

TEMA 6 Redes de Neuronas Artificiales

Francisco José Ribadas Pena

INTELIGENCIA ARTIFICIAL

5º Informática

ribadas@uvigo.es

29 de noviembre de 2005

6.1 Introducción

- **Objetivo:** Usar los principios de organización de los cerebros biológicos para construir sistemas inteligentes. (*IA subsimbólica*)
 - RNA → Emulación (modelo matemático) del funcionamiento del cerebro a bajo nivel.
 - IA simbólica → Simula el comportamiento inteligente (interesa el resultado inteligente)
- **Cerebro/RNAs:** Sistemas masivamente paralelos formados por un gran número de elementos de procesos (EPs) simples (neuronas) interconectados

	CEREBRO	ORDENADOR
EPs	Muchos EPs simples (10^{11} neuronas + 6×10^{14} conex.) Baja velocidad proceso (ms.)	Pocos (1..miles) EPs muy complejos y potentes Alta velocidad proceso (μs)
Memoria	Distribuida entre EPs Direccionable por contenido	Separada de EPs No direcc. por contenido
Procesam.	Altamente distribuido y paralelo Puede aprender/mejorar Alta tolerancia fallos (pérdida neuronas no afecta)	Programas secuenciales y centralizados Proceso fijo y precodificado Poco robusto ante fallos
Aplicaciones	Percepción y predicción	Proceso numérico y simbólico.

- Aplicaciones típicas.
 - En dominios difíciles de formalizar (necesidad aprendizaje)
 - Entradas/salidas muchas dimensiones. Entradas con ruido.
 - Tareas clasificación/reconocimiento de patrones.
 - Comprensión por humanos poco importante.
 - Percepción: {
 - reconocim./generación de voz
 - reconocim. formas (OCR, ...)
 - identificación personas (voz, huellas, iris,...)
 - Predicción: Predicción series temporales: ciclos consumo energía, valores bursátiles, ...)
 - Otras: clasificación, aproximación funciones, filtrado adaptativo de señales

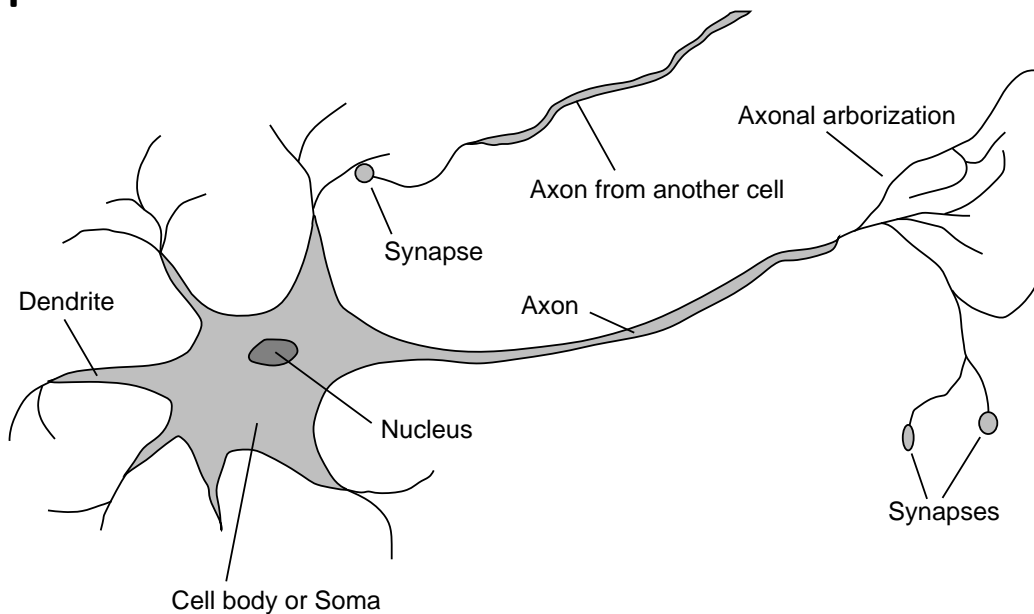
6.2 Neuronas Biológicas y Artificiales

(a) Fundamentos Biológicos

- **Neurona biológica:** Procesador de información muy simple basado en procesos electroquímicos.

10^{11} neuronas, con miles de conex. de entrada y cientos de salida (6×10^{14} conex.)

- **Componentes:**



- **SOMA:** Cuerpo de la célula
 - Realiza el "procesamiento"
- **AXON:** Elemento de salida con múltiples ramificaciones.
 - Transporta impulsos nerviosos a otras neuronas
- **DENDRITAS:** Elementos de entrada
 - Reciben señales de excitación/inhibición de otras neuronas a través de las sinapsis
- **SINAPSIS:** Áreas de contacto entre neuronas
 - Conexiones unidireccionales, dos tipos

{	excitadoras
	inhibidoras
 - No hay contacto físico, (separación)
 - Transmisión de info. en forma electroquímica (iones +/-), gobernada por *neurotransmisores*

■ **Funcionamiento**

- Neurona (soma) "acumula" todos los potenciales positivos y negativos que recibe en sus entradas (dendritas)
- Si la suma de esos impulsos es "suficiente", cambia su potencia y genera su salida en el axón que se propagará a otras neuronas.

■ **Aprendizaje**

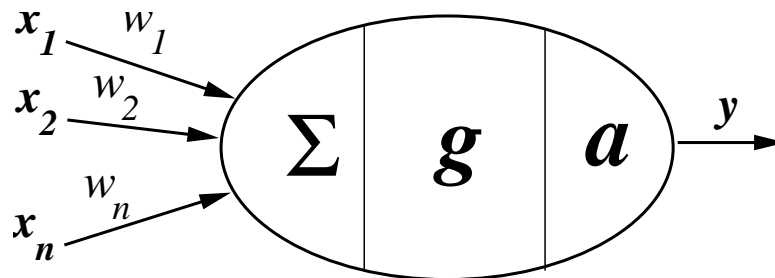
- La intensidad de las sinapsis (conexiones) puede modificarse.
 - Conexiones más o menos fuertes.
 - Permite modificar comportamiento de la neurona para adaptarse a nuevas situaciones (aprendizaje)
- También es posible modificar el comportamiento mediante creación/destrucción de sinapsis y por la muerte de neuronas

