НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені ІГОРЯ СІКОРСЬКОГО»
Кафедра інженерії програмного забезпечення в енергетиці

КУРСОВА РОБОТА

з дисципліни: «Методології розробки інтелектуальних комп'ютерних програм

на тему: « Проектування нейронних мереж »

Студента 3 курсу групи ТВ-22

напряму підготовки *121 Інженерія програмного забезпечення*

Оніщука М.І.

Керівник асистент Гейко О.О.

Національна оцінка _____

Кількість балів: _____ Оцінка: ECTS _____

Члени комісії:

асистент Гейко О.О.

(підпис) (вчене звання, науковий ступінь, прізвище та ініціали)

_____

(підпис) (вчене звання, науковий ступінь, прізвище та ініціали)

_____

(підпис) (вчене звання, науковий ступінь, прізвище та ініціали)

Київ – 2025

Національний технічний університет України

«Київський політехнічний інститут імені Ігоря Сікорського»

Інститут Навчально-науковий атомної та теплової енергетики

Кафедра Інженерія програмного забезпечення в енергетиці

Напрям підготовки 121 Інженерія програмного забезпечення

ЗАВДАННЯ

НА КУРСОВУ РОБОТУ СТУДЕНТУ

Оніщук Максима Івановича

1. Тема роботи « Проектування нейронних мереж »

Керівник курсової роботи асистент Гейко Олег Олександрович

2. Строк подання студентом роботи: «26» травня 2025 р.

3. Вихідні дані до проекту (роботи): Застосування методів штучного інтелекту

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити): Теоретичні основи методів штучного інтелекту. Реалізація логічних функцій, прогнозу часових рядів, розпізнавання образів, генетичних алгоритмів, Q-навчання.

5. Дата видачі завдання: «5» лютого 2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів виконання дипломного проекту (роботи)	Строк виконання етапів проекту (роботи)	Примітка
1.	Отримання завдання	05.02.2025-11.02.2025	Виконано
2.	Розділ 1 - 4	12.02.2025-10.03.2025	Виконано
3.	Розділ 5 - 9	11.03.2025-12.05.2025	Виконано
4.	Розділ 10	13.05.2025-19.05.2025	Виконано
5.	Здача оформленої курсової роботи	20.05.2025-26.05.2025	Виконано
6.	Захист курсової роботи	27.05.2025-09.06.2025	Виконано

Студент  Оніщук М.І.

(підпис) (прізвище та ініціали)

Керівник курсової роботи асистент Гейко Олег Олександрович

(підпис) (прізвище та ініціали) (прізвище та ініціали)

АНОТАЦІЯ

У даній курсовій роботі ми розглянемо та реалізовані основні методи штучного інтелекту, зокрема штучні нейронні мережі, генетичні алгоритми та Q-навчання. У практичній частині даної роботи було передбачено моделювання базових логічних функцій (AND, OR, NOT, XOR) за допомогою простих нейронів, прогнозування часових рядів, на основі алгоритму зворотного поширення помилки, а також побудова системи для розпізнавання образів із різними варіантами архітектури нейронної мережі. Окрім того в межах іншого завдання було реалізовано генетичний алгоритм для знаходження екстремумів певної функції на заданому інтервалі. Завершальна частина роботи включає створення інтелектуального агенту й демонстрацію механізму Q-навчання у вигляді взаємодії агенту з середовищем, змодельованим у формі графу станів.

Обсяг пояснювальної записки 58 аркушів, кількість ілюстрацій – 26, кількість додатків - 4.

ANNOTATION

In this coursework, we will consider and implement the basic methods of artificial intelligence, in particular artificial neural networks, genetic algorithms and Q-learning. The practical part of this work provided for modeling basic logical functions (AND, OR, NOT, XOR) using simple neurons, forecasting time series based on the error backpropagation algorithm, as well as building a system for pattern recognition with other options for the neural network architecture. choosing that within the framework of another task, a genetic algorithm was implemented to find the extrema of a certain function on a given interval. The final part of the work includes the creation of an intelligent agent and a demonstration mechanism for Q-learning in the form of the interaction of the agent with the environment, modeled in the form of a state graph.

The volume of the explanatory note is 58 sheets, the number of illustrations is 26, the number of appendices is 4.

Зміст

ВСТУП

Штучний інтелект (ШІ) є багатогранною дисципліною, спрямованою на наділення комп'ютерних систем когнітивними здібностями, що імітують людське мислення. Шляхом вивчення фундаментальних принципів розпізнавання образів та прийняття рішень у біологічних нейронних мережах, інженери розробляють програмні системи, здатні до автономного навчання, візуального сприйняття, аудіального аналізу та адаптивної взаємодії з довкіллям. Приклади повсякденної інтеграції ШІ включають системи розпізнавання облич у мобільних пристроях, автономне керування транспортними засобами, інтелектуальні голосові асистенти (наприклад, Siri, Alexa) та діалогові агенти (чат-боти) у комерційних платформах. Ці додатки наочно ілюструють поступове включення ШІ у повсякденну діяльність.

Експоненційний розвиток ШІ актуалізував практичні методології для автоматизації процесів прийняття рішень, підвищення точності прогнозування та оптимізації складних систем. Сучасний фахівець у галузі інформатики повинен володіти не лише глибоким розумінням математичних основ цих методів, але й бути здатним до їх практичної програмної реалізації, об'єктивної оцінки якості отриманих результатів та їх коректної інтерпретації у прикладних контекстах .

Ця курсова робота структурована з метою систематизації та поглиблення зазначених компетентностей шляхом виконання комплексу взаємопов'язаних завдань у чотирьох ключових областях:

1. Штучні нейронні мережі для логічного моделювання та прогнозування часових рядів. Метою є розробка нейронних структур, що емулюють базові булеві операції, а також тренування мережі для прогнозування елементів числового ряду із застосуванням алгоритму зворотного поширення помилки.
2. Розпізнавання образів. На основі користувацької вибірки зображень буде побудовано декілька архітектур мереж – від лінійних до глибоких. Проведеться порівняльний аналіз впливу кількості прихованих шарів, вибору функцій активації та оптимізаторів на точність класифікації.
3. Генетичні алгоритми. Для заданої неперервної функції студент реалізує еволюційний пошук глобального максимуму та мінімуму, проводячи аналіз збіжності та адекватності знайдених екстремумів шляхом їх співвіднесення з графічним представленням функції.

4. Інтелектуальні агенти та Q-навчання. Буде сформовано математичну модель середовища у вигляді графа станів. Далі, як за допомогою ручних обчислень, так і програмної імплементації, буде продемонстровано, як автономний агент, керуючись Q-алгоритмом навчання з підкріпленням, з кожною ітерацією ефективніше досягає цільового стану.

Такий набір вправ охоплює ключові парадигми машинного навчання: контрольоване, неконтрольоване, навчання з підкріпленням, а також еволюційні обчислення. Виконання даної роботи дозволить опанувати практичні методи побудови та тренування нейронних мереж різної глибини, набути досвіду обробки часових даних та комп'ютерного зору, зрозуміти переваги та обмеження генетичних методів оптимізації, та освоїти концепцію "навчання дією" (reinforcement learning) на прикладі Q-learning.

Отримані результати ілюструють ефективність різнопланових інструментів ШІ у вирішенні типових інженерних та наукових задач, підтверджуючи їхню інтегральну роль у сучасній практиці інформаційних технологій.

Розділ 1. Аналітичний огляд сфери використання програм на основі штучного інтелекту. Дослідження демонстраційних програмних продуктів.

Сучасне постіндустріальне техногенне суспільство характеризується безпрецедентним прискоренням інформатизації, і в цьому контексті штучний інтелект (ШІ) виступає як ключовий фактор розвитку. Його методи та технології вже глибоко інтегровані в повсякденне життя через інтелектуальні системи, веб-сервіси та мобільні додатки, що значно підвищують продуктивність, оптимізують комунікацію та гарантують безпеку.

Інтелектуальні системи вирізняються унікальною здатністю до розширення знань через навчання та самонавчання, що є основою для високого рівня автоматизації процесів прийняття управлінських рішень. Використовуючи комплекс лінгвістичних та логіко-математичних інструментів, вони можуть взаємодіяти з користувачем через природну мову. Завдяки цим характеристикам, ШІ-системи здатні розв'язувати задачі в умовах неповної та неоднорідної інформації, встановлювати складні причинно-наслідкові зв'язки, ефективно обробляти великі обсяги даних та здійснювати точне прогнозування на основі різноманітних джерел.

У результаті, потенціал застосування ШІ-систем є надзвичайно широким, охоплюючи такі завдання як кластеризація, моделювання, прогнозування, прийняття рішень у наступних сферах:

1. **Медицина**: ШІ допомагає в діагностиці, аналізі медичних даних і персоналізації лікування. Наприклад, аналіз зображень і прогнозування хвороб.
2. **Маркетинг**: Інструменти ШІ створюють персоналізований контент, аналізують поведінку клієнтів і автоматизують кампанії.

3. **Освіта**: ШІ підтримує персоналізоване навчання, автоматизацію оцінювання та підготовку матеріалів.
4. **Фінанси**: ШІ прогнозує ринкові тренди, виявляє шахрайство та оптимізує управління ризиками.
5. **Мультимедіа**: Генеративний ШІ створює тексти, аудіо, відео та зображення.
6. **Кібербезпека**: ШІ моніторить мережі та реагує на загрози в реальному часі.

Далі ми приступимо до поетапного аналізу демонстраційних програмних продуктів, розроблених на основі передових алгоритмів штучного інтелекту.

1.1 IBM Watson Health

IBM Watson оперує дуже складним і витонченим процесом, який передбачає ретельний аналіз великих обсягів медичних даних. Ці дані охоплюють широкий спектр джерел, починаючи від повних історій хвороб пацієнтів і глибоких наукових досліджень, закінчуючи ретельними клінічними випробуваннями і навіть потоками даних про пацієнтів у реальному часі. Використовуючи передові можливості обробки природної мови (NLP) разом з найсучаснішими алгоритмами машинного навчання, Watson не тільки розуміє, але й інтерпретує складні нюанси, закладені в цьому величезному і різноманітному масиві інформації. В результаті Watson стає вправним у генеруванні цінних ідей, виявленні закономірностей і наданні обґрунтованих рекомендацій, які допомагають медичним працівникам приймати обґрунтовані рішення і надавати персоналізовану допомогу пацієнтам.[1]

Відповідно до своєї здатності до комплексного аналізу Watson являється дуже потужним асистентом у сфері охорони здоров’я, який здатен швидко адаптуватись до нової інформації у медичних протоколах або результатів остаточних досліджень. Наприклад, під час встановлення аналізу система буде

враховувати не лише поточні симптоми пацієнта, але й генетичні дані, історію хвороби, стиль життя, а також актуальні дані з наукових журналів та медичних баз знань.

Окрім того, Watson може також використовуватись для прогнозування розвитку хвороб, оцінки ризиків, планування хірургічних втручань, а також у сфері медичних досліджень — для пришвидшення відкриттів нових препаратів і пошуку відповідних кандидатів для участі в клінічних випробуваннях.

В її основі — поєднання кількох технологій: обробка природної мови (NLP), алгоритми машинного навчання, семантичні знання з медичної галузі та знання з наукових джерел. Watson Health аналізує як структуровані, так і неструктуровані медичні дані: історії хвороби, діагнози, результати обстежень, генетичні профілі, клінічні дослідження та статті з медичних журналів.

Система використовує NLP для «читання» текстів, витягування медичних понять та встановлення зв’язків між симптомами, ліками, методами лікування. За допомогою алгоритмів машинного навчання Watson виявляє закономірності, оцінює ризики та формує персоналізовані рекомендації для лікарів. Інтеграція з клінічними інформаційними системами (EMR, FHIR) дозволяє Watson працювати безпосередньо в робочому процесі медичних працівників.

1.2 ChatGPT від OpenAI

ChatGPT , мабуть, найважливіший додаток, випущений за останнє десятиліття. Хоча він починався як чат-бот/технологічна демонстрація для великих мовних моделей (LLM) OpenAI, тепер це набагато більше. Його базові моделі штучного інтелекту можуть шукати в Інтернеті, створювати та оцінювати зображення, і, так, писати кумедні вірші про вашого начальника. А ще він може швидко узагальнювати величезні документи, генерувати комп'ютерний код та

успішно проходити тести, які мало хто з людей може пройти. Тепер це найкращий мультиінструмент.[2]

ChatGPT найкраще можна уявити як інтуїтивно зрозумілий інтерфейс для сучасних моделей штучного інтелекту. Завдяки формату діалогу користувачі можуть взаємодіяти з ним у природній мові — ставити запитання, просити про допомогу чи отримувати пояснення. Даний продукт може не тільки відповідати на запити, а й генерувати текст, формувати ідеї, вести розмови, редагувати та перекладати , писати та пояснювати програмний код різними мовами, генерувати зображення та навіть створювати електронні листи чи технічну документацію.

Ключовою перевагою даної моделі є універсальність, тобто користувач не зобов'язаний володіти якимись технічними знаннями, щоб користуватись інструментами штучного інтелекту. Для цього достатньо всього сформулювати запит звичайною мовою і дана система миттєво виконає завдання, часто пропонуючи кілька варіантів рішень. Це робить ChatGPT потужним помічником у різних сферах — від навчання і досліджень до бізнесу, розробки, дизайну чи побутових завдань.

Дана система, побудована на основі архітектури Transformer, що лежить в основі сучасних великих мовних моделей. Ядром ChatGPT є моделі серії GPT (Generative Pretrained Transformer), які проходять два етапи навчання: попереднє навчання (pretraining) на великому корпусі текстів з відкритих джерел і донавчання з використанням зворотного зв'язку від людей (RLHF — Reinforcement Learning from Human Feedback). Завдяки цьому ChatGPT здатен генерувати зв'язні тексти, підтримувати діалог, пояснювати складні теми, писати код, перекладати мови, формулювати ідеї, а також працювати як універсальний цифровий асистент.

Архітектура ChatGPT складається з кількох основних елементів: токенізатора, який розбиває вхідний текст на зрозумілі моделі одиниці (токени); багатошарового блоку трансформера з механізмом уваги (self-attention), що

дозволяє аналізувати взаємозв’язки між словами у контексті; і декодера, який формує текст у відповідь. Вся обробка відбувається в хмарній інфраструктурі, яка використовує потужні графічні процесори для швидкої обробки запитів.

