UNIVERSIDAD DE GRANADA

OPTIMIZACIÓN DE REDES NEURONALES DE FUNCIONES BASE RADIALES MEDIANTE ALGORITMOS EVOLUTIVOS

TESIS DOCTORAL

Víctor Manuel Rivas Santos

Granada, 2003

Departamento de Arquitectura y Tecnología de Computadores

UNIVERSIDAD DE GRANADA

OPTIMIZACIÓN DE REDES NEURONALES DE FUNCIONES BASE RADIALES MEDIANTE ALGORITMOS EVOLUTIVOS

Memoria presentada por

VÍCTOR MANUEL RIVAS SANTOS

Para optar al grado de

DOCTOR EN INFORMÁTICA

Fdo. Víctor Manuel Rivas Santos

D. Alberto Prieto Espinosa, Catedrático de Universidad y **D. Juan Julián Merelo Guervós**, Profesor Titular de Universidad, del Departamento de Arquitectura y Tecnología de Computadores de la Universidad de Granada

CERTIFICAN

Que la memoria titulada: *"Optimización de Redes Neuronales de Funciones Base Radiales mediante Algoritmos Evolutivos"* ha sido realizada por **D. Víctor Manuel Rivas Santos** bajo nuestra dirección en el Departamento de Arquitectura y Tecnología de Computadores de la Universidad de Granada para optar al grado de Doctor en Informática.

Granada, a 19 Marzo de 2003

Fdo. Alberto Prieto Espinosa
Director de la Tesis

Fdo. Juan Julián Merelo Guervós
Director de la Tesis

A Olga

Agradecimientos

Deseo expresar mi más profundo agradecimiento a todos los que con su ayuda y ánimo han hecho posible que este trabajo de tesis doctoral haya llegado a buen puerto.

En primer lugar a JJ, porque esta tesis es fiel reflejo de la confianza que siempre ha depositado en mí desde que empezara a trabajar con él en 1992. Gracias por su dirección, tanto en la parte científica, como en la parte estética de este trabajo.

A Alberto Prieto, por aceptar dirigirme en este proyecto, pero también porque desde el día que llegué me ha tratado como uno más dentro de su Departamento y me ha hecho sentirme compañero de todos los que lo forman.

A los miembros del grupo GeNeura, por su ayuda en la parte de programación y en la revisión de esta tesis; pero, sobre todo, por sus continuas muestras de afecto. A los que están (Pedro, Gustavo, Maribel, Fátima) y a los que estuvieron (Javi, Carpio, Brad, Linda...): muchas gracias.

A Héctor, porque sabe de matemáticas bastante más que yo y me descubrió lo útil que puede ser SVD cuando se sabe manejarlo. Y al resto de miembros del Departamento de Arquitectura y Tecnología de Computadores de la Universidad de Granada que, de una u otra forma, han intentado poner su granito de arena en esta memoria.

A mis compañeros del Departamento de Informática de la Universidad de Jaén, por animarme incansablemente a continuar mi trabajo. Gracias especialmente a mis compañeros de despacho, Chequin y José Ramón, así como a los compañeros de "coche", Antonio, Juan, Lidia, Nacho y Pedro. E igualmente a Alfonso, Ángel, Paco, Andrés, Rafa, Luis y María José. Ellos, más que el resto, han sabido escucharme en los días en que el trabajo se hacía más cuesta arriba y nunca han escatimado palabras de apoyo que me permitieran salir adelante.

A mi tío Nicolás (o Fray Eloy, según prefieran), por aportar la música que hizo mucho más amenas las largas horas pasadas delante del ordenador.

Doy gracias también a Dios y a mis padres, porque sin ellos habría sido más que difícil realizar esta tesis doctoral.

Y por supuesto, a Olga. Por escuchar, animar y darme motivos por los que continuar programando, experimentando y escribiendo, y por hacer mi parte del trabajo en casa, que no es poca cosa.

Finalmente, a todos los desarrolladores de la comunidad Linux[1].

[1]Durante la realización de esta tesis ninguna aplicación ha sido "pirateada"; se han utilizado exclusivamente programas de libre distribución, todos ellos bajo sistema operativo Linux (distribuciones SuSE y Mandrake, principalmente).

Tesis Doctoral parcialmente subvencionada por la
Comisión Interministerial de Ciencia y Tecnología
con el proyecto TIC2002-04036-C05-04

CICYT
TIC2002-04036-C05-04

Resumen

En esta tesis se describe un nuevo método, **EvRBF**, basado en un algoritmo evolutivo y diseñado para entrenar redes neuronales de funciones base radiales (**RNFBR**). El método automatiza el establecimiento de los valores para los parámetros de la RNFBR, incluido el tamaño de la misma, intentando acercarlos a sus valores óptimos

Adicionalmente, se introduce el concepto de **Objeto Evolutivo** a partir del cual se ha generado la biblioteca de programación **EO**, con la que se ha programado el nuevo método. Los Objetos Evolutivos han permitido la construcción de sistemas que engloban a todos los paradigmas de la computación evolutiva.

A continuación, se realiza una revisión de los diferentes enfoques basados en algoritmos evolutivos que tratan de diseñar redes neuronales artificiales. De forma más particular, se revisan los métodos propuestos en la literatura para el diseño automático de RNFBR, incluyendo tanto algoritmos evolutivos como no evolutivos.

Posteriormente, se describen los nuevos operadores genéticos diseñados para operar con RNFBR y se realiza un estudio para determinar con qué factor de probabilidad deben ser aplicados. Igualmente, se estudian diversos métodos de inicialización de individuos y asignación de *fitness*, concluyendo con un conjunto de parámetros óptimo para ejecutar el método.

Por último, se comprueba la efectividad del método propuesto aplicándolo a diversos problemas de aproximación funcional, clasificación de patrones y estimación de series temporales. Las tasas de error alcanzadas por el método demuestran su capacidad para determinar la arquitectura de las RNFBR y entrenar los diversos parámetros que las componen.

Abstract

This thesis describes a new method, **EvRBF**, based on an evolutionary algorithm, and designed to train radial basis function neural nets (*Redes Neuronales de Funciones Base Radiales*, **RNFBR**). The method automatically establishes the values for the parameters of the RNFBR, including its size, trying to set them to their best values.

Furthermore, **Evolutionary Objects** are introduced. This new concept lead to the creation of the programming library **EO**, used to program the new method. Evolutionary Objects allowed the generation of systems that implement any kind of evolutionary computation paradigm.

Later, a review of the different approaches, mainly based on evolutionary algorithms, developed to automatically design neural nets is shown. A special attention is paid to the methods proposed in literature related to RNFBR design, including both evolutionary and non-evolutionary algorithms.

After that, the new genetic operators designed to modify RNFBR as well as the research developed to set their probabilities of application are described. Moreover, many different methods for individuals initialization and fitness assignation are studied, and we conclude with the set of parameters that must be used to run the method.

Finally, the method's effectiveness is tested against many different problems related to function approximation, pattern recognition and time-series forecasting. Low generalization error rates yielded by EvRBF show their suitability to assess RNFBR architecture as well as the rest of parameters of which they are composed.

Índice general

Índice de figuras

Índice de Tablas

Acrónimos y símbolos utilizados

Acrónimos

ADDER	Operador de adición de neuronas
AE	Algoritmo Evolutivo
AG	Algoritmo Genético
C_RANDOM	Operador de establecimiento de centros aleatorios
C_TUNER	Operador de afinamiento de centros
COM	*Common Object Model*
CORBA	*Common Object Request Broker Architecture*
DEL_CLOSEST	Operador de eliminación de neuronas cercanas
DEL_MANY	Operador de eliminación de neuronas aleatorias
ECM	Error Cuadrático Medio
ECMN	Error Cuadrático Medio Normalizado
EE	Estrategia Evolutiva
EF_TRN_VAL	Función de evaluación basada en un conjunto de entrenamiento y un conjunto de validación
EF_PARTS	Función de evaluación basada en múltiples particiones del conjunto de entrenamiento
EO	Biblioteca de clases en C++ (*Evolutionary Objects*)
EvRBF	*Evolved Radial Basis Function*
IC_KM	Inicialización de centros basada en K-medias
IC_PAT	Inicialización de centros basada en conjunto de entrenamiento
IC_RND	Inicialización de centros aleatorios
IR_RND_CTE	Inicialización de radios de un solo valor aleatorio
IR_RND_VAR	Inicialización de radios de múltiples valores aleatorios
IR_MIN_DIS	Inicialización de radios basada en mínima distancia
IR_RND_DIS2	Inicialización de radios basada en mínima distancia ponderada
MLP	Perceptrón Multicapa (*Multi-Layer Perceptron*)
NAP	Prototipo de atracción al más cercano (*nearest-attracting prototype*)
OE	Objeto Evolutivo
PE	Programación Evolutiva

PG	Programación Genética
R_RANDOM	Operador de establecimiento de radios aleatorios
R_TUNER	Operador de afinamiento de radios
FBR	Función Base Radial
RECM	Raíz (cuadrada) del Error Cuadrático Medio
RECMN	Raíz (cuadrada) del Error Cuadrático Medio Normalizado
RFS	Selección hacia adelante regularizada (*Regularised Forward Selection*)
RNA	Red Neuronal Artificial
RNFBR	Red Neuronal de Funciones Base Radiales
SVD	Descomposición de valores singulares (*Singular Value Decomposition*)
X_AVERAGE	Operador de recombinación promediada
X_FIX	Operador de recombinación de secuencias de longitud fija
X_MULTI	Operador de recombinación uniforme

Símbolos

$\phi(x)$	Función base radial aplicada al vector de entradas x
c_i	Centro de la neurona oculta i
f_p^i	Salida esperada de la neurona i para el patrón de entrada p
f_p	Salida esperada de la red para el patrón de entrada p
h_i	Salida de la neurona oculta i
max_i	Valor máximo para la variable de entrada i
min_i	Valor mínimo para la variable de entrada i
N_{te}	Tamaño del conjunto de test
N_{tr}	Tamaño del conjunto de entrenamiento
N_{tr}^*	Tamaño del conjunto de entrenamiento menos el de validación
n	Dimensión del espacio de entrada
n'	Dimensión del espacio de salida
P	Conjunto de patrones utilizado para entrenamiento
p	Número de patrones en el conjunto P
p'	Número de neuronas en la capa oculta de una red
p_{c_tuner}	Probabilidad de aplicación interna del operador C_TUNER
p_{c_random}	Probabilidad de aplicación interna del operador C_RANDOM
p_{del_many}	Probabilidad de aplicación interna del operador DEL_MANY
p_{r_tuner}	Probabilidad de aplicación interna del operador R_TUNER
p_{r_random}	Probabilidad de aplicación interna del operador R_RANDOM
p_{x_multi}	Probabilidad de aplicación interna del operador X_MULTI
$p_{x\_average}$	Probabilidad de aplicación interna del operador X_AVERAGE
r_i	Radio de la neurona oculta i
w_{ij}, λ_{ij}	Peso de conexión entre la neurona oculta i y la neurona de salida j
x_p	Vector de entradas correspondiente a patrón número p
y_p^i	Salida de la neurona i para el patrón de entrada p
y_p	Salida de la red para el patrón de entrada p

CAPÍTULO 1 / INTRODUCCIÓN

1.1. Introducción

Esta tesis presenta un nuevo método denominado **EvRBF** (***Evolved Radial Basis Function***) y la prueba de su utilidad en la resolución de problemas sintéticos y reales de aproximación funcional, estimación de series temporales y clasificación de patrones.

EvRBF utiliza un **algoritmo evolutivo** (AE) para encontrar, de forma automática, una **Red Neuronal de Funciones Base Radiales** (RNFBR) que resuelva un problema concreto, teniendo como principal objetivo la minimización del error cometido por la red al realizar dicha tarea.

El principal problema que se plantea al construir una RNFBR radica en determinar la estructura más adecuada para la misma, esto es, determinar parámetros tales como el número de neuronas que tendrá en su capa oculta y cómo se configurarán las distintas funciones base radiales (FBR) que operan en cada una de ellas. Posteriormente, una vez se haya determinado la estructura de una RNFBR, bastará con resolver un sistema de ecuaciones para obtener los valores óptimos de los pesos de las conexiones sinápticas. Esta es quizá la mayor diferencia que existe entre las RNFBR y otros modelos de redes neuronales, en los que el mayor problema se centra en hallar un algoritmo que permita estimar los pesos de las conexiones sinápticas. La tarea de diseñar una RNFBR se convierte así en un problema de optimización muy apropiado para ser abordado mediante técnicas de computación evolutiva.

El método EvRBF combina las ventajas que proporciona, por un lado, el algoritmo de búsqueda global que lleva a cabo el AE y, por el otro, la

rapidez con que puede ser entrenada una RNFBR una vez se ha determinado su estructura. Así, el AE tendrá como tarea dotar a tales parámetros de valores pertenecientes a un entorno de los óptimos que se pretenden encontrar, mientras que el algoritmo de entrenamiento de la red podrá ser usado para refinar los enlaces existentes entre las distintas capas estableciendo para ellos pesos óptimos.

La implementación del nuevo método EvRBF se ha llevado a cabo siguiendo la filosofía *Algoritmos + Estructuras de datos = Objetos Evolutivos*, inspirada inicialmente en el trabajo de Michalewicz [Mich99] y plasmada en forma de biblioteca de clases escrita en C++ denominada **EO**. El diseño de esta biblioteca permite la rápida implementación de la mayoría de los paradigmas existentes dentro de la computación evolutiva y la posibilidad de derivar nuevos paradigmas a partir de ellos. La idea subyacente bajo esta filosofía consiste en permitir que los AE operen, si se desea, directamente sobre el espacio de soluciones en vez de sobre una codificación de este espacio, que hasta ahora era lo más habitual. Se posibilita así la creación de operadores evolutivos específicos que aprovechen la información que se tenga de ese espacio de soluciones intentando resolver cada problema de forma óptima.

La efectividad del método EvRBF ha sido probada aplicándolo a diversos problemas, tanto sintéticos como reales. Los resultados obtenidos han sido comparados con los que ofrecen otras técnicas de estimación de series temporales, clasificación y aproximación funcional disponibles en la bibliografía. Como principal conclusión cabe destacar que EvRBF arroja buenos resultados en la mayoría de los experimentos llevados a cabo, con la ventaja de que no es necesario prefijar ninguno de los parámetros relativos a la RNFBR que finalmente será adoptada como la solución del problema.

1.2. Estructura de la tesis

En este primer capítulo de la tesis se exponen cuáles son los objetivos que se persiguen y cuál es la estrategia que se ha seguido para conseguirlos. A continuación se detallan las fases de desarrollo por las que han pasado los tres temas principales sobre los que versa la tesis. En primer lugar se describen de forma genérica las redes neuronales artificiales, indicando sus propiedades de entra las que destaca la capacidad de aprendizaje. A continuación se revisa el campo de la computación evolutiva y se enmarcan los algoritmos evolutivos dentro del contexto de los problemas de búsqueda y optimización. Finalmente, se describe la biblioteca de clases

EO, utilizada para implementar EvRBF. Se pretende con este capítulo establecer una base sobre la que construir el resto de capítulos que componen la tesis.

Tras el presente capítulo de introducción general de la tesis, el capítulo 2 describe de forma detallada las RNFBR, su topología y los distintos algoritmos de entrenamiento que existen. Adicionalmente, se realiza una revisión del estado del arte relativo al diseño de RNA y, más explícitamente RNFBR, mediante AE.

Una vez presentados los aspectos conceptuales de la tesis, se dedica el capítulo 3 a describir en profundidad el método EvRBF. Dado que este método es en esencia un AE, dicho capítulo describe tanto su estructura general como el funcionamiento detallado de los operadores que se han diseñado, las características propias de los individuos y la población que conforman.

A continuación, en el capítulo 4 se realiza un estudio sistemático que pretende establecer el conjunto de parámetros con el que se ha de utilizar EvRBF. Tales parámetros incluyen elección de la función de evaluación y probabilidades de aplicación de operadores, entre otros.

En el capítulo 5 se aplica el método a un conjunto de problemas de estimación de series temporales, clasificación de patrones y aproximación funcional. Tales problemas son resueltos por EvRBF utilizando siempre el mismo conjunto de parámetros que se ha establecido en el anterior capítulo.

La memoria termina con el capítulo 6, en el que se especifican las conclusiones de las tesis y líneas de investigación y desarrollo que se sugieren para el futuro.

1.3. Redes Neuronales Artificiales

Las Redes Neuronales Artificiales (RNA) constituyen un paradigma de computación útil para la resolución de problemas complejos: reconocimiento del habla, reconocimiento de imágenes, planificación de movimientos (obtención de la trayectoria para aparcar un vehículo), percepción compleja (reconocer a una persona dentro de una multitud de un simple vistazo) y en general otros tipos de tareas relacionados con el tratamiento de información que los humanos realizamos de forma rutinaria constantemente.

Una RNA constituye un modelo específico dentro del conjunto general de los modelos estadísticos. Así, parte de la terminología utilizada actualmente en el campo de las redes se corresponde en cierta medida con con-

Estadística	*Redes Neuronales*
Modelo	Red
Estimación	Aprendizaje
Regresión	Aprendizaje supervisado
Interpolación	Generalización
Observaciones	Conjunto de entrenamiento
Parámetros	Pesos sinápticos
Variables independientes	Entradas
Variables dependientes	Salidas

Tabla 1.1: Correspondencia de términos usados en estadística con términos usados en el campo de las redes neuronales artificiales

ceptos desarrollados anteriormente para la estadística, como puede verse en la tabla 1.1

El comienzo de la investigación en RNA tuvo lugar en los años 40, cuando McCulloch y Pitts [W43] describieron por primera vez la denominada **neurona formal**. Desde entonces la investigación en redes neuronales ha tenido una gran expansión, especialmente en el campo del reconocimiento de patrones en el que las RNA son frecuentemente aplicadas.

1.3.1. Estructura de una RNA

Externamente, una RNA se comporta como una función a la que se aplican una serie de valores de entrada y proporciona un conjunto de valores de salida. Internamente, una RNA representa el conocimiento utilizando una estructura formada por muchas unidades de procesamiento simples (neuronas o nodos) conectadas entre sí; por ello, también se las suele denominar **sistemas conexionistas**. Tanto las unidades como los enlaces que las relacionan toman exclusivamente valores numéricos. Se puede idealizar así el modelo conexionista considerando que es un conjunto de elementos de cómputo densamente interconectados. El modelo se puede definir [Hern97] como un grafo ponderado, G, determinado por un conjunto de m nodos o unidades de procesamiento, U, y una topología, Λ , dada por el conjunto de aristas o conexiones entre las unidades:

$$G = \{U, \Lambda\} \tag{1.1}$$

Las unidades del modelo, U, pueden ser descritas por un conjunto de variables continuas y/o discretas denominado vector de salidas del modelo:

$$\vec{y} = \{y_1, y_2, \ldots, y_i, \ldots, y_m\} \quad ; \quad \forall y_i \in \mathbb{R} \tag{1.2}$$

siendo m el número de unidades de procesamiento e y_i la salida de la unidad i. El espacio de salidas $\mathbb{R}^m$ del modelo es el producto cartesiano de todos los espacios de salidas de las variables del vector $\vec{y}$.

La topología de un modelo conexionista viene determinada por el conjunto de conexiones:

$$\Lambda = \{\lambda | \lambda = \{i, j\} \quad i, j \in U\} \tag{1.3}$$

Dependiendo del modelo concreto, las aristas del grafo pueden ser dirigidas o no, dando lugar a conexiones asimétricas o simétricas respectivamente.

El peso de una conexión es generalmente un número real que puede representarse formalmente por:

$$w_{ij} \in \mathbb{R}; \lambda = \{i, j\} \in \Lambda \quad ; \quad i, j \in U \tag{1.4}$$

De forma habitual, las neuronas o unidades de cómputo se disponen formando capas o niveles. Esta disposición topológica en forma de capas puede ser representada mediante una partición del conjunto de unidades en subconjuntos U_c de forma que:

$$U = \bigcup_{c=1}^{n} U_c \quad \textit{siendo} \quad U_c \bigcap U_z = \emptyset \quad ; \quad c < z \quad c, z = 1, 2, \ldots, n \tag{1.5}$$

donde n es el número de capas o niveles del modelo. En la mayoría de las redes que poseen esta estructura topológica definida, las unidades de un determinado nivel sólo tienen conexiones con las unidades del nivel anterior y posterior. En función del problema y de la relación de la red con el medio, se pueden distinguir capas ocultas y capas visibles, siendo en estas últimas en las que suelen agruparse las neuronas de entrada y las neuronas de salida. Así pues, las unidades visibles se corresponden con los niveles U_1 y U_n, denominados nivel de entrada (en el símil biológico correspondería a las neuronas sensoriales o **aferentes**) y nivel de salida (neuronas motoras o **eferentes**), respectivamente. Los niveles intermedios son niveles ocultos. La fig. 1.1 muestra algunas de las topologías más comúnmente utilizadas dentro del campo de las RNA.

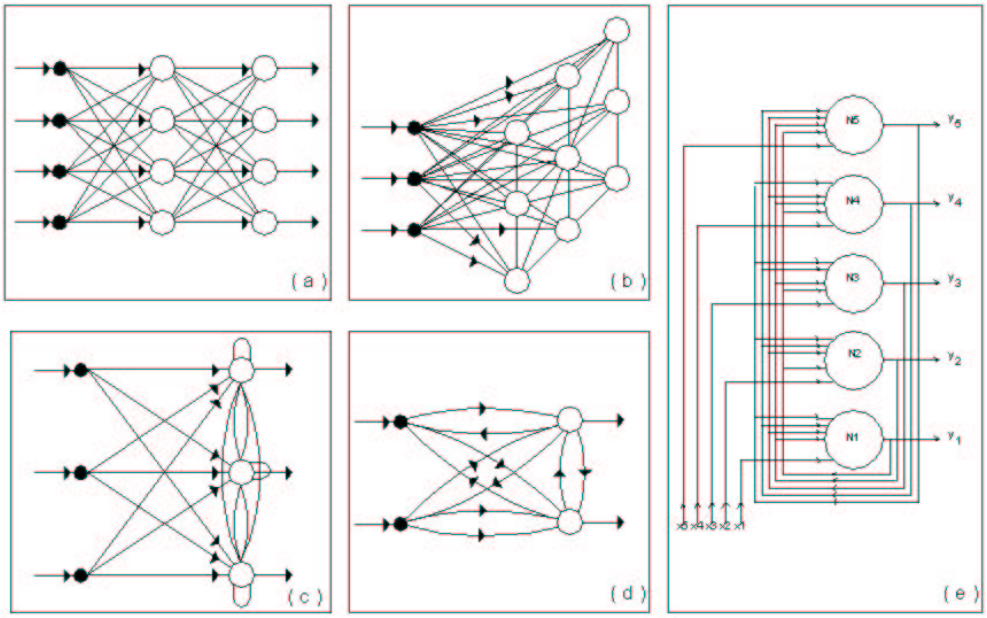


Figura 1.1: Ejemplos de topologías básicas de modelos de RNA. (a) Perceptrón multicapa (MLP); (b) Mapa auto-organizativo de Kohonen; (c) Red Competitiva; (d) Modelo ART1; (e) Red de Hopfield.

La dinámica particular con que actúa cada neurona viene dada por una función denominada **función de activación** que describe el cambio de la salida de la unidad. La dinámica global del modelo aparecerá como resultado de la acumulación de las dinámicas particulares de cada neurona. Es decir, la aplicación masiva de las reglas de activación describe cómo se mueve el vector de estado $\mathbb{R}$ del modelo. La función umbral, la función rampa, la función sigmoidal, o la función de base radial (fig. 1.2) son ejemplos típicos de funciones utilizadas como reglas de activación.

Las RNA se dividen, según su conectividad, en redes **hacia adelante** (*feed-forward*) y **recurrentes**. Será hacia adelante si es posible numerar los nodos de forma ascendente desde las entradas hacia las salidas sin que haya ninguna conexión de un nodo a otro con un número menor. La RNA será recurrente si no existe ninguna numeración posible de ese tipo.

1.3.2. Aprendizaje en RNA

Sin duda, una de las principales peculiaridades de las redes neuronales es que tienen capacidad para el **aprendizaje** (o auto-programación). En el contexto de RNA el aprendizaje es el proceso de actualizar la estructura y

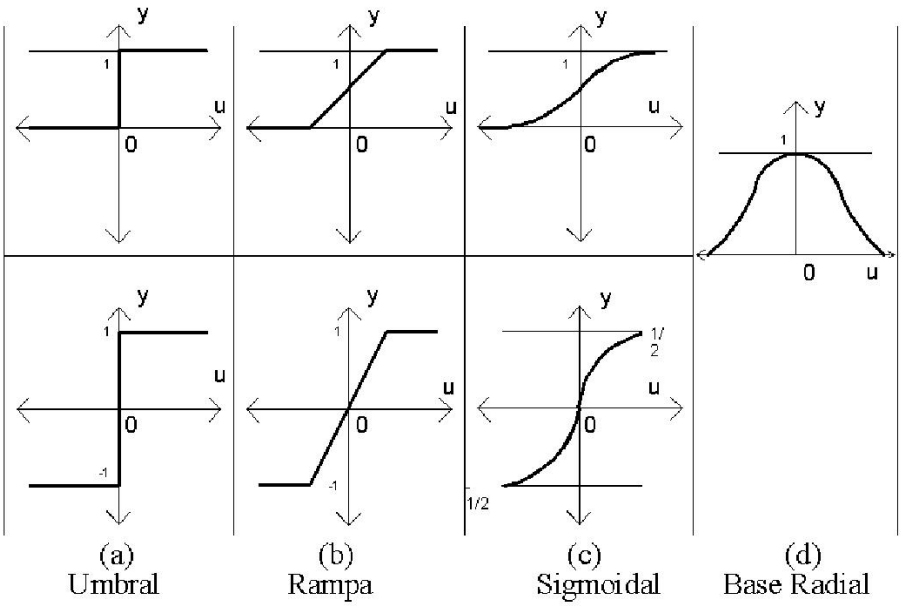


Figura 1.2: Funciones de activación usualmente utilizadas en los modelos neuronales.

pesos de las conexiones de la red con objeto de que realice eficientemente una tarea específica. El aprendizaje se realiza a partir de ejemplos y pretende o bien formar asociaciones entre representaciones que se especifican externamente, o bien construir representaciones internas del entorno.

Se ha ideado una gran cantidad de algoritmos para entrenar los pesos a través de la presentación de ejemplos para una topología fija. Sin embargo, tales algoritmos tienen el problema de que suelen caer en óptimos locales, por lo que la obtención de buenos resultados depende en gran medida de los parámetros de aprendizaje y de los pesos iniciales, así como de la topología de la red. De hecho, tanto las RNA como otros métodos estadísticos clásicos para estimación de series temporales, reconocimiento de patrones o aproximación funcional adolecen del problema de tener gran cantidad de parámetros libres, que deben ajustarse a mano o mediante ensayo y error, intentando obtener buenos resultados.

El objetivo fundamental del algoritmo de aprendizaje es producir redes que **generalicen** (o abstraigan) correctamente casos de un determinado dominio, adquiriendo esa facultad a partir de un aprendizaje previo con un conjunto de casos típicos de entrenamiento dentro de ese dominio. En definitiva, los pesos de las conexiones se modifican de forma que los elementos de cómputo asocien patrones o representen características

emergentes del dominio en cuestión. No obstante, el proceso de entrenamiento de RNA puede dar lugar a otro de los problemas más comunes: el **sobre-entrenamiento** (*overfitting*) [Bish95b]. Una red sobre-entrenada es capaz de predecir muy bien las salidas correspondientes a las entradas que se han utilizado para entrenarla, pero obtendrá pobres resultados al intentar predecir las salidas correspondientes a nuevas entradas, es decir, entradas que no ha *visto* durante el proceso de aprendizaje.

El aprendizaje o **entrenamiento** de la RNA se realiza a partir de un conjunto de casos (ejemplos o muestras) representativos, P. Para cada uno de los elementos de P se efectúa una modificación de los pesos encaminada a conseguir una mejora en las prestaciones de la red. La actualización de la RNA se lleva a cabo mediante reglas de aprendizaje que monitorizan el proceso, existiendo un algoritmo de aprendizaje que determina la forma en que se aplican las reglas para ajustar los pesos.

Existen tres estrategias susceptibles de ser utilizadas para entrenar una RNA:

a) **Aprendizaje supervisado**. Durante la fase de aprendizaje se suministra a la red, junto a los patrones de entrenamiento, la salida esperada para cada elemento de P, o bien una indicación del grado de acuerdo o error de la salida (esto es, un "refuerzo") obtenida con la entrada de entrenamiento (**aprendizaje con refuerzo**).

b) **Aprendizaje no supervisado**. No es necesario presentar a la red la salida asociada a cada entrada del conjunto de aprendizaje (P). La red construye modelos internos a través de la detección de regularidades o correlaciones en los vectores de aprendizaje, que clasifica en grupos disjuntos, sin necesidad de recibir ninguna información adicional [Prie90].

c) **Aprendizaje híbrido**. Supone una combinación de las otras dos estrategias; así en ocasiones, partiendo de un proceso no supervisado para el aprendizaje de la red, sus prestaciones pueden mejorarse mediante una fase adicional de entrenamiento ("ajuste fino") que puede catalogarse de supervisado.

Existen cuatro tipos básicos de reglas de aprendizaje: corrección de error, Bolzmann, Hebbiano y competitivo. A continuación se describen brevemente.

1.3.2.1. Regla de corrección de error

En el aprendizaje supervisado se requiere suministrar a la red parejas de entrenamiento compuestas por un patrón de entrada y la salida que se desea que obtenga de la red. Se suele definir una **función de coste o error** para evaluar la discrepancia de las salidas reales obtenidas, y_p, con las salidas deseadas, f_p, donde p hace referencia a un patrón genérico de entrenamiento. La red ajusta los pesos de acuerdo con un determinado algoritmo y en base a la función de error.

El método LMS (de los mínimos cuadrados) utiliza la siguiente función de error:

$$e = \frac{1}{2} \sum_{q \in U^n} (y_p^q - f_p^q)^2 \quad ; \quad p \in P \tag{1.6}$$

donde U_n es el conjunto de elementos de cómputo que producen salidas externas, P es el conjunto de patrones de aprendizaje, y_p^q es la salida real de la unidad q cuando se aplica el patrón de aprendizaje p, y f_p^q es la correspondiente salida que deseamos obtener.

El método de aprendizaje mediante ajuste de mínimos cuadrados permite minimizar el error e definido en 1.6. Para ello se inicializan los pesos con valores aleatorios y se van cambiando iterativamente en cantidades Δw_{ij} de acuerdo con un determinado algoritmo.

En una red sin realimentación con una capa de elementos de cómputo y salidas de valores continuos, puede utilizarse el algoritmo LMS de aprendizaje del perceptrón ideado por Widrow y Hoff [Widr60]. Este algoritmo indica que el peso de la conexión entre la entrada j y la unidad de salida i, al aplicar la pareja de entrenamiento (y_p, f_p), debe modificarse según:

$$\Delta w_{ij} = \alpha (f_p^i - y_p^i) y_p^j \tag{1.7}$$

donde α es la **tasa de aprendizaje**, una constante usualmente comprendida entre 0 y 1.

El proceso de aprendizaje definido en 1.7 es frecuentemente usado con el tipo de RNA denominado **perceptrón**. Cuando esta RNA dispone de una sola capa de neuronas, el conjunto de patrones que pueden ser clasificados se restringe a aquellos que geométricamente equivalen a regiones de un espacio vectorial separadas por planos. Para solucionar este problema se puede añadir una capa oculta de elementos de procesamiento, entre las capas de entrada y salida, dado lugar al perceptrón multicapa,

MLP. Este tipo de red requiere un nuevo procedimiento de aprendizaje que consiste en, una vez que se ha obtenido la salida errónea, actualizar los pesos comenzando en la capa de salida y yendo hacia la de entrada, por capas sucesivas (método de **retro-propagación**, desarrollado independientemente por Werbos [Werb74], Parker [Park85] y Rumelhart y Zipser [Rume85]).

1.3.2.2. Aprendizaje de Boltzmann

Las máquinas de Bolzmann [Hint84] (que pueden considerarse una generalización de la Red de Hopfield) son redes recurrentes simétricas cuyos elementos de cómputo son unidades binarias (salidas +1 y -1 ó 1 y 0). De entre dichos elementos de cómputo, un conjunto interactúa con el exterior (neuronas visibles), mientras que el resto opera de manera interna (neuronas ocultas). Actúa en dos modos: **captación**, en el que se captan las entradas a través de las neuronas visibles y **funcionamiento libre**, en el que la red actúa libremente hasta alcanzar un estado de equilibrio.

El aprendizaje de Bolzmann es una regla estocástica fundamentada en principios de la termodinámica y de la teoría de la información. Para el aprendizaje se define una **función consenso**:

$$C = \sum_{i,j} w_{ij} h_j h_i \tag{1.8}$$

donde h_j es el estado (0 ó 1) de la unidad j, w_{ij} el peso de la conexión entre las unidades j e i. El objetivo del aprendizaje de Bolzmann es ajustar los pesos de interconexión de forma que la probabilidad de que la unidad i cambie de estado siga la distribución de probabilidad expresada en 1.9:

$$p_i = \frac{1}{1 + e^{-\frac{\Delta C_i}{T}}} \tag{1.9}$$

donde ΔC_i es la variación en el valor de la función de consenso de la red originada por el cambio de estado en la unidad i y T es un parámetro denominado temperatura, debido a la similitud de la expresión (1.9) con la función de distribución de Bolzmann. A temperatura elevada la transición a valores de consenso menores son más probables, permitiéndose una exploración suficiente de los distintos estados de la red. Estas transiciones son menos probables a medida que la temperatura decrece. Si la disminución de la temperatura se hace con suficiente lentitud, se alcanzará el

estado para el que el valor del consenso (respuesta de la red) es el máximo absoluto. Es un proceso análogo al del enfriamiento lento de un sólido (**enfriamiento simulado**, *simulated annealing*), que alcanzará su estado de mínima energía.

1.3.2.3. Regla Hebbiana

La regla de aprendizaje más antigua es la de **aprendizaje Hebbiano no supervisado**. Hebb se basó en que, según experimentos neurobiológicos, si las neuronas interconectadas por una sinapsis están simultáneamente y repetitivamente activas, su peso es selectivamente incrementado.

El algoritmo que propuso Hebb [Hebb49] consiste en minimizar el error cambiando sistemáticamente los pesos con el siguiente valor:

$$\Delta w_{ij} = \alpha h_j h_i \tag{1.10}$$

donde w_{ij} es el peso de la conexión entre las unidades i y j, h_i y h_j son las salidas que tales unidades proporcionan y α es la tasa de aprendizaje. Este procedimiento tiene una base biológica, ya que implica reforzar sólo las conexiones entre elementos de la red que se activen simultáneamente.

Grossberg [Gros76] propuso una modificación a este esquema, de forma tal que:

$$\Delta w_{ij} = \alpha F(h_j - w_{ij}) \tag{1.11}$$

donde F es la función de activación.

1.3.2.4. Reglas de aprendizaje competitivo

En el aprendizaje competitivo [Gros72, vdM73, Koho82, Rume85] la red detecta las regularidades de los patrones de entrada; esto es, capta las características generales que pueden usarse al objeto de agrupar un conjunto de patrones en clases. Es un procedimiento que divide el conjunto de vectores de entrada en un número determinado de clases de tal forma que todos los vectores de cada clase sean similares entre sí y queden representados por una única neurona de salida. Se denomina competitivo porque, en un instante dado, cada unidad compite con las de su vecindad para ser la única activa (*winner-take-all*). Existen evidencias [Hayk94] de que este tipo de aprendizaje se encuentra en las redes neuronales biológicas.

1.4. Computación Evolutiva

El campo de la computación evolutiva ha experimentado un rápido crecimiento en los últimos años. Tanto ingenieros como científicos, unos y otros con diferente formación y conocimientos, colaboran para poder resolver algunos de los más complicados problemas actualmente conocidos. Para ello se utilizan los denominados **Algoritmos Evolutivos** (AE), un conjunto de algoritmos de búsqueda basados en técnicas probabilísticas que están ofreciendo resultados esperanzadores. La aplicación de AE a problemas de optimización es directa. Basta considerar la resolución de éstos problemas como un proceso de búsqueda de los parámetros que configuran de la forma más adecuada el sistema para el cual se requiere un funcionamiento óptimo.

1.4.1. Fundamentos biológicos de la Computación Evolutiva

Los algoritmos evolutivos están inspirados en la **Teoría de la Evolución** que Charles Darwin describiera en su libro *"Sobre el Origen de las Especies por medio de la Selección Natural"* [Darw59]. Para llegar a desarrollarla, Darwin estuvo muy influido por otra teoría defendida a su vez por el geólogo James Hutton, denominada **uniformismo** y que sostenía que la Tierra había sido moldeada no sólo por acontecimientos bruscos, sino también por procesos lentos y graduales [Hutt02]. Influyentes también fueron las hipótesis de Baptiste Lamarck acerca de cómo las especies superiores surgen o *progresan* [Lama09] a partir de las especies menores, impelidas según él por dos fuerzas de origen bien distinto: la herencia de características adquiridas y la existencia de un principio universal que llevaría a toda criatura a ascender dentro de la *Scala Naturae* descrita por Aristóteles unos 2100 años antes. Finalmente, la versión definitiva de la teoría darwiniana se vio completada con la aportación del naturalista Alfred Russell Wallace, el cual le escribiera contándole cómo le había surgido la idea de que la variabilidad de los seres vivos podía proporcionar el material en el cual, mediante la eliminación de los seres menos adaptados, sólo los más aptos continuarían su proceso de mejora evolutiva.

Así pues, y a pesar de ser una visión incompleta del proceso de evolución, el libro de Darwin defendía la hipótesis de que la generación de nuevas especies, más adaptadas al entorno existente en cada momento, se producía gracias a: 1) pequeños cambios que heredan los seres vivos y 2) un proceso de selección natural. No obstante, el mecanismo por el cual las

características eran transmitidas de padres a hijos era totalmente desconocido. El principal problema de la hipótesis creada por Darwin radicaba en que, en algún momento, todos los individuos tendrían las mismas características y serían similares en todo, lo cual contrastaba seriamente con las evidencias recogidas por él mismo a lo largo de sus investigaciones.

No sería hasta mucho más tarde cuando se descubriera que las características que definían a cada individuo se heredaban de forma discreta. Esta nueva teoría, que debemos a Mendel, describe cómo los caracteres de un individuo se toman del padre o de la madre, dependiendo de su carácter dominante o recesivo. A cada una de las distintas características se le denominó **gen** y a los valores que podía tomar cada gen se les llamó **alelos**.

Paradójicamente, los estudios realizados por Mendel fueron totalmente ignorados y hubo que esperar a la década de los 30 para que Sir Ronald Aylmer Fisher relacionara las teorías de Darwin y Mendel en lo que se denominaría la **teoría sintética de la evolución** o teoría neo-darwiniana [Fish30, Fish00]. El acierto de este investigador consistió en demostrar que la teoría acerca de los genes desarrollada por Mendel era capaz de proporcionar un mecanismo que hacía posible la evolución de las especies tal y como había sido defendida por Darwin.

Simultáneamente, el biólogo alemán Walther Flemming describió los cromosomas como ciertos filamentos en los que se agregaba la cromatina del núcleo celular durante la división; poco más adelante se descubrió que las células de cada especie viviente tenían un número fijo y característico de cromosomas. Más tarde, durante los años 50, Watson y Crick descubrieron que la base molecular de los genes estaba en el ADN. Los cromosomas están compuestos por tanto de ADN y los genes están en los cromosomas.

Finalmente, se comprobó que las características concretas de cada individuo dependen tanto del código genético del mismo, al que se empezó a denominar **genotipo**, así como de la presión ambiental, la historia vital del individuo y otros mecanismos dentro del cromosoma. Al cuerpo final resultante de la intervención de ambos factores se le denominó **fenotipo**.

Esta es la base de la teoría del neo-darwinismo, que afirma que la historia de la mayoría de la vida está causada por una serie de procesos que actúan en y dentro de las poblaciones: reproducción, mutación, competición y selección. Bajo estas premisas, la evolución puede definirse como cambios en el conjunto genético de una población.

1.4.2. Desarrollo histórico

Podemos considerar que el primer uso de procedimientos evolutivos en computación se debe a Reed, Toombs y Baricelli [Reed67], los cuales trataron de hacer evolucionar un tahúr que jugaba a un juego de cartas simplificado. Las estrategias de juego consistían en una serie de 4 probabilidades de apuesta alta o baja con una mano alta o baja, con cuatro parámetros de mutación asociados. Se mantenía una población de 50 individuos, existía mutación, había intercambio de probabilidades entre dos padres y es de suponer que los perdedores se eliminaban de la población. Además de crear buenas estrategias, llegaron a la conclusión de que el entrecruzamiento no aportaba mucho a la búsqueda.

Otro tipo de algoritmos inspirados en la naturaleza son los denominados **algoritmos genéticos** (AG) [Holl75, Gold89, Davi96], ideados por John Holland en los 60 y desarrollados e investigados por él mismo y sus colegas de la Universidad de Michigan en los 60 y 70. En contraste con lo que otros investigadores intentaban, Holland no pretendía resolver problemas específicos, sino estudiar formalmente el fenómeno de la adaptación tal y como ocurre en la naturaleza, y desarrollar métodos para aplicar los mecanismos de la adaptación natural a sistemas de computadores. Holland presentó en su libro *"Adaptation in natural and Artificial systems"* [Holl75] el algoritmo genético como una abstracción de la evolución biológica y dio los principios teóricos que dirigen el proceso de evolución de soluciones mediante el uso de AG.

Simultáneamente, Rechenberg [Rech65, Rech73] y Schwefel [Schw75, Schw77] describieron las **estrategias de evolución** (EE) [Schw95, Bäck96], métodos de optimización paramétrica que trabajan con poblaciones de cromosomas compuestos por números reales. Hay diversos tipos de estrategias de evolución. En la más común, se crean nuevos individuos de la población añadiendo un vector mutación a los cromosomas existentes en la población; en cada generación, se elimina un porcentaje de la población, y los restantes generan la población total, mediante mutación y entrecruzamiento. Una revisión sobre las estrategias evolutivas se puede encontrar en Bäck, Hoffmeister y Schwefel [Back91].

A partir de los años 60 se han desarrollado algoritmos o métodos que podríamos llamar **evolutivos modernos** y se han seguido investigando hasta nuestros días. Algunos de ellos coinciden en el tiempo con los algoritmos genéticos, aunque fueron desarrollados independientemente sin conocimiento unos de otros. Entre estos métodos, encontramos la **programación evolutiva** (PE) de Fogel, Owens y Walsh [Foge66, Foge95], inicialmente un intento de usar la evolución para crear máquinas inteligentes,

que pudieran prever su entorno y reaccionar adecuadamente a él. La simulación de la máquina pensante se realizaba mediante un **autómata celular**, esto es, un conjunto de estados y reglas de transición entre ellos, tales que cuando se recibe una entrada, cambia o no el estado actual del autómata y se produce una salida.

Fogel, por ejemplo, trataba de hacer aprender a estos autómatas a encontrar regularidades en los símbolos que se le iban enviando. Como método de aprendizaje usó un algoritmo evolutivo: una población de diferentes autómatas competía para hallar la mejor solución, es decir, predecir cual iba a ser el siguiente símbolo de la secuencia con un mínimo de errores; los peores autómatas, en concreto la mitad de la población, eran eliminados cada generación y sustituidos por otros nuevos resultantes de una mutación de los existentes.

El método desarrollado por Fogel permitió hacer evolucionar autómatas que predecían algunos números primos (por ejemplo, uno de ellos, cuando se le daban los números más altos, respondía siempre que no era primo; la mayoría de los números mayores de 100 son no primos). En cualquier caso, estos primeros experimentos demostraron el potencial de la evolución como método de búsqueda de soluciones novedosas.

Más recientemente, la aplicación de las técnicas de la computación evolutiva a la generación automática de programas de ordenador dio lugar a la denominada **programación genética** (PG) [Koza92]. El inicio de esta nueva técnica podría remontarse posiblemente a Alan Turing por ser el primero en experimentar con una especie de evolución en las formas de aprender comportamientos. Esta evolución implicaba cambios en el procesamiento realizado por una máquina que intentaba no ser reconocida por un operador humano. En 1958, Friedberg [Frie58, Frie59] utilizó una especie de crédito que se asignaba a cada instrucción en función de cómo ayudaba al éxito del programa al que pertenecía. En 1966, Fogel, Owens y Walsh [Foge66] aplicaron mutaciones a programas padres obteniendo así programas hijos ligeramente distintos a ellos en cuanto a su estructura interna. Mucho más tarde, en 1985, Cornel empleó por primera vez el entrecruzamiento de subárboles para hacer evolucionar expresiones simbólicas. Un año después, Hicklin [Hick86] haría lo propio con expresiones en LISP y en 1987 Fujiki y Dickinson [Fuji87] describieron sus experimentos en los que se hacía evolucionar una expresión condicional escrita en LISP mediante operadores muy parecidos a los usados por los AG. Finalmente, la ampliación del entrecruzamiento de subárboles a expresiones simbólicas representadas mediante árboles genéricos fue utilizada por Koza en 1992 para investigar una forma aún muy básica de PG a la que se refirió como algoritmos genéticos jerárquicos.

A partir de las ideas que Michalewicz propuso en su libro *"Genetic Algorithms + Data Structures = Evolution Programs"* [Mich99] se abrió un nuevo campo de actuación en el ámbito del desarrollo de AE. La idea básica consistía en prescindir de la representación usual de los individuos en la población (en cadenas de bits o vectores de números reales, como se venía haciendo) y al mismo tiempo aplicar el paradigma principal de la programación procedural a la computación evolutiva: aplicar algoritmos a estructuras de datos, por ser entidades distintas que deben estar separadas, y formar programas evolutivos mediante la unión e interacción de ambos.

Siguiendo el razonamiento de Michalewicz, y teniendo en cuenta que el paradigma de programación más avanzado actualmente es el de la programación orientada a objetos, se podría definir la **computación evolutiva orientada a objetos** como: *Algoritmos + estructuras de datos = Objetos Evolutivos* (EO, *Evolutionary Objects*). La programación orientada a objetos va a permitir, por tanto, que las estructuras de datos y los algoritmos que a ellas se aplican se encuentren encapsulados en las denominadas **clases**. La interacción con las instancias de esas clases se podrá realizar a través de una interfaz que es común para todas las instancias de una determinada clase.

El uso de Objetos Evolutivos permite acceder a un nivel de abstracción en el que, sin ser realmente algoritmos genéticos, ni programas evolutivos, ni programas genéticos, ni otros paradigmas evolutivos definidos previamente, cualquiera de estos paradigmas puede ser implementado, dando lugar a un nuevo tipo de algoritmos en los cuales son los objetos los que directamente pueden ser hechos evolucionar.

1.4.3. La Computación Evolutiva en el contexto de los problemas de búsqueda y optimización

Antes de entrar en la descripción genérica de los AE y de presentar detalladamente la abstracción de Objetos Evolutivos, conviene situar ambos conceptos dentro el marco de los problemas de búsqueda y optimización, comparándolos con otros métodos de optimización bien conocidos, como la búsqueda tabú y derivados, los procedimientos de escalada o el enfriamiento simulado.

1.4.3.1. Búsqueda Tabú, Búsqueda Dispersa y *Path Relinking*

Aunque los orígenes de la **Búsqueda Tabú** (*Tabu Search*, TS) pueden situarse en diversos trabajos publicados hace alrededor de 25 años, oficial-

mente, el nombre y la metodología fueron introducidos posteriormente por Glover [Glov89]. Por su parte, la **Búsqueda Dispersa** (*Scatter Search*, SS) y ***Path Relinking*** (PR) han derivado de TS, siendo inicialmente parte o ampliación de ella y convirtiéndose, a la postre, en mecanismos diversos aunque interrelacionados.

TS es una técnica para resolver problemas combinatorios de gran dificultad basada en principios generales de Inteligencia Artificial (IA). En esencia es una técnica metaheurística que puede ser utilizada para guiar cualquier procedimiento de búsqueda local en la búsqueda voraz (*greedy*) del óptimo del problema. Por voraz entendemos la estrategia de evitar que la búsqueda quede "atrapada" en un óptimo local que no sea global. A tal efecto, TS usa el concepto de memoria y lo implementa mediante estructuras simples con el objetivo de dirigir la búsqueda teniendo en cuenta la historia de ésta. Es decir, el procedimiento trata de extraer información de lo ya acontecido y actuar en consecuencia. En este sentido puede decirse que hay un cierto aprendizaje y que la búsqueda es inteligente. El principio de TS podría resumirse como:

Es mejor una mala decisión basada en información que una buena decisión al azar, ya que, en un sistema que emplea memoria, una mala elección basada en una estrategia proporcionará claves útiles para continuar la búsqueda. Una buena elección fruto del azar no proporcionará ninguna información para posteriores acciones.

TS comienza de la misma forma que cualquier procedimiento de búsqueda local, procediendo iterativamente de una solución x a otra y en el entorno, $N(x)$, de la primera. Sin embargo, en lugar de considerar todo el entorno de una solución, TS define el entorno reducido $N^*(x)$ como aquellas soluciones disponibles del entorno de x. Así, se considera que a partir de x, sólo las soluciones del entorno reducido son alcanzables.

Existen muchas maneras de definir el entorno reducido de una solución. La más sencilla consiste en etiquetar como tabú las soluciones previamente visitadas en un pasado cercano. Esta forma se conoce como **memoria a corto plazo** (*short term memory*) y está basada en guardar en una lista tabú, T, las soluciones visitadas recientemente. Así en una iteración determinada, el entorno reducido de una solución se obtendría como el entorno usual eliminando las soluciones etiquetadas como tabú.

El objetivo principal de etiquetar las soluciones visitadas como tabú es el de evitar que la búsqueda quede encerrada en un ciclo sin salida. Por ello se considera que tras un cierto número de iteraciones la búsqueda está en una región distinta y pueden eliminarse de la lista T las soluciones

antiguas. De esta forma se reduce el esfuerzo computacional de calcular el entorno reducido en cada iteración. En los orígenes de TS se sugerían listas de tamaño pequeño, actualmente se considera que las listas pueden ajustarse dinámicamente según la estrategia que se esté utilizando.

Adicionalmente, se define un **nivel de aspiración** como una condición o conjunto de ellas tales que, de satisfacerse, permitirían alcanzar una solución aunque se encuentre incluida en la lista tabú. Una implementación sencilla consiste en permitir alcanzar una solución siempre que mejore a la mejor almacenada, aunque esté etiquetada tabú. De esta forma se introduce cierta flexibilidad en la búsqueda y se mantiene su carácter agresivo.

La búsqueda dispersa (SS) y *path relinking* (PR) son por su parte métodos evolutivos que construyen nuevas soluciones para un determinado problema mediante la combinación de otras soluciones preexistentes. Para ello usan estrategias concretas que aprovechan el conocimiento que se tiene sobre el problema en cuestión. Al igual que TS, utilizan formas especiales de memorias adaptativas para mejorar el proceso de búsqueda.

A partir de lo que se denomina el **conjunto de soluciones de referencia**, el procedimiento para generar nuevas soluciones consiste en formar combinaciones lineales que se acomodan en caso de que haya que preservar ciertos requisitos, como es el caso cuando la solución está formada por magnitudes discretas. Las combinaciones resultantes forman como unos caminos entre soluciones e incluso que van más allá de las soluciones conocidas. Las soluciones presentes en dichos caminos pueden a su vez ser semilla de otros nuevos. La diferencia existente entre la SS y PR está en que mientras la primera opera en un subconjunto del espacio euclídeo (aquél determinado por las restricciones impuestas a las soluciones), PR lo hace a lo largo de todo el espacio posible de vecinos de una determinada solución.

Los fundamentos en los que descansan SS y PR son los siguientes:

a) Generalmente, existe información útil sobre la forma o posición de soluciones óptimas contenida en una colección diversa y apropiada de soluciones élite.

b) Es importante proporcionar mecanismos que extrapolen más allá de las regiones comprendidas por las soluciones consideradas cuando dichas soluciones son combinadas en un intento de aprovechar la información que contienen.

c) La probabilidad de utilizar la información contenida en la unión de soluciones élite aumenta cuando al crear nuevas combinaciones se tienen en cuenta múltiples soluciones de forma simultánea.

El método empleado por SS consiste en generar un conjunto de soluciones de referencia inicial. Se emplea entonces un método de **generación de diversidad** para generar un conjunto mayor de soluciones diversas (P). A continuación se usa un método de **mejora** con cada nueva solución en P. De este conjunto P, se toman b soluciones para crear un nuevo conjunto de referencia. Dichas soluciones serán distintas y maximizarán la diversidad del nuevo conjunto de referencia, además quedarán ordenadas en función de la bondad de cada una de ellas. El siguiente paso consiste en generar todos los pares, tríos, etc. de soluciones (este es uno de los parámetros configurables) para aplicar a cada uno de ellos un método de **combinación de soluciones** generando una o más nuevas soluciones. Cada nueva solución es modificada usando de nuevo el método de mejora. Si la solución final es mejor al menos que la peor del conjunto de referencia será incluida en éste, reemplazando a la peor solución, provocando la reordenación del conjunto de referencia y haciendo que desaparezca el par, trío, etc. que la ha generado. El proceso acaba cuando no se admiten nuevas soluciones en el conjunto de referencia.

Como puede verse, el método es muy agresivo en su intento de mejorar las soluciones ya encontradas, y de hecho el mecanismo de generación de diversidad sólo es usado una vez al comienzo del

PR era en sus inicios un método para integrar las tareas de intensificación y diversificación en el contexto de TS. De igual forma, puede considerarse como una extensión de los mecanismos de combinación de SS. En vez de generar directamente una nueva solución al combinar dos o más soluciones existentes, PR genera caminos entre dichas soluciones y más allá de ellas en el espacio de vecindad de las mismas. El carácter de dichos caminos se especifica fácilmente haciendo referencia a atributos de las soluciones que son añadidos, eliminados o modificados en cada uno de los pasos que se da para construir cada camino. Ejemplos de dichos atributos pueden ser enlaces y nodos en un grafo, posiciones de secuencias en una planificación, así como valores para variables o funciones de variables.

Para generar caminos sólo es necesario seleccionar movimientos que empezando por la *solución inicial* introduzcan progresivamente atributos contenidos en la *solución guía* o solución final.

Ambos métodos están siendo estudiados en la actualidad para mejorar su comportamiento añadiendo otros métodos de búsqueda local que intenten encontrar mejores soluciones a partir de las encontradas por ellos.

1.4.3.2. Procedimientos de escalada

Los **métodos indirectos de resolución** buscan el óptimo llevando iterativamente el valor de la función objetivo en la dirección de máxima pendiente. Procediendo de este modo se garantiza encontrar un óptimo local o **subóptimo** (un máximo o mínimo, esto es, un punto rodeado de puntos peores que él) aunque no necesariamente el óptimo global.

Este **procedimiento de escalada** (*hillclimbing*) da lugar a algoritmos de resolución muy sencillos, siempre y cuando se disponga de una buena técnica de determinación del signo de la pendiente. Estos métodos usan una técnica iterativa de mejora, que es aplicada a un sólo punto (el punto actual) en el espacio de búsqueda. En cada iteración se determina una región de máxima pendiente en la vecindad del punto actual y se selecciona el mejor punto de dicha vecindad; ésto suele ser más eficaz que determinar directamente la dirección de máxima pendiente. Si el nuevo punto proporciona un valor de la función objetivo estrictamente mejor que el anterior se le convierte en el punto actual, en caso contrario se elige algún otro vecino. El método termina cuando no sea posible ninguna mejora.

En la fig. 1.3 se muestra un algoritmo simple de optimización por escalada.

Esta técnica tiene dos serias desventajas que van en detrimento de su robustez. Primero, admite, de un modo u otro, que aunque no se pueda dar una forma cerrada a la función objetivo, tiene sentido el concepto de pendiente; ello restringe su campo de aplicación; además, requiere que en todo momento esté completamente definida una dirección preferente de búsqueda. En definitiva, por una causa o por otra, se precisa una gran cantidad de conocimiento específico. En segundo lugar, es esencialmente local, esto es, sólo puede hallar subóptimos y, lo que es peor, en el caso de que haya varios (existe *multimodalidad*) el acabar en uno u otro depende del punto inicial, siendo imposible determinar a priori a qué subóptimo llevará un punto inicial dado. Es más, no hay información acerca del error relativo cometido con respecto al óptimo global. Esas dos características se resumen en que, por un lado, esta técnica de optimización es muy eficiente pero por otro muy específica. Las desventajas de la alta especificidad no se logran compensar con la alta eficiencia, lo que la hace poco robusta.

1.4.3.3. Enfriamiento simulado

La técnica de **enfriamiento simulado** (*simulated annealing*) [Aart89] es un método inspirado en la Naturaleza. Básicamente es una analogía del

```
Seleccionar un punto de partida al azar X_0.
para t=0 hasta LimIteraciones
        E_t = EvaluarFuncionObjetivo(X_t)
        parar = FALSO
        mientras (NO parar)
                P_t= seleccionar un vecino de X_t
                N_t= Evaluar( P_t )
                si (N_t > E_t) entonces
                        X_t = P_t
                        E_t = N_t
                si no
                        parar = VERDAD
        fin_mientras
fin_para
```

Figura 1.3: Estructura general de un algoritmo de escalada.

modo en que los materiales solidifican en un cristal según se van enfriando, es decir, reduciendo su energía hasta un mínimo de energía, ofreciendo máxima resistencia, y la búsqueda del óptimo de un problema. Metropolis et al. [Metr58] propusieron el método para encontrar la configuración en equilibrio de una colección de átomos a una temperatura dada. La conexión entre este algoritmo y la minimización matemática fue propuesta por Kirkpatrick et al. [Kirk83] como una técnica de optimización para problemas combinacionales principalmente. Esta técnica evita muchas de las desventajas de los métodos de escalada, especialmente que la solución no dependa del punto de inicio. Esto se consigue introduciendo una probabilidad de aceptación del nuevo punto, que será 1 si el nuevo punto mejora al antiguo, o bien dependerá de la diferencia entre los valores de bondad de ambos puntos, y de un nuevo parámetro llamado **temperatura**, por su analogía con el símil físico. A menor temperatura, menor es la probabilidad de aceptar un nuevo punto. Durante la ejecución del algoritmo, la temperatura decrece y el algoritmo acaba cuando se alcanza un valor de la temperatura para el cual, virtualmente, ya no se van a aceptar más cambios.

En un problema típico de enfriamiento simulado, se define la función a minimizar (**función de coste**, f), y después, a partir de una solución aleatoria inicial, se van generando k diferentes soluciones (llamado número de cambios) que se comparan con la solución que en ese momento es la ac-

tual. La mejor solución encontrada (menor coste) se adopta como siguiente actual, aunque una solución de mayor coste (peor) se adoptará con una probabilidad que irá decreciendo con el tiempo, de acuerdo al valor de la temperatura.

La búsqueda realizada por el algoritmo actúa del siguiente modo: se genera un punto o estado, s_n, a partir del anterior, s_0, utilizando un operador de cambio (generador de nuevos estados). Si el nuevo estado es mejor, se acepta. Si el nuevo es peor, se aceptará con una probabilidad de:

$$p_a = e^{-\frac{\Delta f(s)}{T}} \tag{1.12}$$

donde $\Delta f(s)$ se define como el incremento de la función de coste y T es la temperatura. Dicho parámetro se va decrementando paulatinamente mediante una **función de reducción de temperatura**. La más simple es:

$$T_{n+1} = \alpha T_n \tag{1.13}$$

donde α es una constante menor que 1. Este **esquema exponencial de enfriamiento** fue propuesto por Kirkpatrick et al. [Kirk83] usando $\alpha = 0.95$. Posteriormente Kirkpatrick [Kirk84] sugirió que T_0 debería depender de f de acuerdo a:

$$T_0 = -\frac{\Delta f^*}{\ln(p_a)} \tag{1.14}$$

donde Δf^* es el incremento en la función objetivo observado y p_a es la probabilidad inicial de aceptación de la solución (se suele utilizar 0.8).

