Universitat de Girona

Departament d'Electrònica, Informàtica i Automàtica

Assignatura:
Disseny de sistemes de supervisió

Pràctica 3:
"XARXES NEURALS"

Pràctica 3:

"XARXES NEURALS"

Objectius de la pràctica:

- Comprensió dels conceptes bàsics del sistemes basats en xarxes neurals
- Utilització de la Neural Network Toolbox de MATLAB
- Disseny, prova i ajust d'un sistema senzill basat en xarxes neurals
- Entendre quina és l'aplicabilitat del sistemes basats en xarxes neurals

Contingut:

1. El Perceptró: una xarxa neural simple

La neurona d'un perceptró, que té com a funció de transferència (FT) un limitador, es mostra en la Fig. 1. El codi MATLAB que permet calcular la sortida d'aquesta neurona es presenta també en la mateixa figura.

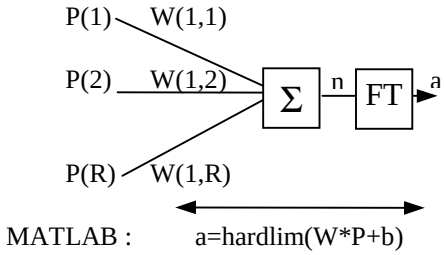


Fig. 1 Perceptró: model d'una neurona

Cada entrada, **j**, és ponderada amb el corresponent pes **W(1,j)**, i la suma de les entrades ponderades és enviada a la funció de transferència de tipus limitador. Aquesta funció de transferència, que és de tipus *step* i que en MATLAB l'anomena **hardlim**, retorna un "0", o bé, un "1" depenent de quin és el rang de l'entrada. La representació gràfica d'aquest comportament es pot veure en la Fig. 2. Es pot veure com el llindar d'activació es pot variar afegint una constant, **b**. Aquest paràmetre s'anomena llindar (*threshold*) o biaix (*bias*, **b**)[1].

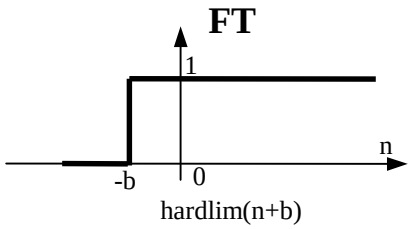
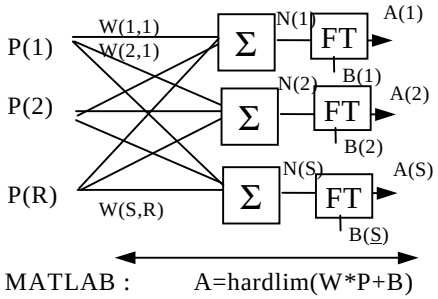


Fig. 2 Perceptró: funció de transferència d'una neurona

Una xarxa neural formada per neurones de tipus perceptró està formada per una sola capa amb S neurones tipus perceptró connectades a R entrades a través d'un conjunt de pesos W(i,j) tal com es mostra en la Fig. 3. Els índexs "i" i "j" indiquen que el pes W(i,j) es el pes de la connexió de la j-èsima entrada de la i-èsima neurona (per tant de la i-èsima component de la sortida de la xarxa).



P:(Rx1), W:(SxR), N:(Sx1), B:(Sx1), A:(Sx1)

Fig. 3 Xarxa neural formada per perceptrons (FT es hardlim, Fig.2)

[1] També el podeu trobar anomenat com a *offset*. En aquest document usarem el terme bias.

1.1 Inicialització

La funció **rands** s'utilitza per inicialitzar els pesos a valors aleatoris. Així per exemple, una xarxa neural formada per 8 neurones i 4 entrades s'inicialitzaria mitjançant

```
R=4; S=8;
[W,B]=rands(S,R);        % R i S son les dimensions del vector d'entrada i sortida resp.
```

La xarxa neural tindrà tantes sortides com neurones, i tantes entrades com nosaltres imposem.

