

Tema 14: Redes Neuronales

Abdelmalik Moujahid, Iñaki Inza, Pedro Larrañaga

Departamento de Ciencias de la Computación e Inteligencia Artificial

Universidad del País Vasco

<http://www.sc.ehu.es/isg/>

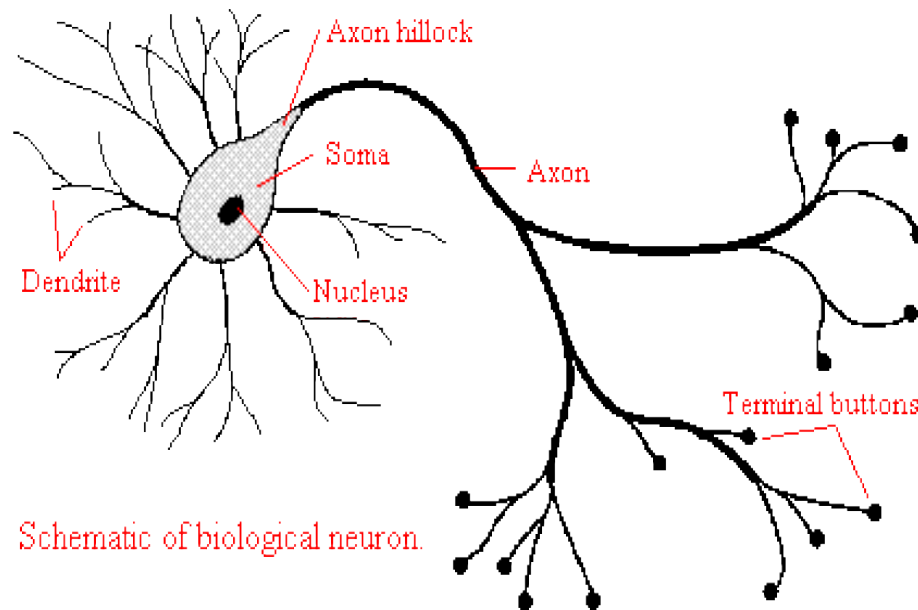
Contenido

- Introducción
- Sistema Neuronal Artificial
- El Asociador Lineal
- El Perceptrón Simple
- La Adalina
- El Perceptrón Multicapa

Introducción

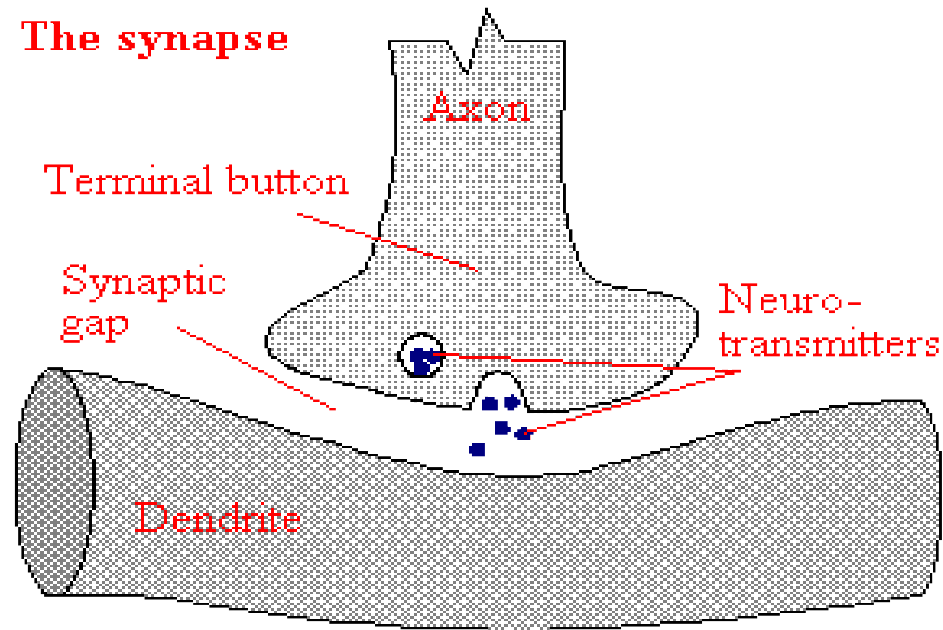
- Redes neuronales (artificiales) paradigma muy popular en Inteligencia Computacional
- En el tema se verán diferentes modelos para clasificación supervisada
- Alta capacidad de predecir correctamente la clase
- Baja transparencia

Introducción



Estructura de una neurona biológica

Introducción



Sinapsis: Conexión entre el axón de una neurona y la dendrita de otra cercana por donde se transmite un impulso nervioso

Introducción

- $100 \cdot 10^6$ neuronas en el cerebro. Procesadores de información
- Neurona:
 - Cuerpo celular o soma (10 a 80 micras)
 - Del soma surge un árbol de ramificaciones (árbol dendrítico)
 - Del soma parte una fibra tubular (axón) (100 micras hasta el metro)
 - El axón se ramifica en su extremo para conectarse con otras neuronas

Sistema Neuronal Artificial

Neurona como procesador de información:

- Dendritas canal de entrada de información
- Soma como órgano de cómputo
- Axón como canal de salida. Recibe impulso de unas 10000 neuronas. Envía impulso a cientos de neuronas
- Algunas neuronas reciben información del exterior

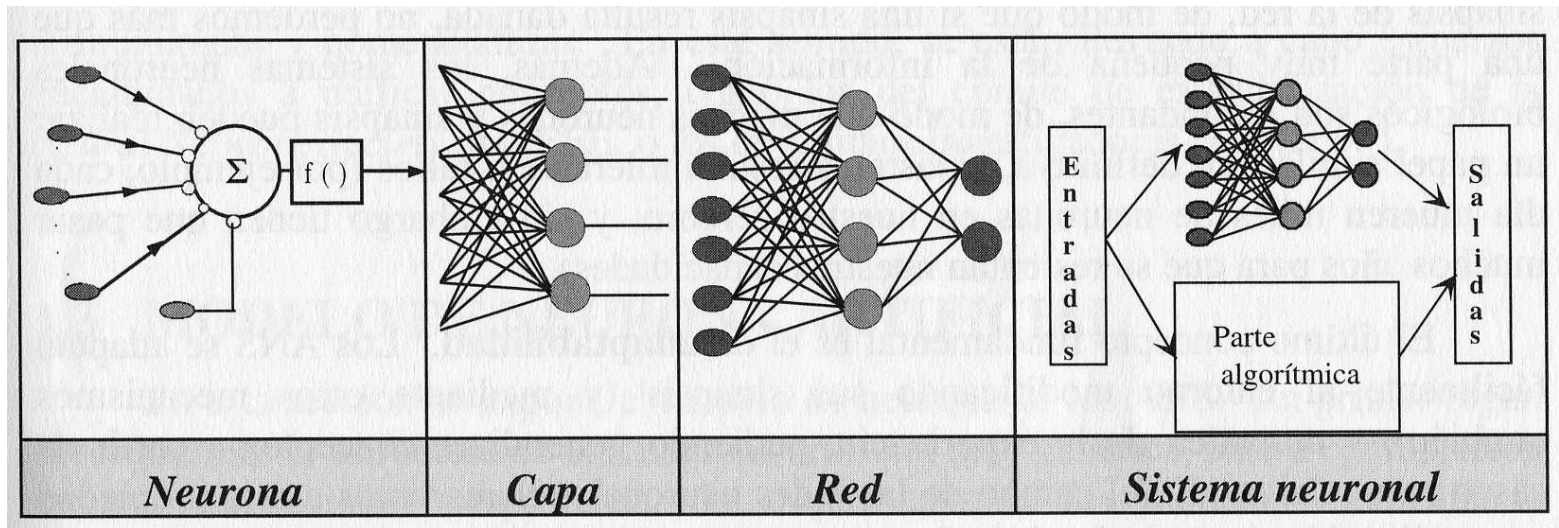
Sistema Neuronal Artificial

Tres conceptos clave a emular:

- *Procesamiento paralelo*
 - Intervención de miles de millones de neuronas
- *Memoria distribuida*
 - En un computador la información está en posiciones de memoria bien definidas
 - En redes neuronales biológicas la información está distribuida por la sinapsis de la red
 - Redundancia para evitar pérdida de información al resultar dañada una sinapsis
- *Adaptabilidad al entorno*
 - Por medio de la información de las sinapsis
 - Aprender de la experiencia
 - Posible generalizar conceptos a partir de casos particulares

Sistema Neuronal Artificial

El modelo estándar de neurona artificial
Rumelhart y McClelland (1986) y McClelland y Rumelhart (1986)

