Redes basadas en la competición

- En ocasiones una red neuronal puede tener dificultades en elegir una salida de entre las posibles especificadas durante el entrenamiento.

- En situaciones donde es conveniente que solo una de varias neuronas responda, se puede introducir un mecanismo para garantizar esta circunstancia: la competición.

- En general, la competición se organiza con conexiones inhibidoras. En muchas ocasiones se utiliza el principio de Winner-Take-All (WTA), que establece que sólo la neurona que gana la competición tiene una salida distinta de cero. Alternativamente se puede utilizar un principio de WinnerLess Competition (WLC).

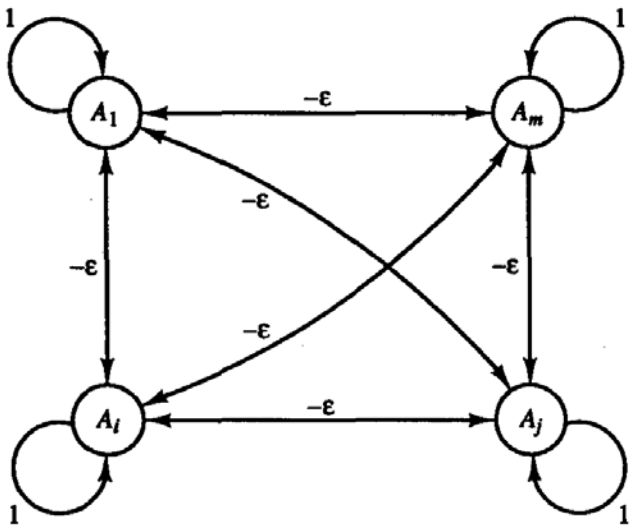
Tipos de redes basadas en la competición

1. Redes competitivas de pesos fijos: no utilizan un algoritmo de aprendizaje que modifique pesos. Las conexiones se definen a priori y lo que cambian son las activaciones de las neuronas:
    - Maxnet
    - Mexican hat
    - Red de Hamming
2. Redes competitivas con aprendizaje:
    - Redes de Kohonen (mapas autoorganizativos o SOM): utilizan aprendizaje no supervisado.
    - Redes LVQ (Learning Vector Quantization – aprendizaje por quantificicación vectorial): utilizan aprendizaje supervisado.

Maxnet

- Se debe a Lippmann, 1987.
- Es una red competitiva de pesos fijos con el principio de Winner-Take-all.
- Se puede utilizar como una subred para seleccionar el nodo que tiene la mayor entrada.
- Consta de m nodos que están completamente interconectados con pesos simétricos de valor -ε y autoconexiones de peso 1 (para considerar el valor previo).
- La función de activación es:

$$f(x) = \begin{cases} x & \text{si } x > 0 \\ 0 & \text{si } x <= 0 \end{cases}$$



Maxnet: algoritmo

Paso 0: Inicializar las activaciones y los pesos ($0<\varepsilon<1/m$):

$a_j(0)$ entrada a la neurona A_j

$w_{ij} = 1$ si $i = j$; $w_{ij} = -\varepsilon$ si $i \neq j$;

Paso 1: Mientras la condición de parada sea falsa, pasos 2-4:

Paso 2: Actualizar la activación de cada nodo

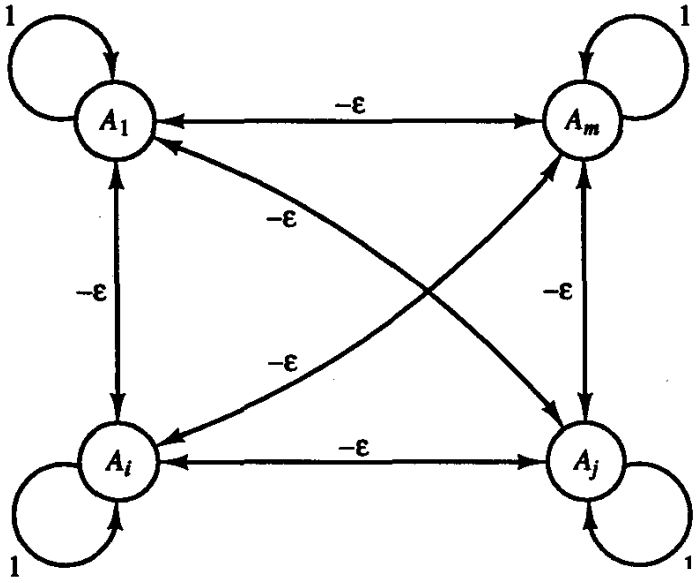
$$a_j(\text{nuevo}) = f[a_j(\text{anterior}) - \varepsilon \sum_{k \neq j} a_k(\text{anterior})], \; j = 1...m$$

Paso 3: Guardar las activaciones para utilizarlas en la siguiente iteración

$$a_j(\text{anterior}) = a_j(\text{nuevo}), \; j = 1...m$$

Paso 4: comprobar la condición de parada: si hay más de un nodo con activación distinta de cero continuar, si no, parar.

Maxnet: ejemplo



$$f(x) = \begin{cases} x & \text{si } x > 0 \\ 0 & \text{si } x <= 0 \end{cases}$$

Maxnet con 4 neuronas y pesos inhibidores ε=0.2, con las siguientes activaciones iniciales:

$$a_1(0) = 0.2 \quad a_2(0) = 0.4 \quad a_3(0) = 0.6 \quad a_4(0) = 0.8$$

$$a_1(1) = 0.0 \quad a_2(1) = 0.08 \quad a_3(1) = 0.32 \quad a_4(1) = 0.56$$

$$a_1(2) = 0.0 \quad a_2(2) = 0.0 \quad a_3(2) = 0.192 \quad a_4(2) = 0.48$$

$$a_1(3) = 0.0 \quad a_2(3) = 0.0 \quad a_3(3) = 0.096 \quad a_4(3) = 0.442$$

$$a_1(4) = 0.0 \quad a_2(4) = 0.0 \quad a_3(4) = 0.008 \quad a_4(4) = 0.422$$

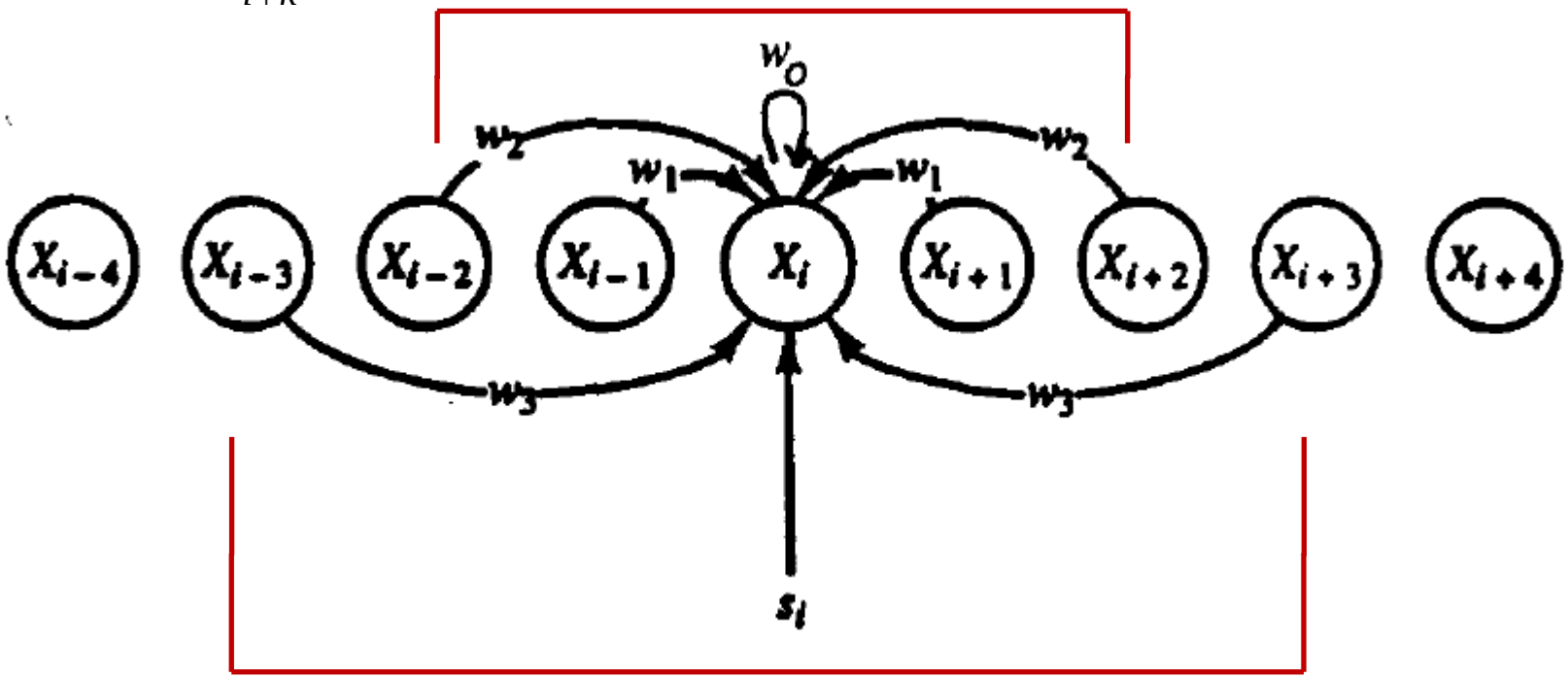
$$a_1(5) = 0.0 \quad a_2(5) = 0.0 \quad a_3(5) = 0.0 \quad a_4(5) = 0.421$$

Mexican Hat

- Se debe a Kohonen, 1989.
- Se puede utilizar como una subred en una capa y por tanto las neuronas reciben una señal externa además de las señales que corresponden a las interconexiones.
- Es una red competitiva de pesos fijos con mejora de contraste de tipo Centro-*On* Periferia-*Off* y pesos fijos.
- Cada neurona está conectada con conexiones excitadoras a un número de neuronas “cooperadoras” próximas y con conexiones inhibidoras (pesos negativos) a un número de neuronas “competidoras” que están más alejadas espacialmente.
- Puede haber un conjunto de neuronas más alejadas con las que una cierta neurona no está conectada. Este patrón de conexiones se repite para cada neurona de una capa.

Mexican Hat

neuronas x_{i+k}

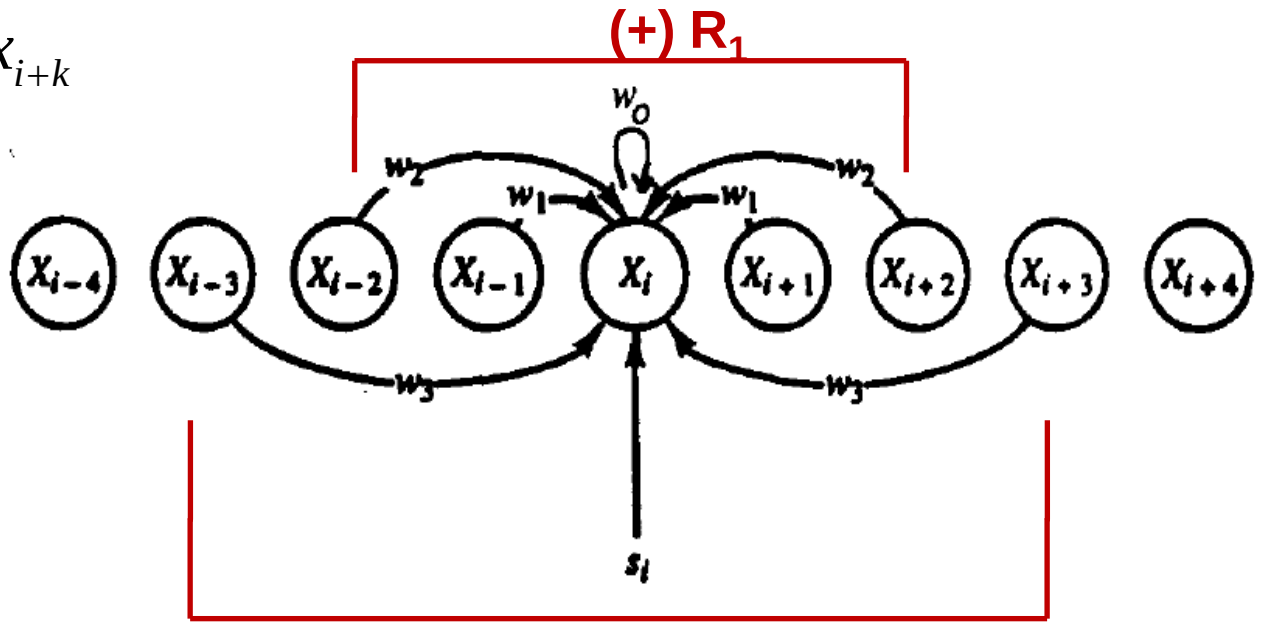


La arquitectura también puede ser multidimensional y con condiciones de contorno periódicas. La función de transferencia para cada neurona es:

$$x_i(\mathrm{t}) = f[s_i(\mathrm{t}) + \sum_k w_k x_{i+k}(\mathrm{t}-1)]$$

Mexican Hat: Notación

neuronas x_{i+k}



R_1: radio de la región con conexiones excitadoras
R_2: radio de la región con conexiones inhibidoras ($R_1 < R_2$)
w_k peso de las interconexiones entre las neuronas X_i y las neuronas X_{i+k} y X_{i-k}:

$$w_k \text{ es positivo para } 0 \le k \le R_1$$

$$w_k \text{ es negativo para } R_1 < k \le R_2$$

x vector de activaciones (***x***_*anterior* es el vector de activaciones en el paso previo), **s** señal externa, *t_max* número de iteraciones.

Mexican Hat: Algoritmo (1/2)

Paso 0: Inicializar los parámetros t_max, R_1 y R_2:

Inicializar los pesos:

$w_k = C_1$ para $k = 0,\ldots, R_1$ $(C_1>0)$

$w_k = C_2$ para $k = R_1+1,\ldots, R_2$ $(C_2<0)$

Paso 1: Presentar el estímulo **s**: $\mathbf{x}=\mathbf{s}$

Almacenar las activaciones en el vector $\boldsymbol{x}\_anterior$

$x\_anterior_i = x_i$ $(i=1,\ldots,n)$

Inicializar el contador de las iteraciones: $t=1$

Paso 2: Mientras $t< t\_max$, ejecutar pasos 3-7:

Paso 3: Calcular la entrada neta $(i=1,\ldots,n)$

$$x_i = C_1 \sum_{k=-R_1}^{R_1} x\_anterior_{i+k} + C_2 \sum_{k=-R_2}^{-R_1-1} x\_anterior_{i+k} + C_2 \sum_{k=R_1+1}^{R_2} x\_anterior_{i+k}$$

Mexican Hat: Algoritmo (2/2)

Paso 4: aplicar la función de activación (una función rampa de 0 a x_max, con pendiente 1):

$$x_i = \min(x\_max, \max(0, x_i)) \quad (i=1,\ldots, n)$$

Paso 5: guardar las activaciones en $\boldsymbol{x}\_anterior$:

$$x\_anterior_i = x_i \ (i=1,\ldots,n)$$

Paso 6: incrementar el contador de iteraciones: $t=t+1$

Paso 7: comprobar la condición de parada:

Si $t < t\_max$ continuar, si no, parar.

El efecto del refuerzo positivo de las neuronas cercanas y el refuerzo negativo de las neuronas lejanas resulta en un incremento de la activación de las neuronas con mayor activación inicial y una reducción de activación de las neuronas con menor estímulo externo.

