Міністерство освіти і науки України

Харківський національний університет імені В. Н. Каразіна

Факультет комп'ютерних наук

Кафедра теоретичної та прикладної системотехніки

«Затверджую»
Зав. кафедри теоретичної та
прикладної системотехніки
_____д.т.н., проф. С. І. Шматков
«___» грудня 2022 р

Пояснювальна записка

до кваліфікаційної роботи
магістра

на тему: «МЕТОДИ НАВЧАННЯ РЕКУРЕНТНОЇ НЕЙРОННОЇ МЕРЕЖІ ДЛЯ ВИРІШЕННЯ ЗАДАЧІ РОЗПІЗНАВАННЯ ОБРАЗІВ»

Захищено на засіданні
Атестаційної комісії № 45
протокол № __ від __.12.2022 р.
Оцінка _____ / _____
Голова Атестаційної комісії
_____ МІНУХІН С.В.
(підпис) (прізвище та ініціали)

Виконав:
студентка 2 курсу, групи КУ– 61
Галузь знань: 15 – Автоматизація та приладобудування
за спеціальністю 151 – Автоматизація та комп'ютерно-інтегровані технології.
КОНДРАТЮК Діана Сергіївна_____

Керівник: д.т.н., професор, завідувач кафедри теоретичної та прикладної системотехніки
ШМАТКОВ Сергій Ігоревич_____

Рецензент: к.ф.-м.н., доцент, в.о. зав. кафедри електроніки і управляючих систем
ХРУСЛОВ Максим Михайлович_____

Харків – 2022

АНОТАЦІЯ

Пояснювальна записка до магістерської атестаційної роботи складається зі вступу, трьох розділів, висновків, списку використаних джерел і трьох додатків. Загальний обсяг роботи складає 76 сторінок, із яких 51 сторінка основної частини з 25 рисунками, 9 формулами, 31 найменуванням списку використаних джерел та чотирма додатками.

Метою кваліфікаційної роботи є аналіз методів навчання рекурентної нейронної мережі та розробка програмної моделі рекурентної нейронної мережі для вирішення задачі розпізнавання образів на мові програмування Python з використанням хмарової платформи Google Colaboratory.

Об'єкт дослідження – процес навчання рекурентної нейронної мережі для розпізнавання образів.

Предмет дослідження – моделі та алгоритми навчання рекурентної нейронної мережі на основі популяційного алгоритму для розпізнавання образів.

Методи дослідження – методи побудови нейронних мереж, методи розпізнавання образів, основи популяційних алгоритмів, методи комбінування нейронних мереж та популяційних алгоритмів.

Область застосування – використання нейронної мережі у цілях розпізнавання образів. Розроблений програмний продукт може широко використовуватися в розпізнаванні зображень, проходження капчі і так далі.

Ключові слова: методи навчання, рекурентна нейронна мережа, розпізнавання, образи, Python, Google Colaboratory, програмна модель.

ABSTRACT

The explanatory note to the master's attestation work consists of an introduction, three sections, conclusions, a list of used sources and three appendices. The total volume of the work is 76 pages, of which 51 pages are the main part with 25 figures, 9 formulas, 31 names of the list of used sources and three appendices.

The purpose of the qualification work is the analysis of recurrent neural network training methods and the development of a recurrent neural network software model to solve the problem of pattern recognition in the Python programming language using the Google Colaboratory cloud platform.

The object of research is the process of training a recurrent neural network for pattern recognition.

The subject of the research is models and algorithms for training recurrent neural networks based on a population algorithm for pattern recognition.

Research methods – methods of building neural networks, pattern recognition methods, basics of population algorithms, methods of combining neural networks and population algorithms.

The field of application is the use of a neural network for the purposes of pattern recognition. The developed software product can be widely used in image recognition, captcha passing and so on.

Keywords: learning methods, recurrent neural network, recognition, images, Python, Google Colaboratory, software model.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ І УМОВНИХ ПОЗНАЧЕНЬ 6

ВСТУП 7

РОЗДІЛ 1. АНАЛІЗ МЕТОДІВ НАВЧАННЯ РЕКУРЕНТНОЇ НЕЙРОННОЇ МЕРЕЖІ ДЛЯ ВИРІШЕННЯ ЗАДАЧІ РОЗПІЗНАВАННЯ ОБРАЗІВ 9

1.1 Аналіз задачі розпізнавання образів 9

1.1.1 Технології розпізнавання образів 9

1.1.2 Аналіз методів, що використовуються для вирішення задачі розпізнавання образів 10

1.2 Аналіз нейронних мереж 17

1.2.1 Нейронні мережі та принципи її побудови 17

1.2.2 Архітектура нейронних мереж 18

1.3 Аналіз методів навчання нейронних мереж 20

Висновок до розділу 1 30

РОЗДІЛ 2. РОЗРОБКА МОДЕЛІ РЕКУРЕНТНОЇ НЕЙРОННОЇ МЕРЕЖІ НА ОСНОВІ ПОПУЛЯЦІЙНОГО АЛГОРИТМУ ДЛЯ РОЗПІЗНАВАННЯ ОБРАЗІВ 32

2.1 Опис вхідних даних 32

2.2 Аналіз структури та методів навчання рекурентної нейронної мережі 33

2.2.1 Аналіз структури рекурентної нейронної мережі 33

2.2.2 Стандартний метод навчання рекурентної нейронної мережі 34

2.3 Аналіз алгоритму рою часток 37

2.3.1 Алгоритм рою часток 37

2.3.2 Навчання рекурентної нейронної мережі на основі рою часток 38

Висновок до розділу 2 40

РОЗДІЛ 3. ПРОГРАМНА РЕАЛІЗАЦІЯ МОДЕЛІ РЕКУРЕНТНОЇ НЕЙРОННОЇ МЕРЕЖІ НА ОСНОВІ ПОПУЛЯЦІЙНОГО АЛГОРИТМУ ДЛЯ РОЗПІЗНАВАННЯ ОБРАЗІВ 42

3.1 Інструментальні засоби реалізації моделі 42
3.2 Програмна реалізація моделі 43
3.3 Дослідження моделі рекурентної нейронної мережі для вирішення задачі розпізнавання 50
Висновок до розділу 3 52
ВИСНОВОК 54
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 56
ДОДАТОК А 60
ДОДАТОК Б 63
ДОДАТОК В 66
ДОДАТОК Г 69

ПЕРЕЛІК СКОРОЧЕНЬ І УМОВНИХ ПОЗНАЧЕНЬ

ШНМ – штучна нейронна мережа;

РНМ – рекурентна нейронна мережа;

ЛДА – лінійний дискримінантний аналіз;

PSO – метод оптимізації роєм часток (Particle swarm optimization).

ВСТУП

В даний час з розвитком обчислювальних систем та інформаційних технологій зростає популярність систем автоматизації та роботизації, як у промисловості та науці, так і у повсякденному житті. Як наслідок зростає потреба в ефективних методах обробки інформації, що надходить. Зокрема завдання такого виду відноситься до завдання розпізнавання образів, що є одним з найважливіших і найважчих завдань аналізу даних.

Розпізнавання образів виникло з урахуванням теоретичної інформатики, прикладної статистики, штучного інтелекту та суміжних дисциплін. Методи розпізнавання образів застосовуються практично у всіх галузях людської діяльності: маркетингу, психології, менеджменту, медицини, хімії та інших. Існує безліч різних завдань розпізнавання образів: розпізнавання символів, мовлення, зображень, автомобільних номерів, штрих-кодів та інших. У роботі вирішується завдання розпізнавання зображень за допомогою рекурентної нейронної мережі.

Актуальність роботи. З розвитком інформаційних систем постійно з'являються нові методи класифікації, машинного навчання та машинного зору, які вирішують завдання розпізнавання образів. У цій роботі вирішується практичний приклад задачі розпізнавання зображень – завдання розпізнавання об'єктів на зображенні.

Метою дослідження є аналіз методів навчання рекурентної нейронної мережі та розробка програмної моделі рекурентної нейронної мережі для вирішення задачі розпізнавання образів на мові програмування Python з використанням хмарової платформи Google Colaboratory.

Об'єкт дослідження – процес навчання рекурентної нейронної мережі для розпізнавання образів.

Методи дослідження: методи навчання нейронних мереж, методи побудови популяційних алгоритмів, методи розпізнавання образів.

Предмет дослідження – моделі та алгоритми навчання рекурентної нейронної мережі на основі популяційного алгоритму для розпізнавання образів.

Завдання дослідження:

1. Виконати аналіз задачі розпізнавання образів.
2. Виконати аналіз методів, що використовуються для вирішення задачі розпізнавання образів.
3. Виконати аналіз нейронних мереж.
4. Виконати аналіз методів навчання нейронних мереж.
5. Розробити програмну модель рекурентної нейронної мережі на основі популяційного алгоритму для розпізнавання образів.

РОЗДІЛ 1
АНАЛІЗ МЕТОДІВ НАВЧАННЯ РЕКУРЕНТНОЇ НЕЙРОННОЇ МЕРЕЖІ ДЛЯ ВИРІШЕННЯ ЗАДАЧІ РОЗПІЗНАВАННЯ ОБРАЗІВ

1.1 Аналіз задачі розпізнавання образів

1.1.1 Технології розпізнавання образів

Розпізнавання зображень, підкатегорія комп'ютерного зору та штучного інтелекту, являє собою набір методів виявлення та аналізу зображень для автоматизації конкретного завдання. Це технологія, яка здатна ідентифікувати місця, людей, предмети та багато інших типів елементів у зображенні та робити з них висновки, аналізуючи їх [1].

Розпізнавання зображень досягається шляхом диференціації зображення на встановлену категорію на основі змісту бачення. Розпізнавання зображень є одним із напрямів машинного навчання. Машинне навчання складається з модуля вилучення особливостей, який вилучає такі важливі ознаки, як ребра, текстури тощо, та модуля класифікації, який класифікує на основі вилучених особливостей. Основне обмеження машинного навчання полягає в тому, що, відокремлюючи, воно може витягувати лише певний набір ознак на зображеннях і не в змозі виділити диференційовані ознаки з навчального набору даних. Цей недолік виправляється за допомогою глибокого навчання [2].

Різні види класифікаторів мають свої переваги і недоліки. Так, класифікатори, в яких використовуються методи статистики мають хорошу математичну обґрунтованість, але при цьому складні у використанні і вимагають знання ймовірного розподілу вихідних даних і оцінки його параметрів (тому їх називають параметричними), а також мають фіксовану структуру моделі. Крім цього, статистичні методи оцінюють тільки ймовірність приналежності об'єкта класу, але не «пояснюють» чому.

Класифікатори, засновані на машинному навчанні не вимагають оцінки параметрів розподілу вихідних даних, а міра подібності в них формалізується за допомогою функції відстані. Такі класифікатори називаються метричними. Як правило, вони простіше в реалізації і використанні, ніж параметричні, а їх результати зручніше для інтерпретації і розуміння. Але при цьому метричні
класифікатори є наближеними моделями – забезпечують рішення тільки в обмеженому числі практично значущих випадків, можуть дати неточне або не єдине рішення. Тому використовувати їх результати потрібно з певною часткою обережності.

Певним компромісом між параметричних і метричними методами є використання для вирішення завдань класифікації штучних нейронних мереж. Дійсно, нейронні мережі є непараметричними моделями, що не вимагають припущень про імовірнісний розподіл даних, але при цьому і не використовують заходи відстаней. Це робить їх універсальними класифікаторами, дозволяючи отримувати результати [3] навіть у випадках, коли параметричні і метричні класифікатори не забезпечує прийнятного рішення.

1.1.2 Аналіз методів, що використовуються для вирішення задачі розпізнавання образів

До числа поширених методів рішення задачі класифікації відносяться:

1. Логістична регресія.
2. Дерева рішень.
3. Метод найближчого сусіда.
4. Машини опорних векторів.
5. Дискримінантний аналіз.

6. Нейронні мережі.

Проведемо аналіз кожного методу для виявлення їх переваг та недоліків.

1. Логістична регресія.

У математичній статистиці логістична регресія – це широко використовувана статистична модель, яка використовує логістичну функцію для моделювання залежності вихідної змінної від ряду входів, коли перша є двійковою.

Це форма множинної регресії, загальною метою якої є аналіз взаємозв'язку між кількома незалежними змінними (також відомими як регресори або предиктори) та залежною змінною. Регресія в загальному вигляді застосовується, коли вхідні і вихідні змінні безперервні. Логістична регресія найкраща, коли вихідна змінна приймає лише два значення.

Важливість логістичної регресії випливає з того, що багато завдань аналізу даних можна вирішити або звести до використання двійкової класифікації [4].

Наприклад, логістична регресія може бути використана для оцінки ймовірності настання (або ненастання) певної події: пацієнт хворий (здоровий), позичальник повернув позику (затримка) тощо. Завдяки цьому логістичну регресію можна розглядати як потужний інструмент підтримки прийняття рішень.

Як відомо, усі моделі регресії можна записати у вигляді формули (1.1):

$$y = F(x_1, x_2, \dots, x_n) \tag{1.1}$$

Наприклад, якщо розглядається результат позики, змінна y встановлюється зі значеннями 1 і 0, де 1 означає, що відповідний позичальник повернув позику, а 0 означає, що не повернув.

Однак це створює проблему: множинна регресія не "знає", що відповідь носить бінарний характер. Це неминуче призводить до моделі із передбачуваними значеннями більше 1 і менше 0. Але такі значення взагалі не припустимі для початкової задачі. Таким чином, множинна регресія просто ігнорує обмеження діапазону для y. Тому це є проблемою даного методу.

2. Дерева рішень.

Дерево рішень – це класифікатор, який будується на основі правил "якщо, то", розташованих у вигляді деревоподібної ієрархічної структури.

В основі роботи дерева рішень лежить процес рекурсивного розбиття вихідної множини об'єктів на підмножини, які асоціюються з попередньо заданими класами. Розбиття проводиться за допомогою вирішальних правил, в яких здійснюється перевірка значень атрибутів по заданій умові.

Дерева рішень створюються на основі навчання з учителем. Багато спостережень використовуються як набір навчальних даних, для яких попередньо встановлена назва класу.

