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

Навчально-науковий інститут прикладного системного аналізу

Кафедра математичних методів системного аналізу

До захисту допущено:

Завідувач кафедри

_____ Оксана ТИМОЩУК

«___»_____20__ р.

Дипломна робота

на здобуття ступеня бакалавра

за освітньо-професійною програмою «Системний аналіз і управління»

спеціальності 124 «Системний аналіз»

на тему: «Дослідження та порівняльний аналіз алгоритмів глибокого навчання Stochastic gradient method, Adam, Adagrad, RMSprop в задачах прогнозування.»

Виконав: студент IV курсу, групи КА-93
Копа Максим Вікторович _____

Керівник: професор, д.т.н.
Зайченко Юрій Петрович _____

Консультант з нормоконтролю: к.ф-м.н.
Статкевич Віталій Михайлович _____

Консультант з економічного розділу: доцент, к.е.н.
Рощина Надія Василівна _____

Рецензент: професор, д.т.н.
Мухін Вадим Євгенович

_____

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.
Студент _____

Київ – 2023

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Навчально-науковий інститут прикладного системного аналізу
Кафедра математичних методів системного аналізу

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 124 «Системний аналіз»

Освітньо-професійна програма «Системний аналіз і управління»

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____Оксана ТИМОЩУК
«___»_____2023 р.

ЗАВДАННЯ
на дипломну роботу студенту
Копі Максиму Вікторовичу

1. Тема роботи «Прогнозування психічного здоров'я на основі даних про музичні вподобання», керівник роботи професор, д.т.н. Зайченко Юрій Петрович, затверджені наказом по університету від «30» травня 2023 р. №2065-с.

2. Строк подання студентом роботи: 15.06.2023 р.

3. Вихідні дані до роботи: набори даних «Boston Housing», «IMDB Movie Reviews», «CIFAR-10», «MNIST», «Reuters».

4. Зміст дипломної роботи: 1. Дослідження предметної області, формування постановки задачі; 2. Математичний опис використаних алгоритмів оптимізації глибокого навчання; 3. Проведення глибокого навчання різних моделей з використанням алгоритмів оптимізації, аналіз та

порівняння результатів; 4. Функціонально-вартісний аналіз програмного продукту.

5. Перелік ілюстративного матеріалу: представлення вихідних даних, аналітичні візуалізації даних в процесі обробки, презентація.

6. Консультанти розділів роботи:

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Економічний	Рощина Надія Василівна, доцент, к.е.н.		

7. Дата видачі завдання: 17.04.2023.

Календарний план

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітка
1.	Підготовка до написання роботи: дослідження літератури за обраною темою, постановка та формалізація задачі	23.04.2023	Виконав
2.	Підготовка першого розділу: огляд доступних методів розв’язку задачі, розгляд вхіднх даних, їх опис	30.04.2023	Виконав
3.	Підготовка другого розділу: математичний опис використаних алгоритмів оптимізації та метрик.	7.05.2023	Виконав
4.	Підготовка третього розділу: підготовка даних, створення моделей, компіляція	14.05.2023	Виконав

	моделей та проведення глибокого навчання, вимірювання метрик та аналіз результатів		
5.	Підготовка четвертого розділу: функціонально-вартісний аналіз	21.05.2023	Виконав
6.	Підготовка пояснювальної записки та презентації	28.05.2023	Виконав

Студент _____ Максим КОПА

Керівник _____ Юрій ЗАЙЧЕНКО

РЕФЕРАТ

Дипломна робота: 113 с., 11 табл., 31 рис., 2 додатки, 22 джерела.

ЗАДАЧА ПРОГНОЗУВАННЯ, ГЛИБОКЕ НАВЧАННЯ, НЕЙРОННІ МЕРЕЖІ, ОПТИМІЗАТОРИ, МЕТОД СТОХАСТИЧНОГО ГРАДІЄНТНОГО СПУСКУ, ОПТИМІЗАТОР ADAM, ADAGRAD, RMSPROP.

Об'єкт дослідження – алгоритми оптимізації глибокого навчання: Метод стохастичного градієнтного спуску (SGD), Adam, Adagrad, RMSprop.

Предмет дослідження – задачі прогнозування в контексті глибокого навчання.

Мета роботи – дослідити та порівняти ефективність чотирьох алгоритмів оптимізації глибокого навчання SGD, Adam, Adagrad та RMSprop в задачах прогнозування.

Результати роботи – в процесі виконання роботи було проведено порівняльний аналіз алгоритмів оптимізації глибокого навчання. Для цього було обрано 5 наборів даних та підготовлено їх до навчання моделей. Було створено 5 моделей з різною архітектурою відповідно до кожного набору даних. Було проведено компіляцію та навчання моделей. На етапі компіляції було визначено функцію втрат для кожної моделі відповідно до задачі а також метрики, за якими буде оцінюватись продуктивність моделі. Після навчання моделі було виміряно відповідні втрати та метрики на тестових даних. Було проведено аналіз всіх результатів та зроблено висновки.

Для виконання цього, було створено програмний продукт на мові програмування Python (Додаток Б).

ABSTRACT

Graduate work: 113 p., 11 tabl., 31 fig., 2 append., 22 sources

FORECASTING TASK, DEEP LEARNING, NEURAL NETWORKS, OPTIMIZERS, STOCHASTIC GRADIENT DESCENT METHOD, ADAM OPTIMIZER, ADAGRAD, RMSPROP.

The object of research is deep learning optimization algorithms: Stochastic Gradient Descent Method (SGD), Adam, Adagrad, RMSprop.

The subject of the research is forecasting tasks in the context of deep learning.

The goal of the work is to investigate and compare the effectiveness of four deep learning optimization algorithms SGD, Adam, Adagrad and RMSprop in forecasting tasks.

The results of the work - in the process of the work, a comparative analysis of deep learning optimization algorithms was carried out. For this, 5 data sets were selected and prepared for model training. Models with different architectures were created according to each data set. Models were compiled and trained. At the compilation stage, the loss function for each model was determined according to the task, as well as the metrics by which the performance of the model will be evaluated. After training the model, the corresponding losses and metrics were measured on the test data. All results were analyzed and conclusions drawn.

To do this, a software product was created in the Python programming language (Appendix Б).

ЗМІСТ

ПЕРЕЛІК ПРИЙНЯТИХ СКОРОЧЕНЬ

Adam – Adaptive moment estimation,

AdaGrad – Adaptive Gradient

RMSprop – Root Mean Squared Propagation

SGD – Stochastic Gradient Method

MSE - Mean Squared Error

MAE – Mean Absolute Error

ReLU – Rectified Linear Unit

ВСТУП

Глибоке навчання є однією з найбільш актуальних та швидкозростаючих галузей машинного навчання, яка використовується для розв'язання різноманітних задач, зокрема прогнозування. Одним з ключових аспектів глибокого навчання є вибір підходящих методів оптимізації, які забезпечують швидку та ефективну збіжність моделі під час тренування.

У даній роботі проводиться дослідження та порівняльний аналіз трьох поширених алгоритмів оптимізації глибокого навчання: Stochastic Gradient Method (SGD), Adam та AdaGrad. Окрім цього, також розглядається алгоритм RMSprop, який є варіацією методу AdaGrad.

Метою роботи є визначення ефективності та порівняння цих алгоритмів в задачах прогнозування. Для досягнення цієї мети, проводяться експерименти на різноманітних задачах, зокрема задачі регресії, класифікації відгуків на основі текстових оглядів, класифікації зображень та розпізнавання рукописних цифр. Кожен алгоритм оптимізації буде використовуватися для навчання відповідних моделей, а результати їхньої продуктивності порівнюватимуться.

Основні параметри порівняння будуть включати точність моделей, втрати (наприклад, середньоквадратичну помилку) та швидкість навчання. Ці параметри дозволять зробити висновки щодо ефективності кожного алгоритму оптимізації в різних задачах.

Використання відповідних алгоритмів оптимізації є важливим аспектом глибокого навчання, оскільки вони можуть впливати на результати моделювання та швидкість навчання. Це дослідження допоможе зрозуміти

переваги та недоліки кожного алгоритму, а також визначити оптимальний вибір для конкретної задачі прогнозування.

Загальна мета цієї роботи полягає в поліпшенні розуміння різних алгоритмів оптимізації глибокого навчання та їх впливу на продуктивність моделей. Це дослідження може бути корисним для науковців та практиків, які працюють в галузі машинного навчання та глибокого навчання і шукають оптимальні методи оптимізації для своїх задач прогнозування.

1 ДОСЛІДЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Актуальність роботи

Глибоке навчання знайшло широке застосування у багатьох сферах сучасного суспільства, включаючи комп'ютерне зорове сприйняття, мовний аналіз, фінансові послуги, медицину тощо.

Воно відіграє важливу роль у сучасному суспільстві, розширюючи можливості комп'ютерної обробки даних та автоматизації процесів, дозволяючи отримувати нові знання з великого обсягу даних, покращувати точність прогнозування та вирішувати складні проблеми, що раніше вважалися недосяжними. Завдяки своїм можливостям глибоке навчання продовжує розвиватися і знаходити нові застосування у різних галузях, сприяючи технологічному прогресу та інноваціям.

В свою чергу, алгоритми оптимізації є важливою складовою глибокого навчання і використовуються для покращення процесу навчання нейронних мереж. У контексті глибокого навчання, оптимізатори відповідають за налаштування параметрів моделі з метою мінімізації функції втрат і досягнення оптимальної точності чи ефективності моделі. Серед популярних у глибокому навчанні можна виділити алгоритм стохастичного градієнтного спуску, Adam, AdaGrad, RMSprop та інші. Кожен з них має свої унікальні особливості та принципи роботи, і вибір оптимізатора залежить від конкретної задачі та характеристик навчальних даних.

Актуальність даної роботи полягає в покращенні результатів глибокого навчання та розвитку галузі в цілому. Порівняльний аналіз алгоритмів глибокого навчання допомагає зрозуміти, як впливають різні методи оптимізації на якість та швидкість навчання моделей. Це дає можливість

вибрати найкращий алгоритм для конкретної задачі, забезпечуючи кращі результати та ефективне використання ресурсів. Дослідження у цій галузі може сприяти покращенню розуміння алгоритмів глибокого навчання та впровадженню передових практик в розробку майбутніх моделей.

1.2 Постановка задачі

Об'єктом дослідження є алгоритми оптимізації глибокого навчання: метод стохастичного градієнтного спуску (SGD), Adam, AdaGrad та RMSprop.

Предмет дослідження є глибоке навчання в задачах прогнозування.

Метою роботи є дослідження, визначення та порівняти ефективності чотирьох алгоритмів оптимізації глибокого навчання SGD, Adam, AdaGrad та RMSprop в задачах прогнозування.

В даній роботі постає задача порівняльного аналізу методів оптимізації. Для цього необхідно дослідити роботу методів оптимізації в умовах різних задач прогнозування та порівняти результати. Порівняльний аналіз методів оптимізації SGD, Adam, AdaGrad, RMSprop можна представити наступними етапами:

1. Вибір задач та наборів даних: Обирається кілька задач прогнозування, які включають різні типи даних, наприклад, структуровані дані, текстові дані або зображення. Для кожної задачі обирається відповідний набір даних.
2. Реалізація моделей: Реалізація моделі глибокого навчання з використанням алгоритмів оптимізації. Важливим є забезпечення того,

щоб моделі були однаковими за своєю архітектурою, за винятком методу оптимізації.

3. Тренування моделей: Тренування моделі з використанням відповідних методів оптимізації. На цьому етапі потрібно зафіксувати важливі параметри, такі як швидкість навчання, кількість епох тренування та розмір пакетів.
4. Вимірювання метрик: Вимірювання метрик ефективності для кожної моделі, такі як точність, втрати або середньоквадратична помилка, на тестовому наборі даних після кожної епохи тренування. Метрики записуються для подальшого порівняння.
5. Аналіз результатів: Порівняння результатів для моделей, що використовують методи SGD, Adam, AdaGrad та RMSprop. Перевірка швидкості збіжності кожного методу, спостереження за зміною метрик протягом тренування та порівняння їх значення на кінцевій точці тренування. Розгляд того, які методи працюють краще для різних типів даних та задач прогнозування.

1.3 Опис даних

В роботі буде використано всього 5 наборів даних, завантажених за допомогою бібліотеки **tenserflow.keras.datasets**:

1. ***Boston Housing***

Набір даних «Boston Housing» — це набір даних, який широко використовується в галузі машинного навчання та статистики. Його вперше представили дослідники з Бюро перепису населення США та Чиказького

університету в 1978 році, і з тих пір він став еталонним набором даних для регресійного аналізу та прогнозного моделювання.

Набір даних містить інформацію про ціни на житло в передмістях Бостона, штат Массачусетс. Він складається з 506 зразків, кожен з яких представляє інше передмістя. Для кожного передмістя є 13 різних особливостей або атрибутів, які описують різні аспекти території. Ці характеристики використовуються для прогнозування середньої вартості будинків, які займають власники, у тисячах доларів (цільова змінна).

13 функцій, включених до набору даних, такі [1]:

1. CRIM: Рівень злочинності на душу населення за містами.
2. ZN: Частка житлової землі, виділеної на ділянки понад 25 000 квадратних футів.
3. INDUS: Частка акрів нероздрібного бізнесу на місто.
4. CHAS: фіктивна змінна річки Чарльз (1, якщо передмістя межує з річкою; інакше 0).
5. NOX: концентрація оксидів азоту (частки на 10 мільйонів).
6. RM: Середня кількість кімнат на житло.
7. AGE: Частка квартир, які займають власники, побудованих до 1940 року.
8. DIS: зважені відстані до п’яти бостонських центрів зайнятості.
9. RAD: Індекс доступності радіальних магістралей.
10. TAX: повна ставка податку на майно за 10 000 доларів США.
11. PTRATIO: Співвідношення учнів/вчителів за містами.
12. B: $1000 \cdot (B_k - 0{,}63)^2$, де B_k — частка темношкірих людей у містах.
13. LSTAT: Відсоток нижчого статусу населення.

Кожна вибірка в наборі даних представляє передмістя, а функції відображають різні характеристики передмістя, які можуть впливати на ціни

на житло. Цільова змінна, середнє значення будинків, які займають власники, надається для кожного передмістя.

«Boston Housing» часто використовується для дослідження методів регресії, оцінки ефективності алгоритмів і вивчення зв’язку між цінами на житло та заданими атрибутами. Це був цінний ресурс для дослідників і практиків машинного навчання в розробці та тестуванні моделей для прогнозування цін на житло та відповідного аналізу.

2. ***IMDb Movie Reviews***

Набір даних «IMDb Movie Reviews» — це популярний набір даних, який часто використовується в обробці природної мови і аналізі настроїв.

Набір даних містить велику кількість оглядів фільмів, зібраних із веб-сайту IMDb, що є популярною онлайновою базою даних фільмів, телешоу тощо (див. рисунок 1.1). Відгуки в цьому наборі даних охоплюють широкий спектр фільмів, жанрів і настроїв, висловлених рецензентами. Кожен відгук у наборі даних позначається одним із двох настроїв: позитивним або негативним, що вказує на те, позитивну чи негативну думку висловив рецензент про фільм. Це налаштування двійкової класифікації робить його придатним для навчання та оцінювання моделей, які мають на меті класифікувати текст за категоріями позитивного чи негативного настрою.

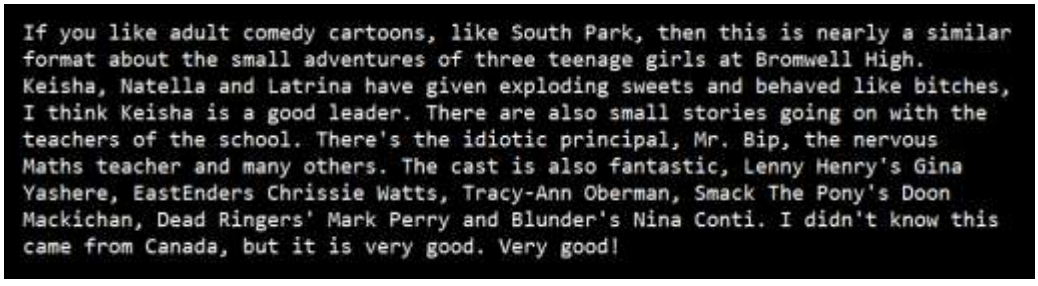


Рисунок 1.1 – Приклад огляду фільму з IMDb Movie Reviews [2]

Важливим є те, що цей датасет насамперед зосереджується на бінарній класифікації настроїв, і рецензії можуть не відображати повного розмаїття

жанрів фільмів, мов або культурного середовища. Тому, працюючи з цим ним, важливо пам’ятати про ці обмеження та розглядати додаткові джерела даних або показники оцінки, щоб отримати повне розуміння ефективності аналізу настроїв.

3. *CIFAR-10*

Набір даних «CIFAR-10» — це широко використовуваний набір даних у сфері комп’ютерного зору та машинного навчання. Він складається з набору позначених зображень, спеціально розроблених для завдань розпізнавання об’єктів. Назва CIFAR-10 розшифровується як «Канадський інститут перспективних досліджень 10», оскільки його створили дослідники з Канадського інституту перспективних досліджень.

Цей набір даних містить загалом 60 000 кольорових зображень, кожне з роздільною здатністю 32x32 пікселя. Ці зображення розділені на десять різних класів, причому кожен клас представляє певну категорію об’єктів (див. рисунок 1.2). Класи в наборі даних CIFAR-10 такі:

1) літак;
2) автомобіль;
3) птах;
4) кішка;
5) олень;
6) собака;
7) жаба;
8) кінь;
9) човен;
10) вантажівка.

Рисунок 1.2 – Зразки зображень набору даних «CIFAR-10» за категоріями [3]

Набір даних розділений на дві підмножини: навчальний набір і тестовий набір. Навчальний набір складається з 50 000 зображень, тоді як тестовий набір містить 10 000 зображень. Такий поділ дозволяє дослідникам і практикам навчати свої моделі на великій кількості даних, а потім оцінювати їх ефективність на невідомих прикладах.

CIFAR-10 створює складне завдання для алгоритмів класифікації зображень через відносно низьку роздільну здатність і наявність різних категорій об’єктів. Його широко використовували як еталонний набір даних для оцінки продуктивності різних моделей комп’ютерного зору та алгоритмів, у тому числі згорткових нейронних мереж. Дослідники та практики використовують його для розробки та тестування підходів до розпізнавання об’єктів, виділення ознак і класифікації зображень.

4. ***MNIST***

База даних MNIST (Модифікована база даних Національного інституту стандартів і технологій, Modified NIST) — це велика база даних рукописних цифр, яка зазвичай використовується для навчання різних систем обробки зображень. Вона була створена шляхом «повторного змішування» зразків з оригінальних наборів даних NIST. Розробники вважали, що, оскільки

навчальний набір даних NIST був узятий від співробітників Американського бюро перепису населення, а тестовий набір даних був узятий від американських старшокласників, він не підходить для експериментів з машинним навчанням. Крім того, чорно-білі зображення від NIST були нормалізовані для розміщення в обмежувальній рамці 28x28 пікселів і згладжені, що ввело рівні сірого кольору [4].

Набір даних MNIST складається з набору рукописних цифр, які зазвичай використовуються для навчання та оцінювання алгоритмів класифікації зображень (див. рисунок 1.3). Він був створений, щоб забезпечити стандартизований еталон для порівняння різних моделей машинного навчання для завдання розпізнавання рукописних цифр.

Рисунок 1.3 – Зразки зображень з набору даних «MNIST» [5]

MNIST містить 60000 тренувальних зображень і 10000 тестових зображень. Половина навчального набору та половина тестового набору були взяті з навчального набору даних NIST, тоді як інші половини були взяті з тестового набору. Такий поділ дозволяє дослідникам і практикам навчати свої моделі на великій кількості даних, а потім оцінювати їх ефективність на невідомих прикладах. Кожне зображення в цьому датасеті представляє одну цифру від 0 до 9. Мета полягає в тому, щоб розробити модель машинного

навчання, яка зможе точно класифікувати ці зображення за відповідними цифрами. Серед даних є мітки для кожного зображення, що вказує справжню цифру, яку воно представляє. Ці мітки використовуються під час навчання, щоб керувати процесом навчання моделі, і під час оцінювання, щоб оцінити її точність.

Набір даних MNIST широко використовувався для розробки та оцінки різних алгоритмів машинного навчання, зокрема в області класифікації зображень і глибокого навчання. Багато популярних моделей і архітектур, таких як згорткові нейронні мережі, були навчені та перевірені на цьому наборі даних. Простота та стандартизованість набору даних MNIST роблять його ідеальною відправною точкою для вивчення та експериментування із завданнями класифікації зображень.

5. ***Reuters***

Набір даних «Reuters» — широко використовуваний набір даних у сфері обробки природної мови і класифікації тексту. Він був спеціально розроблений для завдань категоризації тексту та був створений шляхом збору статей новин від агентства новин Reuters. Його популярність пояснюється його різноманітністю та реальним характером, що дозволяє комплексно оцінювати можливості моделей у обробці складних новинних статей і тем.

Набір даних містить велику кількість статей новин, кожна стаття представляє собою документ. Навчальний набір має 7769 документів, тоді як тестовий – 3019 документів [6]. Ці статті охоплюють різноманітні теми, зокрема бізнес, політику, спорт, технології тощо.

Однією з особливостей набору даних є те, що він підтримує класифікацію за кількома мітками. Це означає, що новинну статтю можна віднести до кількох категорій одночасно, дозволяючи призначати більш

детальні та складні теми. Ця гнучкість відображає реальний сценарій, де статті новин можуть охоплювати кілька тем або мати теми, що збігаються.

1.4 Висновки до першого розділу

Спочатку в даному розділі було розглянуто й обґрунтовано актуальність дослідження та порівняльного аналізу алгоритмів оптимізації глибокого навчання. Далі було сформульовано задачу та сформовано чіткі кроки для її виконання. Також були описані набори даних, з якими буде проводитись робота.

2 ТЕОРЕТИЧНІ ВІДОМОСТІ

2.1 Глибоке навчання та алгоритми оптимізації

Глибоке навчання — це підмножина машинного навчання, яке, по суті, є нейронною мережею з трьома або більше рівнями. Ці нейронні мережі намагаються змоделювати поведінку людського мозку — хоча й далеко не збігаються з його можливостями — дозволяючи йому «навчатися» на великих обсягах даних. Хоча нейронна мережа з одним шаром все ще може робити приблизні прогнози, додаткові приховані шари можуть допомогти оптимізувати та покращити точність.

