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

«___»_____2024 р.

Дипломна робота

на здобуття ступеня бакалавра

за освітньо-професійною програмою «Системний аналіз і управління»

спеціальності 124 «Системний аналіз»

на тему: «Створення фреймворку для машинного навчання моделей регресії»

Виконав:
студент IV курсу, групи КА-02 Тункін Євген Андрійович _____

Керівник:
доцент кафедри ММСА, к. т. н., доц. Савченко Ілля Олександрович _____

Консультант з економічного розділу:
професор кафедри ЕК , д. е. н., проф. Семенченко Наталія Віталіївна _____

Консультант з нормконтролю:
доцент кафедри ММСА, к. ф.-м. н., Статкевич Віталій Михайлович _____

Рецензент:
в. о. завідувачки кафедри ШІ, к. т. н., доц.,
Джигирей Ірина Миколаївна _____

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

Київ – 2024

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
«___»_____20__ р.

ЗАВДАННЯ
на дипломну роботу студенту
Тункіну Євгену Андрійовичу

1. Тема роботи «Створення фреймворку для машинного навчання моделей регресії», керівник роботи Савченко Ілля Олександрович, затверджені наказом по університету від 31.05.2024 р. №2240-с
2. Термін подання студентом роботи 13.06.2024
3. Вихідні дані до роботи: у роботі не використовуються вхідні дані.
4. Зміст роботи: дослідження предметної області, методи побудови фреймворку машинного навчання, реалізація фреймворку машинного навчання, функціонально-вартісний аналіз.
5. Перелік ілюстративного матеріалу (із зазначенням плакатів, презентацій тощо): таблиці та графіки.
6. Консультанти розділів роботи:

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Економічний	Семенченко Н. В., професор		

7. Дата видачі завдання 15.04.2024

Календарний план

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітка
1	Вибір теми і постановка дослідження	21.04.2024	Виконано
2	Збір та аналіз літератури	21.04.2024	Виконано
3	Формулювання задачі дослідження	28.04.2024	Виконано
4	Аналіз актуальності задачі дослідження	28.04.2024	Виконано
5	Розробка програмного продукту	05.05.2024	Виконано
6	Аналіз та впорядкування результатів	12.05.2024	Виконано
7	Оформлення пояснювальної записки	19.05.2024	Виконано
8	Оформлення дипломної роботи та аналіз отриманих результатів	19.05.2024	Виконано

Студент Євген ТУНКІН

Керівник Ілля САВЧЕНКО

РЕФЕРАТ

Дипломна робота: 72 с., 9 табл., 15 рис., 2 додатки, 10 джерел.

МАШИННЕ НАВЧАННЯ, РЕГРЕСІЯ, НЕЙРОННА МЕРЕЖА, АЛГОРИТМ ЗВОРОТНОГО ПОШИРЕННЯ ПОМИЛКИ, МЕТОД ГРАДІЄНТНОГО СПУСКУ, МЕТОД НАЙМЕНШИХ КВАДРАТІВ.

Об’єкт дослідження – задачі машинного навчання на побудову моделей регресії.

Предмет дослідження – алгоритми машинного навчання.

Мета роботи – дослідження різних методів машинного навчання для побудови регресійних нейронних моделей, а саме: алгоритм зворотного поширення похибки, чисельне знаходження похідних, метод градієнтного спуску, алгоритм оптимізації Adam та реалізація одного з них.

Під можливими модифікаціями обраного методу мається на увазі оптимізація розрахунків похідних функції втрат та використання додаткових методів оптимізації.

Програмний продукт було розроблено мовою програмування C#.

ABSTRACT

Thesis: 72 pp., 9 tables, 15 figures, 2 appendices, 10 references

MACHINE LEARNING, REGRESSION, NEURAL NETWORK, BACKPROPAGATION ALGORITHM, GRADIENT DESCENT, LEAST SQUARES METHOD.

The research object is machine learning tasks for building regression models.

The subject of the research is machine learning algorithms.

The purpose of the work is the study of various machine learning methods for building regression neural models, such as backpropagation algorithm, numerical methods of finding derivatives, gradient descent method, Adam optimization algorithm, and the implementation of one of them.

Possible modifications of the chosen method mean optimization of loss function derivative calculations and the use of additional optimization methods.

The software product was developed in the C# programming language.

ЗМІСТ

ВСТУП

У сучасному світі машинне навчання стає все більш важливим елементом розвитку технологій та інновацій. Ця галузь штучного інтелекту дозволяє комп'ютерним системам самостійно вчитися та приймати рішення на основі великих обсягів даних, що робить можливим автоматизацію складних процесів та створення нових продуктів і послуг. Від автоматизованих систем рекомендацій до самокерованих автомобілів, машинне навчання проникає у всі аспекти нашого життя, роблячи його більш зручним, ефективним і безпечним.

Актуальність машинного навчання обумовлена його здатністю вирішувати завдання, які раніше були недоступні для традиційних методів програмування. Завдяки машинному навчанню, комп'ютерні системи можуть обробляти великі обсяги даних, виявляти приховані закономірності та робити точні прогнози. Це має величезне значення для багатьох галузей, включаючи медицину, фінанси, виробництво, маркетинг та багато інших.

Однак, розвиток машинного навчання стикається з рядом викликів, серед яких складність створення та налаштування моделей, потреба в значних обчислювальних ресурсах та спеціалізованих знаннях. Для подолання цих викликів необхідно мати інструменти, які дозволяють спрощувати процес розробки та впровадження машинного навчання. У цьому контексті фреймворки машинного навчання набувають особливої важливості.

Фреймворки машинного навчання - це спеціалізовані програмні платформи, які надають набір інструментів і бібліотек для розробки, тренування, тестування та впровадження моделей машинного навчання. Вони спрощують процес створення алгоритмів, забезпечують зручність у роботі з даними та підвищують ефективність розробки. Наприклад, такі фреймворки як TensorFlow та PyTorch значно полегшують роботу розробників та дослідників, надаючи потужні засоби для реалізації складних моделей машинного навчання.

Застосування фреймворків машинного навчання є надзвичайно широким. Вони використовуються для розробки систем розпізнавання образів і мови, аналізу великих даних, прогнозування фінансових ризиків, оптимізації виробничих процесів, створення персоналізованих рекомендацій та багатьох інших завдань. Наприклад, в медицині фреймворки машинного навчання дозволяють створювати системи для автоматичного діагностування захворювань на основі медичних зображень, що може значно підвищити точність і швидкість діагностики.

Отже, створення фреймворку машинного навчання є важливим кроком у розвитку цієї галузі. Це дозволить спростити та прискорити процес розробки моделей, зробити машинне навчання доступнішим для широкого кола розробників та дослідників, а також сприяти подальшому впровадженню інновацій у різні сфери нашого життя. У даній дипломній роботі буде розглянуто процес створення фреймворку машинного навчання, його архітектуру та можливості, а також наведено приклади його застосування у різних галузях.

РОЗДІЛ 1 ДОСЛІДЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Машинне навчання як спосіб побудови алгоритмів

Машинне навчання – це галузь штучного інтелекту, яка вивчає методи побудови алгоритмів, які можуть самостійно вчитися і покращувати свою продуктивність на основі даних. Основна ідея полягає в тому, щоб використовувати великі обсяги даних для автоматичного визначення шаблонів, а потім використовувати ці знання для прийняття рішень без явного програмування. Машинне навчання має багато переваг у порівнянні з традиційним програмуванням.

1. Відсутність необхідності у чіткому визначенні правил: У традиційному програмуванні програміст самостійно визначає правила та кроки, за якими повинна виконуватися програма. У машинному навчанні модель сама навчається на основі вхідних даних, не потребуючи жорстко визначених правил.

2. Адаптивність до змін: Машинне навчання дозволяє моделям адаптуватися до нових даних та змінюватися з часом без необхідності в ручному перепрограмуванні. При використанні традиційного програмування зміни у навколишньому середовищі потребують змін у коді.

3. Можливість автоматизовувати вирішення складних задач: Машинне навчання дозволяє вирішувати широкий клас задач, від виявлення об’єктів на фото до прогнозування фінансових ризиків. Без використання машинного навчання створити алгоритм, який вирішує ці задачі було б надскладним завданням через велику різноманітність вхідних даних.

Однак, машинне навчання має декілька недоліків.

1. Залежність від даних**:** У машинному навчанні використовуються великі обсяги даних для навчання моделей. Ці дані дозволяють моделям виявляти складні відносини та залежності між вхідними та вихідними даними. У

традиційному програмуванні така залежність визначається заздалегідь програмістом.

2. Потреба у великих обчислювальних ресурсах: Для того щоб натренувати модель, що вирішує певну проблему потрібне дороге та сучасне обладнання, найчастіше GPU (graphic processing unit), CPU (central processing unit), або ще більш сучасні TPU (tensor processing unit), створені спеціально для роботи зі штучним інтелектом. Навіть після того, як модель натренована, використовувати її можливо не на кожному пристрої, через все ще велику потребу в обчислювальних потужностях. Найчастіше ці проблеми виникають при використанні нейронних мереж у смартфонах або ще менших системах, таких як камери спостереження. Ця проблема створила нову галузь машинного навчання – Embedded machine learning.

1.2 Розвиток машинного навчання

Розвиток машинного навчання відбувався протягом багатьох десятиліть і зазнав значного прогресу завдяки поєднанню наукових досліджень, технологічних проривів та зростання обсягів доступних даних.

Перші спроби створення систем штучного інтелекту і машинного навчання, таких як логістична регресія та нейронні мережі, відбувалися у 1950-1970-х роках. Однак обмежені обчислювальні ресурси та обсяги даних обмежували їхню ефективність.

У 1980-1990-х роках машинне навчання пережило період спаду, коли було зафіксовано обмеження та відмови від досліджень через складнощі у навчанні моделей, відсутність доступних даних та низьку ефективність алгоритмів.

З появою Інтернету і збільшенням обсягів цифрових даних машинне навчання у 2000-х роках знову здобуло популярність. Методи, такі як глибоке

навчання та алгоритми з учителем, розвинулися і стали більш ефективними завдяки збільшенню обсягів даних та зростанню обчислювальної потужності.

Наступною віхою розвитку машинного навчання стала епоха глибокого навчання, яка почалась у 2010-х роках. Поява глибоких нейронних мереж дозволила створювати складніші моделі, які можуть здійснювати зручне відображення високорівневих абстракцій у великих обсягах даних.

Глибоке навчання виявилося особливо ефективним у розв'язанні завдань в областях комп'ютерного зору, обробки природної мови та інших сферах. З появою бібліотек та інструментів, таких як TensorFlow, PyTorch та AutoML, використання машинного навчання стало доступнішим і зручнішим. Автоматизовані методи дозволяють швидше розробляти та оптимізувати моделі, зменшуючи необхідність у глибоких знаннях математики та програмування.

Сьогодні машинне навчання продовжує швидко розвиватися. Нові методи і моделі, такі як трансформери та генеративні моделі, відкривають нові можливості у різних галузях, включаючи мовний аналіз, синтез та розпізнавання, а також у сферах здоров'я, автономних систем та фінансів. Збільшення обсягів даних, розвиток алгоритмів та зростання обчислювальної потужності є основними факторами, що сприяють подальшому розвитку машинного навчання.

1.3 Необхідність наявності фреймворку машинного навчання

Використання бібліотек та інструментів машинного навчання має безліч переваг, які значно полегшують розробку, навчання та впровадження моделей. Наведемо найбільш суттєві з них.

1. Зменшення часу розробки: Бібліотеки машинного навчання, такі як TensorFlow, PyTorch, та Scikit-Learn, надають готові реалізації алгоритмів та моделей, що значно зменшує час, потрібний для розробки нових програм.

2. Зменшення вимог до знання математики: При використанні бібліотек машинного навчання, зменшується необхідність у знанні користувачем математичного аналізу та математичної статистики, оскільки більшість алгоритмів вже є реалізованими. Таким чином, програмісту достатньо розібратися з програмним інтерфейсом, який надає бібліотека, і він не має необхідності розуміти, як працюють алгоритми машинного навчання усередині.

3. Легка інтеграція з іншими інструментами: Бібліотеки машинного навчання часто мають зручний інтерфейс та API, що дозволяє легко інтегрувати їх з іншими інструментами та середовищами програмування.

4. Автоматизоване налаштування параметрів: Деякі інструменти, такі як AutoML, надають автоматизовані методи для підбору оптимальних параметрів моделі, що зменшує необхідність ручного налаштування та оптимізації.

5. Широкий вибір готових моделей і алгоритмів: Бібліотеки машинного навчання зазвичай містять великий асортимент готових моделей та алгоритмів, що дозволяє швидко реалізувати рішення для різноманітних завдань.

6. Підтримка різних мов програмування: Більшість бібліотек машинного навчання підтримують різні мови програмування, що дає можливість вибору найзручнішої мови для конкретного проекту чи задачі.

7. Велика спільнота користувачів і підтримка: Багаті спільноти користувачів та розробників бібліотек машинного навчання забезпечують активну підтримку, навчальні матеріали, та допомогу вирішенню проблем.

1.4 Існуючі фреймворки для машинного навчання

На цей момент існує три відомих фреймворки машинного навчання, які часто використовуються на реальних проектах.

1.4.1 Scikit-Learn

Найпростіший з трьох фреймворків, Scikit-Learn, також відомий як sklearn, написаний мовою Python з використанням бібліотеки NumPy для розв'язання задач лінійної алгебри та операцій з масивами. Деякі алгоритми написані мовою Cython (розширений варіант мови Python, націлений на спрощення інтеграції з кодом на мові C) для покращення продуктивності. Перша версія бібліотеки була написана Девідом Корнапе влітку 2007 року в рамках програми Google Summer of Code.

