5 Redes Neuronales Multicapa

5.1 El Perceptrón Multicapa

El interés por la investigación en redes multicapa parte de los trabajos de Rosenblatt (1962) sobre Perceptrones y los de Widrow y sus alumnos sobre Madalines (1962). Los Madalines estaban constituidos por varias unidades de entrada y varios elementos Adalides en la primera capa, y con varios dispositivos lógicos (AND, OR,…) en la segunda capa.

Hemos visto que el Perceptrón simple es capaz de resolver problemas de clasificación e implementar funciones lógicas, como por ejemplo, la función OR, pero es incapaz de implementar funciones como la función lógica XOR (función de paridad). Sobre estas limitaciones, Minsky y Papert (1969) publicaron un libro titulado "Perceptrons" que supuso el abandono por parte de muchos científicos de la investigación en redes neuronales, pues no se encontraba un algoritmo de aprendizaje capaz de implementar funciones de cualquier tipo.

Las limitaciones de las redes de una sola capa hicieron que se plantease la necesidad de implementar redes en las que se aumentase el numero de capas, es decir, introducir capas intermediarias o capas ocultas entre la capa de entrada y la capa de salida de manera que se pudiese implementar cualquier función con el grado de precisión deseado, es decir, que las redes multicapa fuesen aproximadores universales. Por ejemplo, con un Perceptrón con dos capas se puede implementar la función lógica XOR. De manera general, vamos a ver que cualquier *función Booleana* de $\{0,1\}^N \rightarrow \{0,1\}$ se puede implementar con un perceptrón multicapa utilizando una sola capa intermedia de unidades de proceso bipolares. Las entradas y las salidas tambien van a ser valores bipolares (-1 ó 1). Para implementar una función Booleana vamos diseñar una red neuronal con tantas unidades de proceso en la capa intermedia como argumentos de la función Booleana a los que le asigna el valor 1 (salida verdadera). Por lo tanto, a cada patrón de entrada que tenga salida igual a uno le asignamos una unidad de proceso cuyos pesos sinápticos van a ser los mismos valores de la entrada y el umbral o sesgo va a ser igual a N. Finalmente, utilizando una unidad de salida bipolar con sus pesos sinápticos iguales a 1 y un umbral o sesgo también igual a 1 conseguimos la representación deseada.

Por ejemplo, si deseamos implementar la función XOR basta con utilizar dos unidades de proceso en la capa oculta, puesto que hay solo dos patrones de entrada a los que le asigna una salida igual a 1: $(-1,1) \rightarrow 1$ y $(1,-1) \rightarrow 1$. Tomando como pesos sinápticos $\mathbf{w}_1 = (-1,1)$ y $\mathbf{w}_2 = (1,-1)$ garantizamos que el potencial sináptico de una neurona oculta sea máximo cuando el patrón de entrada es el que tiene asignado, y su valor será igual a 2. En caso contrario, el potencial sináptico será inferior a 2. Por lo tanto, solo se activará una unidad de proceso oculta cuando el patrón de entrada tenga como salida deseada la unidad y no se activará ninguna en caso contrario. Finalmente, si utilizamos una unidad de salida binaria cuyos pesos sinápticos y sus umbrales sean iguales a uno obtenemos la representación que perseguíamos, puesto que solo se activará la unidad de salida cuando se active una unidad oculta, es decir, cuando al patrón de entrada le corresponda la salida 1 (ver la figura 1).

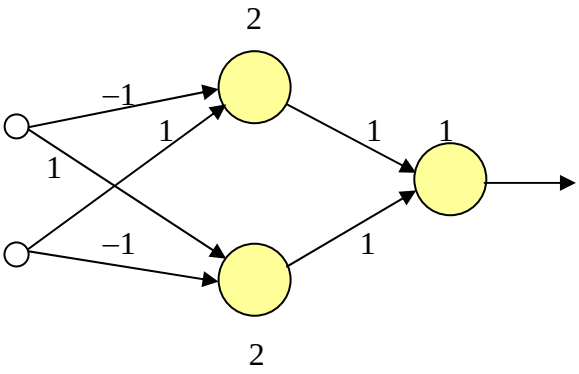


Figura 1. Implementación de la función XOR.

Para entender el perceptrón multicapa tenemos que responder primero a la siguiente pregunta: ¿Qué papel desempeña la capa intermedia? La capa intermedia realiza una proyección de los patrones de entrada en un cubo cuya dimensión viene dada por el número de unidades de la capa oculta. Se trata de realizar una proyección en la que resulten separables linealmente los patrones de entrada de manera que la unidad de salida se pueda realizar una clasificación correcta. En el ejemplo anterior la capa intermedia ha realizado la siguiente transformación: Al patrón de entrada (1, –1) le asigna la salida (1, 0); al patrón de entrada (–1, 1) le asigna la salida (0, 1); al patrón de entrada (1, 1) le asigna la salida (0,0) y al patrón de entrada (–1,–1) le asigna la salida (0,0), como se muestra en la figura 2. Los patrones de entrada no son separables linealmente pero una vez transformados por las neuronas de la capa intermedia sí son ya separables linealmente.

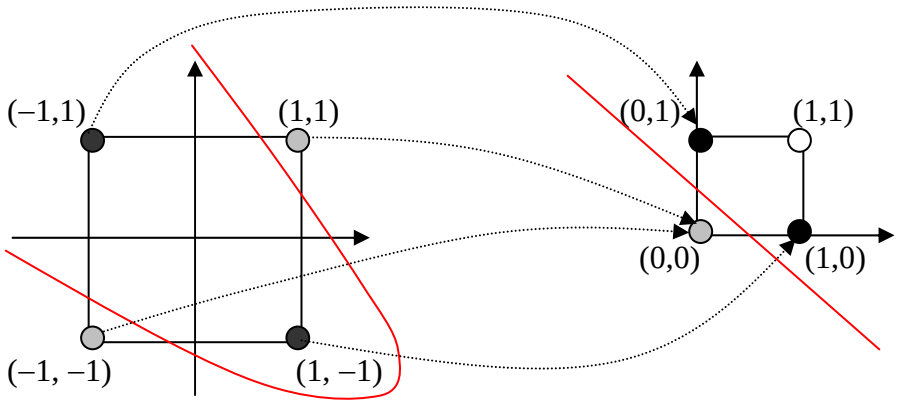


Figura 2. Transformación que realiza la capa intermedia.

Al tener estas redes una topología mas complicada, también se complica la forma para encontrar los pesos correctos, ya que el proceso de aprendizaje es el que decide qué características de los patrones de entrada son representadas por la capa oculta de neuronas. Hemos visto que se puede implementar cualquier función Booleana mediante un perceptrón multicapa. Ahora vamos a ver cómo se puede implementar una función continua cualquiera de R^N en R^M, con el grado de precisión deseado, mediante un perceptrón multicapa. Las entradas serán números reales (señales analógicas) y las unidades de proceso continuas. El problema se centra en desarrollar un algoritmo de aprendizaje que permita la determinación de los pesos sinápticos y de los umbrales a partir de un conjunto de patrones de entrenamiento que nos dan las entradas y salidas deseadas para la red. En 1986 se abrió un nuevo panorama en el campo de las redes neuronales con el redescubrimiento por parte de Rumerlhard, Hinton y Williams del algoritmo de retropropagación. La idea básica de retropropagación fue descubierta por Werbos en su tesis doctoral (1974). Asimismo, algoritmos similares fueron

desarrollados independientemente por Bryson y Ho (1969), Parker (1985) y LeCum (1985). El algoritmo de retropropagación del error es un método eficiente para el entrenamiento de un Perceptrón Multicapa. Se puede decir que puso fin al pesimismo que sobre el campo de las redes neuronales se había puesto en 1969 con la aparición del libro citado de Minsky y Papert. A continuación vamos a estudiar al arquitectura (topología) de este tipo de redes, su dinámica de la computación y la regla de aprendizaje basada en la retropropagación del error cometido por la red al comparar las salidas de la misma con las salidas deseadas.

Topología.

El Perceptrón multicapa es una red de alimentación hacia adelante (feedforward) compuesta por una capa de unidades de entrada (sensores), otra capa de unidades de salida y un número determinado de capas intermedias de unidades de proceso, también llamadas capas ocultas porque no se ven las salidas de dichas neuronas y no tienen conexiones con el exterior. Cada sensor de entrada está conectado con las unidades de la segunda capa, y cada unidad de proceso de la segunda capa está conectada con las unidades de la primera capa y con las unidades de la tercera capa, así sucesivamente. Las unidades de salida están conectadas solamente con las unidades de la última capa oculta, como se muestra en la figura 3.

Con esta red se pretender establecer una correspondencia entre un conjunto de entrada y un conjunto de salidas deseadas, de manera que

$$(x_1, x_2, ..., x_N) \in R^N \rightarrow (y_1, y_2, ..., y_M) \in R^M \quad (1)$$

Para ello se dispone de un conjunto de p patrones de entrenamiento, de manera que sabemos perfectamente que al patrón de entrada $(x_1^k, x_2^k, ..., x_N^k)$ le corresponde la salida $(y_1^k, y_2^k, ..., y_M^k)$, k=1,2,…,p. Es decir, conocemos dicha correspondencia para p patrones. Así, nuestro conjunto de entrenamiento será:

$$\left\{(x_1^k, x_2^k, ..., x_N^k) \rightarrow (y_1^k, y_2^k, ..., y_M^k) : k = 1,2,..., p\right\}$$

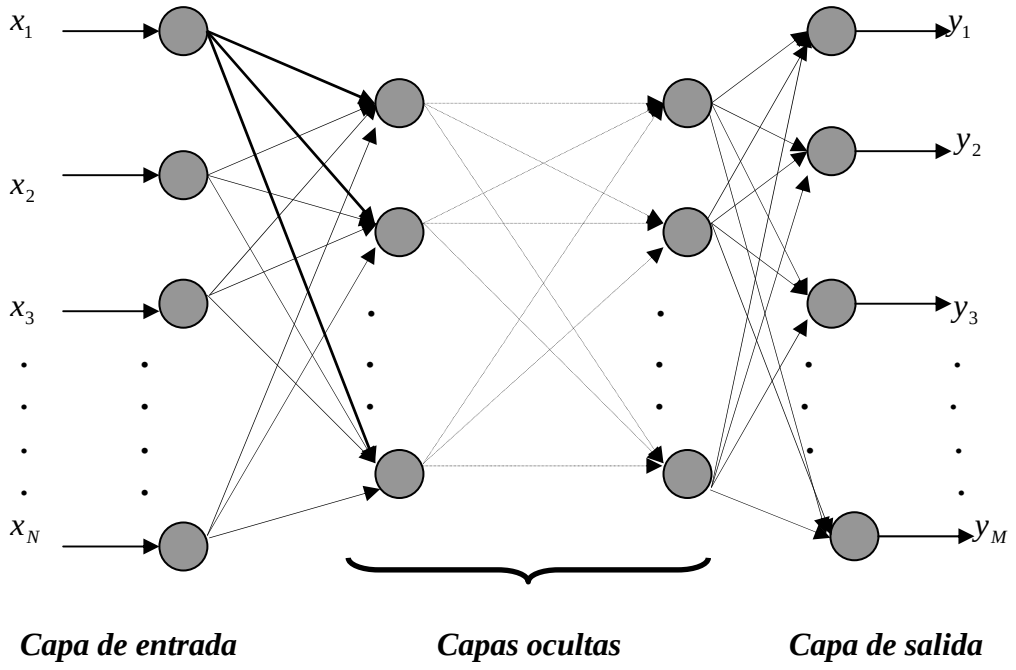


Figura 3. Topología de un Perceptrón Multicapa.

