ПОЛТАВСЬКИЙ ДЕРЖАВНИЙ АГРАРНИЙ УНІВЕРСИТЕТ
Навчально-науковий інститут економіки, управління, права та інформаційних технологій
Кафедра інформаційних систем та технологій

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття ступеня вищої освіти магістр

на тему: **«Моделювання поліноміальних коефіцієнтів із використанням штучних нейронних мереж»**

Виконав здобувач вищої освіти денної форми навчання

Дзеркалій Ігор Володимирович

Виконав: здобувач вищої освіти
за освітньою програмою
Інформаційні управляючі системи та технології
спеціальності 126 Інформаційні системи та технології
ступеня вищої освіти магістр
групи 126ІСТ_мд_2023
Дзеркалій Ігор Володимирович
Керівник: Одарущенко Олег Миколайович
Рецензент: Муравльов Володимир Вячеславович

Полтава – 2024 року

ВСТУП

Нейронні мережі представляють собою потужний напрямок в практиці створення сучасних технічних систем. Теорія нейронних мереж виникла на основі досліджень в області штучного інтелекту зі здатності нервових біологічних систем навчатися аналогічно низькорівневим структурам мозку. Саме тому необхідно використовувати знання з даної галузі в розвитку математичних досліджень.

Штучні нейронні мережі здатні адаптувати свою поведінку залежно від умов зовнішнього середовища. Ця особливість є однією з основних причин, чому вони викликають такий значний інтерес. Після отримання вхідних сигналів (можливо, разом із бажаними виходами) вони самостійно налаштовуються, щоб забезпечити потрібну реакцію. Після завершення навчання мережа може демонструвати стійкість до невеликих змін у вхідних даних. Така властивість, яка дозволяє розпізнавати образи навіть у присутності шуму та спотворень, має критичне значення для вирішення задач розпізнавання в реальному світі. Завдяки цьому нейронні мережі долають обмеження, пов'язані з необхідністю точної відповідності даних, характерних для традиційних комп'ютерів, і відкривають можливості для створення систем, здатних працювати в умовах реального, недосконалого світу. Важливо підкреслити, що узагальнення в нейронних мережах виникає автоматично завдяки їхній структурі, а не за рахунок залучення "людського інтелекту" через спеціально створені програми.

У даній кваліфікаційній роботі розглядається задача відновлення коефіцієнтів полінома за допомогою штучної нейронної мережі.

Актуальність теми дослідження полягає в тому, що вирішення практичних задач за допомогою штучних нейронних мереж є ще не достатньо вивченою у зв'язку з великою кількістю задач, розв'язок яких можна знайти з їх допомогою, а також через те що інші алгоритми пошуку коефіцієнтів полінома не є достатньо ефективними та потребують досить багато часу. Тому

необхідно змоделювати таку систему, яка б відновлювала коефіцієнти полінома по заданому набору табличних значень в скінченній кількості точок. Такі задачі є типовими в області механіки і процесів управління, при аналізі та синтезі електронних ланцюгів і систем, при відновленні зображень та моделюванні теплових процесів, які відбуваються в промислових печах і біохімічних процесів в клітинах живих організмів тощо. Тобто область застосування таких задач є досить обширною, що ще раз доводить актуальність даної теми.

Зв'язок роботи з науковими програмами, темами. Робота відповідає дослідженням в межах науково-дослідної роботи «Перспективи розвитку підприємництва: управлінські, маркетингові, інформаційні та правові аспекти» відповідно до договору №19 від 06.06.2024 р. між ТОВ «ПАФ Гарант» та Полтавським державним аграрним університетом (розділ «Забезпечення гарантоздатності (надійності та функційної безпечності) компонентів інформаційної системи аграрного підприємства»).

Мета досліджень є моделювання, навчання та застосування нейронної мережі для відновлення коефіцієнтів полінома.

Завданнями кваліфікаційної роботи є:

– вивчення архітектури штучної нейронної мережі під назвою багатошаровий персептрон;

– ознайомлення з методом зворотного розповсюдження помилки для навчання багатошарового персептрона;

– вивчення методу моделювання нейронної мережі в системі Matlab;

– створення програми мовою програмування C#, яка б моделювала та навчала нейронну мережу для виконання поставленої задачі.

Об'єктом дослідження є – поліноміальні коефіцієнти та процеси їх відновлення й моделювання.

Предмет дослідження – методи використання штучних нейронних мереж для моделювання та відновлення коефіцієнтів поліномів на основі заданих даних.

Методи досліджень – в ході досліджень було застосовано методи:

– теоретичного аналізу, а саме: вивчення властивостей поліномів і методів їхнього моделювання; аналізу основних принципів роботи штучних нейронних мереж (ШНМ); дослідження існуючих підходів до навчання ШНМ;

– математичного моделювання, а саме: побудови математичної моделі для зв'язку між вхідними даними та поліноміальними коефіцієнтами; формалізації задачі відновлення коефіцієнтів поліномів як задачі регресії;

– машинного навчання, а саме: створення та навчання нейронних мереж для апроксимації функції, що описує поліном; використання алгоритмів оптимізації для підбору вагових коефіцієнтів ШНМ (метод градієнтного спуску); підбір гіперпараметрів моделі (кількість шарів, кількість нейронів, функції активації);

– емпіричного моделювання: збору та підготовки синтетичних і реальних наборів даних для навчання моделі;

– перевірки якості моделі на тестових наборах даних.

Інформаційна база кваліфікаційної роботи сформована з наукових фахових статей, наукових монографій, аналітичних звітів державних та міжнародних експертних груп щодо експлуатації критичних технічних систем, виконаних науково-дослідних та дисертаційних робот заданою тематикою.

Елементи наукової новизни роботи полягають у використанні штучних нейронних мереж для відновлення коефіцієнтів поліномів із врахуванням специфіки вхідних даних і задачі; побудові архітектури ШНМ, оптимізованої для задачі апроксимації коефіцієнтів поліномів.

Практична значущість роботи полягає в розроблені програмної реалізації для відновлення коефіцієнтів поліномів.

Апробація результатів дослідження відбувалася шляхом оприлюднення доповіді на студентській конференції.

Публікації. За результатами проведеного дослідження опубліковані тези: «Моделювання штучної нейронної мережі типу багатошаровий персептрон в системі MATLAB». Матеріали XXI щорічного міждисциплінарного семінару

«Студентські роботи за науковою тематикою кафедри інформаційних систем та технологій ННІ ЕУПІТ ПДАУ», Полтава, 19 листопада 2024 р., С.22-24.

Структура та обсяг роботи. Робота складається з вступу, трьох розділів, висновків, списку використаних джерел та додатків. Загальний обсяг роботи становить 61 сторінку; робота містить 10 рисунків; список використаних джерел, що включає 52 найменування.

РОЗДІЛ 1
АНАЛІЗ МАТЕМАТИЧНИХ МОДЕЛЕЙ ПОБУДОВИ ШТУЧНИХ НЕЙРОМЕРЕЖ

1.1 Нейронні мережі та їх переваги

Розуміння принципів роботи нейрона та структури його зв'язків дозволило дослідникам розробити математичні моделі для перевірки своїх теорій. Уже в перших дослідженнях стало зрозуміло, що ці моделі не лише відтворюють функції мозку, а й здатні виконувати завдання, які мають самостійну цінність. У зв'язку з цим сформувалися дві основні цілі нейронного моделювання, які залишаються актуальними і взаємодоповнюючими:

– перша – дослідити функціонування нервової системи людини на рівні фізіології та психології;

– друга – розробити обчислювальні системи (штучні нейронні мережі), здатні виконувати функції, подібні до роботи мозку.

Перший крок у розвитку штучних нейронних мереж був зроблений у 1943 році, коли нейрофізіолог Уоррен Маккалох (Warren McCulloch) та математик Волтер Піттс (Walter Pitts) опублікували статтю, присвячену роботі штучних нейронів та моделюванню нейронної мережі за допомогою електричних схем.

Одночасно з прогресом у нейроанатомії та нейрофізіології психологи створювали моделі, що описували процеси навчання людини. Однією з найбільш значущих стала модель Дональда Гебба (Donald Hebb). У 1949 році у своїй книзі *«Організація поведінки»* він запропонував закон навчання, який став відправною точкою для розробки алгоритмів навчання штучних нейронних мереж. Цей процес навчання нейромереж також відомий як нейроеволюція. У 1950-1960-х роках група дослідників, поєднавши біологічні та фізіологічні підходи, створила перші штучні нейронні мережі. У 1950-х з'явилися програмні моделі таких мереж. Піонером у цій галузі став Натаніел

Рочестер (Nathanial Rochester) з дослідницької лабораторії IBM. Однак його модель не досягла успіху через стрімкий розвиток традиційних обчислювальних технологій, який відсунув нейронні дослідження на другий план.

Попри це, перші успіхи викликали хвилю активності та оптимізму. Мінскі, Розенблат, Відроу та інші розробили одношарові нейронні мережі, відомі як персептрони. Вони використовувалися для вирішення широкого спектра завдань, таких як прогнозування погоди, аналіз електрокардіограм та створення систем штучного зору. У 1956 році Дартмутський дослідницький проєкт з штучного інтелекту дав поштовх розвитку як штучного інтелекту загалом, так і нейронних мереж зокрема. Це сприяло формуванню двох ключових напрямів у дослідженнях штучного інтелекту:

– створення промислових застосувань (наприклад, експертних систем);

– моделювання роботи мозку.

У 1958 році Джон фон Нейман (John von Neumann) запропонував використання вакуумних трубок для імітації простих функцій нейронів. У 1959 році Бернард Відроу (Bernard Widrow) і Марсіан Гофф (Marcian Hoff) створили моделі ADALINE і MADALINE (Multiple ADAptive LINear Elements). Модель MADALINE діяла як адаптивний фільтр, здатний усувати відлуння на телефонних лініях, і досі знаходить комерційне застосування. Тим часом нейробіолог Френк Розенблатт (Frank Rosenblatt) розпочав роботу над персептроном, який став класичною нейронною мережею. Одношаровий персептрон, реалізований апаратно, використовувався для класифікації вхідних сигналів, розподіляючи їх між двома класами. Деякий час здавалося, що розгадка інтелекту знайдена, і відтворення функцій людського мозку залежить лише від побудови достатньо великої мережі. Однак це уявлення швидко зникло, оскільки мережі не могли вирішувати завдання, подібні до тих, які вони вже успішно виконували. Ці обмеження призвели до періоду інтенсивного аналізу. Марвін Мінскі (Marvin Minsky) разом із Сеймуром Папертом (Seymour Papert) у книзі *«Персептрони»* (1969) довели

математично, що одношарові нейронні мережі, поширені на той час, не здатні вирішувати певні базові задачі, зокрема реалізувати функцію «виключне або» (XOR). Аргументи Мінскі, підкріплені його авторитетом, викликали значну довіру до висновків книги. Результатом цього стала хвиля розчарування серед дослідників, які залишили дослідження нейронних мереж на користь інших перспективних напрямів. Уряди також перерозподілили фінансування, і штучні нейронні мережі майже на два десятиліття опинилися в забутті.

Проте декілька найбільш наполегливих вчених, таких як Тейво Кохонен, Стів Гросберг, Джеймс Андерсон продовжили дослідження. Поступово з'явився теоретичний фундамент, на основі якого сьогодні конструюються найпотужніші багатошарові мережі. Оцінка Мінскі виявилася надто песимістичною, багато з поставлених в його книзі задач розв'язуються зараз мережами за допомогою стандартних процедур.

Наприкінці 1980-х років теорія стала застосовуватися в прикладних областях і з'явилися нові корпорації, що займалися комерційним використанням цієї технології. Наростання наукової активності носило вибуховий характер. У 1987 р. було проведено чотири великих наради зі штучних нейронних мереж і опубліковано понад 500 наукових статей.

Після майже двох десятиліть забуття інтерес до штучних нейронних мереж різко зріс. Фахівці з різних галузей, таких як інженерія, філософія, фізіологія та психологія, зацікавлені потенціалом цієї технології, почали досліджувати її застосування у своїх дисциплінах. Відродження інтересу було зумовлене як теоретичними, так і практичними досягненнями. Неочікувано відкрилися нові можливості використання обчислювальних систем у сферах, які раніше вважалися виключною сферою людського інтелекту. З'явилися перспективи створення машин, здатних навчатися і запам’ятовувати, що нагадує процеси людського мислення, і наповнення терміну «штучний інтелект» новим, глибоким змістом [1-10].