Mexican Hat: Ejemplo (1/2)

Consideramos una red con 7 neuronas y la siguiente función de activación:

$$f(x) = \begin{cases} 0 & \text{si } x < 0 \\ x & \text{si } 0 \le x \le 2 \\ 2 & \text{si } 2 < \text{x} \end{cases}$$

Paso 0: Inicializar los parámetros: R_1=1 y R_2=2, C_1=0.6 y C_2=-0.4

Paso 1: El estímulo externo es (0.0, 0.5, 0.8, 1.0, 0.8, 0.5, 0.0), luego

x = (0.0, 0.5, 0.8, 1.0, 0.8, 0.5, 0.0), que se almacena en

x_anterior = (0.0, 0.5, 0.8, 1.0, 0.8, 0.5, 0.0)

Paso 2: **bucle**

$$x_i = C_1 \sum_{k=-R_1}^{R_1} x\_anterior_{i+k} + C_2 \sum_{k=-R_2}^{-R_1-1} x\_anterior_{i+k} + C_2 \sum_{k=R_1+1}^{R_2} x\_anterior_{i+k}$$

Paso 3 (t=1)

x_1=0.6(0.0)+0.6(0.5)-0.4(0.8)=-0.2

x_2=0.6(0.0)+0.6(0.5)+0.6(0.8)-0.4(1.0)=0.38

x_3=-0.4(0.0)+0.6(0.5)+0.6(0.8)+0.6(1.0)-0.4(0.8)=1.06

x_4=-0.4(0.5)+0.6(0.8)+0.6(1.0)+0.6(0.8)-0.4(0.5)=1.16

x_5=-0.4(0.8)+0.6(1.0)+0.6(0.8)+0.6(0.5)-0.4(0.0)=1.06

x_6=-0.4(1.0)+0.6(0.8)+0.6(0.5)+0.6(0.0)=0.38

x_7=-0.4(0.8)+0.6(0.5)+0.6(0.0)=-0.2

Mexican Hat: Ejemplo (2/2)

$$f(x) = \begin{cases} 0 & \text{si } x < 0 \\ x & \text{si } 0 \le x \le 2 \\ 2 & \text{si } 2 < x \end{cases}$$

Paso 2: bucle (continúa)

Paso 4: **x** = (0.0, 0.38, 1.06, 1.16, 1.06, 0.38, 0.0)

Paso 5-7: $x\_anterior_i = x_i$ (i=1,…,n), t=t+1

Paso 3: (t=2)

x_1=0.6(0.0)+0.6(0.38)-0.4(1.06)=-0.196

x_2=0.6(0.0)+0.6(0.38)+0.6(1.06)-0.4(1.16)=0.39

x_3=-0.4(0.0)+0.6(0.38)+0.6(1.06)+0.6(1.16)-0.4(1.06)=1.14

x_4=-0.4(0.38)+0.6(1.06)+0.6(1.16)+0.6(1.06)-0.4(0.38)=1.16

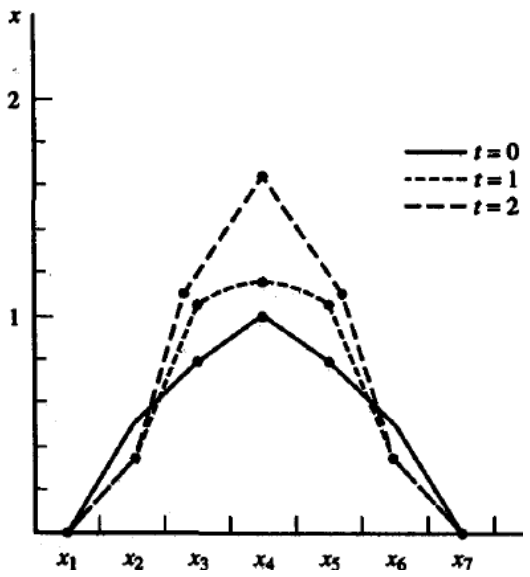
x_5=-0.4(1.06)+0.6(1.16)+0.6(1.06)+0.6(0.38)-0.4(0.0)=1.14

x_6=-0.4(1.16)+0.6(1.06)+0.6(0.38)+0.6(0.0)=0.39

x_7=-0.4(1.06)+0.6(0.38)+0.6(0.0)=-0.196

Paso 4: **x** = (0.0, 0.39, 1.14, 1.66, 1.14, 0.39, 0.0)

Paso 5-7: $x\_anterior_i = x_i$ (i=1,…,n), t=t+1, ……



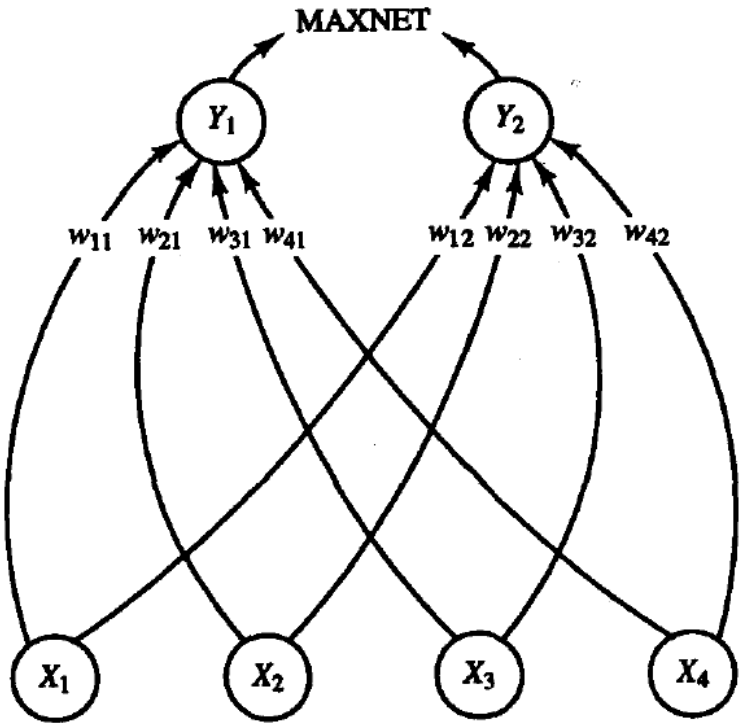
Red de Hamming

- Se debe a Lippmann, 1987.
- Es una red clasificadora por máxima verosimilitud que se puede utilizar para determinar cuál de un conjunto de vectores de ejemplo se parece al vector de entrada (de dimensión n). Los vectores de ejemplo determinan los pesos de la red.
- La medida de similitud entre el vector de entrada y los vectores ejemplos es n menos la distancia de Hamming entre los vectores.
- La distancia de Hamming entre dos vectores es el número de componentes distintas.
- Para vectores bipolares $\mathbf{x}$ e $\mathbf{y}$, $\mathbf{x}\cdot\mathbf{y}=a-d$, donde a es el número de componentes comunes a los dos vectores y d el número de componentes diferentes (la distancia de Hamming): $d=n-a$, $\mathbf{x}\cdot\mathbf{y}=2a-n$ *ó* $2a=\mathbf{x}\cdot\mathbf{y}+n$

Red de Hamming: arquitectura

- Para vectores bipolares $\mathbf{x}$ e $\mathbf{y}$, $\mathbf{x}\cdot\mathbf{y}=a-d$, donde a es el número de componentes comunes a los dos vectores y d el número de componentes diferentes (la distancia de Hamming): $d=n-a$, $\mathbf{x}\cdot\mathbf{y}=2a-n$ *ó* $2a=\mathbf{x}\cdot\mathbf{y}+n$
- Si los pesos se establecen a la mitad del vector ejemplo y el bias a $n/2$, la red encontrará la neurona con el ejemplo más cercano, simplemente encontrando a la neurona con la mayor entrada.
- La red de Hamming utiliza MAXNET como una subred para encontrar la neurona con la mayor entrada neta.
- Los n nodos de entrada se conectan a los m nodos de salida (m es el número de vectores de ejemplo) almacenados en la red.
- Los vectores de entrada y de ejemplo son bipolares.

Red de Hamming: ejemplo de arquitectura



En este ejemplo los vectores de entrada tienen 4 componentes, y se categorizan en una de dos clases.

Red de Hamming:notación

- Dado un conjunto de m vectores de ejemplo bipolares, $\mathbf{e}(1)$, $\mathbf{e}(2),\ldots,\mathbf{e}(m)$, la red de Hamming se utiliza para encontrar el vector que es más parecido al vector de entrada bipolar.

- La entrada neta y_in_j a la neurona Y_j, proporciona el número de componentes comunes entre el vector de entrada y el vector ejemplo $\mathbf{e}(j)$ para la neurona $\mathbf{Y}_j$ (n menos la distancia de Hamming entre estos vectores).

- n: número de neuronas de entradas (nº de componentes del vector de entrada).
- m: número de neuronas de salida (nº de vectores de ejemplo).
- $e(j)$: el vector de ejemplo j-ésimo: $e(j)=(e_1(j),\ldots e_i(j),\ldots., e_n(j))$

Red de Hamming: Algoritmo

Paso 0: Inicializar pesos y los sesgos para almacenar los m vectores de ejemplo:

$$w_{ij} = e_i(j)/2 \quad (i=1,\ldots,n;\ j=1,\ldots,m)$$

$$b_j = n/2 \quad (j=1,\ldots,m)$$

Paso 1: Para cada vector **x** ejecutar los pasos 2-4:

Paso 2: Calcular la entrada neta a cada neurona Y_j:

$$y\_in_j = b_j + \sum_i x_i w_{ij} \quad (j = 1,\ldots,m)$$

Paso 3: Inicializar las activaciones para MAXNET:

$$y_j(0) = y\_in_j \quad (j = 1,\ldots,m)$$

Paso 4: MAXNET itera hasta encontrar el mejor ejemplo

Red de Hamming: Ejemplo (1/5)

Dados los vectores de ejemplo:

$\boldsymbol{e}(1)=(1,-1,-1,-1)$

$\boldsymbol{e}(2)=(-1,-1,-1,1)$

se puede utilizar la red de Hamming para encontrar el vector de ejemplo que es más parecido a los patrones de entrada bipolares (1,1,-1,-1), (1,-1,-1,-1), (-1,-1,-1,1) y (-1,-1,1,1)

Paso 0: Inicializar pesos y los sesgos para almacenar los m vectores de ejemplo:

$$\mathbf{W} = \begin{pmatrix} .5 & -.5 \\ -.5 & -.5 \\ -.5 & -.5 \\ -.5 & .5 \end{pmatrix} \qquad b_1 = b_2 = 2$$

Red de Hamming: Ejemplo (2/5)

Paso 1: Para el vector $\mathbf{x}=(1,1,-1,-1)$ ejecutar los pasos 2-4:

Paso 2: Calcular la entrada neta a cada neurona Y_j:

$$y\_in_1 = b_1 + \sum_i x_i w_{i1} = 2+1=3$$

$$y\_in_2 = b_2 + \sum_i x_i w_{i2} = 2-1=1$$

Estos valores representan la similitud Hamming ya que $(1,1,-1,-1)$ tiene las mismas 1ª, 3ª y 4ª componentes que $\mathbf{e}(1)=(1,-1,-1,-1)$ y porque $(1,1,-1,-1)$ tienen la misma 3ª componente que $\mathbf{e}(2)=(-1,-1,-1,1)$.

Paso 3: $y_1(0)=3 \quad y_2(0)=1$

Paso 4: Puesto que $y_1(0)>y_2(0)$, MAXNET determinará que la neurona Y_1, representa el mejor ejemplo para el vector $\mathbf{x}=(1,1,-1,-1)$

Red de Hamming: Ejemplo (3/5)

Paso 1: Para el vector $\mathbf{x}=(1,-1,-1,-1)$ ejecutar los pasos 2-4:

Paso 2: Calcular la entrada neta a cada neurona Y_j:

$$y\_in_1 = b_1 + \sum_i x_i w_{i1} \;=\; 2+2=4$$

$$y\_in_2 = b_2 + \sum_i x_i w_{i2} \;=\; 2+0=2$$

El vector de entrada $(1,-1,-1,-1)$ coincide con el vector $\mathbf{e}(1)=(1,-1,-1,-1)$ y con $\mathbf{e}(2)=(-1,-1,-1,1)$ en la 2ª y 3ª componentes

Paso 3: $y_1(0)=4 \qquad y_2(0)=2$

Paso 4: Puesto que $y_1(0)>y_2(0)$, MAXNET determinará que la neurona Y_1, representa el mejor ejemplo para el vector $\mathbf{x}=(1,-1,-1,-1)$

Red de Hamming: Ejemplo (4/5)

Paso 1: Para el vector $\mathbf{x}=(-1,-1,-1,1)$ ejecutar los pasos 2-4:

Paso 2: Calcular la entrada neta a cada neurona Y_j:

$$y\_in_1 = b_1 + \sum_i x_i w_{i1} = 2 + 0 = 2$$

$$y\_in_2 = b_2 + \sum_i x_i w_{i2} = 2 + 2 = 4$$

El vector de entrada $(-1,-1,-1,1)$ coincide con el vector $\mathbf{e}(1)=(1,-1,-1,-1)$ en las 2ª y 3ª componentes y con el vector $\mathbf{e}(2)=(-1,-1,-1,1)$ en las 4 componentes

Paso 3: $y_1(0) = 2 \qquad y_2(0) = 4$

Paso 4: Puesto que $y_2(0)>y_1(0)$, MAXNET determinará que la neurona Y_2, representa el mejor ejemplo para el vector $\mathbf{x}=(-1,-1,-1,1)$

Red de Hamming: Ejemplo (5/5)

Paso 1: Para el vector $\mathbf{x}$=(-1,-1,1,1) ejecutar los pasos 2-4:

Paso 2: Calcular la entrada neta a cada neurona Y_j:

$$y\_in_1 = b_1 + \sum_i x_i w_{i1} = 2 - 1 = 1$$

$$y\_in_2 = b_2 + \sum_i x_i w_{i2} = 2 + 1 = 3$$

El vector de entrada (-1,-1,1,1) coincide con el vector $\mathbf{e}(1)$=(1,-1,-1,-1) en la 2ª componente y con $\mathbf{e}(2)$=(-1,-1,-1,1) en la 1ª, 2ª y 4ª componentes.

Paso 3: $y_1(0) = 1 \qquad y_2(0) = 3$

Paso 4: Puesto que $y_2(0) > y_1(0)$, MAXNET determinará que la neurona Y_2, representa el mejor ejemplo para el vector $\mathbf{x}$=(-1,-1,1,1)

Recordatorio: tipos de redes basadas en la competición

1. Redes competitivas de pesos fijos: no utilizan un algoritmo de aprendizaje que modifique pesos. Las conexiones se definen a priori y lo que cambian son las activaciones de las neuronas:
    - Maxnet
    - Mexican hat
    - Red de Hamming
2. Redes competitivas con aprendizaje:
    - Redes de Kohonen (mapas autoorganizativos o SOM): utilizan aprendizaje no supervisado.
    - Redes LVQ (Learning Vector Quantization – aprendizaje por quantificicación vectorial): utilizan aprendizaje supervisado.