1.2 Regla d'aprenentatge

Una xarxa neural formada per perceptrons és entrenada perquè respongui davant d'un vector d'entrada amb el corresponent vector objectiu de sortida, els elements del qual son un "0", o bé, un "1". La regla d'aprenentatge és aplicada a cada neurona per tal de determinar el nou conjunt de valors dels pesos després de la darrera presentanció de les entrades a la xarxa. Els canvis en els elements de la matriu de pesos es calculen mitjançant la funció de MATLAB **learnp**, que retorna els canvis sobre la matriu de pesos a partir del vector d'entrades P, el vector de sortides A (calculat a partir del vector de pesos abans d'actualitzar els pesos) i el vector de sortides objectiu T

```
[dW,dB]=learnp(P,A,T)
```

A continuació es presenta el codi MATLAB que presenta a un perceptró un vector entrades P i actualitza els pesos mitjançant la regla d'aprenentatge i el vector de sortides objectiu T

```
A=harlim(W*P,B);          % Obtenim el vector de sortides.
[dW,dB]=learnp(P,A,T);    % Obtinim els canvis al vector de pesos.
W=W+dW;                   % Obtenim el nou valor dels pesos.
B=B+dB;                   %Obtenim el nou valor de bias
```

Anem a comentar una mica més en detall com funciona la regla d'aprenentatge del perceptró:

"per tot i, ...

- *si A(i)=T(i), aleshores no s'aplica cap canvi*
- *si A(i)=0 i T(i)=1, aleshores P(j) s'afegeix a W(i,j), per tot "j" i s'afegeix un 1 al bias*
- *si A(i)=1 i T(i)=0, aleshores P(j) es resta de W(i,j), per tot "j" i es resta un 1 al bias*

on P és el vector d'entrades, A és el vector de sortides i T és el vector de sortides patró o objectiu".

Aquestes operacions es poden resumir en les següents equacions:

Per tot i i j ...

$$W(i,j)_{new} = W(i,j)_{old} + [T(i) - A(i)]P(j)$$

$$B(i)_{new} = B(i)_{old} + [T(i) - A(i)] \times 1$$

1.3 Entrenament

Els vectors patró són presentats a la xarxa neural un darrera de l'altre. Si la sortida de la xarxa és correcta no s'aplica cap canvi. En cas contrari, els pesos i els bias s'actualitzen utilitzant la regla d'aprenentatge del perceptró. Quan s'ha presentat el conjunt de tots els patrons diem que s'ha finalitzat una **època (epoch)**. Si després de presentar el conjunt de tots els patrons cap d'ells presenta error, l'entrenament

finalitza. En aquest moment, si es presenta qualsevol entrada patró a la xarxa, aquesta respondrà amb la sortida objectiu desitjada. Si un vector P_i que no està contingut en el conjunt d'entrenament es presenta a l'entrada de la xarxa aquesta intentarà generalitzar responent amb una sortida similar al vectors objectiu corresponents als vectors patró d'entrada que més s'assemblin al vector d'entrada P_i.

El codi MATLAB per entrenar un perceptró és el següent:

```
% Fase de Presentació
A=hardlim(W*P,B);
for epoch=1:max_epoch
        % Fase de Verificació
        if all(A==T)
                epoch=epoch-1;
                break;
        end
        % Fase d'Aprenentatge
        [dW,dB]=learnp(P,A,T);
        W=W+dW;
        B=B+dB;
        %Fase de Presentació
        A=hardlim(W*P,B);
end

% Fase de Presentació Final
A=hardlim(W*P,B)
if all(A==T)
        disp("Xarxa Neural entrenada correctament");
end
```

MATLAB disposa d'una funció que realitza tot el procés d'entrenament presentat amb el codi anterior d'una sola vegada. Aquesta funció és la funció **trainp**.

```
[W,B,TE,TR] = trainp(W,B,P,T,ME)
```

On :

W - SxR Matriu de pesos
B - Sx1 vector de bias
P - RxQ Matriu dels vectors d'entrada, on Q és el nombre d'exemples.
P(1,1) és l'entrada 1 de l'exemple 1,
P(2,1) és l'entrada 2 de l'exemple 1, ...
T - SxQ matriu de vectors objectiu.
TP – Paràmetres d'entrenament (opcional).