Нервова система людини є одним із найскладніших механізмів, функціонування якого базується на взаємодії 10^{13} нейронів, які об’єднані за

допомогою 10^{17} синаптичних зв'язків. Кожен нейрон може приймати, обробляти та передавати електрохімічні сигнали по нервовим шляхам. Інтенсивність сигналів, які отримуються нейроном, залежить від активності його ліній зв'язку з іншими нейронами.

Біологічний нейрон складається з трьох базових компонентів: дендритів, соми (тіла клітини) та аксонів. Місцем контакту нервових волокон являються синопсиси, які передають збудження від клітини до клітини.

Дендрити йдуть від тіла нервової клітини до інших нейронів, де вони приймають сигнали в точках з'єднання (синопсисах). Прийняті синапсом вхідні сигнали підводяться до тіла нейрону. Тут вони сумуються, причому одні входи намагаються збудити нейрон, інші – перешкодити його збудженню.

Коли сумарне збудження в тілі нейрона перевищує деякий поріг, нейрон збуджується, посилаючи по аксону сигнал іншим нейронам.

Найважливіші властивості біологічних нейромереж наступні:

– паралельність обробки інформації. Кожен нейрон формує свій вихід тільки на основі своїх входів і власного внутрішнього стану під дією загальних механізмів регуляції нервової системи;

-здатність до повної обробки інформації. Всі відомі людині задачі вирішуються нейронними мережами. До цієї групи властивостей відносяться асоціативність (мережа може відновлювати повний образ за його частиною), здатність до класифікації, узагальнення, абстрагування та багато інших;

– самоорганізація. В процесі роботи біологічні нейронні мережі самостійно під впливом зовнішнього середовища навчаються вирішенню різноманітних задач. Невідомо ніяких принципових обмежень на складність задач, які вирішуються біологічними нейронними мережами. Нервова система сама формує алгоритми власної діяльності, уточнюючи та ускладнюючи їх протягом життя. Людина поки що не змогла створити системи, які мали здатність до самоорганізації та самоускладнення.

Біологічні нейронні мережі є аналоговими системами. Інформація поступає до мережі через велику кількість каналів і кодується за просторовим

принципом: вид інформації визначається номером нервового волокна, по якому вона передається. Амплітуда вхідного впливу кодується щільністю нервових імпульсів, які передаються по волокну. Надійність. Біологічні нейронні мережі мають фантастичну надійність: відмова навіть 10% нейронів в нервовій системі не перериває її роботи.

Під штучною нейронною мережею (ШНМ) будемо розуміти модель біологічної нейронної мережі. ШНМ складаються з багатьох простих елементів – штучних нейронів, функціонуючих паралельно. Штучний нейрон імітує в першому наближенні властивості біологічного нейрона [11-20].

1.2 Навчання штучної нейронної мережі

Властивості нейронних мереж навчатися на прикладах роблять їх більш привабливими в порівнянні з системами, які володіють відповідною системою правил функціонування, сформульованими експериментами.

В контексті ШНМ процес вивчення може розглядатися як настройка архітектури мережі та ваги зв'язків для ефективного виконання спеціальної задачі. Зазвичай нейтронна мережа повинна налагодити ваги зв'язку по маючій навчальній виробці. Функціональні мережі поліпшуються по мірі ітеративного налагоджування вагових коефіцієнтів.

Для конструювання процесу навчання, перед усім, необхідно мати модель навколишнього середовища, в якому функціонує нейронна мережа – знати доступну для мережі інформацію. Ця модель визначає парадигму вивчення. По друге, необхідно знати, як модифікувати вагові параметри – які правила вивчення управляють процесом налагодження ШНМ.

Алгоритм навчання – це процедура, яка використовує правила навчання для коригування вагових коефіцієнтів нейронної мережі. Існує три основні парадигми навчання: з учителем, без учителя (самонавчання) та змішана. У навчанні з учителем нейронна мережа отримує правильні відповіді (виходи)

для кожного вхідного прикладу. Ваги налаштовуються таким чином, щоб відповіді мережі максимально наближалися до заданих правильних значень. Навчання без учителя не потребує знання правильних відповідей для кожного прикладу. У цьому випадку мережа аналізує внутрішню структуру даних і виявляє кореляції між зразками, що дозволяє класифікувати їх за категоріями. Змішане навчання передбачає, що частина ваг коригується за допомогою навчання з учителем, тоді як інша частина налаштовується шляхом самонавчання. Особливий варіант навчання з учителем, відомий як посилене навчання, допускає, що замість точного правильного значення виходу нейронної мережі доступна лише загальна оцінка правильності її роботи [2].

Теорія навчання розглядає три фундаментальних властивостей, пов'язаних з навчанням на прикладах: ємкість, важкість зразків та обчислювальна важкість.

Під ємкістю розуміється кількість зразків, які може запам'ятати мережа, і те, які функції та границі прийняття рішень можуть бути сформовані за допомогою такої мережі. *Важкість зразків* визначає кількість навчальних прикладів, необхідних для досягнення здібності мережі до узагальнення [21].

1.3 Особливості архітектури штучних нейронних мереж

Використання в математичних моделях окремого штучного нейрона в зв'язку з його елементарною структурою не завжди виправдано і практично при вирішенні технічних задач майже не використовується. Більшу цікавість представляє використання груп штучних нейронів, які інформаційно пов'язані між собою. У такому випадку група нейронів взаємопов'язана так, що кожен нейрон отримує на свої входи сигнали від визначених нейронів і з зі свого входу надсилає сигнали іншим нейронам. ШНМ формується шляхом послідовного або паралельного з'єднання визначених блоків нейронів.

Штучну нейронну мережу називають однорідною, якщо всі її елементи мають однакову функцію активації. Інакше ШНМ називають змішаною.

Існують два види ШНМ:

– мережі з прямими зв'язками (одношарові та багатошарові), в яких для кожного вхідного вектора розраховується підсумковий вихідний вектор. Наявність прихованих шарів в багатошарових мережах дозволяє виділяти глобальні можливості даних за допомогою локальних з'єднань за рахунок додаткових синаптичних зв'язків і підвищення рівня взаємодії нейронів;

– мережі із зворотними зв'язками (рекурентні), в яких вихідні сигнали подаються знову на вхід мережі. Поняття зворотного зв'язку характерне для динамічних систем, в яких вихідний сигнал деякого елемента системи впливає на вхідний сигнал цього елементу.

Під зворотним зв'язком розуміється наявність елементів одиничної затримки, що приводить до нелінійної динамічної поведінки мережі.

В ШНМ першого типу відсутні зворотні зв'язки, і потік інформації в цьому випадку прямонаправлений. На відміну від ШНМ з прямим зв'язком, в мережах другого типу вихідні сигнали знову подаються на вхід, тобто ініціюється ітераційний процес. Така структура ШНМ вперше була запропонована Хопфілдом (1982 р.) і отримала назву автоасоціативної.

Функціонування нейромережі полягає в перетворенні вхідних сигналів у часі, в результаті чого змінюється внутрішній стан мережі і формуються вихідні впливи. При моделюванні ШНМ різних архітектур існує ряд ключових аспектів, яким необхідно надати особливу увагу, оскільки вони серйозно впливають на якість рішень, отриманих за допомогою ШНМ [2].

Вибір кількості нейронів і шарів. Немає строго визначеної процедури для вибору кількості нейронів і кількості шарів в мережі. Чим більша кількість нейронів і шарів, тим ширші можливості мережі і тим більш нелінійною може бути залежність вхід-вихід, але мережа повільніше навчається і працює. Кількість нейронів і шарів пов'язано:

– з складністю задачі і кількістю даних для навчання;

– потрібною кількістю входів і виходів мережі;

– наявними ресурсами: пам'яттю і швидкодією ЕОМ, на якій моделюється мережа.

Були спроби записати емпіричні формули для вибору кількості шарів і нейронів, але застосування таких формул виявилось обмеженим.

Якщо в мережі занадто мало нейронів або шарів:

– мережа не навчиться і помилка при роботі залишиться великою;

– на виході мережі не будуть передаватися різкі коливання апроксимуючої функції *y(x)*.

Якщо нейронів або шарів занадто багато:

– швидкодія буде низькою, а пам'яті потрібно буде багато;

– мережа перенавчиться: вихідний вектор буде передавати незначні і несуттєві деталі у вивчаючій залежності *y(x)*, наприклад, шум чи помилкові дані;

– залежність виходу від входу виявиться різко нелінійною: вихідний вектор буде суттєво і непередбачено змінюватись при малій зміні вхідного вектора *x*;

– мережа не буде здатна до узагальнення: в області, де немає або мало відомих точок функції y(x) вектор буде випадковим і непередбачуваним, не буде адекватним для вирішуваної задачі [3].

Підготовка вхідних і вихідних даних. Дані, які подаються на вхід мережі і знімаються з виходу, повинні бути правильно підготовлені. Одним із відомих способів – масштабування:

$$x = (x' - k)c, \tag{1.1}$$

де x' – вихідний вектор,

x – масштабований. Вектор k – усереднене значення множини вхідних даних, c – коефіцієнт масштабування.

Правило корекції помилки. При навчанні з вчителем для кожного вхідного прикладу x заданий бажаний вихід *d*. Реальний вихід мережі y може не співпадати з бажаним. Принцип корекції помилки при навчанні полягає у використанні сигналу (*d–y*) для модифікації ваги, що забезпечує поступове зменшення помилки.

Ініціалізацією архітектури нейронної мережі будемо називати мережу даної архітектури після присвоєння їй деякого початкового набору вагових коефіцієнтів. При цьому різні ініціалізації однієї і тієї ж архітектури ШНМ породжують різні нейронні мережі [24-30].

1.4 Формальний нейрон

Біологічний нейрон – складна система, математичну модель якої досі повністю не побудовано. Введено безліч моделей, що розрізняються обчислювальною складністю і схожістю з реальним нейроном. Одна з найважливіших – формальний нейрон (ФН). Незважаючи на простоту ФН, мережі, побудовані з таких нейронів, можуть сформувати довільну багатомірну функцію на виході.

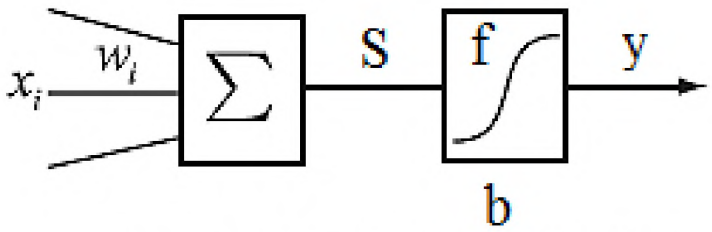

Рисунок 1.1 – Формальний нейрон

Нейрон складається з виваженого суматора і нелінійного елемента. Функціонування нейрона визначається формулами:

$$S = \sum_i w_i x_i \qquad (1.2)$$

$$y = f(S + b) \tag{1.3}$$

де: x_i – вхідні сигнали, сукупність всіх вхідних сигналів нейрона утворює вектор x;

w_i – вагові коефіцієнти, сукупність вагових коефіцієнтів утворює вектор ваг w;

S – зважена сума вхідних сигналів, значення x передається на нелінійний елемент;

b – пороговий рівень даного нейрона;

f – нелінійна функція, названа функцією активації.

Нейрон має кілька вхідних сигналів x і один вихідний сигнал y. Параметрами нейрона, що визначають його роботу, є: вектор ваг w, пороговий рівень b і вид функції активації f.

Активаційна функція має вигляд:

$$f(x) = \begin{cases} 0, \text{ якщо } x < 0 \\ 1, \text{ якщо } x \geq 0 \end{cases}; \tag{1.4}$$

Таку функцію називають пороговою або функцією одиничного стрибка.