Цікавий той факт, що пізніше цього ж року Метью Брюхер почав працювати над нею, як частиною своєї дипломної роботи. Scikit-Learn добре інтегрується з багатьма бібліотеками Python, такими як Matplotlib та plotly для побудови графіків, NumPy для масивів, Pandas, SciPy та багатьма іншими. Основним недоліком цієї бібліотеки є підтримка тільки однієї мови програмування – Python.

1.4.2 TensorFlow

Безкоштовна бібліотека, випущена компанією Google у 2015 році як продовження закритого проекту DistBelief (аналогічної бібліотеки, але для використання всередині компанії). У 2019 році Google випустив TensorFlow 2.0, який мав покращений метод автоматичного диференціювання.

Ця бібліотека є більш низькорівневою ніж Scikit-Learn та дозволяє переносити обчислення на відеокарту (або на декілька відеокарт) при тренуванні моделей для задач комп'ютерного зору. Підтримує багато мов програмування, серед яких Python, C++, JavaScript, C# та інші, також підтримується всіма найпопулярнішими платформами, такими як 64-х бітні Linux, macOS, Windows, Android та iOS.

1.4.3 PyTorch

Є найбільш складним, низькорівневим та популярним фреймворком машинного навчання. Розробляється переважно компанією Facebook. Підтримується на мовах Python та C++, при чому зовнішній інтерфейс для Python є біль відшліфованим та головним напрямком розробки. При тренуванні розподілених моделей великого масштабу, PyTorch програє у продуктивності та масштабованості TensorFlow. При цьому PyTorch є кращим при розробці передових або незвичайних рішень через його більшу гнучкість.

1.5 Постановка задачі машинного навчання

Нехай спостерігається деяка випадкова величина η, яка залежить від іншої випадкової величини ξ (або вектору $\vec{\xi}$), значення яких теж спостерігаються.

Будемо вважати, що η, яку називають вихідною величиною або відкликом залежить лише від однієї випадкової величини ξ, яку називають вхідною або фактором чи предиктором. Припустимо, що було проведено n експериментів зі значеннями фактору ξ $x_1, x_2 \dots, x_n$ та відповідними значенням відклику $y_1, y_2 \dots, y_n$.

Тепер оберемо $f(x)$, яка і буде моделлю. Необхідно за значеннями $x_1, x_2 \dots, x_n$ та $y_1, y_2 \dots, y_n$ оцінити $f(x)$. Спочатку треба визначити вигляд цієї функції. Часто це випливає з постановки задачі або з'ясовується після візуалізації отриманих даних. Наприклад, якщо обрано вигляд

$$f(x) = \sum_{k=1}^{m} \beta_k x_k,$$

де m – кількість змінних у векторі спостереження, то таку модель називають лінійною регресією. Якщо

$$f(x) = \sum_{k=1}^{m} \sum_{i=1}^{p} \beta_{ik} x_k,$$

де m – кількість змінних у векторі спостереження;

p – довільно вибране число, більше за одиницю, таку модель називають поліноміальною регресією,

Нейронною мережею є модель, де вхідний вектор множиться на матрицю, після чого до результату застосовується певна зростаюча нелінійна функція (її називають функцією активації), і цей крок повторюється декілька разів.

Тоді кількістю шарів нейронної мережі буде ця кількість повторів, а кількість прихованих шарів – число на один менше, оскільки останній шар є вихідним. Кількістю нейронів на певному шарі називається кількість стовпців відповідної до нього матриці. Наведемо приклад нейронної мережі:

$$f(x) = ReLU\left(\begin{pmatrix} x^{(1)} & x^{(2)} & x^{(3)} \end{pmatrix} * \begin{pmatrix} w_{11}^{(1)} & w_{12}^{(1)} \\ w_{21}^{(1)} & w_{22}^{(1)} \\ w_{31}^{(1)} & w_{32}^{(1)} \end{pmatrix} \right) * \begin{pmatrix} w_{11}^{(2)} & w_{12}^{(2)} \end{pmatrix}$$

На цьому прикладі зображено нейронну мережу з двома шарами (одним прихованим). Перший з них має два нейрони, а другий – один, оскільки на виході потрібне одне число. У якості функції активації обрано $ReLU$ (rectified linear unit), яка визначається наступним чином:

$$ReLU(x) = \begin{cases} 0, & x \leq 0 \\ \alpha x, & x > 0 \end{cases}$$

Наведемо ще декілька популярних функцій активації:

$$sigmoid(x) = \sigma(x) = \frac{e^x}{1+e^x} - \text{ сігмоїд;}$$

$$tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} - \text{гіперболічний тангенс;}$$

$$lrelu(x) = \max(ax, x) - leaky\ ReLU,$$

де а – маленьке позитивне число, наприклад 0,01.

Отже, задачею машинного навчання є знайти оптимальні параметри для функції-моделі, які б забезпечили найменшу помилку при використанні моделі для знаходження значення y при відомому $\vec{x}$. Процес ітеративного покращення параметрів моделі і називається навчанням моделі.

1.6 Методи, які використовуються фреймворками для машинного навчання

Для навчання моделі вводиться поняття функції втрат – функції від отриманих оцінок y (вони позначаються як $\hat{y}$) та справжніх значень цільової змінної y спостереженнях. Ці функції втрат визначають, наскільки гарно модель відображає залежність між вхідними та цільовою змінними. Чим менше значення функції втрат, тим краще працює модель. Наведемо декілька прикладів функцій втрат:

$\frac{1}{n}\sum_{i=1}^{n}(\hat{y} - y)^2$ – MSE (mean squared error), середньоквадратична похибка;

$\frac{1}{n}\sum_{i=1}^{n}|\hat{y} - y|$ – MAE (mean average error), середня похибка;

$\frac{1}{n}\sum_{i=1}^{n}\frac{|\hat{y}-y|}{y}$ – MAPE (mean average percentage error), середня відсоткова похибка.

При навчанні моделі застосовуються різні способи, але всі вони мають на меті одне й те саме – мінімізувати функцію втрат. Для цієї задачі використовуються ітераційні методи оптимізації першого порядку, які оновлюють значення параметрів моделі у напрямку, який зменшує значення функції втрат (наприклад, метод градієнтного спуску або його варіації).

Найбільша проблема полягає у тому, що ці методи потребують значення похідних, а функція, яка визначає модель є дуже складною. Для цього придумано декілька алгоритмів, які спрощують розрахунок похідних.

1.6.1 Алгоритм зворотного поширення похибки

Метод зворотного розповсюдження помилки є найбільш фундаментальною складовою нейронної мережі. Вперше він був описаний у 1960-ті і майже через 30 років його популяризували Румельхарт, Хінтон і Вільямс у статті під назвою «Learning representations by back-propagating errors».

Цей алгоритм є найбільш розповсюдженим у фреймворках машинного навчання через свою швидкодію та універсальність. Метод використовується для ефективного навчання нейронної мережі за допомогою так званого ланцюгового правила (правила диференціювання складної функції).

Простіше кажучи, після кожного проходу по мережі зворотне поширення виконує прохід у зворотний бік та регулює параметри моделі (ваги та зміщення). Основна ідея полягає в тому, щоб "поширити" помилку з кінця мережі до початку, визначити, які ваги спричиняють більшу помилку, і відповідно оновити їх, щоб зменшити цю помилку.

Основні етапи алгоритму зворотного поширення помилки:

Forward Propagation: На цьому етапі вхідні дані пропускаються через мережу, і кожен шар обчислює свій вихід. Результати вихідного шару порівнюються з очікуваними виходами, і обчислюється величина помилки.

Backward Propagation: Після того як відомо значення помилки, вона поширюється назад через мережу. Це робиться шляхом обчислення похідних функції втрат по відношенню до параметрів мережі (ваги і зсуви) за допомогою правила ланцюжків.

Розглянемо процес навчання моделі за допомогою алгоритму зворотного поширення похибки на прикладі простої чотиришарової нейронної мережі.

Чотиришарова нейронна мережа складається з чотирьох нейронів вхідного шару, чотирьох нейронів на прихованих шарах та 1 нейрона на вихідному шарі (рис. 1.1).

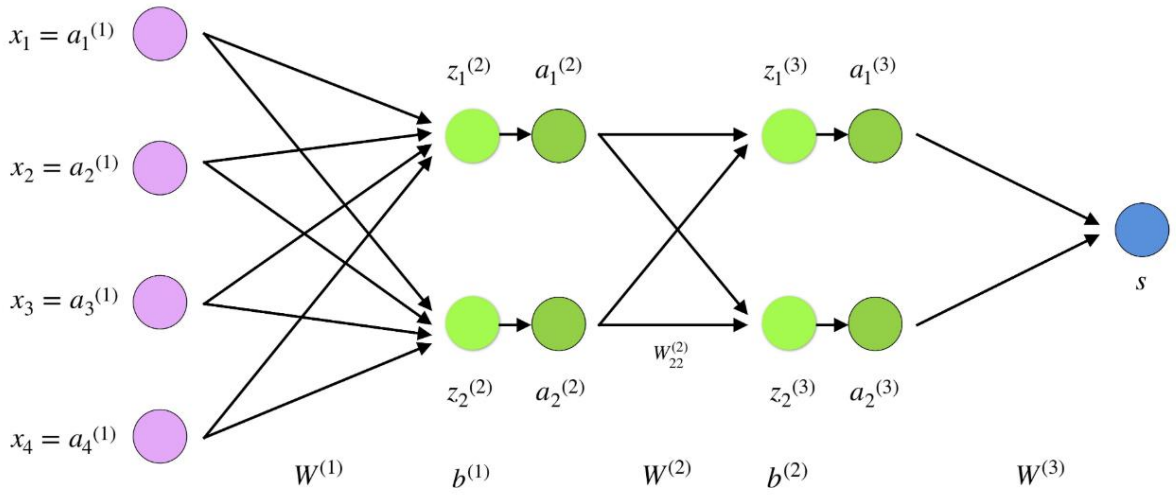


Рисунок 1.1 – Приклад нейронної мережі

На малюнку нейрони фіолетового кольору є вхідними даними. Вони є простими скалярними величинами.

$$x_i = a_i^{(1)}, i \in 1,2,3,4$$

Перший набір активацій (a) дорівнює вхідним значенням.

Кінцеві значення у прихованих нейронах (на малюнку зеленого кольору) обчислюються з використанням z' – зважених входів у шарі 1 та a' активацій у шарі L. Для шарів 2 та 3 рівняння будуть наступними:

Для $l = 2$:

$$z^{(2)} = W^{(1)}x + b^{(1)}$$
$$a^{(2)} = f\left(z^{(2)}\right)$$

Для $l = 3$:

$$z^{(3)} = W^{(2)}a^{(2)} + b^{(2)}$$
$$a^{(2)} = f\left(z^{(3)}\right)$$

$W^{(1)}$ та $W^{(2)}$ – це вати на шарах 2 та 3, а b^2 та b^3–зміщення на цих шарах.

Остання частина нейронної мережі – це вихідний шар, який видає прогнозоване значення. У нашому простому прикладі він представлений у вигляді одного нейрона, забарвленого в синій колір і розраховується так:

$$s = W^{(3)}a^{(3)}$$

Forward Propagation:

$a^{(1)} = x$ – вхідний шар.

$z^{(2)} = W^{(1)}x + b^{(1)}$ – значення нейрону на першому прихованому шарі.

$a^{(2)} = f\left(z^{(2)}\right)$ – значення активації на першому прихованому шарі.

$z^{(3)} = W^{(2)}x + b^{(2)}$ – значення нейрону на другому прихованому шарі.

$a^{(3)} = f\left(z^{(3)}\right)$ – значення активації на другому прихованому шарі.

$s = W^{(3)}a^{(3)}$ – вихідний шар.

Наступним кроком є знаходження значення функції втрат, позначимо її у цьому прикладі через $L = loss(s, y)$. Ґрунтуючись на значенні L, модель знає, наскільки потрібно скоригувати її параметри, щоб наблизитися до очікуваного вихідного значення y. Це відбувається за допомогою методу зворотного розповсюдження помилки.

Backward Propagation:

Зворотне поширення спрямоване на мінімізацію функції втрат шляхом коригування ваги та зміщень мережі. Ступінь коригування визначається градієнтами функції втрат стосовно цих параметрів. Похідна функції L відбиває чутливість до зміни значення функції (вихідного значення) щодо зміни її аргументу x. Градієнт показує, наскільки необхідно змінити параметр x (у позитивну чи негативну сторону), щоб мінімізувати L.

Обчислення цих градієнтів відбувається з допомогою методу, званого ланцюговим правилом.

Для однієї ваги w_{jk}^{l} градієнт дорівнює:

$$\frac{\partial L}{\partial w_{jk}^{l}} = \frac{\partial L}{\partial z_j^{l}} \frac{\partial z_j^{l}}{\partial w_{jk}^{l}} - \text{за правилом диференціювання композиції функцій,}$$

$$z_j^{l} = \sum_{k=1}^{m} \partial w_{jk}^{l} a_k^{l-1} + b_j^{l} \ - \text{за визначенням нейронної мережі,}$$

$m -$ кількість нейронів на шарі з номером $l - 1$, тоді

$$\frac{\partial z_j^{l}}{\partial w_{jk}^{l}} = a_k^{l-1} - \text{диференціювання лінійної функції, отже}$$

$$\frac{\partial L}{\partial w_{jk}^{l}} = \frac{\partial L}{\partial z_j^{l}} a_k^{l-1}$$

Спільна частина в обох рівняннях часто називається «локальним градієнтом» і виражається так:

$$\delta_j^l = \frac{\partial L}{\partial z_j^l}$$

Локальний градієнт можна легко знайти за допомогою диференціювання композиції функцій. Після того, як градієнт знайдений відбуваються кроки, описані раніше у загальному підході до навчання нейронних мереж. Ними є оптимізація параметрів моделі одним з методів оптимізації та перевірка критерію зупинки.

