КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

ІМЕНІ ТАРАСА ШЕВЧЕНКА

Факультет комп’ютерних наук та кібернетики
Кафедра інтелектуальних програмних систем

Кваліфікаційна робота
на здобуття ступеня бакалавра
за спеціальністю 121 Інженерія програмного забезпечення

на тему:

ВИКОРИСТАННЯ НЕЙРОМЕРЕЖ ДЛЯ ЗНАХОДЖЕННЯ НАБЛИЖЕНОГО РОЗВ’ЯЗКУ ДИФЕРЕНЦІАЛЬНИХ РІВНЯНЬ

Виконав студент 4-го курсу
Івашин Дарій Юрійович _____
(підпис)

Науковий керівник:
доцент, кандидат фізико-математичних наук
Ярослав ЛІНДЕР _____
(підпис)

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

Студент _____
(підпис)

Роботу розглянуто й допущено до захисту на засіданні кафедри інтелектуальних програмних систем
«____» _____ 202_ р.,
протокол № ___
Завідувач кафедри
Олександр ПРОВОТАР _____
(підпис)

Київ – 2022

РЕФЕРАТ

Обсяг роботи 35 сторінок, 25 ілюстрацій, 21 джерело посилання.

Ключові слова: АРХІТЕКТУРА НЕЙРОННОЇ МЕРЕЖІ, ГІПЕРПАРАМЕТР, ГЛИБИННА НЕЙРОННА МЕРЕЖА, ГЛИБИННЕ НАВЧАННЯ, ДИФЕРЕНЦІАЛЬНЕ РІВНЯННЯ, МАШИННЕ НАВЧАННЯ, МЕТОД АДАМСА-БЕШФОРТА, МЕТОД РУНГЕ-КУТТА.

Об’єктом дослідження є задача знаходження наближеного розв’язку диференціального рівняння, сформульована як задача оптимізації.

Метою роботи є розробка програмного коду та дослідження використання глибинних нейронних мереж у розв’язку диференціальних рівнянь.

Для розробки програмного коду використовувалася мова програмування Python, програмна бібліотека для машинного навчання TensorFlow, програмне забезпечення NumPy, бібліотека високоякісних наукових інструментів для мови Python SciPy, бібліотека для візуалізації даних двовимірною 2D графікою matplotlib.

У роботі досліджено оптимальну архітектуру нейромережі для вирішення диференціальних рівнянь, досліджено залежність ефективності алгоритму від функції активації. Також порівняно використання глибинних нейронних мереж з традиційними числовими методами для вирішення поставленної задачі.

Результати цієї роботи можуть бути використані при подальшому дослідженнях у галузі прикладного використання нейронних мереж у теоретичних обчисленнях.

ЗМІСТ

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАЧЕННЯ

ГНМ — глибинна нейронна мережа

МЗПП — метод зворотного поширення помилки

АБ — методи Адамса-Башфорта

АМ — методи Адамса-Моултона

АБ2 — метод Адамса-Башфорта другого порядку

РК4 — метод Рунге-Кутти 4 порядку

ВСТУП

Нейронна мережа — це алгоритм оптимізації, де функція втрат є виміром ефективності оптимізації. Якщо якась задача була зведена до вигляду $F(x) = 0$, то її розв’язок можна знайти шляхом мінімізації функції втрат нейронної мережі.

Оцінка сучасного стану об’єкта розробки. За останні 15 років нейронні мережі, або глибинні нейронні мережі, активно та успішно використовувалися в різноманітних дисциплінах, таких як комп’ютерний зір та автоматичне розпізнавання мовлення. Але попри те, що цей алгоритм оптимізації виявився неймовірно ефективним у прикладних дисциплінах, глибинні нейронні мережі мало використовувалися в галузі теоретичних обчислень, і лише нещодавно почалася тенденція використання глибинних нейронних мереж для знаходження наближеного розв’язку диференційних рівнянь.

Актуальність роботи та підстави для її виконання. Не дивлячись на те, що вже існують традиційні числові методи для розв’язання диференційних рівнянь, вони часто не відповідають необхідним потребам, таким як точність обчислень, стабільність, збіжність та швидкість обчислення розв’язку. Саме тому актуальним питанням є запобігання цих недоліків, що в свою чергу досягається використанням глибинних нейроних мереж для вирішення поставленної задачі, тобто для знаходження наближеного розв’язку диференційного рівняння. До того ж, попри все інше, підставою для цієї роботи є також поширення використання глибинних нейронних мереж у теоретичних обчисленнях і внесок у розвиток цієї галузі.

Мета й завдання роботи. За мету роботи ставилася розробка глибинної нейронної мережі для наближеного розв’язку діференційних рівнянь, дослідження кращої архітектури нейронної мережі, а також визначення ефективності використання алгоритмів глибинного навчання в поставленій задачі, порівняти ці алгоритми з традиційними числовими методами розв’язку диференційних рівнянь.

Для досягнення поставленої мети були поставленні такі завдання.

- Систематизувати існуючий досвід.
- Спроектувати та розробити програмний код алгоритму глибинного навчання та традиційних числових алгоритмів.
- Дослідити ефективність розробленої програми та алгоритму глибинного навчання.

Об'єкт, методи й засоби розроблення. Для розробки програмного коду використовувалася мова програмування Python, а також програмна бібліотека для машинного навчання TensorFlow — для розробки алгоритму глибинного навчання; програмне забезпечення, що є розширенням мови Python, NumPy — для зручності обрахунків; бібліотека високоякісних наукових інструментів для мови Python SciPy — для використання традиційних числових алгоритмів; бібліотека для візуалізації даних двовимірною 2D графікою matplotlib.

Можливі сфери застосування. Результати виконаної роботи можуть бути використані у подальшому дослідженні використання алгоритмів глибинного навчання для наближеного розв’язку диференційних рівнянь.

1 ТЕОРЕТИЧНІ ВІДОМОСТІ

1.1 Глибинне навчання

1.1.1 Що таке машинне навчання

Машинне навчання — це область штучного інтелекту, яка присвячена розумінню та створенню алгоритмів, які «навчаються», тобто алгоритмів, які автоматично покращуються, тобто підвищують свою продуктивність при виконанні певного набору завдань, завдяки досвіду. [2] Алгоритми машинного навчання будують модель, грунтуючись на вибіркових даних, або їх ще називають навчальні дані, яка б робила прогнози або приймати рішення без застосування явного програмування. [3]

Машинне навчання тісно пов’язано з задачами оптимізації: завдання навчання зазвичай формулюються як мінімізація деякої функції втрат на навчальній виборці. Функції втрат виражають невідповідність між передбаченнями моделі, яка навчається, і фактичними прикладами результатів проблеми (наприклад, у класифікації потрібно призначити мітку екземплярам, а моделі навчаються правильно прогнозувати попередньо призначені мітки набору приклади). [5]

Задля узагальнення виникає різниця між поняттями оптимізації та машинного навчання: тоді як алгоритми оптимізації мають змогу мінімізувати втрати лише на навчальному наборі, алгоритми машинного навчання здатні на мінімізацію втрат на невидимих вибірках. Характеристика узагальнення різноманітних алгоритмів навчання є активною темою сучасних досліджень, особливо для алгоритмів глибинного навчання.

Основна мета алгоритму — це узагальнити свій досвід. У контексті машинного навчання під узагальненням розуміють здатність машини, що навчається, з достатньою точністю виконувати приклади, небачані завдання після того, як вона навчилася виконувати завдання на навчальних даних. Як правило, генерація навчальних прикладів походить за деякого загалом невідомого розподілу ймовірностей, який може вважитися репрезентативним для простору подій конкретної задачі, і метою алгоритму, або навчальної

машини, є побудова такої моделі, яка б була загальної щодо цього простору подій, і яка б мала змогу достатньо точно прогнозувати відповіді для нових прикладів.

Обчислювальний аналіз алгоритмів машинного навчання та їх продуктивності — це галузь теоретичної інформатики, відома як теорія обчислювального навчання за допомогою моделі Probably Approximately Correct Learning (PAC). Оскільки навчальні набори скінченні, а майбутнє невизначене, теорія навчання зазвичай не дає гарантій продуктивності алгоритмів. Натомість імовірнісні межі продуктивності є досить поширеними. Розклад дисперсії є одним із способів кількісної оцінки помилки узагальнення.

Для найкращої продуктивності в контексті узагальнення складність гіпотези повинна відповідати складності функції, що лежить в основі даних. Якщо гіпотеза менш складна, ніж функція, то модель недостатньо підігнала дані. Якщо у відповідь збільшується складність моделі, то похибка навчання зменшується. Але якщо гіпотеза занадто складна, то модель піддається переобладнанню, і узагальнення буде гіршим. [4]

На додаток до меж продуктивності, теоретики навчання вивчають часову складність і доцільність навчання. У теорії обчислювального навчання обчислення вважаються можливими, якщо їх можна виконати за поліноміальний час. Існують два види результатів часової складності: Позитивні результати показують, що певний клас функцій можна вивчити за поліноміальний час. Негативні результати показують, що деякі заняття неможливо вивчити за поліноміальний час.