Якщо зручно, то використовують пару величин +1 і -1:

$$f(x) = \begin{cases} -1, \text{ якщо } x < 0 \\ 1, \text{ якщо } x \geq 0 \end{cases}. \tag{1.5}$$

Найбільш використовуваними активаційними функціями є також наступні:

– сигмоїдальна функція:

$$f(x) = \frac{1}{1-e^x} \tag{1.6}$$

– функція гіперболічного тангенса:

$$f(x) = \frac{1-e^{-2x}}{1+e^{-2x}}; \tag{1.7}$$

– функція Гауса

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{\frac{-(x-e)^2}{2\sigma^2}} \tag{1.8}$$

– інверсна мультиквадратична функція

$$\mathrm{f(x)} = \frac{1}{\sqrt{(\mathrm{x-c})^2+\mathrm{d}^2}}. \tag{1.9}$$

Монотонні функції активації не впливають на класифікацію, але їх значимість можна підвищити, обравши таким чином, щоб трактувати виходи нейронів як ймовірності приналежності до відповідного класу. Це дає додаткову інформацію при класифікації.

Можна показати, що у згаданому вище випадку гаусівських розподілів ймовірностей сигмоїдальна функція активації нейрона дає ймовірність приналежності до відповідного класу.

Завдяки здібності навчатися, штучні нейронні мережі знайшли застосування при вирішенні широкого кола задач, так чи інакше пов’язаних з обробкою образів і сигналів.

Типовими задачами для нейромереж є:

– апроксимація функцій по набору точок (регресія);

– класифікація даних по заданому набору класів;

– кластеризація даних з виявленням заздалегідь невідомих класів-прототипів;

– відновлення втрачених даних;

– асоціативна пам'ять;

– оптимізація, оптимальне управління [34].

1.5 Багатошаровий персептрон та метод зворотного розповсюдження помилки

Найбільш популярний клас багатошарових мереж прямого розповсюдження створюють багатошарові персептрони, в яких кожен обчислюваний елемент використовує стрибкоподібну або сигмоїдальну функцію активації. Багатошаровий персептрон може формувати складні границі прийняття рішень та реалізувати будь-які булеві функції.

В багатошаровому персептроні нейрони розміщені в декілька шарів. Нейрони першого шару отримують вхідні сигнали, перетворюють їх і через точки розшарування передають нейронам другого шару. Далі спрацьовує другий шар і т.д., до k – го, який видає вихідні сигнали для інтерпретатора і користувача.

Якщо не обговорено особливих випадків, то кожен вихідний сигнал і – го шару подається на вхід всіх нейронів $i+1$ – го. Число нейронів в кожному шарі може бути різним і ніяк не пов'язано з кількістю нейронів в інших шарах. Найбільше розповсюдження отримали тришарові мережі, в яких кожен шар має свою назву: перший – вхідний, другий – прихований, третій – вихідний.

Розглянемо двошарову мережу. Ваги нейронів першого (вхідного шару) позначимо верхнім індексом (1), а вихідний шар – верхнім індексом (2). Вихідні сигнали першого шару позначимо $v_j (j = 1,2,...,K)$, а вихідний шар - $y_j, (j = 1,2,...,M)$. Будемо вважати, що функція активації нейронів задана в сигмоїдальній уніполярній або біполярній формі. Для спрощення опису використовується розширене визначення вхідного вектора мережі у вигляді $x = [x_0, x_1, ..., x_N]^T$, де $x_0 = 1$ відповідає порогу. З вектором x пов'язані два вихідних вектори мережі: вектор фактичних вихідних сигналів $y = [y_0, y_1, ..., y_M]^T$ і вектор очікуваних вихідних сигналів $d = [d_0, d_1, ..., d_M]^T$.

Мета навчання полягає у підборі таких значень ваги $w_{ij}^{(1)}$ і $w_{ij}^{(2)}$ для всіх шарів мережі, щоб при заданому вхідному векторі x отримати на виході

значення сигналів y_i, які з необхідною точністю будуть співпадати з очікуваними значеннями d_i для $i = 1,2,...,M$. Вихідний сигнал i-го нейрона вхідного шару описується функцією:

$$v_j = f\left(\sum_{j=0}^{N} w_{ij}^{(1)} x_j\right). \tag{1.10}$$

У вихідному шарі k-й нейрон виробляє вихідний сигнал:

$$y_k = f\left(\sum_{i=0}^{K} w_{ki}^{(2)} v_i\right) = f\left(\sum_{i=0}^{K} w_{ki}^{(2)} f\left(\sum_{j=0}^{N} w_{ij}^{(1)} x_j\right)\right). \tag{1.11}$$

Із формули випливає, що на значення вихідного сигналу впливають ваги обох шарів, тоді як сигнали, які виробляються в вхідному шарі, не залежать від ваги вихідного шару.

Основу алгоритму зворотного розповсюдження помилки створює цільова функція, сформульована, як правило, у вигляді квадратичної суми різниць між фактичними і очікуваними значеннями вихідних сигналів. Для навчальної вибірки, складеної з p прикладів, цільова функція має вигляд:

$$E(w) = \frac{1}{2}\sum_{j=1}^{p}\sum_{k=1}^{M}(y_k^{(j)} - d_k^{(j)}). \tag{1.12}$$

Мінімізація цільової функції досягається уточненням вектора ваги (навчанням) за формулою:

$$w(k+1) = w(k) + \Delta w, \tag{1.13}$$

$$\Delta w = \eta s(w), \tag{1.14}$$

де: η – коефіцієнт навчання, а $s(w)$ – напрямок в просторі ваг w.

Вибір цього напрямку зазвичай оснований на визначенні градієнта цільової функції відносно ваг всіх шарів мережі. Для ваг вихідного шару ця задача має очевидне рішення. Для інших шарів використовується метод зворотного розповсюдження помилки.

Головною ідеєю методу зворотного розповсюдження помилки є припущення, що помилка розповсюджується по нейромережі в зворотному напрямку (від виходу до входу). Тобто внесок ваги зв'язку в помилку пропорційний значенню сигналу, який проходить по зв'язку і помилці нейрона.

Таким чином, помилка зв'язку двох нейронів залежить від помилки ближнього до виходу мережі нейрона і прохідного по зв'язку сигналу. Помилка «розповсюджується» від виходів до входів. Розглянемо цей метод на прикладі двошарової мережі. В цьому випадку цільова функція визначається так:

$$E(w)=\frac{1}{2}\sum_{k=1}^{M}\left[f\left(\sum_{i=0}^{K}w_{ki}^{(2)}v_i\right)-d_k\right]^2=\frac{1}{2}\sum_{k=1}^{M}\left[f\left(\sum_{i=0}^{K}w_{ki}^{(2)}f\left(\sum_{j=0}^{N}w_{ij}^{(1)}x_j\right)\right)-d_k\right]^2. \qquad (1.15)$$

Компоненти градієнту розраховуються диференціюванням залежності (1.11). В першу чергу визначаються ваги нейронів вихідного шару. Для вихідних ваг отримуємо:

$$\frac{\partial E}{\partial w_{ij}^{(2)}}=(y_i-d_i)\frac{df(u_i^{(2)})}{du_i^{(2)}}v_j, \qquad (1.16)$$

$$u_i^{(2)}=\sum_{j=0}^{K}w_{ij}^{(2)}v_j\ . \qquad (1.17)$$

Якщо ввести позначення $\partial_i^{(2)}=(y_i-d_i)\frac{df(u_i^{(2)})}{du_i^{(2)}}$, то відповідну компоненту градієнта відносно ваг вихідного шару можна представити у вигляді:

$$\frac{\partial E}{\partial w_{ij}^{(2)}} = \partial_i^{(2)} v_{j.} . \tag{1.18}$$

Компоненти градієнта відносно нейронів вхідного шару визначаються так само, але описуються більш складною залежністю, яка випливає з існуючої функції, заданої у вигляді:

$$\frac{\partial E}{\partial w_{ij}^{(1)}} = \sum_{k=1}^{M} (y_k - d_k) \frac{dy_k}{dv_i} \frac{dv_i}{dw_{ij}^{(1)}} . \tag{1.19}$$

Звідси отримуємо:

$$\frac{\partial E}{\partial w_{ij}^{(1)}} = \sum_{k=1}^{M} (y_k - d_k) \frac{df(u_k^{(2)})}{du_k^{(2)}} w_{ki}^{(2)} \frac{df(u_i^{(1)})}{du_i^{(1)}} x_j . \tag{1.20}$$

Якщо ввести позначення:

$$\partial_i^{(1)} \sum_{k=1}^{M} (y_k - d_k) \frac{df(u_k^{(2)})}{du_k^{(2)}} w_{ki}^{(2)} \frac{df(u_i^{(1)})}{du_i^{(1)}} , \tag{1.21}$$

то отримаємо вираз, який визначає компоненти градієнту відносно ваг нейронів вхідного шару у вигляді:

$$\frac{\partial E}{\partial w_{ij}^{(1)}} = \partial_i^{(1)} x_j . \tag{1.22}$$

В обох випадках (формули (1.18) і (1.22)) опис градієнта має аналогічну структуру і представляється добутком двох сигналів: перший відповідає початковому вузлу даного зваженого зв’язку, а другий – величині похибки, перенесеної на вузол, з яким цей зв'язок встановлений. Визначення вектора

градієнта важливо для наступного процесу уточнення ваги. В класичному методі зворотного розповсюдження помилки вектор $s(w)$ у виразі (1.10) задає напрямок антиградієнту, тому:

$$\Delta w = -\eta \nabla E(w). \tag{1.23}$$

Згідно з методом зворотного розповсюдження помилки в кожному циклі навчання виділяються наступні етапи алгоритму:

1. Аналіз нейронної мережі в прямому напрямку передачі інформації при генерації вхідних сигналів, які формують черговий вектор x. В результаті такого аналізу розраховуються значення вихідних сигналів нейронів прихованих (в тому числі вхідного) шарів і вихідного шару, а також відповідні похідні $\frac{\partial f(u_i^{(1)})}{\partial u_i^{(1)}}, \frac{\partial f(u_i^{(2)})}{\partial u_i^{(2)}}, ..., \frac{\partial f(u_i^{(m)})}{\partial u_i^{(m)}}$ функцій активації кожного шару (m - кількість шарів мережі).

2. Створення мережі зворотного розповсюдження помилок шляхом зміни напрямку передачі сигналів, заміна функцій активації їх похідними і подача на минулий вихід (а в цей час – вхід) мережі сигналу у вигляді різниці між фактичним і очікуваним значенням. Для визначеної таким чином мережі необхідно розрахувати значення необхідних зворотних різниць.

3. Уточнення ваги (навчання мережі) проводиться по запропонованим вище формулам для оригінальної мережі та для мережі зворотного розповсюдження помилки.

4. Описаний процес потрібно повторити для всіх навчаючих прикладів, продовжуючи його до виконання умов зупинки алгоритму. Дія алгоритму завершується в момент, коли норма градієнта впаде нижче апріорі заданого значення ε, яке характеризує точність процесу навчання.

Можна визначити всі компоненти градієнту цільової функції, тобто всі частинні похідні функції E по вагам мережі.

Для цього, рухаючись від входів мережі (минулих виходів), потрібно перемножити всі величини, які зустрічаються на шляху (крім ваг $w_{ij}^{(k)}$, для яких розраховується частинна похідна $\frac{\partial E}{\partial w_{ij}^{(k)}}$). Крім того, там, де дуги сходяться до однієї вершини, потрібно виконати додавання добутків, отриманих на цих дугах.

Так, наприклад, щоб порахувати похідну $\frac{\partial E}{\partial w_{21}^{(2)}}$, потрібно перемножити величини $y_2 - d_2, \frac{\partial f(u_2^{(2)})}{\partial u_2^{(2)}}, v_1$, а для обчислення похідної $\frac{\partial E}{\partial w_{21}^{(1)}}$ потрібно вирахувати добутки $\pi_1 = (y_1 - d_1)\frac{\partial f(u_1^{(2)})}{\partial u_1^{(2)}} w_{12}^{(2)}$ і $\pi_2 = (y_2 - d_2)\frac{\partial f(u_2^{(2)})}{\partial u_2^{(2)}} w_{22}^{(2)}$, а потім додати ці добутки і результат помножити на $\frac{\partial f(u_2^{(1)})}{\partial u_2^{(1)}}$ і x_1.