Глибоке навчання має кілька варіантів використання в автомобільній, аерокосмічній промисловості, виробництві, електроніці, медичних дослідженнях та інших галузях. В якості прикладів глибокого навчання:

- Безпілотні автомобілі використовують моделі глибокого навчання для автоматичного виявлення дорожніх знаків і пішоходів.
- Системи захисту використовують глибоке навчання, щоб автоматично позначати зони інтересу на супутникових зображеннях.
- Аналіз медичного зображення використовує глибоке навчання для автоматичного виявлення ракових клітин для медичної діагностики.
- Заводи використовують програми глибокого навчання, щоб автоматично визначати, коли люди чи предмети знаходяться на небезпечній відстані від машин.

Глибоке навчання керує багатьма додатками та службами штучного інтелекту, які покращують автоматизацію, виконуючи аналітичні та фізичні

завдання без втручання людини. Технологія глибокого навчання лежить в основі повсякденних продуктів і послуг, а також нових технологій.

Алгоритми оптимізатора — це метод оптимізації, який допомагає покращити продуктивність моделі глибокого навчання. Ці алгоритми оптимізації регулюють атрибути нейронної мережі, такі як ваги та швидкість навчання. Таким чином, вони допомагають зменшити загальні втрати та підвищити точність. Проблема вибору правильних вагових коефіцієнтів для моделі є складним завданням, оскільки модель глибокого навчання зазвичай складається з мільйонів параметрів. Це викликає потребу вибрати відповідний алгоритм оптимізації для вашої програми.

Основні оптимізатори у глибокому навчанні:

1. Градієнтний спуск: Базовий оптимізатор, який використовується для оновлення параметрів моделі в напрямку, протилежному до градієнту функції втрат. Його ціль - мінімізувати значення функції втрат шляхом пошуку локального або глобального мінімуму.
2. Стохастичний градієнтний спуск (SGD): Використовує лише один випадковий приклад даних у кожній ітерації для оновлення параметрів моделі. Це зменшує обчислювальні витрати, але може сповільнити швидкість збіжності.
3. Варіації SGD: Це варіації стохастичного градієнтного спуску, які використовуються для поліпшення збіжності та уникнення застрягання в локальних мінімумах. До них належать Momentum, Nesterov Accelerated Gradient, AdaGrad, RMSprop та Adam.
4. Адаптивні методи: Оптимізатори, які адаптивно налаштовують швидкість навчання для кожного параметра на основі історії градієнтів. Вони дозволяють швидше навчання для параметрів, які мають менші градієнти, і повільніше навчання для параметрів з великими

градієнтами. Приклади таких оптимізаторів включають AdaGrad, RMSprop, Adam.

5. Регуляризація: Деякі оптимізатори мають вбудовані методи регуляризації, такі як L1 або L2 регуляризація, для контролю перенавчання моделі. Це допомагає зменшити значення параметрів та поліпшити загальну здатність моделі до узагальнення.

Користування певним оптимізатором залежить від конкретної задачі, архітектури мережі та доступних обчислювальних ресурсів. Вибір правильного оптимізатора може виявитися критичним для успішного навчання моделі та досягнення високої продуктивності.

2.2 Метод оптимізації Stochastic Gradient Descent

Стохастичний градієнтний метод (Stochastic Gradient Descent Method, SGD) є алгоритмом оптимізації, який використовується для навчання моделей глибокого навчання. В основі цього методу лежить градієнтний спуск, який використовується для знаходження локального мінімуму функції втрат.

Нехай маємо набір тренувальних даних з мітками (x, y), де x - вхідні дані, а y - відповідні мітки або цільові значення. Задача полягає у пошуку параметрів моделі, які найкраще апроксимують залежність між x і y. Для цього ми використовуємо функцію втрат L, яка вимірює різницю між прогнозованими значеннями моделі і справжніми значеннями.

Метою SGD є мінімізація функції втрат L шляхом оновлення параметрів моделі на кожному кроці навчання. Оновлення параметрів здійснюється шляхом виконання ітераційного процесу, в якому обчислюється градієнт

функції втрат по відношенню до параметрів моделі і виконується крок градієнтного спуску.

Алгоритм SGD для оптимізації моделей глибокого навчання можна описати наступними кроками [7]:

1. Ініціалізувати параметри моделі θ.
2. Ініціалізувати швидкість навчання α.
3. Повторювати до досягнення критерію зупинки:
   - Вибрати випадковий піднабір тренувальних даних (x, y).
   - Обчислити прогнозоване значення моделі $f(x; \theta)$.
   - Обчислити градієнт функції втрат $\nabla L(\theta; x, y)$ по відношенню до параметрів моделі.
   - Оновити параметри моделі: $\theta \leftarrow \theta - \alpha \cdot \nabla L(\theta; x, y)$.
4. Повернути оптимальні значення параметрів моделі θ.

SGD є простим і швидким алгоритмом оптимізації, але він може мати проблеми зі швидкістю збіжності і може страждати від великого варіативності у випадку, коли дані є шумними або незбалансованими. Тому існують модифікації SGD, такі як Adam, AdaGrad, RMSprop, які спроектовані для покращення його збіжності і стійкості.

2.3 Метод оптимізації AdaGrad

AdaGrad — це сімейство алгоритмів для стохастичної оптимізації. Алгоритми, що належать до цього сімейства, подібні до стохастичного градієнтного спаду другого порядку з наближенням Гессе оптимізованої

функції. Назва AdaGrad походить від Adaptive Gradient. Він інтуїтивно адаптує швидкість навчання для кожної функції залежно від передбачуваної геометрії проблеми; зокрема, він має тенденцію призначати вищі показники навчання нечастим функціям, що гарантує, що оновлення параметрів покладаються менше на частоту, а більше на релевантність. Сам алгоритм можна записати наступним чином [8]:

1. Ініціалізувати швидкість навчання (learning rate) α, зазвичай мале значення (наприклад, 0.01).
2. Ініціалізувати лічильник часу $t = 0$.
3. Ініціалізувати суму квадратів градієнтів G_0 на нулі.
4. На кожній ітерації t до досягнення критерію зупинки:
   - Обчислити градієнт g_t моделі на основі поточної партії даних
   - Збільшити лічильник часу: $t \leftarrow t + 1$.
   - Оновити суму квадратів градієнтів: $G_t \leftarrow G_{t-1} + g_t \odot g_t$.
   - Оновити параметри моделі: $\theta_t \leftarrow \theta_{t-1} - \frac{\alpha}{\sqrt{G_t}+\varepsilon} \odot g_t$, де ε - дуже мала константа для недопущення ділення на нуль.
5. Повернути оптимальні значення параметрів моделі θ.

В основі методу AdaGrad лежить ідея, що швидкість навчання адаптивно регулюється для кожного параметра залежно від історії градієнтів. Чим більше попередніх градієнтів було великими, тим менша буде швидкість навчання для даного параметра, що дозволяє уникнути переповнення та надмірної зміни параметрів. І навпаки, параметри, які мають низькі градієнти, матимуть вищу швидкість навчання, що сприяє більш швидкому навчанню та збіжності для цих параметрів.

2.4 Метод оптимізації RMSprop

RMSprop (Root Mean Square Propagation) є адаптивним алгоритмом оптимізації, який використовується для налаштування параметрів моделей глибокого навчання. Цей метод є модифікацією класичного градієнтного спуску, яка дозволяє ефективніше рухатися в напрямку оптимуму шляхом адаптації швидкості навчання для кожного параметра окремо.

Основна ідея RMSprop полягає в тому, щоб надати більшу вагу останнім градієнтам, знижуючи значення швидкості навчання для параметрів, які мають великі градієнти, і збільшуючи швидкість навчання для параметрів з малими градієнтами. Це дозволяє адаптувати швидкість навчання для кожного параметра залежно від його вкладу у загальну функцію втрат.

Алгоритм методу оптимізації RMSprop має наступний вигляд [9]:

1. Ініціалізувати параметри моделі θ.
2. Ініціалізувати швидкість навчання α.
3. Ініціалізувати параметр затухання β (зазвичай близько до 0.9).
4. Ініціалізувати змінну $r = 0$, яка зберігатиме поточні накопичені квадрати градієнтів.
5. Повторювати до досягнення критерію зупинки:
   - Вибрати випадковий піднабір тренувальних даних (x, y).
   - Обчислити прогнозоване значення моделі $f(x;\ \theta)$.
   - Обчислити градієнт функції втрат $\nabla L(\theta;\ x, y)$ по відношенню до параметрів моделі.

- Оновити накопичені квадрати градієнтів: $r \leftarrow \beta \cdot r + (1 - \beta) \cdot \left(\nabla L(\theta;\ x, y)\right)^2$.
- Оновити параметри моделі: $\theta = \theta - \frac{\alpha}{\sqrt{r+\varepsilon}} \cdot \nabla L(\theta;\ x, y)$, де ε - дуже мале число для стабільності обчислень.

6. Повернути оптимальні значення параметрів моделі θ.

На кроці 5 накопичені квадрати градієнтів використовуються для врахування історії градієнтів. Застосування кореня з квадратного середнього до r дозволяє нормалізувати значення градієнтів за їхніми стандартними відхиленнями. Це допомагає регулювати швидкість навчання для кожного параметра окремо, зменшуючи значення швидкості для параметрів з великими градієнтами та збільшуючи її для параметрів з малими градієнтами.

Таким чином, алгоритм RMSprop дозволяє знизити значення швидкості навчання для параметрів, які мають великі градієнти, тим самим покращуючи швидкість збіжності оптимізації моделі.

2.5 Метод оптимізації Adam

Adam (Adaptive Moment Estimation) є одним з популярних методів оптимізації для навчання нейронних мереж. Він комбінує два підходи: адаптивну швидкість навчання та момент оптимізації.

Автори описують Adam як такий, що поєднує в собі переваги двох інших розширень стохастичного градієнтного спуску. зокрема:

- Адаптивний градієнтний алгоритм (AdaGrad), який підтримує швидкість навчання кожного параметра, що покращує продуктивність у проблемах

із рідкими градієнтами (наприклад, проблеми природної мови та комп'ютерного зору).

- Root Mean Square Propagation (RMSprop), яке також підтримує швидкість вивчення кожного параметра, адаптовану на основі середнього нещодавнього значення градієнтів для ваги (наприклад, як швидко він змінюється). Це означає, що алгоритм добре справляється з онлайн та нестаціонарними проблемами (наприклад, із шумом).

Адам реалізує переваги як AdaGrad, так і RMSProp [10].

У методі обчислюється експоненціально зважений середній квадрат градієнтів (перший момент) та експоненціально зважений середній квадрат попередніх градієнтів (другий момент). Ці середні квадрати використовуються для адаптивного налаштування швидкості навчання для кожного параметра.

Adam також використовує момент оптимізації, який допомагає враховувати інерцію градієнтів при оновленні параметрів моделі. Це дозволяє прискорити оптимізацію та зменшити ймовірність застрягання в локальних мінімумах. Метод має параметри, які варто налаштовувати, такі як швидкість навчання, експоненціальні коефіцієнти згладжування та початкові значення моменту. Налаштування цих параметрів може вплинути на процес оптимізації та швидкість збіжності.

Загалом, метод оптимізації Adam є ефективним алгоритмом для навчання нейронних мереж, оскільки поєднує в собі адаптивну швидкість навчання та момент оптимізації для ефективного пошуку оптимальних значень параметрів моделі.

Алгоритм методу можна описати наступними кроками [11]:

1. Ініціалізувати швидкість навчання α (зазвичай невелике значення, наприклад, 0.001).

2. Ініціалізувати експоненціальні коефіцієнти згладжування β_1 та β_2 в проміжку $[0, 1)$ (зазвичай $\beta_1 = 0.9, \beta_2 = 0.999$).
3. Ініціалізувати лічильник часу $t = 0$.
4. Ініціалізувати перші моменти градієнтів $m_0 = 0$ та другі моменти градієнтів $v_0 = 0$.
5. На кожній ітерації t:
   - Обчислити градієнт g_t моделі на підставі поточної партії даних.
   - Збільшити лічильник часу: $t \leftarrow t + 1$.
   - Оновити перший та другий моменти градієнтів:

$$m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t;$$
$$v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2.$$

   - Виправити зміщення моментів:

$$\hat{m}_t \leftarrow m_t \,/\, (1 - \beta_1^t);$$
$$\hat{v}_t \leftarrow v_t \,/\, (1 - \beta_2^t).$$

   - Оновити параметри моделі:

$$\theta_t \leftarrow \theta_{t+1} - \alpha \cdot \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \varepsilon},$$

де ε - мале число для недопущення ділення на нуль.

2.6 Метрики

Метрики в глибокому навчанні є числовими оцінками, які використовуються для вимірювання якості моделей глибокого навчання. Роль метрик в глибокому навчанні полягає в тому, щоб забезпечити об'єктивну і кількісну оцінку моделі. Вони дозволяють порівнювати різні моделі або різні гіперпараметри моделі і допомагають приймати рішення щодо поліпшення моделі.

Існує багато різних видів метрик у глибокому навчанні, і вибір конкретних метрик залежить від задачі, типу даних і особливостей моделі. Основні види метрик включають:

- Метрики класифікації: Використовуються для оцінки якості моделей у задачах класифікації, де потрібно прогнозувати класи. Прикладами можуть слугувати точність (accuracy), повнота (recall), влучність (precision), F1-міра (F1-score), матриця невідповідності (confusion matrix) тощо.
- Метрики регресії: Використовуються для оцінки моделей у задачах регресії, де потрібно прогнозувати числові значення. Наприклад, середня абсолютна помилка (mean absolute error, MAE), середньоквадратична помилка (mean squared error, MSE), коефіцієнт детермінації (R-squared, або R2 Score) тощо.
- Метрики кластеризації: Використовуються в задачах без навчання, де модель кластеризує подібні об'єкти або приклади у групи або кластери на основі схожості. Приклади метрик кластеризації включають Adjusted Rand Index, Normalized Mutual Information, силуетний коефіцієнт (silhouette coefficient) тощо.

- Метрики генеративних моделей: Використовуються для оцінки моделей генеративного навчання, таких як генеративні згорткові мережі (GANs). Приклади: розмір інформації (inception score), розмір інформації Фреше (Fréchet inception distance), рейтинг користувача (user rating) тощо.

Розглянемо більш детально деякі з них.

Матриця невідповідностей є важливою метрикою для оцінки результатів класифікації моделей у задачах машинного навчання. Вона дозволяє візуалізувати та кількісно оцінити взаємодію між прогнозованими та дійсними класами. Матриця представляє собою двовимірний квадратний масив, де рядки відповідають справжнім класам, а стовпці - прогнозованим класам. Кожен елемент матриці невідповідностей показує кількість прикладів, які були класифіковані вірно або невірно.

Основні елементи матриці невідповідностей:

- Істинно позитивний (True Positive, TP): Кількість прикладів, які коректно класифіковані як позитивні.
- Істинно негативний (True Negative, TN): Кількість прикладів, які коректно класифіковані як негативні.
- Хибно позитивний (False Positive, FP): Кількість прикладів, які неправильно класифіковані як позитивні.
- Хибно негативний (False Negative, FN): Кількість прикладів, які неправильно класифіковані як негативні.

Приклад матриці невідповідностей 2×2 зображено на рисунку 2.1:

		Справжній стан	
	загальна сукупність	позитивний стан	негативний стан
Прогнозований стан	позитивний прогнозований стан	**істинно позитивний**	**хибно позитивний**, помилка I роду
	негативний прогнозований стан	**хибно негативний**, помилка II роду	**істинно негативний**

Рисунок 2.1 – Матриця невідповідностей [12].

Матриця невідповідностей дозволяє виміряти ефективність моделі, розкриваючи кількість правильних та помилкових класифікацій. На основі цієї матриці можна обчислити різні метрики, такі як точність (accuracy), повноту (recall), влучність (precision), F1-міру (F1-score) та інші, що допомагають оцінити якість класифікації моделі.

Точність (accuracy) є однією з найпоширеніших і простих метрик в оцінці класифікаційних моделей в машинному навчанні. Вона вимірює відношення кількості правильно класифікованих прикладів до загальної кількості прикладів у наборі даних.

Точність дозволяє оцінити загальну ефективність класифікаційної моделі, враховуючи як правильно визначені позитивні, так і негативні класи. Ця метрика особливо корисна, коли класи в наборі даних мають приблизно однакову важливість або розподілені рівномірно, але в той же час може бути обманливою, коли має місце незбалансований набір даних, де один клас переважає над іншим.

Точність вираховується за формулою [13]:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN};$$

Повнота (recall), також відоме як істиннопозитивний рівень (true positive rate, TPR), вимірює відсоток правильно визначених позитивних прикладів (істинно позитивних) від загальної кількості дійсних позитивних прикладів у наборі даних.

Повнота обчислюється за формулою [14]:

$$Recall = \frac{TP}{TP + FN}$$

Повнота вимірює, наскільки ефективно модель визначає позитивні класи. Воно вказує на здатність моделі виявити всі дійсні позитивні приклади без помилок. Високе значення відновлення означає, що модель добре визначає позитивні приклади і мінімізує кількість хибно негативних результатів.

Повнота особливо важливе в задачах, де важливо виявити всі позитивні приклади, навіть за ризику отримання деякої кількості хибно позитивних результатів. Наприклад, в медичному діагностуванні, виявлення хвороби є критичним, тому повнота може бути більш значущим, ніж точність.

Влучність (precision) є метрикою в оцінці класифікаційних моделей і вимірює відсоток правильно визначених позитивних прикладів (істинно позитивних) від загальної кількості прикладів, які модель визначила як позитивні.

Влучність також вказує на точність моделі у визначенні позитивних класів. Вона вимірює, наскільки точними є результати моделі, коли вона визначає приклади як позитивні. Високе значення влучності означає, що модель мінімізує кількість хибно позитивних результатів і має високу точність у визначенні позитивних класів. Ця метрика важлива в задачах, де важливо

мінімізувати кількість хибно позитивних результатів, наприклад, у системах виявлення спаму.

Влучність обчислюється за формулою [13, 14]:

$$Precision = \frac{TP}{TP + FP}$$

F1-міра (F1-score) є гармонічним середнім між влучністю (precision) та повнотою (recall). Використовується для оцінки збалансованості між точністю і повнотою моделі.

F1-міра обчислюється за формулою [15]:

$$F1 = \frac{2 \cdot p \cdot r}{p + r},$$

де p – влучність моделі, r – повнота моделі.

F1-міра об'єднує як точність, так і повноту, що дозволяє оцінити баланс між цими двома метриками. Ця метрика особливо корисна в задачах, де важливо досягнути як високої точності, так і високого відновлення, наприклад, в задачах пошуку вбудованих систем або в задачах розпізнавання об'єктів.

Середня абсолютна помилка (Mean Absolute Error, MAE) є однією з ключових метрик в оцінці регресійних моделей. Вона вимірює середнє абсолютне значення різниці між спостережуваними значеннями і прогнозованими значеннями.

MAE обчислюється шляхом взяття абсолютної різниці між прогнозованими значеннями та справжніми значеннями для кожного прикладу і обчислення середнього значення цих різниць [16]:

$$MAE = \frac{1}{n} * \sum_{i=1}^{n} |y_i - \hat{y_i}|,$$

де n – кількість прикладів, y_i - прогнозовані значення, $\hat{y_i}$ – справжні значення.

Середня абсолютна помилка вказує на середню величину помилки між прогнозованими значеннями та справжніми значеннями. Чим менше значення MAE, тим краще прогнозні здібності моделі. Ця метрика є простою і інтерпретованою, і часто використовується в регресійних задачах, коли важливо оцінити точність моделі у відтворенні числових значень.

Однак MAE не враховує розподіл помилок і дає однакову вагу кожній помилці, незалежно від її величини. У деяких випадках можуть бути застосовані інші метрики, такі як середньоквадратична помилка, яка надає вагу більшим помилкам.

Середньоквадратична помилка (Mean Squared Error, MSE) вимірює середнє значення квадратів різниці між спостережуваними значеннями і прогнозованими значеннями.

MSE обчислюється шляхом взяття квадрату різниці між прогнозованими значеннями та справжніми значеннями для кожного прикладу, а потім обчислення середнього значення цих квадратів [16]:

$$MSE = \frac{1}{n} * \sum_{i=1}^{n} (y_i - \hat{y_i})^2,$$

де n – кількість прикладів, y_i - прогнозовані значення, $\hat{y_i}$ – справжні значення.

Середньоквадратична помилка дозволяє враховувати величину помилки між прогнозованими значеннями та справжніми значеннями. Чим менше значення метрики, тим краще прогнозні здібності моделі.

MSE є широко використовуваною метрикою в задачах, де важливо оцінити точність моделі у відтворенні числових значень. Враховуючи квадратичну природу, вона є чутливою до великих помилок, отже великі відхилення між прогнозованими та справжніми значеннями сильніше впливають на загальну помилку. Тобто я метрика є особливо корисною, коли великі помилки мають більший вплив на оцінку ефективності моделі.

Коефіцієнт детермінації, також відомий як R2-score використовується для оцінки якості регресійних моделей. Він вимірює пропорцію дисперсії відповіді, яка пояснюється моделлю, відносно загальної дисперсії відповіді.

R2-score набуває значень від 0 до 1 і може інтерпретуватися як відсоток дисперсії відповіді, яка пояснюється моделлю.

Коефіцієнт вимірює, наскільки добре модель адаптується до даних порівняно з базовою моделлю, яка просто прогнозує середнє значення відповіді для всіх спостережень. Якщо метрика рівна 0, то модель не має переваги перед простою базовою моделлю. Якщо ж рівна 1, то модель прогнозує всю дисперсію відповіді ідеально.

Коефіцієнт детермінації вимірюється за формулою [18]:

$$R2 = 1 - \frac{ESS}{TSS},$$

де ESS - сума квадратів розбіжностей між спостережуваними значеннями і прогнозованими значеннями; TSS - загальна сума квадратів розбіжностей між спостережуваними значеннями і середнім значенням відповіді.

2.7 Висновки до другого розділу

В цьому розділи було розглянуто математичні основи алгоритмів оптимізації глибинного навчання та метрик для вимірювання якості моделей глибокого навчання.

Спочатку було викладено невеликі теоретичні відомості з глибокого навчання та алгоритмів оптимізації. Далі було розглянуто методи (алгоритми) оптимізації Стохастичний градієнтний спуск, Adam, AdaGrad та RMSprop. Для кожного методу було наведено теоретичні відомості та покроково описано сам алгоритм методів оптимізації. Також було описано метрики для дослідження ефективності моделей, а саме метрики класифікації та метрики регресії.

3 ПОРІВНЯЛЬНИЙ АНАЛІЗ

3.1 Вибір задач та наборів даних

На цьому етапі було обрано, звантажено та підготовлено до використання різні набори даних. Набори даних можна класифікувати наступним чином: структуровані дані, текстові дані та зображення.

