Capítulo 3

Perceptron multicapa

3.1. Introducción

En este capítulo se estudiará una de las clases de redes de neuronas, conocida como Perceptron multicapa o red multicapa con conexiones hacia adelante. El Perceptron multicapa es una generalización del Perceptron simple y surgió como consecuencia de las limitaciones de dicha arquitectura en lo referente al problema de la separabilidad no lineal. Minsky y Papert [4] mostraron en 1969 que la combinación de varios Perceptrones simples -inclusión de neuronas ocultas- podía resultar una solución adecuada para tratar ciertos problemas no lineales. Sin embargo, los autores no presentaron una solución al problema de cómo adaptar los pesos de la capa de entrada a la capa oculta, pues la regla de aprendizaje del Perceptron simple no puede aplicarse en este escenario. No obstante, la idea de combinar varios Perceptrones sirvió de base para estudios posteriores realizados por Rummelhart, Hinton y Wilians en 1986 [5]. Estos autores presentaron una manera de retropropagación de los errores medidos en la salida de la red hacia las neuronas ocultas, dando lugar a la llamada regla delta generalizada.

Diferentes autores han demostrado independientemente que el Perceptron multicapa es un aproximador universal, en el sentido de que cualquier función contínua en un espacio R^n puede aproximarse con un Perceptron multicapa, con al menos una capa oculta de neuronas. Tal como se explicó en el capítulo anterior, este resultado sitúa al Perceptron multicapa como un modelo matemático útil a la hora de aproximar o interpolar relaciones no lineales entre datos de entrada y salida.

Dentro del marco de las redes de neuronas, el Perceptron multicapa es en la actualidad una de las arquitecturas más utilizadas en la resolución de problemas. Esto es debido, fundamentalmente, a su capacidad como aproximador universal, así como a su fácil uso y aplicabilidad.

Por otra parte, esto no implica que sea una de las redes más potentes y con mejores resultados en sus diferentes áreas de aplicación. De hecho, el Perceptron multicapa posee una serie de limitaciones, como el largo proceso de aprendizaje para problemas complejos dependientes de un gran número de variables; la dificultad para realizar un análisis teórico de la red debido a la presencia de componentes no lineales y a la alta conectividad.

Por otra parte, es necesario señalar que el proceso de aprendizaje de la red busca en un espacio apmplio de funciones, una posible función que relacione las variables de entrada y salida al problema, lo cual puede complicar su aprendizaje y reducir su efectividad en determinadas aplicaciones [6].

3.2. Arquitectura

La arquitectura de Perceptron multicapa se caracteriza porque tiene sus neuronas agrupadas en capas de diferentes niveles. Cada una de las capas está formada por un conjunto de neuronas y se distinguen tres tipos de capas diferentes: la capa de entrada, las capas ocultas y la capa de salida.

Las neuronas de la capa de entrada no actúan como neuronas propiamente dichas, sino que se encargan únicamente de recibir las señales o patrones del exterior y propagar dichas señales a todas las neuronas de la siguiente capa. La última capa actúa como salida de la red, proporcionando al exterior la respuesta de la red para cada uno de los patrones de entrada. Las neuronas de las capas ocultas realizan un procesamiento no lineal de los patrones recibidos.

Como se observa en la figura 2.1, las conexiones del Perceptron multicapa siempre están dirigidas hacia adelante, es decir, las neuronas de una capa se conectan con las neuronas de la siguiente capa, de ahí que reciban también el nombre de redes alimentadas hacia adelante o redes "feedforward. Las conexiones entre las neuronas de la red llevan también asociado un umbral, que el caso del Perceptron multicapa suele tratarse como una conexión más a la neurona, cuya entrada es constante e igual a 1.

Generalmente, todas las neuronas de una capa están conectadas a todas las neuronas de la siguiente capa. Se dice entonces que existe conectividad total o que la red está totalmente conectada.

Aunque en la mayor parte de los casos la arquitectura del Perceptron multicapa está asociada al esquema de la figura 2.1, es posible también englobar dentro de este tipo de redes a arquitecturas con las siguientes caracterícticas:

- Redes con conexiones de todas o ciertas neuronas de una determinada capa a neuronas de capas posteriores, aunque no inmediatamente posteriores.

- Redes en las que ciertas neuronas de ciertas capas no están conectadas a neuronas de la siguiente capa, es decir, el peso de la conexión es constante e igual a cero

Cuando se aborda un problema con el Perceptron multicapa, en la mayoría de los casos se parte de una arquitectura totalmente conectada, es decir, todas las neuronas de una capa están conectadas a todas las neuronas de la siguiente capa. No es posible demostrar que si se utilizan arquitecturas en las que se eliminan o se añaden conexiones de una capa a capas no inmediatamente posteriores, se puedan obtener mejores resultados. Sin embargo, en ocasiones, y debido fundamentalmente a la naturaleza del problema, se pueden encontrar redes multicapa con estas características en sus conexiones [7].

3.2.1. Propagación de los patrones de entrada

El Perceptrón multicapa define una relación entre las variables de entrada y las variables de salida de la red. Esta relación se obtiene propagando hacia adelante los valores de las variables de entrada. Para ello, cada neurona de la red procesa la información recibida por sus entradas y produce una respuesta o activación que se propaga, a través de las conexiones correspondientes, hacia las neuronas de la siguiente capa. A continuación, se muestran las expresiones para calcular las activaciones de las neuronas de la red.

