CAPÍTULO 2

REDES NEURONALES SUPERVISADAS

En los próximos capítulos trataremos algunos de los modelos de redes neuronales más populares. En primer lugar, comenzaremos estudiando la amplia clase de redes unidireccionales organizadas en capas (*feed-forward*) y con aprendizaje supervisado, que son empleadas como clasificadores de patrones y estimadores de funciones. Estos modelos en la literatura son denominados *mapping neural networks* [Hecht-Nielsen 90], o redes neuronales para representación (ajuste) funcional.

Dentro de este gran grupo de redes trataremos el **perceptrón simple**, la **adalina** y el **perceptrón multicapa**. El popular algoritmo de aprendizaje denominado *back-propagation* (retropropagación) o **BP** se aplica precisamente a este último modelo. El perceptrón multicapa con aprendizaje BP (o alguna de sus variantes) es el modelo neuronal más empleado en las aplicaciones prácticas (se estima que el 70% de los desarrollos con redes neuronales hacen uso alguna de sus variantes [Gedeon 95]).

Por último, señalaremos algunas referencias para el lector interesado en ampliar conocimientos. En primer lugar destacaremos el artículo [Hush 93], que constituye una excelente actualización del clásico de Lippmann [Lippmann 87]. Para un estudio en mayor profundidad de estos modelos recomendamos el texto [Bishop 95] y, especialmente, los recientes [Haykin 99] y [Principe 00], donde se recopilan las últimas investigaciones.

2.1 REDES UNIDIRECCIONALES

Muchos problemas del mundo real pueden interpretarse desde el punto de vista de la estimación o aproximación funcional, en el sentido de tratar de encontrar la función que a partir de un conjunto de entradas proporciona la salida deseada. Por ejemplo, si queremos desarrollar un reconocedor de caracteres manuscritos el objetivo será encontrar un sistema que implemente la función que asocia la imagen de una determinada letra o carácter escrito con la clase a la que pertenece. Otro ejemplo

ilustrativo sería el de la predicción de cotizaciones bursátiles, en el que mediante una red neuronal se trataría de encontrar la función que relaciona diversas variables de entrada (cotizaciones previas, tipos de interés, inflación, etc.) con la actual cotización en bolsa de una determinada entidad o empresa.

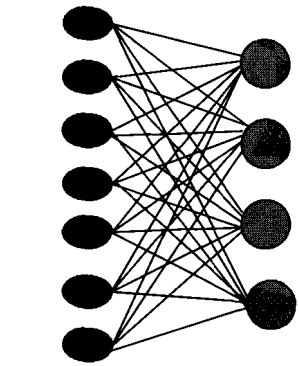
Como hemos adelantado, dentro del grupo de modelos de redes neuronales unidireccionales trataremos especialmente los casos del perceptrón simple, adalina y perceptrón multicapa o MLP (*Multilayer Perceptron*). En primer lugar, hay que destacar que estos modelos presentan un gran interés histórico, pues su evolución representa la historia misma de las redes neuronales. Así, el perceptrón simple y la adalina se propusieron a finales de los años cincuenta, alcanzando una gran popularidad durante los años sesenta, para a continuación sufrir un duro revés a finales de esa década, debido fundamentalmente al riguroso trabajo de Minsky y Papert [Minsky 69], en el que pusieron claramente de manifiesto sus limitaciones. El contraste con el gran interés que el tema había despertado hasta entonces hizo que el campo de las redes neuronales en general entrase en una época oscura durante la década de los setenta, desviándose la mayor parte de los recursos económicos al prometedor campo de la inteligencia artificial. Sin embargo, el tema resurgió en los ochenta debido a diversas circunstancias, como la disponibilidad de ordenadores con potencia suficiente para llevar a cabo simulaciones antes difícilmente abordables, el desarrollo de la integración VLSI, que permitió realizar electrónicamente redes neuronales, y la introducción de nuevos modelos, especialmente el MLP entrenado mediante el algoritmo BP, que superaba los viejos problemas de los modelos predecesores, anulando buena parte de las objeciones "históricas" de Minsky y Papert.

No obstante, el principal interés de los modelos que trataremos en este capítulo es su generalidad y aplicabilidad práctica, además de ilustrar muy bien una amplia clase de problemas y aspectos que aparecen con frecuencia en todo el campo de los ANS. Por todo ello, este capítulo puede considerarse como uno de los centrales de la parte correspondiente a redes neuronales. Por otro lado, resulta interesante saber que algunos de los modelos de aprendizaje que expondremos también son empleados en el entrenamiento de sistemas borrosos, como veremos en la segunda parte.

2.2 EL ASOCIADOR LINEAL: APRENDIZAJE HEBBIANO

Antes de comenzar con los modelos citados estudiaremos el **asociador lineal**, un sencillo ejemplo de red unidireccional que servirá para introducir los conceptos relacionados con el aprendizaje en redes neuronales. Este modelo, mediante una transformación lineal, asocia un conjunto de patrones de entrada a otros de salida.

El asociador lineal consta únicamente de una capa de neuronas lineales, cuyas entradas las denotamos por $\mathbf{x}$ y sus salidas por $\mathbf{y}$, vector que constituye además la respuesta de la red neuronal. Asimismo, denotaremos por $W=\{w_{ij}\}$ a la matriz de pesos sinápticos; cada fila de W contiene los pesos de una neurona $\mathbf{w}_i$



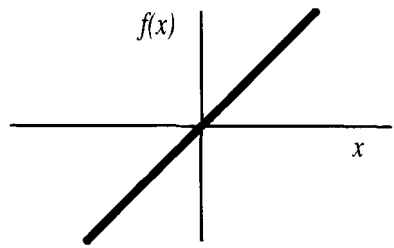


Figura 2.1 *Asociador lineal (función de activación identidad)*

$$\mathbf{W} = \begin{pmatrix} \mathbf{w}_1 & \mathbf{w}_2 & \dots & \mathbf{w}_m \end{pmatrix}^T \tag{2.1}$$

La operación del asociador lineal es simplemente

$$\mathbf{y} = \mathbf{W}\mathbf{x} \tag{2.2}$$

o bien

$$y_i = \sum_{j=1}^{n} w_{ij} x_j \tag{2.3}$$

Por lo tanto, cada neurona *i* del asociador lineal lleva a cabo la suma ponderada de las entradas con sus pesos sinápticos. Es decir, dentro del marco de neurona estándar descrito en el capítulo 1, esta neurona calcula el potencial postsináptico por medio de la convencional suma ponderada, cantidad a la que aplica finalmente una función activación de tipo identidad.

El asociador lineal debe aprender a asociar p pares entrada-salida[1], $\{(\mathbf{x}^\mu, \mathbf{t}^\mu) / 1 \leq \mu \leq p\}$, ajustando sus pesos W de modo que ante un cierto patrón de entrada $\mathbf{x}^\mu$ responda con $\mathbf{t}^\mu$, y que ante entradas similares, $(\mathbf{x}^\mu+\varepsilon)$, responda con salidas también próximas $(\mathbf{t}^\mu+\delta)$ (con ε y δ cantidades pequeñas). El problema se centra en encontrar la matriz de pesos W óptima en el sentido descrito. Para ello, en el campo de las redes neuronales normalmente se hace uso de una **regla de aprendizaje**, que a partir de las entradas y de las salidas deseadas (en el caso del aprendizaje supervisado), proporcione el conjunto óptimo de pesos W.

[1] Usaremos el símbolo *t* (*target*) cuando nos refiramos a las salidas deseadas u objetivo.

Regla de aprendizaje de Hebb

Se trata de uno de los modelos clásicos de aprendizaje en redes neuronales. Donald Hebb en 1949 [Hebb 49] postuló un mecanismo de aprendizaje para la neurona biológica, cuya idea básica consiste en que *cuando un axón presináptico causa la activación de cierta neurona postsináptica, la eficacia de la sinapsis que las relaciona se refuerza.* El trabajo experimental posterior ha confirmado en parte esta teoría [Kandel 92], demostrando la presencia de este tipo de aprendizaje en la neurona biológica, aunque en coexistencia con otros esquemas [Alkon 89] (interacción a nivel presináptico, crecimiento y debilitamiento del axón, modificaciones del metabolismo, desarrollo y muerte celular, etc.).

Este tipo de aprendizaje es simple y local. Gran parte de su importancia radica en que fue pionero, tanto en neurociencias como en neurocomputación, y muchos otros algoritmos más complejos (y más potentes) lo toman como punto de partida.

De una manera general, se denomina **aprendizaje hebbiano** a aquellas formas de aprendizaje que involucran una modificación en los pesos Δw_{ij} proporcional al producto de una entrada j por la salida i de la neurona

$$\Delta w_{ij} = \varepsilon y_i x_j \tag{2.4}$$

siendo ε un parámetro denominado **ritmo de aprendizaje**, que suele ser una cantidad entre 0 y 1. Esta expresión puede considerarse la representación matemática del modelo de aprendizaje descrito por Hebb.

