ПОЛТАВСЬКИЙ УНІВЕРСИТЕТ ЕКОНОМІКИ І ТОРГІВЛІ
Навчально-науковий інститут денної освіти
Форма навчання денна

Кафедра комп'ютерних наук та інформаційних технологій

Допускається до захисту
Завідувач кафедри
_____Олена ОЛЬХОВСЬКА
*(підпис)*

«___»____202_ р.

# КВАЛІФІКАЦІЙНА РОБОТА

**на тему**

**«Розробка програми для демонстрації навчання штучної нейронної мережі на прикладі задачі розпізнавання цифр»**

**зі спеціальності 122 Комп'ютерні науки**
**освітня програма «Комп'ютерні науки»**
**ступеня бакалавра**

**Виконавець роботи** Наконечний Костянтин Романович

_____«___»_____202_ р.
*(підпис)*

**Науковий керівник** к. ф.-м. н. Олексійчук Юрій Федорович

_____«___»_____202_ р.
*(підпис)*

**Рецензент** _____

**ПОЛТАВА 2025**

# РЕФЕРАТ

Наконечний Костянтин Романович - Тема кваліфікаційної роботи - Штучна нейронна мережа, розпізнавання зображень, навчання, графічний інтерфейс, C++, Qt. (ПУЕТ).

Записка викладена на 63 сторінки., містить 15 рисунків ,44 літературне джерело, 2 додатки.

**Об'єкт** - демонстраційне навчальне програмне забезпечення.

**Предмет розробки**– програмне забезпечення для демонстрації процесу навчання штучної нейронної мережі на прикладі задачі розпізнавання рукописних цифр.

**Мета роботи** – розробити програму, яка наочно демонструє процес навчання простої штучної нейронної мережі для класифікації зображень цифр, з можливістю візуалізації її роботи та результатів навчання.

**Методи дослідження** – теоретичний аналіз принципів роботи штучних нейронних мереж, розробка алгоритму навчання, програмна реалізація у середовищі C++/Qt, тестування розробленого додатку.

Проведено аналіз відомих методів для створення штучних нейронних мереж, виявлено позитивні та негативні сторони. Розроблено алгоритм роботи програми з теми «Розробка елементів програми для демонстрації навчання штучної нейронної мережі на прикладі задачі розпізнавання цифр», побудовано його блок-схему. Програмно реалізовано.

**ЗМІСТ**

# ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

| Умовні позначення, символи, скорочення, терміни | Пояснення умовних позначень, символів, скорочень |
|---|---|
| ANN | (Artificial Neural Network) штучна нейронна мережа |
| C++ | Мова програмування загального призначення з підтримкою об’єктно-орієнтованого програмування. |
| Qt | Кросплатформовий інструментарій для створення графічних інтерфейсів користувача |
| MNIST | (Modified National Institute of Standards and Technology) база даних зображень рукописних цифр, що використовується для навчання та тестування моделей розпізнавання |
| GUI | (Graphical User Interface) графічний інтерфейс користувача |

# ВСТУП

Сучасний розвиток технологій у сфері штучного інтелекту й машинного навчання відкриває нові можливості не лише для вирішення практичних задач, а й для створення інструментів, що сприяють кращому розумінню принципів роботи інтелектуальних систем. Штучні нейронні мережі (ШНМ), як один із найважливіших інструментів машинного навчання, широко застосовуються у завданнях розпізнавання образів, зокрема рукописних цифр. Однак попри успішність у прикладних задачах, принципи функціонування ШНМ часто залишаються складними для засвоєння студентами.

**Метою роботи** є створення навчального демонстраційного програмного забезпечення, яке дозволить наочно показати процес навчання та функціонування штучної нейронної мережі на прикладі задачі розпізнавання цифр. Такий підхід сприятиме кращому розумінню теоретичних аспектів та викликатиме інтерес до вивчення цієї тематики.

**Об’єкт дослідження** – процес візуалізації роботи штучної нейронної мережі в навчальному програмному забезпеченні.

**Предмет дослідження** – програмна реалізація та алгоритмічні основи демонстрації навчання ШНМ для задачі розпізнавання рукописних цифр.

Розроблене програмне забезпечення дозволяє користувачу вводити зображення цифри, запускати процес розпізнавання та бачити результат передбачення у зручній графічній формі. Ця система може використовуватись у навчальному процесі як ілюстративний матеріал для викладання основ нейронних мереж та машинного навчання.

**Структура роботи**

Робота складається з чотирьох основних розділів:

- **Розділ 1. Постановка задачі** – викладено мету, завдання, об’єкт і предмет дослідження, а також визначено актуальність теми.

- **Розділ 2. Інформаційний огляд** – представлено короткий аналіз існуючих підходів до реалізації нейронних мереж, приклади подібних систем та обґрунтовано вибір архітектури.
- **Розділ 3. Теоретична частина** – описано основи роботи штучної нейронної мережі, методи її навчання та обробки вхідних даних.
- **Розділ 4. Практична частина** – розглянуто реалізацію програмного забезпечення, структуру програми, графічний інтерфейс користувача та результати її роботи
- **Висновки**
- **Літературні джерела**

# РОЗДІЛ 1. ПОСТАНОВКА ЗАДАЧІ

Штучні нейронні мережі (ШНМ) є одним із найпотужніших інструментів сучасного машинного навчання. Вони активно застосовуються для вирішення задач класифікації, регресії, розпізнавання образів та інших інтелектуальних завдань. Однією з класичних задач, що широко використовується як у дослідженнях, так і в навчальних цілях, є задача розпізнавання рукописних цифр.[1,2]

Штучні нейронні мережі (ШНМ) — це обчислювальні моделі, натхненні біологічною будовою та функціонуванням мозку людини. Вони складаються з великої кількості взаємозв’язаних обчислювальних елементів — нейронів, які згруповані в шари: вхідний, приховані та вихідний.[3-5]

Кожен штучний нейрон виконує просту математичну операцію: приймає значення на входах, зважує їх за допомогою коефіцієнтів (ваг), обчислює суму та застосовує до неї активаційну функцію. Найпоширенішими активаційними функціями є: сигмоїда, ReLU (Rectified Linear Unit), tanh.

Основні типи нейронних мереж:

- Прямого поширення (Feedforward Neural Networks) — найпростіший тип, інформація проходить тільки в одному напрямку.
- Згорткові нейронні мережі (CNN) — застосовуються для обробки зображень.
- Рекурентні нейронні мережі (RNN) — використовуються для роботи з послідовностями (наприклад, текстами чи часовими рядами).
- Генеративно-змагальні мережі (GAN) — створюють нові дані на основі навченої моделі.

ШНМ знаходять широке застосування в багатьох галузях, зокрема:

- Медицина: діагностика захворювань за зображеннями (рентген, МРТ), прогнозування стану пацієнтів.

- Промисловість: контроль якості продукції, прогнозування несправностей обладнання.
- Фінанси: виявлення шахрайських операцій, прогнозування біржових коливань.
- Транспорт: автономне керування транспортними засобами.
- Агропромисловість: моніторинг стану ґрунтів, прогноз урожайності.
- Обробка природної мови: переклад, розпізнавання мови, чат-боти.

Процес створення нейронної мережі включає кілька основних етапів:

1. Збір і підготовка даних: очищення, нормалізація, анотація даних, розбиття на тренувальну та тестову вибірки.
2. Вибір архітектури: визначення кількості шарів, кількості нейронів, типів активаційних функцій.
3. Навчання мережі: застосування алгоритмів оптимізації (наприклад, градієнтного спуску) для налаштування ваг мережі на основі навчальної вибірки.
4. Оцінювання моделі: використання метрик якості (точність, повнота, F-мера, MSE) для тестування продуктивності мережі.
5. Впровадження та експлуатація: інтеграція у програмне середовище або апаратну систему.

Сьогодні існує багато інструментів і фреймворків для створення ШНМ, зокрема:

- TensorFlow, Keras, PyTorch — для Python;
- MATLAB — з вбудованими модулями для нейромереж;
- ONNX, OpenVINO — для інтеграції в мобільні та вбудовані системи.

Актуальність цієї задачі обумовлена не лише її теоретичним значенням, а й широким спектром практичного застосування — від автоматичного розпізнавання поштових індексів до систем оптичного розпізнавання символів у фінансових, медичних та освітніх системах. [6,7]

У межах цієї роботи ставиться завдання створення навчального програмного забезпечення, яке дозволяє в інтерактивному режимі

ознайомитись із процесом навчання ШНМ та оцінити її здатність до класифікації цифр. Програма має дозволяти користувачу вручну вводити зображення цифри, яке далі подається на вхід навченої мережі для класифікації. Крім того, передбачено можливість збереження ваг мережі після навчання та їх завантаження під час наступних запусків, що значно оптимізує час роботи.

З технічного боку, реалізація проекту здійснюється засобами мови програмування C++ із використанням фреймворку Qt для побудови графічного інтерфейсу. Для навчання нейронної мережі використовується класичний датасет MNIST, що містить тисячі зображень рукописних цифр та відповідає всім вимогам для навчання та тестування моделей машинного навчання.

Результатом реалізації має стати повноцінний програмний інструмент з графічним інтерфейсом, що ілюструє принципи роботи нейронної мережі, демонструє процес її навчання та дозволяє оцінити ефективність класифікації цифр на практиці.

# РОЗДІЛ 2. ІНФОРМАЦІЙНИЙ ОГЛЯД

## 2.1 Огляд штучних нейронних мереж та їхні основні принципи

Нейронна мережа, також штучна нейронна мережа, або ШНМ – це один із напрямків у створенні штучного інтелекту, який ставить своїм завданням відтворити аналітичну роботу людського мозку [8].

Математична модель штучної нейронної мережі (ШНМ) бере свій початок із біології та нейрофізіології, зокрема з досліджень функціонування нервової системи живих організмів. Основою для її створення став принцип роботи біологічних нейронів — клітин мозку, які взаємодіють між собою через синапси, формуючи складну мережу обробки й передачі інформації. У живих організмах нейрони отримують вхідні сигнали у вигляді електрохімічних імпульсів, обробляють їх, і залежно від сили сигналу та порогових значень, або передають імпульс далі, або пригнічують його [9-11].

Математично нейронна мережа моделюється у вигляді системи з'єднаних між собою штучних нейронів, кожен з яких виконує просту операцію: обчислює зважену суму вхідних сигналів, застосовує до неї функцію активації та передає результат далі. Такі моделі дозволяють наближено повторити механізм обробки інформації у мозку, що дає змогу розв’язувати складні задачі класифікації, розпізнавання образів, прогнозування тощо.

Апаратна реалізація нейромереж — це фізичне втілення моделі у вигляді мережі обчислювальних елементів або процесорів, що імітують поведінку біологічних нейронів. Кожен такий осередок виконує функцію аналізу й передачі інформації за принципом "вхід–обробка–вихід", а зв’язки між ними відповідають синапсам. У подібних системах один "нейрон" може мати десятки й сотні "синапсів" — каналів зв'язку з іншими елементами, через які здійснюється обмін даними. Перевагою таких реалізацій є здатність до паралельної обробки великих обсягів інформації, що значно підвищує

продуктивність нейромереж при виконанні ресурсоємних завдань, таких як обробка зображень або природної мови. [12-14]

На практиці, ШНМ реалізуються як програмні моделі або спеціалізовані апаратні пристрої (наприклад, нейропроцесори, графічні процесори (GPU), або спеціалізовані інтегральні схеми (ASIC)), здатні забезпечити високу швидкість навчання й обробки даних. Саме завдяки поєднанню біологічного натхнення та обчислювальних технологій, штучні нейронні мережі стали основою для сучасного штучного інтелекту. [15-16]

Кожен з цих зв’язків має певний коефіцієнт, або вагу, на яку множиться значення, що надходить на неї. Таким чином те, що відбувається з інформацією у нейромережі, визначається конфігурацією осередків і вагою міжнейронних зв’язків.

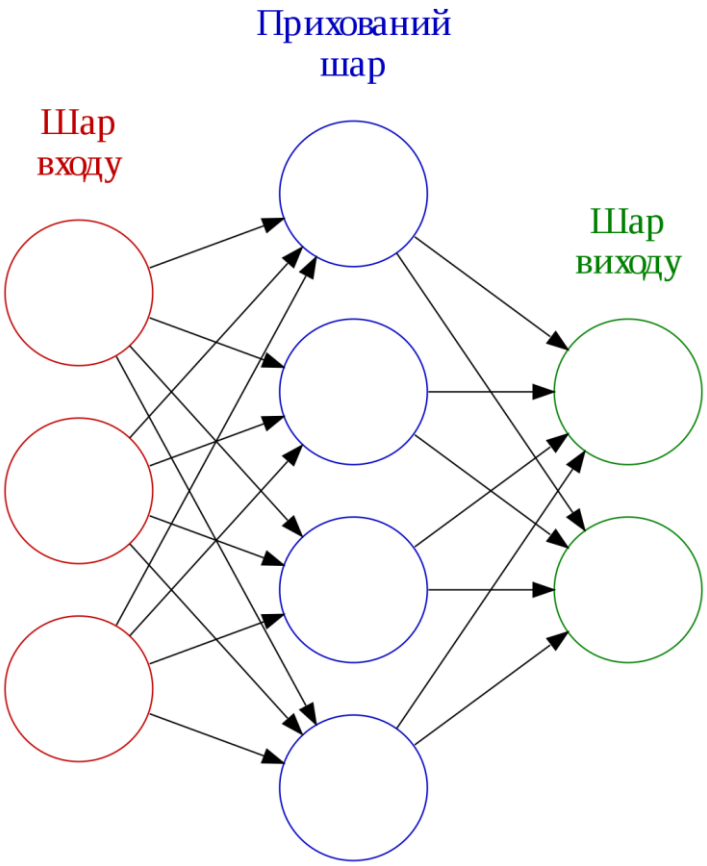



Рисунок 2.1– Шаблон нейронної мережі

Розвиток інтернету та процеси глобалізації сприяли тому, що з’явилося дуже багато інформації, опрацювати яку самотужки людина фізично не в змозі. Нейронні мережі знайшли застосування у:

- аналізі та класифікуванні даних за заданими параметрами;
- формуванні аналітичних прогнозів, керуючись вхідною інформацією;
- порівнянні та розпізнаванні ідентичних даних.

Нейронні мережі навчаються шляхом обробки прикладів, кожен з яких містить відомий «вхід» та «результат», утворюючи ймовірнісно зважені асоціації між ними, які зберігаються в структурі даних самої мережі. Тренування нейронної мережі заданим прикладом зазвичай здійснюють шляхом визначення різниці між обробленим виходом мережі і цільовим виходом. Ця різниця є похибкою.

Потім мережа підлаштовує свої зважені асоціації відповідно до правила навчання і з використанням цього значення похибки. Послідовні підлаштовування призведуть до вироблення нейронною мережею результатів, усе більше схожих на цільові. Після достатньої кількості цих підлаштовувань, тренування можливо припинити на основі певного критерію. Це форма керованого навчання.

Такі системи «навчаються» виконувати завдання, розглядаючи приклади, як правило, без програмування правил для конкретних завдань. Наприклад, у розпізнаванні зображень вони можуть навчитися встановлювати зображення, на яких зображені коти, аналізуючи приклади зображень, мічені вручну як «кіт» та «не кіт», і використовуючи результати для ідентифікування котів на інших зображеннях. Вони роблять це без будь-якого апріорного знання про котів, наприклад, що вони мають хутро, хвости, вуса та котоподібні писки. Натомість, вони автоматично породжують ідентифікаційні характеристики з прикладів, які обробляють.

## 2.2 Аналіз різних архітектур нейронних мереж

Штучні нейронні мережі еволюціонували у широке сімейство методик, які вдосконалили рівень останніх досягнень у багатьох областях. Найпростіші типи мають один або кілька статичних складових, включно з кількістю вузлів, кількістю шарів, вагами вузлів і топологією. Динамічні типи дозволяють одному або декільком із них еволюціонувати шляхом навчання. Останнє набагато складніше, але може скорочувати періоди навчання й давати кращі результати. Деякі типи дозволяють/вимагають навчання «під керуванням» оператора, тоді як інші працюють незалежно.

Деякі типи працюють виключно апаратно, тоді як інші є суто програмними й працюють на комп'ютерах загального призначення.

Типи і різновиди нейронних мереж:

- Нейронна мережа прямого поширення (Feedforward Neural Networks):

Нейронна мережа прямого зв’язку є одним із найпростіших типів штучних нейронних мереж. У цій мережі інформація рухається тільки в одному напрямку — вперед — від вхідних вузлів, через приховані вузли (якщо такі є) і до вихідних вузлів. У мережі немає циклів і петель. Нейронні мережі прямого зв’язку були першим винайденим типом штучних нейронних мереж і є простішими за свої аналоги, такі як рекурентні нейронні мережі та згорткові нейронні мережі. Навчання прямої нейронної мережі передбачає використання набору даних для налаштування вагових коефіцієнтів зв’язків між нейронами. Це робиться за допомогою ітераційного процесу, коли набір даних передається через мережу кілька разів, і щоразу ваги оновлюються, щоб зменшити помилку прогнозу. Цей процес відомий як градієнтний спуск і триває, доки мережа не буде задовільно працювати з навчальними даними. Ефективне для задач класифікації, але може не враховувати просторові залежності у вхідних зображеннях.

