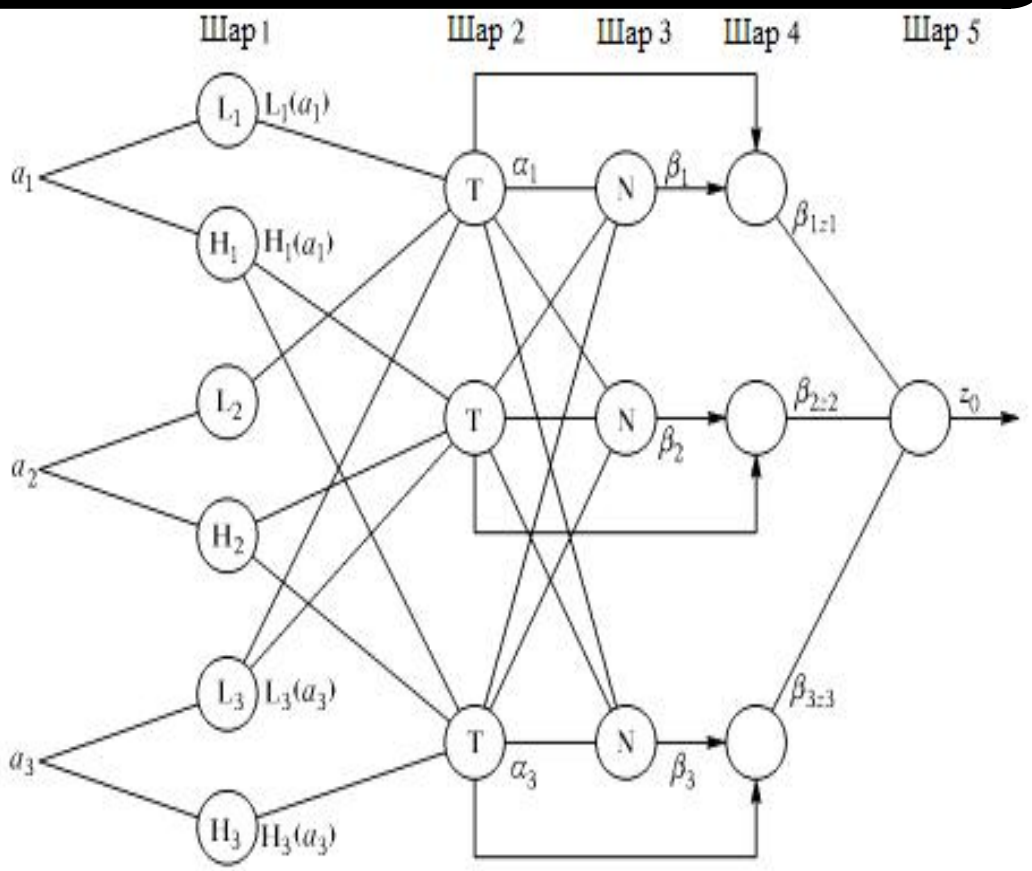
І.М. Пістунов, О.П. Антонюк

НЕЙРОМЕРЕЖЕВІ ТЕХНОЛОГІЇ В ЕКОНОМІЦІ ТА ФІНАНСАХ

З РОЗРАХУНКАМИ НА КОМП'ЮТЕРІ



Дніпропетровськ
2014

Міністерство освіти і науки України
ДЕРЖАВНИЙ ВИЩИЙ НАВЧАЛЬНИЙ ЗАКЛАД
«НАЦІОНАЛЬНИЙ ГІРНИЧИЙ УНІВЕРСИТЕТ»

І.М. Пістунов, О.П. Антонюк

НЕЙРОМЕРЕЖЕВІ ТЕХНОЛОГІЇ В ЕКОНОМІЦІ ТА ФІНАНСАХ З РОЗРАХУНКАМИ НА КОМП'ЮТЕРІ

Навчальний посібник

Дніпропетровськ
НГУ
2014

УДК 004.032.26.330:336(я75.8)
БКК 32.818:65с(я73)
П34

Рекомендовано вченою радою як посібник для студентів спеціальності 7, 8.030502 «Економічна кібернетика» (Протокол № 6 від 01.07.14).

Рецензенти:

Мороз Б.І. – д-р. техн. наук, проф., начальник кафедри інформаційних систем і технологій (Академія митної служби України);

Васильєва Н.К – д-р. екон. наук, доц., завідувач кафедри інформаційних систем і технологій Дніпропетровського державного аграрного університету.

Пістунов І.М.

П34 Нейромережеві технології економіці та фінансах з розрахунками на комп'ютері [Електронний ресурс]: навч. посібн. / І.М. Пістунов, О.П. Антонюк ; Нац. гірн. ун-т. – Електрон. текст. дані. – Д. : НГУ, 2014. – 105 с. – Режим доступу: http://pistunovi.narod.ru/N_M.pdf (дата звернення: 11.07.2014). – Назва з екрана.

Зміст відповідає освітньо-професійній програмі підготовки спеціалістів і магістрів з напряму 7.030502 та 8.030502 «Економічна кібернетика» та програмі дисципліни «Нейронні мережі».

У посібнику подано загальні відомості про нейронні сітки та мережі, їх структуру, методи навчання та сфери їх застосування в діяльності: комерційних організацій, банків та інших фінансових структур.

Велику увагу приділено поясненню можливостей прикладного пакета програм STATISTICA Neural Networks, який дозволяє реалізувати всі сучасні методи застосування нейронних сіток.

У виданні також містяться завдання для самостійного виконання, тому воно може слугувати і як посібник для практичних чи лабораторних занять із застосуванням комп'ютерної техніки.

Адресовано студентам вищих навчальних закладів, може бути корисним для фінансистів, економістів, плановиків, менеджерів та маркетологів.

УДК 004.032.26.330:336(я75.8
БКК 32.818:65с(я73)

ЗМІСТ

ВСТУП 5

1. ТЕОРІЯ НЕЙРОННИХ СІТОК 7

1.1. Структура 7

1.2. Навчання 15
1.2.1. Навчальний алгоритм зворотного поширення 16
1.2.2. Мережі зустрічного поширення 22
1.2.3. Приклад розрахунку вагових коефіцієнтів 30

1.3. Теорія застосування нейронних сіток 34
1.3.1. Автоматична класифікація 34
1.3.2. Прогнозування одновимірної функції 35
1.3.3. Апроксимація багатовимірної функції 36

1.4. Індивідуальне завдання № 1 37

2. ПРАКТИЧНЕ ЗАСТОСУВАННЕЯ НЕЙРОННИХ СІТОК 39

2.1. Порядок застосування програмного пакета Matlab 6 39
2.2.1 Налаштування Matlab 6 39
2.1.2 GUI- інтерфейс для ППП NNT 44

2.2. Індивідуальне завдання № 2 53

2.3. Використання GUI-інтерфейсу пакета нейронних мереж 57
2.3.1. Створення нейронної мережі 57
2.3.2. Навчання нейронної мережі 59
2.3.3. Робота зі створеної мережею 60
2.3.4. Індивідуальне завдання № 3 61
2.3.5.Індивідуальне завдання №4 65

2.4. Прогнозування з використанням нейронної мережі. Тестування нейронної мережі. Підбір параметрів 67

2.4.1. Початок роботи з прогнозування 67
2.4.2. Приклад прогнозування курсу валют 73
2.4.3. Індивідуальне завдання № 5 75

2.5. Кластеризація та класифікація з використанням нейронної мережі 76
2.5.1. Кластеризація за допомогою функції «Clustering Tool» 77
2.5.2. Кластеризація за допомогою самоорганізованої мережі Кохонена 80
2.5.3. Приклад кластеризації даних з використанням нейронної мережі прямого поширення 82
2.5.4. Робота з LVQ нейронною мережею 85
2.5.5. Індивідуальне завдання № 6 87

2.6. Визначення значення залежної змінної за вхідними параметрами 87
2.6.1. Індивідуальне завдання № 7 92

2.7. Розпізнавання образів 92

ВИСНОВКИ 101

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 102

ПРЕДМЕТНИЙ ПОКАЖЧИК 105

ВСТУП

Нейронні сітки – схеми з’єднання однорідних елементів – нейронів. Ця назва стосується не тільки біологічних об’єктів, але й їх математичних моделей. Ці моделі можуть розв’язувати задачі: логічні, стохастичні, керування, моделювання, прогнозування. Таку характеристику на с. 147 – 148 другого тому дає перша у світі Енциклопедія кібернетики, видана у Києві в 1973 році.

В наш час цей термін застосовується для позначення структури математичної моделі, яка, в свою чергу називається **нейронною мережею.**

При моделюванні економічних процесів однією з найбільш складних задач є вибір виду функції апроксимації. Це пояснюється тим, що часто характер залежності одного економічного параметра від іншого є невідомим. Фактично, нам необхідно, знаючи певний набір значень вхідних параметрів, віднести значення вихідного параметра до певного класу чи значення.

Для вирішення поставленої задачі найбільш прийнятним є використання такого математичного методу як нейронні сітки, які стали однією з математичних основ створення інформаційних систем, які володіють штучним інтелектом (ІСШІ).

ІСШІ являє собою досить нову і перспективну технологію, що сприяє новим підходам до вирішення цілого ряду завдань в науці, техніці , економіці та бізнесі. Вони увійшли в практику там , де потрібно вирішувати неформалізовані завдання, для яких немає відомих алгоритмів рішення. Для таких задач нейромережеві технології по ефективності можуть істотно перевищувати традиційні методи .

ІСШІ найкращим чином проявляють себе при великій кількості даних , для яких характерні неявні взаємозалежності і закономірності. Тут вони автоматично витягають і використовують різні залежності, приховані в даних. До завдань, ефективно розв'язуються за допомогою ІСШІ, в даний час відносяться наступні:

- апроксимація функцій (побудова моделей для залежностей, що погано формалізуються), наприклад, в економіці оцінка вартості нерухомості, контроль якості випущеної продукції;
- класифікація образів (визначення приналежності вхідного образу попередньо визначеним класам), наприклад , оцінки кредитоспроможності, оцінка ймовірності банкрутства;
- кластеризація (класифікація образів «без вчителя»), наприклад, аналіз діяльності конкуруючих фірм, стиснення інформації в інформаційних сховищах;
- прогнозування , наприклад , економічних параметрів і фондових індексів, обсягів продажу;

 асоціативна пам'ять (пам'ять, що адресується за змістом), наприклад, для створення сховищ даних, мультимедійних баз даних;

 оптимізація (знаходження рішення, що задовольняє системі обмежень і максимізує або мінімізує цільову функцію), наприклад, прибутковості роботи компанії, вирішує задачі ціноутворення;

 управління , наприклад , товарними запасами, динамічними об'єктами.

І якщо значна частина задач моделювання зводиться до подання у вигляді апроксимаційних залежностей, то неронні сітки виконують роль універсального інструменту для як для апроксимації так і для прогнозування функцій декількох змінних.

Вивчивши матеріали цього посібника студенти оволодіють сучасними методами нейрокомп’ютінгу, що дозволить ім на практиці легко вирішувати складні задачі, які в наш час ставить ринкова економіка перед фахівцями з економічної кібернетики.

Після кожного розділу або пункту подані індивідуальні завдання, які студенти мають виконати протягом часу на засвоєння предмету «Нейронні мережі».

Індивідуальні завдання оформляються як документ, який подається в електронному вигляді, вміщеним на будь-який носій інформації. Формат електронного документу має відповідати електронному процесору Calc Open Office або Excel Microsoft Office або MatLab 6.

Титульний лист оформлюється згідно прикладу, поданому нижче.

Міністерство освіти і науки України
Державний вищий навчальний заклад
«Національний гірничий університет»
Кафедра економічної кібернетики та інформаційних технологій

ІНДИВІДУАЛЬНА РОБОТА З ДИСЦИПЛІНИ
«НЕЙРОННІ МЕРЕЖІ»

Розроби(ла)в ст. гр. ЕК-09-1 Косач-Квітка Л.П.
Варіант № 5
Прийняв проф., д.т.н. Пістунов І.М.

Дніпропетровськ
2014

Кожне виконане завдання повинно містити опис задачі, початкові значення змінних, які обираються за номером по списку студентської групи, вирішення та висновки щодо отриманих результатів.

1. ТЕОРІЯ НЕЙРОННИХ СІТОК

Вивчивши матеріали цього розділу студенти опанують сутність нейронних сіток, порядок їх створення, методи знайдення вагових коефіцієнтів, перевірять засвоєні методи на електронних таблицях Excel.

1.1. Структура

Нейронні сітки – це сітки, що складаються зі зв'язаних між собою простих елементів – формальних нейронів. Ядром використовуваних представлень є ідея про те, що нейрони можна моделювати досить простими формулами, а вся складність процесу моделювання визначається зв'язками між нейронами. Кожен зв'язок представляється як зовсім простий елемент, що служить для передачі сигналу. Для опису кожного нейрону використовується проста і одна й та сама функція, що називається передавальною або функцією активації нейрону.

На рис. 1.1 представлено типову схему нейрону, з яких потім формується нейронна сітка. На ньому видно, що зі входів O_1, O_2, O_3 подається сигнал, який змінюється за допомогою вагових коефіцієнтів W_1, W_2, W_3 шляхом їх перемноження зі вхідними сигналами. Перетворені сигнали надходять до суматора **Σ**, де складаються за алгебраїчними правилами. Вихід суматора, позначений як ***NET***, є входом до функції активації ***F***, де він перетворюється за формулою, яка міститься у цьому блоці. Вихід нейрону позначено як ***OUT***.

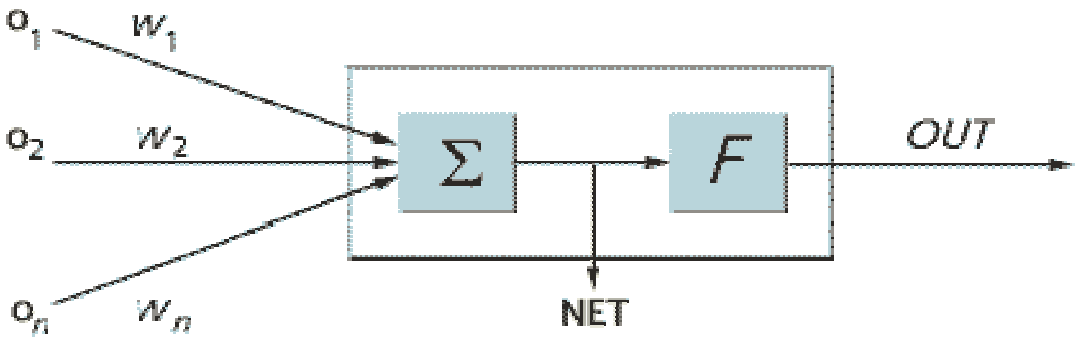


Рис.1.1. Схема нейрону

Очевидно, що кількість вхідних сигналів до нейрона не обмежується трьома, як на прикладі, а залежить тільки від бажання та обізнаності дослідника.

Наведемо систему формул, що пояснюють принцип перетворення вхідних сигналів в одному нейроні.

$$NET = \sum_{i=1}^{N} O_i W_i; \quad OUT = F(NET). \tag{1.1}$$

Звідкіля слідує, що $$OUT = F(\sum_{i=1}^{N} O_i W_i). \tag{1.2}$$

В деяких випадках модель (1.2) змінюється за рахунок додавання так званого порогу збуджуваності θ. Це означає, що вхідний сигнал менший величини θ дорівнює нулю. Тоді (1.2) можна переписати у вигляді

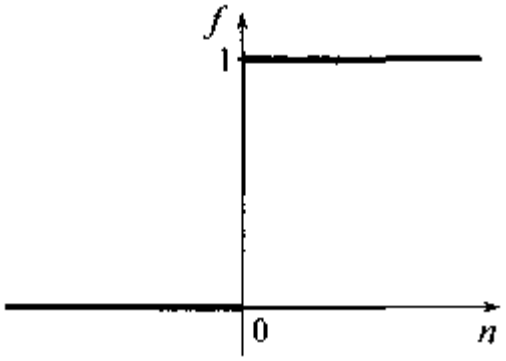
$$OUT = F\left(\sum_{i=1}^{N} (O_i - \theta_i) W_i \right). \tag{1.3}$$

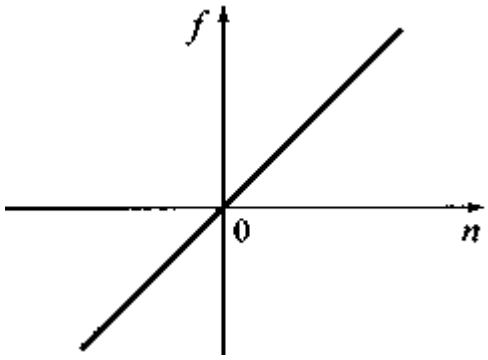
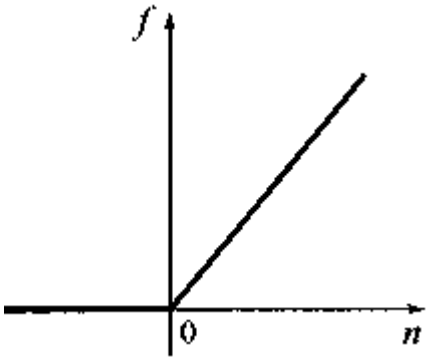
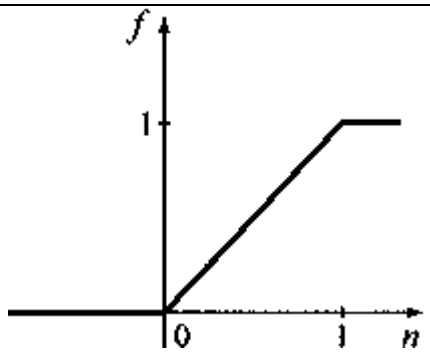
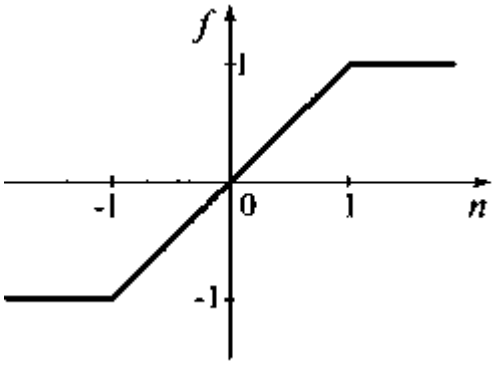
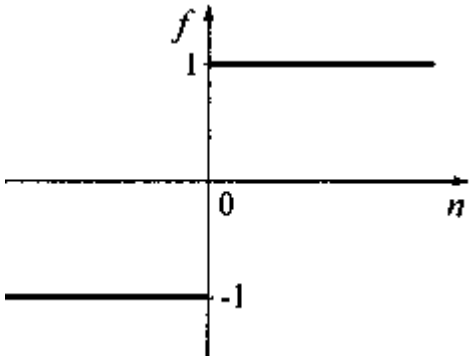
Функції перетворення, що їх застосовують у нейронних сітках наведені у табл. 1.1. В таблиці прийнять такі умовні позначення: f – вихід функції активації, n – вхід функції активації.

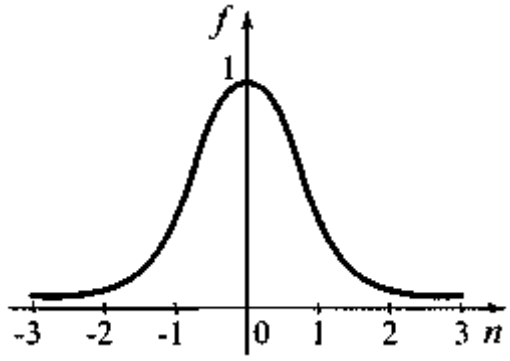
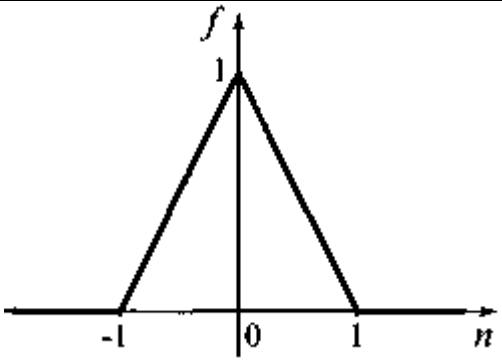
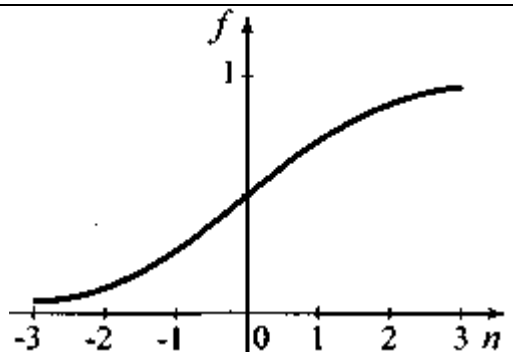
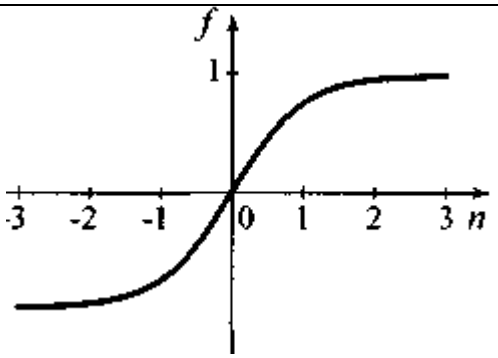
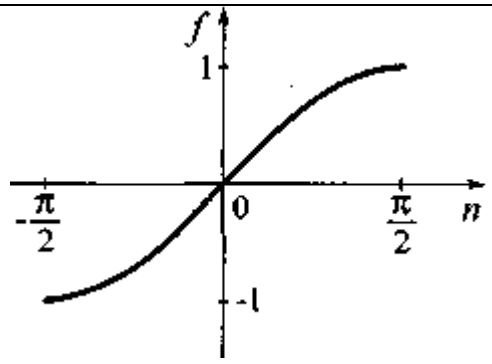
Розглянувши наведені в табл. 1.1 функції, ми бачимо, що всі вони на виході дають одиницю. При необхідності отримати на виході сигнал, відмінний від діапазону [0; 1], його множать на відповідний перехідний коефіцієнт.

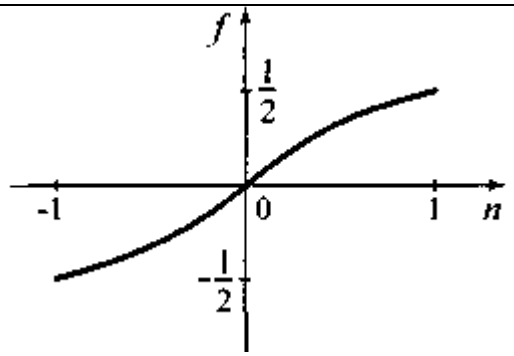
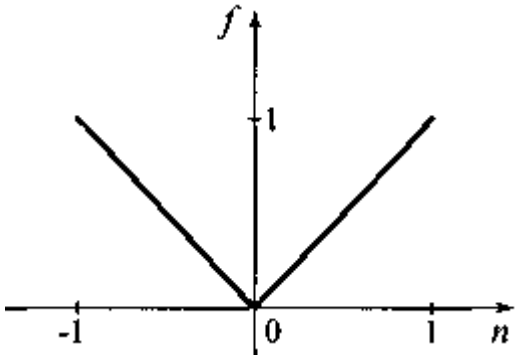
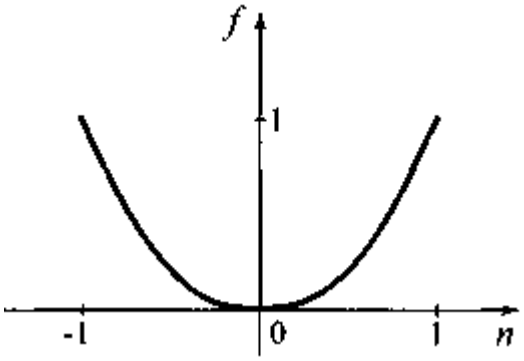
Таблиця 1.1

Типи передавальних (активації) функцій,
що застосовуються в моделях нейронів

Назва функції	Аналітичний вигляд	Геометричне зображення
1	2	3
Хевісайда (ступінчаста функція)	$f(n) = \begin{cases} 0, n < 0 \\ 1, n \geq 0 \end{cases}$	

1	2	3
Лінійна	$f(n) = n$	
Позитивна лінійна (напів-лінійна)	$f(n) = \begin{cases} 0, n < 0 \\ n, n \geq 0 \end{cases}$	
Лінійна з обмеженнями	$f(n) = \begin{cases} 0, n < 0 \\ n, 0 \leq n \leq 1 \\ 1, n > 1 \end{cases}$	
Симетрична лінійна з обмеженнями	$f(n) = \begin{cases} -1, n < -1 \\ n, -1 \leq n \leq 1 \\ 1, n > 1 \end{cases}$	
Симетрична з жорсткими обмеженнями	$f(n) = \begin{cases} -1, n < 0 \\ 1, n \geq 0 \end{cases}$	

Експонен-ційна (радіальна)	$f(n) = e^{-n^2}$	
Трикутна	$f(n) = \begin{cases} 0, n < -1 \\ 1 - \lvert n \rvert, -1 \le n \le 1 \\ 0, n > 1 \end{cases}$	
Логістична	$f(n) = \frac{1}{1 + e^{-n}}$	
Гіперболічна (танген-ціальна)	$f(n) = \frac{2}{1 + e^{-2n}} - 1$	
Синусоїдальна	$f(n) = \sin(n)$	

Раціональна (гіпер-болічна)	$f(n) = \frac{n}{1+\lvert n \rvert}$	
Модульна	$f(n) = \lvert n \rvert$	
Квадратична	$f(n) = n^2$	

Коли функція активації вибрана, починається формування структури нейронної сітки.

Створення структури нейронної сітки починається з визначення кількості шарів нейронів. Шаром називається група нейронів, що отримують однакові вхідні сигнали (рис. 1.2). Таким прийомом моделюється нервова система живої істоти, де кожен сигнал завжди сприймається групою нейронів, але вихідні сигнали з них різняться між собою. Різниця обумовлена різними значеннями вагових коефіцієнтів.

Тобто, нейронний шар можна тлумачити як паралельне з'єднання нейронів.

Матриця W, розміром SxR, елементами якої є вагові коефіцієнти W_{ij} називається ваговою матрицею. В разі, якщо якісь зв'язки вхідних сигналів з нейронами відсутні, відповідний ваговий коефіцієнт у матриці прирівнюється нулю.

$$W = (w_{ij})_{S\times R} = \begin{bmatrix} w_{11} & w_{12} & \dots & w_{1R} \\ w_{21} & w_{22} & \dots & w_{2R} \\ \dots & \dots & \dots & \dots \\ w_{S1} & w_{S2} & \dots & w_{SR} \end{bmatrix} = \begin{bmatrix} \overline{w}_1 \\ \overline{w}_2 \\ \dots \\ \overline{w}_S \end{bmatrix}. \qquad (1.4)$$

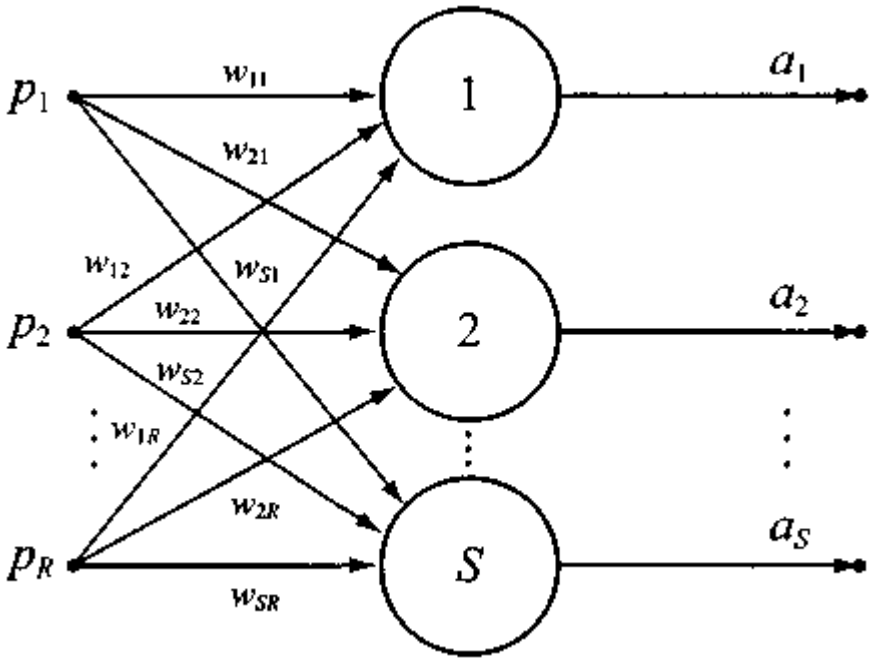


Рис. 1.2. Схема одного шару нейронної сітки з прямими зв'язками:

– $p_1, \dots p_R$ – вхідні сигнали шару нейронної сітки;

– W_{ij} – вагові коефіцієнти i-го нейрона для j-го вхідного сигналу;

– 1, 2…S – номери нейронів у шарі;

– a_i – вихідні сигнали і-го нейрону.

Наступним етапом створення нейронної сітки є визначення кількості шарів та кількості нейронів у кожному шарі. Частіше дослідники обирають поступове зменшення кількості нейронів від першого шару до останнього. Інколи останній шар представляє собою один нейрон.

Далі визначається система зв'язків поміж нейронами різних шарів. В деяких випадках виходи нейронів одного шару повертаються на вхід нейронів попереднього шару (зворотний зв'язок) або ці виходи спрямовуються на входи нейронів, що розташовані у наступних шарах, але з пропуском найближчого шару.

На рис. 1.3 показано приклад однієї з реальних нейронних сіток. В ній вхідні сигнали потрапляють в нейрони першого шару попарно: перший (a_1) – в

L_1 та H_1, другий (a_2) – в L_2 та H_2, третій (a_3) – в L_3 та H_3. Вихід із них також надходить на усі нейрони другого шару, але кількість останніх уже менше, замість шести у першому шарі, у другому тільки три. Вихід із другого шару також надходить до трьох нейронів третього шару, але водночас і до трьох нейронів четвертого шару. Останній шар нейронної сітки представлений тільки одним нейроном.

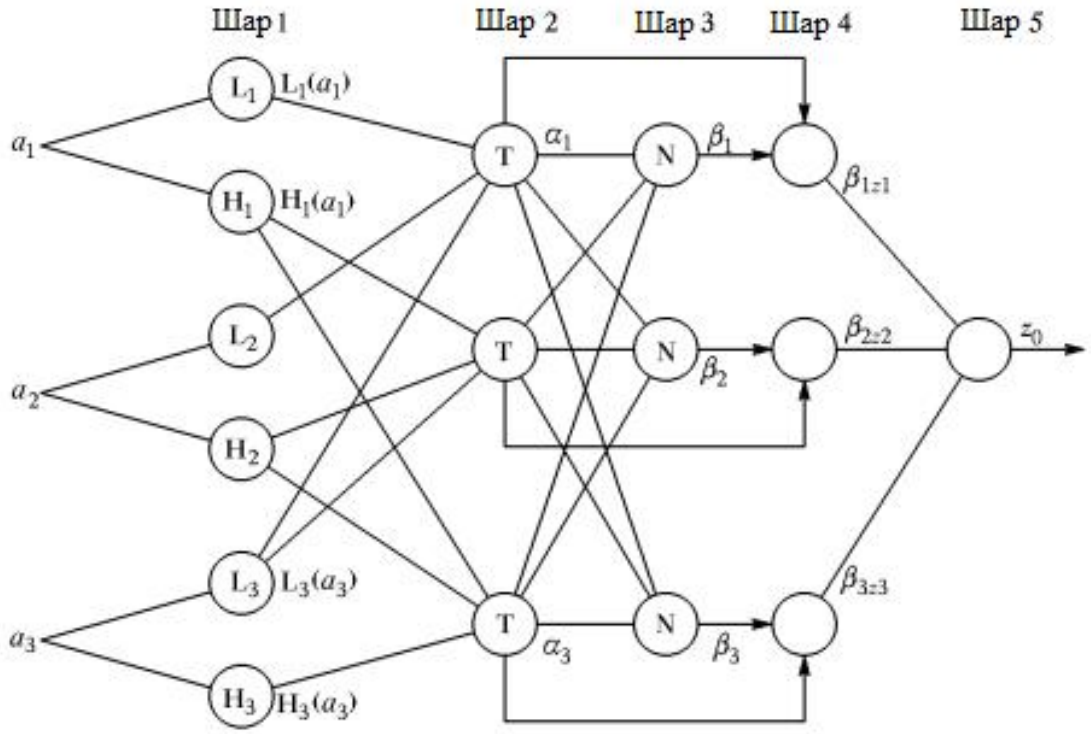


Рис. 1.3. Приклад складної структури нейронної сітки

На рис. 1.4 наведено фотографію структури нервових клітин живого організму. З неї видно, що у живих істот так само, як і в наведеному прикладі, існує багато паралельних і послідовних структур, пов'язаних поміж собою не тільки послідовно, але й зі зворотним зв’язком.

При створенні нейронної сітки складно з першого разу визначити, який її тип буде найбільш ефективно вирішувати поставлені задачі. Тому дослідники користуються рекомендаціями та знахідками своїх попередників, які вже вирішували подібні задачі і знайшли для них найбільш ефективні структури поєднання нейронів та види функцій активації.

Деяку складність викликає створення формули, яку реалізує отримана нейронна сітка. Пояснимо це на прикладі рис. 1.3.

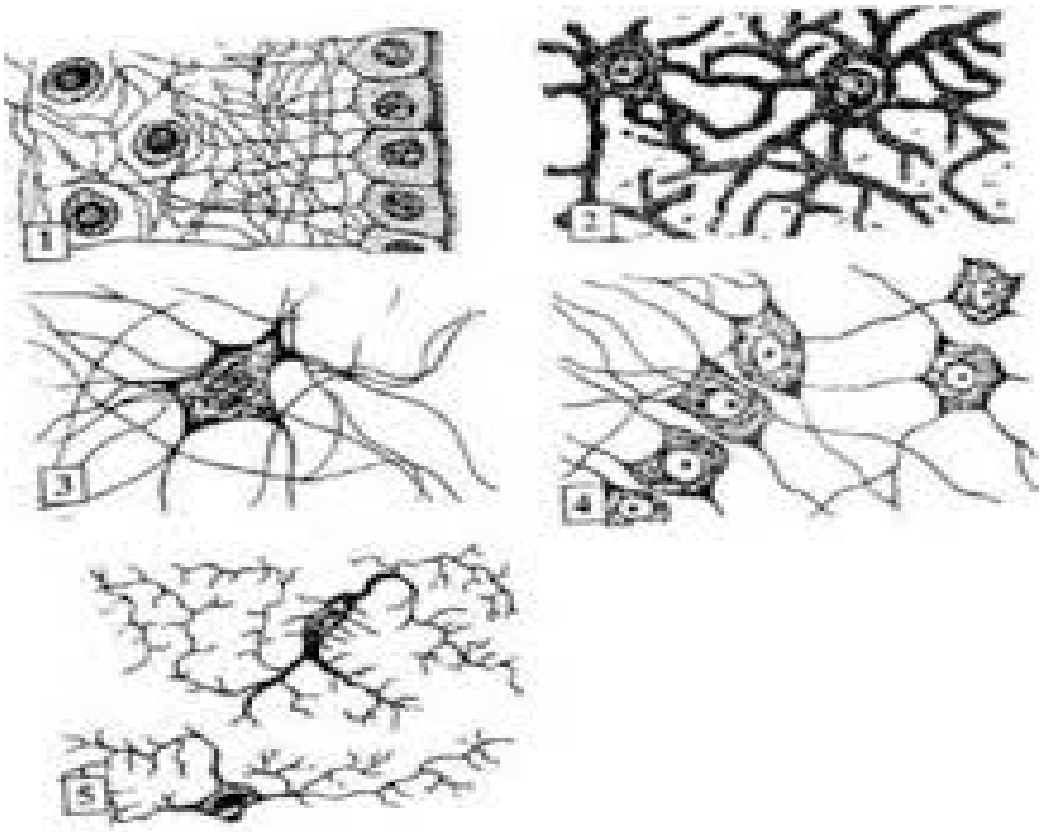


Рис. 1.4. Фотографії нервових клітин живого організму (нейрони) та зв’язків поміж ними (синапси)

Починати цю процедуру потрібно з останнього шару, рухаючись у зворотному напрямку. Потім усі отримані формули можна поєднати в одну підстановкою.

За визначенням, усі функції активації мають вигляд F. А вагові коефіцієнти суматорів позначимо як W_{ijk} для i-го нейрона для j-го вхідного сигналу та k-го шару.

Тоді маємо,

П’ятий шар: $Z_0 = F(\beta_{1z1}\, W_{115} + \beta_{2z2}\, W_{225} + \beta_{3z3}\, W_{335})$;

Четвертий шар: $\beta_{1z1} = F(\beta_1\, W_{114} + a_1\, W_{124})$;

$\beta_{2z2} = F(\beta_2\, W_{214} + a_2\, W_{224})$;

$\beta_{3z3} = F(\beta_2\, W_{314} + a_3\, W_{324})$;

Третій шар: $\beta_1 = F(a_1\, W_{113} + a_2\, W_{123} + a_3\, W_{133})$;

$B_2 = F(a_1\, W_{213} + a_2\, W_{223} + a_3\, W_{233})$;

$\beta_1 = F(a_1\, W_{313} + a_2\, W_{323} + a_3\, W_{333})$;

Другий шар: $a_1 = F(L_1(\mathrm{a}_1)\, W_{112} + L_2(\mathrm{a}_2)\, W_{122} + L_3(\mathrm{a}_3)\, W_{132})$;

$a_2 = F(H_1(\mathrm{a}_1)\, W_{212} + H_2(\mathrm{a}_2)\, W_{222} + L_3(\mathrm{a}_3)\, W_{232})$;

$a_3 = F(H_1(\mathrm{a}_1)\, W_{312} + H_2(\mathrm{a}_2)\, W_{322} + H_3(\mathrm{a}_3)\, W_{332})$;

Перший шар: $L_1(a_1) = F(W_{111}\, a_1)$;

$H_1(a_1) = F(W_{112}\, a_1)$;

$L_2(a_2) = F(W_{113}\, a_2)$;

$H_2(a_2) = F(W_{114}\, a_2)$;

$L_3(a_3) = F(W_{115}\, a_3)$;

$H_3(a_3) = F(W_{116}\, a_3)$.

Підставляючи послідовно формули для першого шару у формули для другого, потім для третього і так далі,отримаємо єдину формулу, що описує створену нами нейронну сітку.

1.2. Навчання

Визначення вагових коефіцієнтів, що стоять у формулах називається навчанням, тому, існує задачник – набір прикладів із заданими відповідями. Ці приклади пред'являються системі по одному. З точки зору таблиці експериментальних даних це означає, що на вхід нейронної сітки подається значення вхідних факторів по одному рядочку з таблиці. Нейрони одержують по вхідних зв'язках сигнали – «умови прикладу», перетворять їх, кілька разів обмінюються перетвореними сигналами і, нарешті, видають відповідь – також набір сигналів. Відхилення від правильної відповіді, тобто відміна від вихідного фактора з таблиці експериментів, штрафується. Навчання складається в мінімізації штрафу як (неявної) функції зв'язків. Неявне навчання приводить до того, що структура зв'язків стає «незрозумілою» – не існує іншого способу її прочитати, крім як запустити функціонування сітки. Стає складно побудувати зрозумілу людині логічну конструкцію, що відтворює дії сітки.

Штучну нейронну мережу інколи називають перцептроном. Цей термін визначив Розенблат [28], який в своєму алгоритмі навчання перцептрону довів, що перцептрон може бути навчений всьому, що він може реалізовувати.

Для навчання мережі образ X подається на вхід і обчислюється вихід Y. Якщо Y правильний, то нічого не міняється. Однак якщо вихід неправильний, то ваги, приєднані до входів, що підсилюють помилковий результат, модифікуються, щоб зменшити похибку. Розенблат експериментував з розпізнаванням цифр електронною схемою, яка мала видавати сигнал 0 або 1 в залежності від невірного або правильного розпізнавання.

