МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ
«ХАРКІВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ»

МЕТОДИЧНІ ВКАЗІВКИ

до виконання лабораторної роботи
«Побудова та навчання радіально-базисної (RBF) мережі у середовищі MATLAB. Неконтрольоване навчання мережі Кохонена»

з навчальної дисципліни «Вступ до нейронних мереж»
для студентів денної та заочної форм навчання
за спеціальністю F2 «Інженерія програмного забезпечення», F3 «Комп'ютерні науки» та F6 «Інформаційні системи та технології»

Затверджено
редакційно-видавничою радою
університету, протокол № 2 від
26.06.2025 р.

Харків
НТУ «ХПІ»
2025

Методичні вказівки до виконання лабораторної роботи «Формування одношарового персептрону у середовищі MATLAB. Симуляція, навчання, вирішення задачі розпізнавання образу»: з навчальної дисципліни «Вступ до нейронних мереж» для студентів денної та заочної форм навчання за спеціальністю F2 «Інженерія програмного забезпечення», F3 «Комп'ютерні науки» та F6 «Інформаційні системи та технології»/ уклад.: Н. Л. Чернова, Д. Б. Аркатов; Нац. техн. ун-т «Харків. політехн. ін-т». – Харків: НТУ «ХПІ», 2025. – 39 с.

Укладачі: Н. Л. Чернова
Д. Б. Аркатов

Рецензент В. В. Москаленко

Кафедра програмної інженерії та інтелектуальних технологій управління

ВСТУП

Нейронна мережа представляє собою математична модель, що навчається шляхом первинного опрацювання великого набору даних, не вимагаючи написання окремого коду під конкретне завдання. Нейронні мережі є одним зі способів машинного навчання, підрозділу штучного інтелекту, і лежать в основі алгоритмів глибокого навчання. Вони здатні шукати закономірності в неструктурованих даних і вирішувати безліч завдань. Останніми роками технологія набула великого розвитку. Переважно її використовують для обробки тексту, відео, аудіо та іншої інформації. Особливої популярності набули нейронні мережі, здатні швидко генерувати зображення з підказки та давати «майже людські» відповіді на запитання або запити природною мовою. Такі моделі не замінюють роботу фахівців, але допомагають оптимізувати рутинні процеси. Тому особливу увагу слід приділити підготовці майбутніх фахівців, знайомих з теоретичними та практичними засадами з розробки сучасних систем, що базується на використанні штучного інтелекту. Запропоноване видання дозволяє студентам без будь-яких базових знань з програмування за час навчання освоїти основні базові команди для побудови та навчання нейронних мереж . Це видання створено на основі окремих практичних завдань з дисципліни «Вступ до нейронних мереж», які виконуються у Національному технічному університеті «Харківський політехнічний інститут» для студентів напряму підготовки F2 «Інженерія програмного забезпечення», F3 «Комп’ютерні науки» та F6 «Інформаційні системи та технології». Виклад матеріалу дозволяє його використати не тільки студентам, а й викладачам під час самостійної підготовки. Кожне завдання лабораторної роботи супроводжується прикладами розв’язання подібних задач. Всі роботи виконувалися в online-версії середовища MATLAB після попередньої реєстрації для отримання відповідного доступу для навчання. Використовуючи доступне програмне забезпечення на власних персональних комп’ютерах і необхідне методичне забезпечення у вигляді розроблених рекомендації для виконання лабораторних робіт, студенти мають можливість виконувати всі етапи самостійно та дистанційно. Сукупність набутих

навичок дозволяє студентам опанувати основи роботи у середовищі MATLAB, різноманітні засоби для навчання нейронних мереж та подальшої симуляції для оцінки результатів. Методичні вказівки є збіркою завдань та інструкцій для закріплення теоретичних знань, отриманих на лекційних заняттях з дисципліни «Вступ до нейронних мереж». Методичні вказівки до виконання лабораторної роботи містять тему, мету, короткі теоретичні відомості, завдання, порядок виконання завдань і список питань для самоперевірки. На захист виконаної студентом лабораторної роботи оформлюється окремий звіт та представляється електронний документ, із виконаним завданням.

Звіт повинен включати наступні пункти:

1. Мета лабораторної роботи.
2. Короткі теоретичні відомості.
3. Індивідуальне завдання (згідно варіанту) лабораторної роботи.
4. Основні етапи виконання роботи та отримані результати.
5. Висновки.

1. RBF МЕРЕЖІ

1.1 Теоретичні відомості

Радіально-базисні мережі реалізують ідею, сформульовану Т. Кавером, яка полягає в тому, що лінійно нероздільна задача розпізнавання образів у просторі Rn може стати лінійно роздільною у просторі вищої розмірності Rp.

Властивості такої мережі повністю визначаються радіально-базисними функціями, що використовуються в нейронах прихованого шару у якості активаційних.

Радиально-базисная функция (РБФ) – це багатовимірна функція, яка залежить від відстані $r=x-c$ між вхідним вектором x і центром c параметру ширини (масштабу) σ, що визначає локальну область вхідного простору, на яку «реагує» дана функція:

$$\varphi(x) = \Phi(\|x - c\|, \sigma) = \Phi(r, \sigma)$$

Таким чином, кожен нейрон прихованого шару обчислює відстань між вхідним вектором та своїм центром і здійснює над ним деяке нелінійне перетворення $\Phi(r,\sigma)$.

Важливо, що на відміну від монотонних активаційних функцій багатошарових мереж, радіально-базисні функції, як правило, симетричні і «накривають» вузьку область вхідного простору.

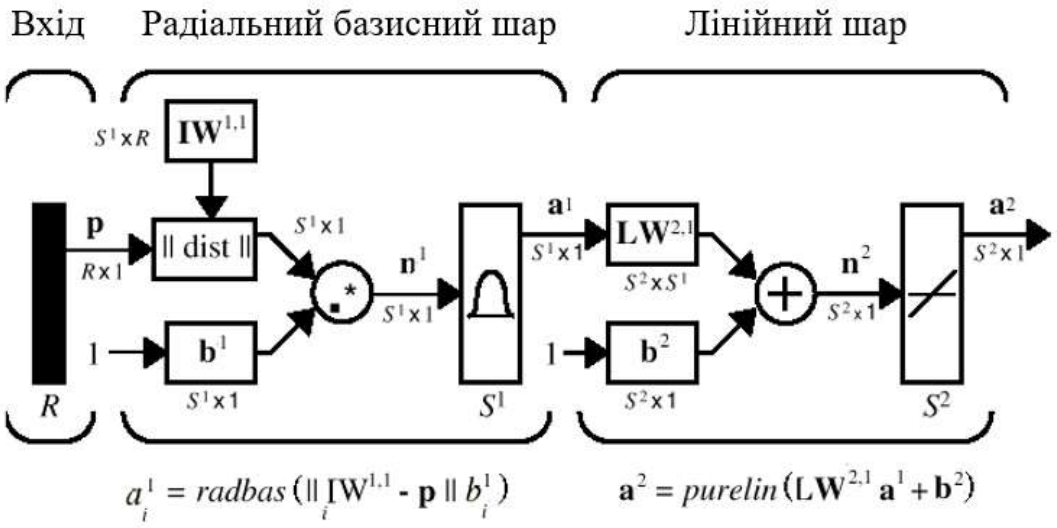
Досить часто радіально-базисні функції мають дзвоноподібну форму.

Найбільшого поширення набули гауссівські функції, що мають пік у центрі с і монотонно спадають у міру віддалення від центру.

В найбільш простій формі РБФ представляє собою НМ, що складається із таких шарів:

- вхідного;
- радіального базисного;
- лінійного.

В задачу вхідного шару входить розподіл вхідних даних по нейронам. Сховані нейрони мають радіально-симетричну функцію активації. Кожен з них призначений для зберігання окремого еталонного образу, який відповідає окремому класу.



Де R – кількість входів мережі;

S^1 – кількість RBF-нейронів у радіальному базисному шарі.

Входами блоку || dist || є вектор входу p і матриця ваг $IW^{1,1}$.

Кожному *i*-му нейрону відповідає *i*-й рядок $IW_i^{1,1}$ матриці ваг.

Виходи блоку || dist || - Евклідові відстані поточного вектору входу p до кожного нейрона RBF-шару (вектор-рядок, що складається з S^1 елементів).

Вихід блоку || dist || множиться поелементно на вектор зміщення b^1 і формує вхід для функції активації.

Вихід першого шару може бути записаний в такій формі:

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

В цьому випадку кожен нейрон радіального базисного шару видасть значення відповідно до того, на скільки близько розташований 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 дорівнювати вазі активного вихідного нейрона. Однак це винятковий випадок, зазвичай вихід формують кілька нейронів з різними значеннями ваг.

Навчання радіально-базисної нейронної мережі може зводиться до одного з таких варіантів:

1) задаються параметри радіальних функцій (центри і радіуси (spreads)), а обчислюються тільки ваги вихідного шару;

2) визначаються шляхом самонавчання (за допомогою методів кластеризації) центри і відхилення, а для корекції ваг вихідного шару використовується навчання з учителем;

3) визначаються всі параметри мережі за допомогою навчання з учителем, наприклад за допомогою методу зворотного поширення помилки.

1.2 Ймовірнісна нейронна мережа PNN (Probabilistic Neural Network)

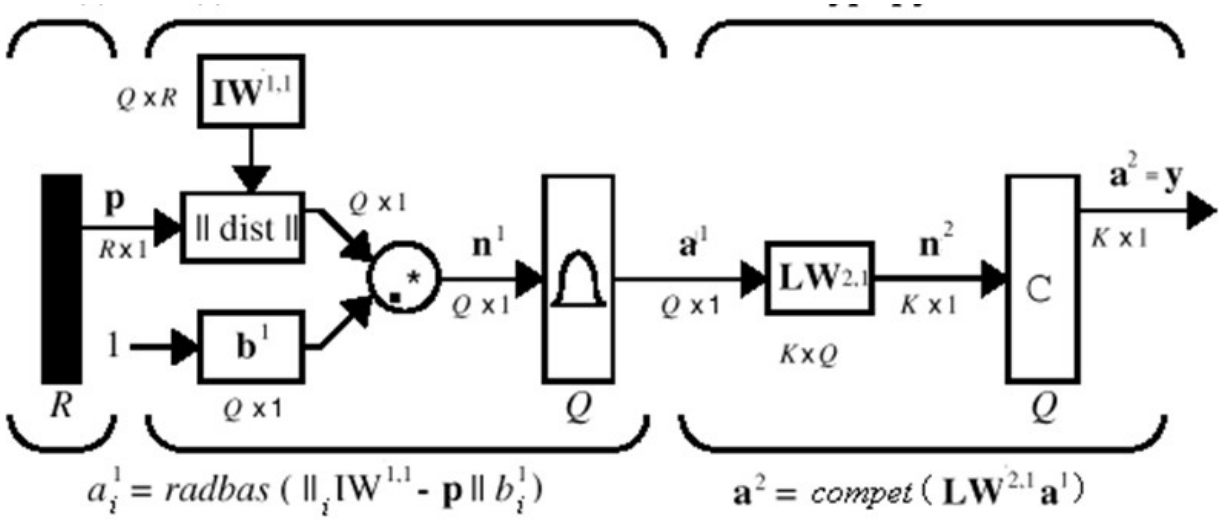
Архітектура мережі PNN базується на архітектурі радіальної базисної мережі, але як другий шар використовують так званий конкуруючий шар, який підраховує ймовірність приналежності вхідного вектора до того чи іншого класу, і, в кінцевому рахунку, зіставляє вектор з тим класом, ймовірність приналежності до якого вище .

