МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ХЕРСОНСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ
Факультет комп'ютерних наук фізики та математики
Кафедра інформатики, програмної інженерії та економічної кібернетики

РЕАЛІЗАЦІЯ МЕТОДУ ПОШУКУ ГЛОБАЛЬНОГО ОПТИМУМУ ІЗ ЗАСТОСУВАННЯМ МОВИ ПРОГРАМУВАННЯ PYTHON

Кваліфікаційна робота (проєкт)
на здобуття ступеня вищої освіти "бакалавр"

Виконав: студент 4 курсу
Спеціальності 121 Інженерія програмного забезпечення
Освітньо-професійної програми «Інженерія програмного забезпечення» першого (бакалаврського) рівня освіти
Коломієць Олег Едуардович
Керівники
кандидат фізико-математичних наук, доцент
Вейцбліт Олександр Йосипович
кандидат фізико-математичних наук, доцент
Єрмолаєв Вадим Анатолійович
Рецензент
кандидат фізико-математичних наук, доцент
Плоткін Яків Давидович

Херсон – 2020

ЗМІСТ

ВСТУП 4
РОЗДІЛ 1 МЕТОДИ ОПТИМІЗАЦІЇ 6
1.1. Поняття оптимізації 6
1.2. Оптимізація в нейронних мережах 8
1.3. Методи оптимізації та їх модифікації 11
1.3.1. Стохастичний градієнт спуск (SGD) 11
1.3.2. Пакетний градієнтний спуск (BGD) 12
1.3.3. Моментум (Momentum) 12
1.3.4. Прискорений градієнт Нестерова (NAG) 12
1.3.5. Адаптивний градієнт (AdaGrad) 12
1.3.6. Ададельта (AdaDelta) 13
1.3.7. Оцінка адаптивного моменту (Adam) 13
1.3.8. Метод Ньютона (Newton) 14
1.3.9. Спряжений градієнт (Conjugate gradient) 14
1.3.10. Квазіньютонівський (Quasi-Newton) 14
1.3.11. Гауса–Ньютона (Gauss-Newton) 15
1.3.12. Левенберга–Марквардта (Levenberg-Marquardt) 15
РОЗДІЛ 2 МОВА ПРОГРАМУВАННЯ PYTHON 16
2.1. Мова Python і її особливості 16
2.2. Середовище розробки PyCharm 18
2.3. Використані бібліотеки 19
2.3.1. NumPy 19
2.3.2. multiprocessing 21
2.3.3. time 23
РОЗДІЛ 3 РЕАЛІЗАЦІЯ 24
3.1. Опис алгоритму 24
3.2. Реалізація алгоритму 25

3.2.1. find_optim 27
3.2.2. find_attractors 28
3.2.3. pr_cells 30
3.3. Робота користувача з програмою 33
ВИСНОВКИ 35
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 36
ДОДАТКИ 39
Додаток А 39
Додаток Б 43
Додаток В 45
Додаток Г 46

ВСТУП

Актуальність: Методи оптимізації застосовуються як математична основа у багатьох технологіях та галузях науки, зокрема у нейронних мережах, задачах штучного інтелекту. Значні перешкоди при реалізації таких задач пов'язані з відсутністю поширених методів оптимізації, які б дозволяли:

1) пошук глобальних екстремумів у заданій області;
2) широке застосовування паралельних операцій при їх реалізації.

Об'єкт дослідження: методи оптимізації, методи розв'язування задач штучного інтелекту.

Предмет дослідження: метод оптимізації, що дозволяє:

1) пошук глобальних екстремумів у заданій області;
2) широке застосовування паралельних операцій при їх реалізації;
3) програмні засоби для реалізації цього методу та його застосування до задач штучного інтелекту.

Мета роботи: реалізація методу оптимізації, що дозволяє:

1) пошук усіх локальних екстремумів (глобального) у заданій n-мірній області;
2) широке застосовування паралельних операцій при їх пошуку.

Завдання:

1) опанувати чисельні методи, зокрема методи оптимізації, що застосовуються у нейронних мережах та задачах штучного інтелекту;
2) реалізувати метод оптимізації, що дозволяє:
    a) пошук глобальних екстремумів у заданій області;
    b) широке застосовування паралельних операцій при їх реалізації.
3) шляхом чисельних експериментів обрати варіант цього методу, найбільш зручний для застосування в нейронних мережах;

4) реалізувати обраний варіант методу в вигляді програмного засобу на мові програмування Python.

РОЗДІЛ 1
МЕТОДИ ОПТИМІЗАЦІЇ

1.1. Поняття оптимізації

Математична оптимізація – це вибір кращого елемента (по деякому критерію) з деякої множини доступних альтернатив [18]. Різні проблеми оптимізації виникають у всіх видах дисциплін, і розвиток методів їх вирішення цікавив математику протягом століть.

У простому випадку проблема оптимізації полягає в максимізації або мінімізації функції шляхом перебору вхідних значень з доступної множини та обчислення значення функції. Більш загально, оптимізація включає пошук "найкращих доступних" значень певної цільової функції з заданою областю визначення (або вхідними даними), включаючи безліч різних типів цільових функцій та різних типів областей.

Проблема оптимізації може бути сформульована наступним чином:
Дано: функція f: A → $\mathbb{R}$ з деякого набору A до реальних чисел.
Знайти: елемент x0 ∈ A такий, що f (x0) ≤ f (x) для всіх x ∈ A ("мінімізація") або такий, що f (x0) ≥ f (x) для всіх x ∈ A ("максимізація").

Таке формулювання називається проблемою оптимізації. Багато насущних і теоретичних проблем можуть бути змодельовані таким чином.

Хоча локальний мінімум настільки ж прийнятний, як і будь-який з сусідніх елементів, глобальний мінімум є принаймні настільки ж хорошим, як і кожен з можливих елементів. Як правило, якщо цільова функція не опукла, в проблемі мінімізації може бути кілька локальних мінімумів. В опуклій функції, якщо є локальний мінімум, який є внутрішнім (не на межі множини можливих елементів), то він також є і глобальним мінімумом, але якщо функція не є опуклою, то вона може

мати більше одного локального мінімуму, не всі з яких обов’язково будуть глобальними мінімумами.

Нижче наведена ілюстрація даної проблеми:

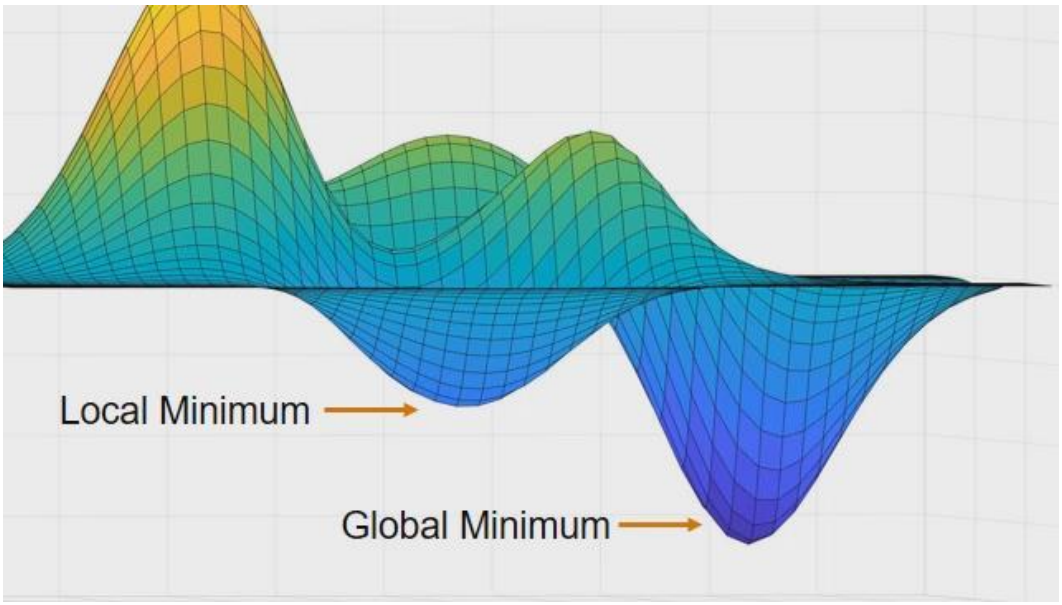

Рис. 1.1. Проблема оптимізації

Велика кількість алгоритмів, запропонованих для вирішення невипуклих задач, включаючи більшість комерційно доступних розв’язувачів – не здатні розмежовувати локально оптимальні рішення та глобально оптимальні рішення, і розглядають перші, як фактичне рішення вхідної проблеми.

Глобальна оптимізація – це галузь прикладної математики, яка займається розробкою детермінованих алгоритмів, здатних гарантувати збіжність за деякий кінцевий час до фактичного оптимального рішення задачі оптимізації невипуклої функції [31].

Проблеми оптимізації виникають і при розв’язуванні задач штучного інтелекту, зокрема при навчанні нейронних мереж.

1.2. Оптимізація в нейронних мережах

Нейронна мережа – математична модель, а також її програмне втілення, побудована за принципом організації та функціонування мереж нервових клітин живого організму [28].