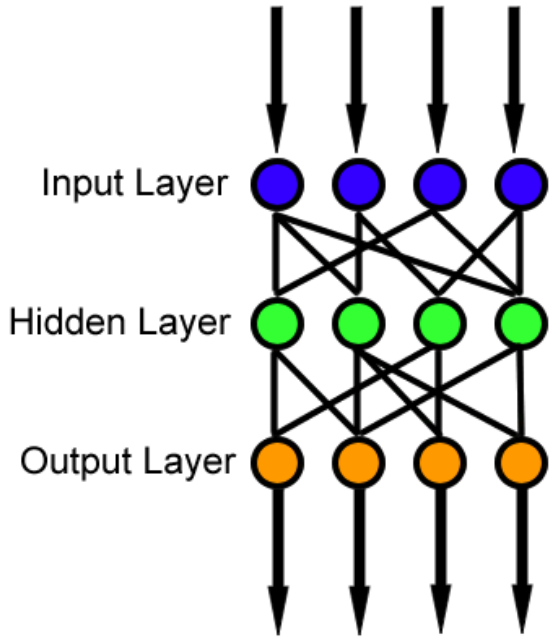



Рисунок 2.2 – Приклад нейронної мережі прямого зв’язку

- Згорткові нейронні мережі (Convolutional Neural Networks - CNN):

Згорткова нейронна мережа — це алгоритм глибокого навчання, який може сприймати вхідне зображення, призначати важливість (вагові значення та зміщення) різним аспектам/об’єктам на зображенні та мати можливість відрізнити один від іншого. Попередня обробка, необхідна в ConvNet, набагато менша порівняно з іншими алгоритмами класифікації. У той час як у примітивних методах фільтри розробляються вручну, після достатнього навчання, ConvNets мають можливість вивчати ці фільтри/характеристики. Згорткові нейронні мережі відрізняються від інших нейронних мереж своєю чудовою продуктивністю з зображенням, мовленням або аудіосигналом.

Широко використовується для класифікації та локалізації об'єктів на зображеннях, включаючи цифри.

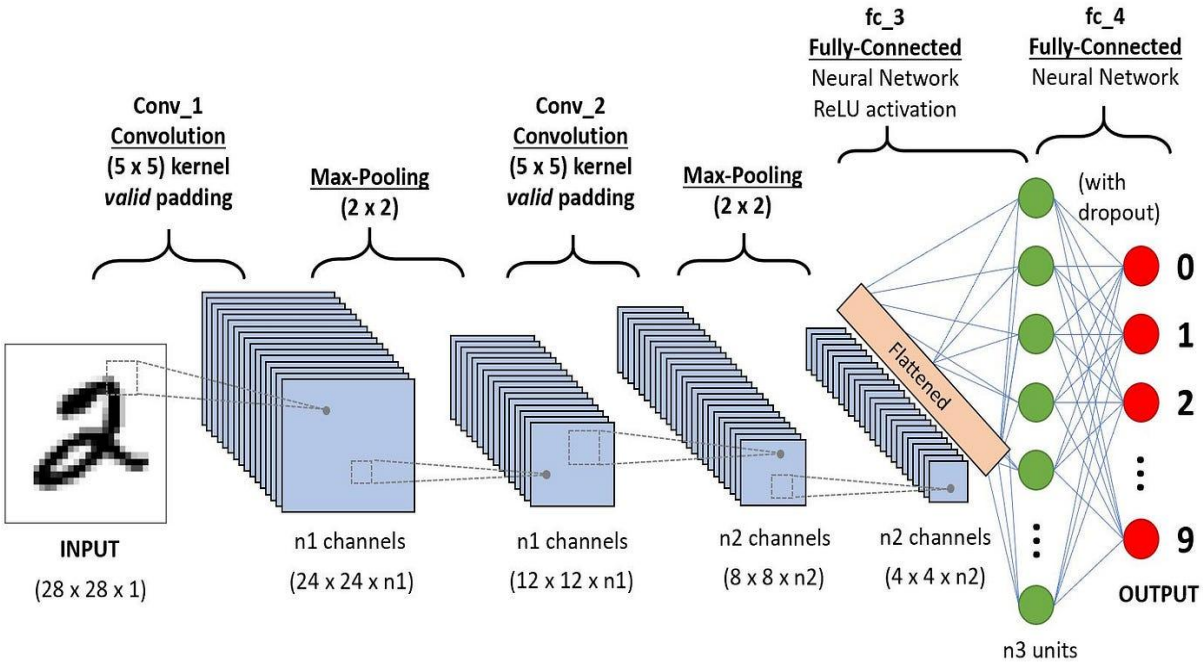



Рисунок 2.3 – Приклад згорткової нейронної мережі

- Рекурентні нейронні мережі (Recurrent Neural Networks - RNN):

Рекурентна нейронна мережа (RNN) — це тип штучної нейронної мережі, яка використовує послідовні дані або дані тимчасових рядів. Ці алгоритми глибокого навчання зазвичай використовуються для порядкових або тимчасових проблем, таких як переклад мови, обробка природної мови (nlp), розпізнавання мовлення та підписи до зображень; вони включені в такі популярні програми, як Siri, голосовий пошук і Google Translate. Подібно до нейронних мереж прямого зв’язку та згорткових нейронних мереж (CNN), рекурентні нейронні мережі використовують навчальні дані для навчання. Вони відрізняються своєю «пам’яттю», оскільки вони беруть інформацію з попередніх вхідних даних, щоб впливати на поточні вхідні та вихідні дані. Іншою відмінною характеристикою рекурентних мереж є те, що вони спільно використовують параметри на кожному рівні мережі. У той час як мережі прямого зв’язку мають різні ваги для кожного вузла, рекурентні нейронні мережі мають однаковий параметр ваги на кожному рівні мережі. Тим не менш, ці ваги все ще коригуються за допомогою процесів зворотного

поширення та градієнтного спуску, щоб полегшити навчання з підкріпленням.

Ефективні в задачах, де важливий порядок елементів, наприклад, у розпізнаванні символів.

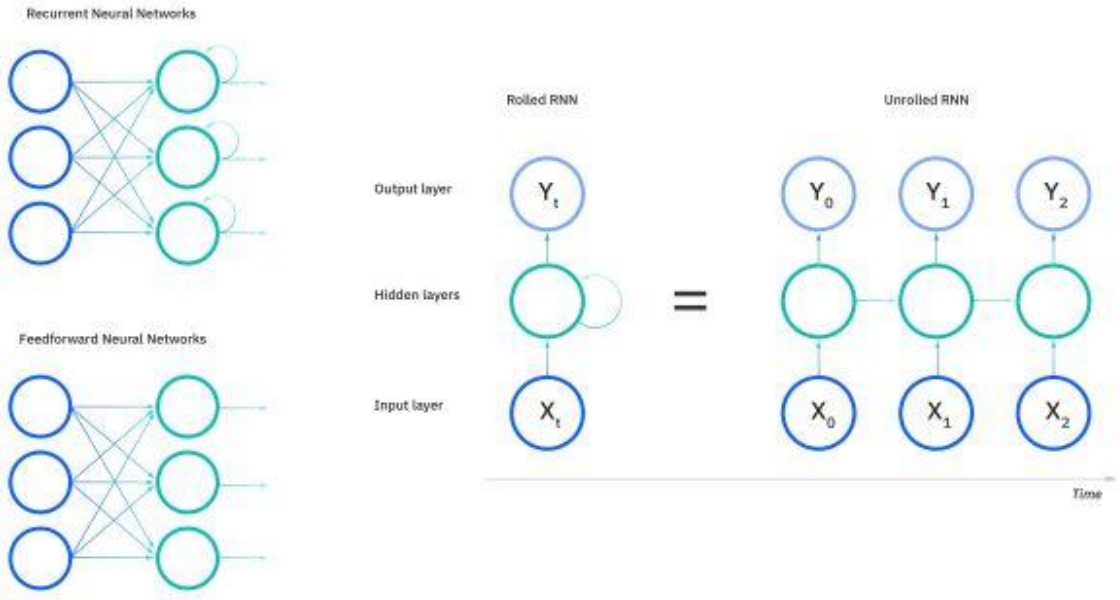



Рисунок 2.4 – Приклад рекурентної нейронної мережі

- Глибокі нейронні мережі (Deep Neural Networks - DNN):

Глибинна нейронна мережа імітує роботу людського мозку. Загалом, вона має вхідний рівень, вихідний рівень і принаймні один проміжний рівень. Чим більше кількість шарів, тим глибша мережа. Кожен із цих рівнів виконує різні типи специфічного сортування та категоризації в процесі, який називається « ієрархією функцій ».

Щоб краще зрозуміти, як працює глибока нейронна мережа, нам потрібно лише спостерігати, як працює людський мозок . Замість того, щоб вивчати структуру обличчя, щоб ідентифікувати людей, наш мозок навчається на відхиленні основного обличчя, яке служить моделлю. Коли ми бачимо обличчя, мозок намагається визначити, чим воно відрізняється від цієї еталонної моделі. Таким чином, такі елементи, як очі, вуха або брови,

переглядаються за частки секунди. Відмінності між сприйнятим обличчям і «базовою» моделлю обличчя кількісно визначаються електричним сигналом різної сили . Усі відхилення комбінуються для отримання результату. Окремі вузли в системі схожі на нейрони в мозку людини. Кожен шар складається з кількох вузлів . Як тільки їх торкаються подразники, запускається процес.

Нейронна мережа інтерпретує дані, зібрані датчиками або безпосередньо введені програмістом. Ці дані можуть бути зображеннями, текстами або навіть звуками, які потім будуть перетворені в числові значення.

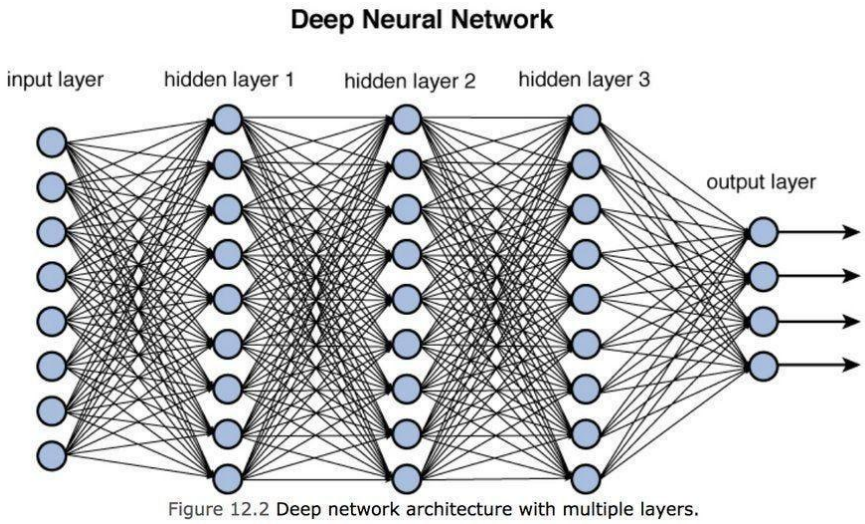



Рисунок 2.5 – Приклад глибокої нейронної мережі

## 2.3 Розгляд існуючих програм для демонстрації роботи нейронних мереж

Актуальність застосування нейронних мереж у сучасному світі виявляється не лише у практичних застосуваннях, але й у сферах творчості та навчання. Зростання інтересу до штучного інтелекту спричинило розширення ринку програм, які дозволяють користувачам демонструвати та експериментувати з нейронними мережами. У цьому розділі роботи буде

проведено огляд існуючих програм, які відображають різноманіття застосувань нейронних мереж у різних контекстах.

- TensorFlow Playground:

TensorFlow Playground є інтерактивним інструментом, що дозволяє користувачам експериментувати з нейронними мережами в онлайн-середовищі[3]. Він забезпечує можливість змінювати параметри мережі та спостерігати за змінами в результатах навчання.

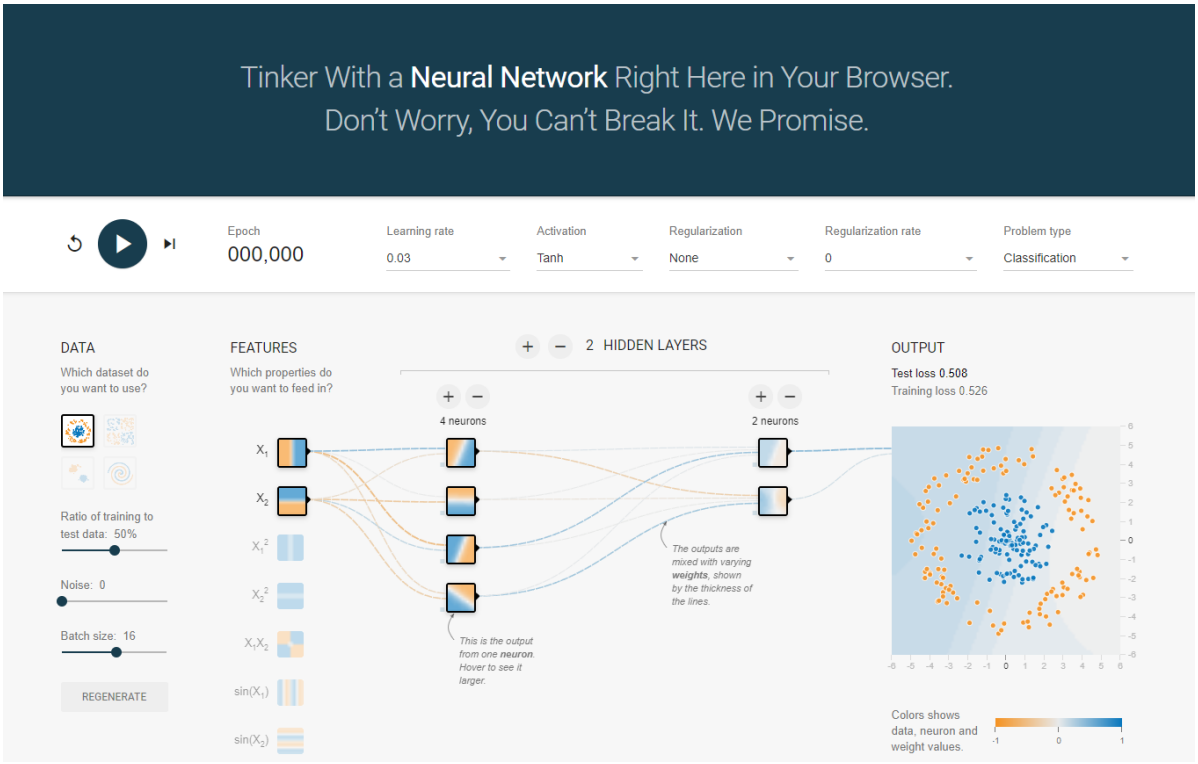



Рисунок 3.6 – TensorFlow Playground

Інтуїтивний інтерфейс, візуалізація процесу навчання, підходить для вивчення основ нейронних мереж.

- Deep Dream:

Розроблений Google, Deep Dream використовує заглиблення нейронних мереж для генерації унікальних та псевдо-художніх зображень[4]. Програма надає користувачеві можливість експериментувати з навчальними моделями та спостерігати за їхньою творчою роботою.

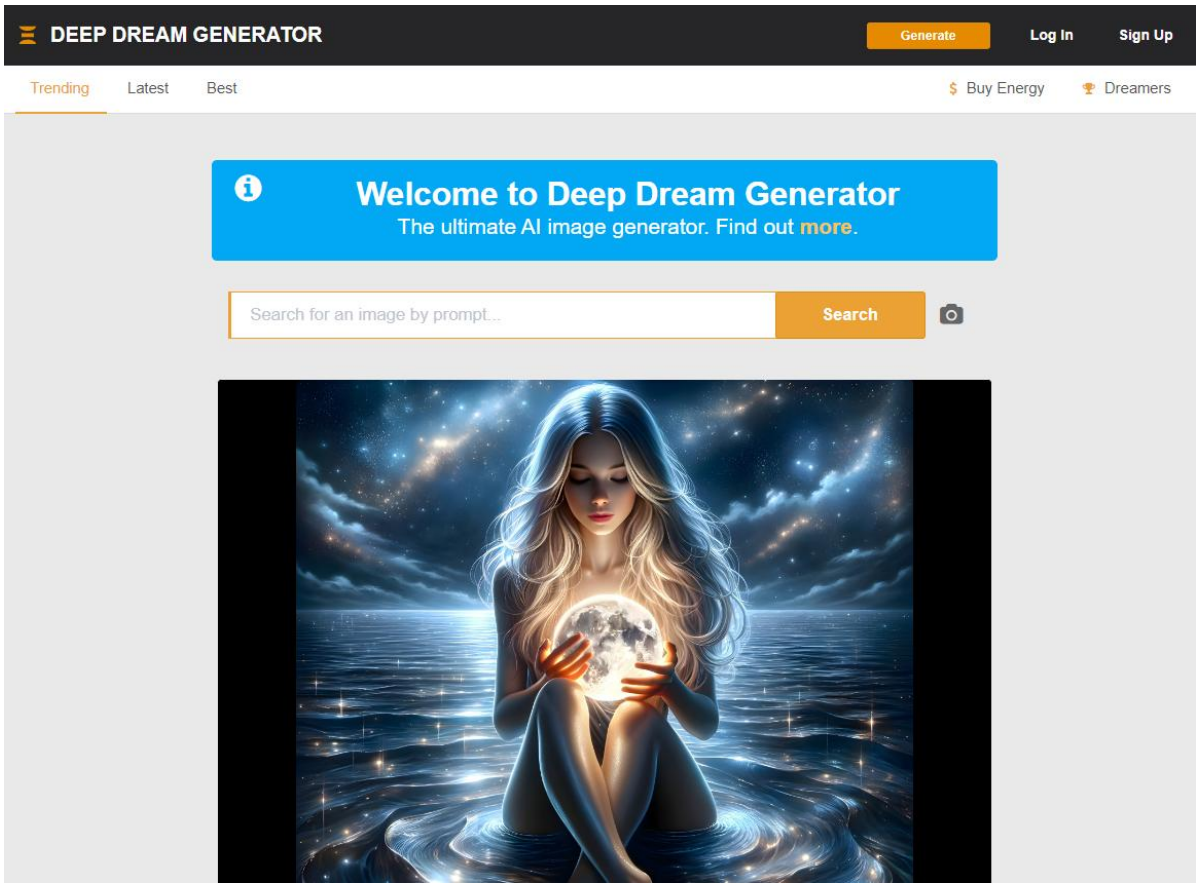



Рисунок 3.7 – Deep Dream

Креативний підхід до використання нейронних мереж, можливість творчого впливу на генерацію зображень.