Передбачається, що задано навчальну множину, що складається з пар векторів вхід/ціль. Кожен вектор мети має m елементів, які вказують клас приналежності, і, таким чином, входи ставляться у відповідність до одного з m класів.

Найбільш важливі переваги PNN-мереж полягають у тому, що вихідне значення має ймовірнісний зміст (і тому його легше інтерпретувати), і в тому, що мережа швидко навчається. При навчанні такої мережі час витрачається практично тільки на те, щоб подавати їй на вхід навчальні спостереження, і мережа працює настільки швидко, наскільки це можливо.

Істотним недоліком таких мереж є їх обсяг. PNN-мережа фактично вміщує всі навчальні дані, тому вона вимагає багато пам'яті і як наслідок працює повільно.

PNN-мережі особливо корисні при пробних експериментах (наприклад, коли потрібно вирішити, які з вхідних змінних використовувати), оскільки завдяки короткому часу навчання можна швидко зробити велику кількість пробних тестів.



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.

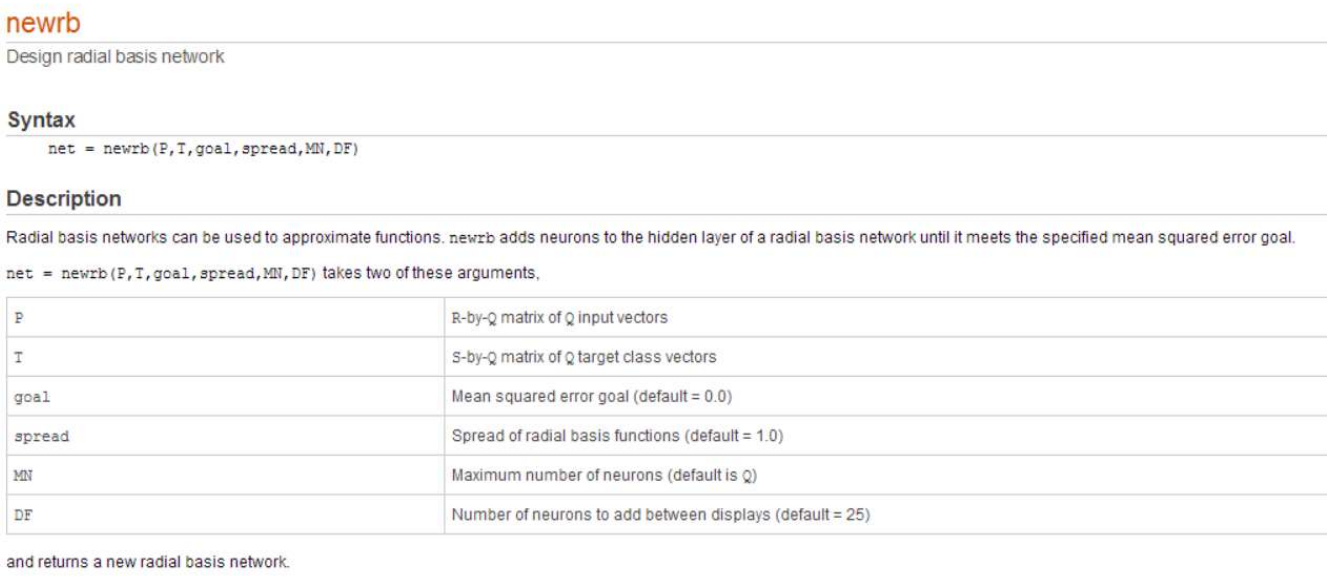
В результаті може бути утворена матриця зв'язності T розміру *K*×*Q*, що складається з нулів і одиниць, рядки якої відповідають класам приналежності, а стовпці - векторам входу. Таким чином, якщо елемент T(*i, j*) матриці зв'язності дорівнює 1, то це означає, що *j*-й вхідний вектор належить до класу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

Вагова матриця другого шару LW^{21} (net.LW {2,1}) відповідає матриці зв'язності T, побудованої для даної навчальної послідовності. Ця операція може бути виконана за допомогою M-функції ind2vec, яка перетворює вектор цілей в матрицю зв'язності T. Твір $T*a^1$ визначає елементи вектора 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 n^2 і 0 в інших випадках. Таким чином, мережа PNN виконує класифікацію векторів входу по K

класах.

Для створення радіальних мереж загального вигляду призначені наступні функції

newrb	Створення радіальної базисної мережі
newrbe	Створення радіальної базисної мережі з нульовою помилкою
newgrnn	Створення узагальненої регресійної мережі
newpnn	Створення ймовірнісної мережі



newrb

Design radial basis network

Syntax

```
net = newrb(P,T,goal,spread,MN,DF)
```

Description

Radial basis networks can be used to approximate functions. `newrb` adds neurons to the hidden layer of a radial basis network until it meets the specified mean squared error goal.

`net = newrb(P,T,goal,spread,MN,DF)` takes two of these arguments,

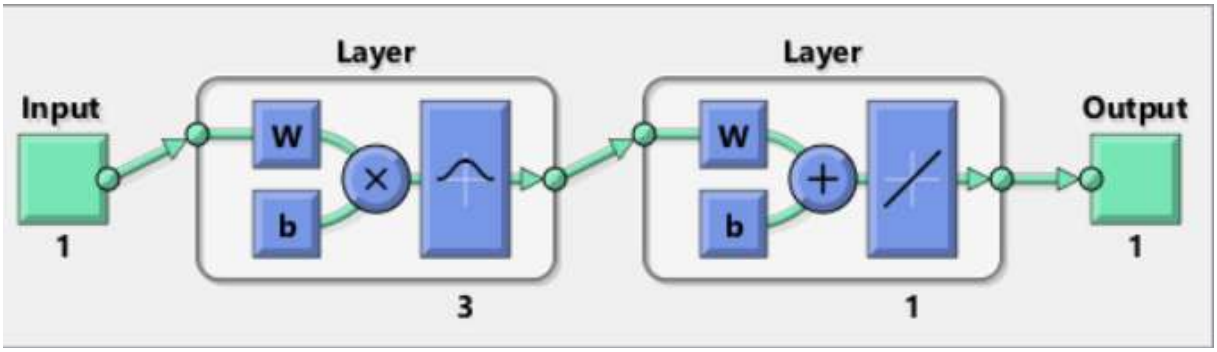
`P`	R-by-Q matrix of Q input vectors
`T`	S-by-Q matrix of Q target class vectors
`goal`	Mean squared error goal (default = 0.0)
`spread`	Spread of radial basis functions (default = 1.0)
`MN`	Maximum number of neurons (default is Q)
`DF`	Number of neurons to add between displays (default = 25)

and returns a new radial basis network.

Чим більший розкид (spread), тим плавніша апроксимація функції. Занадто великий розкид означає, що для підгонки швидкозмінної функції потрібно багато нейронів. Занадто малий розкид означає, що для підгонки гладкої функції потрібно багато нейронів, і мережа може погано узагальнювати. Викличте newrb з різними розкидами, щоб знайти найкраще значення для заданої задачі.

Приклад 1. Створення мережі RBF

```
>> P = [1 2 3];
>> T = [2.0 4.1 5.9];
>> net = newrb(P,T);
```



Функція newrb створює двошарову мережу. Перший шар має нейрони radbas та обчислює вихід дискримінантної функції за допомогою dist, а вихід активаційної функції за допомогою netprod. Другий шар має нейрони purelin та обчислює вихід дискримінантної функції за допомогою dotprod, а вихід активаційної функції за допомогою netsum. Обидва шари мають зсуви.

Спочатку шар radbas не має нейронів. Перелічені нижче кроки повторюються, доки середньоквадратична помилка мережі не стане нижчою за цільове значення goal.

1. Вхідні набори даних подаються на вхід мережі.
2. Знаходять вхідний вектор з найбільшою похибкою.
3. Додається нейрон radbas з вагами, що дорівнюють цьому вектору.
4. Ваги шару purelin оптимізуються для мінімізації похибки.

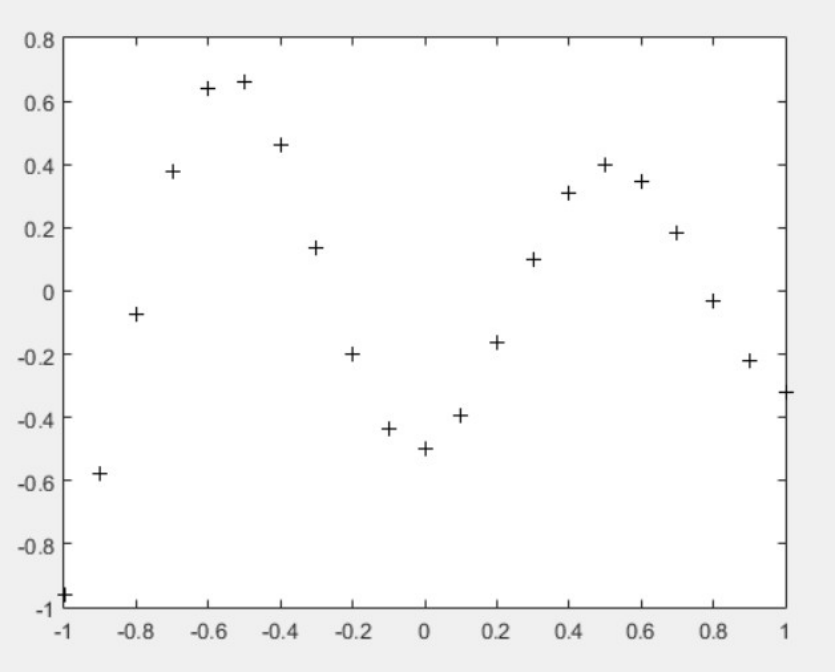
Приклад 2. Інтерполяція функції

`P = -1: .1: 1;` – вектор еталонних виходів

```
T = [-.9602 -.5770 -.0729 .3771 .6405 .6600 .4609 .1336 ...
     -.2013 -.4344 -.5000 -.3930 -.1647 .0988 .3072 .3960 ...
      .3449 .1816 -.0312 -.2189 -.3201];
```

– вектор еталонних входів

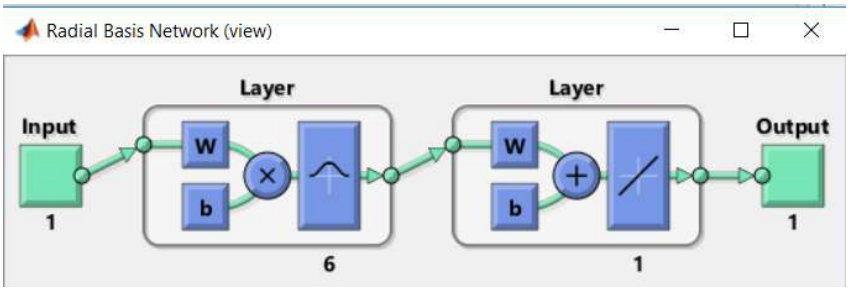
`>> plot (P, T, '+ k')` – точки навчальної множини



GOAL = 0.01; – Припустиме значення функціоналу помилки

SPREAD = 1; – Параметр впливу

`net = newrb (P, T, GOAL, SPREAD);` – Створення мережі



net.layers {1} .size – Число нейронів в прихованому шарі.