Consideremos nuestro asociador lineal. La regla de Hebb se expresa en este caso particular así

$$\Delta w_{ij}^{\mu} = t_i^{\mu} x_j^{\mu} \tag{2.5}$$

y, por lo tanto

$$w_{ij}^{new} = w_{ij}^{old} + \Delta w_{ij}^{\mu} \tag{2.6}$$

Si los pesos de partida son nulos, el valor final de W para las p asociaciones será

$$\mathbf{W} = \mathbf{t}^1\mathbf{x}^{1T} + \mathbf{t}^2\mathbf{x}^{2T} + \ldots + \mathbf{t}^p\mathbf{x}^{pT} \tag{2.7}$$

Empleando la regla de Hebb para el entrenamiento del asociador lineal, si los vectores de entrada $\{\mathbf{x}^1, \mathbf{x}^2, \ldots, \mathbf{x}^p\}$ son ortonormales (ortogonales y de longitud unidad) se cumple

$$\mathbf{W}\mathbf{x}^{\mu} = \left(\mathbf{t}^1\mathbf{x}^{1T} + \ldots + \mathbf{t}^p\mathbf{x}^{pT}\right)\mathbf{x}^{\mu} = \mathbf{t}^1(\mathbf{x}^{1T}.\mathbf{x}^{\mu}) + \ldots + \mathbf{t}^p(\mathbf{x}^{pT}.\mathbf{x}^{\mu}) = \mathbf{t}^{\mu} \tag{2.8}$$

y por tanto, ante la entrada $\mathbf{x}^{\mu}$ se reproduce la respuesta aprendida $\mathbf{t}^{\mu}$, es decir, la regla de Hebb ha conseguido en este caso que la red aprenda a realizar las asociaciones deseadas. El problema reside en que las condiciones son muy restrictivas, pues para

que las asociaciones sean correctas los patrones de entrada deben ser ortonormales. Por ello, si la dimensión del espacio de entrada es *n*, solamente podrá aprender hasta *n* asociaciones. Para almacenar más pares entrada-salida será preciso utilizar otras estrategias.

Eliminando la condición de ortogonalidad, se tiene (manteniendo el requisito de vectores de longitud 1)

$$\mathbf{W}\mathbf{x}^{\mu} = \mathbf{t}^{1}(\mathbf{x}^{1\,T}\mathbf{x}^{\mu}) + \mathbf{t}^{2}(\mathbf{x}^{2\,T}\mathbf{x}^{\mu}) + \ldots + \mathbf{t}^{p}(\mathbf{x}^{p\,T}\mathbf{x}^{\mu}) = \mathbf{t}^{\mu} + \sum_{\nu \neq \mu} \mathbf{t}^{\nu}(\mathbf{x}^{\nu\,T}\mathbf{x}^{\mu}) = \mathbf{t}^{\mu} + \boldsymbol{\delta} \quad (2.9)$$

expresión denominada **expansión señal-ruido**, pues proporciona la salida deseada más un término adicional, que interpretamos como el ruido superpuesto a la señal. Empleando reglas algo más sofisticadas que la de Hebb, como la de la pseudoinversa o la de Widrow-Hoff, se obtendrá una matriz de pesos que logrará además que el ruido $\boldsymbol{\delta}$ sea pequeño comparado con la señal.

Regla de la pseudoinversa

La regla de aprendizaje de Hebb ha sido introducida debido a su plausibilidad biológica. Sin embargo, en general se tratará de deducir los algoritmos de aprendizaje a partir de un cierto criterio a optimizar; *el aprendizaje usualmente se planteará como un procedimiento para alcanzar el conjunto de pesos óptimo que resuelva un problema dado*. Para ello se hace necesario definir el significado de "óptimo" en cada caso concreto, es decir, hay que proponer un criterio que mida el rendimiento de la red neuronal para encontrar una regla de actualización de pesos que lo optimice. Una forma habitual de definir el rendimiento es el error cuadrático medio de las salidas actuales de la red respecto de las deseadas. Para el asociador lineal se tendría

$$E\{w_{ij}\} = (1/p)\sum_{\mu=1}^{p} \| \mathbf{t}^{\mu} - \mathbf{W}\mathbf{x}^{\mu} \|^2 = (1/p)\sum_{\mu=1}^{p}\sum_{i=1}^{n} \left(t_i^{\mu} - \mathbf{W}x_i^{\mu}\right)^2 \quad (2.10)$$

De este modo, un algoritmo de aprendizaje para el asociador lineal debería obtener un conjunto de pesos que minimicen esta expresión del error. Si los vectores $\mathbf{x}^{\mu}$ son ortonormales, el error que proporciona la regla de Hebb (2.5) según (2.10) es cero, lo que indica que esta regla es óptima respecto de la medida de error propuesta si se dispone de vectores de entrada $\mathbf{x}^{\mu}$ ortonormales.

Si denominamos X a una matriz *nxp* que tiene por columnas los vectores de entrada $\mathbf{x}^{\mu}$, X=($\mathbf{x}^1$ $\mathbf{x}^2$ ... $\mathbf{x}^p$), y si llamamos Y a la matriz *mxp* cuyas columnas son los vectores de salida $\mathbf{y}^{\mu}$, Y=($\mathbf{y}^1$ $\mathbf{y}^2$ ... $\mathbf{y}^p$), la ecuación (2.10) se transforma en[2]

---

[2] Definiendo la norma $\|M\|^2$ de una matriz M (*pxq*) de la siguiente forma $\| \mathbf{M} \|^2 = \| (m_{ij}) \|^2 = \sum_{i=1}^{p}\sum_{j=1}^{q} m_{ij}^2$

$$E\{w_{ij}\} = (1/p) \| Y - WX \|^2 \tag{2.11}$$

Con esta nomenclatura, la regla de Hebb se expresa de la forma siguiente

$$W = YX^T \tag{2.12}$$

Una regla de aprendizaje basada en la utilización de la **matriz pseudoinversa** puede escribirse como

$$W = YX^+ \tag{2.13}$$

donde X^+ denota la pseudoinversa[3] de X [Kohonen 89, Hecht-Nielsen 90]. Se puede demostrar que esta elección para W minimiza el error cuadrático medio (ecuaciones 2.10 y 2.11), es decir, que es óptima respecto de este error [Hetch-Nielsen 90]. En [Ritter 91a] se deduce (2.13) a partir de la minimización algebraica de (2.11).

Así como con la regla de Hebb se podían almacenar hasta *n* vectores ortonormales, con la pseudoinversa se pueden almacenar hasta *n* vectores linealmente independientes (por ejemplo, en la página 82 de [Hecht-Nielsen 90], puede verse su demostración). Si pretendemos almacenar más pares entrada-salida, surgirán errores, pero el *mapping* lineal implementado seguirá siendo óptimo en el sentido del error cuadrático medio (se alcanzará el menor error posible).

Debido a que ambas reglas son óptimas según el mismo criterio, la regla de Hebb y la de la pseudoinversa deben estar muy relacionadas. Esta circunstancia es fácil de apreciar, pues si consideramos un conjunto de vectores de entrada ortonormales, la regla de la pseudoinversa (2.13) se convierte en la de Hebb. Por otra parte, si se realiza la expansión en serie de la ecuación (2.13) de la pseudoinversa (véase, por ejemplo, la página 455 de [Rumelhart 86]), el primer término de la serie es precisamente la ecuación (2.12) de la regla de Hebb. Es decir, la regla de Hebb representa en el fondo un caso particular de la más general regla de la pseudoinversa.

Habitualmente para el cálculo de la pseudoinversa se utiliza el **teorema de Greville** [Kohonen 89, Hecht-Nielsen 90], aunque presenta el inconveniente de que para aprender un nuevo patrón se debe recalcular toda la matriz de pesos, lo que no resulta común dentro de la filosofía de los ANS [Ritter 91a], en la se tiende a que todos los modelos sean locales y operen incrementalmente. Se ha mostrado que en la

---

[3] Toda matriz tiene una pseudoinversa. La pseudoinversa A^+ de una matriz A (*pxq*) arbitraria se define como la única matriz que cumple las siguientes propiedades:

$$AA^+A=A$$
$$A^+AA^+=A^+$$
$$AA^+=(AA^+)^T$$
$$A^+A=(A^+A)^T$$

Se puede demostrar que la pseudoinversa de una matriz cuadrada no singular es igual a su inversa. En este sentido, la pseudoinversa representa la generalización del concepto de inversa de una matriz, para el caso de matrices no cuadradas [Kohonen 89].

práctica [Hecht-Nielsen 90] la siguiente forma aproximada del teorema de Greville, local e iterativa, suele proporcionar resultados correctos

$$\mathbf{w}_i^{new} = \mathbf{w}_i^{old} + \varepsilon.\left(t_i^{\mu} - (\mathbf{w}_i^{old})^T \mathbf{x}^{\mu}\right)\mathbf{x}^{\mu} \tag{2.14}$$

siendo ε el ritmo de aprendizaje, parámetro que indica la rapidez en la actualización ($0<\varepsilon<1$). En este esquema iterativo, los patrones deben ser presentados a la red neuronal repetidamente, obteniéndose de esta manera una aproximación a la matriz pseudoinversa mediante cálculos simples y locales.

Esta expresión coincide con la famosa regla de Widrow-Hoff de la adalina que veremos más adelante. Asimismo, en [Rumelhart 86a] se relacionan ambos algoritmos de aprendizaje, mostrándose que en realidad ambas reglas son equivalentes cuando se trata de realizar un aprendizaje estadístico, consistente en asignar en vez de un patrón de entradas a uno de salidas, toda una clase de vectores de entrada (compuesta por un conjunto de patrones estocásticos que se distribuyen gaussianamente en torno al prototipo de la clase) a una clase del espacio de salidas (de similares características). Es decir, la regla de la pseudoinversa y la de Widrow-Hoff coinciden asintóticamente cuando se consideran distribuciones de vectores estocásticos.

