UNIVERSIDAD DE GRANADA

MODELADO DE SISTEMAS COMPLEJOS MEDIANTE ESTRUCTURAS JERÁRQUICAS DE REDES DE FUNCIONES DE BASE RADIAL

TESIS DOCTORAL

MOHAMMED M. M AWAD

GRANADA 2005

Departamento de Arquitectura y Tecnología de Computadores

UNIVERSIDAD DE GRANADA

MODELADO DE SISTEMAS COMPLEJOS MEDIANTE ESTRUCTURAS JERÁRQUICAS DE REDES DE FUNCIONES DE BASE RADIAL

Memoria presentada por

MOHAMMED M. M AWAD

Para optar al grado de

DOCTOR EN INFORMÁTICA

Fdo. MOHAMMED AWAD

Editor: Editorial de la Universidad de Granada
Autor: Mohammed M. M. Awad
D.L.: Gr. 874 - 2005
ISBN: 84-338-3400-2

D. Ignacio Rojas Ruiz, Profesor Titular de Universidad y **D. Héctor Pomares Cintas**, Profesor Titular de Universidad, del Departamento de Arquitectura y Tecnología de Computadores de la Universidad de Granada.

CERTIFICAN

Que la memoria titulada: ***"Modelado de Sistemas Complejos Mediante Estructuras Jerárquicas de Redes de Funciones de Base Radial"*** ha sido realizada por **D. Mohammed M. M. Awad** 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… de mayo de 2005

Fdo. Ignacio Rojas Ruiz
Director de la Tesis

Fdo. Héctor Pomares Cintas
Director de la Tesis

A Mis Padres la luz de mis ojos

AGRADECIMIENTOS

Quisiera expresar un gran sentimiento de gratitud a muchas personas que me ayudaron para la creación de este trabajo. En primer lugar a los directores de este trabajo ***Héctor*** *y* ***Ignacio*** *por tener la paciencia ante mis dudas de novato y por escuchar atentamente los problemas que a lo largo de esta Tesis surgieron. A* ***Héctor*** *por su generosidad al brindarme la oportunidad de recurrir a su capacidad y experiencia científica en un marco de confianza y afecto fundamentales para la concreción de este trabajo.*

A todos los miembros del departamento que me han ayudado desde el comienzo de este trabajo. Ellos han conseguido un verdadero clima de entendimiento y espíritu de colaboración que me han ayudado a remontar las dificultades que han ido surgiendo.
A ***Luis Javier*** *por su ayuda prestada. GRACIAS…*

Dedico el presente trabajo a mis padres que su enseñanza y sus buenas costumbres han creado en mi sabiduría haciendo que hoy tenga el conocimiento de lo que soy. Agradezco mis hermanos por el apoyo que siempre me han brindado con su impulso y animación.

Es difícil dar las gracias a todos aquellos que han colaborado en la realización de la tesis, pues la ayuda ha venido de varias partes. Es por esto, que damos gracias por la atención y el apoyo de nuestros amigos con especial cariño. Muy especialmente. Gracias a Lolo, Pilar, Antonio y Francisco chica.

La realización de este trabajo no habría sido posible sin la beca de la agencia española de cooperación internacional (AECI),… GRACIAS.

Nunca consideres el estudio como una obligación, sino como una oportunidad para penetrar en el bello y maravilloso mundo del saber. ***Albert Einstein***

Índice

CAPÍTULO 1

INTRODUCCIÓN

Este capítulo presenta una breve introducción al tema abordado en esta tesis incluyendo los objetivos que se propone resolver. Se presenta una introducción al problema de estimar una función desconocida a partir de un conjunto de muestras de la misma, (aproximación de funciones a partir de un conjunto finito de vectores de entrada/salida) mediante un sistema jerárquico de redes de funciones de base radial (RBFNs).

En este capítulo también se describe los motivos para realizar este trabajo, y los objetivos que se persigue con su desarrollo, y los beneficios que de él se pueden derivar. Al final se presenta un resumen de cada capítulo de esta memoria.

1.1 Introducción

La aproximación de funciones es el nombre dado a una tarea computacional muy interesante en la ciencia y en la ingeniería. Dicha tarea puede tener nombres diferentes según la implicación, como regresión no lineal, aprendizaje de modelos, etc. Estos no se refieren exactamente a la misma tarea, pero el objetivo fundamental es el mismo. La aproximación de funciones es una de las aplicaciones generales más importantes de las redes neuronales artificiales [LEN-03]. El marco general del problema de aproximación funcional es el siguiente: se supone la existencia de una relación entre varias variables de entrada y la variable de salida, siendo esta relación desconocida. Se trata de construir un aproximador (modelo de caja negra) entre estas entradas y la salida [LEN-03]. La estructura de este aproximador debe ser elegida y el aproximador debe ser optimizado para representar de manera precisa la dependencia de entrada/salida (E/S). Para realizar esta etapa, se dispone de un conjunto de muestras de entradas y salida que constituyen los datos de aprendizaje del approximator [LEN-03]. El problema de estimar una función desconocida F a partir de muestras del tipo $\{(\vec{x}_i;\ y_i);\ i = 1,2,\ldots,\ n\}$ con $y_i = F(\vec{x}_i)$, (es decir, aproximación de funciones a partir de un conjunto finito de vectores de E/S), ha sido y seguirá siendo una cuestión fundamental en una gran variedad de disciplinas científicas e industriales [HAY-99], [GON-02], [HER-03]. Una vez que dicha relación ha sido identificada, puede ser usada para la obtención de nuevas salidas, dadas otras nuevas entradas. Las entradas y salidas pueden ser, en general, variables continuas y/o discretas. En este trabajo utilizaremos variables de salida continuas considerando por tanto, problemas de regresión o aproximación de funciones, a diferencia de los problemas de clasificación donde la variable de salida es categórica. La tarea de aproximación de funciones se ha abordado de diversas formas, pasando por los modelos estadísticos, los modelos de lógica difusa y modelos neuronales [GON-01].

Una limitación fundamental en los sistemas aproximadores es que a medida que aumenta el número de variables de entrada, el número de parámetros suele incrementarse exponencialmente. Este fenómeno denominado por *Bellman* como “la maldición de la dimensionalidad” impide el uso desde el punto de vista computacional,

de la mayoría de los sistemas aproximadores convencionales y obliga a buscar soluciones más específicas.

El trabajo que se presenta en esta memoria está enfocado precisamente a la búsqueda de nuevas arquitecturas de cómputo, capaces de modelar sistemas complejos de aproximación de funciones, sin que el aumento del número de variables de entrada tenga que suponer un incremento exponencial en la complejidad del sistema. Para ello, se va a investigar el uso de una nueva topología jerárquica (Multi-RBFN) formada por una red de redes de funciones de base radial (Radial Basis Functions Neural Networks: RBFNs [CHE-96], [LEO-92], [ORR-95]), con la propiedad de que cada Sub-RBFN se encargue de un conjunto de variables de entrada y no del conjunto completo. De tal forma que se pueda reducir en gran medida el número de parámetros necesarios para definir esta estructura jerárquica Multi-RBFN. Para llevar acabo esta propuesta de sistema de estructura jerárquica, se presenta un nuevo método para seleccionar las variables de entrada más importantes (Input Variable Selection, IVS). Para tal fin, se propondrá un algoritmo que sea capaz de encontrar automáticamente la topología adecuada del sistema jerárquico Multi-RBFN propuesto y optimizar los parámetros del sistema para el modelado de un sistema de aproximación funcional a partir de un conjunto de muestras de E/S [AWA-05b].

Una de las redes neuronales artificiales más populares son las redes de perceptrón multicapa (MLP) desarrolladas por Werbos [WER-74] y Rumelhart [RUM-86]. En este trabajo utilizamos un tipo de redes neuronales denominado redes de funciones de base radial (RBFNs) [POW-87]. Las funciones de base radial (RBF) podrían ser vistas como una alternativa posible a las tentativas mencionadas de usar polinomios complejos para la aproximación de funciones a partir de un conjunto de datos de E/S $\{(\vec{x}_i;\ y_i);\ i = 1,2,\ldots,\ n\}$. Las RBFNs son redes muy simples respecto a otros modelos neuronales y tienen gran capacidad de interpolación y generalización. Su estructura es sencilla, ya que realiza una combinación lineal de varias RBF que constituyen una base para el conjunto de vectores de entrenamiento.

Consideremos que la aproximación de la función de salida objetivo *y,* por una RBFN sea notada por $\hat{F}(\vec{x})$. Esta aproximación será la suma ponderada de *m* funciones

gausianas. Para entender este concepto; considérese una función $y = F(\vec{x})$, donde $\vec{x}$ es un vector $\{x_1,...,x_d\}$ en un espacio *d*-dimensional, la salida de una RBFN se calcula por la siguiente ecuación:

$$F(\vec{x},\Phi,w) = \sum_{j=1}^{m} \varphi_j(\vec{x}) \cdot w_j \tag{1.1}$$

donde $\Phi = \{\varphi_j : j = 1,...,m\}$ y $w = \{w_j : j = 1,...,m\}$ es el conjunto de pesos asociados a cada función base radial RBF. Las RBFNs construidas por funciones gausianas $\varphi(\vec{x},\vec{c},r)$ producen una salida más suave y mejoran la capacidad de interpolación del sistema. Este tipo se define como en la siguiente expresión:

$$\varphi(\vec{x},C,R) = \sum_{j=1}^{m} \exp\left(-\frac{\left\|\vec{x}-\vec{c}_j\right\|^2}{r_j}\right) \tag{1.2}$$

donde $C = \{\vec{c}_1,...,\vec{c}_m\}$ es el conjunto de centros de las funciones gausianas φ, y $R = \{r_1,...,r_m\}$ sus radios.

La Fig.1.1 presenta una aproximación funcional por funciones radiales del tipo gausiano para aproximar una función $y = F(\vec{x})$ por otra función $\hat{F}(\vec{x})$ en una dimensión con *m* funciones de base radial, y con diferentes valores de los radios *r*.

Las RBFNs tienen la ventaja de construir aproximaciones locales, porque cada neurona se especializa en una determinada región del espacio de entrada y construyen una aproximación local en dicha región. Por tanto, la relación entre los datos de entrada y salida es una suma de funciones no lineales y locales para diferentes zonas del espacio de los datos de entrada. Esto produce aprendizaje más rápido, y un cambio del peso de una función base solo afecta la neurona oculta asociada a dicho peso. Estas características diferencian las RBFNs de las redes MLP. Además las RBFNs guardan la propiedad principal de la aproximación universal de funciones [POG-87].

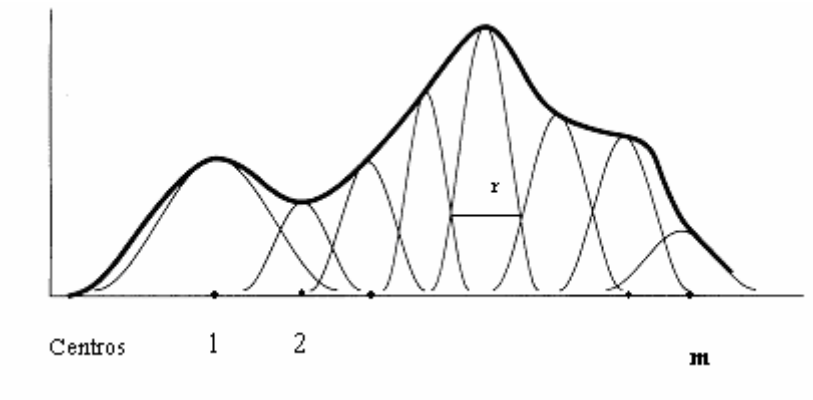


Fig. 1.1 Aproximación por RBFs en una dimensión

La exactitud de aproximación es medida por una función de coste o una función del error entre la salida esperada de la red de funciones de base radial $F(\vec{x})$ y la salida real y. La función del error o criterio de aproximación $\mathrm{E}\left(F(\vec{x}); y\right)$ depende del conjunto de ejemplos $(\vec{x}_i, y_i)$ sobre la opción del criterio de aproximación, y sobre la clase de funciones en las cuales se aproxima la función $\hat{F}(\vec{x})$. Sin embargo, las medidas de error son útiles sólo por motivos teóricos, porque la función objetivo es por lo general dada sólo en la forma de un conjunto. En esta memoria se utiliza como medida del error el error cuadrático medio normalizado *(Normalized Root Mean Squared Error*: NRMSE*)*. Este tipo de medida del error permite comprobar los resultados experimentales con otros resultados obtenidos por otras aproximaciones para en el mismo problema existentes en la literatura. Este error se define por la siguiente ecuación:

$$NRMSE = \sqrt{\frac{\sum_{i=1}^{n}(y_i - F(\vec{x}_i))^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2}} \tag{1.3}$$

donde $\bar{y}$ la media de las salidas de la función objetivo para todos los vectores de los datos de entrada.

La estructura Multi-RBFN propuesta trata de reducir el número de parámetros utilizados en un sistema de aproximación funcional. El número total de parámetros en un sistema típico de RBFNs con respeto a la ecuación (1.1) se calcula por la siguiente ecuación:

$$m \cdot (d + 2) \tag{1.4}$$

donde d es el número de dimensiones del espacio de entradas, y m el número de funciones radiales (RBF). La estructura jerárquica Multi-RBFN propuesta reduce el número de parámetros utilizados y mejora el error de aproximación, lo cual repercute en la complejidad del sistema aproximador e incrementa la eficiencia del proceso de aproximación de funciones a partir de un conjunto de ejemplos de E/S [AWA-05b].

1.2 Motivos para realizar este trabajo

Para objetivos de aproximación de funciones en problemas con número alto de variables de entrada, el número de parámetros suele incrementarse exponencialmente. Este fenómeno, denominado por *Bellman* como "la maldición de la dimensionalidad" impide el uso, desde el punto de vista computacional de la mayoría de los sistemas aproximadores convencionales y obliga a buscar soluciones más específicas. Por otra parte, una red neuronal con muchos parámetros produce una red con complejidad computacional alta, una consecuencia directa de esto; la producción de una red abultada con un número grande de unidades ocultas obstaculiza la convergencia del proceso de aprendizaje y aumentará el tiempo de ejecución.

La maldición de las dimensionalidad [BEL-61], se refiere al crecimiento exponencial del hiper-volumen como una función de dimensionalidad. En el campo de redes neuronales en general [BEN-00], la maldición de la dimensionalidad implica y supone unos problemas relacionados: una red neuronal tiene que cubrir o representar cada parte de su espacio de entrada a fin de saber cómo aquella parte del espacio debería ser trazada sobre un mapa. La cobertura del espacio de entrada consume recursos y en la mayor parte de los casos, la cantidad de recursos necesarios es proporcional al hiper-volumen del espacio de entrada. La formulación exacta depende del tipo de la red utilizada que debería estar probablemente basada en los conceptos de teoría de información y geometría diferencial. La maldición de la dimensionalidad provoca redes con muchas entradas irrelevantes y se comportan relativamente mal. Es decir, si la dimensión del espacio de entrada es alta; la red usa casi todos sus recursos para representar partes irrelevantes del espacio de los datos de entrada. Un algoritmo de aprendizaje de redes neuronales que sea capaz de concentrarse en partes importantes del

espacio de entrada, cuando más alta es la dimensionalidad del espacio de entrada, más datos pueden ser necesarios para averiguar lo que es importante y lo que no es. La selección de variables de entrada (IVS) y el escalamiento de las entradas, fundamentalmente afectan la utilidad del problema, así como la selección del modelo de la red neuronal utilizada [JAI-82], [YU-03].

Existen en la bibliografía un gran número de topologías de redes neuronales y diferentes clases de algoritmos para el aprendizaje y ajuste de los parámetros e interconexiones entre neuronas. Las tres principales formas básicas de aprendizaje son:

a) Aprendizaje no supervisado: consiste en la construcción de modelos neuronales que capturan las regularidades existentes en los vectores de entrada con los cuales es entrenada la red, sin tener que recibir ninguna información adicional.

b) Aprendizaje supervisado: a diferencia del anterior, en este tipo de aprendizaje se requiere especificar cuál es la respuesta deseada que la red debe dar ante la presentación de un estímulo de entrada. En esta memoria se utiliza este tipo de aprendizaje para la aproximación de funciones a partir de un conjunto de datos de E/S.

c) Aprendizaje con refuerzo: en vez de necesitar la salida deseada para cada uno de los patrones de entrada presentados a la red, se precisa una medida, un simple escalar que permita evaluar la salida de la red [BER-92], [LIN-93], [LIN-99]. En el aprendizaje con refuerzo no existe un supervisor que juzgue críticamente la acción de salida en cada instante de tiempo. El aprendizaje del sistema es conducido mediante la asignación de un índice o valor numérico a las diferentes evoluciones del mismo.

Básicamente todos los algoritmos de entrenamiento para RBFNs tratan de encontrar modelos que minimizan dos objetivos: el error de aproximación y la complejidad estructural de la red. Uno de los métodos consiste en la aplicación de métodos de agrupamiento o *clustering* [BEZ-81] para la determinación y optimización de los parámetros que definen los nodos neuronales. Sin embargo, algunos algoritmos de *clustering* tradicionales diseñados para problemas de clasificación, solamente tienen en

cuenta la disposición de los ejemplos de entrenamiento en el espacio de entrada, ignorando por completo la salida de la función objetivo. Por tanto, si se utilizan para inicializar un sistema como el que se propone en esta memoria con salidas en rango continuo, es necesario realizar un análisis más profundo. Por ello en esta memoria se propone un algoritmo de *clustering* para inicializar los centros de las RBFNs que depende de los datos de entrada y su salida objetivo, de forma que se colocarán más prototipos en las zonas del espacio de entrada en las que haya un mayor error de aproximación provocado por cada cluster, dependiendo esto de la salida objetivo, tendiendo a equidistribuir el error de aproximación. Para completar la inicialización con el cálculo del resto de los parámetros (radios r, pesos w, el número adecuado de funciones base) se utilizan algoritmos tradicionales. Otros métodos han sido utilizados para inicializar los valores de los parámetros y optimizar la estructura de las RBFNs como los algoritmos genéticos [HOL-75], [GOL-89], [MIC-99].

El sistema de cómputo Multi-RBFN que se presenta en esta memoria está formado por redes RBFNs con la propiedad de que cada Sub-RBFN se encarga de un conjunto de variables de entrada y no del conjunto completo, de tal forma que se pueda reducir en gran medida el número de parámetros necesarios para lograr un grado de aproximación determinado. Para seleccionar estos conjuntos que van a cada Sub-RBFNs se analiza y aplica un nuevo método para la selección de variables de entrada más importantes y seleccionar cuáles de estas variables deben ir solas o juntas en cada Sub-RBFN. Para tal fin, se propondrá un algoritmo que sea capaz de encontrar automáticamente la topología adecuada del sistema de la estructura jerarquía Multi-RBFN según la estructura de la función o el problema que trata de aproximar, y los valores adecuados para los parámetros de este sistema para el modelado de un sistema a partir de un conjunto de muestras de E/S.

Dicho esto, los objetivos que persiguen esta memoria son los siguientes:

- Desarrollar una nueva arquitectura de cómputo basada en una estructura jerárquica de redes de funciones de base radial (RBFNs) denominada Multi-RBFN.

- Estudio y análisis de diversos algoritmos para la selección de variables de entrada y proponer un nuevo método para seleccionar las variables de entrada más importantes.

- Determinación automática de la topología del sistema Multi-RBFN propuesto a partir de datos de E/S.

- Diseño de un algoritmo para la optimización de todos los parámetros del sistema jerárquico creado (centros $\vec{c}^S$, radios r^S, pesos w^S y el número adecuado de funciones radiales en cada Sub-RBFN), donde S es el número de Sub-RBFNs.

- Evaluación del rendimiento del algoritmo y comparaciones con otros autores.

Estos objetivos se explican con detalles a continuación:

- **Desarrollo de una nueva arquitectura de cómputo basada en una estructura jerárquica de redes de funciones de base radial (Multi-RBFN)**

  En esta parte del trabajo se han considerado diversas posibilidades de arquitecturas jerárquicas basadas en RBFNs, y al final se ha optado por una arquitectura denominada Multi-RBFNs. Esta arquitectura consiste de Sub-RBFNs, cada una de ellas se ocupará de un subconjunto de las variables de entrada (y no de todas), de tal forma que se minimice el número de parámetros necesarios. Por otro lado, estas Sub-RBFNs se unirán de forma lineal para dar la salida final. Esta linealidad en el cálculo de la salida final nos permitirá poder utilizar algoritmos matemáticos exactos para la resolución de sistemas de ecuaciones lineales como es el SVD *(Singular Value Decomposition*) [GON-01] y el OLS *(Orthogonal Least Squares)* [CHE-91a].

- **Estudio y análisis de diversos algoritmos para la selección de variables de entrada y proponer un nuevo método para seleccionarlas, y**

- **Determinación automática de la topología Multi-RBFN a partir de datos de E/S.**

Lo que se pretende en estos dos apartados (que están muy interrelacionados) es encontrar una técnica fiable y robusta capaz de seleccionar aquellas variables de entrada que influyan más significativamente en las salidas. Para ello, proponemos un nuevo método que trata de seleccionar estas variables más importantes del conjunto original de los datos de entrada. Este método de IVS depende de forma principal del cálculo de la distancia entre los máximos y mínimos puntos de la salida con respecto a cada dimensión de los datos de la entrada asociada con la salida objetivo *y*. La idea consiste en dividir los datos de la entrada en cada dimensión respecto a la salida en partes, el número de estas partes depende del número de puntos de los datos de entrada. En cada parte se calcula la distancia entre el punto máximo y el punto mínimo de los datos de la salida que pertenecen a cada parte, después se calcula la dispersión y la media de la distancia de todas las partes. Cuando el valor de la distancia media es pequeño entonces la variable que tratamos es más influyente en la salida y debe ser seleccionada como entrada para el sistema Multi-RBFN, pero si el valor de la distancia media es grande y la dispersión de las distancias en cada tramo es pequeña, entonces la variable es ruidosa y debe ser eliminada. Para la selección del número de Sub-RBFN y cuáles de las variables seleccionadas deben estar en cada Sub-RBFN, se utiliza un método que depende de la dispersión de las distancias. Este método calcula la varianza para cada variable dependiendo de los valores máximos y mínimos de los datos de la salida del sistema que pertenecen a cada partición de los datos para cada variable. Las variables que tiene una varianza menos de un valor umbral se seleccionan como variables que deben ir solas en nuestro sistema jerárquico, cada una a una Sub-RBFN. Después de esta fase, se analizan todos los posibles conjuntos de dos variables (a partir de las variables restantes) con la salida objetivo del sistema, dividendo el espacio de estos dos variables en partes cuadradas. De la misma manera en la primera fase se calcula la varianza para cada posible par de variables dependiendo de los valores máximos y mínimos de los datos de la salida del sistema que pertenecen a cada partición. Los conjuntos de variables que tienen un valor de varianza menor de un valor umbral se seleccionan como variables que van juntas a un Sub-RBFNs. De la misma manera, se analiza los posibles conjuntos de tres, cuatro variables, etc.

- **Diseño de un algoritmo para la optimización de todos los parámetros del sistema jerárquico creado (centros $\vec{c}^S$, radios r^S, pesos w^S y el número adecuado de RBF en cada Sub-RBFN).**

  Para optimizar los parámetros del sistema Multi-RBFN, una vez fijada su topología general, utilizaremos:

  1. Diversos algoritmos de *clustering* para encontrar un buen punto inicial en el espacio de los parámetros que definen las Sub-RBFN. En concreto, para encontrar los valores iniciales de los centros $\vec{c}^S$ de cada Sub-RBFN (que es una RBFN) y los radios r^S. Para ello, contamos con algoritmos como los k-medias [DUD-73], k-medias difuso [BEZ-81], ELBG [RUS-99], *clustering* para aproximación funcional CFA [GON-02], el método de los vecinos más cercanos KNN [MOO-89], etc. En esta memoria se propone un algoritmo nuevo de *clustering* especialmente diseñado para los problemas de aproximación de funciones. Este algoritmo derivado en los algoritmos CFA y ELBG trata de resolver el problema de inicialización de forma más eficaz que otros algoritmos. En este trabajo se utiliza este algoritmo propuesto de *clustering* para inicializar los valores de los centros en el sistema propuesto Multi-RBFN.

  2. Algoritmos de resolución de sistemas de ecuaciones lineales para calcular el peso w^S de salida de cada sub-RBFN de forma óptima. Principalmente buscaremos algoritmos rápidos como es el método de *Cholesky*, aunque teniendo en cuenta la robustez en el cálculo, también estudiaremos el uso de otros algoritmos como el SVD *(Singular Value Decomposition)* [GON-01], y el OLS *(Orthogonal Least Squares)* [CHE-91a]. A lo largo de esta memoria utilizamos el algoritmo SVD *(Singular Value Decomposition)* para calcular los pesos de forma óptima. Este método facilita una solución para cualquier sistema de ecuaciones, con la que se obtiene una reducción del error en la salida de la red. Además muestra que, si se elimina de la red alguna función base radial cuyo valor singular asociado tuviera una magnitud pequeña el error de aproximación no se vería prácticamente afectado.

3. Algoritmos de minimización local (búsqueda del mínimo local más cercano a la configuración inicial). Para ello, estudiaremos los distintos algoritmos existentes y utilizaremos el que ofrezca mejores prestaciones de robustez, en primer lugar, y de velocidad, en segundo lugar. En concreto, analizaremos el algoritmo de descenso en gradiente [DEN-83], descenso en gradiente con factor de aprendizaje adaptativo, gradiente conjugado [POW-77], *Levenberg-Marquardt* [MAR-63], [KEC-01], *Newton-Raphson* [KEC-01], etc. Al largo de esta memoria se utiliza el método de *Levenberg-Marquardt* para la obtención de los mínimos locales. Este método incorpora información sobre la segunda derivada de error para acelerar la búsqueda cuando la configuración de la red se encuentra cerca del mínimo.

- **Evaluación del rendimiento del algoritmo y comparaciones con otros autores.**

  Finalmente, para comprobar el buen funcionamiento del sistema utilizado, usaremos diversos ejemplos, y se ejecutarán los algoritmos propuestos para proporcionar datos sobre el tiempo de ejecución, la topología encontrada, el valor de los parámetros, la tasa de error, la complejidad del algoritmo y se compararán estos resultados con los propuestos por otros autores en la bibliografía.

  El aporte de esta memoria es la obtención de un método automático para el modelado de sistemas complejos a partir de un conjunto de muestras de E/S. Este método se puede utilizar en diferentes campos concretos, como el campo de control, en donde se necesitan sistemas que modelen el comportamiento de una planta para poder controlarla posteriormente; o el campo de la predicción de series temporales, que se necesitan sistemas que modelen el comportamiento de la serie para, una vez obtenido el modelo, poder predecir valores futuros.

1.3 Resumen de los capítulos de la presente memoria

- **Capítulo 1:** realiza breve explicación general del trabajo realizado en esta memoria, comienza con introducción general del problema de aproximación

funcional, explicando la necesidad de esté. Además se describe el tipo de computación fundamental utilizada en este trabajo que es una red neuronal del de funciones de base radial RBFNs y el método propuesto de estructura jerárquica Multi-RBFN utilizada. También se habla de la necesidad de un algoritmo de *clustering* especialmente diseñado para problemas de aproximación funcional. Por otro lado se explica los motivos para realizar este trabajo, empezando con el afecto de la maldición de las dimensionalidad al problema de aproximación funcional y el efecto de la selección de las variables de entrada en resolver problemas de este tipo. Al final se explican con detalle los objetivos y el desarrollo que persigue esta memoria con un resumen de cada capítulo.

- **Capítulo 2:** comienza a describir las redes de funciones de base radial que son el fundamental elemento computacional de nuestro trabajo. Las RBFNs se engloban dentro de las redes de aprendizaje supervisado; son redes multicapa con conexiones hacia delante. Una RBFN es una red totalmente unida de tres capas, una capa de entrada que no realiza ningún cálculo, una única capa oculta, cada neurona de esta capa posee un carácter local, y una capa de salida en que neuronas realizan una combinación lineal de las activaciones de las neuronas de la capa oculta. Se describe también otro tipo de redes neuronales utilizado en este trabajo para comprobar el funcionamiento del algoritmo propuesto de *clustering*. Este tipo denominado redes neuronales *wavelet* (WNNs) tiene ventajas sobre las RBFNs en conseguir la convergencia más rápida y son también capaces de tratar con el problema de la maldición de la dimensionalidad. En este capítulo también se hará un repaso de los algoritmos de optimización tradicionales, la teoría de optimización y las diferentes técnicas de optimización usadas para encontrar los valores de un conjunto de parámetros que minimizan o maximizan la función del error o función de coste del interés.

  Al final del capítulo, se analizan algoritmos de selección de variables de entrada (IVS). Este proceso es realmente importante para reducir la dimensionalidad del espacio de las variables de la entrada y quitar aquellas variables redundantes, irrelevantes o ruidosas. Los efectos inmediatos que produce esta tarea consisten

en aumentar la velocidad de la ejecución de los algoritmos de aprendizaje, mejorar la calidad de los datos y mejorar la exactitud del modelo.

- **Capítulo 3:** presenta un nuevo algoritmo de *clustering* para aproximación funcional derivado *de* los algoritmos de *clustering* para aproximación funcional CFA y "enhanced LBG". Se comienza explicando los algoritmos de *clustering* tradicionales (no supervisados y supervisados). La inicialización de los centros de las RBFNs es una tarea importante para problemas de aproximación funcional. Cada centro tiene que estar situado en una zona del espacio de los datos de la entrada de forma que pueda cubrir los datos de esta zona. El algoritmo propuesto en este capítulo trata de aumentar el número de los clusters en zonas de espacio de entrada donde el error de aproximación es peor, intentado igualarse éste entre los diversos clusters. Para ello, se calcula el error que provoca cada cluster utilizando para ello el error de cada conjunto de datos de entrada que pertenecen a este cluster, intentando igualar el valor de este error en cada cluster. La forma de igualar el error en todos los clusters depende de un proceso de migración que trata de migrar cluster con valor de error pequeño a zonas de clusters con valor de error grande.

- **Capítulo 4:** presenta los resultados experimentales obtenidos por el algoritmo de *clustering* propuesto en el capítulo 3. Estos resultados obtenidos de aplicar el algoritmo propuesto a funciones de una, dos o varias dimensiones, comparándolos con los resultados del algoritmo *clustering* para aproximación de funciones CFA [GON-01], que ha demostrado ser hasta ahora el mejor existente en la bibliografía. Para averiguar la exactitud de algoritmo propuesto se aplica a cuatro tipos de redes neuronales; redes de funciones de base radial normalizada (NRBFNs), redes de funciones de base radial (RBFNs), redes neuronales *wavelet* normalizada (NWNNs) y Redes neuronales *wavelet* (WNNs).

- **Capítulo 5:** describe el algoritmo propuesto para la obtención de una estructura jerárquica de redes neuronales de funciones de base radial RBFNs denominada Multi-RBFNs a partir de datos de E/S. Este modelo divide al principio el problema de aproximación de funciones en problemas más pequeños, siendo

cada uno de estos una red RBFN completa. La estructura final es una serie jerárquica de RBFNs conectadas en paralelo con una única salida para todas las Sub-RBFN. El modelo propuesto tiene algunos componentes básicos: la utilización de un método propuesto para la selección de variables de entrada más importantes. Este método también decide cuáles de estas variables que han sido seleccionadas deben ir solas o juntas a una Sub-RBFN. De esta forma se puede decidir la estructura del sistema Multi-RBFNs. Para la determinación del número de funciones radiales RBF utilizado en cada Sub-RBFN, se utiliza un algoritmo incremental que empieza con un número de RBF igual a uno en cada Sub-RBFNs y añade RBF en una de las Sub-RBFN hasta que llega al número adecuado de funciones radiales suficiente para la estructura Multi-RBFN.

Para el proceso de optimización de los parámetros del sistema propuesto, se utiliza algoritmos de *clustering* para inicializar los centros $\vec{c}$ de las funciones radiales RBFs, algoritmos para inicializar los radios r de las RBFs y algoritmos para resolver ecuaciones lineales que calculan los valores exactos de los pesos w de las RBFs. La salida de cada Sub-RBFNs se calcula utilizando funciones lineales y la salida total de Multi-RBFNs que es la suma lineal de todas las salidas de las Sub-RBFNs se calcula también utilizando funciones lineales.

- **Capítulo 6:** presenta los resultados experimentales obtenidos por el algoritmo de la estructura jerárquica Multi-RBFN propuesto en el quinto capítulo. Estos resultados son de dos tipos: resultados de la selección de la estructura jerárquica Multi-RBFNs adecuada con el número de variables de entrada para cada Sub-RBFNs, y la optimización de los parámetros del sistema para la minimización del error. Estos resultados experimentales han sido obtenidos en aplicar el algoritmo propuesto en problemas de aproximación de funciones en dos o varios dimensiones. En este capítulo también se comprueba la capacidad del algoritmo presentado para modelar sistemas complejos de aproximación funcional de unos conjuntos de entrenamiento ruidosos.

- **Capítulo 7:** Este capítulo presenta las conclusiones del trabajo realizado y las líneas futuras de trabajo que quedan abiertas.

CAPÍTULO 2

FUNDAMENTOS

Este capítulo comienza con la descripción de las redes de funciones de base radial (RBFNs) que son el fundamental elemento computacional en nuestro trabajo. Las RBFNs se engloban dentro de las redes de aprendizaje supervisado, son redes multicapa con conexiones hacia adelante. Se describe también otro tipo de redes neuronales similar a las RBFNs, son las redes neuronales *wavelet* (WNNs), que tiene la ventaja sobre las RBFNs en conseguir la convergencia más rápida, y son también capaces de tratar con la maldición de la dimensionalidad.

Se hará un repaso de los algoritmos de optimización tradicionales. La teoría de optimización y las técnicas diferentes de optimización se utilizan para encontrar los valores de un conjunto de parámetros que minimizan o maximizan una función del error o una función de coste del interés. También se analizan algoritmos de selección de variables de entrada (IVS). Las ventajas de la selección de variables consisten en reducir la dimensionalidad del espacio de variables de la entrada y quitar los datos redundantes, irrelevantes o ruidosos.