Para implementar dicha relación, la primera capa (sensores de entrada) tendrá tantos sensores como componentes tenga el patrón de entrada, es decir, *N*; la capa de salida tendrá tantas unidades de proceso como componentes tengan las salidas deseadas, es decir, *M*, y el número de capas ocultas y su tamaño dependerán de la dificultad de la correspondencia a implementar.

Dinámica de la computación

Como las entradas a las unidades de proceso de una capa son las salidas de las unidades de proceso de la capa precedente, el Perceptrón multicapa con sólo una capa oculta implementa la siguiente función:

$$y_i = g_1\left(\sum_{j=1}^{L} w_{ij} s_j\right) = g_1\left(\sum_{j=1}^{L} w_{ij}\left(g_2\left(\sum_{r=1}^{N} t_{jr} x_r\right)\right)\right) \qquad (2)$$

donde w_{ij} es el peso sináptico de la conexión entre la unidad de salida *i* y la unidad de proceso *j* de la capa oculta; *L* es el número de unidades de proceso de la capa oculta; g_1 es la función de transferencia de las unidades de salida, que puede ser una función logística, una función tangente hiperbólica o la función identidad; t_{jr} es el peso sináptico que conecta la unidad de proceso *j* de la capa oculta con el sensor de entrada *r* y g_2 es la función de transferencia de las unidades de la capa oculta, que puede ser también una función logística, una función tangente hiperbólica o la función identidad.

Una vez que hemos establecido la topología de la red, y su dinámica de la computación, la determinación de los pesos sinápticos nos llevará al diseño completo de la red. Para ello vamos a seguir un proceso de entrenamiento, mediante el cual vamos introduciendo cada uno de los patrones y evaluando el error que se comete entre las salidas obtenidas por la red y las salidas deseadas; entonces se irán modificando los pesos sinápticos según el error cometido, como vamos a ver a continuación.

Algoritmo de retropropagación: La Regla Delta.

Se trata de determinar los pesos de las conexiones sinápticas entre las unidades de proceso de manera que las salidas de la red coincidan con las salidas deseadas, o por lo menos, sean lo más próximas posibles. Es decir, se trata de determinar los pesos de manera que el error total sea mínimo:

$$E = \frac{1}{2}\sum_{k=1}^{p}\sum_{i=1}^{M}\left(z_i(k) - y_i(k)\right)^2 \qquad (3)$$

Aunque minimizar dicha expresión es igual que minimizarla sin dividir por dos, pero hemos divido por dos para que resulte más simplificada la derivación posterior que vamos a realizar.

El algoritmo de retropropagación utiliza el *método del descenso por el gradiente* y realiza un ajuste de los pesos comenzando por la capa de salida, según el error cometido, y se procede propagando el error a las capas anteriores, de atrás hacia delante, hasta llegar a la capa de las unidades de entrada.

Una característica importante de este algoritmo es su capacidad para organizar el conocimiento de la capa intermedia de manera que se pueda conseguir cualquier correspondencia entre la capa de entrada y la de salida.

El funcionamiento del algoritmo de retropropagación del error es el siguiente: Dado un patrón de entrada se aplica como estímulo a la primera capa de neuronas de la red, se va propagando por las siguientes capas hasta que llega a la capa de salida, donde se compara la

salida obtenida con la deseada. A continuación se calcula el error para cada neurona de salida, y este valor de error es transmitido hacia atrás, por todas las capas intermedias, y se van modificando sus pesos sinápticos según dicho error y los valores de las salidas de las unidades de proceso precedentes ponderados por sus pesos sinápticos.

Supongamos que estamos en la iteración *k* donde hemos introducido el patrón cuya salida de unidad *i* es $y_i(k)$ y la salida deseada $z_i(k)$, siendo los pesos sinápticos $w_{ij}(k)$ y $t_{jr}(k)$. $i=1,2,\ldots,M$, $j=1,2,\ldots,L$, $r=1,2,\ldots,N$. Entonces la regla de modificación de los pesos sinápticos de la capa de salida será:

$$w_{ij}(k+1) = w_{ij}(k) + \Delta w_{ij}(k)$$

donde

$$\Delta w_{ij}(k) = -\eta \frac{\partial E}{\partial w_{ij}(k)} = \eta [z_i(k) - y_i(k)] g_1'(h_i) s_j(k) \qquad (4)$$

$$h_i = \sum_{j=1}^{L} w_{ij}(k) s_j(k)$$

Obsérvese que si tomamos como función de transferencia g_1 la función logística entonces

$$g_1'(h_i) = 2\beta g_1(h_i)[1 - g_1(h_i)] ;$$

si tomamos la función tangente hiperbólica entonces

$$g_1'(h_i) = \beta[1 - g_1(h_i)^2]$$

y si tomamos la función identidad,

$$g_1'(h_i) = h_i$$

Vamos a llamar el término delta a la siguiente expresión:

$$\delta_i^2(k) = g'(h_i)[z_i(k) - y_i(k)]$$

Es la cantidad que se va a ir propagándose hacia atrás.

Por lo tanto,

$$\boxed{\Delta w_{ij}(k) = \eta \delta_i^2(k) s_j(k)} \qquad (5)$$

Ahora vamos a ver la variación de pesos de la capa anterior, utilizando la regla de derivación en cadena,

$$\Delta t_{jr} = -\eta \frac{\partial E}{\partial t_{jr}(k)} = -\eta \frac{\partial E}{\partial s_j(k)} \frac{\partial s_j(k)}{\partial t_{jr}(k)}$$

$$= \eta \sum_{i=1}^{M} [z_i(k) - y_i(k)] g_1'(h_i) w_{ij}(k) g_2'(u_j) x_r(k)$$

$$= \eta \sum_{i=1}^{M} \delta_i^2(k) w_{ij}(k) g_2'(u_j) x_r(k)$$

Si llamamos

$$\delta_j^1(k) = g_2'(u_j) \sum_{i=1}^{M} w_{ij}(k) \delta_i^2(k)$$

tenemos que la variación de peso t_{jr} viene dada por la siguiente expresión

$$\Delta t_{jr}(k) = \eta \delta_i^1(k) x_r(k) \tag{6}$$

El error que hemos calculado es el cometido al introducir el *k*-ésimo patrón, es decir, es un error individualizado porque se realiza para cada patrón, por ello diremos que es la regla de ***entrenamiento individualizado***.

También podemos realizar un ***entrenamiento por lotes***, en cuyo caso se introducen los *p* patrones simultáneamente y se evalúan las salidas de la red comparándolas con las salidas deseadas. En este caso, los pesos sinápticos no dependen de *k* pues se cambian solo después de evaluar todos los patrones. En este caso tomamos también la función de error total, pero la dividiremos por *p* para interpretarla como error cuadrático medio por patrón:

$$E = \frac{1}{2p} \sum_{k=1}^{P} \sum_{i=1}^{M} (z_i(k) - y_i(k))^2$$

Así,

$$\Delta w_{ij} = -\eta \frac{\partial E}{\partial w_{ij}} = \eta \frac{1}{p} \sum_{k=1}^{p} [z_i(k) - y_i(k)] g_1'(h_i(k)) s_j(k) \tag{7}$$

$$\Delta t_{jr} = \eta \frac{1}{p} \sum_{k=1}^{p} \sum_{i=1}^{M} [z_i(k) - y_i(k)] g_1'(h_i(k)) w_{ij} g_2'(u_j(k)) x_r(k) \tag{8}$$

5.2 Aprendizaje con Momentos: Regla Delta Generalizada

Hemos visto que el algoritmo de retropropagación se basa en el método del gradiente, con este método nos vamos acercando al mínimo de la función de error cuadrático mediante el descenso por el gradiente. Sin embargo, la función de error suele tener un gran número de mínimos locales y el algoritmo de aprendizaje puede quedar atrapado en un mínimo local no deseado, puesto que en él el gradiente vale cero. Para prevenir que el algoritmo de aprendizaje quede atrapado en mínimos locales podemos hacer que los cambios de los pesos sinápticos dependan del gradiente medio de los puntos de un entorno, en lugar de depender del gradiente de un solo punto, pero dicha modificación suele requerir un gran esfuerzo computacional y no resultar eficiente. Por ello, Hinton y Williams (1986) propusieron que se tuviera en cuenta también el gradiente de la iteración anterior y realizar un promedio con el gradiente de la iteración actual. Dicho promedio produce un efecto de reducción drástica de las fluctuaciones del gradiente en iteraciones consecutivas, puesto que con frecuencia se produce cierto zigzagueo en el descenso por el gradiente que hace que el algoritmo sea lento (poco eficiente). Por lo tanto, para atenuar en cierta manera trayectos de descenso en zigzag se modifica la regla de retropropagación teniendo en cuenta el descenso seguido en el paso anterior y descendiendo por una dirección intermedia entre la dirección marcada por el

gradiente (en sentido opuesto) y la dirección seguida en la iteración anterior, como se expresa en la siguiente ecuación:

$$\Delta w_j(k) = \alpha\ \Delta w_j(k-1) - \eta \frac{\partial E}{\partial w_j(k)}$$
$$= \alpha\ \Delta w_j(k-1) + \eta \delta^2(k) s_j(k) \qquad (9)$$

que corresponde al incremento del peso sináptico de la componente *j* de la unidad de proceso. Llamamos a α ***constante de momentos*** $(0 \le \alpha < 1)$, pues es la que controla el grado de modificación de los pesos teniendo en cuenta la modificación en la etapa anterior. Cuando $\alpha = 0$ tenemos la regla delta (por eso se la conoce como regla delta generalizada).

La inclusión del término momento en el algoritmo de retropropagación tiende a acelerar la bajada en las direcciones similares de descenso al ir acumulando dichos valores ($\Delta w_{ij}(k)$). Mientras que si tenemos direcciones con oscilaciones de signo en iteraciones consecutivas se ajustaran los pesos en cantidades pequeñas, es decir, tendrá un efecto estabilizador.

5.3 Aceleración del proceso de aprendizaje

El proceso de entrenamiento de una red multicapa suele requerir un gran esfuerzo computacional que va a ser proporcional al resultado de multiplicar el número de pesos sinápticos de la red por el número de épocas de entrenamiento y por el número de patrones de entrenamiento utilizados en cada época. Se han propuesto varios métodos para acelerar el proceso de convergencia de la red. Veamos alguno de ellos.

5.3.1 El algoritmo Quickprop

Este algoritmo fue propuesto por Fahlman (1988) y se basa en el método de Newton. Consiste en aproximar la función de error por una parábola en un entorno de cada punto, es decir, $E(w_j) = aw_j^2 + bw_j + c$. Teniendo en cuenta que:

$$E'(k) = \frac{\partial E(k)}{\partial w_j(k)} = 2aw_j(k) + b$$
$$E'(k-1) = \frac{\partial E(k-1)}{\partial w_j(k-1)} = 2aw_j(k-1) + b$$

resulta que

$$2a = \frac{E'(k) - E'(k-1)}{w_j(k) - w_j(k-1)}$$
$$b = E'(k) - \frac{E'(k) - E'(k-1)}{w_j(k) - w_j(k-1)} w_j(k)$$

El mínimo de la función de error aproximada (parábola) se encuentra resolviendo la ecuación $\frac{\partial E(k+1)}{\partial w_j} = 0$ y comprobando que $\frac{\partial^2 E(k+1)}{\partial w_j^2} = 2a$ es mayor que cero (sino sería un máximo).