- Neural Style Transfer (NST) Apps:

Декілька додатків, таких як Prisma та Artisto, використовують техніки NST для перетворення звичайних фотографій у художні шедеври, імітуючи стилі відомих художників [17]. Ці програми дозволяють користувачам взаємодіяти з алгоритмами та створювати унікальні візуальні ефекти.

README.md

# Neural Style Transfer (with app)

This repository includes a Pytorch implementation of the paper "Image Style Transfer Using Convolutional Neural Networks", Leon A. Gatys et al. that can run locally with an app ( `app.py` -developed with Streamlit) or by modifying the input arguments of a script ( `main.py` ).

## Brief description

**Neural Style Transfer** (NST) is a deep learning technique that generates an image based on the content of one image (the *content image*) and the style of another image (the *style image*).

Have a look at the next example generated with this repository. The **content** and **style** images are located at the top-left and top-right corners respectively, and the bottom row contains the **output** (stylized) image.

Рисунок 3.8 – Neural Style Transfer (NST) Apps

Застосування нейронних мереж у творчій сфері, можливість вибору стилів та експериментів із зображеннями.

# РОЗДІЛ 3. ТЕОРЕТИЧНА ЧАСТИНА

## 3.1 Вивчення методів навчання та оптимізації нейронних мереж

Навчання нейронної мережі полягає у зміні її вагових коефіцієнтів таким чином, щоб мінімізувати різницю між прогнозованим результатом та реальним (еталонним) значенням. Цей процес є основним для побудови ефективних моделей машинного навчання. Існують різні підходи до навчання, залежно від типу даних та задачі. На сьогодні відомо три парадигми навчання нейронних мереж, в основу яких покладено особливості машинного навчання [17-19]:

- кероване навчання (навчання з учителем);
- некероване навчання (навчання без учителя);
- навчання з підкріпленням.

Кероване навчання (навчання з учителем) — передбачає, що для кожного вхідного вектора (xi) існує вектор вихідних значень (di). Разом ці два вектора називають навчальною парою (xi , di), а множину навчальних пар — навчальною вибіркою. Процес навчання зводиться до почергового подавання на вхід нейронної мережі навчальних пар, вирахування похибки між дійсним і бажаним значенням нейронної δ=y-d та коригування параметрів мережі в бік зменшення цієї похибки. [20]

Некероване навчання (навчання без учителя)— один із способів машинного навчання, при вирішенні яких випробовувана система спонтанно навчається виконувати поставлене завдання, без втручання з боку експериментатора. З точки зору кібернетики, є одним з видів кібернетичного експерименту. Як правило, це підходить тільки для задач, в яких відомий опис множини об'єктів (навчальна вибірка), і необхідно виявити внутрішні взаємозв'язки, залежності, закономірності, що існують між об'єктами.

Некероване навчання часто протиставляється керованому навчанню, коли для кожного об'єкта, що навчається, примусово задається «правильна відповідь», і потрібно знайти залежність між стимулами та реакціями системи.[21]

Навчання з підкріпленням є проміжним варіантом двох попередніх парадигм. Замість «вчителя» в схему навчання вводиться блок «критика», який відслідковує реакцію середовища на вхідний сигнал і спираючись на неї визначає евристичну похибку, яку покладено в процес навчання мережі. Вказані парадигми базуються на відповідних правилах навчання, які визначають основні особливості їх застосування. [22-23]

Правило навчання покращує продуктивність штучної нейронної мережі. Таким чином, воно оновлює ваги та рівні зсуву мережі, коли в процесі навчання виконуються певні умови. це важлива частина розвитку нейронної мережі.

Типи правил навчання в ШНМ:

- Правило навчання Хебба

Одним із перших і водночас базових підходів до організації навчання в нейронних мережах є правило Хебба, сформульоване у 1949 році канадським нейропсихологом Дональдом Хеббом. Це правило описує принцип синаптичної пластичності, який ґрунтується на біологічних закономірностях функціонування головного мозку.

Суть правила полягає в тому, що синаптичний зв’язок між двома нейронами посилюється, якщо вони активуються одночасно або з незначним часовим зсувом. Іншими словами, якщо нейрон «А» регулярно бере участь в активації нейрона «В», то зв’язок між ними зміцнюється. Це явище можна узагальнено описати відомою фразою: «Нейрони, які активуються разом, об’єднуються».

Дональд Хебб розробив його як алгоритм неконтрольованого навчання в нейронній мережі. Ми можемо використовувати його для покращення ваги вузлів мережі. Коли відбувається наступне явище

- Якщо два сусідніх нейрони працюють в одній фазі в той самий період часу, то вага між цими нейронами повинна збільшитися.
- Для нейронів, що працюють в протилежній фазі, вага між ними повинна зменшуватися.
- Якщо немає кореляції сигналу, вага не змінюється, знак ваги між двома вузлами залежить від знака входу між цими вузлами.
- Коли входи обох вузлів позитивні або негативні, це призводить до сильної позитивної ваги.
- Якщо вхідні дані одного вузла позитивні, а іншого — негативні, існує сильна негативна вага.

У математичному вигляді правило Хебба можна представити як зміну вагового коефіцієнта зв’язку між нейронами:

$$\delta w = \alpha x_i y \qquad (3.1)$$

де δw = зміна ваги,

α — швидкість навчання.

□□ — вхідний вектор

y — вихід.

Це правило є локальним, оскільки для зміни ваги використовується лише інформація, доступна на рівні конкретного синапсу, без необхідності знання глобальної помилки або інших частин мережі.[24-26]

Правило Хебба лежить в основі деяких типів нейронних мереж, зокрема самоорганізуючих карт Кохонена, асоціативних пам’ятей, а також використовується в дослідженнях, пов’язаних із моделюванням когнітивних процесів. Незважаючи на свою простоту, воно стало ключовим етапом у

розвитку теорії штучних нейронних мереж та відкриває шлях до розуміння того, як нейрони можуть навчатися без учителя.

- Правило навчання перцептрона

Перцептрон — це один із найпростіших типів штучних нейронних мереж, запропонований Френком Розенблаттом у 1958 році. Він здатний виконувати лінійно відокремлювану бінарну класифікацію. Основа його функціонування полягає в лінійній комбінації вхідних сигналів з наступним застосуванням порогової функції активації. .[27-29]

Для того щоб перцептрон міг навчатися, було запропоновано правило навчання перцептрона, яке є одним із найпростіших алгоритмів коригування вагових коефіцієнтів на основі помилок класифікації.

Його ввів Розенблат. Це правило для виправлення помилок однорівневої прямої мережі. він є контрольованим за своєю природою та обчислює похибку між бажаним і фактичним виходом, і якщо вихід присутній, виконується лише коригування ваги. .[30-32]

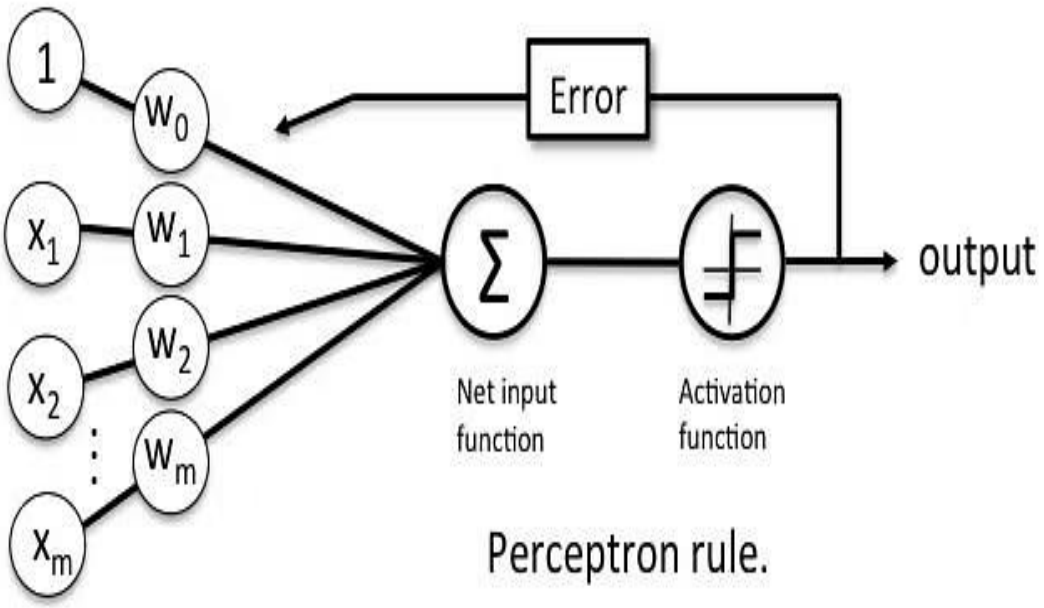



Рисунок 3.1 – Правило навчання перцептрона

- Правило дельта

Правило дельта є одним із базових алгоритмів навчання штучних нейронних мереж, зокрема одношарових перцептронів. Воно ґрунтується на методі градієнтного спуску і використовується для корекції вагових коефіцієнтів з метою мінімізації різниці між бажаним і фактичним виходом нейрона.

Суть правила полягає у тому, що зміна ваги пропорційна похибці на виході нейрона та вхідному сигналу.

Цей метод був розроблений Бернардом Уідроу та Марсіаном Хоффом і залежить від навчання під наглядом і має функцію постійної активації. Він також відомий як метод найменшого середнього квадрата, і він мінімізує помилку в усіх шаблонах навчання.

Він заснований на підході градієнтного спуску, який триває вічно. У ньому стверджується, що модифікація ваги вузла дорівнює добутку помилки на вхід, де помилка є різницею між бажаним і фактичним результатом.[33-35]

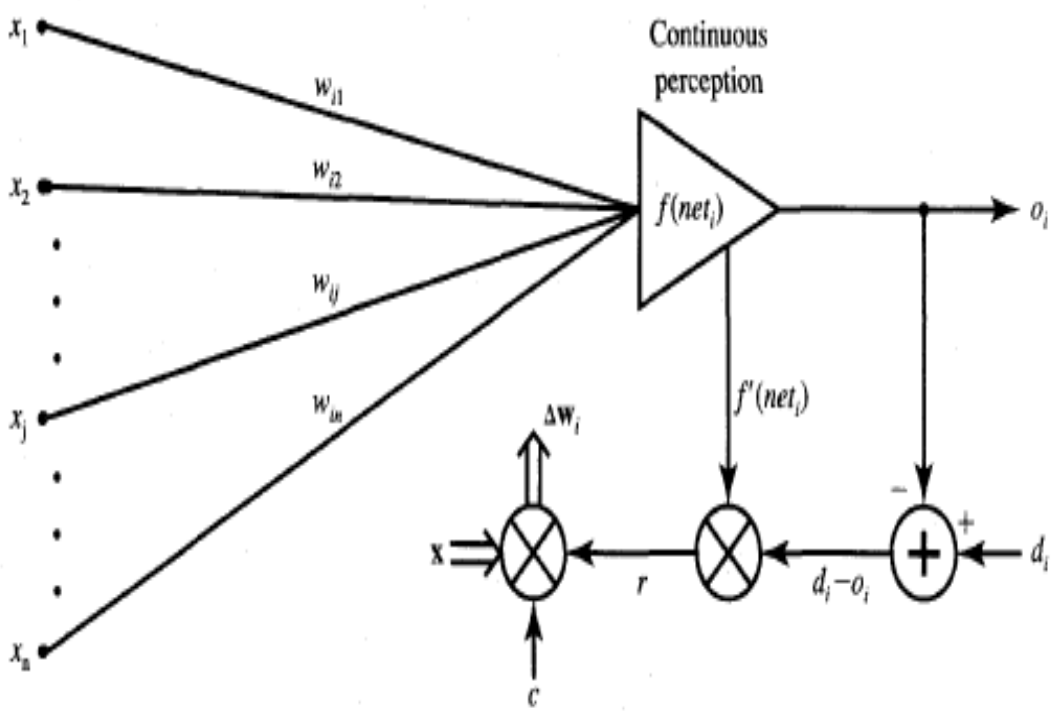



Рисунок 3.2 – Правило дельта

- Кореляційне правило:

Кореляційне правило навчання є одним із фундаментальних методів адаптації вагових коефіцієнтів у штучних нейронних мережах. Воно базується на тому, що зміна ваги між двома нейронами пропорційна кореляції їх активностей — тобто одночасній активації вхідного і вихідного нейрона.[24,29,33]

Суть цього правила полягає в тому, що вага зв'язку збільшується, якщо обидва нейрони активуються одночасно, і зменшується, якщо їх активність неузгоджена. Такий підхід моделює біологічний процес синаптичної пластичності, що лежить в основі навчання в мозку.Правило кореляційного навчання дотримується того ж принципу, що й правило навчання Хебба, тобто якщо два сусідніх нейрони працюють в одній фазі в той самий період часу, то вага між цими нейронами має бути більш позитивною. Для нейронів, що працюють у протилежній фазі, вага між ними має бути більш негативною, але на відміну від правила Хебба, тут контролюється правило кореляції, цільова відповідь використовується для розрахунку зміни ваги.

У математичній формі**:**

$$\square\square = \square\square_{\square}\square_{\square} \qquad (3.2)$$

де δw = зміна ваги,

α = швидкість навчання,

$\square_{\square}$= набір вхідного вектора

$\square_{\square}$ = цільове значення.

- Правило навчання Out Star

Правило навчання Out Star було запропоновано Френком Розенблаттом як метод навчання в штучних нейронних мережах, що базується на кореляції між вхідними сигналами та виходом нейрона. Цей алгоритм є різновидом хеббівського навчання, проте зосереджується на формуванні зв'язків від одного вхідного нейрона до багатьох вихідних.

Ідея правила Out Star полягає в тому, що зміни ваг зв’язків на виході одного нейрона пропорційні активності цього нейрона і відповідним вхідним сигналам, що подаються в мережу [24,29,33]. Формально правило можна подати так. Метод  був представлений Гроссбергом і є процедурою навчання під наглядом.

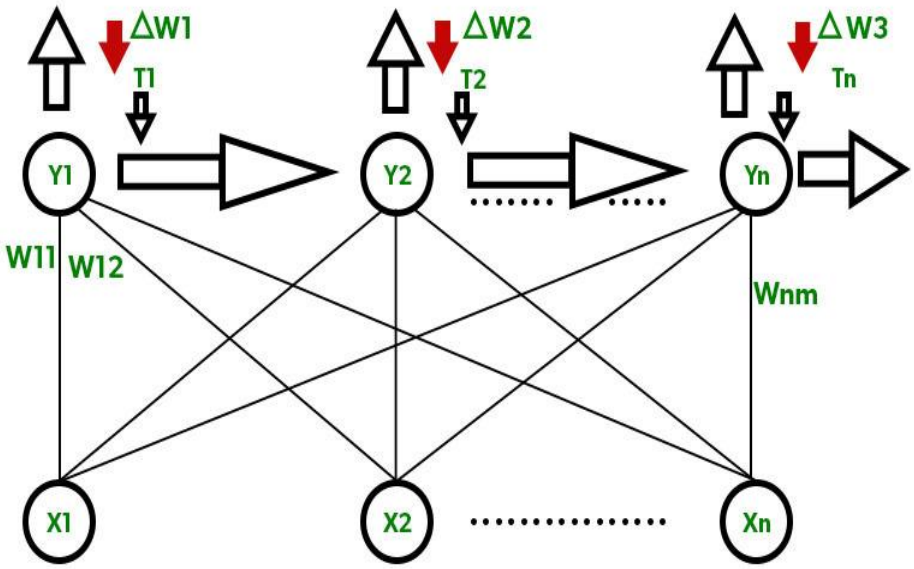



Рисунок 3.3 – Правило навчання Out Star

Правило навчання Out Star реалізується, коли вузли в мережі розташовані в шарі. Тут ваги, пов’язані з певним вузлом, повинні дорівнювати цільовим виходам для вузлів, з’єднаних через ті самі ваги.

$$\Delta w = \alpha(t - y) \qquad (3.3)$$

де α=швидкість навчання,

y=фактичний вихід

t=бажаний вихід для n вузлів шару.

