Міністерство освіти і науки України
Державний університет «Житомирська політехніка»

МЕТОДИЧНІ РЕКОМЕНДАЦІЇ

для проведення лабораторних робіт

з навчальної дисципліни

«ШТУЧНИЙ ІНТЕЛЕКТ В ЗАДАЧАХ КІБЕРБЕЗПЕКИ»

для студентів освітнього рівня «бакалавр»

Житомир
2021

УДК 347.77:(007:004.056)

Рекомендовано до друку науково-методичною радою Державного університету «Житомирська політехніка» (протокол № 6 від 04.11.2021 року)

Рецензенти:
А. А. Єфіменко – кандидат технічних наук, доцент
І. І. Сугоняк – кандидат технічних наук, доцент

Методичні рекомендації для проведення лабораторних робіт з навчальної дисципліни «Штучний інтелект в задачах кібербезпеки» для студентів освітнього рівня «бакалавр». Частина 1/ підг. І. В. Пулеко, П.П. Топольницький, В.О. Філіпов. – Житомир : Державний університет «Житомирська політехніка», 2021. – 124 с.

Методичні рекомендації призначені для студентів освітнього рівня «бакалавр», які навчаються за спеціальністю 125 – Кібербезпека кафедри комп'ютерної інженерії та кібербезпеки факультету інформаційно-комп'ютерних технологій.

У частині 1 викладено теоретичний матеріал для підготовки, завдання та методичні рекомендації до виконання лабораторних робіт № 1-4 з освітньої компоненти «Штучний інтелект в задачах кібербезпеки». Лабораторні роботи побудовані на основі можливостей середовища MATLAB.

Підготували: кандидат технічних наук, доцент **І. В. Пулеко**, кандидат технічних наук, доцент **П. П. Топольницький,** старший викладач **В.О. Філіпов.**

УДК 347.77:(007:004.056)

ЗМІСТ

ВСТУП

Кількість атак на інформаційні системи зростає щороку двозначними темпами. При цьому атаки стають все витонченішими, потенційних цілей, в число яких тепер входять пристрої Інтернету речей і розумні домашні пристрої, - все більше, а збиток від атак - все вище. «Класичні» засоби антивірусної боротьби вже не здатні впоратися з такими епідеміями, і тому на допомогу приходять рішення на базі штучного інтелекту.

Вже зараз методи штучного інтелекту широко застосовуються для рішення цілого ряду задач кібербезпеки, тому вивчення їх дозволить майбутнім фахівцям оволодіти перспективними технологіями в обраній спеціалізації.

Розроблені методичні рекомендації призначені для студентів освітнього рівня «бакалавр», які навчаються за спеціальністю 125 – Кібербезпека кафедри комп’ютерної інженерії та кібербезпеки факультету інформаційно-комп'ютерних технологій.

У частині 1 викладено теоретичний матеріал для підготовки, завдання та методичні рекомендації до виконання лабораторних робіт № 1-4 з освітньої компоненти «Штучний інтелект в задачах кібербезпеки». Лабораторні роботи розраховані на 4 год. і направлені на розвиток фахових компетентностей:

КФ 2. Здатність до використання інформаційно-комунікаційних технологій, сучасних методів і моделей інформаційної безпеки та/або кібербезпеки.

КФ 12. Здатність аналізувати, виявляти та оцінювати можливі загрози, уразливості та дестабілізуючі чинники інформаційному простору та інформаційним ресурсам згідно з встановленою політикою інформаційної та/або кібербезпеки.

Практичні завдання побудовані на основі можливостей середовища MATLAB та з використанням технічної документації на нього.

ЛАБОРАТОРНА РОБОТА № 1

МОДЕЛЮВАННЯ ЕЛЕМЕНТІВ ТЕОРІЇ НЕЧІТКИХ МНОЖИН ТА ФОРМУВАННЯ НЕЧІТКИХ ПРАВИЛ

Мета роботи: *ознайомлення з основними функціями належності та процедурами роботи з нечіткими множинами в середовищі MATLAB та можливості їх застосування при рішенні задач кібербезпеки.*

ТЕОРЕТИЧНІ ВІДОМОСТІ

Для виконання лабораторної роботи необхідно вивчити теоретичний матеріал лекцій по теорії нечітких множин та матеріал поданий у цьому підрозділі.

Функції приналежності нерозривно пов'язані із нечіткими множинами. Тип функції приналежності в значному ступені визначає властивості нечіткої системи [1].

Задавання функцій приналежності можна здійснювати у вигляді списку з явним перерахуванням усіх елементів та відповідних ним значень функції приналежності (наприклад, використовуючи відносні частоти за даними експерименту як значення приналежності), або аналітично у вигляді формул (наприклад, використовуючи типові форми кривих для завдання функцій приналежності з уточненням їхніх параметрів відповідно до даних експерименту).

Існують прямі та непрямі *методи побудови функцій приналежності.*

При використанні *прямих методів* експерт просто задає для кожного $x \in E$ значення $\mu(x)$. *Як* правило, прямі методи побудови функції приналежності використовуються для вимірних понять, таких як швидкість, час, відстань, тиск, температура і т. д., або тоді, коли виділяються полярні значення.

Непрямі методи визначення значень функції приналежності використовуються у випадках, коли немає вимірних елементарних властивостей, через які визначається нечітка множина. Як правило, *це методи попарних порівнянь.* Якщо значення функцій приналежності відомі, наприклад, $\mu_A(x_i) = w_i$, $i = 1, 2, ..., n$, то попарні порівняння можна подати матрицею відношень $A = \{a_{ij}\}$, де $a_{ij} = w_i/w_j$ (операція розподілу).

На практиці експерт сам формує матрицю А, при цьому передбачається, що діагональні елементи дорівнюють 1, а для елементів,

симетричних щодо головної діагоналі, $a_{ij} = 1/a_{ji}$ тобто якщо один елемент оцінюється як в *a* разів більш значущий ніж інший, то цей останній повинний бути в *1/a* разів більш значущим, ніж перший. У загальному випадку задача зводиться до пошуку вектора *w*, що задовольняє рівнянню виду: $Aw = \lambda_{max} w$, де λ_{max} - найбільше власне значення матриці *A*. Оскільки матриця *A* позитивна за побудовою, розв'язок даної задачі існує і є позитивним.

ЗАВДАННЯ НА ЛАБОРАТОРНУ РОБОТУ № 1 ТА МЕТОДИЧНІ РЕКОМЕНДАЦІЇ ДО ЇХ ВИКОНАННЯ

Завдання 1.1. Побудуйте трапецеподібну функцію належності

```
x = 0:0.1:10;
y = trapmf(x,[1 5 7 8]);
plot(x,y)
xlabel('trapmf, P = [1 5 7 8]')
ylim([-0.05 1.05])
```

Графік функції належності занесіть у звіт (рис. 1).

Змініть трапецію так, щоб точки згину були 2,5,6,8.

Графік функції належності занесіть у звіт (рис. 2).

Зробіть висновок чим відрізняються графіки.

Завдання 1.2. Побудуйте трикутну функцію належності

```
x = 0:0.1:10;
y = trimf(x,[3 6 8]);
plot(x,y)
xlabel('trimf, P = [3 6 8]')
ylim([-0.05 1.05])
```

Графік функції належності занесіть у звіт (рис. 3).

Змініть трикутник так, щоб точки згину були 2,5,7.

Графік функції належності занесіть у звіт (рис. 4).

Зробіть висновок чим відрізняються графіки.

Завдання 1.3. Побудуйте функцію належності у формі Z

```
x = 0:0.1:10;
```

```
y = zmf(x,[3 7]);
plot(x,y)
xlabel('zmf, P = [3 7]')
ylim([-0.05 1.05])
```

Графік функції належності занесіть у звіт (рис. 5).
Змініть функцію так, щоб точки згину були 3,5.
Графік функції належності занесіть у звіт (рис. 6).
Зробіть висновок чим відрізняються графіки.

Завдання 1.4. Побудуйте функцію належності у формі S

```
x = 0:0.1:10;
y = smf(x,[1 8]);
plot(x,y)
xlabel('smf, P = [1 8]')
ylim([-0.05 1.05])
```

Графік функції належності занесіть у звіт (рис. 7).
Змініть функцію так, щоб точки згину були 2,7.
Графік функції належності занесіть у звіт (рис. 8).
Зробіть висновок чим відрізняються графіки.

Завдання 1.5. Побудуйте функцію належності у формі Pi

```
x = 0:0.1:10;
y = pimf(x,[1 4 5 10]);
plot(x,y)
xlabel('pimf, P = [1 4 5 10]')
ylim([-0.05 1.05])
```

Графік функції належності занесіть у звіт (рис. 9).
Змініть функцію так, щоб точки згину були 1,3,5,8.
Графік функції належності занесіть у звіт (рис. 10).
Зробіть висновок чим відрізняються графіки.

Завдання 1.6. Побудуйте Гаусову (Gaussian) функцію належності

```
x = 0:0.1:10;
y = gaussmf(x,[2 5]);
plot(x,y)
xlabel('gaussmf, P=[2 5]')
```


Графік функції належності занесіть у звіт (рис. 11).
Змініть функцію так, щоб математичне сподівання було 4, а дисперсія 1.
Графік функції належності занесіть у звіт (рис. 12).
Зробіть висновок чим відрізняються графіки.

Завдання 1.7. Побудуйте дзінкоподібну функцію належності

```
x = 0:0.1:10;
mfparams = [2 4 6];
mftype = 'gbellmf';
y = evalmf(x,mfparams,mftype);
%
% Plot the evaluation.
plot(x,y)
xlabel('gbellmf, P = [2 4 6]')
```

Графік функції належності занесіть у звіт (рис. 13).
Змініть функцію так, щоб згин був 1,3,5.
Графік функції належності занесіть у звіт (рис. 14).
Зробіть висновок чим відрізняються графіки.

Завдання 1.8. Використовуючи будь-яку мову програмування високого рівня розробіть програму побудови функції належності згідно варіантів поданих у таблиці 1

Таблиця 1

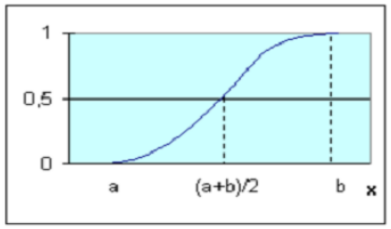
№ за списком у журналі	1	2	3	4	5	6	7	8	9	10
№ варіанту	1	2	3	4	5	6	7	1	2	3

№ за списком у журналі	11	12	13	14	15	16	17	18	19	20
№ варіанту	4	5	6	7	1	2	3	4	5	6

№ за списком у журналі	21	22	23	24	25	26	27	28	29	30
№ варіанту	7	1	2	3	4	5	6	7	1	2

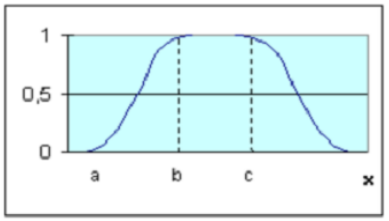
Варіант 1

$$\mu_1(x,a,b) =$$
$$= \begin{cases} 0, & \text{если } x \le a; \\ \dfrac{2(x-a)^2}{(b-a)^2}, & \text{если } a < x \le \dfrac{a+b}{2}; \\ 1-\dfrac{2(x-a)^2}{(b-a)^2}, & \text{если } \dfrac{a+b}{2} < x < b; \\ 1, & \text{если } x \ge b. \end{cases}$$



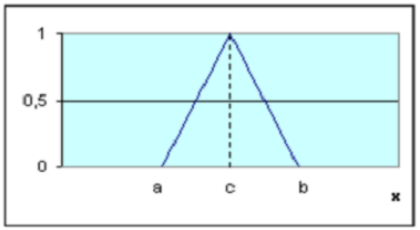
Варіант 2

$$\mu_2(x,a,b,c) =$$
$$= \begin{cases} \mu_1(x,a,b), & \text{если } x < b; \\ 1, & \text{если } b \le x \le c; \\ 1-\mu_1(x,c,c+b-a), & \text{если } x > c. \end{cases}$$



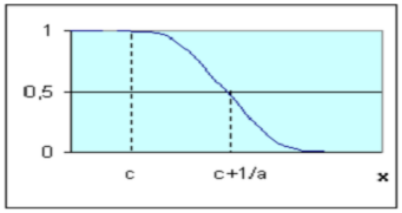
Варіант 3

$$\mu_3(x,a,b,c) =$$
$$= \begin{cases} 0, & \text{если } x \le a; \\ \dfrac{x-a}{c-a}, & \text{если } a < x \le c; \\ \dfrac{b-x}{b-c}, & \text{если } c < x < b; \\ 0, & \text{если } x \ge b. \end{cases}$$



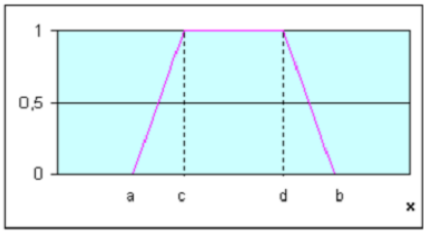
Варіант 4

$$\mu_4(x,a,b,c) =$$
$$= \begin{cases} 1, & \text{если } x \le c; \\ \{1+[a(x-c)]^b\}^{-1}, & \text{если } x > c. \end{cases}$$



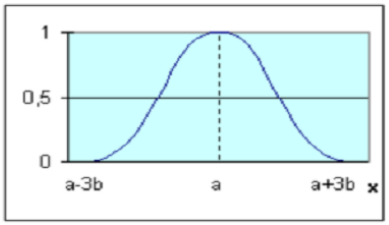
Варіант 5

$$\mu_5(x,a,b,c,d) =$$
$$= \begin{cases} 0, & \text{если } x \le a; \\ \dfrac{x-a}{c-a}, & \text{если } a < x < c; \\ 1, & \text{если } c \le x \le d; \\ \dfrac{b-x}{b-d}, & \text{если } d < x < b; \\ 0, & \text{если } x \ge b. \end{cases}$$



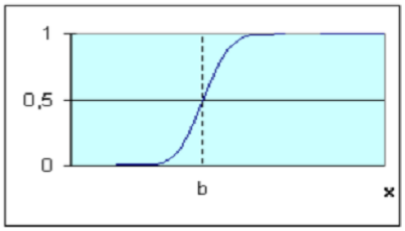
Варіант 6

$$\mu_6(x,a,b) = \exp\left[-\frac{(x-a)^2}{2b^2}\right]$$



Варіант 7

$$\mu_7(x,a,b) =$$
$$= \{1+\exp[-a(x-b)]\}^{-1}$$



Занесіть у звіт текст програми (шрифтом 10 TNR) та побудований графік функції.

Завдання 1.9. Зробіть висновки у яких укажіть:

- призначення функції належності;
- чому на вашу думку використовують так багато різних функцій належності.

Завдання 1.10. За допомогою СНВ зобразити поверхню функції $y = (x_1^2 - 8)\cos(x_2)$ на множині $x1 \in [0,4], x2 \in [0,4]$.

Проектування системи нечіткого виводу слід проводити на основі графічного зображення вказаної залежності. Для цього в М-файлі складемо наступну програму:

```
%Побудова графіка функції y=(x1^2-8)*cos(x2)
%в області x1є[0,4] и x2є[0,4].
n=15;
x1=0:4/(n-1):4;
x2=0:4/(n-1):4;
y=zeros(n,n);
for j=1:n
y(j,:)=(x1.^2-8)*cos(x2(j));
end
surf(x1,x2,y)
xlabel('x1')
```

```
ylabel('x2')
zlabel('y')
title('Target');
```

В результаті виконання цієї програми отримуємо графічне зображення, яке наведено на рис 1.

Зображення занесіть у бланк звіту (рис.15).

Проектування СНВ складається з наступних кроків.

Крок 1. Завантажити основний fis-редактор в (редактор нечіткого виводу) введенням в командному рядку слова **fuzzy.** Після чого з'явиться вікно редактору нечіткого виводу.

Крок 2. Ввести нову вхідну змінну. Для цього вибрати пункт **Add Input** в меню **Edit.**

Крок 3. Перейменувати першу вхідну змінну. Для цього слід зробити одне натиснення лівої кнопки миші на блоці **Input1,** ввести нове позначення **x1** в поле редагування імені поточної змінної і натиснути **<Enter>.**

Крок 4. Перейменувати другу вхідну змінну. Для цього зробити одне натиснення лівої кнопки миші на блоці **input2,** ввести нове позначення **x2** в поле редагування імені поточної змінної і натиснути **<Enter>.**

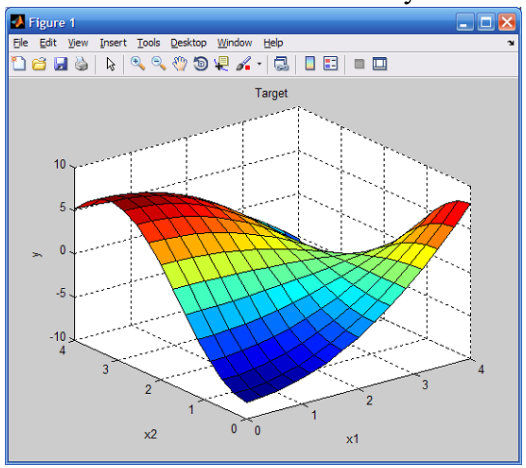


Рис. 1.1.

Крок 5. Перейменувати вихідну змінну. Для цього зробити одне натиснення лівої кнопки миші на блоці **output1,** ввести нове позначення **y** в поле редагування імені поточної змінної і натиснути **<Enter>.**

Крок 6. Задати ім'я системі. Для цього в меню **File** вибрати в підменю **Export to disk** і ввести ім'я файлу, наприклад, **first.**

Крок 7. Перейти в редактор функцій належності. Для цього зробити швидко подвійне натиснення лівої кнопки миші на блоці **x1.**

Крок 8. Задати діапазон зміни змінної **x1.** Для цього надрукувати **0 4** в поле **Range** і натиснути **<Enter>.**

Крок 9. Задати функції належності змінної **x1.** Для лінгвістичної оцінки цієї змінної будемо використовувати 3 терми з трикутними функціями належності. Якщо в вікні немає ще функцій належності, тоді в меню **Edit** слід вибрати команду **Add MFs...** В результаті з'явиться діалогове вікно вибору типу і кількості функцій належності. За замовченням ці 3 терми мають трикутну функцію належності. Тому просто потрібно натиснути **<Enter>.**

Крок 10. Задати *найменування термів змінної* **x1.** Для цього робимо одне натиснення лівою кнопкою миші на графіку першої функції належності. (див. рис. 1.2). Потім вводимо найменування терму, наприклад, **L (Низький),** в полі **Name** і натискаємо **<Enter>.** Потім робимо одне натиснення лівою кнопкою миші на графіку другої функції належності і вводимо найменування терму, наприклад, **A (Середній),** в полі **Name i** натискаємо **<Enter>.** Ще раз робимо одне натиснення лівою кнопкою миші по графіку третьої функції належності і введемо найменування терму, наприклад, **H (Високий),** в полі **Name i** натискаємо **<Enter>.** В результаті отримуємо графічне вікно, яке зображено на рис. 1.2.

Крок 11. Задамо функції належності змінної **x2.** Для лінгвістичної оцінки цієї змінної будемо використовувати 5 термів з гаусовськими функціями належності. Для цього активізуємо змінну **x2** за допомогою натиснення лівої кнопки миші на блоці **x2.** Задамо діапазон змін **x2.** Для цього надрукуємо 0 **4** в полі **Range** (див. рис. 1.3) і натиснемо **<Enter>.** Потім в меню **Edit** виберемо команду **Add MFs....** В діалоговому вікні, що з'явиться, оберемо тип функції належності **gaussmf** в полі **MF type** і **5** термів в полі **Number of MFs.** Після чого натискаємо **<Enter>.**

Крок 12. За аналогією з кроком *10* задамо наступні найменування термів змінної **x2: L (Низький), LA (Нижче середнього), A (Середній), HA (Вище середнього), H (Високий).** В результаті отримуємо графічне вікно, яке зображене на рис. 1.3.

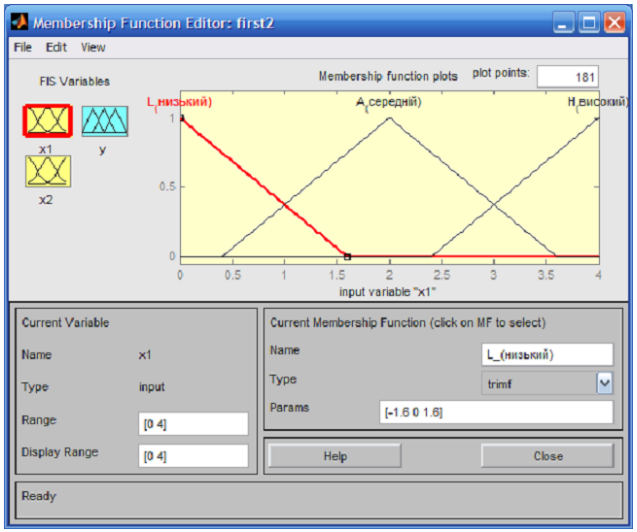


Рис. 1.2.

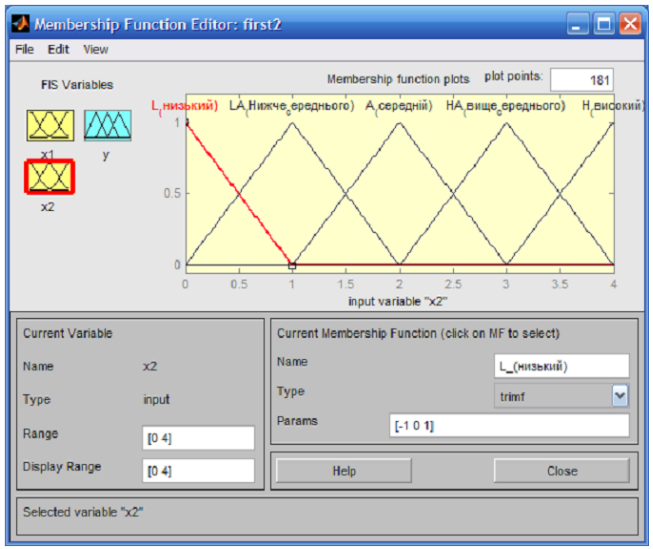


Рис. 1.3.

Крок 13. Задамо функції належності змінної **y.** Для лінгвістичної оцінки цієї змінної будемо використовувати 5 термів з трикутними функціями належності. Для цього активуємо змінну **y** за допомогою натиснення лівої кнопки миші на блоці **y.** Задамо діапазон змін змінної **y.** Для цього

надрукуємо **-10 10** в полі **Range** (див. рис. 1.4) і натиснемо **<Enter>.** Потім в меню **Edit** оберемо команду **Add MFs**. В діалоговому вікні, що з'явиться, виберемо **5** термів в полі **Number of MFs.** Після чого натискаємо **<Enter>.**

Крок 14. За аналогією з кроком *10* задамо наступні найменування термів змінної **y: L (Низький), LA (Нижче середнього) A (середній), HA (Вище середнього), H (Високий).** В результаті отримуємо графічне вікно, яке представлене на рис. 1.4.

Крок 15. Перейдемо в редактор бази знань **RuleEditor.** Для цього оберемо в меню **Edit** команду **Rules** або в меню **View** команду **Edit rules....**

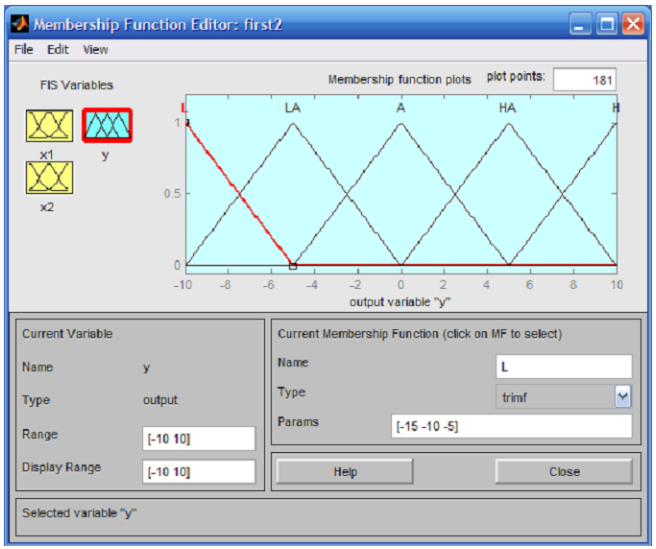


Рис. 1.4.

Крок 16. На основі візуального спостереження за графіком, який зображений на рис. 1.4, сформуємо наступні десять правил:

1. **Якщо** x1=Низький **і** x2=Низький, **тоді** y=Низький;
2. **Якщо** x1=Низький **і** x2=Високий, **тоді** y=Середній;
3. **Якщо** x1=Низький **і** x2=Вище середнього, **тоді** y=Високий;
4. **Якщо** x1=Високий **і** x2=Низький, **тоді** y=Вище середнього;
5. **Якщо** x1=Високий **і** x2=Високий, **тоді** y=Низький;
6. **Якщо** x1=Середній **і** x2=Середній, **тоді** y=Середній;
7. **Якщо** x1=Середній **і** x2=Вище середнього, **тоді** y=Вище середнього;

8. **Якщо** x1=Низький **і** x2=Нижче середнього, **тоді** y=Нижче середнього;

9. **Якщо** x1=Середній **і** x2=Вище середнього, **тоді** y=Середній.

Для введення правила необхідно обрати в меню відповідну комбінацію термів і натиснути кнопку **Add rule.** На рис.1.5 зображено вікно редактору бази знань після введення усіх 9 правил. Число в дужках в кінці кожного правила представляє собою вагові коефіцієнти відповідного правила.

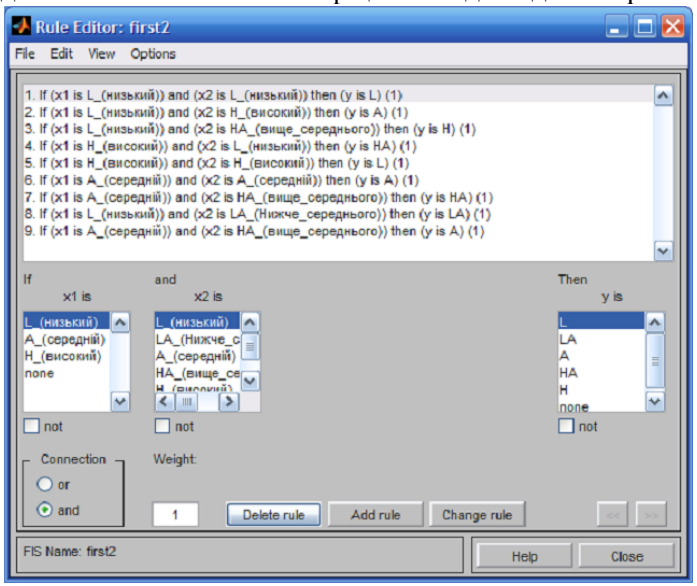


Рис. 1.5.

Потрібно звернути увагу на параметр **Weight,** який вказує вагу нечіткої впевненості в правилі. Його можна задавати в діапазоні [0, і].

Крок 17. Збережемо побудовану систему. Для цього в меню **File** в підменю **Export** оберемо команду **To disk.**

На рис. 1.6 приведено вікно візуалізацій нечіткого логічного виводу. Це вікно активується командою **View rules...** меню **View.** В полі **Input** вказуються значення вхідних змінних, для яких виконується логічний вивід. Тобто, обраховується за алгоритмом Мамдані значення вихідної змінної.

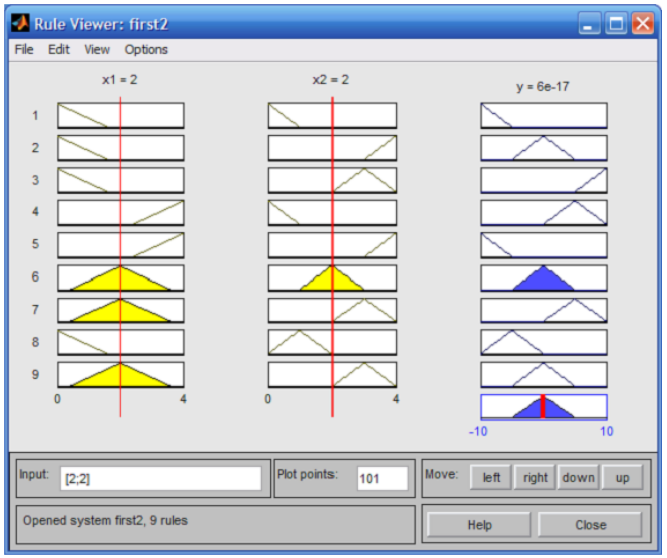


Рис 1.6.

На рис. 1.7 приведена поверхня "входи-вихід", яка відповідає синтезованій системі логічного виводу. Для виводу цього вікна необхідно використати команду **View surface...** меню **View.**

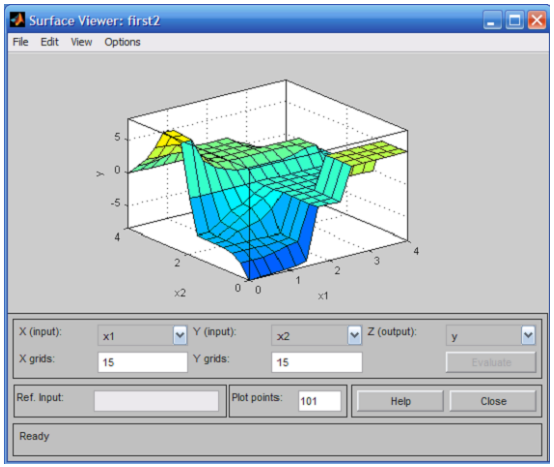


Рис. 1.7.

Отримане зображення системи «вхід-вихід» занесіть у бланк звіту (рис. 16).

Порівнюючи поверхні на рис 15 і рис. 16 звіту, зробіть висновок.

Завдання 1.12. Порівняння звичайної та нечіткої логіки програмуванням в MATLAB

Для ілюстрації значення нечіткої логіки розглянемо як лінійні, так і нечіткі підходи до наступної проблеми: Що таке правильна сума чайових для того, щоб підказати Вам як оцінити сервіс очікування у ресторані?