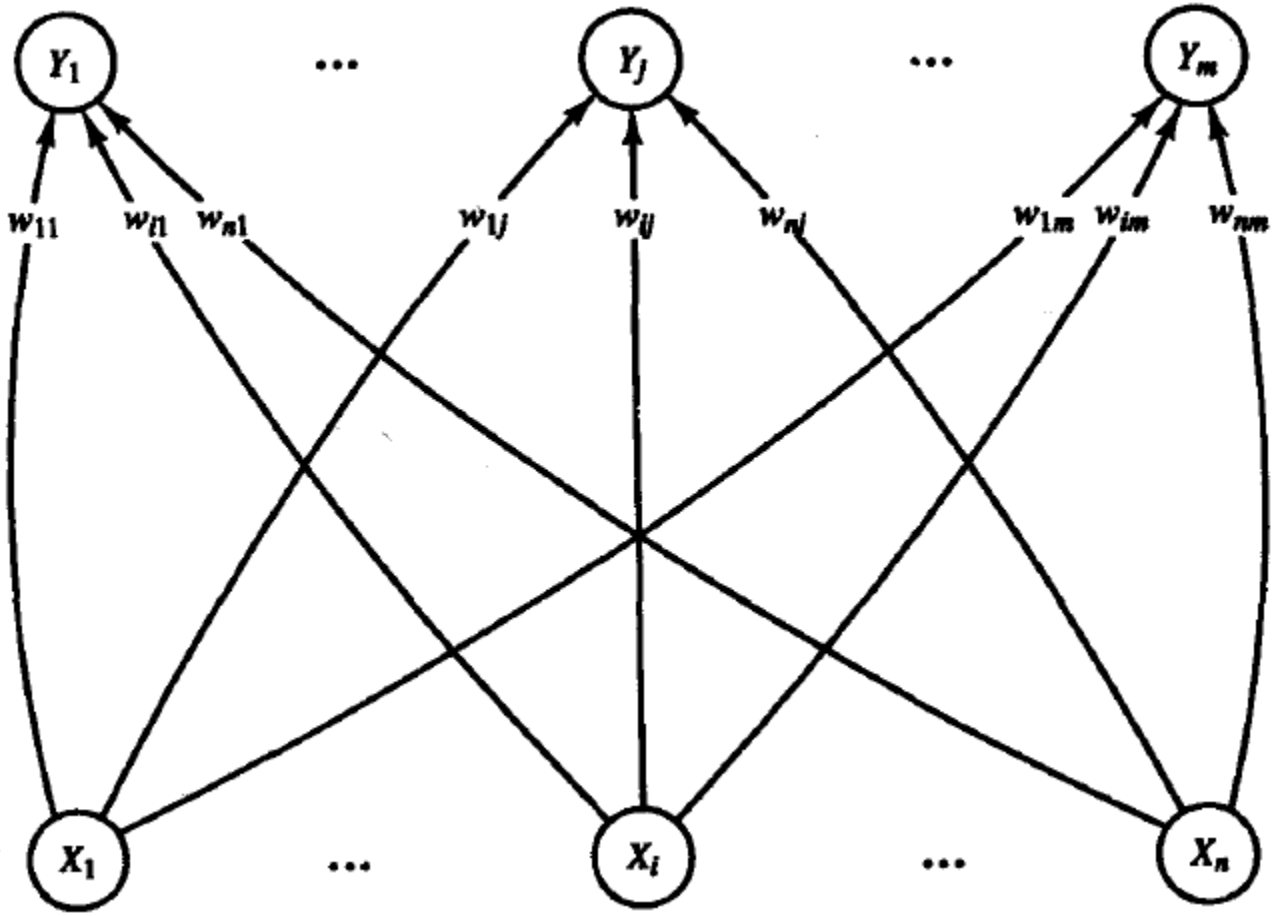
Mapas auto-organizativos de Kohonen

- Se deben a Kohonen, 1989 (SOM).
- También se denominan mapas que preservan la topología puesto que asumen una estructura topológica entre neuronas que forman un *cluster*.
- Hay m neuronas en *cluster*, organizadas en un array mono o multidimensional. Los vectores de entrada tienen n componentes.
- El vector de pesos de una neurona del *cluster* actúa de ejemplo de los patrones de entrada que se asocian con ese *cluster*.
- La red evoluciona con un proceso auto-organizativo en el que la neurona del *cluster* cuyo vector de pesos coincide mejor con el patrón de estímulos es la ganadora.
- Una posible medida de similitud es el cuadrado de la distancia euclídea mínima.

Mapas auto-organizativos de Kohonen

- La neurona que gana y sus vecinas (definidas según la topología de las neuronas del *cluster*), actualizan sus pesos.
- Un ejemplo de vecindad en un *array* lineal de neuronas de *cluster* con radio R alrededor de la neurona J serían las neuronas j que cumplen: $\max(1,J\text{-}R) \leq j \leq \min(J+R,m)$).
- Los vectores de peso de las neuronas vecinas no son, en general, parecidos al patrón de entrada.
- La red puede utilizarse para agrupar o *clusterizar* un conjunto de p vectores de coordenadas continuas $\mathbf{x}=(x_1,\ldots,x_i,\ldots x_n)$ en m *clusters*.
- Los pesos no se multiplican por la señal enviada desde las neuronas de entrada a las neuronas *cluster* a no ser que se utilice una medida de similitud que involucre el producto escalar.

Mapas auto-organizativos: Arquitectura



Mapas auto-organizativos: Arquitectura

Array lineal de neuronas de *cluster*

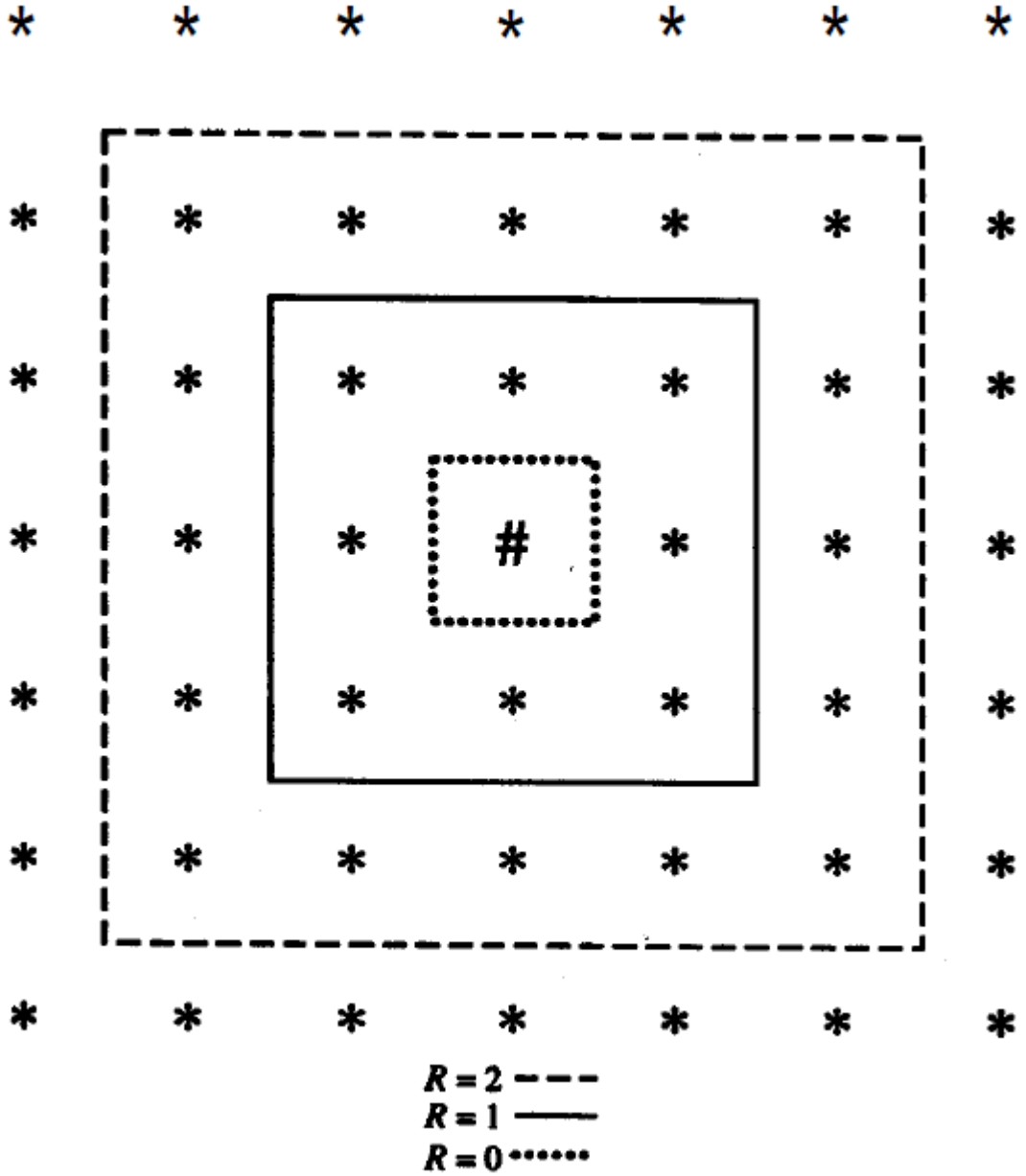
```
*   *   {*   (*   [#]   *)   *}   *   *  *
```

```
[] R=0 () R=1 {} R=2
```

\# neurona ganadora

Mapas auto-organizativos: Arquitectura

Vecindades para una red rectangular

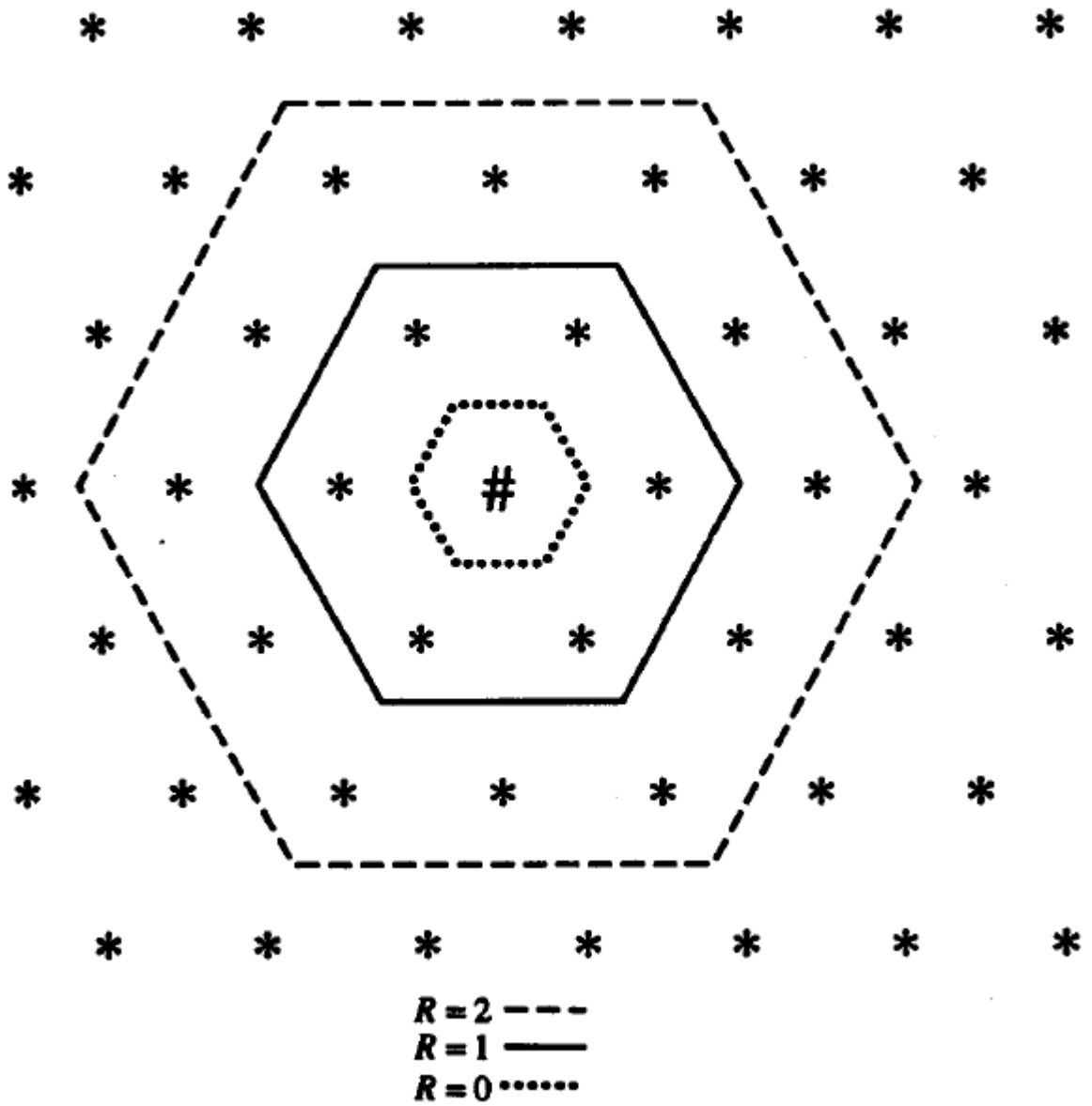


Cada neurona tiene 8 vecinos más próximos.

No hay condiciones de contorno periódicas.

Mapas auto-organizativos: Arquitectura

Vecindades para una red hexagonal



Cada neurona tiene 6 vecinos más próximos.

No hay condiciones de contorno periódicas.

Mapas auto-organizativos: Algoritmo

Paso 0: Inicializar los pesos w_{ij} . Establecer los parámetros de la topología de vecinos y los factores de aprendizaje.

Paso 1: Mientras la condición de parada sea falsa, ejecutar los pasos 2-4:

Paso 2: Para cada vector de entrada $\mathbf{x}$, ejecutar 3-5*:*

Paso 3: Para cada j, calcular: $D(j) = \sum_i (w_{ij} - x_i)^2$

Paso 4: Encontrar el índice J tal que $D(J)$ es mínimo

Paso 5: Para todas las neuronas j dentro de una vecindad de J, y para todo i*:*

$$w_{ij}(nuevo) = w_{ij}(anterior) + \alpha[x_i - w_{ij}(anterior)]$$

Paso 6: Actualizar la tasa de aprendizaje.

Paso 7: Reducir el radio de la vecindad topológica

Paso 8: Comprobar la condición de parada.

Mapas auto-organizativos: Observaciones

- La inicialización de los pesos puede ser aleatoria. Si se tiene algo de información relativa a la *clusterización*, se puede utilizar en la inicialización.
- La tasa de aprendizaje α es una función decreciente del tiempo (de las épocas de entrenamiento). Normalmente es suficiente con una función linealmente decreciente.
- El radio de la vecindad alrededor de una neurona de *cluster* también decrece a medida que el proceso de *clustering* progresa.
- La formación de un mapa ocurre en dos fases: la formación inicial del orden correcto y la de la convergencia final. La segunda fase requiere más tiempo que la primera y requiere un valor pequeño de α.
- El proceso de aprendizaje puede requerir varias iteraciones con el conjunto de entrenamiento.

Mapas auto-organizativos: Ejemplos.
1. Red SOM para *clusterizar* 4 vectores

- Se quiere *clusterizar* los vectores $(1,1,0,0)$; $(0,0,0,1)$; $(1,0,0,0)$ y $(0,0,1,1)$
- El número máximo de *clusters* deseados es $m=2$.
- Se propone al siguiente tasa inicial de aprendizaje y su correspondiente función decreciente: $\alpha(0) = 0.6$

$$\alpha(t+1) = 0.5\alpha(t)$$

- Con solo dos *clusters* disponibles, la vecindad del nodo J se establece de forma que sólo uno de los *clusters* actualiza sus pesos en cada paso de tiempo ($R=0$).