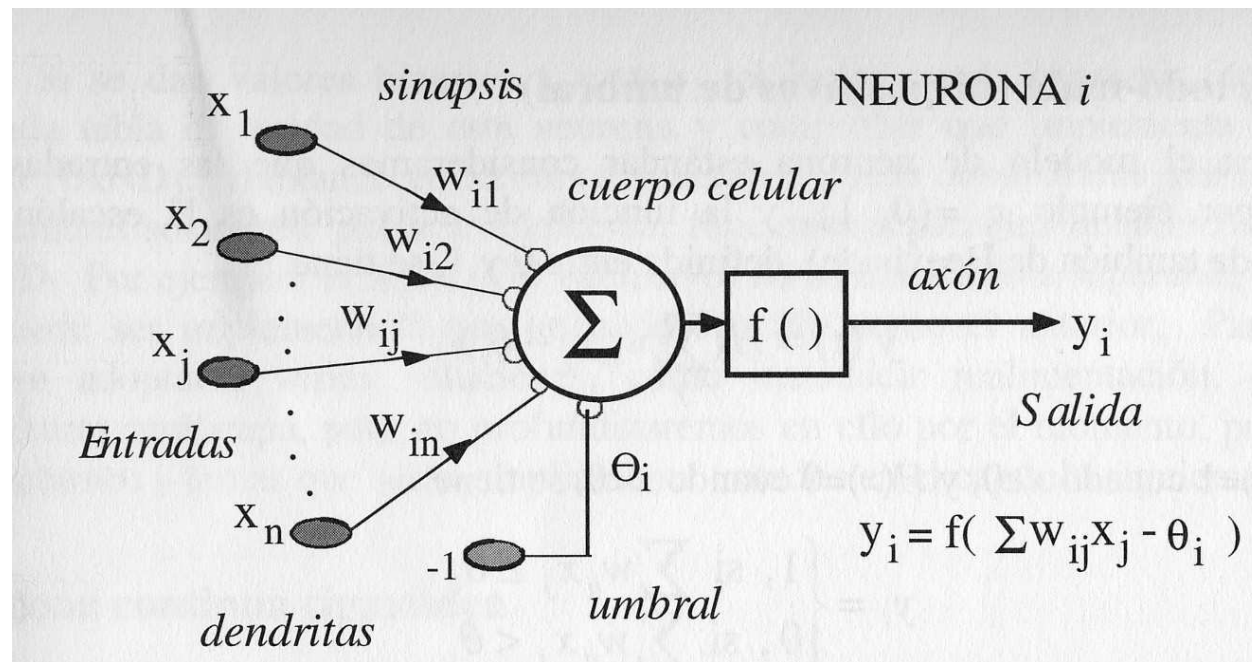


Sistema global de proceso de una red neuronal

Sistema Neuronal Artificial

El modelo estándar de neurona artificial

Rumelhart y McClelland (1986) y McClelland y Rumelhart (1986)



Modelo de neurona artificial standard

Sistema Neuronal Artificial

- *Conjunto de entradas* X_j y unos *pesos sinápticos* w_{ij} , con $j = 1, \dots, n$
- *Regla de propagación* $h_i(x_1, \dots, x_n, w_{i_1}, \dots, w_{i_n}) = \sum_{j=1}^n w_{ij} x_j$
 θ_i umbral, el cual se acostumbra a restar al potencial pos-sináptico

$$h_i(x_1, \dots, x_n, w_{i_1}, \dots, w_{i_n}) = \sum_{j=0}^n w_{ij} x_j = \sum_{j=1}^n w_{ij} x_j - \theta_i$$

- Una *función de activación*, la cual representa simultáneamente la salida de la neurona y su estado de activación

$$y_i = f_i(h_i) = f_i\left(\sum_{j=0}^n w_{ij} x_j\right)$$

Sistema Neuronal Artificial

Ejemplos de funciones de activación

- *Neuronas todo-nada*

$f_i(\sum_{j=1}^n w_{ij}x_j - \theta_i) = f_i(h_i)$ es una función escalonada

$$y_i = \begin{cases} 1 & \text{si } \sum_{j=1}^n w_{ij}x_j \geq \theta_i \\ 0 & \text{si } \sum_{j=1}^n w_{ij}x_j < \theta_i \end{cases}$$

Sistema Neuronal Artificial

Ejemplos de funciones de activación

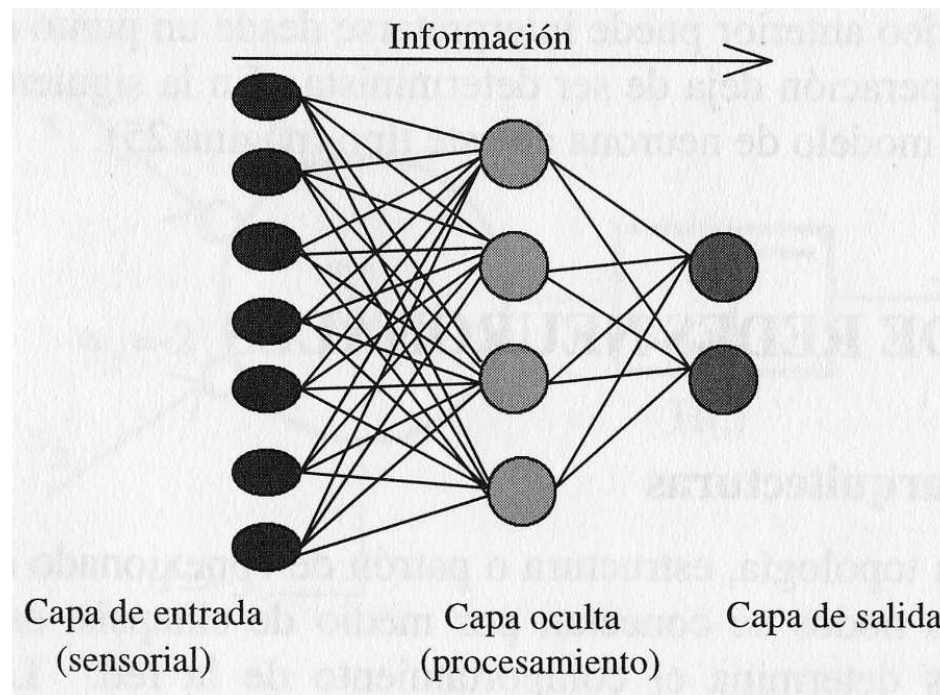
- *Neurona continua sigmoidea*

$$y_i = \frac{1}{1 + e^{- (h_i)}}, \text{ con } y_i \in [0, 1]$$

$$y_i = \frac{e^{(h_i)} - e^{- (h_i)}}{e^{(h_i)} + e^{- (h_i)}}, \text{ con } y_i \in [-1, 1]$$

Sistema Neuronal Artificial

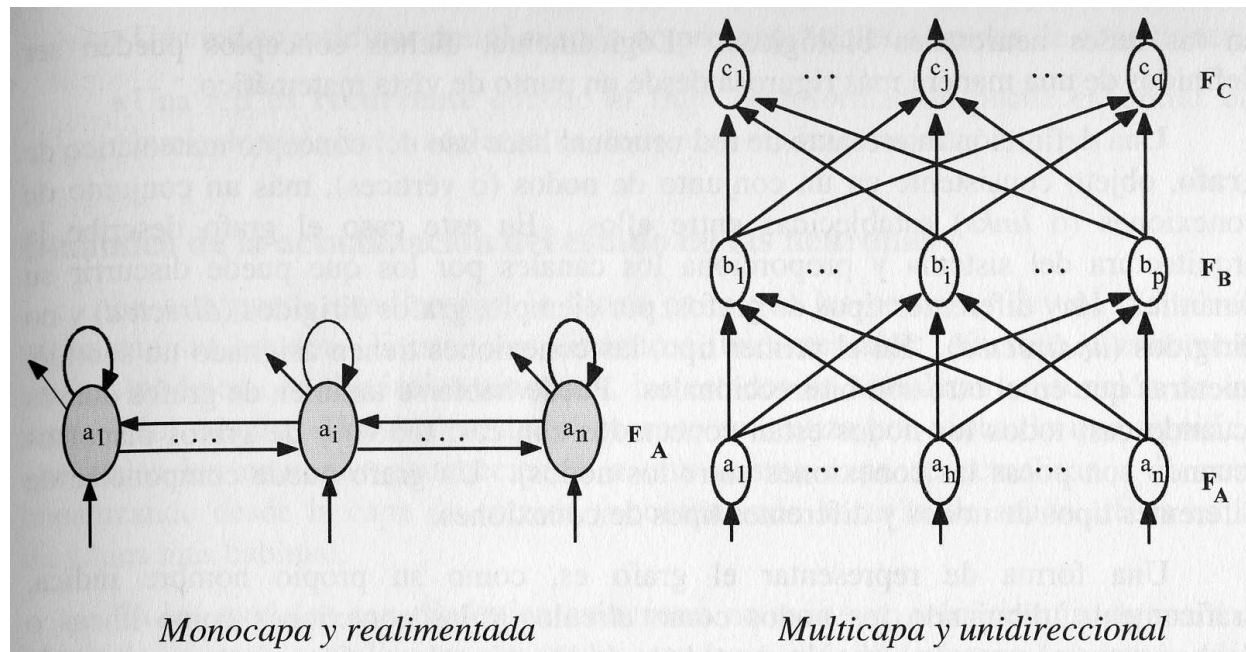
Arquitecturas de redes neuronales



Arquitectura unidireccional con tres capas de neuronas: una capa de entrada, una capa oculta y una capa de salida