(b) Redes de Neuronas Artificiales

- **Objetivo:** Emular funcionamiento de neuronas biológicas
- Una RNA está formada por un conjunto de EPS (neuronas artificiales) unidas por conexiones unidireccionales ponderadas (con un peso asociado)
 - Normalmente se organizan en capas.
- El procesamiento en cada EP es local
 - Depende sólo de $\left\{ \begin{array}{l} \text{entradas} + \text{pesos conexión} \\ \text{estado anterior del EP (opcional)} \end{array} \right.$
- La red se adapta mediante un aprendizaje que modifica los pesos de las conexiones.
 - El conocimiento se almacena en los pesos sinápticos.
- Elementos clave de las RNAs:
 - Procesamiento paralelo: EPs/neuronas operan en paralelo
 - Memoria distribuida: info. almacenada en las conexiones
 - Aprendizaje: modificación pesos de conexiones en base a ejemplos de aprendizaje

(c) Modelo de Neurona Artificial

- **Neurona Artificial/EP:** Dispositivo que genera una salida única a partir de un conjunto de entradas con pesos asociados.
 - Entradas: binarias ($\{0, 1\}$), bipolares ($\{-1, +1\}$), continuas ($[0, 1]$), etc
 - Pesos sinápticos: representan intensidad de la conexión ($[0, 1]$)



$$y = a (g (\Sigma (\vec{x}, \vec{w})))$$

- Caracterizadas por tres funciones:
 1. **Función de transferencia (Σ):** Calcula la entrada al EP.
 - Combina valores de pesos y entradas
 - Ejemplos:

{	suma ponderada: $\sum_{i=1}^n w_i x_i$ (más frecuente)
	distancia: $\sqrt{\sum_{i=1}^n (x_i - w_i)^2}$
 2. **Función de activación (g):** Calcula estado de activación de la neurona en función de las entradas y, opcionalmente, del estado de activación actual.

Ejemplos:

<i>escalón</i>	<i>signo</i>	<i>sigmoideal</i> $\left[g(x) = \frac{1}{1+e^{-x}} \right]$
----------------	--------------	--

3. **Función de salida (a):** Genera la salida de la neurona en función del estado de activación.
 - Normalmente, función identidad, $a(g(x)) = g(x)$.

6.3 Evolución Histórica

■ Orígenes

- *McCulloch, Pitts*(1943): Primer modelo de neurona artificial

- *Hebb*(1949): Aprendizaje neuronal (regla de Hebb)

'' Una sinapsis aumenta su eficacia (peso) si las dos neuronas que conecta tienden a estar activas o inactivas simultáneamente. Ocurre en el caso contrario, una atenuación de ese peso sináptico''

''Si dos unidades están activas simultáneamente, entonces el peso de la conexión entre ellas debe ser incrementado en proporción a esa actividad conjunto''

- *Rosenthal*(1958): Desarrollo *perceptron* (red simple, 1 capa)

- *Widrow, Hoff*(1960): Desarrollo *adaline*

- Primera aplicación industrial real (cancelación ecos linea telef.)

■ Declive Finales 60's-80's

- *Minsky, Papert*: Estudio sobre limitaciones del perceptron.

- Reducción investigación.

- Falta de modelos de aprendizaje.

■ Resurgir

- *Hopfield*(principios 80s): Desarrollo red de *Hopfield*

- Memoria asociativa

- Sólida base teórica (fisiología cerebro + física (termodinámica))

- *Rumelhart, McLellan*(1986): Alg. aprendizaje retropropagación (*back-propagation*)

- Aplicable a perceptrones complejos (multicapa)

6.4 Tipos de R.N.A.

- **RNA:** Conjunto de EPs interconectados.
- **Arquitectura de RNA:** Topología de las conexiones entre EPs.
 - En general, estructura en capas de EPS homogéneos.
 - Capa de entrada: Reciben datos del entorno.
 - Capa de salida: Proporcionan salida de la red al entorno.
 - Capa oculta: No interacciona con el entorno.
 - En otros casos: Posibilidad de conexión intercapa, entre EPs de capas no contiguas, o en sentido inverso (realimentación)
- **Clasificación de RNAs**
 1. Por **estructura de capas:** $\left\{ \begin{array}{l} \text{monocapa} \\ \text{multicapa} \end{array} \right.$
 2. Por **flujo de info. de activación:**
 - a) Redes de propagación hacia adelante (unidireccionales):
 - Flujo de entrada a salida. Respeta orden de las capas.
 - Ej.: perceptron multicapa (percepción, predicción, ...), redes de base radial
 - b) Redes recurrentes:
 - Permiten conexiones en sentido contrario (realimentación).
 - Ej.: red Hopfield (monocapa con interconex. total) (memoria asociativa)
 3. Por **tipo de aprendizaje:**
 - a) Aprendizaje supervisado
 - Red se entrena con pares [entrada, salida_correcta].
 - Ajustar pesos para minimizar error ($salida_correcta - salida_red$)
 - Patrones entrenamiento: $[x_1, x_2, \dots, x_n, o_1, o_2, \dots, o_m]$
 n EPs entrada, m EPs salida
 - Ej.: perceptrones, redes de base radial

b) Aprendizaje no supervisado

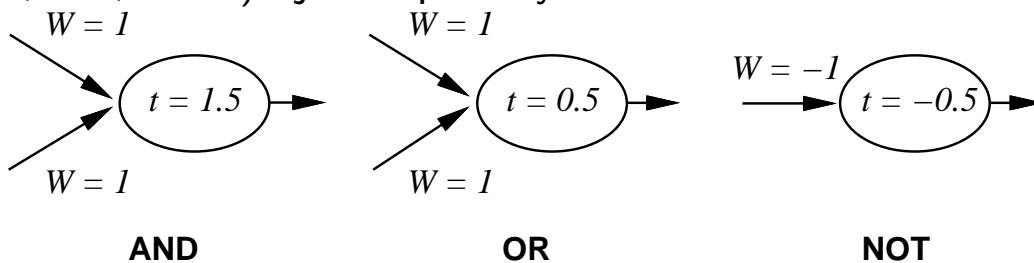
- En entrenamiento, red recibe sólo patrones de entrada.
- La red es capaz de identificar regularidades en esos patrones
- Aprende a extraer características y a clasificar/agrupar patrones de entrada
- Ej.: Mapas auto-organizativos de Kohonen.

