Міністерство освіти і науки України
Державний вищий навчальний заклад
«Приазовський державний технічний університет»
Біомедичної інженерії

Верескун М.В.

НАВЧАННЯ МАШИН ТА ШТУЧНИЙ ІНТЕЛЕКТ

Методичні вказівки
по виконанню лабораторних робіт
з дисципліни «Технлогії програмування»
частина 2 «Навчання машин та штучний інтелект»
для студентів напрямку 163 «Біомедична інженерія»
для всіх форм навчання

Розроблено в рамках проекту «Erasmus+ (CBHE) BioArt: «Інноваційна мультидисциплінарна навчальна програма зі штучних імплантів для біоінженерії для рівнів бакалавр та магістр»
Developed in the frame of project «Erasmus+ (CBHE) BioArt: Innovative Multidisciplinary Curriculum in Artificial Implants for Bio-Engineering BSc/MSc Degrees» (586114-EPP- 1-2017- 1-ES- EPPKA2-CBHE- JP)

Маріуполь
2019

УДК 004.032.26(076)

Навчання машин та штучний інтелект: методичні вказівки до виконання лабораторних робіт з дисципліни «Технології програмування» для студентів напряму 163 «Біомедична інженерія» для всіх форм навчання / уклад. М.В. Верескун. – Маріуполь : ПДТУ, 2019. – 61 с.

У методичних вказівках наведені: тема і мета кожної лабораторної роботи; загальні теоретичні відомості до виконання кожної роботи; перелік необхідних термінів і понять; завдання для виконання робіт; зміст звіту; в завершенні наведено перелік рекомендованої літератури.

Укладачі М.В. Верескун, докт. екон. еаук, доцент

Рецензент Д. С. Міроненко, канд. техн. наук, доцент

Рекомендовано
на засіданні кафедри інформатики,
протокол № 21 від 24 червня 2019 р.

Затверджено
методичною комісією факультету інформаційних технологій,
протокол № 10 від 24 червня 2019 р.

ЗМІСТ

ВСТУП

Курс «Технології прогамування» ч. 2 «Навчання машин та штучний інтелект» призначений для студентів які навчаються з анапрямом 163 «Біомедична інжененрія». Метою дисципліни є формування у майбутніх фахівців основ теоретичних знань і практичних навичок роботи в області навчання машин, функціонування і використання технологій штучних нейронних мереж. У рамках дисципліни розглядаються теоретичні основи побудови штучних нейронних мереж, а також практичні питання використання нейромережевих технологій для вирішення різних практичних завдань.

Курс «Технології програмування», ч. 2 «Навчання машин» будується таким чином, щоб поєднувати теоретичну підготовку в області технологій навчання машин та штучного інтелекту з вивченням спеціалізованих програмних продуктів.

Лабораторні роботи призначені для закріплення знань, отриманих студентом на лекціях і в результаті самостійного вивчення матеріалу. Кожна лабораторна робота розрахована на 2 години виконання студентом на занятті.

Лабораторна робота № 1.
ОСНОВИ ПРОГРАМУВАННЯ В СИСТЕМІ MATLAB

Мета роботи: вивчення основ проблемно-орієнтованої системи програмування MATLAB, що забезпечує роботу з масивами строкових і числових даних, матрицями, структурами, класами і комірками за допомогою матричних операцій, функцій, різноманітних керуючих структур і засобів розширення системи, а також придбання навичок застосування командно-графічного інтерфейсу системи для побудови і дослідження нейронних мереж різної архітектури.

1.1 Загальні теоретичні відомості

1.1.1 Вступ

Потужна інструментальна система MATLAB забезпечує процедурне, операторний, функціональне, логічне, структурний, об'єктно-орієнтоване і візуальне (засобами пакета Simulink) програмування. Вона базується на математико-орієнтованої мови надвисокого рівня, який спрощує запис алгоритмів і відкриває нові методи їх створення.

Мова системи MATLAB за своєю структурою нагадує популярний командний мову Бейсік. Команди мови виконуються в режимі інтерпретації. З його допомогою можна створювати текстові модуліфункціі і модулі-сценарії. Файли, де зберігаються такі модулі, мають розширення *.m і називаються М-файлами, а що знаходяться в них функції – М-функціями. В системі є величезна бібліотека М-функцій в текстовому форматі, які можна модифікувати для досягнення бажаних цілей. Користувач може створювати власні М-функції і включати їх в систему за словниковим принципом.

1.1.2 Базові засоби мови програмування

Система MATLAB може використовуватися в двох режимах: в режимі безпосереднього рахунку (командний режим) і в режимі програмування.

У командному режимі користувач послідовно вводить команди вхідного мови і отримує відповідь. За допомогою цього режиму можна вирішити безліч математичних задач різної складності.

Однак при вирішенні серйозних завдань виникає необхідність збереження використовуваних послідовностей обчислень, а також їх подальших модифікацій. Для вирішення цього завдання система MATLAB має в своєму складі потужний мову програмування високого рівня.

Програмами в системі MATLAB є файли текстового формату з розширенням m, що містять запис програм у вигляді програмних кодів.

Для редагування файлів програм може використовуватися будь-який текстовий редактор, а також спеціальний багатовіконний редактор. Редактор програм системи MATLAB має наступні можливості:

- кольорове підсвічування синтаксису, дозволяє виявити помилки;
- синтаксичний контроль на стадії підготовки М-файлу;
- установка точок переривання при інтерпретації команд;
- автоматична нумерація рядків програми для видачі повідомлень.

Мова програмування системи MATLAB має наступні засоби для побудови команд і написання М-файлів:

1) дані різного типу: **double, numeric, char, cell, array;**
2) константи і змінні: **25, pi, eps, 'Hello', ans, m, n;**
3) оператори, включаючи оператори математичних виразів: +,-,*;
4) вбудовані команди і функції: **help, clear, plot, sin, cos;**
5) функції користувача: **func, map, draw, paint, neuron;**
6) керуючі структури: **if, for, while, switch, try, catch, end;**
7) системні оператори і функції: **realmin, realmax, Inf, Nan;**
8) засоби розширення мови: пакет **NNT**, пакет **Simulink**.

1.1.3 М-файли сценаріїв і функцій

М-файли системи MATLAB поділяються на два класи:

- файли-сценарії, що не мають вхідних параметрів;
- файлы-функції, що мають вхідні параметри.

Файл-сценарій або Script-файл не має списку вхідних параметрів. Він використовує глобальні змінні, тобто такі змінні, значення яких можуть бути змінені в будь-який момент сеансу роботи і в будь-якому місці програми. Для запуску файлу-сценарію з командного рядка MATLAB досить вказати його ім'я в цьому рядку.

Файл-сценарій має наступну структуру:

% Основний комментарій – один рядок (обов'язковий)

% Додатковий комментарій – будь-яке число рядків (не обов'язковий)

Тіло файлу з будь-якими виразами, командами і керуючими структурами.

Основний коментар виводиться при виконанні команд **lookfor** і **help імя_каталога**. Повний коментар виводиться при виконанні команди **help ім'я файлу**, причому висновок виробляється до першої порожнього рядка.

Файл-функція відрізняється від файлу-сценарію насамперед тим, що створена ним функція має вхідні параметри, список яких вказується в круглих дужках. Використовувані в файлах-функціях змінні і імена параметрів є локальними змінними, зміна значень яких в тілі функції не впливає на значення, які ті ж самі змінні можуть мати за межами функції.

Файл-функція має наступну структуру:

function var=f_name (Список_параметрів_передаваемих_ значень)

%Основний комментарій – один рядок (обов'язковий)

%Додатковий комментарій – будь-яке число рядків (не обов'язковий)

Тіло файлу з будь-якими виразами, командами і керуючими структурами.

var=вираз

Остання інструкція **"var=вираз"** вводиться, якщо потрібно, щоб функція повертала результат обчислень. Якщо необхідна більша кількість вихідних параметрів, структура модуля буде мати наступний вид:

function[var1,var2,...]=f_name(Список_параметрів_значень)

% Основний коментар – один рядок (обов'язковий)

% Додатковий коментар – будь-яке число рядків (необов'язковий)

Тіло файлу з будь-якими виразами, командами і керуючими структурами.

var1=вираз

var2=вираз

...

Імена **var, var1, var2 ...** для значень, що повертаються, є глобальними або відомими в тілі М-функції, що здійснює виклик.

1.1.4 Керуючі структури

Крім програм з лінійною структурою, інструкції яких виконуються виключно за порядком, MATLAB дозволяє створювати програми, структура яких нелінійна. Для створення таких програм застосовуються такі керуючі структури:

1. Діалогове введення: **input (рядок);**
2. Умовний оператор: **if ... elsif ... else ... end;**
3. Цикли типу for ... end: **for Заголовок_цикла тіло_цикла end;**
4. Цикли типу while ... end: **while Умова_цикла тіло_цикла end;**
5. Конструкція перемикача: **switch Exp case B1 case b2 ... otherwise end;**
6. Конструкція try ... catch ... end: **try Тіло_try catch Тіло_catch end;**
7. Створення паузи в обчисленнях: **pause, pause (...), pause on, pause off**.

Для організації діалогового виведення використовуються функції **input** і **disp**. Функція **input** повинна виглядати так:

змінна = input (рядок)

При виконанні цієї команди спочатку виводиться рядок, потім відбувається зупинка роботи програми і очікується введення значення. Введення підтверджується натисканням клавіші Enter, після чого введене значення присвоюється змінної.

Функція **disp** призначена для виведення її параметра на екран:

disp (значення, що виводиться)

Умовний оператор **if** в загальному вигляді записується в такий спосіб:

if Умова
Список_інструкцій_If
elseif Умова
Список_інструкцій_Elsif
else
Список_інструкцій_Else
end

Цикли типу **for ... end** зазвичай використовуються для організації обчислень із заданим числом повторюваних циклів. Конструкція такого циклу має такий вигляд:

for var = Вираз, Список_інструкцій end

Вираз найчастіше записується у вигляді **s:d:e**, де **s** – початкова значення змінної циклу **var**, **d** – приріст цієї змінної і **e** – кінцеве значення керуючої змінної, при досягненні якого цикл завершується. За замовчуванням **d** дорівнює **1**.

Цикл типу **while ... end** виконується до тих пір, поки залишається істинним умова:

while Умова
Список_інструкцій
end

Дострокове завершення циклів реалізується за допомогою операторів **break** або **continue**.

Для здійснення множинного вибору (або розгалуження) використовується конструкція з перемикачем типу **switch**:

switch switch_Вираз
case case_Вираз
Список_інструкцій
case
{Case_Вираз1, case_Вираз2, case_Вираз3, ...}
Список_інструкцій
...
otherwise,
Список_інструкцій
end

Case_вираз може бути числом, константою, змінною, вектором комірок або навіть малої змінної. В останньому випадку оператор case правдивий, якщо функція strcmp (значення, вираз) повертає логічне значення "істина".

Конструкція блоку виведення помилок try ... catch ... end має наступний синтаксис:

try
Список_інструкцій
catch
Список_інструкцій
end

Ця конструкція виконує всі списки інструкцій. Якщо в якомусь списку до оператора **catch** з'являється помилка, то виводиться повідомлення про помилку, але системна змінна останньої помилки lasterr не змінюється. В виразах після catch повідомлення про помилку не виводиться.

У всіх керуючих структурах список інструкцій, або тіло, являє собою послідовність виразів, команд або вкладених керуючих структур, що розділяються пропуском, комою або крапкою з комою. Крапка з комою забороняє висновок даних на екран.

Для зупинки програми використовується оператор pause. Він

використовується в наступних формах:

1) **pause** – зупиняє обчислення до натискання будь-якої клавіші;
2) **pause (N)** – зупиняє обчислення на N секунд;
3) **pause on** – включає режим обробки пауз;
4) **pause off** – вимикає режим обробки пауз.

1.1.5 Вектори, матриці і багатовимірні масиви

В системі MATLAB будь-які дані представляються тільки у вигляді масивів: одновимірних (векторів), двовимірних (матриць) і багатовимірних будь розмірності. Система MATLAB виконує складні і трудомісткі операції над векторами і матрицями навіть в режимі прямих обчислень без будь-якого програмування. Підтримується безліч операцій над матрицями, таких як:

1. Створення матриць із заданими властивостями: ones, zeros, rand;
2. Конкатенація матриць: cat (dim, A, B), cat (dim, A1, A2, A3, ...);
3. Перестановка елементів: fliplr, flipud, perms;
4. Обчислення творів і підсумовування: prod, cumprod, sum;
5. Поворот: rot 90 (A), rot 90 (A, k);
6. Виділення трикутних частин матриць: tril (x), tril (x, k), triu;
7. Обчислення супроводжує матриці: compan;
8. Матричні операції векторної алгебри: cand, det, rank, norm;
9. Операції з багатовимірними масивами: +, –, *,. *, /, ./, ^,. ^.

Цікаво відзначити, що навіть звичайні числа і змінні в MATLAB розглядаються як матриці розміру 1x1, що дає однакові форми і методи проведення операцій над звичайними числами і масивами. Дана операція називається векторизацией.

Векторизация забезпечує і спрощення записи операцій, і істотне підвищення швидкості їх виконання. Це також означає, що більшість функцій може працювати з аргументами у вигляді векторів і матриць.

1.1.6 Структури

Масив записів – це новий тип масиву, в якому дозволяється накопичувати у вигляді записів різнорідні дані. Відмінна особливість такого масиву – наявність іменованих полів.

MATLAB підтримує такі функції при роботі з масивами записів:

Функція	Опис
struct	Створити масив записів
fieldnames	Отримати імена полів
getfield	Отримати зміст поля
setfield	Встановити зміст поля
rmfield	Видалити поле
isfield	Істинно, якщо це поле масива записів
isstruct	Істинно, якщо це масив записів

Користувач може розширити склад функцій, створюючи спеціальні М-файли для обробки конкретних даних.

Визначення структури

Структура – це масив записів з іменованими полями, призначеними для зберігання даних; причому поле може містити дані будь-якого типу: вектори, матриці, масиви і структури.

Структуру можна побудувати двома способами:

а) використовуючи оператори присвоювання;

б) використовуючи функцію **struct**.

Для того щоб сформувати найпростішу структуру розміру **1x1**, необхідно присвоїти дані відповідним полях. система MATLAB автоматично формує структуру в міру її заповнення.

Функція **struct** має наступний синтаксис:

str_array = struct ('<ім'я_поля1>', '<значення1>', '<ім'я_поля2>', '<значення2> ', ...).

Використовуючи індексацію, можна легко визначити значення будь-якого поля або елемента структури. Точно також можна привласнити значення будь-якого полю або елементу поля. Щоб звернутися до деякого полю, необхідно ввести точку (.) після імені структури, за яким має слідувати ім'я поля.

Безпосередня індексація – це, як правило, найбільш ефективний спосіб визначити чи привласнити значення поля запису.

Однак якщо використовувалася функція **fieldnames** і відоме ім'я поля, то можна скористатися функціями **setfield** і **getfield**.

Функція **getfield** дозволяє визначити значення поля або елемента поля:

f = getfield (array, {array_index}, 'field', {field_index}),

де аргументи **array_index** і **field_index** задають індекси для структури і поля; вони не є обов'язковими для структури розміру **1x1**. Результат застосування функції **getfield** відповідає елементу такої структури

f = array (array_index) .field (field_index);

За аналогією функція **setfield** дозволяє присвоювати значення полів, використовуючи звернення такого вигляду:

f = setfield (array, {array_index}, 'field', {field_index}, value)

Виконання операцій з полями і елементами полів абсолютно аналогічно операціям з елементами звичайного числового масиву. В обох випадках треба використовувати індексні вирази. Для обробки структур зі специфічною архітектурою полів можуть знадобитися спеціальні функції обробки полів і їх елементів. При написанні М-файлів для обробки структур необхідно пам'ятати, що користувач повинен сам виконати аналіз виникнення можливих помилок, пов'язаних з обробкою полів.