1.3 Descript Overdub

Overdub – це передовий інструмент у Descript, який використовує штучний інтелект для створення клонів голосу з природним звучанням. Ця технологія дозволяє користувачам виконувати перетворення тексту в мовлення, пропонуючи безперешкодний спосіб додавання або редагування закадрового голосу без необхідності додаткових сеансів запису. За допомогою Overdub користувачі можуть змінювати аудіо, просто редагуючи текст, що значно спрощує процес редагування аудіо.

Descript Overdub базується на сучасних технологіях генерації синтетичного голосу, які належать до категорії Text-to-Speech (TTS) систем із використанням глибинних нейронних мереж. Вона базується на двох основних моделях. Перша — акустична модель, яка перетворює текст у спектрограму, що відображає частотні характеристики голосу у часі. Для цього часто використовують sequence-to-sequence архітектури з механізмом уваги, наприклад Tacotron 2 або трансформери. Друга — vocoder, який перетворює спектрограми у звуковий сигнал, використовуючи моделі типу WaveNet або HiFi-GAN, щоб отримати природне звучання.

Модель навчається на записах конкретного користувача, що дозволяє клонувати унікальні особливості голосу — тембр, інтонацію, ритм. Навіть кілька хвилин запису достатньо для створення якісного голосового профілю. Завдяки сучасним технологіям, Overdub синтезує природний і персоналізований голос із обмеженої кількості даних.

Розділ 2. Формулювання завдання на курсову роботу та його деталізація

В рамках курсової роботи заплановано виконання практичних завдань, які включають в себе декілька основних напрямів штучного інтелекту та машинного навчання. Кожне завдання спрямоване не лише на вивчення теоретичних основ відповідних методів, але й на розвиток практичних умінь їх ефективного використання в реальних ситуаціях.

2.1 Завдання моделювання формальних логічних функцій. Прогнозування часових рядів

Нейронна мережа — це метод штучного інтелекту (ШІ) , який навчає комп'ютери обробляти дані способом, натхненним людським мозком. Це тип процесу машинного навчання (МН) , який називається глибоким навчанням , що використовує взаємопов'язані вузли або нейрони в багатошаровій структурі, що нагадує людський мозок. Це створює адаптивну систему, яку комп'ютери використовують для навчання на своїх помилках та постійного вдосконалення. [3]

Кожен нейрон у цій системі приймає вхідні сигнали у вигляді векторів, множить їх на відповідні вагові коефіцієнти, підсумовує отримані значення, а потім пропускає результат через функцію активації, що додає потрібну нелінійність. Існують різні типи активаційних функцій — порогові, лінійні, сигмоїдальні — і кожна з них краще підходить для вирішення певних видів завдань. навантаженням.

2.1.1. Завдання моделювання формальних логічних функцій

Штучний нейрон може відтворювати основні логічні функції:

- **І (AND)** — вихід дорівнює 1 тільки тоді, коли всі входи мають значення 1.
- **АБО (OR)** — вихід становить 1, якщо хоча б один із входів дорівнює 1.
- **НІ (NOT)** — вихід є запереченням вхідного сигналу.
- **Виключне АБО (XOR)** — вихід дорівнює 1, коли входи відрізняються.

Для моделювання простих функцій «І», «АБО» та «НІ» достатньо одного нейрона з правильно налаштованими вагами та порогом активації. Натомість, функція «Виключне АБО» потребує використання багатошарової нейронної мережі, оскільки вона не є лінійно роздільною. завдань.

2.1.2. Завдання прогнозування часових рядів

Часовий ряд — це структура даних, де кожен елемент складається з двох частин: число та часова відмітка, асоційована з цим числом. Прикладів використання часових рядів в повсякденному житті досить багато: прогноз температури на найближчі кілька днів, коливання курсів валют.[4]

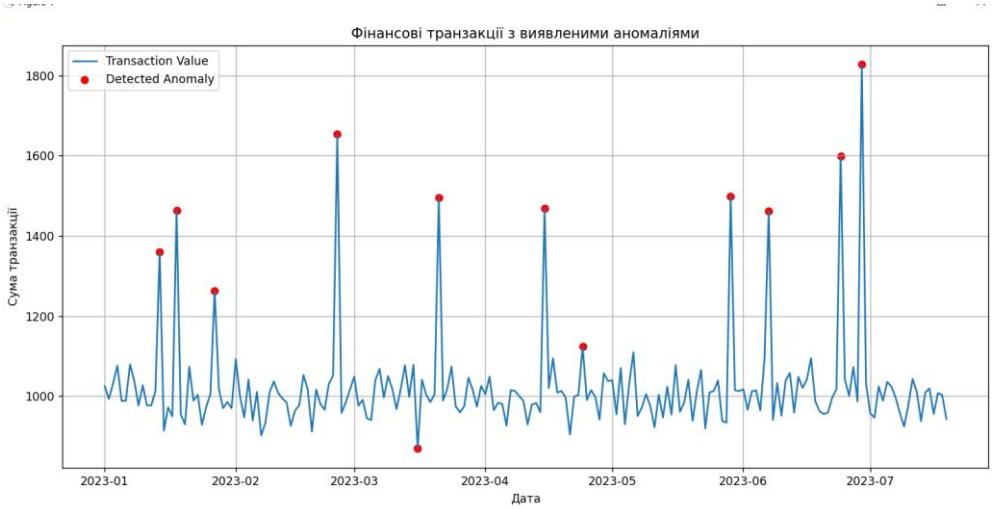

Рисунок 2.1 – Прикладу часового ряду

Нашою ж задачею буде прогнозування даних часових рядів, тобто передбачення даних історичних даних. В нашому випадку для цього треба

спроектувати та реалізувати однонейронну мережу з сигмоїдальною функцією активації.

Навчання цієї моделі здійснюється за допомогою алгоритму зворотного поширення помилки (back propagation), який полягає в поступовому оновленні вагових коефіцієнтів шляхом мінімізації різниці між передбаченими та реальними значеннями.

2.2 Завдання розпізнавання образів за допомогою штучних нейронних мереж

Також штучні нейронні мережі застосовуються для розпізнавання різноманітних образів, тим самим імітуючи роботу біологічної зорової системи.

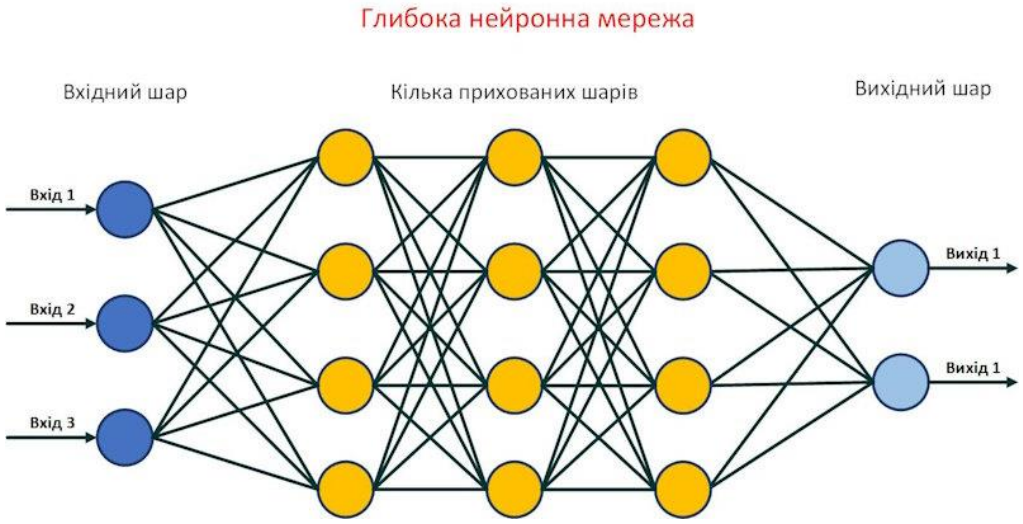

Рисунок 2.2 – Приклад структури нейромережі для розпізнавання образів

На верхньому рівні даної мережі розташовані нейрони(зображені синім кольором на рисунку 2.2), які приймають вхідний сигнал з навколишнього середовища і перетворюють їх у формат, який вже буде придатним для подальшої обробки. Якщо у біологічній системи це електричні імпульси, то у штучній мережі це є цифрові дані.

Ці закодовані сигнали передаються далі до нейронів проміжних рівнів. На нижньому рівні знаходяться ефекторні нейрони(позначені голубим кольором), які відповідають за передачу даних назовні, наприклад, до м’язів у живому організмі або, у випадку штучної мережі, на екран комп’ютера.

Згадані раніше нейрони проміжного рівня(позначені жовтим кольором), які розташовані між сенсорними та ефекторними нейронами, не взаємодіють безпосередньо з навколишнім світом. Вони утворюють приховані шари, які відповідають за основну обробку інформації.

У рамках нашого завдання було створено власну штучну нейронну мережу для розпізнавання образів, визначених у навчальному наборі. Окрім цього, було досліджено як різні архітектурні рішення та параметри навчання будуть впливати на точність класифікації. Для цього було побудовано кілька моделей: без прихованих шарів, з одним та кількома прихованими шарами, з варіаціями кількості нейронів у цих шарах. Також було протестовано різні функції активації, оптимізатори та функції втрат. Після отримання потрібних результатів тестування, було проведено аналіз, на основі якого було зроблено висновки про вплив глибини мережі, вибору активаційних функцій та методів навчання на якість розпізнавання.

2.3 Завдання: Генетичні алгоритми

Генетичний алгоритм — це обчислювальна техніка пошуку для знаходження наближених рішень для оптимізації моделей та задач пошуку. Генетичний алгоритм — це особливий тип еволюційного алгоритму, який використовує методи еволюційної біології, такі як спадковість, біологія мутацій та принципи вибору Дарвіна, щоб знайти оптимальну формулу для прогнозування або зіставлення зі зразком. Генетичні алгоритми часто є гарним вибором для методів прогнозування на основі регресії.[5]

У загальному вигляді генетичний алгоритм починається зі створення початкової популяції випадкових розв’язків задачі, які називають індивідами або хромосомами. Кожен індивід оцінюється за допомогою функції пристосованості (fitness), що визначає, наскільки добре він розв’язує поставлену задачу.

Далі алгоритм проходить через кілька поколінь, де в кожному з них відбуваються такі кроки:

1. **Відбір** — вибір кращих індивідів, які матимуть більше шансів передати свої характеристики наступному поколінню.
2. **Схрещування (кросовер)** — поєднання частин двох або більше обраних індивідів для створення нового потомства з ознаками батьків.
3. **Мутація** — випадкові зміни в деяких генах потомків, що додають різноманітність і допомагають уникнути локальних мінімумів.
4. **Заміненння** — формування нової популяції із потомків і, за потреби, кращих представників попереднього покоління.

Даний цикл буде повторюватись, поки не буде досягнуто певного критерію зупинки — наприклад, досягнення бажаного рівня пристосованості або вичерпання максимальної кількості поколінь.

У рамках дослідження було реалізовано генетичний алгоритм для пошуку екстремумів заданої неперервної функції. Процес включає формування початкової популяції, оцінку пристосованості кожного кандидата, застосування операцій відбору, схрещування і мутації, а також аналіз результатів для визначення найбільш оптимальних значень функції на заданому інтервалі. Такий підхід дозволяє ефективно знаходити глобальні максимуми та мінімуми навіть у складних задачах з багатьма локальними екстремумами.

2.4 Інтелектуальні агенти. Алгоритм Q-навчання

У сучасних системах штучного інтелекту велике значення займають інтелектуальні агенти – автономні програмні або апаратні одиниці, які отримують інформацію про навколишнє середовище через сенсори, аналізують її та, керуючись заданою метою, виконують дії, що впливають на це середовище. Ці агенти можуть змінювати свою поведінку залежно від ситуації, накопиченого досвіду та очікуваної вигоди. Така гнучкість є особливо корисною у задачах управління робототехнікою, логістикою, фінансами та іншими динамічними процесами, де неможливо заздалегідь передбачити всі можливі стани системи.

У рамках курсової роботи необхідно було розробити математичну модель інтелектуального агента та середовища його функціонування: визначити всі можливі стани, вказати цільовий стан, подати середовище у вигляді графа, де вершини відповідають станам, а ребра — діям агента, а також скласти відповідну матрицю суміжності. Далі, без використання програмування, вручну проілюструвати алгоритм Q-навчання, продемонструвавши дві послідовності переходів агента до цілі: спочатку випадкову, а потім із урахуванням накопиченої інформації в матриці Q, детально розраховуючи винагороди та оновлення значень Q, а також позначивши пройдений шлях на графі. Після цього, базуючись на моделі, слід реалізувати програму, яка автоматично побудує граф, оновлюватиме матрицю Q та показуватиме, як агент з кожною ітерацією все ефективніше досягає цільового стану.

2.1.1. Інтелектуальні агенти

Найпростіший тип агента діє за заздалегідь визначеними правилами виду IF умова THEN дія, реагуючи виключно на поточні вхідні сигнали сенсорів. Більш складні агенти мають внутрішнє представлення середовища, можуть оцінювати корисність різних дій і змінювати свою стратегію під час роботи. Залежно від способу взаємодії виділяють фізичних агентів, які за допомогою датчиків і виконавчих механізмів функціонують у реальному світі, та

інформаційних агентів, які аналізують послідовності подій і надають рекомендації користувачам або іншим програмам. Якщо агент здатен самостійно оновлювати та покращувати свої правила поведінки, його називають навчальним або автономним інтелектуальним агентом. Ефективність агента оцінюється за тим, наскільки успішно він досягає поставленої мети, виконуючи дії в умовах неповної інформації та невизначеності.

2.1.2. Алгоритм Q-навчання

Одним із найпоширеніших методів навчання агента є Q-навчання — підхід із підкріплювального навчання. Агент розглядає середовище як множину дискретних станів S, а можливі дії описуються множиною A. Виконуючи дію, агент отримує винагороду R[s, a], яка відображає, наскільки ця дія наблизила його до поставленої мети. В процесі навчання формується матриця Q, що зберігає очікувану сумарну вигоду для кожної пари «стан — дія». Спочатку всі значення Q встановлені в нуль (або мають початкові приблизні оцінки), через що агент діє переважно випадково. Після кожного кроку ці оцінки коригуються на основі отриманої винагороди та передбачуваних вигод у майбутньому. Повторюючи цикл «спостереження → дія → отримання винагороди → оновлення Q» багато разів, агент поступово навчається вибирати у кожному стані найкращу дію — ту, що має максимальне Q-значення. Головною перевагою Q-навчання є те, що він не потребує точної моделі середовища: достатньо мати змогу спостерігати стани й отримувати винагороду. Завдяки цьому метод широко застосовується у задачах навігації роботів, стратегічних іграх, адаптивному керуванні мережами та інших складних чи частково невідомих системах.