Дерево рішень є лінійним класифікатором, тобто виробляє розбиття об’єктів в багатовимірному просторі площинами (в двовимірному випадку – лініями) [5].

Широка популярність дерев рішень обумовлена наступними їх перевагами.

1. Правила в них формуються практично на природній мові, що робить пояснювальну здатність дерев рішень дуже високою.

2. Можуть працювати як з числовими, так і з категоріальним даними.
3. Вимагають відносно невеликої попередньої обробки даних, зокрема, не вимагають нормалізації, створення фіктивних змінних, можуть працювати з пропусками.
4. Можуть працювати з великими обсягами даних.

Разом з тим, деревам рішень притаманний ряд обмежень.

1. Нестійкість – навіть невеликі зміни в даних можуть привести до значних змін результатів класифікації.
2. Оскільки алгоритми побудови дерев рішень є жадібними, вони не гарантують побудови оптимального дерева.
3. Схильність до перенавчання.

3. Метод найближчого сусіда.

Для вирішення проблеми класифікації використовується метод k-найближчого сусіда. Він присвоює класу об'єкти, які включають більшість його k найближчих сусідів у багатовимірному просторі ознак. Це один з найпростіших алгоритмів для навчання моделей класифікації.

Число k – це кількість сусідніх об'єктів у просторі ознак, які порівнюються з об'єктом, що підлягає класифікації. Іншими словами, коли k = 10, кожен об'єкт порівнюється з 10-ма сусідами. Метод широко використовується в технологіях інтелектуального аналізу даних.

Під час навчання алгоритм просто запам'ятовує всі вектори функцій та відповідні назви класів. При роботі з реальними даними, тобто спостереженнями, назви класів яких невідомі, обчислюється відстань між вектором нового спостереження та раніше збереженими спостереженнями. Потім обираються k найближчі до нього вектори і новий об'єкт належить до класу, до якого належить більшість з них.

Вибір параметра k суперечливий. З одного боку, збільшення вартості збільшує надійність класифікації, водночас межі між класами стають менш чіткими. На практиці евристичні методи вибору параметра k, наприклад перехресна перевірка, дають хороші результати [6].

Незважаючи на відносну алгоритмічну простоту, метод показує хороші результати. Основним недоліком є висока обчислювальна складність, яка збільшується квадратично із збільшенням кількості прикладів навчання.

4. Машини опорних векторів.

Машини опорних векторів – це набір алгоритмів «навчання з учителем», які активно використовуються при вирішенні задачі класифікації. Основною ідеєю метода опорних векторів є переведення вихідних векторів у простір вищої розмірності та пошук максимально відокремлюючої гіперплощини в просторі.

Метод визначає межу прийняття рішення разом з максимальною відстанню, яка розділяє майже всі точки на два класи і залишає місце для неправильної класифікації.

Його перевага полягає в тому, що він може визначати як лінійні, так і нелінійні межі, використовуючи функції ядра. Тому він підходить для реальних задач, де дані не завжди повністю розділені прямою лінією [7].

Недоліки.

1. Тривалий час навчання (для великих обсягів даних).
2. Нестабільність щодо шуму: викиди в навчальних даних стають опорними об’єктами пошкодження та мають прямий вплив на конструкцію гіперплощини.
3. Не описані загальні методи побудови ядер і просторів, найбільш придатних для конкретного завдання в разі лінійної нероздільності класів.

5. Дискримінантний аналіз.

Лінійний дискримінантний аналіз (ЛДА), та пов'язаний з ним лінійний дискримінант Фішера – це статистичні методи та методи машинного навчання, які використовуються для пошуку лінійних комбінацій ознак, які найкраще відокремлюють два або більше класи об'єктів чи подій. Отриману комбінацію можна використовувати як лінійний класифікатор або для зменшення розмірності простору ознак перед подальшою класифікацією.

ЛДА тісно пов'язаний з аналізом дисперсії та регресійним аналізом, який також намагається виразити залежну змінну через лінійну комбінацію інших ознак або розмірів. В цих двох методах залежна змінна – числове значення, а в ЛДА – це номінальне значення (позначення класу). Крім того, ЛДА має схожість з методом головних компонентів та факторним аналізом, які шукають лінійні комбінації величин, які найкраще описують дані.

ЛДА реалізує дві тісно пов'язані між собою статистичні процедури.

1. Інтерпретацію міжгрупових відмінностей, коли потрібно відповісти на питання: наскільки добре набір змінних, що використовується, може формувати роздільну поверхню для об'єктів навчальної вибірки і які з цих змінних найбільш інформативні?
2. Класифікація, тобто прогнозування значення фактора групування для екзаменованої групи спостережень.

Для використання ЛДА, функції повинні мати безперервні значення. В іншому випадку слід використовувати аналіз відповідності [8].

6. Нейронні мережі.

Нейронні мережі успішно використовуються в різних сферах – бізнесі, медицині, техніці, геології, фізиці. Нейронні мережі застосовуються на

практиці скрізь, де необхідно вирішити задачі прогнозування, класифікації або управління.

Існує кілька причин цього вражаючого успіху.

1. Великі можливості. Нейронні мережі – надзвичайно потужна техніка моделювання, яка може відтворювати надзвичайно складні залежності. Зокрема, нейронні мережі мають нелінійний характер. Протягом багатьох років лінійне моделювання є основним методом моделювання в більшості областей завдяки добре розробленим методам оптимізації. Для задач, де лінійне наближення є незадовільним, лінійні моделі працюють погано. Крім того, нейронні мережі долають проблему *розмірності*, яке не дозволяє моделювати лінійні залежності з великою кількістю змінних.
2. Простота у використанні. Нейронні мережі навчаються на прикладах. Користувач нейронної мережі вибирає репрезентативні дані, а потім виконує алгоритм навчання, який автоматично сприймає структуру даних. У той же час, користувач повинен мати евристичні знання щодо вибору та підготовки даних, вибору бажаної архітектури мережі та інтерпретації результатів, але знання, необхідні для успішного застосування нейронних мереж, набагато скромніші, ніж, наприклад, при застосування звичайних статистичних методів [9].

Нейронні мережі інтуїтивно привабливі, оскільки вони можуть працювати з великим обсягом даних та мають кращі характеристики для розпізнавання образів ніж методи, які були розглянуті вище.

1.2 Аналіз нейронних мереж

1.2.1 Нейронні мережі та принципи її побудови

Штучна нейронна мережа (ШНМ) – це математична модель та її програмна чи апаратна реалізація, заснована на принципах організації та функціонування біологічної нейронної мережі – мережі нервових клітин у живому організмі.

ШНМ складаються зі штучних нейронів, кожен з яких є спрощеною моделлю біологічного нейрона. Усе, що робить штучний нейрон, – це отримує сигнал із багатьох входів, обробляє їх єдиним способом та передає результат на багатьом іншим штучним нейронам, тобто робить те саме, що й біологічний нейрон. Біологічні нейрони з'єднані між собою аксонами, місця з'єднання називаються синапсами. У синапсах електрохімічні сигнали посилюються або послаблюються. Зв'язки між штучними нейронами називають синаптичними, або просто синапсами. Синапс має один параметр – ваговий коефіцієнт, в залежності від його значення відбувається та чи інша зміна інформації, при передачі від одного нейрона до іншого. Завдяки цьому вхідна інформація обробляється і перетворюється в результат, а навчання ШНМ базується на експериментальному виборі таких вагових коефіцієнтів для кожного синапсу, що призводить до отримання необхідного результату [10].

Структура найпростішої нейронної мережі представлена на рисунку нижче (рис. 1.1). Нейрони вхідного шару позначені синім кольором, нейрони прихованого шару – рожевим, а нейрони вихідного шару – зеленим.

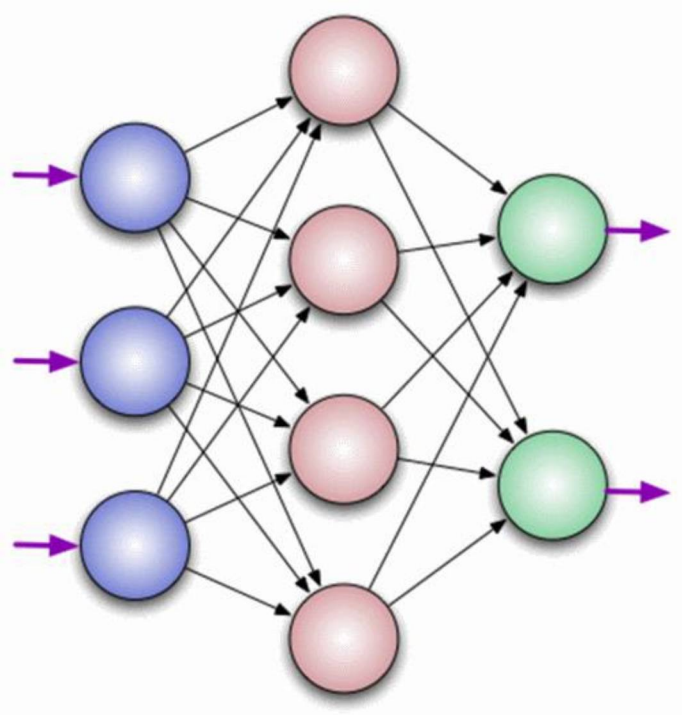

Рисунок 1.1 – Найпростіша нейронна мережа.

1.2.2 Архітектура нейронних мереж

Існує багато типів нейронних мереж.

Мережа прямого поширення – це нейронна мережа без зворотного зв'язку. В цій мережі поширення сигналу односпрямоване, тобто немає зворотного зв'язку. З вхідного шару сигнал обробляється шар за шаром у напрямку вихідного. Після кількох відомих кроків відповідь мережі відображається на вихідному рівні [11].

Мережі прямого поширення добре зрозумілі та відносно прості у реалізації. Недоліком є необхідність у великої кількості нейронів для вирішення складних завдань.

Ланцюги Маркова є свого роду попередниками машин Больцмана і мереж Хопфілда. У ланцюгах Маркова визначається ймовірність переходу з поточного стану в сусідній. Крім того, ці ланцюжки не мають пам'яті, тому

що наступний стан залежить лише від поточного і не залежить від усіх попередніх станів [12].

Мережа Хопфілда (модель Хопфілда) є різновидом мережі асоціативної пам'яті. Це одношарова ШНМ, в якій кожний нейрон з'єднаний з іншими та має вхід і вихід. Функція жорсткої активації повертає два значення: -1 (вимкнено) і +1 (увімкнено). Модель використовує принцип зберігання інформації як динамічно-стабільних атракторів. Енергетична функція зменшується в процесі навчання до тих пір, поки не досягне локального мінімуму (атрактора), де вона залишається постійною [13].

Машини Больцмана схожі на мережі Хопфілда, але вони мають деякі нейрони, позначені як вхідні, а інші, як приховані. Вхідні нейрони виводяться, коли всі нейрони в мережі оновлюють свій стан. Спочатку вагові коефіцієнти задаються випадковим чином, потім навчання виконується за допомогою методу зворотного поширення, або за допомогою алгоритму contrastive divergence (якщо градієнт обчислюється за допомогою ланцюга Маркова). Машини Больцмана є стохастичною нейронною мережею, оскільки у навчанні бере участь ланцюг Маркова.

Мережа Хемінга є ще одним прикладом асоціативної нейронної мережі для зберігання даних. Принцип роботи заснований на обчисленні відстані Хеммінга від вхідного вектора, до всіх векторів вибірки, які відомі мережі. Після отримання вхідного зображення мережа вибирає найближчий до нього зразок та активується відповідний вихід.

Мережа Кохонена – це одношарова мережа з регульованими вагами. Поки один нейрон збуджується, усі інші виходи шару пригнічуються. Вагові коефіцієнти налаштовані таким чином, що вхідні зображення одного класу запускають той самий вихідний нейрон. Так вхідні вектора класифікуються за подібними групами. Це відображає одну з найважливіших властивостей

мережі Кохонена: здатність до узагальнення. Вектор кожного з нейронів мережі замінює групу відповідних класифікованих векторів [14].

Рекурентні нейронні мережі (РНМ) – це тип нейронних мереж, що спеціалізується на обробці послідовностей. Найчастіше їх використовують у таких завданнях, як обробка природної мови (Natural Language Processing) через їхню ефективність в аналізі тексту [15].

Один із нюансів роботи з нейронними мережами полягає в тому, що вони працюють із попередньо заданими параметрами. Вони приймають вхідні дані з фіксованими розмірами та виводять результат, який також є фіксованим. Плюс рекурентних нейронних мереж у тому, що вони забезпечують послідовності з варіативними довжинами як для входу, так і для виходу.

Проаналізувавши види нейронних мереж, можна зробити висновок, що кожний вид спеціалізується в якійсь певній області. Тому, так як рекурентні нейронні мережі спеціалізуються на розпізнаванні тексту, можна провести експеримент, і перевірити, як рекурентна нейронна мережа впорається з задачею розпізнавання образів.

1.3 Аналіз методів навчання нейронних мереж

В даний час нейронні мережі використовують у таких напрямках.

1. Класифікаційний аналіз – поділ вступних даних за будь-якими ознаками. Наприклад, у медицині нейронна мережа полегшує завдання з діагностики: вік пацієнта та його стать, скарги на здоров'я, результати аналізу тощо – все це дозволяє розподілити хворих за ступенем тяжкості стану.

2. Прогнозування – з урахуванням показників можна спрогнозувати наступні події. Наприклад, каршеринг використовує нейронні мережі для виявлення агресивних водіїв, щоб надалі обмежити доступ до авто.

3. Розпізнавання образів – це найпопулярніша область для використання нейронних мереж: ідентифікація символів на папері та банківських картках, розпізнавання осіб для вирішення питань державної безпеки, пошук по картинці в Google та інше.

Машинне навчання нерозривно пов’язане з нейронною мережею і представляє собою роботу, при якій змодельоване середовище імітує процеси напрацювання досвіду людиною, поступово підвищуючи точність результатів.

Існують різні алгоритми навчання нейронних мереж. Проте вони підкоряються двом основним принципам: з учителем і без нього. Якщо проводити аналогію з навчанням людини, то він також здатний набувати досвіду або з наставником, який спрямовуватиме і вказуватиме правильну відповідь, або без неї, орієнтуючись лише на власні спостереження. Різниця між цими двома підходами у тому, що з одних «уроків» вчитель необхідний, а інших досить самостійного засвоєння матеріалу [16].