Retorna:

W - Nova matriu de pesos.
B - Nou vector bias.
TE - Èpoques entrenades.
TR – errors intermedis durant l'entrenament.

1.4 Exercicis de comprensió

Executar la comanda **demo** des de la finestra de comandes de MATLAB. Seleccionar Toolboxes de la finestra esquerra, després Neural Network i realitzar els exercicis següents:

1) Simple neuron & transfer function
La figura que apareix mostra un perceptró d'una sola entrada. Es tracta que proveu les diferents funcions d'activació possibles: **hardlim**, **purelin**, **logis** i **tansig**.
Varieu els valors dels pesos i el bias.

2) Neuron with vector input
En aquest cas disposem de diverses entrades.
Com has d'ajustar el pes perquè reconegui les entrades?

3) Perceptron learning rule
Defineix dos problemes:
- Un que sigui resoluble per la xarxa anterior.
- Un altre que no sigui resoluble.

4) Classificador de taronges i pomes.
Finalitza la demo i executa la comanda **nnd3pc** i comenta quina és el paper de la xarxa neuronal en l'aplicació visualitzada.

1.5 Exercici d'aplicació: implementació de la funció lògica OR

Fent ús de tota la informació presentada fins el moment es demana entrenar una xarxa neural formada per una sola neurona de tipus perceptró per tal de que aprengui el funcionament de la funció lògica OR.

	Funció OR	**Matlab**
Entrades :		
	P(1, :) =[0 0 1 1]	**matriu P = [0 0 1 1 ; 0 1 0 1] ;**
	P(2, :) =[0 1 0 1]	
Sortides desitjades		
	T(1, :) =[0 1 1 1]	**matriu T = [0 1 1 1] ;**

2. Backpropagation: un mecanisme d'aprenentatge eficient

Una neurona d'una xarxa de backpropagation amb R entrades es pot representar simbòlicament mitjançant la Fig. 4. Cada entrada és ponderada amb un pes W. La suma de les entrades ponderades, junt amb un bias, constitueixen l'entrada de la funció de transferència F. Les neurones d'una xarxa de backpropagation poden utilitzar qualsevol funció F diferenciable per a generar la sortida de la xarxa

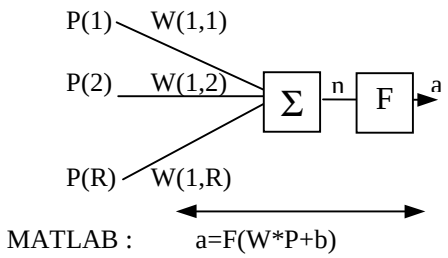


Fig. 4 Backpropagation: model d'una neurona (R entrades i 1 sortida)

Les xarxes de backpropagation utilitzen sovint la funció de transferència, F, log-sigmoidal que en MATLAB s'implementa mitjançant la funció **logsig**. La representació gràfica d'aquesta funció de transferència es presenta en la Fig. 5. La funció **logsig** genera sortides continues entre 0 i 1 al variar

l'entrada de la neurona des de valors positius a valors negatius. Aquesta funció de transferència es pot utilitzar amb bias, o bé, sense. Els bias ofereixen a la xarxa un grau de llibertat més alhora d'aprendre el comportament d'una determina funció. La seva utilització augmenta les possibilitats de què la xarxa trobi una solució acceptable, mentre que a la vegada redueixen el nombre d'èpoques d'aprenentatge necessàries.

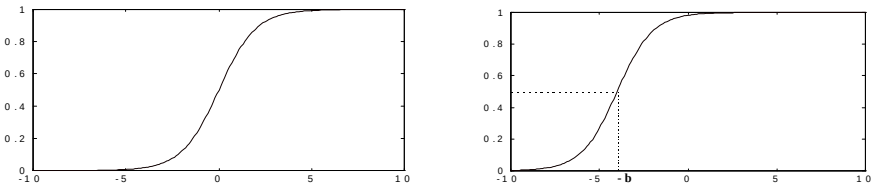


Fig. 5 Backpropagation: funció de transferència **logsig (n)** i **logsig(n+b)** .

De forma alternativa, les xarxes de backpropagation poden utilitzar la funció de transferència (**F**) tan-sigmoidal implementada en MATLAB amb la funció **tansig**. Aquesta funció es presenta en la Fig. 6.