По-перше, розглянемо цю проблему звичайним (чітким) способом та опишемо командами MATLAB, у яких викладено лінійні та кусково-лінійні відносини.

Потім розглянемо ту ж систему, використовуючи нечітку логіку.

** Основна проблема чайових. * Дано число від 0 до 10, яке представляє якість обслуговування в ресторані (де 10 є відмінне обслуговування), що має визначати чайові. (Ця проблема заснована на оплаті чайових, як це зазвичай практикується в США. Середні чайові для їжі в ресторані США становить 15%, хоча фактично сума може змінюватись залежно від якості наданої послуги.)*

Звичайна логіка (чіткий підхід)

Почніть з найпростіших можливих відносин. Припустимо, що чайові завжди дорівнюють 15% від загального рахунку.

```
service = 0:.5:10;
tip = 0.15*ones(size(service));
plot(service,tip)
xlabel('Service')
ylabel('Tip')
ylim([0.05 0.25])
```

Графік функції належності занесіть у звіт (рис. 17).

Цей зв'язок не враховує якість послуги, тому потрібно додати новий терм до рівняння. Оскільки обслуговування оцінюється шкалою від 0 до 10, ви можете задати функцію чайових лінійно від 5%, якщо обслуговування погано, до 25%, якщо послуга відмінна. Тепер співвідношення можна подати так:


```
tip = (.20/10)*service+0.05;
plot(service,tip)
xlabel('Service')
ylabel('Tip')
ylim([0.05 0.25])
```

Графік функції належності занесіть у звіт (рис. 18).

Формула робить те, що ви хочете, щоб зробити. Однак, Ви можете захотіти, щоб чайові відображали також якість їжі. Це розширення задачі визначається наступним чином.

* Розширена проблема чайових. * Дано два набори чисел від 0 до 10 (де 10 - відмінно), які відповідно представляють якість сервісу і якість їжі в ресторані, що повинно визначати чайові.

Спробуйте запрограмувати:

```
food = 0:.5:10;
[F,S] = meshgrid(food,service);
tip = (0.20/20).*(S+F)+0.05;
surf(S,F,tip)
xlabel('Service')
ylabel('Food')
zlabel('Tip')
```

Графік функції належності занесіть у звіт (рис. 19).

У цьому випадку результати виглядають задовільно, але коли придивитися до них пильно, то вони не здаються цілком правильними. Припустимо, ви хочете, щоб обслуговування було більш важливим фактором, ніж якість харчових продуктів. Вкажіть, що на цю послугу припадає на 80% від загальної суми чайових, а продукти складають інші 20%. Спробуйте це рівняння:

```
servRatio = 0.8;
tip = servRatio*(0.20/10*S+0.05) + ...
  (1-servRatio)*(0.20/10*F+0.05);
surf(S,F,tip)
xlabel('Service')
ylabel('Food')
zlabel('Tip')
```

Графік функції належності занесіть у звіт (рис. 20).

Відповідь все ще занадто рівномірно лінійна. Припустимо, вам потрібно більше пологої функції в середині, тобто, потрібно дати 15% чайових, але ви хочете також вказати варіант, якщо обслуговування виключно добре або погане. Цей фактор, у свою чергу, означає, що попередні лінійні відображенння більше не застосовуються. Далі можна використовувати лінійний розрахунок з кусково-лінійною конструкцією. Тепер повернемося до одновимірної проблеми яка просто розглядає обслуговування. Ви можете зв'язати просте призначення умовних чайових, використовуючи логічне індексування.

```
tip = zeros(size(service));
tip(service<3) = (0.10/3)*service(service<3)+0.05;
tip(service>=3 & service<7) = 0.15;
tip(service>=7 & service<=10) = ...
  (0.10/3)*(service(service>=7 & service<=10)-7)+0.15;
plot(service,tip)
xlabel('Service')
ylabel('Tip')
ylim([0.05 0.25])
```

Графік функції належності занесіть у звіт (рис. 21).

Припустимо, ви поширюєте це на два виміри, де ви знову враховуєте і якість їжі.

```
servRatio = 0.8;
tip = zeros(size(S));
tip(S<3) = ((0.10/3)*S(S<3)+0.05)*servRatio + ...
  (1-servRatio)*(0.20/10*F(S<3)+0.05);
tip(S>=3 & S<7) = (0.15)*servRatio + ...
  (1-servRatio)*(0.20/10*F(S>=3 & S<7)+0.05);
tip(S>=7 & S<=10) = ((0.10/3)*(S(S>=7 & S<=10)-
7)+0.15)*servRatio + ...
  (1-servRatio)*(0.20/10*F(S>=7 & S<=10)+0.05);
surf(S,F,tip)
xlabel('Service')
ylabel('Food')
zlabel('Tip')
```

Графік функції належності занесіть у звіт (рис. 22).

Сюжет виглядає добре, але функція надто складна. Було трохи важко програмувати правильно правила, і це, безумовно, буде нелегко для зміни

цього коду в майбутньому. Більш того, алгоритм функціонування ще менш очевидний для того, хто не бачив оригінального дизайну процесу.

Підхід нечіткої логіки

Ви повинні охопити основи цієї проблеми, залишивши осторонь всі фактори, які можуть бути незначними. Якщо ви зробите перелік того, що дійсно має значення у цій проблемі, ви можете добитися наступних описів правил:

*Якщо послуга є поганою, то чайові є малими.

* Якщо послуга хороша, то чайові середні.

* Якщо послуга відмінна, то чайові щедрі (великі).

Порядок, у якому тут представлені правила, довільний. Це не впливає, які правила застосовуються в першу чергу. Якщо ви хочете включити ефект їжі в чайові, додайте наступні два правила:

* Якщо їжа неякісна, то чайові малі.

* Якщо їжа смачна, то чайові щедрі (великі).

Ви можете об'єднати два різних списки правил в один жорсткий список. Три правила, такі як:

* Якщо послуга є поганою або їжа неякісна, то чайові малі.

* Якщо послуга хороша, то чайові середні.

* Якщо послуга відмінна або їжа смачна, то чайові щедрі (великі).

Ці три правила є ядром вашого рішення. Випадково ви просто визначили правила для системи нечіткої логіки. Коли ви надаєте математичного значення для лінгвістичних змінних (наприклад, що є середнє значення) у вас є повна система нечіткого виводу. Методологія нечіткої логіки також повинна враховувати:

* Як комбінуються всі правила?

* Як я можу визначити математично, яке середнє значення для типу змінних?

Деталі методу не дуже сильно змінюються від задачі до задачі - механіка нечіткої логіки не важка для розуміння. Важливо те, щоб ви зрозуміли, що нечітка логіка адаптивна, проста і легко застосовувана.

Рішення задачі

Наступний графік являє собою систему нечіткої логіки, яка вирішує проблему чайових.

```
gensurf(readfis('tipper'))
```

Графік функції належності занесіть у звіт (рис.23).

Цей сюжет породжувався трьома правилами, які враховували обидва фактори обслуговування та якості їжі. Механіка того, як працює нечіткий висновок пояснюється в розділі огляду «Основи теми нечіткої логіки». В темі, Build Mamdani Systems (GUI), вся проблема чайових може бути описана за допомогою програм Fuzzy Logic Toolbox (TM).

** Зауваження **

Розглянемо деякі зауваження щодо наведеного прикладу. Ми знайшли кусочно-лінійне відношення, яке вирішило проблему. Це спрацювало, але це було проблематично, і коли ви записали його як код, це не дуже легко інтерпретувати. І навпаки, система нечіткої логіки на основі деяких твердих тверджень. Крім того, ви змогли додати ще два правила в нижній частині списку, що вплинуло на форму загальної кількості виходу без необхідності скасувати те, що вже було зроблено. Створення подальшої модифікації було відносно легким.

Крім того, за допомогою нечітких правил логіки, утримання структури алгоритму відокремлюється на досить чистих лініях. Поняття середнього чайових може змінюватися з дня на день, від міста до міста, від країни до країни, але основна логіка однакова: якщо послуга хороша, то чайові - середні.

** Відкалібрування методу **

Ви можете швидко відкалібрувати метод просто переміщенням нечіткого набору, що визначає середнє без перезапису нечітких правил логіки.

Ви можете перемикати списки кусково-лінійних функцій, але є більша ймовірність того, що повторне калібрування не буде таким швидким і простим.

У наступному прикладі кусково-лінійну проблему представлення трохи переписано, щоб зробити його більш загальним. Він виконує ту ж функцію, що і раніше, тільки зараз константи можуть бути легко змінені.

```
lowTip = 0.05;
averTip = 0.15;
highTip = 0.25;
tipRange = highTip-lowTip;
badService = 0;
okayService = 3;
goodService = 7;
greatService = 10;
```



```
serviceRange = greatService-badService;
badFood = 0;
greatFood = 10;
foodRange = greatFood-badFood;
```

Якщо послуга є поганою або їжа неякісна, чайові малі

```
if service<okayService
  tip = (((averTip-lowTip)/(okayService-badService)) ...
    *service+lowTip)*servRatio + ...
    (1-servRatio)*(tipRange/foodRange*food+lowTip);
```

Якщо послуга хороша, чайові середні

```
elseif service<goodService
  tip = averTip*servRatio + (1-servRatio)* ...
    (tipRange/foodRange*food+lowTip);
```

Якщо сервіс відмінний або їжа смачна, чайові щедрі

```
else
  tip = (((highTip-averTip)/ ...
    (greatService-goodService))* ...
    (service-goodService)+averTip)*servRatio + ...
    (1-servRatio)*(tipRange/foodRange*food+lowTip);
end
```

Як і у всіх кодах, чим більше узагальнення, що вводиться, тим менше точний алгоритм стає. Ви можете поліпшити ясність, додавши більше коментарів, або, можливо, переписавши алгоритм трохи більше самоочевидних способів. Але кусково-лінійна методологія не є оптимальним способом вирішення цієї проблеми.

Якщо видалити все з алгоритму, за винятком трьох коментарів, що залишилися, то залишаться саме нечіткі правила логіки, які ви раніше записували.

* Якщо послуга є поганою або їжа неякісна, чайові є малі.

* Якщо послуга хороша, чайові – середні.

* Якщо сервіс відмінний або їжа смачна, чайові щедрі (великі).

Завдання 1.13. Зробіть загальні висновки по лабораторній роботі

ЗВІТНІСТЬ ЗА ЛАБОРАТОРНУ РОБОТУ № 1

У звіті з лабораторної роботи необхідно представити всі графіки та висновки згідно завдання.

Назвіть звіт ШІ-КБ-ЛР-1-NNN-XXXXX.doc
де NNN – номер групи
XXXXX – позначення прізвища студента.

Переконвертуйте файл звіту в ШІ-КБ- ЛР-1-NNN-XXXXX.pdf

Надішліть звіт викладачу на електронну пошту.

ПИТАННЯ ДЛЯ САМОКОНТРОЛЮ

1. Яка множина називається «нечіткою»?
2. Чим нечітка логіка відрізняється від звичайної?
3. Які є методи побудови функції належності?
4. Призначення функції належності?
5. Які існують функції приналежності?
6. Чим відрізняється алгоритм Mamdani від алгоритма Sugeno?
7. Що таке лінгвістична змінна?
8. Що таке терм-множина?
9. Що таке фазифікація та дефазифікація змінних?

ЛАБОРАТОРНА РОБОТА № 2

МОДЕЛЮВАННЯ ПРОСТИХ НЕЙРОННИХ МЕРЕЖ

Мета роботи: *дослідження простих нейронних мереж та можливості їх застосування при рішенні задач кібербезпеки.*

ТЕОРЕТИЧНІ ВІДОМОСТІ

Для виконання лабораторної роботи необхідно вивчити теоретичний матеріал лекцій по теорії нейронних мереж та матеріал поданий у цьому підрозділі.

Нейронні мережі (NN - Neural Networks) широко використовуються для вирішення різноманітних завдань. Серед країн, що розвиваються областей застосування NN - обробка аналогових і цифрових сигналів, синтез і ідентифікація електронних ланцюгів і систем. Основи теорії і технології застосування NN широко представлені в пакеті MATLAB. У зв'язку з цим особливо слід відзначити останню версію пакета - MATLAB 6.0, де вперше представлений GUI (Graphical User Interface - графічний інтерфейс користувача) для NN - NNTool.

Прикладами застосування технології нейронних мереж для цифрової обробки сигналів є: фільтрація, оцінка параметрів, детектування, ідентифікація систем, розпізнавання образів, реконструкція сигналів, аналіз часових рядів і стиснення. Згадані види обробки застосовні до різноманітних видів сигналів: звукових, відео, мовним, зображення, передачі повідомлень, геофізичних, локаційним, медичних вимірювань (кардіограми, енцефаллограмми, пульс) і іншим.

В даній статті дано опис NNTool і показана техніка його застосування в ряді завдань синтезу ланцюгів і цифрової обробки сигналів.

Обробка сигналів в технологіях NN виконується за допомогою яких NN без пам'яті, або NN с пам'яттю. І в тому і в іншому випадках ключовим елементом є NN без пам'яті. Подібна роль визначається тією обставиною, що при використанні нейронів з певними функціями активації (передавальними характеристиками) NN є універсальним аппроксіматором. Останнє означає, що в заданому діапазоні зміни вхідних змінних NN може з заданою точністю відтворювати (моделювати) довільну безперервну функцію цих змінних.

Нижче обговорюються питання, що відносяться до так званим NN прямого поширення, тобто без зворотних зв'язків. Чудовим властивістю таких NN є їх стійкість.

Після того як структура NN обрана, повинні бути встановлені її параметри. Вибір структури NN і типів нейронів - самостійний і вельми непросте питання, який тут ми обговорювати не будемо. Що ж стосується значень параметрів, то, як правило, вони визначаються в процесі вирішення деякої оптимізаційної задачі. Ця процедура в теорії NN називається навчанням.

Графічний інтерфейс користувача NNTool дозволяє вибирати структури NN з великого переліку і надає множину алгоритмів навчання для кожного типу мережі.

У роботі розглянуті наступні питання, що стосуються роботи з NNTool:

- призначення графічних керуючих елементів;
- підготовка даних;
- створення нейронної мережі;
- навчання мережі;
- прогін мережі.

Всі етапи роботи з мережами проілюстровані прикладами розв'язання простих завдань. Передбачається, що студент знайомий з основами теорії NN і її термінологією.

Керуючі елементи NNTool

Щоб запустити NNTool, необхідно виконати однойменну команду в командному вікні MATLAB:

```
>> nntool
```

Після цього з'явиться головне вікно NNTool, іменоване "Вікном управління мережами і даними" (Network / Data Manager) (рис. 2.1).

Панель "Мережі і дані" (Networks and Data) має функціональні клавіші з наступними призначеннями:

- Допомога (Help) - короткий опис керуючих елементів цього вікна;
- Нові дані (New Data ...) - виклик вікна, що дозволяє створювати нові набори даних;
- Нова мережа (New Network ...) - виклик вікна створення нової мережі;
- Імпорт (Import ...) - імпорт даних з робочого простору MATLAB в простір змінних NNTool;

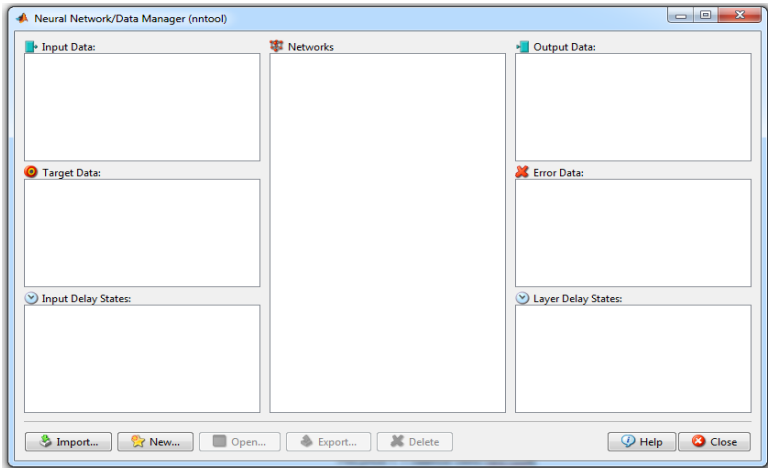


Рис. 2.1.

- Експорт (Export ...) - експорт даних з простору змінних NNTool в робочий простір MATLAB;
- Вид (View) - графічне відображення архітектури обраної мережі;
- Видалити (Delete) - видалення обраного об'єкта.

На панелі "Тільки мережі" (Networks only) розташовані клавіші для роботи виключно з мережами. При виборі покажчиком миші об'єкта будь-якого іншого типу, ці кнопки стають неактивними.

При роботі з NNTool важливо пам'ятати, що клавіші View, Delete, Initialize, Simulate, Train і Adapt (зображені на рис. 2.1 як неактивні) діють стосовно того об'єкту, який зазначений у даний момент виділенням. Якщо такого об'єкта немає, або над виділеним об'єктом неможливо зробити вказане дію, відповідна клавіша неактивна.

ЗАВДАННЯ НА ЛАБОРАТОРНУ РОБОТУ № 2 ТА МЕТОДИЧНІ РЕКОМЕНДАЦІЇ ДО ЇХ ВИКОНАННЯ

Завдання 2.1. Створити нейронну мережу, що виконує логічну функцію "І".

Створення мережі

Виберемо мережу, що складається з одного персептрона з двома входами. У процесі навчання мережі на її входи подаються вхідні дані і про-

водиться зіставлення значення, отриманого на виході, з цільовим (бажаним). На підставі результату порівняння (відхилення отриманого значення від бажаного) обчислюються величини зміни ваг і зсуву, що зменшують це відхилення.

Отже, перед створенням мережі необхідно заготовити набір навчальних і цільових даних. Складемо таблицю істинності для логічної функції "І", де Р1 і Р2 - входи, а А - бажаний вихід (табл. 2.1).

Таблиця 2.1. Таблиця істинності логічної функції "І"

P1	P2	A
0	0	0
0	1	0
1	0	0
1	1	1

Щоб задати матрицю, що складається з чотирьох векторів-рядків, скористаємося кнопкою New Data. У вікні слід провести зміни, показані на рис. 2.2, і натиснути клавішу "Створити" (Create).

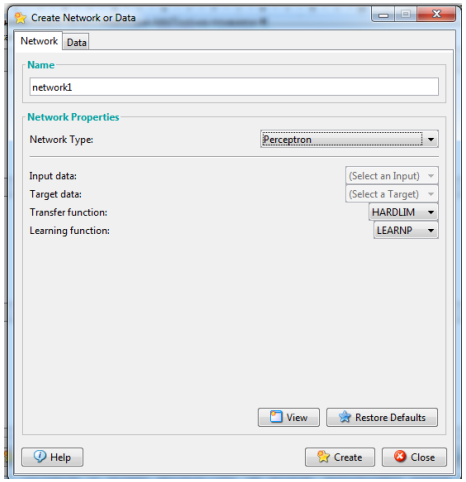


Рис. 2.2.

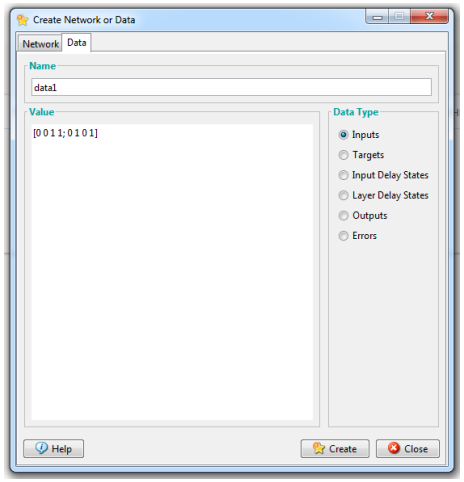


Рис. 2.3

Перейдіть у закладку Data та введіть вхідні дані, як показано на рис. 2.3.

Дані в полі "Значення" (Value) можуть бути представлені будь-яким зрозумілим для MATLAB виразом. Наприклад, попереднє визначення вектора цілей можна еквівалентно замінити рядком виду

bitand ([0 0 1 1], [0 1 0 1]).

Після натискання на Create в розділі Input Data основного вікна з'явиться вектор data1.

Зображення вікна з даними занесіть у бланк звіту (рис.1)

Введіть вектор цілей, як показано на рис. 2.4.

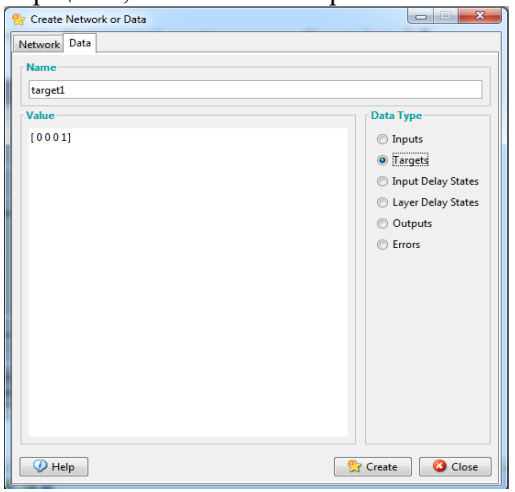


Рис. 2.4

Після натискання на Create в розділі Targets з'явиться вектор target1.

Зображення вікна з цілями занесіть у бланк звіту (рис.2)

Перейдемо до закладки Network

Поля цього вікна несуть наступні смислові навантаження:

• Ім’я мережі (Назва мережі) - це ім'я об'єкта створеної мережі.

• Тип мережі (Network Type) - визначає тип мережі та контекст вибору типу, що вказує на різні параметри в частині, розташованій нижче цього пункту. Таким чином, для різних типів сітки вікно змінює своє зміст.

• Вхідні діапазони (діапазони вхідних даних) - матриця з числом рядків, рівних числу входів. Кожен рядок представляє собою вектор з двома елементами: перший - мінімальне значення сигналу, який буде поданий на відповідний вхід в мережу за умов навчання, другого - максимального. Для того, щоб уникнути цих знань забезпечити випуск списку "Вибрати з

входу", можна автоматично змінити необхідні дані, вказуючи на вхідний змінні.

• Кількість нейронів (Кількість нейронів) - число нейронів у шарі.

• Передаточна функція (Функція передачі) - в цьому пункті вибирається передаточна функція (функція активації) нейронів.

• Функція навчання (функція навчання) - функція, що відповідає за оновлення і зміну мереж в процесі підготовки.

За допомогою кнопки "Вид" можна поглянути на архітектуру створюваної мережі (рис. 2.5). Так, ми можемо переконатись, що всі дії були зроблені вірно. На рис. 2.5 зображена персональна мережа з вихідним блоком, реалізовуючи передаточну функцію з жорстким обмеженням. Кількість нейронів в одному рівні, що символічно відображає величину візуально-стовбурової тканини на вищому рівні і вказує на кількість безпосередньо під блоком передаточної функції. Мережа має два входи, так як розмір стовпчика вхідного вікна – рівний двом.

Отже, структура мережі відповідає нашим завданням. Тепер можна закрити вікно попереднього перегляду, натиснувши клавішу "Закрити" (Close), і підтвердити намір створити мережу, натиснувши "Створити" (Create) в вікні створення мережі.

В результаті виконаних операцій в розділі "Мережі" (Networks) головного вікна NNTool з'явиться об'єкт з ім'ям network1.

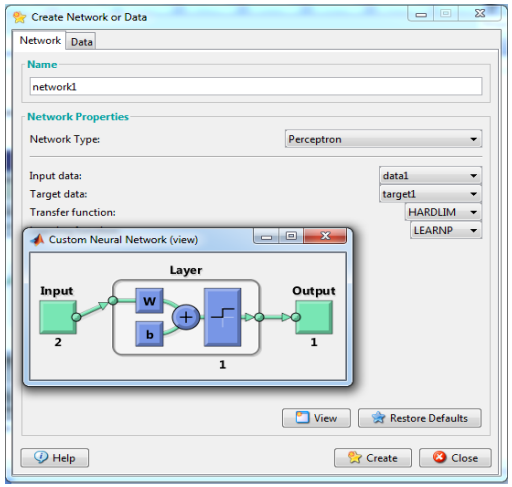


Рис. 2.5

Зображення вікна з архітектурою створюваної мережі занесіть у бланк звіту (рис.3)

Навчання мережі

Наша мета - побудувати нейронну мережу, яка виконує функцію логічного "І". Очевидно, не можна розраховувати на те, що відразу після етапу створення мережі остання буде забезпечувати правильний результат (правильне співвідношення "вхід/вихід"). Для досягнення мети мережу необхідно належним чином навчити, тобто підібрати відповідні значення параметрів. В MATLAB реалізовано більшість відомих алгоритмів навчання нейронних мереж, серед яких представлено два для персептронів мереж розглянутого виду. Створюючи мережу, ми вказали LEARNP в якості опції, що реалізує алгоритм навчання (рис. 2.5).

Повернемося до головного вікна NNTool.

Відзначивши покажчиком миші об'єкт мережі network1, натискаємо два рази на ньому, чим викличемо вікно управління мережею (рис. 2.6).

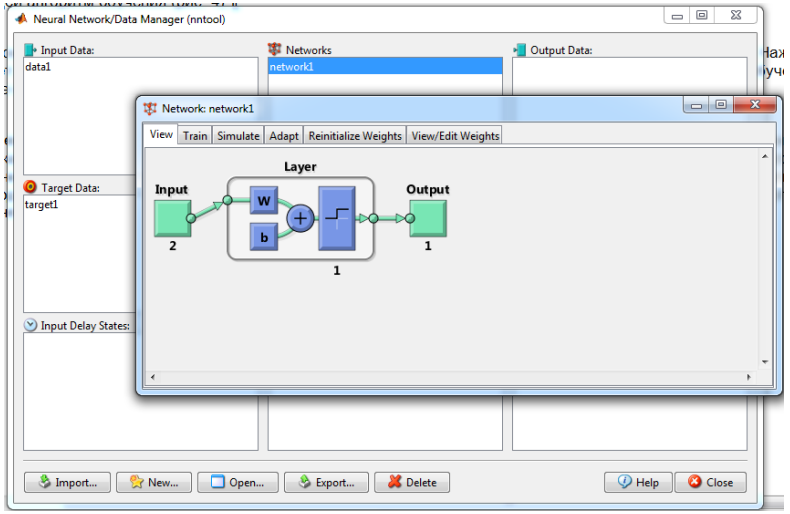


Рис. 2.6

Натискаємо вкладку Train (рис. 2.7).

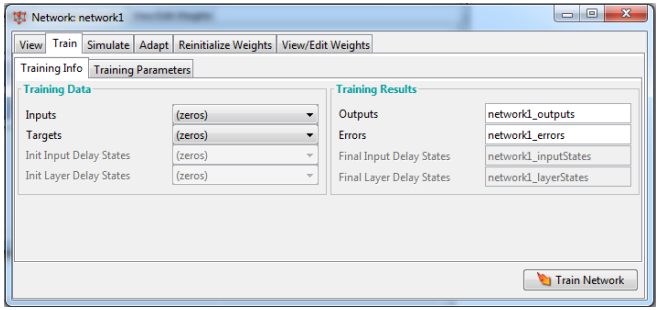


Рис. 2.7

Перед нами виникне вкладка "Train" вікна властивостей мережі, що містить, в свою чергу, ще одну панель вкладок (рис. 2.7). Їх головне призначення - управління процесом навчання. На вкладці "Інформація навчання" (Training info) потрібно вказати набір навчальних даних в поле "Входи" (Inputs) і набір цільових даних в поле "Цілі" (Targets). Поля "Виходи" (Outputs) і "Помилки" (Errors) NNTool заповнює автоматично. При цьому результати навчання, до яких відносяться виходи і помилки, будуть зберігатися в змінних з зазначеними іменами (рис. 2.8).

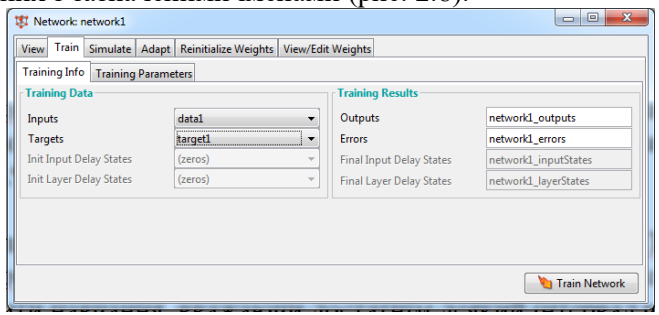


Рис. 2.8

Завершити процес навчання можна, керуючись різними критеріями. Можливі ситуації, коли бажано зупинити навчання, вважаючи достатнім деякий інтервал часу. З іншого боку, об'єктивним критерієм є рівень помилки.

На вкладці "Опції Установки навчання" (Training parameters) для нашої мережі (рис. 2.9) можна встановити наступні поля:

• Кількість епох (epochs) - визначає число епох (інтервал часу), по закінченні яких навчання буде припинено.

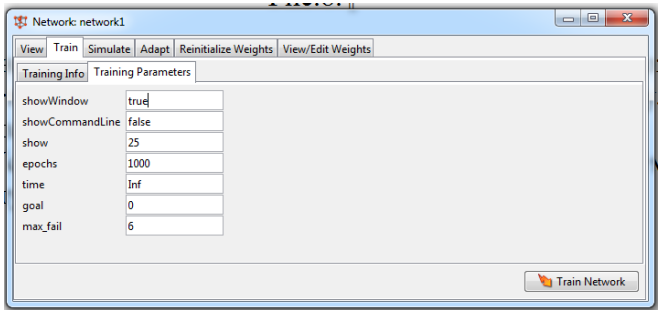


Рис. 2.9

• Епохою називають одноразове передання всіх навчальних вхідних даних на входи мережі.

• Досягнення мети або потрапляння (goal) - тут задається абсолютна величина функції помилки, при якій мета буде вважатися досягнутою.

• Період оновлення (show) - період оновлення графіка кривої навчання, виражений числом епох.

• Час навчання (time) - після закінчення зазначеного в ньому тимчасового інтервалу, вираженого в секундах, навчання припиняється.

