5 Redes Neuronales Multicapa

5.1 La ADALINA

Otro modelo clásico de redes neuronal es la ADALINA (acrónimo de **ADA**ptive **L**inear **NE**uron) o neurona con adaptación lineal que fue introducida por Widrow en 1959. Esta neurona es similar al Perceptrón simple pero utiliza como función de transferencia la función identidad en lugar de la función signo. La salida de la ADALINE es simplemente una función lineal de las entradas (ponderadas con los pesos sinápticos):

$$y = \sum_{j=1}^{N} w_j x_j - \theta$$

Obsérvese que ahora la salida de la red es continua en lugar de binaria.

Si consideramos una entrada adicional con valor $x_{N+1}= -1$ cuyo peso sináptico $w_{N+1}=\theta$, entonces podemos escribir simplemente

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

Así, de forma general podemos tratar el valor umbral θ como un peso sináptico adicional con entrada igual a −1.

Con la ADALINE se pretende implementar la correspondencia entre las entradas y las salidas de un sistema utilizando un conjunto finito de relaciones entre entradas y salidas. Supongamos que disponemos de p patrones de entrada $\{\mathbf{x^1}, \mathbf{x^2}, ..., \mathbf{x^p}\}$ y sus correspondientes salidas deseadas $\{z^1, z^2, ..., z^p\}$. Se trata de determinar los pesos sinápticos que consiguen que las salidas de la red sean lo más parecidas a las salidas deseadas para el conjunto dado de patrones de entrenamiento. Es decir, se trata de determinar los pesos sinápticos de manera que se minimice la función de error cuadrático siguiente:

$$E = \frac{1}{2}\sum_{k=1}^{p}\left(z^k - y(k)\right)^2 = \frac{1}{2}\sum_{k=1}^{p}\left(z^k - \sum_{j=1}^{N+1} w_j(k) x_j^k\right)^2$$

Para ello vamos a seguir el método de descenso por el gradientes, es decir, por la dirección opuesta a la dirección del gradiente. Si en la iteración k hemos introducido el patrón de entrenamiento $\mathbf{x}^k$, cuya salida deseada es z^k, y los pesos sinápticos son $w_j(k)$, j=1,2,…,N, entonces la modificación de los mismos motivada por dicho patrón es:

$$w_j(k+1) = w_j(k) + \Delta w_j(k),$$

donde

$$\Delta w_j(k) = -\eta \frac{\partial E}{\partial w_j(k)}$$

$$= \eta\left[z^k - y(k)\right]x_j^k, \quad j=1,2,\ldots,N \tag{1}$$

El parámetro η controla la longitud del paso que vamos a dar en la dirección opuesta del gradiente. Conforme mayor sea η mayor será la cantidad por la que se modificarán los pesos sinápticos. Dicho parámetro debe ser un valor pequeño para evitar dar pasos demasiado largos, es decir, que nos lleven a soluciones peores que la que teníamos, puesto que el método del gradiente solamente garantiza el decrecimiento de la función de error si nos desplazamos en la direcciones opuesta del gradiente pero en un entorno suficientemente pequeño. A η lo llamaremos parámetro de aprendizaje o **tasa de aprendizaje**.

En el proceso de entrenamiento hemos introducido un patrón en cada iteración, por ello diremos que hemos realizado un **aprendizaje en línea**. También podemos introducir los *p* patrones directamente y comparar las salidas de la red con las salidas deseadas, pasando entonces a actualizar los pesos sinápticos, en cuyo caso diremos que el **aprendizaje** es **por lotes**. La modificación de los pesos sinápticos se hace tomando como función de error el error medio, es decir,

$$E = \frac{1}{2p}\sum_{k=1}^{p}\left(z^k - y(k)\right)^2 = \frac{1}{2p}\sum_{k=1}^{p}\left(z^k - \sum_{j=1}^{N+1} w_j x_j^k\right)^2$$

y así la regla de aprendizaje es

$$\begin{aligned}\Delta w_j &= -\eta\frac{\partial E}{\partial w_j} \\ &= \eta\frac{1}{p}\sum_{k=1}^{p}\left[z^k - y(k)\right]x_j^k\end{aligned} \tag{2}$$

Obsérvese que aquí el peso sináptico no depende del patrón introducido, es el mismo para los *p* patrones puesto que se actualiza conjuntamente según el error medio para los *p* patrones.

