

MODELO DE RED	TOPOLOGÍA	APRENDIZAJE	ASOCIACIÓN AUTOMÁTICO	INFORMAC. DE ENTRADA Y SALIDA	AUTOR(ES)
		ON OFF LINE	SUPERVISADO SUPERVISA	REGLA	
OPTIMAL LINEAR ASSOCIATIVE MEMORY. OLAM	2CAPAS. OFF 1 CAPA CONEX.LAT. AUTO-RECU.	OFF NO SUPERV	HEBBIANO ("OPTIMAL LEAST MEAN SQUARE CORRELATION")	HETEROASOC. AUTOASOC.	ANALOG. WEE 1968 KOHONEN 1973
PERCEPTRON	2 CAPAS Feedforward	OFF SUPERVIS.	CORRECCIÓN ERROR	HETEROASOC	E:ANALOG S:BINARI ROSENBLATT 1958
SHUNTING GROSSBERG, SG	1 CAPA CONEX.LAT. AUTO-RECU.	ON NO SUPERV	HEBBIANO O COMPETITIVO	AUTOASOC.	ANALOG. GROSSBERG 1973
SPARSE DISTRIBUTED MEMORY. SDM	3 CAPAS Feedforward	OFF NO SUPERV	HEBBIANO + RANDOM VECTOR (LVQ) PREPROCESSING	HETEROASOC	BINARIAS KANERVA 1984
TEMPORAL ASSOCIAT. MEMORY. TAM	2 CAPAS FF/FB/FEEDBACK	OFF NO SUPERV	HEBBIANO	HETEROASOC	BINARIAS AMARI 1972
TOPOLOGY PRESERVING MAP. TPM	2 CAPAS FF CONEX. LAT. IMPLIC. AUTOREC.	OFF NO SUPERV	COMPETITIVO	HETEROASOC	ANALOG. KOHONEN 1982

Tabla 3.7.

## CAPÍTULO 4

## REDES NEURONALES CON CONEXIONES HACIA ADELANTE

En este capítulo examinaremos un grupo de redes neuronales que tienen una arquitectura similar. Este es el grupo de las redes con conexiones hacia adelante, las cuales se caracterizan por arquitecturas en niveles y conexiones estrictamente hacia adelante entre las neuronas. Estas redes son todas buenos *clásificadores de patrones* y utilizan *aprendizaje supervisado*.

Este grupo incluye el Perceptrón, las redes ADALINE y MADALINE y la red Back-Propagation. El Perceptrón y las redes ADALINE y MADALINE tienen un importante interés histórico y han abierto el camino para el desarrollo de otras redes neuronales. Por otro lado, la red Back-Propagation es probablemente una de las más utilizadas hoy en día.

### 4.1. EL PERCEPTRÓN

Este fue el primer modelo de red neuronal artificial desarrollado por Rosenblatt en 1958 [Rosenblatt 58]. Despertó un enorme interés en los años 60, debido a su capacidad para aprender a reconocer patrones sencillos: un Perceptrón, formado por varias neuronas lineales para recibir las entradas a la red y una neurona de salida, es capaz de decidir cuándo una entrada presentada a la red pertenece a una de las dos clases que es capaz de reconocer.

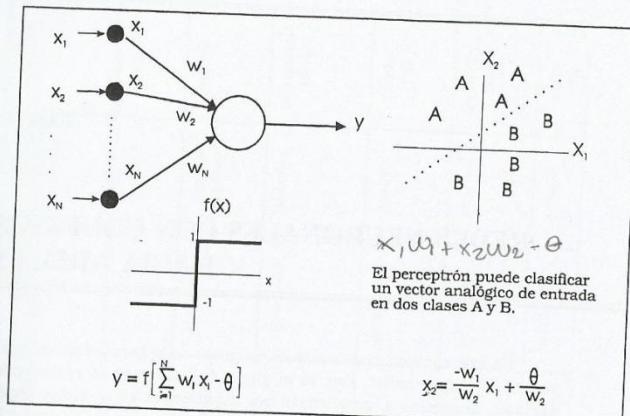


Figura 4.1 El Perceptrón

La única neurona de salida del Perceptrón realiza la suma ponderada de las entradas, resta el umbral y pasa el resultado a una función de trasferencia de tipo escalón. La regla de decisión es responder +1 si el patrón presentado pertenece a la clase A, o -1 si el patrón pertenece a la clase B (Fig. 4.1.). La salida dependerá de la entrada neta (suma de las entradas  $x_i$  ponderadas) y del valor umbral  $\theta$ .

Una técnica utilizada para analizar el comportamiento de redes como el Perceptrón es representar en un mapa las regiones de decisión creadas en el espacio multidimensional de entradas a la red. En estas regiones se visualiza qué patrones pertenecen a una clase y cuáles a otra. El Perceptrón separa las regiones por un hiperplano cuya ecuación queda determinada por los pesos de las conexiones y el valor umbral de la función de activación de la neurona. En este caso, los valores de los pesos pueden fijarse o adaptarse utilizando diferentes algoritmos de entrenamiento de la red.

Sin embargo, el Perceptrón, al constar sólo de una capa de entrada y otra de salida con una única neurona, tiene una capacidad de representación bastante limitada. Este modelo sólo es capaz de discriminar patrones muy sencillos,

linealmente separables. El caso más conocido es la imposibilidad del Perceptrón de representar la función OR-EXCLUSIVA.

La separabilidad lineal limita a las redes con sólo dos capas a la resolución de problemas en los cuales el conjunto de puntos (correspondientes a los valores de entrada) sean separables geométricamente. En el caso de dos entradas, la separación se lleva a cabo mediante una línea recta. Para tres entradas, la separación se realiza mediante un plano en el espacio tridimensional, y así sucesivamente hasta el caso de  $N$  entradas, en el cuál el espacio  $N$ -dimensional es dividido en un hiperplano.

#### 4.1.1. Regla de aprendizaje del Perceptrón

El algoritmo de aprendizaje del Perceptrón es de tipo *supervisado*, lo cual requiere que sus resultados sean evaluados y se realicen las oportunas modificaciones del sistema si fuera necesario. Los valores de los pesos pueden determinar, como se ha dicho, el funcionamiento de la red; estos valores se pueden fijar o adaptar utilizando diferentes algoritmos de entrenamiento de la red. El algoritmo original de convergencia del Perceptrón fue desarrollado por Rosenblatt y lo veremos más adelante. Se pueden usar Perceptrones como máquinas universales de aprendizaje. Desgraciadamente, no puede aprender a realizar todo tipo de clasificaciones: en realidad, sólo se pueden aprender clasificaciones fáciles (problemas de orden 1 en la terminología de Minsky y Papert [Minsky 69]). Esta limitación se debe a que un Perceptrón usa un separador lineal como célula de decisión, con lo cual no es posible realizar sino una sola separación lineal (por medio de un hiperplano).

Como ejemplo de funcionamiento de una red neuronal de tipo Perceptrón, veamos cómo resolver el problema de la función OR. Para esta función, la red debe ser capaz de devolver, a partir de los cuatro patrones de entrada, a qué clase pertenece cada uno. Es decir, para el patrón de entrada 00 debe devolver la clase 0 y para los restantes la clase 1. Para este caso, las entradas serán dos valores binarios. La salida que produce, sin tener en cuenta el valor umbral, es la siguiente:

$$y = f(\sum_i w_i x_i) = f(w_1 x_1 + w_2 x_2)$$

donde

$x_1, x_2$  son las entradas a la neurona (en las neuronas de la capa de entrada, la salida es igual a su entrada).