1.1.2 Що таке глибинне навчання

Глибинне навчання — це галузь машинного навчання, яка повністю заснована на штучних нейронних мережах. Нейронна мережа побудована таким чином, що нагадує роботу людського мозку, тому можна сказати, що глибинне навчання є свого роду імітацією людського мозку. Концепція глибинного навчання не є новою, вперше ці ідеї були описані Френком Розенблатом у 1962-

му році [1]. Проте нині це почало викликати великий ажіотаж, тому що почали з’являтися необхідні потужності для обробки великої кількості даних.

Формальне визначення глибинного навчання наступне — це особливий вид машинного навчання, який досягає великої потужності та гнучкості, навчаючись представляти світ у вигляді вкладеної ієрархії понять, причому кожне поняття визначається стосовно простіших понять, а більш абстрактні уявлення обчислюються в термінах менш абстрактних.

У мозку людини приблизно 100 мільярдів нейронів, кожен з яких пов’язаний через тисячу своїх сусідів. І виникає питання, як відтворити таку структуру в комп’ютері. Ідея полягає в створенні такої штучної структури під назвою штучна нейронна мережа, яка буде складатися пов’язаними між собою вузлами, або нейронами; у цій структурі вузли згруповані в шари; деякі вузли призначені для вхідних значень, тобто формують вхідний шар, деякі — для вихідних значень, тобто формують вихідний шар, а між ними може бути багато вузлів в прихованому шарі.

1.1.3 Глибинні нейронні мережі

Глибинна нейронна мережа (далі ГНМ) — це більш складна нейронна мережа, яка складається з кількох прихованих шарів (рис. 1). Це ускладнення дозволяє вирішувати більш об’ємний спектр задач.

ГНМ складається з нейронів, синапсів, ваг, зміщень, і функцій активації. Ці компоненти, як вже зазначалося, функціонують подібно до людського мозку і можуть бути навчені, як і будь-який інший алгоритм машинного навчання.

ГНМ можуть моделювати складні нелінійні зв'язки. Архітектури ГНМ генерують композиційні моделі, де об'єкт виражається як шарувата композиція примітивів.[6] Додаткові шари дозволяють компонувати об’єкти з нижчих шарів, потенційно моделюючи складні дані з меншою кількістю одиниць, ніж дрібна мережа з такою ж продуктивністю.[7] Наприклад, було доведено, що розріджені багатовимірні поліноми експоненціально легше апроксимувати з ГНМ, ніж з неглибокими мережами.[8]

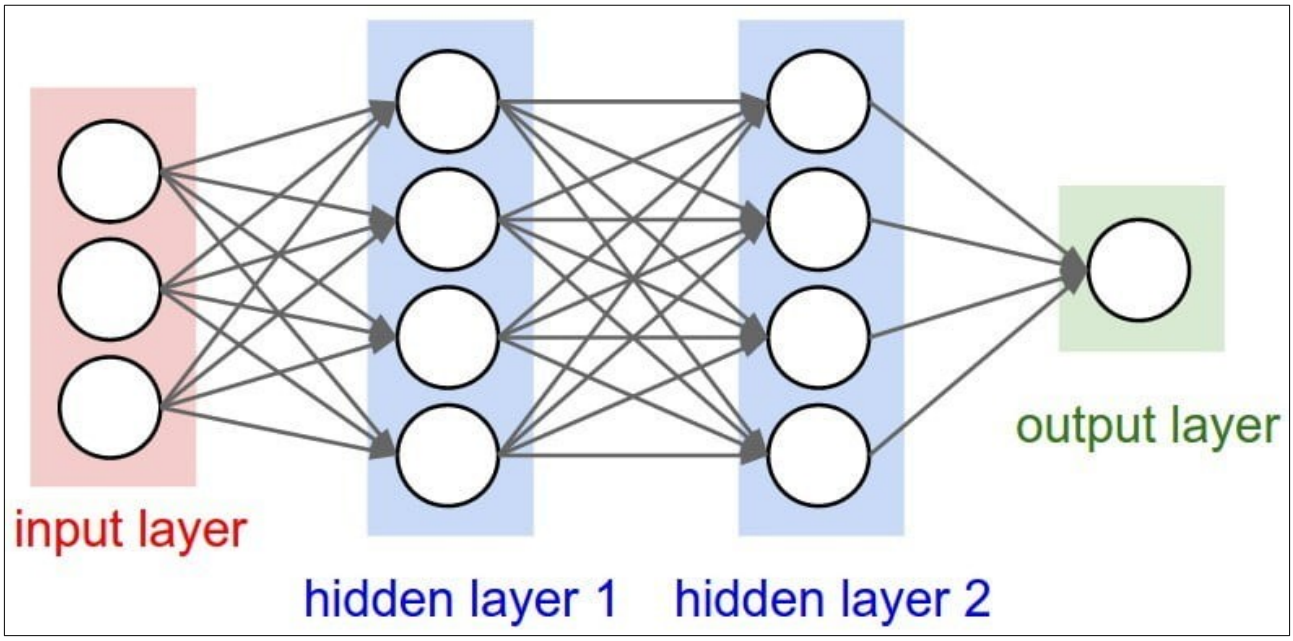


Рисунок 1 — Приклад найпростішої глибинної мережі

Глибокі архітектури включають багато варіантів кількох основних підходів. Кожна архітектура досягла успіху в певних областях. Не завжди можливо порівняти продуктивність кількох архітектур, якщо вони не були оцінені на одних і тих самих наборах даних.

ГНМ – це, як правило, мережі з прямим зв’язком, в яких дані переходять з вхідного рівня на вихідний без циклу назад. Спочатку ГНМ створює карту віртуальних нейронів і призначає випадкові числові значення, або «ваги», зв’язкам між ними. Коефіцієнти ваги та вхідні дані перемножуються, а вихідні дані повертаються від 0 до 1. Якщо мережа не розпізнає точно певний шаблон, алгоритм коригує їх. [9] Таким чином, алгоритм може зробити певні параметри більш впливовими, поки не визначить правильні математичні маніпуляції для повної обробки даних.

Тут і далі будуть використовуватися наступний опис структури ГНМ.

w^l_{jk} — позначення ваги зв’язку від k-го нейрона в (l-1)-му шарі до j-го нейрона в l-му шарі (рисунок 2.а).

b^l_j — позначення зміщення нейрона в l-му шарі (рисунок 2.б).

a^l_j — позначення активації j-го нейрона в l-му шарі (рисунок 2.б).

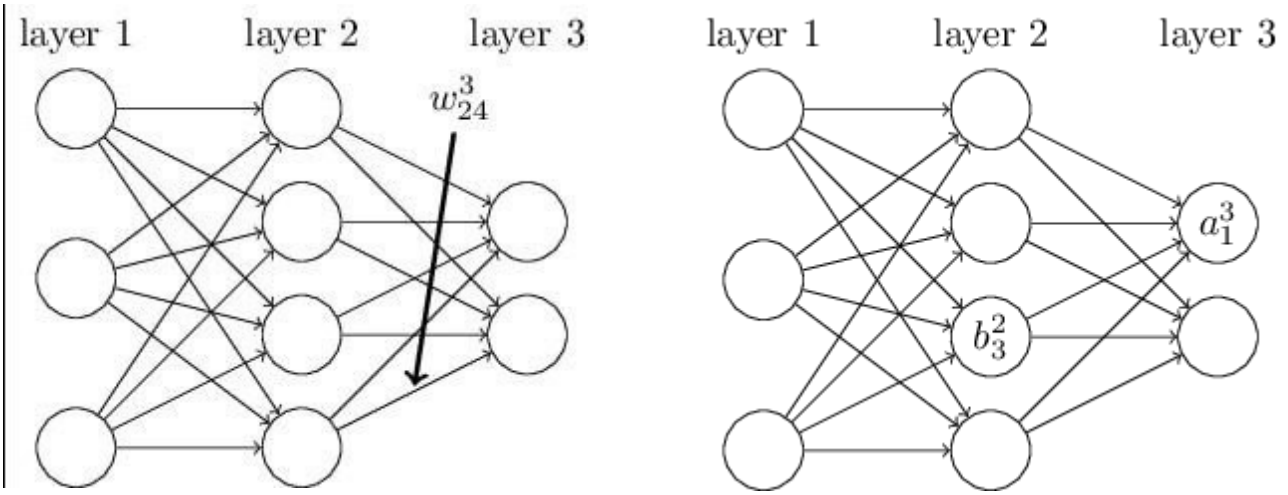


Рисунок 2. а — Позначення зв’язку між нейронами в прихованому шарі та вихідному шарі; б — Позначення зміщення нейрона в прихованому шарі

Тоді:

$$a^l_j = \sigma\left(\sum_k w^l_{jk} a^{l-1}_k + b^l_j\right)$$

1.1.4 Метод зворотного поширення помилки

Метод зворотного поширення помилки (далі МЗПП) — широко використовуваний алгоритм навчання ГНМ із прямим зв’язком. Цей метод полягає в ефективному обчисленні градієнта функції втрат відносно ваг мережі для одного прикладу введення-виведення, на відміну від наївного прямого обчислення градієнта щодо кожної ваги окремо. Ця ефективність робить можливим використання градієнтних методів для навчання багатошарових мереж, оновлюючи вагові показники, щоб мінімізувати втрати. Зазвичай використовуються градієнтний спуск або такі варіанти, як стохастичний градієнтний спуск. МЗПП працює шляхом обчислення градієнта функції втрат щодо кожної ваги за ланцюговим правилом, градієнт обчислюється по одному шару за раз ітеративно, починаючи від останнього шару, щоб уникнути зайвих обчислень проміжних членів у правилі ланцюга; це приклад динамічного програмування. [10]

В основі зворотного поширення лежить вираз для часткової похідної $\partial C/\partial w$ функції вартості C відносно будь-якої ваги w (або зміщення b) у мережі.