5. 2 Neuronas con salida continua: Regla de aprendizaje de Widrow-Hoff

Vamos a considerar unidades de proceso con salidas continuas. Una unidad de proceso continua es aquella cuya salida viene dada por la siguiente expresión:

$$y = g\left(\sum_{j=1}^{N} w_j x_j\right)$$

donde $\mathbf{x} = (x_1, x_2, \ldots, x_N)' \in \Re^N$ en la entrada de la unidad, $\mathbf{w} = (w_1, w_2, \ldots, w_N)' \in \Re^N$ es el vector de pesos sinápticos y *g* es la función de transferencia. La función de transferencia es una función no decreciente y vamos a elegir como posibles funciones de transferencia a alguna de las siguientes funciones:

a) La función logística

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

cuya representación gráfica se muestra en la figura 1. Es una función de aplastamiento puesto que pasa los valores del potencial sináptico, que son del intervalo (–∞, ∞), al intervalo [0, 1].

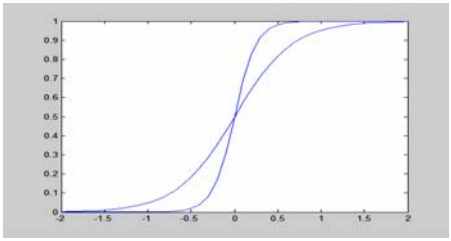


Figura 1. Funciones logísticas.

El parámetro de ganancia β 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.

b) La función tangente hiperbólica,

$$g(x) = \tanh(\beta x) = \frac{e^{\beta x} - e^{-\beta x}}{e^{\beta x} + e^{-\beta x}}$$

cuya representación gráfica se muestra en la figura 2. Es una función de aplastamiento puesto que pasa los valores del potencial sináptico al intervalo [-1, 1].

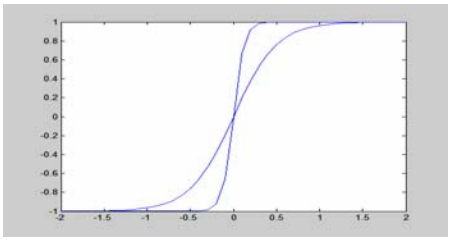


Figura 2. Funciones tangentes hiperbólicas.

También se utiliza dicha función como función de transferencia puesto que su derivada es muy simple, $g'(x) = \beta\left[1 - g(x)^2\right]$, es decir, es también una función de la propia función. Asimismo, cuanto mayor es el parámetro de ganancia β mayor es la pendiente de la curva y más se asemeja a la función signo.

c) La función identidad, $g(x)=x$.

A la cantidad $h = \sum_{j=1}^{N} w_j x_j$ se le llama potencial sináptico. En la figura 3 representamos gráficamente una unidad de proceso continua.

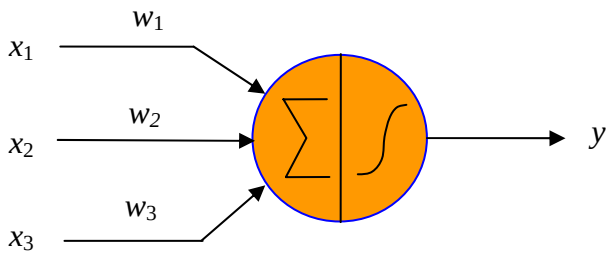


Figura 3. Neurona analógica.