Sea un Perceptron multicapa con C capas -$C-2$ capas ocultas- y n_c neuronas en la capa c, para c = 1,2, ..., C. Sea $W^c = (w_{ij}^c)$ la matriz de pesos donde w_{ij}^c representa el peso de la conexión de la neurona i de la capa c para c = 2, ..., C. Denotaremos a_i^c a la activación de la neurona i de la capa c. Estas activaciones se calculan del siguiente modo:

- Activación de las neuronas de la capa de entrada (a_i^1). Las neuronas de la capa de entrada se encargan de transmitir hacia la red las señales recibidas desde el exterior. Por tanto:
$$a_i^1 = x_i \; para \; i = 1, 2, ..., n_1 \tag{3.1}$$
donde $X = (x_1, x_2, ..., x_{n1})$ representa el vector o patrón de entrada a la red.

- Activación de las neuronas de la capa oculta c (a_i^c). Las neurones ocultas de la red procesan la información recibida aplicando la función de activación f a la suma de los productos de las activaciones que recibe por sus correspondientes pesos, es decir:
$$a_i^c = f\left(\sum_{j=1}^{n_{c-1}} w_{ji}^{c-1} a_j^{c-1} + u_i^c\right) \; para \; i = 1, 2, ..., n_c \; y \; c = 2, 3, ..., C-1 \tag{3.2}$$
donde a_j^{c-1} son las activaciones de las neuronas de la capa $c-1$.

- Activación de las neuronas de la capa de salida (a_i^C). Al igual que en el caso anterior, la activación de estas neuronas viene dada por la función de activación f aplicada a la suma de los productos de las entradas que recibe por sus correspondientes pesos:
$$y_i = a_i^C = f\left(\sum_{j=1}^{n_{C-1}} w_{ji}^{C-1} a_j^{C-1} + u_i^C\right) \; para \; i = 1, 2, ..., n_c \tag{3.3}$$
donde $Y = (y1, y2, ..., y_{nC})$ es el vector de salida de la red.

La función f es la llamada *función de activación*. Para el Perceptron multicapa, las funciones de activación más utilizadas son la función sigmoidal y la función tangente

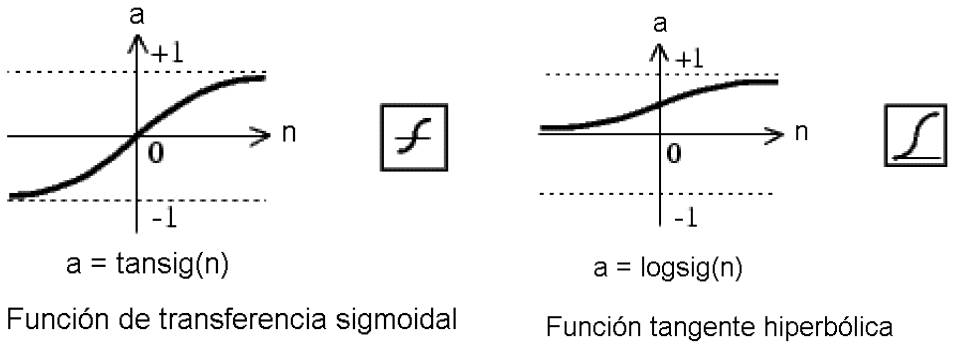

Figura 3.1: Funciones de activación sigmoidal y tangente hiperbólica

hiperbólica. Dichas funciones poseen como imagen un intervalo contínuo de valores dentro de los intervalos $[0, 1]$ y $[-1, 1]$, respectivamente, y vienen dadas por las siguientes ecuaciones:

- Función sigmoidal:

$$f_{sigm}(x) = \frac{1}{1 + e^{-x}} \tag{3.4}$$

- Función tangente hiperbólica:

$$f_{thip}(x) = \frac{1 - e^{-x}}{1 + e^{-x}} \tag{3.5}$$

Ambas son funciones crecientes con dos niveles de saturación: el máximo, que proporciona salida 1, y el mínimo, salida 0 para la función sigmoidal y salida -1, para la tangente hiperbólica, como se observa en la figura 3.1.

En algunas ocasiones, la función de activación en el Perceptron multicapa es común a todas las neuronas de la red y es elegida por el diseñador, elección que se realiza únicamente basándose en los valores de activación que se desee que alcancen las neuronas. Ambas funciones están relacionadas mediante la expresión $f_{thip}(x) = 2f_{sigm}(x) - 1$, por lo que la utilización de una u otra se elige únicamente en función del recorrido que interese.

En otras ocasiones, y dependiendo de la naturaleza del problema, las neuronas de salida se distinguen del resto de neuronas de la red, utilizando otro tipo de función de activación. En este caso, las más usadas son la función identidad y la función escalón. De las ecuaciones 3.1, 3.2 y 3.3, se observa que el perceptrón multicapa define, a través de sus conexiones y neuronas, una función continua no lineal del espacio R^{n_1} -espacio de los patrones de entrada- al espacio R^{n_C} -espacio de los patrones de salida-. Se puede escribir, por tanto, que:

$$Y = F(X, W) \tag{3.6}$$

donde Y es el vector formado por las salidas de la red, X es el vector de entrada a la red, W es el conjunto de todos los parámetros de la red -pesos y umbrales- y F es una función continua no lineal dada por las ecuaciones 3.1, 3.2 y 3.3.

3.2.2. Consideraciones de diseño

Cuando se aborda un problema utilizando el Perceptron multicapa, uno de los primeros pasos a realizar es el diseño de la arquitectura de la red. Este diseño implica la determinación de la función de activación a emplear, el número de neuronas y el número de capas de la red.

Como se ha comentado anteriormente, la elección de la función de activación se suele hacer basándose en el recorrido deseado, y el hecho de elegir una u otra, generalmente, no influye en la capacidad de la red para resolver el problema [7].