Нижче наведена ілюстрація структури типової нейронної мережі:

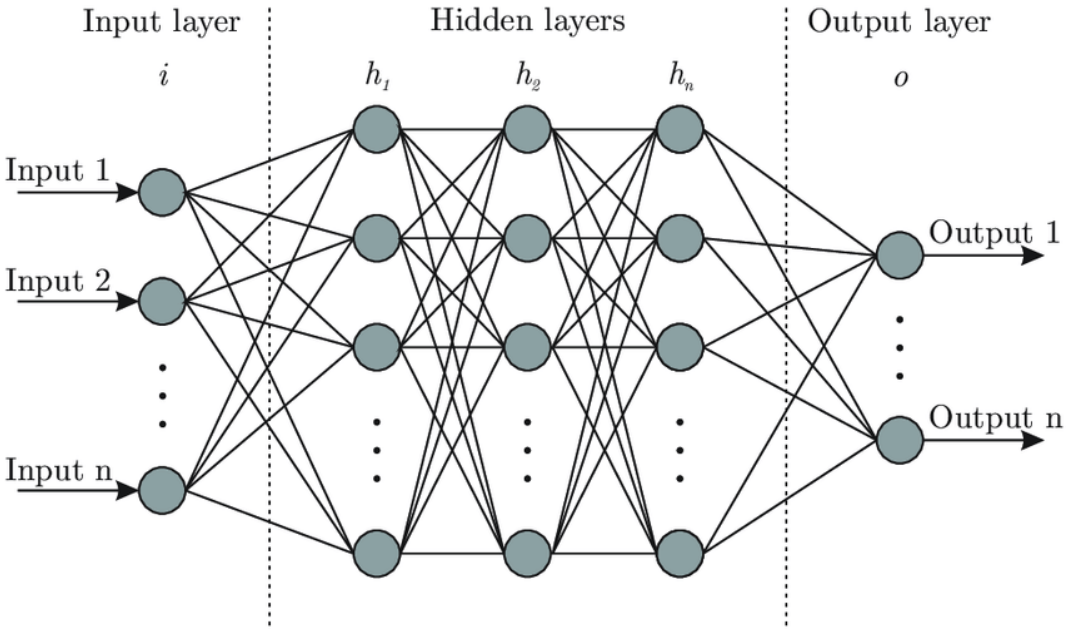

Рис 1.2. Нейронна мережа

Можливість навчання – їх основна перевага перед традиційними алгоритмами. Процесс навчання полягає в знаходженні коефіцієнтів зв'язків (ваг) між шарами нейронів. В процесі навчання ці коефіцієнти корегуються відповідно до бажаного результату, як наслідок мережа набуває здатності виявляти залежність між вхідними і вихідними даними, а також виконувати узагальнення. У разі успішного навчання мережа здатна повертати бажаний результат на підставі даних навчальної вибірки.

Тестування якості навчання нейромережі необхідно проводити на прикладах, які не брали участі в її навчанні. При цьому число тестових прикладів має бути тим більше, чим вище якість навчання. Виходить, що

тестування добре навчених нейронних мереж стає дуже важким завданням.

Така проблема здебільшого пов’язана з проблемою оптимізації. Оскільки не вдається гарантовано знаходити глобальні оптимуми, необхідна більша кількість екземплярів для навчання, що також сповільнює навчання мережі.

В нейронних мережах найчастіше використовуються різновиди алгоритму, що має назву – градієнтний спуск. Градієнтний спуск – метод знаходження локального екстремуму функції за допомогою руху уздовж градієнта [21]. Найбільш простий в реалізації з усіх методів локальної оптимізації. Має досить слабкі умови збіжності, при цьому швидкість збіжності достатньо мала (лінійна).

Нейромережу можна уявити як функцію f, від деяких параметрів x, f (x), тоді обчислюючи градієнтний вектор і рухаючись в певну сторону від x, ми зможемо знайти такий параметр x, при якому значення нашої функції (відповідь нейромережі) буде для нас найкращим.

Нижче наведена ілюстрація методу градієнтного спуску:

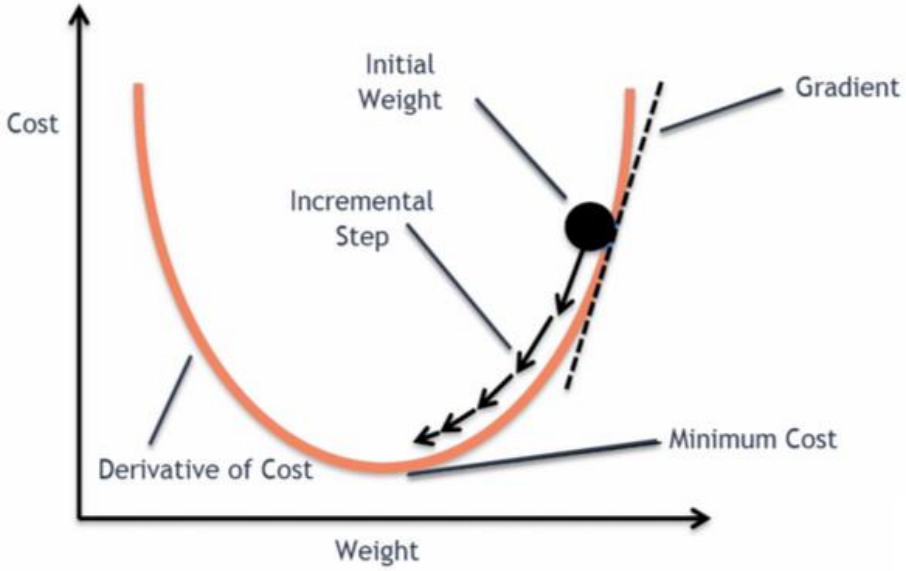

Рис. 1.3. Градієнтний спуск

Однак в реальності не все так просто, тому що насправді параметрів функції f набагато більше, та й сама функція набагато складніше (так, наприклад, для зображення кожен піксель може бути параметром).

Звичайний градієнтний спуск має ряд проблем:

1) зникаючі або вибухаючі градієнти:

   інколи у процесі навчання нейронної, при корегуванні ваг між шарами мережі може статись так, що значення градієнтного вектору у певних точках може бути близьким до 0, а оскільки значення ваг корегуються відповідно до значення градієнтного вектору, то це фактично буде означати “параліч” мережі, при якому навчання припиниться або дуже сповільниться; можливі й ситуації коли значення градієнта стає занадто великим, тоді перестрибуючи великі відстані, пошук хорошого екстремуму буде скоріше всього невдалим, а отже і навчання;

2) застрявання в локальних мінімумах:

   n-мірний простір функції помилки нейронної мережі дуже складний і складається з великої кількості пагорбів, долин, обривів, тощо, це означає велику кількість локальних екстремумів і як наслідок головну проблему: дійшовши до деякого локального мінімуму не є зрозумілим чи потрібно продовжувати рух, чи це вже є глобальний мінімум;

3) застрявання в сідлових точках:

   сідлові точки – це такі точки, в яких значення одних параметрів дорівнює мінімуму, а інших максимуму, при цьому перші похідні в цій точці будуть дорівнювати 0. Наслідки від потрапляння в такі точки при базовому підході означають значне погіршення навчаємості мережі.

Нижче наведена ілюстрація сідлової точки:

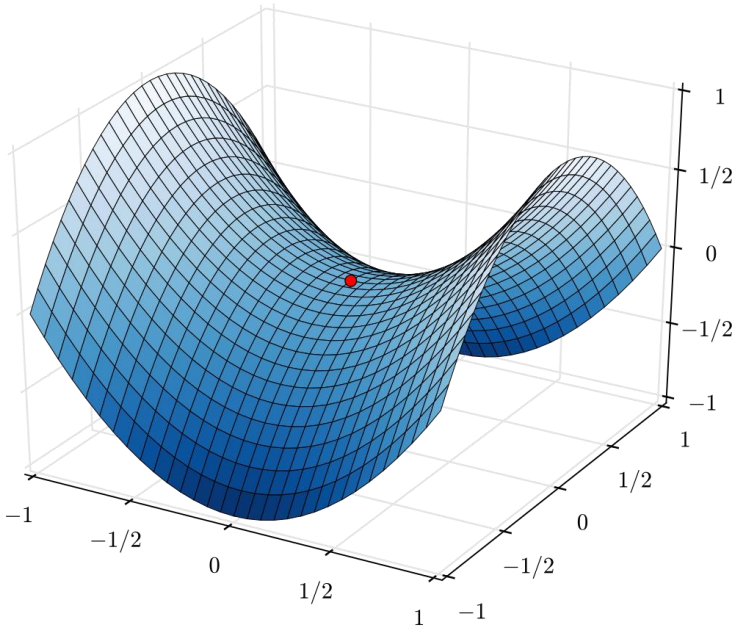

Рис 1.4. Сідлова точка

Тому постійно створюються нові варіації і методи. Нижче приведені одні з найбільш поширених алгоритмів та їх модифікацій.

1.3. Методи оптимізації та їх модифікації

Більшість методів, що використовуються в нейронних мережах є модифікаціями градієнтного спуску. Нище приведені одні з найбільш відомих модифікацій алгоритмів першого порядку.

1.3.1. Стохастичний градієнт спуск (SGD)

Це варіант градієнтного спуску. Він намагається частіше оновлювати параметри моделі. У цьому варіанті параметри моделі змінюються після обчислення втрат на кожному з екземплярів навчання. Отже, якщо набір даних містить 1000 екземплярів, стохастичний спуск буде оновлювати параметри моделі 1000 разів, а не один раз, як у звичайному алгоритмі.