Узагальнення алгоритму навчання перцептрону, зване дельтою-правилом, переносить цей метод на безперервні входи і виходи. Для цього вводиться величини d, яка рівна різниці між необхідним або цільовим виходом T і реальним виходом Y

$$d = (T - Y). \qquad (1.5)$$

Випадок, коли $d = 0$, відповідає ситуації, коли вихід правильний і в мережі нічого не змінюється. У будь-якому з цих випадків перцептронний алгоритм навчання зберігається, якщо d множиться на величину кожного входу x_i і цей добуток додається до відповідної ваги. З метою узагальнення вводиться коефіцієнт "швидкості навчання" h), який множиться на dx_i, що дозволяє управляти середньою величиною зміни ваги.

У алгебраїчній формі запису

$$D_i = hdx_i, \tag{1.6}$$

$$w(n+1) = w(n) + D_i, \tag{1.7}$$

де D_i – корекція, пов'язана з i-м входом x_i; $w_i(n+1)$ – значення ваги i після корекції; $w_i(n)$ - значення ваги i до корекції.

Дельта-правило модифікує ваги відповідно до необхідного і дійсного значень виходу кожної полярності як для безперервних, так і для бінарних входів і виходів. Ці властивості відкрили множину нових застосувань.

1.2.1. Навчальний алгоритм зворотного поширення

Перед початком навчання всім вагам повинні бути надані невеликі початкові значення, вибрані випадковим чином. Це гарантує, що в мережі не станеться насичення великими значеннями ваги, і запобігає ряду інших патологічних випадків. Наприклад, якщо всім вагам надати однакові початкові значення, а для необхідного функціонування потрібні нерівні значення, то мережа не зможе навчитися.

Навчання мережі зворотного поширення вимагає виконання наступних операцій:

1. Вибрати чергову навчальну пару з навчальної множини; подати вхідний вектор на вхід мережі.
2. Обчислити вихід мережі.
3. Обчислити різницю між виходом мережі і необхідним виходом (цільовим вектором навчальної пари).
4. Скорегувати ваги мережі так, щоб мінімізувати похибку.
5. Повторити кроки з 1 по 4 для кожного вектора навчальної множини доти, поки похибка на всій множини не досягне прийнятного рівня.

Операції, що виконуються на кроках 1 і 2, схожі з тими, які виконуються при функціонуванні вже навченої мережі, тобто подається вхідний вектор і обчислюється поточний вихід. Обчислення виконуються пошарово. На рис. 3.3 спочатку обчислюються виходи нейронів прошарку j, потім вони

використовуються як входи прошарку k, обчислюються виходи нейронів прошарку k, які і утворюють вихідний вектор мережі.

На кроці 3 кожний з виходів мережі, які на рис. 3.3 позначені OUT, віднімається з відповідної компоненти цільового вектора, щоб отримати похибку. Ця похибка використовується на кроці 4 для корекції ваги мережі, причому знак і величина змін ваги визначаються алгоритмом навчання (див. нижче).

Після достатнього числа повторень цих чотирьох кроків різниця між дійсними виходами і цільовими виходами повинна зменшитись до прийнятної величини, при цьому кажуть, що мережа навчилася. Тепер мережа використовується для розпізнавання і ваги не змінюються.

На кроки 1 і 2 можна дивитися як на «прохід вперед», оскільки сигнал розповсюджується по мережі від входу до виходу. Кроки 3, 4 складають "зворотний прохід", що тут обчислюється сигнал похибки розповсюджується зворотно по мережі і використовується для налаштування ваг. Ці два проходи тепер будуть деталізовані і виражені в математичній формі.

Прохід вперед. Кроки 1 і 2 можуть бути виражені у векторній формі таким чином: подається вхідний вектор X і на виході продукується вектор Y. Векторна пари вхід – мета X і T береться з навчальної множини. Обчислення проводяться над вектором X, щоб отримати вихідний вектор Y.

Як ми бачили, обчислення в багатошарових мережах виконуються прошарок за прошарком, починаючи з найближчого до входу прошарку. Величина NET кожного нейрона першого прошарку обчислюється як зважена сума входів нейрона. Активаційна функція F «стискає» NET і дає величину OUT для кожного нейрона в цьому прошарку. Коли множина виходів прошарку отримана, вона є вхідною множиною для наступного прошарку. Процес повторюється прошарок за прошарком, поки не буде отримана заключна множина виходів мережі.

Цей процес може бути виражений в стислій формі за допомогою векторної нотації. Ваги між нейронами можуть розглядатися як матриця W. Наприклад, вага від нейрона 8 в прошарку 2 до нейрона 5 прошарку 3 позначається $w_{8,5}$. Тоді NET – вектор прошарку N може бути виражений не як сума добутків, а як добуток X і W. В векторному позначенні $N = XW$. Покомпонентне застосування функції F до NET-вектора N продукує вихідний вектор O. Таким чином, для даного прошарку обчислювальний процес описується наступним виразом

$$O = F(XW). \qquad (1.8)$$

Вихідний вектор одного прошарку є вхідним вектором для наступного, тому обчислення виходів останнього прошарку вимагає застосування рівняння (1.8) до кожного прошарку від входу мережі до її виходу.

Зворотний прохід. *Налаштування ваг вихідного прошарку.* Оскільки для кожного нейрона вихідного прошарку задане цільове значення, то налаштування ваг легко здійснюється з використанням модифікованого дельта-правила. Внутрішні прошарки називають «прихованими прошарками», для їх виходів немає цільових значень для порівняння. Тому навчання ускладнюється.

На рис. 1.5 показаний процес навчання для однієї ваги від нейрона *p* в прихованому прошарку *j* до нейрона *q* у вихідному прошарку *k*. Вихід нейрона прошарку *k*, віднімаючись з цільового значення (*Target*), дає сигнал похибки. Він множиться на похідну стискаючої функції [*OUT*(1 - *OUT*)], обчислену для цього нейрона в прошарку *k*, даючи, таким чином, величину *d*.

$$d = OUT(1 - OUT)(Target - OUT). \tag{1.9}$$

Потім *d* множиться на величину *OUT* нейрона *j*, з якого отримують поточні ваги. Це в свою чергу множиться на коефіцієнт швидкості навчання *h* (переважно від 0,01 до 1,0), і результат додається до ваги. Така ж процедура виконується для кожної ваги від нейрона прихованого прошарку до нейрона у вихідному прошарку.

Наступні рівняння ілюструють це обчислення:

$$Dw_{pq,k} = h\, d_{q,k}\, OUT, \tag{1.10}$$

$$w_{pq,k}(n+1) = w_{pq,k}(n) + Dw_{pq,k}, \tag{1.11}$$

де $w_{pq,k}(n)$ – величина ваги від нейрона *p* в прихованому прошарку до нейрона *q* у вихідному прошарку на кроці *n* (до корекції); зазначимо, що індекс *k* відноситься до прошарку, в якому закінчується дана вага, тобто, згідно з прийнятому в цій книзі узгодженні, з якою він об'єднаний; $w_{pq,k}(n+1)$ – величина ваги на кроці $n+1$ (після корекції); $d_{q,k}$ – величина *d* для нейрона *q*, у вихідному прошарку *k*; $OUT_{p,j}$ – величина *OUT* для нейрона *p* в прихованому прошарку *j*.

Налаштування ваги прихованого прошарку. Розглянемо один нейрон в прихованому прошарку, що передує вихідному шару. При проході вперед цей нейрон передає свій вихідний сигнал нейронам у вихідному прошарку через ті, що з'єднують їх ваги. Під час навчання ці ваги функціонують в зворотному порядку, пропускаючи величину *d* від вихідного прошарку назад до прихованого прошарку. Кожна з цих ваг множиться на величину *d* нейрона, до якого він приєднаний у вихідному прошарку. Величину *d*, необхідна для нейрона прихованого прошарку, отримують в результаті підсумовування всіх добутків і множенням на похідну стискаючої функції (див. рис. 1.6):

$$\delta_{q,k} = OUT_{p,j}(1 - OUT_{p,j})\left[\sum_{q} \delta_{q,k} W_{pq,k}\right] \tag{1.12}$$

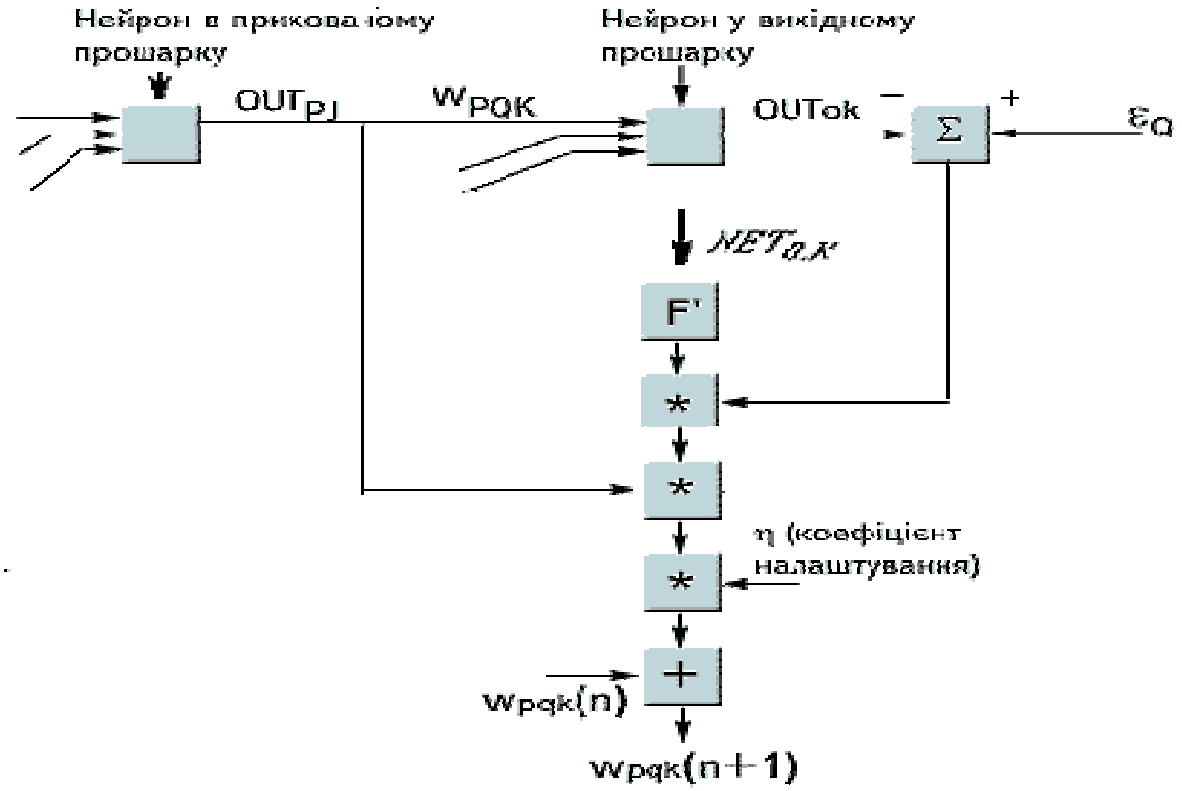


Рис. 1.5. Налаштування ваги у вихідному прошарку

Коли значення *d* отримано, ваги, підтримуючі перший прихований рівень, можуть бути скореговані за допомогою рівнянь (1.10) і (1.11), де індекси модифікуються у відповідності з прошарком.

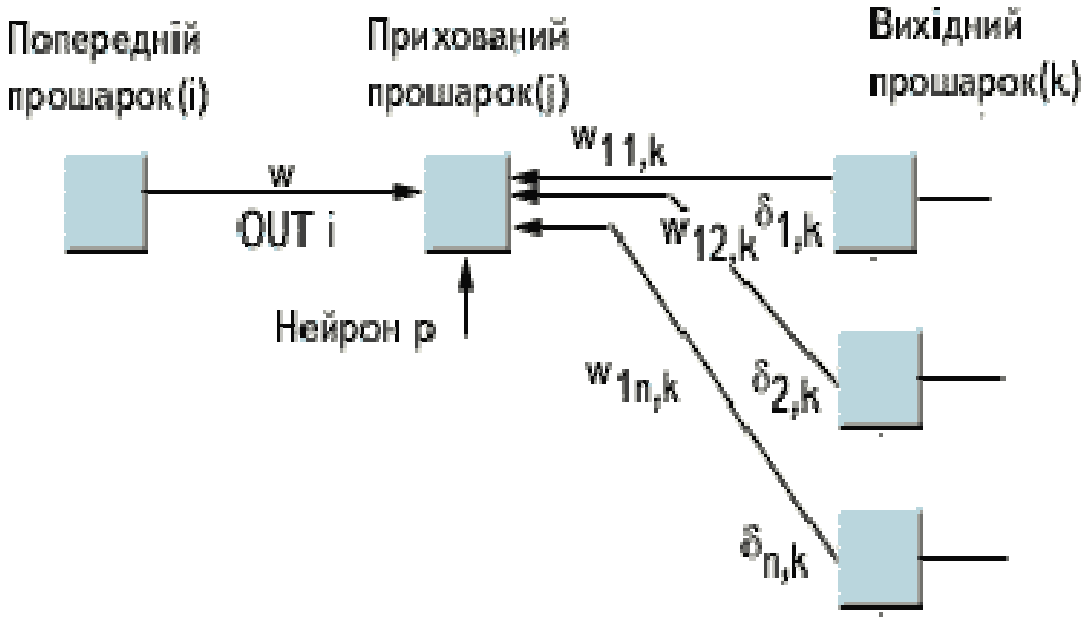


Рис. 1.6. Налаштування ваги в прихованому прошарку

Для кожного нейрона в даному прихованому прошарку повинне бути обчислено d і налаштовані всі ваги, асоційовані з цим прошарком. Цей процес повторюється прошарок за прошарком у напрямі до входу, поки всі ваги не будуть скоректовані.

За допомогою векторних позначень операція зворотного поширення похибки може бути записана значно компактніше. Визначимо множину величин d вихідного прошарку через D_k і множину ваг вихідного прошарку як масив W_k. Щоб отримати $D_{j,d}$ - вектор вихідного прошарку, досить наступних двох операцій:

1. Помножити про-вектор вихідного прошарку D_k на транспоновану матрицю ваги W'_k, що з'єднує прихований рівень з вихідним рівнем.

2. Помножити кожну компоненту отриманого добутку на похідну стискаючої функції відповідного нейрона в прихованому прошарку.

У символьному записі

$$D_j = D_k W'_k \$[0_j \$(I-0_j)], \tag{1.13}$$

де оператор \$ означає покомпонентне вироблення векторів, O_j - вихідний вектор прошарку j і I - вектор, всі компоненти якого рівні 1.

Додавання нейронного зміщення. У багатьох випадках бажано наділяти кожний нейрон навчальним зміщенням. Це дозволяє зсувати початок відліку логістичної функції, даючи ефект, аналогічний налаштуванню порога перцептроного нейрона, і приводить до прискорення процесу навчання. Ця можливість може бути легко введена в навчальний алгоритм за допомогою ваги, що додається до кожного нейрона, приєднаної до +1. Ця вага навчається так само, як і всі інші ваги, за винятком того, що сигнал, що подається на нього завжди рівний +1, а не виходу нейрона попереднього прошарку.

Імпульс. Існує метод прискорення навчання для алгоритму зворотного поширення, що збільшує також стійкість процесу. Цей метод, названий імпульсом, полягає в додаванні до корекції ваги члена, пропорційного величині попередньої зміни ваги. Як тільки відбувається корекція, вона «запам'ятовується» і служить для модифікації всіх подальших корекцій. Рівняння корекції модифікуються таким чином:

$$Dw_{pq,k}(n+1) = h\, d_{q,k}\, OUT_{p,j} + a\, Dw_{pq,k}(n), \tag{1.14}$$

$$w_{pq,k}(n+1) = w_{pq,k}(n) + Dw_{pq,k}(n+1), \tag{1.15}$$

де a – коефіцієнт імпульсу, звичайно встановлюється приблизно 0,9.

Використовуючи метод імпульсу, мережа прагне йти по дну вузьких ярів поверхні похибки (якщо такі є), а не рухатися від схилу до схилу. Цей метод,

мабуть, добре працює на деяких задачах, але дає слабкий або навіть негативний ефект на інших.

Існує також схожий метод, заснований на експонентному згладжуванні, який може мати перевагу в ряді застосувань.

$$Dw_{pq,k}(n+1)= (1- a)\, d_{q,k}\, OUT_{p,j} + a\, Dw_{pq,k}(n). \qquad (1.16)$$

Обчислюється зміна ваг

$$w_{pq,k}(n+1) = w_{pq,k}(n) + hDw_{pq,k}(n+1), \qquad (1.17)$$

де a – коефіцієнт згладжування, в діапазоні від 0,0 до 1,0. Якщо a дорівнює 1,0, то нова корекція ігнорується і повторюється попередня. У області між 0 та 1 корекція ваги згладжується величиною, пропорційною a. Як і раніше, h є коефіцієнтом швидкості навчання, що служить для управління середньою величиною зміни ваги.

Багатьма дослідниками були запропоновані поліпшення і узагальнення описаного вище основного алгоритму зворотного поширення. Література в цій області дуже широка, щоб її можна було тут охопити. Крім того, зараз ще дуже рано давати остаточні оцінки. Деякі з цих підходів можуть виявитися дійсно фундаментальними, інші ж згодом зникнуть. Деякі з найбільш багатообіцяючих розробок обговорюються в цьому розділі.

Наприклад, розроблено цікавий метод поліпшення характеристик навчання мереж зворотного поширення. Для цього треба прийти висновку, що загальноприйнятий від 0 до 1 динамічний діапазон входів і виходів прихованих нейронів неоптимальний. Оскільки величина корекції ваги $Dw_{pq,k}$ пропорційна вихідному рівню нейрона, що породжує $OUT_{p,\ j}$, то нульовий рівень веде до того, що вага не змінюється. При двійкових вхідних векторах половина входів в середньому буде рівна нулю, і ваги, з якими вони пов'язані, не будуть навчатися! Рішення полягає в приведенні входів до значень $\pm S$ і додаванні зміщення до стискаючої функції, щоб вона також приймала значення $\pm S$. Нова стискаюча функція виглядає таким чином:

$$\mathrm{OUT} = -\frac{1}{2} + \frac{1}{1 + e^{-\mathrm{NET}}}. \qquad (1.18)$$

За допомогою таких простих засобів час збіжності скорочується в середньому від 30 до 50%. Це є одним з прикладів практичної модифікації, істотно поліпшуючої характеристику алгоритму.

1.2.2. Мережі зустрічного поширення

Можливості мережі зустрічного поширення перевершують можливості одношарових мереж. Час же навчання в порівнянні із зворотним поширенням може бути зменшене в сто разів. Зустрічне поширення не настільки популярне, як зворотне поширення, але воно може давати рішення в тих застосуваннях, де довга навчальна процедура неможлива.

Мережа зустрічного поширення функціонує подібно столу довідок, здібному до узагальнення. У процесі навчання вхідні вектори асоціюються з відповідними вихідними векторами. Ці вектори можуть бути двійковими, що складаються з нулів і одиниць, або неперервними. Коли мережа навчена, пред'явлення вхідного вектора приводить до необхідного вихідного вектора. Узагальнююча здатність мережі дозволяє отримувати правильний вихід навіть при пред'явленні вхідного вектора, який є неповним або злегка невірним. Це дозволяє використати дану мережу для розпізнавання образів, відновлення образів і підсилення сигналів.

Структура мережі. На рис. 1.7 показана спрощена версія прямої дії мережі зустрічного поширення. На ньому ілюструються функціональні властивості цієї парадигми. Повна двоскерована мережа заснована на тих же принципах, вона обговорюється в цьому розділі пізніше.

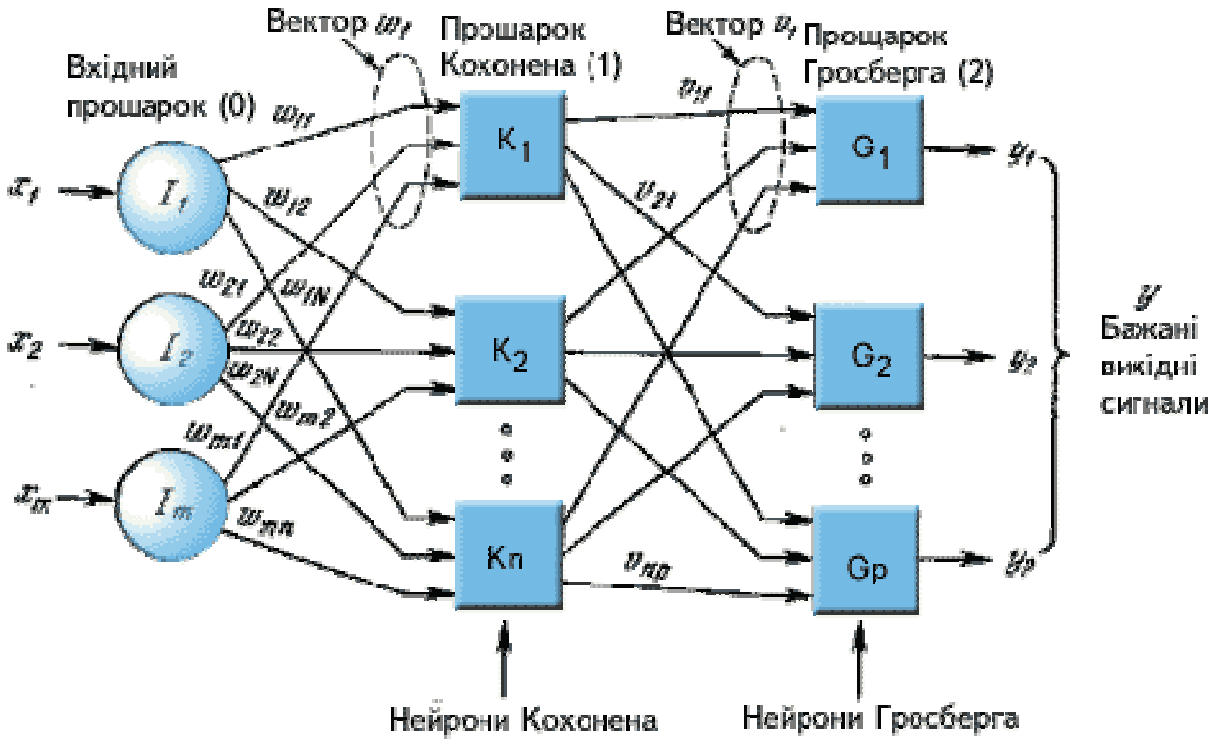


Рис. 1.7. Мережа із зустрічним розпізнаванням без зворотних зв'язків

Нейрони прошарку 0 (показані сферами) служать лише точками розгалуження і не виконують обчислень. Кожний нейрон прошарку 0 сполучений з кожним нейроном прошарку 1 (званого прошарком Кохонена)

окремою вагою w_{mn}. Ці ваги в загальному розглядаються як матриця ваги W. Аналогічно, кожний нейрон в прошарку Кохонена (прошарку 1) сполучений з кожним нейроном в прошарку Гросберга (прошарку 2) вагою v_{np}. Ці ваги утворюють матрицю ваги V. Все це досить нагадує інші мережі, що зустрічалися в попередніх розділах, відмінність, однак, складається в операціях, що виконуються нейронами Кохонена і Гросберга.

Як і в інших мережах, зустрічне поширення функціонує в двох режимах: в нормальному режимі, при якому приймається вхідний вектор X і видається вихідний вектор Y, і в режимі навчання, при якому подається вхідний вектор і ваги корегуються, щоб дати необхідний вихідний вектор.

Прошарок Кохонена. У своїй найпростішій формі прошарок Кохонена функціонує за принципом «переможець забирає все», тобто для даного вхідного вектора один і тільки один нейрон Кохонена видає на виході логічну одиницю, всі інші видають нуль. Нейрони Кохонена можна сприймати як набір електричних лампочок, так що для будь-якого вхідного вектора загоряється одна з них.

Асоційована з кожним нейроном Кохонена множина ваг з'єднує його з кожним входом. Подібно до нейронів більшості мереж вихід NET кожного нейрона Кохонена є просто сумою зважених входів. Це може бути виражене таким чином:

$$NET_j = w_{1j}x_1 + w_{2j}x_2 + ... + w_{mj}x_m, \tag{1.19}$$

де NET_j - це вихід NET нейрона Кохонена j,

$$NET_j = \sum_i x_i w_{ij}, \tag{1.20}$$

або у векторному записі

$$N = XW, \tag{1.21}$$

де N - вектор виходів NET прошарку Кохонена.

Нейрон Кохонена з максимальним значенням NET є "переможцем". Його вихід рівний одиниці, у інших він рівний нулю.

Прошарок Гросберга. Прошарок Гросберга функціонує в схожій манері. Його вихід NET є зваженою сумою виходів k_1, k_2,. .., k_n прошарку Кохонена, що створюють вектор K. Вектор з'єднуючих ваг, позначений через V, складається з ваг v_{11}, v_{21},. .., v_{np}. Тоді вихід NET кожного нейрона Гросберга є

$$NET_j = \sum_i k_i w_{ij}, \tag{1.22}$$

де NET_j - вихід j-*го* нейрона Гросберга, або у векторній формі

$$Y = KV, \tag{1.23}$$

де Y – вихідний вектор прошарку Гросберга, K – вихідний вектор прошарку Кохонена, V - матриця ваги прошарку Гросберга.

Якщо прошарок Кохонена функціонує таким чином, що лише в одного нейрона величина NET рівна одиниці, а у інших рівна нулю, то лише один елемент вектора K не рівний нулю, і обчислення дуже прості. Фактично кожний нейрон прошарку Гросберга лише видає величину ваги, яка зв'язує цей нейрон з єдиним ненульовим нейроном Кохонена.

Навчання прошарку Кохонена. Прошарок Кохонена класифікує вхідні вектори в групи схожих. Це досягається за допомогою такого налаштування ваг прошарку Кохонена, що близькі вхідні вектори активують один і той же нейрон даного прошарку. Задачею прошарку Гросберга є отримання необхідних виходів.

Навчання Кохонена є самонавчанням, що відбувається без вчителя. Тому важко (і не треба) передбачати, який саме нейрон Кохонена буде активуватися для заданого вхідного вектора. Необхідно лише гарантувати, щоб внаслідок навчання розділялися несхожі вхідні вектори.

Бажано (хоч і не обов'язково) нормалізувати вхідні вектори перед тим, як пред'являти їх мережі. Це виконується за допомогою ділення кожної компоненти вхідного вектора на довжину вектора. Ця довжина знаходиться витяганням квадратного кореня з суми квадратів компонент вектора. У алгебраїчному записі

$$x'_i = \frac{x_i}{\sqrt{x_1^2 + x_2^2 + \ldots + x_n^2}}. \tag{1.24}$$

Ї

Це перетворює вхідний вектор в одиничний вектор з тим же самим напрямом, тобто у вектор одиничної довжини в n-мірному просторі.

Рівняння (1.24) узагальнює добре відомий випадок двох вимірювань, коли довжина вектора рівна гіпотенузі прямокутного трикутника, утвореного його x та y компонентами, як це виходить з відомої теореми Піфагора. На рис. 4.2а такий двовимірний вектор V представлений в координатах x-y, причому координата x рівна чотирьом, а координата y трьом. Квадратний корінь з суми квадратів цих компонент рівний п'яти. Ділення кожної компоненти V на п'ять дає вектор V з компонентами 4/5 і 3/5, де V' вказує в тому ж напрямі, що і V, але має одиничну довжину.

На рис. 1.8-1.9 показано декілька одиничних векторів. Вони закінчуються в точках одиничного кола (коло одиничного радіуса), що має місце, коли у мережі лише два входи. У разі трьох входів вектори представлялися б стрілками, що закінчуються на поверхні одиничної сфери. Ці преставлення можуть бути перенесені на мережі, що мають довільне число входів, де кожний вхідний вектор є стрілкою, що закінчується на поверхні одиничної гиперсфери (корисною абстракцією, хоч і не допускає безпосередньої візуалізації).

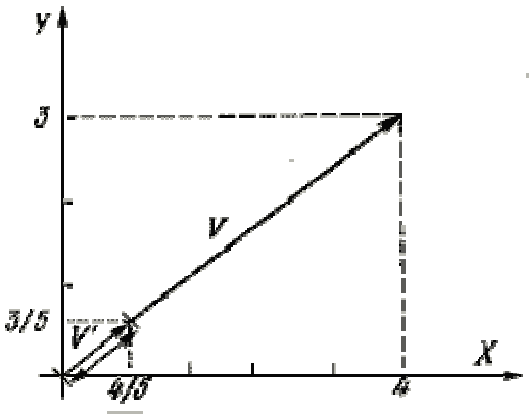


Рис. 1.8. Одиничний вхідний вектор

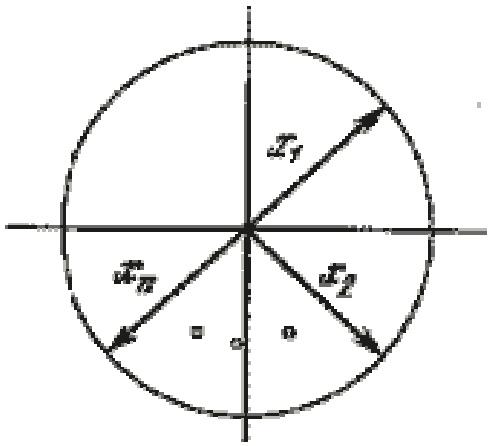


Рис. 1.9. Двовимірні одиничні вектори на одиничному колі

При навчанні прошарку Кохонена на вхід подається вхідний вектор і обчислюються його скалярні добутки з векторами ваги, пов'язаними з всіма нейронами Кохонена. Нейрон з максимальним значенням скалярного добутку оголошується "переможцем" і його ваги налаштовуються. Оскільки скалярний добуток, що використовується для обчислення величин *NET*, є мірою схожості між вхідним вектором і вектором ваги, то процес навчання складається у виборі нейрона Кохонена з ваговим вектором, найбільш близьким до вхідного вектора, і подальшому наближенні вагового вектора до вхідного. Знов зазначимо, що процес є самонавчанням, що виконується без вчителя. Мережа самоналаштовується таким чином, що даний нейрон Кохонена має максимальний вихід для даного вхідного вектора. Рівняння, те, що описує процес навчання має наступний вигляд:

$$w_н = w_с + a\,(x - w_с), \qquad (1.25)$$

де $w_н$ – нове значення ваги, що з'єднує вхідну компоненту x з нейроном-переможцем; w_c – попереднє значення цієї ваги; a – коефіцієнт швидкості навчання, який може змінюватися в процесі навчання.

Кожна вага, пов'язана з нейроном-переможцем Кохонена, змінюється пропорційно різниці між його величиною і величиною входу, до якого він приєднаний. Напрям зміни мінімізує різницю між вагою і його входом.

На рис. 1.10 цей процес показаний геометрично у двовимірному вигляді. Спочатку знаходиться вектор X - W_c, для цього проводиться відрізок з кінця W в кінець X. Потім цей вектор скорочується множенням його на скалярну величину a , меншу одиниці, внаслідок чого виходить вектор зміни d. Остаточно новий ваговий вектор $W_н$ є відрізком, направленим з початку координат в кінець вектора d. Звідси можна бачити, що ефект навчання складається у обертанні вагового вектора в напрямі вхідного вектора без істотної зміни його довжини.

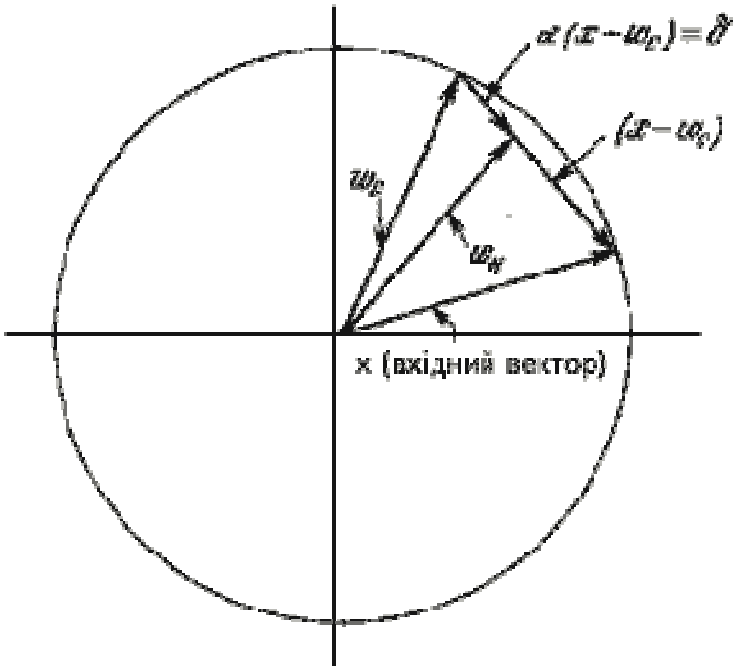


Рис. 1.10. Обертання вагового вектора в процесі навчання
($W_н$ – вектор нових вагових коефіцієнтів,
$W_с$ – вектор старих вагових коефіцієнтів)

Змінна k є коефіцієнтом швидкості навчання, який спочатку звичайно рівний ~ 0,7 і може поступово зменшуватись в процесі навчання. Це дозволяє робити великі початкові кроки для швидкого грубого навчання і менші кроки при підході до остаточної величини.

Якби з кожним нейроном Кохонена асоціювався один вхідний вектор, то прошарок Кохонена міг би бути навчений за допомогою одного обчислення вагою. Ваги нейрона-переможця прирівнювалися б до компонентів повчального вектора (a = 1). Як правило, навчальна множина включає багато схожих між собою вхідних векторів, і мережа повинна бути навчена активувати один нейрон Кохонена для кожного з них. У цьому разі ваги цього нейрона повинні виходити усередненням вхідних векторів, які повинні його активувати. Поступове зменшення величини a зменшує вплив кожного навчального кроку, так що остаточне значення буде середньою величиною від вхідних векторів, на яких відбувається навчання. Таким чином, ваги, асоційовані з нейроном, приймуть значення поблизу «центру» вхідних векторів, для яких даний нейрон є «переможцем».

Вибір початкових значень вагових векторів. Всім вагам мережі перед початком навчання потрібно надати початкові значення. Загальноприйнятою практикою при роботі з нейронними мережами є наданням вагам невеликих випадкових значень. При навчанні прошарку Кохонена випадково вибрані вагові вектори потрібно нормалізувати. Остаточні значення вагових векторів після навчання співпадають з нормалізованими вхідними векторами. Тому

нормалізація перед початком навчання наближає вагові вектори до їх остаточних значень, скорочуючи, таким чином, навчальний процес.

Рандомізація ваг прошарку Кохонена може породити серйозні проблеми при навчанні, оскільки в результаті її вагові вектори розподіляються рівномірно по поверхні гіперсфери. Через те, що вхідні вектори, як правило, розподілені нерівномірно і мають тенденцію групуватися на відносно малій частині поверхні гіперсфери, більшість вагових векторів будуть так віддалені від будь-якого вхідного вектора, що вони ніколи не будуть давати найкращої відповідності. Ці нейрони Кохонена будуть завжди мати нульовий вихід і виявляться некорисними. Більш того залишених ваг, що дають найкращу відповідність, може виявитися дуже мало, щоб розділити вхідні вектори на класи, які розташовані близько один до одного на поверхні гіперсфери.

Допустимо, що є декілька множин вхідних векторів, всі множини схожі, але повинні бути розділені на різні класи. Мережа повинна бути навчена активувати окремий нейрон Кохонена для кожного класу. Якщо початкова щільність вагових векторів в околі навчальних векторів дуже мала, то може виявитися неможливим розділити схожі класи через те, що не буде достатньої кількості вагових векторів в потрібному околі, щоб приписати по одному з них кожному класу вхідних векторів.

Навпаки, якщо декілька вхідних векторів отримані незначними змінами з одного і того ж зразка і повинні бути об'єднані в один клас, то вони повинні включати один і той же нейрон Кохонена. Якщо ж щільність вагових векторів дуже висока поблизу групи злегка різних вхідних векторів, то кожний вхідний вектор може активувати окремий нейрон Кохонена. Це не є катастрофою, оскільки прошарок Гросберга може відобразити різні нейрони Кохонена в один і той же вихід, але це марнотратна витрата нейронів Кохонена.

Найбільш бажане рішення полягає в тому, щоб розподіляти вагові вектори відповідно до щільності вхідних векторів, які повинні бути розділені, вміщуючи тим самим більше вагових векторів в околі великого числа вхідних векторів. На практиці це нездійсненне, однак існує декілька методів наближеного досягнення тих же цілей.

Одне з рішень, відоме під назвою *методу опуклої комбінації* (convex combination method), полягає в тому, що все ваги прирівнюються одній і тій же величині

$$w_i = \frac{1}{\sqrt{n}},$$

де n - число входів і, отже, число компонент кожного вагового вектора. Завдяки цьому всі вагові вектори співпадають і мають одиничну довжину. Кожній компоненті входу X надається значення

$$x_i = \alpha x_i + \frac{1 - \alpha}{\sqrt{n}},$$

де n - число входів. На початку a дуже мале, внаслідок чого всі вхідні вектори мають довжину, близьку до $1/\sqrt{n}$, і майже співпадають з векторами ваг. У процесі навчання мережі a поступово зростає, наближаючись до одиниці. Це дозволяє розділяти вхідні вектори і остаточно приписує їм їх істинні значення. Вагові вектори відстежують один або невелику групу вхідних векторів і в кінці навчання дають необхідну картину виходів. Метод опуклої комбінації добре працює, але сповільнює процес навчання, оскільки вагові вектори підстроюються до мети, що змінюється. Інший підхід складається в додаванні шуму до вхідних векторів. Тим самим вони зазнають випадкових змін, схоплюючи зрештою ваговий вектор. Цей метод також працездатний, але ще більш повільний, ніж метод опуклої комбінації.

Третій метод починає з випадкових ваг, але на початковій стадії навчального процесу налаштовує всі ваги, а не тільки пов'язані з нейроном-переможцем Кохонена. Тим самим вагові вектори зміщуються ближче до області вхідних векторів. У процесі навчання корекція ваги починає проводитися лише для найближчих до переможця нейронів Кохонена. Цей радіус корекції поступово зменшується, так що зрештою корегуються тільки ваги, пов'язані з нейроном-переможцем Кохонена.

Ще один метод наділяє кожний нейрон Кохонена «почуттям справедливості». Якщо він стає переможцем частіше за свою законну частку часу (приблизно $1/k$, де k – число нейронів Кохонена), він тимчасово збільшує свій поріг, що зменшує його шанси на виграш, даючи тим самим можливість навчатися і іншим нейронам.

У багатьох застосуваннях точність результату істотно залежить від розподілу ваг. На жаль, ефективність різних рішень вичерпним чином не оцінена і залишається проблемою.

Режим інтерполяції. Досі ми обговорювали алгоритм навчання, в якому для кожного вхідного вектора активувався лише один нейрон Кохонена. Це називається *методом акредитації*. Його точність обмежена, оскільки вихід повністю є функцією лише одного нейрона Кохонена.

У методі інтерполяції ціла група нейронів Кохонена, що мають найбільші виходи, може передавати свої вихідні сигнали в прошарок Гросберга. Число нейронів в такій групі повинно вибиратися в залежності від задачі, і точних даних відносно оптимального розміру групи немає. Як тільки група визначена, її множина виходів *NET* розглядається як вектор, довжина якого нормалізується на одиницю розподілом кожного значення *NET* на корінь квадратний з суми квадратів значень *NET* в групі. Всі нейрони поза групою мають нульові виходи.

Метод інтерполяції здатний встановлювати більш складну відповідність і може давати більш точні результати. Як і раніше, однак, немає точних даних, що дозволяють порівняти режими інтерполяції і акредитації.