En lo que respecta al número de neuronas y capas, algunos de estros parámetros vienen dados por el problema y otros deben ser elegidos por el diseñador. Así, por ejemplo, tanto el número de neuronas en la capa de entrada, como el número de neuronas en la capa de salida, vienen dados por las variables que definen el problema. En algunas aplicaciones prácticas, no hay lugar a duda sobre el número de entradas y salidas. Sin embargo, existen problemas los que el número de variables de entrada relevantes para el problema no se conoce con exactitud. En estos casos, se disponen de un gran número de variables, algunas de las cuales podrían no aportar información relevante a la red, y su utilización podría complicar el aprendizaje, pues implicaría arquitecturas de gran tamaño y con alta conectividad. En estas situaciones, es conveniente realizar un análisis previo de las variables de entrada más relevantes al problema y descartar aquellas que no aportan información a la red. Este análisis puede llegar a ser una tarea complicada y requerir técnicas avanzadas, como técnicas basadas en análisis de correlación, análisis de componentes principales, análisis de importancia relativa, análisis de sensibilidad o técnicas basadas en algoritmos genéticos, entre otras.

El número de capas ocultas y el número de neuronas en estas capas deben ser elegidos por el diseñador. No existe un método o regla que determine el número óptimo de neuronas ocultas para resolver un problema dado. En la mayor parte de las aplicaciones prácticas, estos parámetros se determinan por prueba y error. Partiendo de una arquitectura ya entrenada, se realizan cambios aumentando o disminuyendo el número de neuronas ocultas y el número de capas hasta conseguir una arquitectura adecuada para el problema a resolver, que pudiera no ser la óptima, pero que proporciona una solución.

Si bien el número de neuronas ocultas puede influir en el comportamiento de la red, como se verá mas adelante -capacidad de generalización de la red-, es necesario indicar que en el caso del Perceptron multicapa, generalmente, el número de neuronas ocultas no es parámetro significativo, pues dado un problema, pueden existir una gran cantidad de arquitecturas capaces de resolver de manera adecuada dicho problema. Además,

añadir o eliminar una neurona oculta no influye, de manera significativa, en la capacidad predictiva de la red.

En la actualidad existen líneas de investigación abiertas centradas en la determinación automática del número de neuronas ocultas, así como de capas ocultas, para cada problema en particular. En el caso del Perceptron multicapa, la mayor parte de estos trabajos se basan en la utilización de técnicas evolutivas, las cuales realizan una búsqueda en el espacion de de las arquitecturas de redes guiada por la optimización del rendimiento de la red [8], [9], [10].

3.3. Algoritmo de Retropropagación

La regla o algoritmo de aprendizaje es el mecanismo mediante el cual se van adaptando y modificando todos los parámetros de la red. En el caso del Perceptron multicapa se trata de un algoritmo de aprendizaje supervisado, es decir, la modificación de los parámetros se realiza para que la salida de la red sea lo más próxima posible a la salida proporcionada por el supervisor o salida deseada.

Por tanto, para cada patrón de entrada a la red es necesario disponer de un patrón de salida deseada.

Puesto que el objetivo es que la salida de la red sea lo más próxima posible a la salida deseada, el aprendizaje de la red se formula como un problema de minimización del siguiente modo:

$$Min_W E \tag{3.7}$$

siendo W el conjunto de parámetros de la red -pesos y umbrales- y E una función error que evalúa la diferencia entre las salidas de la red y las salidas deseadas. En la mayor parte de los casos, la función error se define como:

$$E = \frac{1}{N}\sum_{n=1}^{N} e(n) \tag{3.8}$$

donde N es el número de patrones o muestras y $e(n)$ es el error cometido por la red para el patrón n, dado por:

$$e(n) = \frac{1}{n_C}\sum_{i=1}^{n_C} (s_i(n) - y_i(n))^2 \tag{3.9}$$

siendo $Y(n) = (y_1(n), ..., y_{n_C}(n))$ y $S(n) = (s_1(n), ..., s_{n_C}(n))$ los vectores de salidas de la red y salidas deseadas para el patrón n, respectivamente.

De este modo, si W^* es un mínimo de la función error E, en dicho punto el error es próximo a cero, lo cual implica que la salida de la red es próxima a la salida deseada, alcanzando así la meta de la regla de aprendizaje.

Por tanto, el aprendizaje del Perceptron multicapa es equivalente a encontrar un mínimo de la funión error. La presencia de funciones de activación no lineales hace que la respuesta de la red sea no lineal respecto a los parámetros ajustables, por lo que el problema de minimización es un problema no lineal, y, como consecuencia, tienen que utilizarse técnicas de optimización no lineales para su resolución. Dichas técnicas están, generalmente, basadas en una adaptación de los parámetros siguiendo una cierta dirección de búsqueda. En el contexto de redes de neuronas, y en particular para el perceptron multicapa, la dirección de búsqueda más comúnmente usada es la dirección negativa del gradiente de la función E -método de descenso del gradiente-, pues conforme al cálculo de varias variables, ésta es la dirección en la que la función decrece.

Aunque, estrictamente hablando, el aprendizaje de la red debe realizarse para minimizar el error total, el procedimiento más utilizado está basado en métodos del gradiente estocástico, los cuales consisten en una sucesiva minimización de los errores para cada patrón, $e(n)$, en lugar de minimizar el error total E. Por tanto, aplicando el método de descenso del gradiente estocástico, cada parámetro w de la red se modifica para cada patrón de entrada n de acuerdo con la siguiente ley de aprendizaje.

$$w(n) = w(n-1) - \alpha \frac{\partial e(n)}{\partial w} \tag{3.10}$$

donde $e(n)$ es el error para el patrón n dado por la ecuación (3.9), y α es la razón o tasa de aprendizaje, parámetro que influye en la magnitud del desplazamiento en la superficie del error, como se analizará más adelante.

Debido a que las neuronas de la red están agrupadas en capas de distintos niveles, es posible aplicar el método del gradiente de forma eficiente, resultando el conocido algoritmo de Retropropagación [5] o *regla delta generalizada*. El término de retropropagación se utiliza debido a la forma de implementar el método del gradiente en el Perceptron multicapa, pues el error cometido en la salida de la red es propagado hacia atrás, transformándolo en un error para cada una de las neuronas ocultas de la red.