Вираз говорить нам, як швидко змінюється вартість, коли ми змінюємо ваги та зміщення.

1.1.5 Проблема локального мінімуму

Глобальний мінімум — це точка, в якій функція приймає мінімальне значення. При мінімізації функції за допомогою алгоритмів оптимізації, таких як градієнтний спуск, як у випадку МЗПП, може статися так, що функція може мати мінімальне значення в різних точках. Точки, які здаються мінімумами, але не є точкою, де функція насправді приймає мінімальне значення, називаються локальними мінімумами (рис. 3).

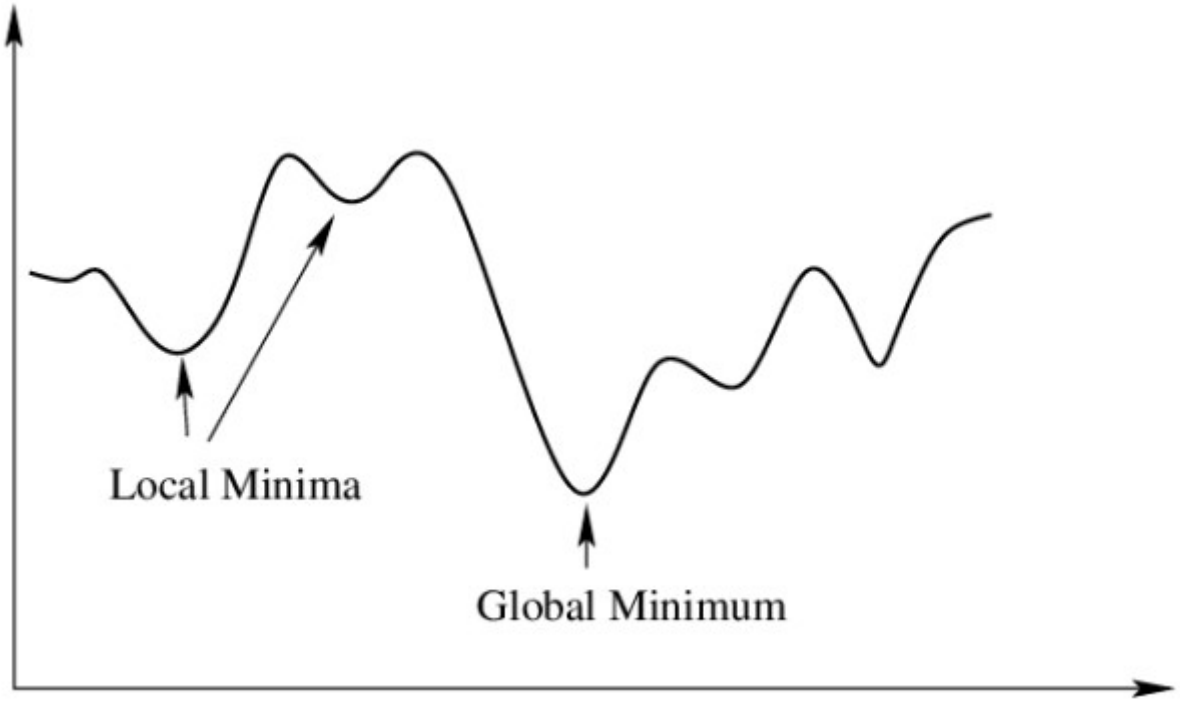


Рисунок 3 — Приклад локального мінімуму

Алгоритми машинного навчання, такі як алгоритми градієнтного спуску, мають ризик застрягти в локальних мінімумах під час навчання моделей, що призведе до низької точності мережі та поганих результатів. Алгоритм оптимізації градієнтний спуск зазвичай знаходить локальні мінімуми більшу частину часу, замість глобальних мінімумів, оскільки градієнт вказує в напрямку локалького спуску, а не в напрямку найкрутішого глобального спуску. Сучасні методи пошуку глобальних мінімумів вимагають або надзвичайно

великої кількості ітерацій, або великої кількості випадкових перезапусків для гарної продуктивності. Глобальні проблеми оптимізації також можуть бути досить складними, коли між локальними мінімумами існують високі бар'єри втрат.

1.1.6 Оптимізатори

Зазвичай формула для звичайного градієнтного спуску має наступний вигляд:

$$\Delta\theta = -\eta \nabla_\theta J(\theta)$$
$$\theta = \theta + \Delta\theta = \theta - \eta \nabla_\theta J(\theta)$$

де θ — параметри мережі;

$J(\theta)$ — функція втрат;

η — швидкість навчання.

Проблема полягає в складності виразу $\nabla_\theta J(\theta)$, адже кожна вага виступає змінною в $J(0)$. Така велика кількість ступіней свободи призводить до наступних проблем:

- Велика кількість локальних мінімумів або сідлових точок;
- Складний ландшафт цільової функції: плато чергуються із регіонами сильної нелінійності. Похідна на плато близька до нуля, а раптовий урвище навпаки може відправити нас занадто далеко;
- Занадто мала швидкість навчання змушує алгоритм сходитися дуже повільно та зупинятися в точках локального мінімуму, занадто велика швидкість — пропускати глобальні мінімуми або зовсім змушує алгоритм розходитись. [11]

Вирішення цих проблем полягає в ідеї накопичення імпульсу. Неформально кажучи, якщо ми якийсь час рухаємось у певному напрямку, то скоріш за все треба рухатись у тому ж напряму ще деякий час у майбутньому.

Юрій Нестеров запропонував метод швидкого градієнта, ідея якого описується наступною формулою [12] :

$$v_t = \alpha v_{t-1} + (1-\alpha) x$$

Щоб накопичити що-небудь, множиться вже накопичене значення на коефіцієнт збереження $0 < a < 1$ та додається чергова величину, помножена на $(1 — a)$. Чим ближче a до одиниці, тим більше вікно накопичення і сильніше згладжування — історія x починає впливати сильніше, ніж кожне чергове x. Якщо $x = 0$ то починаючи з якогось моменту v_t загасає за геометричною прогресією, експоненційно. Застосовуючи цей метод до МЗПП отримуємо наступні формули оновлення мережі:

$$v_t = \alpha v_{t-1} + \eta \nabla_\theta J(\theta)$$
$$\theta = \theta - v_t$$

Зазвичай a береться порядка 0.9. Важливо зазначити, що $1 — a$ не зникло, воно включилось в η.

Іншим рішенням вищезазначених проблем є застосування оптимізатора Adagrad — adaptive gradient. Ідея цього оптимізатора полягає на припущенні, що деякі ознаки можуть бути дуже інформативними, але зустрічатися рідко [13]. І потрібно вміти оновляти параметри нейронної мережі, зважаючи на те, наскільки типовою є ознака, яку фіксує параметр. Щоб цього досягти, пропонується для кожного параметру мережі зберігати сумму квадратів його оновлень. Таким чином, якщо параметр належить до ланцюга нейронів, що часто активуються, то він часто активується, а значить сума швидко накопичується. Тоді формула оновлення параметрів буде мати наступний вигляд:

$$G_t = G_t + g_t^2$$
$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{G_T + e}} g_t \quad (1)$$

де G_t — сума квадратів оновлень;

e — параметр, що зглажує, щоб уникнути ділення на 0.

Недолік Adagrad в тому, що G_t в (1) може збільшуватися скільки завгодно, що через деякий час призводить до занадто маленьких оновлень і паралічу алгоритму. RMSProp покликаний виправити цей недолік.

Модифікуючи ідею Adagrad, меншою мірою оновлюються ваги, які занадто часто оновлюються, але замість повної суми оновлень, використовується усереднений з історії квадрат градієнта. Знову використовується експоненційно загасаюче середнє [14].

Нехай $E[g^2]_t$ — середнє, що біжить, в момент t

$$E\left[g^2\right]_t = \alpha E\left[g^2\right]_{t-1} + (1-\alpha) g_t^2$$

тоді замість (1) отримаємо

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{E\left[g^2\right]_t + e}} g_t$$

Знаменником є корінь із середніх квадратів градієнтів, звідси RMSProp — root mean square propagation.

$$RMS[g]_t = \sqrt{E\left[g^2\right]_t + e}$$

Adam – adaptive moment estimation, ще один алгоритм оптимізації. Він поєднує у собі ідею накопичення руху та ідею слабшого оновлення ваг для типових ознак. Саме цей оптимізатор буде використовуватися при розробці нейронних мереж в рамках цієї роботи.