1. Структуровані дані (набір даних «Boston Housing»):

Задача прогнозування: Прогнозування середньої вартості житла в різних районах на основі певних характеристик (наприклад, кількість кімнат, кримінальна статистика тощо).

Перш за все, набір даних завантажуємо за допомогою бібліотеки **tenserflow.keras.datasets**. Після цього ми нормалізуємо тренувальні дані за допомогою методу **fit_transform()** з бібліотеки **sklearn.preprocessing**. Ця функція обчислює середнє значення і стандартне відхилення кожної ознаки в тренувальному наборі, а потім застосовує стандартизацію до кожного значення ознаки, замінюючи його на відповідне зсунуте та масштабоване значення. Також нормалізуємо тестові дані за допомогою методу **transform()**.

Код цих процесів зображено на рисунку 3.1:

```
# Завантаження даних
(X_train, y_train), (X_test, y_test) = boston_housing.load_data()

# Нормалізація даних
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

Рисунок 3.1 – Завантаження та нормалізація набору даних «Boston Housing»

2. Текстові дані (набір даних «IMDB Movie Reviews»):

Задача прогнозування: Класифікація відгуків на позитивні та негативні на основі текстових оглядів фільмів.

Завантажуємо набір даних за допомогою тієї ж бібліотеки, що й для минулої задачі.

В текстовому наборі даних, кожен відгук представлений у вигляді послідовності слів. Щоб уніфікувати довжини послідовностей, використовуємо **pad_sequences()** з бібліотеки **tensorflow.keras.preprocessing** (див. рисунок 3.2).

```
# Завантаження даних
(X_train, y_train), (X_test, y_test) = imdb.load_data(num_words=10000)

# Передобробка даних
max_sequence_length = 80  # Максимальна довжина послідовності
X_train = pad_sequences(X_train, maxlen=max_sequence_length)
X_test = pad_sequences(X_test, maxlen=max_sequence_length)
```

Рисунок 3.2 – Завантаження та передобробка набору даних «IMDB Movie Reviews»

3. Зображення (набір даних «CIFAR-10»):

Задача прогнозування: Класифікація зображень на 10 категорій об'єктів, таких як автомобілі, літаки, собаки тощо.

Зображення у наборі даних CIFAR10 представлені у вигляді матриць пікселів, де значення пікселів знаходяться в діапазоні від 0 до 255. Під час передобробки, піксельні значення нормалізуються до діапазону від 0 до 1, ділячи всі значення на 255.0.

Мітки класів у цьому наборі даних представлені як цілі числа, які відповідають класам об'єктів. Для побудови моделі класифікації зазвичай використовуються one-hot вектори, де кожен клас має вектор, який

складається з нулів, крім одного значення, яке відповідає класу. Для цього використовується функція **to_categorical()** для конвертації міток у формат one-hot векторів (див. рисунок 3.3.).

```
# Завантаження даних
(X_train, y_train), (X_test, y_test) = cifar10.load_data()

# Передобробка даних
X_train = X_train / 255.0
X_test = X_test / 255.0
y_train = to_categorical(y_train, num_classes=10)
y_test = to_categorical(y_test, num_classes=10)
```

Рисунок 3.3 – Завантаження та передобробка набору даних «CIFAR-10»

4. Зображення (набір даних «MNIST»):

Задача прогнозування: Розпізнавання рукописних цифр на зображеннях набору даних «MNIST». Модель машинного навчання повинна класифікувати зображення цифр на числові категорії від 0 до 9.

Як і у попередньому випадку, нормалізуємо тренувальні та тестові дані, переводячи значення пікселів у значення в інтервалі від 0 до 1. Після цього конвертуємо мітки в one-hot вектори за допомогою **to_categorical()**. (див. рисунок 3.4)

```
# Завантаження набору даних MNIST
(X_train, y_train), (X_test, y_test) = mnist.load_data()

# Переведення значень пікселів у інтервал від 0 до 1
X_train = X_train / 255.0
X_test = X_test / 255.0

# Конвертація міток в one-hot вектори
y_train = to_categorical(y_train, num_classes=10)
y_test = to_categorical(y_test, num_classes=10)
```

Рисунок 3.4 – Завантаження та передобробка набору даних «MNIST»

5. Текстові дані (набір даних «Reuters»):

Задача прогнозування: Класифікація новини на основі їх текстового змісту і передбачати їх категорію або тему.

Перш за все завантажуємо набір даних. При цьому обмежуємо кількість слів, які будуть використовуватись у наборі, 1000 найчастіших.

Після цього, для побудови моделі глибокого навчання, перетворюємо текстові дані у числові вектори: використовуючи метод **sequences_to_matrix** з **tensorflow.keras.preprocessing.text** перетворюємо тренувальні та текстові послідовності слів у бінарні матриці.

В кінці, як і в попередніх випадках, конвертуємо сітки у формат one-hot вектори за допомогою **to_categorical()** (див. рисунок 3.5).

```
# Завантаження даних
max_words = 1000
(X_train, y_train), (X_test, y_test) = reuters.load_data(num_words=max_words)

# Передобробка даних
tokenizer = Tokenizer(num_words=max_words)
X_train = tokenizer.sequences_to_matrix(X_train, mode='binary')
X_test = tokenizer.sequences_to_matrix(X_test, mode='binary')

# Визначення кількості класів
num_classes = max(y_train) + 1
y_train = to_categorical(y_train, num_classes)
y_test = to_categorical(y_test, num_classes)
```

Рисунок 3.5 – Завантаження та передобробка набору даних «Reuters»

3.2 Реалізація моделей

На цьому етапі було реалізовано моделі глибокого навчання з використанням методів оптимізації. Для того, щоб порівняння продуктивності

методів було коректним, архітектура моделей є однаковою для кожного методу оптимізації.

Для *першої задачі прогнозування* (набір даних «Boston Housing») модель глибокого навчання представляє собою послідовну нейромережу. Модель складається з трьох шарів, які використовуються для вирішення задачі регресії.

Перший шар у моделі - Dense шар з 64 нейронами, що використовує функцію активації ReLU (Rectified Linear Unit). Цей шар має вхідну форму, яка визначає розмір вхідних даних моделі. Розмір вхідних даних відповідає розміру X_train, який використовується для навчання моделі.

Другий шар також є Dense з 64 нейронами і функцією активації ReLU. Цей шар приймає вхідні дані від попереднього шару і генерує більш складені абстракції та ознаки з цих даних.

Останній шар - Dense з одним нейроном. Цей шар використовується для прогнозування вихідного значення, оскільки в даному випадку ми маємо задачу регресії. Таким чином, модель повертає одне числове значення.

У цій моделі немає використання функції активації на останньому шарі, оскільки для задачі регресії ми вимагаємо прогнозування без обмежень. Якщо задача була класифікацією, то на останньому шарі можна було б використовувати функцію активації, наприклад, sigmoid або softmax.

Створення моделей для кожного методу оптимізації можна побачити на рисунку 3.6:

```
# Створення моделей
model_sgd = tf.keras.Sequential([
    tf.keras.layers.Dense(64, activation='relu', input_shape=(X_train.shape[1],)),
    tf.keras.layers.Dense(64, activation='relu'),
    tf.keras.layers.Dense(1)
])

model_adam = tf.keras.Sequential([
    tf.keras.layers.Dense(64, activation='relu', input_shape=(X_train.shape[1],)),
    tf.keras.layers.Dense(64, activation='relu'),
    tf.keras.layers.Dense(1)
])

model_adagrad = tf.keras.Sequential([
    tf.keras.layers.Dense(64, activation='relu', input_shape=(X_train.shape[1],)),
    tf.keras.layers.Dense(64, activation='relu'),
    tf.keras.layers.Dense(1)
])

model_rmsprop = tf.keras.Sequential([
    tf.keras.layers.Dense(64, activation='relu', input_shape=(X_train.shape[1],)),
    tf.keras.layers.Dense(64, activation='relu'),
    tf.keras.layers.Dense(1)
])
```

Рисунок 3.6 – Створення моделей глибокого навчання для вирішення задачі регресії

Наступна модель також є послідовною нейромережею. Вона призначена для вирішення задачі класифікації тексту – що відповідає задачі з набором даних «IMDB Movie Reviews».

Перший шар у моделі - Embedding шар. Він використовується для перетворення послідовностей слів або інших токенів у вектори фіксованої довжини, що відображають семантичну інформацію. У даному випадку, він має розмір словника 10 000 токенів і генерує вектори довжиною 128 для кожного токену. Вхідна довжина (input_length) визначає максимальну довжину послідовності, на якій модель буде навчатися.

Другий шар - LSTM (Long Short-Term Memory) шар. LSTM є типом рекурентної нейронної мережі, яка здатна враховувати довгострокові залежності в послідовностях. В даному випадку, LSTM шар має 128 нейронів, що використовуються для розуміння та узагальнення текстових послідовностей.

Останній шар - Dense шар з одним нейроном і функцією активації sigmoid. Цей шар використовується для класифікації тексту на два класи.

Функція активації sigmoid генерує вихідні значення у діапазоні [0, 1], що можна сприймати як ймовірності належності до певного класу.

Загальна архітектура моделі передбачає, що вхідні дані будуть послідовностями токенів (слів). Embedding шар перетворює ці токени в вектори фіксованої довжини, які потім передаються у LSTM шар для узагальнення текстової інформації. Нарешті, Dense шар з функцією активації sigmoid виконує класифікацію тексту на два класи.

Приклад створення такої моделі видно на рисунку 3.7:

```
# Створення моделей
model_sgd = tf.keras.Sequential([
    tf.keras.layers.Embedding(10000, 128, input_length=max_sequence_length),
    tf.keras.layers.LSTM(128),
    tf.keras.layers.Dense(1, activation='sigmoid')
])
```

Рисунок 3.7 – Створення моделі глибокого навчання для вирішення задачі класифікації; моделі для всіх алгоритмів ідентичні

Третя модель глибокого навчання є послідовною нейромережею, призначеною для розпізнавання об'єктів на зображеннях. Вона використовується для задачі класифікації з використанням зображень (набір даних «CIFAR-10»), що мають форму 32x32 пікселів та 3 канали кольору (RGB).

Перший шар у моделі - Conv2D шар. Conv2D шар використовується для виявлення різних ознак та об'єктів на зображенні. У цій моделі Conv2D шар має 32 фільтри з розміром ядра 3x3, використовує функцію активації ReLU і застосовує збереження границі, що забезпечує збереження розміру зображення.

Після Conv2D шару є MaxPooling2D шар, який зменшує розмірність зображення, використовуючи максимальне значення в кожному пікселі

області зазначеного розміру. У даному випадку, використовується область з розміром 2x2.

Після першого блоку Conv2D і MaxPooling2D шарів, повторюється аналогічний другий блок, який складається з Conv2D шару з 64 фільтрами, MaxPooling2D шару і функції активації ReLU.

Після цього використовується Flatten шар, який перетворює вихідні дані з двовимірного представлення в одновимірний вектор.

Після Flatten шару є Dense шар з 64 нейронами і функцією активації ReLU. Цей шар використовується для отримання більш високорівневих ознак з плоского представлення вектора.

Останній шар - Dense шар з 10 нейронами і функцією активації softmax. Цей шар використовується для класифікації зображень на 10 різних класів. Функція активації softmax генерує вихідні значення у вигляді вектора ймовірностей для кожного класу.

Загальна архітектура моделі полягає в послідовному з'єднанні шарів, де вихід попереднього шару є вхідними даними наступного шару. Conv2D і MaxPooling2D шари використовуються для виявлення локальних ознак на зображеннях, після чого шари Flatten і Dense використовуються для класифікації цих ознак на різні класи.

Приклад створення даної моделі (див. рисунок 3.8):

```
# Створення моделей
model_sgd = tf.keras.Sequential([
    tf.keras.layers.Conv2D(32, (3, 3), activation='relu', padding='same', input_shape=(32, 32, 3)),
    tf.keras.layers.MaxPooling2D((2, 2)),
    tf.keras.layers.Conv2D(64, (3, 3), activation='relu', padding='same'),
    tf.keras.layers.MaxPooling2D((2, 2)),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(64, activation='relu'),
    tf.keras.layers.Dense(10, activation='softmax')
])
```

Рисунок 3.8 – Створення моделі глибокого навчання для вирішення задачі класифікації з використанням зображень

Четверта модель глибокого навчання є нейромережею, призначеною для класифікації зображень рукописних цифр з набору даних «MNIST».

Перший шар у моделі - Flatten. Він використовується для перетворення вхідних зображень з форми (28, 28) в одновимірний вектор. Це робить зображення плоским і розгортає його у вектор фіксованої довжини.

Після Flatten шару є Dense шар з 128 нейронами і функцією активації ReLU. Цей шар використовується для отримання більш високорівневих ознак з плоского представлення вектора.

Останній шар - Dense шар з 10 нейронами і функцією активації softmax. Цей шар використовується для класифікації зображень на 10 різних класів (цифри від 0 до 9). Функція активації softmax генерує вихідні значення у вигляді вектора ймовірностей для кожного класу.

Flatten шар розгортає зображення у вектор, після чого Dense шари виконують операції лінійного перетворення та нелінійної активації для класифікації зображень на різні класи.

Приклад створення даної моделі видно на рисунку 3.9.:

```
# Побудова моделей
model_sgd = tf.keras.Sequential([
    tf.keras.layers.Flatten(input_shape=(28, 28)),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dense(10, activation='softmax')
])
```

Рисунок 3.9 – Створення моделей глибокого навчання для вирішення задачі класифікації зображень; моделі для всіх алгоритмів ідентичні

Остання модель глибокого навчання також є послідовною нейромережею, яка призначена для класифікації тексту представленого векторами фіксованої довжини.

Перший шар у моделі - Dense шар з 512 нейронами і функцією активації ReLU. Цей шар використовується для обчислення високорівневих ознак на основі вхідного вектору фіксованої довжини, який представляє текст у форматі мішка слів. Вхідна форма (input_shape) має розмірність (max_words,), де max_words - це максимальна кількість слів у тексті.

Після Dense є Dropout шар з розміром 0.5. Dropout використовується для випадкового вимкнення (з ймовірністю 0.5) випадкових нейронів у попередньому шарі під час тренування. Це допомагає уникнути перенавчання і зробити модель більш стійкою.

Наступний шар - Dense з 256 нейронами і функцією активації ReLU. Цей шар використовується для подальшого обчислення високорівневих ознак на основі вхідних даних після шару Dropout.

Після цього знову використовується Dropout з розміром 0.5 для регуляризації моделі.

Останній шар - Dense шар з num_classes (кількість класів) нейронами і функцією активації softmax. Цей шар використовується для класифікації тексту на num_classes різних класів. Функція активації softmax генерує вектор ймовірностей для кожного класу, де сума всіх ймовірностей дорівнює 1.

Dense шари використовуються для обчислення високорівневих ознак на основі вхідних даних, Dropout шари застосовуються для регуляризації моделі шляхом випадкового вимкнення нейронів, а softmax шар використовується для отримання вектора ймовірностей класів.

Приклад створення даної моделі (див. рисунок 3.10):

```
# Створення моделей
model_sgd = tf.keras.Sequential([
    tf.keras.layers.Dense(512, input_shape=(max_words,), activation='relu'),
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Dense(256, activation='relu'),
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Dense(num_classes, activation='softmax')
])
```

Рисунок 3.10 – Створення моделей глибокого навчання для вирішення задачі класифікації зображень; моделі для всіх алгоритмів ідентичні

3.3 Тренування моделей

На даному етапі були проводиться компіляція та тренування (навчання) всіх моделей.

Для компіляції моделей було використано функцію **model.compile()** з бібліотеки **tensorflow.keras**. Ця функція встановлює параметри, які визначають як модель буде навчатись та оцінюватись. Основними аргументами функції є [19]:

- **optimizer**: Визначає алгоритм оптимізації, який буде використовуватись під час тренування моделі;
- **loss**: Визначає функцію втрат, яку модель буде оптимізувати під час тренування. Вона представляє собою оцінку того, наскільки добре модель працює на тренувальних даних;
- **metrics**: Визначає список метрик, які будуть вимірюватись під час тренування та оцінювання моделі.

Приклад використання даної функції можна побачити на рисунку 3.11, або в Додатку Б.

Після компіляції, модель готова до навчання. Навчання проводилось за допомогою функції model.fit() з тієї ж бібліотеки. Основні аргументи функції такі [19]:

- **x**: Передає навчальні дані моделі;
- **y**: Передає цільові дані (target data) моделі, пов'язані з навчальними даними;
- **batch_size**: Визначає розмір пакета даних, що використовується під час тренування. Модель оновлює ваги після кожного пакета;
- **epochs**: Визначає кількість повних проходів через навчальні дані, що будуть виконані під час тренування;
- **validation_data**: Передає валідаційні дані, на яких будуть обчислені втрати та метрики під час тренування. **validation_split** випадково вибирає частину тренувальних даних і використовує їх в якості валідаційних.

Використання даної функції можна побачити на рисунку 3.12, або в Додатку Б.

Рисунок 3.11 – Компіляція моделей для задачі з набором даних «MNIST»