3.3.1. Obtención de la regla delta generalizada

A continuación se va a desarrollar la regla delta generalizada para el aprendizaje del Perceptron multicapa, que, como ya se ha comentado, no es más que una forma eficiente de aplicar el método del gradiente a los parámetros -pesos y umbrales- de dicha arquitectura. Para el desarrollo de esta regla es necesario distinguir dos casos: uno para los pesos de la capa oculta $C-1$ a la capa de salida y para los umbrales de las neuronas de salida, y otro para el resto de los pesos y umbrales de la red, pues las reglas de modificación de estos parámetros son diferentes.

Adaptación de pesos de la capa oculta $C-1$ a la capa de salida y umbrales de la capa de salida

Sea w_{ji}^{C-1} el peso de la conexión de la neurona j de la capa $C-1$ a la neurona i de la

capa de salida. Utilizando el método de descenso del gradiente (Ecuación 3.10), dicho parámetro se modifica siguiendo la dirección negativa del gradiente del error:

$$w_{ji}^{C-1}(n) = w_{ji}^{C-1}(n-1) - \alpha \frac{\partial e(n)}{\partial w_{ji}^{C-1}} \tag{3.11}$$

Por tanto, para la actualización de dicho parámetro es necesario evaluar la derivada del error $e(n)$ en dicho punto. De acuerdo con la expresión del error (Ecuación 3.9) y teniendo en cuenta, por un lado, que las salidas deseadas $s_i(n)$ para la red son constantes que no dependen del peso y, por otro lado, que el peso w_{ji}^{C-1} sólo afecta a la neurona de salida i, $y_i(n)$ (Ver ecuación 3.3), se obtiene que:

$$\frac{\partial e(n)}{\partial w_{ji}^{C-1}} = -(s_i(n) - y_i(n)) \frac{\partial y_i(n)}{\partial w_{ji}^{C-1}} \tag{3.12}$$

A este punto, hay que calcular la derivada de la neurona de salida $y_i(n)$ respecto al peso w_{ji}^{C-1}. La salida de la red es la función de activación f aplicada a la suma de todas las entradas por sus pesos, como se muestra en la Ecuación (3.3). Aplicando la regla de la cadena para derivar la composición de dos funciones y teniendo en cuenta que, de todos los términos del sumatorio (Ecuación 3.3), el único que interviene en el peso w_{ji}^{C-1} es $w_{ji}^{C-1} a_j^{C-1}$, y por tanto, el único cuya derivada es distinta de cero, se obtiene:

$$\frac{\partial y_i(n)}{\partial w_{ji}^{C-1}} = f' \left(\sum_{j=1}^{n_{C-1}} w_{ji}^{C-1} a_j^{C-1} + u_i^C \right) a_j^{C-1}(n) \tag{3.13}$$

Se define el término δ asociado a la neurona i de la capa de salida -capa C- y al patrón n, $\delta_i^C(n)$, del siguiente modo:

$$\delta_i^C(n) = -(s_i(n) - y_i(n)) f' \left(\sum_{j=1}^{n_{C-1}} w_{ji}^{C-1} a_j^{C-1} + u_i^C \right) \tag{3.14}$$

Reemplazando entonces en la Ecuación (3.12) el valor de la derivada de la neurona de salida $y_i(n)$ dado por (3.13) y de acuerdo con el valor de $\delta_i^C(n)$ definido anteriormente (Ecuación 3.14), se obtiene que:

$$\frac{\partial e(n)}{\partial w_{ji}^{C-1}} = \delta_i^C(n) a_j^{C-1}(n) \tag{3.15}$$

Finalmente, reemplazando la derivada del error $e(n)$ respecto al peso w_{ji}^{C-1} obtenida en (3.15) en la Ecuación (3.11), se obtiene la ley para modificar dicho peso, la cual toma la siguiente expresión:

$$w_{ji}^{C-1}(n) = w_{ji}^{C-1}(n-1) + \alpha\delta_i^C(n)a_j^{C-1} \tag{3.16}$$
$$para\ j = 1, 2, ..., n_{C-1}\ i = 1, 2, ..., n_C$$

En la Ecuación (3.16) obtenida anteriormente se observa que para modificar el peso de la conexión de la neurona j de la capa $C-1$ a la neurona i de la capa de salida, basta considerar la activación de la neurona de que parte la conexión -neurona j de la capa $C-1$- y el término δ de la neurona a la que llega la conexión -neurona de salida i-, término que contiene el error cometido por la red para dicha neurona de salida (ver Ecuación 3.14).

La ley de aprendizaje obtenida anteriormente para modificar los pesos de la última capa puede generalizarse para los umbrales de las neuronas de salida. Como se ha comentado en la sección anterior, en el Perceptron multicapa el umbral de una neurona se trata como una conexión más a la neurona cuya entrada es constante e igual a 1. Siguiendo, entonces, la ley anterior (Ecuación 3.16), se deduce que los umbrales de las neuronas de la capa de salida se modifican con la siguiente expresión:

$$u_i^C(n) = u_i^C(n-1) + \alpha\delta_i^C(n)\ para\ i = 1, 2, ..., n_C \tag{3.17}$$

Adaptación de pesos de la capa c a la capa $c+1$ y umbrales de las neuronas de la capa $c+1$ para $c = 1, 2, ..., C-1$

