

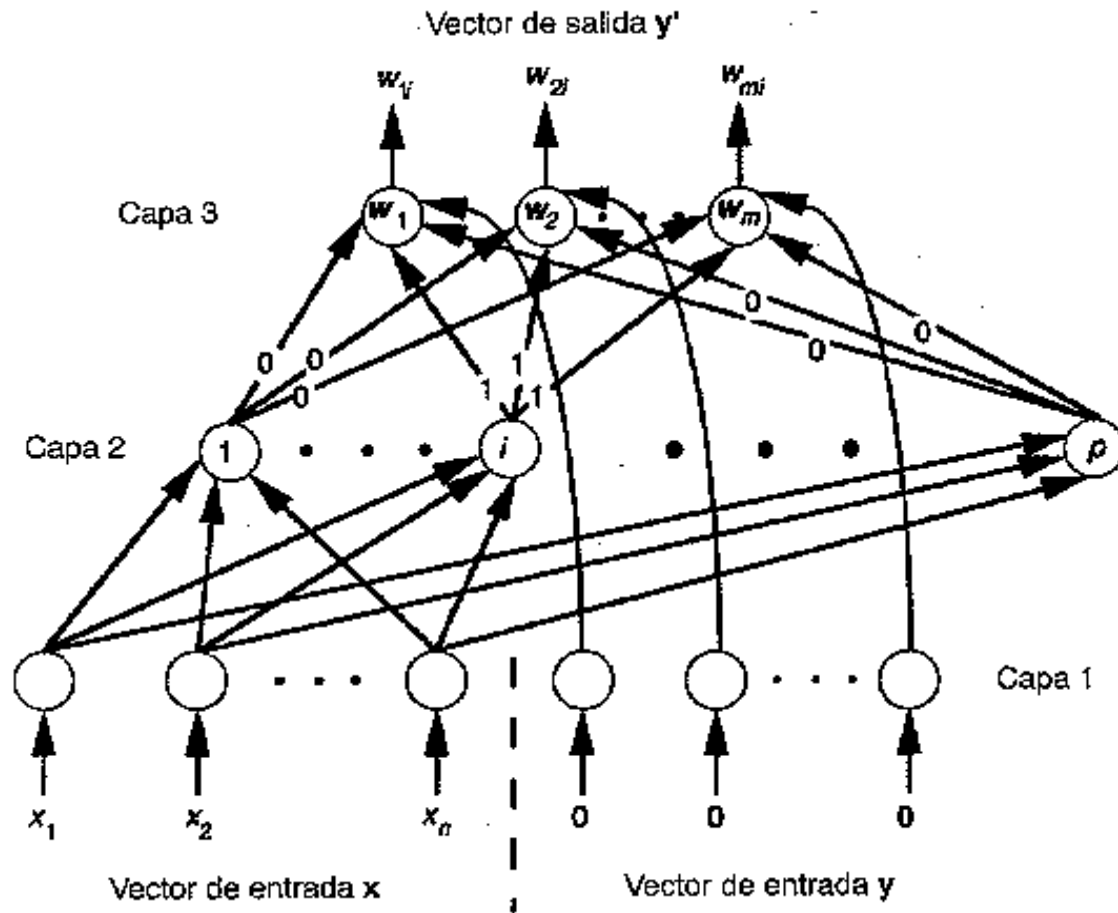


# Counterpropagation Network (CPN)

---

- Dado un conjunto de pares de vectores  $(x_1, y_1), \dots, (x_L, y_L)$ , la CPN puede aprender a asociar un vector  $x$  en la capa de entrada con un vector  $y$  en la capa de salida.
- Se comporta como una función de correspondencia  $y = \phi(x)$
- Si existe la inversa de  $\phi$ , de tal manera que  $x$  sea una función de  $y$ , entonces la CPN aprenderá la correspondencia inversa,  
$$x = \phi^{-1}(y).$$

# Arquitectura de la CPN





# CPN

---

## ■ Características

- Está formada por: la capa de entrada, una capa competitiva y la capa de salida.
- No utiliza un único algoritmo de aprendizaje a lo largo de toda la red, sino que es distinto en cada capa.

## ■ Ventajas

- Se entrena rápidamente.
- Resulta útil para realizar prototipado rápido.

## ■ Desventaja:

- No siempre converge con la precisión esperada.



# CPN

- El algoritmo tradicional sugiere trabajar con vectores normalizados para identificar “similitudes”.
- Veremos como funciona este enfoque y las desventajas de la normalización.
- Luego **reemplazaremos la normalización** por alguna **medida de distancia**.

# Bloques básicos de la CPN

---

## ■ Capa de Entrada

- El algoritmo original indica que los datos de entrada deben ser escalados o normalizados para adaptarse a los cálculos

$$I = \frac{x}{||x||} = \frac{x}{\sqrt{\sum_j x_j^2}}$$

# Bloques básicos de la CPN

---

## ■ Capa Oculta

Para cada elemento la entrada se calcula como:

$$Neta = I * W$$

$$Neta = \| I \| \| W \| \cos(\theta)$$

$$Neta = \cos(\theta)$$

(tanto  $I$  como  $w$  están normalizados)



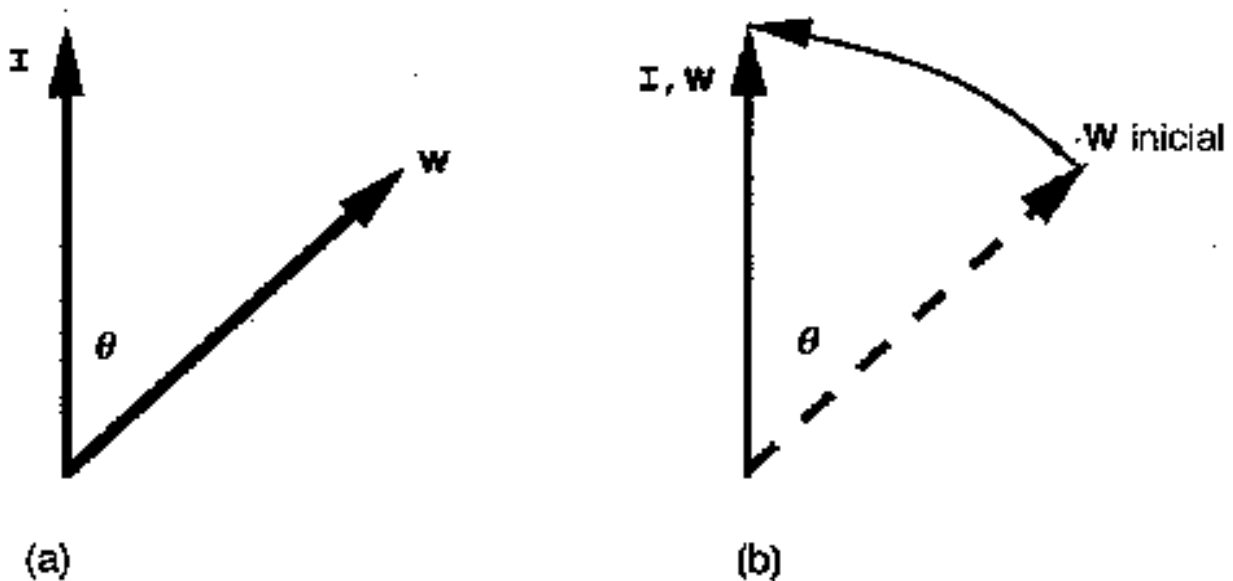
# Bloques básicos de la CPN

---

## ■ Capa oculta (competitiva)

- Cada neurona responde con su valor de entrada neta.
- Esta capa se encarga de reconocer a la neurona con mayor entrada neta como la que representa el espacio al que pertenece el vector de entrada.

