UNIVERSIDAD DE A CORUÑA
FACULTAD DE INFORMÁTICA
DEPARTAMENTO DE COMPUTACIÓN

ESTUDIO, IMPLEMENTACIÓN Y ANÁLISIS DE NUEVOS ALGORITMOS DE APRENDIZAJE Y NUEVAS MEDIDAS DE TOLERANCIA AL RUIDO PARA REDES FUNCIONALES Y NEURONALES

Memoria de Tesis Doctoral

Autor:

Óscar Fontenla Romero

Directores:

Amparo Alonso Betanzos

Enrique Castillo Ron

Dña. Amparo Alonso Betanzos, Profesora Titular de Universidad del Departamento de Computación de la Facultad de Informática de la Universidad de A Coruña

y

D. Enrique Castillo Ron, Catedrático de Universidad del Departamento de Matemática Aplicada y Ciencias de la Computación de la Escuela Superior de Ingenieros de Caminos de la Universidad de Cantabria

CERTIFICAN que:

La memoria que se presenta, titulada *Estudio, implementación y análisis de nuevos algoritmos de aprendizaje y nuevas medidas de tolerancia al ruido para redes funcionales y neuronales* ha sido realizada por D. Óscar Fontenla Romero bajo nuestra dirección en el Departamento de Computación de la Universidad de A Coruña y constituye la Tesis que presenta para optar al grado de Doctor.

Y para que así conste firmamos la presente en A Coruña a 24 de Abril de 2002.

Fdo.: Amparo Alonso Betanzos

Fdo.: Enrique Castillo Ron

A Paola

Agradecimientos

- A Amparo Alonso Betanzos por darme la oportunidad de realizar esta Tesis Doctoral. Gracias también por su apoyo constante día tras día.
- A Enrique Castillo Ron por las innumerables aportaciones a este trabajo. Gracias además por su extrema generosidad.
- A José Carlos Principe por darme la oportunidad de colaborar con un magnífico grupo de investigación. Gracias también por cada una de las sugerencias y discusiones que han ayudado a mejorar el trabajo presentado en esta Tesis.
- A Deniz Erdogmus por tenderme una mano amiga en una ciudad extraña en un país lejano. Asimismo, gracias por la inestimable ayuda y sugerencias que ha aportado en el trabajo presentado en esta memoria.
- A Bertha Guijarro Berdiñas por cada una de sus magníficas críticas.
- A Vicente Moret Bonillo por ofrecerme la posibilidad de formar parte de un extraordinario grupo de investigación.
- A todos los miembros del laboratorio LIDIA: Elena, Eduardo, Juan A., Juan G., Mar, Mariano, Marta y Noelia; por "soportarme" durante todos estos años.
- A mis padres, a mi hermano, a mis abuelos y a toda mi familia por su apoyo incondicional durante la realización de este trabajo.
- En último lugar, y no por ello menos importante, a Paola, por cada uno de los minutos que le ha "robado" esta tesis.

Índice General

Índice de Figuras

Índice de Tablas

Capítulo 1

Introducción

"Si el cerebro humano fuese tan simple que pudiésemos entenderlo, entonces seríamos tan simples que no podríamos entenderlo"

Anónimo

Los orígenes de las redes de neuronas artificiales (RRNNAA) parten del descubrimiento en 1906 de Ramón y Cajal de las neuronas como células independientes en cuanto a su función, estructura y origen. Estos estudios llevaron posteriormente a diversos neurólogos, entre los que destaca D. Hebb (1949) a realizar diversas especulaciones fisiológicas [60]. Sin embargo, los pioneros en este área fueron McCulloch y Pitts [95] al formalizar en 1943 el funcionamiento de una neurona. Posteriormente, en la década de los 50, el trabajo de F. Rosenblatt [117] supuso un importante avance al desarrollar el perceptrón simple, una subclase de red neuronal. A pesar de esta importante investigación, la exposición matemática de la naturaleza de los perceptrones elaborada por M. Minsky y S. Papert [97] en 1969 puso en claro el alcance y las limitaciones de estos novedosos modelos de proceso en el campo de la Inteligencia Artificial. Esta crítica tuvo una amplia repercusión y determinó en gran medida el decaimiento de estos modelos en la década de los setenta. El resurgir de las redes neuronales artificiales no se produjo hasta la década de los 80, en la que G. Hinton y J. A. Anderson [63] publicaron su libro *Parallel Model of Associative Memory* y el posterior trabajo de J. Hopfield [64] un año más tarde.

Aunque no existe una definición universalmente aceptada, una red de neuronas artificiales puede ser definida como un modelo computacional, paralelo y distribuido, basado en la conexión de varios elementos de proceso (neuronas) para que conjuntamente realicen una función común. Las redes están organizadas en una sucesión ordenada de capas, las cuales están formadas por conjuntos disjuntos de neuronas.

Las redes de neuronas artificiales están basadas en las características y el funcionamiento de las neuronas naturales, sin embargo, este tipo de sistemas pueden abordarse desde dos puntos de vista:

- *Biológico.* Pretende el desarrollo de estructuras neuronales para el estudio y modelado de los procesos de aprendizaje biológicos. Este campo de las redes de neuronas recibe el nombre de *neurociencia.*

- *Matemático o Ingenieril.* Persigue la creación de modelos y algoritmos eficientes de aprendizaje máquina para la resolución de problemas prácticos de gran complejidad, independientemente de si esos modelos mimetizan o no los procesos biológicos. Este área de conocimiento es conocida como *neurocomputación.*

El trabajo presentado en esta tesis se encuadra dentro de la segunda aproximación. La neurocomputación comprende tres áreas principales de actividad:

- *Arquitectura y teoría.* Desarrollo y estudio de arquitecturas neuronales, así como la creación de teorías de operación para ellas.

- *Implementación.* Se ocupa de la implementación en hardware de las diferentes arquitecturas neurocomputacionales. Incluye lenguajes específicos, neurocomputadores y medidas de tolerancia al ruido.

- *Aplicaciones.* Consiste en la aplicación práctica de los modelos teóricos con el fin de resolver problemas reales de difícil solución. Las redes de neuronas se han aplicado con éxito en diversas áreas de conocimiento, e.g., economía, medicina, química, meteorología, etc. En concreto, son de gran interés, entre otros, para la clasificación de patrones, aproximación de funciones y procesado de señales.

Las redes de neuronas pueden ser clasificadas, empleando como criterio de clasificación la topología de la red, en los siguientes tipos:

- *Redes con alimentación hacia delante*: Es este tipo de redes, los datos fluyen en un único sentido desde las entradas a las salidas. Las salidas de las neuronas de una capa se conectan sólo con las entradas de las neuronas de la siguiente capa. Por tanto, no es posible que la salida de una neurona esté conectada con la entrada de alguna neurona de la misma capa o de capas previas.

- *Redes recurrentes*: Se diferencian de las anteriores en la existencia de lazos de realimentación en la red. Estos lazos pueden ser entre neuronas de diferentes capas, neuronas de la misma capa o de una neurona consigo misma.

En una red de neuronas es necesario definir un procedimiento por el cual las conexiones del sistema varíen para proporcionar la salida deseada. Este proceso es conocido como aprendizaje. El aprendizaje en una red de neuronas se define como el proceso de modificación de los parámetros (pesos), mediante un procedimiento preestablecido, de forma que la red pueda llevar a cabo de forma efectiva una tarea determinada. Mediante este proceso la red es capaz de "aprender"a solucionar un problema, a partir de un conjunto de ejemplos, y de generalizar para realizar la tarea correctamente sobre datos no empleados durante el aprendizaje. Existen tres paradigmas de aprendizaje básicos:

- *Aprendizaje supervisado*: El aprendizaje supervisado presenta a la red las salidas que debe proporcionar cuando se le muestran unas determinadas señales de entrada. Se observa la salida de la red y se determina la diferencia entre ésta y la señal deseada. Posteriormente, los pesos de la red son modificados de acuerdo con la magnitud del error cometido.

- *Aprendizaje no supervisado*: En el aprendizaje no supervisado no se conoce la señal que debe proporcionar la red de neuronas (salida deseada). En este caso, la red se organiza ella misma agrupando, según sus características o mediante una función que defina el problema a resolver, los diferentes datos de entrada.

- *Aprendizaje por refuerzo*: Es una combinación de las dos anteriores en la cual cada cierto tiempo se presenta a la red una valoración global de cómo lo está haciendo. En este tipo de aprendizaje sólo conocemos si la salida de la red se corresponde o no con la señal deseada, es decir, la información es de tipo booleana (verdadero o falso). A diferencia del aprendizaje supervisado no se conoce la magnitud del error.

El trabajo desarrollado en la presente tesis doctoral está encuadrado en el área de las redes de neuronas con alimentación hacia delante con aprendizaje supervisado. Además, los métodos empleados están desarrollados desde un punto de vista neurocomputacional, por tanto, no se ha intentado crear modelos biológicamente plausibles, sino modelos matemáticos e ingenieriles que funcionen de forma eficaz y eficiente para resolver determinados problemas complejos.

1.1 Motivación

El problema del aprendizaje supervisado en redes de neuronas con alimentación hacia delante puede formularse en términos de minimización de una función de error, $J(\mathbf{w})$, denominada también función de coste u objetivo, donde $\mathbf{w}$ es un vector que

contiene los pesos de la red. Por tanto, el problema consiste en encontrar los pesos óptimos, $\mathbf{w}^*$, que minimizan la función $J(\mathbf{w})$, o equivalentemente, resolver el siguiente problema de optimización

$$\mathbf{w}^* = \arg\min_{\mathbf{w}} J(\mathbf{w}). \tag{1.1}$$

Sin embargo, en las redes multicapa la función de error es, en general, una función con un grado de no linealidad muy elevado y, por tanto, pueden existir muchos mínimos que satisfagan la siguiente condición

$$\nabla J = 0, \tag{1.2}$$

donde ∇J representa el gradiente de J en el espacio de pesos. El mínimo para el cual el valor de la función de coste es más pequeño se denomina mínimo global, mientras que los otros mínimos se denominan mínimos locales. Además, existen otros puntos que satisfacen la condición (1.2), tales como los máximos o los puntos de silla (*saddlepoints*). En general, cualquier vector $\mathbf{w}$ para el cual se satisface la condición (1.2) se denomina punto estacionario. Las diferentes clases de puntos estacionarios se muestran esquemáticamente en la figura 1.1.

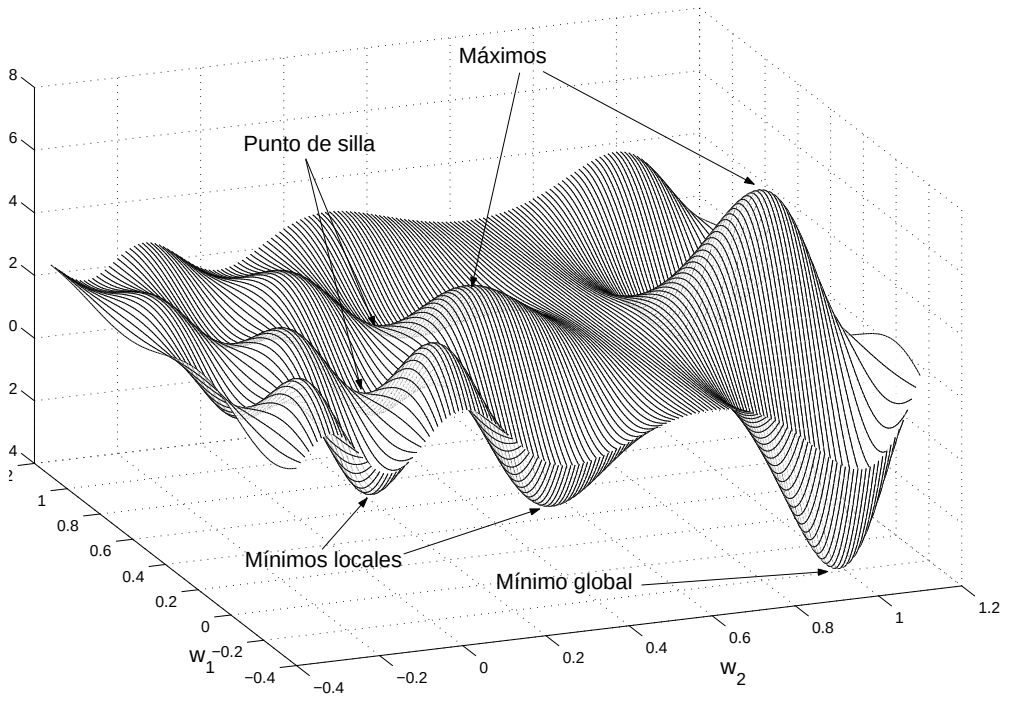


Figura 1.1: Función de error con los diferentes tipos de puntos estacionarios: mínimo global, mínimos locales, máximos y puntos de silla.

Como consecuencia de la no linealidad de la función de error no es posible, en general, desarrollar una solución analítica para encontrar el óptimo global. En lugar de eso, se emplean algoritmos basados en una búsqueda iterativa a través del espacio generado por los pesos. Estos métodos consisten en una sucesión de iteraciones de la forma

$$\mathbf{w}(n+1) = \mathbf{w}(n) + \Delta\mathbf{w}(n), \tag{1.3}$$

donde n es un índice que representa el número de la iteración.

Para una red multicapa con alimentación hacia delante, las derivadas de la función de error respecto a los parámetros de la red se pueden obtener eficientemente. El uso de esta información del gradiente es la idea central de los algoritmos de minimización empleados para el aprendizaje de las redes de neuronas, los cuales son lo suficientemente rápidos para ser de uso práctico en aplicaciones reales. Estos algoritmos emplean la siguiente regla de actualización de los pesos:

$$\mathbf{w}(n+1) = \mathbf{w}(n) - \mu\frac{\partial J}{\partial \mathbf{w}(n)} \tag{1.4}$$

donde μ es una constante pequeña y positiva denominada paso de aprendizaje. Esta regla está basada en la regla general mostrada en (1.3) en la cual se ha sustituido el término $\Delta\mathbf{w}(n)$ por $-\mu\frac{\partial J}{\partial \mathbf{w}(n)}$. Esta regla es la base del algoritmo de retropropagación del error propuesto por Rumelhart et al. [118]. Este algoritmo fue el primer algoritmo eficaz desarrollado para perceptrones multicapa y se basa en el siguiente procedimiento de dos pasos:

- *Propagación de las entradas hacia delante*: en esta fase los pesos del sistema permanecen inalterables y el objetivo es computar la salida de las neuronas de cada capa hasta lograr el valor de las salidas de la red para los parámetros actuales.

- *Propagación del error hacia las entradas*: en esta segunda etapa se calcula el error entre la salidas de la red, calculadas en la fase anterior, y la salida deseada. Este error se propaga hacia atrás, capa a capa, hasta llegar a la capa de entrada. El error correspondiente a cada capa, junto con la información del gradiente, se emplea para la actualización de los pesos.

A pesar de la gran potencia y flexibilidad del algoritmo de retropropagación del error, y en general de todos los algoritmos basados en gradiente, este tipo de métodos presentan algunos problemas. Los dos principales son los siguientes:

- *La convergencia hacia el mínimo global no está garantizada.* El aprendizaje de la red se lleva a cabo mediante la minimización de la función de error. La mayoría de los algoritmos de aprendizaje, entre ellos el de retropropagación del

error, emplean sólo información local del estado actual para la minimización de esta función. Sin embargo, como ya se ha mencionado anteriormente, esta función puede contener diversos mínimos locales. Por tanto, si el estado inicial de la red, determinado por los valores iniciales de los pesos, está en el área de atracción de un mínimo local, establecida por la información del gradiente, el algoritmo se verá irremediablemente atraído por él. La figura 1.2 muestra un ejemplo de este comportamiento. En este ejemplo, si el estado inicial de la red es el indicado por cualquiera de los puntos (2) los algoritmos de gradiente quedarán atrapados en algún mínimo local ya que seguirán la dirección indicada por la flecha. Sin embargo, si el estado inicial es el mostrado en el punto (1) el algoritmo de aprendizaje logrará alcanzar el óptimo global. Por tanto, en este tipo de algoritmos el punto de partida es decisivo en su rendimiento.

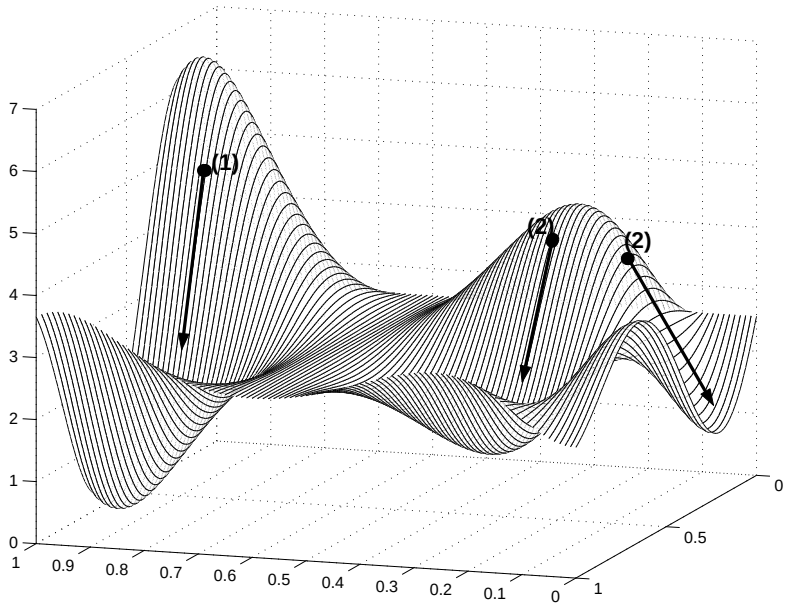


Figura 1.2: Función de error con mínimos locales.

- *Lenta velocidad de convergencia.* Asimismo, los algoritmos basados en descenso de gradiente suelen presentar una convergencia muy lenta hacia la solución. Esto se debe a que en aquellas regiones de la superficie de error cuya derivada sea prácticamente nula el algoritmo avanza muy lentamente. Este hecho es fácilmente observable en la ecuación (1.4), ya que si el término del gradiente es prácticamente nulo el valor de los pesos en el instante $n + 1$, $\mathbf{w}(n + 1)$, es aproximadamente igual al del instante anterior, $\mathbf{w}(n)$. Este fenómeno es especialmente crítico en las mesetas de la función de error. Esto se muestra gráficamente en la figura 1.3 que contiene una función de coste con una región

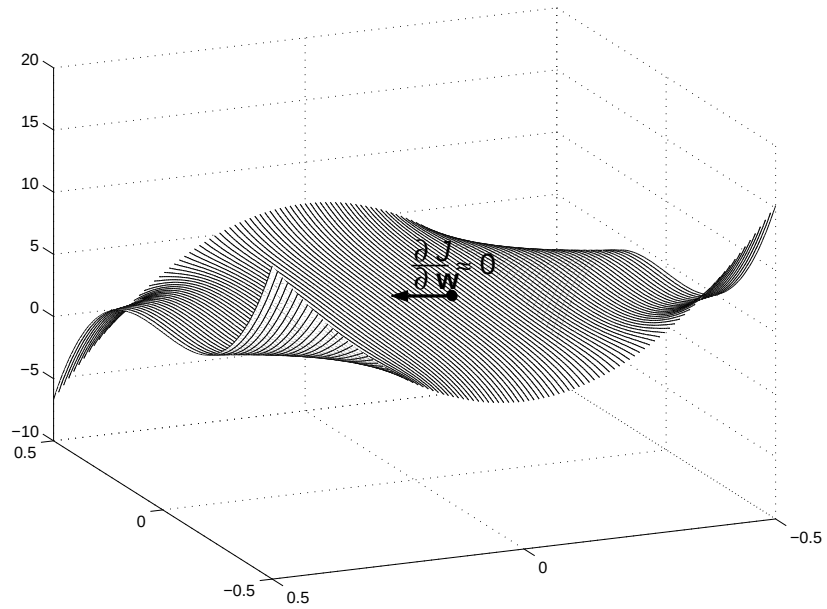


Figura 1.3: Función de error con una meseta.

prácticamente plana (meseta). En esta región la derivada es prácticamente nula por lo cual el algoritmo necesitará muchas iteraciones hasta alcanzar el mínimo global situado en la parte izquierda de la función. Este hecho tiene dos consecuencias directas: a) los algoritmos no son tan eficientes en funciones complejas, y b) provocan la aparición de falsos mínimos locales, ya que un avance lento puede hacer suponer al operador que se ha alcanzado un mínimo local y, por tanto, no hay posibilidad de mejora, cuando si podría haberla después de un gran número de iteraciones. La aparición de mesetas, generadas por la falta de unicidad en la solución, es muy frecuente en funciones de error complejas lo cual repercute muy negativamente en la eficiencia de este tipo de algoritmos.

1.2 Objetivos y estructura de la Tesis

Los principales objetivos de la presente tesis doctoral son:

- Proporcionar nuevos métodos de aprendizaje supervisado para redes de neuronas con alimentación hacia delante que permitan paliar los problemas encontrados en los métodos tradicionales de aprendizaje, i.e., convergencia hacia mínimos locales y lenta velocidad de operación.

- Desarrollo de medidas de inmunidad al ruido para redes neuronales y funcionales. Las redes funcionales han sido presentadas recientemente como unos

potentes modelos que generalizan a las redes de neuronas [25, 26]. En estas redes las funciones asociadas con cada neurona no son fijas sino que deben ser aprendidas a partir de los datos de entrenamiento. Por tanto, no es necesario el uso de pesos entre las neuronas ya que el efecto de éstos queda subsumido por las funciones neuronales. Este tipo de redes han sido empleadas con éxito en diversos problemas reales obteniendo en algunos casos mejores resultados que las redes de neuronas [29, 28].

La presente tesis está organizada en dos partes diferenciadas, cada una de las cuales se corresponde con uno de los objetivos mencionados previamente. En la primera parte de la memoria se presentan los nuevos métodos para el aprendizaje de redes de neuronas, mientras que en la segunda parte se detallan los aspectos concernientes a la inmunidad al ruido y tolerancia a fallos de las redes funcionales y neuronales.

La organización de la primera parte de la memoria es la siguiente. En el capítulo 2 se muestra brevemente el estado del arte, detallando los principales métodos, presentados previamente por otros autores, encaminados a solucionar el problema de los mínimos locales y la lenta velocidad de convergencia.

A continuación, en el capítulo 3 se aborda el problema del aprendizaje en redes de neuronas de una sola capa. Para este caso, se propone una nueva función de error que garantiza la no existencia de mínimos locales y al mismo tiempo permite llevar a cabo el aprendizaje mediante la resolución de un sistema de ecuaciones lineales. Consecuentemente, el algoritmo propuesto es muy veloz y proporciona la solución en un tiempo mucho menor que cualquiera de los algoritmos actuales, basados en métodos iterativos. A continuación, el método se extiende para realizar, al mismo tiempo, el aprendizaje de las funciones neuronales en vez de suponerlas fijas. Este tipo de red, con aprendizaje de funciones, es una aproximación que está a caballo entre las redes neuronales y funcionales. El trabajo presentado en este capítulo se encuentra publicado en la revista *Neural Computation* [27].

Posteriormente, en el capítulo 4 se desarrolla una extensión del método propuesto en el capítulo anterior para que sea aplicable a redes multicapa. El método propuesto está basado en la retropropagación de la salida deseada desde la capa de salida hasta la de entrada. Este proceso permite el aprendizaje de cada una de las capas mediante el uso del sistema de ecuaciones lineales presentado en el capítulo anterior. Esta aproximación proporciona un excelente método de inicialización de los pesos.

En el capítulo 5, y continuando con el problema del aprendizaje en redes multicapa, se propone un novedoso algoritmo basado en la combinación de dos métodos. La aproximación propuesta emplea un procedimiento en dos fases. En la primera fase, todos los pesos de la red se actualizan empleando cualquiera de los algoritmos convencionales. Posteriomente, en la segunda fase los pesos de la última capa se

obtienen usando el método propuesto en el capítulo 3 mientras que los pesos de las capas previas se actualizan con el algoritmo estándar. Este método híbrido proporciona un procedimiento muy eficiente para el entrenamiento de una red multicapa que permite evitar mínimos locales y acelerar la convergencia hacia la solución.

En los capítulos anteriores se aborda el problema de la aproximación de funciones mediante un modelo global de los datos. Sin embargo, en el capítulo 6 se propone un nuevo método basado en la aproximación de una función mediante modelos locales. El sistema presentado combina dos arquitecturas diferentes de redes de neuronas: un mapa autoorganizativo (self-organizing map, SOM) y un perceptrón de una capa. El objetivo del SOM es el de derivar los modelos locales a partir de los datos, mientras que el perceptrón se encarga de realizar una predicción precisa para cada uno de los modelos. El entrenamiento de este segundo subsistema se realiza empleando el método de aprendizaje propuesto en el capítulo 3. El trabajo presentado en este capítulo ha sido aceptado para su publicación en el congreso internacional *International Conference on Artificial Neural Networks (ICANN 2002)* [49].

La segunda parte de la memoria, está organizada de la siguiente manera. En el capítulo 7 se realiza una breve introducción a las redes funcionales. Además, se presentan las principales aportaciones de otros autores y los estudios encaminados al desarrollo de medidas de inmunidad al ruido en redes de neuronas. Algunos de los trabajos presentados en este capítulo forman la base en la que se han inspirado las medidas para redes funcionales propuestas en este trabajo.

En el capítulo 8 se aborda el segundo de los objetivos de la tesis. En él se desarrolla un modelo matemático para predecir la inmunidad al ruido de redes funcionales y neuronales con alimentación hacia delante. Las medidas propuestas permiten determinar *a priori* la degradación del rendimiento del sistema cuando se aplica ruido en las entradas o en los parámetros, de forma que podría utilizarse como un método alternativo para la selección de modelos y para predecir el comportamiento del sistema cuando se implemente en un medio físico. Parte de este capítulo ha sido publicado en el congreso internacional *International Work-Conference on Artificial and Natural Neural Networks (IWANN 2001)* [31].

Finalmente, en el capítulo 9 se presentan las principales conclusiones y aportaciones de esta tesis, así como las posibles líneas de trabajo futuro.

Parte I

Algoritmos de aprendizaje para redes de neuronas artificiales

Capítulo 2

Antecedentes

"La calidad nunca es un accidente;
siempre es el resultado de un esfuerzo de
la inteligencia"

John Ruskin

El método de retropropagación del error ha sido usado ampliamente, para el entrenamiento de redes de neuronas con alimentación hacia delante, durante las últimas dos décadas. Sin embargo, en el capítulo anterior se han mostrado los dos problemas principales que presenta este método, i.e., convergencia hacia mínimos locales y lenta convergencia hacia la solución. Durante los últimos años se han desarrollado diversas propuestas para paliar estas deficiencias. A continuación, se describirán de forma concisa algunas de las soluciones aportadas para evitar cada uno de los problemas.

En cuanto al problema de convergencia hacia mínimos locales se han propuesto varias soluciones, inspiradas en diferentes técnicas:

- El *enfriamiento simulado (simulated annealing)* [74] es uno de los métodos más estudiados y conocidos para intentar solucionar el problema de los mínimos locales. Esta técnica acepta la degradación de la función de error con una probabilidad no nula pero decreciente. En el método de retropropagación del error, el deterioro se logra añadiendo ruido blanco gaussiano en la actualización de los pesos en cada una de las iteraciones del algoritmo. En este caso, la temperatura en la distribución de Boltzman se corresponde con la varianza del ruido. El proceso se inicia con un valor elevado y se reduce progresivamente durante el entrenamiento, de acuerdo con un plan preestablecido, hasta llegar a un valor igual a cero. Un ejemplo de este tipo de aprendizaje es el presentado por Styblinski y Tang [124]. Geman y Geman [52] probaron la existencia de planes de enfriamiento los cuales garantizan la convergencia hacia el mínimo global. Sin

embargo, todos ellos requieren una inmensa cantidad de tiempo, y cualquier plan diseñado para reducir el tiempo lleva, en general, a la degradación del rendimiento del aprendizaje. Además, se han propuesto otras variantes como el *mean field annealing* [16, 106], que consiste en una aproximación determinista que proporciona una mayor velocidad de convergencia. En este método la temperatura controla la pendiente de las funciones sigmoidales, i.e., empezando con una pendiente pequeña que se irá incrementando paulatinamente. Aunque esta aproximación es cincuenta veces más veloz que el enfriamiento simulado para algunos problemas [16], la calidad de la solución es muy dependiente de la elección de la temperatura inicial, el plan y la temperatura final [89].

- *Modificaciones del algoritmo estándar de retropropagación del error*, como la presentada por Fukuoka et al. [50]. La idea básica del método es mantener la derivada de la función neuronal relativamente grande aunque algunos de los errores sean también grandes. Con este propósito, cada peso en la red es multiplicado por un factor en el rango $(0, 1]$ en intervalos constantes durante el proceso de aprendizaje. Otra de las modificaciones encontradas en la bibliografía fue la propuesta por Plaut et al. [107]. Este método, denominado *momentum*, consiste en una modificación de la regla de retropropagación estándar basada en la incorporación de un término adicional que ayuda a la red a evitar algunos mínimos locales.

- *Inicialización apropiada de los pesos.* Kolen y Pollack demostraron que el algoritmo de retropropagación del error es muy sensible a los pesos iniciales de la red [78]. Habitualmente, los pesos se inicializan aleatoriamente con valores pequeños. Sin embargo, una inicialización inapropiada es una de las razones que provoca la convergencia hacia mínimos locales y un avance lento durante el aprendizaje. En concreto, Lee et al. [90] demostraron que unos valores iniciales de los pesos que sean demasiado grandes pueden causar, fácilmente, una saturación prematura de la red. Por otro lado, Wessels y Barnard [136] presentaron un método de inicialización que emplea medidas dirigidas a evitar aquellos estados físicos discernibles que estén correlacionados con un mínimo local.

- *Optimización global.* Cetin et al. [32] propusieron un nuevo método de aprendizaje en el cual la regla de descenso de gradiente empleada en el algoritmo de retropropagación del error es sustituida por un formalismo de descenso global. Esta metodología está basada en un esquema de optimización global el cual formula la optimización en términos del flujo de un sistema dinámico determinístico. Recientemente, Hara et al. [57] han presentado un nuevo método

de optimización inspirado en la dinámica de Glauber que itera la minimización global de la superficie de error respecto a una dirección seleccionada aleatoriamente. En este método se supone una tendencia a evitar mínimos locales de pequeño tamaño. Finalmente, se han propuesto también algunas aproximaciones basadas en algoritmos genéticos [9, 142] .

- *Actualización on-line de los pesos.* En el campo de las redes de neuronas se han propuesto dos esquemas distintos para la actualización de los pesos. El primero de ellos consiste en acumular las derivadas parciales de la función de error respecto a los pesos sobre todos los ejemplos de entrenamiento antes de actualizar los parámetros de la red. Este tipo de entrenamiento es conocido como aprendizaje *batch*. Con esta aproximación se obtiene una estimación más precisa del gradiente real. La otra posibilidad es actualizar los pesos después de cada muestra del conjunto de entrenamiento. Este procedimiento se denomina aprendizaje *on-line*. Con este segundo método, y seleccionando los datos aleatoriamente, se producen pequeñas fluctuaciones que a veces empeoran el valor obtenido de la función de error. Por tanto, si se emplea esta segunda aproximación es posible evitar algunos pequeños mínimos locales [139].

Los métodos mencionados han aportado soluciones relevantes al problema de los mínimos locales. Sin embargo, ninguno de ellos proporciona una solución realmente eficiente, que al mismo tiempo, solucione el problema de la lenta velocidad de convergencia. Además, aquellas aproximaciones que garantizan que la solución proporcionada coincide con el mínimo global requieren una excesiva cantidad de recursos temporales y computacionales, lo que provoca que no sean operativas en entornos reales.

En cuanto a las soluciones encontradas en la bibliografía para el problema de la lenta convergencia cabe destacar las siguientes:

- *Métodos de segundo orden.* En diversos trabajos [5, 23, 82, 104] se ha propuesto el uso de las segundas derivadas de la función de error, tales como los métodos de gradiente conjugado y quasi-Newton [3, 37], para lograr una mejora en la rapidez del sistema. En concreto, fue demostrado por LeCun et al. [88] que estos métodos son más eficientes, en términos de velocidad de convergencia, que los basados sólo en técnicas de descenso de gradiente con derivadas de primer orden. En concreto, Watrous demostró que el método quasi-Newton alcanza una rápida convergencia cerca de un mínimo [132]. Sin embargo, estos métodos requieren una mayor capacidad de memoria durante el proceso de aprendizaje.

- *Paso de aprendizaje adaptativo.* Muchos investigadores han indicado que un paso de aprendizaje constante durante todo el proceso de aprendizaje no es muy adecuado para superficies de error complejas. En este sentido, se han propuesto diversos métodos, de tipo heurístico, para la modificación dinámica del paso de aprendizaje [69, 72, 128]. Además de estos trabajos, Weir propuso un método basado en la autodeterminación de este parámetro [134].

- *Ajuste de la pendiente de las funciones de activación de tipo sigmoidal.* Hush et al. [68] han indicado que las superficies de error, debido al carácter no lineal de las funciones sigmoidales, tienden a tener muchas zonas planas así como otras con una pendiente muy apuntada. Si un método de descenso de gradiente llega a una zona plana de la función de coste, no hay un descenso del error significativo durante un determinado período de tiempo, después del cual el error decrece de nuevo. Este fenómeno se suele denominar saturación prematura. Debido a este fenómeno, muchas veces es necesario una gran cantidad de tiempo para salir de esta zona, lo cual retarda enormemente el proceso de aprendizaje. La solución básica, presentada por Yamada et al. [141] y Rezgui y Tepedelenlioglu [113], consiste en ajustar la pendiente de las funciones neuronales. Posteriormente, Sperduti y Antonina [123] extendieron el método de retropropagación del error añadiendo un procedimiento de descenso de gradiente en los parámetros que determinan la pendiente de las funciones sigmoidales.

- *Mejoras en la función de error.* El algoritmo de retropropagación del error estándar emplea la derivada de las funciones neuronales en la función de error propagada en la red. Sin embargo, la derivada de estas funciones tiene forma de campana de Gauss lo que causa, a veces, una velocidad de convergencia lenta cuando la salida de una neurona es cercana a cero o uno, ya que estos casos la derivada tiene un valor próximo a cero. Para eliminar este efecto sobre la señal de error, van Ooyen y Nienhuis [126] y Krzyzak et al. [84] emplearon un método basado en una función de error semejante a la entropía.

- *Inicialización apropiada de los pesos.* Nguyen y Widrow propusieron un método en el que se asigna a cada neurona de las capas ocultas una parte aproximada del rango de la salida deseada [102]. Asimismo, Drago y Ridella presentaron una aproximación basada en una inicialización estadísticamente controlada. Este método evita que las neuronas alcancen la zona de saturación durante el proceso adaptativo mediante la estimación de los valores máximos que deben tomar inicialmente los pesos [38]. Asimismo, se han presentado otras propuestas basadas en el uso del método de mínimos cuadrados [15, 140].

- *Modificaciones de los algoritmos estándar y de las funciones de error.* En este

sentido, Kumazawa [85] propuso un nuevo esquema de aprendizaje que compensa resultados incompletos del aprendizaje usando una codificación interna entre la relación de las entradas-salidas y diversos planes con el objetivo de diversificar determinadas subredes internas. El algoritmo propuesto por Chella et al. es una aproximación basada en una función *minmax* y en una configuración de los métodos de descenso de gradiente y quasi-Newton. El punto óptimo se alcanza minimizando la función de error de la red sin ningún ajuste de los parámetros internos. Posteriormente, Ihm y Park [70] presentaron un nuevo y veloz algoritmo para evitar la lenta convergencia debida a la oscilación de los pesos en la superficie de error cerca de los valles. Para evitar este problema, derivaron un nuevo término de gradiente modificando el término original con una dirección estimada y descendente en los valles. Wilamowski et al. [138] desarrollaron dos modificaciones del algoritmo de Levenberg-Marquardt. La primera de ellas se realiza sobre el índice de rendimiento y la segunda sobre el cálculo del gradiente.

- *Reescalado de las variables.* Como ya se ha mencionado previamente, el cálculo de la señal de error involucra la derivada de la función neuronal la cual es multiplicada en cada capa. Puesto que en el caso de funciones neuronales sigmoidales esta derivada tiene un valor entre 0 y 1/4, los elementos de la matriz jacobiana pueden diferir bastante en amplitud para cada una de las capas. Esta es una de las causas que provocan muchos de los problemas del método de retropropagación del error. Para solucionar este problema Riegler et al. han propuesto un método basado en el reescalado de estos elementos [115].

Aunque las técnicas descritas han sido aplicadas con éxito para acelerar la convergencia de algunos problemas, no hay suficientes experimentos y discusión sobre sus capacidades para evitar mínimos locales. Además, la mayoría de los métodos introducen parámetros adicionales en el entrenamiento que son dependientes del problema [67].

Capítulo 3

Algoritmo de óptimo global para redes de neuronas de una capa

"*Keep it simple: as simple as possible, but no simpler*"

Albert Einstein

3.1 Introducción

Las redes de neuronas de una capa, i.e., sin capas ocultas, han sido estudiadas extensamente en los años 60, y la historia de éstas ha sido revisada recientemente por Widrow y Lehr [137]. Para una red de una capa con funciones de activación lineales, es posible calcular los pesos óptimos que minimizan la función de error, basada en el error cuadrático medio, empleando simplemente la pseudo inversa de una matriz [18]. Sin embargo, este resultado sólo es posible para el caso de redes lineales empleando como función de error el error cuadrático medio. Si se usan funciones de activación no lineales, como las sigmoidales o las tangentes hiperbólicas, u otra función de error diferente, esta solución ya no es posible. Sin embargo, si la función de activación es diferenciable, como es el caso de las funciones mencionadas previamente, se pueden evaluar fácilmente las derivadas parciales de la función de error con respecto a cada uno de los pesos. Estas derivadas pueden usarse en diversos algoritmos de optimización basados en gradiente para encontrar el mínimo de la función de error. De cualquier forma, el rendimiento de estos algoritmos depende de un número de elecciones durante el diseño de la red (como por ejemplo, el paso de aprendizaje y el término de momentum) la mayoría de los cuales se fijan empleando valores estándar o el método de prueba y error. Además de este problema, es posible que los algoritmos de minimización, para este tipo de redes, queden atrapados en un

mínimo local como ha sido demostrado en diversos estudios [21][22]. En uno se estos estudios, el de Sontag y Sussmann [122], se demuestra este hecho matemáticamente y se presenta también un ejemplo en el cual una red de una capa, con funciones de activación sigmoidales y empleando el error cuadrático medio, contiene un mínimo local que no es el mínimo global. Además, llegaron a la conclusión de que la existencia de mínimos locales se debe al hecho de que la función de error es la superposición de funciones cuyos mínimos están en diferentes puntos. En el caso de funciones de activación lineales, todas las funciones son convexas, y por tanto no hay problema con los mínimos locales, pues la suma de funciones convexas da como resultado una nueva función convexa. Sin embargo, cuando se emplean funciones del tipo sigmoidal como funciones de activación, la suma de éstas en general es no convexa y, por consiguiente, no está garantizado que exista un único mínimo en la función. Además, Auer et al. [2] demostraron que el número de mínimos locales puede crecer exponencialmente con la dimensión de la entrada de la red (d). En particular, los autores probaron que para el error cuadrático medio y funciones de activación de tipo sigmoidal, la función de error de una red de una capa con una única neurona y n ejemplos de entrenamiento puede tener $\lfloor n/d \rfloor^d$ mínimos locales. Además, este hecho es válido también para cualquier función de error y función de activación cuya composición sea continua y tenga un rango definido.

La ausencia de mínimos locales bajo ciertas condiciones de separabilidad o independencia lineal, o modificaciones del error cuadrático medio han sido probadas [22][53]. Sin embargo, el problema de caracterizar estos extremos para el error cuadrático medio estándar, usando sólo un número finito de entradas no ha sido resuelto, ni hay un conocimiento claro acerca de las condiciones que causan la aparición de estos mínimos locales. Coetzee y Stonick [34] propusieron un método para evaluar a posteriori si un conjunto de pesos es la solución única o globalmente óptima para un determinado problema. Aunque los resultados presentados en este trabajo son de gran interés y potencialmente útiles para evaluar la optimalidad y unicidad de una solución, sólo es posible caracterizar los mínimos locales después de que el entrenamiento de la red haya finalizado.

3.2 Motivación

La estructura básica de una red de neuronas de una capa se muestra en la figura 3.1. Esta red está formada por un conjunto de J neuronas de salida que contienen como funciones de activación las funciones invertibles $f_1, f_2, \ldots, f_J$. En este tipo de red el conjunto de ecuaciones que relacionan las entradas (x_{is}) y las salidas (y_{js}) es

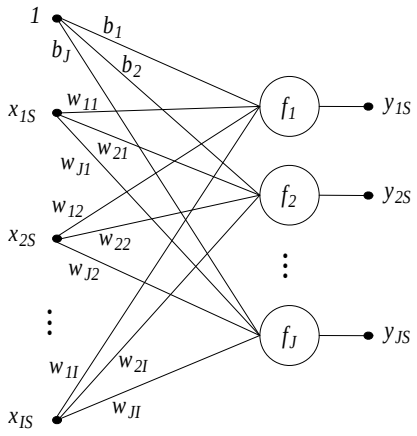


Figura 3.1: Red de neuronas de una capa con diferentes funciones neuronales.

el siguiente:

$$y_{js} = f_j\left(b_j + \sum_{i=1}^{I} w_{ji}x_{is}\right);\ \ j = 1, 2, \ldots, J;\ \ s = 1, 2, \ldots, S, \tag{3.1}$$

donde b_j y $w_{ji}; i = 1, 2, \ldots, I$ es el valor del umbral y de los pesos, respectivamente, asociados con la neurona j ($j = 1, 2, \ldots, J$) y S es el número de datos en el conjunto de entrenamiento. El sistema definido en (3.1) contiene $J \times S$ ecuaciones y $J \times (I + 1)$ incógnitas. Sin embargo, puesto que el número de datos (S) habitualmente es grande ($S >> I + 1$) en la práctica este conjunto de ecuaciones no es compatible, y consecuentemente, no tiene solución. Por lo tanto, es necesario considerar ciertos errores, ε_{js}, y transformar la ecuación (3.1) en

$$\varepsilon_{js} = d_{js} - y_{js} = d_{js} - f_j\left(b_j + \sum_{i=1}^{I} w_{ji}x_{is}\right);\ \ j = 1, 2, \ldots, J;\ \ s = 1, 2, \ldots, S, \tag{3.2}$$

donde d_{js} es el valor de la salida deseada para la neurona j y el dato s.

Para estimar los pesos se minimiza, usualmente, el error cuadrático (EC) definido como:

$$C_1 = \sum_{s=1}^{S}\sum_{j=1}^{J} \varepsilon_{js}^2 = \sum_{s=1}^{S}\sum_{j=1}^{J}\left(d_{js} - f_j\left(b_j + \sum_{i=1}^{I} w_{ji}x_{is}\right)\right)^2. \tag{3.3}$$

Es importante resaltar que debido a la presencia de las funciones neuronales no lineales, f_j, la función mostrada en (3.3) no es lineal ni cuadrática en los pesos w_{ji} y en los umbrales b_j. Por tanto, C_1 puede tener, además del mínimo global, diversos mínimos locales, como ya se ha mencionado en la introducción.

Los métodos actuales empleados para el aprendizaje de los pesos en las redes de neuronas tienen los siguientes inconvenientes:

1. La función a optimizar tiene diversos óptimos locales. Esto implica que el usuario no puede saber si después del aprendizaje ha alcanzado el óptimo global y cuál es la distancia a éste. Por tanto, después de varios entrenamientos es frecuente alcanzar diferentes soluciones, dependiendo del conjunto inicial de pesos empleado.

2. El proceso de aprendizaje requiere una gran cantidad de recursos computacionales comparado con otros modelos.

3.3 Método para el aprendizaje de los pesos basado en un sistema lineal de ecuaciones

Para evitar los problemas indicados anteriormente, en esta sección se propone un nuevo método para el aprendizaje de los pesos en redes de neuronas de una capa. Este método está basado en el siguiente resultado:

Teorema 1 *Sean* $\mathbf{d}, \mathbf{y} \in \mathbb{R}^J$ *la salida deseada y real de la red,* $\mathbf{W} \in \mathbb{R}^{J \times I}$, $\mathbf{b} \in \mathbb{R}^{J \times 1}$ *los pesos y umbrales, y* $\mathbf{f}, \mathbf{f^{-1}}, \mathbf{f'} : \mathbb{R}^J \rightarrow \mathbb{R}^J$ *la función no lineal, su inversa y su derivada. La minimización del error cuadrático medio (ECM) entre* $\mathbf{d}$ *e* $\mathbf{y}$ *en la salida de la no linealidad es equivalente (hasta primer orden) a minimizar el ECM antes de la no linealidad, i.e., entre* $\mathbf{z} = \mathbf{W}\mathbf{x} + \mathbf{b}$ *y* $\bar{\mathbf{d}} = \mathbf{f^{-1}}(\mathbf{d})$*, donde la función inversa se evalúa para cada salida independientemente. En este último caso, cada error debe ser ponderado empleando el valor de la derivada de la no linealidad en el valor correspondiente. Matemáticamente, esta propiedad puede ser definida como:*

$$\min_{\mathbf{W},\mathbf{b}} E[(\mathbf{d}-\mathbf{y})^{\mathbf{T}}(\mathbf{d}-\mathbf{y})] \approx \min_{\mathbf{W},\mathbf{b}} \mathbf{E}[(\mathbf{f'}(\bar{\mathbf{d}}).*\bar{\boldsymbol{\varepsilon}})^{\mathbf{T}}(\mathbf{f'}(\bar{\mathbf{d}}).*\bar{\boldsymbol{\varepsilon}})] \tag{3.4}$$

donde '.' representa el producto de dos vectores componente a componente* $\mathbf{f'}(\bar{\mathbf{d}})$ *y* $\bar{\boldsymbol{\varepsilon}} = \bar{\mathbf{d}} - \mathbf{z}$.

Demostración:
Usando las ecuaciones $\mathbf{y} = \mathbf{f}(\mathbf{z})$ y $\mathbf{d} = \mathbf{f}(\bar{\mathbf{d}})$ se obtiene el siguiente problema de minimización equivalente:

$$\min_{\mathbf{W},\mathbf{b}} E[(\mathbf{d}-\mathbf{y})^T(\mathbf{d}-\mathbf{y})] = \min_{\mathbf{W},\mathbf{b}} E[(\mathbf{f}(\bar{\mathbf{d}})-\mathbf{f}(\mathbf{z}))^T(\mathbf{f}(\bar{\mathbf{d}})-\mathbf{f}(\mathbf{z}))] \tag{3.5}$$

Sea $\bar{\mathbf{d}} = \mathbf{f^{-1}}(\mathbf{d})$ la salida deseada antes de la no linealidad entonces podemos definir el error antes de la función neuronal como $\bar{\boldsymbol{\varepsilon}} = \bar{\mathbf{d}} - \mathbf{z}$. Si la variabilidad de este error ($var(\|\bar{\boldsymbol{\varepsilon}}\|)$) es pequeña entonces se puede emplear la siguiente aproximación,

basada en una serie de Taylor de primer orden, en cada componente del vector de salida:

$$\mathbf{f}(\mathbf{z}) = \mathbf{f}(\bar{\mathbf{d}} - \bar{\boldsymbol{\varepsilon}}) \approx \mathbf{f}(\bar{\mathbf{d}}) - \mathbf{f}'(\bar{\mathbf{d}}) .* \bar{\boldsymbol{\varepsilon}} \tag{3.6}$$

Sustituyendo esta última ecuación en (3.5) se obtiene:

$$\min_{\mathbf{W},\mathbf{b}} E[(\mathbf{d}-\mathbf{y})^T(\mathbf{d}-\mathbf{y})] \approx \min_{\mathbf{W},\mathbf{b}} E[(\mathbf{f}'(\bar{\mathbf{d}}) .* \bar{\boldsymbol{\varepsilon}})^T(\mathbf{f}(\bar{\mathbf{d}}) .* \bar{\boldsymbol{\varepsilon}})] \tag{3.7}$$

■

El teorema anterior es un resultado muy importante pues establece que la minimización del error cuadrático medio en la salida de la red, i.e., después de las funciones no lineales, es equivalente (hasta primer orden) a la minimización del error antes de las funciones neuronales. Por tanto, es posible emplear, alternativamente, cualquiera de las funciones de error mostradas en (3.4) para el aprendizaje de la red. Este resultado es la base del método propuesto en este trabajo. Por tanto, si se usa el problema de minimización definido en la parte derecha de la ecuación (3.4), el sistema de ecuaciones (3.2), el cual mide los errores en la salida de la red (unidades de y_{js}), puede reescribirse de la siguiente forma:

$$\bar{\varepsilon}_{js} = \bar{d}_{js} - z_{js} = f_j^{-1}(d_{js}) - b_j - \sum_{i=1}^{I} w_{ji}x_{is}; \quad s = 1, 2, \ldots, S; \;\; j = 1, 2, \ldots, J, \tag{3.8}$$

que mide los errores en términos de las entradas (unidades de x_{is}). Es importante destacar que en (3.8), a diferencia de lo que sucede en (3.2), los pesos no están dentro de la función de activación (f_j) y, por tanto, el error es lineal respecto a los parámetros de la red (pesos y umbral).

Puesto que para cada j el umbral y los pesos, b_j y $w_{ji}; i = 1, 2, \ldots, I$, están relacionados sólo con la salida y_{js}, y por tanto sólo aparecen en S de las $J \times S$ ecuaciones en (3.8), es evidente que el problema del aprendizaje puede dividirse en J problemas independientes (uno para cada j). Esto significa que la red de neuronas de la figura 3.1 puede ser dividida en J redes más simples, como se muestra en la figura 3.2. Por tanto, a continuación se abordará el problema para uno de los j subproblemas.

En este trabajo se proponen dos métodos alternativos, basados en el resultado del teorema 1, para el aprendizaje de los pesos:

Opción 1: Minimizar la suma de los errores al cuadrado:

$$C_{2j} = \sum_{s=1}^{S} \left(f_j'(\bar{d}_{js})\bar{\varepsilon}_{js}\right)^2 \;\; = \;\; \sum_{s=1}^{S} \left(f_j'(\bar{d}_{js}) \left(f_j^{-1}(d_{js}) - b_j - \sum_{i=1}^{I} w_{ji}x_{is} \right) \right)^2, \tag{3.9}$$

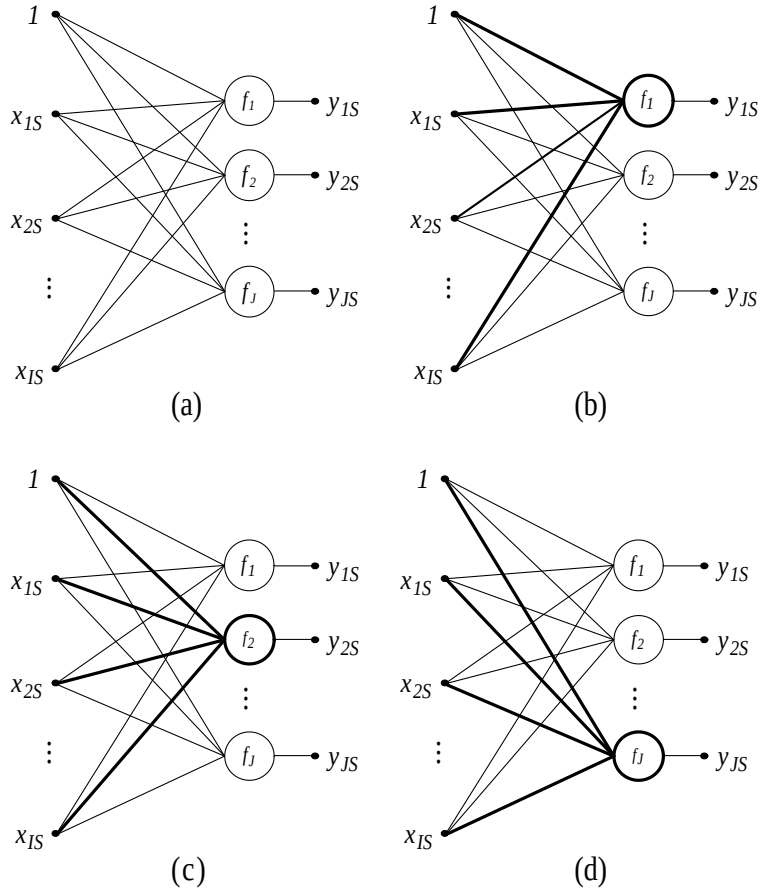


Figura 3.2: (a) Red de neuronas inicial, (b), (c) y (d) las J redes de neuronas independientes.

que conduce al siguiente sistema de ecuaciones lineales:

$$\begin{aligned}
&\frac{\partial C_{2j}}{\partial w_{jp}} = -2\sum_{s=1}^{S}\left(f_j'(\bar{d}_{js})\left(f_j^{-1}(d_{js}) - b_j - \sum_{i=1}^{I} w_{ji}x_{is}\right)\right)x_{ps}f_j'(\bar{d}_{js}) = 0;\\
&p = 1, 2, \ldots, I,\\
&\frac{\partial C_{2j}}{\partial b_j} = -2\sum_{s=1}^{S}\left(f_j'(\bar{d}_{js})\left(f_j^{-1}(d_{js}) - b_j - \sum_{i=1}^{I} w_{ji}x_{is}\right)\right)f_j'(\bar{d}_{js}) = 0,
\end{aligned} \tag{3.10}$$

que puede reescribirse como

$$\left.\begin{array}{l}
\sum_{s=1}^{S} f_j'^2(\bar{d}_{js})x_{ps}b_j + \sum_{i=1}^{I} w_{ji}\sum_{s=1}^{S} f_j'^2(\bar{d}_{js})x_{is}x_{ps} =\\
\sum_{s=1}^{S} f_j'^2(\bar{d}_{js})f_j^{-1}(d_{js})x_{ps};\ p = 1, 2, \ldots, I,\\
\sum_{s=1}^{S} f_j'^2(\bar{d}_{js})b_j + \sum_{i=1}^{I} w_{ji}\sum_{s=1}^{S} f_j'^2(\bar{d}_{js})x_{is} = \sum_{s=1}^{S} f_j'^2(\bar{d}_{js})f_j^{-1}(d_{js})
\end{array}\right\} \tag{3.11}$$

Este sistema de ecuaciones contiene $I+1$ ecuaciones e incógnitas y, por lo tanto, existe una solución única que se corresponde con el óptimo global de la función (salvo casos degenerados).

Opción 2: Alternativamente, se puede minimizar el máximo error absoluto, i.e.:

$$\min_{\mathbf{w},\mathbf{b}}\left\{\max_{s}\left|f_j'(\bar{d}_{js})\left(f_j^{-1}(d_{js}) - b_j - \sum_{i=1}^{I} w_{ji}x_{is}\right)\right|\right\}, \tag{3.12}$$

que puede enunciarse como el siguiente problema de programación lineal:

$$\text{Minimizar } \epsilon$$

sujeto a

$$\left.\begin{array}{rcl}
-f_j'(\bar{d}_{js})b_j - f_j'(\bar{d}_{js})\sum_{i=1}^{I} w_{ji}x_{is} - \epsilon &\leq& -f_j'(\bar{d}_{js})f_j^{-1}(d_{js})\\
f_j'(\bar{d}_{js})b_j + f_j'(\bar{d}_{js})\sum_{i=1}^{I} w_{ji}x_{is} - \epsilon &\leq& f_j'(\bar{d}_{js})f_j^{-1}(d_{js})
\end{array}\right\};\ \ s = 1, 2, \ldots, S \tag{3.13}$$

cuyo óptimo global puede obtenerse fácilmente mediante técnicas de programación lineal. Además, es sencillo encontrar el conjunto de todas las posibles soluciones que conllevan al óptimo global. Esto se puede realizar verificando qué restricciones en (3.13) están activas, y solucionando los correspondientes sistemas de ecuaciones lineales para encontrar los puntos extremos del poliedro resultante.

Consecuentemente, con los métodos propuestos (opción 1 y 2) se obtiene el óptimo global resolviendo un sistema de ecuaciones lineales o bien un problema de programación lineal. Estos métodos requieren muchos menos recursos computacionales que las aproximaciones empleadas actualmente para minimizar C_1 en (3.3).