Беручи до уваги той факт, що для задач з лінійно роздільними множинами (а наше завдання відноситься до цього класу) завжди існує точне рішення, встановимо поріг досягнення мети, рівний нулю. Значення інших параметрів залишимо за замовчуванням. Зауважимо тільки, що поле часу навчання містить запис Inf, який визначає нескінченний інтервал часу (від англійського Infinite - нескінченний).

Повернемося до вкладки навчання (Train) - "Інформація навчання" (Training info). Щоб почати навчання, потрібно натиснути кнопку "Навчити мережу" (Train Network).

З’являється вікно навчання мережі (рис. 2.10).

Після цього, якщо в поточний момент мережа не задовольняє жодному з умов, зазначених в розділі параметрів навчання (Training Parameters), можна визвати вікно, що ілюструє динаміку цільової функції - криву навчання з блоку вікна Plots. За допомогою кнопки "Зупинити навчання" (Stop Training) можна припинити цей процес. З малюнка видно, що навчання було зупинено, коли функція мети досягла встановленої величини (goal = 0).

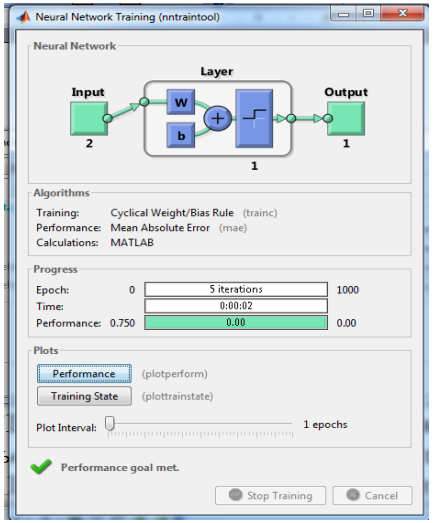


Рис. 2.10

Слід зазначити, що для персептронів, що мають функцію активації з жорстким обмеженням, помилка розраховується як різниця між ціллю та отриманим виходом.

Отже, алгоритм навчання знайшов точне рішення задачі. У методичних цілях переконаємося в правильності рішення задачі шляхом прогону навченої мережі. Для цього необхідно відкрити вкладку "Прогін" (Simulate - моделювати) і вибрати в списку, що випадає "Входи" (Inputs) заготовлені дані. У цьому завданню природно використовувати той же набір даних, що і при навчанні data1. При бажанні (рис. 2.11) можна встановити прапорець "Задати мету" (Supply Targets). Тоді в результаті прогону додатково будуть розраховані значення помилки.

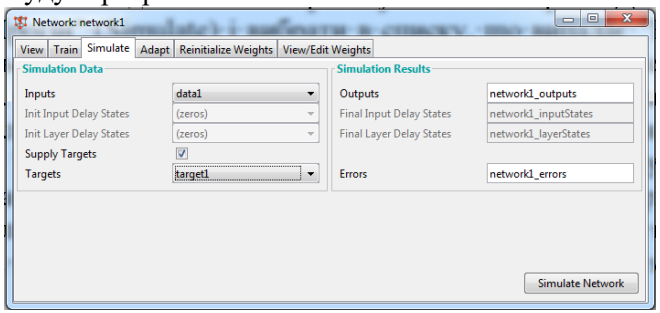


Рис. 2.11

Натискання кнопки "Прогін мережі" (Simulate Network) запише результати прогону в змінну, ім'я якої зазначено в поле "Виходи" (Outputs). Тепер можна повернутися в основне вікно NNTool і, виділивши мишею вихідну змінну network1, натиснути два рази – з'явиться (рис. 2.12) вікно (View). Вміст вікна перегляду збігається зі значенням вектора цілей - мережа працює правильно.

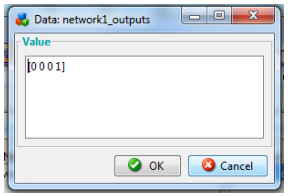


Рис. 2.12

Зображення вікна з вихідною змінною network1 занесіть у бланк звіту (рис.4)

Слід зауважити, що мережа створюється ініціалізованою, тобто значення ваг і зміщень задаються певним чином. Перед кожним наступним досвідом навчання зазвичай початкові умови оновлюються, для чого на вкладці "Ініціалізація" (Initialize) передбачена функція ініціалізації. Так, якщо потрібно провести кілька незалежних дослідів навчання, ініціалізація ваг і зміщень перед кожним з них здійснюється натисканням кнопки "Ініціалізувати ваги" (Initialize Weights).

При виборі нейронної мережі для вирішення конкретного завдання важко передбачити її порядок. Якщо вибрати невиправдано великий порядок, мережа може виявитися занадто гнучкою і може подати просту залежність складним чином. Це явище називається перенавчанням. У разі мережі з недостатньою кількістю нейронів, напроти, необхідний рівень помилки ніколи не буде досягнутий. Тут у наявності надмірне узагальнення.

Для попередження перенавчання застосовується наступна техніка. Дані поділяються на дві множини: навчальну (Training Data) і контрольну (Validation Data). Контрольна множина в навчанні не використовується. На початку роботи помилки мережі на навчальній та контрольній множинах будуть однаковими. У міру того, як мережа навчається, помилка навчання убуває, і, поки навчання зменшує дійсну функцію помилки, помилка на контрольній множині також буде спадати. Якщо ж контрольна помилка перестала зменшуватися або навіть стала рости, це вказує на те, що

навчання слід закінчити. Зупинка на цьому етапі називається ранньою зупинкою (Early stopping).

Таким чином, необхідно провести серію експериментів з різними мережами, перш ніж буде отримана оптимальна. При цьому щоб не бути введеним в оману локальними мінімумами функції помилки, слід кілька разів навчати кожну мережу.

Якщо в результаті послідовних кроків навчання і контролю помилка залишається неприпустимо великою, доцільно змінити модель нейронної мережі (наприклад, ускладнити мережу, збільшивши число нейронів, або використовувати мережу іншого виду). У такій ситуації рекомендується застосовувати ще одну множину - тестову множину спостережень (Test Data), яка представляє собою незалежну вибірку з вхідних даних. Підсумкова модель тестується на цій множині, що дає додаткову можливість переконатися в достовірності отриманих результатів. Очевидно, щоб зіграти свою роль, тестова множина повинна бути використана тільки один раз. Якщо її використовувати для коригування мережі, вона фактично перетвориться на контрольну множину.

Навчання мережі можна проводити в різних режимах. У зв'язку з цим, в NNTool передбачено дві вкладки, що представляють навчальні функції: розглянута раніше вкладка Train і "Адаптація" (Adapt). Adapt (рис. 2.13) вміщує вкладку інформації адаптації (Adaption Info), на якій містяться поля, схожі за своїм призначенням з полями вкладки Training Info і виконують ті ж функції і вкладку параметри адаптації (Adaption Parameters). Остання містить єдине поле "Проходи" (passes). Значення, вказане в цьому полі, визначає, скільки разів все вхідні вектори будуть представлені мережі в процесі навчання.

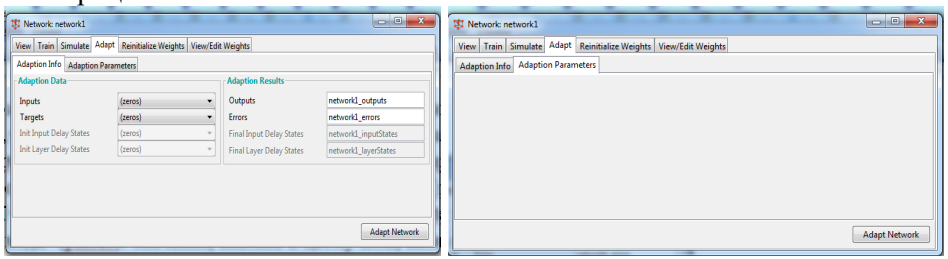


Рис. 2.13

Параметри вкладок "Train" і "Adapt" в MATLAB використовуються функціями train і adapt, відповідно. Детальна довідкова інформація по цих функцій наведено в [2].

Зробіть висновок про точність та збіжність мережі. Висновок занесіть у звіт.

Завдання 2.2. Створити нейронну мережу, що виконує логічну функцію "АБО "

Розглянемо таблицю істинності цієї функції (табл. 2.2).

Таблиця 2.2. Таблиця істинності логічної функції "АБО"

P1	P2	A
0	0	0
0	1	1
1	0	1
1	1	1

По аналогії з пунктом завдання 2.1 навчити мережу для нових вхідних та вихідних даних. Результати занести у звіт (рис. 5-8 звіту). Зробіть висновки чи змінилася структура мережі?

Завдання 2.3. Створити нейронну мережу, що виконує логічну функцію "виключне АБО "

Поділ лінійно-нероздільних множин

Розглянута задача синтезу логічного елемента "І" може трактуватися як завдання розпізнавання лінійно роздільних множин. На практиці ж частіше зустрічаються завдання поділу лінійно нероздільних множин, коли застосування персептронів з функцією активації з жорстким обмеженням не дасть рішення. У таких випадках слід використовувати інші функції активації.

Показовим прикладом лінійно невіддільною завдання є створення нейронної мережі, що виконує логічну функцію "виключне АБО".

Розглянемо таблицю істинності цієї функції (табл. 2.3).

Таблиця 2.3. Таблиця істинності логічної функції "виключне АБО"

P1	P2	A

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

Що ж мається на увазі під "лінійною нероздільністю" множин? Щоб відповісти на це питання, покажемо множину вихідних значень в просторі входів (рис. 2.14), слідуючи наступним правилом: поєднання входів P1 і P2, при яких вихід A звертається в нуль, позначаються кружком, а ті, при яких A звертається в одиницю - хрестиком.

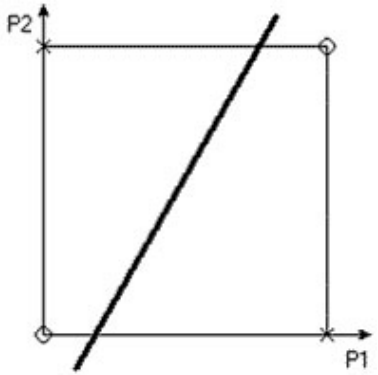


Рис 2.14. Стани логічного елемента "виключне АБО"

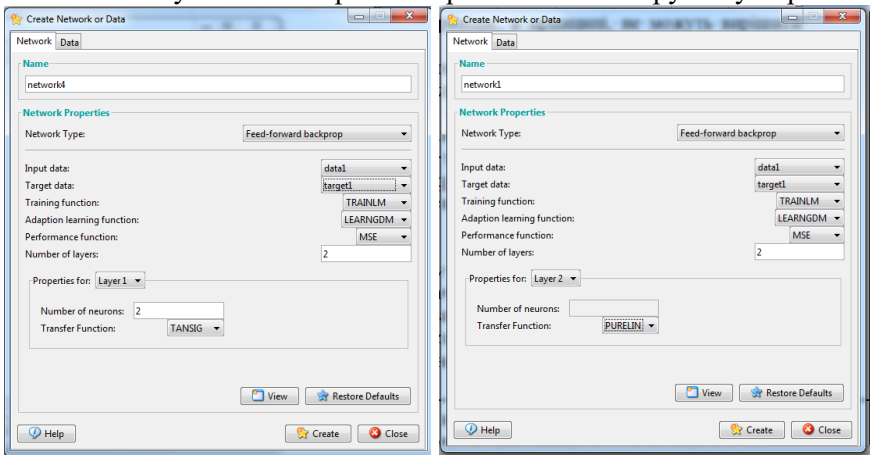
Наша мета - провести межу, що відокремлює множину нулів від множини хрестиків. З побудованої картини на рис. 2.14 видно, що неможливо провести пряму лінію, яка б відокремила нулі від одиниць. Саме в цьому сенсі множина нулів лінійно невіддільна від множини одиниць, і персептрони, розглянуті раніше, в принципі, не можуть вирішити розглянуту задачу.

Якщо ж використовувати персептрони зі спеціальними нелінійними функціями активації, наприклад, сігмоїдною, то рішення задачі можливо.

Введіть данні та цілі згідно таблиці 2.3.

Виберемо персептрон з двома нейронами прихованого шару, у яких функції активації сігмоїдні, і одним вихідним нейроном з лінійною функцією активації (рис. 2.15). В якості опції помилки вкажемо MSE (Mean

Square Error - середній квадрат помилки). Нагадаємо, що функція помилки встановлюється у вікні "Створення мережі" після вибору типу мережі.



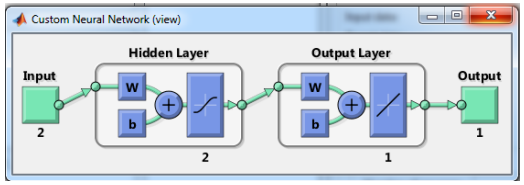


Рис. 2.15

Ініціалізуйте мережу, натиснувши на ній у основному вікні, після чого навчіть її (рис. 2.16), вказавши в якості вхідних значень сформовану раніше змінну data1, в якості цілей - новий вектор, відповідний бажаним виходам.

В процесі навчання мережа не може забезпечити точного рішення, тобто звести помилку до нуля. Однак виходить наближення, яке можна спостерігати по кривій навчання на рис. 2.17.

Слід зазначити, що дана крива може змінювати свою форму від експерименту до експерименту, але, в разі успішного навчання, характер функції буде монотонно убуває.

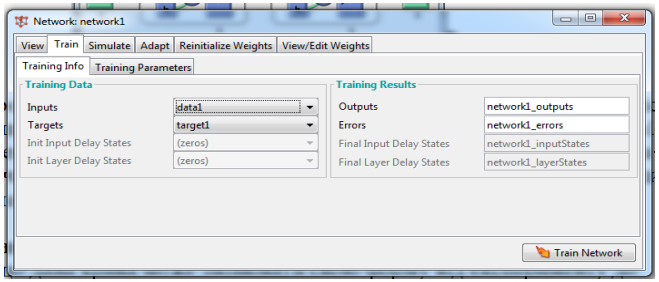


Рис. 2.16

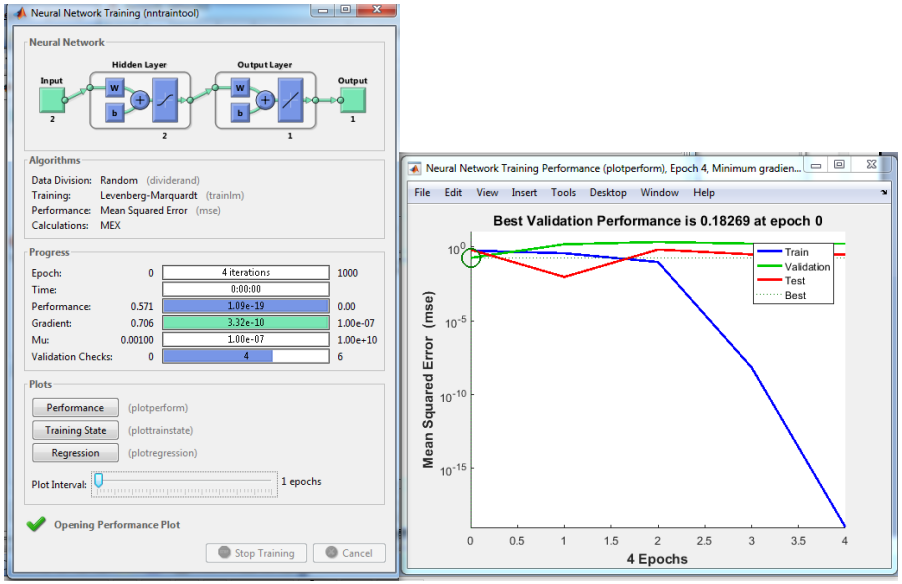


Рис. 2.17

Зображення вікон навчання та помилок навчання (такі як на рис. 2.17) занесіть у звіт рис. 9 та 10 звіту.

В результаті навчання, помилка була мінімізована до вельми малого значення, яке практично можна вважати рівним нулю.

Проведемо симуляцію та отримаємо вихідний вектор (рис. 2.18)

Перевірте правильність функціонування мережі, для цього два рази натисніть на вектор вихідних даних у вікні Output Data та вектор цілей у вікні Target Data. ***Отримані вікна подібні до рис. 2.19 занесіть у звіт (рис. 11-12 звіту)***

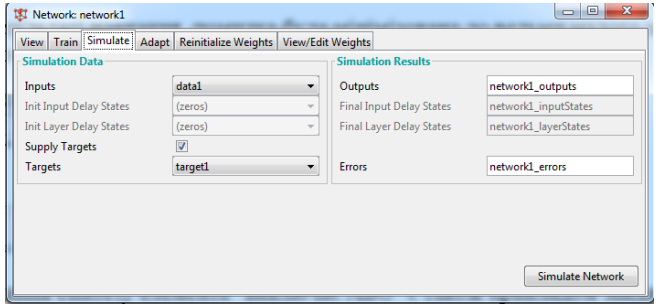


Рис. 2.18

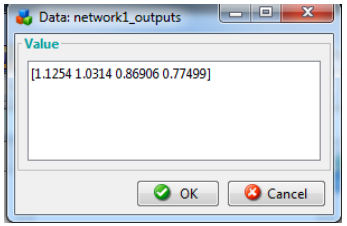


Рис. 2.19

Порівняйте значення у цих вікнах та зробіть висновок про правильність (адекватність) та точність функціонування мережі. Висновок запишіть у звіт.

Завдання 2.4. Створити нейронну мережу, що виконує функцію

Створити нейронну мережу, що виконує функцію $y = x1^2 + x2$, якщо задані послідовності входу P = [1 0.5 0 1; -2 0 0.5 1]

і цілі T = [-1 0.25 0.5 2].

Використовувати нейромережі з параметрами заданими в таблиці 2.4 для варіантів відповідно до номера студента за журналом.

Рекомендації до виконання

Сформуємо послідовності входів і цілей в робочій області GUI-інтерфейсу NNTool, використовуючи вікно **Create New Data**.

Таблиця 2.4 - Параметри нейромереж для варіантів

Номер варіанта	Багатошаровий персептрон		назва функції активації
	Кількість шарів	Кількості нейронів у шарах	
1	2	3-1	логістична сигмоїдна
2	2	2-1	тангенційна сигмоїдна
3	3	3-3-1	логістична сигмоїдна
4	2	5-1	тангенційна сигмоїдна
5	3	2-2-1	логістична сигмоїдна
6	2	10-1	тангенційна сигмоїдна
7	2	5-1	логістична сигмоїдна
8	3	5-5-1	тангенційна сигмоїдна
9	3	3-5-1	логістична сигмоїдна
10	2	4-1	тангенційна сигмоїдна
11	3	4-4-1	логістична сигмоїдна
12	3	3-4-1	тангенційна сигмоїдна
13	2	6-1	логістична сигмоїдна
14	3	6-3-1	тангенційна сигмоїдна
15	3	8-3-1	логістична сигмоїдна
16	2	7-1	тангенційна сигмоїдна
17	3	7-7-1	логістична сигмоїдна
18	3	7-4-1	тангенційна сигмоїдна
19	2	8-1	логістична сигмоїдна
20	3	8-8-1	тангенційна сигмоїдна

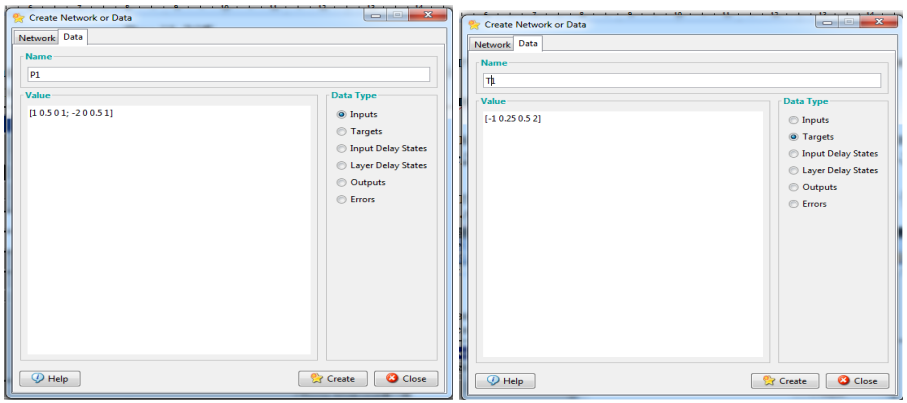


Рис. 2.19

Виберемо нейронну мережу типу **feed-forward backprop** з прямою передачею сигналу і зі зворотним поширенням помилки згідно свого варіанта. Наприклад схема мережі показана на рис. 2.20.

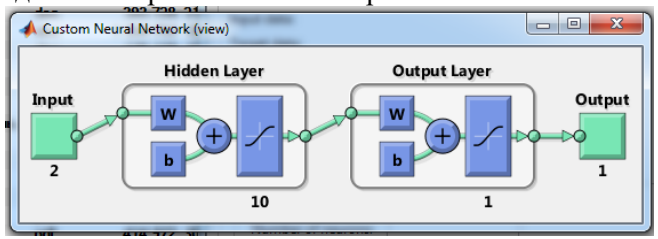


Рис. 2.20

Виконаємо ініціалізацію мережі, для чого виберемо закладку Initialize, відкриється діалогова панель, показана на рис. 2.21. Діапазони значень вихідних даних виберемо по входах з спадаючого меню **Get from input**. Для введення встановлених діапазонів і ініціалізації ваг треба скористатися кнопками **Set Ranges** (Встановити діапазони) і **Initialize Weights** (Ініціалізувати ваги). Якщо потрібно повернутися до колишніх діапазонах, то слід вибрати кнопки **Revert Ranges** (Повернути діапазони) і **Revert Weights** (Повернути ваги).

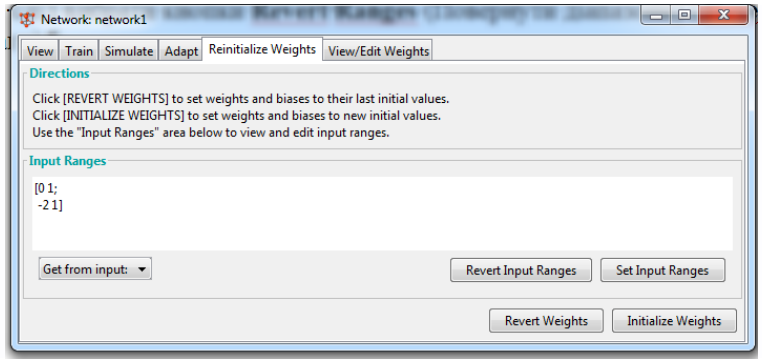


Рис. 2.21

Потім виконується навчання мережі, для чого вибирається закладка **Train** і відкривається діалогова панель.

Панель має дві закладки:

Training Info (Інформація про навчальні послідовності);

Training Parametrs (Параметри навчання);

Застосовуючи ці закладки, можна встановити імена послідовностей входу і цілі, а також параметрів процедури навчання. Тепер можна приступити до навчання мережі (кнопка **Train Network**).

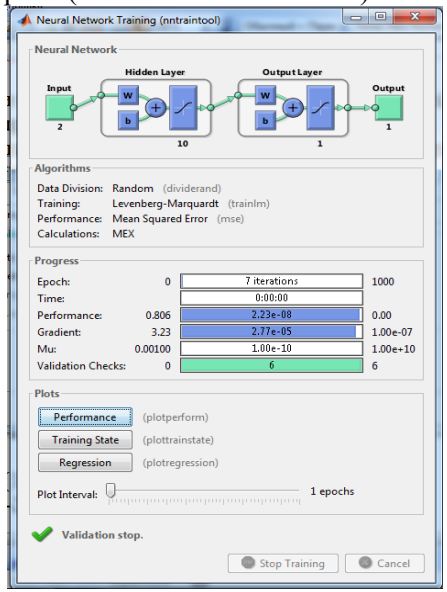


Рис. 2.22

Якість навчання мережі з прямою передачею сигналу на обраній навчальної послідовності пояснюється на рис. 2.23. Практично нульова точність досягається за 7 циклів навчання.

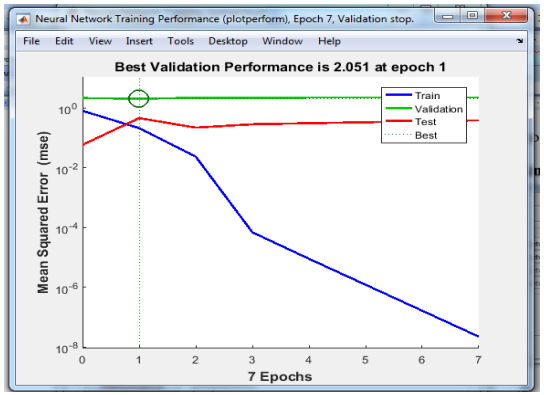


Рис. 2.23

Зображення вікон навчання та помилок навчання (такі як на рис.2.22-2.23) занесіть у звіт рис. 13 та 14 звіту.

Відповідні ваги і зміщення можна побачити, якщо вибрати закладку Weights

Для зручності роботи можна експортувати створену нейронну мережу в робочу область системи MATLAB і отримати інформацію про ваги і зсувах безпосередньо в робочому вікні системи:

```
network1.IW {1, 1}, network1.b {1}
ans = -1.9390 -2.2747
ans = 1.1389
network1.LW {2, 1}, network1.b {2}
ans = -1.5040
ans = 0.5024
```

Перевірте правильність функціонування мережі, для цього два рази натисніть на вектор вихідних даних у вікні Output Data та вектор цілей у вікні Target Data. ***Отримані вікна подібні до рис. 2. 24 занесіть у звіт (рис. 15-16 звіту)***

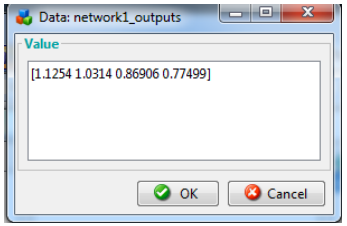


Рис. 2.24

Порівняйте значення у цих вікнах та зробіть висновок про правильність (адекватність) та точність функціонування мережі. Висновок запишіть у звіт.

Завдання 2.5. Програмування feedforward нейронної мережі, у командному режимі без використання графічного інтерфейсу MATLAB

Складіть та виконайте програму по класифікації даних

% Задамо першу множину значень кластера №1.

% Центром цього кластера є точка (2; 0) з розкидом від центру rand (1,50) і кількістю відліків 50.
% T.n x1 і y1 буде масивом розмірність 1x50

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

% Також задамо другу множину кластера №2 з центром (-2;0)

```
x2=-2+rand(1,50)
y2=0+rand(1,50)
plot(x2,y2,'ob')
% Задамо третю множину значень кластера №3 з центром (0;2)
x3=0+rand(1,50)
y3=2+rand(1,50)
plot(x3,y3,'oy')
```

% Задамо четверту множину значень кластера №4 з центром (0;2)

```
x4=0+rand(1,50)
y4=-2+rand(1,50)
plot(x4,y4,'og')
% Побудуємо графік figure (1)
figure (1)
hold on
plot(x1,y1,'or')
plot(x2,y2,'ob')
plot(x3,y3,'oy')
plot(x4,y4,'og')
grid on
hold off
```

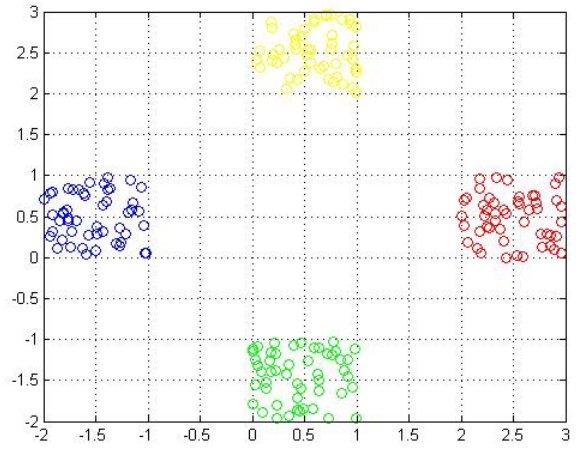



Рис. 2.25

Отриманий графік подібний до рис. 2.25 занесіть у звіт (рис. 17 звіту)

% Задамо цілі чотирьох кластерів, тобто яке значення буде видавати
% Нейронна мережа при симуляції

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

% Поєднаємо всі цілі кластерів

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

% З'єднаємо всю базу знань для нейронної системи в одну матрицю,
% А саме поєднаємо x

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

% поєднаємо y

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

```
y(151:200)=y4;

% і все це з'єднаємо в z

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

% перевіримо отримані розміри

size(T)
size(z)

% Створимо сруктуру feedforward нейронної мережі, а саме створимо
% трьохслойну нейронну систему з 10-ю нейронами на першому шарі, 2-мя
% нейронами на другому шарі і 1-ним нейроном на третьому шарі.
% z - база знань спектра
% T - ціль мережі
% logsig, tansig, purelin - вид активації шару
% 1-ый слой logsig, 2-ой слой logsig, 3-ий слой purelin

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

% Навчимо створену мережу
net = train(net,z,T);
```

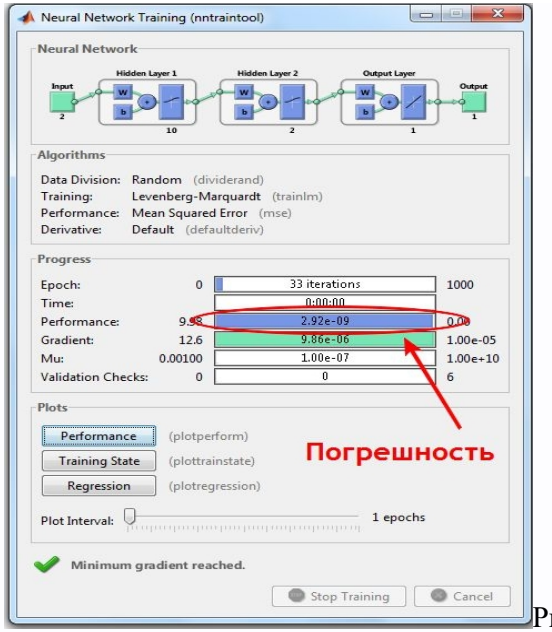



Рис. 2.26

Отримане вікно подібне до рис. 2.26 занесіть у звіт (рис. 18 звіту)

```
% І перевіримо мережу на спектрах за допомогою яких ми навчали мережу

% Перевірка першого кластера
a=sim(net,z(:,1:50))
%100% результат
% Перевірка другого кластера
a=sim(net,z(:,51:100))
%100% результат
% Перевірка третього кластера
a=sim(net,z(:,101:150))
%100% результат
% Перевірка четвертого кластера
a=sim(net,z(:,151:200))
%100% результат
```

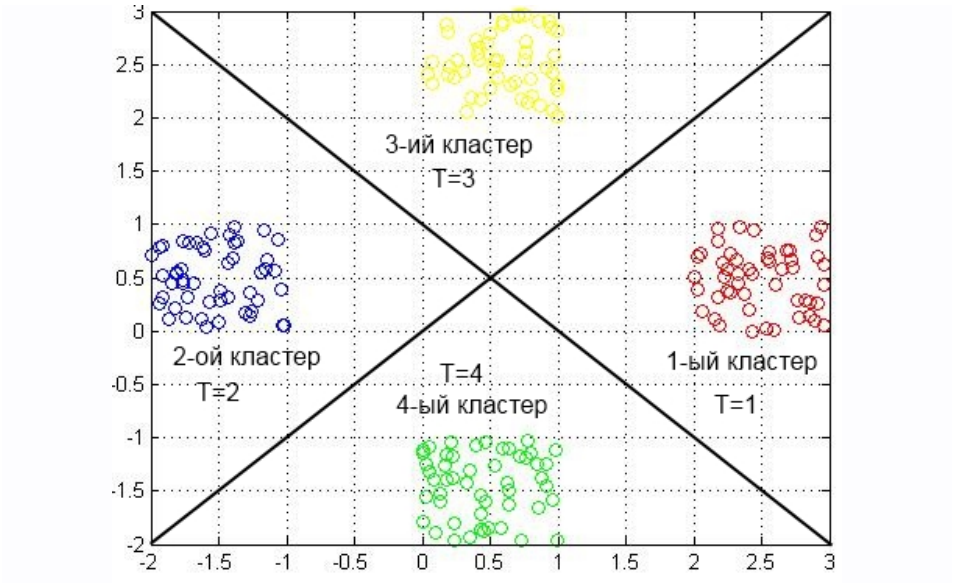


Рис. 2.28

Максимальна помилка 0.0002

Також можете перевірити систему на інших реалізаціях, результат системи буде залежати від відстані між вектором, який ви подаєте і центром кластера.

