МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
КРИВОРІЗЬКИЙ ДЕРЖАВНИЙ ПЕДАГОГІЧНИЙ УНІВЕРСИТЕТ
Факультет фізико-математичний
Кафедра інформатики та прикладної математики

«Допущено до захисту»
Завідувач кафедри
_____ Соловйов В. М.
(підпис)
«___» _____ 2019 р.

Реєстраційний № _____
«___» _____ 2019 р.

МЕТОДИ НЕЙРОМЕРЕЖНОЇ ІДЕНТИФІКАЦІЇ ОБ'ЄКТІВ

Кваліфікаційна робота студента
групи Ім-14
ступінь вищої освіти магістр
014.09 Середня освіта (Інформатика)
Наготнюка Юрія Олександровича
Керівник доктор педагогічних наук, професор Семеріков С. О.
Оцінка:
Національна шкала _____
Шкала ECTS _____ Кількість балів _____
Голова ЕК _____ _____
(підпис) (прізвище, ініціали)

Члени ЕК _____ _____
(підпис) (прізвище, ініціали)

_____ _____
(підпис) (прізвище, ініціали)

_____ _____
(підпис) (прізвище, ініціали)

_____ _____
(підпис) (прізвище, ініціали)

Кривий Ріг – 2019

ЗМІСТ

ВСТУП

Актуальність теми. У 1989 р. Дж. Цибенко було доведено теорему (відому зараз як універсальна теорема апроксимації, або теорема Цибенко [6]), яка стверджує, що штучна нейронна мережа прямого зв'язку з одним прихованим шаром та неперервною сигмоїдальною функцією може апроксимувати будь-яку неперервну функцію n дійсних змінних із будь-якою наперед заданою точністю. Дана теорема є теоретичним підґрунтям для побудови сучасних комп'ютерних систем апроксимації даних різної природи.

Так, у 1996 р. Б. Белідженскі запропоновано алгоритм інкрементальної апроксимації дискретних функцій, представлених парами вигляду «вхід – вихід» за допомогою нейронної мережі з одним прихованим шаром, у який з певним кроком додаються нейрони [5]. У роботах А. Лукьянюка та Б. Белідженскі 1999–2001 рр. розглянуто застосування нейронних мереж до наближення функцій (зокрема, у застосуванні до баз даних) [3; 14]. Н. П. Абовським та ін. у 2002 р. було розглянуто застосування нейромережевих моделей до задач будівельної механіки та деяких задач теорії пластин та оболонок [42]. У роботі В. О. Мохова 2005 року розглянуто принципи розробки алгоритмів прямого синтезу штучних нейронних мереж, що апроксимують, для моделювання функціональних залежностей різних типів [40]. А. І. Купіним у 2010 р. визначено побудову інтелектуальної системи керування на основі комплексного використання нейромережевої ідентифікації та нейрокерування [37].

Незважаючи на достатньо давні (з кінця 1950-х рр.) спроби апаратної реалізації нейромереж, у 1990-ті та 2000-ні роки більшість їх реалізацій була програмною. Це пов'язано як із достатньо пізньою появою ефективних методів навчання нейромереж (1980-ті рр.), так й із недосконалістю їх апаратної реалізації: так, ефективні зразки нейрочипу NeuroMatrix з'явились лише наприкінці 1990-х рр. [7].

Поява відеокарт NVIDIA та ATI, що можуть виконувати роль математичних сопроцесорів, та відповідних API (CUDA, OpenCL та ін.), надало

можливість їх використання у якості нейрочипів. У роботі О. В. Борескова та А. А. Харламова [27] показано приклади застосування CUDA для розв'язання задач обчислювальної математики, а у роботі О. О. Мясищева [41] обґрунтовано доцільність використання технології CUDA для матричних операцій. Ураховуючи, що задача нейромережної апроксимації, як показано С. Хайкіним [11], може бути ефективно подана та розв'язана у матричній формі, доцільним є дослідження проблеми апроксимації даних на основі застосування сучасних GPU та відповідних API до них.

Таким чином, існують протиріччя:

а) між потенціалом використання сучасних нейрочипів та GPU для розв'язання задач нейромережевої апроксимації та недослідженістю проблеми вибору типу мережі та її розмірності для різних типів нейрочипів та GPU;

б) між теоретичною розробленістю проблеми вибору базису апроксимації та відсутністю комп'ютерної системи апроксимації даних на основі застосування нейрочипів із автоматичним вибором базису.

Виділені протиріччя зумовили вибір теми: «Методи нейромережної ідентифікації об'єктів».

Мета дослідження – дослідити проблему використання нейрочипів для нейромережової апроксимації та розробити комп'ютерну систему нейромережевої ідентифікації об'єктів на основі застосування GPU.

Завдання дослідження:

1. Проаналізувати сучасний стан проблеми нейромережевого моделювання з метою виділення проблеми дослідження.

2. Дослідити типи апаратних засобів для розв'язання проблеми дослідження.

3. Розробити та реалізувати новий метод нейромережевої ідентифікації об'єктів для мереж глибинного навчання.

Об'єкт дослідження – комп'ютерні системи апроксимації даних.

Предмет дослідження – комп'ютерні системи апроксимації даних на основі застосування нейрочипів.

Ідея роботи полягає у використанні сучасних GPU у якості нейрочипів.

Методи дослідження: *аналіз* джерел та програмного забезпечення з метою визначення стану розв'язання проблеми дослідження та добору засобів розробки систем нейромережевої ідентифікації об'єктів, *методи програмної інженерії* (проектування, розробка, тестування) для досягнення мети дослідження.

Новизна дослідження полягає у розробці нового метода ідентифікації об'єктів у мережах глибинного навчання.

Структура роботи. Робота складається із вступу, трьох розділів, висновків, списку використаних джерел.

РОЗДІЛ 1
СУЧАСНИЙ СТАН ПРОБЛЕМИ НЕЙРОМЕРЕЖЕВОГО МОДЕЛЮВАННЯ

1.1 Особливості організації моделей

Нейромережеве моделювання знаходить застосування в різних областях людської діяльності. У техніці нейронні мережі широко використовуються при вирішенні задач класифікації, моделювання об'єктів управління та синтезу регуляторів. Такий вражаючий успіх визначається наступними причинами [32]:

1. Нейронні мережі є потужним інструментом імітації процесів і явищ, який дозволяє відтворювати надзвичайно складні залежності.

2. Нейронні мережі використовують механізм навчання. Користувач штучної нейронної мережі підбирає дані, потім запускає алгоритм навчання, який автоматично налаштовує параметри мережі. При цьому від користувача, звичайно, потрібен певний набір евристичних знань про те, як слід відбирати і готувати дані, вибирати потрібну структуру мережі та інтерпретувати результати, проте рівень знань, необхідний для успішного застосування штучної нейромережі, набагато менший, ніж, наприклад, при користуванні традиційними методами (математичного аналізу, інтегро-диференціального числення, математичного програмування та ін.).

Як показує практика використання апарату ШНМ (штучної нейромережі) при вирішенні тих чи інших завдань, найбільш ефективним застосування останнього є у випадках, коли розробник займає позицію «золотої середини», тобто поєднує застосування як теорії ШНМ, так і методів класичного аналізу.

Тим не менш, практично будь-яке завдання можна звести до задачі, що вирішується нейронною мережею. При цьому, як відомо, застосування штучних нейронних мереж супроводжується рядом недоліків.

Для того щоб, наприклад ШНМ реалізовувала задане навчальною вибіркою відображення, вона повинна мати достатнє число нейронів у прихованих шарах. Однак, в даний час, формул для точного визначення

необхідної і достатньої кількості нейронів у мережі за заданою навчальною вибіркою немає [35].

У загальному випадку побудова нейронної мережі проводиться у два етапи:

1. Вибір структури (типу) мережі.

2. Добір вагових коефіцієнтів синаптичних зв'язків мережі (навчання).

На першому етапі вирішуються завдання підбору характеристик нейронів, принципу об'єднання їх у мережу, а також визначення входів і виходів мережі. При цьому сьогодні існує достатня кількість різних нейромережевих структур, ефективність застосування багатьох з яких доведена математично. Проте все-таки більшість підходів для проектування структури та кількісного складу елементів ШНМ залишаються евристичними і часто не дозволяють отримувати однозначних рішень.

На другому етапі проводиться навчання обраної мережі за допомогою налаштування її внутрішніх елементів і зв'язків між ними. Кількість вагових коефіцієнтів синаптичних зв'язків (вагових коефіцієетів), як правило, значна і тому процес навчання може бути тривалим і складним, а також не дозволяє виконати побудову моделі з наперед заданою точністю (яка може бути визначена тільки після проведення випробувань навченої мережі).

У процесі навчання мережі відбувається уточнення параметрів моделі, яка реалізується ШНМ. Шляхом аналізу наявних у розпорядженні аналітика вхідних та вихідних даних вагові коефіцієнти синаптичних зв'язків і значення зсувів для нейронів мережі (звичайно в автоматичному режимі) намагаються налаштувати так, щоб мінімізувати різницю між бажаним сигналом і отриманим на виході змодельованим сигналом. Похибка навчання для конкретної конфігурації ШНМ визначається шляхом апробації всіх можливих результатів спостереження через мережу і порівняння вихідних значень з цільовими. Отримані різниці значень дозволяють сформувати так звану функцію похибок(критерій якості навчання). Найчастіше в якості такої функції приймається сума квадратів похибок.

Зауважимо, що серед ШНМ розрізняють мережі з лінійними і нелінійними

функціями активації для нейронів [34]. При моделюванні нейронних мереж з лінійними функціями активації нейронів, побудова алгоритму, що гарантує досягнення абсолютного мінімуму похибки навчання можливе, а для мереж з нелінійними функціями активації в загальному випадку не можна гарантувати досягнення глобального мінімуму функції похибки [25].

При такому підході до процедури навчання може виявитися корисним геометричний аналіз поверхні функції похибок. Визначимо ваги і зміщення як вільні параметри моделі і їх загальне число позначимо через N; кожному набору таких параметрів поставимо у відповідність один вимір у вигляді похибки мережі. Тоді для будь-яких поєднань ваг і зміщень відповідну функцію похибки мережі можна зобразити у $(N + 1)$-вимірному просторі, а всі такі точки утворюють деяку поверхню, звану поверхнею функції похибок. У даній ситуації мета навчання ШНМ полягає в тому, щоб знайти на цій багатовимірної поверхні глобальний мінімум.

У разі лінійної моделі ШНМ і функції похибок у вигляді суми квадратів, поверхня буде представляти собою параболоїд, який має тільки один мінімум, що може бути знайдений досить просто.

У разі нелінійної моделі поверхня похибок має набагато більш складну будову і має ряд специфічних властивостей, зокрема може мати локальні мінімуми, пласкі ділянки, сідлові точки і довгі вузькі яри.

Очевидно, що пошук глобального екстремуму для функції вищевказаного вигляду може бути в рамках традиційного підходу досить важким або неможливим. При цьому початок процесу пов'язаний, як правило, з довільним вибором точки на поверхні похибок. Алгоритм виконаний таким чином, що результатом процесу має бути знаходження глобального мінімуму. Зрештою, алгоритм зупиняється в деякому мінімумі, котрий може виявитися як локальним мінімумом, так і глобальним.

Таким чином, завдання навчання нейронних мереж зводиться до пошуку глобального екстремуму функції багатьох змінних. Серед алгоритмів вирішення цього завдання слід виділити алгоритми сполучених градієнтів [9] і Левенбсрга-

Марквардта [10].

Однак для багатьох структур нейронних мереж розроблені спеціальні алгоритми навчання. Найбільш популярним з них є алгоритм зворотного поширення похибки (АЗП) [21,18].

При використанні АЗП виникає ряд серйозних труднощів [31]:

– алгоритм не ефективний у разі, коли значення похідних по різним вагам нейромережі сильно відрізняються;

– алгоритм не дозволяє отримати швидку збіжність процесу навчання, а часто взагалі її не гарантує;

– висока ймовірність виникнення ефекту перенавчання;

– алгоритм навчання у разі приведення мережі до одного з локальних мінімумів функціоналу навчання не дозволяє «вивести» її зі сформованої ситуації, що тягне за собою необхідність збільшення числа прихованих шарів і числа нейронів в них, або застосування евристичних прийомів оптимізації.