$w_1, w_2$  son los pesos entre las neuronas de la capa de entrada y la de la capa de salida.

$f$ : Función de salida o transferencia (la función de activación es la función identidad).

Si  $w_1 x_1 + w_2 x_2$  es mayor que 0, la salida será 1, y en caso contrario, será -1 (función de salida en escalón). Como puede observarse, el sumatorio que se le pasa como parámetro (entrada total) a la función  $f$  (función de salida o transferencia) es la expresión matemática de una recta, donde  $w_1$  y  $w_2$  son variables y  $x_1$  y  $x_2$  son las constantes. En la etapa de aprendizaje se irán variando los valores de los pesos obteniendo distintas rectas.

Lo que se pretende al modificar los pesos de las conexiones es encontrar una recta que divida el plano en dos espacios de las dos clases de valores de entrada. Concretamente, para la función OR se deben separar los valores 01, 10 y 11 del valor 00. En este caso, al no existir término independiente en la ecuación porque el umbral  $\theta$  es cero, las posibles rectas pasarán por el origen de coordenadas, por lo que la entrada 00 quedará sobre la propia recta.

Si se pretende resolver el problema de la función AND de la misma manera, se llega a la conclusión de que es imposible si el umbral es cero, ya que no existe ninguna recta que pase por el origen de coordenadas y que separe los valores 00, 01 y 10 de entrada del valor 11, por lo que es necesario introducir un término independiente para poder realizar esta tarea.

Para ello, como se dijo en el capítulo 2, se considera una entrada de valor fijo 1 a través de una conexión con peso  $w_0$ , que representa el umbral ( $w_0 = \theta$ ) y cuyo valor deberá ser ajustado durante la etapa de aprendizaje. Así, el parámetro que se le pasa a la función de transferencia de la neurona

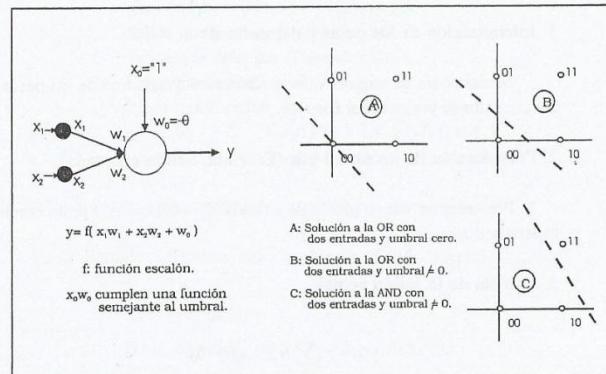


Figura 4.2 Ejemplo del Perceptrón aplicado a la solución de la función OR

queda:  $w_1 x_1 + w_2 x_2 + w_0 l$ , donde  $w_0$  es el término independiente que permitirá desplazar la recta del origen de coordenadas. Si aplicamos esta solución para el caso de la red que calcula la función OR, aumentamos el número de soluciones, ya que, además de las rectas sin término independiente ( $w_0 = 0$ ) que dan solución al problema, existirán otras con término independiente que también lo harán.

En el proceso de entrenamiento, el Perceptrón se expone a un conjunto de patrones de entrada, y los pesos de la red son ajustados de forma que al final del entrenamiento se obtengan las salidas esperadas para cada uno de esos patrones de entrada.

A continuación veremos el algoritmo de convergencia de ajuste de pesos para realizar el aprendizaje de un Perceptrón (aprendizaje por corrección de error) con  $N$  elementos procesales de entrada y un único elemento procesal de salida:

$$\begin{bmatrix} 1 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} 1.5 \\ 0.5 \\ 1.5 \end{bmatrix} = 1.5 \Rightarrow S = f(1.5) = 1$$

Error = (Deseada - Obtenida) = -1

Pesos ( $t+1$ ) = Pesos ( $t$ ) + Error · Entrada

$$\begin{bmatrix} w_0 \\ w_1 \\ w_2 \end{bmatrix} = \begin{bmatrix} 1.5 \\ 0.5 \\ 1.5 \end{bmatrix} + (-1) \cdot \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 0.5 \\ 0.5 \\ 1.5 \end{bmatrix} \Rightarrow \text{Nuevos pesos}$$

b.2.) Se toma el patrón de entrada 01

Entradas:  $x_1 = 0; x_2 = 1; x_0 = 1$

Pesos:  $w_1(t) = 0.5; w_2(t) = 1.5; w_0(t) = 0.5$

Net:  $0 \cdot (0.5) + 1 \cdot (1.5) + 1 \cdot (0.5) = 2$

Salida que produce  $f$  (Obtenida): 1

Salida que debe dar (Deseada): 0

Error que se produce: (Deseada - Obtenida) = 0

Los pesos no se modifican:  $w_i(t+1) = w_i(t)$

b.3.) Puede comprobarse que para las entradas 10 y 11 la salida obtenida es igual que la deseada, por lo que no se varían los pesos. En el caso de que no fuese así, se aplicaría el mismo método que se ha aplicado antes.

Existe un patrón de entrada, 00, para la cual el error cometido no es cero, por tanto, se realiza de nuevo a partir del punto b.

c) Se toman de nuevo los cuatro patrones de entrada:

c.1.) Se toma de nuevo el patrón de entrada 00

Entradas:  $x_1 = 0; x_2 = 0; 1$

Pesos:  $w_1(t) = 0.5; w_2(t) = 1.5; w_0(t) = 0.5$

Net:  $0 \cdot (0.5) + 0 \cdot (1.5) + 1 \cdot (0.5) = 0.5$

Salida que produce  $f$  (Obtenida): 1

Salida que debe dar (Deseada): 0

Error que se comete: -1

Pesos modificados:  $w_1(t+1) = 0.5 + (-1) \cdot 0 = 0.5$

$w_2(t+1) = 1.5 + (-1) \cdot 0 = 1.5$

$w_0(t+1) = 0.5 + (-1) \cdot 1 = -0.5$

En forma matricial:

Entrada · Pesos = net  $\longrightarrow$  Salida =  $f(\text{net})$

$$\begin{bmatrix} 1 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} 0.5 \\ 0.5 \\ 1.5 \end{bmatrix} = 0.5 \Rightarrow S = f(0.5) = 1$$

Error = (Deseada - Obtenida) = 0 - 1 = -1

Pesos ( $t+1$ ) = Pesos ( $t$ ) + Error · Entrada

$$\begin{bmatrix} w_0 \\ w_1 \\ w_2 \end{bmatrix} = \begin{bmatrix} 0.5 \\ 0.5 \\ 1.5 \end{bmatrix} + (-1) \cdot \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} -0.5 \\ 0.5 \\ 1.5 \end{bmatrix} \Rightarrow \text{Nuevos pesos}$$

c.2.) Se toma el patrón de entrada 01

Entradas:  $x_1 = 0; x_2 = 1; 1$

Pesos:  $w_1(t) = 0.5; w_2(t) = 1.5; w_0(t) = -0.5$

Net:  $0 \cdot (0.5) + 1 \cdot (1.5) + 1 \cdot (-0.5) = 1$

Salida que produce  $f$ : 1

Deseada: 1

Error que se comete: 0

No se modifican los pesos

c.3.) Puede comprobarse que para el resto de las entradas, 10 y 11, los pesos no varían.

Sigue habiendo una entrada cuyo error ha sido diferente de cero.

d) Se toman de nuevo los cuatro patrones de entrada:

d.1.) Patrón 00

Entradas:  $x_1 = 0; x_2 = 0; 1$

Pesos:  $w_1(t) = 0,5; w_2(t) = 1,5; w_0(t) = -0,5$   
 $Net: 0 \cdot (0,5) + 0 \cdot (1,5) + 1 \cdot (-0,5) = -0,5$

Salida que produce  $f: 0$   
 deseado: 0

Error que se comete: 0  
 No se varían los pesos

d.2.) Si no han variado los pesos, entonces para el resto de las entradas el error cometido es cero (ver apartados c.2 y c.3).