Розділ 3. Дослідження нейронних мереж. Моделювання роботи нейрона

Нейроінформатика — це напрямок штучного інтелекту, що вивчає методи подання та обробки інформації за допомогою нейронних мереж.

3.1 Інтелектуальні агенти. Алгоритм Q-навчання

Нервова система людини є надзвичайно складною: вона містить приблизно 100 мільярдів нейронів, які утворюють близько квадрильйона (10^{15}) зв'язків, що іноді простягаються на довжину в кілька метрів. Кожен нейрон, хоч і має багато спільних рис з іншими клітинами організму, відрізняється особливою здатністю приймати, обробляти та передавати електрохімічні сигнали, що забезпечують роботу мозкової мережі.

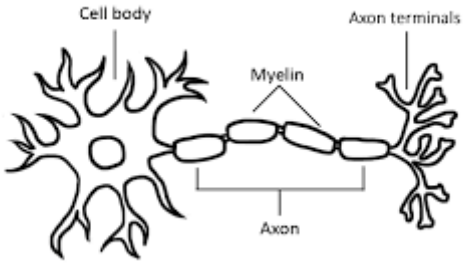

Рисунок 3.1 – Будова нейрона

На рисунку 3.1 представлено спрощене зображення будови нейрона. Дендритні відростки нейрона, які нагадують гілки дерева, з'єднуються із синапсами інших нейронів і отримують від них сигнали. У тілі нейрона всі ці сигнали сумуються: деякі з них активують нейрон, інші, навпаки, гальмують його. Якщо сумарний вплив стимулів перевищує певний поріг, нейрон активується і генерує електричний імпульс, який передається по аксону до інших клітин.

Сигнали, що надходять через синапси, збираються в тілі нейрона, де одні посилюють активацію, а інші її пригнічують. При досягненні критичного рівня збудження нейрон «спрацьовує», посилаючи електричний імпульс далі по аксону до наступних нейронів. Хоча у справжніх нервових мережах ця система має значно більше складних нюансів і варіацій, більшість штучних нейронних мереж моделюють саме цю основну логіку роботи біологічного нейрона.

3.1 Моделі штучних нейроелементів

Нейрон — це математична модель, яка імітує роботу біологічного нейрона для обробки інформації. Будова нейрона зображена на рисунку нижче:

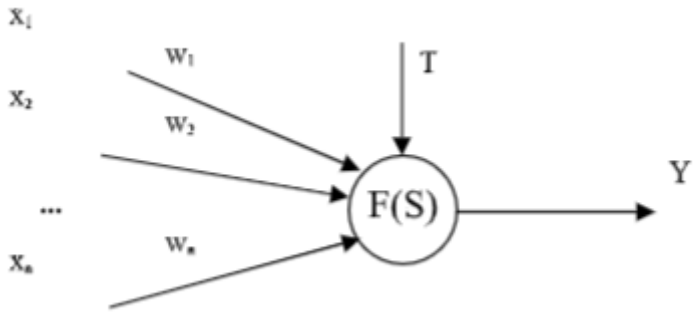

Рисунок 3.2 – Схема штучного нейрона

Будова нейрона включає такі компоненти:

- **Вхідні сигнали (вектори, x1…xn)** — числові значення, що надходять від попередніх нейронів або з зовнішнього середовища.
- **Ваги (коефіцієнти, w1…wn)** — числові параметри, які множать відповідні вхідні сигнали, визначаючи їх важливість.
- **Сума зважених входів(S)** — нейрон підсумовує всі вхідні сигнали, помножені на ваги.
- **Функція активації(F(S))** — нелінійна функція, яка приймає суму зважених входів і визначає вихід нейрона. Вона моделює поріг

активації та забезпечує здатність мережі навчатися складним залежностям.

- **Вихідний сигнал(T)** — результат роботи нейрона, який передається далі в мережі.
- **Порогове значення(T)** – значення, після якого нейрон переходить у стан збудження

Штучний нейрон обробляє вхідні сигнали за допомогою трьох основних етапів: зважування, підсумовування та активації.

На вхід подається набір сигналів $x=\{x_1,x_2,...,x_N\}$ де N — кількість входів. Кожен із цих сигналів пов'язаний з відповідною вагою $w=\{w_1,w_2,...,w_N\}$. Кожен елемент x_N вхідного вектора множиться на відповідну вагу w_N, яка визначає, наскільки цей сигнал буде посилено або приглушено — подібно до того, як синапси в біологічному нейроні можуть підсилювати або ослаблювати імпульси.

Після множення вхідних сигналів на вагові коефіцієнти, отримані значення підсумовуються за допомогою дискримінантної (або постсинаптичної) функції φ, яка формує сумарний потенціал. Цей потенціал передається до активаційної функції F(S), яка, у свою чергу, обчислює кінцевий скалярний вихід нейрона. Таким чином, роботу формального нейрона можна подати у вигляді формули 3.1.

$$y = F(S)(\nwarrow(w, x)) \tag{3.1}$$

Де:

- **x** — вектор вхідних значень (сигналів), що подаються на нейрон;
- **y** — результат (вихідне значення), який формує нейрон після обробки сигналів;
- **F(S)** — активаційна функція, яка визначає остаточний вихід на основі суми зважених сигналів;

- **φ** — дискримінантна (постсинаптична) функція, що виконує зважування та підсумовування вхідних сигналів;
- **w = {w_j}** — вектор вагових коефіцієнтів, що показує вплив кожного сигналу на нейрон;
- **w_0** — додатковий коефіцієнт (зсув або поріг), що використовується для регулювання чутливості нейрона до збудження.

Зважена сума S обчислюється наступним чином:

$$S = w1 * x_1 + w2 * x_2 + \ldots + wN * x_N \quad (3.2)$$

3.3 Функція активації

Функції активації — це математичні функції, які застосовуються до виходу кожного нейрона на рівні перед передачею на наступний рівень. Основні цілі функцій активації:

- **Додавання нелінійних властивостей**: Якщо в мережі відсутні функції активації, її поведінка зводиться до лінійної трансформації, незалежно від кількості шарів. Це суттєво обмежує здатність моделі розпізнавати складні залежності у даних. Активаційні функції забезпечують необхідну нелінійність, завдяки якій мережа може моделювати складні й гнучкі функціональні залежності.
- **Стабілізація вихідних значень**: Активаційні функції також сприяють обмеженню виходу нейрона в певному діапазоні. Це дозволяє уникнути проблем, пов’язаних із занадто великими або занадто малими значеннями градієнтів, що особливо важливо для стабільного й ефективного процесу навчання моделі.

Приклади функції активації:

1. Лінійна функція:

Формула:

$$F(S) = \begin{cases} 0 \; if \;\; S \;<= \; 0, \\ 1 \; if \;\; S \;>= \; 1, \\ S \; else \end{cases} \tag{3.3}$$

Графік:

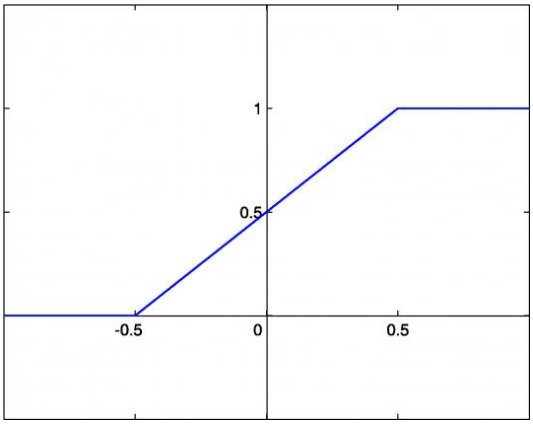

Рисунок 3.3 – Лінійна передавальна функція

2. Порогова функція:

Формула:

$$F(S) \; = \; \begin{cases} 1 \; if \; S \;>= \; 0, \\ 0 \; else \end{cases} \tag{3.4}$$

Графік:

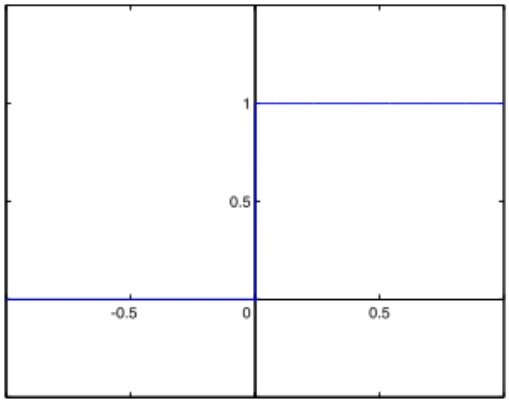

Рисунок 3.4 – Порогова передавальна функція

3. Сигмоїдальна функція:

Формула:

$$F(S) = \frac{1}{(1+exp(-S))} \tag{3.5}$$

Графік:

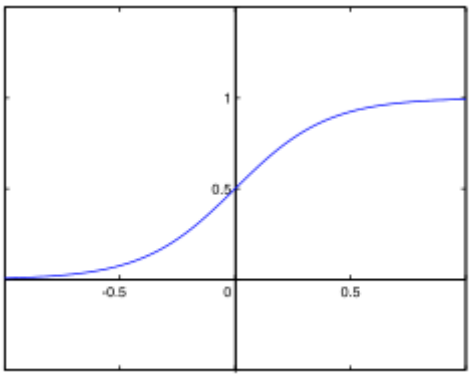

Рисунок 3.5 – Сигмоїдальна передавальна функція

Розділ 4. Розробка нейронної мережі перцептрон

Багатошаровий перцептрон (MLP) – це тип штучної нейронної мережі, що складається з кількох шарів нейронів. Нейрони в MLP зазвичай використовують нелінійні функції активації, що дозволяє мережі вивчати складні закономірності в даних. MLP є важливими в машинному навчанні, оскільки вони можуть вивчати нелінійні зв'язки в даних, що робить їх потужними моделями для таких завдань, як класифікація, регресія та розпізнавання образів.[7]

Принцип дії багатошарового перцептрона полягає в послідовному передаванні вхідних сигналів через шари нейронів. Кожен нейрон прихованого або вихідного шару обчислює зважену суму вхідних сигналів, до якої додається порогове значення (зсув), після чого результат обробляється за допомогою активаційної функції. Саме використання нелінійної функції активації дозволяє БШП моделювати складні, нелінійні залежності у даних.

На етапі навчання багатошаровий перцептрон використовує метод зворотного поширення помилки (back propagation), за допомогою якого мережа поступово коригує свої вагові коефіцієнти, зменшуючи помилку між фактичним і бажаним результатом. Повторюючи цей процес на багатьох прикладах, мережа навчається виконувати поставлене завдання з високою точністю.

Завдяки своїй гнучкості та здатності до узагальнення, багатошаровий перцептрон є основою для багатьох сучасних нейромережевих архітектур і відіграє ключову роль у розвитку штучного інтелекту.

4.1 Розробка перцпетрона

Під час виконання курсової роботи було створено базову модель багатошарового перцептрона (MLP), яка включає один прихований шар і один нейрон на виході.

4.1.1. Структура нейронної мережі

Мережа має 3 шари:

1. Вхідний шар: 3 входи (позначені як нейрони 1, 2, 3).
2. Прихований шар: 3 нейрони (нейрони 4, 5, 6).
3. Вихідний шар: 1 нейрон (нейрон 7).

Зв'язки між нейронами задані ваговими коефіцієнтами w_{ij}, які відображають ступінь впливу одного нейрона на інший. Схематичне зображення структури нейронної мережі наведено на рисунку 4.1.

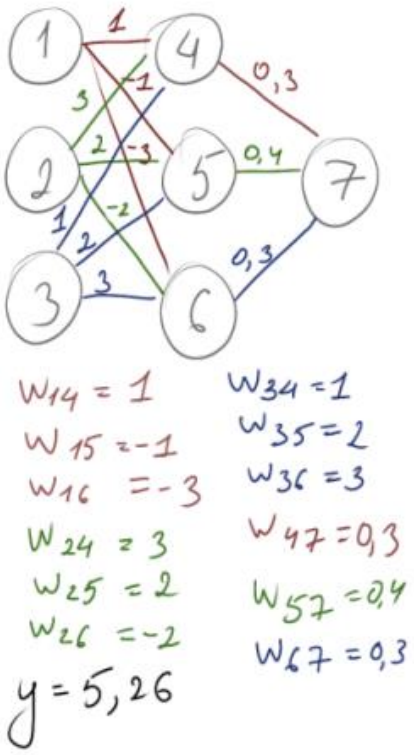

Рисунок 4.1 – Структура нейронної мережі

Ваги між шарами мають задані початкові значення, наприклад:

w14 = 1, w15 = −1, w16 = −3…

w47 = 0.3, w57 = 0.4, w67 = 0.3

Ці значення позначені різними кольорами на рисунку, що дозволяє візуально розрізнити рівні з’єднань у мережі.

4.1.2. Пряме поширення

Обчислюються зважені суми (S) для кожного нейрона прихованого та вихідного шарів, як показано на рисунку нижче.

$$S_4 = X_1 \cdot w_{14} + X_2 \cdot X_{24} + X_3 \cdot w_{34} = 10$$
$$S_5 = X_1 \cdot w_{15} + X_2 \cdot x_{25} + X_3 \cdot W_{35} = 9$$
$$S_6 = X_1 \cdot w_{16} + X_2 \cdot w_{26} + x_3 \cdot w_{36} = 2$$

Рисунок 4.2 – Обчислення зважених сум

Після обчислення застосовується для кожного нейрона функція активація сигмоїди за формулою 3.5.