Навчання прошарку Гросберга. Прошарок Гросберга навчається відносно просто. Вхідний вектор, що є виходом прошарку Кохонена, подається на прошарок нейронів Гросберга, і виходи прошарку Гросберга обчислюються, як

при нормальному функціонуванні. Далі, кожна вага коректується лише в тому випадку, якщо він з'єднаний з нейроном Кохонена, що має ненульовий вихід. Величина корекції ваги пропорційна різниці між вагою і необхідним виходом нейрона Гросберга, з яким він з'єднаний. У символьному записі

$$v_{ijн} = v_{ijc} + b\,(y_j - v_{ijc})k_i, \qquad (1.26)$$

де k_i - вихід *i-го* нейрона Кохонена (тільки для одного нейрона Кохонена він відмінний від нуля); y_j - j-*а*я компонента вектора бажаних виходів.

Спочатку b береться рівним ~0,1 і потім поступово зменшується в процесі навчання.

Звідси видно, що ваги прошарку Гросберга будуть сходитися до середніх величин від бажаних виходів, тоді як ваги прошарку Кохонена навчаються на середніх значеннях входів. Навчання прошарку Гросберга - це навчання з вчителем, алгоритм має в своєму розпорядженні бажаний вихід, по якому він навчається. Самоорганізований прошарок Кохонена дає виходи в недетермінованих позиціях. Вони відображаються в бажані виходи прошарком Гросберга.

Мережа зустрічного поширення повністю. На рис. 1.11 показана мережа зустрічного поширення повністю. У режимі нормального функціонування пред'являються вхідні вектори X і Y, і навчена мережа дає на виході вектори X' та Y', що є апроксимаціями відповідно для X та Y. Вектори X та Y вважаються тут нормалізованими одиничними векторами, отже, вектори, що продукуються на виході також будуть мати тенденцію бути нормалізованими.

У процесі навчання вектори X та Y подаються одночасно і як вхідні вектори мережі, і як бажані вихідні сигнали. Вектор X використовується для навчання виходів X', а вектор Y – для навчання виходів Y прошарку Гросберга. Мережа зустрічного поширення повністю навчається з використанням того ж самого методу, який описувався для мережі прямої дії. Нейрони Кохонена приймають вхідні сигнали як від векторів X, так і від векторів Y. Але, це не відрізняється від ситуації, коли є один великий вектор, складений з векторів X і Y, і не впливає на алгоритм навчання.

Як результат виходить одиничне відображення, при якому пред'явлення пари вхідних векторів породжує їх копії на виході. Це не представляється особливо цікавим, якщо не помітити, що пред'явлення тільки вектора X (з вектором Y, рівним нулю) виробляє як виходи X', так і виходи Y'. Якщо F - функція, що відображає X в Y', то мережа апроксимує її. Також, якщо F зворотна, то пред'явлення тільки вектора Y (прирівнюючи X нулю) продукує X'. Унікальна здатність виробляти функцію і зворотну до неї робить мережу зустрічного поширення корисною в ряді застосувань.

Рис. 1.11 на відміну від первинної конфігурації не демонструє зворотної дії в мережі, по якій вона отримала свою назву. Така форма вибрана тому, що вона також ілюструє мережу без зворотних зв'язків і дозволяє узагальнити поняття, розвинені в попередніх розділах.

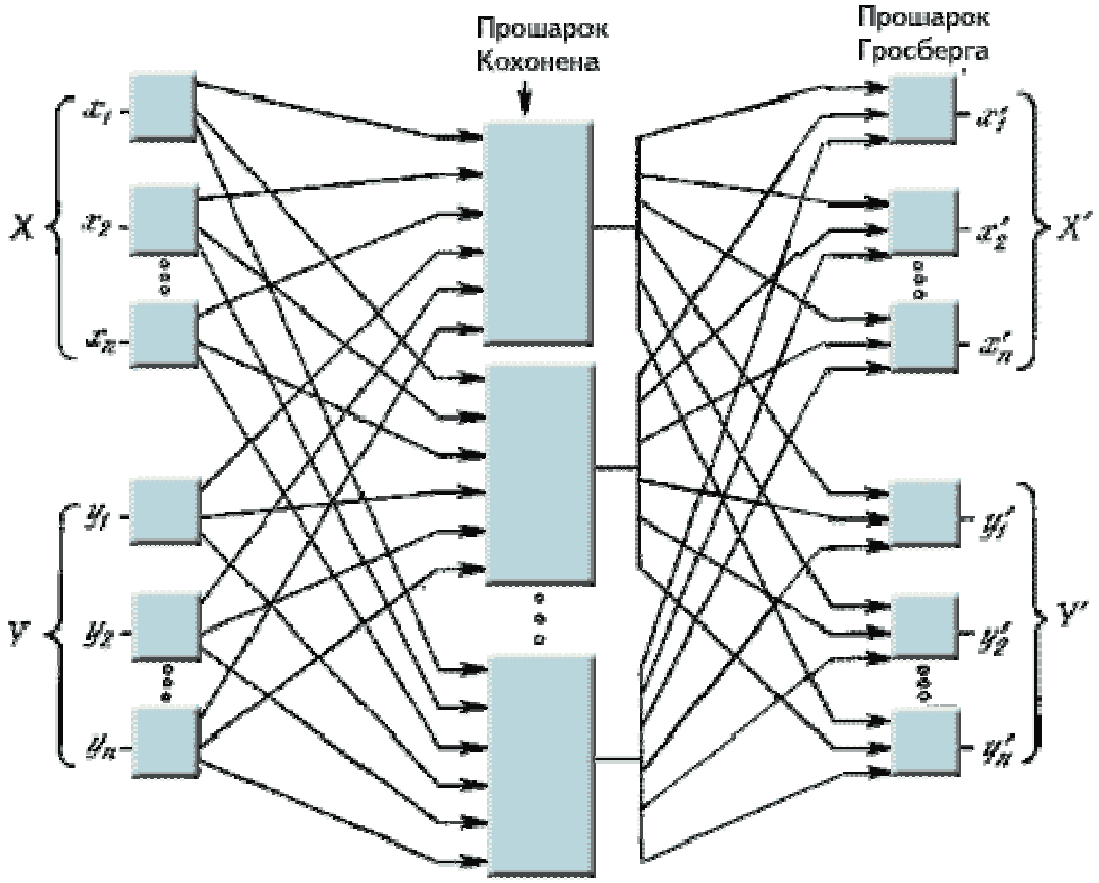


Рис. 1.11. Повна мережа зустрічного поширення

1.2.3. Приклад розрахунку вагових коефіцієнтів

Побудуємо нейронну сітку для двох варіантів економічних процесів. Перший стосується розпізнавання безперервних економічних процесів, а другий – дискретних.

Почнемо зі створення апроксимуючої залежності для даних про кількість замовлень підприємства зв'язку по годинах робочого дня.

В якості моделі було обрано одношаровий перцептрон з логістичною функцією активації та з одним нейроном на три входи. На кожен вхід подавалося значення трьох послідовних значень кількості викликів від початку доби. Кожне значення вхідного параметру x бралося з вагою w та з певним значенням порогової чутливості θ. Таким чином, математична модель такого перцептрона мала наступний загальний вигляд

$$OUT = \frac{1}{1 + \ell^{-(w_1 x_i + w_2 x_{i+1} + w_3 x_{i+2})}} . \qquad (1.27)$$

На кожному наступному кроці, підставлялося нове значення x, як i+2–й елемент перцептрону, а i–е значення x відкидалося. Значення OUT

порівнювалося зі значенням y з розрахунком погрішності прогнозування вигляду $x` = \frac{(x - m_x)}{\sigma_x}$. На кожному кроці розрахунку провадилося корегування ваги та порогової чутливості за правилом

$$\left.\begin{aligned} \Delta w_{ij} &= \varepsilon\left(d_j^s - y_j^s\right)x_{ij} \\ \Delta\theta_j &= -\varepsilon\left(d_j^s - y_j^s\right) \end{aligned}\right\}. \qquad (1.28)$$

де $d = y$, $y = OUT$, ε – число, яке характеризує швидкість навчання. Було встановлено наступне правило зменшення ε на кожному кроці розрахунку $\varepsilon' = \varepsilon/1.5668$, де ε' – нове значення швидкості навчання.

Перед початком навчання перцептрона всі дані були нормовані. В результаті навчання перцептрона було отримано показане на малюнку співпадіння розрахованих і реальних значень *y*. При цьому, сама апроксимуюча формула для нормованих значень параметрів, має вигляд

$$OUT = \frac{1}{1 + \ell^{-(1{,}58x_i + 0{,}37x_{i+1} + 1{,}15x_{i+2} + 1{,}62)}}. \qquad (1.29)$$

На рис. 1.12 показано графіки навчальної вибірки та кривої, яка реалізується за (1.29), в якій бралися три попередніх значення кількості викликів. Можна бачити, що точність апроксимації поступово збільшується, оскільки експериментальна і розрахована криві поступово зближуються.

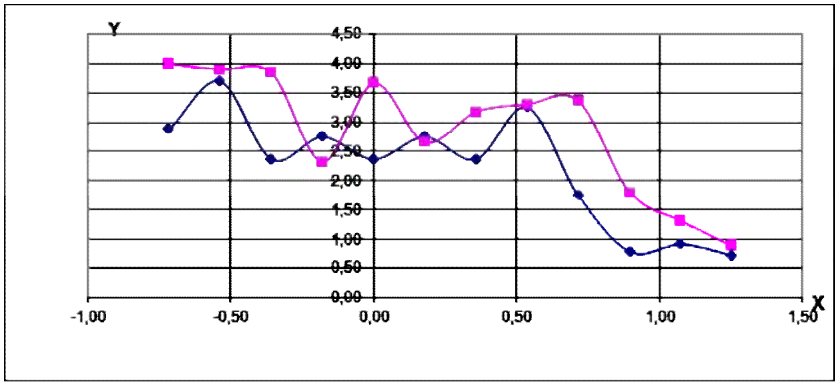


Рис. 1.12. Графік кількості викликів Y по годинам робочого дня X.
(♦ – експериментальна, ■ – розрахована крива)

Побудуємо тепер прогнозуючу модель зміни курсу австрійського шилінга відносно австралійського долара (дані взяти по відомостям НБУ за період 01.02.2000 по 31.10.2000). В цьому випадку нас цікавить передбачення самого факту збільшення чи зменшення крос-курсу.

Для вирішення цієї задачі було розроблено двошаровий перцептрон (тобто, модель системи, що складається з елементарний нейронів) (рис. 1.13), на вхід якого подавалося значення трьох попередніх значеннях крос-курсу австрійського шилінга і робився прогноз вихідного параметра, який являє собою функцію знаку числа

$$OUT = Sigfn(NET) = \begin{cases} 1, NET.0 \\ 0, NET = 0 \\ -1, NET < 0 \end{cases} \qquad (1.30)$$

В якості активуючих функцій першого шару були взяті логістичні. Навчання перцептрона виконувалося так. На кожному наступному кроці, підставлялися нові значення x_i як для дієздатних так для недієздатних вузлів. Причому, вихідними значення d_i бралися числа: -1 – для випадку перебільшення курсу австрійського шилінга відносно австралійського долара, +1 – для протилежного випадку дієздатність якого експертами оцінюється як критична, +1 – для випадку спів падіння курсів. Значення *OUT* порівнювалося зі значенням *di* з розрахунком погрішності прогнозування (7). На кожному кроці розрахунку провадилося корегування ваги та порогової чутливості за правилом (4.25). Було встановлено наступне правило зменшення ε на кожному кроці розрахунку t $\varepsilon'=\varepsilon/1.5668$, де ε' – нове значення швидкості навчання.

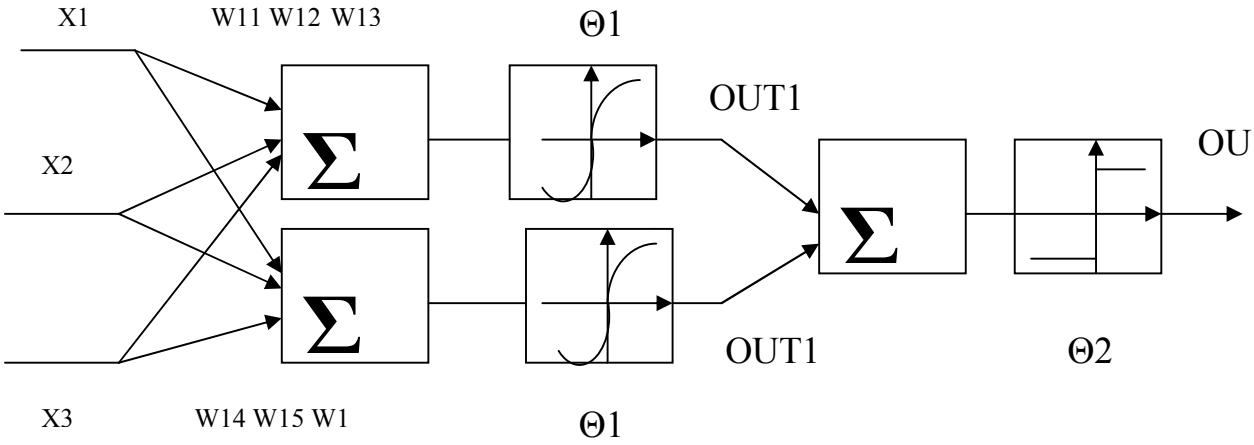


Рис. 1.13. Схема двошарового перцептрона
з трьома входами на кожному нейроні.

На рис. 1.14 представлено схему застосування для розрахунків такого перцептрона.

	X	W1	X*W1	SUM	TETA1	OUT1	W2	OUT1*W2	SUM	Teta2	OUT
X1	0,3	1	-0,3	-1,1	1	0,10545	2	0,21090802	0,21649	2,8	-1
X2	0,3	2	-0,61	-3,2	2	0,00558	1	0,00557715			
X3	0,1	3	-0,23							Real	1
		4	-1,21		Eps		3			Diff	2
1		5	-1,51		Sp		1				
		6	0,46								

Рис. 1.14. Фрагмент таблиці , яка реалізує перцептрон з рис. 1.13.

Стрілки на рис. 1.14 показують порядок передачі даних. Потім вхідні дані поновлювалися на один крок, тобто, використовувалися чергові значення вхідних параметрів для іншого вузла Знаходилася розбіжність поміж значеннями, які видає перцептрон та справжньою оцінкою стану вузла. в порівнянні зі своїм попереднім значенням з корегуванням ваг та порогових функцій. Цей алгоритм показано на рис. 1.15.

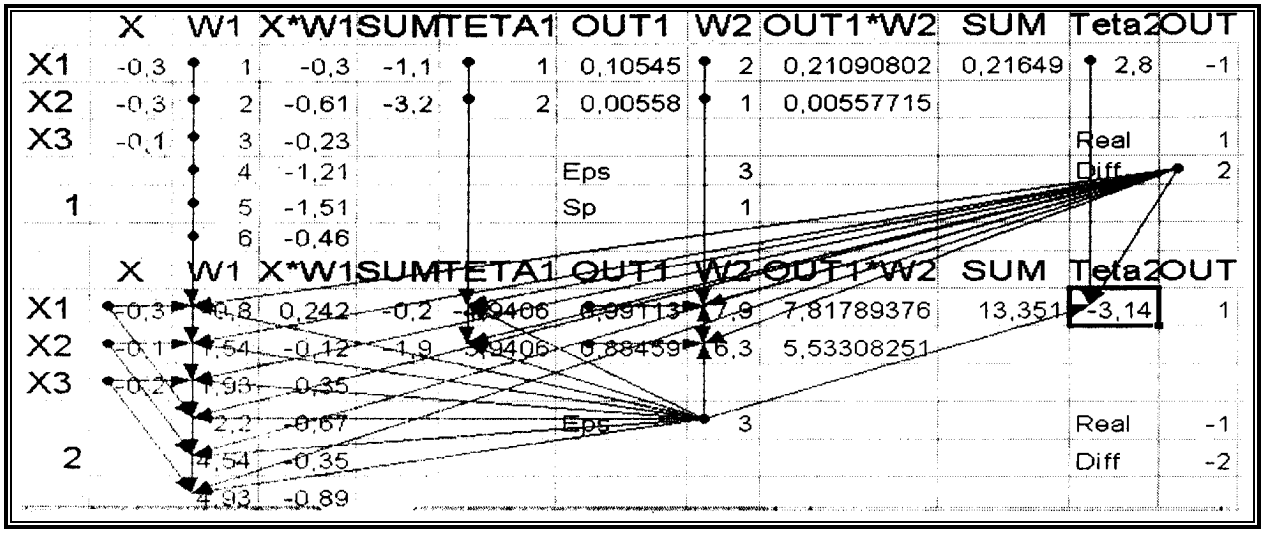


Рис. 1.15. Схема корегування ваг та порогів на наступному кроці, враховуючи результати з попереднього

В процесі навчання вже на 9-му кроці перцептрон давав більшість правильних відповідей на наступних значеннях курсу. Тому, процес навчання було припинено, а результуюча функція набула вигляду

$$OUT = Sign\left[\frac{10{,}37669298}{\left[1+\ell^{-(-1{,}443525248X_1+1{,}561436345X_2+1{,}582503396X_3+4{,}683681367)}\right]}+\frac{7{,}940045065}{\left[1+\ell^{-(1{,}556474752X_1+4{,}561436345X_2+4{,}582503396X_4+-3{,}683681367)}\right]}+2{,}883681367\right]$$

1.3. Застосування нейронних сіток для деяких видів розрахунків

1.3.1. Автоматична класифікація

Нехай існує деякий об'єкт, який характеризується кількома параметрами p_1 .. p_N . Нехай також є M класів об'єктів, C_1 .. C_M. Ми спостерігаємо об'єкт і можемо розрахувати чи виміряти його параметри. Вектор $\boldsymbol{p}$ характеризує об'єкт, що спостерігається:

$$\mathbf{p} = \begin{pmatrix} p_1 \\ \dots \\ p_N \end{pmatrix}$$

На підставі вектора $\boldsymbol{p}$ ми повинні вирішити, до якого класу віднести об'єкт, тобто вибрати C_i, до якого належить об'єкт, що характеризується набором параметрів $\boldsymbol{p}$. Рішення завдання можна представити у вигляді вектора :

$$\mathbf{c} = \begin{pmatrix} c_1 \\ \dots \\ c_M \end{pmatrix}$$

і виконуються умови: $0 \le c_m \le 1$ і $\sum_{m=1}^{M} c_m = 1$ (1)

Тут c_m – ймовірність, з якою об'єкт належить до класу C_m. Якщо розглядати c_m як імовірність, то повинні виконуватися умови (1). Приміром, c_m =0,9 і c_m = означає, що об'єкт з даними набором параметрів $\boldsymbol{p}$ з імовірністю 0,9 відноситься до класу C_1 і з імовірністю 0,1 – до класу C_2.

Якщо створити МСП з N входами і M виходами і навчити його давати на виході вектор **C** , коли на вхід подається $\boldsymbol{p}$, то ми вирішимо поставлену задачу.

Мережа будує відображення $P \rightarrow C$ в процесі навчання. Цілком витягти це відображення мережа не дозволяє, але можна отримати довільну кількість пар ($p \rightarrow c$), пов'язаних відображенням. Для довільного вектора $\boldsymbol{p}$ на вході ми можемо отримати ймовірності приналежності до класів на виході.

Чому на виході будуть отримані саме ймовірності c_m і чи будуть виконуватися умови (1)?

Якщо навчання пройшло успішно, то ми напевно одержимо на виході щось схоже на ймовірності. Це визначається алгоритмом навчання. Але найчастіше виявляється, що компоненти вихідного вектора можуть бути менше 0 або більше 1, а друга умова (1) виконується лише приблизно.

Неточність – наслідок аналогових нейронних мереж. Більшість результатів, що даються нейромережами, неточно. Крім того, при навчанні мережі зазначені умови, що накладаються на ймовірності, не вводяться в мережу безпосередньо, а неявно містяться в множині даних, на яких навчається мережа. Це – друга причина некоректності результату.

Такий спосіб формалізації – не єдиний , але один з вдалих. Можна навчити мережу і по іншому. Нехай у мережі тільки один вихід, і нехай його вихідний сигнал – номер класу m для вектора $\boldsymbol{p}$, пред'явленого на вході. Отже, мережа навчається залежності $m(\boldsymbol{p})$. Якщо навчання пройшло успішно, то коли на вхід мережі поданий вектор $\boldsymbol{p}$, що характеризує об'єкт, на виході буде отримано число m, і нами приймається рішення про приналежність $\boldsymbol{p}$ до класу C_m.

На перший погляд такий спосіб формалізації більш економічний: використовується всього один вихід. Але існує важливий недолік . Розглянемо приклад класифікації (рис. 1.).

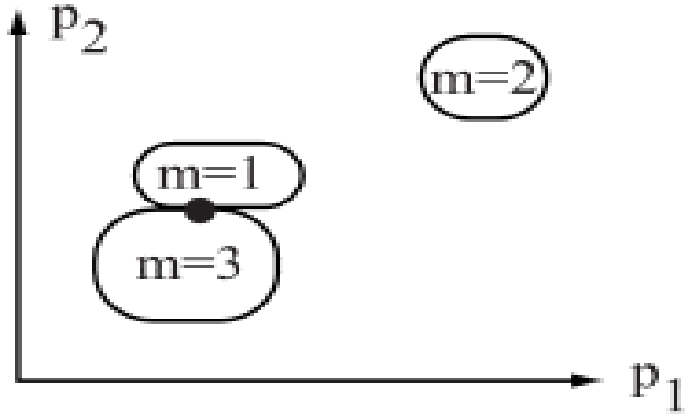


Рис 1.2. Приклад некоректної класифікації .

Нехай потрібно розділити об'єкти за двома ознаками p_1 і p_2, на три класи, $m = 1$, $m = 2$, $m = 3$. Якщо вхідний вектор $\boldsymbol{p}$ прийме значення, позначене жирною крапкою, то вихід мережі, при правильному навчанні, прийме значення $m = 2$, тобто об'єкт буде віднесений до класу 2, абсолютно невідповідному фактичному положенню вектора в просторі $(p_1; p_2)$.

Дане явище виникає, тому мережа схильна інтерполювати вхідні і вихідні дані. Якщо функції активації плавні, вагові коефіцієнти не надто великі, і кількість шарів не надто великий, то вихід мережі теж буде гладким і безперервним. Для близьких $\boldsymbol{p}$ будуть отримані близькі m на виході. Але при вирішенні задачі класифікації таке допущення буває невірним.

Звідси неправильне рішення. Щоб уникнути помилок, можна застосувати інші способи формалізації або впорядкувати номери класів m так , щоб близьким m відповідали близькі в просторі P класи.

1.3.2. Прогнозування одновимірної функції

Нехай задана функція W, θ, визначена на інтервалі часу$[0; t_0]$, де t_0 – поточне значення часу. Потрібно передбачити значення функції при $t > t_0$. Щоб застосувати багатошаровий перцептрон для прогнозування, час потрібно дискретизувати. Будемо вважати, що нам відомі значення функції в моменти часу

$$\begin{pmatrix} x_0 = f(t_0) \\ x_1 = f(t_0 - \delta_1) \\ x_2 = f(t_0 - \delta_1 - \delta_2) \\ \dots \\ x_n = f(t_0 - \delta_1 - \dots \delta_n) \end{pmatrix} = \mathbf{x}, \quad \delta_i > 0$$

Будемо передбачати значення функції в момент часу $(t_0 + \delta_0)$ для $\forall \delta_0 > 0$. δ_0 називається інтервалом прогнозування. Рішенням задачі будемо вважати значення $f(t_0 + \delta_0) = y$.

Побудуємо мережу, що має n входів і 1 вихід. В якості вхідного вектора візьмемо вектор x , а вихідного – один сигнал y.

Така мережа передбачає значення функції в одній точці y по (n +1) відомим значенням функції, заданим вектором x. Вибравши при навчанні мережі набір інтервалів δ, його не можна змінити після навчання. Мережа з даними параметрами W, θ, отриманими при навчанні, може прогнозувати тільки з одним набором δ_i.

Чи можна прогнозувати функцію у вигляді дискретного процесу в часі ? Як передбачити кілька значень функції у різних точках?

Для цього знайдений цікавий спосіб . Виберемо всі інтервали однаковими : $\delta_i = \delta$ = const. Побудуємо і навчимо мережу. Подамо на вхід вектор x зі значеннями функції у відомих точках. Розрахувавши вихід мережі, отримаємо прогнозоване значення функції в точці точці $f(t_0 + \delta_0) = y$. Тепер «зсунемо» компоненти вхідних і вихідних векторів наступним чином (знак рівності означає "привласнити значення"):

$$x_n = x_{n+1}$$
$$\dots$$
$$x_1 = x_0$$
$$x_0 = y.$$

Тепер вихідний вектор став однією з компонент вхідного. Знову розраховуємо вихід, і отримуємо значення функції в точці ($t_0 + 2\delta_0$). Повторивши ці операції, можна прогнозувати функцію в будь-якій кількості точок з дискретним кроком за часом, рівним δ.

1.3.3. Апроксимація багатовимірної функції

Розглянемо багатовимірну функцію $\boldsymbol{y} = f(\boldsymbol{x})$, де вектор $\boldsymbol{y}$ має N_O компонент, а вектор $\boldsymbol{x}$ – N_I компонент. Найпростіший спосіб формалізації – використовувати мережу з N_I входами і N_O виходами.

Компоненти вектора $\boldsymbol{x}$ подаються на вхід мережі, $\boldsymbol{y}$ – знімаються на виході. Мережа навчається на відомих значеннях функції f.

1.5. Індивідуальне завдання № 1. Створення апроксимаційної залежності методом зворотного поширення помилки

Для отримання числових значень треба скористатися електронними таблицями Excel у наступному порядку:

1. Створити стовпчик даних, розміром у 40 клітинок, які містять наступну формулу

=СЛУЧМЕЖДУ((номер вашої залікової книжки – 50)*(ваш номер за списком групи); (номер вашої залікової книжки)*(ваш номер за списком групи+30))/100

2. Відмітити цей стовпчик і занести в пам'ять комп'ютера, натиснувши кнопку „Копировать" або сполучення кнопок "CTRL"+"C".

3. Перенести курсор на стовпчик і через головне меню «Специальная вставка-Только значения» занести туди числові значення, які і будуть першим основним рядком даних для розрахунку за завданням.

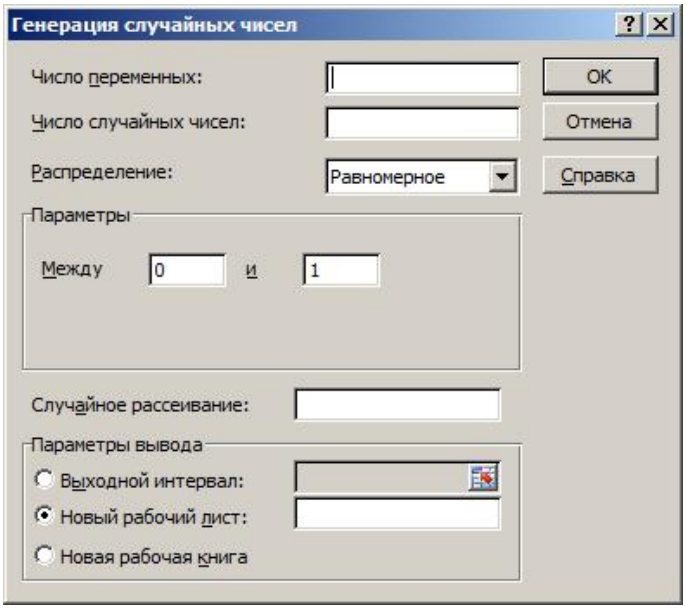
4. Перенести курсор на інший стовпчик і повторити пп. 2-3 для створення ще двох стовпчиків, потрібних для виконання завдання.

5. Будемо вважати перші два стовпчики вхідними факторами, а третій – вихідним.

6. За допомогою електронних таблиць Excel побудувати модель двошарового перцептрону, у якого в першому шарі знаходиться два нейрона, а у другому – один. Активаційна функція всіх нейронів – модифікована логістична, тобто

$$OUT = \frac{1}{1 + EXP(-NET)} - 0,5$$

7. Для первинних значень вагових коефіцієнтів скористайтеся функцією «Геннерация случайных чисел» пакета аналізу Excel. Виберіть рівномірний розподіл і вкажіть число змінних – 1, а число випадкових чисел – 6. Параметри встановити поміж останнім числом залікової книжки плюс 2 зі знаком мінус, та номером за списком групи плюс 10, зі знаком плюс.



Вхідний інтервал вкажіть за місцем розташування таблиці, наведеної нижче, в якій перші

чотири вагові коефіцієнти стосуються першго шару перцептрона, а два останні – другого.

W_{111}	W_{121}	W_{211}	W_{221}	W_{112}	W_{122}

8. Розрахунки проводьте паралельно з таблицею початкових даних. Тобто, для кожного чергового значення вихідно фактора, виконайте розрахунок помилки та величину скоректованих значень вагових коефіцієнтів.

9. Розрахунок припиняється тоді, коли величина похибки прогнозування вихідного сигналу буде менше 5%.

Запитання для самоперевірки

1. Які типи структур нейронних мереж ви знаєте?
2. У чому складність написання формули за побудованою нейронною мережею?
3. Поясніть термін «навчання».
4. Що таке вагові коефіцієнти?
5. Як знайти вагові коефіцієнти для проміжного шару мережі методом зворотного поширення?
6. Опишіть мережу Кохонена.
7. Чим мережі зустрічного поширення кращі за мережі зворотного поширення?
7. Які задачі можна вирішувати із застосуванням нейронних сіток?

За матеріалами розділу студенти опанували основні теоретичні положення щодо структури нейронних мереж, видів активаційних функцій, принципів навчання мережі, напрямків їх застосування.

2. ПРАКТИЧНЕ ЗАСТОСУВАННЯ НЕЙРОННИХ СІТОК

Вивчивши матеріали цього розділу, студенти опанують основні прийоми роботи з програмним пакетом Matlab 6

В цьому розділі зібрані інструкції по користуванню програмним комплексом Matlab 6, який дозволяє реалізовувати нейронні сітки різних конфігурацій, автоматично розраховує вагові коефіцієнти, дозволяє вибрати тип мережі, метод знайдення коефіцієнтів.

Студентам пропонується виконання деяких видів розрахунків, які наочно демонструють можливості цього математичного методу для моделювання, оптимізації, прогнозування.

2.1. Порядок застосування програмного пакета Matlab 6

2.2.1 Налаштування Matlab 6

Робота з системою Matlab починається з того, що ви запускаєте систему з іконки робочого столу системи Windows. У результаті на дисплеї відкривається робочий стіл системи Matlab (рис. 2.1).

Він містить елементи графічного інтерфейсу користувача, які призначені для роботи з файлами, змінними і додатками, пов'язаними з Matlab.

На рис. 2.1 ви бачите п’ять відкритих вікон: командне вікно (***Command Window***), в якому розташовано командний рядок, зліва – вікно поточного каталогу (***Current Directory***) та вікно (***Details),*** справа – вікно (панель) робочої області (***Workspace***) з кнопками розгортки/згортки і вікно передісторії викликів (***Command History***). Також показано вертикальний роздільник вікон, переміщення якого управляє розміщенням вікон у полі екрана. Відзначимо також інформаційне вікно поточного каталогу на інструментальній панелі робочого столу, а також кнопки виклику підказки, закриття та відділення командного вікна від робочого столу. Остання кнопка дозволяє перейти в режим роботи попередніх версій системи *Matlab*.

Описаний робочий стіл з'являється за замовчуванням при запуску системи *Matlab*. Ви можете змінити склад і розміщення робочих вікон, відкриваючи нові, закриваючи наявні, а також змінюючи їх розміри за допомогою вертикальних і горизонтальних роздільників. Крім того, можна відокремлювати вікна від робочого столу і поміщати їх на робочий стіл, використовуючи меню ***View*** інструментальної панелі.

Командне вікно (***Command Window)*** призначене для роботи із змінними, файлами і функціями системи Matlab.

На відміну від попередніх версій в середовищі системи *Matlab* всі оператори як у зоні перегляду командного вікна, так і в командному рядку можуть бути активізовані за допомогою правої кнопки миші. При цьому з'являється контекстне меню з наступними опціями: ***Evaluate Selection*** (Обчислити виділене), ***Open Selection*** (Відкрити виділене), ***Help on Selection*** (Довідка по виділеному), ***Copy*** (Копіювати), ***Paste*** (Вставити).

Для налаштування таких параметрів робочого столу, як тип і розмір шрифту, кольору символів командного рядка, слід скористатися опцією ***Preferences*** меню ***File*** (рис. 2.2).

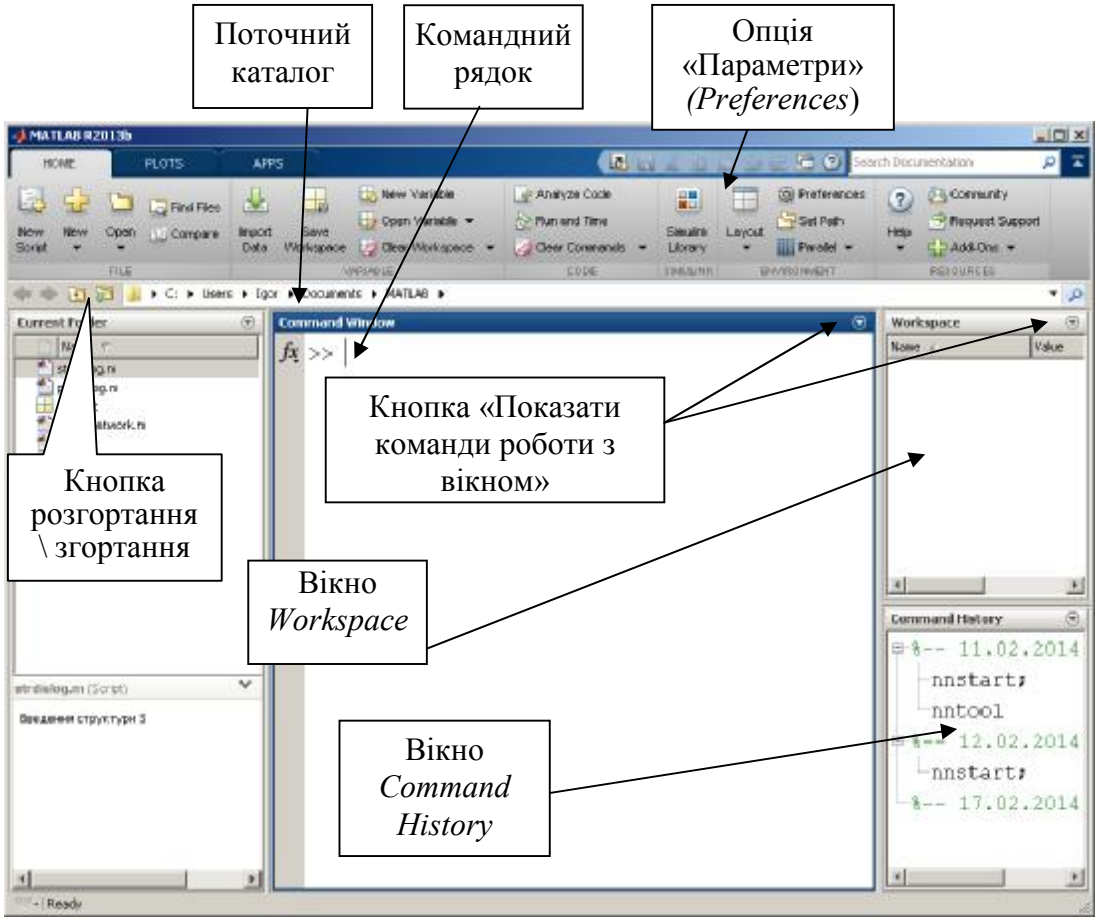


Рис.2.1. Робочий стіл системи Matlab.

Якщо натиснути праву кнопку мишки, з’явиться контекстне меню (рис. 2.3), яке дозволяє виконувати різні операції по роботі зі змінними, аналогічне операціям з літерами у текстовому редакторі: копіювати, видаляти, вставляти, вирізати, тощо. Аналогічно діють і гарячі клавіші, такі як CTRL+ C, CTRL+ V, CTRL+ I, CTRL+ X.

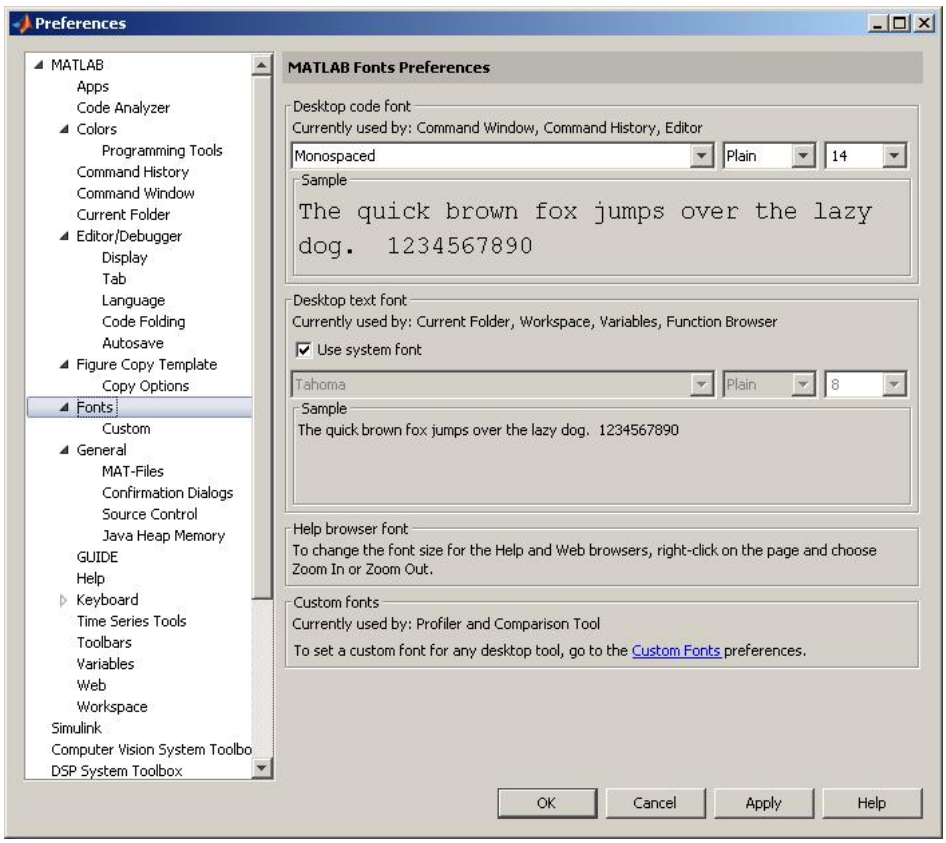


Рис. 2.2. Опція «Параметри» *(Preferences*)

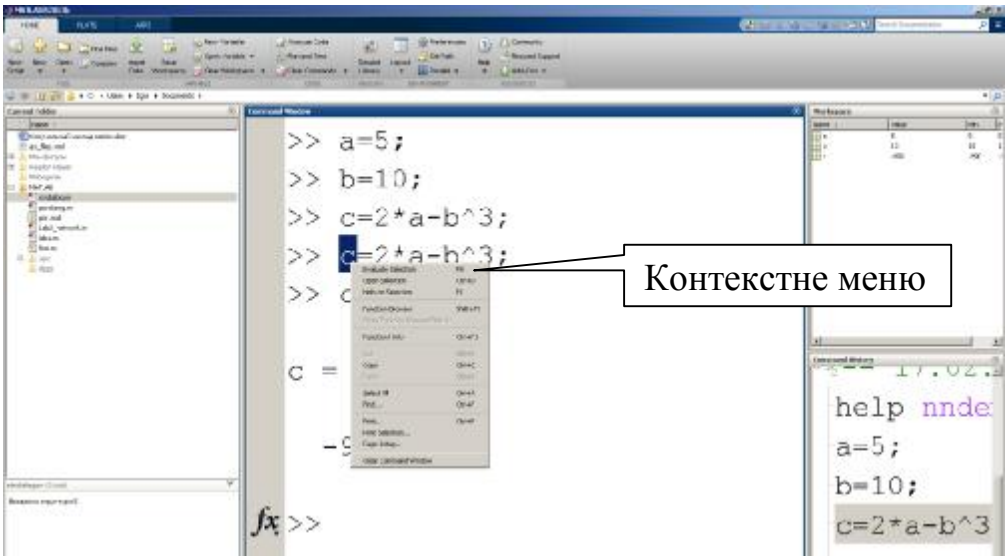


Рис.2.3. Контекстне меню для роботи зі змінними.