Mapas auto-organizativos: Ejemplos.
1. Red SOM para *clusterizar* 4 vectores (1/4)

Paso 0: Inicializar los pesos w_{ij} aleatoriamente

$$\mathbf{W} = \begin{pmatrix} .2 & .8 \\ .6 & .4 \\ .5 & .7 \\ .9 & .3 \end{pmatrix}$$

Inicializar el radio R=0

Inicializar la tasa de aprendizaje: $\alpha(0)=0.6$

Paso 1: Comienzo del entrenamiento

Paso 2: Para el primer vector $(1,1,0,0)$, ejecutar 3-5:

Paso 3: $D(1)= (.2\text{-}1)^2+ (.6\text{-}1)^2+ (.5\text{-}0)^2+ (.9\text{-}0)^2=1.86$

$D(2)= (.8\text{-}1)^2+ (.4\text{-}1)^2+ (.7\text{-}0)^2+ (.3\text{-}0)^2=0.98$

Paso 4: El índice J tal que $D(J)$ es mínimo es 2 (el vector de entrada es más parecido a la neurona 2)

Paso 5: Se actualizan los pesos de neurona ganadora:

$w_{i2}(nuevo)=w_{i2}(anterior)+0.6[x_i\text{-}w_{i2}(anterior)]=0.4w_{i2}(anterior)+0.6x_i$

La matriz de pesos queda:

$$\mathbf{w} = \begin{pmatrix} .2 & ..92 \\ .6 & .76 \\ .5 & .28 \\ .9 & .12 \end{pmatrix}$$

Mapas auto-organizativos: Ejemplos.
1. Red SOM para *clusterizar* 4 vectores (2/4)

Paso 2: Para el segundo vector $(0,0,0,1)$, ejecutar 3-5*:*

Paso 3: $D(1)= (.2-0)^2+ (.6-0)^2+ (.5-0)^2+ (.9-1)^2=0.66$

$D(2)= (.92-0)^2+ (.76-0)^2+ (.28-0)^2+ (.12-1)^2=2.28$

Paso 4: El índice J tal que $D(J)$ es mínimo es 1 (el vector de entrada es más parecido a la neurona 1)

Paso 5: Se actualizan los pesos de neurona ganadora (primera columna de la matriz): $\mathbf{W} = \begin{pmatrix} .08 & .92 \\ .24 & .76 \\ .20 & .28 \\ .96 & .12 \end{pmatrix}$

Paso 2: Para el tercer vector $(1,0,0,0)$, ejecutar 3-5*:*

Paso 3: $D(1)= (.08-1)^2+ (.24-0)^2+ (.2-0)^2+ (.96-0)^2=1.87$

$D(2)= (.92-1)^2+ (.76-0)^2+ (.28-0)^2+ (.12-0)^2=0.68$

Paso 4: El índice J tal que $D(J)$ es mínimo es 2 (el vector de entrada es más parecido a la neurona 2)

Paso 5: Se actualizan los pesos de neurona ganadora (segunda columna de la matriz): $\mathbf{W} = \begin{pmatrix} .08 & .968 \\ .24 & .304 \\ .20 & .112 \\ .96 & .048 \end{pmatrix}$

Mapas auto-organizativos: Ejemplos.
1. Red SOM para *clusterizar* 4 vectores (3/4)

Paso 2: Para el cuarto vector $(0,0,1,1)$, ejecutar 3-5:

Paso 3: $D(1)= (.08-0)^2+ (.24-0)^2+ (.2-1)^2+ (.96-1)^2=0.71$

$D(2)= (.968-0)^2+ (.304-0)^2+ (.112-1)^2+ (.048-1)^2=2.72$

Paso 4: El índice J tal que $D(J)$ es mínimo es 1 (el vector de entrada es más parecido a la neurona 1)

Paso 5: Se actualizan los pesos de neurona ganadora (primera columna de la matriz):

$$\mathbf{W} = \begin{pmatrix} .032 & .968 \\ .096 & .304 \\ .680 & .112 \\ .984 & .048 \end{pmatrix}$$

Paso 6: Reducir la tasa de aprendizaje: $\alpha(t+1) = 0.5(0.6) = 0.3$

La actualización de pesos queda ahora:

$$w_{ij}(nuevo)=w_{ij}(anterior)+0.3[x_i-w_{ij}(anterior)]=0.7w_{ij}(anterior)+0.3x_i$$

La matriz de pesos tras la segunda época de aprendizaje es:

$$\mathbf{W} = \begin{pmatrix} .016 & .980 \\ .047 & .360 \\ .630 & .055 \\ .999 & .024 \end{pmatrix}$$

Mapas auto-organizativos: Ejemplos.
1. Red SOM para *clusterizar* 4 vectores (4/4)

Si se modifica la tasa de aprendizaje de forma que decrezca desde 0.6 a 0.01 en 100 iteraciones (épocas), da como resultado:

- Iteración 0: $\mathbf{W} = \begin{pmatrix} .2 & .8 \\ .6 & .4 \\ .5 & .7 \\ .9 & .3 \end{pmatrix}$
- Iteración 1: $\mathbf{W} = \begin{pmatrix} .032 & .970 \\ .096 & .300 \\ .680 & .110 \\ .980 & .048 \end{pmatrix}$
- Iteración 2: $\mathbf{W} = \begin{pmatrix} .0053 & .99 \\ -.17 & .3 \\ .7 & .02 \\ 1.0 & .0086 \end{pmatrix}$

…..

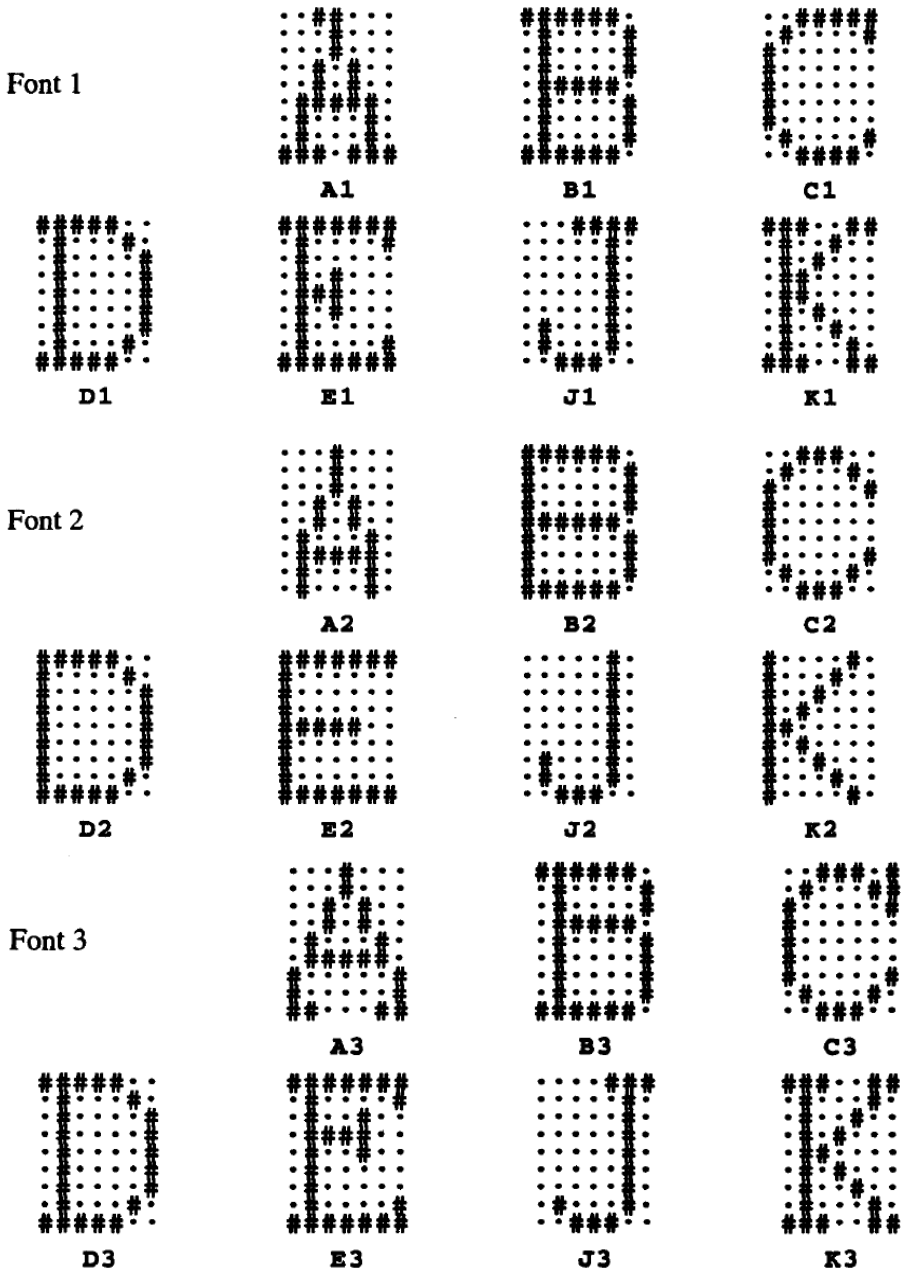
- Iteración 100: $\mathbf{W} = \begin{pmatrix} .0 & 1.0 \\ .0 & .49 \\ .51 & .0 \\ 1.0 & .0 \end{pmatrix}$

La matriz de pesos converge a: $\mathbf{W} = \begin{pmatrix} .0 & 1.0 \\ .0 & .5 \\ .5 & .0 \\ 1.0 & .0 \end{pmatrix}$

La primera columna es el promedio de los dos vectores que están en el *cluster* 1 y la segunda columna es el promedio de los vectores que están en el *cluster* 2.

Mapas auto-organizativos: Ejemplos

Reconocimiento de caracteres (1/3)



25 clusters

Mapas auto-organizativos: Ejemplos

Reconocimiento de caracteres (2/3)

La tasa de aprendizaje se reduce linealmente de 0.6 a 0.01:

•Sin estructura topológica (sólo la neurona ganadora aprende el patrón presentado): Se forman 5 clusters:

neurona	Patrones
3	C1,C2,C3
13	B1,B3,D1,D3,E1,K1,K3,E3
16	A1,A2,A3
18	J1,J2,J3
24	B2,D2,E2,K2

Mapas auto-organizativos: Ejemplos

Reconocimiento de caracteres (3/3)

La tasa de aprendizaje se reduce linealmente de 0.6 a 0.01.

• Con estructura lineal (R=1, se permite que la neurona ganadora J y sus vecinos J+1 y J-1) aprendan.

neurona	Patrones
6	K2
10	J1,J2,J3
14	E1,E3
16	K1,K3
18	B1,B3,D1,D3
20	C1,C2,C3
22	D2
23	B2,E3
25	A1,A2,A3

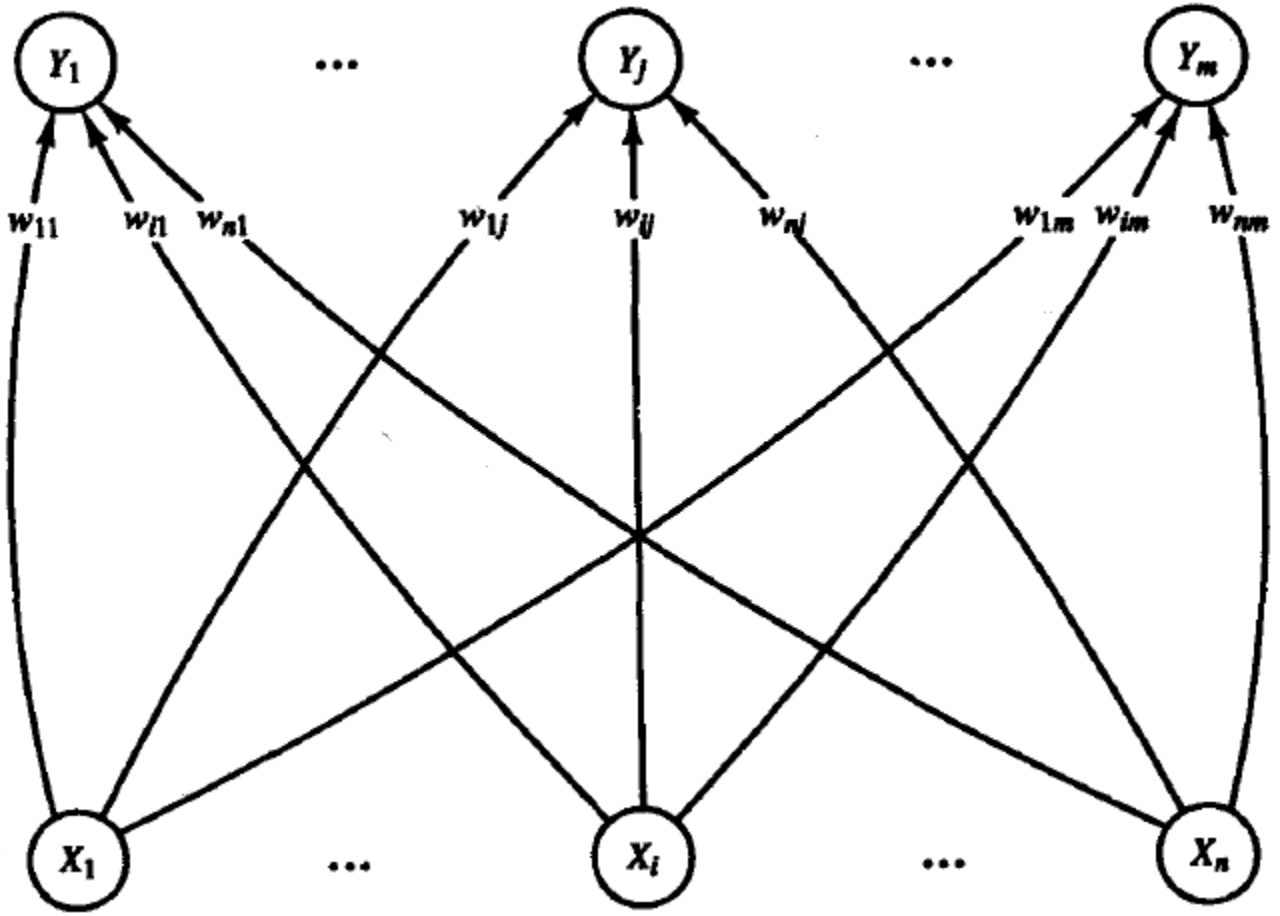
Recordando:
Tipos de redes basadas en la competición

1. Redes competitivas de pesos fijos: no utilizan un algoritmo de aprendizaje que modifique pesos. Las conexiones se definen a priori y lo que cambian son las activaciones de las neuronas:
    - Maxnet
    - Mexican hat
    - Red de Hamming
2. Redes competitivas con aprendizaje:
    - Redes de Kohonen (mapas autoorganizativos o SOM): utilizan aprendizaje no supervisado.
    - Redes LVQ (Learning Vector Quantization – aprendizaje por quantificicación vectorial): utilizan aprendizaje supervisado.