Текст програми занесіть у звіт

Зробіть висновки по роботі у яких укажіть:
- призначення функції активації;
- про можливості нейронних мереж прямого поширення.

ЗВІТНІСТЬ ЗА ЛАБОРАТОРНУ РОБОТУ № 2

У звіті з лабораторної роботи необхідно представити всі графіки та висновки згідно завдання.

Назвіть звіт ШІ-КБ-ЛР-2-NNN-XXXXX.doc
де NNN – номер групи

XXXXX – позначення прізвища студента.

Переконвертуйте файл звіту в ШІ-КБ-ЛР-2-NNN-XXXXX.pdf

Надішліть звіт викладачу на електронну пошту.

ПИТАННЯ ДЛЯ САМОКОНТРОЛЮ

1. Поняття: нейрон, нейромережа, нейрокомп'ютер, нейроінформатика.
2. Класифікація та види моделей нейромереж.
3. Властивості штучних нейромереж.
4. Загальне уявлення про навчання нейромереж.
5. Загальна характеристика та принципи побудови нейромереж.
6. Характеристики процесу навчання нейромереж.
7. Лінійна роздільність і лінійна нероздільність класів.
8. Порівняння моделей та алгоритмів навчання нейромереж прямого поширення.
9. Нейронні мережі у пакеті MATLAB. Модуль Neural Network Toolbox.

ЛАБОРАТОРНА РОБОТА № 3

ДОСЛІДЖЕННЯ НЕЙРОННІХ МЕРЕЖ КОХОНЕНА ТА LVQ

Мета роботи: *вивчити моделі та властивості нейронних мереж Кохонена і LVQ та можливості їх застосування при рішенні задач кібербезпеки; порівняти можливості ШНМ Кохонена з можливостями дискретного одношарового персептрона; ознайомитися з програмними засобами, що моделюють ШНМ Кохонена.*

ТЕОРЕТИЧНІ ВІДОМОСТІ

Для виконання лабораторної роботи необхідно вивчити теоретичний матеріал лекцій по теорії штучних нейронних мереж та матеріал поданий у цьому підрозділі.

Нейронна мережа SOM

Карта ознак самоорганізації Кохонена (Kohonen Self-Organizing Map – SOM) відноситься до класифікаторів, для навчання яких використовуються вибірки образів із заздалегідь не заданою класифікацією (рис.3.1). Тобто, ця ШНМ може здійснювати кластеризацію вхідних даних.

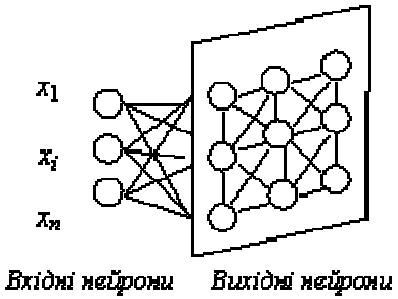


Рис.3.1

Задачею мережі є визначення приналежності вхідного вектора ознак s-го екземпляра вибірки $x^s = \{x^s_1, x^s_2, ..., x^s_N\}^T$ до одного з L можливих кластерів, представлених векторними центрами $w_j = \{w_{j1}, w_{j2}, ..., w_{jN}\}^T$, $j = 1, 2, ..., L$, де ^{T} – символ транспонування.

Позначимо i-у компоненту вхідного вектора x^s у момент часу t як $x^s_i(t)$, а вагу i-го входу j-го вузла в момент часу t як $w_{ij}(t)$.

Якщо вузли SOM є лінійними, а вага i-го входу j-го вузла дорівнює w_{ij}, $i = 1, 2, ..., N$, $j = 1, 2, ..., L$, тоді при відповідних значеннях порогів кожен i-та вихід мережі з точністю до несуттєвих постійних буде дорівнює евклідовій відстані d_j між пред'явленим вхідним вектором x^s_i і j-м центром кластера.

Вважається, що вектор x^s належить до j-го кластера, якщо відстань d_j для j-го центра кластера w_j мінімальна, тобто якщо $d_j \le d_k$ для кожного $k \ne j$.

При навчанні ШНМ пред'являються вхідні вектори без указівки бажаних виходів і корегуються ваги відповідно до алгоритму, що запропонував видатний фінський вчений, академік Теуво Кохонен. Алгоритм Кохонена, що формує SOM, вимагає, щоб біля кожного вузла було визначене поле NE, розмір якого з часом постійно зменшується (рис.3.2).

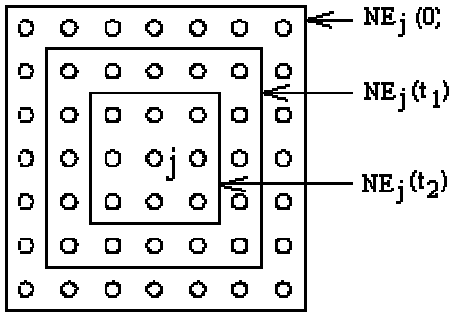


Рис. 3.2

Крок 1. Ініціюються ваги входів вузлів малими випадковими значеннями. Встановлюється початковий розмір поля NE.

Крок 2. Пред'являється новий вхідний вектор x^s.

Крок 3. Обчислюється відстань (метрика) d_j між вхідним вектором і кожним вихідним вузлом j:

$$d_j = \sum_{i=1}^{N} (x_i^s(t) - w_{ji}(t))^2$$

Крок 4. Визначається вузол j^* з мінімальною відстанню d_j.

Крок 5. Корегуються ваги входів вузлів, що знаходяться в полі $NE_j(t)$ вузла j^*, таким чином, щоб нові значення ваг були рівні:

$$w_{ji}(t+1) = w_{ji}(t) + \eta(t)(x_i^s - w_{ji}(t)), \quad j \in NE_j(t), \quad i = 1,2,...,N.$$

При цьому коригувальний приріст $\eta(t)$ ($0 < \eta(t) < 1$) повинний спадати зі зростанням t.

Крок 6. Якщо збіжність не досягнута, то перейти до кроку 2.

Збіжність вважається досягнутою, якщо ваги стабілізувалися і коригувальний приріст η у кроці 5 знизився до нуля.

Якщо число вхідних векторів у навчальній множині є великим стосовно обраного числа кластерів, то після навчання ваги мережі будуть визначати центри кластерів, розподілені в просторі входів таким чином, що функція щільності цих центрів буде апроксимувати функцію щільності ймовірності вхідних векторів. Крім того, ваги будуть організовані таким чином, що топологічно близькі вузли будуть відповідати фізично близьким (у смислі евклідової відстані) вхідним векторам.

Інтерпретація результатів класифікації SOM

Важливо відзначити, що при класифікації за допомогою SOM, номер вузла, до якого віднесений екземпляр, і фактичний номер його класу в загальному випадку не збігаються – розділяючи екземпляри, SOM робить суб'єктивну класифікацію, що не має того реального фактичного змісту, яким ми наділяємо класи.

Результати класифікації SOM можуть бути наділені фактичним смислом шляхом постановки у відповідність номеру кожного вузла SOM номера того фактичного класу, до якого відноситься більша частина екземплярів навчальної вибірки, віднесених SOM до даного вузла. Для здійснення такої постановки можна запропонувати використовувати простий спосіб, заснований на використанні асоціативного запам'ятовуючого пристрою (АЗП).

Алгоритм навчання системи SOM-АЗП має вигляд:

Крок 1. Реалізується навчальний експеримент і визначаються фактичні класи екземплярів. Виробляється навчання SOM для всіх екземплярів навчальної вибірки.

Крок 2. Для кожного вузла SOM підраховується число екземплярів, що відносяться до кожного з фактичних класів.

Крок 3. Кожному вузлу SOM ставиться у відповідність той фактичний клас, до якого відноситься більша частина екземплярів, віднесених SOM до даного вузла. Постановка відповідності виробляється шляхом запису пари (кортежу) <номер вузла SOM, номер класу> в АЗП. У якості АЗП може бути використаний як блок лінійної або динамічної пам'яті, що обслуговується відповідною процедурою, так і нейромережна асоціативна пам'ять:

а) для системи з двома класами – одношаровий дискретний персептрон;

б) для системи з великим числом класів – багатошарова нейронна мережа або комбінація асоціативної пам'яті на основі ШНМ Хопфілда з нейромережним селектором максимуму. При цьому на відповідні входи ШНМ Хопфілда подаються сигнали від кожного з вузлів SOM, а на виході одержують 0, якщо номер вузла SOM не зіставлений даному класу і 1 – якщо зіставлений. Нейромережний селектор максимуму визначає номер вузла ШНМ Хопфілда (тобто номер фактичного класу), для якого вихід дорівнює 1, для всіх інших вузлів SOM вихід ШНМ Хопфілда буде дорівнювати 0.

Нейронна мережа LVQ

Однією з моделей ШНМ, перспективних для діагностики і прогнозування, є квантування навчальних векторів (Learning Vector Quantization – LVQ).

В основі алгоритмів LVQ лежить механізм навчання шару конкуруючих нейронів (конкуруючого шару), контрольований вчителем. Конкуруючий шар може автоматично навчатися класифікувати вхідні вектори. По суті, він представляє собою карту ознак самоорганізації Кохонена. Поділ класів, що визначає карта ознак самоорганізації Кохонена, заснований тільки на відстані між вхідними векторами. Якщо два вхідних вектори дуже близькі, то конкуруючий шар, дуже ймовірно, віднесе їх до одного класу. Однак, поділ на класи, вироблений конкуруючим шаром, як правило, не збігається з тим, що визначає вчитель. Виникає задача розробки механізму, який би дозволяв карті ознак самоорганізації Кохонена здійснювати класифікацію, близьку до заданої вчителем. Методом, що дозволяє вирішувати поставлену задачу, є LVQ.

Нейронна мережа LVQ (рис. 3.3) складається з двох послідовно з'єднаних шарів нейронів: конкуруючого шару і лінійного шару. Обидва шари ШНМ LVQ містять по одному конкуруючому та одному лінійному нейроні на кожен підклас / цільовий клас. Позначимо S^1 – кількість підкласів, S^2– кількість цільових класів (S^1 завжди буде більше, ніж S^2).

Конкуруючий шар здійснює поділ вхідних векторів x на класи, виділяючи центри зосередження вхідних векторів m_i. Для цього визначаються відстані $n^1=\|x - m_i\|$ між вхідними векторами x і початковими значеннями центрів зосередження векторів m_i, $i = 1, 2, .., q$, де q – кількість вхідних векторів.

Лінійний шар перетворює клас вхідного вектора, визначений конкуруючим шаром – підклас a^1, у клас, визначений користувачем – цільовий клас a^2, шляхом множення a^1 на значення ваг LW лінійних нейронів, що

встановлюються рівними 1, якщо цільовий клас і підклас збігаються і 0 – у протилежному випадку. Відповідні добутки $n^2=a^1LW$ подаються на виходи всіх лінійних нейронів, утворюючи бінарний вектор a^2, всі елементи якого рівні 0, за винятком елемента, що відповідає цільовому класу (цей елемент дорівнює 1).

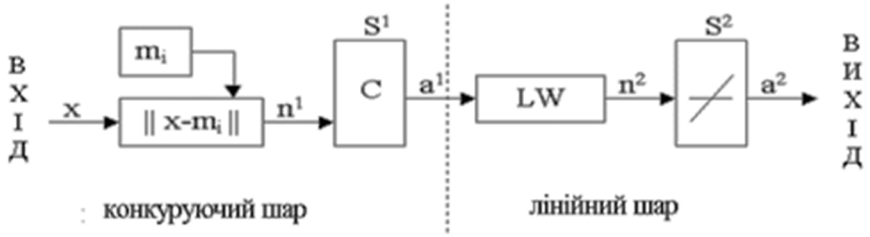


Рис. 3.3

Нехай деяка кількість векторів з вільними параметрами m_i поміщена у вхідний простір для апроксимації різних областей вхідного вектора x їх квантованими значеннями. Кожному класу значень x призначається кілька векторів з вільними параметрами, і потім приймається рішення про віднесення x до того класу, до якого належить найближчий вектор m_i. Нехай індекс c визначає найближчий до x вектор m_i, позначений далі як m_c:

$$c = arg\ \min_i \{\|x - m_i\|\}\ .$$

Значення для m_i, що мінімізують помилку класифікації, можуть бути знайдені як асимптотичні значення в наступному процесі навчання. Нехай $x(t)$ – вхідна вибірка, $m_i(t)$ – представлення послідовності m_i, дискретизованої за часом. Починаючи з правильно визначених початкових значень, основний процес **алгоритму LVQ1** визначають наступні вирази:

$m_c(t+1) = m_c(t) + \alpha(t)[x(t) - m_c(t)]$, якщо x і m_c належать одному та тому самому класу;

$m_c(t+1) = m_c(t) - \alpha(t)[x(t) - m_c(t)]$, якщо x і m_c належать різним класам;

$m_i(t+1) = m_i(t)$, $\forall i \neq c$.

Тут $0 < \alpha(t) < 1$, $\alpha(t)$ може бути константою або монотонно зменшуватися з часом.

Для алгоритму LVQ1 рекомендується, щоб первісне значення α було менше 0.1.

Рішення задачі класифікації в **алгоритмі LVQ2** ідентично алгоритму LVQ1. Однак у процесі навчання LVQ2 два вектори з вільними параметрами m_i і m_j, що є найближчими сусідами x, модифікуються одночасно. Один з них повинний належати до класу 1, а інший – до класу 2. Крім того, x повинний знаходитися в зоні значень, що називається "вікном", яке визначене навколо середини площини, утвореної векторами m_i і m_j. Нехай d_i і d_j – евклідові відстані x від m_i і m_j, тоді x визначено потрапить у "вікно" відносної ширини w, якщо

$$min\left(\frac{d_i}{d_j}, \frac{d_j}{d_i}\right) > s, \qquad \text{де } s = \frac{1-w}{1+w}.$$

Рекомендується, щоб значення відносної ширини "вікна" w знаходилися в межах від 0.2 до 0.3.

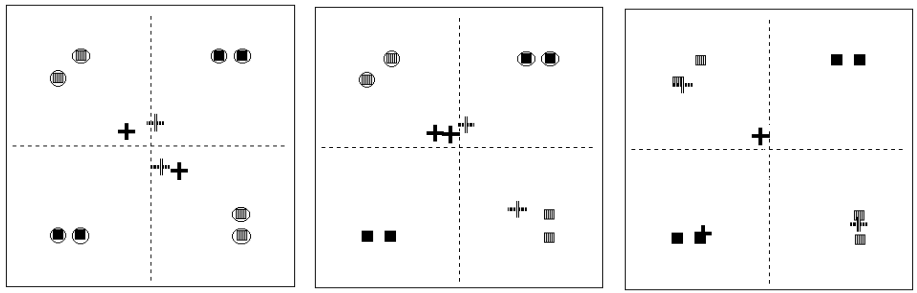
Алгоритм навчання LVQ2 має вигляд:

$m_i(t+1) = m_i(t) - \alpha(t)[x(t) - m_i(t)]$,

$m_j(t+1) = m_j(t) + \alpha(t)[x(t) - m_j(t)]$,

де m_i і m_j – два найближчих до x вектора з вільними параметрами, причому x і m_j належать до одного та того самого класу, в той час як x і m_i належать різним класам, крім того, x повинний попадати в "вікно".

На рис. 3.4 квадратами позначені значення ознак екземплярів, а хрестами – значення ваг ШНМ LVQ. Штрихування фігур позначає їхню приналежність до класу (суцільна – клас 1, пунктирна – клас 2). Помилково класифіковані екземпляри виділені колами.



а) б) в)

Рис. 3.4

На рис. 3.4а) показане початкове розташування ваг ШНМ LVQ – всі екземпляри класифіковані невірно. На рис. 3.4б) для тих же екземплярів показане положення ваг ШНМ LVQ після однієї ітерації навчання за допомогою алгоритму LVQ2 – одночасно модифіковані значення двох векторів, кількість неправильно класифікованих екземплярів зменшилася. На

рис. 3.4в) показане положення ваг ШНМ LVQ після 10 ітерацій навчання – всі екземпляри класифіковані вірно.

ЗАВДАННЯ НА ЛАБОРАТОРНУ РОБОТУ № 3 ТА МЕТОДИЧНІ РЕКОМЕНДАЦІЇ ДО ЇХ ВИКОНАННЯ

Завдання 3.1. Побудуйте мережу Кохонена через командний рядок в Matlab

```
x = simplecluster_dataset;
net = selforgmap([8 8]);
net = train(net,x);
view(net)
y = net(x);
classes = vec2ind(y);
```

Відкриється вікно з контролем навчання мережі (рис. 3.5). Натисканням кнопки SOM Topology викликається вікно з топологією мережі (рис.3.6)

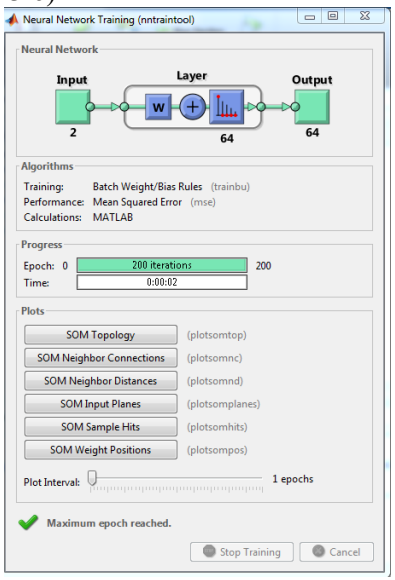


Рис. 3.5

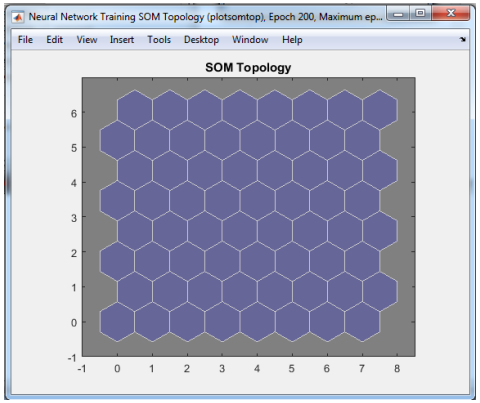
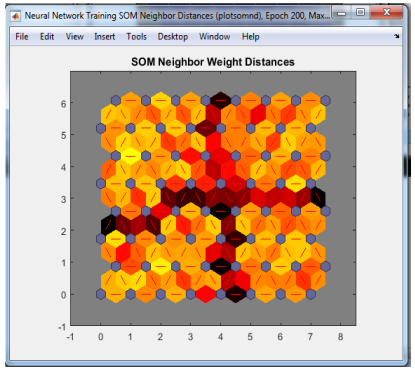


Рис. 3.6

Перегляньте та розберіться, що відображено у інших вікнах, які викликаються натисканням інших доступних кнопок (рис. 3.7-3.8).



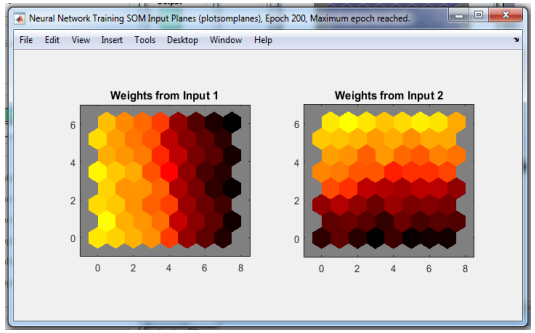
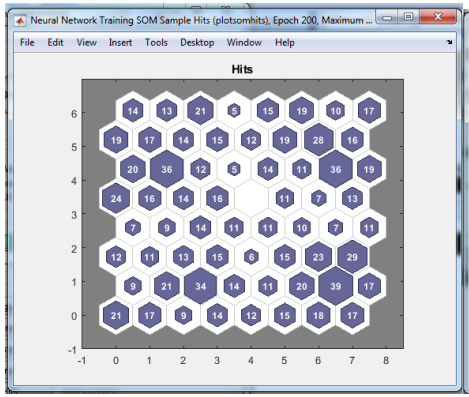


Рис. 3.7



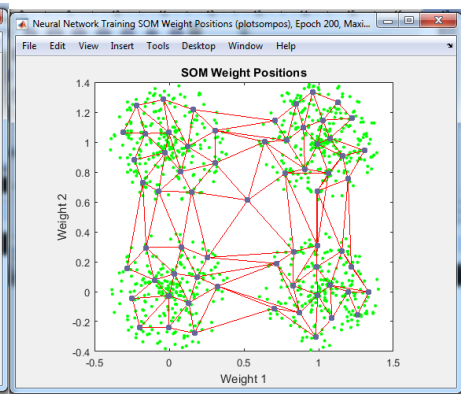


Рис. 3.8

Дайте відповіді на питання: Що відображають відповідні вікна: SOM Neighbor connections; SOM Neighbor weight distances; SOM Input Planes; SOM Sample Hits; SOM Weight Positions. Відповіді занесіть у звіт.

При формуванні карти, що самоорганізується, можна задати певну топологію розташування нейронів. Для цього використовуються М-функції gridtop, hextop, randtop. Для створення простої прямокутної сітки, що складається з шести нейронів розміром 2x3 необхідно використовувати функцію gridtop (рис. 3.9):

```
pos = gridtop (2,3)
plotsom(pos) % Перегляд створеної функції
```

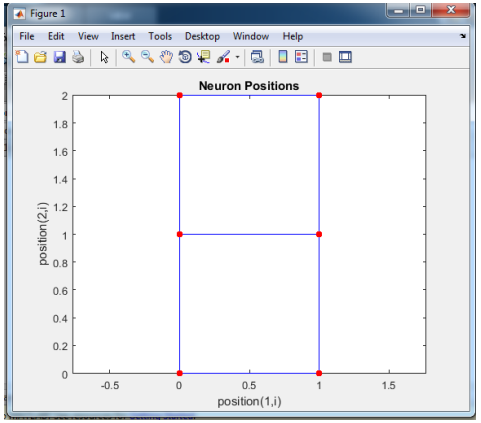


Рис. 3.9

Відповідна сітка показана на рис. 3.9. Мітки position (1, i) і position (2, i) уздовж координатних осей генеруються функцією plotsom і задають позиції розташування нейронів по першій, другій і т. д. розмірностям карти.

На прямокутній сітці нейрон 1 розташований в точці з координатами (0,0), нейрон 2 - в точці (1,0), нейрон 3 - в точці (0,1) і т. д. Зауважимо, що, якщо застосувати команду gridtop, переставивши аргументи місцями, отримаємо інше розміщення нейронів.

Якщо потрібно створити прямокутну сітку розміром 8 × 10 за допомогою функції gridtop необхідно задати програмний код:

```
pos = gridtop(8,10);
plotsom(pos)
```

Гексагональну сітку можна сформувати за допомогою функції hextop

```
pos = hextop(2,3)
pos =
0 1.0000 0.5000 1.5000 0 1.0000
0 0 0.8660 0.8660 1.7321 1.7321.
plotsom (pos) %
```

М-функція hextop використовується за умовчанням при створенні карт Кохонена при застосуванні функції newsom. Сітка з випадковим розташуванням вузлів може бути створена за допомогою функції randtop:

```
pos = randtop (2,3)
```


Завдання 3.2. Побудуйте сітки згідно розмірів для свого варіанту (таблиця 3.1) відповідно до номера по журналу.

Таблиця 3.1

№ за списком у журналі	1	2	3	4	5	6	7	8	9	10
Прямокутна сітка	8×8	2×6	3×4	3×8	8×6	6×8	7×8	4×8	8×9	8×10
Гексагональна сітка	6×4	2×4	3×4	8×8	5×5	8×6	8×7	8×4	9×8	8×10

№ за списком у журналі	11	12	13	14	15	16	17	18	19	20
Прямокутна сітка	2×8	2×5	3×3	3×7	8×5	6×7	5×8	3×8	9×2	8×4
Гексагональна сітка	4×4	2×3	3×4	8×7	4×5	7×6	7×7	8×3	2×9	8×5

№ за списком у журналі	21	22	23	24	25	26	27	28	29	30
Прямокутна сітка	3×8	6×6	3×5	3×6	5×6	6×5	7×7	4×4	4×9	5×10
Гексагональна сітка	4×4	3×4	3×5	8×6	5×6	6×6	4×7	8×5	9×4	10×5

Зображення відповідних сіток занесіть у звіт (рис.1 та 2 звіту)

Завдання 3.3. Дослідження конкурентного навчання мережі

Нейрони в конкурентному шарі навчаються представляти різні області вхідного простору, де відбуваються вхідні вектори.

Створимо множину P, що являє собою набір випадково згенерованих, але кластерних точок даних тесту. Конкурентна мережа буде використана для класифікації цих точок у природні класи.

Розгляньте приклад створення вхідних даних:

```
% Створення вхідних даних X.
bounds = [0 1; 0 1];   % Центри кластерів перебувають у цих межах.
clusters = 8;          % Це кількість кластерів.
points = 10;           % Кількість точок у кожному кластері.
std_dev = 0.05;        % Стандартне відхилення кожного кластера.
```

```
x = nngenc(bounds,clusters,points,std_dev);

% Графік вхідних даних X.
plot(x(1,:),x(2,:),'+r');
title('Input Vectors');
xlabel('x(1)');
ylabel('x(2)');
```

Графік вхідних даних X подано на рис. 3.10

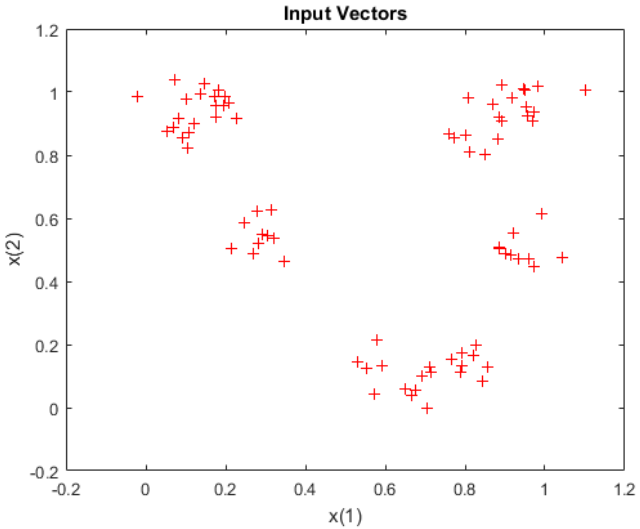


Рис. 3.10

Створіть свої вхідні данні X відповідно до свого варіанту. Вхідні параметри для кожного варіанту подані у таблиці 3.2.

Таблиця 3.2

№ за списком у журналі	1	2	3	4	5	6	7	8	9	10
bounds	[0 1; 0 1]	[0 1; 0 1]	[0 1; 0 1]	[0 1; 0 1]	[0 1; 0 1]	[0 2; 0 2]	[0 2; 0 2]	[0 2; 0 2]	[0 2; 0 2]	[0 2; 0 2]
clusters	8	7	6	8	8	5	6	8	7	6
points	12	10	11	13	9	9	11	10	9	12
std_dev	0.05	0.06	0.05	0.06	0.05	0.06	0.07	0.08	0.09	0.09

№ за списком у журналі	11	12	13	14	15	16	17	18	19	20
bounds	[0 1; 0 1]	[0 1; 0 1]	[0 1; 0 1]	[0 1; 0 1]	[0 1; 0 1]	[0 2; 0 2]	[0 2; 0 2]	[0 2; 0 2]	[0 2; 0 2]	[0 2; 0 2]
clusters	5	6	7	9	7	6	6	8	7	8
points	12	10	11	13	9	9	12	10	10	12
std_dev	0.06	0.05	0.08	0.05	0.06	0.08	0.07	0.05	0.06	0.05

№ за списком у журналі	21	22	23	24	25	26	27	28	29	30
bounds	[0 1; 0 1]	[0 1; 0 1]	[0 1; 0 1]	[0 1; 0 1]	[0 1; 0 1]	[0 2; 0 2]	[0 2; 0 2]	[0 2; 0 2]	[0 2; 0 2]	[0 2; 0 2]
clusters	7	8	9	7	8	9	6	7	8	9
points	11	12	15	10	11	12	13	8	9	11
std_dev	0.09	0.08	0.07	0.06	0.05	0.09	0.08	0.07	0.06	0.08

Графік з вхідними даними занесіть у звіт (рис. 3 звіту)

Для побудови мережі використаємо функцію COMPETLAYER, що приймає два аргументи, *кількість нейронів* і *швидкість навчання*.