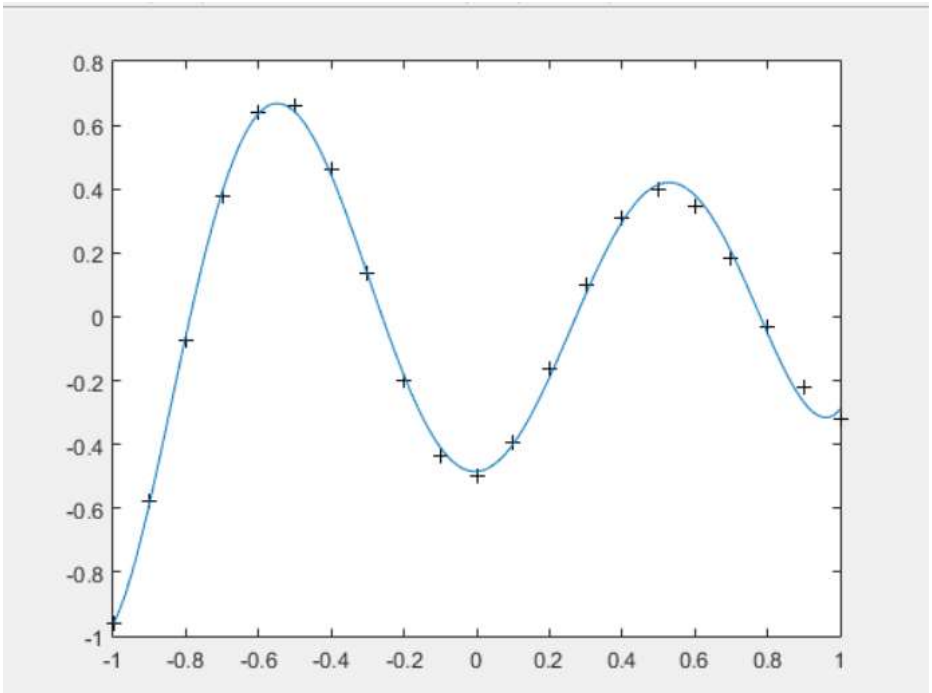
ans = 6

Для заданих параметрів нейронна мережа складається з шести нейронів.

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

```
>> view(net)
>> plot (P, T, '+ k') – Точки навчальної множини
hold on;
X = -1: .01: 1;
```

```
Y = sim (net, X); – Моделювання мережі
plot (X, Y);
```



З аналізу рисунка випливає, що при невеликій кількості нейронів прихованого шару радіальна базисна мережа досить добре апроксимує нелінійну залежність, задану навчальною множиною.

Функція immse дозволяє розрахувати показник якості аппроксимації, а саме mean squared error (MSE).

immse

Mean-squared error

Syntax

```
err = immse(X,Y)
```

Description

`err = immse(X,Y)` calculates the mean-squared error (MSE) between the arrays `X` and `Y`. `X` and `Y` can be arrays of any dimension, but must be of the same size and class.

```
>> MSE=immse(T,Y)

MSE =
   4.6188e-04
```

Приклад 2. В умовах Приклада 1 замінюємо SPREAD = 0.01

```
>> SPREAD = 0.01; – Параметр впливу
```

>> net = newrb (P, T, GOAL, SPREAD); – Створення мережі

>> plot (P, T, '+ k') – Точки навчальної множини

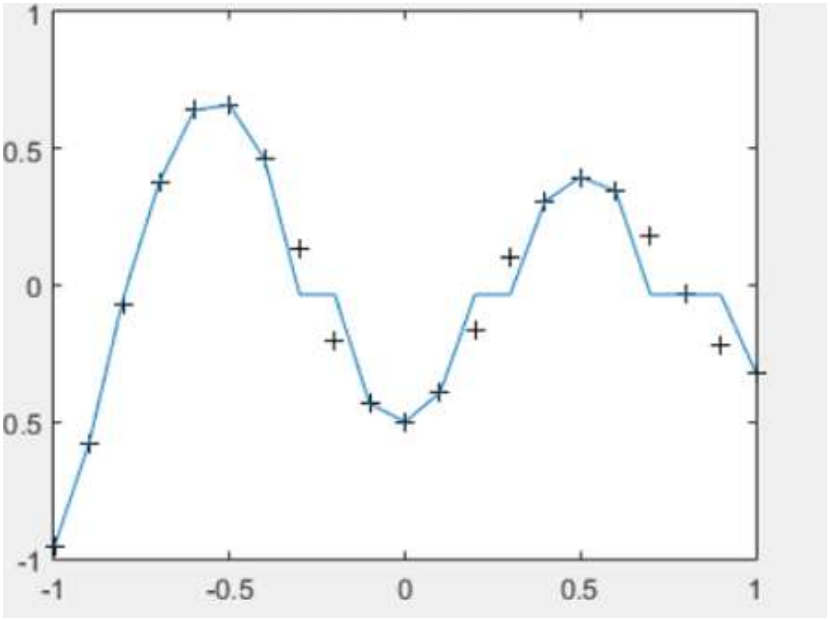
hold on;

X = -1: .01: 1;

Y = sim (net, X); – Моделювання мережі

plot (X, Y);

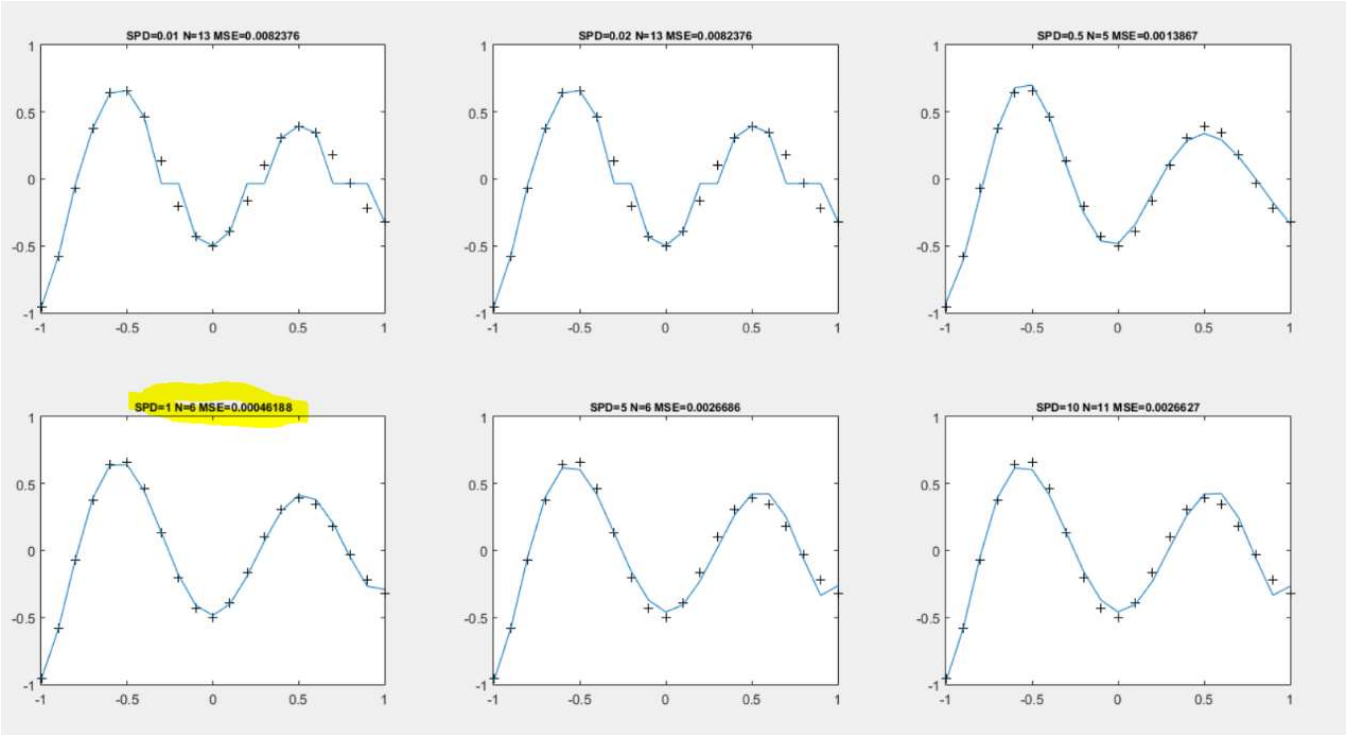
Зверніть увагу, що параметр впливу SPREAD тут дорівнює 0.01. Це означає, що діапазон перекриття вхідних значень становить лише ±0.01, а оскільки навчальні входи задані з інтервалом 0.1, то вхідні значення функціями активації не перекриваються. Це призводить до того, що, по-перше, збільшується кількість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



Показник якості апроксимації значно гірший, ніж у попередньому випадку:

```
>> MSE=immse(T,Y)
MSE =
    0.0082
```

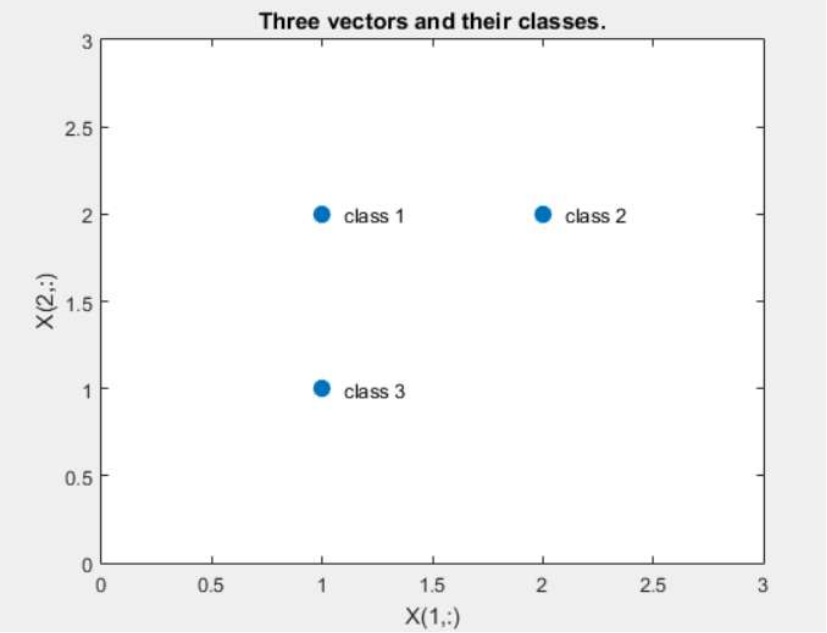
В результаті проведеної низки експериментів бачимо, що найкращі результати отримані для показника SPREAD=1:



Приклад 3. Дослідження мереж PNN

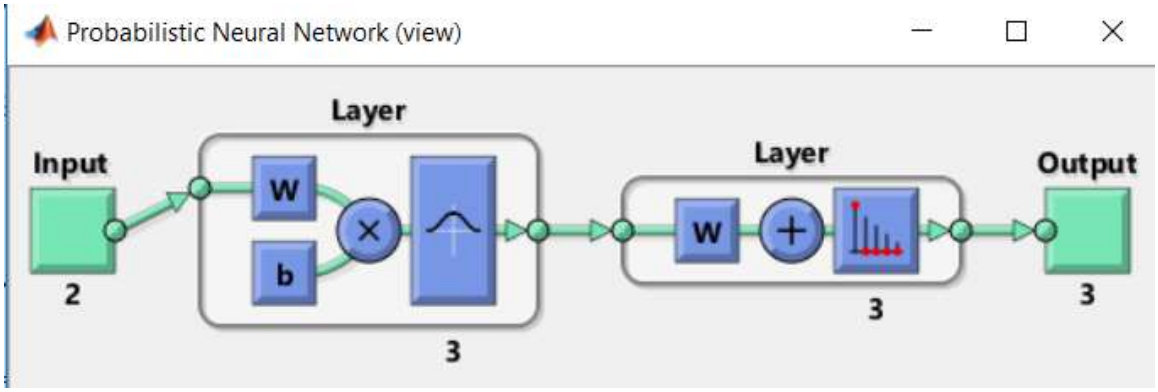
Дано три двоелементних вхідних вектора X і вектор пов'язаних з ними класів Tc. Необхідно створити ймовірнісну нейронну мережу, яка класифікує ці вектори належним чином

```
>> 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,:)')
hold on;
```