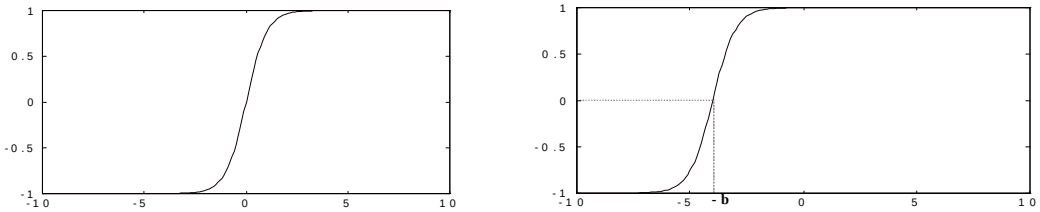


Fig. 6 Backpropagation: funció de transferència **tansig(n)** i **tansig(n+b)**.

De vegades, també es pot utilitzar la funció de transferència lineal que el MATLAB implementa amb la funció **purelin**. Aquesta funció es presenta en la Fig. 7.

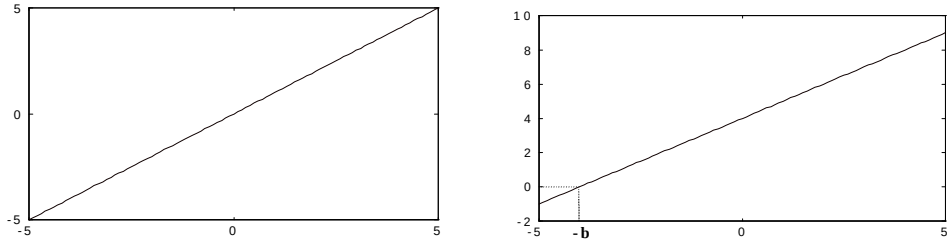


Fig. 7 Backpropagation: funció de transferència **purelin(n) i purelin(n+b)**

En la Fig. 8 es presenta una xarxa neural amb S neurones log-sigmoidals amb R entrades. Les xarxes de backpropagation sovint tenen una o més capes ocultes de neurones sigmoidals seguides per una capa de sortida de neurones lineals. Múltiples capes de neurones amb funcions de transferència no lineals permeten a la xarxa l'aprenentatge de relaciones lineals i no lineals entre l'entrada i la sortida.

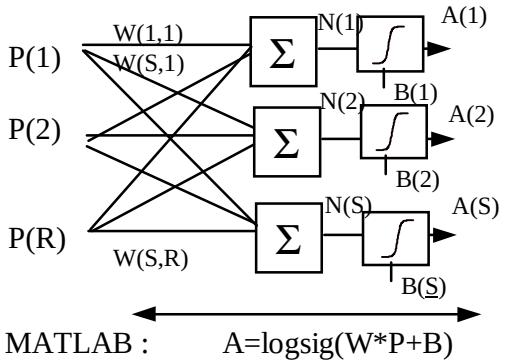


Fig. 8 Backpropagation: estructura d'una xarxa neural d'una capa

En el cas d'utilitzar xarxes amb múltiples capes, afegirem el numero de la capa als noms de les matrius i vectors associats amb la capa. Així, la matriu de pesos i el vector de sortides de la capa 2 d'una xarxa s'anomenaran W2 i A2 respectivament. En la Fig. 9 es presenta una xarxa de backpropagation amb dues capes una tan-sigmoidal i una altra de lineal que fan servir aquesta notació.