Con esta unidad de proceso continua se pretende implementar la correspondencia entre las entradas y las salidas de un sistema utilizando un conjunto finito de relaciones entre entradas y salidas. Supongamos que disponemos de p patrones de entrada $\{\mathbf{x^1}, \mathbf{x^2}, ..., \mathbf{x^p}\}$ y sus correspondientes salidas deseadas $\{z^1, z^2, ..., z^p\}$. Se trata de determinar los pesos sinápticos que consiguen que las salidas de la red sean lo más parecidas a las salidas deseadas para el conjunto dado de los patrones de entrenamiento. Es decir, se trata de determinar los pesos sinápticos de manera que se minimice la función de error cuadrático siguiente:

$$E = \frac{1}{2}\sum_{k=1}^{p}\left(z^k - y(k)\right)^2 = \frac{1}{2}\sum_{k=1}^{p}\left(z^k - g(\sum_{j=1}^{N+1} w_j(k)x_j^k)\right)^2$$

El método de descenso por la dirección opuesta del gradiente nos conduce a la siguiente regla de aprendizaje, conocida con el nombre de regla de Widrow-Hoff, regla de mínimos cuadrados medios o regla LMS (*Least Mean Squares*):

$$\Delta w_j(k) = -\eta \frac{\partial E}{\partial w_j(k)}$$

$$= \eta\left[z^k - y(k)\right]g'(h)x_j^k, \quad j=1,2,\ldots,N$$

donde h es el potencial sináptico.

En el caso de aprendizaje por lotes, tomaremos como función de error cuadrático medio

$$E = \frac{1}{2p}\sum_{k=1}^{p}\left(z^k - y(k)\right)^2 = \frac{1}{2p}\sum_{k=1}^{p}\left(z^k - g(\sum_{j=1}^{N+1} w_j x_j^k)\right)^2$$

y así,

$$\Delta w_j = -\eta \frac{\partial E}{\partial w_j}$$

$$= \eta \frac{1}{p}\sum_{k=1}^{p}\left[z^k - y(k)\right]g'(h)x_j^k \qquad (3)$$

5.3 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 muchas unidades de entrada y muchos elementos Adalides en la primera capa, y con varios dispositivos lógicos (AND, OR,…) en la segunda capa.

Sin embargo, como hemos visto, 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 la función lógica XOR. 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 este 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 capaz se puede implementar la función lógica XOR.

Al tener estas redes una topología mas complicada, también se complicó 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. 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.

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 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 4.
Con esta red se pretender establecer una correspondencia entre un conjunto de entrada y un conjunto de salidas deseadas, de manera que

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

Para ello se dispone de un conjunto de con *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:

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

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 dependerá de la dificultad de la correspondencia a implementar.