En la fig. 1.4 se muestra una versión del algoritmo de enfriamiento simulado descrito en [Mich99]

1.4.4. Estructura genérica de un AE

Aunque existen distintos tipos de AE (algoritmos genéticos, programación evolutiva, estrategias de evolución) y cada uno poseee numerosas variantes, dos son las características que distinguen los AE del resto de algoritmos de búsqueda. En primer lugar, son algoritmos basados en poblaciones. En segundo lugar, existe un intercambio de información entre los individuos de una población o entre individuos de poblaciones distintas. Este intercambio de información es necesario para poder llevar a

$n = 0$
inicializar la temperatura T
seleccionar aleatoriamente un estado s_o
calcular $f(s_o)$
repetir
    para j=1 **hasta** k
        seleccionar un nuevo estado s_n en la vecindad
            de s_o modificando s_o
        calcular $f(s_n)$.
        $\Delta f = f(s_o) - f(s_n)$
        si ($\Delta f < 0$) **O** ($random(0,1) < e^{-\frac{\Delta f}{T}}$)
            entonces $s_o = s_n$
    fin_para
    $T = f_T(T, n)$
    $n = n + 1$
hasta que ($T < T_{min}$)

Figura 1.4: Estructura general de un algoritmo de enfriamiento simulado.

cabo una búsqueda no aleatoria a lo largo del espacio de soluciones y se produce como resultado de aplicar procesos de selección, competición y recombinación entre los distintos individuos.

De forma resumida, un AE puede ser representado según aparece en la fig. 1.5. Los operadores de búsqueda especificados en esta figura suelen denominarse **operadores genéticos** en el ámbito de los AG. Los distintos AE se definirán utilizando diferentes representaciones o esquemas de codificación, así como operadores y métodos de selección. De esta forma, mientras los AG utilizan tanto recombinación como mutación y ponen el énfasis en la evolución genética, la programación evolutiva utiliza solo mutación y se centra más en la evolución del comportamiento de los individuos.

Además de las visión presentada de los AE que podría resumirse en que los individuos más evolucionados sobreviven, existe un segundo punto de vista y es entenderlos como mecanismos de búsqueda guiada del tipo *generación-prueba*. Bajo este punto de vista es más fácil comparar los AE con otros algoritmos de búsqueda, como los ya vistos enfriamiento simulado, búsqueda tabú o escalada.

Generar aleatoriamente una población $P(0)$
y establecer $i = 0$
repetir
Evaluar el *fitness* de cada individuo en $P(i)$
Seleccionar un conjunto de padres en $P(i)$
basándonos en su *fitness*
Obtener la nueva generación de individuos $P(i+1)$
mediante la aplicación de operadores de búsqueda
a los padres seleccionados
hasta que que la población converja o se acabe el tiempo prefijado

Figura 1.5: Estructura general de un algoritmo evolutivo.

La fig. 1.6 muestra el esquema general de los algoritmos de generación-prueba. Varios de los algoritmos de escalada siguen estrictamente dicho esquema. Los algoritmos de enfriamiento simulado aportan la posibilidad de cambiar la solución actual por una menos aceptable en función de cierta probabilidad. Por su parte, los AE pueden ser considerados como la versión extendida a poblaciones del algoritmo general presentado en la figura. Utilizan mutación y recombinación como métodos de búsqueda que perturban la solución actual y la selección para decidir en qué grado una solución es aceptable.

Generar solución inicial aleatoria y denominarla *solucion actual*
repetir
Generar nueva solución a perturbando la actual
Comprobar si nueva solución es *aceptable*
Si es aceptable, adoptarla como la solución actual
Si no, conservar la solución actual sin modificar
hasta que la solución sea satisfactoria

Figura 1.6: Estructura general de un algoritmo de tipo generación-prueba.

1.4.5. Consideraciones sobre los AE

Los AE pueden clasificarse en **elitistas** y **no elitistas**. Los algoritmos elitistas son aquellos en los que la mejor solución encontrada hasta el mo-

mento siempre está presente en la generación actual. Los algoritmos de programación evolutiva para optimización numérica, las estrategias evolutivas $(\mu + \lambda)$ y los algoritmos genéticos elitistas entran dentro de esta definición.

La diferencia más notable que existe entre algoritmos elitistas y no elitistas la encontramos al hacer un estudio de la convergencia de los mismos. Se define la **convergencia global de un AE** como el hecho de que la probabilidad de que la solución encontrada en tiempo n pertenezca al espacio de óptimos locales tienda a 1, todo ello en el límite de n tendiendo hacia infinito. Así, mientras la convergencia de los algoritmos elitistas puede ser estudiada mediante el uso de cadenas de Markov, no existe un método bien definido para su estudio en algoritmos elitistas.

Aunque la convergencia es un aspecto a considerar en la descripción de un AE, no es menos cierto que ejerce un papel limitado en la práctica y apenas ofrece ayuda a la hora de diseñar nuevos algoritmos evolutivos. Por ello, desde el punto de vista del diseño del algoritmo, es más interesante resaltar la complejidad computacional de cada uno de ellos. Este estudio de la complejidad sin embargo es difícil de llevar a cabo pues ha hacerse para cada problema en particular. A pesar de la falta de estudios sobre complejidad en el campo de la computación evolutiva, es bien sabido que muchos investigadores trabajan con algoritmos genéticos en problemas deceptivos, e intentan identificar problemas que resulten difíciles para estos algoritmos y así desarrollar nuevos métodos que permitan resolverlos.

Con relación a los operadores utilizados para encontrar la solución al problema, se suelen clasificar como operadores que reducen la diversidad genética y operadores que la incrementan. Así, la mutación y el entrecruzamiento (recombinación, cruce o *crossover*) está considerados como operadores que, mediante el incremento de dicha diversidad, permiten realizar una exploración del espacio de búsqueda.

La mutación es tradicionalmente considerada como un operador secundario en los AG y, por contra, es un operador principal para la programación evolutiva y las estrategias de evolución.

Por su parte, el entrecruzamiento es un operador considerado principal para los AG. No obstante, en la práctica su eficacia depende del tipo de esquema de codificación utilizado para los individuos así como del problema en particular que se esté resolviendo. De hecho, para un problema de optimización combinatoria dado, lo más usual es diseñar un operador de recombinación específico si queremos garantizar un correcto funcionamiento.

En el polo opuesto a los operadores de mutación y entrecruzamiento

se hallan los operadores de selección, cuya misión consiste en realizar una simulación de la selección natural que se da entre los seres vivos y que hace que las especies mejor adaptadas sobrevivan y las demás se extingan.

Los operadores de selección usados en la mayoría de algoritmos evolutivos pueden ser clasificados en tres categorías: proporcionales al *fitness* (también denominada selección de ruleta), basados en clasificación y basados en torneos.

El **método de la ruleta** es el más antiguo y simple de implementar y utilizar, si bien tienen dificultades para manejar los casos en los que aparecen superindividuos. Además, este operador hace converger el algoritmo de forma prematura si no se añaden métodos que permitan escalar el *fitness* de cada individuo.

El **método basado en la clasificación** no depende en sí del *fitness* del individuo, sino de cómo queda clasificado el individuo en la población con respecto a dicho *fitness*. Existen distintos operadores basados en clasificación y tiene la ventaja de poder ser utilizado en problemas de optimización multiobjetivo, en los cuales no existe una única solución sino un conjunto de pareto-óptimos.

Finalmente, los **métodos basados en torneo** tienen la ventaja de no necesitar toda la población para ser llevados a cabo, lo cual los hace ideales para algoritmos paralelos. Además, la evaluación de los individuos se hace bajo demanda, por lo que en principio no es necesario evaluar todos los nuevos individuos que es en muchos problemas la tarea más costosa. En ellos, la probabilidad de seleccionar un individuo depende de un subconjunto tomado de entre la población total. El ejemplo más típico es el de Boltzman, propuesto por Goldberg en [Gold89].

La elección de un AE para resolver el problema de la configuración de RNFBR se ha hecho teniendo en cuenta que el espacio de búsqueda de las soluciones es particularmente grande. Así, dado un problema a resolver, su espacio de soluciones está formado por todas las RNFBR, compuestas por cualquier número de neuronas en su capa oculta, pudiendo tener cada una de ellas por centro cualquier punto del espacio de entrada y cualquier vector de radios en los rangos permitidos para los mismos. La información que a priori pueda tenerse acerca de la solución sólo hace referencia a los posibles valores para los centros de las FBR. Éstos pueden ser tomados de entre el conjunto de patrones de entrada sin que ello signifique que dichos valores sean los óptimos ni que su número sea el adecuado.

El algoritmo EvRBF propuesto en esta tesis es un algoritmo elitista, por lo que en la última generación siempre está presente la mejor solución encontrada. Utiliza operadores de entrecruzamiento para modificar las - RNFBR de forma local y generar una especie de bloques básicos sobre los

que construir la solución final. Igualmente utiliza operadores de mutación para generar diversidad y explorar el espacio de búsqueda. El operador de selección que utiliza es el de torneo el cual permite a soluciones subóptimas reproducirse con cierta probabilidad y no precisa del entrenamiento y evaluación de todas las nuevas RNFBR generadas por los anteriores operadores.

Los operadores de entrecruzamiento y mutación que se han generado (ver capítulo 3) operan directamente en el espacio de soluciones, esto es, en el espacio de todas las posibles RNFBR para de esta forma poder hacer evolucionar todas las características de las mismas, sin poner restricciones en cuanto a tamaño de los cromosomas o límites de representación de números reales. En EvRBF cada individuo representa una RNFBR completa. De esta forma, se puede evaluar correctamente el error alcanzado por cada red en la fase de entrenamiento y validación, sin necesidad de recurrir a métodos artificiales que propagan el error a cada neurona para así estimar su *fitness* y que pueden depender del problema estudiado en cada momento.

1.5. La biblioteca EO

La implementación del método EvRBF se ha llevado a cabo utilizando la biblioteca en C++ **EO**. La elección de esta biblioteca ha estado fundamentada, por un lado, en la facilidad con la que permite definir los objetos que deben ser hechos evolucionar y los operadores que deben ser aplicados a los mismos. Por otro lado, EO separa perfectamente lo que es la representación del objeto que se hace evolucionar de los operadores que lo manipulan, permitiendo cambiar unos y otros de forma independiente sin variar la estructura del algoritmo principal. Además, el resto de componentes necesarios para terminar de configurar el algoritmo (operadores de selección, reproductores, políticas de reemplazamiento de individuos o gestión de poblaciones, entre otros) vienen ya incluidos en la propia biblioteca por lo que se facilita y agiliza la construcción de un algoritmo al convertirla en una tarea de reutilización de componentes.

La creación de esta biblioteca está inspirada en uno de los libros más conocidos sobre Computación Evolutiva: *"Genetic Algorithms + Data Structures = Evolution Programs"* [Mich99]. La propuesta que hace Michalewicz, el autor, sirve como idea inicial para introducir el concepto de Objeto Evolutivo (*Evolutionary Object*), en cuanto a que EO pretende eliminar el problema de elegir una representación para resolver un problema y a la vez posibilitar la aplicación del paradigma principal de la programación pro-

cedural a la computación evolutiva: los *algoritmos* se aplican a las *estructuras de datos*, ambos son distintos y deben estar separados, aunque juntos forman los programas evolutivos [Mere99, Mere00a, Mere00b].

La principal aportación de la computación orientada a objetos a la anterior propuesta consiste en que los algoritmos y las estructuras de datos a las que se aplican van a estar encapsulados en objetos, regulándose la interacción entre objetos mediante una interfaz. Cada objeto conoce su funcionamiento interno y aquellos que deseen utilizarlo (los **clientes** de ese objeto) habrán de hacerlo mediante la funcionalidad asociada a la interfaz externa del mismo.

A continuación se presentará el concepto de **Objeto Evolutivo** (OE), describiendo las características fundamentales que lo diferencian de otros paradigmas de computación evolutiva.

1.5.1. Características principales de los OE

La biblioteca EO define interfaces para todos los objetos, de forma que quede restringido el modo en que unos usan a los otros. A pesar de las restricciones que introducen dichas interfaces, éstas deben dotar a los objetos de ciertas características básicas para poder hacerlos evolucionar. Estas características fundamentales incluyen que un OE sea **replicable, mutable, combinable** y **comparable**. Estas propiedades se usarán como analogía computacional para los tres criterios evolutivos descritos por Maynard-Smith [Mayn75] (hereditabilidad, variabilidad y fecundidad). Examinemos más detalladamente cada una de ellas:

a) **Replicabilidad**. Es necesario poder obtener copias de un OE cualquiera, ya sea por él mismo o mediante el uso de otros objetos (*replicadores*). También debería ser posible crear OE desde cero (a través *factorías* de objetos).

b) **Mutabilidad**. Un OE debe poder mutar o modificarse de forma que se incremente la diversidad de un conjunto (población) de OE. Esto quiere decir que el OE, por sí mismo o mediante un objeto cuyo cometido será modificar otros objetos (*objeto mutador*), podrá cambiar de diversas formas, pero el funcionamiento de dichos cambios no tiene por qué conocerse desde fuera, ni tampoco tiene por qué haber una representación específica para permitir una mutación de un objeto. Básicamente, el cliente sólo necesita saber que el objeto mutado cambiará de alguna forma.

c) **Combinabilidad**. En algunos casos convendrá combinar dos o más OE para crear uno nuevo (de forma similar a como funciona el entrecruzamiento en los AG). Aunque esto no será posible en todas las ocasiones, cuando lo sea, la operación hará decrecer la diversidad, en el sentido que hace los OE más parecidos entre sí. Tal y como ocurre con la mutación, el funcionamiento interno de la recombinación no tiene por qué ser conocido por el cliente.

d) **Comparabilidad**. Para decidir qué objetos son mejores que otros al realizar una tarea en particular, deberían poderse comparar para decidir cuál de ellos es el mejor. Esto no significa que cada OE deba tener un valor escalar de la función de evaluación (para llevar a cabo una selección proporcional al valor de la función de evaluación), ni siquiera que deba tener un valor explícito, simplemente debe poder determinarse cual es el mejor entre dos. Así, un objeto *selector* puede eliminar el peor y reproducir el mejor (mutándolo o cruzándolo con otros).

1.5.2. Innovaciones introducidas por EO

Las principales diferencias que encontramos entre EO y otros paradigmas de computación evolutiva son [Mere99, Mere00a, Mere00b]:

a) EO es **independiente del paradigma**, lo cual quiere decir que no es un algoritmo genético, ni un programa evolutivo, ni un programa genético, ni un enfriamiento simulado, pero que puede ser cualquiera de ellos y, de hecho, todos esos paradigmas de la computación evolutiva pueden implementarse con EO.

   Por ejemplo, para implementar un AG simple, el objeto mutador debería cambiar, básicamente, el valor de ciertos bits en la cadena; el objeto reproductor aplicaría el procedimiento del operador de cruce; una función de evaluación escalar permitiría la comparación de dos OE y, a partir de éste, se podría aplicar un proceso de selección (ruleta) para reproducir los mejores y eliminar los peores individuos.

b) EO es **independiente de la representación**. EO no necesita información acerca de cómo se representa una solución a un problema, sino simplemente del hecho de que éstas pueden cambiarse y, si es conveniente, combinarse entre sí. De hecho, EO no necesita representar las soluciones de un modo diferente, simplemente codificarlas en el lenguaje orientado a objetos: una solución a un problema es un objeto (perteneciente a cierta clase), que podrá ser mutado, combinado

con otros de su clase y comparado para estimar cuál es el mejor, lo cual es suficiente para hacerlo evolucionar.

c) EO es **independiente del lenguaje de programación**, lo cual quiere decir que no necesita codificar las soluciones como objetos de un lenguaje orientado a objetos en particular, tal y como el paradigma de la Programación Genética hace (representa las soluciones en LISP). Las soluciones se pueden programar en cualquier lenguaje orientado a objetos y hacerlos evolucionar utilizando la biblioteca de EO del mismo lenguaje. De hecho, los OE se pueden programar en diferentes lenguajes de programación y hacerlos evolucionar a la vez, usando modelos de objetos independientes del lenguaje de programación, tal como COM (*Common Object Model*, de Microsoft [Micr00]) o CORBA (*Common Object Request Broker Architecture*, del *Object Management Group* [Henn99, Grou00]).

d) EO es **orientado a objetos desde la base**: no sólo el objeto o la clase del individuo que se hará evolucionar son programados según este paradigma, sino cualquier elemento que intervenga en la evolución (operadores, selectores y los algoritmos mismos).

EO hace especial énfasis en la modularidad a la hora de programar una aplicación evolutiva. Debido a que tanto los operadores como el resto de entidades son objetos, resulta fácil combinar diferentes operadores y objetos de distintas fuentes para construir una aplicación. En la práctica no importa el interior del objeto, sino cómo se pueda acceder a él mediante su interfaz.

EO ha sido el punto de partida para la creación de bibliotecas para Computación Evolutiva escritas en otros lenguajes, pero que conservan la misma filosofía. Actualmente, se trabaja en el desarrollo de una biblioteca escrita en Java, denominada JEO [Doli02][Aren02], e igualmente una biblioteca escrita en Perl, cuyo nombre es OPEAL [Mere02].

1.5.3. Diseño y desarrollo de aplicaciones con EO

La utilización de EO en el desarrollo de un algoritmo evolutivo hace realmente independientes las fases del diseño de las estructuras de datos necesarias y de la posterior implementación del programa que las manipula. Así, EO permite considerar en primer lugar el problema de encontrar una representación adecuada al problema y los operadores genéticos a utilizar, para posteriormente realizar el diseño del algoritmo evolutivo,

utilizando los elementos anteriores del modo en que se haría cualquier otro algoritmo evolutivo.

Durante la **fase de diseño**, normalmente el objeto que se hará evolucionar queda definido por el problema a resolver. Por ejemplo, en el caso del problema del viajante de comercio (TSP, *Travelling Salesman Problem*) se podría utilizar una representación de modo que cada objeto solución sea un vector de enteros que contenga todas las ciudades [Booc94]. Los operadores genéticos utilizados para mutar y recombinar las soluciones deberían tener en cuenta la disposición de las ciudades a visitar. El resto del diseño es, en general, independiente del problema: la selección de los individuos que se reproducirán y el criterio de parada del algoritmo.

En la **fase de programación**, el algoritmo (basado en OE) se programará de forma similar a como se programaría otro algoritmo evolutivo, esto es, según el bucle descrito en la fig. 1.7.

Aunque no hay un soporte teórico para probar el buen funcionamiento de EO, en el peor de los casos funcionaría como una búsqueda aleatoria, pero ya que utiliza recombinación de soluciones, los **bloques constructivos** que producen buenas soluciones se mezclan, con lo cual se alcanzan mejores soluciones.

1. Crear una población inicial de OE (usando factorías o los constructores de cada objeto).
2. Crear una población inicial de OE (usando factorías o los constructores de cada objeto).
3. Repetir hasta que el criterio de parada se cumpla:
   - *a*) Evaluar cada individuo, comparándolos entre sí, para seleccionar los mejores.
   - *b*) Aplicar los operadores genéticos, creando nuevas soluciones (incrementar la diversidad) y combinándolas (decrementar la diversidad).
   - *c*) Sustituir los peores individuos en la población por los recién creados.

Figura 1.7: Estructura de un algoritmo diseñado con EO.

Ya que no se utiliza un alfabeto binario (de hecho no se puede decir que utilice alfabeto alguno), no se pueden aplicar ni el *Teorema de los Esquemas* [Gold89] ni otros resultados teóricos extraídos de las Estrategias Evolutivas [Back91] o Programación Evolutiva [Foge66, Foge95]. En contrapartida, EO lleva la *teoría de los bloques constructivos* [Gold89] un poco más

allá: Mientras que la codificación binaria está respaldada por el *Teorema de los Esquemas* [Holl75, Gold89], la *Teoría del Análisis de Formas* [Radc91] da soporte teórico al uso de alfabetos no binarios. El Análisis de Formas generaliza el Teorema de los Esquemas tratando con la equivalencia de clases de cromosomas en lugar de cromosomas binarios. Un ejemplo es "cromosomas que contienen una permutación en particular" o "cromosomas en los que el segundo componente es el doble que el primero". El Análisis de Formas asegura que si se usan operadores independientes de la representación se pueden definir algoritmos independientes de la representación, que se comportarán de la misma forma que los AG estándar. En EO, puesto que el usuario no está obligado a utilizar una representación en cadenas binaria, y las soluciones a los problemas se codifican de forma *natural*, los bloques que forman las soluciones se encontrarán juntos de forma natural y se heredarán juntos.

1.5.3.1. Individuos de la población

La biblioteca proporciona clases de objetos que actúan como individuos en la población, es decir, objetos evolutivos. Como se ha venido comentando, la biblioteca incluye diversos OE, entre los que se encuentran: cadenas de bits, cadenas de caracteres, vectores de cualquier tipo de dato (en particular, de números enteros o reales) y, por supuesto, RNFBR.

La fig. 1.8 representa la jerarquía de clases de la versión de la biblioteca EO utilizada en el desarrollo de esta tesis (versión 0.8.5) para las poblaciones y los cromosomas, que incluye una clase genérica de cromosomas (**EOBase**, que define la interfaz básica para un EO, incluyendo una función de evaluación y un identificador) y algunos tipos más específicos (**EOBin, EOString, EOVector**). Se pueden obtener OE nuevos, con las características fundamentales ya comentadas, simplemente heredando alguna de las clases ya definidas [Mere99, Mere00a, Mere00b].

1.5.3.2. Operadores genéticos

Los operadores genéticos se encuentran agrupados en dos clases: genéricos y específicos.

Los operadores genéricos se pueden aplicar a muchas clases de OE, sin importar si estos hacen uso de algún tipo de representación. Por ejemplo, un **operador de permutación** se puede aplicar a cualquier objeto de una clase que herede de un vector genérico (**EOVector**), del mismo modo que se puede aplicar un **operador de recombinación**.

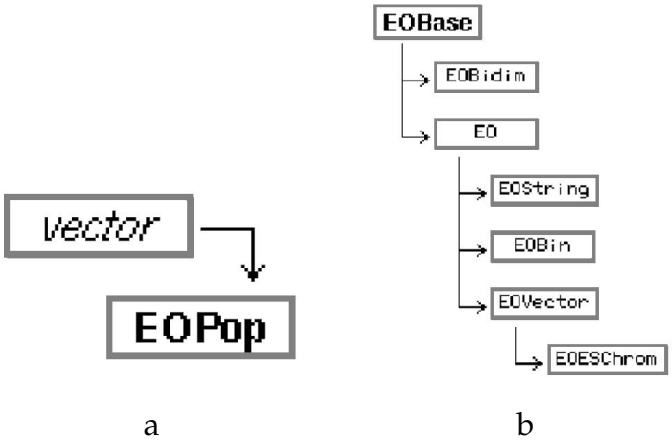


Figura 1.8: Jerarquía de clases de la biblioteca EO para las poblaciones (a) y para los cromosomas (b).

Otros operadores, como el de **mutación** son más específicos del OE a modificar: un objeto mutador de cadenas de bits operará de muy diferente modo a como lo hará un objeto mutador de vectores de números enteros.

Los operadores son objetos cuyo comportamiento es el de una función a la que se pasa el objeto a modificar.

La fig. 1.9 representa la jerarquía de clases de la biblioteca EO para los operadores genéticos, que incluye operadores **unarios** (como el de mutación), **binarios** (como el de cruce), o **n-arios** (aplicados a más de dos OE [Eibe95]). Esas clases definen las características básicas de cada tipo de operador. De ellas descienden los operadores que se utilizarán en las aplicaciones.

El tener los operadores diferenciados de los OE permite trabajar más fácilmente con ellos, ya que al estar definidos como clases independientes de las que se puede heredar, el usuario puede crear sus propios operadores.

1.5.3.3. Operadores de población

Los operadores de población son genéricos y actúan con varios objetos de la población (seleccionando algunos de ellos, creando individuos a partir de otros o sustituyendo algunos individuos por otros).

Las figs. 1.10, 1.11 y 1.12 representan la jerarquía de clases de la biblioteca EO para los operadores de población, que incluye operadores de

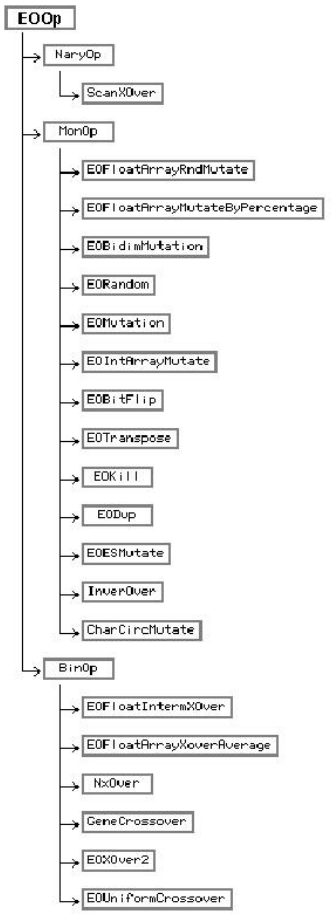


Figura 1.9: Jerarquía de clases de la biblioteca EO para los operadores genéticos.

selección, **reproducción** o **reemplazo**.

Los objetos **EOReplace** (y descendientes) reemplazan, eliminando si es necesario, aquellos individuos de la población peor dotados por los creados con los operadores genéticos.

Un objeto de la clase **EOBreeder** (y descendientes) almacena los operadores genéticos para aplicarlos posteriormente a los individuos seleccionados, creando la descendencia de esa generación. Cada operador recibe como parámetro una prioridad o **tasa de aplicación**. En cada generación, antes de aplicar los operadores genéticos, las tasas de todos los operadores se normalizan dividiéndolas por la tasa total acumulada para obtener la probabilidad de aplicación real de cada operador.

Un **EOSelector** (y descendientes) lleva a cabo la selección de los in-

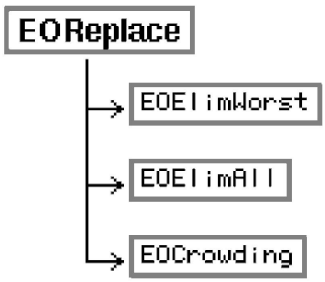


Figura 1.10: Jerarquía de clases de la biblioteca EO para los operadores de reemplazo.

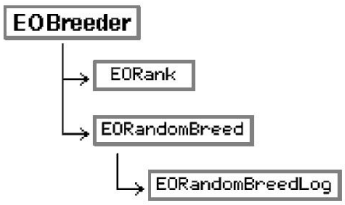


Figura 1.11: Jerarquía de clases de la biblioteca EO para los operadores de reproducción.

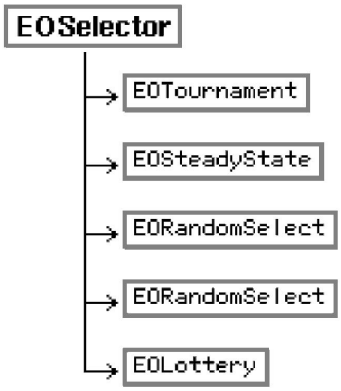


Figura 1.12: Jerarquía de clases de la biblioteca EO para los operadores de selección.

dividuos que se reproducirán aplicándoles los operadores genéticos, de acuerdo a cierta estrategia (torneo, estado estacionario).

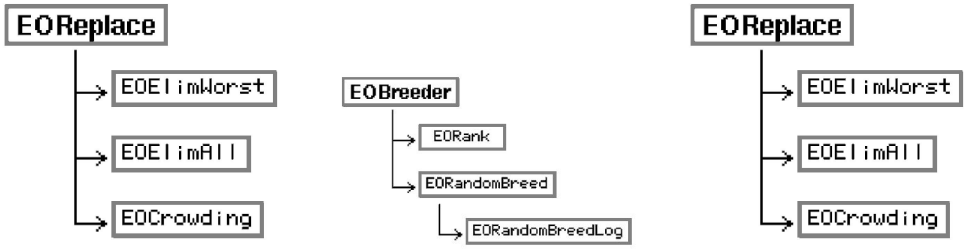


Figura 1.13: Jerarquía de clases de la biblioteca EO para los operadores de reemplazo (a), reproducción (b) y selección (c).

1.5.3.4. Algoritmos

En EO, los algoritmos son tratados también como objetos, de forma que en cualquier momento podemos crear un nuevo objeto de tipo algoritmo, acceder a través de su interfaz a sus componentes internos (objetos población, selector, reproductor, terminador).

Los algoritmos se aplican a una población de OE, haciéndolos evolucionar hasta que se alcanza una condición. Básicamente llevan a cabo el bucle de evaluación, selección, reproducción y substitución.

La implementación actual de la biblioteca incluye los siguientes objetos algoritmo: **EOES** (estrategias de evolución), **EOEasyGA** (un AE simple y flexible), **EOSGA** (el AG de Goldberg [Gold89]) y **EOSA** (enfriamiento simulado).

La fig. 1.14 representa la jerarquía de clases de la biblioteca EO para los algoritmos comentados.

1.6. Conclusiones

En este primer capítulo de la tesis se ha expuesto el problema que se desea resolver: diseñar automáticamente redes neuronales de funciones base radiales (RNFBR) utilizando para ello algoritmos evolutivos (AE). Igualmente, se ha mencionado el método diseñado para realizar esta tarea: -EvRBF, que será descrito y evaluado con mayor detalle en los capítulos 3, 4 y 5.

Antes de entrar a describir en detalle las RNFBR, se ha hecho una introducción al campo de las redes neuronales artificiales, destacando la importancia que tienen tanto el buen diseño de las mismas, como el algorit-

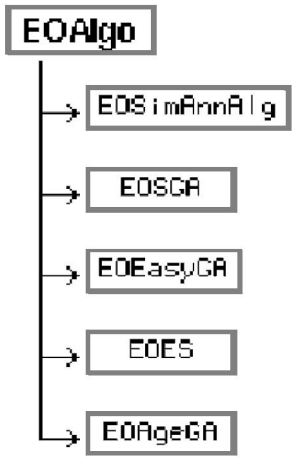


Figura 1.14: Jerarquía de clases de la biblioteca EO para los algoritmos.

mo de aprendizaje usado para optimizar pesos sinápticos o cualquier otro parámetro configurable de la red.

Dado que el método desarrollado, EvRBF, es un algoritmo evolutivo, se ha realizado una introducción al campo de la Computación Evolutiva, incluyendo desde los fundamentos biológicos de la misma hasta consideraciones específicas sobre los distintos AE existentes. Como es propio de los AE, EvRBF está diseñado para buscar las mejores soluciones dentro del conjunto de todas las posibles, es por ello que también se ha revisado el papel que juega la computación evolutiva dentro del campo de los problemas de búsqueda y optimización.

Finalmente, se ha descrito la biblioteca EO que ha servido de base para el desarrollo de EvRBF. La filosofía de esta biblioteca se basa radicalmente en el concepto de objeto evolutivo, entendido éste dentro del campo de los Lenguajes Orientados a Objetos, implementado hoy en día a través de numerosos lenguajes de programación. De esta forma, no sólo el individuo que debe ser evolucionado se representa como un objeto, sino también los distintos operadores genéticos, los mecanismos de selección o reproducción, e incluso los propios algoritmos, lo cual permite desarrollar nuevos objetos de cada uno de estos tipos a partir de las clases ya existentes. Una de las grandes ventajas que aporta esta forma de trabajar es la efectiva distinción entre implementación del objeto e interfaz del mismo, lo cual permite que múltiples algoritmos puedan ser usados con múltiples individuos y que a su vez se puedan aplicar operadores de forma genérica con

independencia de la implementación real de cada objeto.

CAPÍTULO 2 / DISEÑO DE REDES NEURONALES DE FUNCIONES BASE RADIALES

2.1. Introducción

Este capítulo presenta, en primer lugar, la descripción de las RNFBR, en las que se centra esta tesis y, a continuación, una recapitulación de los métodos presentes en la bibliografía que hacen uso de AE para diseñarlas.

Así, el apartado 2.2 comienza describiendo el concepto matemático de función base radial y su utilización en problemas de interpolación y aproximación, origen de las RNFBR y base en la que se sustenta su comportamiento. En los siguientes apartados, se describen igualmente su topología, sus principales características, los métodos no evolutivos existentes para entrenarlas y configurarlas, y las distintas aplicaciones que se les han ido dando. De entre sus características destaca la reseñada en el apartado 2.2.3, que muestra cómo es posible calcular el conjunto de pesos óptimos una vez la estructura de la red ha sido determinada.

Posteriormente, el apartado 2.3 realiza una revisión de los métodos utilizados para hacer evolucionar redes neuronales y, más concretamente, de los métodos evolutivos diseñados para trabajar con RNFBR. Esto es necesario para la posterior presentación y evaluación del método EvRBF (capítulos 3 y 5),

2.2. Redes Neuronales de Funciones Base Radiales

La utilización de Funciones Base Radiales (FBR) como funciones de activación para redes neuronales fue realizada por primera vez por Broomhead y Lowe en 1988 [Broo88]. Desde entonces se les ha prestado una especial atención dado que tienen la característica de ser fácilmente implementables (tanto en lo referente a la estructura de datos que las soportan, como en lo tocante a los algoritmos de aprendizaje y explotación de la red) y por estar demostrado que son aproximadores universales [Ligh92] como lo puedan ser los perceptrones multicapa.

2.2.1. Funciones Base Radiales

Como se describe en [Broo88], las FBR constituyen una técnica matemática para resolver el problema de interpolación estricta en espacios multidimensionales. Este problema queda definido de la siguiente forma:

> *Dado un conjunto de p puntos* $\{x_i; i = 1, 2, \ldots, p\}$ *en* $\mathbb{R}^n$*, conocidos y distintos entre sí, y p números reales* $\{f_i; i = 1, 2, \ldots, p\}$*, se debe encontrar una función* $s : \mathbb{R}^n \to \mathbb{R}$ *que satisfaga las condiciones de interpolación:*

$$s(x_i) = f_i, \quad \forall i = 1, 2, \ldots, p \tag{2.1}$$

Nótese que la función s debe pasar por los puntos conocidos de la función.

La resolución de este problema usando funciones base genéricas consiste en expresar s como una combinación lineal de p funciones, como se indica en la ec. 2.2. Las funciones base, h_i, pueden ser elegidas de forma arbitraria y habitualmente son no lineales. Una vez fijadas tales funciones, la flexibilidad de este modelo, esto es, su capacidad para ajustarse a distintas funciones, estriba en la libertad para elegir diferentes valores para los coeficientes λ_i.

$$s(x) = \sum_{i=1}^{p} \lambda_i h_i(x) \tag{2.2}$$

Uno de los ejemplos más conocidos es el de interpolación mediante una línea recta:

$$s(x) = ax + b \tag{2.3}$$

compuesto por dos funciones base $h_1(x) = x$ y $h_2(x) = 1$, y por los coeficientes $\lambda_1 = a$ y $\lambda_2 = b$. La ventaja de este modelo es que es posible calcular los mejores valores de los coeficientes para un determinado conjunto de datos, sin embargo su flexibilidad es muy reducida.

Las FBR son un tipo concreto de funciones base. Su característica más propia es que la respuesta que producen decrece (o crece) de forma monótona en función de una distancia (normalmente la distancia euclídea) del punto evaluado con respecto a un punto central propio de la función. De esta forma, una FBR queda determinada por los siguientes parámetros:

- **Un punto central**, llamado **centro** de la función, que es un vector de la misma dimensión que el espacio de entradas.
- **Una medida de distancias** entre puntos del espacio de entrada.
- **Un radio** (o conjunto de radios) que permita escalar las distancias de los puntos de entrada con respecto al centro.
- **Una forma explícita** para la función.

La fig. 2.1 muestra alguna de las FBR más comúnmente utilizadas.

Simbólicamente, son varias las formas en las que se puede representar una FBR, tal y como se indica en la siguiente ecuación:

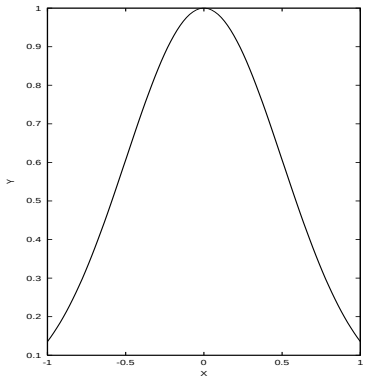
$$\phi(x) \equiv \phi(x, c, r, ||\dots||) \equiv \phi(x, c, r) \equiv \phi\left(\frac{||x - c||}{r}\right) \tag{2.4}$$

donde tanto x como c son puntos del espacio de entrada (es decir, $x, c \in \mathbb{R}^n$), correspondiendo c al punto central de la FBR; por su parte, $||\dots||$ denota una determinada distancia sobre $\mathbb{R}^n$ (normalmente la distancia euclídea, por lo que suele obviarse); finalmente, r es el radio que permite escalar la distancia.

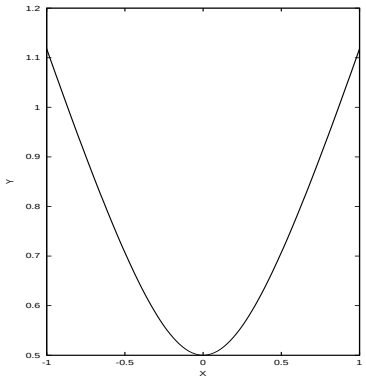
Una vez conocidas las FBR, se define la función de interpolación $s(x)$ como:

$$s(x) = \sum_{i=1}^{m} \lambda_i \phi_i(x, c_i, r_i) \quad x \in \mathbb{R}^n \tag{2.5}$$

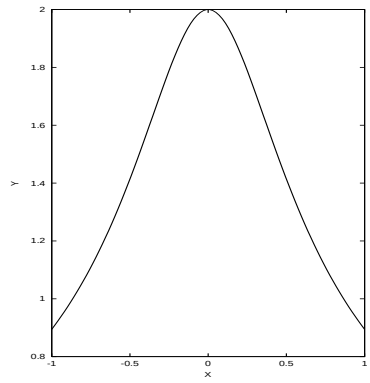
correspondiendo los p puntos centrales c_i con los puntos del espacio de entradas para los cuales se conoce el valor de la función a modelizar, es decir:



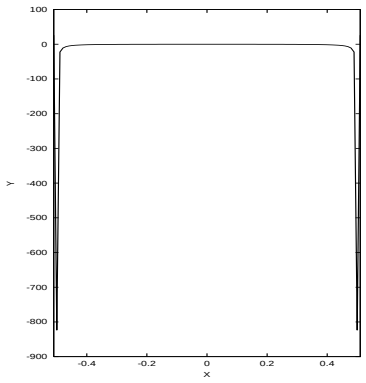
(a) Gaussiana: $e^{\frac{-z^2}{2r^2}}$



(b) Cuadrática: $\sqrt{z^2 + r^2}$



(c) Cuadrática Inversa: $\frac{1}{\sqrt{z^2+r^2}}$



(d) Plato Llano: $\frac{z}{r}^2 log|\frac{z}{r}|$

Figura 2.1: Ejemplos de Funciones Base Radiales (z representa la distancia del punto evaluado al centro, que en este caso es $x = 0$)

$$c_i = x_i \quad ; \forall i = 1, 2, \ldots, p \tag{2.6}$$

Si se introducen las condiciones de interpolación de la ec. 2.1 en la ec. 2.5, obtenemos el siguiente conjunto de ecuaciones lineales para los coeficientes $\{\lambda_i\}$

$$\begin{pmatrix} f_1 \\ \vdots \\ f_p \end{pmatrix} = \begin{pmatrix} A_{11} & \cdots & A_{1p} \\ \vdots & \ddots & \vdots \\ A_{p1} & \cdots & A_{pp} \end{pmatrix} \begin{pmatrix} \lambda_1 \\ \vdots \\ \lambda_p \end{pmatrix} \tag{2.7}$$

cuya expresión en notación matricial es:

$$f = A\lambda \tag{2.8}$$

donde A_{ij} es el valor devuelto por la $j - ésima$ FBR cuando la entrada $i - ésima$ es presentada, esto es:

$$A_{ij} = \phi_j(x_i, c_j, r_j) \quad i, j = 1, 2, \ldots, m \tag{2.9}$$

Dada la existencia de la matriz inversa de A, a partir la ec. 2.8 es posible conocer los valores que han de tomar los coeficientes, pues se pueden calcular los distintos λ_j a través de la expresión:

$$\lambda = A^{-1} f \tag{2.10}$$

Este conjunto de ecuaciones que permiten resolver el problema de la interpolación de $\mathbb{R}^n \rightarrow \mathbb{R}$ puede ser generalizado sin mayor dificultad a problemas de interpolación de $\mathbb{R}^n \rightarrow \mathbb{R}^{n'}$. En este caso, la solución vendría también dada por los coeficientes $\{\lambda_{jk}, j = 1, 2, \ldots, p, k = 1, 2, \ldots, n'\}$, calculados usando nuevamente la ec. 2.10.

No obstante, en la mayoría de los problemas reales no es conveniente utilizar el enfoque de interpolación estricta. Esto es debido a que los puntos conocidos de la función suelen exceder considerablemente el número de grados de libertad necesarios para generar una solución razonablemente buena. La obligación de utilizar tantas FBR como puntos haya conlleva el inconveniente de que el modelo generado es demasiado sensible a las variaciones producidas por datos con ruido o imprecisos.

El modo de eludir esta desventaja del método de interpolación estricta consiste en relajar las condiciones de interpolación de la ec. 2.1. Es decir, no obligar al modelo a devolver el valor exacto de la función en todos los puntos conocidos, sino conformarse con una buena aproximación. Éste es precisamente el enfoque adoptado por las RNFBR, en el cual hay menos FBR que puntos conocidos de la función. Una vez redefinido el problema de esta forma, el cálculo de los valores óptimos para los coeficientes λ_{jk} (que corresponderán a los pesos de red neuronal) se realiza de una forma ligeramente distinta como se verá posteriormente en el apartado 2.2.3.

2.2.2. Topología de una RNFBR

A lo largo de la presente tesis entenderemos por RNFBR una red compuesta por dos capas de neuronas (ver fig. 2.2), donde:

a) La **capa oculta**, contiene neuronas que reciben las entradas procedentes del entorno y les aplican FBR, caracterizadas cada una por su centro, radio y forma específica. Se utilizará la distancia euclídea.

b) La **capa de salida** realiza algún tipo de combinación lineal (normalmente una suma ponderada) de las salidas proporcionadas por la capa anterior añadiendo un factor de **sesgo**.

La fig. 2.2 muestra de forma más detallada esta topología.

Más formalmente, dado que una RNFBR toma un vector de valores de entrada, los cuales procesa hasta obtener un vector de valores de salida, podemos considerarla como una función $s(\vec{x}) : \mathbb{R}^n \rightarrow \mathbb{R}^{n'}$, tal que:

$$s_j(\vec{x}) = \lambda_{0j} + \sum_{i=1}^{p'} \lambda_{ij}\phi_i(\vec{x}, \vec{c}_i, \vec{\vec{r}}_i) \quad ; \; j = 1 \ldots n', s_j \in \mathbb{R}, \vec{x} \in \mathbb{R}^n \qquad (2.11)$$

donde ϕ_i es la FBR de la neurona oculta i; λ_{0j} son los sesgos añadidos a la neurona de salida j; λ_{ij} representa el peso de la conexión existente entre la neurona oculta i y la neurona de salida j; $\vec{c}_i$ son los centros y $\vec{\vec{r}}_i$ los radios de las FBR.

Como se puede observar en la ec. 2.11, el número de neuronas de la capa oculta de la red no es p sino p', siendo generalmente $p' \ll p$. Por consiguiente, dado que habrá menos neuronas que puntos conocidos de la función, los centros de las mismas posiblemente dejarán de coincidir con dichos puntos.

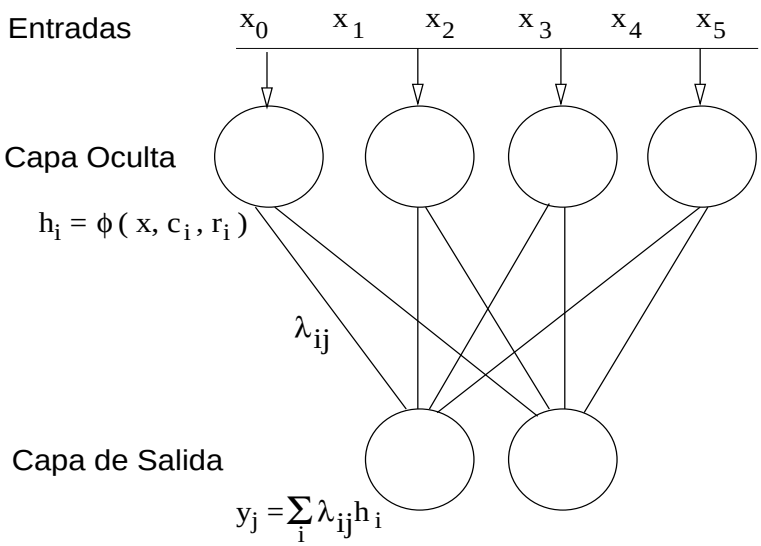


Figura 2.2: Esquema de una red neuronal de funciones base radiales.

Una vez establecido el comportamiento del modelo general, podemos definir con más detalle la función base radial ϕ_i, expresando la ec. 2.4 como:

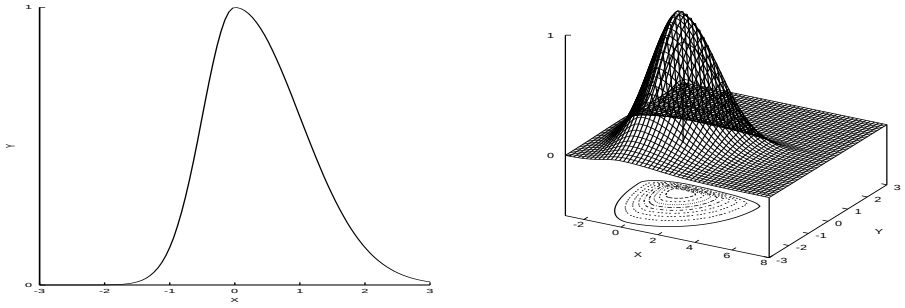
$$\phi(\vec{x}) \equiv \phi(\vec{x}, \vec{c}, \vec{R}(\vec{r}), ||\ldots||) \tag{2.12}$$

donde $\vec{c} = \{c_1, c_2, \ldots, c_n\}$ es el centro (de la misma dimensión que las entradas) y el radio queda definido como un vector de n pares de valores reales: $\vec{r} = \{(r_{1a}, r_{1b}), (r_{2a}, r_{2b}), \ldots, (r_{na}, r_{nb})\}$. La inclusión de estos $2n$ radios permiten, por un lado, que cada dimensión del espacio de entradas pueda ser escalada de forma diferente. Y por el otro, que dentro de una misma dimensión se puedan usar dos radios distintos, uno para los puntos menores o iguales que el centro y otro para los mayores. Es por ello que se define el vector de funciones $\vec{R}$ como:

$$\vec{R} = \{R_i = (x_i \leq c_i)?r_{ia} : r_{ib}\ ;\ \forall i = 1, 2, \ldots, n\} \tag{2.13}$$

Las figs. 2.3(a) y 2.3(b) muestran cómo afecta la utilización de pares de radios cuando se usan FBR gaussianas en espacios de entrada de 1 y 2 dimensiones, respectivamente.

Partiendo de las RNFBR descritas, a las que denominaremos **asimétricas**, es posible aplicar métodos que reducen los grados de libertad con los que opera la red (haciendo menor el número de parámetros a estimar para configurarla). Tales métodos consisten básicamente en considerar algunas



(a) Gaussiana Asimétrica 1D (b) Gaussiana Asimétrica 2D

Figura 2.3: FBR gaussianas asimétricas: utilizan pares de valores para los radios.

políticas más restrictivas en cuanto a los valores de radios a usar. Así, en el caso de **RNFBR parcialmente simétricas**, como la mostrada en la fig. 2.4, tendremos que:

$$R_i = r_{ia} = r_{ib} \quad ; \forall i = 1 \ldots n \tag{2.14}$$

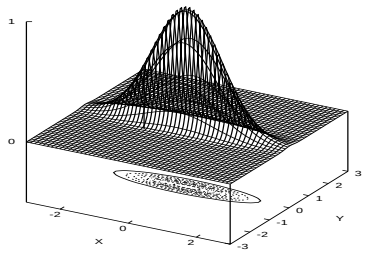
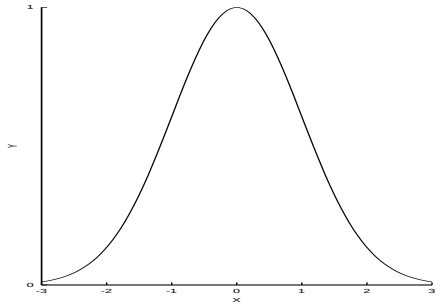
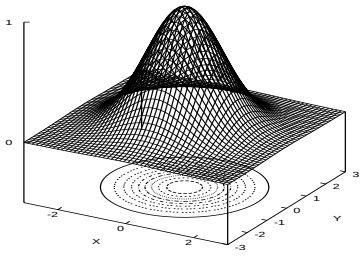


Figura 2.4: FBR gaussiana parcialmente simétrica: utiliza un solo radio para cada dimensión.

Finalmente, es posible restringir aún más la versatilidad de la red utilizando un solo valor, r, para todos los radios. El resultado es una **RNFBR totalmente simétrica**. La ec. 2.15 describe esta situación, de la cual pueden verse muestras en las figs. 2.5(a) y 2.5(b).



(a) Gaussiana Simétrica 1D



(b) Gaussiana Simétrica 2D

Figura 2.5: FBR gaussianas simétricas: utilizan un solo radio para todas las dimensiones.

$$R_i = r_{1a} = r_{1b} = r_{2a} = r_{2b} = \ldots = r_{na} = r_{nb} = r \quad ;\forall i = 1 \ldots n \quad (2.15)$$

La homogeneización propuesta en la ec. 2.15 solo podrá llevarse a cabo cuando el rango de los valores de entrada sea idéntico para cada una de las dimensiones del espacio de entradas, lo cual puede ser cierto por la propia naturaleza del problema a resolver o, en caso contrario, por haberse realizado una normalización de dichos valores de entrada.

Existe una variedad de RNFBR a la que se denomina **redes neuronales de funciones base radiales normalizadas** (RNFBR-N). Se caracterizan por normalizar las salidas proporcionadas por la capa oculta de forma que su suma valga 1. De esta forma, la salida proporcionada por una RNFBR-N viene dada por:

$$s_j(\vec{x}) = \sum_{i=1}^{p'} w_{ij}\phi'_i(\vec{x}, \vec{c}_i, \vec{\vec{r}}_i) \quad j = 1, 2, \ldots, n' \quad (2.16)$$

donde:

$$\phi'_i = \frac{e^{\phi_i}}{\sum_{k=1}^{p'} e^{\phi_k}} \quad (2.17)$$

La distinción entre redes normalizadas y no normalizadas no ha sido especialmente tenida en cuenta por parte de los investigadores. De hecho,

la conveniencia o no de su uso es materia de discusión entre los mismos. Así, por ejemplo, Shorten y Murray-Smith [Shor96] realizaron una serie de experimentos sobre aproximación funcional en los cuales el comportamiento de las redes normalizadas resultaba ser mejor en algunos problemas puntuales. Sin embargo, también describieron algunos de los efectos laterales que conlleva la normalización de RNFBR. En primer lugar, se produce un cambio en la forma de la FBR, cambio que además depende del resto de las FBR de la red, por lo que cada neurona pierde su independencia con respecto a las demás. En segundo lugar, la RNFBR-N cubre totalmente el espacio de entradas, no sólo el definido por el conjunto de valores de entrenamiento. Esto lleva a que el comportamiento de la red sea impredecible cuando se aplica a puntos de una zona no aprendida. Finalmente, incluso el valor central de las FBR puede volverse inútil ya que la neurona de una RNFBR-N puede no alcanzar su máximo (o mínimo) valor en dicho centro.

2.2.3. Cálculo del vector de pesos óptimo

La determinación de los valores para los pesos de las conexiones existentes entre las neuronas de la capa oculta y las de la capa de salida puede hacerse de forma óptima si adaptamos la ec. 2.10 al problema de interpolación no estricta y aplicamos el principio de mínimos cuadrados. En efecto, cuando este principio es usado en aprendizaje supervisado de modelos lineales, tiene la ventaja de llevar a un problema de optimización cuya resolución es particularmente simple.

Para ello, definamos el vector $s(\vec{x}) \in \mathbb{R}^{n'}$ como el conjunto de valores devuelto por una RNFBR cuando el punto x se presenta como entrada. Como se vio en 2.2.2, cada una de las componentes de dicho vector queda definida según:

$$s_j(\vec{x}) = \lambda_{0j} + \sum_{i=1}^{p'} \lambda_{ij} \phi_i(\vec{x}, \vec{c}_i, \vec{r}_i) \tag{2.18}$$

A partir de un conjunto de p pares de valores de entrenamiento $\{(\vec{x}_i, \vec{f}_i)\}$, el principio de mínimos cuadrados nos llevará a minimizar el error calculado como la suma de las diferencias al cuadrado:

$$E = \frac{1}{2} \sum_{i=1}^{p} \sum_{j=1}^{n'} (f_{ij} - s_j(\vec{x}_i))^2 \tag{2.19}$$

Una vez fijados los centros y radios de las neuronas ocultas, la RNFBR se convierte en una combinación lineal de FBR y son los pesos λ_{ij} los que determinan el error cometido. En consecuencia, la solución al problema de minimizar el error cuadrático consistirá en encontrar el conjunto óptimo de pesos que hace esto posible. Utilizando notación matricial, podemos describir el problema mediante la siguiente expresión:

$$\begin{pmatrix} f_{11} & \dots & f_{1n'} \\ \vdots & \vdots & \vdots \\ f_{p1} & \dots & f_{pn'} \end{pmatrix} = \begin{pmatrix} 1 & A_{11} & \dots & A_{1p'} \\ 1 & \vdots & \vdots & \vdots \\ 1 & A_{p1} & \dots & A_{pp'} \end{pmatrix} \begin{pmatrix} \lambda_{01} & \dots & \lambda_{0n'} \\ \vdots & \vdots & \vdots \\ \lambda_{p'1} & \dots & \lambda_{p'n'} \end{pmatrix} \quad (2.20)$$

cuya notación podemos reducir aún más, obteniendo la expresión ya conocida:

$$F = A\lambda \quad (2.21)$$

donde F es el conjunto de valores de salida que se desea obtener, λ es la matriz de pesos desconocidos y A es la denominada **matriz de diseño**. Esta matriz está formada por los valores que devuelven las p' distintas neuronas de la capa oculta cuando les son aplicadas las entradas $\vec{x_1}, \vec{x_2}, \dots, \vec{x_p}$, respectivamente. Nótese que se le ha incorporado una columna con todos sus valores a 1 lo cual permite estimar el valor óptimo de los sesgos de cada neurona de salida al mismo tiempo que se estiman los pesos de las conexiones sinápticas (es decir, cada sesgo puede entenderse como el peso que une cada neurona de salida con una hipotética neurona oculta cuya salida es siempre 1).

De la anterior expresión 2.21 obtenemos que el conjunto de valores óptimos para los coeficientes λ_{ij} viene dado por la siguiente ecuación:

$$\lambda = A^{-1}F \quad (2.22)$$

Si bien el cálculo de A^{-1} puede resultar complejo, al haber reducido nuestro problema a uno de interpolación no estricta, donde $p' << p$, tendremos más datos en el conjunto de entrenamiento que número de pesos a estimar. Por ello, podemos conformarnos con resolver la ecuación:

$$\lambda = A^{-1^*}F \quad (2.23)$$

donde A^{-1^*} no es la matriz inversa, sino la **matriz pseudo-inversa** de la matriz de diseño. Esta matriz puede ser eficientemente calculada mediante

métodos como *Singular Value Decomposition* (SVD), métodos de descenso de gradientes, o utilizando la conocida regla delta:

$$\Delta w_j = \eta g(\vec{x}_i)(f(\vec{x}_i) - y(\vec{x}_i)) \tag{2.24}$$

en la que el factor de aprendizaje, η, y la función $g(\vec{x}_i)$ ponderan la diferencia entre el valor esperado, $f(\vec{x}_i)$, y el valor obtenido por la red, $y(\vec{x}_i)$. Frecuentemente, $g(\vec{x}_i)$ resulta ser la función identidad o la función de activación de las neuronas ocultas, aunque puede ser cualquier función definida sobre el vector de entrada.

En conclusión, una vez fijados los centros y radios de las neuronas de la capa oculta de una RNFBR, se puede obtener de forma eficiente el conjunto de pesos que minimiza el error cuadrático con respecto a un determinado conjunto de entrenamiento. Así, el problema del entrenamiento de RNFBR radica más en la correcta elección de los parámetros que configuran las neuronas ocultas que en el establecimiento de los pesos de las conexiones sinápticas existentes entre la capa oculta y la capa de salida.

2.2.4. Principales propiedades de las RNFBR

Existen ciertas características de las RNFBR que las hacen especialmente interesantes para los investigadores. De entre ellas destacamos:

a) A diferencia de los perceptrones multicapa, las RNFBR trabajan de forma local. Esto hace que se pueda estudiar de forma simple no solo la bondad de la salida proporcionada por la red, sino también el por qué proporciona dicha salida.

b) Una vez que han sido fijados el centro y los radios de cada una de las neuronas de la capa oculta, se pueden hallar de forma analítica los pesos que minimizan el error cuadrático medio calculado con respecto a la diferencia entre las salidas proporcionadas por la red y las salidas esperadas. De hecho, su cálculo se reduce a la resolución de un modelo de ecuaciones lineales, como se vio en el apartado 2.2.3.

c) Bajo ciertas circunstancias, equivalen a controladores difusos y de hecho no es difícil encontrar analogías entre ambos tipos de modelos de control [Reyn96].

2.2.5. Algoritmos de entrenamiento para RNFBR

Atendiendo a Schwenker [Schw01], se puede realizar una clasificación de los distintos métodos no evolutivos que existen para el proceso de entrenamiento de RNFBR. La distinción entre unos métodos y otros se ha realizado en función del número de fases en que se desarrolla cada uno. En todos los casos, se debe partir de un conjunto de entrenamiento compuesto por patrones de entrada para los cuales se conoce el valor de salida.

2.2.5.1. Entrenamiento en una fase

Este método es el más simple de todos. Consiste en utilizar puntos del conjunto de patrones de entrada como centros de las FBR y establecer un único radio a un valor predeterminado, quedando así como únicos parámetros a calcular los pesos de las conexiones. El ejemplo más característico es el de las RNFBR generadas a partir de *Support Vector Machines* (SVM).

Las SVM fueron creadas por Boser, Guyon y Vapnik [Bose92] siendo inicialmente clasificadores binarios. Posteriormente se consiguió relajar las condiciones bajo las que podían utilizarse [Cort95b, Cort95a, Osun97], haciéndolas adecuadas para un mayor número de problemas. Finalmente, los trabajos de Anthony y Bartlett [Anth99] y Shawe-Taylor y Cristianini ([Cris99] proporcionaron el fundamento matemático capaz de explicar el funcionamiento del método.

Básicamente, una SVM define una región de decisión óptima para los elementos de un conjunto de patrones de entrada. La región de decisión queda delimitada por una función que es superposición de funciones locales, expresada en forma de combinación lineal de las mismas. Cada una de estas funciones locales se centra en alguno de los patrones de entrada. El conjunto de puntos centrales a utilizar pueden ser generado de forma eficiente mediante el algoritmo de *Support Vector Learning* (SVL) cuando ciertos parámetros (número de centros a utilizar o cota superior para los coeficientes de la función) son fijados de antemano.