Ми можемо налаштувати мережеві входи (зазвичай виконується автоматично за допомогою TRAIN) і побудувати початкові вектори ваги, щоб побачити їх спробу класифікації.

Вектори ваги (позначені на рис. 3.11 – «o») будуть навчені таким чином, щоб вони відображалися в центрі кластерів вхідних векторів (позначені на рис. 3.11 «+»).

Виконайте дії для своїх вхідних даних.

```
net = competlayer(8,.1);
net = configure(net,x);
w = net.IW{1};
plot(x(1,:),x(2,:),'+r');
hold on;
circles = plot(w(:,1),w(:,2),'ob');
```

Графік з початковими вагами подібний до рис. 3.11 занесіть у звіт (рис. 4 звіту)

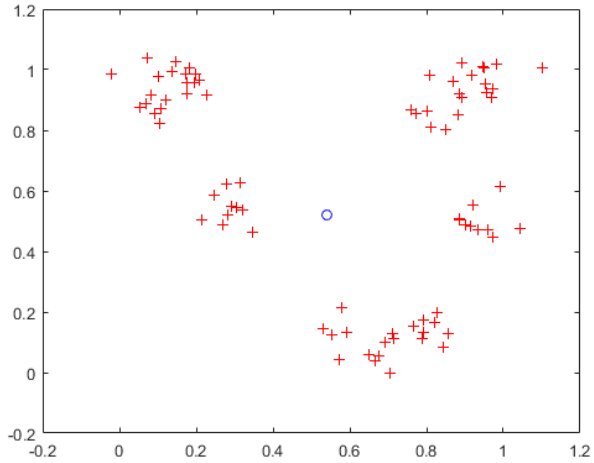


Рис. 3.11

Встановіть кількість епох для тренування перед зупинкою і тренуйте цей конкурсний шар (може тривати кілька секунд).

Складіть оновлені ваги шару на одному графіку (рис. 3.12).

```
net.trainParam.epochs = 7;
net = train(net,x);
w = net.IW{1};
delete(circles);
plot(w(:,1),w(:,2),'ob');
```

Зверніть увагу! Якщо сині кружечки не співпадають з центрами кластерів червоних хрестиків (кількість нейронів недостатня для визначення кількості кластерів), то збільшить кількість нейронів у функції `competlayer(8,.1)` , щоб їх кількість відповідала кількості кластерів для вашого варіанту та проведіть повторне навчання.

Графік з навченими вагами подібний до рис. 3.12 занесіть у звіт (рис. 5 звіту)

Тепер ми можемо використовувати конкурентний шар як класифікатор, де кожен нейрон відповідає іншій категорії.

Тут ми визначимо вхідний вектор x1 як [0; 0,2].

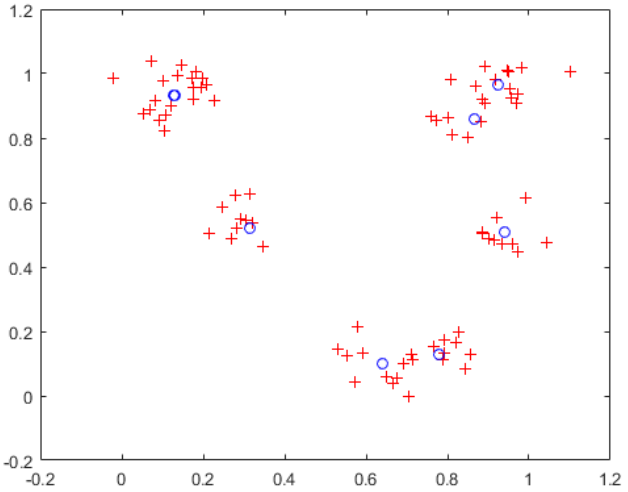


Рис. 3.12

Вихід Y вказує, який нейрон реагує, і тим самим, якому класу належить вхідний сигнал.

```
x1 = [0; 0.2];
y = net(x1)
y =

   0
   1
   0
   0
   0
   0
   0
   0
```

Результати класифікації (вектор y) занесіть у звіт (рис. 6 звіту)

Зробіть висновки по правильності класифікації та запишіть їх у звіт.

Завдання 3.4. Дослідження одномірної самоорганізуючої карти

Нейрони в 2-D шарі вчаться представляти різні області вхідного простору, де знаходяться вхідні вектори. Крім того, сусідні нейрони вчаться

реагувати на подібні входи, таким чином шар вивчає топологію представленого вхідного простору.

Тут 100 одиниць даних створюються на одиничному колі.

Конкурентна мережа буде використана для класифікації цих точок у природні класи.

```
angles = 0:0.5*pi/99:0.5*pi;
X = [sin(angles); cos(angles)];
plot(X(1,:),X(2,:),'+r')
```

Графік функції подано на рис. 3.13

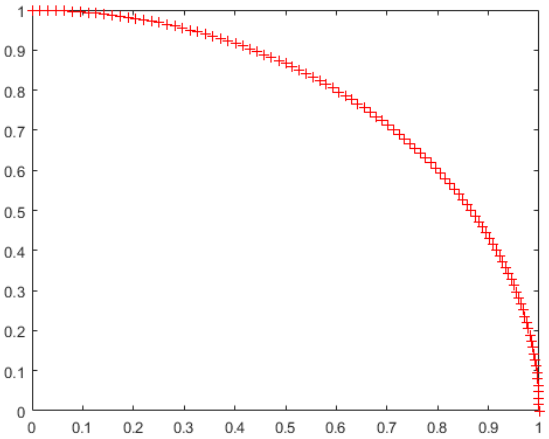


Рис. 3.13

Графік функції занесіть у звіт (рис. 7 звіту)

Створимо карту з 1-мірним шаром із 10 нейронів.

```
net = selforgmap(10);
```

Вкажіть мережу, яка повинна пройти навчання за 10 епох і використовувати TRAIN для навчання мережі на вхідних даних P: data P:

```
net.trainParam.epochs = 10;
net = train(net,X);
```

Вікно тренування мережі подано на рис. 3.14, а топологія створеної мережі – на рис. 3.15

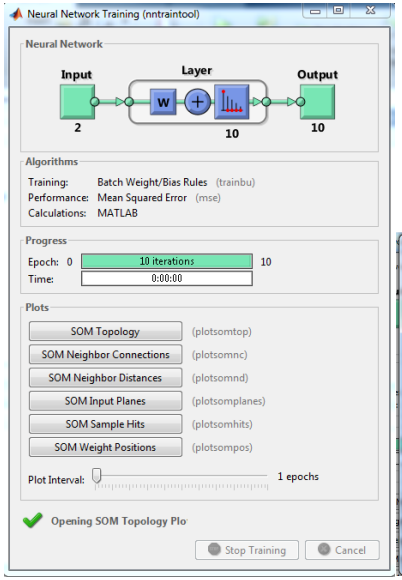


Рис. 3.14

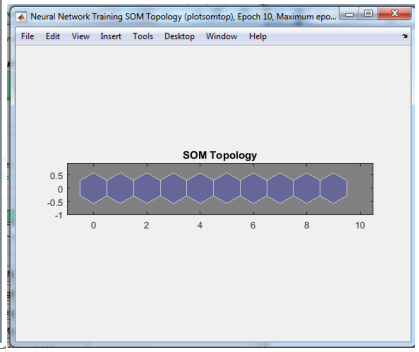


Рис. 3.15

Тепер побудуйте вагові позиції тренованої мережі за допомогою PLOTSOMPOS (рис. 3.16). Сині точки - це вектори ваги нейрона, а червоні лінії з'єднують кожну пару на відстані 1.

```
plotsompos(net)
```

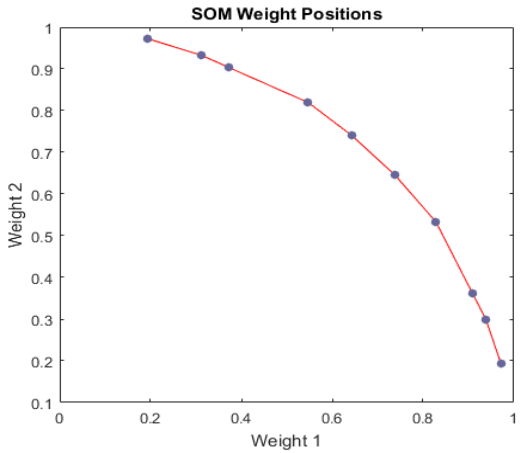


Рис. 3.16

Карта тепер може бути використана для класифікації входів, як X = [1; 0]:

Або нейрон 1 або 10 повинен мати вихід 1, оскільки вищевказаний вхідний вектор був на одному кінці представленого вхідного простору. Перша пара цифр позначає нейрон, а число одиниця вказує його вихід.

```
x = [1;0];
a = net(x)
a =

   1
   0
   0
   0
   0
   0
   0
   0
   0
   0
```

Проведіть дослідження розробленої мережі при вхідних даних згідно варіантів поданих у таблиці 3.3.

Таблиця 3.3

№ за списком у журналі	1	2	3	4	5	6	7	8	9	10
Нейронів у шарі	8	9	10	11	12	8	9	10	11	12
Кількість епох	7	8	9	10	11	12	7	8	9	10
Вектор входів X	[1; 0]	[0; 1]	[3; 0]	[0; 3]	[5; 0]	[6; 0]	[7; 0]	[0; 3]	[1; 4]	[4; 1]

№ за списком у журналі	11	12	13	14	15	16	17	18	19	20
Нейронів у шарі	9	10	11	12	8	9	10	11	12	8
Кількість епох	7	8	9	10	11	12	7	8	9	10
Вектор входів X	[1; 0]	[0; 1]	[3; 0]	[0; 3]	[5; 0]	[5; 0]	[5; 0]	[0; 2]	[1; 4]	[4; 1]

№ за списком у журналі	21	22	23	24	25	26	27	28	29	30
Нейронів у шарі	8	9	10	11	12	8	9	10	11	12
Кількість епох	8	9	10	11	12	7	8	9	10	11
Вектор входів X	[1; 0]	[0; 1]	[2; 0]	[0; 2]	[3; 0]	[4; 0]	[5; 0]	[0; 3]	[1; 4]	[4; 1]

Рисунок з топологією мережі (подібний до рис. 3.15) занесіть у звіт (рис. 8 звіту)

Рисунок з позиціями ваг (подібний до рис. 3.16) занесіть у звіт (рис. 9 звіту)

Рисунок з результатами класифікації (вектор a)занесіть у звіт (рис. 10 звіту)

Зробіть висновки по правильності класифікації та запишіть їх у звіт.

Завдання 3.5. Дослідження двовимірної самоорганізуючої карти

Як і в попередньому завданні, ця самоорганізуюча карта навчиться представляти різні області вхідного простору, де відображаються вхідні вектори. Проте у цьому завданні нейрони організують себе в двовимірну сітку, а не в лінію.

Необхідно класифікувати 1000 двоелементних векторів, що відображаються у прямокутному векторному просторі.

```
X = rands(2,1000);
plot(X(1,:),X(2,:),'+r')
```

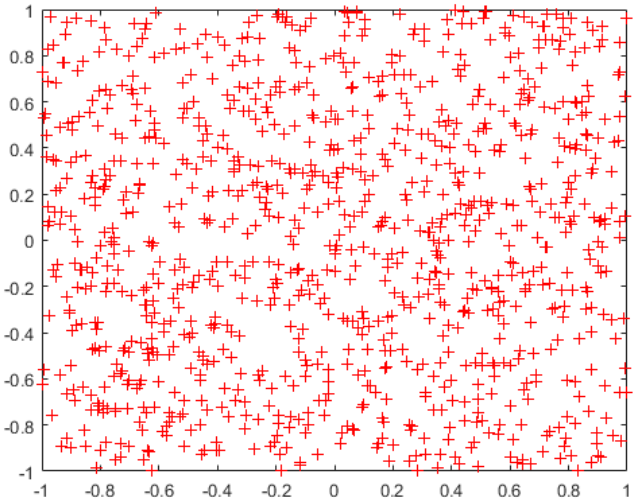


Рис. 3.17

Для класифікації вищевказаних векторів ми будемо використовувати шар нейронів 5 на 6. Ми хотіли б, щоб кожен нейрон реагував на іншу область прямокутника, а сусідні - на сусідні області.

Мережа налаштована відповідно до розмірів входів. Цей крок потрібний тут, тому що ми побудуємо початкові ваги. Зазвичай навчання виконується автоматично за допомогою TRAIN.

```
net = selforgmap([5 6]);
net = configure(net,X);
```

Ми можемо візуалізувати мережу, яку ми тільки що створили за допомогою PLOTSOMPOS.

Кожен нейрон представлений червоною крапкою на місці її двох ваг. Спочатку всі нейрони мають однакові ваги в середині векторів, тому з'являється лише одна точка.

```
plotsompos(net)
```

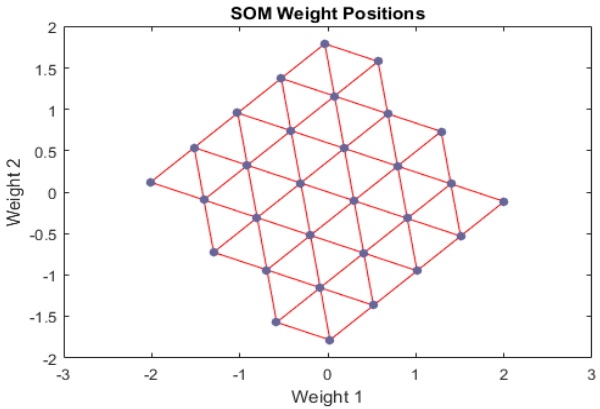


Рис. 3.18

Тепер ми тренуємо карту на 1000 векторів за 1 епоху і переставляємо ваги мережі.

Після навчання відзначимо, що шар нейронів почав самоорганізуватися так, що кожен нейрон тепер класифікує іншу область вхідного простору, а сусідні (пов'язані) нейрони реагують на сусідні області.

```
net.trainParam.epochs = 1;
net = train(net,X);
plotsompos(net)
```

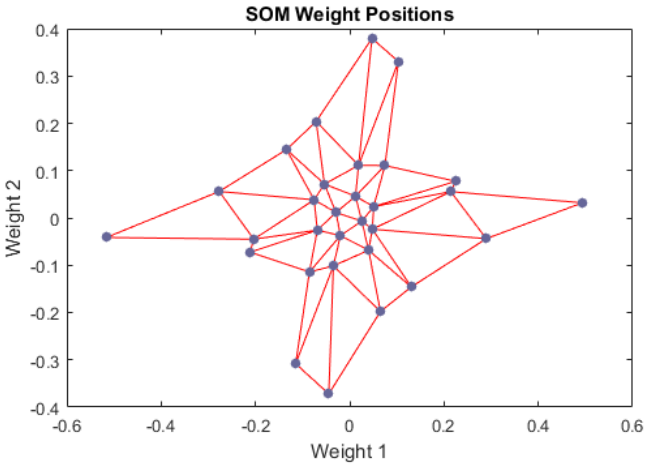



Рис. 3.19

Тепер ми можемо використовувати SIM, щоб класифікувати вектори, передавши їх мережі і побачивши, який нейрон реагує.

```
x = [0.5;0.3];
y = net(x)
y =
```

Нейрон, по підрахунку 26, який відповів «1», тому x належить до цього класу.

Проведіть дослідження двовимірної мережі при вхідних даних згідно варіантів поданих у таблиці 3.4.

Таблиця 3.4

№ за списком у журналі	1	2	3	4	5	6	7	8	9	10
Кількість вхідних векторів	1000	900	800	700	750	850	950	1000	1100	900
Шар нейронів	[8 5]	[5 5]	[6 5]	[5 6]	[7 5]	[8 5]	[5 8]	[5 7]	[5 5]	[5 5]
Кількість епох	1	2	1	2	1	2	1	2	1	2

№ за спис-	11	12	13	14	15	16	17	18	19	20

ком у журналі										
Кількість вхідних векторів	900	800	700	750	850	950	1000	1100	900	800
Шар нейронів	[7 5]	[6 5]	[7 5]	[6 6]	[7 8]	[8 8]	[5 7]	[5 6]	[7 5]	[6 5]
Кількість епох	1	2	1	2	1	2	1	2	1	2

№ за списком у журналі	21	22	23	24	25	26	27	28	29	30
Кількість вхідних векторів	800	700	750	850	950	1000	1100	900	1000	950
Шар нейронів	[8 6]	[5 6]	[6 6]	[5 5]	[8 5]	[7 5]	[5 7]	[7 7]	[6 5]	[5 6]
Кількість епох	1	2	1	2	1	2	1	2	1	2

Вектор класифікації X для всіх варіантів x = [0.5;0.3]

Рисунок вхідних векторів (подібний до рис. 3.17) занесіть у звіт (рис. 11 звіту). Рисунок з топологією ваг (подібний до рис. 3.18) занесіть у звіт (рис. 12 звіту). Рисунок з позиціями ваг після навчання (подібний до рис. 3.19) занесіть у звіт (рис. 13 звіту). Рисунок з результатами класифікації (вектор y) занесіть у звіт (рис. 14 звіту).

Зробіть висновки по правильності класифікації та запишіть їх у звіт.

Завдання 4.6. Дослідження кластеризації за допомогою мереж, що самоорганізуються

Побудувати нейронну мережу, яка кластеризує квіти ірису в природні класи, так що подібні класи згруповані разом. Кожна квітка описується чотирма ознаками:

1. Довжина чашечки квітки у см
2. Ширина чашечки квітки в см
3. Довжина пелюстки в см

4. Ширина пелюстки в см

Необхідно згрупувати зразки в класи на основі подібності між зразками. Нейронна мережа, повинна не тільки визначати класи для відомих вхідних даних, але і класифікувати невідомі вхідні дані.

Карти, що самоорганізуються (SOMs) дуже часто застосовують для класифікацій. Далі, карти класифікації зберігають топологічну інформацію про те, які класи найбільш схожі з іншими. Карти, що самоорганізуються можна створювати з будь-яким бажаним рівнем деталізації. Вони особливо добре підходять для кластеризації даних у багатьох вимірах і з комплексними формами та пов'язаними просторами об'єктів. Тому вони добре підходять для кластерів ірисів.

Чотири атрибути квітки будуть діяти як вхідні дані для SOM, що відобразить їх на 2-мірному шарі нейронів.

Підготовка даних

Дані для кластеризації подаються для SOM шляхом організації їх у вхідну матрицю X. Кожний *i*-тий стовпець вхідної матриці буде мати чотири елементи, що представляють чотири вимірювання, зроблені на одній квітці.

Завантажте такий набір даних.

```
x = iris_dataset;
```

Переглянте розмір входів X.

Зауважте, що X має 150 стовпців. Вони являють собою 150 наборів атрибутів квітки ірису. Вона має чотири ряди для чотирьох вимірювань.

```
size(x)
ans =

   4   150
```

Кластеризація з нейронною мережею

Наступним кроком є створення нейронної мережі, яка навчиться групувати.

selforgmap створює самоорганізуються карти для класифікації зразків з такою деталізацією, наскільки це потрібно, вибравши число нейронів у кожному вимірі шару.

Створимо 2-мірний шар з 64 нейронів, розташованих у гексагональній сітці 8x8. Загалом, більша деталізація досягається при більшій кількості нейронів, та дозволяє моделювати топологію більш складних просторів ознак.

Розмір вхідних даних дорівнює 0, оскільки мережа ще не налаштована на відповідність нашим вхідним даним. Це станеться, коли мережа буде навчена.

```
net = selforgmap([8 8]);
view(net)
```

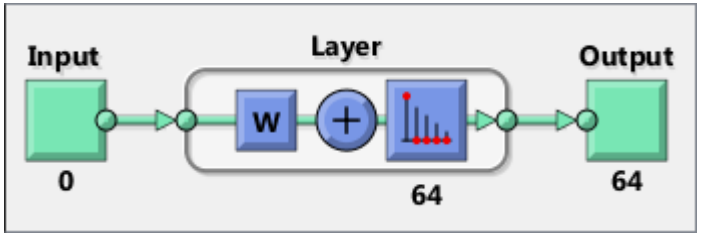


Рис. 3.20

Тепер мережа готова до оптимізації з навчанням.

Інструмент NN Training показує мережу, що навчається, і алгоритми, які використовуються для навчання. Він також відображає стан навчання під час тренування, а критерії, які зупинили тренування, будуть виділені зеленим кольором.

Кнопки внизу відкривають корисні графіки, які можна відкрити під час і після тренування. Посилання поруч з іменами алгоритму і кнопками ділянки відкривають документацію на ці теми.

Навчіть мережу:

```
[net,tr] = train(net,x);
Nntraintool
```

Тут самоорганізуюча карта використовується для обчислення векторів класів кожного з навчальних входів (рис. 3.21). Ці класифікації охоплюють простір ознак, заповнений відомими квітами, і тепер можна використовувати для класифікації нових квітів відповідно. Вихід мережі буде матрицею 64x150, де кожен *i*-й стовпчик являє собою *j*-й кластер для кожного *i*-го вхідного вектора з 1 у його *j*-му елементі.

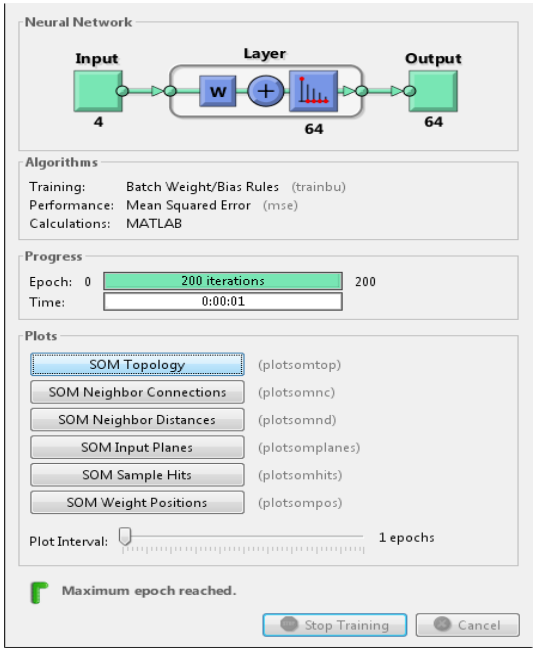


Рис. 3.21

Функція vec2ind повертає індекс нейрона з виходом 1 для кожного вектора. Індекси будуть коливатися в межах від 1 до 64 для 64 кластерів, представлених 64 нейронами.

```
y = net(x);
cluster_index = vec2ind(y);
```

Рис. 3.22 - plotsomtop графік самоорганізуються топологічних карт 64 нейронів, розташованих в гексагональній сітці 8x8. Кожен нейрон навчився представляти інший клас квітки, причому прикріплені нейрони зазвичай представляють подібні класи.

```
plotsomtop(net)
```

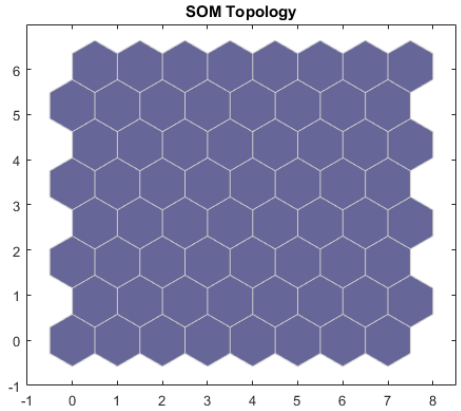


Рис. 3.22

Plotsomhits обчислює класи для кожної квітки і показує кількість квіток у кожному класі (рис. 3.23). Області нейронів з великим числом попадань вказують на класи, що представляють подібні високо населені регіони простору ознак. Натомість ділянки з невеликою кількістю попадань вказують на малонаселені регіони простору ознак.

```
plotsomhits(net,x)
```

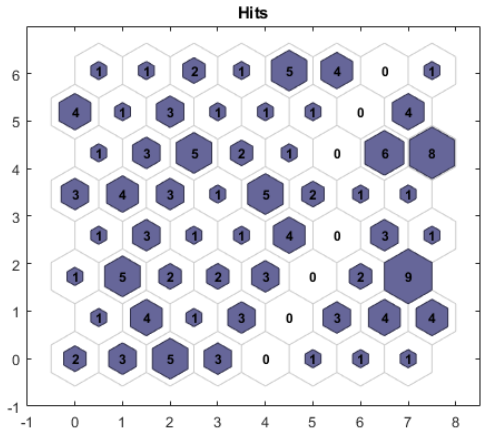


Рис. 3.23

Plotsomnc показує з'єднання сусідів нейронів (рис. 3.24). Сусіди зазвичай класифікують подібні зразки.

```
plotsomnc(net)
```

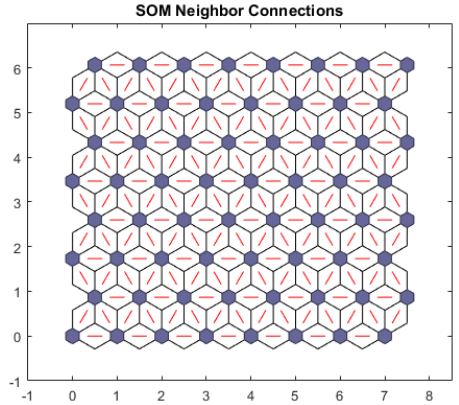



Рис. 3.24

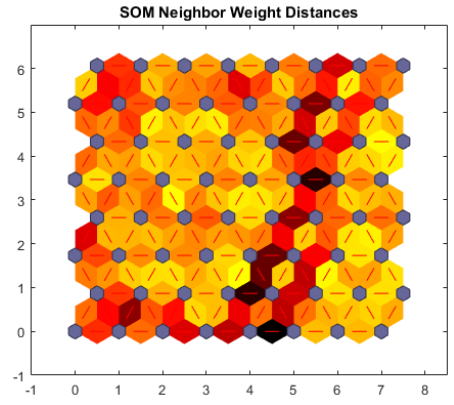


Рис. 3.25

Plotsomnd показує, наскільки віддалений (з точки зору евклідової відстані) клас кожного нейрона від його сусідів (рис. 3.25). З'єднання, які є яскравими, вказують на дуже пов'язані області вхідного простору. У той час як темні з'єднання вказують класи, що представляють регіони простору ознак, які знаходяться далеко один від одного, з невеликою кількістю кольорів або без них між ними.

Довгі межі темних з'єднань, що розділяють великі області вхідного простору, вказують на те, що класи з обох боків кордону являють собою квіти з дуже різними характеристиками.

```
plotsomnd(net)
```

Plotsomplanes показує вагову площину для кожної з чотирьох функцій введення (рис. 3.26). Це візуалізації ваг, що з'єднують кожен вхід з кожним з 64 нейронів в гексагональній сітці 8x8. Більш темні кольори являють собою великі ваги. Якщо два входи мають аналогічні значення ваги (їх кольорові градієнти можуть бути однаковими або в зворотному напрямку), це вказує на їх високу кореляцію.

```
plotsomplanes(net)
```

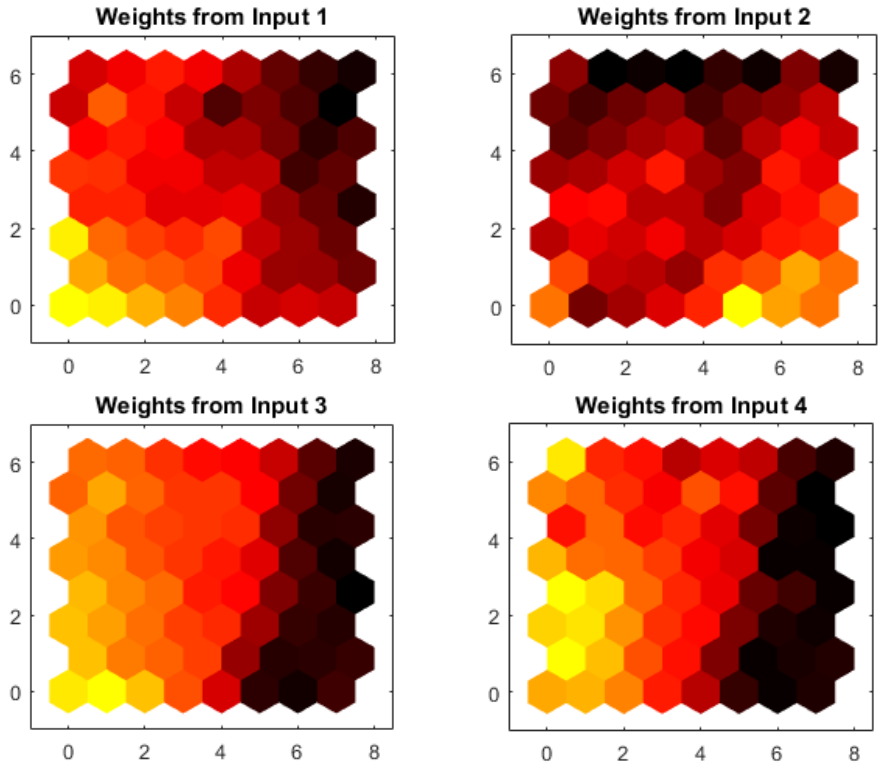


Рис. 3.26

Цей приклад проілюстрував, як розробити нейронну мережу, яка кластеризує квіти ірису на основі чотирьох їх характеристик.