Es decir, se encuentra en $w_j(k+1) = -\frac{b}{2a}$. Por lo tanto,

$$w_j(k+1) = w_j(k) + \left[\frac{E'(k)}{E'(k-1) - E'(k)} \right] \Delta w_j(k-1)$$

donde $\Delta w_j(k-1) = w_j(k) - w_j(k-1)$; o lo que es lo mismo,

$$\Delta w_j(k) = \left[\frac{E'(k)}{E'(k-1) - E'(k)} \right] \Delta w_j(k-1)$$

En la figura 4 se ilustra la aproximación parabólica de la función de error y como actúa el algoritmo.

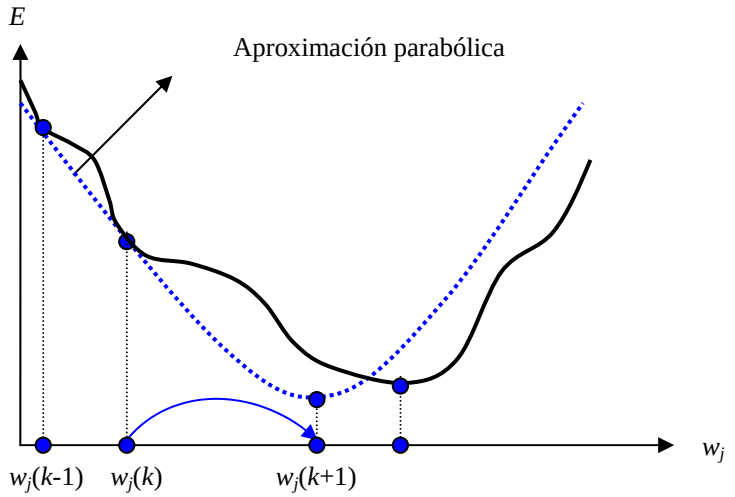


Figura 4. La aproximación parabólica del algoritmo Quickprog.

Sin embargo, el algoritmo no funciona bien cuando el gradiente de la función error no ha decrecido en magnitud ($|E'(k)| \geq |E'(k-1)|$) y, además, no ha cambiado de signo ($\text{sgn}(E'(k)) = \text{sgn}(E'(k-1))$), pues en este caso la parábola ajustada tendría un máximo en ese punto (a<0) y no sería adecuada para esta aproximación, como se muestra en la figura 5, produciendo incremento en la función de error.

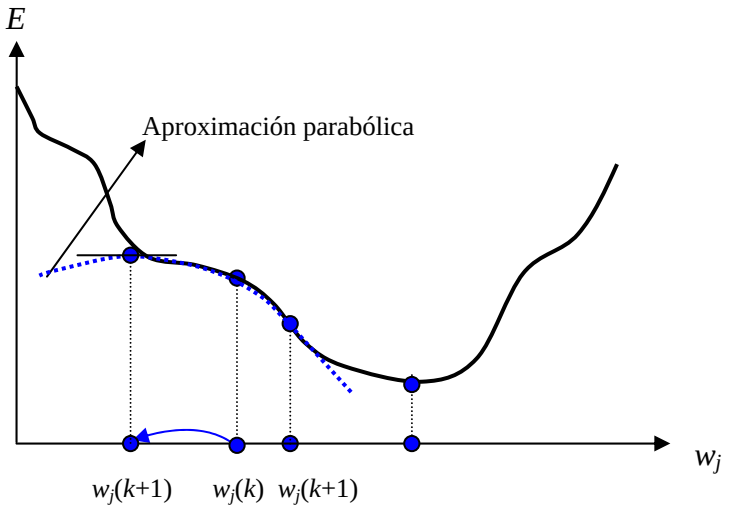


Figura 5. Caso en el que el algoritmo Quickprog no funciona apropiadamente.

5.3.2 El método del gradiente conjugado

El método del gradiente conjugado también se puede aplicar para obtener los pesos sinápticos del perceptrón multicapa que minimizan la función de error. En nuevo vector de pesos sinápticos de la unidad de proceso se determinan de forma iterativa según la expresión:

$$\mathbf{w}(k+1) = \mathbf{w}(k) + \eta(k)\mathbf{d}(k)$$

donde el vector $\mathbf{d}(k)$ nos indica la dirección de búsqueda en la iteración k. En el caso del método del descenso del gradiente, $\mathbf{d}(k) = -\dfrac{\partial E(k)}{\partial \mathbf{w}(k)}$. En el método del gradiente conjugado se busca que una nueva dirección $\mathbf{d}(k)$ que sea conjugada con las k-1 direcciones anteriores, $\mathbf{d}(1)$, $\mathbf{d}(2)$,…,$\mathbf{d}(k\text{-}1)$. Se dice que k vectores no nulos, $\mathbf{d}(1)$, $\mathbf{d}(2)$,…,$\mathbf{d}(k)$, son conjugados (o A-ortogonales) con respecto a una matriz A si son linealmente independientes y $\mathbf{d}(i)^T A\mathbf{d}(j) = 0$, $\forall i \neq j$.

Si utilizamos la aproximación cuadrática q de una función E en un entorno del punto $\mathbf{w}(k)$, obtenida utilizando los tres primeros términos de su desarrollo en serie de Taylor,

$$q(\mathbf{w}) = f(\mathbf{w}_0) + \nabla f(\mathbf{w}_0)^T(\mathbf{w} - \mathbf{w}_0) + \frac{1}{2}(\mathbf{w} - \mathbf{w}_0)^T \mathbf{H}(\mathbf{w}_0)(\mathbf{w} - \mathbf{w}_0)$$

donde $\mathbf{H}$ es la matriz Hessiana de E en el punto $\mathbf{w}_0$, entonces una condición necesaria para que $\mathbf{w}$ corresponda a un mínimo de q es que $\nabla q(\mathbf{w})=0$. Es decir,

$$\nabla E(\mathbf{w}_0) + \mathbf{H}(\mathbf{w}_0)(\mathbf{w} - \mathbf{w}_0) = 0$$

Si existe la inversa de la matriz $\mathbf{H}$ entonces se obtiene un método recurrente (**método de Newton** para el caso multidimensional) para la búsqueda de un mínimo de la función E:

$$\mathbf{w}(k+1) = \mathbf{w}(k) - \mathbf{H}(\mathbf{w}(k))^{-1}\nabla E(\mathbf{w}(k))$$

En general, el método de Newton puede que no converja, aunque si el punto de partida está suficientemente próximo a un punto $\bar{\mathbf{w}}$, tal que $\nabla E(\bar{\mathbf{w}}) = \mathbf{0}$ y $\mathbf{H}(\bar{\mathbf{w}})$ es de rango completo, entonces el método de Newton converge a $\bar{\mathbf{w}}$. Otro inconveniente del método de Newton es que requiere el cálculo de la función Hessiana que suele requerir un gran esfuerzo computacional. Hay modificaciones del método de Newton que garantizan la convergencia independientemente del punto de partida. La búsqueda siguiendo direcciones conjugadas alcanza el punto mínimo en n pasos cuando la función objetivo es cuadrática. La generación de las direcciones conjugadas se lleva a cabo mediante la siguiente regla:

$$\mathbf{d}(k+1) = -\nabla E(\mathbf{w}(k+1)) + \beta(k+1)\mathbf{d}(k)$$

El coeficiente $\beta(k+1)$ se calcula por cualquiera de los métodos siguientes:

- En el método de Fletcher-Reeves

$$\beta(k+1) = \frac{\mathbf{r}_{k+1}^T \mathbf{r}_{k+1}}{\mathbf{r}_k^T \mathbf{r}_k}$$

- En el método de Polar-Ribière

$$\beta(k+1) = \max\left\{\frac{\mathbf{r}_{k+1}^T(\mathbf{r}_{k+1} - \mathbf{r}_k)}{\mathbf{r}_k^T\mathbf{r}_k}, 0\right\}$$

Para determinar la longitud de paso λ en cada iteración se utiliza la siguiente aproximación:

$$E(\mathbf{w} + \lambda\mathbf{d}) \approx E(\mathbf{w}) + \lambda[E'(\mathbf{w})]^T\mathbf{d} + \frac{\lambda^2}{2}\mathbf{d}^T E''(\mathbf{w})\mathbf{d}$$

Igualando a cero la primera derivada,

$$\frac{\partial}{\partial\lambda}E(\mathbf{w} + \lambda\mathbf{d}) \approx [E'(\mathbf{w})]^T\mathbf{d} + \lambda\mathbf{d}^T E''(\mathbf{w})\mathbf{d} = 0$$

obtenemos el valor mínimo de *E* para

$$\lambda = -\frac{\left[E'(\mathbf{w})\right]^T\mathbf{d}}{\mathbf{d}^T E''(\mathbf{w})\mathbf{d}} \tag{10}$$

donde $E''(\mathbf{w})$ es la matriz Hessiana.

Este método utiliza la matriz Hessiana, es decir, la matriz de las derivadas segundas, por lo que es un método de segundo orden que tiene el inconveniente del esfuerzo computacional que requiere el cálculo de la matriz Hessiana en cada iteración, sobre todo en los algoritmos de redes neuronales que contienen multitud de pesos sinápticos (variables). Por ello, se suele emplear el *método de la secante* que consiste en aproximar la derivada segunda por la diferencia de las derivadas primeras en dos puntos vecinos:

$$\frac{\partial^2 E(\mathbf{w} + \lambda\mathbf{d})}{\partial\lambda^2} \approx \frac{E'(\mathbf{w} + \tau\mathbf{d}) - E'(\mathbf{w})}{\tau}$$

Esta cantidad depende del parámetro τ que nos da el radio del entorno. Sustiyendo esta aproximación en la expresión de (10) obtenemos:

$$\lambda = \frac{-\tau\left[E'(\mathbf{w})\right]^T\mathbf{d}}{\mathbf{d}^T\left(E'(\mathbf{w} + \tau\mathbf{d}) - E'(\mathbf{w})\right)\mathbf{d}}$$

5.3.3 Algoritmos de aprendizaje de segundo orden: Modificación de Levenberg-Maquardt.

El algoritmo de Levenberg-Maquardt es una técnica de segundo orden para resolver problemas de optimización que suele ser más eficiente que el método clásico del descenso por gradiente aunque requiere más memoria. Hagan y Menhaj (1994) lo han aplicado a la determinación de los pesos sinápticos en un Perceptrón Multicapa. Si el Perceptrón tiene *N* unidades de salida entonces se tratará de minimizar la función

$$E(w) = \frac{1}{2}\sum_{i=1}^{N}\left[z_i - y_i(\mathbf{w})\right]^2 = \frac{1}{2}e(\mathbf{w})^T e(\mathbf{w})$$

con respecto al vector $\mathbf{w} \in \Re^{M\times N}$, donde

$$e(\mathbf{w})^T = (e_1(\mathbf{w}),...,e_N(\mathbf{w}))$$

$$e_1(\mathbf{w}) = z_i - y_i(\mathbf{w}), \qquad i = 1,2,..., N$$

La derivada de la función $E(\mathbf{w})$ tiene la siguiente forma:

$$g(\mathbf{w}) = \sum_{i=1}^{N} e_i(\mathbf{w}) \nabla e_i(\mathbf{w}) \equiv J(\mathbf{w})^T e(\mathbf{w})$$

donde J es la matriz Jacobiana de $e(w)$:

$$J(w) = \begin{bmatrix} \frac{\partial e_1(\mathbf{w})}{\partial w_{11}} & \cdots & \frac{\partial e_1(\mathbf{w})}{\partial w_{NM}} \\ \frac{\partial e_N(\mathbf{w})}{\partial w_{11}} & \cdots & \frac{\partial e_N(\mathbf{w})}{\partial w_{NM}} \end{bmatrix}$$

La matriz Hessiana viene dada por:

$$H(\mathbf{w}) = \sum_{i=1}^{N} \left[\nabla e_i(\mathbf{w}) (\nabla e_i(\mathbf{w}))^T + e_i(\mathbf{w}) \nabla^2 e_i(\mathbf{w}) \right]$$

$$\equiv J(\mathbf{w})^T J(\mathbf{w}) + S(\mathbf{w})$$

El término S(w) es la parte de $H(w)$ que es una función de las derivadas segundas de $e(w)$.