6.4.1 Modelo de McCulloch-Pitts

- Primer modelo de neurona artificial (1943).
- Entradas (x_i) y salida (o) binarias $\{0, 1\}$.
- Función de activación de tipo escalón con umbral, T .

$$o = \begin{cases} 1 & \text{si } \sum_{i=1}^n w_i x_i \geq T \\ 0 & \text{si } \sum_{i=1}^n w_i x_i < T \end{cases}$$

- Una neurona puede "simular" cualquier función booleana simple (AND, OR, NOT) fijando pesos y umbrales adecuados.



Para AND:

x_1	x_2	$\sum w_i x_i$	o
0	0	0	0
0	1	1	0
1	0	1	0
1	1	2	1

Para OR:

x_1	x_2	$\sum w_i x_i$	o
0	0	0	0
0	1	1	1
1	0	1	1
1	1	2	1

- Funciones complejas = combinación de funciones simples.

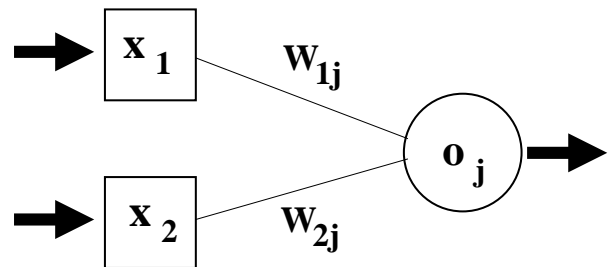
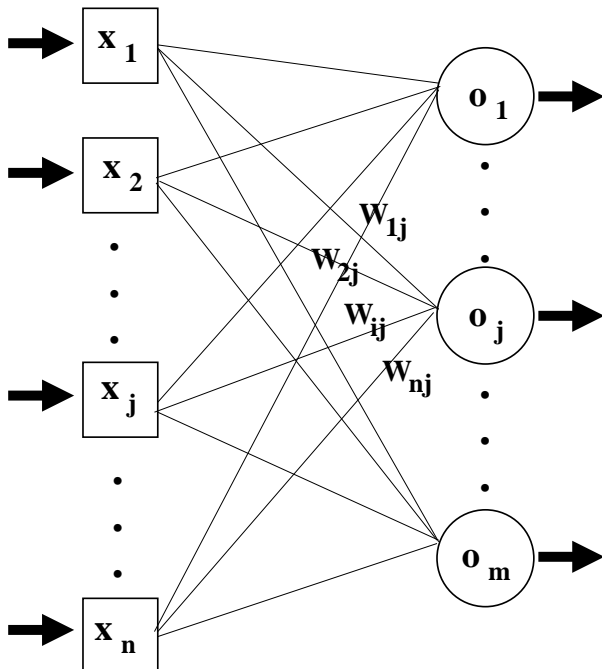
6.4.2 Perceptrón Simple

- Modelo de red neuronal propuesto por Roseblatt (1959)
 - Red monocapa con alimentación hacia adelante.
 - Colección de neuronas similares a McCulloch-Pitts, con entradas continuas $[0, 1]$, umbral T_j y salida bipolar $\{-1, 1\}$.

$$o_j = \begin{cases} 1 & \text{si } \sum_{i=1}^n w_{ij}x_i \geq T_j \\ -1 & \text{en caso contrario} \end{cases}$$

NOTA: normalmente se evita usar el umbral directamente y se sustituye por una entrada ficticia x_0 con valor fijo -1 , cuyo peso w_{0j} se corresponderá con el umbral T_j .

$$o_j = \begin{cases} 1 & \text{si } \sum_{i=0}^n w_{ij}x_i \geq 0 \\ -1 & \text{en caso contrario} \end{cases}$$



- Capaz de aprender a clasificar patrones linealmente separables.
 - Salida +1 si pertenece a la clase, -1 si no pertenece
 - Pesos determinan zonas de clasificación separadas por un hiperplano (en el caso de 2 entradas, la separación será una línea)
 - Ejemplo: con dos entradas:

$$\text{Salida unidad } o_j = 1 \text{ si : } w_{1j}x_1 + w_{2j}x_2 \geq T_j$$

$$\text{Recta separación : } x_2 \geq - \left(\frac{w_{1j}}{w_{2j}} \right) x_1 + \frac{T_j}{w_{2j}}$$

- Minsky y Papert (1969) mostraron esta limitación del perceptrón para representar/aprender funciones no separables linealmente (ejemplo: XOR no separable linealmente, necesita mas de 1 capa)

Aprendizaje en Perceptrón Simple

- Aprendizaje supervisado.
 - Asociación: [patrón entrada, patrón salida].
 - Ajuste de pesos de conexión (aprendizaje) en base al error (salida deseada vs. salida obtenida)

Inicializar pesos aleatoriamente

Repetir hasta tener salidas "suficientemente" correctas (EPOCH)

Para cada patrón de entrenamiento (E,T)

Calcular salida actual (O)

Comparar con salida deseada (T-O)