Una de las funciones locales susceptible de ser utilizada es, precisamente, la función gaussiana. Cuando así se hace, la SVM queda reducida a una RNFBR, formada a partir de FBR totalmente simétricas, todas ellas compartiendo un mismo radio para el cual no existe un mecanismo óptimo de elección.

2.2.5.2. Entrenamiento en dos fases

Este modelo de entrenamiento es el más utilizado. Como en el método anterior, se deben fijar en primer lugar los valores para los centros y radios, para determinar, posteriormente, los valores para los pesos. La diferencia con el anterior método estriba en que los centros no se van a corresponder con patrones del conjunto de entrenamiento.

Para el entrenamiento de los centros se pueden utilizar métodos de agrupamiento o *clustering* no supervisados (como K-medias) [Mood89], métodos para cuantización supervisada de vectores (como LVQ) [Koho90, Schw94] e, incluso, entrenamiento mediante árboles de decisión [Kuba98].

La utilización de k-medias permite generar un conjunto de puntos del espacio de entrada representativos de cada una de las regiones en las que está definida la función a estimar o los patrones a clasificar. Tras el correspondiente establecimiento de radios y pesos, la RNFBR proporcionará soluciones aceptables al menos en aquellas regiones en las que existan bastantes patrones de entrenamiento y en los que la función a estimar se comporte de manera similar a la FBR utilizada. No obstante, al ser un mecanismo no supervisado, no tiene en cuenta que valores de entrada cercanos puedan ofrecer salidas muy distintas entre sí, lo cual limita su utilización.

El inconveniente del k-medias es precisamente el que pretende eludirse con LVQ. Este algoritmo trata de seleccionar puntos del espacio de entrada, representativos nuevamente del conjunto total de los patrones disponibles, pero esta vez teniendo en cuenta las distintas regiones del espacio de salida. Como desventaja, el algoritmo no garantiza que el conjunto de centros sea el óptimo para ser usado dentro de una RNFBR, tanto en el número de ellos como en las posiciones que ocupan.

Finalmente, la utilización de árboles de decisión (como C4.5 de Quinlan [Quin92]) permite realizar una partición del espacio de entrada, estableciendo intervalos dentro de cada uno de los ejes del mismo. Así, se generan hipercubos en el espacio de entrada, en cada uno de los cuales el valor de la función a aproximar se halla dentro de unos límites concretos o, en el caso de reconocimiento de patrones, todos los puntos de dicho hipercubo pertenecen a la misma clase. Una vez generados los hipercubos, se establece su centro geométrico como uno de los centros de la RNFBR que se está construyendo. Sus principales inconvenientes radican en el alto número de centros que pueden llegar a generarse y en la dificultad que entraña usar el método para problemas de aproximación funcional y estimación de series temporales.

Por su parte, el proceso de entrenamiento de los radios es también una tarea crítica [Bish95b], pues tanto puede llevar al sobre-entrenamiento de

la red (radios demasiado pequeños) como a hacerla excesivamente suave (radios demasiado grandes). A pesar de ello, la forma en que generalmente se establecen estos radios es mediante heurísticas.

En el caso de redes totalmente simétricas, el único radio existente puede ser establecido como un valor proporcional a la media de las k menores distancias calculadas entre todos los distintos pares de centros diferentes que se puedan establecer. Para el caso de redes parcialmente simétricas o asimétricas, se pueden establecer el radio de cada FBR usando un valor proporcional a la media de las distancias de los k centros más cercanos a su centro. Si se ha realizado la elección mediante agrupamiento, también puede utilizarse la media de las distancias de los patrones que pertenecen al grupo con respecto al centro elegido como representante. En cualquier caso, tanto el valor k como el coeficiente de proporcionalidad aplicado han de ser establecidos manualmente.

Sea cual sea el método utilizado para determinar los centros y radios de la red, el siguiente paso del entrenamiento en dos fases consiste en modificar los pesos de los enlaces, intentando minimizar el error cometido al aproximar los patrones de entrenamiento.

2.2.5.3. Entrenamiento en tres fases

Basándose en las conclusiones de Michie [Mich94] (según las cuales los anteriores métodos de aprendizaje darían lugar a clasificadores con altos niveles de error de generalización), Schwenker [Schw01] y Cohen [Cohe00a] proponen, de forma independiente, extensiones al algoritmo clásico de retro-propagación que permiten actualizar todos los parámetros de la red en una tercera fase.

En ambos algoritmos, se parte de una red neuronal ya definida, en la cual, de forma iterativa, se realizan pequeños incrementos o decrementos en los valores de centros, radios y pesos sinápticos. Para ello, se definen funciones que determinan cómo debe modificarse cada parámetro a partir de la diferencia existente entre la salida obtenida por la red y la deseada. Además de este dato, en tales funciones intervienen tanto derivadas de distinto orden de las FBR utilizadas, como factores de aprendizaje.

Como otros algoritmos de descenso de gradiente, el principal inconveniente del método proviene de la posibilidad de quedar atrapado en mínimos locales en función de la red de partida utilizada. Esto es debido a que el proceso de modificaciones es guiado exclusivamente por las distintas derivadas de la FBR utilizada por cada neurona. Igualmente, el establecimiento del factor o factores de aprendizaje resulta una tarea crítica, como reconocen los propios autores.

2.2.5.4. Entrenamiento híbrido

Finalmente, existen una serie de métodos que modifican simultáneamente tanto los valores de centros, radios y pesos, como el tamaño de la red. Tales métodos pueden ser **incrementales**, cuando se diseñan para establecer los parámetros de la RNFBR a partir de una red inicialmente vacía, o **decrementales**, cuando parten de una red sobre-especificada y van destruyendo neuronas y/o pesos de forma iterativa hasta alcanzar una determinada condición

De entre los métodos incrementales, destaca el de Mínimos Cuadrados Ortogonales (*OLS*, en inglés) [Chen91], basado a su vez en el mecanismo de selección hacia adelante (*forward selection*) que consiste en añadir neuronas que forman un espacio ortonormal hasta que un determinado nivel de error es alcanzado. Una mejora sobre este algoritmo es el mecanismo de Mínimos Cuadrados Ortogonales Regularizado, en el cual se penalizan aquellos pesos excesivamente grandes evitando así que la red se sobreentrene. Orr, en [Orr95b], utiliza nuevamente la selección hacia adelante como método de crecimiento de la red, pero el método que usa para detener dicho crecimiento está basado en el error devuelto por el método de validación cruzada (*cross-validation*), ya sea en su versión de validación cruzada generalizada o en su versión más radical denominada **dejar uno fuera** (*leave-one-out* o *delete-1*). Un método de características similares es el de Redes de Distribución de Recursos (RAN,*Resource Allocating Networks*) [Plat91a] y Crecimiento de Estructuras de Celdas [Frit94]. Ambas técnicas intentan estimar los centros utilizando algoritmos de escalada, lo cual las hace susceptibles de encontrar sólo mínimos locales.

Precisamente, uno de los más recientes estudios acerca de la creación de RNFBR [Sánc02] utiliza un método incremental para detectar cuáles son las claves en la generación automática de RNFBR. El autor parte de una sola neurona oculta, cuyo centro es el primer elemento del conjunto de entrenamiento. Posteriormente, se añaden sucesivas neuronas hasta que la red pierde su capacidad predictiva. El centro de cada nueva neurona es el centroide de un determinado *cluster* calculado usando un algoritmo de agrupamiento, mientras que el radio es la distancia mínima de dicho centro a los demás ya existentes. El cálculo de los pesos sinápticos se realiza mediante SVD cada vez que una neurona es añadida. Usando el citado método para aproximar las funciones de prueba descritas en 2.25 y 2.26, el autor extrae varias conclusiones. En primer lugar, que es necesario elegir centros que compitan entre sí por ser representativos de una determinada zona, intentando no solapar unos a otros su actuación. En segundo lugar, que existe un número de neuronas tras el cual la red aumenta su cobertura

con respecto al conjunto de entrenamiento, pero no estabiliza su poder de predicción; por ello, para trabajos reales propone poner un umbral a dicho error que cuando sea sobrepasado termine la construcción de la red. Por último, indica que el método utilizado depende excesivamente de la primera neurona elegida, por lo que propone usar repetidamente el algoritmo con distintas reordenaciones del conjunto de entrenamiento.

$$f(x,y) = cos(\pi x) sin(\pi \sqrt{|y|^3}) \;\; ; \;\; x \in [0,1] \; , \; y \in [-0.5, 0.5] \tag{2.25}$$

$$f(x) = \sqrt{x} \;\; ; \;\; x \in [0,1] \tag{2.26}$$

Por su parte, entre los métodos decrementales se encuentra el algoritmo de Leonardis y Bischof [Leon98]. En él, se utiliza la medida de Longitud de Descripción Mínima (MDL, *Minimum Description Length*) para decidir qué neuronas deben ser eliminadas. En efecto, según el principio de MDL, de entre todos los posibles conjuntos de neuronas que se puedan construir debe escogerse el que mejor aproxime el conjunto de entrenamiento y que a su vez tenga el menor número de ellas. En general, los métodos decrementales son muy restrictivos, lo cual les hace encontrar redes no óptimas.

A modo de resumen, puede comprobarse que la mayor parte de los estudios se ha concentrado en la estimación del número de FBR que debe tener la red y en los centros que se les deben asignar, sin que apenas haya sido considerada la tarea de estimación del valor de los radios. Caso aparte son los distintos métodos evolutivos, descritos en el apartado 2.3, así como el algoritmo presentado en esta tesis, el cual ha sido diseñado para hacer evolucionar cada uno de los componentes de la RNFBR buscada.

2.2.6. Aplicaciones de RNFBR

A continuación se enumeran algunas de las más representativas o llamativas aplicaciones que se han dado a las RNFBR de entre las presentes en la literatura. Se puede acudir a [Howl01] para ver una selección de aplicaciones más complejas.

2.2.6.1. Aproximación funcional y estimación de series temporales

Dado que el surgimiento de las RNFBR se debe precisamente a la utilización de FBR en problemas de interpolación, no es de extrañar que muchas de las aplicaciones existentes estén relacionadas con la aproximación de funciones, especialmente de aquellas definidas como series temporales.

De entre los primeros investigadores que abordaron el problema de generación automática de RNFBR destacan Carse y Fogarty [Cars96], así como Whitehead y Choate [Whit96]. En primer lugar, Carse y Fogarty hicieron que un algoritmo genético generase las redes, aplicando sus investigaciones a la aproximación de la función $y = sin(20x^2)$ y a la estimación de la serie temporal de Mackey y Glass. Esta misma serie temporal fue estudiada simultáneamente por Whitehead y Choate utilizando un algoritmo de cooperación y competición entre neuronas que pretendía alcanzar la red deseada. De nuevo dicha serie temporal fue utilizada por los investigadores Chng, Chen y Mulgrew [Chng96], aunque modificada para hacerla no lineal y no estacionaria.

Zheng y Billings en [Zhen96] centraron su investigación en el estudio de sistemas aleatorios lineales y en la simulación de un turbo-generador a partir de la función que lo representa.

Orr [Orr93, Orr95a, Orr98] , por su parte, ha dedicado mayor dedicación al estudio analítico de este tipo de redes, utilizando para ello problemas de aproximación de funciones tales como la denominada *H*ermite definida por:

$$y(x) = (1 + x - 2x^2)e^{-x^2} \tag{2.27}$$

así como la función $y(x) = sin(12x)$ o funciones que calculan el valor de impedancia que se obtiene en un circuito de corriente alterna en función de un valor R de resistencia, una frecuencia angular ω, una inductancia L y una capacidad de condensador C.

Dentro del campo de aplicaciones reales, destacar el trabajo de Sheta y De Jong [Shet01] los cuales utilizan RNFBR para la estimación de una serie temporal real correspondiente al valor de cambio entre la libra británica y el dólar estadounidense. E igualmente, los investigadores Tan y Tang [Tan01] que utilizan el método Taguchi (frecuentemente aplicado a la configuración de sistemas difusos) para estimar los pesos de una RNFBR-, aplicando el algoritmo resultante a un problema de control preciso del movimiento de una plataforma. Por último, Cambiaghi, Gadola y Maffezzoni [Camb96] utilizaron las este tipo de redes para la modelización de neumáticos, necesaria para estudio de dinámica de vehículos.

2.2.6.2. Clasificación de patrones

Son también variadas las aplicaciones que se han dado a las RNFBR dentro del campo del reconocimiento de patrones. Así, en el mismo trabajo anteriormente citado de Whitehead y Choate [Whit96] se aplica un

algoritmo de generación automática de RNFBR al conocido problema de clasificación de plantas Iris. Este mismo problema es utilizado por Burdsall y Giraud-Carrier [Burd97] para evaluar su algoritmo GA-RBF. Aplicaciones similares son también las usadas por Zheng y Billings [Zhen96], los cuales aplican su algoritmo al reconocimiento de patrones en los problemas de enfermedades cardiacas (*heart disease*) y concesión de créditos de un banco australiano (*australian credit*), ambos tomados del repositorio de la Universidad de California, Irvine [Blak98].

Olsen [Olse97] utilizó RNFBR para la verificación de la identidad de una persona basándose en la voz. Para ello, hizo uso de un conjunto de RNFBR cada una de las cuales se especializaba en la verificación de un tipo específico de fonema. Posteriormente, Tsujii, Freedman y Mun [Tsuj99] utilizarán lo que denominan RNFBR Guiadas por Tendencias (*Trend-oriented RBFNN*) para la clasificación de micro-calcificaciones en imágenes preprocesadas de mamografías digitales.

Dentro del campo del reconocimiento de imágenes se encuadran también los trabajos de Inoue y Urahama [Inou00], Schwenker, Kestler y Palm [Schw01] y Willis y Myers [Will01]. Los primeros recurrieron a las RNFBR para llevar a cabo el reconocimiento de objetos con independencia de la vista utilizada. Por su parte, Schwenker utilizó RNFBR aplicadas al reconocimiento de dígitos escritos manualmente, al reconocimiento de objetos 3D a partir de fotografías y a la clasificación de electrocardiogramas de alta resolución. Finalmente, Willis y Myers abordaron el problema del reconocimiento de huellas digitales a partir de impresiones de baja calidad o dañadas.

El inconveniente encontrado en la mayor parte de los algoritmos utilizados por los anteriores investigadores estriba en el hecho de que están excesivamente adecuados al tipo de problema que están estudiando, utilizando heurísticas válidas sólo para el problema en cuestión (esto es, uso de factores de aprendizaje, funciones de *fitness* o FBR especialmente adaptados al problema). En general, dichas heurísticas son difícilmente extrapolables a casos generales, por lo que no dan lugar a mecanismos genéricos de diseño automático de RNFBR.

Por otro lado, algunos de los métodos basados en AE, como los propuestos por Whitehead y Choate o Carse y Fogarty, sufren el problema de tener que fijar previamente parámetros tan importantes como el número de neuronas presentes en la capa oculta. Este es precisamente el problema que intenta solventar EvRBF, al permitir que sean determinados, mediante un AE, todos los parámetros de la red.

2.3. Diseño de RNA mediante AE

La utilización de una RNA para resolver un problema requiere establecer una serie de parámetros, que influyen decisivamente en la rapidez del aprendizaje y en la capacidad de aproximación de la red. En el caso de una RNFBR los parámetros buscados son principalmente el número de neuronas presentes en la capa oculta y la configuración de dichas neuronas. Cada neurona se caracteriza por un punto central, un radio de aplicación y una FBR a aplicar. Una vez fijados estos parámetros se pueden determinar analíticamente los pesos de las conexiones entre las neuronas de la capa oculta y las neuronas de la capa de salida.

La búsqueda de una red neuronal que permita resolver un problema dado puede considerarse como un problema de optimización en el sentido de que, de todas las posibles redes que se puedan crear, habrá un grupo de ellas que resuelvan de forma inmejorable el problema. Para las redes neuronales, además del mecanismo de aprendizaje propio de cada una de ellas, se pueden diseñar AE que supongan un mecanismo adicional en el proceso de configuración y adaptación de las mismas.

Las técnicas de búsqueda global, como los AE, exploran amplias zonas del espacio de soluciones para determinar dónde se hallan las más adecuadas. En general son menos eficientes en términos de tiempo y memoria necesaria que las técnicas de búsqueda local encontrando óptimos locales, por lo que muchas ocasiones es adecuado dejar que el AE seleccione soluciones iniciales en buenas áreas del espacio de búsqueda, para posteriormente localizar los óptimos locales en dichas áreas.

Se pueden encontrar numerosos métodos en la bibliografía que combinan AE y RNA de diversas formas para mejorar la calidad de las soluciones encontradas. Por combinar ambos campos de investigación, AE y RNA, estos métodos suelen ser denominados híbridos y también meméticos por los distintos autores. Los principales enfoques que se han usado para llevar a cabo esta tarea van desde la evolución de los pesos de conexión a la evolución de la arquitectura de la red, como se verá a continuación. Sin embargo, el método EvRBF, presentado en esta tesis permite diseñar de forma integral una RNFBR. El diseño incluye el establecimiento de la arquitectura de la red y los valores para los centros y radios de cada una de las neuronas, todo ello optimizado para la resolución de cada problema sobre el cual se aplique el método.

En los siguientes apartados se verá en qué forma pueden ser útiles los AE a la hora de configurar una RNA, para estimar tanto sus pesos como si su topología, y cómo han sido aplicados al diseño de RNFBR.

2.3.1. Evolución de pesos en las conexiones sinápticas

El establecimiento de los pesos asociados a cada una de las conexiones existentes en una RNA se formula como un problema de minimización de una función de error. Tal función puede ser, por ejemplo, el error cuadrático medio calculado a partir de la salida proporcionada por la red y la salida correcta conocida de antemano. Para minimizar dicho error se han diseñado algoritmos de aprendizaje basados, principalmente, en descenso de gradientes, como pueden ser el algoritmo de retro-propagación (*back-propagation*) o los algoritmos de gradientes conjugados. Los métodos de descenso de gradientes tienen como mayor inconveniente el hecho de quedar frecuentemente atrapados en mínimos locales de la función de error, siendo incapaz de encontrar mínimos globales cuando la función de error no es diferenciable.

Los AE, en conjunción con los mecanismos de aprendizaje tradicionales, permiten salvar el problema de los mínimos locales dado que encuentran conjuntos de valores próximos a los valores óptimos deseados y son menos sensibles a las condiciones iniciales. Para ello hay que definir un *fitness* en cuyo cálculo, usualmente, se incluyen factores relativos tanto al error cometido por la red como a la complejidad de la misma. Esto es posible debido a que, al revés que el caso de los métodos basados en descenso de gradientes, el AE no necesita que la función de error sea diferenciable. Una vez seleccionada la función de *fitness*, el éxito del AE consistirá en la correcta elección del esquema de representación de las soluciones, así como de los operadores que se utilizarán. La forma en que se harán evolucionar las redes pasará por decodificar el cromosoma para crear una nueva red, a la cual se le calculará el *fitness* (inversamente proporcional al error cometido por la red y al grado de complejidad que presente) que servirá para continuar con el AE hasta que la condición de parada sea satisfecha.

Uno de los principales problemas que encuentran los AE al tratar de optimizar los pesos es el denominado **problema de permutación** [Hanc92]. La causa de este problema radica en que distintos cromosomas pueden dar lugar a redes equivalentes en el plano funcional. Más concretamente, la utilización de codificación binaria para representar los cromosomas puede dar lugar a redes en las cuales los nodos se ordenan de formas diversas pero que al ser decodificadas resultan equivalentes. El problema de permutación hace que los operadores de recombinación o entrecruzamiento resulten poco efectivos a la hora de producir nuevas buenas soluciones, especialmente cuando se utilizan cromosomas con codificación binaria.

2.3.2. Evolución de arquitecturas de red

El término **arquitectura de la red** comprende por un lado todo lo relativo a la topología de la misma: número de capas, número de neuronas por capa y conectividad entre las distintas neuronas; y por el otro, las funciones de activación (no necesariamente idénticas para todas las neuronas) de cada uno de los nodos de la red.

La arquitectura de la red juega un papel crucial en el comportamiento de la misma. Dada una tarea que deba ser aprendida por las RNA, una red con muy pocas conexiones y nodos lineales puede no ser capaz de llevar la tarea a cabo debido a que sus capacidades son muy limitadas. Por otro lado, una red excesivamente compleja y nodos no lineales puede resultar sobre-entrenada (*overfitting*), de modo que minimice el error con respecto a los datos usados para el aprendizaje, pero sea incapaz de realizar una buena generalización.

El diseño automático de RNA es un campo en el que aún existe un gran trabajo por hacer. En el apartado 2.2.5.4 se hizo referencia a métodos incrementales y decrementales, los cuales tienen el problema de que pueden caer en mínimos locales, y además son considerados como métodos que analizan subconjuntos del espacio de soluciones, pero nunca el espacio completo. De hecho, existen ciertas características de este espacio de búsqueda que convierten a los AE en candidatos perfectos a la hora de localizar estructuras de RNA óptimas. Dichas características son [Mill89]:

a) El espacio de búsqueda es infinitamente grande, dado que no se debe establecer límite al número de posibles nodos y conexiones.

b) El espacio no es diferenciable con respecto a número nodos y conexiones, pues los cambios en tales valores son discretos y a su vez pueden provocar un efecto de discontinuidad en la salida ofrecida por la red.

c) Es un espacio complejo y en el que aparece ruido; dado que la codificación en un cromosoma de una determinada arquitectura impide relacionar directamente a ésta con su comportamiento, hace que surjan efectos de fuerte epistasia y que sea dependiente del método de evaluación utilizado.

d) El espacio de búsqueda es engañoso. Como ya se ha comentado, arquitecturas similares pueden dar lugar a resultados muy distintos con respecto al objetivo buscado.

e) El espacio de búsqueda es multimodal. Al contrario del punto anterior, arquitecturas muy diferentes entre sí pueden tener comportamientos muy parecidos.

Una vez más, el éxito en la evolución de arquitecturas de RNA está íntimamente ligado a la representación elegida para los individuos en el AE, así como a los operadores diseñados para tal efecto. Así, una de las más importantes decisiones a tomar a la hora de implementar el AE consistirá en decidir cuánta información relativa a la arquitectura de las redes quedará especificada en cada cromosoma. Si toda la información de la arquitectura es almacenada, estaremos hablando de una **codificación directa**. Pero si parte de dicha información es generada a partir del cromosoma utilizando sistemas basados en reglas o algoritmos de entrenamiento, estaremos tratando con **codificación indirecta**.

Las principales ventajas de la codificación directa residen en que es fácil de implementar y posee una gran flexibilidad que hace más fácil al AE la aplicación de tantos operadores como se desee diseñar. Entre sus principales desventajas se halla el problema de escalabilidad, dado que cuanto mayores son las redes, mayor coste de almacenamiento conllevan, y más crece se hace el espacio de búsqueda.

Por el contrario, la codificación indirecta carece de los inconvenientes descritos para la codificación directa, pero también de sus ventajas. Así, aunque la codificación indirecta pueda producir RNA con cromosomas que representan la información de forma más compacta, actualmente no parece ser la más adecuada para encontrar una RNA con un buen poder de generalización. Sin embargo, la codificación indirecta es más similar a la utilizada por los seres vivos, según se desprenden de los descubrimientos realizados en el campo de la neurología.

Existen enfoques evolutivos que permiten evolucionar conjuntamente la arquitectura de la red junto a los pesos de las conexiones. En general, el proceso consiste en ajustar los pesos una vez que se ha encontrado una estructura óptima o muy cercana al óptimo. El principal problema que surge al evolucionar arquitecturas sin hacer evolucionar simultáneamente los pesos se ha denominado **evaluación de fitness ruidoso**, que hace referencia al hecho de que se establezca el *fitness* de una RNA basándose sólo en su estructura, sin usar información relativa al comportamiento de RNA en la tarea de aproximación o clasificación para la que se esté usando. La solución a este problema consiste en entrenar cada una de las arquitecturas de red obtenida por el AE. Además, para minimizar el efecto del factor aleatoriedad, cada arquitectura deberá ser entrenada varias veces, utilizando pesos iniciales aleatorios diferentes, de modo que un valor promediado

de los errores devueltos por la red sea asignado como fitness de dicha red. Este mecanismo de asignación de *fitness*, basado en la capacidad de predicción de la red, es precisamente el usado por el algoritmo presentado en esta tesis.

2.3.3. Diseño de RNFBR mediante AE

Dentro del campo del uso de AE para la evolución de RNFBR, podemos encontrar las aplicaciones más conocidas en los trabajos de Carse y Fogarty [Cars96] y de Whitehead y Choate [Whit96].

En el algoritmo de Carse y Fogarty [Cars96], la estructura de la RNFBR era generada mediante el uso de un conjunto operadores, adaptados a su vez a partir una aplicación previa en la que se hacía evolucionar reglas difusas. Cada cromosoma representaban una RNFBR y se componía de pares de coordenadas *centro-radio*, viniendo impuesto como parámetro externo el máximo número de neuronas a usar. Algunos de los resultados mostrados en el capítulo 5 hacen referencia a los experimentos realizados por estos investigadores.

En cuanto a Whitehead y Choate [Whit96], el método utilizado era muy diferente. En este caso sólo una red era evolucionada por el algoritmo, siendo los individuos del AE las neuronas de la capa oculta de dicha red. Por este motivo, utilizaron un método cooperativo-competitivo para estimar los centros y radios de cada neuronas de la capa oculta. La idea subyacente era conseguir que, cada neurona se especializara en un área determinada del espacio de entradas. Para ello, cada neurona debería cooperar con el resto intentando realizar una buena estimación en aquellas entradas que le correspondiesen. Adicionalmente, las neuronas debían competir entre sí por el espacio, de forma que las neuronas que peor estimación realizasen eran desechadas en favor de las demás.

El principal problema al que estos investigadores se enfrentaron consistió en encontrar la forma de asignar a cada neurona un *fitness* representativo de su bondad. La elección final que realizaron, tras estudiar distintas alternativas, es la representada en la siguiente ecuación:

$$F_i = |w_i|^\beta / E(w_{i'}^2)^\beta \tag{2.28}$$

donde F_i denota el fitness asignado a la neurona i, w_i es el peso de la conexión de la neurona i con la neurona de salida, β es un parámetro del algoritmo determinado empíricamente y $E()$ es el valor medio calculado sobre los pesos $w_{i'}$, que a su vez son los pesos que tendría la red si se utilizaran RBF normalizadas.

En cuanto a los aspectos técnicos del AE, la codificación de cada neurona se correspondía con una cadena de bits. Dicha cadena representaba, en primer lugar, el valor del radio para dicha neurona y, a continuación, un árbol de profundidad 2^n en el cual se codificaban las n coordenadas del punto central.

Los operadores usados eran los normales de mutación y recombinación binaria, así como un operador para "reptar"que descodificaba cadenas binarias a su correspondiente valor real, perturbaba dicho valor y volvía a codificarlo al sistema binario.

El más claro inconveniente del método está en que la solución final depende de un conjunto de parámetros que deben ser fijados de forma certera en función del problema que se esté considerando. Entre dichos parámetros se encuentra el número de neuronas, el cual debe ser dado a priori.

Un trabajo posterior es el realizado por Burdsall y Giraud-Carrier [Burd97], en el que se presenta el algoritmo **GA-RBF**. En él, los autores utilizan un AG para determinar los centros de las distintas RNFBR y lo aplican a la resolución de problemas de clasificación. Cada cromosoma está compuesto por una sucesión variable de números reales que representan las coordenadas de los centros, e incorpora operadores para añadir, eliminar o intercambiar dichos valores. Para evitar el sobre-entrenamiento, las redes son penalizadas en función de la cantidad de neuronas ocultas que tengan. Además, cada individuo está capacitado para reproducirse un número concreto de veces, desapareciendo cuando ha agotado su cupo. Entre sus ventajas destacan la sencillez de sus operadores, la facilidad de cálculo de la función de *fitness* y la rapidez con la que el algoritmo puede converger hacia una solución válida. Entre sus inconvenientes, el hecho de que sólo trate de optimizar los centros, que tanto la función de penalización como la que determina la cantidad de reproducciones permitidas dependan de constantes cuyo valor no se conoce; y por último, que se establezca un límite superior para el número de centros por clase que puede haber en cada red, y que los autores han fijado en 7.

Sheta y De Jong [Shet01], en 2001, utilizaron nuevamente un AG para encontrar centros, radios y pesos de una RNFBR destinada a predecir valores de una serie temporal real, tomada del campo de la Economía. El AG elegido fue el estándar, es decir, uso de cadenas de bits para representar los cromosomas y los operadores de entrecruzamiento y mutación definidos por defecto. Cada cromosoma representaba los p' valores de centro, radio y peso consecutivamente, siendo p' el número de neuronas. El método es rápido, parte de centros, pesos y radios aleatorios y obtiene buenos resultados, aunque utiliza un número preestablecido de neuronas.

Las más recientes aplicaciones tienen que ver con los trabajos de González et al. [Gonz01b, Gonz01a] así como Rivera et al. [Rive02]. El primero parte de un mecanismo de agrupamiento o *clustering* mejorado para la inicialización de los centros de las redes, A continuación hace uso de una interesante colección de operadores, principalmente de mutación, así como de un algoritmo de búsqueda local, los cuales permiten de forma eficiente estimar diversos valores de la configuración de la red. Todo ello es utilizado en un algoritmo multiobjetivo (esto es, tratando de optimizar número de parámetros de la red y error obtenido por la misma), cuya condición de parada está basada en la convergencia de los individuos de la población, en vez de en un determinado número de generaciones. El segundo ofrece un nuevo punto de vista al problema de la construcción de RNFBR. Así, muchas neuronas cooperan, compiten y son modificadas utilizando *evolución difusa* para obtener una red final que resuelve el problema considerado. El término *evolución difusa* proviene del hecho de que se haya utilizado una tabla compuesta de reglas difusas que permite determinar qué operador debe ser aplicado a una determinada neurona para poder mejorar su comportamiento. Algunos de los resultados mostrados en el capítulo 5 están relacionados con el trabajo de estos investigadores.

2.4. Conclusiones

En este capítulo se ha presentado, en primer lugar, el concepto matemático de función base radial y cómo puede ser utilizado para resolver problemas de aproximación funcional. Posteriormente, se han descrito las RNFBR, redes con una sola capa oculta cuya salida es calculada por una FBR caracterizada por un punto del espacio de entradas (el centro) y por uno o más valores de escalado (los radios).

De entre las propiedades de estas redes se ha destacado el que exista un mecanismo preciso para determinar los valores óptimos de los pesos de las conexiones sinápticas, siempre y cuando se hayan determinado el tamaño de la red y los centros y radios de las FBR. Esta propiedad es utilizada por el algoritmo EvRBF, presentado en esta tesis, para generar estructuras de red que pueden ser eficientemente entrenadas y evaluadas por su capacidad de predicción.

Por su facilidad de implementación y uso y por ser aproximadores universales, las RNFBR han sido utilizadas en todo tipo de aplicaciones, relacionadas fundamentalmente con aproximación funcional, estimación de series temporales y clasificación de patrones. Algunas de estas aplicaciones, la más representativas y citadas en la bibliografía relacionada han sido

expuestas a lo largo de este capítulo.

Por otro lado, desde el mismo momento en que fueron ideadas, sus autores promovieron la construcción de algoritmos que automáticamente diseñaran RNFBR. De esta forma, han aparecido numerosos métodos destinados a entrenar y configurar este tipo de redes. Los más significativos, de entre los que no utilizan AE, han sido revisados en el apartado 2.2.5. Igualmente, se han descrito mecanismos para selección de centros y radios; así como métodos que tratan de establecer simultáneamente el tamaño de la red y sus pesos sinápticos bien partiendo de una red mínima a la que añaden neuronas o, en el caso contrario, partiendo de una red sobre-especificada a la que progresivamente se van quitando neuronas.

Por último, se han descrito los métodos basados en la misma filosofía que EvRBF, esto es, AE que permiten evolucionar pesos, arquitecturas o ambos a la vez (apartado 2.3). Las diferencias más notables entre ellos se basan, primeramente, en si usan poblaciones de redes o poblaciones de neuronas para trabajar. En segundo lugar, si intentan diseñar la red de forma parcial o en su totalidad. Y, finalmente, en el tipo de operadores utilizados y funciones de *fitness*, que dependen en gran medida de los factores anteriores.

CAPÍTULO 3 /
EL MÉTODO EVRBF

3.1. Introducción

En los capítulos anteriores se presentaban las distintas herramientas que forman las base de este trabajo. En este capítulo se describe de forma detallada el nuevo método **EvRBF**, desarrollado para el diseño de redes neuronales de funciones base radiales asimétricas mediante algoritmos evolutivos.

Dentro del este capítulo, el apartado 3.2 describe los operadores evolutivos específicamente desarrollados para EvRBF. Se han distinguido los operadores de recombinación o entrecruzamiento (apartado 3.2.1) y los de mutación (apartado 3.2.2). A continuación, en el apartado 3.3, se describe el esquema general del algoritmo EvRBF, para dar paso a una exposición detallada de cada uno de sus componentes en el apartado 3.4. El capítulo acaba con un apartado de conclusiones (3.5).

3.2. Operadores evolutivos

EvRBF se encuentra dentro del paradigma general de AE descrito en la fig. 1.5 (pág. 24). Es pues un algoritmo iterativo en el cual se sucede la creación y evaluación de generaciones de individuos, de forma que en la última de ellas estarán presentes los mejores individuos encontrados en todo el proceso, uno de los cuales será considerado la solución al problema que se esté estudiando. EvRBF es un algoritmo elitista, de estado estacionario y que mantiene una población de individuos de tamaño constante,

si bien el tamaño de los individuos es variable.

La característica más destacada de EvRBF es que opera sobre estructuras de datos que representan directamente RNFBR, sin recurrir a artificiosas representaciones de las mismas. De esta forma, se ha independizado el algoritmo de la implementación específica de la red, pudiendo utilizarse con cualquier representación siempre y cuando se respete el diseño de la interfaz de la misma. Esta característica ya está presente en trabajos relacionados con otro tipo de redes, como perceptrones multicapa [Cast99].

Una consecuencia de lo anterior es que EvRBF incorpora **operadores evolutivos** diseñados para trabajar expresamente con RNFBR. De esta forma, dado que la tarea de EvRBF es encontrar la RNFBR que mejor resuelve un problema, los operadores se han diseñado para tratar de recorrer el espacio de búsqueda de la forma más efectiva posible, minimizando el número de generaciones a ejecutar y maximizando la probabilidad de que la búsqueda sea fructuosa. Así, parte de los operadores que se han generado permiten al algoritmo escapar de óptimos locales, al hacerlo explorar soluciones del problema diferentes entre sí. Adicionalmente, el resto de los operadores permiten refinar las soluciones encontradas en generaciones anteriores, para lo cual desarrollan su actividad en entornos cercanos a los de las soluciones ya encontradas.

Una característica común a todos los operadores es que generan un único individuo al ser aplicados. Si bien esto es normal en los habitualmente llamados operadores de mutación, no lo es tanto en los operadores de entrecruzamiento o recombinación, en los cuales se suelen utilizar n individuos para generar otros n nuevos.

3.2.1. Operadores de recombinación (*crossover*)

La recombinación aplicada a evolución de redes neuronales suele ser de poca efectividad [Yao99], salvo en el caso de redes como RNFBR, cuyas neuronas trabajan de forma local, es decir, que realizan un aprendizaje competitivo y responden a una zona específica del espacio de entrada. Sin embargo, también en estas redes la efectividad de la recombinación puede verse afectada por el número de neuronas e incluso por la representación elegida para el individuo. Así por ejemplo, las representaciones binarias permiten una rápida implementación de este tipo de operadores, pero suelen acarrear el inconveniente de entremezclar los bloques constructivos destinados al establecimiento de los centros con los destinados al establecimiento de los radios o los pesos sinápticos. La codificación directa de RNFBR elegida para EvRBF posibilita que durante la recombina-

ción se realice intercambio de neuronas que aproximan correctamente una determinada zona de la función estudiada. Así se irán generando bloques básicos de neuronas que podrán ser compartidos por toda la población a medida que se sucedan las generaciones.

Se han diseñado tres operadores de recombinación, X_FIX, X_MULTI y X_AVERAGE, que se describen a continuación.

3.2.1.1. Recombinación de secuencias de neuronas: X_FIX

X_FIX substituye una secuencia de neuronas de la capa oculta de una red R_1 por una secuencia de igual tamaño de neuronas de la capa oculta de una red R_2.

Como muestra la fig. 3.1, X_FIX permite el intercambio de información entre las redes sin afectar al tamaño de la capa oculta de la red R_1 a la que es aplicado.

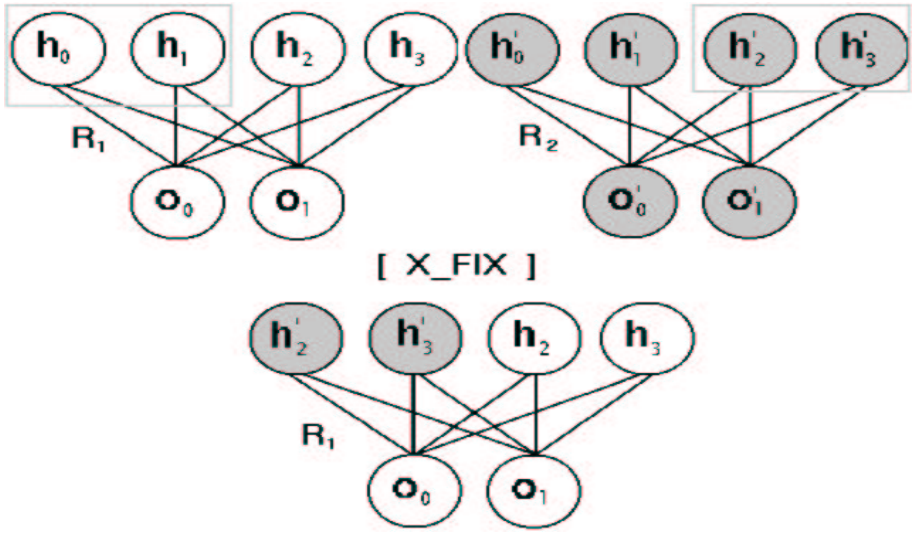


Figura 3.1: Aplicación del operador X_FIX: se produce intercambio de neuronas sin que se modifique el tamaño de la red resultante, R_1.

El propósito que persigue X_FIX es que se produzca intercambio de información entre las redes para llegar a obtener conjuntos de neuronas que funcionen como los bloques constructivos de la solución final.

3.2.1.2. Recombinación multipunto: X_MULTI

Este operador reemplaza con probabilidad p_{x_multi} cada neurona oculta de una red R_1 por una neurona escogida al azar de otra red R_2. Al finalizar, la red R_1 contendrá nueva información conservando el mismo número de neuronas que tenía al principio, como puede verse de forma gráfica en la fig. 3.2. Este tipo de recombinación correspondería con el entrecruzamiento uniforme en el caso de un AG con cromosomas binarios.

La utilización de este operador permite de nuevo el intercambio masivo de información entre las distintas redes, cuya capacidad de generalización será alta al ser EvRBF un algoritmo elitista. Para su correcto uso se precisa, no obstante, estudiar cuál es el valor más adecuado para la **probabilidad de aplicación interna,** p_{x_multi}, tarea que se ha llevado a cabo conforme se detalla en el apartado 4.8.2 (pág. 103).

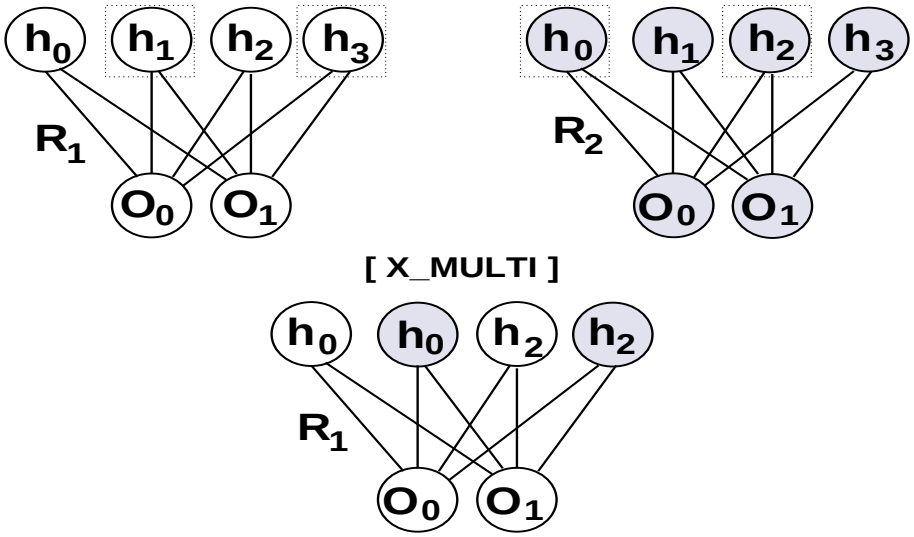


Figura 3.2: Aplicación del operador X_MULTI: neuronas de R_1 escogidas al azar se substituyen por neuronas de R_2, sin cambiar el tamaño de la red resultante, R_1.

3.2.1.3. Recombinación promediada: X_AVERAGE

El operador X_AVERAGE precisa nuevamente de la selección de dos RNFBR al azar R_1y R_2, aunque realiza cambios exclusivamente en la capa de neuronas ocultas de la primera de ellas.

El operador actúa de forma que cada neurona de R_1 tiene probabilidad igual a $p_{x\_average}$ de ser seleccionada (sobre la determinación de este valor,

ver apartado 4.8.3, pág. 104). Para cada neurona seleccionada, se escoge al azar una neurona de R_2 y se establecen tanto los centros como los radios de la neurona de R_1 al valor medio, calculado usando los centros de ambas neuronas, como describen las ecuaciones 3.1 y 3.2.

$$c_{1i} = \frac{(c_{1i} + c_{2i})}{2} \tag{3.1}$$

$$r_{1i} = \frac{(r_{1i} + r_{2i})}{2} \tag{3.2}$$

donde c_{1i} es el valor de la componente $i - ésima$ del punto central de la neurona escogida de la red R_1, c_{2i} corresponde al valor de dicha componente en la neurona procedente de R_2; y análogamente, r_{1i} es el valor de la componente $i - ésima$ del vector de radios de la neurona de la red R_1, correspondiendo r_{2i} al mismo concepto en la neurona R_2. La fig. 3.3 muestra gráficamente el proceso.

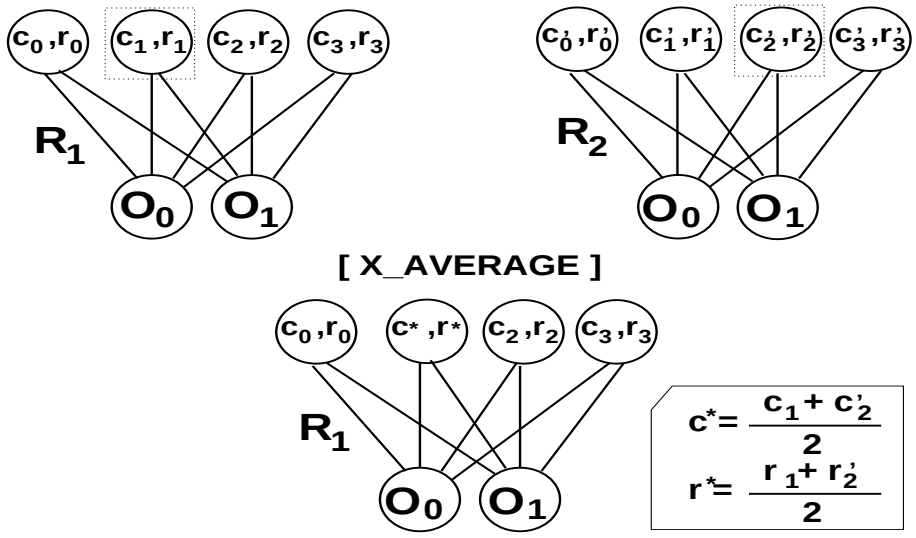


Figura 3.3: Aplicación del operador X_AVERAGE: se establecen los valores de centro (c_{1i}) y radio (r_{1i}) neuronas de R_1 elegidas aleatoriamente de R_1 a valores promediados con centros y radios de neuronas de R_2.

El uso de este operador posibilita la aparición de una neurona que sintetiza la función de las dos que intervienen en la operación. Si la nueva neurona aproxima valores con la misma o mejor eficacia que las anteriores por separado, el proceso de selección natural, junto con el intercambio de neuronas producido mediante recombinación, terminarán por primar a las redes que contengan la nueva neurona sobre las ya existentes.

3.2.2. Operadores de mutación

EvRBF ha sido dotado de un conjunto de operadores unarios de mutación cuya función primordial es realizar una mejor exploración aleatoria del espacio de búsqueda. De esta forma, el algoritmo reduce su dependencia de la población inicial de RNFBR, a la vez que escapa de los posibles óptimos locales que se puedan alcanzar durante su ejecución.

La pretensión de configurar todos los componentes de las RNFBR ha obligado a incluir operadores que actúen sobre los centros, los radios y la cantidad de neuronas que compondrán la capa oculta.

3.2.3. Mutación del valor de los centros: C_TUNER y C_RANDOM

La aplicación de estos operadores permite modificar el punto en que está centrada cada FBR de las neuronas ocultas de la red seleccionada, $R1_1$. La cantidad de neuronas afectadas por cada operador queda determinada por las respectivas probabilidades de aplicación interna: p_{c_tuner} y p_{c_random}, cuya estimación se describe en el apartado 4.8.4 (pág.106).

La diferencia entre ambos operadores consiste en que la modificación realizada puede introducir mayor o menor variedad, en función de si se realiza una simple perturbación del valor actual o de si se cambia éste por un valor elegido aleatoriamente. Así, C_TUNER realiza una perturbación del valor actual c_i utilizando una función de probabilidad gaussiana centrada en el propio c_i y de amplitud igual al radio de la FBR de la neurona, r_i. El nuevo valor tendrá una alta probabilidad de estar en un entorno cercano a c_i lo cual permite afinar la calidad de las soluciones generadas por EvRBF.

Por su parte, C_RANDOM pretende realizar una exploración más amplia del espacio de soluciones al substituir el centro de la neurona por un nuevo centro totalmente aleatorio. Cada uno de los componentes del nuevo centro se elige siguiendo una distribución de probabilidad uniforme en el rango $[min_i, max_i]$, siendo ambos límites el mínimo y máximo valores, respectivamente, de la $i - ésima$ dimensión del espacio de entrada. Los distintos min_i y max_i pueden ser calculados a partir de los patrones de entrada que componen los conjuntos de entrenamiento y test. La elección aleatoria de estos valores trata de evitar la caída en regiones de óptimos locales.

3.2.4. Mutación del valor de los radios: R_TUNER y R_RANDOM

De manera análoga al funcionamiento visto en el apartado anterior para los centros, los operadores R_TUNER y R_RANDOM trabajan modificando los valores de radio de las neuronas ocultas.

R_TUNER modifica el vector de radios de las neuronas afectadas seleccionando valores en el entorno de los ya existentes. Para ello aplica sendas funciones de probabilidad gaussiana centradas en los valores actuales, r_{ia} y r_{ib}, y cuya amplitud es igual a la de la dimensión correspondiente en el espacio de entrada. R_RANDOM opera de igual forma pero asignando un valor aleatorio según una función de probabilidad uniforme.

La decisión de qué neuronas se verán afectadas y cuáles no, se basa de nuevo en la utilización de factores de probabilidad de aplicación interna, p_{r_tuner} y p_{r_random}, respectivamente. El modo en que se han sido determinados estos dos parámetros queda recogido en el apartado 4.8.5 (110.)

3.2.4.1. Incrementador de neuronas: ADDER

La búsqueda de una determinada RNFBR llevada a cabo por EvRBF supone también el establecimiento de la topología más adecuada para dicha red, o lo que es igual, la especificación del número de neuronas que la red ha de tener en su capa oculta.

Para tal fin, se han incluido operadores como ADDER, que añade una nueva neurona a la capa oculta. Los valores para el centro y el vector de radios de la nueva neurona son establecidos aleatoriamente, siempre dentro del rango permitido para cada una de las dimensiones del espacio de entrada, esto es, $[min_i, max_i]$, calculados sendos límites de forma análoga a como se indicó en el apartado 3.2.3.

Posteriormente, el mecanismo de selección de EvRBF debe hacer que si la neurona resultante produce buenos resultados tenga una alta probabilidad de permanecer en las siguientes generaciones, pudiendo así llegar a ser parte de la solución definitiva. Si por el contrario su efecto es perjudicial para la red, ésta perderá oportunidades de reproducirse, impidiendo que el error se propague a próximas generaciones.

3.2.4.2. Decrementadores de neuronas: DEL_MANY y DEL_CLOSEST

Si el anterior operador añadía nuevas neuronas, la aplicación de cualquiera de estos operadores conlleva la desaparición de neuronas de la capa oculta de la red R_1 a la que son aplicados.

El primer operador, DEL_MANY, puede eliminar un número indeterminado de neuronas, ya que es aplicado a cada una de ellas con probabilidad p_{del_many}. El apartado 4.8.7 (pág. 114)recoge el estudio realizado para determinar el valor más adecuado de este parámetro.

Por su parte, el operador DEL_CLOSEST realiza la eliminación de una sola neurona, si bien ésta no es elegida al azar. En lugar de ello, se calculan las distancias euclídeas entre todos los centros de las FBR de las neuronas ocultas. A continuación, se determinan cuáles de ellas se hallan más próximas entre sí para proceder, finalmente, a destruir una de ellas aleatoriamente.

La inclusión de estos dos operadores persigue un doble objetivo. El primero es realizar una reducción efectiva de la complejidad de la red sin que se produzca pérdida en su capacidad de aproximar el conjunto de datos de entrenamiento. El segundo es impedir que las redes puedan llegar a sobreentrenarse, dado que se desea que la solución final posea alta capacidad de generalización, aún cuando el conjunto usado para el entrenamiento no sea perfectamente estimado.

3.3. Esquema general del algoritmo EvRBF

1. Cargar conjuntos de datos de entrenamiento, validación y test.
2. Crear población inicial de individuos, evaluar y asignar un *fitness* a cada individuo
3. Inicializar operadores, comprobadores de condición de parada y monitores de estado.
4. Inicializar algoritmo evolutivo con los componentes anteriormente inicializados
5. Ejecutar el algoritmo evolutivo mientras no se dé la condición de parada, esto es:
   - *a*) Seleccionar individuos de la población actual y crear copias de ellos.
   - *b*) Aplicar operadores a tales copias y evaluarlas asignándoles un *fitness*
   - *c*) Sustituir los peores individuos de la población actual por las copias.
6. Entrenar todos los individuos de la última generación usando los conjuntos de datos de entrenamiento y validación.
7. Obtener el error de generalización de cada individuo de la última generación utilizando el conjunto de datos de test.

Figura 3.4: Estructura general del algoritmo EvRBF.

El algoritmo genérico que describe EvRBF combina elementos del paradigma de computación evolutiva con el del entrenamiento clásico de RNFBR, tal y como refleja la fig. 3.4.

EvRBF es un algoritmo de **estado estacionario**, en los que los peores individuos son reemplazados. Por otro lado, también es un algoritmo **elitista**, pues aunque el mejor individuo encontrado pueda no ser seleccionado para el proceso de reproducción, siempre se incorpora a la siguiente generación inalterado.

3.4. Componentes del algoritmo EvRBF

Una vez vista la estructura genérica del algoritmo, se describe en los siguientes apartados la implementación concreta de los elementos que lo componen, tanto de aquellos propios de EvRBF como de los comunes al resto de AE.

3.4.1. Conjuntos de entrenamiento, validación y test

De forma genérica, la utilización de RNFBR precisa de la existencia de un **conjunto de entrenamiento** (usado para establecer el valor de los pesos sinápticos) y de un **conjunto de test** (que permite valorar la capacidad de generalización de la red). Sin embargo, la introducción del AE en el proceso de configuración y entrenamiento de la red hace necesario el contar con un **conjunto de validación** que permita etiquetar a cada red con un valor de *fitness*.

Los conjuntos de entrenamiento y test se usan para extraer, a partir de ellos, información relativa a las dimensiones de los espacios de entrada y salida (o la cardinalidad del conjunto de clases en problemas de clasificación) y el rango de valores para cada una de dichas dimensiones. Estos datos serán utilizados para diseñar la estructura básica de cada red y para guiar la actuación algunos de los operadores evolutivos ya descritos, como los de mutación de centros y radios.

El algoritmo se ha diseñado para trabajar con patrones de entrada numéricos de cualquier magnitud. Sin embargo, atendiendo a [Sarl98] en referencia a redes del tipo RNFBR, es aconsejable que tales patrones sean normalizados y reescalados. La normalización consigue que los valores de cada dimensión sigan una distribución de media igual a 0 y desviación estándar igual a 1. El reescalado, por su parte, transporta dichos valores al rango $[0, 1]$. De esta forma se evita que aquellas dimensiones cuyos rangos son mayores pesen más al calcular las distancias que se usarán en las FBR.

El proceso de normalización de las entradas se ha llevado a cabo según indica en la siguiente ecuación:

$$x_i^* = \frac{x_i - \bar{x}}{\sigma} \tag{3.3}$$

donde x_i^* corresponde al nuevo valor generado, x_i es el valor real presente en el conjunto de patrones, $\bar{x}$ es la media de dichos valores, σ es la desviación estándar y N es el número total de patrones disponibles.

El proceso de reescalado es igualmente fácil de llevar a cabo y se realiza de la siguiente forma:

$$x_i^* = \frac{x_i - min_i}{max_i - min_i}(max_e - min_e) + min_e \tag{3.4}$$

donde max_e y min_e son los valores superior e inferior, respectivamente, del rango al que queremos escalar, max_i es el máximo valor encontrado para la $i - ésima$ variable de entrada, y min_i es el mínimo correspondiente.

3.4.2. La población

El método EvRBF se encuadra dentro del grupo de algoritmos evolutivos que trabajan con una única población de individuos. A lo largo de la ejecución del algoritmo, el tamaño de la población permanece constante y en cada generación siempre está presente el mejor individuo que se haya encontrado hasta ese momento.

3.4.3. El individuo

EvRBF considera a cada RNFBR como un individuo, por tanto, la población es un conjunto de ellas y la solución al problema buscado viene dada por un determinado individuo, sin necesidad de componer una red superior a partir de varios individuos.

Dado que se utiliza representación directa, cada individuo está compuesto por una capa oculta de neuronas, por una capa de neuronas de salida y por un conjunto de enlaces que relacionan neuronas de una y otra capa. Para cada neurona de la capa oculta se almacena un vector de valores correspondientes al punto en que está centrada dicha neurona, un vector de igual tamaño correspondiente a los radios que se aplican a cada una de las dimensiones del vector punto central y una referencia al tipo de FBR que implementa dicha neurona.

La filosofía de evolución de objetos utilizada en la biblioteca EO ha permitido hacer independiente el algoritmo de la estructura de datos expresa

que se utiliza para almacenar cada individuo, siempre que se respete la interfaz de clase. Así, cada individuo en EvRBF proporciona métodos que permiten a los operadores genéticos acceder y modificar cualquiera de los valores almacenados en el mismo a través de su interfaz.

Más explícitamente, cada individuo incorpora métodos para:

a) Crear, copiar y asignar nuevos individuos.

b) Destruir el individuo.

c) Acceder y modificar cada una de las neuronas de la capa oculta.

d) Crear o destruir neuronas de la capa oculta.

e) Acceder y modificar los valores del vector de pesos de las conexiones entre capa oculta y capa de salida.

f) Obtener un valor de salida ante un patrón que se presenta de entrada a la red.

g) Obtener una descripción de cada uno de los valores almacenados por la red en forma de caracteres de salida.

h) Consultar y establecer el *fitness* asignado al individuo.

Cada individuo en EvRBF es creado bien a partir de un proceso de inicialización de poblaciones, como se describe en el próximo apartado, bien como copia de un individuo ya existente; copia a la cual, posteriormente, se aplicarán operadores genéticos que la harán diferenciarse de su progenitor.

3.4.4. Métodos de inicialización de poblaciones

En este trabajo de tesis doctoral se han considerado y evaluado distintos métodos para llevar a cabo la inicialización de la primera población en cada una de las ejecuciones del algoritmo. Todos los métodos propuestos tienen en común su dependencia, en mayor o menor grado, de los datos presentes en el conjunto de patrones de entrenamiento que se proporciona al algoritmo.

Cualquiera que sea el método de inicialización, el proceso de generación de la población inicial consiste en la creación uno a uno de los individuos que la van a integrar. Por ello, los distintos métodos que se describirán a continuación indican varias de las formas que existen para crear un nuevo individuo.

3.4.4.1. Elección del número de neuronas máximo para la capa oculta

El proceso de creación del individuo comienza con la elección del número máximo de neuronas que formarán su capa oculta. Este dato será sólo un límite para crear la primera población de individuos, ya que en las siguientes no se establece ningún tamaño superior y los operadores que modifican el tamaño de esta capa tienen capacidad para aumentarla o disminuirla libremente.

Condicionados por la variedad de problemas a los que EvRBF será aplicado, se ha considerado establecer este número de neuronas inicial a un determinado porcentaje del tamaño del conjunto de entrenamiento (cuyo valor más adecuado se presenta en el apartado 4.4 (pág. 93). Una vez delimitada esta cota superior, cada RNFBR es creada con un número aleatorio de neuronas que van desde 1 al máximo permitido, inclusive. Se intenta de esta forma generar una población inicial con la mayor diversidad posible.

El problema de la determinación del número de neuronas no se presenta en la capa de salida ya que viene determinado por la dimensión del propio espacio de salida considerado en el problema a resolver. Así, en el caso de aplicar el método a problemas de aproximación funcional o predicción de series temporales, el número de neuronas es directamente igual a la dimensión del espacio de salida. Por otro lado, si el problema es de clasificación de patrones, se genera una neurona de salida por cada una de las distintas clases existentes.

Dado que los pesos de conexión entre capa oculta y capa de salida serán determinados de forma analítica, el proceso de inicialización de individuos se limita a establecer el valor de los centros, radios y FBR de las neuronas de la capa oculta, una vez conocido el número de neuronas que cada individuo tendrá.

3.4.4.2. Selección de los centros de las neuronas

Se han implementado y evaluado (ver 4.3) distintos métodos de inicialización de los centros de las neuronas. En primer lugar, disponemos del método basado en valores aleatorios, que denominaremos **IC_RND**. En él, a partir de los valores máximos y mínimos encontrados para cada una de las dimensiones del espacio de entrada se establecen valores aleatorios para los puntos centrales de las neuronas.

El segundo de los métodos frecuentemente usados para diseñar RNFBR-consiste en seleccionar patrones del conjunto de entrenamiento y colocarlos como centros de las neuronas de la capa oculta. Denominaremos a éste método **IC_PAT**.

Finalmente, un tercer método para establecer los centros de las neuronas lo proporcionan los algoritmos de agrupamiento o *clustering*, tales como el conocido *k-medias*. Es por ello que este método se notará por **IC_KM**.

Como se ve, en todos los casos existe dependencia del conjunto de centros con respecto al conjunto de patrones de entrenamiento. Por ello, para obtener medidas significativas de la bondad del método EvRBF es necesario ejecutarlo varias veces para un mismo problema y, en aquellos casos en que sea posible, hacerlo partiendo de conjuntos de entrenamiento distintos.

3.4.4.3. Selección de los radios de las neuronas

Se han evaluado igualmente distintas posibilidades en cuanto al establecimiento de los valores de los radios . En primer lugar se puede especificar un valor aleatorio constante, **IR_RND_CTE**. Consiste en elegir un único valor, R, de forma aleatoria y utilizarlo para todos los radios y para todas las dimensiones del espacio de entrada. Para obtener R se utiliza una función de probabilidad uniforme en el rango $[0, max]$, donde max es la máxima de las amplitudes de los rangos de las variables de entrada.