1.1.7 Масиви комірок

У систему MATLAB включений спеціальний тип масивів комірок, елементи якого самі, в свою чергу, є масивами. є такі функції для роботи з масивами комірок:

Функція	Опис
cell	Створити масив комірок
celldisp	Показати зміст масива комірок
cellplot	Показати графічну структуру масива комірок
num2cell	Перетворити числовий масив на масив комірок
deal	Обмін даними між будь-якими класами масивів
cell2struct	Перетворити масив комірок на структуру
struct2cell	Перетворити структуру на масив комірок
iscell	Істинно, якщо це масив комірок

Користувач може розширити склад цих функцій, створюючи спеціальні М-файли для обробки конкретних даних.

Визначення масиву комірок

Масив комірок – це масив, де елементами є комірки, які можуть містити будь-який тип масиву, в тому числі і масив комірок. Масиви комірок дозволяють зберігати масиви з елементами різних типів і різних розмірностей.

Створити масиви комірок можна двома способами:

а) використовуючи оператори присвоювання;

б) використовуючи функцію cell, яка дозволяє попередньо розмістити масив, а потім привласнити дані коміркам.

Можна побудувати масив комірок, привласнюючи дані окремих комірок. В цьому випадку система MATLAB автоматично будує масив у міру введення даних. Існує два способи привласнення даних окремих комірок.

Індексація комірок

В цьому випадку необхідно укласти індекси комірок в круглі дужки, використовуючи стандартні позначення для масиву. Укласти вміст комірки в правій частині оператора присвоювання в фігурні дужки {}.

Індексація вмісту

Для того щоб індексувати масив комірок, треба в лівій частині оператора присвоювання вказати елемент комірки у вигляді індексів в фігурних дужках по аналогії з елементами звичайного масиву, а також вказати вміст комірки в правій частині оператора присвоювання.

Існує також два способи для отримання даних з масиву комірок і передачі їх або в деякий числовий масив, або в новий масив комірок:

а) доступ до комірки за допомогою індексації вмісту;

б) доступ до підмножини комірок за допомогою індексації комірок.

Доступ до вмісту комірок (індексація вмісту)

Використовуючи індексування вмісту в правій частині оператора присвоювання, можна отримати доступ до деяких або всіх даних в одній комірці. Визначити змінну в лівій частині оператора присвоювання, щоб запам'ятати вміст комірки. Укласти індексний вираз в правій частині оператора присвоєння в фігурні дужки. Це буде означати, що присвоюється вміст комірок, а не самі комірки.

Доступ до підмножини комірок (індексація комірок)

Використовуючи індексацію комірок, можна перепризначити u1083 будь-який набір комірок іншої змінної для створення нового масиву комірок. Використовуючи двокрапка, можна отримати доступ до підмножини комірок всередині масиву комірок.

Видалення і перевизначення масиву комірок

Видаляючи комірки з масиву, можна зменшити розмірність масиву, застосувавши єдиний оператор присвоювання. За аналогією зі звичайним масивом слід використовувати індексацію вектора для видалення рядка або стовпчика комірок, привласнюючи при цьому порожню матрицю подмассіви:

A (j: k) = []

Таким чином, при видаленні комірок фігурні дужки взагалі не застосовуються в операторах присвоювання.

Як і для звичайних масивів за допомогою функції reshape можна перевизначати розміри масиву комірок, причому загальна кількість комірок має залишатися незмінним; за допомогою функції reshape ні видалити, ні додати комірки не можна.

Опис списків змінних

Масиви комірок можуть бути використані для заміни наступних списків змінних:

а) списків вхідних змінних;

б) списків вихідних змінних;

в) операцій виведення на екран терміналу;

д) квадратних і фігурних дужок при формуванні масивів.

Коли для індексування багатовимірного масиву комірок використовуються двокрапка і фігурні дужки, то система MATLAB обробляє вміст кожної комірки як окрему змінну.

Класи

Класи і об'єкти дозволяють додавати нові типи даних і нові операції. Клас описує тип змінної і визначає, які операції і функції можуть бути застосовані до цього типу змінної. Об'єкт – це структура або зразок деякого класу.

Додавання класів здійснюється в рамках операційного середовища системи MATLAB, яка забезпечує можливість зберігання створених об'єктів і організації каталогу М-файлів, що визначають допустимі методи обробки для даного класу об'єктів.

Каталог класу включає М-функції, що визначають способи, з допомогою яких оператори системи MATLAB, включаючи арифметичні, обробки індексів, конкатенації, обробляють об'єкти даного класу. Перевизначення вбудованих операторів для нового класу об'єктів в рамках об'єктно-орієнтованого підходу називається перевизначенням методів.

У мові MATLAB відсутній механізм оголошення змінних.

Наприклад, оператор **A = zeros (10, 10)** формує звичайну матрицю розміру **10x10**, яка є об'єктом класу **double**. Точно також оператор s = **'Hello world'** створює об'єкт класу **char**.

Те ж саме відноситься і до новостворюваних класах. Ніяких оголошень змінних або об'єктів не потрібно. об'єкти створюються динамічно за допомогою виклику конструктора класу.

Каталог класу

М-файли, що визначають методи для об'єктів даного класу об'єднуються в каталог класу, назва якого задається як **@ <ім'я_класу>.**

Конструктор класу

Каталог класу повинен обов'язково містити М-файл, званий конструктором класу. Назва конструктора має збігатися з назвами класу і каталогу без префікса @. Конструктор створює об'єкти, використовуючи дані у вигляді масиву записів (структури) і приписуючи їм мітку класу.

Функції isa і class. Ці функції використовуються конструктором, але можуть застосовуватися й поза каталогу класу.

Функція **isa (a, 'class_name')** перевіряє, чи належить об'єкт **a** даного класу.

При використанні поза контекстом методів функція class допускає тільки один аргумент.

Команда **class (a)** повертає рядок, що містить ім'я класу для об'єкта **a**.

Перетворення класів. Виклик функції перетворення класу має вид

b = class_name (a),

де **a** – об'єкт деякого класу, відмінного від **class_name**. В цьому випадку система MATLAB шукає метод з ім'ям **class_name** в каталозі класів для об'єкта a. Такий метод перетворює об'єкт одного класу в об'єкт іншого класу. Якщо даний об'єкт вже є об'єктом класу **class_name**, то система MATLAB викликає функцію конструктора, який просто повертає цей об'єкт.

Найбільш важливими функціями перетворення класів є **double** і **char**. Перетворення до класу **double** створює традиційний масив системи MATLAB, хоча це може і не відображати необхідного відповідності для деяких класів. Перетворення до класу **char** корисно для виведення на печатка.

При роботі з об'єктами і методами система MATLAB використовує спеціальне безліч правил, щоб гарантувати виклик необхідної функції. Якщо, по крайней мере, один з аргументів є об'єктом, система MATLAB розглядає список параметрів зліва направо, щоб визначити їх старшинство. Для операторів рівного старшинства вибирається крайній лівий. Потім до нього застосовуються такі правила:

1. Якщо ім'я викликається функції збігається з ім'ям вбудованої функції, то перевіряється, чи існує перевизначена версія цієї функції для цього класу, а потім – для батьківського. Якщо жоден з цих випадків не має місце, видається помилка.

2. Якщо ім'я функції збігається з назвою каталогу класів, перевіряється, чи не є ця функція функцією перетворення, і якщо так, то ця функція перетворення викликається. В іншому випадку викликається конструктор класу.

3. Якщо обидва випадки не підходять, то аналізуються наступні можливості:

а) якщо є метод відповідного типу, то викликається цей метод;

б) якщо є метод батьківського класу, то викликається метод батьківського класу;

в) якщо є функція з таким ім'ям на шляху пошуку, то викликається ця функція;

г) в іншому випадку генерується помилка.

Приватні методи і функції. Каталоги класів можуть мати пов'язані з ними приватні каталоги. Такі каталоги можуть містити як приватні методи, які працюють з об'єктами даного класу, так і приватні функції, які не працюють з об'єктами, але виконують загальні обчислення. Можна, можливо встановлювати приватний каталог в каталозі класу точно так же, як створюється будь-який приватний каталог, т. е. просто створити каталог, іменований private, всередині каталогу **@class_name**.

У багатьох випадках можна змінити поведінку операторів і функцій системи MATLAB, коли в якості аргументу виступає об'єкт. Це здійснюється шляхом перевизначення відповідних функцій. Перевизначення класу відкриває можливість обробки за допомогою цієї функції різних типів даних при довільній кількості вхідних аргументів.

Перевизначення арифметичних операцій. кожен вбудований оператор в системі MATLAB має ім'я. Тому будь-який оператор може бути перевизначений шляхом створення М-файлу з відповідною назвою в каталозі класів.

Перевизначення функцій. Можна перевизначити будь-яку М-функцію, створюючи функцію з тим же ім'ям в каталозі класу. коли функція застосовується до об'єкта, MATLAB насамперед переглядає каталог відповідного класу, а вже потім інші шляхи доступу.

1.2 Завдання для самостійного виконання лабораторної роботи

Завдання 1. Розробити файл-сценарій для побудови графіка синусоїди лінією червоного кольору з виведеною масштабної сіткою в інтервалі **[xmin, xmax]**.

1. Запустити редактор m-файлів і ввести наступну програму**:**

```
% Plot with color red
% Будує графік синусоїди лінією червоного кольору
% З виведеною масштабної сіткою в інтервалі [xmin, xmax]
x = xmin: 0.1: xmax;
plot (x, sin (x), 'r')
grid on
```

2. Зберегти файл під ім'ям «pcr.m».

3. Виконати в командному вікні MATLAB наступну команду:

>> **help pcr**

На екран виведеться інформація, що знаходиться в основному і додатковому коментарі файлу:

Plot with color red

Будує графік синусоїди лінією червоного кольору з виведеною масштабної сіткою в інтервалі [xmin, xmax]

4. Потім запустити сценарій на виконання:

>> **pcr**

??? Undefined function or variable 'xmin'.

Error in ==> C: \ MATLAB6p1 \ work \ pcr.m

On line 4 ==> x = xmin: 0.1: xmax;

На екран виводиться повідомлення про помилку, так як сценарій використовує глобальні змінні, які не визначені. Для успішного виконання прикладу необхідно поставити такі дії:

5. Визначити змінні xmin і xmax

>> **xmin = -10;**

>> **xmax = 10;**

6. Запустити службовий скрипт на виконання

>> **pcr**

Завдання 2. Розробити файл-функцію для вирішення попередньої задачі, виконавши такі дії:

1. Запустити редактор m-файлів і ввести наступну програму:

```
% Plot with color red
% Будує графік синусоїди лінією червоного кольору
% З виведеною масштабної сіткою в інтервалі [xmin, xmax]
function x = fun (xmin, xmax)
x = xmin: 0.1: xmax;
plot (x, sin (x), 'r')
grid on
```

2. Зберегти файл під ім'ям «**fun.m**».

3. Виконати в командному вікні MATLAB наступну команду:

fun (-10,10);

Зверніть увагу на те, що **xmin** і **xmax** передаються як параметри і оголошувати глобальні змінні не потрібно.

Завдання 3. Розробити файл-сценарій для вирішення попередньої завдання з можливістю введення значень користувачем, виконавши такі дії:

1.Запустіть редактор m-файлів і ввести наступну програму:

```
% Plot with color red
% Будує графік синусоїди лінією червоного кольору
% З виведеною масштабної сіткою в інтервалі [xmin, xmax]
disp ( 'введіть xmin і xmax');
xmin = input ( 'xmin =');
xmax = input ( 'xmax =');
x = xmin: 0.1: xmax;
plot (x, sin (x), 'r')
grid on
```

2.Сохраніть файл під ім'ям «**pcrdialog.m**».

3.Виполніть в командному вікні MATLAB наступні команди:

>> **pcrdialog**

Зверніть увагу на те, що **xmin** і **xmax** оголошуються в тілі сценарію, тому оголошувати глобальні змінні не потрібно.

Завдання 4. Створити масив комірок, що містить дані різного типу, за допомогою індексації комірок і індексації вмісту.

1. Для створення масиву комірок за допомогою індексації комірок ввести наступні команди:

>> A (1,1) = {[1 4 3; 0 5 8; 7 2 9]};
>> A (1,2) = { 'Anne Smith'};
>> A (2,1) = {3 + 7i};
>> A (2,2) = {– pi: pi / 10: pi};
>> A
A =
[3x3 double] 'Anne Smith'
[3.0000+ 7.0000i] [1x21 double]

2. Для створення масиву комірок за допомогою індексації вмісту ввести наступні команди:

>> A {1, 1} = [1 4 3; 0 5 8; 7 2 9];
>> A {1, 2} = 'Anne Smith';
>> A {2, 1} = 3 + 7i;
>> A {2, 2} = -pi: pi / 10: pi
A =
[3x3 double] 'Anne Smith'
[3.0000+ 7.0000i] [1x21 double]

3. Для відображення вмісту комірок використовувати функцію **celldisp**:

>> celldisp (A)
A {1,1} =
1 4 3
0 5 8
7 2 9
A {2,1} =
3.0000 + 7.0000i
A {1,2} =
Anne Smith
A {2,2} =
Columns 1 through 9
-3.1416 -2.8274 -2.5133 -2.1991 -1.8850 -1.5708 -1.2566
-0.9425 -0.6283
Columns 10 through 18
-0.3142 0 0.3142 0.6283 0.9425 1.2566 1.5708 1.8850
2.1991
Columns 19 through 21

2.5133 2.8274 3.1416

Завдання 5. Створити масив структур, що містять ім'я, прізвище та рік народження.

1. Для створення масиву структур ввести наступні команди:

```
>> S.name = 'Ed';
>> S.fam = 'Plum';
>> S.year = тисячу дев'ятсот сімдесят дев'ять
S =
name: 'Ed'
fam: 'Plum'
year: 1979
>> S (2) .name = 'Tony';
26
>> S (2) .fam = 'Miller';
>> S (2) .year = 1980
S =
1x2 struct array with fields:
name
fam
year
>> S (3) =
struct ('name', 'Jerry', 'fam', 'Garcia', 'year', 1981)
S =
1x3 struct array with fields:
name
fam
year
```

Завдання 6. Розробити файл-сценарій для вирішення попередньої задачі з можливістю введення значень користувачем.

1. Запустити редактор m-файлів і ввести наступну програму:

```
% Введення структури S
% Структура має три поля:
% Ім'я name, прізвище fam і рік народження year
disp ('Введіть поля структури');
S.name = input ('name =');
S.fam = input ('fam =');
S.year = input ('year =');
disp (S)
```

2. Зберегти файл під ім'ям «**strdialog.m**».

3. Виконати в командному вікні MATLAB наступну команду:

```
>> strdialog
```

Зверніть увагу на те, що **xmin** і **xmax** оголошуються в тілі сценарію і оголошувати глобальні змінні не потрібно.

Завдання 7. На системному диску, де встановлена математична система MATLAB, знайти папки інструментального пакету по нейронних мереж **Neural Network Toolbox (NNT)** (C:\ProgramFiles\MATLAB\R2007b\toolbox\nnet\nnet\ nnnetwork) і скопіювати в робочий каталог (C:\Documents and Settings\ TEMP.VT.001\Мої документи\MATLAB) всі методи класу network, що забезпечують створення, ініціалізацію, навчання, моделювання та візуалізацію нейронної мережі. У робочому каталозі знайти М-файл конструкторів класу і проаналізувати їх структуру.