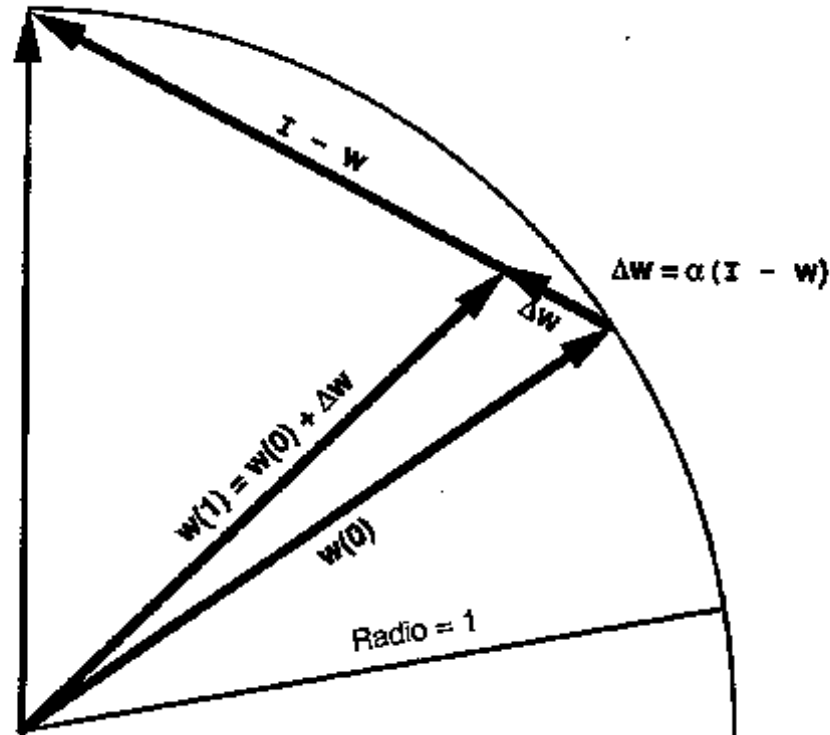
# Aprendizaje (entrada-oculta)



La regla de aprendizaje aplicada a los pesos que relacionan la capa de entrada (con sus valores normalizados) con la capa oculta (a), pretende ir aproximando el vector  $w$  al vector  $I$  (b).



# Aprendizaje (entrada-oculta)



- Los pesos se actualizan de la siguiente forma:

$$w(t+1) = w(t) + \alpha (I - w(t))$$

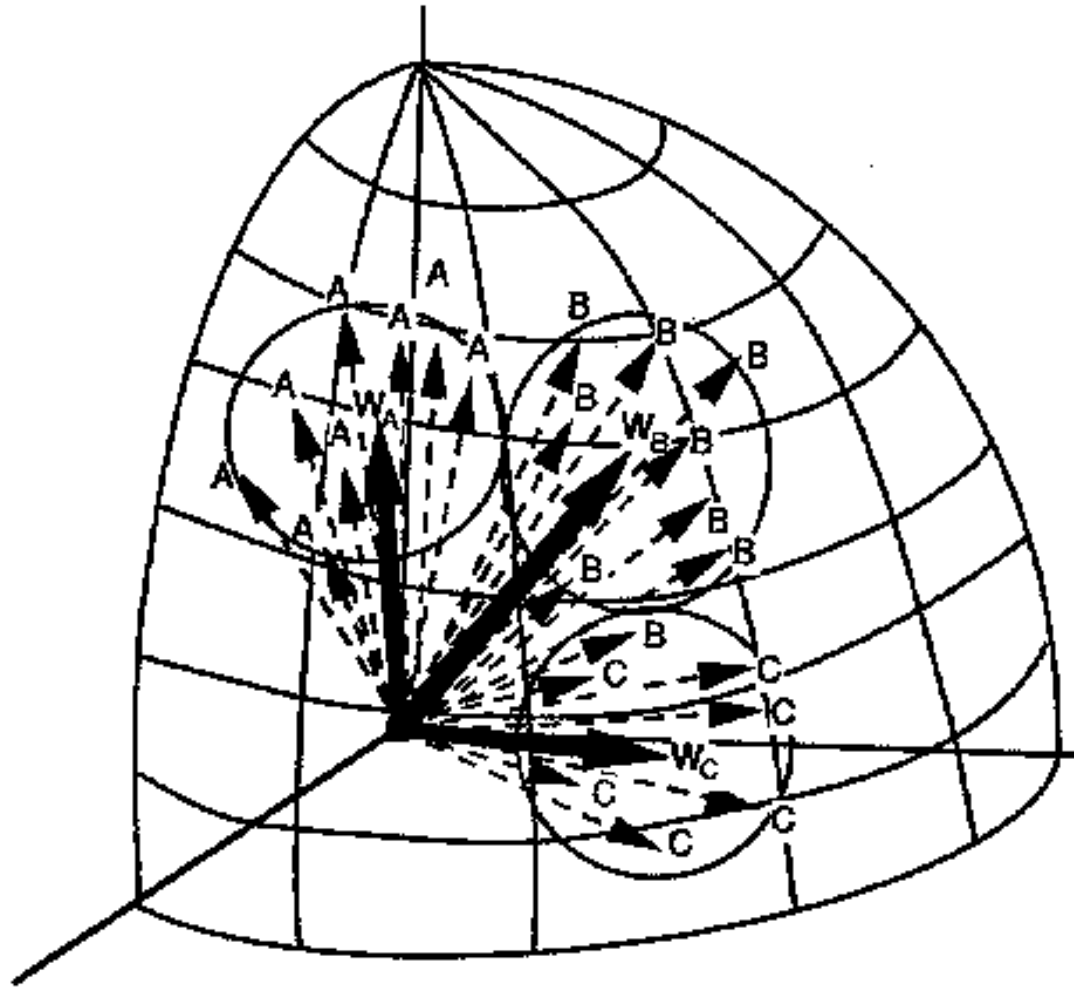


# Aprendizaje (entrada-oculta)

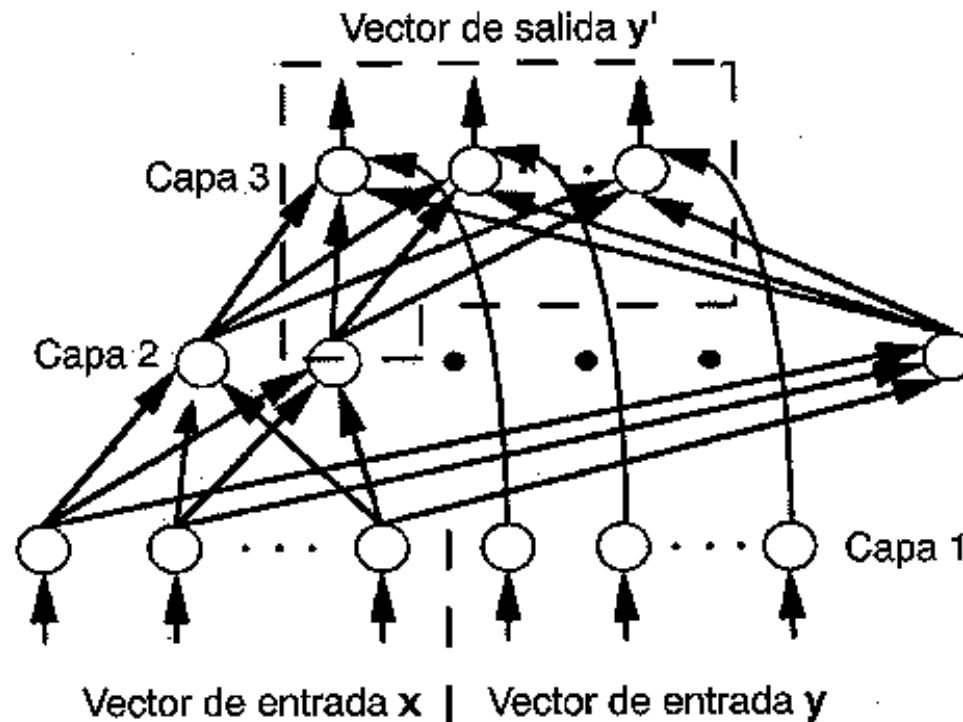
---

- 1) Seleccionar un vector de entrada aleatorio.
- 2) Normalizarlo e ingresarlo a la red.
- 3) Calcular la unidad ganadora.
- 4) Actualizar el vector de pesos utilizando
$$w(t+1) = w(t) + \alpha (x - w)$$
sólo para la ganadora.
- 5) Repetir los pasos 1 a 4 hasta que todos los vectores hayan sido seleccionados por lo menos 1 vez.
- 6) Repetir 5) hasta que todos hayan sido clasificados correctamente.

# Aprendizaje (entrada-oculta)



# Aprendizaje (oculta-salida)



Los pesos se actualizan de la siguiente forma:

$$w_i(t+1) = w_i(t) + \beta(y_i - y') = w_i(t) + \beta(y_i - w_i(t))$$



# Aprendizaje (oculta-salida)

---

- 1) Aplicar el vector de entrada,  $x$  normalizado y su correspondiente vector de salida  $y$ .
- 2) Determinar la unidad ganadora de la capa competitiva.
- 3) Se actualizan los pesos de las conexiones que van de la unidad competitiva ganadora a las unidades de salida según:

$$w(t+1) = w(t) + \beta(y - w(t))$$

- 4) Se repite de 1 a 3 hasta que todos los vectores de todas las clases se correspondan con salidas satisfactorias.