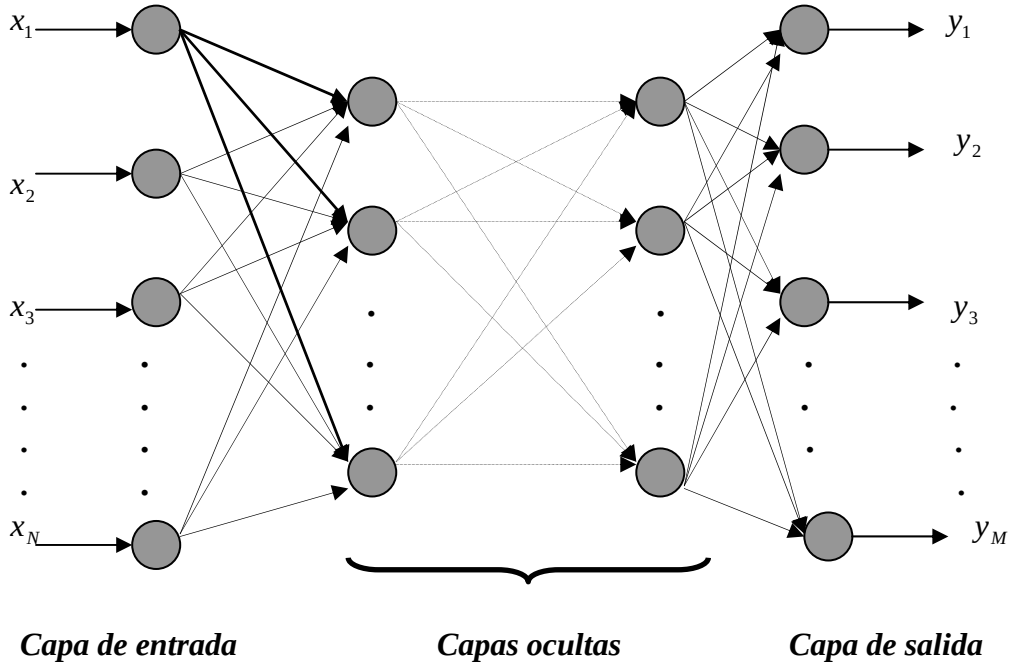


Figura 4. Topología de un Perceptrón Multicapa.

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:

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

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; 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 comente 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:

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

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

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

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

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

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

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 avalúan 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:

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

Así,

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

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

5.4 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. Con frecuencia se produce cierto zigzagueo en el descenso por el gradiente que hace que el algoritmo sea lento (poco eficiente). Para evitar 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 el paso anterior, como se expresa en la siguiente ecuación:

$$\Delta w_{ij}(k) = \alpha\,\Delta w_{ij}(k-1) + \eta\delta_i^2(k)s_j(k) \tag{9}$$

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.

5.5 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:

$$p(C_i/\mathbf{x}) = \frac{p(\mathbf{x}/C_i)p(C_i)}{\sum_{j=1}^{2} p(\mathbf{x}/C_j)p(C_j)} = \frac{p(\mathbf{x}/C_i)p(C_i)}{p(\mathbf{x})} \quad , \qquad i=1,2. \tag{10}$$

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 $\mathbf{x}$ 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 5) con función de transferencia logística (figura 6). Es decir,

$$F(\mathbf{x},\mathbf{t},\mathbf{w}) = \sigma\left(\sum_{i=1}^{M} w_i\, g\left(\sum_{j=1}^{N} t_{ij}\, x_j\right)\right) \tag{11}$$

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

$$g(x) \equiv \frac{1}{1+\exp(-2\beta x)} \tag{12}$$

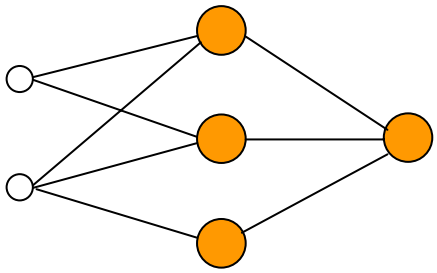


Figura 5. Perceptrón con una única unidad de salida.

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.

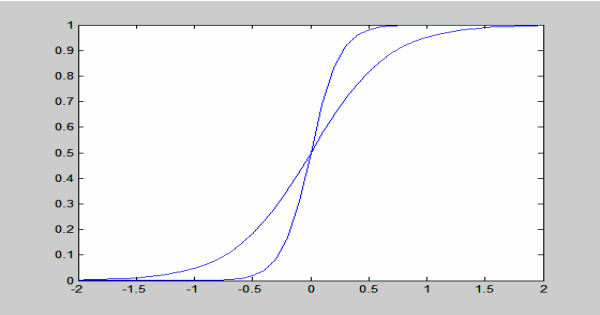


Figura 6. Funciones logísticas.