Тепер розглянемо приклад знаходження градієнту відносно ваги w_{22}^2.

$$\frac{\partial L}{\partial w_{22}^{(2)}} = \frac{\partial L}{\partial z_2^{(3)}} \frac{\partial z_2^{(3)}}{\partial w_{22}^{(2)}} = \frac{\partial L}{\partial a_2^{(3)}} \frac{\partial a_2^{(3)}}{\partial z_2^{(3)}} a_2^{(2)} = \frac{\partial L(\vec{s}, \vec{y})}{\partial a_2^{(3)}} f'\left(z_2^{(3)}\right) a_2^{(2)}$$

Тепер потрібно обрати конкретну функцію втрат. В цьому прикладі розглянемо середньоквадратичну помилку, яка визначається наступним чином:

$$L(\vec{s}, \vec{y}) = \sum_{i=1}^{n} (s_i - y_i)^2$$

Тоді

$$s = W^{(3)} a^{(3)} = w_{11}^{(3)} a_1^{(3)} + w_{12}^{(3)} a_2^{(3)}$$

$$\frac{\partial L(\vec{s}, \vec{y})}{\partial a_2^{(3)}} = \sum_{i=1}^{n} \frac{\partial (s_i - y_i)^2}{\partial a_2^{(3)}} = \sum_{i=1}^{n} 2(s_i - y_i) w_{12}^{(3)} = w_{12}^{(3)} \sum_{i=1}^{n} 2(s_i - y_i)$$

отже,

$$\frac{\partial L}{\partial w_{22}^{(2)}} = w_{12}^{(3)} \sum_{i=1}^{n} 2(s_i - y_i) f'\left(z_2^{(3)}\right) a_2^{(2)}$$

Для знаходження $f'\left(z_2^{(3)}\right)$ потрібно обрати конкретну функцію активації. У якості прикладу візьмемо сігмоїд, який визначається наступним чином:

$$sigmoid(z) = \sigma(z) = \frac{e^z}{1 + e^z},$$

тоді

$$f'\left(z_2^{(3)}\right) = \frac{d}{dz_2^{(3)}} \sigma\left(z_2^{(3)}\right) = \sigma\left(z_2^{(3)}\right)\left(1 - \sigma\left(z_2^{(3)}\right)\right)$$

отже, фінальне значення:

$$\frac{\partial L}{\partial w_{22}^{(2)}} = w_{12}^{(3)} \sum_{i=1}^{n} 2(s_i - y_i) \sigma\left(z_2^{(3)}\right)\left(1 - \sigma\left(z_2^{(3)}\right)\right) a_2^{(2)}$$

У цьому прикладі було розглянуто знаходження градієнту по параметру, який знаходиться на передостанньому шарі нейронної мережі. Якщо взяти шар, який знаходиться ще далі від вихідного, то розрахунки та фінальне значення стануть ще більш складними.

Тим не менш, оскільки нейронна мережа має схожу структуру на кожному з шарів, існують рекурсивні формули для знаходження градієнтів на будь-якому шарі. Розглянемо ці формули без детального доведення:

$$\delta_i^{(L)} = \left(y_i - a_i^{(L)}\left(x_i^{(L)}\right)\right) a_i^{(L)}\left(x_i^{(L)}\right)\left(1 - a_i^{(L)}\left(x_i^{(L)}\right)\right);$$

$$\delta_i^{(l)} = \sum_{k=1}^{n} \delta_i^{(l+1)} w_{ki}^{(l+1)} \frac{\partial a_i^{(l)}}{\partial z_i^{(l)}};$$

$$\frac{\partial L}{\partial w_{ij}^{(l)}} = \frac{1}{n} \delta_i^{(l)} \frac{\partial x_i^{(l)}}{\partial w_{ij}^{(l)}}.$$

У цій роботі ми не будемо реалізовувати алгоритм зворотного поширення похибки через його складність. Натомість, використаємо наступний алгоритм, який знаходить похідні чисельно.

1.6.2 Чисельне знаходження похідних

Цей метод є набагато більш простим, ніж попередній. Для того, щоб знайти похідну функції втрат по певному параметру, його значення спочатку збільшується на невелике число ϵ, розраховується нове значення функції втрат (позначимо його J^+), потім значення цього параметру зменшується на таке саме ϵ, розраховується нове значення функції втрат (позначимо його J^-).

Тоді приблизне значення похідної по конкретному параметру в точці буде рівним

$$\frac{\partial L}{\partial w_{ij}^{(l)}} = \frac{J^+ - J^-}{2\epsilon}$$

Така дія виконується для кожного параметру моделі, і після того, як всі похідні знайдено, виконується одночасне оновлення всіх параметрів моделі за допомогою обраного методу оптимізації.

1.7 Висновки до розділу 1

Машинне навчання є потужним інструментом для створення алгоритмів, які можуть аналізувати дані і робити прогнози без явного програмування. Цей підхід має велике значення у багатьох сферах, включаючи медицину, фінанси, технології та багато інших.

З розвитком обчислювальної потужності та збільшенням обсягу даних, машинне навчання швидко розвивається, виводячи на передові практики та техніки для вирішення різноманітних задач.

Наявність підтримки фреймворку машинного навчання стає ключовою у сучасній дослідницькій та індустріальній практиці, оскільки вони надають зручний і потужний інтерфейс для створення, навчання та оцінки моделей машинного навчання.

Аналіз фреймворків, таких як Scikit-Learn, TensorFlow та PyTorch, показує їхню широку функціональність та ефективність вирішення різноманітних завдань машинного навчання.

Ключовим етапом у процесі розв'язання задачі машинного навчання є чітке визначення цілей та параметрів моделі, а також вибір відповідного підходу для розв'язання конкретної задачі.

РОЗДІЛ 2 МЕТОДИ ПОБУДОВИ ФРЕЙМВОРККУ МАШИННОГО НАВЧАННЯ

2.1 Знаходження похідних

Для знаходження похідних будемо використовувати чисельний метод через простоту його реалізації. Цей метод не потребує додаткової адаптації під поставлену задачу створення фреймворку, але слід пояснити більш детально, як він буде використовуватися.

Під час навчання моделі на кожній ітерації створюється тривимірній масив дійсних чисел, у якому будуть зберігатися значення похідних. Позначимо цей масив D. Тоді $d_{j,k}^{(i)}$ позначає похідну по$w_{j,k}^{(i)}$, де

i – номер шару нейронної мережі;

j – номер стовпця матриці шару i;

k – номер рядка матриці шару i.

Відповідно, через W позначимо тривимірний масив параметрів нейронної мережі. Також введемо позначення:

l – кількість шарів у нейронній мережі;

p_i – кількість нейронів на шарі з номером i;

n – кількість спостережень у наборі даних;

$J(\bar{\hat{y}}, \bar{y})$ – функція втрат від отриманих за допомогою моделі значень залежної змінної та справжніх значень цієї змінної;

m – кількість змінних у векторі спостереження.

Тепер наведемо алгоритм заповнення масиву D значеннями похідних у точці W. Присвоювання значення змінній будемо позначати через «:=».

Повторюємо для всіх можливих комбінацій i, j, k, таких, що $1 \leq i \leq l, 1 \leq j \leq p_i, 1 \leq k \leq p_{i-1}$. Значення p_0 беремо рівним m.

Збільшуємо $w_{j,k}^{(i)}$ на невелике число ϵ.

$$w_{j,k}^{(i)} \coloneqq w_{j,k}^{(i)} + \epsilon$$

Знаходимо значення $\overline{\widehat{y^+}}$. Для цього просто підставляємо значення x у модель:

$$\overline{\widehat{y^+}} = f(x)$$

Далі знаходимо функцію втрат для збільшеного параметру:

$$J^+ \coloneqq J\left(\overline{\widehat{y^+}}, y\right)$$

Робимо значення $w_{j,k}^{(i)}$ на ϵ менше, ніж початкове:

$$w_{j,k}^{(i)} \coloneqq w_{j,k}^{(i)} - 2\epsilon$$

Знаходимо значення $\overline{\widehat{y^-}}$. Для цього також підставляємо значення x у модель:

$$\overline{\widehat{y^-}} = f(x)$$

Далі знаходимо функцію втрат для зменшеного параметру:

$$J^- \coloneqq J\left(\overline{\widehat{y^-}}, y\right)$$

Чисельно знаходимо приблизне значення похідної по цьому параметру та записуємо його в масив похідних:

$$d_{j,k}^{(i)} \coloneqq \frac{J^+ - J^-}{2\epsilon}$$

Повертаємо значення параметру до початкового:

$$w_{j,k}^{(i)} \coloneqq w_{j,k}^{(i)} + \epsilon$$

2.2 Оптимізація параметрів моделі

Після того, як похідні знайдено, необхідно оновити параметри моделі, використовуючи їх. Існує багато методів, які виконують цю задачу. У цій роботі реалізовано декілька найпопулярніших.

2.2.1 Метод градієнтного спуску

Цей метод полягає у тому, щоб перемістити значення параметрів моделі у напрямку антиградієнту (найшвидшого спадання функції). Формально, це виглядає наступним чином: для всіх можливих комбінацій i, j, k, таких, що $1 \leq i \leq l, 1 \leq j \leq p_i, 1 \leq k \leq p_{i-1}$:

$$w_{j,k}^{(i)} \coloneqq w_{j,k}^{(i)} - \alpha d_{j,k}^{(i)}$$

Параметр α називається швидкість навчання (learning rate). Чим більше його значення, тим швидше відбувається навчання моделі, але якщо воно занадто велике, то модель може «перестрибувати» точки мінімуму функції втрат, внаслідок чого тільки збільшувати її.

Це означає, що дослідник або інженер машинного навчання самостійно обирає оптимальне значення цього параметру, відповідно у створеному фреймворку необхідно реалізувати цю можливість.

2.2.2 Оптимізатор Adam

Adam Optimizer є найбільш популярним алгоритмом при навчанні глибоких нейронних мереж та складних моделей. Цей алгоритм утворений об'єднанням алгоритмів Momentum (оптимізатор з імпульсом) та RMSProp (Root Mean Square Propagation).

Головною відмінністю цього алгоритму від базового градієнтного спуску є те, що швидкість навчання є різною для кожного параметра моделі. При цьому швидкості навчання оновлюються на кожній ітерації на основі змін похідних по цьому параметру (так званих моментів).

Розглянемо детальний алгоритм оновлення параметрів моделі:

Спочатку потрібно задати значення параметрам алгоритму, а саме: початкову швидкість навчання (α) та гіперпараметри β_1, β_2 та ϵ.

Далі оновлюється внутрішній стан оптимізатора. Тут похідні будемо позначити за допомогою g.

Оновлюються оцінки першого моменту:

$$m = \beta_1 m + (1 - \beta_1) g$$

Оновлюються оцінки другого моменту (масив похідних множиться поелементно:

$$v = \beta_2 v + (1 - \beta_2)(g * g)$$

Оновлюється зміщення для оцінок першого та другого моменту:

$$\hat{m} = \frac{m}{1 - \beta_1}$$

$$\hat{v} = \frac{v}{1 - \beta_2}$$

Оновлюється значення швидкості навчання для кожного параметра:

$$\alpha = \frac{\alpha\sqrt{1 - \beta_2}}{1 - \beta_1}$$

Оновлюються параметри моделі:

$$\theta = \theta - \frac{\alpha\hat{m}}{\sqrt{\hat{v}} + \epsilon}$$

Тут ділення матриць відбувається поелементно, а початковими значеннями для всіх параметрів вважаються нульові матриці необхідного розміру.

2.3 Системний дизайн фреймворку машинного навчання

Тут під фразою системний дизайн мається на увазі дизайн класів мови програмування, їхній взаємозв’язок та взаємодія. Це важливе питання, оскільки фреймворк необхідно створити таким чином, щоб було легко розширювати його функціонал, при цьому не змінюючи вже написаний код.

Іншими словами, системний дизайн повинен задовольняти так званий Open Closed Principle, суть якого в тому, що класи повинні бути відкритими для розширення, але закритими для модифікації.

У сучасних мовах програмування це досягається використанням абстрактних класів та класів-інтерфейсів. Нижче описано головні класи у фреймворку, які забезпечують відповідність наведеному принципу та іншим принципам SOLID.

1. DataSet – клас зберігає спостереження незалежної та залежної змінної у зручній для обробки формі.

2. IFileDataReader – інтерфейс, який визначає методи для читання даних з файлів різного формату, які створюють датасет. У фреймворку планується реалізувати CsvDataReader для читання з файлів з розширенням csv та ExcelDataAdapter для читання файлів з розширенням xlsx.

3. ActivationFunction – абстрактний клас для визначення функцій активації, планується реалізувати RELU та Sigmoid.

4. IOptimizator – інтерфейс, призначений для реалізації різних методів оптимізації. Має метод, який приймає похідні та поточні параметри, і оптимізує їх. Планується реалізувати метод градієнтного спуску да декілька його варіацій.

5. Model – клас, який позначає створену нейронну мережу із заданими кількістю шарів, їх розміром та методом оптимізації. Має методи для отримання оцінок залежних змінних на заданих значеннях незалежних змінних та навчанні моделі на заданому датасеті.

2.3 Огляд вхідних даних для перевірки роботи фреймворку

Для тестування роботи створеного фреймворку потрібні набори даних, на яких можна поставити задачу навчити регресійну модель. Отримані результати буде розглянуто у розділі 3.

2.3.1 Штучно згенерований датасет

Згенеруємо набір даних з двома змінними. Нехай перша з них є незалежною, і її значення є рівномірно розподіленим на певному проміжку. Нехай друга змінна є лінійно залежною від першої, але із певним «шумом». Більш формально

$$y = \alpha x + \beta + \epsilon,$$

де y – залежна змінна;

x – незалежна змінна;

α – заданий коефіцієнт;

β – задане число;

ϵ – випадкова величина, розподілена нормально з нульовим середнім.

Згенерований датасет зображено на рис. 2.1.