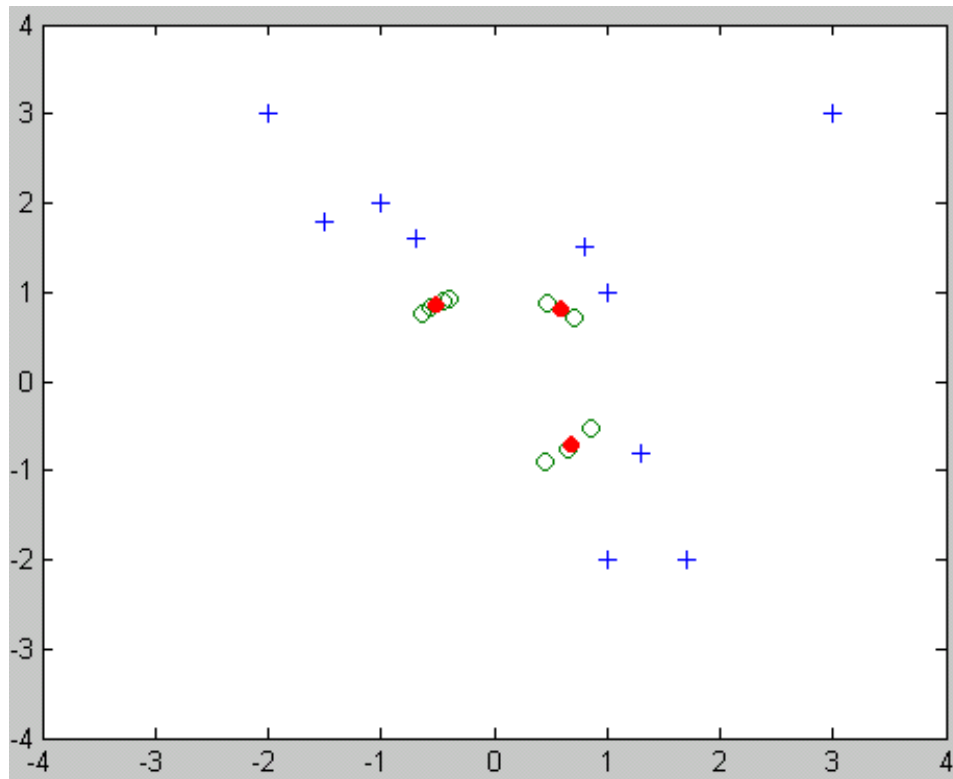
# Ejercicio

---

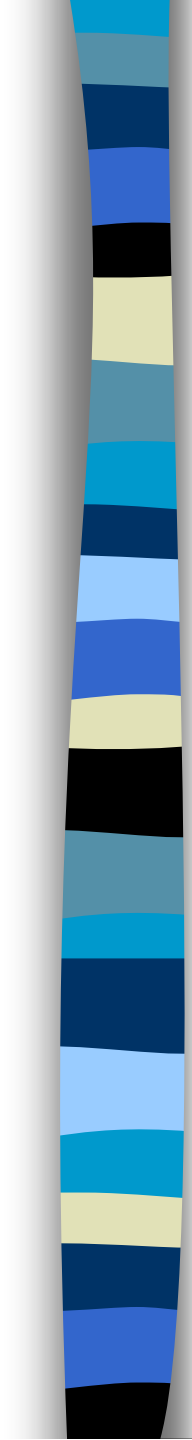
Vector X		Vector Y	
1	1	1	1
-1	2	-0.5	0.5
1	-2	1.5	-1
3	3	2	2
0.8	1.5	1	1
-1.5	1.8	-0.5	0.5
-0.7	1.6	-0.5	0.4
-2	3	-0.8	1.5
1.3	-0.8	1.5	-1
1.7	-2	1.5	-1

# Entrenamiento de los pesos que van desde la capa de entrada a la capa oculta.

---



$P = \begin{bmatrix} 1 & -1 & 1 & 3 & 0.8 & -1.5 & -0.7 & -2 & 1.3 & 1.7; \\ 1 & 2 & -2 & 3 & 1.5 & 1.8 & 1.6 & 3 & -0.8 & -2 \end{bmatrix};$



```
P = [1 -1 1 3 0.8 -1.5 -0.7 -2 1.3 1.7;  
1 2 -2 3 1.5 1.8 1.6 3 -0.8 -2];
```

```
Y = [ 1 -0.5 1.5 2 1 -0.5 -0.5 -0.8 1.5 1.5  
1 0.5 -1 2 1 0.5 0.4 1.5 -1 -1 ];
```

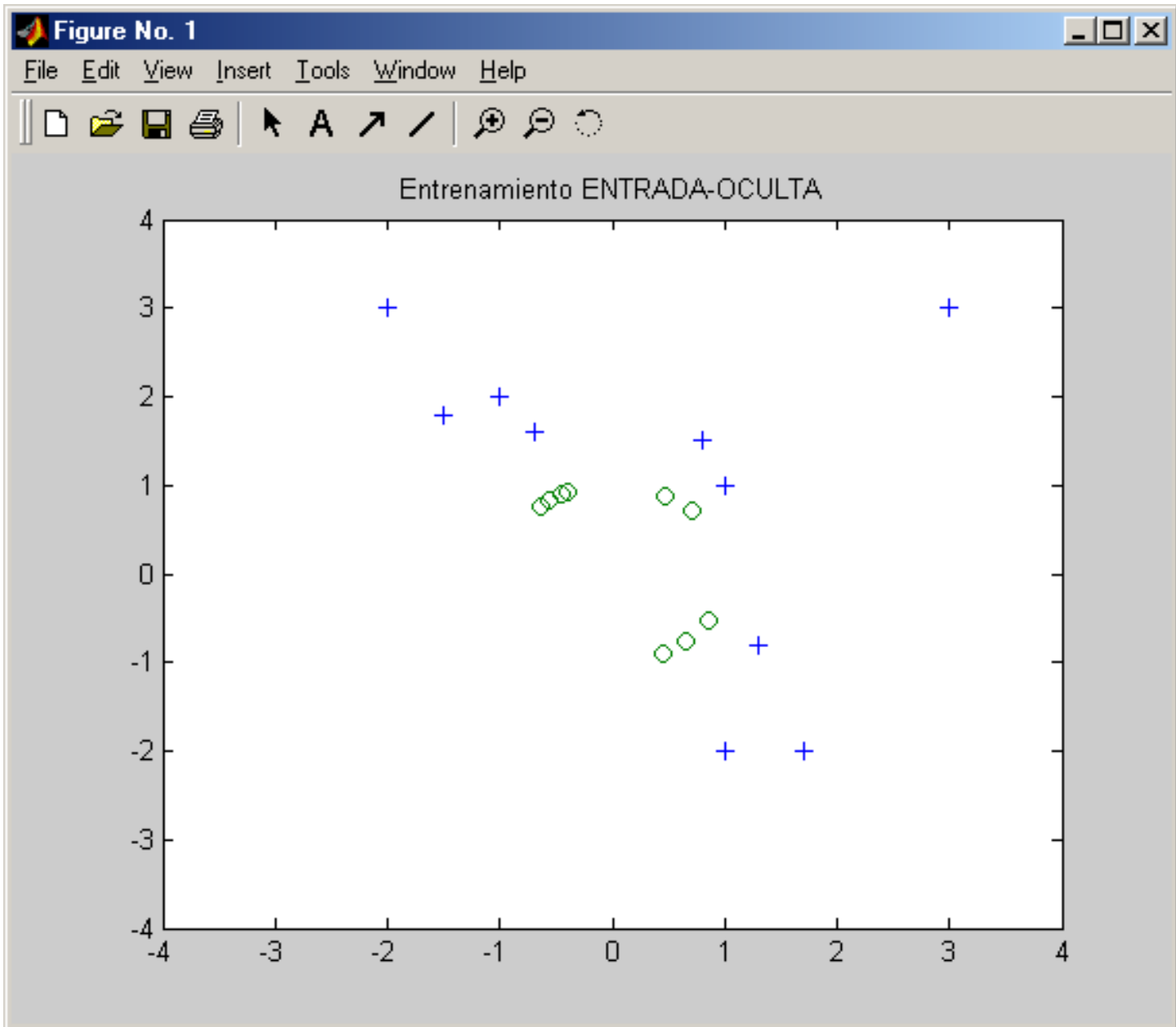
```
[entradas, CantPatrones] = size(P);  
salidas = size(Y,1);
```