3.3.1 Aprendizaje de las funciones neuronales

A continuación, se propone una mejora de los métodos presentados en la sección anterior consistente en el aprendizaje de la inversa de las funciones neuronales f_j^{-1} en vez de suponerlas conocidas a priori. En concreto, las funciones neuronales pueden ser consideradas como una combinación convexa y lineal de un conjunto

$$\{\phi_1(x), \phi_2(x), , \ldots, \phi_R(x)\}$$

de funciones básicas e invertibles, i.e.:

$$f_j^{-1}(x) = \sum_{r=1}^{R} \alpha_{jr}\phi_r(x);\ j = 1, 2, \ldots, J, \tag{3.14}$$

donde $\{\alpha_{jr};\ r = 1, 2, \ldots, R\}$ es el conjunto de coeficientes asociados con la función f_j^{-1}, que deben ser elegidos de forma que la función resultante f_j^{-1} sea invertible. Además, sin pérdida de generalidad, se puede asumir que la función neuronal es creciente.

Por tanto, como en el caso anterior, se proponen dos opciones:

Opción 1: Minimizar, respecto a b_j, $w_{ji}; i = 1, \ldots, I$ y $\alpha_{jr}; r = 1, 2, \ldots, R$, la función

$$C_{2j} = \sum_{s=1}^{S} \bar{\epsilon}_{js}^2 \quad = \quad \sum_{s=1}^{S} \left(\sum_{r=1}^{R} \alpha_{jr}\phi_r(d_{js}) - b_j - \sum_{i=1}^{I} w_{ji}x_{is} \right)^2, \tag{3.15}$$

sujeto a

$$\sum_{r=1}^{R} \alpha_{jr}\phi_r(d_{js_1}) \leq \sum_{r=1}^{R} \alpha_{jr}\phi_r(d_{js_2});\ \ \forall d_{js_1} < d_{js_2},$$

que fuerza a que la función candidata f_j^{-1} sea creciente, al menos en los puntos considerados.

Opción 2: Alternativamente, para cada $j = 1, 2, \ldots, J$, se puede minimizar el máximo error en valor absoluto, i.e.:

$$\min_{w,\alpha} \left\{ \max_{s} \left| b_j + \sum_{i=1}^{I} w_{ji}x_{is} - \sum_{r=1}^{R} \alpha_{jr}\phi_r(d_{js}) \right| \right\}, \tag{3.16}$$

que puede ser planteado como el siguiente problema de programación lineal:

$$\text{Minimizar } \epsilon$$

sujeto a

$$\begin{aligned} b_j + \sum_{i=1}^{I} w_{ji}x_{is} - \sum_{r=1}^{R} \alpha_{jr}\phi_r(d_{js}) - \epsilon &\leq 0 \\ -b_j - \sum_{i=1}^{I} w_{ji}x_{is} + \sum_{r=1}^{R} \alpha_{jr}\phi_r(d_{js}) - \epsilon &\leq 0 \\ \sum_{r=1}^{R} \alpha_{jr}\phi_r(d_{js_1}) &\leq \sum_{r=1}^{R} \alpha_{jr}\phi_r(d_{js_2});\ \ \forall d_{js_1} < d_{js_2} \\ \sum_{r=1}^{R} \alpha_{jr}\phi_r(d_0) &= 1, \end{aligned} \tag{3.17}$$

el cual tiene un óptimo global que puede obtenerse fácilmente.

Para evitar transformaciones con derivadas cuyo valor sea alto es conveniente limitar la primera derivada de la transformación $f_j^{-1}(y_{js})$. Esto se puede realizar, al menos en los puntos considerados, añadiendo las siguientes restricciones adicionales

$$\frac{df_j^{-1}(y)}{dy} = \sum_{r=1}^{R} \alpha_{jr}\phi_r'(y) \geq \beta;\ j = 1, 2, \ldots, J, \tag{3.18}$$

donde β es el límite superior de la derivada de la función inversa que es igual a la inversa de la derivada de la función neuronal f_j.

Una vez obtenida la función f_j^{-1}, para usar la red es necesario trabajar con su inversa, que puede obtenerse empleando, por ejemplo, el método de la bisección. Para evitar problemas con las dimensiones de los datos, es recomendable que éstos sean transformados al hipercubo unitario.

Finalmente, es importante resaltar que en este caso se ha eliminado, en ambas funciones de coste, el término $f_j'(\bar{d}_{js})$ que ponderaba el error de cada dato, $\bar{\epsilon}_{js}$, antes de la no linealidad en las ecuaciones (3.9) y (3.12). Se ha suprimido este término porque en el caso del aprendizaje de funciones no es posible calcularlo durante el aprendizaje. Esto es una consecuencia de que sólo se conoce que la función inversa f_j^{-1} es la combinación lineal de un conjunto de funciones básicas (ecuación (3.14)), y por tanto no es posible calcular su inversa durante el entrenamiento para obtener la expresión de la función neuronal f_j. La supresión de este término produce, en general, una aproximación menos precisa. Sin embargo, como se mostrará posteriormente en las simulaciones realizadas, esta pérdida a veces es inferior a la ganancia producida por el aprendizaje de funciones y, por consiguiente, se produce una mejora en el rendimiento global del sistema.

3.4 Método de aprendizaje alternativo. Estudio de variabilidad

Si se escribe la ecuación (3.1) como

$$b_j + \sum_{i=1}^{I} w_{ji} x_{is} = f_j^{-1}(y_{js}); \quad s = 1, 2, \ldots, S, \tag{3.19}$$

puesto que el número de incógnitas es $(I + 1)$, dado un conjunto con $(I + 1)$ datos es suficiente (si no es un sistema lineal degenerado) para el aprendizaje de los parámetros b_j y $\{w_{ij}; i = 1, \ldots, I\}$. Por tanto, la alternativa propuesta consiste en el aprendizaje de los pesos con subconjuntos de datos (elegidos de forma determinista o aleatoria) conteniendo cada uno $v \geq (I + 1)$ ejemplos de entrenamiento y determinando la variabilidad de los diferentes pesos obtenidos para cada subconjunto. En otras palabras, el algoritmo sería el siguiente:

Repetir los pasos 1 y 2 que se describen a continuación, un número suficiente de veces, esto es, desde $r = 1$ a $r = m$ con un valor de m alto, y posteriormente proceder con los pasos 3 y 4.

Paso 1: Seleccionar, aleatoriamente, un subconjunto D_r de $v \geq (I + 1)$ ejemplos a partir del conjunto inicial de S datos.

Paso 2: Usar el sistema de ecuaciones (3.11) para obtener el vector de pesos y umbral correspondientes $\{w_{ji}^r; i = 1, \ldots, I\}$ y $\{b_j^r\}$.

Paso 3: Calcular las medias μ_{ji}, medianas M_{ji} y desviaciones típicas σ_{ji} de los conjuntos $\{w_{ji}^r\}; \forall i$. Calcular la media μ'_j, mediana M'_j y desviación típica σ'_j del conjunto $\{b_j^r\}$.

Paso 4: Devolver $\hat{w}_{ji} = \mu_{ji}$ y $\hat{b}_j = \mu'_j$, o alternativamente $\hat{w}_{ji} = M_{ji}$ y $\hat{b}_j = M'_j$, como los estimadores de w_{ji} y b_j, respectivamente. Además σ_{ji} y σ'_j proporcionan una medida de precisión de la estimación de w_{ji} y b_j.

3.5 Ejemplos de aplicación

En esta sección se demuestra el buen funcionamiento de los métodos propuestos en las secciones anteriores mediante su aplicación a conjuntos de datos estándar en el campo de las redes de neuronas. Los dos primeros casos (secciones 3.5.1 y 3.5.2) son problemas de clasificación, y se han empleado para comprobar la validez de los métodos presentados en las secciones 3.3 y 3.4. Los dos últimos ejemplos

(secciones 3.5.3 y 3.5.4) son problemas de regresión o aproximación de funciones y se han empleado para comparar el método propuesto en la sección 3.3 y el método que incorpora el aprendizaje de las funciones neuronales, presentado en la sección 3.3.1. Para estimar el error verdadero de las redes de neuronas se realizó en todos los casos, a excepción del ejemplo 3.5.3, una validación cruzada *leaving-one-out* [135]. En este tipo de validación dado un conjunto de datos formado por S elementos se realizarán S entrenamientos diferentes de la red empleando en cada uno de ellos un dato para prueba, distinto en cada simulación, y los $S-1$ datos restantes para el entrenamiento. Finalmente, la sección 3.5.5 contiene un estudio comparativo de la velocidad de convergencia entre los métodos propuestos en la sección 3.3 y varios algoritmos de alto rendimiento.

3.5.1 Ejemplo con los datos del iris de Fisher

El primer experimento fue realizado con el conjunto de datos de la planta de iris propuesto por Fisher en 1936 [47]. Esta es una de las bases de datos más conocidas en la literatura de reconocimiento de patrones. El artículo de Fisher es un clásico en este campo y es utilizado y referenciado frecuentemente. El conjunto de datos contiene tres clases cada una de ellas compuesta por 50 ejemplares, y perteneciente a un tipo de planta de iris, a saber: Setosa, Versicolour y Virginica. Por tanto, el problema consiste en clasificar, en base a cuatro atributos, la clase a la que pertenece cada uno de los ejemplares de la base de datos. La información de los atributos consiste en:

1. longitud del sépalo en cm.

2. anchura del sépalo en cm.

3. longitud del pétalo en cm.

4. anchura del pétalo en cm.

En este ejemplo de clasificación, la primera de las clases (Setosa) es linealmente separable de las otras dos (Versicolour y Virginica), que no lo son entre sí.

Para este ejemplo se empleó la red de neuronas descrita en la figura 3.1 de la sección 3.2, donde $I = 4$ y $J = 1$, y el método de aprendizaje (3.13) presentado en la sección 3.3 y que consiste en la minimización del máximo valor absoluto. Además, a cada una de las clases se le asignó un valor numérico entre 1 y 3. La función neuronal empleada fue la función tangente, cuya inversa se define como

$$f^{-1}(x) = \arctan(x).$$

Tabla 3.1: Pesos obtenidos para los datos del iris, y desviaciones típicas obtenidas empleando (3.13).

Parámetro	Valor	Media	Mediana	Desviación típica
b_1	0.9705	0.9642	0.9567	0.0614
w_{11}	−0.0274	−0.0263	−0.0262	0.0149
w_{12}	−0.0499	−0.0495	−0.0500	0.0170
w_{13}	0.0723	0.0713	0.0717	0.0128
w_{14}	0.0979	0.0996	0.0985	0.0199

Esta función neuronal es la que se empleó también en el siguiente ejemplo. El error cuadrático medio alcanzado fue 0.0337. Los pesos obtenidos después del entrenamiento se muestran en la segunda columna de la tabla 3.1.

Un vez finalizado el entrenamiento de la red se fijaron los umbrales de clasificación óptimos para determinar los límites entre cada una de las clases. La regla de clasificación resultante es la siguiente:

1. Si $\tan(b_j + \sum_{i=1}^{4} w_{ji}x_{is}) \leq 1.4$, la planta s se clasificará en la clase Setosa.

2. Si $1.4 < \tan(b_j + \sum_{i=1}^{4} w_{ji}x_{is}) \leq 2.365$, la planta s se clasificará como Versicolour.

3. Si $2.365 < \tan(b_j + \sum_{i=1}^{4} w_{ji}x_{is})$, la planta s se clasificará como Virginica.

La gráfica presentada en la figura 3.3 muestra el valor de la salida para cada uno de los 150 ejemplos del conjunto de prueba empleados en la validación *leaving-one-out*. En esta gráfica se muestran además los valores que discriminan entre las tres clases representados por las líneas discontinuas. Con este criterio, se consiguió clasificar correctamente el 98.67% de las plantas. La matriz de contingencia en la tabla 3.2 muestra el número de casos clasificados erróneamente. Como se puede observar, el sistema clasifica correctamente todos los ejemplares correspondientes a la clase Setosa y comete sólo dos errores entre las clases que no son linealmente separables, i.e., Versicolour y Virginica.

Finalmente, también se empleó el método alternativo de aprendizaje descrito en la sección 3.4, consistente en el estudio de variabilidad. Para ello, se emplearon 100 subconjuntos de 50 datos seleccionados aleatoriamente. La media, mediana y desviación típica para cada uno de los pesos $\{w_{i1}; i = 1, 2, 3, 4\}$ y umbral b_1 se muestran en la tabla 3.1. Como se puede observar, los diferentes estimadores de los pesos son muy similares y además los valores de las desviaciones típicas sugieren que el modelo

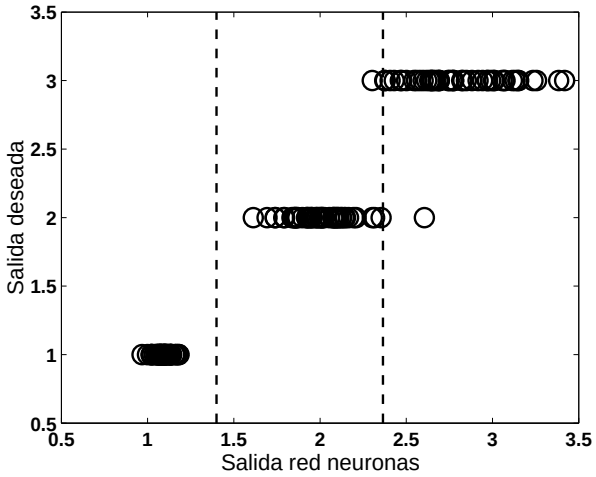


Figura 3.3: Salida de la red de neuronas frente a la salida real para los datos del iris.

Tabla 3.2: Matriz de contingencia para los datos del iris.

		Clasificación real		
		Setosa	Versicolour	Virginica
Red de neuronas	Setosa	50	0	0
	Versicolour	0	49	1
	Virginica	0	1	49

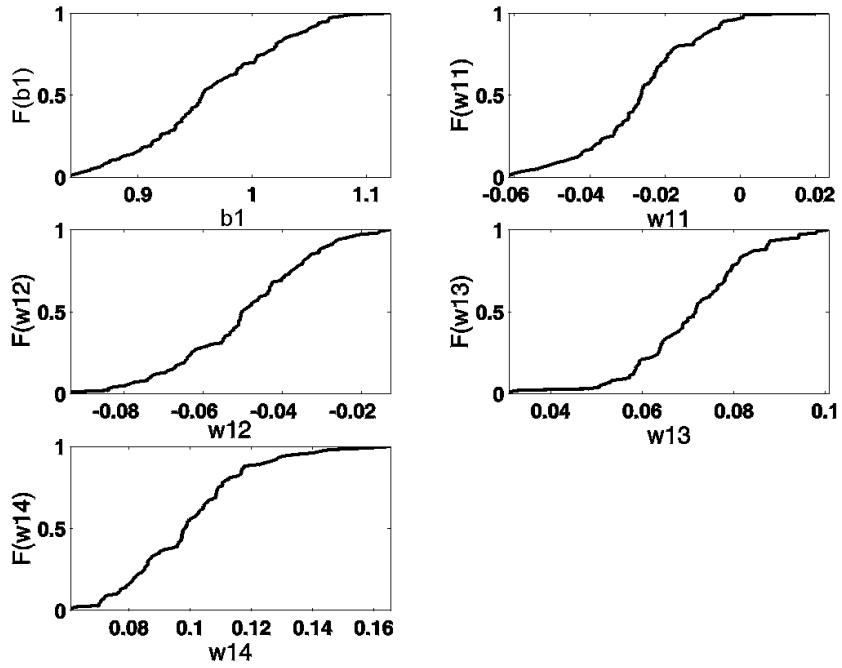


Figura 3.4: Funciones de distribución acumulativas de cada uno de los pesos empleando los datos del iris.

es adecuado para el conjunto de datos del iris. Finalmente, la figura 3.4 contiene la distribución acumulativa empírica obtenida para cada peso.

3.5.2 Ejemplo con los datos de cáncer de mama

El segundo experimento se llevó a cabo empleando la base de datos de cáncer de mama proporcionada por los hospitales de la Universidad de Wisconsin [10]. Esta base de datos contiene 699 casos de pacientes reales, de los cuales 458 se corresponden con un cáncer de tipo benigno (65.5%) y 241 con uno de tipo maligno (34.5%). De esta colección de datos, 16 casos contienen información incompleta. Por lo tanto, estos ejemplos fueron eliminados del conjunto de entrenamiento que contó, finalmente, con 683 datos. El problema consiste en clasificar el cáncer en uno de los dos tipos posibles (0 para benigno y 1 para maligno) a partir de 9 atributos, mostrados en la tabla 3.3.

La gráfica en la figura 3.5 muestra el valor de la salida de la red de neuronas para cada uno de los 683 ejemplos del conjunto de prueba generados mediante la validación *leaving-one-out*. La línea de puntos mostrada en esta figura representa el valor discriminante entre las dos clases.

El criterio de clasificación resultante es el siguiente:

Tabla 3.3: Atributos empleados para la clasificación del cáncer de mama.

Atributo	Rango de valores
Espesor de la masa	1 - 10
Uniformidad del tamaño de las células	1 - 10
Uniformidad de la forma de las células	1 - 10
Adhesión marginal	1 - 10
Tamaño de la célula epitelial	1 - 10
Fragilidad del núcleo	1 - 10
Dureza de la cromatina	1 - 10
Nucleolo normal	1 - 10
Mitosis	1 - 10

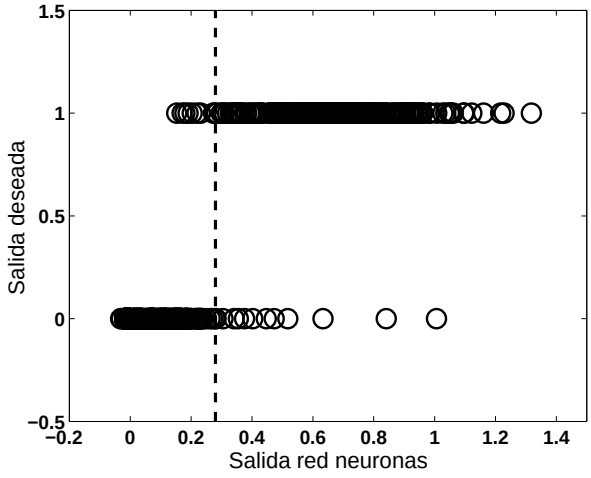


Figura 3.5: Salida de la red de neuronas frente a la salida real para los datos del cáncer de mama.

Tabla 3.4: Matriz de contingencia para los datos de cáncer de mama.

		Clasificación real	
		Benigno	Maligno
Red de neuronas	Benigno	432	9
	Maligno	12	230

Tabla 3.5: Pesos estimados para los datos de cáncer de mama, y las desviaciones típicas asociadas usando (3.10).

Parámetro	Valor	Media	Mediana	Desviación típica
b_1	1.0530	1.0415	1.0411	0.0128
w_{11}	0.0069	0.0112	0.0109	0.0030
w_{12}	0.0048	−0.0028	−0.0015	0.0074
w_{13}	0.0034	0.0089	0.0074	0.0070
w_{14}	0.0018	0.0036	0.0030	0.0041
w_{15}	0.0022	0.0031	0.0029	0.0047
w_{16}	0.0099	0.0089	0.0089	0.0035
w_{17}	0.0042	0.0025	0.0025	0.0047
w_{18}	0.0041	0.0060	0.0059	0.0043
w_{19}	0.0002	0.0033	0.0035	0.0054

1. Si $\tan(b_j + \sum_{i=1}^{9} w_{ji}x_{is}) \leq 0.28$, el cáncer s se clasifica como maligno.

2. Si $\tan(b_j + \sum_{i=1}^{9} w_{ji}x_{is}) > 0.28$, el cáncer s se clasifica como benigno.

Con este criterio el número de casos clasificados correctamente es del 96.92%. La tabla de contingencia de la tabla 3.4 muestra el número de casos clasificados correcta e incorrectamente para cada una de las clases. Empleando el método descrito en las sección 3.4, se obtuvieron los estimadores de los pesos mostrados en la tabla 3.5. Para ello, se emplearon 100 subconjuntos con 300 datos cada uno obtenidos aleatoriamente. La segunda columna de esta tabla muestra el valor de los pesos, después del entrenamiento, en una de las simulaciones realizadas con el método de la sección 3.3. La figura 3.6 contiene las funciones de distribución acumulativas obtenidas empíricamente para cada uno de los pesos. Como se puede observar los estimadores obtenidos son muy similares.

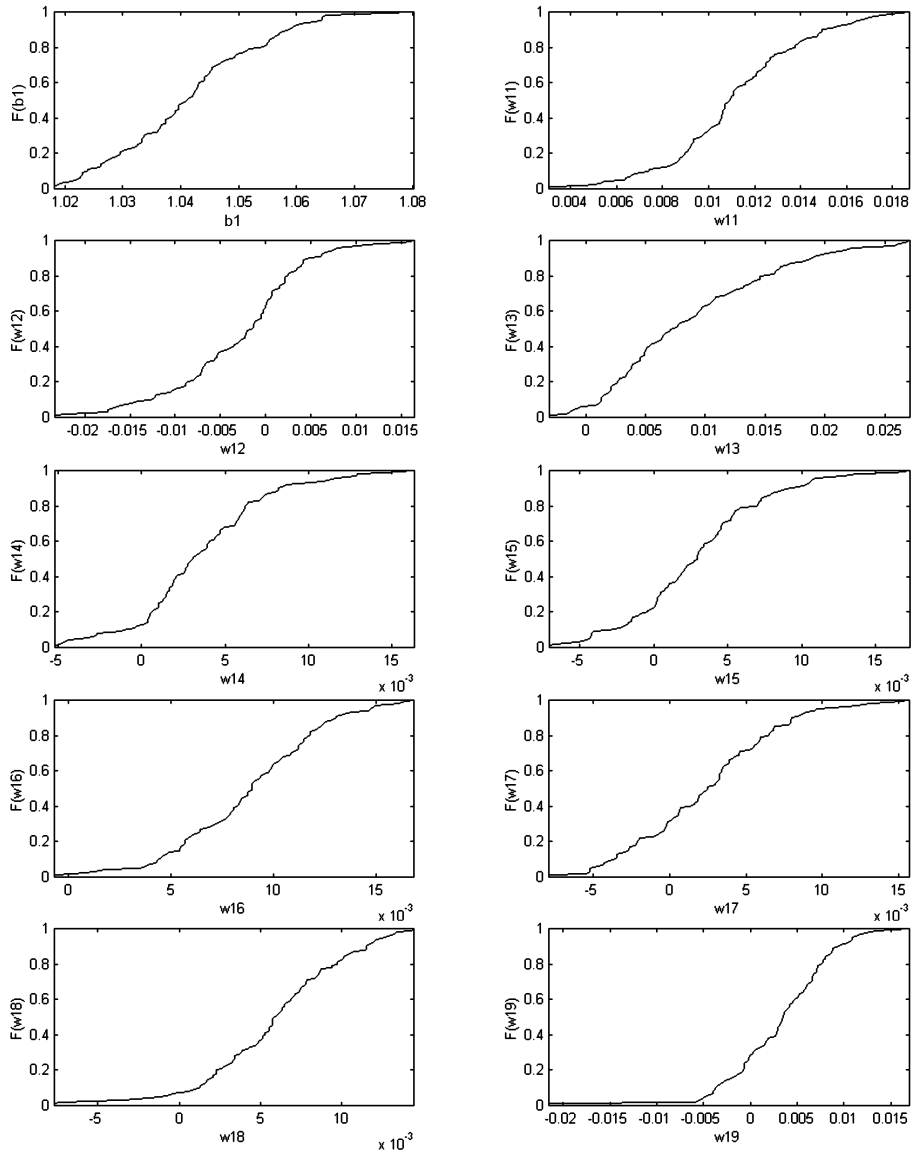


Figura 3.6: Funciones de distribución acumulativas de cada uno de los pesos empleando los datos del cáncer de mama.

3.5.3 Modelado de una función no lineal de tres variables

En este ejemplo, el problema consiste en modelar la siguiente función no lineal

$$y = z^2 + z + sen(z),$$

donde

$$z = 3x_1 + 2x_2 - x_3.$$

Para ello, se empleó el método basado en el sistema de ecuaciones (3.11) de la sección 3.3 y se comparó su rendimiento con el método con aprendizaje de funciones presentado en la sección 3.3.1. Con este fin, se realizaron 100 simulaciones utilizando 800 datos de entrenamiento y 800 de prueba seleccionados aleatoriamente en el dominio de entrada $[0,1] \times [0,1] \times [0,1]$. Los valores de la función no lineal (y) fueron normalizados en el intervalo [0.05,0.95]. Los datos fueron normalizados en este intervalo y no en el [0,1] para evitar la convergencia asintótica de la función neuronal en los extremos. En este caso, la función neuronal empleada fue la función sigmoidal cuya inversa se define como

$$f^{-1}(x) = -\ln\left(\frac{1}{x} - 1\right).$$

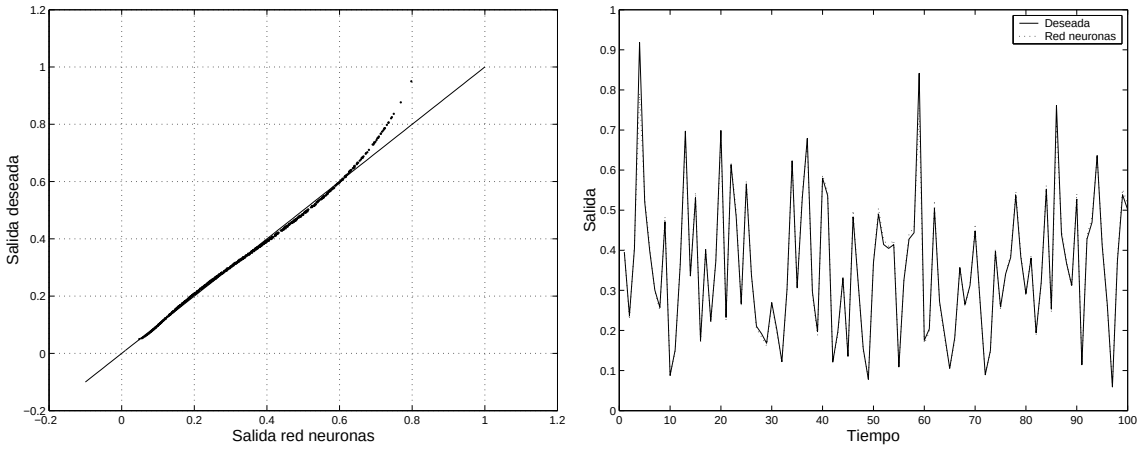
En el método con aprendizaje de funciones se utilizó, como funciones básicas en la ecuación (3.14), la siguiente familia de funciones polinómicas

$$\{\phi_1(y), \phi_2(y), \ldots, \phi_R(y)\} = \{x, x^2, \ldots, x^R\}\ . \tag{3.20}$$

Además, se emplearon diversos valores de R y β en la ecuación (3.18). En concreto, los resultados presentados en este ejemplo fueron logrados con $R = 7$ y $\beta = 0.4$.

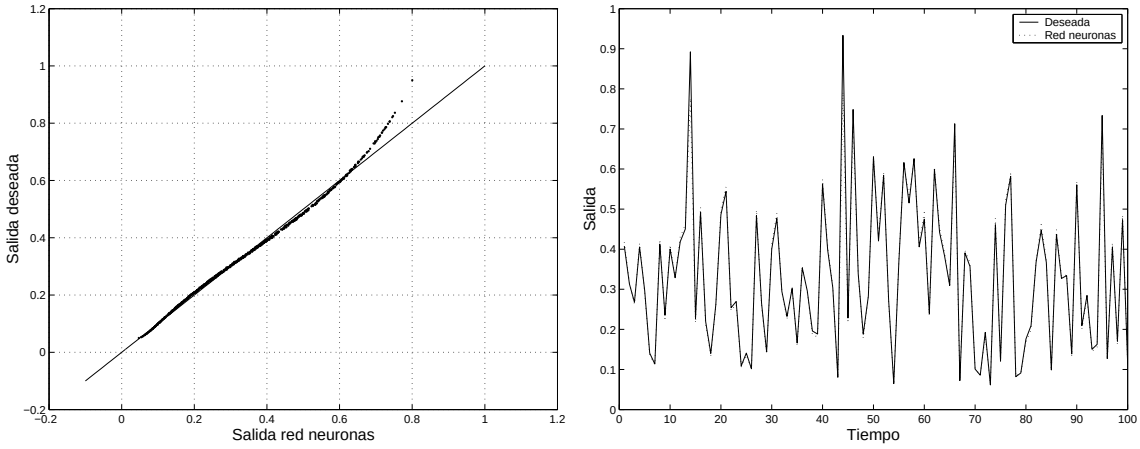
Las figuras 3.7, 3.8 y 3.9 muestran una gráfica de la salida real frente a la salida deseada (subfigura (a)) y los 100 primeros datos de la curva estimada por la red de neuronas (subfigura (b)) entrenada con el sistema de ecuaciones, el algoritmo de Levenberg-Marquardt (que es uno de los algoritmos actuales más eficientes) y el método con aprendizaje de funciones, respectivamente. La tabla 3.6 muestra el error medio y la variabilidad obtenida por cada método en las 100 simulaciones diferentes. Además, se verificó si las diferencias entre el error medio obtenido por ambos métodos eran estadísticamente significativas mediante un *t-test*. Los resultados obtenidos mediante esta prueba demostraron que, para un nivel de significación del 99%, las diferencias en el rendimiento de ambas aproximaciones son estadísticamente significativas. Por tanto, podemos afirmar que el aprendizaje de funciones supone una mejora respecto al método con funciones fijas.

La figura 3.10 muestra la función neuronal obtenida después del entrenamiento empleando el método con aprendizaje de funciones neuronales. La curva formada por las aspas representa la función neuronal obtenida, mientras que la línea discontinua



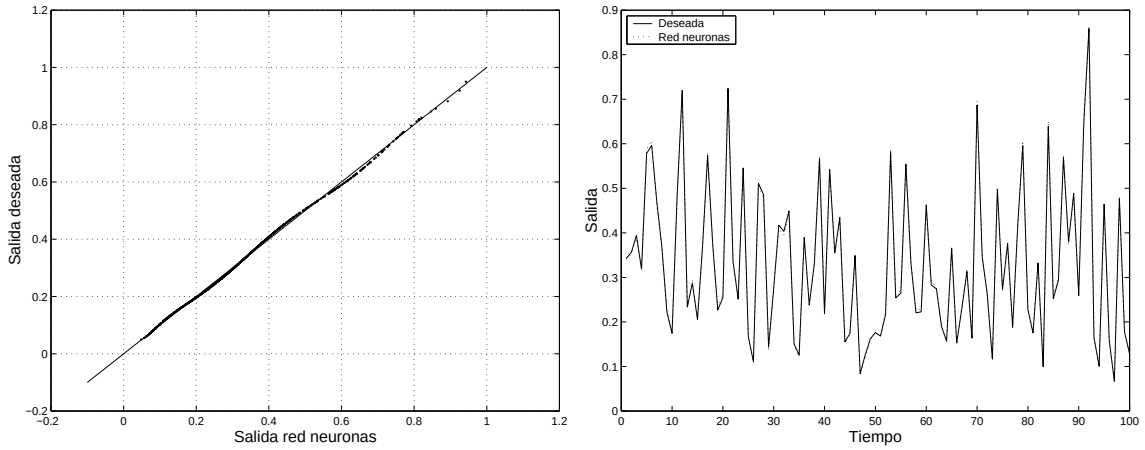
(a) (b)

Figura 3.7: Resultados del método lineal para los datos de la función no lineal: (a) salida real frente a la salida deseada, y (b) primeras 100 muestras de la función real (línea sólida) y la estimada (línea discontinua).



(a) (b)

Figura 3.8: Resultados del algoritmo de Levenberg-Marquardt para los datos de la función no lineal: (a) salida real frente a la salida deseada, y (b) primeras 100 muestras de la función real (línea sólida) y la estimada (línea discontinua).



(a) (b)

Figura 3.9: Resultados del método lineal con aprendizaje de funciones para los datos de la función no lineal: (a) salida real frente a la salida deseada, y (b) primeras 100 muestras de la función real (línea sólida) y la estimada (línea discontinua).

Tabla 3.6: Error medio y variabilidad obtenida por la red de neuronas en las 100 simulaciones para la función no lineal de tres variables.

Método de aprendizaje	Error medio	Variabilidad
Sistema de ecuaciones lineales	1.155e-02	1.729e-04
Con aprendizaje de funciones neuronales	5.438e-03	6.509e-06
Algoritmo de Levenberg-Marquardt	1.189e-02	1.746e-04

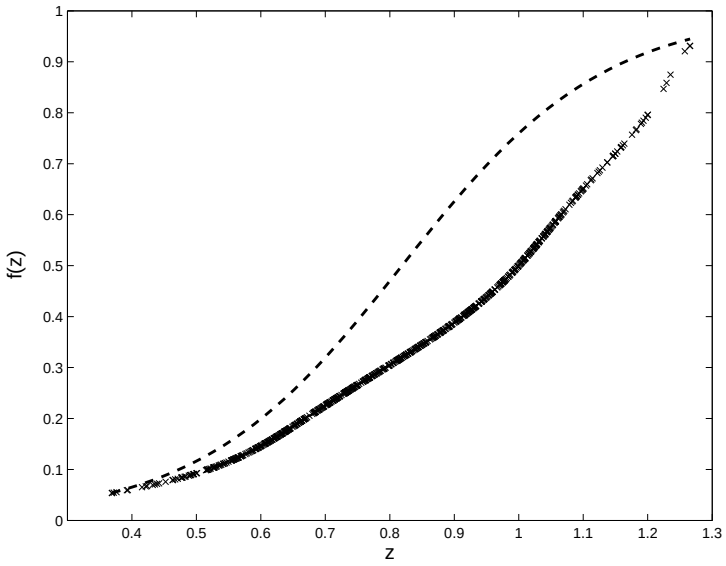


Figura 3.10: Función neuronal obtenida con el aprendizaje de funciones (curva formada por las aspas) y función sigmoide (línea discontinua) empleando los datos de la función no lineal.

es la función sigmoidal empleada para los métodos con funciones fijas. Los valores mostrados en el eje de abscisas se corresponden con los valores reales de z logrados durante el aprendizaje, pero no con los de la función sigmoidal, que han sido ajustados para poder comparar visualmente ambas funciones. Como se puede observar en esta figura la función obtenida es bastante diferente de la función sigmoide empleada normalmente en las redes de neuronas.

3.5.4 Ejemplo con los datos de Box y Jenkins

Este último experimento trata con el problema de modelar un horno de gas presentado por primera vez por Box y Jenkins [20]. El sistema a modelar consiste en un horno de gas en el cual se combinan aire y metano para formar una mezcla de gases que contiene dióxido de carbono (CO_2). La salida del horno, correspondiente a la concentración de CO_2, se mide en los gases de escape de éste. El conjunto de datos está formado por 296 pares $((u(t), y(t))$ procedentes de dos series temporales, $u(t)$ y $y(t)$, que representan la tasa de gas metano y la concentración de CO_2, respectivamente, en el instante de tiempo t en las salidas del horno. El intervalo de muestreo empleado fue de nueve segundos. El objetivo del sistema es predecir el valor de la serie $y(t)$ a partir del siguiente conjunto de entradas $\{y(t-1), y(t-2), y(t-3), y(t-4), u(t-1), u(t-2), u(t-3), u(t-4), u(t-5), u(t-6)\}$. Esto reduce el número de valores para el entrenamiento a 290. Además, antes de rea-

lizar el aprendizaje de la red, la salida deseada ($y(t)$) fue normalizada en el intervalo [0.05,0.95].

Los métodos propuestos en la sección 3.3 (red de neuronas con funciones neuronales fijas empleando la ecuación (3.11) y la sección 3.3.1 (red de neuronas con aprendizaje de funciones neuronales) se emplearon para predecir la concentración de CO_2. En el caso del aprendizaje de funciones, se probaron diversos valores de R y β. Los resultados obtenidos, para el conjunto de prueba, empleando la familia polinómica mostrada en (8.26), $R = 15$ y $\beta = 0.4$ se muestran en la tabla 3.7. Esta tabla contiene además los resultados obtenidos empleando el algoritmo de Levenberg-Marquardt. Para verificar si la diferencia en el error medio de cada uno de los sistemas es significativa se realizó un $t - test$. Como en el caso anterior, para un nivel del 99% las diferencia de rendimiento entre la red con aprendizaje de funciones y las otras dos fue significativa. Por tanto, se puede afirmar que el aprendizaje de funciones supone una mejora en el sistema neuronal.

Tabla 3.7: Error medio y variabilidad obtenida por la red de neuronas en la validación *leaving-one-out* para los datos de Box-Jenkins.

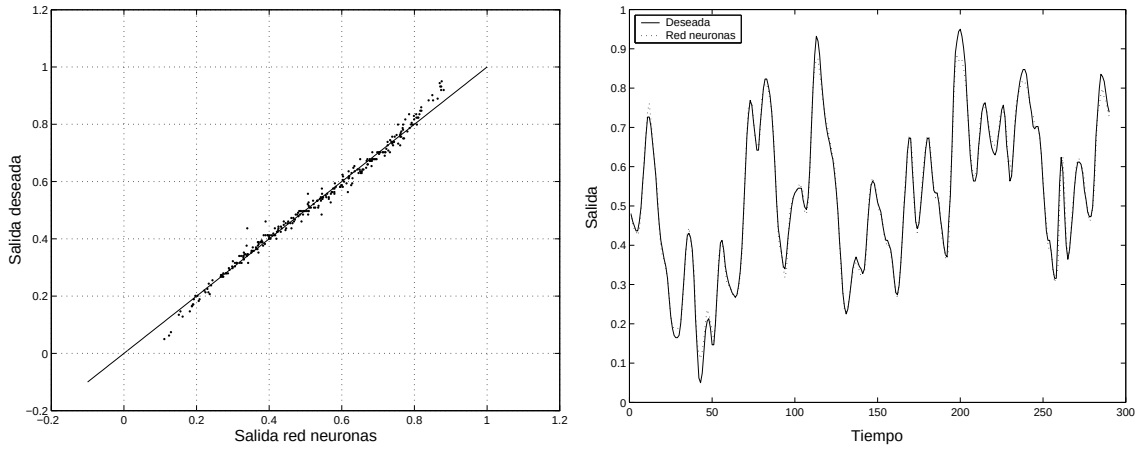
Método de aprendizaje	Error medio	Variabilidad
Sistema de ecuaciones lineales	2.084e-02	2.617e-04
Con aprendizaje de funciones neuronales	1.667e-02	1.794e-04
Algoritmo de Levenberg-Marquardt	2.090e-02	2.626e-04

Las figuras 3.11, 3.12 y 3.13 muestran una gráfica de la salida real frente a la salida deseada (subfigura (a)) y los 100 primeros datos de la curva estimada por la red de neuronas (subfigura (b)) entrenada con el sistema de ecuaciones, el algoritmo de Levenberg-Marquardt y el método con aprendizaje de funciones, respectivamente.

Finalmente, la figura 3.14 muestra la función neuronal obtenida después del aprendizaje empleando el método con aprendizaje de funciones. Como se puede observar, la función de activación obtenida es muy diferente a la del ejemplo anterior.

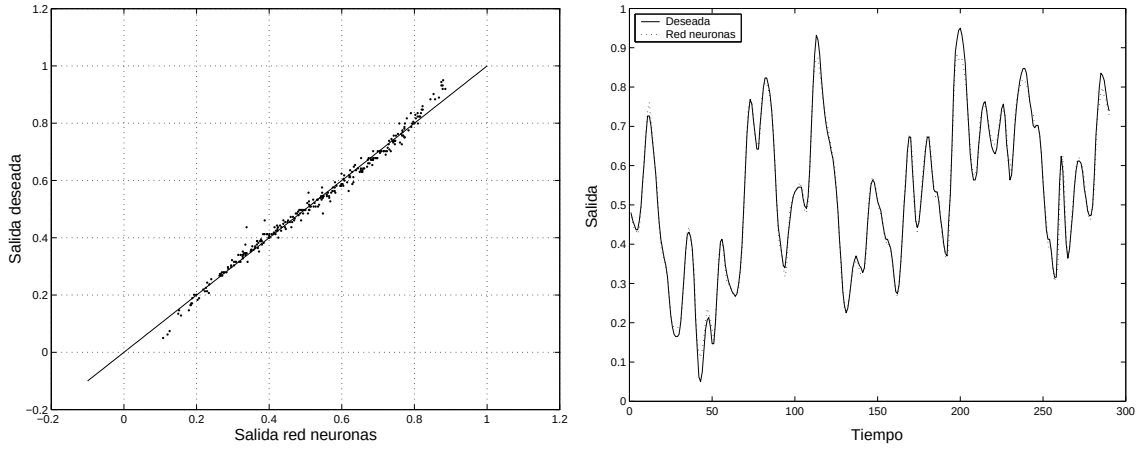
3.5.5 Estudio comparativo de la velocidad de convergencia

En esta sección se muestran los resultados del estudio comparativo, en términos de velocidad de convergencia, entre diversos algoritmos de alto rendimiento y el método propuesto en la sección 3.3 (ecuación (3.11)). Para realizar este estudio se emplearon los conjuntos de datos descritos en las secciones anteriores (secciones 3.5.1, 3.5.2, 3.5.3 y 3.5.4). A continuación se describirá brevemente cada uno de los algoritmos utilizados en este estudio.



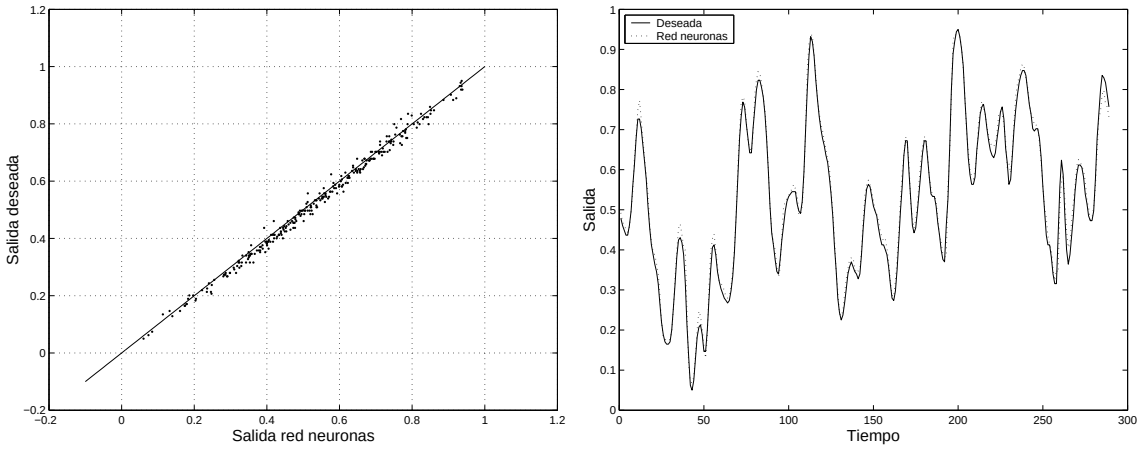
(a) (b)

Figura 3.11: Resultados del método lineal para los datos de Box-Jenkins: (a) salida real frente a la salida deseada, y (b) primeras 100 muestras de la función real (línea sólida) y la estimada (línea discontinua).



(a) (b)

Figura 3.12: Resultados del algoritmo de Levenberg-Marquardt para los datos de Box-Jenkins: (a) salida real frente a la salida deseada, y (b) primeras 100 muestras de la función real (línea sólida) y la estimada (línea discontinua).



(a) (b)

Figura 3.13: Resultados del método lineal con aprendizaje de funciones para los datos de Box-Jenkins: (a) salida real frente a la salida deseada, y (b) primeras 100 muestras de la función real (línea sólida) y la estimada (línea discontinua).

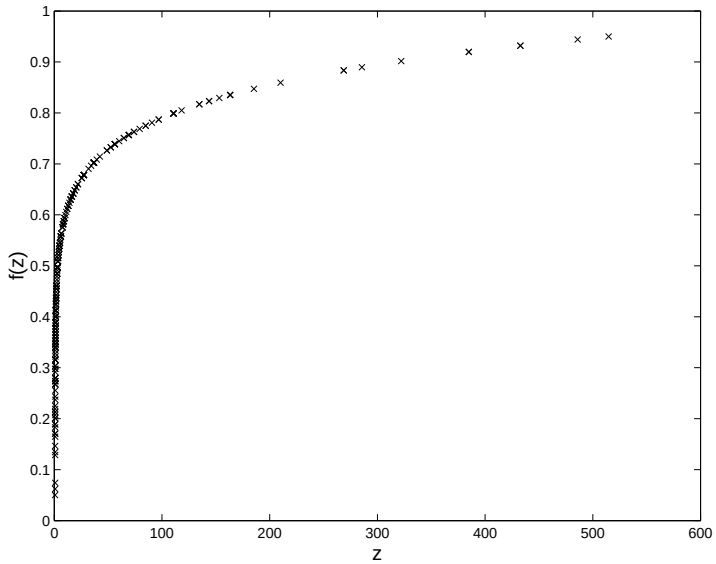


Figura 3.14: Función neuronal obtenida con el aprendizaje de funciones para los datos de Box-Jenkins.

Métodos de aprendizaje empleados en las simulaciones

Los algoritmos iterativos empleados en todas las simulaciones fueron los siguientes:

1. Gradiente conjugado de Fletcher-Reeves (CGF) [48, 55].

   El algoritmo básico de retropropagación del error actualiza los pesos en la dirección del descenso del gradiente (gradiente negativo). Esta es la dirección en la cual la función de coste decrece más rápidamente. Sin embargo, aunque la función decrece rápidamente a través del gradiente negativo, esto no produce necesariamente la convergencia más rápida hacia la solución. En los algoritmos de gradiente conjugado [62, 73] se realiza una búsqueda a través de las direcciones conjugadas, las cuales producen generalmente una convergencia más rápida que los métodos basados en las direcciones de descenso de gradiente. En la mayoría de los métodos de gradiente conjugado, el paso de aprendizaje se determina en cada iteración. Para ello, se realiza una búsqueda en la dirección de gradiente conjugado para determinar el paso de aprendizaje que minimiza la función de error a través de esa línea.

   Todos los algoritmos de gradiente conjugado comienzan la búsqueda en la dirección del descenso del gradiente, i.e.,

   $$\mathbf{p}(0) = -\mathbf{g}(0).$$

   Posteriormente, se realiza una búsqueda en la dirección (*line search*) para determinar la distancia óptima para moverse a través de la dirección actual:

   $$\mathbf{w}(n+1) = \mathbf{w}(n) - \boldsymbol{\alpha}(n)\mathbf{p}(n). \tag{3.21}$$

   Entonces, la nueva búsqueda de la dirección se determina para que sea conjugada con las direcciones anteriores. El procedimiento general para determinar la nueva dirección de búsqueda es combinar la nueva dirección de descenso de gradiente con la dirección anterior:

   $$\mathbf{p}(n) = -\mathbf{g}(n) + \boldsymbol{\beta}(n)\mathbf{p}(n-1).$$

   Las distintas versiones de los métodos de gradiente conjugado se diferencian en la manera en que es calculada la constante. En el método de Fletcher-Reeves el procedimiento de actualización es

   $$\boldsymbol{\beta}(n) = \frac{\mathbf{g}^T(n)\mathbf{g}(n)}{\mathbf{g}^T(n-1)\mathbf{g}(n-1)}$$

   que se corresponde con el ratio entre el cuadrado del gradiente actual y el de la iteración anterior.

2. Gradiente conjugado de Polak-Ribiére (CGP) [48, 55].

   Otra versión de los algoritmos de gradiente conjugado es la propuesta por Polak y Ribiére. Como en el caso del algoritmo de Fletcher-Reeves, la dirección de búsqueda en cada iteración se determina mediante la siguiente regla

   $$\mathbf{p}(n) = -\mathbf{g}(n) + \boldsymbol{\beta}(n)\mathbf{p}(n-1),$$

   sin embargo, en este caso la constante se calcula como

   $$\boldsymbol{\beta}(n) = \frac{\left(\mathbf{g}^T(n) - \mathbf{g}(n-1)\right)\mathbf{g}(n)}{\mathbf{g}^T(n-1)\mathbf{g}(n-1)}.$$

   Este valor se corresponde con el producto interno entre el cambio previo en el gradiente y el gradiente actual, dividido por la norma del gradiente previo.

3. Gradiente conjugado con reactivaciones de Powell-Beale (CGB) [8, 110].

   En todos los algoritmos de gradiente conjugado, la dirección de búsqueda se reanuda periódicamente en la dirección negativa del gradiente. El punto a partir del cual esto sucede es cuando el número de iteraciones es igual al número de pesos y umbrales de la red. Sin embargo, se pueden emplear otros métodos de reactivación que mejoran la eficiencia del entrenamiento, como el propuesto por Powell [110] basado en una versión anterior desarrollada por Beale [8]. Este método reactiva el proceso en la dirección del gradiente sólo cuando el gradiente actual y el de la iteración anterior dejan de ser más o menos ortogonales. Esto se puede verificar con la siguiente desigualdad:

   $$\left|\mathbf{g}^T(n-1)\mathbf{g}(n)\right| \geq 0.2\|\mathbf{g}(n)\|^2.$$

   Por tanto, si se satisface la condición anterior, la dirección de búsqueda se reinicia empleando el gradiente de la función.

4. Gradiente conjugado escalado (SCG) [98].

   Cada uno de los métodos de gradiente conjugado discutidos anteriormente requieren una búsqueda lineal (*line search*) en cada iteración. Esta búsqueda lineal es computacionalmente costosa, puesto que requiere que la red proporcione la salida varias veces para todos datos de entrenamiento en cada iteración. El algoritmo de gradiente conjugado, desarrollado por Moller [98], fue diseñado para evitar esta búsqueda tan costosa. Este algoritmo es demasiado complejo para explicarlo brevemente, pero la idea básica consiste en combinar la aproximación de *model-trust region*, (empleado en el algoritmo de Levenberg-Marquardt, descrito posteriormente) con el método de gradiente conjugado.

5. Quasi-Newton con actualizaciones de Broyden, Fletcher, Goldfarb y Shanno (BFGS) [37, 120].

   El método de Newton es una alternativa a los métodos de gradiente conjugado para una rápida optimización. La regla de aprendizaje básica de este método es

   $$\mathbf{w}(n+1) = \mathbf{w}(n) - \mathbf{H}^{-1}(n)\mathbf{g}(n) \tag{3.22}$$

   donde $\mathbf{H}$ es la matriz hessiana de la función objetivo para los pesos y umbrales actuales. El método de Newton converge a veces más rápidamente que los métodos de gradiente conjugado. Lamentablemente, el cálculo de la matriz hessiana, en redes con alimentación hacia delante, es computacionalmente costoso. Existe una clase de algoritmos que están basados en el método de Newton pero que no requieren el cálculo de segundas derivadas. Estos métodos son conocidos como quasi-Newton o métodos de secante. Éstos utilizan una aproximación de la matriz hessiana, calculada en base al gradiente de la función objetivo, que se actualiza en cada iteración del algoritmo. El método quasi-Newton con mejores resultados es el que se basa en las actualizaciones de Broyden, Fletcher, Goldfarb, y Shanno.

6. Retropropagación con secante de un paso (OSS) [4, 5, 6, 7].

   El cálculo de la matriz hessiana requiere del orden de N^2 operaciones y unidades de memoria para almacenar sus componentes [17], siendo N el número de parámetros de la red. Afortunadamente, alguna de la información de segundo orden se puede calcular a partir de los últimos gradientes y reducir por tanto la complejidad a $\mathcal{O}(N)$. El término método de secante proviene del hecho de que las derivadas se aproximan mediante las secantes a partir de dos valores de la función. Históricamente, el método de secante de un paso es una variante del método Broyden, Fletcher, Goldfarb y Shanno. Sin embargo, este último almacena toda la matriz que aproxima la hessiana, mientras que el método de secante de un paso utiliza sólo vectores calculados a partir del gradiente. En concreto, este método emplea la siguiente regla:

   $$\mathbf{p}(n+1) = -\mathbf{g}(n+1) + \alpha(n)\mathbf{s}(n) + \beta(n)\mathbf{y}(n)$$

   donde los escalares $\alpha(n)$ y $\beta(n)$ se calculan como la siguiente combinación de productos escalares de los vectores $\mathbf{s}(n)$, $\mathbf{g}(n+1)$ y $\mathbf{y}(n)$ (último paso de aprendizaje, gradiente actual y diferencia de gradientes, i.e., $\mathbf{y}(n) = \mathbf{g}(n+1) - \mathbf{g}(n)$):

   $$\alpha(n) = -\left(1 + \frac{\mathbf{y}^T(n)\mathbf{y}(n)}{\mathbf{s}^T(n)\mathbf{y}(n)}\right)\frac{\mathbf{s}^T(n)\mathbf{g}(n+1)}{\mathbf{s}^T(n)\mathbf{y}(n)} + \frac{\mathbf{y}^T(n)\mathbf{g}(n+1)}{\mathbf{s}^T(n)\mathbf{y}(n)}$$

$$\beta(n) = \frac{\mathbf{s}^T(n)\mathbf{g}(n+1)}{\mathbf{s}^T(n)\mathbf{y}(n)}.$$

La dirección de búsqueda al comienzo del aprendizaje es igual al gradiente y se iguala de nuevo cada N iteraciones.

7. Algoritmo de Levenberg-Marquardt (LM) [56].

   Como en los métodos quasi-Newton, el algoritmo de Levenberg-Marquardt [94] fue diseñado para alcanzar una velocidad de entrenamiento de segundo orden sin tener que calcular la matriz hessiana. Cuando la función de coste empleada es el error cuadrático medio, la hessiana puede aproximarse como

   $$\mathbf{H}(n) = \mathbf{J}^T(n)\mathbf{J}(n)$$

   y el gradiente puede calcularse como

   $$\mathbf{g}(n) = \mathbf{J}^T(n)\mathbf{e}(n)$$

   donde $\mathbf{J}$ es la matriz jacobiana que contiene las primeras derivadas de los errores de la red respecto a los pesos y umbrales, y $\mathbf{e}$ es el vector de los errores de la red. La matrix jacobiana se puede calcular mediante la técnica estándar de retropropagación del error (ver [55, 56]) que es mucho menos costosa que calcular la matriz hessiana. El algoritmo de Levenberg-Marquardt usa esta aproximación de la hessiana en la siguiente regla de optimización:

   $$\mathbf{w}(n+1) = \mathbf{w}(n) - (\mathbf{J}^T(n)\mathbf{J}(n) + \mu(n)\mathbf{I})^{-1}\mathbf{g}(n). \tag{3.23}$$

   Cuando el escalar μ es cero, el algoritmo es simplemente un método de Newton, empleando una aproximación de la hessiana, mientras que cuando μ es grande, este método se convierte en un método de descenso de gradiente con un paso de aprendizaje pequeño. El método de Newton es más rápido y preciso cerca de un mínimo, por tanto, el objetivo es cambiar hacia este método tan pronto como sea posible. El valor de μ se decrementa después de cada iteración en la que se haya reducido el valor de la función objetivo y se incrementa sólo cuando el valor de la función de error en la iteración actual crece. De esta forma, la función de error siempre decrece en cada iteración del algoritmo.

8. Retropropagación elástica (RP) [114].

   Los perceptrones multicapa usan típicamente funciones de activación sigmoidales en las neuronas de cada capa. Estas funciones están caracterizadas por el hecho de que su pendiente se aproxima a cero cuando la entrada tiene un valor grande. Esto causa un problema cuando se emplean métodos de descenso de

gradiente para el aprendizaje de este tipo de sistemas, puesto que el gradiente puede tener una magnitud muy pequeña, y por tanto, produce pequeños cambios en los pesos y umbrales. Esto puede suceder incluso cuando los parámetros de la red están lejos de sus valores óptimos.

El objetivo del algoritmo de retropropagación elástica es la eliminación de estos efectos indeseables en las magnitudes de las derivadas parciales. En este método, se tiene en cuenta el signo de la derivada para determinar la dirección de la actualización, mientras que su magnitud no tiene ningún efecto sobre la modificación de los pesos y umbrales. El valor de esta actualización se determina independientemente mediante el siguiente procedimiento: (a) el valor de actualización de un peso o umbral se incrementa por un factor δ_{inc} siempre que la derivada de la función objetivo respecto a ese peso o umbral tenga el mismo signo en dos iteraciones sucesivas, (b) el valor de la actualización se decrementa por un factor δ_{dec} siempre que la derivada cambie de signo respecto a la iteración anterior. Si la derivada es cero, la actualización permanece igual que la iteración anterior. Con este procedimiento, siempre que los pesos sean oscilantes su actualización será reducida; mientras que si los pesos cambian, durante una serie de iteraciones, en la misma dirección, entonces la magnitud de la actualización será aumentada.