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*

$$E_s(\mathbf{t},\mathbf{w}) = \frac{1}{n}\left[\sum_{\mathbf{x}\in\chi_1}(F(\mathbf{x},\mathbf{t},\mathbf{w})-1)^2 + \sum_{\mathbf{x}\in\chi_2}F(\mathbf{x},\mathbf{t},\mathbf{w})^2\right] \tag{13}$$

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*,

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

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

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

se tiene 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,

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

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

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

$$=\int_{R^N}\left[(F(\mathbf{x},\mathbf{t},\mathbf{w})-p(C_1/\mathbf{x})\right]^2\,p(\mathbf{x})d\mathbf{x}+\int_{R^N}p(C_1/\mathbf{x})(1-p(C_1/\mathbf{x}))\,p(\mathbf{x})d\mathbf{x}. \qquad (14)$$

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 7) 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:

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

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

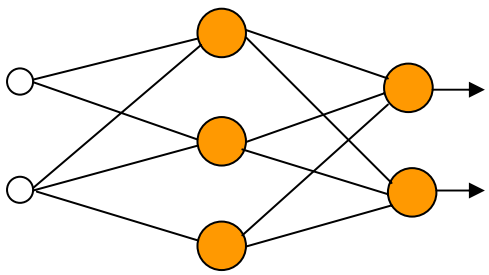


Figura 7. Perceptrón con dos unidades de salida.

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.

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:

$$F_i(\mathbf{x},\mathbf{t},\mathbf{w}) = \frac{e^{\sum_{j=1}^{L} w_{ij} s_j}}{\sum_{m=1}^{C} e^{\sum_{j=1}^{L} w_{ij} s_j}} \qquad (15)$$

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 Aplicación a los 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 basándose en esas 18 características si una persona va a tener recaídas en un periodo posterior de la intervención quirúrgica de 6 años.

En la figura 8 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 4.5. 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.

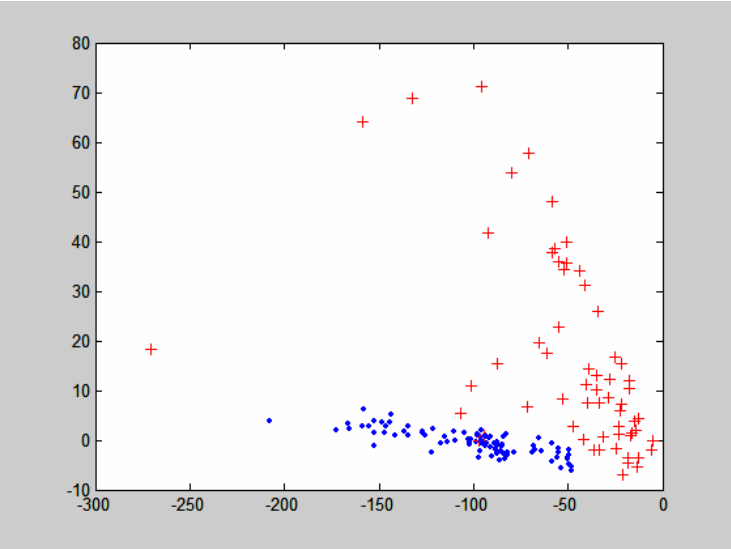


Figura 8. Representación en sus dos primeras componentes principales.

5.7 Aplicación a los 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. Las 200 primeras se han utilizado para el entrenamiento de la red constituida por 9 sensores de entrada, 15 unidades de proceso en la capa oculta (tomadas en base a la experimentación) y una unidad de salida. Los pesos sinápticos han sido determinados mediante el algoritmo de Levemberg-Marquard. 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 9 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.

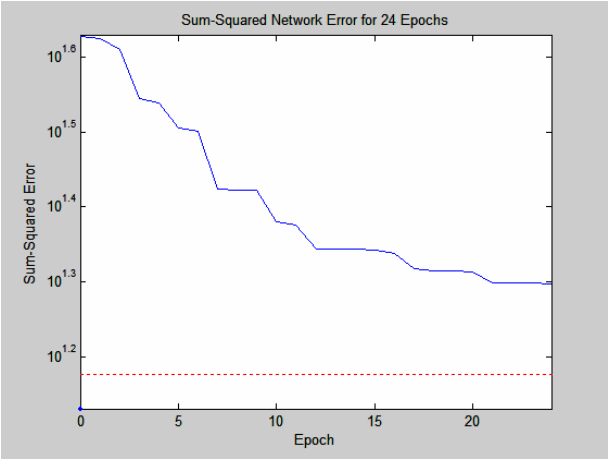


Figura 9. Evolución del error cuadrático.