Redes LVQ (aprendizaje por cuantización vectorial)

- Se deben a Kohonen, 1989.
- En esta red cada neurona de salida representa a una clase o categoría (varias neuronas de salida se utilizan en cada clase).
- El vector de pesos de una neurona de salida actúa de referencia para la clase que representa.
- El aprendizaje es supervisado y aproxima las superficies de decisión del clasificador teórico de Bayes.
- Después del entrenamiento, la red LVQ clasifica un vector de entrada asignándolo a la clase de la neurona de salida que tiene su vector de pesos (vector de referencia) más parecido al vector de entrada.

Redes LVQ: arquitectura



- La arquitectura es la misma que las redes SOM (sin asumir estructura topológica en las neuronas de salida).
- Cada neurona de salida representa a una clase.

Redes LVQ: notación

- $\mathbf{x}$ vector de entrenamiento $(x_1,\ldots x_i,\ldots,x_n)$
- T clase o categoría objetivo del vector de entrenamiento
- $\mathbf{w}_j$ vector de pesos para la j-ésima neurona de salida $(w_{1j},\ldots w_{ij},\ldots,w_{nj})$
- C_j clase o categoría representada por la j-ésima neurona de salida.
- $\|\mathbf{x}-\mathbf{w}_j\|$ distancia euclídea entre el vector de entrada y el vector de pesos de la j-ésima neurona de salida

• El objetivo del algoritmo es encontrar la neurona de salida más próxima al vector de entrada. Si $\mathbf{x}$ y $\mathbf{w}$ pertenecen a la misma clase, entonces los pesos se modifican hacia el nuevo vector de entrada. Si pertenecen a distintas clases, los pesos se modifican para que se alejen del vector de entrada.

Redes LVQ: algoritmo

Paso 0: Inicializar los vectores de referencia (pesos w_{ij}). Inicializar el factor de aprendizaje α.

Paso 1: Mientras la condición de parada sea falsa, ejecutar los pasos 2-6:

Paso 2: Para cada vector de entrada $\mathbf{x}$, ejecutar 3-4*:*

Paso 3: Encontrar el índice J tal que $\|\mathbf{x}-\mathbf{w}_J\|$ sea mínima

Paso 4: Actualizar $\mathbf{w}_J$:

si $T = C_j$, $\mathbf{w}_J(nuevo) = \mathbf{w}_J(anterior) + \alpha[\mathbf{x}-\mathbf{w}_J(anterior)]$

si $T \neq C_j$, $\mathbf{w}_J(nuevo) = \mathbf{w}_J(anterior) - \alpha[\mathbf{x}-\mathbf{w}_J(anterior)]$

Paso 5: Reducir el factor de aprendizaje α.

Paso 6: Comprobar la condición de parada (número máximo de iteraciones o que α alcance un valor pequeño.

Redes LVQ: inicialización de pesos

Los pesos se pueden inicializar:

1. Aleatoriamente (también las clasificaciones)
2. Utilizando los m primeros vectores de entrenamiento (y empleando el resto para entrenar).
3. Utilizando otro algoritmo de clasificación como K-means o una red SOM.

Redes LVQ: ejemplo (1/3)

Se tienen 5 vectores de entrenamiento asociados a dos clases:

vector	clase
(1,1,0,0)	1
(0,0,0,1)	2
(0,0,1,1)	2
(1,0,0,0)	1
(0,1,1,0)	2

- Los primeros vectores se utilizarán para inicializar los dos vectores de referencia.
- La primera neurona de salida representa la clase 1, y la segunda la clase 2 (C_1=1 y C_2=2)
- Los otros vectores (0,0,1,1), (1,0,0,0) y (0,1,1,0) se utilizan para el entrenamiento.

Redes LVQ: ejemplo (2/3)

Paso 0: Inicializar los vectores de referencia (pesos w_{ij}):

$\mathbf{w}_1=(1,1,0,0)$; $\mathbf{w}_2=(0,0,0,1)$; y $\alpha=0.1$

Paso 1: Mientras la condición de parada sea falsa, ejecutar los pasos 2-6 (sólo se muestra la 1ª iteración):

Paso 2: Para el vector de entrada $\mathbf{x}=(0,0,1,1)$ con $T=2$:

Paso 3: $J=2$, ya que $\mathbf{x}$ es más parecido a $\mathbf{w}_2$ que a $\mathbf{w}_1$

Paso 4: Actualizar $\mathbf{w}_2$ ($T=2$ y $C_2=2$):

$$T=C_2,\quad \mathbf{w}_2(nuevo)=(0,0,0,1)+0.1[(0,0,1,1)-(0,0,0,1)]=$$
$$=(0,0,0.1,1)$$

Paso 2: Para el vector de entrada $\mathbf{x}=(1,0,0,0)$ con $T=1$:

Paso 3: $J=1$, ya que $\mathbf{x}$ es más parecido a $\mathbf{w}_1$ que a $\mathbf{w}_2$

Paso 4: Actualizar $\mathbf{w}_1$ ($T=1$ y $C_1=1$):

$$T=C_1,\quad \mathbf{w}_1(nuevo)=(1,1,0,0)+0.1[(1,0,0,0)-(1,1,0,0)]=$$
$$=(0,0.9,0,0)$$

Redes LVQ: ejemplo (3/3)

Paso 2: Para el vector de entrada $\mathbf{x}=(0,1,1,0)$ con $T=2$:

Paso 3: $J=1$, ya que $\mathbf{x}$ es más parecido a $\mathbf{w}_1$ que a $\mathbf{w}_2$

Paso 4: Actualizar $\mathbf{w}_1$ ($T=2$ y $C_1=1$):

$$T=C_1, \quad \mathbf{w}_1(nuevo)=(1,0.9,0,0)-0.1[(0,1,1,0)-(0,0.9,0,1)]=$$
$$=(1.1,0.89,-0.1,0)$$

Paso 5: Esto completa una época del entrenamiento. Se reduce la tasa de aprendizaje y el algoritmo continuaría.

Paso 6: Se comprueba la condición de parada.

Redes de contrapropagación (*counterpropagation*)

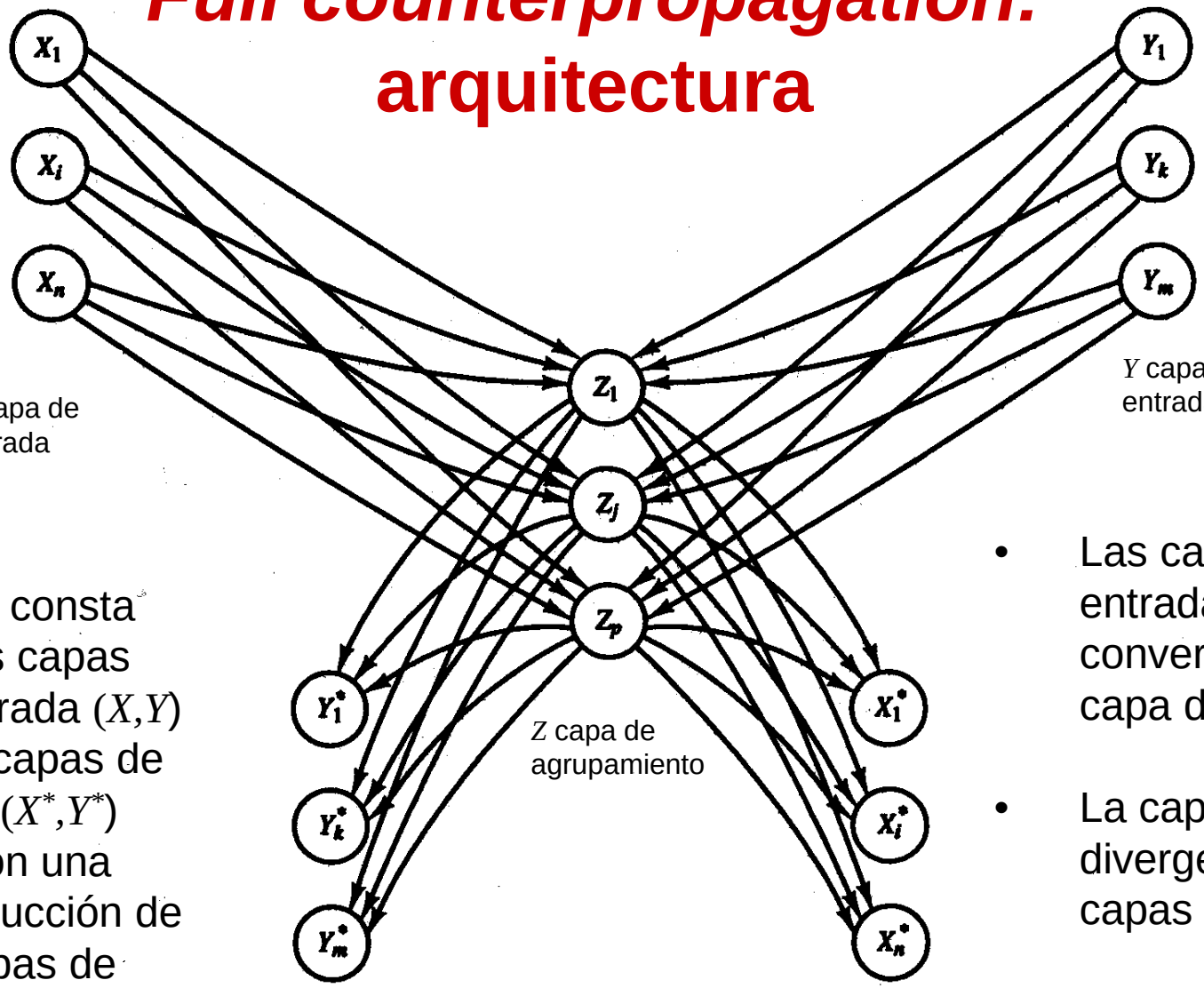
- Se deben a Hecht y Nielsen, 1987.
- Se trata de redes multicapa (capa de entrada, capa de *clustering* o agrupamiento competitivo y capa de salida).
- Se utilizan fundamentalmente para comprimir datos, aproximar funciones y asociar patrones.
- Las redes de contrapropagación aproximan vectores de entrada de entrenamiento construyendo una tabla de referencia de forma adaptativa.
- El entrenamiento tiene dos fases. Durante la primera se agrupan los vectores de entrada con la métrica del producto escalar o una métrica de norma euclídea. Durante la segunda fase, los pesos de las neuronas que forman un *cluster* se actualizan para producir la respuesta deseada.

Redes de contrapropagación (*counterpropagation*)

- Se clasifican en dos tipos:

  1. Contrapropagación completa (Full counterpropagation)
  2. Contrapropagación de flujo directo (Feedforward counterpropagation)

Full counterpropagation: arquitectura



- La red consta de dos capas de entrada (X,Y) y dos capas de salida (X^*,Y^*) que son una reproducción de las capas de entrada.

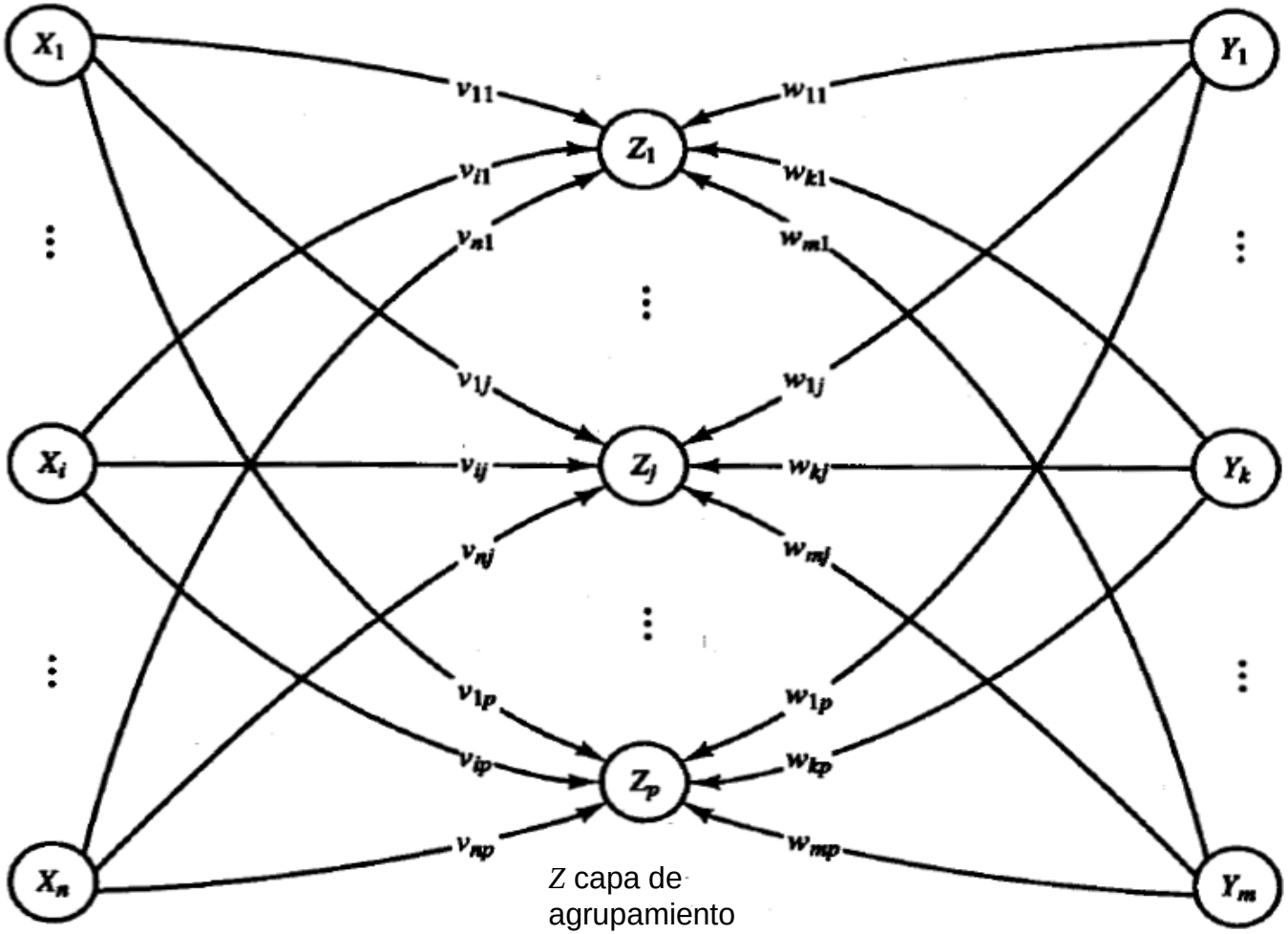
- Las capas de entrada convergen a la capa de *cluster*
- La capa de *cluster* diverge a las capas de salida

Full counterpropagation

- A partir de pares de vectores $\mathbf{x}$:$\mathbf{y}$ la red de full conterpropagation produce una aproximación $\mathbf{x}^*$:$\mathbf{y}^*$ a partir de:
    1. una entrada de un vector $\mathbf{x}$ (sin información del vector $\mathbf{y}$)
    2. Una entrada de un vector $\mathbf{y}$ (sin información del $\mathbf{x}$)
    3. Una entrada parcial del par $\mathbf{x}$:$\mathbf{y}$ (distorsionada o con elementos sin especificar de los dos vectores)

- A partir de los pares de vectores de entrenamiento $\mathbf{x}$:$\mathbf{y}$ se forman los *clusters* durante la primera etapa del entrenamiento. Como resultado de la competición en la capa de *clusters* se elige la neurona ganadora.