Resultados

Para cada conjunto de datos se realizaron 100 ejecuciones diferentes, con pesos inicializados aleatoriamente, en un computador Silicon Graphics Origin 200. Las redes de neuronas se entrenaron empleando, en todos los casos, el error cuadrático medio como función de coste. El entrenamiento de las redes con los algoritmos iterativos finalizaba cuando se cumplía alguna de las siguientes condiciones:

- El valor de la función de error obtenido en la iteración actual alcanza un valor mínimo predeterminado (4×10^{-4}, 3×10^{-4}, 7.7×10^{-3} y 1.7×10^{-2} para los ejemplos 3.5.1, 3.5.2, 3.5.3 y 3.5.4, respectivamente)
- Se sobrepasa un determinado número de iteraciones máximo (2000).
- El valor actual del gradiente de la función de error es menor que un valor mínimo (1×10^{-10}).

En el caso del método propuesto no es necesario establecer ningún criterio de parada ya que la solución se alcanza mediante la resolución de un sistema de ecuaciones. Las tablas 3.8, 3.9, 3.10 y 3.11 muestran los resultados de velocidad de convergencia obtenidos para los datos del iris, cáncer de mama, función no lineal y

Box-Jenkins, respectivamente. El método propuesto en este trabajo está identificado con el acrónimo SEL (sistema de ecuaciones lineales). Las tablas contienen el tiempo medio (T. medio), mínimo (T. min.) y máximo (T. max.) en segundos y la desviación típica (Desv. Típica) de las 100 simulaciones. Además, la columna denominada ratio muestra, para cada algoritmo, cuantas veces es más lento, de media, que el algoritmo más rápido.

Tabla 3.8: Tiempos de convergencia hacia la solución en 100 simulaciones para los datos del iris.

Algoritmo	T. medio (s)	Ratio	T. min. (s)	T. max. (s)	Desv. Típica
SEL	0.0054	1.0	0.0053	0.0059	7.4074×10^{-5}
LM	0.1538	28.48	0.1255	0.1837	1.2203×10^{-2}
CGF	0.7929	146.83	0.1133	1.4683	2.6917×10^{-1}
CGP	0.8885	164.54	0.1260	1.6335	3.2016×10^{-1}
CGB	0.7166	132.70	0.1131	1.2714	2.1160×10^{-1}
SCG	0.6719	124.43	0.1190	1.6152	2.0010×10^{-1}
BFGS	1.3441	248.91	0.7065	1.7931	2.6661×10^{-1}
OSS	7.5704	1401.93	1.4436	22.3979	4.3330
RP	7.2496	1342.52	1.1705	19.5108	4.3517

Tabla 3.9: Tiempos de convergencia hacia la solución en 100 simulaciones para los datos del cáncer de mama.

Algoritmo	T. medio (s)	Ratio	T. min. (s)	T. max. (s)	Desv. Típica
SEL	0.0245	1.0	0.0238	0.0265	3.8419×10^{-4}
LM	0.2351	9.60	0.1989	0.3119	2.2636×10^{-2}
CGF	1.3737	56.07	0.1219	12.5924	1.6299
CGP	1.6820	68.65	0.1338	20.2112	2.8694
CGB	1.1796	48.15	0.1226	7.3879	8.6438×10^{-1}
SCG	0.7870	32.12	0.1296	1.9263	1.9669×10^{-1}
BFGS	2.7466	112.11	0.6902	8.3371	8.4858×10^{-1}
OSS	39.2298	1601.22	2.7709	173.3860	56.0295
RP	1.6540	67.51	0.6553	2.1685	3.1739×10^{-1}

Tabla 3.10: Tiempos de convergencia hacia la solución en 100 simulaciones para los datos de la función no lineal de tres variables.

Algoritmo	T. medio (s)	Ratio	T. min. (s)	T. max. (s)	Desv. Típica
SEL	0.0057	1.0	0.0055	0.0066	1.4836×10^{-4}
LM	0.1768	31.02	0.1642	0.1901	1.1216×10^{-2}
CGF	0.8346	146.42	0.5026	1.3558	1.9879×10^{-1}
CGP	0.8814	154.63	0.4800	1.8231	3.1403×10^{-1}
CGB	0.7844	137.61	0.4426	1.3517	2.2015×10^{-1}
SCG	0.5832	102.32	0.3701	0.8441	1.0821×10^{-1}
BFGS	1.1666	204.67	0.7801	1.4787	1.3563×10^{-1}
OSS	3.3554	588.67	1.6546	7.6841	1.0338
RP	3.1387	550.65	1.0870	4.5230	0.9879

Tabla 3.11: Tiempos de convergencia hacia la solución en 100 simulaciones para los datos de Box-Jenkins.

Algoritmo	T. medio (s)	Ratio	T. min. (s)	T. max. (s)	Desv. Típica
SEL	0.0243	1.0	0.0224	0.0963	7.3094×10^{-3}
LM	0.3178	13.08	0.2306	1.0212	8.9184×10^{-2}
CGF	6.7133	276.27	2.5028	10.7698	1.4806
CGP	13.6579	562.05	4.2101	98.3286	23.1694
CGB	6.0728	249.91	4.2211	8.4621	9.4923×10^{-1}
SCG	6.2298	256.37	3.8638	11.0466	1.2715
BFGS	3.5819	147.40	2.4899	4.2659	3.1732×10^{-1}
OSS	106.2492	4372.39	69.4391	113.0926	9.6851
RP	31.5719	1299.25	31.4639	32.0318	7.8482×10^{-2}

Como puede observarse en las tablas el método propuesto (SEL) es el más veloz en todos los ejemplos. Además, es importante destacar que en el mejor de los casos (función no lineal de tres variables) es 31 veces más rápido que el siguiente mejor algoritmo, mientas que en el peor de los casos (cáncer de mama) es 10 veces más rápido.

3.6 Discusión

Los resultados presentados en las secciones anteriores han demostrado que los métodos propuestos en este capítulo suponen una aportación importante en el de-

sarrollo de algoritmos de aprendizaje para redes de una capa. Las dos ventajas principales de los métodos propuestos en este capítulo son:

- Garantizan que la solución obtenida en diversas ejecuciones es siempre la misma y se corresponde con el óptimo global de la función de error. Este hecho no está garantizado en los algoritmos actuales.

- Los métodos propuestos se basan en la resolución de un sistema de ecuaciones o de un problema de programación lineal, a diferencia de los algoritmos actuales que emplean un proceso iterativo. Consecuentemente, los métodos desarrollados obtienen la solución en un tiempo mucho menor. Este hecho fue corroborado en el estudio llevado a cabo en la sección anterior. En las simulaciones realizadas se comprobó que el método basado en el sistema de ecuaciones es al menos 10 veces más rápido que el algoritmo de Levenberg-Marquardt, que es considerado, actualmente, como el algoritmo más eficiente para el aprendizaje de redes de neuronas con un número moderado de pesos.

Otra ventaja adicional del método propuesto, basado en el sistema de ecuaciones (3.11), es su gran flexibilidad para entornos que requieran aprendizaje incremental. En estos casos, cada vez que aparece un nuevo ejemplo, es necesario que el sistema incorpore ese nuevo dato a su "conocimiento", para que la tarea realizada por el sistema sea cada vez más eficaz. En esta situación es muy conveniente que la red actualice su "conocimiento" sin tener que volver a reutilizar los ejemplos empleados durante el entrenamiento. En los algoritmos empleados usualmente para el aprendizaje de perceptrones multicapa, e.g., retropropagación del error, Levenberg-Marquardt, gradiente conjugado, etc., es necesario reentrenar el sistema con el nuevo dato junto con los empleados durante el aprendizaje previo para actualizar el "conocimiento" de la red. Por el contrario, con el sistema de ecuaciones propuesto no es necesario guardar los datos empleados previamente sino solamente la matriz $\mathbf{A} \in \mathbb{R}^{(I+1)\times(I+1)}$ y el vector $\mathbf{k} \in \mathbb{R}^{I+1}$ que contienen los coeficiente requeridos para resolver el sistema de ecuaciones definido matricialmente como $\mathbf{A}\mathbf{w} = \mathbf{k}$, donde $\mathbf{w}$ es el vector que contiene los parámetros de la red. De esta forma, cuando el sistema obtiene un nuevo ejemplo sólo es necesario actualizar $\mathbf{A}$ y $\mathbf{k}$ sumándole los valores generados a partir de ese dato en (3.11) y resolver de nuevo el sistema de ecuaciones.

Finalmente, los métodos desarrollados se han ampliado para permitir el aprendizaje de las funciones neuronales en vez de asumirlas fijas. En los resultados experimentales se ha comprobado que para algunos problemas esto supone una mejora cualitativa importante y permite incrementar el rendimiento de las redes de neuronas de una capa.

Capítulo 4

Método de inicialización de los pesos para redes multicapa

"Lo último que se sabe cuando se realiza un trabajo es por dónde empezar"

Blaise Pascal

4.1 Introducción

El algoritmo presentado en el capítulo anterior supone un importante avance en el desarrollo de algoritmos de aprendizaje para redes de una capa. Sin embargo, este método no es aplicable directamente a redes multicapa, ya que en este caso no es posible obtener una solución lineal. Esto se debe a que al introducir alguna capa oculta en la red, la función de error alternativa, propuesta en el teorema 1, seguiría siendo lineal respecto a los pesos de la última capa pero no a los pesos de las capas previas, puesto que quedarían dentro de las funciones neuronales. La arquitectura de una red de neuronas multicapa con alimentación hacia delante, y la nomenclatura empleada en este trabajo, se muestra en la figura 4.1. Esta red está formada por M capas, $m = 1, \ldots, M$, donde cada capa contiene N_m neuronas. La salida $\mathbf{y}$ de la capa m se calcula como $\mathbf{y}^{(m)} = \mathbf{f}^{(m)}(\mathbf{z}^{(m)})$, donde $\mathbf{z}^{(m)} = \mathbf{W}^{(m)}\mathbf{y}^{(m-1)} + \mathbf{b}^{(m)}$. Con esta nomenclatura, las entradas de la red $\mathbf{x}$ se corresponden con la variable $\mathbf{y}^{(0)}$.

Dada la arquitectura presentada en la figura 4.1, si se emplea el problema de minimización alternativo, propuesto en el teorema 1 del capítulo anterior, se obtiene

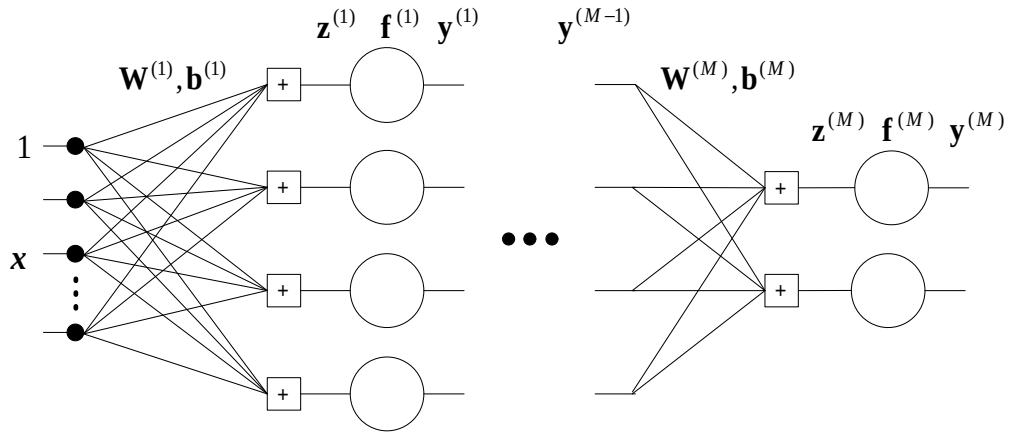


Figura 4.1: Red de neuronas multicapa con alimentación hacia delante.

la siguiente función de error:

$$\begin{aligned} C = \;& E\left[(\mathbf{f}'(\bar{\mathbf{d}}^{(M)}).*(\bar{\mathbf{d}}^{(M)}-\mathbf{z}^{(M)}))^T(\mathbf{f}'(\bar{\mathbf{d}}^{(M)}).*(\bar{\mathbf{d}}^{(M)}-\mathbf{z}^{(M)}))\right] \\ = \;& E\left[(\mathbf{f}'(\bar{\mathbf{d}}^{(M)}).*(\bar{\mathbf{d}}^{(M)}-\mathbf{W}^{(M)}\mathbf{y}^{(M-1)}+\mathbf{b}^{(M)}))^T\right. \\ & \left.\left(\mathbf{f}'(\bar{\mathbf{d}}^{(M)}).*(\bar{\mathbf{d}}^{(M)}-\mathbf{W}^{(M)}\mathbf{y}^{(M-1)}+\mathbf{b}^{(M)})\right)\right] \\ = \;& E\left[(\mathbf{f}'(\bar{\mathbf{d}}^{(M)}).*(\bar{\mathbf{d}}^{(M)}-\mathbf{W}^{(M)}\mathbf{f}^{(M-1)}(\mathbf{W}^{(M-1)}\mathbf{y}^{(M-2)}+\mathbf{b}^{(M-1)})+\mathbf{b}^{(M)}))^T\right. \\ & \left.\left(\mathbf{f}'(\bar{\mathbf{d}}^{(M)}).*(\bar{\mathbf{d}}^{(M)}-\mathbf{W}^{(M)}\mathbf{f}^{(M-1)}(\mathbf{W}^{(M-1)}\mathbf{y}^{(M-2)}+\mathbf{b}^{(M-1)})+\mathbf{b}^{(M)})\right)\right] \end{aligned} \tag{4.1}$$

Como se puede observar en la ecuación (4.1), aunque los pesos de la ultima capa (M) no están dentro de la función no lineal, sí lo están los de la capa anterior ($M-1$). Consecuentemente, no es posible obtener un sistema lineal como en el caso de la red de neuronas de una capa.

Por tanto, el siguiente objetivo abordado en esta tesis consiste en el desarrollo de un nuevo algoritmo que permita extender las ideas presentadas en el capítulo anterior para que puedan emplearse en redes multicapa. Aunque el algoritmo que se presentará posteriormente está desarrollado para una red de dos capas puede ampliarse fácilmente, y de manera sistemática, a cualquier número de capas. Sin embargo, ha sido demostrado en diversos estudios que una red con sólo dos capas y que contenga un suficiente número de neuronas en la capa oculta puede aproximar cualquier función continua y diferenciable [36, 51, 61, 65, 66, 71, 83]. Por consiguiente, esta arquitectura es suficiente en general. En un perceptrón de dos capas, la salida de la capa oculta pueden considerarse como un conjunto de funciones básicas y adaptativas, $\varphi_i(\mathbf{x})$, donde i es un índice sobre el número de neuronas en la capa oculta y $\mathbf{x}$ es el vector de entrada de la red. Por tanto, los pesos y umbrales de la primera

capa de la red son los parámetros de estas funciones adaptativas, $\varphi_i(\mathbf{x})$. Los pesos de la segunda capa puede considerarse como los coeficientes de proyección de la salida deseada al espacio no lineal expandido por las funciones básicas, $\varphi_i(\mathbf{x})$, en un espacio de Riemann multidimensional. En los métodos actuales para el aprendizaje de redes de neuronas tanto las funciones básicas, generadas por los parámetros de la primera capa, como las proyecciones, determinadas por los parámetros de la segunda capa, se optimizan al mismo tiempo. A continuación se propondrá un nuevo método basado en la actualización de las funciones básicas y las proyecciones mediante una aproximación lineal. La idea básica consiste en reflejar la salida deseada en la salida de la función no lineal de las neuronas hacia su entrada, y a continuación resolver analíticamente un problema de regresión lineal para cada una de las capas de pesos. El algoritmo propuesto, comienza la optimización desde la primera capa hasta la última, i.e., actualizando primero las funciones básicas y luego las proyecciones; sin embargo, el proceso inverso también es posible.

4.2 Propagación de la salida deseada desde la capa de salida a la capa de entrada

Teniendo en cuenta la arquitectura del perceptrón multicapa, mostrado en la figura 4.1, parece claro que la retropropagación de la salida deseada hacia la entrada de la red requiere dos etapas distintas. En primer lugar es necesario reflejar la salida deseada desde la salida de la función no lineal, $\mathbf{f}^{(m)}$, hacia su entrada. En segundo lugar, los valores obtenidos en la fase anterior se retropropagan a través de los pesos de la capa actual para obtener los valores que formarán la salida deseada para la salida de la capa previa, $\mathbf{y}^{(m-1)}$. A continuación se detallará cada una de estas etapas.

4.2.1 Propagación de la salida deseada a través de la función neuronal

En este capítulo se considerará como función de coste el error cuadrático medio ponderado, definido como:

$$C = E[(\mathbf{d}^{(M)} - \mathbf{y}^{(M)})^T \mathbf{P} (\mathbf{d}^{(M)} - \mathbf{y}^{(M)})] \tag{4.2}$$

donde $\mathbf{P}$ es una matriz que pondera el error cometido en cada una de las salidas de la red. El error cuadrático medio es un caso específico de esta función de coste en la cual la matriz $\mathbf{P}$ es igual a la identidad, i.e., $\mathbf{P} = \mathbf{I}$. Por tanto, considerando el error cuadrático medio ponderado como función objetivo, el siguiente teorema describe la retropropagación de la salida deseada a través de la función neuronal en una red multicapa.

Teorema 2 *Sean* $\mathbf{d}^{(m)}, \mathbf{y}^{(m)} \in \mathbb{R}^{N_m}$ *la salida deseada y real de la capa* m, $\mathbf{W}^{(m)} \in \mathbb{R}^{N_m \times N_{m-1}}$, $\mathbf{b}^{(m)} \in \mathbb{R}^{N_m \times 1}$ *los parámetros de esa capa, y* $\mathbf{f}^{(N_m)}, {\mathbf{f}^{(N_m)}}^{-1}, \mathbf{f}'^{(N_m)} : \mathbb{R}^{N_m} \rightarrow \mathbb{R}^{N_m}$ *la función no lineal, su inversa y su derivada. La minimización del error cuadrático medio ponderado entre* $\mathbf{d}^{(m)}$ *e* $\mathbf{y}^{(m)}$ *en la salida de la no linealidad es equivalente (hasta primer orden) a minimizar el ECM antes de la no linealidad, i.e., entre* $\mathbf{z}^{(m)} = \mathbf{W}^{(m)}\mathbf{y}^{(m-1)} + \mathbf{b}^{(m)}$ *y* $\bar{\mathbf{d}}^{(m)} = \mathbf{f}^{-1}(\mathbf{d}^{(m)})$, *donde la función inversa se evalúa para cada salida independientemente. En este último caso, cada error debe ser ponderado empleando el valor de la derivada de la no linealidad en el valor correspondiente. Matemáticamente, esta propiedad puede definirse como:*

$$\begin{aligned} &\min_{\mathbf{W^{(m)},b^{(m)}}} E[(\mathbf{d}^{(m)} - \mathbf{y}^{(m)})^T \mathbf{P}^{(m)} (\mathbf{d}^{(m)} - \mathbf{y}^{(m)})] \approx \\ &\min_{\mathbf{W^{(m)},b^{(m)}}} E[(\mathbf{f}'^{(m)}(\bar{\mathbf{d}}^{(m)}) .* \bar{\boldsymbol{\varepsilon}}^{(m)})^T \mathbf{P}^{(m)} (\mathbf{f}'^{(m)}(\bar{\mathbf{d}}^{(m)}) .* \bar{\boldsymbol{\varepsilon}}^{(m)})] \end{aligned} \tag{4.3}$$

donde '.' representa el producto de dos vectores componente a componente* $\mathbf{f}'^{(m)}(\bar{\mathbf{d}}^{(m)})$ *y* $\bar{\boldsymbol{\varepsilon}}^{(m)} = \bar{\mathbf{d}}^{(m)} - \mathbf{z}^{(m)}$.

Demostración:
Usando las ecuaciones $\mathbf{y}^{(m)} = \mathbf{f}^{(m)}(\mathbf{z}^{(m)})$ y $\mathbf{d}^{(m)} = \mathbf{f}^{(m)}(\bar{\mathbf{d}}^{(m)})$ se obtiene el siguiente problema de minimización equivalente:

$$\begin{aligned} &\min_{\mathbf{W^{(m)},b^{(m)}}} E[(\mathbf{d}^{(m)} - \mathbf{y}^{(m)})^T \mathbf{P}^{(m)} (\mathbf{d}^{(m)} - \mathbf{y}^{(m)})] = \\ &\min_{\mathbf{W}^{(m)},\mathbf{b}^{(m)}} E[(\mathbf{f}^{(m)}(\bar{\mathbf{d}}^{(m)}) - \mathbf{f}^{(m)}(\mathbf{z}^{(m)}))^T \mathbf{P}^{(m)} (\mathbf{f}^{(m)}(\bar{\mathbf{d}}^{(m)}) - \mathbf{f}^{(m)}(\mathbf{z}^{(m)}))] \end{aligned} \tag{4.4}$$

Sea $\bar{\mathbf{d}}^{(m)} = {\mathbf{f}^{(m)}}^{-1}(\mathbf{d}^{(m)})$ la salida deseada antes de la no linealidad entonces podemos definir el error antes de la función neuronal como $\bar{\boldsymbol{\varepsilon}}^{(m)} = \bar{\mathbf{d}}^{(m)} - \mathbf{z}^{(m)}$. Si la variabilidad de este error ($var(\|\bar{\boldsymbol{\varepsilon}}^{(m)}\|)$) es pequeña entonces se puede emplear la siguiente aproximación, basada en una serie de Taylor de primer orden, en cada componente del vector de salida:

$$\mathbf{f}^{(m)}(\mathbf{z}^{(m)}) = \mathbf{f}^{(m)}(\bar{\mathbf{d}}^{(m)} - \bar{\boldsymbol{\varepsilon}}^{(m)}) \approx \mathbf{f}^{(m)}(\bar{\mathbf{d}}^{(m)}) - \mathbf{f}'^{(m)}(\bar{\mathbf{d}}^{(m)}) .* \bar{\boldsymbol{\varepsilon}}^{(m)} \tag{4.5}$$

Sustituyendo esta última ecuación en (4.4) se obtiene:

$$\begin{aligned} &\min_{\mathbf{W,b}} E[(\mathbf{d}^{(m)} - \mathbf{y}^{(m)})^T \mathbf{P}^{(m)} (\mathbf{d}^{(m)} - \mathbf{y}^{(m)})] \approx \\ &\min_{\mathbf{W,b}} E[(\mathbf{f}'^{(m)}(\bar{\mathbf{d}}^{(m)}) .* \bar{\boldsymbol{\varepsilon}}^{(m)})^T \mathbf{P}^{(m)} (\mathbf{f}'^{(m)}(\bar{\mathbf{d}}^{(m)}) .* \bar{\boldsymbol{\varepsilon}}^{(m)})] \end{aligned} \tag{4.6}$$

■

Por tanto, empleando el teorema anterior se puede calcular la salida deseada antes de la función no lineal como $\bar{\mathbf{d}}^{(m)} = \mathbf{f}^{-1}(\mathbf{d}^{(m)})$ y minimizar equivalentemente el

error entre $\mathbf{z}^{(m)}$ y $\bar{\mathbf{d}}^{(m)}$. Este resultado se empleará más adelante, cuando se describa detalladamente el algoritmo, para retropropagar la salida deseada en la capa m desde la salida de la función neuronal a la entrada de ésta.

4.2.2 Propagación de la salida deseada a través de los pesos

Una vez retropropagada la salida deseada a la entrada de la función no lineal el resto de la capa está definido por la siguiente ecuación lineal $\mathbf{z}^{(m)} = \mathbf{W}^{(m)}\mathbf{y}^{(m-1)} + \mathbf{b}^{(m)}$. Además, empleando el resultado de la sección anterior tenemos una salida deseada, $\bar{\mathbf{d}}^{(m)}$, para la variable $\mathbf{z}^{(m)}$. En este caso, se supondrá que la matriz de pesos ($\mathbf{W}^{(m)}$) y el vector de umbrales ($\mathbf{b}^{(m)}$) tienen un valor fijo, mientras que el vector $\mathbf{y}^{(m-1)}$ es la variable a optimizar. Puesto que las capas previas sólo producen vectores de salida en un dominio restringido D del espacio vectorial completo, la retropropagación de la salida deseada de $\mathbf{z}^{(m)}$ para obtener un salida deseada para $\mathbf{y}^{(m-1)}$ requiere solucionar un problema lineal y acotado de mínimos cuadrados ponderado. Este resultado se presenta en siguiente teorema:

Teorema 3 *Sea $\bar{\mathbf{d}}^{(m)}, \mathbf{z}^{(m)} \in \mathbb{R}^{N_m}$ la salida deseada y la correspondiente salida antes de la función neuronal de la capa m, $\mathbf{W}^{(m)} \in \mathbb{R}^{N_m \times N_{m-1}}$, $\mathbf{b}^{(m)} \in \mathbb{R}^{N_m \times 1}$ los pesos y los umbrales de esa capa, y $\mathbf{d}^{(m-1)}, \mathbf{y}^{(m-1)} \in \mathbb{R}^{N_{m-1}}$ la salida deseada y la correspondiente salida de la capa $m-1$. La minimización del error cuadrático medio ponderado entre $\bar{\mathbf{d}}^{(m)}$ y $\mathbf{z}^{(m)}$ en la salida de la capa lineal es equivalente a minimizar el error cuadrático medio entre $\mathbf{y}^{(m-1)}$ y $\mathbf{d}^{(m-1)}$. Matemáticamente, esta propiedad puede definirse como:*

$$\begin{aligned}&\min_{\mathbf{y}^{(m-1)} \in D \subset \mathbf{R}^{N_{m-1}\times 1}} E[(\bar{\mathbf{d}}^{(m)} - \mathbf{z}^{(m)})^T \mathbf{P}^{(m)} (\bar{\mathbf{d}}^{(m)} - \mathbf{z}^{(m)})] = \\ &\min_{\mathbf{y}^{(m-1)} \in D \subset \mathbf{R}^{N_{m-1}\times 1}} E[(\mathbf{d}^{(m-1)} - \mathbf{y}^{(m-1)})^T \mathbf{W}^{(m-1)^T} \mathbf{P}^{(m)} \mathbf{W}^{(m)} (\mathbf{d}^{(m-1)} - \mathbf{y}^{(m-1)})]\end{aligned} \tag{4.7}$$

Demostración:
El error cuadrático en la salida de la capa lineal requiere solucionar el siguiente problema de minimización:

$$\begin{aligned}&\min_{\mathbf{y}^{(m-1)} \in D \subset \mathbf{R}^{N_{m-1}\times 1}} E[(\bar{\mathbf{d}}^{(m)} - \mathbf{z}^{(m)})^T \mathbf{P}^{(m)} (\bar{\mathbf{d}}^{(m)} - \mathbf{z}^{(m)})] \\ &= E[(\bar{\mathbf{d}}^{(m)T} \mathbf{P}^{(m)} \bar{\mathbf{d}}^{(m)} + \mathbf{z}^{(m)T} \mathbf{P}^{(m)} \mathbf{z}^{(m)} - 2\bar{\mathbf{d}}^{(m)T} \mathbf{P}^{(m)} \mathbf{z}^{(m)})]\end{aligned}$$

$$= E[\mathbf{z}^{(m)T}\mathbf{P}^{(m)}\mathbf{z}^{(m)}] - 2E[\bar{\mathbf{d}}^{(m)T}\mathbf{P}^{(m)}\mathbf{z}^{(m)}]$$

$$= E[(\mathbf{W}^{(m)}\mathbf{y}^{(m-1)} + \mathbf{b}^{(m)})^T\mathbf{P}^{(m)}(\mathbf{W}^{(m)}\mathbf{y}^{(m-1)} + \mathbf{b}^{(m)})] -$$

$$2E[\bar{\mathbf{d}}^{(m)T}\mathbf{P}^{(m)}(\mathbf{W}^{(m)}\mathbf{y}^{(m-1)} + \mathbf{b}^{(m)})]$$

$$= E[(\mathbf{W}^{(m)}\mathbf{y}^{(m-1)})^T\mathbf{P}^{(m)}\mathbf{W}^{(m)}\mathbf{y}^{(m-1)} + \mathbf{b}^{(m)T}\mathbf{P}^{(m)}\mathbf{b}^{(m)} + 2(\mathbf{W}^{(m)}\mathbf{y}^{(m-1)})^T\mathbf{P}^{(m)}\mathbf{b}^{(m)}] -$$

$$2E[\bar{\mathbf{d}}^{(m)T}\mathbf{P}^{(m)}(\mathbf{W}^{(m)}\mathbf{y}^{(m-1)} + \mathbf{b}^{(m)})]$$

$$= E[\mathbf{y}^{(m-1)T}\mathbf{W}^{(m)T}\mathbf{P}^{(m)}\mathbf{W}^{(m)}\mathbf{y}^{(m-1)}] + 2E[\mathbf{y}^{(m)T}\mathbf{W}^{(m)T}\mathbf{P}^{(m)}\mathbf{b}^{(m)}] -$$

$$2E[\bar{\mathbf{d}}^{(m)T}\mathbf{P}^{(m)}\mathbf{W}^{(m)}\mathbf{y}^{(m-1)}] - 2E[\bar{\mathbf{d}}^{(m)T}\mathbf{P}^{(m)}\mathbf{b}^{(m)}]$$

$$\equiv \min_{\mathbf{y}^{(m-1)} \in D \subset \mathbb{R}^{N_{m-1}\times 1}} E[\mathbf{y}^{(m-1)T}\mathbf{W}^{(m)T}\mathbf{P}^{(m)}\mathbf{W}^{(m)}\mathbf{y}^{(m-1)}] +$$
$$2E[\mathbf{y}^{(m-1)T}\mathbf{W}^{(m)T}\mathbf{P}^{(m)}\mathbf{b}^{(m)}] - 2E[\bar{\mathbf{d}}^{(m)T}\mathbf{P}^{(m)}\mathbf{W}^{(m)}\mathbf{y}^{(m-1)}]$$

donde $\mathbf{P}^{(m)}$ es la matriz de ponderación del error cuadrático medio ponderado. Definiendo $\mathbf{d}^{(m)}$ como la solución óptima para el problema empleando el método de mínimos cuadrados y $\boldsymbol{\varepsilon}^{(m)} \in D' \subset \mathbb{R}^{N_m\times 1}$ como el error entre la entrada $\mathbf{y}^{(m-1)}$ y $\mathbf{d}^{(m-1)}$ (i.e., $\boldsymbol{\varepsilon}^{(m-1)} = \mathbf{d}^{(m-1)} - \mathbf{y}^{(m-1)}$), entonces el problema de minimización anterior es equivalente a realizar la optimización sobre el error $\boldsymbol{\varepsilon}^{(m-1)}$:

$$\begin{aligned}
&\min_{\boldsymbol{\varepsilon}^{(m-1)} \in D'} E[(\mathbf{d}^{(m-1)} - \boldsymbol{\varepsilon}^{(m-1)})^T\mathbf{W}^{(m)T}\mathbf{P}^{(m)}\mathbf{W}^{(m)}(\mathbf{d}^{(m-1)} - \boldsymbol{\varepsilon}^{(m-1)})] + \\
&2E[(\mathbf{d}^{(m-1)} - \boldsymbol{\varepsilon}^{(m-1)})^T\mathbf{W}^{(m)^T}\mathbf{P}^{(m)}\mathbf{b}^{(m)}] \\
&-2E[\bar{\mathbf{d}}^{(m)^T}\mathbf{P}^{(m)}\mathbf{W}^{(m)}(\mathbf{d}^{(m-1)} - \boldsymbol{\varepsilon}^{(m-1)})] \\
= \;& E[(\mathbf{d}^{(m-1)} - \boldsymbol{\varepsilon}^{(m-1)})^T\mathbf{W}^{(m)^T}\mathbf{P}^{(m)}\mathbf{W}^{(m)}(\mathbf{d}^{(m-1)} - \boldsymbol{\varepsilon}^{(m-1)})] \\
&+2E[\mathbf{d}^{(m-1)T}\mathbf{W}^{(m)T}\mathbf{P}^{(m)}\mathbf{b}^{(m)} - \boldsymbol{\varepsilon}^{(m-1)T}\mathbf{W}^{(m)T}\mathbf{P}^{(m)}\mathbf{b}^{(m)}] \\
&-2E[\bar{\mathbf{d}}^{(m)T}\mathbf{P}^{(m)}\mathbf{W}^{(m)}\mathbf{d}^{(m-1)} - \bar{\mathbf{d}}^{(m)T}\mathbf{P}^{(m)}\mathbf{W}^{(m)}\boldsymbol{\varepsilon}^{(m-1)}] \\
= \;& E[(\mathbf{d}^{(m-1)} - \boldsymbol{\varepsilon}^{(m-1)})^T\mathbf{W}^{(m)^T}\mathbf{P}^{(m)}\mathbf{W}^{(m)}(\mathbf{d}^{(m-1)} - \boldsymbol{\varepsilon}^{(m-1)})] \\
&+2E[\mathbf{d}^{(m-1)T}\mathbf{W}^{(m)T}\mathbf{P}^{(m)}\mathbf{b}^{(m)}] - 2E[\boldsymbol{\varepsilon}^{(m-1)T}\mathbf{W}^{(m)T}\mathbf{P}^{(m)}\mathbf{b}^{(m)}] \\
&-2E[\bar{\mathbf{d}}^{(m)T}\mathbf{P}^{(m)}\mathbf{W}^{(m)}\mathbf{d}^{(m-1)}] + 2E[\bar{\mathbf{d}}^{(m)T}\mathbf{P}^{(m)}\mathbf{W}^{(m)}\boldsymbol{\varepsilon}^{(m-1)}]
\end{aligned}$$

$$
\begin{aligned}
&= E[(\mathbf{d}^{(m-1)} - \boldsymbol{\varepsilon}^{(m-1)})^T \mathbf{W}^{(m)^T} \mathbf{P}^{(m)} \mathbf{W}^{(m)} (\mathbf{d}^{(m-1)} - \boldsymbol{\varepsilon}^{(m-1)})] \\
&\quad + 2E[\mathbf{b}^{(m)T} \mathbf{P}^{(m)} \mathbf{W}^{(m)} \mathbf{d}^{(m-1)}] - 2E[\mathbf{b}^{(m)T} \mathbf{P}^{(m)} \mathbf{W}^{(m)} \boldsymbol{\varepsilon}^{(m-1)}] \\
&\quad - 2E[\bar{\mathbf{d}}^{(m)T} \mathbf{P}^{(m)} \mathbf{W}^{(m)} \mathbf{d}^{(m-1)}] + 2E[\bar{\mathbf{d}}^{(m)T} \mathbf{P}^{(m)} \mathbf{W}^{(m)} \boldsymbol{\varepsilon}^{(m-1)}] \\
&= E[\mathbf{d}^{(m-1)T} \mathbf{W}^{(m)T} \mathbf{P}^{(m)} \mathbf{W}^{(m)} \mathbf{d}^{(m-1)} + \boldsymbol{\varepsilon}^{(m-1)T} \mathbf{W}^{(m)T} \mathbf{P}^{(m)} \mathbf{W}^{(m)} \boldsymbol{\varepsilon}^{(m-1)} \\
&\quad - 2\mathbf{d}^{(m-1)T} \mathbf{W}^{(m)T} \mathbf{P}^{(m)} \mathbf{W}^{(m)} \boldsymbol{\varepsilon}^{(m-1)}] \\
&\quad - 2E[\mathbf{b}^{(m)T} \mathbf{P}^{(m)} \mathbf{W}^{(m)} \boldsymbol{\varepsilon}^{(m-1)}] + 2E[\bar{\mathbf{d}}^{(m)T} \mathbf{P}^{(m)} \mathbf{W}^{(m)} \boldsymbol{\varepsilon}^{(m-1)}] \\
&= E[\boldsymbol{\varepsilon}^{(m-1)T} \mathbf{W}^{(m)T} \mathbf{P}^{(m)} \mathbf{W}^{(m)} \boldsymbol{\varepsilon}^{(m-1)}] - 2E[\mathbf{d}^{(m-1)T} \mathbf{W}^{(m)T} \mathbf{P}^{(m)} W^{(m)} \boldsymbol{\varepsilon}^{(m-1)}] \\
&\quad + 2E[\bar{\mathbf{d}}^{(m)T} \mathbf{P}^{(m)} \mathbf{W}^{(m)} \boldsymbol{\varepsilon}^{(m-1)}] - 2E[\mathbf{b}^{(m)T} \mathbf{P}^{(m)} \mathbf{W}^{(m)} \boldsymbol{\varepsilon}^{(m-1)}] \\
&= E[\boldsymbol{\varepsilon}^{(m-1)T} \mathbf{W}^{(m)T} \mathbf{P}^{(m)} \mathbf{W}^{(m)} \boldsymbol{\varepsilon}^{(m-1)}] - 2E[\mathbf{d}^{(m-1)T} \mathbf{W}^{(m)T} \mathbf{P}^{(m)} W^{(m)} \boldsymbol{\varepsilon}^{(m-1)}] \\
&\quad + 2E[(\bar{\mathbf{d}}^{(m)} - \mathbf{b}^{(m)})^T \mathbf{P}^{(m)} \mathbf{W}^{(m)} \boldsymbol{\varepsilon}^{(m-1)}]
\end{aligned}
$$

Sustituyendo en $\mathbf{d}^{(m-1)}$ la solución obtenida por mínimos cuadrados los dos últimos términos se anulan y por tanto el problema de minimización es equivalente a:

$$
\begin{aligned}
&\min_{\boldsymbol{\varepsilon}^{(m-1)} \in D'} E[\boldsymbol{\varepsilon}^{(m-1)T} \mathbf{W}^{(m)T} \mathbf{P}^{(m)} \mathbf{W}^{(m)} \boldsymbol{\varepsilon}^{(m-1)}] \equiv \\
&\min_{\mathbf{y}^{(m-1)} \in D} E[(\mathbf{d}^{(m-1)} - \mathbf{y}^{(m-1)})^T \mathbf{W}^{(m)T} \mathbf{P}^{(m)} \mathbf{W}^{(m)} (\mathbf{d}^{(m-1)} - \mathbf{y}^{(m-1)})]
\end{aligned}
$$

■

Hay dos situaciones que requieren una atención especial en este momento:

- Si $N_m \geq N_{m-1}$, entonces $\mathbf{d}^{(m-1)} = (\mathbf{W}^{(m)T} \mathbf{P}^{(m)} \mathbf{W}^{(m)})^{-1} \mathbf{W}^{(m)T} \mathbf{P}^{(m)} (\bar{\mathbf{d}}^{(m)} - \mathbf{b}^{(m)})$ es la única solución de mínimos cuadrados ponderado para el sistema de ecuaciones lineales sobredeterminado ($\mathbf{W}^{(m)} \mathbf{d}^{(m-1)} + \mathbf{b}^{(m)} = \bar{\mathbf{d}}^{(m)}$ con error cuadrático medio ponderado por la matriz $\mathbf{P}^{(m)}$).

- Si $N_m < N_{m-1}$, entonces se puede emplear la factorización QR para determinar la solución de mínima norma para el sistema de ecuaciones lineales indeterminado que se obtiene ($\mathbf{W}^{(m)} \mathbf{d}^{(m-1)} + \mathbf{b}^{(m)} = \bar{\mathbf{d}}^{(m)}$) [86].

En ambos casos, este resultado proporciona un procedimiento para, dada una salida deseada $\bar{\mathbf{d}}^{(m)}$ para la variable $\mathbf{z}^{(m)}$ de la m-ésima capa, trasladar esta salida deseada como una nueva salida deseada ($\mathbf{d}^{(m-1)}$) para la salida (después de la no linealidad) de la capa previa. Este nuevo valor puede trasladarse a la entrada de la no linealidad

empleando el resultado obtenido en la sección 4.2.1 y así sucesivamente, empleando ambos resultados, hasta llegar a la entrada de la red.

4.3 Cálculo de los pesos óptimos mediante mínimos cuadrados

Una vez que la salida deseada de la red se ha retropropagado de acuerdo a los procedimientos descritos en la sección anterior, la siguiente tarea consiste en encontrar los pesos óptimos de cada capa. Para ello, basta con considerar cada una de las capas de la red como una red lineal de la forma $\mathbf{z} = \mathbf{W}\mathbf{x} + \mathbf{b}, \mathbf{W} \in \mathbb{R}^{J\times I}, \mathbf{b} \in \mathbb{R}^{J\times 1}$ con los datos de entrenamiento $(\mathbf{x}_s, \bar{\mathbf{d}}_s)$, y una matriz $\mathbf{G} = [\gamma_{ij}]$ para el error cuadrático medio ponderado. Es importante resaltar que en este caso se ha eliminado el superíndice que indica el número de la capa por motivos de claridad. Además, en este caso la variable $\mathbf{x}$ representa la entrada de la capa actual que es igual a la salida de la capa previa ($\mathbf{y}^{(m-1)}$) menos para la primera capa que se corresponde con la entrada de la red. El error para cada dato del conjunto de entrenamiento y para cada salida se define como:

$$\bar{\varepsilon}_{js} = \bar{d}_{js} - z_{js}, j = 1, \ldots, J; s = 1, \ldots, S,$$

donde cada salida se calcula como $z_{js} = b_j + \sum_{i=1}^{I} w_{ji}x_{is}, j = 1, \ldots, J; s = 1, \ldots, S$. Los pesos óptimos para esta capa, empleando el teorema 2 y 3, son la solución al siguiente problema de minimización:

$$\min_{\mathbf{W},\mathbf{b}} C = \frac{1}{S}\sum_{s=1}^{S}\sum_{i=1}^{J}\sum_{j=1}^{J}\gamma_{ij}f'(\bar{d}_{is})f'(\bar{d}_{js})\bar{\varepsilon}_{is}\bar{\varepsilon}_{js}. \tag{4.8}$$

$$\frac{\partial C}{\partial w_{kl}} = \frac{1}{S}\sum_{s=1}^{S}\sum_{i=1}^{J}\sum_{j=1}^{J}\gamma_{ij}f'(\bar{d}_{is})f'(\bar{d}_{js})\left[\bar{\varepsilon}_{is}\frac{\partial\bar{\varepsilon}_{js}}{\partial w_{kl}} + \frac{\partial\bar{\varepsilon}_{is}}{\partial w_{kl}}\bar{\varepsilon}_{js}\right]; \tag{4.9}$$
$$k = 1, \ldots, J; l = 1, \ldots, I$$

$$\frac{\partial C}{\partial b_{k}} = \frac{1}{S}\sum_{s=1}^{S}\sum_{i=1}^{J}\sum_{j=1}^{J}\gamma_{ij}f'(\bar{d}_{is})f'(\bar{d}_{js})\left[\bar{\varepsilon}_{is}\frac{\partial\bar{\varepsilon}_{js}}{\partial b_{k}} + \frac{\partial\bar{\varepsilon}_{is}}{\partial b_{k}}\bar{\varepsilon}_{js}\right]; \tag{4.10}$$
$$k = 1, \ldots, J,$$

donde,

$$\frac{\partial\bar{\varepsilon}_{js}}{\partial w_{kl}} = -x_{ls}\delta_{kj} \quad , \quad \frac{\partial\bar{\varepsilon}_{js}}{\partial b_{k}} = -\delta_{kj},$$

siendo δ_{kj} la función delta de Kronecker definida como:

$$\delta_{kj} = \begin{cases} 1 & si \quad k = j \\ 0 & si \quad k \neq j \end{cases}$$

Igualando las derivadas a cero y reordenando los términos se obtiene el siguiente sistema de $(J+1)\times I$ ecuaciones lineales con $(J+1)\times I$ incógnitas:

$$\begin{aligned}
&\sum_{i=1}^{J} b_i\gamma_{ik}\left[\sum_{s=1}^{S} f'(\bar{d}_{ks})f'(\bar{d}_{is})x_{ls}\right] + \sum_{p=1}^{I}\sum_{i=1}^{J} w_{ip}\gamma_{ik}\left[\sum_{s=1}^{S} f'(\bar{d}_{ks})f'(\bar{d}_{is})x_{ls}x_{ps}\right] \\
&\qquad = \sum_{i=1}^{J}\gamma_{ik}\left[\sum_{s=1}^{S} f'(\bar{d}_{ks})f'(\bar{d}_{is})x_{ls}d_{is}\right]; k=1,\ldots,J; l=1,\ldots,I \\
&\sum_{i=1}^{J} b_i\gamma_{ik}\left[\sum_{s=1}^{S} f'(\bar{d}_{ks})f'(\bar{d}_{is})\right] + \sum_{p=1}^{I}\sum_{i=1}^{J} w_{ip}\gamma_{ik}\left[\sum_{s=1}^{S} f'(\bar{d}_{ks})f'(\bar{d}_{is})x_{ps}\right] \\
&\qquad = \sum_{i=1}^{J}\gamma_{ik}\left[\sum_{s=1}^{S} f'(\bar{d}_{ks})f'(\bar{d}_{is})d_{is}\right]; k=1,\ldots,J.
\end{aligned} \tag{4.11}$$

Solucionando el sistema de ecuaciones lineales anterior para las variables w_{ip} y b_i, se obtienen los pesos óptimos para la capa actual. Es importante resaltar también que la matriz $\mathbf{G}$ en este problema de optimización permite tener en cuenta el valor de los pesos de las capas posteriores. En concreto esta matriz se corresponde con la matriz de ponderación de cada capa $\mathbf{W}^{(m-1)^T}\mathbf{P}^{(m)}\mathbf{W}^{(m)}$, obtenida en el teorema 3. En las simulaciones realizadas en este trabajo, detalladas posteriormente, se ha utilizado, en todos los casos, el error cuadrático medio sin ponderar y, por tanto, $\mathbf{P}^{(m)}=\mathbf{I}$. Por otro lado, los términos de la derivada de la función neuronal permiten considerar el efecto de la pendiente de la función.

4.4 Algoritmo de aprendizaje para una red con una capa oculta

En esta sección, se describirá cómo aplicar los resultados mostrados en las secciones anteriores para obtener un método para la inicialización de una red multicapa con alimentación hacia delante. Aunque el algoritmo que se detallará a continuación está orientado para una red de dos capas puede ser extendido fácilmente, y de manera sistemática, a cualquier número de capas. Sin embargo, como ya ha sido mencionado previamente, una red con sólo dos capas y un número suficiente de elementos de procesamiento en la capa oculta puede aproximar cualquier función continua. Por tanto esta arquitectura es suficiente en general. Consideremos una red con dos capas de pesos con N_0 entradas, N_1 neuronas en la capa oculta y N_2 neuronas en la capa de salida. Sean $\{\mathbf{W}^{(1)},\mathbf{b}^{(1)}\}$ y $\{\mathbf{W}^{(2)}$ y $\mathbf{b}^{(2)}\}$ los parámetros de la primera y segunda capa, respectivamente, y $\mathbf{x}_s$ $(s=1,\ldots,N)$ un conjunto de N vectores de entrada. El algoritmo propuesto para la inicialización de los pesos de la red de dos capas se

describe a continuación.

Algoritmo:

1. Seleccionar aleatoriamente los pesos y los umbrales $\mathbf{W}^{(1)}$, $\mathbf{b}^{(1)}$, $\mathbf{W}^{(2)}$ y $\mathbf{b}^{(2)}$. Dado el par $(\mathbf{x}_s, \mathbf{d}_s^{(2)})$, $s = 1, \ldots, N$ evaluar $\mathbf{z}_s^{(1)}$, $\mathbf{y}_s^{(1)}$, $\mathbf{z}_s^{(2)}$, $\mathbf{y}_s^{(2)}$ usando $\mathbf{x}_s$ y los pesos aleatorios. Asignar a C_{opt} el valor del ECM entre $\mathbf{y}_s^{(2)}$ y $\mathbf{d}_s^{(2)}$. Asignar $\mathbf{W}_{opt}^{(1)} = \mathbf{W}^{(1)}$, $\mathbf{b}_{opt}^{(1)} = \mathbf{b}^{(1)}$, $\mathbf{W}_{opt}^{(2)} = \mathbf{W}^{(2)}$ y $\mathbf{b}_{opt}^{(2)} = \mathbf{b}^{(2)}$.

2. Calcular $\bar{\mathbf{d}}_s^{(2)} = {\mathbf{f}^{(2)}}^{-1}(\mathbf{d}_s^{(2)})$, $\forall s$ y emplearla como salida deseada para $\mathbf{z}_s^{(2)}$.

3. Calcular $\mathbf{d}_s^{(1)} = (\mathbf{W}^{(2)T}\mathbf{W}^{(2)})^{-1}\mathbf{W}^{(2)T}(\bar{\mathbf{d}}_s^{(2)} - \mathbf{b}^{(2)})$ y usarla como salida deseada para $y_s^{(1)}$ (esta es la solución para el caso sobredeterminado donde $N_2 > N_1$, para el caso indeterminado se podría emplear la solución de mínima norma).

4. Calcular $\bar{\mathbf{d}}_s^{(1)} = {\mathbf{f}^{(1)}}^{-1}(\mathbf{d}_s^{(1)})$, $\forall s$ y usarla como salida deseada para $\mathbf{z}_s^{(1)}$.

5. Optimizar $\mathbf{W}^{(1)}$ y $\mathbf{b}^{(1)}$ resolviendo el sistema de ecuaciones (4.11), usando $\mathbf{x}_s$ como las muestras de las entradas y $\bar{\mathbf{d}}_s^{(1)}$ como las correspondientes salidas deseadas. Emplear $\mathbf{G}^{(1)} = \mathbf{W}^{(2)T}\mathbf{W}^{(2)}$ como matriz de ponderación para el ECM ponderado. Esta matriz puede ser elegida opcionalmente como la matriz identidad, ya que en general los pesos de las siguiente capa no estarán todavía optimizados y por tanto el uso de estos pesos como factor de ponderación es superfluo. En consecuencia, en la mayoría de los casos es más adecuado emplear $\mathbf{G}^{(1)} = \mathbf{I}$ (este hecho se discutirá en detalle posteriormente).

6. Evaluar $\mathbf{z}_s^{(1)}$ y $\mathbf{y}_s^{(1)}$ usando los nuevos valores de los pesos de la primera capa.

7. Optimizar $\mathbf{W}^{(2)}$ y $\mathbf{b}^{(2)}$ empleando las ecuaciones obtenidas mediante el sistema de ecuaciones (4.11), usando $\mathbf{y}_s^{(1)}$ como datos en las entrada de la red y $\bar{\mathbf{d}}_s^{(2)}$ como las correspondientes salidas deseadas.

8. Evaluar $\mathbf{z}_s^{(2)}$ y $\mathbf{y}_s^{(2)}$ usando los nuevos valores de los pesos de la segunda capa.

9. Calcular el valor de C (el ECM entre $\mathbf{y}_s^{(2)}$ y $\mathbf{d}_s^{(2)}$). Si $C < C_{opt}$ entonces realizar las siguientes asignaciones $C_{opt} = C$, $\mathbf{W_{opt}}^{(1)} = \mathbf{W}^{(1)}$, $\mathbf{b_{opt}}^{(1)} = \mathbf{b}^{(1)}$, $\mathbf{W_{opt}}^{(2)} = \mathbf{W}^{(2)}$ y $\mathbf{b_{opt}}^{(2)} = \mathbf{b}^{(2)}$.

10. Volver al paso 3.

El algoritmo presentado está basado en la propagación de la salida deseada desde la capa de salida hasta la de entrada, para luego, optimizar los pesos desde la primera hasta la última capa. La figura 4.2 muestra de forma esquemática este proceso.

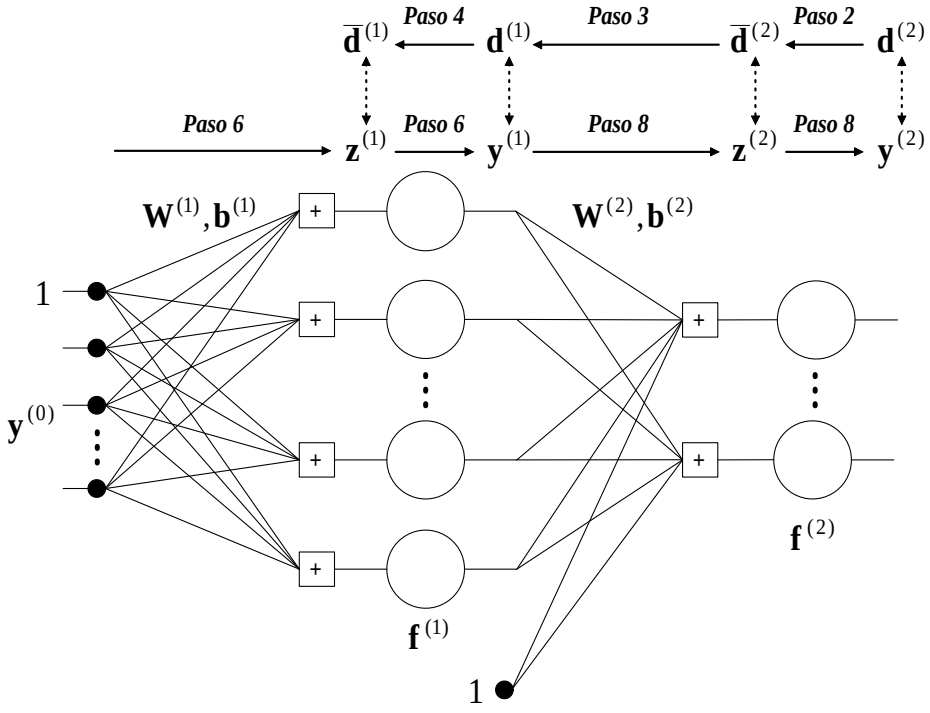


Figura 4.2: Esquema de los pasos del algoritmo basado en la retropropagación de la salida deseada para una red de neuronas con alimentación hacia delante de dos capas.

Alternativamente, se podrían optimizar primero los parámetros de la última capa, posteriormente propagar la salida deseada a través de esta capa empleando los pesos optimizados y así sucesivamente hasta llegar hasta la primera capa. En este algoritmo alternativo las capas de la red son optimizadas empezando en la última y finalizando en la primera. En este trabajo se realizaron diversas pruebas para verificar si existe alguna diferencia entre ambos métodos. Sin embargo, los resultados experimentales mostraron que no existe ninguna diferencia significativa entre ambos métodos y, por lo tanto, pueden emplearse equivalentemente.

Finalmente, una vez que el algoritmo ha sido iterado unas pocas veces los pesos correspondientes al ECM más pequeño, en todas las iteraciones realizadas, pueden emplearse como pesos iniciales del algoritmo estándar de retropropagación del error (en general, como condición inicial de cualquier método de optimización). Para ilustrar mejor el funcionamiento y el comportamiento del algoritmo se mostrará a continuación una analogía con el brazo de un robot. La figura 4.3 presenta el esquema de este sistema que contiene dos ángulos, α_1 y α_2, correspondientes a las uniones del brazo con el cuerpo y del brazo con el antebrazo, respectivamente. En esta analogía, los pesos de ambas capas se corresponden con los ángulos de unión del brazo, uno de los cuales está fijo en un determinado punto (definido por los datos de entrada

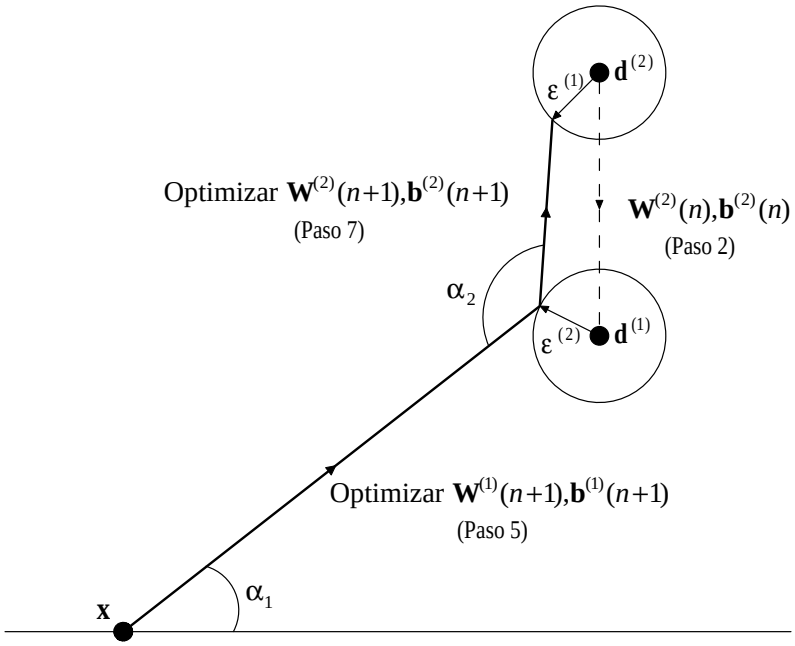


Figura 4.3: Ilustración del funcionamiento del algoritmo de inicialización empleando una analogía con el brazo de un robot.