Actualizar pesos según regla de aprendizaje

- Actualización de pesos: **Regla del Perceptron**

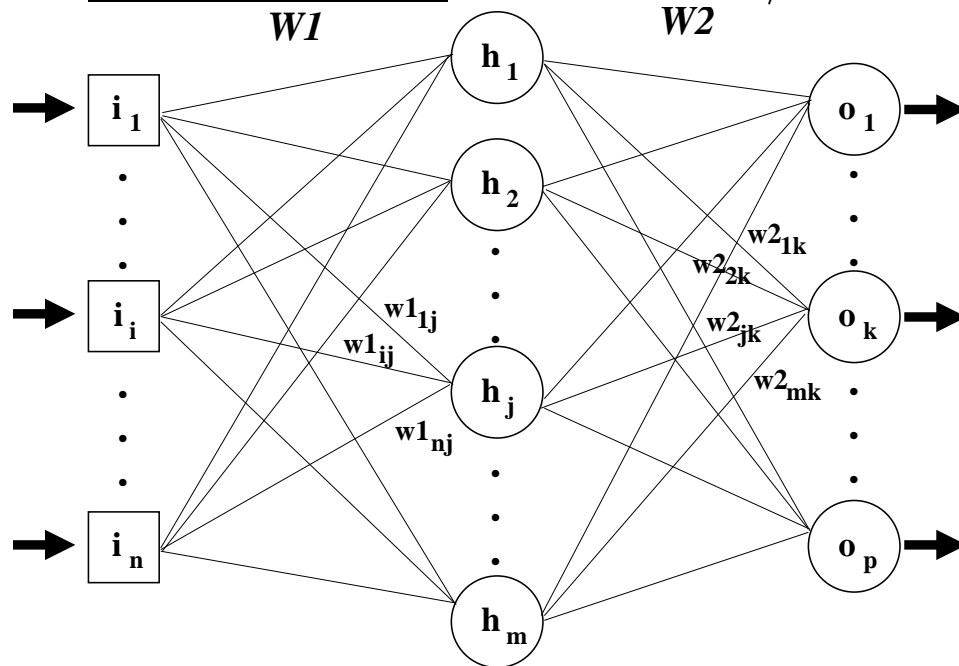
- Para la neurona j , sean: $\begin{cases} t_j : \text{salida } \underline{\text{deseada}} \\ o_j : \text{salida } \underline{\text{obtenida}} \end{cases}$
- Nuevos pesos en neurona j :

$$W_{ij} = W_{ij} + \alpha x_i (t_j - o_j)$$

- Con: α = tasa/velocidad de aprendizaje
 - Gobierna la "brusquedad" de los cambios de pesos.
 - Al inicio del entrenamiento alta (0.9). Al final baja (0.05).
- Interpretación intuitiva:
 - Repartir el error $(t_j - o_j)$ entre los pesos de las entradas (x_i) implicadas.
 - Salida correcta: no cambia pesos
 - Salida *baja* ($t_j = +1; o_j = -1$)
 - ◇ Aumentar pesos de las entradas no nulas.
 - ◇ Variación peso: $w_{ij} = w_{ij} + \beta x_i$ ($\beta = 2\alpha$)
 - Salida *alta* ($t_j = -1; o_j = +1$)
 - ◇ Decrementar pesos de las entradas no nulas.
 - ◇ Variación peso: $w_{ij} = w_{ij} - \beta x_i$ ($\beta = 2\alpha$)
- El umbral se ajusta como un peso mas (w_{0j}).

6.4.3 Perceptrones Multicapa (I)

- Extensión del perceptron simple → añade capas adicionales.
- Capas con interconexión total: entrada, oculta/s, salida.



- Propagación hacia adelante:
 - Patrones de entrada se presentan en capa de entrada.
 - Se propagan hasta generar salida.
- Función activación neuronas: **sigmoidal**.

$$g\left(\sum w_{ij}x_i\right) = \frac{1}{e^{-\sum w_{ij}x_i}}$$

- Entradas y salidas continuas $[0, 1]$
- Pesos de conexión determinan una función que relaciona entradas con salidas.
 - Sin capa oculta: funciones lineal^{te} separables (perceptron simple)
 - Una capa oculta: funciones continuas
 - Dos capas ocultas: funciones no continuas
- Método de entrenamiento: **retropropagación** (*backpropagation*)

6.4.3 Perceptrones Multicapa (Método Retropropagación)

- Propuesto por Rumelhart (1984)
- Objetivo: ajustar pesos para reducir el error cuadrático de las salidas.
- Funcionamiento aprendizaje:

Inicializar pesos aleatoriamente

Repetir hasta tener salidas "suficientemente" correctas (EPOCH)

Para cada patrón de entrenamiento (E,T)

Propagar E por la red para obtener salida actual (O)

Comparar con salida deseada (T-O)

Actualizar pesos hacia atrás, capa a capa.

- Exige función de activación (g) continua y derivable \Rightarrow sigmoideal.

$$g(x) = \frac{1}{e^{-x}} \quad g'(x) = g(x)(1 - g(x))$$

- (1) Ajuste pesos CAPA SALIDA

Nuevos pesos para neurona de salida o_k :

$$W_{2jk} = W_{2jk} + \alpha h_j \Delta_k$$

- Idea base: "Repartir" error obtenido para cada neurona de salida (Δ_k) entre los pesos de sus conexiones de forma proporcional a la intensidad de la entrada recibida (h_j).

- Usa: $\left\{ \begin{array}{l} h_j : \text{activación neurona oculta } h_j \\ (T_k - o_k) : \text{error (salida deseada - salida obtenida)} \\ g'(ent_k) : \text{derivada func. activ. (sigmoideal: } g'(ent_k) = o_k(1 - o_k)) \\ \alpha : \text{tasa de aprendizaje} \end{array} \right.$

$$\Delta_k = g'(ent_k) (T_k - o_k) = o_k (1 - o_k) (T_k - o_k)$$

■ (2) Ajuste pesos CAPA/S OCULTA/S

Nuevos pesos para neurona oculta h_j :

$$W1_{ij} = W1_{ij} + \alpha i_i \Delta_j$$

- Problema: Cuantificar error en las capas ocultas (Δ_j).
- Idea: Propagar la parte proporcional del error en la capa de salida (Δ_k) del cual es "responsable" cada neurona oculta h_j .
 - **NOTA**: En este caso se considera *solo 1 capa oculta* que realimenta hacia atras el error presente en la capa de salida.
Si hubiera mas de 1 capa oculta, la idea seria la misma: realimentar el error presente en la siguiente capa [oculta o de salida].
- **Estimacion del error en neurona oculta h_j** :

$$\Delta_j = g'(ent_j) \sum_{k=1}^p (W2_{jk} \Delta_k) = h_j (1 - h_j) \sum_{k=1}^p (W2_{jk} \Delta_k)$$