Створіть нейронну мережу вказаного у таблиці 3.5 розміру навчіть її на тих самих вхідних даних. Результати занесіть у звіт

Таблиця 3.5

№ за списком у журналі	1	2	3	4	5	6	7	8	9	10
Шар нейронів	[8 7]	[7 8]	[9 8]	[8 9]	[7 5]	[8 9]	[10 8]	[9 8]	[9 9]	[10 10]

№ за списком у журналі	11	12	13	14	15	16	17	18	19	20
Шар нейронів	[7 7]	[7 8]	[6 6]	[8 9]	[9 5]	[8 9]	[10 8]	[9 8]	[9 9]	[10 10]

№ за списком у журналі	21	22	23	24	25	26	27	28	29	30
Шар нейронів	[8 7]	[10 8]	[11 11]	[8 9]	[7 5]	[8 9]	[5 8]	[6 8]	[6 9]	[5 5]

Рисунок навчання мережі (подібний до рис. 3.21) занесіть у звіт (рис. 15 звіту).

Рисунок з обчисленими елементами класів (подібний до рис. 3.23) занесіть у звіт (рис. 16 звіту).

Рисунок з ваговими площинами для кожної з чотирьох функцій (подібний до рис. 3.26) занесіть у звіт (рис. 17 звіту).

Завдання 4.7. Дослідження мережі квантування навчального вектора

Мережа LVQ підготовлена для класифікації вхідних векторів відповідно до заданих цілей.

Нехай X - 10 вхідних 2-елементних векторів, а C - класи, до яких потрапляють ці вектори. Ці класи можуть бути перетворені у вектори, які будуть використовуватися як цілі, T, з IND2VEC.

```
x = [-3 -2 -2  0  0  0  0 +2 +2 +3;
     0 +1 -1 +2 +1 -1 -2 +1 -1  0];
c = [1 1 1 2 2 2 2 1 1 1];
t = ind2vec(c);
```

Виведіть вектор на графік.

```
colormap (hsv);
ділянка (x, c)
title ("Вхідні вектори");
xlabel ('x (1)');
ylabel ('x (2)');
```

Тут нанесені точки даних. RED = клас 1, Cyan = клас 2. Мережа LVQ представляє кластери векторів із прихованими нейронами і групує кластери з вихідними нейронами для формування бажаних класів.

Зображення вхідних даних занесіть у звіт (рис. 18 звіту)

Створіть мережу LVQ.

```
net = lvqnet (4,0,1);
net = configure (net, x, t);
```

Тут LVQNET створює шар LVQ з чотирма прихованими нейронами і швидкістю навчання 0,1. Потім мережа налаштовується на входи X і цілі T. (Конфігурація зазвичай є непотрібним кроком, оскільки виконується автоматично за допомогою TRAIN.)

Вектори ваги конкурентного нейрона будуються наступним чином

```
hold on
w1 = net.IW{1};
plot(w1(1,1),w1(1,2),'ow')
title('Input/Weight Vectors');
xlabel('x(1), w(1)');
ylabel('x(2), w(2)');
```

Щоб навчити мережу, спочатку перекрийте типову кількість епох, а потім тренуйте мережу. По завершенні відобразяться вхідні вектори '+' і вектори 'o' конкурентних нейронів. RED = клас 1, Cyan = клас 2.

```
net.trainParam.epochs=150;
net=train(net,x,t);

cla;
plotvec(x,c);
hold on;
plotvec(net.IW{1}',vec2ind(net.LW{2}),'o');
```

Зображення вхідних векторів і конкурентних нейронів занесіть у звіт (рис. 19 звіту)

Тепер використовуйте мережу LVQ як класифікатор, де кожен нейрон відповідає іншій категорії. Представте вхідний вектор [0,2; 1]. RED = клас 1, Cyan = клас 2.

```
x1 = [0.2; 1];
y1 = vec2ind(net(x1))
```

Результат класифікації y1 запишіть у звіт.

Зробіть висновки по роботі у яких укажіть:

Для чого застосовуються мережі Кохонена та LVQ

ЗВІТНІСТЬ ЗА ЛАБОРАТОРНУ РОБОТУ № 3

У звіті з лабораторної роботи необхідно представити всі графіки та висновки згідно завдання.

Назвіть звіт ШІ-КБ-ЛР-3-NNN-XXXXX.doc
де NNN – номер групи
XXXXX – позначення прізвища студента.

Переконвертуйте файл звіту в ШІ-КБ- ЛР-3-NNN-XXXXX.pdf

Надішліть звіт викладачу на електронну пошту.

ПИТАННЯ ДЛЯ САМОКОНТРОЛЮ

1. Дайте визначення понять: нейронна мережа, метрика, асоціативна пам'ять, SOM, LVQ.
2. Які задачі можна вирішувати на основі SOM і LVQ, а які не можна? Обґрунтуйте і доведіть відповідь. Приведіть приклади.
3. Порівняєте можливості SOM, ШНМ Хопфілда та одношарового дискретного персептрона.
4. Як впливають вид і параметри метрики на точність визначення центрів зосередження екземплярів. Відповідь поясніть і проілюструйте.
5. Запропонуєте способи обліку апріорної інформації про значимість ознак при навчанні SOM.
6. Чи доцільно використовувати SOM для попереднього аналізу даних у задачах діагностики і прогнозування?
7. Запропонуйте стратегію (алгоритм) визначення необхідної кількості нейронів конкуруючого шару LVQ.

ЛАБОРАТОРНА РОБОТА № 4

ДОСЛІДЖЕННЯ РАДІАЛЬНИХ ТА РЕКУРЕНТНИХ НЕЙРОННИХ МЕРЕЖ

Мета роботи: *вивчити моделі і властивості радіальних та рекурентних нейронних мереж та можливості їх застосування при рішенні задач кібербезпеки; дослідити можливості ППП MATLAB щодо проектування радіальних та рекурентних нейронних мереж.*

ТЕОРЕТИЧНІ ВІДОМОСТІ

Для виконання лабораторної роботи необхідно вивчити теоретичний матеріал лекцій по теорії штучних нейронних мереж та матеріал поданий у цьому підрозділі.

Радіальні базисні мережі

Радіальні базисні нейронні мережі складаються з більшої кількості нейронів, ніж стандартні мережі з прямою передачею сигналів і навчанням методом зворотного поширенням помилки, але на їх створення потрібно значно менше часу. Ці мережі особливо ефективні, коли є велика кількість навчальних векторів.

Нижче, крім мереж загального вигляду, обговорюються 2 спеціальних типи радіальних базисних мереж: мережі GRNN (Generalized Regression Neural Networks) для вирішення завдань узагальненої регресії і мережі PNN (Probabilistic Neural Networks) для вирішення імовірнісних задач.

Для створення радіальних мереж загального вигляду призначені M-функції newrbe і newrb, а узагальнених регресійних і імовірнісних - M-функції newgrnn і newpnn відповідно [4].

Основні команди для створення радіальних мереж в MATLAB подані в таблиці 4.1

Таблиця 4.1.

Radial basis networks	Радіальні базисні мережі
New networks	формування мережі
newrb	Створення радіальної базисної мережі
newrbe	Створення радіальної базисної мережі з нульовою помилкою
newgrnn	Створення узагальненої регресійної мережі

newpnn	Створення ймовірнісної мережі
Using networks	Робота з мережею
sim	моделювання мережі
Weight functions	функції зважування
dist	Евклідова відстань
dotprod	Скалярний добуток
normprod	Нормоване скалярний твір
Net input functions	функції накопичення
netprod	Твір зважених входів
netsum	Сума зважених входів
Transfer functions	функції активації
compet	Конкуруюча функція активації
purelin	Функція активації з жорсткими обмеженнями
radbas	Радіальна базисна функція активації
Performance	Функції оцінки якості мережі
mse	Ефективне значення похибка
Signals	перетворення даних
ind2vec	Перетворення індексного вектора в матрицю зв'язності
vec2ind	Перетворення матриці зв'язності в індексний вектор

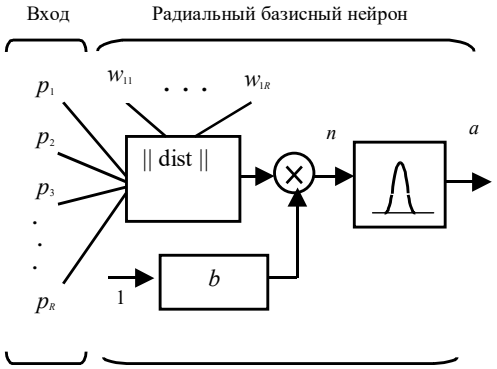
На рис. 4.1 показана радіальна базисна мережа з R входами. функція активації для радіального базисного нейрона має вигляд:

$$\text{radbas}(n) = e^{-n^2}. \qquad (4.1)$$

Вхід функції активації визначається як модуль різниці вектора ваг w і вектора входу p, помножений на зміщення b.

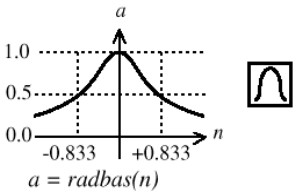
Графік функції активації представлений на рис. 4.2. Ця функція має максимум, рівний 1, коли вхід дорівнює 0. Коли відстань між векторами w і p зменшується, вихід радіальної базисної функції збільшується.

Таким чином, радіальний базисний нейрон діє як індикатор, який формує значення 1, коли вхід p ідентичний вектору ваг w. Зсув b дозволяє коректувати чутливість нейрона radbas. Наприклад, якщо нейрон мав зсув 0.1, то його виходом буде 0.5 для будь-якого вектора входу p і вектора ваги w при відстані між векторами, що дорівнює 8.333, або 0.833/b.



a = radbas (||**p** – **w**|| b)

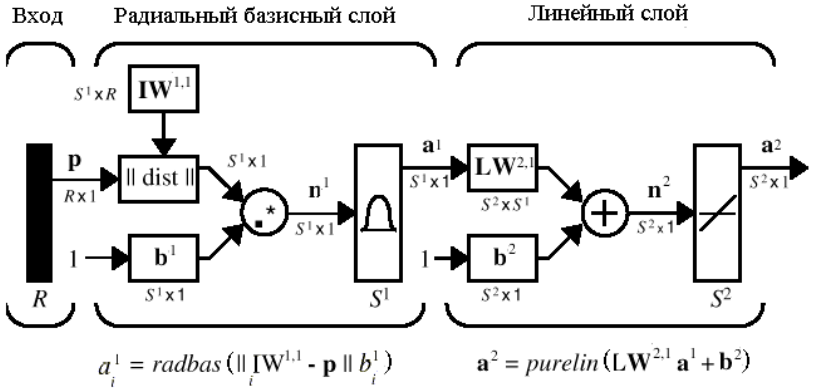
Рис. 4.1



a = *radbas(n)*

Рис. 4.2

Радіальна базисна мережа складається з двох шарів: прихованого радіального базисного шару, що має S1 нейронів, і вихідного лінійного шару, що має S2 нейронів (рис. 4.3).



$a_i^1 = radbas(\| {}_i\mathrm{IW}^{1,1} - \mathbf{p} \| b_i^1)$ $\quad \mathbf{a}^2 = purelin(\mathrm{LW}^{2,1}\,\mathbf{a}^1 + \mathbf{b}^2)$

Рис. 4.3

Входами блоку || dist || на цьому малюнку є вектор входу p і матриця ваг $IW^{1,1}$, а виходом - вектор, що складається з S^1 елементів, які визначаються відстанями між *i*-м вектором входу p і *i*-й вектор-рядком ${}_iIW^{1,1}$ матриці ваг. Такий вектор-рядок будемо називати вектором ваг *i*-го нейрона. Вихід блоку || dist || множиться поелементно на вектор зміщення b^1 і формує вхід функції активації. Тоді вихід першого шару може бути записаний в такій формі:

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

Використовувати такий складний запис при застосуванні ППП Neural Network Toolbox не буде потрібно, оскільки всі операції, пов'язані зі створенням радіальної базисної мережі, оформлені у вигляді спеціальних М-функцій newrbe і newrb.

Для того щоб зрозуміти поведінку мережі, необхідно простежити проходження вектора входу p. При завданні вектора входу кожен нейрон радіального базисного шару видасть значення відповідно до того, як близький вектор входу до вектору ваг кожного нейрона. Таким чином, радіальні базисні нейрони з векторами ваг, що значно відрізняються від вектора входу p, матимуть виходи, близькі до 0, і їх вплив на виходи лінійних нейронів буде незначним. Навпаки, радіальний базисний нейрон з вектором ваг, близьким до вектору входу p, видасть значення, близьке до 1, і це значення буде передано на лінійний нейрон з вагою, відповідною вихідному прошарку. Таким чином, якщо тільки 1 радіальний базисний нейрон має вихід 1, а всі інші мають виходи, рівні або дуже близькі до 0, то вихід лінійного шару буде дорівнює вазі активного вихідного нейрона. Однак це винятковий випадок, зазвичай вихід формують кілька нейронів з різними значеннями ваг.

Радіальна базисна мережа з нульовою помилкою

Для побудови радіальних базисних мереж з нульовою помилкою призначена функція newrbe, яка викликається наступним чином:

net = newrbe (P, T, SPREAD)

Вхідними аргументами функції newrbe є масиви вхідних векторів P і цілей T, а також параметр впливу SPREAD. Вона повертає радіальну базисну мережу з такими вагами і зсувами, що її виходи точно рівні цілям T. Функція newrbe створює стільки нейронів радіального базисного шару, скільки є вхідних векторів в масиві P і встановлює ваги першого шару рівними P'. При цьому зміщення встановлюються рівними 0.8326/SPREAD. Це означає, що рівень перекриття радіальних базисних функцій дорівнює 0.5 і всі входи в діапазоні ±SPREAD вважаються значимими. Зрозуміло, що чим більший діапазон вхідних значень повинен бути прийнятий до уваги, тим більше значення параметра впливу SPREAD має бути встановлено. Це найбільш наочно проявляється при вирішенні завдань апроксимації функцій.

Ваги другого шару $\mathbf{IW}^{21}$ (**IW** {2,1} - в позначеннях системи MATLAB) та зсувів $\mathbf{b}^2$ (b {2}) можуть бути знайдені шляхом моделювання

виходів першого шару a^1 (A {1}) і наступного рішення системи лінійних алгебраїчних рівнянь (СЛАР):

```
[W {2,1} b {2}] * [A {1}; ones] = T
```

Оскільки відомі входи другого шару A {1} і цілі T, а шар лінійний, то для обчислення ваг і зміщень другого шару досить скористатися вирішувачем СЛАР

```
Wb = T / [P; ones (1, size (P, 2))]
```

Тут матриця Wb містить ваги і зміщення (зсуву - в останньому стовпці). Сума квадратів похибок мережі буде завжди дорівнює 0, так як маємо задачу з Q рівняннями (пари вхід/мета) і кожен нейрон має Q + 1 змінних (Q ваг по числу радіальних базисних нейронів і одне зміщення). СЛАР з Q рівняннями і більш ніж Q змінними має вільні змінні і характеризується нескінченним числом рішень з нульовою похибкою.

Таким чином, в результаті навчання функція newrbe створює радіальну базисну мережу з нульовою похибкою на навчальній множині. Єдина умова, яку потрібно виконати, полягає в тому, щоб значення параметра SPREAD було досить великим, щоб активні області базисних функцій перекривалися, щоб покрити весь діапазон вхідних значень. Це дозволяє забезпечити необхідну гладкість апроксимаційних кривих і перешкоджає виникненню явища перенавчання. Однак значення SPREAD не повинно бути настільки великим, щоб радіальна базисна функція оголошувала однаково значущими все значення входу.

Недолік функції newrbe полягає в тому, що вона формує мережу з числом нейронів в прихованому шарі, що дорівнює кількості елементів навчальної вибірки. Тому за допомогою цієї функції не можна отримати прийнятного рішення в разі великих розмірів навчальної множини, що характерно для реальних додатків.

Ітераційна процедура формування мережі

Функція newrb створює радіальну базисну мережу, використовуючи ітеративну процедуру, яка додає по одному нейрону на кожному кроці. Нейрони додаються до прихованого прошарку до тих пір, поки сума квадратів помилок не стане менше заданого значення або не буде використана максимальна кількість нейронів. Ця функція викликається за допомогою команди

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


Входами функції newrb є масиви вхідних і цільових векторів P і T, а також параметри GOAL (допустима середньоквадратична помилка мережі), SPREAD (параметр впливу), а виходом - опис радіальної базисної мережі. Значення параметра SPREAD має бути досить великим, щоб покрити весь діапазон значень входів, але не настільки, щоб ці значення були однаково значущими.

Якщо порівнювати мережі з прямою передачею сигналу і радіальні базисні мережі, то слід зауважити, що при вирішенні одних і тих же завдань вони мають певні переваги одна перед іншою. Так, радіальні базисні мережі з нульовою похибкою мають значно більше нейронів, ніж мережу з прямою передачею сигналу і сигмоїдальними функціями активації в прихованому шарі. Це обумовлено тим, що сигмоїдальні функції активації перекривають великі діапазони значень входу, ніж радіальні базисні функції. З іншого боку, проектування радіальної базисної мережі вимагає значно меншого часу, а при обмеженій точності навчання може зажадати і меншої кількості використовуваних нейронів.

На дослідження цих мереж направлені завдання 4.1-4.3.

Мережі GRNN

Нейронні мережі GRNN (Generalized Regression Neural Network) призначені для вирішення завдань узагальненої регресії, аналізу часових рядів і апроксимації функцій. Характерною особливістю цих мереж є дуже висока швидкість їх навчання.

Архітектура мережі GRNN показана на рис. 4.4. Вона аналогічна архітектурі радіаль-ної базисної мережі, але відрізняється структурою другого шару, в якому використовується блок normprod для обчислення нормованого скалярного добутку рядка масиву вагів $\mathbf{LW}^{21}$ і вектора входу $\mathbf{a}^1$ у відповідності з наступним співвідношенням:

$$\mathbf{n}^2 = \frac{\mathbf{LW}^{21}\mathbf{a}^1}{sum(\mathbf{a}^1)} \ . \qquad (4.2)$$

Перший шар - це радіальний базисний шар з числом нейронів, що дорівнює кількості елементів Q навчальної множини; в якості початкового наближення для матриці ваг вибирається масив $\mathbf{P}'$; зміщення $\mathbf{b}^1$ встановлюється рівним вектор-стовпцю з елементами 0.8326/SPREAD.

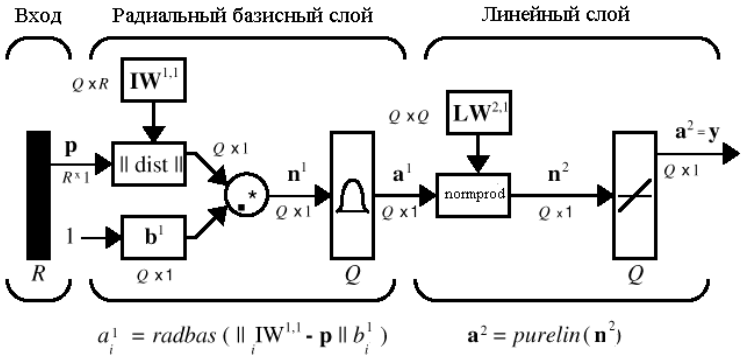


Рис. 4.4

Функція dist обчислює відстань між вектором входу і вектором ваги нейрона; вхід функції активації $\mathbf{n}^1$ дорівнює поелементному добутку зваженого входу мережі на вектор зміщення; вихід кожного нейрона першого шару $\mathbf{a}^1$ є результатом перетворення вектора $\mathbf{n}^1$ радіальною базисною функцією radbas. Якщо вектор ваги нейрона дорівнює транспоновану вектору входу, то зважений вхід дорівнює 0, а вихід функції активації - 1. Якщо відстань між вектором входу і вектором ваги нейрона одно spread, то вихід функції активації буде дорівнює 0.5.

Другий шар - це лінійний шар з числом нейронів, що також дорівнює кількості елементів Q навчальної множини, причому в якості початкового наближення для матриці ваг **LW**{2,1} вибирається масив **T**. Припустимо, що маємо вектор входу $\mathbf{p}_i$, близький до од-ному з векторів входу **p** з навчальної множини. Цей вхід $\mathbf{p}_i$ генерує значення виходу шару $\mathbf{a}_i^1$, близьке до 1. Це призводить до того, що вихід шару 2 буде близький до $\mathbf{t}_i$.

Якщо параметр впливу SPREAD малий, то радіальна базисна функція характеризується різким спадом і діапазон вхідних значень, на який реагують нейрони прихованого шару, виявляється досить малим. Зі збільшенням параметра SPREAD нахил радіальної базисної функції стає більш гладким, і в цьому випадку вже кілька нейронів реагують на значення вектора входу. Тоді на виході мережі формується вектор, що відповідає середньому декількох цільових векторів, відповідних вхідних векторів навчальної множини, близьких до даного вектору входу. Чим більше значення параметра SPREAD, тим більше число нейронів бере участь у формуванні середнього значення, і в підсумку функція, що генерується мережею, стає більш гладкою.

Для створення нейронної мережі GRNN призначена М-функція newgrnn.

На дослідження мереж GRNN направлено завдання 4.4.

Мережі PNN

Нейронні мережі PNN (Probabilistic Neural Networks) призначені для вирішення імовірнісних задач, і зокрема задач класифікації.

Архітектура мережі PNN базується на архітектурі радіальної базисної мережі, але в якості другого шару використовує так званий конкуруючий шар, який подсчіти-кість ймовірність приналежності вхідного вектора до того чи іншого класу і в конеч-ном рахунку зіставляє вектор з тим класом, ймовірність приналежності до якого вище. Структура мережі PNN представлена на рис. 4.5.

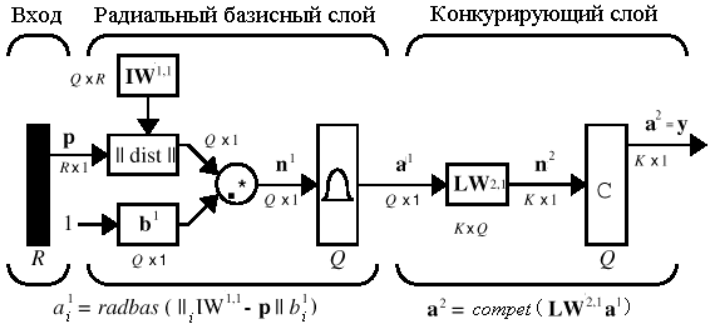


Рис. 4.5

Передбачається, що задано навчальну множину, що складається з Q пар векторів вхід/ціль. Кожен вектор мети має *K* елементів, що вказують клас приналежності, і, таким чином, кожен вектор входу ставиться у відповідність одному з *K* класів.

В результаті може бути утворена матриця зв'язності **T** розміру $K \times Q$, що складається з нулів і одиниць, рядки якої відповідають класам приналежності, а стовпці - векторам входу. Таким чином, якщо елемент **T**(*i, j*) матриці зв'язності дорівнює 1, то це означає, що *j*-й вхідний вектор належить до класу *i*.

Вагова матриця першого шару $\mathbf{IW}^{11}$ (net.IW {1,1}) формується з використанням векторів входу з навчальної множини у вигляді матриці **P'**. Коли подається новий вхід, блок || dist || обчислює близькість нового вектора до векторів навчальної множини; потім обчислені відстані множаться на

зміщення і подаються на вхід функції активації radbas. Вектор навчальної множини, найбільш близький до вектору входу, буде представлений в векторі виходу $\mathbf{a}^1$ числом, близьким до 1.

Вагова матриця другого шару $\mathbf{LW}^{21}$ (net.LW {2,1}) відповідає матриці зв'язності **T**, побудованої для даної навчальної послідовності. Ця операція може бути виконана за допомогою М-функції ind2vec, яка перетворює вектор цілей в матрицю зв'язності **T**. Твір $\mathbf{T}*\mathbf{a}^1$ визначає елементи вектора $\mathbf{a}^1$, відповідні кожному з **K** класів. В результаті конкуруюча функція активації другого шару compet формує на виході значення, рівне 1, для найбільшого за величиною елемента вектора $\boldsymbol{n}^2$ і 0 в інших випадках. Таким чином, мережа PNN виконує класифікацію векторів входу по K класах.

Мережі PNN можуть досить ефективно застосовуватися для рішення завдань класифікації. Якщо задано досить велике навчальне безліч, то рішення, які генеруються мережами, сходяться до рішень, відповідним правилом Байеса. Недолік мереж GRNN і PNN полягає в тому, що працюють вони відносно повільно, оскільки виконують дуже великі обсяги обчислень у порівнянні з другими типами нейронних мереж.

Рекурентні мережі

Розглянемо два типи рекурентних нейронних мереж, що викликають найбільший інтерес для користувачів, - це клас мереж Елмана (Elman) і клас мереж Хопфілда (Hopfield). Характерною особливістю архітектури рекурентної мережі є наявність блоків динамічної затримки і зворотних зв'язків. Це дозволяє таким мережам обробляти динамічні моделі.

Мережі Елмана

Мережа Елмана - це мережа, що складається з двох шарів, в якій прихований шар охоплений динамічним зворотнім зв'язком. Це дозволяє врахувати передісторію спостережуваних процесів і накопичити інформацію для вироблення правильної стратегії управління. Мережі Елмана застосовуються в системах управління рухомими об'єктами, при побудові систем технічного зору і в інших додатках.

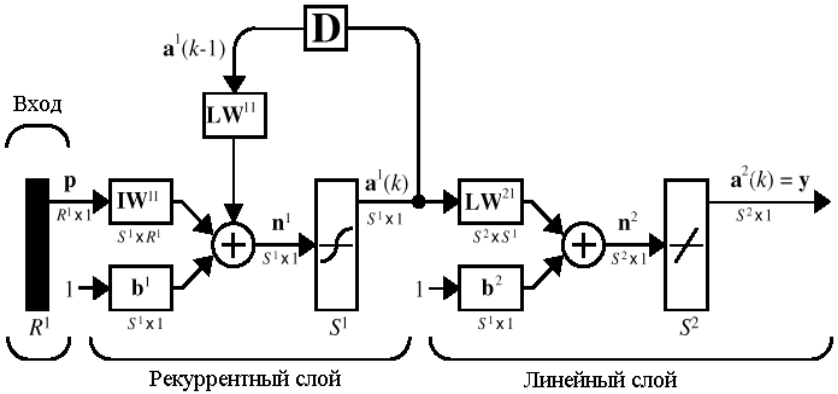
За командою help elman можна отримати наступну інформацію про М-функціях, що входять у склад ППП Neural Network Toolbox і відносяться до побудови мереж Елмана:

Таблиця 4.2

Elman recurrent networks	Рекурентні мережі Елмана
New networks	формування мережі
newelm	Створення мережі Елмана
Using networks	Робота з мережею
sim init adapt train	моделювання ініціалізація адаптація навчання
Weight functions	Вагові функції
dotprod ddotprod	Скалярний добуток Похідна скалярного добутоку
Net input functions	функції накопичення
netsum dnetsum	Сума зважених входів Похідна суми зважених входів
Transfer functions	Функції активації
purelin tansig logsig dpurelin dtansig dlogsig	Лінійна Гіперболічний тангенс Логістична Похідна лінійної функції Похідна гіперболічного тангенса Похідна логістичної функції
Performance functions	Функції оцінки якості мережі
Mse	Середньоквадратичне значення помилки навчання
msereg	Середньоквадратичне значення помилка навчання при застосуванні регуляризації
dmse	Похідна середньоквадратичної помилки навчання
dmsereg	Похідна середньоквадратичної помилки навчання при застосуванні регуляризації
Initialization functions	Функції ініціалізації мережі
initlay initnw	пошарова ініціалізація Функція NW (Nguyen - Widrow)
Learning functions	Функції настройки мережі
learngd	Функція настройки методом градієнтного спуску
learngdm	Функція настройки методом градієнтного спуску зі збуреннями

Adapt functions	**Функції адаптації**
adapt	Адаптація ваг і зміщень
Training functions	Функції навчання
traingd	Метод градієнтного спуск по правилу зворотного поширення помилки
traingdm	Метод градієнтного спуск з обуренням
traingda	Метод градієнтного спуск з адаптацією параметра швидкості настройки

Мережа Елмана - це, як правило, двошарова мережа зі зворотним зв'язком від виходу до входу першого шару (рис. 4.6).



$$\mathbf{a}^1(k) = \mathbf{tansig}(\mathbf{IW}^{11}\,\mathbf{p} + \mathbf{LW}^{11}\,\mathbf{a}^1(k-1) + \mathbf{b}^1) \qquad \mathbf{a}^2(k) = \mathbf{purelin}(\mathbf{LW}^{21}\,\mathbf{a}^1(k) + \mathbf{b}^2)$$

Рис. 4.6

Як функції активації в мережі Елмана часто використовуються: в прихованому, рекурентному шарі - функція гіперболічного тангенса tansig, в лінійному шарі - функція purelin. Таке поєднання функцій активації дозволяє максимально точно апроксимувати функції з кінцевим числом точок розриву. Єдина вимога, що пред'являється до мережі, полягає в тому, щоб прихований шар мав досить велике число нейронів, що необхідно для успішної апроксимації складних функцій.

Відповідно до структурної схеми мережі Елмана сформуємо динамічний опис її рекурентного шару у вигляді рівнянь стану

$$\begin{cases} \mathbf{n}^1(k) = \mathbf{LW}^{11}\mathbf{a}^1(k-1) + \mathbf{IW}^{11}\mathbf{p} + \mathbf{b}^1, \quad \mathbf{a}^1(0) = \mathbf{a}_0^1; \\ \mathbf{a}^1(k) = \text{tansig}(\mathbf{n}^1(k)). \end{cases} \tag{4.3}$$

Ця рекурентна матрична форма рівнянь стану зайвий раз підкреслює назву досліджуваних нейронних мереж.

Другий, лінійний шар є безінерційним і описується співвідношеннями

$$\begin{cases} n^2(k) = \mathbf{LW}^{21}\mathbf{a}^1(k) + \mathbf{b}^2; \\ \mathbf{a}^2(k) = \text{purelin}(\mathbf{n}^2(k)). \end{cases} \tag{4.4}$$

Мережа Елмана досліджується у завданні 4.6.

Для навчання мережі Елмана можуть бути використані як процедура адаптації, так і процедура навчання, що реалізуються за допомогою функцій adapt і train відповідно.

В процесі процедури адаптації на кожному кроці виконуються наступні дії:

• моделювання мережі при подачі повного набору векторів входу і обчислення помилки мережі;

• обчислення наближеного градієнта функціоналу помилки щодо ваг і зсувів методом зворотного поширення помилки;

• налаштування ваг з використанням функції налаштування, обраній користувачем; рекомендується функція learngdm.