Принципово існує три парадигми навчання: з учителем, без вчителя та змішана. У першому випадку на кожен вхідний приклад існує необхідна відповідь. Ваги налаштовуються таким чином, щоб вихідні значення мережі були якомога більш близькі необхідним відповідям. Більш «жорсткий» варіант навчання з учителем припускає, що відома тільки критична оцінка правильності виходу ШНМ, а не самі необхідні значення виходу. Для навчання без вчителя не потрібно знання необхідних відповідей на кожен приклад навчальної вибірки. У цьому випадку відбувається розподіл зразків за категоріями (кластерами) відповідно до внутрішньої структурою даних або ступенем кореляції між зразками. При змішаному навчанні вагові коефіцієнти однієї групи нейронів налаштовуються за допомогою навчання з учителем, а інші – на основі самонавчання.

Жодна з парадигм не містить у своєму базисі прийомів попередньої якісної підготовки (обробки) даних навчальної вибірки з метою скорочення (визначення оптимального) її обсягу і збереження адекватності подання інформації про майбутню модель. Скорочення вибірки дозволило б, у свою чергу, зменшити час

навчання мереж, які навчаються за першими двома парадигмами, а при використанні другої парадигми, можливо, дало б можливість знизити кількість нейронів всередині мережі.

Найбільш поширеними і вивченими в застосуванні на сьогоднішній день є такі штучні нейронні мережі, як багатошаровий перцептрон, мережа з радіальними базисними функціями активації (RBF) і деякі інші. Необхідно відзначити, що для виконання процедури навчання багатошарових мереж необхідні досить тривалі часові витрати, що не дозволяють застосовувати їх у системах реального часу та ін. Зручні у цьому плані нейронні мережі (RBF і подібні) не завжди дозволяють ефективно моделювати функціональні залежності, близькі по поведінці до лінійних (кількість нейронів у створюваних мережах часто надмірно, або відсутні раціональні критерії для проведення мінімізації числа нейронів).

Однак, при моделюванні на персональних комп'ютерах, а особливо при реалізації моделей у вигляді прикладних програм, або апаратних платформ на базі програмованих логічних інтегральних схем (ПЛІС) застосування нейромережевих моделей сьогодні є дуже перспективним.

По-перше, це викликано можливістю створення «швидких» моделей, тобто нейромережеві моделі надають можливість масового розпаралелювання обчислювальних операцій всередині себе.

По-друге, існує розвинена апаратна і програмна база для реалізації цих моделей і отримання ефективних і, в той же час, досить дешевих виробничих рішень.

При всіх наявних перевагах нейромережевих моделей, немає можливості одержувати моделі у формалізованому вигляді з чіткою оптимальною структурою і однозначно розрахованими параметрами для попередньо заданої точності моделі. Це у свою чергу позбавляє визначенності попереднього вибору характеристик обладнання (обчислювальних ресурсів) для подальшої реалізації моделей, а також гарантій працездатності цих моделей з заданими параметрами точності.

Отже, ми бачимо, що завдання створення технології для нейромережевого моделювання функціональних залежностей з заданою точністю, отриманням чіткої оптимальної структури, складу і параметрів штучних нейромереж дійсно існує і досить актуальне.

1.2 Багатошарові нейронні мереж

Перше систематичне вивчення ШНМ було зроблено МакКаллаком та Піттсом [38] на основі простих нейронних моделей – перцептронів, вивчення можливостей яких зумовило згодом появу багатошарових нейронних мереж (БНМ).

Спільною рисою БНМ, сформованих на базі перцептронів, є прямо направлення цих мереж, що характеризується передачею інформації від вхідного шару через приховані шари до вихідного шару[28]. В стандартній топології вузол i в шарі k (k = 1,2, ..., K +1) з'єднується за допомогою вагів $w_{ij}^{(k)}$ з усіма j вузлами попереднього шару k–1 (де k = 0 і до $k = K$ +1 позначають відповідно, вхідний і вихідний шари).

Модифіковані версії можуть мати прямі зв'язки між шарами, зв'язки в межах одного шару, хаотичні зв'язки між шарами натомість регулярних.

Вхідний шар перцептрона служить лише для прийому та ретрансляції вхідних сигналів на нейрони прихованого шару. У прихованих шарах відбувається основне нелінійне перетворення інформації, а вихідний шар здійснює суперпозицію зважених сигналів останнього з прихованих шарів. В якості нелінійності вузли прихованого шару використовують диференційовані сигмоїдальні функції

$$f(s) = \frac{1}{1 - e^{-s}}$$

Під навчанням перцептрона розуміють цілеспрямований процес зміни значень ваг міжшарових синаптичних зв'язків, ітеративно повторюваний до тих пір, поки мережа не придбає необхідні властивості. В основі навчання лежить використання тренувальних даних, об'єднаних в шаблони.

Кожен шаблон $<\bar{X}, D>$ включає в себе вектор відомих вхідних сигналів $\bar{X} = (\bar{X}_1, \bar{X}_2, ..., \bar{X}_v)$ і відповідний йому вектор бажаних вихідних сигналів $D = (D_1, D_2, ..., D_z)$. У процесі навчання на вхід нейронної мережі послідовно подаються дані з тренувального набору шаблонів $\Xi = \{< \bar{X}, D >_q, q = \overline{1, Q}\}$,після чого обчислюється похибка між фактичним $Y = (Y_1, Y_2, ..., Y_z)$ і бажаним виходами мережі

$$e = \| Y_q - D_q \|$$

Тут під нормою ||зазвичай розуміється евклідова відстань між векторами *Y* та *D*.

Далі за допомогою певного правила або алгоритму відбувається така модифікація параметрів мережі, щоб ця похибка зменшилась. Процес повторюється до досягнення мережею здатності виконувати бажаний тип перетворення «вхід-вихід», заданого в неявному вигляді тренувальним набором шаблонів Ξ .

Завдяки навчанню мережа набуває здатність правильно реагувати не тільки на шаблони, пред'явлені в процесі тренування, але також справлятися з іншими наборами даних з допустимого простору входів, які ніколи не з'являлися раніше. У цьому сенсі говорять, що мережа має властивість узагальнення.

Але при узагальненні на виході мережі завжди формується похибка, величина якої заздалегідь визначена бути не може. При цьому похибка має дві складові.

Перша з них обумовлюється недостатньою якістю апроксимації (рівень якості тут важко піддається попередній оцінці), виконуваної мережею кінцевих розмірів.

Друга викликається неповнотою інформації, що пропонується мережі в процесі навчання через обмежений обсяг навчальної вибірки (для визначення необхідного і достатнього обсягу і складу вибірки ніякі правила не застосовуються).

У своїх роботах силу міжшарових синаптичних зв'язків Ф. Розенблат змінював залежно від того, наскільки точно вихід перцептрона збігався з вихідним шаблоном, у відповідності з наступним правилом навчання [9]. Ваги

зв'язків збільшуються, якщо вихідний сигнал, сформований приймаючим нейроном, занадто слабкий, і зменшуються, якщо він занадто високий. Однак це просте правило мінімізації похибки може використовуватися тільки до прямо направлених мереж без прихованих шарів.

Після виконання М. Мінським і С. Пейпертом глибокого аналізу обчислювальної потужності одношарового перцептрона, з'ясувалося, що мережі такого типу не можуть реалізувати навіть логічну функцію «XOR» і їхні обчислювальні здібності дуже обмежені.

Додавання прихованих шарів, як вихід із становища, в той час був уже відомий, але ще не існувало чіткої методики для налаштування ваг такої мережі, З цим завданням успішно впоралися Д. Румельхарт, Дж. Хінтон і Р. Вільямс [18, 17], представивши її рішення у вигляді алгоритму зворотнього поширення похибки, спробу опису якого раніше вже робив П. Вербос [24]. Також незалежно і паралельно з Д. Румельхартом вченими з Новосибірська був розроблений близький до АЗП алгоритм двоїстого функціонування для навчання нейронної мережі [25].

Основу ідеї АЗП можна описати у вигляді послідовності дій, що складається з п'яти етапів.

На першому етапі проводиться ініціалізація ваг і зміщень. Ваги $w_{ij}^{(k)}$ і зміщення $w_{i_0}^{(k)}$ у всіх шарах задаються випадковим чином, як малі величини, наприклад, в інтервалі від -1 до +1.

На другому етапі мережі представляються новий вхідний вектор $\bar{X}$ і відповідний бажаний вихідний вектор D.

Па третьому кроці виконується прямий прохід: розрахунок фактичного виходу. Обчислення виходу $Y_i^{(k)}$ для i-го вузла в k-м прихованому шарі, $k = 1,2,...,K$, та Y_i у вихідному шарі виконується таким чином:

$$Y_i^{(k)} = f_\delta\left(w_{i0}^{(k)} + \sum_{j=1}^{H_{k-1}} w_{ij}^{(k)} Y_j^{(k-1)} \right), (k = 1,2,...,K) \text{ де } Y_i^{(0)} = \bar{X}_j \ ,$$

$$Y_i = f_\delta\left(w_{i0} + \sum_{j=1}^{H_K} w_{ij}^{(K+1)} Y_j^{(K)} \right),$$

де через H_k позначається кількість вузлів в k-м прихованому шарі.

На четвертому кроці виконується зворотний прохід: адаптація ваг і порогів. Для виконання цієї операції використовується рекурсивний алгоритм, що починається на вихідних вузлах і повертається до першого прихованого шару:

$$w_{ij}^{(k)}(t+1) = w_{ij}^{k}(t) + \eta\sigma_i^{(k)} Y_j^{(k-1)}, (k = 1,2,...,K+1)$$

Для $k = K+1$ член $\sigma_i^{(k)}$ описує похибку, відомий:

$$\sigma_i^{(k-t)} = (D_i - Y_i)Y_i(1 - Y_i)$$

і його можна рекурсивно порахувати для всіх інших випадків:

$$\sigma_i^{(k)} = Y_i^{(k)}(1 - Y_i^{(k)})\sum_j \sigma_i^{k-1} w_{ij}^{(k+1)}, (k = 1,2,...,K)$$

Слід зазначити, що тут член $Y_i^{(k)}(1 - Y_i^{(k)})$ є похідною сигмоїдальної функції щодо аргументу. Якщо використовується інша порогова функція, цей член необхідно змінити. Навчальний параметр η (параметр, що характеризує швидкість налаштування мережі) зазвичай вибирається в інтервалі від 0 до +1.

На п'ятому етапі проводиться повернення до повторення другого кроку послідовності.

Як можна було помітити, при використанні АЗП для мережі обчислюється похибка, що виникає у вихідному шарі і обчислюється вектор градієнту як функція вагових коефіцієнтів синаптичних зв'язків і значень зміщення для нейронів. Цей вектор градієнта вказує напрямок найкоротшого спуску по поверхні для даної точки, тому якщо просуватися в цьому напрямку, то похибка зменшиться. Послідовність таких кроків, зрештою, призведе до мінімуму того чи іншого типу.

При великій довжині кроку збіжність алгоритму буде більш швидкою, але виникає серйозна небезпека переступити через рішення (мінімум функції похибок) або піти від нього в неправильному напрямку. Класичним прикладом

такого явища при навчанні ШНМ є ситуація, коли алгоритм дуже повільно просувається по вузькому яру з крутими схилами, переступаючи з одного схилу на інший. Навпаки, при малому кроці, ймовірно, буде вибрано вірний напрям, однак при цьому буде потрібно дуже багато ітерацій (сам процес навчання може сильно затягнутися за часом). На практиці величина кроку вибирається пропорційної крутизні схилу (градієнту функції похибок); такий коефіцієнт пропорційності називається параметром швидкості налаштування. Правильний вибір параметра швидкості налаштування ШНМ залежить від конкретного завдання і зазвичай здійснюється дослідним шляхом; цей параметр може також залежати від часу, зменшуючись по мірі виконання алгоритму.

Алгоритм діє ітеративно, і його кроки прийнято називати епохами або циклами. На кожному циклі на вхід ШНМ послідовно подаються всі навчаючі спостереження, вихідні значення порівнюються з цільовими значеннями, і обчислюється функція похибки. Значення функції похибки, а також її градієнта використовуються для коригування ваг і зміщень, після чого всі дії повторюються. Процес навчання припиняється або коли реалізовано певна кількість циклів, або коли похибка досягає деякого малого значення чи перестає зменшуватися.

Незважаючи на те, що АЗП своєю появою надав можливість навчання багатошарових мереж, він не став інструментом, який дозволив би вирішити корінне питання синтезу нейронних мереж – глобальну оптимізацію параметрів і структури мережі.