6.4.3 Perceptrones Multicapa (Interpretación aprendizaje retroprop.)

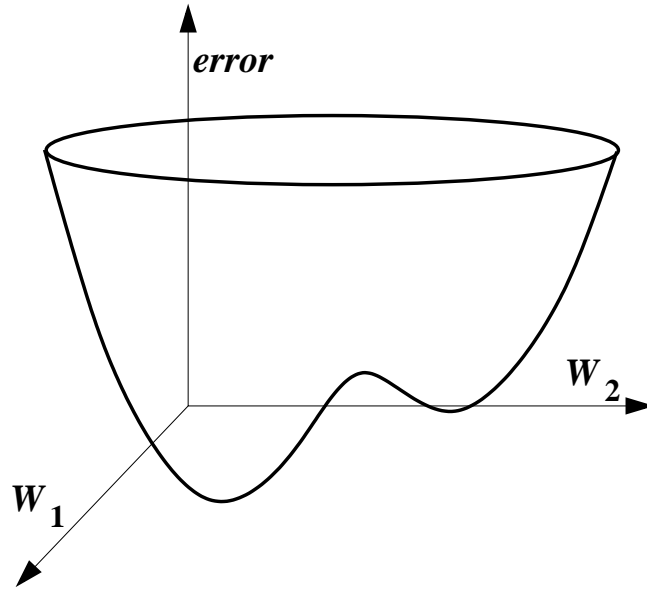
- Aprendizaje retropropag. \approx búsqueda por descenso de gradiente.
- Búsqueda en espacio de estados de tipo "ascenso a colinas"
 - Espacio de búsqueda = conjunto de posibles valores de pesos
 - Métrica de evaluación = error cuadrático (en función de pesos)

$$E(red) = \frac{1}{2} \sum_{k=1}^p (T_k - O_k)^2$$

$$E(pesos) = \frac{1}{2} \sum_{k=1}^p \left(T_k - g \left(\sum_{j=1}^m W_{2jk} g \left(\sum_{i=1}^n W_{1ij} i_i \right) \right) \right)^2$$

- En cada ejemplo de entrenamiento:
 - A partir del error calculado \rightarrow se definen los pesos de la nueva red.
 - Se ajustan pesos en la dirección de **mayor pendiente**.
 \Rightarrow Uso de la derivada g' en el ajuste pesos.
 - Objetivo: Minimizar valor de la función de error.
 - ◇ En realidad se pretende hacer que su derivada ($E'(red)$) sea 0.
 - ◇ Se buscan unos pesos para los que esa función tenga un mínimo ($E'(pesos) = 0$)
- Inconveniente: No garantizada convergencia en una red óptima.
 - Problemas con mínimos locales.
 - Funciones separables linealmente sólo tienen un mínimo local \Rightarrow se puede asegurar que se encontrará ese único mínimo.

- Ejemplo: Superficie de error con 2 pesos/entradas.



6.5 Aprendizaje supervisado en RNAs.

- En general, no se garantiza convergencia.
- En la práctica, es posible entrenamiento adecuado para problemas reales.
 - Entrenamiento lento.
 - Puede requerir muchos ciclos de entrenamiento (*epoch*).
 - Se suele usar: conj. entrenamiento + conj. validación.
 - Verificar si realmente ha aprendido y puede generalizar.
- Criterios de parada:
 - Número fijo de ciclos de entrenamiento (*epochs*)
 - Umbral de error sobre conjunto entrenamiento.
 - Umbral de error sobre conjunto de validación.
- Problemas del aprendizaje.
 1. **¿Cuál es la topología de red adecuada?**
 - Pocas neuronas/capas → incapaz de aprender función.
 - Muchas neuronas/capas → posibilidad de sobreajuste.
 - Además: ¿Qué funciones activación usar?, ¿Cómo codificar entradas/salidas?, etc,...
 2. **Sobreentrenamiento y Sobreajuste**
 - Error pequeño conj. entrenamiento, grande conj. validación.
 - Demasiados ciclos entrenamiento ⇒ Pérdida capacidad generalización.
 - Red demasiado compleja ⇒ Se ajusta a cualquier función.
 - Se memorizan patrones, no se generaliza con otros nuevos.
 3. **Mínimos locales**
 - Se mitigan con múltiples fases de entrenamiento, inicializadas con distintos pesos aleatorios.

Algoritmo de Retropropagacion. Ejemplo

Francisco José Ribadas Pena

INTELIGENCIA ARTIFICIAL

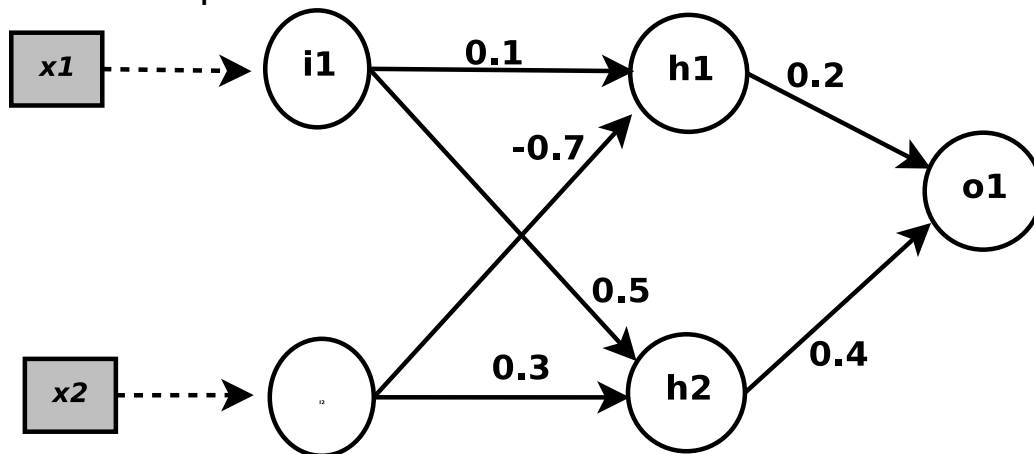
5º Informática

`ribadas@uvigo.es`

12 de diciembre de 2005

Descripción del ejemplo

- Entrenamiento de un perceptron multicapa para realizar la operación XOR
- Descripción de la red.
 - 1 capa oculta
 - 2 neuronas en capa de entrada (i_1, i_2)
 - 2 neuronas en capa oculta (h_1, h_2)
 - 1 neurona en capa de salida (o_1)
- Red inicial con pesos aleatorios



$$W_1 = \begin{pmatrix} 0,1 & 0,5 \\ -0,7 & 0,3 \end{pmatrix} \quad W_2 = \begin{pmatrix} 0,2 \\ 0,4 \end{pmatrix}$$