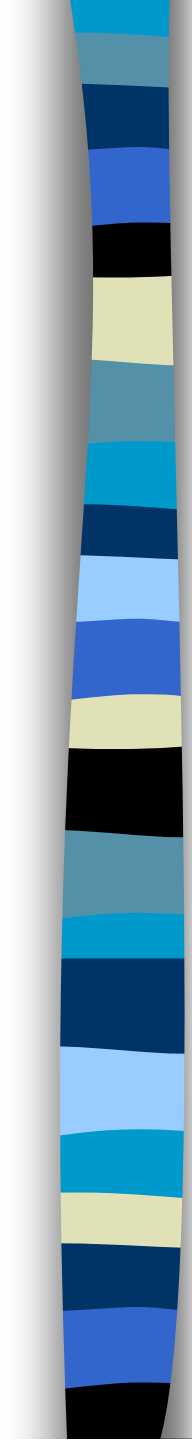
```
% normalizando los vectores de entrada  
P_norm = normalizar(P);
```

```
figure(1)
```

```
plot(P(1,:), P(2,:), '+', P_norm(1,:), P_norm(2,:), 'o')  
axis([-4 4 -4 4])
```



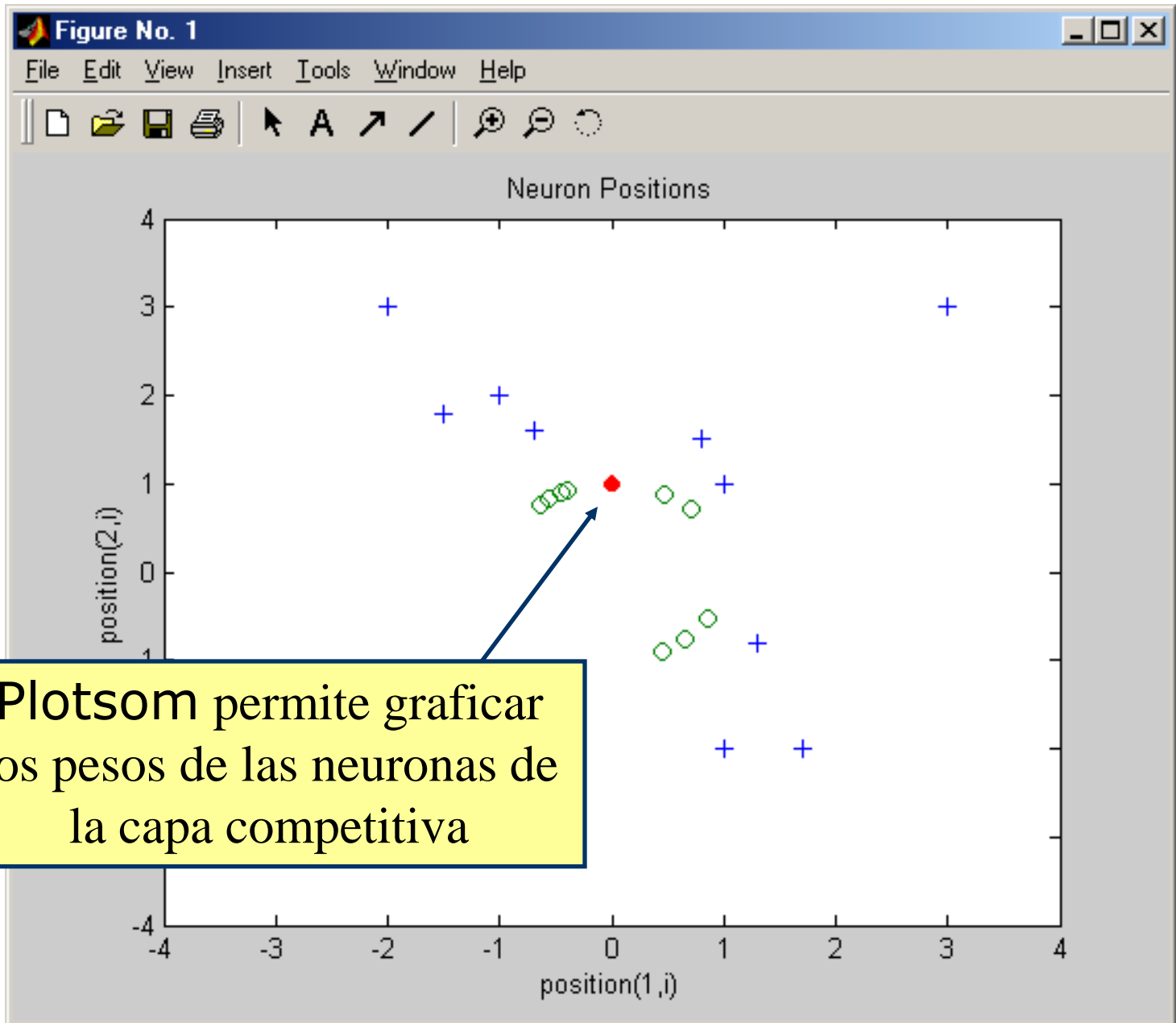




```
W = [ 0  0  0;  
      0.5 0.5 0.5];
```

```
ocultas = length(W);  
W_norm = normalizar(W);
```

```
hold on  
plotsom(W_norm)  
pause(0.2)
```





```
ITE_MAX = 10;
```

```
ite = 0;
```

```
alfa = 0.25;
```

```
while ( ite <= ITE_MAX ) & ("los pesos no cambien mucho"),
```

```
for i=1:CantPatrones,
```

```
    %buscar el W mas proximo
```

```
    [distancia,mayor] = max(P_norm(:,i)' * W_norm);
```

```
    %Actualizar la neurona mas proxima
```

```
    W_norm(:, mayor) = W_norm(:,mayor) +
```

```
        alfa * (P_norm(:, i) - W_norm(:,mayor));
```

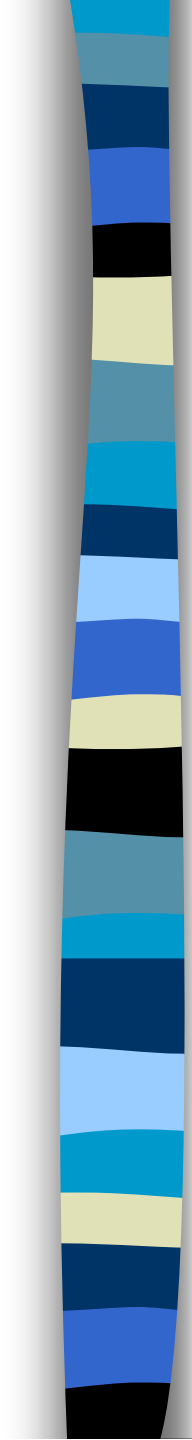
```
    W_norm = normalizar(W_norm);
```

```
end
```

```
    % redibujar
```

```
    ite = ite + 1
```