4.1.3. Зворотне поширення помилки

Спершу виконується обчислення градієнтів: для кожного шару знаходиться похідна сигмоїдальної функції, що ілюструється на рисунку 4.3.

$$2.F(S_4) = \frac{1}{(1+\exp(-S_4))} = \frac{1}{(1+\exp(-10))} = 0,99995460$$
$$F(S_4)' = F(S_4) \cdot (1 - F(S_4)) = 0,00004539753$$
$$F(S_5) = \frac{1}{(1+\exp(-S_5))} = \frac{1}{(1+\exp(-9))} = 0,99987660$$
$$F(S_5)' = F(S_5) \cdot (1 - F(S_5)) = 0,00012338477$$
$$F(S_6) = \frac{1}{(1+\exp(-S_6))} = \frac{1}{(1+\exp(-2))} = 0,88079707$$
$$F(S_6)' = F(S_6) \cdot (1 - F(S_6)) = 0,10499359147$$
$$S_7 = F(S_4) \cdot w_{47} + F(S_5) \cdot w_{57} + F(S_6) \cdot w_{67} = 0,96417614 1$$
$$F(S_7) = \frac{1}{(1+\exp(-S_7))} = 0,72395715 = y$$

Рисунок 4.3 – Обчислення похідної сигмоїдальної функції

Далі розраховуємо похибку за формулою 4.2:

$$\Delta = y - Y \tag{4.2}$$

У результаті обрахунків отримуємо, що похибка дорівнює 4,536…, що представлено на рисунку 4.4.

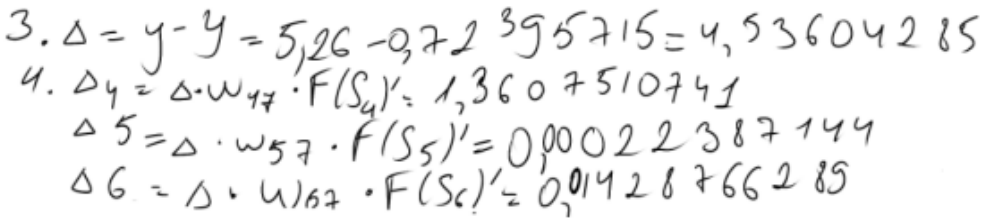

Рисунок 4.4 – Обчислення похибки

4.1.4. Оновлення ваг

Виберемо швидкість навчання $\eta = 0{,}1$.

Оновлення ваг здійснюється на рисунку нижче:

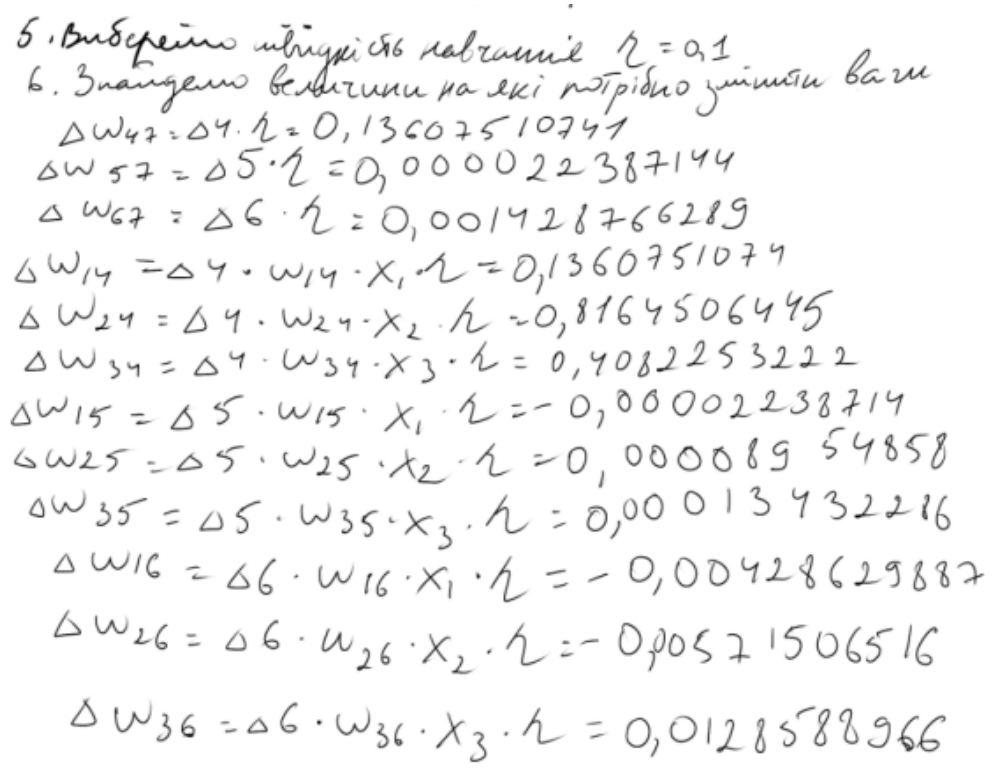

Рисунок 4.5 – Оновлення ваг

Отже, ми успішно розробили та навчили багатошаровий перцептрон з одним прихованим шаром. Проведено повний цикл обчислень: від початкової ініціалізації структури мережі та її вагових коефіцієнтів до прямого поширення сигналу, обчислення помилки та зворотного поширення для оновлення ваг. Завдяки цьому ми отримали нові значення вагових коефіцієнтів, що значно підвищили точність роботи мережі.

Розділ 5. Дослідження штучних нейронних мереж. Моделювання формальних логічних функцій. Прогнозування часових рядів

Штучні нейронні мережі (ШНМ) — це обчислювальні моделі, створені за мотивами людського мозку. Вони складаються з великої кількості підключених вузлів, кожен з яких виконує просту математичну операцію.[8]

Існують два основні методи навчання нейронних мереж: навчання з учителем (контрольоване) та навчання без учителя (неконтрольоване).

При контрольованому навчанні, яке є найпоширенішим, ваги зв’язків спочатку встановлюються випадковим чином, а потім поступово коригуються на основі порівняння фактичного виходу мережі з бажаним результатом. Цей процес повторюється до досягнення потрібної точності, після чого вагові коефіцієнти фіксуються. Проте деякі мережі можуть продовжувати навчання навіть під час експлуатації, щоб адаптуватися до змін у середовищі. Для ефективного навчання потрібна велика та різноманітна вибірка даних, оскільки тренування на одному прикладі може призвести до втрати раніше отриманих знань. Вхідні дані перед подачею до мережі необхідно чисельно кодувати та нормалізувати, що полегшує їх подальшу обробку. Після завершення навчання мережу перевіряють на нових даних, і якщо результати виявляються незадовільними, процес навчання повторюють.

Неконтрольоване навчання ґрунтується на самоорганізації нейронної мережі без зовнішньої оцінки правильності її роботи. Прикладом таких мереж є самоорганізовані карти, які автоматично налаштовують ваги, виявляючи закономірності у вхідних даних через змагання нейронів — змінюються лише ваги нейрона-переможця. Ефективність такого навчання залежить від ємності мережі, складності навчальних прикладів та обчислювальних можливостей системи. Загалом, для успішного навчання нейронної мережі потрібна грамотно

спроєктована архітектура, достатній обсяг якісних даних і потужне апаратне забезпечення.

5.1 Моделювання формальних логічних функцій за допомогою нейронів та нейронних мереж

Моделювання формальних логічних функцій за допомогою штучних нейронів і нейронних мереж здійснюється шляхом налаштування вагових коефіцієнтів та порогових значень нейронів таким чином, щоб їх вихід відповідав заданим логічним операціям. Розглянемо приклад моделювання основних логічних функцій.

Логічна операція «І» (AND) реалізується за допомогою одного штучного нейрона, у якого ваги встановлені як w1 = 1, w2 = 1, а порогове значення T дорівнює 1,5. Функція активації цього нейрона приймає значення F(S) = 1, якщо сума вхідних сигналів S дорівнює або перевищує 1,5, і F(S) = 0, коли S менша за 1,5. Наприклад, при вхідних значеннях (0,0), (0,1) або (1,0) сума сигналів S становить 0 або 1, що є менше за поріг, тому вихід нейрона Y дорівнює 0. Лише у випадку (1,1) сума дорівнює 2, що перевищує поріг, і вихід стає рівним 1, що відповідає таблиці істинності логічної операції AND. На рисунку 5.1 наведена схема штучного нейрона, який моделює логічну функцію «І».

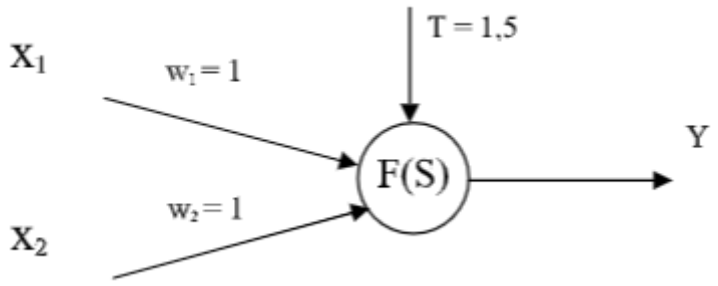

Рисунок 5.1 – Схема штучного нейрону, налаштованого на моделювання логічної функції “І”

Таблиця істинності логічної функції «І» надана в таблиці 5.1.

Таблиця 5.1 – таблиця істинності логічної функції “І”

X1	X2	Y
0	1	0
0	0	0
1	0	0
1	1	1

Логічна операція «АБО» (OR) також може бути змодельована одним штучним нейроном, у якого вагові коефіцієнти встановлені як $w_1 = 1$ і $w_2 = 1$, але порогове значення становить T = 0,5. Функція активації визначається так: F(S) дорівнює 1, якщо сума вхідних сигналів S є більшою або рівною 0,5, і 0 у випадку, коли S менша за 0,5. При вхідних значеннях (0,0) сума S дорівнює 0, тому вихід нейрона становить 0. Для будь-якої іншої комбінації вхідних сигналів сума дорівнює 1 або 2, що перевищує поріг, і вихід стає рівним 1, що відповідає таблиці істинності функції OR. Схема штучного нейрона, який реалізує логічну функцію «АБО», наведена на рисунку 5.2.

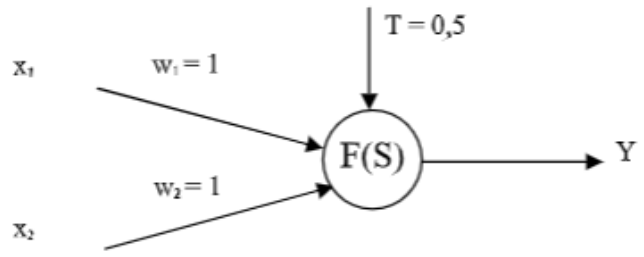

Рисунок 5.2 – Схема штучного нейрону, налаштованого на моделювання логічної функції “АБО”

Таблиця істинності логічної функції «АБО» надана в таблиці 5.2.

Таблиця 5.2 – таблиця істинності логічної функції “АБО”

X1	X2	Y
0	1	1
0	0	0
1	0	0
1	1	1

Логічна операція «НІ» (NOT) може бути змодельована одним нейроном із негативним ваговим коефіцієнтом w = -1,5 та пороговим значенням T = -1. Функція активації в цьому випадку визначається так: F(S) дорівнює 1, якщо сумарний сигнал S більше за -1, і 0, якщо S менше або дорівнює -1. При вході x = 0 сигнал обчислюється як 0 * (-1,5) = 0, що більше за -1, тому вихід Y = 1. Для входу x = 1 сигнал S становить 1 * (-1,5) = -1,5, що менше за -1, і вихід дорівнює 0, що відповідає логіці заперечення. На рисунку 5.3 наведена схема штучного нейрона, який моделює логічну функцію «НІ» .

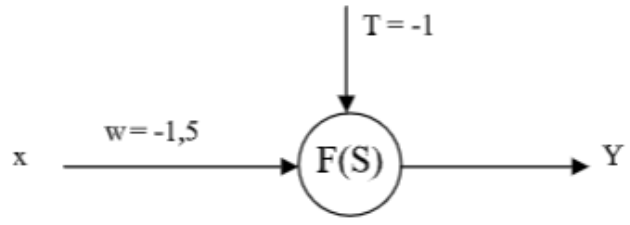

Рисунок 5.3 – Схема штучного нейрону, налаштованого на моделювання логічної функції “НІ”

Таблиця істинності логічної функції «НІ» надана в таблиці 5.3.

Таблиця 5.3 – таблиця істинності логічної функції “НІ”

X	Y
1	0
0	1

Логічна операція «Виключне АБО» (XOR) не може бути реалізована за допомогою одного нейрона, тому для її моделювання застосовується багатошарова нейронна мережа. Вона складається з двох нейронів у першому шарі з вагами w11 = 1, w12 = -1 та w21 = -1, w22 = 1 і порогом T = -0,5, а також одного вихідного нейрона з вагами w31 = 1, w32 = 1 і порогом T = 0,5. Функція активації у кожного нейрона визначається так: F(S) = 1, якщо сума S дорівнює або перевищує 0,5, і 0, якщо S менша за 0,5. Наприклад, при вхідних значеннях (1,1) нейрони першого шару обчислюють суми S1 = 0 та S2 = 0, тому кожен видає 0, а вихідний нейрон також генерує 0, що відповідає результату XOR. Для вхідних (0,1) значень S1 = -1 (0), S2 = 1 (1), вихідний нейрон отримує суму 1 і повертає 1, що також відповідає функції XOR. На рисунку 5.4 наведена схема нейронної мережі, що моделює логічну функцію «Виключне АБО».

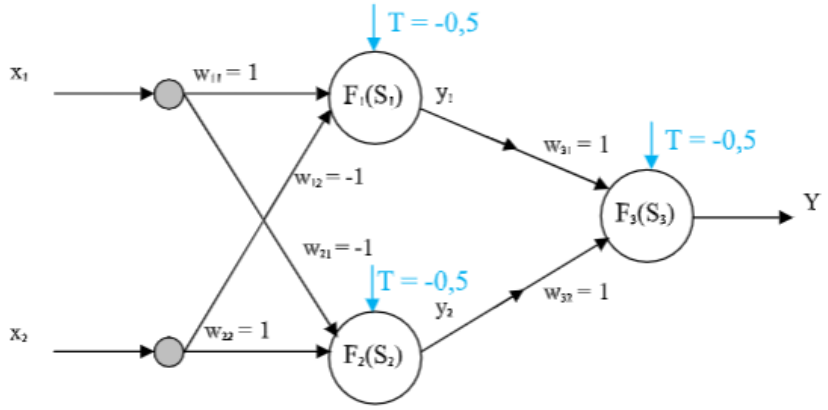

Рисунок 5.4 - Схема штучної нейронної мережі, налаштованої на моделювання логічної функції “ Виключне АБО”

Таблиця істинності логічної функції «Виключне АБО» є в таблиці 5.4.