Full counterpropagation:
Neuronas activas durante la 1ª fase del entrenamiento

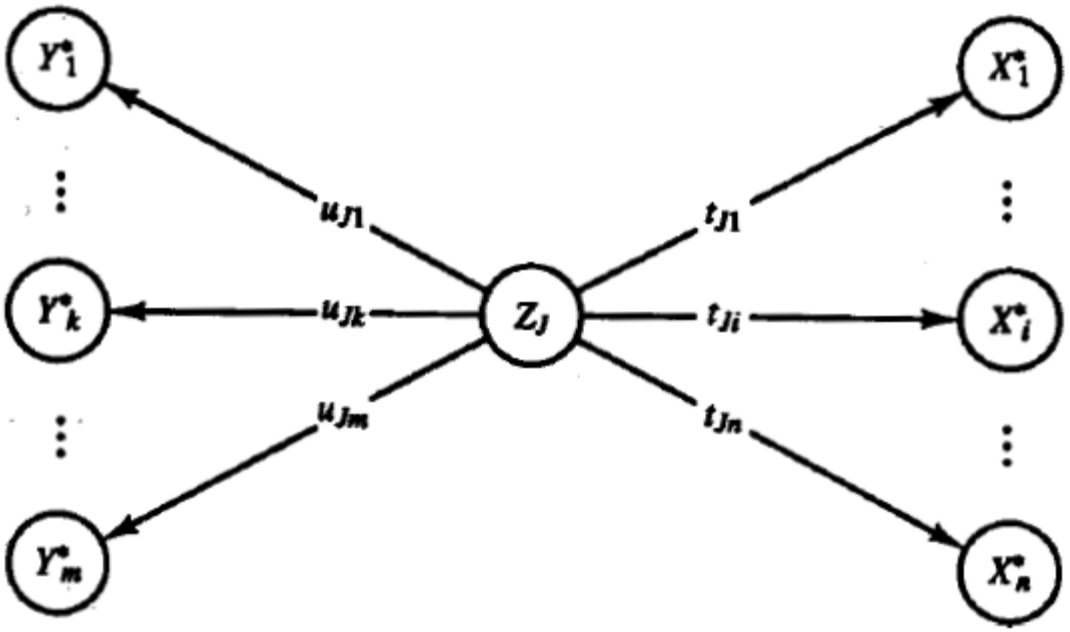


X capa de entrada

Z capa de agrupamiento

Y capa de entrada

Full counterpropagation:
2ª fase del entrenamiento



Y^* capa de salida

X^* capa de salida

Full counterpropagation: entrenamiento

- El entrenamiento ocurre en dos fases. En la primera, las neuronas de entrada X, las neuronas competitivas de *cluster* Z y las neuronas de salida Y están activas.

- En el formalismo original de *counterpropagation* sólo se permite que aprenda una única neurona J de *cluster* (la que gana la competición). La regla de aprendizaje es:

$$v_{iJ}(nuevo) = (1-\alpha)v_{iJ}(anterior) + \alpha x_i, \quad i = 1,...,n$$
$$w_{kJ}(nuevo) = (1-\beta)w_{kJ}(anterior) + \beta y_k, \quad k = 1,...,m$$

Full counterpropagation: entrenamiento

- En la segunda fase, sólo la neurona J que ha ganado la competición permanece activa en la capa de *cluster*. Los pesos de esta neurona a las neuronas de salida se ajustan de forma que el vector de activaciones de las neuronas en la capa de salida Y^* ($\mathbf{y}^*$) sean una aproximación al vector de entrada $\mathbf{y}$; $\mathbf{x}^*$ es una aproximación a $\mathbf{x}$.

- La actualización de pesos para las neuronas de las capas de salida es:

$$u_{Jk}(nuevo) = (1-a)u_{Jk}(anterior) + ay_k, \quad k = 1,...,m$$

$$t_{Ji}(nuevo) = (1-b)t_{Ji}(anterior) + bx_i, \quad i = 1,...,n$$

Full counterpropagation: notación

- $\mathbf{x}$ vector de entrada para el entrenamiento $\mathbf{x}=(x_1,\ldots x_i,\ldots,x_n)$
- $\mathbf{y}$ vector objetivo para el entrenamiento $\mathbf{y}=(y_1,\ldots y_i,\ldots,y_m)$
- z_j activación de la neurona Z_j de la capa de *clusters*
- $\mathbf{x}^*$ aproximación calculada para el vector $\mathbf{x}$
- $\mathbf{y}^*$ aproximación calculada para el vector $\mathbf{y}$
- v_{ij} pesos desde la neurona X_i de la capa X de entrada a la neurona Z_j de la capa de *clusters*.
- w_{kj} pesos desde la neurona Y_k de la capa Y de entrada a la neurona Z_j de la capa de *clusters*.
- u_{jk} pesos desde la neurona Z_j de la capa de clusters a la neurona Y_k^* de la capa de *salida* Y^*
- t_{ji} pesos desde la neurona Z_j de la capa de clusters a la neurona X_i^* de la capa de *salida* X^*
- α,β f. aprendizaje para los pesos a la capa de clusters
- a,b f. aprendizaje para los pesos desde la capa de clusters

Full counterpropagation: algoritmo de aprendizaje (1/3)

Paso 0: Inicializar los pesos y factores de aprendizaje

Paso 1: Mientras la condición de parada para la Fase I sea falsa, ejecutar los pasos 2-7:

Paso 2: Para cada par de entradas de entrenamiento **x:y**, ejecutar 3-5:

Paso 3: Establecer las activaciones de la capa X de entrada al vector $\mathbf{x}$

Establecer las activaciones de la capa Y de entrada al vector $\mathbf{y}$

Paso 4: Encontrar la neurona ganadora (índice J)

Paso 5: Actualizar los pesos para la neurona Z_J:

$$v_{iJ}(nuevo) = (1-\alpha)v_{iJ}(anterior) + \alpha x_i, \quad i = 1,...,n$$

$$w_{kJ}(nuevo) = (1-\beta)w_{kJ}(anterior) + \beta y_k, \quad k = 1,...,m$$

Full counterpropagation: algoritmo de aprendizaje (2/3)

Paso 6: Reducir los factores de aprendizaje α y β.

Paso 7: Comprobar la condición de parada para la Fase I

Paso 8: Mientras la condición de parada para la Fase 2 sea falsa, ejecutar los pasos 9-15 (α y β son valores constantes y pequeños en esta fase):

Paso 9: Para cada par de entradas de entrenamiento **x:y**, ejecutar 10-13:

Paso 10: Establecer las activaciones de la capa X de entrada al vector $\mathbf{x}$

Establecer las activaciones de la capa Y de entrada al vector $\mathbf{y}$

Paso 11: Encontrar la neurona ganadora (índice J)

Full counterpropagation: algoritmo de aprendizaje (3/3)

Paso 12: Actualizar los pesos a la neurona Z_J:

$$v_{iJ}(nuevo) = (1-\alpha)v_{iJ}(anterior) + \alpha x_i, \quad i = 1,...,n$$

$$w_{kJ}(nuevo) = (1-\beta)w_{kJ}(anterior) + \beta y_k, \quad k = 1,...,m$$

Paso 13: Actualizar los pesos de la neurona Z_J a las capas de salida:

$$u_{Jk}(nuevo) = (1-a)u_{Jk}(anterior) + a y_k, \quad k = 1,...,m$$

$$t_{Ji}(nuevo) = (1-b)t_{Ji}(anterior) + b x_i, \quad i = 1,...,n$$

Paso 14: Reducir el factor de aprendizaje

Paso 15: Comprobar la condición de parada para la Fase 2

Full counterpropagation: Neuronas ganadoras

- En los pasos 4 y 11, para decidir qué neurona gana en la capa de *cluster*, se puede utilizar la métrica euclídea: la neurona que gana Z_j es la que tiene la distancia a los vectores de entrada más pequeña:

$$D_j = \sum_i (x_i - v_{ij})^2 + \sum_k (y_k - w_{kj})^2$$

- En caso de empate, se toma la neurona con menor índice.

Full counterpropagation: explotación

Paso 0: Inicializar los pesos

Paso 1: Para cada par de entradas de **x:y**, ejecutar 2-4*:*

Paso 2: Establecer las activaciones de la capa X de entrada al vector **x**

Establecer las activaciones de la capa Y de entrada al vector **y**

Paso 3: Encontrar la neurona de la capa de *cluste*r Z_J más próxima al par de entrada **x** e **y**

Paso 4: calcular las aproximaciones a **x** e **y** :

$$x_i^* = t_{Ji}$$

$$y_k^* = t_{Jk}$$

Full counterpropagation: ejemplo

- Tabla de referencia para la función $y=1/x$ en el intervalo [0.1,10.0]

- Utilizaremos 1 neurona en la capa de entrada X, 1 neurona en la capa de entrada Y, 1 neurona en la capa de salida X^* y 1 neurona en la capa de salida Y^*

- Suponemos que hay disponibles 1000 patrones de entrenamiento (valores de x entre 0.1 y 10.0 y los correspondientes valores de y dados por $1/y$. Los patrones de entrenamiento de entrada se presentan de forma aleatoria.
- Suponemos 10 neuronas en la capa de *cluster*

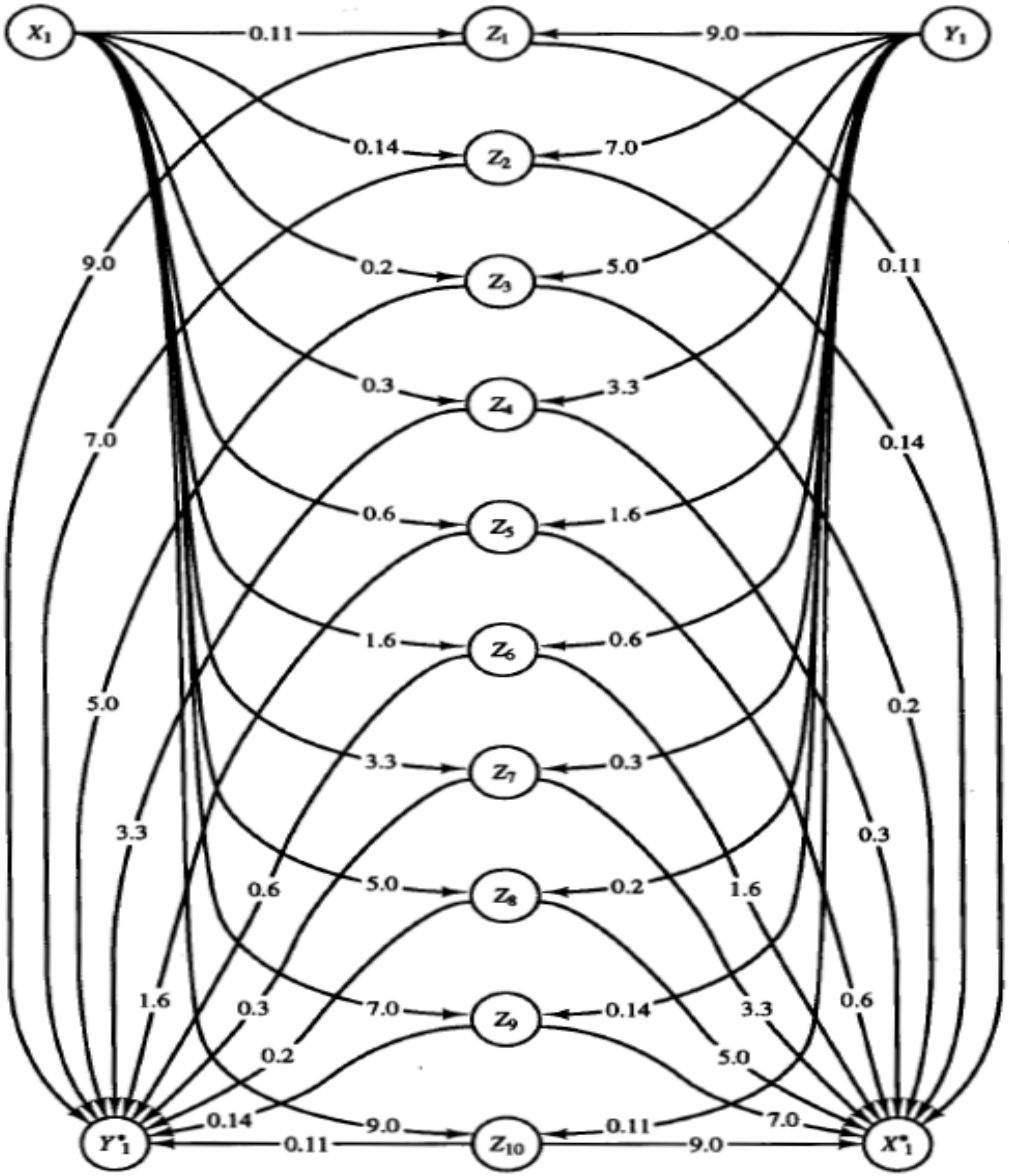
Full counterpropagation: ejemplo

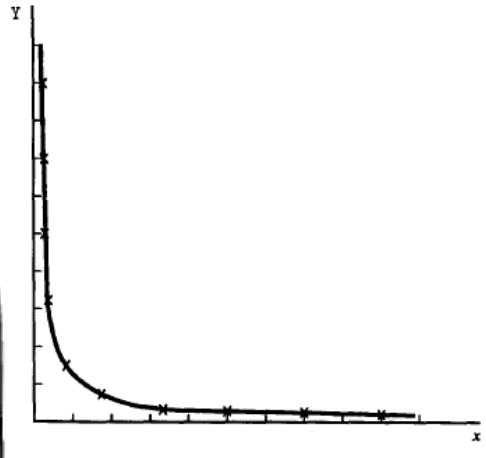
- Después de la primera fase de entrenamiento los pesos a las neuronas de las capas de entrada X e Y son ($\mathbf{v}$,$\mathbf{w}$ respectivamente):

CLUSTER UNIT	v	w
Z_1	0.11	9.0
Z_2	0.14	7.0
Z_3	0.20	5.0
Z_4	0.30	3.3
Z_5	0.6	1.6
Z_6	1.6	0.6
Z_7	3.3	0.30
Z_8	5.0	0.20
Z_9	7.0	0.14
Z_{10}	9.0	0.11

Full counterpropagation: ejemplo

- Después de la segunda fase de entrenamiento los pesos quedan:





Full counterpropagation: ejemplo

Se puede utilizar la red para aproximar el valor de y con x=0.12

Paso 0: Inicializar los pesos

Paso 1: Para x=0.12 e y=0, ejecutar 2-4*:*

Paso 2: Establecer las activaciones de la capa X de entrada al vector $\mathbf{x}$