Con estos nuevos pesos, al calcular la salida que se obtiene para cualquiera de los cuatro patrones de entrada ya no se comete ningún error, por lo que la etapa de aprendizaje concluye.

#### 4.1.2. Solución al problema de la separabilidad lineal

El ejemplo expuesto de ajuste de pesos de una red para solucionar el problema de la función OR no es aplicable, como se dijo anteriormente, a otro problema no trivial, como es la función OR-EXCLUSIVA (XOR). En el caso de esta función se pretende que para los valores de entrada 00 y 11 se devuelva al clase 0, y para los patrones 01 y 10, la clase 1. Como puede comprobarse en la figura 4.3, el problema radica en que no existe ninguna recta que separe los patrones de una clase de los de la otra.

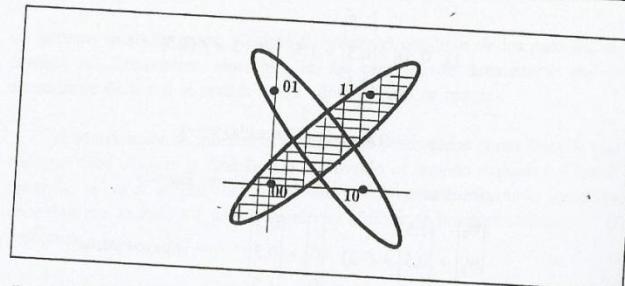


Figura 4.3 Función XOR. No es posible obtener una recta que separe las dos clases

La solución podría darse si descomponiéramos el espacio en tres regiones: una región pertenecería a una de las clases de salida y las otras dos pertenecerían a la segunda clase. Si en lugar de utilizar únicamente una neurona de salida se utilizaran dos, se obtendrían dos rectas, por lo que podrían delimitarse tres zonas. Para poder elegir entre una zona u otra de las tres, es necesario utilizar otra capa con una neurona cuyas entradas serán las salidas de las neuronas anteriores. Las dos zonas o regiones que contienen los puntos (0,0) y (1,1) se asocian a una salida nula de la red, y la zona central se asocia a la salida con valor 1. De esta manera, es posible encontrar una solución al problema de la función XOR.

Por tanto, se ha de utilizar una red de tres neuronas, distribuidas en dos capas, para solucionar el problema de la función XOR. Como ejemplo, el lector puede comprobar que se obtienen los resultados mostrados en la figura 4.4, tomando los siguientes valores:

$$\begin{array}{ll} w_{11} = 1 & w_{12} = 1 \\ w_{21} = 1 & w_{22} = 1 \\ w_{31} = 1 & w_{32} = -1,5 \\ \theta_1 = 0,5 & \theta_2 = 1,5 \quad \theta_3 = 0,5 \end{array}$$

aplicando la expresión obtenida anteriormente para cada neurona:

$$\begin{aligned}y_1 &= f(w_{11}x_1 + w_{12}x_2 - \theta_1) = f(x_1 + x_2 - 0.5) \\y_2 &= f(w_{21}x_1 + w_{22}x_2 - \theta_2) = f(x_1 + x_2 - 1.5) \\y_3 &= f(w_{31}x_1 + w_{32}x_2 - \theta_3) = f(y_1 - 1.5y_2 - 0.5)\end{aligned}$$

$x_1$	$x_2$	$y_1$	$y_2$	$y_3 = XOR$
0	0	$f(-0.5) = 0$	$f(-1.5) = 0$	$f(-0.5) = 0$
0	1	$f(0.5) = 1$	$f(-0.5) = 0$	$f(0.5) = 1$
1	0	$f(0.5) = 1$	$f(-0.5) = 0$	$f(0.5) = 1$
1	1	$f(1.5) = 1$	$f(0.5) = 1$	$f(1.5) = 1$

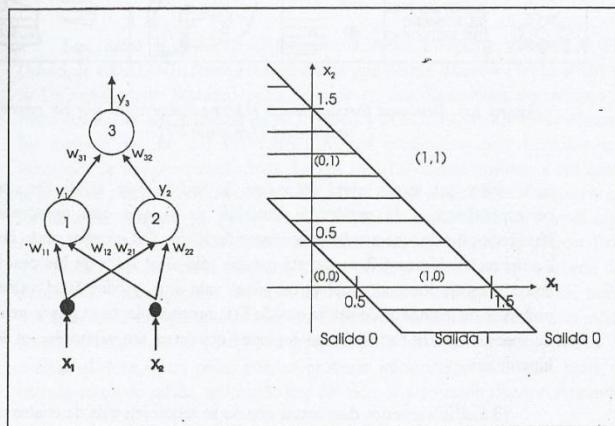


Figura 4.4 Solución del problema de la función XOR

Hay que indicar que para el caso de la XOR se tienen que ajustar seis pesos (sin incluir las conexiones que representan los umbrales). En el caso de los pesos de las conexiones de la capa de salida ( $w_{31}$  y  $w_{32}$ ), el ajuste de los pesos se realiza de forma idéntica a la estudiada anteriormente, pues conocemos la salida deseada. Sin embargo, no se tiene porque conocer cuál debe ser la

salida deseada de las células de la capa oculta, por lo que el método utilizado en la función OR no es aplicable en la función XOR. La solución para el aprendizaje en este tipo de redes, donde existen niveles ocultos, se estudia en los siguientes apartados.

#### 4.2. EL PERCEPTRON MULTINIVEL

Un Perceptrón multinivel o multicapa es una red de tipo *feedforward* compuesta de varias capas de neuronas entre la entrada y la salida de la misma. Esta red permite establecer regiones de decisión mucho más complejas que las de dos semiplanos, como hacía el Perceptrón de un solo nivel.

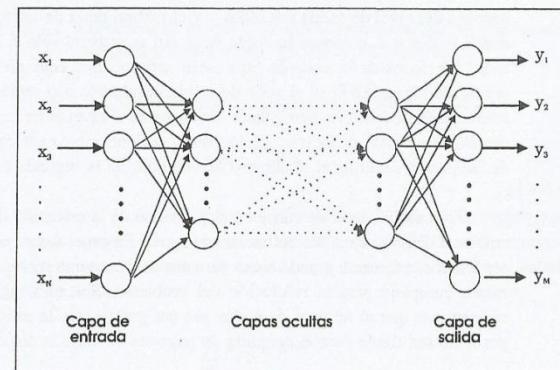


Figura 4.5 Perceptrón multinivel (red feedforward multicapa)

Las capacidades del Perceptrón con dos, tres y cuatro niveles o capas y con una única neurona en el nivel de salida, se muestra en la figura 4.6. En la segunda columna se muestra el tipo de región de decisión que se puede formar con cada una de las configuraciones. En la siguiente columna se indica el tipo

El número de nodos de la 3<sup>a</sup> capa ( $N_3$ ) debe ser mayor que uno cuando las regiones de decisión están desconectadas o *endentadas* y no se pueden formar con una región convexa. Este número, en el peor de los casos, es igual al número de regiones desconectadas en las distribuciones de entrada. El número de neuronas en la 2<sup>a</sup> capa ( $N_2$ ) normalmente debe ser suficiente para proveer tres o más ángulos por cada área convexa generada por cada neurona de la 3<sup>a</sup> capa. Así, deberá de haber más de tres veces el número de neuronas de la 3<sup>a</sup> capa ( $N_2 > 3N_3$ ). En la práctica, un número de neuronas excesivo en cualquier capa puede generar ruido. Por otro lado, si existe un número de neuronas redundantes se obtiene mayor tolerancia a fallos.

