

PRÀCTICA 3: MAPES AUTO-ORGANITZATIUS DE KOHONEN

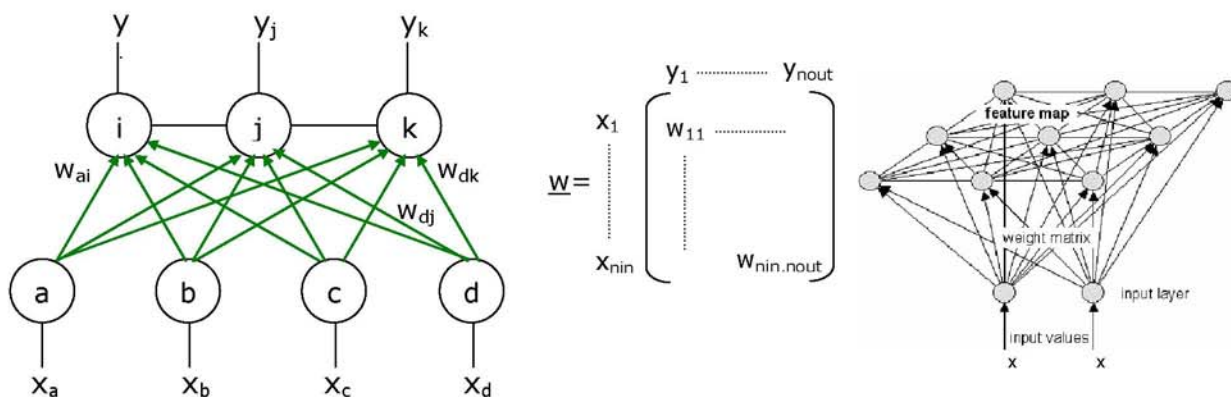
OBJECTIUS

En aquesta sessió de pràctiques treballareu amb l'algorisme no supervisat d'entrenament de Kohonen per a mapes auto-organitzatius (SOM/SOFM). Els objectius de la pràctica són:

- Implementar l'algorisme d'aprenentatge de Kohonen per a SOM.
- Veure l'efecte que tenen sobre les solucions trobades i l'estructura de la xarxa:
 - El nombre de cicles de l'aprenentatge
 - L'arquitectura de la xarxa
 - El coeficient d'aprenentatge
 - El terme de veïnatge

ENTORN DE DESENVOLUPAMENT

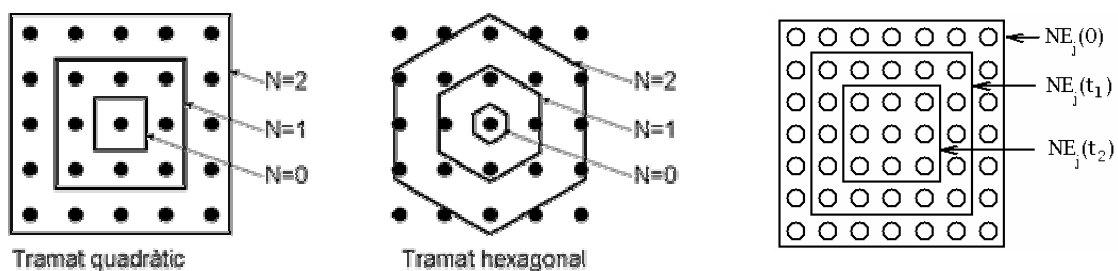
La pràctica es realitzarà en Matlab sobre un conjunt de rutines ja programades que permetran crear la xarxa i avaluar-ne el comportament.



Xarxes SOM

Una xarxa SOM està composta per una capa competitiva i una capa d'entrada. A diferència de les xarxes competitives generalitzades, la capa competitiva de la SOM no té pesos, sinó simples connexions seguint un tram arbritari, mentre que la capa d'entrada conté pesos (és *fully-connected* entre entrades i unitats) adaptables que són els que modificarem per entrenar la xarxa.