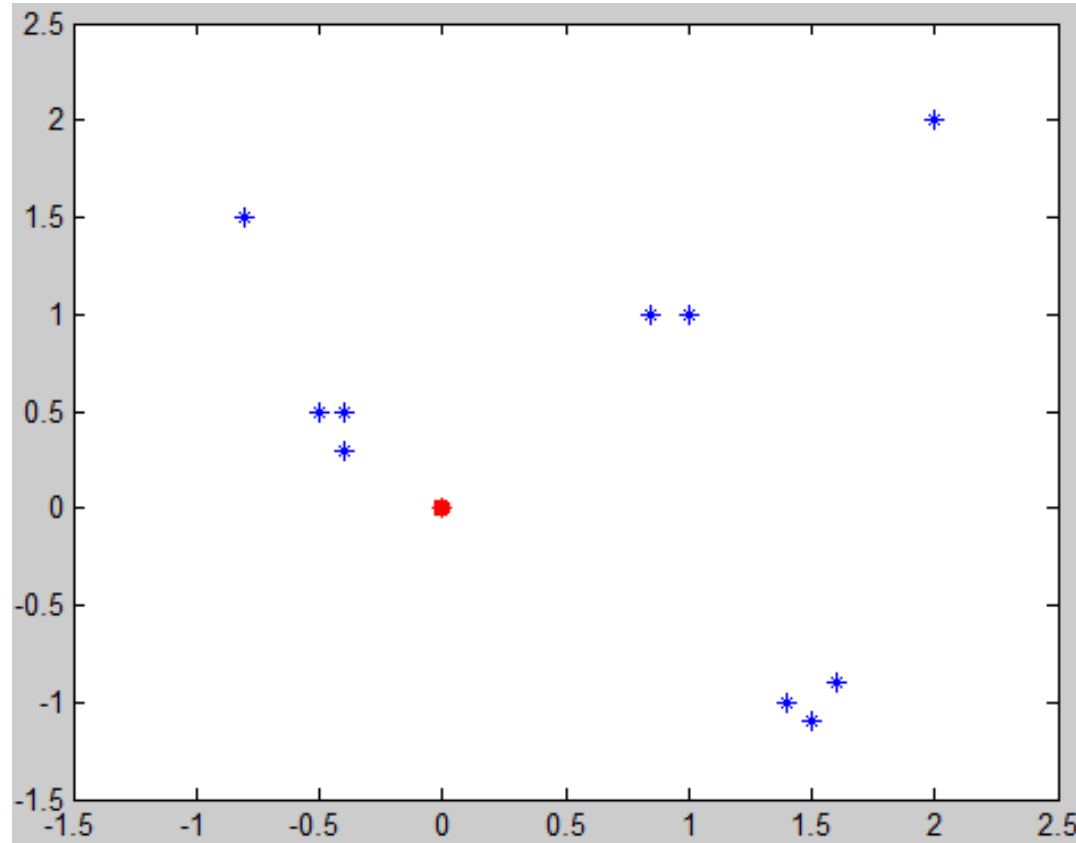
```
end
```



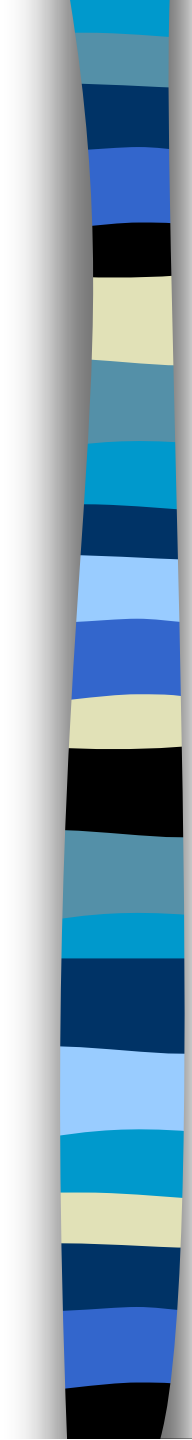
```
% redibujar
hold off
plot(P(1,:), P(2,:), '+', P_norm(1,:),
      P_norm(2,:), 'o', Y(1,:), Y(2,:), 'x')

hold on
plotsom(W_norm)
hold on
axis([-4 4 -4 4])
pause(0.2)
```

# Entrenamiento de los pesos que van desde la capa oculta a la de salida.



$$Y = \begin{bmatrix} 1 & -0.5 & 1.5 & 2 & 1 & -0.5 & -0.5 & -0.8 & 1.5 & 1.5 \\ 1 & 0.5 & -1 & 2 & 1 & 0.5 & 0.4 & 1.5 & -1 & -1 \end{bmatrix};$$



```
beta = 0.25;
W2 = zeros( salidas, ocultas);

ITE_MAX = 50;
while ( ite <= ITE_MAX ) & ( “los pesos no cambien”),

    for i=1:CantPatrones,
        %buscar el W mas proximo
        [distancia,mayor] = max(P_norm(:,i)' * W_norm);

        %Actualizar los pesos que salen de la neurona
        % ganadora
        W2( :, mayor) = W2(:, mayor) +
            beta * (Y(:, i) - W2(:, mayor));
    end
    %redibujar
    ite = ite + 1
end
```



```
%Ganadoras por patron
```

```
[distancia, ganadora] = max((P_norm' * W_norm)')
```

```
W2          %Salida para cada neurona oculta
```

```
W3 = [];
```

```
for i=1:ocultas,
```

```
    suma = [0;0];
```

```
    cant = 0;
```

```
    for j=1:CantPatrones,
```

```
        if ganadora(j)==i
```

```
            suma = suma + Y(:,j);
```

```
            cant = cant + 1;
```

```
        end
```

```
    end
```

```
    W3(:,i) = suma / cant;
```

```
end
```

```
W3
```

Promedia los  
valores de  
salida de los  
patrones que  
pertenecen a la  
misma neurona  
ganadora

Comparar W2 y W3





## Después de entrenar los pesos e/ la capa oculta y la de salida

*%Salida para cada neurona oculta*

W2 =

1.4800	-0.6078	1.3066
-0.9866	0.8356	1.3066

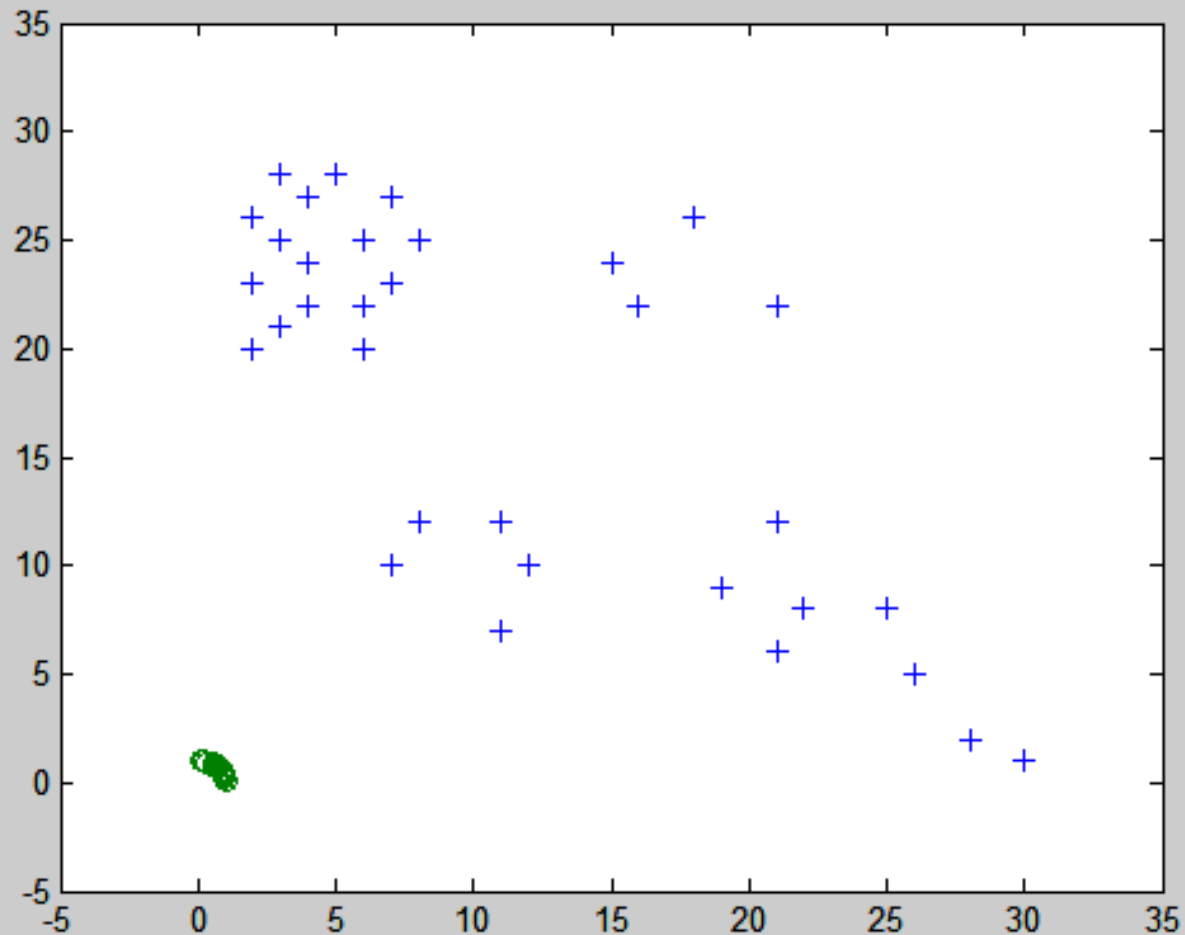
*%W3 no pertenece a la RN es sólo para comparar*