Завдання 8. Запустити на покрокове виконання конструктор мережі network, який не має параметрів, і простежити послідовність дій по формування об'єкта класу network і завданням значень за замовчуванням для його атрибутів.

Завдання 9. Створити об'єкт класу нейронної мережі network, виконавши команду net = network, і вивести на екран всі поля і всі комірки цього об'єкта, використовуючи функцію **celldisp**.

Завдання 10. Виконавши команду **gensim (net)**, отримати на екрані структурну схему створеної мережі, розкрити її блоки за допомогою подвійного клацання миші і з'ясувати сенс параметрів конструктора **network**. Змінюючи значення параметрів, простежити їх вплив на елементи структурної схеми.

1.3 Зміст звіту

1. Титульний аркуш.
2. Короткий опис завдання і копії екранів виконаної за завданням роботи.
3. Висновки про виконану роботу.

Контрольні питання

1. Засоби для побудови команд і написання М-файлів.
2. Структура файлу-сценарію.
3. Структура файлу-функції.
4. Керуючі структури: діалогове введення.
5. Керуючі структури: умовний оператор.
6. Керуючі структури: цикли типу for ... end.
7. Керуючі структури: конструкція перемикача.
8. Керуючі структури: конструкція try ... catch ... end.
9. Керуючі структури: створення паузи в обчисленнях.
10. Операції над матрицями в MATLAB.
11. Масиви в MATLAB.
12. Класи в MATLAB.
13. Приватні методи і функції.
14. Перевизначення арифметичних операцій.
15. Перевизначення функцій.

Лабораторна робота № 2.
МОДЕЛІ ШТУЧНОГО НЕЙРОНА. ФУНКЦІЇ АКТИВАЦІЇ

Мета роботи: вивчення основних моделей штучного нейрона, їх математичного опису, а також функціонального і структурного графічного вигляду, дослідження функцій активації та розглянутих моделей нейронів за допомогою інструментального пакета імітаційного моделювання Simulink системи MATLAB.

2.1 Загальні теоретичні відомості

2.1.1 Простий нейрон

Елементарною клітинкою нейронної мережі є нейрон. Структура нейрона з єдиним скалярним входом показана на рис. 2.1, а.

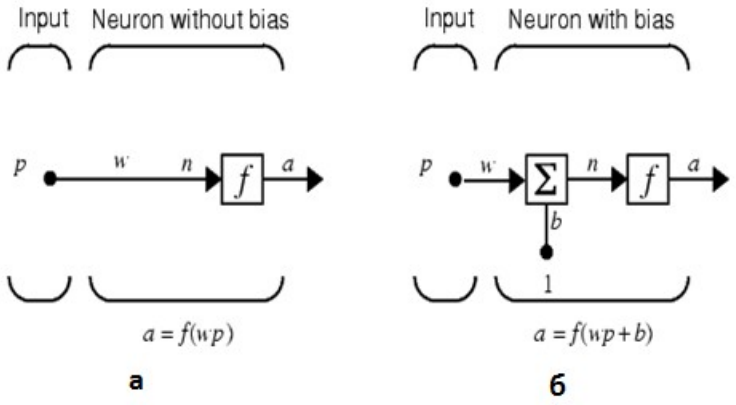


Рисунок 2.1 – Простий нейрон

Скалярний вхідний сигнал p множиться на скалярний ваговий коефіцієнт w, і результуючий зважений вхід w*p є аргументом функції активації нейрона f, яка породжує скалярний вихід a.

Нейрон показаний на рис. 2.1, б, доповнений скалярним зміщенням b. Зміщення підсумовується з виваженим входом w*p і призводить до зрушення аргументу функції f на величину b. Дію зміщення можна звести до схеми зважування, якщо уявити, що нейрон має другий вхідний сигнал із значенням, рівним 1. Вхід n функції активації нейрона як і раніше залишається скалярним і рівним сумі зваженого входу і зміщення b. Ця сума (w*p + b*1) є аргументом функції активації f, а виходом функції активації є сигнал a. Константи w і b є скалярними параметрами нейрона. Основний принцип роботи нейронної мережі полягає в налаштуванні параметрів нейрона з тим, щоб функціонування мережі відповідало деякій бажаній поведінці. Регулюючи ваги або параметри зміщення, можна "навчити" мережу виконувати конкретну роботу; можливо також, що мережа сама буде коригувати свої параметри, щоб досягти необхідного результату.

Рівняння нейрона зі зміщенням має вигляд:

a = f(w*p+b*l).

Як уже зазначалося, зміщення *b* – налаштовуваний скалярний параметр нейрона, який не є входом. В цьому випадку *b* – вага, а константа 1, яка управляє зміщенням, розглядається як вхід і може бути врахована у вигляді лінійної комбінації векторів входу

$$n = [w \quad b]\begin{bmatrix} p \\ 1 \end{bmatrix} = w * p + b * 1.$$

2.1.2 Функції активації

Функції активації (передавальні функції) нейрона можуть мати самий різний вигляд. Функція активації f, як правило, належить до класу сигмоїдальних функцій з аргументом n і виходом a. Нижче розглянуті три найбільш поширені функції активації.

Одинична функція активації з жорстким обмеженням

hardlim. Ця функція описується співвідношенням **a = hardlim(n) = 1(n)** і показана на рис. 2.2. Вона дорівнює 0, якщо n < 0, і дорівнює 1, якщо n ≥ 0.

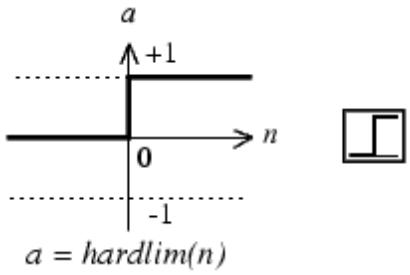


Рисунок 2.2 – Функція активації hardlim

До складу пакета NNT Neural Network Toolbox входить M-функція **hardlim**, що реалізує функцію активації з жорсткими обмеженнями.

Лінійна функція активації **purelin**. Ця функція описується співвідношенням a = purelin(n) = n і показана на рис. 2.3.

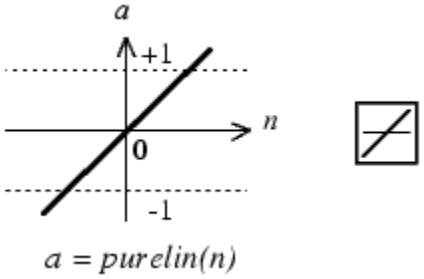


Рисунок 2.3 – Лінійна функція активації purelin

Логістична функція активації **logsig.** Ця функція описується співвідношенням **a = logsig(n) = 1/(1+exp(-n))** і показана на рис. 2.4. Вона належить до класу сигмоїдальних функцій, і її аргумент може приймати будь-яке значення в діапазоні від − ∞ до + ∞, а вихід змінюється в діапазоні від 0 до 1. В пакеті NNT Neural Network Toolbox вона представлена М-функцією **logsig**.

Завдяки властивості до диференціювання ця функція часто використовується в мережах з навчанням на основі методу зворотнього поширення помилки.

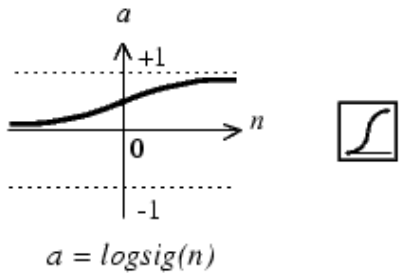


Рисунок 2.4 – Функція logsig

Символ в квадраті в правому верхньому куті графіка характеризує функцію активації. Це зображення вказується на структурних схемах нейронних мереж.

У пакет NNT Neural Network Toolbox включені і інші функції активації. Використовуючи мову MATLAB, користувач може створювати свої власні унікальні функції.

2.1.3 Нейрон з векторним входом

Нейрон з одним вектором входу p з R елементами p1, p2,…, pR показаний на рис. 2.5. Тут кожен елемент входу множиться на ваги w11, w12,…, w1R відповідно, і зважен значення передаються на суматор. Їх сума дорівнює скалярному добутку вектора-рядка W на вектор-стовпець входу p.

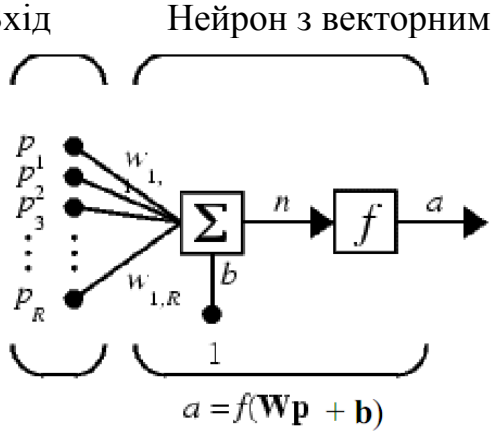


Рисунок 2.5 – Функціональна схема нейрона

Нейрон має зміщення *b*, яке додається до зваженої суми входів. Остаточна сума n визначається як

$$n = w_{11}p_1 + w_{12}p_2 + \cdots + w_{1R}p_R + b.$$

і служить аргументом функції активації *f*. В нотації мови MATLAB цей вислів записується так:

n = W * p + b.

Структура нейрона, показана на рис. 2.5, містить багато зайвих деталей. При розгляді мереж, що складаються з великої кількості нейронів, буде використовуватися укрупненная структурна схема нейрона (рис. 2.6).

Вхід нейрона зображується у вигляді темної вертикальної риси, під якою вказується кількість елементів входу R. Розмір вектора входу p вказується нижче символу p і дорівнює Rx1. Вектор входу множиться на вектор-рядок W довжини R. Як і раніше, константа 1 розглядається як вхід, який множиться на скалярний зміщення b. Входом n функції активації нейрона служить сума зміщення b і добуток W*p. Ця сума перетворюється функцією активації f, на виході якої є вихідна величина нейрона a, яка в даному випадку є скалярною величиною. Структурна схема, наведена на рис. 2.6, називається шаром мережі. Шар характеризується матрицею ваг W, зміщенням b, операціями множення W*p, підсумовування і функцією активації f. Вектор входів p зазвичай не включається в характеристики шару.

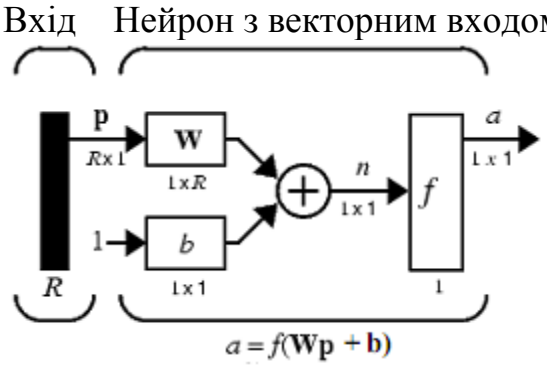
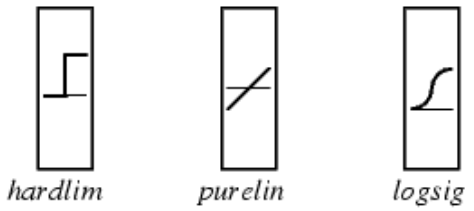


Рисунок 2.6 – Структурна схема нейрона

Кожен раз, коли використовується скорочена назва мережі, розмірність матриць вказується під іменами векторно-матричних змінних. Ця система позначень пояснює будову мережі і пов'язану з нею матричну математику.

На укрупненій структурній схемі для позначення типу функції активації застосовуються спеціальні графічні символи; деякі з них наведені на рис. 2.7, де а – ступінчаста, б – лінійна, в – логістична функції.



а б в

Рисунок 2.7 – Функції активації

2.2 Завдання для самостійного виконання лабораторної роботи

Завдання 1. Для функції активації з жорсткими обмеженнями hardlim і її похідної dhardlim, обумовленими такими співвідношеннями:

$$hardlim(n) = \begin{cases} 0, & n < 0 \\ 1, & n \geq 0 \end{cases},$$

$$dhardlim(n) = \begin{cases} 0, & n < 0 \\ 0, & n \geq 0 \end{cases}.$$

виконати наступні дії:

1. Видати на екран інформацію про ці функції за допомогою наступних команд:

name = hardlim ('name')% – повну назву функції;
inrange = hardlim ('active')% – діапазон входу;
outrange = hardlim ('output')% – діапазон виходу;

2. Побудувати графіки функцій:

n = -5: 0.1: 5;
a = hardlim (n);
da = dhardlim (n);
plot (n, a, 'r')% – графік функції активації – червоний;
hold on
plot (n, da, 'c')% – графік похідної – блакитний;

3. Розрахувати вектори виходу **A** і похідну **dA_dN** для шару з трьох нейронів з вектором входу **N**, що складається з трьох компонентів:

N = [– 0,7; 0,1; 0,8];
A = hardlim(N)% – вектор виходу функції активації;
dA_dN = dhardlim (N, A)% – вектор виходу похідної.

4. Розглянуту послідовність команд оформити у вигляді скрипта і записати в М-файл з ім'ям **hardlimfile**.

Завдання 2. Для симетричної функції активації з жорсткими обмеженнями **hardlims** і її похідної **dhardlms**, обумовленими співвідношеннями

$$hardlims(n) = \begin{cases} -1, & n < 0 \\ 1, & n \geq 0 \end{cases},$$

$$dhardlms(n) = \begin{cases} 0, & n < 0 \\ 0, & n \geq 0 \end{cases}.$$

видати на екран інформацію про ці функції, побудувати їх графіки і розрахувати вектори виходу, скориставшись скриптом з М-файлу **hardlimfile**. Новий скрипт записати в файл під ім'ям **hardlimsfile**.

Завдання 3. Для лінійної функції активації purelin і її похідної, dpurelin, обумовленими співвідношеннями

***purelin* = N;**

***dpurelin* = 1,**

видати на екран інформацію про ці функції, побудувати їх графіки і розрахувати вектори виходу, скориставшись скриптом з М-файлу **hardlimfile**. Новий скрипт записати в файл під ім'ям **purelinfile**.

Завдання 4. Для додатної лінійної функції активації poslin і її похідної dposlin, обумовленими співвідношеннями

$$poslin(n) = \begin{cases} 0, & n < 0 \\ n, & n \geq 0 \end{cases},$$

$$dposlin(n) = \begin{cases} 0, & n < 0 \\ 1, & n \geq 0 \end{cases}.$$

видати на екран інформацію про ці функції, побудувати їх графіки і розрахувати вектори виходу, скориставшись скриптом з М-файлу **hardlimfile**. Новий скрипт записати в файл під ім'ям **poslinfile**.

Завдання 5. Для лінійної функції активації з обмеженнями **satlin** і її похідної **dsatlin**, обумовленими співвідношеннями

$$satlin(n) = \begin{cases} 0, & n < 0 \\ n, & 0 \leq n \leq 1 \\ n, & n > 0 \end{cases},$$

$$dsatlin(n) = \begin{cases} 0, & n < 0 \\ 1, & 0 \leq n \leq 1 \\ 1, & n > 0 \end{cases}.$$

видати на екран інформацію про ці функції, побудувати їх графіки і розрахувати вектори виходу, скориставшись скриптом з М-файлу **hardlimfile**. Новий скрипт записати в файл під ім'ям **satlinfile**.