Sistema Neuronal Artificial

Arquitecturas de redes neuronales



Diferentes arquitecturas de redes neuronales

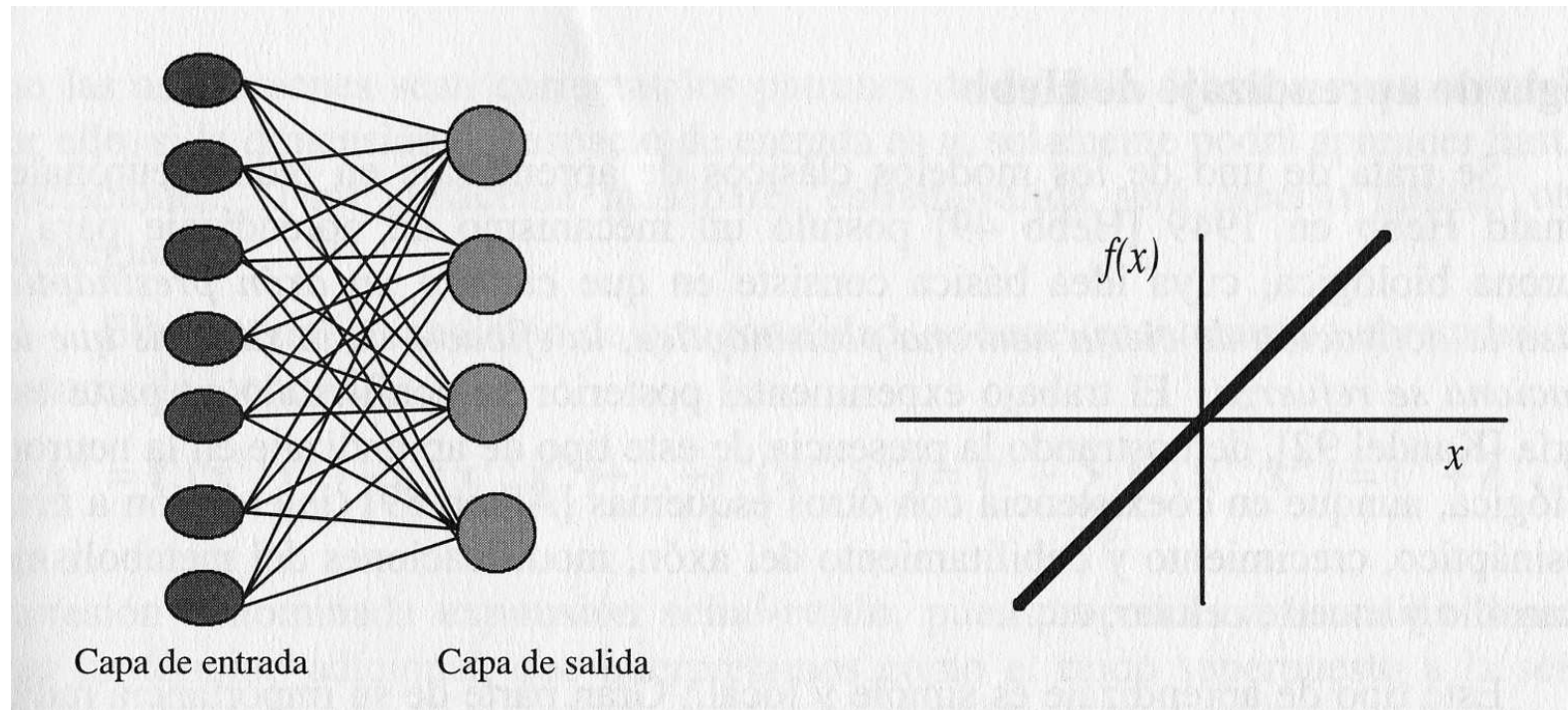
Sistema Neuronal Artificial

Definición de una red neuronal artificial

Grafo dirigido:

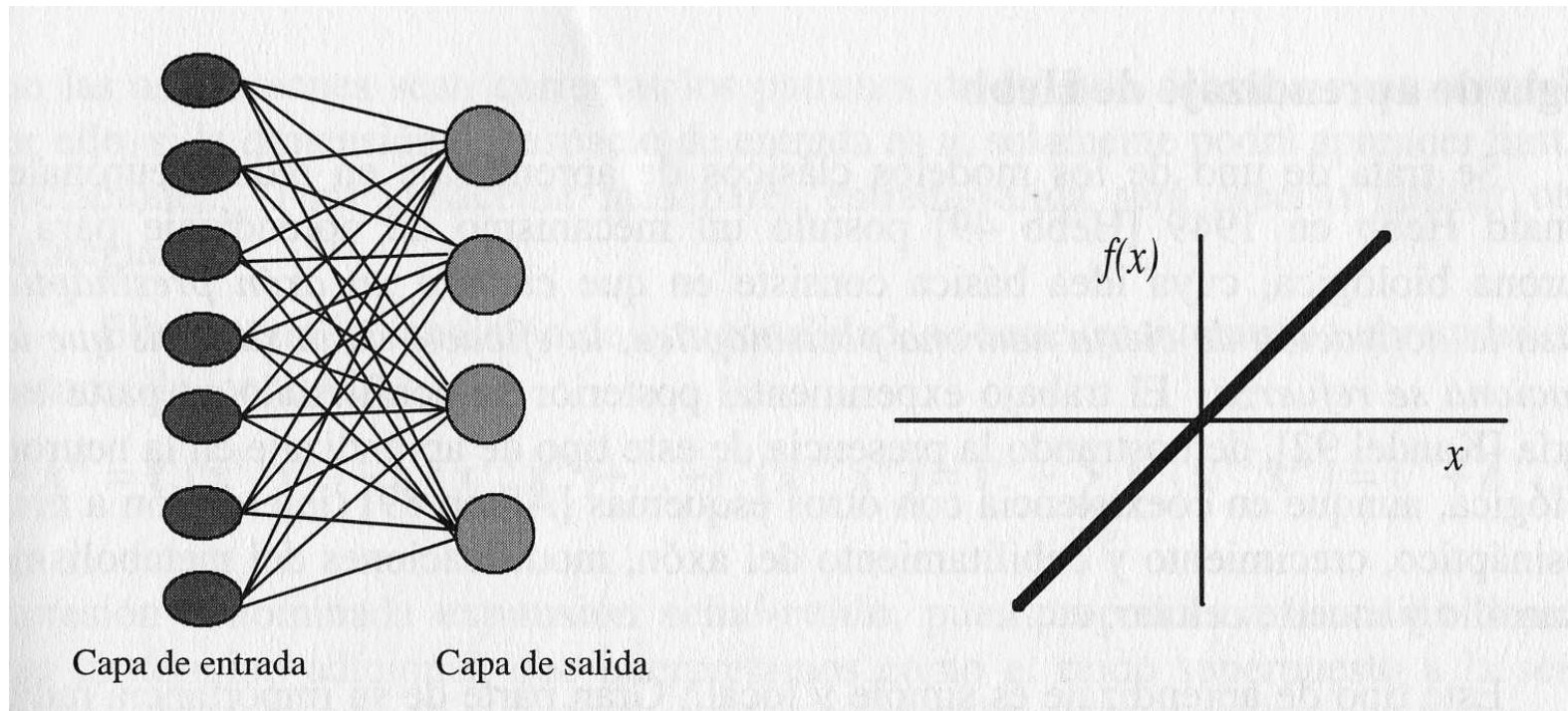
- (i) A cada nodo (neurona) i se le asocia una variable de estado X_i
- (ii) A cada conexión (i, j) entre los nodos (neuronas) i y j se le asocia un peso $w_{ij} \in \mathbb{R}$
- (iii) A cada nodo (neurona) i se le asocia un umbral $\theta_i \in \mathbb{R}$
- (iv) Para cada nodo i se define una función $f_i(x_1, \dots, x_n, w_{i_1}, \dots, w_{i_n}, \theta_i)$ que depende de los pesos de sus conexiones, del umbral y de los estados de los nodos j que estén conectados con el nodo i . El valor de esta función proporciona el nuevo estado del nodo