Con el objetivo de que el desarrollo de la regla de aprendizaje para el resto de los pesos y umbrales de la red sea lo más claro posible, se elige un peso de la capa $C-2$ a la capa $C-1$. Sea w_{kj}^{C-1} el peso de la conexión de la neurona k de la capa $C-2$ a la neurona j de la capa $C-1$. Siguiendo el método de descenso del gradiente, la ley para actualizar dicho peso viene dada por:

$$w_{kj}^{C-2}(n) = w_{kj}^{C-2}(n-1) + \alpha\frac{\partial e(n)}{\partial w_{kj}^{C-2}} \tag{3.18}$$

En este caso, y a diferencia del anterior -pesos hacia la capa de salida-, el peso w_{kj}^{C-2} influye en todas las salidas de la red, por lo que la derivada del error $e(n)$ (Ecuación 3.9) respecto de dicho peso viene dada por la suma de las derivadas para cada una de las salidas de la red, es decir:

$$\frac{\partial e(n)}{\partial w_{kj}^{C-2}} = -\sum_{i=1}^{n_C}(s_i(n) - y_i(n))\frac{\partial y_i(n)}{\partial w_{kj}^{C-2}} \tag{3.19}$$

Para calcular la derivada de la salida $y_i(n)$ respecto al peso w_{kj}^{C-2} es necesario tener en cuenta que este peso influye en la activación de la neurona j de la capa oculta $C-1$,

a_j^{C-1}, y que el resto de las activaciones de las neuronas en esta capa no dependen de dicho peso. Por tanto, y de acuerdo con la Ecuación (3.3), se tiene que:

$$\frac{\partial y_i(n)}{\partial w_{kj}^{C-2}} = f'\left(\sum_{j=1}^{n_{C-1}} w_{ji}^{C-1} a_j^{C-1} + u_i^C\right) w_{ji}^{C-1} \frac{\partial a_j^{C-1}}{\partial w_{kj}^{C-2}} \tag{3.20}$$

Sustituyendo este valor en la Ecuación (3.19) y de acuerdo con la definición de δ en el punto anterior (Ecuación 3.14), se obtiene que:

$$\frac{\partial e(n)}{\partial w_{kj}^{C-2}} = \sum_{i=1}^{n_C} \delta_i^C(n) w_{ji}^{C-1} \frac{\partial a_j^{C-1}}{\partial w_{kj}^{C-2}} \tag{3.21}$$

Para obtener la ley de aprendizaje para el peso w_{kj}^{C-2}, sólo falta derivar la activación de la neurona j de la capa oculta $C-1$, a_j^{C-1}, respecto a dicho peso. De nuevo, aplicando la regla de la cadena a la Ecuación (3.2), dicha derivada es:

$$\frac{\partial a_j^{C-1}}{\partial w_{kj}^{C-2}} = f'\left(\sum_{k=1}^{n_{C-2}} w_{kj}^{C-2} a_k^{C-2} + u_j^{C-1}\right) a_k^{C-2}(n) \tag{3.22}$$

Se define el valor δ para las neuronas de la capa $C-1$, $\delta^{C-1}(n)$, como:

$$\delta_j^{C-1}(n) = f'\left(\sum_{k=1}^{n_{C-2}} w_{kj}^{C-2} a_k^{C-2} + u_j^{C-1}\right) \sum_{i=1}^{n_C} \delta_i^C(n) w_{ji}^{C-1} \tag{3.23}$$

Sustituyendo (3.22) en la Ecuación (3.21) y de acuerdo con el valor de $\delta_j^{C-1}(n)$ definido anteriormente, se obtiene que:

$$\frac{\partial e(n)}{\partial w_{kj}^{C-2}} = \delta_j^{C-1}(n) a_k^{C-2}(n) \tag{3.24}$$

Y como consecuencia, la ley de aprendizaje para modificar el peso w_{kj}^{C-2} viene dada por:

$$w_{kj}^{C-2}(n) = w_{kj}^{C-2}(n-1) + \alpha \delta_j^{C-1}(n) a_k^{C-2}(n) \tag{3.25}$$

$$para\ k = 1, 2, ..., n_{C-2}\ y\ j = 1, 2, ..., n_{C-1}$$

Al igual que la ley obtenida para modificar los pesos de la última capa (Ecuación 3.16), en este caso, también se observa que para modificar el peso de la conexión de la neurona k de la capa $C-2$ a la neurona j de la capa $C-1$, basta considerar la activación

de la neurona de la que parte la conexión -neurona k de la capa $C-2$- y el término δ de la neurona a la que llega la conexión -neurona j de la capa $C-1$- (ver Ecuación 3.25). La diferencia radica en la expresión del término δ. Para los pesos de la capa $C-2$ a $C-1$ dicho término viene dado por la derivada de la función de activación y por la suma de los términos δ asociados a las neuronas de la siguiente capa -en este caso, las neuronas de salida- multiplicados por los pesos correspondientes, como se indica en el Ecuación (3.23).

Llegados a este punto, es posible generalizar la ley dada por la ecuación (3.25) para los pesos de la capa c a la capa $c+1$ ($c = 1, 2, ..., C-2$). Para ello basta tener en cuenta la activación de la que parte la conexión y el término δ de la neurona a la que llega la conexión. Este término se calcula utilizando la derivada de la función de activación y la suma de los términos δ de las neuronas de la siguiente capa. De este modo:

$$w_{kj}^{c}(n) = w_{kj}^{c}(n-1) + \alpha\delta_{j}^{c+1}(n)a_{k}^{c}(n) \tag{3.26}$$

$$para\ k = 1, 2, ..., n_c,\ j = 1, 2, ..., n_{c+1}\ y\ c = 1, 2, ..., C-2$$

donde $a_k^c(n)$ es la activación de la neurona k de la capa c para el patrón n y $\delta_j^{c+1}(n)$ viene dado por la siguiente expresión:

$$\delta_{j}^{c+1}(n) = f'\left(\sum_{k=1}^{n_c} w_{kj}^{c}a_{k}^{c} + u_{j}^{c}\right)\sum_{i=1}^{n_{c+1}}\delta_{i}^{c+2}(n)w_{ji}^{c} \tag{3.27}$$