Вікно передісторії виклику операторів ***Command History*** показано на рис. 2.4. Воно містить рядки операторів, які вводилися в командному вікні з початку сеансу роботи. як виглядає вікно передісторії в Matlab (рис. 1.4). Ці рядки можна активізувати до виконання подвійним клацанням лівої кнопки миші, можна переносити з вікна в командний рядок, а можна використовувати праву кнопку миші. При цьому з'являється контекстне меню з наступними опціями: ***Copy*** (Копіювати), ***Evaluate Selection*** (Обчислити виділене), ***Create M-File*** (Створити М-файл), ***Delete Selection*** (Видалити виділене), ***Delete to***

Selection (Видалити до виділеного), ***Clear Command History*** (Видалити всю передісторію).

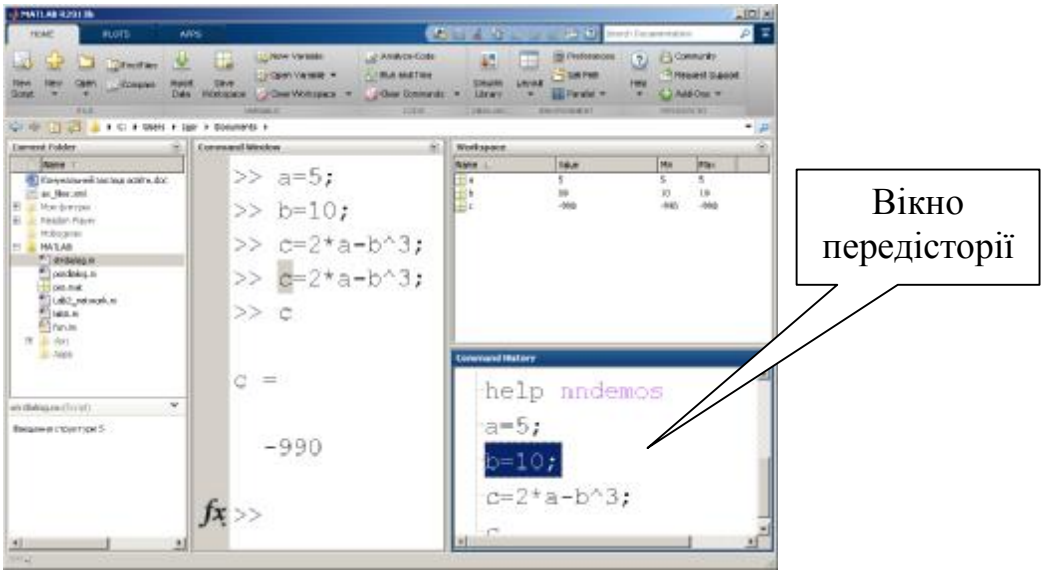
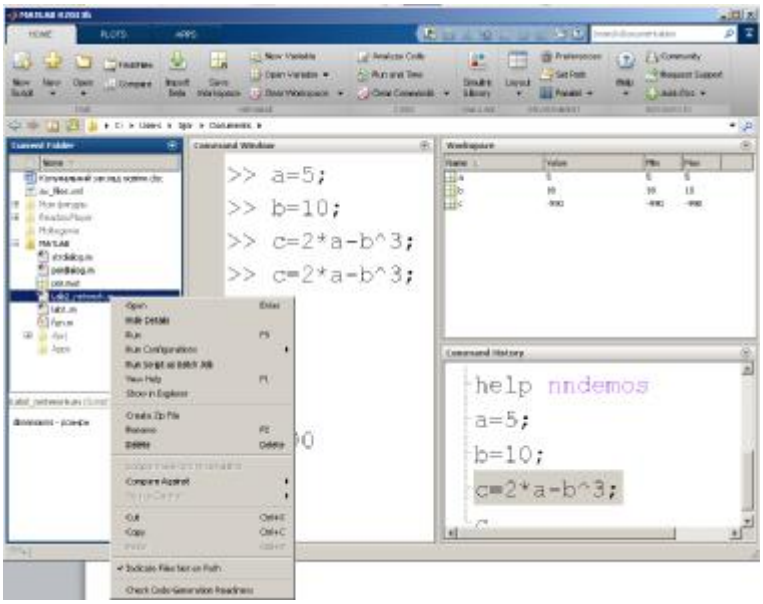


Рис 2.4. Вікно ***Command History*.**

Вікно для перегляду поточного каталогу ***Current Folder*** показано на рис. 2.5, воно дозволяє відкривати, переглядати і виконувати пошук інформації в файлах системи Matlab, мають необхідне розширення. Вікно поточного каталогу в Matlab представлено на рис. 2.6.

Рис. 2.5. Вікно передісторії ***Current Folder***



Контекстне меню, пов'язане з правою кнопкою миші, дозволяє реалізувати наступні дії: ***Open*** (Відкрити), ***Run*** (Виконати), ***View Help*** (Переглянути довідку), ***Import Data*** (Імпортувати дані), ***New File*** (Створити новий файл), ***Rename*** (Перейменувати), ***Delete*** (Видалити), ***Cut*** (Вирізати), **Copy** (Копіювати), ***Paste*** (Вставити), ***Add to Path*** (Додати до шляху доступу), **Refresh** (Оновити). На інструментальній панелі системи Matlab мається інформаційне вікно ***Current Directory***, за допомогою якого завжди забезпечується доступ до списку раніше викликаних поточних каталогів, як це показано на рис. 2.6. Це дозволяє швидко переходити від одного каталогу до іншого, отримуючи доступ до файлам/

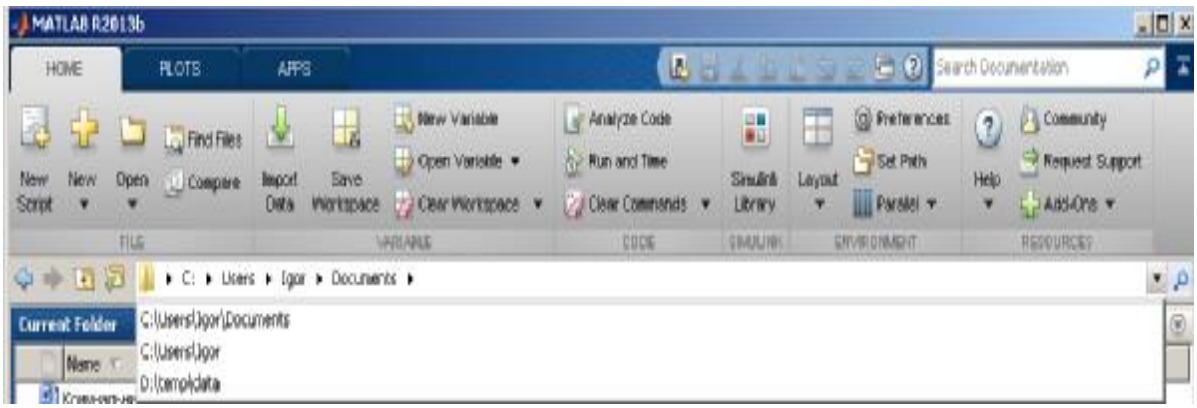


Рис. 2.6. Вікно поточного каталогу

Вікно робочої області ***Workspace*** показано на рис. 2.7 і містить список змінних (іменовані масиви), накопичених в пам'яті в процесі роботи, розширення списку змінних при зверненні до функцій, виконанні *M*-файлів і завантаженні збережених змінних.

На рис. 2.7 показано список змінних, що охоплює всі типи даних: це масиви дійсних і комплексних (*complex*) чисел подвоєною точності *double array*, різновиди цих масивів, опис глобальної змінної *global* і зміні логічного типу *logical*. Крім того, список містить масив рядків *char array*, масив розрідженій матриці *sparse array*, масив записів *struct array*, масив комірок *cell array*, а також спеціальні типи цілочислових масивів *uint8* і *uint32* і об'єкт типу *inline object*. Для кожної змінної вказується розмір масиву *Size*, обсяг пам'яті *Bytes* і тип масиву *Class*. Вікно робочої області, як і всі вікна робочого столу системи *Matlab*, супроводжується контекстним меню, яке включає наступні опції: *Open Selection*. (Відкрити виділене), *Plot Catalog* (Побудувати графік), *Select All* (Виділити все), *Import Data* (Імпорт даних), *Save Selection As…* (Зберегти виділене як…), *Save Workspace As…* (Зберегти робочу область як...), *Delete Selection* (Видалити виділене), *Delete Workspace* (Видалити робочу область).

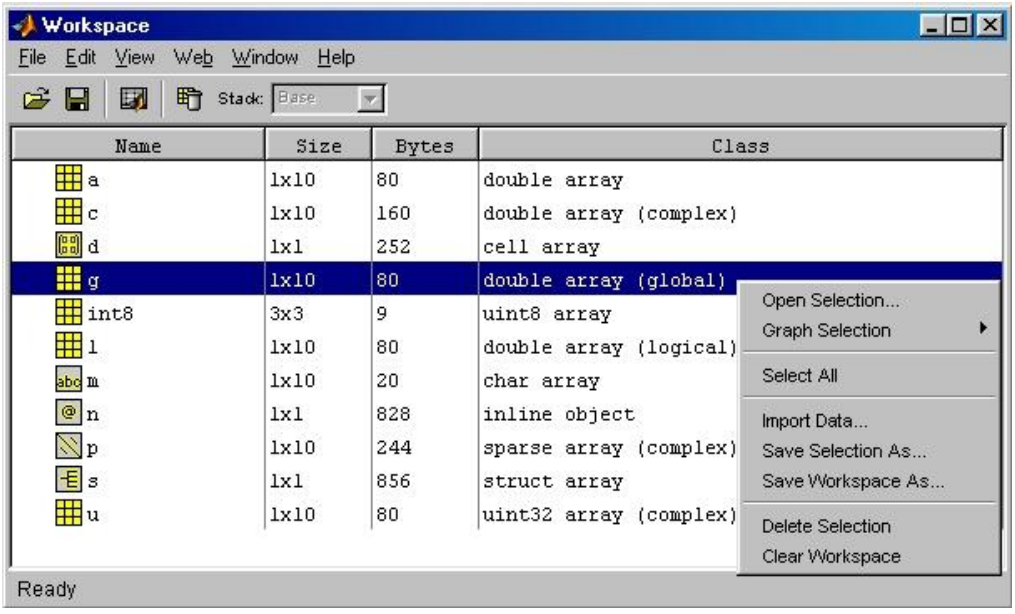


Рис. 2.7. Вікно робочої області ***Workspace.***

2.1.2 GUI- інтерфейс для ППП NNT

Відмінною особливістю 12-го випуску програмних продуктів фірми MathWorks є включення до їх складу різних інструментальних засобів організації діалогу з користувачем. Як правило, це GUI-інтерфейси (GUI - Графічний інтерфейс користувача, NNT- Neural Network Tools). Не є винятком і пакет по нейронних мережах ППП NNT, до складу якого входить інструментальний засіб ***NNTool***. Цей графічний інтерфейс дозволяє, не звертаючись до командного вікна системи MATLAB, виконувати створення, навчання, моделювання, а також імпорт і експорт нейронних мереж і даних, використовуючи тільки інструментальні можливості GUI-інтерфейсу. Звичайно, такі інструменти найбільш ефективні лише на початковій стадії роботи з пакетом, оскільки мають певні обмеження. Зокрема, інтерфейс ***NNTool*** допускає роботу тільки з найпростішими одношаровими і двошаровими нейронними мережами, але при цьому користувач виграє в часі і ефективності освоєння нових об'єктів. Виклик GUI- інтерфейсу *NNTool* можливий або командою **nntool** з командного рядка: >> **nntool**;

Після виклику на екрані терміналу з'являється вікно ***Neural Network / Data Manager*** (Управління мережею / даними) (рис. 2.8).

Тут:

Help – кнопка виклику вікна підказки *Neural Network/Data Manager Help* (рис. 2.1);

New… – кнопка виклику вікна з двома вкладками для формування нейронної сітки та даних *Create Network or Data* (рис. 2.11);

Import.. – кнопка виклику вікна для імпорту або завантаження даних Import Network / Data Manager (рис. 2.9);

Export.. – кнопка виклику вікна для експорту або запису даних у файл Export or Save from Network / Data Manager (рис. 1.10).

Кнопки ***Open, Delete, Export*** стають активними тільки після створення та активізації даних, що відносяться до послідовностей входу, цілі, виходу або помилок мережі. Кнопка ***View*** дозволяє переглянути, а кнопка Delete видалити активізовані дані.

Насамперед розглянемо призначення та способи роботи з перерахованими вище вікнами. Вікно ***Network / Data Manager Help***. Це вікно підказки показано на рис. 2.10 і описує правила роботи з диспетчером ***Neural Network/Data Manager*** при створенні нейронної мережі.

Щоб створити нейронну мережу, необхідно виконати наступні операції:

- Сформувати послідовності входів і цілей (кнопка ***New***…/Вікно ***Create Network or Data***/вкладка *Data* рис.2.11), або завантажити їх з робочої області системи MATLAB або з файлу (кнопка ***Import***).
- Створити нову нейронну мережу (кнопка ***New***…/Вікно ***Create Network or Data***/вкладка *Network*), або завантажити її з робочої області системи MATLAB або з файлу (кнопка ***Import***).

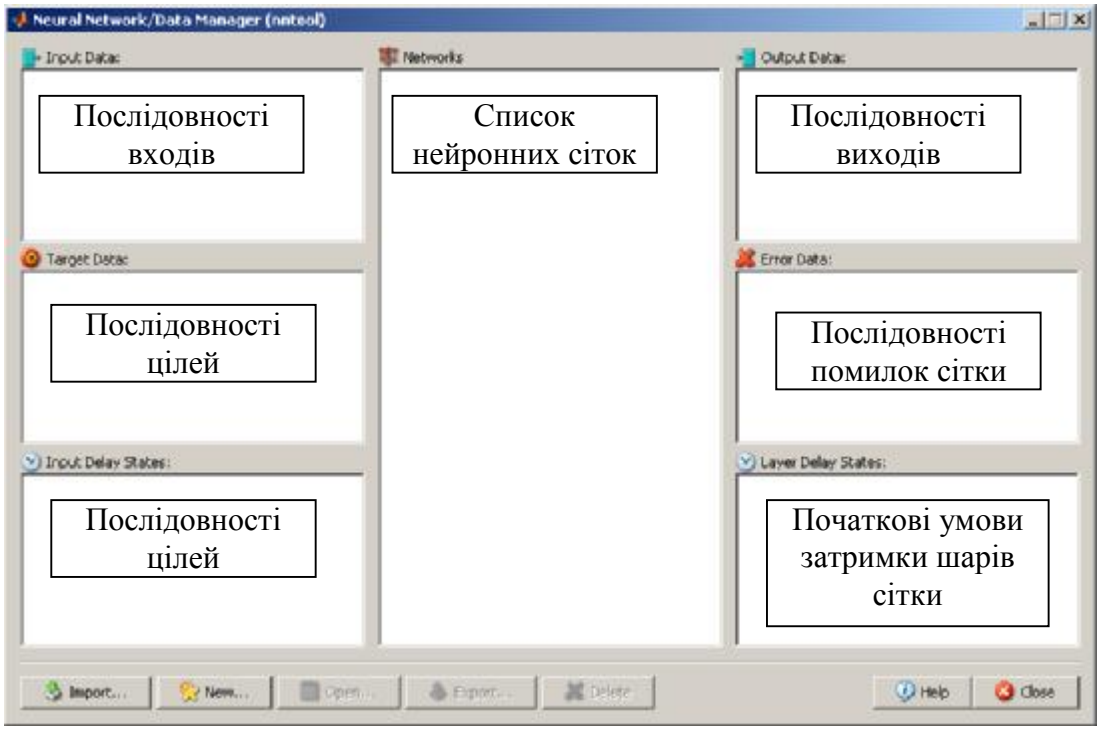


Рис. 2.8 ***Neural Network/Data Manager***

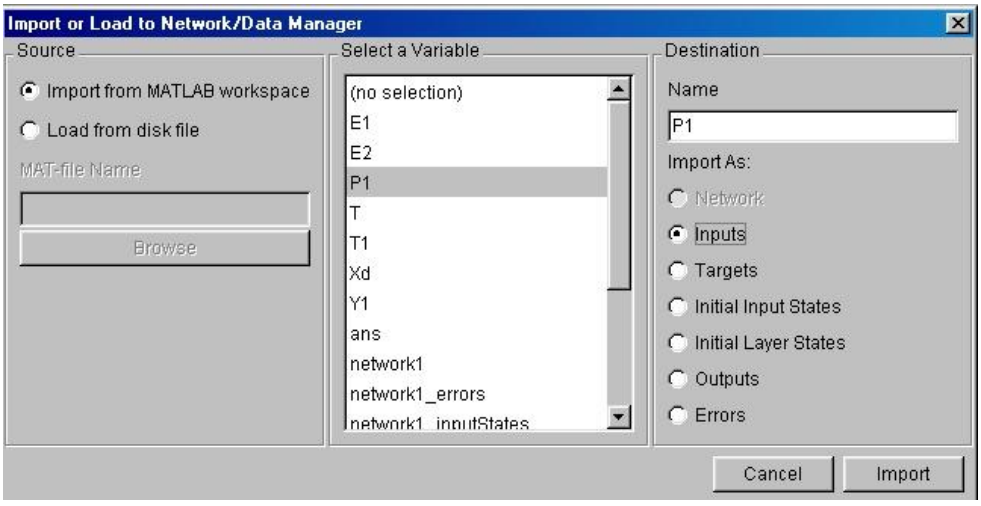


Рис. 2.9 Вікно *Import Network* для імпортування нейронної сітки

•Натиснути кнопку *Train..*, щоб відкрити вікно для завдання параметрів процедури навчання. Відкрити вікно *Network* для перегляду, ініціалізації, моделювання, навчання та адаптації мережі. Вікно ***Create Network or Data*** показано на рис. 2.11 і включає 2 області редагування тексту для запису імені даних (область *Name*) і введення самих даних (область *Value*), а також 6 кнопок для вказівки типу даних, що вводяться. Після введення даних натискаємо кнопку «*Create*» для додавання даних до НС.

Розрізняють такі типи даних:

- ***Inputs*** *(Входи)* – послідовність значень входів;
- ***Targets*** (Цілі) – послідовність значень мети;

- ***Input Delay States*** (Стани лінії затримки входу) – початкові умови лінії затримки на вході;
- ***Layer Delay States*** (Стани лінії затримки шару) – початкові умови лінії затримки в шарі;
- ***Outputs*** (Виходи) – послідовність значень виходу мережі;
- ***Errors*** (Помилки) – різниця значень цілей і виходів.

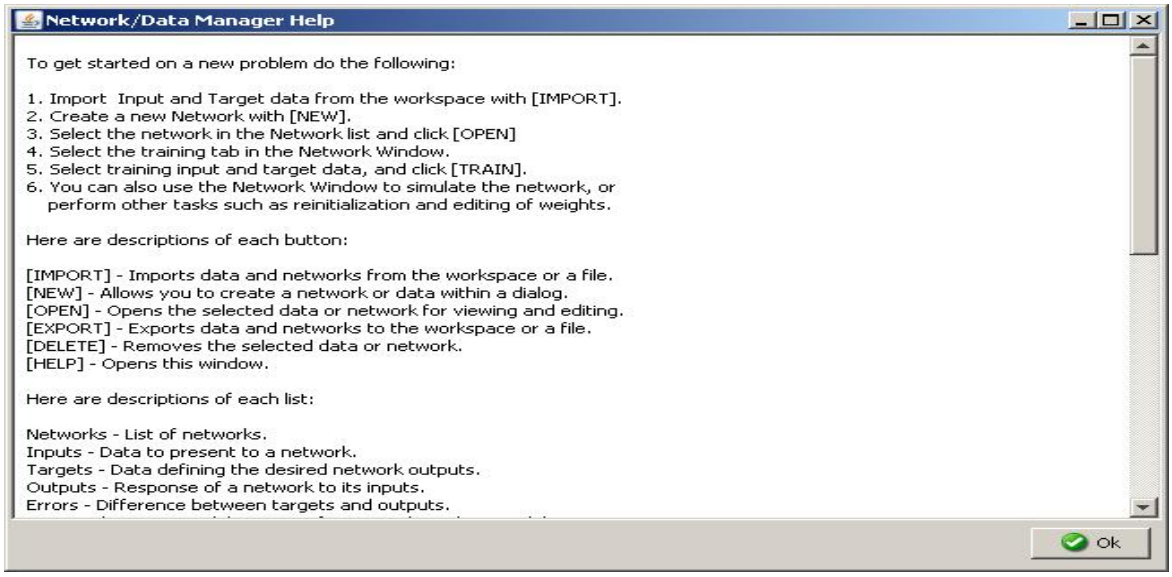


Рис. 2.10. Вікно ***Network / Data Manager Help***

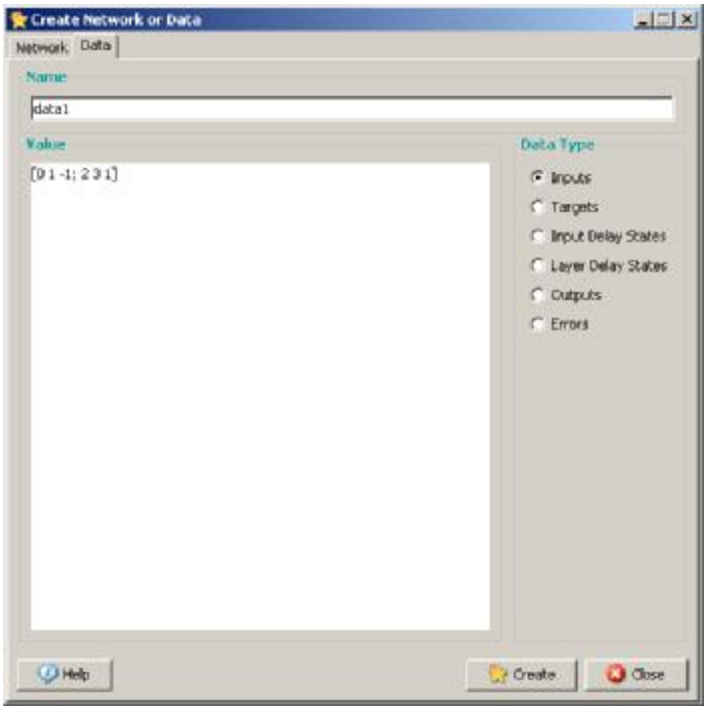


Рис. 2.11 Вікно *Create Network or Data,* вкладка *Data* для введення вхідних даних

Як правило, користувач задає тільки послідовності входу і мети, тобто типи даних ***Inputs*** і ***Targets***. При цьому слід пам'ятати, що при адаптації нейронної мережі дані повинні бути представлені у вигляді масиву комірок. Вікно *Create Network or Data / вкладка Network*. Це вікно показано на рис. 2.12 і включає поля для завдання параметрів створюваної мережі. Залежно від типу мережі кількість полів та їх назви змінюються.

Звернемося до опису полів:

Name (Ім'я мережі) – стандартне ім'я мережі, що привласнюється GUI-інтерфейсом NNTool; в процесі створення нових мереж порядковий

номер буде змінюватися автоматично, за необхідно можна замінити змістовним.

Network Type – необхідно вибрати тип нейронної мережі.

Input data (Вхідні дані) – необхідно вибрати ім'я послідовності значень входу мережі.

Target data – необхідно вибрати ім'я послідовності значень мети.

Training function (Функція навчання) – список функцій для навчання НС.

Adaption learning function (Функції налаштування для режиму адаптації) – список функцій налаштувань.

Performance function (Функція якості навчання) – список функцій оцінки якості навчання.

Number of layers (Кількість шарів) – кількість шарів нейронної мережі.

Properties for (Властивості) – список шарів: Layer 1 (Шар 1), Layer 2 (Шар 2).

Number of neurons (Кількість нейронів) – кількість нейронів у шарі.

Transfer function (Функція активації) – список функцій для активації шару.

Після введення даних натискаємо кнопку «*Create*» для створення НС з вказаним ім'ям, властивостями та вхідними та цільовими значеннями.

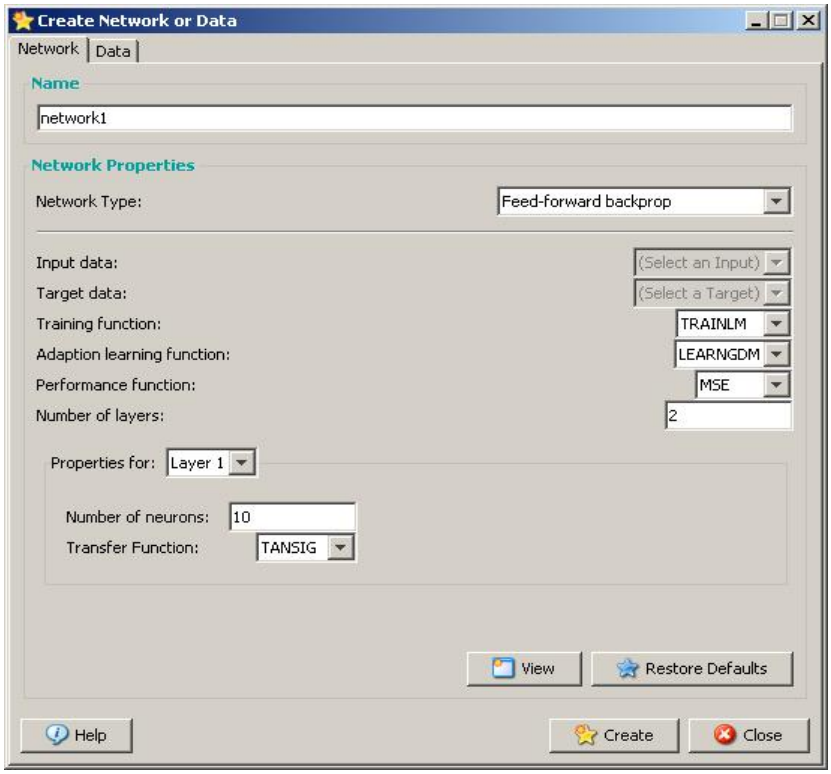


Рис.2.12 Вікно *Create Network or Data,* вкладка *Network* для створення нової нейронної сітки

Вікно *Import or Load to Network / Data Manager* показано на рис. 2.13 і включає 3 поля:

Source (Джерело) – поле для вибору джерела даних. Це або робоча область системи MATLAB (кнопка вибору *Import from MATLAB Workspace*), або файл (кнопка вибору *Load from disk file*).

Якщо вибрана перша кнопка, то в полі ***Select a Variable*** ви можете бачити всі змінні робочої області і, вибравши одну з них, наприклад P_1, можете

визначити її в полі ***Destination*** *(Призначення)* як послідовність входу *Inputs* (*Входи*).

Якщо вибирається кнопка *Load from disk file*, то активізується поле MAT-file *Name* і кнопка *Browse*, що дозволяє почати пошук і завантаження файлу з файлової системи.

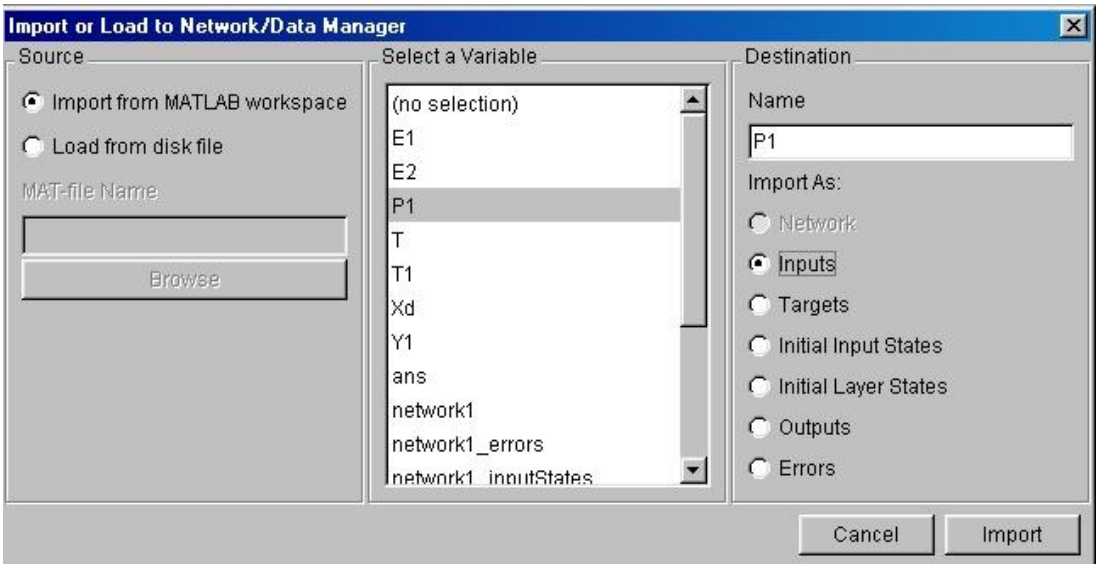


Рис. 2.13 Вікно *Import or Load to Network / Data Manager*

Вікно *Export or Save from Network / Data Manager*. Це вікно показано на рис. 2.14 і дозволяє передати дані з робочої області GUI- інтерфейсу NNTool в робочу область системи MATLAB або записати їх у вигляді файлу на диску.

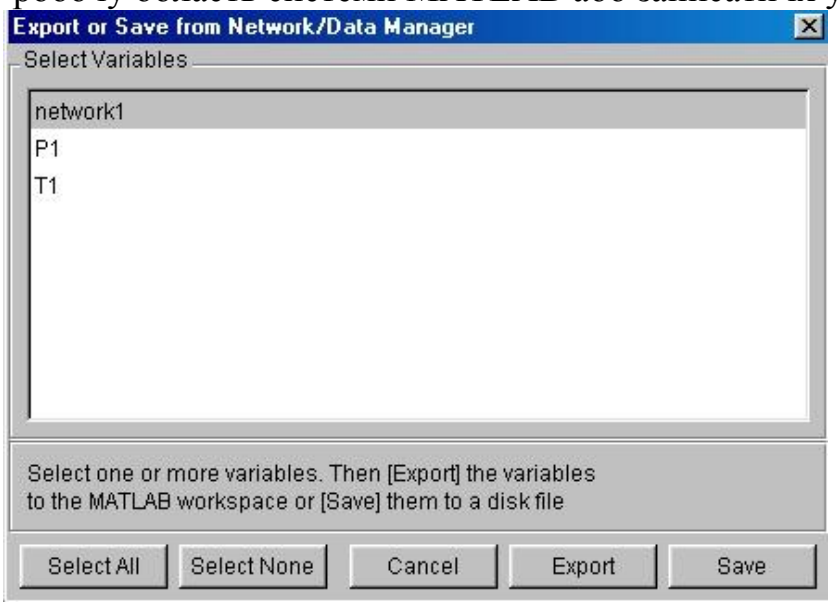


Рис. 1.14 Вікно *Export or Save from Network - Data Manager*

У даному випадку нами обрана змінна *network1*, яка належить до класу *network object* і описує нейронну мережу. Після того як ця змінна експортована в робочу область, можна, наприклад, побудувати модель нейронної мережі в системі *Simulink* за допомогою оператора *gensim*.

Для подальшої роботи з нейронною сіткою необхідно повернутися до вікна *Neural Network/Data Manager*, виділити змінну *network1* подвійним клацанням лівою кнопкою миші, в результаті буде відкрито діалогову панель *Network*, що показана на рис. 2.15.

Воно відкривається тільки в тому випадку, коли у вікні *Neural Network / Data Manager* виділена створена мережа і стають активними вкладки *View*, *Train*, *Simulate*, *Adapt*, *Reinitialize Weights*, *View/Edit Weights*. Панель має 6 закладок: *View* (*Переглянути*) - структура мережі; *Train* (*Навчання*) - навчання мережі; *Simulate* (*Моделювання*) - моделювання мережі; *Adapt* (*Адаптація*) - адаптація і налаштування параметрів мережі; *Reinitialize Weights* (*Ініціалізація ваг*) - завдання початкових ваг і зміщень; *View/Edit Weights* (*Перегляд/Редагування ваг*) - перегляд і редагування встановлених ваг і зсувів. Особливості роботи з відповідними вікнами будуть розглянуті на наведених нижче прикладах створення конкретних нейронних мереж.

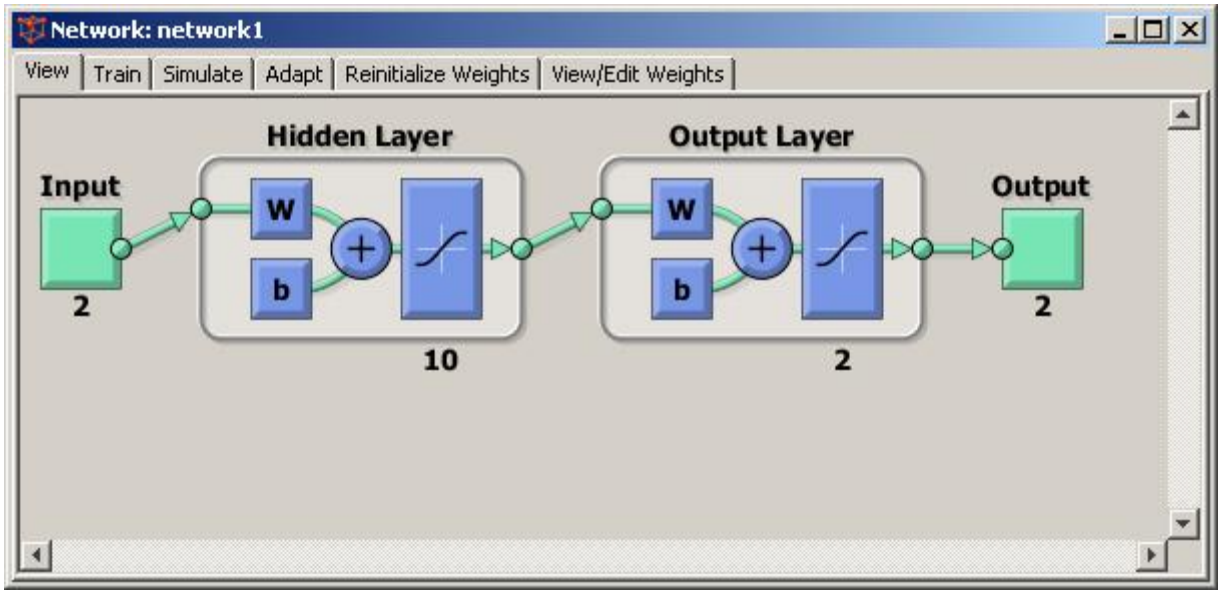


Рис. 2.15 Діалогова панель ***Network***

Типи нейронних мереж Network Type (Тип нейронних мереж) - список мереж, доступних для роботи з інтерфейсом NNTool. Для зручності цей список повторений у наступній табл.2.1. Інтерфейс NNTool дозволяє створювати нейронні мережі тільки з одним або двома шарами.

Таблиця 2.1 – Опис нейронних мереж, які реалізує програма MatLab

№п/п	Тип мережі	Назва мережі
1	Competitive	конкуруюча мережа
2	Cascade - forward backprop	Каскадна мережу з прямим розповсюдженням сигналу і зворотним поширенням помилки
3	Elman backprop	Мережа Елмана зі зворотним поширенням помилки
4	Feed - forward backprop	Мережа з прямим розповсюдженням сигналу і зворотним поширенням помилки
5	Time delay backprop	Мережа з запізненням і зворотним поширенням помилки
6	Generalized	Узагальнена регресійна мережа

№п/п	Тип мережі	Назва мережі
	regression	
7	Hopfield	Мережа Хопфілда
8	Linear layer (design)	Лінійний шар (створення)
9	Linear layer (train)	Лінійний шар (навчання)
10	LVQ	Мережа для класифікації вхідних векторів
11	Perceptron	Персептрон
12	Probabalistic	Імовірнісна мережа
13	Radial basis (exact fit)	Радіальна базисна мережа з нульовою помилкою
14	Radial basis (fewer neurons)	Радіальна базисна мережа з мінімальним числом нейронів
15	Self-organizing map	Самоорганізована карта Кохонена

Примітки:

Для мереж 2, 3, 7 в даній версії інтерфейсу NNTool не забезпечується перегляд структурних схем.

Мережі 5, 9 допускають введення ліній затримок на вході.

Мережі 3 допускають введення ліній затримок в шарі.

Мережі з двома шарами мають послідовну структуру, коли вихід першого шару служить входом другого шару. Виняток становлять мережі 3, які допускають наявність зворотного зв'язку в першому шарі і передачу вхідного сигналу на входи обох шарів.

Приклади нейронних мереж. Перелік усіх прикладів нейронних мереж, включених в ППП NNT, можна отримати по команді:

>> help nndemos

Команда *help nndemos* дозволяє переглянути вбудовані приклади використання нейронних сіток для розв’язання прикладних задач по категоріям.

Ось, деякі з них:

Функція згладжування та апроксимації

fit_house_demo – Оцінювання ціни будинку.

Розпізнавання образів і класифікація

classify_crab_demo - класифікація крабів.

classify_wine_demo - класифікація вина.

cancerdetectdemonnet - виявлення раку.

Кластеризація

cluster_iris_demo – кластеризація ірисів.

Динамічне моделювання і прогнозування

model_maglev_demo – цей приклад ілюструє, як нейронна мережа може моделювати динамічну систему.

Simulink & Control

predcstr - прогнозний контроль ядерного реактору.

mrefrobotarm - Reference управління рукою робота.

Self-Organizing Networks

*democ*1 - Конкурентне навчання.

demosm1 - Одновимірна самоорганізована карта.

demosm2 - двовимірна самоорганізована карта.

Таблиця 2.2. – Функції для створення нейронних сіток

Оператор	**Призначення**
network	Створення шаблона нейронної сітки
Персептрон	
newp	Створення персептрона
newlin	Створення лінійного шару
newlind	Створення лінійного шару шляхом розв'язку лінійного рівняння
Багатошарові сітки	
feedforwardnet	Створення сітки сітки прямої передачі сигналу
newff	Створення сітки прямої передачі сигналу з зворотнім поширенням помилки
newfftd	Створення сітки прямої передачі сигналу з запізненням
newcf	Створення каскадної сітки прямої передачі сигналу
Радиальні базисні сітки	
newrb	Створення радиальної базисної сітки
newrbe	Створення радиальної базисної сітки з нульовою помилкою
newrgrnn	Створення узагальненої регресійної сітки
newpnn	Створення ймовірнісної нейронної сітки
Сітки для розв'язання задач класифікації	
newlvq	Створення сітки для розв'язання задач класифікації
Самоорганізовані сітки	
newc	Створення конкуруючого шару Кохена
newsom	Створення самоорганізовані сітки Кохена
Рекурентні сітки	
newhop	Створення сітки Хопфілда
newelm	Створення сітки Елмана

Переглянути вбудовані функції для роботи з лінійними сітками та отримати необхідну інформацію про вбудовані М-функції, що знаходяться в бібліотеці NNTool можна по команді: **>> *help toolbox\nnet***

Таблиця 2.3. – Опис функцій для роботи з лінійними сітками

Функція	**Призначення**
Формування нейронної сітки	
newlind	Створення лінійного шару
Робота з нейронною сіткою	
sim	Моделювання сітки
init	Ініціалізація нейронної сітки
adapt	Процедура адаптації
train	Процедура навчання
view	Перегляд нейронної сітки
Функції зважування	
drotprod	Вагова функція скалярного добутку векторів
convwf	Вагова функція згортання
negdist	Вагова функція негативної відстані
scalprod	Вагова функція скалярного добутку
mandist	Вагова функція з манхетенською відстанню
Функції накопичування	
netsum	Сума зважених входів
netprod	Поелементний добуток зважених входів
Функції активації	
purelin	Лінійна функція
tansig	Функція гіперболічного тангенса
logsig	Логарифмічно-сигмоідальна функція
Функції ініціалізації	
initlay	Пошарова ініціалізація
initwb	Ініціалізація ваг та шарів
initzero	Ініціалізація нульових ваг та шарів
mse	Середньоквадратична помилка
Функції настройки параметрів персептрону	
learnwh	Правило настройки Вудро-Хофа
Функції адаптації	
adaptwb	Функція адаптації ваг та шарів
Функції навчання	
train	Функція навчання нейронної сітки
trainc	Навчання нейронної сітки з циклічним представленням вхідних даних
trainr	Навчання нейронної сітки з випадковим представленням вхідних даних
trains	Адаптивне навчання нейронної сітки

2.2. Індивідуальне завдання № 2.
Робота з нейронною мережею в командному режимі

Мета роботи: Побудова нейронної мережі для апроксимації обраної функції, використовуючи функції пакета *Neural Networks Toolbox*.