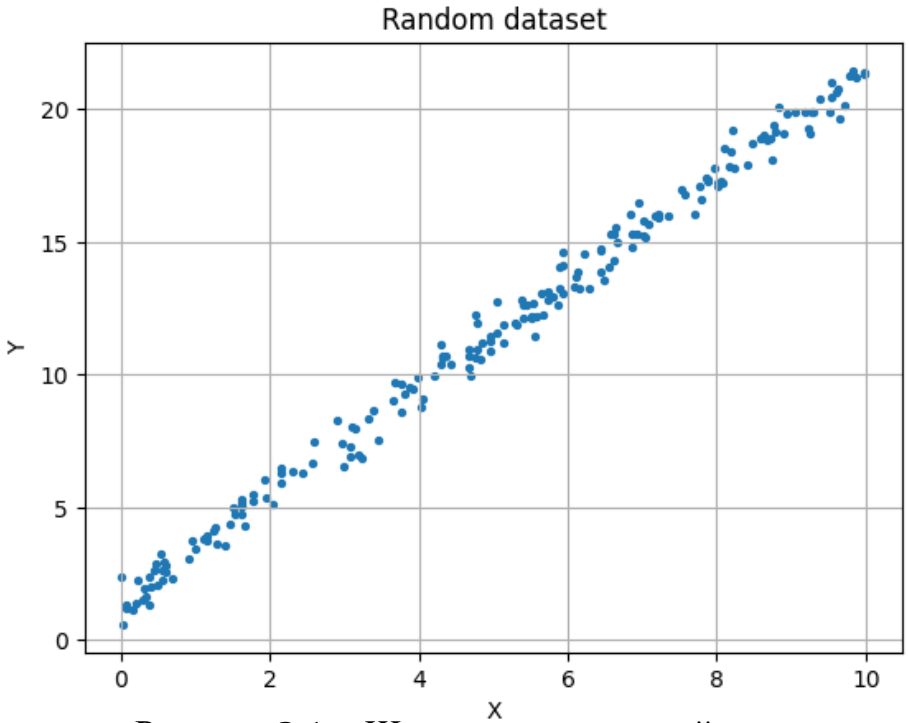


Рисунок 2.1 – Штучно згенерований датасет

Для цього набору даних візьмемо модель вигляду $f(x) = ax + b$. Ця модель є лінійною регресією, частковим випадком нейронної мережі, а саме мережі з одним шаром та одним нейроном у ньому. Очікуваним результатом є те, що параметр a моделі буде приблизно рівним значенню $\alpha = 2$, а параметр b – значенню $\beta = 1{,}5$.

2.3.2 Реальний датасет

У якості реальних даних біло обрано датасет пов'язаний з врожайністю певної ділянки землі та погодних умов, за яких ця врожайність спостерігалась. Задача полягає в тому, щоби знайти спосіб передбачити, об'єм врожаю знаючи, які будуть погодні умови.

Розв'язок цієї задачі допоможе у плануванні продажів та управлінні ризиками компанії, яка займається виробленням агропродукції.

Датасет містить 8 змінних:

1. nitrogen – коефіцієнт вмісту нітрогену у ґрунті.
2. phosphorus – коефіцієнт вмісту фосфору у ґрунті.
3. potassium – коефіцієнт вмісту калію у ґрунті.
4. temperature – температура в градусах Цельсія.
5. humidity – відносна вологість повітря.
6. pH_Value – pH значення у ґрунті.
7. rainfall – кількість опадів у міліметрах.
8. yield – цільова змінна, кількість кілограмів зібраного врожаю.

Декілька записів з датасету наведено на рис. 2.2.

	Nitrogen	Phosphorus	Potassium	Temperature	Humidity	pH_Value	Rainfall	Crop	Yield
0	90	42	43	20.879744	82.002744	6.502985	202.935536	Rice	7000
1	85	58	41	21.770462	80.319644	7.038096	226.655537	Rice	5000
2	60	55	44	23.004459	82.320763	7.840207	263.964248	Rice	7000
3	74	35	40	26.491096	80.158363	6.980401	242.864034	Rice	7000
4	78	42	42	20.130175	81.604873	7.628473	262.717340	Rice	120000

Рисунок 2.2 – Фрагмент набору даних

2.4 Висновки до розділу 2

Для навчання моделі необхідно знайти похідну функції втрат по кожному з її параметрів. Для цього можна використовувати чисельні методи, але вони є занадто повільними при навчанні великих моделей через те що при знаходженні кожної похідної необхідно двічі запускати розрахунок оцінок залежної змінної (виконувати forward propagation) та знаходити значення функції втрат.

Після знаходження похідних для самої оптимізації параметрів можна застосовувати різні методи, такі як градієнтний спуск або його варіації, які зберігають значення похідних на попередній ітерації, змінюють швидкість навчання тощо.

Необхідно реалізувати можливість користувачу самостійно обирати параметри моделі, які не є тренованими, наприклад кількість шарів, їх розмір, швидкість навчання чи тип оптимізатора.

Головним викликом під час створення фреймворку машинного навчання буде не реалізація математичних алгоритмів, а написання чистого коду, який дозволить легко розширювати існуючий функціонал фреймворку без змін у вже написаному коді.

Роботу фреймворка краще спочатку перевірити на штучному датасеті, розподіл змінних якого відомий, а потім натренувати модель для вирішення справжньої задачі зі сфери науки про данні.

РОЗДІЛ 3.РЕЛІЗАЦІЯ ФРЕЙМВОРКУ МАШИННОГО НАВЧАННЯ

3.1 Вибір технології

Для реалізації фреймворку машинного навчання було обрано мову C# на платформі .Net. Для створення продукту також використовувалась інтегрована середа розробки Visual Studio 2022. Оскільки при створенні фреймворку більшість операцій є суто математичними, у роботі не використовуються сторонні бібліотеки.

Вибір мови програмування пояснюється в першу чергу можливістю використовувати розвинене об’єктно-орієнтоване програмування у C#. При реалізації фреймворку стають в нагоді абстрактні класи та інтерфейси, що присутні не у всіх мовах програмування. Наприклад, при використанні ООП для додавання нового оптимізатора достатньо написати реалізацію відповідного інтерфейсу, а не створювати клас з нуля.

Окремою перевагою мови C# є її строга типізованість, що дозволяє зменшувати кількість помилок під час виконання програми, отримуючи та виправляючи їх на етапі компіляції.

Швидкодія мови C# знаходиться на високому рівні, хоч і не є найкращою серед всіх сучасних мов програмування. Більш важливим є той факт, що при розв’язанні задач машинного навчання спеціалісти віддають перевагу простоті написання коду та зручному прикладному інтерфейсу програмування (API) (прикладом може бути величезна популярність мови Python порівняно з C++). В цьому аспекті C# переважає низькорівневі більш швидкі мови, такі як C++.

3.2 Реалізація основних алгоритмів

Як вже було вказано в розділі 1, основними етапами при навчанні моделі є знаходження похідних та оптимізація параметрів моделі на основі знайдених значень. Ці дві дії виконує публічний метод Fit в класі NeuralNetwork (рис. 3.1).

```
public void Fit(double[,] x, double[,] y, int epochCount = 5)
{
    int writeLossEpochsInterval = epochCount / 10;
    for (int i = 0; i < epochCount; i++)
    {
        var derivativaes = GetDerivativesNumerical(x, y);
        Optimizator.Optimize(Weights, derivativaes);
        var loss = Loss(x, y);
        if (i % writeLossEpochsInterval == 0)
        {
            Console.WriteLine($"Loss after epoch {i}: {loss}");
        }
    }
}
```

Рисунок 3.1 – Реалізація методу Fit

Метод Fit приймає кількість ітерацій навчання у якості параметра. Значення функції втрат виводиться через кожну десяту частину від всього процесу навчання.

Знаходження похідних відбувається за описаним у розділі 2 алгоритмом чисельного знаходження похідних. На рис. 3.2 представлено вихідний код методу в класі NeuralNetwork, який знаходить похідні по кожному параметру моделі.

Цей метод, в свою чергу, викликає метод Loss, в який знаходить значення функції втрат на заданих даних.

Після того, як метод GetDerivativesNumerical повертає знайдені значення похідних, вони разом із посиланням на поточні параметри моделі передаються в метод Optimize оптимізатора моделі, якому було задано значення в момент її створення. У якості оптимізатора можуть використовуватись різні методи, оскільки член класу моделі Optimizer оголошено як інтерфейс.

```
private double[][,] GetDerivativesNumericall(double[,] x, double[,] y, double epsilon = 1e-10)
{
    var derivatives = new double[Weights.Length][,];
    for (int i = 0; i < Weights.Length; i++)
    {
        derivatives[i] = new double[Weights[i].GetLength(0), Weights[i].GetLength(1)];
        for (int j = 0; j < Weights[i].GetLength(0); j++)
        {
            for (int k = 0; k < Weights[i].GetLength(1); k++)
            {
                Weights[i][j, k] += epsilon;
                var lossPlus = Loss(x, y);
                Weights[i][j, k] -= 2 * epsilon;
                var lossMinus = Loss(x, y);
                Weights[i][j, k] += epsilon;
                derivatives[i][j, k] = (lossPlus - lossMinus) / (2 * epsilon);
            }
        }
    }

    return derivatives;
}
```

Рисунок 3.2 – Реалізація методу знаходження похідних

3.2 Розв’язок прикладних задач за допомогою фреймворку

3.2.1 Найпростіший набір даних

Для тестування, чи правильно відбувається знаходження похідних та оптимізація параметрів моделі, створимо найпростіший датасет одразу в C#.

Нехай у цьому датасеті закономірність буде такою, що залежна змінна є в два рази більша за незалежну.

Нехай незалежна змінна має наступні значення:

$$x = \begin{pmatrix} 1 \\ 2 \\ 3 \\ 4 \end{pmatrix}$$

Тоді значення залежної змінної є такими:

$$y = \begin{pmatrix} 2 \\ 4 \\ 6 \\ 8 \end{pmatrix}$$

Створимо нейронну мережу без прихованих шарів (таким чином, вона є звичайною лінійною регресією (рис. 3.3)).

```
var model = new NeuralNetwork([1, 1])
{
    Activation = new Relu(),
    LossFunction = new MeanSquaredErrorLoss(),
    Optimizator = new GradientDescentOptimizator() { LearningRate = 0.025 },
};
```

Рисунок 3.3 – Приклад створення нейронної мережі

Натренуємо її на цьому наборі даних (рис. 3.4).

```
double[,] x = {
    { 1, 2, 3, 4 },
};
double[,] y =
{
    { 2, 4, 6, 8 },
};
model.Fit(x, y, 10);
var p = model.Predict(x);
Console.WriteLine($"model weights: a = {model.Weights[0][0, 1]}, b = {model.Weights[0][0, 0]}");
Console.WriteLine($"predicted: {p[0, 0]} {p[0, 1]} {p[0, 2]} {p[0, 3]}");
Console.WriteLine($"real: {y[0, 0]} {y[0, 1]} {y[0, 2]} {y[0, 3]}");
```

Рисунок 3.4 – Приклад тренування моделі

Отриманий результат зображено на рис. 3.5.

```
Loss after epoch 0: 6.976418952804311
Loss after epoch 1: 2.378920750703433
Loss after epoch 2: 0.8188652404099532
Loss after epoch 3: 0.28938151016394803
Loss after epoch 4: 0.10956152365222256
Loss after epoch 5: 0.048381006694523235
Loss after epoch 6: 0.02745601215608607
Loss after epoch 7: 0.02019174502283894
Loss after epoch 8: 0.017564860320564587
Loss after epoch 9: 0.01651380503625011
model weights: a = 2.0989289141080674, b = -0.31274508311616667
predicted: 1.7861838309919007 3.885112745099968 5.984041659208035 8.082970573316103
real: 2 4 6 8
```

Рисунок 3.5 – Отриманий при тренуванні результат

Як бачимо, отримані оцінки незалежної змінної після тренування моделі є близькими до реальних значень незалежної змінної. Оцінки та реальні

значення не співпадають повністю, оскільки ітераційні методи оптимізації не знаходять локальний мінімум функції точно, а лише наближаються до нього.

За способом створення датасету, реальними параметрами лінійної функції є 2 та 0. За отриманими результатами, параметрами моделі є 2,1 та -0,31. Ці значення є близькими, але не збігаються з тієї ж самої причини.

3.2.2 Штучно згенерований набір даних

Наступним етапом перевірки працездатності створеного фреймворку машинного навчання є набір даних з випадковим шумом. Детально цей датасет був описаний у розділі 2.

Для читання даних використовуються створений клас (рис. 3.6).

```
var dataReader = new CsvFileDataReader("data1.csv", [1]);
(double[,] x, double[,] y) = dataReader.Read();
```

Рисунок 3.6 – Читання даних з файлу

Для розв’язку цієї задачі також скористаємось лінійною регресією. Процес створення моделі тут є таким самим, туму не будемо наводити його. Отримані результати зображено на рис. 3.7.

Тут для перевірки близькості прогнозів та реальних значень обрано чотири випадкові записи з набору даних. Як бачимо, вони є дуже близькими.

```
Loss after epoch 0: 78.22701187918547
Loss after epoch 10: 1.111074457760586
Loss after epoch 20: 0.8615889719432112
Loss after epoch 30: 0.7256882194071372
Loss after epoch 40: 0.6202660945964089
Loss after epoch 50: 0.5384439482130873
Loss after epoch 60: 0.47493867277311497
Loss after epoch 70: 0.42564964838426605
Loss after epoch 80: 0.3873944925547638
Loss after epoch 90: 0.35770317882538216
model weights: a = 2.0828098834056443, b = 0.9479089856028711
predicted: 12.7061282954802 13.74011260263082 20.250163245786716 11.66575077588998
real: 13.064620385203543 13.895333851800705 19.882643110452214 11.85933599835237
```

Рисунок 3.7 – Отриманий при тренуванні результат

3.2.3 Реальний набір даних

У якості прикладної задачі розглянемо побудову моделі регресії на датасеті пов’язаному з врожайністю певної ділянки землі. Детально цей датасет описано у розділі 2.