Es posible, también, generalizar la ley de aprendizaje para el resto de umbrales de la red; basta tratarlos como conexiones cuya entrada es constante e igual a 1. La ley para modificarlos viene dada por:

$$u_{j}^{c+1}(n) = u_{j}^{c+1}(n-1) + \alpha\delta_{j}^{c+1}(n) \tag{3.28}$$

$$para\ j = 1, 2, ..., n_{c+1}\ y\ c = 1, 2, ..., C-2$$

Derivada de la función de activación El cálculo de los valores de δ (Ecuaciones 3.15 y 3.27) para cada neurona del Perceptron multicapa requiere el cálculo de la derivada de la función de activación. Como se ha comentado, el Perceptron multicapa puede utilizar dos tipos de funciones de activación -función sigmoidal (Ecuación 3.4) y función tangente hiperbólica (Ecuación 3.5), por lo que los valores de δ dependen, en principio, de la función de activación empleada. A continuación, se van a obtener las derivadas para ambas funciones de activación, quedando así completado el cálculo de los valores de δ.

- Derivada de la función sigmoidal
  Derivando la expresión dada por la Ecuación (3.4), se obtiene que:

$$f_1'(x) = \frac{1}{(1+e^{-x})^2}(-e^{-x}) = \frac{1}{1+e^{-x}}\frac{e^{-x}}{1+e^{-x}} \tag{3.29}$$

  Por tanto,

$$f_1'(x) = f_1(x)(1-f_1(x)) \tag{3.30}$$

  Como consecuencia, cuando se utiliza la función sigmoidal, los valores de δ asociados a las neuronas de salida (Ecuación 3.14) adoptan la siguiente forma:

$$\delta_i^C(n) = -(s_i(n) - y_i(n))y_i(n)(1-y_i(n)) \; para \; i = 1, 2, ..., n_C \tag{3.31}$$

  Y los valores de δ para el resto de las neuronas de la red (Ecuación 3.27), vienen dados por:

$$\delta_j^{c+1}(n) = a_j^c(n)(1-a_j^c(n)) \sum_{i=1}^{n_{c+1}} \delta^{c+2}(n) w_{ji}^c \tag{3.32}$$

$$para \; j = 1, 2, ..., n_{c+1} \; y \; c = 1, 2, ..., C-2$$

- Derivada de la función tangente hiperbólica
  Teniendo en cuenta que $f_2(x) = 2f_1(x) - 1$, la derivada de la función $f_2(x)$ es:

$$f_2'(x) = 2f_1(x)(1-f_1(x)) \tag{3.33}$$

  Cuando se utilice, por tanto, la función de activación tangente hiperbólica, los valores de δ para las neuronas de la red adoptan las expresiones dadas por las Ecuaciones (3.31) y (3.32) multiplicadas por un facto de 2.

En la sección anterior se ha comentado que, generalmente, todas las neuronas del Perceptron multicapa suelen utilizar la misma función de activación, salvo las neuronas de salida, que pueden tener como función de activación la función identidad, es decir, $f(x) = x$. En este caso, la derivada de la función de activación en la salida es 1 y, como consecuencia, los valores de δ para las neuronas de salida adoptan la siguiente expresión:

$$\delta_i^C(n) = -(s_i(n) - y_i(n)) \; para \; i = 1, 2, ..., n_C \tag{3.34}$$

3.3.2. Razón de aprendizaje. Inclusión del momento en la ley de aprendizaje

El cambio en un peso de la red es poporcional al gradiente del error como se refleja en la equación (3.10). Dicha proporcionalidad viene dada por el parámetro α, también llamado razón o tasa de aprendizaje. Este parámetro es el encargado de controlar cuánto se desplazan los pesos de la red en la superficie del error siguiendo la dirección negativa del gradiente. Determina, por tanto, la magnitud de dicho desplazamiento, influyendo así en la velocidad de convergencia del algoritmo.

Valores altos de α, podrían favorecer una convergencia más rápida, pues permiten avanzar rápidamente en la superficie del error. Sin embargo, razones de aprendizaje altas pueden tener consecuencias negativas en el aprendizaje, como que el método se salte un mínimo e incluso que el método oscile alrededor del mínimo. Valores pequeños de α podrían evitar estos problemas, aunque a costa de una convergencia más lenta del algoritmo de aprendizaje, pues la magnitud de los desplazamientos en la superficie del error es más pequeña.

Un método simple para evitar la inestabilidad en el algoritmo de aprendizaje debido a la razón de aprendizaje es modificar la ley de aprendizaje dada por la ecuación 3.10 mediante la inclusión de un segundo término, llamado *momento*, obteniendo así la siguiente ley:

$$w(n) = w(n-1) = -\alpha \frac{\partial e(n)}{\partial w} + \eta \Delta w(n-1) \tag{3.35}$$

donde $\Delta w(n-1) = w(n-1) - w(n-2)$ es el incremento de w en la anterior iteración y η es un número positivo que controla la importancia asignada al incremento anterior.

3.3.3. Proceso de aprendizaje del perceptron multicapa

Como se ha comentado en el apartado anterior, el objetivo del aprendizaje o entrenamiento del perceptrón multicapa es ajustar los parámetros de la red -pesos y umbrales- con el fin de que las entradas presentadas produzcan las salidas deseadas, es decir, con el fín de minimizar la función de error E. En esta sección se van a detallar los pasos que involucra el proceso completo de aprendizaje del perceptron multicapa de acuerdo al algoritmo de retropropagación.