В процесі процедури навчання на кожному циклі виконуються наступні дії:

• моделювання мережі при подачі послідовності вхідних сигналів, порівняння з цільовими виходами і обчислення помилки;

• обчислення наближеного градієнта функціоналу помилки щодо ваг і зсувів методом зворотного поширення помилки;

• налаштування ваг з використанням функції налаштування, обраній користувачем; рекомендується функція traingdx.

Мережі Елмана не забезпечують високої точності рішення, оскільки присутність зворотного зв'язку в рекурентному шарі не дозволяє обчислити точно градієнт функціоналу.

Мережі Хопфілда

Всякий цільової вектор можна розглядати як набір характерних ознак деякого об'єкта. Якщо створити рекурентну мережу, положення рівноваги якої співпадало б з цим цільовим вектором, то таку мережу можна було б розглядати як асоціативну пам'ять. Надходження на вхід такої мережі деякого набору ознак у вигляді початкових умов переводило б її в те чи інше положення рівноваги, що дозволяло б асоціювати вхід з деяким об'єктом. Саме такі асоціативні можливості і мають мережі Хопфілда. Вони відносяться до класу зворотних нейронних мереж, які володіють властивістю,

що за кінцеве число тактів часу вони з довільного початкового стану приходять в стан стійкої рівноваги, що носить назву аттрактор. Кількість таких аттракторів визначає обсяг асоціативної пам'яті мережі Хопфілда.

Спроектувати мережу Хопфілда - це значить створити рекурентну мережу зі множиною точок рівноваги, таких, що при завданні початкових умов мережа в кінцевому рахунку приходить в стан спокою в одній з цих точок. Властивість рекурсії проявляється в тому, що вихід мережі подається назад на вхід. Можна сподіватися, що вихід мережі встановиться в одній з точок рівноваги. Пропонований нижче метод синтезу мережі Хопфілда не є абсолютно досконалим в тому сенсі, що мережа яка синтезується на додаток до бажаних може мати паразитні точки рівноваги. Однак число таких паразитних точок повинно бути зведено до мінімуму за рахунок конструювання методу синтезу. Більш того, область тяжіння точок рівноваги повинна бути максимально великою.

Метод синтезу мережі Хопфілда заснований на побудові системи лінійних диференціальних рівнянь першого порядку, яка задана в деякому замкнутому гіперкубі простору станів і має рішення в вершинах цього гіперкуба. Така мережа дещо відрізняється від класичної моделі Хопфілда, але вона простіша для розуміння і проектування, і ми будемо посилатися на неї як на модифіковану мережу Хопфілда.

За командою help hopfield можна отримати наступну інформацію про М-функції, що входять до складу ППП Neural Network Toolbox і відносяться до побудови модифікованих мереж Хопфілда (таблиця 4.3):

Таблиця 4.3

Hopfield recurrent networks	*Рекурентна модифікована мережа Хопфілда*
New networks	**Формування мережі**
newhop	Створення модифікованої мережі Хопфілда
Weight functions	**Операції з ваговою функцією**
dotprod	Скалярний добуток
Net input functions	**Операції над входами**
netsum	підсумовування
Transfer functions	**Функції активації**
satlins	Симетрична лінійна функція з обмеженнями

Архітектура модифікованої мережі Хопфілда представлена на рис. 4.7.

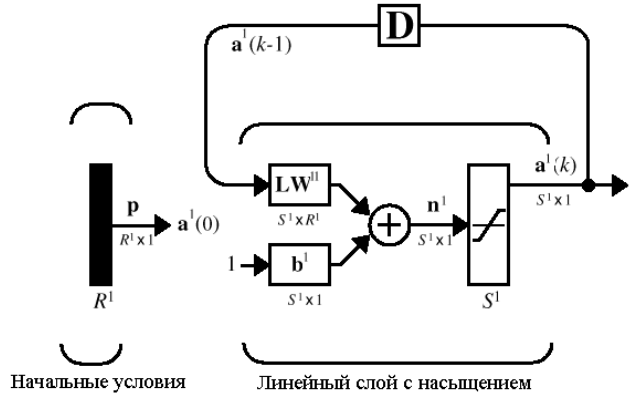


Рис. 4.7

Вхід p встановлює значення початкових умов. У мережі використовується лінійна функція активації з насиченням satlins, яка описується в такий спосіб:

$$a = satlins(n) = \begin{cases} -1, & n < -1; \\ n, & -1 \le n \le 1; \\ 1, & n > 1. \end{cases} \qquad (4.4)$$

Ця мережа може бути промодельована з одним або більшою кількістю векторів входу, які задаються як початкові умови. Після того як початкові умови задані, мережа генерує вихід, який по зворотному зв'язку подається на вхід. Цей процес повторюється багато разів, поки вихід встановиться в положення рівноваги. Можна сподіватися, що кожен вектор виходу в кінцевому рахунку зійдеться до однієї з точок рівноваги, найбільш близької до вхідного сигналу.

Динамічна модель рекурентного шару модифікованої мережі Хопфілда описується наступним чином:

$$\mathbf{a}^1(k) = \text{satlins}(\mathbf{LW}^{11}\mathbf{a}^1(k-1) + \mathbf{b}^1), \quad \mathbf{a}^1(0) = \mathbf{p} \ . \qquad (4.5)$$

При уважному аналізі наведеного співвідношення можна дійсно переконатися, що матриця ваг LW11 рівносильна перехідною матриці динамічної системи, а вектор зсувів b1 - вектору передачі одиничного входу. Необхідно сформувати ці елементи, якщо задані точки рівноваги системи t в вершинах гіперкуба.

Метод проектування модифікованих мереж Хопфілда описаний в роботі [5], на основі цього методу розроблено алгоритм синтезу, який реалізований в ППП NT у вигляді підфункції solvehop2 (t) М-функції newhop.

Якщо задано множину цільових точок рівноваги, представлених матрицею T, то функція newhop повертає матрицю ваг і вектор зсувів для рекурентного шару мережі Хопфілда. При цьому гарантується, що точки стійкої рівноваги будуть відповідати цільовим векторах, але можуть з'явитися і так звані паразитні точки. У процесі синтезу мережі число таких небажаних точок зводиться до мінімуму.

Коли мережа спроектована, вона може бути перевірена з одним або більшим числом векторів входу. Досить імовірно, що вектори входу, близькі до цільових точок, рівноваги знайдуть свої цілі. Здатність модифікованої мережі Хопфілда швидко обробляти набори векторів входу дозволяє перевірити мережу за відносно короткий час.

Мережа Хопфілда досліджується у завданні 4.7.

ЗАВДАННЯ НА ЛАБОРАТОРНУ РОБОТУ № 4 ТА МЕТОДИЧНІ РЕКОМЕНДАЦІЇ ДО ЇХ ВИКОНАННЯ

Завдання 4.1. Застосування радіальної базисної мережі для вирішення задачі апроксимації функції однієї змінної

Подамо функцію $f(x)$ наступним розкладанням:

$$f(x) = \sum_{i=1}^{N} \alpha_i \varphi_i(x) \ , \qquad (4.6)$$

де $\varphi_i(x)$ - радіальна базисна функція.

Тоді ідея апроксимації може бути представлена графічно наступним чином. Розглянемо зважену суму трьох радіальних базисних функцій, заданих на інтервалі [-3 3].

```
p = –3:.1:3;
a1 = radbas(p);
a2 = radbas(p–1.5);
a3 = radbas(p+2);
a = a1 + a2*1 + a3*0.5;
plot(p,a1,p,a2,p,a3*0.5,p,a)
```

Побудуйте ці графіки в MATLAB

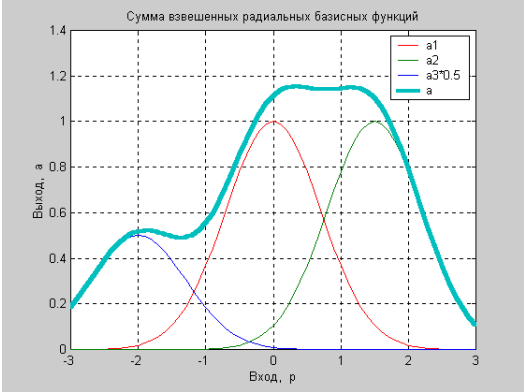


Рис. 4.8

Як випливає з аналізу рис. 4.8 розкладання по радіальних базисних функціях забезпечує необхідну гладкість. Тому їх застосування для апроксимації довільних нелінійних залежностей цілком виправдано. Розкладання виду (4.6) може бути реалізовано на двошаровій нейронній мережі, перший шар якої складається з радіальних базисних нейронів, а другий - з єдиного нейрона з лінійною характеристикою, на якому реалізується підсумовування виходів нейронів першого шару.

Сформуйте навчальну множину і поставте допустиме значення функціоналу помилки, рівне 0.01, параметр впливу визначить рівним 1 і використовуйте ітераційну процедуру формування радіальної базисної мережі:

```
P = -1: .1: 1;
T = [-.9602 -.5770 -.0729 .3771 .6405 .6600 .4609 .1336 ...
    -.2013 -.4344 -.5000 -.3930 -.1647 .0988 .3072 .3960 ...
     .3449 .1816 -.0312 -.2189 -.3201];
GOAL = 0.01; % Допустиме значення функціоналу помилки
SPREAD = 1; % Параметр впливу
net = newrb (P, T, GOAL, SPREAD); % Створення мережі
net.layers {1} .size% Число нейронів в прихованому шарі

ans = 6
```

Для заданих параметрів нейронна мережа складається з шести нейронів і забезпечує наступні можливості апроксимації нелінійних залежностей після навчання.

Моделюючи сформовану нейронну мережу, побудуємо апроксимаційну криву на інтервалі [-1 1] з кроком 0.01 для нелінійної залежності.

```
plot (P, T, '+ k')% Точки навчальної множини
hold on;
X = -1: .01: 1;
Y = sim (net, X); % Моделювання мережі
plot (X, Y); %
```

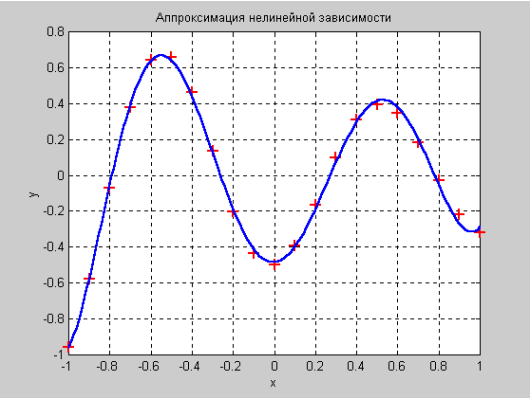


Рис. 4.9

З аналізу рис. 4.9 випливає, що при невеликій кількості нейронів прихованого шару радіальна базисна мережа досить добре апроксимує нелінійну залежність, задану навчальною множиною із 21 точки.

Графік апроксимаційної залежності (подібний до рис. 4.9) занесіть у звіт (рис.1 звіту)

Побудуйте апроксимацію функції для свого варіанту (таблиця 4.1) відповідно до номера по журналу. Вхідні данні вектора T подані у таблиці 4.5.

Таблиця 4.4

№ за списком у журналі	1	2	3	4	5	6	7	8	9	10
Вхідні дані	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10
Функціонал помилки	0.1	0.1	0.1	0.01	0.01	0.01	0.1	0.1	0.1	0.1
Параметр впливу	1	1	1	0.1	0.1	0.1	0.01	0.01	0.01	1

№ за списком у журналі	11	12	13	14	15	16	17	18	19	20
Вхідні дані	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10
Функціонал помилки	0.1	0.1	0.1	0.01	0.01	0.01	0.1	0.1	0.1	0.1
Параметр впливу	1	1	1	0.1	0.1	0.1	0.01	0.01	0.01	1

№ за списком у журналі	21	22	23	24	25	26	27	28	29	30
Вхідні дані	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10
Функціонал помилки	0.1	0.1	0.1	0.01	0.01	0.01	0.1	0.1	0.1	0.1
Параметр впливу	1	1	1	0.1	0.1	0.1	0.01	0.01	0.01	1

Таблиця 4.5

Вхідні дані	
T1	[-.3602 -.3770 -.3729 .-3771 .6405 .6600 .4609 .1336 -.2013 -.4344 -.5000 -.3930 -.1647 .0988 .3072 .3960 .3449 .1816 -.0312 -.2189 -.3201]
T2	[-.9602 -.5770 -.0729 .3771 .6405 .6600 .6609 .6336 -.2013 -.4344 -.5000 -.3930 -.1647 .0988 .3072 .3960 .3449 .1816 -.0312 -.2189]
T3	[-.9602 -.5770 -.0729 .3771 .6405 .6600 .4609 .1336 -.2013 -.4344 -.5000 -.3930 -.1647 .0988 .3072 .3960 .3449 .1816 -.0312 -.2189 -.3201 -.3101 -.3201 -.3101]
T4	[-.9602 -.5770 -.0729 .3771 .6405 .6600 .4609 .1336 -.5013 -.5344 -.5000 -.5930 -.1647 .0988 .3072 .3960 .3449 .1816 -.0312 -.2189 -.3201]
T5	[-.9602 -.5770 -.0729 .3771 .6405 .6600 .4609 .1336 -.2013 -.4344 -.5000 -.3930 -.1647 .0988 .3072 .3960 .3449 .1816 -.0312 -.2189 -.3201 -.3301 -.3901 -.4201]
T6	[-.9602 -.5770 -.0729 .3771 .6405 .6600 .4609 .1336 -.2013 -.4344 -.5000 -.3930 -.1647 .0988 .3072 .3960 .3449 .1816 -.0312 -.2189 -.3201]
T7	[-.9602 -.5770 -.0729 .3771 .6405 .6600 .4609 .1336 -.2013 -.4344 -.5000 -.3930 -.1647 .0988 .3072 .3960 .3449]
T8	[.3771 .6405 .6600 .4609 .1336 -.2013 -.4344 -.5000 -.3930 -.1647 .0988 .3072 .3960 .3449 .1816 -.0312 -.2189 -.3201]

T9	[.0329 .3771 .6405 .6600 .4609 .1336 -.2013 -.4344 -.5000 -.3930 -.1647 .0988 .3072 .3960 .3449 .1816 -.0312 -.2189 -.3201]
T10	[-.9602 -.5770 -.0729 .3771 .6405 .6600 .4609 .1336 -.2013 -.4344 -.5000 -.3930 -.1647 .0988 .3072 .3960 .3449 .-1816 -.0312 -.2189 -.3201 -.3421 -.3602]

Графік апроксимаційної залежності для нових даних (подібний до рис. 4.9) занесіть у звіт (рис.2 звіту)

Зробіть висновок про гладкість функції та якість апроксимації. Висновок запишіть у звіт.

Завдання 5.2. Дослідження впливу параметра SPREAD на структуру радіальної базисної мережі і якість апроксимації

Створіть та запишіть у файл моделі (RBM3.m) радіальну нейрону мережу з такими параметрами:

```
clear all
P = -1:.1:1;
T = [-.9602 -.5770 -.0729  .3771  .6405  .6600  .4609 ...
     .1336 -.2013 -.4344 -.5000 -.3930 -.1647  .0988 ...
     .3072  .3960  .3449  .1816 -.0312 -.2189 -.3201];
GOAL = 0.01; % Допустиме значення функціоналу помилки
SPREAD = 0.01; % Параметр впливу
net = newrb (P, T, GOAL, SPREAD); % Створення мережі
net.layers {1} .size% Число нейронів в прихованому шарі

plot (P, T, '+ k')% Точки навчальної множини
hold on;
X = -1: .01: 1;
Y = sim (net, X); % Моделювання мережі
plot (X, Y);

ans = 13
```

Зверніть увагу, що параметр впливу SPREAD тут встановлений рівним 0.01. Це означає, що діапазон перекриття вхідних значень становить лише ±0.01, а оскільки навчальні входи задані з інтервалом 0.1, то вхідні значення функціями активації не перекриваються. Це призводить до того, що, по-перше, збільшується кількість нейронів прихованого шару з 6 до 13, а по-друге, не забезпечується необхідної гладкості функції, що апроксимується (рис.5.8):

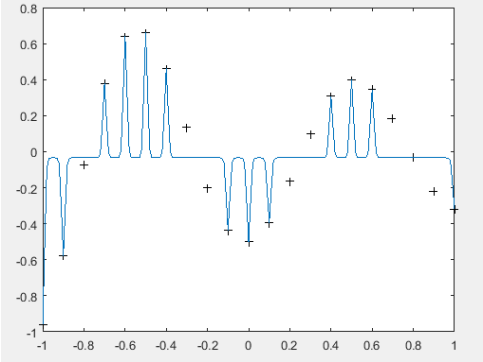


Рис. 4.10

Змінюйте параметр SPREAD кожен раз та запускайте модель з новими параметрами відповідно до таблиці 4.6. ***Данні про кількість нейронів для кожного випадку N занесіть у таблицю 1 звіту***.

Таблиця 4.6

SPREAD	0.01	0.05	0.1	1	5	10	15
N							

Уважно спостерігайте за графіками наскільки близько мережа апроксимує вхідні дані (наскільки близько графік проходить до +).

Графік апроксимаційної залежності для SPREAD = 0.05 та 15 занесіть у звіт (рис.3 та 4 звіту).

Зверніть увагу, коли параметр впливу SPREAD вибирається досить великим (в даному прикладі - 10 або більше), то всі функції активації перекриваються і кожен базовий нейрон видає значення, близьке до 1, для всіх значень входів. Це призводить до того, що мережа не реагує на вхідні значення. Функція newrb намагатиметься будувати мережу, але не зможе забезпечити необхідну точність через обчислювальні проблеми. В процесі обчислень виникають труднощі з оберненням матриць, і про це можуть видаватися попередження; кількість нейронів прихованого шару встановлюється великою (до 21), а точність апроксимації виявляється неприпустимо низькою.

Тобто, параметр впливу SPREAD слід вибирати більшим, ніж крок розбиття інтервалу подання навчальної послідовноності, але меншим розміру самого інтервалу.

Порівняйте графіки на рис.1-4 звіту.

Зробіть висновок з виконаного дослідження який параметр впливу SPREAD слід вибирати для даного завдання. Висновок запишіть у звіт.

Завдання 5.3. Дослідження ітераційної процедури формування радіальних мереж

Застосуйте функцію newrb для створення радіальної базисної мережі.

```
P = -1: .1: 1;
T = [-.9602 -.5770 -.0729 .3771 .6405 .6600 .4609 .1336 ...
    -.2013 -.4344 -.5000 -.3930 -.1647 .0988 .3072 .3960 ...
     .3449 .1816 -.0312 -.2189 -.3201];
plot (P, T, '* r', 'MarkerSize', 4, 'LineWidth', 2)
hold on
% Створення мережі
GOAL = 0.01; % Допустиме значення функціоналу помилки
net = newrb (P, T, GOAL); % Створення радіальної базисної мережі
net.layers {1} .size% Число нейронів в прихованому шарі

% Моделювання мережі
V = sim (net, P); % Вектори входу з навчальної множини
plot (P, V, 'ob', 'MarkerSize', 5, 'LineWidth', 2)
p = [-0.75 -0.25 0.25 0.75];
v = sim (net, p); % Новий вектор входу
plot (p, v, '+ k', 'MarkerSize', 10, 'LineWidth', 2)
```

Відкрийте панель налаштувань графіка (поз. 1 на рис. 4.11)

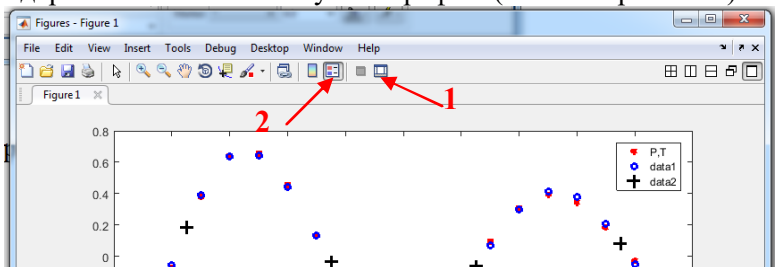


Рис. 4.11

Введіть назви графіків для зірочок (P,T), кружечків (P,V) та хрестиків (p,v) – рис. 4.12.

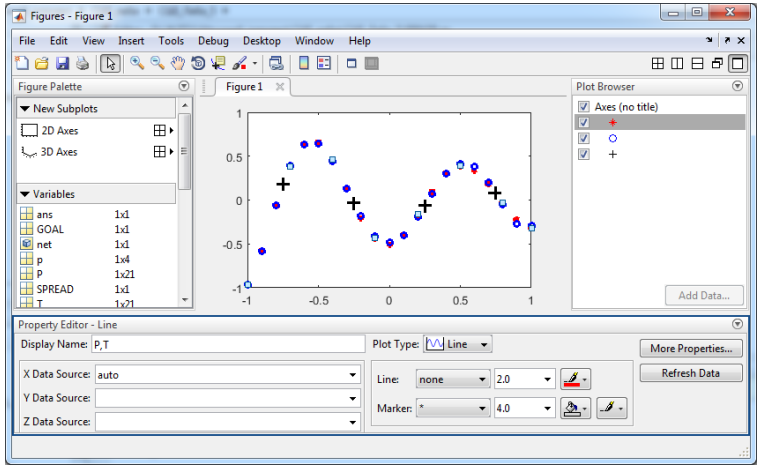


Рис. 4.12

Відповідний підписаний графік з легендою (легенда відображається рис. 4.11 позн. - 2) занесіть у звіт рис. 5 звіту.

У висновки запишіть словами, що зображено на графіках, число нейронів у прихованому шарі.

Завдання 5.4. Дослідження мереж GRNN

Задайте наступну навчальну множину векторів входу і цілей та побудуйте мережу GRNN:

```
P = [4 5 6];
T = [1.5 3.6 6.7];
net = newgrnn (P, T);
net.layers {1} .size% Число нейронів в прихованому шарі

ans = 3
```

Ця мережа має 3 нейрона в прихованому шарі. Промоделюємо побудовану мережу спочатку для одного входу, а потім для послідовності входів з інтервалу [4 7]:

```
p = 4.5;
v = sim (net, p);
p1 = 4: 0.1: 7;
v1 = sim (net, p1);
```

```
plot (P, T, '* k', p, v, 'ok', p1, v1, '- k', 'MarkerSize', 10, 'LineWidth', 2)
```

Зауважте, що для мережі GRNN розмір введеного вектора може відрізнятися від розміру векторів, використовуваних в навчальній послідовності. Крім того, в даному випадку апроксимуюча функція може мати відчутні відмінності від значень, що відповідають навчальній послідовності.

Для створення мережі GRNN використовується функція newgrnn. Приймемо значення параметра впливу SPREAD трохи меншим, ніж крок завдання аргументу функції (в даному випадку 1), щоб побудувати апроксимуючу криву, близьку до заданих точок. Як було досліджено раніше, чим менше значення параметра SPREAD, тим ближче точки апроксимуючої кривої до заданих, але тим менш гладкою є сама крива.

Визначимо навчальну множину у вигляді масивів P і T.

```
P = [1 2 3 4 5 6 7 8];
T = [0 1 2 3 2 1 2 1];
spread = 0.7;

net = newgrnn (P, T, spread);
net.layers {1} .size% Число нейронів в прихованому шарі

ans = 8

A = sim (net, P);
plot (P, T, '* k', 'markersize', 10)
hold on,
plot (P, A, 'ok', 'markersize', 10);
```

Результат показаний на рис. 4.13.

Моделювання мережі для діапазону значень аргументу дозволяє побачити всю апроксимуючу криву, причому можлива екстраполяція цієї кривої за межі області її визначення. Для цього поставимо інтервал аргументу в діапазоні [-1 10]:

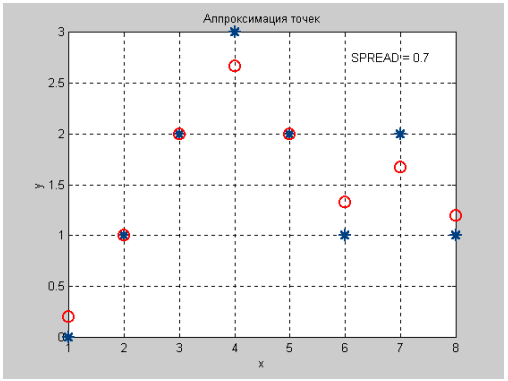


Рис. 4.13

```
P2 = -1: 0.1: 10;
A2 = sim (net, P2);
plot (P2, A2, '- k', 'linewidth', 2)
hold on,
plot (P, T, '* k', 'markersize', 10)
```

Результат показаний на рис. 4.14.

Сформована мережа GRNN використовує всього 8 нейронів в прихованому шарі і досить успішно вирішує завдання апроксимації та екстраполяції нелінійної залежності, відновлених по експериментальних точках.

Виконайте апроксимацію та екстраполяцію функцій мережею GRNN для даних вашого варіанту відповідно до списку за журналом згідно таблиці 4.7.

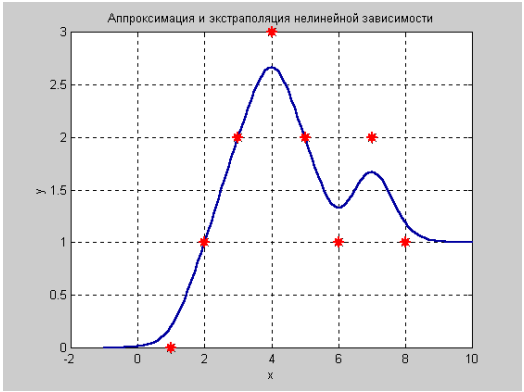


Рис. 4.14

Таблиця 4.7

№	1	2	3	4
P	1 2 3 4 5 6 7 8 9 10	1 2 3 4 5 6 7 8 9 10	1 2 3 4 5 6 7 8 9 10	1 2 3 4 5 6 7 8 9 10
T	0 1 2 3 2 1 2 3 4 5	0 1 2 3 2 1 2 3 4 5	2 1 2 3 4 1 2 3 4 5	0 1 2 2 2 1 2 3 4 6
spread	0.7	0.5	0.8	0.7
P2	[-1 12]	[-1 12]	[-1 12]	[-1 12]

№	5	6	7	8
P	1 2 3 4 5 6 7 8 9	2 3 4 5 6 7 8 9 10	2 3 4 5 6 7 8 9 10	2 3 4 5 6 7 8 9 10
T	0 1 2 3 2 1 2 3 4	1 2 3 2 1 2 3 4 5	1 0 3 2 1 2 3 4 1	1 2 4 2 0 2 3 4 5
spread	0.7	0.5	0.8	0.7
P2	[-1 12]	[-1 12]	[-1 12]	[-1 12]

№	9	10	11	12
P	1 2 3 4 5 6 7 8 9 10	1 2 3 4 5 6 7 8 9 10	1 2 3 4 5 6 7 8 9 10	1 2 3 4 5 6 7 8 9 10
T	3 1 2 3 2 1 2 3 4 5	1 1 2 3 2 1 2 3 4 5	2 1 2 3 4 1 2 3 4 5	0 1 2 3 2 1 2 3 4 6
spread	0.7	0.5	0.8	0.7
P2	[-1 13]	[-1 13]	[-1 13]	[-1 13]

№	13	14	15	16
P	1 2 3 4 5 6 7 8 9	2 3 4 5 6 7 8 9 10	2 3 4 5 6 7 8 9 10	2 3 4 5 6 7 8 9 10
T	0 1 2 3 2 1 2 3 4	1 2 3 2 1 2 3 4 5	1 0 3 2 1 2 3 4 1	1 2 4 2 0 2 3 4 5
spread	0.7	0.5	0.8	0.7
P2	[-1 12]	[-1 12]	[-1 12]	[-1 12]

№	17	18	19	20
P	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9
T	1 1 2 3 2 1 2 3 4	0 1 2 3 4 1 2 3 4	2 1 2 3 4 1 2 3 4	2 1 2 4 2 1 2 3 4
spread	0.7	0.5	0.8	0.7
P2	[-1 12]	[-1 12]	[-1 12]	[-1 12]

№	21	22	23	24
P	1 2 3 4 5 6 7 8 9	2 3 4 5 6 7 8 9 10	2 3 4 5 6 7 8 9 10	2 3 4 5 6 7 8 9 10
T	2 1 2 3 2 1 2 3 1	0 2 3 2 1 2 3 4 4	2 0 3 2 1 2 3 4 1	1 2 4 3 0 2 3 4 3
spread	0.7	0.5	0.8	0.7
P2	[-1 12]	[-1 12]	[-1 12]	[-1 12]

Отримані в результаті графіки для своїх даних подібні до рис. 4.13 та 4.14 занесіть у звіт (рис. 6 та 7 звіту). У висновках укажіть про якість апроксимації та інтерполяції та кількість нейронів у прихованому шарі.

Завдання 5.5. Дослідження мереж PNN

Для створення нейронної мережі PNN призначена М-функція newpnn.

Дано три двоелементних вхідних вектора X і пов'язаних з ними класів Tc. Необхідно створити ймовірнісну нейронну мережу, яка класифікує ці вектори належним чином

```
X = [1 2; 2 2; 1 1]';
Tc = [1 2 3];
plot(X(1,:),X(2,:),'.','markersize',30)
for i = 1:3, text(X(1,i)+0.1,X(2,i),sprintf('class %g',Tc(i))), end
axis([0 3 0 3])
title('Three vectors and their classes.')
xlabel('X(1,:)')
ylabel('X(2,:)') % Рис.4.15
```

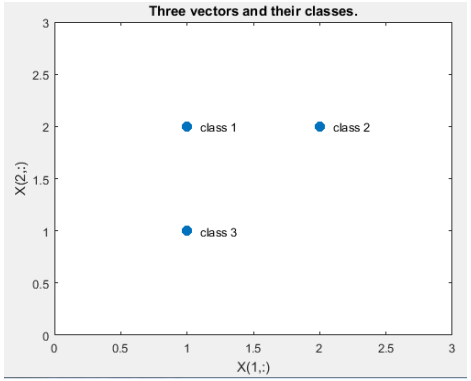


Рис. 4.15

Вектор Tc назвемо вектором індексів класів. Цьому індексному вектору можна поставити у відповідність матрицю зв'язності **T** у вигляді розрідженої матриці. Спочатку ми перетворюємо цільові індекси Tc у вектори **T**. Потім проектуємо ймовірнісну нейронну мережу з NEWPNN. Використовуємо значення SPREAD у 1, оскільки це типова відстань між вхідними векторами.

```
T = ind2vec(Tc);
spread = 1;
net = newpnn(X,T,spread);
```

Масиви X і T задають навчальну множину, що дозволяє виконати формування мережі, промоделювати її, використовуючи масив входів P, і упевнитися, що мережа правильно вирішує завдання класифікації на елементах навчальної множини. У результаті моделювання мережі формується матриця зв'язності, відповідна масиву векторів входу. Для того щоб перетворити її в індексний вектор, призначена M-функція vec2ind.