Таким чином, отримаємо:

$$\begin{aligned}
&\frac{\partial E}{\partial w_{12}^{(2)}} = (\pi_1 + \pi_2)\frac{\partial f(u_2^{(1)})}{\partial u_2^{(1)}} x_1 = \\
&= \left[(y_1 - d_1)\frac{\partial f(u_1^{(2)})}{\partial u_1^{(2)}} w_{12}^{(2)} + (y_2 - d_2)\frac{\partial f(_2^{(2)})}{\partial u_2^{(2)}} w_{22}^{(2)} \right] \frac{\partial f(u_2^{(1)})}{\partial u_2^{(1)}} x_1 = \\
&= x_1 \sum_{k=1}^{2} (y_k - d_k)\frac{\partial f(u_k^{(2)})}{\partial u_k^{(2)}} w_{k2}^{(2)} \frac{\partial f(u_2^{(1)})}{\partial u_2^{(1)}}
\end{aligned} \quad (1.24)$$

Критеріями ефективності вирішення задачі відновлення коефіцієнтів полінома будемо вважати:

– точність апроксимації, яка забезпечується мережею;

– час навчання мережі.

Точність апроксимації, яка забезпечується ШНМ, вимірюється величиною середньоквадратичної помилки апроксимації, яка обчислюється на заданій навчальній вибірці.

Час вивчення ШНМ вимірюється кількістю елементарних математичних операцій, необхідних для налаштування ваг ШНМ, які забезпечать задану

точність апроксимації. Процес навчання може бути поділений на деяку множину циклів налаштування ваг, названих циклами навчання. Часом навчання мережі буде загальне число циклів навчання.

Зазначені критерії ефективності є суперечливими: покращення рішення по одному із цих критеріїв веде до погіршення рішення по другому критерію. Для вирішення цієї суперечливості можна використовувати наступні критерії ефективності:

– задано фіксоване цільове значення помилки апроксимації. Критерієм ефективності служить обчислювальна важкість (загальне число циклів навчання) процесу її досягнення;

– задано фіксоване загальне число циклів навчання. Критерієм ефективності служить помилка апроксимації мережі, яка досягається в результаті роботи алгоритму.

Головними недоліками методу являються:

– повільна збіжність (властивість методу градієнтного спуску);

– в силу застосування методу градієнтного спуску, функціонал середньоквадратичної помилки мінімізується до локального, а не до глобального мінімуму. В більшості випадків цього достатньо, але якщо знайдений локальний мінімум не достатній, то можливим виходом може бути спроба повторити навчання з іншими значеннями матриць V і W на першому кроці;

– мінімізація помилки при налаштуванні ваг для одного з навчальних векторів, може збільшувати помилку для другого вектора. Тому метод може не зійтись. Однак, практика показала, що такі випадки досить рідкісні.

У вищеописаному градієнтному алгоритмі можна використовувати різні градієнтні методи шляхом визначення відповідних напрямків пошуку.

Визначаючи напрямок пошуку, можна трансформувати розглянутий градієнтний алгоритм практично в будь-який алгоритм навчання ШНМ, що дозволяє значним чином спростити розробку нейромережевих систем без прив'язки до відповідного градієнтного алгоритму [31-33].

1.5.1 Послідовний та пакетний режими навчання

В практичних додатках алгоритму зворотного розповсюдження в процесі навчання багатошарового персептрона йому багаторазово надається наперед визначена множина навчаючих прикладів. Один повний цикл надання повного набору прикладів навчання називається епохою. Процес навчання проводиться від епохи до епохи, доки синаптичні ваги і рівні порогу не стабілізуються, а середньоквадратична помилка на всій навчальній множині не зійдеться до деякого мінімального значення. Пропонується випадковим чином змінювати порядок представлення прикладів навчання для різних епох. Такий принцип надання образів робить пошук в просторі ваг стохастичним, попереджуючи потенціальну можливість появи замкнутих циклів в процесі еволюції синаптичних ваг.

Алгоритм зворотного розповсюдження помилки можна реалізувати двома способами.

1. Послідовний режим (sequential mode) навчання методом зворотного розповсюдження також іноді називають стохастичним (stochastic) або інтерактивним (on-line). В цьому режимі коректування ваг проводиться після подачі кожного прикладу. Для прикладу розглянемо епоху, яка складається із N навчальних прикладів, що впорядковані таким чином: $(x(1),d(1)),...,(x(N),d(N))$. Мережі надається перший приклад $(x(1),d(1))$ цієї епохи, після чого виконуються описані вище прямі і зворотні обчислення. В результаті проводиться коректування синаптичних ваг і рівнів порогу в мережі. Після цього мережі надається друга пара $(x(2),d(2))$ в епосі, повторюються прямий і зворотний проходи, які приодять до наступної корекції синаптичних ваг і рівня порогу. Цей процес повторюється, доки мережа не завершить обробку останнього прикладу (пари) даної епохи - $(x(N),d(N))$.

2. В пакетному режимі (batch mode) навчання методом зворотного розповсюдження коректування ваг проводиться після подачі в мережу прикладів навчання (епохи). Для конкретної епохи функція вартості

визначається як середньоквадратична помилка. В цьому виразі внутрішнє сумування по j виконується по всім нейронам вихідного шару мережі, в той час як внутрішнє сумування по k виконується по всім образам даної епохи. При заданому параметрі швидкості навчання η коректування, яка застосовується до синаптичної ваги w, визначається дельта-правилом. В пакетному режимі коректування ваги Δw виконується тільки після проходження мережею всієї множини прикладів.

З точки зору процесів реального часу, послідовний режим є кращим, ніж пакетний, так як вимагає меншого об'єму внутрішнього сховища для кожного синаптичного зв'язку.

Більш того, надаючи навчальні приклади у випадковому порядку (в процесі послідовного коректування ваг), пошук в просторі ваг можна зробити дійсно стохастичним. Це, в свою чергу, скорочує до мінімума можливість зупинки алгоритму в точці якого-небудь локального мінімуму.

Варто відмітити, що стохастична природа послідовного режиму ускладнює побудову теоретичного фундаменту для знаходження умов збіжності алгоритму. В противагу цьому використання пакетного режиму забезпечує точну оцінку вектора градієнта.

Таким чином, збіжність алгоритму до локального мінімуму гарантується при достатньо простих умовах. Окрім того, в пакетному режимі простіше розпаралелити обчислення.

Якщо дані навчання містять декілька копій одних і тих же прикладі, то переважно використовують послідовний режим, так як приклади все одно подаються по одному. Ця перевага особливо помітна при великих наборах даних.

Насамкінець можна сказати, що незважаючи на велику кількість недоліків послідовного режиму алгоритму зворотного розповсюдження, він залишається дуже популярним (особливо при вирішенні задач розпізнавання образів) через дві практичні причини:

1). Цей алгоритм простий в реалізації.

2). Забезпечує ефективне вирішення важких і великих задач [35].

1.5.2 Критерій зупинки

У загальному випадку не існує доведення збіжності алгоритму зворотного розповсюдження, як не існує і якого-небудь чітко визначеного критерію його зупинки. Відомо лише декілька описаних критеріїв, які можна використовувати для зупинки коректування ваг. Кожен із них має свої практичні переваги. Для того, щоб сформулювати такий критерій, цілком логічно думати в термінах унікальних властивостей локального та глобального мінімумів поверхні помилок. Позначимо символом w^* вектор ваг, який забезпечує мінімум, будь він локальним чи глобальним. Необхідною умовою мінімуму є те, що вектор градієнта ∇w (тобто вектор частинних похідних першого порядку) для поверхні помилок в цій точці рівний нульовому. Отже, можна сформулювати критерій збіжності алгоритму навчання зворотного розповсюдження.

Вважається, що алгоритм зворотного розповсюдження збігається, якщо Евклідова норма вектора градієнта досягає достатньо малих значень.

Недоліком такого критерію збіжності є те, що для збіжності навчання можливо потрібно буде достатньо багато часу. Крім цього, необхідно постійно обчислювати вектор градієнта ∇w.

Іншою унікальною властивістю мінімуму є те, що функція вартості (чи міра помилки) $E(w)$ в точці $w = w^*$ стабілізується. Звідси можна вивести ще один критерій збіжності.

Критерієм збіжності алгоритму зворотного розповсюдження є достатньо мала абсолютна інтенсивність змін середньоквадратичної помилки протягом епохи.

Інтенсивність зміни середньоквадратичної помилки зазвичай вважається достатньо малою, якщо вона лежить в межах 0,1-1% за епоху. Іноді використовується зменшене значення – 0,01%. Нажаль, цей критерій може призвести до передчасної зупинки процесу навчання.

Існує ще один корисний і теоретично підкріплений критерій збіжності. Після кожної ітерації навчання мережі тестується на ефективність

узагальнення. Процес навчання зупиняється, коли ефективність узагальнення стає допустимою або коли виявляється, що пік ефективності вже пройдений [36].

1.6 Обгрунтування застосування нейронних мереж

Нейронні мережі не є універсальним рішенням для всіх обчислювальних завдань. У багатьох випадках традиційні комп'ютери та обчислювальні методи залишаються незамінними. Сучасні цифрові обчислювальні машини перевершують людські можливості в числових і символьних обчисленнях. Проте людина легко вирішує складні задачі, пов'язані зі сприйняттям даних, наприклад, розпізнає обличчя в натовпі, демонструючи швидкість і точність, які значно перевершують навіть найпотужніші комп'ютери. Чому ж між цими системами існує настільки суттєва різниця в продуктивності? Для відповіді на це питання порівнюють архітектуру машини фон Неймана та біологічну нейронну систему. Серед задач, які викликають інтерес у вчених та інженерів у контексті нейромоделювання, можна виділити наступні.

Класифікація образів. Це завдання визначення, до якого з попередньо заданих класів належить вхідний образ (наприклад, мовний сигнал або рукописний символ), представлений у вигляді вектора ознак. Серед поширених застосувань – розпізнавання букв, мови, класифікація електрокардіограм та аналіз клітин крові.

Кластеризація або категоризація. Ця задача, також відома як класифікація образів «без учителя», не передбачає використання навчальної множини з відомими класами. Алгоритми кластеризації ґрунтуються на схожості образів, розміщуючи схожі об'єкти в один кластер. Кластеризація застосовується для видобутку знань, стискання даних та аналізу властивостей інформації.

Апроксимація функцій. Припустимо, що є навчальна вибірка ((x_1,y_1), (x_2,y_2)..., (x_n,y_n)) (пари даних вхід-вихід), яка генерується невідомою функцією F, спотвореною шумом. Завдання апроксимації полягає в знаходженні невідомої функції F.

Передбачення/прогноз. Нехай задані n дискретних відліків $\{y(t_1), y(t_2), ..., y(t_n)\}$ у послідовні моменти часу t_1, t_2,..., t_n. Завдання полягає у передбаченні значення $y(t_{n+1})$ у деякий майбутній момент часу t_{n+1}. Передбачення/прогноз мають значний вплив на прийняття рішень у бізнесі, науці і техніці (передбачення цін на фондовій біржі, прогноз погоди).

Оптимізація. Численні проблеми в математиці, статистиці, техніці, науці, медицині й економіці можуть розглядатися як проблеми оптимізації. Задачею алгоритму оптимізації є знаходження такого рішення, що задовольняє системі обмежень і максимізує чи мінімізує цільову функцію.

Пам'ять, що адресується за змістом. В традиційних комп'ютерах звертання до пам'яті доступно тільки за допомогою адреси, що не залежить від змісту пам'яті. Більш того, якщо допущена помилка в обчисленні адреси, то може бути знайдена зовсім інша інформація. Асоціативна пам'ять, чи пам'ять, що адресується за змістом, доступна за вказівкою заданого змісту. Вміст пам'яті може бути викликано навіть по частковому вході чи спотвореному змісту. Асоціативна пам'ять надзвичайно бажана при створенні мультимедійних інформаційних баз даних.

Керування. Розглянемо динамічну систему, задану сукупністю $\{u(t), y(t)\}$, де $u(t)$ є вхідним керуючим впливом, а $y(t)$ - виходом системи в момент часу t. В системах керування з еталонною моделлю метою керування є розрахунок такого вхідного впливу $u(t)$, при якому система діє по бажаній траєкторії, заданою еталонною моделлю. Прикладом є оптимальне керування двигуном [8].

Попри переваги нейронних мереж у певних галузях над традиційними обчислювальними методами, вони не є досконалим рішенням. Нейронні мережі навчаються, але можуть допускати помилки. Крім того, неможливо

гарантувати, що створена мережа є оптимальною для заданої задачі. Використання нейронних мереж потребує від розробника виконання низки вимог. Ці вимоги включають:

– наявність множини даних, яка містить інформацію, здатну точно описати задачу;

– забезпечення достатнього обсягу даних для навчання та тестування мережі;

– розуміння сутності проблеми, яка потребує вирішення;

– вибір функції активації, суматора та методів навчання;

– знання інструментів, що використовуються у процесі розробки;

– відповідний рівень обчислювальної потужності.