Sea $\{(X(n), S(n)), n = 1, ...N\}$ el conjunto de muestras o patrones que representan el problema a resolver, donde $X(n) = (x_1(n), ..., x_{n1}(n))$ son los patrones de entrada a la red, $S(n) = (s_1(n), ..., s_{nc}(n))$ son las salidas deseadas para dichas entradas y N es el número de patrones disponibles. Generalmente, es frecuente encontrar los patrones de entrada y salida normalizados o escalados mediante una transformación lineal en los intervalos [0,1] o [-1,1] dependiendo de las función de activación empleada.

Los pasos que componen el proceso de aprendizaje del perceptron multicapa son los siguientes:

1. Se inicializan los pesos y umbrales de la red. Generalmente, esta inicialización es aleatoria y con valores alrededor de cero.

2. Se toma un patrón n del conjunto de entrenamiento, $(X(n), S(n))$, y se propaga hacia la salida de la red el vector de entrada $X(n)$ utilizando las ecuaciones (3.1), (3.2) y (3.3), obteniéndose así la respuesta de la red para dicho vector de entrada, $Y(n)$.

3. Se evalúa el error cuadrático cometido por la red para el patrón n utilizando la ecuación (3.9).

4. Se aplica la regla delta generalizada para modificar los pesos y umbrales de la red. Para ello se siguen los siguientes pasos:

   a) Se calculan los valores δ para todas las neuronas de la capa de salida utilizando la ecuación (3.14).

   b) Se calculan los valores δ para el resto de las neuronas de la red utilizando la ecuación (3.27) empezando desde la última capa oculta y retropropagando dichos valores hacia la capa de entrada.

   c) Se modifican los pesos y umbrales de la red siguiendo las ecuaciones (3.16) y (3.17) para los pesos y umbrales de la capa de salida y (3.26) y (3.28) para el resto de parámetros de la red.

5. Se repiten los pasos 2, 3 y 4 para todos los patrones de entrenamiento, completando así un ciclo de aprendizaje, o *epoch*.

6. Se evalúa el error total E ecuación (3.8) cometido por la red. Dicho error también recibe el nombre de error de entrenamiento, pues se calcula utilizando los patrones de entrenamiento.

7. Se repiten los pasos 2, 3, 4, 5 y 6 hasta alcanzar un mínimo del error de entrenamiento, para lo cual se realizan m ciclos de aprendizaje.

Existe una variante del proceso de aprendizaje descrito anteriormente en la que los parámetros de la red se modifican una vez que todos los patrones han sido prentados a la red, y no para cada patrón de entrenamiento. Esta variante se conoce como proceso *batch*.

Entrenar una red neuronal para para aprender patrones presentes en los datos de entrada implica presentarle de forma iterativa ejemplos de las respuestas correctas conocidas. El fín del entrenamiento es encontrar un conjunto de pesos y umbrales que determine un mínimo global en la función de error.

Si el modelo no está sobreentrenado, -ver secciones 4.5.1 y 4.5.2- este conjunto de parámetros debería proporcionar una correcta generalización. Los entrenamientos que utilizan el método de descenso del gradiente ajustan progresivamente dichos parámetros en la dirección de máxima pendiente decreciente en la superficie del error. Encontrar el mínimo global de dicha superficie no está garantizado ya que ésta puede incluir muchos mínimos locales en los que el proceso de entrenamiento converge. La inclusión del factor de momento en la función de aprendizaje y la ejecución del entrenamiento con varios

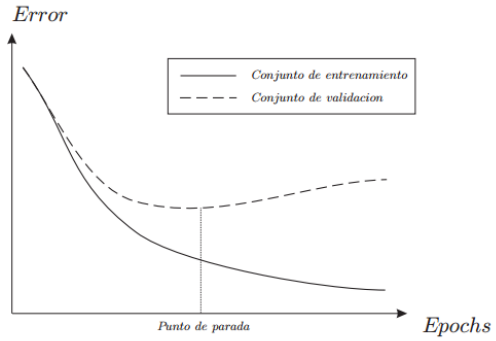

Figura 3.2: Representación del criterio de *early-stopping*

conjuntos iniciales distintos de pesos puede mejorar las probabilidades de encontrar un mínimo global.

Una ventaja de las redes neuronales es su habilidad para adaptarse a los cambios del mercado por medio de un reentrenamiento periódico. Una vez entrenada, el desempeño de una red neuronal se verá degradado a lo largo del tiempo a no ser que se reentrene. Un buen modelo debería ser robusto con respecto a la frecuencia de reentrenamiento y normalmente mejorará a medida que el reentrenamiento se realice de forma más frecuente.

3.3.4. Early stopping

Una manera de acometer el problema de sobre-entrenamiento (overfitting) es extraer un subconjunto de muestras del conjunto de entrenamiento (nótese que el conjunto de test se ha extraído previamente) y utilizarlo de manera auxiliar durante el entrenamiento.

Este subconjunto recibe el nombre de conjunto de validación. La función que desempeña el conjunto de validación es evaluar el error de la red tras cada epoch (o tras cada cierto número de epochs) y determinar el momento en que éste empieza a aumentar. Ya que el conjunto de validación se deja al margen durante el entrenamiento, el error cometido sobre él es un buen indicativo del error que la red cometerá sobre el conjunto de test. En consecuencia, se procederá a detener el entrenamiento en el momento en que el error de validación aumente y se conservarán los valores de los pesos del epoch anterior. Este criterio de parada se denomina *early-stopping*. La Figura 3.2 describe el procedimiento explicado.