Establecer las activaciones de la capa Y de entrada al vector $\mathbf{y}$

Paso 3: Encontrar la neurona ganadora del *cluste*r Z_J

$D_1 = (0.12 - 0.11)^2 + (0.00 - 9.00)2 = 81$

$D_2 = (0.12 - 0.14)^2 + (0.00 - 7.00)2 = 49$

$D_3 = (0.12 - 0.20)^2 + (0.00 - 5.00)2 = 25$

$D_4 = (0.12 - 0.30)^2 + (0.00 - 3.30)2 = 11$

$D_5 = (0.12 - 0.60)^2 + (0.00 - 1.60)^2 = 2.8$

$D_6 = (0.12 - 1.60)^2 + (0.00 - 0.60)^2 = 2.6$

$D_7 = (0.12 - 3.30)^2 + (0.00 - 0.30)^2 = 10.2$

$D_8 = (0.12 - 5.00)^2 + (0.00 - 0.20)^2 = 23.9$

$D_9 = (0.12 - 7.00)^2 + (0.00 - 0.14)^2 = 47.4$

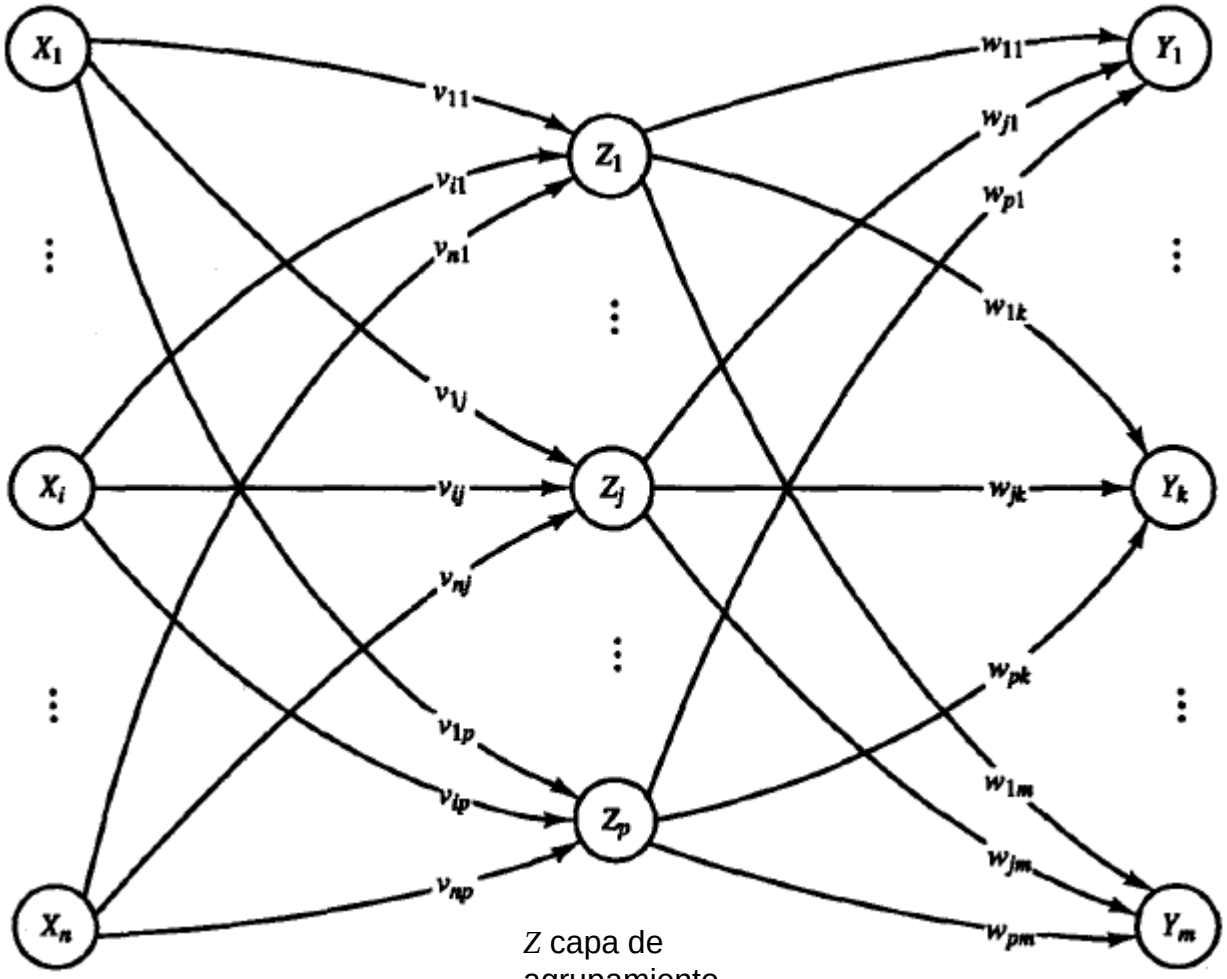
$D_{10} = (0.12 - 9.00)^2 + (0.00 - 0.11)^2 = 78.9$

por tanto J=6. Ejercicio: calcular x* e y*. Repetir calculando distancias solo con respecto a la entrada x

Forward-only counterpropagation

- Se trata de una versión simplificada de la red de full *counterpropagation*
- Produce una aproximación a la función $\mathbf{y}=f(\mathbf{x})$ (se puede utilizar si se conoce la relación de $\mathbf{x}$ con $\mathbf{y}$, pero no la relación de $\mathbf{y}$ a $\mathbf{x}$)
- Los *clusters* se forman utilizando sólo los vectores $\mathbf{x}$ durante la primera etapa del entrenamiento.
- La arquitectura se parece a las redes de retropropagación, pero existen conexiones en la capa de *cluster.*

Forward-only counterpropagation: arquitectura



X capa de entrada

Z capa de agrupamiento

Y capa de salida

Forward-only counterpropagation: entrenamiento

- Primero se presenta el vector de entrada a las neuronas de la capa de entrada. Las neuronas de la capa de *cluster* compiten para aprenderse este vector.
- Después (la tasa de aprendizaje se ha reducido) se entrenan los pesos de la capa de *cluster* a la capa de salida (presentando el vector objetivo en la capa de salida). La neurona ganadora del *cluster* (J) envía un 1 a la capa de salida.
- Utilizando la diferencia entre la señal de entrada w_{Jk} a la neurona k de salida y el valor objetivo y_k, se actualizan los pesos de la neurona ganadora a la capa de salida.

Forward-only counterpropagation: entrenamiento

- La regla de aprendizaje de la capa de entrada a las neuronas de cluster es:

$$v_{iJ}(nuevo) = (1-\alpha)v_{iJ}(anterior) + \alpha x_i, \quad i = 1,...,n$$

- La regla de aprendizaje de las neuronas de cluster a la capa de salida es:

$$w_{Jk}(nuevo) = (1-a)w_{Jk}(anterior) + a y_k, \quad k = 1,...,m$$

Forward-only counterpropagation: algoritmo de aprendizaje (1/3)

Paso 0: Inicializar los pesos y factores de aprendizaje

Paso 1: Mientras la condición de parada para la Fase I sea falsa, ejecutar los pasos 2-7:

Paso 2: Para cada entrada de entrenamiento $\mathbf{x}$

ejecutar 3-5:

Paso 3: Establecer las activaciones de la capa X de entrada al vector $\mathbf{x}$

Paso 4: Encontrar la neurona ganadora (índice J)

Paso 5: Actualizar los pesos para la neurona Z_J:

$$v_{iJ}(nuevo) = (1-\alpha)v_{iJ}(anterior) + \alpha x_i, \quad i = 1,...,n$$

Paso 6: Reducir α.

Paso 7: Comprobar la condición de parada para la Fase I

Forward-only counterpropagation: algoritmo de aprendizaje (2/3)

Paso 8: Mientras la condición de parada para la Fase 2 sea falsa, ejecutar los pasos 9-15 (α es cte y pequeño):

Paso 9: Para cada par de entrenamiento **x:y**

ejecutar 10-13:

Paso 10: Establecer las activaciones de la capa X de entrada al vector **x**

Establecer las activaciones de la capa Y de salida al vector **y**

Paso 11: Encontrar la neurona ganadora (índice J)

Paso 12: Actualizar los pesos a la neurona Z_J:

$$v_{iJ}(nuevo) = (1-\alpha)v_{iJ}(anterior) + \alpha x_i, \quad i = 1,...,n$$

Paso 13: Actualizar los pesos de la neurona Z_J a salida

$$w_{Jk}(nuevo) = (1-a)w_{Jk}(anterior) + \alpha y_k, \quad k = 1,...,m$$

Forward-only counterpropagation: algoritmo de aprendizaje (3/3)

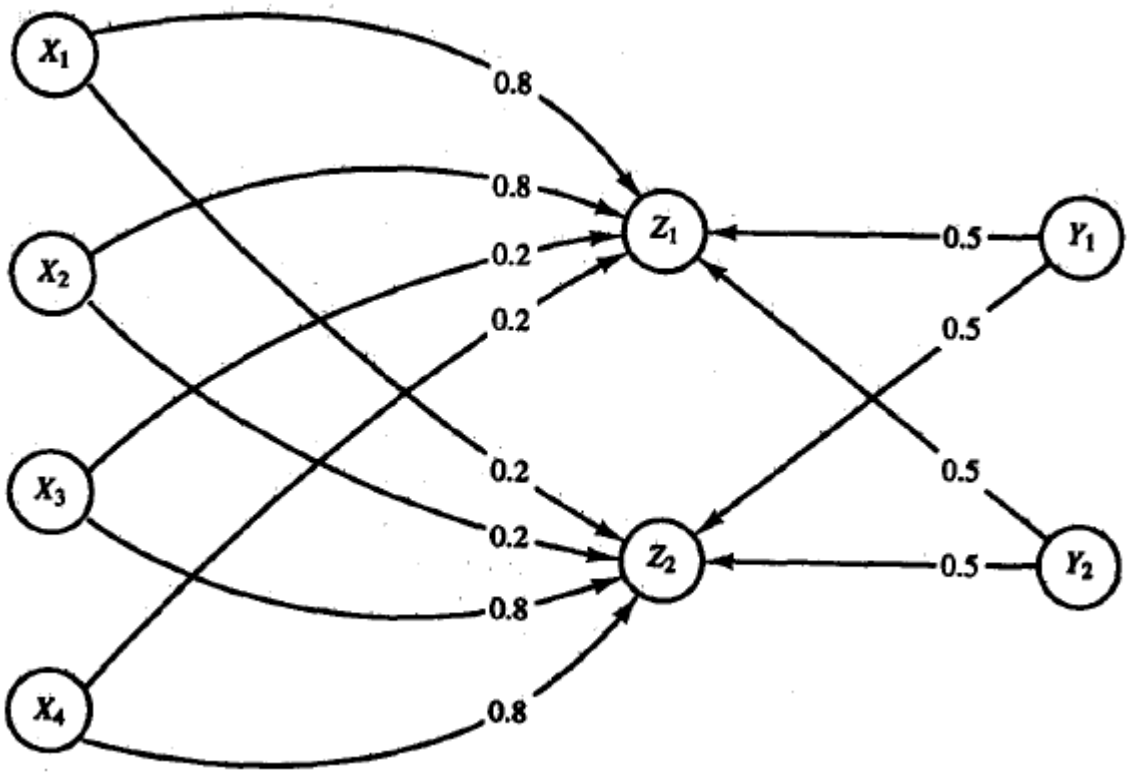
Paso 14: Reducir a.

Paso 15: Comprobar la condición de parada para la Fase II

EJERCICIO

Con la siguiente arquitectura, ejecutar la primera fase de entrenamiento para una red *full conterpropagation* con los siguientes pares de entradas:

x=(1,0,0,0,) **y**=(1,0)
x=(1,0,1,1) **y**=(0,1)



Redes ART:
ADAPTIVE RESONANCE THEORY

- Se deben a Carpenter y Grossberg, 1987.
- Agrupan entradas (*clustering*) con aprendizaje no-supervisado
- Las entradas se pueden presentar en cualquier orden.
- Cuando se presenta un patrón, se selecciona una neurona del *cluster* y se ajustan los pesos del *cluster* para que esa neurona aprenda el patrón.
- Permiten que el usuario de la red controle el grado de similitud de los patrones que están en el mismo *cluster*.
- Se utiliza la similitud relativa de un patrón de entrada respecto al vector de pesos de la neurona del *cluster* (una diferencia de un componente es más significativa en patrones que tienen pocos componentes distinto de cero, respecto a los que tienen muchos).
- Son redes plásticas (con capacidad de aprender bien un patrón en cualquier estado del aprendizaje).

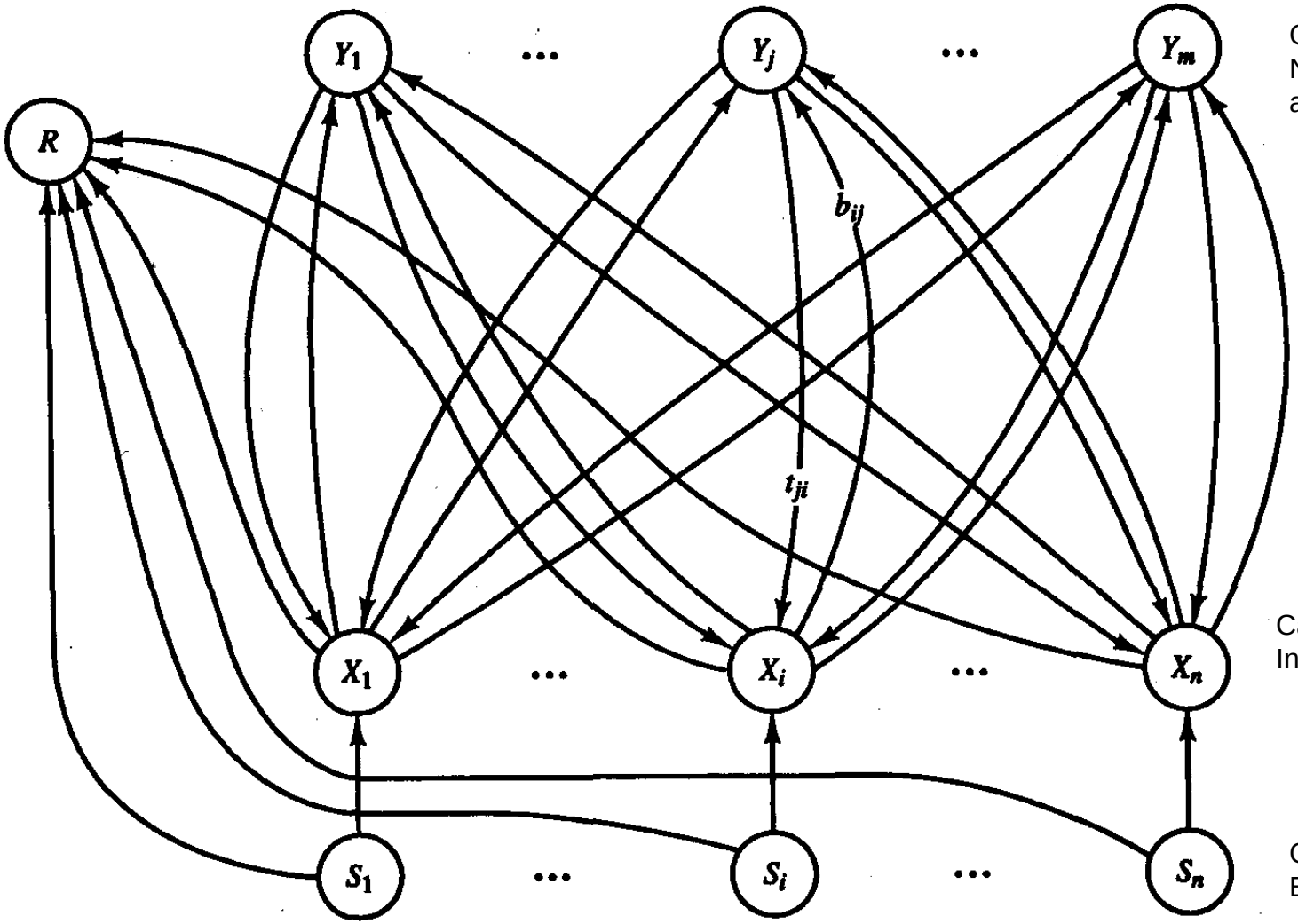
Redes ART: Arquitectura

- La arquitectura de una red ART se compone de tres grupos de neuronas:
  1. Una capa de entrada (capa F_1) dividida en una subcapa de entrada $F_1(\mathrm{a})$ y otra de interfaz $F_1(\mathrm{b})$ (que combina las señales de la subcapa de entrada y la capa de *cluster* F_2 para comparar la similitud de la señal de entrada y el vector de pesos para la neurona de *cluster* que se ha seleccionado para aprender).
  2. Una capa de *cluster* (capa F_2)
  3. Una neurona que implementa un mecanismo para controlar el grado de similitud de patrones localizados en el mismo *cluster* (mecanismo de reseteo).
- La capa de *cluster* F_2 es una capa competitiva: la unidad con mayor entrada neta es la candidata a aprender el patrón de entrada. La activación del resto de neuronas se hace 0.