2.1 Redes de funciones de base radial (RBFNs)

2.1.1 Introducción

Las funciones de base radial (RBF) fueron primero introducidas por Powell [POW-87] [HAY-99]. La utilización de funciones de base radiales (RBFs) como funciones de activación para redes neuronales fue realizada por primera vez por Broomhead y Lowe en 1988 [BRO-88]. La mayor contribución a la teoría, diseño y aplicaciones de las RBFNs se debe a Moody y Darken [MOO-89], Poggio y Girossi [POG-87], y Renals [REN-88]. Uno de los objetivos principales de los autores era construir una red neuronal que requiriese un menor tiempo de aprendizaje que el que necesitan otras redes neuronales como los perceptrones multicapa (MLP) [WER-74]. Desde entonces se ha prestado una especial atención dado que tienen las características de ser fácilmente implementables [RIV-03] (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 [LIG-92], como lo puedan ser MLPs, en el sentido de que son capaces de aproximar cualquier función continua sobre un conjunto de datos de E/S.

Las RBFNs son combinaciones lineales de múltiples funciones locales y no lineales. De este modo, se suele decir que las RBFNs aproximan relaciones complejas mediante una colección de aproximaciones locales menos complejas, dividiendo el problema en varios sub-problemas menos complejos. Esto hace que las aproximaciones construidas por las RBFNs sean de naturaleza diferente a las aproximaciones globales y basadas en hiper-planos que construye una MLP. Una demostración formal de este resultado fue realizado por Park y Sandberg [PAR-91]. De este modo, disponemos de una red neuronal que puede ser apropiada para aplicaciones en tiempo real. Esto se consigue incorporando funciones de activación locales en las neuronas ocultas de la red, lo cual permitirá que sólo unas pocas neuronas ocultas tuvieran que ser procesadas para nuevos patrones de entrada. El carácter local de las RBFNs viene por el uso de las funciones de base radial RBF, generalmente con función de activación del tipo gausiano. Sin embargo, su forma de aprendizaje es completamente diferente.

Las funciones RBF eran inicialmente utilizadas para resolver problemas de interpolación de multi-variedad y análisis numérico, de tal forma, que su perspectiva es semejante en aplicaciones de redes neuronales. El ajuste de una RBFN es equivalente al empleo de una superficie multi-dimensional para interpolar los datos de prueba. En una red neuronal, las unidades ocultas forman un conjunto de las funciones que componen una base aleatoria para el modelo de los datos de entrada (vectores), estas funciones son funciones de base radial [HAY-99].

El diseño de una RBFNs en su forma más básica consta de tres capas separadas. La primera capa de entrada es el conjunto de nodos de la fuente (unidades sensoriales), la segunda capa es una capa oculta de dimensión alta. La capa de salida es la encargada de dar la respuesta de la red al modelo de activación aplicada a la capa de entrada. La transformación del espacio de entrada al espacio de unidad oculta, de esta forma, no es lineal. Por otro lado, la transformación del espacio oculto al espacio de la salida es lineal [HAY-99].

Las características principales de las redes de funciones de base radial RBFNS son:

- Son redes de capa hacia adelante.
- Los nodos ocultos implementan un conjunto de funciones de base radial (p.ej. funciones gausianas).
- Los nodos de salida implementan funciones de suma lineales.
- El entrenamiento de la red se divide en dos etapas:
    - se determinan los pesos de la entrada a la capa oculta.
    - se calculan los pesos de la capa oculta a la capa de salida.
- El entrenamiento / aprendizaje es muy rápido.
- Las RBFNs tienen muy buenas características en la interpolación y aproximación de funciones.

Las RBFNs han sido aplicadas en gran campo de problemas, aunque no han sido aplicadas de forma extendida como las redes MLP. Sin embargo, se han utilizado en diferentes campos, como procesamiento de imágenes [SAH-90], análisis de series

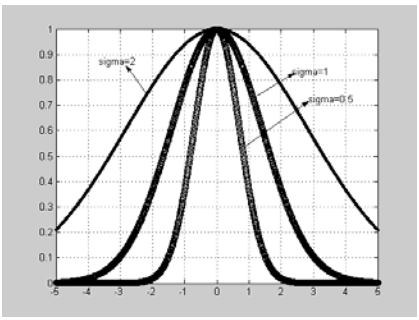
temporales [MOO-89], diagnósticos médicos [LOW-90], reconocimiento automático de habla [NIR-90], etc.

2.1.2 Funciones radiales

Las funciones radiales [ORR-95] son una clase especial de funciones que constituyen la base para una transformación no lineal que introducen unos datos vectoriales. Así, su respuesta decrece o incrementa monótonamente con la distancia de un punto central. Una función radial típica es la función gausiana (que es la más utilizada y común en las RBFNs como función de activación). De este modo, una función radial $\varphi(\vec{x},\vec{c},r)$ se determina por los siguientes parámetros:

- **Centro ($\vec{c}$)** de las funciones radiales, con dimensión igual al espacio de entradas.
- **Radio (*r*)** de las funciones radiales que permita escalar las distancias de los puntos de entrada con respecto al centro, que pueden ser el mismo para todas las dimensiones o pueden ser diferentes.
- **Peso** (*w*) de las funciones radiales.

La salida *y* de la función radial φ en el caso de una entrada escalar se obtiene mediante la ecuación (1.2). La Figura 2.1.*a* ilustra una función de base radial del tipo gausiano con centro $c=0$ y radio $r=1$. Podemos ver la forma de la función con valores de radios diferentes. Para el caso general de una señal del vector de entrada de dos dimensiones, la función radial de salida se ilustra en Fig. 2.1.*b***.**



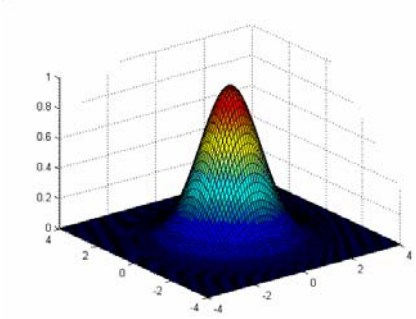


Fig. 2.1 *a*) RBFs con valor de σ diferente *b*) RBF 2D del tipo gausiano

2.1.3 Arquitectura de una RBFN

Las funciones radiales son simplemente una clase de funciones que, en principio, podrían ser empleadas en cualquier tipo de modelo lineal o no lineal y en cualquier tipo de red de una sola capa o de múltiples capas. Sin embargo, desde Broomhead y Lowe [BRO-88], las RBFNs tradicionalmente han sido asociadas con funciones radiales en una red de una sola capa oculta como muestra la Figura 2.2. Las RBFNs simples tienen la arquitectura de un perceptrón de tres capas donde la capa oculta contiene las neuronas con comportamiento no lineal, como podemos observar en Fig.2.2. Las funciones de base radial corresponderán a cada una de las neuronas de la capa oculta.

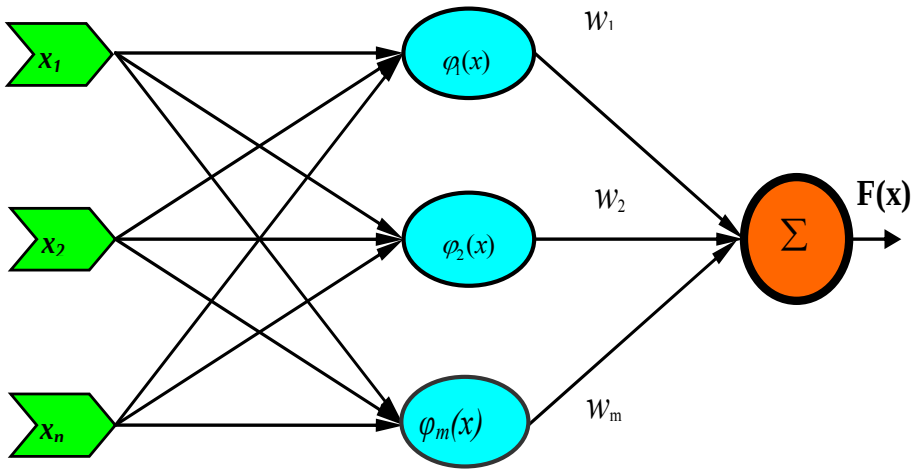


Fig. 2.2 Arquitectura de una red de RBF típica

Una RBFN tiene:

- *La capa de entrada*: un conjunto de neuronas que reciben las señales del exterior, transmitiéndolas a la siguiente capa sin realizar ningún procesado sobre dichas señales.

- *La capa ocultada*: las neuronas de esta capa reciben las señales de la capa de entrada y realizan una transformación local y no lineal sobre dichas señales. Esta capa es la única que incluye componentes no lineales.

- *La capa de salida*: mantiene la respuesta de la red, y realiza una combinación lineal de las activaciones de las neuronas ocultas.

Las RBFNs son redes con conexiones hacia adelante, como muestra Fig.2.2. Éstas se caracterizan así porque las conexiones de la capa de entrada a la capa oculta no llevan asociado ningún peso, mientras que las conexiones de la capa oculta a la capa de salida sí llevan asociado un número real o peso de la conexión, y no existen realimentaciones de ninguna capa a una capa anterior.

2.1.4 Funcionamiento de las RBFNs

Para obtener funcionamiento adecuado de las RBFNs, se necesita una consideración muy cuidadosa en el diseño. La selección de varios parámetros como el número de neuronas en la capa oculta *m* y los valores iniciales de centros $\vec{c}$ y radios *r* de la función base radial debe ser considerada con cuidado, ya que afectan críticamente su funcionamiento, como se muestra Fig.2.3.

Los campos receptivos se centran en las áreas del espacio de entrada y sirven para agrupar los vectores de la entrada similares. Si un vector de entrada $\vec{x}_i$ estuviese cercano al centro de un campo receptivo, entonces aquel nodo oculto será activado, de esta forma, la activación de una neurona oculta depende de la distancia entre cada punto de la entrada y el centro de la función radial $\vec{c}_j$.

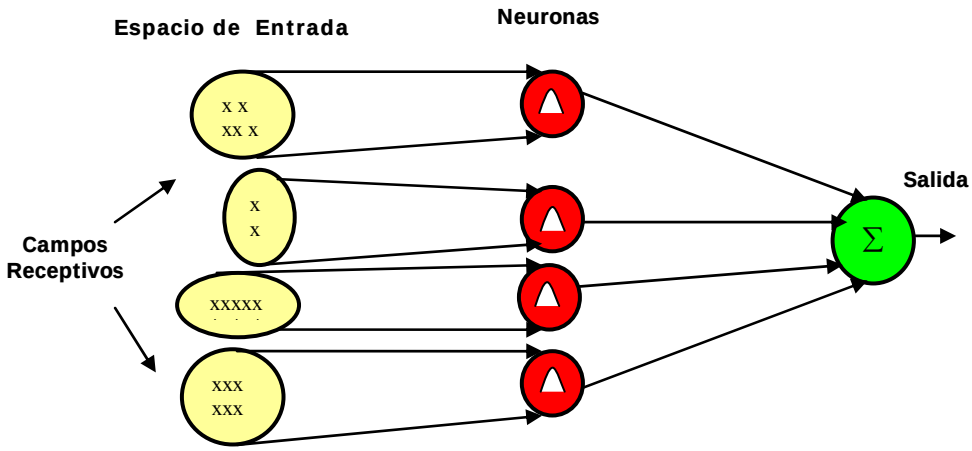


Fig. 2.3 Funcionamiento de una red de RBF.

Cada salida de las unidades ocultas se obtiene calculando la cercanía de la entrada $\vec{x}_i$ a un parámetro dimensional del vector $\vec{c}_j$ asociado con la *j-ésima* unidad oculta. La capa de salida es una capa de neuronas que realiza una transformación lineal de las salidas de los nodos ocultos. Dada una RBFN con *d* neuronas en la capa de entrada, y *m* neuronas en la capa oculta, se produce la activación de la salida denotada como $F(\vec{x})$ que se obtiene utilizando la ecuación (1.1).

La matriz de activación de las neuronas ocultas φ_j, también conocida como funciones de base radial RBFs, determinan las activaciones de las neuronas ocultas de la red en función del vector de la entrada de la red $\vec{x}$ y viene dado por la siguiente ecuación:

$$\varphi_j(\vec{x},\vec{c},r) = \varphi\left(-r_j \left\|\vec{x}-\vec{c}_j\right\|\right) \; para \quad j = 1,2,..., m \tag{2.1}$$

donde φ es la función radial, $\vec{c}_j = (\vec{c}_{j1},...,\vec{c}_{jn})$ son vectores que presentan los centros de la función radial, r_j son números reales que presentan los radios de la función radial, y $\| \ \|$ es la distancia euclídea del vector de entrada $\vec{x}$ al centro $\vec{c}$ definida como:

$$\left\| \vec{x}_i - \vec{c}_j \right\| = \sqrt{\sum_{i=1}^{n} \left(x_i - \vec{c}_{ji} \right)^2} \tag{2.2}$$

La función de base radial φ puede adoptar diferentes formas, entre ellas:

- **Función Gausiana**: esta es la función de base más común y más usada en el campo de las RBFNs [HAR-90].

$$\varphi_j(\vec{x},\vec{c},r) = e^{-\left(\frac{\left\|\vec{x}-\vec{c}_j\right\|}{r_j}\right)^2}$$

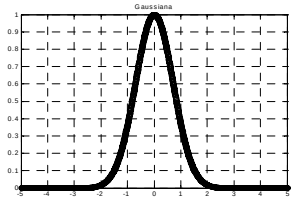


Fig. 2.4 Función gausiana

- **Función multicuadrática:**

$$\varphi_j(\vec{x},\vec{c},r) = \sqrt{r\cdot\left\|\vec{x}-\vec{c}\right\|^2 + 1}$$

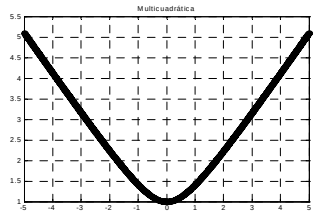


Fig. 2.5 Función multicuadrática

- **Función plato delgado spline:**

$$\varphi_j(\vec{x},\vec{c},r) = r\cdot(\vec{x}-\vec{c})^2 \ \log[r\cdot(\vec{x}-\vec{c})^2]$$

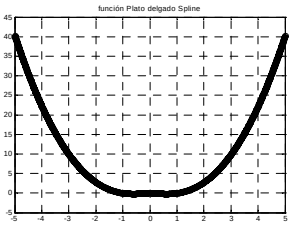


Fig. 2.6 Función plato delgado spline

Existen otras funciones para la activación de las neuronas ocultas en las RBFNs como; función lineal, función cúbica, y función inversa multicuadrática, etc.

La función más utilizada para activar las neuronas ocultas es la función gausiana [MOO-89]. La salida de una red de funciones de base radial (RBFNs) con función de activación del tipo gausiana se obtiene por la siguiente expresión:

$$F(\vec{x}_i,\vec{c}_j,r) \ = \sum_{j=1}^{m} w_j \cdot e^{\left(-r_j\cdot\sum_{i=1}^{n}\left(\vec{x}_i-\vec{c}_{ij}\right)^2\right)} \tag{2.3}$$

Las funciones de base radial φ se caracterizan porque poseen un nivel máximo de activación para valores de entrada cercanos a cero. Dicho nivel decrece a medida que la variable de entrada se aleja de dicho punto. En la ecuación (2.3) se observa que la activación de cada neurona oculta *j* dada por función gausiana alcanzará un valor alto de activación a medida que la entrada $\vec{x}$ se aleja del centro $\vec{c}$ dependiendo del radio *r*. Sólo aquellas neuronas ocultas cuyos centros estén en la vecindad de dicha entrada se van a activar. El resto de las neuronas ocultas permanecerán inactivas o con un menor nivel de activación. Esto no sucede cuando se utiliza funciones de activación del tipo sigmoidal porque éstas se activan en todo el rango de valores [VIÑ-03].

2.1.5 Diseño de RBFNs

En una RBFN el número de entradas y la salida viene dado por el número de variables que definen el problema. Sin embargo, existen aplicaciones en las que podría ser necesario un análisis de las variables más relevantes y significativas que definen el problema, lo que se denomina Input Variable Selection (IVS) que tratamos de utilizar en nuestra estructura jerárquica Multi-RBFN en capítulo 5. En el caso de *m* neuronas en la capa oculta de una RBFN, generalmente, se determinan por prueba y error, variando el número de neuronas hasta que pueda conseguir una red capaz de resolver el problema dado. En los últimos años había demasiado interés por desarrollar métodos que pudieran determinar el número de neuronas ocultas de forma automática. Uno de estos métodos es utilizar algoritmos genéticos [GOL-89], para realizar una búsqueda en el espacio de todas las posibles arquitecturas de redes [GRU-93], [WHI-96]. Otra metodología trata de desarrollar algoritmos incrementales, es decir, algoritmos que comienzan con una neurona oculta y van incrementando nuevas neuronas ocultas. Por lo tanto, estos algoritmos tratan no sólo de encontrar el número óptimo de neuronas, sino también los parámetros de dichas neuronas ocultas, como centros y radios.

2.1.6 Aprendizaje de las RBFNs

El proceso de aprendizaje implica la determinación de todos los parámetros de una RBFN. Estos parámetros son los siguientes; el tipo de la función base utilizada φ (normalmente gausiana), el número de éstas *m,* su vector de centros $\vec{c}_j$, sus radios r_j y los pesos óptimos w_j.

En una red de función de base radial las capas realizan tareas diferentes, de tal forma, que la separación del proceso de optimización es razonable. Así, para el problema de la capa oculta (centros $\vec{c}_j$, radios r_j) el proceso de optimización se centra en el espacio de la entrada, mientras que en la capa de la salida la optimización debe realizarse en base a la salida que se desea obtener.

2.1.6.1 Método de aprendizaje totalmente supervisado

Este método de aprendizaje no conserva las características locales de las RBFNs, en este caso todos los parámetros de RBFN (centros $\vec{c}_j$, radios r_j y pesos w_j) se determinan de forma supervisada utilizando técnicas de descenso gradiente para optimizar estos parámetros [RIV-03]. Estas técnicas actualizan los parámetros de forma iterativa. Por esto se utiliza una función de coste para minimizar el error cuadrático medio de aproximación, es decir, la diferencia entre la salida real y la salida de la red. El error se calcula por la siguiente forma:

$$Er = \sum_{i=1}^{n} \left(y_i - F(\vec{x}_i, \vec{c}, r) \right)^2 \tag{2.4}$$

donde y es la salida real, $F(\vec{x}_i, \vec{c}, r)$ es la salida de la red.

Para los parámetros de una RBFN (centros $\vec{c}_j$, radios r_j, y pesos w_j) se utilizan las siguientes ecuaciones para calcularlos. Estas ecuaciones representan una particularización del algoritmo (*Least Mean Squares*, LMS) [HAY-99] para las RBFNs.

$$\Delta\vec{c}_j = r_j \left[\alpha_1 (y_i - F(\vec{x}_i, \vec{c}, r))\, \varphi(\vec{x}_i) \left\| \vec{x}_i - \vec{c}_j \right\|^2 w_j \right] \tag{2.5}$$

$$\Delta r_j = \frac{r_j}{\sqrt{2r_j}} \left[\alpha_2 (y_i - F(\vec{x}_i, \vec{c}, r))\, \varphi(\vec{x}_i) \left\| \vec{x}_i - \vec{c}_j \right\|^2 w_j \right] \tag{2.6}$$

$$\Delta w_j = \alpha_3 (y_i - F(\vec{x}, \vec{c}, r))\, \varphi(\vec{x}_i) \tag{2.7}$$

Donde $\alpha_1\, \alpha_2\, \alpha_3$ son constantes que influyen en la velocidad de aprendizaje; estas constantes no tienen por qué tomar los mismos valores.

Este método supervisado sufre como desventajas que los radios r de la red no pueden alcanzar valores que hagan que el solapamiento de las activaciones de las neuronas ocultas sea lo más suave posible y la red no puede conservar sus características locales. En este método se inicializan todos los parámetros de la red de forma aleatoria con valores cercanos a cero.

2.1.6.2 Método de aprendizaje híbrido

Este método realiza el aprendizaje de una RBFN en dos fases:

- *Fase no supervisada*: en esta fase se determinan los valores iniciales de centros $\vec{c}_j$, y radios r_j de la red.

- *Fase supervisada*: en esta fase se calcula los valores exactos de los pesos óptimos w_j de la red.

2.1.6.2.1 Fase no supervisada

Las neuronas ocultas de una red de funciones de base radial se caracterizan por representar zonas diferentes del espacio de la entrada de la red. Por eso los centros $\vec{c}_j$, y radios r_j deben ser determinados para alcanzar este destino.

- **Algoritmos de *clustering* para inicializar los centros**

  Las técnicas de *clustering* han sido investigadas durante décadas, se desarrollaron inicialmente como algoritmos para resolver problemas de clasificación [MAC-67], aunque después han sido aplicadas en diferentes campos como redes neuronales artificiales [CHA-96].

  Los algoritmos de *clustering* son métodos de dividir un conjunto de *n* observaciones en *m* grupos, de tal modo, que los miembros de estos grupos sean diferentes. El número de grupos *m* puede ser prescrito o puede ser decidido por el algoritmo. Formalmente, un algoritmo de *clustering* produce una correlación *C: {1,..., n} a {1,..., m}* que asocia un grupo con cada ejemplo.

  Los centros de las funciones de base radial se determinan mediante algoritmos de *clustering* que permiten dividir el espacio de los datos de entrada en clases. El número de estas clases es el número de neuronas ocultas de una RBFN. De este modo, es importante resaltar que los algoritmos de *clustering* no suelen optimizar los centros de las RBFNs, sólo forman la fase inicial de estos parámetros. Entonces los algoritmos de *clustering* estudian las características de

los datos de entrada o los datos de entrada y salida que sirven para inicializar los valores de los centros $\vec{c}_j$.

Podemos clasificar los algoritmos *clustering* en dos tipos:

- Algoritmos de *clustering* no supervisados; algunos algoritmos importantes de este tipo son: el algoritmo k-medias [DUD-73], el algoritmo k-medias difuso [BEZ-81] y el algoritmo ELBG [RUS-99].

- Algoritmos de *clustering* supervisados; algunos algoritmos importantes de este tipo son: algoritmo de *clustering* para aproximación de funciones (CFA) [GON-02], algoritmo de *clustering* difuso condicional [PED-96], [PED-98] y algoritmos de estimación de grupos alternante [RUN-99] y Enhanced *clustering* para aproximación de funciones [AWA-05a].

En este capítulo explicamos de forma escueta tanto algunos algoritmos de *clustering* supervisados como algoritmos no supervisados.

1) **Algoritmo de las k-medias:** su procedimiento sigue un modo simple y fácil de clasificar, es decir, un conjunto de datos de entrada $\vec{x}$ dado por varios clusters fijado a priori. Esta técnica realiza una partición del conjunto de vectores de entrada en *m* grupos y establece la distancia entre los vectores de entrada como la característica de referencia para formar los *m* grupos. Se modifica la partición para reducir la suma de las distancias para cada caso del medio del cluster, esta modificación consiste en dividir cada caso a los más cercanos de los medios de *m* grupos de la partición anterior. Esto conduce a una nueva partición para la cual la suma de distancias es más pequeña que antes. Este método es rápido, pero converge a diferentes mínimos locales según las inicializaciones.

2) **Algoritmo de las k-medias difuso:** este algoritmo considera cada cluster como un conjunto difuso, de esta forma, cada función de pertenencia mide la posibilidad que cada vector de datos de entrada pertenecerá a un cluster. Por consiguiente, cada vector de entrada puede ser adjudicado a múltiples clusters con algún grado de la certeza medida por la función de pertenencia [KAR-95].

De este modo, este algoritmo presenta algunos inconvenientes, como por ejemplo, la aparición de nuevos grupos idénticos, y la finalización en el mínimo local más cercano de la partición.

3) **Algoritmo *enhanced* LBG:** Este algoritmo pertenece a los grupos de k-medias del vector cuantificado y se deriva directamente del algoritmo LBG [RUS-99]. El objetivo de este algoritmo es obtener una contribución igual de la distorsión en cada cluster. Este algoritmo está basado en el concepto de utilidad de cada cluster. Esto permite entender qué clusters están mal colocados y dónde deberían ser movidos para escaparse de la proximidad de un mínimo local en la función de error. Después de la evaluación de las utilidades de los clusters se identifican aquéllos con una utilidad baja para realizar el cambio inteligente de clusters que trata de cambiar el lugar de todos los clusters de utilidad baja cerca de estos con la utilidad alta. Esta modificación permite que el algoritmo se escape de mínimos locales y obtenga una asignación de clusters independiente de la configuración inicial.

4) **Algoritmo de estimación de grupos alternante (ACE):** este algoritmo consiste en determinar los conjuntos de la entrada teniendo en cuenta la salida. ACE utiliza una arquitectura alternante de iteración. Las funciones de *clustering* y pertenencia son seleccionados directamente por el usuario. ACE es una técnica de *clustering* para *clustering* difuso, donde el modelo de función objetivo es desechado y substituido por modelo más general, definido por la arquitectura del algoritmo de optimización alternante y por el usuario especificado, utilizando ecuaciones para actualizar parámetros desconocidos, que pueden o no ser derivados de un criterio de *clustering*. ACE puede ser usado para poner en práctica algoritmos de *clustering* conocidos como k-medias y k-medias difuso, o diseñar nuevas técnicas de *clustering* para mejorar un problema particular.

5) **Algoritmo de *clustering* difuso condicional (CFC):** el objetivo principal de este algoritmo es desarrollar clusters (campos receptivos) que conserven la homogeneidad de los modelos de *clustering*, teniendo en cuanta sus semejanzas en el espacio de entrada así como sus valores respectivos asumidos en el espacio

de salida [PED-98]. La base del algoritmo CFC esta en el algoritmo de k-medias difuso, realizado una extensión de éste. La parte condicional del mecanismo de agrupación reside en las variables de salida de los correspondientes variables de la entrada, una variable de salida de un vector de entrada describe el nivel en el cual este vector interviene en la construcción del grupo. Una consecuencia importante del uso de este método es que, la complejidad computacional reducida partiendo el problema original en una serie de contexto manejado del problema de *clustering*, Este característica hace difícil de comparar este método con otros métodos y no puede ser automatizado [GON-02].

6) **Algoritmo de *clustering* para aproximación de función (CFA):** El algoritmo CFA analiza la variabilidad de la salida de la función objetivo durante el proceso de *clustering* y aumenta el número de clusters en aquellas zonas de entrada donde la función de la salida objetivo es más variable. Este cambio del comportamiento del algoritmo de *clustering* mejora la interpretabilidad del sistema approximator obtenido, comparado con otros modelos utilizados para la clasificación tradicional.

   De este modo, el objetivo de este algoritmo es obtener una configuración mejor para acercarnos al óptimo global. CFA es una versión ponderada del algoritmo k-medias, donde cada vector de aprendizaje es dado un peso según un criterio de variabilidad de salida objetivo. CFA alcanza el estado de convergencia cuando el movimiento del cluster es insignificante, como k-medias. La diferencia es que la ecuación de distorsión usada para k-medias produce una distribución del cluster según la densidad de ejemplos en el espacio de entrada, y la nueva ecuación de distorsión diseñada para CFA concentra más clusters en aquellas zonas de entrada donde la salida es más variable utilizando un proceso de migración como en el algoritmo ELBG [GON-02].

7) **Algoritmo propuesto *enhanced* CFA:** varios algoritmos de *clustering* han sido utilizados para solucionar el problema de inicialización de modelos de aproximación funcional. Muchos algoritmos de *clustering* sufren inconvenientes principales: son lentos y no escalan un número elevado de puntos de datos y

convergen a mínimos diferentes locales basados en las inicializaciones. Este algoritmo propuesto intenta mejorar estos inconvenientes basado en los algoritmos de CFA [GON-02] y ELBG [RUS-99], se calcula el error provocado por cada cluster y migra clusters de zonas que tienen menor error a zonas que tienen mayor error, dependiendo de la salida objetivo. Si un algoritmo de *clustering* puede obtener una igualación de la distorsión en cada cluster, esto significa que cada cluster se encuentra colocado en su lugar adecuado para cubrir su parte del espacio de los datos de entrada y la distribución de los clusters es mejor. El algoritmo aumenta la densidad de clusters en las zonas de los datos de entrada migrando clusters que provoca menor error a zonas de clusters con mayor error, de este modo, se puede obtener un casi igualación del error en todas las zonas de los clusters en el espacio de los datos de entrada. Con esta homogeneidad del error, la distorsión será casi igual para todos los clusters en el espacio de los datos de la entrada, es decir, que cada cluster esta colocado en un lugar adecuado y cubre una determinada zona de los datos de la entrada [AWA-05a]

- **Algoritmos para inicialización de los radios**

Una vez han sido determinados los valores iniciales de los centros $\vec{c}_j = (c_1,...,c_m)$ de las funciones de base radiales, los radios r_j de dichas funciones deben calcularse de manera que cada neurona oculta debe activarse en una zona del espacio de los datos de entrada, de manera que el solapamiento de las zonas de activación de una neurona a las restantes vecinos sea lo más ligero posible para suavizar la interpolación. Estos valores de los radios deben cubrir todo el espacio de la entrada de manera que no quede ningún punto en el espacio de entrada sin cubrir.

Un método tradicional es el uso de un valor fijo de radios para todas las funciones base, utilizando esta ecuación $\sigma = d/\sqrt{2L}$, donde d es la distancia máxima entre los centros escogidos y L es el número de centros [PAR-91], [PAR-93]. Si cada función base tiene un radio individual, entonces el comportamiento de la red mejora [MUS-92], [BEN-02].

El radio de la función de base radial se puede determinar utilizando diversas heurísticas para fijar sus valores [MOO-89], [KAR-97]. De estas técnicas heurísticas las más utilizadas son los k vecinos más cercanos [MOO-89], y la técnica de la distancia media de los vectores de entrada más cercanos [KAR-97], estas dos técnicas heurísticas se describen a continuación:

1) *Heurística de los k vecinos más cercanos* (***knn***): esta técnica fija los radios de cada función base a un valor igual a la distancia euclídea media entre los centros de sus *k* funciones base más cercanos [MOO-89]. C_j a los C_p más cercanos:

$$r_j = \frac{1}{p}\sum_{p}\left\|C_j - C_p\right\| \tag{2.8}$$

2) *la distancia media de los vectores de entrada más cercanos* (***CIV***) : esta técnica determina el radio de la función base dependiendo de la siguiente ecuación:

$$r_j = \frac{\sum_{\vec{x}_i \in C_j}\left\|\vec{x}_i - \vec{c}_j\right\|}{\left|C_j\right|} \tag{2.9}$$

donde C_j es el conjunto de vectores de la entrada que están más cerca de $\vec{c}_j$ que de cualquier centro, y $\left|C_j\right|$ el número de vectores que componen este conjunto. Esta técnica produce menor solapamiento que la técnica de los *k* vecinos más cercanos *(knn)*.

Existe otra técnica bastante efectiva que se centra en determinar el radio de la función base como la media geométrica de la distancia del centro a sus vecinos más cercanos, esta técnica calcula el radio según esta ecuación: $r_j = \sqrt{\left\|C_j - C_t\right\| \left\|C_j - C_s\right\|}$ donde C_t y C_s los dos centros más cercanos al centro C_j [VIÑ-03].

2.1.6.2.2 Fase supervisada

En esta fase se calculan los pesos w_j entre la capa oculta y la capa de la salida de una RBFN. En este caso el objetivo es minimizar el error entre la salida de la red y la salida objetivo. Una vez que los valores de los centros y radios han sido fijados mediante el proceso de inicialización, la red de la función de base radial será lineal, y los pesos dependen del conjunto de entrenamiento y el proceso de aprendizaje está guiado por la minimización de una función del error calculado de esta siguiente forma:

$$Er_n = \frac{1}{2}\sum_{i=1}^{n}\left(F(\vec{x}_i,\vec{c},r)) - y_i\right)^2 \tag{2.10}$$

donde $F(\vec{x}_i,\vec{c},r)$ es la salida de la red de funciones de base radial, y y_i es la salida real.

El objetivo que persigue esta fase es encontrar el conjunto de los pesos óptimos peso óptimo que depende a la siguiente ecuación:

$$y(\vec{x}) \;=\; \sum_{j=1}^{m} w_j \;\cdot\varphi_j(\vec{x}) \tag{2.11}$$

Utilizando la notación matricial, la solución al problema de minimizar el error cuadrático consistirá en encontrar el conjunto óptimo de los pesos que hace esto posible, el problema se describe de forma matricial mediante la siguiente expresión:

$$\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} \;=\; \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_m \end{bmatrix} \begin{bmatrix} \varphi_{11} & \cdots & \varphi_{1m} \\ \varphi_{21} & \cdots & \varphi_{21m} \\ \vdots & & \vdots \\ \varphi_{n1} & \cdots & \varphi_{nm} \end{bmatrix} \tag{2.12}$$

reduciendo la ecuación (2.12), se obtiene la expresión:

$$\vec{y} \;=\; \vec{w}\;\varphi \tag{2.13}$$

donde y es la salida objetivo, w es la matriz del peso desconocido y φ es la matriz de activación. Para calcular la matriz de los pesos se utiliza esta expresión que depende de la ecuación (2.13):

$$\vec{w} \;=\; G\;\;\vec{y} \tag{2.14}$$

donde G es la matriz pseudo-inversa de la matriz de activación φ. Esta matriz puede ser calculada mediante métodos de resolución de ecuaciones lineales. Los métodos mas utilizadas para resolver un sistema de ecuaciones lineales para calcular el peso óptimo son: la descomposición de Cholesky [POM-00], [GOL-96], la descomposición en valores singulares (SVD) [CLI-83] [HOG-91] [GON-01], y el método de mínimos cuadrados ortogonales (OLS) [KEC-01], [CHE-91a]. Estos métodos vienen descritos a continuación:

I. Descomposición de Cholesky

La técnica de descomposición de Cholesky [POM-00], [GOL-96], es la más rápida del resto de técnicas para resolver un sistema de ecuaciones lineales, pero sólo se puede aplicar a sistemas descritos por una matriz cuadrada H simétrica y definida positiva. Es decir que $h_{ij} = h_{ji}$, $i, j = 1,...,n$, y para $\forall \vec{v} \in R^n$, $\vec{v}^t\, H\, \vec{v} > 0$. Cuando se aplica esta técnica al sistema de ecuaciones (2.13), la matriz de activación G debe verificar otra serie de condiciones que dependen de las características de la función de base radial. Multiplicando G^T por la ecuación (2.13) se obtiene:

$$\vec{y}\, G^T = G^T\, G\, \vec{w} \tag{2.15}$$

Sea $\vec{B} = \vec{y} G^T \in R^n$ y $H = G^T\, G \in R^{n x n}$ es la matriz de covarianza del sistema que es simétrica y definida positiva, la expresión que produce ecuación (2.15) será:

$$\vec{B} = H\, \vec{w} \tag{2.16}$$

Esta técnica de Cholesky descompone la matriz H para producir una matriz triangular inferior. Esta técnica solo falla cuando la matriz H no es definida positiva. Una vez se tenga la matriz H se puede utilizar la matriz triangular para resolver un sistema de ecuaciones lineales como la ecuación (2.13), y calcular los pesos óptimos de la red de funciones de base radial. En esta técnica las funciones de base radial deben ser colocadas para cubrir todo el espacio de la entrada, pero si hay funciones de base radial mal colocadas, entonces alguna no se activa con ningún punto de entrenamiento o existe demasiado soleamiento entre las funciones de base radial [RIV-03], esto produce matriz de valores singulares que esta técnica no puede resolver.