- Навчання методом змагання

Навчання методом змагання — це один із підходів до навчання штучних нейронних мереж без учителя, при якому нейрони мережі конкурують за право активуватися у відповідь на вхідний сигнал. Лише один або кілька нейронів, які найбільше відповідають вхідним даним, "виграють змагання" і отримують право оновити свої ваги. [24,33]

Основний принцип полягає у формуванні самоорганізованих кластерів, де кожен нейрон спеціалізується на певному типі вхідних сигналів. Це сприяє виділенню закономірностей і структур у даних без необхідності мати мітки класів.

Воно також відоме як правило «Переможець отримує все» і за своєю природою не контролюється. Тут усі вихідні вузли намагаються конкурувати один з одним, щоб представити вхідний шаблон, і переможець оголошується відповідно до вузла, який має найбільшу кількість виходів, і отримує вихід 1, а решта – 0.

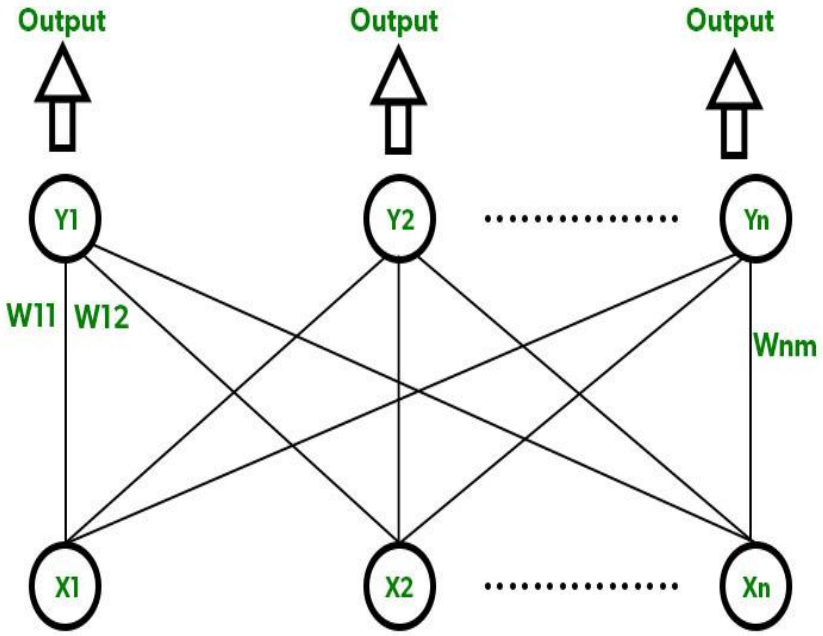



Рисунок 3.4 – Навчання методом змагання

Існує набір нейронів із довільно розподіленими вагами, і функція активації застосовується до підмножини нейронів. Одночасно активний лише один нейрон. Тільки переможець має оновлені ваги, решта залишаються без змін.

З огляду на поставлену задачу — навчити нейронну мережу розпізнавати рукописні цифри на основі набору MNIST — найбільш доцільним було використання керованого навчання, оскільки кожному зображенню у цьому наборі відповідає правильна мітка (цифра від 0 до 9). Це дозволяє напряму зіставляти передбачене значення мережі з очікуваним і коригувати її ваги для покращення результату.

Під час реалізації було обрано алгоритм зворотного поширення помилки (backpropagation) із застосуванням градієнтного спуску (gradient descent) — найпоширенішу комбінацію для навчання багатошарових нейронних мереж. Це один із варіантів дельта-правила, яке полягає в мінімізації похибки між фактичним і бажаним результатом шляхом обчислення градієнта функції помилки та коригування ваг згідно з ним.

## 3.2 Алгоритм роботи навчального програмного забезпечення

Розроблене програмне забезпечення призначене для демонстрації процесу роботи та навчання штучної нейронної мережі. Його структура та алгоритм дій побудовані так, щоб зробити процес максимально наочним і простим для розуміння студентами, які вивчають основи нейронних мереж.

**Алгоритм роботи ПЗ:**

1. **Запуск програми**
   Відбувається ініціалізація головного вікна. Перевіряється наявність файлу з попередньо збереженими вагами мережі.
2. **Завантаження або навчання мережі**

- Якщо файл з вагами існує — мережа автоматично завантажує їх для подальшого використання.
- Якщо файл відсутній — запускається процедура навчання мережі на підмножині зображень з набору MNIST. Після завершення навчання ваги зберігаються у файл для подальшого використання.

3. **Відображення головного вікна**
   Користувачу доступне поле для малювання, де він може вручну зобразити цифру.

4. **Розпізнавання цифри**

   - Зображення цифри перетворюється у формат, придатний для подачі на вхід нейронної мережі (наприклад, нормалізований масив розміром 28×28).
   - Мережа обчислює ймовірності належності введеного зображення до кожної з 10 цифр (0–9).
   - Виводиться передбачена цифра з найвищою ймовірністю.

5. **Очищення поля**
   Користувач може натиснути кнопку «Очистити», щоб стерти малюнок і результат, після чого повторити процес розпізнавання з іншою цифрою.

**Особливості реалізації:**

- Програма реалізована мовою C++ з використанням фреймворку Qt, що забезпечує зручний інтерфейс і можливість інтерактивної взаємодії з мережею.
- Всі обчислення, пов’язані з функціонуванням та навчанням нейронної мережі, реалізовані власними засобами, без сторонніх бібліотек.

- Така реалізація спрямована на освітні цілі — студенти можуть спостерігати за повним процесом навчання, аналізувати ваги, результати, та експериментувати з вхідними даними.

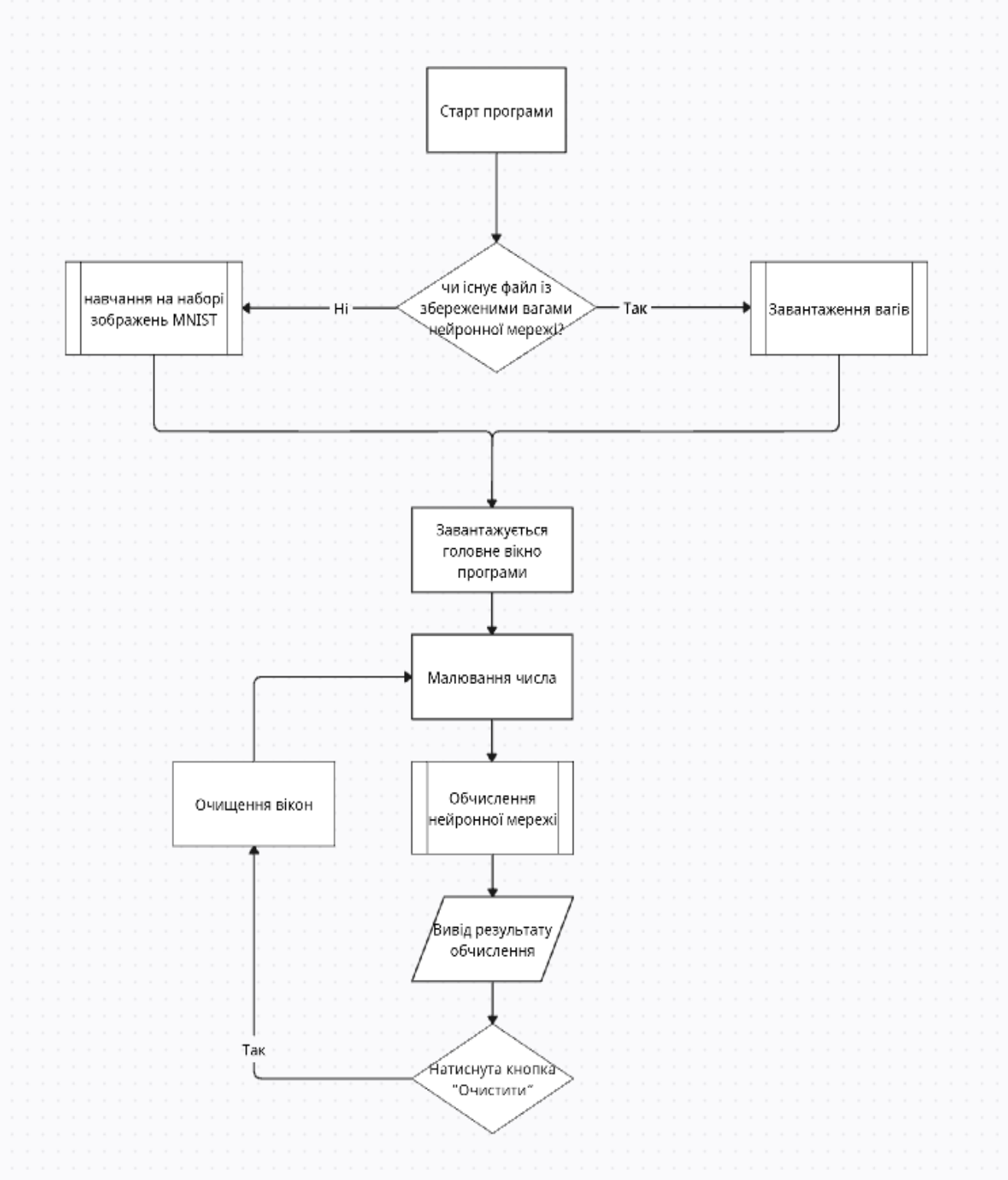



Рисунок 3.10 – Блок-схема алгоритму роботи програми

# РОЗДІЛ 4. ПРАКТИЧНА ЧАСТИНА

## 4.1 Вибір інструментарію для програмування нейронної мережі

У процесі розробки штучних нейронних мереж важливим етапом є вибір відповідного програмного інструментарію. Це включає не лише мову програмування, а й бібліотеки, середовища розробки та платформи, які забезпечують ефективне моделювання, навчання і тестування мереж.

Однією з найпоширеніших мов програмування, яку обирають для створення нейронних мереж, є C++. Вона загальновизнана завдяки своїй високій продуктивності, контролю над пам’яттю, широким можливостям оптимізації та широкому застосуванню в системах, де критичною є швидкодія (наприклад, вбудовані системи, реальний час, графічні процесори).

C++ часто використовують у зв’язці з такими бібліотеками як:

- FANN (Fast Artificial Neural Network Library) — проста у використанні бібліотека для створення багатошарових перцептронів;
- Dlib — сучасна C++ бібліотека з підтримкою машинного навчання, яка включає можливості для побудови та навчання нейронних мереж;
- Shark — бібліотека для чисельних методів, оптимізації та машинного навчання;
- TensorFlow C++ API — інтерфейс до потужної платформи TensorFlow, який дає змогу створювати нейронні мережі на C++ без використання Python.

Незважаючи на складніший синтаксис порівняно з Python, C++ дозволяє створювати більш ефективні та контрольовані реалізації алгоритмів, особливо в умовах, коли обмежені обчислювальні ресурси або потрібно глибоко оптимізувати процес.

Таким чином, використання C++ як основного інструменту програмування нейронних мереж є обґрунтованим вибором для завдань, де необхідна висока продуктивність та контроль над виконанням програми.

C++ - це загально признана об'єктно-орієнтована мова програмування, яка є розширенням мови C [36-38]. Розроблена у 1979 році Бьярном Страуструпом, C++ стала однією з найпопулярніших мов програмування завдяки своїй ефективності та гнучкості.

- C++ дозволяє створювати класи та об'єкти, що полегшує організацію та підтримку коду. Вона включає в себе такі концепції, як спадкування, поліморфізм та інкапсуляція.
- C++ вперше ввів концепцію шаблонів, що дозволяє створювати загальноприйняті функції та класи. Вони забезпечують гнучкість та високу повторюваність коду.
- C++ підтримує широкий спектр вбудованих типів даних, таких як цілі числа, дійсні числа, символи, та інші. Крім того, вона дозволяє визначати власні типи за допомогою структур та класів.
- C++ включає обширну стандартну бібліотеку, яка надає інструменти для введення/виведення, роботи з рядками, алгоритмів сортування, структур даних та багато іншого.

Qt - це крос-платформний фреймворк для розробки програмного забезпечення, який надає широкі можливості для створення графічних інтерфейсів користувача, мережевого програмування, роботи з базами даних, обробки подій і багато іншого. Ось детальний огляд ключових елементів Qt для C++:

MNIST (Modified National Institute of Standards and Technology) - це набір даних, що використовується в машинному навчанні для навчання та тестування моделей класифікації рукописних цифр. Цей набір даних став стандартним в тестуванні та порівнянні алгоритмів машинного навчання [39,40]

MNIST складається з двох частин: тренувальний набір та тестовий набір. Кожне зображення у наборі - це чорно-біле зображення розміром 28x28 пікселів, що представляє рукописну цифру від 0 до 9.

Тренувальний набір містить 60,000 зображень, а тестовий - 10,000 зображень. Це розділення дозволяє оцінювати ефективність алгоритмів на нових, раніше небачених даних.

Основна задача для MNIST - це класифікація рукописних цифр. Кожному зображенню призначена мітка (label), яка вказує на правильну цифру. Задача полягає в тому, щоб навчити модель класифікації правильно розпізнавати цифри на зображеннях. MNIST часто використовується як задача "Hello, World!" у світі машинного навчання. Він дозволяє досліджувати та порівнювати різні архітектури нейронних мереж та інші алгоритми класифікації. Важливою є його доступність та стандартизованість.

MNIST залишається важливим інструментом для вивчення та тестування методів машинного навчання, а його стандартизованість дозволяє дослідникам та розробникам порівнювати свої моделі та результати.

## 4.2 Налаштування та навчання нейронної мережі

Додаємо класи для навчання нейронної мережі (Рисунок 4.1)

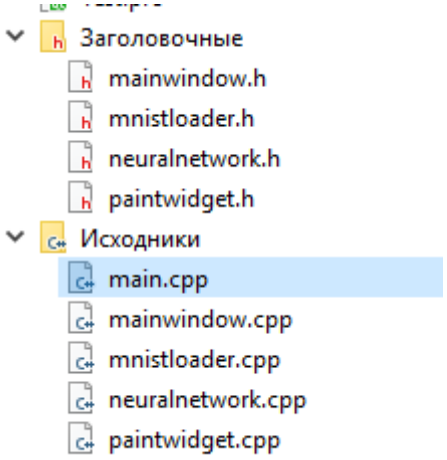



Рисунок 4.1- Класи для роботи з нейронною мережею.

- **Класс NeuralNetwork**

```
class NeuralNetwork : public QObject
{
    Q_OBJECT
public:
    NeuralNetwork(int inputSize, int hiddenSize, int outputSize);

    void train(const QVector<QVector<double>> &trainingInputs,
               const QVector<QVector<double>> &trainingTargets,
               int epochs, double learningRate);

    QVector<double> feedForward(const QVector<double> &inputs);
    QVector<double> getOutput(const QVector<double>& input);
    void saveWeights(const QString &filePath) const;
    void loadWeights(const QString &filePath);
    int getOutputSize() const;
private:
    int inputSize;
    int hiddenSize;
    int outputSize;

    QVector<QVector<double>> weightsInputHidden;
    QVector<QVector<double>> weightsHiddenOutput;

    QVector<double> hiddenBiases;
    QVector<double> outputBiases;

    QVector<double> hiddenLayerOutputs;

    double activateHiddenNeuron(int neuronIndex, const
QVector<double> &inputs);
    QVector<double> applySoftmax(const QVector<double>& logits);
};
```

**int inputSize;** - кількість нейронів у вхідному шарі

**int hiddenSize;** - кількість нейронів у прихованому шарі

**int outputSize;** - кількість нейронів у вихідному шарі

**QVector<QVector<double>> weightsInputHidden;** - матриця ваг між вхідним і прихованим шарами

**QVector<QVector<double>> weightsHiddenOutput;** - матриця ваг між прихованим і вихідним шарами

**QVector<double> hiddenBiases;** - зсуви (biases) для прихованого шару

**QVector<double> outputBiases;** - зсуви для вихідного шару

**QVector<double> hiddenLayerOutputs;** - вихідні значення нейронів прихованого шару

**NeuralNetwork(int inputSize, int hiddenSize, int outputSize);** - конструктор, що ініціалізує мережу з заданою кількістю нейронів у вхідному, прихованому та вихідному шарах. Випадково ініціалізує ваги та зсуви.