- Conjunto de entrenamiento

	Entradas		Salida
	x_1	x_2	t_1
e_1	0	1	1
e_2	1	0	1
e_3	1	1	0
e_4	0	0	0

- Tasa de aprendizaje: $\alpha = 0,25$

Propagación hacia adelante del ejemplo e_1

■ Entradas

- $x_1 = 0, x_2 = 1$
- Salida esperada: $t_1 = 1$

■ Capa oculta

- Neurona h_1 :

$$\begin{aligned} \text{Entrada: } & 0,1 * 0 + (-0,7) * 1 = -0,7 \\ \text{Salida: } & \frac{1}{1+e^{0,7}} = \mathbf{0,332} \end{aligned}$$

- Neurona h_2 :

$$\begin{aligned} \text{Entrada: } & 0,5 * 0 + 0,3 * 1 = 0,3 \\ \text{Salida: } & \frac{1}{1+e^{-0,3}} = \mathbf{0,574} \end{aligned}$$

■ Capa de salida

- Neurona o_1 :

$$\begin{aligned} \text{Entrada: } & 0,2 * 0,332 + 0,4 * 0,574 = 0,296 \\ \text{Salida: } & \frac{1}{1+e^{-0,296}} = \mathbf{0,573} \end{aligned}$$

Abreviado en "notacion vectorial"

- Salida capa oculta:

$$\vec{H} = g(\vec{X} \cdot W_1) = g\left((0, 1) \cdot \begin{pmatrix} 0,1 & 0,5 \\ -0,7 & 0,3 \end{pmatrix}\right) = (0,468, 0,574)$$

- Salida capa de salida:

$$\vec{O} = g(\vec{H} \cdot W_2) = g\left((0,468, 0,574) \cdot \begin{pmatrix} 0,2 \\ 0,4 \end{pmatrix}\right) = (0,573)$$

Ajuste de pesos por retropropagacion del error

(1) Pesos de la capa de salida

■ Neurona o_1 :

- Error real obtenido en neurona o_1 : $t_1 - o_1 = 1 - 0,573 = 0,427$
- Nuevos pesos para neurona o_1 : $W_{2jk} = W_{2jk} + \alpha h_j \Delta_k$

$$\begin{aligned}\Delta_k &= g'(ent_k) * (T_k - o_k) = o_k * (1 - o_k) * (T_k - o_k) \\ \Delta_1 &= 0,573 * (1 - 0,573) * 0,427 = \mathbf{0,1044}\end{aligned}$$

$$W_{211} = W_{211} + \alpha h_1 \Delta_1 = 0,2 + 0,25 * 0,332 * 0,1044 = \mathbf{0,2086}$$

$$W_{221} = W_{221} + \alpha h_2 \Delta_1 = 0,4 + 0,25 * 0,574 * 0,1044 = \mathbf{0,4149}$$

(2) Pesos de la capa oculta

- Formulas de ajuste:
 - Error estimado en neurona h_j :

$$\Delta_j = g'(ent_j) \sum_{k=1}^p (W_{2jk} \Delta_k) = h_j (1 - h_j) \sum_{k=1}^p (W_{2jk} \Delta_k)$$

OJO: W_{2jk} se refiere a los pesos de la capa de salida antiguos (antes del ajuste anterior)

- Nuevos pesos para neurona h_j : $W_{1ij} = W_{1ij} + \alpha i_i \Delta_j$
- Neurona h_1 :
 - Error estimado:

$$\begin{aligned} \Delta_1 &= h_1 * (1 - h_1) * (W_{211} * 0,1044) = \\ &= 0,332 * (1 - 0,332) * (0,2 * 0,1044) = 0,046 \end{aligned}$$

- Nuevos pesos:

$$\begin{aligned} W_{111} &= W_{111} + \alpha i_1 \Delta_1 = 0,1 + 0,25 * 0 * 0,046 = 0,1 \\ W_{121} &= W_{121} + \alpha i_2 \Delta_1 = -0,7 + 0,25 * 1 * 0,046 = -0,684 \end{aligned}$$

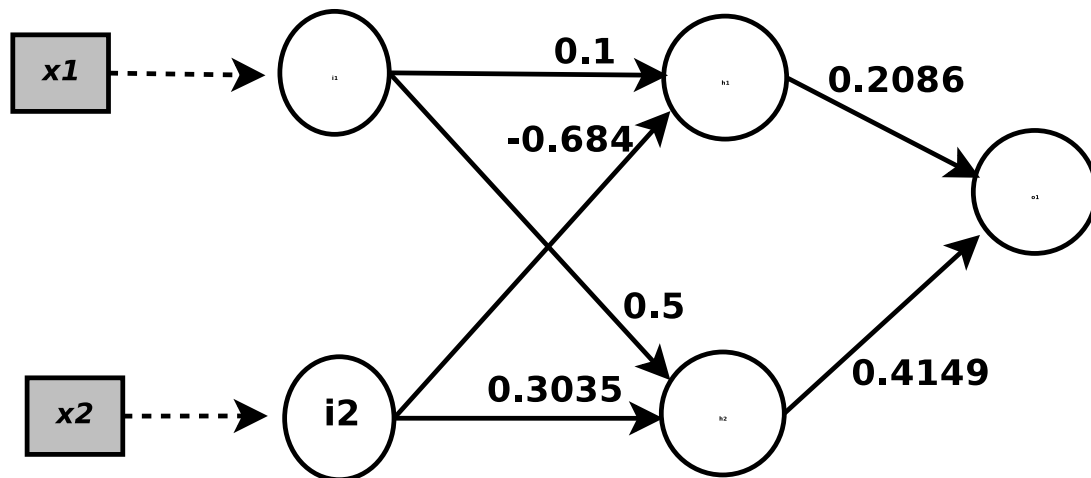
- Neurona h_2 :
 - Error estimado:

$$\begin{aligned} \Delta_2 &= h_2 * (1 - h_2) * (W_{221} * 0,1044) = \\ &= 0,574 * (1 - 0,574) * (0,4 * 0,1044) = 0,0102 \end{aligned}$$

- Nuevos pesos:

$$\begin{aligned} W_{112} &= W_{112} + \alpha i_1 \Delta_2 = 0,5 + 0,25 * 0 * 0,0102 = 0,5 \\ W_{122} &= W_{122} + \alpha i_2 \Delta_2 = 0,3 + 0,25 * 1 * 0,0102 = 0,3025 \end{aligned}$$