II. La descomposición en valores singulares (SVD)

El método SVD ha sido usado en muchos campos como en la compresión de datos, procesamiento de señales, análisis de modelos [KLE-80] y para resolver sistemas de ecuaciones lineales. Esta técnica se utiliza para resolver el problema de las funciones de base radial mal colocadas. En este caso se produce una matriz de activación singular. Cuando dos funciones son casi idénticas en la matriz de activación se producen dos columnas, prácticamente iguales, mientras que si una función base no se activa para casi ningún punto, se producirá una columna casi nula en la matriz de activación φ [RIV-03] [GON-01]. Las propiedades de los valores singulares son descritas detalladamente en lo siguiente:

Si la matriz de activación $G \in R^{mxn}$, entonces existen matrices ortogonales de esta forma:

$$\begin{aligned} U &= [u_1,...,u_m] \in R^{mxm} \\ V &= [v_1,...,v_n] \in R^{nxn} \\ U^T G\ V &= diag(\sigma_1,...,\sigma_p) \end{aligned} \tag{2.17}$$

donde p es el mínimo de *(m,n),* $\sigma_1 \geq \sigma_2 \geq ... \geq \sigma_p \geq 0$. σ_i , $i = 1,...,p$ son los valores singulares de *G*. Los valores singulares son las raíces cuadradas del valor propio λ_i, donde $\sigma_i = \sqrt{\lambda_i}$. La utilización de esta técnica para calcular los pesos óptimos de la red de funciones de base radial viene de la siguiente esta forma: De la ecuación (2.13), $G = V\ diag(\frac{1}{S})\ U^T$ donde $S = diag(\sigma_1,...,\sigma_p)$, entonces la ecuación (2.13) será:

$$\vec{w} = \left[V\ diag(\frac{1}{S})\ U^T \right] \vec{y} \tag{2.18}$$

La utilización de la descomposición en valores singulares facilita una solución para cualquier sistema de ecuaciones, y se puede tener una reducción del error en la salida de la red, también se puede ser utilizado para eliminar alguna función base cuyo valor singular asociado tuviera una magnitud pequeña o el error de aproximación no se vería afectado [GON-01]. También se puede usar SVD para estimar la relevancia de

las funciones base en el comportamiento de una red de funciones de base radial a la hora de simplificar su estructura. Este método sufre inconvenientes en la forma de seleccionar las funciones base más importante de la red, no tiene en cuenta la salida esperada del sistema para cada vector de entrada.

III. El método de los mínimos cuadrados ortogonales (OLS)

El método OLS [CHE-91a] es un método interesante para solucionar problema de sistemas de ecuaciones lineales. Es una técnica iterativa que selecciona en cada iteración la columna de una matriz de activación φ, la cual es la que más contribuye a la disminución del error de aproximación del modelo. El método OLS implica la selección secuencial de los centros de la función de base radial que asegura que tales nuevos centros elegidos son ortogonales a la selección anterior. En la elección de las mejores funciones de base radial, se mide la contribución de cada función de base radial a la disminución del error. Cada centro elegido disminuye el error cuadrado de la salida de la red, y el método se para cuando el error alcanza un nivel aceptable o cuando el número deseado de centros han sido elegidos.

El método OLS transforma las columnas $\vec{z}_j$ de la matriz de activación G en un conjunto de vectores ortogonales $\vec{u}_j$, entonces $G = U\ Q$, donde U es la matriz de columnas $\vec{u}_j$ y Q es una matriz triangular superior con unos en su diagonal. Con la substitución de estos parámetros en la ecuación (2.13) tenemos:

$$\vec{w} = U\ Q\ \vec{y} = U\ \vec{H} \tag{2.19}$$

donde $\vec{H} = Q\ \vec{y}$. El método OLS, igual al SVD a la hora de resolver el problema de las funciones bases mal colocadas, además tiene el inconveniente del SVD, en detectar funciones base poco útiles para la red y encontrar funciones base muy solapadas, eliminando alguna de éstas, aunque ésta tuviera importancia en el error de aproximación de la red. El algoritmo OLS es muy similar a la ortogonalización de *Gram-Schmidt* [GOL-96]. La única diferencia es que en vez de la ortogonalización de la matriz G en la orden de la primera columna a la última columna, ahora es primero la ortogonalización de las columnas que tienen la contribución más grande a la aproximación del vector de salida deseado y.

2.1.6.3 Métodos de aprendizajes evolutivos

Las redes de funciones de base radial pueden optimizar sus parámetros mediante la utilización de algoritmos evolutivos (AEs) [FON-95]. Un problema en las RBFNs que el número de los parámetros tiene que ser puesto antes de cualquier proceso de entrenamiento comienza. Sin embargo, no hay ninguna regla clara de como poner estos parámetros. Estos parámetros determinan el éxito del entrenamiento. Por combinación de algoritmos genéticos [ROD-02] con redes de funciones de base radial, los algoritmos genéticos pueden encontrar estos parámetros [GOL-89], pueden optimizar la arquitectura tanto como los parámetros de una RBFN.

- Aprendizaje de los parámetros de las RBFNs: se utilizan los AG para optimizar los centros y los radios, los pesos se calculan de forma óptima utilizando las técnicas de resolver ecuaciones lineales como SVD, OLS, etc.

- Optimización de la topología de las redes RBF: la optimización de la topología de las RBFNs incluye estos parámetros: el número de variables de entrada, el número de las neuronas en la capa oculta, la función de activación, etc.

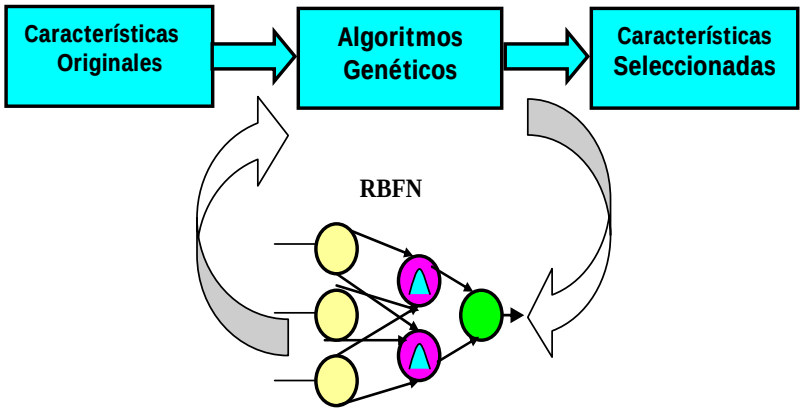


Fig. 2.7 Combinación de AGs y redes RBF.

Este método de optimización de los parámetros de la función de base radial, como radios r_j y centros $\vec{c}_j$, utilizan algoritmos genéticos. La forma de combinación de una red de función radial con algoritmos genéticos se muestra en Fig.2.7, y el algoritmo para calcular estos parámetros se muestra la siguiente Fig.2.8.

```
BEGIN Algoritmo Genético
Generar una población inicial. [centros, radios]
Computar la función de evaluación de cada individuo.[MSE]
WHILE NOT Terminado DO
BEGIN Producir nueva Generación
FOR Tamaño Población DO
BEGIN Ciclo Reproductivo
        Seleccionar dos individuos de la generación
Anterior
[C1….,Ck;R1,…Rk],[C*1,…,C*k;R*1,…,R*k]
Cruce los dos individuos seleccionados para obtener
        dos descendientes
Mutar los dos descendientes con cierta probabilidad.
        Computar la función de evaluación de los dos
        Descendientes Mutados.
        Insertar los dos descendientes mutados en la nueva
        Generación.
Calcular el peso óptimo usando SVD.[Wj]
        END
        IF La población ha convergido THEN
        Terminado =TRUE
END
END
```

Fig. 2.8 Ilustración de un Algoritmo Genético

Los AG comienzan con una población de cromosomas generada aleatoriamente, utilizando el mismo conjunto del entrenamiento para la optimización. Después, la función de fitness (que presenta el error cuadrático medio del proceso del entrenamiento) se utiliza para evaluar la población, estableciendo el fitness para cada cromosoma según su funcionamiento en el conjunto del entrenamiento. La mejor población será seleccionada para la siguiente generación, donde operadores genéticos, como el cruce y la mutación, produzcan una nueva población. El proceso de selección conduce la población al mejor valor del fitness (el error). El cruce y la mutación llevan a explorar las regiones desconocidas del espacio de búsqueda. Eventualmente, la población converge a los mejores parámetros de optimización de centros $\vec{c}_j$ y radios r_j de la función base radial [CAR-95], el proceso se repite hasta que el algoritmo genético encuentra el mejor fitness o hasta que llega al número máximo de generaciones con los mismos operadores genéticos en cada generación.

2.2 Redes neuronales *wavelet* (WNNs)

2.2.1 Introducción

Una red neuronal viene formada de múltiples capas de nodos interconectados con una función de activación en cada nodo y pesos entre estos nodos y la salida de la red. La salida de cada nodo es una función no lineal de todas sus entradas y la red representa una ampliación de la relación no lineal desconocida entre entradas x y salida y. En un espacio representado por las funciones de activación de los nodos de la red, el aprendizaje es visto como una aproximación de una función multi-dimensional. Los tipos de funciones de activación normalmente usados: global y local. Las funciones de activación globales se activan en una variedad grande de valores de entrada y proporcionan una aproximación global a los datos. Las funciones de activación locales son activas sólo en las cercanías inmediatas del valor de entrada dado [VAC-98].

Es conocido que las funciones pueden ser representadas como una suma de funciones de base ortogonales. Tales extensiones pueden ser fácilmente representadas como redes de neuronales teniendo las funciones de base seleccionadas como funciones de activación en cada nodo, y los coeficientes de ampliación como los pesos de la salida. Varias funciones clásicas ortogonales sufren, por lo tanto, de las desventajas de la aproximación que usa funciones globales. Lo que es necesario es un conjunto de funciones base locales y ortogonales [VAC-98]. Una clase especial de funciones, conocidas como *wavelet*, poseen propiedades de buenas localizaciones y son bases ortonormales. Así, pueden ser utilizadas como funciones de activación de una red neuronal como la denominada red neuronal *wavelet* (WNN) [WAL-91]. La idea de utilizar *wavelets* en redes neuronales has sido propuesto recientemente por Zhang y Benveniste [ZHA-92] y Pati y Krishnaprasad [KRI-93]. Las WNNs han tenido gran interés debido a que son aproximadores universales que consiguen la convergencia más rápida que las RBFNs, son capaces de tratar con el problema de la maldición de la dimensionalidad, convergen más rápido [DEL-95]. Además, generalizan a las RBFNs [TUN-02], [CRI-00].

La estructura de las WNNs es similar a las RBFNs, como se mostró en la figura 2.2, excepto que ahora la función de base radial es sustituida por funciones de base ortonormal, y no es necesaria la cubierta radial simétrica. La eficacia de este tipo de redes está en el aprendizaje de la función y su valoración [ZHA-95].

2.2.2 Diseño de redes neuronales *wavelet*

Las WNNs poseen un atributo único, además de la formación de una base ortogonal son también capaces de explicitar y representar el comportamiento de una función en varias resoluciones de variables de entrada. El concepto fundamental en la formulación y el diseño de redes neuronales con *wavelet* como funciones base, es la representación de multi-resolución de funciones que usan *wavelet*. Esto proporciona el marco esencial para el aprendizaje completamente por redes neuronales *wavelet* [VAC-98].

2.2.2.1 *Wavelets* ortogonales

Las *wavelets* son una familia de funciones y cada una viene definida por una parámetro de dilatación A_i que controla el parámetro de escalamiento y la traslación B_i que controla la posición de una función sola, denominada *wavelet* $\psi(x)$ [TUN-02]. El posicionamiento de funciones en un espacio de frecuencias temporales hace que una red *wavelet* puede reflejar las propiedades de frecuencia de la función más exactamente que las RBFN. Considerando un conjunto de aprendizaje de *n*-elementos, la respuesta total de una WNN es:

$$F(x) = w_0 + \sum_{j=1}^{m} w_j \cdot \psi_j \left(\frac{x - B_j}{A_j} \right) \tag{2.20}$$

donde *m* el número de los nodos *wavelet* en la capa oculta, w_j los pesos. Los autores en [ZHA-92] demostraron que las *wavelets* continuas se extienden al espacio $L^2(R_n)$ usando una *wavelet* de producto separable $\psi(x) = \psi_1(x_1)\psi_2(x_2)...\psi_n(x_n)$, y usando los parámetros *A* y *B*, se consigue construir:

$$\psi_{A,B} = \sqrt{|\mathrm{diag(A)}|}\,\psi\left(\mathrm{diag(A)(x - B)}\right) \tag{2.21}$$

Las WNNs pueden ser consideradas como una función de aproximación que estima una función desconocida utilizando la siguiente ecuación:

$$y = F(x) + \varepsilon \tag{2.22}$$

donde *F(x)* es la función de regresión y el término de error ε es una variable aleatoria de ruido con media igual a cero.

Hay varios métodos para la construcción de las WNNs [ZHA-97]. La construcción de una WNN implica dos etapas: Primero, construir una librería *wavelet W* de versiones de función madre de wavelet ψ:

$$W = \{\psi_1 : \psi_1(x) = \alpha_1 \psi(A_i(x - B_i))\} \tag{2.23}$$

donde

$$\alpha_i = \left(\sum_{p=1}^{n} \left[\psi(A_i(x_p - B_i)) \right]^2 \right)^{\frac{1}{2}}, \; i = 1,...,Q \tag{2.24}$$

donde x_p son los datos de entrada, Q el número de *wavelets* en W. Luego se seleccionan las M mejores *wavelets* para los datos de aprendizaje de la librería *wavelet W* [ZHA-95], a fin para construir la función de la regresión que viene dado por la siguiente expresión:

$$f_M(x) = \sum_{i \in I} u_i \cdot \psi_i(x) \tag{2.25}$$

donde I es un conjunto de M-elementos del conjunto $\{1,2,...,Q\}$ y $M \leq Q$. La Fig.2.9 presenta una función *wavelet* en una y dos dimensiones. Para minimizar la función del error dado por esta expresión:

$$Er(I) = \min_{u_i, i \in I} \frac{1}{n} \sum_{p=1}^{n} \left(y_p - \sum_{i \in I} u_i \cdot \psi_i(x_p) \right)^2 \tag{2.26}$$

Zhang deriva dos algoritmos heurísticos, a saber, *stepwise selection* por ortogonalización para decidir las *wavelets* apropiadas en las unidades ocultas y *backward elimination* para elegir el número de unidades ocultas. El número de *wavelets*, *M,* es elegido como el mínimo del criterio de error de predicción final de denominado *Akaike* (FPE) [ZHA-95]. Después de que WNN inicial es construida, se entrena por algoritmos de descenso gradiente como *least mean squares (LMS)* que minimice el error cuadrático medio.

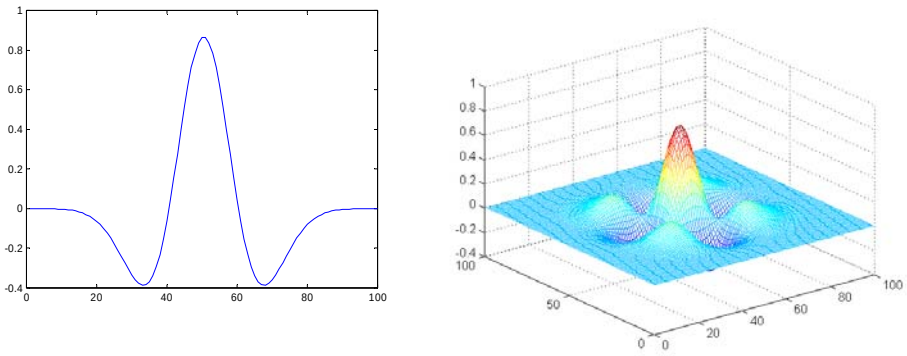


Fig. 2.9 Función Wavelet en una y dos dimensiones

2.3 Algoritmos de optimización

2.3.1 Introducción

La teoría de optimización y sus técnicas diferentes, se utilizan para encontrar el valor de un conjunto de parámetros, que maximizan o minimizan una función de coste o error. Esta función determina si la solución de un problema está bien o no. La función de coste típicamente podría ser la diferencia (o el error) entre la salida de la red y la salida real del problema que trata. La mayor parte de los problemas de optimización en el mundo real tienen un alto número de soluciones. La dirección de búsqueda debería ser especificada para mejorar su capacidad de búsqueda.

Los problemas de optimización están compuestos de tres elementos básicos:

- Una función objetivo que queremos minimizar o maximizar. Por ejemplo, en un proceso de optimización, los parámetros de una red de funciones de base radial van dirigidos a minimizar el error que es la diferencia entre la salida real y la salida actual de una RBFN.

- Un conjunto variables que afectan el valor de la función objetivo. En el problema de optimización de los parámetros de una red de funciones de base radial, las variables podrían incluir los parámetros de la red RBF que queremos optimizar (centros, radios, pesos, etc.).

- Un conjunto de restricciones que permiten a los valores no conocidos tomar ciertos valores y excluir otros.

El problema de optimización consiste entonces en encontrar los valores de las variables que minimizan o maximizan la función objetivo hasta que ninguna solución mejor puede ser encontrada.

Matemáticamente, un problema de optimización consiste en encontrar un vector de parámetros libres $\vec{v} = (v_1,...,v_n) \in N$ donde N es el dominio que define el sistema considerado con criterio de calidad $f: N \to R$, f es la función objetivo que trata de optimizar de forma $f(\vec{v}) \to \max$. En el problema de optimización global, las técnicas de optimización tratan de encontrar un vector $\vec{v}^*$, de modo que:

$$\forall \vec{v} \in N : f(\vec{v}) \leq f(\vec{v}^*) = f^* \tag{2.27}$$

Esta fórmula será válida en el caso de maximización de la función objetivo; en el caso de minimización se varía la desigualdad. Sin embargo, los problemas de multi-modalidad afectan mucho a la optimización, hasta llegar a ser imposibles, es decir que, existen varios extremos locales $\vec{v}^*$:

$$\exists \delta > 0: \forall \vec{v} \in N: \Omega(\vec{v},\vec{v}^\bullet) < \delta \Rightarrow f(\vec{v}) \leq f(\vec{v}^\bullet) \tag{2.28}$$

donde el símbolo δ es la medida de la distancia en *N*. Además existen restricciones en el conjunto *N*. Estas restricciones pueden ser 'restricciones de igualdad', o 'restricciones de desigualdad'. Existen otras dificultades en la resolución de un problema de optimización, como la gran dimensionalidad, la no linealidad, el ruido, etc.

Existen muchos algoritmos de optimización; las primeras estrategias utilizan métodos matemáticos [DEN-83], que buscan explotar las características matemáticas de la función que optimiza. Estos métodos requieren ciertas propiedades en la función objetivo, teniendo en cuenta estas características, $\vec{v}^* \in N$ es un punto fijo de $f(\vec{v})$, si $g(\vec{v}^*) = 0$, donde $g(\vec{v})$ es el gradiente de la función objetivo $f(\vec{v})$ y $g_i(\vec{v}) = \partial f(\vec{v}) / \partial v_i$, el punto $\vec{v}^*$ es un mínimo local de la función $f(\vec{v})$. La matriz de la segunda derivada

(matriz Hessiana) $H(\vec{v})$ se define como $H_{ij}(\vec{v}) = \partial^2 f(\vec{v}) / \partial v_i \, \partial v_j$. Esta matriz es definida positiva en $\vec{v}^*$, es decir $\vec{S}^T H(\vec{v}^*) \vec{S} > 0$, $\forall \vec{S} \neq 0$. Un algoritmo de optimización, en general, comienza con un punto de partida $\vec{v}_0$, calcula la dirección de la búsqueda P_k, determina el tamaño del paso de la búsqueda α_k y la comprueba en cada iteración como se muestra la siguiente expresión:

$$\vec{v}_{k+1} = \vec{v}_k + \alpha_k \, . \, \vec{P}_k \tag{2.29}$$

Estos pasos se repiten hasta que no cumpla la condición de parada. Tras la aplicación de estos pasos se determina una línea de búsqueda y un intervalo que va limitando más y más hasta encontrar la solución deseada.

Entre los métodos más utilizados de optimización se encuentran el método de *Steepest-Descent* [DEN-83], el método de *Newton-Raphson* [KEC-01], el método de *Gauss-Newton* [KEC-01], el método del gradiente conjugado [POW-77] y el método de *Levenberg-Marquardt* [MAR-63], [KEC-01].

2.3.2 El método *Steepest-Descent* (SD)

Este método de optimización es el más sencillo de todos los algoritmos de optimización, bastante robusto y fácil de implementar. Se define el vector de descenso como él opuesto al gradiente de la función objetivo en la configuración actual y obtiene la nueva configuración haciendo una búsqueda lineal en esta dirección. Cada una de las direcciones que se producen en el proceso de búsqueda, son perpendiculares a las anteriores [GON-01]. Esto hace que este método vaya bastante lento para descensos largos, y en la mayoría de las veces no puede ir de forma directa hacia el mínimo porque no encuentra en una dirección perpendicular a la anterior. Otro inconveniente, es que cuando la configuración actual está cerca del mínimo, el gradiente tendrá valor cero y esto decrece el valor del peso óptimo, de forma que cuanto más cerca del mínimo esté, será más lenta la convergencia y no será capaz de atravesar zonas donde la función se mantenga constante.

Para evitar esta posibilidad se calcula el descenso de regresión de la última *n* iteración. En cada iteración del método SD, la dirección de búsqueda es tomada como $-g_k$ el gradiente negativo de la función objetivo en L_k. Recuerde que una dirección de descenso $\vec{P}_k$ satisface $g_k^T \vec{P}_k < 0$. El modo simple de garantizar la reacción negativa de este producto interior es elegir $\vec{P}_k = -g_k$. Esta opción también minimiza el producto interior $-g_k^T \vec{P}_k$ para vectores de longitud de unidad, de modo que, el vector del descenso $\vec{P}_k$ se pone como opuesto al gradiente de la función objetivo *f*, es decir:

$$\vec{P}_k = -\vec{g}_k = -\nabla f(\vec{v}_k) \tag{2.30}$$

El método se suele utilizar en los primeros pasos de algoritmos de búsqueda local más potentes.

2.3.3 El método de *Newton-Raphson*

El método *Newton-Raphson* [KEC-01], o el método de *Newton*, es una técnica poderosa para solucionar ecuaciones numéricas, como el cálculo diferencial, y está basada en la idea simple de la aproximación lineal. El método *Newton-Raphson* usa $\vec{v}_k$, que es un mínimo de la aproximación cuadrática y no de la función de error original no cuadrática, como el siguiente punto corriente, la fórmula iterativa:

$$\vec{v}_{k+1} = \vec{v}_k - H_k^{-1} . \vec{P}_k = \vec{v}_k - \eta \ H_k^{-1} . \vec{P}_k \tag{2.31}$$

donde H_k es una matriz Hessiana de *(N,N),* η es la tasa de aprendizaje determinada por una línea de búsqueda de $\vec{v}_k$ en la dirección de $H_k^{-1} . \vec{P}_k$. La convergencia de este método es rápida cuando $\vec{v}_k$ está muy cerca del punto óptimo $\vec{v}_0$. Sin embargo, la convergencia al mínimo no es garantizada, y H_k no es positivo definido, y el método puede dejar de convergir.

2.3.4 El método del gradiente conjugado

La desventaja principal del método del gradiente estándar es que no funciona bien sobre hipar-superficies que tienen curvaturas diferentes a lo largo de la dirección de la función objetiva diferente. La función de error no es más radialmente simétrica [KEC-01]. Otra

razón de aplicar el gradiente conjugado es después de que la búsqueda lineal sea aplicada, los pasos iterativos sean ortogonales el uno al otro, y éstos necesariamente conducen al cambio no deseado agudo de la dirección de pendiente. El método del gradiente conjugado [POW-77] es un gran método para matrices simétricas y definidas positivas, ya que aprovecha muy bien la estructura de la matriz y tiene muy buenas propiedades de estabilidad numérica.

El método del gradiente conjugado es un método iterativo que a partir de una iteración inicial va calculando sucesivos iteraciones que se van acercando a la solución exacta del sistema lineal. Este método intenta resolver los problemas de avance de método de *Steepest-Descent* mediante vectores conjugados.

Dos vectores distintos $\vec{P}_i$ y $\vec{P}_j$ son conjugados cuando son ortogonales con cualquier matriz positiva (Hessiana) *H*, de modo que:

$$\vec{P}_i^T \; H \, \vec{P}_j = 0 \tag{2.32}$$

Esta definición es como una generalización de la condición de ortogonalidad, *H* es la matriz unidad. Una búsqueda en la dirección $\vec{P}_i$ será perpendicular al gradiente de la función *f,* lo que va manteniendo en las siguientes direcciones. Ahora cada uno de los vectores de búsqueda $\vec{P}_i$ será dependiente de todas las direcciones en las que se ha buscado para encontrar el mínimo de la función *f.* El movimiento iterativo a través de estas direcciones asegura que se consideran direcciones que no se han utilizado antes, por lo que la búsqueda será muy rápida. Este conjunto de direcciones se llaman direcciones *H* ortogonales. Para una función cuadrática, el primer vector se calcula igual que en el método *Steepest-Descent* ecuación (2.30); los siguientes vectores conjugados se calcularán:

$$\vec{P}_{k+1} = -\vec{p}_{k+1} + \beta_k \; \vec{P}_k \tag{2.33}$$

La manera de calcular β_k da como resultado diferentes clases de algoritmos [POW-77]. Una de las formas clásicas viene dada por la fórmula de *Fletcher-Reeves* [FLE-64]:

$$\beta_k = \frac{-\vec{g}_{k+1} \cdot \vec{g}_{k+1}}{\vec{g}_k^T + g_k} \tag{2.34}$$

Fletcher-Reeves [KEC-01] sugiere que la dirección de búsqueda debiera volver periódicamente a la dirección de *Steepest-Descent*. El tamaño del paso α_k para cada dirección se calcula por la fórmula:

$$\alpha_k = \frac{\vec{P}_k^T \ \vec{g}_k}{\vec{P}_k^T \ H \ \vec{P}_k} \tag{2.35}$$

Con este procedimiento, el algoritmo retiene la propiedad de la terminación cuadrática, a condición de que tal terminación se reactive con cada iteración. Aunque resulta difícil conseguir búsquedas precisas por las restricciones impuestas por la resolución y el tiempo de los cálculos realizados. Cuando la función no es cuadrática, existen otras formas de calcular β_k [POW-84], pero la forma de la función cambia y es diferente a una función cuadrática. La convergencia es importante porque el método gradiente conjugado es comúnmente usado para problemas tan grandes en los que no es factible controlar hasta *n* iteraciones, y es útil para problemas que están fuera del alcance de cualquier algoritmo exacto.

2.3.5 El método de *Levenberg-Marquardt*

Este método es una modificación del método de *Newton* [KEC-01], y fue diseñado para minimizar funciones que fueran la suma de los cuadrados de otras funciones no lineales; es por ello que el algoritmo de *Levenberg-Marquardt* [MAR-63], tiene un excelente desempeño en el entrenamiento de redes neuronales donde el rendimiento de la red está determinado por el error medio cuadrático. La evolución hacia el óptimo en este método es más rápida en el caso de que el espacio de búsqueda tenga varias dimensiones [VIA-03]. El método se adapta al contexto de la función para conseguir acercarse rápidamente al óptimo.

El método parte de la aproximación de un conjunto de datos $(\vec{x}_i, y_i) \in R^n$, R, $i = 1,...,n$, la función de aproximación $f(\vec{x}, \vec{q})$, donde $\vec{q}$ es un vector que representa un conjunto de parámetros que caracterizan la función. Para minimizar la función de aproximación se utiliza la función del error que se define como:

$$\mathrm{E}(\vec{q}) \;=\; \sum_{i=1}^{n}\left(\frac{y_i - f(\vec{x}_i,\vec{q})}{\lambda_i}\right) \tag{2.36}$$

donde λ_i es una medida del error (desviación estándar) en el punto *i*. Cuando no se sabe el valor de λ_i se fija a 1.

Esta función del error se aproxima cuando se obtiene el desarrollo en series Taylor de la función *E* en el punto $\vec{q}_i$. Este error se calcula como en la siguiente forma:

$$E(\vec{q}) \approx E(\vec{q}_i) \;+\; \vec{g}_i^{\,T}\,\vec{q} + \frac{1}{2}\vec{q}^{\,T}\; H_i\,\vec{q} \tag{2.37}$$

donde $\vec{g}_i$ es el vector gradiente de *E* en el punto $\vec{q}_i$, y H_i es la matriz Hessiana de *E* en $\vec{q}_i$. Esta aproximación se utiliza cuando está cerca del mínimo, si no se utiliza el método *Steepest-Descent*, para dar un paso en el sentido opuesto al gradiente. Por otro lado, si el error de aproximación es pequeño, se puede saltar directamente al mínimo de esta forma:

$$\vec{q}_{\min} = \vec{q} - H_i^{-1}\;\vec{g}_i \tag{2.38}$$

El cálculo de la matriz Hessiana *H* es muy costoso. En el caso que se conozca exactamente la función del error *E*, se puede calcular tanto el gradiente como la matriz Hessiana *H*. Esta matriz se forma:

$$\frac{\partial^2 E(\vec{q})}{\partial q_k \partial q_l} = 2\sum_{i=1}^{n}\left(\frac{\partial f(\vec{x}_i,\vec{q})}{\partial q_k}\frac{\partial f(\vec{x}_i,\vec{q})}{\partial q_l} - (y_i - \partial f(\vec{x}_i,\vec{q}))\,\frac{\partial^2 f(\vec{x}_i,\vec{q})}{\partial q_k \partial q_l}\right) \tag{2.39}$$

Para simplificar esta ecuación se define:

$$B_k \equiv -\frac{1}{2}\frac{\partial E}{\partial q_k} \;\text{ y }\; A_{kl} \equiv \frac{1}{2}\frac{\partial^2 E}{\partial q_k \partial q_l} \tag{2.40}$$

Con estas expresiones, sea $Z = \left[A_{kl}\right] \;=\; 1/2\;\; H_i$ en ecuación (2.38) tenemos que:

$$\sum_{i=1}^{m} A_{kl}\;\Omega q_l = B_k \tag{2.41}$$

Este conjunto de ecuaciones se resuelve para calcular los incrementos Ωq_l que, añadidos a la configuración actual, proporcionan la siguiente aproximación. En el contexto de mínimos cuadrados, la matriz *Z,* que es igual a la mitad de Hessiana *H,* se suele llamar 'matriz de curvatura'. Entonces, teniendo en cuenta la ecuación (2.41) tendremos:

$$\Omega q_l = \tau B_l \tag{2.42}$$

donde τ es un constante. Las segundas derivadas de la función del error E dependen de la primera y la segunda derivada de la función $f(\vec{x},\vec{q})$, y el cálculo de la segunda derivada de la función $f(\vec{x},\vec{q})$ es complicada debido a la complejidad de la ecuación y su número. Además, cuando el modelo se encuentra al lado del mínimo, aumenta el coste computacional, como la segunda derivada de $f(\vec{x},\vec{q})$ va multiplicada por $(y_k - f(\vec{x},\vec{q}))$, que va teniendo a cero se suele obtener cantidades aleatorias que se cancelan cuando se realiza la suma de los ejemplos de entrenamiento. Teniendo en cuenta eso, la ecuación (2.41) sólo se usa cuando la configuración actual se encuentra cerca del mínimo. También se puede omitir el cálculo de la segunda derivada y redefinir el cálculo de A_{kl} como:

$$A_{kl} = \sum_{i=1}^{n} \left(\frac{\partial f(x_i,\vec{q})}{\partial q_k} \frac{\partial f(x_i,\vec{q})}{\partial q_l} \right) \tag{2.43}$$

El método *Levenberg-Marquardt* desarrolló un método elegante para resolver el problema de la conmutación suave entre los pasos del método *Steepest-Descent* y el cálculo del mínimo de forma directa utilizando la matriz Hessiana [MAR-63]. El método está basado en dos ideas: el valor del constante τ en la ecuación (2.42), para analizar los componentes de la matriz Hessiana. La cantidad calculada por la función E es adimensional, es sólo un número, por lo que sirve de poco para fijar el valor τ. Por otro lado, la constante de proporcionalidad entre q_j y Ωq_j debe tener dimensiones de q_j^2. Revisando los componentes de la matriz *Z*, se puede comprobar que sólo hay una única cantidad con esas dimensiones, y que es $1/A_{jj}$, el recíproco de los elementos de su diagonal, así que debe ser escala de constante. Pero esta escala podría ser demasiado

grande, por lo que se puede dividir por una constante adimensional μ con la posibilidad de que $\mu >> 1$, para poder encontrar el tamaño de los pasos como se desee.

Teniendo en cuenta la ecuación (2.42) se puede reemplazar por esta expresión:

$$\Omega q_l = \frac{1}{\mu A_{ll}} B_l \tag{2.44}$$

donde A_{ll} debe ser positivo, lo que está garantizado por la ecuación (2.43). La segunda idea de *Marquardt* fue que las ecuaciones (2.41) y (2.44) se pueden combinar si se define una nueva matriz Z^* utilizando las siguientes ecuaciones:

$$Z^*_{jj} \equiv Z_{jj}(1+\mu) \qquad Z^*_{jk} \equiv Z_{jk} \qquad (j \neq k) \tag{2.45}$$

y se reemplazan las ecuaciones (2.41) y (2.44) por:

$$\sum_{l=1}^{m} Z^*_{kl}\, \Omega q_l = B_k \tag{2.46}$$

Cuando μ es muy grande, la matriz Z^* se fuerza para ser diagonalmente dominante. Así, la ecuación (2.46) tiende a ser idéntica a (2.44). Por otro lado, cuando μ se aproxima a cero, la ecuación (2.46) tiende a (2.41).