- **Реалізація класу NeuralNetwork**

```
NeuralNetwork::NeuralNetwork(int inputSize, int hiddenSize, int outputSize)
    : inputSize(inputSize), hiddenSize(hiddenSize), outputSize(outputSize) {

    weightsInputHidden.resize(inputSize);
    for (int i = 0; i < inputSize; ++i) {
        weightsInputHidden[i].resize(hiddenSize);
```

```
        for (int j = 0; j < hiddenSize; ++j) {
            weightsInputHidden[i][j] = QRandomGenerator::global()->generateDouble() * 0.02 - 0.01;
        }
    }

    weightsHiddenOutput.resize(hiddenSize);
    for (int i = 0; i < hiddenSize; ++i) {
        weightsHiddenOutput[i].resize(outputSize);
        for (int j = 0; j < outputSize; ++j) {
            weightsHiddenOutput[i][j] = QRandomGenerator::global()->generateDouble() * 0.02 - 0.01;
        }
    }

    hiddenBiases.resize(hiddenSize);
    for (int i = 0; i < hiddenSize; ++i) {
        hiddenBiases[i] = QRandomGenerator::global()->generateDouble() * 0.02 - 0.01;
    }

    outputBiases.resize(outputSize);
    for (int i = 0; i < outputSize; ++i) {
        outputBiases[i] = QRandomGenerator::global()->generateDouble() * 0.02 - 0.01;
    }
}
```

**double activateHiddenNeuron(int neuronIndex, const QVector<double> &inputs);** - обчислює активацію одного нейрона прихованого шару.

```
double NeuralNetwork::activateHiddenNeuron(int neuronIndex, const QVector<double>& inputs) {
    double sum = 0.0;
```

```
    for (int i = 0; i < inputs.size(); ++i) {
        sum += inputs[i] * weightsInputHidden[i][neuronIndex];
    }
    sum += hiddenBiases[neuronIndex];
    return std::tanh(sum);
}
```

**QVector<double> applySoftmax(const QVector<double>& logits); -** застосовує функцію **softmax** до вектору логітів для перетворення у ймовірності.

```
QVector<double> NeuralNetwork::applySoftmax(const QVector<double>& logits) {
    QVector<double> softmaxOutput(logits.size());
    double maxLogit = *std::max_element(logits.begin(), logits.end());
    double sum = 0.0;
    for (int i = 0; i < logits.size(); ++i) {
        softmaxOutput[i] = std::exp(logits[i] - maxLogit);
        sum += softmaxOutput[i];
    }
    for (int i = 0; i < logits.size(); ++i) {
        softmaxOutput[i] /= sum;
    }
    return softmaxOutput;
}
```

**int getOutputSize() const; -** повертає кількість нейронів у вихідному шарі.

```
int NeuralNetwork::getOutputSize() const {
    return outputSize;
}
```

**QVector<double> feedForward(const QVector<double> &inputs); -** обчислює результат роботи мережі для заданих вхідних даних. Повертає ймовірності для кожного класу.

```
QVector<double>NeuralNetwork::feedForward(const
QVector<double>& inputs) {
   hiddenLayerOutputs.resize(hiddenSize);
   for (int i = 0; i < hiddenSize; ++i) {
      hiddenLayerOutputs[i] = activateHiddenNeuron(i, inputs);
   }

   QVector<double> logits(outputSize);
   for (int i = 0; i < outputSize; ++i) {
      double sum = 0.0;
      for (int j = 0; j < hiddenSize; ++j) {
         sum += hiddenLayerOutputs[j] * weightsHiddenOutput[j][i];
      }
      sum += outputBiases[i];
      logits[i] = sum;
   }

   return applySoftmax(logits);
}
```

**void saveWeights(const QString &filePath) const; -** зберігає поточні ваги та зсуви вказаного шляху до файлу.

```
void NeuralNetwork::saveWeights(const QString &filePath) const {
   QFile file(filePath);
   if (!file.open(QIODevice::WriteOnly)) {
      qWarning() << "Не вдалося відкрити файл для запису:" <<
filePath;
```

```
        return;
    }

    QDataStream out(&file);


    out << inputSize << hiddenSize << outputSize;


    for (const auto &row : weightsInputHidden)
        for (double val : row)
            out << val;


    for (const auto &row : weightsHiddenOutput)
        for (double val : row)
            out << val;


    for (double val : hiddenBiases)
        out << val;

    for (double val : outputBiases)
        out << val;

    file.close();
    qDebug() << "Ваги збережено в:" << filePath;
}
```

**void loadWeights(const QString &filePath); -** Завантажує ваги та зсуви з файлу, що був збережений раніше.

```
void NeuralNetwork::loadWeights(const QString &filePath) {
```

```
    QFile file(filePath);
    if (!file.open(QIODevice::ReadOnly)) {
        qWarning() << "Не вдалося відкрити файл для читання:" << filePath;
        return;
    }

    QDataStream in(&file);
    int inSize, hidSize, outSize;
    in >> inSize >> hidSize >> outSize;

    if (inSize != inputSize || hidSize != hiddenSize || outSize != outputSize) {
        qWarning() << "Невідповідність розмірів при завантаженні ваг";
        return;
    }

    for (auto &row : weightsInputHidden)
        for (double &val : row)
            in >> val;

    for (auto &row : weightsHiddenOutput)
        for (double &val : row)
            in >> val;

    for (double &val : hiddenBiases)
        in >> val;

    for (double &val : outputBiases)
        in >> val;

    file.close();
    qDebug() << "Ваги завантажено з:" << filePath;
```

**}**

**QVector<double> getOutput(const QVector<double>& input);** - повертає вихідні значення до застосування softmax (логіти) — може бути корисним для налагодження чи подальшої обробки.

**QVector<double> NeuralNetwork::getOutput(const QVector<double>& input) {**

**return feedForward(input);**

**}**

**void train(const QVector<QVector<double>> &trainingInputs,**

**const QVector<QVector<double>> &trainingTargets,**

**int epochs, double learningRate);** - виконує навчання нейронної мережі з використанням заданого набору даних (trainingInputs, trainingTargets) протягом епох із заданим коефіцієнтом навчання.

**void NeuralNetwork::train(const QVector<QVector<double>>& trainingInputs,**

**const QVector<QVector<double>>& trainingTargets,**

**int epochs, double learningRate) {**

Навчання виконується кілька разів (кількість задається параметром **epochs**), що дозволяє мережі поступово наближатися до правильних відповідей.

**for (int epoch = 0; epoch < epochs; ++epoch) {**

**int correct = 0;**

**for (int i = 0; i < trainingInputs.size(); ++i) {**

На цьому етапі здійснюється передбачення поточного зразка з навчальної вибірки:

- Вхідний вектор inputs подається до мережі, після чого викликається метод **feedForward(inputs)**, який обчислює відповідь мережі — вектор outputs.
- Для визначення класу, який прогнозує мережа, обирається індекс максимально значення у **outputs (predicted)**.
- Аналогічно, правильна відповідь (**actual**) визначається як індекс максимального значення у векторі цільових міток targets.
- Якщо **predicted** збігається з **actual**, це вважається правильним передбаченням, і відповідний лічильник **correct** збільшується.

```
QVector<double> inputs = trainingInputs[i];
QVector<double> targets = trainingTargets[i];
QVector<double> outputs = feedForward(inputs);

int predicted = std::distance(outputs.begin(), std::max_element(outputs.begin(), outputs.end()));
int actual = std::distance(targets.begin(), std::max_element(targets.begin(), targets.end()));
if (predicted == actual) correct++;
```

Даний фрагмент коду реалізує алгоритм зворотного поширення помилки (**backpropagation**) для одношарової штучної нейронної мережі з одним прихованим шаром:

- Обчислення помилки вихідного шару:
  Вектор **outputErrors** визначає різницю між очікуваними значеннями (**targets**) і реальним виходом мережі (**outputs**) для кожного з нейронів вихідного шару.
- Оновлення ваг між прихованим і вихідним шарами:
  Ваги **weightsHiddenOutput** коригуються пропорційно до помилки кожного вихідного нейрона, виходу відповідного прихованого нейрона та коефіцієнта навчання (**learningRate**).
- Оновлення зсувів (**bias**) вихідного шару:
  Значення **outputBiases** також коригуються відповідно до помилки та швидкості навчання.
- Обчислення помилки прихованого шару:
  Помилка для кожного нейрона прихованого шару розраховується як зважена сума помилок вихідного шару, з урахуванням похідної сигмоїдної функції активації.

- Оновлення ваг між вхідним і прихованим шарами: Ваги **weightsInputHidden** змінюються відповідно до помилок прихованого шару, значень на вході та коефіцієнта навчання.
- Оновлення зсувів прихованого шару: Значення **hiddenBiases** оновлюються аналогічно.

```
QVector<double> outputErrors(outputSize);
for (int j = 0; j < outputSize; ++j) {
   outputErrors[j] = targets[j] - outputs[j];
}

for (int j = 0; j < hiddenSize; ++j) {
   for (int k = 0; k < outputSize; ++k) {
      weightsHiddenOutput[j][k] += learningRate * outputErrors[k] * hiddenLayerOutputs[j];
   }
}

for (int k = 0; k < outputSize; ++k) {
   outputBiases[k] += learningRate * outputErrors[k];
}

QVector<double> hiddenErrors(hiddenSize);
for (int j = 0; j < hiddenSize; ++j) {
   double error = 0.0;
   for (int k = 0; k < outputSize; ++k) {
      error += outputErrors[k] * weightsHiddenOutput[j][k];
   }
   hiddenErrors[j] = error * (1 - hiddenLayerOutputs[j] * hiddenLayerOutputs[j]);
}

for (int j = 0; j < inputSize; ++j) {
   for (int k = 0; k < hiddenSize; ++k) {
```

```
                weightsInputHidden[j][k] += learningRate * hiddenErrors[k] * inputs[j];
            }
        }

        for (int j = 0; j < hiddenSize; ++j) {
            hiddenBiases[j] += learningRate * hiddenErrors[j];
        }
    }

    double accuracy = (double)correct / trainingInputs.size() * 100.0;
    qDebug() << "Epoch" << epoch + 1 << "Accuracy:" << accuracy << "%";
  }
}
```

- **Класс MainWindow**

```
class MainWindow : public QMainWindow
{
  Q_OBJECT

public:
  MainWindow(NeuralNetwork *neuralNet, QWidget *parent = nullptr);
  ~MainWindow();

private:
  NeuralNetwork *neuralNetwork;

  int outputNum;
  QLabel *labelGroup1[10];
  QLabel *labelGroup2[10];
```

```
    QProgressBar *progressBars[10];

    QWidget *centralWidget;
    QLabel *resultLabel;
    QGroupBox *groupBox1;
    QGroupBox *groupBox2;
    QGroupBox *groupBox3;

    QGridLayout *mainGridLayout;
    QHBoxLayout *hboxGroup1;
    QVBoxLayout *vboxGroup1;
    QHBoxLayout *hboxGroup2;
    QVBoxLayout *vboxGroup2;
    QVBoxLayout *vboxGroup3;

    PaintWidget *paintWidget;
    QPushButton *cleanButton;

    void predictNumber();

};
```

**NeuralNetwork *neuralNetwork; -** вказівник на нейронну мережу, яка використовується для передбачення.

**int outputNum; -** поточне передбачене число.

**QLabel *labelGroup1[10]; -** мітки для виводу чисел (0–9).

**QLabel *labelGroup2[10]; -** мітки для ймовірностей.

**QProgressBar *progressBars[10]; -** прогрес-бари для графічного відображення ймовірностей.

**QWidget *centralWidget; -** центральний віджет вікна.

**QLabel *resultLabel; -** мітка для відображення остаточного результату розпізнавання.

**QGroupBox *groupBox1; -** група елементів для виводу результатів.

**QGroupBox *groupBox2;** - група елементів для прогрес-барів.

**QGroupBox *groupBox3;** - група Поля для малювання та кнопки.

**QGridLayout *mainGridLayout;** - головний сітковий макет для розміщення всіх елементів.

**QHBoxLayout *hboxGroup1;** і **QVBoxLayout *vboxGroup1;** - горизонтальні макети для відповідних груп.

**QHBoxLayout *hboxGroup2; QVBoxLayout *vboxGroup2; QVBoxLayout *vboxGroup3;** - вертикальні макети.

**PaintWidget *paintWidget;** - віджет, у якому користувач малює цифру.

**QPushButton *cleanButton;** - кнопка для очищення вікна малювання.

**MainWindow(NeuralNetwork *neuralNet, QWidget *parent = nullptr);** - конструктор, який ініціалізує головне вікно програми та встановлює вказівник на нейронну мережу. Розміщує елементи інтерфейсу.

- **Реалізація классу MainWindow**

```
MainWindow::MainWindow(NeuralNetwork *neuralNet, QWidget *parent)
    : QMainWindow(parent), neuralNetwork(neuralNet), outputNum(10)
{
    centralWidget = new QWidget();
    setCentralWidget(centralWidget);
    mainGridLayout = new QGridLayout();
    centralWidget->setLayout(mainGridLayout);
```

Цей блок коду створює та налаштовує графічні елементи для відображення результату розпізнавання цифри:

- **titleLabel —** заголовок "Результат:".
- **resultLabel —** великий текст, де показується передбачене число (спочатку "-").
- **hboxGroup1** — горизонтальний ряд чисел від 0 до 9 (підпис до прогресбарів або просто індикатори).
- **vboxGroup1** — вертикальне розміщення заголовка, результату та ряду чисел.

```
// Результат передбачення
QLabel* titleLabel = new QLabel("Результат:");
titleLabel->setAlignment(Qt::AlignCenter);
titleLabel->setFont(QFont("Arial", 18, QFont::Bold));
resultLabel = new QLabel("-");
resultLabel->setAlignment(Qt::AlignCenter);
resultLabel->setFont(QFont("Arial", 40, QFont::Bold));
resultLabel->setStyleSheet("color: darkblue;");
hboxGroup1 = new QHBoxLayout();
for (int i = 0; i < 10; i++) {
   labelGroup1[i] = new QLabel(QString::number(i));
   labelGroup1[i]->setAlignment(Qt::AlignCenter);
   labelGroup1[i]->setFont(QFont("Arial", 12));
   hboxGroup1->addWidget(labelGroup1[i]);
}
vboxGroup1 = new QVBoxLayout();
vboxGroup1->addWidget(titleLabel);
vboxGroup1->addWidget(resultLabel);
vboxGroup1->addLayout(hboxGroup1);

groupBox1 = new QGroupBox("Результат передбачення");
groupBox1->setLayout(vboxGroup1);
mainGridLayout->addWidget(groupBox1, 0, 0);
```

Цей фрагмент коду створює візуалізацію впевненості нейромережі у вигляді вертикальних прогресбарів:

- **hboxGroup2 —** горизонтальне розташування всіх елементів.
- Для кожного класу (від 0 до 9):
    - Створюється **QProgressBar** (вертикальний, прихований текст, висота 100).
    - Під ним — **QLabel** з номером класу.

    - Обидва елементи додаються до вертикального блоку **vboxGroup2.**
- **vboxGroup2** додається до hboxGroup2.
- Усе це загортається в **QGroupBox** з назвою "Впевненість моделі", який додається в сітку головного вікна.

```
// Впевненість моделі (прогрес-бари)
hboxGroup2 = new QHBoxLayout();
for (int i = 0; i < outputNum; ++i) {
   progressBars[i] = new QProgressBar();
   progressBars[i]->setRange(0, 100);
   progressBars[i]->setOrientation(Qt::Vertical);
   progressBars[i]->setTextVisible(false);
   progressBars[i]->setFixedHeight(100);

   labelGroup2[i] = new QLabel(QString::number(i));
   labelGroup2[i]->setAlignment(Qt::AlignCenter);

   vboxGroup2 = new QVBoxLayout();
   vboxGroup2->addWidget(progressBars[i]);
   vboxGroup2->addWidget(labelGroup2[i]);
   hboxGroup2->addLayout(vboxGroup2);
}

groupBox2 = new QGroupBox("Впевненість моделі");
groupBox2->setLayout(hboxGroup2);
mainGridLayout->addWidget(groupBox2, 1, 0);
```

Цей фрагмент коду відповідає за створення інтерактивного поля для малювання та кнопки очищення:

- **paintWidget —** користувацький віджет, у якому можна малювати (цифри для розпізнавання).
- **cleanButton —** кнопка "Очистити" (висота 40 пікселів).

- Обидва елементи додаються до вертикального розташування **vboxGroup3.**
- Все це оформлюється у **QGroupBox** з назвою "Поле для малювання", який розміщується в основній сітці **(mainGridLayout).**

Зв'язки сигналів:

- При натисканні на кнопку "Очистити" викликається метод **MainWindow::clearAll**, який має очищати поле, скидувати результат і прогресбари.
- При оновленні зображення у полі малювання викликається **MainWindow::predictNumber** для передбачення цифри.

```
// Поле для малювання та кнопка
paintWidget = new PaintWidget();
cleanButton = new QPushButton("Очистити");
cleanButton->setFixedHeight(40);

vboxGroup3 = new QVBoxLayout();
vboxGroup3->addWidget(paintWidget);
vboxGroup3->addWidget(cleanButton);

groupBox3 = new QGroupBox("Поле для малювання");
groupBox3->setLayout(vboxGroup3);
mainGridLayout->addWidget(groupBox3, 2, 0);

connect(cleanButton, &QPushButton::clicked, paintWidget,
&PaintWidget::clearImage);
connect(paintWidget, &PaintWidget::imageUpdated, this,
&MainWindow::predictNumber);
}

void MainWindow::predictNumber() {
QImage img = paintWidget->getImage().scaled(28, 28,
Qt::KeepAspectRatio, Qt::SmoothTransformation);
QVector<double> inputVector;
```

```
    for (int i = 0; i < 28; ++i) {
        for (int j = 0; j < 28; ++j) {
            int pixel = qGray(img.pixel(j, i));
            inputVector.append(1.0 - pixel / 255.0);
        }
    }
    QVector<double> output = neuralNetwork->getOutput(inputVector);
    if (output.size() != outputNum) {
        qDebug() << "Помилка: вихідний вектор має розмір" <<
output.size();
        return;
    }
    int predicted_digit = std::distance(output.begin(),
std::max_element(output.begin(), output.end()));
    resultLabel->setText(QString::number(predicted_digit));
    for (int i = 0; i < outputNum; ++i) {
        progressBars[i]->setValue(static_cast<int>(output[i] * 100));
    }
}

MainWindow::~MainWindow() {
}
```

**void predictNumber(); -** метод розпізнавання. Виконує передбачення числа на основі малюнка, зробленого користувачем. Отримує пікселі з PaintWidget, передає їх у нейронну мережу та відображає результат.