La regla de la pseudoinversa se ha aplicado en redes más complejas, como la de Hopfield [Personnaz 86], proporcionando mejores resultados que la de Hebb, inicialmente propuesta y más conocida y extensamente estudiada (véase, por ejemplo, [Müller 90] y referencias allí citadas).

2.3 EL PERCEPTRÓN SIMPLE (ROSENBLATT, 1959)

Este modelo neuronal fue introducido por Rosenblatt a finales de los años cincuenta [Rosenblatz 62, Hertz 91, Principe 00]. La estructura del perceptrón se inspira en las primeras etapas de procesamiento de los sistemas sensoriales de los animales (por ejemplo, el de visión), en los cuales la información va atravesando sucesivas capas de neuronas, que realizan un procesamiento progresivamente de más alto nivel.

El perceptrón simple es un modelo unidireccional, compuesto por dos capas de neuronas, una sensorial o de entradas, y otra de salida (Figura 2.2). La operación de una red de este tipo, con n neuronas de entrada y m de salida, se puede expresar como

$$y_i(t) = f\left(\sum_{j=1}^{n} w_{ij} x_j - \theta_i\right), \ \forall i, \ 1 \le i \le m \tag{2.15}$$

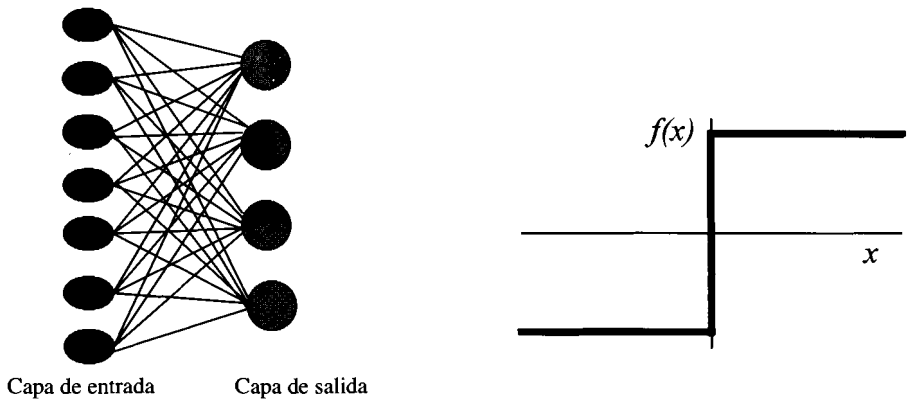


***Figura 2.2** Perceptrón simple y función de transferencia de su neurona*

Las neuronas de entrada no realizan ningún cómputo, únicamente envían la información (en principio consideraremos señales discretas $\{0, +1\}$) a las neuronas de salida (en el modelo original estas neuronas de entrada representaban información ya procesada, no datos directamente procedentes del exterior). La función de activación de las neuronas de la capa de salida es de tipo escalón. Así, la operación de un perceptrón simple puede escribirse

$$y_i = H(\sum_{j=1}^{n} w_{ij} x_j - \theta_i), \ \forall i, \ 1 \leq i \leq m \tag{2.16}$$

con $H(.)$ la función de Heaviside o escalón (capítulo 1). El perceptrón puede utilizarse tanto como clasificador, como para la representación de funciones booleanas, pues su neurona es esencialmente de tipo MacCulloch-Pitts, de salida binaria. La importancia histórica del perceptrón radica en su carácter de dispositivo entrenable, pues el algoritmo de aprendizaje del modelo introducido por Rosenblatt, y que describiremos más adelante, permite determinar automáticamente los pesos sinápticos que clasifican un conjunto de patrones a partir de un conjunto de ejemplos etiquetados.

Mostraremos a continuación que un perceptrón permite realizar tareas de clasificación. Cada neurona del perceptrón representa una determinada clase, de modo que dado un vector de entrada, una cierta neurona responde con 0 si no pertenece a la clase que representa, y con un 1 si sí pertenece. Es fácil ver que una neurona tipo perceptrón solamente permite discriminar entre dos clases **linealmente separables** (es decir, cuyas regiones de decisión pueden ser separadas mediante una única condición lineal o hiperplano[4]). Sea una neurona tipo perceptrón de dos entradas, x_1 y x_2, con salida y, cuya operación se define por lo tanto

[4]Una línea recta, si trabajamos en dos dimensiones.

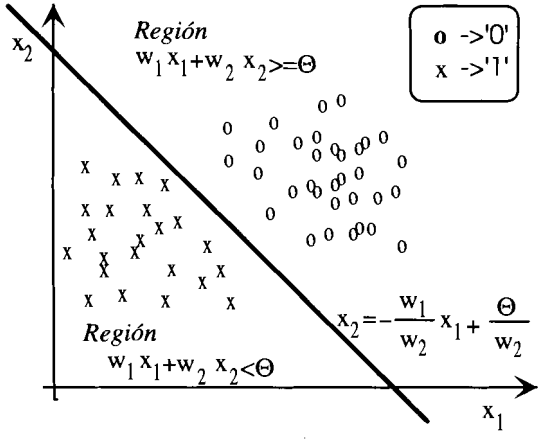


***Figura 2.3** Regiones de decisión en el plano*

$$y = H(w_1 x_1 + w_2 x_2 - \theta) \tag{2.17}$$

o bien

$$y = \begin{cases} 1, \text{ si } w_1 x_1 + w_2 x_2 \geq \theta \\ 0, \text{ si } w_1 x_1 + w_2 x_2 < \theta \end{cases} \tag{2.18}$$

Si consideramos x_1 y x_2 situadas sobre los ejes de abscisas y ordenadas en el plano, la condición

$$w_1 x_1 + w_2 x_2 - \theta = 0 \Rightarrow x_2 = -\frac{w_1}{w_2} x_1 + \frac{\theta}{w_2} \tag{2.19}$$

representa una recta (hiperplano, si trabajamos con n entradas) que divide el plano (espacio) en dos regiones, aquellas para las que la neurona proporciona una salida '0' o '1', respectivamente (Figura 2.3). Luego, efectivamente, una neurona tipo perceptrón representa un discriminador lineal, al implementar una condición lineal que separa dos regiones en el espacio, que representan dos diferentes clases de patrones.

Consideremos la función lógica $NAND_2$ (AND negada de dos entradas), que representamos sobre el plano (Figura 2.4a). En este caso pueden encontrarse unos parámetros w_1, w_2 y θ que determinen una recta que separa perfectamente las regiones correspondientes a los valores lógicos 0 y 1. Por ello, la función lógica NAND se dice separable linealmente, puesto que hemos podido encontrar una única condición lineal que divida ambas regiones[5]. Por ejemplo, un perceptrón con los siguientes parámetros implementa la función NAND: w_1=w_2=-2, y θ=-3 (capítulo 1).

[5]Una definición más rigurosa sería: una función se dice **linealmente separable** cuando su espacio de variables de entrada puede ser dividido en regiones de igual salida mediante una única condición lineal (un hiperplano).

Sin embargo, consideremos la función lógica or-exclusivo o XOR (su salida es el 0 lógico si las variables de entrada son iguales y 1 si son diferentes), y representémosla también en el plano (Figura 2.4b). En este caso podemos apreciar que no se puede encontrar una única condición lineal que separe las regiones correspondientes a los valores de salida 0 y 1, por lo que se dice que la XOR no es separable linealmente. Como la neurona del perceptrón representa en el fondo un discriminador lineal, esta neurona por sí sola no puede implementar la función XOR. Por lo tanto, concluimos con que *la clase de funciones no separables linealmente no puede ser representada por un perceptrón simple.*

Por lo tanto, pese a su gran interés, el perceptrón presenta serias limitaciones, pues solamente puede representar funciones linealmente separables. Así, aunque pueda aprender automáticamente a representar complejas funciones booleanas o resolver con éxito muchos problemas de clasificación (mediante el algoritmo que expondremos más adelante), en otras ocasiones fallará estrepitosamente.

Minsky (uno de los padres de la IA) y Papert [Minsky 69] estudiaron en profundidad el perceptrón, y en 1969 publicaron un exhaustivo trabajo en el que se subrayaba sus limitaciones, lo que resultó decisivo para que muchos de los recursos que se estaban invirtiendo en redes neuronales se desviasen hacia otros campos más prometedores entonces, como era en la época el de la inteligencia artificial.

Reflexionemos un poco sobre el problema de la función XOR para intentar encontrar una solución. Una neurona tipo perceptrón implementa una decisión lineal, observando la Figura 2.4 podemos considerar dos neuronas perceptrón, una implementa la decisión lineal DL1, y la otra la DL2. Consideremos una capa adicional, compuesta por una única neurona perceptrón encargada de componer las regiones en las que el plano queda dividido por las dos neuronas anteriores: si esta neurona se activa únicamente cuando la neurona correspondiente a DL1 está activada y DL2 desactivada, tendremos una red de tres capas (una de ellas oculta, es decir, sin conexión directa al exterior) que implementa la función XOR. Luego una solución a las limitaciones del perceptrón simple puede consistir en incluir más capas en la arquitectura, con lo que tendremos un **perceptrón multicapa**.