Студент має виконати три завдання за наведеними нижче прикладами.

Завдання 1.

Побудуємо нейронну мережу для апроксимації обраної функції, використовуючи функції пакета *Neural Networks Toolbox*. До складу пакету входить більше 160 різних функцій, що дають можливість створювати, навчати і досліджувати нейронні мережі.

Для виконання роботи необхідно запустити *Matlab* і перейти в командне вікно.

Створимо узагальнено-регресійну нейронну мережу (мережа типу GRNN) з ім'ям «*A*», що реалізовує функціональну залежність між входом і виходом виду $y = x^2$ на відрізку [-1, 1], використовуючи такі експериментальні дані:

x = [-1 -0.8 -0.5 -0.2 0 0.1 0.3 0.6 0.9 1],

y = [1 0.64 0.25 0.04 0 0.01 0.09 0.36 0.81 1].

Перевірку якості відновлення наведеної залежності здійснимо, використовуючи дані контрольної вибірки x_1 = [-0.9 -0.7 -0.3 0.4 0.8], яким відповідають значення y_1 = [0.81 0.49 0.09 0.16 0.64].

Процедура створення та використання даної нейронну мережу описується наступним чином (рис. 1), необхідно в командному вікні *Matlab* ввести команди:

```
x = [-1 -0.8 -0.5 -0.2 0 0.1 0.3 0.6 0.9 1];  % Завдання вхідних значень
y = [1 0.64 0.25 0.04 0 0.01 0.09 0.36 0.81 1]; % Завдання цільових значень
A = newgrnn (x, y, 0.01); % Створення нейронну мережу з відхиленням 0.01
Y1 = sim (A, [ -0.9 -0.7 -0.3 0.4 0.8 ]) % Опитування нейронну мережу
```

Результат розрахунку за наведеними вище командами показано на рис.2.17.

Розраховано вектор:

*Y*1 = 0.8200 0.6400 0.0400 0.0900 0.8100

Як видно, точність апроксимації в даному випадку вийшла не дуже високою – максимальна відносна похибка апроксимації становить 30,61 %.

Можна спробувати поліпшити якість апроксимації за рахунок підбору величини відхилення, але в умовах прикладу прийнятний результат легко досягається при використанні мережі з радіальними базисними елементами типу *newrbe*:

Введемо наступні команди:

***A* = *newrbe*(*x*, *y*);**
***Y*1 = *sim* (*A*, [-0.9 -0.7 -0.3 0.4 0.8])** % Опитування нейронну мережу

Результат:

*Y*1 = 0.8100 0.4900 0.0900 0.1600 0.6400

Результати можуть відрізнятися від наведених вище, за рахунок якості навчання, результату навчання.

Неважко побачити, що застосування мережі типу *newrb* призводить тут не просто до інтерполяції заданих навчальною вибіркою значень, а дійсно до точного відновлення заданої залежності – принаймні, для використаних точок контрольної послідовності.

Створену мережу можна зберегти для подальшого використання набором в командному рядку команди *save('A')*; при цьому буде створений файл *A.mat*, тобто файл з ім'ям нейронну мережу і розширенням mat. У наступних сеансах роботи цю мережу можна завантажити, використовуючи функцію *load ('A')*. Природно, допустимі всі інші форми запису операторів *save* і *load*.

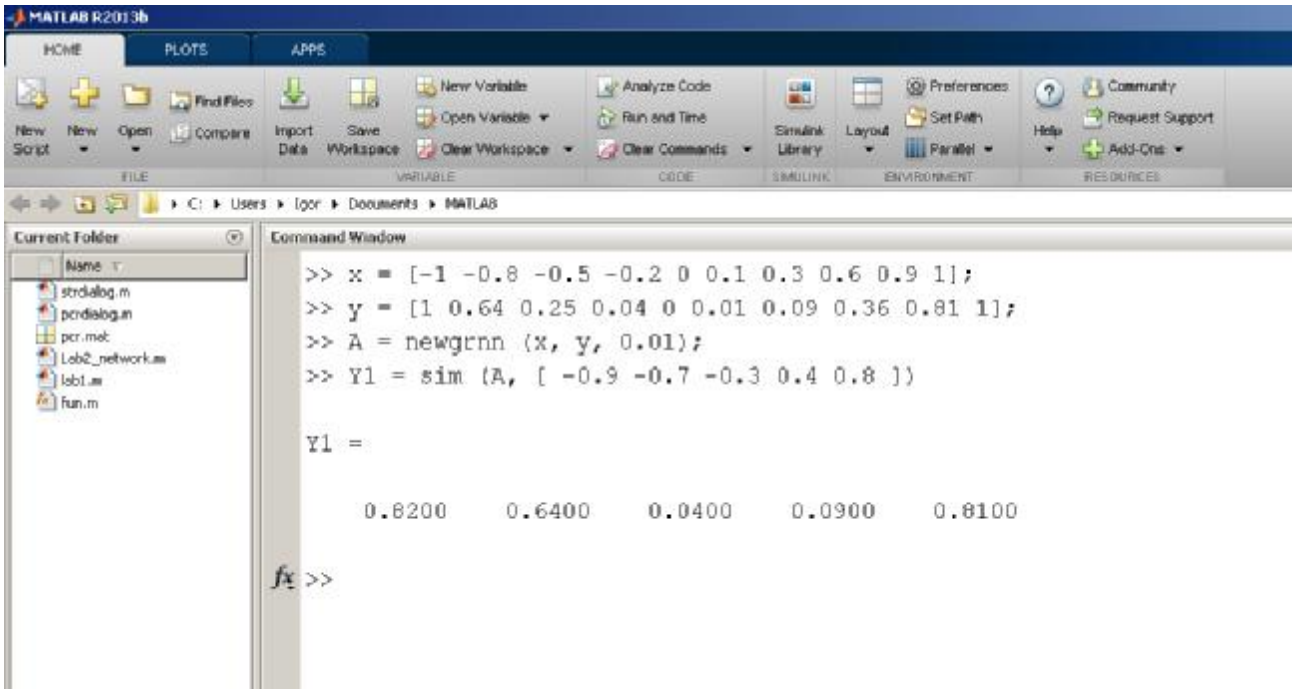


Рис.2.16. Створення та використання даної нейронну мережу

Завдання 2.

Розглянемо тепер завдання відновлення деякої, взагалі кажучи, невідомої залежності за наявними експериментальними даними з використанням лінійної нейронну мережу.

Нехай експериментальна інформація задана значеннями

x = [1.0 1.5 3.0 -1.2], *y* = [0.5 1.1 3.0 -1.0].

Створимо вектори входу і цілей:

```
x = [1.0 1.5 3.0 -1.2];
y = [0.5 1.1 3.0 -1.0];
```

Тепер створимо лінійну нейронну мережу:

```
b = newlind(x, y); % Створення лінійної нейронну мережу з ім'ям b
```

Проведемо опитування мережі для значення входу, рівного 3.0 (цьому, згідно з експериментальними даними, відповідає цільове значення 3.0) (рис. 2.17).

```
y1 = sim(b, 3.0) % Опитування мережі
```

Результат: y1 = 2.7003

Похибка відновлення за даними навчальної вибірки в даному випадку – 10%. Зазначимо, що в умовах як першого, так і другого прикладу дати будь-яку оцінку граничної величини похибки апроксимації неможливо, особливо для значень входів, що виходять за межі діапазону входів навчальної послідовності.

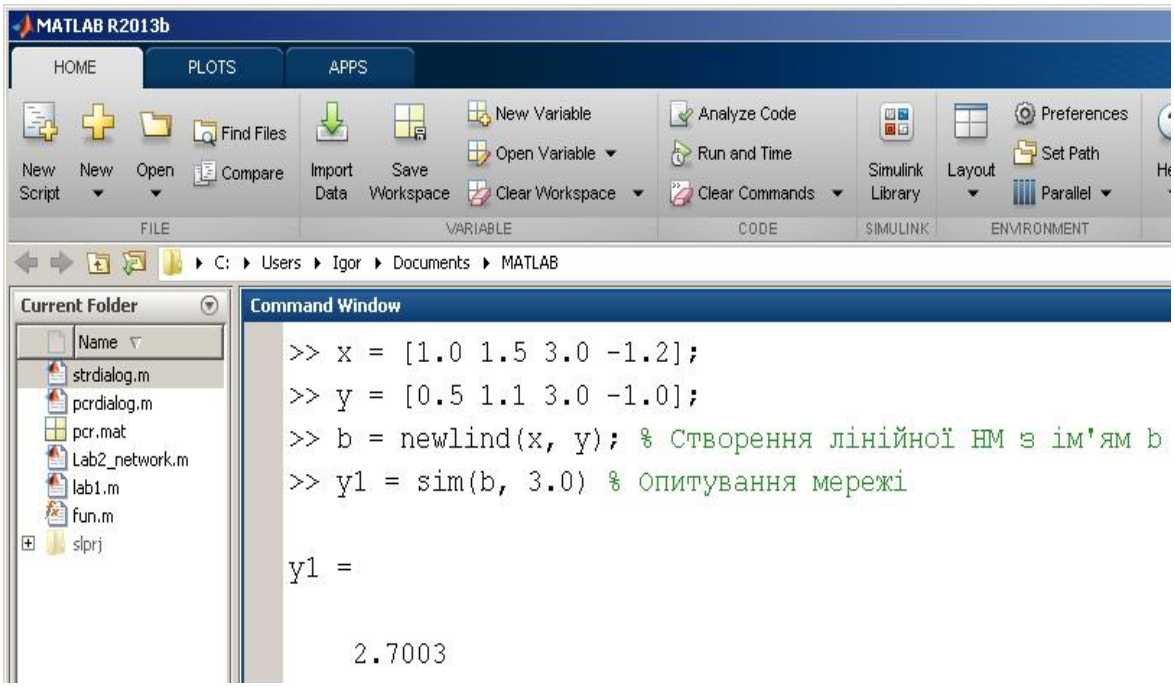


Рис. 2.17. Опитування мережі

На жаль, це є характерною особливістю нейромережевих моделей. Для переважної кількості завдань, що вирішуються за допомогою апарату нейронних мереж - не тільки для задач класифікації або прогнозу - будь-яких імовірнісних оцінок точності одержаних рішень отримати не вдається.

Завдання 3.

У середовищі *Matlab* необхідно побудувати і навчити нейронну мережу для апроксимації заданої таблицею функції, $y_i = f(x_i)$, $i = 1{,}20$. Розробити

програму, яка реалізує нейромережевий алгоритм апроксимації і виводить результати апроксимації у вигляді графіків.

Нехай у середовищі Matlab необхідно побудувати і навчити нейронну мережу для апроксимації заданої таблицею функції $y_i = f(x_i)$ = [2.09 2.05 2.19 2.18 2.17 2.27 2.58 2.73 2.82 3.04 3.03 3.45 3.62 3.85 4.19 4.45 489 5.06 5.63 5.91], i = 1,20.

У математичному середовищі Matlab створюємо новий *M*–File, в якому записуємо код програми створення і навчання нейронної мережі з використанням вбудованих функцій пакету мереж *Neural Networks Toolbox*.

Для вирішення скористаємося функцією *newff*() – створення «класичної» багатошарової НС з навчанням за методом зворотного поширення помилки.

Для цього введемо наступні команди:

```
P = zeros(1,20);
for i = 1:20 % створення масиву
P(i) = i*0.1; % вхідні дані (аргумент)
end
T=[2.09 2.05 2.19 2.18 2.17 2.27 2.58 2.73 2.82 3.04 3.03 3.45 3.62 3.85 4.19
4.45 489 5.06 5.63 5.91]; % цільові значення (значення функції)
net = newff(P,T,5); % створення нейронної мережі
net.trainParam.epochs = 100; % завдання числа епох навчання
net=train(net,P,T); % навчання мережі
y = sim(net,P); % опитування навченої мережі
figure (1);
hold on;
xlabel ('P');
ylabel ('T');
plot(P,T,P,y,'o'), grid;
```

Варіанти індивідуальних завдань.

Створити НС з ім'ям «lab2.mat», що реалізовує функціональну залежність між входом і виходом виду вказаним за варіантом на відрізку [-*N*, *N*] (*N*– номер студента за за списком групи):

1. $Y=5x^3-4x^2+3x-2$;	6. $Y=\ln(Nx)$;
2. $Y=\sin(N\text{x})$;	7. $Y=0{,}5\cos(Nx)$;
3. $Y=Nx^5+(N-1)x^3+3x+N$;	8. $Y=\sqrt{\frac{N}{N+25}}$,
4. $Y=\sqrt{\frac{N^2-N}{N-2}}x$,	9. $Y=\frac{Nx+5}{x+1}$,
5. $Y=N*x$	10. $Y=\sqrt[n]{Nx}$

Застосовуючи різні алгоритми навчання нейронну мережу (*newgrnn, newrbe, newlind, newff)*, визначити похибку апроксимації.

2.3. Використання GUI-інтерфейсу пакета нейронних мереж

2.3.1. Створення нейронної мережі

GUI- інтерфейс для *NNTool* дозволяє, не звертаючись до командного вікна системи *Matlab*, виконувати створення, навчання, моделювання, а також імпорт та експорт нейронних мереж і даних. Звичайно, такі інструменти найбільш ефективні лише на початковій стадії роботи з пакетом, оскільки мають певні обмеження. Зокрема, інтерфейс NNTool допускає роботу тільки з найпростішими одношаровими і двошаровими нейронними мережами, але при цьому користувач виграє у часі і ефективності освоєння нових об'єктів. Виклик GUI- інтерфейсу NNTool можливий або командою *nntool* з командного рядка, або з вікна запуску додатків Launch Pad за допомогою опції NNTool з розділу Neural Network Toolbox. Після виклику на екрані терміналу з'являється вікно **Neural Network / Data Manager** (Управління нейронною мережею/даними).

Для прикладу, створимо, використовуючи графічний інтерфейс користувача, нейронну мережу для виконання операції $y = x^2$ при завданні векторів входу:

$$x = [-1\ -0.8\ -0.5\ -0.2\ 0\ 0.1\ 0.3\ 0.6\ 0.9\ 1];$$

і функції цілі:

$$y = [1\ 0.64\ 0.25\ 0.04\ 0\ 0.01\ 0.09\ 0.36\ 0.81\ 1];$$

Для відкриття основного вікна інтерфейсу необхідно в командному вікні MATLAB ввести команду:

>> nntool;

Виконання команди приведе до відкриття вікна створення нейронної мережі **Neural Network / Data Manager**.

Сформуємо послідовність входів і цілей в робочій області GUI-інтерфейсу, використовуючи вікно **Create Network or Data**.

З цією метою спочатку натиснемо кнопку *New*. Далі в поле *Name* вікна *Create Network or Data* вкладка *Data* введемо спочатку ім'я змінної *x*, а потім в області значень *Value* вектор значень:

[-1 - 0.8 - 0.5 - 0.2 0 0.1 0.3 0.6 0.9 1]

і, використовуючи радіокнопку *Inputs* (у правій частині вікна), вкажемо тип змінних (*Inputs* – Входи).

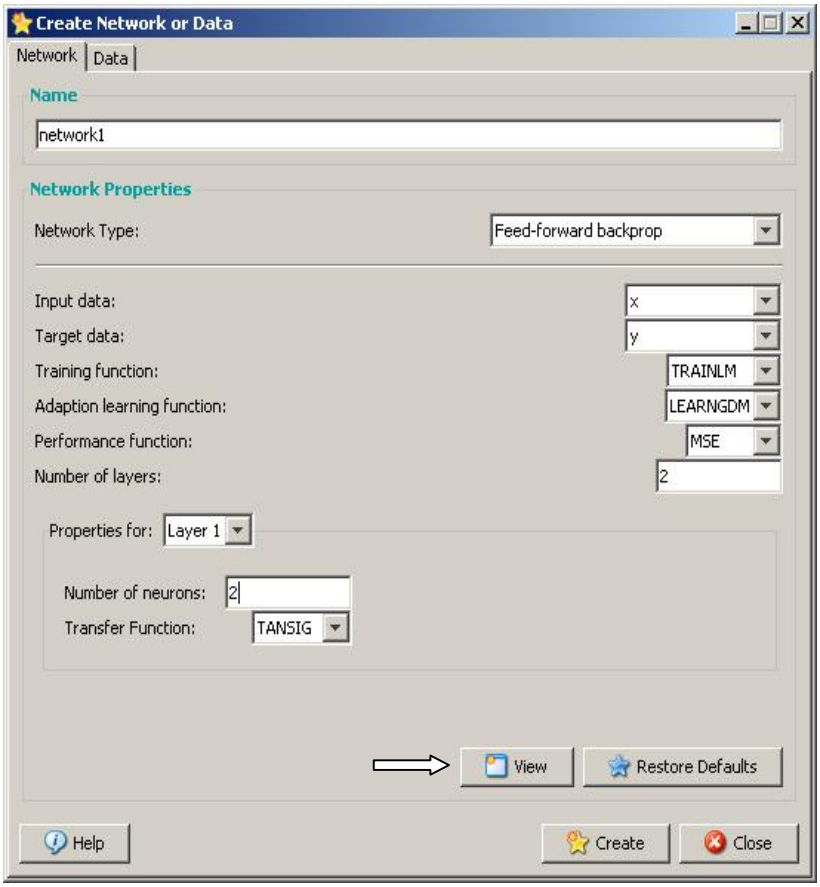
Введення завершимо натисканням кнопки *Create* (Створити). Система сповістить про створення нових даних і додавання їх до мережі.

Аналогічну операцію проробимо для вектора *y*, з зазначенням (радіокнопки *Targets*), що *y* це – вектор цільових даних. Введення завершимо натисканням радіокнопки *Create* (Створити). Система сповістить про створення нових даних і додавання їх до мережі.

Для створення нової нейронної мережі у вікні **Create Network or Data** натиснемо кнопку *New Network*.

У вікні, **Create Network or Data** виберемо нейронну мережу типу *feed - forward backprop* з прямою передачею сигналу і з зворотним поширенням помилки. При створенні мережі збережемо їй ім'я, що дається за замовчуванням (*network1*).

Діапазон входів визначимо (у вікні **Create Network or Data**) за допомогою опції *Input data*, виходів – *Target data*. Кількість нейронів (*Number of neurons*) першого шару (*Layer 1)* встановимо рівною двом.



Решта установок при створенні мережі залишимо за замовчуванням (рис.2,18).

Створення мережі завершимо натисканням кнопки *Create*. Після цього у вікні **Neural Network / Data Manager**, на вкладці *Networks* висвітиться ім'я нової створеної мережі – *network1*. Виберемо це ім'я за допомогою мишки, що призведе до активізації всіх кнопок вказаного вікна

Рис. 2.18. Заповнення полів для створення нейронну мережу.

Натиснувши на кнопку *View* у вікні **Create Network or Data** можна переглянути топологію нейронну мережу (рис.2.19).

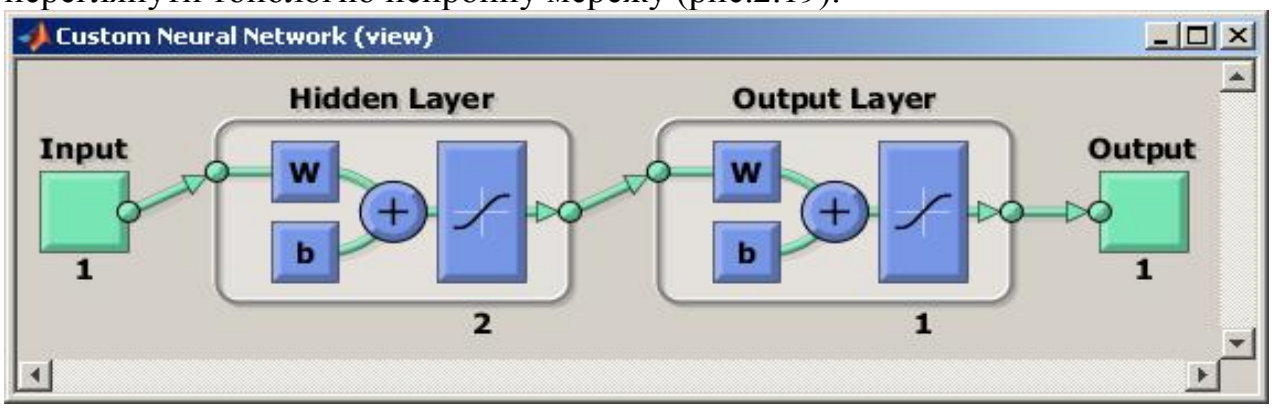


Рис.2.19. Перегляд створеної нейронну мережу.

Подвійне натискання миші на імені нейронну мережу *network1* приведе до відкриття діалогової панелі *Network: network1*.

Для введення в панелі *Network:network1* встановлених діапазонів і ініціалізації ваг скористаємося кнопками *Set Input Ranges* (Встановити діапазони) і *Initialize Weights* (Ініціалізувати ваги). Якщо потрібно повернутися до колишніх діапазонів, то слід вибрати кнопки *Revert Input Ranges* (Повернути діапазони) і *Revert Weights* (Повернути ваги), але в умовах прикладу це не потрібно.

2.3.2. Навчання нейронної мережі

Для навчання створеної мережі, вибирається закладка *Train* в панелі *Network: network1* і відкривається нова діалогова панель, яка має дві закладки:

- *Training Info (*Інформація про навчальні послідовності);
- *Training Parameters* (Параметри навчання).

Застосовуючи ці закладки, можна встановити імена послідовностей входу і цілі (на вкладці *Training Info* – у лівій її частині необхідно вказати *x* і *y*), а також значення параметрів процедури навчання (на вкладці *Training Parameters*; в умовах прикладу збережемо значення за замовчуванням).

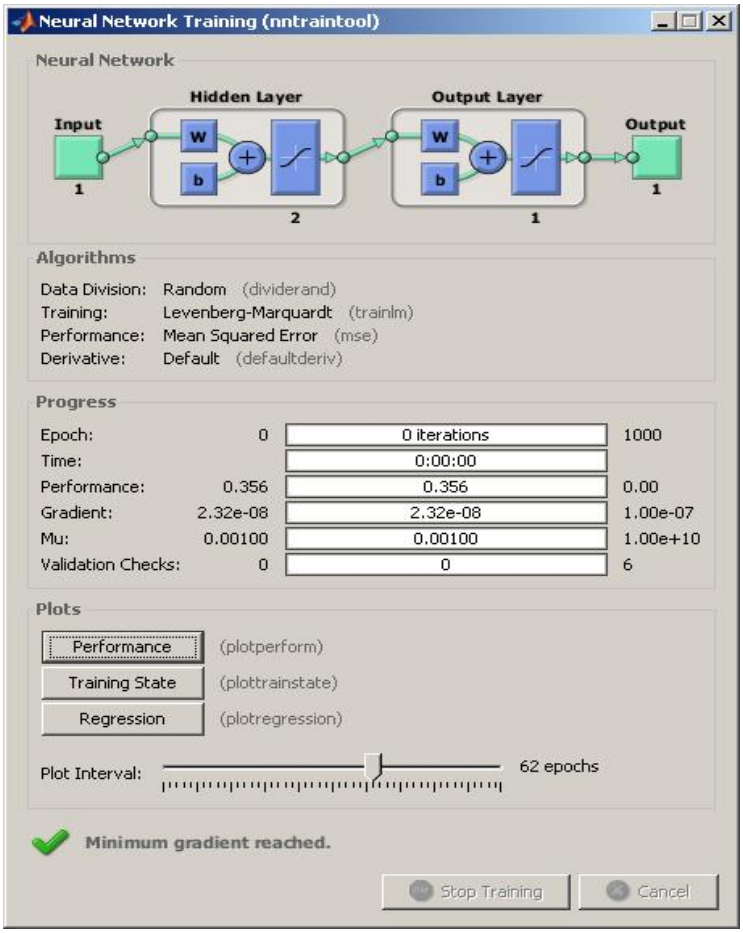


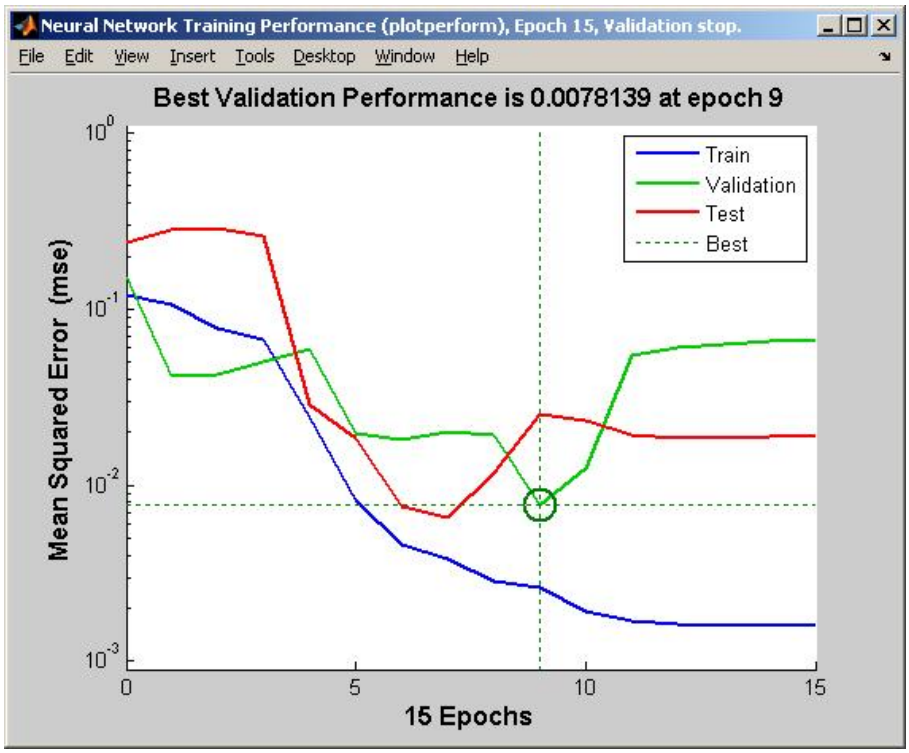
Рис. 2.20. Навчання нейронну мережу

Натискання кнопки *Train Network* викликає навчання мережі (рис .2.20).

Якість навчання мережі на будь-якій навчальній послідовності відображається графіком. Видно, що до кінця процесу навчання помилка стає дуже малою (вид даного малюнка при повторі обчислень може відрізнятися від наведеного).

Результати навчання можна переглянути у вікні *Neural Network/Data Manage*r, активізуючи імена послідовностей виходів *network1_outputs* або помилок *network1_errors*, і використовуючи кнопку *View*.

Слід зазначити, що в даному випадку точність апроксимації заданої функції вийшла не дуже високою – максимальна абсолютна похибка становить 0.055, відносна – 5.5 %, у чому можна переконатися, переглянувши значення помилок (*networkl_errors*) або виходів (*network1_outputs*) мережі.



Зауважимо, що точність апроксимації тут можна було б підвищити, конструюючи мережу з більшим числом нейронів, але при цьому необхідна і більш представницька навчальна вибірка.

Рис. 2.21. Графічне зображення результатів навчання мережі

2.3.3. Робота зі створеної мережею

Для перегляду структурної схеми мережі необхідно, вибравши ім'я мережі (network1), скористатися кнопкою *View*.

При необхідності можна експортувати створену нейронну мережу в робочу область системи MATLAB (натиснувши кнопку *Export* і далі, в відкритому вікні *Export or Save from Neural Network / Data Manager* – кнопки *Select All* (Вибрати все) і *Export*).

Отримати інформацію про ваги і зміщення безпосередньо в робочому вікні системи, виконавши команду:

```
network1.IW{1,1}, network1.b{1}
    ans =    2.6190
             4.9571
    ans =   -2.3744
             2.9881
```

і команду:

```
network1.IW{2,1}, network1.b{2}
```

```
ans = []
ans = 2.1097
```

Тепер можна побудувати модель нейронну мережу в середовищі *Simulink* і відобразити її схему, використовуючи команду:

```
gensim (network1)
```

Ця схема є повною мірою функціональною схемою і може бути застосована для моделювання нейронної мережі. Подвійне клацання на блоці Neural Network розкриває шари мережі, а подвійне клацання на блоці шару мережі розкриває його структуру.

2.3.4. Індивідуальне завдання № 3

Створити, використовуючи графічний інтерфейс користувача, нейронну мережу для виконання операції апроксимації функції з індивідуального завдання № 2.

Створити нейронну мережу, для апроксимації функції з урахуванням впливу шуму. Для цього скористайтеся наведеними нижче командами, які подано як приклад.

```
clear
clc
%-------- Створення і навчання мережі при відсутності шуму ----------
p = [-1:0.05:1]; %вхідний вектор мережі
t = sin (2*pi*p); % вихідний вектор мережі
net = newff(minmax(p), [30,1], {'tansig', 'purelin'}, 'trainbfg'); %створення нейронної мережі
net.trainParam.epochs = 500; % завдання кількості циклів навчання
net.trainParam.show = 50; %кількість циклів для показу проміжних результатів
net.trainParam.goal = 1e-5; %цільова помилка навчання
[net, tr] = train(net, p, t); % навчання мережі при відсутності шуму
%---------- навчання мережі за наявності шуму ----------
netn = net;% ініціалізація мережі
netn.trainParam.epochs = 500; % завдання кількості циклів навчання
netn.trainParam.show = 50; % кількість циклів для показу проміжних результатів
netn.trainParam.goal = 1e-3; % цільова помилка навчання
t1 = [sin(2*pi*p) sin(2*pi*p)]; %вихідний вектор мережі
p1 = [p, (p+randn (size (p))*0.02)]; %вхідний вектор мережі з шумом
[netn, tr]=train(netn, p1, t1); % навчання мережі за наявності шуму
```

```
%-------- Повторне навчання мережі при відсутності шуму ---------
   netn.trainParam.goal=1e-5; % цільова помилка навчання
   [netn, tr] = train (netn, p, t); % повторне навчання мережі при відсутності
шуму
%----- Оцінка ефективності функціонування системи ------
   tic % встановлюємо таймер
   noise = 0:0.01:0.1; % різні рівні шуму
   test = 20; % кількість ітерацій
   network1 = [];
   network2 = [];
   t=sin (2*pi*p); %вихідний вектор
    for noiselevel = noise % організуємо цикл
     errors1 = 0; % початкове значення сумарної помилки мережі net
     errors2 = 0; % початкове значення сумарної помилки мережі netn
     for i=1:test
       P = p + randn (size(p))*noiselevel;  %вхідний вектор
       A = sim(net, P); %використання мережі net
       errors1 = errors1 + sum(sum(abs(A-t)))/2; %сумарна помилка  мережі net
       An = sim(netn, P); % використання мережі netn
       errors2 = errors2+sum(sum(abs(An-t)))/2;% сумарна помилка  мережі
netn
       echo off  % включає повтор команд
      end
      network1 = [network1 errors1/41/20];
      network2 = [network2 errors2/41/20];
    end
    network1
    network2
   toc
   figure(1)
   plot (noise, network1*100, '--b', noise, network2*100, ':r', 'LineWidth', 1.5);
% графіки
% залежності відсотка помилки від рівня шуму
   legend('відсоток помилки мережі net', 'відсоток помилки мережі netn');
%легенда
   xlabel ('Рівень шума','FontSize',12);
   ylabel ('Відсоток помилки','FontSize',12);
   title('Залежність відсотка помилки від рівня вхідного шуму', 'FontSize',
12, 'FontWeight', 'bold');
   grid on
%---------- Використання нейронної мережі ----------
   p2=randn (size (p))*0.03 + p; % вхідний вектор з шумом
   [an,E]=sim(net, p2); % використання нейронної мережі
```

```
figure(2);
plot (p2, t, '+', p2, an, '-', p, t, ':', 'LineWidth', 1.5) % графіки
legend('вход', 'выход', 'sin (2*\pi*t)'); %легенда
title ('Апроксимація функції y=sin(2*\pi*t) двохшаровою нейронною мережею', 'FontSize', 11.5, 'FontWeight', 'bold'); %заголовок
grid on %координатна сітка
E=mse (an-t) % середньоквадратична помилка використання нейронної мережі
gensim (net) % структурна схема нейронної мережі
```

Результати роботи програми представлено на рис. 2.22-2.23.

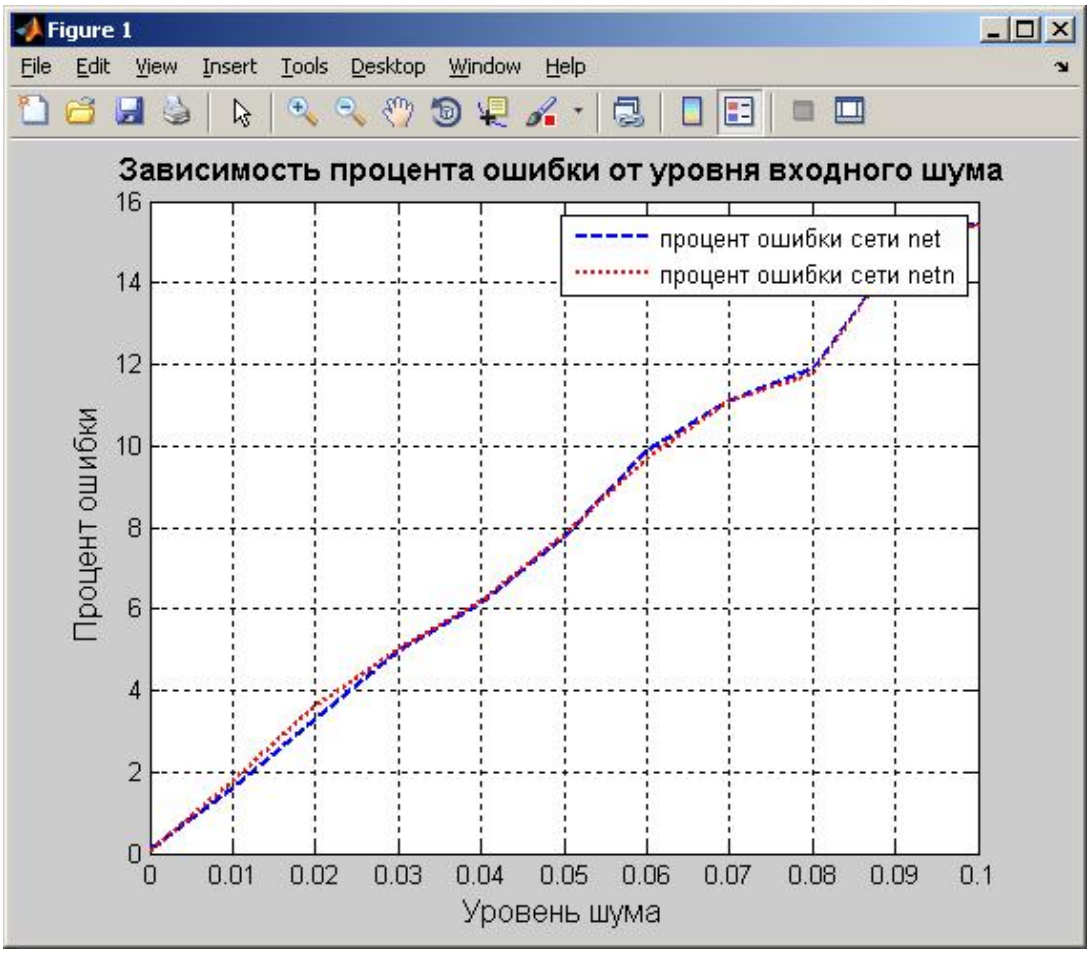


Рис. 2.22. Графічне представлення залежності помилки апроксимації від рівня вхідного шуму

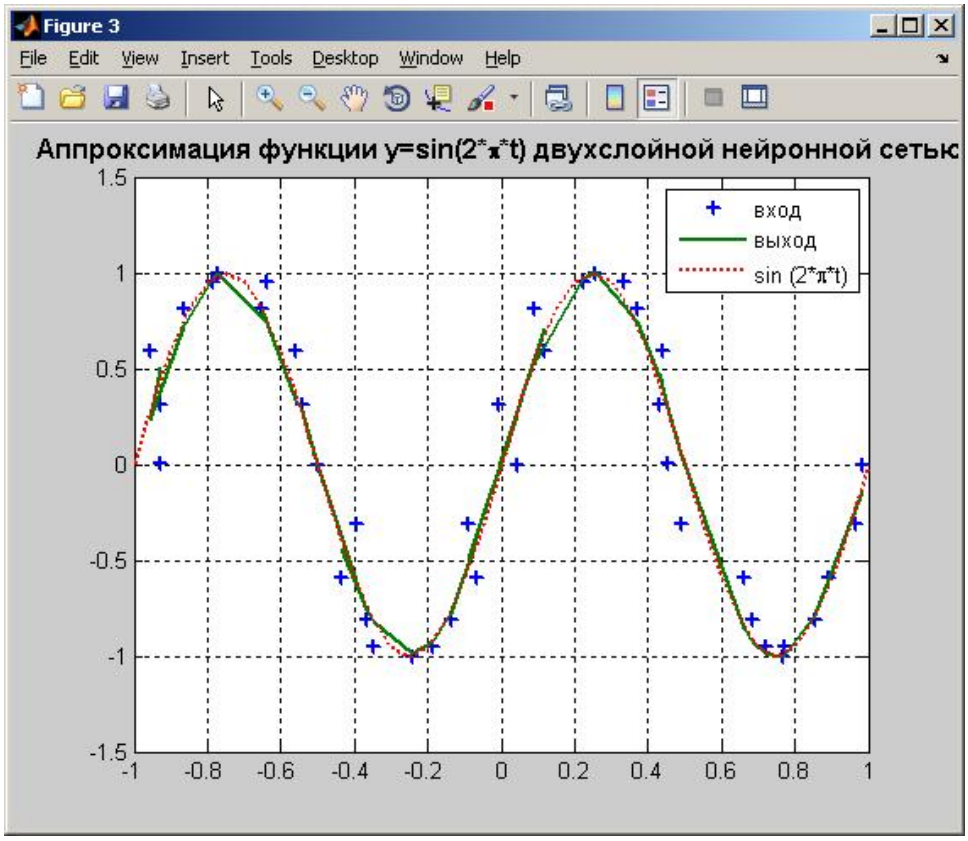


Рис. 2.23. Приклад апроксимації функції $y = \sin(2xt)$ двошаровим перцептроном

Середньоквадратична помилка використання нейронної мережі: E= 0.0158.

Набір команд для апроксимації функції поліномами з урахуванням впливу шуму

```
x = [-1:0.05:1]; %вхідний вектор
y = sin (2*pi*x); % вихідний вектор
p = polyfit (x, y, 14); %знаходження коефіцієнтів полінома
f = polyval (p, x); % обчислення значень полінома
xt = x + 0.03*randn(size (x)); % вхідний вектор з шумом
yp = polyval (p, xt); % обчислення значень полінома за наявності шума
figure (1)
plot(xt, y, '+', xt, yp, '-', x, y, ':', 'LineWidth', 1.5);
legend('вход', 'выход', 'sin(2*\pi*t)');
title('Апроксимація функції y=sin(2*\pi*t) поліномами', 'FontSize', 12);
grid on
err = sqrt (sum ((y - yp).^ 2)/40) % середньоквадратична помилка апроксимації поліномом
```

2.3.5.Індивідуальне завдання №4

Мета роботи: Побудова нейронної мережі для апроксимації обраної функції, використовуючи GUI-інтерфейс пакета нейронних мереж *Neural Networks Toolbox*.