El Asociador Lineal



Arquitectura (izquierda) y función de activación (derecha) del asociador lineal

El Asociador Lineal



Arquitectura (izquierda) y función de activación (derecha) del asociador lineal

El Asociador Lineal

- Unicamente dos capas de neuronas:

Entradas $x_1, \dots, x_n$

Salidas $y_1, \dots, y_m$

- $W \in M(m, n)$ matriz de pesos sinápticos, con elementos w_{ij} con $i = 1, \dots, m$ y $j = 1, \dots, n$
- La operación efectuada por el asociador lineal es:

$$\mathbf{y} = (y_1, \dots, y_m) = W\mathbf{x} = W(x_1, \dots, x_n)$$

$$\text{o bien } y_i = \sum_{j=1}^n w_{ij}x_j, \text{ con } i = 1, \dots, m$$

- Potencial pos-sináptico por medio de la suma ponderada, posteriormente una función de activación identidad

El Asociador Lineal

- Aprender a asociar N pares entrada-salida

$$D = \{(\mathbf{x}^r, \mathbf{y}^r), r = 1, \dots, N\}$$

ajustando la matriz de pesos W

- Ante entradas similares a $\mathbf{x}^r$ responda con salidas similares a $\mathbf{y}^r$
- El problema radica en encontrar la matriz de pesos W óptima en el sentido anterior
- Regla de aprendizaje, que a partir de las entradas y de las salidas deseadas proporcione el conjunto óptimo de pesos W

El Asociador Lineal

- *Aprendizaje Hebbiano* (Hebb, 1949): cuando un axón presináptico causa la activación de cierta neurona pos-sináptica, la eficacia de la sinapsis que las relaciona se refuerza
- Aprendizaje es simple y local. Pionero en neurocomputación
- Modificación de pesos, Δw_{ij} , proporcional al producto de una entrada x_j y de una salida y_i de la neurona
- $\Delta w_{ij} = \varepsilon y_i x_j$, donde a $0 < \varepsilon < 1$ se le denomina *ritmo de aprendizaje*
- Para el ejemplo $(\mathbf{x}^r, \mathbf{y}^r)$ la regla de actualización de pesos es:

$$w_{ij}^{new} = w_{ij}^{old} + \Delta w_{ij}^r$$

$$\text{con } \Delta w_{ij}^r = \varepsilon y_i^r x_j^r$$

El Asociador Lineal

- Deducir los *algoritmos de aprendizaje* a partir de un cierto criterio a optimizar
- Proponer un *criterio que mida el rendimiento de la red neuronal* para encontrar una regla de actualización de pesos que la optimice
- *Error cuadrático medio* de las salidas actuales de la red respecto de las deseadas

$$\frac{1}{N} \sum_{r=1}^N \|\mathbf{y}^r - W\mathbf{x}^r\|^2 = \frac{1}{N} \sum_{r=1}^N \sum_{i=1}^m \sum_{j=1}^n (y_i^r - W x_j^r)^2$$

- El problema del aprendizaje de los pesos de la red neuronal se transforma en el de obtener un conjunto de pesos que minimicen la expresión anterior

El Asociador Lineal

- Error cuadrático medio:

$$\frac{1}{N} \sum_{r=1}^N \|\mathbf{y}^r - W\mathbf{x}^r\|^2 = \frac{1}{N} \sum_{r=1}^N \sum_{i=1}^m \sum_{j=1}^n (y_i^r - Wx_j^r)^2$$

- $\mathbf{X} = (\mathbf{x}^1, \dots, \mathbf{x}^N)$, una matriz $n \times N$ que tiene por columnas los vectores de entrada
- $\mathbf{Y} = (\mathbf{y}^1, \dots, \mathbf{y}^N)$ una matriz $m \times N$ cuyas columnas son los vectores de salida
- Error cuadrático medio:

$$\frac{1}{N} \|\mathbf{Y} - W\mathbf{X}\|^2$$

- La minimización de la expresión anterior cuando $W = \mathbf{Y}\mathbf{X}^T$
- Una regla de aprendizaje: $W = \mathbf{Y}\mathbf{X}^+$, donde $\mathbf{X}^+$ denota la matriz pseudoinversa de $\mathbf{X}$

El Perceptrón Simple

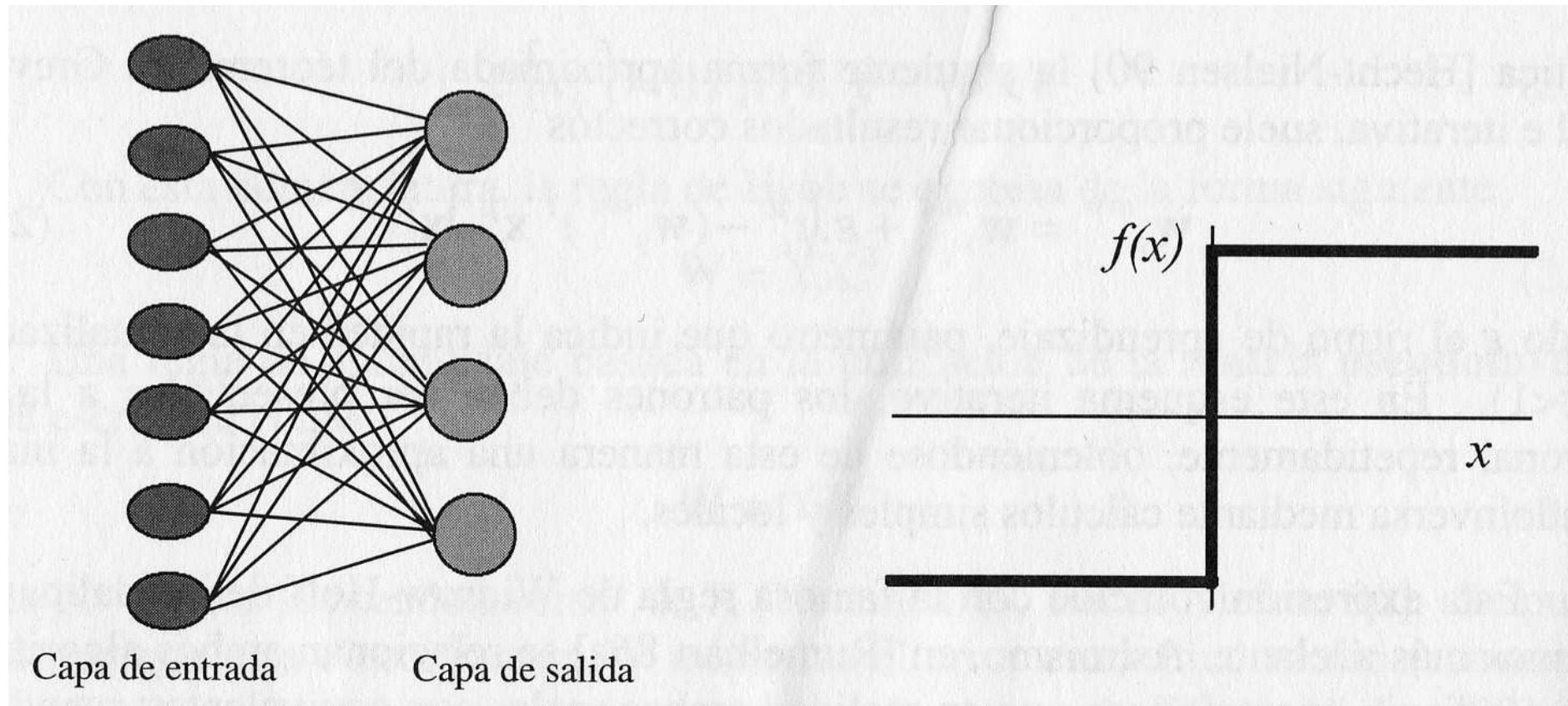
- El perceptrón (Rosenblatt, 1962) modelo unidireccional compuesto por dos capas de neuronas, una de entrada y otra de salida
- n neuronas de entrada y m neuronas de salida se puede expresar como:

$$y_i = f \left(\sum_{j=1}^n w_{ij} x_j - \theta_i \right)$$

con $i = 1, \dots, m$

- Neuronas de entrada son discretas
- Función de activación de las neuronas de la capa de salida es de tipo escalón
- Dispositivo entrenable: determinar automáticamente los pesos sinápticos que clasifican un conjunto de patrones etiquetados