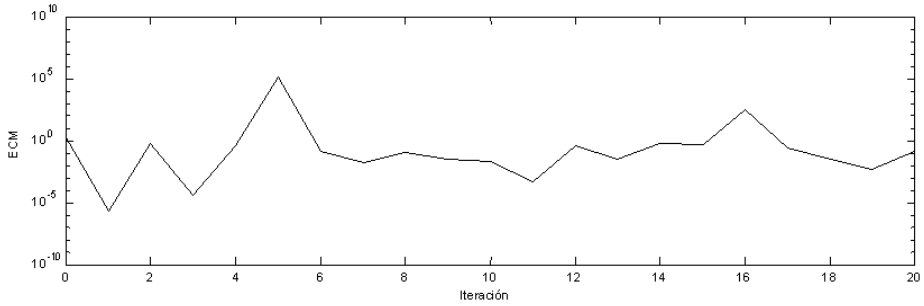
de la red) mientras que el otro trata de alcanzar la posición en el espacio deseado (correspondiente a la salida deseada de la red). La longitud limitada del brazo está asociada con la elección restringida de las funciones básicas y el espacio generado por éstas. En cada iteración del algoritmo, el brazo del robot evalúa primero la posición deseada para el ángulo α_1, dependiendo del valor actual del ángulo α_2 (esto se corresponde con la retropropagación del error a través de las capas). A continuación, el sistema mueve la unión correspondiente al ángulo α_1 (pesos de la primera capa) hasta llevar al brazo tan cerca como sea posible de su posición deseada (retropropagación de la salida deseada a través de la primera capa). Finalmente, se mueve la unión correspondiente al ángulo α_2 (pesos de la segunda capa) hasta colocar el antebrazo tan cerca como sea posible de la posición deseada (actual salida deseada de la red).

4.5 Discusión del método

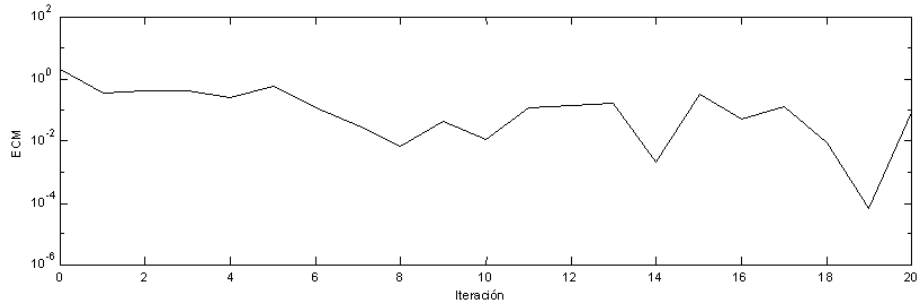
Hay varias cuestiones que destacar en el algoritmo de inicialización propuesto. La más importante es por qué el algoritmo presentado se propone como método de inicialización de los pesos y no como un método de entrenamiento. Es evidente, por la forma de derivar el algoritmo, que en cada una de las iteraciones del proceso no se impone, explícitamente, una reducción del error cuadrático medio respecto a la

iteración anterior. De hecho, ha sido observado en diversas simulaciones que, aunque el algoritmo alcanza un valor del error cuadrático medio muy pequeño, cercano al óptimo global, en unas pocas iteraciones, no converge necesariamente hacia un mínimo al final del entrenamiento. Por tanto, este método debe emplearse como un procedimiento rápido y eficaz para obtener un punto de inicio cercano al óptimo. Además, si se almacena el error y los pesos óptimos de la mejor de las iteraciones (comparando el error obtenido con la iteración actual) se puede emplear éste en lugar de los pesos obtenidos en la última iteración del algoritmo. Una segunda cuestión acerca del hecho discutido anteriormente es que el algoritmo de aprendizaje no presenta una curva de error decreciente durante el entrenamiento. De hecho, fue observado que en cada iteración el algoritmo salta de una región del espacio de pesos a otra completamente diferente. Esto no debe considerarse como una debilidad del algoritmo, sino al contrario, ya que esta búsqueda estocástica es el punto fuerte del método como algoritmo de búsqueda efectivo para determinar un buen conjunto de pesos. Además, empleando la analogía del brazo de un robot presentada en la sección anterior, se puede observar cómo el algoritmo produce muchos espacios de proyección posibles sólo limitados por el extenso conjunto de funciones básicas (pesos de la primera capa) generadas por el perceptrón multicapa.

Otra cuestión importante es la selección de las matrices de ponderación del error cuadrático medio en cada una de las capas. Aunque los modelos matemáticos muestran que se deben considerar los efectos de amplificación de la salida de la capa siguiente para seleccionar la matriz de ponderación $\mathbf{G}$, en algunos casos, podría ser mejor utilizar la matriz identidad como se menciona en la nota del paso 4 del algoritmo. La razón de esto es que el modelo matemático usa la asunción de que los pesos de la siguiente capa, que son empleados en la función objetivo de la primera capa, están cerca de los óptimos. Sin embargo, los parámetros de la segunda capa están, en general, lejos del óptimo al comienzo del entrenamiento y, por tanto, sus valores no deberían tener mucha relevancia en el cálculo del conjunto de pesos óptimo de la primera capa. Este hecho se aprecia mejor con un ejemplo práctico. La figura 4.4 muestra un escenario de aprendizaje típico usando el algoritmo de entrenamiento propuesto. El problema empleado fue la predicción de una serie temporal donde los datos de entrenamiento contienen 1000 muestras de la serie caótica de Mackey-Glass. La topología empleada fue 7-5-1 (7 entradas, 5 neuronas ocultas y una neurona de salida) con funciones tangentes hiperbólicas como funciones neuronales en la capa oculta y funciones lineales en la neurona de salida. El número de entradas es consistente con el teorema de Taken sobre la dimensión de embebido de series caóticas [59, 125]. La dos subfiguras contienen la curva de aprendizaje (error cuadrático medio frente al número de iteraciones) obtenida empleando el algoritmo propuesto. La figura 4.4(a) contiene la curva correspondiente al método con matriz de ponderación en la primera



(a)



(b)

Figura 4.4: Curva de aprendizaje para el algoritmo de inicialización de los pesos usando (a) $\mathbf{G^{(1)}} = \mathbf{I}$ y (b) $\mathbf{G^{(1)}} = \mathbf{W}^{(2)T}\mathbf{W}^{(2)}$ como matriz de ponderación.

capa igual a la identidad ($\mathbf{G^{(1)}} = \mathbf{I}$); mientras que la figura 4.4(b) emplea la matriz obtenida teóricamente a partir de los pesos de la segunda capa ($\mathbf{G^{(1)}} = \mathbf{W}^{(2)T}\mathbf{W}^{(2)}$).

En este ejemplo se puede observar que la elección de la matriz de ponderación como la matriz identidad proporciona un error cuadrático medio más pequeño en una sola iteración del algoritmo. Además, en esta figura se puede apreciar el comportamiento estocástico del algoritmo, mencionado previamente. Asimismo, una sugerencia importante para mejorar el rendimiento del algoritmo, para los problemas de clasificación, consiste en añadir un pequeño ruido a la salida deseada como se sugiere en [131]. Esto es muy importante porque en los problemas de clasificación la salida deseada es constante e igual para muchos datos de entrada. Esto provoca que la solución propuesta, basada en un sistema de ecuaciones lineales, presente problemas al resolver el sistema debido a que la matriz de coeficientes puede estar mal condicionada[1]. La introducción del ruido ayuda al algoritmo a solucionar este problema y a determinar el conjunto de pesos que conduce a un error cuadrático medio pequeño en

[1]Esto puede resolverse ortogonalizando (usando polinomios ortogonales, por ejemplo)

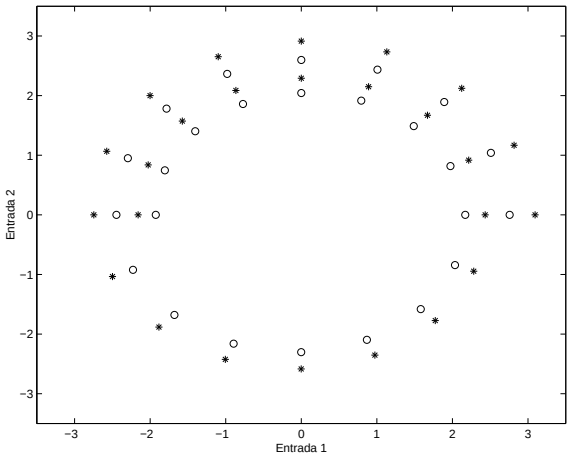


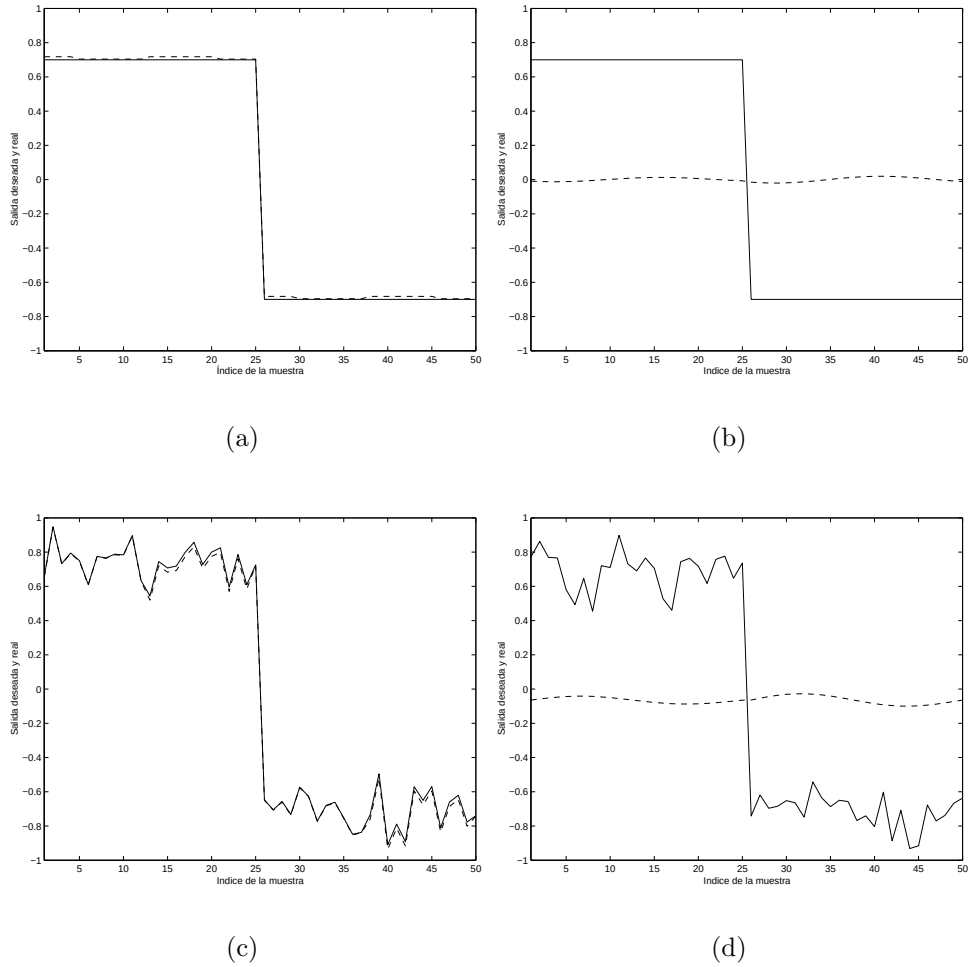
Figura 4.5: Datos del problema de clasificación de la espiral.

pocas iteraciones. Esta hipótesis fue verificada en varios problemas de clasificación, incluyendo el problema del *XOR* generalizado y el de la espiral [45, 46]. A continuación, se muestran los resultados obtenidos para este último problema. La topología de la red empleada fue 2-50-1 con funciones tangentes hiperbólicas tanto en la capa oculta como en la de salida. El conjunto de entrenamiento contenía 50 datos (véase la figura 4.5), con sus correspondientes salidas deseadas. Es importante resaltar que en este problema cuando el número de neuronas en la capa oculta es, al menos, igual al número de datos de entrenamiento, existe una solución para alcanzar una clasificación perfecta, i.e., sin errores. Se realizaron cuatro experimentos diferentes:

- Salida deseada sin ruido y utilizando la matriz de ponderación teórica propuesta en el paso 4 del algoritmo: $\mathbf{G^{(1)}} = \mathbf{W}^{(2)T}\mathbf{W}^{(2)}$.
- Salida deseada sin ruido y empleando la matriz de ponderación $\mathbf{G^{(1)}} = \mathbf{I}$, es decir, empleando el error cuadrático medio no ponderado.
- Salida deseada con ruido aditivo y usando la matriz de ponderación basada en los pesos de la segunda capa de la red: $\mathbf{G^{(1)}} = \mathbf{W}^{(2)T}\mathbf{W}^{(2)}$.
- Salida deseada con ruido aditivo y empleando la matriz de ponderación $\mathbf{G^{(1)}} = \mathbf{I}$.

Los resultados obtenidos se muestran en la figura 4.6. Es evidente, observando las subfiguras 4.6(b) y 4.6(d), que el entrenamiento llevado a cabo empleando como matriz de ponderación la matriz identidad no funciona bien en problemas de clasificación. Se ha llegado a esta conclusión mediante diversas simulaciones con diferentes conjuntos de datos de clasificación. Evidentemente, en este tipo de problemas el factor de ponderación es importante, mientras que lo es menos en problemas de regresión donde la salida deseada presenta una mayor variabilidad. Además, se comprobó en

las pruebas realizadas que aunque ambos métodos (con matriz de ponderación y sin ella) alcanzan un error cuadrático medio muy pequeño y un error de clasificación cero sobre el conjunto de entrenamiento (ver subfiguras 4.6(a) and 4.6(c)), la red multicapa entrenada con ruido en las salidas deseadas alcanza usualmente la solución en un menor número de iteraciones que la misma red entrenada sin ruido.



(a) (b)

(c) (d)

Figura 4.6: Resultados con los datos de la espiral empleando (a) la salida deseada sin ruido y $\mathbf{G^{(1)}} = \mathbf{W}^{(2)T}\mathbf{W}^{(2)}$, (b) la salida deseada sin ruido y $\mathbf{G^{(1)}} = \mathbf{I}$, (c) la salida deseada con ruido y $\mathbf{G^{(1)}} = \mathbf{W}^{(2)T}\mathbf{W}^{(2)}$, y (d) salida deseada con ruido y $\mathbf{G^{(1)}} = \mathbf{I}$.

Finalmente, el último aspecto a destacar está relacionado con el diseño de la inversa de la función neuronal. Al propagar la salida deseada desde la salida a la entrada de la red, a través de las diversas capas de pesos, ésta puede amplificarse hasta alcanzar valores que excedan el rango de la función neuronal de la capa previa ([-1,1] en el caso de funciones tangente hiperbólica). Por ello, cuando se traslada la

salida deseada a través de la función neuronal, como se menciona en el paso 3 del algoritmo, se debe emplear una extensión periódica de la función inversa. Por ejemplo, usando la función tangente hiperbólica, y la tangente como inversa, se soluciona este problema. Para otras funciones de tipo sigmoidal, las funciones inversas pueden extenderse periódicamente en la definición de la inversa.

4.6 Resultados experimentales

A continuación, se muestran los experimentos realizados para verificar el comportamiento del algoritmo de retropropagación de la salida deseada propuesto en este capítulo. Para realizar las simulaciones se emplearon tres conjuntos de datos: la serie temporal de un láser de la competición de Santa Fe [133], el índice bursátil Dow-Jones [91] y el modelo de la presión del motor de un automóvil [109, 44]. Para cada uno de los conjuntos de datos se emplearon redes de neuronas con funciones de activación de tipo sigmoidal en las neuronas de la capa oculta, mientras que en la capa de salida, con una única neurona, se utilizó una función lineal. Además, todos los conjuntos de entrenamiento estuvieron formados por 1000 datos con sus correspondientes salidas deseadas. Las simulaciones se llevaron a cabo empleando el método de Monte Carlo y usando, para cada conjunto de datos, 100 simulaciones con pesos iniciales escogidos aleatoriamente. Estos pesos, fueron empleados tanto para el entrenamiento de la red mediante el algoritmo de retropropagación del error (RE) como para la inicialización basada en el método de retropropagación de la salida deseada (RSD) propuesto en este capítulo. En este último caso se emplearon dos aproximaciones distintas. La primera de ellas consistió en la inicialización sólo de la última capa empleando los pasos 5-7 del algoritmo propuesto durante una iteración (RSD1). La segunda aproximación consistió en la inicialización de ambas capas de la red empleando el algoritmo presentado (desde el paso 1 al 9) durante tres iteraciones (RSD2). En el primer caso, los pesos de la primera capa se igualaron a los valores aleatorios correspondientes a la simulación de Monte Carlo. Esta aproximación permite evaluar la necesidad de la inicialización de los parámetros de la primera capa en el rendimiento final de la red. Además de los métodos mencionados, i.e., RE, RSD1 y RSD2, se entrenó la red empleando una aproximación híbrida basada en la inicialización de todos pesos usando el método propuesto y luego empleando el método de retropropagación del error (RSD+RE). En este último caso se emplearon de nuevo los dos métodos alternativos de inicialización (de una capa o dos) como condición inicial del algoritmo de retropropagación. Puesto que el método de inicialización propuesto proporciona un estado inicial cercano a una solución óptima, el método RSD+RE requiere muchas menos iteraciones, comparado con el algoritmo RE con inicialización aleatoria, para lograr la convergencia hacia la solución. Para los tres conjuntos de datos se empleó el método

de retropropagación para entrenar la red durante 1000, 2000 y 200 iteraciones, respectivamente. Sin embargo, para el caso del algoritmo RSD+RE se emplearon 250, 500 y 50 iteraciones. Para todas las fases del algoritmo de retropropagación del error de empleó la *toolbox* de redes de neuronas de Matlab^{TM} y el número de iteraciones empleadas fueron determinadas experimentalmente para garantizar la convergencia del algoritmo de retropropagación.

4.6.1 Ejemplo con los datos de la serie temporal de la competición de Santa Fe

El primero de los experimentos se llevó a cabo empleando una red de neuronas multicapa para predecir la serie temporal de un láser propuesta en la competición de Santa Fe. Esta serie representa la intensidad de las pulsaciones de un láser de infrarrojos en estado caótico. El objetivo del sistema es predecir el valor de la serie en el instante $t+1$ empleando los valores en los instantes previos t, $t-1$ y $t-2$. La topología de la red empleada en este caso fue 3-11-1. La figura 4.7 muestra la serie temporal empleada en el entrenamiento.

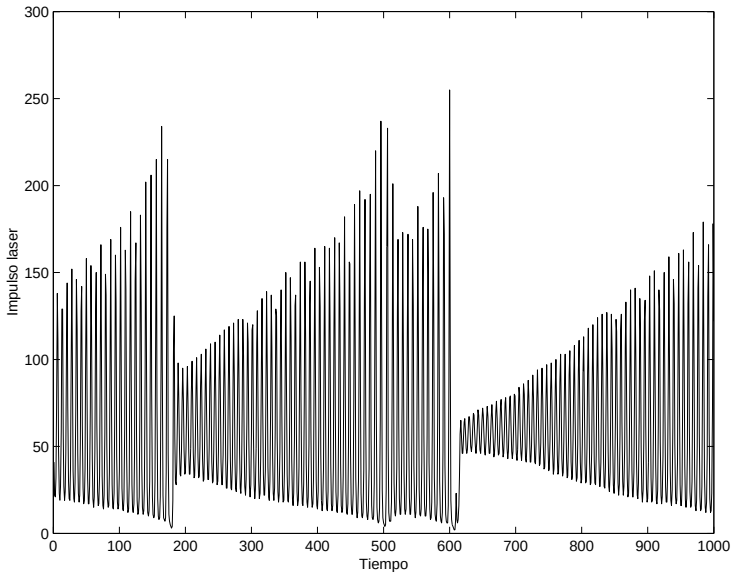


Figura 4.7: Serie temporal del láser.

Los resultados de los experimentos realizados para predecir esta serie temporal se muestran en las figuras 4.8, 4.9 y 4.10.

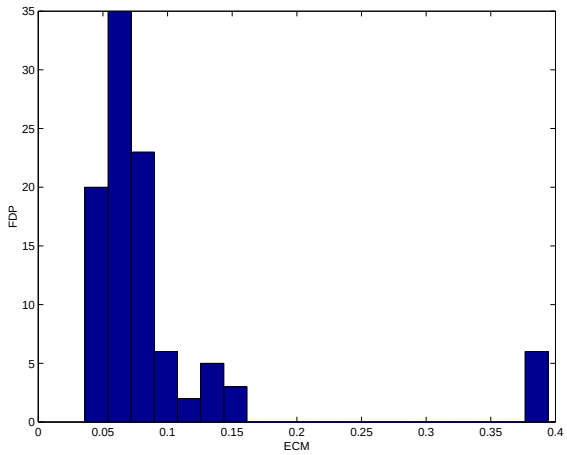


Figura 4.8: Datos del láser: histograma del ECM final para el método RE con inicialización aleatoria.

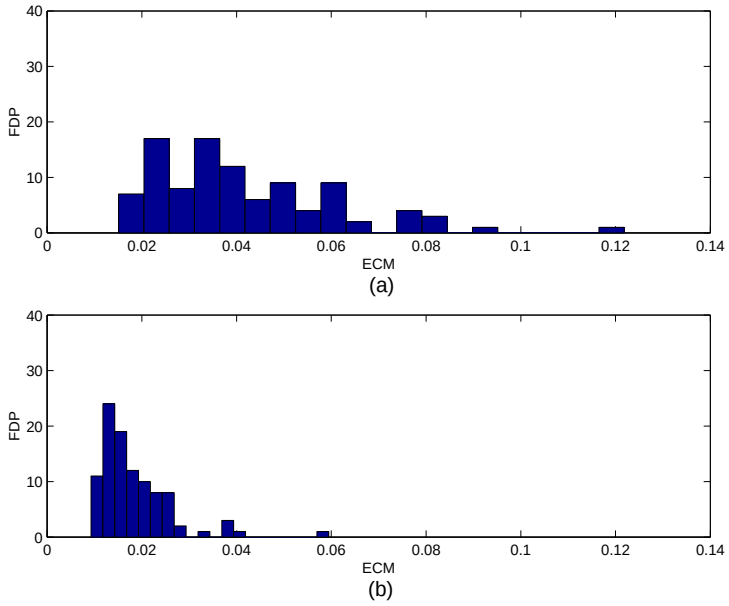


Figura 4.9: Datos del láser: histograma del ECM final para (a) el método RSD para la última capa (RDS1) y (b) el algoritmo de RE usando como pesos iniciales los obtenidos por RDS1.

En concreto, la figura 4.8 muestra los resultados obtenidos para el algoritmo de retropropagación del error empleando la inicialización aleatoria de los pesos. Esta figura contiene el histograma del error cuadrático medio obtenido al final del proceso de aprendizaje (1000 iteraciones) en las 100 simulaciones realizadas. Por otro lado, los resultados obtenidos con los métodos RSD1 y RSD2 se muestran en las figuras

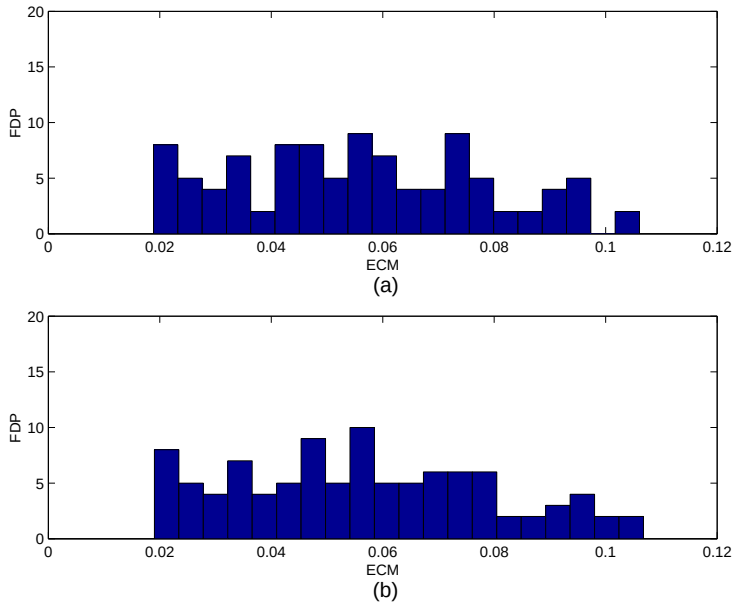


Figura 4.10: Datos del láser: histograma del ECM final para (a) el método RSD para las dos capas (RDS2) y (b) el algoritmo de RE usando como pesos iniciales los obtenidos por RDS2.

4.9(a) y 4.10(a), respectivamente. Además, las figuras 4.9(b) y 4.10(b) muestran los resultados del algoritmo de retropropagación empleando como pesos iniciales los obtenidos por los métodos RSD1 y RSD2, respectivamente. Como se puede observar en estas figuras, los resultados obtenidos no mejoran significativamente los obtenidos previamente por RSD1 y RSD2, lo que indica que la red se encuentra cerca del óptimo o de algún mínimo local. Además, en los resultados obtenidos es importante destacar que el algoritmo de retroprogación de la salida deseada con inicialización aleatoria obtiene mejores resultados en tres iteraciones que el algoritmo de retropropagación del error en 1000 iteraciones.

4.6.2 Ejemplo con los datos del índice Dow-Jones

El segundo conjunto de datos empleado en las simulaciones fue la serie temporal formada por el valor del índice Dow-Jones durante los años 1994-1996. En este caso se empleó una red de neuronas con una topología 5-7-1. El objetivo del sistema es predecir el valor del índice en un día t empleando el valor de los 5 días previos. La figura 4.11 muestra la serie temporal empleada en el entrenamiento de la red.

Los resultados obtenidos empleando el algoritmo de retropropagación del error y el método de inicialización de los pesos propuesto se muestran en las figuras 4.12, 4.13 y 4.14. La figura 4.12 contiene el histograma construido a partir del error cuadrático

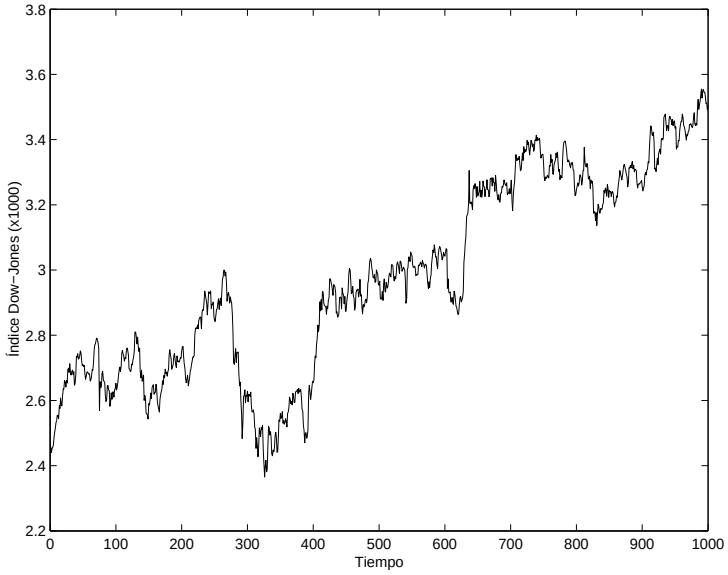


Figura 4.11: Serie temporal del índice bursátil Dow-Jones.

medio obtenido al final del entrenamiento en las 100 simulaciones de Monte Carlo.

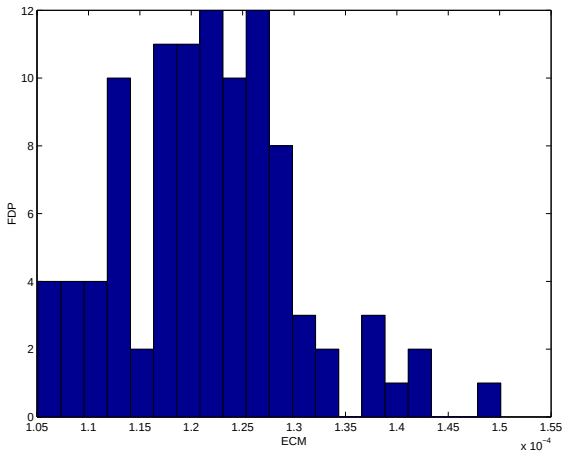


Figura 4.12: Datos del índice Dow-Jones: histograma del ECM final para el método RE con inicialización aleatoria.

La figura 4.13 presenta el histograma del método de inicialización de los pesos empleado sólo para la última capa de la red (RSD1). Finalmente, la figura 4.14 contiene el histograma del método de inicialización de los pesos empleado para todas las capas de la red (RSD2).

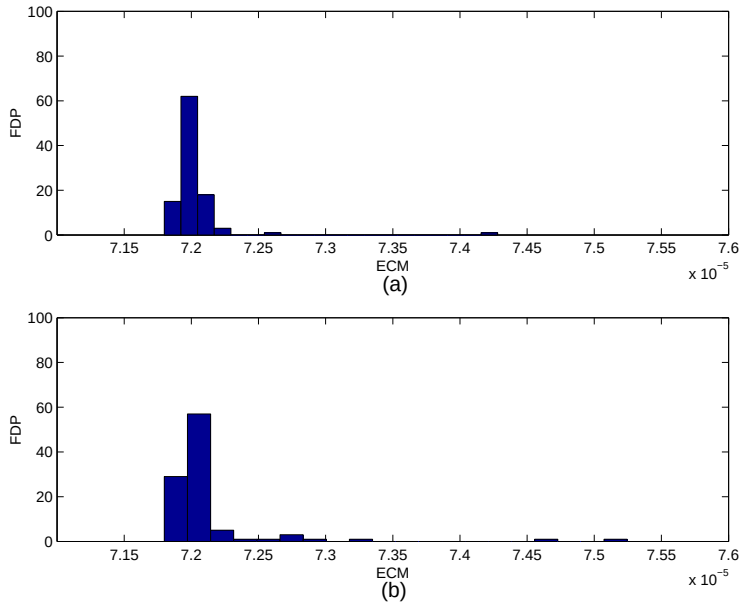


Figura 4.13: Datos del índice Dow-Jones: histograma del ECM final para (a) el método RSD para la última capa (RSD1) y (b) el algoritmo de RE usando como pesos iniciales los obtenidos por RDS1.

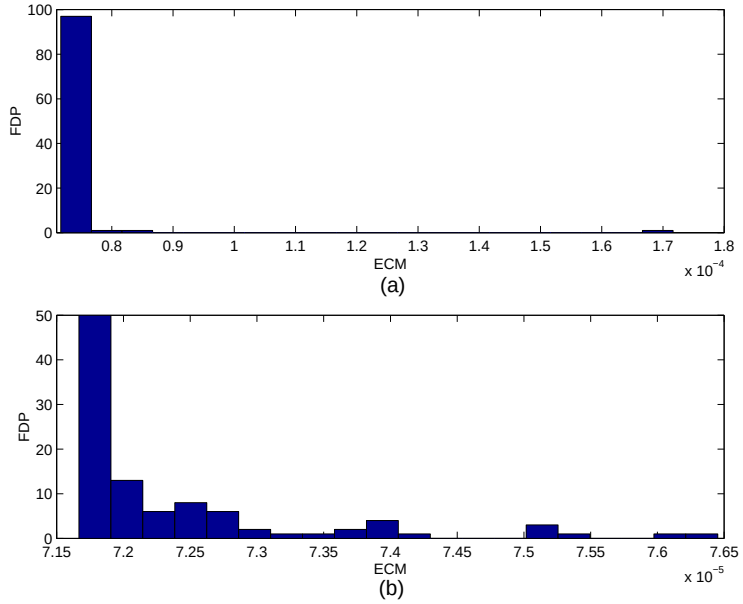


Figura 4.14: Datos del índice Dow-Jones: histograma del ECM final para (a) el método RSD para las dos capas (RSD2) y (b) el algoritmo de RE usando como pesos iniciales los obtenidos por RDS2.

En las gráficas presentadas se puede observar que los métodos de inicialización

RSD1 y RSD2 presentan unos resultados similares. Además al aplicar el método de retropropagación del error no mejora significativamente el error obtenido, por tanto, los valores proporcionados por el método de inicialización de los pesos están cerca de los óptimos o cerca de un mínimo local. También se puede apreciar que el método de retropropagación del error inicializado con pesos aleatorios siempre obtiene un error significativamente mayor que los obtenidos con el método propuesto. Esto parece indicar que el algoritmo de retropropagación ha quedado atrapado cerca de un mínimo local.

4.6.3 Ejemplo con los datos de un motor

Finalmente, en la última de las simulaciones se empleó una red de neuronas para modelar la dinámica no lineal del motor de un automóvil. La entrada del sistema es el ángulo del acelerador que controla el caudal de aire en el colector, y el objetivo es predecir la presión del motor en el instante actual usando cuatro datos de entrada:

- x_1 y x_2: valor valor actual y anterior de la entrada del motor (ángulo del acelerador que controla la cantidad de flujo de aire en el colector).

- x_3 y x_4: salida del motor (presión) en los instantes $t-1$ y $t-2$.

La topología de la red empleada fue 4-5-1. La figura 4.15 muestra la serie temporal empleada en el entrenamiento.

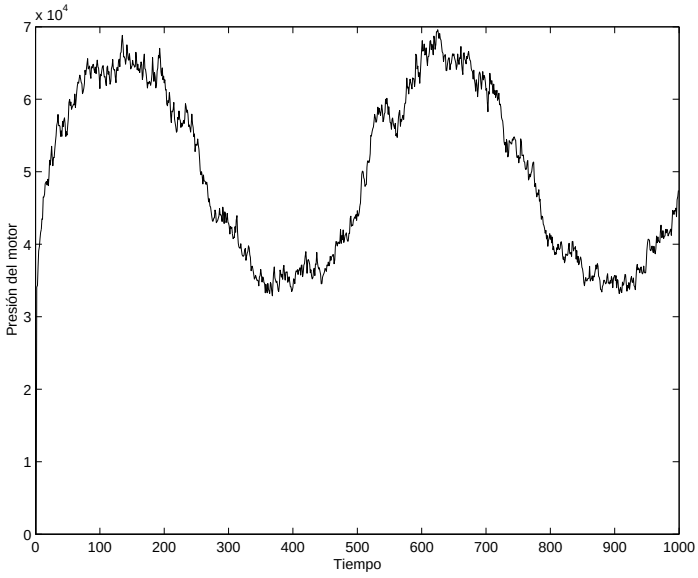


Figura 4.15: Serie temporal de la presión ejercida en el motor de un automóvil.

Los resultados obtenidos para el algoritmo de retropropagación del error empleando pesos inicializados aleatoriamente se muestran en la figura 4.16. Esta figura contiene el histograma del error cuadrático medio obtenido al final del entrenamiento en las 100 simulaciones.

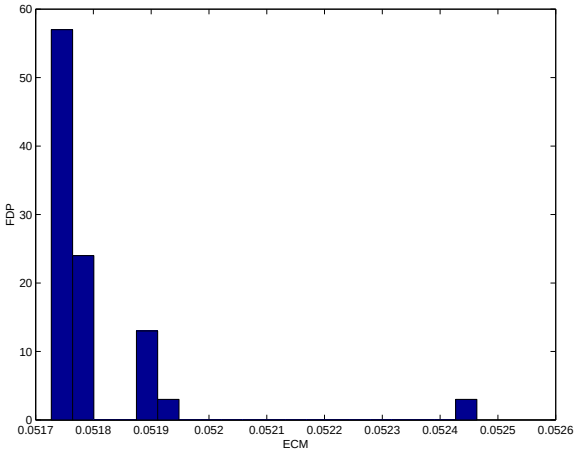


Figura 4.16: Datos del motor: histograma del ECM final para el método RE con inicialización aleatoria.

Asimismo, las figuras 4.17 y 4.18 presentan los mismos resultados pero para el método de inicialización propuesto para la última capa (RSD1) y para ambas capas (RSD2), respectivamente. Es importante resaltar que en este caso, el método RSD1 obtiene un error cuadrático medio cercano a 5×10^{-2} (similar al de RE con inicialización aleatoria) mientras que el RSD2 obtiene unos resultados mucho mejores, ya que el error es del orden de 10^{-5}. Además, y como en los ejemplos anteriores, el uso posterior del algoritmo de retropropagación no mejora significativamente el error obtenido por los métodos de inicialización. Asimismo, los resultados obtenidos con el método de retropropagación empleando pesos iniciales aleatorios muestran que el método ha quedado atrapado en un mínimo local sobre un nivel de error de 5×10^{-2}. Estos resultados demuestran claramente que la inicialización de ambas capas empleando el método de inicialización propuesto es mejor que la inicialización sólo de la capa de salida y que el algoritmo de retropropagación con pesos aleatorios.

4.7 Discusión

En este capítulo se ha propuesto un nuevo algoritmo de inicialización de los pesos para redes de neuronas multicapa basado en la *retropropagación de la salida deseada* que permite obtener, mediante un sistema de ecuaciones lineales, los pesos de cada capa. En los experimentos realizados, se demostraron las ventajas del método de

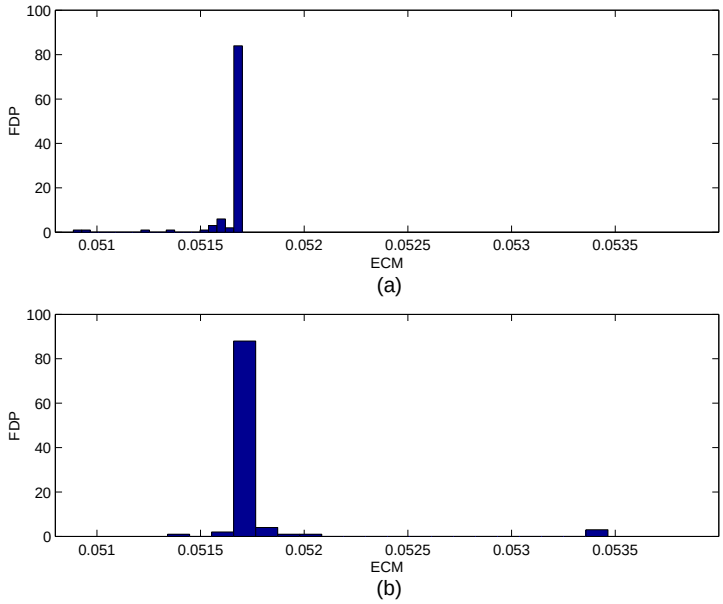


Figura 4.17: Datos del motor: histograma del ECM final para el método RSD para la última capa.

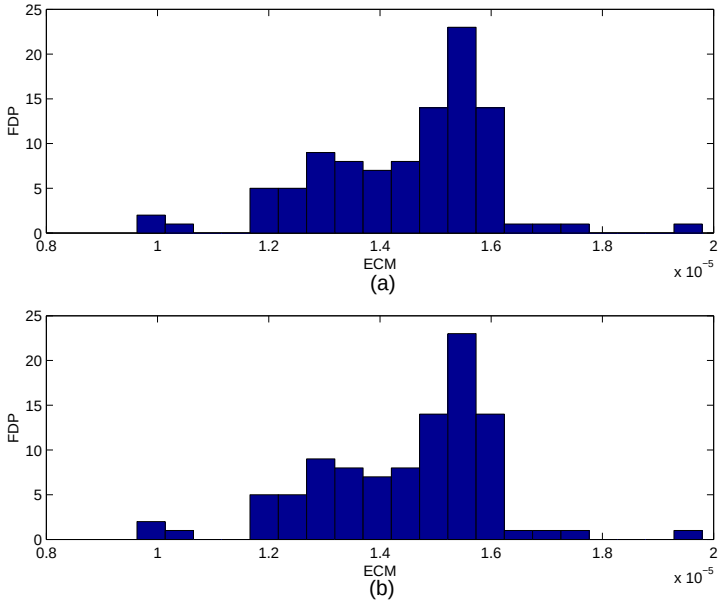


Figura 4.18: Datos del motor: histograma del ECM final para el método RSD para las dos capas.

inicialización propuesto frente a una inicialización aleatoria para el entrenamiento de redes de neuronas mediante el método de retropropagación del error. Además, en los dos primeros ejemplos no se apreció una gran diferencia entre la inicialización de ambas capas o sólo la de la capa de salida. Sin embargo, en el tercer experimento realizado (datos de la presión del motor) se comprobaron claramente las ventajas de la inicialización de ambas capas sobre la inicialización únicamente de la última capa.

Las principales características del algoritmo propuesto en este capítulo son las siguientes:

- Requiere un coste computacional muy pequeño comparado con los métodos actuales. Esto es debido a que para estimar los pesos óptimos de cada capa es necesario resolver sólo un sistema de ecuaciones lineales. Este cálculo necesita muchos menos recursos computacionales, memoria y carga del procesador, que el necesario para estimar las derivadas de primer y segundo orden de la función de error.

- Permite obtener una buena solución en muy pocas iteraciones (2 a 5 habitualmente). Este hecho, unido a lo mencionado en el punto anterior, hace que el algoritmo sea muy apropiado para aquellas aplicaciones que requieran una respuesta eficaz y muy rápida, como en sistemas de control o monitorización en tiempo real.

- Se ha comprobado en las simulaciones realizadas que si se entrena el algoritmo de retropropagación del error empleando a) pesos inicializados aleatoriamente y b) el método de inicialización de pesos propuesto, en el segundo caso se obtienen soluciones equivalentes en un menor número de iteraciones. En concreto, en los tres experimentos realizados se comprobó que el algoritmo desarrollado alcanza mejores resultados en 3 iteraciones que el método de retropropagación del error en 1000, 2000 y 200 iteraciones, respectivamente.

- El algoritmo no converge hacia un mínimo al final del aprendizaje. Esto es debido a que no es posible establecer algún mecanismo que determine que el error obtenido en la iteración actual sea menor que el obtenido en la anterior. Esto provoca que la búsqueda sea estocástica en el espacio de pesos. Por ello, el algoritmo propuesto proporciona como resultado final el que sea mejor en todas las iteraciones realizadas.

Capítulo 5

Algoritmo de aprendizaje para redes multicapa

"*Quizá la existencia de una respuesta depende solamente de que se haga la pregunta adecuada*"

Robert Duncan

5.1 Introducción

En el capítulo anterior se ha presentado un nuevo algoritmo para la inicialización de los pesos en una red de neuronas multicapa con alimentación hacia delante. Se ha comprobado que el método desarrollado es muy eficiente pero, sin embargo, no está garantizada la convergencia hacia un mínimo de la función de error al final del proceso de entrenamiento. Como hemos visto anteriormente, este inconveniente se debe a que no es posible guiar al algoritmo para que el error obtenido en la iteración actual sea menor que en la iteración anterior. Por lo tanto, la búsqueda es estocástica en el espacio de pesos. El siguiente objetivo de este trabajo se centró en el desarrollo de un nuevo método de aprendizaje que permita emplear los resultados desarrollados para redes de una capa (capítulo 3) en redes multicapa. Al igual que en el capítulo anterior, en este caso se aborda el problema del aprendizaje de redes de neuronas de dos capas. A pesar de ello, el método presentado en este capítulo puede generalizarse fácilmente para cualquier número de capas.

En los métodos actuales para el aprendizaje de redes de neuronas con alimentación hacia delante se emplea la misma regla de actualización de los pesos para ambas capas. Estos métodos calculan el error actual de la red, $\boldsymbol{\varepsilon} = \mathbf{d} - \mathbf{y}$, a partir del cual utilizan cierta información de primer orden (gradiente, $\mathbf{g}$) y/o de segundo orden (hessiana,

H) para actualizar los pesos mediante una regla de optimización preestablecida y específica para cada método. Este proceso se muestra de forma esquemática en la figura 5.1. Mediante este procedimiento, estos métodos actualizan las funciones básicas de la red (pesos de la primera capa) al mismo tiempo que las proyecciones (pesos de la segunda capa). Por tanto, la optimización de ambas capas de pesos se realiza simultáneamente. Sin embargo, en este capítulo se presentará un nuevo método que emplea dos procedimientos distintos para la optimización de los pesos de la primera y la segunda capa. Este método, en cada iteración, optimiza en primer lugar las funciones básicas de la red para luego obtener las proyecciones óptimas para éstas.

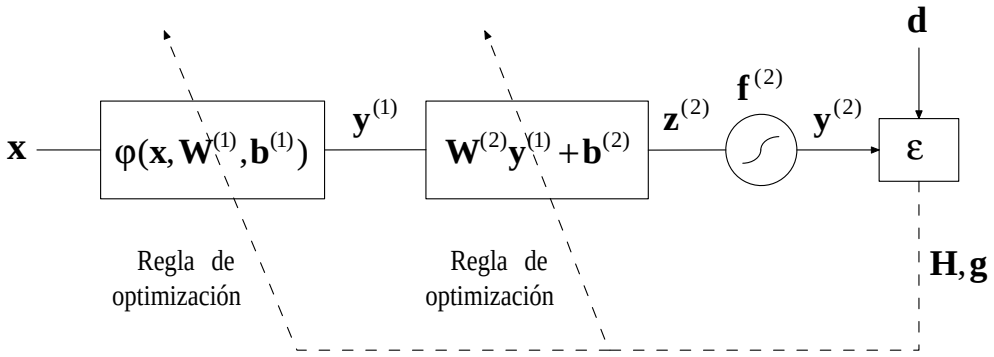


Figura 5.1: Red de neuronas con alimentación hacia delante de dos capas en la cual la actualización de todos los pesos se realiza con la misma regla de optimización.

5.2 Algoritmo propuesto

El nuevo algoritmo desarrollado, ilustrado esquemáticamente en la figura 5.2, emplea dos métodos de aprendizaje diferentes para la actualización de los pesos: a) cualquiera de los métodos estándar de aprendizaje para los pesos de la primera capa, y b) el sistema de ecuaciones lineales para redes de una capa, presentado en el tercer capítulo (ecuación (3.11)), para los pesos de la segunda capa.

En este algoritmo se ha empleado el sistema de ecuaciones lineales (3.11), en el cual la variable vectorial $\mathbf{x}$, que se correspondía con la entrada en una red de neuronas de una capa, se ha reemplazado por la salida de primera capa $\mathbf{y}^{(1)}$. El método propuesto funciona de la siguiente manera: en las primera fase del entrenamiento, i.e., durante las primeras iteraciones, todas las capas de la red se optimizan empleando una de las reglas actuales de optimización, como se muestra en la figura 5.1; en la segunda fase, cuando el error de la red permanece más o menos estable durante un par de iteraciones, el proceso de actualización cambia al mostrado en la figura 5.2. En este último caso los pesos de la primera capa se siguen actualizando con la

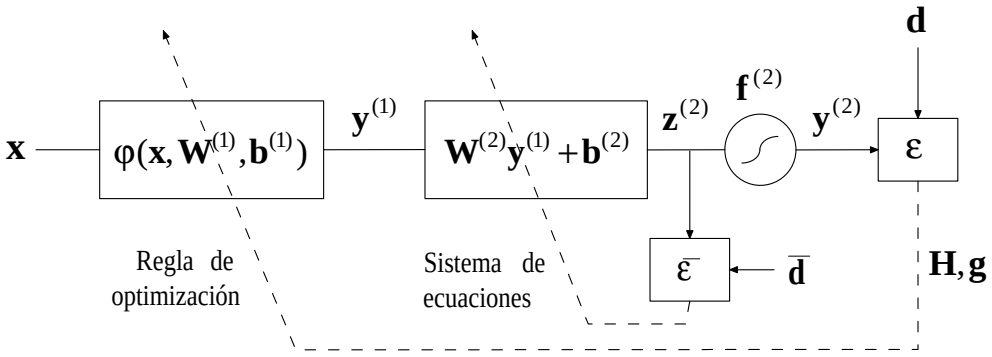


Figura 5.2: Red de neuronas con alimentación hacia delante de dos capas en la cual la actualización de los pesos de la primera capa se realiza con una regla de optimización estándar y los de la segunda mediante un sistema de ecuaciones lineales.

regla de optimización estándar, sin embargo, los pesos de la última capa se obtienen óptimamente, empleando el sistema de ecuaciones lineales, para los pesos actuales de la primera capa.

En concreto, el algoritmo propuesto se detalla a continuación:

1. *Seleccionar valores aleatorios para* $\mathbf{W}^{(1)}$, $\mathbf{b}^{(1)}$, $\mathbf{W}^{(2)}$ *y* $\mathbf{b}^{(1)}$.

2. *Actualizar los pesos y los umbrales usando cualquier regla de optimización estándar:*

   - $\mathbf{W}^{(1)}(n+1) = \mathbf{W}^{(1)}(n)+\Delta\mathbf{W}^{(1)}(n+1)$; $\mathbf{b}^{(1)}(n+1) = \mathbf{b}^{(1)}(n)+\Delta\mathbf{b}^{(1)}(n+1)$
   - $\mathbf{W}^{(2)}(n+1) = \mathbf{W}^{(2)}(n)+\Delta\mathbf{W}^{(2)}(n+1)$; $\mathbf{b}^{(2)}(n+1) = \mathbf{b}^{(2)}(n)+\Delta\mathbf{b}^{(2)}(n+1)$

   donde $\Delta\mathbf{W}^{(1)}(n+1)$, $\Delta\mathbf{b}^{(1)}(n+1)$, $\Delta\mathbf{W}^{(2)}(n+1)$ *y* $\Delta\mathbf{b}^{(2)}(n+1)$ *están determinados por la regla de optimización empleada.*

3. *Evaluar el criterio de parada. Si no se satisface, entonces ir al paso 4; en otro caso, ir al paso 7.*

4. *Calcular el valor de la función de coste en la iteración actual* ($C(n)$). *Si* $|C(n)-C(n-1)| < \lambda$ *(donde* λ *es un umbral fijo) entonces is al paso 5; en otro caso, ir al paso 2.*

5. *Actualizar los pesos y umbrales de la primera capa usando la regla de optimización estándar y obtener los pesos y umbrales de la segunda capa utilizando el método de mínimos cuadrados:*

   - $\mathbf{W}^{(1)}(n+1) = \mathbf{W}^{(1)}(n)+\Delta\mathbf{W}^{(1)}(n+1)$; $\mathbf{b}^{(1)}(n+1) = \mathbf{b}^{(1)}(n)+\Delta\mathbf{b}^{(1)}(n+1)$

- $\mathbf{W}^{(2)}(n+1)$ *y* $\mathbf{b}^{(2)}(n+1)$ *se obtienen empleando el sistema de ecuaciones (3.11). En este caso, el vector* $\mathbf{x}$ *en la ecuación (3.11) (que representa los datos de entrada en la red de una capa) se sustituye por la salida de la primera capa, i.e.,* $\mathbf{y}^{(1)}$.

6. *Evaluar el criterio de parada. Si no se cumple, ir al paso 5; en otro caso, ir al paso 2.*

7. *Fin del aprendizaje.*

En el algoritmo presentado es importante mencionar algunas cuestiones interesantes. La primera de ellas está relacionada con el criterio de parada del algoritmo. Como puede observarse existen dos criterios de parada distintos, uno asociado con la primera etapa del algoritmo (actualización de ambas capas mediante una regla estándar) y otro con el método híbrido. Sin embargo, como se puede apreciar, el único criterio que determina el fin del entrenamiento es el asociado con la primera etapa, mientras que el otro criterio sólo fuerza un cambio al otro método y no una finalización del algoritmo. El objetivo es que sea el algoritmo estándar, y no el método híbrido, el que determine cuándo no es posible mejorar la función de error. Esto se debe a que el método híbrido, al conseguir unas proyecciones óptimas para las bases actuales (pesos de la primera capa), obtiene muchas veces una mejora pequeña en las siguientes iteraciones del algoritmo, debido a un cambio insignificante en las bases previas, lo que puede provocar una parada temprana del algoritmo. Otra cuestión a mencionar, derivada de lo explicado anteriormente, es que el algoritmo puede alternar entre ambos métodos más de una vez, ya que el criterio de parada del método híbrido devuelve el control al método clásico, pudiéndose producir varios ciclos de conmutaciones entre ambos métodos. Otro hecho a destacar es que el algoritmo desarrollado se ha presentado usando un criterio de conmutación (paso 4) basado en la diferencia entre el error en el instante actual y el instante previo. Sin embargo, se pueden emplear otros criterios alternativos, como por ejemplo, cuando el error supere un determinado umbral, se alcance una iteración prefijada, etc.

Finalmente, aunque se ha limitado el algoritmo para dos capas, la idea es fácilmente ampliable a cualquier número capas. En este caso, la última capa seguiría actualizándose empleando el sistema de ecuaciones lineales, mientras que el resto de las capas se actualizarían con la regla de aprendizaje estándar. Sin embargo, esto no es necesario, en general, puesto que, como se ha mencionado anteriormente, un perceptrón multicapa con una única capa oculta y suficiente número de neuronas ocultas puede aproximar cualquier función continua y diferenciable.

5.3 Discusión del método

Para comprender mejor el comportamiento y las ventajas del algoritmo propuesto, se discutirá a continuación su funcionamiento empleando algunos ejemplos, ilustrados en la figura 5.3.

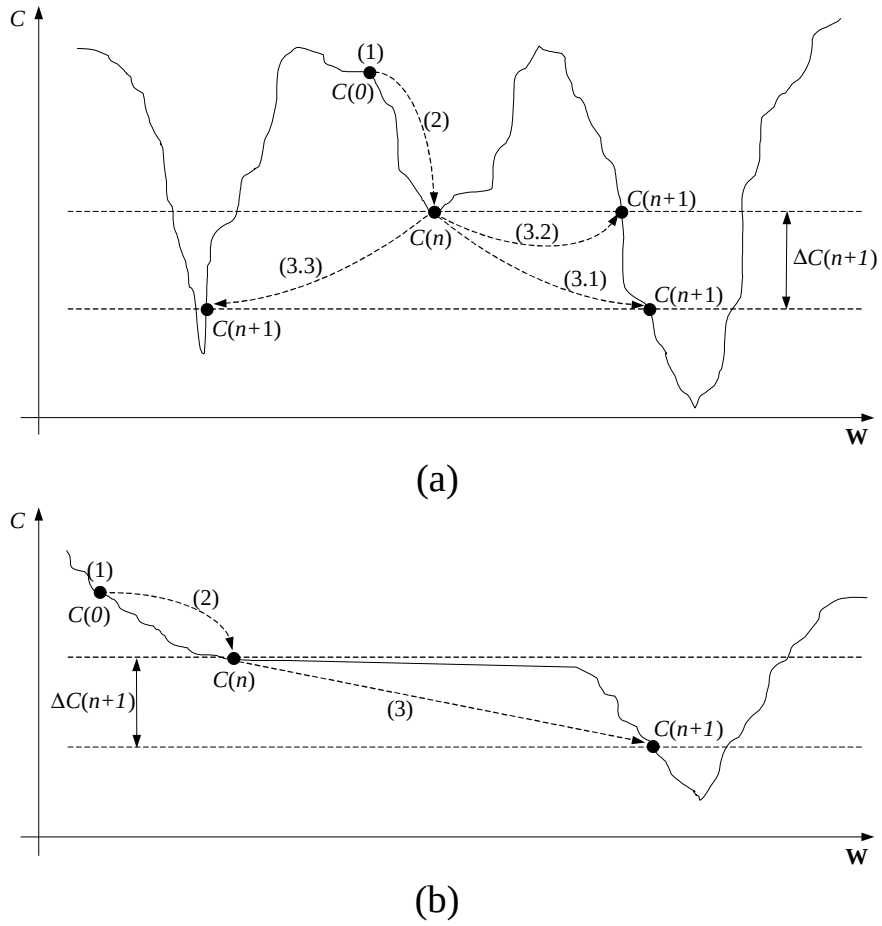


Figura 5.3: Ejemplos del comportamiento del algoritmo que permite (a) evitar mínimos locales y (b) acelerar la velocidad de convergencia.