- **Класс PaintWidget**

```
class PaintWidget : public QWidget
{
    Q_OBJECT
public:
```

```
    explicit PaintWidget(QWidget *parent = nullptr);
    void clearImage() {
        image.fill(Qt::white);
        update();
        emit imageUpdated();
    }

    QImage getImage() const {
        return image;
    }

private:
    QImage image;

 protected:
    void paintEvent(QPaintEvent *) override {
        QPainter painter(this);
        painter.drawImage(0, 0, image);
    }

    void mouseMoveEvent(QMouseEvent *event) override {
        if (event->buttons() & Qt::LeftButton) {
            QPainter painter(&image);
            painter.setPen(QPen(Qt::black, 20, Qt::SolidLine,
Qt::RoundCap, Qt::RoundJoin));
            painter.drawPoint(event->pos());
            update();
            emit imageUpdated();
        }
    }

signals:
    void imageUpdated();
};
```

**QImage image; -** зображення, у якому зберігається результат малювання. Виступає як "полотно" для користувача.

**explicit PaintWidget(QWidget *parent = nullptr); -** конструктор який ініціалізує віджет.

- **Реалізація классу PaintWidget**

```
PaintWidget::PaintWidget(QWidget *parent)
    : QWidget{parent}
{
    setFixedSize(280, 280);
    image = QImage(280, 280, QImage::Format_Grayscale8);
    image.fill(Qt::white);

}
```

**void clearImage();** - очищає зображення, заливаючи його білим кольором, і оновлює віджет. Також випромінює сигнал **imageUpdated()** для повідомлення інших частин програми.

**QImage getImage() const;** - повертає поточне зображення, намальоване користувачем. Цей метод викликається перед передачею зображення в нейронну мережу для розпізнавання.

**void paintEvent(QPaintEvent *) override; -** відповідає за промальовку зображення на екрані. Використовується при оновленні віджета.

**void mouseMoveEvent(QMouseEvent *event) override; -** викликається під час руху миші. Якщо натиснута ліва кнопка миші, малює чорну крапку на зображенні у позиції курсора. Застосовується перо з товщиною 20 пікселів.

**void imageUpdated();** - сигнал про оновлення зображення. Використовується для автоматичного оновлення передбачення після кожного малювання.

Цей клас — ключовий елемент взаємодії користувача з програмою. Він забезпечує введення даних у вигляді зображення, яке надалі аналізується нейронною мережею.

- **Класс MNISTLoader**

  **class MNISTLoader {**

  **public:**

  **static MNISTData loadMNIST(const QString &imagePath, const QString &labelPath);**

  **};**

  **static MNISTData loadMNIST(const QString &imagePath, const QString &labelPath);** - цей метод зчитує та перетворює сирові бінарні файли MNIST у зручний формат для використання в навчанні нейронної мережі.

- **Реалізація классу MNISTLoader**

  **MNISTData MNISTLoader::loadMNIST(const QString &imagePath, const QString &labelPath) {**
  **MNISTData data;**

  **QFile imgFile(imagePath);**
  **QFile lblFile(labelPath);**

```
    if (!imgFile.open(QIODevice::ReadOnly) ||
!lblFile.open(QIODevice::ReadOnly)) {
        qDebug() << "не вдалось відкрити MNIST!";
        return data;
    }

    // Прочитати заголовок image-файлу
    imgFile.seek(4);
    quint32 numImages;
    imgFile.read(reinterpret_cast<char*>(&numImages), 4);
    numImages = qFromBigEndian(numImages);

    quint32 numRows, numCols;
    imgFile.read(reinterpret_cast<char*>(&numRows), 4);
    imgFile.read(reinterpret_cast<char*>(&numCols), 4);
    numRows = qFromBigEndian(numRows);
    numCols = qFromBigEndian(numCols);

    int imageSize = numRows * numCols;



    lblFile.seek(4);
    quint32 numLabels;
    lblFile.read(reinterpret_cast<char*>(&numLabels), 4);
    numLabels = qFromBigEndian(numLabels);

    if (numImages != numLabels) {
        qDebug() << "Кількість малюнків і міток неспівпадає";
        return data;
    }

    for (quint32 i = 0; i < numImages; ++i) {
        QByteArray imageBytes = imgFile.read(imageSize);
```

```
            if (imageBytes.size() != imageSize) break;

            QVector<double> image;
            image.reserve(imageSize);


            for (int j = 0; j < imageSize; ++j) {
                quint8 pixel = static_cast<quint8>(imageBytes[j]);
                double normPixel = static_cast<double>(pixel) / 255.0;
                // Захист від аномалій
                normPixel = std::clamp(normPixel, 0.0, 1.0);
                image.append(normPixel);
            }

            data.images.append(image);

            char labelChar;
            if (lblFile.read(&labelChar, 1) != 1) break;
            data.labels.append(static_cast<quint8>(labelChar));
        }

        qDebug() << "Images:" << data.images.size();
        if (!data.images.isEmpty() && !data.images[0].isEmpty()) {
            qDebug() << "First pixel image[0]:" << data.images[0][0];
        }

        return data;
    }
```

- **Структура MNISTData**

```
    struct MNISTData {
        QVector<QVector<double>> images;
        QVector<int> labels;
```

**};**

**QVector<QVector<double>> images; -** вектор зображень, кожне з яких подане як вектор дійсних чисел (пікселі 28×28 у вигляді одного масиву з 784 значеннями від 0 до 1).

**QVector<int> labels; -** вектор відповідних міток (цифр від 0 до 9), які є правильними класами для кожного зображення.

Комбінація MNISTLoader та MNISTData дозволяє зручно завантажити та зберігати навчальні дані, необхідні для роботи нейронної мережі. Вони ізольовані в окремий модуль, що спрощує структуру програми й забезпечує повторне використання коду.[41]

## 4.3 Реалізація графічного інтерфейсу

Графічний інтерфейс користувача (GUI) у створеній програмі реалізовано з метою забезпечення зручної взаємодії користувача з нейронною мережею. Його основним призначенням є спрощення процесу введення даних, запуску навчання мережі, перегляду результатів класифікації або прогнозування, а також візуалізації структури мережі та її параметрів.

Для створення графічного інтерфейсу було використано Qt, яка дозволяє розробляти багатофункціональні, кросплатформні віконні додатки на мові C++.

Основні елементи GUI:

- Головне вікно програми, яке містить панель інструментів, меню та поле для відображення результатів.
- Форма для введення вхідних даних, де користувач може вводити значення вручну або завантажувати з файлу.

• Кнопки керування: запуск навчання, зупинка, очищення, збереження/завантаження моделі.

• Інформаційне вікно, де відображаються повідомлення про статус навчання, точність, помилки.

• Графічний блок візуалізації — реалізовано можливість побачити активацію нейронів або зміни ваг.

Графічний інтерфейс користувача (GUI) у створеній програмі реалізовано за допомогою засобів бібліотеки Qt, що забезпечує зручну та інтерактивну взаємодію з нейронною мережею. Основне вікно програми надає користувачу можливість намалювати цифру, отримати результат передбачення та очистити поле для нового введення (Рисунок 4.2).


Рисунок 4.2 вікно програми

Інтерфейс оптимізовано для інтуїтивного використання: усі кнопки мають підписи та підказки, елементи згруповано за функціональністю. Дизайн витримано в єдиному стилі з урахуванням принципів зручності користувача (UX/UI).

# ВИСНОВКИ

У результаті виконання випускної кваліфікаційної роботи було реалізовано програмне забезпечення, яке демонструє процес навчання та роботи штучної нейронної мережі на прикладі задачі розпізнавання рукописних цифр.

У ході розробки було досягнуто наступних результатів:

- проаналізовано теоретичні основи побудови та навчання штучних нейронних мереж;
- розроблено нейронну мережу з однією прихованою шарою, реалізовану мовою програмування C++ з використанням бібліотеки Qt для графічного інтерфейсу;
- забезпечено можливість навчання мережі на даних набору MNIST;
- реалізовано збереження та завантаження ваг мережі для зручності повторного використання;
- створено інтуїтивно зрозумілий інтерфейс користувача, що дозволяє вводити рукописні цифри, отримувати результат передбачення та візуально оцінювати ймовірності класифікації;

- протестовано функціональність програми та підтверджено її коректну роботу на практичних прикладах.

Завдяки розробленій програмі користувачі можуть краще зрозуміти, як працює навчання нейронної мережі та як вона виконує класифікацію зображень. Програма також може бути використана в освітніх цілях для демонстрації принципів машинного навчання та нейронних мереж.

У перспективі можливе розширення функціональності: додавання підтримки інших типів архітектур нейронних мереж, реалізація більш гнучких налаштувань навчання, інтеграція з камерами або планшетами для реального введення рукописних даних, а також підтримка багатопотокової обробки для прискорення тренування.

## СПИСОК ЛІТЕРАТУРИ


1. Гудзь С. О. Штучний інтелект : навч. посіб. / С. О. Гудзь. — Київ : КНЕУ, 2020. — 212 с.
2. Хархота О. І. Основи машинного навчання та штучного інтелекту / О. І. Хархота. — Львів : ЛНУ імені Івана Франка, 2021. — 185 с.
3. Bishop C. M. Neural Networks for Pattern Recognition / C. M. Bishop. — Oxford : Oxford University Press, 1995. — 482 p.
4. Goodfellow I., Bengio Y., Courville A. Deep Learning / I. Goodfellow, Y. Bengio, A. Courville. — Cambridge : MIT Press, 2016. — 800 p.
5. TensorFlow. Офіційний сайт [Електронний ресурс]. — Режим доступу: https://www.tensorflow.org/ — Дата звернення: 25.05.2025.
6. PyTorch. Офіційний сайт [Електронний ресурс]. — Режим доступу: https://pytorch.org/ — Дата звернення: 25.05.2025.
7. https://www.geeksforgeeks.org/ - What is a neural network? - https://www.geeksforgeeks.org/neural-networks-a-beginners-guide/
8. https://www.geeksforgeeks.org/ - Types Of Learning Rules in ANN. - https://www.geeksforgeeks.org/types-of-learning-rules-in-ann/

9. Симоненко, В. І. Інтелектуальні інформаційні системи : навч. посіб. / В. І. Симоненко, Т. В. Глушкова. — Харків : ХНУРЕ, 2018. — 212 с.
10. Goodfellow, I. Deep learning / Ian Goodfellow, Yoshua Bengio, Aaron Courville. — Cambridge : MIT Press, 2016. — 775 p. URL: https://www.deeplearningbook.org/ (дата звернення: 25.05.2025).
11. Ситник, М. М. Основи штучного інтелекту та нейромереж / М. М. Ситник. — Львів : ЛНУ імені Івана Франка, 2021. — 240 с.
12. Qt Documentation – Graphics View Framework [Електронний ресурс]. — Режим доступу: https://doc.qt.io/qt-6/graphicsview.html (дата звернення: 25.05.2025).
13. Neural networks in C++ – FANN Library [Електронний ресурс]. — Режим доступу: http://leenissen.dk/fann/wp/ (дата звернення: 25.05.2025).
14. Yegorov, S. C++ Machine Learning [Електронний ресурс] / S. Yegorov. — Режим доступу: https://github.com/davidrmiller/neuralnet (дата звернення: 25.05.2025).
15. Lecun, Y. Gradient-Based Learning Applied to Document Recognition / Y. LeCun, L. Bottou, Y. Bengio, P. Haffner // Proceedings of the IEEE. — 1998. — Vol. 86, No. 11. — P. 2278–2324. DOI: 10.1109/5.726791
16. Ільченко, О. І. Програмна реалізація штучної нейронної мережі для класифікації зображень / О. І. Ільченко, А. В. Сидоренко // Вісник ХНУРЕ. — 2022. — № 1 (75). — С. 58–63.
17. Bishop C. M. Pattern Recognition and Machine Learning / Christopher M. Bishop. – New York : Springer, 2006. – 738 p.
18. Goodfellow I., Bengio Y., Courville A. Deep Learning / Ian Goodfellow, Yoshua Bengio, Aaron Courville. – Cambridge : MIT Press, 2016. – 775 p. – ISBN 978-0-262-03561-3.
19. Géron A. Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow. Concepts, Tools, and Techniques to Build Intelligent Systems / Aurélien Géron. – 2nd ed. – Sebastopol : O'Reilly Media, 2019. – 822 p.

20. Rumelhart D. E., Hinton G. E., Williams R. J. Learning representations by back-propagating errors // Nature. – 1986. – Vol. 323. – P. 533–536.
21. Kingma D. P., Ba J. L. Adam: A method for stochastic optimization [Electronic resource]. – 2014. – Available at: https://arxiv.org/abs/1412.6980
22. LeCun Y., Bengio Y., Hinton G. Deep learning // Nature. – 2015. – Vol. 521, No. 7553. – P. 436–444.
23. He K., Zhang X., Ren S., Sun J. Deep Residual Learning for Image Recognition [Electronic resource] // Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016. – P. 770–778. – Available at: https://arxiv.org/abs/1512.03385
24. Goodfellow I., Bengio Y., Courville A. Deep Learning / I. Goodfellow, Y. Bengio, A. Courville. – Cambridge : MIT Press, 2016. – 775 p.
25. LeCun Y., Bengio Y., Hinton G. Deep learning // Nature. – 2015. – Vol. 521, No. 7553. – P. 436–444.
26. Kingma D. P., Ba J. L. Adam: A method for stochastic optimization [Електронний ресурс] / D. P. Kingma, J. Ba. – 2014. – Режим доступу: https://arxiv.org/abs/1412.6980
27. Розенблатт Ф. Перцептрон: принципи роботи / Ф. Розенблатт // Наукові записки. – 1958. – Т. 1, № 1. – С. 65–78.
28. Гудмен І. Штучні нейронні мережі / І. Гудмен. – Київ : Наукова думка, 2020. – 312 с.
29. Іванов С. П. Нейронні мережі і глибоке навчання : навч. посіб. / С. П. Іванов, Л. В. Коваль. – Харків : ХНУРЕ, 2021. – 218 с.
30. Козак А. І., Прудкий В. І. Штучний інтелект: теорія і практика / А. І. Козак, В. І. Прудкий. – Львів : ЛНУ імені Івана Франка, 2022. – 304 с.
31. Goodfellow I., Bengio Y., Courville A. Deep Learning / I. Goodfellow, Y. Bengio, A. Courville. – Cambridge : MIT Press, 2016. – 775 p.
32. LeCun Y., Bengio Y., Hinton G. Deep learning // Nature. – 2015. – Vol. 521, No. 7553. – P. 436–444.