Завдання 6. Для симетричного лінійної функції активації **satlins** з обмеженнями і її похідної **dsatlins**, обумовленими співвідношеннями

$$satlins(n) = \begin{cases} -1, & n < -1 \\ n, & -1 \le n \le 1 \\ 1, & n > 1 \end{cases},$$

$$dsatlins(n) = \begin{cases} 0, & n < -1 \\ 1, & -1 \le n \le 1 \\ 0, & n > 1 \end{cases},$$

видати на екран інформацію про ці функції, побудувати їх графіки і розрахувати вектори виходу, скориставшись скриптом з М-файлу **hardlimfile**. Новий скрипт записати в файл під ім'ям **satlinsfile**.

Завдання 7. Для радіальної базисної функції активації **radbas** і її похідної **dradbas**, обумовленими співвідношеннями

$$radbas = e^{-2*n};$$

$$dradbas = -2 * n * e^{-2*n},$$

видати на екран інформацію про ці функції, побудувати їх графіки і розрахувати вектори виходу, скориставшись скриптом з М-файлу **hardlimfile**. Новий скрипт записати в файл під ім'ям **radbasfile**.

Завдання 8. Для трикутної функції активації **tribas** і її похідної **dtribas**, обумовленими співвідношеннями

$$tribas(n) = \begin{cases} 0, & n < -1 \\ 1 - abs(n), & -1 \le n \le 1 \\ 0, & n > 1 \end{cases},$$

$$dtribas(n) = \begin{cases} 0, & n < -1 \\ 1, & -1 \le n \le 0 \\ -1, & 0 < n \le 1 \\ 0, & n > 1 \end{cases},$$

видати на екран інформацію про ці функції, побудувати їх графіки і розрахувати вектори виходу, скориставшись скриптом з М-файлу **hardlimfile**. Новий скрипт записати в файл під ім'ям **tribasfile**.

Завдання 9. Для логістичної функції активації **logsig** і її похідної **dlogsig** обумовленими співвідношеннями

$$logsig(n) = 1/(1 + e^{-n}),$$

$$dlogsig(n) = e^{-n}/(1 + e^{-n})^2,$$

видати на екран інформацію про ці функції, побудувати їх графіки і розрахувати вектори виходу скориставшись скриптом з М-файлу **hardlimfile**. Новий скрипт записати в файл під ім'ям **logsigfile**.

Завдання 10. Для гіперболічної тангенціальної функції активації **tansig** і її похідної **dtansig** обумовленими співвідношеннями

$$tansig(n) = 2/(1 + e^{-2*n}) - 1,$$

$$dtansig(n) = 1 - \text{tansig}^2\, n,$$

видати на екран інформацію про ці функції, побудувати їх графіки і розрахувати вектори виходу, скориставшись скриптом з М-файлу **hardlimfile**. Новий скрипт записати в файл під ім'ям **tansigfile**.

Завдання 11. Для конкуруючої функції активації **compet** використовуваної для формування ймовірних і самоорганізованих нейронних мереж виконати наступні дії:

1. Видати на екран інформацію про цю функцію за допомогою наступного скрипта:

Name = compet ('name')% – cometitive;
Inrange = compet ('active')% – -in: inf;
Outrange = compet ('output')% – 0 1.

2. Побудувати столбцову діаграму для вектора входу і для вектора виходу, використовуючи шар з чотирьох нейронів:

N = [0; 1; – 0.5; 0.5];
A = compet (N);
subplot (2, 1, 1),% – подокна 2 × 1; висновок в 1-е;
bar (N),% – стовпчикова діаграма;
ylabel ('N')% – мітка осі ординат;
subplot (2, 1, 2), bar (A), ylabel ('A')% – у 2-му подвікні.

3. Розглянуту послідовність команд оформити у вигляді скрипта в М-файл з ім'ям **competfile**.

Завдання 12. Виконати дії 11-го завдання для конкуруючої функції активації з м'яким максимумом **softmax**, записавши при цьому новий скрипт в М-файл з ім'ям **softmaxfile**.

Завдання 13. Для простого нейрона з одним подвійним входом P і функції активації **hardlim** підібрати ваговий коефіцієнт W і зміщення b таким чином, щоб забезпечити інвертування вхідного сигналу, тобто заміну нуля одиницею, а одиницю нулем.

Завдання 14. Для нейрона з двома подвійними входами p_1 і p_2 і функцією активації **hardlim** підібрати вагові коефіцієнти і зміщення таким чином, щоб нейрон виконував функції логічного додавання і логічного множення.

Завдання 15. Для нейрона з двома подвійними входами p_1 і p_2 і функцією активації **hardlim** підбирати вагові коефіцієнти W_{11} і W_{12} та зміщення b таким чином, щоб класифікувати вхідні подвійні набори на два класи – нульовий і перший:

а) {00, 01} – нульовий клас, {10, 11} – перший клас;
б) {11} – нульовий клас, {00, 01, 10} – перший клас;
в) {00, 11} – нульовий клас, {01, 10} – перший клас;
г) {00, 11} – перший клас, {01, 10} – нульовий клас.

Завдання 16. Для нейрона з двома безперервними входами p_1 і p_2 та функції активації **hardlim** побудувати графік розділяючої лінії, яка визначається рівнянням

$$W_{11}p_1 + W_{12}p_2 + b = 0,$$

вважаючи, що значення вагових коефіцієнтів W_{11}, W_{12} і зміщення b задані. Переконатися, що набори входів p_1 і p_2 по різні боки від розділяючої лінії належать різним класам і що не всяка множина наборів значень входів можна розділити на два класи, використовуючи нейрон розглянутого типу.

2.3 Зміст звіту

1. Титульний лист.

2. Короткий опис завдання і копії екранів виконаної за завданням роботи.
3. Висновки про виконану роботу.

Контрольні питання

1. Структура нейрона з єдиним скалярним входом.
2. Одинична функція активації з жорстким обмеженням.
3. Нейрон з одним вектором входу.
4. Функція активації hardlim.
5. Лінійна функція активації purelin.
6. Сигмоїдальна функція.
7. Метод зворотнього поширення помилки.

Лабораторна робота № 3.
ШТУЧНІ НЕЙРОННІ МЕРЕЖІ

Мета роботи: вивчення архітектури штучних нейронних мереж, способів їх графічного зображення у вигляді функціональних і структурних схем і програмного уявлення у вигляді об'єктів спеціального класу network, що включають масив структур з атрибутами мережі і набір необхідних методів для створення, ініціалізації, навчання, моделювання і візуалізації мережі , а також придбання навичок побудови мереж різної архітектури за допомогою інструментального програмного пакета Neural Network Toolbox системи MATLAB.

3.1 Загальні теоретичні відомості

Хоча окремі нейрони і здатні після деякої процедури навчання вирішувати ряд завдань штучного інтелекту, все ж для ефективного вирішення складних завдань по розпізнаванню образів, ідентифікації та класифікації об'єктів, розпізнавання і синтезу мови, оптимальному управлінню застосовують досить великі групи нейронів, утворюючи з них штучні нейронні мережі у вигляді пов'язаних між собою шарів, що нагадують біологічні нейронні (нервові) мережі людини і тварин.

Існує безліч способів організації штучних нейронних мереж, які можуть містити різне число шарів нейронів. Нейрони можуть бути пов'язані між собою як всередині окремих шарів, так і між шарами. Залежно від напрямку передачі сигналу ці зв'язки можуть бути прямими або зворотними. Шар нейронів, безпосередньо приймає інформацію із зовнішнього середовища, називається вхідним шаром, а шар, що передає інформацію у зовнішню середу, – вихідним шаром. Решта шарів, якщо вони є в мережі, називаються проміжними, або прихованими. У ряді випадків такого функціонального розподілу шарів мережі не проводиться, так що входи і виходи можуть приєднуватися до будь-яких верств і мати довільне число компонент.

Структура, або архітектура мережі штучних нейронів залежить від тієї конкретної задачі, яку повинна вирішувати мережа. Вона може бути одношаровою без зворотних зв'язків або із зворотними зв'язками, двошаровою з прямими зв'язками, тришаровою зі зворотними зв'язками і т.д. Мережі зі зворотними зв'язками називають часто рекурентними.

Опис архітектури штучної нейронної мережі крім вказівки числа шарів і зв'язків між ними має включати відомості про кількість нейронів в кожному шарі, вигляді функцій активації в кожному шарі, наявності зміщень для кожного шару, наявності компонент вхідних, вихідних і цільових векторів, а в ряді випадків і характеристики топології шарів. Наприклад, для апроксимації будь-якої функції з кінцевим числом точок розриву широко використовується мережа з прямою передачею сигналів. У цій мережі є декілька шарів з сигмоїдальна функціями активації. Вихідний шар містить нейрони з лінійними

функціями активації. Дана мережа не має зворотного зв'язку, тому її називають мережею з прямою передачею сигналів (FF-net).

Графічно штучна нейронна мережа зображується у вигляді функціональної або структурою схеми. На функціональній схемі мережі за допомогою геометричних фігур зображуються її функціональні блоки, а стрілками показуються входи, виходи і зв'язку. У блоках і на стрілках вказуються необхідні позначення і параметри.

Структурна схема мережі зображується за допомогою типового набору блоків, з'єднувальних елементів і позначень, прийнятих в інструментальному програмному пакеті Neural Network Toolbox системи MATLAB і пакеті імітаційного моделювання Simulink тієї ж системи. Структурна схема мережі може бути укрупненої або детальної, причому ступінь деталізації визначається користувачем. Системи позначень блоків, елементів і параметрів мережі є векторно-матричної, прийнятої в системі MATLAB. Якщо в позначенні використовується два індексу, то, як правило, перший індекс (індекс рядка) вказує адресата, або пункт призначення, а другий індекс (індекс стовпця) – джерело структурної схеми мережі. Структурні схеми створюються системою автоматично за допомогою команди gensim. Якщо елементом вектора або матриці на структурній схемі є складний об'єкт, то використовуються відповідно осередок і масив комірок.

Програмним поданням, або обчислювальною моделлю штучної нейронний мережі, є об'єкт спеціального класу network, який включає масив структур з атрибутами мережі і набір методів, необхідних для створення мережі, а також для її ініціалізації, навчання, моделювання і візуалізації. Клас Network має два загальних конструктора, один з яких не має параметрів і забезпечує створення масиву структур з нульовими значеннями полів, а другий – має мінімальний набір параметрів для створення моделі нейронної мережі, добудовувати потім до потрібної конфігурації за допомогою операторів присвоювання. Для створення нейронних мереж певного виду використовуються спеціальні конструктори.

3.2 Завдання для самостійного виконання лабораторної роботи

Завдання 1. Створити обчислювальну модель нейронної мережі з двома виходами, трьома шарами і одним цільовим входом, використовуючи загальний конструктор мережі з параметрами.

Net = network (numInputs, numLayers, biasConnect, inputConnect, layerConnect, outputConnect, targetConnect)

Зв'язки між шарами повинні бути тільки прямими, входи необхідно з'єднати з першим шаром, а вихід – з останнім. Вихід повинен бути цільовим, перший шар повинен мати зміщення.

Сенс і значення параметрів конструктора для створення моделі мережі заданої архітектури такі:

numImputs = 2 – кількість входів мережі;

numLayers = 3 – кількість шарів в мережі;

biasConnect = [1; 0; 0] – матриця зв'язності для зміщень розміру numLayers * 1;

inputConnect = [1 + 1; 0 0; 0 0] – матриця зв'язності для входів розміру numLayers * numlmputs;

layerConnect = [0 0 0; 1 0 0; 0 1 0] – матриця зв'язності для шарів розміру numLayers * numLayers;

outputConnect = [0 0 1] – матриця зв'язності для виходів розміру 1 * numLayers;

targetConnect = [0 0 1] – матриця зв'язності для цілей розміру 1 * numLayers.

Порядок виконання завдання наступний:

1. Створити шаблон мережі, виконавши команду

net = network (2,3, [1; 0; 0], [1 1; 0 0; 0 0], [0 0 0; 1 0 0; 0 1 0], [0 0 1], [0 0 1])

2. Перевірити значення полів обчислювальної моделі нейронної мережі net і їх відповідність заданим значенням в списку параметрів.

3. Перевірити значення обчислюваних полів моделі, які доповнюють опис архітектури мережі

numOutputs = 1 – кількість виходів мережі;

numTargets = 1 – кількість цілей мережі;

numlnputDelays = 0 – максимальне значення затримки для входів мережі.

numLayersDelays = 0 – максимальне значення затримки для шарів мережі.

Зауважимо, що кожен вихід і кожна мета приєднуються до одного або декількох шарів при цьому кількість компонент виходу або цілі дорівнює кількості нейронів у відповідному шарі. Для збільшення можливості моделі в мережу включають лінії затримки або на її входах, або між шарами. Кожна лінія затримує сигнал на один такт. Параметри **numlnputDelays** і **NumLayerDelays** визначають максимальне число ліній для будь-якого входу або шару відповідно.

4. Проаналізувати структурну схему побудованої мережі, виконавши команду **gensim (net)** і деталізуючи блоки за допомогою подвійного клацання лівої кнопки миші з даного блоку. На структурних схемах штучних нейронних мереж в пакеті **NNT** можуть використовуватися такі символи:

а) ***Neural Network*** – штучна нейронна мережа з позначеннями входів р {1}, р {2}, ... і виходу у {1};

б) входи ***Inputl***, або р {1} і ***Input2***, або р {2};

в) дисплей у {1};

г) ***Layer*** 1, ***Layer*** 2, ***Layer*** 3, ... шари нейронів з позначеннями входів р {1}, р {2], а {1}, а {2}, ... і виходів а {1}, а {2}, а {3}, ..., у {1};

д) ***TDL*** – лінії затримки (***Time Delay)*** з іменами ***Delays1***, ***Delays2***, ..., які забезпечують затримку вхідних сигналів або сигналів між шарами нейронів на 1, 2, 3, ... такту;

е) ***Weights*** – вагова матриця для вхідних сигналів або сигналів між шарами нейронів; розмір матриці ваг для кожного вектора входу ***SxR***, де ***S*** – число нейронів вхідного шару, а ***R*** – число компонент вектора входу, помножене на число затримок; розмір матриці для сигналів від шару ***j*** до шару і дорівнює ***SxR***, де ***S*** – число нейронів в шарі **i**, а ***R*** – число нейронів в шарі j, помножене на число затримок;

ж) ***dotprod*** – блок зважування вхідних сигналів і сигналів між шарами, на виході якого виходить сума зважених, т. е. помножених на відповідні ваги компонент сигналу;

з) ***mux*** – концентратор вхідних сигналів і сигналів між шарами, перетворює набір скалярних сигналів в вектор, а набір векторів в один вектор сумарної довжини;

і) ***netsum*** – блок підсумовування компонент для кожного нейрона шару: компонент від декількох векторів входу з урахуванням затримок, зміщення і т. п .;

к) ***hardlim***, ***purelin*** і т. д. – блоки функцій активації;

л) ***pd {1, 1}, pd {1, 2}, ad {2, 1},*** ... – сигнали після ліній затримки (***d – delay***);

м) ***iz {1, 1}, iz {1, 2}, lz {2, 1}, lz {3, 2}*** – вектор-сигнали з виходу концентратора;

н) ***bias*** – блок ваг зміщень для шару нейронів; о) ***IW*** – масив комірок з матрицями ваг входів: ***IW {i, j}*** – матриці для шару ***i*** від вхідного вектора ***j***;

п) ***LW*** – масив комірок з матрицями ваг для шарів***: LW {i, j}*** – матриці для шару ***i*** від шару ***j***.

5. Проаналізувати всі параметри кожного блоку структурної схеми даної нейронної мережі і в разі необхідності звернутися до довідкової системи пакета NNT.