La metodologia d'entrenament d'una SOM és força intuïtiva. L'algorisme de Kohonen calcula inicialment la sortida de totes les unitats (típicament utilitzant la distància euclidiana com a mesura de proximitat entre el vector de pesos i el vector d'entrades). Donades les sortides de totes les unitats, l'algorisme tria llavors la unitat de màxima activació i la assigna com a guanyadora. L'entrenament té lloc llavors mitjançant la regla d'actualització de pesos $\underline{w}^j = \underline{w}^j + \eta \cdot (\underline{x}_i - \underline{w}^j)$, que s'aplica a la unitat guanyadora i a totes les seves veïnes definides segons el tram de connexió i un coeficient de distància sobre aquest (el coeficient de veïnatge N_j). La terminació de l'algorisme és en funció del valor del coeficient d'aprenentatge η , que disminueix a cada època, multiplicat per una constant en l'interval (0,1). Cada n èpoques (paràmetre arbitrari), es disminueix també el coeficient de veïnatge ($N_j = N_j - 1$) provocant la segregació progressiva d'unitats veïnes i donant lloc al mapa auto-organitzat.



L'algorisme de Kohonen queda doncs resumit com:

Algorisme de Kohonen

Iterar

Per tots els vectors entrada

- 1 - Aplicar vector entrada x_i i avaluar la sortida de cada unitat.
- 2 - Trobar la unitat guanyadora u_j (mínima distància a entrada).
- 3 - Entrenar pesos unitat guanyadora i veïnes ($\in N_j$)

$$\underline{w}^j = \underline{w}^j + \eta \cdot (\underline{x}_i - \underline{w}^j)$$

Fi

Decrementar η

Decrementar N_j

Fins terminació (en funció de η)

Definició de la xarxa

La rutina per definir l'estructura de la xarxa SOFM ja està programada; és la rutina *sofm*(*nin*, *nout*, *distfun*, *topology*):

sofm.m

```

%Construeix la xarxa SOFM
%Defineix els camps essencials de la SOFM:
% El tipus de xarxa
% El número de sortides (i.e. unitats) i entrades
% Les dimensions del vector de pesos
% La topologia (1D - linear o circular en el nostre cas)
% La funció d'activació a utilitzar
% i realitza una inicialització bàsica dels pesos a zero
function net = sofm(nin, nout, distfun, topology)

%Definir camps de la xarxa
net.type = 'sofm';
net.nin = nin;
net.nout = nout;
net.nwts = nin*nout;

%Llistat de funcions de distància
dists = {'de2', 'dot'};

%Assignació funció de distància
if sum(strcmp(distfun, dists)) == 0
    error('Funció activació no definida.');
```

```

else
    net.dist = distfun;
end

%Llistat de topologies
topos = {'linear', 'circular'};

%Assignació de topologia
if sum(strcmp(topos, topology)) == 0
    error('Topologia no definida.');
```

```

else
    net.topology = topology;
end

%Basic weight initialization
net.w = zeros(nin, nout);

end

```

Aquesta funció rep com a paràmetres d'entrada el número d'entrades i sortides (i.e. unitats) de la xarxa, així com un *string* que defineix la funció d'activació a utilitzar (només utilitzarem la distància euclidiana al quadrat) i un altre que defineix la topologia de la xarxa (que podrà ser *linear* o *circular*, però sempre 1D en aquesta pràctica). La funció duu a terme també una inicialització de pesos, però és una inicialització trivial a zero i caldrà fer una inicialització correcta a la funció d'entrenament.

```

>> xarxa=sofm(2,12,'de2','linear')
xarxa =
    type: 'sofm'
    nin: 2
    nout: 12
    nwts: 24
    dist: 'de2'
    topology: 'linear'
    w: [2x12 double]

```

Algorisme de Kohonen

En la pràctica, haureu d'implementar l'algorisme de Kohonen que es mostra a dalt i que subdividirem en dos apartats:

Funció `sofmfwd`

D'una banda, caldrà implementar la funció de propagació de la xarxa $y = \text{sofmfwd}(\text{net}, x)$, que genera una matriu de sortides a partir de l'estructura de xarxa i la matriu d'entrades que rep:

sofmfwd.m

```
%Funció que calcula les sortides de la xarxa a partir d'entrades i la pròpia estructura de xarxa
function y = sofmfwd(net, x)

%calcular el número de patrons d'entrada
ndata = size(x, 1);
%dimensionar el vector de sortida
y=zeros(ndata,net.nout);

%per cada patró d'entrada
    %depenent de la funció de propagació
        %calcular sortida unitats
        %avaluar unitat guanyadora

%retornar matriu amb 1 a unitat guanyadora i 0 a les altres

end
```