3.4. Algunos algoritmos de entrenamiento avanzados

3.4.1. Levenberg-Marquardt

El método de Levenberg-Marquardt (LM) [29] [30] es un tipo de algoritmo de entrenamiento similar a los denominados quasi-Newton. Para entender la funcionalidad del método LM es conveniente empezar por comprender el método de Newton. El método de Newton es un método de segundo orden que constituye una alternativa a los métodos de gradiente conjugado para optimización rápida. El paso básico del método de Newton durante el epoch n es

$$\mathbf{w}(n+1) = \mathbf{w}(n) - \mathbf{H}(n)^{-1} g(n) \tag{3.36}$$

donde **w** es el vector de pesos, **g**(n) es el vector gradiente actual y **H**(n) es la matriz Hessiana (de derivadas segundas) de la función de error respecto a los pesos, evaluada en los valores actuales de pesos y umbrales. El método de Newton generalmente converge más rápido que los métodos de gradiente conjugado. Desafortunadamente, la matriz Hessiana es compleja y costosa de calcular para las redes feedforward. Los algoritmos denominados quasi-Newton (método de la secante) calculan y actualizan una aproximación de la matriz Hessiana sin necesidad de resolver las derivadas de segundo orden en cada iteración. La actualización realizada es función del gradiente.

Al igual que los métodos quasi-Newton, el algoritmo LM fue diseñado para acometer entrenamientos rápidos de segundo orden sin tener que calcular la matriz Hessiana. Cuando la función de error tiene la forma de una suma de cuadrados (el caso típico en las redes feedforward), entonces la matriz Hessiana puede aproximarse como

$$\mathbf{H} = \mathbf{J}^T \mathbf{J} \tag{3.37}$$

donde **J** es la matriz Jacobiana que contiene las primeras derivadas de la función de error de la red respecto a los pesos y umbrales. El gradiente se obtiene como

$$\mathbf{g} = \mathbf{J}^T \mathbf{e} \tag{3.38}$$

siendo **e** el vector de errores de la red. El cálculo de la matriz Jacobiana se reduce a un simple cálculo del algoritmo de retropropagación que es mucho más sencillo que construir la matriz Hessiana. El método LM actualiza los pesos de forma similar al método de Newton pero introduciendo los cambios anteriores:

$$\mathbf{w}(n+1) = \mathbf{w}(n) \left[\mathbf{J}^T J + \mu \mathbf{I}\right]^1 \mathbf{J}^T \mathbf{e} \tag{3.39}$$

donde el parámetro μ actúa como tasa de aprendizaje. Cuando μ es cero, el algoritmo se convierte en el método de Newton empleando la forma aproximada del cálculo de la matriz Hessiana:

$$\mathbf{w}(n+1) = \mathbf{w}(n) \mathbf{H}^1 \mathbf{g} \tag{3.40}$$

Cuando μ es grande, se puede despreciar la aportación de la matriz Hessiana al cálculo y obtenemos el método de descenso de gradiente con un tamaño de paso $(1/\mu)$:

$$\mathbf{w}(n+1) = \mathbf{w}(n)\frac{1}{\mu}\mathbf{g} \tag{3.41}$$

El método de Newton es más eficaz cerca de un mínimo de error, así que el objetivo es migrar al método de Newton tan pronto como sea posible. Así pues, se reduce μ tras cada paso exitoso (cada vez que el error se logra reducir) y sólo se aumenta cuando el error aumenta respecto a la iteración anterior. Con esta metodología, la función de error siempre disminuye tras cada iteración del algoritmo.

El algoritmo LM parece ser el método más rápido para entrenar redes feedforward de tamaño moderado (hasta varios centenares de parámetros). Su principal desventaja es que requiere almacenar las matrices Jacobianas que, para ciertos conjuntos de datos, pueden ser muy grandes, lo que significa un gran uso de memoria.

3.4.2. Regularización bayesiana

La regularización constituye un criterio de parada de entrenamiento alternativo al *earlystopping* visto en la Sección 3.3.4. El método de regularización bayesiana (BR) implica modificar la función de coste, normalmente el MSE. El MSE sobre el conjunto de datos de test (MSEd) se expresa mediante una suma de cuadrados de los errores individuales e_i de cada muestra de un conjunto de test de N datos de la siguiente forma:

$$MSE_d = \frac{1}{N}\sum_{i=1}^{N}(e_i)^2 \tag{3.42}$$

La modificación comentada tiene el propósito de mejorar la capacidad de generalización del modelo. Para conseguirlo, la función de coste de la Ecuación 3.42 se amplía con la adición de un término MSE_w que incluye el efecto de la suma de cuadrados de los pesos de la red. Es decir:

$$MSE_w = \frac{1}{N}\sum_{i=1}^{N}(e_w)^2 \tag{3.43}$$

Y la función de coste modificada queda así:

$$MSE = \beta \cdot MSE_d + \alpha \cdot MSE_w \tag{3.44}$$

donde β y α son parámetros que deben ser ajustados según la metodología Bayesiana de MacKay [31] [**?**]. Esta metodología asume que los pesos y umbrales son variables aleatorias que siguen distribuciones específicas (normalmente Gaussianas). Los parámetros de regularización están relacionados con las varianzas desconocidas asociadas a estas distribuciones. Se pueden estimar estos parámetros mediante técnicas estadísticas. Una característica de este algoritmo es que proporciona una medida de cuántos parámetros de la red (pesos y umbrales) están siendo efectivamente usados por ésta. Este número de

parámetros efectivos debería llegar a ser aproximadamente constante, sin importar cuán grande se haga el número de parámetros "reales" de la red. Para esto se asume que la red ha sido entrenada durante suficientes epochs y que el entrenamiento ha convergido.

Se conoce que la técnica de regularización óptima requiere la costosa construcción de la matriz Hessiana. Como se ha indicado en la sección anterior, existen algoritmos que solventan esta dificultad mediante aproximaciones de dicha matriz. Un algoritmo que se emplea a menudo en combinación con la técnica de regularización Bayesiana es el algoritmo LM. Otra consideración importante es que el algoritmo BR funciona mejor cuando tanto el rango de entrada como el de salida se encuentran en [1, 1]. Si no se da el caso para un determinado problema, es conveniente escalar los datos.