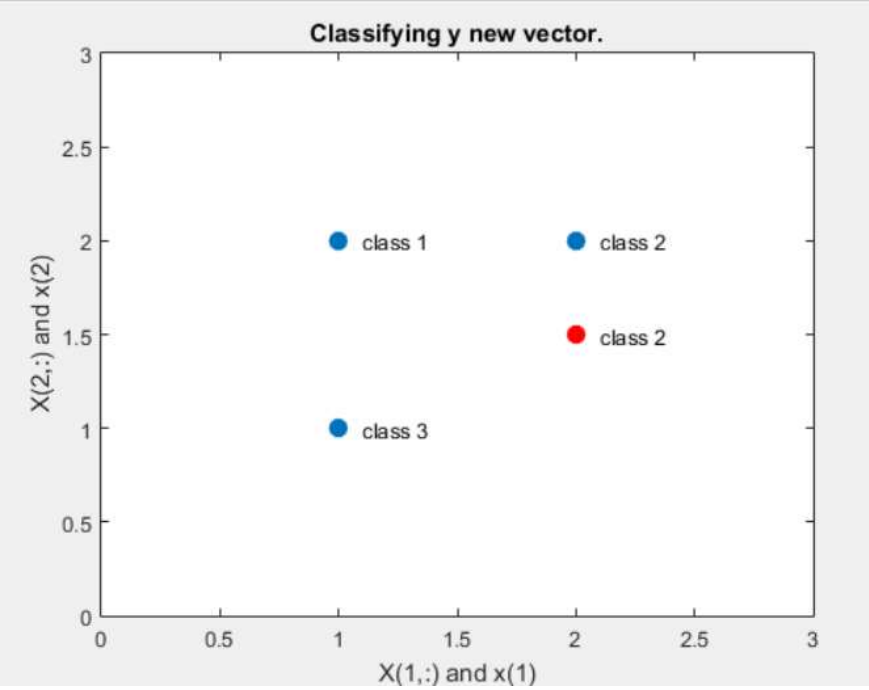
Перетворюємо цільові індекси Tc у вектори:

```
>> T = ind2vec(Tc);
>> Tc
Tc =
     1     2     3
>> T
T =
   (1,1)         1
   (2,2)         1
   (3,3)         1
>> spread = 1;
net = newpnn(X,T,spread);
>> view(net);
```



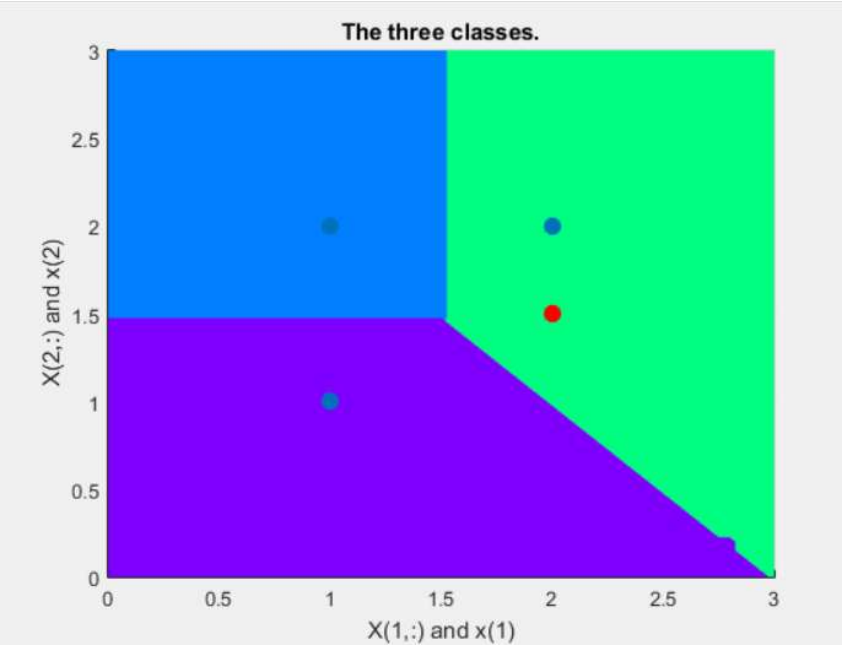
У результаті моделювання мережі формується матриця зв'язності, відповідна масиву векторів входу. Для того щоб перетворити її в індексний вектор, призначена М-функція vec2ind.

```
Yc = vec2ind(Y);
>> Y
Y =
     1     0     0
     0     1     0
     0     0     1
>> Yc
Yc =
     1     2     3
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)')
```



Альтернативна візуалізація:

```
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)')
```



2 МЕРЕЖІ КОХОНЕНА

2.1 Теоретичні відомості

Мережа Кохонена принципово відрізняється від розглянутих раніше мереж, оскільки використовує неконтрольоване навчання і навчальна множина складається лише із значень вхідних змінних.

Мережа навчається методом послідовних наближень. У процесі навчання на входи подаються дані, але мережа при цьому підлаштовується не під еталонне значення виходу, а під закономірності у вхідних даних(здійснює самонавчання).

В результаті навчання вектори вагових коефіцієнтів нейронів стають прототипами класів.

Відмінна риса цієї мережі – відображати вхідну інформацію, зберігаючи відношення сусідніх вхідних елементів, тобто зберігаючи топологічну структуру.

Це широко використовується для перетворення багатовимірних вихідних даних в одно- або двовимірні карти ознак.

Мережа складається з одного шару нейронів, які використовують як дискримінантну – функцію відстані, а як активаційну – функцію winner-take-all (WTA). Число входів кожного нейрона дорівнює розмірності вектора входів. Кількість нейронів збігається з необхідною кількістю класів, на які потрібно розбити об'єкти.

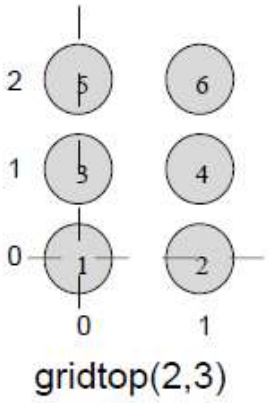
Конкурентна активаційна функція видає 1 для того вихідного елемента вектору a1, який відповідає нейрону-переможцю. Усі інші вихідні елементи в a1 дорівнюють 0.

Топологія Gridtop. Наприклад, припустімо, що вам потрібен масив 2 на 3 із шести нейронів. Ви можете отримати це за допомогою

```
>> pos = gridtop(2,3)
pos =
     0     1     0     1     0     1
     0     0     1     1     2     2
```

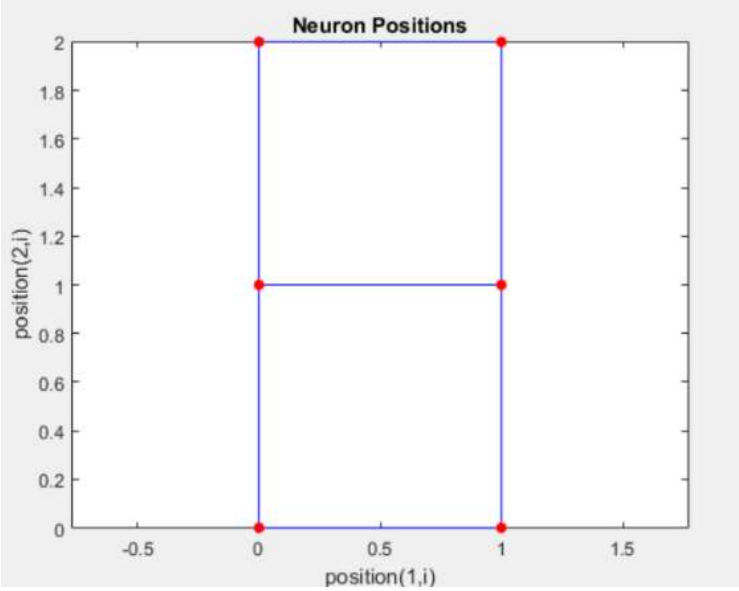
Тут нейрон 1 має позицію (0,0), нейрон 2 має позицію (1,0), нейрон 3 має

позицію (0,1) ……

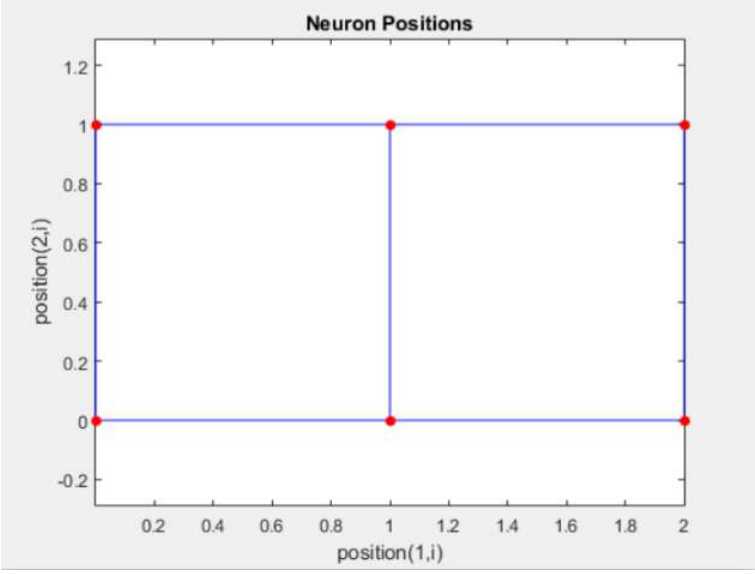


gridtop(2,3)

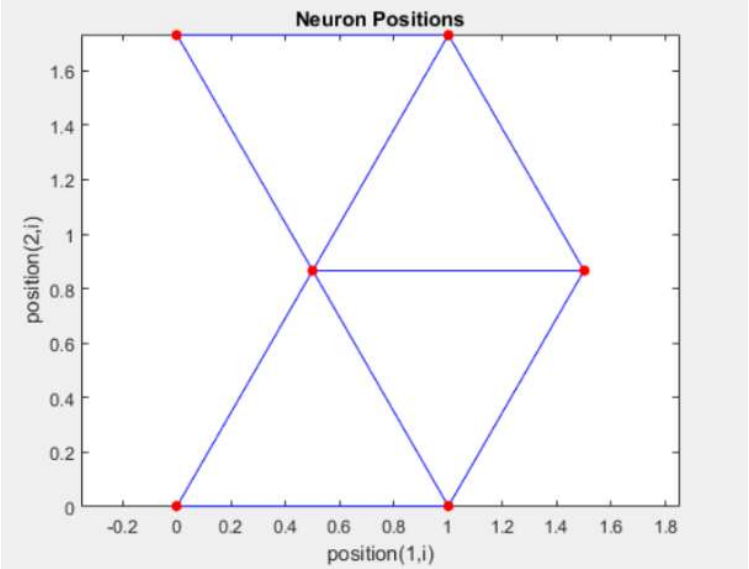
```
>> plotsom(pos)
```



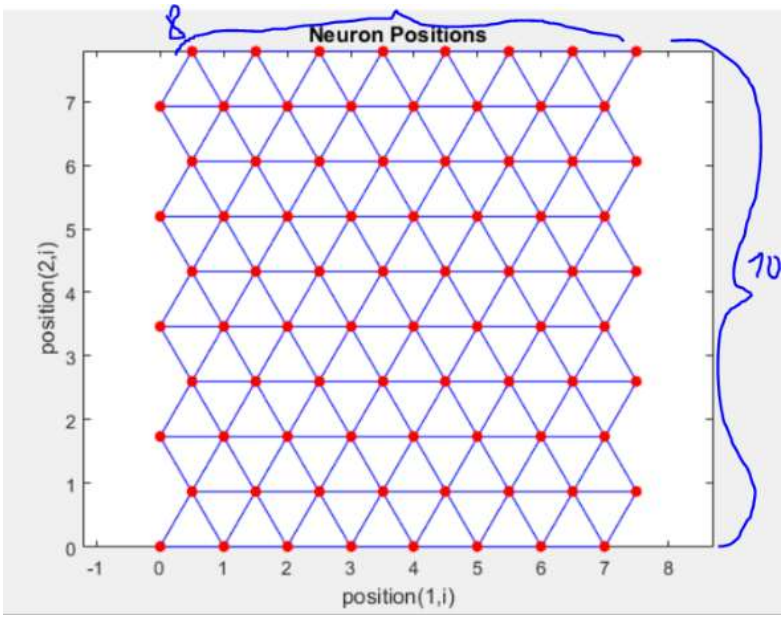

```
>> plotsom(gridtop(3,2))
```



```
>> 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)
```