1. Процес навчання з учителем.

При такому процесі нейронні мережі пропонують вибірку навчальних прикладів. Дані подають «вхід» мережі, очікуючи отримати правильний «вихід», тобто. відповідь, яка дасть нейронна мережа після обробки всередині своєї структури. Результат порівнюють із еталонним, тобто. правильною відповіддю. Якщо нейронна мережа видає неправильне рішення, необхідно відкоригувати вагові коефіцієнти зв'язку і запустити процес заново, тим самим домагаючись зниження відсотка помилкових відповідей.

Навчальні приклади надходять у нейронну мережу у певній послідовності. Для кожної відповіді відбувається розрахунок помилки та підстроювання ваг. Все це відбувається до тих пір, поки неправильні

відповіді по всьому обсягу навчального матеріалу не ухвалять значення допустимих показників.

Такий тип навчання має відмінну рису – рівень помилкових відповідей, який з'ясовують шляхом порівняння запланованих показників із реальними. За допомогою багаторазового повторення процесу відбувається виявлення вартісної функції, тобто різниці між очікуваними та поточними результатами.

Навчання з учителем підходить для вирішення питань, у яких відомий потрібний результат. Наприклад, для класифікації зображень, розпізнавання звуків або голосу, прогнозування, функції апроксимації.

2. Процес навчання без вчителя.

Він передбачає наявність лише вступних даних. Алгоритми навчання нейронних мереж без вчителя коригують вагові коефіцієнти таким чином, щоб нейронна мережа могла з схожих за певним принципом даних на «вході» видати результат, що виявляє інші взаємозв'язки та закономірності між цими даними. У процесі навчання відбувається виділення параметрів, притаманних моделям навчального матеріалу, і подальше об'єднання цих моделей угруповання за подібними ознаками.

Дані, які надходять на "вхід", після обробки нейронною мережею складуться в ту чи іншу відповідь. Однак до навчання не можна передбачити, у якій формі ця відповідь надійде. Відповідно, сам процес навчання повинен зумовлювати трансформацію результату на зрозумілу форму. Як правило, можна легко відстежити, який взаємозв'язок задав даним нейронної мережі в процесі їхньої обробки.

Алгоритми навчання нейронних мереж без вчителя використовують дані без класифікації чи міток. Нейронна мережа сама вибудовує логічний ланцюжок і засвоює розуміння цих дій, орієнтуючись лише на вступні дані.

По суті, це повторює людське самонавчання: індивід, роблячи будь-які дії, робить висновки про правильність чи хибність рішення, орієнтуючись на наслідки.

Навчання без учителя застосовують для кластеризації, язикових моделей, виявлення аномалій, статистичних моделей [17].

Виділяють чотири основні види алгоритмів навчання нейронних мереж.

1. Метод зворотного розповсюдження.

Він є одним із основних способів навчання та містить у своїй основі алгоритм обчислення градієнтного спуску. Іншими словами, рухаючись вздовж градієнта, відбувається розрахунок локального максимуму та мінімуму функції.

Для кращого розуміння процесу необхідно перевести функцію у графік, який відображатиме залежність значень помилки від ваги синапсу. На отриманій кривій потрібно визначити точку з найменшим та найбільшим показником. У той же час необхідно графічно відобразити всі ваги і розрахувати для кожного з них глобальний мінімум.

Значення градієнта матиме векторну величину, яка дасть уявлення про напрям і крутість схилу. Пошук значення градієнта здійснюється шляхом обчислення похідної від функції необхідної точки. Така точка матиме значення ваги, розподілене випадковим чином. У ній слід проводити розрахунок градієнта та визначати спрямованість руху спуску. Обчислення необхідно проводити послідовно у всіх точках, поки не буде досягнуто локального мінімуму, що зупиняє подальший спуск.

Щоб подолати цей скрутний етап, потрібно задати таке значення для моменту, який дозволить пройти ділянку графіка і опинитися у потрібній точці. У разі недостатнього значення подолати опуклість не вдасться, а якщо

значення буде надто великим, то висока ймовірність «проскоку» глобального мінімуму.

На загальну швидкість навчання нейронної мережі впливає не тільки момент прискорення, а й ще одне значення, що є гіперпараметром, що визначається методом підбору.

Найбільш сприятливе поєднання значень неможливо знати заздалегідь. Воно виявляється в ході кількох навчань та коригування в потрібну сторону.

Самим методом навчання є процес, у якому надходять дані, які поширюються між нейронами за допомогою синапсів. Передача здійснюється доти, доки дані не досягнуть шару «виходу», трансформувавшись у відповідь. Ця операція зветься «передача вперед».

Як тільки відповідь отримана, відбувається розрахунок помилки, і відповідно до неї виконується зворотна передача. Мета такої дії – приведення синаптичних ваг до оптимальних значень під час руху від вихідного шару до вхідного.

Для такого алгоритму навчання нейронних мереж необхідно використовувати функції активації, що диференціюються. Це пов’язано з тим, що поширення у зворотному напрямку визначається різницею між відповідями, а також добутком між ним та похідною функцією від вхідного значення.

Для успішного навчання потрібно передати помилку на всі ваги нейронної мережі. При розрахунку помилки можна вирахувати і дельту на вихідному шарі. Вона буде методично переходити від нейрона до нейрона [18].

2. Метод пружного розповсюдження.

Він був запропонований як альтернатива попередньому способу навчання, який потребує занадто багато часу і стає незручним, якщо

результати потрібно отримати в короткий термін. Для збільшення швидкості операцій було розроблено багато допоміжних алгоритмів, у тому числі методика пружного поширення.

Цей метод є основним під час навчання за принципом epoch (один повний прохід датасету через нейронну мережу). Для припасування вагових коефіцієнтів він використовує лише знаки похідних окремого випадку. У цьому обов'язково витримувати правило, що дозволяє визначити значення корекції коефіцієнта ваги.

Якщо закріпити ключові показники підстроювання вагів, то можна не налаштовувати глобальні параметри – це є додатковим плюсом використання методу. Причому є готові значення таких показників. Їх застосування рекомендовано, але жорстких рамок на вибір значень немає.

Щоб величина ваги була надмірно великою чи, навпаки, маленькою, слід оперувати значенням корекції з встановленими межами. При розрахунку цього значення необхідно дотримуватись правила.

Якщо в певній точці похідна змінює свій знак з «+» на «–», то це говорить про зростання помилки. Тому вагу потрібно змінити у менший бік. У протилежній ситуації – вагу треба збільшити.

У цьому випадку порядок операцій буде таким:

1. Визначення значення корекції.

2. Розрахунок приватних похідних.

3. Розрахунок нової величини корекції вагових значень.

4. Коригування ваг.

Якщо умова зупинки алгоритму не виконується, відбувається повернення до розрахунку похідних, і цикл запускається спочатку.

Завдяки методу пружного поширення збіжність нейронної мережі досягає в строки, значно менші, ніж за попередній алгоритм [19].

3. Генетичний алгоритм навчання.

Ще один поширений підхід – це навчання нейронної мережі генетичним алгоритмом. За своїм принципом він схожий з еволюційними процесами природи, що ґрунтуються на комбінуванні (схрещуванні) результатів.

Іншими словами, відбувається природний відбір, де нове покоління є продуктом комбінації результатів із найкращими властивостями. Якщо результат такого схрещування не підходить за якимись критеріями, то відбір відбувається знову, поки продукт стане досконалим.

Завершення алгоритму відбувається у момент, коли закінчуються відведені йому спроби чи час на мутацію. При цьому результат може бути недосягнутим. Даний метод використовується для покращення показників ваг нейронної мережі за умови, що структура задана за умовчанням. Вага при цьому має бути прописана двійковим кодом, а повний набір ваги сформує підсумковий результат. Розрахунок помилки на виході зумовлює оцінку ефективності [20].

4. Популяційний алгоритм.

Популяційні алгоритми є дуже перспективним напрямом у сфері оптимізації та моделювання. За допомогою цих алгоритмів вирішуються безліч завдань на графах, завдання компонування та складання розкладів, проводиться налаштування та навчання штучних нейронних мереж, та багато іншого [21].

Модель роєвого інтелекту має на увазі наявність так званої «багатоагентної системи», яка визначається як система, що складається з безлічі інтелектуальних агентів – програм, здатних самостійно протягом деякого, досить тривалого проміжку часу виконувати поставлене завдання. На рисунку 1.2 зображено різноманітні методи реалізації роєвого інтелекту.

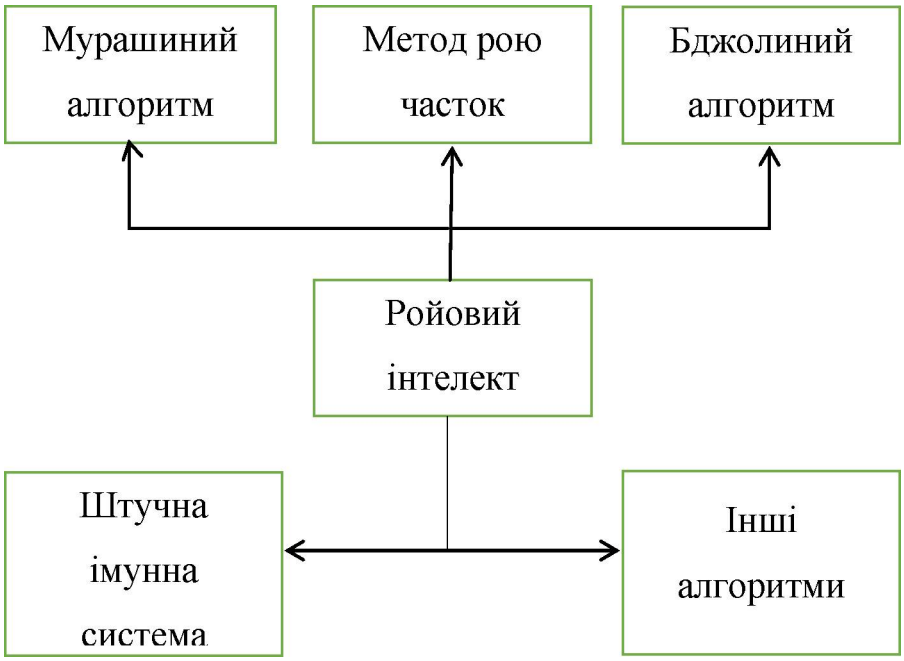

Рисунок 1.2 – Класифікація структурних методів роєвого інтелекту.

1. Метод колонії Мурах.

Основною ідеєю алгоритму є моделювання поведінки мурах. Колонія є системою з простими правилами автономного поведінки особин. Однак, незважаючи на примітивність поведінки кожної окремої мурашки, поведінка всієї колонії виявляється досить розумною [22]. Таким чином, основою поведінки мурашиної колонії служить низькорівнева взаємодія, завдяки якому, в цілому, колонія є розумною багатоагентною системою. Взаємодія визначається через спеціальну хімічну речовину – феромон, що відкладається мурахами на пройденому шляху. При виборі напрямку руху мураха виходить не тільки з бажання пройти найкоротший шлях, але і з досвіду інших мурах, інформацію про яку отримує безпосередньо через рівень концентрації феромонів на кожному шляху. Отже, концентрація феромонів визначає

бажання особини вибрати той чи інший шлях. Однак за такого підходу неминуче попадання в локальний оптимум.

Ця проблема вирішується завдяки випаровування феромонів, яке є негативним зворотним зв'язком.

2. Метод рою часток.

У методі оптимізації роєм часток (particle swarm optimization – PSO) агентами є частинки у просторі параметрів оптимізації задачі. У кожний момент часу (на кожній ітерації) частинки мають у цьому просторі деяке положення та вектор швидкості. Для кожного положення частки обчислюється відповідне значення цільової функції, і на цій основі за певними правилами частка змінює своє положення та швидкість у просторі пошуку. В основу методу PSO покладена соціальна-психологічна поведінкова модель натовпу. Існує декілька різновидів методу. У канонічному методі рою часток (1995 р.) на кожній ітерації при визначенні наступного положення частки враховується інформація о найкращій частинці у складі «сусідів» цієї частки, і навіть інформація про дану частинці на тій ітерації, коли цій частинці відповідало найкраще значення цільової функції. Модифікація канонічної моделі FIPS враховує значення цільової функції, які відповідають усім часткам рою; у деяких моделях частинки групуються у декілька роїв тощо [23].

3. Алгоритм колонії бджіл.

Його суть полягає у моделюванні поведінки колонії бджіл у пошуках нектару. У живій природі принцип роботи бджолиного вулика ґрунтується на чіткому розподілі обов’язків між окремими його індивідами. Усіх бджіл у вулику можна розділити на 3 групи:

1. Робочі бджоли.

2. Бджоли-дослідники.

3. Бджоли-розвідники.

Робочі бджоли займаються пошуком джерел нектару та забезпечують інформацією про якість досліджених ділянок інших бджіл (дослідників). Бджоли-дослідники весь цей час знаходяться у вулику та отримують інформацію про об'єкт дослідження тільки від робочих бджіл. Бджоли-розвідники, у свою чергу, здійснюють випадковий пошук нових джерел нектару. Алгоритм може використовуватись у завданнях оптимізації. Необхідною умовою для його застосування є деяка топологічна відстань або його аналог на галузі рішень. Роботу алгоритму можна розбити на два етапи: ініціалізація та локальний пошук [24].

4. Штучна імунна система.

Штучна імунна система – це така обчислювальна система, яка здатна адаптуватися і використовувати схожі з реальною імунною системою принципи та механіки. У даного методу є три основні теорії, які описують його функціонування та взаємодію між

Елементами.

1. Теорія негативного відбору.

2. Теорія імунної мережі.

3. Теорія клональної селекції.

Інші алгоритми роєвого інтелекту найменш поширені та рідко використовуються. Перелік таких алгоритмів.

1. Метод крапель води, що знаходиться або найбільш близькі, або найбільш оптимальні «шляхи для води», подібно до річок.

2. Алгоритм зозулі – заснований на паразитуванні, подібно до того, як деякі види зозуль відкладали яйця в чужі гнізда, згодом навчившись імітувати кольори чужих яєць.