En el primer paso del algoritmo propuesto los pesos se inicializan aleatoriamente, lo cual está representado gráficamente por la condición inicial indicada por el punto (1) en la figura. A continuación, el algoritmo minimiza la función de coste (pasos 2 a 4) hasta que éste queda atrapado en un mínimo local (punto (2) en la figura 5.3(a)) o alcanza una meseta (punto (2) en la figura 5.3(b)). En estos casos, el algoritmo propuesto se mueve del paso 4 al 5. Posteriormente, en la siguiente iteración del algoritmo, los pesos de la segunda capa ($\mathbf{W}^{(2)}$) se obtienen óptimamente, empleando

el sistema de ecuaciones (3.11) propuesto en el capítulo 3, para los correspondientes valores de los pesos de la primera capa ($\mathbf{W}^{(1)}$). Esto produce un decremento (ΔC) en la función de coste asociada a la mejora obtenida cuando los pesos actuales $\mathbf{W}^{(2)}(n)$, no óptimos en general, son sustituidos por los pesos óptimos $\mathbf{W}^{(2)}(n+1)$. Este cambio en los pesos de la segunda capa puede producir un "salto" a un punto que estará más cerca del óptimo global en la superficie de error. La ventaja del método propuesto es que este "salto" está controlado, puesto que el nuevo punto en la superficie de error ((3.1), (3.2) o (3.3)) está situado, al menos, en el mismo nivel de error (línea superior discontinua), i.e., $C(n+1) \leq C(n)$. Por tanto, existen dos posibilidades, mostradas gráficamente en la figura 5.3(a):

- El "salto" producido mueve al sistema desde el punto (2) a otro punto que está situado en un nivel de error más bajo, $C(n+1) < C(n)$. En este caso puede ocurrir que:
  - El nuevo punto esté cerca del óptimo global y, por tanto, el método de optimización lo alcanzará. Este es el caso mostrado con el punto (3.1) en la figura.
  - El método ha evitado un mínimo local pero el nuevo punto está cerca de otro mínimo local (punto (3.3) en la figura). En este caso el mínimo global no se alcanzará en este paso del algoritmo, pero se mejora el valor de la función de error. Además, el método propuesto permite diversas conmutaciones (cambios entre el paso 4 y 5) durante el proceso de aprendizaje, por tanto, es posible que un "salto" en una iteración posterior aproxime al sistema cerca del óptimo global.
- El "salto" mueve a la red desde el punto (2) a otro punto con el mismo nivel de error, i.e., $C(n+1) = C(n)$ (punto (3.2) en la figura). Este no es un caso muy probable, al menos en el primer cambio entre el paso 4 y 5, puesto que los algoritmos de aprendizaje minimizan la función de coste actualizando los pesos de todas las capas al mismo tiempo y, por lo tanto, es improbable que las proyecciones (pesos de la segunda capa) sean óptimas para los pesos de las funciones básicas (pesos de la primera capa). De nuevo, como en el caso anterior, existen dos posibilidades: el cambio mueve los pesos en la dirección del óptimo global o a otro punto cercano a un mínimo local y, por consiguiente, se puede emplear el mismo razonamiento que en el punto anterior.

Como se verá posteriormente en la sección de resultados, este "salto" provoca un decremento drástico de la función de error. Este decremento, dependiendo de la situación, puede proceder del cambio en los pesos de la primera o de la segunda capa:

- En el primer caso, el contorno de la superficie de error en un punto determinado, después de un "salto", puede producir que la actualización de los pesos de la primera capa, empleando una de las reglas de optimización estándar, conlleve un decremento sustancial del error. Esto se corresponde con una rápida corrección de las funciones básicas influenciada por un cambio abrupto en las proyecciones.

- En el segundo caso, la razón del decremento del error proviene del procedimiento de optimización lineal en sí mismo, que provoca un cambio brusco en las proyecciones hacia los valores óptimos para las funciones básicas actuales (pesos de la primera capa).

La figura 5.4 contiene otro ejemplo que demuestra cómo el algoritmo es capaz de evitar los mínimos locales y acelerar la convergencia hacia la solución.

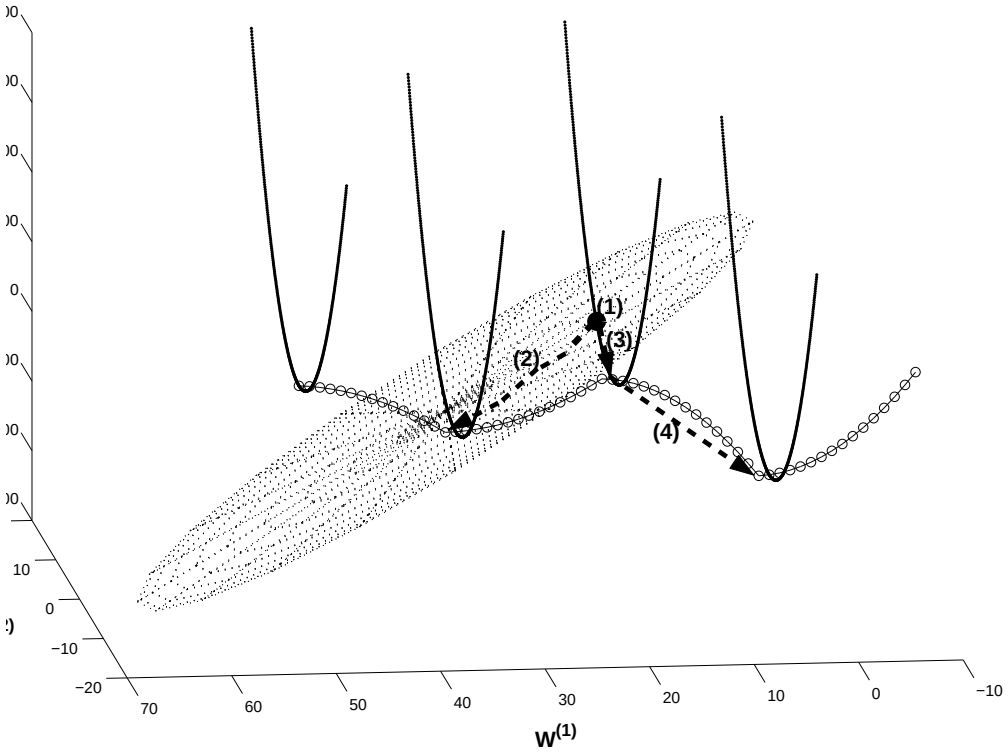


Figura 5.4: Ejemplo de superficie de error con un mínimo local y un mínimo global.

En esta figura se muestra una superficie de error C en función de los pesos de la primera y la segunda capa ($\mathbf{W}^{(1)}$ y $\mathbf{W}^{(2)}$). En esta función de error dado un valor fijo de $\mathbf{W}^{(2)}$ existe una curva representada por la sucesión de puntos (círculos) en la figura. Como se puede observar, esta curva contiene además del óptimo global un mínimo local. Cada uno de los puntos representados es el mínimo de la parábola correspondiente a la solución del problema de mínimos cuadrados dado por el sistema

de ecuaciones lineales (en la figura sólo se muestran cuatro parábolas por motivos de claridad). Además la elipse dibujada representa, figurativamente, el área de atracción del mínimo local. Si se emplea cualquier método de optimización basado en descenso de gradiente al alcanzar, por ejemplo, el punto (1) el algoritmo se verá inevitablemente atraído por el mínimo local (indicado por la flecha señalada con el (2)). Sin embargo, si se emplea el método de aprendizaje híbrido, al estar en el punto (1) se obtendrán los pesos $\mathbf{W}^{(2)}$ óptimos para el valor actual de $\mathbf{W}^{(1)}$ y por tanto se desplazará al punto indicado por la flecha (3) que se corresponde con el mínimo de la parábola. El nuevo punto obtenido no está en el área de atracción del mínimo local sino del mínimo global y, por lo tanto, en las siguientes iteraciones del algoritmo (indicado por la flecha (4)) se alcanzará el óptimo global, evitando así el mínimo local.

El algoritmo propuesto es una aproximación híbrida que combina las propiedades ventajosas de los métodos de optimización local y global. Los métodos locales (e.g., descenso de gradiente, Levenberg-Marquardt) preservan toda la información sobre la historia del proceso de aprendizaje, mientras que las aproximaciones globales (e.g., enfriamiento simulado, algoritmos genéticos) realizan una búsqueda estocástica, tan eficiente como sea posible, en un área extensa del espacio de pesos. El método propuesto en este capítulo asiste a los algoritmos actuales empleando estas cualidades mediante la aplicación de "saltos" controlados que imitan la búsqueda estocástica de los métodos globales.

Hasta el momento, hemos visto cómo el algoritmo es capaz de evitar mínimos locales (ejemplo en la figura 5.3(a)) y acelerar la velocidad de convergencia (ejemplo en la figura 5.3(b)) de cualquier método de optimización para el aprendizaje de redes de neuronas hacia delante. Adicionalmente, es importante destacar que la aproximación desarrollada podría emplearse ignorando los pasos 2 a 4. Esto significa que, durante todo el proceso de entrenamiento, i.e., desde la primera iteración del algoritmo, los pesos de la segunda capa se obtienen usando el sistema de ecuaciones presentado en el tercer capítulo. Este método alternativo proporciona una buena aproximación, ya que alcanza un error pequeño en muy pocas iteraciones del proceso de aprendizaje (en la sección 5.4 se mostrarán algunos resultados al respecto), sin embargo, tiene el inconveniente de que como los pesos están optimizados aleatoriamente, las proyecciones obtenidas (pesos de la segunda capa) son óptimas para unas bases aleatorias (pesos de la primera capa) que, en general, están lejos de las óptimas. Por tanto, resulta arduo para el algoritmo obtener las bases óptimas empleando esta aproximación. Por esta razón, durante las primeras iteraciones del algoritmo se propone el uso de una de las reglas de optimización estándar para el aprendizaje de todos los pesos de la red (ver figura 5.1), con el objetivo de conseguir una buena aproximación de las bases para, posteriormente, cambiar al método híbrido (ver figura 5.2).

5.4 Resultados experimentales

En esta sección se muestran los resultados comparativos entre el método propuesto y el algoritmo de Levenberg-Marquardt. Este algoritmo fue usado como estándar de comparación porque es considerado como el método más rápido para redes de neuronas con un número moderado de pesos (hasta varios cientos). Este hecho ha sido corroborado también en los resultados experimentales obtenidos en el tercer capítulo, en la sección 3.5.5. Por tanto, los términos $\Delta\mathbf{W}^{(1)}(n+1)$ y $\Delta\mathbf{W}^{(2)}(n+1)$ en el paso 2, y el término $\Delta\mathbf{W}^{(1)}(n+1)$ en el paso 5 del método propuesto son iguales a $(\mathbf{H}+\mu\mathbf{I})^{-1}\mathbf{g}$ en cada una de las iteraciones del algoritmo. Además, en todas las simulaciones se emplearon funciones sigmoidales en las neuronas de todas las capas y $\mu = 0.01$ como paso de aprendizaje inicial. Para realizar el estudio comparativo se emplearon tres conjunto de datos: el índice bursátil Dow-Jones, la serie caótica de Mackey-Glass y la serie temporal de un láser empleada en la competición de Santa Fe. Para cada uno de los datos se llevaron a cabo 100 simulaciones diferentes, escogiendo de forma aleatoria los pesos iniciales. Una cuestión importante a destacar es que aunque los pesos se seleccionaron aleatoriamente, se emplearon los mismos para el algoritmo de Levenberg-Marquardt y el método propuesto y, por tanto, ambos parten de la misma condición inicial, es decir, desde el mismo punto en la superficie de error.

5.4.1 Ejemplo con los datos del índice Dow-Jones

El primer conjunto de datos empleado en las simulaciones fue la serie temporal del índice Dow-Jones [91]. El objetivo de la red es el de predecir el próximo valor de la serie basándose en los cinco valores previos. En este caso, la topología de la red empleada fue 5-7-1, y para realizar el entrenamiento del sistema se emplearon 1000 muestras. La figura 5.5 contiene los resultados obtenidos para este conjunto de datos empleando el algoritmo de Levenberg-Marquardt estándar, subfigura (a), y el método híbrido, subfigura (b), propuesto en este capítulo. Los histogramas representan el error cuadrático medio en la última iteración del entrenamiento (500) para cada una de las aproximaciones. Como se puede observar, el algoritmo de Levenberg-Marquardt obtiene valores del error cuadrático medio en el rango $[2.75 \times 10^{-4}, 3.54 \times 10^{-4}]$. Sin embargo, los resultados alcanzados con la aproximación híbrida son mejores porque los valores del error obtenido se encuentran en el intervalo $[2.59 \times 10^{-4}, 2.77 \times 10^{-4}]$. Las figuras 5.6 y 5.7 contienen las curvas de error obtenidas durante el proceso de aprendizaje para el algoritmo de Levenberg-Marquardt y el método híbrido, respectivamente.

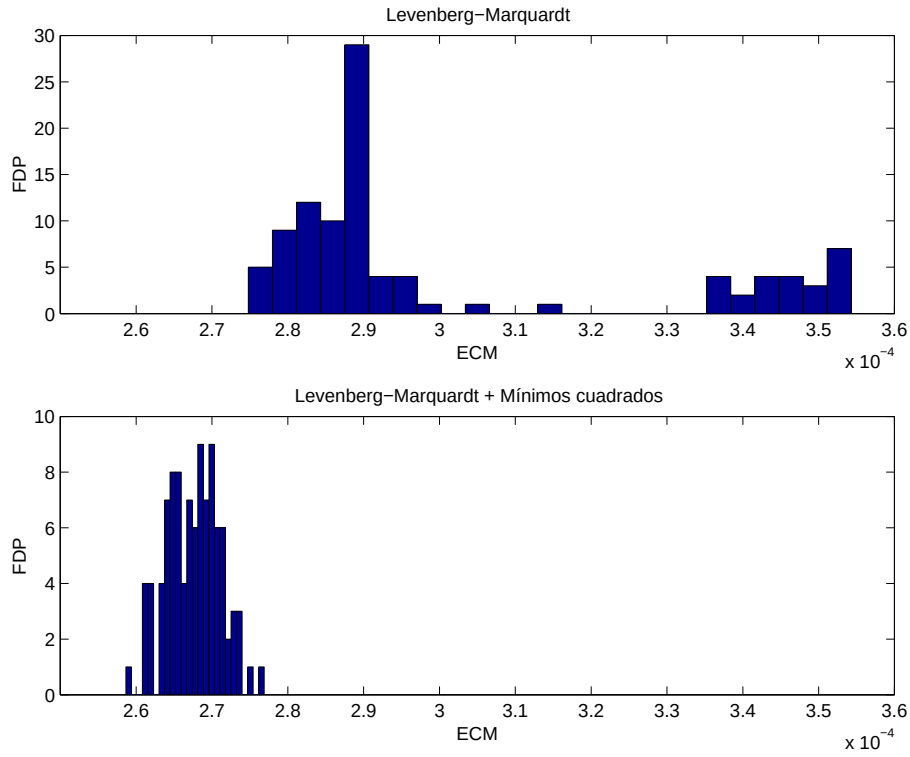


Figura 5.5: Función de densidad de probabilidad (FDP) para los datos de Dow-Jones empleando (a) el algoritmo de Levenberg-Marquardt y (b) el método híbrido.

En este caso, el cambio entre el método de Levenberg-Marquardt y el híbrido (paso 4 y 5 del algoritmo propuesto) se realiza sobre la iteración 120 del entrenamiento de la red. Es importante resaltar que estas figuras contienen sólo los valores del error cuadrático medio después de la iteración 80, para mostrar en mayor detalle las últimas iteraciones del algoritmo (cuando el error es pequeño). Se puede observar, que el algoritmo de Levenberg-Marquardt requiere, para todas las simulaciones, un mayor número de iteraciones para converger hacia el error final. Además, la figura 5.7 muestra que el método híbrido es capaz de reducir el error muy rápidamente (obsérvese el decremento abrupto del error sobre la iteración 120) en todos los casos. Esto se debe a que las bases de la red (pesos de la primera capa) son actualmente una buena aproximación de las óptimas, pero las proyecciones necesitan todavía ser optimizadas para las bases actuales. El método propuesto es capaz de encontrar las proyecciones óptimas en una única iteración, empleando el sistema de ecuaciones lineales, lo que provoca el decremento abrupto en la función de error en todas las simulaciones. Otro detalle importante a resaltar es que en las 100 simulaciones las curvas obtenidas fueron muy similares y convergen en todos los casos a un valor del error cuadrático medio muy similar. Este es un resultado muy interesante, porque

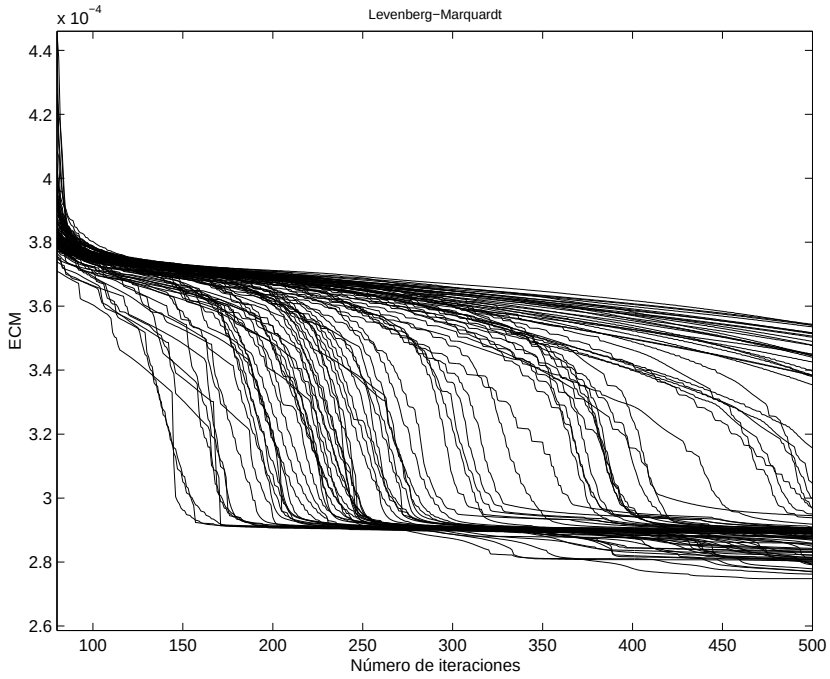


Figura 5.6: Curvas de error durante el entrenamiento de la red empleando el algoritmo de Levenberg-Marquardt para los datos de Dow-Jones.

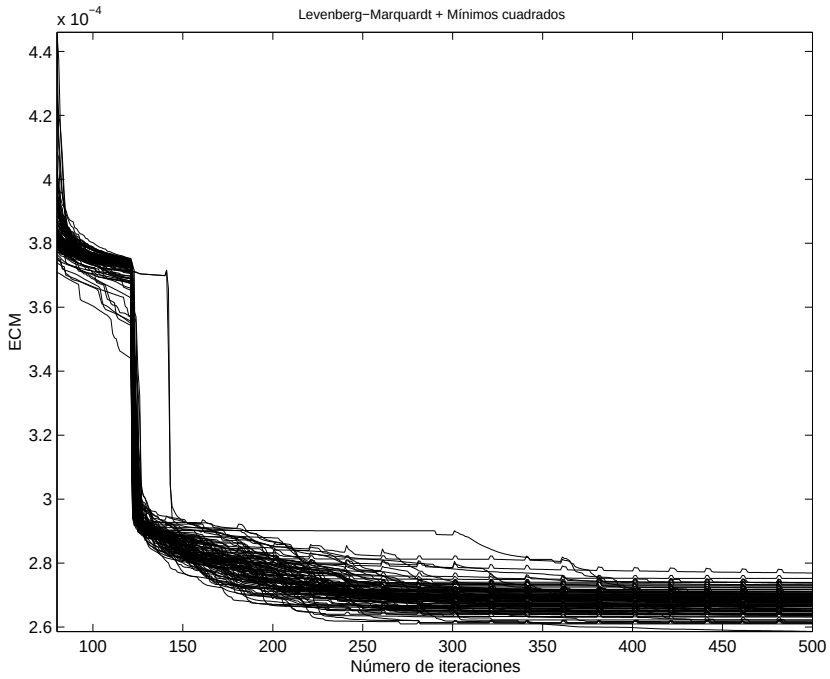


Figura 5.7: Curvas de error durante el entrenamiento de la red empleando el método híbrido para los datos de Dow-Jones.

significa que el proceso de entrenamiento es independiente (al menos más que en los métodos convencionales) de los pesos iniciales empleados en la red. Por el contrario, las curvas de error obtenidas por el método de Levenberg-Marquardt son muy diferentes, lo que indica que el rendimiento del algoritmo es mucho más dependiente de los valores iniciales de los parámetros. En este caso, no convergen en la misma iteración (los valores oscilan entre 150 y 500 iteraciones), ni al mismo valor de la función de error (desde 2.8 hasta 3.5 $\times 10^{-4}$). Por consiguiente, se puede afirmar que el método propuesto es un buen procedimiento para obtener un proceso de aprendizaje más homogéneo, eficaz y rápido que el obtenido por los métodos actuales de aprendizaje para redes de neuronas artificiales.

5.4.2 Ejemplo con los datos de la serie temporal de la competición de Santa Fe

El segundo conjunto de datos empleado en las simulaciones fue la serie temporal del láser de la competición de Santa Fe [133]. Para modelar esta serie se empleó una topología 3-11-1. En este caso se usaron 1000 muestras para el entrenamiento de la red. La figura 5.8 presenta los histogramas obtenidos en las 100 simulaciones para el algoritmo de Levenberg-Marquardt, subfigura (a), y el método híbrido, subfigura (b). El error cuadrático medio obtenido por el primero de los algoritmos en cada una de las 100 simulaciones estuvo en el intervalo $[7.56 \times 10^{-5}, 4.42 \times 10^{-4}]$, mientras que en el segundo método los valores del error estuvieron en el intervalo $[6.27 \times 10^{-5}, 3.79 \times 10^{-4}]$. En este caso, como en el ejemplo anterior, el método híbrido permite evitar algunos mínimos locales. Además, como se puede apreciar cualitativamente en esta figura, el error cuadrático medio es menor que 1.5×10^{-4} en 42 de las 100 simulaciones, en el caso del algoritmo de Levenberg-Marquardt, mientras que para el método híbrido se consigue reducir el error por debajo de ese umbral en 63 de las simulaciones. En cualquier caso, para verificar rigurosamente si la diferencia entre ambas distribuciones es estadísticamente significativa se empleó un método no paramétrico. En concreto, se utilizó la prueba de Kolmogorov-Smirnov. La hipótesis nula empleada fue suponer que ambos métodos tienen la misma distribución para un nivel de confianza del 99% ($\alpha = 0.01$). El resultado de esta prueba ($p = 1.22 \times 10^{-5}$) mostró que se puede rechazar la hipótesis nula, puesto que $\alpha > p$. Consecuentemente, se puede afirmar que la diferencia entre ambas distribuciones es significativa y, por tanto, el método propuesto mejora el rendimiento del algoritmo estándar de Levenberg-Marquardt.

Finalmente, la figura 5.9 muestra algunos ejemplos de las curvas de error obtenidas durante el proceso de entrenamiento en las 100 simulaciones realizadas. La línea continua se corresponde con la curva del algoritmo de Levenberg-Marquardt y la línea discontinua representa la aproximación híbrida. Obsérvese que en esta figura

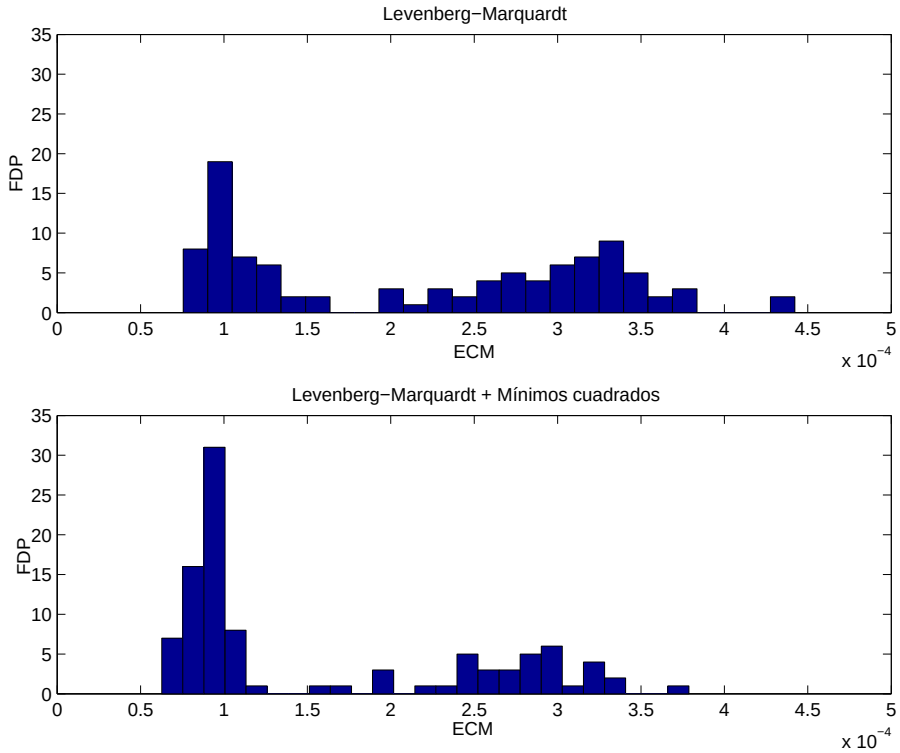


Figura 5.8: Función de densidad de probabilidad (FDP) para los datos del láser empleando (a) el algoritmo de Levenberg-Marquardt y (b) el método híbrido.

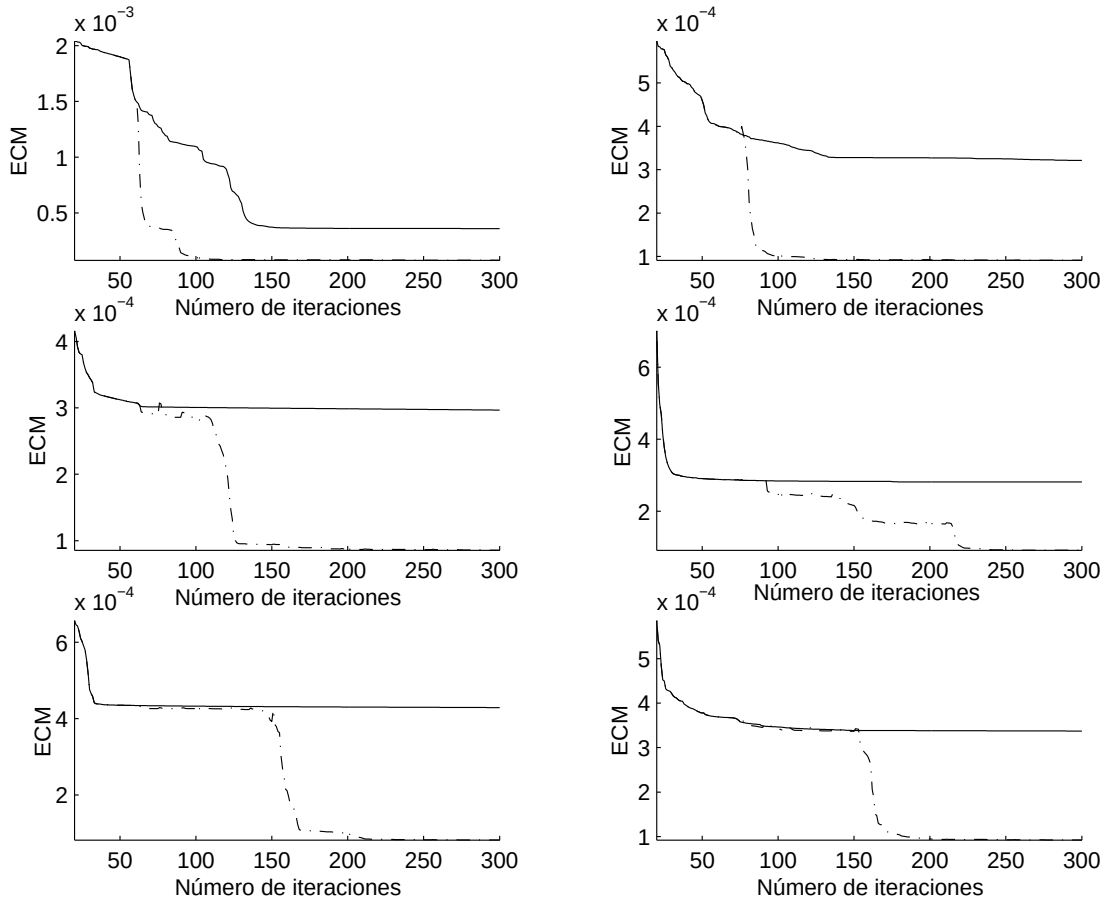


Figura 5.9: Algunos ejemplos de curvas de error obtenidas empleando el algoritmo de Levenberg-Marquardt (línea contínua) y el método híbrido (línea discontinua).

sólo se han mostrado las curvas a partir de la iteración número 30 por las mismas razones que en ejemplo anterior. En la figura 5.9 se puede apreciar que las curvas obtenidas por el método de Levenberg-Marquardt son casi planas durante las últimas iteraciones del entrenamiento, lo que significa que el método está "atrapado" en un mínimo local o en una meseta. En cualquier caso, parece claro que el algoritmo no es capaz de mejorar el error obtenido. Por el contrario, el método propuesto evita las situaciones anteriores y mejora considerablemente el error obtenido por la otra aproximación. Por tanto, podemos afirmar que el método desarrollado permite evitar mínimos locales y acelerar, además, la convergencia de los métodos actuales.

5.4.3 Ejemplo con los datos de la serie de Mackey-Glass

El último conjunto de datos empleado en las simulaciones fue el correspondiente a la serie caótica de Mackey-Glass [93]. Los valores de esta serie temporal se obtienen mediante la siguiente ecuación diferencial:

$$\frac{dx(t)}{dt} = -0.1x(t) + \frac{0.2x(t-\tau)}{1+x^{10}(t-\tau)} \ . \tag{5.1}$$

En este experimento se emplearon los datos disponibles en la base de datos de la Universidad Carnegie Mellon (CMU Learning Benchmark Archive[1]), que fueron generados empleando $\tau = 17$. El conjunto de entrenamiento está formado por 2000 muestras y, en este caso, la topología de la red empleada fue la 3-6-1. En este problema, el objetivo de la red de neuronas era predecir el valor de la serie en el instante $t+1$ a partir de tres valores previos t, $t-1$ y $t-2$. En este ejemplo, el número de entradas no es consistente con el teorema de embebido para series caóticas de Taken [125], que determina el número de muestras previas necesarias para realizar una buena aproximación de la serie, y que en este caso particular es aproximadamente igual a siete. Sin embargo, este hecho fue provocado a propósito para dificultar la tarea de modelado de la red.

Los resultados obtenidos en las 100 simulaciones se muestran en la figura 5.10. Esta figura contiene los histogramas correspondientes al error cuadrático medio obtenido en la última iteración del algoritmo (300) en cada una de las simulaciones. La subfiguras (a) y (b) contienen el histograma correspondiente al algoritmo de Levenberg-Marquardt y al método híbrido, respectivamente. Como se puede observar, el método desarrollado consigue, como en los ejemplos anteriores, un rendimiento mejor que el algoritmo de Levenberg-Marquardt. Las figuras 5.11 y 5.12 contienen las curvas de error obtenidas durante el proceso de aprendizaje para el algoritmo de Levenberg-Marquardt y el método híbrido, respectivamente.

[1] http://legend.gwydion.cs.cmu.edu/neural-bench/benchmark/mackey-glass.html

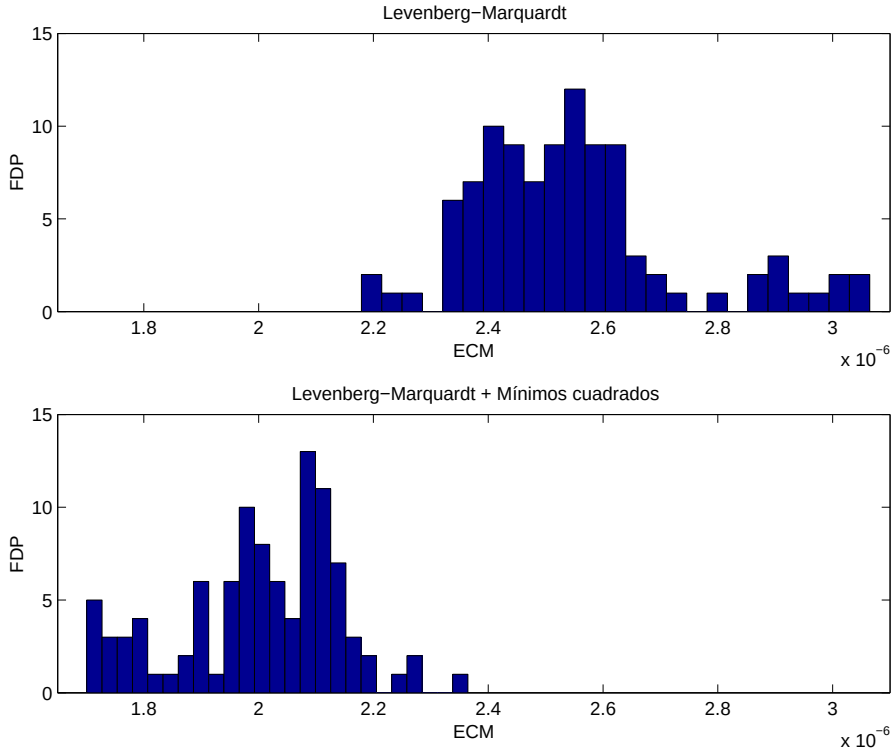


Figura 5.10: Función de densidad de probabilidad (FDP) para los datos de la serie caótica de Mackey-Glass empleando (a) el algoritmo de Levenberg-Marquardt y (b) el método híbrido.

Como se puede observar el método híbrido propuesto alcanza más rápidamente la convergencia hacia la solución y las curvas obtenidas son mucho más homogéneas. Asimismo, en este experimento se estudió el comportamiento del método propuesto cuando la conmutación entre el paso 4 y 5 se realiza en diferentes iteraciones del proceso de entrenamiento. Con este objetivo, se realizaron dos simulaciones diferentes, forzando la conmutación en la trigésima (LS30) y en la decimoquinta (LS15) iteración del algoritmo. Los resultados obtenidos se muestran en la figura 5.13. Las subfiguras (a), (c) y (e) contienen los histogramas construidos a partir del error en la última iteración para el algoritmo de Levenberg-Marquart y los métodos LS30 y LS15, respectivamente. En estas subfiguras se puede apreciar que los métodos LS30 y LS15 mejoran los resultados del Levenberg-Marquardt, ya que en éste último los valores obtenidos están en el intervalo $[2.18 \times 10^{-6}, 3.06 \times 10^{-6}]$ mientras que en los otros dos se alcanzan valores en los intervalos $[1.70 \times 10^{-6}, 2.36 \times 10^{-6}]$ y $[1.73 \times 10^{-6}, 2.35 \times 10^{-6}]$, respectivamente. Además, se puede observar que los histogramas obtenidos por los métodos LS30 y LS15 son muy similares, aunque la probabilidad de obtener un valor de error más pequeño es mayor en el LS30, puesto que existe un mayor número de casos situados en la parte izquierda del histograma. En concreto, el número de casos

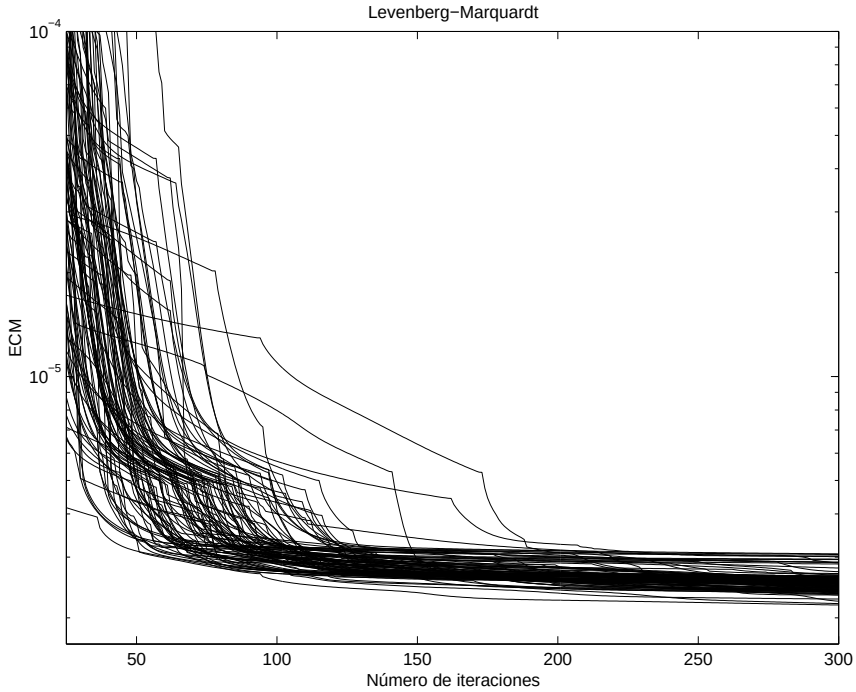


Figura 5.11: Curvas de error durante el entrenamiento de la red empleando el algoritmo de Levenberg-Marquardt para la serie temporal de Mackey-Glass.

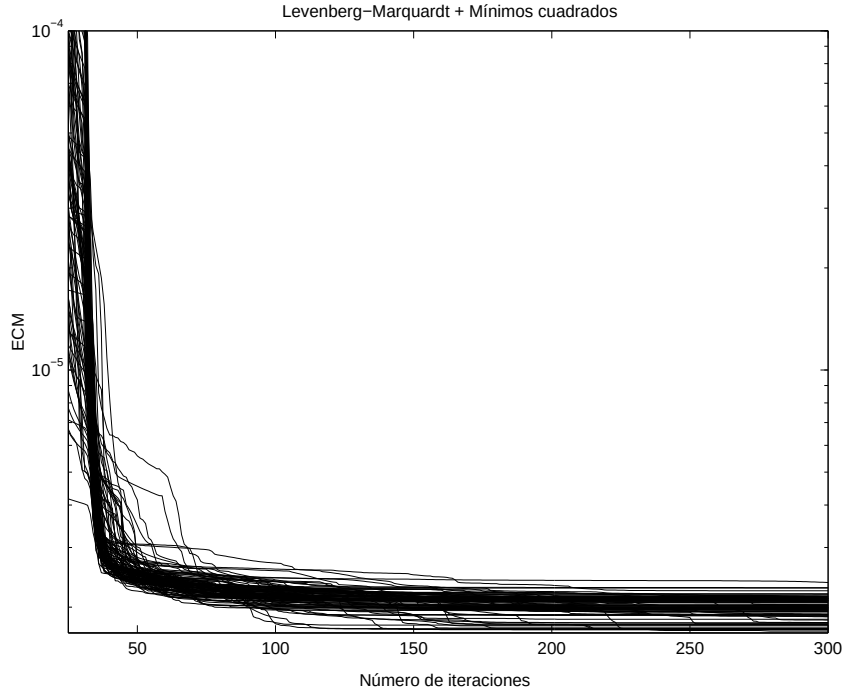


Figura 5.12: Curvas de error durante el entrenamiento de la red empleando el método híbrido para la serie temporal de Mackey-Glass.

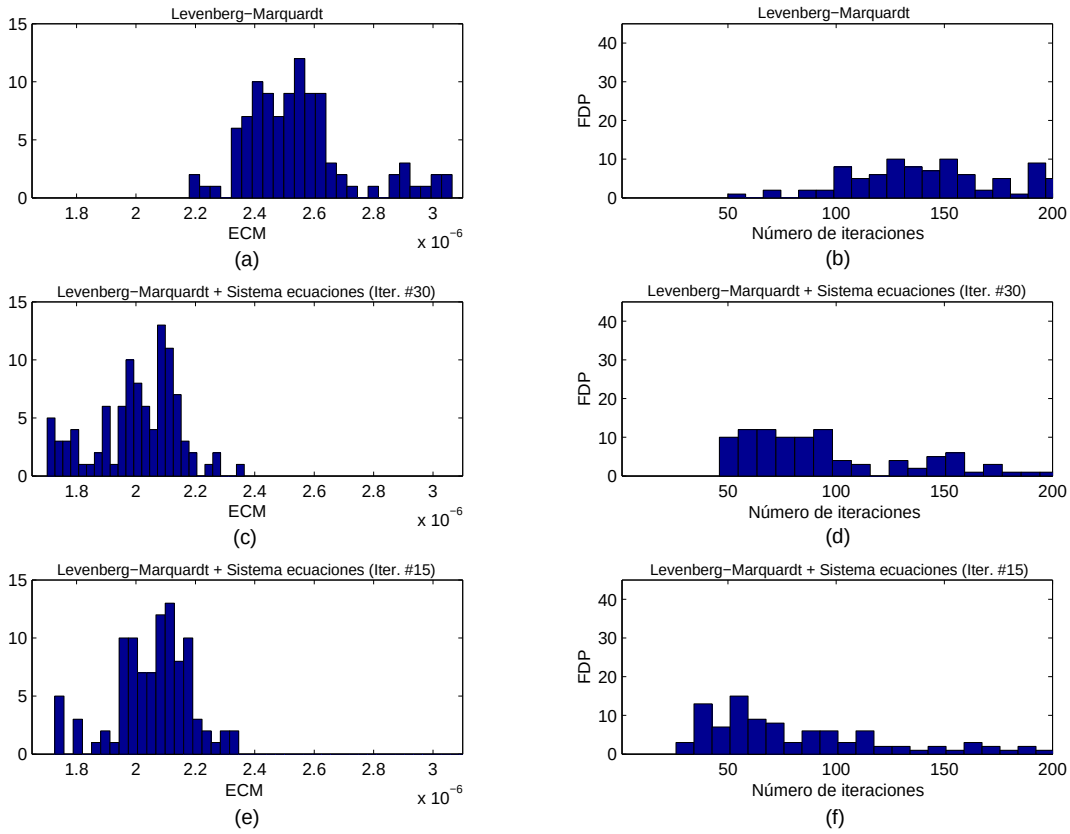


Figura 5.13: Histogramas del error final del entrenamiento (a), (c) y (e) y número de iteraciones para la convergencia (b), (d), (f) para los algoritmos de Levenberg-Marquardt, híbrido con conmutación en la trigésima iteración e híbrido con conmutación en la decimoquinta iteración, respectivamente.

menores que 1.85×10^{-6} es de 17 y 8 para los métodos LS30 y LS15, respectivamente. Por otro lado, las subfiguras (b), (d) y (f) contienen los histogramas correspondientes al tiempo de convergencia para cada uno de los métodos anteriores. En este experimento, se consideró el tiempo de convergencia como la iteración en la cual el algoritmo alcanza un valor en el intervalo $[\epsilon_{min}, \epsilon_{min} + 0.5\epsilon_{min}]$ donde ϵ_{min} es el error mínimo obtenido durante el proceso de aprendizaje. Finalmente, la tabla 5.1 muestra el valor medio y la desviación típica del tiempo de convergencia obtenido en las 100 simulaciones en cada uno de los algoritmos.

Tabla 5.1: Media y desviación típica del tiempo de convergencia en las simulaciones de Monte Carlo.

Algoritmo	Media	Desviación típica
Levenberg-Marquardt	78.98	24.95
LM+LS (Conmutación en iteración #30)	41.72	8.02
LM+LS (Conmutación en iteración #15)	31.02	13.44

En esta tabla el método propuesto fue identificado con el acrónimo LM+LS. Como se puede observar en esta tabla el tiempo de convergencia medio y la desviación típica en el método de Levenberg-Marquardt es mayor que en los otros dos casos. Además, aunque el método LS30 es mejor que el LS15, en términos del error final obtenido, ésta última tiene la ventaja de alcanzar el error final más rápidamente. Esto es debido a que la conmutación entre el algoritmo de optimización estándar y el híbrido se realiza en una iteración más temprana del entrenamiento. Por tanto, y como conclusión de este hecho, se puede forzar la conmutación en una de las primeras iteraciones del método para alcanzar una mayor velocidad de convergencia. Sin embargo, el precio a pagar es que el error obtenido al final del entrenamiento es probablemente mayor que si la conmutación se realiza más adelante. En cualquier caso, esto puede ser de utilidad en diversas aplicaciones donde se requiere una solución suficientemente buena, tan rápido como sea posible, como por ejemplo en un sistema de monitorización o control en tiempo real, aunque no sea la óptima.

5.5 Discusión

En este capítulo se ha presentado un nuevo algoritmo de aprendizaje que combina la potencia de los métodos actuales (e.g., Levenberg-Marquardt, gradiente conjugado, quasi-Newton, etc.) para el aprendizaje de las funciones básicas de la red (pesos de la primera capa), y el método para redes de una capa, basado en un sistema de

ecuaciones lineales, propuesto en el capítulo 3 para obtener óptimamente las proyecciones correspondientes (pesos de la segunda capa). Los resultados obtenidos en la sección anterior demostraron que el método propuesto en este capítulo es un algoritmo eficiente para el aprendizaje de redes de neuronas multicapa con alimentación hacia delante y que mejora el rendimiento del algoritmo de Levenberg-Marquardt estándar. Las principales ventajas del método propuesto son las siguientes:

- *Permite evitar mínimos locales.* En la sección 5.3 se han discutido las propiedades del método que permiten evitar mínimos locales. En ella se ha mostrado que al obtener los pesos de la segunda capa de la red mediante el sistema de ecuaciones lineales es posible desplazarse a otra región de la superficie de error que esté fuera del área de atracción de un mínimo local. Este hecho permite evitar, al menos, los mínimos locales asociados a la segunda capa de la red. Estas suposiciones teóricas fueron confirmadas en las simulaciones realizadas, donde se corroboró que el método propuesto, partiendo de las mismas condiciones iniciales, es capaz de evitar algunos mínimos locales en los cuales quedó atrapado el algoritmo de Levenberg-Marquardt. Consecuentemente, es mucho más probable obtener un error menor al llevar cabo el aprendizaje de la red de neuronas.

- *Acelera la convergencia de los métodos actuales.* Este hecho se produce a causa de dos motivos. El primero de ellos es que al no estar basado en el gradiente no se ve afectado por la aparición de mesetas y, por tanto, evita los problemas presentados por este tipo de métodos. En segundo lugar, los pesos óptimos para la segunda capa se obtienen en una única iteración y, por tanto, reducen drásticamente el número de pasos requeridos por los métodos convencionales. Estos hechos fueron confirmados en todos los experimentos realizados, en los cuales, el método híbrido consigue siempre la solución en un menor número de iteraciones. Además, es importante mencionar que el algoritmo desarrollado es un procedimiento rápido de aprendizaje porque está basado en un método analítico que permite obtener la solución óptima empleando un sistema de ecuaciones lineales, el cual no requiere un coste computacional elevado.

- *Entrenamiento más homogéneo.* Se ha comprobado en todas las simulaciones realizadas que el método desarrollado proporciona un proceso de aprendizaje más uniforme, puesto que los resultados obtenidos y la forma de alcanzarlos, no son tan dependientes del estado inicial de la red, es decir, de los pesos iniciales empleados. Esto es una consecuencia de los "saltos" controlados del método, que mejoran el error significativamente en un número muy pequeño de iteraciones.

- *No requiere parámetros adicionales.* La mayoría de los algoritmos de entrenamientos propuestos por otros autores introducen parámetros adicionales en el entrenamiento (e.g., el término momentum, pasos de aprendizaje adicionales), que son dependientes del problema, ya que deberán ser ajustados para cada caso en concreto. Sin embargo, el algoritmo desarrollado no utiliza ningún parámetro de entrenamiento adicional. Esto supone una ventaja importante pues hace que el proceso de aprendizaje sea más independiente del conjunto de datos empleado, ya que no requiere el ajuste de estos parámetros.

Es importante resaltar también que aunque en todas las simulaciones realizadas en este capítulo se ha empleado en algoritmo de Levenberg-Marquardt para la actualización de los pesos de la primera capa de la red, se puede emplear alternativamente cualquiera de los algoritmos actuales, e.g., gradiente conjugado, métodos quasi-Newton, retropropagación del error, etc.

Otro aspecto a destacar del método propuesto es que representa una aproximación híbrida que combina las propiedades ventajosas de los métodos de optimización local y global. Esto es una consecuencia del uso del método basado en un sistema de ecuaciones lineales para obtener de forma óptima los pesos de la segunda capa (proyecciones) a partir de los pesos actuales de la primera capa (bases). Este mecanismo proporciona un sistema de "saltos" controlados que permiten desplazarse a otro lugar de la superficie de error, relativamente lejano, que emula la búsqueda estocástica de los métodos de optimización global.

Finalmente, es importante comentar algunos aspectos sobre la conmutación entre el algoritmo de optimización estándar y el método híbrido (pasos 4 y 5 del algoritmo propuesto). En este sentido se ha considerado el uso del método híbrido, i.e., regla de optimización estándar para los pesos de la primera capa y sistema de ecuaciones lineales para los pesos de la segunda, desde una iteración temprana del algoritmo, incluso desde la primera. Esta aproximación acelera enormemente la convergencia del método, sin embargo, como ha sido corroborado en los estudios experimentales, aumenta la probabilidad de quedarse atrapado en un mínimo local, comparado con una conmutación más tardía. Como se ha mencionado previamente, esto proviene del hecho de que las bases no están todavía optimizadas, ya que se inicializan con valores aleatorios, y por tanto no es muy útil obtener las proyecciones óptimas para estas bases. Por consiguiente, es más adecuado, en general, utilizar durante las primeras iteraciones cualquiera de los métodos estándar, para luego cambiar al método híbrido. A pesar de ello, el método híbrido puede emplearse desde la primera iteración en aquellas aplicaciones que no requieran el óptimo global, pero sí una rápida y eficaz respuesta del sistema, por ejemplo, en un sistema de control en tiempo real.

Capítulo 6

Modelado local mediante redes de neuronas

"*Divide y vencerás*"

Filipo de Macedonia

6.1 Introducción

El problema del aprendizaje a partir de ejemplos puede considerarse equivalente al problema de la aproximación de funciones multivariable [108], i.e., encontrar una transformación $f : \mathbb{R}^N \longrightarrow \mathbb{R}^M$ a partir de un conjunto de datos. Estos métodos para la aproximación de funciones se pueden dividir en métodos globales y métodos locales. En los primeros solamente se emplea un modelo para caracterizar toda la función. Este tipo de modelos globales proporciona buenos resultados con datos estacionarios pero, sin embargo, cuando éstos son no estacionarios la identificación de un modelo global resulta una tarea mucho más complicada. Durante los últimos años, el modelado local de funciones ha experimentado un gran interés por parte de la comunidad científica, ya que estos métodos pueden, a menudo, superar algunos de los problemas de los métodos globales [121]. Los modelos locales están basados en la división de los datos en pequeños subconjuntos que son modelados independientemente, en la mayoría de los casos usando métodos lineales [121]. En esta aproximación, la división de los datos se realiza, habitualmente, mediante el uso de algún algoritmo de agrupamiento, como por ejemplo, el algoritmo *k-means* [99] o los mapas autoorganizativos [127, 130]. Sin embargo, los métodos locales presentan una serie de problemas, como una lenta convergencia y requerimientos de memoria muy grandes.

En los dos capítulos anteriores de esta memoria se han desarrollado nuevos algoritmos para el aprendizaje de redes de neuronas multicapa con alimentación hacia

delante. Los métodos propuestos están basados en un modelado global de los datos de entrada. El objetivo de este capítulo es el desarrollo de un nuevo método para el modelado local de funciones. En concreto, se propone un nuevo sistema neuronal compuesto de un mapa autoorganizativo de Kohonen y un conjunto de redes de neuronas de una capa. Como se mostrará posteriormente, el uso de las redes de una capa, entrenadas con el sistema de ecuaciones presentado en el capítulo 3, permite la generación de los modelos locales de forma muy eficaz y rápida.

6.2 Modelado dinámico no lineal

En el análisis de los datos generados por sistemas físicos, se puede asumir en muchos casos que éstos proceden de un sistema dinámico expresado mediante la siguiente ecuación diferencial:

$$\frac{d\mathbf{x}(t)}{dt} = \mathbf{g}(\mathbf{x}(t)) \tag{6.1}$$

donde $\mathbf{x}(t)$ representa el estado del sistema. Si $\mathbf{g}$ es una función lineal de los estados del sistema entonces el sistema generado es un sistema lineal, en caso contrario, el sistema será no lineal. Independientemente de la forma de la función $\mathbf{g}$, un sistema dinámico es aquel cuyo estado varía en el tiempo. Este tipo de sistemas describen gran cantidad de fenómenos reales, como por ejemplo, sistemas biológicos, flujos de fluidos, circuitos eléctricos y movimientos astronómicos. Sin embargo, en un entorno real no se conocen las ecuaciones que definen el sistema dinámico y sólo se tiene un conjunto de observaciones del sistema. Packard et al. [103] mostraron experimentalmente, y Takens [125] demostró matemáticamente, que dado un conjunto de muestras de un sistema dinámico,

$$\mathbf{x}(n) = [x(n), x(n-\tau), \ldots, x(n-(N-1)\tau)]$$

donde τ es un retardo temporal constante, se puede crear una trayectoria en un espacio euclídeo de dimensión N que preserva las invariantes dinámicas (dimensión de correlación y exponentes de Lyapunov) del sistema original. La dimensión N del espacio debe ser mayor que $2D$, donde D es la dimensión del sistema dinámico original. Este resultado es muy interesante, pues demuestra que la información del estado de un sistema puede recuperarse empleando un número suficiente de muestras de los datos de entrada. En términos matemáticos, este resultado significa que existe una transformación Φ invertible desde un espacio M de dimensión K del sistema original a un espacio euclídeo $\mathbb{R}^N$ de reconstrucción. Dicha transformación se denomina de embebido, y el teorema se conoce como teorema de embebido de Takens [125]. De

acuerdo con este teorema, cuando $N > 2D$ existe una transformación $\mathbf{f} : \mathbb{R}^N \longrightarrow \mathbb{R}^N$ tal que

$$\mathbf{x}(n+1) = \mathbf{f}(\mathbf{x}(n)). \tag{6.2}$$

La ecuación (6.2) representa un sistema de múltiple entrada y salida. En el caso concreto de un sistema de una salida, se obtiene una transformación $f : \mathbb{R}^N \longrightarrow \mathbb{R}$ tal que

$$x(n+1) = f(\mathbf{x}(n)). \tag{6.3}$$

La ecuación (6.3) define un sistema determinístico autoregresivo no lineal de los datos de entrada del sistema. La existencia de este modelo predictivo es la base teórica del modelado dinámico, de forma que permite construir, a partir de un vector de entrada, un modelo para aproximar la transformación f.

El modelado dinámico se basa en un proceso de dos etapas. La primera consiste en la transformación de los datos del sistema observado en una trayectoria en un espacio de reconstrucción utilizando una técnica de embebido [133]. El método más empleado para realizar esto consiste en una línea de retardos temporales con un tamaño determinado por el teorema de Takens. La dimensión D del sistema dinámico puede estimarse mediante diversos métodos, como el algoritmo de dimensión de correlación [54]. Después de esta etapa se obtiene un conjunto de vectores de dimensión N que crean una trayectoria en un espacio euclídeo. La figura 6.1 muestra un ejemplo de este hecho en un espacio de tres dimensiones. Las líneas discontinuas representan los vectores $\mathbf{x}(n+i), i = 0, \ldots, 5$, obtenidos en la etapa de embebido, y la línea continua representa la trayectoria formada por estos vectores. En la segunda etapa se construye el modelo predictivo de (6.3) a partir de la trayectoria, en el espacio de reconstrucción, obtenida en la etapa previa. El objetivo del modelado dinámico no lineal es aproximar la transformación $f(\mathbf{x})$ en (6.3) que genera la trayectoria en el espacio de reconstrucción $\mathbf{x}(n) \rightarrow \mathbf{x}(n+1)$. A diferencia del caso lineal, no existe ningún algoritmo que determine perfectamente la función f.

Actualmente, existen una gran cantidad de técnicas para el modelado dinámico no lineal, que pueden clasificarse en dos categorías: modelos globales y modelos locales. En el primer caso, el objetivo es aproximar la función $f(\mathbf{x})$ de forma global mediante otra función $\hat{f}(\mathbf{x})$ compuesta de funciones básicas,

$$\hat{f}(\mathbf{x}) = \sum_{i=1}^{n} \alpha_i \phi_i(\mathbf{x}) \tag{6.4}$$

donde α_i son los coeficientes y $\phi_i(\mathbf{x}); i = 1, 2, \cdots, n$ es un conjunto de funciones básicas. Empleando este método se genera una única función $\hat{f}(\mathbf{x})$ que aproxima todos los puntos en el espacio de reconstrucción. Una alternativa a esta predicción es el modelado local, en el cual la función estimada es la concatenación de diversos

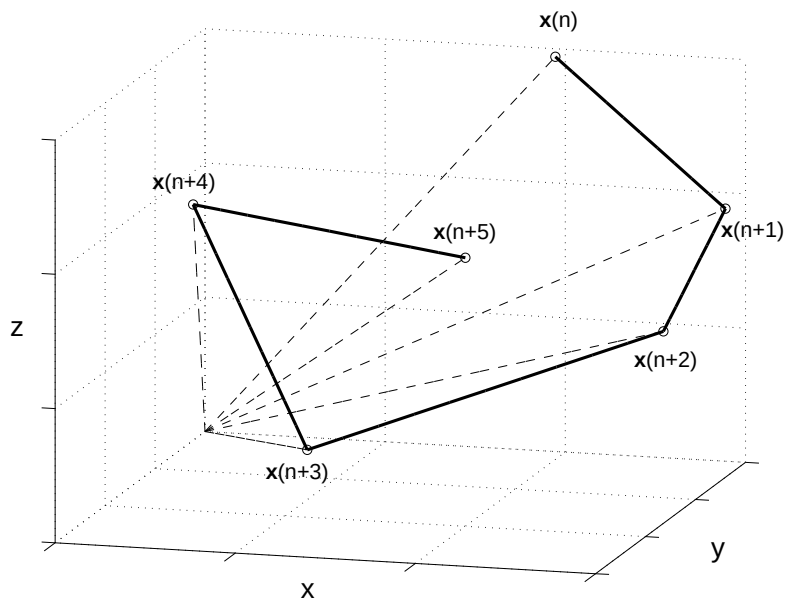


Figura 6.1: Trayectoria formada por la línea de retardos temporales.

estimadores locales [35], i.e,

$$\hat{f}(\mathbf{x}) = \bigcup_{i=r}^{m} \hat{f}_r(\mathbf{x}) \tag{6.5}$$

donde el símbolo de la unión significa, en este caso, que la función $f(\mathbf{x})$ está definida en diferentes espacios. Con esta aproximación la transformación $\hat{f}(\mathbf{x})$ está descompuesta en una familia de funciones $\hat{f}_r(\mathbf{x})$, $r = 1, \ldots, R$, donde cada una de ellas ajusta solamente un conjunto de puntos en el espacio de reconstrucción. Lawrence et al. [87] demostraron experimentalmente que el modelado local es muy adecuado para la aproximación de funciones. Además, aquellas funciones que son complejas de modelar de forma global pueden estimarse más fácilmente con modelos locales.