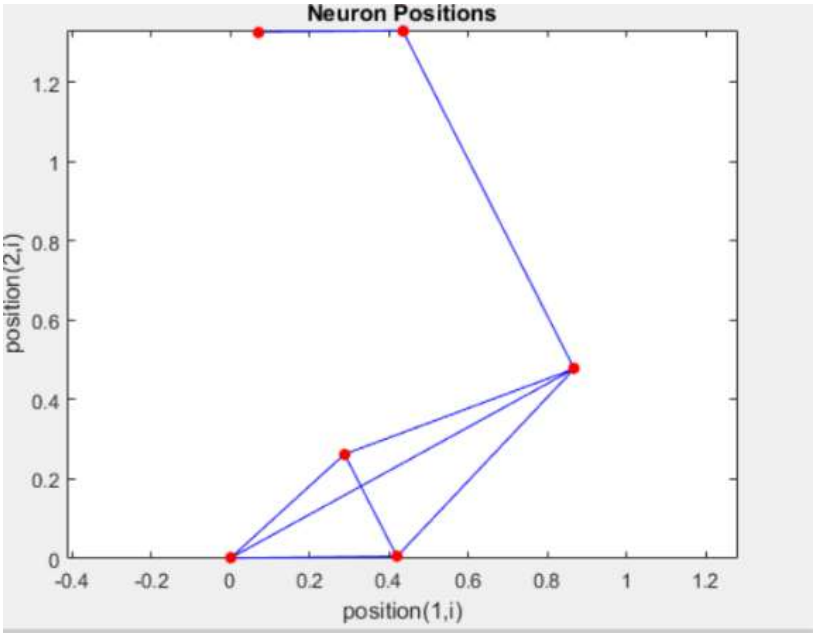

```
pos = hextop(8,10);
>> plotsom(pos)
```



Зверніть увагу, що hextop є шаблоном за замовчуванням для мереж SOFM, згенерованих за допомогою newsom!

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

```
pos =

         0    0.4199    0.2875    0.8676    0.0685    0.4346
         0    0.0036    0.2616    0.4772    1.3274    1.3301
>> plotsom(pos)
```



2.2 Distance Functions (dist, linkdist, mandist, boxdist)

Евклідова відстань(dist)

```
>> pos2 = [0 1 2; 0 1 2]
pos2 =
     0     1     2
     0     1     2
>> D2 = dist(pos2)
D2 =
         0    1.4142    2.8284
    1.4142         0    1.4142
    2.8284    1.4142         0
```

Функція відстані шару (boxdist)

```
>> d = boxdist(pos)
d =
     0     1     1     1     2     2
     1     0     1     1     2     2
```

```
     1     1     0     1     1     1
     1     1     1     0     1     1
     2     2     1     1     0     1
     2     2     1     1     1     0
```

*Функція відстані шару (*linkdist)

Відстань linkdist – це лише кількість зв'язків або кроків, які потрібно зробити, щоб дістатися до нейрона, що розглядається.

```
>> d=linkdist(pos)
d =
     0     1     1     2     2     3
     1     0     2     1     3     2
     1     2     0     1     1     2
     2     1     1     0     2     1
     2     3     1     2     0     1
     3     2     2     1     1     0
>> W1 = [1 2; 3 4; 5 6]
W1 =
     1     2
     3     4
     5     6
>> P1 = [1;1]
P1 =
     1
     1
>> Z1 = mandist(W1,P1)
Z1 =
       1
       5
       9
```

Загальний синтаксис команд:

```
net = newsom(P,[d1,d2,...],tfcn,dfcn,steps,in)
```

Конкурентні шари використовуються для вирішення задач класифікації.

NET = newsom(P,[D1,D2,...],TFCN,DFCN,OLR,OSTEPS,TLR,TNS) takes,

P -– Матриця RxQ з Q репрезентативних вхідних векторів.

D_i – Розмір i-го шару, значення за замовчуванням = [5 8].

TFCN – Функція топології, за замовчуванням = 'hextop'.

DFCN – Функція відстані, за замовчуванням = 'linkdist'.

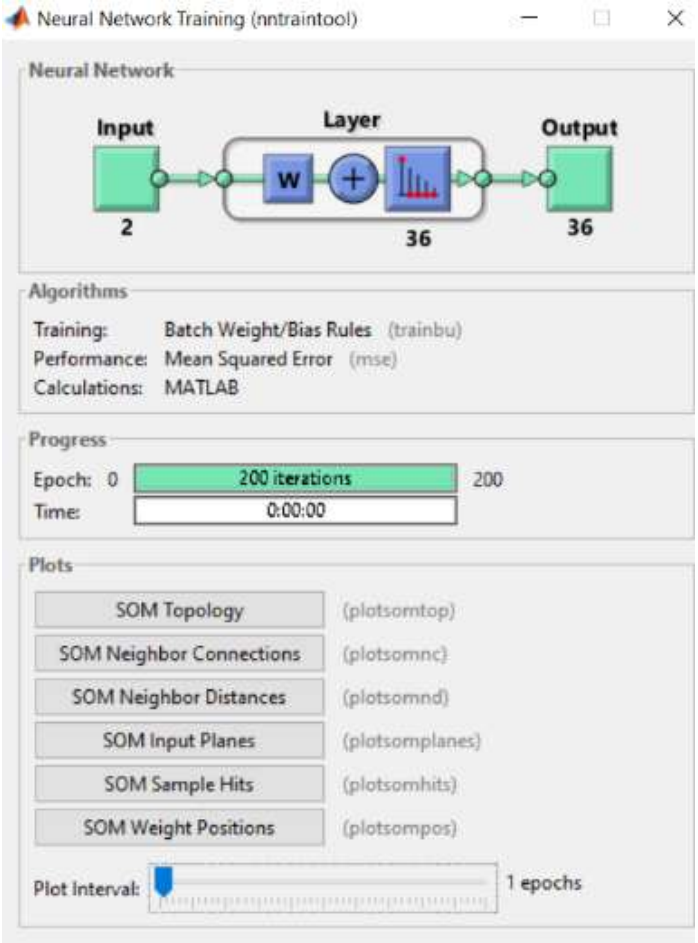
STEPS – Кроки для зменшення сусідства до 1, за замовчуванням = 100.

IN - Початковий розмір мікрорайону, за замовчуванням = 3.

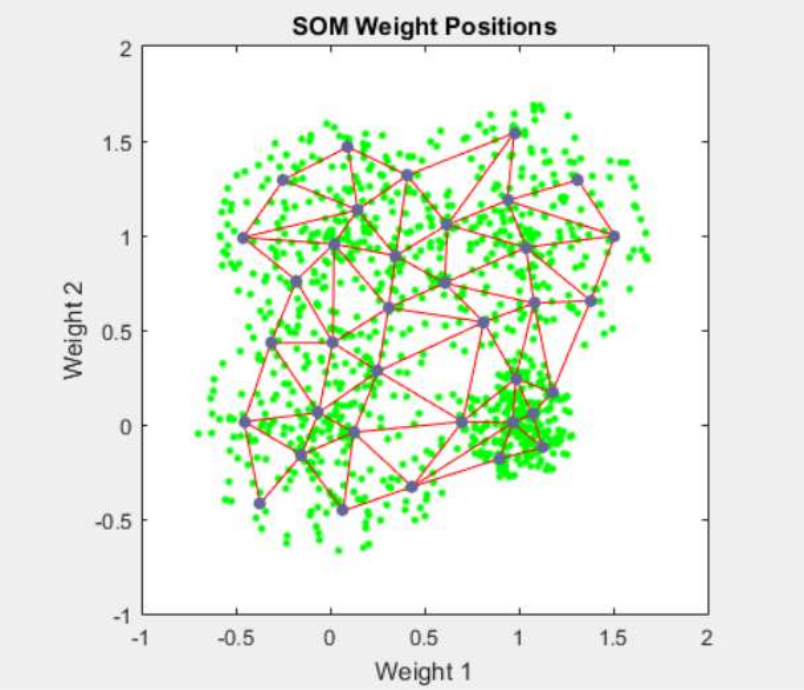
і повертає нову самоорганізуючусь карту.

Функція топології TFCN може мати тип HEXTOP, GRIDTOP або RANDTOP. Функція відстані може мати тип LINKDIST, DIST або MANDIST.

```
>> load simplecluster_dataset
net = newsom(simpleclusterInputs,[6 6]);
[net2,tr] = train(net,simpleclusterInputs);
```



Вихідні дані, нейронні центри та зв’язки між ними:



Коли вхідний простір має велику розмірність, ви не можете візуалізувати всі ваги одночасно. У цьому випадку натисніть SOM Neighbor Distances. На цьому малюнку використовується наступне кодування кольорів:

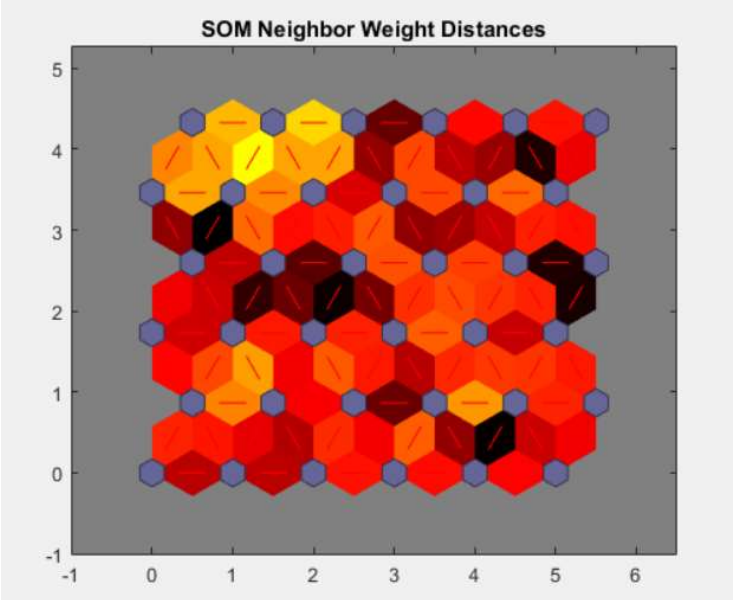
1. Сині шестикутники представляють нейрони.

2. Червоні лінії з’єднують сусідні нейрони.

3. Кольори в областях, що містять червоні лінії, вказують на відстані між нейронами.