Таблиця 5.4 – таблиця істинності логічної функції “ Виключне АБО”

X1	X2	Y
0	1	1
0	0	0
1	0	1
1	1	0

5.2 Прогнозування часових рядів

Часовий ряд — це послідовність значень деякої змінної, впорядкована за часом. Кожне значення в такому ряді відповідає певному моменту або інтервалу часу. Часові ряди використовуються для аналізу динаміки процесів, прогнозування майбутніх значень, виявлення трендів, сезонних коливань та аномалій у даних. Приклади часових рядів — щоденні температури, курси валют, продажі товарів по місяцях, показники сенсорів у промисловості тощо.

Один із сучасних методів прогнозування базується на застосуванні штучних нейронних мереж. Навіть проста модель з одним нейроном здатна навчитися передбачати наступні елементи часового ряду. Цей нейрон обчислює зважену суму трьох попередніх значень і використовує сигмоїдальну функцію активації для формування прогнозованого значення, що описано у формулах 5.1 та 5.2.

$$S_i = x_{i-3} * w_3 + x_{i-2} * w_2 + x_{i-1} * w_3 \quad (5.1)$$

$$Y = \frac{10}{1+e\text{^}(-S)} \quad (5.2)$$

де x_{i-n} – попередні значення мітки, w_n – вагові коефіцієнти, Y – прогноз

Навчання моделі здійснюється за методом зворотного поширення помилки з метою мінімізації квадратичної похибки між очікуваними та реальними значеннями.

Розглянемо приклад часового ряду, наданий в таблиці 5.5.:

Таблиця 5.5 – Приклад часового ряду

X1	X2	X3	X4	X5	X6	X7	X8	X9	X10	X11	X12	X13	X14	X15
0.58	3.38	0.91	5.80	0.91	5.01	1.17	4.67	0.60	4.81	0.53	4.75	1.01	5.04	1.07

Перші 13 значень нашого ряду використовуються для навчання моделі, дві останні – для тестування її.

Розділ 6. Дослідження нейронних мереж Хопфілда

Нейронна мережа Хопфілда — це вид рекурентної штучної нейронної мережі, яка використовується для розв’язання задач асоціативної пам’яті, оптимізації та розпізнавання образів. Вона складається з нейронів, які взаємно з’єднані, причому кожен нейрон пов’язаний з усіма іншими (крім самого себе) симетричними вагами.

Основні характеристики мережі Хопфілда:

- **Рекурентність:** Кожен нейрон може впливати на всі інші нейрони, а стан мережі оновлюється поступово.
- **Динаміка роботи:** Мережа поступово переходить у стійкий стан (атрактор), який відповідає одному із збережених шаблонів або рішень.
- **Асоціативна пам’ять:** Мережа здатна «відновлювати» повні патерни на основі часткових або зашумлених входів.

6.1 Навчання мережі Хопфілда

Навчання в нейронній мережі Хопфілда має особливий характер, оскільки воно не потребує класичного процесу навчання з учителем, як у багатьох інших

нейромережевих моделях. Замість цього вона запам’ятовує певну кількість шаблонів (прикладів), які пізніше може відтворювати або доповнювати при подачі неповної чи зашумленої інформації. Цей процес називається асоціативним навчанням або навчанням пам’яті.

Як відбувається навчання в мережі Хопфілда:

1. **Представлення навчальних зразків:** Кожен шаблон, який потрібно запам’ятати, перетворюється у вектор, елементи якого зазвичай мають значення +1 або -1. Наприклад, бітова послідовність 0 і 1 замінюється відповідно на -1 і +1 для математичної зручності.
2. **Ініціалізація ваг:** На початку ваги між нейронами встановлюються як нульові. У процесі навчання вони поступово налаштовуються згідно з обраними зразками.
3. **Використання правила Хебба:** Для збереження кожного шаблону застосовується спеціальне правило корекції ваг (так зване правило Хебба). Згідно з цим правилом, вага зв’язку між двома нейронами збільшується, якщо вони мають однакові значення в шаблоні (обидва +1 або обидва -1), і зменшується, якщо їх значення протилежні.Важливо, що вага зв’язку нейрона із самим собою wiiw_{ii}wii завжди дорівнює нулю, тобто самозв’язки відсутні.
4. **Завершення навчання:** Після того як усі шаблони оброблено й ваги обчислено, навчання вважається завершеним. Мережа готова до використання: вона може розпізнавати чи відновлювати збережені образи за допомогою механізму асоціативного згадування.

Розділ 7. Використання навчання з вчителем для задач розпізнавання образів, класифікації об’єктів

Контрольоване машинне навчання - це алгоритм, який використовує марковані навчальні дані для прогнозування результатів немаркованих даних. У контрольованому навчанні ви використовуєте добре марковані дані для навчання машини. Поряд з неконтрольованим навчанням і навчанням з підкріпленням, це одна з трьох основних парадигм машинного навчання.[9]

Основна ідея полягає в тому, що Мережа вчиться знаходити залежність між вхідними прикладами та відповідними виходами. Після успішного навчання вона здатна класифікувати нові (раніше не бачені) об'єкти з подібними характеристиками.

Всього навчання контрольованого навчання поділяється на наступні етапи**:**

1. **Формування навчальної вибірки:** Створюється набір прикладів, де кожен вхідний вектор (наприклад, зображення) має відповідну правильну мітку (назву об’єкта, клас). Наприклад: Вхід — зображення цифри "5", Вихід (мітка) — клас "цифра 5".
2. **Подача прикладів у нейронну мережу:** Мережа обчислює свій вихід, порівнюючи його з правильним результатом із навчальної вибірки.
3. **Обчислення помилки:** Різниця між очікуваним (правильним) і фактичним виходом визначається як помилка мережі.
4. **Оновлення ваг:** На основі обчисленої помилки за допомогою алгоритму, як-от зворотне поширення помилки (back propagation), ваги мережі коригуються, щоб зменшити помилку в наступних ітераціях.
5. **Повторення процесу:** Процес повторюється для великої кількості прикладів протягом багатьох епох, поки помилка не стане мінімальною або не буде досягнута задана точність.

Наприклад бувають наступні архітектури, які використовують навчання вчителем:

- Багатошаровий перцептрон (MLP) — класична модель для класифікації.
- Згорткові нейронні мережі (CNN) — спеціально розроблені для роботи з зображеннями.
- Рекурентні нейронні мережі (RNN) — підходять для послідовних даних, наприклад, відео або мовлення.

Розділ 8. Застосування генетичних алгоритмів для оптимізації функції

Генетичний алгоритм — це метод оптимізації та пошуку рішень, натхненний природним добором і механізмами еволюції в біології. Його суть полягає в ітеративному удосконаленні набору можливих рішень (які називаються "особинами" або "хромосомами") з використанням таких процесів, як селекція, схрещування та мутація. На рисунку нижче наведено просту схему роботи генетичного алгоритму:

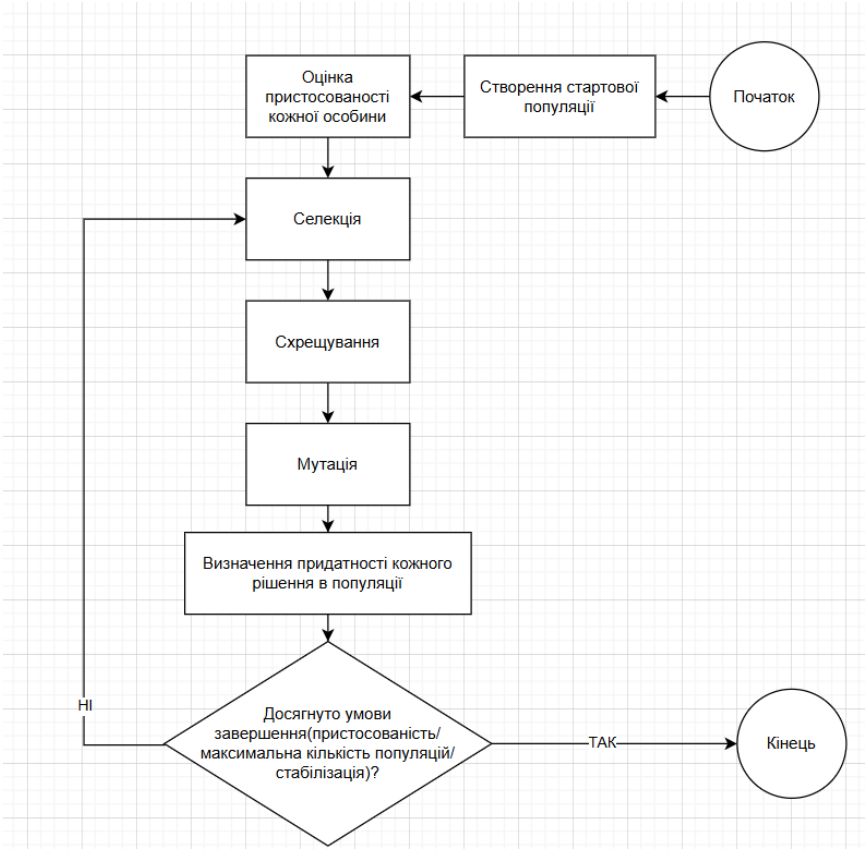

Рисунок 8.1. – Загальна схема генетичного алгоритму

Цикл роботи генетичного алгоритму складається з кількох послідовних етапів, які безперервно повторюються до досягнення певної умови завершення. На початковому етапі створюється початкова популяція — множина потенційних розв’язків задачі, де кожне рішення подається у вигляді хромосоми. Здебільшого ці хромосоми генеруються випадковим чином, щоб охопити

якомога ширший простір пошуку, однак у деяких випадках застосовують евристики або попередні знання для формування частини початкової популяції.

Наступним кроком є оцінювання кожної хромосоми за допомогою функції пристосованості. Ця функція визначає, наскільки добре певне рішення виконує поставлену задачу. Хромосоми з вищими значеннями функції пристосованості вважаються кращими кандидатами для подальшого відбору. Відповідно до цих оцінок здійснюється селекція — вибір особин, які будуть брати участь у створенні нового покоління. Існують різні методи селекції, але загальна ідея полягає в тому, щоб дати кращим рішенням більший шанс на передавання своїх ознак нащадкам, зберігаючи при цьому генетичне різноманіття.

Після відбору відбувається схрещування, або кросовер — процес комбінування генетичної інформації від двох батьків для створення одного або кількох нащадків. Схрещування дозволяє об’єднувати успішні ознаки батьківських хромосом у надії створити ще більш вдалі рішення. Існують різні типи схрещування: одноточкове, двоточкове, рівномірне тощо. Вибір методу схрещування і його ймовірність впливають на якість нових рішень.

Після схрещування проводиться мутація — випадкова зміна вмісту хромосоми. Мутація потрібна для підтримки варіативності в популяції та запобігання застряганню алгоритму в локальних екстремумах. Вона може проявлятися як інверсія біта, заміна значення в числовій хромосомі або перестановка елементів. Ймовірність мутації зазвичай невелика, щоб уникнути руйнування вже знайдених вдалих рішень.

Після виконання схрещування та мутації формується нове покоління, яке замінює або частково витісняє попереднє. Деякі з найкращих рішень попереднього покоління можуть зберігатися без змін — це називається елітизмом і дозволяє не втратити вже досягнуті вдалі варіанти. Повний цикл — оцінка, селекція, схрещування, мутація та оновлення популяції — повторюється до моменту завершення роботи алгоритму, що визначається заздалегідь

встановленими умовами, наприклад, кількістю поколінь або досягнутим рівнем якості розв’язання.

У результаті багаторазового повторення цього циклу популяція поступово еволюціонує, вдосконалюючи свої характеристики та наближаючись до найкращого можливого рішення. Генетичний алгоритм імітує процес природного добору, де виживають найкращі, а завдяки комбінації спадковості та випадковості виникають нові, потенційно ефективні рішення.

Розділ 9. Інтелектуальні агенти. Створення алгоритму Q-навчання

Інтелектуальні агенти становлять основу багатьох сучасних систем штучного інтелекту. Їх головне завдання полягає в тому, щоб сприймати навколишнє середовище, аналізувати отриману інформацію, приймати рішення і виконувати дії, спрямовані на досягнення визначених цілей. При цьому важливо не лише дати формальне визначення інтелектуального агента, а й усвідомити механізми, які забезпечують його здатність адаптуватися до змін та вдосконалювати поведінку в умовах, що постійно змінюються.

Серед сучасних підходів до формування адаптивної поведінки агентів особливе місце посідає навчання з підкріпленням, зокрема метод Q-навчання. Цей підхід ґрунтується на поступовому накопиченні знань через безпосередню взаємодію агента із середовищем. У цьому розділі буде розглянуто поняття інтелектуального агента, його основні властивості та функції, а також принципи реалізації Q-навчання як одного з ключових методів формування поведінки таких систем.

9.1 Створення алгоритму Q-навчання

Алгоритм Q-навчання працює за циклом взаємодії агента зі середовищем:

1. **Ініціалізація**: спочатку створюється таблиця Q-значень (Q-таблиця), де кожній парі «стан-дія» відповідає значення Q, яке показує, наскільки вигідно виконувати цю дію в даному стані. Спочатку всі Q-значення можуть бути встановлені у нуль або випадкові значення.
2. **Вибір дії**: агент знаходиться в певному стані і обирає дію. Для балансу між дослідженням і використанням знань зазвичай застосовують ε-жадібну стратегію: з ймовірністю ε агент вибирає

випадкову дію (дослідження), а з ймовірністю 1-ε — дію з найвищим Q-значенням (використання знань).

3. **Виконання дії і отримання винагороди**: агент виконує вибрану дію, середовище переходить у новий стан, і агент отримує винагороду за цю дію.
4. **Оновлення Q-значення**: агент оновлює Q-значення для пари «попередній стан — вибрана дія»
5. **Повторення**: кроки вибору дії, виконання і оновлення Q-значень повторюються багато разів, поки агент не навчиться вибирати дії, що максимізують сумарну винагороду.

Розділ 10. Візуалізація роботи програми

У рамках курсової роботи було виконано низку завдань, що охоплюють ключові напрямки штучного інтелекту та машинного навчання. Зокрема, було реалізовано моделювання логічних функцій і прогнозування часових рядів із використанням штучних нейронних мереж, проведено задачі з розпізнавання образів, здійснено оптимізацію функцій за допомогою генетичних алгоритмів, а також реалізовано навчання інтелектуальних агентів за методом Q-навчання.

10.1 Реалізація завдання моделювання формальних логічних функцій. Прогнозування часових рядів

мережі, оскільки вона не є лінійно роздільною. завдань.