6. Поставити нульові послідовності сигналів для входів ***P = [0 0; 0 0]***

і зробити моделювання мережі ***A = sim (net, P).***

7. Поставити діапазони вхідних сигналів і вагові матриці за допомогою наступних присвоювання:

net.inputs {1} .range = [0 1]; net.inputs {2} .range = [0 1]; net.b {1} = -%; net.IW {1, 1} = [0.5]; net.IW {1, 2} = [0.5]; net.LW {2, 1} = [0.5]; net.LW {3, 2} = [0.5].

Виконати команду ***gensim (net)*** і перевірити параметри блоку.

8. Вивести на екран поля обчислювальної моделі і їх вміст, використовуючи функцію ***celldisp***. Переконатися в правильності значень полів моделі.

9. Промоделювати створену статичну мережу, т. Е. Мережу без ліній затримки, використовуючи групове і послідовне уявлення вхідних сигналів

PG = [0.5 1; 1 0.5];
PS = {[0,5 1]; [1 0.5]};
AG = sim (net, PG);
AS = sim (net, PS).

Переконатися, що для статичної мережі групове і послідовне представлення вхідних сигналів дають один і той же результат.

10. Доповнити архітектуру створеної нейронної мережі лініями затримки під час приймання сигналу і для сигналів між 2-м і 3-м шарами, перетворивши таким чином статичну мережу в динамічну:

net.inputWeights {1, 1} .delays = [0 1];
net.inputWeights {1, 2} .delays = [0 1];
net.layerWeights {3, 2} .delays = [0 1 2].

Побудувати структурну схему динамічної мережі і з'ясувати сенс використовуваних операторів присвоювання.

11. Скорегувати вагові матриці:

net.IW {1, 1} = [0.5 0.5];
net.IW {1, 2} = [0.5 0.25];
net.LW {3, 2} = [0.5 0.25 1].

12. Промоделювати динамічну мережу, використовуючи групове і послідовне уявлення вхідних сигналів:

AG = sim (net, PG);
AS = sim (net, PS).

Переконатися, що групове уявлення вхідних сигналів спотворює результат, так як в цьому випадку робота однієї мережі замінюється паралельною роботою двох (по числу послідовностей) однакових мереж з нульовими початковими значеннями сигналів на виходах ліній затримки.

13. Вивести на друк поля обчислювальної моделі і їх вміст, використовуючи функцію ***celldisp***.

14. Зберегти вміст командного вікна в М-файлі для подальшого використання.

Завдання 2. Створити таку саму динамічну мережу **asgnet**, використовуючи конструктор класу **network** без параметрів і задаючи значення відповідних полів обчислювальної моделі за допомогою операторів присвоювання. Переконатися в ідентичності мереж **net** і **as– gnet**. Порівняти результати роботи отриманих мереж.

Завдання 3. Використовуючи блоки імітаційного моделювання інструментального пакету Simulink системи MATLAB, побудувати модель динамічної мережі **asgnet**, провести дослідження моделі, перевірити адекватність її поведінки поведінки моделі **net** і оформити електронний звіт за допомогою генератора **Report Generator**.

Завдання 4. Використовуючи конструктор класу **network** з параметрами і оператори присвоювання для полів і комірок об'єктів цього класу, побудувати, видати на екран і промоделювати штучні нейронні мережі наступній архітектури:

а) одношарова мережу з трьома нейронами, трьома двухкомпонентними входами і одним цільовим виходом;

б) тришарова мережу з прямою передачею сигналів і з трьома нейронами в кожному шарі; кількість входів – три з двома, п'ятьма і трьома компонентами; для всіх верств є зміщення; вихід – один;

в) тришарова мережа, в якій кожен шар з'єднаний з усіма іншими; вхід – один і складається з двох компонентів; кількість нейронів в кожному шарі – три; шари мають зміщення;

г) тришарова динамічна мережа з трьома нейронами в кожному шарі; число входів – три, з них кожен складається з трьох компонентів; є зміщення на всіх шарах; лінії затримки затримують сигнали на один і два такту і включені між усіма верствами, а також на вході;

д) квадратна мережу з десятьма шарами і десятьма нейронами в кожному шарі; десять векторів підключаються по одному до кожного шару; є десять виходів від всіх шарів мережі; зміщення підключені до кожного шару.

3.3 Зміст звіту

1. Титульний лист.
2. Короткий опис завдання і копії екранів виконаної за завданням роботи.
3. Висновки про виконану роботу.

Контрольні питання

1. Класи задач, у яких використовуються багатошарові нейронні мережи.
2. Вхідний шар нейронної мережі.
3. Проміжні (приховані) шари нейронної мережі.
4. Типи архітектур мереж штучних нейронів.
5. Мережа з прямою передачею сигналів (FF-net).
6. Графічне зображення штучних нейронних мереж.
7. Програмне подання (обчислювальна модель) штучної нейронної мережі.

Лабораторна робота № 4.
МЕТОДИ І АЛГОРИТМИ НАВЧАННЯ ШТУЧНИХ НЕЙРОННИХ МЕРЕЖ

Мета роботи: вивчення та набуття навичок практичного застосування методів і алгоритмів ініціалізації і навчання штучних нейронних мереж, а також оволодіння способами їх розробки.

4.1 Загальні теоретичні відомості

Після того як сформована архітектура нейронної мережі, повинні бути задані початкові значення ваг і зміщень, або іншими словами, мережа повинна бути ініціалізована. Така процедура виконується за допомогою методу init для об'єктів класу network. Оператор виклику цього методу має вигляд:

net = init (net).

Перед викликом цього методу в обчислювальної моделі мережі необхідно поставити такі властивості:

- **net.initFcn** – для визначення функцій, які будуть виконуватися для завдання початкових матриць ваг та ваг шарів, а також початкових векторів зміщень;
- **net.layers{i}.initFcn** – для задання функції ініціалізації і-го шару;
- **net.biases{i}.initFcn** – для задання початкового вектора зміщення і-го шару;
- **net.inputWeights{i, j}.initFcn** – для задання функції обчислення матриці ваг до шару і від входу j;
- **net.layerWeight{i, j}.initFcn** – для задання функції обчислення матриці ваг до шару і від входу j;
- **net.initParam** – для завдання параметрів функцій ініціалізації.

Спосіб ініціалізації мережі визначається заданням властивостей і **net.initFcn net.layers{i}.initFcn**. Для мереж з прямою передачею сигналів по стандарту використовується **net.initFcn = 'Initlay'**, що дозволяє для кожного шару використовувати власні функції ініціалізації, що задаються властивістю **net.layers {i} .initFcn** з двома можливими значеннями: **'initwb'** і **'Initnw'**.

Функція **initwb** дозволяє використовувати власні функції ініціалізації для кожної матриці ваг і для кожного вектора зміщень, при цьому можливими значеннями для властивостей **net.inputWeights{i, j}.initFcn** і **net.layerWeight{i, j}.initFcn** є: **'Initzero', 'Midpoint', 'randnc', 'rands'**, а для властивості **net.biases{i}.initFcn** – значення **'Initcon', 'initzero' і 'rands'**. Для мереж без зворотніх зв'язків з лінійними функціями активаці ваги зазвичай ініціалізуються випадковими значеннями з інтервалу [-1 1]. Функція **initnw** реалізує алгоритм Nguyen-Widrow і застосовується для шарів, що використовують Сигмоїдальні функції активації. Ця функція генерує початкові ваги і зміщення для шару так,

щоб активні області нейронів були розподілені рівномірно щодо області входів, що забезпечує мінімізацію числа нейронів мережі і час навчання. Іншими можливими значеннями властивості **net.initFcn** є: **'Initcon', 'Initnw', 'Initwb' і 'Initzero'**.

Крім функції **initnw** наступні функції виробляють безпосередньо ініціалізацію ваг і зміщень:

- **initzero** присвоює матрицям ваг і векторам зміщень нульові значення;
- **rands** присвоює матрицям ваг і векторам зміщень випадкові значення з діапазону [-1 1];
- **randnr** присвоює матриці ваг випадкові нормовані рядки з діапазону [-1 1];
- **randnc** присвоює матриці ваг випадкові нормовані стовпці з діапазону [-1 1];
- **midpoint** присвоює елементам вектора зміщення початкові рівні зміщення, що залежать від числа нейронів в шарі, і використовується разом з функцією налаштування **learncon.**

Таким чином, **задання функцій** ініціалізації для обчислювальної моделі нейронної мережі є багатоступеневим і виконується за наступним алгоритмом:

1. Вибрати для властивості **net.initFcn** одне з можливих значень: '**initzero'**, '**initcon'**, '**initnw'**, '**initwb'** або **'initlay'**.

2. Якщо обрані значення **'initzero', 'initcon'** або **'initnw'**, То задання функцій ініціалізації мережі завершено.

3. Якщо вибране значення **'initwb'**, То перехід до кроку 6.

4. Якщо вибране значення **'initlay'**, То переходимо до шарів і для кожного шару і властивості **net.layers{i}.initFcn** необхідно задати одне ізвозможних значень: '**initnw'** або **'initwb'**.

5. Якщо для і-го шару вибрано значення **'initnw'**, то для цього шару завдання функцій ініціалізації завершено.

6. Якщо для всіх шарів мережі або для якогось шару встановлена властивість **'initwb'**, то для цих шарів необхідно задати властивості **net.biases {i}.initFcn**, вибравши його з набору: **'initzero', 'rands'** або '**initcon'**, А також властивості **net.layerWeights {i, j}.initFcn**, використовуючи наступні значення: **'initzero', 'midpoint', 'randnc', 'randnr'** або '**rands'**. Зауважимо, що за допомогою оператора **revert (net)** можна повернути значення ваг і зміщень до раніше встановлених значень.

Після ініціалізації нейронної мережі її необхідно навчити рішенню конкретної прикладної задачі. Для цих цілей потрібно зібрати навчальний набір даних, що містить цікаві для ознаки досліджуваного об'єкта, використовуючи наявний досвід. Спочатку слід включити всі ознаки, які, на думку аналітиків і експертів, є істотними; на наступних етапах цю безліч, ймовірно, буде скорочено. Зазвичай для цих цілей використовуються евристичні правила, які встановлюють зв'язок між кількістю необхідних спостережень і розміром мережі. Зазвичай кількість спостережень на порядок більше числа зв'язків в мережі і зростає по нелінійному закону, так що вже при досить невеликій

кількості ознак, наприклад 50, може знадобитися величезне число спостережень. Ця проблема носить назву «прокляття розмірності». Для більшості реальних задач буває досить декількох сотень або тисяч спостережень.

Після того як зібраний навчальний набір даних для проектованої мережі, проводиться автоматичне налаштування ваг і зміщень за допомогою процедур навчання, які мінімізують різницю між бажаним сигналом і отриманим на виході в результаті моделювання мережі. Ця різниця носить назву «помилки навчання». Використовуючи помилки навчання для всіх наявних спостережень, можна сформувати функцію помилок або критерій якості навчання. Найчастіше в якості такого критерію використовується сума квадратів помилок. Для лінійних мереж при цьому вдається знайти абсолютний мінімум критерію якості, для інших мереж досягнення такого мінімуму не гарантується. Це пояснюється тим, що для лінійної мережі критерій якості, як функція ваг і зміщення, є параболоїдом, а для інших мереж – дуже складною поверхнею в N + 1-вимірному просторі,

З урахуванням специфіки нейронних мереж для них розроблені спеціальні алгоритми навчання. Алгоритми діють ітеративно, по кроках. Величина кроку визначає швидкість навчання і регулюється параметром швидкості настройки. При великому кроці є велика ймовірність пропуску абсолютного мінімуму, при малому кроці може сильно зрости час навчання. Кроки алгоритму прийнято називати епохами або циклами.

На кожному циклі на вхід мережі послідовно подаються всі навчальніспостереження, вихідні значення порівнюються з цільовими значеннями і обчислюється функція критерію якості навчання – функція помилки. Значення функції помилки, а також її градієнта використовуються для корегування ваг та зміщень, після чого всі дії повторюються. Процес навчання припиняється з таких трьох причин, якщо:

а) реалізовано задану кількість циклів;
б) помилка досягла заданої величини;
в) помилка досягла деякого значення і перестала зменшуватися.

У всіх цих випадках мережа мінімізувала помилку на деякій обмеженій навчальній множині, а не на безлічі реальних вхідних сигналів при роботі моделі. Спроби ускладнити модель і знизити помилку на заданій навчальній множині можуть привести до зворотного ефекту, коли для реальних даних помилка стає ще більше. Ця ситуація називається явищем перенавчання нейронної мережі.

Для того щоб виявити ефект перенавчання нейронної мережі, використовується механізм контрольної перевірки. З цією метою частина навчальних спостережень резервується як контрольні спостереження і не використовується при навчанні мережі. У міру навчання контрольні спостереження застосовуються для незалежного контролю результату. Якщо на деякому етапі помилка на контрольній множині перестала зменшуватися, навчання слід припинити навіть в тому випадку, коли помилка на навчальній множині продовжує зменшуватися, щоб уникнути явища перенавчання. В

цьому випадку слід зменшити кількість нейронів або шарів, так як мережа є занадто потужною для вирішення даного завдання. Якщо ж, навпаки, мережа має недостатню потужність, щоб відтворити залежність, то явище перенавчання швидше за все спостерігатися не буде і обидві помилки – навчання і контролю – не досягнуть необхідного рівня.

Таким чином, для відшукання глобального мінімуму помилки доводиться експериментувати з великим числом мереж різної конфігурації, навчаючи кожну з них кілька разів і порівнюючи отримані результати. Головним критерієм вибору в цих випадках є контрольна похибка. При цьому застосовується правило, згідно з яким з двох нейронних мереж з приблизно рівними контрольними похибками слід вибирати ту, яка простіше.

Необхідність багаторазових експериментів веде до того, що контрольна множина починає відігравати ключову роль у виборі нейронної мережі, тобто бере участь в процесі навчання. Тим самим її роль як незалежного критерію якості моделі послаблюється, оскільки при великій кількості експериментів виникає ризик перенавчання нейронної мережі на контрольній множині. Для того, щоб гарантувати надійність обраній моделі мережі, резервують ще тестову множину спостережень. Підсумкова модель тестується на даних з цієї множини, щоб переконатися, що результати, досягнуті на навчальній та контрольній множинах, реальні. При цьому тестова множина повинна використовуватися тільки один раз, інакше вона перетвориться в контрольну множину.

Отже, процедура побудови нейронної мережі складається з наступних кроків:

1. Вибрати початкову конфігурацію мережі у вигляді одного шару з числом нейронів, рівним половині загальної кількості входів і виходів.
2. Навчити мережу і перевірити її на контрольній множині, додавши в разі потреби додаткові нейрони і проміжні шари.
3. Перевірити, чи не перенавчана мережа. Якщо має місце ефект перенавчання, то зробити реконфігурацію мережі.

Для того щоб проектована мережа успішно вирішувала завдання, необхідно забезпечити представництво навчальної, контрольної і тестової множини. Але краще постаратися зробити так, щоб спостереження різних типів були представлені рівномірно. Добре спроектована мережа повинна мати властивість узагальнення, коли вона, будучи навченою на деякій множині даних, набуває здатності видавати правильні результати для досить широкого класу даних, в тому числі і не представлених при навчанні.

Інший підхід до процедури навчання мережі можна сформулювати, якщо розглядати її як процес, зворотний моделюванню. В цьому випадку потрібно підібрати такі значення ваг і зміщень, які забезпечували б необхідну відповідність між входами і бажаними значеннями на виході. Така процедура навчання носить назву процедури адаптації і досить широко застосовується для налаштування параметрів нейронних мереж.