Como g(w) es el gradiente de la función de error $E(w)$, si tomamos

$$\nabla \mathbf{w} = -\alpha M(\mathbf{w}) g(\mathbf{w})$$

entonces, para $M(\mathbf{w})=HI$ y $\alpha = 1$, obtenemos el método del descenso de mayor pendiente, y para $M(\mathbf{w}) = H^{-1}(\mathbf{w})$ y $\alpha = 1$, el método de Newton.

Un compromiso entre estos dos métodos se consigue al tomar $M = [\mu I + H(\mathbf{w})]^{-1}$ donde μ es algún valor no negativo. Siempre hay un E que hace que M sea definida positiva.

Así, tomaremos, en la iteración k, la dirección de descenso

$$d(k) = -[\mu I + H(\mathbf{w}(k))]^{-1} g(\mathbf{w}(k))$$

y determinaremos los nuevos pesos según la expresión

$$\mathbf{w}(k+1) = \mathbf{w}(k) + \alpha(k) d(k)$$

Un valor ideal de $\alpha(k)$ sería aquel α que minimiza $E(\mathbf{w}(k) + \alpha d(k))$.

Ante la dificultad que presenta a veces el cálculo de la matriz E, si sustituimos la matriz H por el término de primer orden $J(\mathbf{w})^T J(\mathbf{w})$, suponiendo así que $S(\mathbf{w}) \cong 0$, obtenemos el método de Levenberg-Maquardt, en el que:

$$d(k) = -\left[\mu I + J(\mathbf{w}(k))^T J(\mathbf{w}(k))\right]^{-1} g(\mathbf{w}(k))$$

El parámetro μ se incrementa o disminuye en cada paso:

- Si $E(\mathbf{w}(k+1)) \leq E(\mathbf{w}(k))$, entonces en la siguiente iteración el parámetro μ se divide por algún factor β (se suele tomar β=10).

- Si $E(\mathbf{w}(k+1)) > E(\mathbf{w}(k))$ entonces en la siguiente iteración el parámetro μ se multiplica por el factor β (se suele tomar de partida μ=0.01).

Asimismo, si en el método de Newton aproximamos la matriz Hessiana *H(w)* por $J(\mathbf{w}(k))^T J(\mathbf{w}(k))$ (suponiendo $S(\mathbf{w}) \cong 0$), obtenemos el método de Gauss-Newton, donde

$$d(k) = -\left[J(\mathbf{w}(k))^T J(\mathbf{w}(k))\right]^{-1} g(\mathbf{w}(k))$$

Obsérvese que cuando μ es pequeño el algoritmo de Levenberg-Marquardt se corresponde con el algoritmo de Gauss-Newton y cuando μ es grande se corresponde con el algoritmo del descenso de mayor pendiente (con longitud de paso $1/\mu$).

5.4 Elección de la tasa de aprendizaje η

Si la tasa de aprendizaje es demasiado pequeña se garantiza que la función de error disminuya cuando nos movemos en sentido opuesto a la dirección del gradiente, pero la convergencia del algoritmo será muy lenta debido a que estamos dando pasos muy cortos, y por lo tanto necesitaremos muchos pasos; en cambio, si la tasa de aprendizaje es demasiado grande el algoritmo oscilará aumentado y disminuyendo la función de error y no alcanzará un mínimo. En general, se desea que los pasos sean largos cuando el algoritmo se encuentra lejos de un mínimo local de la función de error y que vayan decreciendo conforme se acerca a un mínimo local. La tasa de aprendizaje óptima para una convergencia rápida es el valor inverso del mayor autovalor de la matriz Hessiana, **H,** de la función criterio de error evaluada en el valor concreto del vector de pesos sinápticos **w**. Sin embargo, la determinación de la matriz Hessiana es muy costosa computacionalmente con lo cual resulta un método ineficiente. Una solución a dicho inconveniente consiste en aproximar el producto **Hz**, siendo **z** un vector elegido arbitrariamente (aleatorio), mediante su desarrollo de Taylor:

$$\mathbf{Hz} = \frac{1}{\alpha}\left[\nabla E(\mathbf{w} + \alpha\mathbf{z}) - \nabla E(\mathbf{w})\right]$$

donde α es una constante positiva pequeña. A continuación se aplica el método de la potencia que se usa para determinar el autovector de una matriz que corresponde al mayor autovalor de la misma, que consiste en iterar el proceso

$$\mathbf{z}^{k+1} \leftarrow \frac{\mathbf{Hz}^k}{\left\|\mathbf{z}^k\right\|}$$

Es decir,

$$\mathbf{z} \leftarrow \frac{1}{\alpha}\left[\nabla E(\mathbf{w} + \alpha\frac{\mathbf{z}}{\|\mathbf{z}\|}) - \nabla E(\mathbf{w})\right]$$

El vector **z** converge a $|\lambda_{max}|\mathbf{v}_{max}$, donde $\mathbf{v}_{max}$ es el autovector normalizado de la matriz **H** correspondiente al autovalor mayor λ_{max}. Por lo tanto, la norma del vector **z** obtenido en el proceso iterativo nos da una estimación de $|\lambda_{max}|$ y su valor inverso se puede utilizar como tasa de aprendizaje (este método fue propuesto por Le Cun, Simard y Pearlmutter en 1993, en el trabajo titulado Automatic learning rate maximization by on-line estimation of the Hessian eigvectors, que aparece en *Adnavces in Neural Information Procesing Systems 5* (Denver, 1992), editado por S.J. Hanson, J.D. Cowan, y C.L. Giles, pp.156-163. Morgan Kaufmann, San Mateo, California).

También se han propuesto muchos métodos heurísticos alternativos. Algunos de ellos son los siguientes:

- Incrementar la tasa de aprendizaje cuando $\nabla E(k)$ es próximo a $\nabla E(k\text{-}1)$ y disminuirla en caso contrario (Franzini, 1987)
- Multiplicar la tasa de aprendizaje por $a > 1$ si $\frac{\partial E(k)}{\partial w_i}$ y $\frac{\partial E(k-1)}{\partial w_i}$ tienen el mismo signo y por $b \in (0,1)$ en caso contrario (Silva y Almeida, 1990)
- Mantener la tasa de aprendizaje si $\nabla E(k)$ y $\nabla E(k\text{-}1)$ tiene el mismo signo y reducirla a la mitad en caso contrario (Pflug, 1990)
- Multiplicar la tasa de aprendizaje por un cantidad $a > 1$ (y próxima a uno) cuando haya decrecido la función de error, con el fin de avanzar más rápidamente, y multiplicarla por $b < 1$ (y próxima a 1) en caso contrario.

5.5 Interpretación de la salida de un Perceptrón Multicapa

Cuando se aplica el perceptrón multicapa a resolver problemas de clasificación puede parecer un procedimiento *ad hoc* cuya salida es difícil de interpretar, pero no es así. De hecho, vamos a demostrar que cuando se entrena utilizando el algoritmo de retropropagación del error, que se basa en el criterio de mínimo error cuadrático, entonces la salida del perceptrón va a ser un ajuste de mínimos cuadrados de la distribución *a posteriori*, es decir, de la función discriminante de Bayes.

Si $p(\mathbf{x}/C_1)$ es la distribución de probabilidad de los patrones de la clase C_1 y $p(\mathbf{x}/C_2)$ es la distribución de probabilidad de los patrones de la clase C_2, sabemos que por la formula de Bayes la distribución de probabilidad a posteriori de la clase C_i viene dada por la expresión:

$$p(C_i/\mathbf{x}) = \frac{p(\mathbf{x}/C_i)p(C_i)}{\sum_{j=1}^{2} p(\mathbf{x}/C_j)p(C_j)} = \frac{p(\mathbf{x}/C_i)p(C_i)}{p(\mathbf{x})} \quad , \qquad i=1,2. \tag{11}$$

y que la **regla de decisión de Bayes** consiste en elegir la clase C_i que tiene una mayor probabilidad a posteriori, es decir, dado un patrón **x** lo asigno a la clase C_1 si $p(C_1/\mathbf{x}) > p(C_2/\mathbf{x})$. Es bien conocido que la regla de Bayes minimiza la probabilidad de clasificación incorrecta.

Sea $F(\mathbf{x},\mathbf{t},\mathbf{w})$ la salida de un Perceptrón Multicapa con una capa oculta de neuronas y una neurona de salida (figura 6) con función de transferencia logística (figura 7). Es decir,

$$F(\mathbf{x},\mathbf{t},\mathbf{w}) = \sigma\left(\sum_{i=1}^{M} w_i\, g\left(\sum_{j=1}^{N} t_{ij}\, x_j\right)\right) \tag{12}$$

donde $\mathbf{x}=(x_1,x_2,...,x_N)'$ es el vector de entrada que se va a clasificar, $\boldsymbol{t}=((t_{ij}))$ es la matriz de pesos sinápticos de la primera capa, $\mathbf{w}=(t_1,t_2,...,t_M)'$ es el vector de pesos sinápticos de la última capa, $g(x)$ es una función *sigmoidea* y $\sigma(x)$ la función *logística* que viene dada por la expresión:

$$g(x) \equiv \frac{1}{1+\exp(-2\beta x)} \tag{13}$$

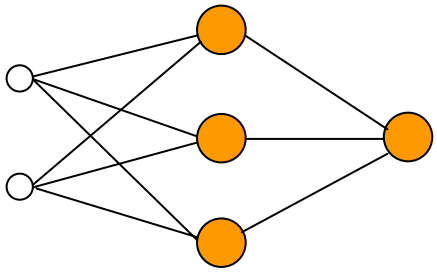


Figura 6. Perceptrón con una única unidad de salida.

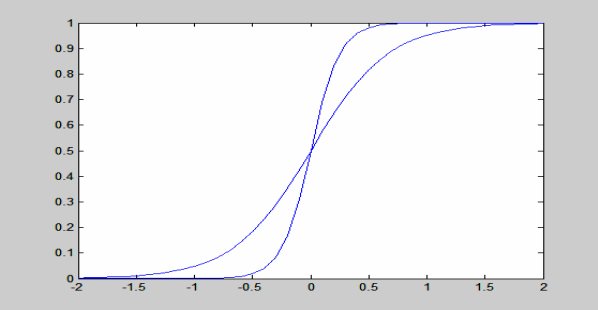


Figura 7. Funciones logísticas.

El parámetro de ganancia β de dicha función controla la pendiente de la curva, es decir, cuanto mayor es β la curva tiene más pendiente y se aproxima más a la función escalón. Se utiliza dicha función como función de transferencia puesto que su derivada, que después vamos a utilizar en la regla de aprendizaje, es muy simple, $g'(x) = 2\beta g(x)[1-g(x)]$, es decir, es una función de la propia función.