10.1.1. Завдання моделювання формальних логічних функцій

У першій частині роботи було реалізовано програму, призначену для побудови та навчання штучної нейронної мережі з метою моделювання основних логічних операцій — І (AND), АБО (OR), НІ (NAND) та Виключне АБО (XOR). Крім того, програма має можливість адаптації для задач прогнозування часових рядів.

Для розв’язання поставлених задач була створена багатошарова нейронна мережа, що включає вхідний шар, прихований шар з трьома нейронами та один вихідний нейрон. У якості активаційної функції як у прихованому, так і у вихідному шарах застосовується сигмоїда, що дає змогу ефективно відображати нелінійні залежності у даних.

Навчання нейронної мережі здійснюється за допомогою алгоритму зворотного поширення помилки (back propagation), який забезпечує корекцію вагових коефіцієнтів на основі різниці між фактичними вихідними значеннями мережі та очікуваними результатами.

На наступному рисунку зображено результат моделювання логічних функцій:

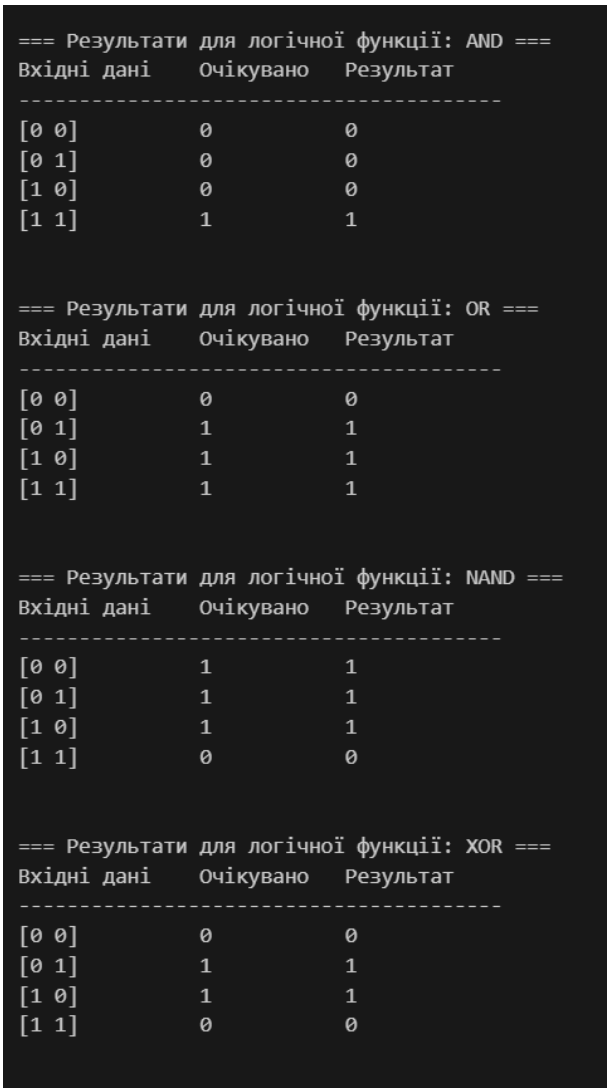

Рисунок 10.1 - Результат моделювання базових логічних функцій

10.1.2. Завдання прогнозування часових рядів

У другій частині проєкту реалізовано штучну нейронну мережу з трьома входами, прихованим шаром із трьома нейронами та одним виходом. Мета — навчити мережу відтворювати логічну функцію за таблицею істинності.

Навчання відбувається на основі випадково ініціалізованих ваг та сигмоїдальної функції активації. У процесі прямого поширення обчислюються значення прихованого шару та виходу. Потім визначається помилка між

очікуваним і фактичним результатом, і ваги коригуються методом зворотного поширення з урахуванням градієнта.

Процес повторюється багаторазово до досягнення точності. Після навчання мережу тестують, порівнюючи її вихід з очікуваними результатами. Реалізація демонструє основи роботи ШНМ: обробку сигналів, навчання й моделювання логічних залежностей.

На діаграмі зображеній нижче наведено графік зменшення сумарної помилки прогнозування на нашому наборі:

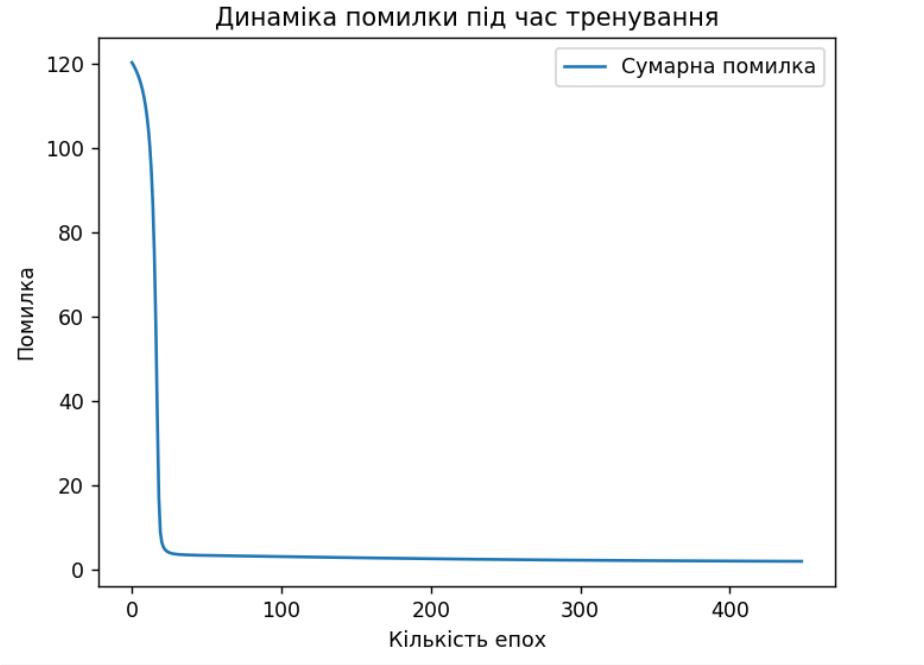

Рисунок 10.2 – Діаграма зміни помилки

Результат прогнозу зображено нижче:

```
Прогнозуємо x14: 5.278150645943769
Дійсне x14: 5.04
Прогнозуємо x15: 0.7686029701916877
Дійсне x15: 1.07
Похибка на тестових даних: 16.45%
```

Рисунок 10.3 – Результат прогнозу часового ряду

10.2 Реалізація розпізнавання образів за допомогою штучних нейронних мереж.

У даному завданні було побудовано штучну нейронну мережу для розпізнавання наступних геометричних фігур:

- Коло
- Квадрат
- Ромб
- Еліпс
- Трикутник

Кожен образ кодувався як бінарна матриця розміром 6 на 6, де одиниці позначали частини букви, а нулі — фон. Ці матриці були перетворені у вектори розміром 36 елементів, що ставали на вході нейронної мережі.

Мережа містить три шари: вхідний з 36 нейронами (відповідає розміру вхідного вектора), прихований шар(опціональний) та вихідний шар, який містить масив довжиною три, який описує певну фігуру

Навчання відбувалося за методом зворотного поширення помилки з градієнтним спуском і додаванням моментума для стабільнішої та швидшої збіжності. У кожній епосі обчислювався вихід мережі для всіх вхідних прикладів, після чого визначалась похибка між передбаченим і правильним класом, і на її основі оновлювались ваги з використанням похідної сигмоїди(кодом передбачено використання також функції ReLu та тангенціального еліпса).

Підсумкові результати роботи програми наведено на рисунках 10.3 та 10.4. Тут було використано сигмоїдальну функцію активації та прихованого шару з 8 нейронами:

Рисунок 10.3 – Класифікація спотворених фігур мережею

```
Епоха 0, середня втрата: 0.27277382793340477
Епоха 100, середня втрата: 0.22068035309154058
Епоха 200, середня втрата: 0.19667453790157335
Епоха 300, середня втрата: 0.17882194362838164
Епоха 400, середня втрата: 0.164046709595388
Епоха 500, середня втрата: 0.1511080976111065
Епоха 600, середня втрата: 0.139396337542107
Епоха 700, середня втрата: 0.12858742622244107
Епоха 800, середня втрата: 0.11850196953146012
Епоха 900, середня втрата: 0.10905279064493545
Точність: 80.0%
```

Рисунок 10.4 – Динаміка зменшення помилки та точність

10.3 Реалізація генетичного алгоритму

У цьому завданні було розв’язано проблему пошуку максимуму та мінімуму функції Y(x) = cos(x^2) / x на відрізку від 0 до 5 за допомогою генетичного алгоритму — евристичного методу, що моделює природний відбір для знаходження оптимального рішення.

Спочатку була визначена цільова функція та основні параметри алгоритму: межі пошуку XMIN і XMAX, точність дискретизації E, розмір популяції (30 індивідів), кількість поколінь (50) і ймовірність мутації (2%). Ці налаштування впливають на деталізацію пошуку, тривалість обчислень і частоту змін у генетичному матеріалі хромосом під час еволюції.

Генетичний алгоритм оперує популяцією бінарних хромосом, кожна з яких кодує потенційне рішення задачі. Спочатку створюється початкова випадкова популяція. На кожній ітерації оцінюється функція пристосованості (фітнес), яка

визначає, наскільки добре кожна хромосома відповідає поставленій меті — у цьому випадку це квадратична різниця між значенням функції Y(x) та середнім значенням по популяції, що сприяє пошуку рішень, віддалених від середнього.

Для формування наступного покоління застосовується турнірна селекція: з двох випадково обраних особин вибирається та, що має кращий фітнес. Обрані батьки проходять операцію кросовера — обмін частинами хромосом, що дозволяє поєднати їх характеристики та отримати потенційно кращих нащадків. Після кросовера кожна дитина зазнає мутації з невеликою ймовірністю, що вводить випадкові зміни, підтримуючи генетичне різноманіття та допомагаючи уникнути застрягання у локальних оптимумах.

Цей цикл повторюється протягом заданої кількості поколінь, поступово покращуючи якість рішень. Наприкінці алгоритм повертає хромосому з найкращим (максимальним або мінімальним, залежно від налаштувань) значенням фітнесу, яку декодують у відповідне значення x, а також обчислюють значення цільової функції.

Для кращого розуміння результатів було створено графік функції, що наведений на рисунку 10.5:

Рисунок 10.5 – Графік функції з визначеними екстремумами

```
Max: x=0.0010, f(x)=999.9999999994999
Min: x=1.7251, f(x)=-0.5717441698423242
Фінальна середньо квадратична помилка: 0.00000000000002546521
```

Рисунок 10.6 – Координати точок екстремуму і значення помилки

10.4 Реалізація алгоритму Q-навчання

У цьому завданні реалізовано математичну модель інтелектуального агента та його зовнішнього середовища у вигляді графа, де вершини відображають стани агента, а ребра — дії, які переводять його з одного стану в інший. На основі заданої матриці переходів та винагород буде побудована матриця суміжності цього графа. Варіант завдання наведено на рисунку 10.7.

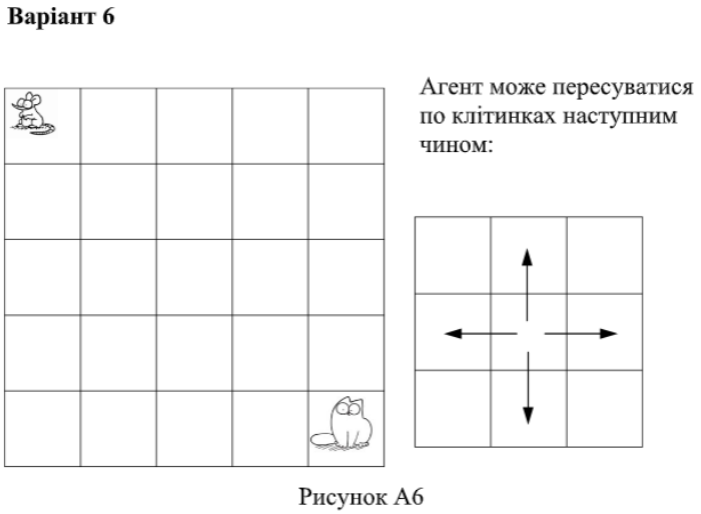

Рисунок 10.7. – Варіант завдання алгоритму Q-навчання

Спочатку створюється матриця винагород (R), яка задає, наскільки вигідним є перехід з однієї клітинки в іншу. Усі звичайні переходи мають значення -1, а переходи до цільової клітинки — винагороду +100. Далі створюється Q-матриця, яка поступово навчається: в кожному з 500 епізодів

агент випадково стартує, виконує дії, оновлює свої оцінки дій на основі винагород і майбутніх очікуваних вигід.

Для вибору дій використовується баланс між випадковим вибором (exploration) і використанням вже вивчених знань (exploitation). Найкращі дії оновлюють Q-матрицю, що допомагає агенту з часом знаходити ефективніші шляхи.

Після завершення навчання агент будує оптимальний шлях із початкової точки до мети, а результат візуалізується на сітці — крок за кроком показується, як агент рухається до цілі. Результат зображено на рисунку 10.8:

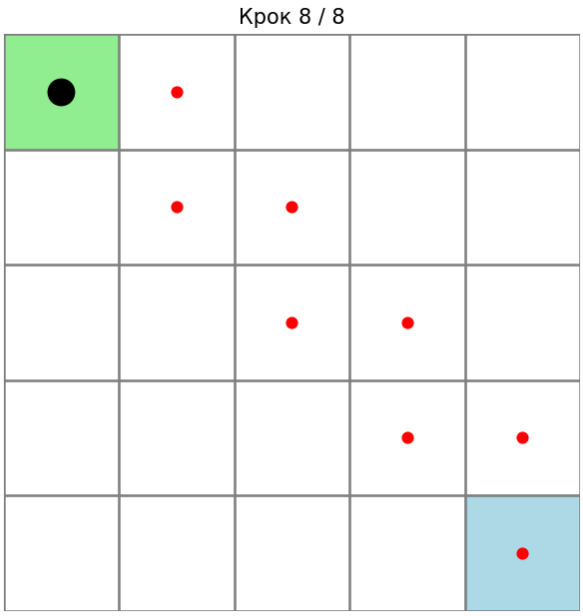

Рисунок 10.8 – Результат алгоритму

Висновок

У межах курсової роботи було комплексно досліджено та реалізовано кілька ключових підходів у галузі штучного інтелекту й машинного навчання. Основна увага була приділена практичній реалізації та моделюванню типових задач, що дозволило не лише закріпити теоретичні знання, а й набути цінного прикладного досвіду.