Una segunda opción consistirá en establecer valores aleatorios variables, **IR_RND_VAR**, generados a partir de los rangos proporcionados por los patrones del conjunto de entrenamiento para cada una de las dimensiones del espacio de entrada. Las tercera y cuarta opciones, que se denominarán **IR_MIN_DIS** e **IR_MIN_DIS2** , consisten en establecer el radio de cada neurona como la mínima distancia no nula existente al resto de neuronas multiplicada por una factor de escalado, y todo ello con respecto a cada una de las dimensiones del espacio de entrada. La diferencia estará en el factor de escalado usado en una y otra, que será 1 en el caso de **IR_MIN_DIS** (como se indica en [Sánc02]) y de 1.75 para **IR_MIN_DIS2** (como se indica en [Howl01]).

En el apartado 4.5 (pág. 94) se describe detalladamente el conjunto de experimentos llevados a cabo para determinar qué métodos de inicialización de poblaciones e individuos son los más adecuados para utilizar.

3.4.5. La función de evaluación

El valor de *fitness* asignado a cada RNFBR se ha calculado utilizando principalmente una medida de la bondad de la red cuando es utilizada para predecir las salidas asociadas a puntos de entrada que no han sido utilizados en el proceso de entrenamiento. Es decir, el tamaño de la red no

es tenido en cuenta para asignar el valor de *fitness* de la misma. No obstante, en el caso comparación de redes (como ocurre en la selección mediante torneo), si dos de ellas tuvieran exactamente el mismo *fitness* sería elegida aquella cuyo tamaño fuese menor. La probabilidad de que haya de recurrirse a esta comparación de tamaños es prácticamente nula en problemas de aproximación funcional y estimación de series temporales, y algo más significativa cuando se alcanzan las últimas generaciones del AE en problemas de clasificación de patrones.

Cualquiera que sea la estrategia adoptada para evaluar la capacidad de generalización de la red, siempre comienza por una etapa de entrenamiento de la red, que da lugar a una posterior etapa de explotación de la misma. Como consecuencia de lo indicado en el apartado 2.2.3 (pág. 48), la fase de entrenamiento consistirá en aplicar el algoritmo de *Singular Value Decomposition* para obtener el vector de pesos y sesgos óptimos para el conjunto de entrenamiento utilizado.

Dentro de las posibilidades que aporta la utilización de un conjunto de entrenamiento para asignar el *fitness*, se han estudiado las dos más habituales: separación de conjuntos de entrenamiento y validación, y uso de particiones múltiples.

A continuación se describen ambos métodos. En el apartado 4.5 se detalla el estudio realizado para determinar cuál de ellos es más adecuado para su uso con EvRBF.

3.4.5.1. Conjuntos de entrenamiento y validación: EF_TRN_VAL

Este método es el más popular, por ser el más simple de implementar y más rápido que el de particiones múltiples en cuanto a su ejecución. Para poder aplicarlo se debe disponer de dos conjuntos disjuntos de pares entrada-salida, el primero denominado **conjunto de entrenamiento** y el segundo **conjunto de validación**. El proceso de evaluación de la red comienza por entrenarla usando el conjunto de entrenamiento. A continuación, si estamos tratando un problema de aproximación funcional o estimación de series temporales, se presentan a la red los patrones del conjunto de validación y se computa el *fitness* como la inversa de la raíz cuadrada del error cuadrático medio calculado sobre las salidas conocidas, es decir:

$$fitness = \left(\sqrt{\frac{\sum_{i=1}^{m} (f_i - y(x_i))^2}{m}} \right)^{-1} \tag{3.5}$$

donde x_i, es el $i - ésimo$ patrón de entrada, m es el número de patrones en el conjunto de validación, f_i es la salida que deseamos obtener, e $y(x_i)$ es

la salida ofrecida por la red cuando se le presenta dicho patrón.

La utilización de la inversa del error en vez del error mismo permite que el algoritmo evolutivo pueda trabajar con un problema de maximización y hace que las diferencias entre los posibles valores de *fitness* sean más significativas.

En el caso de que el problema que se desee resolver mediante EvRBF sea de clasificación, el método EF_TRN_VAL es similar excepto en lo concerniente al cálculo del *fitness*. Para este problema, EvRBF genera redes con tantas neuronas de salida como posibles valores tenga la variable de salida (esto es, las posibles clases a las que pueda pertenecer cada patrón). Dado un vector de entradas, la salida de la red será el número de la neurona que ofrezca un valor de salida más alto. De esta forma, el *fitness* asignado a cada red será el porcentaje de patrones del conjunto de validación correctamente clasificados. Dicho porcentaje se calcula con respecto al número total de patrones que componen el conjunto de validación.

3.4.5.2. Particiones múltiples: EF_PARTS

Este segundo método está diseñado para evitar el posible sesgo introducido al usar una determinada división de los patrones en conjuntos de entrenamiento y validación[1].

Su funcionamiento parte de un único conjunto de entrenamiento que contiene todos los patrones disponibles para la fase de entrenamiento y validación de la red. Este conjunto es particionado en k conjuntos disjuntos de tamaño similar. Una vez obtenidas las particiones, la red se entrena y valida k veces. En cada una de ellas se utilizan $k-1$ conjuntos para entrenamiento y el conjunto restante para validación. El valor de *fitness* final, para el caso de problemas de aproximación funcional o estimación de series temporales, se establece a la inversa del valor promedio de las raíces cuadradas de los ECM calculados en las k evaluaciones, como se indica en la siguiente ecuación:

$$fitness = \left(\frac{\sum_{j=1}^{k} \sqrt{\frac{\sum_{i=1}^{m_j} (f_{ji} - y(x_{ji}))^2}{m_j}}}{k} \right)^{-1} \tag{3.6}$$

[1]La inclusión de este método en EvRBF está motivada, principalmente, por la conversación mantenida por el autor de esta tesis con algunos de los asistentes al congreso IWANN'2001, en el que se expuso el trabajo [Riva01], y a los que, desde estas líneas, quisiera expresar mi agradecimiento.

siendo:

$$\sum_{j=1}^{k} m_j = m \tag{3.7}$$

donde k es el número de particiones, m_j es el número de patrones presente en la partición $j - ésima$, f_{ji} es el valor de salida esperado para la $i - ésima$ entrada en dicha partición, x_{ji} es tal entrada y, finalmente, $y(x_{ji})$ es el valor obtenido por la red evaluada cuando la procesa.

Al igual que para EF_TRN_VAL, si el problema al que nos enfrentamos es de clasificación, el valor de *fitness* para el método EF_PARTS se calculará como la media de los distintos porcentajes de patrones correctamente clasificados para cada una de las k particiones de los patrones realizada.

Nótese que el valor de k puede calcularse fácilmente como el mayor entero menor o igual a la raíz cuadrada del tamaño del conjunto de patrones de evaluación.

Un caso extremo de este método es el denominado **dejar-uno-fuera**, que consiste en dividir los m patrones en un conjunto de entrenamiento de tamaño $m - 1$ y un conjunto de validación de tamaño 1 y promediar el error obtenido en las m posibles formas de realizar tal particionamiento.

El apartado 4.5 recoge el estudio llevado a cabo para determinar qué método de evaluación resulta el más adecuado para ser usado con EvRBF.

3.4.6. Condición de parada y monitores de estado

Dentro de los componentes que forman EvRBF, dos de los más simples de implementar han sido, por un lado, un conjunto de objetos que permiten determinar cuándo se debe detener el proceso de búsqueda iterativo; por el otro, objetos que permiten extraer información conforme el algoritmo se ejecuta.

3.4.6.1. Condición de parada

La biblioteca EO, utilizada para implementar EvRBF, permite especificar cualquier condición de parada para el AE que se pueda desear e incluso combinaciones lógicas de varias de ellas. EvRBF permite utilizar, en primer lugar, el **número de generaciones** como condición de parada. Este es el método más frecuentemente utilizado en la literatura por lo que es el que se ha usado a la hora de evaluar el método y compararlo con otros ya existentes, como se verá en el capítulo 5.

Adicionalmente, se han incorporado tres métodos más que permiten establecer la condición de parada. Así, se puede especificar un determinado **umbral de error** y el algoritmo deberá estar en ejecución hasta que dicho umbral sea alcanzado, independientemente de cuántas generaciones le tome esta tarea. Por otro lado, otra condición de parada consistiría en especificar el **máximo número de generaciones que se van a realizar sin que se haya encontrado un individuo mejor**. Por último, es posible utilizar una combinación lógica de las anteriores, especificando un número máximo de generaciones, un umbral de error y un número máximo de generaciones sucesivas sin mejora del *fitness*, de modo que el algoritmo deje de ejecutarse en cuanto una o más de las condiciones de parada se cumplan.

3.4.6.2. Monitores de estado

EvRBF es capaz de proporcionar datos acerca del proceso de ejecución y elaborar estadísticas a partir de dichos datos utilizando para ello los objetos monitores incorporados en la biblioteca EO, así como nuevos monitores desarrollados expresamente a partir de dicha biblioteca.

Los monitores utilizados en EvRBF son los siguientes:

a) **Monitor de número de generación**. Muestra el número de generación en el que se encuentra el algoritmo evolutivo.

b) **Monitores de mejor *fitness*, peor *fitness* y *fitness* promedio**. Muestran, respectivamente, el mejor valor de *fitness* de los individuos de la población actual, el peor de ellos y el valor promedio de cada generación.

c) **Monitores de tamaño**. Informan del tamaño del mejor individuo (aquel con mejor *fitness*), el menor y mayor tamaño encontrados y el tamaño promedio de todos los individuos de la población actual.

El algoritmo está diseñado de forma que estos monitores se activan cada vez que se comprueba si se ha alcanzado o no la condición de parada. Como queda descrito en el esquema de la fig. 3.4, esta comprobación se realiza antes de empezar una nueva iteración del bucle, es decir, antes de seleccionar individuos de la generación actual para generar una nueva población.

3.4.7. Operadores aplicados a poblaciones

Se han utilizado tres operadores aplicados a poblaciones en el diseño de EvRBF : el operador de **selección** de individuos para ser reproducidos, el operador de **reproducción** y el operador de **substitución**.

3.4.7.1. Operador de selección

Está diseñado para seleccionar un conjunto de individuos de entre los de una población evaluada, es decir, una población en la que todos los individuos tienen un valor de *fitness* asociado.

De los distintos métodos existentes en computación evolutiva, el utilizado en EvRBF es el denominado **torneo**, que se lleva a cabo con un número constante de individuos, cuyo valor ha sido establecido conforme a lo que se indica en el apartado 4.7 (pág. 98).

La elección de este operador concreto se ha basado en la posibilidad que ofrece a individuos con relativo mal *fitness* para que pasen a formar parte de una nueva población, para que así, mediante los operadores aplicados a individuos, pueda dar lugar a una buena solución al problema.

El operador de selección se aplica en cada generación un número de veces igual al número de individuos a reemplazar en la población actual. Se generan así nuevos individuos que dan lugar a una subpoblación que denominaremos **subpoblación de individuos seleccionados**. Cada uno de los individuos de esta subpoblación es copia exacta de alguno de los individuos de la población actual que, por tanto, permanece inalterada.

3.4.7.2. Operador de reproducción

Su tarea consiste en aplicar un operador genético a cada uno de los componentes de la subpoblación de individuos seleccionados, creando así redes que dejan de ser copias exactas de sus padres.

Cada aplicación del operador de reproducción consiste en aplicar el algoritmo de la fig. 3.5 sobre el conjunto de individuos seleccionados.

Como puede observarse en el segundo paso del algoritmo 3.5, se aplica un solo operador a cada individuo y se usa el mecanismo de la ruleta para elegir el operador a aplicar en cada momento. Así pues, cada operador genético debe llevar asociado un valor de **tasa de aplicación** que determina con qué probabilidad, P_v, será elegido frente a los demás, conforme se indica en la siguiente ecuación:

1. Elegir secuencialmente un individuo de la subpoblación de seleccionados
2. Seleccionar, mediante el mecanismo de la ruleta, un operador a aplicar a dicho individuo
3. En caso de que el operador requiera de otro individuo para ser aplicado, seleccionar aleatoriamente uno distinto del actual.
4. Aplicar el operador al individuo, que quedará de esta forma modificado.

Figura 3.5: Aplicación del operador de reproducción.

$$P_v = \frac{Tasa_v}{\sum_u Tasa_u} \quad ; \quad \forall u \in Operadores \tag{3.8}$$

donde P_v es la probabilidad de que el operador v sea aplicado, $Tasa_v$ es la tasa de aplicación asignada a tal operador y $Operadores$ representa el conjunto de operadores existentes. El valor de las distintas tasas de aplicación ha sido objeto de estudio en el apartado 4.8.

3.4.7.3. Operador de substitución

Por las características de EvRBF, la aplicación de este operador simplemente elimina de la población actual un determinado número de individuos, en concreto aquellos que tienen peor *fitness*, e inserta en la población la subpoblación de individuos seleccionados y modificados tras haberle aplicado los anteriores operadores genéticos. Como resultado, tras aplicar este último operador el tamaño de la población vuelve a ser igual al de la población inicial.

3.5. Conclusiones

En el presente capítulo se ha descrito el algoritmo EvRBF, un algoritmo evolutivo capaz de trabajar con RNFBR y hacerlas evolucionar tanto en su arquitectura como en la información que almacenan.

De esta forma, se han descrito cada uno de los componentes del algoritmo: operadores genéticos, operadores de población, condiciones de parada, función de evaluación y mecanismos de inicialización de poblaciones. Se ha prestado especial atención a la descripción y justificación de los distintos operadores genéticos. Tales operadores han sido específicamente diseñados para trabajar con RNFBR y constituyen la principal dife-

rencia de EvRBF con respecto a otros enfoques de creación automática de este tipo de redes.

CAPÍTULO 4 / ESTIMACIÓN DE PARÁMETROS PARA EL MÉTODO EVRBF

4.1. Introducción

El anterior capítulo presentaba los elementos de que consta el algoritmo EvRBF. Sin embargo, su efectividad pasa obligatoriamente por seleccionar algún método concreto de inicialización de poblaciones y determinar el valor de algunos parámetros con los que debe ser ejecutado. En este capítulo se describen los experimentos sistemáticos que se han realizado para determinar el valor más apropiado para cada uno de dichos parámetros de ejecución. Tanto la metodología seguida para realizar los experimentos como los resultados obtenidos están descritos a lo largo de sus distintos apartados.

La necesidad de estimar estos parámetros contrasta con algunos ejemplos de algoritmos evolutivos que se pueden hallar en la literatura, denominados por sus autores **libres de parámetros** [Mats02, Sawa98] y en torno a los que existe cierta polémica. Se argumenta por parte de sus detractores que se trata de algoritmos con parámetros y decisiones fijadas de antemano (tipo de operadores, forma de aplicarlos, etcétera) por lo que en realidad no se podrían considerar como libres de parámetros, sino como algoritmos que se muestran útiles para una amplia gama de problemas sin necesidad de cambiar algunos de sus parámetros.

Las distintas divisiones que se han establecido en este capítulo analizan los experimentos realizados para determinar el valor más adecuado de cada uno de los componentes vistos en el capítulo anterior. Así, la sección

4.3 se centra en la estimación del método de inicialización de poblaciones más adecuado para usar con EvRBF. Posteriormente, la sección 4.4 estudia la determinación del límite de neuronas iniciales que se debe utilizar. A continuación, en 4.5, se prueban las funciones de evaluación presentadas en 3.4.5. Igualmente, las secciones 4.6 y 4.7 describen, respectivamente, los experimentos y conclusiones relativos al porcentaje de individuos que deben ser reemplazados en cada nueva generación, así como el número de ellos que deben ser comparados al seleccionar un individuo para reproducir mediante el mecanismo de torneo. Finalmente, se determinan las tasas de aplicación y probabilidades de aplicación internas de los distintos operadores en 4.8 y se concluye con la sección 4.9, en la que se recogen los valores estimados para todos estos parámetros, que a su vez serán utilizados en el siguiente capítulo para la evaluación del método.

4.2. Función utilizada para la determinación de los parámetros

El problema utilizado para evaluar las distintas configuraciones de parámetros de EvRBF está tomado de [Sánc02], uno de los más recientes trabajos que abordan el tema del diseño automático de RNFBR. En concreto, se trata de aproximar la función definida en la ecuación:

$$f(x,y) = \cos(\pi x)\,\sin(\pi\sqrt{|y|^3}) \tag{4.1}$$

Como se puede comprobar, es una función definida de $\mathbb{R}^2$ en $\mathbb{R}$, para la que se han generado sendos conjuntos de entrenamiento y test compuestos por 121 y 100 patrones, respectivamente. Los valores para estos patrones de entrada y sus respectivas salidas se han calculado según las siguientes ecuaciones:

$$x_i^{Tr} = a_x + id_x, \quad i = 0,1,\ldots,N_x - 1 \tag{4.2}$$
$$y_j^{Tr} = a_y + jd_y, \quad j = 0,1,\ldots,N_y - 1 \tag{4.3}$$
$$z_k^{Tr} = f(x_i^{Tr}, y_j^{Tr}), \qquad k = N_x i + j \tag{4.4}$$
$$x_l^{Te} = a_x + (l + 0.5)d_x, \quad l = 0,1,\ldots,N_x - 2 \tag{4.5}$$
$$y_m^{Te} = a_y + (m + 0.5)d_y, \quad m = 0,1,\ldots,N_y - 2 \tag{4.6}$$
$$z_n^{Tr} = f(x_m^{Tr}, y_m^{Tr}), \quad n = (N_x - 1)l + m \tag{4.7}$$

donde x_i^{Tr} e y_j^{Tr} corresponden a los variables de entrada del conjunto de entrenamiento, x_l^{Te} e y_m^{Te} corresponden a las del conjunto de test, z_k^{Tr} e z_n^{Te} son las respectivas salidas, $N_x = N_y = 11$ y $d_x = d_y = 0.1$. La fig. 4.1 muestra ambos conjuntos de forma gráfica.

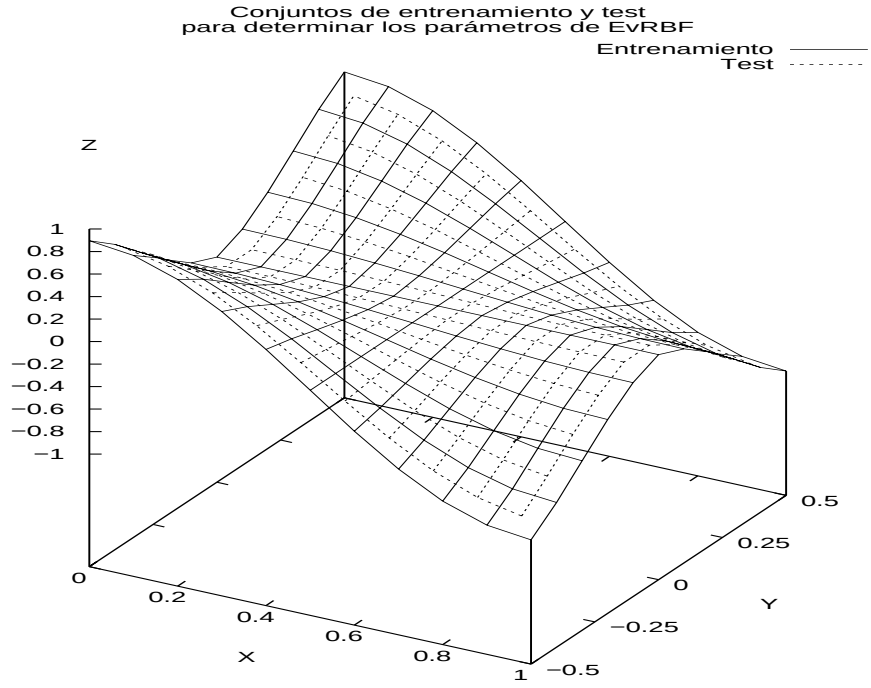


Figura 4.1: Representación de los conjuntos de entrenamiento y test usados en la determinación de los parámetros de ejecución de EvRBF. Ambos conjuntos han sido generados a partir de la función definida según las ec. 4.1 a 4.7.

Una vez generados los conjuntos de entrenamiento y test, y en función de lo indicado en el apartado 3.4.1, los valores de las variables de entrada han sido normalizados y reescalados según los procedimientos allí descritos. Este será igualmente el mecanismo utilizado en el siguiente capítulo de evaluación del método.

4.3. Selección del método de inicialización de poblaciones

La primera serie de experimentos que se han llevado a cabo pretenden determinar cuáles de los métodos de inicialización de centros y radios descritos en 3.4.4 resultan ser los más adecuados.

Dado que se van a considerar tres mecanismos distintos para establecer los centros iniciales (IC_RND, IC_PAT e IC_KM) y cuatro para establecer los radios (IR_RND_CTE, IR_RND_VAR, IR_MIN_DIS e IR_MIN_DIS2), se ha evaluado cada uno de los binomios *selección de centros-selección de radios* para comprobar cuál de ellos es el que ofrece mejores resultados. En todos los casos, se han asignado valores por defecto a los parámetros que se precisan para ejecutar EvRBF, que corresponden a los mostrados en la tabla 4.1. Este conjunto de valores no es óptimo, pero empíricamente se han mostrado válidos para resolver diversos problemas durante las fases de implementación y prueba del método. Como se puede apreciar en dicha tabla, cada uno de los experimentos ha sido ejecutado 5 veces utilizando siempre los mismos conjuntos de entrenamiento y test, pues es la única opción si se desea mantener la definición original de este problema concreto. En cada ejecución se ha separado del conjunto de entrenamiento un 25 % de patrones, elegidos aleatoriamente, que integrarán el conjunto de validación, necesario para establecer el *fitness*, pues se utiliza el método EF_TRN_VAL como función de evaluación. En cada ejecución se han utilizado 100 individuos y el algoritmo se ha iterado durante 100 generaciones. La población inicial en cada ejecución es diferente y generada aleatoriamente. El límite de neuronas de la primera generación se ha establecido a un 5 % del tamaño del conjunto de patrones de entrenamiento. En cada generación se reemplaza el 30 % de los individuos y los que se reproducen son elegidos mediante torneo de tamaño 3. La tasa de aplicación de los operadores es igual a 1 para todos, por lo que la probabilidad de aplicación es igual para cada operador. Por su parte, para los operadores que además necesitan una probabilidad de aplicación interna ésta ha sido establecida a 0.5.

Para poder comparar las distintas combinaciones *selección de centros-selección de radios* se ha considerado el menor ECM sobre el conjunto de test (es decir, el error de generalización) devuelto por los individuos de la última generación en cada una de las ejecuciones del algoritmo. El valor de bondad de cada combinación es el promedio de los 5 mejores ECM surgidos de las 5 ejecuciones realizadas para dicha combinación. La tabla 4.2 muestra dicho error de generalización. La representación gráfica de

Parámetro	*Valor*
Repeticiones de cada experimento	5
Tamaño del conjunto de validación	25 %N_{Tr}
Tamaño población	100
Generaciones	100
Límite neuronas primera generación	5 %N^*_{Tr}
Población reemplazada	30 %
Tamaño torneo	3
Tasa aplicación operadores	1
Prob. interna operadores	0.5
Función evaluación	*EV_TRN_VAL*

Tabla 4.1: Valores por defecto de los parámetros de EvRBF. Se utilizan inicialmente en los experimentos realizados para determinar el método de inicialización de poblaciones. N_{Tr} es el tamaño del conjunto de entrenamiento. N^*_{Tr} es dicho tamaño menos el del conjunto de validación. La tasa de aplicación de operadores determina su probabilidad de aplicación, como reflejaba el apartado 3.5. La probabilidad interna del operador es la probabilidad de afectar a cada neurona de la red, como se indicaba en los apartados 3.2.1.2, 3.2.4 y 3.2.3.

dichos datos es la que muestra la fig. 4.2.

La primera conclusión del análisis de los distintos métodos de inicialización de centros y radios es que no hay ninguno de ellos que varíe radicalmente el comportamiento de EvRBF. No obstante, la fig. 4.2 muestra como los tres mejores resultados se alcanzan cuando se eligen patrones del conjunto de entrenamiento como centros para las neuronas, seguidos por métodos en los que el algoritmo k-medias es el que elige los centros. Esto implica que el algoritmo está utilizando la información recogida de los patrones de entrada para comenzar la búsqueda en aquellas regiones en las que se conoce que está definida la función, dado que por la propia naturaleza de las RNFBR para aquellas regiones de las que no se conocen los valores a estimar, la salida proporcionada debe ser nula. En cuanto al método de elección de centros aleatorios, IC_RAND, se muestra que puede conseguir mejores resultados que los anteriores cuando se utilizan conjuntamente radios basados en distancias. La decisión final será tomar IC_PAT como método de selección de centros para ejecutar en lo sucesivo el algoritmo EvRBF.

En cuanto al establecimiento del valor de los radios, la fig. 4.2 muestra como están representados todos los métodos de inicialización de radios dentro de los 4 mejores resultados. No obstante, una vez elegido IC_PAT como iniciador de centros, hemos de restringirnos a los métodos IR_MIN_DIS2, IR_RND_CTE e IR_RND_VAR, y más en concreto al primero, por ser el que devuelve el menor ECM de entre ellos. Básicamente, la

		Centros	
Radios	*IC_RND*	*IC_PAT*	*IC_KM*
IR_RND_CTE	$2.2 \times 10^{-4} \pm 1.8 \times 10^{-4}$	$1.2 \times 10^{-4} \pm 0.6 \times 10^{-4}$	$1.6 \times 10^{-4} \pm 0.9 \times 10^{-4}$
IR_RND_VAR	$1.9 \times 10^{-3} \pm 0.8 \times 10^{-4}$	$1.3 \times 10^{-4} \pm 1.2 \times 10^{-4}$	$1.8 \times 10^{-4} \pm 1.1 \times 10^{-4}$
IR_MIN_DIS	$1.9 \times 10^{-3} \pm 0.6 \times 10^{-4}$	$1.9 \times 10^{-4} \pm 1.1 \times 10^{-4}$	$1.4 \times 10^{-4} \pm 0.4 \times 10^{-4}$
IR_MIN_DIS2	$1.7 \times 10^{-4} \pm 0.5 \times 10^{-4}$	$\mathbf{1.1 \times 10^{-4}} \pm 0.7 \times 10^{-4}$	$2.5 \times 10^{-4} \pm 2.3 \times 10^{-4}$

Tabla 4.2: ECM promedio y desviación estándar obtenido utilizando combinaciones de los distintos métodos de inicialización de centros (columnas) y radios (filas). Los resultados muestran que la utilización de IC_PAT e IR_MIN_DIS2 son los que consiguen una mayor reducción del error.

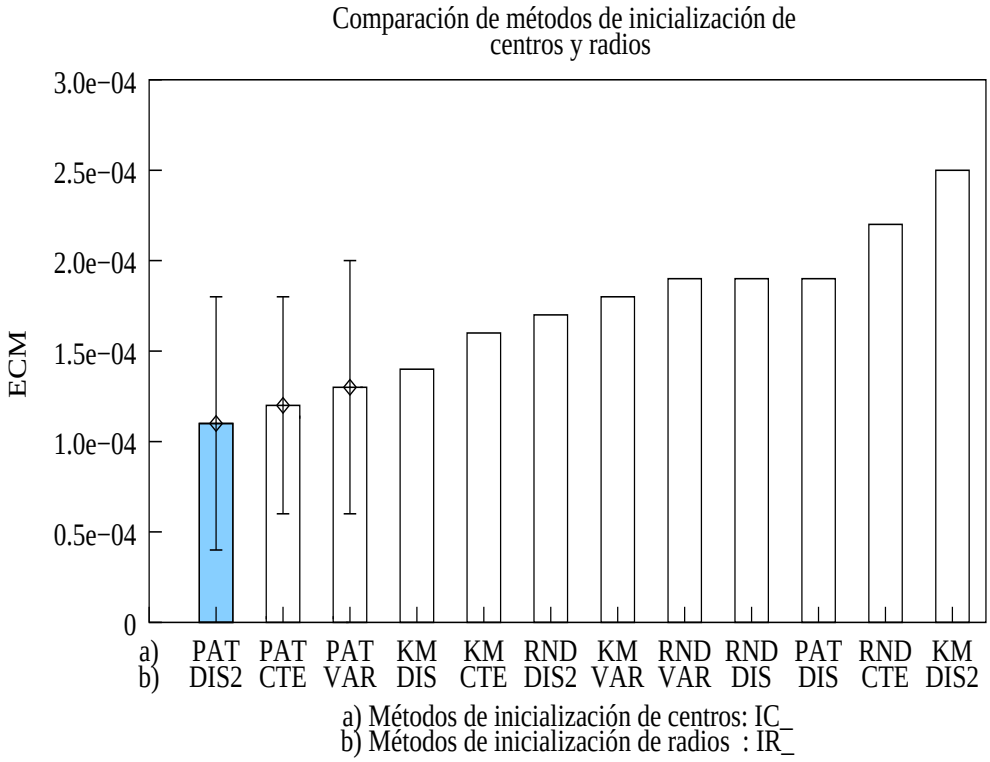


Figura 4.2: ECM de generalización obtenido utilizando combinaciones de los distintos métodos de inicialización de centros y radios. Se representa la desviación estándar para los tres mejores valores. Los resultados muestran la utilización conjunta de IC_PAT e IR_MIN_DIS2 es la que consigue una mayor reducción del error.

ventaja de IR_MIN_DIS2 estriba en la capacidad que aporta a las neuronas de cooperar unas con otras cuando intentan aproximar puntos del espacio de entrada que se hallan entre ellas.

A modo de resumen, en los experimentos que se siguen a continuación

Límite neuronas inicial	*ECM*	*Tamaño red*	*Tiempo (seg.)*
1 %	$1.8 \times 10^{-4} \pm 1.4 \times 10^{-4}$	$\mathbf{16} \pm 3$	$\mathbf{13} \pm 1$
5 %	$2.0 \times 10^{-4} \pm 1.4 \times 10^{-4}$	$\mathbf{16} \pm 4$	$\mathbf{13} \pm 2$
10 %	$0.5 \times 10^{-4} \pm 0.2 \times 10^{-4}$	23 ± 1	20 ± 1
15 %	$0.4 \times 10^{-4} \pm 0.1 \times 10^{-4}$	23 ± 4	26 ± 3
20 %	$\mathbf{0.3 \times 10^{-4}} \pm 0.2 \times 10^{-2}$	25 ± 2	31 ± 4

Tabla 4.3: ECM, tamaño de red y tiempo en segundos para distintos límites al tamaño inicial de los individuos de la primera generación. El límite se representa en forma de porcentaje sobre el tamaño del conjunto de entrenamiento. Aunque el mejor error se consigue a costa de mayor número de neuronas, la elección del 20 % supone una considerable desventaja tanto con respecto al tamaño de las redes como, especialmente, al tiempo invertido.

así como en los descritos en el siguiente capítulo, el método de inicialización de poblaciones utilizará selección de centros basado en patrones (IC_PAT) y selección de radios basados en la distancia a la más cercana multiplicada por un factor de 1.75 (IR_MIN_DIS2).

4.4. Selección del límite de neuronas iniciales

El siguiente conjunto de experimentos intenta determinar si el valor del 5 % del tamaño del conjunto de entrenamiento, utilizado anteriormente, es el más adecuado para ser fijado como límite superior del número de neuronas de la capa oculta de la población inicial de RNFBR. Nuevamente, se ha procedido a ejecutar 5 veces el algoritmo probando distintos valores para este parámetro. En concreto se ha experimentado con la posibilidad de usar los valores 1 %, 5 %, 10 %, 15 %, y 20 %. Al resto de parámetros se les han vuelto a asignar los valores mostrados en la tabla 4.1 y los métodos de inicialización de centros y radios utilizados han sido IC_PAT e IR_MIN_DIS2, respectivamente.

Los resultados, promediados sobre las 5 ejecuciones, son los mostrados de forma numérica en la tabla 4.3 y gráficamente en las figs. 4.3 y 4.4.

Los resultados obtenidos muestran por un lado algo esperado, y es que a medida que se utiliza un número mayor de neuronas el error cometido decrece. Sin embargo, ha de establecerse una cota a este número dado que el incremento del límite para las neuronas de la población inicial supone un incremento del tamaño de las redes generadas y, consecuentemente, del tiempo de ejecución que se necesita. Es por ello que se ha adoptado la utilización del valor 10 % como el más adecuado para el algoritmo - EvRBF dado que ofrece una reducción del ECM significativa con respecto a porcentajes menores y representa un buen compromiso con respecto al

incremento del tamaño de las redes de la última generación y al tiempo de ejecución.

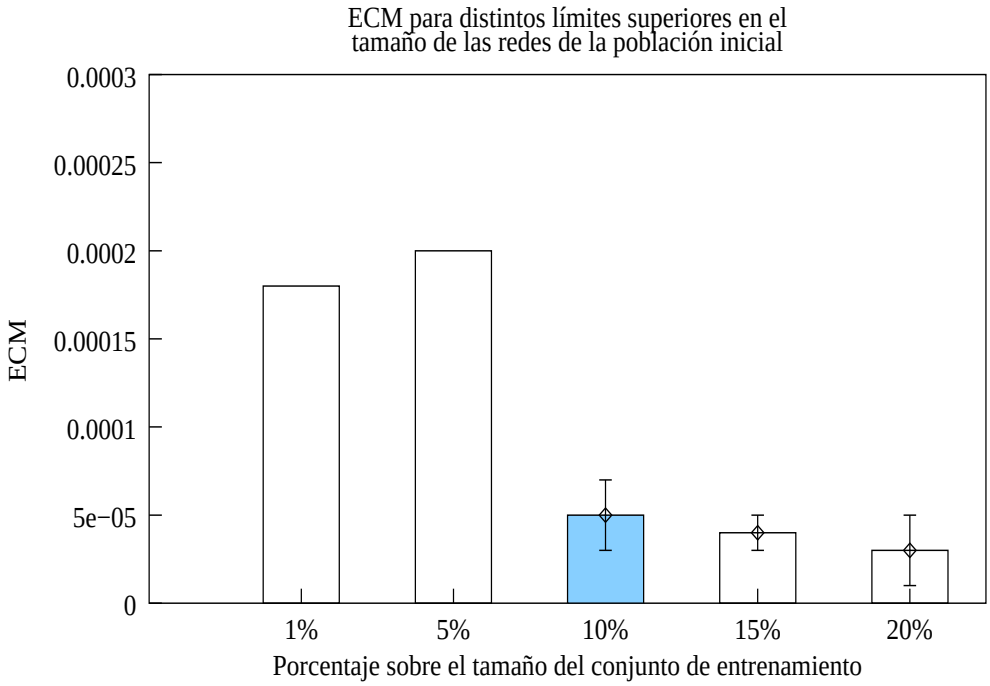


Figura 4.3: ECM alcanzado con respecto a distintos límites para el tamaño de las redes que componen la población inicial. Dicho límite se establece como porcentaje del tamaño del conjunto de entrenamiento. Para los tres mejores valores se ha representado también el valor de la desviación estándar. Los datos muestran que, a pesar de que el mejor valor es el devuelto por un límite igual al 20 % de los patrones de entrada, la utilización del mismo supone un incremento tanto en el número de neuronas como en el tiempo utilizado. Por ello se usará el valor del 10 %, dado que constituye un buen compromiso entre los valores de ECM, tamaño de redes y tiempo necesario.

4.5. Selección de la función de evaluación

La segunda serie de experimentos realizados ha servido para determinar cuál debe ser el método elegido para asignar un valor de *fitness* a cada individuo; esto es, si será el método de particiones múltiples (EF_PARTS), o por el contrario el de un solo conjunto de entrenamiento y otro de validación (EF_TRN_VAL). Ambos métodos fueron descritos 3.4.5.

Los experimentos se han llevado a cabo de forma similar a la descrita en el apartado anterior. De esta forma, el método de particiones múltiples

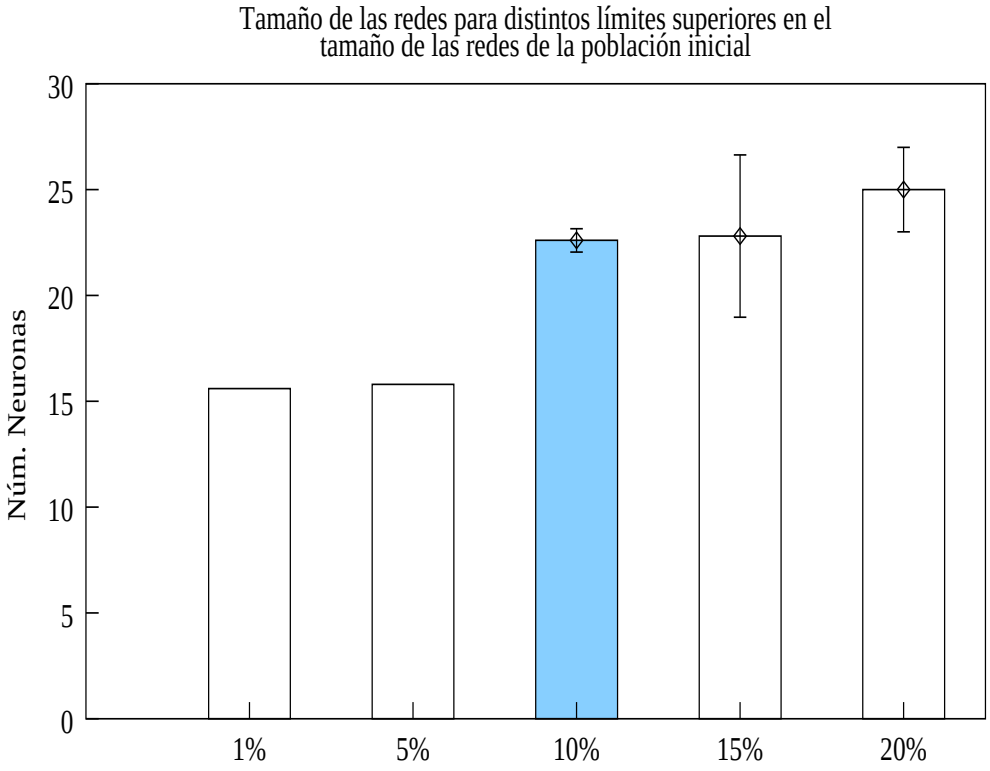


Figura 4.4: Número de neuronas de la capa oculta en las redes de la última población para distintos límites a dicho número para las redes de la población inicial. Dichos límites están expresados en el eje de abscisas como porcentajes sobre el tamaño del conjunto de entrenamiento. Para aquellos valores que devolvían los mejores ECM se ha representado también la desviación estándar. El gráfico muestra que cuanto mayor es el límite establecido, mayores son las neuronas presentes en la última generación. No obstante, las redes generadas con los valores 10 %, 15 % y 20 % mantienen tamaños similares, a la vez que reducen significativamente el ECM con respecto a porcentajes más bajos, como se observa en la fig. 4.3.

ha sido probado en 5 ejecuciones del algoritmo partiendo de poblaciones iniciales aleatorias, utilizando IC_PAT e IR_MINS_DIS2 para la inicialización de centros y radios, y el 10 % del tamaño del conjunto de entrenamiento como límite superior para el número de neuronas de la capa oculta de la primera generación. Análogamente, el segundo de los métodos comentados ha sido probado en otras 5 ejecuciones bajo las mismas condiciones. Los resultados obtenidos para los dos métodos de asignación de *fitness* considerados se muestran en la tabla 4.4.

El análisis de los resultados muestra que ambos métodos ofrecen resultados similares en cuanto al ECM conseguido (un poco más reducido

Función de Evaluación	*ECM*	*Tamaño*	*Tiempo*
EF_TRN_VAL	$\mathbf{0.7 \times 10^{-4}} \pm 0.3 \times 10^{-4}$	$\mathbf{18} \pm 4$	$\mathbf{18} \pm 2$
EF_PARTS	$\mathbf{0.5 \times 10^{-4}} \pm 0.1 \times 10^{-4}$	20 ± 2	196 ± 14

Tabla 4.4: ECM de generalización, tiempo de ejecución y tamaño de las redes para distintas funciones de evaluación. EF_TRN_VAL ofrece un error y tamaño de red muy similar a EF_PARTS, pero lo hace en un tiempo 10 veces menor.

en el caso de EF_PARTS) y al tamaño de las redes (en este caso a favor de EF_TRN_VAL). Sin embargo es clara la superioridad del método basado en un solo conjunto de entrenamiento y un solo conjunto de validación en cuanto al tiempo que requiere, pues es 10 veces menor que el requerido si se utilizan particiones múltiples. La utilización del conjunto de validación es suficiente para conseguir que EvRBF no sobre-entrene las redes que va construyendo, permitiendo que mantengan una buena capacidad de generalización a lo largo del proceso evolutivo.

Consecuentemente, el resto de experimentos llevados a cabo, tanto en este capítulo como en el siguiente de evaluación del método, utilizarán EF_TRN_VAL cómo función de evaluación que asigne el *fitness* a los individuos.

4.6. Determinación del porcentaje de reemplazo

Una vez elegidos el método de inicialización poblaciones y el mecanismo con el que dotar de *fitness* a cada individuo, han sido utilizados para determinar el porcentaje de individuos que deben ser reemplazados cada vez que se desea crear una nueva población. Análogamente, se deberán generar tantos nuevos individuos mediante los operadores genéticos como los que vayan a ser reemplazados. Así, el valor concreto que deseamos estimar se corresponde con el parámetro denominado **población reemplazada** presente en la tabla 4.1.

La metodología empleada ha sido similar a los casos anteriores, comprobando la efectividad del algoritmo ante distintos porcentajes de reemplazo, calculados dichos porcentajes sobre el tamaño de la población inicial. De esta forma, se han considerado 5 valores para este parámetro, a saber: 10 %, 30 %, 50 %, 70 % y 90 %.

No obstante, para cada uno de los valores de este parámetro se ha utilizado un número distinto de generaciones, pues el porcentaje de individuos a reemplazar determina la cantidad de evaluaciones de RNFBR que se llevarán a cabo. De esta forma, mediante la variación en el número de

evaluaciones se ha conseguido que en todos los casos se llegue a una solución final tras un número similar de evaluaciones, permitiendo que la comparación sea significativa.

El número de evaluaciones que se ha tomado como referencia es el de 3000, llevadas a cabo con los parámetros usados por defecto (ver tabla 4.1), esto es, 100 generaciones, reemplazando el 30 % de la población, que resulta ser de 100 individuos. La tabla 4.5 recoge el número de generaciones que se ha usado para cada valor del porcentaje de reemplazo.

Porcentaje de Reemplazo	*Número de Generaciones*
10 %	300
30 %	100
50 %	60
70 %	45
90 %	35

Tabla 4.5: Número de generaciones con que se ejecutará EvRBF en función del porcentaje de individuos que serán reemplazados en cada nueva generación. La combinación de ambos parámetros permite mantener el número de evaluaciones alrededor de las 3000.

El resto de parámetros ha continuado tal y como se mostraba en la tabla 4.1. Se han vuelto a realizar 5 ejecuciones del algoritmo para cada uno de los valores a probar, partiendo en cada ejecución de una población inicial aleatoria y utilizando los mismos conjuntos de entrenamiento y test que en las anteriores ocasiones.

Porcentaje de reemplazo	*ECM*	*Tamaño*	*tiempo*
10 %	$\mathbf{0.5 \times 10^{-4}} \pm 0.2 \times 10^{-4}$	22 ± 2	26 ± 2
30 %	$0.9 \times 10^{-4} \pm 0.1 \times 10^{-4}$	$\mathbf{17} \pm 2$	17 ± 1
50 %	$1.4 \times 10^{-4} \pm 0.9 \times 10^{-4}$	18 ± 3	15 ± 2
70 %	$0.9 \times 10^{-4} \pm 0.5 \times 10^{-4}$	19 ± 2	16 ± 1
90 %	$1.0 \times 10^{-4} \pm 0.2 \times 10^{-4}$	$\mathbf{17} \pm 1$	$\mathbf{14} \pm 2$

Tabla 4.6: ECM, tamaño de red y tiempo de ejecución promedios obtenidos para distintos porcentajes de población de individuos reemplazados en cada nueva generación. El mejor ECM se consigue cuando el 10 % de la población es reemplazada y substituida por un número igual de nuevos individuos, aunque ello se hace a costa de un mayor número de neuronas y un mayor tiempo de ejecución. El resultado más equilibrado se consigue cuando el 30 % de la población es reemplazo, pues se alcanza el mejor error de generalización con el menor número de neuronas y en tiempo similar al mínimo encontrado.

El resultado obtenido, cuyos datos pueden verse en la tabla 4.6, o representados en la fig. 4.5, muestra que el mejor resultado se obtiene cuando reemplazamos el 10 % de la población, aunque se consigue gracias al incremento del número de generaciones que lleva asociado, lo cual lleva a la

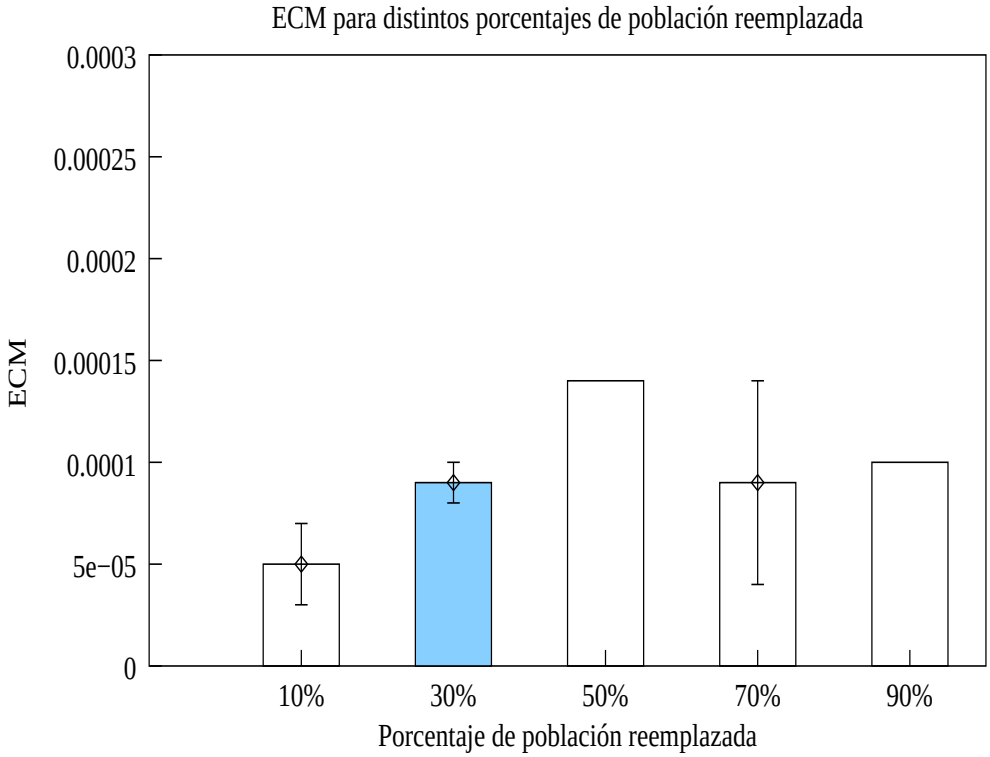


Figura 4.5: ECM para distintos porcentajes de individuos reemplazados en cada nueva generación. Para aquellos valores que ofrecen los mejores resultados se ha representado también la desviación estándar. El mejor valor se alcanza cuando se reemplaza el 10 % de la población, aunque el porcentaje que representa un mejor compromiso entre capacidad de generalización y tamaño de la red es 30 %.

creación de redes más grandes. El resultado más equilibrado, sin embargo, corresponde a la utilización de un 30 % de reemplazo de población. Con este porcentaje se consiguen las redes de menor tamaño que mejor generalizan, resultando un buen compromiso entre ambos factores. La decisión final ha consistido en asignar al porcentaje de reemplazo el valor del 30 %.

4.7. Determinación del tamaño del torneo

Una vez hemos estudiado los mecanismos de inicialización, funciones de evaluación y porcentajes de población a reemplazar, procedemos a estimar el número de individuos que serán comparados cada vez que el operador de selección sea aplicado, esto es, el tamaño del torneo.

Se han realizado experimentos para determinar cuál es el valor más

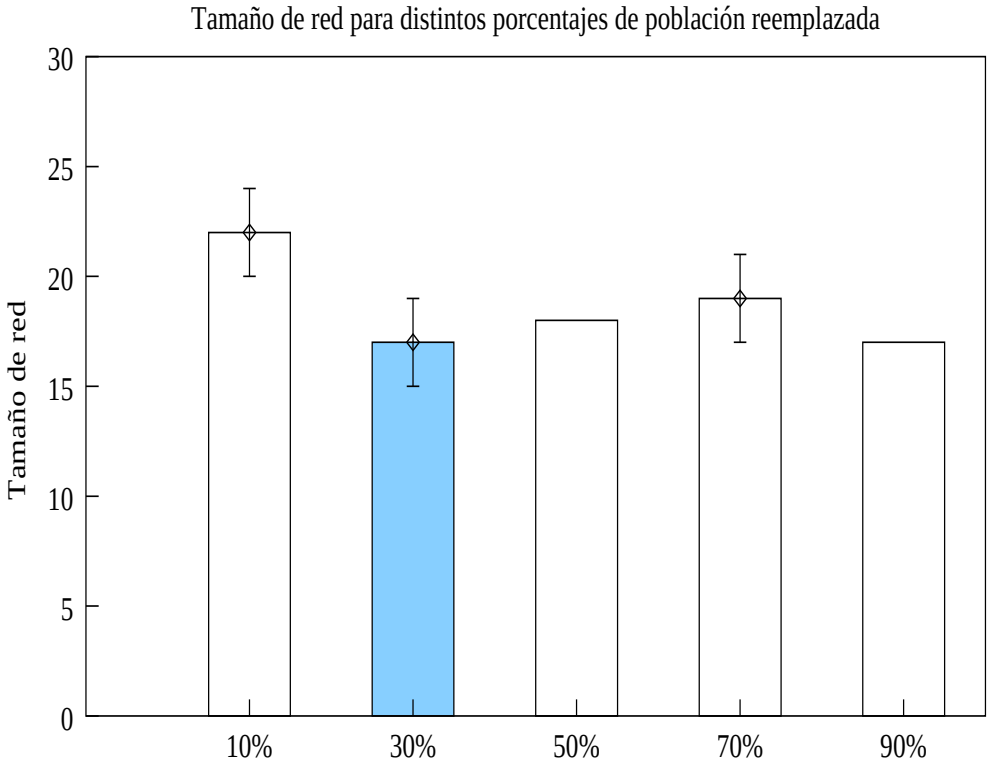


Figura 4.6: Tamaños de red para distintos porcentajes de individuos reemplazados en cada nueva generación. Para aquellos porcentajes que ofrecían los mejores resultados en cuanto al ECM conseguido (ver fig. 4.5), se ha representado también la desviación estándar. El mejor valor se alcanza cuando se reemplazan el 30 % ó el 90 % de la población. Sin embargo, el primero de estos valores tienen mayor capacidad de generalización.

apropiado, utilizando los valores ya conocidos para el método de iniciación de poblaciones, límite superior de neuronas de la primera generación, función de evaluación y porcentaje de población reemplazada. Los distintos valores para el tamaño de torneo que se han tenido en cuenta han sido: 2, 3, 4, 5, 8, 10, 15 y 20. Los operadores genéticos han sido aplicados en función de las tasas ya comentadas anteriormente (ver tabla 4.1) y los resultados reflejan el valor promedio a lo largo de 5 ejecuciones por cada valor del parámetro en estudio.

Tanto la tabla 4.7 como la fig. 4.7 muestran que, aunque con valores grandes de torneo se pueda reducir el error de generalización, se hace a costa de redes mucho mayores. Por ello, los valores más acertados para el tamaño del torneo son los pequeños. De esta forma, EvRBF es un algoritmo que no ejerce una alta presión selectiva, permitiendo que se reproduzcan individuos cuyo *fitness* no es el mejor hasta el momento. Esto

Tamaño de Torneo	*ECM*	*Tamaño*	*Tiempo*
2	$0.7 \times 10^{-4} \pm 0.2 \times 10^{-4}$	18 ± 1	17 ± 2
3	$0.9 \times 10^{-4} \pm 0.1 \times 10^{-4}$	17 ± 2	17 ± 1
4	$1.4 \times 10^{-4} \pm \times 0.910^{-4}$	20 ± 4	20 ± 2
5	$0.7 \times 10^{-4} \pm 0.3 \times 10^{-4}$	23 ± 3	22 ± 6
8	$0.5 \times 10^{-4} \pm 0.3 \times 10^{-4}$	25 ± 2	25 ± 2
10	$1.2 \times 10^{-4} \pm 0.4 \times 10^{-4}$	24 ± 4	25 ± 5
15	$0.5 \times 10^{-4} \pm 0.1 \times 10^{-4}$	31 ± 5	33 ± 8
20	$0.6 \times 10^{-4} \pm 0.3 \times 10^{-4}$	33 ± 6	37 ± 9

Tabla 4.7: ECM, tamaño y tiempo promedios obtenidos para distintos tamaños de torneo. Aun cuando el menor error de generalización se alcanza con torneo igual a 5, 8 y 15, es significativamente mejor el resultado hallado con tamaños pequeños, como 2 y 3, dado que se consiguen errores muy similares, con tamaños de red mucho más pequeños. Por este motivo, se utilizarán sólo 2 individuos en la utilización de la selección mediante torneo.

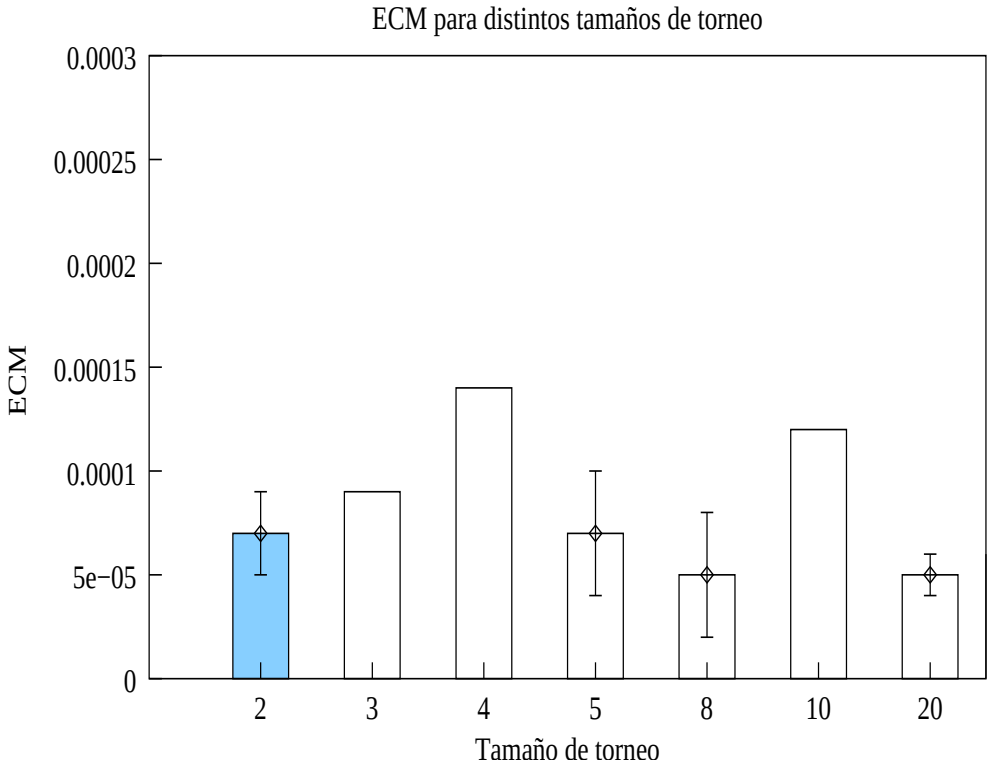


Figura 4.7: ECM promedio para distintos tamaños de torneo, utilizado para la selección de los individuos que se reproducirán. Para los tres mejores resultados se ha representado su desviación estándar. La mejor relación ECM-tamaño de red (ver tabla 4.7) se obtienen cuando el tamaño es pequeño; más concretamente cuando éste es igual a 2.

redunda en una mayor variabilidad dentro de la población e impide el sobre-entrenamiento de los individuos. Más concretamente, la utilización de sólo 2 individuos en cada una de las aplicaciones del operador de selección arroja un ECM equiparable a los menores hallados, así como las redes más pequeñas y los menores tiempos de ejecución. Por tanto, el valor que se utilizará como tamaño de torneo será de 2 individuos.

4.8. Determinación de la tasa de aplicación de los operadores evolutivos

Continuando con la misma filosofía aplicada en los apartados anteriores, se han llevado a cabo numerosos experimentos para determinar la mejor tasa de aplicación de los operadores descritos en 3.2, así como la probabilidad de aplicación interna a cada neurona en los casos que fuese necesario.

Para determinar la tasa más adecuada para un determinado operador, se han realizado varios experimentos probando distintos valores de la tasa del operador estudiado y manteniendo las de los demás operadores a 1. De esta forma, se han probado 5 valores distintos como tasa de aplicación del operador en estudio: 0, 0.5, 1, 2 y 4, como muestra la tabla 4.8. La probabilidad con que un determinado operador es posteriormente aplicado se calcula según la ec. 3.8 (pág. 85) , es decir, es el resultado de dividir su tasa de aplicación por la suma de las tasas de aplicación de todos los operadores.

La utilización de una tasa de aplicación igual a 0 pretende descubrir cuáles de los operadores podrían ser eliminados, siempre que se usen en conjunción con el resto. Esto permitirá crear un subconjunto de operadores mínimo cuya aplicación, garantice la generación de RNFBR con alto poder de generalización.

Tasas de aplicación de operadores genéticos
0
0.5
1
2
4

Tabla 4.8: Valores considerados para la tasas de aplicación de los operadores genéticos. Estos valores han sido utilizados en los diversos estudios realizados para estimar con cuál de ellos debe ser utilizado cada operador genético. La tasa de aplicación determina la probabilidad de aplicación de un operador según la ec. 3.8 (pág. 85).

Probabilidad de aplicación interna
0.1
0.25
0.5
0.75
1

Tabla 4.9: Valores considerados para la probabilidad de aplicación interna de los operadores genéticos. Se ha experimentado con estos valores para determinar el más adecuado para cada uno de los operadores que precisan de este parámetro. La probabilidad de aplicación interna hace referencia a la probabilidad con que cada neurona de la red es afectada por el operador.

Dado que la forma de actuación de operadores como X_MULTI, X_AVERAGE o C_RANDOM, entre otros, depende también de una probabilidad de aplicación interna, estos operadores han sido evaluados variando no sólo su tasa de aplicación, sino también dicha probabilidad de aplicación interna. Debido a esto, para el estudio de los operadores que requieren de este parámetro, el número de experimentos que se ha llevado a cabo ha sido aún mayor. Así, se han realizado combinaciones de cada uno de los valores de la tasa de aplicación no nulos con 5 valores distintos para la probabilidad de aplicación interna, a saber: 0.1, 0.25, 0.5, 0.75 y 1, como muestra la tabla 4.9. Por cada uno de los pares de valores *tasa de aplicación-probabilidad interna*, se ha ejecutado el algoritmo 5 veces, manteniendo el resto de operadores con tasa de aplicación igual a 1 y con probabilidad de aplicación interna igual a 0.5, en aquellos que la precisan.

El análisis final de los resultados se ha realizado registrando el menor ECM devuelto en cada ejecución de EvRBF, calculado usando los individuos de la población final aplicados sobre el conjunto de test. Posteriormente, se ha realizado la media aritmética considerando los 5 valores de ECM registrados por cada tasa de aplicación o por cada par *tasa de aplicación-probabilidad interna*. Este valor medio es el mostrado en las figs. 4.8 a 4.19.

