Мережі кластеризації та класифікації даних

У процесі аналізу великих інформаційних масивів даних незмінно виникають завдання, пов'язані з дослідженням топологічної структури даних, їх об'єднанням у групи (кластери), розподілом за класами тощо. Це можуть бути економічні, фінансові, науково-технічні, медичні та інші додатки, де потрібне вирішення таких практичних завдань, як стиск даних, їх зберігання та пошук, визначення характеристик об'єкта з обмеженим набором ознак. Такі завдання можуть бути успішно вирішені із застосуванням спеціального класу нейронних мереж, що самоорганізовуються.

Самоорганізаційні нейронні мережі

Властивість самоорганізації одна із найпривабливіших властивостей нейронних мереж. Такою властивістю володіють нейронні мережі, що самоорганізовуються, описані фінським ученим Т. Кохоненом. Нейрони мережі, що самоорганізовуються, можуть бути навчені виявленню груп (кластерів) векторів входу, що володіють деякими загальними властивостями. При вивченні нейронних мереж, що самоорганізовуються, або мереж Кохонена, суттєво розрізняти мережі з невпорядкованими нейронами, які часто називають шарами Кохонена, та мережі з упорядкованими нейронами, які часто називають картами Кохонена. Останні відображають структуру даних таким чином, що близьким кластерам даних на карті відповідають близько розташовані нейрони.

Для створення саморганізаційних нейронних мереж, що є шаром або картою Кохонена, призначені М-функції newc і newsom відповідно.

За командою help selforg можна отримати наступну інформацію про М-функції, що входять до складу ППП Neural Network Toolbox і які стосуються побудови мереж Кохонена:

Self-organizing networks	*Самоорганізаційні мережі*
New networks	**Формування мережі**
newc newsom	**Створення прошароку Кохонена** **Створення карти Кохонена**
Using networks	**Робота з мережею**
sim init adapt train	Моделювання Ініціалізація Адаптація Навчання
Weight functions	**Функції відстані та зважування**
negdist	**Від'ємна евклідова відстань**
Net input functions	**Функції накопичення**
netsum	**Сума зважених входів**
Transfer функцій	**Функції активації**
compet	**Конкуруюча функція активації**
Topology functions	**Опції опису топології мережі**
gridtop hextop randtop	Прямокутна сітка Гексагональна сітка Сітка з випадково розподіленими вузлами
Distance functions	**Функції відстані**
dist boxdist mandist	Євклідова відстань Відстань максимального координатного зсуву Відстань сумарного координатного зсуву Відстань зв'язку

linkdist	
Initialization функцій	**Функції ініціалізації мережі**
initlay initwb initcon midpoint	Пошарова ініціалізація Ініціалізація ваг та зміщень Ініціалізація зсувів з урахуванням чутливості нейронів Ініціалізація ваг за правилом середньої точки
Learning functions	**Опції налаштування параметрів**
learnk learncon learnsom	Правило налаштування ваги для прошароку Кохонена Правило налаштування зсувів для прошароку Кохонена Правило налаштування ваги карти Кохонена
Adapt functions	**Функції адаптації**
adaptwb	Адаптація ваг та зміщень
Training functions	**Функції навчання**
trainwb1	Повекторне навчання ваг та зміщень
Demonstrations	**Демонстраційні приклади**
democ1 demosm1 demosm2	Налаштування прошароку Кохонена Одновимірна карта Кохонена Двовимірна карта Кохонена

Прошарок Кохонена

Розглянемо самоорганізаційну нейронну мережу з єдиним прошарком, завдання якої полягає в тому, щоб правильно згрупувати (кластеризувати) вектори входу, що надходять на неї.

Архітектура мережі

Архітектуру прошароку Кохонена показано на рис.1.

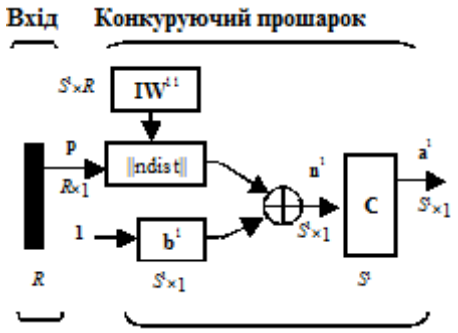


Рис. 1

Неважко переконатися, що це прошарок конкуруючого типу, оскільки в ньому застосовано конкуруючу функцію активації. Крім того, архітектура цього прошароку дуже нагадує архітектуру прихованого прошароку радіально базисної мережі. Тут використано блок ndist для обчислення від'ємної евклідової відстані між вектором входу p і рядками матриці ваг IW^{11}. Вхід функції активації n^1 – результат додавання обчисленої відстані з вектором зміщення b. Якщо всі зміщення нульові, максимальне значення n^1 не може перевищувати 0. Нульове значення n^1 можливе тільки коли вектор входу p виявляється рівним вектору ваги одного з нейронів. Якщо зсуви відмінні від 0, то можливі і додатні значення для елементів вектора n^1.

Конкуруюча функція активації аналізує значення елементів вектора n^1 і формує виходи нейронів рівні 0 для всіх нейронів, крім одного нейрона – переможця, який має на вході максимальне значення. Таким чином, вектор виходу прошароку a^1 має

єдиний елемент, що дорівнює 1, який відповідає нейрону-переможцю, а інші рівні 0. Така активаційна характеристика може бути описана наступним чином:

$$a_i^1 = \begin{cases} 1, & i = i^*, \ i^* = \arg(\max \ n_i^1) \\ 0, & i \neq i^* \end{cases}$$

Зауважимо, що ця активаційна характеристика встановлюється не на окремий нейрон, а на прошарок. Тому така активаційна характеристика і отримала назву конкуруючої. Номер активного нейрона i^* визначає ту групу (кластер), до якої найближчий вхідний вектор.