```
# Навчання моделей
epochs = 30
start_time_sgd = time.time()
history_sgd = model_sgd.fit(X_train, y_train, epochs=epochs, validation_data=(X_test, y_test), verbose=1)
end_time_sgd = time.time()
execution_time_sgd = end_time_sgd - start_time_sgd
```

Рисунок 3.12 – Навчання моделі з використанням алгоритму оптимізації SGD для задачі з набором даних «MNIST» з замірюванням часу виконання

3.4 Вимірювання метрик

На даному етапі ми збираємо розраховані метрики в результаті виконання навчання моделей. Використані метрики залежать від задачі прогнозування для якої ми проводили глибоке навчання. В задачі регресії були використані метрики MAE, MSE, R2 score та Explained Variance Score. Для задач класифікації, в свою чергу, замірювались метрики Accuracy, Precision, Recall та F1-score.

Для ефективного порівняння моделі з різними оптимізаторами, на етапі навчання моделі були зафіксовані значення втрат та метрик, обчислені на тренувальних та валідаційних даних. Валідаційні дані використовуються для оцінки продуктивності моделі під час навчання і дозволяють порівнювати різні методи оптимізації в контексті того, як вони впливають на втрати та метрики моделі.

З іншого боку, після навчання моделі найкраще проводити порівняння на незалежних тестових даних, які не брали участь під час навчання та валідаційні моделі. Використання таких даних дозволяє оцінити реальну продуктивність моделі на нових прикладах, які раніше не були побачені моделлю, допомагаючи уникненню перенавчання і визначенню того, як добре модель генералізує на нові дані.

Всі вимірювання були занесенні в таблиці (див. таблиці 3.1-3.5) для кожної задачі а також було створено графіки значень втрат та метрик, виміряних для тренувальних та валідаційних даних, відносно епох (див. рисунки 3.13-3.22).

1. Для задачі регресії:

Розмір тренувальних даних: (404, 13)

Розмір тестових даних: (102, 13)

Для навчання моделі тренувальні дані були розділені наступним чином: 85% тренувальні дані, 15% валідаційні дані.

Таблиця 3.1 – Вимірювання метрик для задачі регресії з набором даних «Boston Housing»

Алгоритм оптимізації	Втрати (loss)	MAE	MSE	R2 score	Explained Variance	Час виконання	Кількість епох
SGD	10.8916	2.3318	10.8916	0.7969	0.7969	12.86 sec	150
Adam	18.5565	2.6977	18.5565	0.5178	0.5178	8.88 sec	
AdaGrad	69.1414	6.8074	69.1414	0.0230	0.0230	9.53 sec	
RMSprop	15.3131	2.5649м	15.3131	0.6317	0.6317	9.15 sec	

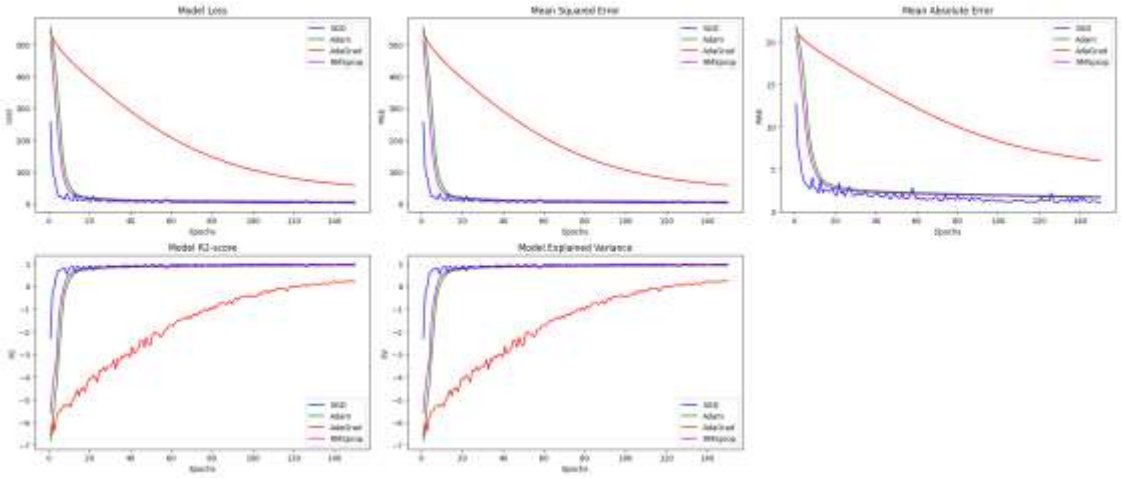


Рисунок 3.13 – Графіки залежності втрат та метрик тренувальних даних в задачі регресії від епохи для кожного методу оптимізації

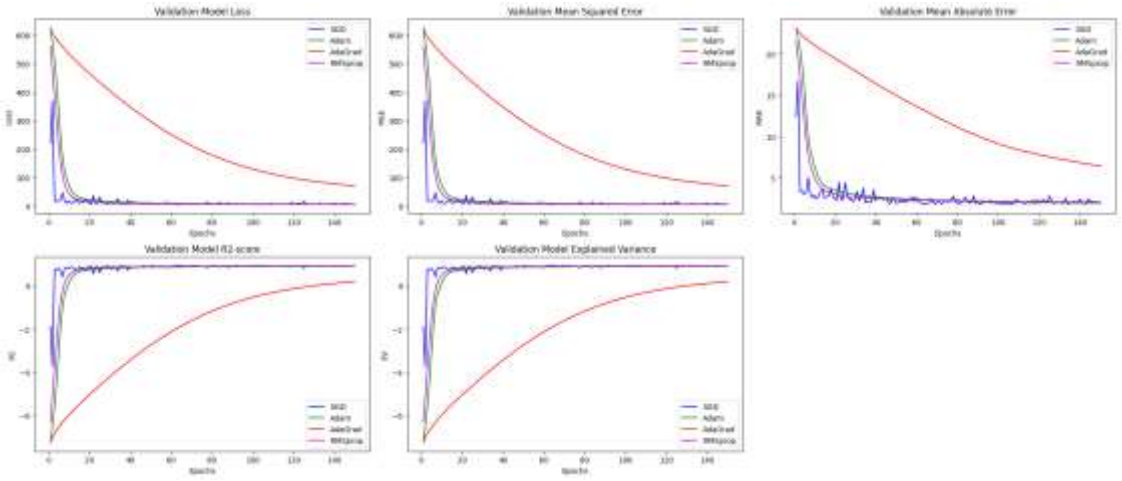


Рисунок 3.14 – Графіки залежності втрат та метрик валідаційних даних в задачі регресії від епохи для кожного методу оптимізації

2. Для задачі класифікації з набором даних «IMDB Movie Reviews»:

Розмір тренувальних даних: (25000, 60)

Розмір тестових даних: (25000, 60)

Для навчання моделі тренувальні дані були розділені наступним чином: 80% тренувальні дані, 20% валідаційні дані.

Таблиця 3.2 – Вимірювання метрик для задачі класифікації з набором даних «IMDB Movie Reviews»

Алгоритм оптимізації	Втрати (loss)	Accuracy	Precision	Recall	F1 score	Час виконання	Кількість епох
SGD	0.6908	0.5650	0.6029	0.3809	0.4573	401.59 sec	10
Adam	1.0796	0.7867	0.8006	0.7637	0.7764	530.05 sec	
AdaGrad	0.6924	0.5419	0.5303	0.7339	0.6091	354.91 sec	
RMSprop	0.6782	0.7949	0.8506	0.7154	0.7714	424.68 sec	

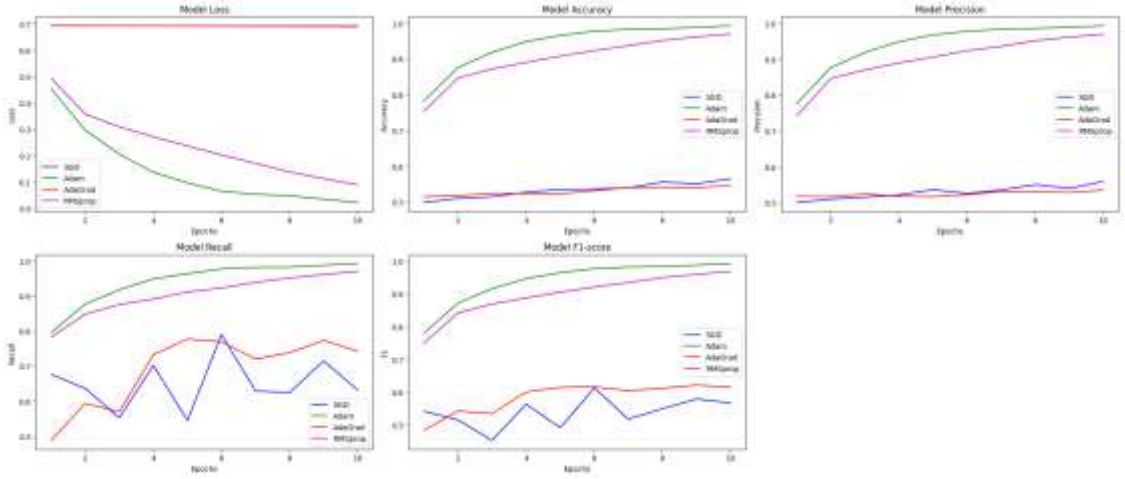


Рисунок 3.15 – Графіки залежності втрат та метрик тренувальних даних від епохи в задачі класифікації з набором даних «IMDB Movie Reviews»

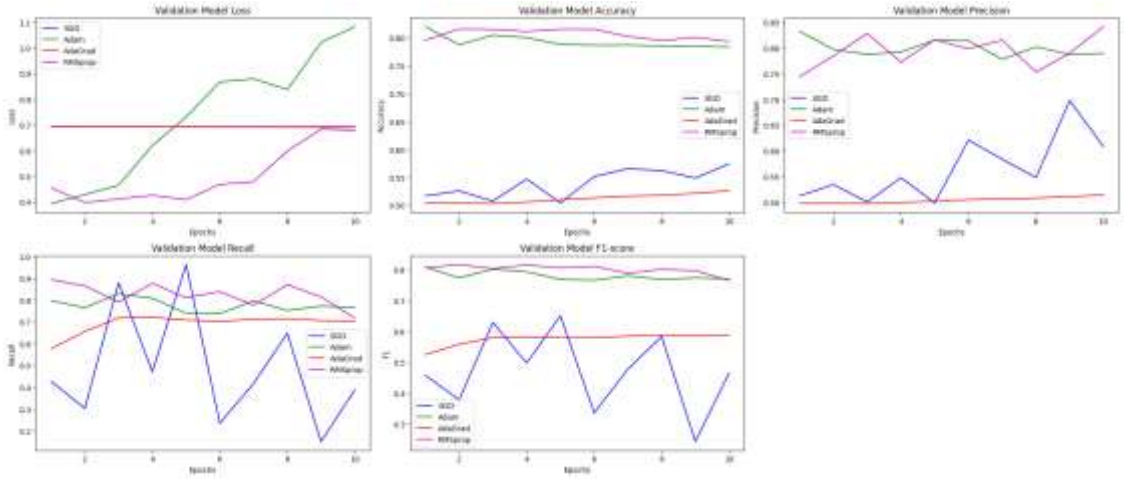


Рисунок 3.16 – Графіки залежності втрат та метрик валідаційних даних від епохи в задачі класифікації з набором даних «IMDB Movie Reviews»

3. Для задачі класифікації зображень з набором даних «CIFAR-10»:

Розмір тренувальних даних: (50000, 32, 32, 3)

Розмір тестових даних: (10000, 32, 32, 3)

Для навчання моделі тренувальні дані були розділені наступним чином: 80% тренувальні дані, 20% валідаційні дані.

Таблиця 3.3 – Вимірювання метрик для задачі класифікації з набором даних «CIFAR-10»

Алгоритм оптимізації	Втрати (loss)	Accuracy	Precision	Recall	F1 score	Час виконання	Кількість епох
SGD	1.0233	0.6405	0.7734	0.5069	0.6094	398.60 sec	10
Adam	1.0614	0.6622	0.7203	0.6089	0.6588	461.71 sec	
AdaGrad	1.6220	0.4353	0.7077	0.1213	0.2034	465.36 sec	
RMSprop	1.0189	0.6962	0.7426	0.6634	0.6999	483.40 sec	

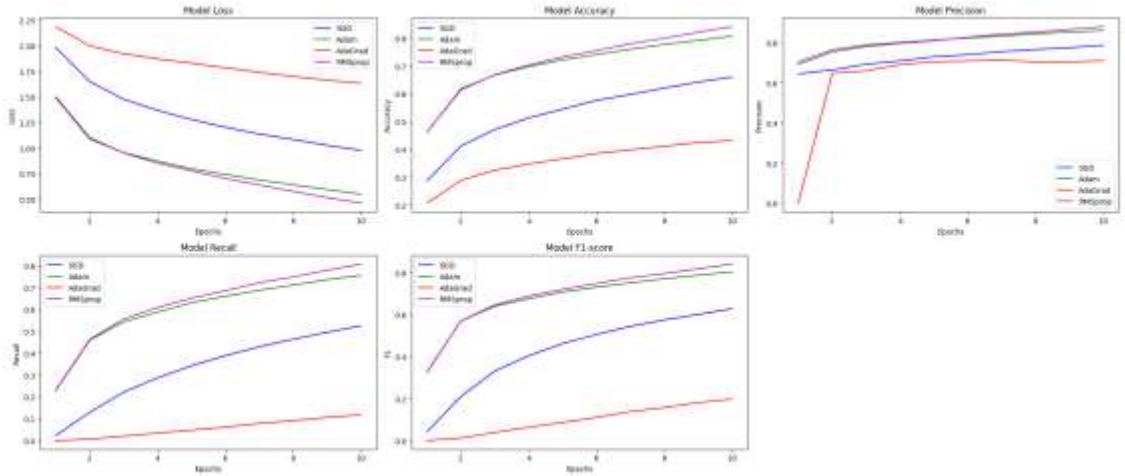


Рисунок 3.17 – Графіки залежності втрат та метрик тренувальних даних від епохи в задачі класифікації з набором даних «CIFAR-10»

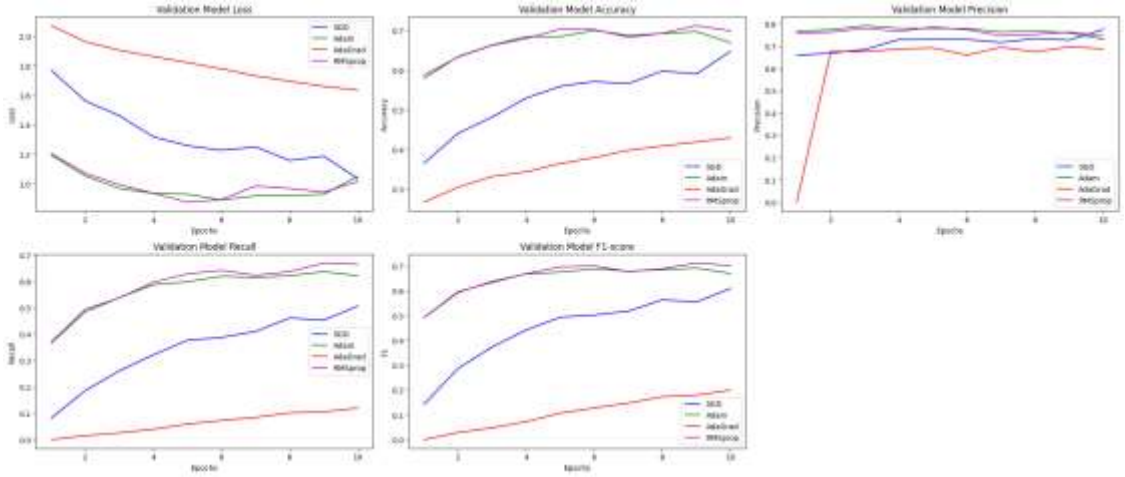


Рисунок 3.18 – Графіки залежності втрат та метрик валідаційних даних від епохи в задачі класифікації з набором даних «CIFAR-10»

4. Для задачі класифікації зображень з набором даних «MNIST»:

Розмір тренувальних даних: (60000, 28, 28)

Розмір тестових даних: (10000, 28, 28)

Для навчання моделі тренувальні дані були розділені наступним чином: 80% тренувальні дані, 20% валідаційні дані.

Таблиця 3.4 – Вимірювання метрик для задачі класифікації з набором даних «MNIST»

Алгоритм оптимізації	Втрати (loss)	Accuracy	Precision	Recall	F1 score	Час виконання	Кількість епох
SGD	0.1201	0.9642	0.9717	0.9582	0.9648	111.67 sec	25
Adam	0.1159	0.9777	0.9783	0.9777	0.9780	135.93 sec	
AdaGrad	0.2760	0.9255	0.9501	0.9000	0.9236	126.16 sec	
RMSprop	0.1122	0.9795	0.9800	0.9793	0.9797	123.90 sec	

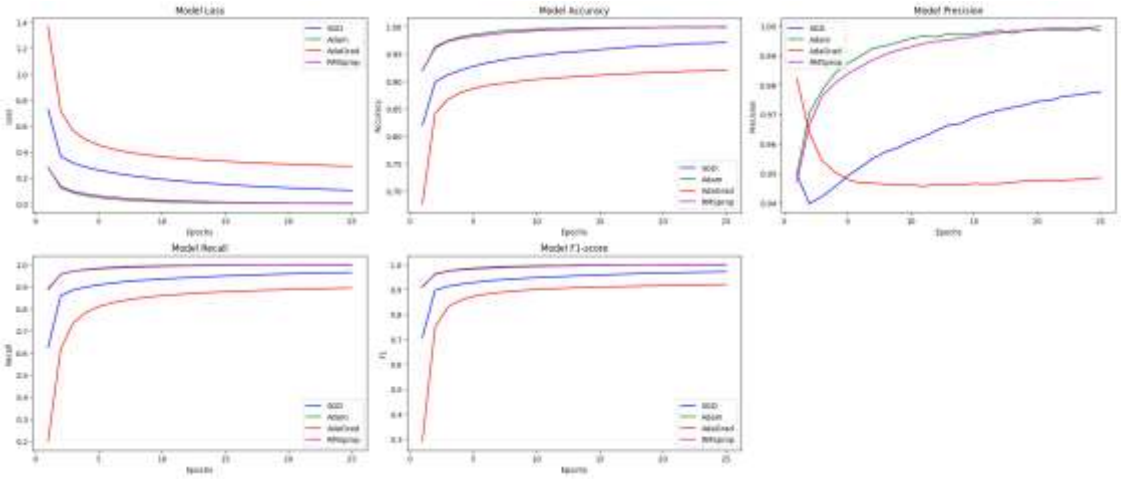


Рисунок 3.19 – Графіки залежності втрат та метрик тренувальних даних від епохи в задачі класифікації з набором даних «MNIST»

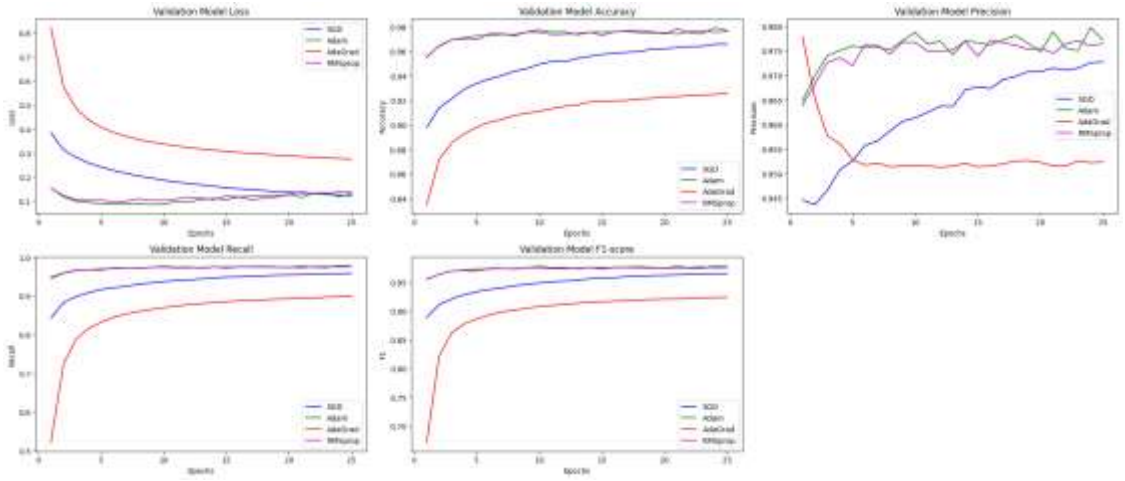


Рисунок 3.20 – Графіки залежності втрат та метрик валідаційних даних від епохи в задачі класифікації з набором даних «MNIST»

5. Для задачі класифікації з набором даних «Reuters»:

Розмір тренувальних даних: (8982, 1000)

Розмір тестових даних: (2246, 1000)

Для навчання моделі тренувальні дані були розділені наступним чином: 80% тренувальні дані, 20% валідаційні дані.

Таблиця 3.5 – Вимірювання метрик для задачі класифікації з набором даних «Reuters»

Алгоритм оптимізації	Втрати (loss)	Accuracy	Precision	Recall	F1 score	Час виконання	Кількість епох
SGD	1.1666	0.7324	0.8903	0.6362	0.7353	43.70 sec	20
Adam	1.3309	0.7858	0.8161	0.7685	0.7899	72.99 sec	
AdaGrad	1.5598	0.6456	0.9058	0.4969	0.6348	53.41 sec	
RMSprop	2.2609	0.7778	0.8163	0.7480	0.7787	83.41 sec	

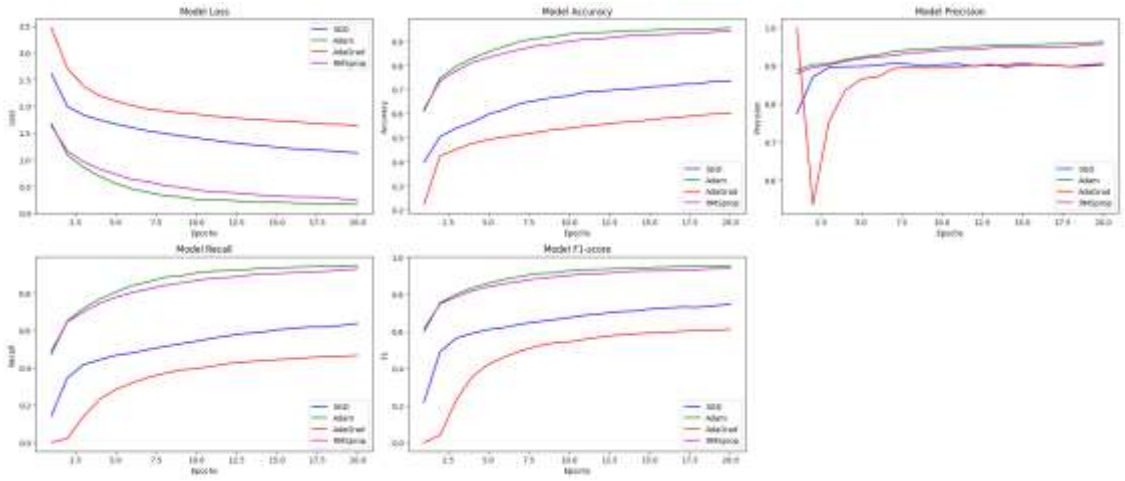


Рисунок 3.21 – Графіки залежності втрат та метрик тренувальних даних від епохи в задачі класифікації з набором даних «Reuters»

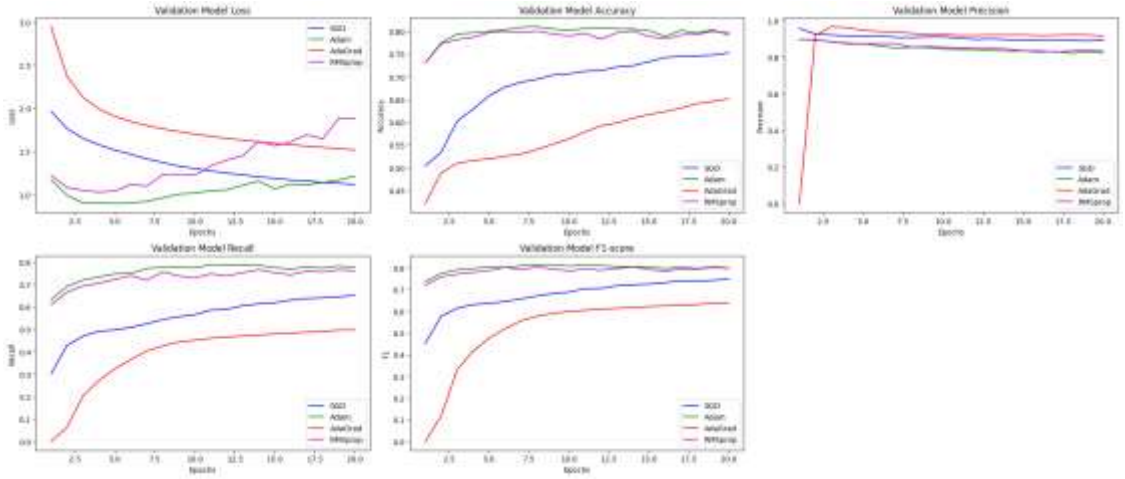


Рисунок 3.22 – Графіки залежності втрат та метрик валідаційних даних від епохи в задачі класифікації з набором даних «Reuters»

3.5 Аналіз результатів

Аналізуючи результати вирішення п’яти задач прогнозування за допомогою глибокого навчання з використанням алгоритмів оптимізації, можна зробити наступні спостереження:

1. Задача регресії:
    - Найкращі результати показав алгоритм SGD з найменшою значенням втрат і найкращими значенням всіх метрик, але час навчання виявився найбільшим.
    - Adam і RMSprop також показали гарні результати, але з вищими втратами та гіршими R2 Score та Explained Variance порівняно з SGD.
    - AdaGrad найгірші результати з вищою втратою та низькими значеннями R2 Score та Explained Variance.
2. Задача класифікації з набором даних "IMDb Movie Reviews":
    - Adam та RMSprop показали найкращі результати. RMSprop має найкращі значення втрат, точності та влучності, а Adam – повноти і F1-міри. Втім, Adam має найгірші втрати та час виконання.
    - AdaGrad показав прийнятні, але все ж нижчі результати. Також, він виявився найшвидшим.
    - SGD показав найгірші повноту та F1-міру, але трохи кращі за AdaGrad точність та влучність.
3. Задача класифікації з набором даних "CIFAR10":
    - Знову найкращі результати показали Adam та RMSprop, лиш трохи відрізняючись один від одного.
    - SGD показав гірші результати порівняно з 2 іншими, але все ж був достатньо хорошим. Більше того, в даній задачі він виявився найшвидшим.
    - AdaGrad показав найгірші значення з усіх метрик та втрат, окрім часу виконання.
4. Задача класифікації з набором даних "MNIST":

- RMSprop цього разу показав найкращі результати втрат та метрик..
- Adam має гірші результати порівняно з RMSprop, але все ж доволі близький до нього.
- SGD показав гірші результати з вищою втратою та нижчими значеннями точності, влучності, повноти і F1-міри. Також він має найнижчий час виконання. З іншого боку, найдовше працював Adam.
- AdaGrad знову показав найгірші результати з вищою втратою і нижчими значеннями точності, влучності, повноти і F1-міри порівняно з Adam і SGD, але все одно мав прийнятний результат. Також з графіків помітно, що під час навчання моделі влучність поступово падає.

5. Задача класифікації з набором даних "Reuters":
   - Adam показав найкращі результати в точності, повноті та F1-мірі, але виявився найгіршим у влучності (Precision).
   - SGD показав прийнятні результати з порівняно нижчими точністю, повнотою та F1-мірі, але досяг гарного значення влучності, а також є найменші втрати та час виконання.
   - Adagrad показав гірші результати з вищою втратою та нижчими значеннями точності, повноти та F1-міри, але цього разу виявився найкращим у влучності. Також він був досить швидким.
   - RMSprop мав кращу за Adam влучність та майже такі ж точності, повноті та F1-міру, але водночас має найгірші втрати та час виконання.
   - Згідно з графіками, втрати для методів Adam та RMSprop на валідаційному наборі даних поступово зростає, що відрізняється від

відповідних на тренувальному наборі. Це може свідчити про перенавчання даних моделей.

Отже, ми можемо підсумувати результати наступним чином:

- Найефективнішим алгоритмом оптимізації в більшості розглянутих задач виявився RMSprop. Він досягав найвищої точності прогнозування та найкращих значень метрик, таких як Accuracy, Precision, Recall та F1 Score.
- SGD також показав непогані результати у деяких задачах, особливо в задачі регресії, де він мав найнижчу втрату та найвищі R2 Score та Explained Variance.
- Adam також продемонстрував загалом чудові результати і хоча іноді не досягав найвищих значень метрик, все ж був близький до цього.
- AdaGrad виявився менш ефективним у порівнянні з іншими, показуючи менші значення точності прогнозування та втрат. Особливо це було помітно в задачах класифікації, де метод часто показував найгірші значення метрик.
- SGD та AdaGrad зазвичай виконуються швидше, ніж Adam та RMSprop, особливо в великих задачах з великим розміром даних.

3.6 Висновки до третього розділу

В цьому розділі було проведено порівняльний аналіз алгоритмів оптимізації глибокого навчання SGD, Adam, AdaGrad та RMSprop.

Спочатку було обрано задачі прогнозування та відповідні набори даних. Також було проведена передобробка цих даних для подальшого глибокого навчання.

Наступним кроком було створено моделі для кожної задачі. Після цього відповідні моделі були скомпільовані, підготовлюючи їх до навчання, а також проведене саме навчання. На етапі компіляції було визначено втрати та метрики, що будуть оцінюватися.

Після навчання моделі для кожної задачі було обчислено втрати та метрики на незалежних тестових даних, що не використовувались в процесі навчання та валідаційні моделі. Всі результати були занесені в таблиці, а за допомогою раніше записаних даних створені графіки зміни втрат та метрик в залежності від епох під час глибокого навчання. Врешті-решт, були проаналізовані результати та зроблені відповідні висновки.

4 ФУНКЦІОНАЛЬНО-ВАРТІСНИЙ АНАЛІЗ

В заданому розділі буде проведено оцінювання основних характеристик для майбутнього програмного продукту, що спеціалізується на порівнянні ефективності методів оптимізації глибокого навчання SGD, Adam, AdaGrad і RMSprop.

В даному дослідженні показано різні варіанти реалізації для забезпечення найбільш коректної та оптимальної стратегії вибору, що має вплив на економічні фактори та сумісність з майбутнім програмним продуктом. Для цього застосовувався апарат функціонально-вартісного аналізу.

Функціонально-вартісний аналіз (ФВА) передбачає собою технологію, що дозволяє оцінити реальну вартість продукту або послуги незалежно від організаційної структури компанії. ФВА проводиться з метою виявлення резервів зниження витрат за рахунок ефективніших варіантів виробництва, кращого співвідношення між споживчою вартістю виробу та витратами на його виготовлення. Для проведення аналізу використовується економічна, технічна та конструкторська інформація.

Алгоритм функціонально-вартісного аналізу включає в себе визначення послідовності етапів розробки продукту, визначення повних витрат (річних) та кількості робочих часів, визначення джерел витрат та кінцевий розрахунок вартості програмного продукту.

4.1 Постановка задачі проектування

У роботі застосовується метод ФВА для проведення техніко-економічного аналізу розробки системи прогнозу стійкості фінансових показників. Оскільки рішення стосовно проектування та реалізації компонентів, що розробляється, впливають на всю систему, кожна окрема підсистема має її задовольняти. Тому фактичний аналіз представляє собою аналіз функцій програмного продукту, призначеного для збору, обробки та проведення аналізу даних по компанії.

Технічні вимоги до програмного продукту є наступні:

- функціонування на персональних комп'ютерах із стандартним набором компонентів;
- зручність та зрозумілість для користувача;
- швидкість обробки даних та доступ до інформації в реальному часі;
- можливість зручного масштабування та обслуговування;
- мінімальні витрати на впровадження програмного продукту.

4.2 Обґрунтування функцій програмного продукту

Головна функція F_0 – розробка можливого програмного продукту, яка дозволяє аналізувати різні характеристики, що безпосередньо впливають на стійкість підприємства. Беручи за основу цю функцію, можна виділити наступні:

F_1 – вибір мови програмування.

F_2 – вибір фреймворків для глибокого навчання.

F_3 – вибір середовища програмування.

Кожна з цих функцій має декілька варіантів реалізації:

Функція F_1:

а) Python.

б) R.

Функція F_2.

а) Keras.

б) PyTorch.

Функція F_3:

а) Jupyter Notebook/JupyterLab.

б) Google Colab.

Варіанти реалізації основних функцій наведені у морфологічній карті системи (рис. 4.1).

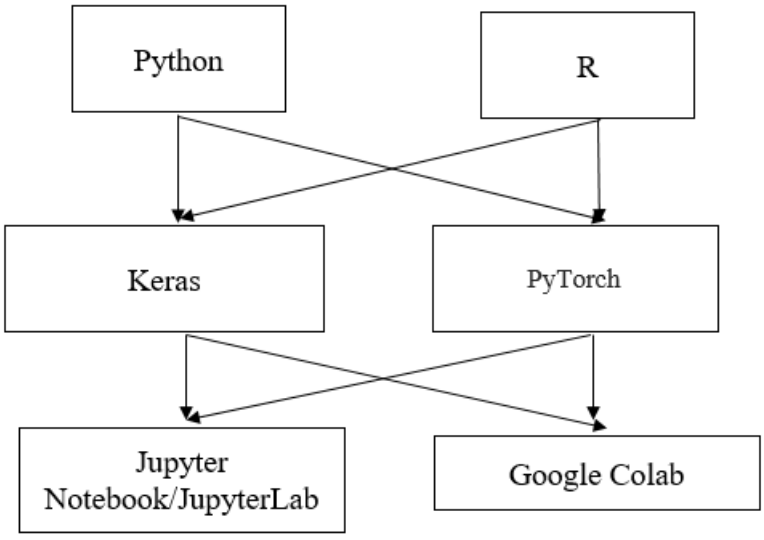


Рисунок 4.1 – Морфологічна карта

Морфологічна карта відображає множину всіх можливих варіантів основних функцій. Позитивно-негативна матриця показана в таблиці 4.1.

Таблиця 4.1 - Позитивно-негативна матриця

Функції	Варіанти реалізації	Переваги	Недоліки
F_1	А	Велика кількість бібліотек та модулів для аналізу даних; зручний синтаксис.	Вимагає додаткових зусиль для оптимізації та паралельної обробки даних; може бути повільним для великих обсягів даних.
	Б	Має велику кількість пакетів та бібліотек для статистичного аналізу та візуалізації даних; легке завантаження даних та робота з ними.	Обмежена підтримка паралельних обчислень.
F_2	*А*	Простий та інтуїтивно зрозумілий інтерфейс для побудови та навчання нейромереж; широкий спектр підтримуваних архітектур та моделей.	Обмежена гнучкість порівняно з іншими бібліотеками.
	Б	Гнучкість та зручність розробки, зокрема моделей нейромереж; легкість відладки.	Потребує більше зусиль для масштабування при роботі з великими наборами даних або розподіленими системами.
F_3	*А*	Інтерактивність та експериментування з кодом, даними та візуалізацією; легко поєднує код, текст, графіки та інші елементи для зручного дослідження та документування.	Обмежена продуктивність для обробки великих обсягів даних; потребує додаткових ресурсів для запуску на власному сервері.
	Б	Безкоштовне хостування та доступ до обчислювальних ресурсів Google; швидкий початок роботи завдяки передвстановленим бібліотекам.	Обмежені ресурси для великих проектів та обробки великих обсягів даних; необхідність інтернет-з'єднання для доступу та виконання коду.

На основі аналізу позитивно-негативної матриці робимо висновок, що при розробці програмного продукту деякі варіанти реалізації функцій варто відкинути, тому, що вони не відповідають поставленим перед програмним продуктом задачам. Ці варіанти відзначені у морфологічній карті.

Функція F_1:

Віддаємо перевагу простоті синтаксису. Для спрощення роботи по написанню коду варіант Б має бути відкинутий.

Функція F_2:

Програма допускає обрання обох варіантів. Можливо використати варіанти А чи Б.