Ініціалізація початкових параметрів мережі здійснюється тут випадковим чином, а сам АЗП, відомий у статистиці як метод стохастичною апроксимації, є за своєю суттю не більше ніж локальним методом.

У силу даної обставини АЗП не може гарантувати закінчення процесу навчання в точці глобального екстремуму. Разом із тим, не викликає сумнівів, що похибка (1.2), яка використовується для оцінки якості ШНМ, є багатоекстремальною функцією параметрів мережі, і тому для пошуку її мінімуму потрібно відповідно, глобальний підхід.

Тим не менш, АЗП до цих пір є однією з ключових позицій в теорії чисельних методів моделювання.

1.3 Мережі з радіальними базисними функціями активації

Мережі з радіальними базисними функціями активації (чи RBF -мережі) складаються з більшого числа нейронів, ніж стандартні БНМ (багатошарові нейронні мережі), що навчаються по методу зворотного поширення похибки, але на їх створення потрібно значно менше часу. Ці мережі особливо ефективні, коли доступна велика кількість навчальних прикладів.

Існує досить багато різних типів і похідних від RBF-мереж [39]. Найбільш відомими серед них є наступні ШНМ.

Нейронні мережі GRNN (Generalized Regression Neural Network). Найбільш відомий їх опис дав у своїй роботі П. Вассерман [22]. Ці ШНМ призначені для вирішення завдань узагальненої регресії, аналізу часових рядів і апроксимації функцій.

Нейронні мережі PNN (Probabilistic Neural Network), які також описані в [22] і застосовуються для вирішення імовірнісних завдань, зокрема завдань класифікації.

Фінським ученим Т. Кохоненом [13] описаний цікавий клас самоорганізуючих ШНМ, нейрони яких можуть бути навчені виявленню груп (кластерів) векторів входу, що мають деякі загальними властивості. Прийнято істотно розрізняти мережі з неврегульованими нейронами, які часто називають шарами Кохонена, та ШНМ з впорядкуванням нейронів – карти Кохонена. Для карт Кохонена характерне відображення структури даних таким чином, що близьким кластерам даних на карті відповідають близько розташовані нейрони.

Нейронні мережі LVQ (Learning Vector Quantization) є розвитком мереж Кохонена, і, як правило, виконують і кластеризацію і класифікацію векторів входу.

Структура і загальний принцип дії перерахованих тут ШНМ (повністю, або частково) співпадає з RBF-мережами загального вигляду. Тому далі усі основні

ідеї розглядатимуться відносно RBF-мереж загального вигляду, основною відмітною характеристикою яких є те, що вони піддаються дуже простій рекурсії, що не потребує налаштування [11].

Свого часу велика увага була приділена доказу універсальності нейронних мереж для вирішення завдань апроксимації довільної функції з будь-якою мірою точності. Так як і для мереж перцептронного типу з сигмоїдальними функціями активації [12, 6], через деякий час це було зроблено і для RBF-мереж [16].

Відповідно до того, як це описано в роботі С. Хайкина [11], тренування RBF-мережі, що здійснює апроксимацію функції, заданої в неявному вигляді набором шаблонів полягає в наступному (рис. 1.1).

Мережа характеризується трьома особливостями:

1) єдиний прихований шар;

2) тільки нейрони прихованого шару мають нелінійну активаційну функцію;

3) синаптичні ваги усіх нейронів прихованого шару дорівнюють одиниці. Нехай V – кількість входів мережі, H – кількість нейронів прихованого шару, Z – кількість виходів мережі.

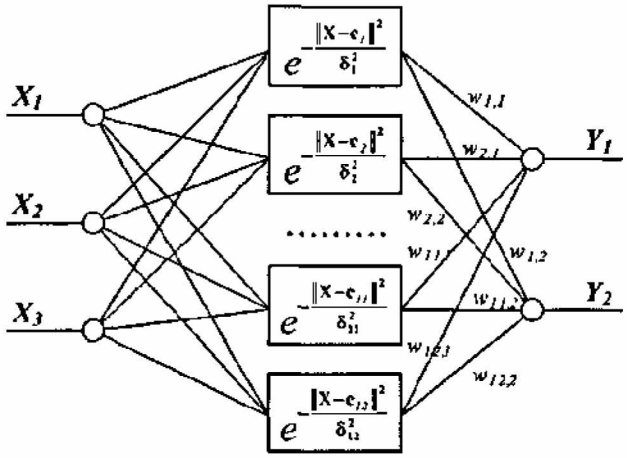

Рис. 1.1 RBF-мережа загального вигляду

Припустимо, що розмір Q набору тренувальних шаблонів Ξ не занадто великий і що шаблони розміщено досить розріджено в просторі вхідних сигналів мережі $\overline{X} = (\overline{X}_1, \overline{X}_2, ..., \overline{X}_v)$.

Уведемо наступні позначення:

$c = (c_1, c_2, ..., c_y)$ – вектор координат центру активаційної функції нейрона прихованого шару;

δ_j – ширина вікна активаційної функції j-го нейрона прихованого шару;

$f(X, c) = e^{-\|X-c\|^2/\delta^2} = e^{-\sum_{j=1}^{j}\|X_j-c_j\|^2/\delta^2}$ – радіально-симетрична активаційна функція нейрона прихованого шару;

w_{ij} – вага зв'язку між i-м нейроном вихідного шару та j-м нейроном прихованого шару.

Синтез і навчання мережі включає декілька етапів, об'єднаних наступним алгоритмом.

1. Вибирається розмір прихованого шару H рівний кількості тренувальних шаблонів Q. Синоптичні ваги нейронів прихованого шару приймаються рівними 1.

2. Центри активаційних функцій нейронів прихованого шару розміщуються в точках простору вхідних сигналів мережі, які входять в набір тренувальних шаблонів $\Xi : c_j = \bar{X}_j, j = \overline{1, H}$.

3. Ширини вікон активаційних функцій нейронів прихованого шару $\delta_j, j = \overline{1, H}$ вибираються досить великими, але так, щоб вони не накладалися один на одного в просторі вхідних сигналів мережі.

4. Визначаються ваги нейронів вхідного шару мережі $w_{ij}, i = \overline{1, Z}, j = \overline{1, H}$.

Мережі для цього пред'являється увесь набір тренувальних шаблонів. Вихід i-го нейрона вихідного шару для p-го шаблону буде рівний:

$$Y_i = w_{i1} f(\bar{X}_p, c_1) + w_{i2} f(\bar{X}_p, c_2) + ... + w_{iH} f(\bar{X}_p, c_H) =$$
$$= w_{i1} f(\bar{X}_p, \bar{X}_1) + w_{i2} f(\bar{X}_p, \bar{X}_2) + ... + w_{iH} f(\bar{X}_p, \bar{X}_H) = D_i,$$

Розписуючи це рівняння для усіх виходів мережі і усіх шаблонів, маємо наступне рівняння в матричній формі:

$$\Phi w^T = D,$$

де $\Phi = \begin{pmatrix} f_{11} \cdots f_{1H} \\ f_{21} \cdots f_{2H} \\ \cdots\cdots\cdots \\ f_{H1} \cdots f_{HH} \end{pmatrix}$ – інтерполяційна матриця;

$$f_{ij} = f(\overline{X}_i, \overline{X}_j);$$

$w = \begin{pmatrix} w_{11} \cdots w_{1Z} \\ w_{21} \cdots w_{2Z} \\ \cdots\cdots\cdots \\ w_{H1} \cdots w_{HZ} \end{pmatrix}$ – матриця вихідних синоптичних вагів;

$D = \begin{pmatrix} D_{11} \cdots D_{1Z} \\ D_{21} \cdots D_{2Z} \\ \cdots\cdots\cdots \\ D_{H1} \cdots D_{HZ} \end{pmatrix}$ – матриця вихідних шаблонів;

Рішення

$$w^T = \Phi^{-1} D$$

дає шукані значення вихідних синоптичних ваг, що забезпечують проходження інтерполяційної поверхні через тренувальні шаблони в просторі вихідних сигналів мережі.

Недоліком RBF-мережі є те, що похибка апроксимації в точках вхідного простору, не співпадаючих з центрами активаційних функцій, залежить від того, наскільки вдало проектувальником вибрані ширини вікон, і чи адекватна кількість тренувальних шаблонів складності функціонального перетворення.

Процедура налаштування синоптичних ваг є далеко не єдиною і не останньою проблемою, що зустрічається при навчанні мережі. Куди складнішим і досі відкритим питанням залишається формування набору тренувальних шаблонів, адекватно того, що описує дане функціональне перетворення.

1.4 Аналіз можливих перспектив реалізації і застосування штучних нейронних мереж

У попередніх пунктах були описані різні підходи до побудови

нейромережевих моделей. Прийнято вважати, що нейронні мережі і апарат інтегро-дифференційного числення сьогодні є двома різними (емпіричною і класичною) гілками в моделюванні [28]. При цьому слід зазначити, що створення нейромереж не завжди пов’язане з проблемами моделювання.

ШНМ здатні запам’ятовувати, а після – відтворювати поведінку об’єктів в ситуаціях, які їм були невідомі. При цьому існує думка, що для нейронних мереж аналітична форма подання знань недоступна [28, 35, 44], ШНМ здатні запам’ятати і узагальнити тільки конкретні емпіричні знання.

Для класичної гілки характерне те, що перед синтезом моделі об’єкта або процесу заздалегідь обов'язково проводиться аналіз, в ході котрого об'єкт умовно розділяється на дещо простіші складові, кожна з яких піддається ретельнішому дослідженню. Як результат, формується проста, легко формалізована модель об’єкту з гранично зрозумілою фізичною інтерпретацією.

При використанні сучасних засобів обчислювальної техніки головною перевагою першої гілки є можливість побудови «швидкої» моделі об’єкту (на основі ідеї масового паралелізму обчислень усередині ШНМ та із застосуванням відповідних апаратних засобів). Друга ж гілка зручна побудовою простих зрозумілих моделей, легко формалізованих після проведення попереднього аналізу.

Оскільки спектр завдань моделювання із застосуванням нейронних мереж великий і постійно розширюється, то природно, що завдання формування нейромережевих моделей з очевидною інтерпретацією, їх налаштування, а також оптимізація структур ШНМ є актуальною. Особливо треба відмітить останні тенденції в розробці перепрограмованих апаратних моделей для проведення паралельних обчислень. У цьому плані значних результатів в створенні мікропроцесорів, поєднаних з FPGA, досягла каліфорнійська компанія Stretch, Inc. Одно з останніх рішень, виконане у вигляді єдиного кристала з RISC-процесором і матрицею програмованих вентилів, випущено у вигляді серії мікропроцесорів Stretch 55000. Електричні ланцюги FPGA можна швидко і багаторазово налаштовувати програмними методами при ціні кристала всього 50

доларів США. Опубліковані дані по проведенню порівняльних тестів роботи вбудованого RISC-процесора самостійно та з модулем FPGA показали, що використання FPGA прискорює [43]: підрахунок контрольної суми ГСР-пакетів у 16 разів; швидке перетворення Фур'є (по 256 комплексних точках) – у 51 раз; корекцію похибок у GSM-протоколі (Viterbi-декодер) у 120 разів.

Таким чином, перераховані факти роблять дуже привабливою можливість розробки нових, сучасніших методик проектування, навчання і використання ШНМ з реалізацією останніх на основі сучасної технологічної бази.

Виходячи з вищевикладеного, можна відмітити, що:

– використання традиційних засобів побудови нейромережевих моделей є дуже ефективним, але не завжди гарантуючим позитивний результат, тривалим процесом, що вимагає для проведення наявності певного досвіду у проектувальника;

– широке використання програмних засобів нейромережевого моделювання часто базується на емпіричному уявленні щодо досліджуваного об'єкту (процесу). Аналіз (чи пряме дослідження) об'єкту моделювання з формуванням структури і параметрів із раніше заданою точністю для моделі зазвичай не практикується.

Висновки до розділу 1

1. Визначено труднощі, що виникають при використанні традиційного для навчання штучних нейронних мереж алгоритму зворотного поширення помилки:

– алгоритм не ефективний у разі, коли значення похідних по різним вагам нейромережі суттєво відрізняються;

– алгоритм не дозволяє отримати швидку збіжність процесу навчання, а часто взагалі її не гарантує;

– виникає висока ймовірність виникнення ефекту перенавчання;

– алгоритм навчання у разі приведення мережі до одного з локальних мінімумів функціоналу навчання не дозволяє «вивести» її зі сформованої ситуації, що тягне за собою необхідність збільшення числа прихованих шарів і

числа нейронів в них, або застосування евристичних прийомів оптимізації.