Перш за все, було реалізовано модель штучного нейрону для відтворення базових логічних функцій (AND, OR, NOT), а також проаналізовано обмеження одношарових мереж на прикладі задачі XOR. Це стало підґрунтям для розуміння важливості глибших архітектур нейронних мереж.

У другій частині роботи було реалізовано метод прогнозування часових рядів із використанням зворотного поширення помилки. Модель показала здатність навчатись на історичних даних і передбачати майбутні значення, що є основою для численних реальних застосувань — від фінансів до технічної діагностики.

Окрему роль у дослідженні відіграв генетичний алгоритм, який продемонстрував ефективність у задачах глобальної оптимізації. Було успішно знайдено екстремальні значення складної функції, що підтвердило здатність еволюційних методів обходити локальні мінімуми та максимуми.

Завершальним етапом стало моделювання поведінки інтелектуального агента з використанням алгоритму Q-learning. Агент навчився самостійно знаходити оптимальний шлях до цілі у дискретному середовищі, представленому у вигляді графа. Було показано, як завдяки навчанню з підкріпленням агент поступово покращує свою стратегію дій на основі накопиченого досвіду.

У результаті виконаної роботи сформовано цілісне уявлення про основні методи штучного інтелекту — від класичних нейронних моделей до адаптивного навчання в динамічних середовищах. Курсова робота стала важливим кроком у практичному освоєнні сучасних інтелектуальних технологій.

Список використаної літератури

1. Rashmin Mishra. AI Revolutionizing Healthcare: A Look at IBM Watson's Impact in the Healthcare Industry | *We are community* URL: https://wearecommunity.io/communities/healthcare/articles/5012
2. How does ChatGPT work? | *Zapier* URL: https://zapier.com/blog/how-does-chatgpt-work/
3. What is the neural network? | *AWS.Amazom* URL: https://aws.amazon.com/what-is/neural-network/
4. Dmitry Melanchenko. Часові ряди та що з ними робити. Гайд для початківців | *DOU* URL: https://dou.ua/forums/topic/40751/
5. Genetic Algorithm | *ScienceDirect* URL: https://www.sciencedirect.com/topics/engineering/genetic-algorithm
6. Функція активації виконується на вхідних чи вихідних даних шару? | *EITCA Academy* URL: https://surli.cc/yyoblu
7. Sejal Jaiswal. Multilayer Perceptrons in Machine Learning: A Comprehensive Guide | *DataCamp* URL: https://www.datacamp.com/tutorial/multilayer-perceptrons-in-machine-learning
8. Jonh McGonagle. Artificial Neural Network | *Brilliant* URL: https://www.datacamp.com/tutorial/multilayer-perceptrons-in-machine-learning
9. Supervised Learning | *itwiki* URL: https://itwiki.dev/data-science/ml-reference/ml-glossary/supervised-learning

ДОДАТОК 1

Моделювання формальних логічних функцій. Прогнозування часових рядів.

НТУУ «КПІ» ІАТЕ ІПЗЕ

Листів 9

Київ – 2025

Моделювання логічних функцій:

Task1.py:

```
import numpy as np


# Активаційна функція і її похідна
def sigmoid(x):
    return 1 / (1 + np.exp(-x))


def sigmoid_derivative(output):
    return output * (1 - output)


# Ініціалізація ваг і зсувів
def initialize_weights(input_size, hidden_size, output_size):
    np.random.seed(1)
    weights = {
        'input_hidden': np.random.rand(input_size, hidden_size),
        'hidden_output': np.random.rand(hidden_size, output_size)
    }
    biases = {
        'hidden': np.random.rand(hidden_size),
        'output': np.random.rand(output_size)
```

```
    }

    return weights, biases


# Пряме проходження

def forward_pass(inputs, weights, biases):

    hidden_input = np.dot(inputs, weights['input_hidden']) + biases['hidden']

    hidden_output = sigmoid(hidden_input)

    final_input = np.dot(hidden_output, weights['hidden_output']) + biases['output']

    final_output = sigmoid(final_input)

    return hidden_output, final_output


# Зворотне поширення

def backpropagation(inputs, expected_output, hidden_output, final_output, weights, biases, lr):

    output_error = expected_output - final_output

    output_delta = output_error * sigmoid_derivative(final_output)


    hidden_error = output_delta.dot(weights['hidden_output'].T)

    hidden_delta = hidden_error * sigmoid_derivative(hidden_output)


    weights['hidden_output'] += lr * np.dot(hidden_output.T, output_delta)

    biases['output'] += lr * np.sum(output_delta, axis=0)
```

```
    weights['input_hidden'] += lr * np.dot(inputs.T, hidden_delta)
    biases['hidden'] += lr * np.sum(hidden_delta, axis=0)


# Тренування
def train_network(inputs, expected_output, epochs=10000, learning_rate=0.1):
    input_size = inputs.shape[1]
    hidden_size = 3
    output_size = 1
    weights, biases = initialize_weights(input_size, hidden_size, output_size)

    for _ in range(epochs):
        hidden_output, final_output = forward_pass(inputs, weights, biases)
        backpropagation(inputs, expected_output, hidden_output, final_output, weights, biases, learning_rate)

    return weights, biases

# Прогноз
def predict(input_data, weights, biases):
    _, final_output = forward_pass(input_data, weights, biases)
```

```
    return final_output


# Вивід результатів у зручному форматі
def print_predictions(func_name, inputs, expected_output, weights, biases):
    print(f"\n=== Результати для логічної функції: {func_name} ===")
    print(f"{'Вхідні дані':<15}{'Очікувано':<12}{'Результат'}")
    print("-" * 40)
    for i, input_vector in enumerate(inputs):
        output = predict(np.array([input_vector]), weights, biases)
        predicted_value = round(output[0][0])
        print(f"{str(input_vector):<15}{expected_output[i][0]:<12}{predicted_value}")
    print()


# Основна функція
def test_logical_functions():
    inputs = np.array([
        [0, 0],
        [0, 1],
        [1, 0],
        [1, 1]
    ])
```

```
    logic_outputs = {
        'AND': np.array([[0], [0], [0], [1]]),
        'OR': np.array([[0], [1], [1], [1]]),
        'NAND': np.array([[1], [1], [1], [0]]),
        'XOR': np.array([[0], [1], [1], [0]])
    }

    for name, expected in logic_outputs.items():
        weights, biases = train_network(inputs, expected)
        print_predictions(name, inputs, expected, weights, biases)

test_logical_functions()
```

Прогнозування часових рядів:

main.py:

```
import random
import math
import matplotlib.pyplot as plt

# Сигмоїдальна функція активації
def sigmoid(x):
```

```
    return 10 / (1 + math.exp(-x))


# Похідна сигмоїдальної функції
def der_sigmoid(x):
    return (math.exp(-x) / (1 + math.exp(-x)) ** 2) * 10


# Тренування
def train_neuron(data, learning_rate=0.01, max_epochs=10_000, error_threshold=0.0001):
    # Випадкова генерація синаптичних ваг
    weights = [random.uniform(-1, 1) for _ in range(3)]


    # Кількість навчальних прикладів
    n = len(data) - 3
    previous_total_error = float('inf')
    errors = []
    # цикл тренування
    for epoch in range(max_epochs):


        total_error = 0
        deltas = [0, 0, 0]
```

```
for i in range(n):


    x = [data[i], data[i+1], data[i+2]]


    y = data[i+3]

    # Зважувальна сума(вхід нейрона)

    w_sum = sum(x[j] * weights[j] for j in range(3))


    # Прогнозоване значеннння(сигмоїдальна функція активації)

    y_pred = sigmoid(w_sum)


    # Квадратична помилка

    error = (y_pred - y) ** 2

    total_error += error


    # Похідна помилки для зворотного поширення (градієнт)

    delta = (y_pred - y) * der_sigmoid(w_sum)


    # Дельти для кожної ваги

    for j in range(3):

        deltas[j] += -learning_rate * delta * x[j]
```

```
    # Оновлені ваги
    for k in range(3):
        weights[k] += deltas[k] / n

    errors.append(total_error)

    # Якщо зміна помилки менше 0.01 * кікість навчальних наборів
    if abs(previous_total_error - total_error) < error_threshold * n:
        break

    previous_total_error = total_error

plt.plot(range(0, len(errors)), errors, label='Сумарна помилка')
plt.xlabel('Кількість епох')
plt.ylabel('Помилка')
plt.title('Динаміка помилки під час тренування')
plt.legend()
plt.show()
# Навчені вагові коефіцієнти
return weights
```

```
# Функція прогнозу

def predict_next(data, weights):

    x1, x2, x3 = data[-3], data[-2], data[-1]

    s = x1 * weights[0] + x2 * weights[1] + x3 * weights[2]

    return sigmoid(s)


# 1 - 13 дані для тренування, 14 і 15 - тестові

data = [0.58, 3.38, 0.91, 5.80, 0.91, 5.01, 1.17, 4.67, 0.60, 4.81, 0.53, 4.75, 1.01, 5.04, 1.07]


# Тренування нейрону, поверне ваги

weights = train_neuron(data[:13])


# Прогнозуєм

predicted_x14 = predict_next(data[:13], weights)

predicted_x15 = predict_next(data[1:14], weights)


print(f"Прогнозуємо x14: {predicted_x14}")

print(f"Дійсне x14: {data[13]}")

print(f"Прогнозуємо x15: {predicted_x15}")

print(f"Дійсне x15: {data[14]}")
```

```
# Обчислюємо похибку

print(f"Похибка на тестових даних: {round((abs(predicted_x14 - data[13])/data[13] + abs(predicted_x15 - data[14])/data[14])*50, 2)}%")
```

ДОДАТОК 2

Використання навчання з вчителем для задач розпізнавання образів, класифікації об’єктів..

НТУУ «КПІ» ІАТЕ ІПЗЕ

Листів 13

Київ – 2025

train.csv:

001100

010010

100001

100001

010010

001100

000

000000

011110

011110

011110

011110

000000

100

001000

011100

111110

111110

011100

001000

010

000000

011110

100001

100001

011110

000000

001

100000

110000

111000

111100

111110

111111

111

test.csv:

001110

010011

100001

100001

000010

001100

000

000000

011100

011110

010110

011110

000100

100

001000

011000

101110

111110

010100

001000

010

000000

011111

100001

110001

011110

000000

001

100000

111000

111000

111100

110110

111010

111

Pw2.py:

```
import csv

import math
```

```
import random

import matplotlib.pyplot as plt


filepath = "train.csv"


def get_data(filepath):
    X = []
    y = []
    with open(filepath) as f:
        for i, row in enumerate(csv.reader(f)):
            goal_index = i // 7
            row = row[0]
            if len(row) == 6:
                if len(X) <= goal_index:
                    X.append([])
                X[goal_index].extend(map(int, row))
            elif len(row) == 3:
                if len(y) <= goal_index:
                    y.append([])
```

```
            y[goal_index].extend(map(int, row))
    return X, y


import math
import random


class NeuralNetwork:
        def __init__(self, layer_sizes, learning_rate=0.1, activation="sigmoid", loss="mse"):
        self.layer_sizes = layer_sizes
        self.learning_rate = learning_rate
        self.activation = activation
        self.loss = loss


        # Ініціалізація ваг і зсувів
        self.weights = [
            [
                [random.uniform(-1, 1) for _ in range(layer_sizes[l])]
                for _ in range(layer_sizes[l + 1])
            ] for l in range(len(layer_sizes) - 1)
```

```
        ]
        self.biases = [
            [random.uniform(-1, 1) for _ in range(layer_sizes[l + 1])]
            for l in range(len(layer_sizes) - 1)
        ]

        # Збереження виходів
        self.outputs = [[0.0 for _ in range(size)] for size in layer_sizes]

        # Мапа для виводу результатів
        self.results = {
            (0, 0 ,0): "коло", (1, 0, 0): "квадрат", (0, 1, 0): "ромб",
            (0, 0, 1): "еліпс", (1, 1, 1): "трикутник"
        }

    def activate(self, x):
        if self.activation == "sigmoid":
            return 1 / (1 + math.exp(-x))
        elif self.activation == "relu":
```

```
        return max(0, x)
    elif self.activation == "tanh":
        return math.tanh(x)
    return x

def activate_derivative(self, activated_x):
    if self.activation == "sigmoid":
        return activated_x * (1 - activated_x)
    elif self.activation == "relu":
        return 1 if activated_x > 0 else 0
    elif self.activation == "tanh":
        return 1 - activated_x ** 2
    return 1

def forward(self, x):
    self.outputs[0] = x[:]
    for l in range(1, len(self.layer_sizes)):
        for j in range(self.layer_sizes[l]):
            weighted_sum = sum(
```

```
                self.weights[l - 1][j][k] * self.outputs[l - 1][k]
                for k in range(self.layer_sizes[l - 1])
            ) + self.biases[l - 1][j]
            self.outputs[l][j] = self.activate(weighted_sum)
    return self.outputs


def mse_loss(self, y_true, y_pred):
    return sum((yt - yp) ** 2 for yt, yp in zip(y_true, y_pred)) / len(y_true)


def mae_loss(self, y_true, y_pred):
    return sum(abs(yt - yp) for yt, yp in zip(y_true, y_pred)) / len(y_true)


def backward(self, y_true):
    num_layers = len(self.layer_sizes)
    local_gradients = [[0.0 for _ in range(size)] for size in self.layer_sizes[1:]]

    # Градієнти вихідного шару
    for i in range(self.layer_sizes[-1]):
        output = self.outputs[-1][i]
```

```
            error = y_true[i] - output
            local_gradients[-1][i] = error * self.activate_derivative(output)

        # Зворотне поширення градієнтів
        for l in reversed(range(len(self.layer_sizes) - 1)):
            for i in range(self.layer_sizes[l + 1]):
                for j in range(self.layer_sizes[l]):
                    delta = self.learning_rate * local_gradients[l][i] * self.outputs[l][j]
                    self.weights[l][i][j] += delta
                self.biases[l][i] += self.learning_rate * local_gradients[l][i]

            if l > 0:
                for j in range(self.layer_sizes[l]):
                    gradient_sum = sum(
                        self.weights[l][i][j] * local_gradients[l][i]
                        for i in range(self.layer_sizes[l + 1])
                    )
                    local_gradients[l - 1][j] = gradient_sum * self.activate_derivative(self.outputs[l][j])
```

```
    # Повернення втрати
    if self.loss == 'mse':
        return self.mse_loss(y_true, self.outputs[-1])
    elif self.loss == 'mae':
        return self.mae_loss(y_true, self.outputs[-1])
    return 0


def train(self, X_train, y_train, epochs=1000):
    for epoch in range(epochs):
        total_loss = 0
        for x, y in zip(X_train, y_train):
            self.forward(x)
            loss = self.backward(y)
            total_loss += loss
        if epoch % 100 == 0:
            print(f"Епоха {epoch}, середня втрата: {total_loss / len(X_train)}")


def evaluate(self, X_test, y_test):
    correct = 0
```

```
    for x, y in zip(X_test, y_test):
        raw_pred = self.forward(x)[-1]  # беремо тільки вихідний шар
        pred = [1 if p >= 0.5 else 0 for p in raw_pred]
        pred_fig = self.results.get(tuple(pred), '?')
        true_fig = self.results.get(tuple(y), '?')
        if pred_fig == true_fig:
            correct += 1
    accuracy = correct / len(X_test)
    print(f"Точність: {accuracy * 100}%")


def visualize_predictions(self, X_test, y_test, num_samples=5):
    fig, axes = plt.subplots(1, num_samples, figsize=(15, 3))
    for i in range(num_samples):
        x, y = X_test[i], y_test[i]
        raw_pred = self.forward(x)[-1]  # беремо лише вихід нейромережі
        pred = [1 if p >= 0.5 else 0 for p in raw_pred]  # округлення до 0 або 1
        # Перетворення списку x у 2D-масив 6x6
        image = [x[j*6:(j+1)*6] for j in range(6)]
```

```
            axes[i].imshow(image, cmap="Blues")

                axes[i].set_title(f"Прогноз: {self.results.get(tuple(pred), '?')}\nПравильна: {self.results.get(tuple(y), '?')}")

            axes[i].axis("off")

        plt.show()



X_train, y_train = get_data("train.csv")

X_test, y_test = get_data("test.csv")


nn = NeuralNetwork(layer_sizes=[36, 8, 3], learning_rate=0.01, activation="sigmoid", loss="mse")

nn.train(X_train, y_train, epochs=1000)

nn.evaluate(X_test, y_test)

nn.visualize_predictions(X_test, y_test, num_samples=5)
```

ДОДАТОК 3

Застосуванням генетичних алгоритмів для оптимізації функції.