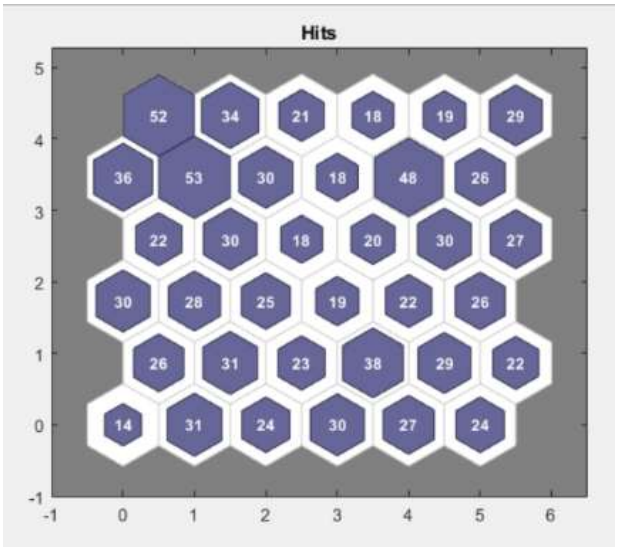
4. Темніші кольори позначають більші відстані.

5. Світліші кольори позначають менші відстані.



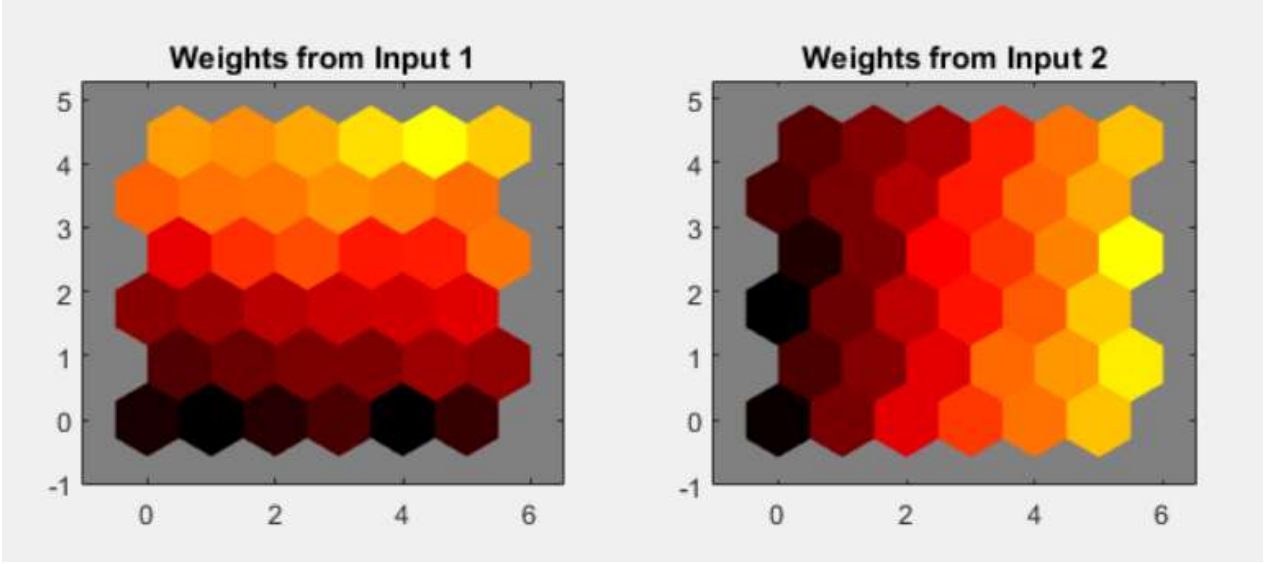
Група світлих сегментів з’являється у верхній лівій області, обмежена кількома темнішими сегментами. Це групування вказує на те, що мережа згрупувала дані у дві групи. Ці дві групи можна побачити на попередньому рисунку.

Інший малюнок може сказати вам, скільки точок даних пов’язано з кожним нейроном, натиснувши SOM Sample Hits.



Найкраще, якщо дані будуть рівномірно розподілені між нейронами. У цьому прикладі дані зосереджені трохи більше у верхніх лівих нейронах, але загалом розподіл досить рівномірний.

Ви також можете візуалізувати вагові коефіцієнти, натиснувши SOM Weight Planes.



Цей рисунок є візуалізацією вагових коефіцієнтів, які з'єднують кожен вхід із кожним із нейронів. (Темніші кольори представляють більшу вагу.) Якщо моделі з'єднання двох входів дуже схожі, можна припустити, що входи були сильно корельовані. У цьому випадку вхід 1 має з'єднання, які дуже відрізняються від з'єднань входу 2.

Приклад 1. Згенерувати навчальний датасет, який містить однорідні групи (кластери) даних.

`bounds = [0 1; 0 1];` – Центри кластерів перебувають у цих межах.

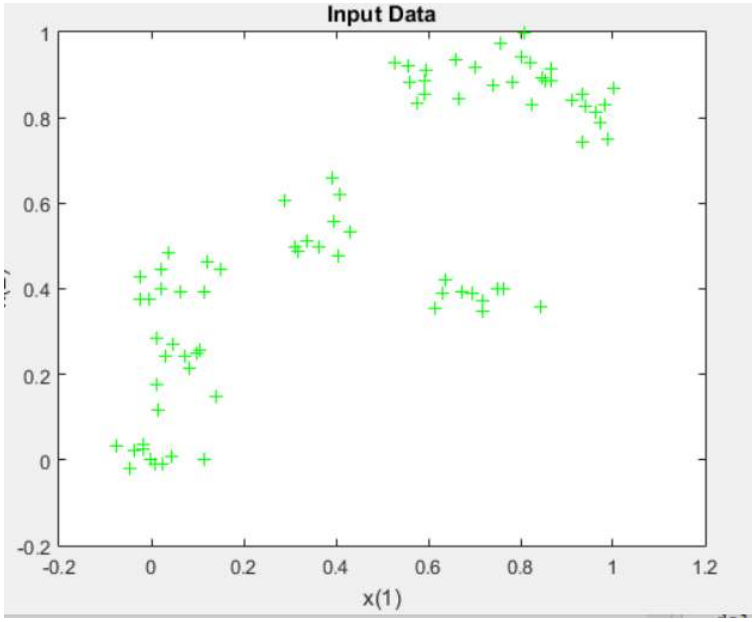
`clusters = 8;` – Це кількість кластерів.

`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,:),'+g');
title('Input Data');
xlabel('x(1)');
ylabel('x(2)');
```



Для побудови мережі використаємо функцію competlayer, що приймає 1 обов’язквий аргумент – кількість нейронів.

competlayer

Competitive layer

Syntax

```
competlayer(numClasses,kohonenLR,conscienceLR)
```

Description

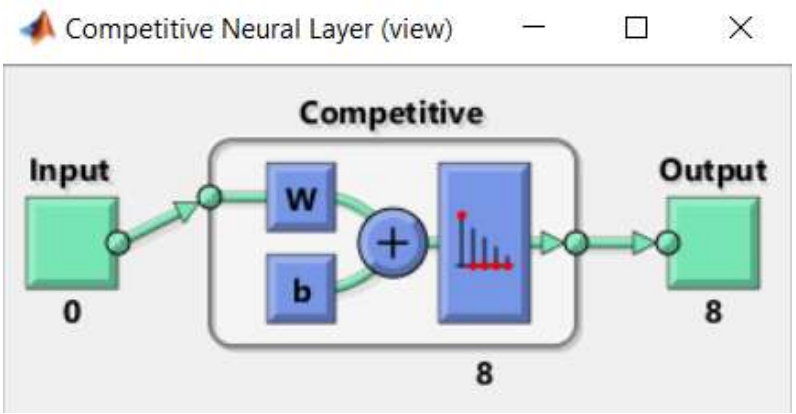
Competitive layers learn to classify input vectors into a given number of classes, according to similarity between vectors, with a preference for equal numbers of vectors per class.

`competlayer(numClasses,kohonenLR,conscienceLR)` takes these arguments,

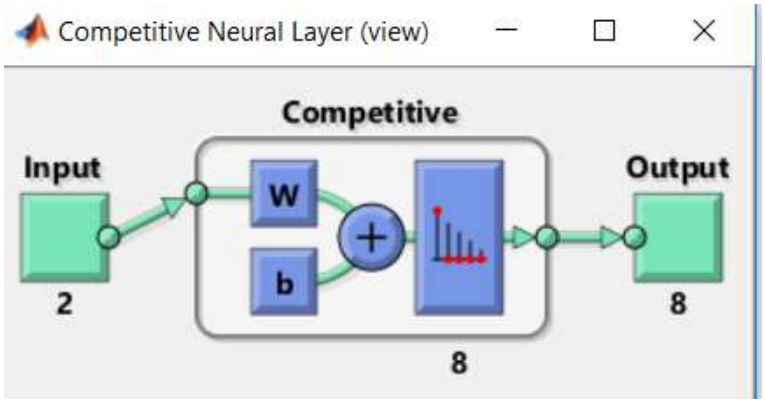
`numClasses`	Number of classes to classify inputs (default = 5)
`kohonenLR`	Learning rate for Kohonen weights (default = 0.01)
`conscienceLR`	Learning rate for conscience bias (default = 0.001)

and returns a competitive layer with `numClasses` neurons.

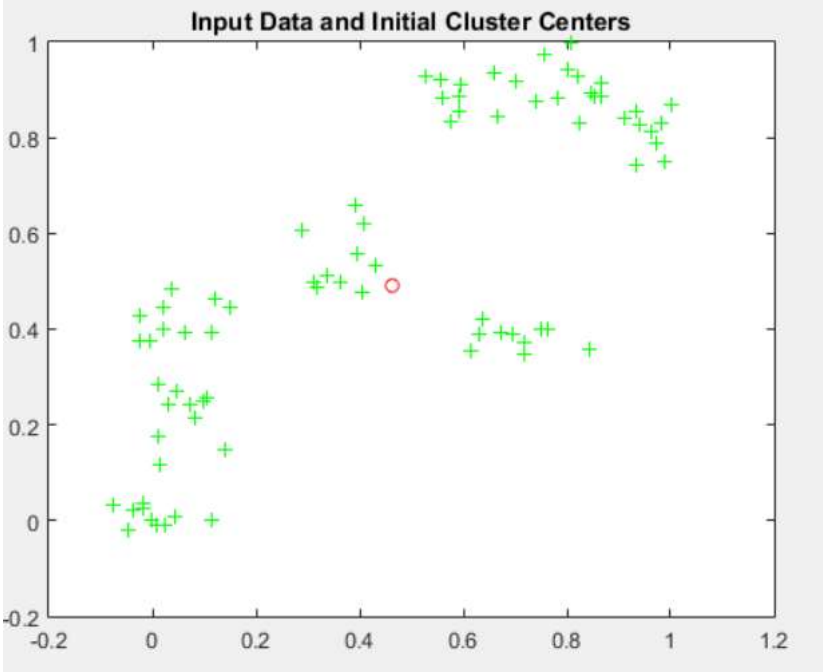
```
>> net = competlayer(8,.1);
>> view(net);
```




```
>> net = configure(net,x);
>> view(net);
```



```
>> w = net.IW{1}
w =
    0.4638    0.4904
    0.4638    0.4904
    0.4638    0.4904
    0.4638    0.4904
    0.4638    0.4904
    0.4638    0.4904
    0.4638    0.4904
    0.4638    0.4904