Студенту пропонується створити нейронну мережу і виконати апроксимацію функції (табл. 2.4) з урахуванням впливу шуму, розподіленого за нормальним законом. Вибір варіанта завдання здійснюється за номером по списку групи.

Потрібно:

1. Вибрати архітектуру нейронної мережі (число шарів, число нейронів у шарах, активаційні характеристики в шарах) для апроксимації функції.
2. Виконати навчання мережі за алгоритмом зворотного розповсюдження помилки і порівняти будь-які два методу оптимізації.
3. Провести тестування навченої нейронної мережі. Результати вивести в графічній формі.
4. Побудувати графік залежності середньоквадратичної помилки апроксимації функції нейронною мережею від кількості нейронів у прихованому шарі.
5. Виконати апроксимацію функції створеної нейронної мережею і поліномами за методом найменших квадратів. Побудувати графіки функції, що апроксимується, її наближених значень (сума значень функції, що апроксимується і випадкових величин, розподілених за нормальним законом) і результату апроксимації.
6. Порівняти ефективність апроксимації функції нейронною мережею і поліномами за методом найменших квадратів.
7. Написати програму на *M* – мові пакета Matlab.
8. Звіт по роботі представляється в письмовому вигляді.

У письмовому звіті повинно міститися:

1. Постановка завдання і вихідні дані.
2. Графіки, що відображають навчання мережі та тестування.
3. Графіки, що відображають результати роботи мережі і числові значення середньоквадратичної похибки.
4. Дослідження впливу кількості нейронів на ефективність роботи мережі.
5. Привести приклад обчислення функції в окремих точках і відповідні значення, отримані за допомогою нейронної мережі.
6. Висловити думку про пакет Neural Networks Toolbox. Переваги, недоліки, зручність роботи, зручність представлення результатів.

Таблиця 2.4 – Набір функцій для апроксимації

№ за списком групи	Функція
1	$y = 2\sin^2 x,\ \ x \in [0;1]$
2	$y = 4\dfrac{\cos x}{x},\ \ x \in [-\pi;\pi]$
3	$y_1 = 0{,}5x^2 - 4{,}8x + 3{,}5,\ \ y_2 = x^3 - 12,$ $y_3 = x + 3{,}5,\ \ x \in [0;1]$
4	$y_1 = \dfrac{x_1 - x_2}{x_1 + x_2},\ \ y_2 = \dfrac{1}{y_1},\ \ x_1, x_2 \in [1;10]$
5	$y_1 = \lvert x_1 - x_2 \rvert,\ \ y_2 = x_1 + x_2,\ \ y_3 = x_1 \cdot x_2,$ $x_1, x_2 \in [1;10]$
6	$y_1 = x_1 \cdot \sin x_2,\quad x_1 \in [1;10],$ $x_2 \in [-90^\circ;90^\circ]$
7	$y_1 = x_1 \cdot x_2 + x_3,\ \ y_2 = 2y_1,\ \ x_1, x_2, x_3 \in [1;10]$
8	$y_1 = 1{,}5 \cdot x_1 + \lvert x_2 - 2x_3 \rvert,\ \ y_2 = x_3 - y_1,$ $x_1, x_2, x_3 \in [1;10]$
9	$y = \sqrt{x_1^2 + x_2^2},\quad x_1, x_2 \in [-10;10]$
10	$y_1 = 2{,}3 \cdot x_1 \cdot x_2 - 0{,}5 \cdot x_1^2 + 1{,}8 \cdot x_2^2,\ \ y_2 = y_1^2,$ $x_1, x_2 \in [1;10]$
11	$y = 2 \cdot x_1 + 5 \cdot x_1 \cdot x_2 + x_2,\ \ x_1, x_2 \in [-5;5]$
12	$y = \sin x_1 \cdot \sin x_2 \sin x_3,\ \ x_1, x_2, x_3 \in [1;\pi]$
13	$y = 2 \cdot x_1 \cdot \cos x_2,\ \ x_1 \in [0;1],\ \ x_2 \in [0;\pi]$
14	$y = x_1 + x_2 + x_3,\ \ x_1, x_2, x_3 \in [1;10]$
15	$y = 3\sin^2 x,\ \ x \in [0;1]$
16	$y = 7\dfrac{\cos x}{x^2},\ \ x \in [-\pi;\pi]$
17	$y = x_1 \cdot \cos x_2,\ \ x_1 \in [1;10],$ $x_2 \in [-90^\circ;90^\circ]$
18	$y_1 = \dfrac{x_1 - x_2}{x_1 + x_2},\ \ y_2 = \dfrac{1}{y_1},\ \ x_1, x_2 \in [1;10]$
19	$y = x^3 - 12,\ \ x \in [0;1]$
20	$y = \sqrt{x_1^2 + x_2^2},\ \ y_2 = arctg\dfrac{x_2}{x_1},\ \ x_1, x_2 \in [-10;10]$
21	$y = 4\dfrac{\cos x}{3x},\ \ x \in [-\pi;\pi]$
22	$y = 0{,}7x_1 \cdot \sin 2x_2,\quad x_1 \in [1;10],$ $x_2 \in [-90^\circ;90^\circ]$
23	$y = x_1 + x_2 - x_3,\ \ x_1, x_2, x_3 \in [1;10]$

№ за списком групи	Функція
24	$y = 5\sin^3 x, \quad x \in [0;1]$
25	$y = -3x^3 + 10, \quad x \in [0;5]$
26	$y = 0{,}3x^2 - 5{,}2x + 3{,}5,$ $x \in [0;1]$
27	$y = -2\sin 3x, \quad x \in [0;1]$
28	$y = 3x_1 - x_2 - x_3, \quad x_1, x_2, x_3 \in [1;10]$
29	$y = 2x + 1{,}5, x \in [0;1]$
30	$y = x^3 - \frac{1}{2x}, \quad x \in [1;2]$

2.4. Прогнозування з використанням нейронної мережі. Тестування нейронної мережі. Підбір параметрів

2.4.1. Початок роботи з прогнозування

У командному рядку *Matlab* набираємо команду:

>> nnstart;

Команда *nnstart* відкриває вікно (рис. 2.24), що містить меню пускових кнопок для налаштування нейронної мережі для розпізнавання образів, класифікації, кластеризації та прогнозування динамічних часових рядів. Він також містить посилання на списки наборів даних, приклади та іншу корисну інформацію для початку роботи.

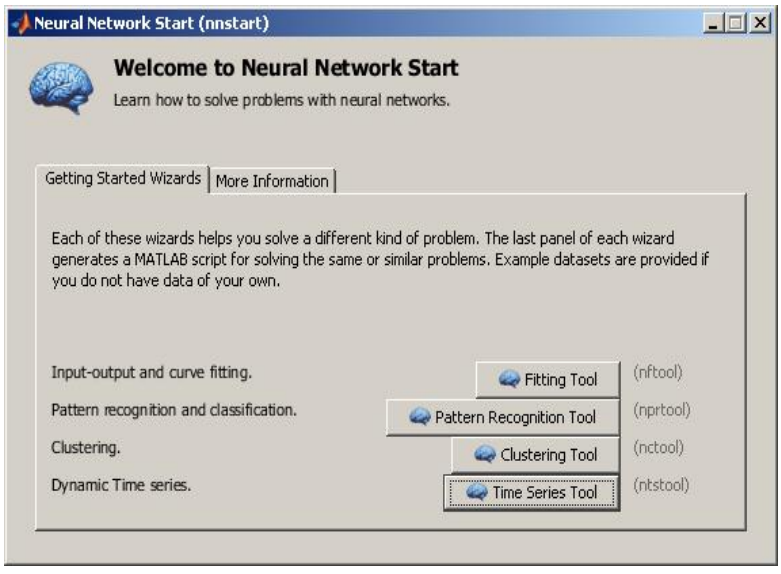


Рис.2.24. Меню для початку роботи з нейронними мережами.

Кожен з наведених нижче майстрів допоможе вирішити різні види проблеми. Остання панель кожного майстра генерує сценарій *Matlab* для вирішення такої ж або аналогічної проблеми. Приклади вхідних даних надаються, якщо у вас немає даних самостійно.

У вікні (рис.2.24) натискаємо кнопку «*Time Series Tool*».

Прогнозування є свого роду динамічною фільтрацією, в якому останні значення одного або декількох часових рядів використовуються для прогнозування наступних значень. Динамічні нейронні мережі, які включають в себе лінії затримки використовуються для нелінійної фільтрації та прогнозування.

Прогнозні моделі також використовуються для ідентифікації системи (або динамічного моделювання), в якому ви будуєте динамічні моделі фізичних систем. Ці динамічні моделі важливі для аналізу, моделювання, моніторингу та контролю різних систем, у тому числі виробничих систем, хімічних процесів, в робототехніці і аерокосмічних системах.

Цей інструмент дозволяє вирішити три види нелінійних задач часових рядів, показаних. Натиснувши кнопку «*Time Series Tool*» перейдемо на вкладку для прогнозування часових рядів з використанням нейронних сіток, на якій в правій панелі необхідно вибрати модель за якою буде здійснюватися прогнозування (рис.2.25):

1. Нелінійна авто регресійна з зовнішнім входом: прогнозування значень *y(t)* за даними *d* попередніх значень *y(t)* та значень іншого ряду *x(t)*.

$$y(t)=f(x(t-1),\ldots,x(t-d), y(t-1),\ldots, y(t-d))$$

2. Нелінійна авто регресія: прогнозування значень *y(t)* за даними *d* попередніх значень *y(t)*.

$$y(t)=f(y(t-1),\ldots, y(t-d))$$

3. Нелінійна модель Вхід–Вихід: прогнозування значень *y(t)* за даними *d* попередніх значень *x(t)*.

$$y(t)=f(x(t-1),\ldots,x(t-d))$$

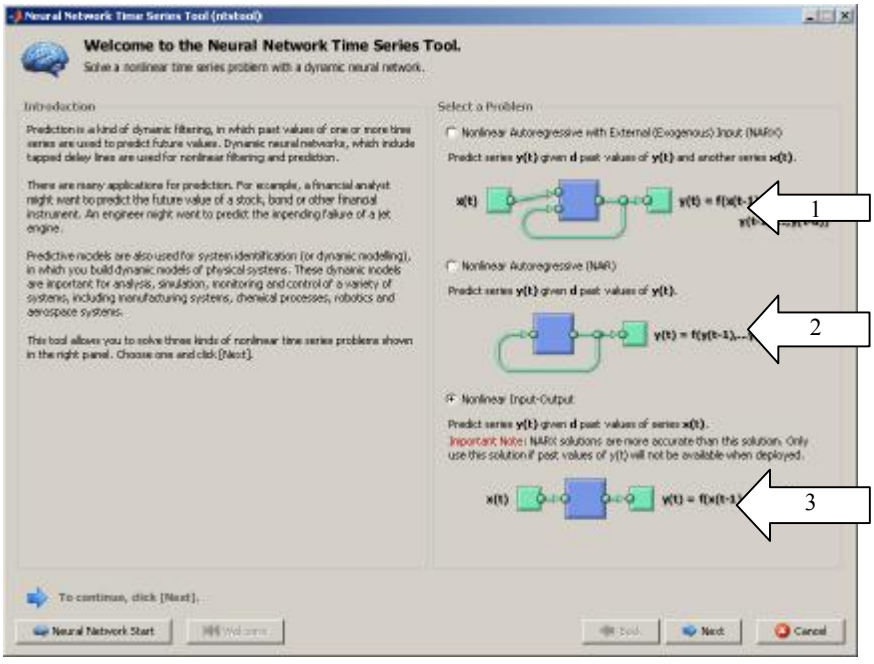


Рис.2.25. Вибір моделі для прогнозування

Оберіть одну з моделей. У наступному вікні, після натискання «*Next*» пропонується вибрати дані для навчання мережі. Можна вибрати як з робочого простору *Matlab* (рис.2.26) так і довантажити зовнішні дані (рис.2.26) «1» – «Вхідні дані»,«2» – «Вектор цільових даних».

Для навчання сітки необхідно завантажити вхідні дані «*Inputs*» та вектор цільових даних «*Targets*». Якщо дані завантажено коректно, то стає активною кнопка «*Next*» – натискаємо. З'являється вкладка «***Validation and test data***» – «Перевірка та випробування даних».

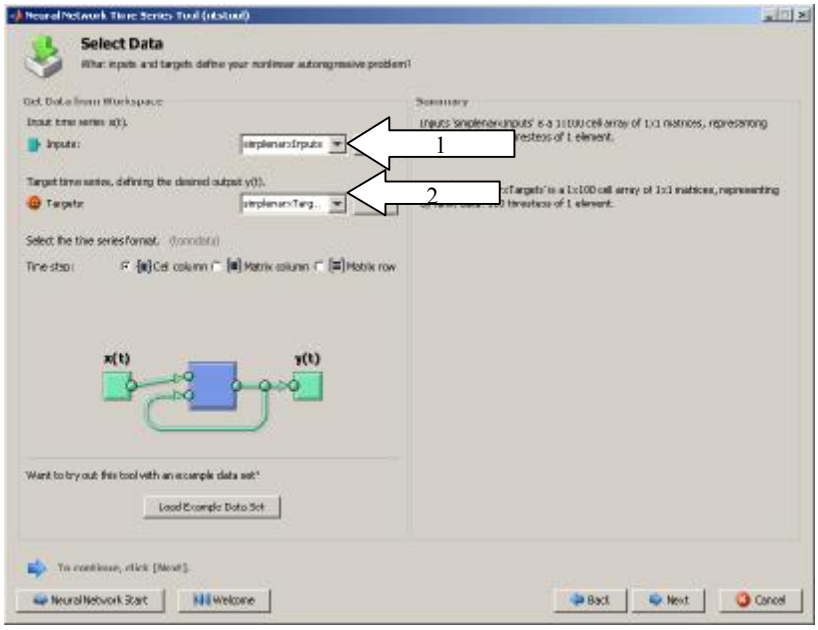


Рис. 2.26. Завантаження вхідних даних

Вектор цільових даних розбивається на три групи:

Training (Тренування) – дані будуть представлені в мережу під час навчання, а мережа налаштовується відповідно до мінімізації помилки.

Validation (Перевірка) – ця частина даних використовується для вимірювання якості роботи мережі та припинення навчання, коли якість зупиняє поліпшуватися.

Testing (Тестування) – дані не мають впливу на навчання, так як застосовуються для вимірювання продуктивності мережі під час і після навчання.

При таких установках, вхідні вектори та цільові вектори даних будуть випадковим чином розділені на три групи:

- 70% буде використовуватися для навчання.
- 15% буде використовуватися для перевірки, що навчання мережі закінчено.
- останні 15% будуть використані для незалежного тестування мережі.

Представлений розподіл відсотків даних рекомендується залишити за замовчуванням, якщо немає інших міркувань. Натискаємо кнопку «*Next*».

З'являється вкладка «***Network Architecture»*** – «Архітектура нейронної сітки» за допомогою якої встановлюється архітектура нейронної сітки (рис. 2.27), в якому ми повинні вказати:

Number of Hidden Neurons (Кількість прихованих нейронів) – вкажіть кількість прихованих нейронів.

Number of Delays d (Кількість затримок *d*) – вкажіть кількість затримок.

Також, необхідно повернутися до цієї панелі і змінити кількість нейронів або затримки, якщо якість прогнозування незадовільна після навчання.

Мережа буде створена і навчена у формі розімкнутого контуру, як показано нижче. Без зворотного зв'язку (покрокове навчання) більш ефективно, ніж у замкнутому контурі (багатоступінчасте) навчання. Натискаємо кнопку «*Next*».

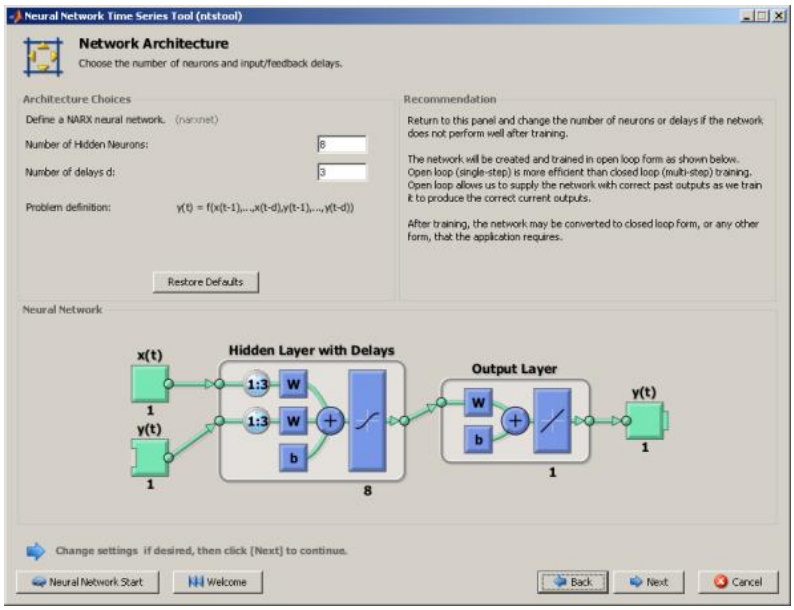
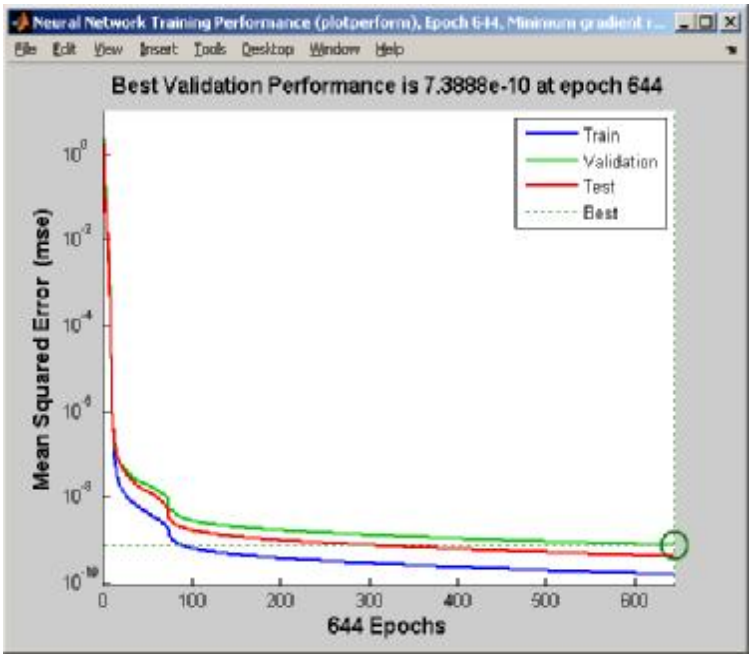


Рис.2.27 Створення архітектури нейронної сітки

Деякий час сітка створюється. Переходимо до вкладки «***Train Network***» – «Навчання сітки» – під тренуванням тут розуміють встановлення відповідності вхідним даним цільових. Наведено параметри навчання НС – навчання відбувається з використанням алгоритму Левенбрга-Маргуардта з зворотнім поширенням помилки. Навчання автоматично зупиняється, коли зміна ваг (процес узагальнення) перестає поліпшуватися, коли значення середньоквадратичної помилки стає не більше ніж деяке наперед задане значення для зразків перевірки.

Натискаємо кнопку «*Train*».

Після завершення процесу навчання з'являється вікно з повідомленням про його закінчення. Зверху в ньому наведено інформацію про архітектуру нейронної сітки, алгоритм навчання, сам процес навчання: кількість ітерацій, час, продуктивність, градієнт, середньоквадратичну помилку, перевірку відповідності.



В розділі «**Plots**» можна переглянути графіки, представлені нижче (2.28-2.30).

Рис. 2.28. Натиснувши кнопку «*Performance*»

З наведених вище графіків видно, що найкраща продуктивність системи досягається за 644 ітерацій, коли досягається заданий

рівень помилки для даних використовуваних для перевірки (зелена смуга).

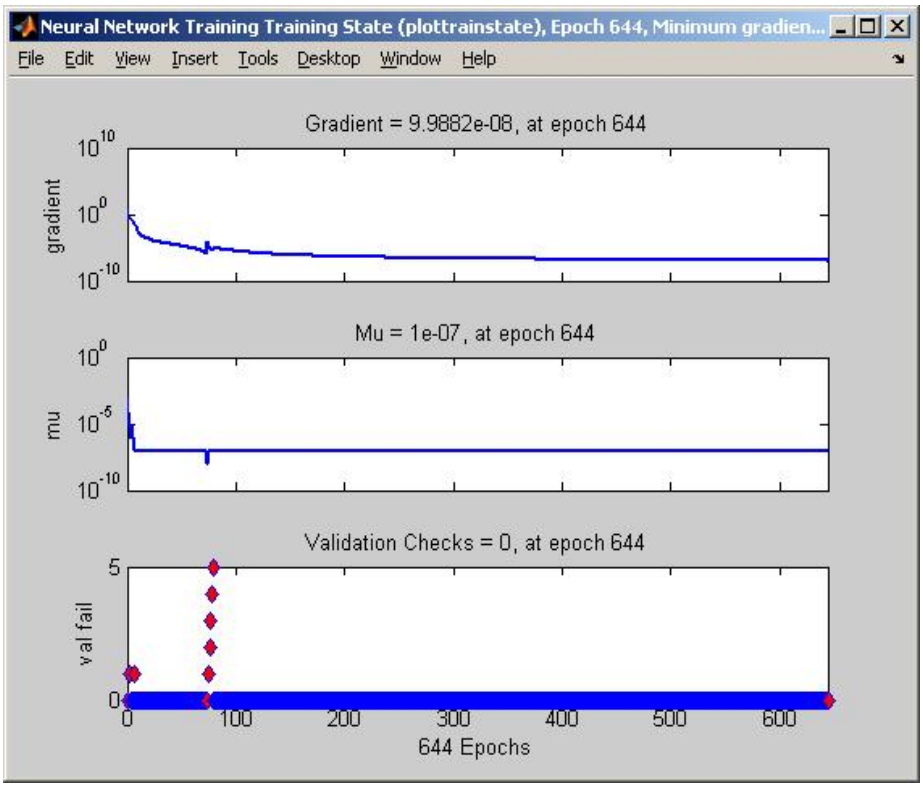


Рис.2.29. Натиснувши кнопку «*Training State*»:

На графіках показано зменшення градієнту, середньо-квадратичної похибки та перевірка відповідності за 644 ітерації.

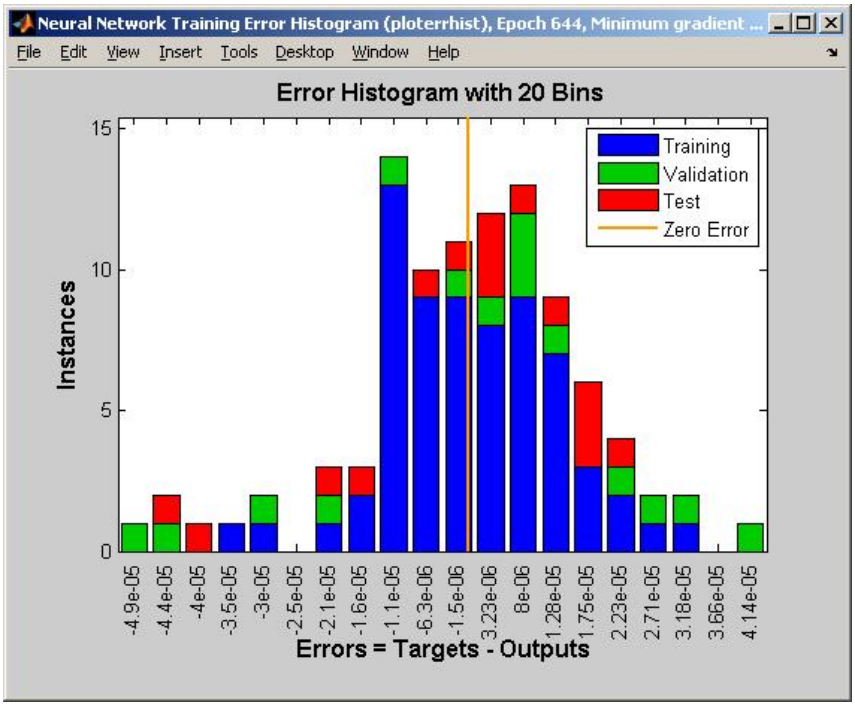


Рис 2.30. Натиснувши кнопку «*Error Histogram*»

На графіку наведено різницю похибок вектора цільових та вхідних значень. Перегляд гістограми помилок дозволяє додатково перевірити продуктивність (якість) мережі.

Сині смуги представляють навчальні дані, зелені смуги представляють дані для перевірки, і червоні прямокутники представляють тестові дані. Гістограма може дати вам уявлення про викиди – аномальні значення вхідних послідовностей. Якщо викиди є допустимими точками даних, але істотно відмінні від інших даних, то мережа має екстраполювати ці точки. Ви повинні зібрати більше даних, що містять подібні точки до точок викиду, і перенавчити мережу.

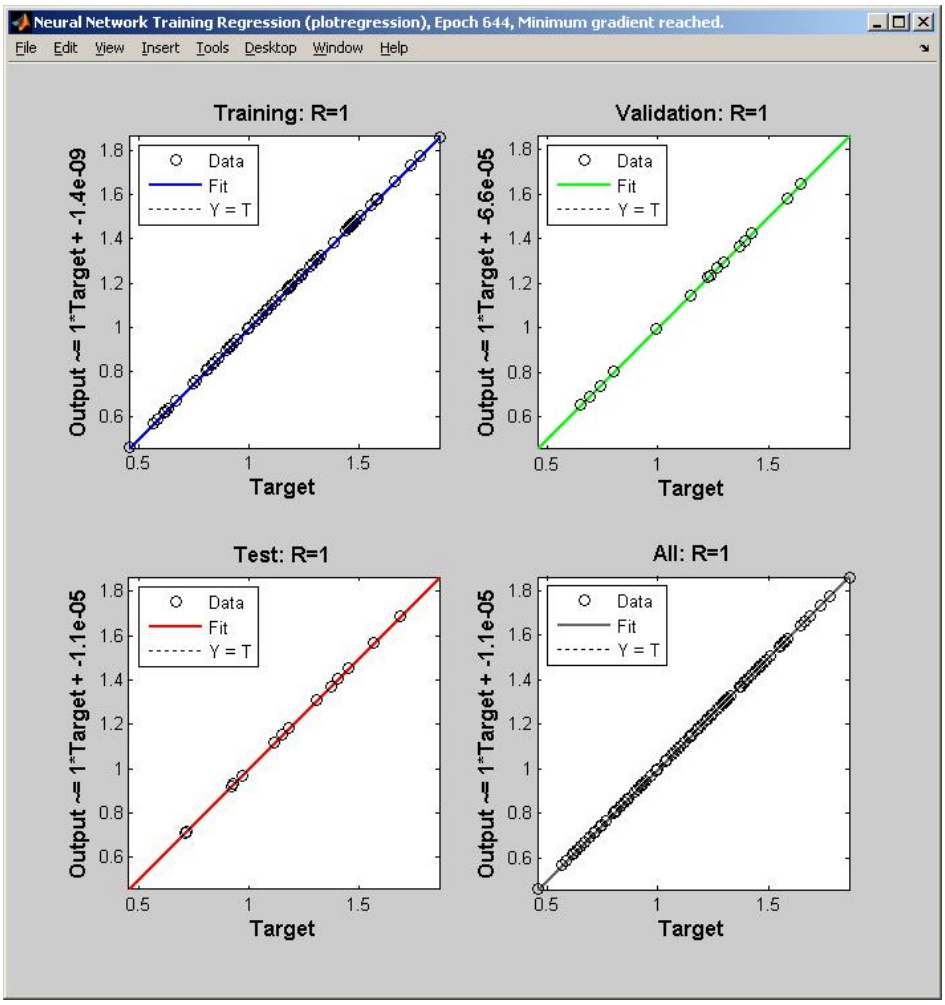


Рис. 2.31. Натиснувши кнопку «*Regression*»

Представлені графіки регресії відображають відношення мережевих виходів до цільових значень для навчаючої, перевірочної та тестової послідовностей. Для ідеального підбору, дані повинні бути розташовані вздовж лінії в 45 градусів, де виходи мережі дорівнюють цілям. Це завдання, реалізовано досить добре для всіх наборів даних, значення *R* в кожному випадку дорівнює 1. Якщо потрібні ще більш точні результати, можна перенавчити мережу, натиснувши «*Retrain*», або команду *nftool*. Це змінить початкові ваги зміщень мережі, і може привести до поліпшеної мережі після перепідготовки. Інші варіанти наведені на наступній панелі.

Натискаємо кнопку «*Next*». Переходимо до вкладки «***Evaluate Network***» – «Оцінювання якості сітки». На цій вкладці проводиться тестування нейронної сітки на додаткових даних (завантажуються в поля «*Inputs*» та «*Targets*») і вирішується чи задовільна якість отриманих результатів. Поліпшити якість результатів, в даному випадку, прогнозування можна:

- Навчання знову, кнопка «*Train Again*»;
- Коректування структури НС, кнопка «Adjust Network Size»;
- Завантаження більшого набору даних кнопка «*Import Larger Data Set*».

Якщо продуктивність (якість) прогнозування на навчальній вибірці задовільна, але на випробувальних даних продуктивність (якість) значно гірша, що може свідчити про перенавчання, то необхідно зменшення кількості нейронів може поліпшити результати. Якщо продуктивність (якість) навчання погана, то ви можете збільшити кількість нейронів.

Коли досягли прийнятної якості прогнозування, натикаємо «*Next*», переходимо на вкладку «***Deploy Solution***» – «*Розгортання Рішення*».

Використовуйте цю панель для генерації розгорнутої версії для подальшого моделювання вашої нейронної мережі. Ви можете використовувати згенерований код або діаграму, щоб краще зрозуміти, як побудована нейронна мережа обчислює виходи з входів, коректування та подальшого використання нейронної мережі.

На наступній вкладці «***Save Results***» – «*Збереження Результатів*» використовуйте кнопки на цьому екрані для створення сценаріїв або для збереження результатів.

2.4.2. Приклад прогнозування курсу валют

Маємо середньорічні щомісячні дані курсу гривні по відношенню інших до валют з 1996 по 2013 роки (табл. 2.5). Необхідно зробити прогноз курсу гривні на 2014 рік.

Таблиця 2.5 – Крос-курси валют

Рік / Назва валюти	1996	1997	1998	1999	2000	2001	2002	2003	2004	2005	2006	2007	2008	2009	2010	2011	2012	2013
100 англ. фунт стерл.	285,64	305,08	406,89	668,45	824,99	773,94	799,84	871,28	973,91	933,76	929,45	1010,7	966,88	1219,5	1226,9	1277,7	1266,4	1269,4
100 доларів США	182,95	186,17	244,95	413,04	544,02	537,21	532,66	533,27	531,92	512,47	505,00	505,00	526,72	779,12	793,56	796,76	799,10	799,12
100 швейц. франків	148,25	128,45	171,01	274,53	322,46	318,71	342,88	396,14	428,18	412,75	402,78	421,16	486,09	719,50	762,61	901,41	852,08	856,05
1000 яп. єн	16,84	15,41	19,03	36,61	50,52	44,27	42,59	46,13	49,21	46,68	43,40	42,92	51,40	83,45	90,52	100,04	100,19	100,17

Вводимо вхідні дані:

Код Matlab *M*–файлу:

```
GBP=[285.64 305.08  406.89  668.45  824.99  773.94  799.84  871.28  973.91
933.76  929.45  1010.69 966.88  1219.48 1226.89 1277.71 1266.38  1269.4];
USD=[182.95 186.17  244.95  413.04  544.02  537.21  532.66  533.27  531.92
512.47  505 505 526.72  779.12  793.56  793.56  799.1 799.12];
CHF=[148.25 128.45  171.01  274.53  322.46  318.71  342.88  396.14  428.18
412.75  402.78  421.16  486.09  719.5   762.61  901.41  852.08  856.05];
JPY=[16.84  15.41   19.03   36.61   50.52   44.27   42.59   46.13   49.21   46.68
43.4    42.92   51.4    83.45   90.52   100.04  100.19 100.17];
```

Використовуємо метод ковзного середнього для підготовки даних.

Функція реалізує ковзне вікно, де m – розмір вікна. Враховуючи що даних мало, варто взяти m=3, тобто 3 роки в одному вікні. Утворений таким чином набір даних починається з 1997 і закінчується 2012 роком. Тому як вікно 2012–го включає 2011, 2012 і 2013 роки. Тому прогноз прийдеться робити на 2 роки вперед. Щоб отримати прогноз на 2014 рік.

Код Matlab *M*–файлу:

```
function y=SlWind(x,m)
lenx=size(x,2);
imin=1+(m-1)/2;
imax=lenx-(m-1)/2;
iset=imin:imax;
leni=length(iset);
y=zeros(size(x,1),leni);
for ik=1:leni
  ind=(iset(ik)-(m-1)/2):(iset(ik)+(m-1)/2);
  y(:,ik)=sum(x(:,ind),2)/m;
end
end
```

Код Matlab *M*-файлу:

```
function y2014=Curr(y)
targetSeries=tonndata(y,true,false);
feedbackDelays=2:3; % мінімум 2 роки в лінію затримок, так як розробляється прогноз
                    % на 2 роки вперед
hiddenLayerSize=2; % якщо взяти більше нейронів, то параметрів мережі більше ніж даних
net = narnet(feedbackDelays,hiddenLayerSize);
net.inputs{1}.processFcns={'removeconstantrows','mapminmax'};
[inputs,inputStates,layerStates,targets] = preparets(net,{},{},targetSeries);
net.divideFcn='';
net.trainFcn='trainbr';
net.performFcn ='mse';  % средньоквадратична похибка
net.plotFcns={'plotperform','plottrainstate','plotresponse','ploterrcorr','plotinerrcorr'};
net= train(net, inputs, targets, inputStates, layerStates); % тренування
outputs=net(inputs, inputStates, layerStates);
errors=gsubtract(targets, outputs);
performance=perform(net,targets, outputs);
%view(net);
nets=removedelay(net,2);
[xs,xis,ais,ts]=preparets(nets,{},{},targetSeries);
ys=nets(xs,xis,ais);
y2014=ys(end)
view(nets)
end
```

Виклик в Командному вікні:

y= SlWind(GBP,3); %Підготовимо дані для нейронну мережу методом ковзного середнього
y2014=Curr(y) % Потім викличемо функцію для прогнозування на 2014 рік

Результат роботи програми:

y2013 = [1.3588e+03]

Тобто курс гривні у 2014 році по відношенню до 100 англійських фунтів стерлінгів становитиме 1358,8 грн.

2.4.3. Індивідуальне завдання № 5

Мета: Прогнозування з використанням *nnstart* та програмно.

1. Вибрати архітектуру нейронної мережі (число шарів, число нейронів у шарах, активаційні характеристики в шарах) для апроксимації функції.
2. Виконати навчання мережі.
3. Провести тестування навченої нейронної мережі. Результати вивести в графічній формі.
4. Написати програму на *M* – мові пакета Matlab.
5. Звіт по роботі представляється в письмовому вигляді.

У письмовому звіті повинно міститися:

1. Постановка завдання і вихідні дані.
2. Графіки, що відображають навчання мережі та тестування.
3. Графіки, що відображають результати роботи мережі і числові значення середньоквадратичної похибки.
4. Порівняти прогноз, отриманий за допомогою нейронну мережу та трендового аналізу в MS Excell.
5. Ваша думка про пакет Neural Networks Toolbox.

Студенту пропонується створити нейронну мережу і виконати апроксимацію функції (табл. 2.6) з урахуванням впливу шуму, розподіленого за нормальним законом.

Виконати прогнозування на наступний рік курсу гривні по відношенню до інших валют, згідно варіанту (табл. 2.6). Статистичні дані отримати: ***minfin.com.ua***

Таблиця 2.6. – Варіанти індивідуальних завдань

Варіант	**Назва валюти**
1	100 австралійських доларів
2	100 австрійських шилінгів

Варіант	Назва валюти
3	1000 бельгійських франків
4	100 датських крон
5	100 ірландських фунтів
6	100 шведських крон
7	1000 іспанських песет
8	10000 італійських лір
9	100 канадських доларів
10	100 німецьких марок
11	100 норвезьких крон
12	100 польських злотих
13	10 російських рублів
14	100 сінгапурських доларів
15	100 словацьких крон
16	1000 угорських форинтів
17	100 юанів женьміньбі (Китай)
18	100 фінських марок
19	100 французьких франків
20	100 чеських крон
21	10 російських рублів
22	100 французьких франків
23	1000 бельгійських франків
24	100 датських крон
25	100 ірландських фунтів
26	100 шведських крон
27	1000 іспанських песет
28	10000 італійських лір
29	100 канадських доларів
30	100 німецьких марок

2.5. Кластеризація та класифікація з використанням нейронної мережі

Нейронні мережі добре працюють для класифікації даних, сегментування ринку, угрупувань даних. Наприклад, є дані, що описують економічний об’єкт (форма власності, кількість працівників, розмір уставного капіталу, кредитів, інвестицій, видатки, доходи, відрахування і т.д.) або дані про потенційного клієнта, який подав заявку про отримання кредиту (вік, обсяг доходів, стаж, кваліфікація, місце роботи, майно, що знаходиться у власності) необхідно віднести його до певної групи ризику, щодо повернення кредиту, щоб визначити оптимальні для клієнта та банку умови кредитування.

У завданнях кластеризації навчаємо нейронну мережу для групування даних за подібністю. Наведений нижче інструментарій кластеризації з використанням нейронної мережі допоможе вибрати дані, створювати і навчати мережу, і оцінити її продуктивність за допомогою різних засобів візуалізації.

Розпізнавання образів це процес навчання нейронної мережі для розподілу в правильні цільові класи набору вхідних образів. Після навчання мережа може бути використана для класифікації образів на яких не навчалась.

2.5.1. Кластеризація за допомогою інструменту «Clustering Tool»

У командному рядку *Matlab* набираємо команду:

>> nnstart;

В вікні, що з'явиться, (рис. 2.32) обираємо кнопку «**Clustering Tool**» для відкриття інструменту для кластеризації даних (рис. 2.33).