En cuanto a los modelos globales, se han desarrollado gran cantidad de métodos para seleccionar el número y tipo de las funciones y obtener los coeficientes. En particular, las redes de neuronas se han empleado ampliamente para solucionar este problema [58][111]. La gran mayoría de las aproximaciones neuronales desarrolladas se basan en el modelo global de los datos presentado en la ecuación (6.4).

6.3 Mapas autoorganizativos de Kohonen

En 1982 Kohonen propuso un nuevo paradigma neuronal denominado mapas autoorganizativos (*Self-organizing maps*, SOM) [75, 76, 77]. Este tipo de red consiste, normalmente, en un conjunto de elementos de procesamiento (neuronas) agrupados

en una malla de dos dimensiones. Los mapas autoorganizativos son estructuras neuronales basadas en un tipo de aprendizaje no supervisado denominado aprendizaje competitivo. En el trascurso de este aprendizaje, las neuronas de la red compiten entre ellas para ser la "ganadora", la cual proporcionará la salida de la red. Durante el aprendizaje, el SOM adapta sus pesos de forma que el espacio N-dimensional de entrada se proyecta en un mapa M-dimensional, donde $M < N$, preservando las relaciones de proximidad de las entradas en el mapa. La capa de entrada del SOM está formada for N unidades, una por cada uno de los componentes del vector de entrada $\mathbf{x}_s = [x_{1s}, \ldots, x_{Ns}]^T,\ s = 1, \ldots, S$. La capa de salida está formada por una malla bidimensional de $P \times Q$ neuronas, cada una de las cuales está conectada a todas las entradas mediante un vector de pesos $\mathbf{w}_i = [w_{1i}, \ldots, w_{Ni}]^T,\ i = 1, \ldots, P \times Q$. La principal diferencia entre el SOM y otras redes de aprendizaje competitivo es la forma de actualización de los pesos, que en este caso no se produce sólo en la neurona ganadora, sino también en todas las neuronas del mapa.

La respuesta del SOM para una entrada $\mathbf{x}_s$ está determinada por la neurona ganadora de la red, r, que contiene el vector de pesos más semejante a la entrada actual. La neurona ganadora se determina, usualmente, mediante la siguiente regla:

$$r = \arg\min_i \|\mathbf{w}_i - \mathbf{x}_s\| \tag{6.6}$$

donde $\|\cdot\|$ representa la distancia euclídea de un vector. En el aprendizaje de la red, todos los pesos se actualizan empleando la siguiente regla de aprendizaje:

$$\Delta\mathbf{w}_i(n+1) = \eta(n)h_{ir}(n)(\mathbf{x}_s - \mathbf{w}_i) \tag{6.7}$$

donde $h_{ir}(n)$ es una función de vecindad espacial y $\eta(n)$ es el paso de aprendizaje que varía durante el aprendizaje mediante la siguiente ecuación:

$$\eta(n) = \frac{1}{a + bn}\ , \tag{6.8}$$

donde a y b son constantes. La función de vecindad empleada habitualmente es:

$$h_{ir}(n) = \exp\left(\frac{\|\mathbf{c}_i - \mathbf{c}_r\|^2}{2\sigma^2(n)}\right), \tag{6.9}$$

donde $\mathbf{c}_i \in \mathbb{R}^2$ y $\mathbf{c}_r \in \mathbb{R}^2$ son las coordenadas correspondientes a las neuronas i y r, respectivamente, y $\sigma(n)$ determina el conjunto de neuronas vecinas de la ganadora que serán incluidas en el proceso de aprendizaje. El valor de σ variará durante el entrenamiento de la siguiente forma:

$$\sigma(n) = \frac{1}{c + dn}\ , \tag{6.10}$$

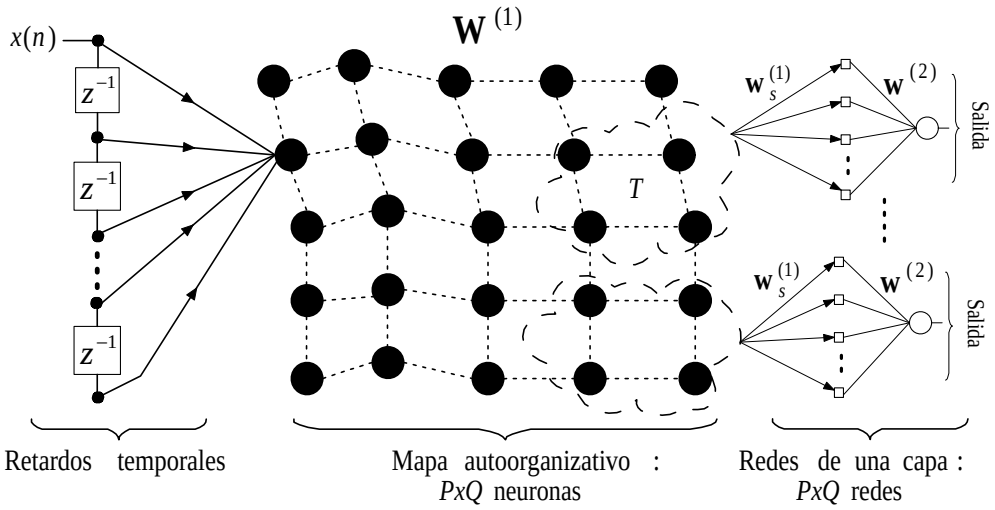


Figura 6.2: Estructura del sistema neuronal durante el aprendizaje.

donde c y d son constantes.

Al final del proceso de aprendizaje, el mapa autoorganizativo realiza una transformación, $\Psi : X \longrightarrow Y$, desde un espacio de entrada X a un espacio de salida Y (malla de neuronas de la red), que presenta, entre otras, las siguientes propiedades:

- La transformación Ψ lleva a cabo una discretización y reducción del espacio de entrada [81], i.e., convierte las relaciones estadísticas no lineales y complejas de los datos de entrada de alta dimensionalidad en relaciones geométricas sencillas en un espacio de pequeña dimensionalidad.

- La malla obtenida por el SOM forma un espacio de salida ordenado topológicamente, es decir, las regiones que se encuentran adyacentes en el espacio de entrada estarán asociadas a neuronas que son vecinas espacialmente [116]. Esta característica reduce los errores producidos por ruido en las entradas, porque el sistema activará neuronas vecinas, y por tanto semejantes, en el espacio de salida.

6.4 Método de modelado local

El método propuesto en este capítulo pertenece a la aproximación de funciones basada en modelos locales. Para ello, el sistema está compuesto de tres partes diferenciadas, mostradas esquemáticamente en la figura 6.2:

1. *Una capa de embebido* implementada mediante una línea de retardos temporales. Esta fase transforma el espacio original de los datos de entrada, $x(n)$, en un

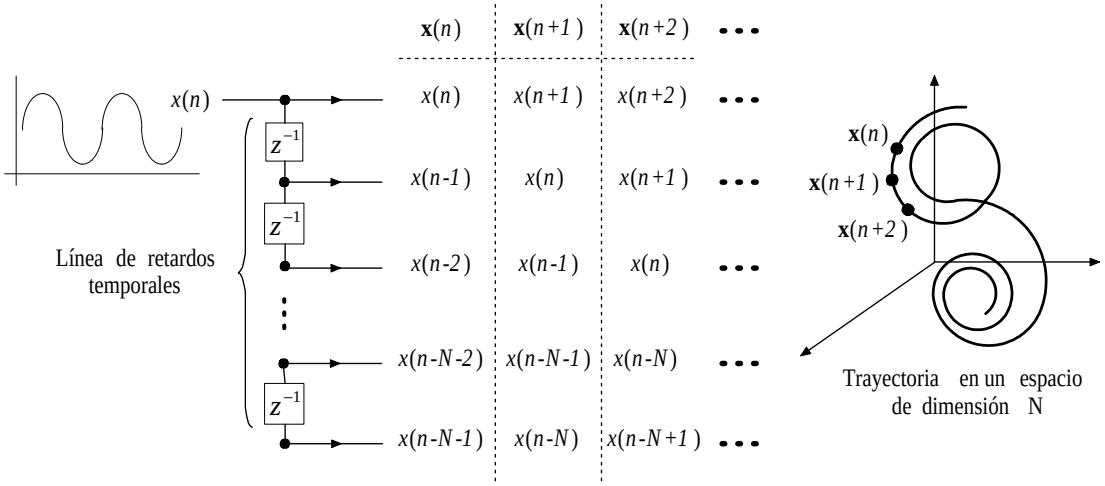


Figura 6.3: Transformación de los datos de entrada en una trayectoria de dimensión N mediante una línea de retardos temporales.

espacio de reconstrucción (ver figura 6.3). La salida de esta etapa es un conjunto de vectores de dimensión N, $\mathbf{x}(n) = [x(n), x(n-\tau), \ldots, x(n-(N-1)\tau)]^T, n = 1, \ldots, S$, creados a partir de los datos de entrada, que forman una trayectoria en un espacio euclídeo.

2. *Un mapa autoorganizativo* entrenado empleando el aprendizaje de Kohonen [77]. La entrada de esta red es el par $(d(n), \mathbf{x}(n))$, donde $d(n)$ es la salida deseada. En el ámbito de la predicción de serie temporales $d(n) = x(n+\gamma\tau)$, donde γ es un determinado punto futuro. El SOM, formado por $P \times Q$ neuronas, representará la dinámica del sistema en la malla de salida pero enriquecida con relaciones de vecindad, i.e., los estados que estén adyacentes en el espacio de reconstrucción estarán representados por neuronas vecinas en el espacio de salida de la red. Finalmente, es importante mencionar que aunque la salida deseada, $d(n)$, se ha utilizado como una entrada para el entrenamiento del SOM, una vez finalizado este proceso la red no utilizará posteriormente esta información cuando opere en un entorno real.

3. *Un conjunto de redes de neuronas de una capa.* El objetivo de este subsistema es construir los modelos locales a partir de los pesos de la red autoorganizativa, $\mathbf{w}_k^{(1)} = [w_{0k}^{(1)}, w_{1k}^{(1)}, \ldots, w_{Nk}^{(1)}]^T; k = 1, \ldots, P \times Q$. El peso $w_{0k}^{(1)}$ está asociado a la entrada $d(n)$ y los otros N con el vector $\mathbf{x}(n)$. Para el entrenamiento de las $P \times Q$ redes de neuronas se empleó el algoritmo desarrollado en el tercer capítulo de esta memoria. Por tanto, los pesos y umbral óptimos para cada una de las redes de una capa se obtiene empleando el siguiente sistema de ecuaciones

lineales con $N+1$ ecuaciones e incógnitas:

$$\begin{aligned}
&\sum_{i=1}^{N}\left(\sum_{s\in T} w_{is}^{(1)} w_{ps}^{(1)} f'(w_{0s}^{(1)})\right) w_i^{(2)} &&= \sum_{s\in T}\left(f^{-1}(w_{0s}^{(1)}) - b^{(2)}\right) f'(w_{0s}^{(1)}) w_{ps}^{(1)};\\
&p = 1, 2, \ldots, N,\\
&\sum_{i=1}^{N}\left(\sum_{s\in T} w_{is}^{(1)} f'(w_{0s}^{(1)})\right) w_i^{(2)} &&= \sum_{s\in T}\left(f^{-1}(w_{0s}^{(1)}) - b^{(2)}\right) f'(w_{0s}^{(1)}),
\end{aligned} \tag{6.11}$$

donde $w_i^{(2)}, i = 1, \ldots, N$ y $b^{(2)}$ son, respectivamente, los pesos y el umbral de la red de una capa con una única salida y T es el conjunto formado por la neurona ganadora del SOM y N_L vecinas.

El aprendizaje del sistema se realiza en dos fases. En la primera, se realiza el aprendizaje del SOM empleando todos los datos del conjunto de entrenamiento. Una vez finalizada esta fase los pesos de esta red no se modificarán posteriormente. La siguiente fase consiste en el aprendizaje del conjunto de $P \times Q$ redes de neuronas de una capa. Para ello, el SOM analiza de nuevo todos los datos de entrada para determinar cual es la neurona ganadora para cada vector de entrada. Los pesos de esta neurona y los de las N_L vecinas se emplearán para el aprendizaje de la red de una capa asociada a la neurona ganadora. El mejor valor para N_L deberá determinarse experimentalmente. El objetivo de esta fase es obtener un conjunto de modelos locales para aproximar la dinámica global de los datos.

Una vez finalizado el entrenamiento de la red, el sistema neuronal operará de la siguiente forma (mostrado esquemáticamente en la figura 6.4):

- Dado un vector de entrada $\mathbf{x}(n)$ (proporcionado por la capa de embebido) se obtiene la neurona ganadora de la SOM (usando la ecuación 6.6). Recalcar de nuevo que en este proceso no se usa la salida deseada $d(n)$ y que sólo se emplea el vector $\mathbf{x}(n)$ para determinar la neurona ganadora.

- A continuación, se activará la red de una capa asociada a la neurona ganadora. En este sentido el SOM actúa como un dispositivo de selección (ver figura 6.4). La red seleccionada proporcionará la salida del sistema empleando como entrada el vector $\mathbf{x}(n)$.

Previamente, se han desarrollado algunas aproximaciones basadas en mapas auto-organizativos para el modelado local de funciones [100, 129]. Sin embargo, el trabajo presentado en este capítulo presenta varias diferencias, la principal de las cuales es que el ajuste de los modelos locales se realiza usando los pesos del SOM. Esta metodología fue también empleada por Principe et al. [112], sin embargo, en este caso los autores emplearon una aproximación basada en un modelo lineal de los datos. Koskela et al.

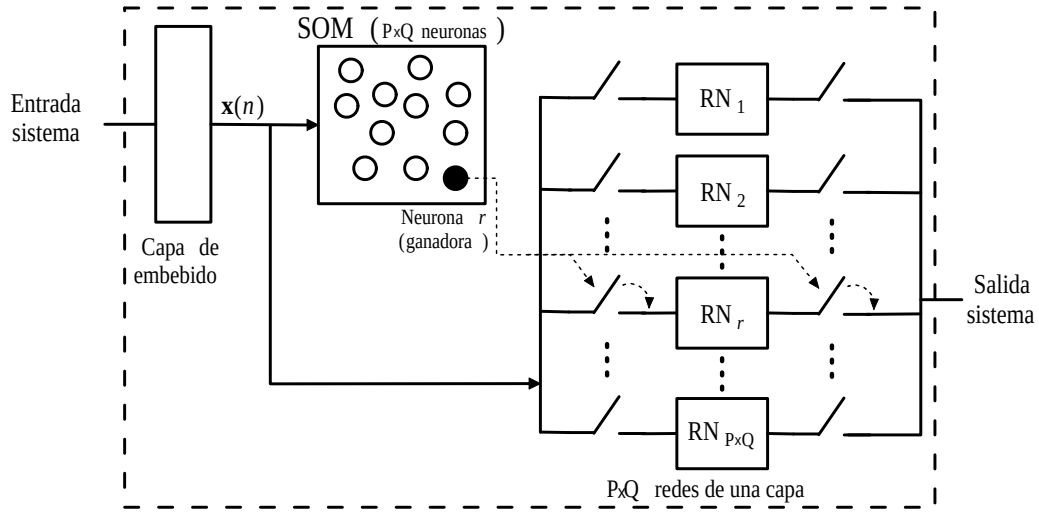


Figura 6.4: Estructura y funcionamiento del sistema neuronal.

[79, 80] presentaron también una aproximación basada en mapas autoorganizativos recurrentes y modelos locales lineales.

El uso del SOM para estimar los diferentes modelos locales necesarios para aproximar la función original presenta una serie de ventajas importantes:

- Esta red mitiga la discontinuidad en el modelado local. Crutchfield y McNamara [35] mostraron experimentalmente que el modelado dinámico no funciona correctamente cuando no existe una continuidad entre los diferentes modelos locales. Debido a las características de las SOM, enunciadas anteriormente, este tipo de red asegura la semejanza entre neuronas vecinas y, por tanto, de los vectores de pesos que las definen. Esto, unido a que para ajustar los modelos locales, mediante las redes de neuronas de una capa, se emplean además de la neurona ganadora un conjunto de neuronas vecinas, asegura una continuidad entre los distintos modelos locales.

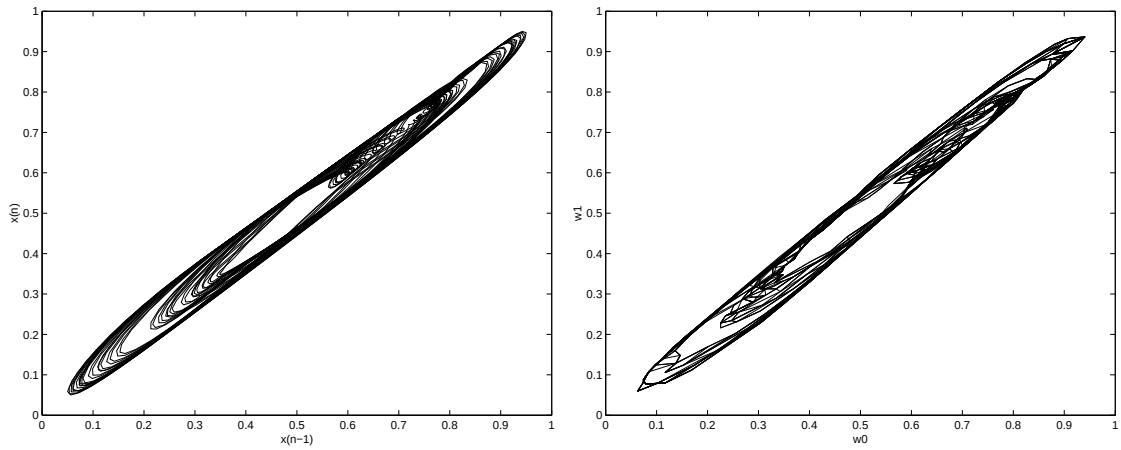
- La representación del espacio de entrada realizada por el mapa autoorganizativo simplifica enormemente la construcción de los modelos locales. Uno de los principales inconvenientes de los métodos de modelado local es la necesidad de construir las relaciones de vecindad a partir de los datos de entrada, ya que esta operación es computacionalmente muy costosa. Sin embargo, con el uso del SOM este proceso se simplifica en gran manera, ya que no es necesario volver hacia atrás en los datos de entrada para estimar los agrupamientos más idóneos.

6.5 Resultados experimentales

El sistema neuronal propuesto fue validado usando tres conjuntos de datos: la serie caótica de Mackey-Glass, los datos del láser de la competición de Santa Fe y la serie caótica de Lorenz. En todos los casos, el objetivo del sistema era predecir el valor de la serie en el instante $x(n+1)$ ($\tau = 1; \gamma = 1$) dado un conjunto de muestras previas. Los valores de las constantes a, b, c y d (ecuaciones (6.8) y (6.10)), empleados en el entrenamiento del SOM, fueron determinados experimentalmente y en todas las simulaciones se utilizaron los siguientes: $a = 1$, $b = 1/25000$, $c = 1/8000$ y $d = 1/800$.

6.5.1 Ejemplo con los datos de la serie caótica de Mackey-Glass

El primer conjunto de datos empleado en las simulaciones fue la serie caótica de Mackey-Glass. En este caso se empleó un SOM formado por 20×20 neuronas. La dimensión de embebido empleada en este caso fue 7 y el número de vecinos empleados para el entrenamiento de las redes de una capa fue de 41. El conjunto de entrenamiento y prueba de la red contenían 2000 y 1000 muestras, respectivamente. La figura 6.5(a) muestra la trayectoria bidimensional formada por el conjunto de entrenamiento, mientras que la figura 6.5(b) presenta la trayectoria obtenida por las neuronas ganadoras en el SOM.



(a) (b)

Figura 6.5: (a) Trayectoria bidimensional de la serie caótica de Mackey-Glass en el espacio de reconstrucción y (b) trayectoria formada por las neuronas ganadoras del SOM.

En estas figuras se puede observar que ambas trayectorias son muy parecidas y, por consiguiente, la red es capaz de simular la dinámica del sistema no lineal. La figura 6.6 contiene la serie empleada en el conjunto de prueba (línea continua) y la salida del sistema neuronal propuesto (línea discontinua).

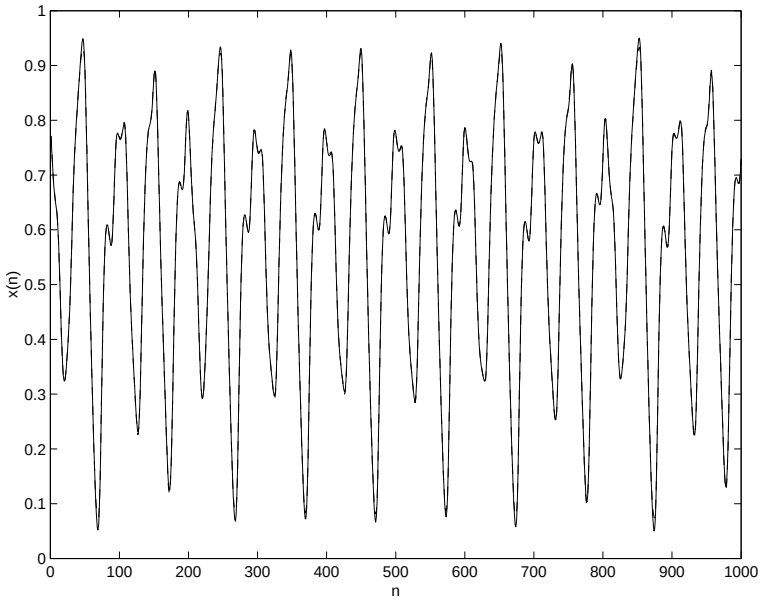


Figura 6.6: Serie temporal de Lorenz (línea continua) y salida de la red de neuronas propuesta (línea discontinua).

Como se puede observar la diferencia entre ambas es inapreciable visualmente, lo que implica que el sistema realiza una excelente aproximación de la serie. La figura 6.7 corrobora este hecho. Esta figura muestra la salida real de la red frente a la salida deseada para el conjunto de prueba. Como se puede apreciar, los puntos obtenidos siguen casi perfectamente la recta ideal.

6.5.2 Ejemplo con los datos de la serie temporal del láser

El segundo conjunto de datos empleado fue la serie temporal del láser de la competición de Santa Fe. En este caso se empleó un SOM formado por 30×30 neuronas. La dimensión de embebido empleada en este caso fue 5 y el número de vecinos empleados para el entrenamiento de las redes de una capa fue de 52. El conjunto de entrenamiento de la red contenía 2000 muestras y el de prueba 1000. La figura 6.8(a) contiene la trayectoria bidimensional formada por los datos de entrenamiento. La figura 6.8(b) presenta la trayectoria obtenida por las neuronas ganadoras en el SOM.

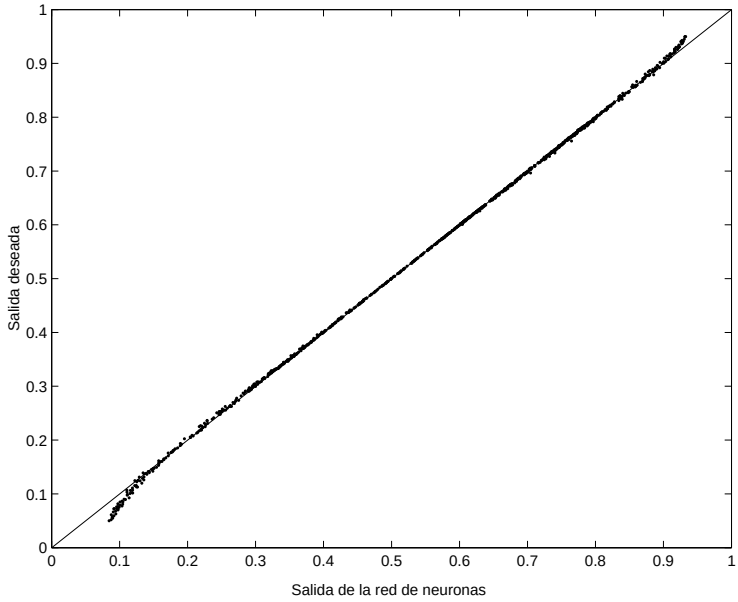
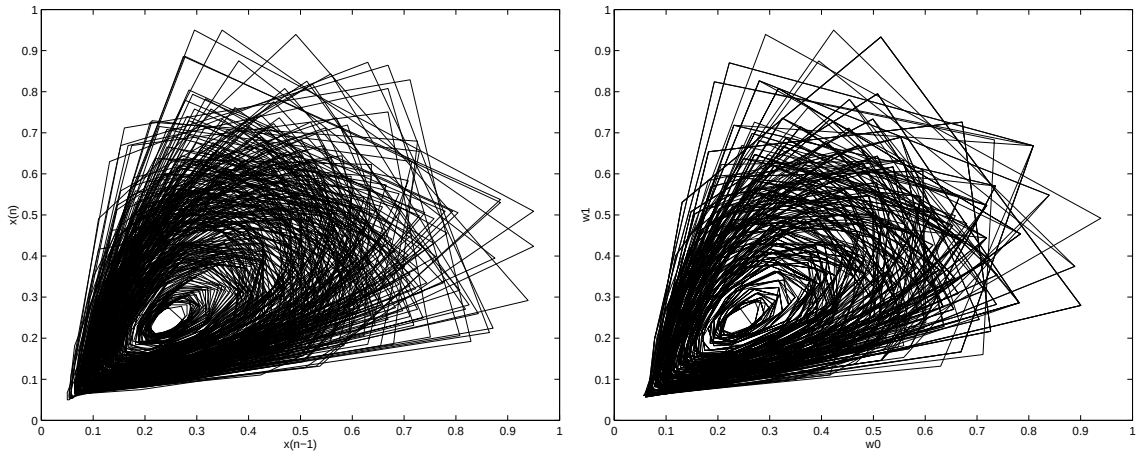


Figura 6.7: Salida real de la red frente a la salida deseada para los datos de Mackey-Glass.



(a) (b)

Figura 6.8: (a) Trayectoria bidimensional de la serie del láser en el espacio de reconstrucción y (b) trayectoria formada por las neuronas ganadoras del SOM.

Como se puede apreciar en estas figuras, ambas trayectorias son muy parecidas y, por lo tanto, la red es capaz de simular la dinámica del sistema de entrada. La figura 6.9 contiene la serie empleada en el conjunto de prueba (línea continua) y la salida del sistema neuronal propuesto (línea discontinua). El error cuadrático medio

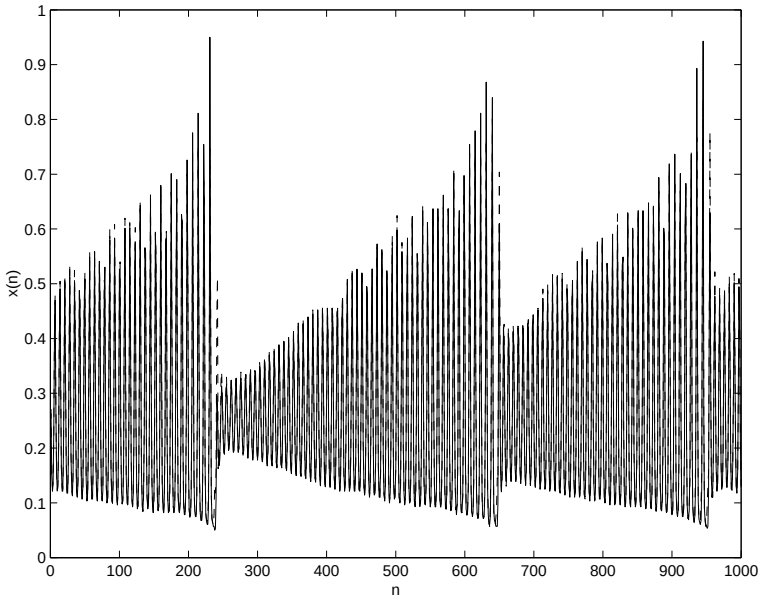


Figura 6.9: Serie temporal del láser (línea continua) y salida de la red propuesta (línea discontinua).

normalizado (ECMN) obtenido en el conjunto de prueba fue de 1.36×10^{-2}. La figura 6.10 muestra la salida real de la red frente a la salida deseada para el conjunto de prueba.

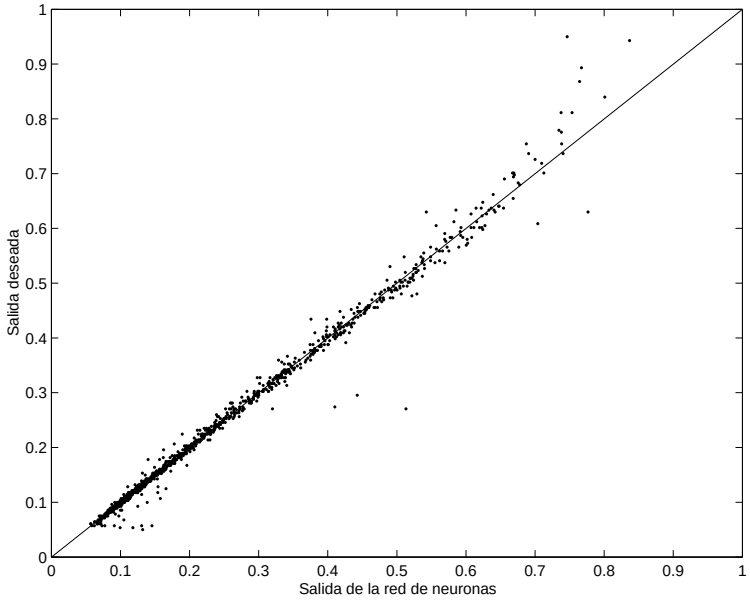


Figura 6.10: Salida real de la red frente a la salida deseada para los datos del láser.

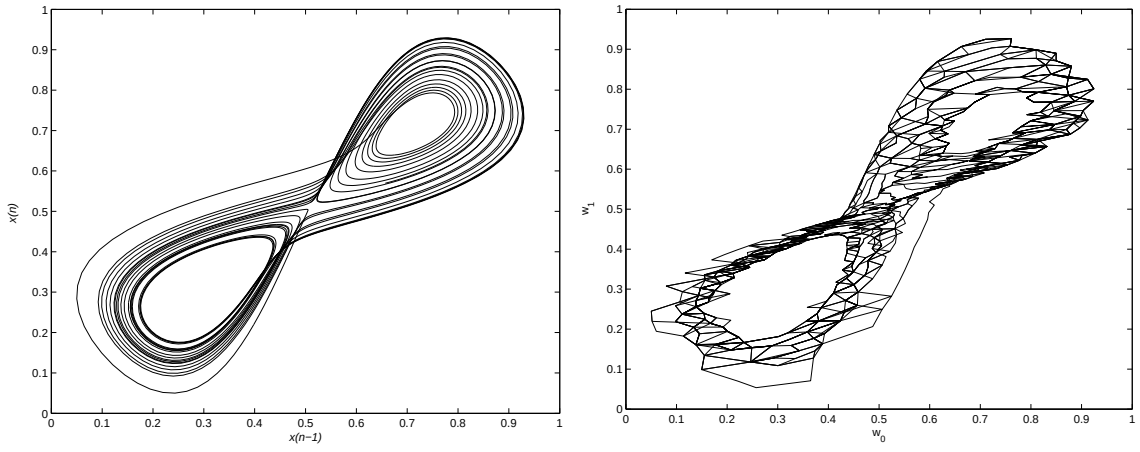
6.5.3 Ejemplo con los datos de la serie caótica de Lorenz

El último conjunto de datos empleado fue la serie caótica de Lorenz. Las ecuaciones de Lorenz [92] son:

$$\begin{aligned} \frac{dx}{dt} &= \sigma(y - x) \\ \frac{dy}{dt} &= x(r - z) - y \\ \frac{dz}{dt} &= xy - bz \end{aligned} \tag{6.12}$$

donde σ, r y b son constantes. En este trabajo se usaron los siguientes valores $\sigma = 10.0$, $r = 28.0$, $b = 8/3$, para los cuales el sistema presenta una dinámica caótica.

En este caso, se utilizó una malla de 25×25 neuronas en el SOM. La dimensión de embebido (N) empleada fue 4. Para realizar el entrenamiento de las redes de neuronas de una capa, se emplearon los pesos de la neurona ganadora y 41 vecinos. El conjunto de entrenamiento usado en este experimento contenía 3000 muestras y 1000 el conjunto de prueba. La figura 6.11(a) muestra la trayectoria bidimensional de los datos de entrenamiento, y la figura 6.11(b) contiene la trayectoria de las neuronas ganadoras del mapa autoorganizativo.



(a) (b)

Figura 6.11: (a) Trayectoria bidimensional de la serie caótica de Lorenz en el espacio de reconstrucción y (b) trayectoria formada por las neuronas ganadoras del SOM.

Como se puede apreciar en estas figuras el SOM ha capturado, de nuevo, la dinámica de los datos de entrada, ya que la trayectoria creada por las neuronas ganadoras reproduce fielmente la trayectoria del espacio de reconstrucción de la serie

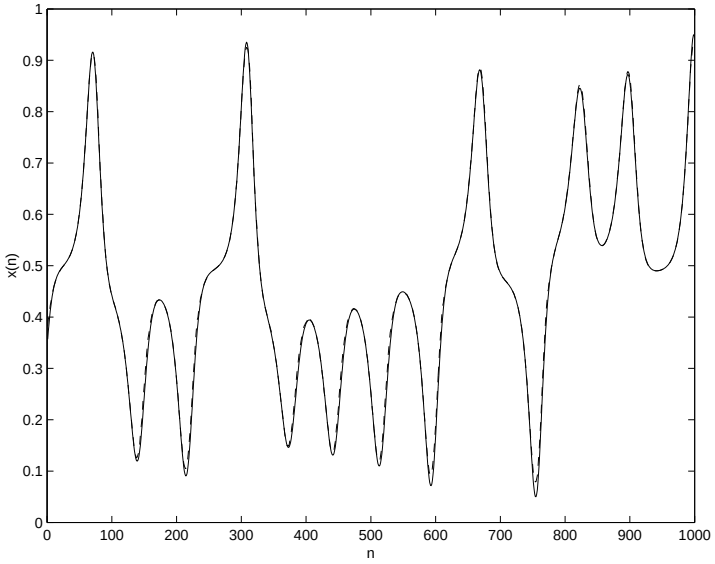


Figura 6.12: Serie temporal de Lorenz (línea continua) y salida de la red propuesta (línea discontinua).

temporal. La figura 6.12 muestra la serie de Lorenz empleada como conjunto de prueba (línea continua). Asimismo, la figura contiene la salida de la red después del entrenamiento (línea discontinua). Como se puede observar la diferencia entre ambas es prácticamente inapreciable visualmente. La figura 6.13 muestra la salida real de la red frente a la salida deseada para el conjunto de prueba.

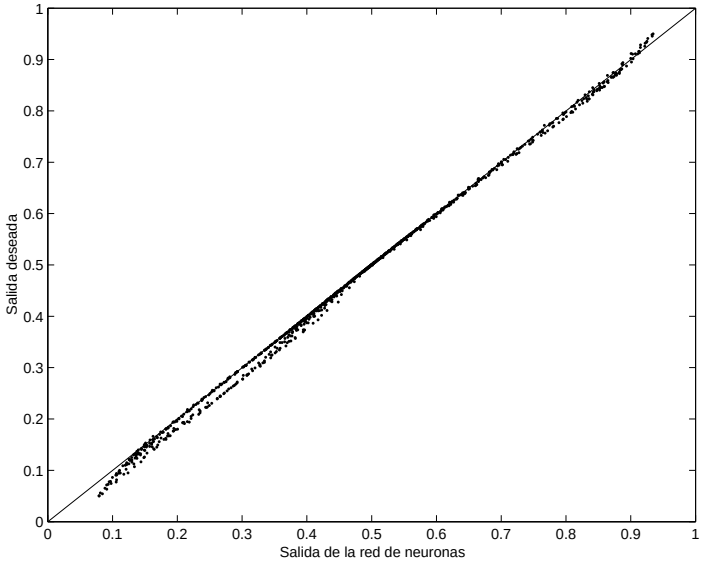


Figura 6.13: Salida real de la red frente a la salida deseada para los datos de Lorenz.

En esta figura se aprecia que los errores cometidos por el sistema son muy pequeños, puesto que los puntos están muy cerca de la recta ideal.

Además, para este conjunto de datos se realizó un estudio comparativo con un perceptrón multicapa entrenado con el algoritmo de Levenberg-Marquardt (LM). El entrenamiento del perceptrón con el algoritmo LM finalizaba cuando se cumplía alguna de las siguientes condiciones:

- El valor del gradiente de la función de error es menor que un valor mínimo (1×10^{-10}).
- Se alcanza un determinado número de iteraciones máximo (2000).

Este estudio se realizó para comparar el método de modelado local propuesto en este capítulo, en términos de error y tiempo de ejecución medio, con uno de los métodos estándar de modelado global en redes de neuronas. La tabla 6.1 muestra los resultados de este estudio, para el cual se realizaron 100 simulaciones, inicializando aleatoriamente los pesos en cada una de ellas.

Tabla 6.1: Tiempo medio de ejecución, error medio y desviación típica en las 100 simulaciones para la serie de Lorenz.

Algoritmo	Ratio	T. medio (s)	D. Típica T.	ECMN medio	D. Típica E.
SOM+SEL	1.0	16.94	6.419×10^{-2}	1.748×10^{-3}	1.735×10^{-4}
LM (4-5-1)	12.18	206.44	6.62	5.424×10^{-3}	1.177×10^{-2}
LM (4-7-1)	16.85	285.50	8.06	6.168×10^{-3}	6.724×10^{-3}
LM (4-9-1)	22.15	375.21	16.68	1.219×10^{-2}	1.648×10^{-2}
LM (4-11-1)	27.61	467.49	2.35	1.080×10^{-2}	1.158×10^{-2}

La tabla contiene los resultados para el método propuesto (SOM+SEL) y cuatro topologías diferentes: 4-5-1, 4-7-1, 4-9-1 y 4-11-1. En concreto, se muestra para cada red de neuronas el tiempo medio de ejecución (T. medio), la desviación típica (D. Típica T.) de este tiempo, la media del error cuadrático medio normalizado (ECMN medio) y la desviación típica (D. Típica E.) del error en las 100 ejecuciones. Además, la columna denominada ratio muestra, para cada algoritmo, cuantas veces es más lento, de media, que el algoritmo más rápido. Como se puede observar, el método propuesto es casi 12 veces más rápido que una red de neuronas 4-5-1 entrenada con el algoritmo de Levenberg-Marquardt, que es la que obtiene mejores resultados entre todas las topologías entrenadas. Además, el método SOM+SEL obtiene el error medio y la desviación típica más pequeñas en las simulaciones realizadas.

Las figuras 6.14(a), 6.14(b), 6.14(c) y 6.14(d) contienen los histogramas del ECMN del conjunto de prueba en las 100 simulaciones para la red SOM+SEL y tres de las

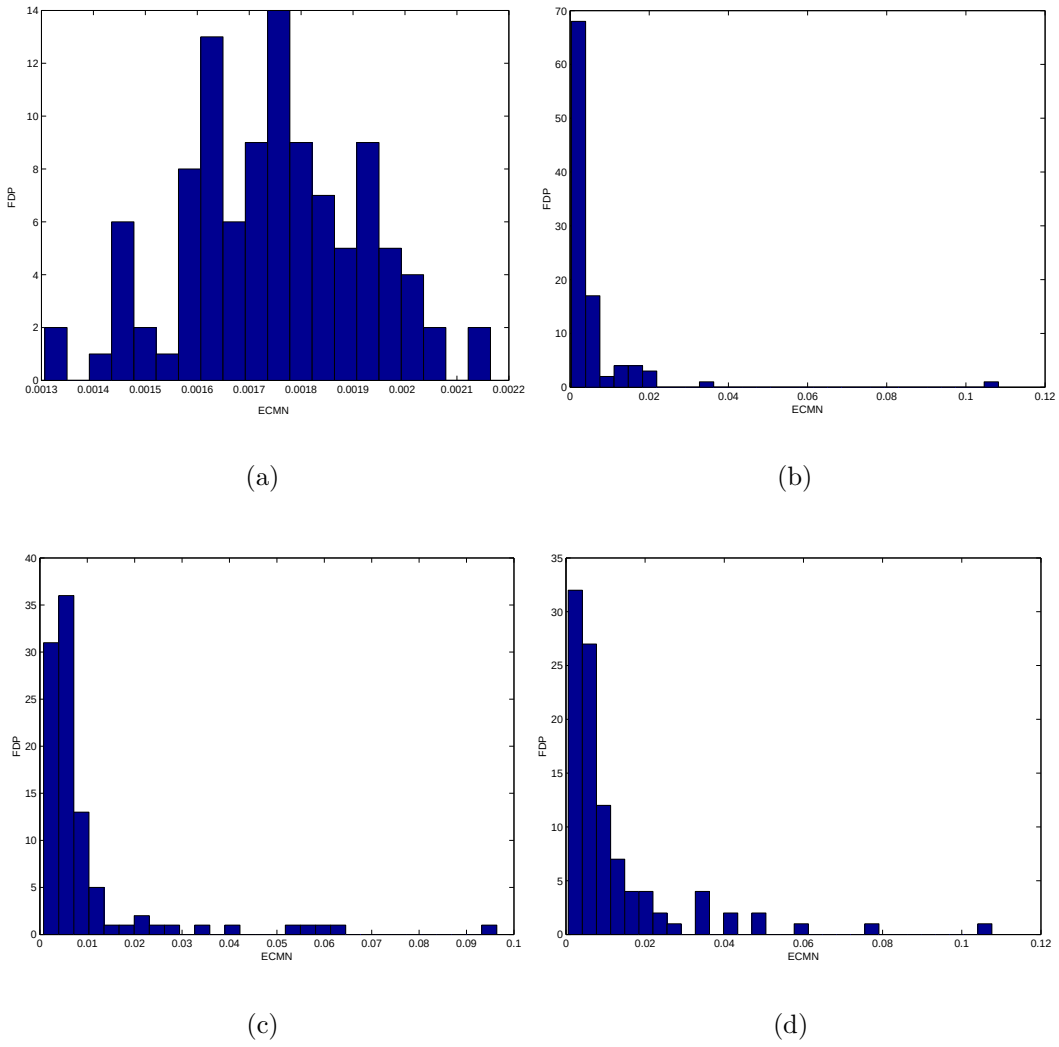


Figura 6.14: Histograma del ECMN al final del entrenamiento para (a) el método propuesto, (b) la red de neuronas 4-5-1, (c) la red de neuronas 4-7-1 y (d) la red de neuronas 4-9-1.

topologías: 4-5-1, 4-7-1 y 5-9-1, respectivamente. Como se observa en estas figuras el error alcanzado por el sistema SOM+LS presenta una menor variabilidad. Además, no es tan propenso a quedar atrapado en mínimos locales como el otro método empleado, puesto que todos los errores obtenidos son muy similares. Por el contrario, las diversas topologías de la red de neuronas entrenada con Levenberg-Marquardt quedan atrapadas varias veces en algún mínimo local, como se puede observar en las figuras 6.14(b), 6.14(c) y 6.14(d).

6.6 Discusión

En este capítulo se ha presentado un nuevo sistema neuronal híbrido basado en la creación de modelos locales de los datos de entrada. Esta aproximación combina la eficacia de los mapas autoorganizativos para obtener los modelos locales necesarios para lograr una buena aproximación, y la potencia del algoritmo propuesto en el tercer capítulo de este trabajo para ajustar de forma óptima los modelos locales mediante el uso de redes de neuronas de una capa.

Las principales características del método propuesto son:

- *Entrenamiento muy rápido del sistema.* Esto es una consecuencia directa de dos hechos: a) la rapidez del mapa autoorganizativo para determinar los modelos locales, y b) la gran velocidad del entrenamiento de las redes de una capa empleando el sistema de ecuaciones lineales propuesto en el tercer capítulo. Por ejemplo, si se emplea una malla en el SOM de 30×30 neuronas será necesario entrenar 900 redes de neuronas de una capa. Este proceso es bastante costoso con los métodos tradicionales, debido al gran número de redes a entrenar, sin embargo, con el aprendizaje basado en el sistema de ecuaciones este aprendizaje se realiza muy rápidamente. Por ello, el método propuesto es un procedimiento muy eficiente y rápido como ha sido corroborado en todos los experimentos realizados.

- *Entrenamiento más homogéneo.* Como se ha mostrado en el estudio comparativo de la sección 6.5.3, el método de modelado local presentado en este capítulo obtiene un histograma mucho más compacto (i.e., sin tanta variabilidad) que el de un perceptrón multicapa. Este hecho es muy beneficioso pues provoca que diversos entrenamientos del sistema neuronal proporcionen resultados muy similares, lo que genera una mayor confianza en el usuario. Esta homogeneidad está basada en el entrenamiento de las redes de una capa (que ajustan los modelos locales) empleando el sistema de ecuaciones propuesto en el tercer capítulo.

- *Mayor continuidad entre los modelos locales.* El uso de las redes de una capa con el entrenamiento óptimo mediante el sistema de ecuaciones asegura un entrenamiento más homogéneo de las redes, i.e., proporciona siempre la misma salida (no hay varianza como el caso de los algoritmos actuales), lo que provoca que exista aún una mayor continuidad entre los modelos locales derivados por el SOM. Esta continuidad entre los modelos repercute positivamente en el rendimiento global del sistema [35].

- *Aprendizaje incremental.* El aprendizaje incremental es inherente al algoritmo de Kohonen para la actualización de los pesos del SOM, ya que este tipo de aprendizaje permite actualizar su conocimiento con nuevos datos sin necesidad de utilizar de nuevo todo el conjunto de entrenamiento. Asimismo, el método de aprendizaje para redes de una capa propuesto presenta esta característica, como ya ha sido comentado previamente en la sección 3.6. Por tanto, el conocimiento del sistema en su conjunto puede actualizarse de manera sencilla.

Finalmente, es interesante mencionar que el modelado local es un método muy potente para la aproximación de funciones, ya que permite descomponer un problema complejo en pequeñas partes que, en general, serán más fáciles de abordar. Hasta el momento, el principal problema de este tipo de métodos era que requerían una gran cantidad de recursos computacionales para estimar los modelos locales adecuados, ya que era necesario analizar repetidamente los datos hasta encontrar los agrupamientos más idóneos. Sin embargo, con el uso del mapa autoorganizativo se simplifica enormemente esta tarea y, además el uso de las redes de una capa permite obtener, rápidamente, una buena aproximación para cada uno de los modelos locales.

Parte II

Medidas de inmunidad al ruido y tolerancia a fallos para redes funcionales y neuronales

Capítulo 7

Descripción de conceptos previos

"*Bien acierta quien sospecha que siempre yerra*"

Francisco de Quevedo y Villegas

La segunda parte de este trabajo de Tesis Doctoral aborda el desarrollo de medidas de inmunidad al ruido y tolerancia a fallos para redes funcionales y, por extensión, para redes neuronales con alimentación hacia delante. Por ello, antes de describir las medidas propuestas, en este capítulo se realizará una breve introducción a las redes funcionales y, posteriormente, una revisión de los estudios de inmunidad al ruido y tolerancia a fallos, realizados por otros autores, para redes de neuronas.

7.1 Redes funcionales

Las redes funcionales son un nuevo paradigma computacional introducido por Castillo et al. [25, 26] que generaliza a las redes de neuronas multicapa con alimentación hacia delante. Este tipo de redes combina conocimiento del problema, para determinar la estructura de la red, y datos, para estimar las funciones neuronales. En las redes funcionales las funciones neuronales son desconocidas e inicialmente se pueden suponer funciones multiargumento y multivariadas. Una característica importante de las redes funcionales es la posibilidad de tratar con restricciones funcionales determinadas por propiedades que pueden conocerse del modelo (e.g., asociatividad, distributividad, etc.). Estas restricciones se representan gráficamente mediante salidas coincidentes de algunas neuronas. Esto permite escribir el valor de estas unidades de salida de varias formas diferentes (una por cada salida coincidente) y conduce a un sistema de ecuaciones funcionales que se deduce directamente de la topología de la red. La resolución de este sistema de ecuaciones funcionales puede conducir a impor-

tantes simplificaciones de la topología de la red inicial y de las funciones neuronales, en el sentido de que algunas de las neuronas pueden ser eliminadas o simplificadas (reduciendo el número de argumentos). La figura 7.1 muestra algunos ejemplos de topologías de redes funcionales.

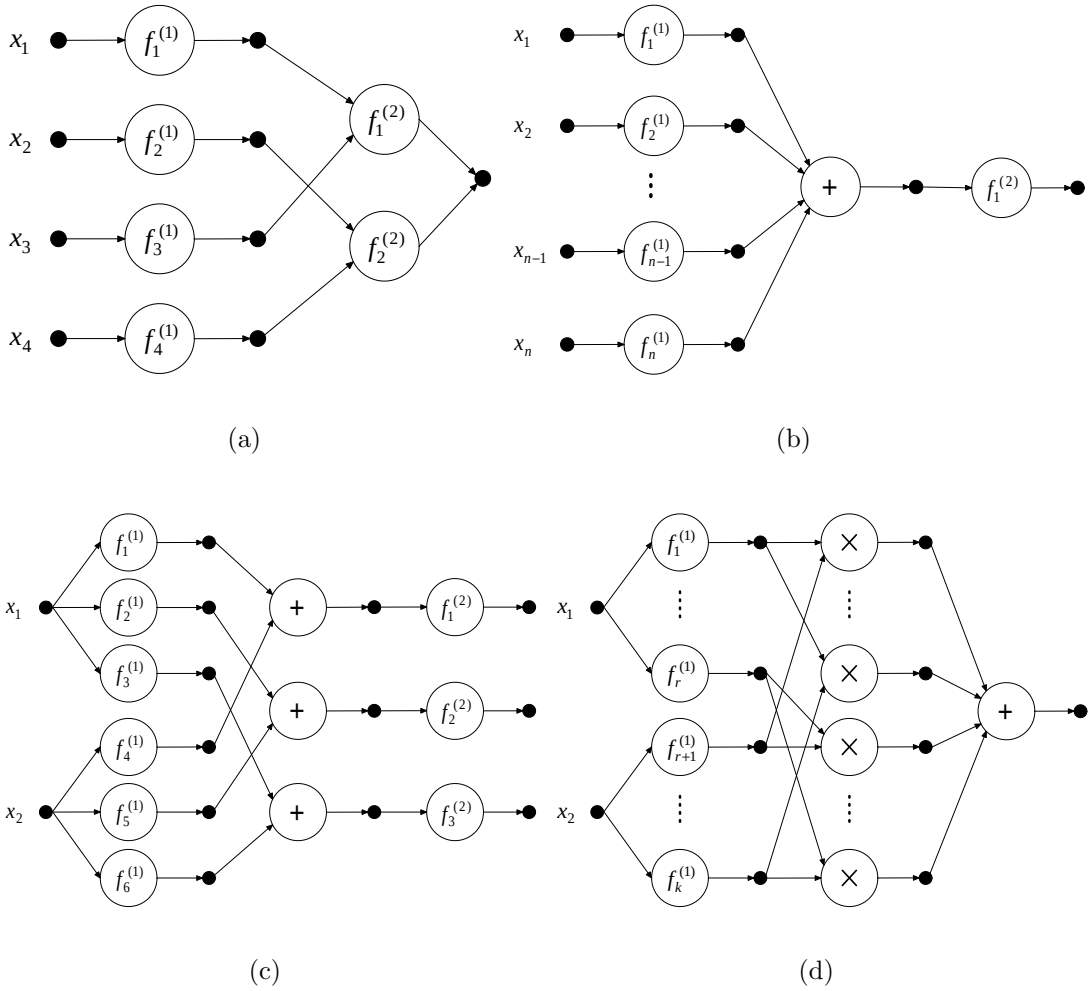


Figura 7.1: Algunos ejemplos de redes funcionales: (a) red inicial de la asociatividad generalizada, (b) red simplificada de la asociatividad generalizada, (c) red de salida múltiple dependiente y (d) red separable.

Una red funcional consta de los siguientes elementos [26]:

- *Una capa de entrada y otra de salida de unidades de almacenamiento.* Estas capas contienen los datos de entrada y salida de la red, respectivamente, y están representadas mediante pequeños círculos negros (véase figura 7.1).

- *Una o varias capas de unidades procesadoras o neuronas funcionales.* Estas unidades evalúan un conjunto de valores de entrada, que pueden provenir de unidades intermedias de almacenamiento o de entrada, y calculan un conjunto de valores de salida que serán empleados por la capa siguiente. Con este fin, cada neurona tiene asociada una función que puede ser multivariada y multiargumento. Las neuronas se representarán mediante círculos con el nombre de la función en su interior. Por ejemplo, la figura 7.1(b) contiene n funciones neuronales en la primera capa de neuronas, denominadas $f_i^{(1)}, i = 1, \ldots, n$.

- *Ninguna, una o varias capas de unidades de almacenamiento intermedias.* Estas capas contienen elementos que almacenan información intermedia producida por las neuronas. Las unidades intermedias se representan mediante pequeños círculos negros. Es importante destacar que estas capas permiten forzar la coincidencia de las salidas de las neuronas, véase por ejemplo la salida de la red en la figura 7.1(a).

- *Un conjunto de enlaces dirigidos.* Conectan unidades de entrada o intermedias con unidades neuronales, y unidades neuronales con unidades intermedias o de salida. Las conexiones se representan por flechas, indicando la dirección del flujo de la información.

A diferencia de las redes neuronales, en las redes funcionales hay dos tipos de aprendizaje:

- *Aprendizaje estructural*, que involucra dos procesos: a) selección de la topología inicial de la red, basada en algunas propiedades conocidas del problema a solucionar, y b) simplificación de la topología empleando ecuaciones funcionales [1, 30], que conduce a una arquitectura equivalente más simple.

- *Aprendizaje paramétrico*, que concierne a la estimación de las funciones neuronales. Este proceso puede realizarse considerando combinaciones lineales de familias funcionales y estimando los parámetros asociados con los datos disponibles. Este tipo de aprendizaje generaliza la idea de estimación de pesos de las conexiones de una red de neuronas. El aprendizaje paramétrico de estas redes se puede realizar empleando un método lineal, en el caso de aprendizaje supervisado, o un método no lineal para aprendizaje no supervisado.

Las principales diferencias entre las redes funcionales y neuronales son:

- En las redes funcionales es posible seleccionar la topología de la red empleando información disponible de los datos o del conocimiento previo del problema a resolver. Por el contrario, las redes de neuronas estándar sólo utilizan los datos.

- En las redes neuronales estándar sólo se aprenden los pesos, ya que las funciones neuronales se suponen fijas y conocidas. Sin embargo, en las redes funcionales se aprenden las funciones neuronales durante el aprendizaje estructural y el aprendizaje paramétrico.

- En las redes funcionales no se usan pesos, ya que se pueden incorporar dentro de las funciones neuronales.

- En las redes funcionales se permite que las funciones neuronales sean multiargumento y multivariadas. En las redes neuronales las funciones de activación tienen un único argumento, aunque éste es una combinación lineal de todas las entradas (funciones pseudo-multiargumento).