2. Показано доцільність матричного подання штучної нейронної мережі для визначення вагових коефіцієнтів її міжшарових з'єднань.

3. Визначено підходи до прискорення алгоритмів навчання з використанням нової технологічної бази.

РОЗДІЛ 2
АПАРАТНІ ЗАСОБИ ДЛЯ РЕАЛІЗАЦІЇ ЗАДАЧІ НЕЙРОМЕРЕЖЕВОЇ ІДЕНТИФІКАЦІЇ ОБ'ЄКТІВ

2.1 Загальні відомості про нейрочипи

Нейропроцесор – це кристал, який забезпечує виконання нейромережевих алгоритмів в реальному масштабі часу [33].

Серед різновидів кристалів, що використовуються в якості нейропроцесорів виділимо наступні: спеціалізовані нейрочіпи; замовні кристали (ASIC); мікроконтролери, що можуть бути вбудовані (mC); процесори загального призначення (GPP); логічні інтегральні схеми, що можуть бути перепрограмовані (FPGA, ПЛІС); процесори цифрової обробки сигналів (ПЦОС); трансп'ютери.

Спеціалізовані нейрочіпи часто реалізуються на основі процесорних матриць(систолічних процесорів). Такі нейрочіпи близькі до звичайних RISC-процесорів, об'єднують у своєму складі деяке число процесорних елементів, а додаткова логіка, та логіка управління, як правило, будуються на базі додаткових схем.

Розрізняють також нейросигнальні процесори, ядро яких є типовий ПЦОС, а реалізована на кристалі додаткова логіка забезпечує виконання характерних нейромережевих операцій(наприклад, додатковий векторний процесор і тому подібне).

Для оцінки продуктивності пристроїв (реалізованих на основі ПЦОС і ПЛІС), вживаних для ЦОС (цифрової обробки сигналів), контролюється час виконання типових операцій ЦОС, таких як цифрова фільтрація, ШПФ та ін. У свою чергу, для оцінки продуктивності нейропроцесорів і нейрокомп'ютерів застосовується ряд спеціальних показників (параметрів):

– ММАС – мільйонів множень з накопиченням в секунду;

– CUPS (Connections Update per Second) – число змінених значень вагів у секунду (оцінює швидкість навчання);

– CPS (Connections per Second) – число з'єднань (множень з накопиченням) на секунду (оцінює продуктивність);

– CPSPW = CPS/*Nw*, де *Nw* - число синапсів в нейроні;

– CPPS – число з'єднань примітивів в секунду:

Орієнтація процесорів на виконання нейромережевих операцій обумовлює, з одного боку, підвищення швидкостей обміну між пам'яттю і паралельними арифметичними пристроями, а з іншої сторони, зменшення часу вагового підсумовування(множення і накопичення) за рахунок застосування фіксованого набору команд типу регістр-регістр.

Основна відмінність нейрочіпів від інших процесорів – це забезпечення високого паралелізму обчислень за рахунок застосування спеціалізованого нейромережевого логічного базису або конкретних архітектурних рішень. Використання можливості представлення нейромережевих алгоритмів для реалізації на нейромережевому логічному базисі якраз і є основною передумовою різкого збільшення продуктивності нейрочіпів.

Нейропроцесори можна класифікувати за такими ознаками:

– за типом логіки нейропроцесори розділяють на цифрові, аналогові і гібридні;

– за типом реалізації нейромережевих алгоритмів – з повністю апаратною реалізацією і з програмно-апаратною реалізацією (коли нейроалгоритми зберігаються в ПЗП);

– за характером реалізації нелінійних перетворень;

– нейропроцесори з жорсткою структурою нейронів (апаратно реалізовані) і нейрокристали із структурою, що можна настроїти (перепрограмовані);

– за гнучкістю структури нейронних мереж – нейропроцесори з жорсткою і змінною нейромережевою структурою.

Виробництво спеціалізованих нейрочіпів ведеться у багатьох країнах світу.

Більшість з них орієнтуються на закрите використання (так як створюються для конкретних прикладних систем), проте серед нейрочіпів є й універсальні кристали.

Окрім широко спектру фірм і корпорацій, дослідження в області сучасних нейропроцесорів проводять багато лабораторій і університетів, серед яких можна відмітити [29]:

– у США: Naval Lab, MIT Lab, Пенсильванский університет, Колумбійський університет, Університет Арізони, Університет Ілінойса та ін.;

– у Європі: Берлінський технічний університет, Технічний університет в Карлсруе та ін.;

– у Росії: МФТІ, Ульянівський державний технічний університет, МДТУ ім. М. Е. Баумана.

2.2 Застосування GPU в якості нейрочіпа

Відомим обмеженням розвитку нейромережевих алгоритмів слід визнати високі обчислювальні витрати на реалізацію таких методів [36]. До традиційних способів вирішення цієї проблеми відносять організацію паралельних і розподілених обчислень на спеціалізованому апаратному забезпеченні, такому як нейронні чіпи, систолічні нейропроцесори, ПЛІС, розподілені кластерні системи [30], GRID-технології.

Апаратно-програмний комплекс CUDA (Compute Unified Device Architecture) дозволяє використати процесори відеокарт (GPU) як прискорювачі наукових і інженерних розрахунків і проводити обчислення, по ефективності порівняні з сучасними кластерними системами [8], [1].

Принципова відмінність архітектури полягає в наступному: виконання команд на кластері відбувається в стилі MIMD (багато потоків команд, багато потоків даних – коли набір процесорів незалежно виконує різні набори команд, оброблює різні набори даних), а в середовищі CUDA характеризується стилем SIMD (один потік команд, багато потоків даних – коли декілька процесорів виконують одну і ту ж команду над різними даними, зазвичай елементами масиву).

Особливістю устаткування, що підтримує технологію CUDA, є можливість забезпечувати на порядок більшу (в порівнянні з кластерами) пропускну

спроможність при роботі з пам'яттю. Зокрема, при обчисленні кулонівської іонізації надвеликих структур, таких як віруси, може знадобитися декілька днів роботи кластера середніх розмірів, тоді як рішення тієї ж задачі на персональному комп'ютері, оснащеному відеокартою з підтримкою CUDA, можна провести менш ніж за годину [20].

Ще однією важливою характеристикою рішень на CUDA є вартість. Так, персональний суперкомп'ютер NVIDIA Tesla C2050 має продуктивність 520GFLOPS (мільярдів операцій плаваючої арифметики в секунду) і коштує в 10 разів менше традиційного кластера тієї ж продуктивності, побудованого на основі тільки центральних процесорів(CPU) [15], і, що більше важливо, має в 20 разів більшу ефективність обчислень на ват потужності.

Висновки до розділу 2

1. Визначено відмінності нейрочіпів від інших процесорів: забезпечення високого паралелізму обчислень за рахунок застосування спеціалізованого нейромережевого логічного базису або конкретних архітектурних рішень. Показано, що використання можливості представлення нейромережевих алгоритмів для реалізації на нейромережевому логічному базисі є основною передумовою різкого збільшення швидкості реалізації алгоритмів навчання штучних нейронних мереж.

2. Обгрунтовано доцільність використання апаратно-програмного комплексу CUDA в якості нейрочіпу.

РОЗДІЛ 3
РЕАЛІЗАЦІЯ ЗАДАЧІ НЕЙРОМЕРЕЖЕВОЇ ІДЕНТИФІКАЦІЇ ОБ'ЄКТІВ

3.1 Апроксимація

Апроксимація – процес добору емпіричної функції $\varphi(x)$ для визначення з дослідних даних функціональної залежності $y = \varphi(x)$. Основне завдання апроксимації – побудова наближеної (апроксимуючої) функції, що найближче проходить біля даних точок або біля цієї безперервної функції.

Емпіричні формули служать для аналітичного представлення даних досліджень.

У найпростішому випадку завдання апроксимації експериментальних даних виглядає таким чином: нехай є якісь дані, отримані практичним шляхом(в ході експерименту або спостереження), які можна представити парами чисел (x, y). Залежність між ними відображає таблиця:

x_i	x_0	x_1	x_2	…	x_n
y_i	y_0	y_1	y_2	…	y_n

На основі цих даних потрібно дібрати функцію $y = \varphi(x)$, яка найкращим чином згладжувала б експериментальну залежність між змінними і за можливості точно відображала б загальну тенденцію залежності між x та y, виключаючи погрішності вимірів і випадкові відхилення. Це означає, що відхилення $y_i - \varphi(x_i)$ у якомусь сенсі були б найменшими.

Зазвичай завдання апроксимації ділиться на дві частини:

1. Спочатку встановлюють вид залежності $y = f(x)$ і, відповідно вид емпіричної формули.

2. Після цього визначаються чисельні значення невідомих параметрів (констант) вибраної емпіричної формули, для яких наближення до заданої функції виявляється найкращим.

3.2 Нейромережева апроксимація

Багатошаровий перцептрон можна розглядати як практичний механізм реалізації нелінійного відображення «вхід-вихід» загального вигляду. Універсальну теорему апроксимації можна розглядати як природне розширення теореми Вейерштраса [23]. Ця теорема стверджує, що будь-яка безперервна функція на замкнутому інтервалі дійсної осі може бути представлена рядом поліномів, що абсолютно і рівномірно сходиться.

Теорема про універсальну апроксимацію для нелінійного відображення «вхід-вихід» формулюється так:

Нехай $\varphi(\cdot)$ – обмежена, не постійно монотонно зростаюча неперервна функція. Нехай I_{m_0} – m_0-вимірний одиничний гіперкуб $[0,1]^{m_0}$. Нехай простір неперервних на I_{m_0} функцій позначається $C(I_{m_0})$. Тоді для будь-якої функції $fC(I_{m_0})$ та $\varepsilon > 0$ існує таке ціле число m_1 та множина дійсних констант α_i, β_i та ω_{ij}, де $i = 1,\ldots, m_1, j = 1,\ldots, m_0$, що

$$F(x_1,\ldots,x_{m_0}) = \sum_{i=1}^{m_1} \alpha_i \varphi\left(\sum_{j=1}^{m_0} \omega_{ij} x_j + b_i \right) \quad (3.1)$$

є реалізацією апроксимації функції $f(\cdot)$, тобто

$$\left| F(x_1,\ldots,x_{m_0}) - f(x_1,\ldots,x_{m_0}) \right| < \varepsilon$$

для всіх $x_1, x_2, \ldots, x_{m_0}$, приналежних до вхідного простору.

Теорема про універсальну апроксимацію безпосередньо застосована до багатошарового перцептрона. По-перше, відмітимо, що в моделі багатошарового перцептрона як функції активації використовується обмежена, монотонно зростаюча логістична функція $1/[1+\exp(-\upsilon)]$, що задовольняє умовам, що накладаються теоремою на функцію $\varphi(\cdot)$. По-друге, відмітимо, що вираз описує вихідний сигнал перцептрона наступного вигляду:

– мережа містить m_0 вхідних вузлів і один прихований шар, що складається з m_1 нейронів. входи позначені $x_1, x_2, \ldots, x_{m_0}$;

– прихований нейрон *i* має синаптичні ваги $\omega_1,...,\omega_{m_0}$ і поріг b_i;

– вихід мережі є лінійною комбінацією вихідних сигналів прихованих нейронів, зважених синаптическими вагами вихідного нейрона – $\alpha_1,...,\alpha_{m_1}$.

Теорема про універсальну апроксимацію є теоремою існування, тобто математичним доказом можливості апроксимації будь-якої неперервної функції.