По стандарту для мереж з прямою передачею сигналів в якості критерію навчання використовується функціонал, який представляє собою суму квадратів помилок між виходами мережі і їх цільовими значеннями:

$$J = \frac{1}{2}\sum_{q=1}^{Q}\sum_{i=1}^{S}(t_i^q - d_i^q)^2 ,$$

де Q – обсяг вибірки;

q – номер вибірки;

i – номер виходу;

t_i^q – цільове q значення для i-го виходу вибірки q;

d_i^q – сигнал на i-му виході при подачі вхідних сигналів q-ї вибірки.

Метою навчання мережі є мінімізація цього функціоналу за допомогою зміни ваг і зміщень.

В цей час розроблено кілька методів мінімізації функціоналу помилки на основі відомих методів визначення екстремумів функцій декількох змінних. Всі ці методи можна розділити на три класи:

а) методи нульового порядку, в яких для знаходження мінімуму використовується тільки інформація про значеннях функціоналу в заданих точках;

б) методи першого порядку, в яких використовується градієнт функціоналу помилки по параметрам, що використовує приватні похідні функціоналу;

в) методи другого порядку, в яких використовуються другі похідні функціоналу.

Для лінійних мереж задача знаходження мінімуму функціонала (параболоїда) зводиться до вирішення системи лінійних рівнянь, що включають ваги, зміщення, вхідні навчальні значення і цільові виходи і таким чином, може бути вирішена без використання ітераційних методів. У всіх інших випадках треба використовувати методи першого або другого порядку. Якщо використовується градієнт функціоналу помилки, то

$$X_{k+1} = X_k - \alpha_k g_k,$$

де X_k і X_{k+1} – вектори параметрів на k-й і (k+1)-й ітераціях;

α_k – параметр швидкості навчання;

g_k – градієнт функціоналу, відповідний k-й ітерації.

Якщо використовується сполучений градієнт функціоналу, то на першій ітерації напрямок руху p_0 вибирають проти градієнта g_0 цієї ітерації:

$$p_0 = g_0 .$$

Для наступних ітерацій напрямок p_k вибирають як лінійну комбінацію векторів g_k і p_{k-1}:

$$p_k = -g_k + \beta_k p_{k-1} ,$$

а вектор параметрів розраховують за формулою:

$$X_{k+1} = X_k + \alpha_k p_k ,$$

Для методів другого порядку розрахунок параметрів на k-му кроці проводять за формулою (метод Ньютона):

$$X_{k+1} = X_k - H_k^{-1} g_k ,$$

де H_k – матриця других часних похідних цільової функції (матриця Тессі);

g_k – вектор градієнта на k-й ітерації. Обчислення матриці Тессі вимагає великих витрат машинного часу, тому її замінюють наближеними виразами (квазіньютонівськи алгоритми).

Градієнтними алгоритмами навчання є:

GD – алгоритм градієнтного спуску;

GDM – алгоритм градієнтного спуску з обуренням;

GDA – алгоритм градієнтного спуску з вибором параметра швидкості **н**алаштування;

Rprop – пороговий алгоритм зворотного поширення помилки;

GDX – алгоритм градієнтного спуску з обуренням і адаптацією параметра швидкості налаштування.

Алгоритмами, заснованими на використанні методу сполучених градієнтів, є:

CGF – алгоритм Флетчера-Рівса;

CGP – алгоритм Полак-Ребейри;

CGB – алгоритм Бієль-Пауелла;

SCG – алгоритм Молер.

Квазіньютонівськими алгоритмами є:

DFGS – алгоритм Бройде, Флетчера, Гольдфарба і Шанно;

OSS – однокроковий алгоритм методу січних площин (алгоритм Баттіні);

LM – алгоритм Левенберга-Марквардта;

BR – алгоритм Левенберга-Марквардта з регуляризації по Байєсу.

В процесі роботи алгоритмів мінімізації функціоналу помилки часто виникає завдання одновимірного пошуку мінімуму вздовж заданого напрямку. Для цих цілей використовується метод золотого перетину GOL, алгоритм Брента BRE, метод половинного ділення та кубічної інтерполяції HYB, алгоритм Чараламбуса CHA і алгоритм перебору з поверненням BAC.

4.2 Завдання для самостійного виконання лабораторної роботи

Завдання 1. Адаптувати параметри одношарової статичної лінійної мережі з двома входами для апроксимації лінійної залежності виду t = 2 p1 + p2, виконавши такі дії:

1. За допомогою конструктора лінійного шару

net = newlin (PR, s, id, lr),

де **PR** – масив розміру Rx2 мінімальних і максимальних значень для **R** векторів входу;

s – число нейронів в шарі;

id – опис лінії затримання на вході шару;

lr – параметр швидкості налаштування, сформувати лінійну мережу:

net = newlin ([– 1 1; –1 1], 1, 0, 0).

2. Підготувати навчальні послідовності у вигляді масивів комірок, використовуючи залежності $t = 2p_1 + p_2$ і чотири пари значень p_1 і p_2 (довільні):

P = {[-1; 1] [-1/3; 1/4] [1/2; 0] [1/6; 2/3]};

T = {-1 -5/12 1 1}.

3. Для угруповання подання навчальної послідовності перетворити масиви комірок в массиви чисел:

P1 = [P{:}], T1 = [T{:}].

4. Виконати команди **net** і **gensim(net)**, Проаналізувати поля обчислювальної моделі і структурну схему мережі і записати в зошит значення полів, що визначають процес налаштування параметрів мережі (ваг і зміщень):

- **net.initFcn** – функція для завдання початкових матриць ваг та векторів зміщень;
- **net.initParam** – набір параметрів для функції **initFcn**, які можна визначити за допомогою команди **help(net.initFcn)**, де **initFcn** – задана функція ініціалізації: **initcon, initlay, initnw, initnwb, initzero**;
- **net.adaptFcn** – функція адаптації нейронної мережі, яка використовується при виклику методу adapt класу network: **adaptwb** або **trains**;
- **net.adaptParam** – параметри функції адаптації, визначаються з допомогою команди **help(net.adaptFcn)**;
- **net.trainFcn** – функція навчання нейронної мережі, яка використовується при виклику методу train класу network: **trainb, trainbfg, traingbr, trainc, traincgb, traincgt, traincgp, trainngd, traingda, traingdm, traingdx, trainlm, trainoss, trainr, trainrp, trainscg**;

• **net.trainParam** – параметри функції навчання, визначаються з допомогою команди help(net.trainFcn);

• **net.performFcn** – функція оцінки якості навчання, яка використовується при виклику методу train: mae, mse, msereg, sse;

• **net.performParam** – параметри функції оцінки якості навчання, визначаються за допомогою команди help(net.performFcn);

• **net.layers{1}.initFcn** – функція ініціалізації параметрів шару: initnw, initwb;

• **net.layers{1}.transferFcn** – функція активації, яка для лінійного шару повинна бути purelin;

• **net.layers{1}.netInputFcn** – функція накопичення для шару: netprod, netsum;

• **net.biases{1}.initFcn** – функція ініціалізації вектора зміщень: initcon, initzero, rands;

• **net.biases{1}.lean** – індикатор налаштування: 0 – за допомогою методу adapt, 1 – за допомогою методу train;

• **net.biases{1}.learnFcn** – функція налаштування вектора зміщень: learncon, learngd, learngdm, learnnp, learnwh;

• **net.biases{1}.learnParam** – параметри функції налаштування, які визначаються з допомогою команди help.(net.biases{1} .learnFcn);

• **net.inputWeights{1, 1}.initFcn** – функція ініціалізації ваг входу: initzero, midpoint, randnc, randnr, rands;

• **net.inputWeights{1,1}.learn** – індикатор настройки: 0 – з допомогою методу adapt, 1 – з допомогою методу train;

• **net.inputWeights{1,1}.learnFcn** – функція налаштування ваг: learngd, learngdm, learnhd, learnis, learnk, learnlv1, learnlv2, learnos, learnnp, learnpn, learnsom, learnnwh;

• **net.inputWeights{1,1}.learnParam** – параметри функції налаштування, що визначаються за допомогою команди **help(net.inputWeights{1,1}.learnParam)**;

• **net.inputWeights {1,1}.weightFcn** – функція для обчислення зважених входів для шару: **dist, dotprod, mandist,negdist, norm-prod**;

• для багатошарових мереж параметри **net.inputWeights{i, j}**, пов'язаний-ні з навчанням такі, як **initFcn, learn, learnFcn, learnParam, weightFcn**, мають той же сенс і можуть приймати такі ж значення, що і відповідні параметри для **net.inputWeights {1,1}**.

5. Виконати один цикл адаптації мережі з нульовим параметром швидкості налаштування:

[net1, a, e,] = adapt (net, P, T);

net1.IW{1,1} % – матриця ваг після адаптації;

a % – чотири значення виходу;

e % – чотири значення помилки.

6. Ініціалізувати нулями ваги входів і зміщень і задати параметри швидкості налаштування для них відповідно 0.2 і 0:

net.IW {1} = [0 0];

net.b {1} = 0;
net.inputWeights {1,1} .learnParam.lr = 0.2;
net.biases {1} .learnParam.lr = 0.

Нульове значення параметра швидкості налаштування для зміщення обумовлено тим, що задана залежність $t = 2p_1 + p_2$ не має постійної складової.

7. Виконати один цикл адаптації до заданого значення параметра швидкості адаптації:

[net1, a, e] = adapt (net, P, T);
net1.IW {1,1} % – значення ваг в мережі net1 змінилися;
a % – чотири значення виходу мережі net1;
e % – чотири значення помилки мережі net1.

8. Виконати адаптацію мережі net за допомогою 30 циклів:

for i = 1:30,
net, a {i}, e {i}] = adapt (net, P, T);
W (i, :) = net.IW{1,1};
end;
cell2mat(a{30}) % – значення виходу на останньому циклі; cell2mat(e{30}) % – значення помилки на останньому циклі; W(30, :) % – ваги після 30 циклів;
mse(cell2mat(e{30})) % – функція помилок: 0.0017176.

Тут **cell2mat** – функція перетворення масиву числових комірок в масив чисел, а **mse** – функція середньоквадратичної помилки.

9. Побудувати графіки залежності значень виходів мережі і вагових коефіцієнтів, а також середньоквадратичної помилки від числа циклів, використовуючи функцію subplot для поділу екрана на вікна:

subplot (3,1,1)
for i = 1: 1: 30, plot (i, cell2mat (a {i}), 'k'),
hold on
end;
xlabel(''), ylabel ('Виходи a (i)')
grid
subplot (3,1,2)
plot (0:30, [[0 0]; W], 'k')
xlabel (''), ylabel ('Ваги входів W (i)')
grid
subplot (3,1,3)
for i = 1:30, E (i) = mse (e {i}); end
semilogy (1:30, E, '+ k')
xlabel ('Цикли'), ylabel('Помилка'), grid

10. Адаптувати розглянуту модель статичної мережі для апроксимації тієї ж залежності і з тими ж вимогами до похибки, використовуючи груповий спосіб подання навчальної послідовності:

P = [-1 -1/3 1/2 1/6; 1 1/4 0 2/3];
T = [-1 -5/12 1 1];
net = newlin([– 1 1;1 1], 1, 0, 0);

```
net = IW{1} = [0 0];      % – присвоювання початкових ваг;
net.l{1} = 0;             % – присвоювання початкового зміщення;
net.inputWeughts {1,1}.learnParam.lr = 0.2;
EE = 10;
i = 1;                    % – для підрахунку кількості циклів;
while EE > 0.0017176
[net, a{i}, e{i}, pf] = adapt (net, P, T);
W (i, :) = net .IW{1,1};
EE = mse(e{i});
ee(i) = EE;
i = i + 1;
end;
```

11. Проаналізувати результати і зробити їх порівняння з результатами для послідовного подання навчальної послідовності:

```
W (63, :)
cell2mat (a{63})
EE = mse (e{63})
mse (e{1})
```

12. Для отриманих результатів побудувати графіки і порівняти їх з попередніми:

```
subplot (3,1,1)
for i = 1: 1: 63, plot (i, cell2mat (a{i}), 'k'),
hold on
end;
xlabel ( ''), ylabel ('Виходи a(i''),  grid
subplot (3,1,2)
plot (0:63, [[0 0]; W], 'k')
xlabel (''), ylabel ('Ваги входів W (i)'), grid
subplot (3,1,3)
semilogy (1:63, ee, '+ k')
xlabel ( 'Цикли'), ylabel ( 'Помилка'), grid
```

Завдання 2. Адаптувати параметри одношарової динамічної мережі з одним входом та однією лінією затримки для апроксимації рекурентного співвідношення y(t) = 2 p (t) + p (t -1), виконавши такі дії:

1. Так як для динамічних мереж груповий спосіб представлення навчальної множини не використовується, підготувати дані для послідовної адаптації у вигляді масивів комірок значень входу і цілі:

```
P = {-1/2 1/3 1/5 1/4);          % – значення входу p(t);
T = {-1 1/6  11/15 7/10};        % – значення мети y(t).
```

2. Виконати адаптацію і побудування графіків

```
net = newlin([– 1 1], 1, [0 1], 0.5);    % – help(newlin);
Pi = {0};                        % – початкова умова для лінії затримки;
net.IW{1} = [0 0];               % – ваги для основного і затриманого входів;
```

```
net.biasConnect = 0;              % – значення зміщення;
EE = 10; i = 1;
while EE > 0.0001
[net, a{i}, e{i}, pf] = adapt(net, P, T);
W (i, :) = net.IW{1,1};
EE = mse(e{i});
ee(i) = EE;
i = i +1;
end;
W (22, :) = a{22};
EE subplot(3,1,1)
for i = 1:1:22, plot (i, cell2mat(a{i}),'k')
hold on
end;
xlabel(''),ylabel('Виходи a(i)'), grid
subplot(3,1,2)
plot(0:22, [[0 0]; W], 'k')
xlabel(''), ylabel('Ваги входів W(i)'), grid
subplot(3,1,3)
semilogy(1:22, ee, '+k')
xlabel('Цикли'), ylabel('Помилка') grid
```

Завдання 3. Навчити нейронну мережу, для якої модель і залежність виходу від входів приведені в завданні 1, виконавши команди і використовуючи послідовний і груповий способи подання навчальної послідовності:

а) для послідовного способу:

```
net = newlin ([-1 1; -1 1], 1, 0, 0);
net.IW{1} = [0 0];
net.b{1} = 0;
P = {[-1 1][– 1/3; 1/4][1/2 0][1/6; 2/3]};
T = {-1 -5/12 1 1};
net.inputWeights{1, 1}.learnParam.lr = 0.2;
net.biases{1}.learnParam.lr = 0;
net.trainParam.epochs = 30;
net1 = train(net, P, T);
W = net1.IW{1}                    % – параметри після навчання:
Y = sim(net1, p)
EE = mse([Y{:}] – [T{:}])         % – помилка 1.3817*e-003
```

б) для групового засобу:

```
P = [-1 -1/3; 1/2 1/6; 11/40 2/3];
T = [-1 -5/12 1 + 1]
net1 = train(net, P, T);
W = net1.IW{1}                          % – параметри після навчання:
Y = sim (net1, P)
```

EE = mse(y-T)% – та ж помилка 1.3817*e-003

Завдання 4. Навчити динамічну лінійну мережу, розглянуту в 2-му завданні і порівняти результати, виконавши наступні команди:

```
net = newlin ([– 1 1], 1, [0 1], 0.5)
Pi = {0}                          % – початкова умова лінії
затримки;
 net.IW {1} = [0 0]               % – початкові ваги входів;
net.biasConnect = 0;              % – зміщення відсутній;
net.trainParam.epochs = 22;
P = {-1/2 1/3 1/5 1/4};           % – вектор входу;
T = {-1 1/6 11/15 7/10};          % – вектор мети;
net1 = train (net, P, T, Pi);     % – навчання мережі;
W = net1.IW {1}                   % – ваги після навчання мережі;
Y = sim (net1, P);                % – моделювання нової мережі;
EE = mse ([Y {:}] – [T {:}])      % – помилка = 3.6514e-005.
```