W3 =

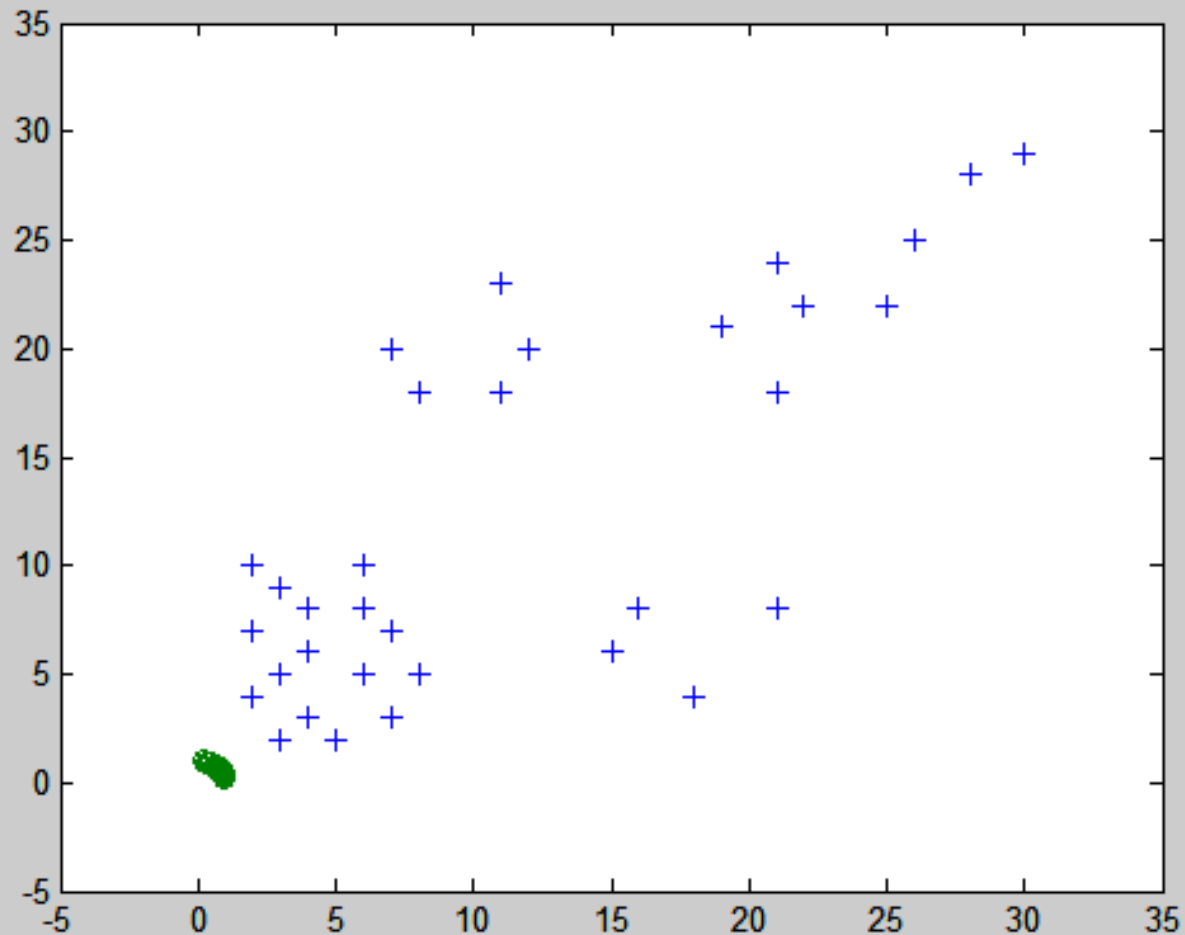
1.5000	-0.5750	1.3333
-1.0000	0.7250	1.3333

- Note que la salida obtenida por el entrenamiento iterativo se corresponde con el promedio de los valores de salida de los patrones de cada grupo. <sub>25</sub>

# Normalización de la entrada



# Normalización de la entrada





# Capa Competitiva

- El objetivo de esta capa es agrupar los datos de entrada. Las neuronas compiten entre si por representar a los patrones.
- La **normalización** de los datos de entrada **será reemplazada** por una **medida de similitud**.



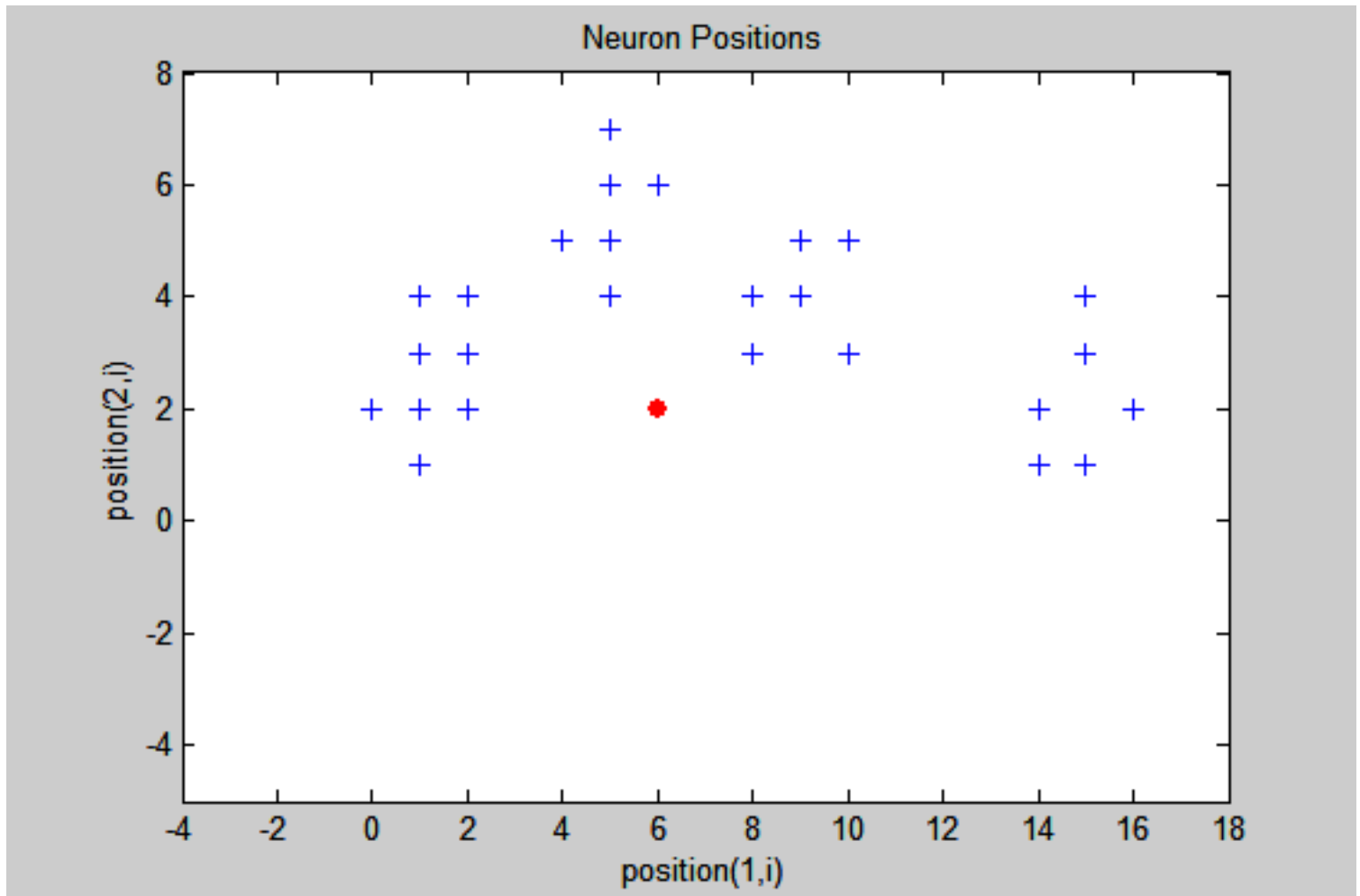
# Capa Competitiva

- Cuando una entrada  $I$  es presentada a la red, las neuronas de la capa competitiva calculan su entrada neta como:

$$neta_j = -distancia(I, W_j)$$

- Igual que antes, será considerada ganadora la que posea el valor más alto.