Від Нестерова Adam відрізняється тим, що накопичується не $\Delta\theta$, а значення градієнта. Крім того, ми хочемо знати, як часто змінюється градієнт. Автори алгоритму запропонували для цього оцінювати ще й середню нецентровану дисперсію:

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

1.2 Диференціальні рівняння

Диференціальні рівняння — це рівняння, у яких встановлюється залежність між однією чи багатьма невідомими функціями та їх похідними. Зазвичай функції є представленням якихось фізичних величин, похідні представляють їхню швидкість зміни, і саме диференціальне рівняння встановлює зв’язок між ними.

Порядком диференціального рівняння називають найвищий порядок похідної, що зустрічається в рівнянні.

Рішенням диференціального рівнняня порядку *n* на інтервалі *(a, b)* називають будь-яку функцію *y(x)*, яка задовольняє розглянутому диференціальному рівнянню на інтервалі *(a, b*) та є *n* разів диференційованою на цьому інтервалі.

Початковими, або граничними, умовами називають набір умов, які додатково накладаються на рішення рівняння та які дозволяють визначити шукане рішення. За таких умов розв’язок може бути один. Іншими словами, початкові умови — це значення розв’язку та/або його похідної або похідних у конкретних точках.

Звичайне диференціальне рівняння — це рівняння, яке включає лише звичайні похідні. Тобто рівняння виду $F\left(x,\phi(x),\nabla\phi(x),\ldots,\nabla^{j}\phi(x)\right)=0.$

Задача Коші — одна з основних задач теорії диференціальних рівнянь. Задача полягає в вирішенні звичайного диференціального рівняння разом із початковою умовою, яка визначає значення невідомої функції в даній точці області визначення. Зазвичай ця точка є початковою точкою області визнечення.

1.3 Традиційні числові методи рішення диференціальних рівнянь

1.3.1 Метод Рунге-Кутта

Метод Рунге-Кутта є ефективним та широко використовуваним методом розв’язування задачі Коші диференціальних рівнянь.

Нехай треба вирішити задачу Коші диференціального рівняння першого порядку

$$\begin{cases} y'=f(x,y), a\le x\le b \\ y(a)=y_0 \end{cases}$$

Щоб застосувати метод Рунге-Кутта, інтервал *[a, b]* розділяється на *N* підінтервалів як $[x_n, x_{n+1}]$ і використовуючи теорему про середнє значення для інтегралів, отримуємо

$$y(x_{n+1})-y(x_n)=\int_{x_n}^{x_{n+1}} f(x,y(x))\,dx=hf(\xi,y(\xi))$$

де $h = x_{n+1}-x_n$, ξ належить $[x_n,x_{n+1}]$, тобто

$$y(x_{n+1})=y(x_n)+hf(\xi,y(\xi))$$

Якщо апроксимути $f(\xi,y(\xi))$ значеннями лінійної комбінації $f(\xi_1,y(\xi_1))$, $f(\xi_2,y(\xi_2))$, …, $f(\xi_m,y(\xi_m))$ на інтервалі $[x_n,x_{n+1}]$, то прийдемо до загальної форми методу Рунге–Кутта [19]:

$$y_{n+1}=y_n+h\sum_{i=1}^{m}c_i f\left(\xi_i,y\left(\xi_i\right)\right)$$

Вибираючи різні значення параметрів m, c_i та ξ_i, ми можемо отримати різну форму розрахункової формули Рунге–Кутта; можна отримати формулу розрахунку Рунге–Кутта вищого порядку, вибравши відповідні значення параметрів. Найбільш поширеною є наступна формула Рунге-Кутта

$$\begin{cases} y_{n+1}=y_n+\frac{1}{6}\left(K_1+2K_2+2K_3+K_4\right) \\ K_1=hf\left(x_n,y_n\right) \\ K_2=hf\left(x_n+\frac{1}{2}h,y_n+\frac{1}{2}K_1\right) \\ K_3=hf\left(x_n+\frac{1}{2}h,y_n+\frac{1}{2}K_2\right) \\ K_4=hf\left(x_n+h,y_n+K_3\right) \end{cases}$$

Метод у такому вигляді називають методом Рунге-Кутта четвертого порядку.

1.3.2 Метод Адамса-Бешфорта

Розглянемо загальне диференціальне рівняння

$$\int_D f\left(\overline{x}\right)d\,\overline{x}\approx\sum_{i=1}^{N}w_i f\left(\overline{x^i}\right)$$

Нехай вже відомі розв’язки на множині значень X_i $(i=0, 1, ..., n)$. Тоді можна записати рівняння

$$y_{n+1}=y_n+\int_{t_n}^{t_{n+1}}\frac{dy}{dt}dt=y_n+\int_{t_n}^{t_{n+1}}f\left(y,t\right)dt$$

Методи Адамса засновані на ідеї апроксимації підінтегрального виразу поліномом у межах інтервалу (x_n, x_{n+1}). Використання полінома k-го порядку призводить до методу $k+1$-го порядку. Існує два типи методів Адамса: явний і неявний. Явний тип називається методами Адамса-Башфорта (АВ), а неявний — методами Адамса-Моултона (АМ).

Методи Адамса-Башфорта першого порядку є просто прямим і зворотним методами Ейлера відповідно. Досить популярні версії цих методів другого порядку (отримані за допомогою лінійного інтерполанта). Метод Адамса-Башфорта другого порядку (AB2) визначається за формулою

$$y_{n+1} = y_n + \frac{h}{2}\left(3f\left(y_n, t_n - f\left(y_{n-1}, t_{n-1}\right)\right)\right)$$

Важливо зауважити, що метод AB2 є явним, а тому лише умовно стабільним.

Метод Адамса-Моултона другого порядку (AM2) — це неявна техніка, яку іноді називають правилом трапеції. AM2 визначається за наступною формулою

$$y_{n+1} = y_n + \frac{h}{2}\left(f\left(y_{n+1}, t_{n+1} + f\left(y_n, t_n\right)\right)\right)$$

1.4 Знаходження наближеного розв’язку диференціального рівняння за допомогою глибинного навчання

Алгоритм буде описуватися за тим припущенням, що система з m диференціальних рівнянь вирішується ГНМ з m входами, m виходами та одним прихованим шаром, який складається з *k* вузлів. В такому випадку вихід ГНМ N_m можна описати наступною формулою:

$$N_m\left(\vec{x}, \backslash\{w, \vec{b}\backslash\}\right) = \sum_{k,n} w_{mk}^f \, g\left(w_{kn}^h x_n + b_k^h\right) + b_m^f$$

де *g* — це функція активації $g : R^k \text{ -> } R^k$, яка застосовується до кожного нейрона поелементно.

Система з *m* диференціальниї рівнянь може бути представлена в наступній загально прийнятій формі:

$$F_m\left(x, \phi_m(x), \nabla\phi(x), \ldots, \nabla^j \phi_m(x)\right) = 0 \quad (2)$$

де $\phi_m(x)$ — це шукана функція;

$\nabla^j \phi_m(x)$ — це похідна *j* порядку. Також до системи йдуть граничні, або як ще прийнято їх називати — початкові, умови. Завдяки такому запису системи задача може бути інтерпретована як задачі мінімазації: рішення $\phi_m(x)$ потрібно апроксимувати таким чином, щоб квадрат лівої частини (2) був мінімальним.

Таким чином, квадрат лівої частини (2) можна вважати функцією втрат, або цільовою функцією. Тоді апроксимація функції $\phi_m(x)$ буде дорівнювати виходу нейронної мережі N_m. В такому випадку можна зробити початкові умови частиною цільової функції, а саме до квадрату лівої частини (2) додавати суму квадратів різниць між значенням функції в точці з граничних умов та тим значенням, яке насправді повинно бути. Якщо з інтервалу, на якому повинна бути знайдена апроксимація необхідної функції, взяти скінченну кількість точок, зазначивши їх як тренувальну вибірку x^i, тоді апроксимація може бути знайдена шляхом підрахунку набору ваг та зсувів $\{w;\ b\}$ таких, що цільова функція буде набувати найменшого значення. Для тренувальнох вибірки з i_{max} точок цільова функція набуває наступного вигляду

$$L\left(\backslash\{ w, \vec{b} \backslash\}\right) = \frac{1}{i_{max}} \sum_{i,m} \widehat{F_m}\left(\vec{x^i}, \widehat{\phi_m}\left(\vec{x^i}\right), \ldots, \nabla^j \widehat{\phi_m}\left(\widehat{x^i}\right)\right)^2 + \sum_{B.C.} \left(\nabla^p \widehat{\phi_m}\left(\vec{x_b}\right) - K\left(\vec{x_b}\right)\right)^2$$

де друга сума позначає квадрат помилки початкових умов. Це можуть бути умови Діріхле або Неймана. [15]

2 ВИКОРИСТАННЯ НЕЙРОМЕРЕЖ ДЛЯ ЗНАХОДЖЕННЯ НАБЛИЖЕНОГО РОЗВ’ЯЗКУ ДИФЕРЕНЦІАЛЬНИХ РІВНЯНЬ

2.1 Розробка глибинної нейронної мережі

Програмний код розроблявся на мові програмування Python. Підставами для цього є зручні інструменти для математичних обрахунків, а також бібліотеки, оптимізовані для роботи з алгоритмами машинного навчання. [16] Зокрема використовувалося розширення мови Python, що додає підтримку великих багатовимірних масивів і матриць, разом з великою бібліотекою високорівневих математичних функцій для операцій з цими масивами — Numpy. А також бібліотека програмного забезпечення з відкритим вихідним кодом для машинного навчання та штучного інтелеку — Tensorflow. [17]

Перш за все для інкапсуляції даних про систему диференціальних рівнянь було створено клас ODESystem. Цей клас зберігає наступну інформацію про систему:

1. Інтервал, на якому апроксимується система.
2. Початкові умови системи.
3. Масив диференціальних рівнянь у вигляді масиву функціональних об’єктів

Основна логіка алгоритму апроксимації рішення диференціальних рівнянь винесена в клас ODESolver. Цей клас зберігає інформацію про систему диференціальних рівнянь у вигляді вище зазначеного класу ODESystem, а також саму ГНМ, яка вирішує поставлену задачу, та обраний оптимізатор для алгоритму навчання.