Завдання 5. Створити і форматувати тришарову мережу з двома входами для подальшого навчання мережі методом зворотного поширення помилки, виконавши наступні дії:

1. Створити шаблон мережі:

```
net5 = network(2, ...            % – число входів;
3, ...                            % – число шарів мережі;
[1; 1; 1], …                      % – зв'язок зміщень;
[0 0 0; 1 0 0; 0 1 0], …          % – зв’язок шарів;
[0 0 1], …                        % – зв'язок виходів;
[0 0 1]);                         % – зв’язок цілей.
```

2. Налаштувати параметри мережі для ініціалізації за допомогою алгоритму Нгуєна-Відроу для забезпечення можливості використання методу зворотного поширення:

```
net5.initFcn = 'initlay';         % – для мережі;
net5.layers{1}.initFcn = 'initnw'; %– для 1-го шару;
net5.layers{2}.initFcn = 'initnw'; %– для 2-го шару;
net5.layers{3}.initFcn = 'initnw'; %– для 3-го шару.
```

3. Проініціалізувати мережу для її подальшого навчання методом зворотного поширення помилки:

```
net5 = init(net5);
net5.IW{1, 1}        % – матриця ваг для 1-го входу;
net5.IW{1, 2}        % – матриця ваг для 2-го входу;
net5.LW{2, 1}        % – матриця ваг для 2-го шару;
net5.LW{3, 2}        % – матриця ваг для 3-го шару;
net5.b{1}            % – матриця зміщення для 1-го шару;
net5.b{2}            % – матриця зміщення для 2-го шару;
net5.b{3}            % – матриця зміщення для 3-го шару.
```

4. Промоделювати мережу з початковими значеннями ваг і зміщень:

P = [0.5 1; 1 0.5]; **% – значення вхідних векторів;**
Y = sim (net5) **% – моделювання мережі.**

Завдання 6. Створити і ініціалізувати тришарову мережу з двома входами для подальшого навчання різними методами, виконавши такі дії:

1. Створити шаблон мережі, скориставшись шаблоном net5:

net6 = net5; **% – створення нової копії мережі;**
net6 = revert(net5); **% – повернення до стандартних параметрів налаштування.**

2. Налаштувати параметри мережі з допомогою функції ініціалізації нульових значень ваг і зміщень **initzero**:

net6.initFcn = ′initlay′;
net6.layers{1}.initFcn = ′initnw′; net6.layers{2}.initFcn = ′initnw′; net6.layers{3}.initFcn = ′initnw′; net6.inputWeights{1 ,1}.initFcn = ′initzero′;
net6.inputWeights{1 ,2}.initFcn = ′initzero′;
net6.layerWeights{2 ,1}.initFcn = ′initzero′;
net6.layerWeights{3 ,2}.initFcn = ′initzero′;
net6.biases{1}.initFcn = ′initzero′; net6.biases{2}.initFcn = ′initzero′; net6.biases{3}.initFcn = ′initzero′; net6.init(net6); **% – ініціалізація мережі.**

3. Видати на екран матриці ваг і зміщення, використовуючи команди 3-го пункту 5-го завдання.

4. Промоделювати мережу і повернути її до початкових значень ваг і зміщень:

net6 = sim(net6);
net6 = revert(net6).

Завдання 7. Створити і ініціалізувати тришарову мережу з двома входами, використовуючи такі функції ініціалізації:

а) **rands** – для задання випадкових ваг і зміщень;
б) **randnc** – для задання випадкової матриці з нормованими стовпцями;
в) **randnv** – для задання випадкової матриці з нормованими рядками;
г) **initcon** – для задання рівних зміщень;
д) **midpoint** – для задання матриці середніх значень.

Для створення і ініціалізації мережі використати команди 6-го завдання, замінюючи в них функцію **initzero** на розглянуті функції ініціалізації, а мережа **net6** – на мережу **net7**.

Завдання 8. Створити двошарову нейронну мережу з прямою передачею сигналу, одним входом, двома нейронами в першому шарі і одним нейроном у другому шарі, налаштувати мережу для навчання з використанням алгоритму градієнтного спуску **GD**, навчити цю мережу і шляхом її моделювання оцінити якість навчання. Порядок виконання завдання наступний:

1. Створити нейронну мережу з прямою передачею сигналу:

net8 = newff([0 5], … **% – діапазони значень входу;**
[2 1], … **% – кількість нейронів в шарах;**

{′tansig′, % – функція активації для 1-го шару; ′logsig′} % – функція активації для 2-го шару;
′traingd′); % – ім'я функції навчання.

2. Переконатися, що ваги і зміщення кожного шару ініціалізовані за допомогою алгоритму Нгуєна-Відроу:

net8.initFcn % – повинно бути ′initlay′; net8.layers{1}.initFcn % – повинно бути ′initnw′;
net8.layers{2}.initFcn % – повинно бути ′initnw′;
net8.IW{1,1} % – вага входу;
net8.LW{2,1} % – ваги для 2-го шару.
net8.b{1}
net8.b{2}

3. Поставити навчальні послідовності входів і цілей T:

P = [0 1 2 3 4 5]; % – вектор входу;
T = [0 0 0 1 1 1]; % – вектор цілей.

4. Видати на екран параметри навчальної функції traingd і їх стандартні значення:

info = traingd(′pdefaults′)
info = epochs: 100 % – максимальна кількість циклів
% – навчання;
show: 25 % – інтервал виведення даних;
goal: 0 % – граничне значення критерію
% навчання;
time: Inf % – максимальний час навчання;
min_grad: 1.0e-006 % – максимальне значення градієнта
% критерія якості;
max_fail: 5 % – максимально допустимий рівень
% перевищення помилки контрольної
% підмножини в порівнянні з
% навчальним.

Процедура навчання припиниться, коли буде виконано одну з наступних умов:

а) значення функції якості стало менше граничного **goal**;
б) градієнт критерію якості став менше значення **min_grad**;
в) досягнуто граничне значення циклів навчання **epochs**;
г) перевищено максимальний час, відпущений на навчання **time**;
д) помилка контрольної підмножини перевищила помилку навчальної більш ніж в **max_fail** разів.

Функція **traingd** передбачає, що функції зважування **dotprod**, накопичення netsum і активації **transig** або **rogsig** мають похідні. Для обчислення похідних критерію якості навчання **perf** по змінним ваги і зміщення використовується метод зворотного поширення.

Відповідно до методу градієнтного спуску вектор змінних отримує наступне прирощення:

dx = lr * dperf / dx,

де **tr** – параметр швидкості налаштування, рівний стандартним 0,01. Функцією одночасного пошуку мінімуму уздовж заданого напрямку в даній мережі є функція **srchbac**.

5. Навчити розглянуту мережу:

net8.trainParam.epochs = 500;
net8.trainParam.90al = 0.01;
[net8, TR] = train(net8,P,T);
TR % – характеристики процедури навчання.

6. Провести моделювання мережі і оцінити якість її навчання:

Ynet8 = sim(net8, P) % – цільові значення.

Завдання 9. Повторити 8-е завдання для наступних функцій навчання: **traingda, traingdm, traingdx, trainrp, traincgf, traincgp, traincgb, trainscg, trainbfg, trainoss, trainlm.** Порівняти отримані результати.

Завдання 10. Створити і навчити мережу для апроксимації синусоїдальної функції, зашумленной нормально розподіленим шумом, виконавши наступні дії:

1. Задати навчальні послідовності:

P = [-1: .05: 1];
T = sin[2*pi*P] + 0.1*randn(size(P));

2. Сформувати мережу з прямою передачею сигналу:

net10 = newff([-1 1], [20 1], {′transig′, ′purelin′}, … ′trainbr′);

3. Налаштувати мережу:

net10.trainParam.epochs = 50;
net10.trainParam.show = 10; % для відображення.

4. Навчити мережу і побудувати графік функції що апроксимується і графік виходу мережі:

net10 = train(net, P,T);
Y = sim(net, P);
plot(P, Y, P, T, ‘+’) % – два графіка.

5. Змінюючи кількість нейронів у першому шарі, досліджувати якість апроксимації.

Завдання 11. Створити мережу і провести її послідовну адаптацію, виконавши наступні команди:

net11 = newff([-1 2; 0 5], [3, 1], … {′tansig′, ′purelin′}, ′traingd′);
net11.inputWeights{1, 1}.learnFcn = ′learngd′;
net11.layerWeights{2, 1}.learnFcn = ′learngd′;
net11.biases{1}.learnFcn = ′learngd′;
net11.biases{2}.learnFcn = ′learngd′;
net11.layerWeights{2, 1}.learnParam.lr = 0.2;
P = [-1 -1 2 2; 0 5 0 5];

T = [-1 -1 1 1];
P = num2cell(P,1);
T = num2cell(T,1);
net11.adaptParam.passes = 50;
[net11, a, e] = adapt(net11, P,T);
a = sim(net11, P) % [-1.02] [-0.99624] [1.0279] [1.0021];
mse(e) % – має бути 5,5909e-004.

Завдання 12. Створити мережу і провести її послідовну адаптацію, використовуючи наступні функції налаштування ваг і зміщень: **learngdm, learnlv1, learnlv2, learnk, learncon, learnis, learnos, learnsom, learnh, learnhd.** Порівняти алгоритм налаштування для однієї і тієї ж навчальної послідовності.

Завдання 13. Створити, навчити і апробувати багатошарову нейронну мережу з прямою передачею сигналу для ухвалення рішення про зарахування до вищого навчального закладу абітурієнтів, які здали вступні іспити з математики, фізики та української мови. Правила прийому такі:

1. Прохідний бал для абітурієнтів, які не мають пільг, має дорівнювати 11;
2. Задовільні оцінки з математики та фізики для цієї категорії абітурієнтів неприпустимі;
3. Абітурієнти, які мають пільги, зараховуються при будь-яких позитивних оцінках за всіма трьома предметами.

Для навчання мережі слід використовувати всі вивчені методи адаптації та навчання та провести аналіз їх ефективності. Слід також визначити мінімальну кількість шарів і нейронів, що забезпечує задовільне рішення поставленої задачі. Для формування навчальної, контрольної і тестової множини побудувати дискретну імітаційну модель, використовуючи інструментальний пакет **Simulink**.

4.3 Зміст звіту

1. Титульний лист.
2. Короткий опис завдання і копії екранів виконаної за завданням роботи.
3. Висновки про виконану роботу.

Контрольні питання

1. Ініціалізація нейронної мережи.
2. Підготування початкового набору даних.
3. Дайте пояснення, що таке помилка навчання.
4. Причини, за яких припиняється процес навчання нейронної мережі.
5. Ефект перенавчання нейронній мережі.
6. Етапи побудови нейронної мережі.
7. Мета навчання нейронної мережі.

8. Градієнтні алгоритми навчання.
9. Алгоритми, засновані на використанні методу сполучених градієнтів.
10. Квазіньютонівськи алгоритми.

Лабораторна робота № 5.
ДОСЛІДЖЕННЯ РАДІАЛЬНИХ БАЗИСНИХ МЕРЕЖ ЗАГАЛЬНОГО ВИГЛЯДУ

Мета роботи: вивчення архітектури радіальних базисних нейронних мереж загального вигляду і спеціальних функцій для їх створення і автоматичної настройки ваг і зміщень, ознайомлення з демонстраційними прикладами і їх скриптами; придбання навичок побудови таких мереж для класифікації векторів і апроксимації функцій.

5.1 Загальні теоретичні відомості

Радіальна, базисна мережа загального вигляду – це двошаровий нейронна мережа з **R** входами, кожен з яких може складатися з декількох елементів. Передавальної функцією нейронів вхідного шару є колоколообразная симетрична функція наступного виду:

$$radbas(n) = e^{-n^2}.$$

Ця функція має максимум, рівний 1, при n = 0 і плавно убуває при збільшенні n, досягаючи значення 0.5 при n = ± 0.833. Передавальної функцією нейронів вихідного шару є лінійна функція **perelin**.

Функція зважування для вхідного шару обчислює евклідову відстань між кожним рядком матриці ваг і кожним стовпцем матриці входів:

$$dist(w_i, p_j) = sqrt\left((w_i - p_j)^2\right).$$

Потім ця величина множиться на зміщення нейрона і надходить на вхід передавальної функції, так що

a{i} = radbas(net.prod(dist(net.IW{1, 1}, p).net.b{i})).

Для нейронів вихідного шару функцією зважування є скалярний добуток **dotprod**, а функцією накопичення – функція підсумовування зважених входів і зваженого зміщення **netsum.**

Для того щоб зрозуміти поведінку радіальної базисної мережі загального вигляду, необхідно простежити проходження вектора входу **p**. При завданні значень елементам вектора входу кожен нейрон вхідного шару видає значення відповідно до того, як близький вектор входу до вектору ваг кожного нейрона. Таким чином, нейрони з векторами ваг, значно відрізняються з вектором входу **p**, матимуть виходи, близькі до 0, і їх вплив на виходи лінійних нейронів вихідного шару буде незначне. Навпаки, вхідний нейрон, ваги якого близькі до вектору **p**, видасть значення, близьке до одиниці.

Для побудови радіальних базисних мереж загального вигляду і автоматичної настройки ваг і зміщень використовуються дві функції **newrbe** і **newrb**. Перша дозволяє побудувати радіальну базисну мережу з нульовою помилкою, друга дозволяє управляти кількістю нейронів вхідного шару. Ці функції мають наступні параметри:

net = newrbe (P, T, SPREAD),

net = newrb (P, T, GOAL, SPREAD),

де **P** – масив розміру **RxQ** вхідних векторів, причому **R** – число елементів вектора входу,

Q – кількість векторів в послідовності;

T – масив розміру **SxQ** з **Q** векторів ланцюга і **S** класів;

SPREAD – параметр впливу, що визначає крутизну функції **radbas**, значення по умовчання якого дорівнює одиниці;

GOAL – середня квадратична помилка, при цьому значення за замовчуванням дорівнює 0.0.

Параметр впливу **SPREAD** істотно впливає на якість апроксимації функції: чим більше його значення, тим більш гладкою буде апроксимація. Занадто велика його значення призведе до того, що для отримання гладкою апроксимації функції, що швидко змінюється, буде потрібна велика кількість нейронів: занадто мале значення параметра **SPREAD** потребують більшої кількості нейронів для апроксимації гладкої функції. Зазвичай параметр впливу **SPREAD** вибирається більшим, ніж крок розбиття інтервалу завдання навчальної послідовності, але меншим розміру самого інтервалу.

Функція newrbe встановлює ваги першого шару рівним **P'**, а зміщення – рівними **0.8326/SPREAD**, в результаті радіальна базисна функція перетинає значення 0.5 при значеннях евклідової відстані ± **SPREAD**. Ваги другого шару **LW{2,1}** і зміщення **b{2}** визначаються шляхом моделювання виходів першого шару **A{1}** та подальшого вирішення системи лінійних рівнянь:

[LW {2,1} b {2}] * [A {1}; ones] = T.

Функція **newrb** формує мережу таким чином. Спочатку перший шар не має нейронів. Мережа моделюється і визначається вектор входу з найбільшою похибкою, додається нейрон з функцією активації **radbas** і вагами, рівними вектору входу, потім обчислюються вагові коефіцієнти лінійного шару, щоб не перевищити середній допустимої квадратичної помилки.