Оскільки параметри моделі часто оновлюються, оновлюється і градієнтний вектор, тому збільшується вірогідність знайти кращий екстремум.

1.3.2. Пакетний градієнтний спуск (BGD)

Це поліпшення як стохастичного так і стандартного градієнтного спуску. У ньому параметри моделі оновлюються після кожного пакету. Набір даних поділяється на різні партії і після кожної партії параметри оновлюються.

1.3.3. Моментум (Momentum)

Моментум був винайдений для зменшення великої дисперсії у стохастичному спуску і пом’якшення збіжності. Він прискорює збіжність відповідно до потрібного напрямку та зменшує коливання до нерелевантного напрямку.

У цьому методі використовується гіперпараметр, відомий як імпульс, що символізується «γ».

1.3.4. Прискорений градієнт Нестерова (NAG)

Моментум може бути хорошим методом, але якщо імпульс занадто великий, алгоритм може пропустити локальні мінімуми і продовжити зростати. Отже, для вирішення цієї проблеми був розроблений даний алгоритм.

Цей метод “дивиться вперед” і виконує обчислення на основі приблизного значення майбутнього параметра, а не поточного.

1.3.5. Адаптивний градієнт (AdaGrad)

Одним з недоліків усіх вище наведених оптимізаторів є те, що швидкість навчання є однаковою для всіх параметрів і для кожного

екземпляру. Цей оптимізатор змінює швидкість навчання. Він змінює швидкість навчання "η" для кожного параметра на кожному кроці "t".

Однак він має недолік – оскільки він зберігає і накопичує квадрати градієнтів для підрахунків, з часом, швидкість навчання стає близькою до 0, що фактично означає неможливість подальшого навчання. Для усунення цієї проблеми були запропоновані такі методи як AdaDelta.

1.3.6. Ададельта (AdaDelta)

Це модифікація адаптивного градієнту, що прагне усунути проблему згасаючої швидкості навчання. Замість накопичення усіх квадратів попередніх градієнтів, Ададельта обмежує кількість раніше накопичених градієнтів деяким фіксованим розміром w.

У даному алгоритмі використовується експоненціально рухоме середнє, а не сума всіх градієнтів.

1.3.7. Оцінка адаптивного моменту (Adam)

Алгоритм адаптивної оцінки моменту працює з використанням імпульсів першого та другого порядку. Ідея, полягає в тому, що ми не хочемо пересуватись дуже швидко тому, що можемо перестрибнути мінімум, ми хочемо трохи зменшити швидкість для ретельного пошуку.

Крім зберігання експоненціально згасаючого середнього квадратів градієнтів, як AdaDelta, Адам також зберігає експоненціально згасаюче середнє минулих градієнтів M (t).

Вищезазначені алгоритми є алгоритмами першого порядку оскільки використовують першу похідну функції похибки. Існують й методи, що застосовують другу похідну – алгоритми другого порядку. Вони використовують матриці Гессе та інші схожі матриці для вирішення деяких проблем простішого піходу – першого порядку, однак обчислення таких матриць може бути досить “важким” завданням. До таких

алгоритмів належать: метод Ньютона, спряженого градієнта, квазіньютонівський, Гауса–Ньютона, Левенберга–Марквардта та інші.

1.3.8. Метод Ньютона (Newton)

Даний метод є повним алгоритмом другого порядку. Він обчислює наступний крок для оновлення використовуючи обернені матриці Гессе. Алгоритм Ньютона дуже ефективний і вимагає лиш одну ітерацію для досягнення точки мінімуму, однак його обчислювальна складність O(N^3) на ітерацію, також він вимагає багато пам’яті для збереження великих NxN матриць. Що робить його непрактичним для застосування безпосередньо до нейронних мереж.

Однак існують багато методів, що використовують наближені матриці до матриць Гессе і можуть бути практично застосовані.

1.3.9. Спряжений градієнт (Conjugate gradient)

Перший наближений підхід називається спряженим градієнтом. Він використовує метод Полак–Рібьєра як напрям спряження для адаптивної зміни напряму спуску на кожній ітерації. Використання спряженого напряму зменшує необхідність великих обчислювальних потужностей, так як це O(N) метод.

Цей метод не використовує матрицю Гессе безпосередньо змінюючи поточний крок оновлення з напрямком попередньої ітерації. Крім цього, спряжений градієнт використовує метод лінійного пошуку для визначення довжини кроку тому може бути використаний лише для пакетного навчання.

1.3.10. Квазіньютонівський (Quasi-Newton)

Квазіньютонівський метод використовує наближення до матриць Гессе (деяку оцінку) для розрахунків оновлень на кожній ітерації. Існує декілька алгоритмів оцінки. Найбільш відомим є алгоритм Бройдена–

Флетчера–Гольдфарба–Шанно (BFGS). Метод значно зменшує обчислювальну складність, зменшуючи її до O(N^2), порівняно до методу Ньютона.

Однак метод все одно має недолік у вигляді необхідності збереження NxN матриць. Існують модифіцкації, що намагаються зменшити об’єм необхідної пам’яті, однак, навіть так цей метод має гіршу продуктивність ніж методи першого порядку.

1.3.11. Гауса–Ньютона (Gauss-Newton)

Ще один з методів наближення до оберненого Гессіана. Алгоритм Гауса–Ньютона використовується для вирішення завдань нелінійним методом найменших квадратів.

Алгоритм є модифікацією методу Ньютона для знаходження мінімуму функції. На відміну від методу Ньютона, алгоритм Гаусса–Ньютона може бути використаний тільки для мінімізації суми квадратів, але його перевага в тому, що метод не вимагає обчислення других похідних, що може виявитися складною задачею.

Тим не менш, звичайна варіант алгоритму не здатний впоратись з невизначеним Гессіаном.

1.3.12. Левенберга–Марквардта (Levenberg-Marquardt)

Даний метод був запропонований в зв’язку з недоліками методу Гауса–Ньютона. Може бути розглянутий як комбінація методу Ньютона і градієнтного спуску. Метод вводить параметр регуляризації для запобігання проблем з матрицею Гессе. Однак це може вважатись недоліком, оскільки він потребує окремого налаштування для хороших результатів.

В цій роботі буде запропонована і реалізована власний оптимізаційний алгоритм, що базується на градієнтному спуску.

РОЗДІЛ 2
МОВА ПРОГРАМУВАННЯ PYTHON

Далі нами будуть приведені та описані технології та засоби, що були використані при реалізації запропонованого алгоритму оптимізації.

2.1. Мова Python і її особливості

Python – інтерпретована мова програмування високого рівня, загального призначення. Філософія дизайну Python підкреслює читабельність коду завдяки помітному використанню пробілу [16]. Його мовні конструкції та об'єктно-орієнтований підхід допомогають програмістам написати чіткий логічний код.

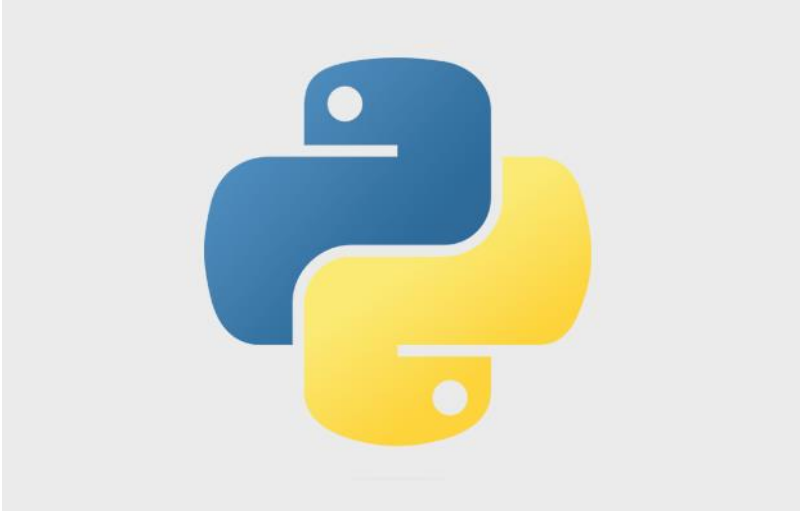

Рис 2.1. Python

Python має динамічну типізацію та автоматичний збирач сміття. Він підтримує декілька парадигм програмування, включаючи структурне, об'єктно-орієнтоване та функціональне. Python відомий своєю багатофункціональною стандартною бібліотекою.

Майже всі програмісти, що займаються машинним навчанням використовують Python для своєї роботи. Від розробки до підтримки,

Python допомагає розробникам бути продуктивними та впевненими у програмі, яку вони розробляють.

Існує багато переваг використання Python в машинному навчанні. Ці переваги збільшують популярність Python:

1) простий і постійний:

Python – це проста мова, що пропонує надійний код.

Машинне навчання – це складні алгоритми та різнобічні робочі процеси. Отже простота Python допомагає розробникам впоратися зі складними алгоритмами, економити час, шляхом зосередження розробників на вирішення проблем задач штучного інтелекту, а не на технічних особливостях мови;

2) гнучкість:

Python відомий як найбільш гнучка мова в машинному навчанні. Він надає різні варіанти для користувачів. Фактор гнучкості зменшує вірогідність зробити помилку. Це дозволяє програмістам взяти ситуацію повністю під свій контроль і працювати над проблемою комфортно.