La funció $y = \text{sofmfwd}(\text{net}, x)$ rep com a entrada una matriu d'entrades on cada fila representa un patró i cada columna un component d'entrada. Com a sortida, genera també una matriu de sortides on cada fila és la sortida corresponent al patró d'entrada d'aquella mateixa fila a la matriu d'entrades i cada columna correspon al valor d'activació d'una unitat. Donat que Kohonen no implementa competició dinàmica entre les neurones, sinó que utilitza un àrbitre extern (l'algorisme), la funció $y = \text{sofmfwd}(\text{net}, x)$ no retorna directament la sortida de cada unitat, sinó que es limita a buscar el mínim de les sortides (la distància euclidiana mínima) per assignar una guanyadora, i retorna un vector on totes les posicions són zero excepte les de la unitat guanyadora (que és 1).

Funció `sofmlearn`

L'altra funció a implementar és la funció que duu a terme l'aprenentatge pròpiament dit. La funció $\text{net} = \text{sofmlearn}(\text{net}, x, \text{alpha}, \text{alphadec}, \text{trainstop}, \text{neighborhood}, \text{neighbordec}, \text{graphics})$ és l'encarregada de rebre una xarxa (net) i retornar-la entrenada aplicant l'algorisme de Kohonen.

La funció rep com a paràmetres la xarxa (*net*), el conjunt de patrons d'entrada (*x*), el coeficient d'aprenentatge (*alpha*), el decrement multiplicatiu a aplicar a cada època sobre el coeficient d'aprenentatge (*alphadec*) i el criteri de terminació en funció del coeficient d'aprenentatge (*trainstop*, el valor pel qual aturem l'aprenentatge quan *alpha* l'assoleix). A més, la funció rep també el coeficient de veïnatge (*neighborhood*) i cada quantes èpoques volem decrementar-lo (*neighbordec*). Finalment, la funció rep també una booleana (*graphics*) per determinar si es vol observar, o no, el gràfic d'evolució de la xarxa en temps real.

sofmfwd.m

```
%funció que duu a terme l'entrenament de la xarxa
%implementa un doble bucle (fins terminació / per cada patró)
%rep com entrades la xarxa, les entrades, el coeficient d'aprenentatge i el seu decrement, la condició de terminació, el
veïnatge i el seu decrement, i el booleà graphics per mostrar o no el gràfic d'evolució de la xarxa
function net = sofmlearn(net,x,alpha,alphadec,trainstop,neighborhood,neighbordec, graphics)

%inicialització aleatòria (0,1) de pesos ponderada sobre sqrt(entrades)

%mentre el coeficient d'aprenentatge més gran que criteri stop
  %per cada patró
    %calcular sortida pel patró
    %buscar unitat amb activació màxima
    %entrenar unitat guanyadora i veïnes
    %variant l'actualització en funció de la topologia

%incrementar èpoques
%decrementar coeficient
%decrementar veïnatge si toca
%plotejar unitats si toca
```

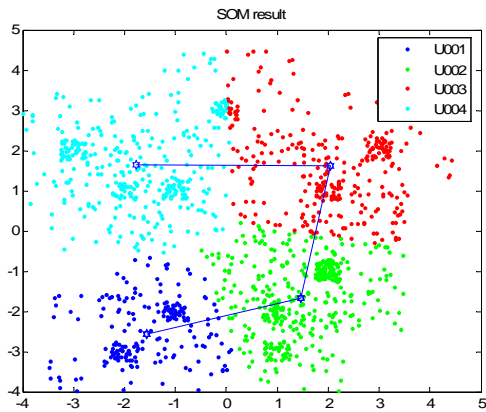
Cal recordar que l'entrenament ha de suportar dos tipus de malla: *linear* i *circular*, que requeriran actualitzacions lleugerament diferents en funció del veïnatge.

Rutines addicionals

Teniu també algunes rutines addicionals programades, que serveixen bàsicament per avaluar la xarxa i resseguir-ne la seva evolució.