- En las redes funcionales, la salida de varias neuronas pueden ser conectadas por unidades de almacenamiento indicando que los valores asociados tienen que ser iguales. Cada una de estas conexiones comunes representa una restricción funcional al modelo y permite escribir el valor de estas unidades de diferentes formas. Esto conduce a un sistema de ecuaciones funcionales. Resolviendo este sistema la topología inicial de la red puede simplificarse.

- Las redes funcionales son extensiones de las redes neuronales.

Finalmente, es interesante destacar que las redes funcionales han sido empleadas en varios problemas prácticos mejorando, en algunos casos, los resultados obtenidos por las redes de neuronas (véase, por ejemplo, [28, 29]).

7.2 Inmunidad al ruido y tolerancia a fallos en redes de neuronas

Las redes de neuronas artificiales fueron desarrolladas inicialmente inspirándose en las redes de neuronas naturales y, por lo tanto, ha sido asumido habitualmente que este tipo de sistemas es tolerante a fallos. Sin embargo, algunos trabajos recientes han demostrado que las redes de neuronas artificiales no son, inherentemente, tolerantes a fallos [24, 105, 119]. En concreto, en el campo de las redes de neuronas con alimentación hacia delante se han realizado diversos estudios para comprobar la influencia de perturbaciones en la entrada y en los pesos de estos sistemas [18, 19, 40, 96].

En el caso de perceptrones multicapa se puede demostrar que dada una determinada topología, se pueden obtener diferentes conjuntos de pesos después del entrenamiento, dependiendo de la condición inicial de los parámetros de la red. Estas soluciones pueden presentar un rendimiento similar, respecto al error medio o error

de clasificación alcanzado, pero, sin embargo, ser muy diferentes en cuanto a la tolerancia a fallos. Consecuentemente, algunos conjuntos de pesos presentarán una mayor tolerancia frente a perturbaciones que otros. Además, cuando el proceso de aprendizaje de la red se realiza en un computador y posteriormente ésta es implementada en algún medio físico, las diferencias entre la precisión alcanzada durante el entrenamiento y la precisión de la implementación puede afectar significativamente al rendimiento de la red. Asimismo, en el caso de implementaciones analógicas, los valores de los pesos pueden variar dentro de unos márgenes de tolerancia, lo cual afectará al rendimiento teórico alcanzado en las simulaciones [41]. Por consiguiente, es de gran utilidad disponer de alguna medida para determinar, a priori, la tolerancia a fallos (ruido en los pesos) y la inmunidad al ruido (ruido en las entradas), de forma que se pueda tener un criterio de selección entre diferentes conjuntos de pesos que presenten un rendimiento similar. La mayoría de los trabajos relacionados con la tolerancia a fallos la evalúan experimentalmente como la degradación del aprendizaje respecto al rendimiento obtenido, cuando los fallos se presentan por medio de simulaciones [101]. En este sentido, Segee y Carter [119] propusieron medir la tolerancia a fallos a partir de simulaciones empleando la hipótesis del peor caso. Sin embargo, esta aproximación fue considerada poco realista por Edwards y Murray [42] que propusieron una medida, denominada *saliency*, para evaluar la tolerancia a fallos frente a las desviaciones de los pesos. Esta medida se calcula a partir de la matriz hessiana del error de la salida respecto a los pesos de la red [43], indicando un valor pequeño de ésta que la superficie de error alrededor del punto actual (determinado por el valor de los pesos) no es muy abrupta y, por consiguiente, un cambio en los pesos, producirá un cambio relativamente pequeño en el rendimiento del sistema.

Asimismo, Choi y Choi propusieron la sensibilidad estadística como una medida de tolerancia a fallos o inmunidad al ruido para perceptrones multicapa [33]. La sensibilidad estadística es diferente de la sensibilidad de la salida, que hace referencia a la primera derivada de la salida de la red respecto a los pesos [39]. La sensibilidad estadística evalúa la variación del rango de la salida de una neurona cuando sus pesos o entradas están distorsionadas. Posteriormente, y basándose en el trabajo de Choi y Choi, Bernier et al. [12, 13, 14] presentaron una nueva medida para estimar la inmunidad al ruido producido por desviaciones en las entradas de un perceptrón multicapa. Estos autores demostraron la relación existente entre el error cuadrático medio y la sensibilidad estadística. Además, propusieron una nueva medida, denominada sensibilidad cuadrática media, que predice de manera precisa la degradación del ECM cuando se aplica ruido en las entradas o en los pesos del sistema. Esta medida puede ser empleada como un criterio de selección entre diferentes redes y conjuntos de pesos que presenten un rendimiento similar pero que tengan una respuesta diferente al ruido. Posteriormente, los mismos autores desarrollaron una medida similar, pero

para redes con funciones de base radial (*radial basis functions*) [11].

Sin embargo, esta clase de estudios no se han realizado para redes funcionales. Por ello, en el siguiente capítulo se proponen unas nuevas medidas para redes funcionales basadas en la sensibilidad estadística. Como se verá posteriormente, estas medidas también son aplicables a redes de neuronas con alimentación hacia delante y generalizan las propuestas por Bernier et al. [11, 12, 13, 14]. Como se ha mencionado anteriormente, estos autores propusieron varias medidas, basadas en la sensibilidad estadística, dependiendo del tipo de red empleada, perceptrón multicapa o redes de base radial. Sin embargo, la medida que se propone a continuación es aplicable a ambas y, en general, a cualquier tipo de red de neuronas con alimentación hacia delante.

Capítulo 8

Inmunidad al ruido y tolerancia a fallos en redes funcionales

"Los errores son inevitables. Lo que cuenta es cómo respondemos a ellos"

Nikki Giovanni

8.1 Sensibilidad estadística de una red funcional

Considérese la red funcional generalizada de la figura 8.1. Esta red está compuesta de M capas, donde cada capa m tiene N_m neuronas. Cada neurona i en la capa m está conectada a las N_{m+1} neuronas de la capa $m+1$. El conjunto $\{y_1^{(0)}, \ldots, y_{N_0}^{(0)}\}$ es la entrada de la red y la salida de la neurona i en la capa m es

$$y_i^{(m)} = f_i^{(m)}(y_1^{(m-1)}, \ldots, y_{N_{m-1}}^{(m-1)});\ i = 1, \ldots N_m;\ m = 1, \ldots, M, \tag{8.1}$$

donde cada $f_i^{(m)}$ se define como la siguiente composición de funciones:

$$\begin{gathered} f_i^{(m)}(y_1^{(m-1)}, \ldots, y_{N_{m-1}}^{(m-1)}) = g_i^{(m)}(h_{i1}^{(m)}(y_1^{(m-1)}), \ldots, h_{iN_{m-1}}^{(m)}(y_{N_{m-1}}^{(m-1)})); \\ i = 1, \ldots N_m;\ m = 1, \ldots, M\ . \end{gathered} \tag{8.2}$$

Las funciones $g_i^{(m)}$ son conocidas y fijas durante el proceso de entrenamiento, y cada función $h_{ij}^{(m)}$ es una combinación lineal de funciones conocidas que será determinada durante el proceso de aprendizaje. Por tanto, se tiene que

$$\begin{gathered} h_{ij}^{(m)}(y_j^{(m-1)}) = \sum_{z=1}^{n_{ij}^{(m)}} a_{ijz}^{(m)} \phi_{ijz}^{(m)}(y_j^{(m-1)}); \\ i = 1, \ldots N_m;\ j = 1, \ldots N_{m-1};\ m = 1, \ldots, M, \end{gathered} \tag{8.3}$$

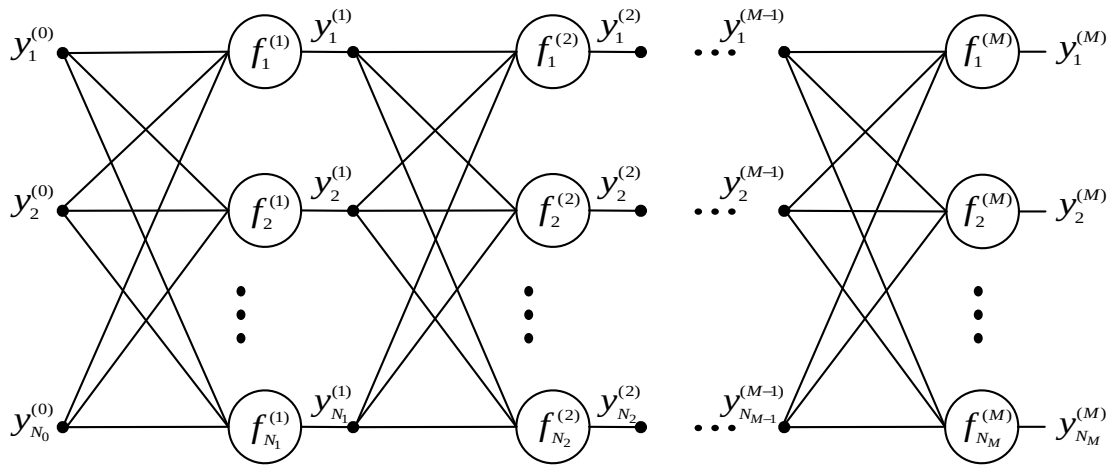


Figura 8.1: Modelo de red funcional generalizado.

donde $n_{ij}^{(m)}$ es el número de funciones básicas y los coeficientes $a_{ijz}^{(m)}$ son los parámetros de la red funcional. La red funcional presentada en la figura 8.1 es un modelo generalizado a partir de los propuestos por Castillo et al. [25][26][28]. Además, es importante destacar que el modelo presentado es también una generalización de una red de neuronas multicapa con alimentación hacia delante. Por ejemplo, con la arquitectura propuesta se puede modelar una red de neuronas con alimentación hacia delante con funciones de activación tangentes hiperbólicas empleando

$$\begin{gathered} g_i^{(m)}(h_{i1}^{(m)}(y_1^{(m-1)}),\ldots,h_{iN_{m-1}}^{(m)}(y_{N_{m-1}}^{(m-1)})) = \\ \tanh(h_{i1}^{(m)}(y_1^{(m-1)})+\cdots+h_{iN_{m-1}}^{(m)}(y_{N_{m-1}}^{(m-1)}))\ , \\ \phi_{ijz}^{(m)}(y_j^{(m-1)}) = y_j^{(m-1)}, \\ n_{ij}^{(m)} = 1;\ \forall i,j. \end{gathered} \tag{8.4}$$

En este caso, los pesos de la capa m de la red de neuronas ($w_{ij}^{(m)};\ i=1,\ldots,N_m;\ j=1,\ldots,N_{m-1}$) son equivalentes a los parámetros $a_{ij1}^{(m)}$ en el modelo propuesto.

En la red presentada, la salida de una neurona, $y_i^{(m)}$, cambiará si hay variaciones en los valores de los parámetros $a_{ijz}^{(m)}$, o bien, de las entradas de la neurona ($y_j^{(m-1)};\ j=1,\ldots,N_{m-1}$). Si estos cambios son pequeños, se pueden aproximar como

$$\Delta y_i^{(m)} \approx \sum_{r=1}^{N_{m-1}} \left(\frac{\partial y_i^{(m)}}{\partial y_r^{(m-1)}} \Delta y_r^{(m-1)} + \sum_{s=1}^{n_{ir}^{(m)}} \frac{\partial y_i^{(m)}}{\partial a_{irs}^{(m)}} \Delta a_{irs}^{(m)} \right) \ . \tag{8.5}$$

Sea $u_{ij}^{(m)} = h_{ij}^{(m)}(y_j^{(m-1)});\ i=1,\ldots N_m;\ j=1,\ldots N_{m-1};\ m=1,\ldots,M$, entonces las

derivadas parciales en la ecuación 8.5 son iguales a

$$\frac{\partial y_i^{(m)}}{\partial y_r^{(m-1)}} = \frac{\partial y_i^{(m)}}{\partial u_{ir}^{(m)}} \frac{\partial u_{ir}^{(m)}}{\partial y_r^{(m-1)}} = \frac{\partial y_i^{(m)}}{\partial u_{ir}^{(m)}} \sum_{z=1}^{n_{ir}^{(m)}} a_{irz}^{(m)} \frac{d\phi_{irz}^{(m)}(y_r^{(m-1)})}{dy_r^{(m-1)}} \tag{8.6}$$

y

$$\frac{\partial y_i^{(m)}}{\partial a_{irs}^{(m)}} = \frac{\partial y_i^{(m)}}{\partial u_{ir}^{(m)}} \frac{\partial u_{ir}^{(m)}}{\partial a_{irs}^{(m)}} = \frac{\partial y_i^{(m)}}{\partial u_{ir}^{(m)}} \phi_{irs}^{(m)}(y_r^{(m-1)}) \ . \tag{8.7}$$

Choi y Choi propusieron la sensibilidad estadística como una medida de tolerancia a las perturbaciones en perceptrones multicapa [33]. La sensibilidad estadística, que mide las alteraciones de la salida cuando hay cambios en los valores de los parámetros, se define como

$$S_i^{(m)} = \lim_{\sigma \to 0} \frac{\sqrt{var(\Delta y_i^{(m)})}}{\sigma} \ , \tag{8.8}$$

donde σ es la desviación típica de los cambios y $var(\Delta y_i^{(m)})$ es la varianza de la desviación de la salida debido a esos cambios. La sensibilidad estadística mide la variación esperada de la salida frente a perturbaciones, que pueden proceder de las entradas o bien de los pesos. Dado un determinado rango de perturbaciones, especificado por la desviación típica correspondiente, cuanto mayor es la sensibilidad estadística mayor será la variación de la salida. Si se consideran las perturbaciones sobre los parámetros, la sensibilidad estadística evalúa la tolerancia a fallos de una neurona. Por otro lado, cuando las perturbaciones afectan a las entradas, la sensibilidad estadística es una medida de inmunidad al ruido.

La sensibilidad estadística es una medida diferente que la sensibilidad de la salida. Esta última se calcula a partir de la matriz jacobiana del error respecto a los parámetros y mide la dependencia de la salida con respecto de los valores de los parámetros (está relacionado con las primeras derivadas). Un valor de la sensibilidad de la salida igual a cero implica que la salida es independiente de los valores de los parámetros y, por tanto, la red no ha aprendido el patrón de entrada. Sin embargo, la sensibilidad estadística, a diferencia de la sensibilidad de la salida, está relacionada con la matriz hessiana del error respecto a los parámetros, por consiguiente, constituye una medida de la ausencia de cambios abruptos en la superficie de error (alrededor del punto actual). Consecuentemente, un valor pequeño de la sensibilidad estadística implica una variación pequeña de la salida cuando se producen perturbaciones en los parámetros de la red.

En las siguientes secciones se muestran las expresiones obtenidas en este trabajo, en base a la sensibilidad estadística, para estimar el cambio en la salida de una neurona cuando se producen variaciones en las entradas o bien en los parámetros. En ambos casos se ha considerado que el ruido es de carácter aditivo. Este tipo de ruido se

usa habitualmente para modelar los efectos de la cuantificación en implementaciones digitales o para modelar los efectos de *offset* de los transistores.

8.2 Sensibilidad estadística para ruido en las entradas

Para obtener la medida de inmunidad al ruido se asume que la distribución del ruido tiene media cero y que además no existe una correlación entre las entradas. Realizando estas asunciones, se satisfacen las siguientes condiciones

- (e.1) $E[\Delta y_i^{(0)}] = 0;\ i = 1, \ldots, N_0$,
- (e.2) $E[\Delta y_i^{(0)} \Delta y_j^{(0)}] = \sigma^2 \delta_{ij};\ i, j = 1, \ldots, N_0$,

donde el parámetro σ representa la desviación típica de la distribución del ruido, δ es la delta de Kronecker y $E[\cdot]$ es el operador esperanza. A continuación se presenta un teorema que será empleado posteriormente para el desarrollo de la medida.

Teorema 4 *Si* $E[\Delta y_i^{(0)}] = 0;\ \forall r = 1, \ldots, N_0$ *entonces* $E[\Delta y_i^{(m)}] = 0;\ \forall i = 1, \ldots, N_m;$ $\forall m = 1, \ldots, M$

Demostración:
La demostración de este teorema se realizará por inducción en m y asumiendo que los parámetros de la red están libres de errores, i.e., $\Delta a_{irs}^{(m)} = 0;\ \forall i, r, s, m.$

Para $m = 1$:

$$\begin{aligned} E[\Delta y_i^{(1)}] &= E\left[\sum_{r=1}^{N_0}\left(\frac{\partial y_i^{(1)}}{\partial y_r^{(0)}}\Delta y_r^{(0)} + \sum_{s=1}^{n_{ir}^{(1)}}\frac{\partial y_i^{(1)}}{\partial a_{irs}^{(1)}}\Delta a_{irs}^{(1)}\right)\right] \\ &= \sum_{r=1}^{N_0}\left(\frac{\partial y_i^{(1)}}{\partial y_r^{(0)}}E[\Delta y_r^{(0)}] + \sum_{s=1}^{n_{ir}^{(1)}}\frac{\partial y_i^{(1)}}{\partial a_{irs}^{(1)}}E[\Delta a_{irs}^{(1)}]\right) = 0\ . \end{aligned}$$

Supóngase que el resultado es cierto para $m-1$, i.e., $E[\Delta y_i^{(m-1)}] = 0\ \forall i = 1, \ldots, N_{m-1}$, entonces para la capa m se tiene:

$$\begin{aligned} E[\Delta y_i^{(m)}] &= E\left[\sum_{r=1}^{N_{m-1}}\left(\frac{\partial y_i^{(m)}}{\partial y_r^{(m-1)}}\Delta y_r^{(m-1)} + \sum_{s=1}^{n_{ir}^{(m)}}\frac{\partial y_i^{(m)}}{\partial a_{irs}^{(m)}}\Delta a_{irs}^{(m)}\right)\right] \\ &= \sum_{r=1}^{N_{m-1}}\left(\frac{\partial y_i^{(m)}}{\partial y_r^{(m-1)}}E[\Delta y_r^{(m-1)}] + \sum_{s=1}^{n_{ir}^{(m)}}\frac{\partial y_i^{(m)}}{\partial a_{irs}^{(m)}}E[\Delta a_{irs}^{(m)}]\right) = 0\ . \end{aligned}$$

■

Una vez demostrado el resultado anterior, el siguiente teorema presenta la sensibilidad estadística, frente a perturbaciones en las entradas, para cada una de las neuronas de la red funcional.

Teorema 5 *La sensibilidad estadística de la neurona i en la capa m al ruido en las entradas se puede expresar como*

$$S_i^{(m)} = \sqrt{\sum_{r=1}^{N_{m-1}} \sum_{z=1}^{N_{m-1}} \frac{\partial y_i^{(m)}}{\partial y_r^{(m-1)}} \frac{\partial y_i^{(m)}}{\partial y_z^{(m-1)}} C_{rz}^{(m-1)}};\ \forall i = 1, \ldots, N_m;\ \forall m = 1, \ldots, M \quad (8.9)$$

donde los términos $C_{rz}^{(m)}$ se calculan recursivamente mediante la siguiente expresión:

$$C_{rz}^{(m)} = \sum_{k=1}^{N_{m-1}} \sum_{s=1}^{N_{m-1}} \frac{\partial y_r^{(m)}}{\partial y_k^{(m-1)}} \frac{\partial y_z^{(m)}}{\partial y_s^{(m-1)}} C_{ks}^{(m-1)} \ . \quad (8.10)$$

Demostración:

$$\begin{aligned}
E[\Delta y_i^{(m)} \Delta y_k^{(m)}] &= E\left[\left(\sum_{r=1}^{N_{m-1}} \frac{\partial y_i^{(m)}}{\partial y_r^{(m-1)}} \Delta y_r^{(m-1)} + \sum_{s=1}^{n_{ir}^{(m)}} \frac{\partial y_i^{(m)}}{\partial a_{irs}^{(m)}} \Delta a_{irs}^{(m)} \right) \right. \\
&\qquad \left. \left(\sum_{z=1}^{N_{m-1}} \frac{\partial y_k^{(m)}}{\partial y_z^{(m-1)}} \Delta y_z^{(m-1)} + \sum_{t=1}^{n_{kz}^{(m)}} \frac{\partial y_k^{(m)}}{\partial a_{kzt}^{(m)}} \Delta a_{kzt}^{(m)} \right) \right] \\
&= \sum_{r=1}^{N_{m-1}} \sum_{z=1}^{N_{m-1}} \frac{\partial y_i^{(m)}}{\partial y_r^{(m-1)}} \frac{\partial y_k^{(m)}}{\partial y_z^{(m-1)}} E[\Delta y_r^{(m-1)} \Delta y_z^{(m-1)}] \\
&\quad + \frac{\partial y_i^{(m)}}{\partial y_r^{(m-1)}} \sum_{t=1}^{n_{kz}^{(m)}} \frac{\partial y_k^{(m)}}{\partial a_{kzt}^{(m)}} E[\Delta y_r^{(m-1)} \Delta a_{kzt}^{(m)}] \\
&\quad + \frac{\partial y_k^{(m)}}{\partial y_z^{(m-1)}} \sum_{s=1}^{n_{ir}^{(m)}} \frac{\partial y_i^{(m)}}{\partial a_{irs}^{(m)}} E[\Delta a_{irs}^{(m)} \Delta y_z^{(m-1)}] \\
&\quad + \sum_{s=1}^{n_{ir}^{(m)}} \sum_{t=1}^{n_{kz}^{(m)}} \frac{\partial y_i^{(m)}}{\partial a_{irs}^{(m)}} \frac{\partial y_k^{(m)}}{\partial a_{kzt}^{(m)}} E[\Delta a_{irs}^{(m)} \Delta a_{kzt}^{(m)}] \\
&= \sum_{r=1}^{N_{m-1}} \sum_{z=1}^{N_{m-1}} \frac{\partial y_i^{(m)}}{\partial y_r^{(m-1)}} \frac{\partial y_k^{(m)}}{\partial y_z^{(m-1)}} E[\Delta y_r^{(m-1)} \Delta y_z^{(m-1)}] \ .
\end{aligned}$$

Si se define $C_{rz}^{(m)}$ como $C_{rz}^{(m)} \equiv \dfrac{E[\Delta y_r^{(m)} \Delta y_z^{(m)}]}{\sigma^2}$ entonces

$$E[\Delta y_i^{(m)} \Delta y_k^{(m)}] = \sigma^2 \sum_{r=1}^{N_{m-1}} \sum_{z=1}^{N_{m-1}} \frac{\partial y_i^{(m)}}{\partial y_r^{(m-1)}} \frac{\partial y_k^{(m)}}{\partial y_z^{(m-1)}} C_{rz}^{(m-1)} \tag{8.11}$$

donde

$$C_{rz}^{(m)} = \sum_{k=1}^{N_{m-1}} \sum_{s=1}^{N_{m-1}} \frac{\partial y_r^{(m)}}{\partial y_k^{(m-1)}} \frac{\partial y_z^{(m)}}{\partial y_s^{(m-1)}} C_{ks}^{(m-1)} \ . \tag{8.12}$$

Empleando $var(\Delta y_i^{(m)}) = E[(\Delta y_i^{(m)})^2] - (E[\Delta y_i^{(m)}])^2$ y el teorema 4 en la ecuación (8.8) se obtiene:

$$S_i^{(m)} = \frac{\sqrt{E[(\Delta y_i^{(m)})^2]}}{\sigma} \ . \tag{8.13}$$

Sustituyendo $E[(\Delta y_i^{(m)})^2] = \sigma^2 C_{ii}^{(m)}$ en la ecuación (8.19) se obtiene:

$$S_i^{(m)} = \frac{\sqrt{\sigma^2 C_{ii}^{(m)}}}{\sigma} = \sqrt{\sum_{r=1}^{N_{m-1}} \sum_{z=1}^{N_{m-1}} \frac{\partial y_i^{(m)}}{\partial y_r^{(m-1)}} \frac{\partial y_i^{(m)}}{\partial y_z^{(m-1)}} C_{rz}^{(m-1)}} \ . \tag{8.14}$$

■

Por último, la condición inicial para la ecuación (8.10), $C_{rz}^{(0)} = \delta_{rz}$, se puede obtener fácilmente empleando la definición de $C_{rz}^{(m)}$ y (e.2).

8.3 Sensibilidad estadística para ruido en los parámetros

A continuación, se analizará la sensibilidad estadística de las neuronas cuando las perturbaciones se producen sobre los parámetros de la red. En este caso, también se considerará que la distribución del ruido tiene media cero y que no hay una correlación entre las perturbaciones que afectan a cada parámetro. Por lo tanto, se cumplen las siguientes condiciones:

- (p.1) $E[\Delta a_{irs}^{(m)}] = 0; \ \forall i, r, s, m$
- (p.2) $E[\Delta a_{irs}^{(m)} \Delta a_{i'r's'}^{(m')}] = \sigma^2 \delta_{ii'} \delta_{rr'} \delta_{ss'} \delta_{mm'}; \ \forall i, r, s, m, i', r', s', m'$

donde σ representa la desviación típica de la distribución del ruido, δ es la función delta de Kronecker y $E[\cdot]$ es el operador esperanza. Dadas estas condiciones se demostrará el siguiente teorema, que será empleado posteriormente.

Teorema 6 *Si* $E[\Delta a_{irs}^{(m)}] = 0;\ \forall i, r, s, m$ *entonces* $E[\Delta y_i^{(m)}] = 0$

Demostración:
Se realizará la demostración por inducción sobre m y asumiendo que las entradas de la red están libres de errores, i.e., $\Delta y_i^{(0)} = 0;\ i = 1, \ldots, N_0$.

Para $m = 1$:

$$\begin{aligned} E[\Delta y_i^{(1)}] &= E\left[\sum_{r=1}^{N_0}\left(\frac{\partial y_i^{(1)}}{\partial y_r^{(0)}}\Delta y_r^{(0)} + \sum_{s=1}^{n_{ir}^{(1)}}\frac{\partial y_i^{(1)}}{\partial a_{irs}^{(1)}}\Delta a_{irs}^{(1)}\right)\right] \\ &= \sum_{r=1}^{N_0}\left(\frac{\partial y_i^{(1)}}{\partial y_r^{(0)}}E[\Delta y_r^{(0)}] + \sum_{s=1}^{n_{ir}^{(1)}}\frac{\partial y_i^{(1)}}{\partial a_{irs}^{(1)}}E[\Delta a_{irs}^{(1)}]\right) = 0 \end{aligned}$$

Suponiendo que el resultado es cierto para $m-1$, i.e., $E[\Delta y_i^{(m-1)}] = 0;\ \forall i = 1, \ldots, N_{m-1}$, entonces para la capa m:

$$\begin{aligned} E[\Delta y_i^{(m)}] &= E\left[\sum_{r=1}^{N_{m-1}}\left(\frac{\partial y_i^{(m)}}{\partial y_r^{(m-1)}}\Delta y_r^{(m-1)} + \sum_{s=1}^{n_{ir}^{(m)}}\frac{\partial y_i^{(m)}}{\partial a_{irs}^{(m)}}\Delta a_{irs}^{(m)}\right)\right] \\ &= \sum_{r=1}^{N_{m-1}}\left(\frac{\partial y_i^{(m)}}{\partial y_r^{(m-1)}}E[\Delta y_r^{(m-1)}] + \sum_{s=1}^{n_{ir}^{(m)}}\frac{\partial y_i^{(m)}}{\partial a_{irs}^{(m)}}E[\Delta a_{irs}^{(m)}]\right) = 0 \end{aligned}$$

■

Una vez demostrado el resultado anterior a continuación se muestra, en el siguiente teorema, la sensibilidad estadística de cada una de las neuronas de la red funcional cuando se producen perturbaciones sobre sus parámetros.

Teorema 7 *La sensibilidad estadística de la neurona i en la capa m al ruido en los parámetros se puede expresar como*

$$S_i^{(m)} = \sqrt{\sum_{r=1}^{N_{m-1}}\left[\left(\sum_{z=1}^{N_{m-1}}\frac{\partial y_i^{(m)}}{\partial y_r^{(m-1)}}\frac{\partial y_i^{(m)}}{\partial y_z^{(m-1)}}C_{rz}^{(m-1)}\right) + \sum_{s=1}^{n_{ir}^{(m)}}\left(\frac{\partial y_i^{(m)}}{\partial a_{irs}^{(m)}}\right)^2\right]} \tag{8.15}$$

donde los términos $C_{rz}^{(m)}$ *se calculan recursivamente como*

$$C_{rz}^{(m)} = \sum_{p=1}^{N_{m-1}}\left[\left(\sum_{k=1}^{N_{m-1}}\frac{\partial y_r^{(m)}}{\partial y_p^{(m-1)}}\frac{\partial y_z^{(m)}}{\partial y_k^{(m-1)}}C_{pk}^{(m-1)}\right) + \delta_{rz}\sum_{s=1}^{n_{rp}^{(m)}}\sum_{t=1}^{n_{zp}^{(m)}}\frac{\partial y_r^{(m)}}{\partial a_{rps}^{(m)}}\frac{\partial y_z^{(m)}}{\partial a_{zpt}^{(m)}}\delta_{st}\right] \tag{8.16}$$

o, alternativamente,

$$C_{rz}^{(m)} = \begin{cases} \sum_{p=1}^{N_{m-1}} \left[\left(\sum_{k=1}^{N_{m-1}} \frac{\partial y_r^{(m)}}{\partial y_p^{(m-1)}} \frac{\partial y_z^{(m)}}{\partial y_k^{(m-1)}} C_{pk}^{(m-1)} \right) + \sum_{s=1}^{n_{rp}^{(m)}} \left(\frac{\partial y_r^{(m)}}{\partial a_{rps}^{(m)}} \right)^2 \right] & si\ r = z, \\ \sum_{p=1}^{N_{m-1}} \sum_{k=1}^{N_{m-1}} \frac{\partial y_r^{(m)}}{\partial y_p^{(m-1)}} \frac{\partial y_z^{(m)}}{\partial y_k^{(m-1)}} C_{pk}^{(m-1)} & si\ r \neq z. \end{cases} \tag{8.17}$$

Demostración:

$$\begin{aligned} E[\Delta y_i^{(m)} \Delta y_k^{(m)}] &= E\left[\left(\sum_{r=1}^{N_{m-1}} \frac{\partial y_i^{(m)}}{\partial y_r^{(m-1)}} \Delta y_r^{(m-1)} + \sum_{s=1}^{n_{ir}^{(m)}} \frac{\partial y_i^{(m)}}{\partial a_{irs}^{(m)}} \Delta a_{irs}^{(m)} \right) \right. \\ &\quad \left. \left(\sum_{z=1}^{N_{m-1}} \frac{\partial y_k^{(m)}}{\partial y_z^{(m-1)}} \Delta y_z^{(m-1)} + \sum_{t=1}^{n_{kz}^{(m)}} \frac{\partial y_k^{(m)}}{\partial a_{kzt}^{(m)}} \Delta a_{kzt}^{(m)} \right) \right] \\ &= \sum_{r=1}^{N_{m-1}} \sum_{z=1}^{N_{m-1}} \frac{\partial y_i^{(m)}}{\partial y_r^{(m-1)}} \frac{\partial y_k^{(m)}}{\partial y_z^{(m-1)}} E[\Delta y_r^{(m-1)} \Delta y_z^{(m-1)}] \\ &\quad + \frac{\partial y_i^{(m)}}{\partial y_r^{(m-1)}} \sum_{t=1}^{n_{kz}^{(m)}} \frac{\partial y_k^{(m)}}{\partial a_{kzt}^{(m)}} E[\Delta y_r^{(m-1)} \Delta a_{kzt}^{(m)}] \\ &\quad + \sum_{s=1}^{n_{ir}^{(m)}} \frac{\partial y_i^{(m)}}{\partial a_{irs}^{(m)}} \frac{\partial y_k^{(m)}}{\partial y_z^{(m-1)}} E[\Delta a_{irs}^{(m)} \Delta y_z^{(m-1)}] \\ &\quad + \sum_{s=1}^{n_{ir}^{(m)}} \sum_{t=1}^{n_{kz}^{(m)}} \frac{\partial y_i^{(m)}}{\partial a_{irs}^{(m)}} \frac{\partial y_k^{(m)}}{\partial a_{kzt}^{(m)}} E[\Delta a_{irs}^{(m)} \Delta a_{kzt}^{(m)}] \\ &= \sum_{r=1}^{N_{m-1}} \sum_{z=1}^{N_{m-1}} \frac{\partial y_i^{(m)}}{\partial y_r^{(m-1)}} \frac{\partial y_k^{(m)}}{\partial y_z^{(m-1)}} E[\Delta y_r^{(m-1)} \Delta y_z^{(m-1)}] \\ &\quad + \sum_{s=1}^{n_{ir}^{(m)}} \sum_{t=1}^{n_{kz}^{(m)}} \frac{\partial y_i^{(m)}}{\partial a_{irs}^{(m)}} \frac{\partial y_k^{(m)}}{\partial a_{kzt}^{(m)}} E[\Delta a_{irs}^{(m)} \Delta a_{kzt}^{(m)}] \end{aligned}$$

Si se define $C_{rz}^{(m)}$ como $C_{rz}^{(m)} \equiv \dfrac{E[\Delta y_r^{(m)} \Delta y_z^{(m)}]}{\sigma^2}$ entonces:

$$\begin{aligned} E[\Delta y_i^{(m)} \Delta y_k^{(m)}] &= \sum_{r=1}^{N_{m-1}} \sum_{z=1}^{N_{m-1}} \frac{\partial y_i^{(m)}}{\partial y_r^{(m-1)}} \frac{\partial y_k^{(m)}}{\partial y_z^{(m-1)}} \sigma^2 C_{rz}^{(m-1)} \\ &\quad + \sum_{s=1}^{n_{ir}^{(m)}} \sum_{t=1}^{n_{kz}^{(m)}} \frac{\partial y_i^{(m)}}{\partial a_{irs}^{(m)}} \frac{\partial y_k^{(m)}}{\partial a_{kzt}^{(m)}} \sigma^2 \delta_{ik} \delta_{rz} \delta_{st} \\ &= \sigma^2 \sum_{r=1}^{N_{m-1}} \left[\left(\sum_{z=1}^{N_{m-1}} \frac{\partial y_i^{(m)}}{\partial y_r^{(m-1)}} \frac{\partial y_k^{(m)}}{\partial y_z^{(m-1)}} C_{rz}^{(m-1)} \right) \right. \\ &\quad \left. + \delta_{ik} \sum_{s=1}^{n_{ir}^{(m)}} \sum_{t=1}^{n_{kr}^{(m)}} \frac{\partial y_i^{(m)}}{\partial a_{irs}^{(m)}} \frac{\partial y_k^{(m)}}{\partial a_{krt}^{(m)}} \delta_{st} \right] \end{aligned}$$

donde

$$C_{rz}^{(m)} = \sum_{p=1}^{N_{m-1}} \left[\left(\sum_{k=1}^{N_{m-1}} \frac{\partial y_r^{(m)}}{\partial y_p^{(m-1)}} \frac{\partial y_z^{(m)}}{\partial y_k^{(m-1)}} C_{pk}^{(m-1)} \right) + \delta_{rz} \sum_{s=1}^{n_{rp}^{(m)}} \sum_{t=1}^{n_{zp}^{(m)}} \frac{\partial y_r^{(m)}}{\partial a_{rps}^{(m)}} \frac{\partial y_z^{(m)}}{\partial a_{zpt}^{(m)}} \delta_{st} \right] \tag{8.18}$$

Empleando $var(\Delta y_i^{(m)}) = E[(\Delta y_i^{(m)})^2] - (E[\Delta y_i^{(m)}])^2$ y el teorema 6 en la ecuación (8.8) se obtiene:

$$S_i^{(m)} = \frac{\sqrt{E[(\Delta y_i^{(m)})^2]}}{\sigma} \, . \tag{8.19}$$

Sustituyendo $E[(\Delta y_i^{(m)})^2] = \sigma^2 C_{ii}^{(m)}$ en la ecuación (8.19) se obtiene:

$$\begin{aligned} S_i^{(m)} &= \frac{\sqrt{\sigma^2 C_{ii}^{(m)}}}{\sigma} \\ &= \sqrt{\sum_{r=1}^{N_{m-1}} \left[\left(\sum_{z=1}^{N_{m-1}} \frac{\partial y_i^{(m)}}{\partial y_r^{(m-1)}} \frac{\partial y_i^{(m)}}{\partial y_z^{(m-1)}} C_{rz}^{(m-1)} \right) + \delta_{ii} \sum_{s=1}^{n_{ir}^{(m)}} \sum_{t=1}^{n_{ir}^{(m)}} \frac{\partial y_i^{(m)}}{\partial a_{irs}^{(m)}} \frac{\partial y_i^{(m)}}{\partial a_{irt}^{(m)}} \delta_{st} \right]} \\ &= \sqrt{\sum_{r=1}^{N_{m-1}} \left[\left(\sum_{z=1}^{N_{m-1}} \frac{\partial y_i^{(m)}}{\partial y_r^{(m-1)}} \frac{\partial y_i^{(m)}}{\partial y_z^{(m-1)}} C_{rz}^{(m-1)} \right) + \sum_{s=1}^{n_{ir}^{(m)}} \left(\frac{\partial y_i^{(m)}}{\partial a_{irs}^{(m)}} \right)^2 \right]} \end{aligned} \tag{8.20}$$

■

Es sencillo obtener la condición inicial para la ecuación (8.16), $C_{ik}^{(0)} = 0$, empleando el hecho de que las entradas están libres de errores, i.e., $\Delta y_i^{(0)} = 0;\ i = 1, \ldots, N_0$.

8.4 Sensibilidad cuadrática media

En las dos secciones previas se han presentado las expresiones de la sensibilidad estadística para perturbaciones en las entradas y en los parámetros. Sin embargo, la sensibilidad estadística constituye una medida de la estabilidad de la salida de una neurona particular dada una determinada entrada. En esta sección se propone una medida, denominada sensibilidad cuadrática media, que permite evaluar la sensibilidad global del modelo de red funcional propuesto frente a las perturbaciones. Para ello, se empleará el error cuadrático medio como función de coste, ya que es la función empleada habitualmente en el aprendizaje de redes funcionales para medir el error entre la salida real de la red y la deseada y, por tanto, para evaluar el rendimiento del sistema. Formalmente, el ECM se define como

$$ECM = \frac{1}{T}\sum_{t=1}^{T}\sum_{i=1}^{N_M}\epsilon_i^2(t) = \frac{1}{T}\sum_{t=1}^{T}\sum_{i=1}^{N_M}(d_i(t) - y_i^{(M)}(t))^2 \tag{8.21}$$

donde T es el número de patrones de entrada y $d_i(t)$ la salida deseada.

Si las entradas de la red están afectadas por desviaciones, el ECM se verá alterado. Si se considera la serie de Taylor correspondiente a la ecuación (8.21), se obtiene la siguiente expresión:

$$\begin{aligned} ECM' = \; & ECM_0 + \frac{1}{T}\sum_{t=1}^{T}\sum_{i=1}^{N_M}\frac{\partial\epsilon_i^2(t)}{\partial y_i^{(M)}}\Delta y_i^{(M)}(t) \\ & + \frac{1}{2T}\sum_{t=1}^{T}\sum_{i=1}^{N_M}\sum_{j=1}^{N_M}\frac{\partial^2\epsilon_i^2(t)}{\partial y_i^{(M)}\partial y_j^{(M)}}\Delta y_i^{(M)}(t)\Delta y_j^{(M)}(t) + 0 \\ = \; & ECM_0 + \frac{2}{T}\sum_{t=1}^{T}\sum_{i=1}^{N_M}(y_i^{(M)}(t) - d_i(t))\Delta y_i^{(M)}(t) \\ & + \frac{1}{T}\sum_{t=1}^{T}\sum_{i=1}^{N_M}(\Delta y_i^{(M)}(t))^2 \end{aligned} \tag{8.22}$$

donde ECM_0 es el error cuadrático medio obtenido al final del proceso de entrenamiento. Considerando la esperanza de la ecuación (8.22) y teniendo en cuenta que $E[(\Delta y_i^{(M)})^2] - (E[\Delta y_i^{(M)}])^2 = \sigma^2(S_i^{(M)})^2$ (empleando la ecuación (8.8)) y que $E[\Delta y_i^{(M)}] = 0$ (por el teorema 4 o el teorema 6 dependiendo si las perturbaciones son en las entradas o en los parámetros) se obtiene que

$$E[ECM'] = ECM_0 + \frac{\sigma^2}{T}\sum_{t=1}^{T}\sum_{i=1}^{N_M}(S_i^{(M)}(t))^2 \; . \tag{8.23}$$

El segundo término en la parte derecha de la ecuación anterior fue denominado sensibilidad cuadrática media (SCM) por Bernier et al. [14][13], debido a su similitud con la definición del error cuadrático medio. Por lo tanto, se obtiene:

$$E[ECM'] = ECM_0 + \sigma^2 SCM \tag{8.24}$$

donde

$$SCM = \frac{1}{T}\sum_{t=1}^{T}\sum_{i=1}^{N_M}(S_i^{(M)}(t))^2 \ . \tag{8.25}$$

La ecuación (8.24) muestra una relación directa entre la degradación del error cuadrático medio y la sensibilidad cuadrática media. Puesto que el valor de MSE_0 y de la SCM puede evaluarse sobre todo el conjunto de datos después del entrenamiento, es posible predecir el valor de ECM' cuando las entradas o los parámetros están desplazados de sus valores nominales en un rango con desviación típica igual a σ. Además, como se puede observar en la ecuación (8.24), un valor pequeño en la sensibilidad cuadrática media implica una degradación proporcional en el error cuadrático medio. Por tanto, se propone el uso de la SCM como una medida apropiada para medir la tolerancia frente al ruido. Si el ruido se produce sobre las entradas de la red se empleará la ecuación (8.9) para el cálculo de $S_i^{(M)}$, y si por el contrario se produce sobre los parámetros de la red se usará la ecuación (8.15).

8.5 Resultados experimentales

En esta sección se presentan los resultados de algunos experimentos realizados para verificar la validez de la medida desarrollada para la inmunidad al ruido y la tolerancia a fallos. Esta medida fue usada para estimar la degradación del error cuadrático medio, producida por perturbaciones en las entradas o en las salidas, de dos modelos diferentes de redes funcionales: una red de la asociatividad generalizada y una red separable.

8.5.1 Red funcional de la asociatividad generalizada

El primer experimento se realizó utilizando la red funcional de la asociatividad generalizada mostrada en la figura 8.2.

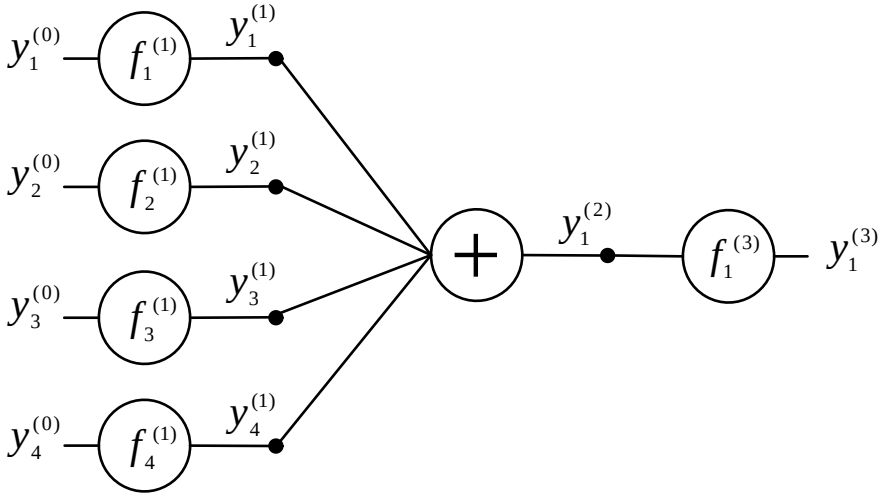


Figura 8.2: Red funcional de la asociatividad generalizada.

Todas las funciones empleadas en la primera y tercera capa $\{f_i^{(1)}, f_1^{(3)};\ i = 1, \ldots, 4\}$ utilizaron como funciones básicas en la ecuación (8.3) la siguiente familia polinómica:

$$\begin{gathered}\{\phi_{ij1}, \phi_{ij2}, \ldots, \phi_{ijn_{ij}^{(m)}}\} = \{1, x, x^2, \ldots, x^{n_{ij}^{(m)}-1}\} \\ m = 1, 2, 3; i = 1, \ldots, N_m; j = 1, \ldots, N_{m-1}\end{gathered} \tag{8.26}$$

En todas las funciones el número de parámetros empleados fue tres, i.e., $n_{ij}^{(m)} = 3;\ m = 1, 3$.

La red fue empleada para predecir la serie caótica de Mackey-Glass y para ello, se usaron 3000 muestras para el entrenamiento ($200 \leq t < 3200$) y 500 para el conjunto de prueba ($5000 \leq t < 5500$). El objetivo de la red era predecir un valor futuro (t+85) a partir de cuatro muestras previas ($t - 18$, $t - 12$, $t - 6$ y t). Una vez finalizado el entrenamiento de la red se obtuvo un error cuadrático medio en el conjunto de prueba de 1.0916×10^{-2}. La tabla 8.1 contiene los valores de los parámetros $a_{ijz}^{(m)}$ (ecuación 8.3) obtenidos para cada una de las funciones neuronales ($f_i^{(m)}$).

Posteriormente, se empleó la medida propuesta (ecuación (8.24)) sobre el conjunto de prueba para estimar el error cuadrático medio cuando se producen perturbaciones en las entradas y en los parámetros de las red. Para comprobar el rendimiento de la medida desarrollada se compararon los resultados obtenidos con los de 100 simulaciones, en cada una de las cuales se añadió un ruido aleatorio que sigue una distribución normal con media cero y desviación típica σ.

Tabla 8.1: Valores de los parámetros $a_{ijz}^{(m)}$ para cada función $f_i^{(m)}$ en la red funcional de la asociatividad generalizada.

$f_1^{(1)}$	$f_2^{(1)}$	$f_3^{(1)}$	$f_4^{(1)}$	$f_1^{(3)}$
0.86241	1.25957	1.18293	1.89757	0.46248
0.00360	−0.08743	0.75037	−0.64879	0.67186
0.13398	−0.17214	−0.93331	−0.24878	−0.13434

Ruido en las entradas

La tabla 8.2 muestra los resultados cuando el ruido se produce sobre las entradas de la red variando el valor de σ entre 0.01 y 0.2. Esta tabla contiene el error cuadrático medio estimado (P) por la ecuación (8.24) y el valor medio obtenido y el intervalo de confianza, para un nivel del 0.95, en las 100 simulaciones (S). La figura 8.3 muestra gráficamente los resultados obtenidos en el intervalo $[0.01, 0.1]$. La curva continua representa el valor obtenido por la medida y los puntos representan el valor medio obtenido en las simulaciones. Como se puede observar la estimación obtenida es bastante buena. La figura 8.4 muestra los resultados sobre un intervalo mayor ($[0.01, 0.2]$). En esta figura se puede observar que cuando el valor de σ aumenta, la estimación no es tan precisa. Esto es una consecuencia de dos hechos:

- La sensibilidad estadística asume valores de σ pequeños (véase ecuación (8.8)).
- La aproximación empleada en la ecuación (8.5) es más exacta cuanto más pequeño sea el valor del incremento en las entradas (o en los parámetros si el ruido afecta a éstos).

En cualquier caso, la medida propuesta proporciona buenos resultados cuando el ruido inducido no es muy grande.

Ruido en los parámetros

La red de la asociatividad generalizada fue empleada también para comprobar el comportamiento de la medida cuando el ruido se produce sobre los parámetros de la red. En este caso los valores de σ utilizados pertenecían al intervalo $[0.001, 0.02]$. Como se puede observar en la tabla 8.1, los valores absolutos de los parámetros de la red están en el intervalo $[0.00360, 1.89757]$, por consiguiente, en el peor de los casos ($\sigma = 0.02$) el parámetro con un valor más pequeño puede sufrir perturbaciones de hasta un 1111%, respecto a su valor nominal, mientras que para el parámetro con un valor más grande, las perturbaciones son de hasta un 2%.

Tabla 8.2: Valor del ECM estimado por la ecuación (8.24) (P) y el ECM medio y el intervalo de confianza obtenido en las 100 simulaciones (S) cuando se producen perturbaciones en las entradas de la red de la asociatividad generalizada.

Desviación típica (σ)	P ($\times 10^{-2}$)	S ($\times 10^{-2}$)
0.01	1.0959	$1.0959 \pm 1.1813e-3$
0.02	1.1087	$1.1086 \pm 2.3459e-3$
0.03	1.1301	$1.1298 \pm 3.4990e-3$
0.04	1.1599	$1.1594 \pm 4.6455e-3$
0.05	1.1984	$1.1973 \pm 5.7912e-3$
0.06	1.2453	$1.2434 \pm 6.9416e-3$
0.07	1.3008	$1.2977 \pm 8.1027e-3$
0.08	1.3648	$1.3601 \pm 9.2805e-3$
0.09	1.4374	$1.4304 \pm 1.0481e-2$
0.10	1.5185	$1.5084 \pm 1.1710e-2$
0.11	1.6082	$1.5941 \pm 1.2975e-2$
0.12	1.7064	$1.6872 \pm 1.4279e-2$
0.13	1.8131	$1.7876 \pm 1.5630e-2$
0.14	1.9284	$1.8951 \pm 1.7033e-2$
0.15	2.0522	$2.0094 \pm 1.8493e-2$
0.16	2.1845	$2.1304 \pm 2.0015e-2$
0.17	2.3254	$2.2578 \pm 2.1603e-2$
0.18	2.4748	$2.3915 \pm 2.3262e-2$
0.19	2.6327	$2.5311 \pm 2.4996e-2$
0.20	2.7992	$2.6765 \pm 2.6808e-2$

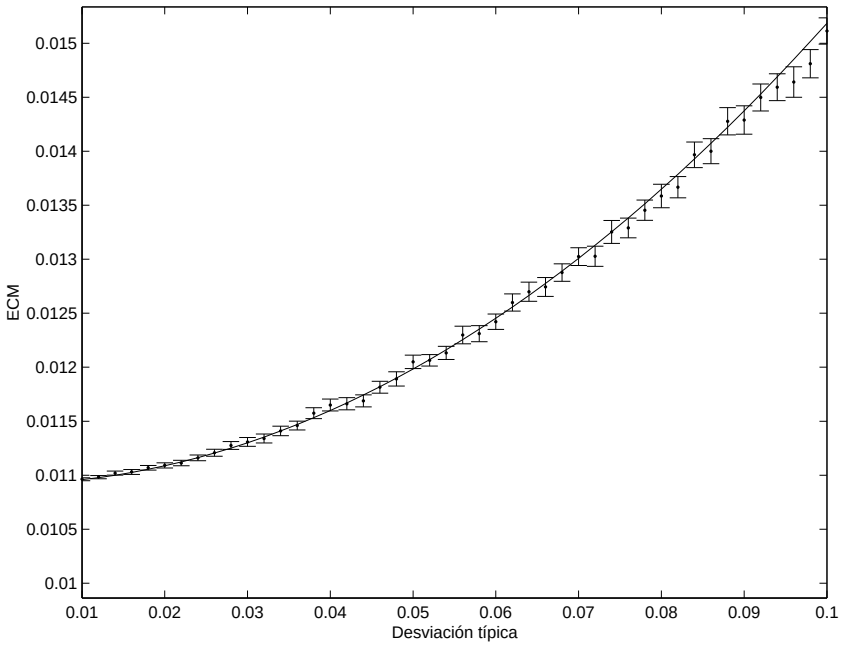


Figura 8.3: ECM estimado (curva continua) y ECM medio obtenido en las simulaciones (puntos) para la red de la asociatividad generalizada empleando valores de σ entre 0.01 y 0.1.

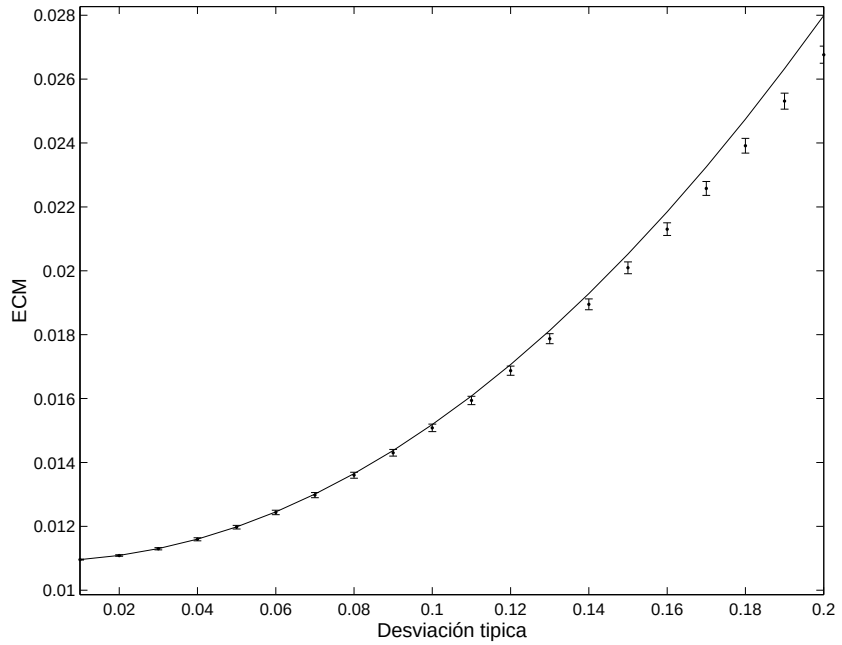


Figura 8.4: ECM estimado (curva continua) y ECM medio obtenido en las simulaciones (puntos) para la red de la asociatividad generalizada empleando valores de σ entre 0.01 y 0.2.

La tabla 8.3 muestra los resultados obtenidos en este caso. Esta tabla contiene el error cuadrático medio (P) estimado por la ecuación (8.24) y el valor medio obtenido y el intervalo de confianza, para un nivel del 0.95, en las 100 simulaciones (S).

Tabla 8.3: Valor del ECM estimado por la ecuación (8.24) (P) y el ECM medio y el intervalo de confianza obtenido en las 100 simulaciones (S) cuando se producen perturbaciones en los parámetros de la red de la asociatividad generalizada.

Desviación típica (σ)	P ($\times 10^{-2}$)	S ($\times 10^{-2}$)
0.001	1.1240	1.1192 ± 0.0083
0.002	1.2210	1.2023 ± 0.0328
0.003	1.3828	1.3411 ± 0.0738
0.004	1.6092	1.5356 ± 0.1312
0.005	1.9004	1.7858 ± 0.2051
0.006	2.2562	2.0920 ± 0.2956
0.007	2.6768	2.4542 ± 0.4028
0.008	3.1620	2.8725 ± 0.5268
0.009	3.7120	3.3469 ± 0.6675
0.010	4.3266	3.8777 ± 0.8251
0.011	5.0059	4.4648 ± 0.9997
0.012	5.7500	5.1085 ± 1.1914
0.013	6.5587	5.8087 ± 1.4002
0.014	7.4321	6.5657 ± 1.6262
0.015	8.3703	7.3796 ± 1.8694
0.016	9.3731	8.2504 ± 2.1301
0.017	10.4407	9.1782 ± 2.4082
0.018	11.5729	10.1633 ± 2.7039
0.019	12.7698	11.2057 ± 3.0173
0.020	14.0315	12.3055 ± 3.3483

La figura 8.5 contiene el error cuadrático medio obtenido por la medida propuesta (curva continua), además de los valores medios obtenidos en las 100 simulaciones (puntos) para cada uno de los valores de σ y el intervalo de confianza para cada uno de ellos. Esta figura muestra que los valores estimados representan una buena aproximación de los valores obtenidos en las simulaciones. Por lo tanto, la medida propuesta proporciona un buen estimador de la degradación del rendimiento, producido por perturbaciones en los parámetros, en la red funcional.