Los principales pasos del método de *Levenberg-Marquardt* vienen dados de esta manera:

1. Calcular el error en el nuevo vector $\mathrm{E}(\vec{q}_0)$.
2. Establecer $\mu = 0.001$.
3. Repetir mientras no se cumple la condición de parada.
   a. Resolver el sistema de ecuaciones (2.46).
   b. Calcular $\vec{q}_{k+1} = \vec{q}_k + \Omega\vec{q}$.
   c. Si el error ha aumentado por lo tanto la actualización, entonces retrae el paso, y aumenta μ por un factor de 10 o algún tal factor significante. Si el error se ha disminuido a consecuencia de la actualización, entonces acepta el paso y disminución por un factor de 10 más o menos. $\mathrm{E}(\vec{q}_{k+1}) \geq \mathrm{E}(\vec{q}_k)$ entonces $\mu_{k+1} = \mu_k \times 10$, si no: $\mu_{k+1} = \mu_k / 10$

Si el error aumenta es decir que la aproximación cuadrática no trabaja bien y no es probable que esta cerca de un mínimo, entonces debería aumentar μ a fin de mezclar más hacia el simple descenso gradiente. A la inversa, si el error se disminuye, la aproximación trabaja bien, y espera que nos hagamos más cerca a un mínimo. Entonces, μ disminuido para tener una cuenta más sobre la matriz Hessiana. Este método es un método heurístico, no es óptimo para algún criterio de velocidad o error final bien definido. Esto es simplemente resuelta una optimización buena. De esta forma, se ha convertido en un estándar virtual para optimización del medio de modelos no lineales. Este método es mucho más rápido que el descenso gradiente.

2.4 Selección de variables de entrada

2.4.1 Introducción

La selección de variables de entrada (*Input Variable Selection*, IVS) o la selección de atributos, ha sido un tema de investigación tradicional desde los años 70 [MOC-71]. IVS es un tema amplio que sirve para investigar disciplinas como la estadística [NAR-77], reconocimiento de patrones [JAI-82], [STE-76], [KIT-87], aprendizaje de máquinas [HAL-99], las redes neuronales [SET-97], etc. IVS ha sido y sigue siendo la fuente de muchas investigaciones en muchas áreas de la aplicación de problemas reales para subconjuntos con decenas o cientos de miles de variables que estuvieran disponibles, como sistemas biomédicos, industriales, ambientales, etc. El problema ocurre cuando se desarrollan modelos de multi-variables aleatorias y cuando los mejores conjuntos de entradas no son conocidos.

La selección de un subconjunto de variables de entrada es un proceso de identificación que consiste en quitar tanta información irrelevante y redundante como sea posible. Esto reduce la dimensionalidad de los datos y permite a los algoritmos de aprendizaje funcionar más rápido y con más eficacia, y la mejora de la calidad de datos aumentando la exactitud del modelo resultante [YU-03].

Para explicar el proceso de la selección de variables de entrada de forma matemática, consideremos X es el espacio original de las variables con número de dimensiones d y $\hat{x}$ es el espacio seleccionado de las variables con número de dimensiones $\hat{d}$, $\hat{x} \subseteq X$, $J(\hat{x})$ es la función de selección. Sin pérdida de generalidad, se asume que un valor más grande de J indica un mejor espacio de variables de entrada. El objetivo es maximizar $J()$. Generalmente, el problema de selección de variables de entrada para encontrar un sub-espacio de variables de la entrada $\hat{x} \subseteq X$ se calcula de forma general de la siguiente forma:

$$J(\overline{X}) = \max_{Z \in x, |Z| = \overline{d}} J(Z) \tag{2.47}$$

Si una búsqueda exhaustiva es realizada, entonces tenemos que considerar todas las combinaciones posibles del conjunto:

$$\binom{d}{\hat{d}} \tag{2.48}$$

El número de combinaciones crece exponencialmente, haciendo la búsqueda exhaustiva impracticable para valores grandes de *d,* incluso para valores moderados de *d*. Entonces, realizando la búsqueda exhaustiva es poco práctico. La obtención del mejor subconjunto de variables de entrada es, por lo general, intratable y muchos problemas relacionados con la selección de variables de entrada han sido mostrados para ser difíciles [BLU-92].

Existen tres clases de estrategias para selección de variables de entrada:

1- El número de variables $\hat{d}$ es dado, y la tarea de los algoritmos de búsqueda es decidir qué variable de $\hat{d}$ constituyen el subconjunto óptimo de características seleccionadas.

2- Buscar la dimensionalidad más pequeña de las variables de la entrada para la cual la actuación de discriminación excede un valor especificado.

3- Selecciona el subconjunto de variables de la entrada óptimas que tienen una compensación entre las clases de discriminación y el tamaño del subconjunto.

¿Por qué seleccionar las variables de la entrada?

- Cuando aumentan la dimensionalidad de entrada, la complejidad computacional aumenta y las exigencias de memoria del modelo también.
- El aprendizaje es más difícil con entradas no-requeridas.
- La falta de convergencia y la mala exactitud del modelo pueden resultar por usar entradas adicionales no-requeridas.
- Si la mayor parte de las variables de entrada son ruidosas, entonces la mayor parte de los parámetros serán inútiles.
- El entendimiento de modelos complejos son más difíciles que los modelos simples que dan resultados comparables.
- Sirven para más entendimiento y eficacia.

La determinación de las variables de entrada que son relevantes a la tarea del aprendizaje es un objetivo importante en el aprendizaje cuando la inclusión de variables irrelevantes o redundantes puede reducir la actuación de diferentes algoritmos de aprendizaje.

2.4.2 Características generales de un método de IVS

Los investigadores han estudiado varios aspectos de la selección de variables de entrada. Se argumenta que las cuatro cuestiones siguientes que afectan a la naturaleza de la búsqueda [YU-03] pueden caracterizar cualquier método de selección de variables de entrada:

A. El punto de partida: Para comenzar la búsqueda, debemos seleccionar un punto en el espacio de los subconjuntos de variables de entrada. Este proceso puede afectar la dirección de la búsqueda. Una opción sería comenzar con alguna variable y sucesivamente añadir atributos. En este caso, la búsqueda procede de forma incremental por el espacio de

búsqueda *(forward selection)*. Al contrario, la búsqueda puede comenzar con todas las variables de la entrada y sucesivamente quitarlas. En este caso, la búsqueda procede hacia atrás por el espacio de búsqueda *(backward selection)*. Un tercer método podría consistir en combinarse hacia delante y hace atrás *(forward and backward selection)* y el punto debe comenzar en algún sitio intermedio [HAL-99].

B. **La organización del procedimiento de búsqueda**: Una búsqueda exhaustiva del sub-espacio de variables de la entrada es prohibitiva [HAL-99], excepto para un pequeño número inicial de variables de la entrada. Con las variables de la entrada iniciales N existen 2^N subconjuntos posibles. Las estrategias heurísticas de búsqueda son más factibles que exhaustivas y pueden dar resultados buenos, aunque no garanticen un descubrimiento del subconjunto óptimo.

C. **La estrategia de evaluación:** Los subconjuntos de variables de entrada evaluados son el factor de diferencia más grande entre algoritmos de selección de variables para el proceso de aprendizaje. La selección de variables de entrada del tipo *filter* (FFS) funciona independiente de cualquier variable indeseable de algoritmos de aprendizaje [KOH-95], [KOH-96], que son filtrados de los datos antes de que el aprendizaje comience. Otro método argumenta que la tendencia de un algoritmo de inducción particular debería ser considerada seleccionando variables de entrada. Este método, denominado 'la selección de características del tipo *wrapper*' (WFS) [KOH-95], [KOH-96] usa un algoritmo de inducción junto con una técnica de nueva prueba estadística como la validación cruzada para estimar la exactitud final de subconjuntos de de variables seleccionadas.

D. **El criterio para parar la búsqueda:** durante el proceso de evaluación, podríamos querer parar la búsqueda. Un seleccionador de variables de entrada debe decidir cuándo dejar de averiguar el espacio de subconjuntos de variables. Según la estrategia de evaluación, un seleccionador de variables de entrada podría dejar de añadir o quitar variables de entrada

cuando ninguna de las alternativas mejore el mérito de un subconjunto de variables de entrada corriente [HAL-99]. El algoritmo podría seguir revisando el subconjunto de variables de entrada mientras el mérito no degrada. Una opción adicional podría seguir generando subconjuntos de variables de entrada hasta el alcance del extremo opuesto del espacio de búsqueda y, luego, seleccionar lo mejor.

La selección de variables de entrada puede ser generalmente considerada como un problema de optimización. Para un problema de optimización general, se puede usar la categoría de árbol de optimización que divide técnicas de optimización en optimización discreta y optimización continua [YU-03], de los cuales ambos divididos en otras subcategorías.

A continuación describimos dos modelos típicos: el modelo de selección del tipo *wrapper* (WFS) [KOH-97] [KOH-95], y el modelo de selección del tipo *filter* (FFS) [JOH-94] [KOH-95].

2.4.3 IVS tipo *wrapper* (WFS)

La estrategia del modelo de selección de variables de entrada WFS [KOH-97], [KOH-95] usa un algoritmo de inducción para estimar el mejor subconjunto de variables de entrada buscadas en el conjunto de entrenamiento y utilizan una exactitud estimada del resultado del clasificador como su mérito.

Los métodos WFS tienen mejores resultados que los métodos del tipo FFS, porque son afinados a la interacción específica entre un algoritmo de inducción y sus datos de entrenamiento. La desventaja de este modelo es menos manejable debido al coste prohibitivo de controlar el algoritmo de clasificación muchas veces cuando la dimensionalidad es bastante alta [DOA-92].

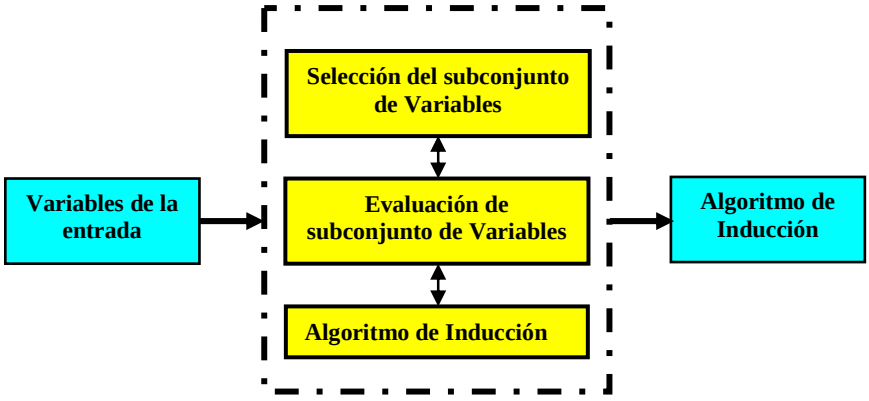


Fig. 2.10 Selección del Tipo *Wrapper*

2.4.4 IVS tipo *Filter* (FFS)

El modelo de selección de variables de la entrada del tipo FFS [KOH-95] fue el primer método utilizado para la selección de variables de entrada. Se utiliza un criterio independiente de búsqueda para encontrar el subconjunto de variables de entrada apropiado antes de realizar el algoritmo de aprendizaje. Así fue denominado 'el método tipo *filter* (FFS)' por John, Kohavi y Pfleger [JOH-94], [KOH-95], ilustrado en la figura 2.11.

Fig. 2.11 Selección del tipo *Filter*

Este método empieza filtrar eliminando atributos irrelevantes antes de que la inducción ocurra, es decir, la búsqueda se realiza independientemente de un algoritmo de inducción. La ventaja del modelo del tipo FFS consiste en que no necesita controlar de nuevo el algoritmo para cada algoritmo de inducción cuando se ejecuta sobre un conjunto de variables de la entrada. Generalmente, el modelo del tipo FFS es un método computacional eficiente, y es práctico para conjuntos de datos con dimensionalidad muy alta. Todos los métodos del tipo FFS usan la heurística basada en las características generales de los datos mejor que un algoritmo de aprendizaje para evaluar el mérito de

subconjuntos de variables de entrada. Los métodos del tipo *filter* son generalmente mucho más rápidos que los métodos del tipo *wrapper*, y son más prácticos para usarlos sobre datos de dimensionalidad alta.

2.4.5 Análisis de componentes principales (PCA)

2.4.5.1 Introducción

El análisis de componentes principales (PCA) [SMI-02], [GUR-00], es una técnica estadística de síntesis de la información o reducción de las dimensiones (número de variables) que conserva la estructura de varianza/covarianza. Es decir, ante un conjunto de datos con muchas variables, el objetivo será reducirlas a un menor número perdiendo la menor cantidad de información posible. La reducción de dimensionalidad de un conjunto de variables de entrada es un paso de proceso previo común utilizado para reconocimiento de modelo y aplicaciones de clasificación y en esquemas de compresión [SMI-02]. El análisis de componentes principales (PCA) es un método popular. Sin embargo, esto tiene la desventaja que mide todas las variables de la entrada originales que se usan en la proyección para reducir el espacio dimensional, por unas combinaciones lineales de las variables originales. Los nuevos componentes principales o factores serán una combinación lineal de las variables originales, y además serán independientes entre sí [COH-99].

En muchos problemas reales que necesitan reducir la dimensionalidad de un sistema es un paso esencial antes de que cualquier análisis de los datos pueda ser realizado. El criterio general para reducir la dimensión es el deseo de conservar la mayor parte de la información relevante de los datos originales según algunos criterios óptimos. En reconocimiento de modelo y problemas de clasificación generales, los métodos como PCA, análisis de componentes independientes (ICA) [LEI-04] han sido extensivamente usados para seleccionar las variables de entrada [COH-99]. Estos métodos encuentran una correlación entre el espacio de variables de entrada original a un espacio de variables de entrada con dimensión inferior. En algunas aplicaciones se podrían desear para escoger un subconjunto de variables de entrada originales, la ventaja encontrar una correlación en al que se usen todas las variables de entrada originales, y encontrar este subconjunto de variables de entrada podrían estar en el coste de cálculos de variables de

entrada innecesarias [COH-99]. Las propiedades óptimas del método PCA han atraído una investigación como método para seleccionar las variables de entrada.

2.4.6 Proceso de IVS utilizando PCA

Se considera una transformación lineal de un vector aleatorio $X \in R^n$ con media cero y matriz de covarianza Cov_x a una dimensión de vector aleatorio $Y \in R^q \quad q < n$. La elección de las componentes se realiza de tal forma que la primera recoja la mayor proporción posible de la variabilidad original; la segunda componente debe recoger la máxima variabilidad posible no recogida por el primero, y así sucesivamente. Del total de los componentes se elegirán aquellos que recojan el porcentaje de variabilidad que se considere suficiente. A éstos se les denominará componentes principales.

$$Y = A_q^T \; X \tag{2.49}$$

donde

$$A_q = \begin{bmatrix} A_q^1 \\ A_q^2 \\ \vdots \\ A_q^n \end{bmatrix} \text{ y } X = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \tag{2.50}$$

$A_q^T \; A_q = I_q$, donde I_q es la matriz identidad de orden q. En PCA, A_q es una matriz $q \times n$ cuyas columnas son q vectores propios ortonormales correspondientes a los valores propios más grandes de la matriz de covarianza Cov_x. Se denota la matriz de varianza-covarianza de X entonces:

$$Var(Y_q) = A_n{}^T \Sigma \, A_n \tag{2.51}$$

$$Cov(Y_q, Y_k) = A_n{}^T \Sigma \, A_k \tag{2.52}$$

La seleccionar las variables de entrada por este método se hará por los siguientes pasos:

1- *Obtener los datos* y *restar la media:* tenemos que restar el medio de cada una de las dimensiones de datos. El medio restado es el promedio a través de cada dimensión. De este modo, todos los valores de X tienen restado los medios de los valores de $\overline{X}$ de todos los puntos de datos, y todos los valores de Y tienen $\overline{Y}$ restado de ellos. Este produce un conjunto de datos cuyo medio es cero.

2- *Calcular la matriz de covarianza:* La varianza es otra medida de la extensión de datos en un conjunto de datos. De hecho es casi idéntico a la desviación estándar. La matriz de covarianza se calcula mediante la ecuación (2.52).

3- *Calcular los vectores y valores propios (eigenvectors y eigenvalues) de la matriz de covarianza:* Ya que la matriz de *covarianza* es cuadrada, podemos calcular los vectores propios y valores propios para esta matriz. Estos son bastante importantes, cuando ellos nos dicen la información útil sobre nuestros datos.

4- *Elegir los componentes y la formación de un vector de características:* Aquí hay que comprimir de datos y reducir la dimensionalidad. Si vemos los vectores propios y valores propios de la sección anterior, se puede notar que los valore propios son valores bastante diferentes. De hecho, resulta que los vectores propios con valores propios más altos son los componentes principales del conjunto de datos.

5- *Sacar los nuevos conjuntos de datos:* Una vez que hemos elegido los componentes (vectores propios) que deseamos conservar en nuestros datos, formamos un vector de características, simplemente se transporta el vector y se multiplica con el conjunto de datos ajustados de datos originales transportados.

$$FinalData = RowFeatureVector \times RowDataAdjust$$

Donde *RowFeatureVector* es la matriz de vectores propios en las columnas transportadas de modo que éstos están en las filas, con el más significativo (el de mayor valor propio) en lo más alto, y *RowDataAjust* son los datos medios adaptados y transformados, es decir, que los datos están en cada columna, con cada fila que sostiene una dimensión separada.

6- *Conseguir los datos originales:* Antes de que hagamos esto, recordemos que sólo si se cogieran todos los vectores propios en nuestra transformación vamos a conseguir exactamente los datos originales. Si hemos reducido el número de vectores propios en la transformación final, entonces los datos recuperados han perdido alguna información. Recordemos que la transformación:

$$FinalData = RowFeatureVector \ x \ RowDataAdjust$$

Puede ser invertida para recuperar los datos originales:

$$RowDataAdjust = RowFeatureVector^{-1} \ x \ FinalData$$

Finalmente:

$$RowOrigindlData = (RowFeatureVector^{-1} \ x \ FinalData) + OriginalMean$$

Esta fórmula también se aplica cuando no tenemos todos los vectores propios en el vector de características.

2.4.7 Árboles de decisión

Un árbol de decisión proporciona una forma para desplegar visualmente un problema y después organizar el trabajo de cálculos que deben realizarse. Los árboles de decisión son especialmente útiles cuando deben tomarse una serie de decisiones. Estos árboles de decisión son métodos para aproximar funciones de valores discretos a entrada y a salida. La función de salida viene representada por un árbol de decisión. Los nodos representan atributos de entrada, y los arcos representan los diferentes valores que éstos pueden tomar; las hojas son los valores de salida de la función. En cada nodo se testea un atributo, y se baja por la rama asociada al valor de la instancia, el proceso se repite hasta llegar a una hoja en donde está el resultado. Cada rama del árbol es una restricción sobre los valores expresada como una conjunción. Los árboles se pueden ver como las disyunciones de las restricciones representadas por sus ramas. Los árboles de decisión se aplican cuando: las instancias se representan como parejas atributo-valor. La función objetivo toma valores discretos, las descripciones requieren disyunciones, el conjunto de entrenamiento puede contener errores y las instancias de entrenamiento pueden no tener todos los atributos [YU-03].

La mayoría de los algoritmos para el aprendizaje de un árbol de decisión, utilizan una técnica “greedy”. C4.5 [QUI-93], y su precursor, ID3 [QUI-86], son algoritmos que resumen datos de aprendizaje en la forma de un árbol de decisión. Junto con sistemas que inducen reglas lógicas, los algoritmos de árbol de decisión han demostrado ser muy populares en la práctica. Esto se debe en parte a su robustez y velocidad de ejecución, y al hecho de que las descripciones de concepto explícitas producidas facilitan la interpretabilidad. La Fig.2.12 muestra un árbol de decisión. Los nodos en el árbol

corresponden a las características, y las ramas a sus valores asociados. Las hojas del árbol corresponden a clases. Para clasificar un nuevo caso, simplemente se deben examinar las características probadas en los nodos del árbol y seguir las ramas correspondientes a sus valores observados en el caso. Al alcanzar una hoja, el proceso se termina, y la clase en la hoja es adjudicada al caso.

Haciendo referencia a la figura, la utilización del árbol de decisión para clasificar un ejemplo de un día (soleado, caliente, normal, falso) al principio implica el examen de la característica en la raíz del árbol (Perspectiva).

El valor para la perspectiva en el nuevo caso es soleado, entonces la rama izquierda es seguida. Después el valor para Humedad es evaluado, en este caso el nuevo caso tiene el valor normal, entonces la rama derecha es seguida. Este nos trae a un nodo de hoja y el caso es adjudicado la clase Si.

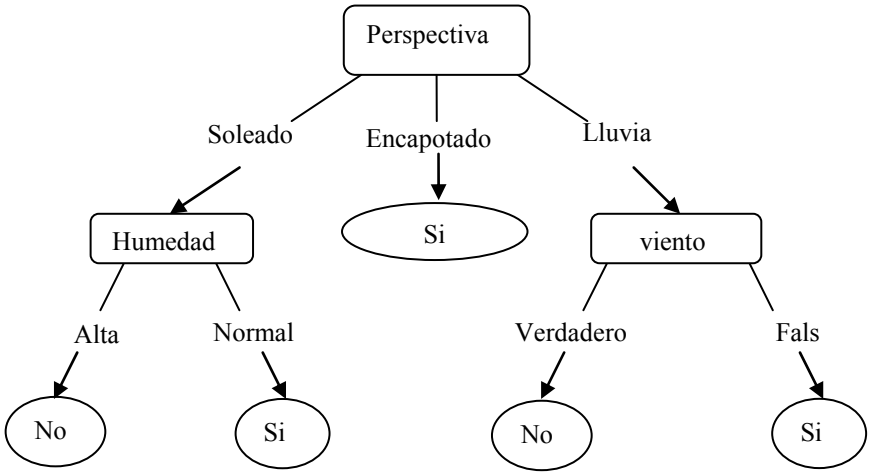


Fig. 2.12 Un árbol de decisión para el conjunto de datos. Las ramas corresponden a los valores de atributos; las hojas indican clasificaciones.

Para construir un árbol de decisión a partir de datos de entrenamiento, C4.5 e ID3 emplean una metodología *greedy* que usa una medida teórica de información como su guía. La elección de un atributo para la raíz del árbol divide los casos de entrenamiento en subconjuntos correspondiente a los valores del atributo. Si la entropía de las etiquetas de clase en estos subconjuntos es menos que la entropía de las etiquetas de clase en el

conjunto de entrenamiento entero, entonces la información ha sido ganada. C4.5 usa el criterio de proporción de ganancia '*gain ratio criterion'* [QUI-86], para seleccionar el atributo para estar en la raíz del árbol. El criterio de proporción de ganancia selecciona, de entre aquellos atributos con una media o mejor ganancia, el atributo que maximiza la proporción de su ganancia dividida en su entropía. El algoritmo es aplicado recurrentemente para formar sub-árboles, terminándose cuando un subconjunto dado contiene casos de sólo una clase.

La diferencia principal entre C4.5 e ID3 es que C4.5 poda sus árboles de decisión. La poda simplifica árboles de decisión y reduce la probabilidad de sobre-ajustar los datos de entrenamiento [QUI-87]. El C4.5 poda usando el límite superior (*upper bound*) de un intervalo de confianza sobre el error de nueva substitución. Un nodo es sustituido por su mejor hoja cuando el error estimado de la hoja está dentro de una desviación estándar del error estimado del nodo. C4.5 ha resultado ser un punto de referencia con el que comparar el rendimiento de los distintos algoritmos de aprendizaje. Sin embargo, quitar la información irrelevante y redundante puede reducir el tamaño de los árboles inducidos por C4.5 [JKP-94]. Los árboles más pequeños son preferidos porque son más fáciles para entender.

2.4.8 Entropía e información mutua

2.4.8.1 Introducción

La selección de variables de entrada posee un papel importante en problemas de aproximación funcional, seleccionando sólo los atributos relevantes que producen interpretación más alta y una aproximación del problema más eficaz y con mejor error de aproximación. Uno de los métodos más populares para seleccionar variables de entrada es, como ya hemos visto, el análisis de componentes principales (PCA) que transforma directamente varias variables posiblemente correlacionadas en un número más pequeño de variables no-correlacionadas que explican la mayor variabilidad posible en los datos. El inconveniente principal de este método consiste en que no es invariante en la transformación y considera solo una relación lineal entre variables [JOL-86]. Una de las contribuciones más importantes para encontrar atributos relevantes uno tras otro, es la utilización de los árboles de decisión, vistos en la sección anterior. Pero estos

métodos tienen algunos problemas de memoria o tiempo de ejecución [BAT-94]. Sin embargo, hay otros métodos que seleccionan entradas a priori basadas sólo en el conjunto de los datos. Entonces la carga computacional sería menor que en casos de modelos dependientes. Los métodos independientes seleccionan un conjunto de entradas optimizando un criterio basado en combinaciones diferentes de entradas. El criterio calcula las dependencias entre cada combinación de entradas y la salida correspondiente, usando los conceptos de previsibilidad, correlación, información mutua u otra estadística [JI-05]. La información mutua es usada como un criterio para seleccionar las mejores variables de entrada (de un conjunto de variables posibles). Es un indicador bueno de la relación entre variables, y ha sido usada como una medida en varios algoritmos de selección de variables de entrada (IVS). Sin embargo, el cálculo de la información mutua es difícil, y la interpretación de un algoritmo de selección de variables depende de la exactitud de la información mutua.

La información mutua (MI) mide las dependencias entre las variables aleatorias para valorar el contenido de información. Fano [FAN-61] ha mostrado que maximizando la información mutua entre los datos transformados y el objetivo deseado, se consigue una inferior probabilidad de error. Esta idea ha inspirado a Battiti en el desarrollo de su método de selección de variables, que denominó MIFS [BAT-94]. El método evalúa información mutua entre variable individuo y las etiquetas de clases, y seleccionan aquellas variables que tienen información mutua máxima con etiquetas de clases menos redundantes. El inconveniente de este método consiste en que esto no tiene en cuenta cómo las variables trabajan juntas [RIP-96].

2.4.8.2 IVS utilizando entropía e información mutua

En el proceso de seleccionar de variables de entrada, se trata de reducir el número de las variables de entrada, excluyendo variables irrelevantes o redundantes, utilizando los datos de entrada como fuente de información. La información mutua (MI) entre dos variables, sea X e Y, es la cantidad de información obtenida de X en la presencia de Y, y viceversa. MI puede ser usado para evaluar las dependencias entre variables arbitrarias, y ha sido aplicado para la selección de variables y la separación ciega de fuentes [YAN-

97]. Para las dos variables arbitrarias de arriba, la información mutua entre ellas vendría dada por la siguiente expresión:

$$I(X,Y) = H(X) - H(X,Y) \tag{2.53}$$

Donde *H (.)* es la entropía, que se define por la siguiente expresión:

$$H(X) = -\sum_{x \in X} P(x) \cdot log\, P(x) \tag{2.54}$$

La información mutua entre esas dos variables vendría dada, por tanto, por:

$$I(X;Y) = \sum_{x \in X} \sum_{y \in Y} P(x,y)\ log \left[\frac{P(x,y)}{P(x)\,P(y)} \right] \tag{2.55}$$

El método *FRn-k* [BAT-94] se basa en este concepto para seleccionar las variables $\hat{d}$ más relevantes de un conjunto de variables d, de tal forma que maximice la información sobre una clase. De la teoría de información entre dos variables aleatorias miden la cantidad de la información común contenida en estas variables [COV-91]. Ji en [JI-05] presentó un método basado en la información mutua, que calcula la información mutua entre todos los conjuntos de entradas posibles y salidas. Se utiliza *Least Squares Support Vector Machines* como modelos no lineales para evitar problemas de mínimos locales. La novedad de este método consiste en su capacidad de estimar MI entre dos variables de cualquier espacio dimensional. La idea básica es estimar *H (.)* de la distancia media a los vecinos más cercanos (sobre todo X_i).

2.5 Conclusiones

Este capítulo ha mostrado el esquema clásico de la arquitectura de las redes de funciones de base radiales RBFNs que son el principal elemento computacional de nuestro trabajo. Las RBFN tienen un carácter local, es decir que cada neurona oculta se especializa en una determinada región del espacio de entrada y construyen aproximación locales dicha zona, que permite un aprendizaje rápido y cualquier cambio del peso de una neurona afecta solo a esta neurona.

En el caso de optimizar los parámetros de las RBFNs se han descrito muy brevemente los tipos de aprendizaje con algoritmos de *clustering* no supervisados (k-medias, k-medias difuso, ELBG) y supervisados (ACE, CFC, CFA) más usados en la literatura

para inicializar los valores de los centros. En el siguiente capítulo se describirán con mayor detalle. También se han visto algoritmos tradicionales para inicializar los valores de los radios de las RBF (Knn, CIV). Para el cálculo exacto de los pesos de una RBFN se han presentado los métodos más utilizados en la literatura para resolver un sistema lineal (la descomposición de Cholesky, la descomposición SVD, y el método OLS), con la comparación entre estos algoritmos para utilizar lo mejor posible.

Otro tipo de redes neuronales han sido explicadas: Las denominadas redes neuronales *wavelet* WNN. Este tipo de redes tiene las mismas características de las redes de funciones de base radial salvo que la función de activación es una función ortogonal denominada *wavelet*. Las WNN son aproximadores universales que consiguen la convergencia más rápida de las RBFNs, son capaces de tratar con el problema de la maldición de la dimensionalidad. Este tipo se utilizará, junto con las RBF, para comprobar el algoritmo propuesto presentado en el capítulo tres.

Se han introducido los mecanismos de minimización del error y la comparación entre ellos. Estos métodos se aplicarán a los parámetros de las RBFNs para minimizar el criterio del error utilizado en este trabajo que es el error cuadrático medio normalizado NRMSE.

Por último, se han presentado algunos métodos tradicionales de selección de las variables de entrada: métodos del tipo *wrapper*, métodos del tipo *filter*, PCA, árboles de decisión e información mutua.

CAPÍTULO 3

NUEVO ALGORITMO DE CLUSTERING PARA APROXIMACIÓN FUNCIONAL: ECFA

La inicialización de los centros de las redes de función de base radial es un tema importante para el problema de aproximación funcional. Cada uno de los centros tiene que estar situado en una zona del espacio de datos de entrada de forma que cubra los datos de esta zona.

Este capítulo presenta un algoritmo de *clustering* diseñado para problemas de aproximación de funciones derivado de los algoritmos *clustering* de aproximación funcional CFA y enhanced LBG. Dicho algoritmo trata de migrar clusters a las zonas del espacio de entrada donde el error en la aproximación es mayor, de modo que se intenta igualar el valor de la distorsión en cada cluster, viniendo ésta determinada principalmente por la distribución del error de aproximación, lo que produce un mejor reparto de los clusters disponibles por el espacio de los datos de entrada.

3.1 Introducción

Los métodos de agrupamiento o *clustering* representan una técnica fundamental en muchos campos de investigación como clasificación [GRI-84], inteligencia artificial, procesamiento de audio y de vídeo [PAL-93], [COS-96], reconocimiento de modelos [FUK-90], visión por computadora, etc. Las técnicas de *clustering* han sido extensamente investigadas durante décadas en varios campos, sobre todo desde principios de los años 60 [JAI-99], [BEZ-84]. Se desarrollaron inicialmente como algoritmos de clasificación [MAC-67], aunque después han sido aplicados en diferentes campos relacionados con la aproximación de funciones utilizando redes neuronales artificiales [CHA-96].

Los algoritmos de *clustering* son métodos que tratan de dividir un conjunto de variables de *n* observaciones en *m* grupos, de modo que los miembros de estos grupos sean diferentes. A cada uno de estos grupos se les denomina *clusters*. El número de grupos *m* puede ser prescrito, o puede ser decidido por el algoritmo. Un algoritmo de *clustering* produce una correlación *C: {1,..., n} a {1,..., m}* asociando a cada ejemplo, un determinado grupo. De este modo, el objetivo del *clustering* es determinar la agrupación intrínseca en un conjunto de datos no etiquetados. ¿Pero cómo decidir qué constituye un *clustering* bueno? No se ha podido demostrar que haya ningún criterio absoluto mejor que sea independiente del objetivo final del *clustering*. Por consiguiente, es el usuario el que debe suministrar este criterio, de tal modo que el resultado del *clustering* satisfaga sus necesidades. Cada algoritmo de *clustering* debe ser capaz de: descubrir clusters de forma aleatoria, tratar con ruido, ser insensible al orden en el que se introducen los objetos, tratar con clases diferentes de entradas, ser utilizables (recursos de memoria y velocidad de ejecución) y estable (pequeños errores en la descripción de los objetos conducen a pequeños cambios de *clustering*).

Los centros $\vec{c}$ de las funciones de base radial (RBFs), se determinan mediante algoritmos de *clustering* que permiten dividir el espacio de los datos de entrada en clases denominadas *clusters*. El número de estos *clusters m* es el número de neuronas ocultas de la red de función de base radial. Cada uno de estos *clusters* debería estar situado en una zona del espacio de los datos de entrada, para activar los ejemplos de

entrada y/o salida. Cualquier cluster que no se active por ningún ejemplo de datos de la entrada, será una sobrecarga y aumentará la complejidad de la red. Es importante resaltar que los algoritmos de *clustering* no suelen optimizar los centros de las RBFNs, sino que sólo forman la fase inicial de estos parámetros. Entonces los algoritmos de *clustering* estudian las características de los datos de entrada y/o salida según su tipo supervisado o no supervisado, que sirve para inicializar los valores de los centros $\vec{c}_j$.

La mayoría de las técnicas de *clustering* fueron diseñadas inicialmente para resolver problemas de clasificación [HAR-75]. El problema de la aproximación de funciones es similar al de clasificación, salvo que las entradas y las salidas son numéricas. El problema consiste en predecir la salida dada por las entradas viendo ejemplos de E/S. La aproximación de funciones se aplicó en muchos campos como la clasificación, reconstrucción de señales, identificación de sistemas, sistemas de control, etc. Para aplicar un método de *clustering* en un problema de aproximación de funciones, debemos intentar aumentar la densidad de *clusters* en las zonas de entrada donde la variedad de la salida objetivo presenta una respuesta más complicada. En concreto, en las zonas donde los *clusters* provocan mayor error, y de este modo podemos conseguir igualdad o casi igualdad en la distorsión en cada *cluster*, y de esta forma podemos conseguir un error pequeño en la aproximación [AWA-05a].