3.3 Програмна реалізація ідентифікації об'єкта багатошаровим перцептроном

Розглянемо структуру класу для моделювання багатошарового перцептрону на основі бібліотеки стандартних шаблонів C++:

```
#include <math.h>
#include <stdlib.h>
#include <cstdlib>
#include <iostream>
#include <fstream>
#include <vector>
```

Визначимо структури даних: dvector – одновимірний вектор, matrix – двовимірний вектор (вектор векторів) та threedmatrix – тривимірний вектор (вектор матриць). Останній застосуємо для збереження матриць вагових коефіцієнтів шарів мережі:

```
using namespace std;

typedef vector<double> dvector;
typedef vector<dvector> matrix;
typedef vector<matrix> threedmatrix;
```

Для зручності виконання операцій уведення та виведення перевизначимо відповідні оператори:

```
ostream& operator<<(ostream& os, dvector d)
{
    for(int i=0;i<d.size();i++)
        os<<d[i]<<" ";
    return os;
}

ostream& operator<<(ostream& os, vector<int> d)
```

```
{
    for(int i=0;i<d.size();i++)
        os<<d[i]<<" ";
    return os;
}

istream& operator>>(istream& is, dvector &d)
{
    for(int i=0;i<d.size();i++)
        is>>d[i];
    return is;
}

istream& operator>>(istream& is, vector<int> &d)
{
    for(int i=0;i<d.size();i++)
        is>>d[i];
    return is;
}

ostream& operator<<(ostream& os, matrix d)
{
    for(int i=0;i<d.size();i++)
        os<<d[i]<<"\n";
    return os;
}

istream& operator>>(istream& is, matrix &d)
{
    for(int i=0;i<d.size();i++)
        is>>d[i];
    return is;
}
```

Визначимо інтерфейс класу mnn (Multilayer Neural Network). Приватними даними класу є вектор матриць вагових коефіцієнтів coeffs, кількість шарів мережі number_of_layers, вектор neurons, який містить кількість нейронів на кожному шарі. Приватний метод make_matrix використовується для створення матриць заданої розмірності.

Загальнодоступними даними класу є матриці еталонів, розділені на вхід (input_data) та вихід (output_data). Для зручності також зберігаються й відповідні їм нормалізовані дані norm_input_data та norm_output_data. Вектори input_min, input_max, output_min, output_max містять мінімальні та максимальні значення входу та виходу відповідно.

Набір загальнодоступних методів схарактеризуємо при їх визначенні:

```
class mnn
{
    threedmatrix coeffs;
    int number_of_layers;//>=3
    vector<int> neurons;
    matrix make_matrix(int rows, int cols);
public:
    matrix input_data, output_data,
           norm_input_data, norm_output_data;
    dvector input_min, input_max,
           output_min, output_max;

    mnn(vector<int> number_of_neurons);
    void show_network();
    bool load_network(const char *filename);
    bool save_network(const char *filename);
    void randomize_coeffs(double low, double high);
    bool load_data(const char *name);
    dvector forward(dvector in);
    double train_step(bool out);
};
```

Конструктор класу mnn в якості параметра приймає вектор number_of_neurons, кожний елемент якого містить додатну кількість нейронів на відповідному шарі мережі. Розмірність вектора визначає кількість шарів number_of_layers, яка для мережі глибинного навчання повинна бути не менше трьох. Якщо ці умови виконуються, створюється масив матриць коефіцієнтів зв'язків шарів мережі coeffs – їх кількість на одиницю менше за кількість шарів мережі, а розмірність визначається з урахуванням того, що на кожному шарі, крім останнього, додається нейрон зміщення:

```
mnn::mnn(vector<int> number_of_neurons)
{
    //check 1
    if(number_of_neurons.size()<3)
    {
        cerr<<"Number of layers should be more then 3\n";
        exit(1);
    }
    number_of_layers=number_of_neurons.size();
    neurons=number_of_neurons;
    //check 2
    for(int i=0;i<number_of_layers;i++)
        if(number_of_neurons[i]<1)
        {
            cerr<<"Number of neurons should be more then 0\n";
            exit(2);
```

```
        }
    //make data structures
    coeffs=vector<matrix>(number_of_layers-1);
    for(int i=0;i<coeffs.size();i++)
        coeffs[i]=make_matrix(number_of_neurons[i]+1,
number_of_neurons[i+1]);
}
```

Метод randomize_coeffs використовується для встановлення випадкових значень коефіцієнтів зв'язку в діапазоні від low до high:

```
void mnn::randomize_coeffs(double low, double high)
{
    for(int i=0;i<coeffs.size();i++)
        for(int j=0;j<coeffs[i].size();j++)
            for(int k=0;k<coeffs[i][j].size();k++)
                coeffs[i][j][k]+=((double)rand()/RAND_MAX)*(high-low)+low;
}
```

Допоміжний метод створення матриці заданої розмірності – rows рядків, cols стовпців:

```
matrix mnn::make_matrix(int rows, int cols)
{
    matrix res(rows);

    for(int i=0;i<rows;i++)
        res[i]=dvector(cols);
    return res;
}
```

Метод show_network використовується для відображення архітектури мережі, метод save_network – для її файлового збереження, а load_network – для завантаження:

```
void mnn::show_network()
{
    cout<<"Number of layers is "<<number_of_layers<<endl<<endl;
    for(int i=0;i<number_of_layers;i++)
        cout<<"Number of neurons on layer "<<(i+1)<<" is
"<<neurons[i]<<endl;
    cout<<endl;
    for(int i=0;i<coeffs.size();i++)
        cout<<"Connections between "<<(i+1)<<" and "<<(i+2)<<" layers:\n"
                <<coeffs[i]<<endl;
}

bool mnn::save_network(const char *filename)
```

```
{
    ofstream f(filename);

    if(!f)
        return false;
    f<<number_of_layers<<endl;
    f<<neurons<<endl<<endl;
    for(int i=0;i<coeffs.size();i++)
        f<<coeffs[i]<<endl;

    f<<endl<<input_min<<endl;
    f<<input_max<<endl;
    f<<output_min<<endl;
    f<<output_max<<endl;

    return true;
}

bool mnn::load_network(const char *filename)
{
    ifstream f(filename);

    if(!f)
        return false;
    if(!(f>>number_of_layers))
        return false;

    neurons=vector<int>(number_of_layers);
    if(!(f>>neurons))
        return false;

    coeffs=vector<matrix>(number_of_layers-1);
    for(int i=0;i<coeffs.size();i++)
    {
        coeffs[i]=make_matrix(neurons[i]+1, neurons[i+1]);
        if(!(f>>coeffs[i]))
            return false;
    }

    input_min=dvector(neurons[0]);
    input_max=dvector(neurons[0]);
    output_min=dvector(neurons[neurons.size()-1]);
    output_max=dvector(neurons[neurons.size()-1]);

    if(!(f>>input_min>>input_max>>output_min>>output_max))
        return false;

    return true;
}
```

Для завантаження еталонних даних застосовується метод load_data: крім власне завантаження – створення матриці еталонів вхідних (input_data) та

вихідних (output_data) даних, в ньому виконується знаходження мінімальних та максимальних значень входу (input_min, input_max) та виходу (output_min, output_max) та обчислюються їх нормалізовані у діапазоні [0; 1] значення (norm_input_data та norm_output_data):

```
bool mnn::load_data(const char *name)
{
    //File data:
    //n1 n2 n3: n1 - number of lines, n2 - input size, n3 - output size
    //e.g.
    //4 2 1
    //0 0 0
    //1 0 1
    //1 1 1
    //0 1 0

    ifstream f(name);
    if(!f)
    {
        cerr<<"Can not open file "<<name<<endl;
        return false;
    }
    int n1, n2, n3;
    f>>n1>>n2>>n3;
    if(n1<1)
    {
        cerr<<"Number of lines in file should be not less 1"<<endl;
        return false;
    }
    if(n2!=neurons[0])
    {
        cerr<<"Input size is not equal to number of neurons at input
layers"<<endl;
        return false;
    }
    if(n3!=neurons[neurons.size()-1])
    {
        cerr<<"Output size is not equal to number of neurons at output
layers"<<endl;
        return false;
    }
    input_data=matrix(n1);
    output_data=matrix(n1);
    for(int i=0;i<n1;i++)
    {
        input_data[i]=dvector(n2);
        output_data[i]=dvector(n3);
        f>>input_data[i]>>output_data[i];
    }
    //find max, min
```

```
    input_min=input_max=input_data[0];
    output_min=output_max=output_data[0];

    for(int i=1;i<n1;i++)
    {
        for(int j=0;j<n2;j++)
        {
            if(input_min[j]>input_data[i][j])
                input_min[j]=input_data[i][j];
            if(input_max[j]<input_data[i][j])
                input_max[j]=input_data[i][j];
        }
        for(int j=0;j<n3;j++)
        {
            if(output_min[j]>output_data[i][j])
                output_min[j]=output_data[i][j];
            if(output_max[j]<output_data[i][j])
                output_max[j]=output_data[i][j];
        }
    }
    norm_input_data=input_data;
    norm_output_data=output_data;

    for(int i=1;i<n1;i++)
        for(int j=0;j<n2;j++)
            norm_input_data[i][j]=(input_data[i][j]-input_min[j])
                    /(input_max[j]-input_min[j]);

    for(int i=1;i<n1;i++)
        for(int j=0;j<n3;j++)
            norm_output_data[i][j]=(output_data[i][j]-output_min[j])
                    /(output_max[j]-output_min[j]);
    return true;
}
```

В якості функції активації нейрону застосуємо сигмоїдальну:

```
double sigma(double s)
{
    return 1/(1+1/exp(s));
}
```

Оператор для обчислення добутку вектора на матрицю не просто реалізує скалярний добуток, а й застосовує до кожного компонента вектора результату функцію активації:

```
dvector operator*(dvector v, matrix m)
{
    dvector res(m[0].size());
```

```
    for(int i=0;i<res.size();i++)
    {
        for(int j=0;j<v.size();j++)
            res[i]+=v[j]*m[j][i];
        res[i]=sigma(res[i]);
    }
    return res;
}
```

Метод forward реалізує проходження сигналу від першого до останнього шару. Для цього вхідний вектор in спочатку нормалізується, далі до нього додається нейрон зміщення, який дорівнює одиниці, та знаходиться добуток розширеного вектора на матрицю вагових коефіцієнтів. Процедура повторюється, доки сигнал не досягне останнього шару. Отриманий на ньому вектор денормалізується та повертається:

```
dvector mnn::forward(dvector in)
{
    //check
    if(in.size()!=neurons[0])
    {
        cerr<<"Input size is not equal number of neurons on input
layer"<<endl;
        exit(3);
    }
    //normalize
    dvector temp=in;
    for(int i=0;i<in.size();i++)
        temp[i]=(in[i]-input_min[i])/(input_max[i]-input_min[i]);
    //cyclic forward
    for(int i=0;i<coeffs.size();i++)
    {
        //add 1 to end;
        temp.push_back(1);
        temp=temp*coeffs[i];
    }
    //denormalize
    for(int i=0;i<temp.size();i++)
        temp[i]=temp[i]*(output_max[i]-output_min[i])+output_min[i];

    return temp;
}
```

Допоміжний метод dist повертає евклідову відстань між векторами:

```
double dist(dvector v1, dvector v2)
{
```

```
    //check
    if(v1.size()!=v2.size())
    {
        cerr<<"Different vector sizes"<<endl;
        exit(8);
    }
    double res=0;
    for(int i=0;i<v1.size();i++)
        res+=pow(v1[i]-v2[i],2);
    return sqrt(res);
}
```

Метод train_step реалізує один крок навчання. Вважатимемо, що він відбувся, якщо в результаті зміни коефіцієнтів мережі відхилення еталонного виходу від обчисленого (загальна помилка) зменшилась:

```
double mnn::train_step(bool out)
{
    double total_error=0, new_total_error;

    for(int i=0;i<input_data.size();i++)
    {
        dvector res=forward(input_data[i]);
        total_error+=dist(res, output_data[i]);
    }
    if(out)
        cout<<"train_step: Total error is "<<total_error<<endl;

    if(out)
    {
        cout<<"\n\nBefore train\n\n";
        show_network();
    }
    threedmatrix old=coeffs;//save old
    do
    {
        new_total_error=0;
        //change coeffs
        randomize_coeffs(-0.1, 0.1);
        for(int i=0;i<input_data.size();i++)
        {
            dvector res=forward(input_data[i]);
            new_total_error+=dist(res, output_data[i]);
        }
        if(out)
        {
            cout<<"train_step: New Total error is
"<<new_total_error<<endl;

            cout<<"\n\nAfter train\n\n";
            show_network();
```

```
        }

        if(new_total_error>=total_error)
        {
            coeffs=old;
            if(out)
            {
                cout<<"\n\nShould be coeffs Before train:::\n\n";
                show_network();
            }
        }
    }while(new_total_error>=total_error);
    return new_total_error;
}
```

Проілюструємо роботу класу на прикладі ідентифікації набору даних «іриси Фішера» (https://en.wikipedia.org/wiki/Iris_flower_data_set). Структура мережі подана на рис. 3.1.