# Ejemplo



CPN2.m

# Ejemplo

```
P = [14 10 9 9 10 1 1 1 1 0 4 5 5 5 5 6 8 8 2 2 2 14 15 15 15 16;  
      1 3 4 5 5 1 2 3 4 2 5 4 5 6 7 6 3 4 2 3 4 2 1 3 4 2];
```

```
figure(1)
```

```
plot(P(1,:), P(2,:), '+')
```

```
axis( [-10 12 -5 8] )
```

```
ocultas = 4;
```

```
W = [ 0 0 0 0;  
      2 2 2 2];
```

```
hold on
```

```
plotsom(W)
```



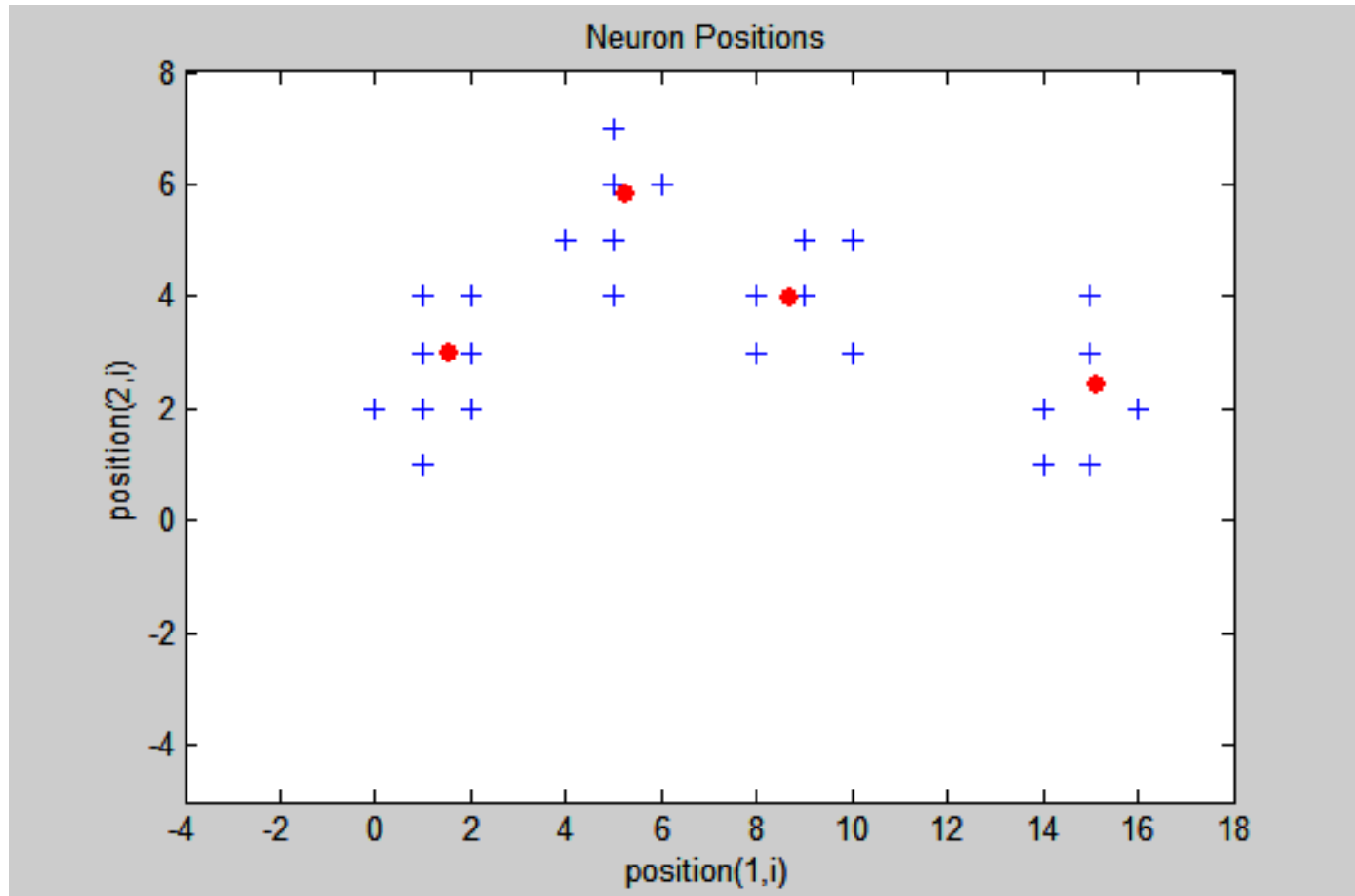
# Aprendizaje

- Mientras los pesos no se modifiquen demasiado
  - Para cada patrón de entrada
    - Calcular la neurona competitiva ganadora
    - Actualizar los pesos que llegan a ella.
  - Graficar los pesos de todas las neuronas competitivas.



# Ejemplo

CPN2.m





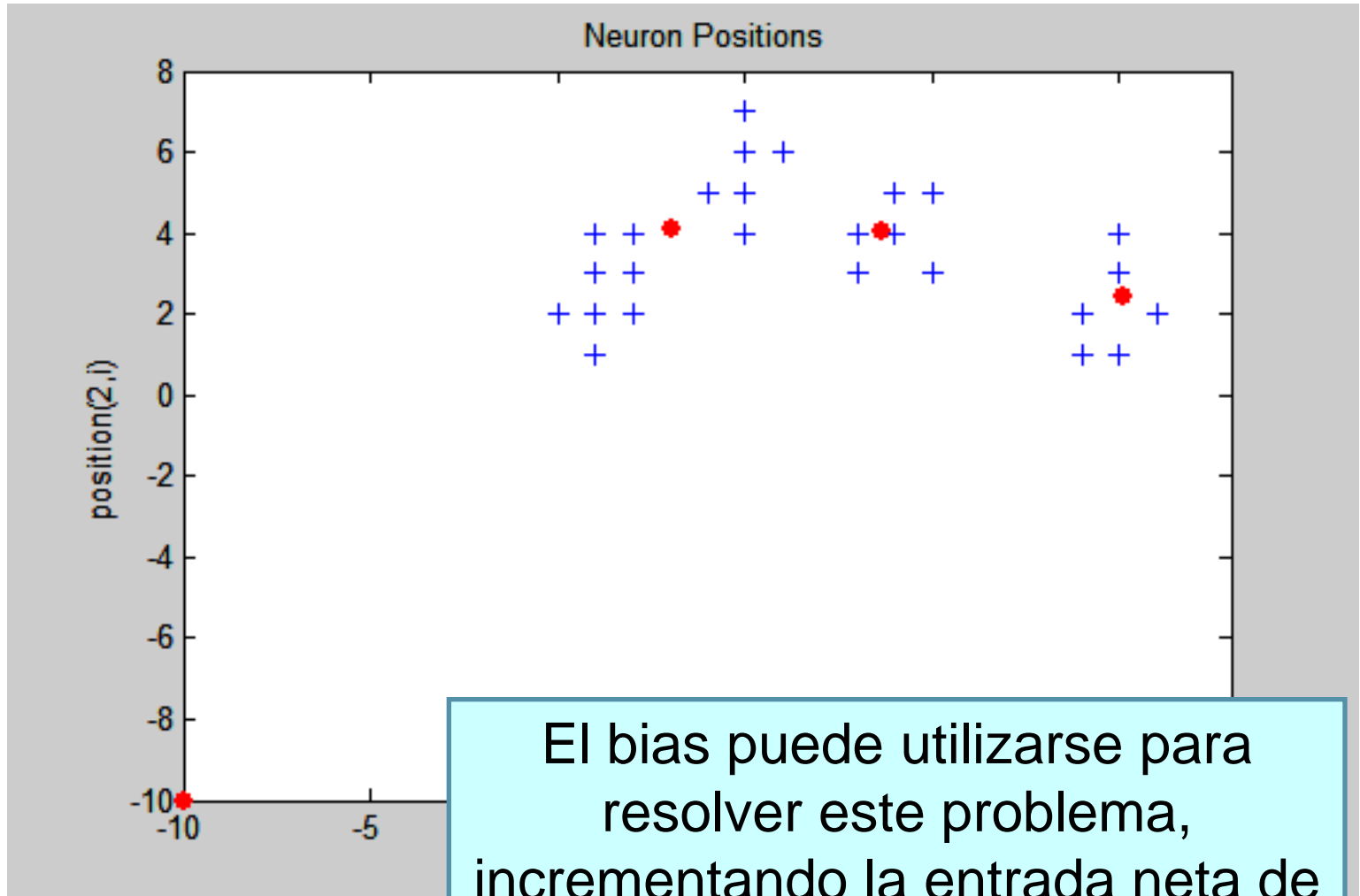
# Aprendizaje

## ■ Problema

- Puede ocurrir que algunos  $W$  no se actualicen correctamente. Si comienzan muy lejos de los vectores de entrada, nunca ganarán.
- Rehacer el ejemplo anterior comenzando con algún  $W$  en  $(-10,10)$ .