3. Метод альтруїзму, заснований на тому, що кожен агент «піклується» про оточуючих, не зважаючи на себе.

4. Метод гравітаційного пошуку – укладений у дотриманні закону всесвітнього тяжіння (всі тіла притягуються одне до одного), а саме у пошуках найбільш якісних, «важких» агентів [25].

Після проведення аналізу методів навчання нейронних мереж, популяційні алгоритми виявились дуже перспективним напрямом у сфері оптимізації та моделювання. За допомогою цього алгоритму вирішуються безліч завдань, у тому числі проводиться налаштування та навчання штучних нейронних мереж. Тому був обраний саме цей алгоритм для задачі розпізнавання образів.

Розглянувши всі основні алгоритми роєвого інтелекту: метод рою частинок, мурашиний алгоритм, алгоритм штучної бджолиної колонії та алгоритм штучної імунної системи було прийнято рішення використовувати алгоритм рою часток. Був проведений порівняльний аналіз алгоритмів, в результаті якого були виявлені основні сильні сторони методів, галузі їх застосування а також перспективи розвитку кожного, і найбільш придатним виявився алгоритм рою часток. Цей алгоритм і використовуватиметься для навчання рекурентної нейронної мережі.

Висновок до розділу 1

В цьому розділі було проведено аналіз методів вирішення задачі розпізнавання образів. В результаті аналізу переваг та недоліків цих методів було обрано нейронні мережі, оскільки вони можуть працювати з великим обсягом даних та мають кращі характеристики для розпізнавання образів.

Також було проведено аналіз типів нейронних мереж. І так як рекурентні нейронні мережі спеціалізуються на розпізнаванні тексту, було

вирішено провести експеримент, і перевірити, як рекурентна нейронна мережа впорається з задачею розпізнавання образів.

Після аналізу методів навчання нейронних мереж, де були розглянуті такі методи, як: метод зворотного розповсюдження, метод пружного розповсюдження, генетичний алгоритм, та популяційний алгоритм, було обрано популяційний алгоритм, так як він є дуже перспективним напрямом у сфері оптимізації та моделювання. За допомогою цього алгоритму вирішуються безліч завдань, у тому числі проводиться налаштування та навчання штучних нейронних мереж.

Були розглянуті основні алгоритми роєвого інтелекту та виявлені основні сильні сторони методів, галузі їх застосування а також перспективи розвитку кожного, і найбільш придатним виявився алгоритм рою часток.

РОЗДІЛ 2
РОЗРОБКА МОДЕЛІ РЕКУРЕНТНОЇ НЕЙРОННОЇ МЕРЕЖІ НА ОСНОВІ ПОПУЛЯЦІЙНОГО АЛГОРИТМУ ДЛЯ РОЗПІЗНАВАННЯ ОБРАЗІВ

2.1 Опис вхідних даних

Було прийнято рішення використовувати датасет MNIST. Набір MNIST – це велика колекція рукописних цифр. Це дуже популярний набір даних у галузі обробки зображень. Його часто використовують для тестування алгоритмів машинного навчання. MNIST – це скорочення від модифікованої бази даних Національного інституту стандартів та технологій [26]. MNIST містить колекцію з 70,000 зображень 28 x 28 рукописних цифр від 0 до 9. Набір даних вже поділено на набори для навчання та тестування.

Цифри були нормалізовані за розміром та розташовані в центрі зображення фіксованого розміру. Приклад даних з датасету можна побачити на рисунку 2.1.

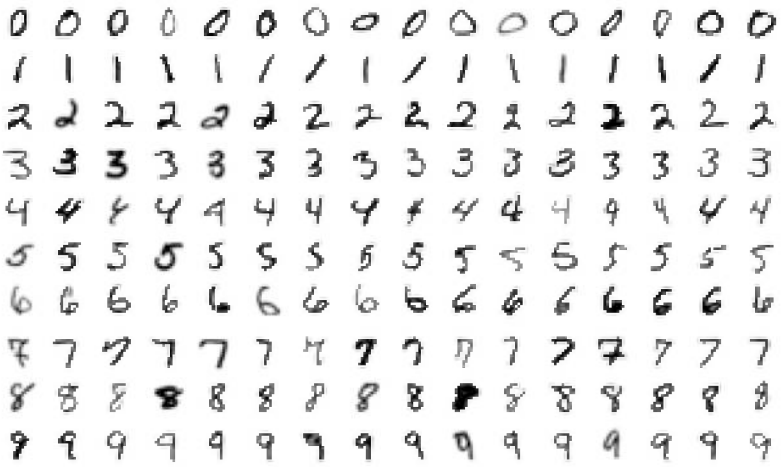

Рисунок 2.1 – Приклад даних з датасету.

2.2 Аналіз структури та методів навчання рекурентної нейронної мережі

2.2.1 Аналіз структури рекурентної нейронної мережі

Розглянемо структуру LSTM-шару. Центральним поняттям тут є блок пам’яті (memory cell), який зі станом мережі h, обчислюється на кожному кроці, використовуючи поточне вхідне значення $x^{(t)}$ і значення блоку на попередньому кроці $c^{(t-1)}$. Вхідний фільтр (input gate) $i^{(t)}$ визначає, наскільки значення блоку пам’яті на поточному етапі має впливати на результат. Значення фільтра варіюються від 0 (повністю ігнорувати вхідні значення) до 1, що забезпечується областю значень сигмоїдальної функції:

$$i^{(t)} = \sigma\left(W^i x^t + U^i h^{(t-1)}\right). \quad (2.1)$$

Фільтр забування (forget gate) дозволяє виключити при обчисленнях значення пам'яті попереднього кроку:

$$f^{(t)} = \sigma(W^f x^{(t)} + U^f h^{(t-1)}) \quad (2.2)$$

На основі всіх даних, що надходять у момент часу t, обчислюється стан блоку пам’яті $c^{(t)}$ на поточному кроці, використовуючи фільтри (2.1) та (2.2):

$$\tilde{c}^{(t)} = tahn\left(W^c x^{(t)} + U^c h^{(t-1)}\right);$$
$$c^{(t)} = f^{(t)} \cdot c^{(t-1)} + i^{(t)} \cdot \tilde{c}^{(t)}. \quad (2.3)$$

Вихідний фільтр (output gate) аналогічний двом попереднім і має вигляд:

$$o^{(t)} = \sigma\left(W^o x^{(t)} + U^o h^{(t-1)}\right). \quad (2.4)$$

Підсумкове значення LSTM-шару визначається вихідним фільтром (2.4) та нелінійною трансформацією над станом блоку пам'яті (2.3) [27]:

$$h^{(t)} = o^{(t)} \cdot \tanh\left(c^{(t)}\right). \quad (2.5)$$

2.2.2 Стандартний метод навчання рекурентної нейронної мережі

Навчання за допомогою рекурентних мереж давно вважалося складним. Але навчання з рекурентними мережами може бути особливо складним через складність вивчення довгострокових залежностей. Проблеми затухання та розширення градієнтів виникають під час зворотного поширення помилок через багато часових кроків. Як приклад розглянемо мережу з одним вхідним вузлом, одним вихідним вузлом і одним повторюваним прихованим вузлом (рис. 2.2). Тепер розглянемо вхідні дані, передані в мережу в момент часу τ, і помилку, обчислену в момент часу t, припускаючи, що вхід дорівнює нулю на проміжних етапах часу. Зв'язування вагових коефіцієнтів між кроками часу означає, що рекурентне ребро в прихованому вузлі j завжди має однакову вагу. Отже, внесок входу в момент часу τ у вихід у момент часу t буде або експоненціально швидко зростати, або наближатися до нуля, оскільки t − τ зростає. Отже, похідна помилки щодо вхідних даних буде або збуджуватися, або затухати.

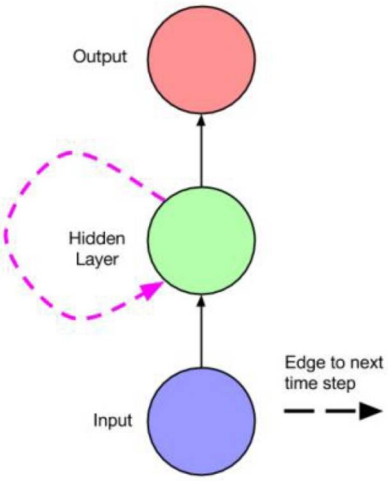

Рисунок 2.2 – Приклад простої архітектури РНМ.

Яке з двох явищ має місце, залежить від того, чи вага повторюваного ребра |wjj | > 1 або |wjj | < 1 і на функцію активації в прихованому вузлі (рисунок 8). Враховуючи сигмоподібну функцію активації, проблема зникаючого градієнта є більш актуальною, але з випрямленою лінійною одиницею max(0, x) легше уявити збуджений градієнт.

Truncated backpropagation through time (TBPTT) є одним із рішень проблеми збудженого градієнта для постійно діючих мереж. За допомогою TBPTT встановлюється деяка максимальна кількість часових кроків, уздовж яких може поширюватися помилка. Архітектура LSTM, описана нижче, використовує ретельно розроблені вузли з повторюваними ребрами з фіксованою одиничною вагою як рішення проблеми затухаючого градієнта. Проблема локальних оптимумів є перешкодою для ефективного навчання, з якою неможливо впоратися просто шляхом зміни архітектури мережі. Оптимізація навіть однієї мережі прямого зв'язку прихованого рівня є проблемою. Однак нещодавні емпіричні та теоретичні дослідження показують, що на практиці це питання може бути не таким важливим, як вважалося раніше.

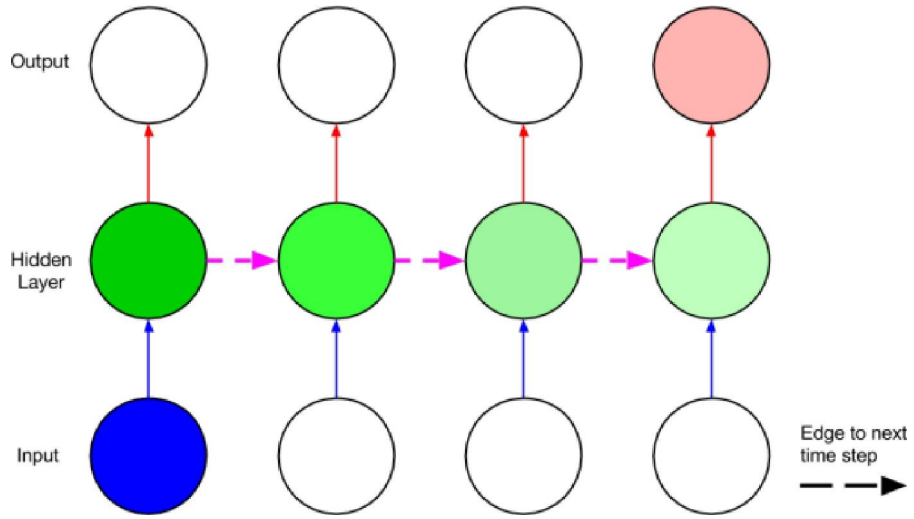

Рисунок 2.3 – Візуалізація проблеми затухаючого градієнта.

Загалом, разом із удосконаленими архітектурами, поясненими нижче, швидкі впровадження та краща евристика відстеження градієнта зробили навчання РНМ можливим. Реалізація прямого та зворотного розповсюдження з використанням графічних процесорів, таких як пакети Theano і Torch, спростило впровадження швидких алгоритмів навчання. У 1996 році, до введення LSTM, було показано, що спроби навчити рекурентні мережі подолати довгі часові прогалини працюють не краще, ніж випадкове вгадування. Проте тепер РНМ часто успішно навчаються.

Для деяких завдань доступне програмне забезпечення, яке можна запускати на одному графічному процесорі та отримувати переконливі результати за години. Мартенс і Сацкевер повідомили про успішне навчання рекурентних нейронних мереж за допомогою усіченого підходу Ньютона без Гессена та застосували цей метод до мережі, яка вчиться генерувати текст по одному символу за раз. Експериментальні результати включають демонстрацію покращеної продуктивності в рекурентних мережах.

Проблеми стандартного навчання рекурентної нейронної мережі можна вирішити за допомогою методу навчання (оптимізації) на основі рою часток [28].

2.3 Аналіз алгоритму рою часток

2.3.1 Алгоритм рою часток

Метод рою часток призначений для вирішення завдань багатовимірної безперервної оптимізації і заснований на моделювання соціальної поведінки колоній тварин, що виконують колективний пошук місць із найкращими умовами існування.

У реалізації даного алгоритму багатовимірний простір пошуку заповнюється роєм часток. Координати частинки у просторі однозначно визначають розв'язання задачі оптимізації. Крім координат кожна з частинок описується швидкістю переміщення та прискоренням. У процесі переміщення частки здійснюють “прочісування” простору рішень і цим знаходять поточний оптимум, якого наступному кроці прагнуть інші частки. Кожна частка запам'ятовує своє найкраще положення, дані про яке передаються сусіднім часткам, які прагнуть цього значення [29].

Схема роботи алгоритму виглядає так:

1. Створюється вихідна «випадкова» група часток.

2. Для кожної частки обчислюється цільова функція.

3. Найкраща частка з точки зору цільової функції оголошується центром тяжіння.

4. Вектори швидкостей всіх часток спрямовуються до цього “центру”, причому чим далі частка від нього, тим більшим прискоренням вона володіє.

5. Здійснюється розрахунок нових координат частки у просторі рішень.

6. Кроки 2-5 повторюються задане число разів або поки не виконається умова зупинки.

7. Останній “центр тяжкості” оголошується знайденим оптимальним рішенням.

