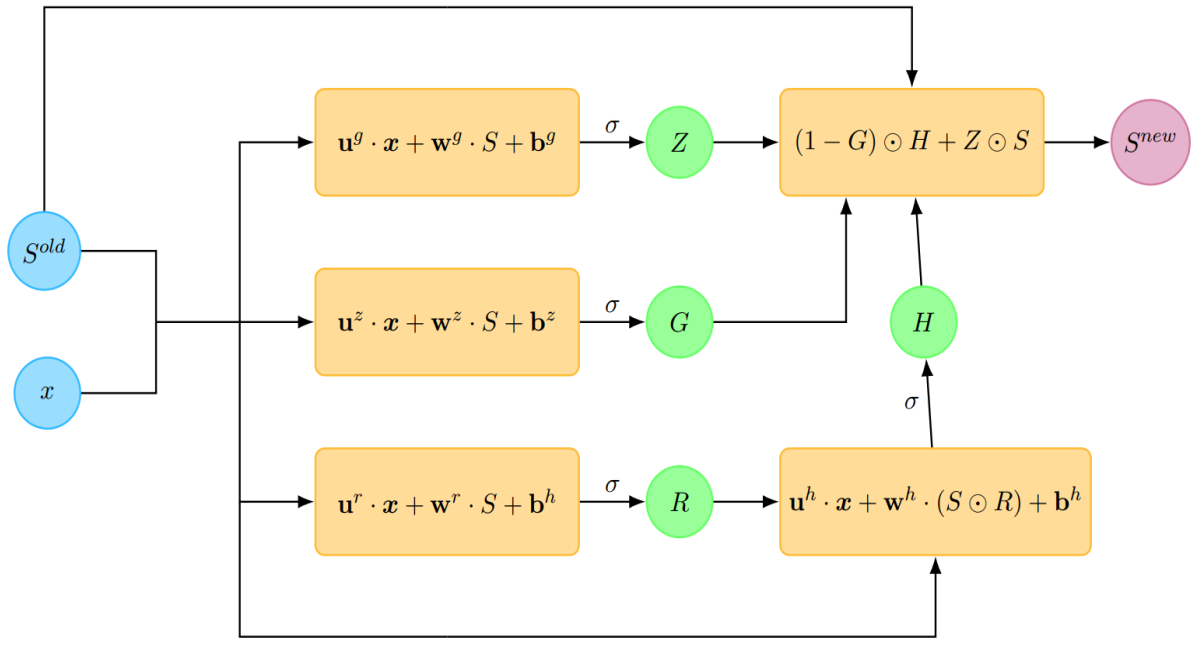

Рисунок 2.11. Операції в середині одного LSTM шару

В обраній задачі використовується модель, що комбінує у собі LSTM шар з звичайними лінійними шарами. В програмі нейромережа складається з вхідного лінійного шару, з трьох LSTM шарів та з вихідного лінійного. Клас LSTM шару реалізований наступним чином (рис. 2.12).