4.8.1. Tasa de aplicación del operador X_FIX

El operador X_FIX, cuya forma de actuar queda recogida en el apartado 3.2.1.1, precisa únicamente del establecimiento de su tasa de aplicación. Es por ello, que se han realizado un total de 25 experimentos, que han producido los resultados mostrados en la tabla 4.10, cuya representación gráfica se muestra en la fig. 4.8.

Los resultados generados para el operador X_FIX muestran que los mejores valores que se pueden utilizar son 0.5 y 1, aunque este último obtie-

Tasa Aplicación	*ECM*	*Tamaño*
0	$1.1 \times 10^{-4} \pm 0.5 \times 10^{-4}$	19 ± 2
0.5	$0.9 \times 10^{-4} \pm 0.4 \times 10^{-4}$	20 ± 3
1	$0.9 \times 10^{-4} \pm 0.2 \times 10^{-4}$	17 ± 2
2	$1.1 \times 10^{-4} \pm 0.6 \times 10^{-4}$	15 ± 2
4	$1.4 \times 10^{-4} \pm 0.6 \times 10^{-4}$	16 ± 2

Tabla 4.10: Determinación de la tasa de aplicación del operador X_FIX. Las columnas muestran los valores promedios de ECM y tamaño de red a lo largo de 5 ejecuciones de EvRBF para cada uno de los valores de la tasa de aplicación. El menor error de generalización se obtiene cuando la tasa de aplicación es 0.5 ó 1, aunque con este último valor se consiguen redes más pequeñas.

ne redes sensiblemente más pequeñas. El intercambio de información que produce este operador resulta por tanto efectivo y se perfila como uno de los operadores que formarán parte del conjunto de operadores mínimos.

4.8.2. Tasa de aplicación del operador X_MULTI

El operador X_MULTI presenta el primer caso en el que se han de establecer simultáneamente los parámetros de tasa de aplicación y probabilidad de aplicación interna, p_{x_multi}. Por ello se ha probado en primer lugar la posibilidad de utilizar el operador con tasa de aplicación igual a 0 y, posteriormente, las distintas combinaciones que surgen de utilizar las restantes tasas de aplicación de la tabla 4.8 junto a las probabilidades de aplicación internas de la tabla 4.9.

	Prob. de aplicación Interna				
Tasa Aplicación	*0.1*	*0.25*	*0.5*	*0.75*	*1*
Nula			1.0×10^{-4}		
0.5	1.0×10^{-4}	0.9×10^{-4}	0.9×10^{-4}	1.0×10^{-4}	0.8×10^{-4}
1	1.2×10^{-4}	$\mathbf{0.6 \times 10^{-4}}$	0.8×10^{-4}	1.0×10^{-4}	1.6×10^{-4}
2	1.1×10^{-4}	1.4×10^{-4}	1.1×10^{-4}	1.0×10^{-4}	1.0×10^{-4}
4	2.4×10^{-4}	1.1×10^{-4}	1.4×10^{-4}	1.1×10^{-4}	1.2×10^{-4}

Tabla 4.11: Determinación de la tasa de aplicación del operador X_MULTI. Se presenta el ECM promedio a lo largo de 5 ejecuciones del algoritmo para cada una de las combinaciones *tasa de aplicación-probabilidad de aplicación interna*, salvo en el caso de tasa de aplicación nula en el que no es necesario establecer probabilidad interna. El mejor resultado se obtiene cuando se usa una tasa de aplicación igual a 1 y probabilidad interna, p_{x_multi}, igual a 0.25.

La tabla 4.11 muestra el valor promedio del mínimo ECM encontrado en las 5 ejecuciones realizadas para cada uno de los experimentos realizados. La representación gráfica de dichos datos es la que muestra la fig. 4.9.

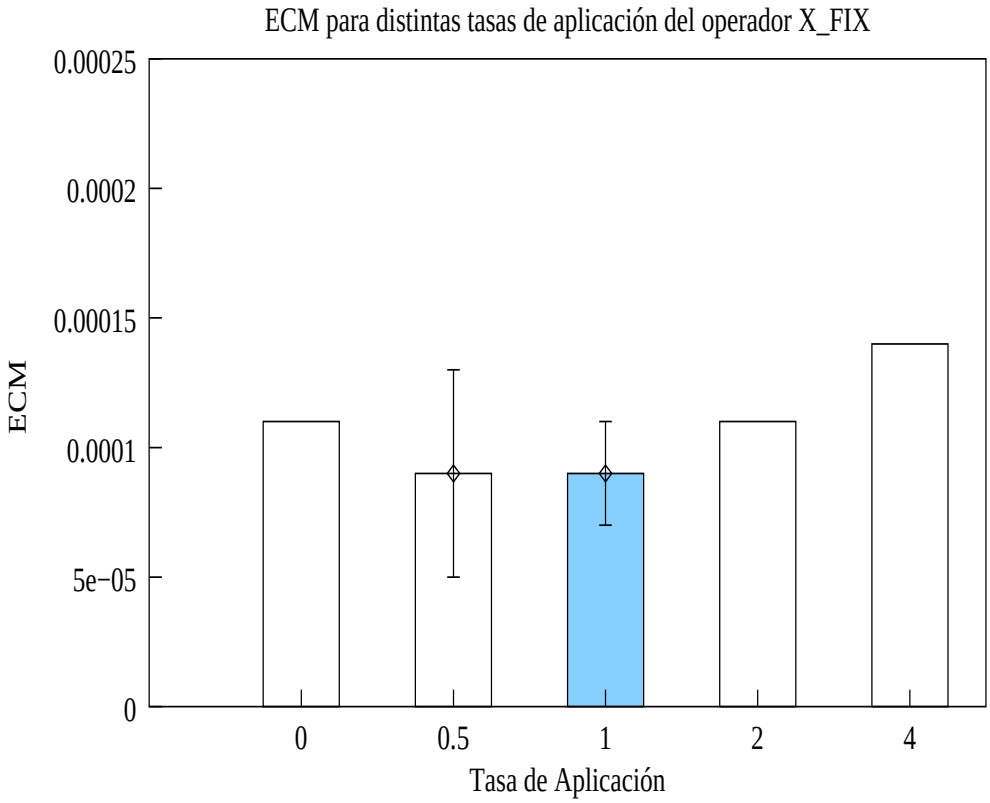


Figura 4.8: Determinación de la tasa de aplicación del operador X_FIX. Se muestra el ECM promedio calculado sobre 5 ejecuciones para distintos valores de tasa de aplicación. Los valores de tasa de aplicación que ofrecen menor ECM han sido representados junto con su desviación estándar. El mejor resultado se consigue cuando la tasa de aplicación es establecida a valores medios-bajos, como 0.5 y 1.

Se puede observar que el ECM más pequeño encontrado es de 0.6×10^{-4}. Este valor se obtiene cuando se asigna a este operador una tasa de aplicación igual a 1, al tiempo que se mantiene su tasa de probabilidad interna a 0.25. Al igual que ocurría con X_FIX, este segundo operador de recombinación se muestra nuevamente efectivo, lo cual indica que con el tipo de redes RNFBR, el intercambio de información resulta adecuado para evolucionar hacia una buena solución final.

4.8.3. Tasa de aplicación del operador X_AVERAGE

El operador de recombinación promediada es el segundo que presenta la circunstancia de tener que ser estudiado con respecto al valor de su tasa de aplicación junto con el valor de la probabilidad interna, $p_{x\_average}$.

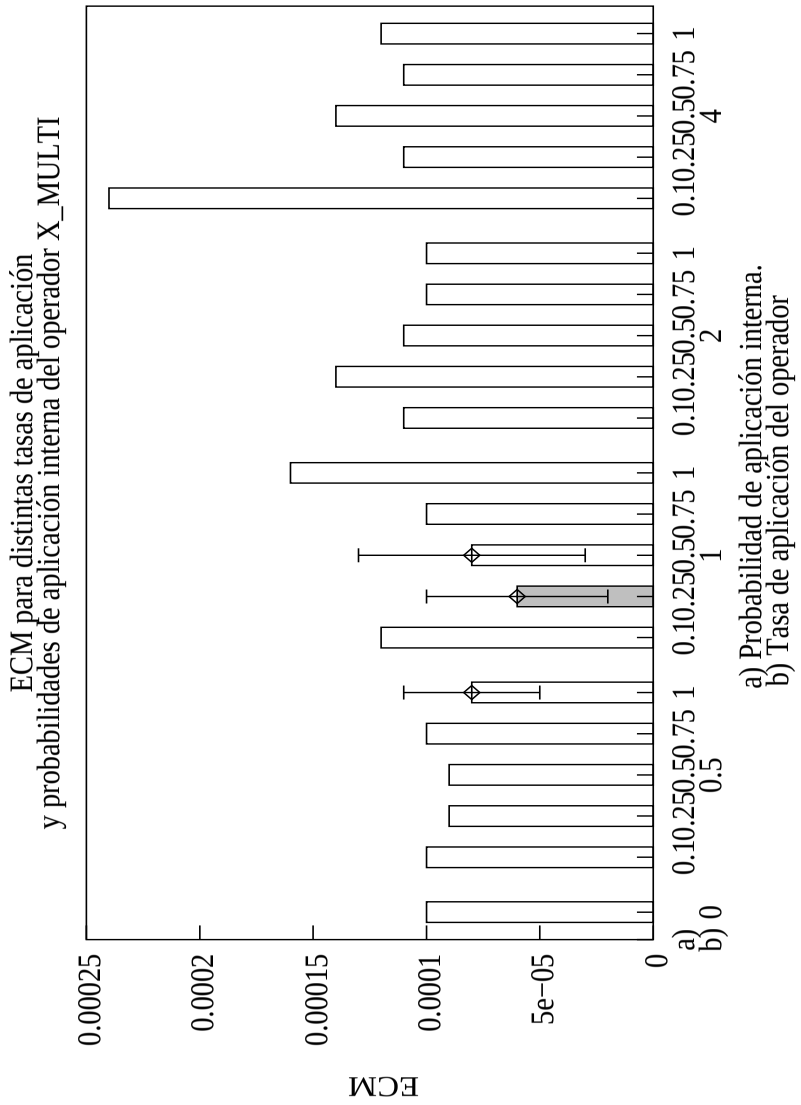


Figura 4.9: Determinación de la tasa de aplicación del operador X_MULTI. Se representa el ECM promediado a lo largo de 5 ejecuciones del algoritmo, para las distintas combinaciones de *tasa de aplicación-probabilidad de aplicación interna*. Un caso excepcional lo constituye la de tasa de aplicación nula, pues no es necesario establecer probabilidad interna. Para los mejores resultados, se ha añadido la representación de la desviación estándar. El mejor resultado se obtiene cuando se usa una tasa de aplicación igual a 1 y probabilidad interna, p_{x_multi}, igual a 0.25.

	Prob. de aplicación Interna				
Tasa Aplicación	*0.1*	*0.25*	*0.5*	*0.75*	*1*
Nula			1.0×10^{-4}		
0.5	0.8×10^{-4}	1.0×10^{-4}	1.2×10^{-4}	$\mathbf{0.6 \times 10^{-4}}$	1.0×10^{-4}
1	1.1×10^{-4}	1.4×10^{-4}	0.8×10^{-4}	0.8×10^{-4}	1.0×10^{-4}
2	1.8×10^{-4}	0.9×10^{-4}	1.1×10^{-4}	1.2×10^{-4}	1.3×10^{-4}
4	2.4×10^{-4}	1.2×10^{-4}	1.0×10^{-4}	0.8×10^{-4}	1.2×10^{-4}

Tabla 4.12: Determinación de la tasa de aplicación del operador X_AVERAGE. Se presenta el ECM promedio a lo largo de 5 ejecuciones del algoritmo para cada una de las combinaciones *tasa de aplicación-probabilidad de aplicación interna*, salvo en el caso de tasa de aplicación nula en el que no es necesario establecer probabilidad interna. El mejor resultado se obtiene cuando se usa una tasa de aplicación igual a 0.5 y probabilidad interna, $p_{x\_average}$, igual a 0.75.

Por ello, la mecánica que se ha utilizado es similar a la vista en el anterior apartado, es decir, se ha probado cada uno de los valores de probabilidad interna con cada uno de los valores de tasas de aplicación, salvo, obviamente, cuando ésta última es nula. Los resultados han sido de nuevo promediados a lo largo de 5 generaciones y ésto es precisamente lo que puede verse en la tabla 4.12, cuya representación gráfica corresponde a la fig. 4.10.

Los resultados invitan a utilizar el operador X_AVERAGE con una tasa de aplicación igual a 0.5, afectando cada vez que es utilizado a las tres cuartas partes de las neuronas de la red sobre la que se aplica, esto es, $p_{x\_average}$ debe ser establecido a 0.75. De esta forma, aunque la agrupación de la información contenida en las dos neuronas con las que opera reduce la diversidad global de la población, se permite aproximar con mayor exactitud la zona de entradas ubicada entre dos neuronas ya establecidas.

4.8.4. Tasa de aplicación de los operadores C_TUNER y C_RANDOM

Una vez concluido el estudio de los operadores de recombinación, comienza la determinación de los valores que deben asociarse al uso de los operadores de mutación incorporados a EvRBF.

En primer lugar, han sido evaluados los operadores C_TUNER y C_RANDOM, que se ocupan de modificar los valores establecidos para los centros de las neuronas. Ambos operadores llevan implícitos sendas probabilidades de aplicación interna p_{c_tuner} y p_{c_random}, respectivamente, que han sido también determinadas mediante los experimentos llevados a cabo.

La función representada en la fig. 4.1 ha sido de nuevo la elegida para

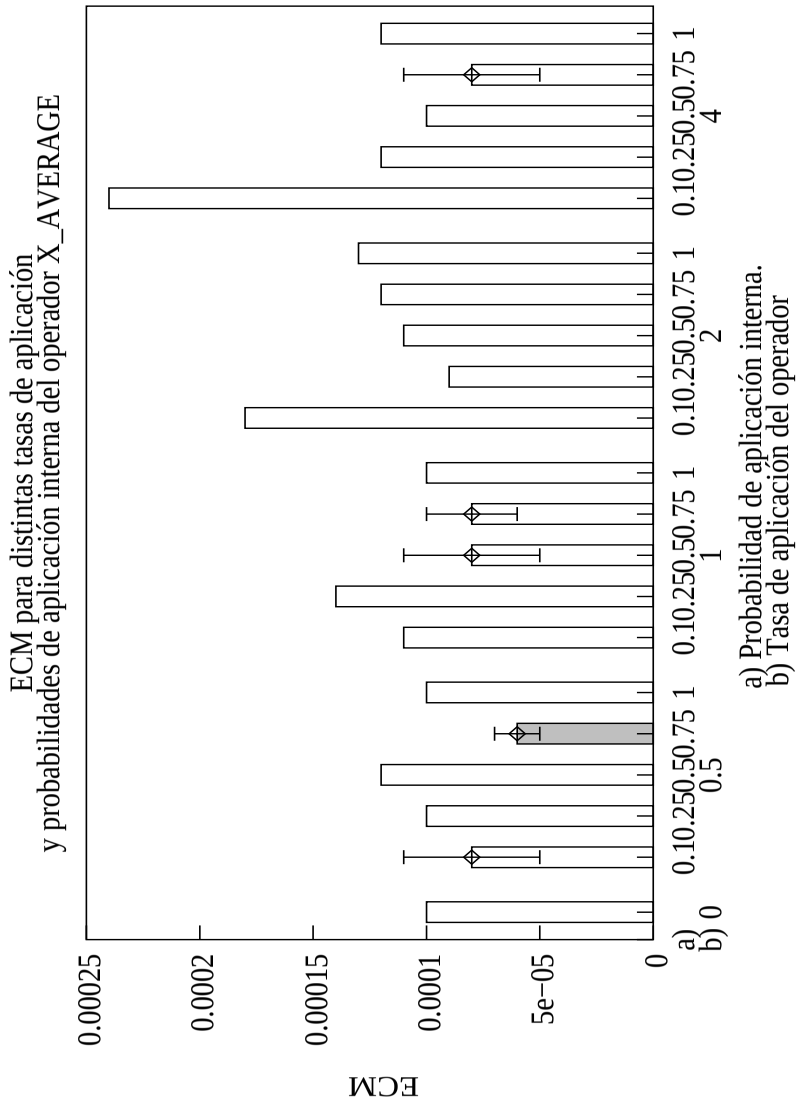


Figura 4.10: Determinación de la tasa de aplicación del operador X_AVERAGE. Se representa el ECM promediado a lo largo de 5 ejecuciones del algoritmo, para las distintas combinaciones de *tasa de aplicación-probabilidad de aplicación interna*. Un caso excepcional lo constituye la de tasa de aplicación nula, pues no es necesario establecer probabilidad interna. Para los mejores resultados se ha representado también la desviación estándar. El mejor resultado se obtiene cuando se usa una tasa de aplicación igual a 0.5 y probabilidad interna, $p_{x\_average}$, igual a 0.75.

ser aproximada, utilizando para ello distintos valores tanto de tasa de aplicación como de probabilidad interna, al igual que se hiciera con X_MULTI y X_AVERAGE. Para cada uno de los operadores se han realizado por tanto 21 experimentos distintos, repitiendo cada uno de ellos un total de 5 veces.

Una vez finalizadas las ejecuciones, los resultados recogidos en las mismas son los que se muestran en las tablas 4.13 y 4.14. La primera de ellas es la relativa al "afinador" de centros, C_TUNER, mientras que la segunda es la que pertenece a la elección de radios aleatorios, C_RANDOM. Gráficamente, los datos del error de generalización pueden observarse en las figs. 4.11 y 4.12, respectivamente.

	Prob. de aplicación Interna				
Tasa Aplicación	*0.1*	*0.25*	*0.5*	*0.75*	*1*
Nula			1.0×10^{-4}		
0.5	1.0×10^{-4}	0.9×10^{-4}	1.2×10^{-4}	$\mathbf{0.6 \times 10^{-4}}$	0.8×10^{-4}
1	1.1×10^{-4}	0.8×10^{-4}	0.8×10^{-4}	1.0×10^{-4}	1.4×10^{-4}
2	1.1×10^{-4}	0.8×10^{-4}	1.3×10^{-4}	1.2×10^{-4}	1.3×10^{-4}
4	0.8×10^{-4}	1.6×10^{-4}	0.8×10^{-4}	1.3×10^{-4}	1.2×10^{-4}

Tabla 4.13: Determinación de la tasa de aplicación del operador C_TUNER. Se presenta el ECM promedio a lo largo de 5 ejecuciones del algoritmo para cada una de las combinaciones *tasa de aplicación-probabilidad de aplicación interna*, salvo en el caso de tasa de aplicación nula en el que no es necesario establecer probabilidad interna. El mejor resultado se obtiene cuando se usa una tasa de aplicación igual a 0.5, junto a probabilidad interna, p_{c_tuner}, igual a 0.75.

La conclusiones con respecto a C_TUNER que pueden extraerse tras analizar las salidas de las distintas ejecuciones muestran que el operador debe ser aplicado con una tasa media-baja, aunque cada vez que se hace puede variar una cantidad considerable de neuronas. Así, la tasa de aplicación va a ser establecida a 0.5 y la probabilidad p_{c_tuner} será fijada a 0.75, por ser estos los valores que devuelven el menor ECM, como se refleja en la tabla 4.13.

El mismo comportamiento descrito para C_TUNER debe ser adoptado en la utilización de C_RANDOM. Encontramos, no obstante, que el valor utilizado para la tasa de aplicación puede ser de 1 ó de 2, dado que con ambos se consigue el menor ECM de forma más frecuente. En concreto, el valor más regular en cuanto a capacidad de generalización de redes es 1. Para esta tasa de aplicación, debe elegirse una probabilidad de aplicación igual a 1, por ser ésta la que arroja redes más pequeñas como muestra la fig. 4.13.

Las tasas de aplicación de los operadores de modificación de centros

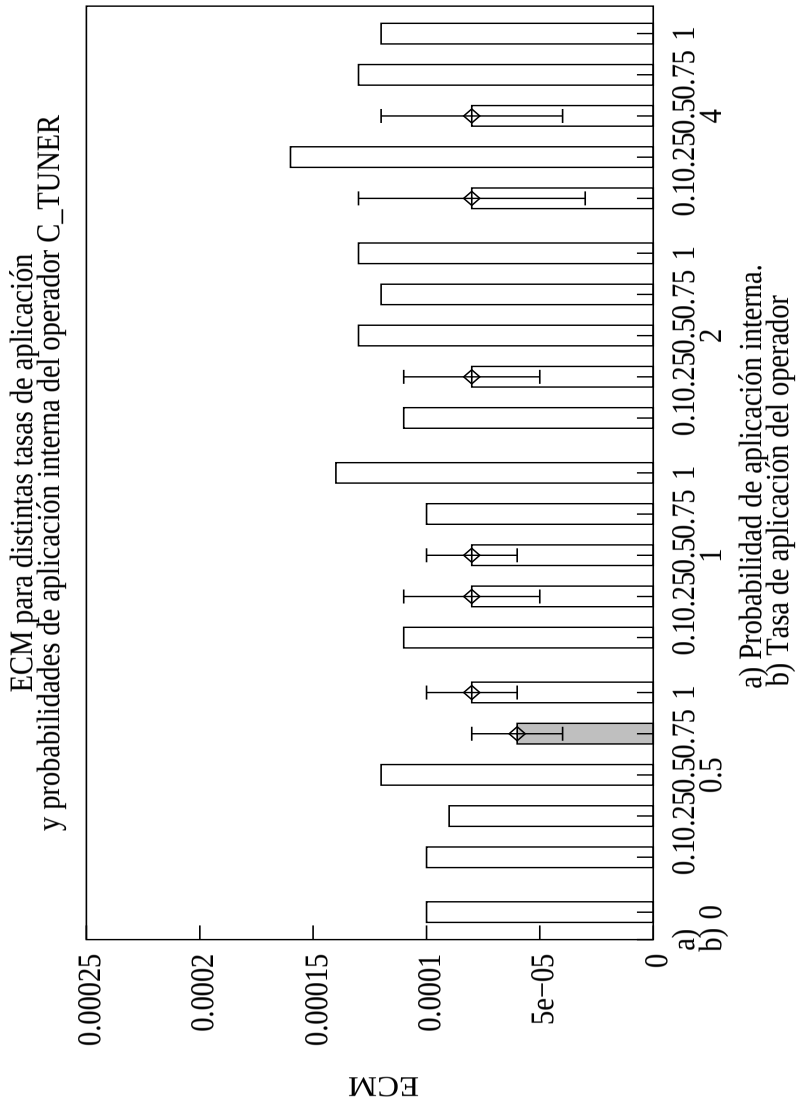


Figura 4.11: Determinación de la tasa de aplicación del operador C_TUNER. Se representa el ECM promediado a lo largo de 5 ejecuciones del algoritmo, para las distintas combinaciones de *tasa de aplicación-probabilidad de aplicación interna*. Un caso excepcional lo constituye la de tasa de aplicación nula, pues no es necesario establecer probabilidad interna. Para los mejores resultados se ha representado también la desviación estándar. El mejor resultado se obtiene cuando se usa una tasa de aplicación igual a 0.5 y probabilidad interna, p_{c_tuner}, igual a 0.75.

	Prob. de aplicación Interna				
Tasa Aplicación	*0.1*	*0.25*	*0.5*	*0.75*	*1*
Nula			1.1×10^{-4}		
0.5	1.1×10^{-4}	$\mathbf{0.9 \times 10^{-4}}$	1.2×10^{-4}	$\mathbf{0.9 \times 10^{-4}}$	1.5×10^{-4}
1	1.0×10^{-4}	$\mathbf{0.9 \times 10^{-4}}$	1.0×10^{-4}	$\mathbf{0.9 \times 10^{-4}}$	$\mathbf{0.9 \times 10^{-4}}$
2	$\mathbf{0.9 \times 10^{-4}}$	$\mathbf{0.9 \times 10^{-4}}$	1.7×10^{-4}	1.0×10^{-4}	1.0×10^{-4}
4	2.3×10^{-4}	$\mathbf{0.9 \times 10^{-4}}$	1.1×10^{-4}	1.4×10^{-4}	1.2×10^{-4}

Tabla 4.14: Determinación de la tasa de aplicación del operador C_RANDOM. Se presenta el ECM promedio a lo largo de 5 ejecuciones del algoritmo para cada una de las combinaciones *tasa de aplicación-probabilidad de aplicación interna*, salvo en el caso de tasa de aplicación nula en el que no es necesario establecer probabilidad interna. Los mejores resultados se obtienen con varios valores, aunque los más homogéneos se consiguen cuando la tasa de aplicación es 1 ó 2. No obstante, las redes más pequeñas se consiguen cuando la tasa de aplicación es igual a 1 y la probabilidad de aplicación es también 1, como muestra la fig. 4.13.

así como sus probabilidades internas proporcionan a EvRBF un mecanismo idóneo para permitir la exploración del espacio de búsqueda, en busca de nuevos centros que puedan cubrir áreas no examinadas hasta el momento.

4.8.5. Tasa de aplicación de los operadores R_TUNER y -R_RANDOM

Dentro de los operadores que afectan directamente a los parámetros de configuración de cada neurona, procedemos al estudio de los que modifican el valor del radio de las mismas. Así, tanto el operador R_TUNER como el operador R_RANDOM permiten al AE buscar valores dentro del espacio de búsqueda de radios para las redes a generar.

Al igual que los mutadores de centros, estos operadores necesitan ser evaluados de forma que podamos establecer la tasa de aplicación de cada uno de ellos, así como las correspondientes probabilidades con que serán aplicados a cada neurona: p_{r_tuner} y p_{r_random}.

Nuevamente, manteniendo el resto de parámetros a los valores indicados en la tabla 4.1 y utilizando los mecanismos de inicialización de centros y radios, función de evaluación, límite de neuronas y tamaño de torneo determinados en los apartados anteriores, se han realizado experimentos que combinan distintas tasas de aplicación de los operadores con posibles probabilidades de afectar internamente a la red. Los resultados para el operador R_TUNER, media aritmética de los obtenidos tras la ejecución de EvRBF un total de 5 veces por cada par de valores a probar, están reco-

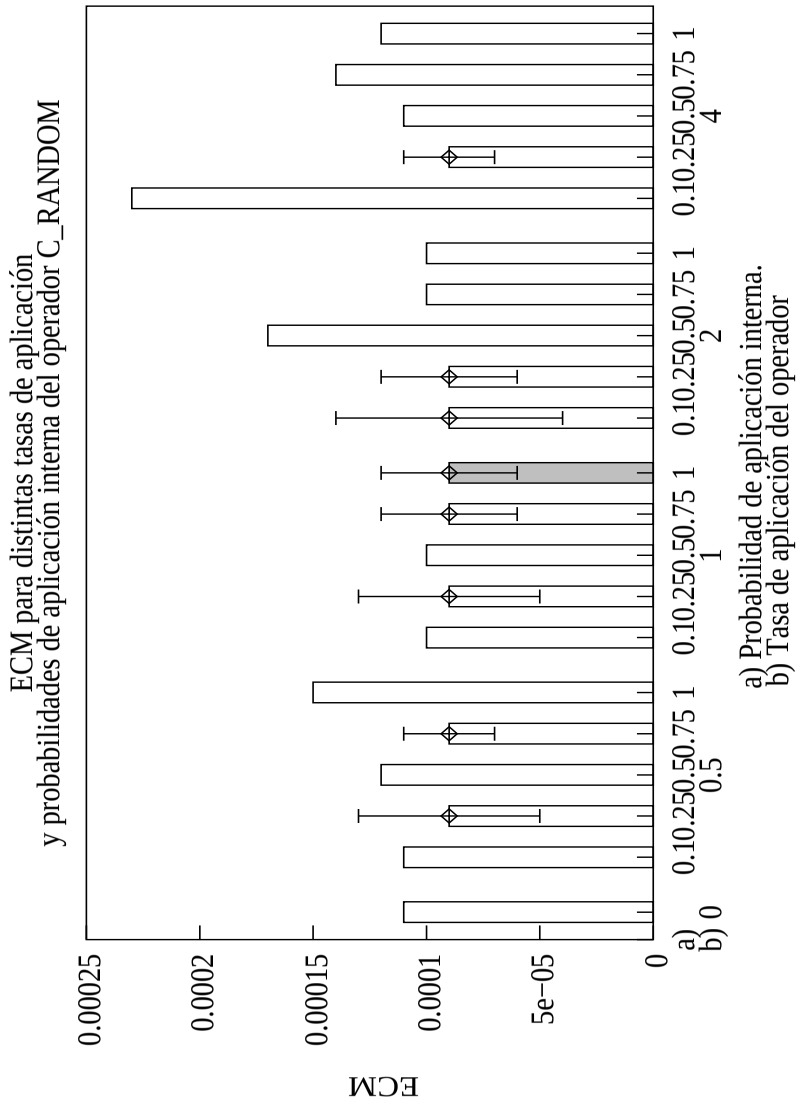


Figura 4.12: Determinación de la tasa de aplicación del operador C_RANDOM. Se representa el ECM promediado a lo largo de 5 ejecuciones del algoritmo, para las distintas combinaciones de *tasa de aplicación-probabilidad de aplicación interna*. Un caso excepcional lo constituye la de tasa de aplicación nula, pues no es necesario establecer probabilidad interna. Para los mejores resultados se ha representado también la desviación estándar. Los mejores resultados se obtienen con varios valores, aunque los que se comportan de forma más homogénea corresponden con tasas de aplicación igual a 1 ó 2.

gidos en la tabla 4.8.5 y en la fig. 4.14. Por su parte, los relativos a R_RANDOM, conseguidos mediante un proceso análogo, se hallan en la tabla 4.16 y en la fig. 4.15.

	Prob. de aplicación Interna				
Tasa Aplicación	*0.1*	*0.25*	*0.5*	*0.75*	*1*
Nula			1.1×10^{-4}		
0.5	0.8×10^{-4}	$\mathbf{0.6 \times 10^{-4}}$	0.7×10^{-4}	0.7×10^{-4}	0.8×10^{-4}
1	0.9×10^{-4}	1.4×10^{-4}	0.8×10^{-4}	1.0×10^{-4}	1.5×10^{-4}
2	1.1×10^{-4}	0.7×10^{-4}	1.0×10^{-4}	1.0×10^{-4}	1.2×10^{-4}
4	1.4×10^{-4}	0.8×10^{-4}	1.0×10^{-4}	1.0×10^{-4}	1.5×10^{-4}

Tabla 4.15: Determinación de la tasa de aplicación del operador R_TUNER. Se presenta el ECM promedio a lo largo de 5 ejecuciones del algoritmo para cada una de las combinaciones *tasa de aplicación-probabilidad de aplicación interna*, salvo en el caso de tasa de aplicación nula en el que no es necesario establecer probabilidad interna. Los mejores resultados se obtienen cuando se usa la combinación $\{0.5, 0.25\}$.

La determinación de los valores con que R_TUNER debe ser empleado en el AE muestra que los valores de aplicación bajos son los que obtienen mejores resultados. Así, la mejor combinación de valores *tasa de aplicación-probabilidad de aplicación interna*, que es la que devuelve el menor error de generalización, es la formada por una tasa de aplicación igual a 0.5 junto con una probabilidad de aplicación interna, p_{r_tuner}, igual a 0.25.

En cuanto al operador R_RANDOM, los resultados de los experimentos (ver tabla 4.16 y fig. 4.15) indican que la tasa de aplicación más idónea es 1, combinada con un factor de aplicación 0.75.

	Prob. de aplicación Interna				
Tasa Aplicación	*0.1*	*0.25*	*0.5*	*0.75*	*1*
Nula			1.0×10^{-4}		
0.5	1.0×10^{-4}	0.9×10^{-4}	0.8×10^{-4}	1.0×10^{-4}	1.5×10^{-4}
1	0.8×10^{-4}	0.9×10^{-4}	0.8×10^{-4}	$\mathbf{0.6 \times 10^{-4}}$	0.7×10^{-4}
2	1.1×10^{-4}	0.9×10^{-4}	1.3×10^{-4}	1.0×10^{-4}	0.7×10^{-4}
4	0.9×10^{-4}	2.0×10^{-4}	0.9×10^{-4}	1.7×10^{-4}	1.4×10^{-4}

Tabla 4.16: Determinación de la tasa de aplicación del operador R_RANDOM. Se presenta el ECM promedio a lo largo de 5 ejecuciones del algoritmo para cada una de las combinaciones *tasa de aplicación-probabilidad de aplicación interna*, salvo en el caso de tasa de aplicación nula en el que no es necesario establecer probabilidad interna. Los mejores resultados se obtienen cuando se usa tasa de aplicación 1, combinada con probabilidad interna, p_{r_random}, igual a 0.75.

Una vez más, la aplicación prudente del operador que modifica los radios trata de buscar a lo largo del espacio los valores más adecuados

que permitan a la red aproximar mejor la función objeto de estudio, pero sin convertir la tarea en una búsqueda totalmente aleatoria.

4.8.6. Tasa de aplicación del operador ADDER

La última serie de experimentos que se han llevado a cabo han permitido estimar las tasas de aplicación de los operadores destinados a modificar el tamaño de las redes. La utilización de esta clase de operadores es, por un lado, necesaria si deseamos configurar todos los detalles de la red, y por el otro, un riesgo al ser un factor determinante de la capacidad de generalización de la misma.

En primer lugar, se ha abordado el estudio del operador que añade neuronas a la red: ADDER. Al igual que ocurría con X_FIX, sólo es necesario establecer la tasa de aplicación del operador, dado que no se aplica ninguna probabilidad interna a cada una de las neuronas. Según el procedimiento ya descrito, se han probado los valores 0, 0.5, 1, 2 y 4 como tasa de aplicación, ejecutando 5 veces el algoritmo con cada uno de ellos. El resto de operadores se han mantenido con tasas de aplicación iguales a 1 y probabilidades de aplicación internas iguales a 0.5, en aquellos casos que eran necesarias. Los resultados se han presentado en la tabla 4.17. Su representación gráfica es la de la fig. 4.16.

Tasa Aplicación	*ECM*	*Tamaño*	*Tiempo*
0	$4.9 \times 10^{-4} \pm 1.4 \times 10^{-4}$	$\mathbf{8} \pm 1$	$\mathbf{10} \pm 1$
0.5	$1.5 \times 10^{-4} \pm 0.7 \times 10^{-4}$	15 ± 3	14 ± 2
1	$1.1 \times 10^{-4} \pm 0.8 \times 10^{-4}$	17 ± 3	16 ± 3
2	$0.6 \times 10^{-4} \pm 0.2 \times 10^{-4}$	24 ± 2	19 ± 1
4	$\mathbf{0.3} \times \mathbf{10^{-4}} \pm 0.1 \times 10^{-4}$	29 ± 2	27 ± 2

Tabla 4.17: Determinación de la tasa de aplicación del operador ADDER. Las columnas muestran los valores promedios de ECM, tamaño de red y tiempo de ejecución a lo largo de 5 ejecuciones de EvRBF para cada uno de los valores de la tasa de aplicación. El menor error de generalización se obtiene cuando la tasa de aplicación es 4, aunque es a costa de un número considerablemente mayor de neuronas, lo cual redunda en mayor coste de tiempo. El valor más equilibrado en cuando a ECM, tamaño y tiempo es 2.

Los resultados muestran en primer lugar que necesitamos utilizar este operador pues en caso contrario el rendimiento del algoritmo baja considerablemente. Por otro lado, la utilización de una tasa de aplicación media-alta, en este caso 2, es la que ofrece el mejor compromiso en cuanto a error, tamaño y tiempo de ejecución. La utilización de una tasa mayor permite obtener mayor capacidad de generalización, aunque se hace a costa de generar redes de mayor tamaño, que a su vez ralentizan la ejecución

del algoritmo. No obstante, la elección del valor final se ha postergado al apartado 4.8.8, en el que se evalúa la acción combinada de este operador de adición de neuronas con el operador que las elimina, DEL_MANY.

4.8.7. Tasa de aplicación de los operadores DEL_MANY y DEL_CLOSEST

Para finalizar, se han evaluado los distintos operadores que permiten reducir el número de neuronas de la capa oculta. De los dos operadores que existen, sólo DEL_MANY necesita ser evaluado con distintos valores de probabilidad de aplicación interna, p_{del_many}, no así DEL_CLOSEST para el cual sólo habremos de especificar una tasa de aplicación.

Comenzando por el más simple de ellos, DEL_CLOSEST, se ha ejecutado 5 veces EvRBF por cada uno de los valores de la tasa de aplicación considerados (ver tabla 4.8), y manteniendo los parámetros recogidos en la tabla 4.1, exceptuando el tamaño del torneo (establecido a 2 desde el apartado 4.7), el límite superior de neuronas (fijado al 10 % del tamaño del conjunto de entrenamiento en el apartado 4.4) y el porcentaje de individuos reemplazados (igual al 70 % desde el apartado 4.6). Los resultados obtenidos para DEL_CLOSEST se encuentran en la tabla 4.18. La interpretación gráfica de dicha tabla se corresponde con la fig. 4.18.

Tasa Aplicación	*ECM*	*Tamaño*	*Tiempo*
0	$\mathbf{0.6 \times 10^{-4}} \pm 0.3 \times 10^{-4}$	18 ± 2	17 ± 2
0.5	$0.9 \times 10^{-4} \pm 0.4 \times 10^{-4}$	17 ± 4	16 ± 1
1	$0.9 \times 10^{-4} \pm 0.2 \times 10^{-4}$	18 ± 3	15 ± 2
2	$1.4 \times 10^{-4} \pm 0.8 \times 10^{-4}$	15 ± 1	14 ± 1
4	$1.4 \times 10^{-4} \pm 0.5 \times 10^{-4}$	$\mathbf{14} \pm 2$	$\mathbf{13} \pm 1$

Tabla 4.18: Determinación de la tasa de aplicación del operador DEL_CLOSEST. Las columnas muestran los valores promedios de ECM, tamaño de red y tiempo de ejecución a lo largo de 5 ejecuciones de EvRBF para cada uno de los valores de la tasa de aplicación. El menor error de generalización se obtiene cuando la tasa de aplicación es 0.

Los resultados generados para el operador DEL_CLOSEST muestran que no es del todo rentable para el algoritmo su utilización. En su ausencia, la mayor aplicación del resto de operadores consiguen que el resultado sea favorable, sin que las diferencias de tamaño sean considerables. Es por ello que se ha decidido eliminar este operador del conjunto mínimo de operadores necesarios para ejecutar EvRBF.

En cuanto al operador DEL_MANY, en la tabla 4.8.7 se pueden ver los valores alcanzados en los experimentos realizados para determinar el me-

jor par *tasa de aplicación-probabilidad de aplicación interna*, tomando los posibles valores para ambos parámetros de las tablas 4.8 y 4.9, respectivamente, y promediados sobre un total de 5 ejecuciones por pareja de parámetros.

	Prob. de aplicación Interna				
Tasa Aplicación	*0.1*	*0.25*	*0.5*	*0.75*	*1*
Nula			1.0×10^{-4}		
0.5	0.7×10^{-4}	0.7×10^{-4}	$\mathbf{0.6 \times 10^{-4}}$	0.7×10^{-4}	0.8×10^{-4}
1	1.3×10^{-4}	1.5×10^{-4}	0.9×10^{-4}	0.8×10^{-4}	1.2×10^{-4}
2	1.0×10^{-4}	1.2×10^{-4}	1.0×10^{-4}	1.0×10^{-4}	1.1×10^{-4}
4	2.3×10^{-4}	1.3×10^{-4}	0.9×10^{-4}	1.2×10^{-4}	1.1×10^{-4}

Tabla 4.19: Determinación de la tasa de aplicación del operador DEL_MANY. Se presenta el ECM promedio a lo largo de 5 ejecuciones del algoritmo para cada una de las combinaciones *tasa de aplicación-probabilidad de aplicación interna*, salvo en el caso de tasa de aplicación nula en el que no es necesario establecer probabilidad interna. Los mejores resultados se obtienen cuando se usa una tasa de aplicación igual a 0.5 y una probabilidad de aplicación interna, p_{del_many}, igual a 0.5.

Los resultados obtenidos para el operador DEL_MANY, recogidos en la tabla 4.8.7 y la fig 4.19, muestran que el mejor error de generalización se consigue cuando la tasa de aplicación es baja, 0.5, combinada con una probabilidad de aplicación interna alta, 0.5. De esta forma, aunque el DEL_MANY sea usado por el AE menos que el resto de operadores, cuando actúa reduce considerablemente el tamaño de la red, permitiendo a ésta comenzar una nueva búsqueda a partir de una topología diferente. No obstante, el siguiente apartado muestra la utilización conjunta de este operador junto a ADDER, para comprobar en qué grado afectan la adición o supresión de neuronas en igual probabilidad.

4.8.8. Tasa de aplicación conjunta de los operadores ADDER y DEL_MANY

Una vez se han estudiado los distintos operadores que influyen en el tamaño de las redes y determinado que el uso del operador DEL_CLOSEST es perjudicial para hallar redes con buena capacidad predictiva, se ha estimado conveniente estudiar en qué modo resulta afectado EvRBF cuando se aplican los operadores ADDER y DEL_MANY en la misma proporción.

Para ello se han realizado experimentos en los que a ambos operadores se ha asignado la misma tasa de aplicación, tomando los valores de la tabla 4.8. Dado que el operador de eliminación necesita de una probabilidad de aplicación interna, se han evaluado las distintas tasas de aplicación

con cada una de las probabilidades internas indicadas en la tabla 4.9. Los resultados de tales experimentos están recogidos en la tabla 4.20, así como en las figs. 4.20 y 4.21.

	Prob. de aplicación Interna				
Tasa Aplicación	*0.1*	*0.25*	*0.5*	*0.75*	*1*
Nula			3.8×10^{-4}		
0.5	1.5×10^{-4}	0.9×10^{-4}	1.5×10^{-4}	1.2×10^{-4}	0.9×10^{-4}
1	1.0×10^{-4}	0.8×10^{-4}	0.8×10^{-4}	0.8×10^{-4}	1.0×10^{-4}
2	0.9×10^{-4}	0.8×10^{-4}	0.5×10^{-4}	0.7×10^{-4}	0.6×10^{-4}
4	0.7×10^{-4}	1.3×10^{-4}	0.5×10^{-4}	$\mathbf{0.4 \times 10^{-4}}$	0.7×10^{-4}

Tabla 4.20: Determinación de la tasa de aplicación conjunta de los operadores ADDER y DEL_MANY. Se presenta el ECM promedio a lo largo de 5 ejecuciones del algoritmo para cada una de las combinaciones *tasa de aplicación-probabilidad de aplicación interna*, salvo en el caso de tasa de aplicación nula en el que no es necesario establecer probabilidad interna. Los mejores resultados se obtienen para las combinaciones de valores $\{4, 0.5\}$, $\{2, 0.5\}$ y $\{4, 0.75\}$, aunque el uso de tasa de aplicación igual a 2 produce redes más pequeñas como muestra la fig. 4.21.

Los resultados ofrecidos por la evaluación combinada de los operadores ADDER y DEl_MANY muestran que el primero tiene más peso sobre el segundo, dado que EvRBF se guía exclusivamente del *fitness* de los individuos, el cual es mayor conforme se aumenta el número de neuronas, al menos mientras no se llegue al caso del sobre-entrenamiento. Es por ello, que una tasa de aplicación igual a 4 es la que obtiene mejor ECM. No obstante, es posible encontrar una capacidad de generalización prácticamente similar, pero usando menos neuronas cuando se utiliza una tasa de aplicación igual a 2 para ambos operadores, junto con una probabilidad de aplicación interna para DEL_MANY igual a 0.5. Finalmente, esta pareja de valores ($\{2, 0.5\}$) son los que van a ser utilizados para ejecutar EvRBF, como muestra la tabla 4.22.

4.9. Conclusiones

En el presente capítulo, y como trabajo necesario para el siguiente, se han detallado los estudios realizados para determinar los valores que deben asignarse a los parámetros con los que EvRBF debe ser ejecutado.

Los experimentos se han llevado a cabo intentando minimizar el sesgo introducido tanto por los datos utilizados, como por las poblaciones de partida. El análisis de los resultados ha permitido determinar cómo debe hacerse la inicialización de la población inicial, tanto en la forma de asignar centros y radios, como en el límite superior que debe imponerse al

número de neuronas de los individuos que la forman. También ha permitido conocer qué método resulta más adecuado para establecer el valor de *fitness* de cada individuo y con qué cantidad de ellos debe llevarse a cabo el proceso de selección para la posterior reproducción.

Finalmente, el conjunto de estudios realizado también ha servido para determinar con qué tasa de aplicación se debe emplear cada uno de los operadores genéticos en el algoritmo. E igualmente, para aquellos operadores que requieren el establecimiento de una probabilidad de aplicación interna, se han evaluado todas las posibles combinaciones de ambos parámetros lo que ha permitido precisar cuál de ellas es la más adecuada.

A modo de resumen, se han recogido en las tablas 4.21 y 4.22 los valores finales para los parámetros con que será ejecutado EvRBF en los experimentos presentados en el próximo capítulo. La primera de ellas muestra dichos valores, excluyendo los pertenecientes a los operadores genéticos, esto es, las distintas tasas de aplicación y la probabilidades de aplicación interna. Estos valores son los indicados en la tabla 4.22.

Parámetro	*Valor*
Tamaño población	100
Generaciones	100
Límite neuronas primera generación	$10\,\%N^*_{Tr}$
Inicialización de centros	IC_PAT
Inicialización de radios	IR_MIN_DIS2
Población reemplazada	30 %
Tamaño torneo	2
Función evaluación	EV_TRN_VAL

Tabla 4.21: Parámetros de ejecución de EvRBF. Los valores relativos a las tasas de aplicación y probabilidades de aplicación interna correspondientes a los operadores genéticos se han recogido en la tabla 4.22.

Operador	*Tasa Aplicación*	*Prob. Interna*
X_FIX	1	-
X_MULTI	1	0.25
X_AVERAGE	0.5	0.75
C_TUNER	0.5	0.75
C_RANDOM	1	1
R_TUNER	0.5	0.25
R_RANDOM	1	0.75
ADDER	2	-
DEL_CLOSEST	0	-
DEL_MANY	2	0.5

Tabla 4.22: Valores de tasa de aplicación y probabilidad de aplicación interna para los operadores genéticos con que se ejecutará EvRBF. DEL_CLOSEST no será utilizado al haberse establecido a 0 el valor más idóneo para su tasa de aplicación.

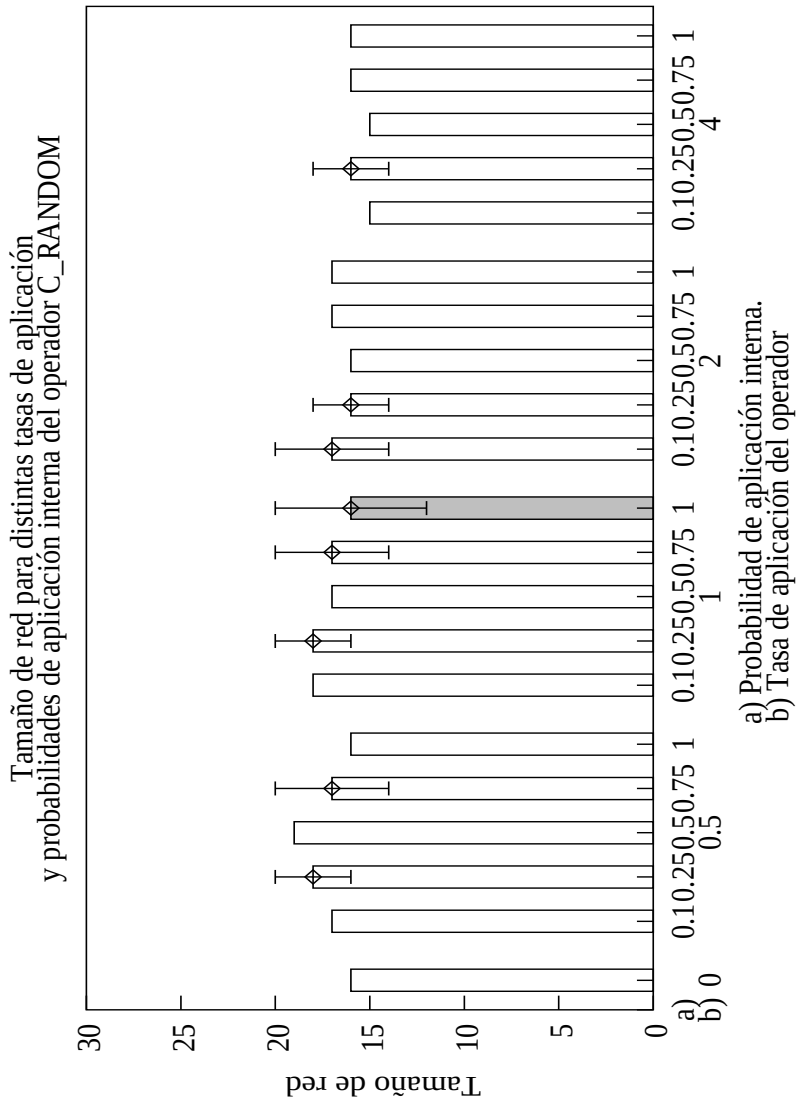


Figura 4.13: Determinación de la tasa de aplicación del operador C_RANDOM. Se representa el tamaño promediado a lo largo de 5 ejecuciones del algoritmo, para las distintas combinaciones de *tasa de aplicación-probabilidad de aplicación interna*. Un caso excepcional lo constituye la de tasa de aplicación nula, pues no es necesario establecer probabilidad interna. Para las combinaciones de los parámetros que devolvían mejor ECM, se ha añadido también la desviación estándar. La utilización de tasa de aplicación igual a 1 y probabilidad p_{c_random} igual a 1, o bien tasa de aplicación igual a 2 y probabilidad p_{c_random} igual a 0.25 proporcionan redes cercanas a las más pequeñas, a la vez que obtienen el menor ECM (ver tabla 4.14).

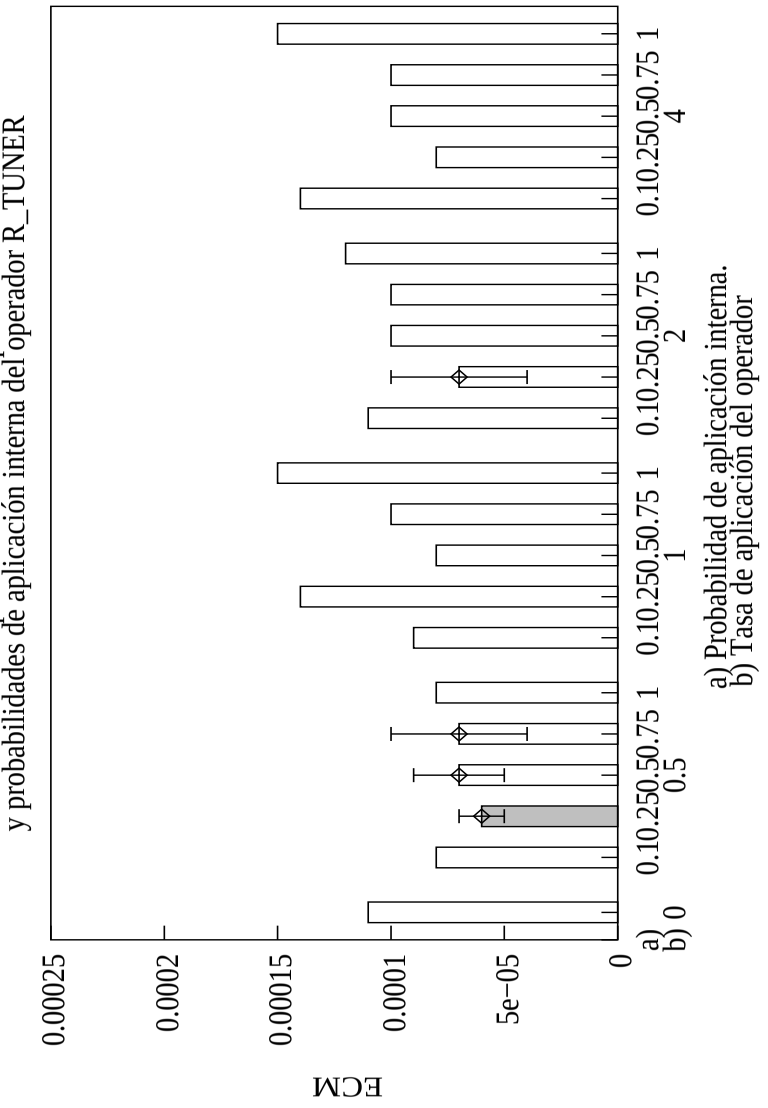


Figura 4.14: Determinación de la tasa de aplicación del operador R_TUNER. Se representa el ECM promediado a lo largo de 5 ejecuciones del algoritmo, para las distintas combinaciones de *tasa de aplicación-probabilidad de aplicación interna*. Un caso excepcional lo constituye la de tasa de aplicación nula, pues no es necesario establecer probabilidad interna. Para los mejores resultados se ha representado también la desviación estándar. El ECM más pequeño se obtiene para la combinación $\{0.5, 0.25\}$.

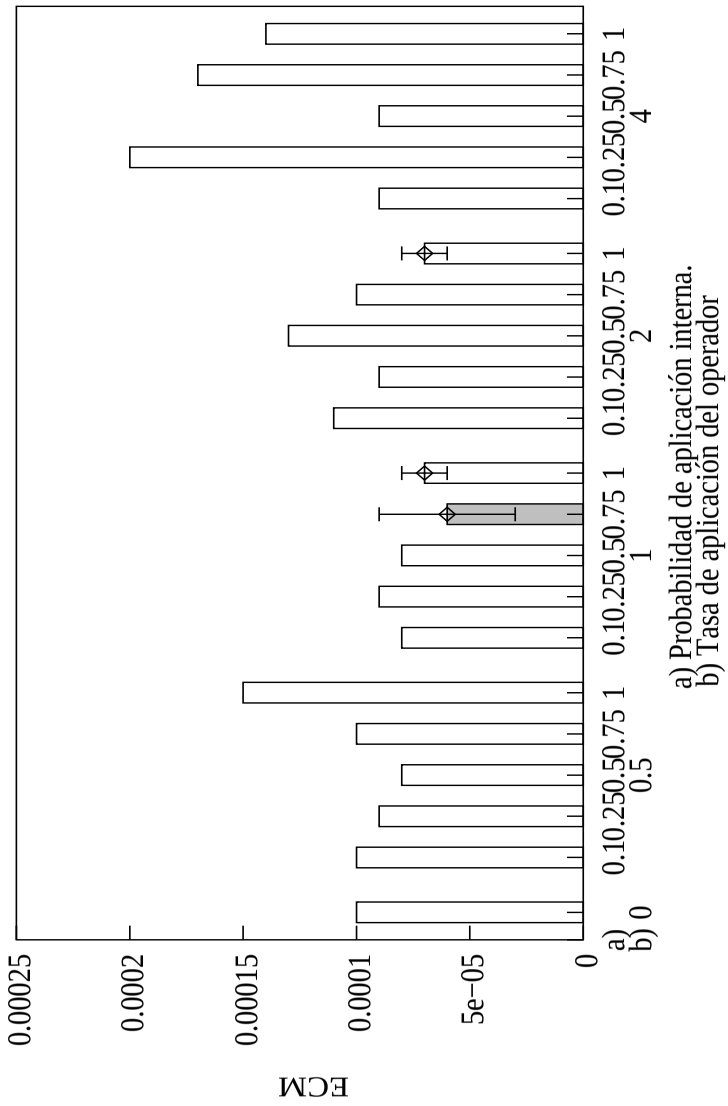


Figura 4.15: Determinación de la tasa de aplicación del operador R_RANDOM. Se representa el ECM promediado a lo largo de 5 ejecuciones del algoritmo, para las distintas combinaciones de *tasa de aplicación-probabilidad de aplicación interna*. Un caso excepcional lo constituye la de tasa de aplicación nula, pues no es necesario establecer probabilidad interna. Para los mejores resultados se ha representado también la desviación estándar. El mejor resultado se obtiene cuando se usa una tasa de aplicación igual a 1, junto con una probabilidad interna, p_{r_random}, igual a 0.75.

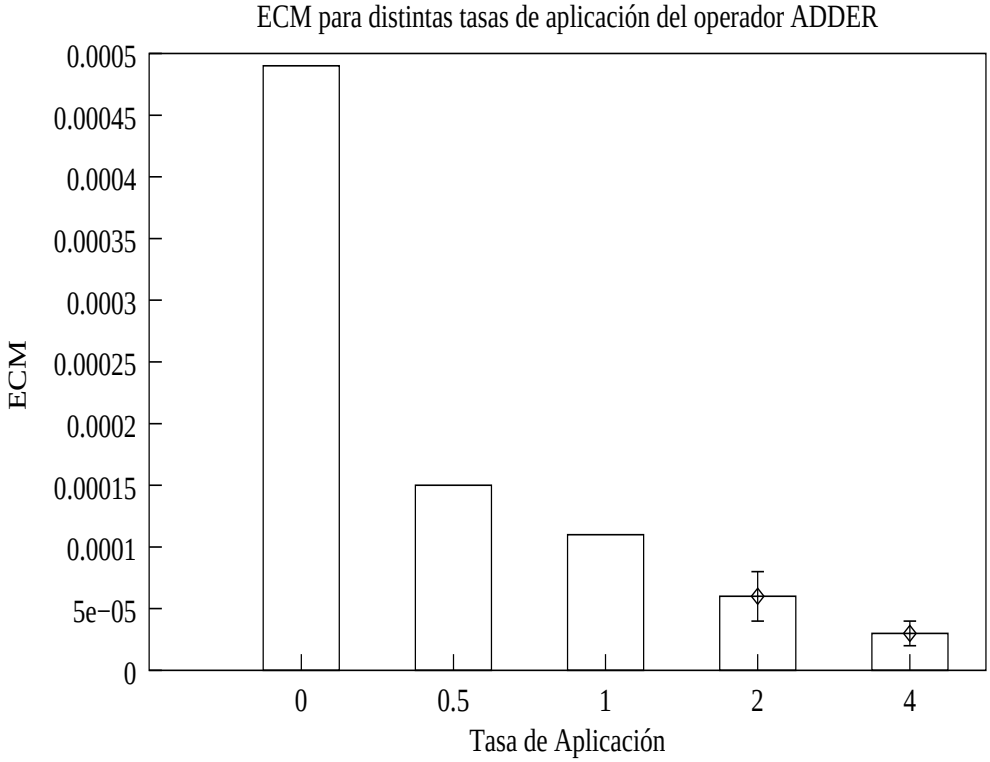


Figura 4.16: Determinación de la tasa de aplicación del Operador ADDER. Se muestra el ECM promedio calculado sobre 5 ejecuciones para distintos valores de tasa de aplicación y la desviación estándar para los errores más pequeños. El mejor resultado se consigue cuando la tasa de aplicación es establecida a 4, aunque se consigue un mejor equilibrio entre ECM y tamaño de redes cuando se utiliza un tasa igual a 2, como muestra la fig. 4.17. En cualquier caso, la elección de los valores finales se ha postergado al apartado 4.8.8 en el que e evalúa la acción combinada de ADDER con el operador de eliminación DEL_MANY

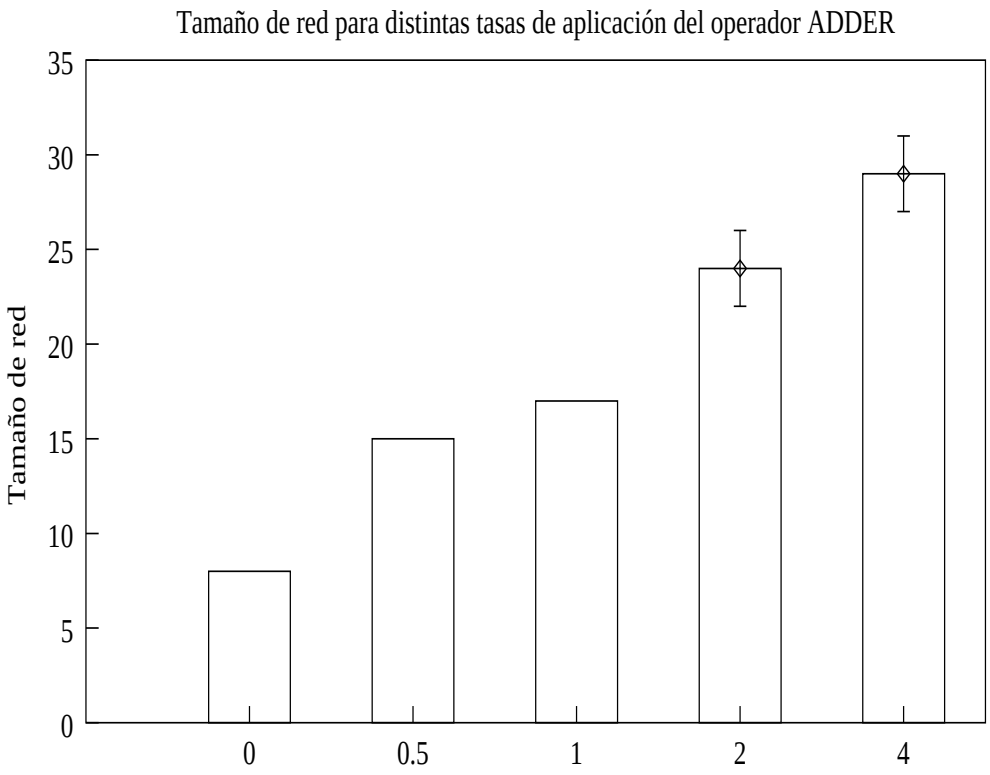


Figura 4.17: Determinación de la tasa de aplicación del Operador ADDER. Se muestra el tamaño de redes promedio calculado sobre 5 ejecuciones para distintos valores de tasa de aplicación y, para aquellos valores que devolvían el menor ECM, se muestra también la tasa de aplicación. Una tasa de aplicación igual a 2 representa un buen compromiso entre el ECM obtenido (ver fig. 4.16) y el tamaño de las redes finales.