### 4.3 LAS REDES ADALINE Y MADALINE

Las redes ADALINE (ADaptive LINEar Element) y MADALINE (Multiple ADALINE) fueron desarrolladas por Berné Widrow [Widrow 60] en la Universidad de Stanford poco después de que Rosenblatt desarrollara el Perceptrón. Las arquitecturas de ADALINE y MADALINE son esencialmente las mismas que las del Perceptrón. Ambas estructuras usan neuronas con funciones de transferencia escalón. La red ADALINE está limitada a una única neurona de salida, mientras que MADALINE puede tener varias. La diferencia fundamental respecto al Perceptrón se refiere al mecanismo de aprendizaje. ADALINE y MADALINE utilizan la denominada *regla Delta* de Hidrow-Hoff o regla del mínimo error cuadrado medio (LMS), basada en la búsqueda del mínimo de una expresión del error entre la salida deseada y la salida lineal obtenida antes de aplicarle la función de activación escalón (frente a la salida binaria utilizada en el caso del Perceptrón). Debido a esta nueva forma de evaluar el error, estas redes pueden procesar información analógica, tanto de entrada como de salida, utilizando una función de activación lineal o sigmoidal

En cuanto a la estructura de la red ADALINE, que es casi idéntica a la del Perceptrón elemental, sus autores la consideran formada por un elemento denominado *combinador adaptativo lineal* (ALC), que obtiene una salida lineal ( $s$ ) que puede ser aplicada a otro elemento de commutación bipolar, de forma que si la salida del ALC es positiva, la salida de la red ADALINE es +1; si la salida del ALC es negativa, entonces la salida de la red ADALINE es -1 (Figura 4.7.).

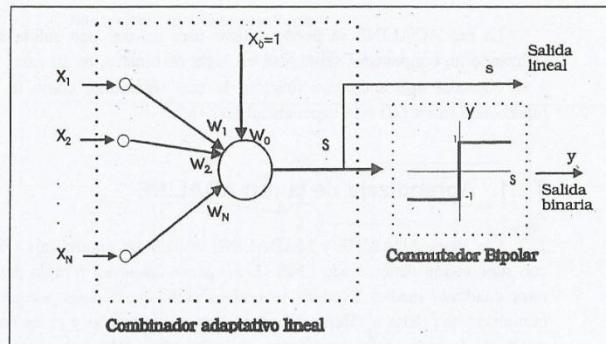


Figura 4.7 Estructura de la red ADALINE, compuesta por un combinador adaptativo lineal y una función de salida bipolar

El ALC realiza el cálculo de la suma ponderada de las entradas:

$$s = w_0 + \sum_{j=1}^N w_j x_j$$

Como en el caso del Perceptrón, el umbral de la función de transferencia se representa a través de una conexión flichticia de peso  $w_0$ . Si tenemos en cuenta que para esta entrada se toma el valor de  $x_0 = 1$ , se puede escribir la anterior ecuación de la forma:

$$s = \sum_{j=0}^N w_j x_j = X \cdot W^T$$

Esta es la salida lineal que genera el ALC. La salida binaria correspondiente de la red ADALINE es, por tanto:

$$y(t+1) = \begin{cases} +1 & s > 0 \\ y(t) & s = 0 \\ -1 & s < 0 \end{cases}$$

La red ADALINE se puede utilizar para generar una salida analógica utilizando un conmutador sigmoidal, en lugar de binario; en tal caso, la salida y se obtendrá aplicando una función de tipo sigmoidal, como la tangente hiperbólica ( $\tanh(s)$ ) o la exponencial ( $(I/I+e^x)$ )

#### 4.3.1. Aprendizaje de la red ADALINE

Las redes ADALINE y MADALINE utilizan un aprendizaje OFF LINE con supervisión denominado LMS (*Least Mean Squared*) o regla del mínimo error cuadrado medio. También se conoce como *regla delta* porque trata de minimizar una delta o diferencia entre el valor observado y el deseado en la salida de la red; como ocurre con el Perceptrón, sólo que ahora la salida considerada es el valor previo a la aplicación de la función de activación de la neurona o, si se prefiere, la salida obtenida al aplicar una función de activación lineal.

*Función de activación función tanh*

La regla de aprendizaje de mínimos cuadrados (*Least Mean Square*) es un método para hallar el vector de pesos  $W$  deseado, el cuál deberá ser único y asociar con éxito cada vector del conjunto de vectores o patrones de entrada  $\{X^1, X^2, X^3, \dots, X^L\}$  con su correspondiente valor de salida correcto (o deseado)  $d_k$ ,  $k=1, \dots, L$ . Nótese que el problema de hallar un conjunto de pesos  $W$  que para un único vector de entrada  $X$  dé lugar a un valor de salida correcto resulta sencillo, lo que no ocurre cuando se dispone de un conjunto de vectores de entrada, cada uno con su propio valor de salida asociado. El entrenamiento de la red consiste en adaptar los pesos a medida que se vayan presentando las combinaciones entrada-salida se realiza un proceso automático de pequeños ajustes en los valores de los pesos hasta que se obtienen las salidas correctas.

La primera cuestión que debemos resolver es la de definir qué significa obtener el mejor vector de pesos obtenido a partir de unas parejas de valores ejemplo  $(X^i, d^i)$  de forma que, una vez encontrado, desearemos que al aplicar todos los vectores de entrada se obtenga como resultado el valor de salida correcto. Como vimos en apartados anteriores, se trata de eliminar o, por lo menos, minimizar la diferencia entre salida deseada y salida real para todos los vectores de entrada.

Concretamente, la regla de aprendizaje LMS minimiza el error cuadrado medio, definido como:

$$\langle e_k^2 \rangle = \frac{1}{2L} \sum_{k=1}^L e_k^2$$

donde  $L$  es el número de vectores de entrada (patrones) que forman el *conjunto de entrenamiento*, y  $e_k$  la diferencia entre la salida deseada y la obtenida cuando se introduce el patrón  $k$ -ésimo, que, en el caso de la red ADALINE, se expresa como  $e_k = (d_k - s_k)$ , siendo  $s_k$  la salida del ALC; es decir:

$$s_k = X_k \cdot W^T = \sum_{j=0}^N w_j x_{kj}$$

La función de error es una función matemática definida en el espacio de pesos multidimensional para un conjunto de patrones dados. Es una superficie que tendrá muchos mínimos (global y locales), y la regla de aprendizaje va a buscar el punto en el espacio de pesos donde se encuentra el mínimo global de esta superficie. Aunque la superficie de error es desconocida, el método de gradiente decreciente consigue obtener información local de dicha superficie a través del gradiente. Con esta información se decide qué dirección tomar para llegar hasta el mínimo global de dicha superficie.

Basándose en el método del gradiente decreciente, se obtiene una regla (regla delta o regla LMS) para modificar los pesos de tal manera que hallamos un nuevo punto en el espacio de pesos más próximo al punto mínimo. Es decir, las modificaciones en los pesos son proporcionales al gradiente decreciente de la función error  $\Delta w_i = -\alpha (\partial e_i / \partial w_i)$ . Por tanto, se deriva la función error con respecto a los pesos para ver cómo varía el error con el cambio de los pesos.