3.2 Algoritmos de *clustering*

Hay muchos modos diferentes de expresar y formular el problema de *clustering*, de modo que los resultados obtenidos y sus interpretaciones dependen del problema de *clustering* para el que fue formulado al principio [FUN-01]. En general, los algoritmos tradicionales de *clustering* tratan de agrupar algunos datos definidos en base a un conjunto de propiedades numéricas, de modo que los datos dentro de un grupo son más similares que los datos en grupos diferentes. Por lo tanto, un algoritmo particular de *clustering* tiene que tener un criterio para medir las semejanzas de los datos.

Una definición matemática del proceso de *clustering*, como se declara en [GRA-98], da un conjunto de vectores $X \in R^{nxd}$, un conjunto de datos que representan un conjunto de puntos *n*. $X = \{x_i : i = 1,...,n\}$ en R^n. El objetivo es la partición del conjunto de datos *X*

en *m* (*m* denota el número total de *clusters*) a grupos $c = \{c_1, ..., c_m\}$, tales que cada dato que pertenece al mismo grupo es diferente de los datos en otros grupos. El resultado del algoritmo es un particionamiento $X \to C$ de los $\vec{x}_i$ en clusters $\vec{c}_j$ $j = 1, ..., m$ [FUN-01].

Los algoritmos de *clustering* desarrollados al principio para problemas de clasificación [HAR-75], han sido más tarde aplicados en el campo de la aproximación funcional utilizando redes neuronales artificiales. En el aprendizaje de RBFNs se utilizan algoritmos de *clustering* para inicializar los valores de los centros de las funciones de base radial [KAR-97]. Dentro del campo del diseño de las RBFNs, estos algoritmos estarían enmarcados dentro de la fase de preproceso de datos de entrada de la red. El objetivo de esta fase es revelar la estructura interna de este conjunto de datos de entrada, agrupándose o identificándose grupos de datos que tienen una o varias características comunes. Una vez completada esta fase, lo normal es insertar una función de base radial RBF en el centro geométrico de cada uno de estos grupos identificados [RIV-03].

La determinación del RBFs para insertar clusters termina con el establecimiento de radios *r* para cada uno de ellos, relacionado con el espacio ocupado por el grupo dentro del cual están localizados. Es importante destacar que estos algoritmos de *clustering* no son una estrategia en el diseño de RBFNs, son una parte de la fase inicial en el diseño de una red de este tipo. Esta primera fase sirve para realizar la primera determinación o la inicialización de los centros $\vec{c}$, y hasta el radio *r* del RBFNs. Así, el proceso del diseño se complementaría con otras estrategias que refinan mucho más los parámetros finales de cada RBF sobre todo, y de la red completa RBFN en general.

Podemos distinguir algoritmos *clustering* principalmente de dos tipos:

- *Algoritmos de clustering no supervisados*: la asignación de grupos en este tipo de algoritmos no tiene en cuenta la información de la salida. Algunos algoritmos importantes de este tipo son: el algoritmo k-medias [DUD-73], el algoritmo k-medias difuso [BEZ-81] y el algoritmo ELBG [RUS-99].

- *Algoritmos de clustering supervisados*: estos interfieren para mejorar los algoritmos de *clustering* no supervisados. Algunos algoritmos importantes de este tipo son: algoritmo de *clustering* para aproximación de funciones (CFA) [GON-02], algoritmo de *clustering* difuso condicional [PED-96], [PED-98], algoritmos de estimación de grupos alternante [RUN-99]. Enhanced CFA [AWA-05a].

Aquí en este capítulo explicamos de forma amplia algunos algoritmos de *clustering*, tanto supervisados como algoritmos no supervisados y la explicación de los algoritmos de *clustering* tradicionales con detalle. Este conocimiento es fundamental para comprender el nuevo algoritmo de *clustering* para aproximación funcional (ECFA) que se propone en este capítulo.

3.3 Algoritmos de *clustering* no supervisados

Un procedimiento de *clustering* no supervisado debe usar los datos de entrada no etiquetados para estimar los valores de los parámetros para el problema de clasificar los datos. El objetivo principal es identificar los clusters que forman naturalmente los datos de entrada. El proceso comienza con valores arbitrarios adjudicados por el software, uno para cada cluster (el número de clusters es introducido como un dato más por parte del usuario). La asignación de grupos en este tipo de algoritmos no tiene en cuenta la información de la salida. Algunos algoritmos importantes de *clustering* de este tipo son el algoritmo de k-medias, el algoritmo de k-medias difuso y el algoritmo de ELBG. Estos se explican a continuación.

3.3.1 Algoritmo de las k-medias (*K-means*)

Es uno de los algoritmos de *clustering* no supervisados propuesto por Duda y Hart en [DUD-73], en su procedimiento sigue un modo simple y fácil de clasificar un conjunto de datos de entrada X por varios clusters (número m de clusters fijado a priori). Esta técnica realiza una partición del conjunto de vectores de entrada X en m grupos y establece la distancia entre los vectores de entrada como la característica de referencia para formar los grupos (ver Fig.3.1). Así, la función objetivo es la función de minimización J que viene dada por la siguiente expresión:

$$J = \sum_{j=1}^{m} \sum_{i=1}^{n} \beta_j(\vec{x}_i) \ \left\| \vec{x}_i - \vec{c}_j \right\|^2 \tag{3.1}$$

donde *n* es el número de vectores de entrada, *m* es el número de clusters y $\|\cdot\|$ la distancia euclídea. La función de pertenencia β de un vector de entrada $\vec{x}_i$ al cluster $\vec{c}_j$ está definida en el rango de la salida *{0,1}* como en la siguiente expresión:

$$\beta_{c_j}(\vec{x}_i) = \begin{cases} 1 & if \left\| \vec{x}_i - \vec{c}_j \right\|^2 < \left\| \vec{x}_i - \vec{c}_l \right\|^2 \quad \forall l \neq j \ , \ l = 1,...,m \\ 0 & en\ otro\ caso \end{cases} \tag{3.2}$$

Esta función de pertenencia produce una partición de Voronoi [GER-92] del conjunto de vectores de entrada de forma:

$$C_j = \bigcup_{j=1}^{m} C_j \ \text{con} \ C_j \cap C_i = \phi \qquad \forall \ j \neq i \tag{3.3}$$

donde C_j se define como en la siguiente expresión:

$$C_j = \left\{ \vec{x}_i \in X : \beta_{c_j}(\vec{x}_i) = 1 \right\} \tag{3.4}$$

Cada vector de entrada sólo se asigna a un cluster dependiendo de la cercanía del vector a este cluster. Una vez que todos los puntos de los datos de entrada han sido asignados a un cluster, se recalculan los clusters de la partición mediante esta ecuación, que asigna a cada cluster el centroide de los vectores de entrada pertenecientes a su cluster asociado como en la expresión siguiente:

$$\vec{c}_j = \frac{\sum_{i=1}^{n} \beta_{c_j}(\vec{x}_i) \ \vec{x}_i}{\sum_{i=1}^{n} \beta_{c_j}(\vec{x}_i)} \tag{3.5}$$

Después de modificar los clusters, se chequea la condición de parada del algoritmo. Para ello se utiliza un valor umbral τ como una medida de parada del algoritmo comparando la distorsión de la partición actual con la distorsión de la partición anterior, y si el cambio es menor del umbral τ se para el algoritmo. El valor de la distorsión ε se calcula por la siguiente expresión:

$$\varepsilon = \sum_{j=1}^{m} \varepsilon_j \tag{3.6}$$

donde ε_j es la distorsión que produce cada cluster, y se define como en la siguiente expresión:

$$\varepsilon_j = \sum_{\vec{x}_j \in C_j} \left\| \vec{x}_i - \vec{c}_j \right\|^2 \tag{3.7}$$

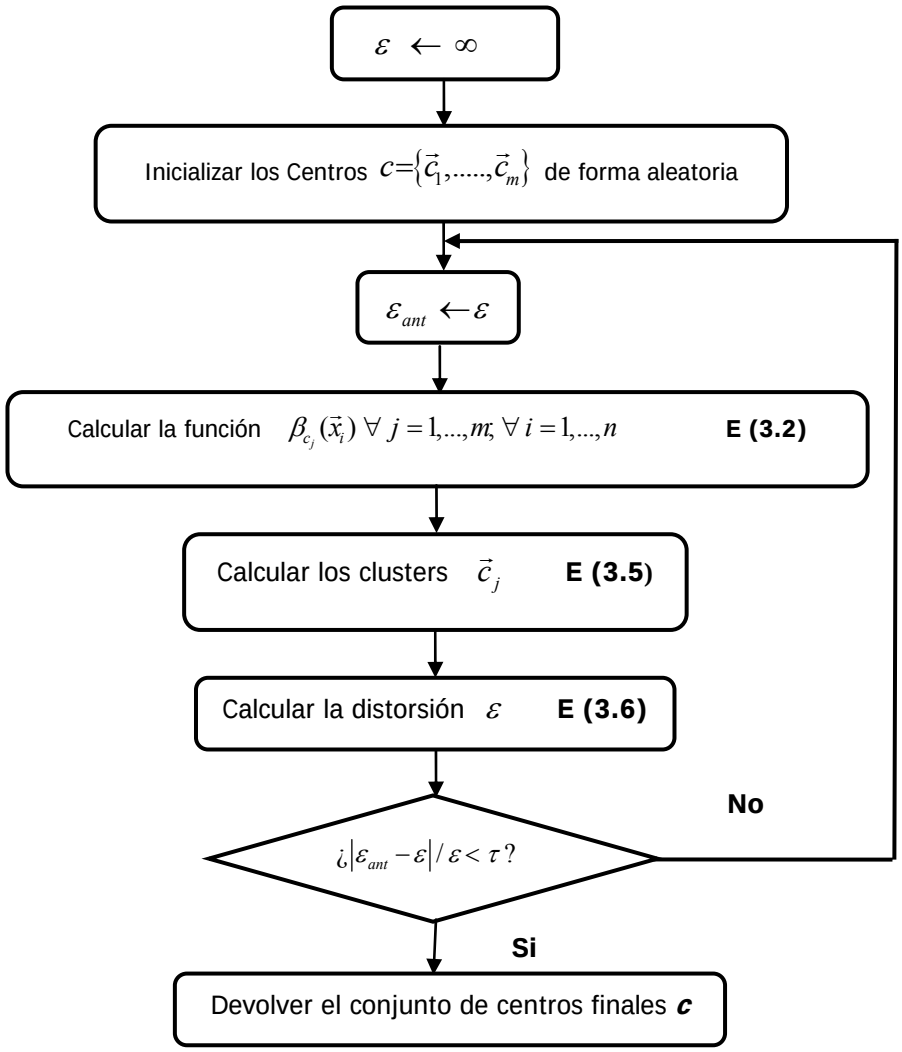


Fig. 3.1 El algoritmo de k-medias

Dada una partición inicial este algoritmo encuentra el mínimo local más cercano en el espacio de todas las posibles particiones.

Este algoritmo muestra la estructura interna del conjunto de datos de entrada, datos similares que están cerca geométricamente pertenecerán al mismo grupo. Mientras que cualquier dato que esté lejano del cluster será difícil que pertenezca a ese cluster. Este método es muy rápido, pero converge a diferentes mínimos locales basados en las inicializaciones, y no detecta clusters degradados o vacíos.

3.3.2 Algoritmo de las k-medias difuso

El algoritmo k-medias difuso [BEZ-84] ha sido aplicado con éxito a una amplia variedad de problemas de *clustering*. Es una generalización del algoritmo anterior en el sentido de que la partición del conjunto de datos de entrada *X* es una partición difusa: la función de pertenencia de este algoritmo difiere de la función de pertenencia del algoritmo k-medias en que cada cluster se considera como un conjunto difuso con una función de pertenencia $\beta_{c_j}(\vec{x}_i)$ con rango de salida [0,1] [BEZ-84]. Ahora, un vector de entrada puede ser asignado a varios clusters con valores de pertenencia distintos, que dependerán de la cercanía del vector de entrada a sus respectivos prototipos. Este algoritmo recibe como entrada el conjunto de datos de entrada *X*, el número de clusters *m,* un valor umbral distorsión τ, y un parámetro $\vartheta \in (1,+\infty)$ que indica el grado de *difusividad* de la partición del conjunto de datos de entrada. Si $\vartheta \to 1+$ la partición se aproxima a la partición como en algoritmo de k-medias, pero si $\vartheta \to +\infty$ todos los puntos pertenecen a todos los clusters. La salida es un conjunto de *m* clusters, cada uno es el centroide de los vectores de entrada que han sido asignados al cluster que representa. La función de pertenencia $\beta_{c_j}(\vec{x}_i)$ $X \to [0,1]$ de un vector de entrada $\vec{x}_i$ a un cluster $\vec{c}_j$ se calcula mediante esta expresión:

$$\beta_j(x_i) = \frac{1}{\sum_{l=1}^{m} \left(\frac{\left\| \vec{x}_i - \vec{c}_j \right\|^2}{\left\| \vec{x}_i - \vec{c}_l \right\|^2} \right)^{1/h-1}} \tag{3.8}$$

donde $h \in (1,\infty)$ determina el grado de *difusividad* de la partición del conjunto de datos de entrada producido por el algoritmo. Si $h \to 1$ la partición está más cerca de una partición clara obtenida con el algoritmo k-medias, mientras que si $h \to \infty$, el algoritmo produce una partición en la cual todos los puntos pertenecen a todos los grupos [KAR-

95]. La nueva posición del cluster $\vec{c}_j$ debe ser adaptada a la nueva función de pertenencia haciendo:

$$C_j = \frac{\sum_{i=1}^{n}\left[\beta_{C_j}(\vec{c}_i)\right]^h \vec{x}_i}{\sum_{i=1}^{n}\left[\beta_{C_j}(\vec{x}_i)\right]^h} \tag{3.9}$$

El valor de la distorsión ε para el algoritmo se calcula como en la ecuación (3.6).

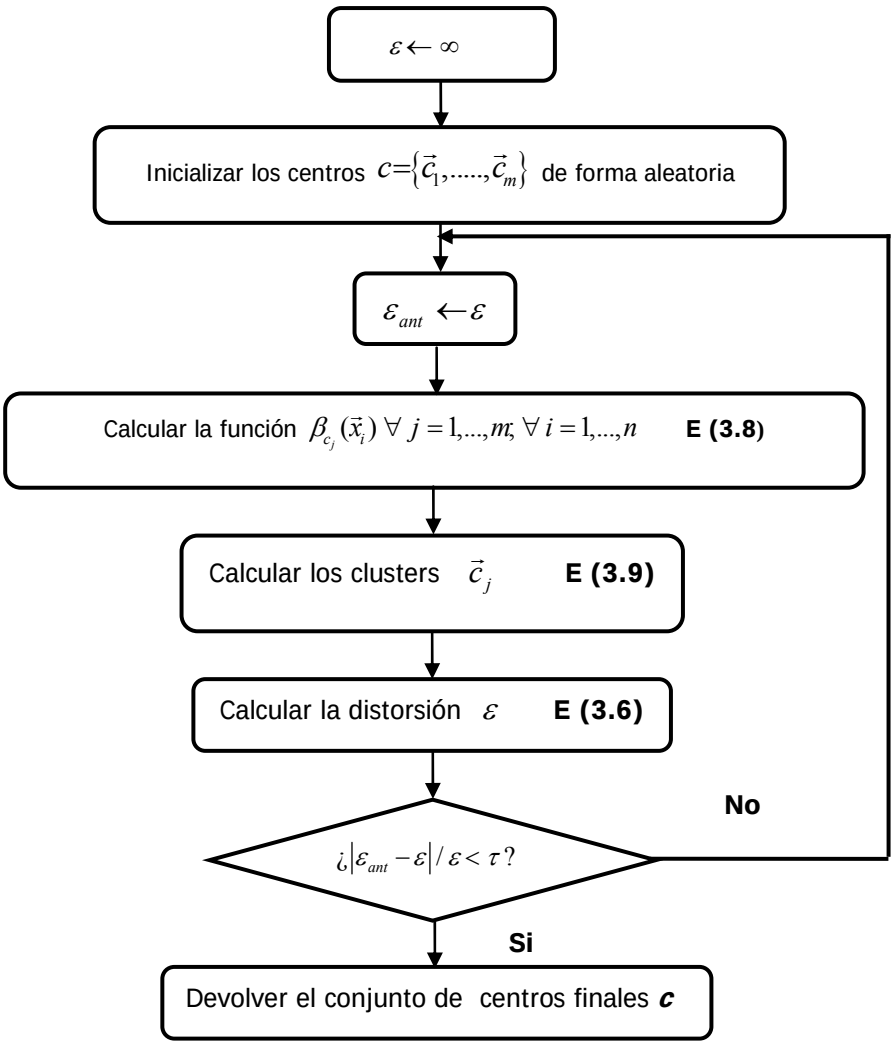


Fig. 3.2 El algoritmo de k-medias difuso.

Se chequea la condición de parada del algoritmo. Para ello se utiliza un valor umbral τ como una medida de parada del algoritmo comparando la distorsión de la partición actual con la distorsión de la partición anterior, y si el cambio es menor del umbral τ se para el algoritmo. De todos modos el algoritmo sufre algunos inconvenientes: pueden

aparecer de nuevo grupos idénticos, y termina en el mínimo local más cercano de la partición. Con respecto al algoritmo anterior, este algoritmo es más costoso en tiempo de ejecución.

3.3.3 Algoritmo *Enhanced LBG*

Este algoritmo de *clustering* pretende solucionar los problemas que presentan los algoritmos anteriores de k-medias y k-medias difuso, los cuales solo producen cambios locales hasta alcanzar el mínimo local más cercano del punto de partida o el punto inicial [RIV-03]. Como muestra la figura 3.3, el cluster $\vec{c}_4$ no esta influenciado por ningún punto de entrada $\vec{x}_i$ (círculos blancos), es un cluster vacío que no se moverá mediante el proceso de *clustering*. Mientras que el resto de los clusters $\vec{c}_1, \vec{c}_2, \vec{c}_3$ están influenciados por puntos de entrada $\vec{x}_i$. Por otro lado, si la zona donde encuentra el cluster $\vec{c}_1$ existen dos clusters se producirían mejores resultados, y el conjunto de datos de entrada queda cubierta de forma más uniforme [GON-01].

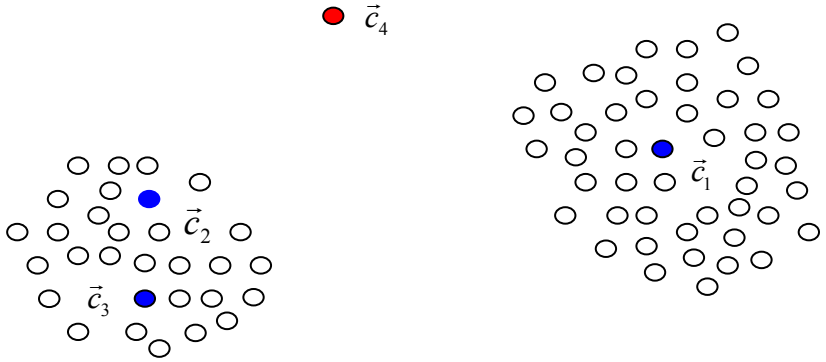


Fig. 3.3 Ejemplos de clusters mal colocados ELBG

Este ejemplo muestra la necesidad de desarrollar una técnica que reconozca las situaciones de los clusters y sea capaz de decidir qué cluster debe moverse y hacia dónde moverse. Para vencer este inconveniente, Russo y Patanè propusieron el algoritmo ELBG [RUS-99]. Este algoritmo pertenece a los grupos de k-medias de vector cuantificado y se deriva directamente del algoritmo LBG [RUS-99]. El objetivo de este algoritmo es obtener una contribución igual a la distorsión total enunciado por [GER-79], y que dice: “Cada cluster hace una contribución igual a la distorsión total en

una cuantización óptima de alta resolución". Este algoritmo se muestra en Fig.3.5. Se introduce formalmente una nueva cantidad denominada utilidad de un cluster. Esto permite el tratamiento de los inconvenientes descritos anteriormente de un punto de vista único. La utilidad de un cluster $\vec{c}_j$ se define como en la siguiente expresión:

$$U_j = \frac{\varepsilon_j}{\varepsilon_{mean}} \qquad j = 1,...m \tag{3.10}$$

donde ε_{mean} representa la distorsión media de la partición, se define como en [RUS-99]:

$$J_{mean} = \frac{1}{k}\sum_{j=1}^{m} \varepsilon_j \tag{3.11}$$

El teorema de la distorsión total [GER-79] implica que, en una partición óptima del conjunto de los datos de entrada, todos los clusters tendrían una utilidad igual a 1. El algoritmo ELBG trata de llegar a esta condición mediante un proceso de migración de clusters, clusters con utilidad menor que 1 migran a zonas de clusters con utilidad mayor que 1. Los pasos de la migración vienen dados de esta forma:

a. Elegir un cluster que tiene la utilidad menor que 1.
b. Elegir otro cluster con la utilidad mayor que 1 de forma aleatoria, los clusters con mayor utilidad tendrán más probabilidad para ser elegidos.
c. Desplazar el cluster con menor utilidad a la zona del cluster con mayor utilidad, realizar ajustes locales usando la diagonal de los datos que pertenecen al cluster con mayor utilidad y dividir estos datos a los dos clusters utilizando el algoritmo k-medias con k = 2.
d. Calcular la distorsión de la nueva partición.
e. Si la distorsión $\varepsilon_{despt} \leq \varepsilon_{ant}$ se acepta el desplazamiento, si no se rechaza el desplazamiento.

En la Fig. 3.3, según el valor de las utilidades, las ecuaciones anteriores determinarían que la utilidad del cluster $\vec{c}_4$ es igual a cero (cluster mal colocado), y los clusters $\vec{c}_2$, $\vec{c}_3$ tienen utilidad menor que 1, pero el cluster $\vec{c}_1$ tendrá utilidad mayor que 1. Con estos valores de utilidad de los clusters y según el proceso de migración, el cluster $\vec{c}_4$ y

alguno de los clusters $\vec{c}_2, \vec{c}_3$ se mueven a la zona del cluster $\vec{c}_1$. Con este movimiento, sus valores pueden acercarse a un valor de utilidad igual a 1.

Este intercambio inteligente de clusters, trata de mover todos los clusters de utilidad baja y acercarlos a los que tienen una utilidad alta, como muestra la Fig. 3.4.

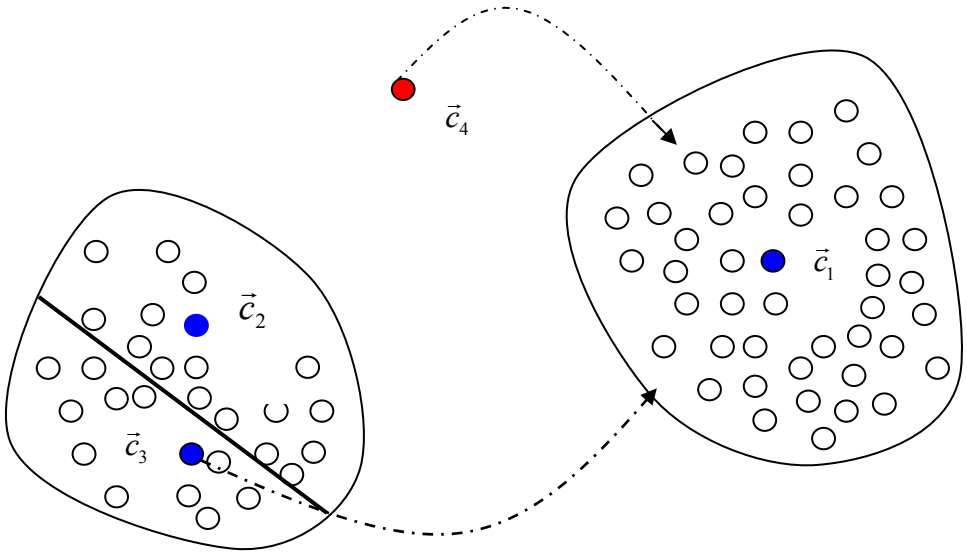


Fig. 3.4 Situación de los clusters antes de la migración. ELBG

Si el cluster $\vec{c}_4$ o el cluster $\vec{c}_3$ migra a la zona del cluster $\vec{c}_1$, hay que hacer unos ajustes locales al cluster $\vec{c}_1$, que va a recibir el nuevo cluster $\vec{c}_4$ o $\vec{c}_3$, los vectores que pertenecían al cluster $\vec{c}_1$ tienen que ser asignados de forma dependiente a la cercanía de cada punto al cluster $\vec{c}_1$ y al cluster que ha migrado ($\vec{c}_4$ o $\vec{c}_3$). La posición donde se coloca el cluster migrado, $\vec{c}_4$ o $\vec{c}_3$, se calcula utilizando el paralelepípedo que contiene la zona del cluster $\vec{c}_1$, alojando $\vec{c}_1$ y a $\vec{c}_4$ o $\vec{c}_3$ en su diagonal principal. Por eso la diagonal se divide en tres segmentos de forma que el segmento central tenga una longitud igual al doble de la longitud de los otros dos segmentos y se calculan los clusters en los extremos del segmento central. Después de realizar esta tarea de inicialización los clusters $\vec{c}_1$ y $\vec{c}_4$ o $\vec{c}_3$ se ajustan aplicando el algoritmo de k-medias con $k = 2$ de forma local a los datos de la zona del cluster $\vec{c}_1$, dividiéndolos a los

clusters $\vec{c}_1$ y $\vec{c}_4$ o $\vec{c}_3$. Después de este proceso se utiliza la actualización para actualizar los clusters y para que los datos del cluster migrado pertenezcan al cluster más cercano.

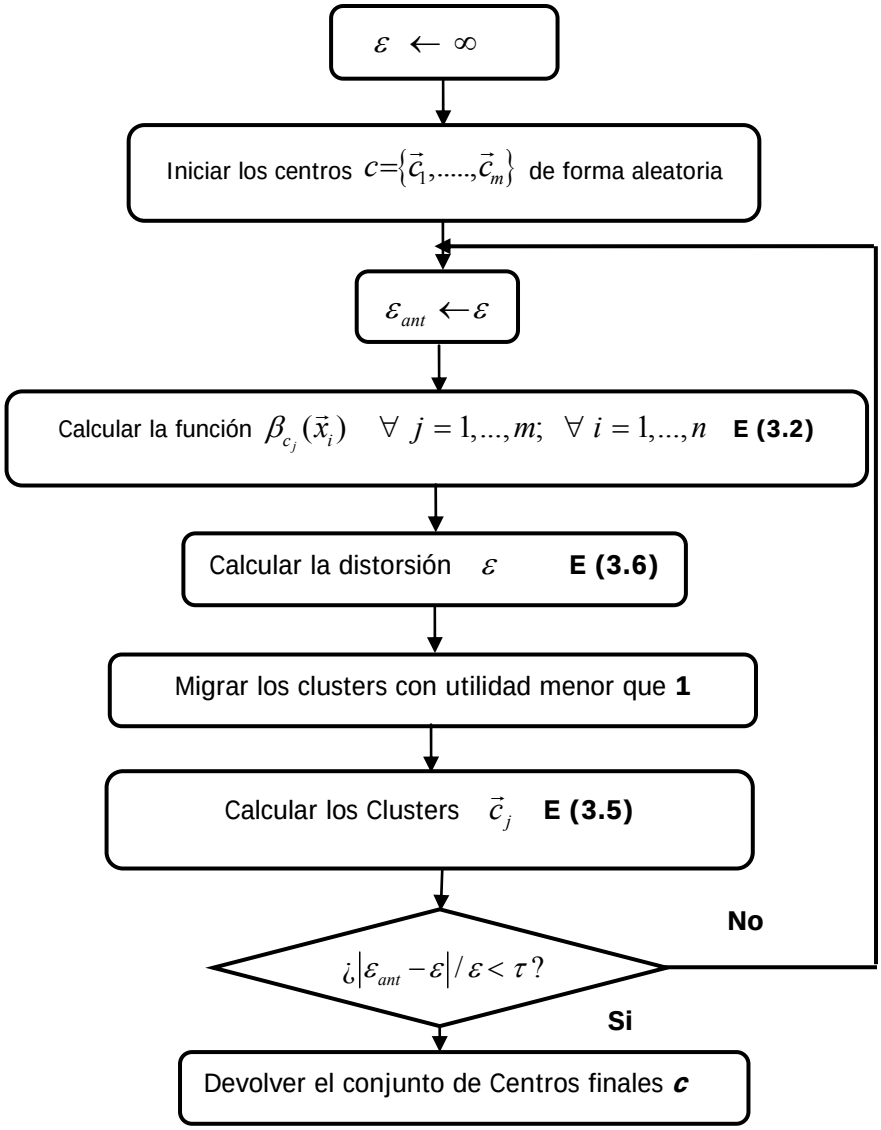


Fig. 3.5 El algoritmo ELBG

Una vez terminado el proceso de migración, se compara la distorsión ε de los clusters antes del desplazamiento de forma:

$$\varepsilon_{ant} = \varepsilon_1 + \varepsilon_2 + \varepsilon_3 + \varepsilon_4 \tag{3.12}$$

con la distorsión después del desplazamiento de esta forma:

$$\varepsilon_{desp} = \hat{\varepsilon}_1 + \hat{\varepsilon}_2 + \hat{\varepsilon}_3 + \hat{\varepsilon}_4 \quad (3.13)$$

Si resulta que $\varepsilon_{despt} \leq \varepsilon_{ant}$ la migración será aceptada, si no será rechazada. Esta modificación permite que el algoritmo se escape de mínimos locales y obtenga una asignación de clusters independiente de la configuración inicial, y con esto se resolverían, en principio, los problemas que sufren los algoritmos de las k-medias y k-medias difuso.

3.4 Algoritmos de *clustering* Supervisados

Estos interfieren para mejorar los algoritmos de *clustering* no supervisado, ya que tienen en cuenta la salida de problema. Algunos algoritmos importantes de este tipo son: el algoritmo de estimación de grupos alternante (ACE) [RUN-99], el algoritmo de *clustering* difuso condicional [PED-96], [PED-98] y el algoritmo de *clustering* para aproximación de funciones (CFA) [GON-02].

3.4.1 Algoritmo de estimación de grupos alternante (ACE)

El algoritmo de estimación de grupos alternante ACE [RUN-99], es un algoritmo de *clustering* supervisado, es decir, que consiste en determinar los conjuntos de la entrada teniendo en cuenta el conjunto de salida. ACE utiliza una arquitectura de iteración alternante, y las funciones de *clustering* y pertenencia son seleccionadas directamente por el usuario. Prácticamente cada modelo puede ser realizado como un caso de ACE. El usuario, sin embargo, podría estar interesado en la elección de formas de función que considere más útiles para una aplicación dada. ACE abandona el modelo de función objetivo y presenta un modelo más general, que es definido por la arquitectura del algoritmo de optimización alternante y por las ecuaciones especificadas por el usuario para actualizar la matriz de la partición y los centros de los clusters. Los casos de ACE pueden o no pueden optimizar una función particular objetivo. Cuando el usuario selecciona ecuaciones de actualización que no corresponden a un modelo de función objetivo, como resultado del ACE, los centros de los clusters son estimados por particiones de actualización alternante y prototipos. Por lo tanto a este modelo se le llama estimación de grupos alternante (ACE).

EL algoritmo ACE es una técnica de *clustering* para *clustering* difuso, donde el modelo de función objetivo es desechado y sustituido por un modelo más general definido por la arquitectura del algoritmo de optimización alternante conducido por condiciones necesarias para extremos locales, y por el usuario especificando ecuaciones para actualizar los parámetros desconocidos, que pueden o no ser derivados de un criterio de *clustering*. El ACE puede ser usado para poner en práctica algoritmos de *clustering* conocidos como k-medias y k-medias difuso, o diseñar nuevas técnicas de *clustering* para un problema particular.

3.4.2 Algoritmo de *clustering* difuso condicional (CFC)

El objetivo principal de este algoritmo es desarrollar clusters que conserven la homogeneidad de los modelos de *clustering*, tener en cuanta sus semejanzas en el espacio de entrada así como sus valores respectivos asumidos en el espacio de salida [PED-98]. La base del algoritmo CFC está en el algoritmo de k-medias difuso, siendo una extensión de éste. La parte condicional del mecanismo de agrupación reside en las variables de salida $y_i,...,y_n$ de las correspondientes variables de entrada. Es decir, una variable de salida y_i de un vector de entrada $\vec{x}_i$ describe el nivel en el cual este vector interviene en la construcción del grupo. Esto realiza una definición lingüística en el espacio de la salida, que se va a corresponder con un conjunto difuso D, $D : \Re \rightarrow [0,1]$. De esta forma $y_i = D(f_i)$, donde f_i es la salida de la red para el vector de la entrada, que expresa el grado de pertenencia de la salida y_i al conjunto difuso D.

La forma que la salida y_i se asocia a las funciones de pertenencia del vector de entrada $\vec{x}_i$, $\beta_{c_j}(\vec{x}_i)$,...., $\beta_{c_k}(\vec{x}_i)$ no es la única función de pertenencia [RIV-03], si se tiene que cumplir que la salida de la red f_i se distribuya aditivamente a lo largo de las entradas de la columna *i* de la matriz de partición que viene dada por esta forma:

$$\sum_{i=1}^{m} \beta_{c_j}(\vec{x}_i) \quad = \quad f_i \qquad i = 1,..., n \tag{3.14}$$

De este modo, si el vector de entrada $\vec{x}_i$, es poco significativo en el contexto del conjunto difuso *D*, entonces $D(f_i) = 0$, y el vector de la entrada $\vec{x}_i$ será excluido del proceso de *clustering* cuando $\beta_{c_j}(\vec{x}_i) = 0$ para cualquier valor de *j*.

Por otro lado, si $D(f_i) = 1$ el vector de entrada $\vec{x}_i$ contribuirá de forma máxima al proceso de *clustering*, y la función de pertenencia como en la siguiente expresión:

$$\beta_j(x_i) = \frac{f_i}{\sum_{l=1}^{m} \left(\frac{\left\| \vec{x}_i - \vec{c}_j \right\|^2}{\left\| \vec{x}_i - \vec{c}_l \right\|^2} \right)^{1/h-1}} \tag{3.15}$$

donde $h \in (1, \infty)$ determina el grado de *difusividad* de la partición del conjunto de datos de entrada producido por el algoritmo.