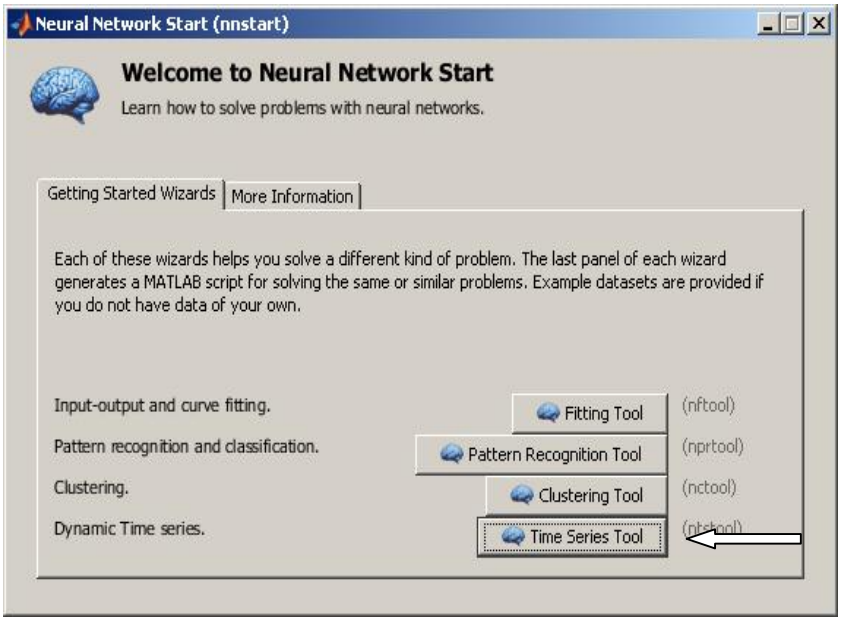


Рис. 2.32. Запуск

Вікно (рис.2.33) буде відкрито і по команді:

>> nprtool;

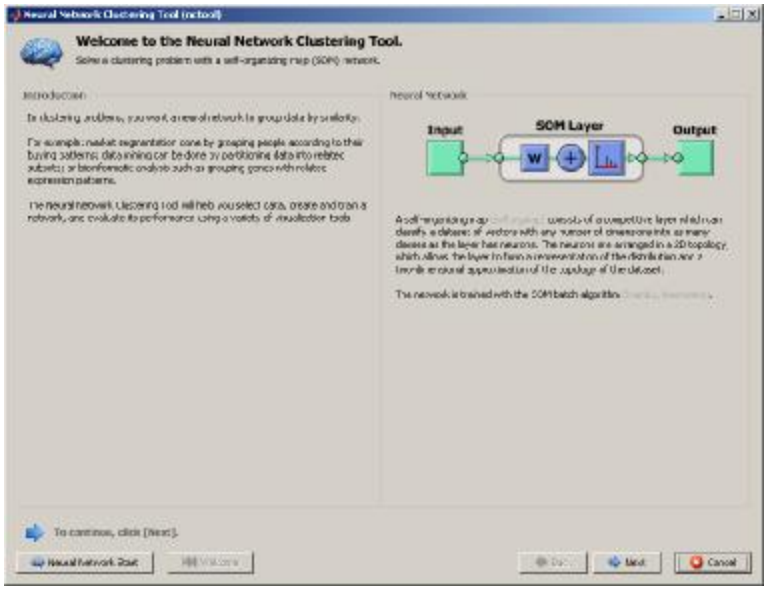


Рис. 2.33. Інструментарій для кластеризації.

Самоорганізована карта (*selforgmap*) складається з шару, який може класифікувати масив даних векторів будь-якої вимірності на стільки класів, скільки нейронів в цьому шарі. Нейрони організовані в 2D топології, що дозволяє

сформувати уявлення про розподіл і двовимірне наближення топології набору даних. Мережа навчається за пакетним алгоритмом *SOM* (*trainbu, learnsomb*).

Натисніть кнопку «*Next*» для відкриття вкладки «**Select Data**», далі кнопку «*Load Example Data Set*», відкриється вікно для обрання образів для кластеризації. Оберіть дані рис. 3, натиснувши кнопку «*Import*», для повернення в вікно «**Select Data»**.

Рис. 2.34 Вибір вхідних даних для кластеризації.

Обираємо «*Simple Clusters*» файл містить *simpleclusterInputs* – матрицю з 1000 двоелементних векторів (2x1000).

Фрагмент даних:

1	2	3	4	5	6	7	8	9	10
-0,35	0,823	1,003	0,110	1,278	0,845	0,903	0,423	0,924	-0,105
0,339	0,559	0,146	-0,074	0,615	1,064	0,636	0,504	0,951	-0,198

Натисніть кнопку «*Next*» для переходу на вкладку «**Network Architecture**» рис. 2.35. Необхідно вказати кількість нейронів у *SOM*-шарі, тобто кількість рядків та стовпців встановлено 10. Загальна кількість нейронів 100, за необхідності можна змінити.

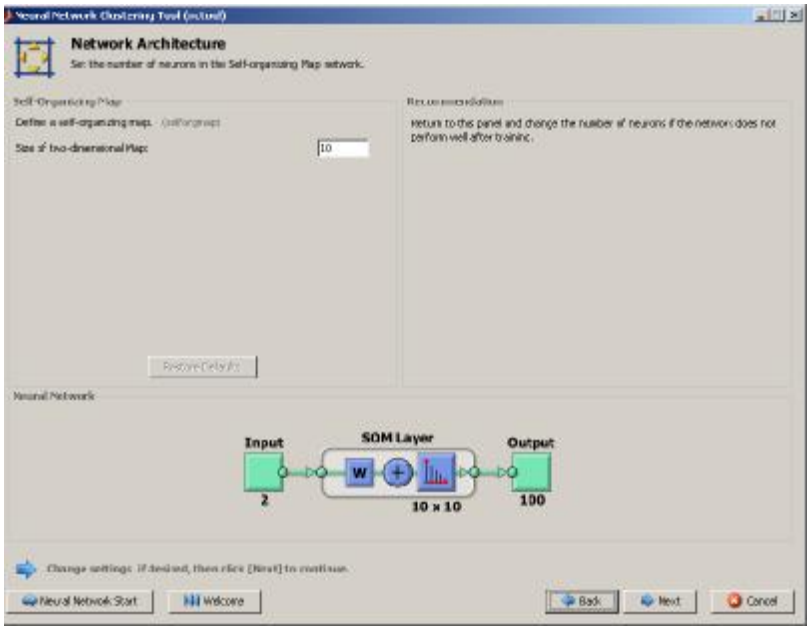


Рис.2.35. Вкладка «**Network Architecture**».

Натисніть «*Next*» для переходу на вкладку «**Train Network**», натискаємо кнопку «*Train*», відбувається процес навчання нейронну мережу (рис. 2,36).

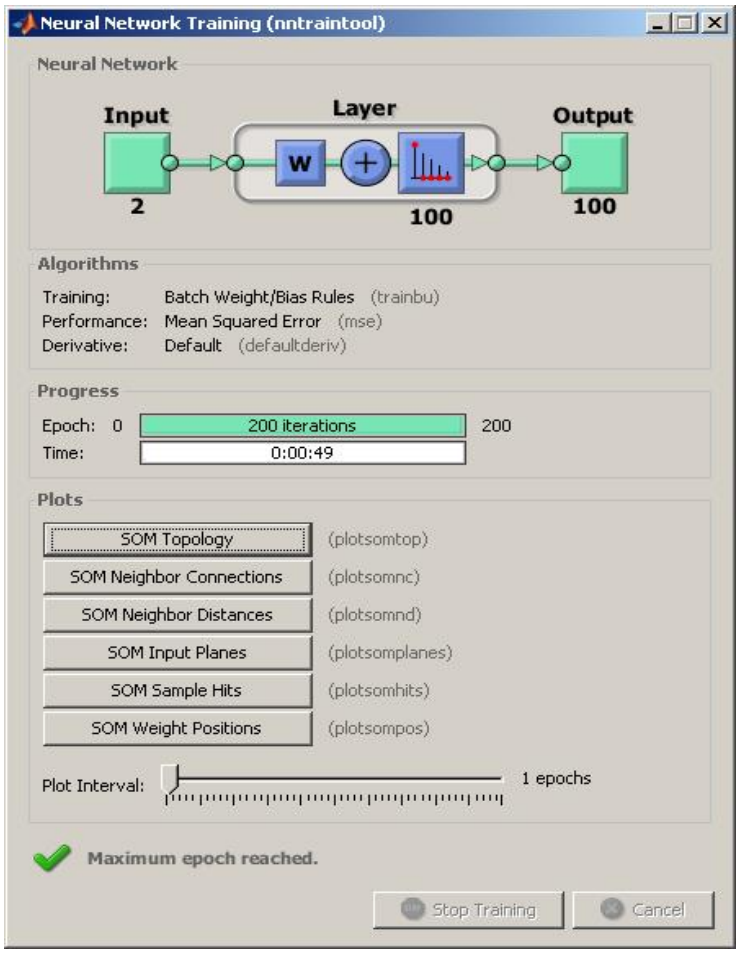


Рис. 2.36. Навчання нейронну мережу за SOM алгоритмом

Навчання триває протягом максимальної кількості епох, що дорівнює 200.

При SOM навчанні, ваговий вектор, пов'язаний з кожним нейроном рухається, щоб стати центром кластера вхідних векторів. Крім того, нейрони, які є суміжними один з одним в топології повинні також переміщатися близько один до одного у вхідному просторі, тому можна візуалізувати входи високої розмірності простір у двох вимірах топології мережі. Для візуалізації результатів кластеризації використовують інструменти на панелі «Plots», натисніть «SOM Sample Hits» та «SOM Weight Positions» рис. 2.37.

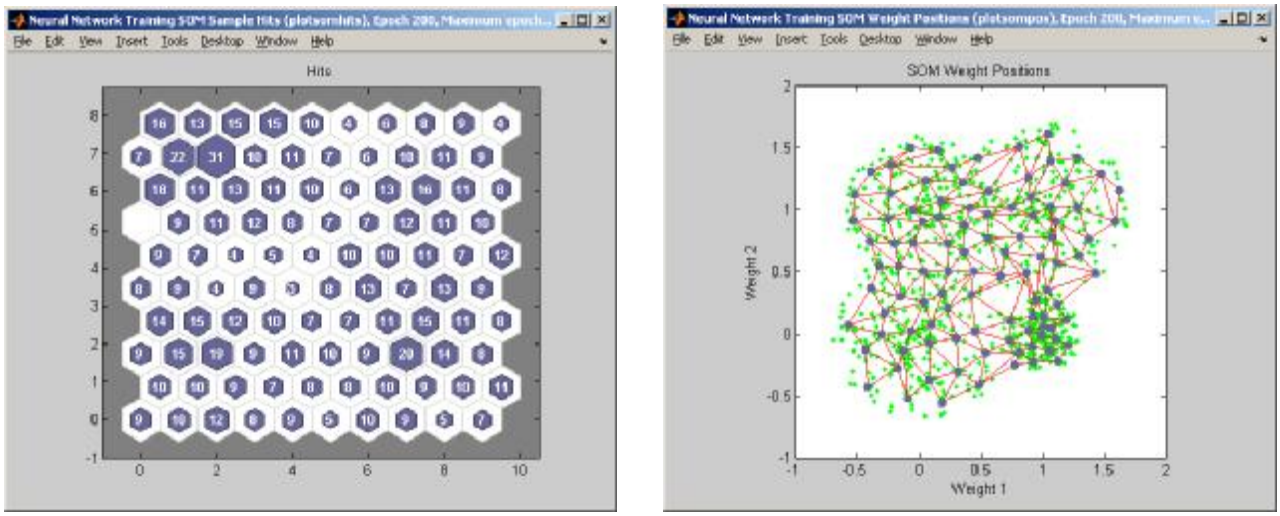


Рис. 2.37. Результати кластеризації

За замовчуванням SOM топологія є шестикутником. На рис. 2.37 показано розташування нейронів в топології, і показує, скільки навчальних даних

пов'язано з кожним з нейронів (центрами кластерів). Топологія 10 на 10, містить 100 нейронів. Максимальна кількість елементів, що потрапили до одного кластеру 31, тобто, є 31 вхідний вектор в цьому кластері.

В «**Neural Network Clustering Tool**» для подальшої роботи з мережею, або для кластеризації нових даних, натисніть «*Next*». Якщо ви незадоволені якістю кластеризації цією мережею вихідних або нових даних, ви можете збільшити кількість нейронів, або надати більший набір навчальних даних. При отримання задовільної якості експортуйте мережу у «*Workspace*» і збережіть результати.

2.5.2. Кластеризація за допомогою самоорганізованої мережі Кохонена

Цей спосіб навчання мережі використовується, якщо не відомі значення цільового вектора (значення *T*). Як буде видно з цього прикладу, система сама автоматично визначить і рознесе центри кластерів. Також є параметри для тонкої настройки мережі, їх можна подивитися при навчанні мережі, такі як кількість епох, градієнт, похибка і т.д. До них потрібно звертатися за наступною формою: ***і'мя_мережі.команда***, наприклад *net.time*.

Нехай маємо кількість нейронів – 4. Позначимо базу знань – *z*. Встановимо розмірність бази знань – 2x120. Тепер задамо множину значень кластеру № 1.

Центром кластера буде є точка (3, 0) з невеликим розкидом від центру. Елементи множини створимо за допомогою датчика випадкових чисел, розподілених за рівномірним законом з кількістю елементів 30. Відповідно отримаємо x_1 та y_1 – масиви розмірністю 1x30. Ці операції відповідають наступні команді

```
y1=0+rand(1,30)
x1=3+rand(1,30)
```

Згенеровані значення виведемо на графік командою

```
plot(x1,y1,'ob')
```

На рис. 2. 38 представлено приклад роботи цієї команди

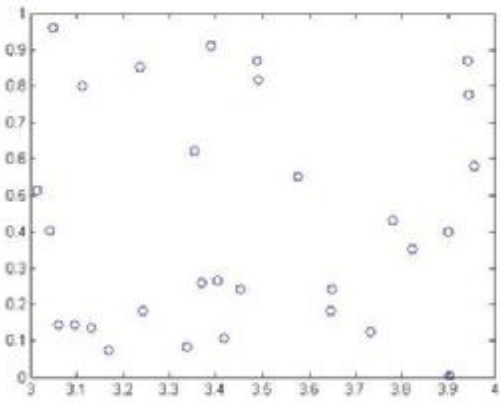


Рис. 2.38. Приклад роботи команди PLOT

Аналогічним чином задамо множину значень другого кластеру з центром (-3;0) та переглянемо отримані значення

```
y2=0+rand(1,30)
x2=-3+rand(1,30)
plot(x2,y2,'or') %
```

Далі задамо множину значень третього и четвертого кластерів з центрами (0;3) и (0;-3)

```
y3=3+rand(1,30)
x3=0+rand(1,30)
x4=0+rand(1,30)
y4=-2+rand(1,30)
```

Для наочності виведемо всі вхідні значення на одному графіку командою ***figure(1)*** (рис.2. 39)

```
figure (1)
hold on
plot(x3,y3,'og')
plot(x4,y4,'oy')
plot(x1,y1,'ob')
plot(x2,y2,'or')
grid on
hold off
```

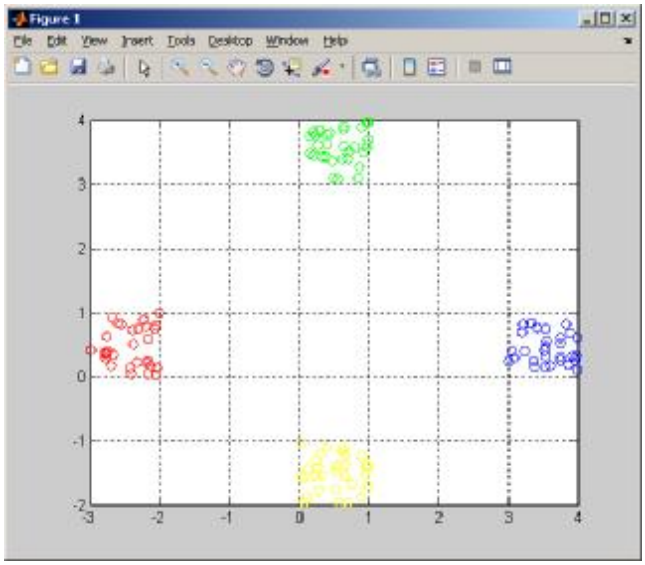


Рис. 2.39. Вхідні дані.

Перед тим як подавати згененровані дані на вхід нейромережі, необхідно з'єднати всі вхідні дані в одну матрицю для цього з'єднаємо x_1, x_2, x_3, x_4 в масив x та y_1, y_2, y_3, y_4 в масив y. Далі масиви x та y присвоїмо в змінну z.

```
x(1:30)=x1;
x(31:60)=x2;
x(61:90)=x3;
x(91:120)=x4;
y(1:30)=y1;
y(31:60)=y2;
y(61:90)=y3;
y(91:120)=y4;
z(1,1:120)=x
z(2,1:120)=y
```

Задамо сітку карти Кохонена як гексогональну 2х2

```
net = newsom(z,[2 2]);
```

Проведемо цикл навчання, рис. 2.40

```
net = train(net, z)
```

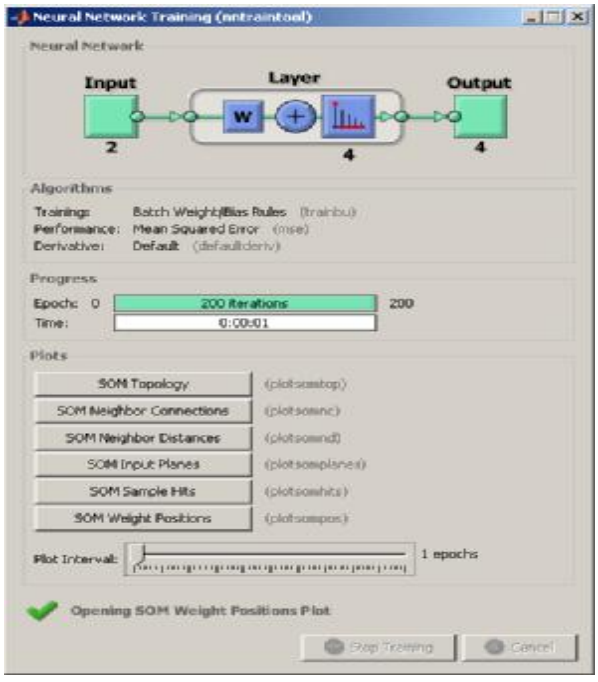


Рис. 2.40. Навчання нейронної мережі

На рис. 2.41 показано вид карти Кохонена, який було задано в нашому прикладі

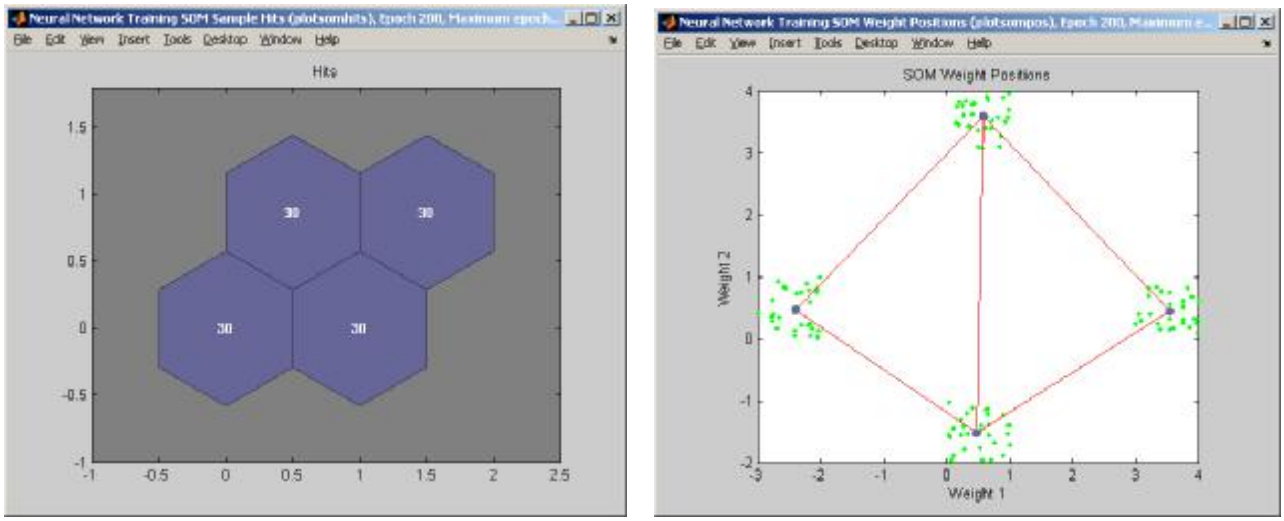


Рис.2.41. Розподіл по кластерам вхідних даних, котрі були використані при навчанні.

Перевіримо отриману мережу на даних на яких навчали мережу.

```
a = sim(net,z)
```

Аналогічно можна перевірити на інших даних.

2.5.3. Приклад кластеризації даних з використанням нейронної мережі прямого поширення

В навчальних цілях задамо чотири множини по 50 елементів з центрами (2;0), (-2;0), (0;2), (0;-2).

Для створення мереж з прямою передачею сигналу в ППП NNT призначена функція *newff*. Вона має 4 вхідних аргументи і 1 вихідний аргумент – об'єкт класу *network*. Перший вхідний аргумент – це масив розміру $R\times2$, що містить допустимі межі значень (мінімальне і максимальне) для кожного з R елементів вектора входу; другий – масив для завдання кількості нейронів кожного шару; третій – масив комірок, що містить імена функцій активації для кожного шару; четвертий – ім'я функції навчання

Спочатку будемо діяти аналогічно п.2.5.2, створюючи масиви задопомогою датчика випадкорвих чисел, розподілених рівномірно.

Задамо множину значень кластера № 1. Центром цього кластеру є точка (2; 0), кількість елементів 50, x_1 та y_1 масиви розмірністю 1x50. Також побудуемо цей масив на графіку командами

```
x1=2+rand(1,50)
y1=0+rand(1,50)
plot(x1,y1,'or')
```

Задамо другу множину елементів кластеру №2 з центром (-2;0)

```
x2=-2+rand(1,50)
y2=0+rand(1,50)
plot(x2,y2,'ob')
```

Далі задамо третю множину елементів кластеру №3 з центром (0;2)

```
x3=0+rand(1,50)
y3=2+rand(1,50)
plot(x3,y3,'oy')
```

Аналогічно задамо четверту множину елементів кластера №4 з центром (0;2)

```
x4=0+rand(1,50)
y4=-2+rand(1,50)
plot(x4,y4,'og')
```

Виведемо множину вхідних значень на графік командою ***figure(1)*** **(** рис. 2.42)

```
figure(1)
hold on
plot(x1,y1,'+k')
plot(x2,y2,'sb')
plot(x3,y3,'r*')
plot(x4,y4,'ok')
grid on
hold off
```

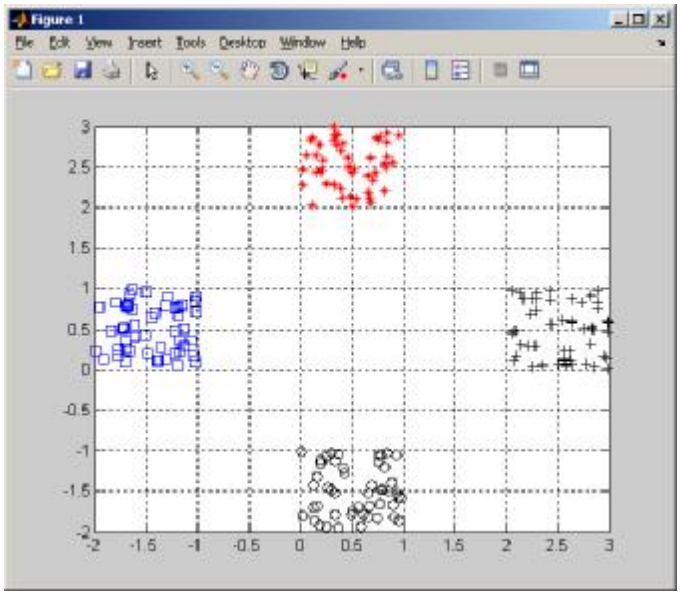


Рис.2.42. Множина вхідних значень

Задамо назви чотирьох кластерів, значення які буде видавати нейронна мережа при симуляції

```
T1(1:50)=1;
T2(1:50)=2;
T3(1:50)=3;
T4(1:50)=4;
```

З'єднаємо всі назви кластерів

```
T(1:50)=T1;
T(51:100)=T2;
T(101:150)=T3;
T(151:200)=T4;
```

З'єднаємо всі вхідні значення в одну матрицю, а саме з'єднаємо *x*

```
x(1:50)=x1;
x(51:100)=x2;
x(101:150)=x3;
x(151:200)=x4;
```

Далі з'єднаємо *y*

```
y(1:50)=y1;
y(51:100)=y2;
y(101:150)=y3;
y(151:200)=y4;
```

Вектори *x* та *y* з'єднаємо в *z*

```
z(1,1:200)=x;
z(2,1:200)=y;
```

Перевіримо розмірності векторів

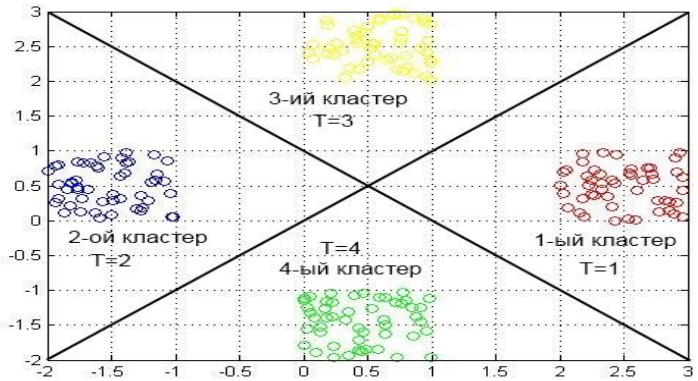
```
size(T) % результат 1x200
size(z)  % результат 2x200
```

Створимо структуру нейронної мережі прямого поширення, а саме трьохшарову нейронну мережу з 10-ма нейронами в першому шарі, 2-ма нейронами в другому шарі та 1-ним нейроном у третьому шарі. Тут *z* – вхідні вектори, *T* – константи мережі, ***logsig, tansig, purelin*** – вид активації шару. 1-ий шар ***logsig***, 2-ий шар ***logsig***, 3-ій шар ***purelin***

```
net = newff(z,T,[10,2],{'logsig','logsig'})
```

Навчимо створену мережу командою

```
net = train(net,z,T);
```



Навчання тривало 19 ітерацій, поки значення середньоквадратичної помилки при тестуванні мережі на перевірочній послідовності не досягло $1x\ 10^{-15}$ (рис.2.43).

Ліворуч наведено результати кластеризації за допомогою нейронну мережу прямого поширення рис.2.43.

Рис. 2.43. Результат кластеризації за наведеним прикладом.

2.5.4. Робота з LVQ нейронною мережею

Нижче розглядаються мережі для класифікації вхідних векторів, або *LVQ* (Learning Vector Quantization) – мережі. Як правило, вони виконують і кластеризацію та класифікацію векторів входу. Ці мережі є розвитком самоорганізованих мереж Кохонена.

За командою ***help lvq*** можна одержати наступну інформацію про М-функції, що входять до складу ППП Neural Network Toolbox і відносяться до побудови *LVQ* – мереж.

Для прикладу розглянемо мережу – LVQ мережу (Learning Vector Quantization) з кількістю нейронів – 5, розмірністю бази знань – 2x200. Ім'я бази знань – *z*. Задамо п'ять множин значень кластерів, центри яких будуть в наступних точках:

1. [0;3]
2. [0;-3]
3. [3;0]
4. [-3;0]
5. [0;0]

```
y0=0+rand(1,40)
x0=0+rand(1,40)
y1=0+rand(1,40)
x1=3+rand(1,40)
y2=0+rand(1,40)
x2=-3+rand(1,40)
y3=3+rand(1,40)
x3=0+rand(1,40)
y4=-3+rand(1,40)
x4=0+rand(1,40)
```

Ось так виглядає розподіл множини усіх точок нейромережі (рис.2.44) після застосування наступних команд

```
figure (1)
hold on
plot(x0,y0,'o')
plot(x1,y1,'ob')
plot(x2,y2,'or')
plot(x3,y3,'og')
plot(x4,y4,'oy')
grid on
hold off
```

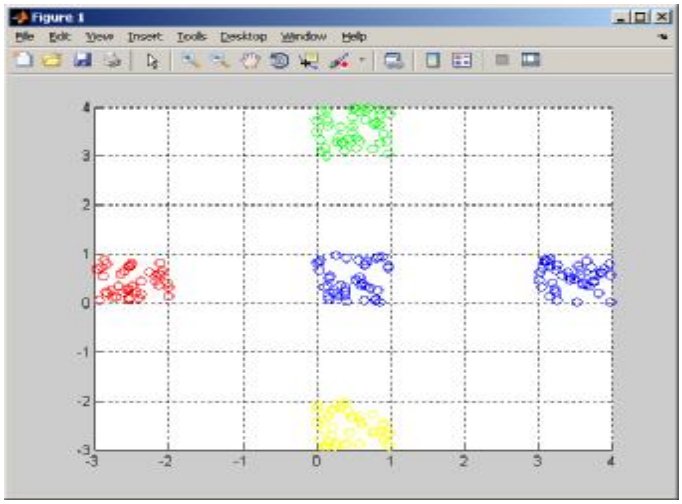


Рис.2.44. Множина вхідних значень

Задамо імена п'яти цільових кластерів та з'єднаємо всі імена кластерів в одну матрицю для подальшої обробки

```
T0(1:40)=5; % 5-й кластер
T3(1:40)=4; % 4-й кластер
T4(1:40)=3; % 3-й кластер
T1(1:40)=1; % 1-й кластер
T2(1:40)=2; % 2-й кластер
T(1:40)=T1;% об'єднаємо в одну матрицю
T(41:80)=T2
T(81:120)=T3;
T(121:160)=T4;
T(161:200)=T0;
```

Перед тим як подавати це на вхід нейромережі, необхідно з'єднати всю базу знань для нейронної мережі в одну матрицю, а саме з'єднаємо *x* та *y* і все це присвоїмо в змінну *z*

```
x(1:40)=x1;
x(41:80)=x2;
x(81:120)=x3;
x(121:160)=x4;
x(161:200)=x0;
y(1:40)=y1;
y(41:80)=y2;
y(81:120)=y3;
y(121:160)=y4;
y(161:200)=y0;
z(1,1:200)=x;
z(2,1:200)=y;
Tc = ind2vec(T);% перетворення у вектор
net = newlvq(z,5,[.2 .2 .2 .2 .2]);
net = train(net,z,Tc);
```

Перевіримо на тих спектрах, які брали участь у навчанні

```
a= sim (net, z)
```

Перевіримо на спектрах, наприклад з похибкою 0,1, які не брали участь в навчанні

```
a = sim (net, (z+0.1))
```

2.5.5. Індивідуальне завдання № 6

Мета: Кластеризація та класифікація на основі графічного інтерфейсу користувача *nnstart* та в командному режимі.

Згенеруйте чотири множини по 30 значень з координатами (-*N*,0), (0,*N*), (*N*,0), (0,-*N*), *N* – номер за списком. Згенеровані сукупності розбийте на чотири кластери з використанням:

- самоорганізованої мережі Кохонена.
- нейронної мережі прямого поширення.
- LVQ нейронної мережі.

Визначте якість класифікації**.**

Оформіть звіт.

2.6. Визначення значення залежної змінної за вхідними параметрами

Нейронні мережі можуть бути використані для функцій підбору значення функції в залежності від вхідних параметрів, коли вид зв'язку $y = f(x)$ аналітично встановити неможливо.

Припустимо, наприклад, що у вас є статистичні дані за якими визначають вартість об'єктів нерухомості. Необхідно, створити нейронну мережу, яка може передбачити вартість будинку (в $ 1000), з урахуванням 13 факторів, що впливають вартість нерухомості: географічних, економічних та технічних. Нехай, маємо статистичні дані про 506 будинків, для яких визначено ці 13 характеристик і пов'язані з ними ринкові вартості.

Вирішити це завдання можна двома способами:

- Використовуючи графічний користувальницький інтерфейс, *nftool*, як описано нижче.
- Використовуючи функції командного рядка, як описано в розділі в *Help* «*Використання функцій командного рядка*».

Як правило, краще почати з GUI, а потім використовують GUI для автоматичного створення сценаріїв командного рядка. Перед використанням будь-якого методу, спочатку необхідно визначити проблему шляхом вибору набору даних. Кожен графічний інтерфейс має доступ до багатьох вибіркових наборів даних, які можна використовувати, при моделювання (див. *Neural Network Toolbox* «Приклади наборів даних»). Якщо у вас є конкретне завдання, яке ви хочете вирішити, ви можете завантажити свої дані в робочу область.

Відкрийте інструмент для моделювання нейронної мережі *Neural Network Start* за допомогою команди.

```
>> nnstart;
```

У вікні натисність кнопку «*Fitting Tool*».

Функція підбору – процес навчання нейронної мережі на наборі вхідних для отримання відповідного (асоційованого, зв’язаного) набору цільових виходів. Після того, як нейронну мережу навчена підбирати дані за навчальною виборкою, утворюється узагальнений зв'язок «вхід–вихід», що може бути використаний для генерації виходів для входів на яких нейронну мережу не навчався.

Приклад. Навчання нейронної мережі для визначенняи середньої ціни об’єкту нерухомості, за статистичними даними.

Файл вхідних даних house_dataset.MAT, завантажує дві змінні:

houseInputs – матриця 13x506 містить тринадцять характеристик 506 різних об’єкту нерухомості в деякому районі міста.

1. Рівень злочинності на душу населення в даному районі міста (на 10 тис. населення).
2. Площа прибудинкової території ($м^2$).
3. Площа зелених насаджень ($м^2$).
4. «1», якщо поряд з річкою, «0» в іншому випадку.
5. Концентрація оксидів азоту (частин на 10 млн.).
6. Площа господарських приміщень ($м^2$).
7. Загальна площа об’єкту нерухомості ($м^2$).
8. Відстань до п'яти центрів зайнятості (км).
9. Індекс доступності транспортних магістралей (1–5).
10. Вартість комунальних послуг (грн./міс.).
11. Термін оренди (роки).
12. Щільність населення (чол./$км^2$).
13. Відсоток малозабезпеченого населення (на 10 тис. населення).

Наприклад: в стовпцях значення атрибутів об’єкту, рядки – об’єкти.

	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**
1	0,00632	18	2,31	0	0,538	6,575	65,2	4,09	1	296	15,3	396,9	4,98
2	0,02731	0	7,07	0	0,469	6,421	78,9	4,9671	2	242	17,8	396,9	9,14
3	0,02729	0	7,07	0	0,469	7,185	61,1	4,9671	2	242	17,8	392,83	4,03
4	0,03237	0	2,18	0	0,458	6,998	45,8	6,0622	3	222	18,7	394,63	2,94
5	0,06905	0	2,18	0	0,458	7,147	54,2	6,0622	3	222	18,7	396,9	5,33

houseTargets – матриця 1x506 серединних значень вартості об’єктів нерухомості в тисячах доларів.

№ об'єкту	**Вартість**
1	24
2	21,6
3	34,7
4	33,4
5	36,2

[*X*, *T*] = *house_dataset* завантажує вхідні та цільові значення змінних, натисканням на кнопку «*Load Example Data Set*» в вікні «***Select Data window***», далі натискаємо «Next». У вікні «***Validation and Test Data***» вхідні дані розподілено наступним чином: 70% – для навчання нейронну мережу, для перевірки достовірності та тестування обрано по 15% вихідних даних.

При таких установках, вхідні та цільові вектори будуть випадковим чином розділені на три групи наступним чином:

- 70% буде використовуватися для навчання.
- 15% буде використовуватися для перевірки, що навчання мережі закінчено.
- Останні 15% будуть використані для незалежного тестування мережі.

Натисніть кнопку «*Next*».

Стандартна мережа, що використовується для підбору є двошаровою мережею з прямим розповсюдженням, з сигмоїдальною передавальною функцією в прихованому шарі і лінійною передавальною функцією у вихідному шарі. За замовчуванням число нейронів в прихованому шарі дорівнює 10. Ви можете збільшити це число, якщо якість розпізнавання нейронну мережу не задовільна (рис. 2.45).

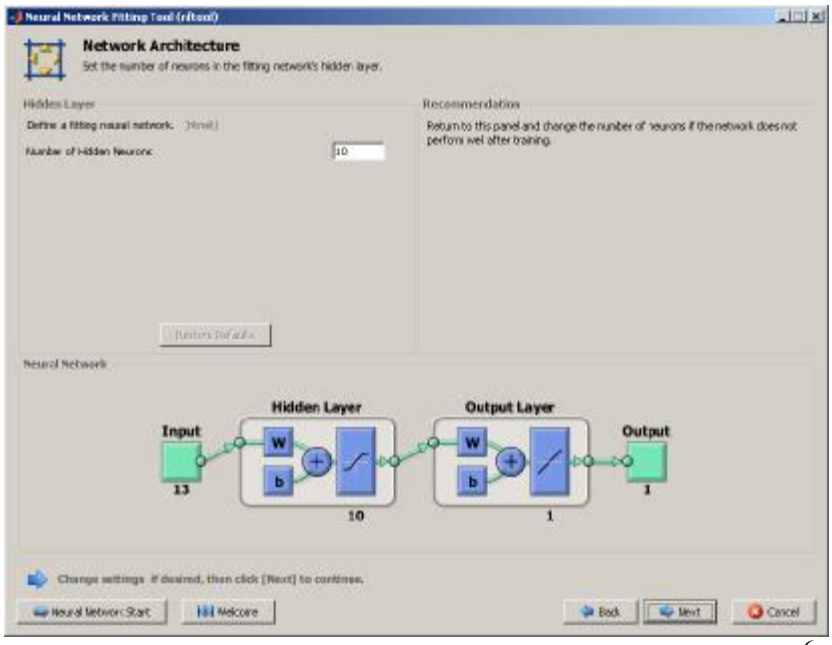


Рис. 2.45. Створення архітектури нейронної сітки

Натисніть «*Next*», для переходу на вкладку «***Train Network***», далі натисніть «*Train*».

Тренування триває до досягнення заданого рівня похибки НС (10^{-6}) на даних для перевірки достовірності.

Зверху на вікні навчання наведено інформацію про архітектуру нейронної сітки, алгоритм навчання, процес навчання: кількість ітерацій, час, продуктивність, градієнт, середньоквадратичну помилку, перевірку відповідності.

Для оцінки якості навчання, переглянемо графіки на панелі «*Plots*» (рис. 2.46).

Наведена нижче інформація використовується для перевірки продуктивності мережі. Представлені графіки регресії (рис. 2.46) відображають відношення мережевих виходів до цільових значень для навчаючої, перевірочної та тестової послідовностей. Для ідеального підбору, дані повинні бути розташовані вздовж лінії в 45 градусів, де виходи мережі дорівнюють цілям. Це завдання, реалізовано досить добре для всіх наборів даних, значення *R* для навчальної вибірки 0,98, для перевірочної *R*=0,92, для тестової *R*=0,91, що є задовільним. Якщо потрібні більш точні результати, можна перенавчити мережу, натиснувши «*Retrain*», або команду *nftool*.

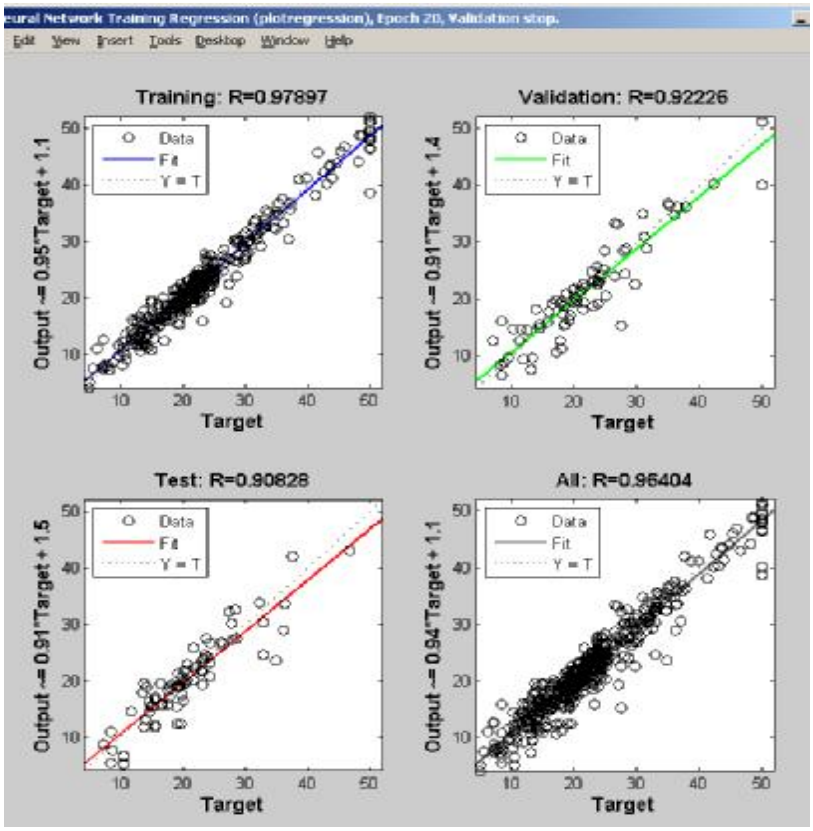


Рис. 2.46. Графіки «*Regression*»

Для додаткового аналізу якості результатів нейронну мережу. На панелі «*Plots*», натисніть «*Error Histogram*» (рис. 2.47).