Нейронна мережа створюється при конструкції класу за інформацією про архітектуру, яку було передано до класу. Ця інформація складається з кількості слоїв у нейроннії мережі, кількості нейронів у кожному слої, а також використовувана функція активації. Нейронна мережа є об’єктом класу tf.keras.Model. [18]

Головний метод інтерфейсу цього класу є метод train, у якому відбувається навчання нейронної мережі. Навчання мережі продовжується доти, поки величина цільової функції перевищує значення, яке було передано аргументом у метод. У методі train викликається службовий приватний метод train_step. Логіка цього методу описується діаграмою на рисунку 5.

2.2 Визначення найкращої архітектури

2.2.1 Залежність часу навчання від функції активації

З самого початку проводилося дослідження кращої функції активації, за якої б навчання нейронної мережі відбувалося за найменший час. Для дослідження обиралися наступні функції активації:

- sigmoid — сигмовидна функція, також називається функцією здавлення, оскільки її область визначення є множиною всіх дійсних чисел, а областю значень є інтервал (0, 1). Визначається формулою (3) та графіком на рисунку 6.

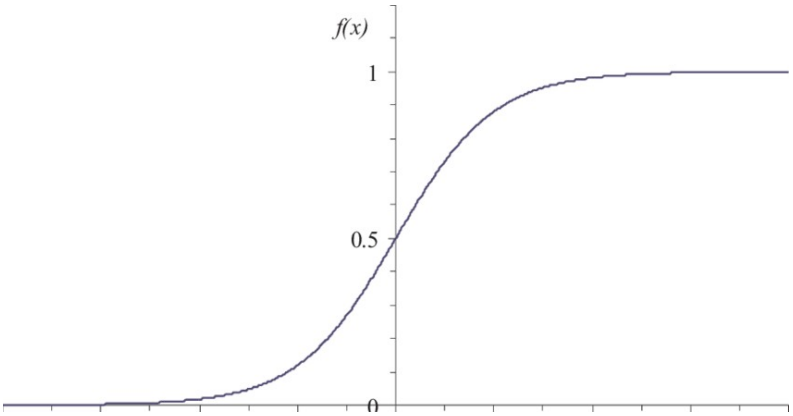


Рисунок 6 — графік сигмовидної функції

$$f(x)=\frac{1}{1+e^{-x}} \quad (3)$$

- swish — функція активації, яка визначається наступною формулою

$$f(x)=x * sigmoid(\beta x)$$

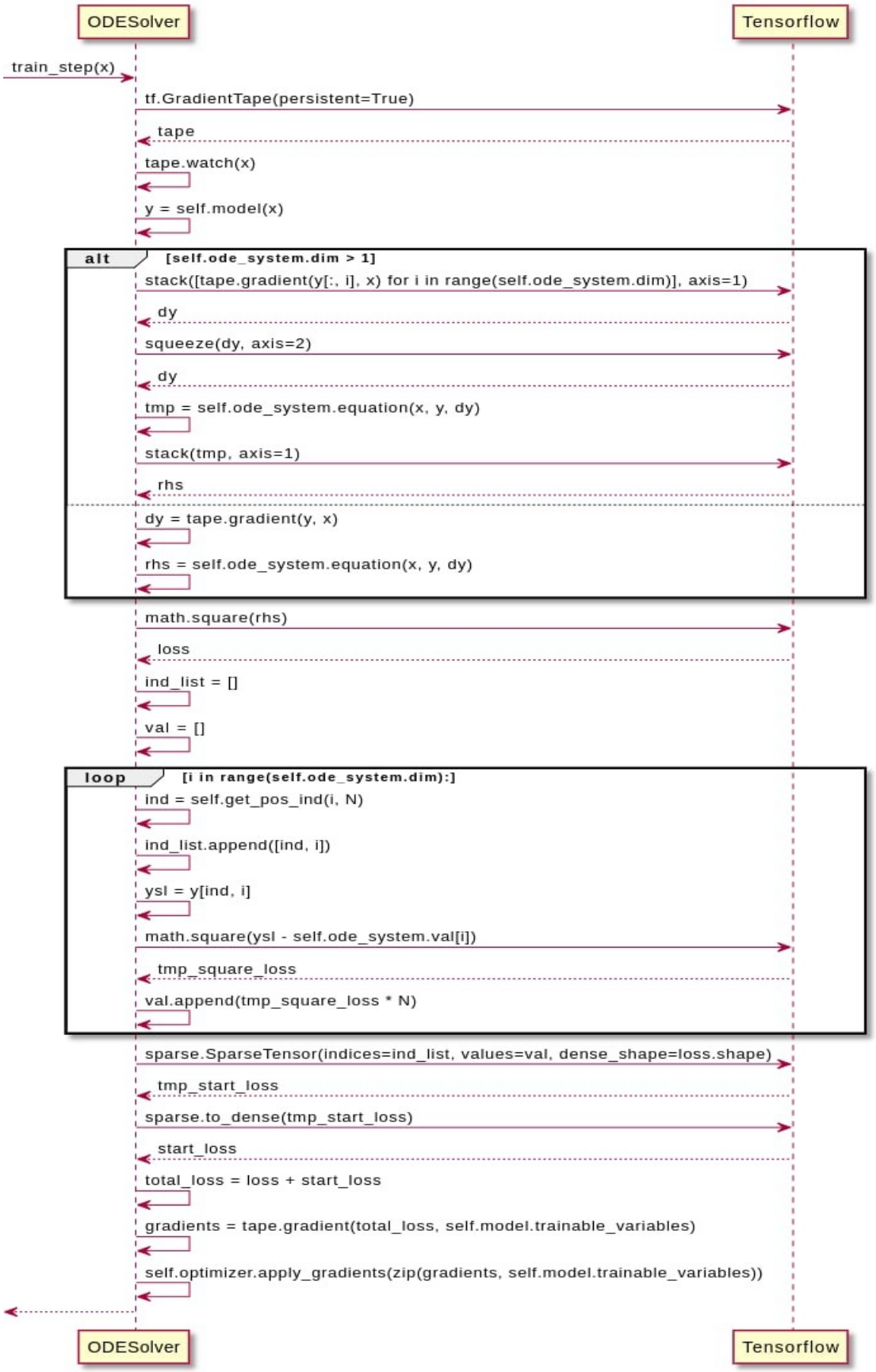


Рисунок 5 — Логіка методу train_step

де β — гіперпараметр, проте він рідко визначається при навчанні, і зазвичай дорівнює 1 [20]; має графік, визначений на рисунку 7.

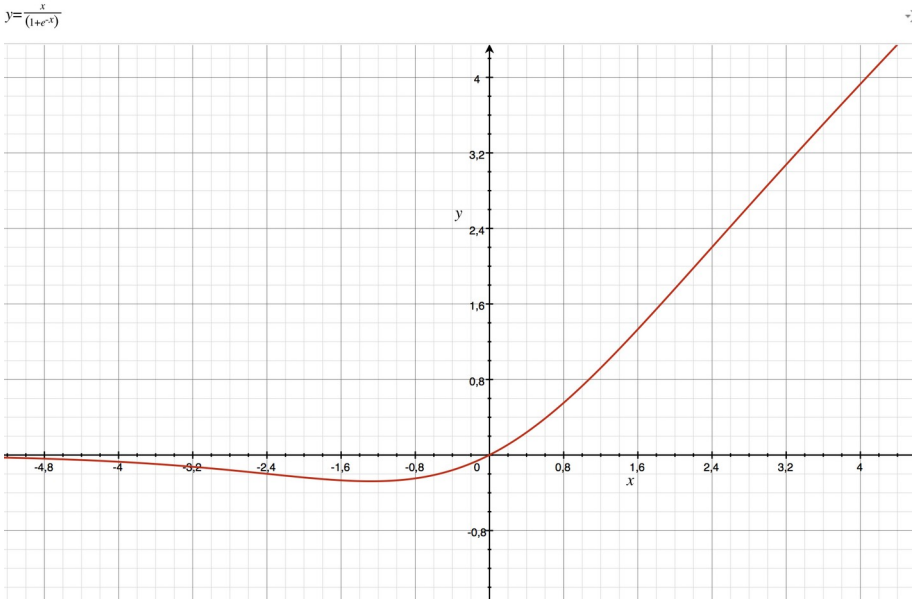


Рисунок 7 — графік функції активації swish

- selu — масштабовані експоненціальні лінійна одиниця, функція активації визначається формулою (4) та графіком на рисунку 8.