Програмісти можуть комбінувати Python з іншими мовами, щоб отримати бажані результати. Крім того, ця мова пропонує можливість вибрати будь-яку парадигму програмування.

Програмуючи на Python не потрібно компілювати вихідний код. Програмісти можуть впроваджувати свої зміни в коді і швидко бачити результати;

3) бібліотеки та фреймворки:

Для розробки найкращих рішень розробники вимагають добре структурованого та добре перевіреного середовища. Алгоритми машинного навчання дуже складні, але Python – це рятувальний інструмент із широким спектром бібліотек та фреймворків.

Ці бібліотеки допомагають програмісту і скорочують час розробки. Python має багатий стек технологій і має різний набір бібліотек для машинного навчання;

4) читабельність:

Python легко читати, тому розробники можуть легко розуміти код. Також вони можуть внести зміни в нього, для відповідності їх потребам. Більше того, можливість помилок, плутанини та конфліктів дуже незначна. Таким чином, розробники можуть ефективно обмінюється ідеями, інструментами та алгоритмами з іншими;

5) незалежність від платформи:

Платформна незалежність відображає універсальність мови програмування. Це стосується фреймворка або мови програмування, яка дозволяє розробнику реалізовувати речі на одному комп'ютері та використовувати їх на іншому.

Python – мова, що не залежить від платформи.

Все це робить Python лідером серед технологій для розв'язування задач штучного інтелекту і тому він був обраний нами як мова програмування для реалізації поставлених мети.

2.2. Середовище розробки PyCharm

PyCharm – це інтегроване середовище розробки (IDE), яке використовується в програмуванні, спеціально для мови Python [27].

Рис 2.2. PyCharm

PyCharm надає багатий функціонал, для зручної і швидкої розробки, а саме:

1) аналіз коду з доповненням коду, виділенням синтаксису та помилок, швидкими виправленнями;
2) навігація за проектами та файлами: спеціалізовані перегляди проектів, перегляди структури файлів та швидкий перехід між файлами, класами, методами та вживаннями;
3) рефакторинг Python: включає перейменування, метод вилучення, введення змінної, введення константи та інші;
4) інтегрований налагоджувач Python;
5) інтегровані юніт-тести;
6) інтегрований контроль версій;
7) підтримка таких наукових інструментів, як matplotlib, numpy та scipy.

Вищезазначений функціонал дозволяє швидко відлагоджувати програму при розробці складних алгоритмів. Саме тому PyCharm був обраний нами як IDE.

2.3. Використані бібліотеки

2.3.1. NumPy

NumPy – основний пакет для наукових обчислень з Python. Він містить серед іншого:

1) потужний об'єкт N-розмірного масиву;
2) складні функції перетворення;
3) інструменти для інтеграції коду C / C ++;
4) корисну лінійну алгебру, перетворення Фур'є та інструменти для роботи з випадковими числами [7].

Окрім очевидних застосувань, NumPy також може бути використаний як ефективний багатовимірний контейнер для загальних

даних. Можна визначити довільні типи даних. Це дозволяє NumPy легко та швидко інтегруватись із найрізноманітнішими базами даних.

При роботі з великими масивами NumPy дозволяє максимально пришвидшити усі можливі операції над елементами масиву, створення нових масивів, ефективно керувати пам’яттю комп’ютера, оскільки фактично усі операції виконуються на C++. Крім того масиви в NumPy водночас є і матрицями, що дозволяє виконувати не лише поелементні операції, а й, наприклад, перемноження матриць.

В нашій роботі використовуються різні типи масивів і матриць, тому ефективна і перевірена структура даних є незамінним інструментом, що надає значні переваги над стандартними об’єктами Python.

Нами були використані наступні функції:

1) array() – створює об’єкт масиву на основі стандартного списку Python, містить багато опціональних параметрів один з яких – бажаний тип даних;
2) zeros() – створює об'єкт масиву потрібної форми і типу даних, заповнений нулями;
3) ones() – створює об'єкт масиву потрібної форми і типу даних, заповнений одиницями;
4) frombuffer() – дозволяє інтерпретувати буфер як 1-мірний масив. Надзвичайно корисна і важлива функція, що довзволяє ефективно керувати пам’яттю;
5) copy() – вертає копію масиву;
6) sum() – вертає суму елементів отриману по заданій осі n-мірного масиву;
7) subtract() – вертає масив, що складається з поелементної різниці двох масивів-аргументів ;
8) prod() – вертає добуток елементів отриманий по заданій осі n-мірного масиву;

9) less() – вертає масив, що складається з поелементного порівняння елементів масивів-аргументів;
10) array_equal() – вертає результат порівняння двох масивів поелементно;
11) tolist() – конвертує і вертає n-мірний масив як вкладений список;
12) reshape() – змінює форму масиву не змінюючи даних;
13) ravel() – вертає сплющений 1-мірний масив на основі n-мірного масиву.

2.3.2. multiprocessing

Одна з основних ідей, що лежить в основі створення модифікації алгоритму оптимізації полягає в використанні паралельних обчислень. В стандартній бібліотеці Python існують кілька варіантів для різних форм паралельності, зокрема threading та multiprocessing.

Модуль threading не надає можливості одночасних обчислень у зв'язку з глобальним блокуванням інтерпрітатора (GIL). Тому нами був обраний модуль multiprocessing.

multiprocessing – це пакет, який підтримує створення процесів, використовуючи API, схоже до модулю threading [13]. Він пропонує як локальну, так і віддалену паралельність, ефективно обходячи глобальне блокування інтерпретатора, використовуючи підпроцеси замість потоків. І хоча при такому підході потрібно більше ресурсів на синхронізацію процесів, оскільки кожен модуль працює із власною пам'яттю, завдяки цьому модуль дозволяє програмісту повністю використовувати декілька процесорів на комп'ютері.

У нашій роботі ми використали наступні засоби, а саме функції:

1) Process() – створює об'єкт-процес, має декілька аргументів, найголовніші target – функція, що буде виконуватись при запуску процесу та args – аргументи цієї цільової функції, якщо такі є.
   Кожен запущений процес являє собою окремий процес Python;

2) RawArray() – створює незахищений масив, що може бути переданий до будь-якого процесу, без його копіювання. Таким чином різні процеси можуть отримати доступ до одного й того ж самого масиву.

   Хоча використання типів даних без блокування є небезпечним, у нашому випадку це є обґрунтованим, оскільки кожен процес буде працювати з окремими даними в масиві;

3) RawValue() – створює незахищену змінну типу integer, що може бути передана до будь-якого процесу. Зручно використовувати для комунікації між головним і дочірніми процесами;

4) Manager().list() – створює об’єкт-список, що може бути переданий до будь-якого процесу, без його копіювання.

   Головними відмінностями від RawArray є можливість додавати та вилучати елементи, таким чином його розмір не є константним, однак він має захист з використанням блокування, що робить його не дуже придатним для паралельних обчислень, в тім об’єкт такого типу має інше призначення в нашій реалізації;

5) start() – метод об’єкта Process, що запускає процес. Як вже згадувалось раніше, кожен процес являє собою окремий процес Python, тому запуск процесів достатньо “важкий” і потребує багато часу та ресурсів.

   При розробці багатопоточної програми необхідно мати на увазі, що при малих обчисленнях продуктивність може бути меншою аніж при використанні лише 1 процесу. У нашому випадку, оскільки метою є деякий варіант глобальної оптимізації, використання більш ніж одного процесу має сенс;

6) join() – метод об’єкта Process, що приєднує процесс після завершення його роботи. Цей метод необхідно викликати для остаточного завершення роботи і звільнення пам’яті;

7) cpu_count() – повертає кількість ядер в комп’ютері. Кількість створюваних процесів необхідно розраховувати з врахуванням

кількості ядер комп’ютера, оскільки велика кількість процесів лише збільшить затрати на ресурси та час на міжпроцесорну синхронізацію;

8) get_lock() – метод об’єкта RawValue, що дозволяє блокувати об’єкт від впливу інших процесів. Необхідно використовувати у випадках коли операція над об’єктом не є атомарною, наприклад при інкрементації.

2.3.3. time

Цей модуль надає різні функції, пов'язані з часом. Більшість функцій, визначених у цьому модулі, викликають функції на платформі C з тим самим ім'ям [14].

В нашій роботі, ми використовуємо метод sleep() у цільових функціях процесів модуля multiprocessing для зменшення затрат на ресурси під час їх простоювання. Також можна отримати час затрачений на роботу програми з допомогою методу time().

РОЗДІЛ 3
РЕАЛІЗАЦІЯ

У цьому розділі буде описаний алгоритм, а також його реалізація і взаємодія з нею з точки зору користувача.

3.1. Опис алгоритму

Створений алгоритм має наступну послідовність кроків:

1) обране скалярне n-мірне поле розбивається на деяку кількість клітинок m, створюється список розміру m;
2) в кожній з m клітинок списку виділяються k точок для яких знаходяться їх образи відповідно до градієнтного методу;
3) визначаються ті клітинки в які потрапив хоча б 1 образ і відзначаються у списку;
4) клітинки в які не потрапив жоден образ відкидаються;
5) кроки 2-4 повторюються поки кількість клітинок у списку не буде сталою;
6) отримані в результаті клітинки розбивається на деяку кількість t підклітинок;
7) кроки 2-6 повторюються поки не буде задовільнений певний критерій.