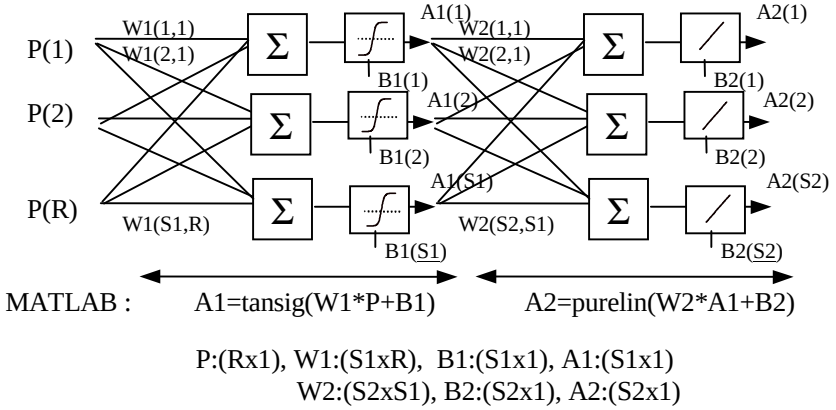


Fig. 9 Backpropagation: estructura d'una xarxa neural de dues capes

2.1 Inicialització

La inicialització de la matriu de pesos d'una xarxa de backpropagation es fa amb valors aleatoris entre -1 i 1. Aquesta inicialització aleatòria es pot fer mitjançant la funció de MATLAB **rands**. Així, per exemple, per una xarxa neural de backpropagation de dues capes, amb 5 entrades, 4 neurones a la capa oculta i 3 neurones a la capa de sortida, que sera entrenada amb els patrons P (entrades) per obtenir les sortides T, el codi d'inicialització és el següent. Les matrius d'entrada i de sortides desitjades (P i T) estan formades per:

P : Matriu amb Q vectors columna de 5 components. (R=5)
T : Matriu amb Q vectors columna de 3 components (S2=3)

on Q son el número de vectors que s'utilizaran en l'entrenament i per tant el mateix número de vectors de sortida (objectiu) per comparar amb les sortides de la xarxa.
Així el codi Matlab serà :

```
[R,Q]=size(P);
S1=4;                % 4 neurones capa oculta
[S2,Q]=size(T);      %
```

```
[W1,B1]=rands(S1,R);   % Pesos i bias capa oculta
[W2,B2]=rands(S2,S1);  % pesos i bias capa de sortida
```

Cal observar que el nombre d'entrades de la xarxa i el nombre de neurones en la capa de sortida ve fixat pels vectors entrada/objectiu que la xarxa ha d'aprendre. El nombre de neurones en les capes ocultes és decisió del dissenyador de la xarxa. Quantes més neurones, més possibilitat de què la xarxa assoleixi una solució. En canvi, quantes més neurones més temps d'aprenentatge serà necessari i més grans seran les matrius de pesos (més complexitat de càlcul).

2.2 Regla d'aprenentatge

La regla d'aprenentatge donada per l'algorisme backpropagation permet l'entrenament de xarxes no lineals multicapa per a aplicacions de classificació i associació de patrons. La restricció d'aquest algorisme és que les funcions de transferència (o activació) han de ser funcions derivables, ja que l'ajust dels pesos es fa seguint una algorisme de minimització de l'error quadràtic mig (LMS, Least mean squared) mitjançant la disminució del gradient ('gradient descent') a cada iteració efectuada per l'ajust dels pesos. Per això és necessari calcular les derivades de l'error (vector delta). I l'error es defineix com la diferència entre les sortides teòriques i les reals).

Les derivades de l'error calculades a la sortida de la última capa (ja que és la única que podem comparar amb els patrons desitjats), es propaguen cap a les capes internes (ocultes). A Matlab el càlcul dels delta vectors (D) es fa amb les funcions **deltalin(A,E)**, **deltalog(A,E)** i **deltatan(A,E)** en el cas de la capa oculta i **deltalin(A,D,W), deltalog(A,D,W), deltatan(A,D,W)** pel cas de capes ocultes. Els paràmetres corresponen a : A : vector de sortides, E : vector error, D i W : són els delta vectors i els pesos de la capa anterior. S'utiliza una funció o altra d'acord amb la funció de transferècnia desitjada en cada capa.

A mesura que es va propagant l'error cap a les capes ocultes, l'actualització dels pesos (i bias) es fa segons aquests valors seguint amb els increments calculats d'aquesta forma:

$\Delta W(i,j) = lr\ D(i)P(i)$ — lr : velocitat d'aprenentatge.
$\Delta B(i) = lr\ D(i)$ — D(i) : vector delta propagat.
P(i) : Entrades de la capa en qüestió.