$$\begin{cases} f(x)=\lambda x \, if \; x \geq 0 \\ f(x)=\lambda x\left(e^{x}-1\right) if \; x<0 \end{cases} \quad (4)$$

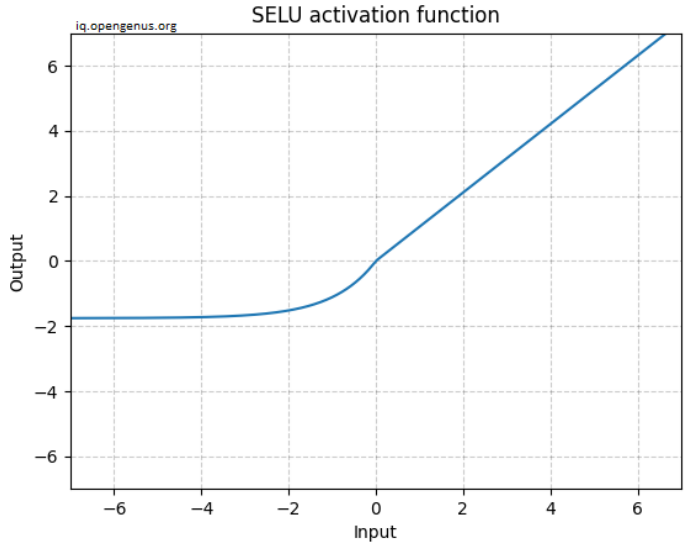


Рисунок 8 — графік функції активації selu

- elu — експоненціальна лінійна одиниця, визначається формулою (5) та графіком на рисунку 9.

$$\begin{cases} f(x) = x \, if \, x > 0 \\ \lambda(\exp(x) - 1) \, if \, x \le 0 \end{cases} \quad (5)$$

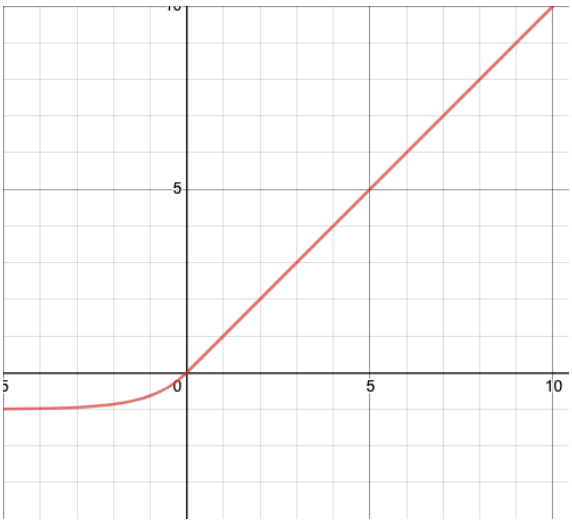


Рисунок 9 — графік функції активації elu

- hard sigmoid — жорстка сигмовидна функція, визначається формулою (6), на рисунку 10 зображується порівняння функцій активації sigmoid та hard sigmoid.

$$f(x) = max\left(0, min\left(1, \frac{(x+1)}{2}\right)\right) \quad (6)$$

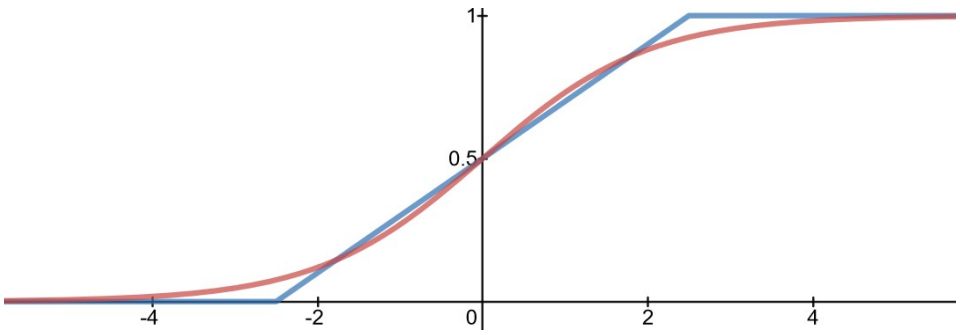


Рисунок 10 — порівняння sigmoid та hard sigmoid

- softplus — визначається формулою (7) та графіком на рисунку 11.

$$f(x) = \log(1 + \exp(x)) \quad (7)$$

- tanh — дотична гіперболічна функція, визначається за формулою (8) та графіком на рисунку 12.

$$f(x)=\frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}=\frac{2}{1+e^{-2x}}-1 \quad (8)$$

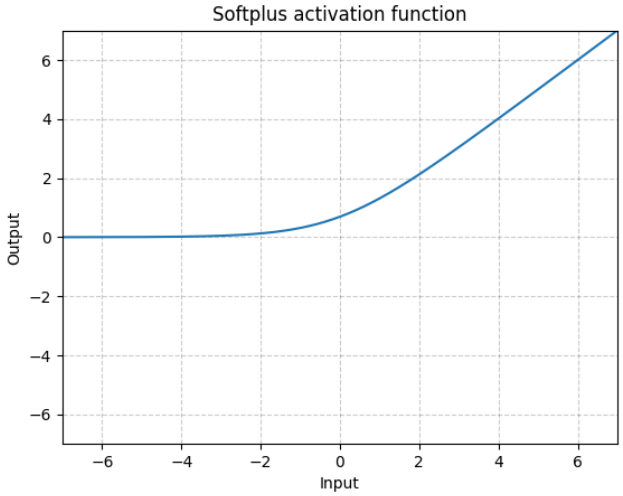


Рисунок 11 — графік функції активації softplus

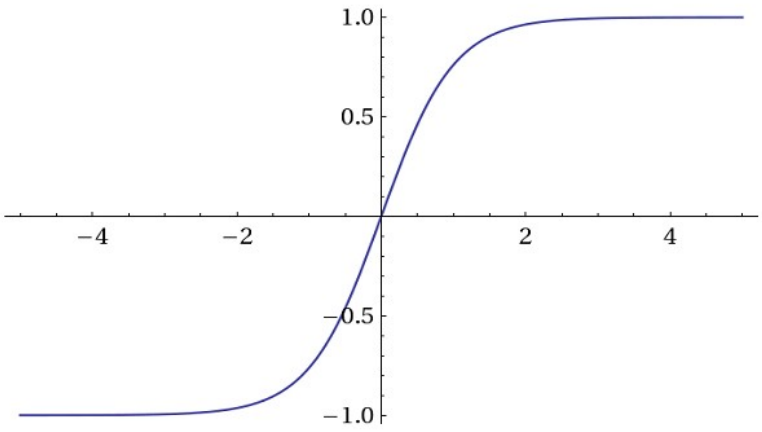


Рисунок 12 — графік функції активації tanh

Дослідження проводилося на нейроннії мережі, яка вирішувала наступну систему диференціальних рівнянь:

$$\begin{cases} x_0' = x_1 \\ x_1' = 6x_0^2 \\ x_0(1)=1 \\ x_1(1)=-2 \\ 1 \le t \le 2 \end{cases} \quad (9)$$

Система вирішувалася ГНМ, яка складалася з 5 слоїв по 10 нейронів кожний. Для більшої об’єктивності оцінки нейронна мережа навчалася

незалежно декілька разів. Рахувався середній час, за який нейронна мережа навчалася, тобто за який значення цільової функції більше не перевищувало задане значення, яке тут і далі буде називатися точністю навчання. При цьому дослідженні це значення дорівнювало 10^{-3}. Так само враховувалася середня кількість ітерацій навчання, необхідна для навчання мережі з заданою точністю.

2.2.2 Залежність часу навчання від архітектури

Після дослідження найоптимальніших функцій активацій, проводилося дослідження найкращої архітектури нейронної мережі, за якої б час навчання був найменшим. Дослідження проводилося методом проб і помилок [21]. Для дослідження обиралася та сама система диференціальних рівнянь (9).

Для визначення кращої архітектури створювалася ГНМ з певною архітектурою та визначався середній час, за який мережа навчалася вирішувати систему диференціальних рівнянь (43) з точністю 10^{-3}. Так само визначалася середня кількість ітерацій навчання.

Архітектура нейронної мережі визначалась кількістю слоїв, кількістю нейронів у слоях та функцією активації. Граничні кількості слоїв та нейронів визначались заздалегідь, і дорівнювали приблизно інтервалу *[6, 18]* для кількості слоїв та інтервалу *[5, 25]* для кількості нейронів у слоях. Функції активації в архітектурі були обрані серед тих, що показали найкращий результат у попередньому дослідженні.

Отримані архітектури, за яких нейромережа навчалась найшвидше, порівнювались з традиційними числовими методами знаходження розв’язку диференціальних рівнянь, а саме з методом Рунге-Кутта 4 порядку та методом АБ2.

2.3 Порівняння алгоритму глибинного навчання з традиційними числовими методами

Традиційні числові методи, такі як метод Рунге-Кутти та АБ2, попри швидкість знаходження розв’язку диференціальних рівнянь, можуть використовуватися лише для вирішення задач Коші. Цей недолік є вагомим у багатьох задачах теорії диференціальних рівнянь, коли граничні умови задані не

в початковій точці інтервалу, на якому треба знайти розв’язок. На противагу цьому алгоритми глибинного навчання цього недоліку не мають, і можуть вирішувати задачу диференціальних рівнянь з будь-якими заданими початковими умовами.