Nueva red



Para el mismo ejemplo e_1 , la salida será un poco mejor (más cercana al objetivo)

- Salida capa oculta:

$$\vec{H} = g(\vec{X} \cdot W1) = g\left((0, 1) \cdot \begin{pmatrix} 0,1 & 0,5 \\ -0,684 & 0,3035 \end{pmatrix}\right) = (0,335, 0,575)$$

- Salida capa de salida:

$$\vec{O} = g(\vec{H} \cdot W2) = g\left((0,335, 0,575) \cdot \begin{pmatrix} 0,2086 \\ 0,4149 \end{pmatrix}\right) = (0,576)$$

Mapas auto-organizativos

Francisco José Ribadas Pena

INTELIGENCIA ARTIFICIAL

5º Informática

`ribadas@uvigo.es`

14 de diciembre de 2005

Aprendizaje no supervisado y aprendizaje competitivo

Aprendizaje no supervisado

- Los algoritmos de aprendizaje no supervisado no necesitan de un *supervisor externo* que juzgue (a priori o sobre la marcha) los resultados del proceso de aprendizaje.
 - No se presentan las salidas objetivo que se quieren asociar al patrón de entrada
 - Los algoritmos de aprendizaje solo manejan patrones de entrada
- Se pretende que la red descubra por si misma rasgos comunes, regularidades, correlaciones o categorías en los datos de entrada y los incorpora a su estructura interna de conexiones (pesos).
 - Se dice que la red se auto-organiza
 - La unica info. que se usa son las similitudes y diferencias entre las entradas
 - Este tipo de aprendizaje exige que en los datos de entrada exista cierta "redundancia" para poder identificar esas regularidades.
- Regla de Hebb: primera aproximación al aprendizaje no supervisado (sin info. externa)
 - Basado en evidencias fisiológicas
 - *Cuando una axon de una célula A está lo suficientemente cerca para excitar a una célula B, y toma parte repetidamente en el proceso de disparo de dicha célula, se produce un cambio metabólico en una o ambas células, que hace que la eficacia con la que A dispara a B se vea incrementada.*
 - De forma abreviada: *cuando una neurona activa a otra, la sinapsis entre ambas queda reforzada*
 - No depende de factores externos, las células simplemente se influyen unas a otras.
 - Múltiples formas de traducir esta idea a mecanismos de ajuste de pesos

Aprendizaje competitivo

- Familia de modelos de RNA que soportan aprendizaje supervisado
 - Normalmente estructuradas en 2 capas: $\left\{ \begin{array}{l} \text{capa de entrada} \\ \text{capa de competicion} \end{array} \right.$
- **Objetivo:** aprender a categorizar/ agrupar los datos de entrada
 - Se persigue que datos parecidos hagan reaccionar a las mismas neuronas
 - Se consigue haciendo que cada neurona se especialice en determinado "tipo" de patrones de entrada
 - Las neuronas juegan el papel de "prototipos" de los datos de entrada
- **Idea:** Para cada patrón de entrada se restringe la actualización de pesos **sólo** a la/las neuronas de la capa de competicion cuyo grado de activación haya sido más alto
 - Neuronas ganadoras = neuronas con "mayor" nivel de activación
 - Neuronas ganadoras se refuerzan a si mismas (opcionalmente, tb. a sus vecinas)
- Las neuronas compiten por la entrada
- La neurona ganadora (junto con sus pesos) representa al prototipo que se le asigna al dato de entrada

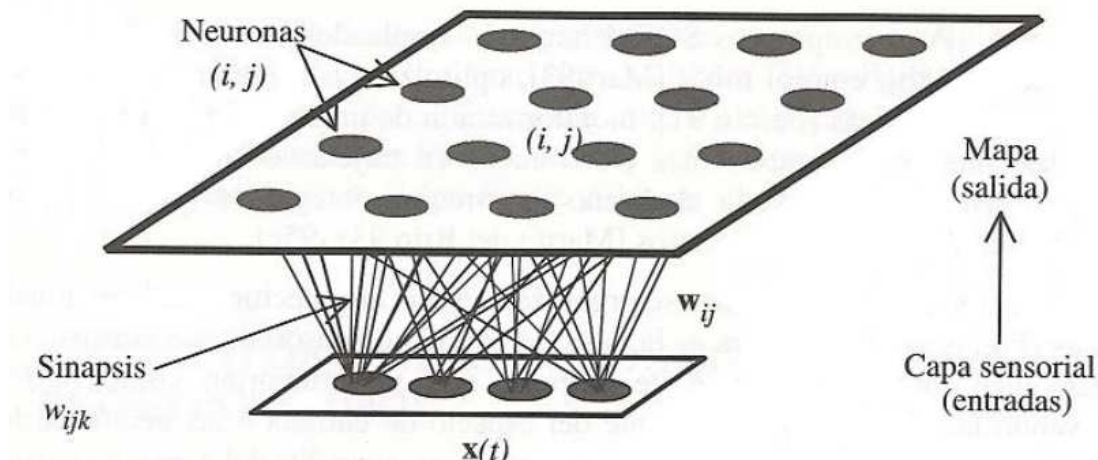
Mapas auto-organizativos

Fundamentos

- En la corteza cerebral existen zonas especializadas en ciertas tareas
 - Neuronas asociadas a características similares son vecinas
 - Se generan "mapas" donde se "ordenan" las características de los estímulos recibidos
- Teuvo Kohonen(1982) diseñó un modelo de red competitiva que emula esta idea
 - Los SOM (*self-organizing map*) reciben un conjunto de datos y construyen una representación (mapa neuronal) de menor dimensión
 - Neuronas asociadas con vectores similares tendrán pesos similares
- Características generales de los SOM
 - Los pesos sinápticos son representativos de determinados tipos de patrones de entrada
 - Los patrones de entrada son presentados a todas las neuronas simultáneamente
 - Uso de aprendizaje competitivo: solo la neurona con mejor respuesta es tomada en cuenta
 - Ajuste de pesos basado en los conceptos de auto-refuerzo y vecindad

Estructura de los SOM de Kohonen

- Red neuronal de 2 capas: $\left\{ \begin{array}{l} \text{capa de entrada} \\ \text{capa de competicion} \end{array} \right.$