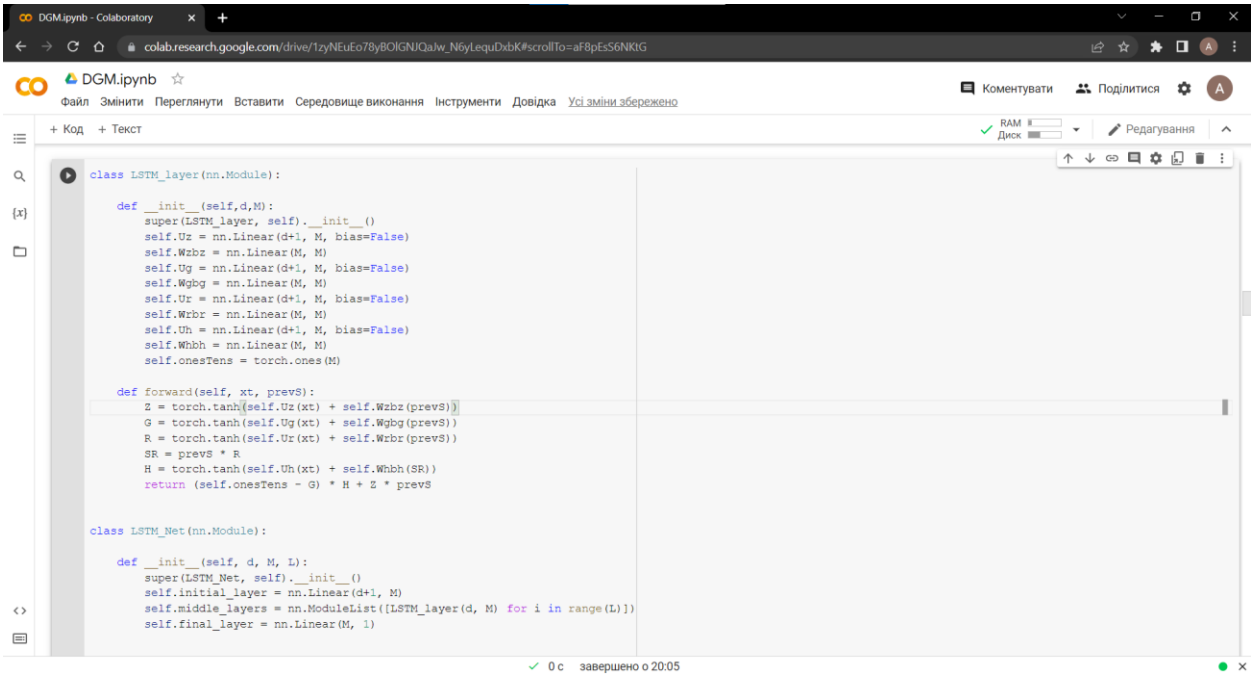

Рисунок 2.12. Демонстрація коду програми класу LSTM.

На вхід мережа приймає параметри d – розмірність простору диференціального рівняння, M – кількість нейронів в одному шарі та L – кількість шарів. В дослідженні було використано $d = 2$, $M = 50$ та $L = 3$.

Тренування нейронної мережі з покращенням зайняло значно більше часу, але після тренування нової запропонованої нейронної мережі отримали значно кращі результати, що підтверджуються наступним графіком зміни функції втрат від кількості епох (рис. 2.13).

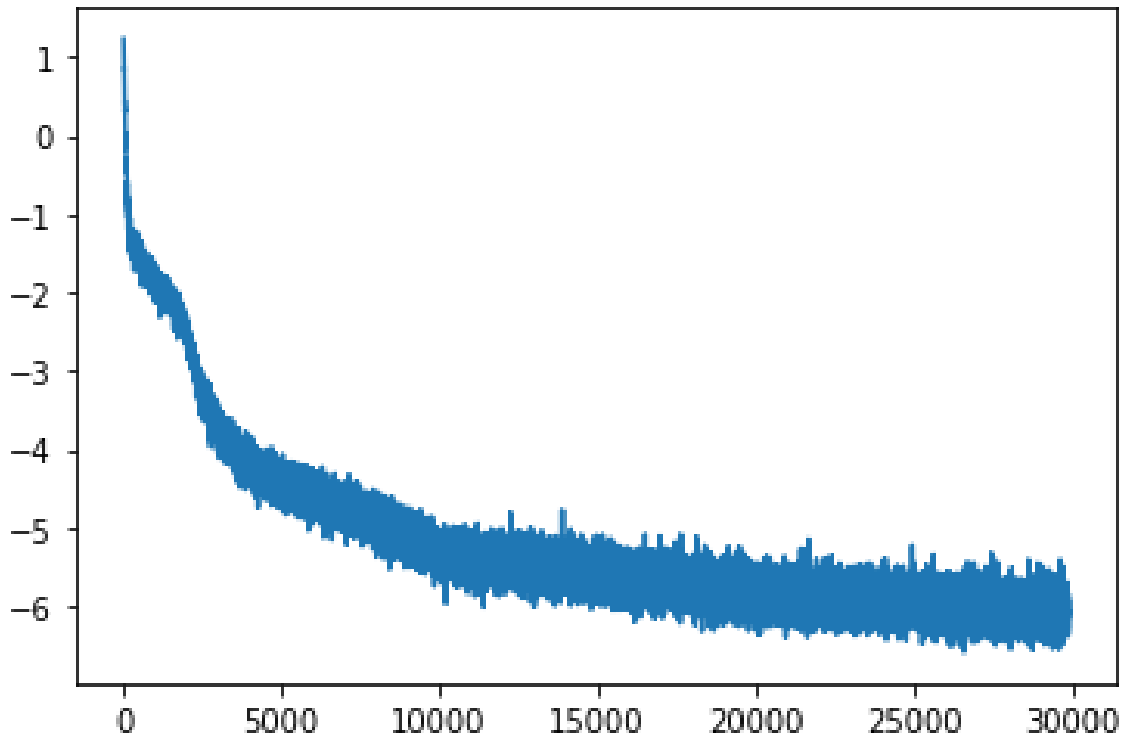

Рисунок 2.13. Графік зміни функції втрат з кількістю епох для рівняння теплопровідності з використанням LSTM архітектури