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.

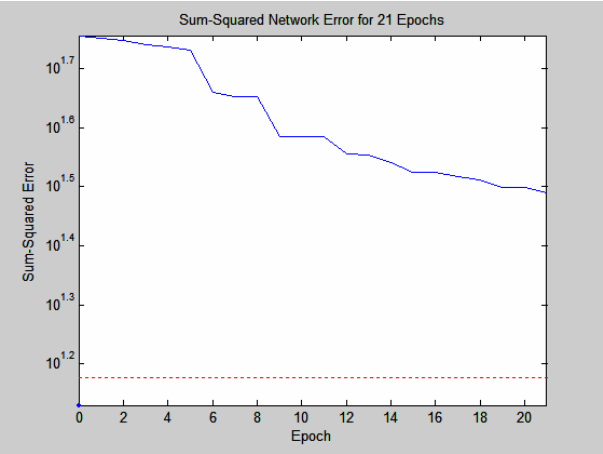


Figura 10. Evolución del error cuadrático.

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.

5.8 Aplicación a los 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 11 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

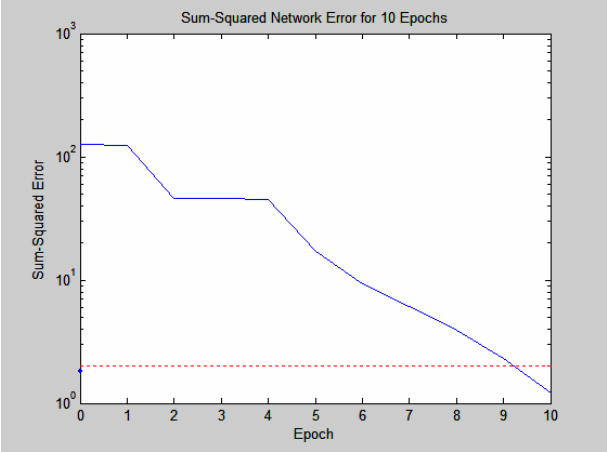


Figura 11. Evolución del error cuadrático.

5.9 Aplicación a los 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 12 se muestra la evolución del error cuadrático medio durante el proceso de aprendizaje y se puede observar como 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.

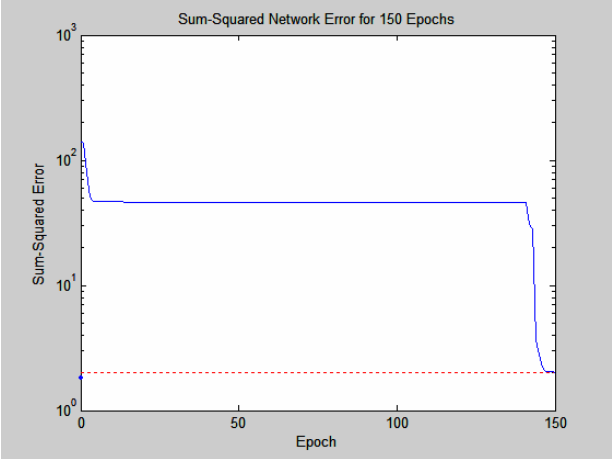


Figura 12. Evolución del error cuadrático.

5.10 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 13, 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.