Sea **X** el vector aleatorio, con distribución de probabilidad p(**x**), que nos describe todos los vectores que se van a clasificar. Disponemos de un conjunto finito de patrones de cada clase como conjunto de entrenamiento: χ_1 es un conjunto vectores de características (patrones) de la clase C_1 y χ_2 otro conjunto de vectores de características de la clase C_2. La salida deseada de la red para el patrón de entrada **x** de dicho conjunto de entrenamiento será:

$$t(\mathbf{x}) = \begin{cases} 1 & \text{si} \quad \mathbf{x} \in C_1 \\ 0 & \text{si} \quad \mathbf{x} \in C_2 \end{cases}$$

Se desea determinar los pesos sinápticos de la red de manera que la salida de la misma sea igual a 1 para los vectores de C_1 y 0 para los vectores de C_2. Por lo tanto, en el Perceptrón se determinan los pesos sinápticos utilizando el algoritmo de *retropropagación del error* que trata de minimizar el *error cuadrático medio muestral*

$$E_s(\mathbf{t},\mathbf{w}) = \frac{1}{n}\left[\sum_{\mathbf{x}\in\chi_1}(F(\mathbf{x},\mathbf{t},\mathbf{w})-1)^2 + \sum_{\mathbf{x}\in\chi_2}F(\mathbf{x},\mathbf{t},\mathbf{w})^2\right] \tag{14}$$

Teniendo en cuenta que

$$E_s(\mathbf{t},\mathbf{w}) = \frac{n_1}{n}\frac{1}{n_1}\sum_{\mathbf{x}\in\chi_1}(F(\mathbf{x},\mathbf{t},\mathbf{w})-1)^2 + \frac{n_2}{n}\frac{1}{n_2}\sum_{\mathbf{x}\in\chi_2}F(\mathbf{x},\mathbf{t},\mathbf{w})^2$$

y que por la *ley fuerte de los grandes números*,

$$\frac{n_1}{n} \xrightarrow{c.s.} p(C_1), \qquad \frac{n_2}{n} \xrightarrow{c.s.} p(C_2)$$

$$\frac{1}{n_1}\sum_{\mathbf{x}\in\chi_1}(F(\mathbf{x},\mathbf{t},\mathbf{w})-1)^2 \xrightarrow{c.s.} \int_{R^N}(F(\mathbf{x},\mathbf{t},\mathbf{w})-1)^2\, p(\mathbf{x}/C_1)d\mathbf{x}$$

$$\frac{1}{n_2}\sum_{\mathbf{x}\in\chi_2}(F(\mathbf{x},\mathbf{t},\mathbf{w}))^2 \xrightarrow{c.s.} \int_{R^N}(F(\mathbf{x},\mathbf{t},\mathbf{w}))^2\, p(\mathbf{x}/C_2)d\mathbf{x}$$

resulta que cuando $n\to\infty$,

$$E_s(\mathbf{t},\mathbf{w}) \xrightarrow{c.s.} p(C_1)\int_{R^N}(F(\mathbf{x},\mathbf{t},\mathbf{w})-1)^2\, p(\mathbf{x}/C_1)d\mathbf{x} + p(C_2)\int_{R^N}(F(\mathbf{x},\mathbf{t},\mathbf{w}))^2\, p(\mathbf{x}/C_2)d\mathbf{x}$$

Desarrollando el término de la derecha queda,

$$p(C_1)\int_{R^N}(F(\mathbf{x},\mathbf{t},\mathbf{w})^2 + 1 - 2F(\mathbf{x},\mathbf{t},\mathbf{w}))\, p(\mathbf{x}/C_1)d\mathbf{x} + p(C_2)\int_{R^N}(F(\mathbf{x},\mathbf{t},\mathbf{w}))^2\, p(\mathbf{x}/C_2)d\mathbf{x}$$

$$= \int_{R^N}(F(\mathbf{x},\mathbf{t},\mathbf{w})^2\, p(\mathbf{x})d\mathbf{x} + p(C_2)\int_{R^N}(1-2F(\mathbf{x},\mathbf{t},\mathbf{w}))\, p(\mathbf{x}/C_1)d\mathbf{x}$$

$$= \int_{R^N}(F(\mathbf{x},\mathbf{t},\mathbf{w})^2\, p(\mathbf{x})d\mathbf{x} + \int_{R^N}(1-2F(\mathbf{x},\mathbf{t},\mathbf{w}))\, p(C_1/\mathbf{x})\, p(\mathbf{x})d\mathbf{x}$$

$$= \int_{R^N}\left[(F(\mathbf{x},\mathbf{t},\mathbf{w}) - p(C_1/\mathbf{x})\right]^2 p(\mathbf{x})d\mathbf{x} + \int_{R^N} p(C_1/\mathbf{x})(1-p(C_1/\mathbf{x}))\, p(\mathbf{x})d\mathbf{x}\,. \tag{15}$$

El segundo término no depende de $F(\mathbf{x},\mathbf{t},\mathbf{w})$ y el primer término será menor conforme más se aproxime $F(\mathbf{x},\mathbf{t},\mathbf{w})$ a $p(C_1/\mathbf{x})$. Por lo tanto, en el límite de una sucesión de infinitos patrones, la salida del perceptrón multicapa cuando utiliza el aprendizaje por *retropropagación* del error se aproximará a la distribución a posteriori verdadera, $p(C_1/\mathbf{x})$, en el sentido de mínimos cuadrados.

Análogamente, si ponemos dos neuronas en la capa de salida (ver figura 8) de manera que para los patrones de la clase C_1 la salida deseada de la primera neurona sea 1 y la salida deseada de la otra neurona sea 0 y para la clase C_2 ocurra lo contrario, entonces

$$E_s(\mathbf{t},\mathbf{w}) = \frac{1}{n}\left[\sum_{\mathbf{x}\in\chi_1}\left((F_1(\mathbf{x},\mathbf{t},\mathbf{w})-1)^2 + F_2(\mathbf{x},\mathbf{t},\mathbf{w})^2\right) + \sum_{\mathbf{x}\in\chi_2}\left((F_2(\mathbf{x},\mathbf{t},\mathbf{w})-1)^2 + F_1(\mathbf{x},\mathbf{t},\mathbf{w})^2\right)\right]$$

y se llega a que $E_s(\mathbf{t},\mathbf{w})$ converge de forma casi segura a la expresión siguiente:

$$\int_{R^N}\left[(F_1(\mathbf{x},\mathbf{t},\mathbf{w}) - p(C_1/\mathbf{x})\right]^2 p(\mathbf{x})d\mathbf{x} + \int_{R^N} p(C_1/\mathbf{x})(1-p(C_1/\mathbf{x}))\,p(\mathbf{x})d\mathbf{x}$$

$$+ \int_{R^N}\left[(F_2(\mathbf{x},\mathbf{t},\mathbf{w}) - p(C_2/\mathbf{x})\right]^2 p(\mathbf{x})d\mathbf{x} + \int_{R^N} p(C_2/\mathbf{x})(1-p(C_2/\mathbf{x}))\,p(\mathbf{x})d\mathbf{x}$$

Por lo tanto, la salida de la primera unidad de salida se aproxima a la distribución a posteriori $p(C_1/\mathbf{x})$, es decir, $F_1(\mathbf{x},\mathbf{t},\mathbf{w}) \cong p(C_1/\mathbf{x})$, y la salida de la segunda unidad de salida se aproxima a la distribución a posteriori $p(C_2/\mathbf{x})$, es decir, $F_2(\mathbf{x},\mathbf{t},\mathbf{w}) \cong p(C_2/\mathbf{x})$.

Este resultado se puede extender al caso de que tengamos *c* clases diferentes, utilizando una unidad de salida para cada clase.

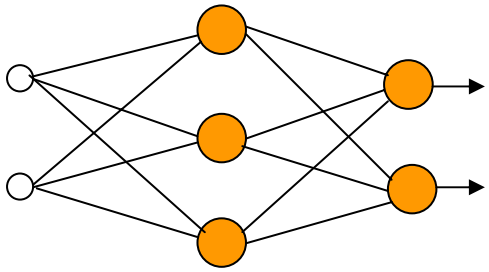


Figura 8. Perceptrón con dos unidades de salida.

Por otra parte, aunque los resultados anteriores nos dicen que las salidas de cada unidad de proceso serán probabilidades (*a posteriori*) para una cantidad infinita de patrones de entrenamiento, en la práctica vamos a disponer de conjuntos finitos de patrones de entrenamiento por lo que las salidas no tienen por qué representar probabilidades. De hecho puede ocurrir que no sumen uno las salidas de las unidades de salida. Por eso, la red puede no ser apropiada cuando se pretende estimar dichas probabilidades, aunque sea adecuada como clasificador.

Una solución a este problema puede ser elegir unidades de salida con función de transferencia no lineal (exponencial) en lugar de logísticas, normalizando las salidas para que su suma sea 1. Por ejemplo, utilizar la salida de la unidad de proceso *i* siguiente:

$$F_i(\mathbf{x},\mathbf{t},\mathbf{w}) = \frac{e^{\sum_{j=1}^{L} w_{ij}s_j}}{\sum_{m=1}^{c} e^{\sum_{j=1}^{L} w_{ij}s_j}} \qquad (16)$$

para *c* unidades de salida.

Este es el método *softmax* que viene a ser una versión suavizada o continua del método "el *ganador se lleva todo*" (winer-take-all) en el que la salida máxima se transforma a 1 y las demás a 0.

5.6 El Perceptrón Multicapa como aproximador universal

El resultado principal que vamos a ver aquí es que un perceptrón con el número suficiente de unidades de proceso ocultas en una sola capa intermedia (oculta) que tienen como función de transferencia una función sigmoidea y con una función de transferencia lineal en la unidad de salida es capaz de aproximar cualquier función continua f de R^n en R con el grado de precisión deseado.

El resultado clásico que motivó la aplicación de las redes neuronales como aproximadotes universales fue el teorema de Kolmogorov establecido en 1957, cuyo enunciado es el siguiente:

Teorema de Kolmogorov
Cualquier función continua $f(x_1, x_2,\ldots,x_n)$ definida en $[0\ 1]^n$, $n\geq 2$, se puede representar mediante la expresión

$$f(x_1, x_2, ..., x_n) = \sum_{i=1}^{2n+1} g_i \left[\sum_{j=1}^{n} \phi_{ij}(x_j) \right]$$

donde las funciones g_i son funciones continuas y reales de una sola variable, elegidas adecuadamente, y las funciones ϕ_{ij} son continuas y monótonas crecientes independientes de f.

Este resultado establece que cualquier función vectorial continua de R^n en R^m definida sobre un conjunto compacto se puede expresar en términos de sumas y composiciones de un número finito de funciones de una sola variable.

Sin embargo, las redes neuronales constituyen una clase de funciones más reducida que la establecida por el teorema de Kolmogorov ya que limitan las funciones ϕ_{ij} a funciones sigmoidales. Una demostración rigurosa de que las redes neuronales son aproximadotes universales fue dada independientemente por Cybenko (1989), Funahashi (1989) y por Hornik, Stinchcombe y White (1989).