Funció sofmtest

La funció *sofmtest(net,x)* rep com a paràmetres la xarxa entrenada i un conjunt de test, i genera un resultat gràfic on es pot observar sobre les dades d'entrada l'estructura de la xarxa, amb els seus nodes i connexions de la malla, i l'estructura de *clusters* que l'activació de cadascun d'aquests produeix sobre el conjunt d'entrada en diferents colors.



```
function somfstest(net,x);
class=cell(net.nout,1);
y=somfwd(net,x);
ndata=size(y,1);

for i=1:ndata,
    [m,p]=max(y(i,:));
    class(p)=[class(p);x(i,:)];
end

llegend=[];
figure,
hold on
axis([-4 5 -4 5])

plotstring1=['b','g','r','c','m','y','k'];
plotstring2=['.', 'o', 'x', '*', 's', 'd', 'v', '^', '<', '>', 'p', 'h'];

a=1; b=1;
for i=1:net.nout
    if ~isempty(class{i})
        plot(class{i}(:,2),class{i}(:,1), [plotstring1(a),plotstring2(b)]);
        llegend=[llegend;sprintf('U%03.0f',i)];
    end
    hold on
    axis([-4 5 -4 5])
    a=a+1;
    if (a>7)
        a=1;
        b=b+1;
        if (b>13)
            b=1;
        end
    end
end

end

legend(llegend);

classy=[];
for i=1:net.nout
    if ~isempty(class{i})
        classy(i,:)=mean(class{i});
        plot(classy(:,2),classy(:,1),'hb');
    end
end
plot(classy(:,2),classy(:,1));

set(gca, 'box', 'on')

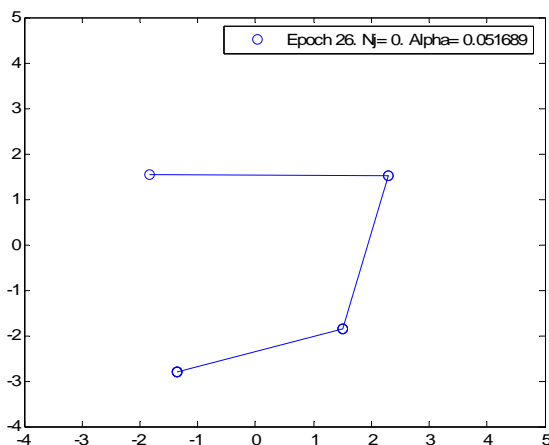
title('SOM result')
```

Funció plotplot

La funció [] = plotplot(net,x,n,alf,nb) serveix per observar gràficament l'evolució de la malla de connexió de la capa competitiva de Kohonen. La funció, que podeu cridar des de *sofmlearn* al completar cada època (d'aquí la opció *graphic* d'aquesta), rep com a paràmetres la xarxa, el conjunt d'entrades, el número d'època en que ens trobem (*n*), el coeficient d'aprenentatge actual (*alf*) i el veïnatge actual. Amb aquests valors mostra la posició de cada unitat (i la seva interconnexió amb les altres unitats).

```
function [] = plotplot(net,x,n,alf,nb)

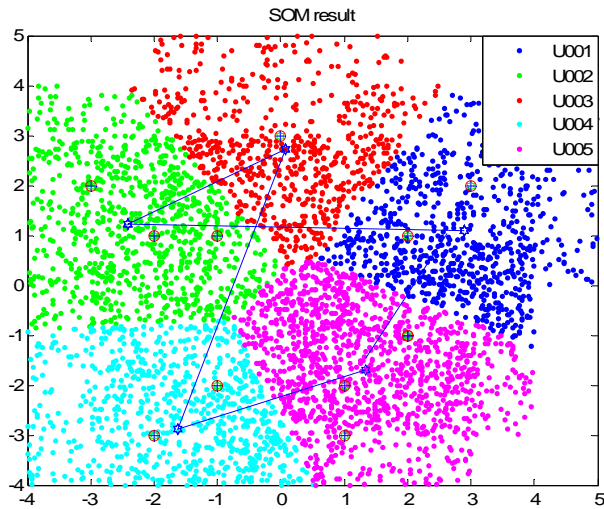
pause(0.01);
for i=1:net.nout
    plot(net.w(2,:),net.w(1,:),'bo');
    hold on;
end
plot(net.w(2,:),net.w(1,:));
legend(sprintf('Epoch %d. Nj= %d. Alpha= %1.6f',n,nb,alf));
axis([-4 5 -4 5]);
hold off;
```