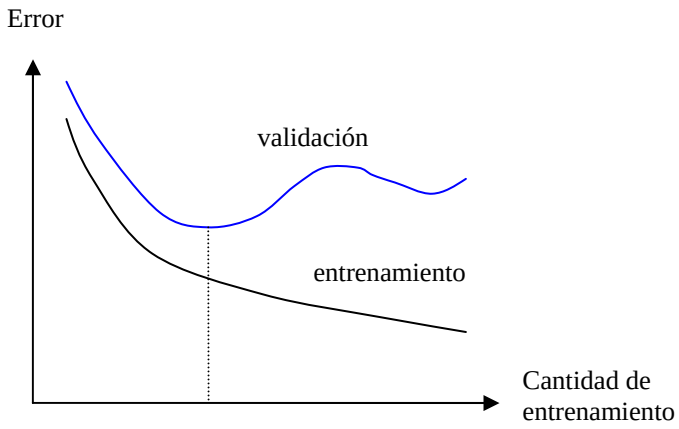


Figura 13. Errores de entrenamiento y validación.

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.

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

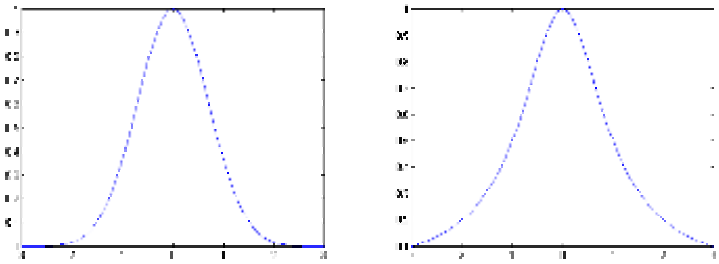
Otras funciones radialmente simétricas utilizadas son la *cuádrica inversa de Hardy*,

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

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 14(a) representamos la función Gaussiana y en la 14(b) la función cuadrática de Hardy.



(a) (b)

Figura 14. (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 15 se muestra la arquitectura de una red RBF con tres nodos en la capa oculta.

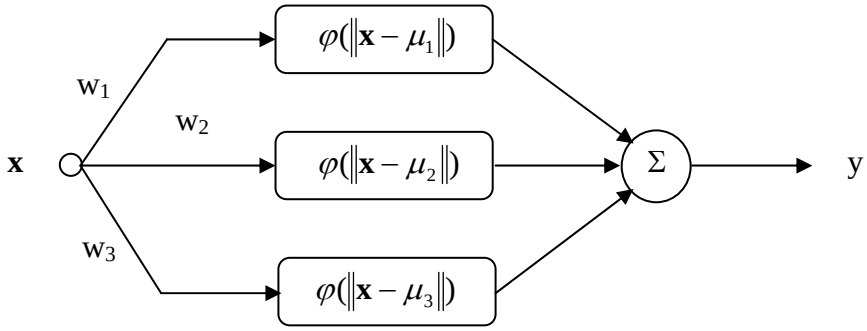


Figura 15. Una red RBF con tres nodos en la capa oculta.

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

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

donde φ es una función simétrica radialmente.
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*:

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

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

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

resulta que

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

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

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

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

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

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

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).

EJEMPLO:

Diseña una red RBF que aísle las esquinas de un cuadrado. Es decir, que clasifique los patrones (0,0), (2,0), (0,2) y (2,2) en la clase C_1 y los patrones (1,0), (1,1), (1,2), (0,1) y (2,1) en la clase C_2 (ver la figura 16).

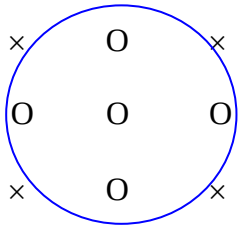


Figura 16. Patrones de entrenamiento para aislar las esquinas del cuadrado.

Solución

Se puede utilizar un único nodo en la capa oculta, con centro μ=(1,1), y un nodo en la capa de salida que implemente la función escalón, con valor 0 si la entrada es menor o igual a la entrada correspondiente a las esquinas y valor 1 en otro caso.

También se pueden utilizar cuatro nodos en la capa oculta cuyos centros tienen las coordenadas de las esquinas y una dispersión muy pequeña. El nodo de salida se activa si y sólo si uno de los nodos de la capa está activado significativamente.