Aplicamos la regla de la cadena para el cálculo de dicha derivada:

$$\Delta w_i = -\alpha \frac{\partial \langle e_k^2 \rangle}{\partial w_i} = -\alpha \frac{\partial \langle e_k^2 \rangle}{\partial s_k} \cdot \frac{\partial s_k}{\partial w_i}$$

Se calcula la primera derivada:

$$\frac{\partial \langle e_k^2 \rangle}{\partial s_k} = \frac{\partial \left[ \frac{1}{2} (d_k - s_k)^2 \right]}{\partial s_k} = \frac{1}{2} [2 (d_k - s_k)(-1)] = - (d_k - s_k) = - \epsilon_k$$

por tanto, queda:

$$\frac{\partial \langle e_k^2 \rangle}{\partial s_k} = - \epsilon_k$$

Teniendo en cuenta que  $s_k$  es la salida lineal:

$$s_k = \sum_{j=0}^N w_j x_{k_j}$$

calculamos la segunda derivada de la expresión de  $\Delta w_i$ :

$$\frac{\partial s_k}{\partial w_i} = \frac{\partial (\sum_{j=0}^N (w_j x_{k_j}))}{\partial w_i} = \frac{\partial (w_i x_{k_i})}{\partial w_i} = \hat{y}_{k_i}$$

Así pues, el valor del gradiente del error producido por un patrón dado ( $k$ ) es:

$$\frac{\partial \langle e_k^2 \rangle}{\partial w_i} = - \epsilon_k x_{k_i}$$

Las modificaciones en los pesos son proporcionales al gradiente descendente de la función error:

$$\Delta w_i = - \alpha (- \epsilon_k x_{k_i}) = \alpha \epsilon_k x_{k_i} = \alpha (d_k - s_k) x_{k_i}$$

$$w_i(t+1) = w_i(t) + \alpha (d_k - s_k) x_{k_i}$$

siendo  $\alpha$  la constante de proporcionalidad o tasa de aprendizaje.

En notación matricial, quedaría:

$$W(t+1) = W(t) + \alpha \epsilon_k X_k = W(t) + \alpha (d_k - s_k) X_k$$

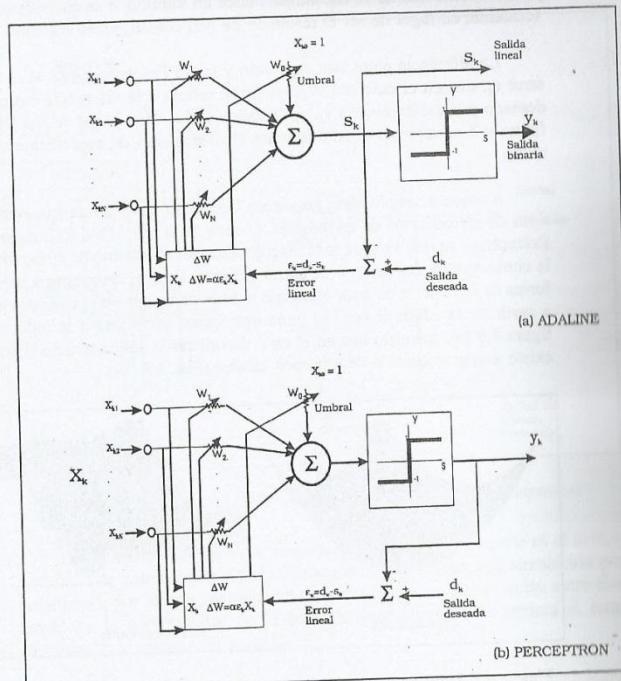


Figura 4.8. Ajuste de los pesos durante el aprendizaje en las redes ADALINE (a) y Perceptrón (b) cuando ante un patrón de aprendizaje  $x_k$  se desea una salida  $d_k$  [Widrow 90]

Esta expresión representa la modificación de pesos obtenida al aplicar el algoritmo LMS, y es parecida a la obtenida anteriormente para el caso del Perceptrón.  $\alpha$  es el parámetro que determina la estabilidad y la velocidad de aprendizaje.

convergencia del vector de pesos hacia el valor de error mínimo. Los cambios en dicho vector deben hacerse relativamente pequeños en cada iteración, sino podría ocurrir que no se encontrase nunca un mínimo, o se encontrase sólo por accidente, en lugar de ser el resultado de una convergencia sostenida hacia él.

La diferencia entre esta expresión y la del Perceptrón está el en valor del error  $\epsilon_k$ , que en el caso del Perceptrón se refería a la diferencia entre el valor deseado y la salida binaria  $y_k$ , no la salida lineal  $s_k$  de la red ADALINE. En la figura 4.8 se representa gráficamente el mecanismo de aprendizaje de ambas redes.

Aunque a simple vista no parece que exista gran diferencia entre ambos tipos de mecanismos de aprendizaje, el caso de la red ADALINE mejora al del Perceptrón, ya que va a ser más sencillo alcanzar el mínimo de error, facilitando la convergencia del proceso de entrenamiento. Esto se demuestra a través de la forma de la función de error cuadrado medio  $\langle \epsilon^2 \rangle$ , que en el caso de calcularse a partir de la salida lineal ( $s_k$ ) tiene una forma semejante a la indicada en la figura 4.9 (a), mientras que en el caso de utilizar la salida binaria (Perceptrón) existe una gran cantidad de mínimos locales (Fig. 4.9 (b)).

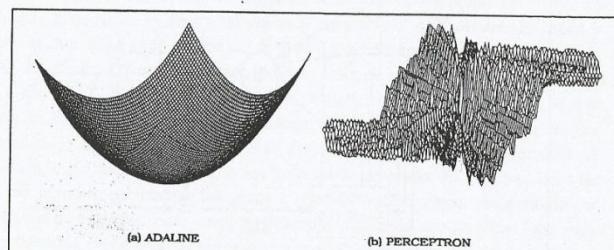


Figura 4.9 Función de error cuadrado medio  $\langle \epsilon^2 \rangle$  cuando se utiliza para su evaluación la salida lineal  $s_k$  (a) o la salida binaria (b) [Widrow 90]

Existe una situación intermedia que consistiría en utilizar lo que se conoce como salida sigmoidal, en lugar de la salida lineal de ADALINE o la binaria del Perceptrón. En tal caso, la función de activación sería del tipo sigmoidal (Figura 4.10 (a)) y la superficie de error tendría la forma indicada en la figura 4.10 (b), con un gran mínimo global (como en el caso de ADAINE) y varios mínimos locales (aunque en menor medida que el Perceptrón).

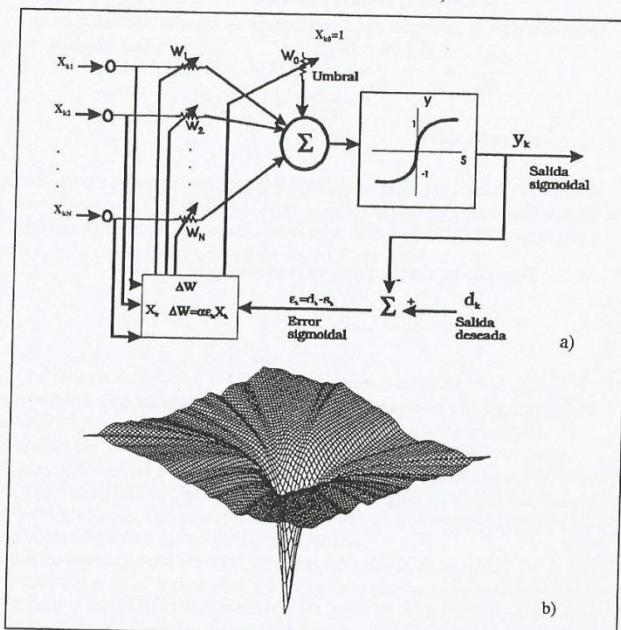


Figura 4.10 Ajuste de los pesos utilizando la salida sigmoidal (a) y su efecto sobre la función de error cuadrado medio  $\langle \epsilon^2 \rangle$  (b) [Widrow 90]