Нехай потрібно вирішити наступну систему диференціальних рівнянь з визначеними початковими умовами та інтервалом значень незалежної змінної

$$\begin{cases} x' = x - y + 2\sin(t) \\ y' = 2x - y \end{cases}$$
$$x(0) = 1$$
$$y(0) = 1$$
$$-1 \le t \le 2$$

Традиційні числові методи для знаходження розв’язку системи диференціальних рівнянь не мають змоги розв’язати цю систему, у той час коли алгоритми глибинного навчання показують достатньо точний результат, ілюстрацію якого можна побачити на рисунку 13. На рисунку результат навчання ГНМ порівнюється зі вже відомим аналітичним розв’язком

$$x = \cos(t) + \sin(t) + t\sin(t) - t\cos(t)$$
$$y = 3\sin(t) - 2t\cos(t) + \cos(t)$$

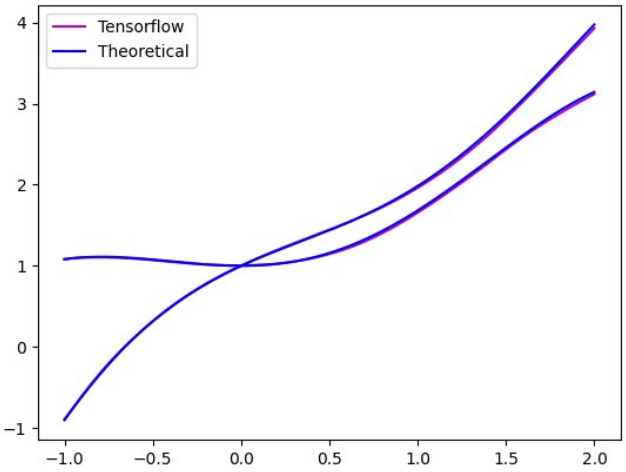
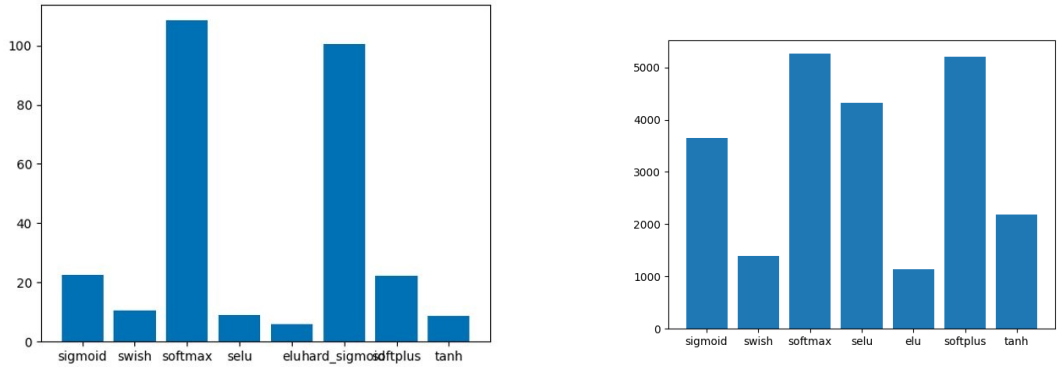


Рисунок 13 — порівняння результатів навчання ГНМ з аналітичним розв’язком

3 РЕЗУЛЬТАТИ

3.1 Оптимальні функцій активації

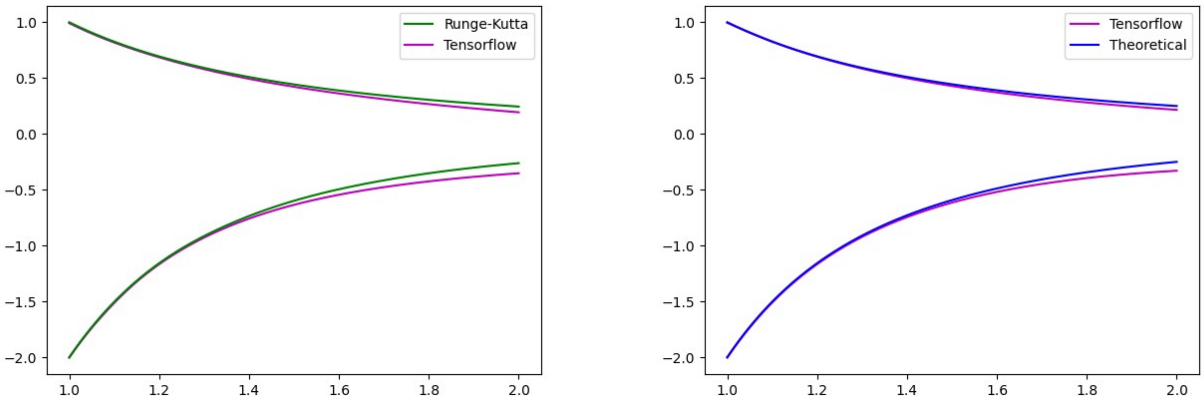
Шляхом дослідження описаного в розділі 2.2.1 було визначено наступні оптимальні для використання функції активації: swish, selu, elu та tanh. Це можна спостерігати по графікам на рисунку 14.а та на рисунку 14.б. Як можна бачити з графіків, кращий результат у порівнянні з іншими показує функція активації elu. Така ж тенденція буде спостерігатися й при визначенні кращої архітектури.



а) б)

Рисунок 14 — Результати запуску навчання нейронної мережі з різними функціями активації: а) Залежність часу навчання в секундах від функції активації; б) Залежність кількості ітерацій навчання від функції активації

На рисунку 15.а можна спостерігати порівняння результатів роботи алгоритмів глибинного навчання з використанням функції активації elu та методу Рунге-Кутти 4 порядку при вирішуванні системи диференціальних рівнянь (3). На рисунку 15.б можна спостерігати порівняння результату роботи нейромережі з аналітичним розв’язком (3).



а) б)

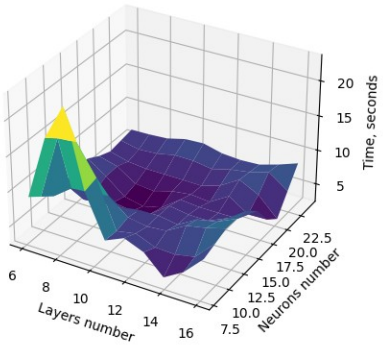
Рисунок 15 — порівняння роботи ГНМ з функцією активації elu з: а) числовим методом РК4; б) аналітичним рішенням

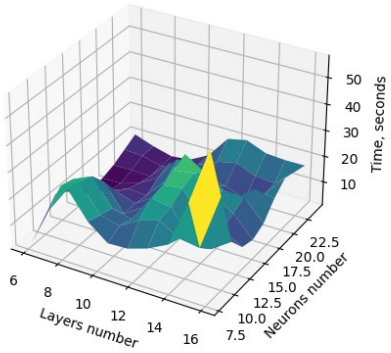
3.2 Краща архітектура нейромережі

Результати дослідження, яке було описане в розділі 2.2.2 можна побачити на рисунку 16. Як можна бачити з графіків, краще за всіх себе поводить (у цьому контексті мається на увазі — показує найменший час навчання) архітектура з функцією активації elu — найкращий результат у вигляді часу, за який нейромережа навчилась апроксимувати розв’язок диференціального рівняння, 2.19 секунди досягається при використанні 10 слоїв по 16 нейронів кожен. Менш гладка функція залежності часу від архітектури нейромережі спостерігається при використанні функції активації selu — найкращий результат 2.06 секунди досягається при використанні 6 слоїв по 8 нейронів кожен. Кращий результат для архітектур з використанням функцій активації swish та tanh — 6.22 секунди та 4.13 секунди відповідно — досягається при використанні відповідно 12 слоїв по 12 нейронів кожний та 14 слоїв по 8 нейронів кожний.

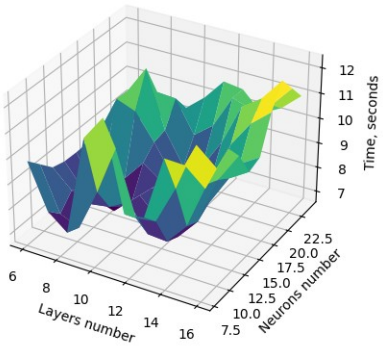
На рисунку 17 можна спостерігати порівняння результатів роботи нейромережі з оптимальними архітектурами з аналітичним розв’язком (3).