Функція F_3:

Віддаємо перевагу інтерактивності та зручності дослідження і документування результатів. Це варіант А.

Таким чином, будемо розглядати такий варіанти реалізації ПП:

F_1а $-$ F_2а $-$ F_3а;
F_1а $-$ F_2б $-$ F_3а;

Для оцінювання якості розглянутих функцій обрана система параметрів, описана нижче.

4.3 Обґрунтування системи параметрів програмного продукту

На основі даних, розглянутих вище, визначаються основні параметри вибору, які будуть використані для розрахунку коефіцієнта технічного рівня.

Для того, щоб охарактеризувати програмний продукт, будемо використовувати наступні параметри:

X1 – швидкодія мови програмування;

X2 – об'єм пам'яті для обчислень та збереження даних;

X3 – час навчання даних;

X4 – потенційний об'єм програмного коду.

Гірші, середні і кращі значення параметрів вибираються на основі вимог замовника й умов, що характеризують експлуатацію програмного продукту, як показано у таблиці 4.2.

Таблиця 4.2 – Основні параметри програмного продукту

Назва Параметра	Умовні позначення	Одиниці виміру	Значення параметра		
			гірші	середні	кращі
Швидкодія мови програмування	X1	оп/мс	10000	14000	20000
Точність моделі	X2	%	50	80	99
Час попередньої обробки даних	X3	с	20	10	5
Час навчання моделі	X4	с	300	180	50

За даними таблиці 4.3 будуються графічні характеристики параметрів – рис. 4.2 – рис. 4.5.

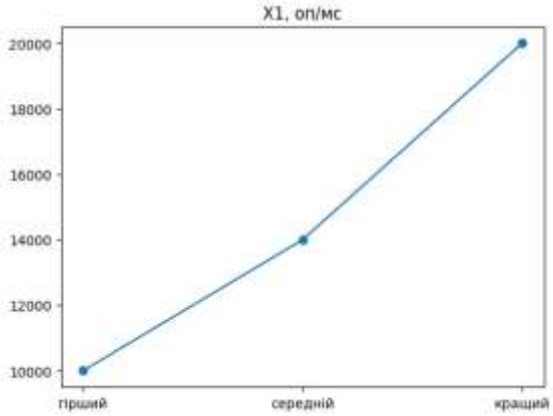


Рисунок 4.2 – X1, Швидкодія мови програмування

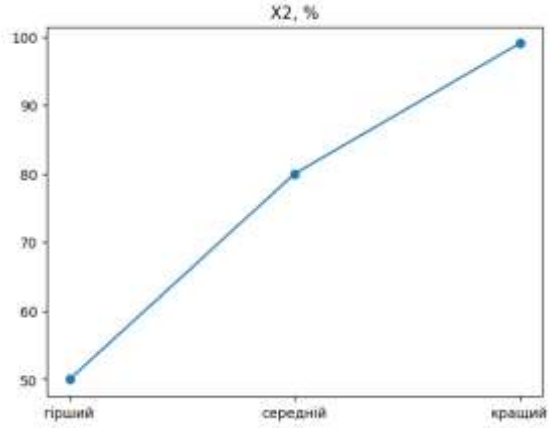


Рисунок 4.3 – X2, Точність моделі

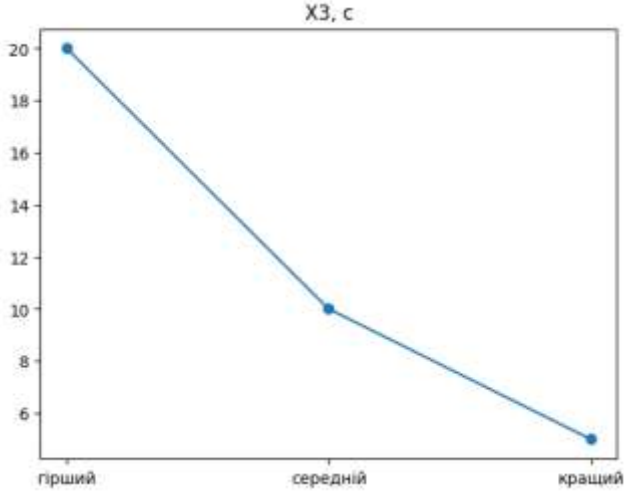


Рисунок 4.4 – X3, Час попередньої обробки даних

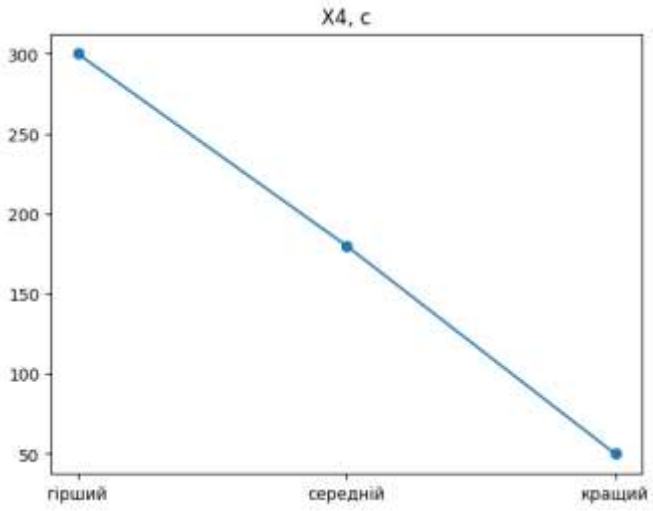


Рисунок 4.5 – X4, Час навчання моделі

4.4 Аналіз експертного оцінювання параметрів

Після детального обговорення й аналізу кожний експерт оцінює ступінь важливості кожного параметру для конкретно поставленої цілі – розробка програмного продукту, який дає найбільш точні результати при знаходженні параметрів моделей адаптивного прогнозування і обчислення прогнозних значень.

Значимість кожного параметра визначається методом попарного порівняння. Оцінку проводить експертна комісія із 7 людей. Визначення коефіцієнтів значимості передбачає:

- визначення рівня значимості параметра шляхом присвоєння різних рангів;
- перевірку придатності експертних оцінок для подальшого використання;
- визначення оцінки попарного пріоритету параметрів;

- обробку результатів та визначення коефіцієнту значимості.

Результати експертного ранжування наведені у таблиці 4.3.

Таблиця 4.3 - Результати ранжування параметрів

Позначення параметра	Назва параметра	Одиниці виміру	Ранг параметра за оцінкою експерта							Сума рангів R_i	Відхилення Δ_i	Δ_i^2
			1	2	3	4	5	6	7			
X1	Швидкодія мови програмування	оп/мс	4	2	4	3	4	3	3	23	5,5	30,25
X2	Точність моделі	%	1	1	2	2	1	1	2	10	-7,5	56,25
X3	Час попередньої обробки даних	с	3	4	3	4	3	4	4	25	7,5	56,25
X4	Час навчання моделі	с	2	3	1	1	2	2	1	12	-5,5	30,25
	Разом		10	10	10	10	10	10	10	70	0	173

Для перевірки степені достовірності експертних оцінок, визначимо наступні параметри:

а) сума рангів кожного з параметрів і загальна сума рангів:

$$R_i = \sum_{j=1}^{N} r_{ij} R_{ij} = \frac{Nn(n+1)}{2} = 70, \qquad (4.1)$$

де N – число експертів, n – кількість параметрів;

б) середня сума рангів:

$$T = \frac{1}{n} R_{ij} = 17{,}5 \tag{4.2}$$

в) відхилення суми рангів кожного параметра від середньої суми рангів:

$$\Delta_i = R_i - T. \tag{4.3}$$

Сума відхилень по всім параметрам повинна дорівнювати 0;

г)загальна сума квадратів відхилення:

$$S = \sum_{i=1}^{N} \Delta_i^2 = 173 \tag{4.4}$$

Порахуємо коефіцієнт узгодженості:

$$W = \frac{12S}{N^2(n^3 - n)} = \frac{12 \cdot 173}{7^2(4^3 - 4)} = 0{,}7061 > W_k = 0{,}67. \tag{4.5}$$

Ранжування можна вважати достовірним, тому що знайдений коефіцієнт узгодженості перевищує нормативний, котрий дорівнює 0,67.

Скориставшись результатами ранжирування, проведемо попарне порівняння всіх параметрів і результати занесемо у таблицю 4.4.

Таблиця 4.4 - Попарне порівняння параметрів.

Параметри	Експерти							Кінцева оцінка	Числове значення
	1	2	3	4	5	6	7		
X1 і X2	<	<	<	<	<	<	<	<	0,5
X1 і X3	<	>	<	>	<	>	>	>	1,5
X1 і X4	<	>	<	<	<	<	<	<	0,5
X2 і X3	>	>	>	>	>	>	>	>	1,5
X2 і X4	>	>	<	<	>	>	<	>	1,5
X3 і X4	<	<	<	<	<	<	<	<	0,5

Числове значення, що визначає ступінь переваги *i*–го параметра над *j*–тим, a_{ij} визначається по формулі:

$$a_{ij} = \begin{cases} 1.5 \text{ при } X_i > X_j \\ 1.0 \text{ при } X_i = X_j \\ 0.5 \text{ при } X_i < X_j \end{cases}. \quad (4.6)$$

З отриманих числових оцінок переваги складемо матрицю $A = \| a_{ij} \|$.

Для кожного параметра зробимо розрахунок вагомості $K_{\text{ві}}$ за наступними формулами:

$$K_{\text{ві}} = \frac{b_i}{\sum_{i=1}^{n} b_i} \quad (4.7)$$

$$b_i = \sum_{i=1}^{N} a_{ij} \quad (4.8)$$

Відносні оцінки розраховуються декілька разів доти, поки наступні значення не будуть незначно відрізнятися від попередніх (менше 2%). На другому і наступних кроках відносні оцінки розраховуються за наступними формулами:

$$K_{\text{ві}} = \frac{b_i'}{\sum_{i=1}^{n} b_i'}, \tag{4.9}$$

$$b_i' = \sum_{i=1}^{N} a_{ij} b_j \tag{4.10}$$

Як видно з таблиці 4.5, різниця значень коефіцієнтів вагомості не перевищує 2%, тому більшої кількості ітерацій не потрібно.

Таблиця 4.5 - Розрахунок вагомості параметрів

Параметри x_i	Параметри x_j				Перша ітер.		Друга ітер.	
	X1	X2	X3	X4	b_i	$K_{\text{ві}}$	b_i^1	$K_{\text{ві}}^1$
X1	1	0,5	1,5	0,5	3,5	0,21875	12,25	0,2076
X2	1,5	1	1,5	1,5	5,5	0,34375	21,25	0,3602
X3	0,5	0,5	1	0,5	2,5	0,15625	9,25	0,1568
X4	1,5	0,5	1,5	1	4,5	0,28125	16,25	0,2754
Всього:					16	1	59	1

4.5 Аналіз рівня якості варіантів реалізації функцій

Визначаємо рівень якості кожного варіанту виконання основних функцій окремо.

Абсолютні значення параметрів *X2*, *X3* та *X4* відповідають технічним вимогам умов функціонування даного ПП.

Абсолютне значення параметра *X1* (швидкість роботи мови програмування) обрано не найгіршим.

Коефіцієнт технічного рівня для кожного варіанта реалізації ПП розраховується так (таблиця 4.6):

$$K_K(j) = \sum_{i=1}^{n} K_{вi,j} B_{i,j}\,, \qquad (4.11)$$

де n – кількість параметрів;

$K_{вi}$– коефіцієнт вагомості i–го параметра;

B_i – оцінка i–го параметра в балах.

Таблиця 4.6 - Розрахунок показників рівня якості варіантів реалізації основних функцій ПП

Основні функції	Варіант реалізації функції	Параметри	Абсолютне значення параметра	Бальна оцінка параметра	Коефіцієнт вагомості параметра	Коефіцієнт рівня якості
F1	А	X1	15000	11	0,2076	2,2836
F3	А	X2	80	23	0,3602	8,2846
	Б	X3	10	9	0,1568	1,4112
F4	А	X4	120	18	0,2754	4,9572

За даними з таблиці 4.6 за формулою:

$$K_K = K_{\text{ТУ}}[F_{1k}] + K_{\text{ТУ}}[F_{2k}] + \ldots + K_{\text{ТУ}}[F_{zk}], \qquad (4.12)$$

визначаємо рівень якості кожного з варіантів:

$$\text{К}_{\text{К1}} = 2{,}2836 + 8{,}2846 + 4{,}9572 = 15{,}5254;$$
$$\text{К}_{\text{К2}} = 2{,}2836 + 1{,}4112 + 4{,}9572 = 8{,}6520.$$

Як видно з розрахунків, кращим є 1 варіант, для якого коефіцієнт технічного рівня має найбільше значення.

4.6 Економічний аналіз варіантів розробки ПП

Для визначення вартості розробки ПП спочатку проведемо розрахунок трудомісткості.

Всі варіанти включають в себе два окремих завдання:

1. Розробка проекту програмного продукту;
2. Розробка програмної оболонки;

Завдання 1 за ступенем новизни відноситься до групи А, завдання 2 – до групи Б. За складністю алгоритми, які використовуються в завданні 1 належать до групи 1; а в завданні 2 – до групи 3.

Для реалізації завдання 1 використовується довідкова інформація, а завдання 2 використовує інформацію у вигляді даних.

Проведемо розрахунок норм часу на розробку та програмування для кожного з завдань.

Загальна трудомісткість обчислюється як:

$$T_O = T_P \cdot K_П \cdot K_{СК} \cdot K_М \cdot K_{СТ} \cdot K_{СТ.М}, \quad (4.13)$$

де T_P – трудомісткість розробки ПП;

$K_П$ – поправочний коефіцієнт;

$K_{СК}$ – коефіцієнт на складність вхідної інформації;

$K_М$ – коефіцієнт рівня мови програмування;

$K_{СТ}$ – коефіцієнт використання стандартних модулів і прикладних програм;

$K_{СТ.М}$ – коефіцієнт стандартного математичного забезпечення

Для першого завдання, виходячи із норм часу для завдань розрахункового характеру степеню новизни А та групи складності алгоритму 1, трудомісткість дорівнює: T_P =51 людино-днів. Поправочний коефіцієнт, який враховує вид нормативно-довідкової інформації для першого завдання: $К_П$ = 1,1. Поправочний коефіцієнт, який враховує складність контролю вхідної та вихідної інформації для всіх завдань рівний 1: $К_{СК}$ = 1. Оскільки при розробці першого завдання використовуються стандартні модулі, врахуємо це за допомогою коефіцієнта $К_{СТ}$ = 0,8. Тоді загальна трудомісткість програмування першого завдання дорівнює:

$$T_1 = 51 \cdot 1{,}1 \cdot 0{,}8 = 44{,}88 \text{ людино-днів.}$$

Проведемо аналогічні розрахунки для подальших завдань.

Для другого завдання (використовується алгоритм третьої групи складності, степінь новизни Б), тобто T_P = 25 людино-днів, $К_П$ = 1,0, $К_{СК}$ = 1, $К_{СТ}$ = 0,8:

$$T_2 = 25 \cdot 1 \cdot 0{,}8 = 20 \text{ людино-днів.}$$

Складаємо трудомісткість відповідних завдань для кожного з обраних варіантів реалізації програми, щоб отримати їх трудомісткість:

$$T_I = (44{,}88 + 20 + 15{,}53 + 20) \cdot 8 = 803{,}28 \text{ людино-годин.}$$
$$T_{II} = (44{,}88 + 20 + 8{,}65 + 20) \cdot 8 = 748{,}24 \text{ людино-годин.}$$

Найбільш високу трудомісткість має варіант I.

В розробці беруть участь один програмісти з окладом 21000 грн., один аналітик в області даних з окладом 20500 грн. Визначимо середню зарплату за годину за формулою:

$$С_ч = \frac{М}{T_m \cdot t} грн., \quad (4.14)$$

де М – місячний оклад працівників;

T_m – кількість робочих днів тиждень;

t – кількість робочих годин в день.

$$С_ч = \frac{21000 + 20500}{3 \cdot 21 \cdot 8} = 82{,}34 \text{ грн.} \quad (4.15)$$

Тоді, розрахуємо заробітну плату за формулою:

$$С_{ЗП} = С_ч \cdot Т_i \cdot К_Д, \quad (4.16)$$

де $С_ч$– величина погодинної оплати праці програміста;

$Т_i$ – трудомісткість відповідного завдання;

$К_Д$ – норматив, який враховує додаткову заробітну плату.

Зарплата розробників за варіантами становить:

I. $С_{ЗП} = 82{,}34 \cdot 803{,}28 \cdot 1.2 = 79370{,}49$ грн.

II. $С_{ЗП} = 82{,}34 \cdot 748{,}24 \cdot 1.2 = 73932{,}09$ грн.

Відрахування на єдиний соціальний внесок становить 22%:

I. $С_{ВІД} = С_{ЗП} \cdot 0.22 = 79370{,}49 \cdot 0.22 = 17454{,}9$ грн.

$$\text{II. } С_{ВІД} = С_{ЗП} \cdot 0.22 = 73932{,}09 \cdot 0.22 = 16265{,}05 \text{ грн.}$$

Тепер визначимо витрати на оплату однієї машино-години. ($С_М$)

Так як одна ЕОМ обслуговує одного програміста з окладом 21000 грн., з коефіцієнтом зайнятості 0,2 то для однієї машини отримаємо:

$$С_Г = 12 \cdot M \cdot K_З = 12 \cdot 21000 \cdot 0{,}2 = 50400 \text{ грн.}$$

З урахуванням додаткової заробітної плати:

$$С_{ЗП} = С_Г \cdot (1 + K_З) = 50400 \cdot (1 + 0{,}2) = 60480 \text{ грн.}$$

Відрахування на соціальний внесок:

$$С_{ВІД} = С_{ЗП} \cdot 0{,}22 = 60480 \cdot 0{,}22 = 13305{,}6 \text{ грн.}$$

Амортизаційні відрахування розраховуємо при амортизації 25% та вартості ЕОМ – 23000 грн.

$$С_А = К_{ТМ} \cdot К_А \cdot Ц_{ПР} = 1{,}4 \cdot 0{,}25 \cdot 23000 = 8050 \text{ грн.,}$$

де $К_{ТМ}$– коефіцієнт, який враховує витрати на транспортування та монтаж приладу у користувача;

$К_А$– річна норма амортизації;

$Ц_{ПР}$– договірна ціна приладу.

Витрати на ремонт та профілактику розраховуємо як:

$$С_Р = К_{ТМ} \cdot Ц_{ПР} \cdot К_Р = 1{,}4 \cdot 23000 \cdot 0{,}08 = 2676 \text{ грн.},$$

де $К_Р$– відсоток витрат на поточні ремонти.

Ефективний годинний фонд часу ПК за рік розраховуємо за формулою:

$$Т_{ЕФ} = (Д_К - Д_В - Д_С - Д_Р) \cdot t_З \cdot К_В = (365 - 104 - 12 - 16) \cdot 8 \cdot 0.7 = \\ = 1304{,}8 \text{ години},$$

де $Д_К$ – календарна кількість днів у році;

$Д_В$, $Д_С$ – відповідно кількість вихідних та святкових днів;

$Д_Р$ – кількість днів планових ремонтів устаткування;

t –кількість робочих годин в день;

$К_В$– коефіцієнт використання приладу у часі протягом зміни.

Витрати на оплату електроенергії розраховуємо за формулою:

$$С_{ЕЛ} = Т_{ЕФ} \cdot N_С \cdot K_З \cdot Ц_{ЕН} = 1304{,}8 \cdot 0{,}2 \cdot 0{,}8 \cdot 4{,}87 = 1016{,}7 \text{ грн.},$$

де $N_С$ – середньо-споживча потужність приладу;

$К_З$– коефіцієнтом зайнятості приладу;

$Ц_{ЕН}$ – тариф за 1 КВт-годин електроенергії.

Накладні витрати розраховуємо за формулою:

$$С_Н = Ц_{ПР} \cdot 0{,}67 = 23000 \cdot 0{,}67 = 15410 \text{ грн.}$$

Тоді, річні експлуатаційні витрати будуть:

$$C_{ЕКС} = C_{ЗП} + C_{ВІД} + C_{А} + C_{Р} + C_{ЕЛ} + C_{Н}, \quad (4.17)$$

$$C_{ЕКС} = 60480 + 13305{,}6 + 8050 + 2676 + 1016{,}7 + 15410 = 100938{,}3 \text{ грн.}$$

Собівартість однієї машино-години ЕОМ дорівнюватиме:

$$C_{М-Г} = C_{ЕКС}/T_{ЕФ} = 100938{,}3/1016{,}7 = 99{,}28 \text{ грн/год.}$$

Оскільки в даному випадку всі роботи, які пов‘язані з розробкою програмного продукту ведуться на ЕОМ, витрати на оплату машинного часу, в залежності від обраного варіанта реалізації, складає:

$$C_{М} = C_{М-Г} \cdot T, \quad (4.18)$$

I. $C_{М} = 99{,}28 \cdot 803{,}28 = 79749{,}64$ грн.

II. $C_{М} = 99{,}28 \cdot 748{,}24 = 74285{,}27$ грн.

Накладні витрати складають 67% від заробітної плати:

$$C_{Н} = C_{ЗП} \cdot 0{,}67, \quad (4.19)$$

I. $C_{Н} = 79370{,}49 \cdot 0{,}67 = 53178{,}23$ грн.

II. $C_{Н} = 73932{,}09 \cdot 0{,}67 = 49534{,}5$ грн.

Отже, вартість розробки ПП за варіантами становить:

$$C_{ПП} = C_{ЗП} + C_{ВІД} + C_{М} + C_{Н}, \quad (4.20)$$

I. $C_{ПП} = 79370{,}49 + 17454{,}9 + 79749{,}64 + 53178{,}23 = 229753{,}26$ грн.

II. $C_{ПП} = 73932{,}09 + 16265{,}05 + 74285{,}27 + 49534{,}5 = 214016{,}91$ грн.

4.7 Вибір кращого варіанту ПП техніко-економічного рівня

Розрахуємо коефіцієнт техніко-економічного рівня за формулою:

$$К_{ТЕР_j} = К_{Кj} / С_{Фj}, \quad (4.21)$$

$$К_{ТЕР1} = 15{,}5254/229753{,}26 = 6{,}7574 \cdot 10^{-5},$$

$$К_{ТЕР2} = 8{,}6520/214016{,}91 = 4{,}0427 \cdot 10^{-5}.$$

Як бачимо, найбільш ефективним є перший варіант реалізації програми з коефіцієнтом техніко-економічного рівня $К_{ТЕР1} = 6{,}7574 \cdot 10^{-5}$.

Після виконання функціонально-вартісного аналізу програмного комплексу що розроблюється, можна зробити висновок, що з альтернатив, що залишились після першого відбору двох варіантів виконання програмного комплексу оптимальним є перший варіант реалізації програмного продукту. У нього виявився найкращий показник техніко-економічного рівня якості $К_{ТЕР1} = 6{,}7574 \cdot 10^{-5}$.

Цей варіант реалізації програмного продукту має такі параметри:

- Вибір мови програмування – Python;
- Вибір фреймворку для глибокого навчання – Keras;
- Вибір середовища програмування – Jupyter Notebook/JupyterLab.

Даний варіант виконання програмного комплексу дозволяє розробнику мати більш зручні можливості для розробки, а користувачу зручні для дослідження та документування результати у вигляді текстових та графічних даних.

4.8 Висновки до четвертого розділу

В даній частині було проведено повний функціонально-вартісний аналіз програмного продукту. Також було знайдено оцінку основних функцій програмного продукту.

В результаті виконання функціонально-вартісного аналізу програмного комплексу що розроблюється, було визначено та проведено оцінку основних функцій програмного продукту, а також знайдено параметри, які його характеризують.

На основі аналізу вибрано варіант реалізації програмного продукту.

ВИСНОВОК

В цій роботі було проведено порівняльний аналіз чотирьох алгоритмів оптимізації глибокого навчання: Stochastic Gradient Descent (SGD), Adam, AdaGrad і RMSprop, в контексті задач прогнозування. Для цього були використані різноманітні набори даних та проведені вимірювання метрик для оцінки результатів.

В результаті аналізу можна зробити висновок, що алгоритм оптимізації RMSprop є найефективнішим у більшості розглянутих задач. Він досягав найвищої точності прогнозування та мав кращі значення метрик, таких як Accuracy, Precision, Recall та F1 Score.

Лиш трохи поступаючись йому, Adam також показав гарні результати в більшості задач. Як і RMSprop, він демонстрував загалом стабільну та швидку збіжність, що робить його привабливим вибором для глибокого навчання.

Слід відзначити, що SGD також показав добрі результати в деяких задачах, особливо в задачі регресії, де він мав найнижчу втрату та кращі значення метрик. Проте він показав дещо гіршу стабільність збіжності в порівнянні з двома іншими.