Teorema de Cybenko (1989)
Sea φ cualquier función sigmoidea (por ejemplo, la función logística). Dada cualquier función continua en $[0\ 1]^n$ (o en cualquier conjunto compacto de R^n) y un $\varepsilon > 0$, existe unos vectores $\mathbf{w}_1, \mathbf{w}_2,\ldots,\mathbf{w}_N$, $\boldsymbol{\alpha}$ y $\boldsymbol{\theta}$, y una función parametrizada $G(\cdot, \mathbf{w}, \boldsymbol{\alpha}, \boldsymbol{\theta})$ de $[0\ 1]^n$ en R tal que

$$\left| G(\mathbf{x}, \mathbf{w}, \boldsymbol{\alpha}, \boldsymbol{\theta}) - f(\mathbf{x}) \right| < \varepsilon, \quad \forall \mathbf{x} \in [0,\ 1]^n$$

donde

$$G(\mathbf{x}, \mathbf{w}, \boldsymbol{\alpha}, \boldsymbol{\theta}) = \sum_{i=1}^{N} \alpha_j \varphi\left(\mathbf{w}_j^T \mathbf{x} + \theta_j \right)$$

y $\mathbf{w}_j \in R^n$, α_j, $\theta_j \in R$, $\mathbf{w} = (\mathbf{w}_1, \mathbf{w}_2,\ldots,\mathbf{w}_N)$, $\boldsymbol{\alpha} = (\alpha_1, \alpha_2,\ldots, \alpha_N)$ y $\boldsymbol{\theta} = (\theta_1, \theta_2,\ldots, \theta_N)$.

Dicho resultado establece que un perceptrón multicapa con una única capa intermedia de unidades ocultas es capaz de aproximar uniformemente cualquier función multivariante con el grado de precisión deseado. Por lo tanto, cuando se aplica el perceptrón multicapa a un problema real para aproximar una función continua y no se consigue la precisión deseada es porque no se ha conseguido una determinación adecuada de los pesos sinápticos de la red o no se ha utilizado el número apropiado de unidades de proceso en la capa oculta.

Hornik, Stinchcombe y White (1990) también demostraron que estas redes no solo pueden aproximar una función sino también su derivada. Además, dichas redes puede aproximar funciones que no son diferenciables en el sentido clásico pero que poseen una derivada generalizada, como ocurre con las funciones diferenciables a trozos.

5.7 El Perceptrón Multicapa como clasificador universal

En 1989, Cybenko demostró también que el perceptrón multicapa con una sola capa intermedia que tiene un número suficiente de unidades de proceso con función de transferencia una función sigmoidea y con una unidad de salida que tiene una función de transferencia lineal es un clasificador universal, es decir, puede aproximar, con la precisión que se desee, cualquier función de la forma:

$$f(\mathbf{x}) = j \text{ siempre y cuando } \mathbf{x} \in P_j$$

donde f es una función de $\mathbf{A}^n$ en el conjunto $\{1,2,\ldots,k\}$, $\mathbf{A}^n$ es un conjunto compacto (cerrado y acotado) de $\boldsymbol{R}^n$ y $P_1, P_2,\ldots, P_k$ es una partición de $\mathbf{A}^n$ en k subconjuntos disjuntos.

5.8 Validación

Para estudiar el clasificador diseñado se utiliza el **método de validación cruzada** (cross-validation) introducido por Cover (T. M. Cover. Learning in pattern recognition. In Satoshi Watanabe, editor, *Methodologies of Pattern Recognition*, pages 111-132. Academic press, New York, 1969). Dicho método consiste en dividir los datos muestrales en dos partes; una parte se utiliza como conjunto de entrenamiento para determinar los parámetros del clasificador neuronal y la otra parte, llamada *conjunto de validación*, se utiliza para estimar el error de generalización, es decir, la tasa de clasificación incorrecta del clasificador con datos diferentes a los utilizados en el proceso de entrenamiento. Es importante que el conjunto de validación no contenga datos utilizados en la fase de entrenamiento (un error metodológico conocido como "comprobación sobre el conjunto de entrenamiento"). Se suelen utilizar el 90% de los datos para entrenar la red y el 10% restante como conjunto de validación.

Como el objetivo final es que el clasificador consiga un error de generalización pequeño entonces se entrenará el clasificador hasta que alcance un mínimo de dicho error de validación. Para muchos problemas, el comportamiento típico del error de entrenamiento de un clasificador decrece monótonamente durante la fase de entrenamiento, como se muestra en la figura 9, mientras que el error sobre el conjunto de validación decrece hasta un punto a partir del cual crece, lo que indica que a partir del mismo el clasificador realiza un *superajuste* sobre los datos de entrenamiento. Por ello, el proceso de entrenamiento debe finalizar cuando se alcance el primer mínimo de la función del error de validación.

Una generalización de este método, llamada validación cruzada con n pliegues, consiste en dividir aleatoriamente el conjunto de entrenamiento en n conjuntos disjuntos de igual tamaño, p/n. El clasificador se entrena n veces, cada vez con un conjunto de validación diferente y se toma como tasa de error el valor medio de las n tasas de error obtenidas.

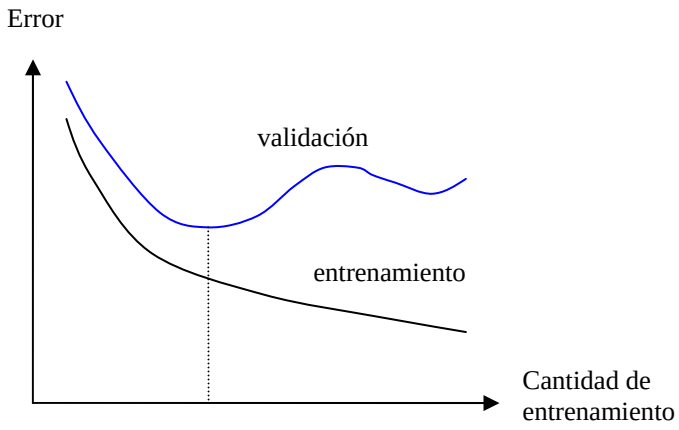


Figura 9. Errores de entrenamiento y validación.

5.9 Aplicaciones

5.9.1 Datos de RECIDIVA

Estos datos constan de 18 características del melanoma que ha sido quitado quirúrgicamente a cada una de 151 personas. Después de un periodo de 6 años se ha comprobado qué personas han recaído (recidiva). Se trata de predecir si una persona va a tener recaídas en un periodo posterior de la intervención quirúrgica de 6 años basándose en esas 18 características.

En la figura 9 se muestra la representación de los datos en sus dos primeras componentes principales, indicando con el símbolo '+' los que corresponden a recidiva y con el símbolo '·' los que no.

Se ha utilizado un perceptrón multicapa con 18 sensores de entrada, 4 unidades de proceso en la capa oculta y una unidad de salida. Para el aprendizaje se han utilizado 136 observaciones. En 8 épocas de entrenamiento se alcanza un error cuadrático inferior a 0.00434981. La curva de decrecimiento de error durante el aprendizaje se muestra en la figura 10. Para comprobar la capacidad de generalización de la red se han tomado las 15 muestras restantes (10%) y se ha comprobado como el número total de clasificaciones incorrectas para las 151 observaciones es cero.

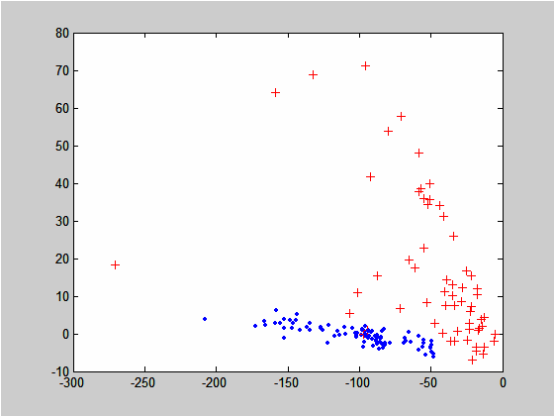


Figura 10. Representación en sus dos primeras componentes principales.

5.9.2 Datos PIMA

Esta base de datos contiene 9 características basadas en datos personales y en resultados de exámenes médicos de 532 mujeres indias, mayores de 21 años, que vivían cerca de Phoenix (Arizona) para estudiar la diabetes de acuerdo la los criterios de la Organización Mundial de la Salud. De las 9 características hemos seleccionado 7 (número de embarazos, concentración de glucosa, presión sanguínea diastólica, grosor de la piel en el triceps, insulina, índice de masa corporal y la edad en años). La característica novena vale uno si la mujer es diabética y cero si no lo es. Se trata de predecir si una mujer es, o no, diabética basándose en las 7 características seleccionadas. Las 200 primeras muestras se han utilizado para el entrenamiento de la red constituida por 7 sensores de entrada, 15 unidades de proceso en la capa oculta (tomadas en base a la experimentación) con función de transferencia la función tangente hiperbólica y una unidad de salida que utiliza la función logística como función de transferencia. Los pesos sinápticos han sido determinados mediante el algoritmo de la regla delta generalizada. Una de las soluciones encontradas consigue clasificar incorrectamente a 26 patrones de los 200 patrones en tan sólo 24 épocas de entrenamiento.

Si utilizamos los 180 primeros patrones para el entrenamiento de la red, una de las soluciones encontradas consigue 25 clasificaciones incorrectas de los 180 patrones en 24 épocas de entrenamiento. Si utilizamos los 20 patrones restantes como conjunto de validación para ver la capacidad de generalización de la red encontramos que clasifica incorrectamente a 6 de los 20 patrones (30%). La figura 11 muestra la evolución del error cuadrático en las 24 primeras épocas, a partir de las cuales la red se estabiliza puesto que el gradiente que nos da el valor de actualización de los pesos sinápticos es prácticamente cero.

Los pesos sinápticos de la primera capa son:

W1 =

0.4858	0.6285	-0.1325	0.0278	-0.1851	-0.6218	-0.7860
-0.3899	-0.0039	-0.4284	-0.2864	0.7529	-0.0445	-0.7848
1.0777	-0.0308	-0.1237	-1.0837	1.1821	0.5931	0.1606
0.1076	0.1275	-0.2808	-0.2642	1.1839	0.1743	-0.5810
-0.1318	0.3152	0.1215	-0.0401	0.1914	-0.1255	-0.7730
0.9491	0.0931	0.0667	1.5683	-0.8449	-0.4000	-1.3429
0.4756	0.0139	0.0459	-0.0872	0.2207	0.4754	-0.2023
0.4823	-0.2214	0.4757	0.3966	0.5582	0.1984	0.2832
0.0106	-0.0087	-0.0239	-0.0021	0.0023	-0.8396	0.0129
-0.1274	-0.1483	0.5260	-0.4451	0.8116	-0.0213	-1.2768
-1.6608	0.0097	-0.1166	-0.0828	0.0273	-0.5957	0.3786
0.0041	-0.0725	-0.1005	-0.1916	-0.0631	0.0683	-0.1123
1.1836	0.1413	0.0245	0.0704	-0.6884	-0.0408	0.7135
0.9585	0.1508	0.1700	-0.3429	-0.3922	0.4199	-0.4280
1.6352	-0.0437	-0.2414	-0.1180	0.1925	0.2150	0.2309

el sesgo,

1.2600
1.0592
1.6264
0.1228

-2.9812
-2.2371
-1.1710
-2.0635
2.8125
-0.6704
-1.9751
-0.1821
1.3062
0.7628
1.5146

los de la segunda capa, W2,

0.2126 0.0683 -0.2299 0.8454 0.5351 -0.2546 0.2105 0.0589 -2.7998 -0.7581
0.6980 -0.0289 -0.1227 -0.2077 1.4184

y el sesgo b2 = -0.8262.