Для розв’язання цієї задачі використаємо декілька різних варіантів архітектур нейронних мереж, від найпростішої лінійної регресії до багатошарової нейронної мережі. Варіанти архітектур наведені у табл. 3.1.

Таблиця 3.1 – Варіанти архітектури нейронної мережі

Номер нейронної мережі	Розміри прихованих шарів
1	-
2	[16]
3	[16, 16]

Також випробуємо різні функції активації. Отримані значення функції втрат після навчання моделі на 1000 ітераціях для різних варіантів моделі наведено у табл. 3.2.

Таблиця 3.2 – Отримані значення функції втрат

Архітектура НМ/Функція активації	Relu	LeakyRelu	Sigmoid	Гіперболічний тангенс
1	0,88	0,88	0,88	0,88
2	0,167	0,3	0,592	0,115
3	0,21	0,038	0,749	0,011

Як бачимо, для першого варіанта архітектури значення функції втрат є однаковим для всіх функцій активації. Це пояснюється тим, що модель у цьому випадку є лінійною, і тому вона не може пристосуватись до розподілу реальних даних. Після того, як модель досягає найкращого значення функції активації з можливих, її подальше навчання не приносить покращення значення функції втрат.

Для всіх варіантів архітектури нейронної мережі, гіперболічний тангенс виявляється найкращою функцією активації. Це не дивно, оскільки ця функція активації дійсно забезпечує найкраще навчання моделі та використовується найчастіше у сучасному машинному навчанні.

Найгіршою функцією активації є Сігмоїд, це також підтверджується світовими практиками, оскільки Сігмоїд є найпершою винайденою функцією активації, але зараз знайдені більш кращі аналоги.

Також слід зазначити, що значення функції активації зменшується при ускладненні архітектури нейронної мережі. Це також більш ніж логічно, оскільки чим більше параметрів має модель, тим легше їй пристосуватись до розподілу наданих даних.

Найкращою моделлю є нейронна мережа з двома прихованими шарами розміром по 16 нейронів кожний (третій варіант архітектури) та гіперболічним тангенсом у якості функції активації. На рис. 3.8 наведено лістинг фреймворку під час навчання цієї моделі. Повний лістинг під час навчання всіх варіантів моделей наведено у додатку А.

```
Running architecture 3 with activation TanH
Loss after epoch 0: 2.792033814112329
Loss after epoch 100: 0.36339241600917177
Loss after epoch 200: 0.20204841364931125
Loss after epoch 300: 0.11703434002562338
Loss after epoch 400: 0.07036468842890167
Loss after epoch 500: 0.04553034373676408
Loss after epoch 600: 0.031109638710393572
Loss after epoch 700: 0.021773838738216637
Loss after epoch 800: 0.015513894974357617
Loss after epoch 900: 0.011320678592251875
predicted: 6913.4900026043815 4676.981209348185 327.6164675486657 519.5153220402053
real: 7000 4600 350 700
```

Рисунок 3.8 – Отриманий результат при тренуванні найкращої моделі

Як бачимо, отримані прогнози є надзвичайно близькими до реальних значень врожайності.

3.4 Висновки до розділу 3

У розділі обґрунтовано вибір мови програмування для реалізації фреймворку машинного навчання, розглянуто недоліки та переваги обраної мови C#.

Також у розділі 3 наведено найголовніші приклади коду для реалізації навчання моделі, яке складається зі знаходження похідних функції втрат та оптимізації параметрів. Крім цього, у розділі наведені приклади використання фреймворку, такі як створення моделей, завантаження даних, тренування моделей та отримання прогнозів.

Найголовнішою частиною розділу є вирішення практичних задач за допомогою фреймворку. Тут розглянуто три задачі, складність яких поступово збільшується. Для перших двох залежність між змінними відома, тому можливо оцінити відповідність отриманих оцінок параметрів залежності до реальних. В обох випадках вони є близькими

Третьою задачею є реальне завдання для спеціаліста з науки про дані, пов’язане з передбаченням врожайності маючи інформацію про погодні умови. Для вирішення цієї задачі було натреновано 12 різних моделей та серед них обрано найкращу.

РОЗДІЛ 4 ФУНКЦІОНАЛЬНО-ВАРТІСНИЙ АНАЛІЗ ПРОГРАМНОГО ПРОДУКТУ

В цьому розділі була дана оцінка основним характеристикам створюваного програмного продукту, що є фреймворком машинного навчання. Цей продукт сприятиме роботі спеціалістів з науки про дані та інженерів машинного навчання, допомагаючи підготовлювати дані та тренувати моделі машинного навчання.

У даному дослідженні розглянуто вісім різних варіантів реалізації програмного продукту для вибору найбільш оптимальної стратегії, яка забезпечить конкурентоспроможність продукту та його високу якість, а також, за можливості зменшить витрати на реалізацію та впровадження. Для проведення аналізу та вибору найкращої стратегії використовується апарат функціонально вартісного аналізу.

Функціонально-вартісний аналіз (ФВА) являє собою методику, що дозволяє визначити фактичну вартість продукту або послуги, не потребуючи інформації про структуру організації компанії. Мета ФВА – виявити можливості для зменшення витрат шляхом застосування більш ефективних виробничих процесів і досягнення кращого співвідношення витратами на створення продукту та його цінністю для споживачів. Для проведення аналізу використовуються економічні, технічні та конструкційні дані.

Алгоритм ФВА полягає у визначенні конкретних етапів розробки продукту, розрахунку необхідної кількості робочих годин та об’єму річних витрат, а також ідентифікації джерел витрат. В кінці ФВА проводиться розрахунок остаточної вартості програмного продукту.

4.1 Постановка задачі проектування

У даній роботі ФВА застосовується для проведення техніко-економічного аналізу розробки фреймворку машинного навчання, тому необхідно провести аналіз функцій програмного продукту, призначеного для підготовки даних та тренування нейронних мереж.

Серед технічних вимог до продукту можна виділити наступні:

1. Зручність прикладного інтерфейсу програмування (API).
2. Швидкість обробки даних.
3. Можливість додавання нової функціональності без змін вже написаного коду.
4. Низькі витрати на впровадження фреймворку.

4.2 Обґрунтування функцій програмного продукту

Головною функцією F_0 – є розробка програмного продукту, який дозволяє тренувати моделі регресії на великих обсягах даних. На основі цієї функції, можна виділити наступні:

1. F_1 – мова програмування.
2. F_2 – вибір алгоритму оптимізації параметрів моделі.
3. F_3 – вибір способу розповсюдження програмного продукту.

Кожну з цих функцій можна реалізувати у декілька варіантів:

Функція F_1:

а) Python;

б) C#.

Функція F_2.

а) чисельне знаходження похідних;

б) алгоритм зворотного поширення помилки.

Функція F_3:

а) публікація продукту у вигляді стандартної бібліотеки;

б) публікація вихідного коду на GitHub.

Варіанти імплементації головних функцій показані у морфологічній карті системи (рис. 4.1).

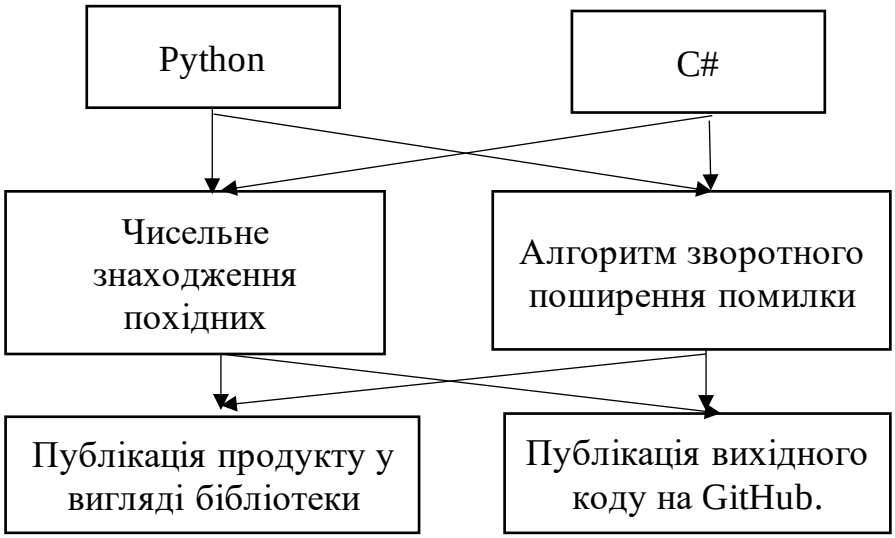


Рисунок 4.1 – морфологічна карта системи

З морфологічної карти отримуємо множину можливих варіантів головних функцій. Позитивно-негативна матриця наведена в табл. 4.2.

Таблиця 4.2 – Позитивно-негативна матриця

Функції	Варіант реалізації	Переваги	Недоліки
F_1	*A*	Низький поріг входу	Погана продуктивність
	B	Строга типізація, компільованість, доволі висока продуктивність, крос-платформеність	Необхідність ліцензії на інтегровану середу розробки
F_2	*A*	Простота реалізації	Велика обчислювальна складність
	B	Низька обчислювальна складність	Неймовірно висока складність реалізації
F_3	*A*	Простота публікації	Залежність від екосистеми мови програмування
	B	Спільнота користувачів може допомагати покращувати фреймворк та виправляти помилки	Необхідність робити вихідний код відкритим, що може призвести до неліцензованого використання програмного продукту.

Провівши аналіз позитивно-негативної матриці можна зробити висновок, що деякі варіанти реалізації функцій не відповідають задачам, які повинен виконувати програмний продукт. Ці варіанти реалізації можна одразу відкинуті.

F_1:

Програма допускає використання будь-якого варіанту.

F_2:

Перевагу надаємо простоті реалізації для більш швидкого створення фреймворку із меншими витратами. Варіант В відкидаємо.

F_3:

Перевагу надаємо простоті публікації для більш швидкого створення фреймворку із меншими витратами. Варіант В відкидаємо.

Таким чином, маємо наступні варіанти реалізації ПП:

$$F_1 A - F_2 A - F_3 A$$
$$F_1 \mathrm{B} - F_2 A - F_3 A$$

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

4.3 Обґрунтування системи параметрів програмного продукту

Визначимо основні параметри для розрахунку коефіцієнта технічного рівня, використовуючи розглянуті вище функції.

Для характеризування програмного продукту, використаємо такі параметри:

1. *X1* – продуктивність мови програмування.
2. *X2* – точність розв’язку.
3. *X3* – популярність обраного ресурсу серед спільноти користувачів.

Гірші, середні і кращі значення визначаються, беручи до уваги вимоги замовника й умов експлуатації програмного продукту. Обрані значення параметрів наведено у табл. 4.3.

Таблиця 4.3 – Основні параметри програмного продукту

Ім’я параметра	Позначення	Одиниці виміру	Значення параметра		
			гірші	середні	кращі
Продуктивність мови програмування	X1	операцій/мс	60	80	110
Точність розв’язку	X2	Частка одиниці	10E-4	10E-5	10E-6
Популярність ресурсу	X3	Млн завантажень	10	50	100

За даними табл. 4.2 будуються графічні характеристики параметрів (рис. 4.2 – 4.4).