Нижче наведена ілюстрація блок-схеми:

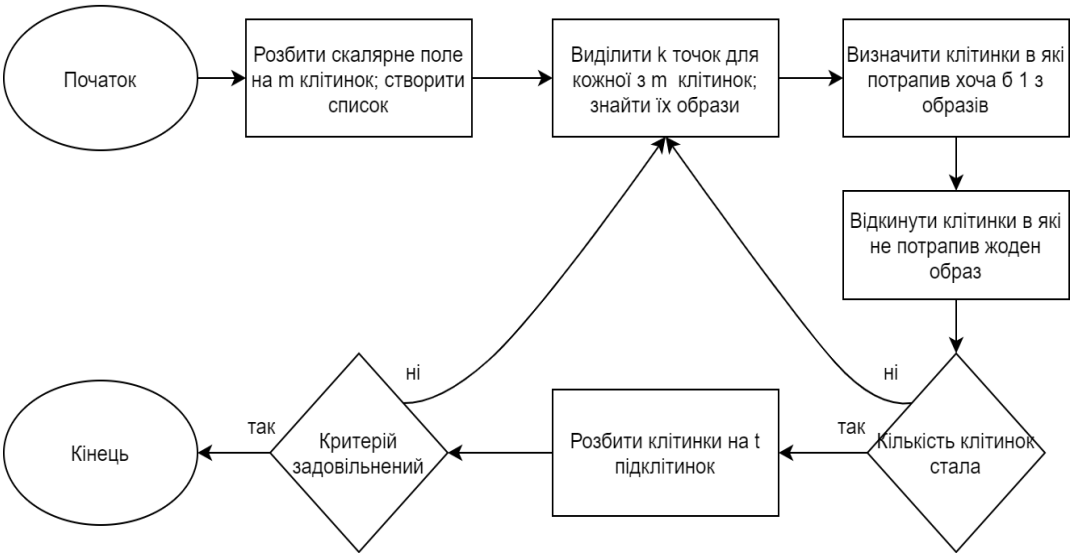

Рис. 3.1. Розроблений алгоритм

Таким чином вдається знайти усі можливі локальні оптимуми з деякою заданою точність. У даному алгоритмі кроки 2-3 передбачають максимальну паралелізацію операцій для протидії масивним обчисленням. Кількість клітинок, точок, підклітинок визначається довільно відповідно до обраної цільової функції, розміру поля, тощо.

3.2. Реалізація алгоритму

При реалізації даного алгоритму нами були реалізовані наступні функції:

1) f() – для отримання значення цільової функції в точці;
2) get_grad() – для отримання значення градієнтного вектору в точці;
3) get_dist() – для отримання значення відстані між двома точками;
4) calc_edges_len() – для розрахунку довжин граней клітинок;
5) pr_cells() – для здійснення обчислень над клітинками;
6) find_attractors() – для здійснення одної ітерації (пункт 5) алгоритму;
7) find_optim() – для здійснення роботи всього алгоритму;

8) br_field() – для розбиття n-мірного скалярного поля на клітинки;

9) br_cell() – для розбиття клітинки на точки;

10) br_cells() – для розбиття клітинок на підклітинки.

Першим чином підключаємо бібліотеки:

```
import numpy as np
import multiprocessing as mp
import time
import math

import mymodule as mm
```

Далі в головній частині програми необхідно встановити значення наступних змінних:

1) r – список, що містить границі n-мірного поля;
2) l_rate – коефіцієнт швидкості навчання, що використовується у градієнтному методі;
3) lim – критерій, що означає максимальну кількість разів ділення клітинок на підклітинки (пункт 6);
4) dim – розмірність простору;
5) div – кількість поділів по кожній осі n-мірного поля, звідси кількість клітинок div^dim;
6) sec_div – кількість поділів по кожної осі n-мірної клітинки, звідси кількість точок в клітинці sec_div^dim;
7) sub_div – кількість поділів по кожної осі n-мірної клітинки, звідси кількість підклітинок sub_div^dim.

Далі створюються необхідні для роботи алгоритму структури даних:

```
e_l = calc_edges_len(r, div)
cells = mp.RawArray('f', div ** dim * dim)
cells_buf = np.frombuffer(cells, 'f').reshape((div ** dim, dim))
cells_buf[:] = mm.br_field(r, e_l, dim, div)
```

*content_l = mp.RawArray('b', div ** dim)*

content_l_buf = np.frombuffer(content_l, 'b')

*content_l_buf[:] = np.ones(shape=(1, div ** dim), dtype=np.bool).ravel()*

Тут e_l – довжини граней клітинок, cells – отримані в результаті поділу n-мірного поля координати клітинок, content_l – список клітинок.

Створення спочатку “порожнього” масиву RawArray, а потім заповнення його з використанням функції frombuffer дає набагато більшу швидкість, аніж створення масиву одразу в RawArray.

Також створюються структури даних для взаємодії між процесами:

instr = mp.RawValue('i', 0)

rdycntr = mp.Value('i', 0)

ctpr = mp.Manager().list()

Тут instr – використовується для надання інструкцій процесам, rdycntr – для перевірки готовності процесів, ctpr – список, що містить номери клітинок, що мають опрацювати кожен з процесів.

3.2.1. find_optim

Наступним кроком викликається головна функція find_optim():

cells = find_optim(content_l, cells, e_l, dim, sec_div, sub_div, l_rate, lim, instr, rdycntr, ctpr)

Результатом її є список, що містить фінальний набір клітинок. Ця функція є досить абстрактною оскільки головною її метою є слідкування за критерієм lim, та створення нових масивів після поділення клітинок.

Отож першим кроком в ній викликається функція find_attractors():

vol1 = len(content_l)

find_attractors(content_l, cells, edges_len, dim, sec_div, l_rate, instr, rdycntr, ctpr)

vol2 = np.sum(content_l)

vol1 та vol2 – об’єми, що дорівнюють кількості активних (залишених) клітинок. Далі йде перевірка на виконання критерію:

while e < lim:

Поки критерій задовільний ділимо клітинки, створюємо для них нові масиви і знову викликаємо find_attractors():

```
np_cells = np.frombuffer(cells, 'f').reshape((vol1, dim))
edges_len = [i/sub_div for i in edges_len]
new_cells = [np_cells[i] for i in range(0, vol1) if content_l[i]]
new_cells = mm.br_cells(new_cells, vol2, edges_len, dim, sub_div)
vol1 = vol2*sub_div**dim
cells = mp.RawArray('f', int(vol1*dim))
cells_buf = np.frombuffer(cells, 'f').reshape((vol1, dim))
cells_buf[:] = new_cells
content_l = mp.RawArray('b', int(vol1))
content_l_buf = np.frombuffer(content_l, 'b')
content_l_buf[:] = np.ones(shape=(1, vol1), dtype=np.bool).ravel()
find_attractors(content_l, cells, edges_len, dim, sec_div, l_rate, instr, rdycntr, ctpr)
vol2 = np.sum(content_l)
e += 1
```

В результаті отримуємо клітинки, що залишились і вертаємо їх:

```
np_cells = np.frombuffer(cells, 'f').reshape((vol1, dim))
l_cells = [np_cells[x] for x in range(0, len(np_cells)) if content_l[x] == 1]
return l_cells
```

3.2.2. find_attractors

Перейдемо до функції find_attractors(). Ця функція виконує одну ітерацію алгоритму (пункт 5). Саме в ній створюються процеси і виконується керування ними.

```
shape = len(content_l)
previous_state = np.zeros(shape=(1, shape), dtype=np.bool)
numofpr = mp.cpu_count()
```

np_content_l = np.frombuffer(content_l, 'b')

prcs = [mp.Process(target=pr_cells, args=(content_l, cells, shape, edges_len, dim, sec_div, l_rate, instr, rdycntr, ctpr, i)) for i in range(0, numofpr)]

previous_state – список, що дорівнює попередньому значенню списку content_l, оскільки робота цієї функції буде продовжуватися до тих пір поки кількість клітинок не буде сталою, такий список є необхідної для перевірки цієї умови. prcs – список процесів, що будуть використані, максимальна їх кількість дорівнює numofpr – кількості ядер комп'ютера.

Запускаємо процеси:

for i in range(0, numofpr):

prcs[i].start()

Поки кількість клітинок не є сталою:

while not np.array_equal(np_content_l, previous_state):

Початкова інструкція встановлюється на 0, що періодично поміщає процеси в сон, для економії ресурсів, і виконується розподіл клітинок між процесами:

instr.value = 0

time.sleep(0.01)

previous_state = np.copy(np_content_l)

distrib = [[] for i in range(0, numofpr)]

cur_p_num = 0

for i in range(0, shape):

if np_content_l[i] == 1:

distrib[cur_p_num].append(i)

cur_p_num += 1

if cur_p_num == numofpr:

cur_p_num = 0

Оскільки в список content_l будуть занесені нові дані, спочатку його потрібно обнулити:

np_content_l[:] = np.zeros(shape=(1, shape), dtype=np.bool).ravel()

Далі встановлюються інструкції 1 та 2. Перед встановленням нової інструкції необхідно зачекати поки усі процеси закінчать роботу пов’язану з попередньою:

```
instr.value = 1
while rdycntr.value < numofpr:
    time.sleep(0.005)
rdycntr.value = 0
instr.value = 2
while rdycntr.value < numofpr:
    time.sleep(0.005)
rdycntr.value = 0
```

Коли кількість клітинок, не змінюється після 1 ітерації у циклі, цикл припиняється, а процеси приєднуються:

```
instr.value = 3
for i in range(0, numofpr):
    prcs[i].join()
```

3.2.3. pr_cells

pr_cells() – цільова (target) функція процесів. В ній відбуваються основні обчислення пов’язані з градієнтним методом та його проблемами, такими як сідлові точки.