La funció de Matlab **[dW,dB]=learnbp(P,D,lr)** es pot utilitzar amb aquest propòsit. Només cal tenir en compte que P i D s'han de presentar com matrius, amb els vectors columnes corresponent a tots els patrons que es volen entrenar.

2.3 Entrenament

La forma d'entrenar una xarxa és la següent : Es presenten els vectors d'entrada i a partir de les sortides obtingudes i les desitjades es calcula un error. Es calcula la mitja dels errors quadràtics per tots els patrons i si es inferior a un llindar prefixat, l'entrenament s'atura. En cas contrari es calcula el vector delta a la sortida de la xarxa i es propaga enrera (backpropagation) cap ales capes ocultes. A partir dels vectors delta de cada capa s'actualitzen els pesos d'acord amb la regla d'aprenentatge anterior. I es repeteix el procés fins que l'error sigui inferior a un llindar predeterminat.

Aquest procediment es pot fer de forma còmoda a Matlab utilitzant la funció **trainbp** que permet l'entrenament de xarxes de varies capes. En aquest cas cal els passos a seguir son :

a) Crear una matriu **P** amb els vectors d'entrada com a columnes.
b) Crear la matriu de sortides desitjades **T**, amb igual número de vectors que **P**.
c) Crear un vector de paràmetres : **TP =[disp_freq max_epoch err_goal lr] ;**
   disp_freq : Visualització de resultats intermedis.

max_epoch : Numero màxim d'iteracions en l'entrenament.
err_goal : llindar per l'erro mínim a assolir.
lr : velocitat d'aprenentatge

Per cada capa caldrà a més a més :

- Inicialitzar (aleatòriament) les matrius W i B de pesos i bias.
- Especificar el tipus de xarxa (funció de transferència) en un string. Fes help per veure els tipus.

Amb tot això podem fer crides a la funció anterior com la següent, pel cas d'una xarxa de dues capes :

[W1,B1,W2,B2,epochs, TR]=trainbp(W1,B1,'F1',W2,B2,'F2',P,T,TP) ;

Wi – Matriu de SixR de pesos per la capa i-essima.
Bi – Vector d' S1x1 d'ofset o bias per la capa i-essima.
Fi – Funció de transferència (cadena) per la capa i-essima.
P - Matriu de RxQ de vectors d'entrada.
T - Matriu de SxQ de vectors objectiu.
TP – Paràmetres d'entrenament (opcional).

Retorna els nous pesos i ofsets o bias i
Wi – Nous pesos.
Bi – Nous bias.
TE – L'actual nombre d'èpoques entrenades
TR – Errors intermijos: [fila d'errors]

2.4 Exercicis de comprensió

1) Backpropagation.
Finalitza la demo i executa la comanda **nnd11bc**.
Entrena la xarxa almenys dues vegades. Digues quines són els exemples, pesos i bias inicials, i digues quins són els que resulten després de 3 iteracions de l'algorisme backpropagation. Digues quin és l'error en cada iteració.

2) Adaptative noise cancellation
Torna a la finestra de demos (executa la commanda **demo**, etc...) i selecciona la demostració amb el títol mencionat.
Què és el que fa la xarxa? Quins resultats s'obtenen?

3) Adaptative noise cancellation in an Airplain
Es tracta d'un exercici similar a l'anterior. Ara, però, podem canviar el paràmetre d'aprenentatge α.
Quin és l'efecte de posar un α molt gran? I de posar-lo molt petit?
Per acabar d'entendre com afecta α, tanca la finestra i executa la commanda **nnd9sdq**.

4) Generalization
Torna a la finestra de demo i selecciona l'anomenada generalization.
Quin és l'efecte d'augmentar el nombre d'unitats ocultes? Com ho interpretes?

2.5. Exercici d'aplicació: reconeixement de caràcters

Es desitja dissenyar i entrenar una xarxa neural que sigui capaç de reconèixer les 26 lletres de l'alfabet. Es suposa que es disposa d'un sistema de visió artificial capaç de digitalitzar cada lletra centrada en el camp de visió del sistema de reconeixement. El resultat d'aquesta digitalització es representa mitjançant una graella de 5 x 7 pixels booleans. El problema però apareix pel fet de què el sistema de visió artificial no és perfecte i junt amb la lletra digitalitzada s'hi afegeix soroll. En aquest exercici es demana crear un sistema capaç de classificar les 5x7 entrades de forma que doni a la sortida la lletra corresponent.