El Perceptrón Simple



Arquitectura (izquierda) y función de transferencia (derecha) de un perceptrón simple

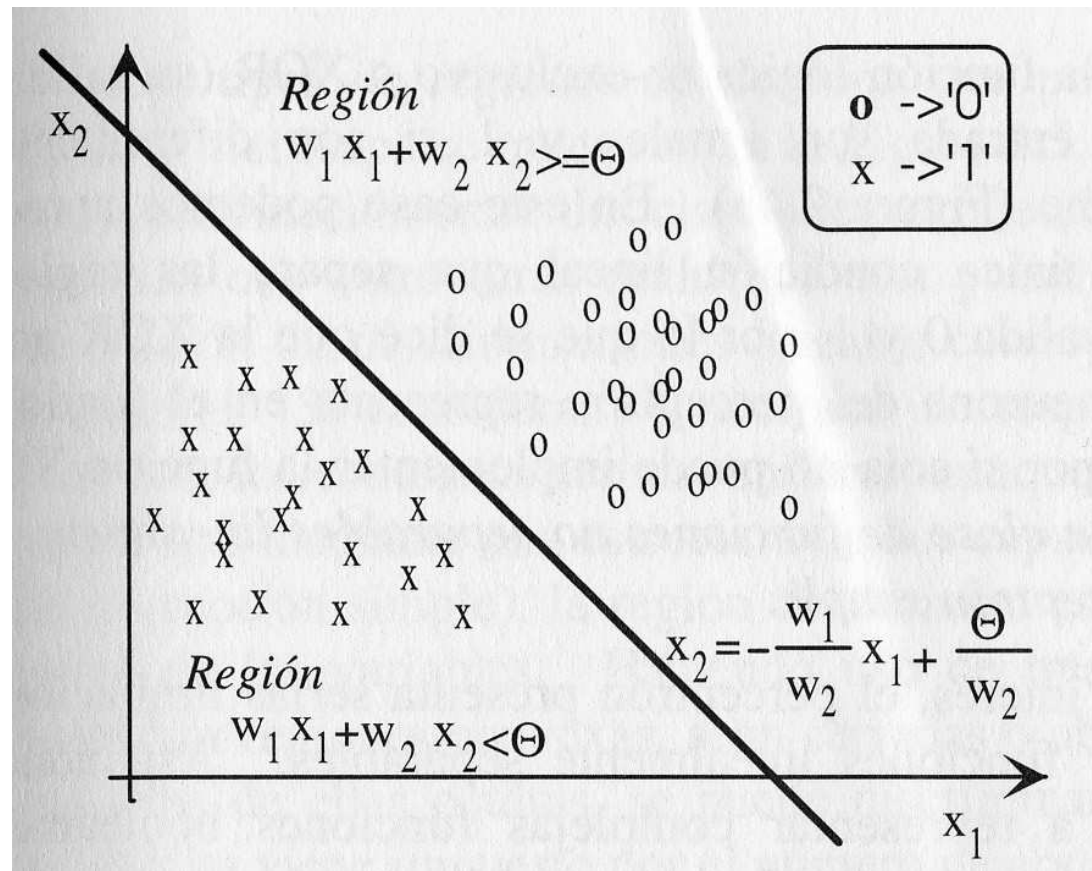
El Perceptrón Simple

- El perceptrón simple tan sólo puede discriminar entre dos clases linealmente separables
- Ejemplo: x_1 y x_2 dos neuronas de entrada, la operación efectuada por el perceptrón simple consiste en:

$$y = \begin{cases} 1 & \text{si } w_1x_1 + w_2x_2 \geq \theta \\ 0 & \text{si } w_1x_1 + w_2x_2 < \theta \end{cases}$$

- Si consideramos x_1 y x_2 situadas sobre los ejes de abscisas y ordenadas respectivamente, la condición $w_1x_1 + w_2x_2 - \theta = 0$ es equivalente a $x_2 = -\frac{w_1}{w_2}x_1 + \frac{\theta}{w_2}$ y representa una recta que define la región de decisión determinada por el perceptrón simple
- Minsky y Papert (1969) trabajo exponiendo las limitaciones del perceptrón simple, como consecuencia recursos de las redes neuronales a otros campos de la Inteligencia Artificial

El Perceptrón Simple



Región de decisión correspondiente a un perceptrón simple con dos neuronas de entrada

El Perceptrón Simple

Algoritmo de Aprendizaje (Rosenblatt, 1962)

- Pertenece al grupo de algoritmos fundamentados en la *corrección de errores*
- $\mathbf{x}^r$ conjunto de patrones de entrada, $r = 1, \dots, N$
- c^r conjunto de clases verdaderas de dichos patrones, $r = 1, \dots, N$
- Variables de entrada como las de salida toman dos posibles valores: -1 y $+1$

El Perceptrón Simple

Algoritmo de Aprendizaje (Rosenblatt, 1962)

- Actualización de pesos:
 - Si ante la presentación del r -ésimo patrón la respuesta que proporciona el perceptrón simple es correcta, no actualizaremos los pesos
 - Si la respuesta es incorrecta los pesos se modificarán según la regla de Hebb
- Es decir:

$$\Delta w_{ij}^r(t) = \begin{cases} 2\varepsilon c_i^r x_j^r & \text{si } y_i^r \neq c_i^r \\ 0 & \text{si } y_i^r = c_i^r \end{cases}$$

El Perceptrón Simple

Algoritmo de Aprendizaje (Rosenblatt, 1962)

- La regla anterior se puede reescribir:

$$\Delta w_{ij}^r(t) = \varepsilon (c_i^r - y_i^r) x_j^r$$

- La actualización de pesos únicamente podrá tomar los valores -2ε , 0 y $+2\varepsilon$
- A nivel práctico llegar a un compromiso para el valor del ritmo de aprendizaje, ε
 - Valor pequeño de ε implica un aprendizaje lento
 - Valor excesivamente grande de ε puede conducir a oscilaciones excesivas de los pesos no aconsejables en el proceso de entrenamiento

El Perceptrón Simple

Algoritmo de Aprendizaje (Rosenblatt, 1962)

- El proceso de aprendizaje es *iterativo*
- Configuración sináptica inicial con *pesos pequeños aleatorios*
- Se presentan los *patrones una y otra vez*, con objeto de que los pesos se ajusten iterativamente según la regla anterior:

$$\Delta w_{ij}^r(t) = \varepsilon(c_i^r - y_i^r)x_j^r$$

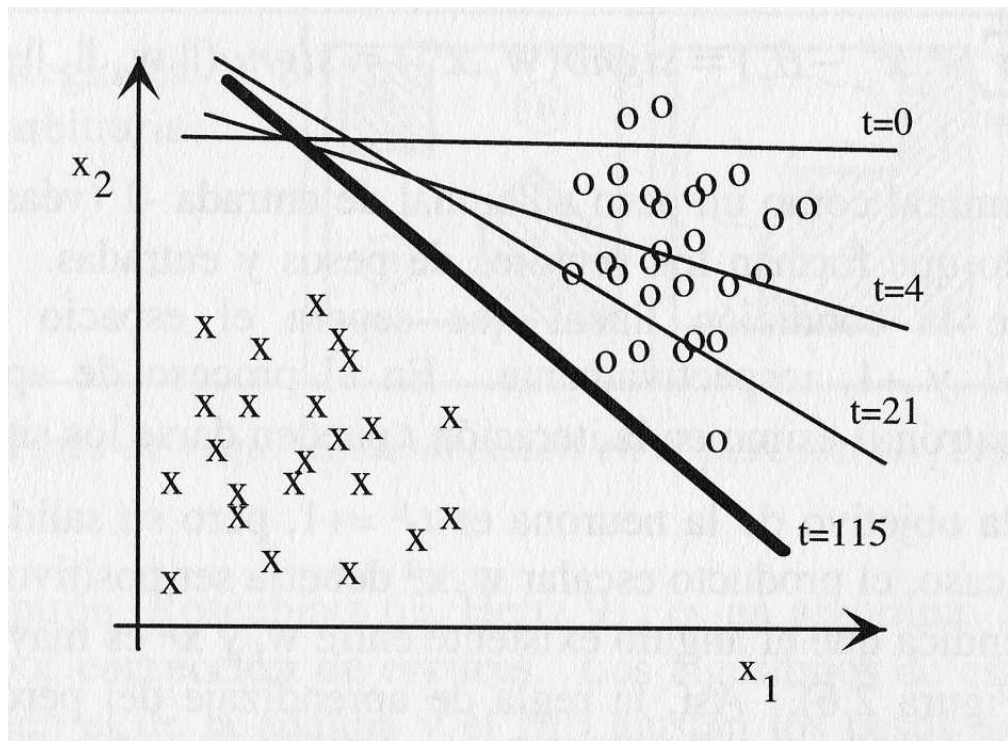
- El *ajuste de los pesos* en la iteración t debido a *todo el conjunto de aprendizaje* será:

$$w_{ij}(t+1) = w_{ij}(t) + \sum_{r=1}^N \Delta w_{ij}^r(t)$$

- Hasta que *todos los patrones queden bien clasificados* si es posible

El Perceptrón Simple

Algoritmo de Aprendizaje (Rosenblatt, 1962)