Алгоритм AdaGrad також показали прийнятні результати, але його ефективність часто була меншою порівняно з іншими алгоритмами. Він досяг менших значень точності прогнозування та втрат в деяких задачах, особливо в задачах класифікації. Проте, цей метод може бути доцільним для використання варіантом в деяких специфічних випадках.

Також важливим є те, що алгоритми SGD та AdaGrad показали загалом кращий час виконання глибокого навчання. Це можна пов’язати з тим, що

Adam та RMSprop вимагають більше часу для обчислень, оскільки вони включають додаткові операції, такі як обчислення моменту або експоненційно зваженого квадратичного середнього.

Загалом, на підставі цього дослідження можна рекомендувати використання алгоритму RMSprop для глибокого навчання в задачах прогнозування. Проте, вибір оптимального алгоритму може залежати від конкретної задачі та набору даних, тому рекомендується провести додатковий аналіз та експерименти для визначення найкращого варіанту в конкретному випадку.

Для подальшого розвитку роботи можна збільшити кількість задач прогнозування, розглянути продуктивність моделей з іншими архітектурами, або, наприклад, врахувати більше метрик.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Boston Housing. *Kaggle*. URL: https://www.kaggle.com/c/boston-housing
2. IMDb Movie Reviews. *PapersWithCode*. URL: https://paperswithcode.com/dataset/imdb-movie-reviews
3. Alex Krizhevsky. The CIFAR-10 dataset. URL: https://www.cs.toronto.edu/~kriz/cifar.html
4. MNIST database. *Wikipedia*. URL: https://en.wikipedia.org/wiki/MNIST_database
5. Tim Hargreaves. Is it Time to Ditch the MNIST Dataset?. *Ttested*. URL: https://www.ttested.com/ditch-mnist/
6. Martin Thoma. The Reuters Dataset. 2017. URL: https://martin-thoma.com/nlp-reuters/
7. Ian Goodfellow, Yoshua Bengio, Aaron Courville. Deep Learning (Adaptive Computation and Machine Learning series). Cambridge, Massachusetts, London, England: MIT Press, 2016. 801 p.
8. Daniel Villarraga. AdaGrad. *Cornell University Computational Optimization Open Textbook*. 2021. URL: https://optimization.cbe.cornell.edu/index.php?title=AdaGrad
9. Tieleman, T. and Hinton, G. Lecture 6.5 «RMSprop: Divide the Gradient by a Running Average of Its Recent Magnitude». COURSERA: Neural Networks for Machine Learning, 2012. URL: https://www.youtube.com/watch?v=defQQqkXEfE&t=1s&ab_channel=ColinReckons

10. Jason Brownlee. Gentle Introduction to the Adam Optimization Algorithm for Deep Learning. *Machine Learning Mastery*. July 3, 2017 (upd. January 13, 2021). URL: https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/

11. Diederik P. Kingma, Jimmy Ba, Adam: A Method for Stochastic Optimization. International Conference on Learning Representations, 2015. URL: https://arxiv.org/abs/1412.6980

12. Матриця невідповідностей. *Wikipedia*. URL: https://uk.wikipedia.org/wiki/Матриця_невідповідностей

13. Accuracy and precision. *Wikipedia*. URL: https://en.wikipedia.org/wiki/Accuracy_and_precision

14. Precision and recall. *Wikipedia*. URL: https://en.wikipedia.org/wiki/Precision_and_recall

15. F-score. *Wikipedia*. URL: https://en.wikipedia.org/wiki/F-score

16. Shervin Minaee. Popular Machine Learning Metrics. Part 1: Classification & Regression Evaluation Metrics. *Towards Data Science*: October 28, 2019. URL: https://towardsdatascience.com/20-popular-machine-learning-metrics-part-1-classification-regression-evaluation-metrics-1ca3e282a2ce

17. Коефіцієнт детермінації. *Wikipedia*. URL: https://uk.wikipedia.org/wiki/Коефіцієнт_детермінації

18. Tensorflow Keras documentation. URL: https://www.tensorflow.org/api_docs/python/tf/keras

19. Ayush Gupta. A Comprehensive Guide on Optimizers in Deep Learning. *Analytics Vidghya*: October 7, 2021 (upd. May 18, 2023). URL:https://www.analyticsvidhya.com/blog/2021/10/a-comprehensive-

guide-on-deep-learning-optimizers/#What_Are_Optimizers_in_Deep_Learning?

20. Optimization Algorithms. *Dive Into Deep Learning*. URL: https://d2l.ai/chapter_optimization/

21. Jean de Dieu Nyandwi. The Ultimate Guide on Choosing the Right Neural Network Architecture For the Right Data. *Medium*: September 9, 2021. URL: https://jeande.medium.com/the-ultimate-guide-on-choosing-the-right-neural-network-architecture-for-the-right-data-e6dac305836e

22. Avijeet Biswal. Top 10 Deep Learning Algorithms You Should Know in 2023. *simplilearn*: February 16, 2023 URL: https://www.simplilearn.com/tutorials/deep-learning-tutorial/deep-learning-algorithm

ДОДАТОК А ПРЕЗЕНТАЦІЯ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«Київський Політехнічний Інститут
ім. Ігоря Сікорського»

НН Інститут прикладного системного аналізу

ДОСЛІДЖЕННЯ ТА ПОРІВНЯЛЬНИЙ АНАЛІЗ АЛГОРИТМІВ ГЛИБОКОГО НАВЧАННЯ Stochastic Gradient Method, Adam, AdaGrad, RMSprop В ЗАДАЧАХ ПРОГНОЗУВАННЯ

Виконав: студент IV курсу, групи КА-93
Копа Максим Вікторович

Керівник: професор, д.т.н.
Зайченко Юрій Петрович

Київ – 2023

АКТУАЛЬНІСТЬ РОБОТИ

Актуальність даної роботи полягає в покращенні результатів глибокого навчання та розвитку галузі в цілому. Порівняльний аналіз алгоритмів глибокого навчання допомагає зрозуміти, як впливають різні методи оптимізації на якість та швидкість навчання моделей. Це дає можливість вибрати найкращий алгоритм для конкретної задачі, забезпечуючи кращі результати та ефективне використання ресурсів. Дослідження у цій галузі може сприяти покращенню розуміння алгоритмів глибокого навчання та впровадженню передових практик в розробку майбутніх моделей.

3

ПОСТАНОВКА ЗАДАЧІ

- ***Об'єктом дослідження*** є алгоритми оптимізації глибокого навчання: метод стохастичного градієнтного спуску (SGD), Adam, AdaGrad та RMSprop.
- ***Предмет дослідження*** є глибоке навчання в задачах прогнозування.
- ***Метою роботи*** є дослідження, визначення та порівняти ефективності чотирьох алгоритмів оптимізації глибокого навчання SGD, Adam, AdaGrad та RMSprop в задачах прогнозування.

4

ПОСТАНОВКА ЗАДАЧІ

Для проведення порівняльного аналізу методів глибокого навчання SGD, Adam, AdaGrad і RMSprop в задачах прогнозування, можна виконати наступні кроки:

5

1. Ініціалізувати параметри моделі θ.
2. Ініціалізувати швидкість навчання α.
3. Повторювати до досягнення критерію зупинки:
   - Вибрати випадковий піднабір тренувальних даних (x, y).
   - Обчислити прогнозоване значення моделі $f(x; \theta)$.
   - Обчислити градієнт функції втрат $\nabla L(\theta; x, y)$ по відношенню до параметрів моделі.
   - Оновити параметри моделі: $\theta \leftarrow \theta - \alpha \cdot \nabla L(\theta; x, y)$.
4. Повернути оптимальні значення параметрів моделі θ.

Stochastic Gradient Descent

6

1. Ініціалізувати швидкість навчання α; зазвичай мале значення (наприклад, 0.01).
2. Ініціалізувати лічильник часу $t = 0$.
3. Ініціалізувати суму квадратів градієнтів G_0 на нулі.
4. На кожній ітерації t до досягнення критерію зупинки:
   - Обчислити градієнт g_t моделі на основі поточної партії даних
   - Збільшити лічильник часу: $t \leftarrow t + 1$.
   - Оновити суму квадратів градієнтів: $G_t \leftarrow G_{t-1} + g_t \odot g_t$.
   - Оновити параметри моделі: $\theta_t \leftarrow \theta_{t-1} - \frac{\alpha}{\sqrt{G_t + \varepsilon}} \odot g_t$, де ε - дуже мала константа для недопущення ділення на нуль.
5. Повернути оптимальні значення параметрів моделі θ.

AdaGrad

7

RMSprop

1. Ініціалізувати параметри моделі θ.
2. Ініціалізувати швидкість навчання α.
3. Ініціалізувати параметр затухання β (зазвичай близько до 0.9).
4. Ініціалізувати змінну $r = 0$, яка зберігатиме поточні накопичені квадрати градієнтів.
5. Повторювати до досягнення критерію зупинки:
   - Вибрати випадковий піднабір тренувальних даних (x, y).
   - Обчислити прогнозоване значення моделі $f(x;\ \theta)$.
   - Обчислити градієнт функції втрат $\nabla L(\theta;\ x, y)$ по відношенню до параметрів моделі.
   - Оновити накопичені квадрати градієнтів: $r \leftarrow \beta \cdot r + (1-\beta) \cdot \left(\nabla L(\theta;\ x, y)\right)^2$.
   - Оновити параметри моделі: $\theta = \theta - \frac{\alpha}{\sqrt{r+\varepsilon}} \cdot \nabla L(\theta;\ x, y)$, де ε - дуже мале число для стабільності обчислень.
6. Повернути оптимальні значення параметрів моделі θ.

8

Adam

1. Ініціалізувати швидкість навчання α (зазвичай невелике значення, наприклад, 0.01).
2. Ініціалізувати експоненціальні коефіцієнти згладжування β_1 та β_2 в проміжку $[0, 1)$ (зазвичай $\beta_1 = 0.9, \beta_2 = 0.999$).
3. Ініціалізувати лічильник часу $t = 0$.
4. Ініціалізувати перші моменти градієнтів $m_0 = 0$ та другі моменти градієнтів $v_0 = 0$.
5. На кожній ітерації t:
   - Обчислити градієнт g_t моделі на підставі поточної партії даних.
   - Збільшити лічильник часу: $t \leftarrow t + 1$.
   - Оновити перший та другий моменти градієнтів: $m_t \leftarrow \beta_1 \cdot m_{t-1} + (1-\beta_1) \cdot g_t$; $v_t \leftarrow \beta_2 \cdot v_{t-1} + (1-\beta_2) \cdot g_t^2$
   - Виправити зміщення моментів: $\hat{m}_t \leftarrow m_t\ /\ (1-\beta_1^t)$; $\hat{v}_t \leftarrow v_t\ /(1-\beta_2^t)$.
   - Оновити параметри моделі: $\theta_t \leftarrow \theta_{t+1} - \alpha \cdot \frac{\hat{m}_t}{\sqrt{\hat{v}_t}+\varepsilon}$, де ε - мале число для недопущення ділення на нуль.
6. Повернути оптимальні значення параметрів моделі θ.

9

Структуровані дані – набір даних «Boston Housing»:

- Завданням є прогнозування середньої вартості житла в різних районах на основі певних характеристик.

Текстові дані – набір даних «IMDB Movie Reviews»:

- Задача прогнозування полягає в класифікації відгуків на позитивні та негативні на основі текстових оглядів фільмів.

Зображення – набір даних «CIFAR-10»:

- В даній задачі потрібно класифікувати зображення на 10 категорій об'єктів, таких як автомобілі, літаки, собаки тощо.

Зображення – набір даних «MNIST»:

- Задача прогнозування: Розпізнавання рукописних цифр на зображеннях набору даних «MNIST». Модель машинного навчання повинна класифікувати зображення цифр на числові категорії від 0 до 9.

Текстові дані – набір даних «Reuters»:

- Задача прогнозування: Класифікація новини на основі їх текстового змісту і передбачати їх категорію або тему.

10

Boston Housing

- Dense з 64 нейронами, що використовує функцію активації ReLU (Rectified Linear Unit).
- Dense з 64 нейронами і функцією активації ReLU.
- Dense шар з одним нейроном.

IMDB Movie Reviews

- Embedding шар. Має розмір словника 10 000 токенів і генерує вектори довжиною 128 для кожного токену.
- LSTM (Long Short-Term Memory) шар з 128 нейронами.
- Dense шар з одним нейроном і функцією активації sigmoid.

CIFAR-10

- Conv2D шар. Має 32 фільтри з розміром ядра 3x3, використовує функцію активації ReLU.
- MaxPooling2D шар, який зменшує роздільність зображення.
- Аналогічний другий блок з Conv2D з 64 фільтрами, MaxPooling2D і функцією активації ReLU.
- Flatten шар.
- Dense з 10 нейронами і функцією активації softmax.

MNIST

- Flatten для перетворення вхідних зображень з форми (28, 28) в одновимірний вектор.
- Dense з 128 нейронами і функцією активації ReLU.
- Dense з 10 нейронами і функцією активації softmax.

Reuters

- Dense шар з 512 нейронами і функцією активації ReLU.
- Dropout шар з розміром 0.5.
- Dense з 256 нейронами і функцією активації ReLU.
- Dropout з розміром 0.5.
- Dense шар з num_classes нейронами і функцією активації softmax.

11

ТРЕНУВАННЯ МОДЕЛЕЙ

На даному етапі були проводиться компіляція та тренування (навчання) всіх моделей.

Для компіляції моделей було використано функцію model.compile() з бібліотеки tensorflow.keras. Ця функція встановлює параметри, які визначають як модель буде навчатись та оцінюватись. В задачі регресії були використані метрики MAE, MSE, R2 score та Explained Variance Score. Для задач класифікації, в свою чергу, замірювались метрики Accuracy, Precision, Recall та F1-score.

Після компіляції, модель готова до навчання. Навчання проводилось за допомогою функції model.fit() з тієї ж бібліотеки.

Для ефективного порівняння моделі з різними оптимізаторами, на етапі навчання моделі були зафіксовані значення втрат та метрик, обчислені на тренувальних та валідаційних даних. Валідаційні дані використовуються для оцінки продуктивності моделі під час навчання і дозволяють порівнювати різні методи оптимізації в контексті того, як вони впливають на втрати та метрики моделі.

З іншого боку, після навчання моделі найкраще проводити порівняння на незалежних тестових даних, які не брали участь під час навчання та валідації моделі. Використання таких даних дозволяє оцінити реальну продуктивність моделі на нових прикладах, які раніше не були побачені моделлю, допомагаючи уникненню перенавчання і визначенню того, як добре модель генералізує на нові дані.

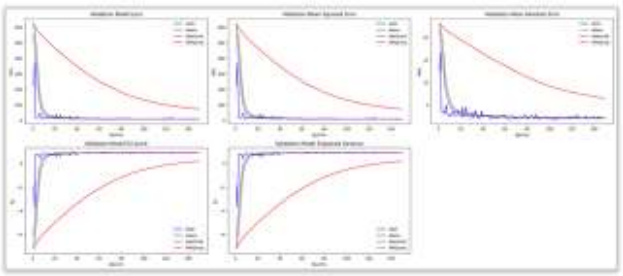
12

ВИМІРЮВАННЯ МЕТРИК

Boston Housing

Розмір тренувальних даних: (404, 13)
Розмір тестових даних: (102, 13)

Алгоритм оптимізації	Втрати (loss)	MAE	MSE	R2 score	Explained Variance	Час виконання	Кількість епох
SGD	10.8916	2.3318	10.8916	0.7969	0.7969	12.86 sec	150
Adam	18.5565	2.6977	18.5565	0.5178	0.5178	8.88 sec	
AdaGrad	69.1414	6.8074	69.1414	0.0230	0.0230	9.53 sec	
RMSprop	15.3131	2.5649	15.3131	0.6317	0.6317	9.15 sec	



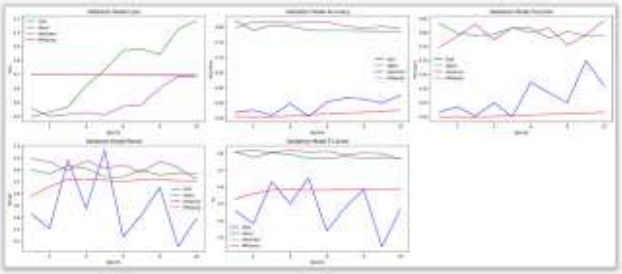
13

ВИМІРЮВАННЯ МЕТРИК

IMDb Movie Reviews

Розмір тренувальних даних: (25000, 60)
Розмір тестових даних: (25000, 60)

Алгоритм оптимізації	Втрати (loss)	Accuracy	Precision	Recall	F1 score	Час виконання	Кількість епох
SGD	0.6908	0.5650	0.6029	0.3809	0.4573	401.59 sec	10
Adam	1.0796	0.7867	0.8006	0.7637	0.7764	530.05 sec	
AdaGrad	0.6924	0.5419	0.5303	0.7339	0.6091	354.91 sec	
RMSprop	0.6782	0.7949	0.8506	0.7154	0.7714	424.68 sec	



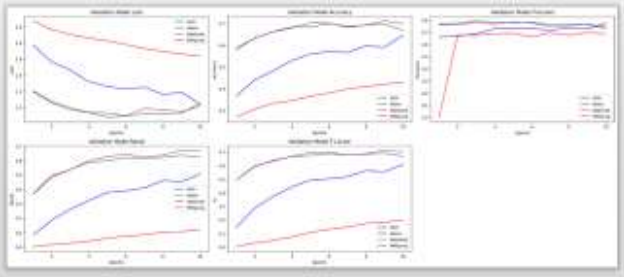
14

ВИМІРЮВАННЯ МЕТРИК

CIFAR-10

Розмір тренувальних даних: (50000, 32, 32, 3)
Розмір тестових даних: (10000, 32, 32, 3)

Алгоритм оптимізації	Втрати (loss)	Accuracy	Precision	Recall	F1 score	Час виконання	Кількість епох
SGD	1.0233	0.6405	0.7734	0.5069	0.6094	398.60 sec	10
Adam	1.0614	0.6622	0.7203	0.6089	0.6588	461.71 sec	
AdaGrad	1.6220	0.4353	0.7077	0.1213	0.2034	465.36 sec	
RMSprop	1.0189	0.6962	0.7426	0.6634	0.6999	483.40 sec	



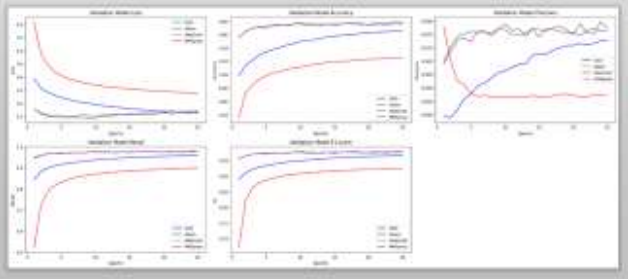
15

Розмір тренувальних даних: (60000, 28, 28)
Розмір тестових даних: (10000, 28, 28)

ВИМІРЮВАННЯ МЕТРИК

MNIST

Алгоритм оптимізації	Втрати (loss)	Accuracy	Precision	Recall	F1 score	Час виконання	Кількість епох
SGD	0.1201	0.9642	0.9717	0.9582	0.9648	111.67 sec	25
Adam	0.1159	0.9777	0.9783	0.9777	0.9780	135.93 sec	
AdaGrad	0.2760	0.9255	0.9501	0.9000	0.9236	126.16 sec	
RMSprop	0.1122	0.9795	0.9800	0.9793	0.9797	123.90 sec	



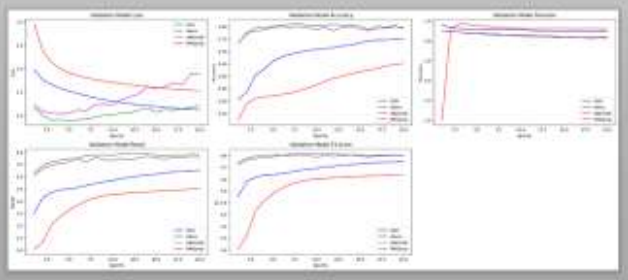
16

Розмір тренувальних даних: (8982, 1000)
Розмір тестових даних: (2246, 1000)

ВИМІРЮВАННЯ МЕТРИК

Reuters

Алгоритм оптимізації	Втрати (loss)	Accuracy	Precision	Recall	F1 score	Час виконання	Кількість епох
SGD	1.1666	0.7324	0.8903	0.6362	0.7353	43.70 sec	20
Adam	1.3309	0.7858	0.8161	0.7685	0.7899	72.99 sec	
AdaGrad	1.5598	0.6456	0.9058	0.4969	0.6348	53.41 sec	
RMSprop	2.2609	0.7778	0.8163	0.7480	0.7787	83.41 sec	



ВИСНОВОК

В результаті аналізу можна зробити висновок, що алгоритм оптимізації RMSprop є найефективнішим у більшості розглянутих задач. Він досягав найвищої точності прогнозування та мав кращі значення метрик, таких як Accuracy, Precision, Recall та F1 Score.

Лиш трохи поступаючись йому, Adam також показав гарні результати в більшості задач. Як і RMSprop, він демонстрував загалом стабільну та швидку збіжність, що робить його привабливим вибором для глибокого навчання.

Слід відзначити, що SGD також показав добрі результати в деяких задачах, особливо в задачі регресії, де він мав найнижчу втрату та кращі значення метрик. Проте він показав дещо гіршу стабільність збіжності в порівнянні з двома іншими.

Алгоритм AdaGrad також показали прийнятні результати, але його ефективність часто була меншою порівняно з іншими алгоритмами. Він досяг менших значень точності прогнозування та втрат в деяких задачах, особливо в задачах класифікації. Проте, цей метод може бути доцільним для використання варіантом в деяких специфічних випадках.

Також важливим є те, що алгоритми SGD та AdaGrad показали загалом кращий час виконання глибокого навчання. Це можна пов'язати з тим, що Adam та RMSprop вимагають більше часу для обчислень, оскільки вони включають додаткові операції, такі як обчислення моменту або експоненційно зваженого квадратичного середнього.

ДЯКУЮ ЗА УВАГУ

ДОДАТОК Б ЛІСТИНГ ПРОГРАМИ

Завантаження бібліотек:

```
import time
import matplotlib.pyplot as plt
import tensorflow as tf

from tensorflow.keras.datasets import boston_housing, mnist, cifar10, imdb,
reuters
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.preprocessing.text import Tokenizer
from sklearn.preprocessing import StandardScaler
```

Функції метрик:

```
# Визначення функції R2-score
def r2_score(y_true, y_pred):
    residual = tf.reduce_sum(tf.square(tf.subtract(y_true, y_pred)))
    total = tf.reduce_sum(tf.square(tf.subtract(y_true, tf.reduce_mean(y_true))))
    r2 = tf.subtract(1.0, tf.divide(residual, total))
    return r2

# Визначення функції Explained Variance Score
def explained_variance(y_true, y_pred):
    total = tf.reduce_sum(tf.square(tf.subtract(y_true, tf.reduce_mean(y_true))))
    residual = tf.reduce_sum(tf.square(tf.subtract(y_true, y_pred)))
    evs = tf.subtract(1.0, tf.divide(residual, total))
    return evs

# Визначення функції f1_score
def f1_score(y_true, y_pred):
    true_positives = tf.reduce_sum(tf.round(tf.clip_by_value(y_true * y_pred, 0,
1)))
    predicted_positives = tf.reduce_sum(tf.round(tf.clip_by_value(y_pred, 0, 1)))
    possible_positives = tf.reduce_sum(tf.round(tf.clip_by_value(y_true, 0, 1)))
    precision = true_positives / (predicted_positives +
tf.keras.backend.epsilon())
    recall = true_positives / (possible_positives + tf.keras.backend.epsilon())
    f1 = 2 * (precision * recall) / (precision + recall +
tf.keras.backend.epsilon())
    return f1
```

Функція побудови графіків:

```
def plot(loss='val_loss', metric1='val_accuracy', metric2='val_precision',
metric3='val_recall', metric4='val_f1_score',
         titles=['Validation Model Loss', 'Validation Model Accuracy',
'Validation Model Precision', 'Validation Model Recall', 'Validation Model F1-
score'],
         ylabels=['Loss', 'Accuracy', 'Precision', 'Recall', 'F1']):

    # Графіки значень втрат та метрик
    epochs = range(1, len(history_sgd.history[loss]) + 1)
    plt.figure(figsize=(21, 9))

    # Графіку значень третьої loss
    plt.subplot(2, 3, 1)
    plt.plot(epochs, history_sgd.history[loss], 'b', label='SGD')
    plt.plot(epochs, history_adam.history[loss], 'g', label='Adam')
    plt.plot(epochs, history_adagrad.history[loss], 'r', label='AdaGrad')
    plt.plot(epochs, history_rmsprop.history[loss], 'm', label='RMSprop')
    plt.title(titles[0])
    plt.xlabel('Epochs')
    plt.ylabel(ylabels[0])
    plt.legend()

    # Графіку значень першої метрики
    plt.subplot(2, 3, 2)
    plt.plot(epochs, history_sgd.history[metric1], 'b', label='SGD')
    plt.plot(epochs, history_adam.history[metric1], 'g', label='Adam')
    plt.plot(epochs, history_adagrad.history[metric1], 'r', label='AdaGrad')
    plt.plot(epochs, history_rmsprop.history[metric1], 'm', label='RMSprop')
    plt.title(titles[1])
    plt.xlabel('Epochs')
    plt.ylabel(ylabels[1])
    plt.legend()

    # Графіку значень другої метрики
    plt.subplot(2, 3, 3)
    plt.plot(epochs, history_sgd.history[metric2], 'b', label='SGD')
    plt.plot(epochs, history_adam.history[metric2], 'g', label='Adam')
    plt.plot(epochs, history_adagrad.history[metric2], 'r', label='AdaGrad')
    plt.plot(epochs, history_rmsprop.history[metric2], 'm', label='RMSprop')
    plt.title(titles[2])
    plt.xlabel('Epochs')
    plt.ylabel(ylabels[2])
    plt.legend()

    # Графіку значень третьої метрики
    plt.subplot(2, 3, 4)
```

```
    plt.plot(epochs, history_sgd.history[metric3], 'b', label='SGD')
    plt.plot(epochs, history_adam.history[metric3], 'g', label='Adam')
    plt.plot(epochs, history_adagrad.history[metric3], 'r', label='AdaGrad')
    plt.plot(epochs, history_rmsprop.history[metric3], 'm', label='RMSprop')
    plt.title(titles[3])
    plt.xlabel('Epochs')
    plt.ylabel(ylabels[3])
    plt.legend()

    # Графіку значень четвертої метрики
    plt.subplot(2, 3, 5)
    plt.plot(epochs, history_sgd.history[metric4], 'b', label='SGD')
    plt.plot(epochs, history_adam.history[metric4], 'g', label='Adam')
    plt.plot(epochs, history_adagrad.history[metric4], 'r', label='AdaGrad')
    plt.plot(epochs, history_rmsprop.history[metric4], 'm', label='RMSprop')
    plt.title(titles[4])
    plt.xlabel('Epochs')
    plt.ylabel(ylabels[4])
    plt.legend()

    plt.tight_layout()
    plt.show()
```

Задача регресії з набором даних «Boston Housing»:

```
# Завантаження даних
(X_train, y_train), (X_test, y_test) = boston_housing.load_data()

# Нормалізація даних
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Виведення розмірів даних
print("Розмір тренувальних даних:", X_train.shape)
print("Розмір тестових даних:", X_test.shape)

# Створення моделей
model_sgd = tf.keras.Sequential([
    tf.keras.layers.Dense(64, activation='relu',
input_shape=(X_train.shape[1],)),
    tf.keras.layers.Dense(64, activation='relu'),
    tf.keras.layers.Dense(1)
])

model_adam = tf.keras.Sequential([
```

```
    tf.keras.layers.Dense(64, activation='relu',
input_shape=(X_train.shape[1],)),
    tf.keras.layers.Dense(64, activation='relu'),
    tf.keras.layers.Dense(1)
])

model_adagrad = tf.keras.Sequential([
    tf.keras.layers.Dense(64, activation='relu',
input_shape=(X_train.shape[1],)),
    tf.keras.layers.Dense(64, activation='relu'),
    tf.keras.layers.Dense(1)
])

model_rmsprop = tf.keras.Sequential([
    tf.keras.layers.Dense(64, activation='relu',
input_shape=(X_train.shape[1],)),
    tf.keras.layers.Dense(64, activation='relu'),
    tf.keras.layers.Dense(1)
])

# Компіляція моделей
model_sgd.compile(optimizer='sgd', loss='mean_squared_error',
metrics=['mae','mse', r2_score, explained_variance]) # tf.keras.optimizers.SGD()
model_adam.compile(optimizer='adam', loss='mean_squared_error',
metrics=['mae','mse', r2_score, explained_variance]) # tf.keras.optimizers.Adam()
model_adagrad.compile(optimizer='adagrad', loss='mean_squared_error',
metrics=['mae','mse', r2_score, explained_variance]) #
tf.keras.optimizers.Adagrad()
model_rmsprop.compile(optimizer='rmsprop', loss='mean_squared_error',
metrics=['mae','mse', r2_score, explained_variance]) #
tf.keras.optimizers.RMSprop()

# Навчання моделей
epochs = 150
start_time_sgd = time.time()
history_sgd = model_sgd.fit(X_train, y_train, epochs=epochs, batch_size=32,
validation_split=0.15, verbose=0)
end_time_sgd = time.time()
execution_time_sgd = end_time_sgd - start_time_sgd

start_time_adam = time.time()
history_adam = model_adam.fit(X_train, y_train, epochs=epochs, batch_size=32,
validation_split=0.15, verbose=0)
end_time_adam = time.time()
execution_time_adam = end_time_adam - start_time_adam

start_time_adagrad = time.time()
```

```
history_adagrad = model_adagrad.fit(X_train, y_train, epochs=epochs,
batch_size=32, validation_split=0.15, verbose=0)
end_time_adagrad = time.time()
execution_time_adagrad = end_time_adagrad - start_time_adagrad

start_time_rmsprop = time.time()
history_rmsprop = model_rmsprop.fit(X_train, y_train, epochs=epochs,
batch_size=32, validation_split=0.15, verbose=0)
end_time_rmsprop = time.time()
execution_time_rmsprop = end_time_rmsprop - start_time_rmsprop

print('Execution time (SGD): {:.2f} seconds'.format(execution_time_sgd))
print('Execution time (Adam): {:.2f} seconds'.format(execution_time_adam))
print('Execution time (AdaGrad): {:.2f} seconds'.format(execution_time_adagrad))
print('Execution time (RMSprop): {:.2f} seconds'.format(execution_time_rmsprop))

# Отримання остаточних значень втрат та метрик
loss_sgd, mae_sgd, mse_sgd, r2_sgd, ev_sgd = model_sgd.evaluate(X_test, y_test)
loss_adam, mae_adam, mse_adam, r2_adam, ev_adam = model_adam.evaluate(X_test,
y_test)
loss_adagrad, mae_adagrad, mse_adagrad, r2_adagrad, ev_adagrad =
model_adagrad.evaluate(X_test, y_test)
loss_rmsprop, mae_rmsprop, mse_rmsprop, r2_rmsprop, ev_rmsprop =
model_rmsprop.evaluate(X_test, y_test)

print("SGD - Loss: {:.4f}, MAE: {:.4f}, MSE: {:.4f}, R2 Score: {:.4f}, Explained
Variance: {:.4f}".format(loss_sgd, mae_sgd, mse_sgd, r2_sgd, ev_sgd))
print("Adam - Loss: {:.4f}, MAE: {:.4f}, MSE: {:.4f}, R2 Score: {:.4f}, Explained
Variance: {:.4f}".format(loss_adam, mae_adam, mse_adam, r2_adam, ev_adam))
print("AdaGrad - Loss: {:.4f}, MAE: {:.4f}, MSE: {:.4f}, R2 Score: {:.4f},
Explained Variance: {:.4f}".format(loss_adagrad, mae_adagrad, mse_adagrad,
r2_adagrad, ev_adagrad))
print("RMSprop - Loss: {:.4f}, MAE: {:.4f}, MSE: {:.4f}, R2 Score: {:.4f},
Explained Variance: {:.4f}".format(loss_rmsprop, mae_rmsprop, mse_rmsprop,
r2_rmsprop, ev_rmsprop))

# Графіки значень втрат та метрик з тренувалних даних
plot('loss', 'mse', 'mae', 'r2_score', 'explained_variance',
     titles=['Model Loss', 'Mean Squared Error', 'Mean Absolute Error', 'Model
R2-score', 'Model Explained Variance'],
     ylabels=['Loss', 'MSE', 'MAE', 'R2', 'EV'])

# Графіки значень втрат та метрик з валідаційних даних
plot('val_loss', 'val_mse', 'val_mae', 'val_r2_score', 'val_explained_variance',
     titles=['Validation Model Loss', 'Validation Mean Squared Error',
'Validation Mean Absolute Error', 'Validation Model R2-score', 'Validation Model
Explained Variance'],
     ylabels=['Loss', 'MSE', 'MAE', 'R2', 'EV'])
```

Задача класифікації з набором даних «IMDB Movie Reviews»:

```
# Завантаження даних
(X_train, y_train), (X_test, y_test) = imdb.load_data(num_words=10000)

# Передобробка даних
max_sequence_length = 60  # Максимальна довжина послідовності
X_train = pad_sequences(X_train, maxlen=max_sequence_length)
X_test = pad_sequences(X_test, maxlen=max_sequence_length)

# Виведення розмірів даних
print("Розмір тренувальних даних:", X_train.shape)
print("Розмір тестових даних:", X_test.shape)

# Створення моделей
model_sgd = tf.keras.Sequential([
    tf.keras.layers.Embedding(10000, 128, input_length=max_sequence_length),
    tf.keras.layers.LSTM(128),
    tf.keras.layers.Dense(1, activation='sigmoid')
])

model_adam = tf.keras.Sequential([
    tf.keras.layers.Embedding(10000, 128, input_length=max_sequence_length),
    tf.keras.layers.LSTM(128),
    tf.keras.layers.Dense(1, activation='sigmoid')
])

model_adagrad = tf.keras.Sequential([
    tf.keras.layers.Embedding(10000, 128, input_length=max_sequence_length),
    tf.keras.layers.LSTM(128),
    tf.keras.layers.Dense(1, activation='sigmoid')
])

model_rmsprop = tf.keras.Sequential([
    tf.keras.layers.Embedding(10000, 128, input_length=max_sequence_length),
    tf.keras.layers.LSTM(128),
    tf.keras.layers.Dense(1, activation='sigmoid')
])

# Компіляція моделей
model_sgd.compile(optimizer='sgd', loss='binary_crossentropy',
metrics=['accuracy', tf.keras.metrics.Precision(name='precision'),
tf.keras.metrics.Recall(name='recall'), f1_score])
```