Тепер ми тестуємо мережу на проектування вхідних векторів. Ми робимо це, імітуючи мережу і перетворюючи її векторні виходи в індекси.

```
Y = net(X);
Yc = vec2ind(Y);
plot(X(1,:),X(2,:),'.','markersize',30)
axis([0 3 0 3])
for i = 1:3,text(X(1,i)+0.1,X(2,i),sprintf('class %g',Yc(i))),end
title('Testing the network.')
xlabel('X(1,:)')
ylabel('X(2,:)')
```

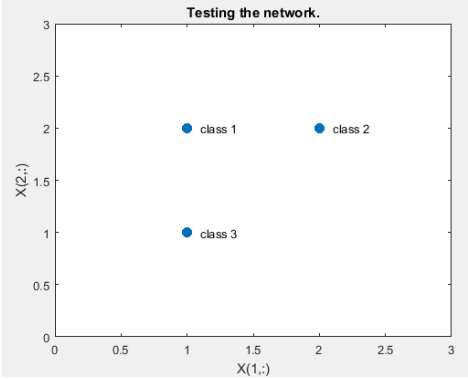


Рис. 4.16

Класифікуємо новий вектор у нашій мережі.

```
x = [2; 1.5];
y = net(x);
ac = vec2ind(y);
hold on
plot(x(1),x(2),'.','markersize',30,'color',[1 0 0])
text(x(1)+0.1,x(2),sprintf('class %g',ac))
hold off
title('Classifying y new vector.')
xlabel('X(1,:) and x(1)')
ylabel('X(2,:) and x(2)')
```

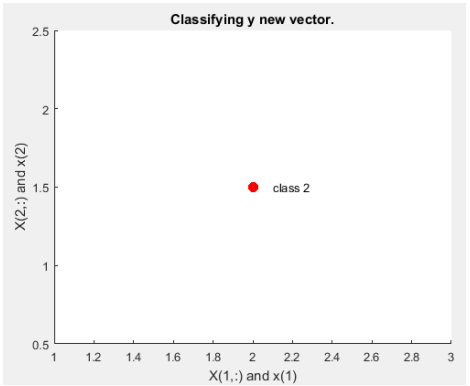



Рис. 4.17

Наступна діаграма показує, як імовірнісна нейронна мережа ділить вхідний простір на три класи.

```
x1 = 0:.05:3;
x2 = x1;
[X1,X2] = meshgrid(x1,x2);
xx = [X1(:) X2(:)]';
yy = net(xx);
yy = full(yy);
m = mesh(X1,X2,reshape(yy(1,:),length(x1),length(x2)));
m.FaceColor = [0 0.5 1];
m.LineStyle = 'none';
hold on
m = mesh(X1,X2,reshape(yy(2,:),length(x1),length(x2)));
m.FaceColor = [0 1.0 0.5];
m.LineStyle = 'none';
m = mesh(X1,X2,reshape(yy(3,:),length(x1),length(x2)));
m.FaceColor = [0.5 0 1];
m.LineStyle = 'none';
plot3(X(1,:),X(2,:),[1 1 1]+0.1,'.','markersize',30)
plot3(x(1),x(2),1.1,'.','markersize',30,'color',[1 0 0])
hold off
view(2)
title('The three classes.')
xlabel('X(1,:) and x(1)')
ylabel('X(2,:) and x(2)')
```

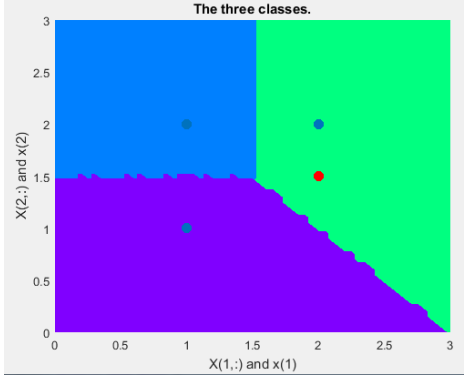


Рис. 4.18

Проведіть дослідження для даних свого варіанта (табл. 4.8)

Таблиця 4.8

№	**1**	**2**	**3**	**4**
X	[1 2; 2 2; 1 2]	[1 2; 2.2 2; 1 2]	[1.5 2; 2 2; 1 2]	[1 2.5; 2 2; 1 2]
spread	0.7	1	0.8	0.7

№	**5**	**6**	**7**	**8**
X	[1.5 2; 2 2; 1 2]	[1 2; 2.5 2; 1 2]	[1.5 2; 2.3 2; 1 2]	[1 2.5; 2 2; 1.7 2]
spread	1	0.9	0.6	0.9

№	**9**	**10**	**11**	**12**
X	[1 2.3; 2 2; 1 2]	[1.7 2; 2.2 2; 1 2]	[1.3 2; 2 2; 1 2]	[1 2.8; 2 2; 1 2]
spread	0.7	1	0.8	0.7

№	**13**	**14**	**15**	**16**
X	[1.2 2; 1.8 2; 1 2]	[0.5 2; 2.5 2; 1 2]	[0.5 2; 2.3 2; 1 2]	[1 2.5; 2 2; 1.3 2]
spread	1	0.9	0.6	0.9

№	**17**	**18**	**19**	**20**
X	[0.7 2; 2 2; 1 2]	[1.1 2; 2.3 2; 1 2]	[1.6 2; 2 2.1; 1 2]	[1 2.2; 2 2; 1 2]
spread	0.7	1	0.8	0.7

№	**21**	**22**	**23**	**24**
X	[1.3 2; 2 2; 1 2]	[1.1 2; 2.5 2; 1 2]	[1.5 2; 2.3 2; 1 2]	[1 2.6; 2 2; 1.2 2]
spread	1	0.9	0.6	0.9

Отримані в результаті графіки для своїх даних подібні до рис. 4.17 та 4.18 занесіть у звіт (рис. 8 та 9 звіту). У висновках укажіть про якість апроксимації та інтерполяції та кількість нейронів у прихованому шарі.

Завдання 4.6. Дослідження рекурентної нейронної мережі Елмана

Мережу Елмана будемо досліджувати на прикладі задачі детектування амплітуди гармонійного сигналу. Нехай відомо, що на вхід нейронної мережі надходять вибірки з деякого набору синусоїд. Потрібно виділити значення амплітуд цих синусоїд.

Сформуємо вибірки з набору двох синусоїд з амплітудами 1.0 і 2.0:

```
p1 = sin (1:20);
p2 = sin (1:20) * 2;
```

Цільовими виходами такої мережі є вектори

```
t1 = ones(1,20);
t2 = ones(1,20)*2;
```

Сформуємо набір векторів входу і цільових виходів:

```
p = [p1 p2 p1 p2];
t = [t1 t2 t1 t2];
```

Сформуємо навчальну послідовність у вигляді масивів осередків:

```
Pseq = con2seq(p);
Tseq = con2seq(t);
```

У ППП NNT для створення мережі Елмана передбачена М-функція newelm. Задача, що вирішується, вимагає, щоб мережа Елмана на кожному кроці спостереження значень вибірки могла виявити єдиний її параметр - амплітуду синусоїди. Це означає, що мережа повинна мати 1 вхід і 1 вихід:

```
R = 1; % Число елементів входу
S2 = 1;% Число нейронів вихідного шару
```

Рекурентний шар може мати будь-яке число нейронів, і чим складніше завдання, тим біль-шиї кількість нейронів потрібно. Зупинимося на 10 нейронах рекурентного шару:

```
S1 = 10; % Число нейронів рекурентного шару
```

Елементи входу для даного завдання змінюються в діапазоні від -2 до 2. Для навчання використовується метод градієнтного спуску зі збуренням і адаптацією параметра швидкості настройки, реалізований у вигляді М-функції traingdx:

```
net = newelm([–2 2],[S1 S2],{'tansig','purelin'},'traingdx');
```

Мережа використовує такі функції адаптації, ініціалізації, навчання і оцінки якості:

```
adaptFcn: 'adaptwb'
initFcn: 'initlay'
performFcn: 'mse'
trainFcn: 'traingdx'
```

Шари мережі Елмана мають наступні характеристики:

```
net.layers{1}
ans =
dimensions: 10
    distanceFcn: 'dist'
     distances: [10×10 double]
       initFcn: 'initnw'
    netInputFcn: 'netsum'
     positions: [0 1 2 3 4 5 6 7 8 9]
          size: 10
    topologyFcn: 'hextop'
    transferFcn: 'tansig'
      userdata: [1×1 struct]
```

```
net.layers{2}
ans =
dimensions: 1
    distanceFcn: 'dist'
     distances: 0
       initFcn: 'initnw'
    netInputFcn: 'netsum'
     positions: 0
          size: 1
    topologyFcn: 'hextop'
    transferFcn: 'purelin'
      userdata: [1×1 struct]
```

Прихований шар використовує функцію активації tansig, яка для мережі Елмана приймається за замовчуванням; ініціалізація ваг і зміщень реалізується методом NW (Nguen - Widrow) за допомогою М-функції initnw. Другий шар використовує лінійну функцію активації purelin.

За замовчуванням для настройки ваг і зміщень використовується функція learngdm, а для оцінки якості навчання - функція mse.

Для навчання мережі Елмана використовується М-функція train. Її вхідними аргументами є навчальні послідовності Pseq і Tseq, як метод навчання використовується метод зворотного поширення помилки зі збуренням і адаптацією параметра швидкості настройки. Кількість циклів

навчання приймається рівним 1000, періодичність виведення результатів - 20 циклів, кінцева похибка навчання - 0.01:

```
net.trainParam.epochs = 1000;
net.trainParam.show = 25;
net.trainParam.goal = 0.01;
[net,tr] = train(net,Pseq,Tseq);
```

Отримаємо вікно навчання мережі (рис.5.17). Натиснувши на Performansce отримаємо графік помилки (рис.5.18).

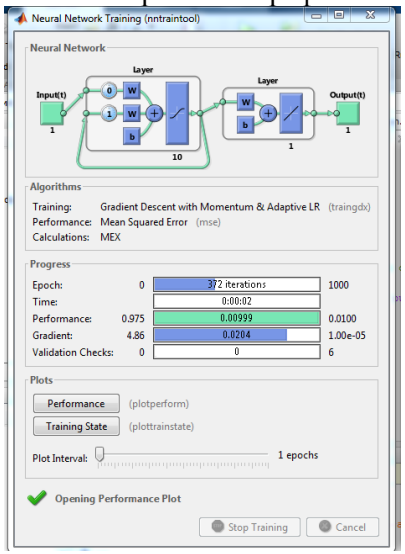


Рис. 4.19

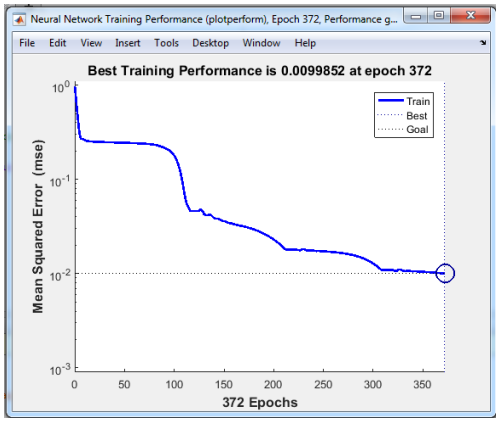


Рис. 4.20

Необхідна точність навчання забезпечується за 372 цикла. Тепер можна перевірити роботу сформованої мережі.

Перевірка мережі

Будемо використовувати для перевірки мережі входи навчальної послідовності:

```
figure(2)
a = sim(net,Pseq);
time = 1:length(p);
plot(time, t, '-', time, cat(2,a{:}))
axis([1 80 0.8 2.2]) % Рис.4.21
```

Увага! Якщо ви копіюєте текст програми з методичних рекомендацій через Ctrl+c Ctrl+v то зверніть увагу, що при копіюванні знак «-»

не завжди розпізнається редактором матлаба. Якщо не виводиться графік то у функції `plot()` *перенаберіть знак* `'-'`.

На рис. 4.21 наведені графіки вхідного і вихідного сигналів.

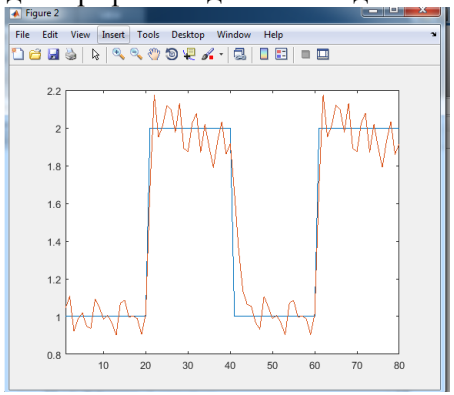


Рис. 4.21

Отриманий графік подібний до рис. 4.21 занесіть у звіт (рис. 10 звіту)

Як випливає з аналізу малюнка, мережа справляється з вирішенням задачі детектування амплітуди на наборах навчальної множини. Однак неясно, як вона буде вести себе на інших наборах входу. Чи володіє побудована мережа Елмана властивістю узагальнення? Спробуємо перевірити це, виконавши такі дослідження.

Подамо на мережу набір сигналів, складений з двох синусоїд з амплітудами 1.6 і 1.2 відповідно:

```
p3 = sin(1:20)*1.6;
t3 = ones(1,20)*1.6;
p4 = sin(1:20)*1.2;
t4 = ones(1,20)*1.2;
pg = [p3 p4 p3 p4];
tg = [t3 t4 t3 t4];
pgseq = con2seq(pg);
figure(3)
a = sim(net,pgseq);
ime = 1:length(pg);
plot(time, tg, '-', time, cat(2,a{:}))
axis([1 80 0.8 2.2])
```

Результат представлений на рис. 4.22.

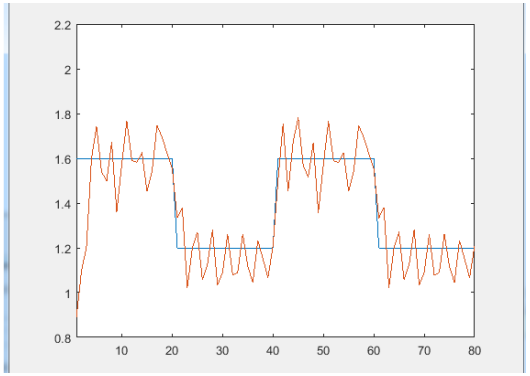


Рис. 4.22

На цей раз мережа гірше справляється із завданням. Мережа прагне детектувати значення амплітуди, але робить це не дуже точно. Покращене узагальнення могло бути отримано, навчаючи мережу на більшу кількість амплітуд, ніж тільки на значення 1.0 і 2.0. Використання трьох або чотирьох гармонійних сигналів з різними амплітудами може привести до набагато кращого детектора амплітуд.

Поміняйте значення амплітуди для даних свого варіанту відповідно до таблиці 4.9. Проведіть детектування відповідно до своїх даних.

Таблиця 4.9

№ за списком у журналі	1	2	3	4	5	6	7	8	9	10
`p3, t3`	2.1	2.0	1.9	1.8	1.7	1.6	1.5	1.4	1.3	1.2
`p4, t4`	1.1	1.1	1.1	1.1	1.1	1.1	1.1	1.1	1.1	1.1

№ за списком у журналі	11	12	13	14	15	16	17	18	19	20
`p3, t3`	2.1	2.0	1.9	1.8	1.7	1.6	1.5	1.4	1.8	1.6
`p4, t4`	1.2	1.2	1.2	1.2	1.2	1.2	1.2	1.2	1.2	1.2

№ за списком у журналі	21	22	23	24	25	26	27	28	29	30
`p3, t3`	2.1	2.0	1.9	1.8	1.7	1.6	1.5	1.6	2.0	2.1
`p4, t4`	1.3	1.3	1.3	1.3	1.3	1.3	1.3	1.3	1.3	1.3

Отриманий графік для своїх даних подібний до рис. 4.22 занесіть у звіт (рис. 11 звіту). Зробіть висновок про якість детектування для свого варіанту.

Завдання 4.7. Дослідження рекурентної нейронної мережі Хопфілда

Припустимо, що потрібно створити модифіковану мережу Хопфілда з двома точками рівноваги, заданими в тривимірному просторі:

```
T = [-1 -1 1; 1 -1 1] '
T =
   -1 1
   -1 -1
    1 + 1
```

Виконаємо синтез мережі, використовуючи М-функцію newhop:

```
net = newhop (T);
```

Переконаємося, що розроблена мережа має стійкі стани в цих двох точках. Виконаємо моделювання мережі Хопфілда, взявши до уваги, що ця мережа не має входів і містить рекурентний шар; в цьому випадку цільові вектори визначають початкове перебування шару Ai, а другий аргумент функції sim визначається числом цільових векторів:

```
      Ai = T;
      [Y,Pf,Af] = sim(net,2,[],Ai);
      Y
Y =
  -1 1
  -1 -1
   1 + 1
```

Дійсно, стійкі положення рівноваги мережі знаходяться в призначених точках.

Задамо іншу початкову умову у вигляді масиву осередків:

```
      Ai = {[-0.9; -0.8; 0.7]};
      Ai{1,1}
ans =
  -0.9000
  -0.8000
   0.7000
```


Ця точка розташована поблизу першого положення рівноваги, так що можна очікувати, що мережа буде сходитися саме до цієї точки. При такому способі виклику функції sim в якості другого параметра вказується такт дискретності і кількість кроків моделювання:

```
      [Y, Pf, Af] = sim (net, {1 5}, {}, Ai);
      Y{1}
Y =
  -1
  -1
   1
```

Дійсно, з заданого початкового стану мережу повернулася в стійке положення рівноваги. Бажано, щоб мережа поводилася аналогічним чином при задаванні будь-якої початкової точки в межах куба, вершини якого складені з усіх комбінацій чисел 1 і -1 в тривимірному просторі. На жаль, цього не можна гарантувати, і досить часто мережі Хопфілда включають небажані паразитні точки рівноваги.

Розглянемо мережу Хопфілда з чотирма нейронами і визначимо 4 точки рівноваги у вигляді наступного масиву цільових векторів:

```
      T = [1 –1; –1 1;  1 1; –1 –1]'
T =
     1 -1 1 -1
    -1  1 1 -1
```

На рис. 4.23 показані ці 4 точки рівноваги на площині станів мережі Хопфілда.

```
plot(T(1,:), T(2,:),'*r') % Рис.5.21
axis([–1.1 1.1 –1.1 1.1])
title('Точки рівноваги мережі Хопфілда')
xlabel('a(1)'), ylabel('a(2)')
```

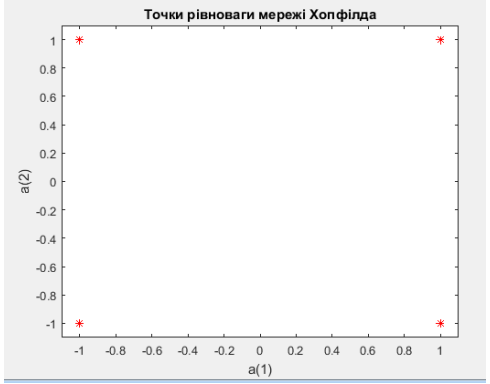


Рис. 4.23

Розрахуємо ваги і зміщення модифікованої мережі Хопфілда, використовую М-функцію newhop:

```
net = newhop (T);
W = net.LW {1,1}
b = net.b {1,1}
W =
    1.1618 0
         0 1.1618
b =
  3.5934e-017
  3.5934e-017
```

Перевіримо, чи належать вершини квадрата до мережі Хопфілда:

```
Ai = T;
[Y,Pf,Af] = sim(net,4,[],Ai)

Y =
1 -1 1 -1
-1 1 + 1 -1
Pf = []
Af =
1 -1 1 -1
-1 1 + 1 -1
```

Як і слід було очікувати, виходи мережі рівні цільовим векторам.

Тепер перевіримо поведінку мережі при випадкових початкових умовах:

```
plot(T(1,:), T(2,:),'*r'), hold on
axis([–1.1 1.1 –1.1 1.1])
```



```
xlabel('a(1)'), ylabel('a(2)')
new = newhop(T);
[Y,Pf,Af] = sim(net,4,[],T);
for i =1:25
   a = {rands(2,1)};
   [Y,Pf,Af] = sim(net,{1,20},{},a);
   record = [cell2mat(a) cell2mat(Y)];
   start = cell2mat(a);
  plot(start(1,1), start(2,1), 'kx', record(1,:), record(2,:))
end
```

Результат представлений на рис. 4.24.

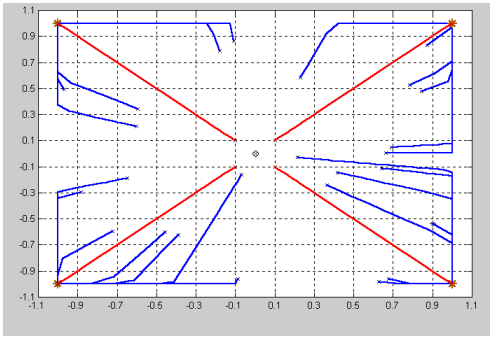


Рис. 4.24

Отриманий графік подібний до рис. 4.24 занесіть у звіт (рис. 12 звіту).

Розробка двохнейронної мережі Хопфілда

Мережа Хопфілда, що складається з двох нейронів, спроектована з двома стійкими точками рівноваги і моделюється з використанням наведених вище функцій.

Ми хотіли б отримати мережу Хопфілда, яка має дві стабільні точки, визначені двома цільовими векторами в T.

```
T = [+1 -1; ...
    -1 +1];
```

Визначимо ділянку, де стабільні точки показані в кутах. Всі можливі стани 2-нейронної мережі Хопфілда містяться в межах кордонів ділянок.

```
plot(T(1,:),T(2,:),'r*')
axis([-1.1 1.1 -1.1 1.1])
```

```
title('Hopfield Network State Space')
xlabel('a(1)');
ylabel('a(2)');
```

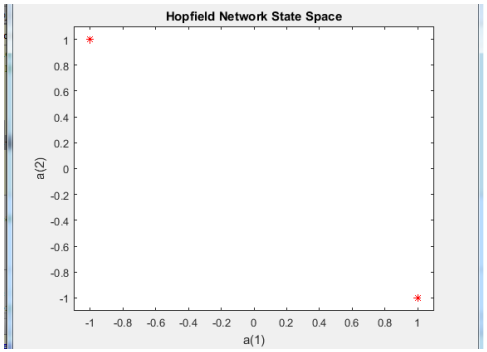


Рис. 4.25

Функція NEWHOP створює мережі Хопфілда з урахуванням стійких точок T.

```
net = newhop (T);
```

Спочатку ми перевіряємо, що цільові вектори дійсно стабільні. Ми перевіряємо це, надаючи цільові вектори мережі Hopfield. Вона повинна повернути ці цілі незмінними, і насправді вона їх повертає.

```
[Y,Pf,Af] = net([],[],T);
Y
```

Далі ми визначаємо випадкову початкову точку і моделюємо мережу Хопфілда на 20 кроків. Вона повинна досягти однієї зі своїх стійких точок.

```
a = {rands (2,1)};
[y, Pf, Af] = net ({20}, {}, a);
```

Ми можемо зробити графік діяльності мереж Хопфілда.

Звичайно, мережа закінчується в лівому верхньому або нижньому правому куті ділянки.

```
record = [cell2mat(a) cell2mat(y)];
start = cell2mat(a);
hold on
```



```
plot(start(1,1),start(2,1),'bx',record(1,:),record(2,:))
```

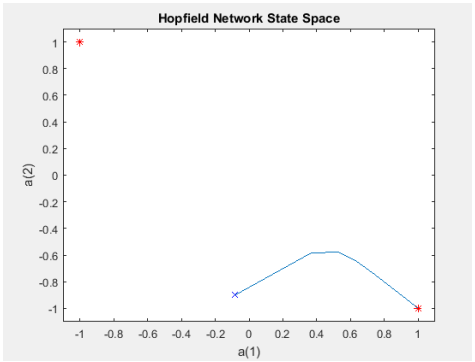


Рис. 4.26

Повторюємо моделювання для ще 25 початкових умов.

Зверніть увагу, що якщо мережа Хопфілда починається ближче до верхнього лівого, вона перейде в лівий верхній і навпаки. Ця здатність знаходити найближчу пам'ять до початкового входу - це те, що робить мережу Hopfield корисною.

```
color = 'rgbmy';
for i=1:25
  a = {rands(2,1)};
  [y,Pf,Af] = net({20},{},a);
  record=[cell2mat(a) cell2mat(y)];
  start=cell2mat(a);
  plot(start(1,1),start(2,1),'kx',record(1,:),record(2,:),color(rem(i,5)+1))
end
```

Отриманий графік подібний до рис. 4.27 занесіть у звіт (рис. 13 звіту).

Зробіть висновки по роботі у яких укажіть для виконання яких завдань застосовуються:

радіальні базисні нейронні мережі
нейронні мережі GRNN
нейронні мережі PNN
нейронні мережі Елмана
нейронні мережі Хопфілда.

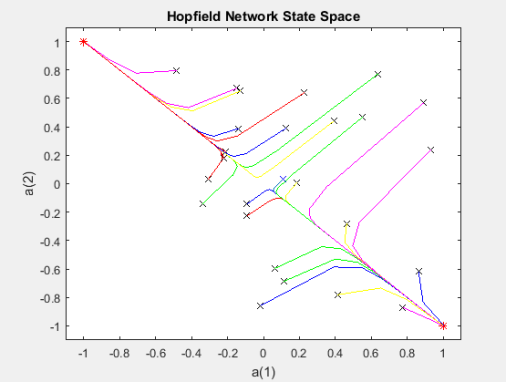


Рис. 4.27

ЗВІТНІСТЬ ЗА ЛАБОРАТОРНУ РОБОТУ № 4

У звіті з лабораторної роботи необхідно представити всі графіки та висновки згідно завдання.

Назвіть звіт ШІ-КБ-ЛР-4-NNN-XXXXX.doc
де NNN – номер групи
XXXXX – позначення прізвища студента.

Переконвертуйте файл звіту в ШІ-КБ- ЛР-4-NNN-XXXXX.pdf

Надішліть звіт викладачу на електронну пошту.

ПИТАННЯ ДЛЯ САМОКОНТРОЛЮ

1. Які нейронні мережі називаються рекурентними?
2. Які мережі називаються радіальними базисними нейронними мережами?
3. Особливості архітектури та застосування радіальної базисної нейронної мережі?
4. Особливості архітектури та застосування нейронної мережі GRNN?
5. Особливості архітектури та застосування нейронної мережі PNN?
6. Особливості архітектури та застосування нейронної мережі Елмана?
7. Особливості архітектури та застосування нейронної мережі Хопфілда?

ЗАКІНЧЕННЯ

Споконвічне змагання «щита і меча» в плані використання штучного інтелекту для захисту від кібератак і для посилення їх потужності, вже йде і його інтенсивність буде швидко наростати. Від того, хто більше досягне успіху в цій гонці, багато в чому залежить безпека ІТ-систем вже в найближчі роки.

Із-за досить великого обсягу матеріалу методичну розробку довелося розділити на дві частини.

У частині 1 викладено теоретичний матеріал для підготовки, завдання та методичні рекомендації до виконання лабораторних робіт № 1-4 за темами «Нечіткі множини» та «Штучні нейронні мережі».

У частині 2 методичних рекомендацій планується доповнити матеріал за темами: «Генетичні алгоритми», «Машинне навчання» та «Комп'ютерний зір».

ЛІТЕРАТУРА

1. Методи та системи штучного інтелекту: Навчальний посібник для студентів напряму підготовки 6.050101 «Комп'ютерні науки» / Уклад. : А.С. Савченко, О. О. Синельніков. – К. : НАУ, 2017. – 190 с.

2. Кібербезпека та інформаційні технології : монографія. – Х. : ТОВ «ДІСА ПЛЮС», 2020. – 380 с.

3. Кондратенко Ю. П. Нечіткі множини та нечітка логіка. Методичні рекомендації та вказівки для виконання лабораторних робіт студентами спеціальності 122 «Комп'ютерні науки» / Ю. П. Кондратенко, Г. В. Кондратенко, Є. В. Сіденко ; під ред. д-р техн. наук, професор Ю. П. Кондратенка. – Миколаїв : ЧНУ ім. Петра Могили, 2019. – 36 с.

4. Навчальний посібник «Методи та системи штучного інтелекту» для студентів спеціальності 122 «Комп'ютерні науки та інформаційні технології» / В.О. Трусов, І.М. Удовик, Г.М. Коротенко, Л.М. Коротенко, А.Т. Харь. – Д.: Державний ВНЗ «Національний гірничий університет», 2017. – 112 с.

5. Медведев В. С., Потемкин В. Г. Нейронные сети. MATLAB 6./Под общ. ред. В. Г. Потемкина. – М.: ДИАЛОГ-МИФИ, 2001. – 630 с. – (Пакеты прикладных программ; Кн. 4).

6. Антоненко В. М. Сучасні інформаційні системи і технології: управління знаннями : навчальний посібник / В. М. Антоненко, С. Д. Мамченко, Ю. В. Рогушина. – Ірпінь : Національний університет ДПС України, 2016. – 212 с.

7. Кононюк А. Ю. Нейроні мережі і генетичні алгоритми – К.:«Корнійчук», . 2008. – 446 с.

8. В.Дьяконов, В.Круглов. Математические пакеты расширения MATLAB. Специальный справочник. СанПТб, Питер, 2011 - 470 с.