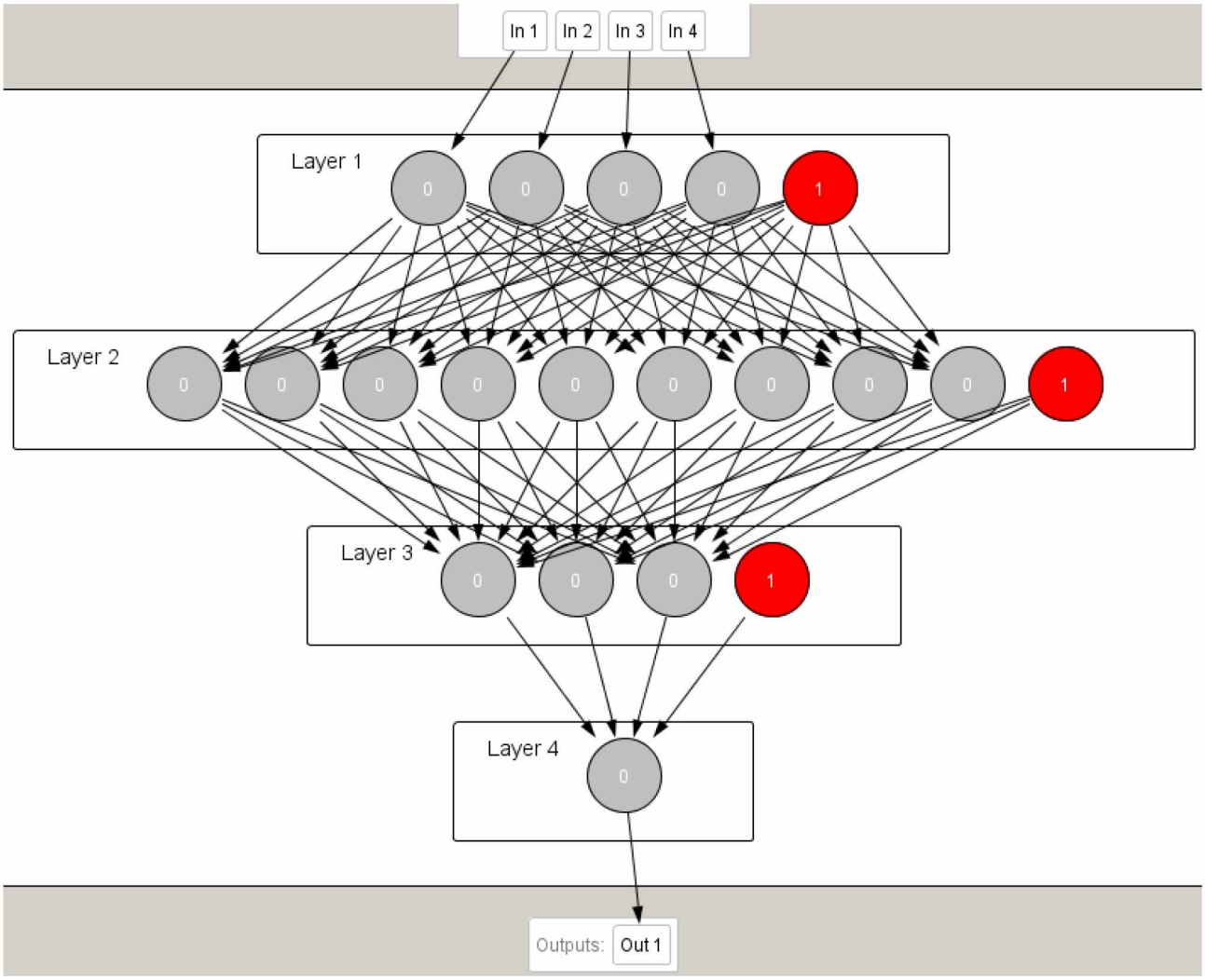

Рис. 3.1. Архітектура мережі для ідентифікації ірисів Фішера

Файл даних матиме наступний вигляд:

```
150 4 1

5.1   3.5   1.4   0.2   1
4.9   3.0   1.4   0.2   1
4.7   3.2   1.3   0.2   1
...
6.2   3.4   5.4   2.3   3
5.9   3.0   5.1   1.8   3
```

Дані передається у програму через командний рядок:

```
int main(int argc, char** argv) {

    if(argc!=2)
    {
        cout<<"Usage:\n\t"<<argv[0]<<" datafilename"<<endl;
        return 1;
    }
```

Задамо архітектуру мережі , створюємо за нею об'єкт класу mnn та завантажуємо дані:

```
    vector<int> number_of_neurons(4);
    number_of_neurons[0]=4;//0 - input layer
    number_of_neurons[1]=9;//1 - first hidden layer
    number_of_neurons[2]=3;//2 - second hidden layer
    number_of_neurons[3]=1;//3 - output layer with 1 neuron

    mnn three(number_of_neurons);
    three.show_network();
    three.load_data(argv[1]);
```

До початку навчання мережі обчислимо загальну помилку total_error як суму відстаней обчислених та еталнних векторів виходу:

```
    double total_error=0;
    for(int i=0;i<three.input_data.size();i++)
    {
        dvector res=three.forward(three.input_data[i]);
        total_error+=dist(res, three.output_data[i]);
    }
```

Якщо файл із збереженою мережею відсутній, виконується навчання до досягнення заданої точності, а на кожному кроці навчання мережа зберігається у файлі. Якщо файл із збереженою мережею наявний, уточнюємо, чи дійсно

користувач бажає нею скористатися:

```
    double newerr=total_error;

    if(!three.load_network("network.txt"))
        for(long i=0;newerr>0.25;i++)
        {
            newerr=three.train_step(false);
            cout<<"main: Total error is "<<newerr<<endl;
            three.save_network("network.txt");
        }
    else
    {
        cout<<"Do you with to retrain (1/0)? ";
        int ans;
        cin>>ans;
        if(ans==1)
            for(long i=0;newerr>0.25;i++)
            {
                newerr=three.train_step(false);
                cout<<"main: Total error is "<<newerr<<endl;
                three.save_network("network.txt");
            }
    }
```

Після завершення навчання знову оцінюємо помилку:

```
    total_error=0;
    cout<<"\n\nResults:\n\n";
    for(int i=0;i<three.input_data.size();i++)
    {
        dvector res=three.forward(three.input_data[i]);
        cout<<"Vector "<<three.input_data[i]<<", forwarded via network,
gives vector "
                <<res<<" - should be "<<three.output_data[i]<<endl;
        total_error+=dist(res, three.output_data[i]);
    }
    cout<<"main: Total error is "<<total_error<<endl;

    return 0;
}
```

3.4 Експериментальна перевірка розробленого методу нейромережевої ідентифікації об’єктів

Для поставленої задачі ідентифікації ірисів Фішера, кожен з яких був позначений відповідно числом 1, 2 та 3, початкова помилка склала 3.35894. З використанням розробленого ПЗ вона була зменшена до значення 2.04674 за 470

ітерацій. Визначені значення вагових коефіцієнтів:

а) зв’язку першого шару з другим:

```
3.74768 0.495993 -0.369194 -2.48554 0.914698 1.80775 0.90524 1.91384 -
1.90787
3.82049 -0.181792 2.16201 4.46752 0.0955214 -7.48551 -3.66681 4.08906 -
3.13042
-10.0625 3.84192 1.18527 -1.15013 -4.00952 3.10632 3.11545 -13.0914
2.79975
-13.6905 0.242969 -5.18144 -2.33239 -3.30966 3.61553 2.95562 -13.6947
1.99647
12.1436 0.718406 2.32753 0.551609 1.55943 -1.40306 -2.40168 15.1604 -
0.148058
```

б) зв’язку другого шару з третім:

```
-0.57533 -19.778 -3.50749
-4.1452 1.55784 -3.64133
0.792685 -5.71566 1.31487
0.65003 -3.66623 0.709116
8.61658 -1.94436 1.87522
-5.65876 8.96097 2.07444
-5.45783 9.33795 3.07354
2.943 -21.0196 -0.905777
-4.02994 2.00143 -1.58835
-3.37292 3.56752 -2.391
```

в) зв’язку третього шару з четвертим:

```
-11.9692
14.3229
-1.22024
0.00154389
```