Позиція частки регулюється формулою, яку можна побачити нижче.

$$v_{i+1} = \chi \cdot [v_i a_1 \cdot rnd() \cdot (pbest_i - x_i)\, a_2 \cdot rnd() \cdot (gbest_i - x_i)] \tag{2.6}$$

2.3.2 Навчання рекурентної нейронної мережі на основі рою часток

Алгоритм нейронної мережі LSTM на основі оптимізації рою часток (PSO-LSTM) включає багато параметрів у побудову моделі прогнозування LSTM, з яких ваги є найважливішими. Щоб отримати кращі результати прогнозування, ми використовуємо алгоритм оптимізації рою часток для оптимізації цих параметрів. PSO може уникнути попадання мережевої конвергенції в локальне оптимальне рішення. Ми беремо ваги прихованого шару LSTM як вхідні дані для рою частинок. Початкова вихідна помилка LSTM використовується як придатність рою частинок, а потім оцінюється продуктивність частинки відповідно до умов. Випадковий початковий рій частинок оновлює свій власний параметр відповідно до індивідуального екстремуму та глобального екстремуму. У кожному ітераційному процесі формула для оновлення швидкості та положення частинки така:

$$V_{id}^{k+1} = \omega V_{id}^{k} + c_1 r_1 (P_{id}^{k} - X_{id}^{k}) + c_2 r_2 (P_{gd}^{k} - X_{gd}^{k}) \tag{2.7}$$

$$X_{id}^{k+1} = X_{id}^{k} + V_{id}^{k+1} \tag{2.8}$$

У наведених вище формулах d = 1,2,..., D;

i = 1,2,..., n;

P_{id} – індивідуальний екстремум;

P_{gd} – глобальний екстремум;

V_{id} – швидкість часток;

X_{id} – положення часток;

ω – інерційна вага;

k – поточний час ітерації;

c_1, c_2 – фактори навчання;

r_1, r_2 – випадкові числа, які розподілені в [0,1].

Коли алгоритм PSO оптимізує алгоритм LSTM, значення вектора оптимального положення частинок у зграї часток використовується послідовно як початкове значення кожної ваги в мережі LSTM. Розмір кожної частинки можна розрахувати відповідно до структури моделі нейронної мережі, а середню квадратичну помилку кожного вихідного нейрона даного навчального набору прийнято як функцію придатності рою часток. Потім ми обчислюємо значення відповідності кожної частинки відповідно до функції відповідності, чим менше значення, тим менша помилка виходу мережі. Це також означає кращу продуктивність відповідних частинок. Положення частинок постійно оновлюється, щоб похибка вихідного шару мережі поступово зменшувалася. У кожній ітерації частинка з найменшою помилкою береться як поточна оптимальна частинка [30].

Блок-схема розробленої моделі рекурентної нейронної мережі на основі популяційного алгоритму показана на рисунку 2.4.

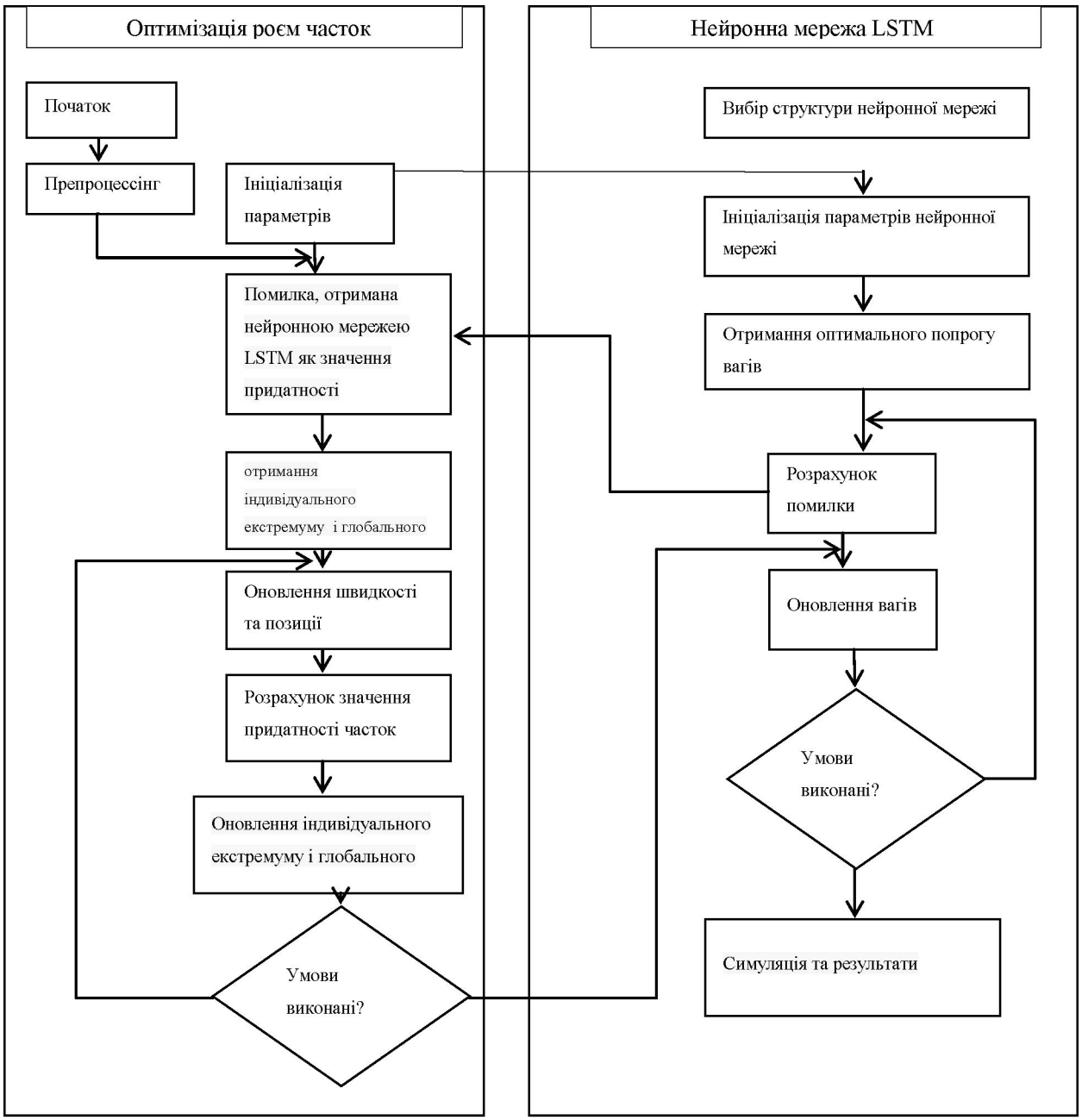

Рисунок 2.4 – Блок-схема розробленої моделі РНМ на основі популяційного алгоритму.

Висновок до розділу 2

В цьому розділі було розроблено модель рекурентної нейронної мережі на основі популяційного алгоритму. Був приведений опис вхідних даних моделі, використовуючи датасет MNIST. Розглянуто структуру рекурентної нейронної мережі та стандартний метод навчання рекурентної нейронної

мережі, в якому були виявлені проблеми, які можна вирішити за допомогою методу навчання (оптимізації) на основі рою часток.

Саме цей тип популяційного алгоритму і буде використовуватися у програмній реалізації моделі рекурентної нейронної мережі.

РОЗДІЛ 3.
ПРОГРАМНА РЕАЛІЗАЦІЯ МОДЕЛІ РЕКУРЕНТНОЇ НЕЙРОННОЇ МЕРЕЖІ НА ОСНОВІ ПОПУЛЯЦІЙНОГО АЛГОРИТМУ ДЛЯ РОЗПІЗНАВАННЯ ОБРАЗІВ

3.1 Інструментальні засоби реалізації моделі

Програмна модель рекурентної нейронної мережі була реалізована на мові програмування Python 3.7.

Python широко використовується в багатьох сферах, він дуже популярний в сфері машинного навчання. Він потужний, простий і підтримує спеціальні пакети, які підвищують його ефективність. І головне, стандартна бібліотека Python містить у собі усі готові інструменти для роботи з нейронними мережами та базами даних.

Було обрано бібліотеку NumPy. Ця бібліотека значно полегшує роботу з обчисленнями, дозволяє використовувати центральний процесор (CPU) і графічний процесор (GPU), в якому реалізована технологія CUDA від Nvidia.

CPU – це центральний процесор, який виконує операції з даними. GPU – це графічний процесор, який швидше обробляє дані, оскільки він виконує завдання не послідовно, як CPU, а паралельно.

В якості бекенда, крім NumPy, для розрахунків була використана бібліотека SciPy.

Бібліотека SciPy залежить від NumPy, який забезпечує зручну та швидку маніпуляцію з N-мірним масивом. Бібліотека SciPy створена для роботи з масивами NumPy та надає безліч зручних та ефективних чисельних методів, таких як процедури чисельної інтеграції та оптимізації. Разом вони працюють на всіх популярних операційних системах, швидко встановлюються та безкоштовні. NumPy та SciPy прості у використанні, але

досить потужні, щоб залежати від провідних світових вчених та інженерів [31]. Для програмної реалізації моделі була використана хмарна платформа Colaboratory, яка розроблена Google.

Colaboratory – це інструмент, який використовується для швидкого навчання і тестування різних моделей автоматичного навчання. Colaboratory є хмарним сервісом, який призначений для навчання та досліджень в області машинного навчання. В ньому встановлені бібліотеки NumPy і SciPy.

3.2 Програмна реалізація моделі

Першим кроком було завантаження бібліотек, в яких є функції, які необхідні для роботи (рис.3.1).

```
!pip install scikit-learn

Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.7/dist-packages (1.0.2)
Requirement already satisfied: joblib>=0.11 in /usr/local/lib/python3.7/dist-packages (from scikit-learn) (1.2.0)
Requirement already satisfied: scipy>=1.1.0 in /usr/local/lib/python3.7/dist-packages (from scikit-learn) (1.7.3)
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.7/dist-packages (from scikit-learn) (3.1.0)
Requirement already satisfied: numpy>=1.14.6 in /usr/local/lib/python3.7/dist-packages (from scikit-learn) (1.21.6)

import functools
import numpy as np
import sklearn.metrics
import scipy.special
import sklearn.datasets
import sklearn.model_selection
```

Рисунок 3.1 – Завантаження бібліотек.

Наступним кроком була розробка класу нейронної мережі з основними функціями (рис.3.2).

```
class RNN:
    def __init__(self, shape, weights=None):
        self.shape = shape
        self.num_layers = len(shape)
        if weights is None:
            self.weights = []
            for i in range(self.num_layers-1):
                W = np.random.uniform(size=(self.shape[i+1], self.shape[i] + 1))
                self.weights.append(W)
        else:
            self.weights = weights


    def run(self, data):
        layer = data.T
        for i in range(self.num_layers-1):
            prev_layer = np.insert(layer, 0, 1, axis=0)
            o = np.dot(self.weights[i], prev_layer)
            layer = scipy.special.expit(o)
        return layer
```

Рисунок 3.2 – Розробка класу нейронної мережі з основними функціями.

Далі було розроблено дві функції переробки вагів у вектор, та вектору у ваги, для того, щоб рій часток міг з ними працювати (рис. 3.3).

```
def weights_to_vector(weights):
    w = np.asarray([])
    for i in range(len(weights)):
        v = weights[i].flatten()
        w = np.append(w, v)
    return w

def vector_to_weights(vector, shape):
    weights = []
    idx = 0
    for i in range(len(shape)-1):
        r = shape[i+1]
        c = shape[i] + 1
        idx_min = idx
        idx_max = idx + r*c
        W = vector[idx_min:idx_max].reshape(r,c)
        weights.append(W)
    return weights
```

Рисунок 3.3 – Функція переробки вагів.

Після цього було розроблено клас рою часток, в якому зберігаються всі основні параметри, константи chi, phi_p та phi_g які були знайдені в дослідженнях у вільному доступі. Такі значення цих констант найбільш популярні та використовуються в багатьох моделях (рис.3.4).

```
class ParticleSwarm(object):
    def __init__(self, cost_func, num_dimensions, num_particles, chi=0.72984, phi_p=2.05, phi_g=2.05):
        self.cost_func = cost_func
        self.num_dimensions = num_dimensions

        self.num_particles = num_particles
        self.chi = chi
        self.phi_p = phi_p
        self.phi_g = phi_g

        self.X = np.random.uniform(size=(self.num_particles, self.num_dimensions))
        self.V = np.random.uniform(size=(self.num_particles, self.num_dimensions))

        self.P = self.X.copy()
        self.S = self.cost_func(self.X)
        self.g = self.P[self.S.argmin()]
        self.best_score = self.S.min()
```

Рисунок 3.4 – Розробка класу рою часток.

Далі було розроблено дві функції: мінімізації та результату (рис. 3.5).

```
def minimize_pso(cost_func, num_dimensions, num_iterations):
    num_particles = num_dimensions * 2
    swarm = ParticleSwarm(cost_func, num_dimensions, num_particles)
    return swarm.minimize(num_iterations)


class PSOResult(object):
    def __init__(self, best_particle, best_score, num_iterations):
        self.best_particle = best_particle
        self.best_score = best_score
        self.num_iterations = num_iterations
```

Рисунок 3.5 – Функції мінімізації та результату.

Наступним кроком було необхідно оновляти швидкість та позицію кожної частки. Програмну реалізацію цих функцій можна побачити на рисунку 3.6.

```
def _update(self):
    # Обновление скоростей
    R_p = np.random.uniform(size=(self.num_particles, self.num_dimensions))
    R_g = np.random.uniform(size=(self.num_particles, self.num_dimensions))

    self.V = self.chi * (self.V \
            + self.phi_p * R_p * (self.P - self.X) \
            + self.phi_g * R_g * (self.g - self.X))

    # Обновление позиций
    self.X = self.X + self.V

    # Лучшие результаты
    scores = self.cost_func(self.X)

    better_scores_idx = scores < self.S
    self.P[better_scores_idx] = self.X[better_scores_idx]
    self.S[better_scores_idx] = scores[better_scores_idx]

    self.g = self.P[self.S.argmin()]
    self.best_score = self.S.min()
```

Рисунок 3.6 – Оновлення швидкості та позиції кожної частки.