La aplicación del proceso iterativo de aprendizaje (algoritmo de aprendizaje de una red ADALINE) consta de los siguientes pasos:

1. Se aplica un vector o patrón de entrada,  $X_k$ , en las entradas del ADALINE.
2. Se obtiene la salida lineal  $s_k = X_k \cdot W^T = \sum_{j=0}^N w_j x_j$  y se calcula la diferencia con respecto a la deseada  $\epsilon_k = d_k - s_k$

$d_k$

## 3. Se actualizan los pesos

$$W(t+1) = W(t) + \alpha e_k X_k$$

$$w(t+1) = w_j(t) + \alpha e_k x_{k_j}$$

4. Se repiten los pasos del 1 al 3 con todos los vectores de entrada ( $L$ ).

## 5. Si el error cuadrado medio:

$$\langle e_k^2 \rangle = \frac{1}{2L} \sum_{k=1}^L e_k^2$$

es un valor reducido aceptable, termina el proceso de aprendizaje; sino, se repite otra vez desde el paso 1 con todos los patrones.

Cuando se utiliza una red ADALINE para resolver un problema concreto, es necesario determinar una serie de aspectos prácticos, como el número de vectores de entrenamiento necesarios, hallar la forma de generar la salida deseada para cada vector de entrenamiento, o la dimensión óptima del vector de pesos, o cuáles deberían ser los valores iniciales de los pesos, así como si es necesario o no un valor umbral  $\theta$ , o cuál debe ser el valor de  $\alpha$ , o cuándo se debe finalizar el entrenamiento, etc. En general, la solución de estas cuestiones depende del problema concreto que se pretenda resolver, por lo que no se pueden dar respuestas genéricas concretas.

Respecto al número de componentes del vector de pesos, si el número de entradas está bien definido, entonces habrá un peso por cada entrada, con la opción de añadir o no un peso para la entrada del umbral. Incluir este término puede ayudar a la convergencia de los pesos proporcionando un grado de libertad adicional.

La solución es diferente cuando sólo se dispone de una señal de entrada. En estos casos, la aplicación más común es el *filtro adaptativo* para, por ejemplo, eliminar el ruido de la señal de entrada, la cual se muestrea en varios instantes de tiempo, de forma que cada muestra representa un grado de libertad

que se utiliza para ajustar la señal de entrada a la salida deseada (Fig. 4.11). La idea consiste en utilizar el menor número de muestras (así obtenemos una convergencia más rápida) siempre que se obtengan resultados satisfactorios.

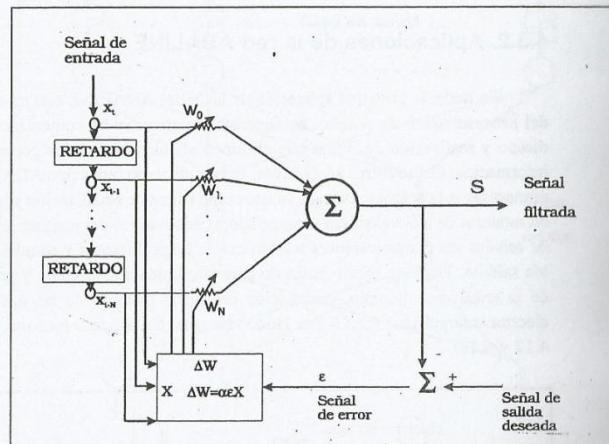


Figura 4.11 Red ADALINE utilizada como filtro digital adaptativo

La dimensión del vector de pesos tiene una influencia directa en el tiempo necesario de entrenamiento (sobre todo cuando se realiza una simulación por ordenador), por lo que generalmente se debe tomar un compromiso entre este aspecto y la aceptabilidad de la solución (normalmente, se mejora el error aumentando el número de pesos).

*nota de apunte*  
El valor del parámetro  $\alpha$  tiene una gran influencia sobre el entrenamiento. Si  $\alpha$  es demasiado grande, la convergencia es posible que no se produzca, debido a que se darán saltos en torno al mínimo sin alcanzarlo. Si  $\alpha$  es demasiado pequeño, alcanzaremos la convergencia, pero a costa de una etapa de aprendizaje más larga.

En cuanto al momento en el que debemos detener el entrenamiento, éste depende, sobre todo, de los requisitos de salida del sistema: se detiene el

basada en un principio de mínima perturbación [Widrow 88]. El entrenamiento equivale a hacer una reducción del número de neuronas de salida incorrectos para cada una de las tramas de entrenamiento que se den como entrada (nótese que la salida de la red es una serie de unidades bipolares). Este método se puede aplicar mediante el siguiente algoritmo [Freeman 91]:

1. Se aplica un vector a las entradas de MADALINE y se hace que se propague hasta las unidades de salida.
2. Se cuenta el número de valores incorrectos que hay en la capa de salida, denominándose error a dicho número.
3. Para las unidades de la capa de salida:
  - Se selecciona la primera neurona que no haya sido seleccionada antes y cuya salida lineal esté más próxima a cero. Ésta es la neurona que puede cambiar su salida binaria con el menor cambio de sus pesos y, según el principio de mínima perturbación, debe tener prioridad en el proceso de aprendizaje.
  - Se cambian los pesos de la neurona seleccionada de tal modo que cambie su salida binaria.
  - Se hace que se propague el vector de entrada hacia adelante, partiendo de las entradas y en dirección a las salidas, una vez más.
  - Se admite el cambio de pesos si ha dado lugar a una reducción del error; en caso contrario, se restauran los pesos originales.
4. Se repite el paso 3 para todas las capas, salvo la de salida.
5. Para todas las unidades de la capa de salida:
  - Se selecciona el par de neuronas que no hayan sido seleccionadas anteriormente y cuyas salidas lineales estén más próximas a cero.
  - Se aplica una corrección de pesos a ambas neuronas para modificar el valor de su salida.
  - Se hace que se propague hacia adelante el vector de entradas, desde las entradas hasta las salidas.
  - Se admite el cambio de pesos si ha dado lugar a una reducción del error; en caso contrario, se restauran los pesos originales.
6. Se repite el paso 5 para todas las capas, salvo la de entrada.

Los pasos 5 y 6 se pueden repetir con grupos de tres, cuatro o mayor número de neuronas hasta obtener resultados satisfactorios. Se considera que las parejas son apropiadas para redes que tengan un máximo de 25 neuronas por capa, aproximadamente.

#### 4.4. LA RED BACKPROPAGATION

En 1986, Rumelhart, Hinton y Williams [Rummelhart 86], basándose en los trabajos de otros investigadores [Werbos 74][Parker 82] formalizaron un método para que una red neuronal *aprendiera* la asociación que existe entre los patrones de entrada a la misma y las clases correspondientes, utilizando más niveles de neuronas que los que utilizó Rosenblatt para desarrollar el Perceptrón. Este método, conocido en general como *backpropagation* (propagación del error hacia atrás), está basado en la generalización de la regla delta y, a pesar de sus propias limitaciones, ha ampliado de forma considerable el rango de aplicaciones de las redes neuronales.