Evolución de las regiones de decisión establecidas por el perceptrón simple

El Perceptrón Simple

Algoritmo de Aprendizaje (Rosenblatt, 1962)

- Rosenblatt (1962) demostró que:
 - *Si la función a representar es linealmente separable*, el algoritmo anterior siempre converge en un tiempo finito y con independencia de los pesos de partida
 - *Si la función a representar no es linealmente separable*, el proceso de entrenamiento oscilará
- El algoritmo para cuando consigue clasificar correctamente todos los ejemplos
- Puede ocurrir que línea de discriminación quede muy cerca de las muestras de uno de los grupos

La Adalina

ADaptative LInear Neuron (Adalina), Widrow y Hoff (1960)

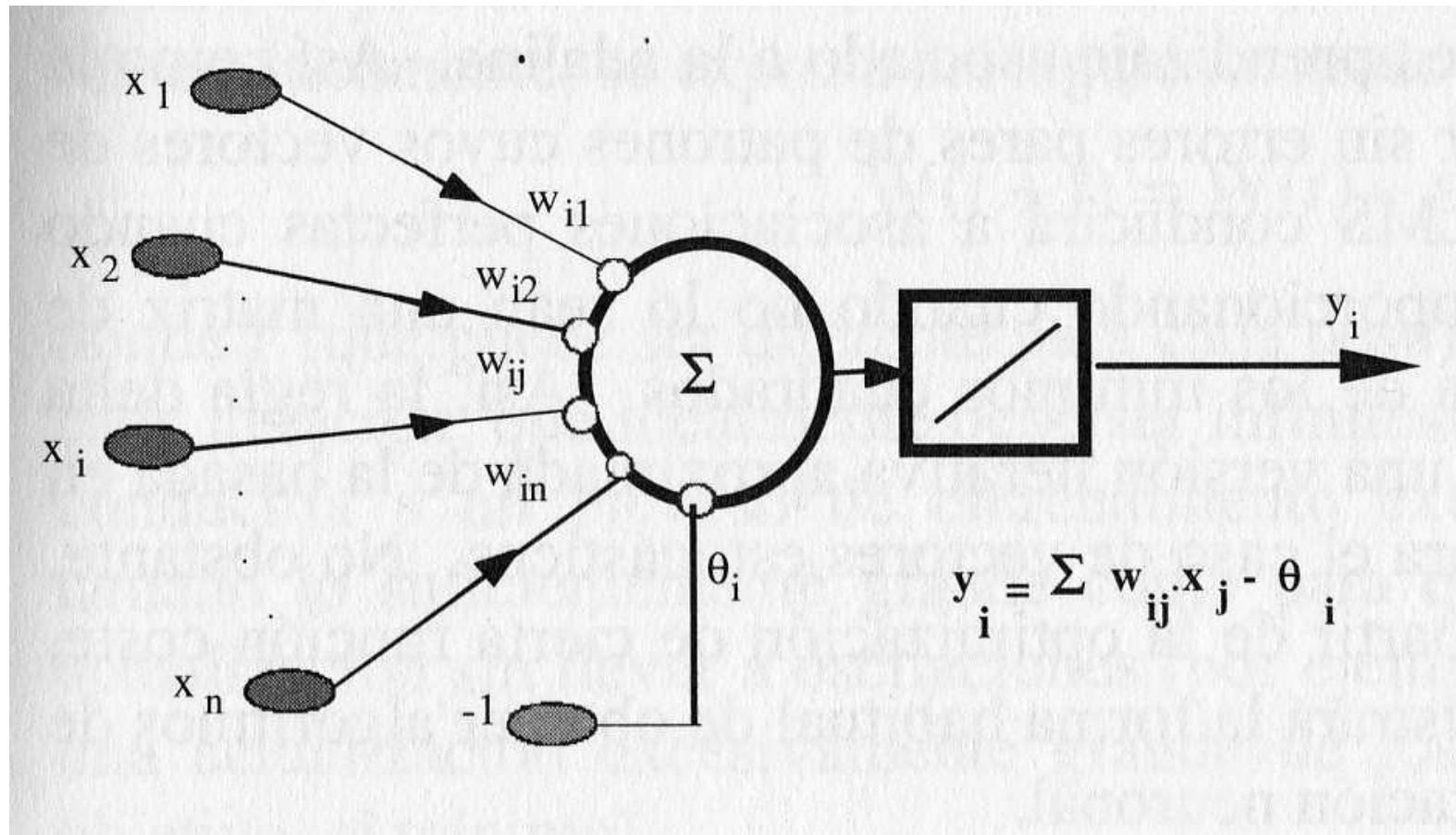
- Entradas pueden ser continuas
- Neurona similar a la del perceptrón simple con función de activación lineal

$$y_i(t) = \sum_{j=1}^n w_{ij}x_j - \theta_i$$

con $i = 1, \dots, m$

- θ_i , *bias*, parámetro que proporciona un grado de libertad adicional al modelo (no umbral de disparo)

La Adalina



Neurona lineal de la Adalina

La Adalina

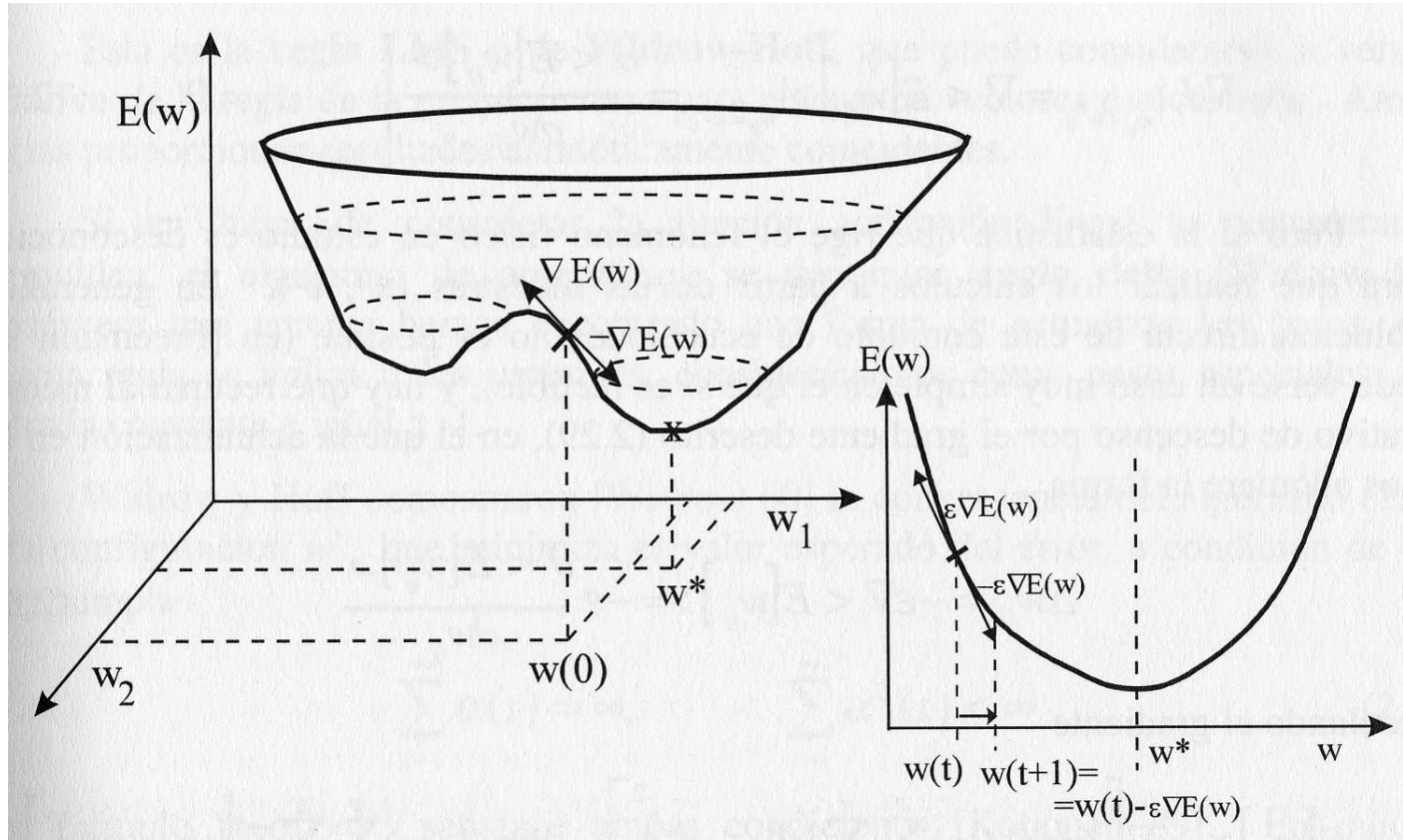
- Incorpora aspectos del asociador lineal y del perceptrón simple
- Diferencia en el aprendizaje
 - Regla actualización de pesos LMS (*least mean square*)
 - Adaptar los pesos de manera proporcional al error cometido