Створення мережі

Для формування прошароку Кохонена призначено М-функцію newc. Покажемо, як вона працює на простому прикладі. Припустимо, що заданий масив із чотирьох двоелементних векторів, які треба розділити на 2 класи:

```
p = [.1.8.1.9; .2.9.1.8]
p =
0.1000 0.8000 0.1000 0.9000
0.2000 0.9000 0.1000 0.8000
```

У цьому прикладі неважко побачити, що 2 вектори розташовані поблизу точки (0,0) і 2 вектори поблизу точки (1,1). Сформуємо прошарок Кохонена з двома нейронами для аналізу двоелементних векторів входу з діапазоном значень від 0 до 1:

```
net = newc([0 1; 0 1],2);
```

Перший аргумент вказує діапазон вхідних значень, другий визначає кількість нейронів у прошарку. Початкові значення елементів матриці ваг задаються як середнє значення максимального і мінімального значень, тобто в центрі інтервалу вхідних значень; це реалізується за замовчуванням за допомогою М-функції midpoint під час створення мережі. Переконаємося, що це справді так:

```
wts = net.IW {1,1}
wts =
0.5000 0.5000
0.5000 0.5000
```

Визначимо характеристики прошароку Кохонена:

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

З цього опису випливає, що мережа використовує функцію евклідової відстані dist, функцію ініціалізації initwb, функцію обробки входів netsum, функцію активації compet та функцію опису топології hextop.

Характеристики зсувів такі:

```
net.biases{1}
ans =
```

```
initFcn: 'initcon'
learn: 1
learnFcn: 'learncon'
learnParam: [1×1 struct]
size: 2
userdata: [1×1 struct]
```

Зміщення задаються функцією initcon і для ініціалізованої мережі рівні

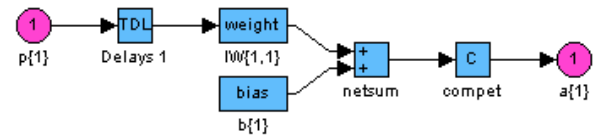
```
net.b{1}
ans =
5.4366
5.4366
```

Функцією налаштування зсувів є функція learncon, що забезпечує налаштування з урахуванням параметра активності нейронів.

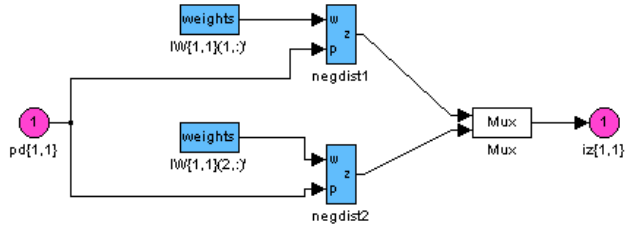
Елементи структурної схеми прошароку Кохонена показано на рис. 2 а-б і можуть бути отримані за допомогою оператора

```
gensim(net)
```

Вони наочно пояснюють архітектуру та функції, які використовуються при побудові прошароку Кохонена.



а



б

Рис. 2

Тепер, коли сформована самоорганізаційна нейронна мережа, потрібно навчити її вирішенню завдання кластеризації даних. Нагадаємо, кожен нейрон блоку compet конкурує за право відповісти на вектор входу *p*. Якщо всі зміщення дорівнюють 0, то нейрон з вектором ваги, найближчим до вектора входу *p*, виграє конкуренцію і повертає на виході значення 1; усі інші нейрони повертають значення 0.

Правило навчання прошароку Кохонена

Правило навчання прошароку Кохонена (або *правило Кохонена*) полягає у тому, щоб налаштувати належним чином елементи матриці ваг. Припустимо, що нейрон ***i*** переміг при подачі входу $p(q)$ на кроці самонавчання q, тоді рядок ***i*** матриці ваг коригується відповідно до правила Кохонена наступним чином:

$$ {}_i\mathbf{IW}_{11}(q) = {}_i\mathbf{IW}_{11}(q-1) + \alpha\left(\mathbf{p}(q) - {}_i\mathbf{IW}_{11}(q-1)\right). $$

Правило Кохонена є рекурентним співвідношенням, яке забезпечує корекцію рядка ***i*** матриці ваг додаванням зваженої різниці вектора входу і значення рядка на попередньому кроці. Таким чином, вектор ваг, найближчий до вектора входу, модифікується так, щоб відстань між ними стала ще меншою. Результат такого навчання полягатиме в тому, що нейрон, що переміг, ймовірно, виграє конкуренцію і в тому випадку, коли буде

представлений новий вхідний вектор, близький до попереднього, і його перемога менш ймовірна, коли буде представлений вектор, що істотно відрізняється від попереднього. Коли на вхід мережі надходить все більше і більше векторів, нейрон, що є найближчим, знову коригує свій ваговий вектор. Зрештою, якщо в прошарку є достатня кількість нейронів, то кожна група близьких векторів виявиться пов'язаною з одним із нейронів прошароку. У цьому полягає властивість самоорганізації прошароку Кохонена.

Налаштування параметрів мережі за правилом Кохонена реалізовано у вигляді М-функції learnk.

Правило налаштування зміщень

Одне з обмежень будь-якого конкуруючого прошароку у тому, деякі нейрони виявляються незадіяними. Це проявляється в тому, що нейрони, що мають початкові вагові вектори, значно віддалені від векторів входу, ніколи не виграють конкуренції, незалежно від того, як довго продовжується навчання. В результаті виявляється, що такі вектори не використовуються при навчанні, і відповідні нейрони ніколи не виявляються переможцями. Такі нейрони-невдахи називають "*мертвими*" нейронами, оскільки вони не виконують жодної корисної функції. Щоб виключити таку ситуацію і зробити нейрони чутливими до векторів, що надходять на вхід, використовуються зсуви, які дозволяють нейрону стати конкурентним з нейронами-переможцями. Цьому сприяє додатне зміщення, яке додається до від'ємної відстані до віддаленого нейрона.

Відповідне правило налаштування, що враховує нечутливість мертвих нейронів, реалізоване у вигляді М-функції learncon і полягає в наступному. На початку процедури налаштування всім нейронам конкуруючого прошароку надається однаковий параметр активності

$$c_0 = \frac{1}{N},$$

де N – кількість нейронів конкуруючого прошароку, що дорівнює кількості кластерів. У процесі налаштування М-функція learncon коригує цей параметр таким чином, щоб його значення для активних нейронів ставали більшими, а для неактивних нейронів меншими. Відповідна формула для вектора приросту параметрів активності виглядає наступним чином:

$$\Delta\mathbf{c} = lr * (\mathbf{a}_i^1 - \mathbf{c}),$$

де lr - параметр швидкості налаштування; $a_{i^*}^1$ - Вектор, елемент *i** якого дорівнює 1, а інші - 0.