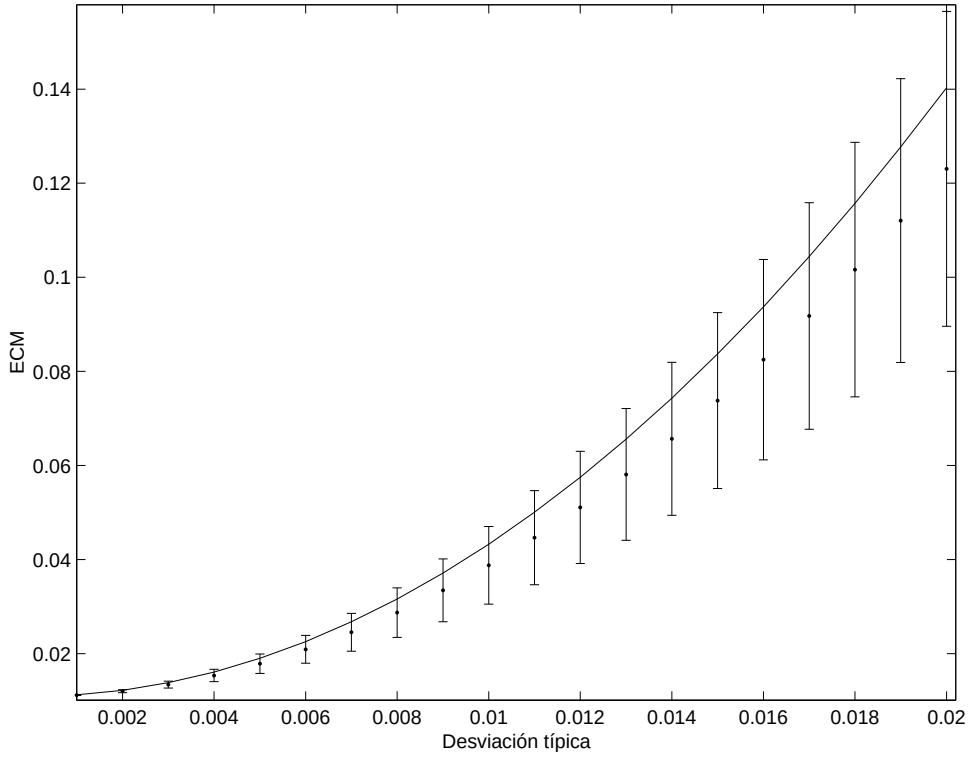


Figura 8.5: ECM estimado (curva continua) y ECM medio obtenido en las simulaciones (puntos) para la red de la asociatividad generalizada cuando se produce ruido en los parámetros.

8.5.2 Red funcional separable

El siguiente experimento se llevó a cabo empleando la red funcional separable mostrada en la figura 8.6.

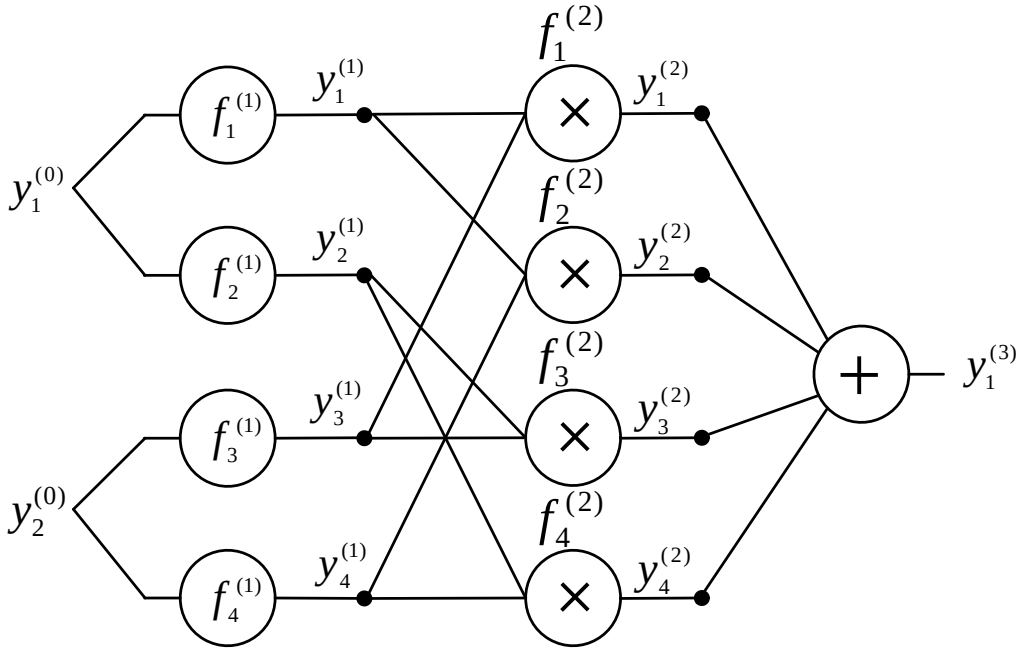


Figura 8.6: Arquitectura de la red funcional separable empleada en las simulaciones.

Las funciones neuronales $f_1^{(1)}$ y $f_3^{(1)}$ fueron definidas empleando como funciones básicas en la ecuación (8.3) la siguiente familia polinómica:

$$\{\phi_{ij1}, \phi_{ij2}, \ldots, \phi_{ijn_{ij}^{(1)}}\} = \{1, x, x^2, \ldots, x^{n_{ij}^{(1)}-1}\} \ , \tag{8.27}$$

mientras que las funciones neuronales $f_2^{(1)}$ and $f_4^{(1)}$ emplearon la siguiente familia de funciones:

$$\{\phi_{ij1}, \phi_{ij2}, \ldots, \phi_{ijn_{ij}^{(1)}}\} = \{sin(x), sin(2x), \ldots, sin(n_{ij}^{(1)}x)\} \ . \tag{8.28}$$

El número de parámetros empleados en cada una de ellas ($n_{ij}^{(1)}$) fue igual a siete. Por último, las funciones de la segunda capa se definieron de la siguiente forma:

$$f_1^{(2)} = a_1^{(2)} y_1^{(1)} y_3^{(1)},$$
$$f_2^{(2)} = a_2^{(2)} y_1^{(1)} y_4^{(1)},$$
$$f_3^{(2)} = a_3^{(2)} y_2^{(1)} y_3^{(1)},$$

$$f_4^{(2)} = a_4^{(2)} y_2^{(1)} y_4^{(1)}.$$

La red funcional descrita se empleó para ajustar la función generalizada de Rosenbrock definida como:

$$f(\mathbf{x}) = \sum_{i=1}^{n-1} 100(x_{i+1} - x_i^2)^2 + (x_i - 1)^2 \tag{8.29}$$

donde n es la dimensión del vector de entrada $\mathbf{x} = (x_1, x_2, \ldots x_n)^T$. En este caso, se empleó en las simulaciones la función de Rosenbrock de dos dimensiones ($n = 2$). Además, los valores de las entradas se limitaron en el intervalo $[-2.048, 2.048]$ y los valores de la función fueron normalizados finalmente en el intervalo $[0, 1]$. Posteriormente, la red fue entrenada empleando 1681 datos generados uniformemente en el espacio de entrada. El error cuadrático medio obtenido en el conjunto de prueba (529 datos) fue 6.8067×10^{-3}. La tabla 8.4 muestra el valor de cada uno de los parámetros ($a_{ijz}^{(m)}$) de la red funcional después del entrenamiento.

Tabla 8.4: Valores de los parámetros $a_{ijz}^{(m)}$ para cada función $f_i^{(m)}$ en la red funcional separable.

$f_1^{(1)}$	$f_2^{(1)}$	$f_3^{(1)}$	$f_4^{(1)}$	$f_1^{(2)}$	$f_2^{(2)}$	$f_3^{(2)}$	$f_4^{(2)}$
0.34030	1.11790	0.47029	1.13764	0.15283	−0.05047	−0.02205	0.00473
0.12981	−0.02222	0.03696	−0.02152	-	-	-	
0.33370	−0.01116	0.06073	−0.03263	-	-	-	
−0.01239	0.00190	−0.03080	0.00041	-	-	-	
0.20734	−0.00951	−0.00044	0.01475	-	-	-	
−0.00112	0.00684	−0.00245	−0.01311	-	-	-	
−0.00024	0.00889	0.00007	0.00599	-	-	-	

Ruido en entradas

En primer lugar, se comprobó la medida propuesta cuando se producen perturbaciones en las entradas de la red. La tabla 8.5 muestra los resultados para este caso cuando se varía el valor de σ entre 0.01 y 0.1. Esta tabla contiene el error cuadrático medio estimado (P) por la ecuación (8.24) y el valor medio obtenido y el intervalo de confianza, para un nivel del 0.95, en las 100 simulaciones (S).

La figura 8.7 ilustra gráficamente los resultados obtenidos en el intervalo $[0.01, 0.1]$. La curva continua representa el valor obtenido por la sensibilidad cuadrática media y los puntos representan el valor medio obtenido en las simulaciones. Como se puede observar la medida proporciona una buena estimación de la degradación del error.

Tabla 8.5: Valor del ECM estimado por la ecuación (8.24) (P) y el ECM medio y el intervalo de confianza obtenido en las 100 simulaciones (S) cuando se producen perturbaciones en las entradas de la red separable.

Desviación típica (σ)	P ($\times 10^{-3}$)	S ($\times 10^{-3}$)
0.010	6.8995	6.8951 ± 0.0062
0.015	6.9156	6.9142 ± 0.0102
0.020	6.9383	6.9305 ± 0.0116
0.025	6.9674	6.9610 ± 0.0167
0.030	7.0030	6.9956 ± 0.0192
0.035	7.0451	7.0447 ± 0.0232
0.040	7.0937	7.0928 ± 0.0264
0.045	7.1487	7.1348 ± 0.0372
0.050	7.2102	7.2173 ± 0.0369
0.055	7.2782	7.3027 ± 0.0454
0.060	7.3526	7.3803 ± 0.0381
0.065	7.4336	7.4472 ± 0.0470
0.070	7.5209	7.5610 ± 0.0512
0.075	7.6148	7.6727 ± 0.0652
0.080	7.7152	7.7603 ± 0.0605
0.085	7.8220	7.8938 ± 0.0801
0.090	7.9353	8.0079 ± 0.0748
0.095	8.0550	8.1284 ± 0.0872
0.100	8.1813	8.2591 ± 0.0927

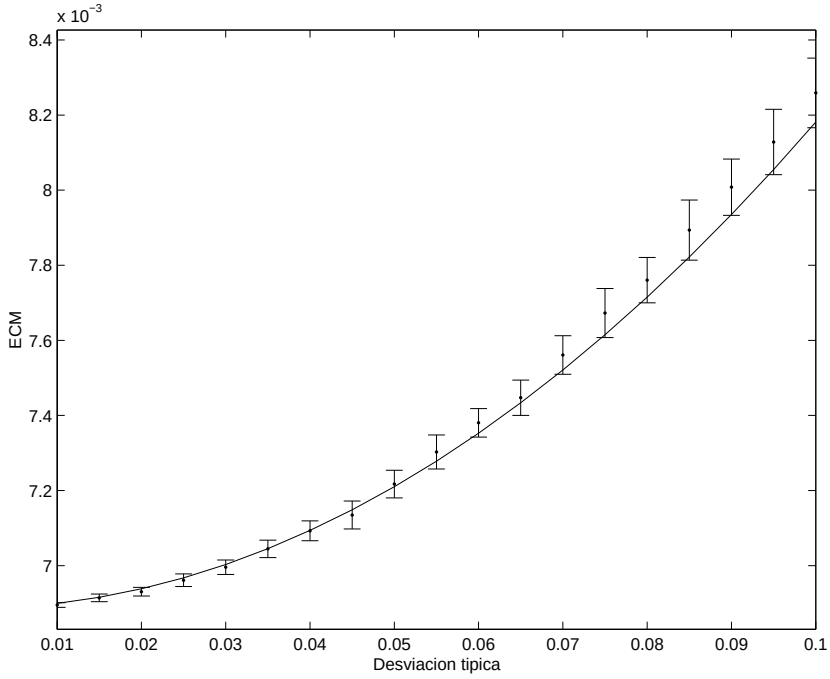


Figura 8.7: ECM estimado (curva continua) y ECM medio obtenido en las simulaciones (puntos) para la red separable cuando se produce ruido en las entradas.

Ruido en parámetros

Asimismo, se empleó la red separable para comprobar la validez de la media propuesta cuando el ruido se produce en los parámetros de esta red funcional. La figura 8.8 (a) muestra gráficamente los valores de la función de Rosenbrock original y (b) la aproximación obtenida por la red separable después del entrenamiento. Las figuras 8.8 (c) y (d) contienen la salida de la red, en una de las simulaciones, cuando a los parámetros se les añadió un ruido aleatorio con desviación típica igual a 0.01 y 0.02, respectivamente. Como se puede apreciar, la distorsión sufrida es bastante apreciable.

Como en los casos anteriores, los resultados de la medida fueron comparados con los obtenidos en 100 simulaciones, en cada una de las cuales se añadió un ruido aleatorio con distribución normal de media cero y desviación típica entre 0.001 y 0.02. En la tabla 8.4 se puede observar que los valores, en valor absoluto, de los parámetros de la red, después del entrenamiento, están en el intervalo $[0.00007, 1.13764]$, por tanto, en el peor de los casos ($\sigma = 0.02$), el parámetro con un valor más pequeño sufre perturbaciones de hasta un 57134%, respecto a su valor nominal, mientras que el parámetro con un valor más grande está afectado por desviaciones de hasta un 3.52%.

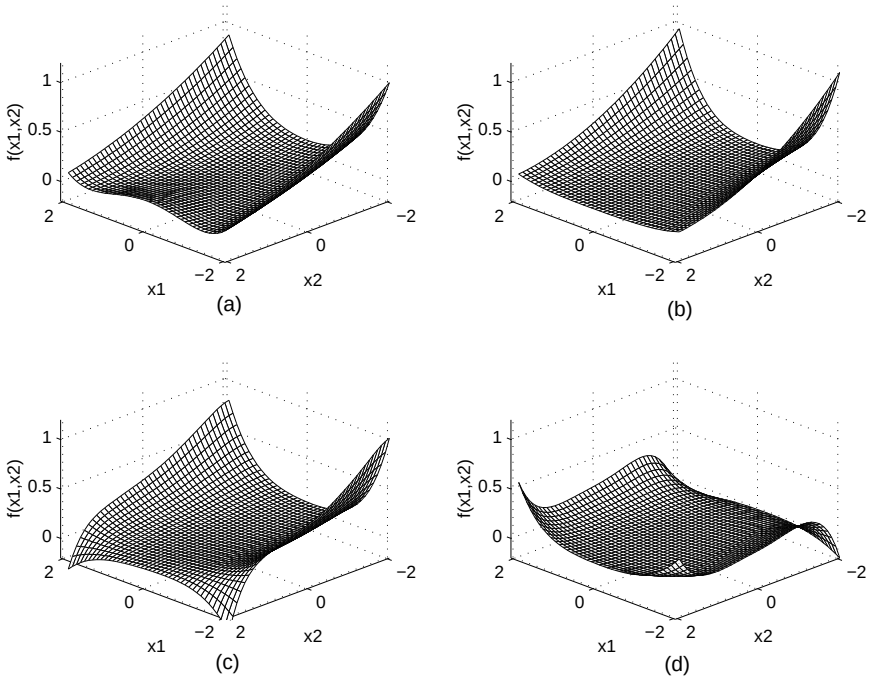


Figura 8.8: (a) Función de Rosenbrock de dos variables, (b) salida de la red después del entrenamiento, (c) y (d) salida de la red cuando los parámetros están afectados por ruido aleatorio con $\sigma = 0.01$ y $\sigma = 0.02$, respectivamente.

La tabla 8.6 muestra los resultados obtenidos en este caso. Esta tabla contiene el error cuadrático medio estimado por la medida (P) y el ECM medio obtenido mediante las 100 simulaciones (S) para cada valor de σ. Además, para cada uno de los valores obtenidos experimentalmente se muestra el intervalo de confianza para un nivel del confianza de 0.95.

Tabla 8.6: Valor del ECM estimado por la ecuación (8.24) (P) y el ECM medio y el intervalo de confianza obtenido en las 100 simulaciones (S) cuando se producen perturbaciones en los parámetros de la red separable.

Desviación típica (σ)	P ($\times 10^{-3}$)	S ($\times 10^{-3}$)
0.001	6.8520	6.8865 ± 0.0775
0.002	6.9877	6.9720 ± 0.1514
0.003	7.2139	7.1882 ± 0.2239
0.004	7.5307	7.5630 ± 0.2887
0.005	7.9379	8.4009 ± 0.5829
0.006	8.4356	8.8374 ± 0.5083
0.007	9.0238	9.5343 ± 1.0715
0.008	9.7025	10.2706 ± 0.9550
0.009	10.4717	11.0637 ± 1.1935
0.010	11.3313	12.1791 ± 1.0733
0.011	12.2815	13.2375 ± 1.3818
0.012	13.3222	15.3153 ± 2.8624
0.013	14.4533	16.1275 ± 1.8913
0.014	15.6750	18.1056 ± 2.6656
0.015	16.9871	19.2553 ± 3.3168
0.016	18.3897	21.0654 ± 3.5127
0.017	19.8828	22.2173 ± 3.8266
0.018	21.4665	24.5083 ± 4.0556
0.019	23.1406	24.6944 ± 4.4101
0.020	24.9052	28.2439 ± 5.4568

La figura 8.9 ilustra gráficamente los resultados obtenidos. La curva continua representa los valores obtenidos por la medida propuesta, mientras que la curva formada por los puntos se corresponde a los valores obtenidos mediante las simulaciones. Además, para cada punto se muestra el intervalo de confianza asociado. De nuevo, ambas curvas son muy similares, lo cual indica que la medida desarrollada proporciona una buena estimación de la degradación del rendimiento de la red funcional en presencia de ruido en los parámetros.

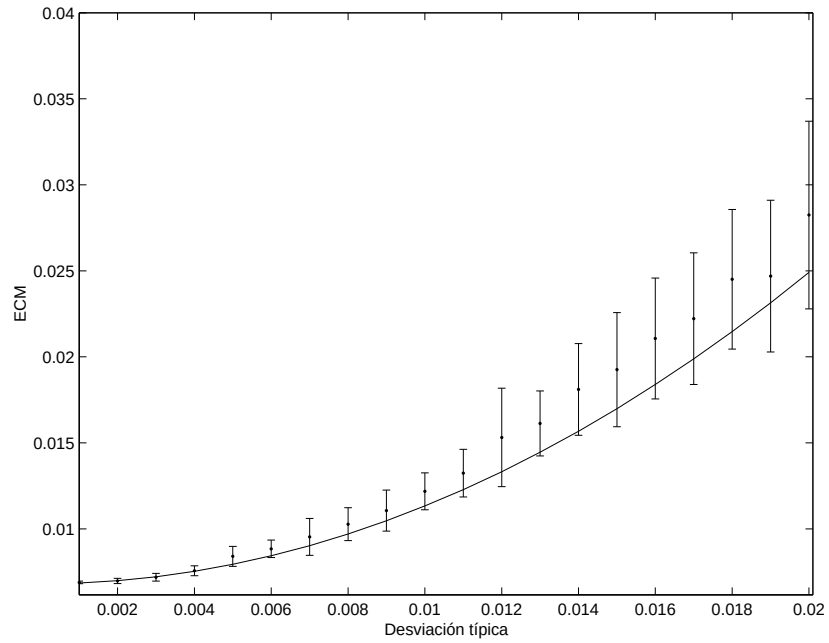


Figura 8.9: ECM estimado (curva continua) y ECM medio obtenido en las simulaciones (puntos) para la red separable cuando se produce ruido en los parámetros.

8.6 Discusión

En este capítulo se ha propuesto una medida, denominada sensibilidad cuadrática media, para estimar la tolerancia a fallos y la inmunidad al ruido en redes funcionales. Esta medida, basada en la sensibilidad estadística, puede ser empleada para:

- *Estimar la degradación del error producido por perturbaciones en las entradas o en los parámetros.* Las perturbaciones en las entradas pueden producirse por diversos motivos, por ejemplo, debido a defectos de fabricación de los componentes o ruido en los sistemas de adquisición (sensores). En cuanto a las desviaciones en los parámetros pueden proceder de defectos en los componentes, interferencias producidas por dispositivos externos, efectos de cuantificación, tolerancias de los componentes, etc. La degradación del error producida por estas perturbaciones depende directamente de la sensibilidad cuadrática media (véase ecuación (8.24)). Si el valor de esta medida es pequeño entonces la degradación del error cuadrático medio también lo será. Por tanto, es posible estimar después del entrenamiento cual será el ECM cuando la red se vea afectada por un ruido con una determinada desviación típica.

- *Seleccionar un modelo de entre varios posibles.* Cuando se realiza el entrenamiento de la red se pueden obtener diversos conjuntos de parámetros que

alcancen un error muy similar y, sin embargo, ser muy diferentes en cuanto a la tolerancia al ruido. Como la medida propuesta cuantifica la existencia de cambios abruptos en la superficie de error en las proximidades del punto actual (determinado por los parámetros), puede emplearse para poder elegir entre todos los conjuntos de parámetros alternativos. Este hecho se produce también cuando se emplean diversos modelos de red. En este caso diversas arquitecturas funcionales pueden lograr un error similar al resolver un problema y, sin embargo, presentar una tolerancia diferente a la aparición de alteraciones en sus parámetros. Por todo ello, la sensibilidad cuadrática media puede ser empleada como criterio de selección entre diversos modelos alternativos.

La medida fue validada empleando dos arquitecturas diferentes de redes funcionales y dos conjuntos de datos. Como se ha demostrado en las simulaciones realizadas, la medida propuesta proporciona una estimación precisa cuando el ruido producido no es muy grande. Cuando el valor de la desviación típica de las perturbaciones es elevado, la estimación no es tan precisa, como consecuencia de que la sensibilidad estadística asume valores de σ pequeños y de que la aproximación del incremento producido en la salida de una neurona (ecuación (8.5)) es más exacta cuanto más pequeño sea el valor del incremento en las entradas o en los parámetros. En cualquier caso, la medida propuesta proporciona buenos resultados cuando la amplitud de las perturbaciones no es muy grande. Por último, es importante destacar que, como se ha mostrado anteriormente, las redes de neuronas con alimentación hacia delante son un caso concreto del modelo de red funcional generalizado mostrado en la figura 8.1. Por consiguiente, la medida propuesta es aplicable también para este tipo de redes.

Capítulo 9

Conclusiones, principales aportaciones y trabajo futuro

"If we knew what it was we were doing,
it would not be called research, would it?"

Albert Einstein

Parte I: Algoritmos de aprendizaje para redes de neuronas artificiales

Algoritmos de aprendizaje para redes de neuronas de una capa:

- Los algoritmos desarrollados garantizan que la solución obtenida en diversas ejecuciones es siempre la misma y se corresponde con el óptimo global de la función de error. Este hecho no está garantizado en los algoritmos actuales.
- Los métodos propuestos se basan en la resolución de un sistema de ecuaciones o de un problema de programación lineal, a diferencia de los algoritmos actuales que emplean un proceso iterativo. Consecuentemente, los métodos desarrollados obtienen la solución en un tiempo mucho menor.
- Otra característica adicional del método propuesto, basado en un sistema de ecuaciones lineales, es su gran flexibilidad para entornos que requieran aprendizaje incremental.
- Los métodos desarrollados se han ampliado para permitir el aprendizaje de las funciones neuronales en vez de asumirlas fijas. En los resultados experimentales

se ha comprobado que para algunos casos esto supone una mejora cualitativa importante y permite incrementar el rendimiento de las redes de neuronas de una capa.

Método de inicialización de los pesos para redes multicapa:

- Requiere un coste computacional muy pequeño comparado con los métodos actuales. Esto es debido a que para estimar los pesos óptimos de cada capa es necesario resolver sólo un sistema de ecuaciones lineales. Este cálculo necesita muchos menos recursos computacionales (memoria y carga del procesador), que el requerido para estimar las derivadas de primer y segundo orden de la función de error.

- Permite obtener una buena solución en muy pocas iteraciones. Este hecho, unido a lo mencionado en el punto anterior, hace que el algoritmo sea muy apropiado para aquellas aplicaciones que requieran una respuesta eficaz (aunque no necesariamente la óptima) y muy rápida, como en sistemas de control o monitorización en tiempo real.

- Se ha comprobado en las simulaciones realizadas que si se entrena el algoritmo de retropropagación del error empleando pesos inicializados aleatoriamente y con el método de inicialización propuesto, en este segundo caso se obtienen soluciones equivalentes en un menor número de iteraciones.

- El algoritmo no converge, necesariamente, hacia un mínimo al final del aprendizaje. Este hecho es una consecuencia de que no es posible establecer algún mecanismo que fuerce a que el error obtenido en la iteración actual sea menor que el obtenido en la anterior. Esto provoca una búsqueda estocástica en el espacio de pesos.

Algoritmo de aprendizaje para redes multicapa:

- El algoritmo de aprendizaje propuesto acelera la convergencia de los métodos actuales. Este hecho se produce a causa de dos motivos. El primero de ellos es que al no estar basado en el gradiente no se ve afectado por la aparición de mesetas y, por tanto, evita los problemas presentados por este tipo de métodos. En segundo lugar, los pesos óptimos para la segunda capa se obtienen en una única iteración y, por lo tanto, se reduce drásticamente el número de pasos requeridos en comparación con los métodos convencionales. Estos hechos fueron confirmados en todos los experimentos realizados, en los cuales el método híbrido consigue siempre la solución en un menor número de iteraciones. Además, es

importante mencionar que el algoritmo desarrollado es un procedimiento rápido de aprendizaje porque está basado en un método analítico que permite obtener la solución óptima empleando un sistema de ecuaciones lineales, el cual no requiere un coste computacional elevado.

- Produce un entrenamiento más homogéneo de la red. En todas las simulaciones realizadas se ha comprobado que el algoritmo desarrollado proporciona un proceso de aprendizaje más uniforme, puesto que los resultados obtenidos y la forma de alcanzarlos no son tan dependientes del estado inicial de la red, es decir, de los pesos iniciales empleados. Esto es una consecuencia de los "saltos" controlados del método, que mejoran el error significativamente en un número muy pequeño de iteraciones.

- Permite evitar algunos mínimos locales, ya que al obtener los pesos de la segunda capa de la red mediante el sistema de ecuaciones lineales es posible desplazarse a otra región de la superficie de error que esté fuera del área de atracción de un mínimo local. Estas suposiciones teóricas fueron confirmadas en las simulaciones realizadas, donde se corroboró que el método propuesto, partiendo de las mismas condiciones iniciales, es capaz de evitar algunos mínimos locales en los cuales quedó atrapado el algoritmo de Levenberg-Marquardt. Consecuentemente, es mucho más probable obtener un error menor al llevar cabo el aprendizaje de la red.

- Finalmente, el algoritmo propuesto no requiere ningún parámetro adicional. La mayor parte de los algoritmos propuestos por otros autores introducen parámetros adicionales en el entrenamiento que son dependientes del problema, ya que deberán ser ajustados para cada caso en concreto. Sin embargo, el algoritmo desarrollado no utiliza ningún parámetro de entrenamiento adicional. Esto supone una ventaja importante pues hace que el proceso de aprendizaje sea más independiente del conjunto de datos empleado, ya que no requiere el ajuste de estos parámetros.

Método de modelado local basado en el SOM y redes de neuronas de una capa:

- Permite un entrenamiento muy rápido del sistema. Esto es una consecuencia directa de la rapidez del SOM para determinar los modelos locales y la gran velocidad del entrenamiento de las redes de una capa empleando el sistema de ecuaciones lineales propuesto. Este proceso sería bastante costoso si se empleasen los métodos tradicionales, debido al gran número de redes a entrenar (una por cada neurona del SOM). Sin embargo, con el aprendizaje basado en el sistema de ecuaciones este aprendizaje se realiza muy rápidamente. Por ello,

el método propuesto es un método muy eficiente y rápido, como ha sido corroborado en los experimentos realizados.

- Consigue un entrenamiento más homogéneo del sistema. En las simulaciones realizadas se ha mostrado que el método de modelado local obtiene un histograma con mucha menos variabilidad que el de un perceptrón multicapa entrenado con el algoritmo de Levenberg-Marquadt. Este hecho es muy beneficioso pues provoca que diversos entrenamientos del sistema neuronal proporcionen resultados muy similares. Esta homogeneidad está basada en el entrenamiento de las redes de una capa (que ajustan los modelos locales) empleando el sistema de ecuaciones lineales.

- Es muy adecuado para entornos que requieran un aprendizaje incremental. El aprendizaje incremental es inherente al algoritmo de Kohonen para la actualización de los pesos del SOM, ya que este tipo de aprendizaje permite actualizar su conocimiento con nuevos datos sin necesidad de utilizar de nuevo todo el conjunto de entrenamiento. Asimismo, el método de aprendizaje para redes de una capa propuesto presenta esta característica. Por tanto, el conocimiento del sistema en su conjunto puede actualizarse de manera sencilla.

- El uso del SOM como método de elección de los modelos locales es muy beneficioso y conveniente debido a las características de proximidad espacial y vecindad de las neuronas que presenta este tipo de red. Esto produce una continuidad entre los distintos modelos locales necesaria para la aproximación local de funciones.

En cuanto al posible trabajo futuro, basado en los métodos propuestos en la primera parte de esta Tesis Doctoral, se proponen las siguientes líneas de investigación:

- Ampliación de los métodos de aprendizaje de redes de una capa para su posible uso con otras funciones de error alternativas.

- Análisis de la posible extensión de los algoritmos de una capa para otras arquitecturas de redes de neuronas como, por ejemplo, las redes recurrentes.

- Análisis de una posible modificación del algoritmo de inicialización de redes multicapa basado en la retropropagación de la salida deseada de manera que sea posible disminuir el error en cada una de las iteraciones.

- Estudio de nuevos algoritmos de aprendizaje para redes multicapa con alimentación hacia delante basados en los métodos y resultados obtenidos en este trabajo.

- Conceptualmente, el modelado local de funciones es una técnica muy atractiva pues sigue la estrategia del divide y vencerás, que se basa en la división de un problema complejo en pequeños subproblemas más sencillos. Los resultados obtenidos con el modelado local muestran que esta es una línea de investigación relevante, por lo que sería de gran interés el desarrollo de nuevos algoritmos de aprendizaje basados en esta aproximación.

Parte II: Medidas de inmunidad al ruido y tolerancia a fallos para redes funcionales y neuronales

- La medida desarrollada permite estimar, después de entrenamiento, la degradación del rendimiento de una red funcional en presencia de ruido en las entradas (inmunidad al ruido) o en los parámetros (tolerancia a fallos). Por tanto, es posible predecir a priori el comportamiento de la red funcional cuando se producen alteraciones en los valores ideales obtenidos durante el entrenamiento.

- La medida propuesta puede emplearse como criterio alternativo para la selección de modelos, ya que permite comparar la tolerancia al ruido de diversos conjuntos de parámetros o diferentes arquitecturas de redes funcionales.

En cuanto al trabajo futuro se propone la siguiente línea de investigación:

- Desarrollo de nuevos algoritmos de entrenamiento y funciones de error que incorporen la sensibilidad cuadrática como un término más a minimizar. De este modo, el entrenamiento de la red puede obtener un error similar, pero ser más tolerante al ruido en los pesos o en las entradas. Además, en las redes de neuronas se ha demostrado que el uso de ruido en las entradas aumenta la capacidad de generalización de estos sistema. Por ello, la medida de inmunidad al ruido desarrollada se podría emplear para aumentar la capacidad de generalización de las redes funcionales incluyéndola como parte del aprendizaje.

Apéndice I: Notación y abreviaturas empleadas

Notación matemática

Notación	Significado	Ejemplos
Letra minúscula normal	Variable escalar	i, j
Letra minúscula negrilla	Vector columna	$\mathbf{x}, \mathbf{y}$
Letra mayúscula negrilla	Matriz	$\mathbf{W}, \mathbf{H}$
Superíndice ^{T}	Traspuesto	$\mathbf{x}^T, \mathbf{W}^T$
Superíndice $^{-1}$	Inversa	$\mathbf{W}^{-1}, f^{-1}(x)$
$E[\cdot]$	Operador esperanza	$E[x]$
$var(\cdot)$	Operador varianza	$var(x)$
∇	Gradiente	∇J
Δ	Incremento	Δy_i
∂	Derivada parcial	$\frac{\partial y_i}{\partial y_r}$
$\|\cdot\|$	Norma euclídea	$\|\mathbf{x}\|$
δ_{ij}	Delta de Kronecker	
σ	Desviación típica	

Abreviaturas

EC	-	Error cuadrático
ECM	-	Error cuadrático medio
ECMN	-	Error cuadrático medio normalizado
FDP	-	Función de densidad de probabilidad
LM	-	Levenberg-Marquardt
RNA	-	Red de neuronas artificiales
RE	-	Retropropagación del error
RSD	-	Retropropagación de la salida deseada
SCM	-	Sensibilidad cuadrática media
SEL	-	Sistema de ecuaciones lineales
SOM	-	Self-Organizing Map

Bibliografía

[1] Aczél J. "Lectures on Functional Equations and their Applications". Academic Press, New York (1966).

[2] Auer P., Hebster M. y Warmuth M. K. Exponentially many local minima for single neurons. En Touretzky D. S., Mozer M. C. y Hasselmo M. E., editores, "Advances in Neural Information Processing Systems", tomo 8, páginas 316–322, Cambridge (1996). The MIT Press.

[3] Barnard E. Optimization for training neural nets. *IEEE Transactions on Neural Networks* **2**(5), 498–508 (1992).

[4] Battiti R. Accelerated back-propagation learning: Two optimization methods. *Complex Systems* **3**(4), 331–342 (1989).

[5] Battiti R. First and second order methods for learning: Between steepest descent and newton's method. *Neural Computation* **4**(2), 141–166 (1992).

[6] Battiti R. y Masulli F. BFGS optimization for faster and automated supervised learning. *Proceedings of the International Neural Network Conference (INNC 90)* páginas 757–760 (1990).

[7] Battiti R. y Tecchiolli G. Learning with first, second and no derivatives: a case study in high energy physics. *Neurocomputing* **6**, 181–206 (1994).

[8] Beale E. M. L. A derivation of conjugate gradients. En Lootsma F. A., editor, "Numerical methods for nonlinear optimization", London (1972). Academic Press.

[9] Bengio S., Bengio Y. y Cloutier J. Use of genetic programming for the search of a new learning rule for neural networks. *Proceedings of the First IEEE World Congress on Computational Intelligence and Evolutionary Computation* páginas 324–327 (1994).

[10] Bennett K. P. y Mangasarian O. L. Robust linear programming discrimination of two linearly inseparable sets. *Optimization Methods and Software, Gordon & Breach Science Publishers* **1**, 23–34 (1992).

[11] Bernier J. L., González J., Cañas A. y Ortega J. Assessing the noise immunity of radial basis function neural networks. En Mira J. y Prieto A., editores, "Lecture Notes in Computer Sciences", tomo 2085, páginas 136–143, New York (2001). Springer-Verlag.

[12] Bernier J. L., Ortega J., Rojas I. y Prieto A. Improving the tolerance of multilayer perceptrons by minimizing the statistical sensitivity to weight deviations. *Neurocomputing* **31**, 87–103 (2000).

[13] Bernier J. L., Ortega J., Ros E., Rojas I. y Prieto A. A new measure of noise immunity and generalization ability for MLPs. *International Journal of Neural Systems* **9**(6), 511–521 (1999).

[14] Bernier J. L., Ortega J., Ros E., Rojas I. y Prieto A. A quantitative study of fault tolerance, noise immunity, and generalization ability of MLPs. *Neural Computation* **12**, 2941–2964 (2000).

[15] Biegler-Konig F. y Barnmann F. A learning algorithm for multilayer neural networks based on linear least squares problems. *Neural Networks* **6**, 127–131 (1993).

[16] Bilbro G. L., Snyder W. E., Gamier S. J. y Gault J. W. Mean field annealing: A formalism for constructing GNC-like algorithms. *IEEE Transactions on Neural Networks* **3**, 131–138 (1992).

[17] Bishop C. M. Exact calculation of the hessian matrix for the multilayer perceptron. *Neural Computation* **4**, 949–501 (1992).

[18] Bishop C. M. "Neural Networks for Pattern Recognition". Oxford University Press, New York (1995).

[19] Bishop C. M. Training with noise is equivalent to tikhonov regularization. *Neural Computation* **7(1)**, 108–116 (1995).

[20] Box G. E. P. y Jenkins G. M. "Time series analysis, forecasting and control". Holden Day, San Francisco (1970).

[21] Brady M., Raghavan R. y Slawny J. Back propagation fails to separate where perceptrons succeed. *IEEE Transactions on Circuits and Systems* **36**, 665–674 (1989).

[22] Budinich M. y Milotti E. Geometrical interpretation of the back-propagation algorithm for the perceptron. *Physica A* **185**, 369–377 (1992).

[23] Buntine W. L. y Weigend A. S. Computing second derivatives in feed-forward networks: A review. *IEEE Transactions on Neural Networks* **5**(3), 480–488 (1993).

[24] Carter M. J., Rudolph F. J. y Nucci A. J. Operational fault-tolerance of cmac networks. En Touretzsky D. S., editor, "Advances in Neural Information Processing Systems", tomo 2, páginas 340–347, San Mateo, CA (1990). Morgan Kaufman.

[25] Castillo E. Functional networks. *Neural Processing Letters* **7(3)**, 151–159 (1998).

[26] Castillo E., Cobo A., Gutierrez J. M. y Pruneda R. E. "Functional Networks with Applications. A Neural-Based Paradigm". Kluwer Academic Publishers, Dordrecht (1998).

[27] Castillo E., Fontenla-Romero O., Alonso-Betanzos A. y Guijarro-Berdiñas B. A global optimum approach for one-layer neural networks. *Neural Computation* **14**(6), 1429–1449 (2002).

[28] Castillo E. y Gutierrez J. M. A comparison of functional networks and neural networks. *Proceedings of the IASTED International Conference on Artificial Intelligence and Soft Computing* páginas 439–442 (1998).

[29] Castillo E., Gutierrez J., Hadi A. y Lacruz B. Some applications of functional networks in statistics and engineering. *Technometrics* **43**, 10–24 (2001).

[30] Castillo E. y Ruiz-Cobo R. "Functional Equations in Science and Engineering". Marcel Dekker (1992).

[31] Castillo E., Fontenla-Romero O., Guijarro-Berdiñas B. y Alonso-Betanzos A. A measure of noise immunity for functional networks. En Mira J. y Prieto A., editores, "Lecture Notes in Computer Science. Connectionist Models of Neurons, Learning Processes, and Artificial Intelligence", tomo 2084, páginas 293–300, Heidelberg, Germany (2001). Springer-Verlag.

[32] Cetin B. C., Burdick J. W. y Barhen J. Global descent replaces gradient descent to avoid local minima problem in learning with artificial neural networks. *Proceedings of the IEEE International Conference on Neural Networks* **2**, 836–842 (1993).

[33] Choi J. Y. y Choi C. Sensitivity analysis of multilayer perceptron with differentiable activation functions. *IEEE Transactions on Neural Networks* **3(1)**, 101–107 (1992).

[34] Coetzee F. M. y Stonick V. L. On uniqueness of weights in single layer perceptrons. *IEEE Transactions on Neural Networks* **7**(2), 318–325 (1996).

[35] Crutchfield J. P. y McNamara B. S. Equations of motion from a data series. *Complex systems* **1**, 417–421 (1987).

[36] Cybenko G. Approximation by superposition of a sigmoidal function. *Mathematics of Control, Signals, Systems* **2**, 303–314 (1989).

[37] Dennis J. E. y Schnabel R. B. "Numerical Methods for Unconstrained Optimization and Nonlinear Equations". Prentice-Hall, Englewood Cliffs, NJ (1983).

[38] Drago G. P. y Ridella S. Statistically controlled activation weight initialization (SCAWI). *IEEE Transactions on Neural Networks* **3**, 899–905 (1992).

[39] Edwards P. J. y Murray A. F. Can deterministic penalty terms model the effects of synaptic weight noise on network fault-tolerance? *International Journal of Neural Systems* **6**(4), 401–416 (1995).

[40] Edwards P. J. y Murray A. F. Modelling weight and input-noise in MLP learning. *Proceedings of the International Conference on Neural Networks* **1**, 78–83 (1996).

[41] Edwards P. J. y Murray A. F. Fault tolorance via weight noise in analog VLSI implementations of MLPs - A case study with EPSILON. *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing* **45**(9), 1255–1262 (1998).

[42] Edwards P. J. y Murray A. F. Towards optimally distributed computation. *Neural Computation* **10**, 997–1015 (1998).

[43] Edwards P. J. y Murray A. F. Weight saliency regularisation in augmented networks. *Proceedings of the European Symposium on Artificial Neural Networks (ESANN)* páginas 261–266 (1998).

[44] Erdogmus D., Genc A. U. y Principe J. C. A neural network perspective to extended luenberger observers. *Institute of Measurement and Control Special Feature on Recent Advances in Neural Networks* **Enero (Parte 2)** (2002).

[45] Erdogmus D. y Principe J. C. Generalized information potential criterion for adaptive system training. *to appear in IEEE Transactions on Neural Networks* (2002).

[46] Fancourt C. y Principe J. C. Optimization in companion search spaces: The case of cross-entropy and the levenberg-marquardt algorithm. *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing (ICASSP)* (2001).

[47] Fisher R. A. The use of multiple measurements in taxonomic problems. *Annual Eugenics* **7 (Parte II)**, 179–188 (1936).

[48] Fletcher R. y Reeves C. M. Function minimization by conjugate gradients. *Computer Journal* **7**, 149–154 (1964).

[49] Fontenla-Romero O., Alonso-Betanzos A., Castillo E., Principe J. C. y Guijarro-Berdiñas B. Local modeling using self-organizing maps and single layer neural networks. En "Lectures Notes in Computer Science", Heidelberg, Germany (2002). Springer-Verlag.

[50] Fukuoka Y., Matsuki H., Minamitani H. y Ishida A. A modified backpropagation method to avoid false local minima. *Neural Networks* **11**(6), 1059–1072 (1998).

[51] Funahashi K. On the approximate realization of continuous mappings by neural networks. *Neural Networks* **2**(3), 183–192 (1989).

[52] Geman S. y Geman D. Stochastic relaxation, gibbs distributions, and the bayesian restoration of images. *IEEE Transactions on Pattern Analysis and Machine Intelligence* **6**, 721–741 (1984).

[53] Gori M. y Tesi A. On the problem of local minima in backpropagation. *IEEE Transactions on Pattern Analysis and Machine Intelligence* **14(1)**, 76–85 (1992).

[54] Grassberger P. y Procaccia I. Characterization of strange attractors. *Physical Review Letters* **50**(5), 346–349 (1983).

[55] Hagan M. T., Demuth H. B. y Beale M. H. "Neural Network Design". PWS Publishing, Boston, MA (1996).

[56] Hagan M. T. y Menhaj M. Training feedforward networks with the marquardt algorithm. *IEEE Transactions on Neural Networks* **5**(6), 989–993 (1994).

[57] Hara K., Nose H. y Ohwada M. A novel line search type algorithm avoidable of small local minima. *Proceedings of the International Joint Conference on Neural Networks* **3**, 2048–2053 (2001).

[58] Haykin S. "Neural Networs: A Comprehensive Foundation". Macmillan, New York (1994).

[59] Haykin S. y Principe J. C. Dynamic modeling with neural networks. *IEEE Signal Processing Magazine* **15**(3), 66 (1988).

[60] Hebb D. O. "The Organization of Behaviour". Wiley, New York (1949).

[61] Hecht-Nielsen R. Theory of the back-propagation neural network. *Proceedings of the International Joint Conference on Neural Networks* **1**, 593–605 (1989).

[62] Hestenes M. R. y Stiefel E. Methods of conjugate gradients for solving linear systems. *Journal of Research of the National Bureau of Standars* **49**(6), 409–436 (1952).

[63] Hinton G. E. y Anderson J. A. "Parallel Models of Associative Memory". Erlbaum, Hillsdale (1981).

[64] Hopfield J. J. Neural networks and physical systems with emergent collective computational abilities. *Proceeedings of the National Academy of Sciences* **79**, 2554–2558 (1982).

[65] Hornik K. Approximation capabilities of multilayer feedforward networks. *Neural Networks* **4**(2), 251–257 (1991).

[66] Hornik K., Stinchcombe M. y White H. Multilayer feedforward networks are universal approximators. *Neural Networks* **2**(5), 359–366 (1989).

[67] Hush D. R. y Horne B. G. Progress in supervised neural networks; what's new since lippmann? *IEEE Signal Processing Magazine* **10**, 8–39 (1993).

[68] Hush D. R., Horne B. G. y Salas J. M. Error surfaces for multilayer. *IEEE Transactions on Systems, Man, and Cybernetics* **22**, 1152–1161 (1992).

[69] Hush D. R. y Salas J. M. Improving the learning rate of back-propagation with the gradient reuse algorithm. *Proceedings of the IEEE Conference of Neural Networks* **1**, 441–447 (1988).

[70] Ihm B. C. y Park D. J. Acceleration of learning speed in neural networks by reducing weight oscillations. *Proceedings of the International Joint Conference on Neural Networks* **3**, 1729–1732 (1999).

[71] Ito Y. Representation of functions by superpositions of a step or sigmoid function and their applications to neural network theory. *Neural Networks* **4**(3), 385–394 (1991).

[72] Jacobs R. A. Increased rates of convergence through learning rate adaptation. *Neural Networks* **1**(4), 295–308 (1988).

[73] Johansson E. M., Dowla F. U. y Goodman D. M. Bacpropagation learning for multilayer feedforward neural networks using the conjugate gradient method. *International Journal of Neural Systems* **2**(4), 291–301 (1992).

[74] Kirkpatrick S., Gelatt C. D. y Vecchi M. P. Optimization by simulated annealing. *Science* **220**, 671–680 (1983).

[75] Kohonen T. Self-organized formation of topologically correct feature maps. *Biological Cybernetics* **43**, 141–152 (1982).

[76] Kohonen T. The self-organizing map. *Proceedings of the IEEE* **78**, 1464–1480 (1990).

[77] Kohonen T. "Self-Organizing Feature Maps". Springer-Verlag, New York (1995).

[78] Kolen J. F. y Pollack J. B. Back propagation is sensitive to initial conditions. En Lippmann R. P., Moody J. E. y Touretzky D. S., editores, "Advances in Neural Information Processing Systems", tomo 3, páginas 860–867, San Mateo, CA (1991). Morgan Kaufmann.

[79] Koskela T., Varsta M., Heikkonen J. y Kaski K. Recurrent SOM with local linear models in time series prediction. *Proceedings of European Symposium on Artificial Neural Networks (ESANN)* páginas 167–172 (1998).

[80] Koskela T., Varsta M., Heikkonen J. y Kaski K. Time series prediction using recurrent SOM with local linear models. *International Journal of Knowledge-Based Intelligent Engineering Systems* **2**(1), 60–68 (1998).

[81] Kostelich E. J. y Yorke J. A. Noise reduction: Finding the simplest dynamical system consistent with the data. *Physica D* **41**, 183–191 (1990).

[82] Kramer A. H. y Sangiovanni-Vincentelli A. Efficient parallel learning algorithms for neural networks. En Touretzky D., editor, "Advances in Neural Information Processing Systems", páginas 40–48, San Mateo, CA (1989). Morgan Kaufmann.

[83] Kreinovich V. Y. Arbitrary nonlinearity is sufficient to represent all functions by neural networks: a theorem. *Neural Networks* **4**(3), 381–383 (1991).

[84] Krzyzak A., Dai W. y Suen C. Y. Classification of large set of handwritten characters using modified back propagation model. *Proceedings of the International Joint Conference on Neural Networks* **3**, 225–232 (1990).

[85] Kumazawa I. A learning scheme of neural networks which improves accuracy and speed of convergence using redundant and diversified network structures. *Proceedings of the IEEE International Joint Conference on Neural Networks* **2**, 1349–1354 (1991).

[86] Lang S. "Linear Algebra". Springer-Verlag, New York, 3 edición (1987).

[87] Lawrence S., Tsoi A. C. y Back A. D. Function approximation with neural networks and local methods: Bias, variance and smoothness. *Australian Conference on Neural Networks* páginas 16–21 (1996).

[88] LeCun Y., Kanter I. y Solla S. A. Second order properties of error surfaces: Learning time and generalization. En Lippmann R., Moody J. y Touretzky D., editores, "Advances in Neural Information Processing Systems", tomo 3, páginas 918–924, San Mateo, CA (1991). Morgan Kaufmann.

[89] Lee B. W. y Sheu B. J. Paralleled hardware annealing for optimal solutions on electronic neural networks. *IEEE Transactions on Neural Networks* **4**, 588–599 (1993).

[90] Lee Y., Oh S. H. y Kim M. W. An analysis of premature saturation in back propagation learning. *Neural Networks* **6**, 719–728 (1993).

[91] Ley E. On the peculiar distribution of the US stock indices first digits. *The American Statistician* **50**(4), 311–314 (1996).

[92] Lorenz E. N. Deterministic nonperiodic flow. *Journal of the Atmospheric Sciences* **20**, 130–141 (1963).

[93] Mackey M. C. y Glass L. Oscillation and chaos is physiological control systems. *Science* **197**, 287–289 (1977).

[94] Marquardt D. An algorithm for least squares estimation of non-linear parameters. *Journal for the Society of Industrial and Applied Mathematics* **11**, 431–441 (1963).

[95] McCulloch W. S. y Pitts W. A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics* **5**, 115–133 (1943).

[96] Minnix J. I. Fault tolerance of the backpropagation neural network trained on noisy inputs. *Proceedings of the International Joint Conference on Neural Networks* **1**, 847–852 (1992).

[97] Minsky M. L. y Papert S. A. "Perceptrons". MIT Press, Cambridge, MA (1969).

[98] Moller M. F. A scaled conjugate gradient algorithm for fast supervised learning. *Neural Networks* **6**, 525–533 (1993).

[99] Moody J. y Darken C. J. Fast learning in networks of locally-tuned processing units. *Neural Computation* **1**(2), 281–294 (1989).

[100] Moshou D. y Ramon H. Extended self-organizing maps with local linear mappings for function approximation and system identification. *Proceedings of the Workshop on Self Organizing Maps* páginas 181–186 (1997).

[101] Neti C., Schneider M. H. y Young E. D. Maximally fault tolerance neural networks. *IEEE Transactions on Neural Networks* **3**(1), 14–23 (1992).

[102] Nguyen D. y Widrow B. Improving the learning speed of 2-layer neural networks by choosing initial values of the adaptive weights. *Proceedings of the International Joint Conference on Neural Networks* **3**, 21–26 (1990).

[103] Packard N. H., Crutchfield J., Farmer J. D. y Shaw R. S. Geometry from a time series. *Physical Review Letters* **45**, 712 (1980).

[104] Parker D. B. Optimal algorithms for adaptive networks: second order back propagation, second order direct propagation, and second order hebbian learning. *Proceedings of the IEEE Conference on Neural Networks* **2**, 593–600 (1987).

[105] Pathak D. y Koren I. Complete and partial fault tolerance of feedforward neural nets. *IEEE Transactions on Neural Networks* **6**(2), 446–456 (1995).

[106] Peterson C. y Anderson J. R. A mean field theory learning algorithm for neural networks. *Complex Systems* **1**, 995–1019 (1987).

[107] Plaut D., Nowlan S. y Hinton G. E. Experiments on learning by back propagation. *Technical Report CMU-CS-86-126, Department of Computer Science, Carnegie Mellon University* (1986).

[108] Poggio T., Girosi F. y Jones M. From regularization to radial, tensor and additive splines. *Neural Networks for Signal Processing, Proceedings of the IEEE Workshop* **3**, 3–10 (1993).

[109] Powell J. D., Fekete N. P. y Chang C. F. Observer-based air-fuel ratio control. *IEEE Control Systems Magazine* **18**(5), 72–83 (1998).

[110] Powell M. J. D. Restart procedures for the conjugate gradient method. *Mathematical Programming* **12**, 241–254 (1977).

[111] Principe J. C. y Kuo J.-M. Dynamic modelling of chaotic time series with neural networks. En Tesauro G., Touretzky D. y Leen T., editores, "Advances in Neural Information Processing Systems", tomo 7, páginas 311–318. The MIT Press (1995).

[112] Principe J. C., Wang L. y Motter M. A. Local dynamic modeling with self-organizing maps and applications to nonlinear system identification and control. *Proceedings of the IEEE* **86**(11), 2240–2258 (1998).

[113] Rezgui A. y Tepedelenlioglu N. The effect of the slope of the activation function on the back propagation algorithm. *Proceedings of the International Joint Conference on Neural Networks* **1**, 707–710 (1990).

[114] Riedmiller M. y Braun H. A direct adaptive method for faster backpropagation learning: The RPROP algorithm. *Proceedings of the IEEE International Conference on Neural Networks* (1993).

[115] Rigler A. K., Irvine J. M. y Vogl T. P. Rescaling of variables in back propagation learning. *Neural Networks* **4**, 225–229 (1991).

[116] Ritter H. Learning with the self-organizing map. En Kohonen T., Makisara K., Simula O. y Kangas J., editores, "Artificial Neural Networks", tomo 1, páginas 379–384, Amsterdam (1991). North Holland.

[117] Rosenblatt F. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review* **65**, 386–408 (1958).

[118] Rumelhart D. E., Hinton G. E. y Willian R. J. Learning representations of back-propagation errors. *Nature* **323**, 533–536 (1986).

[119] Segee B. E. y Carter M. J. Comparative fault tolerance of pararell distributed processing networks. *IEEE Transactions on Computers* **43**(11), 1323–1329 (1994).

[120] Shanno D. F. Conjugate gradient methods with inexact searches. *Mathematics of Operations Research* **3**(3), 244–256 (1978).

[121] Singer A. C., Wornell G. W. y Oppenheim A. V. Codebook prediction: a nonlinear signal modeling paradigm. *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing (ICASSP)* **5**, 325–329 (1992).

[122] Sontag E. y Sussmann H. J. Backpropagation can give rise to spurious local minima even for networks without hidden layers. *Complex Systems* **3**, 91–106 (1989).

[123] Sperduti A. y Antonina S. Speed up learning and network optimization with extended back propagation. *Neural Networks* **6**, 365–383 (1993).

[124] Styblinski M. A. y Tang T. S. Experiments in nonconvex optimization: stochastic approximation with function smoothing and simulated annealing. *Neural Networks* **3**, 467–483 (1990).

[125] Takens F. Detecting strange attractors in turbulence. En Rand D. y Young L., editores, "Dynamical systems and Turbulence (Lecture Notes in Mathematics)", tomo 898, páginas 365–381, New York (1980). Springer-Verlag.

[126] van Ooyen A. y Nienhuis B. Improving the convergence of the back-propagation algorithm. *Neural Networks* **5**, 465–471 (1992).

[127] Vesanto J. Using the SOM and local models in time-series prediction. *Proceedings of the Workshop on Self Organizing Maps* páginas 209–214 (1997).

[128] Vogl T. P., Mangis J. K., Rigler A. K., Zink W. T. y Alkon D. L. Accelerating the convergence of back-propagation method. *Biological Cybernetics* **59**, 257–263 (1988).

[129] Walter J. y Ritter H. Rapid learning with parametrized sel-organizing maps. *Neurocomputing* **12**, 131–153 (1996).

[130] Walter J., Ritter H. y Schulten K. Non-linear prediction with self-organizing maps. *Proceedings of the International Joint Confererence on Neural Networks* **1**, 589–594 (1990).

[131] Wang C. y Principe J. C. Training neural networks with additive noise in the desired signal. *IEEE Transactions on Neural Networks* **10**(6), 1511–1517 (1999).

[132] Watrous R. L. Learning algorithms for connectionist networks: applied gradient methods of nonlinear optimization. *Proceedings of the IEEE Conference on Neural Networks* **2**, 619–627 (1987).

[133] Weigend A. S. y Gershenfeld N. A. "Time Series Prediction: Forecasting the Future and Understanding the Past". Addison-Wesley, Reading, MA (1994).

[134] Weir M. K. A method for self-determination of adaptive learning rates in back propagation. *Neural Networks* **4**, 371–379 (1991).

[135] Weiss S. M. y Kulikowski C. A. "Computer Systems That Learn: Classification and Prediction Methods from Statistics, Neural Nets, Machine Learning and Expert Systems". Morgan Kaufmann, San Mateo, CA (1990).

[136] Wessels L. F. A. y Barnard E. Avoiding false local minima by proper initialization of connections. *IEEE Transactions on Neural Networks* **3**(6), 899–905 (1992).

[137] Widrow B. y Lehr M. A. 30 years of adaptive neural networks: perceptron, madeline and backpropagation. *Proceedings of the IEEE* **78(9)**, 1415–1442 (1990).

[138] Wilamowski B. M., Iplikci S., Kaynak O. y Efe M. O. An algorithm for fast convergence in training neural networks. *Proceedings of the International Joint Conference on Neural Networks* **2**, 1778–1782 (2001).

[139] Xu L., Klasa S. y Yuille A. Recent advances on techniques of static feedforward networks with supervised learning. *International Journal of Neural Systems* **3**, 253–290 (1992).

[140] Yam Y. F. y Chow T. W. S. A new method in determining the initial weights of feedforward neural networks. *Neurocomputing* **16**(1), 23–32 (1997).

[141] Yamada K., Kami H., Tsukumo J. y Temma T. Handwritten numeral recognition by multilayer neural network with improved learning algorithm. *Proceedings of the International Joint Conference on Neural Networks* **2**, 259–266 (1989).

[142] Yao X. Evolving artificial neural networks. *Proceedings of the IEEE* **87**, 1423–1447 (1999).

Índice de Materias