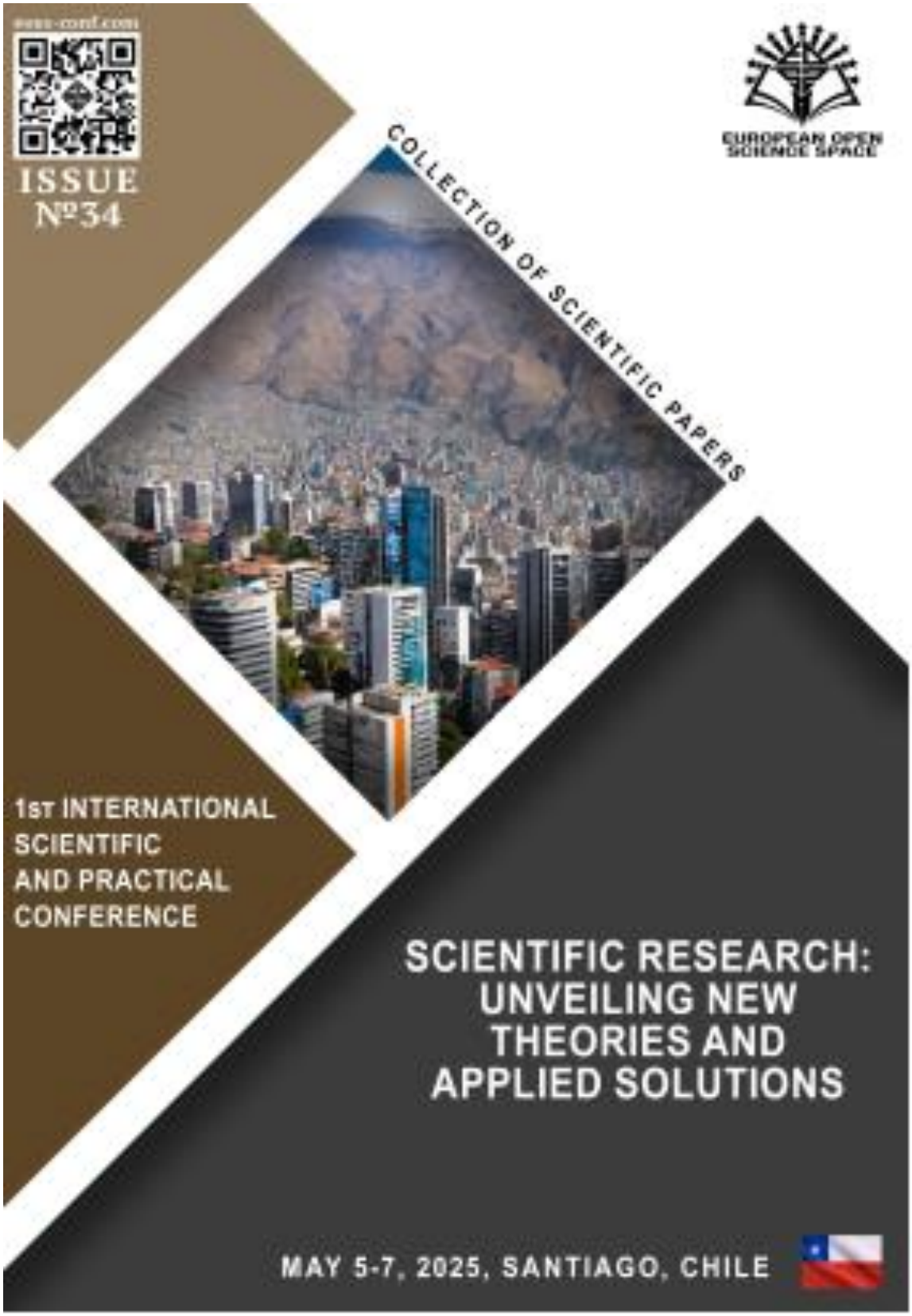
33. Гудмен І. Штучні нейронні мережі / І. Гудмен. – Київ : Наукова думка, 2020. – 312 с.
34. Rumelhart D. E., Hinton G. E., Williams R. J. Learning representations by back-propagating errors // Nature. – 1986. – Vol. 323. – P. 533–536.
35. Goodfellow I., Bengio Y., Courville A. Deep Learning / I. Goodfellow, Y. Bengio, A. Courville. – Cambridge : MIT Press, 2016. – 775 p.
36. Stroustrup B. The C++ Programming Language / Bjarne Stroustrup. – 4th ed. – Boston : Addison-Wesley, 2013. – 1376 p.
37. FANN – Fast Artificial Neural Network Library [Електронний ресурс]. – Режим доступу: http://leenissen.dk/fann/wp/
38. Dlib C++ Library [Електронний ресурс]. – Режим доступу: http://dlib.net/
39. Shark Machine Learning Library [Електронний ресурс]. – Режим доступу: http://image.diku.dk/shark/
40. TensorFlow C++ API Guide [Електронний ресурс]. – Режим доступу: https://www.tensorflow.org/guide/cc
41. Наконечний К. Олексійчук Ю. Розробка елементів програми для демонстрації навчання штучної нейронної мережі на прикладі задачі розпізнавання цифр. Scientific Research: Unveiling New Theories and Applied Solutions: збірник наукових праць за матеріалами 1-ї Міжнародної науково-практичної конференції (5–7 травня 2025 р., Сантьяго, Чилі). – European Open Science Space, 2025. с. 88–91
42. Гусак Ю. С., Олексійчук Ю. Ф. Програмна реалізація елементів тренажеру з теми «Моделювання булевих функцій за допомогою елементарного персептрону» дисципліни «Нейронно-мережеві технології в інформатиці» // Науково-практичний семінар "Комп’ютерні науки і прикладна математика" (КНіПМ-2020), Полтава, ПУЕТ.
43. Кильник В. В., Олексійчук Ю. Ф. Програмна реалізація елементів тренажеру з теми «Навчання елементарного персептрону» дисципліни «Нейронно-мережеві технології в інформатиці» // КОМП’ЮТЕРНІ

НАУКИ І ПРИКЛАДНА МАТЕМАТИКА (КНіПМ-2018): матеріали науково-практичного семінару. Випуск 1 – Полтава: Кафедра ММСІ ПУЕТ, 2018. – С. 54-58.

44. Вовк О. В., Олексійчук Ю. Ф. Використання штучного інтелекту для виявлення шкідливих веб-адрес // Актуальні питання розвитку науки та забезпечення якості освіти у XXI столітті : тези доповідей XLVII Міжнародної наукової студентської конференції за підсумками науково-дослідних робіт студентів за 2023 рік. - С. 423-425.

# ДОДАТКИ

# ДОДАТОК А


ISSUE
№34
EUROPEAN OPEN SCIENCE SPACE
COLLECTION OF SCIENTIFIC PAPERS
1st INTERNATIONAL SCIENTIFIC AND PRACTICAL CONFERENCE
SCIENTIFIC RESEARCH: UNVEILING NEW THEORIES AND APPLIED SOLUTIONS
MAY 5-7, 2025, SANTIAGO, CHILE

EUROPEAN OPEN SCIENCE SPACE

Proceedings of the 1st International Scientific and Practical Conference

# "Scientific Research: Unveiling New Theories and Applied Solutions"

May 5-7, 2025

Santiago, Chile

## Collection of Scientific Papers

Chile, 2025

UDC 01.1

Collection of Scientific Papers with the Proceedings of the 1st International Scientific and Practical Conference «Scientific Research: Unveiling New Theories and Applied Solutions» (May 5-7, 2025. Santiago, Chile). European Open Science Space, 2025. 244 p.

ISBN 979-8-89704-965-3 (series)
DOI 10.70286/EOSS-05.05.2025

The conference is included in the Academic Research Index ResearchBib International catalog of scientific conferences.

The conference is registered in the database of scientific and technical events of UkrISTEI to be held on the territory of Ukraine (Certificate №47 dated 6.01.2025).



The materials of the collection are presented in the author's edition and printed in the original language. The authors of the published materials bear full responsibility for the authenticity of the given facts, proper names, geographical names, quotations, economic and statistical data, industry terminology, and other information.

ISBN 979-8-89704-965-3 (series)

EUROPEAN OPEN SCIENCE SPACE

Proceedings of the 1st International Scientific and Practical Conference
"Scientific Research: Unveiling New Theories and Applied Solutions"
May 5-7, 2025
Santiago, Chile

**References**

1. American Cancer Society. Cancer Facts & Figures 2023. Atlanta: American Cancer Society, 2023.
2. R. Corona et al., "Interobserver variability on the histopathologic diagnosis of cutaneous melanoma and other pigmented skin lesions," J. Clin. Oncol., vol. 14, no. 4, pp. 1218-1223, Apr. 1996.
3. A. Esteva et al., "Dermatologist-level classification of skin cancer with deep neural networks," Nature, vol. 542, no. 7639, pp. 115-118, Feb. 2017.
4. The Cancer Genome Atlas Network, "Genomic classification of cutaneous melanoma," Cell, vol. 161, no. 7, pp. 1681-1696, Jun. 2015.
5. M. Salvi et al., "Stain normalization techniques for histopathology image analysis: A comprehensive review," IEEE Access, vol. 9, pp. 123456-123470, Sep. 2021.

# РОЗРОБКА ЕЛЕМЕНТІВ ПРОГРАМИ ДЛЯ ДЕМОНСТРАЦІЇ НАВЧАННЯ ШТУЧНОЇ НЕЙРОННОЇ МЕРЕЖІ НА ПРИКЛАДІ ЗАДАЧІ РОЗПІЗНАВАННЯ ЦИФР

Науковий керівник:
**Олексійчук Юрій**
к ф.-м. наук
**Наконечний Костянтин**
здобувач вищої освіти, КН б-41
Полтавський університет економіки і торгівлі, Україна

Сучасний розвиток технологій, зокрема у галузі штучного інтелекту та машинного навчання, привертає увагу дослідників і фахівців з усього світу. Однією з ключових областей застосування є розробка та вдосконалення штучних нейронних мереж для розв'язання завдань розпізнавання образів. Задача розпізнавання цифр не тільки є важливою для комп'ютерного зору та обробки зображень, але й має величезний практичний потенціал у сферах від сучасних технологій банківської справи до автоматизованої медицинської діагностики [1].

Зростання кількості доступних даних та збільшення обчислювальних можливостей відкривають нові можливості для реалізації та впровадження нейронних мереж в різноманітних галузях життя. Поглиблене вивчення та розуміння принципів функціонування таких мереж, а також їхнє навчання та використання, стає важливим завданням для молодих дослідників та розробників [2,3].

Proceedings of the 1st International Scientific and Practical Conference
"Scientific Research: Unveiling New Theories and Applied Solutions"
May 5-7, 2025
Santiago, Chile

Реалізована штучна нейронна мережа має наступну архітектуру:

Вхідний шар: кількість нейронів відповідає кількості пікселів у зображенні MNIST ($28 \times 28 = 784$).

Прихований шар: Один прихований шар з 100 нейронів. Вихідний шар: 10 нейронів, які відповідають класам цифр від 0 до 9.

Активуюча функція: у прихованому шарі використовується Leaky ReLU:

$$f(x) = \begin{cases} x, & x > 0 \\ 0.01x, & x < 0 \end{cases} \quad (1)$$

У вихідному шарі застосовується Softmax, яка перетворює вихідні значення у ймовірності належності до кожного з 10 класів.

Функція втрат: використовується кросентропія (cross-entropy):

$$L = -\sum_{i=1}^{C} y_i \times \log(\hat{y}_i) \quad (2)$$

де $y_i$ — істинне значення (one-hot), а $\hat{y}_i$ — ймовірність, передбачена softmax для класу i.

Реалізація передбачає об'єднання softmax + кросентропії через похідні.

Навчання нейронної мережі здійснюється за допомогою методу зворотного поширення помилки (backpropagation) та градієнтного спуску (gradient descent). Загальний алгоритм виглядає наступним чином:

1. Ініціалізація ваг та зміщень: Ваги між шарами та зміщення ініціалізуються випадковими значеннями за схемою Xavier. Генерація значень здійснюється з рівномірного розподілу у межах $[-x, x]$, де $x = \sqrt{2/(n_{\text{вхід}} + n_{\text{вихід}})}$.

2. Навчання по епохах: для кожної епохи проводиться ітерація по всіх зображеннях навчальної вибірки (можливе випадкове перемішування).

3. Пряме поширення (feedforward): обчислення активацій прихованого шару за допомогою Leaky ReLU. Обчислення виходу мережі та застосування Softmax до вихідного шару для отримання ймовірностей класів.

4. Обчислення помилки (loss): похибка вихідного шару обчислюється як різниця між передбаченими ймовірностями (після Softmax) та one-hot міткою правильного класу.

5. Зворотне поширення помилки: похибки прихованого шару обчислюються з урахуванням похідної Leaky ReLU. Розрахунок градієнтів для всіх ваг та зміщень.

6. Оновлення ваг: ваги та зміщення оновлюються з використанням градієнтного спуску:

$$w = w - \eta \cdot \Delta w \quad (3)$$

де $\eta$ — швидкість навчання (*learning rate*), а $\Delta w$— похідна помилки по відповідній вазі. Значення змін обмежуються функцією clamp в межах [−1,1] для уникнення великих коливань.

7. Оцінка точності: після кожної епохи розраховується точність класифікації на навчальній вибірці. Додатково виводяться середні абсолютні значення ваг і зміщень для моніторингу їх змін та стабільності навчання.

Proceedings of the 1st International Scientific and Practical Conference
"Scientific Research: Unveiling New Theories and Applied Solutions"
May 5-7, 2025
Santiago, Chile

Штучна нейронна мережа реалізована мовою C++ з використанням фреймворку Qt. Такий підхід дозволив поєднати ефективність низькорівневого коду з можливостями зручної роботи з GUI та обробкою даних.

Основні особливості реалізації:

- Без використання сторонніх бібліотек машинного навчання (як-от TensorFlow або PyTorch) — вся логіка реалізована вручну.
- Клас NeuralNetwork інкапсулює основну функціональність: ініціалізацію, пряме поширення, навчання та оновлення параметрів.
- Завантаження та обробка MNIST-даних реалізоване у власному класі MNISTData, що читає зображення та мітки у форматі IDX.
- Генерація випадкових чисел для ініціалізації ваг виконується через кастомний генератор RandomGenerator, заснований на QRandomGenerator.

Користувач може:

- Малювати цифру у спеціальному полі — інтерактивному віджеті (PaintWidget), що підтримує вільне малювання за допомогою миші.
- Очистити поле малювання натисканням кнопки "Очистити", що дозволяє повторно вводити нову цифру.
- Миттєво отримати результат розпізнавання — після кожного введення зображення мережа автоматично обробляє його та виводить передбачену цифру.
- Переглядати ймовірності для кожної цифри (0–9) у вигляді вертикальних прогрес-барів, що відображають рівень "впевненості" мережі в кожному варіанті.
- Аналізувати розподіл вихідного вектору нейромережі, що візуалізовано для зручності сприйняття та контролю якості передбачення.
- Повністю написано вручну, без сторонніх ML-бібліотек.
- Створено за допомогою C++ та Qt, що робить додаток кросплатформним.
- Інтерфейс простий і інтуїтивно зрозумілий — з ним може впоратися навіть користувач без спеціальної підготовки.

У роботі розроблено навчальний інструмент на C++/Qt, який реалізує процес навчання простої ШНМ для задачі розпізнавання цифр. Серед ключових переваг: відсутність сторонніх бібліотек, повна контрольованість процесу, візуалізація всіх етапів роботи мережі. Програма демонструє можливість ефективної реалізації ШНМ базовими засобами C++, що дозволяє використовувати її для викладання, досліджень або самостійного вивчення нейронних мереж.

**Список використаних джерел**

1. Сімон, К. А., & Ніколюк, П. К. (2022). Нейронні мережі. Комп'ютерні технології обробки даних. 170-173.

Proceedings of the 1st International Scientific and Practical Conference
"Scientific Research: Unveiling New Theories and Applied Solutions"
May 5-7, 2025
Santiago, Chile
EUROPEAN OPEN SCIENCE SPACE

2. Samek, W., Montavon, G., Lapuschkin, S., Anders, C. J., & Müller, K. R. (2021). Explaining deep neural networks and beyond: A review of methods and applications. Proceedings of the IEEE, 109(3), 247-278..
3. Mijwel, M. M. (2021). Artificial neural networks advantages and disadvantages. Mesopotamian Journal of Big Data, 2021, 29-31.

# MODELING OF A DIGITAL LABORATORY SYSTEM WITH A HIGH-PERFORMANCE COMPUTING SERVER

**Murava O.I.**
software engineering studentess
**Kornuta V.A.**
PhD, Associate Professor
Ivano-Frankivsk National Technical University of Oil and Gas

In today's rapidly developing information technologies, the introduction of digital solutions in the field of education and scientific research is becoming particularly relevant. One of the promising areas is the creation of digital laboratories.

A digital laboratory is an organizational system that provides students and teachers with the opportunity to use digital tools and technologies in the educational process. It is designed to facilitate access to knowledge, promote the active involvement of students in the study of disciplines and the development of practical skills [1].

In parallel, there is an increasing importance of high-performance computing (High-Performance Computing) that allows you to effectively process large data sets and perform resource-intensive calculations in a wide variety of fields of science and technology. High-performance computing (HPC): powerful computational systems, from simple (for example, 1 CPU + 8 GPU) to world level supercomputers [2].

Combining the concept of a digital laboratory with the capabilities of HPC servers opens up new opportunities for modernizing the educational process and optimizing research activities.

The purpose given work is development conceptual models digital laboratory integrated with the server highly productive calculations that provides comfortable access to computational resources for carrying out complex simulations and analysis.