Неважко переконатися, що для всіх нейронів, крім нейрона-переможця, прирости від'ємні. Оскільки параметри активності пов'язані зі зміщенням співвідношенням (в позначках MATLAB)

$$\mathbf{b} = \exp(1)./\mathbf{c},$$

то з цього випливає, що зміщення для нейрона-переможця зменшиться, а зміщення для інших нейронів трохи збільшаться.

М-функція learncon використовує таку формулу для розрахунку приростів вектора зміщень

$$\Delta\mathbf{b} = \exp(1 - \log(\mathbf{c})) - \mathbf{b}.$$

Параметр швидкості налаштування lr за замовчуванням дорівнює 0.001, і його величина зазвичай на порядок менша за відповідне значення для М-функції learnk. Збільшення зсувів для неактивних нейронів дозволяє розширити діапазон покриття вхідних значень і неактивний нейрон починає формувати кластер. Зрештою, він може почати притягувати нові вхідні вектори.

Це дає дві переваги. Якщо нейрон не виграє конкуренції, тому що його вектор ваг істотно відрізняється від векторів, що надходять на вхід мережі, його зміщення в міру навчання стає досить великим і він стає конкурентоспроможним. Коли це відбувається,

його вектор ваги починає наближатися до певної групи векторів входу. Як тільки нейрон починає перемагати, його зміщення починає зменшуватися. Таким чином, завдання активізації "мертвих" нейронів виявляється вирішеним. Друга перевага, пов'язана з налаштуванням зсувів, полягає в тому, що вони дозволяють вирівняти значення параметра активності та забезпечити "притягування" приблизно однакової кількості векторів входу. Таким чином, якщо один із кластерів притягує більшу кількість векторів входу, ніж інший, то біль заповнена область притягне додаткову кількість нейронів і будеподілена на менші по розміру кластери.

Навчання мережі

Реалізуємо 10 циклів навчання. Для цього можна використовувати функції train або adapt:

```
net.trainParam.epochs = 10;
net = train (net, p);
net.adaptParam.passes = 10;
[net, y, e] = adapt (net, mat2cell (p));
```

Зауважимо, що для мереж з конкуруючим прошарком за замовчуванням використовується навчальна функція trainwb1, яка на кожному циклі навчання випадково вибирає вхідний вектор і пред'являє мережі; після цього проводиться корекція ваг і зміщень.

Виконаємо моделювання мережі після навчання:

```
a = sim (net, p);
ac = vec2ind(a)
ac = 2 1 2 1
```

Бачимо, що мережа навчена класифікації векторів входу на 2 кластери: перший розташований в околі вектора (0, 0), другий – в околі вектора (1, 1). Результуючі ваги та зміщення рівні:

```
wts1 = net.IW {1,1}
b1 = net.b {1}
wts1 =
0.58383 0.58307
0.41712 0.42789
b1 = 5.4152
5.4581
```

Зауважимо, що перший рядок вагової матриці дійсно близький до вектора (1, 1), тоді як другий рядок близький до початку координат. Таким чином, сформована мережа навчена класифікації входів. У процесі навчання кожен нейрон у прошарку, ваговий вектор якого близький до групи векторів входу стає визначальним для цієї групи векторів. В кінцевому рахунку, якщо є достатня кількість нейронів, кожна група векторів входу матиме нейрон, який виводить 1, коли представлений вектор цієї групи, і 0 в іншому випадку, або, іншими словами, формується кластер. Таким чином, прошарок Кохонена справді вирішує завдання кластеризації векторів входу.

Приклад:

Функціонування прошароку Кохонена можна пояснити наочно, використовуючи графіку системи MATLAB. Розглянемо 48 випадкових векторів на площині, що формують 8 кластерів, що групуються біля своїх центрів. На графіку, наведеному на рис. 3 показано 48 двоелементних векторів входу.

Сформуємо координати випадкових точок та побудуємо план їх розташування на площині:

```
c = 8; n = 6; % Число кластерів, векторів у кластері
```

```
d = 0.5; % Середньоквадратичне відхилення від центру кластера
x = [-10 10; -5 5]; % Діапазон вхідних значень
[r, q] = size (x); minv = min(x')'; maxv = max(x')';
v = rand (r, c). * ((maxv - minv) * ones (1, c) + minv * ones
(1, c));
t = c * n; % Число точок
v = [vvvvvv]; v=v+randn(r,t)*d; % Координати точок
P = v;
plot(P(1,:), P(2,:),'+k') % Рис.3
title(' Вектори входу'), xlabel('P(1,:)'), ylabel('P(2,:)')
```

Вектори, показані на рис. 3, відносяться до різних класів.

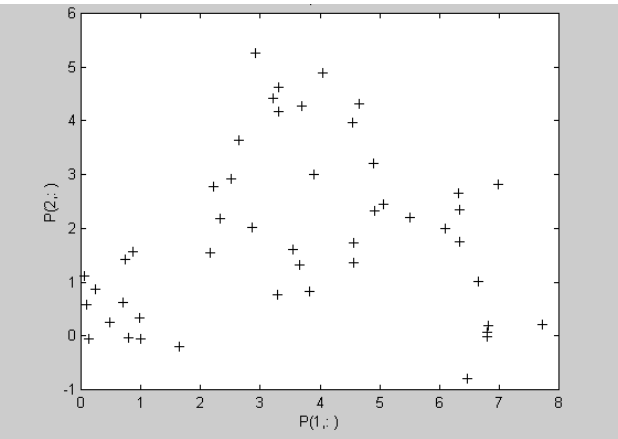


Рис. 3

Застосуємо конкуруючу мережу з восьми нейронів для того, щоб розподілити їх за класами:

```
net = newc ([-2 12; -1 6], 8, 0.1);
w0 = net.IW {1}
b0 = net.b {1}
c0 = exp (1).
```

Початкові значення ваг, зміщень та параметрів активності нейронів представлені в таблиці:

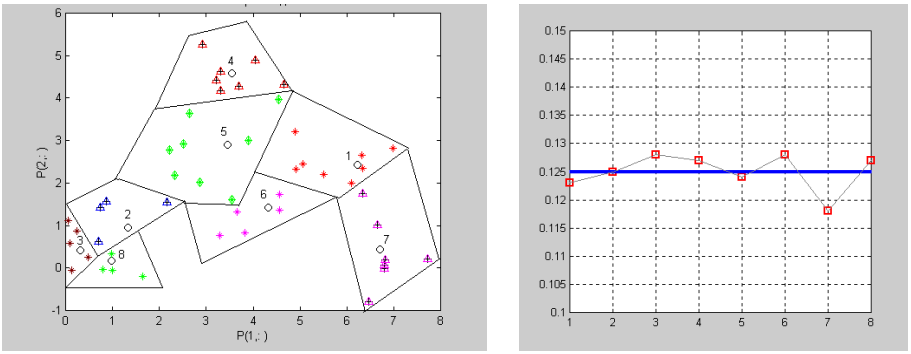
w0 =	b0 =	c0 =
0.50.25	21.746	0.125
0.50.25	21.746	0.125
0.50.25	21.746	0.125
0.50.25	21.746	0.125
0.50.25	21.746	0.125
0.50.25	21.746	0.125
0.50.25	21.746	0.125
0.5 0.25	21.746	0.125

Після навчання протягом 500 циклів отримаємо:

```
net.trainParam.epochs = 500;
net = train (net, P);
w = net.IW {1}
bn = net.b {1}
cn = exp(1).
```

```
w =                    bn =          cn =
6.2184 2.4239          22.137        0.123
1.3277 0.94701         21.718        0.125
0.31139 0.40935        21.192        0.128
3.543 4.5845           21.472        0.127
3.4617 2.8996          21.957        0.124
4.3171 1.4278          21.185        0.128
6.7065 0.43696         23.006        0.118
0.97817 0.17242        21.42         0.127
```

Як випливає з наведених таблиць, центри кластеризації розподілилися по восьми областях, показаних на рис. 4 а; зміщення відхилилися в обидві сторони від вихідного значення 21.746 так само, як і параметри активності нейронів, показані на рис. 4, б.



а *б*

Рис. 4

Розглянута самоорганізаційна мережа Кохонена, є типовим прикладом мережі, яка реалізовує процедуру навчання без вчителя.

Демонстраційний приклад democ1 також пояснює процедуру навчання самоорганізаційної мережі.

Карта Кохонена

Мережа, що самоорганізовується, у вигляді карти Кохонена призначена для вирішення завдань кластеризації вхідних векторів. На відміну від прошароку Кохонена, карта Кохонена підтримує таку топологічну властивість, коли близьким кластерам вхідних векторів відповідають близько розташовані нейрони.

Початкова топологія розміщення нейронів у прошарку Кохонена визначається М-функціями gridtop, hextop або randtop, що відповідає розміщенню нейронів у вузлах або прямокутної, або гексагональної сітки, або у вузлах сітки з випадковою топологією. Відстані між нейронами обчислюються за допомогою спеціальних функцій обчислення відстаней dist, boxdist, linkdist та mandist.

Карта Кохонена визначення нейрона-переможця використовує ту ж процедуру, яка застосовується і у прошарку Кохонена. Однак на карті Кохонена одночасно змінюються вагові коефіцієнти сусідніх нейронів відповідно до наступного співвідношення, яке називається *правилом Кохонена*:

$$_i\mathbf{w}(q) = (1-\alpha)\,_i\mathbf{w}(q-1) + \alpha\mathbf{p}(q).$$

В цьому випадку окіл нейрона-переможця включає всі нейрони, які знаходяться в межах деякого радіусу d:

$$N_i(d) = \{j,\ d_{ij} \le d\}.$$

Щоб пояснити поняття околу нейрона, звернемося до рис. 5.

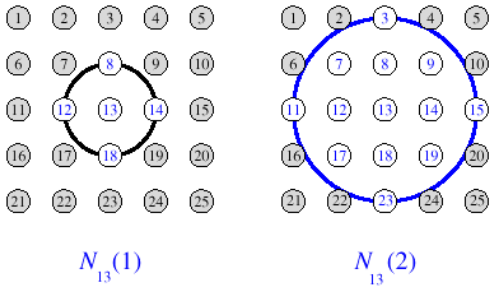


Рис. 5

Ліва частина малюнка відповідає околу радіусу 1 для нейрона-переможця з номером 13; права частина – околу радіуса 2 того ж нейрона. Опис цих околів виглядає наступним чином:

$$\begin{cases} N_{13}(1) = \{8, 12, 13, 14, 18\}; \\ N_{13}(2) = \{3, 7, 8, 9, 11, 12, 13, 14, 15, 17, 18, 19, 23\}. \end{cases}$$

Зауважимо, що топологія карти розташування нейронів не обов'язково має бути двовимірною. Це можуть бути і одновимірні та тривимірні карти, і навіть карти великих розмірностей. У випадку одномірної карти Кохонена, коли нейрони розташовані вздовж лінії, кожен нейрон матиме лише двох сусідів у межах радіусу 1 або єдиного сусіда, якщо нейрон розташований на кінці лінії. Відстань між нейронами можна визначати різними способами, використовуючи прямокутні або гексагональні сітки, проте це не впливає на характеристики мережі, пов'язані з класифікацією вхідних векторів.

Топологія карти

Як зазначалося вище, можна встановити різні топології для карти розташування нейронів, використовуючи М-функції gridtop, hextop, randtop.

Розглянемо найпростішу прямокутну сітку розміру 2×3 для розміщення шести нейронів, яка може бути сформована за допомогою функції gridtop:

```
pos = gridtop(2,3)
pos =
0 1 0 1 0 1
0 0 1 1 2 2
plotsom(pos) % Мал.6
```

Відповідна сітка показано на рис. 6. 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. розмірності карти.