Починається вона зі створення певних змінних та множини, що будуть застосовані далі:

```
np_cells = np.frombuffer(cells, 'f').reshape((shape, dim))
np_content_l = np.frombuffer(content_l, 'b')
self_hit = set()
flag1 = True
flag2 = True
```

Керування процесами здійснюється з допомогою об’єкта instr, в середині циклу:

while True:

Всього можливі 4 інструкції: 0, 1, 2 і 3. Перша інструкція – 0, застосовується для економії ресурсів та зміни значення змінних flag1 та flag2:

if instr.value == 0:

flag1 = True

flag2 = True

time.sleep(0.005)

Друга інструкція – 1, застосовується коли необхідно виконати обчислення , згідно з градієнтним методом та занести результати у список:

elif (instr.value == 1) and flag1:

flag1 застосовується для запобігання повторного виконання обчислень, коли поточний процес завершив обчислення, а інший ні, тоді flag1 встановлюється як False, що означає те, що процес вже виконав необхідні обчислення.

Далі для кожної з клітинок знаходяться образи точок, що були виділені в них:

for i in ctpr[pid]:

points = mm.br_cell(np_cells[i], edges_len, dim, sec_div)

*grads = [get_grad(x)*l_rate for x in points]*

points = np.subtract(points, grads)

Для кожного з образів знаходиться клітинка в яку він потрапляє. В даному алгоритмі клітинки були розбиті таким чином, що вони знаходяться в лексикографічному порядку, тому задача знаходження клітинки в яку потрапляє образ стає простішою, оскільки можна використати алгоритм схожий до сортування вставками:

for j in points:

is_in = False

u_bound = shape

```
l_bound = 0
while l_bound != (u_bound - 1):
    temp = u_bound - (u_bound - l_bound)/2
    if np.less(np_cells[math.floor(temp)], j).prod():
        l_bound = math.ceil(temp)
    else:
        u_bound = math.floor(temp)
if np.less(np_cells[l_bound], j).prod():
    np_content_l[l_bound] = 1
    is_in = True
```

Якщо образ точки клітинки потрапив в ту ж саму клітинку, то ця інформація заноситься до множини self_hit. Такі клітинки далі проходять додаткову перевірку:

```
if (l_bound == i) and (l_bound != 0) and (l_bound != (shape - 1)):
    set(self_hit).add(i)
```

Якщо ж образ точки деякої клітинки вийшов за межі обраного скалярного поля, то таку клітинку доцільно залишити:

```
if not is_in:
    np_content_l[i] = 1
```

Після завершення операцій для кожної з клітинок, значення flag1 змінюється і інкрементується значення об’єкта rdycntr, що дає головній програмі зрозуміти кількість процесів які виконали обчислення:

```
flag1 = Falseмс
with rdycntr.get_lock():
    rdycntr.value += 1
```

Третя інструкція – 2, застосовується після виконання обчислень інструкції 1, в ній перевіряються клітинки, образи яких потрапили в себе ж. В ній також застосовується допоміжна змінна flag2, як і при інструкції 1:

```
elif (instr.value == 2) and flag2:
```

```
for i in self_hit:
    img = np.subtract(np_cells[i], get_grad(np_cells[i])*l_rate)
    st_dist = get_dist(img, np_cells[i])
    for j in range(0, 2):
        img = np.subtract(img, get_grad(img)*l_rate)
    if st_dist < get_dist(img, np_cells[i]):
        np_content_l[i] = 0
flag2 = False
self_hit = []
with rdycntr.get_lock():
    rdycntr.value += 1
```

Четверта інструкція – 3, застосовується коли необхідно завершити роботу процесів:

```
elif instr.value == 3:
    break
```

Коли жоден з варіантів не може бути виконаний, наприклад при instr = 1, flag1 = False виконується гілка else, для економії ресурсів:

```
else:
    time.sleep(0.005)
```

3.3. Робота користувача з програмою

Програма складається з 2 файлів: start_module.py та main_module.py.

start_module необхідний для генерації функцій br_field(), br_cell(), br_cells(). Оскільки користувач може на свій розсуд змінювати такі змінні як: dim, div, sec_div, sub_div то вищезазначені функції не мають чіткої структури, тому користувач має запустити файл start_module з бажаними параметрами, на основі яких функції create_code_1(), create_code_2(), create_code_3() фактично згенерують файл mymodule.py, що буде містити

необхідні функції. Крім того згенерується файл d.txt, в якому будуть збережені параметри для їх подальшого використання у main_module.

У модулі main_module є дві функції, що потребують налаштування від користувача: f() та get_grad(). Оскільки користувач може змінювати цільову функцію на власний розсуд, то він має змінити формули знаходження значень цільової функції та її градієнту. Такі змінні як r, l_rate, lim також корегуються користувачем відповідно до його потреб.

Нижче наведена ілюстрація файлової структури:

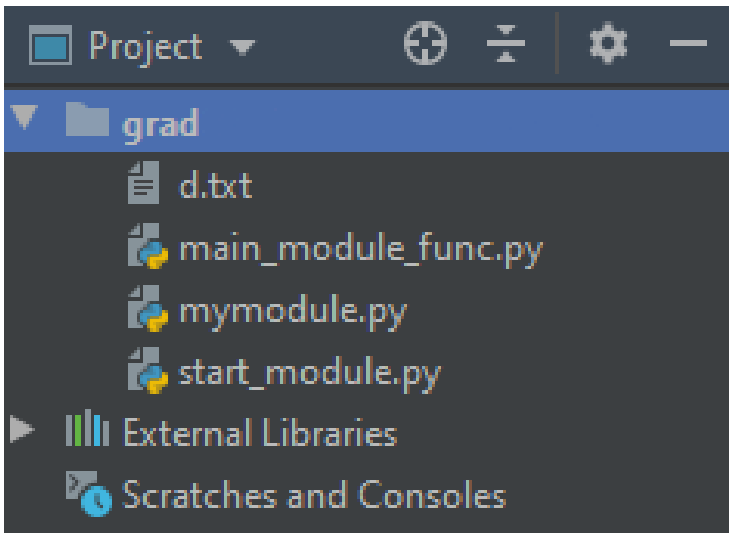

Рис. 3.2. Файлова структура

ВИСНОВКИ

У ході виконання роботи була досягнута поставлена мета: реалізований алгоритм, та створений програмний засіб для знаходження глобального оптимуму деякої цільової функції f в заданому n-мірному просторі.

Мета була досягнута виконанням наступних завдань:

1) були опановані чисельні методи, зокрема методи оптимізації, що застосовуються у нейронних мережах та задачах штучного інтелекту;
2) був реалізований метод оптимізації, з застосуванням паралельних операцій що дозволяє пошук глобального екстремуму у заданій області;
3) шляхом чисельних експериментів був обраний варіант цього методу, найбільш зручний для застосування в нейронних мережах;
4) був реалізований обраний варіант методу в вигляді програмного засобу на мові програмування Python.

Як результат – програмний засіб, а також алгоритм, що можуть бути застосовані в реальних задачах оптимізації.

В подальшому планується випробування алгоритму та програмного засобу в задачах штучного інтелекту.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1) The 5 Best Python IDE's and Code Editors for 2019 [Електронний ресурс] – Режим доступу: https://www.pythonforbeginners.com/development/5-best-python-ides-and-code-editors-2019
2) A Beginner's Guide to Neural Networks and Deep Learning [Електронний ресурс] – Режим доступу: https://pathmind.com/wiki/neural-network
3) Global Optimization Test Problems [Електронний ресурс] – Режим доступу: https://www.mat.univie.ac.at/~neum/glopt/test.html
4) Various Optimization Algorithms For Training Neural Network [Електронний ресурс] – Режим доступу: https://medium.com/@sdoshi579/optimizers-for-training-neural-network-59450d71caf6
5) Python Tutorial [Електронний ресурс] – Режим доступу: https://www.w3schools.com/python/
6) Data buffer [Електронний ресурс] – Режим доступу: https://en.wikipedia.org/wiki/Data_buffer
7) NumPy Documentation [Електронний ресурс] – Режим доступу: https://numpy.org/doc/
8) Машинное обучение — это легко [Електронний ресурс] – Режим доступу: https://habr.com/ru/post/319288/
9) Why is Python Used for Machine Learning? [Електронний ресурс] – Режим доступу: https://hackernoon.com/why-python-used-for-machine-learning-u13f922ug
10) How to execute a dynamic code in Python [Електронний ресурс] – Режим доступу: https://clay-atlas.com/us/blog/2019/08/20/python-english-dynamic-code-exec/
11) Optimization Test Problems [Електронний ресурс] – Режим доступу: https://www.sfu.ca/~ssurjano/optimization.html