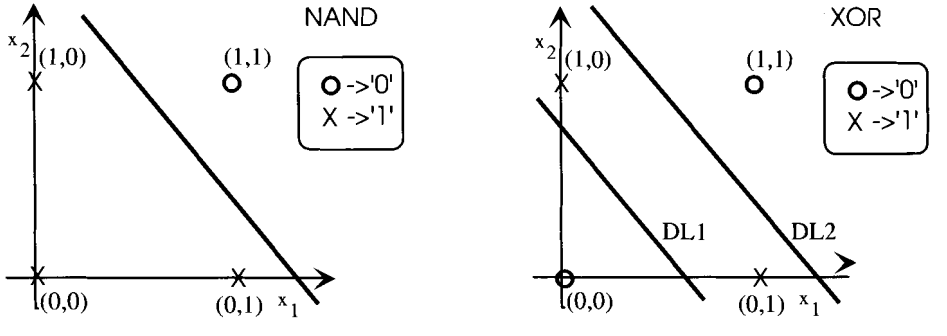


Figura 2.4 *Funciones lógicas NAND (a) y XOR (b)*

Los tipos de regiones de decisión que pueden formarse mediante estructuras simples y multicapa se muestran en la Figura 2.5. El problema a resolver en este caso es la discriminación entre dos clases de patrones, la clase A y la B. Los contornos que se forman con estructuras multicapa son polinomiales, puesto que consisten en la composición de discriminaciones lineales correspondientes a diferentes neuronas de tipo umbral. Si consideramos neuronas de respuesta continua, por ejemplo de tipo sigmoideo, los contornos serían similares, aunque sin esquinas. De la figura podemos apreciar que mediante una estructura de dos capas, sin capa oculta (la que corresponde a un perceptrón simple), la región de decisión es un hiperplano que separa en dos el espacio de las variables. Haciendo uso de tres capas, con una oculta, se pueden discriminar regiones convexas, sean cerradas o abiertas. Con una estructura de cuatro capas, dos de ellas ocultas, se puede discriminar regiones de forma arbitraria, cuyo único límite viene impuesto por el número de nodos empleados.

Por ejemplo, en [Müller 90] pueden encontrarse demostraciones más formales relacionadas con los conceptos expuestos. Por ejemplo, allí se demuestra que *toda función booleana puede ser representada por una red neuronal unidireccional con una única capa oculta*, lo que fue demostrado por primera vez por Denker y otros a mediados de los ochenta [Denker 87].

A finales de los sesenta ya se apuntaba como solución a las limitaciones del perceptrón introducir capas ocultas, pero el problema residía en que si bien se disponía de un algoritmo de aprendizaje para el perceptrón simple (el denominado algoritmo del perceptrón), no se disponía de ningún procedimiento que permitiese obtener automáticamente los pesos en uno multicapa, con neuronas ocultas. Este problema, denominado de "asignación de crédito" a las neuronas sin conexión directa con el exterior (consistente en cómo medir la contribución al error en la salida de la red neuronal de cada uno de los nodos ocultos que precisamente no tienen una conexión directa con ella) fue resuelto no mucho más tarde por Paul Werbos [Werbos 74], pero fue preciso esperar hasta mediados de los años ochenta para que el grupo PDP (junto con otros grupos de forma independiente) redescubriera un algoritmo similar, que denominaron *back-propagation* o BP [Rumelhart 86a], y diera a conocer a la comunidad internacional su gran potencial para la resolución de problemas prácticos.

2.3.1 Algoritmo de aprendizaje del perceptrón

La importancia del perceptrón radica en su carácter de dispositivo entrenable, pues el algoritmo de aprendizaje introducido por Rosenblatt permite que el perceptrón determine automáticamente los pesos sinápticos que clasifican un determinado conjunto de patrones etiquetados.

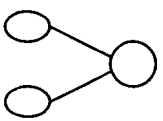
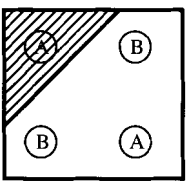
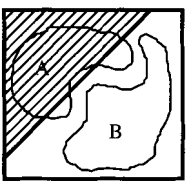
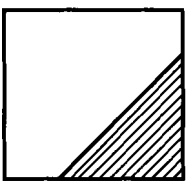
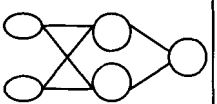
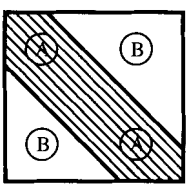
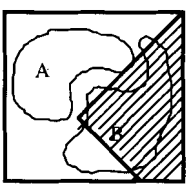
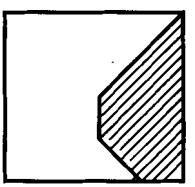
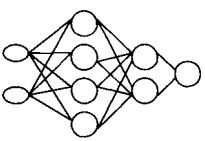
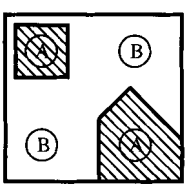
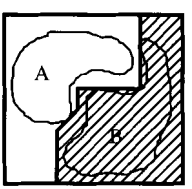
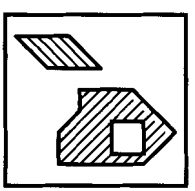
Arquitectura	Región de decisión	Ejemplo 1: XOR	Ejemplo 2: clasificación	Regiones más generales
Sin capa oculta 	Hiperplano (dos regiones)			
Una capa oculta 	Regiones polinomiales convexas			
Dos capas ocultas 	Regiones arbitrarias			

Figura 2.5 *Tipos de regiones de decisión en el perceptrón [Lippmann 87]*

El del perceptrón [Rosenblatz 62, Hertz 91] es un algoritmo de aprendizaje de los denominados por **corrección de errores**. Los algoritmos de este tipo (en el que incluiríamos también el de la adalina y el BP) ajustan los pesos en proporción a la diferencia existente entre la salida actual de la red y la salida deseada, con el objetivo de minimizar el error actual de la red.

Introduciremos sin más dilación la regla de aprendizaje. Sea un conjunto de p patrones $\mathbf{x}^\mu$, μ=1,..,p, con sus salidas deseadas $\mathbf{t}^\mu$. Tanto las entradas como las salidas solamente pueden tomar los valores -1 o 1 (o bien, 0 o 1, según definamos los niveles lógicos). Se tiene una arquitectura de perceptrón simple, con pesos iniciales aleatorios, y se requiere que clasifique correctamente todos los patrones del conjunto de aprendizaje (lo cual es posible solamente si son separables linealmente). Actuaremos del siguiente modo, ante la presentación del patrón μ-ésimo, si la respuesta que proporciona el perceptrón es correcta, no actualizaremos los pesos; si es incorrecta, los modificaremos según la regla de Hebb de la sección 2.2. Se tiene

$$\Delta w_{ij}^{\mu}(t) = \begin{cases} 2\varepsilon t_i^{\mu} x_j^{\mu}, & \text{si } y_i^{\mu} \neq t_i^{\mu} \\ 0 \quad , & \text{si } y_i^{\mu} = t_i^{\mu} \end{cases} \tag{2.20}$$

que se puede reescribir del siguiente modo

$$\Delta w_{ij}^{\mu}(t) = \varepsilon.(t_i^{\mu} - y_i^{\mu})x_j^{\mu} \tag{2.21}$$

que es la forma habitual de expresar la **regla del perceptrón**. En su utilización práctica, se debe llegar a un compromiso para el valor del ritmo de aprendizaje ε, puesto que un valor pequeño implica un aprendizaje lento, mientras que uno excesivamente grande puede conducir a oscilaciones en el entrenamiento, al introducir variaciones en los pesos excesivamente amplias. Al ser las entradas y las salidas discretas {-1,+1}, también lo será la actualización de los pesos (2.21), que únicamente podrá tomar los valores 0 o $\pm 2\varepsilon$.

Una forma mucho más gráfica de introducir la regla del perceptrón es la siguiente. Sea la neurona i tipo perceptrón {-1, +1}, cuyo vector de pesos es $\mathbf{w}_i$. Se presenta el patrón de entrada $\mathbf{x}^{\mu}$, la salida objetivo de la neurona i ante este patrón es t_i^{μ}. La operación de la neurona la escribimos como

$$y_i^{\mu}(t) = signo(\sum_{j=1}^{n} w_{ij}x_j^{\mu} - \theta_i) = signo(\mathbf{w}_i.\mathbf{x}^{\mu}) = signo(\| \mathbf{w}_i \| . \| \mathbf{x}^{\mu} \| \cos(\phi)) \tag{2.22}$$

considerando el umbral como un peso adicional de entrada -1 (véase el capítulo 1), y siendo ϕ el ángulo que forman los vectores de pesos y entradas. La hipersuperficie $\mathbf{w}_i.\mathbf{x}^{\mu}=0$ establece la condición lineal que separa el espacio en dos regiones, etiquetadas por -1 y +1, respectivamente. En el proceso de aprendizaje, ante la presentación del patrón μ-ésimo en la iteración t pueden darse los siguientes casos:

a) La salida objetivo de la neurona es t_i^{μ} =+1, pero su salida actual es y_i^{μ}=-1. En este caso, el producto escalar $\mathbf{w}_i.\mathbf{x}^{\mu}$ debería ser positivo, pero es negativo, lo cual indica que el ángulo existente entre $\mathbf{w}_i$ y $\mathbf{x}^{\mu}$ es mayor de 90° ($\phi \in [\pi/2, 3\pi/2]$, Figura 2.6). Así, la regla de aprendizaje del perceptrón debería en este caso acercar $\mathbf{w}_i$ a $\mathbf{x}^{\mu}$ para reducir el ángulo que forman, y eventualmente conseguir que sea inferior a 90° ($\mathbf{w}_i.\mathbf{x}^{\mu}>0$), lo cual se puede realizar del siguiente modo (véase la Figura 2.6a)