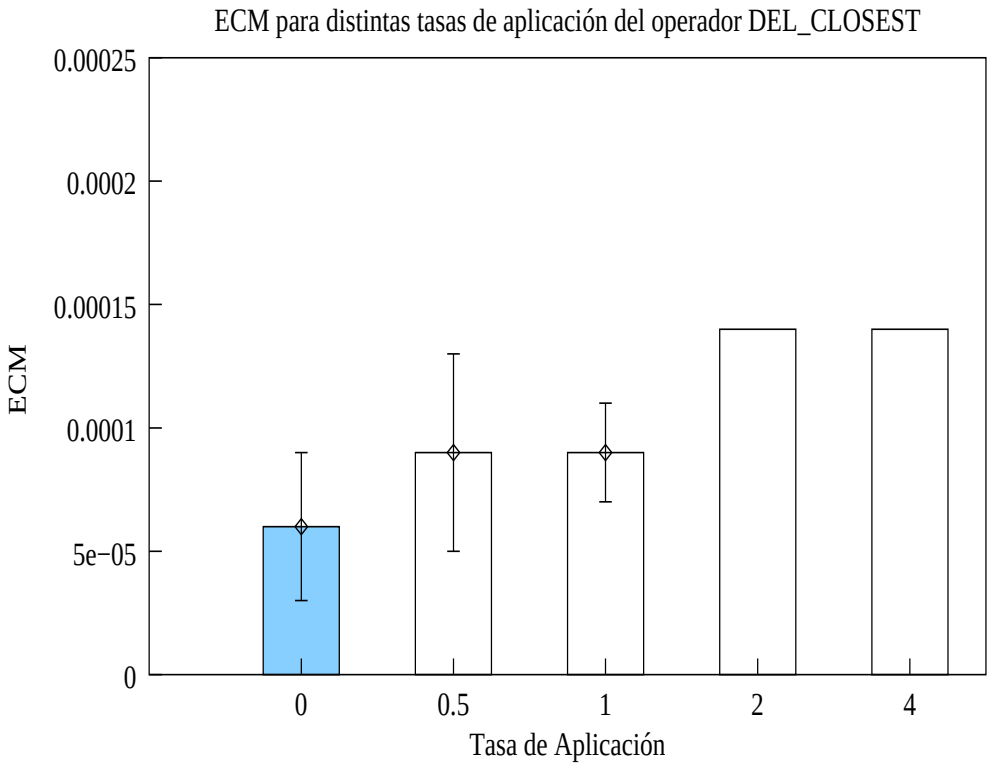


Figura 4.18: Determinación de la tasa de aplicación del Operador DEL_CLOSEST. Se muestra el ECM promedio calculado sobre 5 ejecuciones para distintos valores de tasa de aplicación y la desviación estándar de los mejores resultados. El error más pequeño se consigue cuando la tasa de aplicación es establecida a 0.

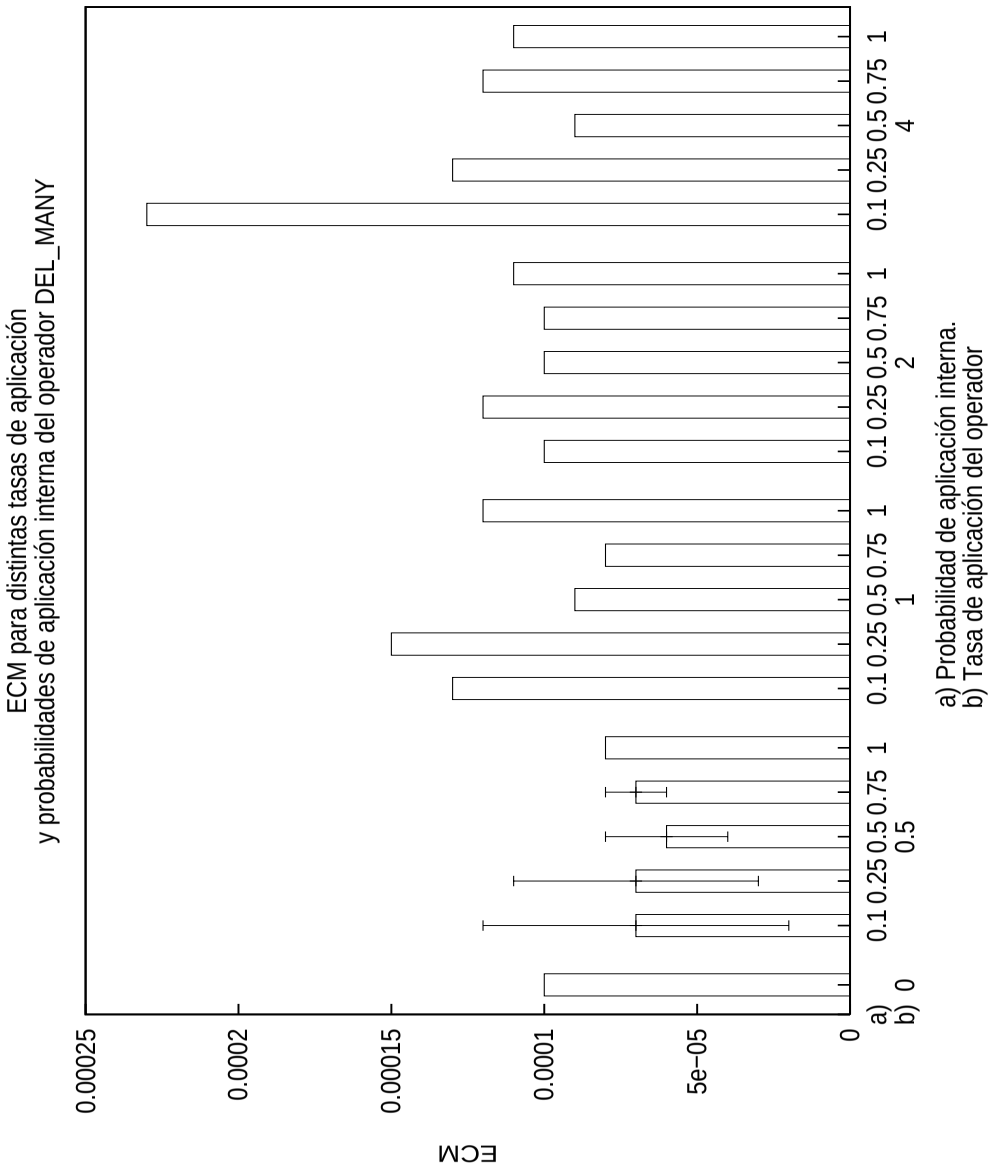


Figura 4.19: Determinación de la tasa de aplicación del operador DEL_MANY. Se representa el ECM promediado a lo largo de 5 ejecuciones del algoritmo, para las distintas combinaciones de *tasa de aplicación-probabilidad de aplicación interna*. Un caso excepcional lo constituye la de tasa de aplicación nula, pues no es necesario establecer probabilidad interna. Para los mejores resultados se ha representado también la desviación estándar. El error más pequeño se obtiene para tasa de aplicación igual a 0.5 y probabilidad de aplicación interna, p_{del_many}, igual a 0.5.

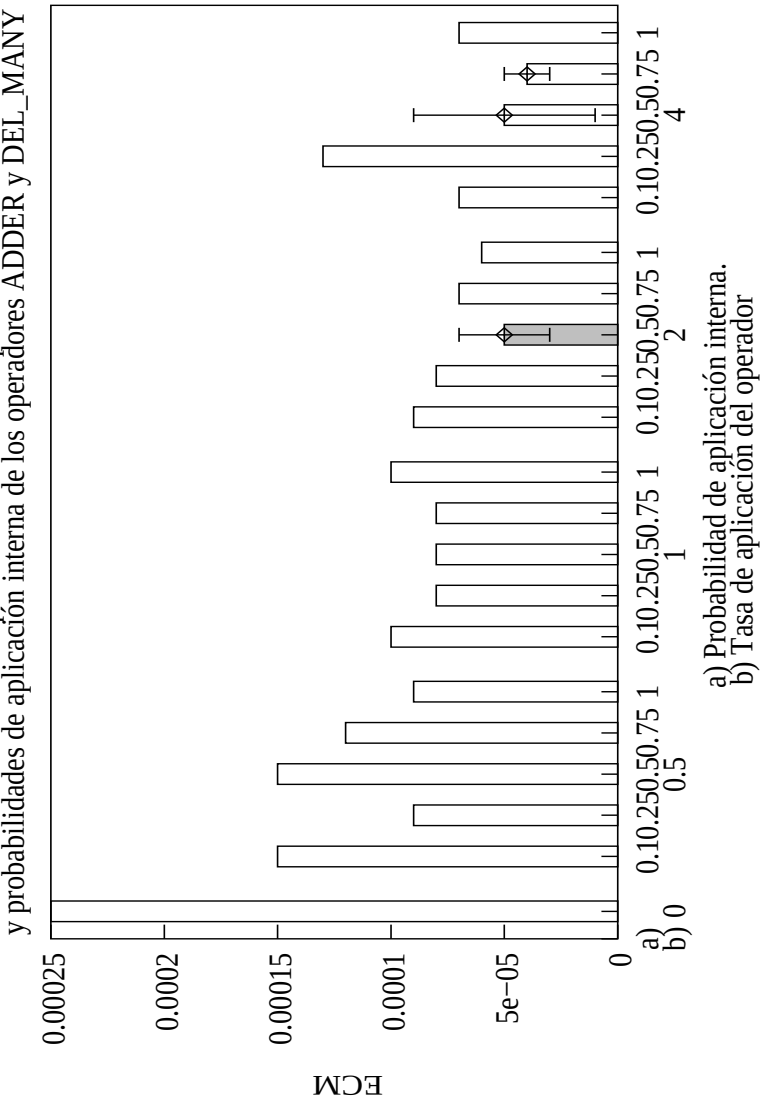


Figura 4.20: Determinación de la tasa de aplicación conjunta de los operadores ADDER y DEL_MANY. Se representa el ECM promediado a lo largo de 5 ejecuciones del algoritmo, para las distintas combinaciones de *tasa de aplicación-probabilidad de aplicación interna*. Un caso excepcional lo constituye la de tasa de aplicación nula, pues no es necesario establecer probabilidad interna. Para aquellos valores que obtienen los mejores resultados se ha representado también la desviación estándar. Los errores más pequeños se obtienen para las combinaciones de valores $\{4, 0.5\}$, $\{2, 0.5\}$ y $\{4, 0.75\}$. No obstante, el uso de tasa de aplicación igual a 2 produce redes más pequeñas como muestra la fig. 4.21.

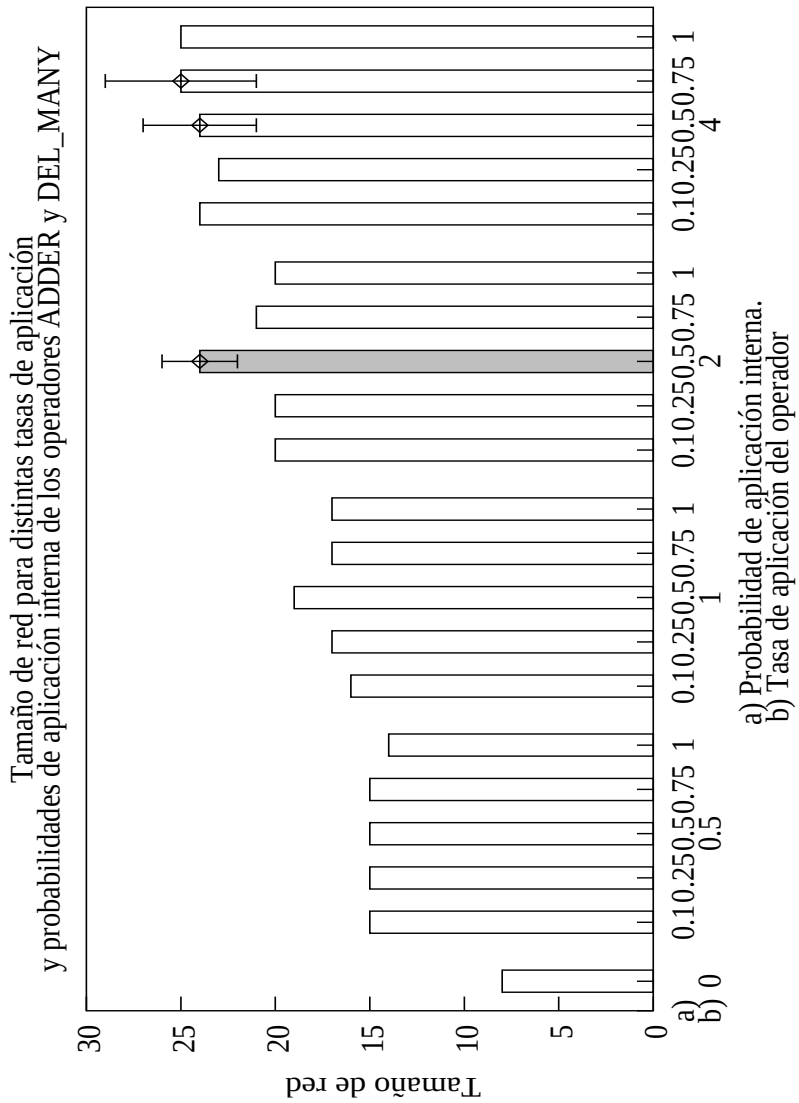


Figura 4.21: Determinación de la tasa de aplicación conjunta de los operadores ADDER y DEL_MANY. Se representa el tamaño de red promediado a lo largo de 5 ejecuciones del algoritmo, para las distintas combinaciones de *tasa de aplicación-probabilidad de aplicación interna*. Un caso excepcional lo constituye la de tasa de aplicación nula, pues no es necesario establecer probabilidad interna. Para las combinaciones que ofrecían los errores más pequeños de ECM (ver fig. 4.20) se ha representado también la desviación estándar. Los errores más pequeños se obtenían para las combinaciones de valores $\{4, 0.5\}$, $\{2, 0.5\}$ y $\{4, 0.75\}$, y este gráfico muestra que, de entre ellos, la tasa de aplicación igual a 2 es la que produce redes más pequeñas.

CAPÍTULO 5 / EVALUACIÓN DEL MÉTODO EvRBF

En este capítulo se detallan el comportamiento que ofrece EvRBF cuando es aplicado a tareas de aproximación funcional, clasificación de patrones y estimación de series temporales. Para ello, se han seleccionado problemas, en su mayor parte sintéticos, que han sido utilizados previamente en la literatura para evaluar algoritmos que trataban la generación automática de RNFBR.

Los distintos problemas con los que se ha trabajado han sido agrupados por secciones. Así, encontraremos la sección de aproximación funcional, en la que se han abordado problemas de distinto número de variables de entrada, así como distinta cantidad de ruido añadido a las salidas. En cuanto a la clasificación de patrones, se han utilizado conocidas bases de datos, como son la de plantas *Iris*, la de enfermedades del corazón (*Heart Disease*) o la de *Cáncer de pulmón*. Todas ellas son fácilmente accesibles y frecuentemente utilizadas para medir la bondad de los nuevos algoritmos que surgen. Finalmente, para la estimación de series temporales se han incluido las series *mapa doble* y *mapa cuadrático* que usaran Broomhead y Lowe en su trabajo inicial, así como la serie de Mackey y Glass que es ampliamente utilizada por los investigadores para comparar sus algoritmos.

5.1. Metodología empleada

El sistema que se ha utilizado para evaluar EvRBF en todos estos problemas ha sido siempre similar, estando cada uno de ellos definido mediante los correspondientes ficheros de entrenamiento y test.

Para cada uno de los problemas considerados se han realizado diversas ejecuciones de EvRBF, variando el número de generaciones para el AE. El resto de parámetros han sido los mostrados en las tablas 4.21 y 4.22 del capítulo anterior. Los números de generaciones considerados han sido: 10, 25, 50, 75 y 100. Además, para cada una de estas cifras el algoritmo ha sido ejecutado 5 veces, partiendo en cada una de ellas de poblaciones iniciales distintas, por ser generadas utilizando los métodos IC_PAT e IR_MIN_DIS2 para inicialización de centros y radios, respectivamente.

En aquellos casos en los que ficheros de entrenamiento y test a generar dependen de factores aleatorios (en cuanto a los valores de las variables de entrada), se han generado 5 ficheros de entrenamiento y test distintos; esto es, uno por cada una de las ejecuciones de EvRBF con un determinado número de generaciones. De esta forma se ha intentado minimizar el efecto del posible sesgo introducido por conjuntos de datos determinados.

Los resultados que se ofrecen en los distintos apartados se recogen, en sendas tablas, dos por cada problema considerado. La primera de dichas tablas muestra los resultados obtenidos por EvRBF para el problema en cuestión al que es aplicado. Por su parte, la segunda tabla permite comparar los resultados obtenidos por distintos algoritmos aplicados a cada problema junto con los de EvRBF.

De los resultados propios de EvRBF (primera de las tablas) se consignan el error obtenido, el tamaño de red, el porcentaje de parámetros que dicho tamaño representa con respecto al número de datos presente en los conjuntos de entrenamiento y test y, finalmente, el tiempo de ejecución del algoritmo, expresado en segundos. Estos datos se detallan para los distintos números de generaciones empleados y han sido promediados a partir de las 5 ejecuciones realizadas por problema y número de generaciones. Además, de las 100 redes generadas por EvRBF en cada ejecución, se ha tenido en cuenta sólo la red que mejor aproximaba el conjunto de test, una vez finalizado el AE y entrenadas todas las redes de la última generación con el conjunto de entrenamiento al completo.

La medida del error que se ofrece varía de unos problemas a otros para poder así comparar con los resultados presentes en la literatura. De este modo, se ha utilizado el ECM o alguna variante del mismo para problemas de aproximación funcional y estimación de series temporales. La variantes son el ECM normalizado o ECMN (esto es, el ECM dividido por la

desviación estándar de los valores de salida del conjunto de test), la raíz cuadrada del ECM o RECM e, incluso, la raíz cuadrada del ECM normalizado (RECMN).

En cuanto a las columnas etiquetadas como *tamaño de red*, hacen referencia al número de neuronas ocultas que componen la RNFBR. En función de este tamaño se puede calcular el **número de parámetros libres** que componen la red. Así, dada una red de tamaño p', el número de parámetros libres (pl) que la componen se calcula como:

$$pl = 3np' + n'p' \tag{5.1}$$

donde n es la dimensión del espacio de entradas y n' es la dimensión del espacio de salidas. La forma concreta de la ec. 5.1 se debe a que por cada neurona existe un centro de dimensión n, un radio de dimensión $2n$ y n' pesos de conexión de dicha neurona a las neuronas de salida.

Por su parte, el porcentaje que pl parámetros libres ($porc(pl)$) suponen sobre el número de datos presentes en los conjuntos de entrenamiento y test se calcula como:

$$porc(pl) = \frac{pl}{(n+n')(N_{tr}+N_{te})} * 100 \tag{5.2}$$

donde N_{tr} y N_{te} son, respectivamente, el número de patrones que forman el conjunto de entrenamiento y el conjunto de test.

Atendiendo a las reglas heurísticas utilizadas con frecuencia en la resolución de problemas como los que nos ocupan, se considerarán redes válidas aquellas cuyo $porc(pl)$ represente en torno al 10 % o menos del número de datos disponibles. Como se verá posteriormente, algunas ejecuciones de EvRBF han generado redes con excesivos parámetros libres que, por este motivo, no han sido incluidas en las tablas de comparación con otros algoritmos.

Con respecto al tamaño de las redes, dentro del apartado 5.2.2, dedicado a la aplicación de EvRBF a la función polinomial *Hermite*, se han incluido figuras en las que se representa la evolución del mismo en ejecuciones típicas de EvRBF. Las conclusiones obtenidas a partir de dichas figuras son aplicables al resto de problemas considerados.

Por último, el resultado referente al tiempo de ejecución de EvRBF mide el tiempo transcurrido desde que se ejecuta la primera instrucción del algoritmo hasta que se ejecuta la última. Esto incluye la creación de los distintos objetos definidos en la biblioteca **EO**, creación, entrenamiento y evaluación de los individuos de la primera generación, la ejecución del AE, el entrenamiento de las redes de la última generación y la evaluación

de las mismas utilizando el conjunto de test. Todos los experimentos han sido ejecutados en un ordenador personal, dotado de un microprocesador AMD Athlon 2100+ y 512 MB de memoria RAM, bajo sistema operativo Linux (versión de kernel 2.4.18) y sin ejecutar simultáneamente aplicaciones de usuario, esto es, con la carga exclusiva de las tareas rutinarias que realiza el S.O.

En cuanto a las tablas comparativas, se hace referencia a los errores obtenidos por los distintos algoritmos y, cuando es posible, al tamaño de red con el que se obtuvo dicho error. En el caso de los problemas de aproximación funcional f_1 a f_4 (apartados 5.2.1.1 a 5.2.1.4), para el método de Pomares se indica el número de reglas que genera, dado que es un método que genera un sistema difuso para resolver el problema. A partir del número de reglas, se pude calcular el número de parámetros libres como:

$$pl = 2p' - 2 \tag{5.3}$$

siendo p' el número de reglas. Tales parámetros se corresponden con los límites del rango en que está definida cada regla (2 por cada regla), estando el límite inferior de la primera y el superior de la última impuestos por la definición del propio problema.

5.2. Evaluación de EvRBF en problemas de aproximación funcional

La primera serie de experimentos realizados trata el problema de la aproximación de funciones. Este tipo de problema simula la tarea de prever la salida que ofrecerá un determinado sistema que dependa de un conjunto de variables de entrada, sin que se conozca exactamente cuál es el papel que juega cada una de dichas variables.

Los distintos casos que se han contemplado incluyen funciones de una sola variable de entrada (apartados 5.2.1.1 a 5.2.1.4), funciones de una variable de entrada con ruido añadido a la salida (apartado 5.2.2) y funciones de múltiples variables de entrada con ruido añadido en la salida (apartado 5.2.3).

5.2.1. Aproximación de las funciones f_1, f_2, f_3 y f_4.

Este primer conjunto de funciones ha sido previamente utilizado en trabajos como [Gonz01a, Rive02, Poma00] y pretenden determinar la capacidad que tiene el método para emular el comportamiento de sistemas

que dependen de una sola variable de entrada y devuelven a su vez una sola salida. Su elección viene determinada principalmente por el hecho de tener varios puntos de inflexión, por lo que la red debe configurarse de modo que se adapte a cada uno de los distintos máximos y mínimos locales existentes a lo largo del rango del espacio de entrada en que está definida la variable independiente.

Para evaluar cada una de estas funciones, se ha generado un fichero de entrenamiento compuesto por 100 patrones y un fichero de test que contiene 1000 patrones. En todos los casos, la variable de entrada toma valores equidistribuidos dentro del rango en el que cada función ha sido definida. Una vez generados los ficheros, han sido normalizados y reescalados, como ya se hiciera en el anterior capítulo y como queda descrito en el apartado 3.4.1 (pág. 75).

Los métodos con los que se van a comparar los resultados sobre estas funciones se encuentran recogidos en [Poma00], [Gonz01a] y [Rive02]. En el primero de ellos, Pomares realiza la aproximación de la función utilizando lógica difusa. Los otros dos, sin embargo, emplean RNFBR y han sido ya tratados en el capítulo 2, apartado 2.3.3. Así, [Gonz01a] utiliza un AE y operadores aplicados a RNFBR para hallar la solución más adecuada; mientras que [Rive02] utiliza lógica difusa para determinar qué operaciones se deben aplicar a las redes para mejorar su comportamiento.

Dado que todos los autores utilizan para dar sus resultados la medida del ECM, éste ha sido también el modo elegido para medir el error ofrecido por EvRBF y comparar los distintos métodos. Por este mismo motivo, la única excepción la constituye la última función en la cual se ha utilizado la raíz cuadrada del ECM normalizado (RECMN).

5.2.1.1. Función $f_1 = \text{sen}(2\pi x)$

El primer problema al que ha sido aplicado EvRBF es la aproximación de la función f_1, definida según:

$$f_1(x) = \text{sen}(2\pi x) \quad ; \quad x \in [-1, 1] \tag{5.4}$$

Para generar los ficheros de entrenamiento y test se han asignado valores equidistribuidos en el rango $[-1, 1]$ a la variable de entrada, X; por su parte, la variable de salida es el resultado de aplicar directamente la ec. 5.4 a cada una de dichas entradas, sin añadir ningún ruido adicional. La fig. 5.1 muestra gráficamente las salidas correspondientes a cada uno de los valores de entrada de los conjuntos de entrenamiento y test.

La tabla 5.1 muestra los resultados obtenidos por EvRBF en función

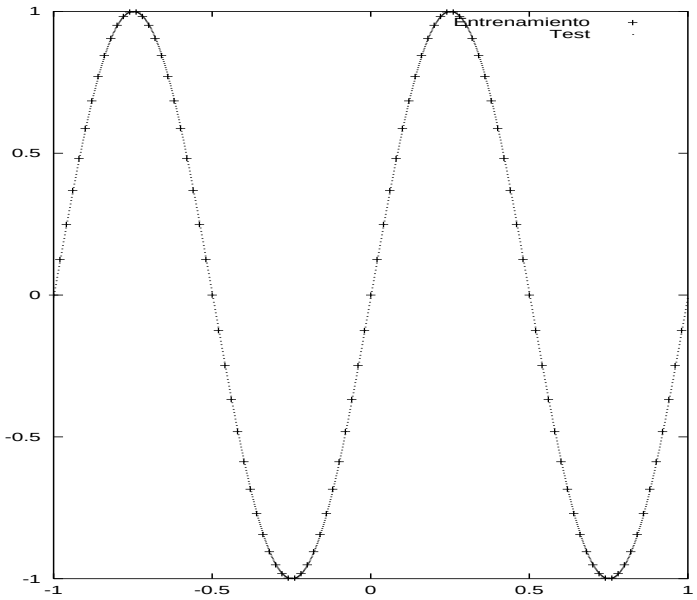


Figura 5.1: Representación de los ficheros de entrenamiento y test generados para la función f_1, definida en la ec. 5.4.

del número de generaciones durante las que se itera el AE. En dicha tabla se observa el error promedio cometido por el algoritmo para distinto número de generaciones e, igualmente, el tamaño de red, con el porcentaje de parámetros que dicho tamaño supone, así como el tiempo de ejecución expresado en segundos. Por su parte, la tabla 5.2 muestra los resultados obtenidos comparándolos con los hallados por los métodos ya citados.

Los resultados mostrados en la tabla 5.1 muestran que, para esta función, la utilización de un mayor número de generaciones conlleva mejora en el error alcanzado por EvRBF, es decir, no se llega a sobre-entrenar en

Generaciones	*ECM*	*Neuronas*	*Parámetros*	*Tiempo (segs.)*
10	5.0e-4±3.7e-4	9±1	1.80 %	2.4±0.5
25	1.7e-5±1.6e-5	14±2	2.80 %	5.4±0.5
50	1.5e-7±1.1e-7	21±2	4.20 %	13.8±0.8
75	5.5e-8±6.1e-8	27±2	5.40 %	28.2±1.3
100	9.2e-9±1.2e-8	29±4	5.80 %	42.2±6.8

Tabla 5.1: Resultados de EvRBF en la aproximación de la función f_1 definida en la ec. 5.4. Se muestra el ECM, tamaños de red, porcentaje de parámetros libres que supone cada tamaño con respecto al número de datos presentes en los conjuntos de entrenamiento y test, y tiempo de ejecución expresado en segundos, todo ello para distinto número de generaciones en el AE.

Algoritmo		*ECM*	*Tamaño Red / Número de Reglas*
Pomares		0.026	10
González		0.0028	6
Rivera		0.08	7
EvRBF	*10 gen.*	5.0e-4	9
EvRBF	*25 gen.*	1.7e-5	14
EvRBF	*50 gen.*	1.5e-7	21
EvRBF	*75 gen.*	5.5e-8	27
EvRBF	*100 gen.*	**9.2e-9**	29

Tabla 5.2: Comparación del ECM conseguido por los métodos descritos en [Poma00], [Gonz01a], [Rive02] y por EvRBF en la aproximación de la función f_1, definida en la ec. 5.4. Se incluye ECM y el tamaño para distintos números de generaciones en EvRBF. Se comprueba que el método presentado en esta tesis consigue mejores aproximaciones con redes de tamaño comparable a las del resto de autores y permite disminuir el error cometido conforme se le permite agregar nuevas neuronas a las redes que genera.

ningún momento a las redes. No obstante, la disminución de dicho error se consigue a costa de redes cada vez mayores, aunque sin llegar a sobrepasar en ningún caso el límite del 10 % que consideramos máximo para que el modelo sea válido. Destaca también el hecho de que los tiempos de ejecución sean muy cortos, y ello a pesar de que en cada generación 30 nuevos individuos son generados, les son calculados sus pesos óptimos mediante SVD y son evaluados para asignarles un valor de *fitness*.

En cuanto a la comparación de los datos obtenidos por EvRBF con respecto al resto de métodos, en la tabla 5.2 podemos observar que aunque el mejor error se consigue tras 100 generaciones, bastan 10 para que nuestro método consiga mejores resultados, con un tamaño de red no mucho mayor al de las redes obtenidas por González y Rivera e inferior al número de reglas difusas utilizado por Pomares.

5.2.1.2. Función $f_2 = 3x(x-1)(x-1.9)(x+0.7)(x+1.8)$

El siguiente problema de aproximación funcional al que se ha aplicado EvRBF consiste en la estimación de valores para la función f_2, definida como:

$$f_2 = 3x(x-1)(x-1.9)(x+0.7)(x+1.8) \quad ; \quad x \in [-2.1, 2.1] \tag{5.5}$$

Nuevamente, se han utilizado un conjunto de 100 patrones para entrenar las redes y otro de 1000 para comprobar la capacidad de generalización

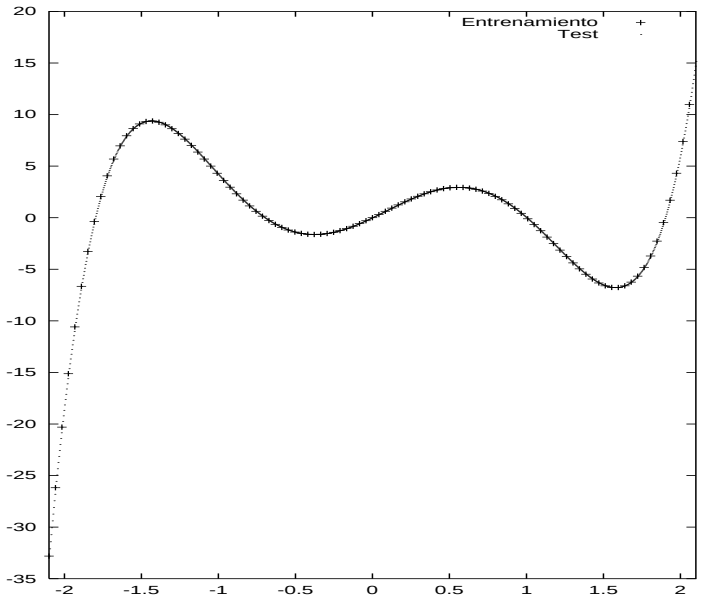


Figura 5.2: Representación de los ficheros de entrenamiento y test generados para la función f_2, definida en la ec. 5.5.

de cada una de ellas. La variable independiente toma valores equidistribuidos en el rango $[-2.1, 2.1]$. Los distintos puntos utilizados para entrenamiento y test, con sus respectivas salidas, se representan en la fig. 5.2.

Al igual que en el caso de la función f_1, los datos mostrados en la tabla 5.3 reflejan que para esta función se pueden llegar a ejecutar hasta 100 iteraciones del AE sin que exista sobre-entrenamiento de las redes generadas. Así, la columna del ECM muestra una progresiva mejora del error conforme se alarga el período de ejecución del algoritmo, aunque, nuevamente, se hace en detrimento del tamaño promedio de las redes. Los tiempos de

Generaciones	*ECM*	*Neuronas*	*Parámetros*	*Tiempo (segs.)*
10	0.4±0.2	7± 1	1.40 %	2.2±0.4
25	0.0030±0.0008	11±2	2.20 %	5.2±0.4
50	5.1e-5±3.6e-5	17±3	3.40 %	13.8±1.1
75	1.0e-5±0.6e-5	21±3	4.20 %	23.0±2.9
100	5.2e-6±6.7e-6	26±3	5.20 %	37.4±3.5

Tabla 5.3: Resultados de EvRBF en la aproximación de la función f_2, definida en la ec. 5.5. Las distintas columnas muestran el ECM, tamaños de red y porcentaje de parámetros libres que dicho tamaño supone con respecto al número de datos presentes en los conjuntos de entrenamiento y test, y tiempo de ejecución expresado en segundos, todo ello para distinto número de generaciones en el AE.

Algoritmo		*ECM*	*Tamaño Red / Número de Reglas*
Pomares		0.46	7
González		0.30	5
Rivera		0.33	7
EvRBF	*10 gen.*	0.4	7
EvRBF	*25 gen.*	0.003	11
EvRBF	*50 gen.*	5.1e-5	17
EvRBF	*75 gen.*	1.0e-5	21
EvRBF	*100 gen.*	**5.2e-6**	26

Tabla 5.4: Comparación del ECM conseguido por los métodos descritos en [Poma00], [Gonz01a], [Rive02] y por EvRBF en la aproximación de la función f_2, definida en la ec. 5.5. Se incluye ECM y el tamaño para distintos números de generaciones en EvRBF. Los resultados demuestran que, para un número de neuronas similar, EvRBF alcanza errores comparables al resto de los métodos. No obstante, es capaz de generar redes con mejores prestaciones a medida que se añaden neuronas, llegando a alcanzar tasas de error muy pequeñas.

ejecución son similares a los del problema de la aproximación de f_1 por tener ambos las mismas características.

La comparación que se puede realizar entre los distintos métodos hallados en la literatura se muestra en la tabla 5.4. Como puede observarse, EvRBF alcanza errores comparables al resto de los métodos para tamaños de redes también comparables, lo cual ocurre utilizando únicamente 10 generaciones. No obstante, en cuanto se le permite continuar el proceso de búsqueda, el algoritmo es capaz de generar redes con mejores prestaciones a medida que se añaden neuronas, llegando a alcanzar tasas de error muy pequeñas. De hecho, basta utilizar 25 generaciones, en vez de 10, para encontrar redes que son entre 2 y 1.05 veces mayores que las encontradas por los demás métodos, pero con un ECM 100 veces mejor.

5.2.1.3. Función $f_3 = 3e^{-x^2}\operatorname{sen}(\pi x)$

El siguiente problema de aproximación funcional al que se ha aplicado EvRBF consiste en la estimación de valores para la función f_3, definida como:

$$f_3 = 3e^{-x^2}\operatorname{sen}(\pi x) \quad ; \quad x \in [-3,3] \tag{5.6}$$

que genera los ficheros de entrenamiento y test mostrados gráficamente en la fig. 5.3.

El comportamiento de EvRBF en la aproximación de la función f_3, reflejado en la tabla 5.5, es similar a los anteriores casos. A medida que se

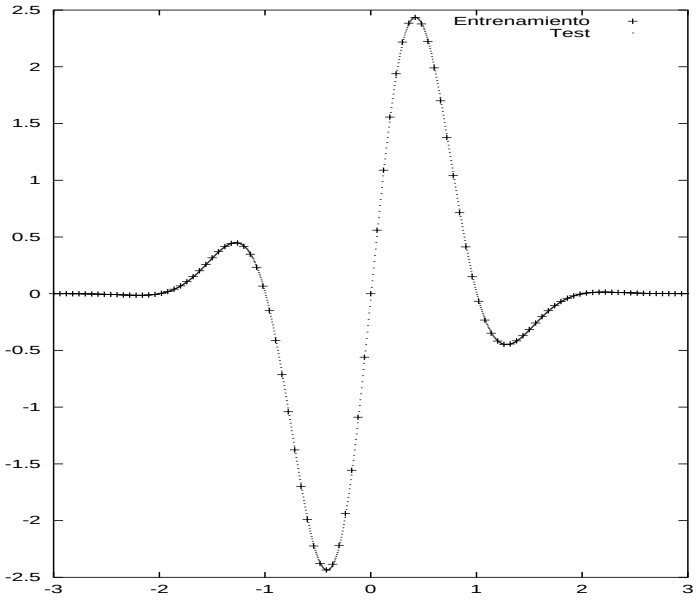


Figura 5.3: Representación de los ficheros de entrenamiento y test generados para la función f_3, definida en la ec. 5.6.

Generaciones	*ECM*	*Neuronas*	*Parámetros*	*Tiempo (segs.)*
10	0.07 ± 0.04	6 ± 1	1.20 %	1.8 ± 0.4
25	0.0011 ± 0.0005	13 ± 1	2.60 %	5.4 ± 0.5
50	5.7e-5±4.7e-5	20 ± 3	4.00 %	15.0 ± 1.7
75	2.5e-5±1.8e-5	27 ± 2	5.40 %	25.2 ± 3.1
100	1.1e-5±0.5e-5	31 ± 2	6.20 %	46.6 ± 3.2

Tabla 5.5: Resultados de EvRBF en la aproximación de la función f_3, definida en la ec. 5.6. Se muestran el ECM promedio, tamaño de red promedio, porcentaje de parámetros libres que supone cada tamaño de red con respecto al número de datos presentes en los conjuntos de entrenamiento y test, y tiempos de ejecución promedios expresados en segundos, todo ello para distinto número de generaciones en el AE.

Algoritmo		*ECM*	*Tamaño Red / Número de Reglas*
Pomares		0.0020	10
González		1.7e-5	6
Rivera		1.3e-4	6
EvRBF	*10 gen.*	0.07	6
EvRBF	*25 gen.*	0.001	13
EvRBF	*50 gen.*	5.7e-5	20
EvRBF	*75 gen.*	2.5e-5	27
EvRBF	*100 gen.*	**1.1 e-5**	31

Tabla 5.6: Comparación del ECM conseguido por los métodos descritos en [Poma00], [Gonz01a], [Rive02] y por EvRBF en la aproximación de la función f_3, definida en la ec. 5.6. Se incluye ECM y el tamaño para distintos números de generaciones en EvRBF-. Se comprueba que el método propuesto es capaz de conseguir errores más pequeños, aunque utilizando más neuronas que el resto de los métodos con los que se compara.

utiliza un mayor número de generaciones, se produce una reducción efectiva del ECM alcanzado por el algoritmo, existiendo diferencias cercanas a 3 órdenes de magnitud entre la aproximación alcanzada con 10 generaciones y la alcanzada con 50. Nuevamente, ello es debido a que se utiliza un mayor número de neuronas, que se van generando mediante el AE y que pasan a formar parte de la solución por conseguir una mejor aproximación de la función estudiada.

La tabla 5.6 permite comparar los resultados producidos por EvRBF- con los producidos por Rivera, González y Pomares en la aproximación de la función f_3. La conclusión más clara es que EvRBF es capaz de encontrar redes que ofrezcan menor ECM, aunque es a expensas de un mayor tamaño, que en este caso es hasta 5 veces mayor que los más pequeños encontrados. De hecho, para tamaños de red similares, los resultados de EvRBF son sólo comparables a los de Pomares. No obstante, cabe indicar que las redes halladas por EvRBF tras 100 iteraciones del AE tienen un tamaño promedio de 31 neuronas que, como muestra la tabla 5.5, suponen un total del 6.20 % de parámetros libres, calculados sobre el total de datos presentes en los conjuntos de entrenamiento y test. Es decir, representan un modelo válido para ser tenido en cuenta como aproximador de la función f_3.

5.2.1.4. Función $f_4 = e^{-3x} sen(10\pi x)$

La última de las funciones tomadas de [Rive02] con la que se ha evaluado EvRBF es la función f_4. La expresión que la define es la siguiente:

$$f_4 = e^{-3x} sen(10\pi x) \quad ; \quad x \in [0,1] \tag{5.7}$$

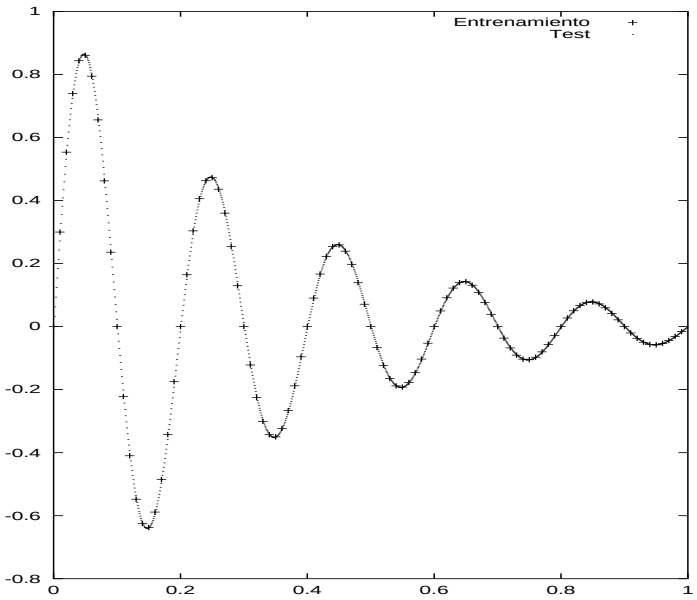


Figura 5.4: Representación de los ficheros de entrenamiento y test generados para la función f_4, definida en la ec. 5.7.

Como en los apartados anteriores, mostramos gráficamente los conjuntos de entrenamiento y test en la fig. 5.4.

Antes de comentar los resultados de EvRBF diremos que se ha utilizado la raíz cuadrada del ECM normalizado (RECMN) como medida de la bondad de la aproximación a la función para así poder comparar posteriormente con el resto de métodos.

La utilización de la función f_4 para evaluar al método EvRBF permite observar, mediante los datos presentes en la tabla 5.7, que no es una función que pueda ser bien aproximada mediante este método. Así, la utilización de más generaciones y el crecimiento de la redes que ello lleva asociado no consiguen rebajar el error cometido o, al menos, no de la forma en que se hacía para las funciones f_1, f_2 y f_3. Se observa que, tanto los tiempos de ejecución como los tamaños de red obtenidos son similares a los que se conseguían para las anteriores funciones. En este caso, por tanto, EvRBF no consigue colocar los centros en posiciones óptimas ni estimar los valores más correctos para los radios.

Los datos consignados en la tabla 5.8 muestran que EvRBF puede conseguir errores cercanos a los cometidos por los métodos de Pomares y Rivera, aunque utilizando entre 2 y 3 veces más neuronas (esto es, 23 neuronas usadas por EvRBF, frente a 12 y 8, respectivamente). Para esta función, es evidente que el mejor método es el propuesto por González, dado que

Generaciones	*RECMN*	*Neuronas*	*Parámetros*	*Tiempo (segs.)*
10	0.7± 0.2	7±4	1.40 %	2.4±0.0
25	0.7± 0.2	6±5	1.20 %	3.4±1.5
50	0.6± 0.3	9±8	1.80 %	7.2±6.1
75	0.2±0.3	23±7	4.60 %	23.8±6.7
100	0.4±0.3	22±11	4.40 %	26.6±16.4

Tabla 5.7: Resultados de EvRBF en la aproximación de la función f_4, definida en la ec. 5.7. Las distintas columnas muestran, por este orden, raíz cuadrada del ECM normalizado (RECMN) promedio, tamaños de red promedios, porcentajes de parámetros libres que dichos tamaños suponen con respecto al número de datos presentes en los conjuntos de entrenamiento y test, y tiempo de ejecución promedio expresado en segundos, todo ello para diferente número de generaciones en el AE.

Algoritmo		*RECMN*	*Tamaño Red / Número de Reglas*
Pomares		0.1	12
González		**0.06**	8
Rivera		0.1	8
EvRBF	*10 gen.*	0.7	7
EvRBF	*25 gen.*	0.7	6
EvRBF	*50 gen.*	0.6	9
EvRBF	*75 gen.*	0.2	23
EvRBF	*100 gen.*	0.4	22

Tabla 5.8: Comparación de la raíz cuadrada del ECM normalizado (RECMN) conseguido por los métodos descritos en [Poma00], [Gonz01a], [Rive02] y por EvRBF en la aproximación de la función f_4, definida en la ec. 5.7. Se incluye RECMN y el tamaño para distintos números de generaciones en EvRBF. Aunque, según refleja la tabla 5.7, EvRBF puede encontrar redes con capacidad de predicción muy similar a las mostradas por el resto de los métodos, en promedio, las redes generadas con EvRBF son peores que las generadas por el resto de autores y ello independientemente del número de neuronas usadas.

consigue rebajar de forma notable el error utilizando redes de 8 neuronas.

5.2.1.5. Conclusiones para las funciones f_1, f_2, f_3 y f_4

La utilización del conjunto de funciones f_1 a f_4 en la evaluación del método EvRBF vuelve a mostrar uno de los conceptos básicos de la programación de algoritmos, y es que nunca existe un método que sea el mejor para todo tipo de problemas. La utilización de operadores muy simples en EvRBF le permite trabajar con rapidez, aunque hace que precise de un mayor número de neuronas para poder alcanzar niveles de error bajos. No obstante, todas las funciones han podido ser aproximadas utilizando redes cuyo número de parámetros no excede el 10 % de los datos presentes en los conjuntos de entrenamiento y test, por lo que deben ser considerados modelos válidos para la resolución de estos problemas.

5.2.2. Aproximación de la función *Hermite*

El siguiente problema de aproximación funcional con el que se ha evaluado el método EvRBF es la denominada *función polinomial Hermite*. Esta función ha sido utilizada de forma profusa en [Orr95b], [Orr96], [Orr99] y [Cohe00b], precisamente para estudiar métodos en los que se intentan generar RNFBR de forma automática.

La función Hermite está definida de la siguiente forma:

$$H(x) = 1.1(1 - x + 2x^2)e^{\frac{-x^2}{2}} \;\; ; \;\; x \in [-5, 5] \tag{5.8}$$

En este problema se introduce un nuevo factor como es la adición de ruido a los valores de salida del conjunto de entrenamiento. Dicho ruido se ha modelado usando una función de probabilidad gaussiana, centrada en 0. Se ha experimentado con distintos niveles de ruido utilizando para ello distintos valores de varianza, a saber: 0.1, 0.01 y 0.001. Se pretende de esta forma realizar un estudio del impacto de la cantidad de ruido añadido en la efectividad del método empleado para aproximar la función.

La forma en que se han desarrollado los experimentos para esta función es similar a la utilizada por los distintos autores que la trataron previamente. En lo concerniente al conjunto de entrenamiento, se han generado, para cada uno de los niveles de ruido considerados, 5 conjuntos distintos (uno por cada una de las ejecuciones que vamos a realizar del algoritmo). Cada uno de estos conjuntos está compuesto por 40 puntos, cuyas entradas han sido elegidas aleatoriamente en el rango $[-5, 5]$, siguiendo una función de probabilidad uniforme. Las salidas proporcionadas por la

función $H(x)$ han sido posteriormente modificadas mediante la adición del ruido correspondiente. Por su parte, el fichero de test ha sido siempre el mismo, dado que está compuesto por 200 patrones, cuyas entradas están equidistribuidas a lo largo del intervalo $[-5, 5]$ y cuyas salidas no han sido modificadas mediante la adición de ruido.

La fig. 5.5.a muestra la función Hermite original; la fig. 5.5.b muestra el único conjunto de test que se ha generado; finalmente, las figs. 5.6, 5.8 y 5.10 muestran algunos de los conjuntos de entrenamiento utilizados, que corresponden con varianzas de 0.1, 0.01 y 0.001, respectivamente, para el error añadido a las respectivas salidas.

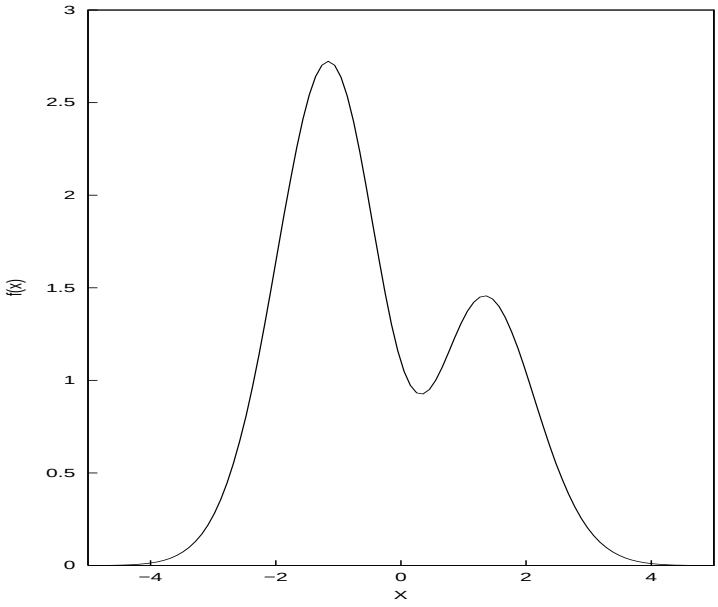
Los métodos hallados en la literatura utilizados para la aproximación de esta función son RFS (*Regularised Forward Selection*) [Orr95b] y PRBFN (*Perceptron-Radial Basis Net*) [Cohe00b].

El primero de ellos, RFS, es un método en el que se penalizan aquellos pesos de las conexiones entre la capa oculta y la de salida que son demasiado grandes; de esta forma, se elude el sobre-entrenamiento de la red evitando que aproxime exactamente todos los puntos del conjunto de entrenamiento. La aplicación de regularización conlleva inevitablemente el establecimiento del factor óptimo de penalización de los pesos, que en el caso concreto de RFS puede ser establecido de forma analítica. Además, el método se completa con un mecanismo de selección de centros para las RBF basado en validación cruzada, esto es, se evalúan los posibles centros candidatos con distintos conjuntos de entrenamiento y validación y se elige aquél que reduce de forma más efectiva el error.

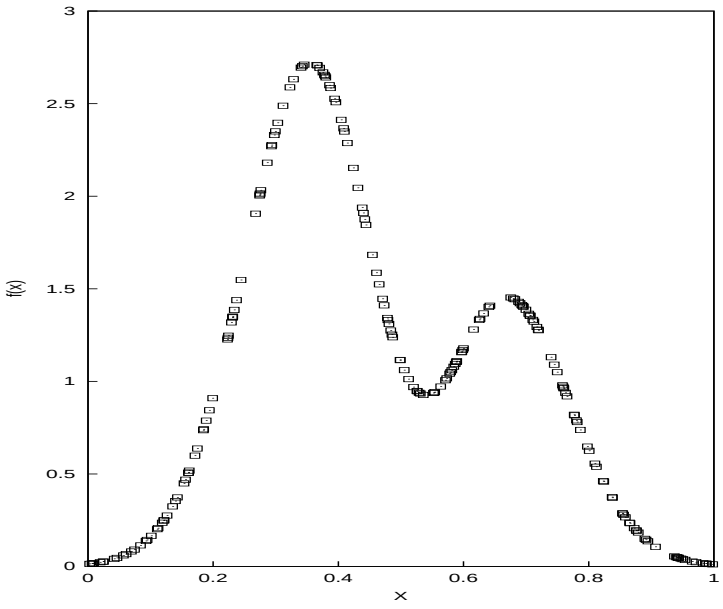
Por su parte, PRBFN hace referencia a un tipo de redes que combinan en su capa oculta neuronas con funciones de activación del tipo RBF y neuronas con funciones de activación no basadas en medidas o distancias (como ocurre con las neuronas de los perceptrones). El algoritmo de entrenamiento de las redes basa su actuación en dos mecanismos complementarios: la elección de un buen conjunto de centros iniciales (realizado mediante k-medias o un derivado del mismo propuesto por los autores) y la utilización de un algoritmo de descenso de gradientes para la estimación de forma iterativa de los mejores centros, radios y pesos de la red. El método controla el número de parámetros libres de la red que construye fijando previamente el número de neuronas que compondrán la red.

En ambos trabajos, [Orr95b] y [Cohe00b] se incluyen comparaciones con los resultados de métodos como RAN-EKF [Kadi92] (basado en redes de asignación de recursos o *resource-allocating networks*) y redes del tipo RBF diseñadas según los estudios de Bishop en [Bish91] y [Bish95a].

A continuación se describen los experimentos realizados con EvRBF sobre la función polinomial Hermite para distintos niveles de ruido.



(a) Función original



(b) Valores de salida del fichero de test

Figura 5.5: Representación de los valores de salida de la función *Hermite*, definida en la ec. 5.8.

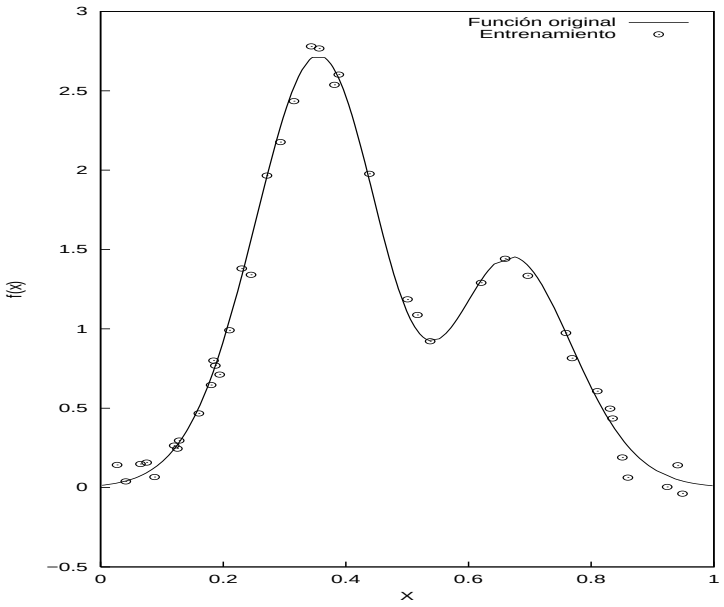


Figura 5.6: Representación de los valores de salida del fichero de entrenamiento de la función *Hermite*, definida en la ec. 5.8, cuando se añade ruido: $\sigma = 0.1$.

5.2.2.1. Adición de ruido con $\sigma = 0.1$

Comenzamos el estudio de la función polinómica Hermite añadiendo a las salidas de los distintos conjuntos de entrenamiento creados un error gaussiano centrado en cero y con varianza igual a 0.1. La representación gráfica de los valores de salida finales es la que muestra la fig. 5.6.

Los resultados que se obtienen con EvRBF cuando se aplica el nivel de error más alto son los recogidos en la tabla 5.9, que a su vez son comparados en la tabla 5.10 con el resto de métodos ya comentados.

Los resultados ofrecidos por EvRBF y mostrados en la tabla 5.9 indican, en primer lugar, que no es necesario realizar muchas ejecuciones del algoritmo sobre este conjunto de datos. De hecho, no existen apenas diferencias en los resultados obtenidos entre 10 y 100 ejecuciones, salvo en el tamaño de las redes, que aumenta conforme se incrementan el número de generaciones. La causa de este hecho puede estar en el reducido número de patrones de entrenamiento con que se cuenta y del cual hay que extraer un conjunto aún menor que se usa para la validación. Esto deja poco margen de maniobra al AE al no existir suficientes datos que le permitan discriminar acerca de la tendencia que debe seguirse en la generación de las redes.

Los resultados también muestran que también en este caso los modelos

Generaciones	*RECM*	*Neuronas*	*Parámetros*	*Tiempo (segs.)*
10	0.07±0.03	7±1	5.8 %	0.8±0.4
25	0.04±0.01	9±1	7.5 %	2.0±0.7
50	0.05±0.02	9±1	7.5 %	4.2±0.4
75	0.06±0.01	12±4	10.00 %	7.4±1.1
100	0.08±0.01	12±4	10.00 %	9.6±1.5

Tabla 5.9: Resultados de EvRBF en la aproximación de la función *Hermite*, definida en la ec. 5.8, con ruido añadido: $\sigma = 0.1$. Las distintas columnas muestran, respectivamente, la raíz cuadrada del ECM (RECM) promedio, tamaño de red promedio y porcentaje de parámetros libres que supone dicho tamaño con respecto al número de datos presentes en los conjuntos de entrenamiento y test, y finalmente, el tiempo de ejecución promedio en segundos empleado por EvRBF, todo ello para diferente número de generaciones del AE.

creados por EvRBF son válidos, dado que el número de parámetros libres se mantienen muy próximos al 10 % con respecto el conjunto de datos disponibles en los ficheros de entrenamiento y test. Adicionalmente, se ha representado para este problema (fig. 5.7) la evolución del tamaño de los individuos, en una ejecución típica de EvRBF utilizando 100 generaciones. La tendencia de dicha evolución es claramente ascendente, especialmente en las primeras generaciones, reduciéndose paulatinamente la tendencia a aumentar de tamaño. No obstante, se puede observar que el individuo con mejor *fitness* no se corresponde generalmente con el mayor de los individuos, sino que tiene un número de neuronas cercano a la media. Es decir, aunque el operador de adición de neuronas es fundamental para el éxito del algoritmo, el resto de ellos está permitiendo que redes con menor número de neuronas mejoren la aproximación realizada, lo cual a su vez permite regular de forma automática el tamaño de las redes. Puede observarse que, en muchos casos, el tamaño del mejor individuo decae con respecto a la generación anterior; esto es debido a que el algoritmo encuentra mejores combinaciones de centros y radios que permiten, con menos neuronas, aumentar la capacidad de generalización. Los resultados finales abundan sobre este hecho. Así, los mejores individuos de la última población tienen de promedio tan solo 12 neuronas (ver última fila de la tabla 5.9) frente a las 20 que tiene el mayor de los individuos encontrados.