La Adalina

- Aprendizaje de los pesos como un problema de optimización de una determinada función de coste
- Método de optimización es el *descenso por el gradiente* del error actual cometido por la red neuronal
- Optimizador local en un espacio continuo

$$E : \mathbb{R}^{(n \times m) + m} \longrightarrow \mathbb{R}$$
$$w_{11}, \dots, w_{1n}, \dots, w_{m1}, \dots, w_{mn}, \theta_1, \dots, \theta_m \longrightarrow E(w_{11}, \dots, w_{1n}, \dots, w_{m1}, \dots, w_{mn}, \theta_1, \dots, \theta_m)$$

La Adalina



Superficie de error $E(w)$ en el espacio de pesos (izquierda) y método de descenso por el gradiente (derecha) en el que se basa la regla de aprendizaje de la Adalina

La Adalina

Descenso por el gradiente

1. Partir en $t = 0$ de una cierta configuración de peso $\mathbf{w}^0$
2. Calcular la dirección de la máxima variación de la función $E(\mathbf{w})$ en $\mathbf{w}^0$, la cual vendrá dada por su gradiente en $\mathbf{w}^0$
3. Modificar los parámetros $\mathbf{w}$ siguiendo el sentido opuesto al indicado por el gradiente de $E(\mathbf{w})$
4. Iterar los pasos 2 y 3 hasta alcanzar un óptimo local

La Adalina

- Función de coste a minimizar: $E(\mathbf{w}) = \frac{1}{2} \sum_{r=1}^N \sum_{i=1}^m (c_i^r - y_i^r)^2$
- La función de error mide el error cuadrático correspondiente a las salidas actuales de la red respecto de los objetivos
- Teniendo en cuenta que $y_i^r = \sum_{j=1}^n w_{ij} x_j^r - \theta_i$, se obtiene:

$$\frac{\partial E(w_{ij})}{\partial w_{ij}} = - \left(\frac{1}{2} \right) 2 \sum_{r=1}^N (c_i^r - y_i^r) \frac{dy_i^r}{dw_{ij}} = - \sum_{r=1}^N (c_i^r - y_i^r) x_j^r$$

- Regla de adaptación LMS:

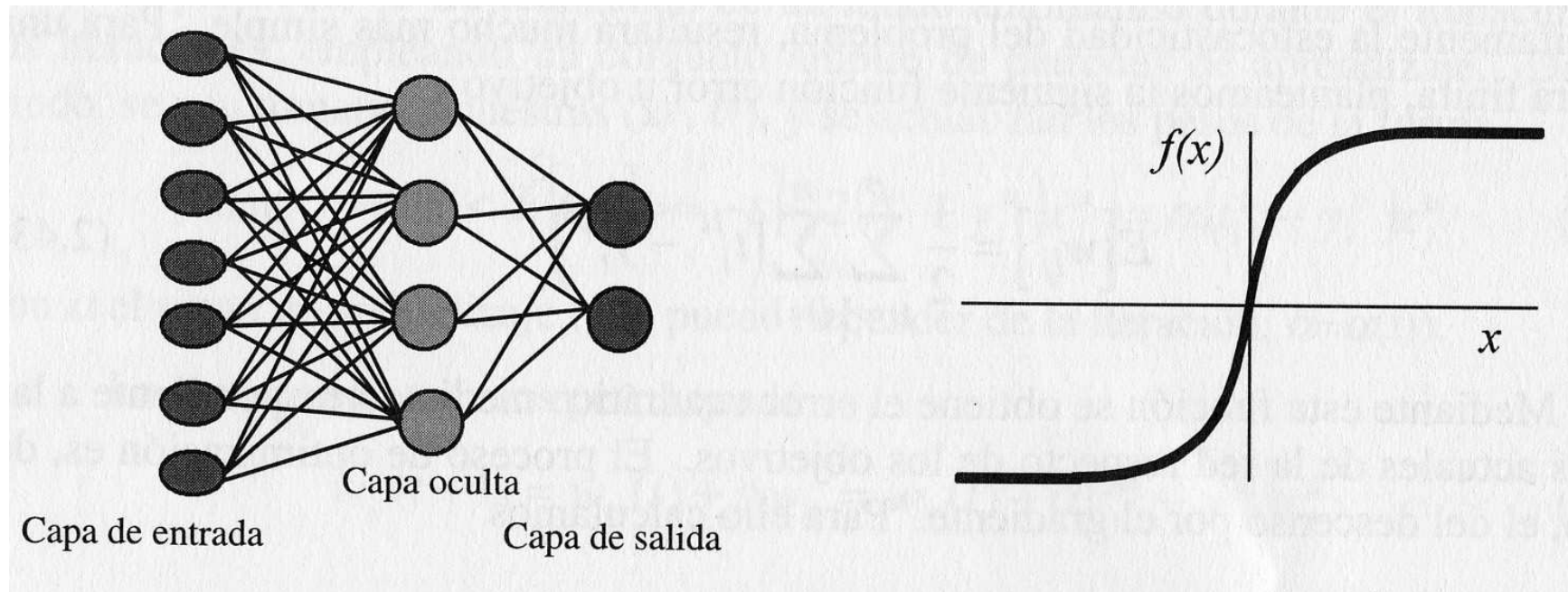
$$\Delta w_{ij} = -\varepsilon \frac{\partial E(w_{ij})}{\partial w_{ij}} = \varepsilon \sum_{r=1}^N (c_i^r - y_i^r) x_j^r$$

- Adalina adaptaciones continuas de los pesos derivadas de la función de activación lineal

El Perceptrón Multicapa

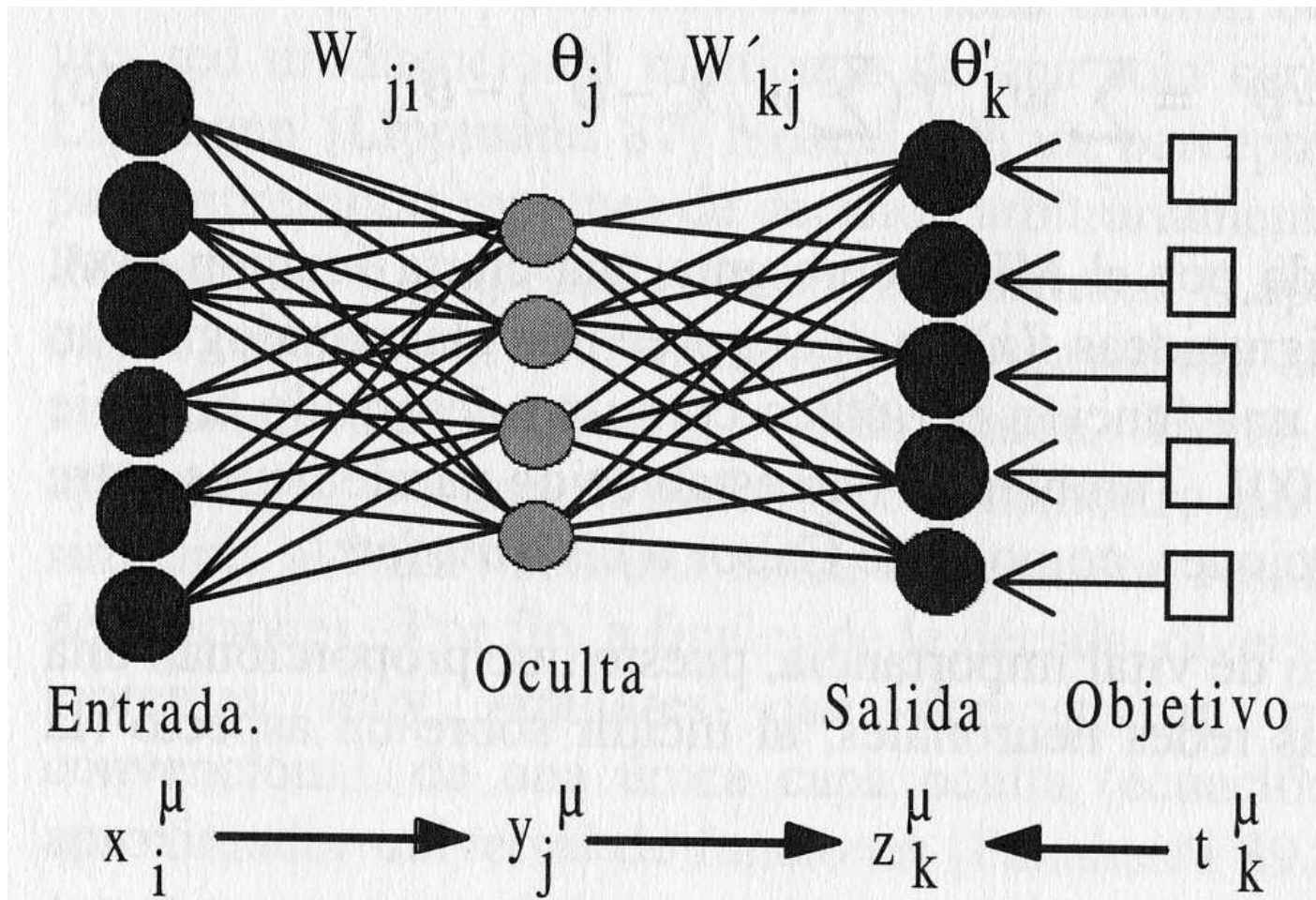
- Perceptrón simple limitaciones, tan sólo discriminar patrones separados por un hiperplano
- Incluyendo capas ocultas superar dichas limitaciones
- Perceptrón multicapa *MultiLayer Perceptron (MLP)*
- Algoritmo de entrenamiento basado en la retropropagación del error *Back-Propagation (BP)* (Werboz, 1974 y Rumelhart y col., 1986)