```
model_adam.compile(optimizer='adam', loss='binary_crossentropy',
metrics=['accuracy', tf.keras.metrics.Precision(name='precision'),
tf.keras.metrics.Recall(name='recall'), f1_score])
model_adagrad.compile(optimizer='adagrad', loss='binary_crossentropy',
metrics=['accuracy', tf.keras.metrics.Precision(name='precision'),
tf.keras.metrics.Recall(name='recall'), f1_score])
model_rmsprop.compile(optimizer='rmsprop', loss='binary_crossentropy',
metrics=['accuracy', tf.keras.metrics.Precision(name='precision'),
tf.keras.metrics.Recall(name='recall'), f1_score])

# Навчання моделей
epochs = 10
start_time_sgd = time.time()
history_sgd = model_sgd.fit(X_train, y_train, epochs=epochs,
validation_split=0.2, verbose=1)
end_time_sgd = time.time()
execution_time_sgd = end_time_sgd - start_time_sgd

start_time_adam = time.time()
history_adam = model_adam.fit(X_train, y_train, epochs=epochs,
validation_split=0.2, verbose=1)
end_time_adam = time.time()
execution_time_adam = end_time_adam - start_time_adam

start_time_adagrad = time.time()
history_adagrad = model_adagrad.fit(X_train, y_train, epochs=epochs,
validation_split=0.2, verbose=1)
end_time_adagrad = time.time()
execution_time_adagrad = end_time_adagrad - start_time_adagrad

start_time_rmsprop = time.time()
history_rmsprop = model_rmsprop.fit(X_train, y_train, epochs=epochs,
validation_split=0.2, verbose=1)
end_time_rmsprop = time.time()
execution_time_rmsprop = end_time_rmsprop - start_time_rmsprop

print('Execution time (SGD): {:.2f} seconds'.format(execution_time_sgd))
print('Execution time (Adam): {:.2f} seconds'.format(execution_time_adam))
print('Execution time (AdaGrad): {:.2f} seconds'.format(execution_time_adagrad))
print('Execution time (RMSprop): {:.2f} seconds'.format(execution_time_rmsprop))

# Отримання остаточних значень втрат та метрик
loss_sgd, accuracy_sgd, precision_sgd, recall_sgd, f1_sgd =
model_sgd.evaluate(X_test, y_test)
loss_adam, accuracy_adam, precision_adam, recall_adam, f1_adam =
model_adam.evaluate(X_test, y_test)
loss_adagrad, accuracy_adagrad, precision_adagrad, recall_adagrad, f1_adagrad =
model_adagrad.evaluate(X_test, y_test)
```

```
loss_rmsprop, accuracy_rmsprop, precision_rmsprop, recall_rmsprop, f1_rmsprop =
model_rmsprop.evaluate(X_test, y_test)

print("SGD - Loss: {:.4f}, Accuracy: {:.4f}, Precision: {:.4f}, Recall: {:.4f},
F1 Score: {:.4f}".format(loss_sgd, accuracy_sgd, precision_sgd, recall_sgd,
f1_sgd))
print("Adam - Loss: {:.4f}, Accuracy: {:.4f}, Precision: {:.4f}, Recall: {:.4f},
F1 Score: {:.4f}".format(loss_adam, accuracy_adam, precision_adam, recall_adam,
f1_adam))
print("AdaGrad - Loss: {:.4f}, Accuracy: {:.4f}, Precision: {:.4f}, Recall:
{:.4f}, F1 Score: {:.4f}".format(loss_adagrad, accuracy_adagrad,
precision_adagrad, recall_adagrad, f1_adagrad))
print("RMSprop - Loss: {:.4f}, Accuracy: {:.4f}, Precision: {:.4f}, Recall:
{:.4f}, F1 Score: {:.4f}".format(loss_rmsprop, accuracy_rmsprop,
precision_rmsprop, recall_rmsprop, f1_rmsprop))

# Графіки значень втрат та метрик з тренувалних даних
plot(loss='loss', metric1='accuracy', metric2='precision', metric3='recall',
metric4='f1_score',
         titles=['Model Loss', 'Model Accuracy', 'Model Precision', 'Model
Recall', 'Model F1-score'],
         ylabels=['Loss', 'Accuracy', 'Precision', 'Recall', 'F1'])

# Графіки значень втрат та метрик з валідаційних даних
plot()
```

Задача класифікації з набором даних «CIFAR-10»:

```
# Завантаження даних
(X_train, y_train), (X_test, y_test) = cifar10.load_data()

# Передобробка даних
X_train = X_train / 255.0
X_test = X_test / 255.0

y_train = to_categorical(y_train, num_classes=10)
y_test = to_categorical(y_test, num_classes=10)

# Виведення розмірів даних
print("Розмір тренувальних даних:", X_train.shape)
print("Розмір тестових даних:", X_test.shape)

# Створення моделей
model_sgd = tf.keras.Sequential([
    tf.keras.layers.Conv2D(32, (3, 3), activation='relu', padding='same',
input_shape=(32, 32, 3)),
    tf.keras.layers.MaxPooling2D((2, 2)),
```

```
    tf.keras.layers.Conv2D(64, (3, 3), activation='relu', padding='same'),
    tf.keras.layers.MaxPooling2D((2, 2)),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(64, activation='relu'),
    tf.keras.layers.Dense(10, activation='softmax')
])

model_adam = tf.keras.Sequential([
    tf.keras.layers.Conv2D(32, (3, 3), activation='relu', padding='same',
input_shape=(32, 32, 3)),
    tf.keras.layers.MaxPooling2D((2, 2)),
    tf.keras.layers.Conv2D(64, (3, 3), activation='relu', padding='same'),
    tf.keras.layers.MaxPooling2D((2, 2)),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(64, activation='relu'),
    tf.keras.layers.Dense(10, activation='softmax')
])

model_adagrad = tf.keras.Sequential([
    tf.keras.layers.Conv2D(32, (3, 3), activation='relu', padding='same',
input_shape=(32, 32, 3)),
    tf.keras.layers.MaxPooling2D((2, 2)),
    tf.keras.layers.Conv2D(64, (3, 3), activation='relu', padding='same'),
    tf.keras.layers.MaxPooling2D((2, 2)),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(64, activation='relu'),
    tf.keras.layers.Dense(10, activation='softmax')
])

model_rmsprop = tf.keras.Sequential([
    tf.keras.layers.Conv2D(32, (3, 3), activation='relu', padding='same',
input_shape=(32, 32, 3)),
    tf.keras.layers.MaxPooling2D((2, 2)),
    tf.keras.layers.Conv2D(64, (3, 3), activation='relu', padding='same'),
    tf.keras.layers.MaxPooling2D((2, 2)),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(64, activation='relu'),
    tf.keras.layers.Dense(10, activation='softmax')
])

# Компіляція моделей
model_sgd.compile(optimizer='sgd', loss='categorical_crossentropy',
metrics=['accuracy', tf.keras.metrics.Precision(name='precision'),
tf.keras.metrics.Recall(name='recall'), f1_score])
model_adam.compile(optimizer='adam', loss='categorical_crossentropy',
metrics=['accuracy', tf.keras.metrics.Precision(name='precision'),
tf.keras.metrics.Recall(name='recall'), f1_score])
```

```
model_adagrad.compile(optimizer='adagrad', loss='categorical_crossentropy',
metrics=['accuracy', tf.keras.metrics.Precision(name='precision'),
tf.keras.metrics.Recall(name='recall'), f1_score])
model_rmsprop.compile(optimizer='rmsprop', loss='categorical_crossentropy',
metrics=['accuracy', tf.keras.metrics.Precision(name='precision'),
tf.keras.metrics.Recall(name='recall'), f1_score])

# Навчання моделей
epochs = 10
start_time_sgd = time.time()
history_sgd = model_sgd.fit(X_train, y_train, epochs=epochs,
validation_split=0.2, verbose=1)
end_time_sgd = time.time()
execution_time_sgd = end_time_sgd - start_time_sgd

start_time_adam = time.time()
history_adam = model_adam.fit(X_train, y_train, epochs=epochs,
validation_split=0.2, verbose=1)
end_time_adam = time.time()
execution_time_adam = end_time_adam - start_time_adam

start_time_adagrad = time.time()
history_adagrad = model_adagrad.fit(X_train, y_train, epochs=epochs,
validation_split=0.2, verbose=1)
end_time_adagrad = time.time()
execution_time_adagrad = end_time_adagrad - start_time_adagrad

start_time_rmsprop = time.time()
history_rmsprop = model_rmsprop.fit(X_train, y_train, epochs=epochs,
validation_split=0.2, verbose=1)
end_time_rmsprop = time.time()
execution_time_rmsprop = end_time_rmsprop - start_time_rmsprop

print('Execution time (SGD): {:.2f} seconds'.format(execution_time_sgd))
print('Execution time (Adam): {:.2f} seconds'.format(execution_time_adam))
print('Execution time (AdaGrad): {:.2f} seconds'.format(execution_time_adagrad))
print('Execution time (RMSprop): {:.2f} seconds'.format(execution_time_rmsprop))

# Отримання остаточних значень втрат та метрик
loss_sgd, accuracy_sgd, precision_sgd, recall_sgd, f1_sgd =
model_sgd.evaluate(X_test, y_test)
loss_adam, accuracy_adam, precision_adam, recall_adam, f1_adam =
model_adam.evaluate(X_test, y_test)
loss_adagrad, accuracy_adagrad, precision_adagrad, recall_adagrad, f1_adagrad =
model_adagrad.evaluate(X_test, y_test)
loss_rmsprop, accuracy_rmsprop, precision_rmsprop, recall_rmsprop, f1_rmsprop =
model_rmsprop.evaluate(X_test, y_test)
```

```
print("SGD - Loss: {:.4f}, Accuracy: {:.4f}, Precision: {:.4f}, Recall: {:.4f},
F1 Score: {:.4f}".format(loss_sgd, accuracy_sgd, precision_sgd, recall_sgd,
f1_sgd))
print("Adam - Loss: {:.4f}, Accuracy: {:.4f}, Precision: {:.4f}, Recall: {:.4f},
F1 Score: {:.4f}".format(loss_adam, accuracy_adam, precision_adam, recall_adam,
f1_adam))
print("AdaGrad - Loss: {:.4f}, Accuracy: {:.4f}, Precision: {:.4f}, Recall:
{:.4f}, F1 Score: {:.4f}".format(loss_adagrad, accuracy_adagrad,
precision_adagrad, recall_adagrad, f1_adagrad))
print("RMSprop - Loss: {:.4f}, Accuracy: {:.4f}, Precision: {:.4f}, Recall:
{:.4f}, F1 Score: {:.4f}".format(loss_rmsprop, accuracy_rmsprop,
precision_rmsprop, recall_rmsprop, f1_rmsprop))


# Графіки значень втрат та метрик з тренувалних даних
plot(loss='loss', metric1='accuracy', metric2='precision', metric3='recall',
metric4='f1_score',
         titles=['Model Loss', 'Model Accuracy', 'Model Precision', 'Model
Recall', 'Model F1-score'],
         ylabels=['Loss', 'Accuracy', 'Precision', 'Recall', 'F1'])

# Графіки значень втрат та метрик з валідаційних даних
plot()
```

Задача класифікації з набором даних «MNIST»:

```
# Завантаження набору даних MNIST
(X_train, y_train), (X_test, y_test) = mnist.load_data()

# Переведення значень пікселів у діапазон від 0 до 1
X_train = X_train / 255.0
X_test = X_test / 255.0

# Конвертація міток в one-hot вектори
y_train = tf.one_hot(y_train, 10)
y_test = tf.one_hot(y_test, 10)

# Виведення розмірів даних
print("Розмір тренувальних даних:", X_train.shape)
print("Розмір тестових даних:", X_test.shape)

# Побудова моделей
model_sgd = tf.keras.Sequential([
    tf.keras.layers.Flatten(input_shape=(28, 28)),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dense(10, activation='softmax')
```

```
])

model_adam = tf.keras.Sequential([
    tf.keras.layers.Flatten(input_shape=(28, 28)),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dense(10, activation='softmax')
])

model_adagrad = tf.keras.Sequential([
    tf.keras.layers.Flatten(input_shape=(28, 28)),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dense(10, activation='softmax')
])

model_rmsprop = tf.keras.Sequential([
    tf.keras.layers.Flatten(input_shape=(28, 28)),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dense(10, activation='softmax')
])

# Компіляція моделей
model_sgd.compile(optimizer='sgd', loss='categorical_crossentropy',
metrics=['accuracy', tf.keras.metrics.Precision(name='precision'),
tf.keras.metrics.Recall(name='recall'), f1_score])
model_adam.compile(optimizer='adam', loss='categorical_crossentropy',
metrics=['accuracy', tf.keras.metrics.Precision(name='precision'),
tf.keras.metrics.Recall(name='recall'), f1_score])
model_adagrad.compile(optimizer='adagrad', loss='categorical_crossentropy',
metrics=['accuracy', tf.keras.metrics.Precision(name='precision'),
tf.keras.metrics.Recall(name='recall'), f1_score])
model_rmsprop.compile(optimizer='rmsprop', loss='categorical_crossentropy',
metrics=['accuracy', tf.keras.metrics.Precision(name='precision'),
tf.keras.metrics.Recall(name='recall'), f1_score])

# Навчання моделей
epochs = 25
start_time_sgd = time.time()
history_sgd = model_sgd.fit(X_train, y_train, epochs=epochs,
validation_split=0.2, verbose=1)
end_time_sgd = time.time()
execution_time_sgd = end_time_sgd - start_time_sgd

start_time_adam = time.time()
history_adam = model_adam.fit(X_train, y_train, epochs=epochs,
validation_split=0.2, verbose=1)
end_time_adam = time.time()
execution_time_adam = end_time_adam - start_time_adam
```

```
start_time_adagrad = time.time()
history_adagrad = model_adagrad.fit(X_train, y_train, epochs=epochs,
validation_split=0.2, verbose=1)
end_time_adagrad = time.time()
execution_time_adagrad = end_time_adagrad - start_time_adagrad

start_time_rmsprop = time.time()
history_rmsprop = model_rmsprop.fit(X_train, y_train, epochs=epochs,
validation_split=0.2, verbose=1)
end_time_rmsprop = time.time()
execution_time_rmsprop = end_time_rmsprop - start_time_rmsprop

print('Execution time (SGD): {:.2f} seconds'.format(execution_time_sgd))
print('Execution time (Adam): {:.2f} seconds'.format(execution_time_adam))
print('Execution time (AdaGrad): {:.2f} seconds'.format(execution_time_adagrad))
print('Execution time (RMSprop): {:.2f} seconds'.format(execution_time_rmsprop))

# Отримання остаточних значень втрат та метрик
loss_sgd, accuracy_sgd, precision_sgd, recall_sgd, f1_sgd =
model_sgd.evaluate(X_test, y_test)
loss_adam, accuracy_adam, precision_adam, recall_adam, f1_adam =
model_adam.evaluate(X_test, y_test)
loss_adagrad, accuracy_adagrad, precision_adagrad, recall_adagrad, f1_adagrad =
model_adagrad.evaluate(X_test, y_test)
loss_rmsprop, accuracy_rmsprop, precision_rmsprop, recall_rmsprop, f1_rmsprop =
model_rmsprop.evaluate(X_test, y_test)

print("SGD - Loss: {:.4f}, Accuracy: {:.4f}, Precision: {:.4f}, Recall: {:.4f},
F1 Score: {:.4f}".format(loss_sgd, accuracy_sgd, precision_sgd, recall_sgd,
f1_sgd))
print("Adam - Loss: {:.4f}, Accuracy: {:.4f}, Precision: {:.4f}, Recall: {:.4f},
F1 Score: {:.4f}".format(loss_adam, accuracy_adam, precision_adam, recall_adam,
f1_adam))
print("AdaGrad - Loss: {:.4f}, Accuracy: {:.4f}, Precision: {:.4f}, Recall:
{:.4f}, F1 Score: {:.4f}".format(loss_adagrad, accuracy_adagrad,
precision_adagrad, recall_adagrad, f1_adagrad))
print("RMSprop - Loss: {:.4f}, Accuracy: {:.4f}, Precision: {:.4f}, Recall:
{:.4f}, F1 Score: {:.4f}".format(loss_rmsprop, accuracy_rmsprop,
precision_rmsprop, recall_rmsprop, f1_rmsprop))

# Графіки значень втрат та метрик з тренувалних даних
plot(loss='loss', metric1='accuracy', metric2='precision', metric3='recall',
metric4='f1_score',
         titles=['Model Loss', 'Model Accuracy', 'Model Precision', 'Model
Recall', 'Model F1-score'],
         ylabels=['Loss', 'Accuracy', 'Precision', 'Recall', 'F1'])

# Графіки значень втрат та метрик з валідаційних даних
```

```
plot()
```

Задача класифікації з набором даних «MNIST»:

```
# Завантаження даних
max_words = 1000
(X_train, y_train), (X_test, y_test) = reuters.load_data(num_words=max_words)

# Передобробка даних
tokenizer = Tokenizer(num_words=max_words)
X_train = tokenizer.sequences_to_matrix(X_train, mode='binary')
X_test = tokenizer.sequences_to_matrix(X_test, mode='binary')

# Визначення кількості класів
num_classes = max(y_train) + 1
y_train = to_categorical(y_train, num_classes)
y_test = to_categorical(y_test, num_classes)

# Виведення розмірів даних
print("Розмір тренувальних даних:", X_train.shape)
print("Розмір тестових даних:", X_test.shape)

# Створення моделей
model_sgd = tf.keras.Sequential([
    tf.keras.layers.Dense(512, input_shape=(max_words,), activation='relu'),
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Dense(256, activation='relu'),
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Dense(num_classes, activation='softmax')
])

model_adam = tf.keras.Sequential([
    tf.keras.layers.Dense(512, input_shape=(max_words,), activation='relu'),
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Dense(256, activation='relu'),
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Dense(num_classes, activation='softmax')
])

model_adagrad = tf.keras.Sequential([
    tf.keras.layers.Dense(512, input_shape=(max_words,), activation='relu'),
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Dense(256, activation='relu'),
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Dense(num_classes, activation='softmax')
])

model_rmsprop = tf.keras.Sequential([
```

```
    tf.keras.layers.Dense(512, input_shape=(max_words,), activation='relu'),
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Dense(256, activation='relu'),
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Dense(num_classes, activation='softmax')
])

# Компіляція моделей
model_sgd.compile(optimizer='sgd', loss='categorical_crossentropy',
metrics=['accuracy', tf.keras.metrics.Precision(name='precision'),
tf.keras.metrics.Recall(name='recall'), f1_score])
model_adam.compile(optimizer='adam', loss='categorical_crossentropy',
metrics=['accuracy', tf.keras.metrics.Precision(name='precision'),
tf.keras.metrics.Recall(name='recall'), f1_score])
model_adagrad.compile(optimizer='adagrad', loss='categorical_crossentropy',
metrics=['accuracy', tf.keras.metrics.Precision(name='precision'),
tf.keras.metrics.Recall(name='recall'), f1_score])
model_rmsprop.compile(optimizer='rmsprop', loss='categorical_crossentropy',
metrics=['accuracy', tf.keras.metrics.Precision(name='precision'),
tf.keras.metrics.Recall(name='recall'), f1_score])

# Навчання моделей
epochs = 20
batch_size = 32
start_time_sgd = time.time()
history_sgd = model_sgd.fit(X_train, y_train, batch_size=batch_size,
epochs=epochs, validation_split=0.2, verbose=1)
end_time_sgd = time.time()
execution_time_sgd = end_time_sgd - start_time_sgd

start_time_adam = time.time()
history_adam = model_adam.fit(X_train, y_train, batch_size=batch_size,
epochs=epochs, validation_split=0.2, verbose=1)
end_time_adam = time.time()
execution_time_adam = end_time_adam - start_time_adam

start_time_adagrad = time.time()
history_adagrad = model_adagrad.fit(X_train, y_train, batch_size=batch_size,
epochs=epochs, validation_split=0.2, verbose=1)
end_time_adagrad = time.time()
execution_time_adagrad = end_time_adagrad - start_time_adagrad

start_time_rmsprop = time.time()
history_rmsprop = model_rmsprop.fit(X_train, y_train, batch_size=batch_size,
epochs=epochs, validation_split=0.2, verbose=1)
end_time_rmsprop = time.time()
execution_time_rmsprop = end_time_rmsprop - start_time_rmsprop
```

```
print('Execution time (SGD): {:.2f} seconds'.format(execution_time_sgd))
print('Execution time (Adam): {:.2f} seconds'.format(execution_time_adam))
print('Execution time (AdaGrad): {:.2f} seconds'.format(execution_time_adagrad))
print('Execution time (RMSprop): {:.2f} seconds'.format(execution_time_rmsprop))

# Отримання остаточних значень втрат та метрик
loss_sgd, accuracy_sgd, precision_sgd, recall_sgd, f1_sgd =
model_sgd.evaluate(X_test, y_test)
loss_adam, accuracy_adam, precision_adam, recall_adam, f1_adam =
model_adam.evaluate(X_test, y_test)
loss_adagrad, accuracy_adagrad, precision_adagrad, recall_adagrad, f1_adagrad =
model_adagrad.evaluate(X_test, y_test)
loss_rmsprop, accuracy_rmsprop, precision_rmsprop, recall_rmsprop, f1_rmsprop =
model_rmsprop.evaluate(X_test, y_test)

print("SGD - Loss: {:.4f}, Accuracy: {:.4f}, Precision: {:.4f}, Recall: {:.4f},
F1 Score: {:.4f}".format(loss_sgd, accuracy_sgd, precision_sgd, recall_sgd,
f1_sgd))
print("Adam - Loss: {:.4f}, Accuracy: {:.4f}, Precision: {:.4f}, Recall: {:.4f},
F1 Score: {:.4f}".format(loss_adam, accuracy_adam, precision_adam, recall_adam,
f1_adam))
print("AdaGrad - Loss: {:.4f}, Accuracy: {:.4f}, Precision: {:.4f}, Recall:
{:.4f}, F1 Score: {:.4f}".format(loss_adagrad, accuracy_adagrad,
precision_adagrad, recall_adagrad, f1_adagrad))
print("RMSprop - Loss: {:.4f}, Accuracy: {:.4f}, Precision: {:.4f}, Recall:
{:.4f}, F1 Score: {:.4f}".format(loss_rmsprop, accuracy_rmsprop,
precision_rmsprop, recall_rmsprop, f1_rmsprop))

# Графіки значень втрат та метрик з тренувалних даних
plot(loss='loss', metric1='accuracy', metric2='precision', metric3='recall',
metric4='f1_score',
         titles=['Model Loss', 'Model Accuracy', 'Model Precision', 'Model
Recall', 'Model F1-score'],
         ylabels=['Loss', 'Accuracy', 'Precision', 'Recall', 'F1'])

# Графіки значень втрат та метрик з валідаційних даних
plot()
```