# Ejemplo

CPN3.m



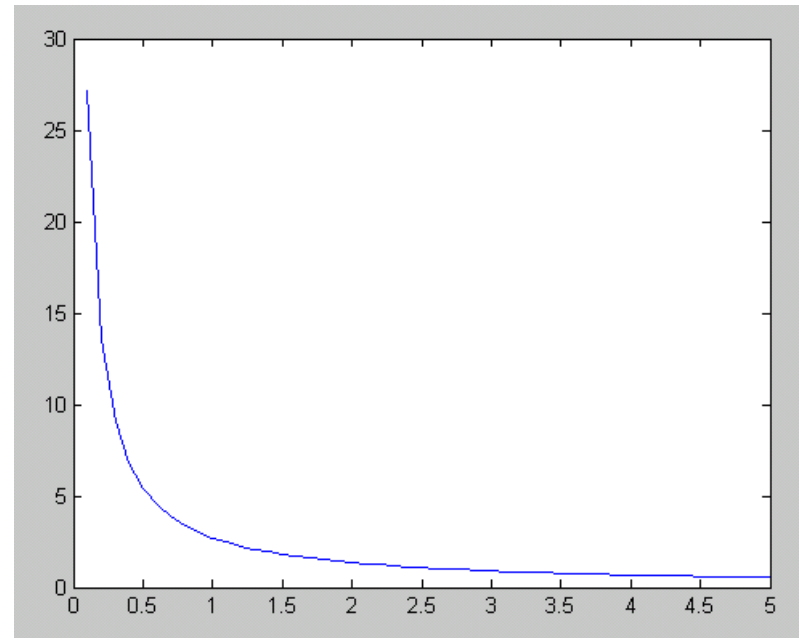


# Aprendizaje (entrada-oculta)

- Dado que se considera ganadora a la neurona con mayor entrada neta, el valor del bias de cada neurona competitiva debería disminuir cada vez que la neurona gana e incrementarse en caso contrario.
- De esta forma, todas las neuronas tienen las mismas posibilidades de ganar.
- ¿Cómo se modifica el valor del bias?

# Bias

- Debe utilizarse una función que dependa de la cantidad de veces que haya ganado una neurona competitiva



# Bias

- Valor inicial del bias

```
% s = cant. de neuronas ocultas
```

```
c = ones(s,1)/s;
```

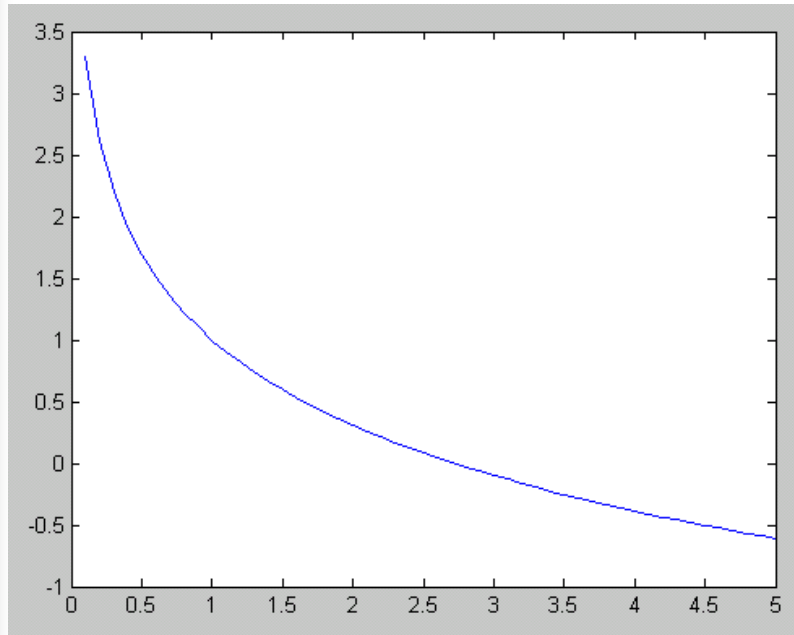
```
b = exp(1 - log(c));
```

- Todas las neuronas competitivas recibirán el mismo bias inicial.

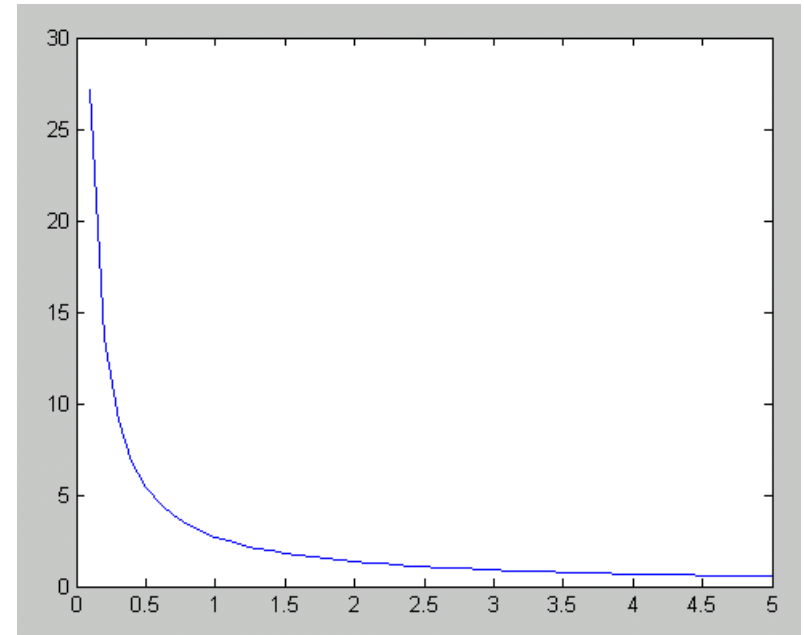
$$b = \exp(1 - \log(c))$$

donde  $c = 1 / (\text{cantidad de neuronas competitivas})$

# Valor inicial del bias



$$1 - \log(b)$$



$$\exp(1 - \log(b))$$

ej: con 3 neuronas ocultas, el valor inicial será

$$\exp(1 - \log(1/3)) = 8.1548$$



# Bias

$$y = \exp(1 - \log(x))$$

$$\log(y) = 1 - \log(x)$$

$$\log(y) - 1 = -\log(x)$$

$$1 - \log(y) = \log(x)$$

$$\exp(1 - \log(y)) = x$$



# Modificación del bias

- **Ejemplo:** Sea  $a = [1 ; 0 ; 0]$  el vector de salida obtenido luego de presentar el patrón a la capa competitiva (ganó la 1er. neurona)

```
alfa = 0.25;                                % b=[8.15;8.15;8.15]
```


```
% obtener el valor original
```

```
c = exp(1-log(b))                            % c = [1/3;1/3;1/3]
```

```
% actualizar para equilibrar las posibilidades
```

```
c = (1-alfa) * c + alfa * a                  % c = [1/2;1/4;1/4]
```

$(1 - 0.25) * 1/3 + 0.25 * 1$



# Modificación del bias

- **Ejemplo:** Sea  $a = [1 ; 0 ; 0]$  el vector de salida obtenido luego de presentar el patrón a la capa competitiva (ganó la 1er. neurona)


```
alfa =0.25;                                % b=[8.15;8.15;8.15]
```

```
% obtener el valor original
```

```
c = exp(1-log(b))                          % c = [1/3;1/3;1/3]
```

```
% actualizar para equilibrar las posibilidades
```

```
c = (1-alfa) * c + alfa * a                % c = [1/2;1/4;1/4]
```

$$(1 - 0.25) * 1/3 + 0.25 * 0$$


# Modificación del bias

- **Ejemplo:** Sea  $a = [1 ; 0 ; 0]$  el vector de salida obtenido luego de presentar el patrón a la capa competitiva (ganó la 1er. neurona)

```
alfa = 0.25;                                % b=[8.15;8.15;8.15]
% obtener el valor original
c = exp(1-log(b))                            % c = [1/3;1/3;1/3]
% actualizar para equilibrar las posibilidades
c = (1-alfa) * c + alfa * a                  % c = [1/2;1/4;1/4]

% nuevo bias
b = exp(1-log(c)) - alfa * b                 % c =[3.39;8.83;8.83]
```

# Ejemplo

CPN4\_conBias.m

- Agregar el bias al ejemplo anterior donde uno de los pesos está lejos de los patrones