Redes ART: Arquitectura

- Las neuronas de la capa interfaz $F_1(\mathrm{b})$ combinan la información de las neuronas de entrada y las de *cluster*. El que se permita que la neurona de *cluster* aprenda el patrón de entrada depende de la similitud entre el vector de pesos de la conexión de la neurona de la capa F_2 a la capa $F_1(\mathrm{b})$ y el vector de entrada.
- Esta decisión la toma la neurona de reseteo, utilizando las señales que recibe de la entrada (a) y del interfaz (b) de la capa de entrada.
- Si no se permite que la neurona de *cluster* aprenda, se inhibe y se selecciona otra neurona de *cluster* como candidata.

Redes ART: Arquitectura



Capa F_2
Neuronas de agrupamiento

Pesos b: bottom-up

t: top-down

Capa F_1(b)
Interfaz

Capa F_1(a)
Entrada

La neurona R está conectada también a toda la capa F_2

Redes ART: conceptos

- Una etapa de entrenamiento en ART consiste en la presentación de un patrón de entrada.
- Antes de presentar un patrón de entrada, las activaciones de todas las neurona de la red se establecen a cero. Todas las neuronas de la capa de *cluster* que se han inhibido en la etapa de aprendizaje anterior vuelven a competir
- Una vez que se presenta un patrón, continua enviando su señal de entrada hasta que la etapa de aprendizaje termina.
- El grado de similitud para que dos patrones se asignen a la misma neurona de *cluster* se controla con un parámetro conocido con el nombre de *vigilancia.*
- *Una neurona de la capa* F_2 (cluster) *puede estar:*
  1. *Activa:* d=1
  2. *Inactiva:* d=0 *pero pudiendo participar en la competición*
  3. *Inhibida:* d=0 *sin participar en la competición posterior*

Redes ART: notación

- n número de componentes del vector de entrada
- m máximo número de *clusters* que se pueden formar
- b_{ij} pesos bottom-up (de la neurona X_i de la capa $F_1(\mathrm{b})$ a la neurona Y_j de la capa F_2.
- t_{ji} pesos top-down (de la neurona Y_j de la capa F_2 a la X_i de la capa $F_1(\mathrm{b})$).
- ρ parámetro de vigilancia
- $\mathbf{s}$ vector de entrada (binario ART1, en ART2 contínuo)
- $\mathbf{x}$ vector de activaciones para la capa $F_1(\mathrm{b})$ (binario).
- $\| \mathbf{x} \|$ norma del vector $\mathbf{x}$ definida como la suma de componentes x_i

Red ART: algoritmo de aprendizaje (1/2)

Paso 0: Inicializar los parámetros y pesos:

$$L > 1, \quad 0 < \rho \leq 1, \quad 0 < b_{ij} < \frac{L}{L-1+n}, \quad t_{ji} = 1$$

Paso 1: Mientras la condición de parada sea falsa, ejecutar los pasos 2-13:

Paso 2: Para cada entrada de entrenamiento ejecutar 3-12

Paso 3: Establecer las activaciones de F_2 a cero

Establecer las activaciones de $F_1(a)$ al vector de entrada $\mathbf{s}$

Paso 4: calcular la norma de $\mathbf{s}$: $\|\mathbf{s}\| = \Sigma_i s_i$

Paso 5: Enviar la señal de entrada de $F_1(a)$ a $F_1(b)$: $x_i = s_i$

Paso 6: Para cada neurona F_2 que no está inhibida:

$$\text{Si } y_j \neq -1, \quad y_j = \sum_i b_{ij} x_i$$

Red ART: algoritmo de aprendizaje (2/2)

Paso 7: Mientras que el reseteo sea verdad, 8-11:

Paso 8: Encontrar J tal que $y_J \geq y_j$ para todas las j

Si y_J=-1, todas las neuronas están inhibidas y no se puede agrupar este patrón.

Paso 9: establecer la activación $\mathbf{x}$ de $F_1(b)$: $x_i = s_i t_{Ji}$

Paso 10: calcular la norma de $\mathbf{x}$: $\|\mathbf{x}\| = \Sigma_i x_i$

Paso 11: Comprobar el reseteo:

$$\text{Si } \frac{\|\mathrm{x}\|}{\|\mathrm{s}\|} < \rho, \quad y_J = -1 \text{ (se inhibe el nodo } J \text{ (y se continua con el paso 7)}$$

$$\text{Si } \frac{\|\mathrm{x}\|}{\|\mathrm{s}\|} \geq \rho, \text{ se salta al paso } 12$$

Paso 12: actualizar los pesos para la neurona J:

$$b_{ij}(nuevo) = \frac{Lx_i}{L - 1 + \|\mathbf{x}\|}, \quad t_{Ji}(nuevo) = x_i$$

Paso 13: comprobar parada

Red ART: ejemplo (1/12)

Agrupar los vectores $(1,1,0,0)$, $(0,0,0,1)$, $(1,0,0,0)$, $(0,0,1,1)$

Paso 0: Inicializar los parámetros y pesos:

$$L = 2, \quad \rho = 0.4, \quad b_{ij} = 0.2, \quad t_{ji} = 1$$

Paso 1: Mientras la condición de parada sea falsa, ejecutar los pasos 2-13:

Paso 2: Para el primer vector $(1,1,0,0)$, ejecutar 3-12

Paso 3: Establecer las activaciones de F_2 a cero

Establecer las activaciones de $F_1(a)$ al vector de entrada $\mathbf{s}=(1,1,0,0)$

Paso 4: calcular la norma de $\mathbf{s}$: $\|\mathbf{s}\|=\Sigma_i s_i=2$

Paso 5: establecer las activaciones para cada neurona de $F_1(b)$: $\mathbf{x}=(1,1,0,0)$

Red ART: ejemplo (2/12)

Paso 6: Calcular la entrada neta a cada nodo de F_2:

$$y_1 = .2(1) + .2(1) + .2(0) + .2(0) = 0.4,$$

$$y_2 = .2(1) + .2(1) + .2(0) + .2(0) = 0.4,$$

$$y_3 = .2(1) + .2(1) + .2(0) + .2(0) = 0.4.$$

Paso 7: Mientras que el reseteo sea verdad, 8-11:

Paso 8: Encontrar J tal que $y_J \geq y_j$ para todas las j

Como todas las neuronas tienen la misma entrada J=1.

Paso 9: restablecer la activación **x** de $F_1(b)$: $x_i = s_i t_{Ji}$

$x_i = s_i t_{1i}$ *con* t_1=(1,1,1,1), luego **x**=(1,1,0,0)

Paso 10: calcular la norma de **x:** $\|\mathbf{x}\| = \Sigma_i x_i = 2$

Paso 11: Comprobar el reseteo:

$$\text{como } \frac{\|\mathrm{x}\|}{\|\mathrm{s}\|} = 1 \geq \rho = 0.4, \text{ el reseteo es falso y se salta al paso } 12$$

Red ART: ejemplo (3/12)

Paso 12: actualizar los pesos para la neurona J:

$$b_{i1}(nuevo) = \frac{2x_i}{1+\|\mathbf{x}\|}, \text{ y la matriz de pesos bottom-up: } \begin{pmatrix} .67 & .2 & .2 \\ .67 & .2 & .2 \\ 0 & .2 & .2 \\ 0 & .2 & .2 \end{pmatrix}$$

$$t_{Ji}(nuevo) = x_i, \text{ y la matriz de pesos top-down: } \begin{pmatrix} 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{pmatrix}$$

Red ART: ejemplo (4/12)

Agrupar los vectores $(1,1,0,0)$, $(0,0,0,1)$, $(1,0,0,0)$, $(0,0,1,1)$

Paso 2: Para el segundo vector $(0,0,0,1)$, ejecutar 3-12

Paso 3: Establecer las activaciones de F_2 a cero

Establecer las activaciones de $F_1(a)$ al vector de entrada $\mathbf{s}=(0,0,0,1)$

Paso 4: calcular la norma de $\mathbf{s}$: $\|\mathbf{s}\|=\Sigma_i s_i=1$

Paso 5: establecer las activaciones para cada neurona de $F_1(b)$: $\mathbf{x}=(0,0,0,1)$

Red ART: ejemplo (5/12)

Paso 6: Calcular la entrada neta a cada nodo de F_2:

$$y_1 = .67(0) + .67(0) + 0(0) + 0(1) = 0.0,$$

$$y_2 = .2(0) + .2(0) + .2(0) + .2(1) = 0.2,$$

$$y_3 = .2(0) + .2(0) + .2(0) + .2(1) = 0.2.$$

Paso 7: Mientras que el reseteo sea verdad, 8-11:

Paso 8: Encontrar J tal que $y_J \geq y_j$ para todas las j

Como Y_2 e Y_3 tienen la misma entrada J=2.

Paso 9: restablecer la activación **x** de $F_1(b)$: $x_i = s_i t_{Ji}$

$x_i = s_i t_{2i}$ *con* t_2=(1,1,1,1), luego **x**=(0,0,0,1)

Paso 10: calcular la norma de **x:** $\|\mathbf{x}\| = \Sigma_i x_i = 1$

Paso 11: Comprobar el reseteo:

$$\text{como } \frac{\|\mathrm{x}\|}{\|\mathrm{s}\|} = 1 \geq \rho = 0.4, \text{ el reseteo es falso y se salta al paso } 12$$

Red ART: ejemplo (6/12)

Paso 12: actualizar los pesos para la neurona J:

$$\text{actualizar } \mathbf{b}_2\text{; la matriz de pesos bottom - up: } \begin{pmatrix} .67 & 0 & .2 \\ .67 & 0 & .2 \\ 0 & 0 & .2 \\ 0 & 1 & .2 \end{pmatrix}$$

$$\text{actualizar } \mathbf{t}_2\text{; y la matriz de pesos top - down: } \begin{pmatrix} 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 \end{pmatrix}$$

Red ART: ejemplo (7/12)

Agrupar los vectores $(1,1,0,0)$, $(0,0,0,1)$, $(1,0,0,0)$, $(0,0,1,1)$

Paso 2: Para el tercer vector $(1,0,0,0)$, ejecutar 3-12

Paso 3: Establecer las activaciones de F_2 a cero

Establecer las activaciones de $F_1(a)$ al vector de entrada $\mathbf{s}=(1,0,0,0)$

Paso 4: calcular la norma de $\mathbf{s}$: $\|\mathbf{s}\|=\Sigma_i s_i=1$

Paso 5: establecer las activaciones para cada neurona de $F_1(b)$: $\mathbf{x}=(1,0,0,0)$

Red ART: ejemplo (8/12)

Paso 6: Calcular la entrada neta a cada nodo de F_2:

$$y_1 = .67(1) + .67(0) + 0(0) + 0(0) = 0.67,$$

$$y_2 = 0(1) + 0(0) + 0(0) + 1(0) = 0.0,$$

$$y_3 = .2(1) + .2(0) + .2(0) + .2(0) = 0.2.$$

Paso 7: Mientras que el reseteo sea verdad, 8-11:

Paso 8: Encontrar J tal que $y_J \geq y_j$ para todas las j

Como Y_1 tiene la mayor entrada neta J=1.

Paso 9: restablecer la activación **x** de $F_1(b)$: $x_i = s_i t_{Ji}$

$x_i = s_i t_{1i}$ *con* t_1=(1,1,0,0), luego **x**=(1,0,0,0)

Paso 10: calcular la norma de **x:** $\|\mathbf{x}\| = \Sigma_i x_i = 1$

Paso 11: Comprobar el reseteo:

$$\text{como } \frac{\|\mathrm{x}\|}{\|\mathrm{s}\|} = 1 \geq \rho = 0.4, \text{ el reseteo es falso y se salta al paso } 12$$

Red ART: ejemplo (9/12)

Paso 12: actualizar los pesos para la neurona J:

$$\text{actualizar } \mathbf{b}_1;\ \text{la matriz de pesos bottom - up}: \begin{pmatrix} 1 & 0 & .2 \\ 0 & 0 & .2 \\ 0 & 0 & .2 \\ 0 & 1 & .2 \end{pmatrix}$$

$$\text{actualizar } \mathbf{t}_1;\ \text{y la matriz de pesos top - down}: \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 \end{pmatrix}$$

Red ART: ejemplo (10/12)

Agrupar los vectores $(1,1,0,0)$, $(0,0,0,1)$, $(1,0,0,0)$, $(0,0,1,1)$

Paso 2: Para el cuarto vector $(0,0,1,1)$, ejecutar 3-12

Paso 3: Establecer las activaciones de F_2 a cero

Establecer las activaciones de $F_1(a)$ al vector de entrada $\mathbf{s}=(0,0,1,1)$

Paso 4: calcular la norma de $\mathbf{s}$: $\|\mathbf{s}\|=\Sigma_i s_i=2$

Paso 5: establecer las activaciones para cada neurona de $F_1(b)$: $\mathbf{x}=(0,0,1,1)$

Red ART: ejemplo (11/12)

Paso 6: Calcular la entrada neta a cada nodo de F_2:

$$y_1 = 1(0) + 0(0) + 0(1) + 0(1) = 0.0,$$

$$y_2 = 0(0) + 0(0) + 0(1) + 1(1) = 1.0,$$

$$y_3 = .2(0) + .2(0) + .2(1) + .2(1) = 0.4.$$

Paso 7: Mientras que el reseteo sea verdad, 8-11:

Paso 8: Encontrar J tal que $y_J \geq y_j$ para todas las j

Como Y_2 tiene la mayor entrada neta J=2.

Paso 9: restablecer la activación **x** de $F_1(b)$: $x_i = s_i t_{Ji}$

$x_i = s_i t_{2i}$ *con* t_2=(0,0,0,1), luego **x**=(0,0,0,1*)*

Paso 10: calcular la norma de **x:** $\|\mathbf{x}\| = \Sigma_i x_i = 1$

Paso 11: Comprobar el reseteo:

$$\text{como } \frac{\| \mathrm{x} \|}{\| \mathrm{s} \|} = 0.5 \geq \rho = 0.4, \text{ el reseteo es falso y se salta al paso } 12$$

Red ART: ejemplo (12/12)

Paso 12: actualizar los pesos para la neurona J:

$$\text{actualizar } \mathbf{b}_1;\ \text{la matriz de pesos bottom - up no cambia}: \begin{pmatrix} 1 & 0 & .2 \\ 0 & 0 & .2 \\ 0 & 0 & .2 \\ 0 & 1 & .2 \end{pmatrix}$$

$$\text{actualizar } \mathbf{t}_1;\ \text{y la matriz de pesos top - down no cambia}: \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 \end{pmatrix}$$

Paso 13: comprobar parada (termina una época). No hay cambios si se presentan otra vez estos vectores.