>> plot(x(1,:),x(2,:),'+g');
>> title('Input Data and Initial Cluster Centers');
>> hold on;
>> circles = plot(w(:,1),w(:,2),'or');
```



Зараз усі нейронні центри мають однакові координати (зливаються в одну точку)

Обмежимо кількість епох та здійснемо тренування:

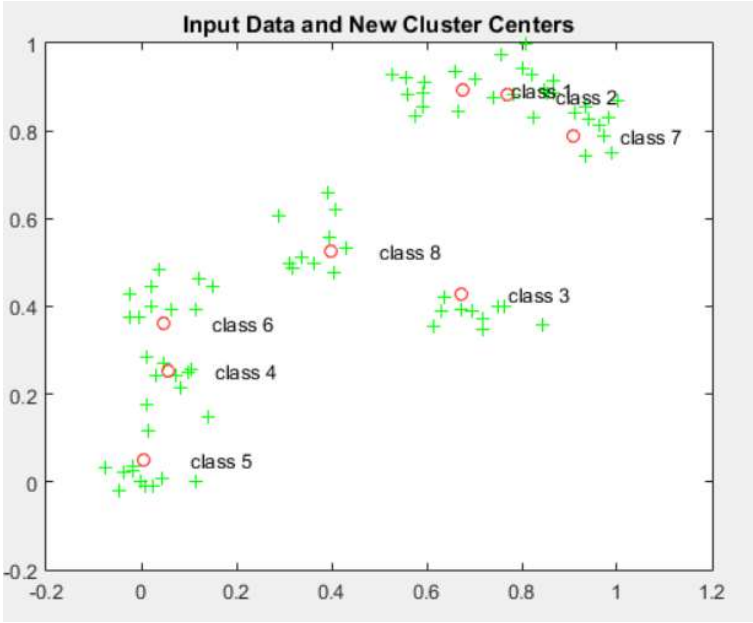
```
>> net.trainParam.epochs = 10;
>> net = train(net,x);
>> w = net.IW{1}

w =
    0.6767    0.8948
    0.7706    0.8825
    0.6712    0.4287
    0.0554    0.2539
    0.0042    0.0516
    0.0479    0.3627
    0.9066    0.7890
    0.3989    0.5260
```

`>> delete(circles);` – прибрали попередні кластерні центри

`>> title('Input Data and New Cluster Centers');`

`>> for i = 1:8, text(w(i,1)+0.1,w(i,2),sprintf('class %g',i)), end` – підписи для кластерів

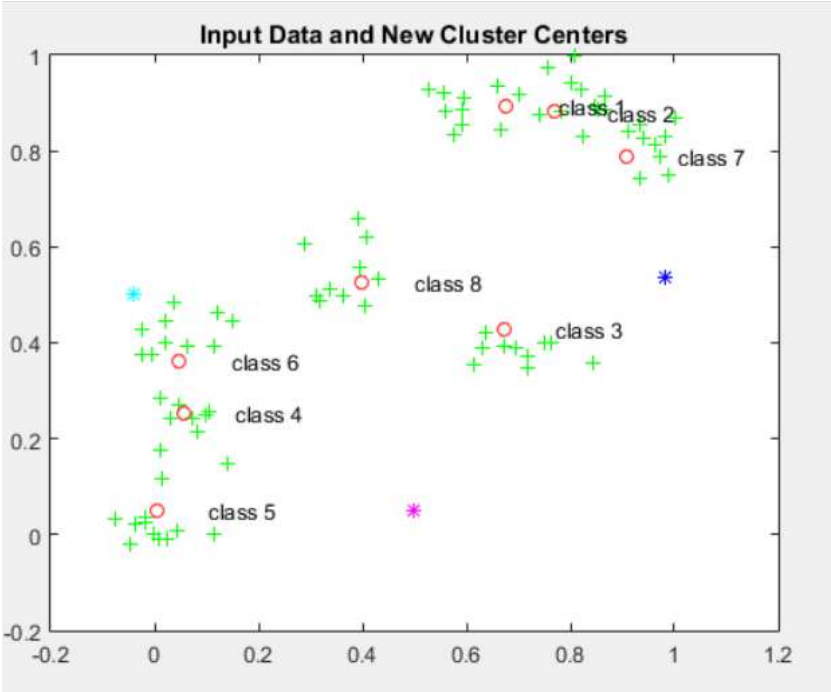
`>> circles = plot(w(:,1),w(:,2),'or');` – нові кластерні центри



Задати довільним чином три точки та здійснити їх класифікацію:

```
>> x1=[0.9807    0.5363]
```

```
>> x2=[ 0.5000    0.1000]
>> x3=[ 0.2000    1.0200]
>> circle1 =  plot(x1(:,1),x1(:,2),'*b');
>> circle2 =  plot(x2(:,1),x2(:,2),'*m');
>> circle3 =  plot(x3(:,1),x3(:,2),'*c');
```




```
y1=net(x1')
y1 =
     0
     0
     0
     0
     0
     0
     1
     0
```

%X1 класифіковано до 7-го кластеру

```
>> y2=net(x2')
y2 =
       0
```

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

%X2 класифіковано до 3-го кластеру

```
>> y3=net(x3')
y3 =
     0
     0
     0
     0
     0
     1
     0
     0
```

%X3 класифіковано до 6-го кластеру

Побудуємо карту Кохонена

selforgmap

Self-organizing map

Syntax

```
selforgmap(dimensions,coverSteps,initNeighbor,topologyFcn,distanceFcn)
```

Description

Self-organizing maps learn to cluster data based on similarity, topology, with a preference (but no guarantee) of assigning the same number of instances to each class.

Self-organizing maps are used both to cluster data and to reduce the dimensionality of data. They are inspired by the sensory and motor mappings in the mammal brain, which also appear to automatically organizing information topologically.

`selforgmap(dimensions,coverSteps,initNeighbor,topologyFcn,distanceFcn)` takes these arguments,

`dimensions`	Row vector of dimension sizes (default = `[8 8]`)
`coverSteps`	Number of training steps for initial covering of the input space (default = 100)
`initNeighbor`	Initial neighborhood size (default = 3)
`topologyFcn`	Layer topology function (default = `'hextop'`)
`distanceFcn`	Neuron distance function (default = `'linkdist'`)

Приклад 2. Згенерувати навчальний датасет, який містить однорідні групи

(кластери) даних.

Наступний код дозволяє згенерувати датасет з напередвизначеною кількістю кластерів (але не видає маркування даних по кластерам).

`bounds = [0 1; 0 1];` – Центри кластерів перебувають у цих межах.

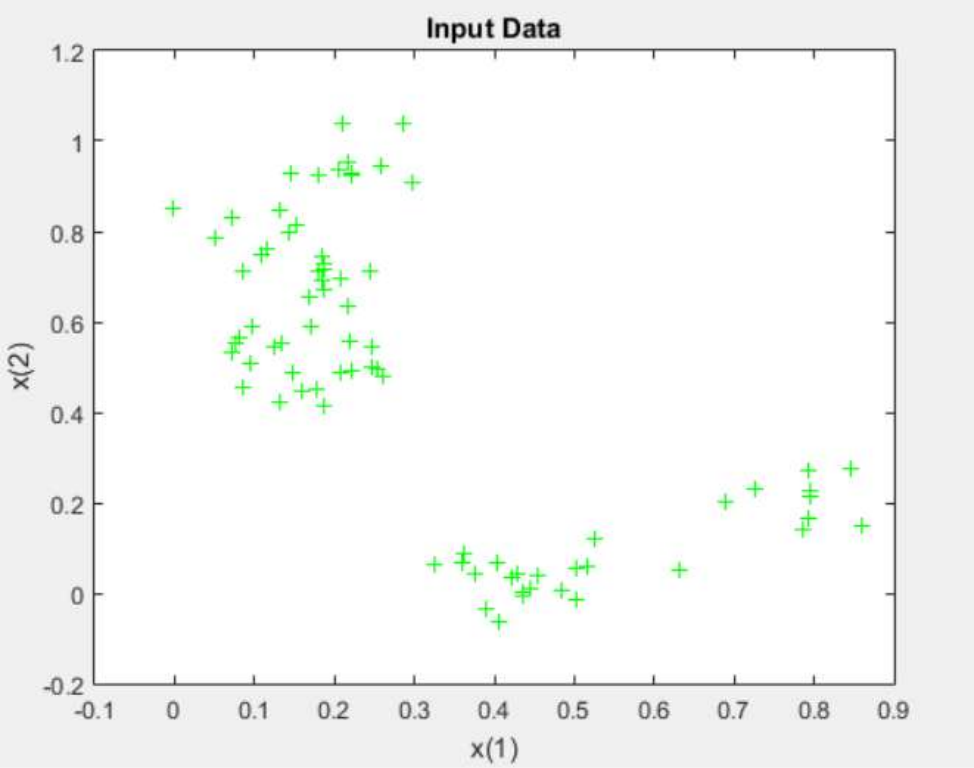
`clusters = 8;` – Це кількість кластерів.

`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,:),'+g');
title('Input Data');
xlabel('x(1)');
ylabel('x(2)');
```



Далі побудуємо мережу Кохонена для того, щоб провести класифікацію цих даних на 8 груп.

Для побудови мережі використаємо функцію competlayer, що приймає 1

обов’язквий аргумент – кількість нейронів.

competlayer

Competitive layer

Syntax

```
competlayer(numClasses,kohonenLR,conscienceLR)
```

Description

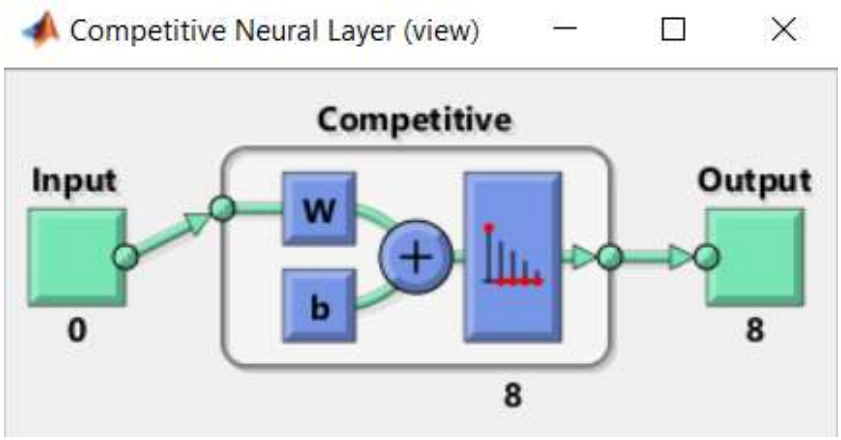
Competitive layers learn to classify input vectors into a given number of classes, according to similarity between vectors, with a preference for equal numbers of vectors per class.

`competlayer(numClasses,kohonenLR,conscienceLR)` takes these arguments,

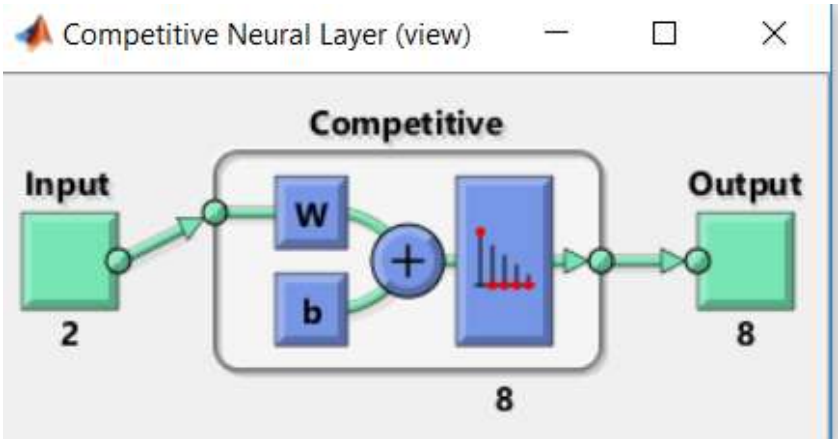
`numClasses`	Number of classes to classify inputs (default = 5)
`kohonenLR`	Learning rate for Kohonen weights (default = 0.01)
`conscienceLR`	Learning rate for conscience bias (default = 0.001)

and returns a competitive layer with `numClasses` neurons.

```
>> net = competlayer(8,.1);
>> view(net);
```




```
>> net = configure(net,x);
>> view(net);
```



```
>> w = net.IW{1}
w =
    0.4638    0.4904
    0.4638    0.4904
    0.4638    0.4904
    0.4638    0.4904
    0.4638    0.4904
    0.4638    0.4904
    0.4638    0.4904
    0.4638    0.4904
```

На поточному кроці (до застосування процедури тренування) вагові коефіцієнти усіх нейронів є однаковими.