3.4.3 Algoritmo de *clustering* para aproximación de función (CFA)

La mayoría de los métodos presentados anteriormente han sido diseñados para la resolución de problemas de reconocimiento de modelos en problemas de clasificación. Esto significa que estos modelos no se adaptan bien al problema de aproximación funcional. La aproximación de funciones y la clasificación tienen ciertas diferencias, algunas de estas diferencias son [GON-01]:

- La variable de salida en clasificación puede tomar valores en un intervalo de etiquetas válidas definido a priori, pero en aproximación funcional la función de salida puede tomar cualquier valor dentro de un intervalo de números reales.

- La producción de una salida diferente de la salida objetivo es aceptable en aproximación funcional, siempre que esté cerca con un valor de error pequeño. Mientras que en problemas de clasificación no suele haber definido un operador de distancia entre las clases, ni siquiera tienen que estar relacionadas entre sí.

- La aproximación persigue una interpolación de valores que no se han presentado durante el entrenamiento. En el problema de clasificación la interpolación no tiene sentido.

El algoritmo de *clustering* para aproximación de funciones CFA [GON-02], analiza la variabilidad de la salida de la función objetivo durante el proceso de *clustering* y aumenta el número de clusters en aquellas zonas de entrada donde la función objetivo es más variable, mejorando el desacuerdo explicado por el aproximador. Este cambio del comportamiento del algoritmo de *clustering* mejora la interpretación del sistema aproximador obtenido, comparado con otros modelos utilizados para la clasificación tradicional [GON-02]. CFA proporciona un método que incorpora una serie de características para ayudar en problemas de aproximación funcional. El objetivo final no debe ser obtener un modelo de aproximación exacto cuando esto se puede obtener mediante métodos matemáticos que usan la minimización. Este algoritmo puede obtener soluciones prácticamente ideales si comienza con parámetros iniciales adecuados. De este modo, el objetivo de este algoritmo será obtener la mejor configuración posible, siendo ésta, aquélla que conduzca, tras un método de minimización local, a la consecución del mejor óptimo global.

Este algoritmo recibe como entrada: un conjunto de vectores de entrada $X = \{\vec{x}_1, ... \vec{x}_n\}$, la salida esperada $y_i = F(\vec{x}_i)$ para cada combinación de entrada *X,* el número de clusters y un valor umbral de distorsión, para establecer la condición de finalización del algoritmo. La salida del algoritmo es un conjunto de *m* clusters $(\vec{c}_1, \ldots, \vec{c}_m)$, que representan los grupos en los que debemos situar las funciones de base radial para aproximar los vectores de entrada *X*.

CFA revela la estructura de los datos de entrada, de forma que se preserve la homogeneidad de los datos de salida de los ejemplos que pertenecen al mismo cluster. Por eso CFA incorpora un conjunto $O = \{o_1, ..., o_m\}$, que representa la estimación de la salida esperada para los puntos que pertenecen al mismo cluster. Este valor de o_j se obtiene como una media ponderada de las salidas de los datos de entrenamiento que

pertenecen al cluster j-ésimo. El algoritmo CFA trata de minimizar una función de distorsión ε que se define de esta forma:

$$\varepsilon = \frac{\sum_{j=1}^{m} \sum_{\vec{x}_i \in C_j}^{n} \left\| \vec{x}_i - \vec{c}_j \right\|^2 w_{ij}}{\sum_{j=1}^{m} \sum_{\vec{x}_i \in C_j}^{n} w_{ij}} \tag{3.16}$$

La organización básica del algoritmo CFA se muestra en Fig. 3.6. El algoritmo CFA empieza generar los centros iniciales $C = \{\vec{c}_1, ..., \vec{c}_m\}$ y sus salidas estimadas $O = \{o_1, ..., o_m\}$ de forma aleatoria. La partición de los vectores de entrada se realiza mediante las ecuaciones (3.2), (3.3) y (3.4).

Un paso fundamental de algoritmo CFA es la actualización de los centros de los clusters $(\vec{c}_1, \ldots, \vec{c}_m)$ y sus salidas estimadas $O = \{o_1, ..., o_m\}$. Esta actualización se lleva a cabo mediante un proceso iterativo que calcula los valores del centro de cada cluster $\vec{c}_j$ y su salida estimada o_j como una media ponderada de los vectores de entrenamiento pertenecientes al cluster j-ésimo y su salida. Las expresiones utilizadas para actualizar $\vec{c}_j$ y o_j son:

$$\vec{c}_j = \frac{\sum_{\vec{x}_i \in C_j} \vec{x}_i \psi_{ij}}{\sum_{\vec{x}_i \in C_j} \psi_{ij}} \qquad i = 1,...,n, \quad j = 1,...m \tag{3.17}$$

$$o_j = \frac{\sum_{\vec{x}_i \in C_j} f(\vec{x}_i) \psi_{ij}}{\sum_{\vec{x}_i \in C_j} \psi_{ij}} \quad i = 1,...,n, \quad j = 1,..., m \tag{3.18}$$

$$\psi_{ij} = \frac{\left| f(\vec{x}_i) - o_j \right|}{\max_{l=1}^{n} \{f(\vec{x}_l)\} - \min_{l=1}^{n} \{f(\vec{x}_l)\}} + \mu_{\min} \tag{3.19}$$

donde $\mu_{\min} > 0$, ψ_{ij} pondera la influencia de cada ejemplo de entrenamiento $\vec{x}_i$ en la posición final del cluster C_j. La influencia de ψ_{ij} será mayor en el resultado final cuando la distancia entre la salida esperada para $\vec{x}_i$ y la salida estimada para uno de los

clusters C_j sea mayor. La ecuación que calcula ψ_{ij}, que es parte fundamental en el algoritmo CFA, tiene varias partes: una primera que calcula la suma de la distancia normalizada (en el intervalo [0,1]) entre la salida del sistema $f(\vec{x}_i)$ y la salida esperada del sistema o_j, y una segunda parte $\mu_{\min}$ que es un umbral de mínima contribución, así cuando no exista controversia, ψ_{ij} tomará el valor de este umbral $\mu_{\min}$. Cuando el valor de $\mu_{\min}$ decrece, el algoritmo CFA aumenta el número de clusters en la zona del espacio de entrada donde la función objetivo es más variable. Por otro lado si $\mu_{\min}$ crece, el algoritmo CFA empieza a dar más importancia a la distancia entre los vectores de entrada $\vec{x}_i$ y los centros $\vec{c}_j$, de este modo la influencia del valor de $\mu_{\min}$ es importante en el algoritmo CFA. El paso de actualización de los clusters depende para su finalización del valor de la distorsión ε en la ecuación (3.16). Este paso empieza calculando el valor de la distorsión en cada iteración, actualizando posteriormente los centros $\vec{c}_j$, según la ecuación (3.17). Después actualiza las salidas estimadas o_j según la ecuación (3.18) y ψ_{ij} según (3.19). Después para parar el proceso de actualización, se vuelve a calcular el valor de la distorsión y el algoritmo finaliza cuando llega a un valor menor del valor umbral.

Una vez se cumple el proceso de actualización, el algoritmo aplica el proceso de migración, este proceso es aleatorio para evitar la convergencia hacia mínimos locales. El proceso trata de migrar clusters de zonas del espacio de entrada donde la función objetivo sea más estable hacia zonas donde la función objetivo sea más variable. Para esta tarea se utiliza el concepto de la utilidad de los clusters, exactamente como en el algoritmo ELBG [RUS-99]. Se calcula la utilidad mediante las ecuaciones (3.10), (3.11) en la sección 3.3.3. El último paso es calcular la distorsión ε usando la ecuación (3.16), si $\left|\varepsilon_{ant} - \varepsilon_{act}\right| / \varepsilon_{act} < \tau$ se para el algoritmo. Si no, se repite el proceso. τ es un valor umbral que para el algoritmo cuando llega a su valor.

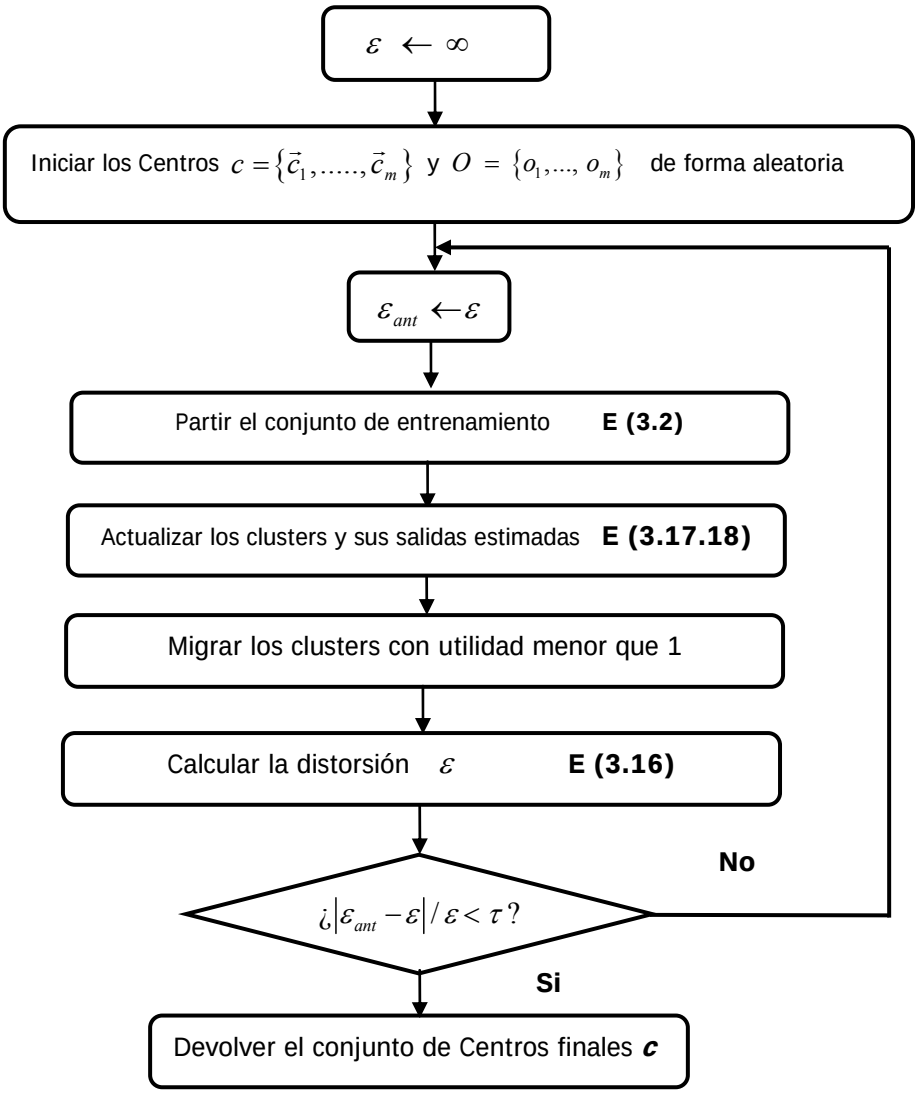


Fig. 3.6 El algoritmo CFA

El algoritmo CFA es una versión ponderada del algoritmo k-medias donde cada vector de aprendizaje tiene un peso según un criterio de variabilidad de salida. CFA alcanza el estado de convergencia cuando el movimiento del cluster es insignificante, como en el de k-medias. El objetivo para ambas técnicas es el mismo: alcanzar una configuración en la cual cada uno de los clusters hace una contribución igual a la distorsión total. La diferencia es que la ecuación de distorsión usada para k-medias produce una distribución del cluster según la densidad de ejemplos en el espacio de entrada y la nueva ecuación de distorsión diseñada para CFA concentra más clusters en aquellas zonas de entrada donde la salida es más variable [GON-02]. El CFA proporciona unos

mejores resultados en el problema de aproximación funcional, pero necesita un tiempo grande de ejecución para converger, especialmente en la parte de actualización de los clusters y no escala bien a un número grande de ejemplos de datos.

3.5 Nuevo algoritmo de *clustering* para aproximación funcional: ECFA

Varios algoritmos de *clustering* han sido utilizados para solucionar el problema de inicialización de los centros de las RBFNs en modelos de aproximación funcional. La mayoría de los algoritmos de *clustering* fueron en principio diseñados para problemas de clasificación y reconocimiento de patrones. La salida en ambos problemas es diferente, siendo finita y continua para la aproximación de funciones, pero infinita y discreta para la clasificación [MAC-67]. La mayor parte de algoritmos de *clustering* sufren dos inconvenientes principales cuando tratan de resolver problemas de aproximación de funciones: primero, los algoritmos de *clustering* son lentos y no escalan a un número grande de vectores de datos, y segundo, convergen a diferentes mínimos locales basados en las inicializaciones. Este algoritmo ECFA trata de resolver estos inconvenientes.

El algoritmo propuesto denominado ECFA [AWA-05a] está basado en los algoritmos CFA [GON-02] y ELBG [RUS-99]. ECFA trata de calcular el error provocado por cada cluster teniendo en cuenta el verdadero error cometido por la red para cada una de las salidas objetivo de que disponemos de la propia función que queremos aproximar, aumentando el número de clusters en las zonas donde éstos provocan el mayor error de aproximación. Para ello, utilizamos la función del error E entre la salida de la RBFN $f(\vec{x}_i)$ y la salida objetivo y. Este error es fundamental en el cálculo de la distorsión ε que se pretende minimizar. El error E se utiliza como la función ψ en el algoritmo CFA.

Si un algoritmo de *clustering* puede obtener la igualación del error o la distorsión total de cada cluster [GER-79], esto significa que cada cluster ha encontrado su lugar correcto en el espacio de entrada y la distribución de los clusters es la óptima.

Nunca debemos olvidar, que el objetivo ultimo de nuestro método de *clustering* es escoger aquella configuración inicial en un problema de aproximación de funciones a partir de la cual podamos encontrar el mínimo global. Con ese fin, nuestro algoritmo de *clustering* aumenta la densidad de clusters $\vec{c}_j$ en las zonas de los datos de entrada donde el cluster produce mayor error teniendo en cuenta la variabilidad de la salida objetivo, y migra clusters con menor error a zonas de clusters con mayor error. De este modo, intentamos obtener la homogeneidad del error en todas las zonas de los clusters en el espacio de entrada, sin que ello signifique que dicha configuración sea la que minimice el error total. Con esta igualdad del error la distorsión será casi igual para todos los clusters en el espacio de los datos de entrada, es decir, que cada cluster está colocado en su lugar adecuado y cubre su zona de los datos de entrada que pertenecen a ese cluster.

El algoritmo propuesto ECFA [AWA-05a] es rápido, con una complejidad computacional menor que el algoritmo CFA. El algoritmo es una aproximación de una estrategia global para obtener los clusters. El método también es simple de implementar. El algoritmo comienza la inicialización de los cluster utilizando el algoritmo de k-medias, aunque cualquier otro algoritmo se podría haber utilizado para obtener dicha configuración inicial. En el siguiente paso se hará un proceso de desplazamiento local de los clusters, este paso es un proceso iterativo que trata de hacer ajustes locales de los clusters, por eso se usa dentro de este proceso, otro de partición de los datos de entrada que depende de la distancia entre cada vector de entrada y cada cluster. En este proceso también se calcula el error de la red de funciones de base radial RBFNs, y se actualizan los valores centrales de los clusters usando la ecuación parecida a la (3.17). Las expresiones exactas se presentarán más adelante. El desplazamiento local finaliza cuando la mejora en el valor de la distorsión ε alcanza un valor por debajo de un determinado umbral τ.

El proceso de migración es un poco diferente de la migración en los algoritmos CFA y ELBG, es más rápida en alcanzar la solución. Se utiliza el concepto de utilidad U_j como en los algoritmos CFA y ELBG como medida del error de cada cluster E_j como en las ecuaciones (3.10) y (3.11). Los clusters que tengan utilidad menor que 1 migran a zonas de clusters que tienen utilidad mayor que 1. Una vez ha migrado un cluster a la

zona de otro, se utiliza el algoritmo de las k-medias con k=2 para dividir los datos de esta zona entre esos dos clusters [AWA-05a].

Como consecuencia de estos procesos, el algoritmo puede converger a soluciones finales, para un mismo grado de aproximación funcional, usando un número de clusters menor que los otros algoritmos aplicados al mismo problema, y con un tiempo de ejecución menor. Resultados y comparaciones entre el algoritmo propuesto y el algoritmo CFA se presentan en capítulo 4. Se puede utilizar este algoritmo para encontrar el número mínimo de centros que satisfacen un problema dado de aproximación de funciones. La organización básica del algoritmo ECFA se muestra en la Fig 3.7. En esta figura se muestra la descripción general del algoritmo ECFA. Los pasos del algoritmo ECFA son: inicialización de los clusters, el proceso del desplazamiento local de los clusters, y la migración de los clusters con utilidad menor que 1 a zonas de clusters con utilidad mayor que 1. Estos pasos se explican con detalle a continuación.

3.5.1 Inicialización de los clusters mediante k-medias

La mayoría de los algoritmos de *clustering* utilizan la inicialización aleatoria para los clusters, en el algoritmo ECFA se usa el algoritmo k-medias para inicializar los valores de los clusters. La utilización del algoritmo k-medias no afecta al algoritmo en el tiempo de ejecución porque la inicialización se realiza sólo una vez. El algoritmo de las k-medias ya fue descrito en detalle en la sección 3.3.1.

3.5.2 Desplazamiento local de los clusters

El desplazamiento local se usa para mover los clusters de forma local, es decir, en este proceso los clusters sólo se desplazan en unos rangos muy pequeños con el único fin de minimizar localmente la distorsión total. Esta parte del algoritmo es muy importante porque en esta parte se calculan variables que son fundamentales para los procesos del algoritmo y porque se ejecuta un gran número de veces: el desplazamiento local se usa antes y después del proceso de migración. El proceso de desplazamiento local se muestra en Fig 3.8.

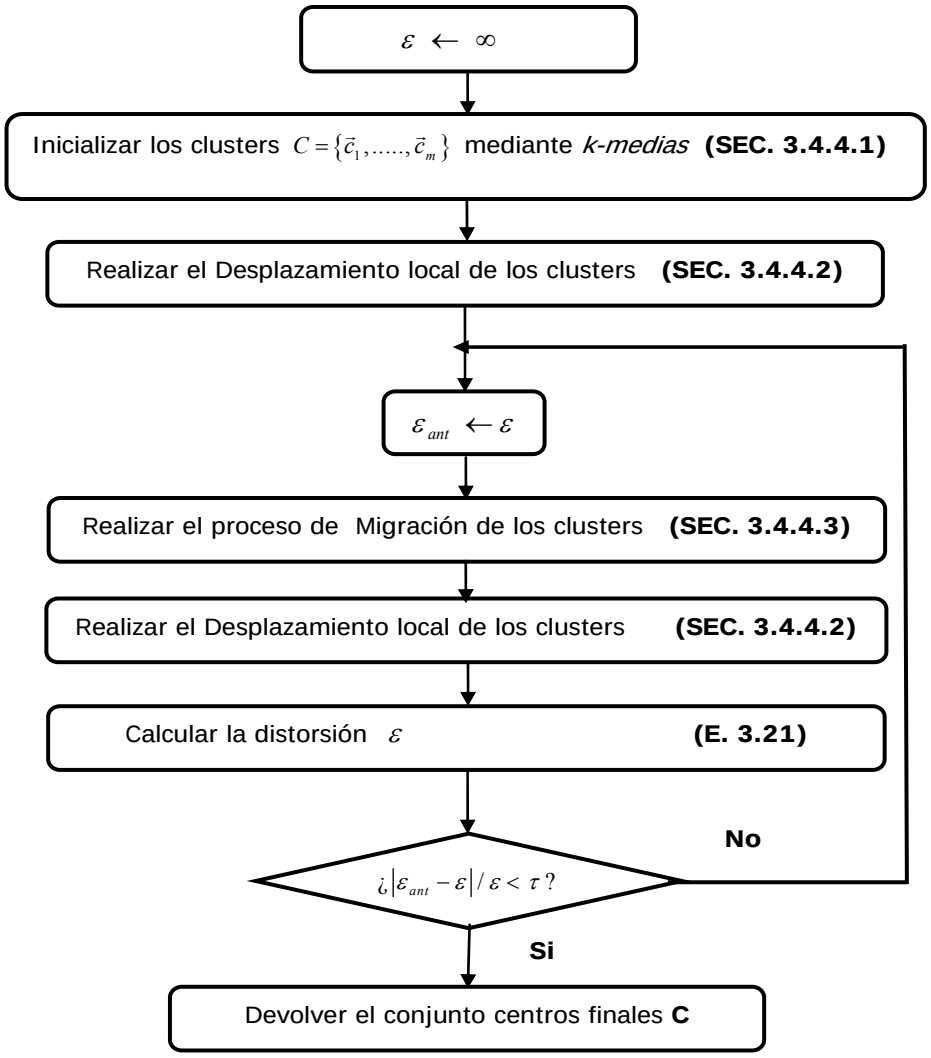


Fig. 3.7 El algoritmo ECFA

Los pasos descritos en dicha figura se explican con detalle a continuación.

3.5.2.1 Partición de los vectores de entrenamiento

La partición de los vectores de entrenamiento se realiza mediante la aplicación de la ecuación de la función de pertenencia (3.2), igual que en el algoritmo de las k-medias.

$$\beta_{c_j}(\vec{x}_i) = \begin{cases} 1 & if \left\|\vec{x}_i - \vec{c}_j\right\|^2 < \left\|\vec{x}_i - \vec{c}_l\right\|^2 \quad \forall l \neq j \ , \ l = 1,...,m \\ 0 & en\ otro\ caso \end{cases} \tag{3.20}$$

Esta función de pertenencia produce una partición de Voronoi [GER-92] del conjunto de los vectores de entrada, que se calcula mediante las ecuaciones (3.3) y (3.4), viniendo cada cluster de dicha partición dado por:

$$C_j = \left\{ \vec{x}_i \in X : \beta_{c_j}(\vec{x}_i) = 1 \right\} \tag{3.21}$$

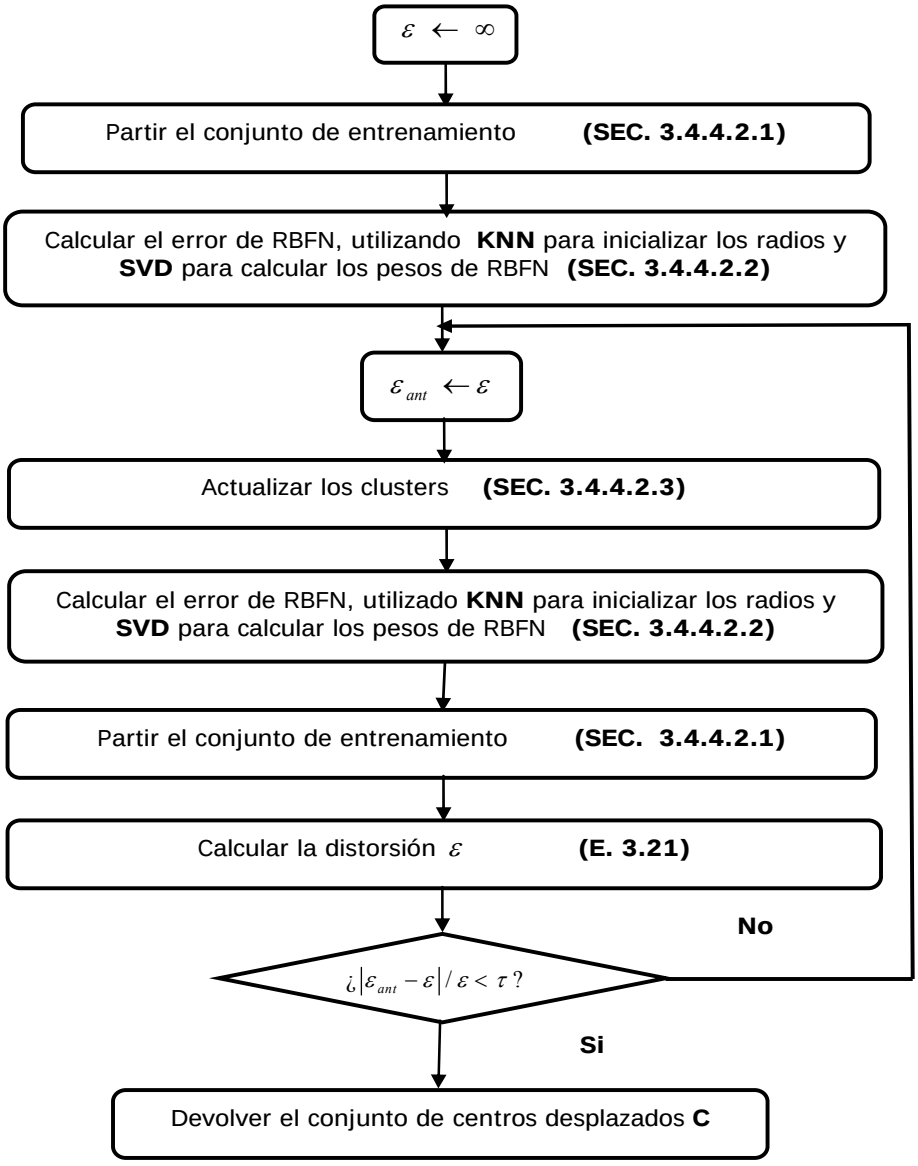


Fig. 3.8 Desplazamiento local de los clusters

3.5.2.2 Cálculo de la distribución de los errores de aproximación

El cálculo del error E es un parte fundamental en el algoritmo ECFA. Este error es el error existente entre la salida del sistema aproximador (aquí supondremos, sin pérdida de generalidad, que es una RBFN) y la salida objetivo del problema. Para inicializar y optimizar los parámetros de la RBFN, se utiliza el algoritmo heurístico de los k vecinos

más cercanos (**knn**), con valor de k = 1, (Capítulo 2, Sección 2.1.6.2.1) para inicializar los radios *r* y técnica SVD para calcular los valores exactos de los pesos *w* (Capítulo 2, Sección 2.1.6.2.2).

Una vez que los valores de los centros $\vec{c}_j$, radios r_j y pesos w_j han sido fijados, se empieza el proceso de entrenamiento de RBFN y se calcula el error de la red, que es la diferencia entre la salida objetivo *y*, y la salida de la red $F(\vec{x})$. Este error se calcula por la siguiente expresión:

$$E_i = E(\vec{x}_i) = |y_i - F(\vec{x}_i)| \tag{3.22}$$

Este error es una parte fundamental del algoritmo, y dependerá de la red aproximadora utilizada, y de la estimación de sus parámetros. De este error dependen los procesos de la actualización de los clusters, la migración y el cálculo de la distorsión.

3.5.2.3 Actualización de los clusters

Después de la partición de los clusters y el cálculo del error, los clusters deben ser actualizados. La actualización es realizada por un proceso iterativo que actualiza los datos de entrenamiento que pertenecen a cada cluster. La ecuación que actualiza los clusters viene dada por:

$$\vec{c}_j = \frac{\sum_{\vec{x}_i \in C_j} \vec{x}_i E_i}{\sum_{\vec{x}_i \in C_j} E_i} \qquad i = 1,...,n, \quad j = 1,...m \tag{3.23}$$

3.5.2.4 Cálculo de la distorsión ε

Es el cálculo de la contribución de cada cluster en la distorsión total. En el proceso de desplazamiento local la distorsión es usada como medida para parar el proceso de desplazamiento local de los clusters. La ecuación usada para calcular la distorsión ε es:

$$\varepsilon = \frac{\sum_{j=1}^{m} \sum_{\vec{x}_i \in C_j} \left\| \vec{x}_i - \vec{c}_j \right\|^2 E_i}{\sum_{j=1}^{m} \sum_{\vec{x}_i \in C_j} E_i} \tag{3.24}$$

La distorsión se usa también en el paso de migración para calcular la mejora de la migración de clusters y parar el proceso.

3.5.3 El proceso de migración

La idea del proceso de migración consiste en intentar migrar clusters de las zonas que tienen pequeño error de aproximación a las zonas donde este error es grande, es decir, intenta hacer que la contribución de cada cluster a la distorsión total sea lo más homogénea posible. El proceso de migración es un proceso aleatorio para evitar mínimos locales. Existe otra manera para migrar los clusters que mueven los clusters con valor máximo de utilidad a la zona de clusters que tienen valor mínimo de utilidad. Este método sufre un problema, y es que cuando dos valores de utilidad son iguales el movimiento del cluster será imposible.

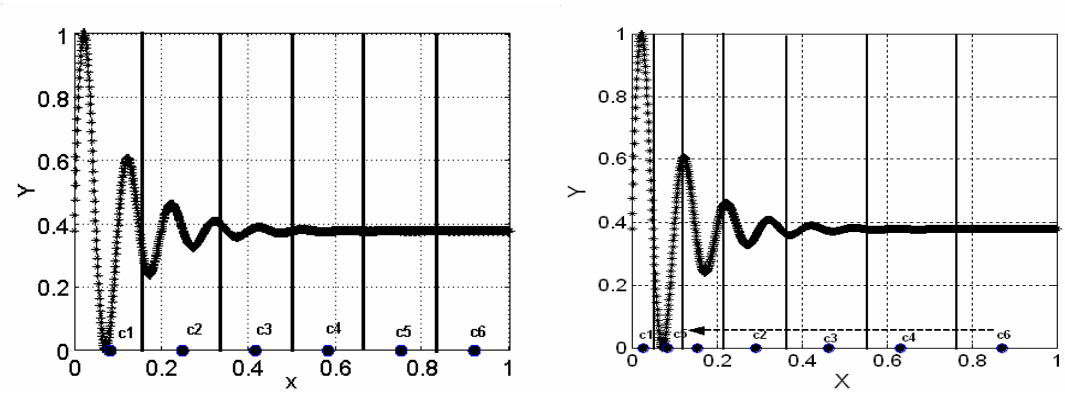


Fig. 3.9 *a*) clusters antes de la migración. *b*) clusters después de la migración

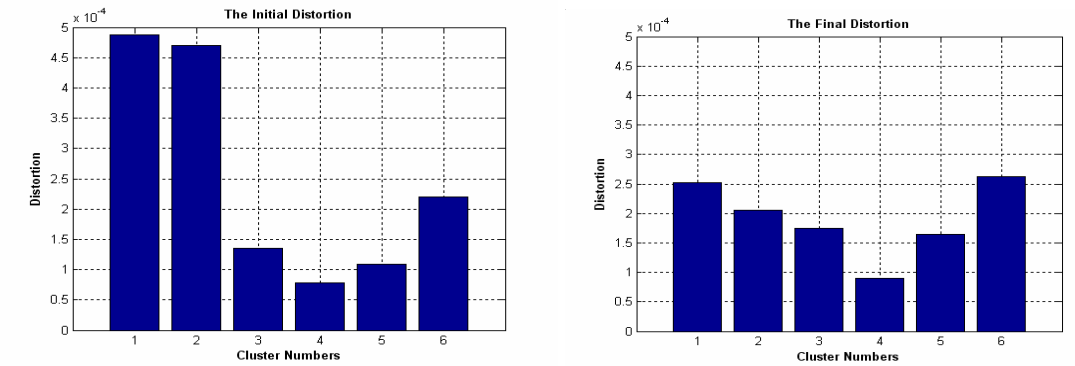


Fig. 3.10 *a*) distorsión antes de la migración *b*) distorsión después de la migración

El algoritmo propuesto se usa acceso aleatorio en el proceso de migración que mueve cluster asignados en zonas de entrada donde la función objetivo es estable a zonas donde la variabilidad de salida es más alta. Se usa el proceso de selección por ruleta para seleccionar un cluster que tiene utilidad menor que 1 ($U_j<1$) y mueve este cluster a la zona de otros clusters que tienen la utilidad mayor que 1 ($U_j>1$), seleccionado por el proceso de selección por ruleta también. La Fig. 3.9 muestra la situación de los clusters antes y después del proceso de migración. Este paso de migración es necesario porque en la iteración de la partición y en el desplazamiento local de clusters sólo se mueven los clusters de manera local. Pero el objetivo del algoritmo propuesto (ECFA) es descubrir también aquellas zonas de entrada donde el error de aproximación a la función objetivo es alto, y donde es necesario aumentar la densidad de clusters. En la Fig. 3.11 se muestra el proceso de migración por pasos.

En la Fig. 3.10 se muestra un ejemplo de la distorsión inicial y final de la función con 6 clusters. En el parte *a* de la figura mostramos la distorsión en cada cluster antes del proceso de migración de los clusters; el valor de la distorsión es muy diferente de un cluster a otro. Pero en la parte *b* la distorsión esta cerca de ser la misma en cada cluster**.**

El proceso de migración del algoritmo ECFA (ver Fig. 3.11) es diferente del proceso de migración en ELBG y el algoritmo CFA. En estas técnicas después del proceso de movimiento de clusters usan los movimientos de tres clusters y los clusters son asignados en la diagonal principal que contiene los clusters. Esta diagonal es dividida en tres partes, con los segmentos de lado siendo iguales a la mitad del central y los dos clusters son inicializados en los extremos del segmento central. Pero en el algoritmo ECFA después del movimiento de clusters que tienen utilidad menor que 1 a la zona de clusters que tienen utilidad mayor que 1, se usa el algoritmo de k-medias para dividir los datos que pertenecen al cluster con utilidad mayor que 1 entre los dos clusters, (esta manera es menos compleja y necesita menos tiempo de ejecución) así se obtienen los nuevos clusters. Después de este movimiento, es necesario realizar el desplazamiento local a todos los clusters. Después de terminar estos cambios de desplazamientos locales, la migración de los clusters es sólo aceptada si la distorsión total es rebajada, si no será rechazada.