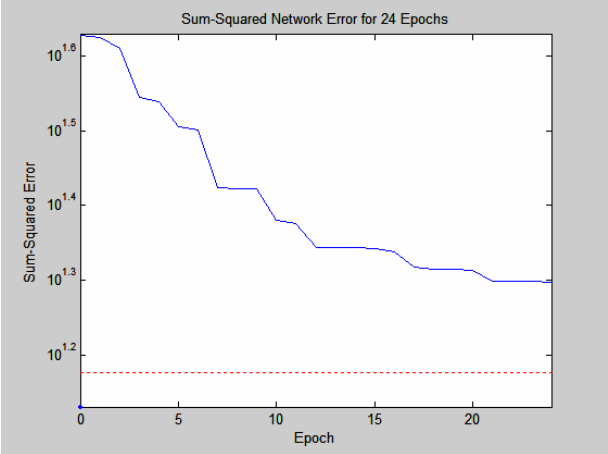


Figura 11. Evolución del error cuadrático.

Asimismo, otra solución encontrada en 21 épocas de entrenamiento consigue 34 clasificaciones incorrectas de los 180 patrones utilizados, pero su capacidad de generalización para los 20 patrones restantes no utilizados en el entrenamiento es de sólo 4. En la figura 12 se presenta la evolución del error cuadrático.

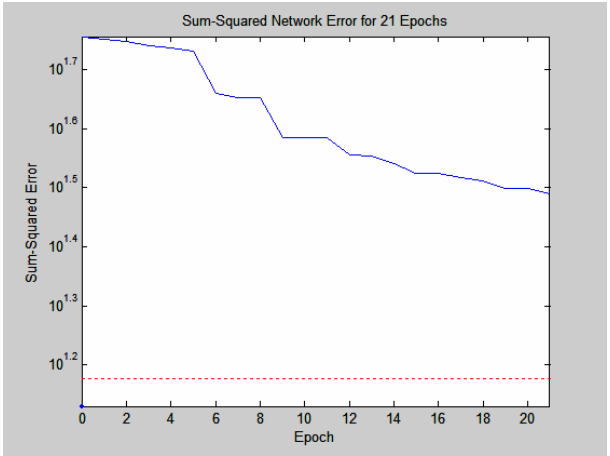


Figura 12. Evolución del error cuadrático.

Los pesos correspondientes son:

W1 =

0.9743	-1.2236	-0.5486	-0.5368	-0.7969	3.7587	0.0938
1.7824	0.7654	0.4027	1.0070	-2.2330	14.1620	0.2915
-6.8767	0.1815	0.6050	0.1745	0.1923	2.5899	1.3829
0.7483	-0.1852	-0.2957	-0.6864	0.6980	5.7074	-0.7147
-4.5352	-0.7071	0.6371	-0.4602	0.7169	-4.7118	1.4213
5.5077	-0.1403	0.1432	1.8649	-1.5639	0.9295	2.1004
5.0267	-0.4064	0.8031	-0.7507	1.6509	-6.7135	-1.6839
-9.4174	-0.8730	-0.6255	-0.9331	0.9061	-11.1677	3.7675
-1.2051	-1.7949	-0.0414	-0.6419	0.4024	14.6567	0.3722
1.2209	0.5477	0.1653	-0.7024	0.5564	-20.9413	0.2319
-9.8622	0.5622	1.4869	2.4050	-4.8697	-2.2580	1.2455
1.1941	-0.4631	0.5025	-2.2257	5.4896	-5.5059	0.1349
-5.2663	-0.2148	-0.2763	3.4970	-0.8887	0.9874	-0.8539
-0.4789	0.1684	0.1668	3.9973	-3.5513	-7.1269	-5.6749
1.3567	-0.0082	0.1627	0.1557	-0.1991	6.1799	-0.4086

W2 =

7.9140 11.0773 2.2166 -1.3982 -0.5943 -0.8034 -9.9048 -1.0340 -3.5285 - 2.6504 -6.0204 -9.2823 -9.1139 -5.8628 5.3814

b1 =

8.7434
-5.3387
-2.1929
8.4715
1.5662
1.5137
8.8589
-1.8640
1.2668
-1.8919
3.9033
-0.8009
-3.7432
1.5862
-10.4264

y b2 = 5.8551.

El conjunto de validación se utiliza con el fin de asegurar la capacidad de generalización de la red y evaluar la calidad de la red durante el proceso de entrenamiento (llamada validación cruzada); se necesita para analizar si hay fenómeno de superajuste (o sobreentrenamiento), es decir, cuando una red entrenada con los mismos patrones consigue con ellos un número reducido de clasificaciones incorrectas pero es peor que otra que consigue un número mayor

en su capacidad de generalización, puesto que se ha concentrado en peculiaridades del conjunto de entrenamiento a costa de perder muchas de las regularidades necesarias para una buena generalización.

5.9.3 Datos de VINOS

Esta base de datos está constituida por 178 patrones, cada uno contiene 13 características del vino; los 59 primeros corresponden a una clase de vino, los 71 siguientes a otra y los 48 últimos a una tercera clase. Se ha utilizado un perceptrón multicapa con 13 sensores de entrada, sólo 3 unidades de proceso en la capa oculta y tres unidades de proceso en la capa de salida, una para cada clase, de manera que la salida deseada para un patrón de la primera clase es del tipo (1 0 0), es decir, 1 es la salida de la primera unidad de salida y cero la de las otras dos. Se han utilizado 160 patrones para el entrenamiento de la red y 18 (10%) para su validación. En sólo 10 épocas de entrenamiento se ha encontrado una solución que consigue cero clasificaciones incorrectas tanto para el conjunto de entrenamiento como para el conjunto de validación. En la figura 13 se muestra la evolución del error cuadrático de la red en las 10 épocas de entrenamiento. Los pesos sinápticos encontrados son los siguientes:

W1 =

-0.1944 0.4696 1.7331 -0.2956 0.0046 -0.2013 1.2478 -0.0525 -0.8279 -0.1674 -0.7454 0.3278 0.0080
-0.0861 0.5000 0.8783 -0.0867 0.0077 0.1097 -0.9583 -1.3499 -0.2856 0.5341 -1.2643 -0.5861 0.0024
-0.0055 0.0068 -0.1578 -0.0137 -0.0010 -0.0132 0.2486 -0.0789 0.0261 0.0031 0.4819 0.2341 0.0003

W2 =

4.8716 0.0010 -1.7799
-3.3730 -4.2823 -1.2516
-0.9858 3.4819 -3.9164

b1 =

-4.1902
-0.8760
-0.3445

b2 =

0.5802
-2.5466
-0.8875

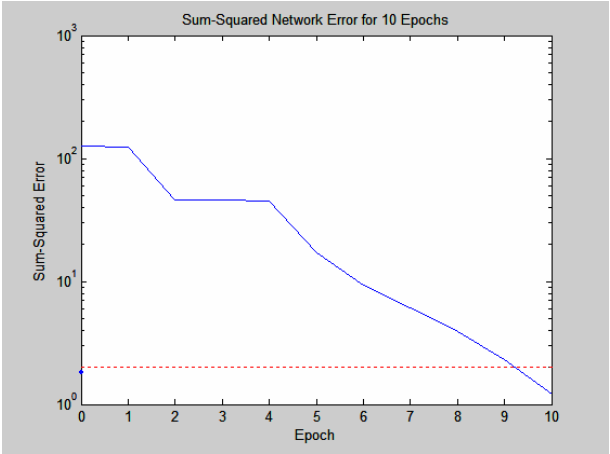


Figura 13. Evolución del error cuadrático.

5.9.4 Datos IRIS

La base de IRIS consta de 150 patrones, cada uno constituido por los valores de 4 características de hojas de tres tipos de lirios. Para predecir el tipo de lirio según dichas características se ha utilizado un perceptrón multicapa con 4 sensores de entrada, 10 unidades de proceso en la capa oculta y 3 unidades de salida. La salida deseada para un patrón de la primera clase es (1 0 0), es decir, la primera unidad de salida debe tomar el valor 1 y las demás cero; para la clase 2 debe ser (0 1 0) y para la clase 3 la salida (0 0 1). La red se ha entrando con el 90% de los datos de cada clase, es decir, 45 datos por clase, y el conjunto de validación está constituido por los 15 restantes (5 por clase). Después de 150 épocas de entrenamiento la red ha encontrado los siguientes pesos sinápticos:

W1 =

-0.1181	-0.3203	-1.3543	-0.6693
0.7473	-0.0010	-1.0781	1.5804
0.1427	-0.4478	1.4147	1.4359
-2.1036	7.1013	-0.7474	-4.4170
0.7563	-1.2584	1.6291	0.6776
-0.5431	0.0169	3.6037	0.9457
-0.0119	1.0728	0.8450	0.8468
0.7321	-0.2218	0.2551	-0.5852
0.8239	-0.0441	0.6436	0.1737
8.6185	-1.9219	-9.8375	-4.3685

W2 =

0.7074	1.4425	-6.3092	0.6826	-1.8839	-0.3120	-0.4119	1.2890	-0.4112	1.0356
1.6040	-2.4268	9.4841	3.9259	-1.6290	-0.1364	-0.6412	-2.2303	-0.7816	4.5983
-1.2742	3.0682	-0.5510	-5.5143	-1.9031	-1.4499	-1.8799	-1.0711	-0.7027	-9.1279

b1 =

-3.0157
-3.1013
-5.2344

1.5110
2.0673
2.2105
-0.9351
1.8922
0.0428
5.3556

b2 =

0.7694
1.0506
-2.6554

En la figura 14 se muestra la evolución del error cuadrático medio durante el proceso de aprendizaje y se puede observar cómo consigue una tasa de clasificación incorrecta para los patrones de entrenamiento de sólo 2 patrones (un poco antes de la época 150). Asimismo, cuando se aplica la red a los 15 patrones de validación se consiguen cero clasificaciones incorrectas.

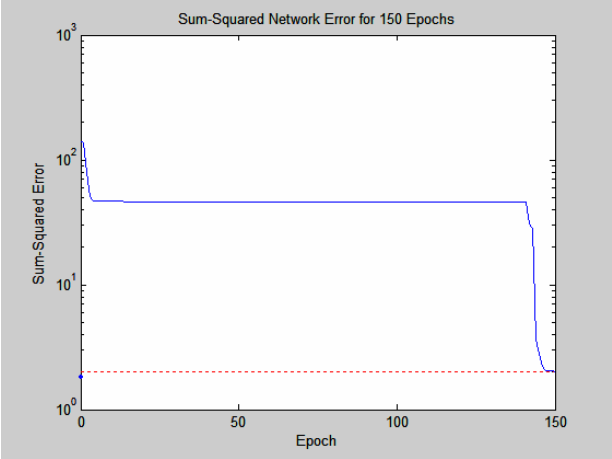


Figura 14. Evolución del error cuadrático.

5.10 Redes neuronales con funciones de base radial

Ahora vamos a describir un modelo de red neuronal artificial donde las unidades de proceso (nodos) tienen una respuesta afinada localmente, como ocurre en muchas neuronas del sistema biológico nervioso. Estas células nerviosas tienen características de respuesta que son "selectivas" para algún rango finito del espacio de las señales de entrada. El modelo que presentamos está motivado por el artículo sobre funciones de base radial de Medgassy (1961) y las aplicaciones a interpolación de funciones de Micchelli (1986) y Powell (1987), a la estimación de la función de densidad, de Parzen (1962) y Specht (1990), y a la aproximación de funciones multivariantes suaves, de Poggio y Girosi (1989).

Una función es **simétrica radialmente** (radial basis function (RBF)) si sus valores (salida) dependen de la distancia de su argumento (vector de entrada) a un vector almacenado propio de la función.