Нейронні мережі відкривають новий підхід до обчислень, що вимагає від розробника навичок, які виходять за межі традиційного програмування [9].

Висновки до розділу 1

У даному розділі було проведено аналіз математичних моделей, які лежать в основі побудови штучних нейронних мереж (ШНМ), а також досліджено основні принципи їх функціонування. Розглянуто базові елементи нейронної мережі, включаючи штучний нейрон, його математичну модель та функції активації, які визначають поведінку мережі в задачах навчання. Особливу увагу приділено багатошаровим перцептронам, які є основою для розв'язання задач регресії та класифікації. Проведено аналіз алгоритму зворотного поширення помилки, який є стандартним методом навчання ШНМ. Було встановлено, що ефективність навчання залежить від коректного вибору функції втрат, алгоритму оптимізації та гіперпараметрів моделі. Також досліджено архітектури нейронних мереж, включаючи їх глибину, кількість шарів, кількість нейронів у шарах та типи активаційних функцій. Проаналізовано, як зміни в архітектурі впливають на здатність моделі до

узагальнення даних та уникнення перенавчання. Окремо розглянуто проблеми, які виникають під час навчання нейронних мереж, зокрема затухання або вибух градієнтів, надлишкову складність моделі та необхідність нормалізації даних. Відзначено роль регуляризації та методів попередження перенавчання, таких як Dropout і L2-регуляризація. У результаті аналізу було зроблено висновок, що штучні нейронні мережі є універсальними апроксиматорами, здатними моделювати довільні функції, включаючи задачі відновлення поліноміальних коефіцієнтів. Проте для досягнення високої точності необхідно ретельно добирати структуру мережі та налаштовувати процес навчання. Результати цього розділу формують теоретичну базу для подальшого моделювання та практичної реалізації ШНМ у задачах відновлення поліномів.

РОЗДІЛ 2
РОЗРОБЛЕННЯ НЕЙРОННОЇ МЕРЕЖІ

2.1 Постановка задачі розроблення нейронної мережі

Для функції:

$$y = a_0 + a_1 x + a_2 x^2 + a_3 x^3 + a_4 x^4 + a_5 x^5 + a_6 x^6 + a_7 x^7 \quad (2.1)$$

створити нейронну мережу, таку, що при подачі значення змінної X та значення функції Y на вхід нейронної мережі на виході отримаємо масив значень коефіцієнтів полінома $a_i, i = \overline{1,7}.$

Практичні задачі до розроблення:

– отримання навиків розробки прикладних програм для створення та вивчення багатошарового персептрона в системі Matlab;

– формування навчаючої вибірки для визначення значень коефіцієнта полінома по заданому набору табличних значень за допомогою штучної нейронної мережі багатошаровий персептрон;

– розробка програмного забезпечення штучної нейронної мережі в системі Matlab;

– розробка програмного продукту для відновлення коефіцієнтів полінома мовою програмування C#.

2.2 Функції пакету Neural Networks Toolbox системи Matlab

До складу пакету Neural Networks входять більш ніж 150 різних функцій, які створюють власну макромову програмування і дозволяють користувачу створювати, навчати та використовувати найрізноманітніші нейронні мережі.

За допомогою команди *help nnet* можна отримати перечень вхідних в пакет функцій. Для отримання довідки з будь-якої функції можна використовувати команду *help ім'я_функції*. Дані функції за своїм призначенням діляться на ряд груп. Розглянемо основні з них.

Функції активації і пов'язані з ними функції:

– compet(X) – функція конкуренції – в якості аргументу використовує матрицю *X*, стовпці якої асоціюються з векторами входів. Повертає розріджену матрицю з одиничними елементами, індекси яких відповідають індексам найбільших елементів кожного стовпця. Дана функція використовується при створені ШНМ із шаром нейронів, що «змагаються» (як, наприклад, в мережах зустрічного розповсюдження).

– hardlim(X) – гранична функція активації з границею $\theta = 0$; аргумент має те саме значення, що і для попередньої команди. Повертає матрицю, розмір якої рівний розміру матриці *X*, а елементи мають значення 0 або 1 – в залежності від знака відповідного елементу *X*.

– logsig(X) – сигмоїдальна логістична функція. Повертає матрицю, елементи якої є значеннями логістичної функції від аргументів – елементів матриці *X*.

– purelin(X) – повертає матрицю значень лінійної функції активації.

– tansig(X) – повертає матрицю значень сигмоїдальної (гіперболічний тангенс) функції.

– tribas(X) – повертає матрицю значень трикутної функції належності.

– dlogsig(X,Y) – похідна сигмоїдальної логістичної функції.

– dpurelin(X,Y) – похідна лінійної функції активації. Повертається матриця з одиничними елементами.

– dtansig(X,Y) – повертає матрицю значень похідної сигмоїдальної функції – гіперболічного тангенса [10].

Функції навчання нейронних мереж дозволяють встановлювати алгоритм і параметри навчання ШНМ заданої конфігурації за бажанням користувача.

В групу входять наступні функції:

– [net,tr] = trainbfg(net,Pd,Tl,Ai,Q,TS,VV,TV) – функція навчання, яка реалізує різновид квазиньютонівського алгоритму зворотного розповсюдження помилки. Аргументи функції:

– net – ім'я навчальної ШНМ;

– Pd – найменування масиву затриманих входів навчаючої вибірки;

– Tl – масив цільових значень виходів;

– Ai – матриця початкових умов вхідних затримок;

– Q – кількість навчальних пар в одному циклі навчання;

– TS – вектор часових інтервалів;

– VV – пустий масив чи масив перевіряючих значень;

– TV – пустий масив чи масив тестових даних.

Функція повертає навчену нейронну мережу *net* і набір записів *tr* для кожного циклу навчання.

Процес навчання відбувається у відповідності зі значеннями наступних параметрів (у дужках приведені значення за замовчуванням):

– net.trainParam.epochs (100) – задана кількість циклів навчання;

– net.trainParam.show (25) – кількість циклів для показу проміжних результатів;

– net.trainParam.goal (0) – цільова помилка навчання;

– net.trainParam.time (∞) – максимальний час навчання в секундах;

– net.trainParam.min_grad (10^{-6}) – цільове значення градієнта;

– net.trainParam.max_fail (5) – максимально допустима кратність перевищення помилки перевіряючої вибірки;

– net.trainParam.searchFcn ('srchcha') – ім'я одномірного алгоритму оптимізації, що використовується.

Далі коротко розглянемо основні функції навчання ШНМ:

– [net,tr] = traincgb(net,Pd,Tl,Ai,Q,TS,VV*)* – функція навчання НМ, яка реалізує алгоритм спряжених градієнтів (так званий метод Поуела-Білля).

– [net,tr] = traincgf(net,Pd,Tl,Ai,Q,TS,VV) – функція навчання НМ, яка реалізує алгоритм зворотного розповсюдження помилки спільно з методом оптимізації Флетчера-Поуела.

– [net,tr] = traingd(net,Pd,Tl,Ai,Q,TS,VV) – функція, що реалізує «класичний» алгоритм зворотного розповсюдження помилки.

– [net,tr] = trainlm(net,Pd,Tl,Ai,Q,TS,VV) – дана функція повертає ваги і зміщення НМ, використовуючи алгоритм оптимізації Левенберга-Марквардта [10].

Функції створення нейронних мереж

– network – функція створення нейронної мережі користувача.

Запис: net=network(numInputs, numLayers, biasConnect, inputConnect, layerConnec, outputConnect,targetConnect)

Опис. Функція повертає створену нейронну мережу з іменем net і з наступними характеристиками (в дужках приведені значення за замовчуванням):

– numInputs – кількість входів (0);

– numLayers – кількість шарів (0);

– biasConnect – булевий вектор з числом елементів, рівним кількості шарів (нулі);

– inputConnect – булева матриця з числом рядків, рівним кількості шарів, і числом стовпців, рівним кількості входів (нулі);

– layerConnect – булева матриця з числом рядків і стовпців, рівним кількості шарів (нулі);

– outputConnect – булевий вектор-рядок з числом елементів, рівним кількості шарів (нулі);

– targetConnect – вектор-рядок такий же, як і попередній (нулі).

– net = newcf(PR,[S1 S2…SN1],{TF1 TF2…TFN1},BTF,BLF,PF) – функція створення багатошарової ШНМ зі зворотнім розповсюдженням помилки – так званої каскадної НМ. Така мережа містить N1 прихованих

шарів, використовує вхідні функції типу dotprod і netsum, ініціалізація мережі відбувається функцією initnw. Аргументи функції:

– PR – $R \times 2$ – матриця мінімальних та максимальних значень R вхідних елементів;

– Si – розмір *i*-го прихованого шару, для *N1* шарів;

– TFi – функція активації нейронів *i*-го шару, за замовчуванням ‘tansig’;

– BTF – функція навчання мережі, за замовчуванням ‘traingd’;

– BLF – функція налаштування ваг та зміщень, за замовчуванням ‘learngdm’;

– PF – функція помилки, за замовчуванням ‘mse’.

– net = newelm(PR,[S1 S2…SN1],{TF1 TF2…TFN1},BTF,BLF,PF) – функція створення мережі Елмана. Аргументи – такі ж, як і в попередньої функції.

– net = newff(PR,[S1 S2…SN1],{TF1 TF2…TFN1},BTF,BLF,PF) – функція створення «класичної» багатошарової НМ з навчанням методом зворотного розповсюдження помилки.

– net = newfftd(PR,ID,[S1 S2…SN1],{TF1 TF2…TFN1},BTF,BLF,PF) – така ж, як і попередня функція, але з наявністю затримок по входам. Додатковий аргумент *ID* – вектор вхідних затримок.

Функції використання нейронних мереж

– [Y,Pf,Af] = sim(net,P,Pi,Ai) – функція, яка моделює роботу нейронної мережі. Аргументи: *net* – ім’я мережі, *P* – її входи, *Pi* – масив початкових умов вхідних затримок (за замовчуванням вони нульові), *Ai* – масив початкових затримок шару нейронів (за замовчуванням вони нульові). Функція повертає значення виходів *Y* і масиви кінцевих умов затримок.

Аргументи *Pi*, *Ai*, *Pf*, *Af* використовуються тільки у випадках, коли мережа має затримки по входах і по шарам нейронів.

– net = init(net) – функція ініціалізує нейронну мережу з іменем *net,* встановлюючи ваги мережі у відповідності з установками net.initFcn і net.initParam.

– [net,Y,E,Pf,Af] = adapt(net,P,T,Pi,Ai) – функція адаптації нейронної мережі. Виконує адаптацію мережі у відповідності з установками *net.adaptFcn* і *net.adaptParam*. Тут *E* – помилки мережі, *T* – цільові значення виходів (за замовчуванням – нуль); інші аргументи – як у команди *sim*.

– [net,tr] = train(net,P,T,Pi,Ai) – функція виконує навчання нейронної мережі у відповідності з установками *net.trainFcn* і *net.trainParam*. Тут *tr* – інформація про виконання процесу навчання (кількість циклів і відповідна помилка навчання).

– disp(net) – функція повертає розгорнуту інформацію про структуру і властивостях нейронної мережі.

– display(net) – те ж, що і попередня команда, але додатково повертає ім'я нейронної мережі [11].

Графічні функції:

– hintonw(W,maxw,minw) – функція повертає так званий хінтоновський графік матриці ваг, при якому кожен ваговий коефіцієнт відображається квадратом площею, пропорціональною величині даного коефіцієнта. Знак відображається кольором квадрата. Аргументи: *W* – матриця ваг, *maxw* і *minw* – мінімальне і максимальне значення її коефіцієнтів (можуть не задаватися).

– hintonw(W,b,maxw,minw) – те ж, що і попередня функція, але на графіку відображаються не тільки ваги, а й зміщення.

– plotbr(tr,name,epoch) – функція повертає графіки зміни критерія якості нейронної мережі в процесі навчання при використанні байсівського методу. Аргументи: *tr* – запис процесу навчання, *name* – ім'я ШНМ, *epoch* – кількість циклів навчання (за замовчуванням – довжина запису навчання).