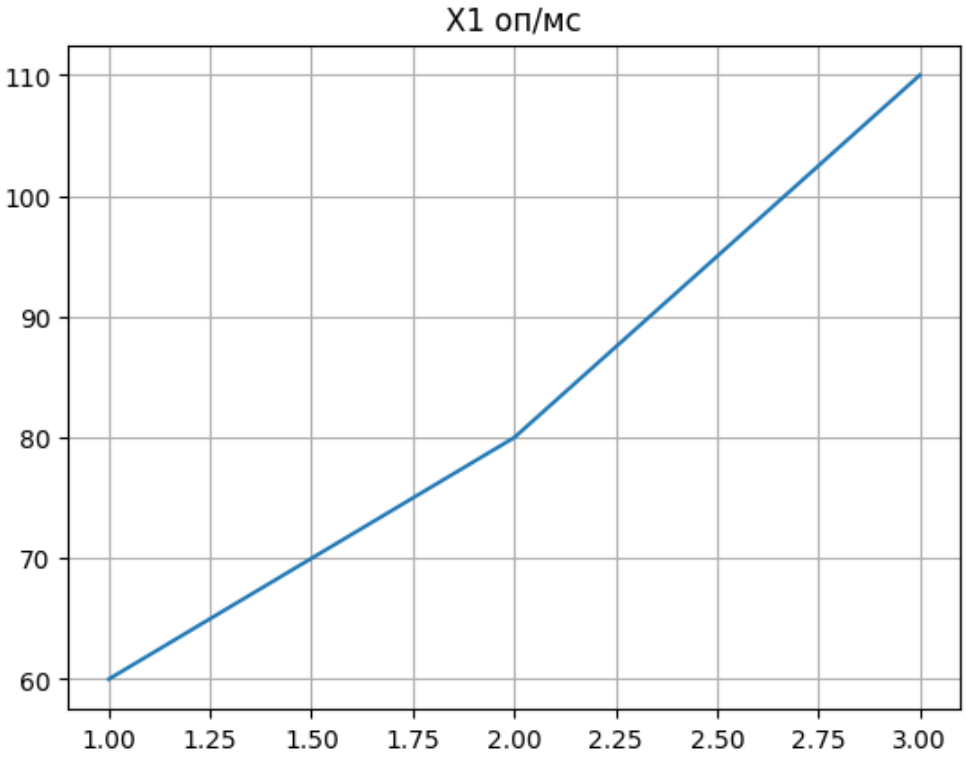


Рисунок 4.2 – Швидкодія мови програмування

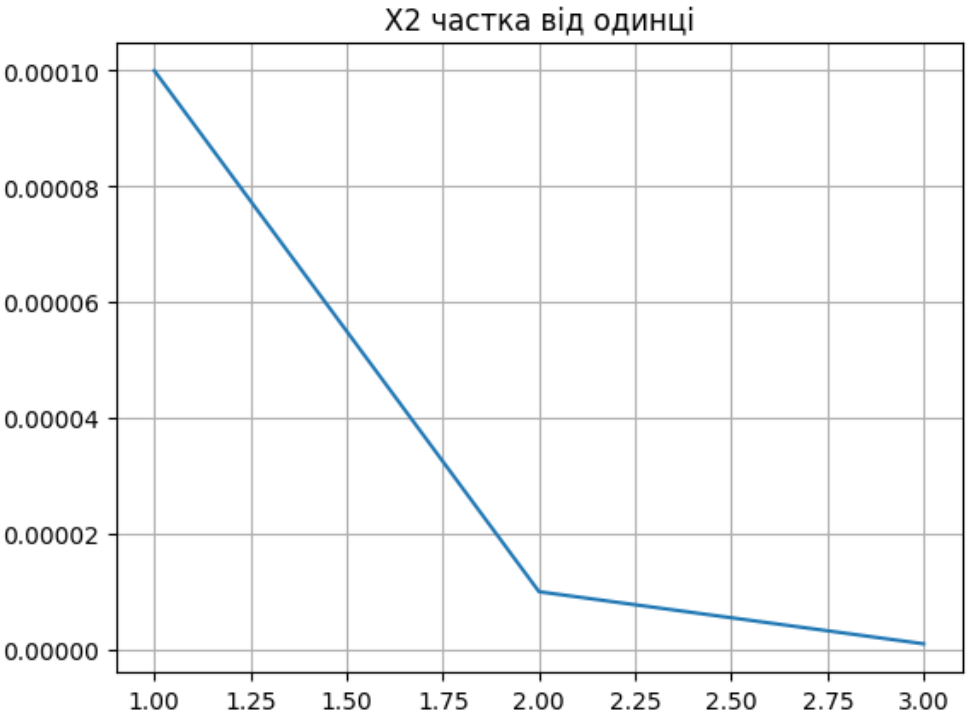


Рисунок 4.3 – Точність розв’язку

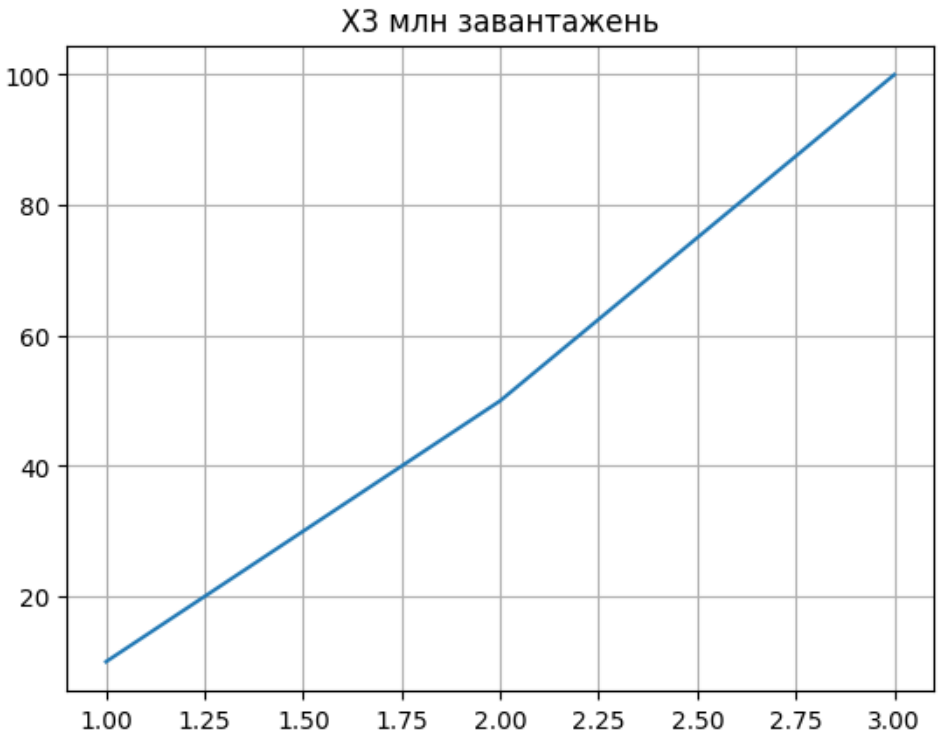


Рисунок 4.4 – Спосіб розповсюдження фреймворку

4.4 Аналіз експертного оцінювання параметрів

Для оцінювання параметрів експерти проводять їх обговорювання, а потім кожний з них проводить детальний аналіз, таким чином оцінюючи важливість кожного з параметрів для програмного продукту, що забезпечує машинне навчання моделей регресії максимально ефективно.

Для знаходження значимості кожного параметра використовуємо метод попарного порівняння. Для отримання оцінки залучається комісія експертів, яка складається з 6 людей. Для визначення коефіцієнтів виконуються наступні дії:

1. Присвоєння параметру різних рангів для визначення рівня його значимості.
2. Перевірка достовірності експертних оцінок.
3. Знаходження оцінки попарного пріоритету параметрів.
4. Визначення коефіцієнту значимості.

Результати ранжування експертами наведено у табл. 4.3

Таблиця 4.3 – Результати ранжування параметрів

Параметр	Назва	Одиниці виміру	Ранг						Сума рангів R_i	Відхи-лення Δ_i	Δ_i^2
			1	2	3	4	5	6			
X1	Продуктивність мови програмування	Операцій/мс	2	2	1	2	2	1	10	-2	4
X2	Точність розв’язку	Mb	3	3	3	3	3	3	18	6	36
X3	Популярність ресурсу	Млн заванта-жень	1	1	2	1	1	2	8	-4	16
	Разом		6	6	6	6	6	6	36	0	56

Знайдемо суму рангів кожного з параметрів і загальну суму рангів:

$$R_i = \sum_{j=1}^{N} r_{ij} R_{ij} = \frac{Nn(n+1)}{2} = 36,$$

де N – число експертів;

n – кількість параметрів.

Тепер знайдемо середню суму рангів:

$$T = \frac{1}{n} R_{ij} = 12$$

Після цього обчислимо відхилення суми рангів для кожного параметра:

$$\Delta_i = R_i - T$$

Тоді знаходимо загальну суму квадратів відхилення:

$$S = \sum_{i=1}^{N} \Delta_i^2 = 56$$

Насамкінець, обчислимо коефіцієнт узгодженості:

$$W = \frac{12S}{N^2(n^3 - n)} = \frac{12 \cdot 56}{6^2(3^3 - 3)} = 0,(7) > W_k = 0,67$$

Експертне ранжування є достовірним, оскільки коефіцієнт узгодженості є більшим за 0,67, тобто нормативне значення.

За отриманими результатами проводимо попарне порівняння між параметрами. Результати наведено у табл. 4.4.

Таблиця 4.4. Попарне порівняння параметрів

Параметри	Номер експерту						Остаточна оцінка	Значення
	1	2	3	4	5	6		
X1 і X2	<	<	<	<	<	<	<	0,5
X1 і X3	>	>	<	>	>	<	>	1,5
X2 і X3	>	>	>	>	>	>	>	1,5

Ступінь переваги параметра з індексом i над параметром з індексом j визначається за наступною. формулою:

$$a_{ij} = \begin{cases} 1.5 \text{ при } X_i > X_j \\ 1.0 \text{ при } X_i = X_j \\ 0.5 \text{ при } X_i < X_j \end{cases}$$

Використовуємо отримані оцінки для того, щоб скласти матрицю $A = \|a_{ij}\|$.

Розраховуємо коефіцієнт вагомості $K_{\text{ві}}$ за наступними формулами для кожного параметра:

$$K_{\text{ві}} = \frac{b_i}{\sum_{i=1}^{n} b_i}$$

$$b_i = \sum_{i=1}^{N} a_{ij}$$

Розраховуємо оцінки ітераційним методом, доки значення оцінок на певній ітерації не будуть відрізнятися від значень на попередній ітерації менш ніж на два відсотки. Ітерації виконуються за наступними формулами:

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

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

З табл. 4.5 робимо висновок, що відмінність мі значеннями коефіцієнтів є незначною, тобто не більше двох відсотків, тому ітераційний процес можна зупинити.

Таблиця 4.5 – Вагомості параметрів

x_i	x_j			Ітерація 1		Ітерація 2	
	X1	X2	X3	b_i	$K_{\text{ві}}$	b_i^1	$K_{\text{ві}}^1$
X1	1	0,5	1,5	3	0,(3)	8	0,32
X2	1,5	1	1,5	4	0,(4)	11,5	0,46
X3	0,5	0,5	1	2	0,(2)	5,5	0,22
Всього:				16	1	25	1

4.5 Аналіз рівня якості варіантів реалізації функцій

Рівень якості виконання основних функцій у кожному з варіантів реалізації визначається окремо.

Абсолютні значення параметрів *X2* (Точність розв'язку) та *X3* (Популярність ресурсу) задовольняють технічні вимоги до програмного продукту.

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

Для кожного варіанта реалізації програмного продукту знаходимо коефіцієнт технічного рівня за наступною формулою:

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

де n – кількість параметрів;

$B_{i,j}$ – оцінка i–го параметра в балах.

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

Результати обчислень наведено у табл. 4.6.

Таблиця 4.6 – Вагомості параметрів

Функція	Варіант реалізації	Абсолютне значення параметру	Оцінка параметру у балах	Вагомість параметру	Коефіцієнт якості
F1	A	60	3,6	0,32	1,152
	B	100	6	0,32	1,92
F2	A	10E-5	5	0,46	2,3
F3	A	80	7,5	0,22	1,65

За даними табл. 4.6 для кожного з варіантів обчислюємо його рівень якості:

$$K_{\text{ТЕР1}} = 1{,}152 + 2{,}3 + 1{,}65 = 5{,}102$$

$$K_{\text{ТЕР2}} = 1{,}92 + 2{,}3 + 1{,}65 = 5{,}87$$

З отриманих значень бачимо, що оптимальнішим є другий варіант, оскільки у нього коефіцієнт технічного рівня є більшим.

4.6 Економічний аналіз варіантів розробки програмного продукту

Для знаходження вартості створення програмного продукту почнемо з розрахунку трудомісткості.

Всі варіанти складаються з двох окремих завдань:

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

Ступенем новизни завдання 1 є група А, при цьому алгоритми, використовувані у цьому завданні відносяться до групи 1. Для завдання ступенем новизни є група Б, а складність його алгоритмів належить до групи 3.

При виконанні 1 використовується довідкова інформація, при цьому завдання 2 потребує інформації у вигляді даних.

Знайдемо норми часу на розробку кожного з завдань. Загальна трудомісткість знаходиться за наступною формулою:

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

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.

За нормами часу, завдання 1 розрахункового характеру зі степенем новизни А та алгоритму складності 1, має трудомісткість : T_P =37 людино-днів. Поправочний коефіцієнт, при використанні нормативно-довідкової інформації має значення: $K_П$ = 1.8. Поправочний коефіцієнт, що бере до уваги складність контролю вихідної та вхідної інформації рівний: $K_{СК}$ = 1. Необхідно врахувати, що при виконанні першого завдання використовуються стандартні модулі, тобто коефіцієнт $K_{СТ}$ = 0.9. Таким чином, трудомісткість першого завдання має значення:

$$T_1 = 37 * 1{,}8 * 0{,}9 = 59{,}94 \text{ людино-днів}$$

Друге завдання має алгоритм третьої групи складності, степінь новизни Б. Тоді T_P =29 людино-днів, $K_П$ =0.9, $K_{СК}$ = 1, $K_{СТ}$ =0.8:

$$T_2 = 29 * 0{,}9 * 0{,}8 = 20{,}88 \text{ людино-днів}$$

Для кожного з варіантів реалізації продукту сумуємо трудомісткості відповідних завдань. Отримуємо трудомісткість кожного завдання:

$$T_I = (59{,}94 + 20{,}88 + 4{,}8 + 20{,}88) * 8 = 852 \text{ людино-годин.}$$

$$T_{II} = (59{,}94 + 20{,}88 + 6{,}91 + 20{,}88) * 8 = 868{,}88 \text{ людино-годин.}$$

Варіант 2 має вищу трудомісткість.

Нехай оклад розробника становить 4200 грн., і над продуктом працює два розробники. Знайдемо середню зарплату за годину, використовуючи наступну формулу:

$$С_Ч = \frac{М}{t \cdot T_m} \text{ грн.,}$$

де М – місячний оклад працівників;

t – кількість робочих годин в день.

T_m – кількість робочих днів тиждень;

$$С_Ч = \frac{48000 + 48000}{3 \cdot 21 \cdot 8} = 190{,}476 \text{ грн.}$$

Тепер знайдемо заробітну плату за наступною формулою:

$$С_{ЗП} = С_Ч \cdot К_Д \cdot Т_i,$$

де $С_Ч$ – погодинна оплата праці програміста;

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

$Т_i$ – трудомісткість відповідного завдання;

Зарплата розробників по варіантах має наступні значення:

I. $С_{ЗП} = 190{,}476 * 852 * 1.2 = 194742{,}857$ грн.

II. $С_{ЗП} = 190{,}476 * 868.88 * 1.2 = 198601{,}143$ грн.

Відрахування на ЄСВ (єдиний соціальний внесок) становить 22%:

I. $С_{ВІД} = С_{ЗП} * 0.22 = 194742,857 * 0.22 = 42843,4286$ грн.

II. $С_{ВІД} = С_{ЗП} * 0.22 = 173776 * 0.22 = 43692,2514$ грн.

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

Так як одна ЕОМ використовується одним розробником з окладом 48000 грн. та має коефіцієнт зайнятості 0,2, маємо:

$$С_Г = 12 * М * К_З = 12 * 48000 * 0,2 = 115200 \text{ грн.}$$

Враховуючи додаткову заробітну плату:

$$С_{ЗП} = С_Г \cdot (1 + К_З) = 115200 \cdot (1 + 0.2) = 138240 \text{ грн.}$$

Відрахування на ЄСВ:

$$С_{ВІД} = С_{ЗП} \cdot 0.22 = 138240 \cdot 0,22 = 30412,8 \text{ грн.}$$

Амортизаційні відрахування знаходимо при вартості ЕОМ в 40000 грн та значенням амортизації 25%.

$$С_А = К_{ТМ} \cdot Ц_{ПР} \cdot К_А = 1.4 \cdot 40000 \cdot 0.25 = 14000 \text{ грн.},$$

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

$Ц_{ПР}$– договірна ціна приладу;

$К_А$– річна норма амортизації.