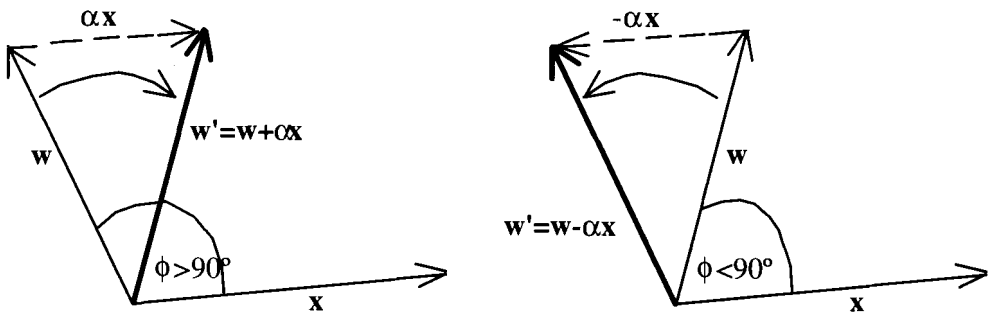
$$\mathbf{w}_i^{\mu}(t+1) = \mathbf{w}_i^{\mu}(t) + \alpha.\mathbf{x}^{\mu} \tag{2.23}$$

b) La salida objetivo de la neurona es t_i^{μ} =-1, pero su salida actual es y_i^{μ}=+1. Razonando al revés que en el caso anterior, la regla de aprendizaje deberá alejar $\mathbf{w}_i$ de $\mathbf{x}^{\mu}$, por lo tanto en este caso (Figura 2.6b)

$$\mathbf{w}_i^{\mu}(t+1) = \mathbf{w}_i^{\mu}(t) - \alpha.\mathbf{x}^{\mu} \tag{2.24}$$

c) La salida objetivo de la neurona t_i^{μ} coincide con su salida actual y_i^{μ}. En este caso la regla de aprendizaje no actúa.

Es fácil comprobar que los tres casos se resumen en la siguiente regla



Caso a), objetivo $\mathbf{w}_i.\mathbf{x}^\mu > 0$ Caso b), objetivo $\mathbf{w}_i.\mathbf{x}^\mu < 0$

Figura 2.6. *Regla del perceptrón, cuando salida actual y objetivo no coinciden*

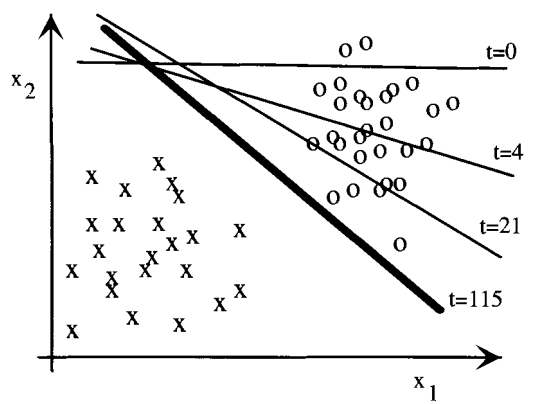


Figura 2.7 *Regiones de decisión que establece iterativamente el perceptrón durante el aprendizaje (en la iteración 115 ha conseguido separar ya las dos clases)*

$$\mathbf{w}_i(t+1) = \mathbf{w}_i(t) + (\alpha/2).\mathbf{x}^\mu(t_i^\mu - y_i^\mu) \tag{2.25}$$

y llamando $\varepsilon \equiv \alpha/2$, se tiene

$$\Delta\mathbf{w}_i(t) = \varepsilon.\mathbf{x}^\mu(t_i^\mu - y_i^\mu) \tag{2.26}$$

que es la regla del perceptrón (2.21) ya conocida.

Es importante remarcar que el proceso de aprendizaje es iterativo: se parte de una configuración sináptica de partida (de pesos pequeños aleatorios, habitualmente), y se presentan una y otra vez los patrones, para que los pesos se ajusten iterativamente según (2.21), hasta que todos queden bien clasificados. El hiperplano que establece el límite entre dos clases se desplaza lentamente hasta conseguir separarlas por completo

(si ello es posible), como se puede apreciar en la Figura 2.7. El ajuste de los pesos en la iteración t debido a todo el conjunto de aprendizaje será

$$w_{ij}(t+1) = w_{ij}(t) + \sum_{\mu=1}^{p} \Delta w_{ij}^{\mu}(t) \tag{2.27}$$

Rosemblatt demostró que si la función a representar es linealmente separable, este algoritmo siempre converge en un tiempo finito y con independencia de los pesos de partida. Por otra parte, si la función no es linealmente separable, el proceso de entrenamiento oscilará. Una prueba de la convergencia del algoritmo puede encontrarse, por ejemplo, en [Hertz 91]. Por otro lado, el algoritmo del perceptrón se detiene tan pronto como consigue clasificar correctamente todos los ejemplos, por lo que con frecuencia la línea de discriminación queda muy cerca de las muestras de uno de los grupos (en la Figura 2.7 ha quedado cerca de los patrones '0'). Para obtener una discriminación óptima ("en medio" de ambos grupos) se han introducido algoritmos como el denominado **Adatron** (véase, por ejemplo, [Principe 00]).

2.4 LA ADALINA (WIDROW, 1961)

Otro de los modelos clásicos es la **Adalina** (*Adaline*), introducida por Widrow en 1959 [Widrow 60, 88], cuyo nombre proviene de *ADAptive Linear Neuron*[6]. Este modelo utiliza una neurona similar a la del perceptrón, pero de respuesta lineal (Figura 2.8), cuyas entradas pueden ser continuas. Por otra parte, a diferencia del nodo del asociador lineal, el de la adalina incorpora un parámetro adicional denominado *bias*, que traduciremos como umbral, aunque debe tenerse en cuenta que no se trata de un umbral de disparo como el del perceptrón, sino de un parámetro que proporciona un grado de libertad adicional[7]. De este modo, la ecuación de la adalina queda

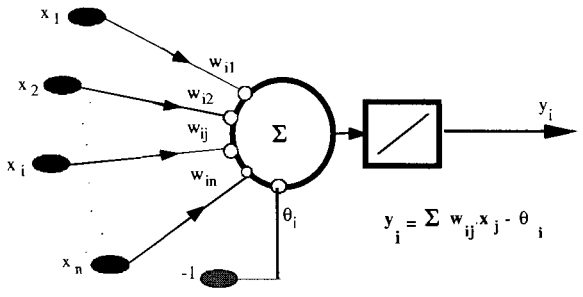


Figura 2.8 *Neurona lineal de la adalina*

[6] Re-bautizado *ADAptive LINear Element* cuando los ANS perdieron popularidad, según comentó el propio Widrow en la conferencia que impartió en la *Neural Network Computation Conference* (Snowbird, Utah, 1987) [Anderson 88].

[7] Un asociador lineal realiza combinaciones lineales de las entradas (rotaciones y dilataciones); la adalina, al incorporar un *bias*, realiza transformaciones afines (las cuales, además de las anteriores, incluyen traslaciones).

$$y_i(t) = \sum_{j=1}^{n} w_{ij} x_j - \theta_i, \ \forall i, \ 1 \leq i \leq m \tag{2.28}$$

No obstante, la diferencia más importante con el perceptrón y con el asociador lineal reside en la regla de aprendizaje que implementa. En la adalina se utiliza la **regla de Widrow-Hoff**, también conocida como **regla LMS** (*Least Mean Squares*, mínimos cuadrados), que conduce a actualizaciones de tipo continuo, siendo la actualización de los pesos proporcional al error que la neurona comete.

Este ANS es un modelo muy conocido y ampliamente utilizado, aunque en ocasiones se hace más referencia a su carácter de dispositivo adaptativo lineal que a su naturaleza neuronal. La adalina se viene utilizando con asiduidad desde los años sesenta como filtro adaptativo, por ejemplo, para cancelar el ruido en la transmisión de señales (un ejemplo clásico es su empleo como supresor de ecos en las comunicaciones telefónicas por satélite [Widrow 88]; para el interesado en profundizar en el tema, una interesante introducción al tratamiento de señal con la adalina se expone en [Freemann 92]). De este modo, y desde hace años, millones de módems en todo el mundo incluyen una adalina.

Su utilidad se ve limitada por tratarse de un sistema lineal. Así, solamente podrá separar correctamente patrones linealmente independientes, fallando en ocasiones ante patrones linealmente separables, que el perceptrón siempre discrimina. No obstante, ante patrones no separables linealmente, los resultados que proporciona son en promedio mejores que los del perceptrón [Widrow 90, Hertz 90, Gallant 93], pues la adalina siempre opera reduciendo el error cuadrático medio al mínimo posible.

2.4.1 Regla LMS

La regla de Widrow-Hoff o LMS, que en un caso particular es conocida como regla delta, constituye el algoritmo de aprendizaje asociado a la adalina. Así como la regla de Hebb es capaz de almacenar sin errores pares de patrones cuyos vectores de entrada sean ortogonales, la regla LMS conducirá a asociaciones perfectas cuando sean linealmente independientes, proporcionando cuando no lo sean una matriz de pesos óptima desde el punto de vista de los mínimos cuadrados. Así, la regla delta puede considerarse en realidad como una versión iterativa aproximada de la basada en la pseudoinversa [Rumelhart 86a], para el caso de vectores estocásticos. No obstante, la forma de derivar la regla LMS a partir de la optimización de cierta función coste, que presentaremos a continuación, ilustrará la forma habitual de obtener algoritmos de aprendizaje en el campo de la computación neuronal.