12) A Comprehensive Guide to Types of Neural Networks [Електронний ресурс] – Режим доступу: https://www.digitalvidya.com/blog/types-of-neural-networks/
13) multiprocessing — Process-based parallelism [Електронний ресурс] – Режим доступу: https://docs.python.org/3/library/multiprocessing.html
14) time — Time access and conversions [Електронний ресурс] – Режим доступу: https://docs.python.org/3/library/time.html
15) Методы оптимизации нейронных сетей [Електронний ресурс] – Режим доступу: https://habr.com/ru/post/318970/
16) Python (programming language) [Електронний ресурс] – Режим доступу: https://en.wikipedia.org/wiki/Python_(programming_language)
17) Multiprocessing Vs. Threading In Python: What You Need To Know. [Електронний ресурс] – Режим доступу: https://timber.io/blog/multiprocessing-vs-multithreading-in-python-what-you-need-to-know/
18) Mathematical optimization [Електронний ресурс] – Режим доступу: https://en.wikipedia.org/wiki/Mathematical_optimization
19) Maxima, minima, and saddle points [Електронний ресурс] – Режим доступу: https://www.khanacademy.org/math/multivariable-calculus/applications-of-multivariable-derivatives/optimizing-multivariable-functions/a/maximums-minimums-and-saddle-points
20) Python/C API Reference Manual [Електронний ресурс] – Режим доступу: https://docs.python.org/3/c-api/index.html
21) Gradient descent [Електронний ресурс] – Режим доступу: https://en.wikipedia.org/wiki/Gradient_descent
22) Numpy and Scipy Documentation [Електронний ресурс] – Режим доступу: https://docs.scipy.org/doc/
23) Test functions for optimization [Електронний ресурс] – Режим доступу: https://en.wikipedia.org/wiki/Test_functions_for_optimization

24) Speed Up Your Python Program With Concurrency [Електронний ресурс] – Режим доступу: https://realpython.com/python-concurrency/
25) A hitchhiker guide to python NumPy Arrays [Електронний ресурс] – Режим доступу: https://towardsdatascience.com/a-hitchhiker-guide-to-python-numpy-arrays-9358de570121
26) The 3 Best Optimization Methods in Neural Networks [Електронний ресурс] – Режим доступу: https://towardsdatascience.com/the-3-best-optimization-methods-in-neural-networks-40879c887873
27) PyCharm [Електронний ресурс] – Режим доступу: https://en.wikipedia.org/wiki/PyCharm
28) Artificial neural network [Електронний ресурс] – Режим доступу: https://en.wikipedia.org/wiki/Artificial_neural_network
29) Things I Wish They Told Me About Multiprocessing in Python [Електронний ресурс] – Режим доступу: https://www.cloudcity.io/blog/2019/02/27/things-i-wish-they-told-me-about-multiprocessing-in-python/
30) Python 3.8.3rc1 documentation [Електронний ресурс] – Режим доступу: https://docs.python.org/3/
31) Global optimization [Електронний ресурс] – Режим доступу: https://en.wikipedia.org/wiki/Global_optimization
32) Review of second-order optimization techniques in artificial neural networks backpropagation [Електронний ресурс] – Режим доступу: https://iopscience.iop.org/article/10.1088/1757-899X/495/1/012003/pdf

ДОДАТКИ

Додаток А

```
import numpy as np
import multiprocessing as mp
import time
import math

import mymodule as mm


def f(p):
    return math.sin(p[0])/(3*p[0])


def get_grad(p):
    grad = np.array([(p[0]*math.cos(p[0]) - math.sin(p[0]))/(3*p[0]**2)])
    return grad


def get_dist(p1, p2):
    dist = math.sqrt(np.sum(np.subtract(p1, p2)**2))
    return dist


def calc_edges_len(ranges, div):
    edges_len = []
    for i in ranges:
        edges_len.append((i[1] - i[0])/div)
    return edges_len


def pr_cells(content_l, cells, shape, edges_len, dim, sec_div, l_rate, instr,
rdycntr, ctpr, pid):
    np_cells = np.frombuffer(cells, 'f').reshape((shape, dim))
    np_content_l = np.frombuffer(content_l, 'b')
    self_hit = set()
    flag1 = True
    flag2 = True
    while True:
        if instr.value == 0:
            flag1 = True
            flag2 = True
            time.sleep(0.005)
        elif (instr.value == 1) and flag1:
            for i in ctpr[pid]:
                points = mm.br_cell(np_cells[i], edges_len, dim, sec_div)
                grads = [get_grad(x)*l_rate for x in points]
                points = np.subtract(points, grads)
                # points = np.add(points, grads)
                for j in points:
                    is_in = False
                    u_bound = shape
                    l_bound = 0
                    while l_bound != (u_bound - 1):
                        temp = u_bound - (u_bound - l_bound)/2
                        if np.less(np_cells[math.floor(temp)], j).prod():
```

```
                            l_bound = math.ceil(temp)
                        else:
                            u_bound = math.floor(temp)
                    if np.less(np_cells[l_bound], j).prod():
                        np_content_l[l_bound] = 1
                        is_in = True
                        if (l_bound == i) and (l_bound != 0) and (l_bound !=
(shape - 1)):
                            set(self_hit).add(i)
                    if not is_in:
                        np_content_l[i] = 1
            flag1 = False
            with rdycntr.get_lock():
                rdycntr.value += 1
        elif (instr.value == 2) and flag2:
            for i in self_hit:
                img = np.subtract(np_cells[i], get_grad(np_cells[i])*l_rate)
                # img = np.add(np_cells[i], get_grad(np_cells[i])*l_rate)
                st_dist = get_dist(img, np_cells[i])
                for j in range(0, 1):
                    img = np.subtract(img, get_grad(img)*l_rate)
                    # img = np.add(img, get_grad(img)*l_rate)
                if st_dist < get_dist(img, np_cells[i]):
                    np_content_l[i] = 0
            flag2 = False
            self_hit = []
            with rdycntr.get_lock():
                rdycntr.value += 1
        elif instr.value == 3:
            break
        else:
            time.sleep(0.005)


def find_attractors(content_l, cells, edges_len, dim, sec_div, l_rate, instr,
rdycntr, ctpr):
    shape = len(content_l)
    previous_state = np.zeros(shape=(1, shape), dtype=np.bool)
    numofpr = mp.cpu_count()
    np_content_l = np.frombuffer(content_l, 'b')
    with open(FILENAME, 'a') as file:
        file.write(' '.join([str(x) for x in np_content_l.tolist()]) + "\n")
    prcs = [mp.Process(target=pr_cells, args=(content_l, cells, shape,
edges_len, dim, sec_div, l_rate, instr, rdycntr, ctpr, i)) for i in range(0,
numofpr)]
    for i in range(0, numofpr):
        prcs[i].start()
    while not np.array_equal(np_content_l, previous_state):
        instr.value = 0
        time.sleep(0.01)
        previous_state = np.copy(np_content_l)
        distrib = [[] for i in range(0, numofpr)]
        cur_p_num = 0
        for i in range(0, shape):
            if np_content_l[i] == 1:
                distrib[cur_p_num].append(i)
                cur_p_num += 1
                if cur_p_num == numofpr:
                    cur_p_num = 0
        ctpr[:] = distrib
        np_content_l[:] = np.zeros(shape=(1, shape), dtype=np.bool).ravel()
        instr.value = 1
```

```
        while rdycntr.value < numofpr:
            time.sleep(0.005)
        rdycntr.value = 0
        instr.value = 2
        while rdycntr.value < numofpr:
            time.sleep(0.005)
        rdycntr.value = 0
        with open(FILENAME, 'a') as file:
            file.write(' '.join([str(x) for x in np_content_l.tolist()]) +
"\n")
    instr.value = 3
    for i in range(0, numofpr):
        prcs[i].join()


def find_optim(content_l, cells, edges_len, dim, sec_div, sub_div, l_rate, lim,
instr, rdycntr, ctpr):
    vol1 = len(content_l)
    find_attractors(content_l, cells, edges_len, dim, sec_div, l_rate, instr,
rdycntr, ctpr)
    vol2 = np.sum(content_l)
    e = 0
    while e < lim:
        with open(FILENAME, 'a') as file:
            file.write("div\n")
        # l_rate /= 1.25
        np_cells = np.frombuffer(cells, 'f').reshape((vol1, dim))
        edges_len = [i/sub_div for i in edges_len]
        new_cells = [np_cells[i] for i in range(0, vol1) if content_l[i]]
        new_cells = mm.br_cells(new_cells, vol2, edges_len, dim, sub_div)
        vol1 = vol2*sub_div**dim
        cells = mp.RawArray('f', int(vol1*dim))
        cells_buf = np.frombuffer(cells, 'f').reshape((vol1, dim))
        cells_buf[:] = new_cells
        content_l = mp.RawArray('b', int(vol1))
        content_l_buf = np.frombuffer(content_l, 'b')
        content_l_buf[:] = np.ones(shape=(1, vol1), dtype=np.bool).ravel()
        find_attractors(content_l, cells, edges_len, dim, sec_div, l_rate,
instr, rdycntr, ctpr)
        vol2 = np.sum(content_l)
        e += 1
    np_cells = np.frombuffer(cells, 'f').reshape((vol1, dim))
    l_cells = [np_cells[x] for x in range(0, len(np_cells)) if content_l[x] ==
1]
    return l_cells


if __name__ == "__main__":
    r = [[0.1, 20]]
    l_rate = 2
    lim = 8
    FILENAME = "a.txt"
    FILENAME2 = "d.txt"
    with open(FILENAME, 'w'):
        pass
    with open(FILENAME2, 'r') as file:
        s = [int(x) for x in file.readline().split(' ')]
    dim = s[0]
    div = s[1]
    sec_div = s[2]
    sub_div = s[3]
    e_l = calc_edges_len(r, div)
```

```
    cells = mp.RawArray('f', div ** dim * dim)
    cells_buf = np.frombuffer(cells, 'f').reshape((div ** dim, dim))
    cells_buf[:] = mm.br_field(r, e_l, dim, div)
    content_l = mp.RawArray('b', div ** dim)
    content_l_buf = np.frombuffer(content_l, 'b')
    content_l_buf[:] = np.ones(shape=(1, div ** dim), dtype=np.bool).ravel()
    instr = mp.RawValue('i', 0)
    rdycntr = mp.Value('i', 0)
    ctpr = mp.Manager().list()
    start_time = time.time()
    cells = find_optim(content_l, cells, e_l, dim, sec_div, sub_div, l_rate,
lim, instr, rdycntr, ctpr)
    res = [f(x) for x in cells]
    print([x.tolist() for x in cells])
    print(res)
    print(min(res))
    print("time taken - ", time.time() - start_time)
```