Отже, використання архітектури LSTM дійсно покращує Глибокий метод Гальоркіна, що використовується для розв'язання диференціальних рівнянь з частковими похідними.

ВИСНОВКИ

В результаті виконання роботи було розглянуто питання вдосконалення та розробки нових алгоритмів для розв’язання динамічних диференціальних рівнянь з частковими похідними Глибоким методом Гальоркіна.

Для досягнення поставлених в роботі завдань було визначено інструменти та математичну базу для вирішення проблеми. В якості інструментів для розроблення було обрано сервіс Google Colab – інтегроване програмне середовище (IDE) мовою Python, та бібліотеки для машинного навчання Numpy та PyTorch.

В роботі було досліджено:

- існуючі алгоритми та підходи до розв’язання диференціальних задач в частинних похідних;
- застосування різних архітектур нейронних мереж до розв’язання диференціальних рівнянь з частковими похідними;
- побудовано, досліджено та реалізовано власний чисельний алгоритм машинного навчання, що може з достатньою точністю вирішувати початково-крайові диференціальні задачі.
- виконано тестові розрахунки запропонованим алгоритмом.

Розроблений алгоритм є вдосконаленням методів розв’язання просторово розподілених диференціальних початково-крайових задач, які звичайно є математичними моделями для різноманітних природничих процесів.

Запропонований програмний продукт може застосовуватися як для потреб навчального процесу, так і у прикладних галузевих дослідженнях для представлення поведінки рішень модельних задач процесів, які описуються диференціальними рівняннями у часткових похідних.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Randall J LeVeque. Finite difference methods for ordinary and partial differential equations: steady-state and time-dependent problems, volume 98. Siam, 2007.
2. Susanne Brenner and Ridgway Scott. The mathematical theory of finite element methods, volume 15. Springer Science & Business Media, 2007.
3. Kingma, D. P., Ba, J., 2014. ADAM: A method for stochastic optimization. arXiv preprint arXiv:1412.6980.
4. Shalev-Shwartz, S., Ben-David, S., 2014. Understanding Machine Learning: From Theory to Algorithms. Cambridge University Press.
5. Sirignano, J. and K. Spiliopoulos (2018). DGM: A deep learning algorithm for solving partial differential equations. Journal of Computational Physics 375, 1339–1364.
6. Hochreiter, Sepp & Schmidhuber, Jürgen. (1997). Long Short-term Memory. Neural computation. 9. 1735-80. 10.1162/neco.1997.9.8.1735.
7. Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. Communications of the ACM, 60:84–90, 2012.
8. Evans, L. C., 2010. Partial Differential Equations. American Mathematical Society, Providence, R.I
9. NumPy Documentation [Електронний ресурс] – Режим доступу до ресурсу: https://numpy.org/doc/stable/index.html
10. PyTorch Documentation [Електронний ресурс] – Режим доступу до ресурсу: https://pytorch.org/docs/stable/index.html
11. G. Cybenko, Approximation by superposition of a sigmoidal function, Mathematics of Control, Signals and Systems, Vol. 2, pp. 303-314, 1989

12. Bishop, C. M., 2006. Pattern Recognition and Machine Learning. Springer
13. Srivastava, R. K., Greff, K., Schmidhuber, J., 2015. Highway networks. arXiv preprint.
14. Touzi, N., 2012. Optimal Stochastic Control, Stochastic Target Problems, and Backward SDE. Vol. 29. Springer Science & Business Media.
15. Martin Hutzenthaler, Arnulf Jentzen, Thomas Kruse, and Tuan Anh Nguyen. A proof that rectified deep neural networks overcome the curse of dimensionality in the numerical approximation of semilinear heat equations. SN Partial Differential Equations and Applications, 1:1–34, 2020.