El algoritmo de propagación hacia atrás, o retropropagación, es una regla de aprendizaje que se puede aplicar en modelos de redes con más de dos capas de células. Una característica importante de este algoritmo es la representación interna del conocimiento que es capaz de organizar en la capa intermedia de las células para conseguir cualquier correspondencia entre la entrada y la salida de la red. Ya se ha mostrado en este capítulo que en muchos casos, como la resolución del problema de la OR exclusiva, es imposible encontrar los pesos adecuados para establecer la correspondencia entre la entrada y la salida mediante una red sin capas intermedias. Con una capa de neuronas ocultas, sí es posible establecer dicha correspondencia.

De forma simplificada, el funcionamiento de una red *backpropagation* (*backpropagation net*, *BPN*) consiste en un aprendizaje de un conjunto predefinido de pares de entradas-salidas dados como ejemplo, empleando un ciclo *propagación-adaptación* de dos fases: primero se aplica un patrón de entrada como estímulo para la primera capa de las neuronas de la red, se va propagando a través de todas las capas superiores hasta generar una salida, se compara el resultado obtenido en las neuronas de salida con la salida que se desea obtener y se calcula un valor del error para cada neurona de salida. A continuación, estos errores se transmiten hacia atrás, partiendo de la capa de

salida, hacia todas las neuronas de la capa intermedia que contribuyan directamente a la salida, recibiendo el porcentaje de error aproximado a la participación de la neurona intermedia en la salida original. Este proceso se repite, capa por capa, hasta que todas las neuronas de la red hayan recibido un error que describa su aportación relativa al error total. Basándose en el valor del error recibido, se reajustan los pesos de conexión de cada neurona, de manera que en la siguiente vez que se presente el mismo patrón, la salida esté más cercana a la deseada; es decir, el error disminuya.

La importancia de la red *backpropagation* consiste en su capacidad de autoadaptar los pesos de las neuronas de las capas intermedias para *aprender* la relación que existe entre un conjunto de patrones dados como ejemplo y sus salidas correspondientes. Para poder aplicar esa misma relación, después del entrenamiento, a nuevos vectores de entrada con ruido o incompletas, dando una salida activa si la nueva entrada es parecida a las presentadas durante el aprendizaje. Esta característica importante, que se exige a los sistemas de aprendizaje, es la capacidad de *generalización*, entendida como la facilidad de dar salidas satisfactorias a entradas que el sistema no ha visto nunca en su fase de entrenamiento. La red debe encontrar una *representación interna* que le permita generar las salidas deseadas cuando se le dan las entradas de entrenamiento, y que pueda aplicar, además, a entradas no presentadas durante la etapa de aprendizaje para clasificarlas según las características que comparten con los ejemplos de entrenamiento.

#### 4.4.1. La regla delta generalizada

La regla propuesta por Widrow en 1960 (regla delta) ha sido extendida a redes con capas intermedias (regla delta generalizada) con conexiones hacia adelante (*feedforward*) y cuyas células tienen funciones de activación continuas (lineales o sigmoidales), dando lugar al algoritmo de retropropagación (*backpropagation*). Estas funciones continuas son no decrecientes y derivables. La función sigmoidal pertenece a este tipo de funciones, a diferencia de la función escalón que se utiliza en el Perceptrón, ya que esta última no es derivable en el punto en el que se encuentra la discontinuidad.

Este algoritmo utiliza también una función o superficie de error asociada a la red, buscando el estado estable de mínima energía o de mínimo error a

través del camino descendente de la superficie del error. Por ello, realimenta el error del sistema para realizar la modificación de los pesos en un valor proporcional al gradiente decreciente de dicha función de error.

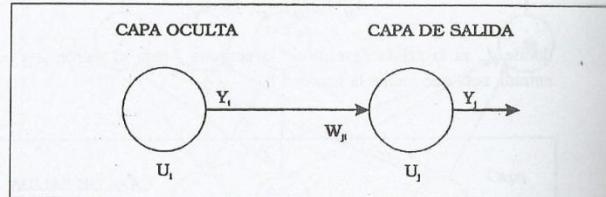


Figura 4.16 Conexión entre una neurona de una capa oculta con una neurona de salida

##### 4.4.1.1. Funcionamiento del algoritmo

El método que sigue la regla delta generalizada para ajustar los pesos es exactamente el mismo que el de la regla delta utilizada en el Perceptrón y ADALINE; es decir, los pesos se actualizan de forma proporcional a la delta, o diferencia entre la salida deseada y la obtenida ( $\delta = \text{sal. deseada} - \text{sal. obtenida}$ ).

Dada una neurona (unidad  $U_i$ ) y la salida que produce,  $y_i$  (Fig. 4.16), el cambio que se produce en el peso de la conexión que une la salida de dicha neurona con la unidad  $U_j$  ( $w_{ij}$ ) para un patrón de aprendizaje  $p$  determinado es:

$$\Delta w_{ji}(t+1) = \alpha \delta_{pj} y_{pi}$$

En donde el subíndice  $p$  se refiere al patrón de aprendizaje concreto y  $\alpha$  es la constante o tasa de aprendizaje.

El punto en el que difieren la regla delta generalizada de la regla delta es en el valor concreto de  $\delta_{pj}$ . Por otro lado, en las redes multnivel, a diferencia de las redes sin neuronas ocultas, en principio no se puede conocer la salida deseada de las neuronas de las capas ocultas para poder determinar los pesos en función del error cometido. Sin embargo, inicialmente sí podemos conocer la

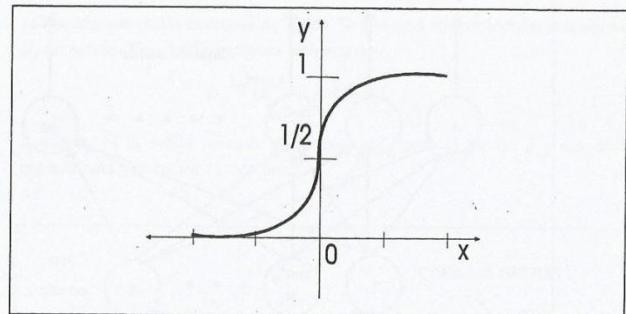


Figura 4.19 Función de activación sigmoidal:  $y = \frac{1}{1+e^{-x}}$

A continuación se presentan, a modo de síntesis, los pasos y fórmulas a utilizar para aplicar el algoritmo de entrenamiento, cuya demostración se realizará en el apartado 4.4.4:

*Paso 0. establecer unos aceptables para cada etapa*

#### Paso 1

*Iniciarizar los pesos* de la red con valores pequeños aleatorios.

#### Paso 2

*Presentar un patrón de entrada,  $X_p: x_{p1}, x_{p2}, \dots, x_{pN}$ , y especificar la salida deseada* que debe generar la red:  $d_1, d_2, \dots, d_M$  (si la red se utiliza como un clasificador, todas las salidas deseadas serán cero, salvo una, que será la de la clase a la que pertenece el patrón de entrada).

#### Paso 3

*Calcular la salida actual de la red*, para ello presentamos las entradas a la red y vamos calculando la salida que presenta cada capa hasta llegar a la capa de salida ésta será al salida de la red  $y_1, y_2, \dots, y_M$ . Los pasos son los siguientes:

- Se calculan las entradas netas para las neuronas ocultas procedentes de las neuronas de entrada.

Para una neurona  $j$  oculta :

$$\text{net}_{pj}^h = \sum_{i=1}^N w_{ji}^h x_{pi} + \theta_j^h$$

en donde el índice  $h$  se refiere a magnitudes de la capa oculta (*hidden*); el subíndice  $p$ , al  $p$ -ésimo vector de entrenamiento, y  $j$  a la  $j$ -ésima neurona oculta. El término  $\theta$  puede ser opcional, pues actúa como una entrada más.