Результати ідентифікації за визначеними коефіцієнтами подано нижче:

```
Vector 5.1 3.5 1.4 0.2, forwarded via network, gives 1.00002 - should be 1
Vector 4.9 3 1.4 0.2, forwarded via network, gives 1.00002 - should be 1
Vector 4.7 3.2 1.3 0.2, forwarded via network, gives 1.00002 - should be 1
Vector 4.6 3.1 1.5 0.2, forwarded via network, gives 1.00002 - should be 1
Vector 5 3.6 1.4 0.3, forwarded via network, gives 1.00002 - should be 1
Vector 5.4 3.9 1.7 0.4, forwarded via network, gives 1.00002 - should be 1
Vector 4.6 3.4 1.4 0.3, forwarded via network, gives 1.00002 - should be 1
Vector 5 3.4 1.5 0.2, forwarded via network, gives 1.00002 - should be 1
Vector 4.4 2.9 1.4 0.2, forwarded via network, gives 1.00003 - should be 1
Vector 4.9 3.1 1.5 0.1, forwarded via network, gives 1.00002 - should be 1
Vector 5.4 3.7 1.5 0.2, forwarded via network, gives 1.00002 - should be 1
Vector 4.8 3.4 1.6 0.2, forwarded via network, gives 1.00002 - should be 1
Vector 4.8 3 1.4 0.1, forwarded via network, gives 1.00002 - should be 1
Vector 4.3 3 1.1 0.1, forwarded via network, gives 1.00002 - should be 1
```

```
Vector 5.8 4 1.2 0.2, forwarded via network, gives 1.00001 - should be 1
Vector 5.7 4.4 1.5 0.4, forwarded via network, gives 1.00002 - should be 1
Vector 5.4 3.9 1.3 0.4, forwarded via network, gives 1.00002 - should be 1
Vector 5.1 3.5 1.4 0.3, forwarded via network, gives 1.00002 - should be 1
Vector 5.7 3.8 1.7 0.3, forwarded via network, gives 1.00002 - should be 1
Vector 5.1 3.8 1.5 0.3, forwarded via network, gives 1.00002 - should be 1
Vector 5.4 3.4 1.7 0.2, forwarded via network, gives 1.00002 - should be 1
Vector 5.1 3.7 1.5 0.4, forwarded via network, gives 1.00002 - should be 1
Vector 4.6 3.6 1 0.2, forwarded via network, gives 1.00001 - should be 1
Vector 5.1 3.3 1.7 0.5, forwarded via network, gives 1.00005 - should be 1
Vector 4.8 3.4 1.9 0.2, forwarded via network, gives 1.00003 - should be 1
Vector 5 3 1.6 0.2, forwarded via network, gives 1.00003 - should be 1
Vector 5 3.4 1.6 0.4, forwarded via network, gives 1.00003 - should be 1
Vector 5.2 3.5 1.5 0.2, forwarded via network, gives 1.00002 - should be 1
Vector 5.2 3.4 1.4 0.2, forwarded via network, gives 1.00002 - should be 1
Vector 4.7 3.2 1.6 0.2, forwarded via network, gives 1.00002 - should be 1
Vector 4.8 3.1 1.6 0.2, forwarded via network, gives 1.00002 - should be 1
Vector 5.4 3.4 1.5 0.4, forwarded via network, gives 1.00002 - should be 1
Vector 5.2 4.1 1.5 0.1, forwarded via network, gives 1.00001 - should be 1
Vector 5.5 4.2 1.4 0.2, forwarded via network, gives 1.00001 - should be 1
Vector 4.9 3.1 1.5 0.2, forwarded via network, gives 1.00002 - should be 1
Vector 5 3.2 1.2 0.2, forwarded via network, gives 1.00002 - should be 1
Vector 5.5 3.5 1.3 0.2, forwarded via network, gives 1.00002 - should be 1
Vector 4.9 3.6 1.4 0.1, forwarded via network, gives 1.00002 - should be 1
Vector 4.4 3 1.3 0.2, forwarded via network, gives 1.00002 - should be 1
Vector 5.1 3.4 1.5 0.2, forwarded via network, gives 1.00002 - should be 1
Vector 5 3.5 1.3 0.3, forwarded via network, gives 1.00002 - should be 1
Vector 4.5 2.3 1.3 0.3, forwarded via network, gives 1.00059 - should be 1
Vector 4.4 3.2 1.3 0.2, forwarded via network, gives 1.00002 - should be 1
Vector 5 3.5 1.6 0.6, forwarded via network, gives 1.00004 - should be 1
Vector 5.1 3.8 1.9 0.4, forwarded via network, gives 1.00003 - should be 1
Vector 4.8 3 1.4 0.3, forwarded via network, gives 1.00003 - should be 1
Vector 5.1 3.8 1.6 0.2, forwarded via network, gives 1.00002 - should be 1
Vector 4.6 3.2 1.4 0.2, forwarded via network, gives 1.00002 - should be 1
Vector 5.3 3.7 1.5 0.2, forwarded via network, gives 1.00002 - should be 1
Vector 5 3.3 1.4 0.2, forwarded via network, gives 1.00002 - should be 1
Vector 7 3.2 4.7 1.4, forwarded via network, gives 1.99998 - should be 2
Vector 6.4 3.2 4.5 1.5, forwarded via network, gives 2.00007 - should be 2
Vector 6.9 3.1 4.9 1.5, forwarded via network, gives 1.99954 - should be 2
Vector 5.5 2.3 4 1.3, forwarded via network, gives 1.99944 - should be 2
Vector 6.5 2.8 4.6 1.5, forwarded via network, gives 1.99911 - should be 2
Vector 5.7 2.8 4.5 1.3, forwarded via network, gives 1.99998 - should be 2
Vector 6.3 3.3 4.7 1.6, forwarded via network, gives 1.9999 - should be 2
Vector 4.9 2.4 3.3 1, forwarded via network, gives 1.99994 - should be 2
Vector 6.6 2.9 4.6 1.3, forwarded via network, gives 1.99989 - should be 2
Vector 5.2 2.7 3.9 1.4, forwarded via network, gives 2.00007 - should be 2
Vector 5 2 3.5 1, forwarded via network, gives 1.99977 - should be 2
Vector 5.9 3 4.2 1.5, forwarded via network, gives 2.00008 - should be 2
Vector 6 2.2 4 1, forwarded via network, gives 1.99955 - should be 2
Vector 6.1 2.9 4.7 1.4, forwarded via network, gives 1.9997 - should be 2
Vector 5.6 2.9 3.6 1.3, forwarded via network, gives 1.99964 - should be 2
Vector 6.7 3.1 4.4 1.4, forwarded via network, gives 2.00003 - should be 2
Vector 5.6 3 4.5 1.5, forwarded via network, gives 1.99979 - should be 2
```

```
Vector 5.8 2.7 4.1 1, forwarded via network, gives 2 - should be 2
Vector 6.2 2.2 4.5 1.5, forwarded via network, gives 2.00002 - should be 2
Vector 5.6 2.5 3.9 1.1, forwarded via network, gives 2.00004 - should be 2
Vector 5.9 3.2 4.8 1.8, forwarded via network, gives 2.01833 - should be 2
Vector 6.1 2.8 4 1.3, forwarded via network, gives 2.00008 - should be 2
Vector 6.3 2.5 4.9 1.5, forwarded via network, gives 2.00349 - should be 2
Vector 6.1 2.8 4.7 1.2, forwarded via network, gives 1.99995 - should be 2
Vector 6.4 2.9 4.3 1.3, forwarded via network, gives 2.00003 - should be 2
Vector 6.6 3 4.4 1.4, forwarded via network, gives 2 - should be 2
Vector 6.8 2.8 4.8 1.4, forwarded via network, gives 1.99923 - should be 2
Vector 6.7 3 5 1.7, forwarded via network, gives 2.00197 - should be 2
Vector 6 2.9 4.5 1.5, forwarded via network, gives 1.99958 - should be 2
Vector 5.7 2.6 3.5 1, forwarded via network, gives 1.99949 - should be 2
Vector 5.5 2.4 3.8 1.1, forwarded via network, gives 1.99999 - should be 2
Vector 5.5 2.4 3.7 1, forwarded via network, gives 2 - should be 2
Vector 5.8 2.7 3.9 1.2, forwarded via network, gives 2.0001 - should be 2
Vector 6 2.7 5.1 1.6, forwarded via network, gives 2.99995 - should be 2
Vector 5.4 3 4.5 1.5, forwarded via network, gives 1.99974 - should be 2
Vector 6 3.4 4.5 1.6, forwarded via network, gives 1.9999 - should be 2
Vector 6.7 3.1 4.7 1.5, forwarded via network, gives 1.9998 - should be 2
Vector 6.3 2.3 4.4 1.3, forwarded via network, gives 1.99892 - should be 2
Vector 5.6 3 4.1 1.3, forwarded via network, gives 1.99999 - should be 2
Vector 5.5 2.5 4 1.3, forwarded via network, gives 1.99986 - should be 2
Vector 5.5 2.6 4.4 1.2, forwarded via network, gives 1.99993 - should be 2
Vector 6.1 3 4.6 1.4, forwarded via network, gives 1.99999 - should be 2
Vector 5.8 2.6 4 1.2, forwarded via network, gives 2.00003 - should be 2
Vector 5 2.3 3.3 1, forwarded via network, gives 2.00002 - should be 2
Vector 5.6 2.7 4.2 1.3, forwarded via network, gives 2.00003 - should be 2
Vector 5.7 3 4.2 1.2, forwarded via network, gives 1.99983 - should be 2
Vector 5.7 2.9 4.2 1.3, forwarded via network, gives 2.00016 - should be 2
Vector 6.2 2.9 4.3 1.3, forwarded via network, gives 2.00008 - should be 2
Vector 5.1 2.5 3 1.1, forwarded via network, gives 1.99938 - should be 2
Vector 5.7 2.8 4.1 1.3, forwarded via network, gives 2.00014 - should be 2
Vector 6.3 3.3 6 2.5, forwarded via network, gives 3 - should be 3
Vector 5.8 2.7 5.1 1.9, forwarded via network, gives 3 - should be 3
Vector 7.1 3 5.9 2.1, forwarded via network, gives 3 - should be 3
Vector 6.3 2.9 5.6 1.8, forwarded via network, gives 3 - should be 3
Vector 6.5 3 5.8 2.2, forwarded via network, gives 3 - should be 3
Vector 7.6 3 6.6 2.1, forwarded via network, gives 3 - should be 3
Vector 4.9 2.5 4.5 1.7, forwarded via network, gives 3 - should be 3
Vector 7.3 2.9 6.3 1.8, forwarded via network, gives 3 - should be 3
Vector 6.7 2.5 5.8 1.8, forwarded via network, gives 3 - should be 3
Vector 7.2 3.6 6.1 2.5, forwarded via network, gives 3 - should be 3
Vector 6.5 3.2 5.1 2, forwarded via network, gives 3 - should be 3
Vector 6.4 2.7 5.3 1.9, forwarded via network, gives 3 - should be 3
Vector 6.8 3 5.5 2.1, forwarded via network, gives 3 - should be 3
Vector 5.7 2.5 5 2, forwarded via network, gives 3 - should be 3
Vector 5.8 2.8 5.1 2.4, forwarded via network, gives 3 - should be 3
Vector 6.4 3.2 5.3 2.3, forwarded via network, gives 3 - should be 3
Vector 6.5 3 5.5 1.8, forwarded via network, gives 3 - should be 3
Vector 7.7 3.8 6.7 2.2, forwarded via network, gives 3 - should be 3
Vector 7.7 2.6 6.9 2.3, forwarded via network, gives 3 - should be 3
Vector 6 2.2 5 1.5, forwarded via network, gives 3 - should be 3
```

```
Vector 6.9 3.2 5.7 2.3, forwarded via network, gives 3 - should be 3
Vector 5.6 2.8 4.9 2, forwarded via network, gives 3 - should be 3
Vector 7.7 2.8 6.7 2, forwarded via network, gives 3 - should be 3
Vector 6.3 2.7 4.9 1.8, forwarded via network, gives 3 - should be 3
Vector 6.7 3.3 5.7 2.1, forwarded via network, gives 3 - should be 3
Vector 7.2 3.2 6 1.8, forwarded via network, gives 3 - should be 3
Vector 6.2 2.8 4.8 1.8, forwarded via network, gives 3 - should be 3
Vector 6.1 3 4.9 1.8, forwarded via network, gives 2.99995 - should be 3
Vector 6.4 2.8 5.6 2.1, forwarded via network, gives 3 - should be 3
Vector 7.2 3 5.8 1.6, forwarded via network, gives 2.99957 - should be 3
Vector 7.4 2.8 6.1 1.9, forwarded via network, gives 3 - should be 3
Vector 7.9 3.8 6.4 2, forwarded via network, gives 3 - should be 3
Vector 6.4 2.8 5.6 2.2, forwarded via network, gives 3 - should be 3
Vector 6.3 2.8 5.1 1.5, forwarded via network, gives 1.99682 - should be 3
Vector 6.1 2.6 5.6 1.4, forwarded via network, gives 2.99982 - should be 3
Vector 7.7 3 6.1 2.3, forwarded via network, gives 3 - should be 3
Vector 6.3 3.4 5.6 2.4, forwarded via network, gives 3 - should be 3
Vector 6.4 3.1 5.5 1.8, forwarded via network, gives 3 - should be 3
Vector 6 3 4.8 1.8, forwarded via network, gives 2.99167 - should be 3
Vector 6.9 3.1 5.4 2.1, forwarded via network, gives 3 - should be 3
Vector 6.7 3.1 5.6 2.4, forwarded via network, gives 3 - should be 3
Vector 6.9 3.1 5.1 2.3, forwarded via network, gives 3 - should be 3
Vector 5.8 2.7 5.1 1.9, forwarded via network, gives 3 - should be 3
Vector 6.8 3.2 5.9 2.3, forwarded via network, gives 3 - should be 3
Vector 6.7 3.3 5.7 2.5, forwarded via network, gives 3 - should be 3
Vector 6.7 3 5.2 2.3, forwarded via network, gives 3 - should be 3
Vector 6.3 2.5 5 1.9, forwarded via network, gives 3 - should be 3
Vector 6.5 3 5.2 2, forwarded via network, gives 3 - should be 3
Vector 6.2 3.4 5.4 2.3, forwarded via network, gives 3 - should be 3
Vector 5.9 3 5.1 1.8, forwarded via network, gives 3 - should be 3
```

Отриманий результат є суттєво кращим у порівнянні з використанням традиційного АЗП або інших методів оптимізації [19], проте час добору коефіцієнтів при виконанні на CPU є суттєво більшим, що й зумовлює доцільність застосування GPU в якості нейрочіпу.

Висновки до розділу 3

1. Розглянуто постановку задачі нейромережевої апроксимації як способу ідентифікації об’єктів з використанням теореми Цибенко.

2. Показано доцільність вибору в якості архітектури штучної нейронної мережі для апроксимації будь-якої неперервної функції багатошарового перцептрону.

3. Розроблено метод визначення коефіцієнтів зв’язку шарів мережі для

штучних нейронних мереж глибинного навчання з кількістю прихованих шарів більше одного. На основі обраного векторно-матричного подання спроектовано програмне забезпечення для реалізації методу. Показано високий рівень точності ідентифікації об’єктів порівняно з алгоритмом зворотного поширення помилки. Визначено доцільність використання GPU для реалізації розробленого програмного забезпечення.

ВИСНОВКИ

1. Визначено основні проблеми, що виникають при використанні методу зворотного поширення помилки (не ефективний у разі, коли значення похідних по різним вагам нейромережі суттєво відрізняються; не дозволяє отримати швидку збіжність процесу навчання; виникає висока ймовірність виникнення ефекту перенавчання тощо). На прикладі штучних нейронних мереж з різними функціями активації показано доцільність матричного подання мережі для визначення вагових коефіцієнтів її міжшарових з'єднань.

2. Розглянуто підхід до прискорення алгоритмів навчання з використанням нової технологічної бази – нейрочіпів. Визначено основну відмінність нейрочіпів від інших процесорів – забезпечення високого паралелізму обчислень за рахунок застосування спеціалізованого нейромережевого логічного базису або конкретних архітектурних рішень. Показано, що використання можливості представлення нейромережевих алгоритмів для реалізації на нейромережевому логічному базисі є основною передумовою різкого збільшення швидкості реалізації алгоритмів навчання штучних нейронних мереж. Обґрунтовано доцільність використання апаратно-програмного комплексу CUDA в якості нейрочіпу.