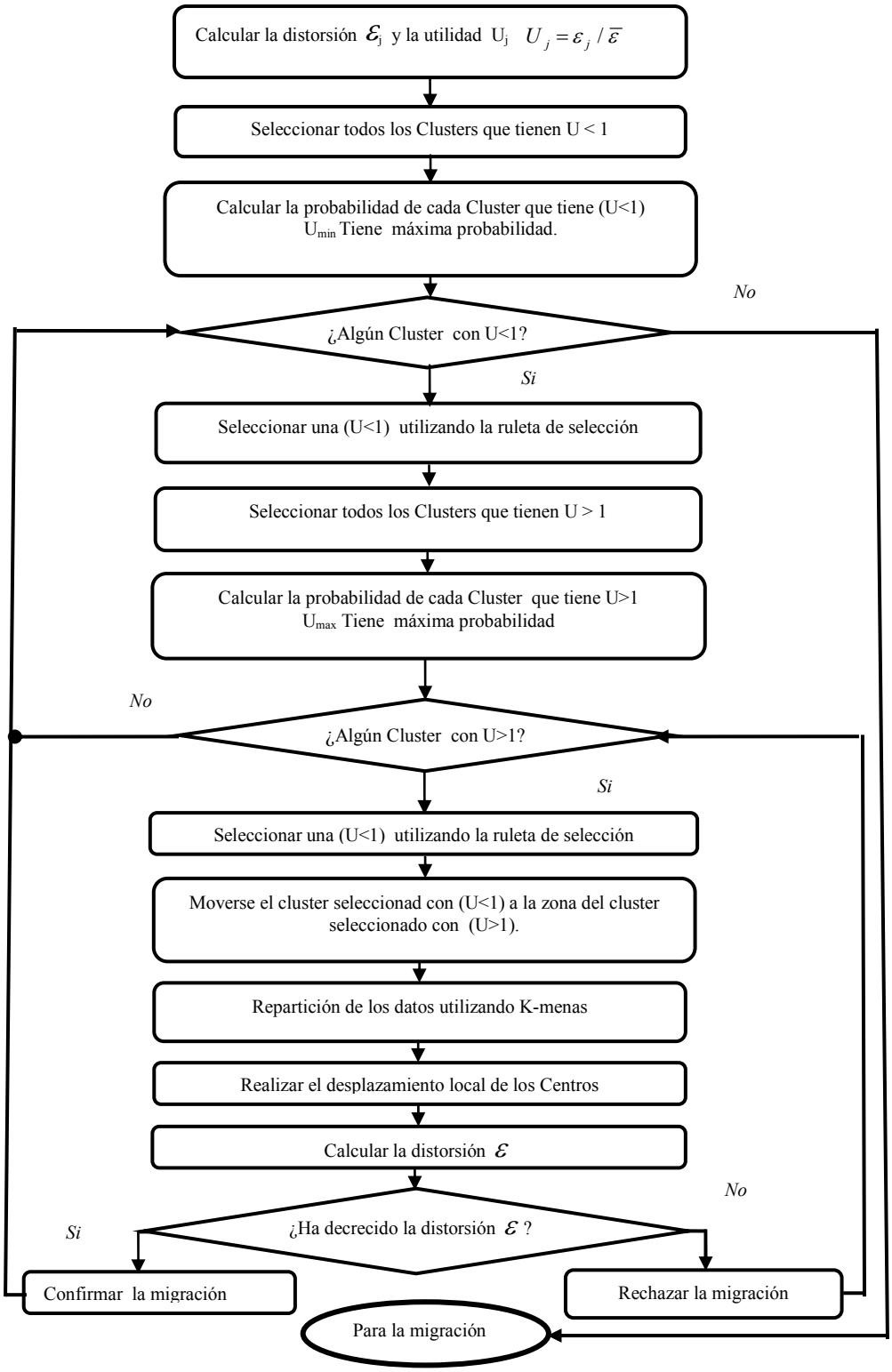


Fig. 3.11 El proceso de migración en el algoritmo ECFA

El algoritmo CFA incorpora unos parámetros del conjunto que representan una estimación de la respuesta de salida de cada cluster. El valor de cada parámetro es calculado como un promedio ponderado de las respuestas de salida de los datos de aprendizaje que pertenecen al cluster y concentra más clusters en aquellas regiones de entrada donde la respuesta de salida es más variable. El algoritmo ECFA utiliza la función objetivo de salida del problema que trata de aproximar, usando el concepto del error entere la salida de una RBFN, como la medida de la respuestas de cada cluster en su zona de los datos de entrenamiento y concentra más clusters en aquellas regiones de entrada donde la respuesta de la función objetivo es más variable. Depende en esto del error que provoca cada cluster que pertenece a una zona de los datos de entrada, por lo que será un algoritmo más robusto y más adecuado a cada tipo de red en particular, ya que usa como fuente de información el error de aproximación que dicha red consigue, para una configuración de clusters dada.

3.6 Conclusiones

En este capítulo se ha presentado los algoritmos de *clustering* tradicionales más utilizados. Esos algoritmos se dividen en dos tipos: no supervisados (k-medias, k-medias difuso, y Enhanced LBG) y supervisados (ACE, CFC, y CFA).

Se ha presentado un algoritmo de *clustering* para aproximación de funciones derivado del algoritmo de *clustering* para aproximación funcional *CFA* y el algoritmo de *enhanced LBG*. El algoritmo propuesto trata de aumentar el número de clusters en la zona del espacio de la entrada donde el error de aproximación de la red es mayor, teniendo en cuenta el grado de responsabilidad de cada cluster en este error. El objetivo final es intentar igualar la distribución del error de aproximación a lo largo de cada cluster, con el fin de que esta configuración sea adecuada para abordar un proceso de optimización local posterior. Si un algoritmo de *clustering* puede obtener la igualdad en la distorsión total, significa que cada cluster ha encontrado su lugar correcto en el espacio de entrada y la distribución de los clusters es la mejor.

La mayor parte de algoritmos de *clustering* sufren inconvenientes principales cuando tratan de resolver problemas de aproximación de funciones: los algoritmos de *clustering* son lentos y no escalan a un número grande de vectores de datos y convergen a

diferentes mínimos locales basados en las inicializaciones. Este algoritmo ECFA mejora estos inconvenientes como se mostrarán los resultados en el capítulo siguiente.

El algoritmo ECFA es rápido, con una complejidad menor que el algoritmo CFA, y es una aproximación de una estrategia global de obtener los clusters. El método es también simple de implementar.

El algoritmo que se presenta puede converger con número de clusters menor que el de estos algoritmos aplicados al mismo problema, su tiempo de ejecución y el error de aproximación son mejores respeto al algoritmo CFA, como se mostrarán los resultados del capítulo siguiente. Se puede utilizar este algoritmo para encontrar el número mínimo de centros que satisfacen un problema dado de aproximación de funciones.

CAPÍTULO 4

RESULTADOS EXPERIMENTALES DEL ALGORITMO DE CLUSTERING *ECFA*

Valorar un algoritmo de *clustering* que trata de resolver problemas de aproximación de funciones, teniendo en cuenta su carácter estocástico, es decir, que no siempre va a encontrar el mismo conjunto de soluciones, no es tarea sencilla. De tal forma habría que ejecutarlo varias veces y medir tanto el grado de aproximación conseguido, como su robustez. Además, otro factor también se ha de tener en cuenta: el tiempo de ejecución del algoritmo.

En este capítulo se presenta los resultados experimentales del algoritmo de *clustering* propuesto *ECFA* aplicado a una serie de funciones con distintas características. Este conjunto de funciones ya han sido utilizadas por otros autores para valorar sus algoritmos. Se aplica el algoritmo en cuatro tipos redes neuronales diferentes: redes de funciones de base radial normalizadas (NRBFN), redes de funciones de base radial (RBFN), redes neuronales *wavelet* (WNN) y redes neuronales *wavelet* normalizadas (NWNN).

4.1 Introducción

En el capítulo anterior se ha presentado una nueva metodología para inicializar los valores de los centros de una RBFN. Dicha metodología constaba de una serie de pasos:

1. Inicializar los valores de los centros utilizando el algoritmo de *clustering* k-medias.

2. El uso de un proceso de desplazamiento local para mover los centros de forma local. Este proceso costaba en algunas fases:

   - Primero se realiza la partición de los datos según el número de centros, teniendo en cuenta la cercanía de cada punto a cada centro.
   - El uso de la red aproximadora para calcular el error en cada zona de cada cluster.
   - Actualización de los centros de cada cluster.
   - Evaluación del proceso utilizando el cálculo de la distorsión.

3. El uso de un proceso de migración para mover los centros de zonas de menor error a zonas de mayor error.

A lo largo de este capítulo se estudiarán cada una de estas etapas utilizando distintas funciones que nos permitirán comprobar el rendimiento del algoritmo ECFA. Estas funciones han sido seleccionadas de la bibliografía y utilizadas por diferentes autores para valorar sus algoritmos diseñados para el problema de aproximación funcional. Los resultados del algoritmo propuesto ECFA se comparan con algoritmos de *clustering* y sobre todo con el algoritmo de *clustering* para aproximación funcional CFA [GON-01], que ha demostrado ser el mejor algoritmo de *clustering* para este tipo de problemas hasta la fecha [GON-01]. Para realizar las comparaciones se trabaja con dos parámetros principales para medir la actuación del algoritmo. El primero de ellos (y el más importante) es el error cuadrático medio normalizado (NRMSE) que nos mide el grado de aproximación final a los datos de E/S extraídos de la función a aproximar. Este error se calcula después del proceso de *clustering* y después de proceso de optimización local. En el proceso de optimización local se utilizan algoritmos tradicionales como SVD,

Knn, *Levenberg-Marquardt*, etc. ya comentados en el capítulo 2. El segundo parámetro es el tiempo de ejecución del algoritmo de *clustering*. El número de funciones base (número de clusters) en estos experimentos empieza con un número mínimo de 4 clusters y se aumenta de forma ordenada hasta que llega a un número suficiente de funciones base para aproximar la función.

Las simulaciones se realizan por el programa de software Matlab bajo del sistema operativo Windows XP. Matlab es un programa para trabajar con algoritmos matemáticos, gráficos e informes y simulación de software; con muchas funciones matemáticas, estadísticas, y de ingeniería ya implementadas. MATLAB da el acceso inmediato a la informática numérica de alto rendimiento. Esta funcionalidad es ampliada con su carácter interactivo, capacidades gráficas de crear gráficas, imágenes, superficies, y representaciones volumétricas. Las cajas de herramientas (toolboxes) de MATLAB son colecciones de los algoritmos, escritos por expertos en sus campos que proporcionan el número de aplicación específico de técnicas sin escribir el código.

Con el fin de demostrar la necesidad de incluir información sobre el tipo de red que se usará para realizar la aproximación en el propio algoritmo de *clustering*, todas las funciones serán aproximadas por cuatro tipos distintos de redes neuronales. Estos tipos de redes neuronales son: redes de funciones de base radial normalizada (NRBFN), redes de funciones de base radial (RBFN), redes neuronales *wavelet* normalizada (NWNN), redes neuronales *wavelet* (WNN). Las redes (NRBFN y NWNN) normalizadas se caracterizan porque en la obtención de su salida, se divide ésta por los grados de activación a las diferentes funciones base (RBF ó WNN):

$$F(\vec{x},\Phi,w)=\sum_{j=1}^{m}\frac{\varphi_j(\vec{x})}{\sum_{j=1}^{m}\varphi_j(\vec{x})}\cdot w_j \tag{4.1}$$

Además, se analizará la robustez del algoritmo ECFA ante el ruido, robustez del algoritmo ECFA ante la distribución y número de datos, y se analizará el uso de ECFA para el problema del modelado y predicción de series temporales.

4.2 Análisis detallado del algoritmo ECFA

En esta sección se analizarán ejemplos, con distintas complejidades cuyo fin es poner de relieve las principales características del algoritmo ECFA. Se utiliza un ejemplo de una dimensión y otro de dos dimensiones, ya que en ellos es posible una representación gráfica del funcionamiento del algoritmo.

4.2.1 Función de una dimensión $Var(x)$

Esta función $Var(x)$ se caracteriza por su alta variabilidad en la salida de la función objetivo cuando la entrada *x* esta cerca del valor cero 0 como muestra la Fig. 4.1.

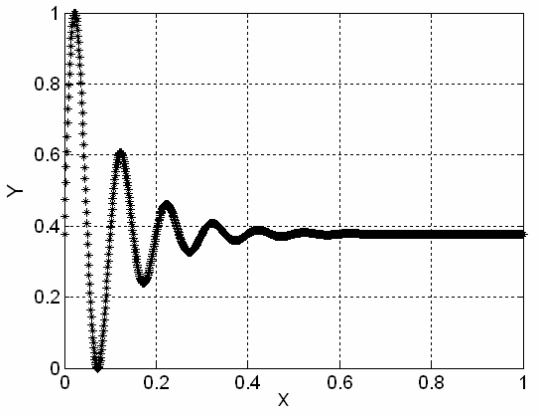


Fig. 4.1 Función objetivo de la función $Var(x)$

Un buen algoritmo de *clustering* tiene que colocar más clusters en la zona donde la salida objetivo tiene mayor variabilidad ya que es de esperar que así se reducirá el error de aproximación. En la Fig. 4.1 se puede observar que la variabilidad de la función $Var(x)$ se aumenta cuando *x* esta cerca del 0, y su salida casi constante para $x \geq 6$. Esta función ha sido utilizada por [GON-01] para valorar su algoritmo de *clustering* CFA.

$$Var(x) = \frac{sen(2\pi x)}{e^x} \quad x \in [0, 10] \tag{4.2}$$

4.2.1.1 Inicialización de los centros mediante k-medias

Para poner de manifiesto el efecto de la inicialización de los centros mediante el método de las k-medias consideramos la función $Var(x)$ con inicialización por este método y también de forma aleatoria. La mayoría de los algoritmos de *clustering* utilizan la inicialización aleatoria para los clusters, en el algoritmo ECFA se usa el algoritmo k-

medias para inicializar los valores de los centros. Para mostrar la diferencia de la posición de los centros entre los dos métodos, colocamos 6 centros en espacio de los datos de la función $Var(x)$ con las dos formas de inicialización.

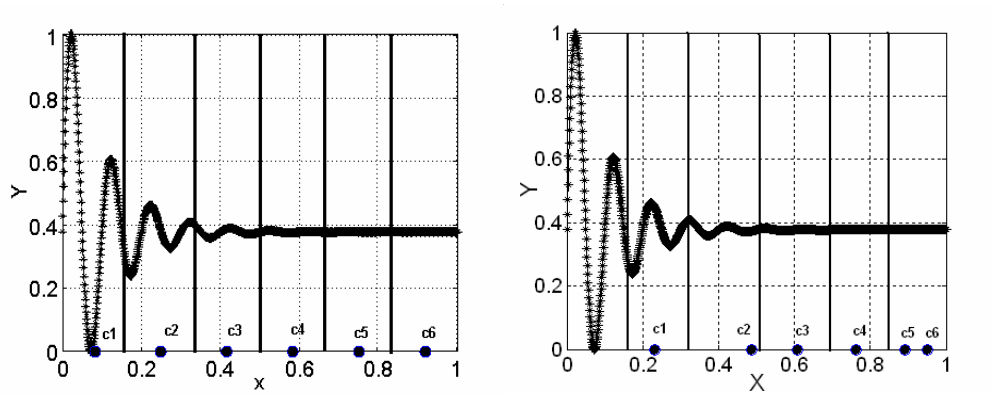


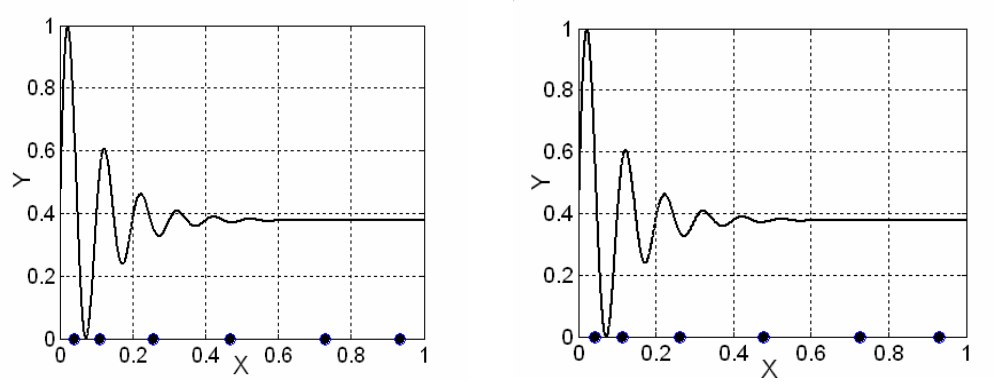
Fig. 4.2.*a*) Inicialización de centros por k-medias. *b*) Inicialización aleatoria de centros

De la Fig. 4.2 se puede observar que los centros se colocan de forma ordenada en el caso de inicialización por k-medias, pero en la inicialización de forma aleatoria los centros c_5 y c_6 se colocaron de mala forma. La utilización del algoritmo k-medias no afecta al algoritmo en el tiempo de ejecución porque se realiza sólo una vez. El algoritmo de las k-medias ya fue descrito en detalle en la sección 3.3.1.

4.2.1.2 El proceso del Desplazamiento local de los centros

Este proceso se usa para mover los centros de forma local, después del proceso de migración, los datos del centro que ha sido trasladado a zona de cluster con mayor error se quedan sin pertenecer a ningún *cluster*. Con este proceso estos datos pertenecerán a los centros más cercanos. Para estudiar el efecto de este proceso sobre la colocación de los centros en el espacio de los datos de entrada, colocamos 6 centros en espacio de los datos de la función $Var(x)$.

Después de inicializar los centros por el algoritmo k-medias, se calcula el error que provoca cada centro entre la salida objetivo y la salida real de RBFN de los datos que pertenecen a este centro. Los centros que tengan menor error se mueven a zonas de centros con mayor error (Fig. 4.3.*a*). Los datos de este centro se asignan a resto de los centros según la cercanía (Fig. 4.3.*b*).



(a) *(b)*

Fig. 4.3 El proceso de Desplazamiento local antes y después de la migración

4.2.1.3 El proceso de migración

El proceso de migración trata de mover centros que provocan menor error a zonas de centros con mayor error. Este proceso afecta mucho a la posición de los centros, ya que lo que se pretende es igualar el error en cada zona definida por los clusters.

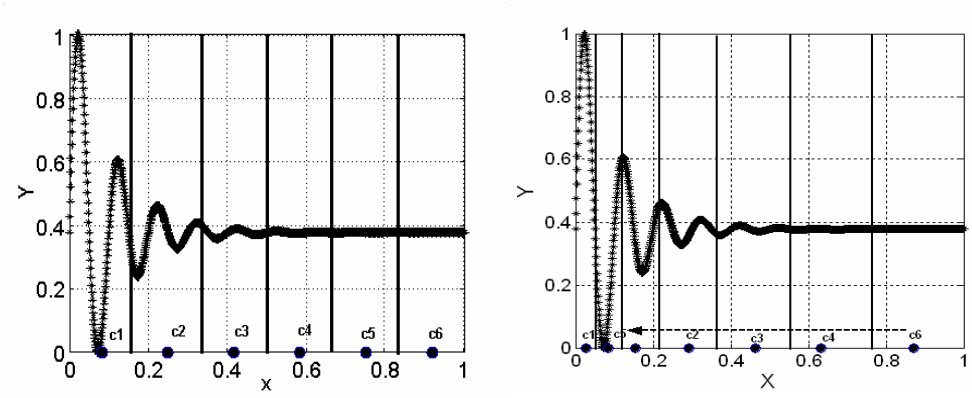


Fig. 4.4. Los centros antes y después del proceso de migración

El proceso de migración consiste en intentar migrar clusters de las zonas que tienen pequeño error de aproximación a las zonas donde este error es grande, es decir, intenta hacer que la contribución de cada cluster a la distorsión total sea lo más homogénea posible. De la Fig. 4.4 el centro c_5 se migra a la zona del centro c_1, ya que ahí es donde se necesita una mayor densidad de clusters para minimizar el error de aproximación. El efecto del proceso de migración depende, obviamente del cálculo de la distorsión, cuya

distribución es la función que pretendemos de minimizar, como se muestra en la Figura 3.10

4.2.2 Función de dos dimensiones $g(x_1, x_2)$

Esta función se utilizó por [PED-98], en su algoritmo de *clustering* difuso condicional (CFC), y por [GON-02] para comprobar sus resultados en el algoritmo CFA. Esta función se define por esta ecuación:

$$g(x_1,x_2) = \left[(x_1-2)(2x_1+1)\right]/(1+x_1^2)\ .\ \left[(x_2-2)(2x_2+1)\right]/(1+x_{21}^2) \qquad x_1,x_2 \in [-5,5] \qquad (4.3)$$

En la Fig. 4.5 se muestra la salida objetivo de la función $g(x_1, x_2)$, un conjunto de entrenamiento formado por 441 puntos distribuidos en una rejilla de 21 × 21 celdas en el dominio de la entrada.

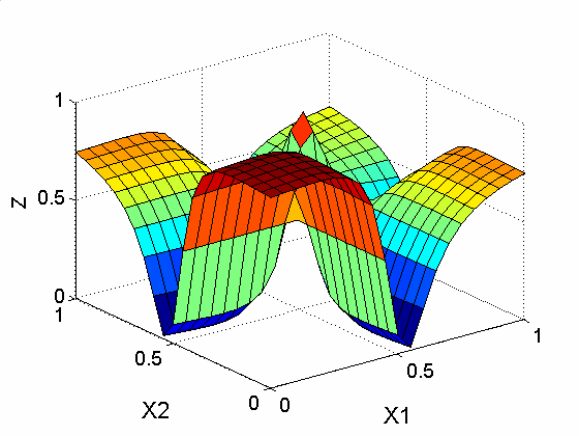


Fig. 4.5 Función objetivo *g*

4.2.2.1 Inicialización de los centros mediante k-medias

Para poner de manifiesto el efecto de la inicialización de los centros mediante k-medias consideramos la función $g(x_1, x_2)$ (Fig. 4.5) con inicialización por k-medias y de forma aleatoria. Para mostrar la diferencia de la posición de los centros entre los dos métodos, colocamos 6 centros en espacio de los datos de la función $g(x_1, x_2)$ con las dos formas de inicialización.

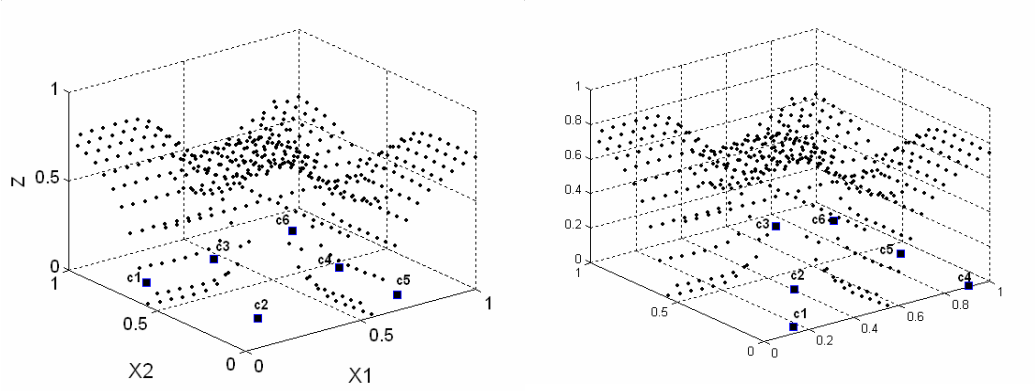
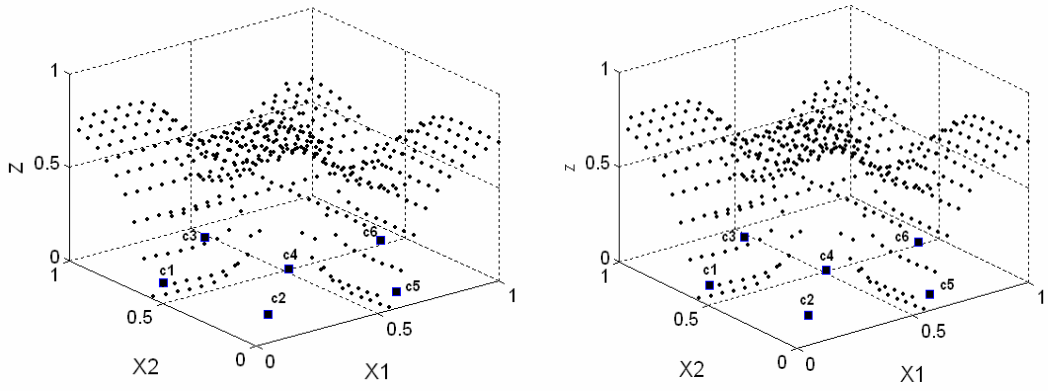


Fig. 4.6.*a*) Inicialización de centros por k-medias. *b*) Inicialización aleatoria de centros

De la Fig. 4.6 se puede observar que los centros se colocan de forma más adecuada usando la inicialización mediante el algoritmo de las k-medias.

4.2.2.2 El proceso del Desplazamiento local de los centros

Para estudiar el efecto de este proceso a la colocación de los centros en el espacio de los datos de entrada en una función de dos dimensiones, colocamos 6 centros en espacio de los datos de la función $g(x_1, x_2)$.



(a) ***(b)***

Fig. 4.7 El proceso de Desplazamiento local antes y después de la migración

4.2.2.3 El proceso de migración

El proceso de migración trata de mover centros que provocan menor error a zonas de centros con mayor error. De la Fig. 4.8 los centros c_4 y c_6 se migran a zonas con mayor error.

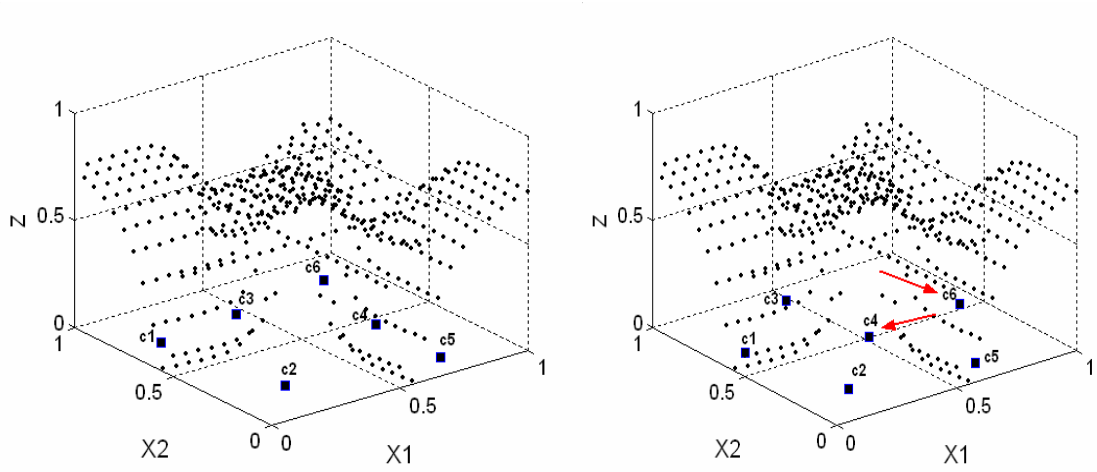


Fig. 4.8.*a*) Los centros antes y después del proceso de migración

4.3 Comparativa del algoritmo *ECFA* con otros algoritmos de clustering

En esta sección se describe dos ejemplos que comparan los resultados del algoritmo ECFA con otros algoritmos de *clustering* para la inicialización de los centros de una RBFN.

- Primer ejemplo

  Los datos de entrada de esta función $Var(x)$ (Fig. 4.1) están formados por 1000 puntos equidistribuidos en el intervalo de la función utilizada. A este conjunto de entrenamiento se le han aplicado los algoritmos de *clustering* de las k-medias, de las k-medias difuso con $\beta = 2$, ELBG, CFA con $\mu_{\min}$ = 0.001 (Capítulo 3, Sección 3.4.3), y ECFA.

  Se comienza con número de 4 funciones base y se sigue aumentando hasta un número adecuado de funciones base (Max =10).

  En las tablas de resultados se muestra:

  NRMSE$_C$: la media de error NRMSE obtenido tras el proceso de *clustering* (en 5 ejecuciones)

  NRMSE$_T$: la media del error NRMSE de entrenamiento (en 5 ejecuciones), tras el proceso de optimización local.

DevS: la desviación Standard del NRMSE tras las 5 ejecuciones.

RBF	k-medias	k-medias Difuso	ELBG	CFA	ECFA
4	0.984 ± 0.002	0.990 ± 7E-5	0.984 ± 0.005	0.952 ± 0.001	0.931 ± 0.003
5	0.979 ± 0.004	0.987 ± 9E-5	0.980 ± 8E-4	0.940 ± 0.018	0.912 ± 0.004
6	0.972 ± 0.005	0.985 ± 2E-4	0.970 ± 0.007	0.901 ± 0.006	0.883 ± 0.021
7	0.964 ± 0.007	0.983 ± 4E-4	0.961 ± 0.002	0.852 ± 0.031	0.829 ± 0.009
8	0.960 ± 0.006	0.98 ± 3E-4	0.960 ± 0.011	0.821 ± 0.058	0.810 ± 0.003
9	0.947 ± 0.013	0.977 ± 7E-4	0.954 ± 0.004	0.802 ± 0.002	0.734 ± 0.005
10	0.939 ± 0.015	0.976 ± 5E-4	0.945 ± 0.006	0.781 ± 0.001	0.721 ± 0.007

Tabla 4.1 Media y desviación estándar del NRMSE obtenido después la inicialización ($NRMSE_C$) para los algoritmos k-medias, k-medias difuso, ELBG, CFA y ECFA para aproximar la función *Var(x)*

RBF	k-medias	k-medias Difuso	ELBG	CFA	ECFA
4	0.851 ± 0.015	0.835 ± 0.205	0.211 ± 0.118	0.562 ± 0.051	0.331 ± 0.0
5	0.818 ± 0.034	0.812 ± 0.341	0.202 ± 0.132	0.345 ± 0.028	0.312 ± 0.064
6	0.759 ± 0.263	0.783 ± 0.317	0.152 ± 0.122	0.334 ± 0.008	0.113 ± 0.041
7	0.344 ± 0.281	0.248 ± 0.115	0.111 ± 0.072	0.275 ± 0.016	0.090 ± 0.007
8	0.227 ± 0.367	0.150 ± 0.162	0.093 ± 0.064	0.151 ± 0.028	0.082 ± 0.023
9	0.252 ± 0.386	0.299 ± 0.160	0.073 ± 0.057	0.132 ± 0.026	0.034 ± 0.031
10	0.086 ± 0.099	0.285 ± 0.333	0.064 ± 0.039	0.060 ± 0.015	0.021 ± 0.009

Tabla 4.2 Media y desviación estándar del NRMSE obtenido después de la minimización ($NRMSE_T$) para los algoritmos k-medias, k-medias difuso, ELBG, CFA y ECFA para aproximar la *función Var(x)*

De la Tabla 4.1 y Tabla 4.2 se puede comprobar cómo el algoritmo ECFA produce mejores configuraciones que el resto de los algoritmos con los que se compara, y cuando el número de clusters aumenta, reduce el error de aproximación considerablemente. Este efecto se debe a que el algoritmo ECFA es capaz de detectar las zonas donde el cluster provoca mayor error y colocar más clusters en dicha zona. En la Tabla 4.2 se aprencian los resultados finales tras la aplicación del algoritmo *Levenberg-Marquardt*. Se puede observar que los resultados de los algoritmos k-medias, k-medias difuso son bastante variables, esto debido a que dichos métodos solo realizan una búsqueda local a partir de la configuración inicial. Los algoritmos ELBG, CFA y ECFA evitan este problema mediante un proceso de migración que les permite buscar la configuración adecuada.

De este ejemplo podemos ver cómo la inicialización llevada mediante algoritmos de *clustering* tradicionales hace que la aplicación de un algoritmo de minimización del error converja hacía mínimos locales diferentes, lo que hace que las desviaciones sobre el error de aproximación final obtenidos sean mayores. Sin embargo, las desviaciones obtenidas sobre el error de aproximación final de la red inicial con los algoritmos ELBG, CFA y ECFA son bastante menores y sobre todo el algoritmo propuesto ECFA converge hacia prácticamente la misma configuración inicial independientemente de la configuración inicial de la que parta.

La Figura 4.9 muestra distintas aproximaciones de la función $Var(x)$ obtenidas por el algoritmo ECFA.

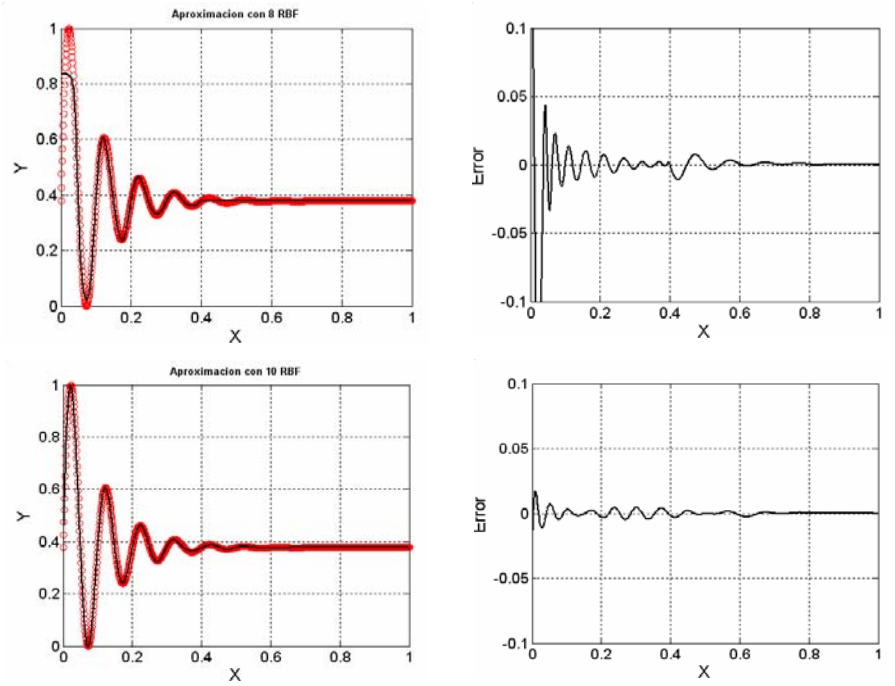


Fig. 4.9 Distintas aproximaciones de la función $Var(x)$

- Segundo ejemplo

  Esta función $dc(x)$ compara el comportamiento del algoritmo ECFA con otros algoritmos de *clustering* que tiene en cuenta la salida objetivo. Esta función se

utilizó por [RUN-99] en el algoritmo de estimación de grupos alternante (ACE). La Fig. 4.10 muestra la función:

$$dc(x) = \ 0.2 \ + \ 0.8(x + sen(2\pi x)) \qquad x \in [0\ ,1] \tag{4.4}$$

Esta función has sido utilizada también por [GON-01], para validar la capacidad de aproximación de su algoritmo CFA.