Сині смуги представляють похибки навчальних даних, зелені смуги представляють похибки даних для перевірки, і червоні прямокутники представляють похибки тестових даних. Гістограма може дати вам уявлення про викиди – аномальні значення вхідних послідовностей.

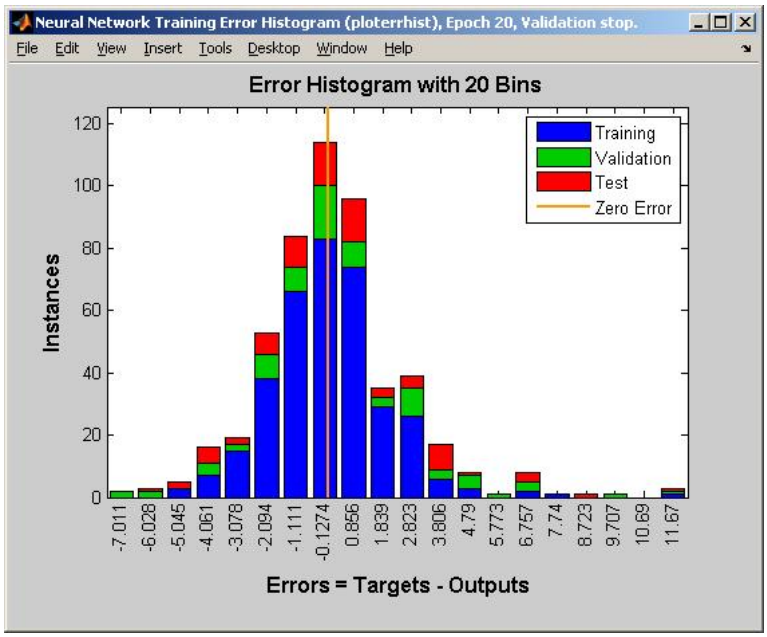


Рис. 2.47 Гістограма «*Error Histogram*»

На гістограмі (рис. 2.47) більшість помилок знаходиться в діапазоні [-5; 6], також є

точки навчаючої, перевірочної та тестової вибірок, для яких похибка між цільовим значенням та виходом нейронну мережу дорівнює 11,67. Ці викиди також видно на рис. 4, на графіку для тестових даних цільове значення близько 33, а вихідне 22. Необхідно перевірити чи ці викиди є аномальними значеннями чи належать до генеральної сукупності. Якщо викиди є допустимими точками вхідних даних, але істотно відмінні від інших даних, то мережа має екстраполювати ці точки. Ви повинні зібрати більше даних, що містять подібні точки до точок викиду, і перенавчити мережу.

Натискаємо кнопку «*Next*». Переходимо до вкладки «***Evaluate Network***» – «Оцінювання якості сітки». На цій вкладці проводиться тестування нейронну мережу на додаткових даних (завантажуються в поля «*Inputs*» та «*Targets*») і вирішується чи задовільна якість отриманих результатів. Поліпшити якість результатів, в даному випадку, прогнозування можна:

- Навчання знову, кнопка «*Train Again*»;
- Коректування структури НС, кнопка «*Adjust Network Size*»;
- Завантаження більшого набору даних кнопка «*Import Larger Data Set*».

Якщо продуктивність (якість) прогнозування на навчальній вибірці задовільна, але на випробувальних даних продуктивність (якість) значно гірша, що може свідчити про перенавчання, то необхідно зменшити кількості нейронів може поліпшити результати. Якщо продуктивність (якість) навчання незадовільна, то ви можете збільшити кількість нейронів.

Коли досягли прийнятної якості прогнозування, натикаємо «*Next*», переходимо на вкладку «***Deploy Solution***» – «*Розгортання Рішення*».

Використовуйте цю панель для генерації розгорнутої версії програмного коду для подальшого моделювання нейронної мережі. Ви можете використовувати згенерований код або діаграми, для того щоб краще зрозуміти, як побудована нейронна мережа обчислює виходи з входів, коректування та подальшого використання НС.

На наступній вкладці «***Save Results***» – «*Збереження Результатів*» використовуйте кнопки для створення сценаріїв або для збереження результатів.

Натисніть кнопку «Simple Script» для збереження створеної нейронної мережі в середовищі Matlab для подальшого аналізу та модифікації.

Описаний вище порядок дій реалізується наступним набором команд

```
x = houseInputs;
t = houseTargets;
 % Створення нейронну мережу для підбору
hiddenLayerSize = 10; % кількість нейронів у прихованому шарі
net = fitnet(hiddenLayerSize);
```

```
% Налаштування розподілу даних для навчання, перевірки, тестування
net.divideParam.trainRatio = 70/100;
```

```
net.divideParam.valRatio = 15/100;
net.divideParam.testRatio = 15/100;
 % Тренування нейронної мережі
[net,tr] = train(net,x,t);
 % Тестування нейронної мережі
y = net(x);
e = gsubtract(t,y); % похибки нейронної мережі
performance = perform(net,t,y)
```

```
% Перегляд нейронної мережі
view(net)
% Графіки
% Розкоментуйте рядки нижче для побудови різних графіків.
%figure, plotperform(tr)
%figure, plottrainstate(tr)
%figure, plotfit(net,x,t)
%figure, plotregression(t,y)
%figure, ploterrhist(e)
```

2.6.1. Індивідуальне завдання № 7

1. Виконайте збережений скрипт з командного вікна *Matlab* та прокоментуйте результати.
2. Згенеруйте вхідні дані для перевірки роботи нейронну мережу.

2.7. Розпізнавання образів

В даний час багато завдань, що мають важливе практичне значення і які не мали прийнятного рішення в минулому, можуть бути вирішені з використанням нейронних мереж. Прикладом є пристрій для розпізнавання образів, що може читати символи. Машина, яка читає банківські чеки, може виконувати за той же самий час набагато більше перевірок, ніж людина. Цей вид додатків зберігає час і гроші, а також усуває умови, при яких людина виконує монотонну, періодично повторювану роботу. Демонстраційна програма сценарій, як розпізнавання символів може бути виконане в мережі зі зворотним поширенням.

Постановка завдання

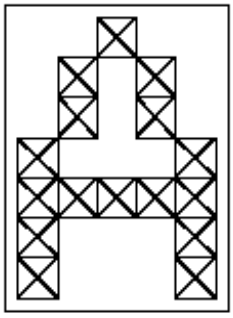
Потрібно створити нейронну мережу для розпізнавання 26 символів латинського алфавіту. Як датчик передбачається використовувати систему розпізнавання, яка виконує оцифровку кожного символу. У результаті кожен

символ буде представлений шаблоном розміру 5×7. Наприклад, символ А може бути представлений, як це показано на рис. 2.48 *а* та *б*.

Однак система зчитування символів зазвичай працює неідеально і окремі елементи символів можуть виявитися зчитаними недосконало (рис. 2.49).

0	0	1	0	0
0	1	0	1	0
0	1	0	1	0
1	0	0	0	1
1	1	1	1	1
1	0	0	0	1
1	0	0	0	1

а



б

Рис. 2.48. Шаблон символу «А»

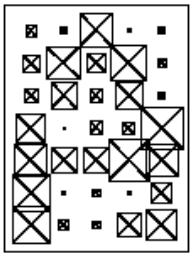


Рис. 2.49 Зчитаний символ з вадами

Проектована нейронна мережа повинна точно розпізнавати ідеальні вектори входу і з максимальною точністю відтворювати зашумлені вектори. *M*-функція *prprob* визначає 26 векторів входу, кожен з яких містить 35 елементів, цей масив називається алфавітом. *M*-функція формує вихідні змінні *alphabet* і *targets*, які визначають масиви алфавіту і цільових векторів. Масив *targets* визначається як *eye* (26). Для того щоб відновити шаблон для *i*-ї букви алфавіту, треба виконати наступні оператори:

```
[alphabet, targets] = prprob;
ti = alphabet(:, i);
letter{i} = reshape(ti, 5, 7)';
letter{i}
```

Визначимо шаблон для символу А, який є першим елементом алфавіту:

```
[alphabet, targets] = prprob;
i = 1;
ti = alphabet(:, i);
letter{i} = reshape(ti, 5, 7)';
letter{i}
```

Виконання цих команд дасть наступний результат

```
ans =     0     0     1     0     0
          0     1     0     1     0
          0     1     0     1     0
          1     0     0     0     1
          1     1     1     1     1
          1     0     0     0     1
          1     0     0     0     1
```

Нейронна мережа

На вхід мережі надходить вектор входу з 35 елементами; вектор виходу містить 26 елементів, тільки один з яких дорівнює 1, а решта – 0. Правильно функціонуюча мережа повинна відповісти вектором зі значенням 1 для елемента, що відповідає номеру символу в алфавіті. Крім того, мережа повинна бути здатною розпізнавати символи в умовах дії шуму. Передбачається, що шум – це випадкова величина із середнім значенням 0 і стандартним відхиленням, меншим або рівним 0.2.

Архітектура мережі

Для роботи нейронної мережі потрібно 35 входів і 26 нейронів у вихідному шарі. Для вирішення завдання виберемо двошарову нейронну мережу з логарифмічними сігмоїдальними функціями активації в кожному шарі. Така функція активації обрана тому, що діапазон вихідних сигналів для цієї функції визначено від 0 до 1, і цього достатньо, щоб сформувати значення вихідного вектора. Структурна схема такої нейронної мережі показана на рис. 2.50.

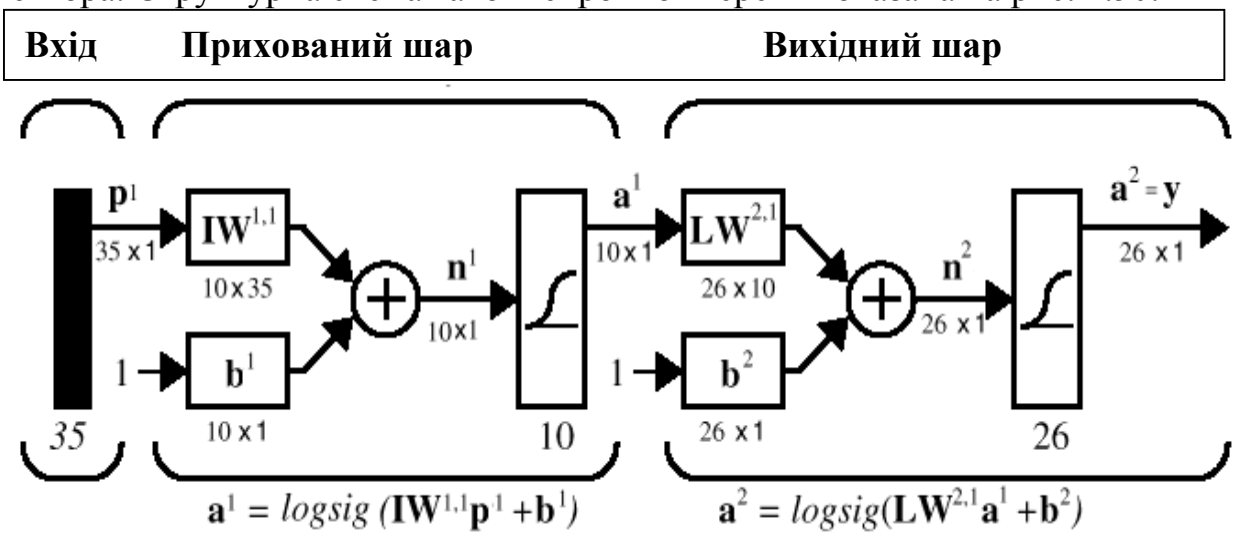


Рис. 2.50. Структурна схема нейронної мережі.

Прихований шар має 10 нейронів. Таке число нейронів вибрано на основі досвіду і розумних припущень. Якщо при навчанні мережі виникнуть труднощі, то можна збільшити кількість нейронів цього рівня. Мережа навчається так, щоб сформувати одиницю в єдиному елементі вектора виходу, позиція якого відповідає номеру символу, і заповнити решту вектора нулями. Однак наявність шумів може призводити до того, що мережа не формуватиме вектора виходу, що складається точно з одиниць і нулів. Тому по завершенні етапу навчання

вихідний сигнал обробляється *M*-функцією *compet*, яка привласнює значення 1 єдиному елементу вектора виходу, а всім іншим – значення 0.

Ініціалізація мережі.

Викличемо *M*-файл *prprob*, який формує масив векторів входу *alphabet* розміру 35×26 з шаблонами символів алфавіту і масив цільових векторів *targets*:

```
[alphabet,targets] = prprob;
[R,Q] = size(alphabet);
[S2,Q] = size(targets);
```

Двошарова нейронна мережа створюється за допомогою команди *newff*:

```
S1 = 10;
net = newff(minmax(alphabet),[S1 S2],{'logsig' 'logsig'},'traingdx');
net.LW{2,1} = net.LW{2,1}*0.01;
net.b{2} = net.b{2}*0.01;
```

Структура нейронної мережі, отримана за допомогою команди **gensim(net),** представлена на рис.2.51.

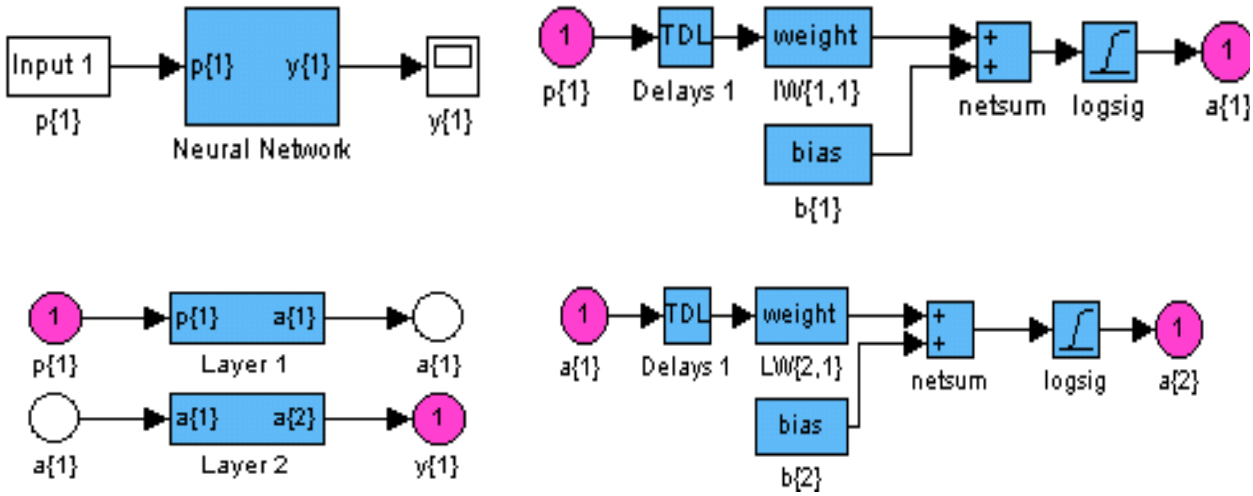


Рис. 2.51. Структура нейронної мережі

Навчання

Щоб створити нейронну мережу, яка може обробляти зашумлені вектори входу, слід виконати навчання мережі як на ідеальних, так і на зашумлених векторах. Спочатку мережа навчається на ідеальних векторах, поки не буде забезпечена мінімальна сума квадратів похибок. Потім мережа навчається на 10 наборах ідеальних і зашумлених векторів. Дві копії вільного від шуму алфавіту використовуються для того, щоб зберегти здатність мережі класифікувати ідеальні вектори входу. На жаль, після того, як описана вище мережа навчилася класифікувати сильно зашумлені вектори, вона втратила здатність правильно класифікувати деякі вектори, вільні від шуму. Отже, мережа знову треба навчити на ідеальних векторах. Це гарантує, що мережа буде працювати правильно, коли на її вхід буде переданий ідеальний символ. Навчання

виконується за допомогою функції *trainbpx*, яка реалізує метод зворотного поширення помилки зі збуренням і адаптацією параметра швидкості налаштування.

Мережа спочатку навчається за відсутності шуму з максимальним числом циклів навчання 5000 або до досягнення допустимої середньої квадратичної похибки, рівній 0,1 (див. команди та рис. 2.52)

```
P = alphabet;
T = targets;
net.performFcn = 'sse';
net.trainParam.goal = 0.1;
net.trainParam.show = 20;
net.trainParam.epochs = 5000;
net.trainParam.mc = 0.95;
[net,tr] = train(net, P, T); %
```

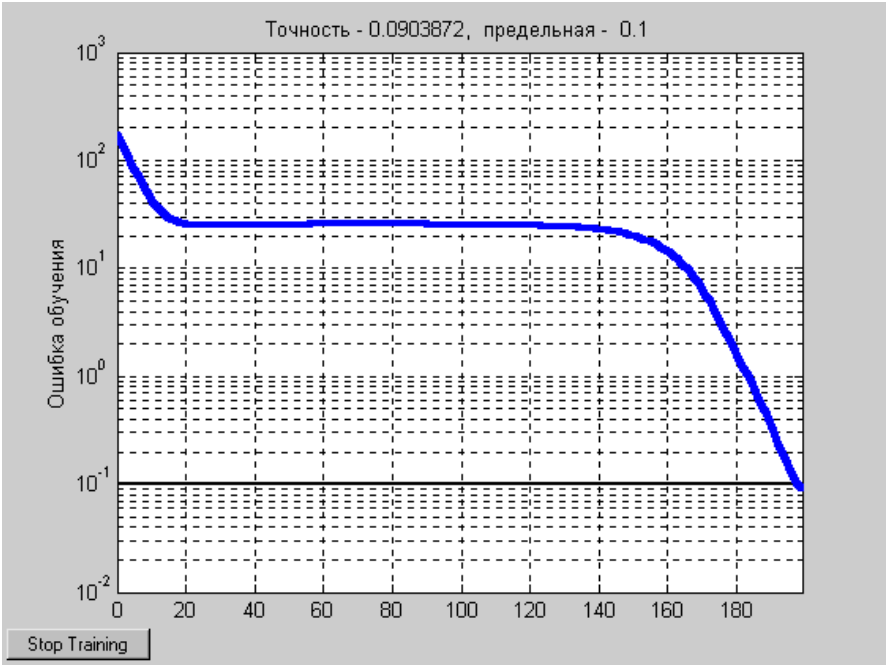


Рис. 2.52. Графік навчання мережі за відсутності шуму.

Щоб спроектувати нейронну мережу, не чутливу до впливу шуму, навчимо її із застосуванням двох ідеальних і двох зашумлених копій векторів алфавіту. Цільові вектори складаються з чотирьох копій векторів. Зашумлені вектори мають шум із середнім значенням 0,1 і 0,2. Це навчає нейрон правильно розпізнавати зашумлені символи і в той же час добре розпізнавати ідеальні вектори. При навчанні з шумом максимальне число циклів навчання скоротимо до 300, а до допустимих похибка збільшимо до 0,6 (див. команди та рис. 2.53).

```
netn = net;
netn.trainParam.goal = 0.6;
```

```
netn.trainParam.epochs = 300;
T = [targets targets targets targets];
```

```
for pass = 1:10
P = [alphabet, alphabet,...
(alphabet + randn(R,Q)*0.1),...
(alphabet + randn(R,Q)*0.2)];
[netn,tr] = train(netn,P,T);
end %
```

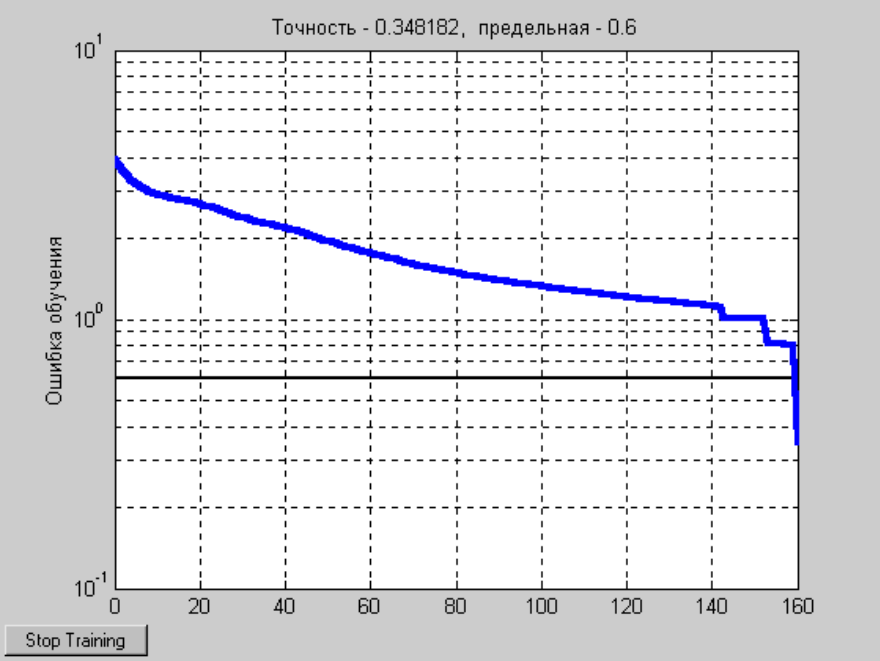


Рис. 2.53. Графік навчання мережі на зашумлених вхідних векторах

Оскільки нейронна мережа навчалася в присутності шуму, то має сенс повторити її навчання без шуму, щоб гарантувати, що ідеальні вектори входу класифікуються правильно.

```
netn.trainParam.goal = 0.1; % Гранична середньоквадратична похибка
netn.trainParam.epochs = 500 ; % Максимальна кількість циклів навчання
net.trainParam.show = 5 ; % Частота виводу результатів на екран
[ netn, tr ] = train ( netn, P, T) ;
```

Ефективність нейронної мережі будемо оцінювати в такий спосіб. Розглянемо дві структури нейронної мережі: мережу 1, навчену на ідеальних послідовностях, і мережу 2, навчену на зашумлених послідовностях. Перевірка функціонування проводиться на 100 векторах входу при різних рівнях шуму. Наведемо фрагмент сценарію *appcr1*, який виконує ці операції:

```
noise_range = 0:.05:.5;
max_test = 100;
network1 = [];
network2 = [];
T = targets;
% Виконати тест
```

```
for noiselevel = noise_range
  errors1 = 0;
  errors2 = 0;
  for i=1:max_test
    P = alphabet + randn(35,26)*noiselevel;
    % Тест для мережі 1
    A = sim(net,P);
    AA = compet(A);
    errors1 = errors1 + sum(sum(abs(AA–T)))/2;
    % Тест для мережі 2
    An = sim(netn,P);
    AAn = compet(An);
    errors2 = errors2 + sum(sum(abs(AAn–T)))/2;
    echo off
  end
 % Середні значення помилок (100 послідовностей з 26 векторів цілей
  network1 = [network1 errors1/26/100];
  network2 = [network2 errors2/26/100];
end
```

Тестування реалізується наступним чином. Шум з середнім значенням 0 і стандартним відхиленням від 0 до 0,5 з кроком 0,05 додається до векторів входу. Для кожного рівня шуму формується 100 зашумлених послідовностей для кожного символу і обчислюється вихід мережі. Вихідний сигнал обробляється *M*- функцією *compet* з тією метою, щоб вибрати тільки один з 26 елементів вектора виходу. Після цього оцінюється кількість помилкових класифікацій і обчислюється відсоток помилки.

Відповідний графік похибки мережі від рівня вхідного шуму показаний на рис. 2.54.

```
plot ( noise_range, network1 * 100, '-', noise_range, network2 * 100 );
```

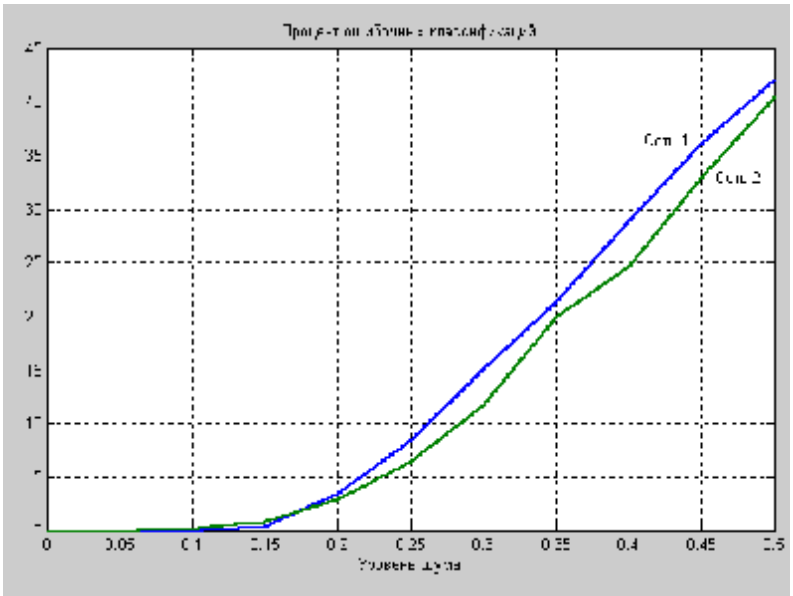


Рис. 2.54. Графік залежності похибки мережі від рівня вхідного шуму.

Мережа 1 навчена на ідеальних векторах входу, а мережа 2 – на зашумлених. Навчання мережі на зашумлених векторах входу значно знижує похибку розпізнавання реальних

векторів входу. Мережі мають дуже малі похибки, якщо середньоквадратичне значення шуму знаходиться в межах від 0,00 до 0,05. Коли до векторів був доданий шум з середньоквадратичним значенням 0,2, в обох мережах почали виникати помітні помилки. При цьому похибки нейронної мережі, навченої на зашумлених векторах, на 3–4% нижче, ніж для мережі, навченої на ідеальних вхідних послідовностях. Якщо необхідна більш висока точність розпізнавання, мережа може бути навчена або протягом більш тривалого часу, або з використанням більшої кількості нейронів у прихованому шарі. Можна також збільшити розмір векторів, щоб користуватися шаблоном з більш дрібною сіткою, наприклад 10×14 точок замість 5×7. Перевіримо роботу нейронної мережі для розпізнавання символів. Сформуємо зашумлений вектор входу для символу *J* (рис. 2.55)

```
noisyJ = alphabet(:,10) + randn(35,1)*0.2;
plotchar(noisyJ); % Зашумлений символ J (рис.8)
A2 = sim(net, noisyJ);
A2 = compet(A2);
answer = find(compet(A2) == 1)
answer = 10
plotchar(alphabet(:,answer)); % Розпізнаний символ J
```

Нейронна мережа виділила 10 правильних елементів і відновила символ J без помилок (рис. 2.56).

Це завдання демонструє, як може бути розроблена проста система розпізнавання зображень. Зауважимо, що процес навчання не складався з єдиного звернення до навчальної функції. Мережа була навчена кілька разів при різних векторах входу. Навчання мережі на різних наборах зашумлених векторів дозволило навчити мережу працювати з зображеннями, спотвореними шумами, що характерно для реальної практики.

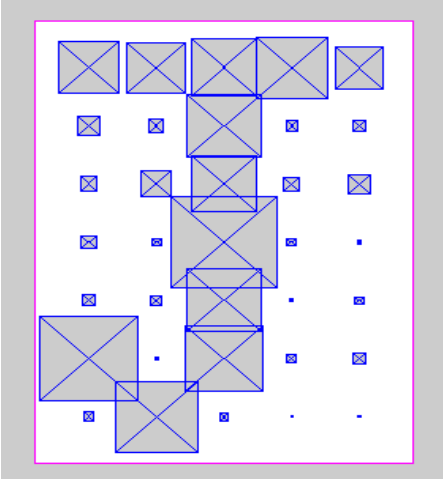


Рис. 2.55. Зашумлений символ *J*

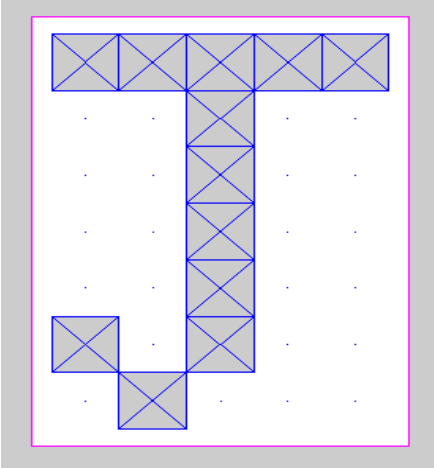


Рис. 2.56. Розпізнаний символ *J*

Запитання для самоперевірки

1. Для чого потрібно виконувати класифікацію економічних об’єктів?
2. Що таке карта Кохонена?
3. Визначте різницю між мережами прямого та зворотного поширення.
4. Для чого потрібно визначати залежну змінну серед вибірки економічних даних?
5. Які команди дозволяють побудувати гістограму?
6. Що таке розпізнавання образів?

Вивчивши матеріали цього розділу, студенти засвоїли набір прийомів і команд Matlab 6, які дозволяють проводити класифікацію об’єктів, створювати економіко-математичні моделі, прогнозувати процеси, що відбуваються в економіці.

ВИСНОВКИ

Використання програмного пакету Matlab 6 дозволяє з легкістю виконувати складні розрахунки, застосовуючи найновіші розробки обчислюваної математики в напрямку нейронних мереж.

Тепер для дослідника, прикладного економіста, плановика відкриваються широкі можливості не тільки по створенню економіко-математичних моделей, які володіють майже стовідсотковою точністю апроксимації, що забезпечує можливість визначати значення залежних параметрів для нових, раніше не існуючих значень незалежних. Ці моделі також мають високі прогнозуючі якості, що відкриває широку перспективу для більш точного планування поточного виробництва, а також для розробки нових інвестиційних проектів. Адже інвестиційні проекти потребують прогнозу таких показників як: рівень інфляції, активність споживачів товару, ціни на сировину та готову продукцію, тощо.

Важливим напрямком використання нейронних мереж є підбір значень параметрів моделі за таблицею експериментальних даних. Цей напрямок є особливо актуальним при створенні моделей періодичних процесів, які зустрічаються надзвичайно часто, адже зміна сезонів року викликає періодичну зміну споживацьких інтересів. Але періодичні моделі представляють собою трансцендентні функції, тому визначення їх коефіцієнтів завжди викликає певні проблеми. Завдяки нейронним мережам ці проблеми усунуті.

Класифікація економічних об’єктів завжди була важливим напрямком роботи економістів, які сферою своєї діяльності вважають макроекономіку. Визначення однорідних груп економічних об’єктів дозволяє точніше планувати та керувати розвитком регіону, а то й усієї держави. Нейронні мережі дозволяють проводити розбиття певних об’єктів на групи за критерієм близькості до центра кластері і віддаленості від інших кластерів.

Розпізнавання образів, описане в останньому пункті посібника може бути використане не тільки для автоматизації зчитування рукописних літер. Цей метод має значно ширші сфери застосування. Так, методом розпізнавання образів можна серед великої групи підприємств визначити наприклад, такі, що ухиляються від оподаткування. Або, визначити такі параметри торгівлі цінними паперами на валютних, стокових та товарних біржах, які найкраще характеризують прибутковість цих активів.

Отже, шановні читачі, у ваших руках тепер є надпотужний інструмент, який дозволить вам вирішувати економічні задачі будь-якої складності. Користуйтеся ним.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Amari S. Dualistic geometry of manifold of higher-order neurons // Neural networks. 1991. V.4. – P.443-451.
2. Amari S. Mathematical foundations of neurocomputing // Proceedings of the IEEE. 1990. V.78. N.9. – P.1443-1462.
3. Amit D.J., Gutfreund H., Sompolinsky H. Spin-glass model of neural networks // Physical Review A. 1985. V.32. N.2. – P.1007-1018.
4. Amit D.J., Gutfreund H., Sompolinsky H. Storing infinite number of patterns in a spin-glass model of neural networks // Physical Review Letters. 1985. V.55. N.14. – P.1530-1533.
5. Carpenter G.A., Grossberg S. A massively parallel architecture for selforganizing neural pattern recognition machine // Comput. Vision, Graphics, Image Process. 1987. V.37. N.1. – P. 54-115.
6. Conrad M. Evolutinary learning circuits // J. Theor. Biol. 1974. V.46. N.1. – P.167-188.
7. Fukushima K. Neocognitron: A hierarchical neural network capable for visual pattern recognition // Neural networks. 1988. V.1. N.2. – P.119-130.
8. Gutfreund H. Neural networks with hierarchically correlated patterns // Physical Review A. 1988. V.37. N.2. – P.570-577.
9. Hebb D.O. The organization of behavior. A neuropsychlogical theory. N.Y.: Wiley & Sons, 1949. – 355 p.
10. Hopfield J.J. Neural networks and physical systems with emergent collective computational abilities // Proc. Natl. Acad. Sci. USA. 1982. V.79. N.8. – P.2554-2558.
11. Hopfield J.J. Neurons with gradual response have collective computational properties like those of two-state neurons // Proc. Natl. Acad. Sci. USA. 1984. V.81. N.10. – P.3088-3092.
12. Hopfield J.J., Tank D.W. Computing with neural circuits: A model. // Science. 1986. V.233. N.464. – P.625-633.
13. http://alife.narod.ru/lectures/neural/Neu_index.htm
14. http://ap-economics.narod.ru/
15. http://kpi.ua/
16. http://neuroforex.net
17. http://tora-centre.ru/program
18. http://www.calsci.com
19. http://www.investzem.ru

20. http://www.keldysh.ru/pages/BioCyber/Lectures/Lecture11/Lecture11.html
21. http://www.krugosvet.ru – Энциклопедия.
22. http://www.neuroproject.ru
23. http://www.onlinedown.com
24. http://www.orc.ru
25. http://www.statsoft.ru
26. Kohonen T. Self-organized formation of topologically correct feature maps // Biol.Cybern. 1982. V.43. N.1. – P.56-69.
27. Poggio T., Girosi F. Networks for approximation and learning // Proceedings of the IEEE. 1990. V.78. N.9. – P.1481-1497.
28. Rosenblatt F. 1962. Principles of Neurodinamics. New York: Spartan Books.
29. Rumelhart D.E., Hinton G.E., Williams R.G. Learning representation by back-propagating error // Nature. 1986. V.323. N.6088. – P. 533-536.
30. Барский А.Б. Нейронные сети: распознавание, управление, принятие решений. – М.: Финансы и статистика, 2004. – 176 с.: ил. – (Прикладные информационные технологии).
31. Башмаков А.И., Башмаков И.А. Интеллектуальные информационные технологии: учебное пособие.– М.: Изд-во МГТУ им.Баумана, 2005. – 304 с
32. Енциклопедія кібернетики. Т. 2. – Київ : Українська радянська енциклопедія, 1973. – 574 с.
33. Жильцов В.В., Чувикова В.В. Моделирование интеллектуальной системы технической диагностики нефтегазовых скважин с использованием программы «Statistica Neural Networks» // Всероссийская научно-техническая конференция «Роль механики в создании эффективных материалов, конструкций и машин XXI века»: труды – Омск, 2006. – С.20-23.
34. В.В. Жильцов, В.В. Чувикова. Практикум по нейросетевым технологиям: учебно-методическое пособие. – Омск: СибАДИ, 2010. – 60 с.
35. Комашинский В.И., Смирнов Д.А. Нейронные сети и их применение в системах управления и связи. – М.: Горячая линия – Телеком, 2003. – 94 с.
36. Енциклопедія кібернетики. Т. 2. – Київ : Українська радянська енциклопедія, 1973. – 574 с.
37. Либерман .Е.А. Молекулярная вычислительная машина клетки (МВМ). Общие соображения и гипотезы // Биофизика. 1972. Т.17. N.5. С.932-943.
38. Мак-Каллок У.С., Питтс У. Логическое исчисление идей, относящихся к нервной активности // Автоматы, под ред. Шеннона К.Э. и Маккарти Дж. М.: ИЛ, 1956. С. 362 – 384.
39. Минский М., Пейперт С. Перцептроны. Мир, 1971.

40. Модели и методы искусственного интеллекта. Применение в экономике: учебное пособие / М.Г. Матвеев, А.С. Свиридов, Н.А. Алейников.– М.: Финансы и статистика; ИНФРА-М, 2008.– 488 с.
41. Нейронные сети, генетические алгоритмы и нечеткие системы/ пер. с польск. И.Д. Рудинского; Д. Рутковская, М. Пилиньский, Л. Рутковский. – М.: Горячая линия – Телеком, 2006.– 452 с.
42. Решения и развитие интеллектуальной технологии мониторинга и управления механизированным фондом скважин / В.В. Жильцов, А.В. Дударев, В.П. Демидов и др. //Нефтяное хозяйство. – 2006. – №10.– С.12–14.
43. Розенблат Ф. Принципы нейродинамики. Перцептроны и теория механизмов мозга. Мир, 1965.
44. Романов В.П. Интеллектуальные информационные системы в экономике: Учебное пособие / Под ред. д.э.н., проф. Н.П. Тихомирова. — М.: Издательство «Экзамен», 2003. – 496 с.
45. Уоссермен Ф. Нейрокомпьютерная техника. Теория и практика. М.: Мир, 1972. – 238 с.
46. Фомин С.В., Беркенблит М.Б. Математические проблемы в биологии. М.: Наука, 1973. – 200 с.
47. Фон Нейман Дж. Вероятностная логика и синтез надежных организмов из ненадежных компонент. // Автоматы, под ред. Шеннона К.Э. и Маккарти Дж. М.: ИЛ, 1956. С. 68 – 139.
48. Фон Нейман Дж. Теория самовоспроизводящихся автоматов. М.: Мир, 1971. – 382 с.
49. Фролов А.А., Муравьев И.П. Информационные характеристики нейронных сетей. М.: Наука, 1988. – 160 с.
50. Фролов А.А., Муравьев И.П. нейронные модели ассоциативной памяти. М.: Наука, 1987. – 160 с.
51. Хакен Г. Информация и самоорганизация: Макроскопический подход к сложным системам. М.: Мир, 1991. 240 с.
52. Цитоловский Л.Е. Интегративная деятельность нервных клеток при записи следа памяти // Успехи физиол. наук. 1986. Т.17. N.2. С.83-103.
53. Ясницкий Л.Н. Введение в искусственный интеллект: учебное пособие для студ. высш. учеб. заведений. – М.: Издательский центр «Академия», 2005.–176 с.

ПРЕДМЕТНИЙ ПОКАЖЧИК

GUI- інтерфейс для ППП NNT 44
Автоматична класифікація 34
Апроксимація багатовимірної функції 36
Визначення значення залежної змінної за вхідними параметрами 87
Використання GUI-інтерфейсу пакета нейронних мереж 57
Кластеризація за допомогою самоорганізованої мережі Кохонена 80
Кластеризація за допомогою функції «Clustering Tool» 77
Кластеризація та класифікація з використанням нейронної мережі 76
Мережі зустрічного поширення 22
Навчальний алгоритм зворотного поширення 16
Навчання нейронної мережі 59
Налаштування Matlab6 39
Підбір параметрів 67
Порядок застосування програмного пакета Matlab6 39
Початок роботи з прогнозування 67
Приклад кластеризації даних з використанням нейронної мережі прямого поширення 82
Приклад прогнозування курсу валют 73
Приклад розрахунку вагових коефіцієнтів 30
Прогнозування з використанням нейронної мережі 67
Прогнозування одновимірної функції 35
Робота з LVQ нейронною мережею 85
Робота зі створеної мережею 60
Розпізнавання образів 92
Створення нейронної мережі 57
Теорія застосування нейронних сіток 34
Тестування нейронної мережі 67

Навчальне видання

Пістунов Ігор Миколайович
Антонюк Оксана Петрівна

НЕЙРОМЕРЕЖЕВІ ТЕХНОЛОГІЇ В ЕКОНОМІЦІ ТА ФІНАНСАХ З РОЗРАХУНКАМИ НА КОМП'ЮТЕРІ

Навчальний посібник

Видано за редакцією автора

Підписано до видання 01.07.2014.
Електронний ресурс. Авт. арк. 3,78.

Підготовлено й видано
в Державному вищому навчальному закладі
«Національний гірничий університет».
Свідоцтво про внесення до Державного реєстру ДК № 1842, від 11.06.2004 р..
49600, м. Дніпропетровськ, просп. К. Маркса, 19.