3. Розглянуто подання задачі ідентифікації об'єктів як постановку задачі нейромережевої апроксимації, показано доцільність вибору в якості архітектури штучної нейронної мережі для апроксимації будь-якої неперервної функції багатошарового перцептрону. Розроблено метод визначення коефіцієнтів зв'язку шарів мережі для штучних нейронних мереж глибинного навчання з кількістю прихованих шарів більше одного. На основі обраного векторно-матричного подання спроектовано програмне забезпечення для реалізації методу. Показано високий рівень точності ідентифікації об'єктів порівняно з алгоритмом зворотного поширення помилки. Визначено доцільність використання GPU для реалізації розробленого програмного забезпечення.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Belgian researchers develop desktop supercomputer [Электронный ресурс] // FASTRA. – [2008]. – Режим доступа : http://fastra.ua.ac.be/en/index.html
2. Beliczynski B. Incremental approximation by one-hidden-layer neural networks: discrete functions rapprochement / Bartlomiej Beliczynski // Proceedings of the IEEE International Symposium on Industrial Electronics, 1996. ISIE '96. – Vol. 1. – P. 392-397.
3. Beliczyński B. Przyrostowa aproksymacja funkcji za pomocą sieci neuronowych / Bartłomiej Beliczyński // Prace Naukowe Politechniki Warszawskiej. Elektryka. – 2000, z. 112. – S. 3-77.
4. CUDA C Best Practices Guide. – Version 3.2. – NVIDIA Corporation, 2010. – http://developer.download.nvidia.com/compute/cuda/3_2/toolkit/docs/CUDA_C_Best_Practices_Guide.pdf.
5. CUDA CUBLAS Library – PG-05326-032_V02 August, 2010 – http://www.cs.cmu.edu/afs/cs/academic/class/15668-s11/www/cuda-doc/CUBLAS_Library.pdf
6. Cybenko G. Approximation by Superposition of a Sigmoidal Function / G. Cybenko // Mathematics of Control, Signals, and System. – 1989. – №2. – P. 303-314.
7. DSP и RISC объединились [Електронний ресурс] / Дмитрий Фомин, Владимир Черников, Павел Виксне, Михаил Яфраков, Павел Шевченко // Открытые системы. – 1999. – № 5-6. – Режим доступу : http://www.osp.ru/os/1999/05-06/179822/
8. Fatica M. CUDA for High Performance Computing / Fatica M. // Materials of HPC-NA Workshop 3, January 2009.
9. Fletcher R. Function minimization by conjugate gradients / R. Fletcher, C. M. Reeves // The Computer Journal. – 1964. – Vol. 7. – P. 149–154.
10. Hagan M. T. Training feedforward networks with the Marquardt algorithm / Martin T. Hagan, Mohammad B. Menhaj // IEEE Transactions on Neural

Networks. – 1994, Vol. 5. – No. 6. – P. 989–993.

11. Haykin S. Neural Network. A Comprehensive Foundation. Second Edition / Simon Haykin. – Singapore : Pearson Education, 2005. – 823 p.
12. Hornik K. Multilayer Feedforward Networks are Universal Approximators / Kur Hornik, Maxwell Stinchcomb and Halber White // Neural Networks. – 1989. – Vol. 2. – P. 359–366.
13. Kohonen T. Self-Organization and Associative Memory. 2nd ed. / Kohonen T. – Berlin : Springer-Verlag, 1987.
14. Łukianiuk A. Wykorzystanie algorytmów sieci neuronowych w aplikacjach bazy danych Oracle [Електронний ресурс] / Andrzej Łukianiuk, Bartłomiej Beliczyński // V Konferencja PLOUG, Zakopane, Październik. – 1999. – 11 p. – Режим доступу : http://www.ploug.org.pl/konf_99/pdf/17.pdf
15. New NVIDIA Tesla GPUs Reduce Cost Of Supercomputing By A Factor Of 10 [Электронный ресурс] // NVIDIA – World Leader in Visual Computing Technologies. – [2009]. – Режим доступа : http://www.nvidia.com/object/io_1258360868914.html
16. Park J. Universal approximation using radial basis function networks / J. Park, I. W. Sandberg // Neural Computing. – 1991. – Vol. 3. – P. 246–257.
17. Rumelhart D. E. Learning Internal Representation by Error Propagation / D. E. Rumelhart, G. E. Hinton, and R. J. Williams // Parrallel Distributed Processing. – Vol. I. Foundations / Eds. : D. E. Rumelhart and J. L. McClelland. – Cambridge : MIT Press, 1986. – P. 318–362.
18. Rumelhart D. E. Learning representation by back-propagation errors / David E. Rumelhart, Geoffrey E. Hinton & Ronald J. Williams // Nature. – 1986. – Vol. 323. – 9 October. – P. 533–536.
19. Semerikov S. Computer Simulation of Neural Networks Using Spreadsheets: Dr. Anderson, Welcome Back [Electronic resource] / Serhiy Semerikov, Illia Teplytskyi, Yuliia Yechkalo, Oksana Markova, Vladimir Soloviev, Arnold Kiv // ICTERI 2019: ICT in Education, Research and Industrial Applications. Integration, Harmonization and Knowledge Transfer : Proceedings of the 15th

International Conference on ICT in Education, Research and Industrial Applications. Integration, Harmonization and Knowledge Transfer. Volume II: Workshops. Kherson, Ukraine, June 12-15, 2019 / Edited by : Vadim Ermolayev, Frédéric Mallet, Vitaliy Yakovyna, Vyacheslav Kharchenko, Vitaliy Kobets, Artur Korniłowicz, Hennadiy Kravtsov, Mykola Nikitchenko, Serhiy Semerikov, Aleksander Spivakovsky. – (CEUR Workshop Proceedings, Vol. 2393). – P. 833-848. – Access mode : http://ceur-ws.org/Vol-2393/paper_348.pdf

20. Stone J. E. Accelerating molecular modeling applications with graphics processors / John E. Stone, James C. Phillips, Peter L. Freddolino, David J. Hardy, Leonardo G. Trabuco, Klaus Schulten // Journal of Computational Chemistry. – 2007. – Vol. 28. – P. 2618–2640.
21. Vogl T. P. Accelerating the convergence of the back-propagation method / T. P. Vogl, J. K. Mangis, A. K. Rigler, W. T. Zink, D. L. Alkon // Biological Cybernetics. – 1988. – Vol. 59. – P. 256–264.
22. Wasserman P. D. Advanced Method in Neural Computing / Wasserman P. D. – New York : Van Nostrand Reinhold, 1993.
23. Weierstrass K. Uber die analytische Darstellbarkeit sogenannter willkürlicher Funktionen einer reellen veränderlichen / Weierstrass K. // Sitzungsberichte der Akademie der Wissenschaften, Berlin, 1985, p. 633-639, 789-905.
24. Werbos P. J. Beyond regression: New tools for prediction and analysis in the behavioral sciences : PhD Thesis / Werbos P. J. ; Harvard University, – Cambridge, 1974.
25. Барцев С. И. Принцип двойственности в организации адаптивных сетей обработки информации / Барцев С. И., Гилев С. Е., Охонин В. А. // Динамика химических и биологических систем. – Новосибирск : Паука. Сиб. отд-ние, 1989. – С. 6–55.
26. Боресков А.В. Основы работы с технологией CUDA : учеб. пособие / А. В. Боресков, А. А. Харламов. - М. : ДМК Пресс, 2010. – 230 с.
27. Боресков А. В. Основы работы с технологией CUDA / А. В. Боресков, А. А. Харламов. – М. : ДМК Пресс, 2010. – 232 с.

28. Вороновский Г. К. Генетические алгоритмы, искусственные нейронные сети и проблемы виртуальной реальности / Вороновский Г. К., Махотило К. В., Петрашев С. Н., Сергеев С. А. – Харьков : Основа, 1997. – 112 с.
29. Галушкин А. И. Некоторые исторические аспекты развития элементной базы вычислительных систем с массовым параллелизмом (80- и 90- годы) / А. И. Галушкин // Нейрокомпьютер. – 2000. – № 1. – С. 68-82.
30. Довженко А. Ю. Параллельная нейронная сеть с удаленным доступом на базе распределенного кластера ЭВМ / А. Ю. Довженко, С. А. Крашаков // Тезисы докл. II международн. симп. «Компьютерное обеспечение химических исследований» (Москва, 22-23 мая 2001 г.) и III Всерос. школы-конф. по квантовой и вычисл. химии им. В. А. Фока. – 2001. – С. 52-53.
31. Зенкевич О. Конечные элементы и аппроксимация / О. Зенкевич, К. Морган. – М. : Мир, 1986. – 318 с.
32. Комарцова Л. Г. Нейрокомпьютеры : учебное пособие для вузов. – 2-е изд., перераб и доп. / Комарцова Л. Г., Максимов А. В. – М. : Изд-во МГТУ им. Н. Э. Баумана, 2004. – 400 с.
33. Круг П. Г. Нейронные сети и нейрокомпьютеры : учебное пособие по курсу «Микропроцессоры» / П. Г. Круг. – М. : Издательство МЭИ, 2002. – 176 с.
34. Круглов В. В. Искусственные нейронные сети. Теория и практика / Круглов В. В., Борисов В. В. – М. : Горячая линия-Телеком, 2001. – 382 с.
35. Круглов В. В. Искусственные нейронные сети. Теория и практика. – 2-е издание, стереотип. / В. В. Круглов, В. В. Борисов. – М. : Горячая линия-Телеком, 2002. – 382 с.
36. Кульчин Ю. Н. Нейро-итерационный алгоритм томографической реконструкции распределенных физических полей в волоконно-оптических измерительных системах / Ю. Н. Кульчин, Б. С. Ноткин, В. А. Седов // Компьютерная оптика. – 2009. – Т. 33, № 4. – С. 446-455.
37. Купін А. І. Узгоджене інтелектуальне керування стадіями технологічного процесу збагачення магнетитових кварцитів в умовах невизначеності :

автореф. дис. ... д-ра техн. наук : 05.13.07 / Купін Андрій Іванович; Криворізький технічний університет. – Кривий Ріг, 2010. – 36 с.

38. Маккаллок У. С. Логическое исчисление идей, относящихся к нервной активности / Уоррен С. Мак-Каллок и Вальтер Питтс // Автоматы / Под. ред. К. Э. Шеннона и Дж. Маккарти. – М. : Издательство иностранной литературы, 1956. – С. 363–384.
39. Медведев В. С. Нейронные сети. MATLAB 6 / В. С. Медведев, В. Г. Потёмкин ; под общ. ред. к. т. н. В. Г. Потёмкина. – М. : ДИАЛОГ-МИФИ, 2002. – 496 с. – (Пакеты прикладных программ ; кн. 4).
40. Мохов В. А. Разработка алгоритмов прямого синтеза аппроксимирующих искусственных нейронных сетей : диссертация ... кандидата технических наук : 05.13.11 – математическое и программное обеспечение вычислительных машин, комплексов и компьютерных сетей / Мохов Василий Александрович ; Федеральное агентство по образованию РФ, Южно-Российский региональный центр информатизации Ростовского государственного университета. – Ростов-на-Дону, 2005. – 179 с.
41. Мясищев А. А. Сопоставление производительностей GPU и CPU для матричного умножения с двойной точностью / А. А. Мясищев // Теорія та методика навчання математики, фізики, інформатики : збірник наукових праць. Випуск X : в 3-х томах. – Кривий Ріг : Видавничий відділ НМетАУ, 2012. – Т. 3 : Теорія та методика навчання інформатики. – С. 99–107.
42. Нейронные сети и аппроксимация функций : учеб. пособие / Н. П. Абовский [и др.] ; М-во образования Рос. Федерации, Краснояр. гос. архитектур.-строит. акад. – Красноярск : КрасГАСА, 2002. – 133 с.
43. Николаевич В. С-ускоритель [Электронный ресурс] / Николаевич В. // Компьютерра. – 2004. – №24. – Режим доступа : http://old.computerra.ru/print/33955/
44. Уоссермен Ф. Нейрокомпьютерная техника / Уоссермен Ф. – М. : Мир, 1992. – 240 с.