Aprendizaje como optimización de una función coste

Expondremos en este punto una metodología que permita derivar de forma sistemática reglas de aprendizaje para arquitecturas concretas. El método consistirá en

proponer una función error o coste que mida el rendimiento actual de la red, función que dependerá de los pesos sinápticos. Dada esta función error, introduciremos un procedimiento general de optimización que sea capaz de proporcionar una configuración de pesos que correspondan a un extremal (en general, mínimo) de la función propuesta. El método de optimización aplicado a la función coste proporcionará una regla de actualización de pesos, que en función de los patrones de aprendizaje modifique iterativamente los pesos hasta alcanzar el punto óptimo de la red neuronal.

El método de optimización (minimización) más habitualmente empleado es el denominado **descenso por el gradiente**. Para ello, se comienza definiendo una función coste $E(.)$ que proporcione el error actual E que comete la red neuronal, que será una función del conjunto de pesos sinápticos W, $E=E(\mathrm{W})$, E: $\Re^n \rightarrow \Re$. De esta manera, podemos imaginarnos la representación gráfica de esta función, como una hipersuperficie con montañas y valles (Figura 2.9), en la que la posición ocupada por un valle se corresponde con una configuración de pesos W' localmente óptima, al tratarse de un mínimo local de la función error. El objetivo del aprendizaje será encontrar la configuración de pesos que corresponde al mínimo global de la función error, aunque con frecuencia en una red genérica deberemos conformarnos con un mínimo local suficientemente bueno.

Para encontrar la configuración de pesos óptima mediante descenso por el gradiente se opera del siguiente modo. Se parte en $t=0$ de una cierta configuración W(0), y se calcula el sentido de la máxima variación de la función $E(\mathrm{W})$ en W(0), que vendrá dado por su gradiente en W(0). El sentido de la máxima variación (máximo gradiente) apuntará hacia una colina del paisaje de la hipersuperficie de $E(.)$. A continuación se modifican los parámetros W siguiendo el sentido contrario al indicado por el gradiente de la función error. De este modo se lleva a cabo un descenso por la hipersuperficie del error, aproximándose en una cierta cantidad al valle, un mínimo (local); el proceso se itera hasta alcanzarlo (véase [Rumelhart 86a] para más detalles). Matemáticamente, se expresa del siguiente modo

$$\mathrm{W}(t+1) = \mathrm{W}(t) - \varepsilon.\nabla E(\mathrm{W}) \tag{2.29}$$

donde ε (que puede ser diferente para cada peso) indica el tamaño del paso tomado en cada iteración, que idealmente debe ser infinitesimal. Como una elección de este tipo conduciría a un proceso de entrenamiento extremadamente lento, se toma de un tamaño lo suficientemente grande como para que cumpla el compromiso de rápida actualización sin llevar a oscilaciones (por ejemplo, en la Figura 2.9 se puede ver que una actualización excesivamente grande de los pesos nos llevaría lejos de nuestro objetivo, el mínimo).

Es fácil comprobar matemáticamente que, efectivamente, una actualización de este tipo conduce a un mínimo del funcional de error $E(\mathrm{W})$. Supongamos una matriz de pesos $\mathrm{W}=\{w_{ij}\}$, y calculemos la variación que en $E(\mathrm{W})$ se produce en la iteración t

$$\delta\left(E(w_{ij})\right)=\sum_{ij}\frac{\partial E(w_{ij})}{\partial w_{ij}}\delta w_{ij} \tag{2.30}$$

pero, por (2.24), la variación en los pesos es -ε (idealmente infinitesimal, puesto que por el cálculo de δw_{ij} debe serlo) multiplicado por el gradiente, por lo tanto

$$\delta(f(w_{ij}))=\sum_{ij}\frac{\partial E(w_{ij})}{\partial w_{ij}}\left(-\varepsilon\frac{\partial E(w_{ij})}{\partial w_{ij}}\right)=-\varepsilon\sum_{ij}\left(\frac{\partial E(w_{ij})}{\partial w_{ij}}\right)^2\leq 0 \tag{2.31}$$

luego la variación en la función error es siempre menor que cero, por lo que siempre disminuye. Mediante este procedimiento se asegura alcanzar un mínimo local de la función, aunque puede no coincidir, en general, con el mínimo global (veremos que en el caso de la adalina sí se cumple este hecho).

Primera derivación de la regla de Widrow-Hoff. Aproximación estocástica

A continuación, aplicaremos este método a la adalina, que por ser una red de tipo lineal permite un análisis teórico detallado. Más adelante lo aplicaremos al caso del perceptrón multicapa. La respuesta de una neurona de la adalina es lineal

$$y_i=\sum_{j=1}^{n}w_{ij}x_j-\theta_i,\ \ \forall i,\ 1\leq i\leq m \tag{2.32}$$

y sus salidas son continuas, por ejemplo en [0,+1]. Si definimos $w_{i0}\equiv\theta_i$, y $x_0\equiv -1$, podemos reescribir esta expresión de la manera siguiente

$$y_i=\sum_{j=0}^{n}w_{ij}x_j=\mathbf{w}_i^T\mathbf{x} \tag{2.33}$$

de modo que podemos tratar al umbral θ_i como un peso adicional, pero cuya entrada es constante e igual a -1.

Deduciremos la regla de Widrow-Hoff en el marco del formalismo de la **aproximación estocástica**, que se introdujo para la resolución de complejos problemas de regresión no lineal en los que, o bien se desconoce la estadística del problema, o bien ésta es complicada (véase [Kohonen 89, White 89b] y referencias allí citadas).

Se dispone de un conjunto de patrones de entrada **x**, y salidas deseadas **t**. En general, son variables estocásticas que presentan una cierta dispersión estadística. En la situación más habitual, se desconoce la forma estadística del problema, solamente puede obtenerse un conjunto de muestras $(\mathbf{x}^\mu, \mathbf{t}^\mu)$, $\mu=1,..,p$.

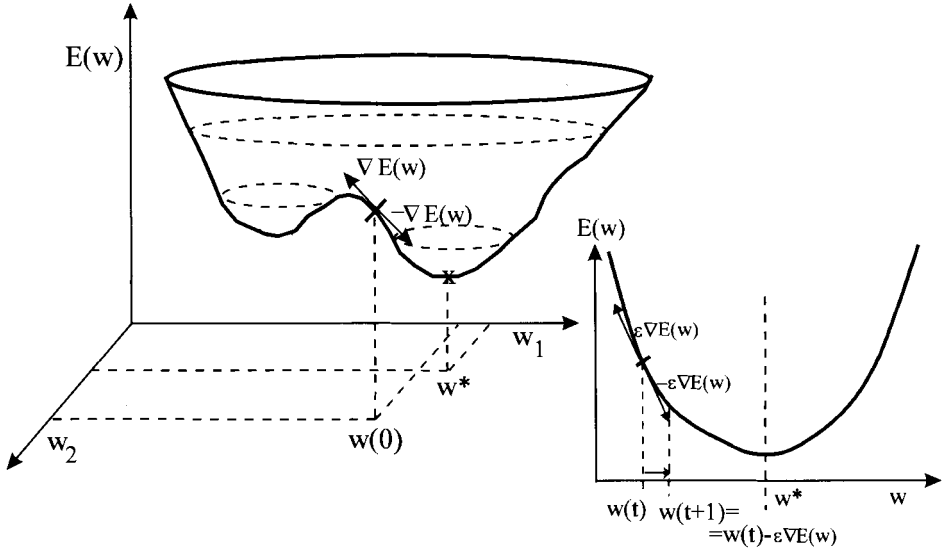


Figura 2.9 *Superficie de error E(w) en el espacio de los pesos w, y descenso por el gradiente hacia un mínimo (local)*

Ante una entrada **x**, la adalina responde con una salida **y** dada por (2.28) que, en principio, no coincidirá con la salida deseada **t**. Planteamos la siguiente función error, que compara las salidas actuales con las objetivo, y que depende de los pesos de la red

$$J \equiv < E[w_{ij}] > = < (t_i - y_i)^2 > = \lim_{p\to\infty}(1/p)\sum_{\mu=1}^{p}\sum_{i=1}^{n}\left(t_i^{\mu} - y_i^{\mu}\right)^2 \qquad (2.34)$$

Obsérvese que al tratar con variables estocásticas, el error se ha definido como un valor esperado. Aunque se ha utilizado como criterio de error el cuadrático medio (lo que se hace con asiduidad en los ANS), debe tenerse en cuenta que el error puede definirse también de otras maneras. Se puede decir que, en general, *la medida de error refleja las hipótesis realizadas sobre la forma del ruido presente en los datos.* El error anterior, como promedio de la suma de errores cuadráticos, supone que los errores de cada variable se distribuyen gaussianamente, y que éstos son independientes [Weigend 93]. De hecho, esta medida del error puede derivarse haciendo uso del principio de máxima verosimilitud a partir de la suposición de que el ruido en los datos del espacio de salida es gaussiano [White 89a] Por ejemplo, en [Hertz 91, Weige 93] se muestra otro tipo de función error, basada en un criterio de **entropía**, y que se aplica cuando los patrones de salida son binarios, con lo que los errores se distribuirán binómicamente.