– plotep(w,b,e) – функція відображає позиції ваг і зміщень на поверхні помилки ШНМ. Аргументи: *w,b,e* – відповідно, матриці ваг, зміщень і помилок. Повертається вектор, який використовується для продовження графіка, створеного функцією *plotes*.

– plotes(wv,bv,e,v) – функція повертає графік поверхні помилки нейрона з одним входом. Аргументи: *wv*, *bv* – відповідно, набори значень ваг і зміщення нейрона, *e* – матриця значень помилки, *v* – опція виду зображення.

– plotpc(W,b) – функція повертає графік лінії рішення для пересептрона. Аргументи: *W* – матриця ваг, *b* – вектор зміщень. Використовується разом з функцією *plotpv*.

– plotperf(tr,goal,name,epoch) – повертає графік зміни критерія якості ШНМ в процесі навчання. Аргументи: *tr* – запис процесу навчання, *goal* – цільове значення критерія, *name* – ім'я ШНМ, *epoch* – кількість циклів навчання.

– plotpv(p,t) – функція повертає графічне відображення вхідних *p* і цільових *t* векторів персептрона.

– plotsom(pos) – функція повертає графічне представлення розташування нейронів у самоорганізуючих картах.

– plotv(M,t) – функція графічного зображення векторів. Аргументи: *M* – матриця з двома рядками, стовпці якої асоційовані з векторами, *t* – опція, яка задає тип лінії.

– plotvec(M,C,m) – функція графічного зображення векторів різними кольорами. Аргументи: *M* – матриця з двома рядками, стовпці якої асоційовані з векторами, *C* – рядок задавання кольорів, *m* – тип точок, що вказують кінці векторів (за замовчуванням '+') [12].

Інші функції:

– errsurf(p,t,wv,bv,f) – функція, яка повертає матрицю значень поверхні помилок нейрона з одним входом і одним виходом в залежності від значень ваги та зміщення. Аргументи:

– p – вектор значень входу;

– t – вектор значень виходу;

– wv – набір значень ваг нейрона;

– bv – набір значень зміщення;

– f – назва функції активації, що реалізується (рядок).

– maxlinlr(P) – повертає максимальну величину коефіцієнта навчання для лінійного шару нейронів. Тут *P* – матриця входів.

При записі в формі maxlinlr(P,'bias') функція повертає максимальну величину коефіцієнта навчання для лінійного шару нейронів зі зміщенням.

– gensim(net,st) – функція генерує нейромережевий блок Simulink для подальшого моделювання ШНМ засобами цього пакету.

– initlay(net) – функція ініціалізації шарів нейронної мережі. В якості аргументу використовує ім'я (ідентифікатор) мережі *net.* Повертає нейронну мережу, шари нейронів в якій ініціалізовані у відповідності з функцією *net.layers{i}.initFcn*. У формі *initlay(code),* де рядкова змінна *code* може приймати значення *'pnames'* або *'pdefault'*, функція повертає інформацію про імена або про значення за замовчуванням параметрів ініціалізації.

– init(net,i) – функція ініціалізації шару i. Повертає ШНМ, ваги і зміщення в i-му шарі якої оновлені у відповідності з алгоритмом ініціалізації Нгуєна-Відроу (так, що зони «впливу» кожного нейрону в шарі розподілені рівномірно).

– initwb(net,i) – майже те ж, що і в попередньому випадку, але ваги та зміщення i-го шару ініціалізуються у відповідності з їх власними функціями іініціалізації.

– ddotprod – функція визначення похідної від результату *Z* множення матриці ваг *W* на матрицю входів *P* [13].

2.3 Моделювання штучної нейронної мережі типу багатошаровий персептрон в системі Matlab

Нехай необхідно відновити коефіцієнти полінома 7-го порядку, який описується формулою (1.9). Для розв'язання даної задачі потрібно виконати наступні дії:

1. Вибір діапазону зміну параметрів. Значення, які близькі до 0 та сам 0 виключаються у зв'язку з тим, що значення x не визначені, коли коефіцієнти функції дорівнюють нулю.

2. Кількість вхідних і еталонних векторів виберемо рівними $M = 700$. Цього достатньо для виявлення тенденції навчання, а процес навчання не займе багато часу.

3. Тестові масиви і еталони підготуємо за допомогою програми mas1.mat (див. додаток А). В цій програмі ми формуємо масив вхідних даних pp розмірності 2x700 і еталони T розмірності 7x700.

За допомогою цієї програми формується матриця P з $M = 700$ вхідних векторів-стовпців, кожен з яких сформований із 7-ми точок вихідної функції із випадково вибраними значеннями параметрів $ax^1, ax^2, ..., ax^7$ і матриця T еталонів зі 700-ми еталонними векторами-стовпцями, кожен з яких сформований із 7-ми відповідних еталонних значень, а також матриця x, яка містить значення аргументів.

Матриці P, x, T будуть використовуватися при навчанні мережі. Для подальшого навчання необхідно привести входи і виходи нейронної мережі до того вигляду, який потребує система Matlab. Отже, за допомогою команди *pp = [P', x']* ми присвоюємо масиву *pp* об'єднання транспонованого вектора P значень функції і транспонованого вектора x аргументів функції.

При кожному новому запуску цієї програми будуть формуватися масиви з новими значеннями компонентів векторів, як вхідних, так і еталонних.

4. Для вирішення поставленої задачі використаємо тришаровий персептрон з 2-ма нейронами у вхідному шарі та активаційною функцією *tansig*, 15-ма нейронами у прихованому шарі з активаційною функцією *tansig* та 7-ма нейронами у вихідному шарі з активаційною функцією *purelin*.

Мережа потрібної конфігурації створюється, використовуючи наступну команду:

net = newff([-1,1;-1,1],[2,15,7],{'tansig','tansig','purelin'},'trainlm').

5. Для навчання мережі використаємо метод зворотного розповсюдження помилки. Перед навчанням необхідно задати потрібні параметри. Задаємо функцію оцінки якості навчання *net.performFcn = 'sse'*. У цьому випадку в якості оцінки вираховується сума квадратичних відхилень виходів мережі від еталонів. Задаємо критерій закінчення навчання – значення відхилення, при якому навчання буде вважатися закінченим: *net.trainParam.goal = 0.01*. Задаємо максимальну кількість циклів навчання *net.trainParam.epochs = 1000*. Після того, як буде виконана ця кількість циклів, навчання буде завершено.

Навчання проводиться командою *[net,tr] = train(net, pp, T)*.

Після закінчення навчання мережі її збережено в файлі *nn1.mat* (див. додаток А).

Перед тим, як використовувати нейронну мережу, необхідно дослідити ступінь достовірності результатів обчислення мережі на тестовому масиві вхідних векторів.

У якості тестового використовується масив, компоненти якого відрізняються від компонентів масиву, який використовувався для навчання. Для отримання тестового масиву варто ще раз скористатися програмою *mas1*.

Для оцінки достовірності результатів роботи мережі необхідно використати результати регресійного аналізу, отримані при порівнянні еталонних значень зі значеннями, отримані на виході мережі, коли на вхід представлені вхідні вектори тестового масиву.

У середовищі Matlab для цього використовується функція *postreg*. Наступний набір команд ілюструє описану процедуру:

load mas1 – створення тестового масиву *pp*

a = sim(net, pp); – обробка тестового масиву

[m, b, r] = postreg(a(1,:), T(1,:)) – регресійний аналіз результатів обробки.

Порівняємо компоненти двох еталонних векторів з відповідними компонентами вихідних векторів мережі.

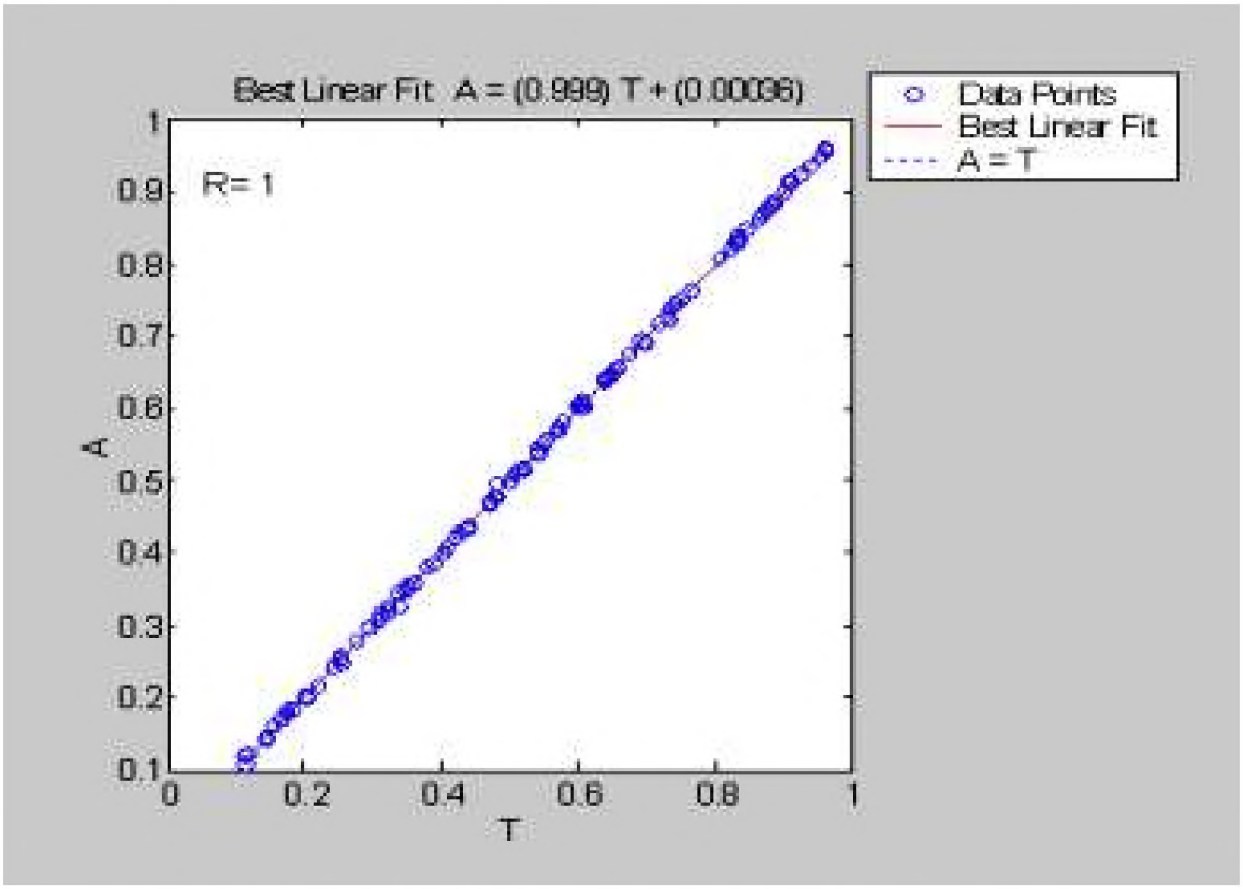

Рисунок 2.1 – Регресійний аналіз результатів обробки

Бачимо, що всі точки розташовані на прямій, що говорить про коректне функціонування мережі на тестовому масиві (рис. 2.1).

2.4 Опис роботи програмного модуля в системі Matlab

Для відновлення коефіцієнтів полінома 7-го порядку, який описується формулою (1.9), був створений програмний модуль в системі Matlab.

Щоб запустити модуль на виконання необхідно зробити наступні кроки:

1. Відкрити середовище Matlab, ввести в командне вікно *load nn1* та натиснути Enter.

2. Після цього ввести команду *a=sim(net,[Y; X])*, де *X* та *Y* – числові значення вектора аргументів та значення функції (14). Наприклад, якщо введемо *a=sim(net,[2.784; 0.847])*, то це означає, що ми хочемо знайти

коефіцієнти a_i полінома 7-го порядку, в якого значення функції $y = 2.784$, а значення аргументу функції $x = 0.847$. Після введення даних отримаємо вектор-стовпець значень коефіцієнтів полінома.

3. Для перевірки правильності обчислень введемо в командне вікно наступну команду *polyval(a, 0.847).* Ця команда видає значення функції полінома 7-го порядку, а тому його можна порівняти з введеним нами значенням функції на кроці 2. Значення між очікуваним та отриманим результатом обчислень можуть відрізнятися, але не більше, ніж на ± 1, оскільки у функції (1.9) ми маємо вільний член a_0, який є пороговим значенням та знаходиться в межах $[-1;1]$.