Обмежимо кількість епох та здійснімо тренування:

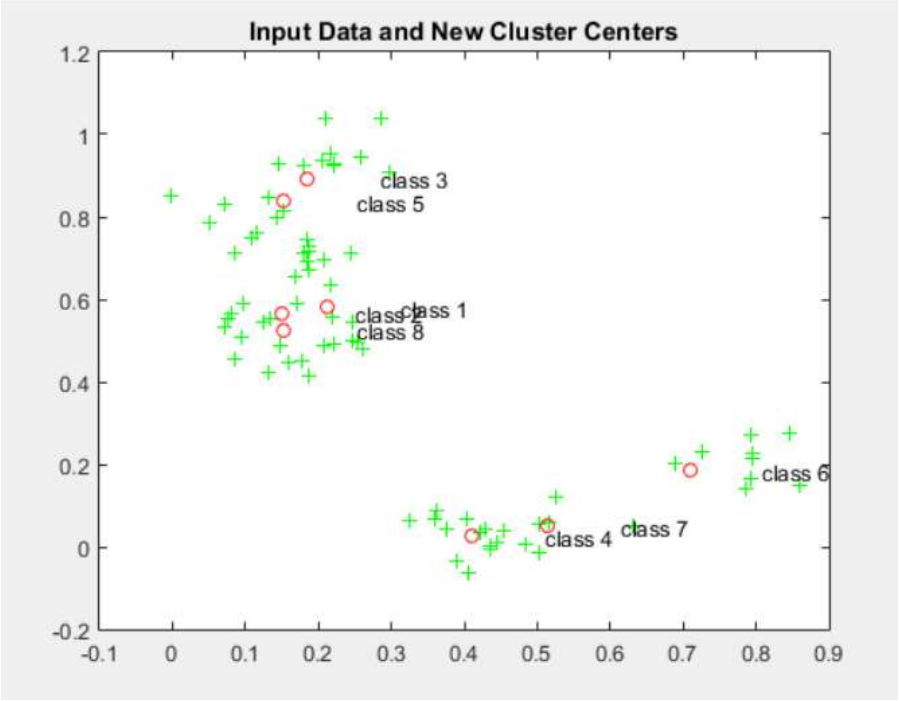
```
>> net.trainParam.epochs = 10;
>> net = train(net,x);
>> w = net.IW{1}
w =
    0.2120    0.5806
    0.1501    0.5671
    0.1837    0.8932
    0.4105    0.0257
    0.1532    0.8379
    0.7088    0.1849
    0.5134    0.0503
    0.1534    0.5256
```

```
>> plot(x(1,:),x(2,:),'+g');
>> hold on;
>> title('Input Data and New Cluster Centers');
>> for i = 1:8, text(w(i,1)+0.1,w(i,2),sprintf('class %g',i)), end
```

– підписи для кластерів

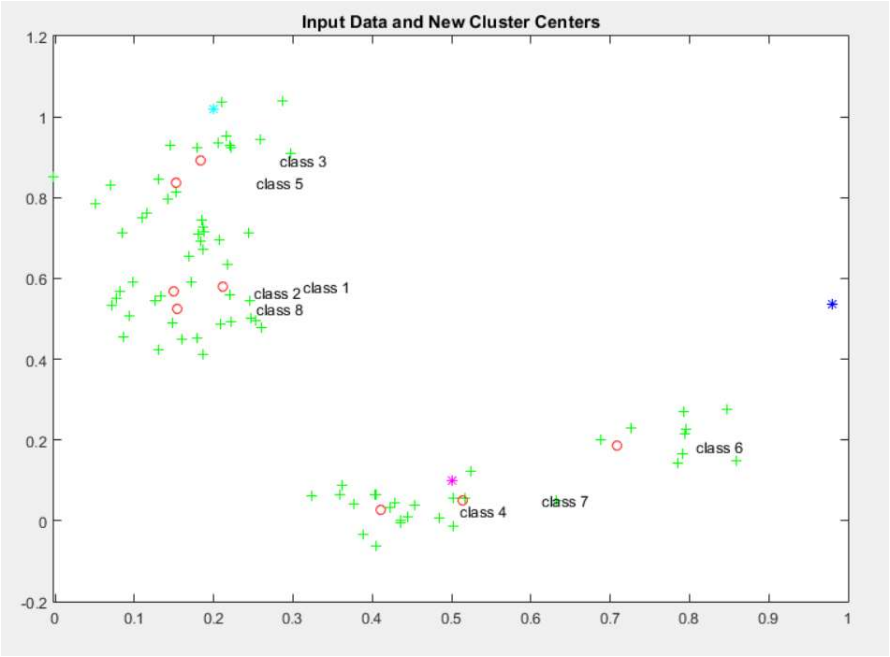
```
>> circles = plot(w(:,1),w(:,2),'or');
```

– нові кластерні центри



Задати довільним чином три точки:

```
>> x1=[0.9807    0.5363]
>> x2=[ 0.5000    0.1000]
>> x3=[ 0.2000    1.0200]
>> hold on;
>> circle1 =  plot(x1(:,1),x1(:,2),'*b');
>> circle2 =  plot(x2(:,1),x2(:,2),'*m');
>> circle3 =  plot(x3(:,1),x3(:,2),'*c');
```



та здійснити їх класифікацію:

```
>> y1=net(x1')
y1 =
         0
         0
         0
         0
         0
         1
         0
         0
```

% Точку віднесено до 6-го кластеру

```
>> y2=net(x2')
y2 =
     0
     0
     0
     0
     0
     0
     1
     0
```

% Точку віднесено до 7-го кластеру

```
>> y3=net(x3')
y3 =
        0
        0
        1
        0
        0
        0
        0
        0
```

% Точку віднесено до 3-го кластеру

Розглянемо, як було розподілено по кластерах навчальні дані. Для цього здійснемо симуляцію

```
>> result=sim(net,x);
```

У якості відповіді отримуємо матрицю 8 на 80

Нижче наведено фрагмент її фрагмент. Інтерпретація: 1-й зразок даних було класифіковано до 4-го кластера, 2-й зразок – до 7-го кластера.

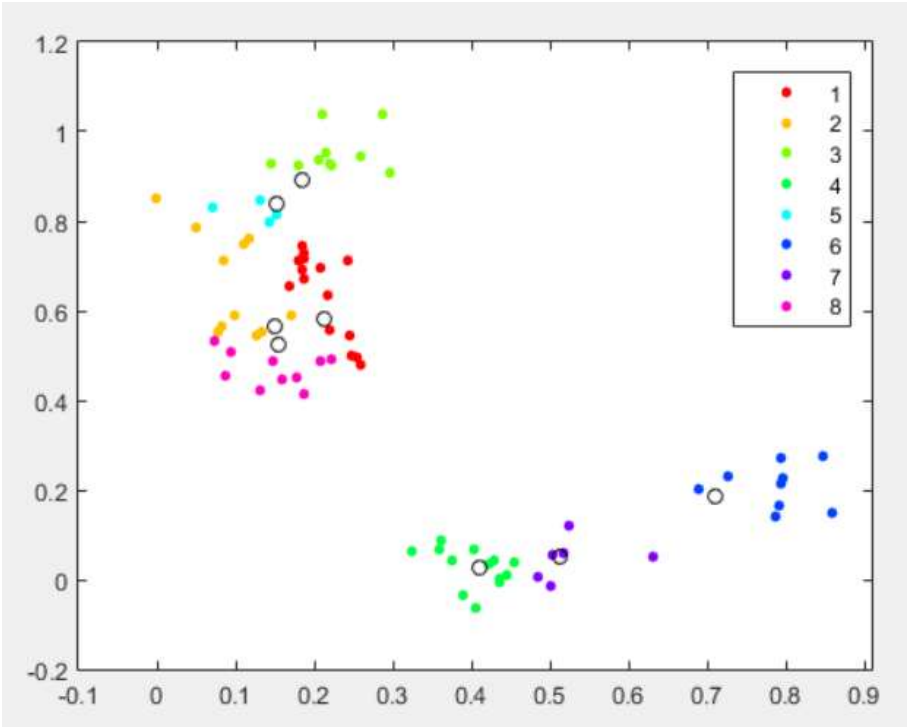
```
result =

  Columns 1 through 23

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

Можна перевести цей результат у більш зручний формат:

```
>> res=vec2ind(result);
```

```
res =

  Columns 1 through 23

     4     7     2     8     3     1     6     1     4     4

  Columns 24 through 46

     1     7     7     5     8     3     1     6     1     4

  Columns 47 through 69

     6     1     4     7     5     2     3     8     6     1
```

```
>> gscatter(x(1,:),x(2,:),res);
>> hold on;
>> circles = plot(w(:,1),w(:,2),'ok'); % 'ok' - means black (k) circles (o)
```



Бачимо, що обрана попередньо кількість кластерів не є оптимальною.

2.3 Індивідуальне завдання

1. Згенерувати навчальний датасет за допомогою функції nngenc з параметрами, що відповідають Вашому варіанту.

2. Натренувати мережу з відповідною кількістю нейронів у конкурентному шарі.

3. Самостійно згенерувати три зразки даних та здійснити процедуру розпізнавання за допомогою створеної мережі.

4. Зробити висновки щодо отриманих результатів.

Варіант	bounds	clusters	points	std_dev
1	[-1 1; 0 1]	6	15	0,05
2	[0 1; -1 1]	7	12	0,03
3	[1 2; 1 2]	8	10	0,05
4	[-1 1; 0 1]	6	15	0,03
5	[0 1; -1 1]	7	12	0,05
6	[1 2; 1 2]	8	10	0,03
7	[-1 1; 0 1]	6	15	0,05
8	[0 1; -1 1]	7	12	0,03
9	[1 2; 1 2]	8	10	0,05
0	[-1 0; 1 2]	6	15	0,03

2.4 Контрольні питання

1. У чому полягає відмінність мереж Кохонена між собою та у порівнянні з іншими нейромережами?

2. Яка математична основа радіально-базисних нейронних мереж?

3. Які існують моделі та принципи синтезу архітектури радіально-базисних нейронних мереж?

4. Як відбувається навчання радіально-базисних нейромереж?

5. Як відбувається функціонування радіально-базисних нейромереж?

6. Які переваги і недоліки можна визначити для радіально-базисних мереж?

7. У чому полягає навчання мережі Кохонена?

8. Які дистанційні функції використовуються при побудові мереж Кохонена?

Навчальне видання

Методичні вказівки
до виконання лабораторної роботи
«Побудова та навчання радіально-базисної (RBF) мережі у середовищі MATLAB. Неконтрольоване навчання мережі Кохонена»
з навчальної дисципліни «Вступ до нейронних мереж»
для студентів денної та заочної форм навчання
за спеціальністю F2 «Інженерія програмного забезпечення», F3 «Комп'ютерні науки» та F6 «Інформаційні системи та технології»

Укладачі:
ЧЕРНОВА Наталя Леонідівна
АРКАТОВ Денис Борисович

Відповідальний за випуск доц. Копп А.М.
Роботу до видання рекомендував проф. Гамаюн І. П.

В авторській редакції

План 2025 р., поз. 630

Підп. до друку Гарнітура Times New Roman.
Видавничий центр НТУ «ХПІ»,
вул. Кирпичова, 2, м. Харків, 61002
Свідоцтво про державну реєстрацію ДК № 5478 від 21.08.2017 р.
Електронна версія