Витрати на профілактику та ремонт знаходимо за формулою:

$$С_Р = К_{ТМ} \cdot Ц_{ПР} \cdot К_Р = 1.4 \cdot 40000 \cdot 0.08 = 4480 \text{ грн.},$$

де $К_Р$– відсоток витрат на поточні ремонти.

Ефективний годинний фонд часу ПК за рік знаходимо за формулою:

$$Т_{ЕФ} = (Д_К - Д_Р - Д_В - Д_С) \cdot t3 \cdot КВ = (365 - 12 - 16 - 104) \cdot 8 \cdot 0{,}35 = \\ = 627{,}2 \text{ години},$$

де $Д_К$ – календарна кількість днів у році;

$Д_Р$ – кількість днів планових ремонтів устаткування;

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

t –кількість робочих годин в день;

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

Розмір витрат на електроенергію знаходимо за формулою:

$$С_{ЕЛ} = Т_{ЕФ} \cdot Ц_{ЕН} \cdot N_С \cdot K_З = 627{,}2 \cdot 5{,}67082 \cdot 0{,}2 \cdot 0{,}3 = 213{,}4043 \text{ грн.},$$

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

$N_С$ – середньо-споживча потужність приладу;

$К_З$– коефіцієнтом зайнятості приладу.

Розмір накладних витрат знаходимо за формулою:

$$С_Н = 0.67 Ц_{ПР} = 0{,}67 \cdot 40000 = 26800 \text{ грн.}$$

Таким чином, знаходимо розмір річних експлуатаційних витрат:

$$С_{ЕКС} = С_{ЗП} + С_{ВІД} + С_А + С_Р + С_{ЕЛ} + С_Н;$$

$$С_{ЕКС} = 138240 + 30412{,}8 + 14000 + 4480 + 213{,}4 + 26800 = \\ = 214146{,}2 \text{ грн.}$$

Тепер знаходимо собівартість однієї машино-години:

$$C_{М-Г} = \frac{C_{ЕКС}}{T_{ЕФ}} = \frac{214146,2}{627,2} = 341,432 \text{ грн/год.}$$

Всі роботи, пов‘язані з розробкою програмного продукту ведуться на ЕОМ, тому знаходимо розмір витрат на оплату машинного часу за наступною формулою:

$$C_М = C_{М-Г} \cdot T,$$

I. $C_М = 341,432 \cdot 852 = 290900,13$ грн.

II. $C_М = 341,432 \cdot 868,88 = 296663,5$ грн.

Розмір накладних витрат становить 67% від заробітної плати:

$$C_Н = C_{ЗП} \cdot 0,67,$$

I. $C_Н = 194742,857 \cdot 0,67 = 130477,71$ грн.

II. $C_Н = 198601,143 \cdot 0,67 = 133062,766$ грн.

Отже, вартість розробки програмного продукту в залежності від варіанту становить:

$$C_{ПП} = C_{ЗП} + C_{ВІД} + C_М + C_Н,$$

I. $C_{ПП} = 194742,857 + 42843,4286 + 290900,13 + 130477,71 =$
$= 658964,23$ грн.

II. $C_{ПП} = 198601,143 + 43692,2514 + 286578,346 + 133062,766 =$
$= 672019,65$ грн.

4.7 Вибір кращого варіанту ПП техніко-економічного рівня

Коефіцієнт техніко-економічного рівня знайдемо за наступною формулою:

$$К_{ТЕР\,j} = К_{Кj} / С_{Фj},$$

$$К_{ТЕР_1} = \frac{5{,}102}{649075} = 7{,}8604 \cdot 10^{-6},$$

$$К_{ТЕР_2} = \frac{5{,}87}{661934{,}5} = 8{,}8679 \cdot 10^{-6}.$$

З отриманих значень робимо висновок, що більш оптимальним є другий варіант реалізації, для якого значення коефіцієнту техніко-економічного рівня становить $8{,}8679 \cdot 10^{-6}$.

Цей варіант реалізації програмного продукту має такі параметри:

Обрана мова програмування – C#;

Використання чисельного методу знаходження похідних для оптимізації параметрів моделі;

Публікація програмного продукту у вигляді бібліотеки, яку можна встановити за допомогою звичайного менеджеру пакетів (у випадку C# це NuGet Package Manager).

Обраний варіант створення програмного продукту дає користувачу зручний прикладний інтерфейс програмування та швидку реалізацію програмної оболонки.

4.8 Висновки до розділу 4

В даному розділі дипломної роботи був проведений повний функціонально-вартісний аналіз створюваного програмного продукту. Крім цього, була знайдена оцінка основних функцій програмного продукту.

Під час проведення функціонально-вартісного аналізу розроблюваного програмного комплексу було визначено основні функції програмного продукту, та проведено оцінку параметрів, які його характеризують.

За результатами функціонально-вартісного аналізу було обрано найкращий варіант реалізації програмного продукту.

ВИСНОВКИ

У роботі було розглянуто методологію машинного навчання для побудови моделей регресії, зокрема глибоких нейронних мереж. Було виявлено, що тренування нейронної мережі складається з двох головних етапів: знаходження похідних та оптимізація параметрів моделі. Для реалізації першого етапу було розглянуто два методи: алгоритм зворотного поширення похибки та чисельний метод знаходження похідних. Для реалізації другого етапу також було розглянуто два методи: градієнтний спуск та алгоритм оптимізації Adam.

Після розбору теоретичного матеріалу було реалізовано програмний продукт, що дозволяє будувати глибокі нейронні мережі, тренувати їх та робити прогнози. У створеному програмному продукті використовуються всі розглянуті методи, окрім алгоритму зворотного поширення помилки через складність його реалізації.

За допомогою реалізованого фреймворку машинного навчання було створено та натреновано нейронну мережу, що вирішує реальну задачу по прогнозуванню врожайності базуючись на даних про погодні умови.

Можливими покращеннями фреймворку є розширення можливостей попередньої обробки даних, наприклад нормалізація ознак, створення нових ознак та видалення непотрібних. Також у майбутньому слід додати більше функцій втрат, функцій активації та оптимізаторів, оскільки це дозволить дослідникам створювати більш складні моделі, що призведе до вирішення ще більш широкого класу задач.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Sebastian Raschka. Python Machine Learning: Machine Learning and Deep Learning with Python, scikit-learn, and TensorFlow 2. Birmingham: Packt Publishing, 2019. 772 p.
2. Understanding Backpropagation Algorithm. URL: https://towardsdatascience.com/understanding-backpropagation-algorithm-7bb3aa2f95fd
3. How Does Backpropagation in a Neural Network Work? URL: https://builtin.com/machine-learning/backpropagation-neural-network
4. Understanding the Adam Optimization Algorithm: A Deep Dive into the Formulas. URL: https://jamhuri.medium.com/understanding-the-adam-optimization-algorithm-a-deep-dive-into-the-formulas-3ac5fc5b7cd3
5. Jeffrey Richter. CLR via C# (Developer Reference). New York: Microsoft Press, 2012. 896 p.
6. Підручник з машинного навчання для початківців: що таке, основи ML. URL: https://www.guru99.com/uk/machine-learning-tutorial.html
7. Understanding the Machine Learning Life Cycle. URL: https://deepchecks.com/understanding-the-machine-learning-life-cycle
8. Evaluation Of Major Deep Learning Frameworks. URL: https://analyticsindiamag.com/evaluation-of-major-deep-learning-frameworks
9. What is Adam Optimizer? URL: https://www.geeksforgeeks.org/adam-optimizer
10. Gradient Descent and Back-Propagation. URL: https://mriquestions.com/back-propagation.html

ДОДАТОК А ЛІСТИНГ ТРЕНУВАННЯ МОДЕЛЕЙ

```
Running architecture 1 with activation Relu
Loss after epoch 0: 3.6970329066960255
Loss after epoch 100: 0.8802970862776721
Loss after epoch 200: 0.8802959786760505
Loss after epoch 300: 0.8802959786738946
Loss after epoch 400: 0.8802959786739416
Loss after epoch 500: 0.8802959786739082
Loss after epoch 600: 0.8802959786737327
Loss after epoch 700: 0.8802959786740311
Loss after epoch 800: 0.8802959786738743
Loss after epoch 900: 0.880295978674147
predicted: 2793.387175828916 2882.7860330729454
2784.597050368524 2506.7983844719
real: 7000 4600 350 700
Running architecture 1 with activation LeakyRelu
Loss after epoch 0: 3.7654069124603065
Loss after epoch 100: 0.8802965412108518
Loss after epoch 200: 0.880295978674926
Loss after epoch 300: 0.8802959786736463
Loss after epoch 400: 0.8802959786738896
Loss after epoch 500: 0.88029597867367
Loss after epoch 600: 0.8802959786738881
Loss after epoch 700: 0.8802959786736057
Loss after epoch 800: 0.8802959786740591
Loss after epoch 900: 0.8802959786741978
predicted: 2793.3864293917804 2882.7869056224363
2784.5978841100073 2506.798844976824
real: 7000 4600 350 700
Running architecture 1 with activation Sigmoid
Loss after epoch 0: 2.9304983513827008
Loss after epoch 100: 0.8802968944922352
Loss after epoch 200: 0.880295978676169
Loss after epoch 300: 0.8802959786736552
Loss after epoch 400: 0.8802959786738689
Loss after epoch 500: 0.8802959786737358
Loss after epoch 600: 0.8802959786738156
Loss after epoch 700: 0.8802959786736283
Loss after epoch 800: 0.8802959786740843
Loss after epoch 900: 0.8802959786738076
predicted: 2793.3885523652307 2882.787560517896
2784.5965516546757 2506.800174750097
real: 7000 4600 350 700
Running architecture 1 with activation TanH
Loss after epoch 0: 3.0044732802669434
Loss after epoch 100: 0.8802961653680537
Loss after epoch 200: 0.8802959786739514
Loss after epoch 300: 0.8802959786738601
Loss after epoch 400: 0.8802959786742608
Loss after epoch 500: 0.8802959786738602
```

```
Loss after epoch 600: 0.8802959786741946
Loss after epoch 700: 0.880295978673834
Loss after epoch 800: 0.8802959786737583
Loss after epoch 900: 0.880295978674017
predicted: 2793.3870007286077 2882.7848120720623
2784.5990849785303 2506.798952901669
real: 7000 4600 350 700
Running architecture 2 with activation Relu
Loss after epoch 0: 2.8837094553787574
Loss after epoch 100: 0.5862714148802323
Loss after epoch 200: 0.45926526409478163
Loss after epoch 300: 0.38025660172082515
Loss after epoch 400: 0.326172524754808
Loss after epoch 500: 0.2863124126977211
Loss after epoch 600: 0.25533762616533745
Loss after epoch 700: 0.22930922838731632
Loss after epoch 800: 0.20306310862362334
Loss after epoch 900: 0.16706497862838868
predicted: 6192.065707438436 4068.2066771767522
1303.884621101826 2056.5809048902865
real: 7000 4600 350 700
Running architecture 2 with activation LeakyRelu
Loss after epoch 0: 3.253071809034994
Loss after epoch 100: 0.7204074706269105
Loss after epoch 200: 0.634201964232379
Loss after epoch 300: 0.5503978754140634
Loss after epoch 400: 0.4798270223445599
Loss after epoch 500: 0.4310709280894916
Loss after epoch 600: 0.3896099435035148
Loss after epoch 700: 0.35171909031114096
Loss after epoch 800: 0.32501169271725056
Loss after epoch 900: 0.30766560689189987
predicted: 4906.57162257777 3850.9819588353307 -
673.7663683500318 743.1127760038703
real: 7000 4600 350 700
Running architecture 2 with activation Sigmoid
Loss after epoch 0: 1.7262207319793332
Loss after epoch 100: 0.8791458564687439
Loss after epoch 200: 0.8441065127014606
Loss after epoch 300: 0.8165995276346992
Loss after epoch 400: 0.7874977449300227
Loss after epoch 500: 0.7553918378965753
Loss after epoch 600: 0.7195557647820582
Loss after epoch 700: 0.6798710897863605
Loss after epoch 800: 0.6369022593659223
Loss after epoch 900: 0.5919121042693211
predicted: 3637.945205685772 3232.958326472906
1133.5138901812231 2374.263577961867
real: 7000 4600 350 700
Running architecture 2 with activation TanH
Loss after epoch 0: 1.7747741240491903
Loss after epoch 100: 0.6120157060341888
Loss after epoch 200: 0.4515371451082862
```

```
Loss after epoch 300: 0.36702811739852365
Loss after epoch 400: 0.30877359854584013
Loss after epoch 500: 0.2549676059951581
Loss after epoch 600: 0.205079913363778
Loss after epoch 700: 0.16621288093182388
Loss after epoch 800: 0.13734440213304458
Loss after epoch 900: 0.11528484277753195
predicted: 7030.752084940061 4316.314317384254 -21.7101725513171
1482.126596977786
real: 7000 4600 350 700
Running architecture 3 with activation Relu
Loss after epoch 0: 6.845338794349734
Loss after epoch 100: 0.5762925047239983
Loss after epoch 200: 0.45630015614356023
Loss after epoch 300: 0.3623858459721692
Loss after epoch 400: 0.2967511179618907
Loss after epoch 500: 0.2713475815790658
Loss after epoch 600: 0.23618771540168998
Loss after epoch 700: 0.22089180736614322
Loss after epoch 800: 0.21417440086457326
Loss after epoch 900: 0.2104070675307775
predicted: 6946.4956530299905 4546.8801808081225
1600.9476970374876 1600.9476970374876
real: 7000 4600 350 700
Running architecture 3 with activation LeakyRelu
Loss after epoch 0: 60.74212051352963
Loss after epoch 100: 0.6045666271983096
Loss after epoch 200: 0.41882216973600994
Loss after epoch 300: 0.28145792310676876
Loss after epoch 400: 0.17211824887611918
Loss after epoch 500: 0.10996808908455481
Loss after epoch 600: 0.07686938459293419
Loss after epoch 700: 0.05770308319077443
Loss after epoch 800: 0.04599972126046899
Loss after epoch 900: 0.038370294976392834
predicted: 6784.5033391759425 4477.386994615528
316.83117902924914 714.603052193308
real: 7000 4600 350 700
Running architecture 3 with activation Sigmoid
Loss after epoch 0: 1.1047351647349024
Loss after epoch 100: 0.9129794246431702
Loss after epoch 200: 0.8751240042148112
Loss after epoch 300: 0.85925218112175
Loss after epoch 400: 0.8455961756325658
Loss after epoch 500: 0.8311281970965116
Loss after epoch 600: 0.8148892538714773
Loss after epoch 700: 0.7961988635083519
Loss after epoch 800: 0.7745001659997922
Loss after epoch 900: 0.7494158145545412
predicted: 3296.4968110353902 2501.19300074665
1897.1420261823991 2312.765241433489
real: 7000 4600 350 700
Running architecture 3 with activation TanH
```

```
Loss after epoch 0: 2.792033814112329
Loss after epoch 100: 0.36339241600917177
Loss after epoch 200: 0.20204841364931125
Loss after epoch 300: 0.11703434002562338
Loss after epoch 400: 0.07036468842890167
Loss after epoch 500: 0.04553034373676408
Loss after epoch 600: 0.031109638710393572
Loss after epoch 700: 0.021773838738216637
Loss after epoch 800: 0.015513894974357617
Loss after epoch 900: 0.011320678592251875
predicted: 6913.4900026043815 4676.981209348185
327.6164675486657 519.5153220402053
real: 7000 4600 350 700
```