Перевіримо коректність роботи програми, подавши на вхід наступні значення [1.587 2.478 1.896 3.247 4.011; 0.784 0.859 0.698 0.912 0.847]. Результат бачимо на рис. 2.2.

```
>> a=sim(net,[1.587 2.478 1.896 3.247 4.011; 0.784 0.859 0.698 0.912 0.847])
a =
    0.2936    0.5463    0.4028    0.6942    0.9316
    0.2919    0.5466    0.4529    0.7000    0.9528
    0.3683    0.5845    0.4495    0.7113    0.9613
    0.6293    0.5226    0.4524    0.5802    0.8921
    0.3020    0.4821    0.5153    0.6736    0.8604
    0.4188    0.5456    0.5016    0.6229    0.8199
    0.5565    0.4907    0.5269    0.5605    0.6249
```

Рисунок 2.2 – Результат роботи програми

Використавши функцію *polyval,* визначимо значення полінома для кожного прикладу і порівняємо отримані результати. Отже, маємо:

Введені значення: 1.587 2.478 1.896 3.247 4.011

Отримані значення: 1.668 2.439 1.510 3.462 3.733

Як бачимо, різниці між відповідними введеними та отриманими значеннями не перевищують ± 0.9, а отже, можемо зробити висновок про те, що програма працює коректно.

Висновки до розділу 2

У даному розділі було виконано розроблення штучної нейронної мережі на основі багатошарового персептрона для вирішення задачі визначення коефіцієнтів полінома за заданими табличними значеннями. Отримані результати демонструють практичну реалізацію підходів машинного навчання та їх інтеграцію в сучасні програмні середовища. Було створено прикладну програму для проєктування, навчання та аналізу багатошарового персептрона. Система Matlab надала широкі можливості для реалізації процесу моделювання та тестування. Розроблено методику формування навчальної вибірки для задачі апроксимації поліномів. Вибірка включала табличні значення функції, які використовувалися для навчання нейронної мережі. Забезпечено нормалізацію та підготовку даних, що сприяло підвищенню точності моделі. У Matlab реалізовано багатошаровий персептрон, який навчався за алгоритмом зворотного поширення помилки. Здійснено підбір архітектури мережі, включаючи кількість шарів, кількість нейронів у шарах та типи функцій активації. Проведено тестування моделі, яке показало її здатність точно відновлювати поліноміальні коефіцієнти.

РОЗДІЛ 3
ПРОГРАМНА РЕАЛІЗАЦІЯ

3.1 Функціонал інстументарію, що застосовується для розроблення

Для програмної реалізації поставленої задачі було обрано мову програмування C#.

Програма виконує наступне:

– генерує тестову та навчальну вибірки для навчання штучної нейронної мережі;

– моделює ШНМ архітектури багатошаровий персептрон;

– проводить навчання ШНМ методом зворотного розповсюдження помилки;

– видає отримані результати розрахунків, обчислює значення функції (2.1) та порівнює з вхідними параметрами.

C# – це об'єктно-орієнтована мова програмування з безпечною системою типізації, створена для платформи .NET. Її розробниками стали Андерс Гейлсберг, Скот Вілтамут і Пітер Гольде під керівництвом Microsoft Research (в межах компанії Microsoft). Синтаксис C# схожий на C++ і Java. Мова відрізняється суворою статичною типізацією та підтримує такі можливості, як поліморфізм, перевантаження операторів, вказівники на методи класів, атрибути, події, властивості, обробка винятків і коментарі у форматі XML. Успадкувавши багато від мов-попередників, таких як C++, Delphi, Modula і Smalltalk, C# уникає моделей, які виявилися проблемними під час створення програмних систем, наприклад множинного спадкування класів (як у C++) [14].

C# є близьким родичем мови програмування Java. Java була розроблена компанією Sun Microsystems у відповідь на вимоги глобального розвитку інтернету, що потребував розподілених обчислень. Використовуючи як основу популярну мову C++, Java усунула потенційно небезпечні елементи, такі як

вказівники без контролю доступу до пам'яті. Для підтримки розподілених обчислень була введена концепція віртуальної машини та машинно-незалежного байт-коду, який став своєрідним посередником між вихідним кодом програм і апаратними інструкціями комп'ютерів або інших пристроїв. Java здобула значну популярність і була ліцензована також компанією Microsoft. Однак з часом Sun почала звинувачувати Microsoft у тому, що при створенні свого клонованого варіанту Java, Microsoft обмежує її сумісність лише з платформою Windows, що суперечить ідеї машинно-незалежного середовища виконання і порушує ліцензійну угоду. Microsoft відмовилася задовольнити вимоги Sun, що призвело до судового процесу. Суд визнав правоту Sun і зобов'язав Microsoft припинити використання Java поза ліцензією. У цій ситуації компанія Microsoft вирішила, скориставшись своєю впливовою позицією на ринку, розробити власний аналог Java — мову, у якій корпорація буде повністю контролювати процес. Цю мову назвали C#. Вона успадкувала від Java концепцію віртуальної машини (середовище .NET), байт-код (MSIL) та підвищену безпеку вихідного коду програм, одночасно врахувавши досвід роботи з Java. Однією з новацій C# стало значно простіше взаємодія з програмами, написаними іншими мовами, що є важливим для великих проектів. Якщо програми, написані на різних мовах, працюють на платформі .NET, то сама платформа бере на себе забезпечення сумісності, включаючи типи даних. На сьогодні C# є головною мовою корпорації Microsoft, оскільки вона найбільш повно використовує новітні можливості .NET. Інші мови підтримуються, але вважаються менш оптимальними для повноцінного використання платформи .NET.

Символ # у назві мови можна трактувати як дві пари плюсових знаків ++, що символізують розвиток мови порівняно з C++ (подібно до переходу від C до C++), а також як музичний символ дієз, що разом з літерою C утворює англійську назву ноти до-дієз. Останній варіант і став основою для назви мови. Хоча символ # (октоторп) зазвичай використовується для позначення номера на більшості клавіатур і відрізняється від музичного дієза ♯ (Unicode

U+266F), Microsoft, як розробник мови, неодноразово закликала своїх користувачів прийняти таку стилізацію.

Основним компілятором C# є Microsoft Visual C#. Також існують інші компілятори C#, які зазвичай включають реалізації Common Language Infrastructure і бібліотеки класів .NET.:

– проєкт Microsoft Rotor (який тепер зветься Shared Source Common Language Infrastructure, ліцензований тільки для навчального і дослідницького використання) забезпечує реалізації CLR runtime і компілятор C#, і підмножину бібліотек фреймворка Common Language Infrastructure, відповідно до специфікації ECMA;

– проєкт SharpDevelop від компанії icsharpcode, який використовується як альтернатива Visual Studio . Забезпечує повну реалізацію Common Language Infrastructure. Зовнішній вигляд IDE дуже нагадує Microsoft Visual C#,що робить комфортним перехід від однієї середи до іншої.

– проєкт Mono, початий компанією Ximian і продовжений її покупцем і наступником = Novell, забезпечує відкритий компілятор C#, повну відкриту реалізацію Common Language Infrastructure, включаючи потрібні бібліотеки фреймворка відповідно до специфікації ECMA, і близьку до повної реалізацію власницьких бібліотек класів Microsoft .NET до .NET 2.0, але не специфічних бібліотек .NET 3.0 і .NET 3.5, як для Mono 2.0.

– проєкт DotGNU також надає відкритий компілятор C#, близьку до повної реалізацію Common Language Infrastructure, включаючи потрібні бібліотеки фреймворка відповідно до специфікації ECMA, і підмножину деяких залишених власницьких біліотек класів Microsoft .NET до .NET 2.0 (які не документовані або не включені у специфікації ECMA, але включені у стандартне визначення Microsoft .NET Framework).

– DotNetAnywhere Micro Framework Common Language Runtime націлений на вбудовані системи, і підтримує майже всі специфікації C# 2.0 [16].

3.2 Опис розробленого програмного забезпечення

Для того, щоб запустити програму на виконання, необхідно двічі натиснути на файл *NeuroNetwork.exe*, який знаходиться в папці *NeuroNetwork*. Після запуску повинно з'явитися вікно програми, яке зображено на рис. 3.

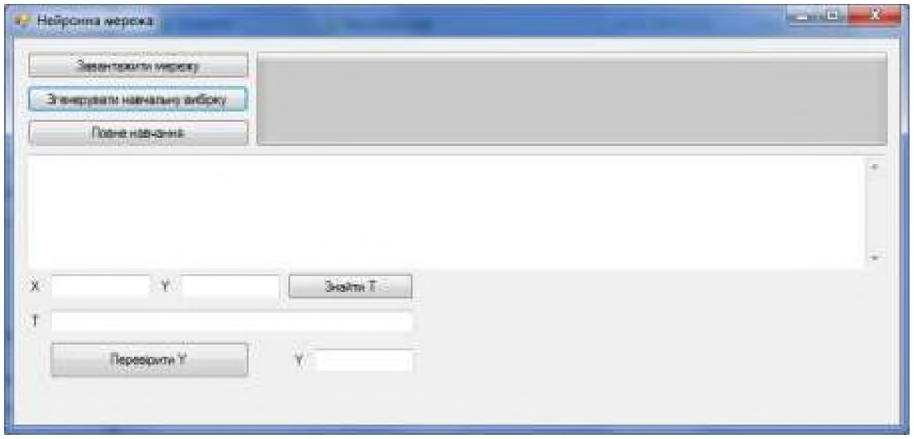

Рисунок 3.1 – Інтерфейс запуску

Розберемо детально, що виконує програма при натисканні кнопок, які зображені на рис. 3. 1.

При натисканні кнопки Завантажити мережу ми отримаємо навчену мережу і в поле виводу будуть записані значення ваг кожного шару штучної нейронної мережі. Результат після натискання кнопки Завантажити мережу зображений на рис. 3.2.

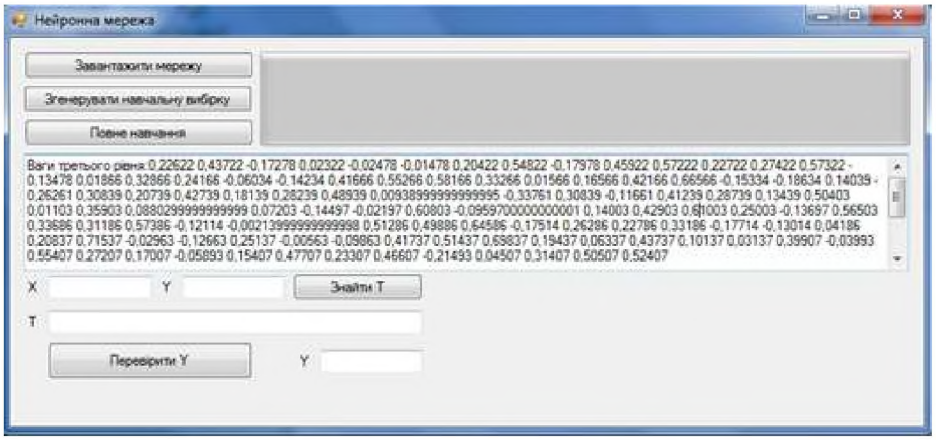

Рисунок 3.2 – Завантаження нейронної мережі

Після того, як завантажили навчену нейронну мережу, можемо вводити значення аргументу *X* та значення функції *Y* у відповідні текстові поля та натиснути кнопку Знайти *T*, після цього спрацює нейронна мережа та видасть як результат масив значень коефіцієнтів полінома 7-го порядку. Введемо значення *X=0,745* та *Y=2,475.* Результат зображено на рис. 3.3.