Коли модель повністю була реалізована, необхідно було завантажити дані, розділити їх на навчальну та тестову вибірки, визначити кількість класів та нормалізувати їх (рис.3.7).

```
# Загрузить цифры MNIST из sklearn
num_classes = 10
mnist = sklearn.datasets.load_digits()
X, X_test, y, y_test = sklearn.model_selection.train_test_split(mnist.data, mnist.target)

num_inputs = X.shape[1]

y_true = np.zeros((len(y), num_classes))
for i in range(len(y)):
    y_true[i, y[i]] = 1

y_test_true = np.zeros((len(y_test), num_classes))
for i in range(len(y_test)):
    y_test_true[i, y_test[i]] = 1

shape = (num_inputs, 64, 32, num_classes)

cost_func = functools.partial(eval_neural_network, shape=shape, X=X, y=y_true.T)

swarm = ParticleSwarm(cost_func, num_dimensions=dim_weights(shape), num_particles=30)
```

Рисунок 3.7 – Завантаження даних, поділ їх на навчальну та тестову вибірки, визначення кількості класів та нормалізування.

Після того як модель була готова, було проведено її тренування на тестовій вибірці (рис.3.8).

```
# Тренировка...
i = 0
best_scores = [(i, swarm.best_score)]
print_best_particle(best_scores[-1])
while swarm.best_score>1e-6 and i<500:
    swarm._update()
    i = i+1
    if swarm.best_score < best_scores[-1][1]:
        best_scores.append((i, swarm.best_score))
        print_best_particle(best_scores[-1])
```

Рисунок 3.8 – Тренування моделі на тестовій вибірці.

Останнім кроком було проведено тестування моделі на тестовій вибірці (рис.3.9).

```
best_weights = vector_to_weights(swarm.g, shape)
best_nn = RNN(shape, weights=best_weights)
y_test_pred = np.round(best_nn.run(X_test))
print(sklearn.metrics.classification_report(y_test_true, y_test_pred.T))
```

Рисунок 3.9 – Тестування моделі на тестовій вибірці.

Коли програмна модель була повністю реалізована, було прийнято рішення реалізувати програмну модель рекурентної нейронної мережі для розпізнавання образів на основі генетичного алгоритму, для порівняння з основною моделлю. Вибір моделі на основі генетичного алгоритму обумовлений тим, що після аналізу багатьох досліджень в оптимізації рекурентної нейронної мережі, генетичний алгоритм має перевагу над звичайними методами навчання, та точність розпізнавання є більшою.

Проведемо аналіз основних функцій моделі рекурентної нейронної моделі на основі генетичного алгоритму.

Налаштування основних параметрів генетичного алгоритму (рис.3.10).

```
# Настраиваем параметры генетического алгоритма
DNA_size = 2
DNA_size_max = 8      # Длина каждой хромосомы
POP_size = 20         # Численность популяции
CROSS_RATE = 0.5      # Частота скрещивания
MUTATION_RATE = 0.01 # Коэффициент вариации
N_GENERATIONS = 40    # Кол-во итераций


# получаем данные
x_train, x_test, y_test, y_train = load_data()
```

Рисунок 3.10 – Налаштування основних параметрів генетичного алгоритму.

Функція створення нової популяції (рис3.11).

```
def select(pop, fitness):
    # Выполняется операция отбора
    idx = np.random.choice(np.arange(POP_size), size=POP_size, replace=True, p=fitness / fitness.sum())
    # Выбранная популяция формирует начальную популяцию
    return pop[idx]
```

Рисунок 3.11 – Функція створення нової популяції.

Функція кросоверу (рис.3.12).

```
def crossover(parent, pop):
    # Если случайное число меньше вероятности пересечения, происходит пересечение
    if np.random.rand() < CROSS_RATE:
        # Выбираем одну популяцию из 20 популяций для перехода
        i_ = np.random.randint(0, POP_size, size=1)  # номер хромосомы
        # Здесь будет сгенерировано 8-мерное двоичное число и преобразовано в тип bool. True указывает, что позиция пересекается, а False указывает, что она не пересекается.
        cross_points = np.random.randint(0,2,size=DNA_size_max).astype(np.bool) # используем True и False, чтобы указать, следует ли заменить

        for i, point in enumerate(cross_points):
            # Часть I
            if point == True and pop[i_, i] * parent[i] == 0:
                cross_points[i] = False

            # Часть II
            if point == True and i < 2:
                cross_points[i] = False

        # Замена i_ генов в соответствующих позициях
        parent[cross_points] = pop[i_, cross_points]
    return parent
```

Рисунок 3.12 – Функція кросоверу.

Функція мутації (рис.3.13).

```
def mutate(child):
    # Операция мутации предназначена только для последних 6-битных параметров
    for point in range(DNA_size_max):
        if np.random.rand() < MUTATION_RATE:
            # Только параметры после 2-битных параметров участвуют в изменении
            if point >= 2:
                if child[point] != 0:
                    child[point] = np.random.randint(32, 257)
    return child
```

Рисунок 3.13 – Функція мутації.

Основна функція розрахунків (рис.3.14).

```
# Присвоим значения инициализированной популяции. Первые два столбца — количество слоев, а последние шесть столбцов — количество нейронов.
for i in range(POP_size):
    # Случайным образом выбираем случайный массив из [32256], чтобы сформировать числовую информацию о нейронах.
    pop_neurons = np.random.randint(32, 257, size=(pop_layers[i].sum(),))
    # Два столбца с информацией о количестве слоев и шесть столбцов с информацией о количестве нейронов объединяются и умножаются на информацию о 8-м
    pop_stack = np.hstack((pop_layers[i], pop_neurons))
    # Инициализируем популяцию
    for j, gene in enumerate(pop_stack):
        pop[i][j] = gene

# фитнес-функция популяции вычисляется в заданных пределах числа итераций
for each_generation in range(N_GENERATIONS):
    # Инициализируем fitness
    fitness = np.zeros([POP_size,])
    # Пройдем через 20 популяций и манипулируем генами
    for i in range(POP_size):
        pop_list = list(pop[i])
        # Ген на хромосоме i
        # Удалить ген, присвоенный 0
        for j,each in enumerate(pop_list):
            if each == 0.0:
                index = j
                pop_list = pop_list[:j]
        # Преобразуем ген в тип int
        for k,each in enumerate(pop_list):
            each_int = int(each)
            pop_list[k] = each_int
        # Заполним рассчитанный фитнес в массиве пригодности
        fitness[i] = get_fitness(pop_list)
        # Выведем результаты
        print('The first %d Dai di %d The fitness of chromosomes is %f' % (each_generation+1, i+1, fitness[i]))
        print('This chromosome is:', pop_list)
    print('Generation: ', each_generation + 1, ' Most fitted DNA: ', pop[np.argmax(fitness), :], ' The fitness is: ', fitness[np.argmax(fitness)])
```

Рисунок 3.14 – Основна функція розрахунків.

3.3 Дослідження моделі рекурентної нейронної мережі для вирішення задачі розпізнавання

План дослідження.

1. Першим кроком потрібно завантажити датасет MNIST у дві моделі, після цього алгоритм автоматично нормалізує та приведе у нормальний вид вхідні дані.
2. Далі потрібно навчити дві моделі розпізнавати образи на основі тестувальної вибірки.
3. Після навчання, коли всі ваги налаштовані, потрібно провести тестування на навчальній вибірці.
4. Оцінити якість розпізнавання та швидкість роботи обох алгоритмів.
5. Зробити висновки щодо проведеного дослідження

Були розроблені дві програмні моделі рекурентної нейронної мережі на основі генетичного та популяційного алгоритмів.

Завантаження датасету MNIST можна побачити на рисунку 3.15.

```
# Загрузить цифры MNIST из sklearn
num_classes = 10
mnist = sklearn.datasets.load_digits()
X, X_test, y, y_test = sklearn.model_selection.train_test_split(mnist.data, mnist.target)
```

Рисунок 3.15 – Завантаження датасету MNIST.

Далі було проведено навчання моделі рекурентної нейронної мережі на основі популяційного алгоритму (рис.3.16).

```
# Тренировка...
i = 0
best_scores = [(i, swarm.best_score)]
print_best_particle(best_scores[-1])
while swarm.best_score>1e-6 and i<500:
    swarm._update()
    i = i+1
    if swarm.best_score < best_scores[-1][1]:
        best_scores.append((i, swarm.best_score))
        print_best_particle(best_scores[-1])
```

Рисунок 3.16 – Навчання моделі рекурентної нейронної мережі на основі популяційного алгоритму.

Коли були налаштовані ваги програмної моделі, потрібно провести тестування на тестовій вибірці (рис.3.17).

```
# Тестирование...
best_weights = vector_to_weights(swarm.g, shape)
best_nn = RNN(shape, weights=best_weights)
y_test_pred = np.round(best_nn.run(X_test))
print(sklearn.metrics.classification_report(y_test_true, y_test_pred.T))
```

Рисунок 3.17 –Тестування на тестовій вибірці.

Тепер можна провести порівняльний аналіз цих моделей, для визначення, який метод навчання є кращим. Було завантажено датасет MNIST, де 60,000 образів було виділено на навчальну вибірку та 10,000 на тестувальну вибірку. Результати виконання роботи моделі генетичного алгоритму можна побачити на рисунку 3.18.

```
The first 1 Dai di 1 The fitness of chromosomes is 0.960300
This chromosome is: [2, 3, 160, 124, 42, 207, 173]
lstm:  (None, 28, 204)
lstm:  (None, 217)
dense:  (None, 226)
dense:  (None, 191)
dense:  (None, 143)
last_dense  (None, 10)
Training set shape  (60000, 28, 28)
1688/1688 [==============================] - 325s 190ms/step - loss: 0.5980 - accuracy: 0.8965 - val_loss: 0.2521 - val_accuracy: 0.9613
```

Рисунок 3.18 – Результати виконання роботи моделі генетичного алгоритму.

Як можна побачити, точність розпізнавання моделі дорівнює 0,89.

Результати виконання роботи моделі популяційного алгоритму можна побачити на рисунку 3.19.

```
Новая наилучшая частица найдена на итерации #496 со среднеквадратической ошибкой: 0.06867739746741808
              precision    recall  f1-score   support

           0       0.98      0.90      0.94        52
           1       0.00      0.00      0.00        41
           2       0.86      0.71      0.78        45
           3       0.00      0.00      0.00        44
           4       0.92      0.75      0.83        48
           5       0.00      0.00      0.00        41
           6       0.95      0.81      0.88        48
           7       0.00      0.00      0.00        41
           8       0.00      0.00      0.00        49
           9       0.00      0.00      0.00        41
```

Рисунок 3.19 – Результати виконання роботи моделі популяційного алгоритму.

В цьому випадку, точність алгоритму досягає до 0,98.

Вже на цьому етапі можна побачити, що модель на основі популяційного алгоритму має перевагу над моделлю з генетичним методом навчання. Різниця точності алгоритмів дорівнює 0,09.

Можна зробити висновок, що модель на основі популяційного алгоритму має перспективи у подальшому аналізі більш складних датасетів, так як точність алгоритму є більше, ніж у генетичного алгоритму, який в свою чергу є не менш популярним та ефективним.

Висновок до розділу 3

В цьому розділі були описані інструментальні засоби для програмної реалізації моделі.

Були розроблені програмні моделі рекурентної нейронної мережі на основі популяційного алгоритму та генетичного алгоритму для проведення порівняльного аналізу. Програмні моделі були реалізовані на мові

програмування Python 3.7 та для програмної реалізації моделей була використана хмарна платформа Colaboratory, в якій встановлені бібліотеки NumPy і SciPy.

Після проведення порівняльного аналізу моделей на основі популяційного та генетичного алгоритмів було виявлено, що модель на основі популяційного алгоритму має перевагу, так як її точність складає 0,98, та вона має перспективи у подальшому аналізі більш складних датасетів. Модель на основі генетичного алгоритму має точність розпізнавання 0,89, який в свою чергу є не менш популярним та ефективним.

ВИСНОВОК

В ході виконання кваліфікаційної роботи в першому розділі було проведено аналіз задачі розпізнавання образів. Після проведення аналізу, було обрано нейронні мережі, оскільки вони можуть працювати з великим обсягом даних та мають кращі характеристики для розпізнавання образів.

Також було проведено аналіз типів нейронних мереж. І так як рекурентні нейронні мережі спеціалізуються на розпізнаванні тексту, було вирішено провести експеримент, і перевірити, як рекурентна нейронна мережа впорається з задачею розпізнавання образів.

Після аналізу методів навчання нейронних мереж, було обрано популяційний алгоритм, так як він є дуже перспективним напрямом у сфері оптимізації та моделювання.

В другому розділі було розроблено модель рекурентної нейронної мережі. Був приведений опис вхідних даних моделі. Також було розглянуто структуру рекурентної нейронної мережі та стандартний метод навчання рекурентної нейронної мережі, в якому були виявлені проблеми, які можна вирішити за допомогою методу навчання (оптимізації) на основі рою часток.

Саме цей тип популяційного алгоритму і використовувався у програмній реалізації моделі рекурентної нейронної мережі.

В третьому розділі розроблені програмні моделі рекурентної нейронної мережі на основі популяційного алгоритму та генетичного алгоритму для проведення порівняльного аналізу. Було виявлено, що модель на основі популяційного алгоритму має перспективи у подальшому аналізі більш складних датасетів, так як точність алгоритму є більше, ніж у генетичного алгоритму, який в свою чергу є не менш популярним та ефективним.

Дану модель та результати, отримані в ході розробки та тестування, можна буде використовувати у всіх галузях людської діяльності, наприклад для розпізнавання номерів товарних знаків або номерів автомобілів. Дану модель можна удосконалювати для розв'язання більш складних задач з більш складним датасетом.