5.2 Завдання для самостійного виконання лабораторної роботи

Завдання 1. Створити радіальну базисну мережу з нульовою помилкою для навчальної послідовності **P = 0:3** і **T = [0,0 2.0 4.1 5.9]**, проаналізувати

структурну схему побудованої мережі і значення параметрів її обчислювальної моделі, виконавши такі дії:

1. Створити радіальну базисну мережу з нульовою помилкою:

```
P = 0:3;
T = [0.0 2.0 4.1 5.9];
net = newrbe(P, T);
```

2. Проаналізувати структурну схему побудованої мережі:

```
gensim(net);
```

3. Проаналізувати параметри обчислювальної моделі мережі:

```
net net.layers{1}.size          % – число нейронів у першому шару;
net.layers{2}.size              % – число нейронів у другому шару;
net.layers{1}.initFcn
net.layers{1}.netInputFcn
net.layers{1}.transferFcn
net.layers{2}.initFcn
net.layers{2}.transferFcn
net.layers{2}.netInputFcn
net.inputWeights{1, 1}.initFcn
net.inputWeights{1, 1}.weightFcn
net.inputWeights{2, 1}.initFcn
net.inputWeights{2, 1}.weightFcn
net.inputWeights{1, 1}.learnFcn
net.IW{1, 1}, net.b{1} net.LW{2, 1}, net.b{2}
net.inputWeights{1, 1}, net.biases{1}
net.inputWeights{2, 1}, net.biases{2}
```

4. Виконати моделювання мережі і побудувати графіки:

```
plot(P, T, '*r', 'MarkerSize', 2, 'LineWidth', 2)
hold on
V=sim(net, P);
plot(P, V, '*b', 'MarkerSize', 9, 'LineWidth', 2)
P1=0.5:2.5;
Y=sim(net, P1);
plot(P1, Y, '*k', 'MarkerSize', 10, 'LineWidth', 2)
```

Завдання 2. Створити радіальну базисну мережу для навчальної послідовності **P = 0:3** і **T = [0,0 2.0 4.1 5.9]** при середньою квадратичною помилку 0.1, проаналізувати структурну схему побудованої мережі і значення параметрів її обчислювальної моделі, виконавши дії завдання і замінивши в командах функцію **newrbe(P, T)** на **newrb(P, T, 0.1)**.

Завдання 3. Створити радіальну базисну мережу з нульовою помилкою для великого числа значень входу і цілі, виконавши такі дії:

1. Поставити навчальну послідовність і побудувати для неї графік:

```
P = -1:0.1:1;
```

T = [-0.9602 -0.5770 -0.0729 0.3771 0.6405 0.6600 …
0.4609 0.1336 -0.2013 -0.4344 -0.5000 -0.3930…
-0,1647 0.988 0.3072 0.3960 0.3449 0.1816 – 0.0312 -0.2189 -0.3201];
plot(P, T, '*r', 'MarkerSize', 4, 'LineWidth', 2)
hold on
2. Створити мережу і визначити число нейронів у шарах:
net = newrbe(P, T);
net.layers{1}.size % – у першому шарі – 21 нейрон;
plot(P, V, '*b', 'MarkerSize', 10, 'LineWidth', 2).

Завдання 4. Створити радіальну базисну мережу для великого числа значень входу і цілі при середньою квадратичною помилку 0.01, виконавши дії завдання 3 і замінивши функцію **newrbe(P, T)** на функцію **newrb(P, T, 0.01)**.

Завдання 5. Провести апроксимацію функції f (x) за допомогою радіальних базисних функцій φi (x) у вигляді наступного розкладу в ряд:

$$f(x) = \sum_{i=1}^{n} \alpha_i \phi_i(x),$$

тут коефіцієнти розкладання по радіальних базисних функціях:
p = -3:0.1:3;
a1 = radbas(P);
a2 = padbas(P-1.5);
a3 = radbas(P+2);
a = a1 + a2 + 0.5*a3;
plot(P, a1, p, a2, p, 0.5*a3, p, a).
Як видно з графіка, розкладання по радіальних базисних функціях забезпечує необхідну гладкість. Розкладання зазначеного виду може бути реалізовано на двошарової нейронної мережі, перший шар якої складається з радіальних базисних нейронів, а другий – з єдиного нейрона з лінійною характеристикою, на якому підсумовуються виходи нейронів першого шару.

Завдання 6. Дослідити гладкість апроксимації при наступних значеннях параметра **SPREAD: 1, 0.01** і **12,** використовуючи команди з четвертого завдання.

5.3 Зміст звіту

1. Титульний лист.
2. Короткий опис завдання і копії екранів виконаної за завданням роботи.
3. Висновки про виконану роботу.

Контрольні питання

1. Скільки шарів містить радіальна базисна мережа загального вигляду?
2. Функція активації нейронів вхідного шару радиальної базисної мережи.
3. Передавальна функція нейронів вихідного шару радиальної базисної мережи.
4. Функція зважування для вхідного шару радиальної базисної мережи.
5. Опишіть типову поведінку радіальної базисної мережі загального вигляду.
6. Яка величина надходить на вхід передавальної функції радіальної базисної мережі загального вигляду?
7. Функціся зважування для нейронів вихідного шару радіальної базисної мережі загального вигляду.
8. Які функції використовують для побудови радіальних базисних мереж загального вигляду і автоматичної настройки ваг і зміщень?

Лабораторна робота № 6.
ДОСЛІДЖЕННЯ МЕРЕЖ ХОПФІЛДА

Мета роботи: вивчення архітектури рекурентних нейронних мереж Хопфилда і спеціальних функцій для їх створення, зважування входів, накопичення і активізації; ознайомлення з демонстраційними прикладами і їх скриптами, а також придбання навичок побудови таких мереж для вирішення завдань розпізнавання образів і створення асоціативної пам'яті.

6.1 Загальні теоретичні відомості

Мережа Хопфілда відноситься до класу зворотних нейронних мереж. Вона має один нейронний шар з функціями зважування **dotprod**, накопичення **netsum** і лінійною обмеженою функцією активізації **satlins**. Шар охоплений динамічної зворотним зв'язком з вагами **LW{1,1}** і має зміщення.

Мережі Хопфілда володіють тією відмінною властивістю, що при надходженні на вхід деякого вектора ознак у вигляді початкових умов, вони за кінцеве число тактів часу приходять в стан стійкої рівноваги, залежить від вхідного вектора. Це дозволяє асоціювати вхідний вектор з деяким станом мережі, або об'єктом предметної області. Стану стійкої рівноваги називаються також атракторами мережі. Крім цільових аттракторов в мережі можуть мати місце паразитні, які не відповідають ніяким векторах входу. При створенні мережі Хопфілда спеціальні алгоритми синтезу зводять до мінімуму число таких паразитних точок рівноваги і забезпечують максимальний розмір області тяжіння для точок рівноваги.

Функція для створення мережі Хопфілда має вигляд:

net=newhop(T),

де **T** – масив розміру **R * Q**, який об'єднує **Q** цільових векторів зі значеннями **+1** або **-1** для елементів;

R – число елементів вектора входу.

Після того як початкові умови задані у вигляді масиву **T**, що визначає ряд цільових вершин замкнутого гіперкуба, мережа для кожної вершини генерує вихід, який по зворотного зв'язку подається на вхід. Цей процес при створенні мережі повторюється багато разів, поки її вихід встановиться в положення рівноваги для кожної з цільових вершин. При подачі потім довільного вхідного вектора мережу Хопфілда переходить в результаті рекурсивного процесу до однієї з точок рівноваги, найбільш близької до вхідного сигналу.

Динамічна модель рекуррентного шару однієї з модифікацій мережі Хопфілда описується наступним чином:

$$\begin{cases} a^1(k) = satlins(LW^{11}a^1(k-1) + b^1); \\ a^1(0) = p. \end{cases}$$

Коли мережа Хопфілда спроектована, вона може бути перевірена з одним або більшим числом векторів входу. Досить імовірно, що вектори входу, близькі до цільових точок рівноваги, знайдуть свої цілі. Здатність мережі Хопфілда швидко обробляти набори векторів входу дозволяє перевірити мережу за відносно короткий час. Спочатку слід перевірити, що точки рівноваги цільових векторів дійсно належать вершин гіперкуба, а потім можна визначити області тяжіння цих точок і виявити паразитні точки рівноваги.

6.2 Завдання для самостійного виконання лабораторної роботи

Завдання 1. Створити мережу Хопфілда з двома стійкими точками в тривимірному просторі, виконавши наступні команди:

```
T =[-1 -1 1 ; 1 -1 1]';          % – цільові вершини
net =newhop(T);                  % – створення мережі
gensim(net)                      % – параметри та структура мережі
Ai = T ;                         % – початкові умови для ліній затримки
Y = sim(net,2,[], Ai);           % – мережа стійка
Ai ={[-0.9 ; -0.8 ; 0.7]};
Y=sim(net,{1 5},{}, Ai);         % – мережа забезпечила перехід до
                                 % стійкого стану
```

Завдання 2. Створити мережу Хопфілда з чотирма стійкими точками в двомірному просторі, виконавши наступні команди:

```
T = [1 -1; -1 1; 1 1; -1 -1]'
plot(T(1, : ),T(2, : ), '*r')    % – точка рівноваги
axis([-1.1 1.1 -1.1 1.1]);
title('Точки рівноваги проектуємої мережі');
xlabel('a(1)'), ylabel('a(2)')
net =newhop(T);
W=net.LW{1,1} b=net.b{1,1}
Ai = T;
Y=sim(net,4,[],Ai)
plot(T(1, : ),T(2, : ), '*r'), hold on
axis([-1.1 1.1 -1.1 1.1])
xlabel('a(1)'), ylabel('a(2)')
new=newhop(T);
[Y,Pf,Af]=sim(net,4,[],T);
for i=1:25
a={rands(2,1)};
[Y,Pf,Af]=sim(net,{1,20},{},a);
record=[cell2mat(a),cell2mat(Y)]
start=cell2mat(a);
```

```
plot(start(1,1), start(2,1),'kx',record(1,:),record(2,:))
end
```

6.3 Зміст звіту

1. Титульний лист.
2. Короткий опис завдання і копії екранів виконаної за завданням роботи.
3. Висновки про виконану роботу.

Контрольні питання

1. До якого класу відноситься нейронна мережа Хопфілда?
2. Які компоненти містить мережа Хопфілда?
3. Назвіть відмінні властивості мережи Хопфілда.
4. Що дозволяє асоціювати вхідний вектор мережи Хопфілда з деяким станом мережі, або об'єктом якоїсь предметної області?
5. Поясніть, що таке атрактори мережі.
6. Чив відрізняються цільові та паразитні атрактори мережи?
7. Який вигляд має функція створення мережі Хопфілда?
8. Яким чином може бути описана модель рекуррентного шару мережі Хопфілда?

ПЕРЕЛІК РЕКОМЕНДОВАНИХ ДЖЕРЕЛ

Базові

1. Stephen Marsland. Machine Learning: An Algorithmic Perspective / Stephen Marsland. – 2015. – 452 p.,
2. Yaser S. Abu-Mostafa. Learning from data / Yaser S. Abu-Mostafa. – 2017. – 215 p.
3. Deep Learning / Ian Goodfellow, Yoshua Bengio, Aaron Courville. – 2016. – 800 p.
4. LISA lab, University of Montreal Deep Learning Tutorial. – 2015. – 173 p.
5. Neil Wilkins Artificial Intelligence: An Essential Beginner's Guide to AI, Machine Learning, Robotics, The Internet of Things, Neural Networks, Deep Learning, Reinforcement Learning, and Our Future Paperback. – Publisher: Bravex Publications, 2019,112 p.
6. Mariya Yao Applied Artificial Intelligence: A Handbook For Business Leaders Kindle Edition, Publisher: TOPBOTS, 2018, 246 p.
7. Richie Dorsey Machine Learning for Beginners: A Complete Guide for Getting Started with Machine Learning Kindle Edition.- Publisher: Amazon.com Services LLC, 2019, 162 p.
8. Svein Linge, Hans Petter Langtangen Programming for Computations - Python: A Gentle Introduction to Numerical Simulations with Python (Texts in Computational Science and Engineering Book 15) 1st ed. 2016 Edition, Kindle Edition. – Publisher: Springer; 1st ed. 2016 edition (July 25, 2016), 232 pages
9. Stephen Richard Data Analysis from Scratch with Python: The Complete Beginner's Guide for Machine Learning Techniques and A Step By Step NLP using Python Guide To Expert (Including Programming Interview Questions) Kindle Edition.- Publisher: Amazon.com Services LLC, 2019, 134 p.

10. Допоміжні

11. Tom M. Mitchell. Machine Learning [Електронний ресурс] / Tom M. Mitchell. – Режим доступу: http://www.cs.cmu.edu/~tom/mlbook.html
12. Feldman, R.The text mining handbook: advanced approaches in analyzing unstructured data [Текст] / R. Feldman, J. Sanger. – Cambridge University Press, 2007. – 410 p.
13. Люгер, Д. Искусственный интеллект: стратегии и методы решения сложных проблем [Текст] / Д. Люгер. – Издательский дом «Вильямс», 4е изд. М.: – 2003. – 864 с.

14. Рассел, С. Искусственный интеллект. Современный подход [Текст] / С.Рассел, П.Норвиг, 2-е изд.: Пер. с англ. – М.: Издательский дом «Вильямс», 2006. – 1408 с.
15. Bezdek, J.C. Pattern Recognition with Fuzzy Objective Function Algorithms [Текст] / J.C. Bezdek // N.Y.: Plenum Press, 1981. – 272 p.
16. Люгер, Дж.О. Искусственный интеллект: стратегии и методы решения сложных проблем / Дж.О. Люгер. - М.: Диалектика, 2016. - 864 с.
17. Нильсон, Н. Принципы искусственного интеллекта / Н. Нильсон. - М.: Радио и связь, 2015. - 373 с.
18. Рассел, С. Искусственный интеллект: современный подход / С. Рассел, П. Норвиг. - М.: Вильямс, 2016. - 578 с.
19. Слэйгл, Дж. Искусственный интеллект / Дж. Слэйгл. - М.: Мир, 2016. - 320 с.
20. Тей, А. Логический подход к искусственному интеллекту / А. Тей, П. Грибомон, и др.. - М.: Мир, 2015. - 432 с.
21. Акинин, М.В. Нейросетевые системы искусственного интеллекта в задачах обработки изображений / М.В. Акинин, М.Б. Никифоров, А.И. Таганов. - М.: ГЛТ , 2016. - 152 с.
22. Raheem, N. (2019). Big Data : A Tutorial-Based Approach (Vol. First edition). Boca Raton: Chapman and Hall/CRC.
23. Акинин, М. В. Нейросетевые системы искусственного интеллекта в задачах обработки изображений / М.В. Акинин, М.Б. Никифоров, А.И. Таганов. - М.: РиС, 2016. - 152 с.

24. Інформаційні ресурси

25. Towards Data Science, https://towardsdatascience.com/
26. DataSet, https://www.kaggle.com/kernels.
27. Scikit-learn tutorial: statistical-learning for sientific data processing, http:// gael-varoquaux.info/scikit-learn-tutorial/
28. Andrej Karpathy. The unreasonable effectiveness of recurrent neural networks. http://karpathy.github.io/2015/05/21/ rnn-effectiveness/, 2015. Online; accessed 11-December-16
29. Andrew Moore. Statistical Data Mining Tutorials [http://www.autonlab.org/tutorials/]
30. Ускоренный курс машинного обучения с API TensorFlowhttps, http://developers.google.com/machine-learning/crash-course/

This project has been funded with support from the European Commission. This publication / communication reflects the views only of the author, and the Commission cannot be held responsible for any use which may be made of the information contained there in.