ДОДАТОК Б ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ

ДИПЛОМНА РОБОТА

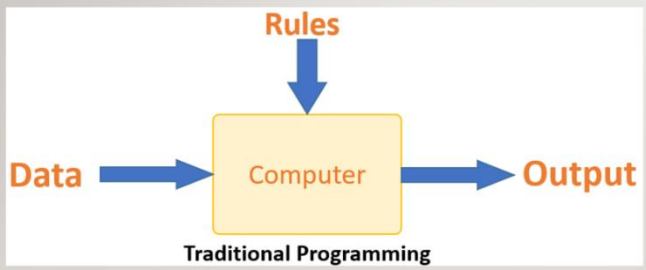
НА ТЕМУ: «СТВОРЕННЯ ФРЕЙМВОРКУ ДЛЯ МАШИННОГО НАВЧАННЯ МОДЕЛЕЙ РЕГРЕСІЇ

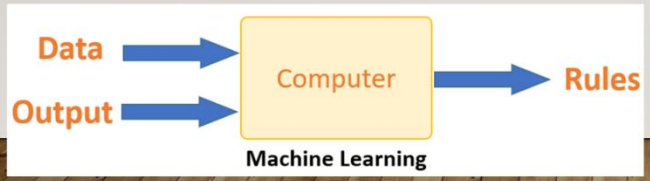
ВИКОНАВ: ТУНКІН Є. А., КА-02

КЕРІВНИК: ДОЦ. САВЧЕНКО І. О.

2

ЗАГАЛЬНИЙ ОГЛЯД МАШИННОГО НАВЧАННЯ





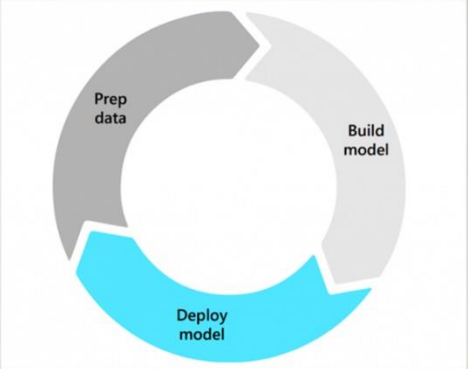
Машинне навчання – це галузь штучного інтелекту, яка вивчає методи побудови алгоритмів, які можуть самостійно вчитися і покращувати свою продуктивність на основі даних. Основна ідея полягає в тому, щоб використовувати великі обсяги даних для автоматичного визначення шаблонів, а потім використовувати ці знання для прийняття рішень без явного програмування.

Розвиток машинного навчання відбувався протягом багатьох десятиліть і зазнав значного прогресу завдяки поєднанню наукових досліджень, технологічних проривів та зростання обсягів доступних даних.

3

ПОСТАНОВКА ЗАДАЧІ НА ДИПЛОМНУ РОБОТУ

Завданням дипломної роботи є створення фреймворку (бібліотеки), яка дозволить використовувати машинне навчання людям, що не є спеціалістами у цій галузі. Фреймворк має надавати можливості для завантаження і попередньої обробки даних, а також створення та тренування моделей регресії. Користувачам також слід надати можливість вибору архітектури моделі, її компонентів та значень гіперпараметрів.



4

ІСНУЮЧІ РІШЕННЯ

Оскільки складність побудови моделі машинного навчання з нуля є дуже високою, існує багато фреймворків, які вирішують цю та інші проблеми. Всі вони належать провідним IT-компаніям, які вкладають мільйони доларів у розвиток та популяризацію цих продуктів. Найпопулярнішими фреймворками машинного навчання є TensorFlow (Google), PyTorch (Meta) та Scikit-Learn. Всі вони підтримуються широким колом мов програмування та платформ.

5

ІСНУЮЧІ МЕТОДИ ПОБУДОВИ ФРЕЙМВОРКУ МАШИННОГО НАВЧАННЯ

Нейронною мережею є модель, де вхідний вектор множиться на матрицю, після чого до результату застосовується певна зростаюча нелінійна функція (її називають функцією активації), і цей крок повторюється декілька разів.

$$f(x) = ReLU\left(\begin{pmatrix} x^{(1)} & x^{(2)} & x^{(3)} \end{pmatrix} * \begin{pmatrix} w_{11}^{(1)} & w_{12}^{(1)} \\ w_{21}^{(1)} & w_{22}^{(1)} \\ w_{31}^{(1)} & w_{32}^{(1)} \end{pmatrix}\right) * \begin{pmatrix} w_{11}^{(2)} & w_{12}^{(2)} \end{pmatrix}$$

$$ReLU(x) = \begin{cases} 0, & x \le 0 \\ \alpha x, & x > 0 \end{cases}$$

Функція втрат:

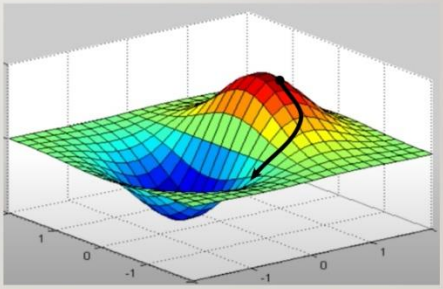
$$J\left(\widehat{y}, y\right) = \frac{1}{n}\sum_{i=1}^{n}(\hat{y}_i, y_i)^2$$

6

ОБРАНІ МЕТОДИ ПОБУДОВИ ФРЕЙМВОРКУ МАШИННОГО НАВЧАННЯ

Для обчислення похідних функції втрат було обрано чисельний метод через простоту його реалізації.

Для оптимізації параметрів моделі реалізовано градієнтний спуск та метод Adam, що використовується для прискорення алгоритму градієнтного спуску шляхом врахування «експоненційно зваженого середнього» градієнтів. Використання середніх значень змушує алгоритм наближатися до мінімумів швидше.



7

РЕАЛІЗАЦІЯ ЗНАХОДЖЕННЯ ПОХІДНИХ

Похідна функції втрат по параметру $w_{j,k}^{(i)}$ знаходиться за наступною формулою:

$$d_{j,k}^{(i)} \coloneqq \frac{J^{+} - J^{-}}{2\epsilon},$$

Де J^{+}та J^{-} – значення функції втрат при збільшенні та зменшенні параметра $w_{j,k}^{(i)}$ на ϵ відповідно.

```
private double[][,] GetDerivativesNumericall(double[,] x, double[,] y, double epsilon = 1e-10)
{
    var derivatives = new double[Weights.Length][,];
    for (int i = 0; i < Weights.Length; i++)
    {
        derivatives[i] = new double[Weights[i].GetLength(0), Weights[i].GetLength(1)];
        for (int j = 0; j < Weights[i].GetLength(0); j++)
        {
            for (int k = 0; k < Weights[i].GetLength(1); k++)
            {
                Weights[i][j, k] += epsilon;
                var lossPlus = Loss(x, y);
                Weights[i][j, k] -= 2 * epsilon;
                var lossMinus = Loss(x, y);
                Weights[i][j, k] += epsilon;
                derivatives[i][j, k] = (lossPlus - lossMinus) / (2 * epsilon);
            }
        }
    }

    return derivatives;
}
```

8

РЕАЛІЗАЦІЯ ОПТИМІЗАЦІЇ ПАРАМЕТРІВ МОДЕЛІ

```
2 references
public void Optimize(double[][,] weights, double[][,] grads)
{
    for (int i = 0; i < weights.Length; i++)
    {
        weights[i] = weights[i].Subtract(grads[i].Multiply(LearningRate));
    }
}
```

```
2 references
public void Optimize(double[][,] weights, double[][,] grads)
{
    if (m is null || v is null || mBias is null || vBias is null)
    {
        m = new double[weights.Length][,];
        mBias = new double[weights.Length][,];
        v = new double[weights.Length][,];
        vBias = new double[weights.Length][,];
        for (int i = 0; i < weights.Length; i++)
        {
            m[i] = new double[weights[i].GetLength(0), weights[i].GetLength(1)];
            mBias[i] = new double[weights[i].GetLength(0), weights[i].GetLength(1)];
            v[i] = new double[weights[i].GetLength(0), weights[i].GetLength(1)];
            vBias[i] = new double[weights[i].GetLength(0), weights[i].GetLength(1)];
        }

    }

    for (int i = 0; i < weights.Length; i++)
    {
        m[i] = m[i].Multiply(Beta1).Add(grads[i].Multiply(1 - Beta1));
        v[i] = v[i].Multiply(Beta2).Add(grads[i].ElementWise(grads[i], (a, b) => a * b).Multiply(1 - Beta2));
        mBias[i] = m[i].Multiply(1 / (1 - Beta1));
        vBias[i] = v[i].Multiply(1 / (1 - Beta2));
        LearningRate *= Math.Sqrt(1 - Beta2) / (1 - Beta1);
        weights[i] = weights[i].Subtract(mBias[i].ElementWise(vBias[i].ElementWise(a => Math.Sqrt(a) + Epsilon), (a, b) => a / b).Multiply(LearningRate));
    }
}
```

9

ПРИКЛАД ПОБУДОВИ НЕЙРОННОЇ МЕРЕЖІ

```
var model = new NeuralNetwork([1, 1])
{
    Activation = new Relu(),
    LossFunction = new MeanSquaredErrorLoss(),
    Optimizator = new GradientDescentOptimizator() { LearningRate = 0.025 },
};
```

ПРИКЛАД ВИРІШЕННЯ РЕАЛЬНОЇ ЗАДАЧІ

10 У якості реальних даних було обрано датасет пов'язаний з врожайністю певної ділянки землі та погодних умов, за яких ця врожайність спостерігалась. Задача полягає в тому, щоби знайти спосіб передбачити, об'єм врожаю знаючи, які будуть погодні умови. Датасет містить 8 змінних:

nitrogen – коефіцієнт вмісту нітрогену у ґрунті;

phosphorus – коефіцієнт вмісту фосфору у ґрунті;

potassium – коефіцієнт вмісту калію у ґрунті;

temperature – температура в градусах Цельсія;

humidity – відносна вологість повітря;

pH_Value – pH значення у ґрунті;

rainfall – кількість опадів у міліметрах;

yield – цільова змінна, кількість кілограмів зібраного врожаю.

	Nitrogen	Phosphorus	Potassium	Temperature	Humidity	pH_Value	Rainfall	Crop	Yield
0	90	42	43	20.879744	82.002744	6.502985	202.935536	Rice	7000
1	85	58	41	21.770462	80.319644	7.038096	226.655537	Rice	5000
2	60	55	44	23.004459	82.320763	7.840207	263.964248	Rice	7000
3	74	35	40	26.491096	80.158363	6.980401	242.864034	Rice	7000
4	78	42	42	20.130175	81.604873	7.628473	262.717340	Rice	120000

РОЗВ'ЯЗОК ТА АНАЛІЗ РЕЗУЛЬТАТІВ

Для розв'язку поставленої задачі було запропоновано 3 варіанти архітектури нейронної мережі:

Номер нейронної мережі	Розміри прихованих шарів
1	-
2	[16]
3	[16, 16]

Після тренування моделей результати було внесено у таблицю:

Архітектура НМ/Функція активації	Relu	LeakyRelu	Sigmoid	Гіперболічний тангенс
1	0,88	0,88	0,88	0,88
2	0,167	0,3	0,592	0,115
3	0,21	0,038	0,749	0,011

ДЯКУЮ ЗА УВАГУ