Funcions testl i testc

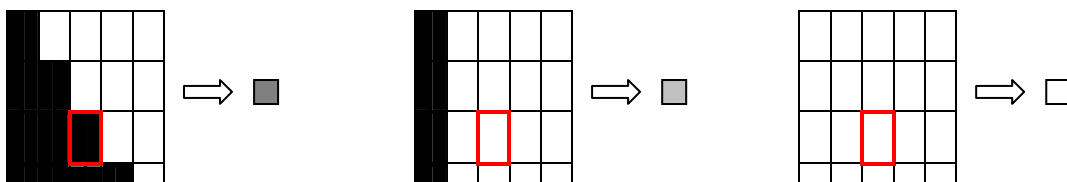
Les funcions (de fet *scripts*) *testl* i *testc* serveixen per testejar, respectivament, el funcionament de l'algorisme d'aprenentatge en el cas d'una topologia lineal (*testl*) o circular (*testc*). Els scripts carreguen un set de dades i criden a les funcions *sofm* i *sofmlearn* per entrenar una xarxa. Sobre els resultats de *sofptest*, sobreimprimeixen els centroides reals (coneguts per construcció del problema) de cada

cluster amb un símbol \oplus , i els centroides trobats per la xarxa amb estrelles. Els clusters d'entrada que activen cada node apareixen en diferents colors.



PART OPCIONAL: SEGMENTACIÓ D'IMATGES

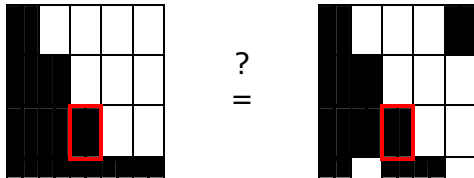
La segmentació d'imatges consisteix en dividir la imatge en regions que tenen un nivell de intensitat, tramat o color semblant. Per dur-la a terme, normalment es considera el veïnatge d'un punt. És a dir, la segmentació depèn dels valors d'intensitat que tenen un punt i els seus veïns (fins a una certa distància o radi: mida de la finestra). El patró que defineixen els punts de la finestra correspondrà a un nivell de gris en la nostra nova imatge. Per exemple:



de forma que la imatge original queda segmentada en una imatge més senzilla:



Aquest problema és clarament un problema de *clustering*, on hem de decidir quin nivell de gris correspon a cadascun dels píxels de la imatge. És a dir, decidir, per exemple, si als dos píxels de baix els correspon, o no, el mateix nivell de gris:



En altres paraules, el que pretenem és determinar si ambdós grups de píxels pertanyen a la mateixa classe (*cluster*) de sortida. Evidentment, aquest és un problema idoni per xarxes competitives, on el tipus de segmentació vindrà definit per la mida de la finestra que considerem (número entrades) i pel número de sortides (*clusters*) de que disposa la xarxa.

Utilitzeu la xarxa SOFM creada a la part obligatòria per segmentar imatges, passant-li com entrades els píxels necessaris dependent de la mida de finestra que utilitzeu (i.e. 9 entrades per mida 3). Proveu la xarxa amb diferents mides de finestra (e.g. 2, 3 o 4) i amb diferent número de sortides a la xarxa (e.g. 4, 16; diferents intensitats de grisos de sortida).

Per poder fer aquesta part necessitareu programar dues funcions addicionals: la funció Data.m [$x=Data(im,n,samples)$], que agafa d'una imatge (*im*) un número (*samples*) de píxels determinat de forma aleatòria amb els veïns corresponents (*n*) i ens els retorna com a conjunt d'aprenentatge, i la funció TestIm.m [$imout=TestIm(net,im)$], que rep una imatge (*im*) i la propaga a través de la xarxa entrenada (*net*) per retornar la imatge segmentada (*imout*).