El Perceptrón Multicapa



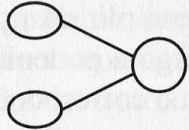
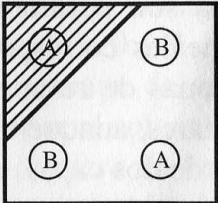
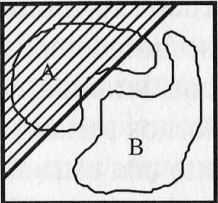
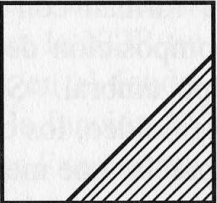
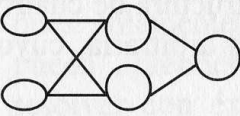
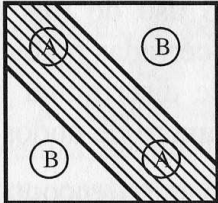
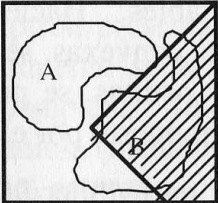
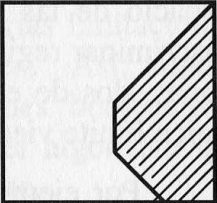
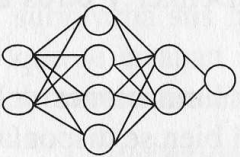
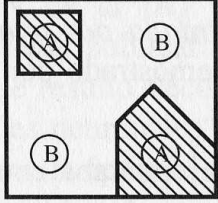
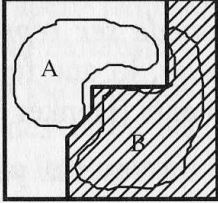
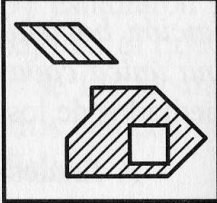
Arquitectura (izquierda) y función de activación (derecha) para el perceptrón multicapa

El Perceptrón Multicapa



Arquitectura del perceptrón multicapa

El Perceptrón Multicapa

Arquitectura	Región de decisión	Ejemplo 1: XOR	Ejemplo 2: clasificación	Regiones más generales
Sin capa oculta 	Hiperplano (dos regiones)			
Una capa oculta 	Regiones polinomiales convexas			
Dos capas ocultas 	Regiones arbitrarias			

Regiones de decisión obtenidas para el perceptrón simple (arriba), el perceptrón multicapa con una capa oculta (en medio) y el perceptrón multicapa en dos capas ocultas (abajo)

El Perceptrón Multicapa

- x_i con $i = 1, \dots, n$ neuronas de entrada de la red
- y_j con $j = 1, \dots, o$ neuronas de la capa oculta
- z_k con $k = 1, \dots, s$ neuronas de la capa final
- c_k con $k = 1, \dots, s$ salidas objetivo (variable clase)
- w_{ij} pesos conectando neuronas de entrada con neuronas de capa oculta
- θ_j umbrales de las neuronas de la capa oculta
- w'_{kj} pesos conectando neuronas de capa oculta con neuronas de salida
- θ'_k umbrales de las neuronas de salida

El Perceptrón Multicapa

- MLP con una única capa oculta
- Función de activación para la capa oculta de tipo sigmoide
- Función de activación para la capa final de tipo lineal

$$z_k = \sum_{j=1}^o w'_{kj} y_j - \theta'_k = \sum_{j=1}^o w'_{kj} f \left(\sum_{i=1}^n w_{ji} x_i - \theta_j \right) - \theta'_k$$

con $k = 1, \dots, s$ y $f(x)$ función de tipo sigmoide

El Perceptrón Multicapa

Retropropagación del error (*Back Propagation (BP)*)

- $E(\mathbf{w}, \mathbf{w}', \boldsymbol{\theta}, \boldsymbol{\theta}') = \frac{1}{2} \sum_{r=1}^N \sum_{k=1}^m (c_k^r - z_k^r)^2$
con $z_k^r = \sum_{j=1}^o w'_{kj} y_j^r - \theta'_k = \sum_{j=1}^o w'_{kj} f \left(\sum_{i=1}^n w_{ji} x_i^r - \theta_j \right) - \theta'_k$
- Minimización por descenso por el gradiente

$$\Delta w'_{kj} = \varepsilon \sum_{r=1}^N \left(c_k^r - \left(\sum_{j=1}^o w'_{kj} y_j^r - \theta'_k \right) \right) y_j^r$$

$$\Delta w_{ji} = \varepsilon \sum_{r=1}^N \Delta_j^r x_i^r$$

$$\text{con } \Delta_j^r = \left(c_k^r - \left(\sum_{j=1}^o w'_{kj} y_j^r - \theta'_k \right) \right) w'_{kj} \frac{\partial f \left(\sum_{i=1}^n w_{ji} x_i^r - \theta_j \right)}{\partial \left(\sum_{i=1}^n w_{ji} x_i^r - \theta_j \right)}$$

El Perceptrón Multicapa

Algoritmo de aprendizaje (*Back Propagation (BP)*)

Paso 1. Establecer aleatoriamente los pesos y umbrales iniciales ($t := 0$)

Paso 2. Para cada patrón r del conjunto de entrenamiento

2.1 Obtener la respuesta de la red frente al patrón r -ésimo

2.2 Calcular las señales de error asociadas $\left(c_k^r - \left(\sum_{j=1}^o w'_{kj} y_j^r - \theta'_k \right) \right)$ y $\left(\Delta_j^r \right)$

2.3 Calcular el incremento parcial de los pesos y umbrales debidos al patrón r

Paso 3. Calcular el incremento total actual, extendido a todos los patrones, de los pesos $\Delta w'_{kj}$ y Δw_{ji} . Hacer lo mismo con los umbrales

Paso 4. Actualizar pesos y umbrales

Paso 5. Calcular el error total

Hacer $t := t + 1$ y volver al Paso 2 si todavía el error total no es satisfactorio

El Perceptrón Multicapa

- Implícito en el BP el concepto de propagación hacia atrás
- $\left(c_k^r - \left(\sum_{j=1}^o w'_{kj} y_j^r - \theta'_k \right) \right)$ señales de error en la salida de la red
se usan para calcular $\Delta w'_{kj}$
- Estas señales de error se propagan hacia atrás, obteniéndose la señal de error en la capa oculta

$$\left(c_k^r - \left(\sum_{j=1}^o w'_{kj} y_j^r - \theta'_k \right) \right) w'_{kj} \frac{\partial f \left(\sum_{i=1}^n w_{ji} x_i^r - \theta_j \right)}{\partial \left(\sum_{i=1}^n w_{ji} x_i^r - \theta_j \right)}$$

- Con esta señal de error se calcula Δw_{ji}