El problema que nos planteamos consiste en encontrar la configuración de pesos sinápticos w^*_{ij} que minimiza (2.34), es decir

$$\nabla J\,|_{w_{ij}=w^*_{ij}} = \nabla < E[w_{ij}] >|_{w_{ij}=w^*_{ij}} = \left.\frac{\partial < E[w_{ij}] >}{\partial w_{ij}}\right|_{w_{ij}=w^*_{ij}} = 0 \qquad (2.35)$$

Pero si la estadística que rige el fenómeno físico en estudio es desconocida, habrá que realizar los cálculos a partir de las muestras ($\mathbf{x}^\mu$, $\mathbf{t}^\mu$). En general, la resolución directa de este conjunto de ecuaciones no es posible (en [Freemann 93] puede verse un caso muy simple en el que sí es factible), y hay que recurrir al método iterativo de descenso por el gradiente descrito (2.29), en el que la actualización en los pesos adquiere la forma

$$\Delta w_{ij} = -\varepsilon \nabla < E[w_{ij}] >= -\varepsilon \frac{\partial < E[w_{ij}] >}{\partial w_{ij}} \qquad (2.36)$$

Calculando el gradiente

$$\nabla < E[w_{ij}] > = \nabla\left[\lim_{p\to\infty}(1/p)\sum_{\mu=1}^{p}\sum_{i=1}^{n}\left(t_i^\mu - y_i^\mu\right)^2\right] = \lim_{p\to\infty}(1/p)\sum_{\mu=1}^{p}\sum_{i=1}^{n}\nabla\left[\left(t_i^\mu - y_i^\mu\right)^2\right] =$$

$$= \lim_{p\to\infty}(1/p)\sum_{\mu=1}^{p}\sum_{i=1}^{n}2\left(t_i^\mu - y_i^\mu\right)\left(-x_j^\mu\right) = -2 < \left(t_i - y_i\right)x_j > \qquad (2.37 \text{ y } 2.38)$$

Por lo tanto, para estimar el valor esperado del error, habría que tomar una numerosa cantidad de muestras, y promediar. La idea de la aproximación estocástica (y la idea que Widrow y Hoff propusieron) consiste en lo siguiente: en vez de utilizar un valor promedio del gradiente de la función error $<E[w_{ij}]>$, se aproximará su valor en cada iteración por el valor concreto proporcionado por el par ($\mathbf{x}^\mu$, $\mathbf{t}^\mu$) actualmente presentado a la red, es decir

$$\nabla < E[w_{ij}] >= -2 < \left(t_i - y_i\right)x_j > \approx -2\left(t_i^\mu - y_i^\mu\right)x_j^\mu \qquad (2.39)$$

El promedio se irá realizando de un modo automático durante el transcurso de las iteraciones, empleando un conjunto amplio de patrones de aprendizaje. De este modo, se van tomando muestras ($\mathbf{x}^\mu$, $\mathbf{t}^\mu$), y se actualizan los pesos de la forma

$$\Delta w_{ij} = -\varepsilon \nabla < E[w_{ij}] > \approx -\varepsilon\left[-2\left(t_i^\mu - y_i^\mu\right)x_j^\mu\right] = \alpha\left(t_i^\mu - y_i^\mu\right)x_j^\mu \qquad (2.40)$$

con α el ritmo de aprendizaje (que puede depender de la iteración, $\alpha=\alpha(t)$).

La actualización de los pesos queda

$$w_{ij}(t+1) = w_{ij}(t) + \Delta w_{ij} = w_{ij}(t) + \alpha\left(t_i^\mu - y_i^\mu\right)x_j^\mu \qquad (2.41)$$

Ésta es la **regla LMS** o de **Widrow-Hoff**, que puede considerarse la versión iterativa de la regla de la pseudoinversa para el caso de vectores estocásticos. Ambas reglas proporcionan resultados asintóticamente coincidentes.

Si en lugar de considerar la función activación lineal la consideramos sigmoidea, el algoritmo de aprendizaje se denomina **regla delta** [Widrow 90]. Obsérvese que aunque hemos encontrado una forma de actualizar los pesos, esta misma regla se aplica a los umbrales, considerándolos como pesos especiales, con entrada constante e igual a -1.

Widrow y Hoff demostraron [Widrow 60] la convergencia del algoritmo (2.41) a la configuración w^*_{ij} que minimiza el valor esperado del error, a condición de que $\alpha(t)$ cumpla

$$\sum_{t=1}^{\infty} \alpha(t) = \infty, \qquad \sum_{t=1}^{\infty} \alpha^2(t) < \infty \tag{2.42}$$

por ejemplo $\alpha=\alpha(t)=t^{-1}$ satisface ambas condiciones [Kohonen 89]. En muchas ocasiones es suficiente con que α tome un valor pequeño ($0<\alpha<1$). La interpretación de las condiciones (2.42) es que éstas garantizan que el aprendizaje no se lleve a cabo ni excesivamente rápido ni muy lentamente. Como señaló Grossberg [Grossberg 82], el sistema que aprende debe ser suficientemente estable como para recordar los patrones antiguos (un ritmo de aprendizaje excesivamente grande los *borraría*, pues la presente actualización sería de gran magnitud), pero suficientemente plástico como para aprender los nuevos (un ritmo muy pequeño, provoca actualizaciones diminutas, por lo que el proceso de entrenamiento se dilataría excesivamente en el tiempo). Esto constituye el denominado **dilema de la plasticidad frente a la estabilidad** [Grossberg 82]. La primera condición (2.42) asegura que el sistema sea plástico, mientras que la segunda asegura la condición de estabilidad.

Una deducción simple de la regla de Widrow-Hoff

Vamos a exponer ahora una formulación alternativa que, por no considerar explícitamente la estocasticidad del problema, resultará mucho más simple. Para una muestra finita, planteamos la siguiente función error u objetivo

$$E[w_{ij}] = \frac{1}{2} \sum_{\mu=1}^{p} \sum_{i=1}^{n} \left(t_i^{\mu} - y_i^{\mu} \right)^2 \tag{2.43}$$

Mediante esta función se obtiene el error cuadrático medio correspondiente a las salidas actuales de la red respecto de los objetivos. El proceso de optimización es, de nuevo, el del descenso por el gradiente. Para ello calculamos

$$\frac{\partial E[w_{ij}]}{\partial w_{ij}} = -(1/2).2\sum_{\mu=1}^{p}\left(t_i^\mu - y_i^\mu\right)\frac{dy_i^\mu}{dw_{ij}} = -\sum_{\mu=1}^{p}\left(t_i^\mu - y_i^\mu\right)x_j^\mu \qquad (2.44)$$

y el incremento en los pesos queda

$$\Delta w_{ij} = -\varepsilon\frac{\partial E[w_{ij}]}{\partial w_{ij}} = \varepsilon\sum_{\mu=1}^{p}\left(t_i^\mu - y_i^\mu\right)x_j^\mu \qquad (2.45)$$

Esta expresión es, de nuevo, la regla LMS. Conviene observar que mientras que en la regla del perceptrón se llevan a cabo actualizaciones discretas en los pesos, en la adalina la regla LMS produce actualizaciones de tipo continuo, de modo que a un mayor error se tiene una actualización mayor. Otra diferencia entre ambos algoritmos es que la regla del perceptrón converge en un número finito de iteraciones (en cuanto consigue clasificar correctamente todos los patrones), mientras que la regla LMS se acerca asintóticamente a la solución, pues el tamaño de los incrementos se hace cada vez menor. Es importante remarcar que ante patrones no linealmente separables, la adalina proporciona mejores resultados que el perceptrón [Widrow 90, Hertz 90], pues realiza un ajuste óptimo en el sentido de los mínimos cuadrados, mientras que el perceptrón no alcanzará ninguna solución.

Debido a la linealidad de la neurona de la adalina, la función error $E[w_{ij}]$ es cuadrática en los pesos, por lo que define una superficie en forma de paraboloide (una demostración rigurosa aparece, por ejemplo, en [Hecht-Nielsen 90]). Un paraboloide, como la parábola en el plano, usualmente posee un único mínimo, aunque en ocasiones puede presentar una forma degenerada, con uno o más *canales*, pero todos de la misma profundidad. En cualquiera de los dos casos la función $E[w_{ij}]$ es mínima en ese punto o en cualquiera de los de los canales, y la regla (2.45) nos lleva directamente a él, puesto que siempre desciende por la superficie de error [Hech-Nielsen 90]. Por ello, *la regla LMS alcanza siempre el mínimo global*, sin importar la configuración de pesos de partida, constituyendo uno de los pocos casos en redes neuronales en el que se puede realizar una afirmación de este tipo.

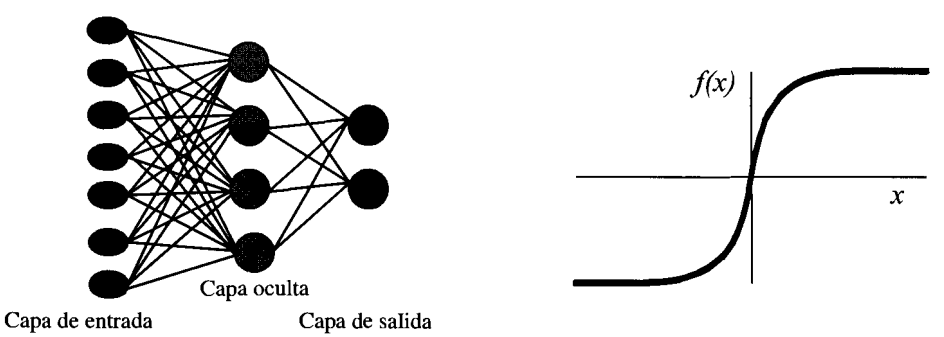


Figura 2.10 *Perceptrón Multicapa y función de transferencia de la neurona*

Por último, no queremos concluir la sección dedicada a la adalina sin citar también la denominada **madalina** (*madaline=many adalines*, es decir *muchas adalinas*), que constituye la versión multicapa de la adalina [Widrow 88, Freemann 92], introducida para superar los problemas de la adalina, en cuanto que red monocapa. Es decir, podemos afirmar que la madalina es a la adalina lo que el perceptrón simple es al multicapa. Recomendamos las referencias citadas al lector interesado en estudiar más ampliamente el modelo.