Passos a seguir:

1) Definir el problema
2) Definir l'estructura de la xarxa *feed forward*. Es recomana una estructura de xarxa de 2 nivells, amb 10 neurones a la capa oculta.
3) Inicialitzar la xarxa
4) Entrenar la xarxa amb exemples sense soroll
5) Un cop entrenada la xarxa, prova el seu funcionament comprovant que l'error de la sortida és 0 pel conjunt d'aprenentatge.
6) Testejar la xarxa per valors pel que no ha estat entrenada

Proporcionar els resultats en forma de gràfics.

Dades:

1. En el fitxer prprob.m disposeu de la definició de l'alfabet expressat en matrius de 5x7 pixels, de forma que podeu inicialitzar dues matrius, alfabet i patrons, corresponents a exemples de lletres d'alfabet sense sorolls a través de la instrucció:

[alfabet,patrons] = prprob;

Cada patró es representa per un vector de 26 valors, on el valor activat indica la lletra a la qual correspon (per exemple, per la lletra B, tot seran 0's excepte el segon valor). Per tant, **patrons** és una matriu de 26x26 (un vector patró per a cada lletra).

2. Per la implementació pots utilitzar les funcions següents de la ToolBox de Xarxes neurals de Matlab :

initff - Per la inicialització de la xarxa. Per exemple, per una xarxa amb 5 neurones ocultes:

```
S1 = 5;
[W1,B1,W2,B2] = initff(P,S1,'logsig',T,'logsig');
```

on: W1 i W2 són les matrius de pesos de la primera i segona capa respectivament;
B1 i B2 són els bias per la primera i segona capa respectivament;
P és la matrius d'exemples; logsig és la funció d'activació a emprar en les dues capes de
la xarxa; i
T és la matriu de patrons.

trainbp - Per l'entrenament de la xarxa d'acord amb l'algorisme Backpropagation. Exemple:

```
[W1,B1,W2,B2,ep,tr] = trainbp(W1,B1, 'logsig',W2,B2, 'logsig',P,T,tp);
```

On tp són el conjunt de paràmetres opcionals

```
tp = [df me eg lr];
```

I cada paràmetre pot ser, per exemple:

```
df = 20;        % Freqüència de display de resultats intermitjos
me = 5000;      % Màxim nombre d'èpoques a entrenar.
eg = 0.02;      % Error quadrat global.
lr = 0.01;      % Factor d'aprenentatge.
```

El resultat retornat són els pesos finals de cada capa i l'evolució de l'error a cada època (a **tr**).

simuff - Per simular el comportament de la xarxa. Exemple, per una xarxa de 2 nivells amb funcions d'activació log-sigmoidals:

```
A2 = SIMUFF(P,W1,B1,'logsig',W2,B2,'logisg')
```

On:

P - Matriu d'entrada
Wi – Pesos de la matriu en la capa i.
Bi – Vector de bias de la capa i
logsig – funcions de tansferència de les capes.

Retorna la sortida del darrer nivell.

3. Altres funcions de MATLAB:

randn - Generació de nombres aleatòris amb distribució normal. Exemple:

```
randn(35,26)
```

Genera una matriu de 35*26 de nombres aleatòris.

sum - Funció suma que, aplicada sobre una matriu, proporciona un vector corresponent a la suma de cada columna.

4. Per testejar la xarxa recordeu que la fórmula de l'error quadràtic mitjà és:

$$\frac{1}{m}\sum_{i=1}^{m} e_i$$

on *m* és el nombre d'exemples, i e_i és l'error per a cada exemple i es calcula com:

$$e_i = \frac{1}{2}\sum_{j=1}^{n} (A_j - T_j)^2$$

on n és el nombre d'unitats en la capa de sortida.

2.6 Ampliació a l'exercici d'aplicació (addicional)

Entrenar la xarxa amb exemples amb soroll i veure com millora el resultat per un input no esperat.