За результатом роботи було представлено доклад на науково-технічну міжнародну конференцію «Комп'ютерне моделювання у наукоємних технологіях (КМНТ —2022)» 23-24 листопада 2022 року, Харків, Україна.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Основи теорії розпізнавання образів : навч. посіб. : у 2 ч. / А. С. Довбиш, І. В. Шелехов. – Суми : Сумський державний університет — 2015. — Ч. 1. — 109 с.
2. Harshali M. Image classification using Deep learning — 2015
3. Hongkai W. Zongwei Z. Yingci L. Zhonghua C. Peiou L. Comparison of machine learning methods for classifying mediastinal lymph node metastasis of non small cell lung cancer from 18F-FDG PET/CT images — 2017
4. Logistic regression and machine learning [Електронний ресурс]. Режим доступу: https://www.ibm.com/topics/logistic-regression (Дата звернення – 13.09.2022)
5. Decision Tree Classification Algorithm [Електронний ресурс]. Режим доступу: https://www.javatpoint.com/machine-learning-decision-tree-classification-algorithm (Дата звернення – 13.09.2022)
6. The nearest neighbor method [Електронний ресурс]. Режим доступу: https://buildingai.elementsofai.com/Machine-Learning/the-nearest-neighbor-method (Дата звернення – 14.09.2022)
7. Support Vector Machines [Електронний ресурс]. Режим доступу: https://scikit-learn.org/stable/modules/svm.html (Дата звернення – 14.09.2022)
8. Linear Discriminant Analysis [Електронний ресурс]. Режим доступу: https://towardsdatascience.com/lda-linear-discriminant-analysis-how-to-improve-your-models-with-supervised-dimensionality-52464e73930f (Дата звернення – 14.09.2022)

9. Artificial Neural Network Tutorial [Електронний ресурс]. Режим доступу: https://www.javatpoint.com/artificial-neural-network (Дата звернення – 15.09.2022)
10. Artificial neural network [Електронний ресурс]. Режим доступу: https://www.inf.ed.ac.uk/teaching/courses/nlu/assets/reading/Gurney_et_al.pdf (Дата звернення – 17.09.2022)
11. Direct Channel of Distribution [Електронний ресурс]. Режим доступу: https://yourbusiness.azcentral.com/direct-channel-distribution-13217.html (Дата звернення – 17.09.2022)
12. Markov Chains [Електронний ресурс]. Режим доступу: https://setosa.io/ev/markov-chains/ (Дата звернення – 19.09.2022)
13. Hopfield Network [Електронний ресурс]. Режим доступу: https://www.javatpoint.com/artificial-neural-network-hopfield-network (Дата звернення – 21.09.2022)
14. Нейронна мережа Хеммінга. Нейронна мережа Хебба. Мережі Кохонена [Електронний ресурс]. Режим доступу: https://fkti5301.github.io/exam_tickets_ai_2018_novakova/tickets/21.html (Дата звернення – 21.09.2022)
15. Recurrent Neural Networks [Електронний ресурс]. Режим доступу: https://www.mygreatlearning.com/blog/types-of-neural-networks/ (Дата звернення – 25.09.2022)
16. Introduction to Machine Learning, Neural Networks, and Deep Learning [Електронний ресурс]. Режим доступу: https://tvst.arvojournals.org/article.aspx?articleid=2762344 (Дата звернення – 02.10.2022)

17. 5 Algorithms to Train a Neural Network [Електронний ресурс]. Режим доступу: https://www.datasciencecentral.com/5-algorithms-to-train-a-neural-network/ (Дата звернення – 04.10.2022)
18. Understanding Backpropagation Algorithm [Електронний ресурс]. Режим доступу: https://towardsdatascience.com/understanding-backpropagation-algorithm-7bb3aa2f95fd (Дата звернення – 10.10.2022)
19. An Information-Based Machine Learning Approach to Elasticity Imaging [Електронний ресурс]. Режим доступу: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5423826/ (Дата звернення – 10.10.2022)
20. Introduction to Genetic Algorithms [Електронний ресурс]. Режим доступу: https://towardsdatascience.com/introduction-to-genetic-algorithms-including-example-code-e396e98d8bf3 (Дата звернення – 15.10.2022)
21. The Basics of Genetic Algorithms in Machine Learning [Електронний ресурс]. Режим доступу: https://www.section.io/engineering-education/the-basics-of-genetic-algorithms-in-ml/ (Дата звернення – 16.10.2022)
22. Introduction to Ant Colony Optimization [Електронний ресурс]. Режим доступу: https://www.geeksforgeeks.org/introduction-to-ant-colony-optimization/ (Дата звернення – 20.10.2022)
23. Particle Swarm Optimization [Електронний ресурс]. Режим доступу: https://www.sciencedirect.com/topics/engineering/particle-swarm-optimization (Дата звернення – 24.10.2022)
24. Artificial bee colony algorithm [Електронний ресурс]. Режим доступу:

http://www.scholarpedia.org/article/Artificial_bee_colony_algorithm (Дата звернення – 26.10.2022)

25. Artificial immune system [Електронний ресурс]. Режим доступу: https://en.wikipedia.org/wiki/Artificial_immune_system (Дата звернення – 28.10.2022)
26. MNIST database [Електронний ресурс]. Режим доступу: https://en.wikipedia.org/wiki/MNIST_database (Дата звернення – 30.10.2022)
27. Deep learning: RNNs and LSTM [Електронний ресурс]. Режим доступу: https://www.sciencedirect.com/topics/engineering/recurrent-neural-network (Дата звернення – 02.11.2022)
28. Рекуррентные нейронные сети (RNN) [Електронний ресурс]. Режим доступу: https://habr.com/ru/post/487808/ (Дата звернення – 02.11.2022)
29. A Tutorial on Particle Swarm Optimization [Електронний ресурс]. Режим доступу: https://analyticsindiamag.com/a-tutorial-on-particle-swarm-optimization-in-python/ (Дата звернення – 03.11.2022)
30. Particle Swarm Optimization trained recurrent neural network for voltage instability prediction [Електронний ресурс]. Режим доступу: https://www.sciencedirect.com/science/article/pii/S231471721730020X (Дата звернення – 05.11.2022)
31. Python Numpy Tutorial (with Jupyter and Colab) [Електронний ресурс]. Режим доступу: https://cs231n.github.io/python-numpy-tutorial/ (Дата звернення – 06.11.2022)

ДОДАТОК А

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Харківський національний університет імені В. Н. Каразіна

Факультет комп'ютерних наук
Кафедра теоретичної та прикладної системотехніки
Рівень вищої освіти (освітньо-кваліфікаційний рівень) Магістр
галузь знань 15 – Автоматизація та приладобудування
спеціальність 151 – Автоматизація та комп'ютерно-інтегровані технології

ЗАТВЕРДЖУЮ
Завідувач кафедри теоретичної
та прикладної системотехніки
_____д.т.н., проф. Шматков С. І.
10.12.2021 року

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Кондратюк Діани Сергіївни

(прізвище, ім'я, по батькові студента)

1. Тема роботи «Методи навчання рекурентної нейронної мережі для вирішення задачі розпізнавання образів»_____

керівник роботи Шматков Сергій Ігоревич, професор, д. т. н.

(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від "___"_____20__року
№_____

2.Строк подання студентом роботи___30 листопада 2022_____

3. Перелік питань, які потрібно розробити

1. Аналіз методів навчання рекурентної нейронної мережі.
2. Розробка програмної моделі рекурентнох нейронної мережі для вирішення задачі розпізнавання образів.
3. Оцінка працездатності та порівняння з іншою моделлю для отримання висновків щодо якості виконання.

4. План роботи

№ з/п	Назви етапів роботи	Термін виконання етапів роботи
1	Проведення аналізу літературних джерел по методам навчання рекурентної нейронної мережі	10.12.2021 – 12.01.2022
2	Розробка алгоритму навчання рекурентної нейронної мережі для вирішення задачі розпізнавання образів	13.01.2022 – 31.01.2022
3	Програмна реалізація моделі рекурентної нейронної мережі для вирішення задачі розпізнавання образів	01.02.2022 – 27.02.2022
4	Проведення тесту працездатності	28.02.2022 – 29.03.2022
5	Оцінка якості роботи моделі та порівняння з іншою	30.03.2022 – 30.04.2022

6	Розробка висновків та практичних рекомендацій	01.05.2022 – 31.05.2022
7	Написання пояснювальної записки	01.06.2022 – 30.08.2022
8	Представлення роботи науковому керівнику	31.08.2022 – 01.09.2022
9	Написання звіту з науково-дослідної практики	01.09.2022 – 12.10.2022
10	Написання звіту з преддипломної практики	13.10.2022 – 20.11.2022

5. Дата видачі завдання _____10.12.2021_____

Студент ___Д. С. Кондратюк_____
ініціали, прізвище підпис

Керівник роботи __С. І. Шматков _____
ініціали, прізвище підпис

ДОДАТОК Б

Технічне завдання

на розробку програмного виробу «Методи навчання рекурентної нейронної мережі для вирішення задачі розпізнавання образів»

1.	Введення	1.1. Назва: Методи навчання рекурентної нейронної мережі для вирішення задачі розпізнавання образів. 1.2. Галузь застосування: розпізнавання, аналіз.
2.	Підстава для розробки	2.1. Навчальний план за спеціальністю 151 – Автоматизація та комп'ютерно-інтегровані технології 2.2. Завдання на кваліфікаційну роботу магістра № _____ від «___» _____2022 (представити як Додаток А до пояснювальної записки до кваліфікаційної роботи).
3.	Призначення розробки	3.1. Мета розробки: аналіз методів навчання рекурентної нейронної мережі та розробка програмної моделі рекурентної нейронної мережі для вирішення задачі розпізнавання образів. 3.2. Призначення розробки надає можливість розпізнавати вхідний потік даних за допомогою програмної моделі. 3.3. Вихідні дані розробки: статистичні експериментальні дані, отримані від програмної моделі.
4.	Технічні вимоги до програмного виробу	4.1. Вимоги до функціональних характеристик: модель повинна давати точні результати розпізнавання при завантаженні датасету. 4.2. Вимоги до надійності: забезпечувати точність отриманих результатів 4.3.Вимоги до умов експлуатації: немає 4.4. Вимоги до складу і параметрів технічних засобів: ПК, вихід до інтернету. 4.5. Вимоги до інформаційної та програмної сумісності: немає 4.6. Вимоги до маркування та упаковки: немає 4.7. Вимоги до транспортування і зберігання: на звичайних носіях інформації 4.8. Спеціальні вимоги: немає.

5.	Вимоги до програмної документації	Програмною документацією до виробу «Методи навчання рекурентної нейронної мережі для вирішення задачі розпізнавання образів» вважати: 1) Справжнє Технічне завдання на розробку виробу (представити у вигляді Додатку Б до пояснювальної записки до кваліфікаційної роботи). 2) Методику дослідження (у вигляді глави 3.3 пояснювальної записки до кваліфікаційної роботи). 3) Опис моделі (представити в Розділі 3 пояснювальної записки до кваліфікаційної роботи). 4) Код програми (представити в Додатку Г до пояснювальної записки до кваліфікаційної роботи).	
6.	Вимоги до техніко-економічних показників	1) Оцінка економічної ефективності – непотрібна.	
7.	Стадії і етапи розробки	Дата	Назва етапу
		від 10 грудня 2021 до 12 січня 2022	Проведення аналізу літературних джерел по методам навчання рекурентної нейронної мережі.
		від 13 січня 2022 до 31 січня 2022	Розробка алгоритму навчання рекурентної нейронної мережі для вирішення задачі розпізнавання образів.
		від 1 лютого 2022 до 27 лютого 2022	Програмна реалізація моделі рекурентної нейронної мережі для вирішення задачі розпізнавання образів.
		від 28 лютого 2022 до 29 березня 2022	Проведення тесту працездатності.
		від 30 березня 2022	Оцінка якості роботи моделі

		до 30 квітня 2022	та порівняння з іншою
		від 1 травня 2022 до 31 травня 2022	Розробка висновків та практичних рекомендацій.
		від 1 червня 2022 до 30 серпня 2022	Написання пояснювальної записки.
		від 31 серпня 2022 до 1 вересня 2022	Представлення роботи науковому керівнику.
		від 1 вересня 2022 до 12 жовтня 2022	Написання звіту з науково-дослідної практики.
		від 13 жовтня 2022 до 20 листопада 2022	Написання звіту з преддипломної практики.
8.	Порядок контролю і приймання програмного продукту (моделі)	1. Перевірку ходу розробки програми виконувати раз в 3 тижні. 2. Захист розробленої моделі провести на засіданні Атестаційної комісії. 3. Пояснювальну записку подати на паперових носіях в 1 примірнику і в електронному вигляді в 1 примірнику на CD-R компакт-диску.	

Виконавець
студент групи КУ- 61
Кондратюк Д. С.
_____

Замовник
д. т. н., професор
Шматков С.І.
_____

ДОДАТОК В

ПРОГРАМА І МЕТОДИКА ВИПРОБУВАНЬ

програмного виробу «Модель рекурентної нейронної мережі на основі популяційного алгоритму»

1 Об'єкт випробувань

1.1 Об'єктом випробувань є модель рекурентної нейронної мережі для вирішення задачі розпізнавання образів.

2. Мета випробувань

Перевірка відповідності функціональності програмної реалізації заявленим функціональним можливостям в технічному завданні (Додаток Б до пояснювальної записки до кваліфікаційної роботи).

3. Загальні положення

3.1 Підставою для проведення випробувань є наказ про призначення атестаційної комісії.

3.2 Місце і тривалість випробувань

Приймальні (приймально-здавальні) випробування проводяться на базі комп'ютерного класу кафедри в період роботи атестаційної комісії.

3.3 Обсяг випробувань

Приймальні випробування програмного виробу проводяться в обсязі відповідному цієї Програми і методики випробувань.

3.4 Організації, які беруть участь у випробуваннях

Приймальні випробування проводяться атестаційною комісією напередодні засідання (або в процесі засідання) за участю Замовника, Виконавця та інших осіб, присутніх на засіданні.

4. Вимоги до програми або програмного виробу

Програма повинна:

1. Представляти з себе модель рекурентної нейронної мережі на основі популяційного алгоритму.
2. Забезпечувати максимальну точність навчання.
3. Програма повинна розпізнавати образи датасету MNIST.
4. Для виконання програми необхідний ПК з доступом до мережі

Інтернет та браузером.

5. Програма працює на різних операційних системах: Linux, Windows та інших.

6. Вимоги до маркування та упаковки (не висуваються).

7. Вимоги до транспортування і зберігання (не висуваються).

8. Спеціальні вимоги (не пред'являються).

5. Вимоги до програмної документації

Склад програмної документації, що подається на випробування, включає:

1) Справжнє Технічне завдання на розробку програмного виробу (представити у вигляді Додатку Б до пояснювальної записки до кваліфікаційної роботи).

2) Програму і методику випробувань розробленого програмного виробу (представити у вигляді Додатку В до пояснювальної записки до кваліфікаційної роботи).

3) Опис моделі (представити в Розділі 3 пояснювальної записки до кваліфікаційної роботи).