En cuanto a la comparación de los resultados obtenidos por EvRBF- frente a los obtenidos por los métodos descritos en [Kadi92], [Orr95b], [Bish91] y [Cohe00b], que podemos encontrar en la tabla 5.10, destaca el hecho de que tanto EvRBF como PRBFN obtengan errores un orden de magnitud mejores que los obtenidos por el resto de métodos, y todo ello para un número de neuronas comparable al de los métodos que consiguen las redes más pequeñas. Esto indica una ventaja de EvRBF sobre el resto

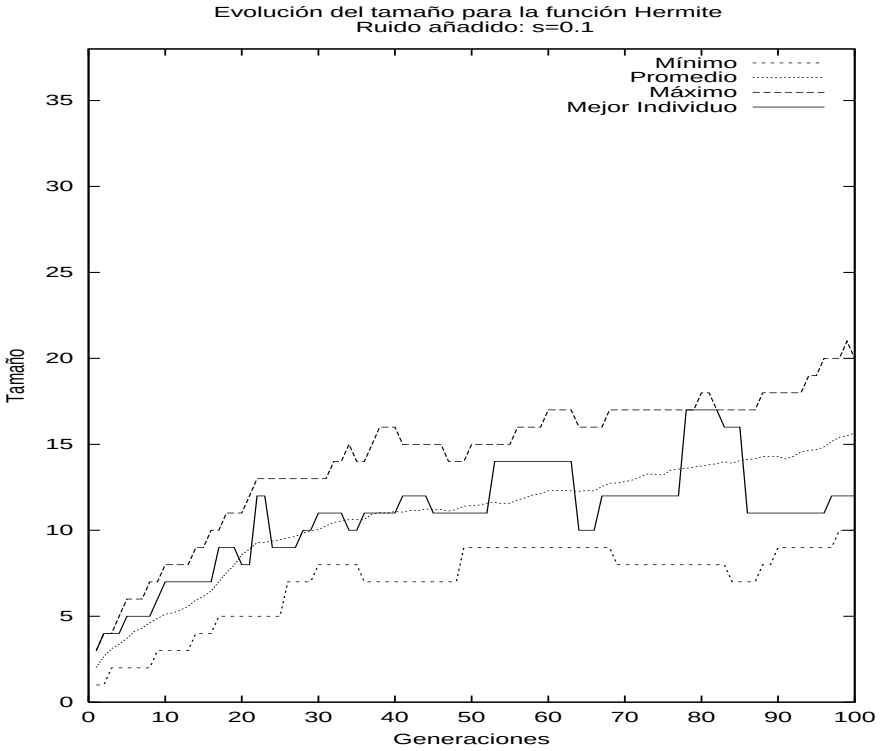


Figura 5.7: Evolución del tamaño de las redes en una ejecución típica de EvRBF en la aproximación de la función *Hermite* con ruido $\sigma = 0.1$. Se muestra el menor tamaño, el promedio, el mayor y el del individuo con mejor *fitness*. Se aprecia que el mejor individuo no se corresponde necesariamente con el de mayor tamaño. Así, el mejor error final lo produce una red con 12 neuronas (ver tabla 5.9), existiendo en la población redes mucho mayores, como puede observarse.

Algoritmo		*RECM*	*Tamaño Red*
RAN-EKF		0.30	30
RFS		0.15	7
RBF-Bishop		0.13	-
PRBFN		**0.04**	-
EvRBF	*10 gen.*	0.07	7
EvRBF	*25 gen.*	**0.04**	9
EvRBF	*50 gen.*	0.05	9
EvRBF	*75 gen.*	0.06	12
EvRBF	*100 gen.*	0.08	12

Tabla 5.10: Comparación de la raíz cuadrada del ECM (RECM) conseguido por los métodos descritos en [Kadi92], [Orr95b], [Bish91] y [Cohe00b], y el obtenido por EvRBF en la aproximación de la función *Hermite*, definida en la ec. 5.8, con ruido añadido: $\sigma = 0.1$. Se incluye la RECM y el tamaño para distintos números de generaciones en EvRBF. Como puede comprobarse, el error obtenido por nuestro método es un orden de magnitud menor que el obtenido por el resto y sólo igualado por PRBFN, siendo comparable el número de neuronas utilizado.

de métodos en cuanto a la forma en que le afecta el error. La posible causa de este hecho habremos de buscarla en la utilización de un conjunto de validación para obtener el *fitness* de las redes. Así, al evaluar las redes dentro del AE en función de su capacidad de generalización y no del error que consiguen cuando intentan aproximar los patrones usados para su entrenamiento, se les impide aproximar de manera exacta las salidas de los patrones de entrenamiento, produciendo como consecuencia que no se aprenda el ruido añadido a dichos patrones.

Por otro lado, el método que presentamos tiene claras ventajas sobre PRBFN en dos sentidos. El primero es que EvRBF estima no sólo el valor de centros y radios, sino también el número de neuronas que deben ser utilizadas. De hecho, un trabajo previo de dichos autores [Cohe00a] en el que intentaban optimizar también este parámetro devolvía errores mucho mayores para el mismo problema. La segunda ventaja está en la posibilidad de utilizar como RBF en EvRBF cualquier función que pueda imaginarse, independientemente de si es diferenciable o no. Esto no ocurre con PRBFN dado que es un método basado en descenso de gradientes que precisa de las derivadas primera y segunda de las RBF utilizadas para poder modificar los valores de centros, radios y pesos.

5.2.2.2. Adición de ruido con $\sigma = 0.01$

A continuación, y siguiendo con los estudios realizados por Orr, se va a reducir el nivel de ruido añadido a las salidas del conjunto de entrena-

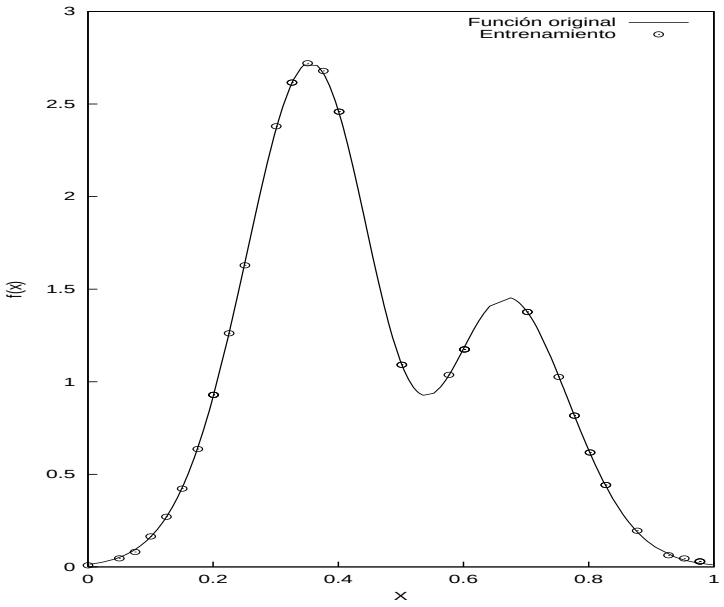


Figura 5.8: Representación de los valores de salida del fichero de entrenamiento de la función *Hermite*, definida en la ec. 5.8, cuando se añade ruido: $\sigma = 0.01$.

miento de modo que su varianza sea 10 veces menor a la del apartado anterior. La representación gráfica de uno de los 5 conjuntos de entrenamiento generados usando este error es la mostrada en la fig. 5.8

Los resultados que genera EvRBF, recogidos en la tabla 5.11, muestran que en este caso el incremento del número de generaciones utilizadas tienen un efecto positivo en la disminución del error final conseguido. Así, si con 10 y 25 generaciones el error era muy similar al alcanzado cuando añadíamos ruido $\sigma = 0.1$, la utilización de 50 generaciones permite redu-

Generaciones	*RECM*	*Neuronas*	*Parámetros*	*Tiempo (segs.)*
10	0.092±0.030	6±1	5.0 %	0.8±0.4
25	0.077±0.147	11±1	9.0 %	1.8±0.4
50	0.011±0.003	16±3	13.0 %	6.0±0.0
75	0.011±0.004	19±4	16.0 %	10.6±2.1
100	0.010±0.002	25±4	21.0 %	18.0±4.2

Tabla 5.11: Resultados de EvRBF en la aproximación de la función *Hermite*, definida en la ec. 5.8, con ruido añadido: $\sigma = 0.01$. Las distintas columnas muestran la raíz cuadrada del ECM (RECM) promedio, tamaños de red promedio, porcentaje de parámetros libres que supone dicho tamaño con respecto al número de datos presentes en los conjuntos de entrenamiento y test, y tiempo de ejecución promedio expresado en segundos, para diferente número de generaciones del AE.

Algoritmo		*RECM*	*Tamaño Red*
RAN-EKF		0.15	16
RFS		0.07	12
EvRBF	*10 gen.*	0.092	6
EvRBF	*25 gen.*	0.077	11
EvRBF	*50 gen.*	**0.011**	16

Tabla 5.12: Comparación de la raíz cuadrada del ECM (RECM) conseguido por los métodos descritos en [Kadi92] y [Orr95b], así como por por EvRBF en la aproximación de la función *Hermite*, definida en la ec. 5.8, con ruido añadido: $\sigma = 0.01$. Se incluye la RECM y el tamaño para distintos números de generaciones en EvRBF. Al igual que en el caso anterior, en el que $\sigma = 0.1$, el error obtenido por EvRBF es menor que el obtenido por RAN-EKF, similar al obtenido por RFS, para el mismo tamaño de red, y mejor que cualquiera de ellos cuando se añaden nuevas neuronas.

cirlo de forma considerable, aunque a costa de la adición de nuevas neuronas. No obstante, la utilización de mayor número de generaciones resulta de nuevo infructuosa pues, como puede verse, aunque con 100 generaciones se alcanza el menor de los errores (0.010) no es significativamente distinto del alcanzado con la mitad de generaciones (0.011). Además, la cantidad de neuronas generadas hace muy elevado el número de parámetros libres, excediendo excesivamente del 10 % cuando se utilizan 75 ó 100 generaciones.

La evolución del tamaño de las redes a lo largo de una ejecución típica de EvRBF para este problema es la representada en la fig. 5.9. Se observa que, nuevamente, un mayor número de neuronas no implica necesariamente una mejor capacidad de generalización. Así, el uso del conjunto de operadores genéticos que se han creado permite que no sean las redes de mayor tamaño las que obtengan los mejores resultados, como en principio se podría esperar. No obstante, dado que el método se basa principalmente en el error obtenido por las redes para asignarles un *fitness*, es normal que la tendencia de evolución del tamaño sea ascendente. Esto también puede explicar que sea mayor el tamaño de las redes generadas cuando el error es $\sigma = 0.01$ con respecto a las que se obtenían para $\sigma = 0.1$. En este último caso (ver fig. 5.7), las redes mayores estarían aproximando excesivamente el ruido contenido en los patrones de entrenamiento, por lo que su *fitness* (esto es, capacidad de generalización) sería peor. Por el contrario, a medida que el ruido decrece, un mayor número de neuronas favorece una mejora en el *fitness*, y es por ello que encontramos redes hasta un 50 % más grandes.

En comparación con los métodos RAN-EKF [Kadi92] y RFS [Orr95b], la tabla 5.12 muestra que, al igual que para $\sigma = 0.1$, el error obtenido

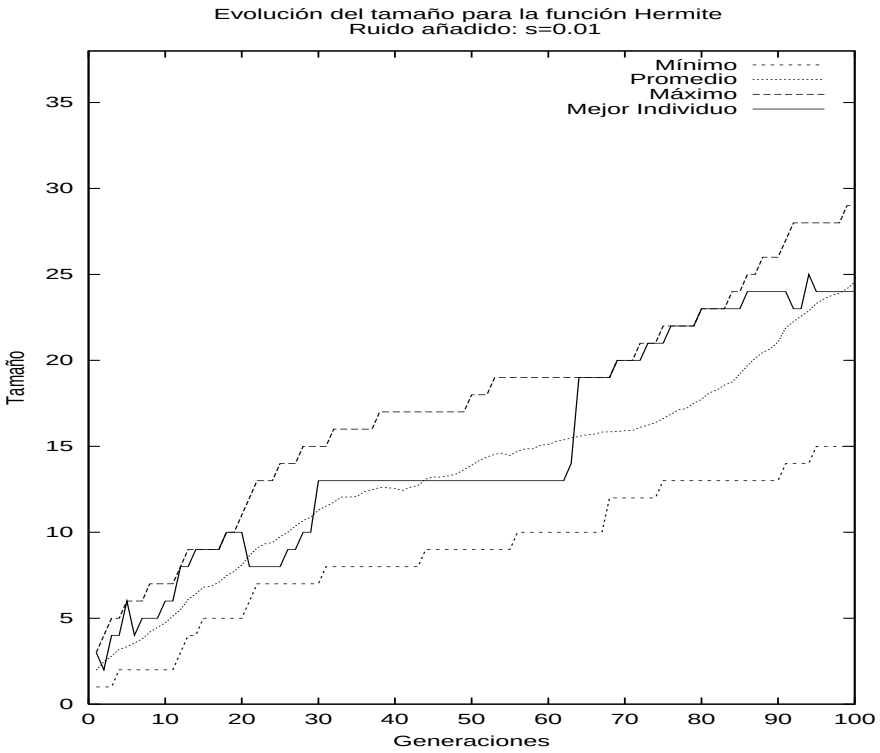


Figura 5.9: Evolución del tamaño de las redes en una ejecución típica de EvRBF en la aproximación de la función *Hermite* con ruido $\sigma = 0.01$. Se muestra el menor tamaño, el promedio, el mayor y el del individuo con mejor *fitness*. Al igual que para el caso de $\sigma = 0.1$, se observa que existen redes en la última generación cuyo tamaño excede el de las mejores redes halladas (ver tabla 5.11). Es decir, EvRBF está regulando el número de neuronas óptimo a pesar de que basa el establecimiento del *fitness* en el error cometido por las redes.

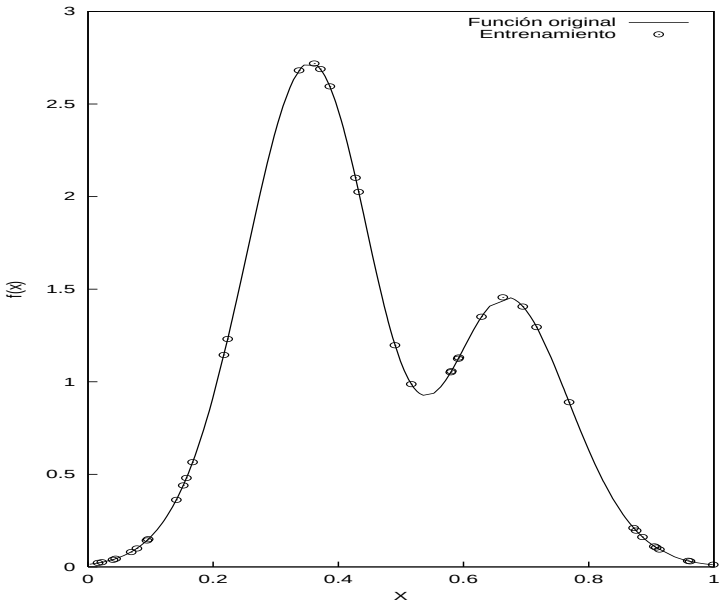


Figura 5.10: Representación de los valores de salida del fichero de entrenamiento de la función *Hermite*, definida en la ec. 5.8, cuando se añade ruido: $\sigma = 0.001$.

por EvRBF es menor que el obtenido por RAN-EKF, aunque es similar al obtenido por RFS, para el mismo tamaño de red. No obstante, cuando se utilizan 50 generaciones en vez de 25, EvRBF mejora el error cometido por cualquiera de los demás métodos, generando redes de tamaño similar a las que se obtienen con RAN-EKF y sólo 1.3 veces mayores de las generadas mediante RFS.

5.2.2.3. Adición de ruido con $\sigma = 0.001$

Para terminar con la aplicación de EvRBF a la función polinomial Hermite, se ha reducido el nivel de error hasta un límite prácticamente imperceptible, en el que $\sigma = 0.001$. De hecho, en la gráfica de la fig. 5.10 puede observarse que las salidas de los patrones del conjunto de entrenamiento coinciden prácticamente con los valores de la función original.

El hecho de que se introduzca menos error en los datos de entrenamiento tiene como primer resultado una disminución del error de generalización que obtienen las redes generadas mediante EvRBF como puede observarse en la tabla 5.13. Así, podemos comprobar que cuando se utilizan 50 generaciones, se obtienen redes de tamaño idéntico a las que se creaban con $\sigma = 0.01$, pero con un error de generalización que es la mitad del que se conseguía en tales condiciones (0.005 frente a 0.011). Nueva-

Generaciones	*ECM*	*Neuronas*	*Parámetros*	*Tiempo (segs.)*
10	0.073±0.032	7±1	6.0 %	0.8±0.4
25	0.024±0.023	11±3	9.0 %	1.8±0.4
50	0.005±0.003	16±3	13.0 %	5.4±0.5
75	0.006±0.001	18±2	15.0 %	11.2±0.4
100	0.006±0.004	23±4	19.0 %	17.4±0.5

Tabla 5.13: Resultados de EvRBF en la aproximación de la función *Hermite*, definida en la ec. 5.8, con ruido añadido: $\sigma = 0.001$. Las distintas columnas muestran la raíz cuadrada del ECM (RECM) promedio, tamaños de red promedio, porcentaje de parámetros libres que supone dicho tamaño con respecto al número de datos presentes en los conjuntos de entrenamiento y test, y tiempo de ejecución promedio expresado en segundos, para diferente número de generaciones del AE.

mente, se detecta una disminución significativa del error cuando se emplean 50 generaciones (el error es de 0.005), en comparación con el error alcanzado cuando se detiene la ejecución del AE tras 25 generaciones (el error es de 0.024). E igualmente, se comprueba que el método no es capaz de reducir este error aún cuando un número mayor que 50 generaciones sea empleado. De hecho, el error comienza a aumentar y se generan redes cuyo número de parámetros libres excede en mucho el entorno de los 10 % que consideramos válidos.

Como en los casos anteriores para tasas mayores de ruido, se ha incluido una gráfica de la evolución del número de neuronas durante la ejecución típica de EvRBF sobre este problema (fig. 5.11). Las conclusiones, nuevamente, reflejan lo ya comentado en los apartados anteriores: el mejor individuo de cada generación no es aquél que tiene el mayor número de neuronas, como en principio cabría esperar de un método que para comparar los individuos se basa exclusivamente en el error que cometen; antes bien, el mejor individuo de cada generación es aquél que mejor combina su tamaño con unos valores de centros y radios adecuados. En cuanto a la evolución del tamaño con respecto a los anteriores niveles de ruido, se observa que las redes obtenidas coinciden en tamaño con las generadas para $\sigma = 0.01$ (ver fig. 5.9) y, por tanto, mantienen la misma relación con respecto al tamaño para $\sigma = 0.1$. Es decir, en este caso, la disminución del ruido no ha sido lo suficientemente significativa como para que se produzcan diferencias notables en el tamaño de las redes.

Por otro lado, la principal conclusión que podemos sacar de los resultados mostrados en la tabla comparativa 5.14 es que, nuevamente, el algoritmo presentado se comporta mejor que el resto cuando el proceso evolutivo hace que se añadan nuevas neuronas. Así, si se permite iterar hasta 50 generaciones, el error es aproximadamente 10 veces menor que el alcanzado

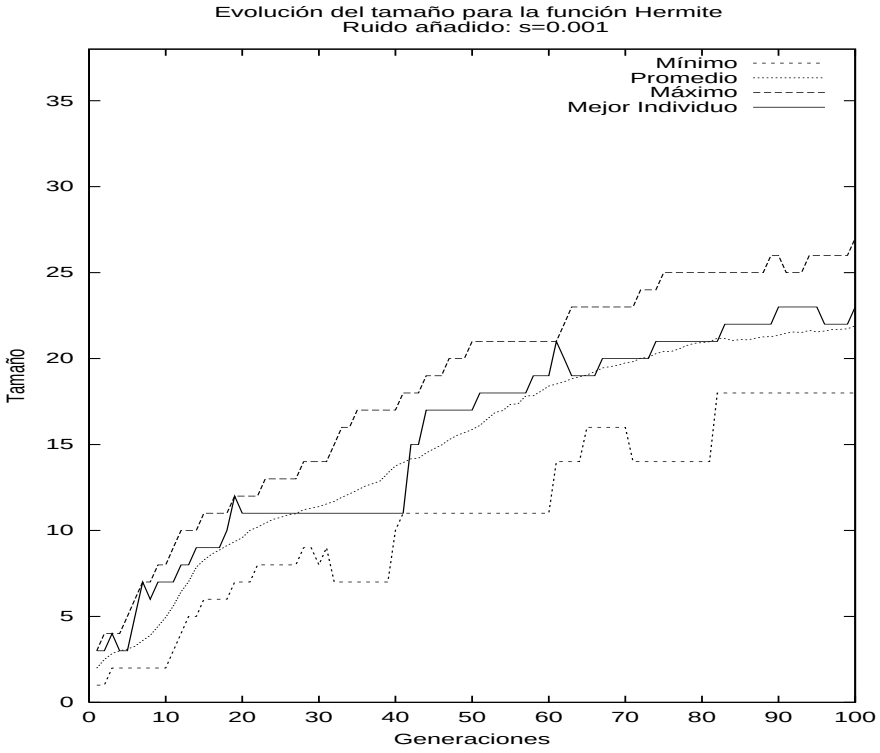


Figura 5.11: Evolución del tamaño de las redes en una ejecución típica de EvRBF en la aproximación de la función *Hermite* con ruido $\sigma = 0.001$. Se muestra el menor tamaño, el promedio, el mayor y el del individuo con mejor *fitness*. Al igual que para las tasas de ruido anteriores, se observa que no siempre las redes más grandes son las que obtienen un mejor *fitness*, sino que la modificación de los centros y radios realizada por los operadores genéticos permite adaptar redes más pequeñas para que produzcan errores más bajos.

Algoritmo		*RECM*	*Tamaño Red*
RAN-EKF		0.03	13
RFS		0.04	15
EvRBF	*10 gen.*	0.073	7
EvRBF	*25 gen.*	0.024	11
EvRBF	*50 gen.*	**0.005**	16

Tabla 5.14: Comparación de la raíz cuadrada del ECM (RECM) conseguido por los métodos descritos en [Kadi92] y [Orr95b], así como por por EvRBF en la aproximación de la función *Hermite*, definida en la ec. 5.8, con ruido añadido: $\sigma = 0.001$. Se incluye la RECM y el tamaño para distintos números de generaciones en EvRBF. Nuevamente, el algoritmo presentado se comporta mejor que el resto cuando el proceso evolutivo hace que se añadan nuevas neuronas. Además, para las redes generadas con EvRBF cuyo tamaño es muy similar aunque inferior al de las generadas con RAN-EKF y RFS, el error cometido es ligeramente mejor que el obtenido con dichos métodos.

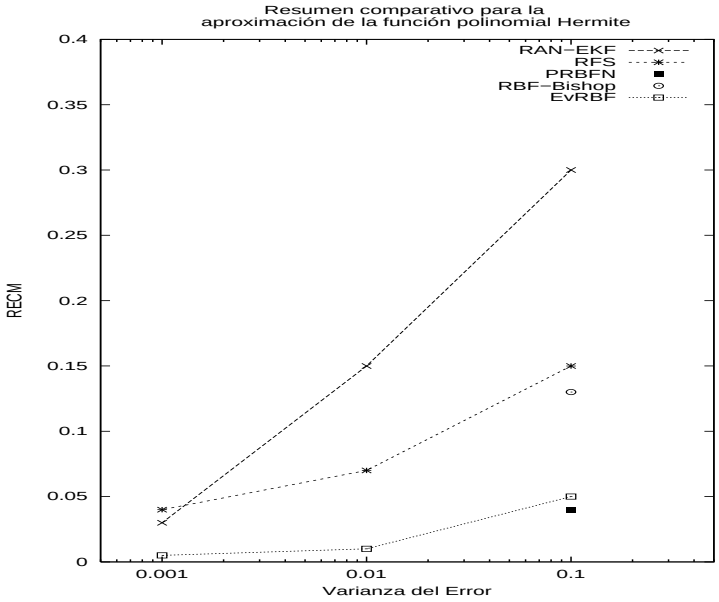


Figura 5.12: Comparativa del error, expresado como raíz cuadrada del ECM (RECM), obtenido mediante los métodos RAN-EKF, RFS, PRBFN y EvRBF para distintos niveles de ruido en la aproximación de la función polinomial *Hermite*. La aproximación ofrecida por EvRBF resulta ser mejor que la del resto de métodos, salvo en el caso de $\sigma = 0.1$, en el que es prácticamente similar a PRBFN.

tanto por RFS (0.04) como por RAN-EKF (0.03), con un tamaño de red muy similar (16 neuronas, frente a 15 y 13, respectivamente).

5.2.2.4. Conclusiones para la función polinomial *Hermite*

Tras estudiar el comportamiento de EvRBF en la aproximación de la función polinomial Hermite para distintos niveles de ruido añadido, mostramos a modo de resumen un gráfico comparativo (ver fig. 5.12) de los resultados obtenidos por nuestro método con los obtenidos por otros hallados en la literatura. En dicho gráfico, de los diversos errores obtenidos por EvRBF, sólo se muestran los relativos al uso de 50 generaciones en el AE. Como puede observarse, el error cometido por EvRBF resulta ser menor en todos los casos que el que cometen el resto de los métodos considerados, manteniendo tamaños de red muy similares, como se ha mostrado en los apartados 5.2.2.1 a 5.2.2.3. Tan sólo cuando $\sigma = 0.1$ encontramos que el método PRBFN obtiene un error ligeramente inferior al obtenido por EvRBF, aunque ya se comentaba en el apartado 5.2.2.1 que nuestro método ofrece otras ventajas, como la no restricción de las FBR a funciones diferenciables y la determinación dinámica del número de neuronas

que deben formar las redes. Los autores de PRBFN no describen experimentos ni resultados para valores de la varianza del ruido distintos de 0.1, a pesar de utilizar las mismas fuentes bibliográficas que se han usado en esta tesis.

5.2.3. Aproximación de la función de *Friedman*

La última de las funciones que vamos a aproximar mediante EvRBF simula un circuito de corriente alterna compuesto por una resistencia (R ohmios), un condensador (C faradios) y una bobina (L henrios), y al cual también afecta un valor de frecuencia angular de la corriente (ω radianes por segundo). Los anteriores cuatro parámetros corresponden a las variables de entrada y el problema consiste en determinar el valor de impedancia (Z), definido según la siguiente ecuación:

$$Z(x) = \sqrt{R^2 + \left(\frac{\omega L - 1}{\omega C}\right)^2} \tag{5.9}$$

donde:

$$x = \{R, \omega, L, C\}, \tag{5.10}$$
$$0 \leq R \leq 100, \tag{5.11}$$
$$40\pi \leq \omega \leq 560\pi, \tag{5.12}$$
$$0 \leq L \leq 1, \tag{5.13}$$
$$1e-6 \leq C \leq 11e-6 \tag{5.14}$$

El problema aborda los posibles problemas ocasionados por el ruido, es por ello que a los valores de salida de los distintos conjuntos de entrenamiento que se van a crear se les añade ruido gaussiano, con media igual a 0 y varianza $\sigma_Z = 175$.

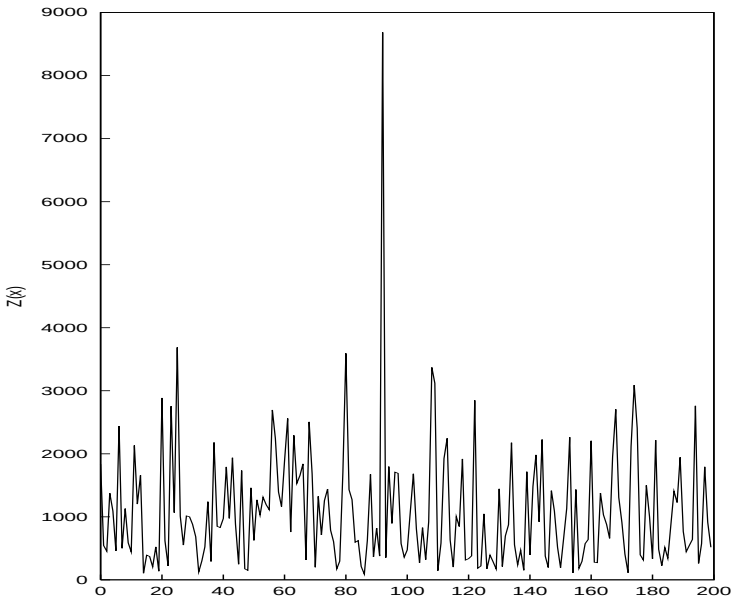
Al igual que la función *Hermite*, este problema ha sido usado por autores como Orr [Orr95b] [Orr96] [Orr99] y Cohen e Intrator [Cohe00b], aunque originalmente fue utilizado por Friedman [Frie91], de ahí que suela ser conocido por este nombre. Los métodos por tanto que se han utilizado para intentar aproximar esta función son los descritos para la función *Hermite*, esto es, RFS de Orr y PRBFN de Cohen e Intrator, que a su vez comparan con los resultados ofrecidos por las RNFBR de Bishop [Bish91] [Bish95a].

Generaciones	*ECMN*	*Neuronas*	*Parámetros*	*Tiempo (segs.)*
10	0.10 ± 0.02	10 ± 3	6.50 %	5 ± 1
25	0.12 ± 0.05	16 ± 4	10.40 %	14 ± 4
50	0.09 ± 0.02	17 ± 5	11.05 %	34 ± 7
75	0.11 ± 0.02	26 ± 3	16.90 %	68 ± 8
100	0.11 ± 0.02	28 ± 7	18.20 %	96 ± 19

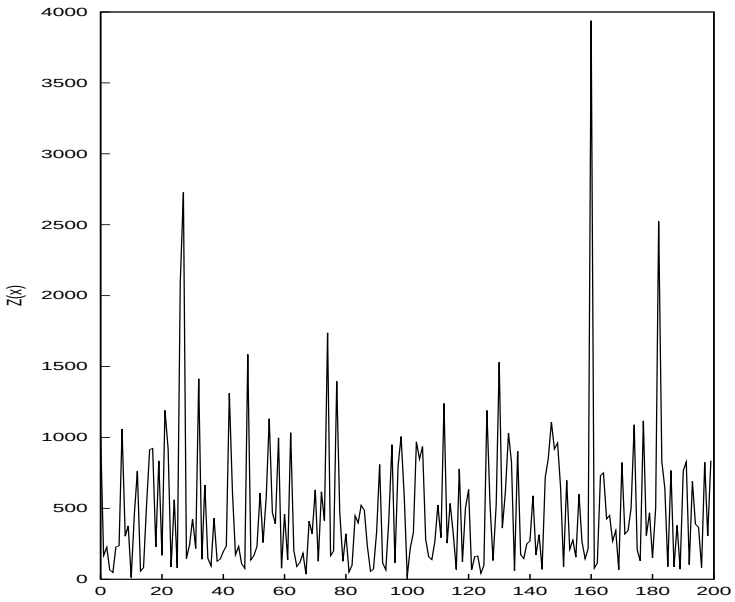
Tabla 5.15: Resultados de EvRBF en la aproximación de la función *Friedman*, definida en la ec. 5.9. Las sucesivas columnas muestran el número de generaciones empleado en el AE y los valores obtenidos de ECM normalizado, tamaño de red, porcentaje de parámetros libres que dicho tamaño con respecto al número de datos presentes en los conjuntos de entrenamiento y test, y tiempo de ejecución expresado en segundos.

Siguiendo la misma metodología empleada por el resto de autores, se han generado conjuntos de entrenamiento compuestos por 200 patrones de entrada con sus respectivas salidas, calculadas usando la expresión 5.14. Adicionalmente, el problema aborda los posibles problemas ocasionados por el ruido, es por ello que a tales valores de salida se les ha añadido ruido gaussiano, con media igual a 0 y varianza $\sigma_Z = 175$. Los valores de entrada de los distintos patrones de los conjuntos de entrenamiento y test se eligen de forma aleatoria, siguiendo una función de distribución uniforme en el rango en el que cada variable R, C, L y ω está definida. Esto posibilita la ejecución de EvRBF utilizando conjuntos distintos cada vez, para así minimizar el posible sesgo que pudieran introducir un par de conjuntos específicos. Más concretamente, se han generado 5 conjuntos de entrenamiento y test distintos, que han sido usados en la evaluación de nuestro método para esta función. En la fig. 5.13.a se muestran las salidas presentes en uno de los conjuntos de entrenamiento y en la fig. 5.13.b se muestra la misma información pero relativa al conjunto de test correspondiente. Dado que existen cuatro variables de entrada, el eje de abcisas muestra simplemente el orden en que se encuentran dichas salidas dentro de los conjuntos de entrenamiento y test, respectivamente.

Los resultados que obtiene EvRBF en la aproximación de la función *Friedman* están detallados en la tabla 5.15. Al igual que el resto de autores, se ha utilizado el ECM normalizado para dar una medida de la bondad de la estimación de las redes generadas con EvRBF. El ECM normalizado se consigue dividiendo el ECM por la desviación estándar de las salidas del conjunto de test. De esta forma, la columna de la tabla 5.15 etiquetada como *ECMN* muestra que basta con utilizar 10 generaciones para encontrar una red que aproxime de forma satisfactoria la función. El hecho de que con un mayor número de generaciones aumente el tamaño de las redes indica que el método es capaz de encontrar redes que aproximan mejor



(a) Valores de salida del fichero de entrenamiento



(b) Valores de salida del fichero de test

Figura 5.13: Representación de los valores de salida de la función *Friedman*, definida en la ec. 5.9.

Algoritmo		*ECMN*	*Tamaño Red*
RFS		0.14	
RBF-Bishop		0.17	
PRBFN		0.20	
EvRBF	*10 gen.*	0.10	10
EvRBF	*25 gen.*	0.12	16
EvRBF	*50 gen.*	**0.09**	17

Tabla 5.16: Comparación del ECM normalizado conseguido por los métodos descritos en [Orr95b], [Bish91] y [Cohe00b] y por EvRBF en la aproximación de la función *Friedman*, definida en la ec. 5.9. Se incluye el ECM normalizado y el tamaño de red para distintos números de generaciones en EvRBF. Los resultados muestran que el peor de los errores hallados con EvRBF es tan bueno como el mejor de los hallados con el resto de métodos. En promedio, el método propuesto genera redes con mayor capacidad de generalización que los restantes.

los datos de los conjuntos de entrenamiento y validación, aunque ello no deriva en un aumento de la capacidad predictiva de la red, es decir, en una mejor aproximación del conjunto de test. De hecho, se puede observar que a partir de 75 generaciones el error de generalización es peor, lo cual indica que se está produciendo un sobre-entrenamiento de las redes. Además, para 75 y 100 generaciones, el tamaño de las redes hacen que el número de parámetros libres sobrepase excesivamente el 10 % sobre los datos presentes en los conjuntos de entrenamiento y test que hemos marcado como límite aceptable para considerar el modelo válido.

En relación a los resultados obtenidos por el resto de los métodos, los datos consignados en la tabla 5.16 muestran que incluso el peor de los errores obtenidos con EvRBF (0.12) es ligeramente mejor que el mejor de los hallados con el resto de métodos (que en este caso corresponde al método RFS de Orr, cuyo ECM normalizado es de 0.14). En promedio, el método propuesto genera redes con mayor capacidad de generalización que los restantes, superando también a PRBFN. De hecho, en este caso, este último método consigue el peor rendimiento de todos los considerados, cuando en la resolución de la función polinomial *Hermite* para $\sigma = 0.1$ se comportaba tan bien como EvRBF. Ninguno de los autores hace mención al tamaño de las redes generadas, por lo que no es posible realizar una comparación atendiendo a este parámetro. En cualquier caso, los resultados relativos a EvRBF pertenecen a redes que podemos considerar modelos perfectamente válidos en cuanto al número de parámetros libres que utilizan.

5.3. Evaluación de EvRBF en problemas de clasificación

La segunda tarea en la que ha sido evaluado EvRBF ha consistido en la clasificación de patrones. Se pueden establecer analogías entre este tipo de problemas y el de aproximación funcional si consideramos que la RNFBR intenta aproximar la función de probabilidad que sigue cada una de las posibles clases en que pueden ser clasificados los distintos patrones de entrada. Para ello, aún cuando los patrones de los ficheros de entrenamiento y test estén compuestos por $d_e + 1$ datos (d_e pertenecientes a las variables de entrada y 1 perteneciente a la clase), las redes deben generarse de forma que acepten d_e valores de entrada y produzcan d_s salidas, donde d_s es el número de clases distintas en que pueden ser clasificados los patrones. De esta forma, cuando se le presenta un patrón de entrada a la red, la salida que proporciona es el número de la neurona de la capa de salida que devuelve el mayor valor. Dicha salida debe coincidir con la clase a la que pertenece dicho patrón.

Los problemas considerados son ampliamente conocidos y están tomados de distintos repositorios accesibles a través de Internet. En concreto se han utilizado las bases de datos *Iris* (apartado 5.3.1), *Enfermedad de corazón* (*Heart Disease*, apartado 5.3.2) y *Cáncer de pulmón* (apartado 5.3.3)

5.3.1. Clasificación de la base de datos *Iris*

El primero de los problemas de clasificación que ha sido abordado es el que correspondiente a la conocida base de datos *Iris* [Blak98]. Los patrones en ella registrados contienen datos relativos a tres tipos distintos de plantas Iris. Cada uno de dichos tipos está representado por aproximadamente 50 patrones, que a su vez se componen de cuatro entradas numéricas (longitud y anchura de sépalo y longitud y anchura de pétalo) y una salida (0, 1 ó 2) indicando el tipo de planta al que pertenece. No existen valores nulos ni error en la clasificación de los patrones. Se disponen de un total de 156 patrones, los cuales están divididos al cincuenta por ciento en un conjunto de entrenamiento y un conjunto de test.

Los métodos utilizados para resolver este problema con los que va a ser comparado EvRBF son los utilizados por Whitehead y Choate [Whit96] y Burdsall y Giraud [Burd97] dado que ambos utilizan AE para generar - RNFBR que son aplicadas al problema en cuestión. De hecho, ambos han sido ya reseñados en el apartado 2.3.3 de la presente tesis. Adicionalmente, en [Burd97] se hace referencia a resultados proporcionados por una

Generaciones	*Porcentaje de Aciertos*	*Neuronas*	*Parámetros*	*Tiempo (segs.)*
10	98.2 ± 0.7	6 ± 1	8.14 %	3 ± 0
25	99.0 ± 0.6	6 ± 2	8.14 %	7 ± 1
50	98.7 ± 0.0	6 ± 1	8.14 %	13 ± 1
75	99.0 ± 0.6	7 ± 2	9.49 %	20 ± 3
100	98.5 ± 1.1	5 ± 2	6.78 %	25 ± 5

(b)

Tabla 5.17: Resultados de EvRBF en la clasificación de la base de datos *Iris*. Las distintas columnas muestran el porcentaje de aciertos, tamaño de red, porcentaje de parámetros libres que supone dicho tamaño con respecto al número de datos presentes en los conjuntos de entrenamiento y test, y tiempo de ejecución expresado en segundos. Estos datos están promediados para 5 ejecuciones de EvRBF y para distinto número de generaciones empleadas.

RNFBR generada de forma manual optimizada para el problema (se indica como *RBF* en la tabla 5.18 de comparación de resultados), por un clasificador NAP (*nearest-attracting prototype*), basado en k-medias y desarrollado por los propios Burdsall y Giraud y, finalmente, por un perceptrón multicapa (se indica como *MLP* en la tabla de comparación de resultados).

La tabla 5.17 muestra los resultados que obtuvo EvRBF en la clasificación de la base de datos *Iris*. Como en el resto de experimentos, están promediados para 5 ejecuciones del algoritmo en las que se parten de poblaciones iniciales distintas. En el caso concreto de este problema, sólo se ha hecho uso de un conjunto de entrenamiento y un conjunto de test, que son los indicados por los suministradores de estos datos. Como puede observarse en dicha tabla, tanto el porcentaje de aciertos (entre 98.5 % y 99.0 %) como el tamaño de las redes generadas (entre 5 y 7) son muy similares independientemente del número de generaciones empleadas. Esto se debe a que, en general, en la inmensa mayoría de las ejecuciones se consigue clasificar correctamente el 100 % de los datos presentes en los conjuntos de entrenamiento y validación, siendo éste el último el utilizado por el AE para asignar el *fitness* lo cual hace que, por mucho que continúe el AE, sea inviable mejorar los individuos hallados. No obstante, y a pesar de la excelente clasificación que se realiza durante la ejecución del AE, esto no lleva al sobre-entrenamiento de las redes para este problema concreto, dado que la capacidad de generalización de las redes es muy alta.

Comparando con el resto de métodos, la tabla 5.18 muestra que EvRBF consigue las mejores tasas de acierto (99 % de nuestro método, frente a 97.17 %, de Whitehead y Choate) manteniendo muy bajo el número de neuronas a utilizar. Adicionalmente, se observa que el método evolutivo propuesto por Burdsall y Giraud no consigue mejorar los resultados ofre-

Algoritmo		*Porcentaje de Aciertos*	*Tamaño Red*
Whitehead-Choate		97.19	15
Burdsall-Giraud		96.3	6-9
RBF		96.3	-
MLP		96.0	-
NAP		95.6	-
EvRBF	*10 gen.*	98.2	6
EvRBF	*25 gen.*	**99.0**	6
EvRBF	*50 gen.*	98.7	6
EvRBF	*75 gen.*	**99.0**	7
EvRBF	*100 gen.*	98.5	5

Tabla 5.18: Comparación del porcentaje de aciertos conseguido por los métodos de Whitehead y Choate [Whit96], Burdsall y Giraud [Burd97], una RNFBR optimizada manualmente para este problema, un perceptrón multicapa (MLP), un clasificador NAP (*nearest-attracting prototype*) y por EvRBF en la clasificación de la base de datos *Iris*. Los resultados relativos a las filas *RBF*, *MLP* y *NAP* se han tomado de [Burd97]. En cuanto a EvRBF, se han consignado el porcentaje de aciertos y el tamaño para distintos números de generaciones. Puede comprobarse que EvRBF consigue las mejores tasas de acierto manteniendo muy bajo el número de neuronas a utilizar.

cidos por una RNFBR optimizada de forma manual y que sólo el método de Whitehead y Choate, también un método evolutivo, y por supuesto EvRBF, consiguen superar a los demás en la búsqueda del mejor clasificador. Finalmente, aunque se ha preferido no incluir datos sobre máximos y mínimos en esta tesis, cabe decir que algunas de las redes encontradas por EvRBF han conseguido clasificar el 100 % de los patrones del conjunto de test.

5.3.2. Clasificación de la base de datos *Heart Disease*

El siguiente problema trata sobre el diagnóstico de enfermedades de corazón (*heart disease*) y está tomado del mismo repositorio que el anterior [Blak98]. Consta de 270 observaciones realizadas sobre pacientes, compuesta cada una por 13 valores correspondientes a las variables de entrada y un valor correspondiente a la salida. De las variables de entrada, 7 toman valores reales (edad, presión de sangre en reposo, concentración de colesterol, pulsaciones máximas por minuto, índice de repolarización ventricular producida por el ejercicio con respecto al estado de reposo, índice de duración de dicho ejercicio y número de vasos sanguíneos mayores que resultan coloreados mediante fluoroscopia), 3 toman valores binarios (sexo, nivel de azúcar en sangre superior a 120 mg/dl y si el ejercicio produjo dolor de pecho) y 3 corresponden a valores nominales (resultados del elec-

Generaciones	*Porcentaje de Aciertos*	*Neuronas*	*Parámetros*	*Tiempo (segs.)*
10	88 ± 2	12 ± 6	12.15 %	13 ± 1
25	87 ± 2	12 ± 4	12.15 %	35 ± 6
50	87 ± 3	18 ± 5	18.22 %	88 ± 30
75	86 ± 3	14 ± 6	15.29 %	118 ± 30
100	85 ± 2	17 ± 4	17.21 %	206 ± 71

Tabla 5.19: Resultados de EvRBF en la clasificación de la base de datos *Heart Disease*. Las distintas columnas muestran el porcentaje de aciertos, tamaño de red, porcentaje de parámetros libres que supone dicho tamaño con respecto al número de datos presentes en los conjuntos de entrenamiento y test, y tiempos de ejecución. Estos datos están promediados para 5 ejecuciones de EvRBF y para distintos valores del número de generaciones utilizadas.

trocardiograma en reposo, tipo de dolor de pecho y estado del corazón, con 3, 4 y 3 valores, respectivamente). No obstante, todos los valores son numéricos y por tanto manipulables por RNFBR . En cuanto a la salida, puede tomar los valores 1 y 2, indicando la ausencia o presencia de enfermedad de corazón, respectivamente.

La utilización de este conjunto de datos permite comparar EvRBF con otro de los métodos hallados en la literatura ideados para la generación de RNFBR, en este caso, el método de Leonardis y Bischof [Leon98], cuyo funcionamiento fue ya comentado en el apartado 2.2.5. Las principales características de este método son, en primer lugar, que realiza operaciones para modificar los centros, radios y pesos de la red mediante el uso de técnicas de descenso de gradientes; y, en segundo lugar, que parte de una red sobre-especificada, a la cual va eliminando neuronas de forma progresiva haciendo una selección basada en el principio de longitud de descripción mínima (MDL, *Minimum Description Length*). Básicamente, este principio consiste en evaluar distintos subconjuntos formados con las neuronas existentes, quedándose con el más pequeño que sea capaz de mantener el porcentaje de aciertos de la red por encima de un determinado umbral. La utilización combinada de ambas características les permite obtener redes de tamaño aceptable y alta capacidad de predicción.

El comportamiento que ofrece EvRBF en el problema de clasificación de *Heart Disease* puede verse en la tabla 5.19. Se puede comprobar que el incremento de generaciones conlleva un empeoramiento tanto en los porcentajes de aciertos de como en los tamaños de las redes generadas. Esto indica que, para este problema, el algoritmo induce el sobre-entrenamiento de las redes, permitiendo una mejor clasificación de los datos presentes tanto en el conjunto de entrenamiento como en el de validación, pero desvirtuando de forma progresiva la capacidad de generalización de las redes

Algoritmo		*Porcentaje de Aciertos*	*Tamaño Red*
Leonardis-Bishop		86	16
Gaussian Masking		82	24
EvRBF	*10 gen.*	**88**	12
EvRBF	*25 gen.*	87	12

Tabla 5.20: Comparación del porcentaje de aciertos conseguido por el método de Leonardis y Bischof [Leon98], el método denominado *Gaussian Masking* (cuyos resultados se han tomado del mismo trabajo) y por EvRBF en la clasificación de la base de datos *Heart Disease*. Se incluye porcentaje de aciertos y el tamaño para distintos números de generaciones en EvRBF. Se han omitido los resultados relativos a 50, 75 y 100 generaciones por producir redes inaceptablemente grandes. Puede comprobarse que, tanto para 10 como para 25 generaciones, EvRBF alcanza tasas de acierto mejores que las presentadas por el resto de métodos. No obstante, como se mostraba en la tabla 5.19, un mayor número de generaciones hace que los individuos se sobre-entrenen, decayendo su capacidad de generalización que, no obstante, se mantiene similar al del resto de métodos.

creadas. Además, para números de generaciones mayores que 25, el porcentaje de parámetros libres que componen las redes es inaceptablemente elevado, por lo que no podemos considerarlas soluciones válidas para este problema.

Destaca también el hecho de que sea mayor el tiempo empleado por EvRBF en este problema en comparación con el utilizado en todos los anteriores (así, por ejemplo, al utilizar 100 generaciones, el tiempo empleado en *Heart Disease* es unas 2 veces el empleado en la función *Friedman*, 5 veces el empleado para aproximar las funciones f_1 a f_4, 8 veces el que se necesita para la clasificación de *Iris* y 10 veces el necesario para aproximar la función polinomial *Hermite*). La causa reside en que, para este problema, la dimensión del espacio de entradas es notablemente mayor que la del resto de problemas considerados, lo cual conlleva más aplicaciones de los operadores genéticos que actúan sobre neuronas concretas, así como un mayor coste en la computación de los pesos sinápticos entre la capa oculta y la capa de salida.

La tabla 5.20 muestra los resultados que ha ofrecido EvRBF, en cuanto al porcentaje de patrones del conjunto de test que consiguen clasificar correctamente las redes que genera. Dichos resultados se comparan con los obtenidos por el método de Leonardis y Bischof [Leon98] y con el método denominado *Gaussian Masking* [Roy95] que utiliza un algoritmo de agrupamiento y programación lineal para generar redes neuronales. Los resultados relativos a este último están tomados del propio trabajo de Leonardis. En dicha tabla se muestran, además, los tamaños de red conseguidos por los distintos métodos. No obstante, se han suprimido los valores

correspondientes a EvRBF para 50, 75 y 100 generaciones pues, como se aprecia en la tabla 5.19, las redes que se generan tienen tamaños inaceptables. A la vista de todos estos resultados, podemos observar que EvRBF mejora levemente los resultados alcanzados por los otros dos métodos, con tamaños de redes más pequeños. Aún más, incluso si considerásemos las redes sobre-entrenadas generadas por EvRBF cuando emplea más de 25 generaciones, podríamos anotar que su capacidad de generalización es aproximadamente igual que la de Leonardis y Bischof y sensiblemente superior a la ofrecida por *Gaussian Masking*.

5.3.3. Clasificación de la base de datos *Cáncer*

Para finalizar con esta sección de evaluación de EvRBF en problemas de clasificación, se ha utilizado la conocida base de datos *Cáncer*, utilizada inicialmente por Prechelt [Prec94]. Los datos que recoge pertenecen a un problema de diagnóstico de cáncer de pulmón en el que se intenta determinar si un determinado tumor es maligno o benigno en función de la descripción de las células analizadas mediante examen al microscopio. Al igual que los anteriores problemas, el conjunto de datos ("*Wisconsin breast cancer database*"), proviene del repositorio de pruebas para aprendizaje automático del UCI [Blak98].

Cada uno de los patrones que componen los conjuntos de entrenamiento y test contiene 9 atributos de entrada (grosor del grupo de células, uniformidad del tamaño de las células, uniformidad de la forma de las células, adhesión marginal, tamaño del epitelio de una sola célula, núcleos desnudos, suavidad cromática, núcleos normales, mitosis) y un atributo de salida (0 -benigno- ó 1 -maligno). El 65.5 % de los patrones pertenecen a tumores benignos y el 34.5 % a tumores malignos.

En realidad, existen tres versiones distintas del problema, en función de la separación que se hace entre el conjunto de entrenamiento y el conjunto de test. Para la evaluación con este problema hemos utilizado el llamado ***Cancer1a***, que es el conjunto utilizado por Grönroos [Grön98] de los tres propuestos por Prechelt [Prec94]. Para esta variante del problema, el conjunto de entrenamiento está compuesto por 524 patrones, mientras que el de test tiene 124.

Dentro de la literatura revisada, no se ha encontrado ningún trabajo relacionado con RNFBR en el que se trate la clasificación de esta base de datos. Por este motivo, los datos obtenidos por EvRBF van a ser comparados con los que devuelve el método G-Prop [Cast00] y con otros similares citados por el autor en el mismo trabajo. G-Prop es un AE de similares

Generaciones	*Porcentaje de Aciertos*	*Neuronas*	*Parámetros*	*Tiempo (segs.)*
10	98.9 ± 0.4	29 ± 12	10.94 %	81 ± 32
25	98.7 ± 0.3	29 ± 10	10.94 %	219 ± 87
50	98.7 ± 0.5	27 ± 14	10.18 %	481 ± 220
75	98.4 ± 0.7	29 ± 16	10.94 %	695 ± 459
100	98.5 ± 0.8	29 ± 13	10.94 %	909 ± 442

Tabla 5.21: Resultados de EvRBF en la clasificación de la base de datos *Cáncer*. Las columnas muestran, para distintos número de generaciones, el porcentaje de aciertos, tamaño de red, porcentaje de parámetros libres que supone dicho tamaño con respecto al número de datos presentes en el conjunto de test, y tiempo de ejecución expresado en segundos. Los resultados están promediados sobre 5 ejecuciones del algoritmo para cada número de generaciones considerado.

características a EvRBF pero diseñado para la evolución de perceptrones multicapa. Incorpora así operadores de adición y eliminación de neuronas, al igual que de entrecruzamiento. Una característica notable del mismo es que el entrenamiento de las redes, realizado mediante *Back-Propagation*, es usado como un operador genético más, aplicado sólo cuando es seleccionado en función de su probabilidad de aplicación.

Para la base de datos sobre *Cáncer*, EvRBF arroja los resultados que se muestran en la tabla 5.21. Como se puede observar, a medida que se utilizan más generaciones, el porcentaje de aciertos empeora, el tamaño permanece igual y el tiempo de ejecución prácticamente se duplica para cada nuevo valor del número de generaciones. Así, cuando utilizamos sólo 10 generaciones, obtenemos un porcentaje de aciertos del 98.9 %, con redes compuestas por 29 neuronas en promedio y necesitando 81 segundos para ejecutarse. Sin embargo, al duplicar el número de generaciones, el porcentaje de aciertos es 2 décimas peor (98.7 %), manteniéndose el tamaño de red y triplicándose el tiempo de ejecución. La situación es por tanto análoga a la del anterior problema, *Heart Disease*: un mayor número de generaciones lleva a que las redes generadas se especialicen en los datos usados para entrenamiento y validación, haciendo que merme la capacidad de generalización de la red de forma progresiva. En cualquier caso, el número de parámetros libres que en promedio componen las redes nos permiten afirmar que, para este problema, EvRBF devuelve modelos de clasificación válidos independientemente del número de generaciones durante las que sea iterado.

En la tabla 5.22 se pueden observar los distintos porcentajes de aciertos conseguidos por los métodos de Prechelt y Grönroos, por perceptrones multicapa entrenados mediante *Quick-Propagation*, por los métodos - SA-Prop y G-Prop (creación y entrenamiento de perceptrones multicapa

Algoritmo		*Porcentaje de Aciertos*	*Tamaño Red*
Prechelt		98.6	-
Grönroos		98.0	-
Quick-Propagation		97.0	38
SA-Prop		98.9	-
G-Prop		**99.1**	14
EvRBF	*10 gen.*	98.9	29
EvRBF	*25 gen.*	98.7	29
EvRBF	*50 gen.*	98.7	27
EvRBF	*75 gen.*	98.4	29
EvRBF	*100 gen.*	98.5	29

Tabla 5.22: Comparación del porcentaje de aciertos conseguido por los métodos de Prechelt y Grönroos, por perceptrones multicapa entrenados mediante *Quick-Propagation*, por los métodos SA-Prop y G-Prop (creación y entrenamiento de perceptrones multicapa mediante enfriamiento simulado y AE, respectivamente) de Castillo y por EvRBF en la clasificación de la base de datos *Cancer*. Salvo los relativos a EvRBF, los resultados mostrados están tomados de [Cast00]. Para EvRBF, se incluyen el porcentaje de aciertos y el tamaño de red conseguidos con distintos números de generaciones. Los datos reflejan que EvRBF se comporta tan bien como el resto de métodos, siendo superado exclusivamente por G-Prop, tanto en número de aciertos como en la utilización de redes más pequeñas.

mediante enfriamiento simulado y AE, respectivamente) de Castillo y por EvRBF en la clasificación de la base de datos *Cancer*. Salvo los relativos a EvRBF, todos los resultados mostrados están tomados de [Cast00].

En general, el comportamiento de EvRBF es tan bueno como el del resto de métodos considerados. Dejando aparte el empeoramiento de sus prestaciones para un mayor número de generaciones, podemos observar que el peor de los resultados ofrecidos por EvRBF (98.4 %) es ligeramente inferior al obtenido por Prechelt (98.6 %), pero mejor que los errores proporcionados por Grönroos (98.0 %) y por *Quick-Propagation* (97.0 %), que resulta ser el peor de todos, como es natural por ser un método básico de descenso de gradientes.

Por otro lado, si se comparan los mejores resultados, podemos ver que el mejor de los ofrecidos por EvRBF (98.9 %) se alcanza cuando se itera el algoritmo durante 10 generaciones. Dicho resultado es mejor que todos los comentados anteriormente, igual al conseguido por Castillo con su algoritmo SA-Prop y sólo peor que G-Prop, del mismo autor. No obstante, si bien la diferencia entre G-Prop y EvRBF en el porcentaje de patrones bien clasificados es ciertamente pequeña (99.1 % frente a 98.9 %, respectivamente), el primero destaca en lo concerniente al tamaño de las redes: G-Prop genera redes con la mitad de neuronas que las generadas por EvRBF y casi la tercera parte de las que es necesario utilizar con el algoritmo de entrenamiento *Quick-Propagation* para obtener resultados comparables.

5.4. Evaluación de EvRBF en estimación de Series Temporales

EvRBF ha sido utilizado también en la tarea de estimar valores para series temporales. La predicción de una serie temporal es un problema similar al de una aproximación funcional en la que las variables de entrada toman valores que a su vez son salidas generadas previamente por la propia función. De forma general, el problema que plantean las series temporales radica en que, a priori, no se conoce la expresión de la función que la define, ni en su forma ni en lo relativo a qué valores de los ya generados hay que utilizar para determinar las nuevas salidas de la serie.

En la presente tesis se ha tratado sólo el problema de la determinación de la forma específica de la función que define la serie temporal. Por este motivo, se han utilizado tres series temporales sintéticas, para las cuales se conoce el número de variables de entrada y los valores que éstas toman. Las dos primeras están tomadas del trabajo original de Lowe y Broomhead [Broo88] y fueron utilizadas, precisamente, para introducir el concepto de RNFBR. La tercera es la conocida serie de Mackey y Glass, utilizada frecuentemente para evaluar todo tipo de métodos de predicción.

Existe una cuarta serie temporal formada por datos verídicos que describe la tasa de cambio entre libras británicas y dólares americanos. Los datos han sido promediados semanalmente y corresponden el período comprendido entre 31 de diciembre de 1979 y 26 de octubre 1983[1], y ha sido utilizada por Sheta y De Jong [Shet01]. No obstante, a pesar de que EvRBF ha sido evaluado sobre ella y obtiene, en principio, mejores resultados que los alcanzados por dichos investigadores, existen lagunas dentro de la descripción del trabajo que realizan que nos impiden asegurar que se estén realizando los mismos experimentos y que, por tanto, los resultados sean comparables. A este respecto, han resultado infructuosos los intentos por obtener más información sobre dicha publicación por lo que, finalmente, se ha optado por no incluir el trabajo desarrollado para evaluar EvRBF sobre esta serie temporal[2].

[1] Esta serie temporal está accesible *http://pacific.commerce.ubc.ca/xr/data.html*, gracias al trabajo realizado por el profesor Werner Antweiler, de la Universidad de British Columbia, Vancouver, Canada.

[2] No obstante, el autor de esta tesis desea hacer público su agradecimiento al Dr. De Jong por el interés mostrado en lo concerniente a este tema, como así pudo comprobar en conversación privada mantenida durante la celebración del congreso PPSN 2003.

5.4.1. Aproximación de las series temporales *mapa doble* y *mapa cuadrático*

La primera serie de experimentos relacionados con la estimación de series temporales se ha realizado sobre los mismos problemas con los que se enfrentaron los pioneros en el uso de RNFBR. Ambas series han sido tomadas del trabajo de Lowe y Broomhead [Broo88], los cuales se refieren a ellas como el **mapa doble** y el **mapa cuadrático**.