Додаток Б

```
def create_code_1(dim, div):
    tabs = 1
    code = "def br_field(ranges, edges_len, dim, div):\n\tcells =
np.empty(shape=(div**dim, dim), " \
           "dtype=np.single)\n\tcurr_cell = 0\n\t"
    for i in range(0, dim):
        tabs += 1
        code = ''.join([code, "for p", str(i), " in range(0, ", str(div),
"):\n", "\t"*tabs])
    code += "cells[curr_cell] = ["
    for i in range(0, dim):
        code = ''.join([code, "ranges[", str(i), "][0] + edges_len[", str(i),
"]*p", str(i), ", "])
    code = ''.join([code, "]\n", "\t"*tabs, "curr_cell += 1\n\treturn cells"])
    return code


def create_code_2(dim, sec_div):
    tabs = 1
    code = "def br_cell(st_point, edges_len, dim, sec_div):\n\tpoints =
np.empty(shape=(sec_div**dim, dim), " \
           "dtype=np.single)\n\tcurr_point = 0\n\t"
    for i in range(0, dim):
        tabs += 1
        code = ''.join([code, "for p", str(i), " in range(0, ", str(sec_div),
"):\n", "\t"*tabs])
    code += "points[curr_point] = ["
    for i in range(0, dim):
        code = ''.join([code, "st_point[", str(i), "] + edges_len[",  str(i),
"]*p", str(i), ", "])
    code = ''.join([code, "]\n", "\t"*tabs, "curr_point += 1\n\treturn
points"])
    return code


def create_code_3(dim, sub_div):
    tabs = 2
    code = "def br_cells(cells, shape, edges_len, dim, sub_div):\n\tnew_cells =
np.empty(shape=(shape*(sub_div**dim), " \
           "dim), dtype=np.single)\n\tcurr_cell = 0\n\tfor i in cells:\n\t\t"
    for i in range(0, dim):
        tabs += 1
        code = ''.join([code, "for p", str(i), " in range(0, ", str(sub_div),
"):\n", "\t"*tabs])
    code += "new_cells[curr_cell] = ["
    for i in range(0, dim):
        code = ''.join([code, "i[", str(i), "] + edges_len[", str(i), "]*p",
str(i), ", "])
    code = ''.join([code, "]\n", "\t"*tabs, "curr_cell += 1\n\treturn
new_cells"])
    return code


DIM = 1
DIV = 400
SEC_DIV = 1
SUB_DIV = 2
```

```
if __name__ == "__main__":
    with open("mymodule.py", "w") as f:
        f.write(''.join(["import numpy as np", "\n\n\n", create_code_1(DIM,
DIV), "\n\n\n", create_code_2(DIM, SEC_DIV), "\n\n\n", create_code_3(DIM,
SUB_DIV)]))
    with open("d.txt", "w") as f:
        f.write(' '.join([str(DIM), str(DIV), str(SEC_DIV), str(SUB_DIV)]))
```

Додаток В

```
import numpy as np


def br_field(ranges, edges_len, dim, div):
   cells = np.empty(shape=(div**dim, dim), dtype=np.single)
   curr_cell = 0
   for p0 in range(0, 200):
      cells[curr_cell] = [ranges[0][0] + edges_len[0]*p0, ]
      curr_cell += 1
   return cells


def br_cell(st_point, edges_len, dim, sec_div):
   points = np.empty(shape=(sec_div**dim, dim), dtype=np.single)
   curr_point = 0
   for p0 in range(0, 1):
      points[curr_point] = [st_point[0] + edges_len[0]*p0, ]
      curr_point += 1
   return points


def br_cells(cells, shape, edges_len, dim, sub_div):
   new_cells = np.empty(shape=(shape*(sub_div**dim), dim), dtype=np.single)
   curr_cell = 0
   for i in cells:
      for p0 in range(0, 2):
         new_cells[curr_cell] = [i[0] + edges_len[0]*p0, ]
         curr_cell += 1
   return new_cells
```

Додаток Г

КОДЕКС АКАДЕМІЧНОЇ ДОБРОЧЕСНОСТІ
ЗДОБУВАЧА ВИЩОЇ ОСВІТИ ХЕРСОНСЬКОГО
ДЕРЖАВНОГО УНІВЕРСИТЕТУ

Я, Коломієць Олег Едуардович,
учасник(ця) освітнього процесу Херсонського державного університету, **УСВІДОМЛЮЮ**, що академічна доброчесність – це фундаментальна етична цінність усієї академічної спільноти світу.

ЗАЯВЛЯЮ, що у своїй освітній і науковій діяльності **ЗОБОВ'ЯЗУЮСЯ**:

– дотримуватися:
- вимог законодавства України та внутрішніх нормативних документів університету, зокрема Статуту Університету;
- принципів та правил академічної доброчесності;
- нульової толерантності до академічного плагіату;
- моральних норм та правил етичної поведінки;
- толерантного ставлення до інших;
- дотримуватися високого рівня культури спілкування;

– надавати згоду на:
- безпосередню перевірку курсових, кваліфікаційних робіт тощо на ознаки наявності академічного плагіату за допомогою спеціалізованих програмних продуктів;
- оброблення, збереження й розміщення кваліфікаційних робіт у відкритому доступі в інституційному репозитарії;
- використання робіт для перевірки на ознаки наявності академічного плагіату в інших роботах виключно з метою виявлення можливих ознак академічного плагіату;

– самостійно виконувати навчальні завдання, завдання поточного й підсумкового контролю результатів навчання;

– надавати достовірну інформацію щодо результатів власної навчальної (наукової, творчої) діяльності, використаних методик досліджень та джерел інформації;

– не використовувати результати досліджень інших авторів без використання покликань на їхню роботу;

– своєю діяльністю сприяти збереженню та примноженню традицій університету, формуванню його позитивного іміджу;

– не чинити правопорушень і не сприяти їхньому скоєнню іншими особами;

– підтримувати атмосферу довіри, взаємної відповідальності та співпраці в освітньому середовищі;

– поважати честь, гідність та особисту недоторканність особи, незважаючи на її стать, вік, матеріальний стан, соціальне становище, расову належність, релігійні й політичні переконання;

– не дискримінувати людей на підставі академічного статусу, а також за національною, расовою, статевою чи іншою належністю;

– відповідально ставитися до своїх обов'язків, вчасно та сумлінно виконувати необхідні навчальні та науково-дослідницькі завдання;

– запобігати виникненню у своїй діяльності конфлікту інтересів, зокрема не використовувати службових і родинних зв'язків з метою отримання нечесної переваги в навчальній, науковій і трудовій діяльності;

– не брати участі в будь-якій діяльності, пов'язаній із обманом, нечесністю, списуванням, фабрикацією;

– не підроблювати документи;

– не поширювати неправдиву та компрометуючу інформацію про інших здобувачів вищої освіти, викладачів і співробітників;

– не отримувати і не пропонувати винагород за несправедливе отримання будь-яких переваг або здійснення впливу на зміну отриманої академічної оцінки;

– не залякувати й не проявляти агресії та насильства проти інших, сексуальні домагання;

– не завдавати шкоди матеріальним цінностям, матеріально-технічній базі університету та особистій власності інших студентів та/або працівників;

– не використовувати без дозволу ректорату (деканату) символіки університету в заходах, не пов'язаних з діяльністю університету;

– не здійснювати і не заохочувати будь-яких спроб, спрямованих на те, щоб за допомогою нечесних і негідних методів досягати власних корисних цілей;

– не завдавати загрози власному здоров'ю або безпеці іншим студентам та/або працівникам.

УСВІДОМЛЮЮ, що відповідно до чинного законодавства у разі недотримання Кодексу академічної доброчесності буду нести академічну та/або інші види відповідальності й до мене можуть бути застосовані заходи дисциплінарного характеру за порушення принципів академічної доброчесності.

09.05.2020 (дата) | (підпис) | Олег Коломієць (ім'я, прізвище)