НТУУ «КПІ» ІАТЕ ІПЗЕ

Листів 10

Київ – 2025

Pw3.py:

```
import random

import math

import matplotlib.pyplot as plt


# Кодування дійсного числа у бінарний рядок заданої довжини

def encode(x, a, b, n_bits):

    max_int = 2**n_bits - 1

    ratio = (x - a) / (b - a)

    int_val = int(ratio * max_int)

    return format(int_val, f'0{n_bits}b')


# Декодування бінарного рядка у дійсне число з врахуванням меж [a, b]

def decode(chromosome, a, b):

    max_int = 2**len(chromosome) - 1

    int_val = int(chromosome, 2)

    return a + (b - a) * int_val / max_int


# Оцінка пристосованості (fitness): значення функції або його мінус (для мінімізації)
```

```
def fitness(chromosome, func, a, b, optimize_for):
    x = decode(chromosome, a, b)
    val = func(x)
    return val if optimize_for == 'max' else -val


# Мутація хромосоми: з ймовірністю mutation_rate змінюємо кожен біт
def mutate(chromosome, mutation_rate):
    result = []
    for bit in chromosome:
        if random.random() < mutation_rate:
            result.append('1' if bit == '0' else '0')  # інвертуємо біт
        else:
            result.append(bit)
    return ''.join(result)


# Одноточковий кросовер: обмінюємо частини хромосом
def crossover(p1, p2):
    point = random.randint(1, len(p1)-1)
    return p1[:point] + p2[point:], p2[:point] + p1[point:]
```

```
# Обчислення справжнього екстремуму функції перебором з малим кроком
def calc_true_extreme(func, a, b, precision, optimize_for):
    xs = []
    x = a
    step = precision / 10
    while x < b:
        xs.append(x)
        x += step
    ys = [func(x) for x in xs]
    if optimize_for == 'max':
        idx = max(range(len(ys)), key=ys.__getitem__)
    else:
        idx = min(range(len(ys)), key=ys.__getitem__)
    return xs[idx], ys[idx]


# Основна функція генетичного алгоритму
def genetic_algorithm(func, a, b, precision, pop_size=50, epochs=100,
                      mutation_rate=0.01, optimize_for='max'):
```

```
# Визначення кількості біт для кодування
n_bits = math.ceil(math.log2((b - a) / precision))

# Початкова популяція з випадкових особин
population = [encode(random.uniform(a, b), a, b, n_bits) for _ in range(pop_size)]

# Справжнє значення максимуму або мінімуму
true_x, true_y = calc_true_extreme(func, a, b, precision, optimize_for)
error_history = []

# Основний цикл еволюції
for _ in range(epochs):
    # Оцінка популяції
    scored = [(ind, fitness(ind, func, a, b, optimize_for)) for ind in population]
    scored.sort(key=lambda x: x[1], reverse=True)

    # Збереження поточної похибки (квадрат різниці від істинного значення)
    best_fitness = scored[0][1]
```

```
error = ((best_fitness if optimize_for=='max' else -best_fitness) - true_y )**2
error_history.append(error)


# Турнірна селекція (вибір кращого з двох випадкових)
def tournament():
    i1, i2 = random.sample(range(pop_size), 2)
        return population[i1] if fitness(population[i1], func, a, b, optimize_for) > fitness(population[i2], func, a, b, optimize_for) else population[i2]


new_population = []
# Створення нової популяції шляхом схрещування і мутації
while len(new_population) < pop_size:
    p1 = tournament()
    p2 = tournament()
    c1, c2 = crossover(p1, p2)
    c1 = mutate(c1, mutation_rate)
    c2 = mutate(c2, mutation_rate)
    new_population.extend([c1, c2])


population = new_population[:pop_size]
```

```
    # Пошук найкращого результату в фінальній популяції
      best_chromosome = max(population, key=lambda ch: fitness(ch, func, a, b, optimize_for))
    best_x = decode(best_chromosome, a, b)
    best_y = func(best_x)

    return best_x, best_y, error_history, (true_x, true_y)

# Побудова графіку функції з відмітками знайдених максимуму і мінімуму
def plot_function_with_extrema(func, a, b, precision, max_res, min_res):
    x_vals = []
    x = a
    step = precision
    while x < b:
        x_vals.append(x)
        x += step
    y_vals = [func(x) for x in x_vals]

    plt.figure(figsize=(10, 6))
```

```
plt.plot(x_vals, y_vals, label='f(x)', linewidth=2)


plt.scatter(max_res[0], max_res[1], color='red', s=80, label='Max', zorder=5)
plt.annotate(f'Max\nx={max_res[0]:.3f}\ny={max_res[1]:.3f}',
        (max_res[0], max_res[1]), textcoords="offset points", xytext=(0, 10),
        ha='center', color='red')


plt.scatter(min_res[0], min_res[1], color='blue', s=80, label='Min', zorder=5)
plt.annotate(f'Min\nx={min_res[0]:.3f}\ny={min_res[1]:.3f}',
        (min_res[0], min_res[1]), textcoords="offset points", xytext=(0, -25),
        ha='center', color='blue')


plt.title("Функція зі знайденим мінімумом та максимумом")
plt.xlabel("x")
plt.ylabel("f(x)")
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()
```

```
# Побудова графіку історії похибки (квадрат помилки на кожній ітерації)
def plot_error_history(error_history, label):
    plt.figure(figsize=(8, 5))
    plt.plot(error_history, label=label, color='green')
    plt.xlabel("Epoch")
    plt.ylabel("Squared Error")
    plt.title(f"Convergence to True Optimum ({label})")
    plt.grid(True)
    plt.legend()
    plt.tight_layout()
    plt.show()


# Цільова функція для оптимізації
def target_function(x):
    return math.cos(x**2) / x


# Параметри
precision = 0.001
```

```
a, b = precision, 5

pop_size = 30

epochs = 50

mutation_rate = 0.02


# Запуск алгоритму для максимуму

max_x, max_y, max_error_hist, true_max = genetic_algorithm(target_function, a, b, precision,

                                                           pop_size=pop_size,

                                                           epochs=epochs,

                                                           mutation_rate=mutation_rate,

                                                           optimize_for='max')


# Запуск алгоритму для мінімуму

min_x, min_y, min_error_hist, true_min = genetic_algorithm(target_function, a, b, precision,

                                                           pop_size=pop_size,

                                                           epochs=epochs,

                                                           mutation_rate=mutation_rate,

                                                           optimize_for='min')
```

```
# Виведення результатів
print(f"Max: x={max_x:.4f}, f(x)={max_y}")
print(f"Min: x={min_x:.4f}, f(x)={min_y}")

# Побудова графіків похибок
plot_error_history(max_error_hist, "Maximization")
plot_error_history(min_error_hist, "Minimization")

# Обчислення середньої квадратичної похибки
final_mean_error = (max_error_hist[-1] + min_error_hist[-1]) / 2
print(f"Фінальна середньо квадратична помилка: {final_mean_error:.20f}")

# Побудова функції з позначенням максимуму і мінімуму
plot_function_with_extrema(target_function, a, b, precision,
                           max_res=(max_x, max_y),
                           min_res=(min_x, min_y))
```

ДОДАТОК 4

Інтелектуальні агенти. Створення алгоритму Q-навчання.

НТУУ «КПІ» ІАТЕ ІПЗЕ

Листів 8

Київ – 2025

Pw4.py:

```
import numpy as np

import matplotlib.pyplot as plt


class QLearningAgent:

    def __init__(self, dim=5, goal=0, start=24, alfa=0.1, gamma=0.8, epsilon=0.2, episodes=500):

        self.dim = dim           # Розмір сітки (dim x dim)

        self.goal = goal         # Цільова клітинка (індекс)

        self.start = start       # Початкова клітинка (індекс)

        self.alfa = alfa         # Швидкість навчання (learning rate)

        self.gamma = gamma       # Коефіцієнт дисконтної винагороди

        self.epsilon = epsilon   # Імовірність вибору випадкової дії (exploration)

        self.episodes = episodes # Кількість епізодів тренування


        # Побудова матриці винагород (R) та ініціалізація Q-матриці

        self.R = self._build_R()

        self.Q = np.zeros_like(self.R, dtype=float)  # Матриця оцінок Q, спочатку нулі


    def _state2coord(self, s):
```

```
    return divmod(s, self.dim)


def _coord2state(self, i, j):
    return i * self.dim + j


def _available_actions(self, state):
    # Повертає список допустимих переходів (дій) із заданого стану
    i, j = self._state2coord(state)
    actions = []
    # Перевіряємо 4 напрямки: вгору, вниз, вліво, вправо
    for di, dj in [(-1, 0), (1, 0), (0, -1), (0, 1)]:
        ni, nj = i + di, j + dj
        # Якщо координати в межах сітки, додаємо дію
        if 0 <= ni < self.dim and 0 <= nj < self.dim:
            actions.append(self._coord2state(ni, nj))
    return actions


def _build_R(self):
    # Створення матриці винагород R
```

```
    size = self.dim * self.dim
    R = np.full((size, size), -1)  # Спочатку всі переходи -1 (негативна винагорода)
    for s in range(size):
        for a in self._available_actions(s):
            R[s, a] = -1              # Дозволені переходи отримують -1 (рух)
    # Встановлюємо позитивну винагороду за перехід у цільову клітинку
    for a in self._available_actions(self.goal):
        R[a, self.goal] = 100
    return R


def train(self):
    for _ in range(self.episodes):
        # Випадковий початковий стан
        state = np.random.randint(0, self.dim * self.dim)
        while state == self.goal:
            state = np.random.randint(0, self.dim * self.dim)


        # Поки не досягнемо цілі, робимо кроки
        while state != self.goal:
```

```
actions = self._available_actions(state)  # Доступні дії


# Вибір дії: випадкова з ймовірністю epsilon (exploration)
if np.random.rand() < self.epsilon:
    next_state = np.random.choice(actions)
else:
    # Вибір кращої дії згідно з Q (exploitation)
    qs = [self.Q[state, a] for a in actions]
    max_q = max(qs)
    best = [a for a, q in zip(actions, qs) if q == max_q]
    next_state = np.random.choice(best)


# Отримуємо винагороду за перехід
reward = self.R[state, next_state]
old_q = self.Q[state, next_state]


# Оновлення Q-значення за формулою
self.Q[state, next_state] = old_q + self.alfa * (
    reward + self.gamma * np.max(self.Q[next_state]) - old_q)
```

```
            # Переходимо до наступного стану
            state = next_state

def get_optimal_path(self):
    # Відновлення оптимального шляху із Q-матриці
    state = self.start
    path = [state]
    visited = set()

    while state != self.goal:
        actions = self._available_actions(state)
        qs = [self.Q[state, a] for a in actions]
        max_q = max(qs)
        best = [a for a, q in zip(actions, qs) if q == max_q]
        next_state = np.random.choice(best)

        path.append(next_state)
```

```
        if next_state in visited:
            break
        visited.add(next_state)

        state = next_state

        if len(path) > 100:
            break
    return path

def visualize_path(self, path):
    fig, ax = plt.subplots(figsize=(6, 6))

    for idx in range(len(path)):
        ax.clear()

        for i in range(self.dim + 1):
            ax.plot([-0.5, self.dim - 0.5], [i - 0.5, i - 0.5], color='gray')
            ax.plot([i - 0.5, i - 0.5], [-0.5, self.dim - 0.5], color='gray')
```

```
ax.set_xlim(-0.5, self.dim - 0.5)

ax.set_ylim(self.dim - 0.5, -0.5)

ax.set_aspect('equal')

ax.axis('off')

gi, gj = self._state2coord(self.goal)

ax.add_patch(plt.Rectangle((gj - 0.5, gi - 0.5), 1, 1, color='lightgreen'))



si, sj = self._state2coord(self.start)

ax.add_patch(plt.Rectangle((sj - 0.5, si - 0.5), 1, 1, color='lightblue'))



for s in path[:idx + 1]:

    r, c = self._state2coord(s)

    ax.plot(c, r, 'o', color='red')



ci, cj = self._state2coord(path[idx])

ax.plot(cj, ci, 'o', color='black', markersize=15)



ax.set_title(f"Крок {idx} / {len(path) - 1}")
```

```
            plt.pause(0.4)


        plt.ioff()
        plt.show()


agent = QLearningAgent()
agent.train()
optimal_path = agent.get_optimal_path()
agent.visualize_path(optimal_path)
```