Para ambas series temporales, el valor de la función para un instante t va a quedar determinado por el valor de la misma en el instante $t-1$. Así, vamos a considerar el mapa doble, T_d, como una secuencia ordenada de la función expresada en la ecuación 5.15. La representación de dicha función en el intervalo $[0,1]$ es la mostrada por la figura 5.14.

$$x_{n+1} = 2x_n(módulo1) \tag{5.15}$$

De igual forma, consideraremos el mapa cuadrático como una secuencia ordenada de la función expresada en la ecuación 5.16, cuya representación es la mostrada en la fig 5.15.

$$x_{n+1} = 4x_n(1 - x_n) \tag{5.16}$$

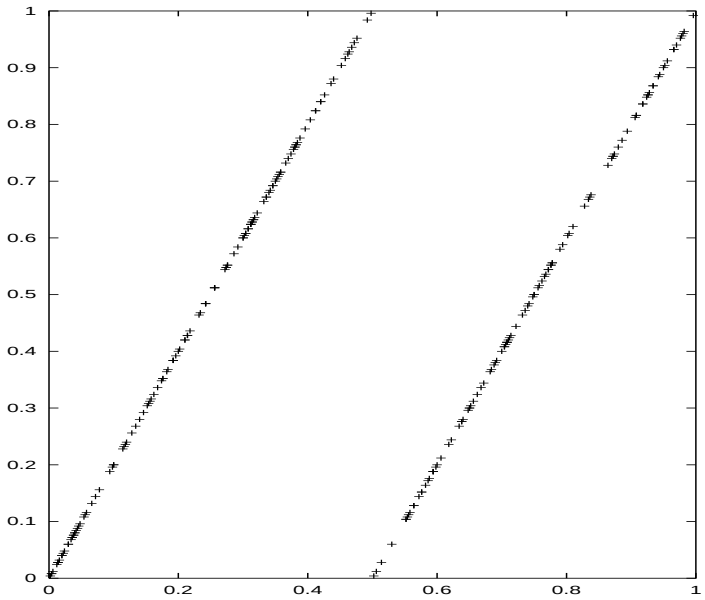
Ambas funciones, mapa doble y mapa cuadrático, son conocidas por ser caóticas en el intervalo $[0,1]$.

La estrategia utilizada por Broomhead y Lowe consistió en tomar como centros para las FBR de la red neuronal un número n de valores uniformemente espaciados en el intervalo $(0,1)$. El valor de n, esto es, el número de neuronas de la capa oculta era uno de los parámetros ajustables del modelo.

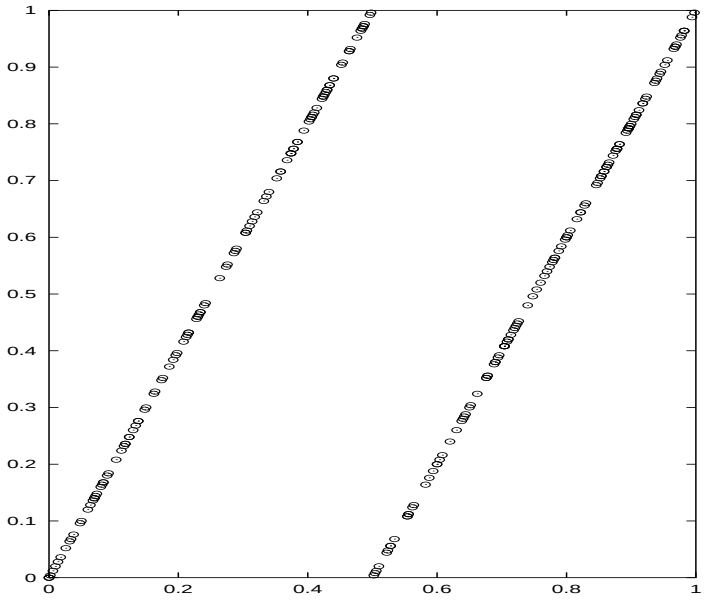
Una vez establecida la red neuronal, se utilizaron 250 puntos como conjunto de entrenamiento para calcular los pesos de las conexiones entre la capa oculta y la capa de salida. La bondad de la red final construida se comprobó haciendo que estimase los valores de otros 250 puntos distintos a los anteriores.

Como conclusión general los autores destacaban cómo el incremento en el número de neuronas utilizadas mejoraba la fiabilidad de las predicciones hechas por la red. No obstante, ésto ocurría hasta alcanzar un determinado valor tras el cual un aumento en el tamaño de la red llevaba a menores errores en la predicción de los puntos usados para entrenamiento, pero mayores en la predicción de puntos usados para la generalización.

Para ambos problemas, las resultados relativos a errores de generalización van a expresarse utilizando el logaritmo del error normalizado

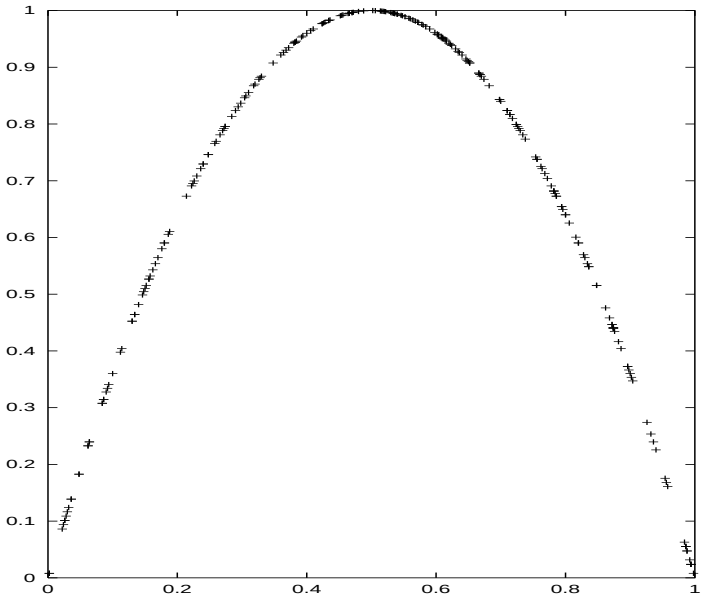


(a) Valores de salida del fichero de entrenamiento

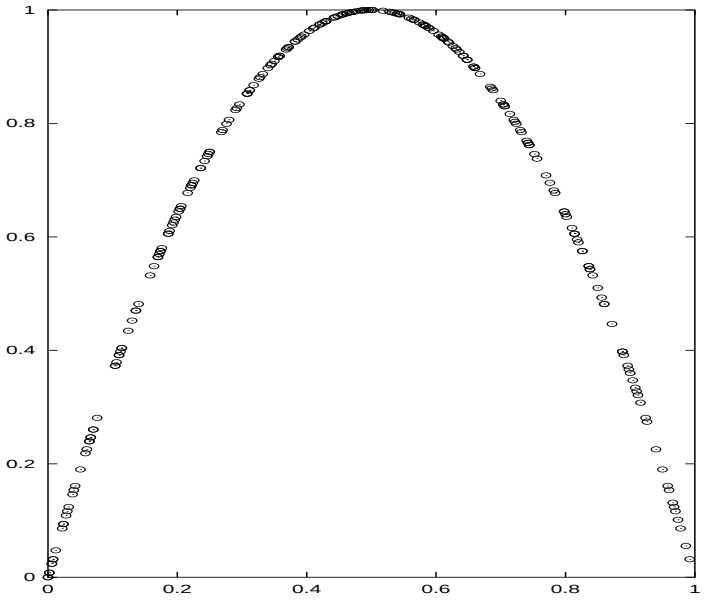


(b) Valores de salida del fichero de test

Figura 5.14: Representación gráfica de los valores de salida de la serie temporal *mapa doble*, definida en la ec. 5.15.



(a) Valores de salida del fichero de entrenamiento



(b) Valores de salida del fichero de test

Figura 5.15: Representación de los valores de salida de la serie temporal *mapa cuadrático*, definida en la ec. 5.16.

Generaciones	*log(ECMN)*	*Neuronas*	*Parámetros*	*Tiempo (segs.)*
10	-2.1 $\pm$ 0.2	18 $\pm$ 1	7.36 %	8.8 $\pm$ 0.4
25	-2.1 $\pm$ 0.4	20 $\pm$ 4	7.92 %	23.4 $\pm$ 0.5
50	-2.3 $\pm$ 0.5	20 $\pm$ 10	7.84 %	44.0 $\pm$ 14.5
75	-2.6 $\pm$ 0.8	31 $\pm$ 2	12.24 %	84.8 $\pm$ 4.3
100	-2.2 $\pm$ 0.5	26 $\pm$ 1	10.48 %	115.2 $\pm$ 10.8

Tabla 5.23: Resultados de EvRBF en la estimación de la serie temporal *mapa doble*, definida en la ec. 5.15. Las distintas columnas muestran el logaritmo del ECM normalizado (log(ECMN)), tamaño de red, porcentaje de parámetros libres que supone dicho tamaño con respecto al número de datos presentes en los conjuntos de entrenamiento y test, y tiempo de ejecución expresado en segundos. Esto valores están promediados para 5 ejecuciones de EvRBF por cada uno de los distintos números de generaciones empleados.

Algoritmo		*log(ECMN)*	*Tamaño Red*
Broomhead y Lowe		-0.85	50
Broomhead y Lowe		-0.7	100
Broomhead y Lowe		-0.9	150
Broomhead y Lowe		-1.15	200
EvRBF	*10 gen.*	-2.1	18
EvRBF	*25 gen.*	-2.1	20
EvRBF	*50 gen.*	-2.3	20
EvRBF	*75 gen.*	**-2.6**	31
EvRBF	*100 gen.*	-2.2	26

Tabla 5.24: Comparación del logaritmo del ECM normalizado (log(ECMN)) conseguido Broomhead y Lowe [Broo88] con distintos tamaños de red y por EvRBF en la estimación de la serie temporal *mapa doble*, definida en la ec. 5.15. Se ha incluido el log(ECMN) y el tamaño de red para distintos números de generaciones en EvRBF. Los resultados muestran que, independientemente del número de generaciones, EvRBF obtiene un error más pequeño utilizando muchas menos neuronas que los autores Broomhead y Lowe.

(log(ECMN)), medida que usaron Broomhead y Lowe y que se define como:

$$Error = log_{10}\left(\frac{ECM}{\sigma^2}\right) \tag{5.17}$$

donde ECM es el error cuadrático medio cometido por la red sobre el conjunto de test y σ^2 es la desviación estándar de los valores de salida de dicho conjunto.

Resultados en las tablas 5.23 y 5.24, así como en las tablas 5.25 y 5.26,

Comenzando el análisis de resultados por el problema del mapa doble, la tabla 5.23 refleja que el mejor valor se obtiene cuando se utilizan 75 generaciones (log(ECMN)=-2.6), aunque en general los errores son muy similares entre sí, independientemente de las generaciones empleadas. Además,

Generaciones	*log(ECMN)*	*Neuronas*	*Parámetros*	*Tiempo (segs.)*
10	-10.8 $\pm$ 0.4	18 $\pm$ 2	7.20 %	8.8 $\pm$ 0.4
25	-11.2 $\pm$ 0.4	21 $\pm$ 2	8.40 %	22.0 $\pm$ 2.9
50	-11.6 $\pm$ 0.5	22 $\pm$ 2	8.80 %	44.8 $\pm$ 2.4
75	-12.4 $\pm$ 1.1	28 $\pm$ 3	11.20 %	77.6 $\pm$ 6.5
100	-12.4 $\pm$ 0.9	22 $\pm$ 5	8.80 %	98.0 $\pm$ 20.5

Tabla 5.25: Resultados de EvRBF en la estimación de la serie temporal *mapa cuadrático*, definida en la ec. 5.16. Las distintas columnas muestran el logaritmo del ECM normalizado (log(ECMN)), tamaño de red, porcentaje de parámetros libres que supone dicho tamaño con respecto al número de datos presentes en los conjuntos de entrenamiento y test, y tiempo de ejecución expresado en segundos. Esto valores están promediados para 5 ejecuciones de EvRBF por cada uno de los distintos números de generaciones empleados.

Algoritmo		*log(ECMN)*	*Tamaño Red*
Broomhead y Lowe		-8.0	50
Broomhead y Lowe		-8.7	100
Broomhead y Lowe		-8.1	150
Broomhead y Lowe		-8.0	200
EvRBF	*10 gen.*	-10.8	18
EvRBF	*25 gen.*	-11.2	21
EvRBF	*50 gen.*	-11.6	22
EvRBF	*75 gen.*	**-12.4**	28
EvRBF	*100 gen.*	**-12.4**	22

Tabla 5.26: Comparación del logaritmo del ECM normalizado (log(ECMN)) conseguido Broomhead y Lowe [Broo88] con distintos tamaños de red y por EvRBF en la estimación de la serie temporal *mapa cuadrático*, definida en la ec. 5.16. Se ha incluido el log(ECMN) y el tamaño de red para distintos números de generaciones en EvRBF. Se puede observar que para cualquier número de generaciones con que sea ejecutado EvRBF, obtiene una mejor estimación de la serie temporal que la realizada por Broomhead y Lowe, con la ventaja de que además se utilizan menos neuronas.

en todos los casos se generan redes cuyo número de parámetros libres es inferior o sobrepasa levemente el margen aceptable del 10 % sobre el conjunto de datos presente en los ficheros de entrenamiento y test.

En cuanto a la comparación con los resultados ofrecidos por Broomhead y Lowe, en la tabla 5.24 puede verse que, independientemente del número de neuronas usadas por tales autores, EvRBF consigue errores mucho más pequeños (entre -2.6 y -2.1, frente a -1.15 ó -0.9) con redes que son aproximadamente la mitad de voluminosas que las más pequeñas empleadas por dichos investigadores (entre 18 y 31 neuronas usadas por EvRBF, frente a 50 utilizadas por ellos).

En cuanto a los resultados para el problema del mapa cuadrático, la tabla 5.25 muestra, en primer lugar, que el error más pequeño (-12.4) se

consigue usando 75 ó 100 generaciones. Además, no se produce sobreentrenamiento en las redes y, a medida que se utiliza un mayor número de generaciones, se consigue disminuir el error alcanzado. En cuanto al tamaño de las redes, se pueden observar que en promedio se generan modelos válidos para la estimación de esta serie temporal.

En términos comparativos, los datos consignados en la tabla 5.26 muestran claramente la ventaja que supone la utilización de EvRBF, dado que se consigue mejorar el error desde -8.7 (mejor error alcanzado por Broomhead y Lowe) hasta -12.4, esto es, 4 órdenes de magnitud inferior, al estar expresado el error en escala logarítmica.

Otro tanto ocurre con el tamaño de las redes, pues EvRBF consigue mejorar el error asociado a redes de 50 neuronas, utilizando exclusivamente 10 generaciones y creando redes con 18 neuronas por término medio.

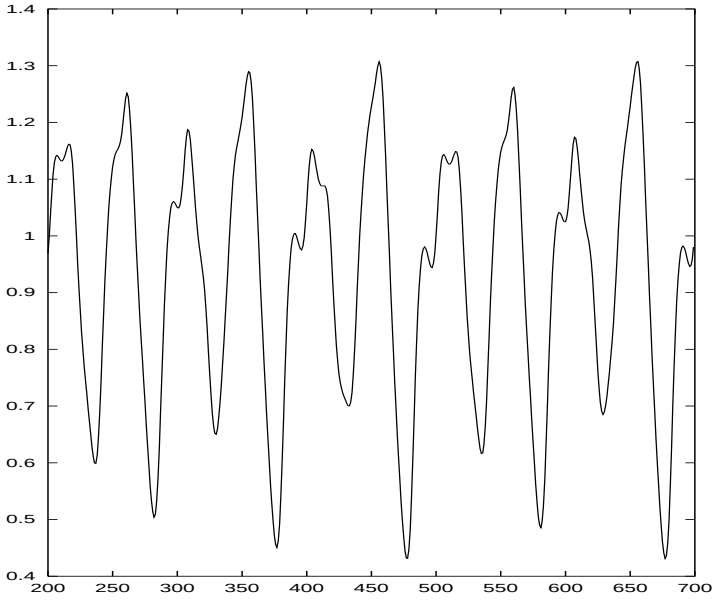
5.4.2. Estimación de la serie temporal de *Mackey y Glass*

Como último ejemplo de la evaluación que se ha hecho de EvRBF se ha considerado el problema de estimar la serie temporal caótica creada por Mackey y Glass. Los términos del problema son los mismos descritos en [Plat91b], habiendo sido utilizada esta serie temporal en gran cantidad de trabajos, de entre los que destacamos el de Whitehead y Choate [Whit96], el de Carse y Fogarty [Cars96] y el de González [Gonz01a]. Al igual que ellos, se desea predecir el valor de la serie temporal para un instante $t + 85$ a partir de los valores almacenados en los instantes t, $t - 6$, $t - 12$ y $t - 18$, utilizando para ello un conjunto de entrenamiento compuesto de 500 pares entrada-salida.

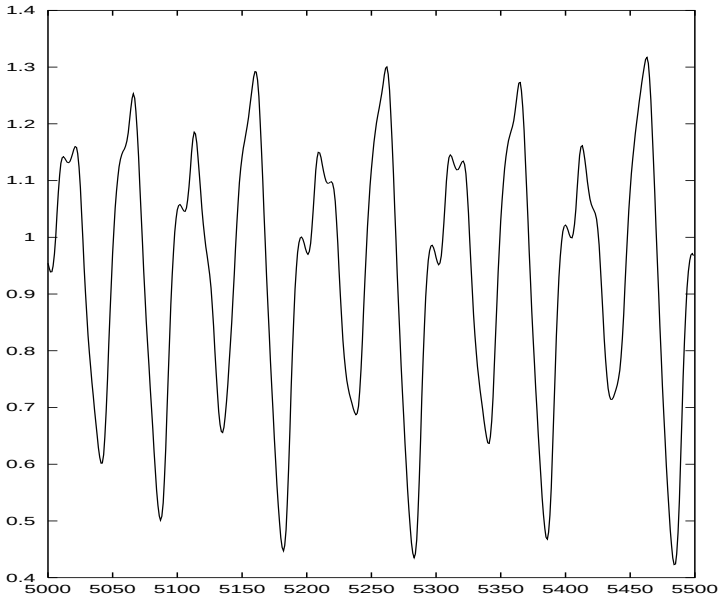
La ecuación diferencial que define la serie temporal de Mackey y Glass es la correspondiente a la ecuación 5.18. Los datos utilizados para estos experimentos corresponden a esta serie cuando se utilizan los valores $a = 0.2$, $b = 0.1$ y $T = 17$. La representación gráfica de esta serie temporal es la que muestra la figura 5.16.

$$\frac{dx(t)}{dt} = -bx(t) + a\frac{x(t-T)}{1+x(t-T)^{10}} \tag{5.18}$$

En realidad, la obtención de los datos que conforman la serie de Mackey y Glass puede realizarse de dos formas. En primer lugar se puede aplicar la ecuación 5.18, siguiendo un método denominado **Runge-Kutta de segundo orden**, con incremento igual a 0.1. En segundo lugar, tales datos pueden ser descargados de la colección de recursos para aprendizaje



(a) Valores de salida del fichero de entrenamiento



(b) Valores de salida del fichero de test

Figura 5.16: Representación de los valores de salida de la serie temporal de *Mackey y Glass*, definida en la ec. 5.18.

Generaciones	*RECMN*	*Neuronas*	*Parámetros*	*Tiempo (segs.)*
10	0.301 ± 0.006	37 ± 2	9.62 %	52 ± 1
25	0.247 ± 0.010	39 ± 3	10.14 %	150 ± 20
50	0.209 ± 0.006	47 ± 1	12.22 %	359 ± 12
75	0.197 ± 0.020	48 ± 3	12.48 %	514 ± 54
100	0.185 ± 0.009	55 ± 4	14.30 %	803 ± 72

Tabla 5.27: Resultados de EvRBF en la estimación de la serie temporal *Makecy y Glass*, definida en la ec. 5.18. Las distintas columnas muestran la raíz cuadrada del ECM normalizado (RECMN), tamaño de red, porcentaje de parámetros libres que supone tal tamaño con respecto al número de datos presentes en los conjuntos de entrenamiento y test, y tiempo de ejecución expresado en segundos. Estos datos están promediados para 5 ejecuciones del AE con cada uno de los valores de número de generaciones.

de la CMU[3]. Esta segunda forma es la que hemos utilizado.

Existen distintos problemas de estimación que se pueden formular a partir de la serie de Mackey y Glass. El planteado en esta tesis consiste en estimar el valor de la serie en el instante $t + 85$ conocidos los valores de las misma en los instantes t, $t - 6$, $t - 12$ y $t - 18$. De esta forma, se han generado sendos conjuntos de entrenamiento y test, en los cuales cada muestra contiene 4 valores correspondientes a las entradas y 1 valor de salida. Tal y como hiciera Platt, los datos que forman el conjunto de entrenamiento son los que están en el rango de $t = 200$ a $t = 700$, mientras que los que integran el conjunto de test pertenecen al rango $t = 5000$ a $t = 5500$.

En cuanto a la forma de medir el error cometido en la estimación del conjunto de test, se ha utilizado la raíz cuadrada del ECM normalizado (RECMN), como harán el resto de algoritmos con los que va a compararse EvRBF.

Por su parte, los métodos con los que compararemos los resultados de EvRBF son los anteriormente citados de Carse y Fogarty [Cars96], Whitehead y Choate [Whit96], y González [Gonz01a], pues, al igual que EvRBF, utilizan AE para configurar RNFBR que estimen esta serie temporal. En el apartado 2.3.3 se pueden encontrar descripciones de la forma en que cada uno de ellos actúa.

De entre los resultados obtenidos por EvRBF, presentes en la tabla 5.27, podemos destacar que existe una mejora de la reducción del error conforme se utiliza un mayor número de generaciones. Así, para 10 generaciones el error obtenido es de 0.301, mientras que cuando se usan 100 generaciones, dicho error se reduce hasta 0.185. No obstante, y como ya viene siendo habitual, este descenso se consigue a costa de un notable incremento del

[3]Disponible como parte del *CMU Learning Benchmark Archive*, accesible en múltiples direcciones Web.

Algoritmo		*RECMN*	*Tamaño Red*
Carse y Fogarty		0.25	40
Carse y Fogarty		0.18	60
Carse y Fogarty		0.15	100
Whitehead y Choate		0.18	50
Whitehead y Choate		0.11	75
Whitehead y Choate		**0.05**	125
González		0.09	25
EvRBF	*10 gen.*	0.30	37
EvRBF	*25 gen.*	0.25	39
EvRBF	*50 gen.*	0.21	47
EvRBF	*75 gen.*	0.20	48

Tabla 5.28: Comparación de la raíz cuadrada del ECM normalizado (RECMN) conseguido por los métodos descritos en [Cars96], [Whit96], [Gonz01a] y por EvRBF en la estimación de la serie temporal de *Mackey y Glass*, definida en la ec. 5.18. Se incluyen la RECMN y el tamaño para distintos números de generaciones en EvRBF. Los resultados obtenidos por EvRBF son comparables a los obtenidos por Whitehead-Choate y Carse-Fogarty, para igual número de neuronas. No obstante, tales autores consiguen mejores resultados conforme se añaden nuevas neuronas, cosa que en este caso no ocurre a EvRBF. En cualquier caso, destacan los errores obtenidos por Whitehead y Choate y por González, siendo éste último el que utiliza las redes más pequeñas de todos los métodos.

número de neuronas que hacen que para 100 generaciones el porcentaje de parámetros libres que componen en promedio las redes generadas sea excesivamente superior al 10 %.

La tabla 5.28 nos permite comparar los resultados obtenidos por EvRBF- y el resto de métodos ya citados. Dicha tabla muestra que nuestro método obtiene errores comparables a los obtenidos por Whitehead-Choate y Carse-Fogarty, para tamaños de redes similares. Sin embargo, tales autores consiguen mejores resultados conforme se añaden nuevas neuronas, cosa que ocurre a EvRBF según se ve en la tabla 5.27, aunque a costa de obtener redes inaceptablemente grandes. En cualquier caso, destacan los errores obtenidos por Whitehead y Choate, así como por González, dado que son un orden de magnitud inferiores a los demás. Adicionalmente, éste último es el que genera las redes más pequeñas de todos los métodos, consiguiendo el mejor ratio entre error y tamaño de red.

5.5. Conclusiones

En este capítulo se han detallado los resultados ofrecidos por EvRBF cuando es aplicado a problemas de aproximación funcional, clasificación de patrones y estimación de series temporales. Los distintos problemas re-

cogen casos tanto de distintas dimensiones del espacio de entrada, como de distintas tasas de ruido aplicadas a las salidas deseadas. Del mismo modo, se han incluido tablas que permiten comparar los resultados alcanzados por EvRBF en cuanto a error y número de neuronas con los alcanzados por distintos basados también en RNFBR.

Los resultados muestran que EvRBF puede ser aplicado satisfactoriamente al tipo de problemas que se han considerado. En gran parte de los casos, EvRBF obtiene los errores más pequeños (o en el caso de clasificación, los mayores índices de acierto) utilizando redes de tamaño comparable al de resto de métodos, cuando no menores y, en cualquier caso, siempre utilizando un número de parámetros libres menor o muy cercano al 10 % del número de datos que componen los distintos conjuntos de entrenamiento y test.

Cabe destacar los resultados obtenidos por EvRBF en los problemas de aproximación funcional en los cuales se añade error a las salidas esperadas. Esto, unido a la relativa rapidez de ejecución (teniendo en cuenta que es un AE), permite contemplar su utilización en entornos reales en los que los datos de salida puedan sufrir alteraciones o en los que exista alguna incertidumbre en cuanto al valor real de salida que se debe asignar a cada vector de entradas.

CAPÍTULO 6 / CONCLUSIONES

El trabajo de tesis doctoral presentado en esta memoria ha reunido dos métodos computacionales (redes neuronales artificiales de funciones base radiales y computación evolutiva) con el objeto de automatizar el diseño de redes neuronales artificiales para aproximación de funciones, clasificación de patrones y estimación de series temporales.

En la primera parte de este trabajo se presentó una nueva abstracción de computación evolutiva (Objetos Evolutivos), que engloba todos los paradigmas de la computación evolutiva, abstrayendo las características comunes a todos ellos.

Posteriormente, se describieron los aspectos teóricos de las redes neuronales artificiales, centrándonos en el tipo de red cuya optimización ha sido objeto de este trabajo: las redes neuronales de funciones base radiales (RNFBR). De este modo, se presentaron los principales problemas que plantea el diseño de redes neuronales artificiales: establecimiento del tamaño de la capa de neuronas ocultas y determinación de las componentes que configuran cada neurona oculta.

El análisis de dichos problemas y de los distintos enfoques encontrados en la bibliografía para resolverlos, nos llevó a establecer una metodología de diseño y a proponer un sistema que llevase a cabo una optimización global de RNFBR.

Dicho sistema, basado en AE y RNFBR, busca los principales parámetros de diseño de una red neuronal artificial para abordar problemas de aproximación funcional, clasificación de patrones y estimación de series temporales.

Las principales aportaciones y conclusiones obtenidas quedan resumidas a continuación:

- El método propuesto en esta memoria, EvRBF, determina de forma automática el número de neuronas de la capa oculta, los valores de los centros y radios asociadas a las mismas y los pesos de las conexiones sinápticas existentes entre dichas neuronas y las neuronas de la capa de salida. Para ello se basa en un algoritmo evolutivo y en un algoritmo de entrenamiento de RNFBR (capítulo 3).

- En lugar de utilizar una topología de red pre-establecida, las RNFBR de la población son inicializadas con diferente número de neuronas en su capa oculta. De esta forma no se limita al algoritmo a explorar unas predeterminadas zonas del espacio de búsqueda (capítulo 3).

- Para llevar a cabo la evolución se han propuesto varios operadores específicos, entre los que se encuentran operadores de recombinación, adición, eliminación y modificación tanto de neuronas de la capa oculta, como de los componentes de las mismas (capítulo 3).

- El AE ha sido diseñado de forma que no restrinja el crecimiento de las redes que genera, siendo los operadores genéticos los que provoquen la obtención del número óptimo de neuronas. De esta forma, la función de *fitness* se basa principalmente en el error cometido por las redes en la fase de validación para considerar una mejor que otra. Sólo en caso de *fitness* exactamente iguales (altamente improbable en problemas de aproximación funcional y estimación de series temporales) se considera mejor aquella cuyo tamaño es menor (capítulo 3).

- El método propuesto no necesita prefijar manualmente ningún parámetro acerca de la red EvRBF, solamente es necesario asignar un valor inicial para ciertos parámetros del AE. Por ello, se ha desarrollado un estudio sistemático que ha permitido establecer los valores más adecuados para estos parámetros, los cuales han sido posteriormente utilizados para evaluar la validez del método en los distintos problemas ya reseñados (capítulo 4).

- Se han mostrado los resultados obtenidos por el algoritmo ante diversos ejemplos, tanto sintéticos como reales, de los problemas que se deseaba resolver. El comportamiento mostrado por el método ante ellos demuestra su validez y robustez en la tarea de configurar una RNFBR que suponga una solución satisfactoria para cada uno de los problemas presentados (capítulo 5).

- El método propuesto es capaz de obtener redes con alta capacidad de generalización, especialmente en problemas donde las salidas se ven afectadas de errores añadidos. Además, se construyen redes que mantienen un tamaño similar al alcanzado usando otros métodos y que, generalmente, contienen un número de parámetros libres que representan entorno al 10 % o menos del número de datos presentes en los conjuntos de entrenamiento y test (capítulo 5).

Finalmente, podemos afirmar que los objetivos propuestos al desarrollar esta tesis doctoral se han cumplido:

- Se ha desarrollado un sistema, EvRBF, que hace uso de un AE y RNA para el diseño automático de RNFBR para problemas de aproximación funcional, clasificación de patrones y estimación de series temporales.
- Se ha comprobado que EvRBF es capaz de determinar automáticamente la arquitectura de red y los parámetros asociados a cada una de las neuronas de la capa oculta de la misma.
- Se han diseñado varios operadores genéticos específicos para RNFBR (recombinación, adición, eliminación y modificación de neuronas ocultas y los parámetros de las mismas).
- Se ha probado el método desarrollado con diversos problemas de aproximación funcional, clasificación de patrones y estimación de series temporales como conjuntos de datos para test. En los experimentos realizados, se han obtenido resultados satisfactorios, en muchos casos mejores que los de los métodos con que se comparaba, todo ello con mínimas necesidades de ajuste de parámetros.
- No se ha limitado en ningún momento el número de neuronas que pueden llegar a tener las distintas redes. De hecho, sería trivial incorporar mecanismos que impidieran añadir neuronas a las redes cuyo tamaño hace que se excedan en el número de parámetros libres. Igualmente, se podría optar por reducir el número de radios con los que cuenta cada neurona. De esta forma, disminuiría el número de parámetros y se podría seguir comparando con los distintos métodos presentes en la literatura pues, la mayoría de ellos, no abordan el tema de la generación de redes totalmente asimétricas. Sin embargo, se ha evaluado el método tal cual es, sin acudir a este tipo de soluciones que permitieran mejorar, al menos de forma superficial, el comportamiento del algoritmo.

Durante el desarrollo del presente trabajo, y directamente relacionados con él, se han remitido, a diferentes revistas y congresos internacionales, las siguientes publicaciones:

- J.J. Merelo, M. G. Arenas, J. Carpio, P.A. Castillo, V. M. Rivas, G. Romero, M. Schoenauer. **Evolving Objects**. *In M. Graña, editor, FEA (Frontiers of Evolutionary Algorithms) proceedings. 2000.*
- J.J. Merelo, J. Carpio, P.A. Castillo, V. Rivas, G. Romero. **Evolving Objects**. *in Proc. of Int'l Workshop on Evolutionary Computation (IWEC'2000) pp. 202-208. State Key Laboratory of Software Engineering. Wuhan University. 2000.*
- V. M. Rivas, J. J. Merelo, I. Rojas, G. Romero, P. A. Castillo, J. Carpio: **Evolving two-dimensional fuzzy systems**, *Fuzzy Sets and Systems, online*[1] *on 28 October 2002.*
- V.M. Rivas, J.J. Merelo, P.A. Castillo. **Evolving RBF Networks**, *in proceedings of the International Workshop in Artificial and Natural Networks (IWANN'2001). José Mira, Alberto Prieto(eds.). Connectionist Models of Neurons, Learning Processes, and Artificial Intelligencce. Ed. Springer Verlag. pp. 506-513. Vol. 1084. 2001.*
- V.M. Rivas, J.J. Merelo, P.A. Castillo. **Evolved RBF Networks for Time-Series Forecasting and Function Approximation**.*In proceedings of the Parallel Problem Solving from Nature, (PPSN VII). Parallel Problem Solving from Nature - PPSN VII, 7th International Conference, pp. 505-516, 2002*
- F. Martínez , M.T. Martín , V.M. Rivas, M.C. Díaz, L.A. Ureña **Using Neural Networks for Multiword Recognition in IR** *In proceedings of Seventh International ISKO Conference. pp. 559-564. Granada. 2002*

El desarrollo del presente trabajo ha permitido identificar una serie de temas y líneas de investigación originales, que se considera de interés abordar, a corto plazo, de la siguiente manera:

- Creación de nuevos operadores que permitan afinar los valores de centros y radios ya encontrados. Estos operadores realizarían una búsqueda local en el entorno de los valores actuales.

[1] http://www.sciencedirect.com/science/article/B6V05-473FP1C-3/2/757187cb901387920d57e20bd5665d99

- Creación de nuevos operadores que permitan modificar la FBR utilizada como función de activación en cada una de las neuronas ocultas.

- Establecimiento de un mejor conjunto de valores para los parámetros utilizando el método estadístico ANOVA, el cual permite comparar significativamente los resultados ofrecidos por distintas configuraciones de estos parámetros.

- Incremento de las funcionalidades del método para que permita evolucionar el número de entradas. Esto sería especialmente interesante en el caso de problemas de estimación de series temporales.

- Implementación del modelo de RNFBR construido, así como del AE, en los lenguajes Java y Perl. De esta forma estarían disponibles, no sólo para los usuarios de la biblioteca EO, sino también para los de las bibliotecas OPEAL [Mere02] y JEO [Aren02].

A más largo plazo se consideran de gran interés los siguientes temas:

- Profundizar en la relación existente entre las RNFBR y los sistemas difusos, de forma que se pueda utilizar EvRBF para generar automáticamente éstos últimos.

- Aplicación del método a nuevos problemas reales tales como la simulación de sistemas de entrada/salida, estimación de series temporales del campo de la Economía, la detección de multipalabras en sistemas de Recuperación de Información y la clasificación de resultados de investigaciones en el campo de la Bioquímica (tanto en forma de imágenes de microscopio, como en forma de patrones compuestos por datos numéricos, nominales y lógicos relativos a pacientes con enfermedades intestinales).

- Una vez estén completamente desarrolladas las versiones de EO para otros lenguajes de programación orientados a objetos (Perl y Java, principalmente) podrían usarse modelos de objetos como *SOAP (Simple Object Access Protocol)*, *COM (Common Object Model)* o *CORBA (Common Object Request Broker Architecture)*, que permitieran programar las RNFBR en cualquier lenguaje de programación. Ésto permitiría llevar a cabo la evolución de objetos programados en diferentes plataformas y lenguajes bajo un sistema operativo que soporte los citados modelos de objetos. De esta forma se habilitaría al usuario a

programar su individuo RNFBR con las características que desee en el lenguaje de programación que él domine (no se restringe al C++). De esta forma será más fácil llevar a cabo algoritmos de metaevolución repartidos entre diferentes procesadores.

- Implementación de las RNFBR y los operadores sobre ellas definidos en formato XML. Esto permitirá compartirlos más fácilmente e insertarlos en los entornos de programación de RNA que se empiezan a desarrollar actualmente en Internet[2].

Consideramos, como principal conclusión de este trabajo, que los métodos de diseño de redes neuronales artificiales basados en algoritmos evolutivos presentan importantes ventajas frente a otros métodos de diseño, debido principalmente a la capacidad del algoritmo evolutivo para optimizar todos los parámetros que componen la red sin establecer restricciones previas.

[2]El más claro ejemplo de estos entornos es JOONE (http://joone.sourceforge.net/Documentation-Proposal.html)

Referencias Bibliográficas

[Aart89] E.H.L. Aarts and J. Korst. *Simulated Annealing and Boltzmann Machines*. John Wiley, Chechester, U.K., 1989.

[Anth99] M. Anthony and P. Bartlett. *Learning in Neural Networks : Theoretical Foundations*. Cambridge University Press, 1999.

[Aren02] Maribel García Arenas, Brad Dolin, Juan Julián Merelo, Pedro Angel Castillo, Ignacio Fernández De Viana, and Marc Schoenauer. *JEO: Java Evolving Objects*. In W. B. Langdon, E. Cantú-Paz, K. Mathias, R. Roy, D. Davis, R. Poli, K. Balakrishnan, V. Honavar, G. Rudolph, J. Wegener, L. Bull, M. A. Potter, A. C. Schultz, J. F. Miller, E. Burke, and N. Jonoska, editors, *GECCO 2002: Proceedings of the Genetic and Evolutionary Computation Conference*, page 991, New York, 9-13 July 2002. Morgan Kaufmann Publishers.

[Back91] T. Back, F. Hoffmeister, and H.P. Schwefel. *A survey of evolution strategies*. In R. K. Belew and L.B. Booker, eds., Proceedings of the Fourth International Conference on Genetic Algorithms. Morgan Kaufmann, 1991.

[Bäck96] T. Bäck. *Evolutionary Algorithms in Theory and Practice: Evolution Strategies, Evolutionary Programming, and Genetic Algorithms*. Oxford University Press, New York, 1996.

[Bish91] C. Bishop. *Improving the generalisation properties of radial basis function neural networks*. Neural Computation, 3(4):579–588, 1991.

[Bish95a] C. Bishop. *Training with noise is equivalent to Tikhonov regularization*. Neural Computation, 7(1):108–116, 1995.

[Bish95b] C.M Bishop. *Neural Networks for Pattern Recognition*. Oxford University Press, 1995.

[Blak98] C.L. Blake and C.J. Merz. *UCI Repository of machine learning databases*, 1998.

[Booc94] G. Booch. *Object-Oriented Analysis and Design*. Benjamin/Cummings, second edition, 1994.

[Bose92] Bernhard E. Boser, Isabelle Guyon, and Vladimir Vapnik. *A Training Algorithm for Optimal Margin Classifiers*. In *Computational Learing Theory*, pages 144–152, 1992.

[Broo88] D. S. Broomhead and D. Lowe. *Multivariable Functional Interpolation and Adaptative Networks*. Complex Systems, 11:321–355, 1988.

[Burd97] Ben Burdsall and Christophe Giraud-Carrier. *GA-RBF: A Self-Optimising RBF Network*. In *Proceedings of the Third International Conference on ArtificialNeural Networks and Genetic Algorithms (ICANNGA'97)*, pages 348–351. Springer-Verlag, 1997.

[Camb96] D. Cambiaghi, M. Gadola, D.Vetturi, and L. Manzo. *Genetic algorithms for tyre modelling fitting in automotive dynamics system*. In *Proceedings of the 29th ISATA*, June 1996.

[Cars96] B. Carse and T.C. Fogarty. *Fast evolutionary learning of minimal radial basis function neural networks using a genetic algorithm.* In T.C. Fogarty, editor, *Proceedings of the Evolutionary Computing*, pages 1–22. Springer-Verlag, 1996.

[Cast99] P.A. Castillo, J.J. Merelo, V. Rivas, G. Romero, and A. Prieto. *G-Prop: Global Optimization of Multilayer Perceptrons using GAs*. Submitted to Neurocomputing (2nd revision), 1999.

[Cast00] P. A. Castillo. *Optimización de perceptrones multicapa mediante algoritmos evolutivos*. PhD thesis, Universidad de Granada, 2000.

[Chen91] S. Chen, C. F. Cowan, and P. M. Grant. *Orthogonal Least Squares algorithm for learning Radial Basis Function Networks*. IEEE Transactions on Neural Networks, 2(2):302–309, 1991.

[Chng96] E. S. Chng, S. Chen, and B. Mulgrew. *Gradient Radial Basis Function Networks for Nonlinear and Nonstationary Time Series Prediction*. IEEE Transactions on Neural Networks, 7(1):190–194, January 1996.

[Cohe00a] S. Cohen and N. Intrator. *Global optimization of RBF networks. Submitted to IEEE TNN*, 2000.

[Cohe00b] Shimon Cohen and Nathan Intrator. *A Hybrid Projection Based and Radial Basis Function Architecture*. Lecture Notes in Computer Science, 1857:147–154, 2000.

[Cort95a] C. Cortes. *Prediction of Generalization Ability in Learning Machines*. PhD thesis, Department of Computer Science, University of Rochester, Rochester, NY, 1995.

[Cort95b] C. Cortes and V. Vapnik. *Support vector networks*. Machine Learning, 20:273–297, 1995.

[Cris99] Nello Cristianini and John Shawe-Taylor. *Advances in Kernel Methods - Support Vector Learning*, chapter Large Margin Classifiers and Bayesian Voting Schemes, pages 55–68. MIT Press, 1999.

[Darw59] C. Darwin. *On the origin of species by means of natural selection or the preservation of favored races in the struggle for life*. Murray, 1859.

[Davi96] L. Davis, editor. *Handbook of Genetic Algorithms*. International Thomson Publishing, January 1996.

[Doli02] Brad Dolin, M. G. Arenas, and J.J. Merelo. *Opposites Attract: Complementary Phenotype Selection for Crossover in Genetic Programming*. In *Proceedings of the Seventh Conference on Parallel Problem Solving from Nature*, volume 2439 of Lecture Notes in Computer Science, pages 142–152, Granada, Spain, September 7-11 2002. Springer-Verlag.

[Eibe95] A.E. Eiben, C.H.M. van Kemenade, and J.N. Kok. *Orgy in the Computer: Multi-Parent Reproduction in Genetic Algorithms*. in Proc. of The Third European Conference on Artificial Life (ECAL'95). Lecture Notes in Artificial Intelligence, vol. 929, pp.935-945. F. Morán, A. Moreno, J.J. Merelo, P. Chacón (Eds.). Springer-Verlag. Granada, Spain, 1995.

[Fish30] R. A. Fisher. *The genetical theory of natural selection*. Oxford, The Clarendon Press, 1930.

[Fish00] Ronald Aylmer Fisher, J. H. Bennett, and Henry Bennett. *The Genetical Theory of Natural Selection*. Oxford University Press, 2000.

[Foge66] L.J. Fogel, A.J. Owens, and M.J. Walsh. *Artificial Intelligence through Simulated Evolution*. Wiley, 1966.

[Foge95] D.B. Fogel. *Evolutionary Computation: Toward a New Philosophy of Machine Intelligence*. IEEE Press, 1995.

[Frie58] R.M. Friedberg. *A learning machine, Part I*. IBM Journal of Research, 2(1):2–13, January 1958.

[Frie59] R.M. Friedberg, B. Duhnam, and J.H. North. *A learning machine, Part II*. IBM Journal of Research, 3(3):282–287, July 1959.

[Frie91] J. H. Friedman. *Multivariate adaptative regression splines (with discussion)*. Annals of Statistics, 19:1–141, 1991.

[Frit94] B. Fritzke. *Supervised learning with growing cell structures*. In J.D. Cowan, G. Tesauro, and J. Aspector, editors, *Advances in Neural Information Processing Systems*, volume 6, pages 255–262. Morgan Kaufmann, 1994.

[Fuji87] Cory Fujiki and John Dickinson. *Using the Genetic Algorithm to Generate Lisp Source Code to Solve the Prisoner's Dilemma*. In *Proceedings of Second International Conference on Genetic Algorithms*, pages 236–240, 1987.

[Glov89] Fred Glover. *Tabu Search*. ORSA Journal on Computing, 1(3):190–206, Summer 1989. Referencia básica sobre Tabu Search.

[Gold89] D.E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley, Reading, MA, 1989.

[Gonz01a] J. González. *Identificación y Optimizacin de Redes de Funciones Base Radiales para Aproximación Funcional* . PhD thesis, Universidad de Granada, septiembre 2001.

[Gonz01b] J. González, I. Rojas, H. Pomares, and M. Salmerón. *Expert Mutation Operators for the Evolution of Radial Basis Function Neural*

Networks. In J. Mira and A. Prieto, editors, *Connectionits Models of Neurons, Learning Processes, and Artificial Intelligence*, volume 2084 of Lecture Notes in Computer Science, pages 538–545. IWANN'2001, June 2001.

[Grön98] M.A. Grönroos. *Evolutionary Design of Neural Networks*. Master of Science Thesis in Computer Science. Department of Mathematical Sciences. University of Turku., 1998.

[Gros72] S. Grossberg. *Neural Espectation: Cerebellar and retinal Analogs of Cells Fired by Learnable or Unlearned Pattern Classes*. Kybernetik, Vol. 10, pp. 59-67, 1972.

[Gros76] S. Grossberg. *Adaptive Pattern Classification and Universal Recoding, I: Parallel Development and Coding of Neural Feature Detectors*. Biological Cybernetics, vol. 23, pp. 121-134, 1976.

[Grou00] Object Management Group. *Object Management Group Home Page*. Website at http://www.omg.org/, 2000.

[Hanc92] P.J.B. Hancock. *Genetic algorithms and permutation problems: a comparison of recombination operators for neural net structure specification*. In D. Whitley and J.D. Schaffer, editors, *Proc. of the Int'l Workshop on Combinations of Genetic Algorithms and Neural Networks (COGANN-92)*, pages 108–122. IEEE Computer Society Press, 1992.

[Hayk94] S. Haykin. *Introduction to the theory of neural computation*. McMillan, 1994.

[Hebb49] D.O. Hebb. *The Organization of Behavior*. John Wiley and Sons, 1949.

[Henn99] M. Henning and S. Vinoski. *Advanced CORBA Programming with C++*. Addison-Wesley Professional Computing Series. ISBN 0-201-37927-9. Addison-Wesley Longman, Inc., 1999.

[Hern97] G. Hernández. *Máquinas de Boltzmann para la optimización combinatoria y la simulación del aprendizaje*. Tesis doctoral. Depto. de Ciencias de la Computación e Inteligencia Artificial, Universidad del País Vasco, 1997.

[Hick86] Joseph F. Hicklin. *Application of the Genetic Algorithm to Automatic Program Generation, Master of Science Thesis*. Department of Computer Science, University of Idaho, 1986.

[Hint84] G.E. Hinton, D.H. Ackley, and T.J. Sejnowski. *Boltzmann Machines: Constraint Satisfaction Networks that Learn*. Tech. Rep. CMU-CS-84-119 and Carnegie Mellon Univ. Pittsburgh, 1984.

[Holl75] J.H. Holland. *Adaptation in Natural and Artificial Systems*. University of Michigan Press (Second Edition: MIT Press, 1992), 1975.

[Howl01] Robert J. Howlett and Lakhmi C. Jain, editors. *Radial Basis Function 2*. Physica-Verlag, 2001.

[Hutt02] J. Hutton. *The Theory of the Earth*. William Creech, Edinburgh, 1802.

[Inou00] K. Inoue and K. Urahama. *Learning of view-invariant pattern recognizer with temporal context*. Pattern Recognition, 33:1665–1674, 2000.

[Kadi92] V. Kadirkamanathan and M.Ñiranjan. *A function estimation approach to sequential learning with neurals networks*. Neural Computation, 4(3):415–447, 1992.

[Kirk83] S. Kirkpatrick, C.D. Gerlatt, and M.P. Vecchi. *Optimization by Simulated Annealing*. Science 220, 671-680, 1983.

[Kirk84] S. Kirkpatrick. *Optimization by Simulated Annealing - Quantitative Studies*. J. Stat. Phys. 34, 975-986, 1984.

[Koho82] T. Kohonen. *Clustering, taxonomy, and topological maps of patterns*. In Proc. of the 6th Int. Conf. on Pattern Recognition. IEEE Computer Society Press, 1982.

[Koho90] T. Kohonen. *The self-organizing map*. Procs. IEEE, vol. 78, no.9, pp. 1464-1480, 1990.

[Koza92] J.R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, 1992.

[Kuba98] M. Kubat. *Decision Trees can initialize radial-basis-function networks*. IEEE Transactions on Neural Networks, 9:813–821, 1998.

[Lama09] J.B. Lamarck. *Philosophie zoologique, ou Exposition des considérations relatives à l'histoire naturelle des animaux*. Dentu, Paris, 1809. Accesible en http://www.iicparis.org/lamarck/new/pageframe.asp?bookid=29&pageid=3539.

[Leon98] A. Leonardis and H. Bischof. *And efficient MDL-based construction of RBF networks*. Neural Networks, 11:963–973, 1998.

[Ligh92] W. A. Light. *Some aspects of Radial Basis Function approximation*. Approximation Theory, Spline Functions and Applications, 356:163–190, 1992.

[Mats02] S. Matsui, I. Watanabe, and K. Tokoro. *A Parameter-Free Genetic Algorithm for a Fixed Channel Assignment Problem with Limited Bandwidth*. In J. J. Merelo, P. Adamidis, H. G. Beyer, J. L. Fernández-Villacañas, and H.-P. Schwefel, editors, *Parallel Problem Solving from Nature - PPSN VII, 7th International Conference, Granada, Spain, September 7-11, 2002, Proceedings*, volume 2439 of Lecture Notes in Computer Science, pages 789–799. Springer, 2002.

[Mayn75] J. Maynard-Smith. *The theory of evolution*. Penguin, 1975.

[Mere99] J.J. Merelo, J. Carpio, P.A. Castillo, V.M. Rivas, and G. Romero. *Evolving Objects*. Technical report available at http://geneura.ugr.es/~jmerelo/EOPaper/, 1999.

[Mere00a] J.J. Merelo, M. G. Arenas, J. Carpio, P.A. Castillo, V. M. Rivas, G. Romero, and M. Schoenauer. *Evolving Objects*. In M. Grana, editor, FEA (Frontiers of Evolutionary Algorithms) proceedings, 2000.

[Mere00b] J.J. Merelo, J. Carpio, P.A. Castillo, V. Rivas, and G. Romero. *Evolving Objects*. in Proc. of Int'l Workshop on Evolutionary Computation (IWEC'2000) pp. 202-208. State Key Laboratory of Software Engineering. Wuhan University, 2000.

[Mere02] J. J. Merelo. *OPEAL, una librería de algoritmos evolutivos en Perl*. In *Actas primer congreso español algoritmos evolutivos, AEB02*, pages 54–59, 2002.

[Metr58] N. Metropolis, A.W. Rosenbluth, M.N. Rosenbluth, A.H. Teller, and E. Teller. *Equations of State Calculations by Fast Computing Machines*. J. Chem. Phys. 21,1087-1092, 1958.

[Mich94] D. Michie, D. Spiegelhalter, and C. Taylor. *Machine Learning, neural and statistical classification*. Ellis Horwood, 1994. Tomado de un articulo de Schwenker, 2001.

[Mich99] Zbigniew Michalewicz. *Genetic algorithms + data structures = evolution programs*. Springer-Verlag, NewYork USA, 3 edition, 1999.

[Micr00] Microsoft. *Microsoft COM Technologies - Information and Resources for the Component Object Model-based technologies*. Website at http://www.microsoft.com/com/, 2000.

[Mill89] G.F. Miller, P.M Todd, and S.U. Hegde. *Designing neural networks using genetic algorithms*. In J.D.Schaffer, editor, Proceedings of the Third International Conference on Genetic Algorithms, pages 379-384, San Mateo, 1989, 1989.

[Mood89] J. E. Moody and C. Darken. *Fast learning in networks of locally tuned processing units*. Neural Computation, 2(1):281–294, 1989.

[Olse97] Jesper O. Olsen. *Speaker verification based on Phonetic Decision Making*. In *Proc. of EUROSPEECH'97*, Rhodes, Greece, October 1997.

[Orr93] M. J. L. Orr. *Regularised centre recruitment in Radial Basis Function Networks*. Technical report, Centre for Cognitive Science, Edimburgh University, 1993. Research paper 59.

[Orr95a] M. J. L. Orr. *Local Smoothing of Radial Basis Function Networks*. In *Proceedings of the International Symposium on Neural Networks*, Hsinchu, Taiwan, 1995.

[Orr95b] M. J. L. Orr. *Regularisation in the Selection of Radial Basis Function Centres*. Neural Computation, 7(3):606–623, 1995.

[Orr96] Mark J. L. Orr. *Introduction to Radial Basis Function Networks*. Centre for Cognitive Science, University of Edinburgh, April 1996.

[Orr98] M. J. L. Orr. *Optimising the Widths of Radial Basis Functions*. V Brazilian Congress on Neural Networks, 1998.

[Orr99] Mark J. L. Orr. *Recent Advances Radial Basis Function Networks*. Institute for Adaptive and Neural Computation, University of Edinburgh, June 1999.

[Osun97] Edgar Osuna, Robert Freund, and Federico Girosi. *An Improved Training Algorithm for Support Vector Machines*. In *Proc. of IEEE NNSP'97*, Amelia Island, FL, september 1997.

[Park85] D.B. Parker. *Learning Logic*. Tech. Report TR-47, centre for Computational Research in Economics and manegement Science, MIT, 1985.

[Plat91a] J. Platt. *A resource-allocating network for function interpolation*. Neural Computation, 3(2):213–225, 1991.

[Plat91b] John Platt. *Learning by Combining Memorization and Gradient Descent*. Advances in Neural Information Processing Systems, 3, 1991.

[Poma00] H. Pomares, I. Rojas, J. Ortegaa, J. González, and A. Prieto. *A systematic approach to Self-Generating Fuzzy Rule-Table for Function Approximation*. IEEE Trans. Syst., Man., and Cyber., 30:431–447, 2000.

[Prec94] Lutz Prechelt. *PROBEN1 — A Set of Benchmarks and Benchmarking Rules for Neural Network Training Algorithms*. Technical Report 21/94, Fakultät für Informatik, Universität Karlsruhe, D-76128 Karlsruhe, Germany, September 1994.

[Prie90] A. Prieto, P. Martin-Smith, J.J. Merelo, F.J. Pelayo, J. Ortega, F.J. Fernández, and B. Pino. *Simulation and hardware implementation of copetitive learning neural networks*. Lecture Notes in Physics vol. 368, pp. 189-204. Springer-Verlag, 1990.

[Quin92] J. Quinlan. *C4.5 programs for machine learning*. Morhan Kaufmann, 1992.

[Radc91] N.J. Radcliffe. *Equivalence class analysis of genetic algorithms*. Complex Systems 5, no.2: 183-205, 1991.

[Rech65] I. Rechenberg. *Cybernetic solution path of an experimental problem*. Ministry of Aviation, Royal Aircraft Establishment (U.K.), 1965.

[Rech73] I. Rechenberg. *Evolutionsstrategie: optimierung Technischer Systeme nach Prinzipien der Biologischen Evolution*. Frommann-Hozboog (Stuttgart), 1973.

[Reed67] J. Reed, R. Toombs, and N. A. Barricelli. *Simulation of biological evolution and machine learning*. Journal of Theoretical Biology 17: 319-342, 1967.

[Reyn96] L.M. Reyneri. *Unification of Neural and Fuzzy Computing Paradigms*. In *Proc. of AT-96, 1-st Int'l Symposium on Neuro-Fuzzy Systems*, Lausanne, Schweiz, August 1996.

[Riva01] V.M. Rivas, J.J. Merelo, and P.A. Castillo. *Evolving RBF Neural Networks*. In José MIra and Alberto Prieto, editors, *Connectionist Models of Neurons, Learning Processes, and Artificial Intelligence - IWANN'2001*, volume 2084 of Lecture Notes in Computer Science, pages 506–513. Springer, June 2001.

[Rive02] A.J. Rivera, J. Ortega, M.J. del Jesus, and J. González. *Aproximación de funciones con evolución difusa mediante cooperación y competición de RBFs*. In *Actas del I Congreso Español de Algoritmos Evolutivos y Bioinspirados, AEB'02*, pages 507–514, February 2002.

[Roy95] A. Roy, S. Govil, and R. Miranda. *An algorithm to generate radial basis function (rbf)-like nets for classification problems*. Neural Networks, 8(2):179–201, 1995.

[Rume85] D.E. Rumelhart and D. Zipser. *Feature discovery by competitive learning*. Cognitive Science, Vol. 9, pp. 75-112, 1985.

[Sarl98] W.S. Sarle. *Neural Network FAQ. Newsgroup: comp.ai.neural-nets. Part 2*, 1998. ftp://ftp.sas.com/pub/neural/FAQ.html.

[Sawa98] H. Sawai and S. Kizu. *Parameter-Free Genetic Algorithm Inspired by "Disparity Theory of Evolution"*. In A. E. Eiben, Thomas Bäck, Marc Schoenauer, and Hans-Paul Schwefel, editors, *Parallel Problem Solving from Nature - PPSN V, 5th International Conference, Amsterdam, The Netherlands, September 27-30, 1998, Proceedings*, volume 1498 of Lecture Notes in Computer Science, pages 702–711. Springer, 1998.

[Schw75] H.P. Schwefel. *Evolutionsstrategie und numerische optimierung*. Ph.D. thesis, Technische Universitat Berlin, 1975.

[Schw77] H.P. Schwefel. *Numerische optimierung von Computer-Modellen mittels der Evolutionsstrategie*. Basel: Birkhauser, 1977.

[Schw94] F. Schwenker, H. Kestler, G. Palm, and M. Höher. *Similarities of LVQ and RBF learning*. In *Proc. IEEE International Conference SMC*, pages 646–651, 1994.

[Schw95] H.P. Schwefel. *Evolution and Optimum Seeking*. Wiley, New York, 1995.

[Schw01] F. Schwenker, H. A. Kestler, and G. Palm. *Three learning phases for radial-basis-function networks*. Neural Networks, 14:439–458, 2001.

[Shet01] Alaa F. Sheta and Kenneth de Jong. *Time-series forecasting using GA-tuned radial basis functions*. Information Sciences, 133(3-4):221–228, April 2001.

[Shor96] R. Shorten and R. Murray-Smith. *Side effects of Normalising Radial Basis Function networks*. International Journal of Neural Systems, 7(2):167–179, May 1996.

[Sánc02] V.D. Sánchez. *Searching for a solution to the automatic RBF network design problem*. Neurocomputing, 42:147–170, 2002.

[Tan01] K.K. Tan and K.Z. Tang. *Taguchi-tuned radial basis function with application to high precision motion control*. Artificial Intelligence in Engieering, 15:25–36, 2001.

[Tsuj99] O. Tsujii, M. T. Freedman, and S. K. Mun. *Classification of microcalcifications in digital mammograms using trend-oriented radial basis function neural networks*. Pattern Recognition, 32:891–903, 1999.

[vdM73] C. von der Malsburg. *Self-organization of orientation sensitive cells in striate cortex*. Kybernetik, vol. 14, pp. 85-100, 1973.

[W43] McCulloch W and Pitts W. *A logical calculus of the ideas immanent in nervous activity*. Bulletin of Mathematical Biophysics (5), 115-133, 1943.

[Werb74] P.J. Werbos. *Beyond Regression: New tools for prediction and analysis in the behavioral sciences*. Ph.D. Thesis, Harvard University, 1974.

[Whit96] B. A. Whitehead and T. D. Choate. *Cooperative-Competitive Genetic Evolution of Radial Basis Function Centers and Widths for Time Series Prediction*. IEEE Transactions on Neural Networks, 7(4):869–880, July 1996.

[Widr60] B. Widrow and M.E. Hoff. *Adaptive swtching circuits*. IRE WESCON Convention Record, New York, 1960.

[Will01] A. J. Willis and L. Myers. *A cost-effective fingerprint recognition system for use wit h low-quality prints and damaged fingertips*. Pattern Recognition, 34:255–270, 2001.

[Yao99] X. Yao. *Evolving Artificial Neural Networks*. Proceedings of the IEEE, 87(9):1423-1447, 1999.

[Zhen96] G. L. Zheng and S. A. Billings. *Radial Basis Function Network Configuration Using Mutual Information and The Orthogonal Least Squares Algorithm*. Neural Networks, 9:1619–1637, 1996.