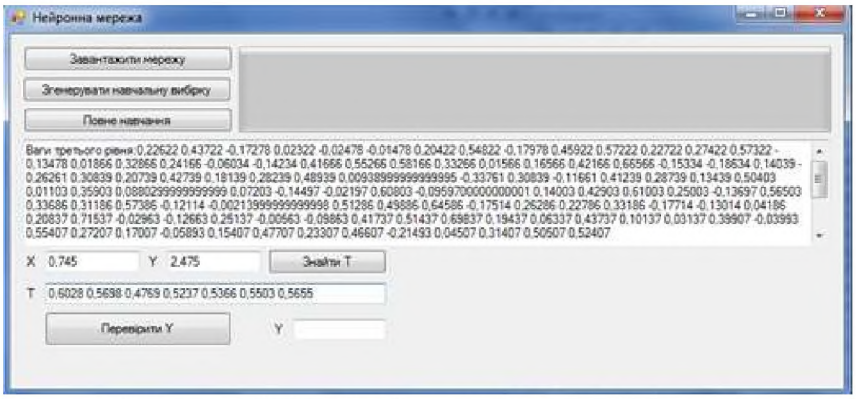

Рисунок 3.3 – Робота нейронної мережі

Отже, ми отримали сім значень коефіцієнтів полінома. Тепер потрібно перевірити чи адекватно нейронна мережа відновила коефіцієнти. Для цього натиснемо кнопку Перевірити *Y*. Результатом виконання буде значення функції (14), яке отримується при підстановці в дану функцію отриманих значень коефіцієнтів $a_i, i = \overline{1,7}$ та вхідного значення аргументу *x*. Отримане значення не повинно перевищувати ± 0.9, оскільки ми маємо ще вільний коефіцієнт a_0, який може змінюватися в таких межах. Результат зображено на рис.3.4.

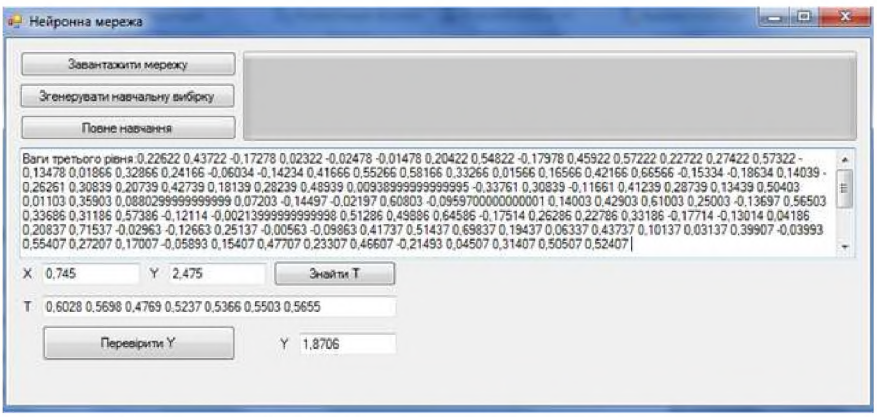

Рисунок 3.4 – Результат перевірки

На рис. 3.4 бачимо, що отримане значення $y = 1.8706$ відрізняється від вхідного значення $Y = 2.475$ на $0.604 < 0.9$, а отже, результат є адекватним.

Рисунок 3.5 – Процес навчання нейронної мережі

Розглянемо далі роботу останніх двох кнопок Згенерувати навчальну вибірку та Повне навчання. Вони служать для того, щоб можна було навчити нейронну мережу заново на деякій іншій навчальній вибірці. Для початку потрібно натиснути на кнопку Згенерувати навчальну вибірку і програма автоматично згенерує масив із 700 значень еталонних вхідних та вихідних даних і запише в текстове поле повідомлення Навчальна вибірка згенерована. Після цього можна натискати кнопку Повне навчання, яка запустить процедуру навчання нейронної мережі на згенерованій перед цим навчальній вибірці. Потім потрібно зачекати поки мережа повністю навчиться. В результаті ми отримаємо масив ваг та навчену нейронну мережу. Значення ваг програма записує у текстовий файл під назвою *weights.txt*. Після кожного нового натискання кнопки Повне навчання ваги будуть перезаписуватися у файл. Результат зображено на рис. 3.5. Код програми розміщено в додатку Б.

3.3 Порівняння результатів отриманих в системі Matlab та мовою програмування C#

Для того, щоб порівняти результати роботи нейронної мережі, яка була змодельована в двох різних середовищах програмування проведемо числові експерименти. Отже, введемо в якості вхідних даних значення $y = 2.236,\ x = 0.798$.

Результат роботи нейронної мережі, яка змодельована в системі Matlab зображено на рис. 3.6. Результат роботи програми, яка створена мовою програмування C# зображено на рисунку 3.7.

Отримали значення коефіцієнтів:

– в системі *Matlab*

$a_1 = 0.4221, a_2 = 0.5143, a_3 = 0.5142, a_4 = 0.5482, a_5 = 0.5422, a_6 = 0.5628, a_7 = 0.5124,$
$y = 2.0693.$

– в програмі мовою *C#*

$a_1 = 0.4411, a_2 = 0.5214, a_3 = 0.5823, a_4 = 0.5512, a_5 = 0.5531, a_6 = 0.5697, a_7 = 0.5447,$
$y = 2.1504.$

```
>> load nn1
>> a=sim(net,[2.236; 0.798])
a =
    0.4221
    0.5143
    0.5142
    0.5482
    0.5422
    0.5628
    0.5124
>> polyval(a,0.798)
ans =
    2.0693
```

Рисунок 3.6 – Результат роботи програми, виконаної в системі Matlab

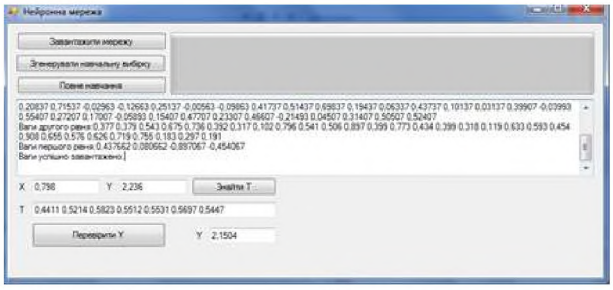

Рисунок 3.7 – Результат роботи програми, виконаної в Microsoft Visual Studio

Отже, можемо зробити висновок, що значення коефіцієнтів полінома та значення функції, отримані в різних програмах, дещо відрізняються, але на допустиму величину. І це суттєво не впливає на кінцеву мету дослідження – відновлення коефіцієнтів полінома.

3.4 Техніко-економічне обґрунтування розробки програмного продукту

Витрати на розроблення ПЗ здійснюються за формулою:

$$Кп = Кпр + Кпк + Кт \quad (3.1)$$

де $Кпр$ – витрати на проєктування ПЗ;

$Кпк$ – витрати на створення компонент ПЗ, з яких складається продукт;

$Кт$ – витрати на тестування та відлагодження ПЗ.

Загальний розрахунок на розробку ПЗ можливо виконати за формулою:

$$Кп = Фз/п\ [(1+βд)(1+βс)+βн+βпр]+t_{ПЕОМ}, \quad (3.2)$$

де $Фз/п$ – фонд основної заробітної платні розробників та інших виконавців;

$βд$ – коефіцієнт додаткової заробітної платні, можливо прийняти 0,10...0,15; $βс$ – коефіцієнт відрахування, який дорівнює 0,26;

$βн$ – коефіцієнт накладних витрат організації 0,6...0,8;

$βпр$ – коефіцієнт інших витрат 0,1...0,2;

$t_{ПЕОМ}$ – машинний час, затрачений час на тестування та відлагодження.

Відповідно до цього підходу обчислено, що загальний час роботи одного програміста над ПЗ становив 216 годин. За умови оплати праці 20 грн на годину, загальна суму розробки склала 4320 грн.

Висновки до розділу 3

У цьому розділі було здійснено програмну реалізацію штучної нейронної мережі для відновлення коефіцієнтів полінома мовою програмування C# та порівняння результатів роботи нейронної мережі, розробленої в середовищі Matlab, із результатами, отриманими в C#. Розробка програмного забезпечення на мові C# дозволила реалізувати багатошаровий персептрон, який функціонує аналогічно до моделі, розробленої в Matlab. Було використано алгоритм зворотного поширення помилки для навчання мережі та апроксимації поліноміальних коефіцієнтів. Вся обробка даних, навчання мережі та тестування були реалізовані в середовищі C# з використанням основних бібліотек для роботи з матрицями та чисельними методами. Створено інтерфейс користувача, який дозволяє вводити табличні значення та отримувати коефіцієнти полінома після навчання нейронної мережі. Це забезпечує зручність у використанні програми в реальних умовах. Модель в C# реалізована таким чином, що її можна інтегрувати в різноманітні програмні комплекси та адаптувати для застосування в різних сферах. Після реалізації нейронної мережі в обох середовищах, проведено порівняльний аналіз точності отриманих результатів. В обох випадках були використані однакові навчальні та тестові набори даних. Порівняння показало, що результати, отримані в Matlab та C#, є дуже схожими, з незначними відмінностями, що можуть бути зумовлені різними методами оптимізації, а також точністю обчислень у різних середовищах. У порівнянні продуктивності Matlab та C#, C# показав кращу ефективність при виконанні завдань на великих наборах даних завдяки кращій оптимізації виконання на компіляторі C# та використанню .NET бібліотек для роботи з матрицями. Виявлено, що час навчання нейронної мережі в C# на аналогічних даних може бути дещо довшим порівняно з Matlab, оскільки Matlab має оптимізовані вбудовані функції для числових обчислень, які часто мають високу ефективність. Однак цей недолік можна зменшити шляхом подальшої оптимізації коду на C#.

Розроблена в C# нейронна мережа має великі перспективи для інтеграції в прикладні системи, де важлива швидкість виконання та простота взаємодії з іншими програмними продуктами. Крім того, програма на C# може бути розширена для роботи з іншими типами нейронних мереж або адаптована для роботи в реальному часі з потоками даних.

У підсумку, порівняння результатів роботи нейронних мереж, реалізованих в Matlab і C#, показало, що обидва підходи мають свої переваги в залежності від контексту використання. Matlab є більш зручним для швидкого розроблення та тестування моделей, у той час як C# надає більшу гнучкість і кращу продуктивність для розробки готових прикладних рішень, що можуть бути використані у реальних умовах.

ВИСНОВКИ

У процесі виконання кваліфікаційної роботи на тему «Моделювання поліноміальних коефіцієнтів із використанням штучних нейронних мереж» були досягнуті такі основні результати та висновки:

1. Теоретична база та обґрунтування вибору методів: було проведено детальний аналіз існуючих методів відновлення поліноміальних коефіцієнтів, зокрема традиційних чисельних методів та методів на основі штучних нейронних мереж. Вибір штучної нейронної мережі, зокрема багатошарового персептрона, для задачі апроксимації поліномів обґрунтовано її універсальністю як апроксиматора довільних функцій та здатністю ефективно працювати з великими наборами даних;

2. Розробка та реалізація нейронної мережі: у результаті роботи було розроблено штучну нейронну мережу на основі багатошарового персептрона для відновлення коефіцієнтів полінома. Це дозволило досягти високої точності у відновленні коефіцієнтів за заданими табличними значеннями. Програмна реалізація в середовищі Matlab та C# показала високу ефективність і точність, що підтвердило можливість використання нейронних мереж у практичних задачах математичного моделювання;

3. Порівняння результатів у Matlab та C#: було проведено порівняння результатів, отриманих за допомогою нейронної мережі, реалізованої в Matlab і C#. Порівняння показало, що обидва середовища демонструють схожі результати в точності відновлення поліноміальних коефіцієнтів, але C# забезпечує кращу продуктивність на великих наборах даних. Це дозволяє рекомендувати використання C# для інтеграції нейронних мереж в прикладні програмні продукти, що потребують високу ефективність;

4. Практичне застосування та перспективи розвитку: розроблене програмне забезпечення є надійним інструментом для апроксимації поліноміальних коефіцієнтів і може бути адаптоване для використання в реальних прикладних задачах, таких як технічні розрахунки, моделювання

природних процесів, економічні прогнози та інші. Подальше вдосконалення моделі може включати використання інших типів нейронних мереж або покращення алгоритмів навчання для досягнення ще більшої точності та ефективності;

5. Наукова новизна та внесок у розвиток галузі: наукова новизна роботи полягає в застосуванні штучних нейронних мереж для відновлення поліноміальних коефіцієнтів з використанням нових підходів у розробці та реалізації таких моделей. Порівняння програмних рішень на різних платформах, а також розробка інтегрованого програмного продукту, є важливим внеском у розвиток практичного застосування нейронних мереж у математичному моделюванні.

Таким чином, проведена робота довела доцільність використання нейронних мереж для апроксимації поліноміальних коефіцієнтів і відкриває нові можливості для їх застосування в інженерії та науці.