4) Код програми (представити в Додатку Г до пояснювальної записки до кваліфікаційної роботи).

6. Засоби і порядок випробувань

6.1 Засоби випробувань

Для виконання програми необхідний ПК з доступом до мережі Інтернет та браузером.

Програма працює на різних операційних системах: Linux, Windows та інших.

Для проведення випробувань необхідно зайти в хмарну платформу Google Colaboratory.

6.2 Порядок проведення випробувань

1. Перевірка програмної документації

1.1. Перевірка складу програмної документації. Перевірку здійснювати за критерієм наявності, представленої в ТЗ документації.

1.2. Перевірка якості програмної документації. Перевірку здійснювати за критерієм відповідності вимогам ГОСТ 19.301-79 ЕСПД. «Програма і методика випробувань».

2. Перевірка працездатності моделі

Тест 1

2.1. Перевірка працездатності

2.1.1. Перевірку здійснювати за критерієм аналізу отриманих даних.

```
Новая наилучшая частица найдена на итерации #487 со среднеквадратической ошибкой: 0.06912542085537125
Новая наилучшая частица найдена на итерации #488 со среднеквадратической ошибкой: 0.06906896262586788
Новая наилучшая частица найдена на итерации #489 со среднеквадратической ошибкой: 0.06897695663912563
Новая наилучшая частица найдена на итерации #490 со среднеквадратической ошибкой: 0.06897237988497951
Новая наилучшая частица найдена на итерации #491 со среднеквадратической ошибкой: 0.06887784279005574
Новая наилучшая частица найдена на итерации #492 со среднеквадратической ошибкой: 0.06884918003498144
Новая наилучшая частица найдена на итерации #495 со среднеквадратической ошибкой: 0.06880232616384611
Новая наилучшая частица найдена на итерации #496 со среднеквадратической ошибкой: 0.06867739746741888
```

Рисунок В.1 – Робота моделі.

Тест 2

2.2. Перевірка отриманих результатів.

2.2.1. Перевірку здійснювати за критерієм аналізу якості розпізнавання.

	precision	recall	f1-score	support
0	0.98	0.90	0.94	52
1	0.00	0.00	0.00	41
2	0.66	0.71	0.78	45
3	0.00	0.00	0.00	44
4	0.92	0.75	0.83	48
5	0.00	0.00	0.00	41
6	0.95	0.81	0.88	48
7	0.00	0.00	0.00	41
8	0.00	0.00	0.00	49
9	0.00	0.00	0.00	41
micro avg	0.93	0.34	0.50	450
macro avg	0.37	0.32	0.34	450
weighted avg	0.40	0.34	0.37	450
samples avg	0.34	0.34	0.34	450

Рисунок В.2 – Результати розпізнавання.

Висновки: випробування вважаються успішно пройденими, якщо були виконані всі перевірки з пунктів 1,2.

Виконавець
студент групи КУ-61
Кондратюк Д.С.
_____

Замовник
д.т.н., професор
Шматков С.І.
_____

ДОДАТОК Г

Лістинг коду програмної моделі на основі генетичного алгоритму

```
import numpy as np
import pandas as pd
import tensorflow as tf
import matplotlib as plt
from tensorflow.keras.layers import Input, LSTM, Dropout, Dense, BatchNorm
alization
from tensorflow.keras import optimizers, losses, metrics, models, Sequenti
al

def load_data():
        (x_train, y_train), (x_test, y_test) = tf.keras.datasets.mnist.loa
d_data()

        x_train, x_test = x_train / 255.0, x_test / 255.0

    return x_train,x_test,y_test,y_train


def lstm_mode(inputs, units_num, sequences_state):
    lstm = LSTM(units_num, return_sequences=sequences_state)(inputs)
    print("lstm: ", lstm.shape)
     return lstm


 def dense_mode(input, units_num):

     dense = Dense(units_num, kernel_regularizer=tf.keras.regularizers.l2(0
 .001), activation='relu')(input)
     print("dense: ", dense.shape)
         drop_out = Dropout(rate=0.2)(dense)
         dense_bn = BatchNormalization()(drop_out)
     return dense, drop_out, dense_bn


 def aim_function(x_train, y_train, x_test, y_test, num):

     lstm_layers = num[0]
         lstm_units = num[2:2 + lstm_layers]
         lstm_name = list(np.zeros((lstm_layers,)))
     lstm_dense_layers = num[1]
```

```
        lstm_dense_units = num[2 + lstm_layers: 2 + lstm_layers + lstm_den
se_layers]
    lstm_dense_name = list(np.zeros((lstm_dense_layers,)))
    lstm_dense_dropout_name = list(np.zeros((lstm_dense_layers,)))
    lstm_dense_batch_name = list(np.zeros((lstm_dense_layers,)))
        inputs_lstm = Input(shape=(x_train.shape[1], x_train.shape[2]))

        for i in range(lstm_layers):
        if i == 0:
            inputs = inputs_lstm
        else:
            inputs = lstm_name[i - 1]
        if i == lstm_layers - 1:
            sequences_state = False
        else:
            sequences_state = True
                lstm_name[i] = lstm_mode(inputs, lstm_units[i], sequences_
state=sequences_state)

        for i in range(lstm_dense_layers):
        if i == 0:
            inputs = lstm_name[lstm_layers - 1]
        else:
            inputs = lstm_dense_name[i - 1]

        lstm_dense_name[i], lstm_dense_dropout_name[i], lstm_dense_batch_n
ame[i] = dense_mode(inputs, units_num=lstm_dense_units[i])

        outputs_lstm = Dense(10, activation='softmax')(lstm_dense_batch_na
me[lstm_dense_layers - 1])
    print("last_dense ", outputs_lstm.shape)

    LSTM_model = tf.keras.Model(inputs=inputs_lstm, outputs=outputs_lstm)
        LSTM_model.compile(optimizer=optimizers.Adam(),
                           loss='sparse_categorical_crossentropy',
                           metrics=['accuracy'])
    print("Training set shape ", x_train.shape)

    history = LSTM_model.fit(x_train, y_train,batch_size=32, epochs=1, val
idation_split=0.1, verbose=1)
        acc = LSTM_model.evaluate(x_test, y_test, verbose=0)
    return acc[1]
```

```
import numpy as np
import pandas as pd
import matplotlib as plt
import os

os.environ['TF_CPP_MIN_LOG_LEVEL'] = '2'

DNA_size = 2
DNA_size_max = 8
POP_size = 20
CROSS_RATE = 0.5
MUTATION_RATE = 0.01
N_GENERATIONS = 40


x_train, x_test, y_test, y_train = load_data()

def get_fitness(x):
    return aim_function(x_train, y_train, x_test, y_test, num=x)

def select(pop, fitness):
    idx = np.random.choice(np.arange(POP_size), size=POP_size, replace=Tru
e, p=fitness / fitness.sum())
    return pop[idx]

def crossover(parent, pop):

    if np.random.rand() < CROSS_RATE:
                i_ = np.random.randint(0, POP_size, size=1)
        cross_points = np.random.randint(0,2,size=DNA_size_max).astype(np.
bool)

        for i, point in enumerate(cross_points):

            if point == True and pop[i_, i] * parent[i] == 0:
                cross_points[i] = False

            if point == True and i < 2:
                cross_points[i] = False

        parent[cross_points] = pop[i_, cross_points]
    return parent


def mutate(child):
        for point in range(DNA_size_max):
```

```
        if np.random.rand() < MUTATION_RATE:
            if point >= 2:
                if child[point] != 0:
                    child[point] = np.random.randint(32, 257)
    return child

pop_layers = np.zeros((POP_size, DNA_size), np.int32)
pop_layers[:, 0] = np.random.randint(1, 4, size=(POP_size,))
pop_layers[:, 1] = np.random.randint(1, 4, size=(POP_size,))

pop = np.zeros((POP_size, DNA_size_max))
for i in range(POP_size):
        pop_neurons = np.random.randint(32, 257, size=(pop_layers[i].sum()
,))
    pop_stack = np.hstack((pop_layers[i], pop_neurons))
        for j, gene in enumerate(pop_stack):
        pop[i][j] = gene

for each_generation in range(N_GENERATIONS):
    fitness = np.zeros([POP_size,])
    for i in range(POP_size):
        pop_list = list(pop[i])
        for j,each in enumerate(pop_list):
            if each == 0.0:
                index = j
                pop_list = pop_list[:j]
        for k,each in enumerate(pop_list):
            each_int = int(each)
            pop_list[k] = each_int
                fitness[i] = get_fitness(pop_list)
        print('The first %d Dai di %d The fitness of chromosomes is %f' %
(each_generation+1, i+1, fitness[i]))
        print('This chromosome is:', pop_list)
    print('Generation: ', each_generation + 1, ' Most fitted DNA: ', pop[n
p.argmax(fitness), :], ' The fitness is: ', fitness[np.argmax(fitness)])

    pop = select(pop,fitness)

    pop_copy = pop.copy()
    for parent in pop:
        child = crossover(parent,pop_copy)
        child = mutate(child)
        parent = child
```

Лістинг коду програмної моделі на основі популяційного алгоритму

```
import functools
import numpy as np
import sklearn.metrics
import scipy.special
import sklearn.datasets
import sklearn.model_selection


def minimize_pso(cost_func, num_dimensions, num_iterations):
    num_particles = num_dimensions * 2
    swarm = ParticleSwarm(cost_func, num_dimensions, num_particles)
    return swarm.minimize(num_iterations)


class PSOResult(object):
    def __init__(self, best_particle, best_score, num_iterations):
        self.best_particle = best_particle
        self.best_score = best_score
        self.num_iterations = num_iterations


class ParticleSwarm(object):
    def __init__(self, cost_func, num_dimensions, num_particles, chi=0.729
84, phi_p=2.05, phi_g=2.05):
        self.cost_func = cost_func
        self.num_dimensions = num_dimensions

        self.num_particles = num_particles
        self.chi = chi
        self.phi_p = phi_p
        self.phi_g = phi_g

        self.X = np.random.uniform(size=(self.num_particles, self.num_dime
nsions))
        self.V = np.random.uniform(size=(self.num_particles, self.num_dime
nsions))

        self.P = self.X.copy()
        self.S = self.cost_func(self.X)
        self.g = self.P[self.S.argmin()]
        self.best_score = self.S.min()
```

```
    def _update(self):
        R_p = np.random.uniform(size=(self.num_particles, self.num_dimensi
ons))
        R_g = np.random.uniform(size=(self.num_particles, self.num_dimensi
ons))

        self.V = self.chi * (self.V \
                + self.phi_p * R_p * (self.P - self.X) \
                + self.phi_g * R_g * (self.g - self.X))

        self.X = self.X + self.V

        scores = self.cost_func(self.X)

        better_scores_idx = scores < self.S
        self.P[better_scores_idx] = self.X[better_scores_idx]
        self.S[better_scores_idx] = scores[better_scores_idx]

        self.g = self.P[self.S.argmin()]
        self.best_score = self.S.min()

    def minimize(self, max_iter):
        for i in range(max_iter):
            self._update()

        return PSOResult(
            best_particle=self.g,
            best_score=self.best_score,
            num_iterations=max_iter
        )


class RNN:
    def __init__(self, shape, weights=None):
        self.shape = shape
        self.num_layers = len(shape)
        if weights is None:
            self.weights = []
            for i in range(self.num_layers-1):
                W = np.random.uniform(size=(self.shape[i+1], self.shape[i]
 + 1))
                self.weights.append(W)
        else:
            self.weights = weights
```

```
    def run(self, data):
        layer = data.T
        for i in range(self.num_layers-1):
            prev_layer = np.insert(layer, 0, 1, axis=0)
            o = np.dot(self.weights[i], prev_layer)
            layer = scipy.special.expit(o)
        return layer


def dim_weights(shape):
    dim = 0
    for i in range(len(shape)-1):
        dim = dim + (shape[i] + 1) * shape[i+1]
    return dim

def weights_to_vector(weights):
    w = np.asarray([])
    for i in range(len(weights)):
        v = weights[i].flatten()
        w = np.append(w, v)
    return w
def vector_to_weights(vector, shape):
    weights = []
    idx = 0
    for i in range(len(shape)-1):
        r = shape[i+1]
        c = shape[i] + 1
        idx_min = idx
        idx_max = idx + r*c
        W = vector[idx_min:idx_max].reshape(r,c)
        weights.append(W)
    return weights

def eval_neural_network(weights, shape, X, y):
    mse = np.asarray([])
    for w in weights:
        weights = vector_to_weights(w, shape)
        nn = RNN(shape, weights=weights)
        y_pred = nn.run(X)
        mse = np.append(mse, sklearn.metrics.mean_squared_error(np.atleast
_2d(y), y_pred))
    return mse

def print_best_particle(best_particle):
```

```
swarm = ParticleSwarm(cost_func, num_dimensions=dim_weights(shape), num_pa
rticles=30)

i = 0
best_scores = [(i, swarm.best_score)]
print_best_particle(best_scores[-1])
while swarm.best_score>1e-6 and i<500:
    swarm._update()
    i = i+1
    if swarm.best_score < best_scores[-1][1]:
        best_scores.append((i, swarm.best_score))
        print_best_particle(best_scores[-1])

best_weights = vector_to_weights(swarm.g, shape)
best_nn = RNN(shape, weights=best_weights)
y_test_pred = np.round(best_nn.run(X_test))
print(sklearn.metrics.classification_report(y_test_true, y_test_pred.T))


    print("Новая наилучшая частица найдена на итерации #{i} со среднеквадр
атической ошибкой: {score}".format(i=best_particle[0], score=best_particle
[1]))


num_classes = 10
mnist = sklearn.datasets.load_digits()
X, X_test, y, y_test = sklearn.model_selection.train_test_split(mnist.data
, mnist.target)

num_inputs = X.shape[1]

y_true = np.zeros((len(y), num_classes))
for i in range(len(y)):
    y_true[i, y[i]] = 1

y_test_true = np.zeros((len(y_test), num_classes))
for i in range(len(y_test)):
    y_test_true[i, y_test[i]] = 1

shape = (num_inputs, 64, 32, num_classes)

cost_func = functools.partial(eval_neural_network, shape=shape, X=X, y=y_t
rue.T)
```