Agafeu diferents imatges en grisos (o convertiu-les a grisos) que tinguin una temàtica semblant (carreteres, edificis, habitacions... <http://images.google.com/> és un bon lloc per obtenir-les) i utilitzeu-ne una per obtenir patrons aleatoris d'aprenentatge i entrenar la xarxa. Apliqueu llavors la segmentació a les altres imatges i comproveu que la xarxa ha après bé. En Matlab la imatge es pot llegir directament en BMP amb la comanda: $im=imread('im.bmp')$. Per visualitzar les

imatges podeu utilitzar `imshow(uint8(im))`. Recordeu que una imatge en Matlab es tracta igual que una matriu. També necessitareu la funció `rand` per a generar números aleatoris i la funció `double()` que fa un *cast* dels valors sencers de la imatge per a poder treballar amb ells com si fossin dades reals.

LLIURAMENT

La pràctica consta de dues parts diferenciades. En la primera, que és la que obligatòriament s'ha de completar al final de la sessió, cal haver programat les funcions per propagar i entrenar la xarxa (`sofmfwd` i `sofmlearn`), i executat les funcions `testl` i `testc` per demostrar que la xarxa funciona correctament. S'ha de comentar també, a la vista dels resultats, sobre quin problema creieu que pot ser idoni aplicar una xarxa SOFM. Es pot provar a més la xarxa amb algun altre conjunt de dades diferent del que utilitzen `testl` i `testc` (`dataset4`).

La segona part de la pràctica és opcional i es detalla a la secció anterior. El que cal lliurar a la part opcional és un informe breu i concís (dues o tres pàgines haurien de ser suficients) on es mostrin i descriguin algunes de les imatges utilitzades (no cal mostrar-les totes), la metodologia seguida (proves amb diferents veïnatsges, finestres, sortides, etc.), el codi implementat, els resultats (altra vegada no cal mostrar-los tots) i les conclusions que n'extraieu (quin tipus de finestra va millor, perquè, etc.), així com possibles millores que hagueu considerat i/o implementat (segmentació en color, etc.) i els problemes que comporten. Raoneu també la conveniència d'utilitzar el veïnatge diferent de zero (i.e. $N_j > 0$) enfront un esquema competitiu normal sense veïnatge ($N_j = 0$): millora la segmentació? Es pot aprofitar el veïnatge per millorar la segmentació?

Com s'ha esmentat abans, la primera part és obligatòria i s'ha de completar en acabar la sessió de pràctiques. L'informe de la part *opcional* s'ha de lliurar no més tard de la 4^a sessió de pràctiques.

APÈNDIX: OPERACIONS AMB MATRIUS I EVALUACIÓ

```
>> x=[[1 2 3 4 5];[6 7 8 9 10]]
```

```
x =
```

```
1 2 3 4 5  
6 7 8 9 10
```

```
>> y=[[1 2];[3 4];[5 6];[7 8];[9 10]]
```

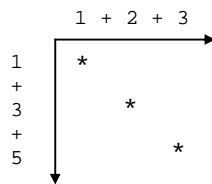
```
y =
```

```
1 2  
3 4  
5 6  
7 8  
9 10
```

```
>> x*y
```

```
ans =
```

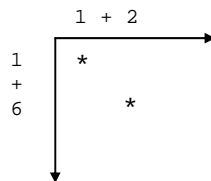
```
95 110  
220 260
```



```
>> y*x
```

```
ans =
```

```
13 16 19 22 25  
27 34 41 48 55  
41 52 63 74 85  
55 70 85 100 115  
69 88 107 126 145
```



```
>> x'
```

```
ans =
```

```
1 6  
2 7  
3 8  
4 9  
5 10
```

```
>> y'
```

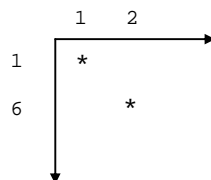
```
ans =
```

```
1 3 5 7 9  
2 4 6 8 10
```

```
>> x.*y'
```

```
ans =
```

```
1 6 15 28 45  
12 28 48 72 100
```



```
>> size(x)
```

```
ans =
```

```
2 5
```

```
>> size(y)
ans =
     5     2

>> size(x,1)
ans =
     2

>> ones(3,1)
ans =
     1
     1
     1

>> zeros(2,6)
ans =
     0     0     0     0     0     0
     0     0     0     0     0     0
```

```
>> w='sin(3) '
w =
sin(3)

>> eval(w)
ans =
    0.1411

>> sprintf('cos(%s)',w)
ans =
cos(sin(3))

>> eval(sprintf('cos(%s)',w))
ans =
    0.9901
```