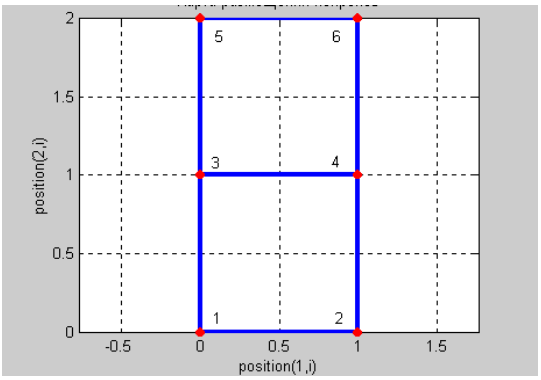


Рис. 6

Тут нейрон 1 розташований у точці з координатами (0,0), нейрон 2 – у точці (1,0), нейрон 3 – у точці (0,1) і т. д. Зауважимо, що якщо застосувати команду gridtop, переставивши аргументи місцями, отримаємо інше розміщення нейронів:

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

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) % Мал.7
```

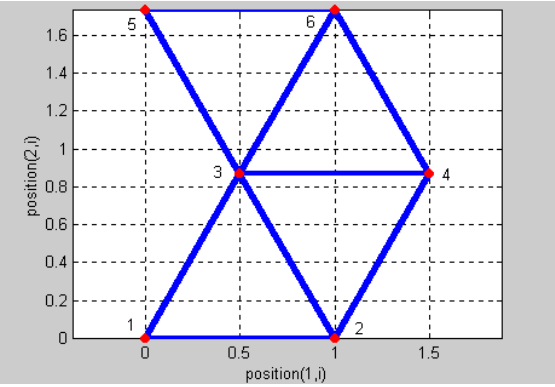


Рис. 7

Зауважимо, що М-функція hextop використовується за замовчуванням при створенні карт Кохонена при застосуванні функції newsom.

Сітка з випадковим розташуванням вузлів може бути створена за допомогою функції randtop:

```
pos = randtop(2,3)
pos =
   0.061787 0.64701 0.40855 0.94983 0 0.65113
0 0.12233 0.90438 0.54745 1.4015 1.5682
plotsom(pos) % Рис. 8
```

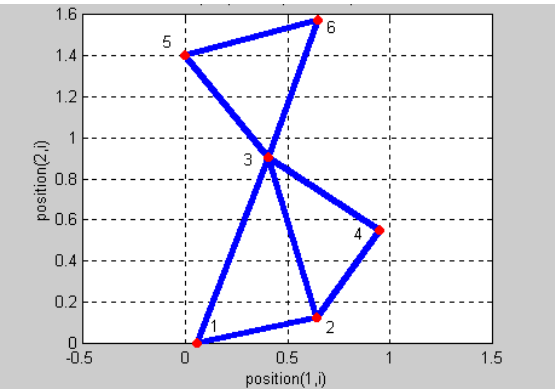


Рис. 8

Функції для розрахунку відстаней

У ППП NNT використовується 4 функції розрахунку відстаней між вузлами сітки.

Функція dist обчислює евклідову відстань між нейронами, розміщеними у вузлах сітки, відповідно до формули

$$d = \text{sqrt}(\text{sum}((\mathbf{pos}_i - \mathbf{pos}_j).\text{^}2)),$$

де posi, posj - вектори координат нейронів з номерами i і j.

Звернемося до прямокутної сітки з шести нейронів (див. рис. 6) та обчислимо відповідний масив відстаней:

```
pos = gridtop(2,3);
d = dist(pos)
d =
0       1       1    1.4142       2    2.2361
1       0  1.4142       1    2.2361       2
1    1.4142       0       1       1    1.4142
1.4142 1       1       0    1.4142       1
2    2.2361       1    1.4142       0       1
2.2361 2  1.4142       1       1       0
```

Цей масив розміру 6×6 описує відстані між нейронами і містить по діагоналі нулі, оскільки вони визначають відстань нейрона до самого себе, а потім, рухаючись вздовж рядка, - до другого, третього і т. д.

На рис. 9 показано розташування нейронів у вузлах прямокутної сітки. Введемо поняття околу прямокутної сітки. У цьому випадку окіл розміру 1 або просто окіл 1 включає базовий нейрон і його безпосередніх сусідів; окіл 2 включає нейрони з околу 1 та їх сусідів.

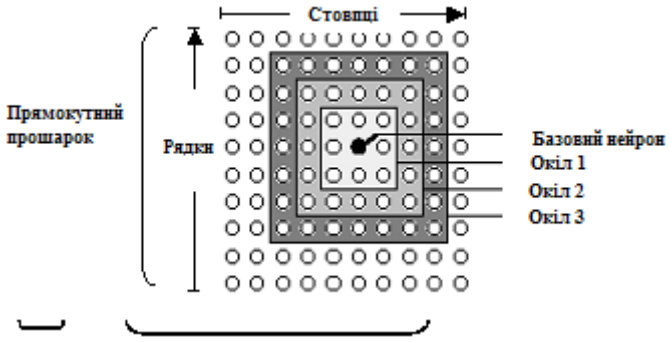


Рис. 9

Розмір, відповідно і номер околу, визначається максимальним значенням координати зміщення нейрона від базового. Відстань між нейронами, яка вводиться таким способом, називається відстанню максимального координатного зсуву і може бути обчислена за формулою

$$d = \max(\text{abs}(\mathbf{pos}_i - \mathbf{pos}_j)),$$

де posi, posj - вектори координат (положення) нейронів з номерами *i* і *j*.

Для обчислення цієї відстані в ППП NNT призначена М-функція boxdist. Для конфігурації нейронів, показаної на рис. 6 ці відстані рівні:

```
pos = gridtop(2,3);
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
```

Відстань максимального координатного зміщення між базовим нейроном 1 і нейронами 2, 3 і 4 дорівнює 1, оскільки вони знаходяться в околі 1, а відстань між базовим нейроном і нейронами 5 і 6 дорівнює 2, і вони знаходяться в околі 2. Відстань

максимального координатного зміщення від нейронів 3 і 4 до всіх інших нейронів дорівнює 1.

Визначимо іншу відстань між нейронами, яка враховує ту кількість зв'язків, які необхідно встановити, щоб утворити шлях руху від базового нейрона. Якщо встановлено S нейронів, положення яких визначається векторами p_i, $i = 1, \ldots, S$, то відстань зв'язку між ними визначається співвідношенням

$$d_{ij} = \begin{cases} 1, & dist(\mathbf{pos}_i - \mathbf{pos}_j) \le 1; \\ 2, & \forall k, d_{ik} = d_{kj} = 1; \\ 3, & \forall (k_1, k_2), d_{ik_1} = d_{k_1k_2} = d_{k_2j} = 1; \\ N, & \forall (k_1, k_2 \ldots k_N), d_{ik_1} = d_{k_1k_2} = \ldots = d_{k_Nj} = 1; \\ S, & y \quad інших \quad випадках. \end{cases}$$

Якщо евклідова відстань між нейронами менша або дорівнює 1, то відстань зв'язку приймається рівною 1; якщо між нейронами з номерами *i* та *j* є єдиний проміжний нейрон з номером *k*, то відстань зв'язку дорівнює 2, і т.д.

Для обчислення відстані зв'язку в ППП NNT призначено функції linkdist. Для конфігурації нейронів, показаної на рис. 6 ці відстані рівні:

```
pos = gridtop(2,3);
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
```

Відстань зв'язку між базовим нейроном 1 і нейронами 2, 3 дорівнює 1, між базовим нейроном і нейронами 4 і 5 дорівнює 2, між базовим нейроном і нейроном 6 дорівнює 3.

Нарешті, визначимо відстань максимального координатного зміщення за формулою

$$d = \text{sum}(\text{abs}(\mathbf{pos}_i - \mathbf{pos}_j)),$$

де posi, posj – вектори розташування нейронів із номерами *i* та *j*.

Для обчислення відстані максимального координатного зміщення в ППП NNT призначена функції mandist. Знову звернемося до зміни нейронів на рис. 6:

```
pos = gridtop(2,3);
d = mandist (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
```

У разі прямокутної сітки вона збігається з відстанню зв'язку.

Архітектура мережі

Архітектура самоорганізаційної карти Кохонена показана на рис. 10.

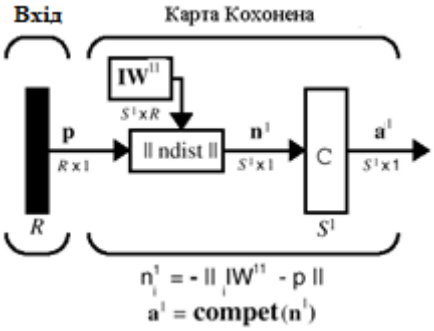


Рис. 10

Ця архітектура аналогічна структурі прошароку Кохонена за виключенням того, що не використовуються зміщення. Конкуруюча функція активації повертає 1 для елемента виходу a^1, відповідного нейрону, що переміг; усі інші елементи вектора a^1 дорівнюють 0.

Однак у мережі Кохонена виконується перерозподіл нейронів, що є сусідніми з нейроном, що переміг. При цьому можна вибирати різні топології розміщення нейронів та різні метрики для обчислення відстаней між нейронами.

Створення мережі

Для створення самоорганізаційної карти Кохонена, у складі ППП NNT передбачена М-функція newsom. Припустимо, що потрібно створити мережу для обробки двоелементних векторів входу з діапазоном зміни елементів від 0 до 2 і від 0 до 1 відповідно. Передбачається використовувати гексагональну сітку розміру 2×3. Тоді для формування такої нейронної мережі достатньо скористатися оператором

```
net = newsom([0 2; 0 1], [2 3]);
net.layers{1}
ans =
dimensions: [2 3]
distanceFcn: 'linkdist'
distances: [6×6 double]
initFcn: 'initwb'
netInputFcn: 'netsum'
positions: [2×6 double]
size: 6
topologyFcn: 'hextop'
transferFcn: 'compet'
userdata: [1×1 struct]
```

З аналізу характеристик цієї мережі випливає, що вона використовує за замовчуванням гексагональну топологію hextop та функцію відстані linkdist.

Для навчання мережі задамо наступні 12 двоелементних векторів входу:

```
P = [0.1 0.3 1.2 1.1 1.8 1.7 0.1 0.3 1.2 1.1 1.8 1.7; ...
     0.2 0.1 0.3 0.1 0.3 0.2 1.8 1.8 1.9 1.9 1.7 1.8];
```

Побудуємо на топографічній карті початкове розташування нейронів карти Кохонена та вершини векторів входу (рис. 11):

```
plotsom(net.iw{1,1},net.layers{1}.distances)
hold on
plot(P(1,:),P(2,:),'*k','markersize',10)
```

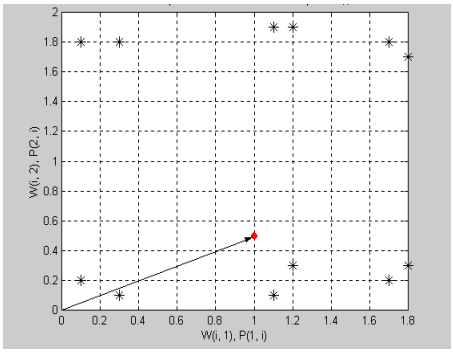


Рис. 11

Вектори входу позначені символом * і розташовані по периметру малюнка, а початкове розташування нейронів відповідає точці координат (1, 0.5).

Навчання мережі

Навчання самоорганізаційної карти Кохонена, реалізовується повекторно незалежно від того, виконується навчання мережі за допомогою функції trainwb1 або адаптація за допомогою функції adaptwb. У будь-якому випадку функція learnsom виконує налаштування елементів вагових векторів нейронів.

Насамперед визначається нейрон-переможець і коригуються його вектор ваг і вектори ваг сусідніх нейронів відповідно до співвідношення

$$\mathbf{dW} = lr * \mathbf{A2} * (\mathbf{p}' - \mathbf{w}),$$

де lr - параметр швидкості навчання; A2 – масив параметрів сусідства для нейронів, розташованих в околі нейрона-переможця i, який обчислюється за співвідношенням

$$A2(i,q) = \begin{cases} 1, & a(i,q) = 1; \\ 0.5, & a(j,q) = 1 \,\&\, D(i,j) \le nd; \\ 0, & у \quad інших \quad випадках, \end{cases}$$

де a(i, q) – елемент виходу нейронної мережі; D(i, j) – відстань між нейронами i та j; nd – розмір околу нейрона-переможця.

Вагові вектори нейрона-переможця та сусідніх нейронів змінюються залежно від значення параметра сусідства. Ваги нейрона-переможця змінюються пропорційно до параметра швидкості навчання, а ваги сусідніх нейронів – пропорційно до половини значення цього параметра.

Процес навчання карти Кохонена включає 2 етапи: етап упорядкування векторів вагових коефіцієнтів у просторі ознак та етап підлаштування. При цьому використовуються такі параметри мережі:

Параметри навчання та налаштування карти Кохонена			*Значення за замовчуванням*
Кількість циклів навчання	net.trainParam.epochs	N	>1000
Кількість циклів на етапі упорядкування	net.inputWeights{1,1}.learnParam.order_steps	S	1000
Параметр швидкості навчання на етапі впорядкування	net.inputWeights{1,1}.learnParam.order_lr	order_lr	0.9
Параметр швидкості навчання на етапі підлаштування	net.inputWeights{1,1}.learnParam.tune_lr	tune_lr	0.02
Розмір околу на етапі підлаштування	net.inputWeights{1,1}.learnParam.tune_nd	tune_nd	1

У процесі побудови карти Кохонена змінюються два параметри: розмір околу та параметр швидкості навчання.

Етап упорядкування. На цьому етапі використовується фіксована кількість кроків. Початковий розмір околу призначається рівним максимальній відстані між нейронами для обраної топології і потім зменшується до величини, що використовується на етапі підлаштування відповідно до наступного правила:

$$nd = 1.00001 + (\max(d) - 1)(1 - \frac{s}{S}),$$

де max(d) – максимальна відстань між нейронами; s – номер поточного кроку.

Параметр швидкості навчання змінюється за правилом

$$lr = tune\_lr + (order\_lr - tune\_lr)(1 - \frac{s}{S}).$$

Таким чином він зменшується від значення order_lr до значення tune_lr.

Етап підлаштування. Цей етап триває протягом частини процедури навчання, яка залишилась. Розмір околу на цьому етапі залишається постійним та рівним

$$nd = tune\_nd + 0.00001.$$

Параметр швидкості навчання змінюється за таким правилом:

$$lr = tune\_lr \frac{S}{s}.$$

Параметр швидкості навчання продовжує зменшуватись, але дуже повільно, і саме тому цей етап називається підлаштуванням. Мале значення околу та повільне зменшення параметра швидкості навчання добре налаштовують мережу при збереженні розміщення, знайденого на попередньому етапі. Кількість кроків на етапі підлаштування має значно перевищувати кількість кроків на етапі розміщення. На цьому етапі відбувається тонка настройка ваги нейронів по відношенню до набору векторів входу.

Як і у випадку прошароку Кохонена, нейрони карти Кохонена будуть упорядковуватися так, щоб при рівномірній щільності векторів входу нейрони карти Кохонена також були рівномірно розподілені. Якщо вектори входу розподілені нерівномірно, то і нейрони на карті Кохонена матимуть тенденцію розподілятися відповідно до щільності розміщення векторів входу.

Отже, під час навчання карти Кохонена вирішується не лише завдання кластеризації вхідних векторів, а й виконується часткова класифікація.

Виконаємо навчання карти Кохонена розміру 2×3 з гексагональною сіткою та з метрикою, що визначається відстанню зв'язку

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

Для навчання мережі задамо 12 двоелементних векторів входу