а) б)





в) г)



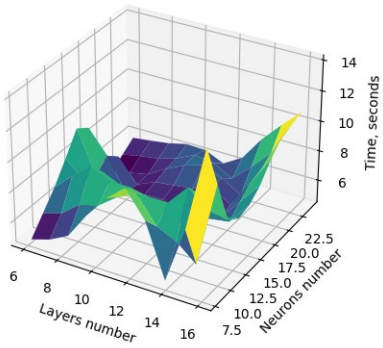
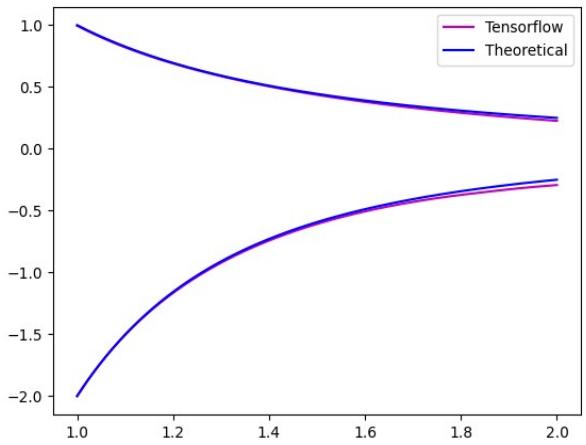
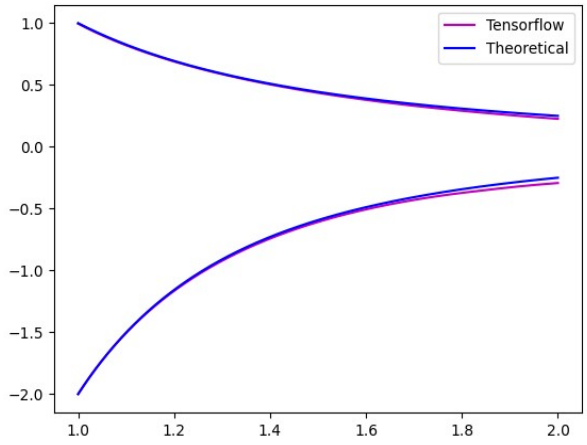


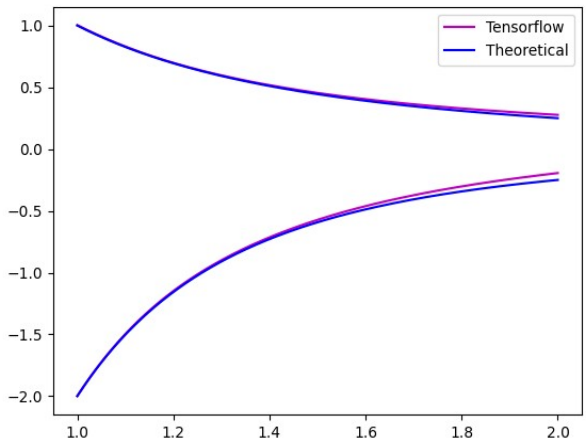
Рисунок 16 — Графік залежності часу навчання від архітектури ГНМ з функцією активації: а) elu; б) selu; в) swish; г) tanh



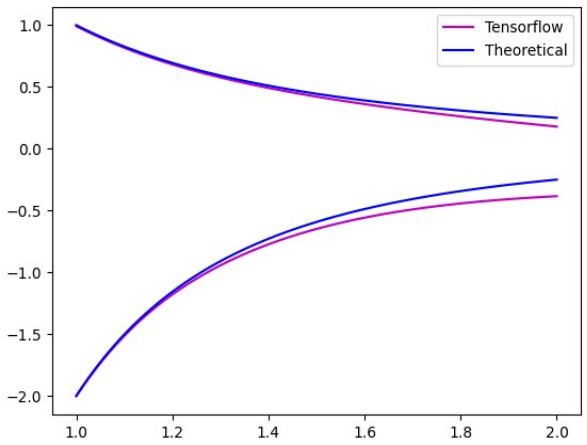
а)



б)



в)



г)

Рисунок 17 — Порівняння аналітичного розв’язку з результатами роботи нейромережі з оптимальною архітектурою при використанні: а) функції активації elu; б) функції активації selu; в) функції активації swish; г) функції активації tanh

ВИСНОВКИ

Протягом виконання даної роботи було проаналізовано та систематизовано існуючий досвід про теоретичну частину використання глибинних нейронних мереж для вирішення практичних задач. Було досліджено та описано алгоритм знаходження наближеного розв’язку диференціальних рівнянь за допомогою алгоритмів глибинного навчання. Спроектовано та розроблено програмний код для вирішення поставлених задач.

Так само було досліджено залежність ефективності вирішення поставлених задач від різних гіперпараметрів нейронної мережі. Було встановлено, що кращими функціями активації, за яких нейронна мережа показує найбільш продуктивну поведінку, є функції selu, elu, tanh і swish. Визначено оптимальну архітектуру нейромережі відповідно для кожної функції активації. Так для функції активації selu оптимальна архітектура — 6 слоїв по 8 нейронів; для elu — 10 слоїв по 16 нейронів; для tanh — 14 слоїв по 8 нейронів; для swish — 12 слоїв по 12 нейронів.

Насамкінець було порівняно алгоритми глибинного навчання з традиційними числовими методами у вирішенні задачі знаходження розв’язку диференціальних рівнянь. Було визначено, що алгоритми глибинного навчання ефективно справляються зі спектром задач, з яким не здатні впоратися традиційні числові методи.

Отримані результати є вагомим вкладом у розвиток області використання нейромереж у теоретичних обчисленнях, а також можуть бути використані у подальших дослідженнях у цій області.

ПЕРЕЛІК ДЖЕРЕЛ

[1] “Who Is the Father of Deep Learning” / Tappert, Charles C. — International Conference on Computational Science and Computational Intelligence (CSCI), 2019.

[2] “Machine Learning” / Mitchell, Tom. — New York: McGraw Hill, 1997

[3] “Automated Design of Both Topology and Sizing of Analog Electrical Circuits Using Generetic Programmin. Artificial Intelligence in Design” / Koza, John R.; Bennet, Forrest H.; Andre, David; Keane, Martin A. — Springer, Dordrecht, 1996f

[4] “Intoduction to Machine Learning” / Alpaydin, Ethem. — London: The MIT Press, 2010

[5] “Improving First and Second Order Methods by Modeling Uncertaintly” / Le Roux, Nicolas; Bengio, Yoshua; Fitzgibbon, Andrew. — Optimization for Maching Learning / MIT Press, 2012

[6] “Deep neural networks for object detection” / Szegedy, Christian; Toshev, Alexander; Erhan, Dumitru. — Advances in Neural Information Processing Systems, 2013

[7] “Learning Deep Architectures for AI” / Bengio, Yoshua. — Foundations and Trends in Machine Learning, 2009

[8] “The Power of Deeper Netdoworks For Expressing Natural Functions” / Rolnick, David; Tegmark, Max. — International Conference on Learning Representations, 2018

[9] “Is Artificial Intelligence Finally Coming into Its Own?” / Hof, Robert D. — MIT Technology Review, 2019

[10] “This table-filling strategy is sometimes called dynamic programming” / Goodfellow, Bengio; Goodfellow, Courville. — 2016

[11] “Deep Learning” [Електронний ресурс] / Ian Goodfellow, Yoshua Bengio, Aaron Courville. —MIT Press, 2016. — Режим доступу: https://www.deeplearningbook.org/

[12] “Introductory Lectures on Convex Optimization: A Basic Course” / Nesterov, Yu. — Springer, 2004

[13] “Adaptive Subgradient Methods for Online Learning and Stochastic Optimization” / John Duchi, Elad Hazan, Yoram Singer. — Journal of Machine Learning Research 12, 2011

[14] “Training of Deep Neural Networks based on Distance Measures using RMSProp” / Thomas Kurbiel, Shahrzad Khaleghian. — eprint arXiv:1708.01911, 2017

[15] “Solving differential equations with neural networks: Applications to the calculation of cosmological phase transitions” [Електронний ресурс] / Maria Laura Piscopo, Michael Spannowsky, Philip Waite. — eprint arXiv:1902.05563, 2019. — Режим доступу:
https://arxiv.org/pdf/1902.05563.pdf

[16] “Why Use Python for AI and Machine Learning” [Електронний ресурс] / Angela Beklemysheva. — Steelkiwi inc., 2020. — Режим доступу:
https://steelkiwi.com/blog/python-for-ai-and-machine-learning/

[17] “Tensorflow: A System for Large-Scale Machine Learning” / Abadi, Martín; Barham, Paul; Chen, Jianmin; Chen, Zhifeng; Davis, Andy; Dean, Jeffrey; Devin, Matthieu; Ghemawat, Sanjay; Irving, Geoffrey; Isard, Michael; Kudlur, Manjunath; Levenberg, Josh; Monga, Rajat; Moore, Sherry; Murray, Derek G.; Steiner, Benoit; Tucker, Paul; Vasudevan, Vijay; Warden, Pete; Wicke, Martin; Yu, Yuan; Zheng, Xiaoqiang. — 2016

[18] “Tensorflow. API Documentation” [Електронний ресурс]. — Режим доступу:
https://www.tensorflow.org/api_docs

[19] “Modeling and Analysis of Modern Fluid Problems” / Liancun Zheng, Xinxin Zhang. — Elsevier Inc., 2017

[20] “ Searching for Activation Functions” / Prajit Ramachandran, Barret Zoph. — 2018. — Режим доступу:
https://paperswithcode.com/paper/searching-for-activation-functions

[21] “Blind variation and selective retention in creative thoughts as in other knowledge processes” / Campbell, Donald T. — Psychological Review, 1960