Una función radialmente simétrica muy utilizada es la *Gaussiana*, que viene dada por la expresión:

$$\varphi_1(u) \propto \exp(-u^2/\sigma^2)$$

donde u es la distancia Euclídea entre el vector de entrada **x** y el vector centro μ, es decir, $u=\|\mathbf{x}-\mu\|$.

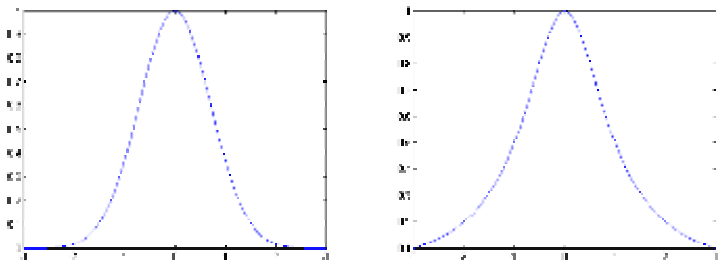
Otras funciones radialmente simétricas utilizadas son la *cuádrica inversa de Hardy*,

$$\varphi_2(u) = (\sigma^2 + u^2)^{\beta}, \quad \beta < 0$$

y la *hiperesférica*

$$\varphi_3(u) = \begin{cases} 1 & \text{si} \quad u \le \sigma \\ 0 & \text{si} \quad u > \sigma \end{cases} .$$

En la figura 15(a) representamos la función Gaussiana y en la 15(b) la función cuadrática de Hardy.



(a) (b)

Figura 15. (a) Función Gaussiana. (b) Función cuádrica inversa de Hardy.

Una red de neuronas artificiales constituida por una capa oculta de unidades de proceso (nodos) conectadas a las entradas (sensores) y a una neurona de salida, con transmisión directa (feedforward), donde las funciones de transferencia de los nodos de la capa oculta son funciones simétricas radialmente, se dice que es una **red con funciones de base radial** (red RBF). En la figura 16 se muestra la arquitectura de una red RBF con tres nodos en la capa oculta.

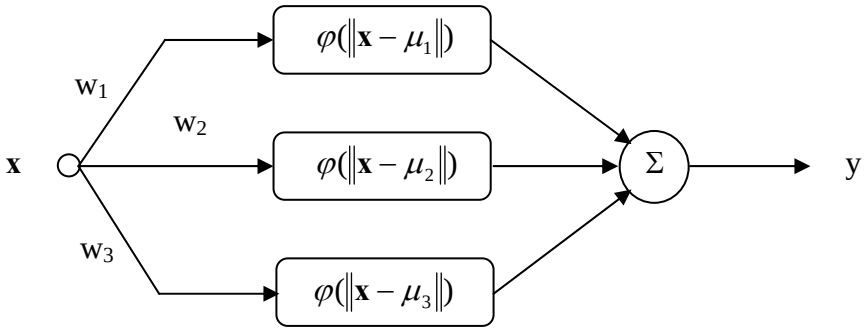


Figura 16. Una red RBF con tres nodos en la capa oculta.

Por lo tanto, la salida de una red RBF viene dada por la expresión

$$y = \sum_{i=1}^{M} w_i \varphi(\|\mathbf{x} - \mu_i\|) \qquad (16)$$

donde φ es una función simétrica radialmente.

También se suele representar una red con funciones de base radial mediante una capa intermedia de unidades de base radial y una capa de unidades de salida con función de transferencia la función identidad, como se muestra en la figura 17. Los pesos de la unidad de base radial *i* vienen dados por μ_i y su salida por la expresión $\varphi(\|\mathbf{x}-\mu_i\|)$ que es la función de base radial aplicada a la distancia que hay entre el vector de entrada **x** y el vector de parámetros μ_i. Los pesos de la unidad o unidades de salida viene dados por los valores w_i.

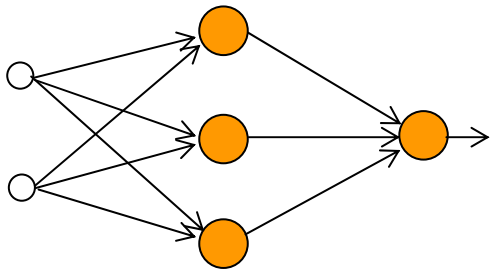


Figura 17. Una red RBF con tres nodos FBR en la capa oculta y una unidad de salida.

Ahora se trata de determinar el conjunto de parámetros de la red mediante un *proceso de aprendizaje* utilizando un conjunto de patrones de entrenamiento. Para ello disponemos del conjunto $\{\mathbf{x}_k,\ k=1,2,...,p\}$ de p patrones de entrenamiento. Conocemos además la salida deseada correspondiente a cada patrón. Sea z_k la salida deseada correspondiente al patrón $\mathbf{x}_k$, $k=1,2,...,p$. Nuestro objetivo es aprender los valores de los parámetros de la red que hacen mínimo el error cuadrático total correspondiente a los patrones de entrenamiento. Es decir, deseamos minimizar la expresión:

$$E=\sum_{k=1}^{p}E_k=\sum_{k=1}^{p}(z_k-y_k)^2$$

Si para ello utilizamos el método del gradiente para determinar los pesos que tiene la neurona de salida obtenemos:

$$\frac{\partial E_k}{\partial w_i}=-2(z_k-y_k)\varphi(\|\mathbf{x}-\mu_i\|)$$

y la modificación del peso w_i viene dada por la dirección opuesta al gradiente, es decir, según la siguiente *regla de aprendizaje*:

$$\Delta w_i=\eta_i(z_k-y_k)\varphi(\|\mathbf{x}-\mu_i\|) \qquad (17)$$

Análogamente, para la obtención de la *regla de aprendizaje* de los centros de cada nodo, derivamos la función de error, obteniendo

$$\frac{\partial E_k}{\partial \mu_{ij}}=2(z_k-y_k)(-w_i\frac{\partial\varphi(\|\mathbf{x}-\mu_i\|)}{\partial\|\mathbf{x}-\mu_i\|^2}\frac{\partial\|\mathbf{x}-\mu_i\|^2}{\partial\mu_{ij}})$$

Si utilizamos la función *g* definida por la expresión

$$g(u^2)=\varphi(u),$$

entonces como

$$\frac{\partial \varphi\left(\left\|\mathbf{x}-\mu_i\right\|\right)}{\partial\left\|\mathbf{x}-\mu_i\right\|^2}=\frac{\partial g\left(\left\|\mathbf{x}-\mu_i\right\|^2\right)}{\partial\left\|\mathbf{x}-\mu_i\right\|^2}=g'\left(\left\|\mathbf{x}-\mu_i\right\|^2\right)$$

resulta que

$$\frac{\partial E_k}{\partial \mu_{ij}}=4w_{ij}(z_k-y_k)g'\left(\left\|\mathbf{x}-\mu_i\right\|^2\right)(x_{kj}-\mu_{ij})$$

y, por lo tanto, la *regla de aprendizaje* es:

$$\Delta\mu_{ij}=-\eta w_{ij}(z_k-y_k)g'\left(\left\|\mathbf{x}-\mu_i\right\|^2\right)(x_{kj}-\mu_{ij}) \tag{18}$$

Si utilizamos la función *Gaussiana*, las *reglas de aprendizaje* vienen dadas por las expresiones siguientes:

$$\Delta w_i=\eta_i(z_k-y_k)\exp(-\left\|\mathbf{x}-\mu_i\right\|^2/\sigma^2) \tag{19}$$

$$\Delta\mu_{ij}=-\eta w_{ij}(z_k-y_k)\exp\left(-\left\|\mathbf{x}-\mu_i\right\|^2/\sigma^2\right)(x_{kj}-\mu_{ij}) \tag{20}$$

Una regla similar se obtiene también para los parámetros de dispersión σ_i.

Sin embargo, estos algoritmos de aprendizaje, para los centros y las dispersiones de los nodos, conllevan una cantidad considerable de cómputo. Por ello, se han propuesto en la literatura otras técnicas más rápidas. Así, mediante procedimientos de agrupación de datos se pueden estimar los centros (μ_i) de los nodos y sus dispersiones (σ_i).

5.11 Construcción de una Red neuronal de tamaño óptimo: Procedimientos de poda

Las redes neuronales más pequeñas son preferibles a las más grandes que realizan una misma tarea por varias razones: tienen un menor número de parámetros, el entrenamiento es más rápido y suelen tener una mayor capacidad de generalización al utilizar nuevos patrones. Sin embargo, actualmente no hay un procedimiento sencillo ni general para determinar el tamaño óptimo de una red neuronal. Tres maneras de intentar conseguir un diseño satisfactorio son las siguientes:

- Partir de una red neuronal de gran tamaño y *podarla* eliminándole unidades de proceso y conexiones hasta conseguir un tamaño satisfactorio.
- Comenzar con una red neuronal muy pequeña e ir incrementando su tamaño añadiendo unidades de proceso, conexiones o capas hasta conseguir un tamaño satisfactorio.
- Partir de una red de tamaño suficiente y podar las conexiones y unidades de proceso que se consideren poco relevantes. A continuación se añaden nuevas unidades de proceso con pesos aleatorios y se vuelve a entrenar la red. Este proceso se continúa hasta que se consigue un tamaño aceptable y un comportamiento satisfactorio.

A continuación vamos a estudiar algunos procedimientos de poda. Para diseñar una red neuronal que sea apropiada para resolver un problema de predicción o de clasificación se suele partir de una red grande y después seguir un proceso de poda (eliminación) de unidades de proceso y conexiones hasta conseguir una red de tamaño más reducido y con comportamiento satisfactorio (una mayor capacidad de generalización).

Algunos procedimientos para realizar la poda de una red neuronal son:

- Eliminar de la red aquellas conexiones cuyos pesos sinápticos sena de pequeña magnitud
- Eliminar aquellas conexiones cuya existencia no afecte significativamente a las salidas de la red. Para realizar este tipo de poda basta con ir comparando las salidas de la red cuando un peso sináptico es reducido a cero.
- Eliminar aquellos sensores de entrada que producen cambios insignificantes en la salida de la red. Este supone reducir la dimensionalidad de los patrones de entrada al detectar aquellas componentes de entrada que son innecesarias.
- El método OBD (*optimal brain damage*), de lesión cerebral óptima, propuesto por Lecun, Denker y Solla (1990), trata de identificar aquellos pesos sinápticos que pueden ser podados examinando las derivadas segundas de la función de error contenidas en la matriz Hessiana. La variación que produce en el error cuadrático medio una pequeña perturbación Δw_{ij} en el peso w_{ij} se aproxima por

$$\Delta E = \frac{\partial E}{\partial w_{ij}} \Delta w_{ij} + \frac{1}{2} (\frac{\partial^2 E}{\partial w_{ij}^2}) (\Delta w_{ij})^2$$

que considera los dos primeros términos del desarrollo de E en series de Taylor, mientras que el algoritmo de retropropagación sólo considera el primer término. Además, cuando la red finaliza el entrenamiento en un mínimo local de E, entonces $\partial E/\partial w_{ij} \approx 0$, y así

$$\Delta E \approx \frac{1}{2} (\frac{\partial^2 E}{\partial w_{ij}^2}) (\Delta w_{ij})^2 .$$

Como la poda de una conexión supone pasar su peso sináptico del valor w_{ij} al valor 0, es decir, ocurre cuando $\Delta w_{ij} = -w_{ij}$, entonces la condición para realizar la poda de dicha conexión es que el cambio en el error resultantes sea insignificante, es decir, que la cantidad

$$\Delta E \approx \frac{1}{2} (\frac{\partial^2 E}{\partial w_{ij}^2}) (w_{ij})^2$$

sea suficientemente pequeña.