2.5 EL PERCEPTRÓN MULTICAPA (GRUPO PDP, 1986)

Si añadimos capas intermedias (ocultas) a un perceptrón simple, obtendremos un perceptrón multicapa o MLP (*Multi-Layer Perceptron*). Esta arquitectura suele entrenarse mediante el algoritmo denominado retropropagación de errores o BP, o bien haciendo uso de alguna de sus variantes o derivados, motivo por el que en muchas ocasiones el conjunto *arquitectura MLP + aprendizaje BP* suele denominarse **red de retropropagación**, o simplemente BP.

Como se describe en [Hecht-Nielsen 90], el proceso de desarrollo del BP resulta una curiosa historia de redescubrimientos y olvidos. Al parecer, fue Werbos quien introdujo por primera vez el BP en su tesis doctoral en 1974 [Werbos 74], pero el hecho no tuvo demasiada repercusión en la época. Años más tarde, hacia 1984, el BP fue redescubierto por D. Parker, y casi a la vez (1985) por el grupo del PDP (Rumelhart, Hinton, MacClelland..., [Rumelhart 86b, 86a]), quienes realmente lo popularizaron. Además, existe un procedimiento matemático recursivo empleado en control, de apariencia similar al BP, que data de 1969.

Pese a todo, el mérito del éxito del BP se debe al trabajo del grupo PDP, que lo presentaron a la comunidad internacional como una técnica útil de resolución de problemas complejos [Rumelhart 86a], lo que despertó el interés, no sólo por el perceptrón, sino por el campo de la neurocomputación en general. Los importantes requisitos de cómputo que el algoritmo BP precisa no podían ser satisfechos con los medios disponibles a principios de los setenta, por lo que el *primer descubrimiento* del BP [Werbos 74] era quizás algo prematuro. Por fin en los años ochenta los computadores eran suficientemente potentes como para permitir la aplicación del BP a problemas de interés, lo cual permitió que el grupo PDP pudiera mostrar su gran potencial de aplicabilidad a la resolución de tareas complejas.

La estructura del MLP se presenta en las Figuras 2.10 y 2.11. Denominaremos x_i a las entradas de la red, y_j a las salidas de la capa oculta y z_k a las de la capa final (y globales de la red); t_k serán las salidas objetivo (*target*). Por otro lado, w_{ij} son los pesos de la capa oculta y θ_j sus umbrales, w'_{kj} los pesos de la capa de salida y θ'_k sus umbrales. La operación de un MLP con una capa oculta y neuronas de salida lineal (estructura que constituye, como veremos, un aproximador universal de funciones) se expresa matemáticamente de la siguiente manera

$$z_k = \sum_j w'_{kj}\, y_j - \theta'_i = \sum_j w'_{kj}\, f(\sum_i w_{ji} x_i - \theta_j) - \theta'_i \qquad (2.46)$$

siendo $f(.)$ de tipo sigmoideo (Figura 2.10), como por ejemplo, las siguientes

$$f(x) = \frac{1}{1+e^{-x}} \qquad (2.47a) \qquad\qquad f(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} = \tanh(x) \qquad (2.47b)$$

proporcionando la primera una salida en el intervalo [0,+1], y en el [-1,+1] la segunda.

Ésta es la arquitectura más común de MLP, aunque existen numerosas variantes, como incluir neuronas no lineales en la capa de salida (del mismo tipo que las (2.47), solución que se adopta especialmente en problemas de clasificación), introducir más capas ocultas, emplear otras funciones de activación, limitar el número de conexiones entre una neurona y las de la capa siguiente, introducir dependencias temporales o arquitecturas recurrentes [Werbos 90], etc.

2.5.1 El MLP como aproximador universal de funciones

El desarrollo del MLP durante los últimos treinta años ha resultado curioso. Partiendo de un perceptrón monocapa y observando sus limitaciones computacionales, se llegó a la arquitectura perceptrón multicapa, y aplicándolo a numerosos problemas, se comprobó experimentalmente que éste era capaz de representar complejos *mappings* y de abordar problemas de clasificación de gran envergadura, de una manera eficaz y relativamente simple. Sin embargo, faltaba una demostración teórica que permitiese explicar sus aparentemente enormes capacidades computacionales.

Este proceso histórico comienza con McCulloch y Pitts, quienes mostraron [McCulloch 43] que mediante su modelo de neurona (esencialmente un dispositivo de umbral) podría representarse cualquier función booleana; mucho más tarde, Denker y otros [Denker 87] demostraron que toda función booleana podía ser representada por una red unidireccional multicapa de una sola capa oculta. Por las mismas fechas, Lippmann [Lippmann 87] mostró que un perceptrón con dos capas ocultas bastaba para representar regiones de decisión arbitrariamente complejas (véase la sección 1.3). Por otra parte, Lapedes y Farber demostraron [Lapedes 87] que un perceptrón de dos capas ocultas es suficiente para representar cualquier función arbitraria (no necesariamente booleana). Más tarde, Hecht-Nielsen [Hecht-Nielsen 87, 90] aplicando el teorema de Kolmogorov demostró que una arquitectura de características similares al MLP, con una única capa oculta, resultaba ser un aproximador universal de funciones. Por fin, a finales de la década, diversos grupos propusieron casi a la par teoremas muy similares que demostraban matemáticamente que un MLP convencional, de una única capa oculta (ecuación 2.46), constituía, en efecto, un aproximador universal de funciones [Funahashi 89, Hornik 89]. A título de ejemplo, enunciaremos uno de estos teoremas.

Teorema [Funahashi 89]. Sea $f(x)$ una función no constante, acotada y monótona creciente. Sea K un subconjunto compacto (acotado y cerrado) de $\Re^n$. Sea un número real $\varepsilon \in \Re$, y sea un entero $k \in Z$, tal que $k \geq 3$, que fijamos. En estas condiciones, se tiene que:

Cualquier *mapping* $\mathbf{g}:\mathbf{x} \in K \rightarrow (g_1(\mathbf{x}), g_2(\mathbf{x}),...,g_m(\mathbf{x})) \in \Re^m$, con $g_i(\mathbf{x})$ sumables en K, puede ser aproximado en el sentido de la topología L_2 en K por el *mapping* entrada-salida representado por una red neuronal unidireccional (MLP) de k capas (k-2 ocultas), con $f(x)$ como función de transferencia de las neuronas ocultas, y funciones lineales para las de las capas de entrada y de salida. En otras palabras:

$\forall \varepsilon > 0$, $\exists$ un MLP de las características anteriores, que implementa el mapping

$$\mathbf{g}':\mathbf{x} \in K \rightarrow (g'1(\mathbf{x}), g'2(\mathbf{x}),...,g'm(\mathbf{x})) \in \Re m \tag{2.48}$$

de manera que

$$d_{L_2(K)}(\mathbf{g},\mathbf{g}') = \left(\sum_{i=1}^{m} \int_K | g_i(x_1,...,x_n) - g_i'(x_1,...,x_n) | \; d\mathbf{x} \right)^{1/2} < \varepsilon \tag{2.49}$$

#

Es fácil observar que las funciones sigmoideas empleadas habitualmente en el MLP (ecuación (2.47)) cumplen las condiciones exigidas a $f(x)$. En [Hornik 89] se llega a un resultado similar, considerando funciones de activación sigmoideas, no necesariamente continuas.

En resumen, *un MLP de una única capa oculta puede aproximar hasta el nivel deseado cualquier función continua en un intervalo*[8], por lo tanto, las redes neuronales multicapa unidireccionales son **aproximadores universales de funciones**. A partir de la expresión que define la operación de este tipo de red

$$g_k'(\mathbf{x}) = \sum_j w'_{kj} \, y_j - \theta'_i = \sum_j w'_{kj} \, f(\sum_i w_{ji} x_i - \theta_j) - \theta'_i \tag{2.50}$$

podemos observar que la $\mathbf{g}'(\mathbf{x})$ dada por el MLP representa una cierta función $\mathbf{g}(\mathbf{x})$, como un desarrollo en funciones sigmoideas $f(x)$, lo cual posee una clara analogía con la representación convencional de una función periódica como un desarrollo en serie de Fourier de sinusoides [Principe 00]. También se han establecido paralelismos entre el MLP y otros tipos de transformaciones, como la de Gabor o las *wavelets*.

Los teoremas citados resultan de vital importancia, puesto que proporcionan una sólida base teórica al campo de las redes neuronales, al incidir sobre un aspecto (la

---

[8] El teorema permite elegir k con la restricción $k \geq 3$; si se elige $k=3$ se tiene una sola capa oculta. No obstante, pueden emplearse más capas ocultas, obteniéndose en ocasiones resultados más eficientes o una mejor generalización.