- Se calculan las salidas de las neuronas ocultas:

$$y_{pj} = f_j^h(\text{net}_{pj}^h)$$

- Se realizan los mismos cálculos para obtener las salidas de las neuronas de salida (capa  $o$ : *output*)

$$\text{net}_{pk}^o = \sum_{j=1}^L w_{kj}^o y_{pj} + \theta_k^o$$

$$y_{pk} = f_k^o(\text{net}_{pk}^o)$$

#### Paso 4

*Calcular los términos de error para todas las neuronas.*

Si la neurona  $k$  es una *neurona de la capa de salida*, el valor de la *delta* es:

$$\delta_{pk}^o = (d_{pk} - y_{pk}) f_k^{o'}(\text{net}_{pk}^o)$$

La función  $f$ , como se citó anteriormente, debe cumplir el requisito de ser derivable, lo que implica la imposibilidad de utilizar una función escalón. En

general, disponemos de dos formas de función de salida que nos pueden servir: la función lineal de salida ( $f_k(\text{net}_{jk}) = \text{net}_{jk}$ ) y la función sigmoidal representada en la figura 4.19 y definida por la expresión:

$$f_k(\text{net}_{jk}) = \frac{1}{1+e^{-\text{net}_{jk}}}$$

La selección de la función de salida depende de la forma en que se decida representar los datos de salida: si se desea que las neuronas de salida sean binarias, se utiliza la función sigmoidal, puesto que esta función es casi biestable y, además, derivable. En otros casos es tan aplicable una función como otra.

Para la función lineal, tenemos:  $f_k^{o'} = 1$ , mientras que la derivada de una función  $f$  sigmoidal es:

$$f_k^{o'} = f_k^o(1 - f_k^o) = y_{pk}(1 - y_{pk})$$

por lo que los términos de error para las neuronas de salida quedan:

$$\delta_{pk}^o = (d_{pk} - y_{pk})$$

para la salida lineal, y

$$\delta_{pk}^o = (d_{pk} - y_{pk}) y_{pk}(1 - y_{pk})$$

para la salida sigmoidal.

 Si la neurona  $j$  no es de salida, entonces la derivada parcial del error no puede ser evaluada directamente. Por tanto, se obtiene el desarrollo a partir de valores que son conocidos y otros que pueden ser evaluados.

La expresión obtenida en este caso es:

$$\delta_{pj}^h = f_j^{h'}(\text{net}_{pj}^h) \sum_k \delta_{pk}^o w_{kj}^o$$

donde observamos que el error en las capas ocultas depende de todos los términos de error de la capa de salida. De aquí surge el término de *propagación hacia atrás*. En particular, para la función sigmoidal:

$$\delta_{pj}^h = x_{pi} (1 - x_{pi}) \sum_k \delta_{pk}^o w_{kj}^o$$

donde  $k$  se refiere a todas las neuronas de la capa superior a la de la neurona  $j$ . Así, el error que se produce en una neurona oculta es proporcional a la suma de los errores *conocidos* que se producen en las neuronas a las que está conectada la salida de ésta, multiplicado cada uno de ellos por el peso de la conexión. Los umbrales internos de las neuronas se adaptan de forma similar, considerando que están conectados con pesos desde entradas auxiliares de valor constante.

#### Paso 5

##### Actualización de los pesos

Para ello, utilizamos el algoritmo recursivo, comenzando por las neuronas de salida y trabajando hacia atrás hasta llegar a la capa de entrada, ajustando los pesos de la forma siguiente:

Para los pesos de las neuronas de la capa de salida:

$$w_{kj}^o(t+1) = w_{kj}^o(t) + \Delta w_{kj}^o(t+1) ; \\ \Delta w_{kj}^o(t+1) = \alpha \delta_{pk}^o y_{pj}$$

*error de la capa oculta*

y para los pesos de las neuronas de la capa oculta:

$$\begin{aligned} w_{ji}^h(t+1) &= w_{ji}^h(t) + \Delta w_{ji}^h(t+1) ; \\ \Delta w_{ji}^h(t+1) &= \alpha \delta_{pj}^h x_{pi} \end{aligned}$$

En ambos casos, para acelerar el proceso de aprendizaje, se puede añadir un término *momento* (Apartado 4.4.1.2.) de valor:  $\beta(w_{ij}^0(t) - w_{ij}^0(t-1))$  en el caso de la neurona de salida, y  $\beta(w_{ji}^h(t) - w_{ji}^h(t-1))$  cuando se trata de una neurona oculta.

**Paso 6** El proceso se repite hasta que el término de error

$$E_p = \frac{1}{2} \sum_{k=1}^M \delta_{pk}^2$$

resulta aceptablemente pequeño para cada uno de los patrones aprendidos.

#### 4.4.3. Consideraciones sobre el algoritmo de aprendizaje

El algoritmo de *backpropagation* encuentra un valor mínimo de error (local o global) mediante la aplicación de pasos descendentes (gradiente descendente). Cada punto de la superficie de la función de error corresponde a un conjunto de valores de los pesos de la red. Con el gradiente descendente, siempre que se realiza un cambio en todos los pesos de la red, se asegura el descenso por la superficie del error hasta encontrar el valle más cercano, lo que puede hacer que el proceso de aprendizaje se detenga en un mínimo local de error.

Por tanto, uno de los problemas que presenta este algoritmo de entrenamiento de redes multicapa es que busca minimizar la función de error, pudiendo caer en un mínimo local o en algún punto estacionario, con lo cual no se llega a encontrar el mínimo global de la función del error. Sin embargo, ha de tenerse en cuenta que no tiene porqué alcanzarse el mínimo global en todas las aplicaciones, sino que puede ser suficiente con un error mínimo preestablecido.

##### 4.4.3.1. Control de la convergencia

En las técnicas de gradiente decreciente es conveniente avanzar por la superficie de error con incrementos pequeños de los pesos. Esto se debe a que tenemos una información local de la superficie y no se sabe lo lejos o lo cerca que se está del punto mínimo. Con incrementos grandes, se corre el riesgo de pasar por encima del punto mínimo sin conseguir estacionarse en él. Con incrementos pequeños, aunque se tarde más en llegar, se evita que ocurra esto.

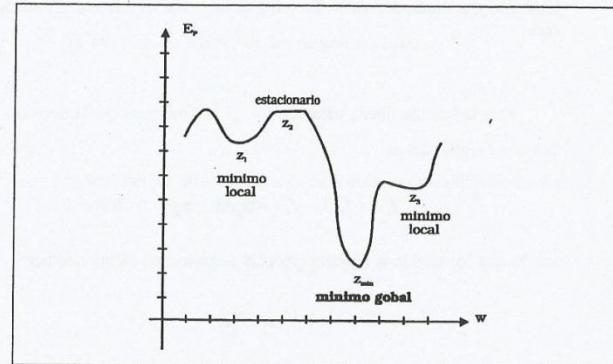


Figura 4.20 Ejemplo representativo de una forma de la superficie de error, donde  $w$  representa los posibles valores de la matriz de pesos de la red

El elegir un incremento paso adecuado influye en la velocidad con la que converge el algoritmo. Esta velocidad se controla a través de la constante de proporcionalidad o tasa de aprendizaje  $\alpha$ . Normalmente,  $\alpha$  debe ser un número pequeño (del orden de 0,05 a 0,25), para asegurar que la red llegue a asentarse en una solución. Un valor pequeño de  $\alpha$  significa que la red tendrá que hacer un gran número de iteraciones. Si esa constante es muy grande, los cambios de pesos son muy grandes, avanzando muy rápidamente por la superficie de error, con el riesgo de saltar el mínimo y estar oscilando alrededor de él, pero sin poder alcanzarlo.