```
P = [0.1 0.3 1.2 1.1 1.8 1.7 0.1 0.3 1.2 1.1 1.8 1.7; ...
    0.2 0.1 0.3 0.1 0.3 0.2 1.8 1.8 1.9 1.9 1.7 1.8];
```

Задамо кількість циклів навчання рівним 2000:

```
net.trainParam.epochs = 2000;
net.trainParam.show = 100;
net = train (net, P);
plot(P(1,:),P(2,:),'*','markersize',10)
hold on
plotsom(net.iw{1,1},net.layers{1}.distances)
```

Результат навчання подано на рис. 12.

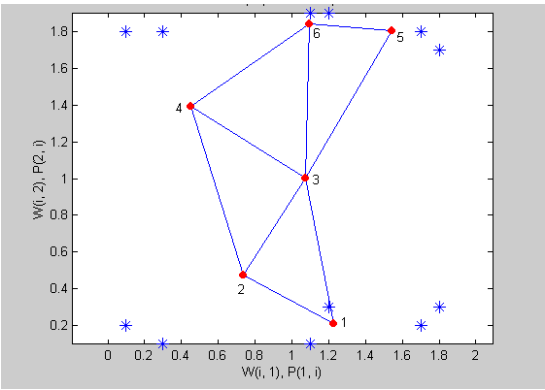


Рис. 12

Положення нейронів та їх нумерація визначаються масивом вагових векторів, який для даного прикладу має вигляд:

```
net.IW{1}
ans =
1.2163 0.20902
0.73242 0.46577
1.0645 0.99109
0.4551 1.3893
1.5359 1.8079
1.0888 1.8433
```

Якщо промоделювати карту Кохонена на масиві навчальних векторів входу, буде отримано наступний вихід мережі:

```
a = sim (net, P)
a =
(2,1) 2
(2,2) 2
(1,3) 1
(1,4) 1
(1,5) 1
(1,6) 1
(4,7) 4
(4,8) 4
(6,9) 6
(6,10) 6
(5,11) 5
(5,12) 5
```

Це означає, що вектори входів 1 та 2 віднесені до кластера з номером 2, вектори 3–6 – до кластера 1, вектори 7–8 – до кластера 4, вектори 9–10– до кластера 6, а вектори11–12– до кластера 5. Номер кластера на малюнку відповідає номеру відповідного нейрона на карті Кохонена.

Якщо сформувати довільний вектор входу, то карта Кохонена має вказати його належність до того чи іншого кластера:

```
a = sim (net, [1.5; 1])
a = (3,1) 1
```

В даному випадку представлений вектор входу віднесений до кластера з номером 3. Зверніть увагу, що такого вектора такого у навчальної послідовності не було.

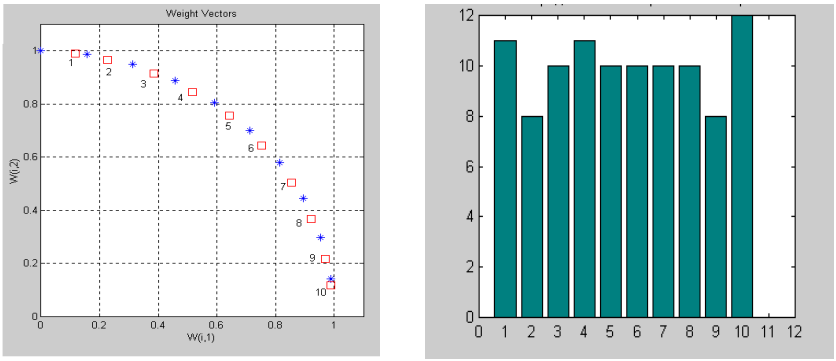
Розглянемо ще 2 приклади одновимірної та двовимірної карт Кохонена.

Одновимірна карта Кохонена

Розглянемо 100 двоелементних вхідних векторів одиничної довжини, рівномірно розподілених в межах від 0 до 90°:

```
angles = 0:0.5*pi/99:0.5*pi;
P = [sin(angles); cos(angles)];
plot(P(1,1:10:end), P(2,1:10:end),'*b'), hold on
```

Графік вхідних векторів наведено на рис. 13 а, і на ньому символом * відзначено положення кожного 10-го вектора.



а *б*

Рис. 13

Сформуємо саморганізаційну картку Кохонена у вигляді одномірного прошароку з 10 нейронів і виконаємо навчання протягом 2000 циклів:

```
net = newsom([0 1; 0 1], [10]);
net.trainParam.epochs = 2000;
net.trainParam.show = 100;
[net, tr] = train (net, P);
plotsom(net.IW{1,1},net.layers{1}.distances) % Рис.13,а
figure(2)
a = sim (net, P);
bar(sum(a')) % Рис. 13,б
```

Вагові коефіцієнти нейронів, що визначають центри кластерів, відзначені на рис. 13, *а* цифрами. На рис. 13, *б* показано розподіл навчальних векторів за кластерами. Як і очікувалося, вони розподілені практично рівномірно з розкидом від 8 до 12 векторів у кластері.

Таким чином, мережа підготовлена до кластеризації вхідних векторів. Визначимо, до якого кластера буде віднесено вектор [1; 0]:

```
a = sim (net, [1; 0])
a = (10,1) 1
```

Як і слід було очікувати, його віднесено до кластера з номером 10.

Двовимірна карта Кохонена

Цей приклад демонструє навчання двовимірної карти Кохонена. Спочатку створимо навчальний набір випадкових двовимірних векторів, елементи яких розподілені за рівномірним законом в інтервалі [-1 1]:

```
P = rands (2,1000);
plot(P(1,:),P(2,:),'+') % Рис.14
```

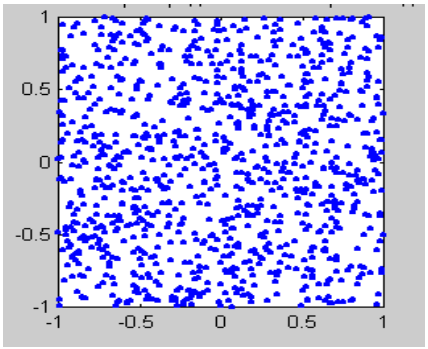
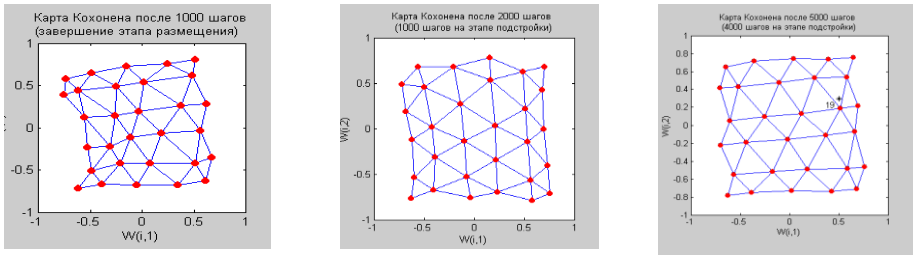


Рис. 14

Для кластеризації векторів входу створимо самоорганізаційну картку Кохонена розміру 5×6 з 30 нейронами, розміщеними на гексагональній сітці:

```
net = newsom([-1 1; -1 1], [5,6]);
net.trainParam.epochs = 1000;
net.trainParam.show = 100;
net = train (net, P);
plotsom(net.IW{1,1},net.layers{1}.distances)
```

Результуюча карта після етапу розміщення показана на рис. 15 *а*. Продовжимо навчання та зафіксуємо карту після 1000 кроків етапу підлаштування (рис. 15, б), а потім після 4000 кроків (рис. 15, в). Неважко переконатися, що нейрони карти рівномірно покривають область векторів входу.



а *б* *в*

Рис. 15

Визначимо приналежність нового вектора до одного із кластерів карти Кохонена та побудуємо відповідну точку на рис. 15, в:

```
a = sim (net, [0.5; 0.3])
a = (19,1) 1
hold on, plot(0.5,0.3,'*k') % Рис.15,в
```

Неважко переконатися, що вектор віднесений до 19 кластера.

Промоделюємо навчену карту Кохонена, використовуючи масив векторів входу:

```
a = sim (net, P);
bar(sum(a')) % Рис.16
```

З аналізу рис.16 випливає, що кількість векторів вхідної послідовності, віднесених до певного кластера, коливається від 13 до 50.

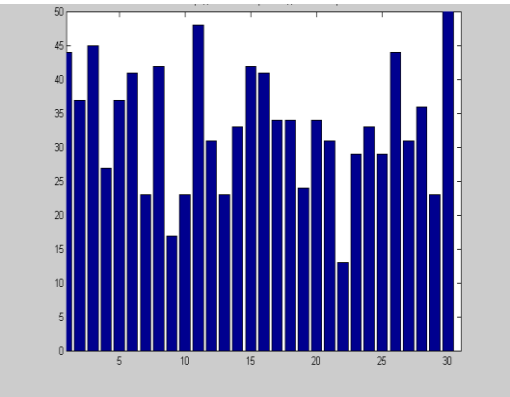


Рис. 16

Таким чином, у процесі навчання двовимірної самоорганізаційної карти Кохонена виконано кластеризацію масиву векторів входу. Слід зазначити, що на етапі розміщення було виконано лише 20 % від загальної кількості кроків навчання, тобто 80 % загального часу навчання пов'язані з тонкою підбудовою вагових векторів. Фактично на цьому етапі виконується певною мірою класифікація вхідних векторів.

Прошарок нейронів карти Кохонена можна уявляти собі у вигляді гнучкої сітки, яка натягнута на простір вхідних векторів. У процесі навчання карти, на відміну від прошароку Кохонена, беруть участь сусіди нейрона-переможця, і, таким чином, топологічна карта виглядає більш упорядкованою, ніж області кластеризації прошароку Кохонена.

Читач може продовжити вивчення самоорганізаційних мереж звернувшись до демонстраційних програм demosm1 і demosm2.