Digital laboratories are being implemented by with help specialized software cloud security services and virtualization platforms. The most widespread of the tools are:

- MATLAB , LabVIEW , Simulink — for engineering modeling;
- Python , Jupyter Notebooks , R — for analysis data and developments algorithms;

## ДОДАТОК Б


## ДОДАТОК В

- **Файл mainwindow.h**

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include "neuralnetwork.h"
#include "paintwidget.h"
#include <QMainWindow>
#include <QLabel>
#include <QGridLayout>
#include <QHBoxLayout>
#include <QVBoxLayout>
#include <QGroupBox>
#include <QProgressBar>
```

```
#include <QPushButton>
class MainWindow : public QMainWindow
{
    Q_OBJECT

public:
    MainWindow(NeuralNetwork *neuralNet, QWidget *parent = nullptr);
    ~MainWindow();

private:
    NeuralNetwork *neuralNetwork;

    int outputNum;
    QLabel *labelGroup1[10];
    QLabel *labelGroup2[10];
    QProgressBar *progressBars[10];

    QWidget *centralWidget;
    QLabel *resultLabel;
    QGroupBox *groupBox1;
    QGroupBox *groupBox2;
    QGroupBox *groupBox3;

    QGridLayout *mainGridLayout;
    QHBoxLayout *hboxGroup1;
    QVBoxLayout *vboxGroup1;
    QHBoxLayout *hboxGroup2;
    QVBoxLayout *vboxGroup2;
    QVBoxLayout *vboxGroup3;

    PaintWidget *paintWidget;
    QPushButton *cleanButton;
```

```
    void predictNumber();

private slots:
    void clearAll();

};
#endif // MAINWINDOW_H
```

- **Файл mnistloader.h**

```
#ifndef MNISTLOADER_H
#define MNISTLOADER_H

#include <QVector>
#include <QFile>
#include <QDataStream>
#include <QDebug>
#include <QtEndian>

struct MNISTData {
    QVector<QVector<double>> images;
    QVector<int> labels;
};

class MNISTLoader {
public:
    static MNISTData loadMNIST(const QString &imagePath, const QString &labelPath);
};

#endif // MNISTLOADER_H
```

- **Файл neuralnetwork.h**

```
#ifndef NEURALNETWORK_H
```

```
#define NEURALNETWORK_H

#include <QObject>
#include <QVector>
#include <QRandomGenerator>
#include <QDebug>

class NeuralNetwork : public QObject
{
   Q_OBJECT
public:
   NeuralNetwork(int inputSize, int hiddenSize, int outputSize);

   void train(const QVector<QVector<double>> &trainingInputs,
              const QVector<QVector<double>> &trainingTargets,
              int epochs, double learningRate);

   QVector<double> feedForward(const QVector<double> &inputs);
   QVector<double> getOutput(const QVector<double>& input);
   void saveWeights(const QString &filePath) const;
   void loadWeights(const QString &filePath);
   int getOutputSize() const;
private:
   int inputSize;
   int hiddenSize;
   int outputSize;

   QVector<QVector<double>> weightsInputHidden;
   QVector<QVector<double>> weightsHiddenOutput;

   QVector<double> hiddenBiases;
   QVector<double> outputBiases;

   QVector<double> hiddenLayerOutputs;
```

```
    double activateHiddenNeuron(int neuronIndex, const QVector<double> &inputs);
    QVector<double> applySoftmax(const QVector<double>& logits);



};

#endif // NEURALNETWORK_H
```

- **Файл paintwidget.h**

```
#ifndef PAINTWIDGET_H
#define PAINTWIDGET_H

#include <QWidget>
#include <QPainter>
#include <QMouseEvent>
#include <QPushButton>
#include <QDebug>
#include <QImage>
class PaintWidget : public QWidget
{
    Q_OBJECT
public:
    explicit PaintWidget(QWidget *parent = nullptr);
    void clearImage() {
        image.fill(Qt::white);
        update();
        emit imageUpdated();
    }

    QImage getImage() const {
        return image;
    }
```

```
private:
    QImage image;

 protected:
    void paintEvent(QPaintEvent *) override {
        QPainter painter(this);
        painter.drawImage(0, 0, image);
    }

    void mouseMoveEvent(QMouseEvent *event) override {
        if (event->buttons() & Qt::LeftButton) {
            QPainter painter(&image);
            painter.setPen(QPen(Qt::black, 20, Qt::SolidLine,
Qt::RoundCap, Qt::RoundJoin));
            painter.drawPoint(event->pos());
            update();
            emit imageUpdated();
        }
    }

signals:
    void imageUpdated();
};

#endif // PAINTWIDGET_H
```

- **Файл main.cpp**

```
#include "mainwindow.h"
#include "mnistloader.h"
#include <QApplication>

int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
```

```
NeuralNetwork *network = new NeuralNetwork(28 * 28, 100, 10);

MNISTData mnistData = MNISTLoader::loadMNIST(
    "MNIST/train-images.idx3-ubyte",
    "MNIST/train-labels.idx1-ubyte");

qDebug() << "Loaded images:" << mnistData.images.size();
qDebug() << "First label:" << mnistData.labels[0];

QVector<QVector<double>> oneHotTargets;
for (int label : mnistData.labels) {
    QVector<double> target(10, 0.0);
    target[label] = 1.0;
    oneHotTargets.append(target);
}

QString weightsPath = "saved_weights.dat";


if (QFile::exists(weightsPath)) {
    network->loadWeights(weightsPath);
} else {
    network->train(mnistData.images, oneHotTargets, 5, 0.01);
    network->saveWeights(weightsPath);
}


qDebug() << "Output after training for first image:";
qDebug() << network->feedForward(mnistData.images[0]);

MainWindow w(network);
w.show();

return a.exec();
```

```
}
```

- **Файл mainwindow.cpp**

```
#include "mainwindow.h"
#include <QVBoxLayout>
#include <QHBoxLayout>
#include <QImage>
#include <QDebug>
#include <QtMath>
#include <algorithm>

MainWindow::MainWindow(NeuralNetwork *neuralNet, QWidget *parent)
    : QMainWindow(parent), neuralNetwork(neuralNet), outputNum(10)
{
    centralWidget = new QWidget();
    setCentralWidget(centralWidget);
    mainGridLayout = new QGridLayout();
    centralWidget->setLayout(mainGridLayout);

    // Результат передбачення
    QLabel* titleLabel = new QLabel("Результат:");
    titleLabel->setAlignment(Qt::AlignCenter);
    titleLabel->setFont(QFont("Arial", 18, QFont::Bold));

    resultLabel = new QLabel("-");
    resultLabel->setAlignment(Qt::AlignCenter);
    resultLabel->setFont(QFont("Arial", 40, QFont::Bold));
    resultLabel->setStyleSheet("color: darkblue;");

    hboxGroup1 = new QHBoxLayout();
    for (int i = 0; i < 10; i++) {
        labelGroup1[i] = new QLabel(QString::number(i));
        labelGroup1[i]->setAlignment(Qt::AlignCenter);
```

```
    labelGroup1[i]->setFont(QFont("Arial", 12));
    hboxGroup1->addWidget(labelGroup1[i]);
}

vboxGroup1 = new QVBoxLayout();
vboxGroup1->addWidget(titleLabel);
vboxGroup1->addWidget(resultLabel);
vboxGroup1->addLayout(hboxGroup1);

groupBox1 = new QGroupBox("Результат передбачення");
groupBox1->setLayout(vboxGroup1);
mainGridLayout->addWidget(groupBox1, 0, 0);

// Впевненість моделі (прогрес-бари)
hboxGroup2 = new QHBoxLayout();
for (int i = 0; i < outputNum; ++i) {
    progressBars[i] = new QProgressBar();
    progressBars[i]->setRange(0, 100);
    progressBars[i]->setOrientation(Qt::Vertical);
    progressBars[i]->setTextVisible(false);
    progressBars[i]->setFixedHeight(100);

    labelGroup2[i] = new QLabel(QString::number(i));
    labelGroup2[i]->setAlignment(Qt::AlignCenter);

    vboxGroup2 = new QVBoxLayout();
    vboxGroup2->addWidget(progressBars[i]);
    vboxGroup2->addWidget(labelGroup2[i]);
    hboxGroup2->addLayout(vboxGroup2);
}

groupBox2 = new QGroupBox("Впевненість моделі");
groupBox2->setLayout(hboxGroup2);
mainGridLayout->addWidget(groupBox2, 1, 0);
```

```
    // Поле для малювання та кнопка
    paintWidget = new PaintWidget();
    cleanButton = new QPushButton("Очистити");
    cleanButton->setFixedHeight(40);

    vboxGroup3 = new QVBoxLayout();
    vboxGroup3->addWidget(paintWidget);
    vboxGroup3->addWidget(cleanButton);

    groupBox3 = new QGroupBox("Поле для малювання");
    groupBox3->setLayout(vboxGroup3);
    mainGridLayout->addWidget(groupBox3, 2, 0);

    connect(cleanButton, &QPushButton::clicked, this,
&MainWindow::clearAll);
    connect(paintWidget, &PaintWidget::imageUpdated, this,
&MainWindow::predictNumber);
}

void MainWindow::clearAll() {
    paintWidget->clearImage();
    resultLabel->setText("-");
    for (int i = 0; i < outputNum; ++i) {
        progressBars[i]->setValue(0);
    }
}

void MainWindow::predictNumber() {
    QImage img = paintWidget->getImage().scaled(28, 28,
Qt::KeepAspectRatio, Qt::SmoothTransformation);
    QVector<double> inputVector;
    for (int i = 0; i < 28; ++i) {
        for (int j = 0; j < 28; ++j) {
```

```
            int pixel = qGray(img.pixel(j, i));
            inputVector.append(1.0 - pixel / 255.0);
        }
    }
    QVector<double> output = neuralNetwork->getOutput(inputVector);
    if (output.size() != outputNum) {
        qDebug() << "Помилка: вихідний вектор має розмір" << output.size();
        return;
    }
    int predicted_digit = std::distance(output.begin(), std::max_element(output.begin(), output.end()));
    resultLabel->setText(QString::number(predicted_digit));
    for (int i = 0; i < outputNum; ++i) {
        progressBars[i]->setValue(static_cast<int>(output[i] * 100));
    }
}

MainWindow::~MainWindow() {
}
```

- **Файл mnistloader.cpp**

```
#include "mnistloader.h"

MNISTData MNISTLoader::loadMNIST(const QString &imagePath, const QString &labelPath) {
    MNISTData data;

    QFile imgFile(imagePath);
    QFile lblFile(labelPath);

    if (!imgFile.open(QIODevice::ReadOnly) || !lblFile.open(QIODevice::ReadOnly)) {
        qDebug() << "не вдалось відкрити MNIST!";
        return data;
```

```
}

// Прочитати заголовок image-файлу
imgFile.seek(4);
quint32 numImages;
imgFile.read(reinterpret_cast<char*>(&numImages), 4);
numImages = qFromBigEndian(numImages);

quint32 numRows, numCols;
imgFile.read(reinterpret_cast<char*>(&numRows), 4);
imgFile.read(reinterpret_cast<char*>(&numCols), 4);
numRows = qFromBigEndian(numRows);
numCols = qFromBigEndian(numCols);

int imageSize = numRows * numCols;



lblFile.seek(4);
quint32 numLabels;
lblFile.read(reinterpret_cast<char*>(&numLabels), 4);
numLabels = qFromBigEndian(numLabels);

if (numImages != numLabels) {
   qDebug() << "Кількість малюнків і міток неспівпадає";
   return data;
}

for (quint32 i = 0; i < numImages; ++i) {
   QByteArray imageBytes = imgFile.read(imageSize);
   if (imageBytes.size() != imageSize) break;

   QVector<double> image;
   image.reserve(imageSize);
```

```
        for (int j = 0; j < imageSize; ++j) {
            quint8 pixel = static_cast<quint8>(imageBytes[j]);
            double normPixel = static_cast<double>(pixel) / 255.0;
            // Захист від аномалій
            normPixel = std::clamp(normPixel, 0.0, 1.0);
            image.append(normPixel);
        }

        data.images.append(image);

        char labelChar;
        if (lblFile.read(&labelChar, 1) != 1) break;
        data.labels.append(static_cast<quint8>(labelChar));
    }

    qDebug() << "Images:" << data.images.size();
    if (!data.images.isEmpty() && !data.images[0].isEmpty()) {
        qDebug() << "First pixel image[0]:" << data.images[0][0];
    }

    return data;
}
```

- **Файл neuralnetwork.cpp**

```
#include "neuralnetwork.h"
#include <cmath>
#include <QRandomGenerator>
#include <QDebug>
#include <QFile>
#include <QDataStream>

NeuralNetwork::NeuralNetwork(int inputSize, int hiddenSize, int outputSize)
```

```
    : inputSize(inputSize), hiddenSize(hiddenSize),
outputSize(outputSize) {

    weightsInputHidden.resize(inputSize);
    for (int i = 0; i < inputSize; ++i) {
        weightsInputHidden[i].resize(hiddenSize);
        for (int j = 0; j < hiddenSize; ++j) {
            weightsInputHidden[i][j] = QRandomGenerator::global()-
>generateDouble() * 0.02 - 0.01;
        }
    }

    weightsHiddenOutput.resize(hiddenSize);
    for (int i = 0; i < hiddenSize; ++i) {
        weightsHiddenOutput[i].resize(outputSize);
        for (int j = 0; j < outputSize; ++j) {
            weightsHiddenOutput[i][j] = QRandomGenerator::global()-
>generateDouble() * 0.02 - 0.01;
        }
    }

    hiddenBiases.resize(hiddenSize);
    for (int i = 0; i < hiddenSize; ++i) {
        hiddenBiases[i] = QRandomGenerator::global()->generateDouble()
* 0.02 - 0.01;
    }

    outputBiases.resize(outputSize);
    for (int i = 0; i < outputSize; ++i) {
        outputBiases[i] = QRandomGenerator::global()->generateDouble()
* 0.02 - 0.01;
    }
}
```

```
double NeuralNetwork::activateHiddenNeuron(int neuronIndex, const QVector<double>& inputs) {
    double sum = 0.0;
    for (int i = 0; i < inputs.size(); ++i) {
        sum += inputs[i] * weightsInputHidden[i][neuronIndex];
    }
    sum += hiddenBiases[neuronIndex];
    return std::tanh(sum);
}

QVector<double> NeuralNetwork::applySoftmax(const QVector<double>& logits) {
    QVector<double> softmaxOutput(logits.size());
    double maxLogit = *std::max_element(logits.begin(), logits.end());
    double sum = 0.0;
    for (int i = 0; i < logits.size(); ++i) {
        softmaxOutput[i] = std::exp(logits[i] - maxLogit);
        sum += softmaxOutput[i];
    }
    for (int i = 0; i < logits.size(); ++i) {
        softmaxOutput[i] /= sum;
    }
    return softmaxOutput;
}

int NeuralNetwork::getOutputSize() const {
    return outputSize;
}

QVector<double> NeuralNetwork::feedForward(const QVector<double>& inputs) {
    hiddenLayerOutputs.resize(hiddenSize);
    for (int i = 0; i < hiddenSize; ++i) {
        hiddenLayerOutputs[i] = activateHiddenNeuron(i, inputs);
```

```
    }

    QVector<double> logits(outputSize);
    for (int i = 0; i < outputSize; ++i) {
        double sum = 0.0;
        for (int j = 0; j < hiddenSize; ++j) {
            sum += hiddenLayerOutputs[j] * weightsHiddenOutput[j][i];
        }
        sum += outputBiases[i];
        logits[i] = sum;
    }

    return applySoftmax(logits);
}

void NeuralNetwork::saveWeights(const QString &filePath) const {
    QFile file(filePath);
    if (!file.open(QIODevice::WriteOnly)) {
        qWarning() << "Не вдалося відкрити файл для запису:" <<
filePath;
        return;
    }

    QDataStream out(&file);



    out << inputSize << hiddenSize << outputSize;



    for (const auto &row : weightsInputHidden)
        for (double val : row)
            out << val;
```

```
    for (const auto &row : weightsHiddenOutput)
        for (double val : row)
            out << val;


    for (double val : hiddenBiases)
        out << val;

    for (double val : outputBiases)
        out << val;

    file.close();
    qDebug() << "Ваги збережено в:" << filePath;
}

void NeuralNetwork::loadWeights(const QString &filePath) {
    QFile file(filePath);
    if (!file.open(QIODevice::ReadOnly)) {
        qWarning() << "Не вдалося відкрити файл для читання:" <<
filePath;
        return;
    }

    QDataStream in(&file);
    int inSize, hidSize, outSize;
    in >> inSize >> hidSize >> outSize;

    if (inSize != inputSize || hidSize != hiddenSize || outSize != outputSize)
{
        qWarning() << "Невідповідність розмірів при завантаженні
ваг";
        return;
    }
```

```
    for (auto &row : weightsInputHidden)
        for (double &val : row)
            in >> val;

    for (auto &row : weightsHiddenOutput)
        for (double &val : row)
            in >> val;

    for (double &val : hiddenBiases)
        in >> val;

    for (double &val : outputBiases)
        in >> val;

    file.close();
    qDebug() << "Ваги завантажено з:" << filePath;
}

QVector<double> NeuralNetwork::getOutput(const QVector<double>& input) {
    return feedForward(input);
}

void NeuralNetwork::train(const QVector<QVector<double>>& trainingInputs,
                          const QVector<QVector<double>>& trainingTargets,
                          int epochs, double learningRate) {
    for (int epoch = 0; epoch < epochs; ++epoch) {
        int correct = 0;
        for (int i = 0; i < trainingInputs.size(); ++i) {
            QVector<double> inputs = trainingInputs[i];
            QVector<double> targets = trainingTargets[i];
            QVector<double> outputs = feedForward(inputs);
```

```
        int predicted = std::distance(outputs.begin(), std::max_element(outputs.begin(), outputs.end()));
        int actual = std::distance(targets.begin(), std::max_element(targets.begin(), targets.end()));
        if (predicted == actual) correct++;

        QVector<double> outputErrors(outputSize);
        for (int j = 0; j < outputSize; ++j) {
            outputErrors[j] = targets[j] - outputs[j];
        }

        for (int j = 0; j < hiddenSize; ++j) {
            for (int k = 0; k < outputSize; ++k) {
                weightsHiddenOutput[j][k] += learningRate * outputErrors[k] * hiddenLayerOutputs[j];
            }
        }

        for (int k = 0; k < outputSize; ++k) {
            outputBiases[k] += learningRate * outputErrors[k];
        }

        QVector<double> hiddenErrors(hiddenSize);
        for (int j = 0; j < hiddenSize; ++j) {
            double error = 0.0;
            for (int k = 0; k < outputSize; ++k) {
                error += outputErrors[k] * weightsHiddenOutput[j][k];
            }
            hiddenErrors[j] = error * (1 - hiddenLayerOutputs[j] * hiddenLayerOutputs[j]);
        }

        for (int j = 0; j < inputSize; ++j) {
            for (int k = 0; k < hiddenSize; ++k) {
```

```
                weightsInputHidden[j][k] += learningRate *
hiddenErrors[k] * inputs[j];
            }
        }

        for (int j = 0; j < hiddenSize; ++j) {
            hiddenBiases[j] += learningRate * hiddenErrors[j];
        }
    }

    double accuracy = (double)correct / trainingInputs.size() * 100.0;
    qDebug() << "Epoch" << epoch + 1 << "Accuracy:" << accuracy
<< "%";
  }
}
```

- **Файл paintwidget.cpp**

```
#include "paintwidget.h"

PaintWidget::PaintWidget(QWidget *parent)
  : QWidget{parent}
{
  setFixedSize(280, 280);
  image = QImage(280, 280, QImage::Format_Grayscale8);
  image.fill(Qt::white);




}
```