- Capa de entrada recibe la señal de entrada a la red (no hay procesamiento)
 - Su dimension, n , depende del problema
 - Sera un vector $\vec{X} = (x_1, x_2, \dots, x_n)$
- Capa de competicion: formada por m neuronas
 - Cada neurona de competicion esta conectada con todas la neuronas de entrada
 - Los pesos de la neurona i formaran un vector de n dimensiones: $\vec{W}_i = (w_{1i}, w_{2i}, \dots, w_{ni})$
 - No hay conexion entre las neuronas de competicion
 - Si existe una relacion de *vecindad* usada en el aprendizaje

Funcionamiento de los SOM de Kohonen

- En el modo de operación **normal** permanecen fijos los pesos.
 - Cada neurona i calcula la similitud entre el vector de entrada \vec{X} y su vector de pesos \vec{W}_i
 - Vence aquella con mayor similitud.
- En la fase de **aprendizaje**, la neurona vencedora ajusta sus pesos aproximándose cada vez más a los de \vec{X} .
 - Por la función de vecindad también actualizan sus pesos neuronas vecinas a la vencedora

Aprendizaje en los mapas auto-organizativos (I)

- Inicialmente los pesos de cada neurona se establecen aleatoriamente
- Durante entrenamiento, se elige al azar un vector de entrada con el que se realizan 2 tareas
 - determinar neurona ganadora
 - modificacion de pesos

■ (1) Selección de la neurona ganadora

- Al recibir un patron \vec{X} cada neurona compara su vector de pesos con el vector de entrada
 - distintas funciones de distancia posible
 - la mas usual es la *distancia euclidea*

$$d(\vec{X}, \vec{W}_i) = \sqrt{\sum_{j=1}^n (x_j - w_{ji})^2} \quad \forall 1 \leq i \leq m$$

- La única neurona ganadora es aquella con los pesos mas parecidos al patron
 - solo se activara la neurona cuya distancia sea la menor

$$g(\text{neurona}_i) = \begin{cases} 1 & \text{si } d(\vec{X}(t), \vec{W}_i) = \min_k \{d(\vec{X}, \vec{W}_k)\} \\ 0 & \text{en otro caso} \end{cases}$$

■ (2) Ajuste de pesos

- Solo se realizara ajuste de pesos en la neurona ganadora y sus "vecinas"
- La idea es que dicha neurona se "especialice" en patrones similares
- Se ajustan los pesos para hacerlos mas parecidos al patron de entrada que provoco la activacion

Idea base: acercar vector de pesos al vector de entrada
- Ajuste de pesos controlado por dos parametros que varian con el tiempo (num. patrones procesados)
 - funcion de vecindad y amplitud del "vecindario"
 - tasa de aprendizaje

Aprendizaje en los mapas auto-organizativos (II)

Ajuste de pesos

- Nuevos pesos despues de la iteracion t para la neurona i

$$\vec{W}_i^{t+1} = \vec{W}_i^t + \alpha(t+1) H_g(t, |i - G|) [\vec{X} - \vec{W}_i^t]$$

- Con $\alpha(t+1)$ el valor de la tasa de aprendizaje para la presente iteracion
 - Se va reduciendo con el tiempo (una función determina como decrece a medida que aumentan las iteraciones)
 - Al principio: valores grandes, provocan cambios relativamente bruscos en la organiz. de las neuronas
 - Al final: valores casi nulos, para que la red se estabilice y converga
- Con $H_g(t, |i - G|)$ una media que indica el grado de vecindad entre la neurona i y la neurona ganadora G
 - Valdra 1 para la neurona ganadora G y 0 para las que no esten en su vecindad
 - Su valor se va haciendo menor a medida que se reduce la vecindad con G
 - Su efecto tb. disminuye con el tiempo (a medida que aumentan las iteraciones)

Nota: $|i - G|$ denota la distancia entre las neuronas i y G en la capa competitiva en función del tipo de vecindad considerado

- **Idea base:** acercar los pesos al patron de entrada
 - El nuevo peso es el resultado de sumar al antiguo una fraccion (determinada por α y H_g) de la diferencia entre el peso antiguo y su componente correspondiente del vector de entrada.

Esquema general del algoritmo de aprendizaje en SOM

1. Inicializar pesos
 - Asignar a los pesos valores aleatorios pequeños
2. Presentar una entrada
 - El conjunto de aprendizaje se presenta repetidas veces hasta llegar a la convergencia de la red
 - Actualizar α (reducir su valor)
3. Propagar el patron de entrada hasta la capa de competicion
 - Obtener los valores de salida (distancias) de las neuronas de dicha capa
4. Seleccionar la neurona ganadora G
 - La de menor distancia al patron
5. Actualizar conexiones entre capa de entrada y la neurona C
 - Actualizar tambien los pesos de sus vecinas segun el grado de vecindad
6. Si α se mantiene por encima del umbral de parada, volver a 2, en caso contrario FIN

Aplicaciones típicas

Tipos de problemas que pueden resolver los SOM

- Condicionará como se interprete lo que representa la salida de un SOM

Agrupamiento (*clustering*) A partir de un conjunto de entrada se desea determinarse si se puede dividir ese conjunto en diferentes clases.

- Permite determinar qué clases existen
- Permite decidir a qué clase pertenece cada dato de entrada
→ determinando su neurona ganadora
- Permite caracterizar cada una de esas clases
→ mediante los pesos de cada neurona (\approx clase)

Prototipado Similar al anterior, en lugar de conocer la clase del dato, interesa obtener un prototipo de la clase a la que pertenece.

- Usa los pesos de la ganadora para determinar ese prototipo

Codificación Se obtiene a la salida de la red una versión codificada del dato de entrada.

- Se busca una salida de menor dimensión de la entrada

Análisis de componentes principales Se trata de detectar qué vectores de conj. de entrada caracterizan en mayor grado ese conjunto de datos.

- Los demás vectores podrán eliminarse sin una pérdida significativa de info

Extracción y relación de características Se pretende organizar los vectores de entrada en un mapa topológico

- A partir de la red entrenada, patrones parecidos producirán respuestas similares en neuronas cercanas
- Si existe una organización global de patrones de entrada, se verá reflejada en la salida de la red.
- Se ubican entradas parecidas y/o relacionadas en zonas próximas de la red