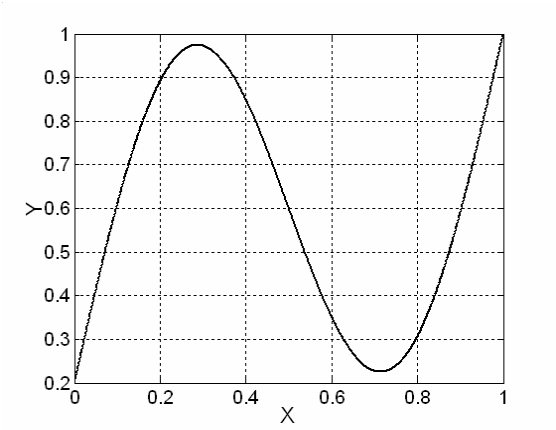


Fig. 4.10 Función objetivo $dc(x)$

RBF	ACE	CFA	ECFA
4	1.22 5 ± 0.045	0.380 ± 0.035	0.205 ± 0.021
5	1.179 ± 0.020	0.327 ± 0.078	0.203 ± 0.002
6	0.625 ± 0.018	0.309 ± 0.089	0.108 ± 0.003
7	0.633 ± 0.090	0.181 ± 0.051	0.121 ± 0.002
8	0.523 ± 0.072	0.167 ± 0.048	0.092 ± 0.008
9	0.714 ± 0.295	0.161 ± 0.039	0.084 ± 0.003
10	0.526 ± 0.096	0.098 ± 0.024	0.071 ± 0.002

Tabla 4.3 Media y desviación estándar del NRMSE obtenido después la inicialización ($NRMSE_C$) para los algoritmos ACE, CFA y ECFA para aproximar la función *dc(x)*

- Tercer ejemplo

  Los datos de entrada de la función $g(x_1, x_2)$ (Fig.4.5) están formados por 441 puntos distribuidos en una rejilla de 21 × 21 celdas en el dominio de la entrada. A este conjunto de entrenamiento se le han aplicado los algoritmos de *clustering* de las k-medias, de las k-medias difuso con $\beta = 2$, CFA con $\mu_{\min}$ = 0.001 (Capítulo 3, Sección 3.4.3), y nuestro ECFA.

Se comienza con un número de 4 funciones RBF y se sigue aumentando hasta un número adecuado de funciones base (Max =10)

RBF	k-medias	k-medias Difuso	CFA	ECFA
4	0.992 ± 0.002	0.995 ± 2E-4	0.942 ± 0.013	0.962 ± 0.003
5	0.989 ± 0.004	0.991 ± 5E-4	0.910 ± 0.039	0.868 ± 0.041
6	0.978 ± 0.005	0.988 ± 1E-4	0.861 ± 0.009	0.732 ± 0.053
7	0.969 ± 0.007	0.977 ± 9E-4	0.820 ± 0.038	0.712 ± 0.011
8	0.956 ± 0.006	0.965 ± 2E-4	0.774 ± 0.042	0.740 ± 0.006
9	0.932 ± 0.013	0.957 ± 9E-4	0.732 ± 0.051	0.634 ± 0.058
10	0.912 ± 0.015	0.953 ± 2E-4	0.731 ± 0.024	0.680 ± 0.019

Tabla 4.4 Media y desviación estándar del NRMSE obtenido después la inicialización para los algoritmos k-medias, k-medias difuso, CFA y ECFA para aproximar la función *Var(x)*

RBF	k-medias	k-medias Difuso	CFA	ECFA
4	0.831 ± 0.021	0.866 ± 0.197	0.532 ± 0.051	0.609 ± 0.000
5	0.798 ± 0.040	0.723 ± 0.131	0.425 ± 0.028	0.512 ± 0.064
6	0.639 ± 0.212	0.613 ± 0.176	0.424 ± 0.008	0.293 ± 0.041
7	0.423 ± 0.011	0.433 ± 0.095	0.362 ± 0.016	0.269 ± 0.007
8	0.359 ± 0.037	0.427 ± 0.151	0.341 ± 0.028	0.148 ± 0.023
9	0.331 ± 0.236	0.393 ± 0.137	0.312 ± 0.026	0.123 ± 0.031
10	0.267 ± 0.081	0.375 ± 0.121	0.230 ± 0.015	0.071 ± 0.009

Tabla 4.5 Media y desviación estándar del NRMSE obtenido después la minimización para los algoritmos k-medias, k-medias difuso, CFA y ECFA para aproximar la función *Var(x)*

De la Tabla 4.4 y Tabla 4.5 se puede comprobar cómo el algoritmo ECFA produce mejores configuraciones que el resto de los algoritmos con los que se compara, y cuando el número de clusters aumenta, reduce el error de aproximación considerablemente. Este efecto se debe a que el algoritmo ECFA es capaz de detectar las zonas donde el cluster provoca mayor error y colocar más clusters en dicha zona. En la Tabla 4.5 se vuelve a mostrar los resultados de aproximación finales tras la aplicación del algoritmo *Levenberg-Marquardt*. Se puede observar que los resultados de los algoritmos k-medias, k-medias difuso son bastante variables, esto debido a que dichos métodos solo realizan una búsqueda local a partir de la configuración inicial. Los algoritmos, CFA y ECFA evitan este problema mediante un proceso de migración que les permite buscar la configuración adecuada.

De este ejemplo podemos ver de nuevo cómo la inicialización llevada mediante los algoritmos de *clustering* tradicionales hace que la aplicación de una algoritmo de

minimización del error converja hacía mínimos locales diferentes y no adecuados, lo que hace que las desviaciones sobre el error de aproximación final obtenidos sean mayores. Sin embargo, las desviaciones obtenidas sobre el error de aproximación final de la red inicializada con los algoritmos CFA y ECFA son bastante menores y sobre todo para el algoritmo ECFA, lo que demuestra que el algoritmo propuesto ECFA converge hacía la misma configuración inicial independientemente de la configuración inicial de la que parta, y que, además, la configuración final encontrada es una de las mejores posibles.

La Figura 4.11 muestra distintas aproximaciones de la función $g(x_1, x_2)$ obtenidas por el algoritmo ECFA.

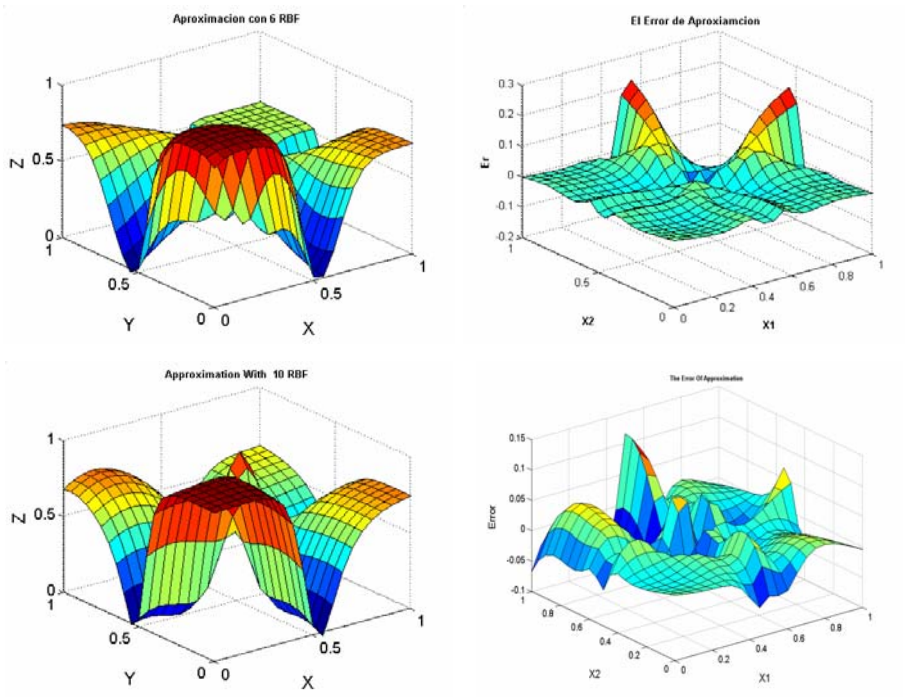


Fig. 4.11 Distintas aproximaciones de la función $g(x_1, x_2)$

De los resultados obtenidos, se puede observar los resultados obtenidos por el algoritmo ECFA son mejores de los resultados obtenidos de otros algoritmos de *clustering* tradicionales como k-medias, k-medias difuso y ELBG, e incluso que el algoritmo CFA, específicamente diseñado para problemas de aproximación funcional. En adelante, para seguir comprobando las características y el rendimiento del algoritmo ECFA propuesto, las comparaciones de los resultados se harán siempre entre CFA y ECFA.

Finalmente, veamos el comportamiento del algoritmo ECFA para otras funciones de mayor número de entradas. En este caso, las representaciones gráficas no son posibles aunque, como siempre, además de mostrar los resultados que obtiene el algoritmo propuesto ECFA, se compara estos resultados con los resultados obtenidos por el algoritmo CFA aplicado a estas funciones, que ya demostró ser el mejor en las comparaciones [GON-01], [GON-02]. La comparación se realiza considerando los valores del error cuadrático medio normalizado (NRMSE). Este error se mide después del proceso de *clustering* como se explica en el Capítulo 3, y después del proceso de optimización local, utilizando algoritmos tradicionales como SVD, Knn y *Levenberg-Marquardt*, para todos los ejemplos. También se compara el tiempo de ejecución para que el algoritmo converja. El número de funciones base (número de clusters) en estos experimentos empieza con número mínimo de 4 clusters y se aumenta de forma ordenada hasta que llega a un número suficiente de funciones base para aproximar una función.

- **Función** $f_{10}(x_1, x_2, x_3, x_4)$

$$f_{10}(x_1, x_2, x_3, x_4) = e^{(2x_1 \sin(\pi x_4))} + \sin(x_2 x_3) \quad x_1, x_2, x_3, x_4 \in [-0.25, 0.25] \qquad (4.5)$$

Esta función generada por 10000 puntos sus resultados de la media del NRMSE en 5 ejecuciones, después del proceso de *clustering* y después de utilizar el algoritmo de minimización de los errores *Levenberg-Marquardt*, obtenidos por la ejecución del algoritmo ECFA.

	ECFA						CFA					
m	**NRMSE $_C$**	**DevS**	**NRMSE$_T$**	**DevS**	**NRMSE$_{min}$**	**Tiempo$_C$**	**NRMSE $_C$**	**DevS**	**NRMSE$_T$**	**DevS**	**NRMSE$_{min}$**	**Tiempo$_C$**
4	0.99	9E-3	0.202	5E-2	0.166	13.4	0.99	1E-2	0.773	1E-2	0.666	61.5
5	0.98	2E-2	0.100	4E-2	0.062	19.5	0.99	3E-3	0.164	6E-3	0.153	71.6
6	0.77	2E-2	0.041	6E-3	0.033	49.9	0.99	8E-3	0.151	1E-2	0.141	66.9
7	0.68	1E-2	0.027	6E-3	0.018	30.8	0.94	5E-2	0.026	2E-3	0.025	175.1
8	0.65	9E-2	0.014	3E-3	0.010	52.5	0.94	8E-2	0.023	2E-3	0.021	155.2
9	0.65	1E-2	0.008	4E-3	0.003	45.2	0.85	2E-2	0.011	6E-3	0.008	212.5
10	0.60	9E-2	0.005	3E-4	0.005	91.5	0.64	1E-2	0.008	1E-3	0.007	190.4

Tabla 4.6: Comparativa de resultados de los algoritmos ECFA y CFA en el caso NRBFN

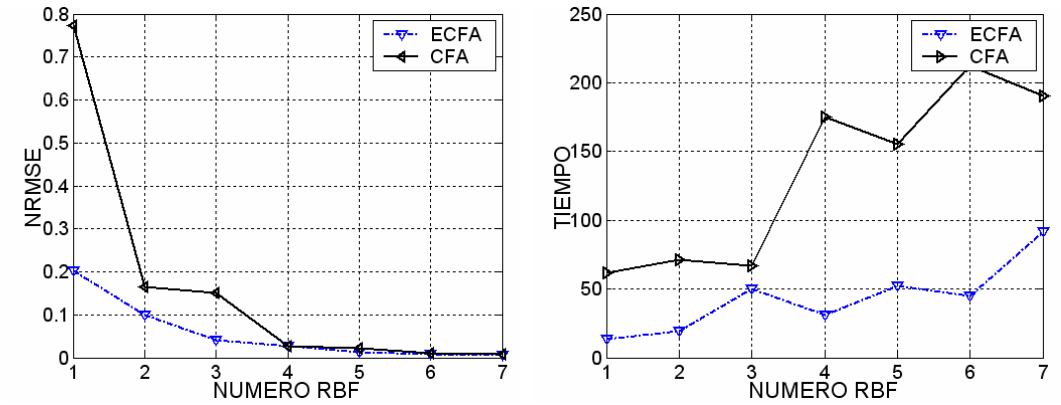


Fig. 4.12 Comparación $NRMSE_T$ y Tiempo de clustering entre ECFA y CFA en el caso NRBFN

- **Función** $f_{11}(x_1, x_2, x_3, x_4, x_5, x_6)$

$$f_{11}(x_1,x_2,x_3,x_4,x_5,x_6) = 10\sin(\pi x_1 x_2) + 20(x_3 + 0.5)^2 + 10\,x_4 + 5\,x_5 + 0x_4$$
$$x_1,x_2,x_3,x_4,x_5,x_6 \in [-1, 1] \qquad (4.6)$$

Esta función ha sido generada por 15000 puntos. En la siguiente tabla se muestran los resultados obtenidos de la media del NRMSE en 5 ejecuciones, después del proceso de *clustering* y después de utilizar el algoritmo de minimización local *Levenberg-Marquardt*.

	ECFA						CFA					
m	**$NRMSE_C$**	**DevS**	**$NRMSE_T$**	**DevS**	**$NRMSE_{min}$**	**$Tiempo_C$**	**$NRMSE_C$**	**DevS**	**$NRMSE_T$**	**DevS**	**$NRMSE_{min}$**	**$Tiempo_C$**
4	0.89	1E-2	0.27	4E-2	0.245	24.1	0.96	4E-2	0.25	5E-2	0.25	347.3
5	0.77	9E-2	0.23	3E-3	0.231	42.3	0.80	2E-2	0.24	3E-3	0.24	352.9
6	0.72	1E-2	0.22	1E-2	0.201	39.2	0.91	6E-3	0.25	4E-2	0.20	276.6
7	0.59	2E-2	0.19	2E-2	0.137	71.7	0.85	1E-1	0.24	5E-2	0.20	313.0
8	0.63	2E-2	0.18	1E-2	0.101	53.3	0.69	2E-1	0.15	1E-2	0.14	277.8
9	0.55	5E-2	0.19	3E-2	0.108	127.6	0.58	6E-2	0.20	1E-2	0.19	353.4
10	0.57	4E-2	0.13	4E-2	0.061	92.9	0.55	2E-2	0.23	3E-2	0.19	344.1

Tabla 4.7: Comparativa de resultados de los algoritmos ECFA y CFA en el caso NRBFN

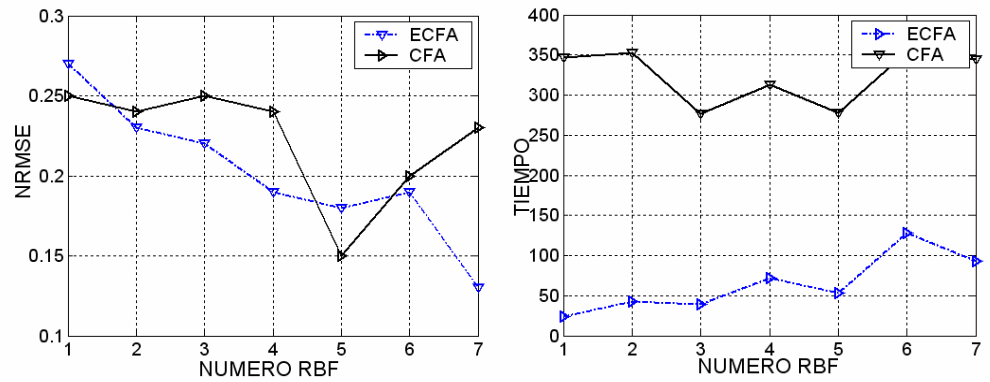


Fig. 4.13 Comparación $NRMSE_T$ y Tiempo de clustering entre ECFA y CFA en el caso NRBFN

4.4 Uso del algoritmo ECFA para otros tipos de redes neuronales aproximadoras

Con el fin de demostrar la necesidad de incluir información sobre el tipo de red que se usará para realizar la aproximación en el propio algoritmo de *clustering*, todas las funciones de este apartado serán aproximadas por cuatro tipos distintos de redes neuronales. Estos tipos de redes neuronales son: redes de funciones de base radial normalizada (NRBFNs), redes de funciones de base radial (RBFNs), redes neuronales *wavelet* normalizada (NWNN), redes neuronales *wavelet* (WNN).

A continuación se muestra una gran cantidad de ejemplos de funciones de una, dos y más dimensiones aplicadas a los cuatro tipos de redes neuronales (NRBFNs, RBFNs, NWNNs y WNNs). Estos resultados se comparan con el algoritmo CFA [GON-02] con valor de μ_{min} = 0.001 (Capítulo 3, Sección 3.4.3).

En las tablas de resultados se muestra: NRMSEC: la media del error NRMSE obtenido tras el proceso de *clustering*, **$NRMSE_T$**: la media del error NRMSE de entrenamiento (en 5 ejecuciones), tras el proceso de optimización local, **$NRMSE_{min}$**: el valor mínimo del NRMSE en 5 ejecuciones, **DevS**: la desviación Standard del NRMSE tras las 5 ejecuciones y **$Tiempo_C$**: la media del tiempo que tarda el algoritmo de *clustering*.
Al ser ejemplos sin ruido, el error de test es prácticamente igual al error de entrenamiento, por lo que en todas las tablas se muestra el error NRMSE de entrenamiento.

4.4.1 ECFA en funciones de una dimensión

En primer lugar, se utiliza funciones de una sola dimensión. Los conjuntos de los datos de entrada de estas funciones están formados por 1000 puntos equidistribuidos en el intervalo de la función utilizada, con un número de funciones base que empieza con 4 RBF y sigue aumentando hasta un número adecuado de funciones base, en estos resultados experimentales se utiliza número máximo de funciones base igual a 10.

- **Función** $Var(x)$

$$Var(x) = \frac{sen(2\pi x)}{e^{x}} \quad x \in [0,\ 10] \tag{4.7}$$

La salida objetivo de esta función se presenta en la Fig. 4.1

	ECFA						CFA					
m	**NRMSE $_C$**	**DevS**	**NRMSE$_T$**	**DevS**	**NRMSE$_{min}$**	**Tiempo$_C$**	**NRMSE $_C$**	**DevS**	**NRMSE$_T$**	**DevS**	**NRMSE$_{min}$**	**Tiempo$_C$**
4	0.94	6E-3	0.55	1E-2	0.46	1.4	0.92	2E-3	0.53	9E-2	0.47	14.2
5	0.91	2E-2	0.47	2E-2	0.45	1.1	0.89	1E-3	0.53	6E-2	0.47	11.1
6	0.87	2E-3	0.42	4E-2	0.37	2.0	0.87	1E-2	0.44	4E-2	0.39	18.9
7	0.82	1E-3	0.23	2E-2	0.22	2.9	0.86	1E-2	0.41	4E-2	0.36	30.2
8	0.79	3E-2	0.31	1E-2	0.20	5.4	0.85	2E-2	0.22	9E-3	0.21	14.2
9	0.74	3E-2	0.02	5E-3	0.02	6.5	0.84	2E-2	0.23	1E-2	0.20	81.9
10	0.70	1E-2	0.03	2E-2	0.02	9.7	0.83	2E-2	0.14	2E-2	0.12	167.5

Tabla 4.8.a) Comparativa de resultados de los algoritmos ECFA y CFA en el caso de NRBFN

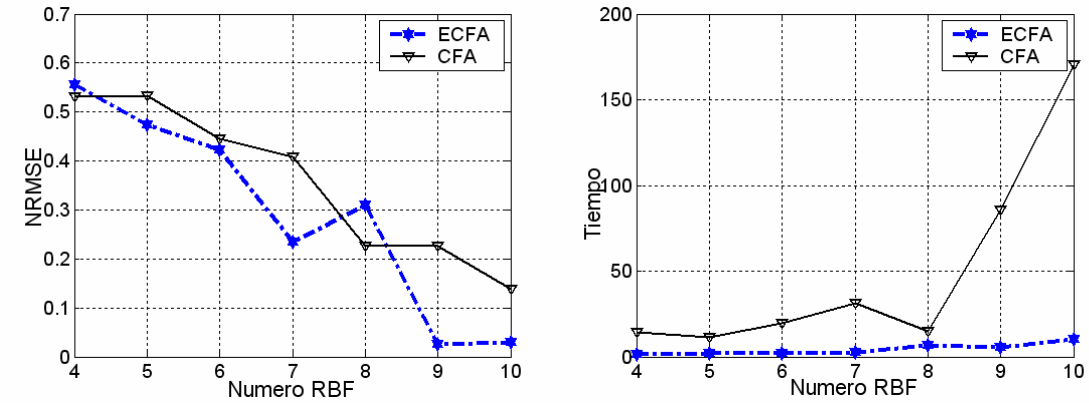


Fig. 4.14 Comparación NRMSE$_T$ y Tiempo de clustering entre ECFA y CFA en el caso NRBFN

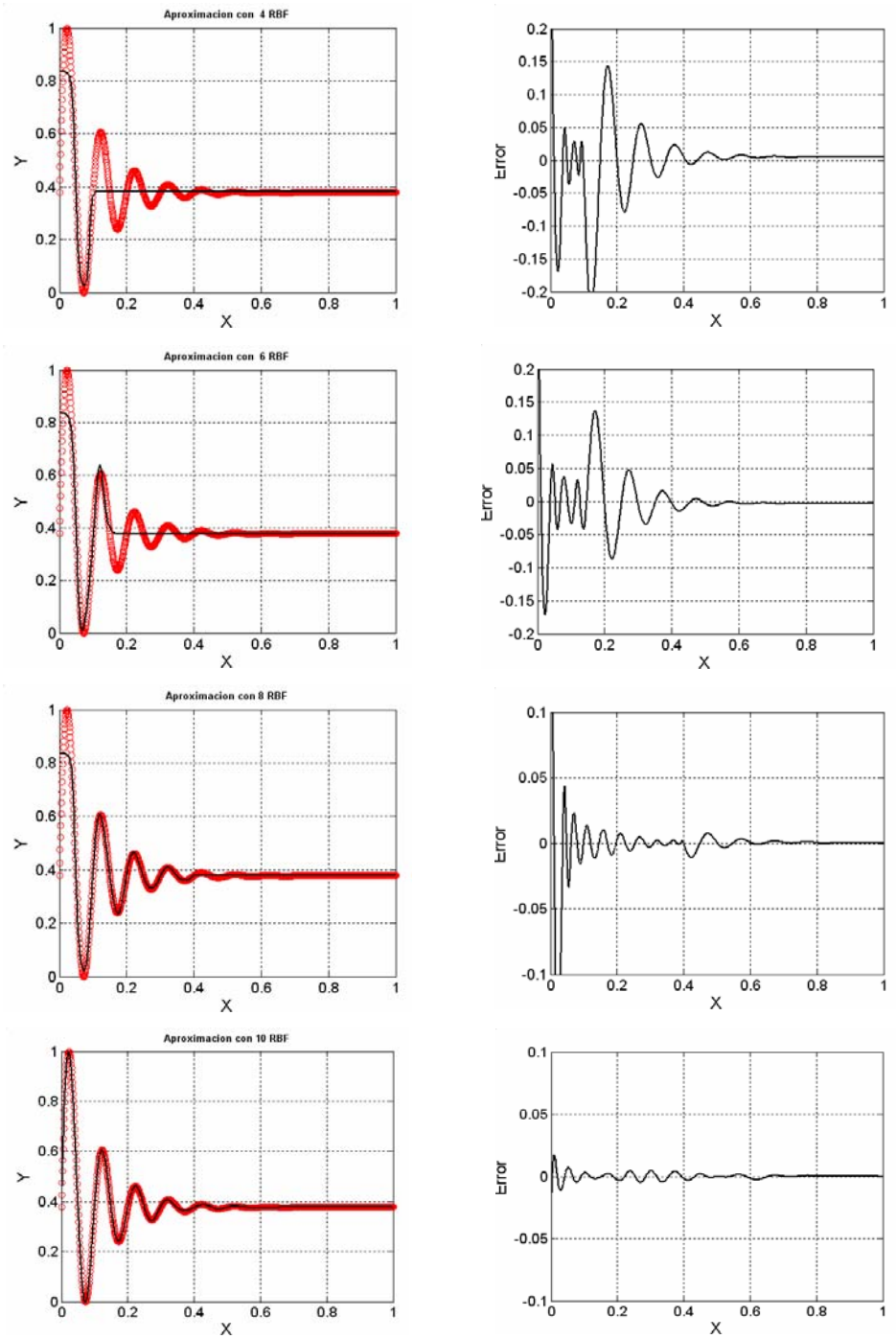


Fig. 4.15 Distintas aproximaciones de la función $Var(x)$

	ECFA						CFA					
m	**NRMSE $_C$**	**DevS**	**NRMSE$_T$**	**DevS**	**NRMSE$_{min}$**	**Tiempo$_C$**	**NRMSE $_C$**	**DevS**	**NRMSE$_T$**	**DevS**	**NRMSE$_{min}$**	**Tiempo$_C$**
4	1.06	0.00	0.33	0.0	0.42	1.7	1.01	4E-3	0.56	1E-2	0.54	14.2
5	1.04	2E-3	0.31	6E-2	0.16	2.1	0.98	4E-3	0.34	2E-2	0.32	11.1
6	1.03	2E-2	0.11	4E-2	0.08	9.4	0.96	8E-3	0.3	3E-2	0.27	18.9
7	1.00	1E-2	0.09	7E-3	0.06	10.3	0.94	2E-2	0.27	1E-3	0.25	30.2
8	0.95	9E-2	0.08	2E-2	0.07	26.4	0.95	1E-2	0.15	2E-2	0.13	14.2
9	0.94	5E-2	0.03	3E-2	0.03	19.8	0.92	3E-2	0.13	1E-2	0.12	81.9
10	0.84	1E-2	0.02	9E-3	0.02	47.2	0.88	1E-1	0.06	6E-3	0.05	167.5

Tabla 4.8.*b*): Comparativa de resultados de los algoritmos ECFA y CFA en el caso de RBFN

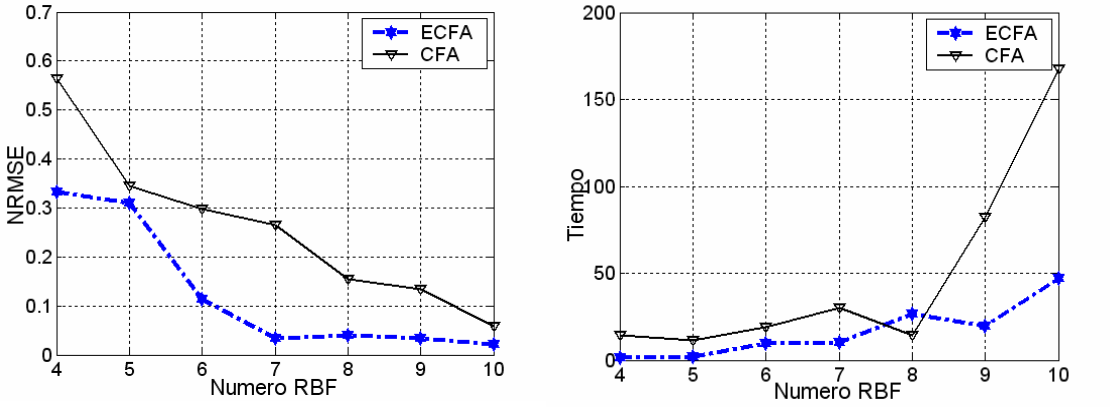


Fig. 4.16 Comparación NRMSE$_T$ y Tiempo de clustering entre ECFA y CFA en el caso RBFN

	ECFA						CFA					
m	**NRMSE $_C$**	**DevS**	**NRMSE$_T$**	**DevS**	**NRMSE$_{min}$**	**Tiempo$_C$**	**NRMSE $_C$**	**DevS**	**NRMSE$_T$**	**DevS**	**NRMSE$_{min}$**	**Tiempo$_C$**
4	0.94	2E-4	0.39	6E-4	0.39	1.2	0.92	1E-3	0.47	2E-2	0.46	14.2
5	0.90	2E-2	0.29	7E-2	0.21	1.9	0.89	4E-4	0.53	5E-2	0.46	11.1
6	0.87	4E-3	0.34	7E-2	0.22	2.0	0.87	1E-2	0.43	8E-2	0.35	18.9
7	0.83	6E-3	0.19	3E-2	0.16	2.8	0.84	2E-2	0.45	9E-2	0.38	30.2
8	0.78	2E-2	0.10	1E-2	0.03	6.9	0.85	4E-3	0.41	6E-2	0.32	14.2
9	0.72	2E-2	0.03	2E-2	0.02	5.7	0.83	5E-2	0.28	5E-2	0.22	81.9
10	0.71	1E-2	0.02	7E-3	0.01	10.3	0.80	5E-2	0.14	1E-2	0.12	167.5

Tabla 4.8.*c*): Comparativa de resultados de los algoritmos ECFA y CFA en el caso de NWNN

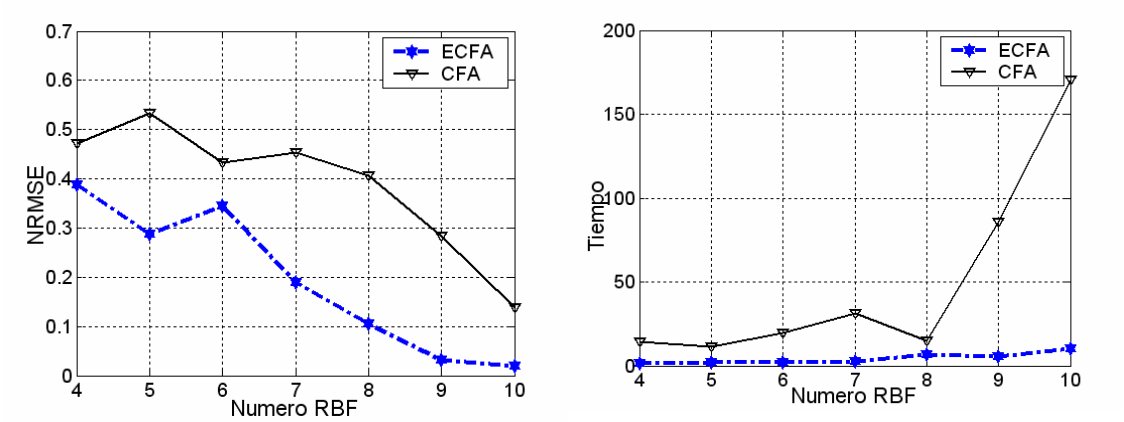


Fig. 4.17 Comparación $NRMSE_T$ y Tiempo de clustering entre ECFA y CFA en el caso NWNN

	ECFA						CFA					
m	**$NRMSE_C$**	**DevS**	**$NRMSE_T$**	**DevS**	**$NRMSE_{min}$**	**$Tiempo_C$**	**$NRMSE_C$**	**DevS**	**$NRMSE_T$**	**DevS**	**$NRMSE_{min}$**	**$Tiempo_C$**
4	0.88	6E-3	0.27	9E-3	0.47	2.8	0.84	1E-3	0.60	7E-2	0.51	14.2
5	0.85	5E-3	0.10	7E-3	0.16	1.3	0.79	1E-4	0.5	5E-2	0.49	11.1
6	0.86	0.00	0.15	4E-3	0.08	6.1	0.79	7E-3	0.36	6E-2	0.26	18.9
7	0.84	1E-3	0.11	3E-2	0.06	9.4	0.77	1E-2	0.27	2E-2	0.25	30.2
8	0.80	1E-4	0.07	2E-2	0.04	20.2	0.77	1E-2	0.24	1E-2	0.23	14.2
9	0.83	4E-2	0.03	1E-2	0.02	12.7	0.76	3E-3	0.14	2E-2	0.10	81.9
10	0.78	5E-2	0.02	9E-3	0.03	43.2	0.77	8E-3	0.15	1E-2	0.14	167.5

Tabla 4.8.*d*): Comparativa de resultados de los algoritmos ECFA y CFA en el caso de WNN

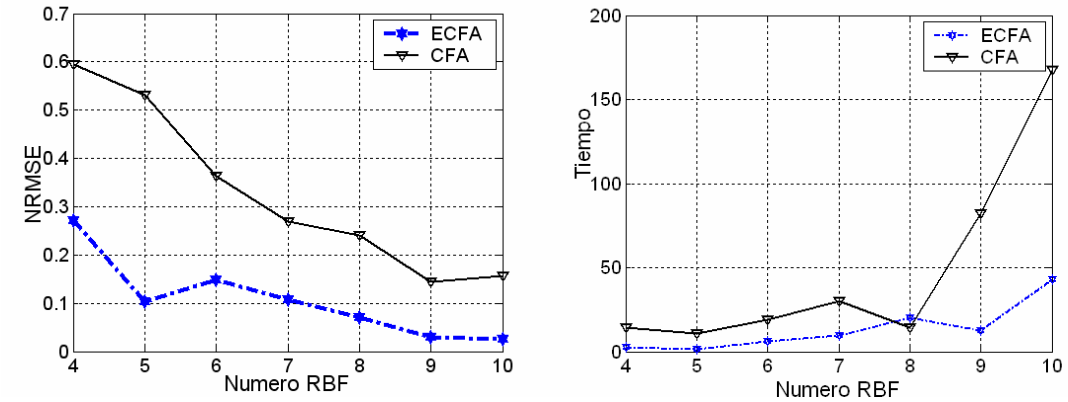


Fig. 4.18 Comparación $NRMSE_T$ y Tiempo de clustering entre ECFA y CFA en el caso WNN

- **Función $wm_{11}(X)$**

  Esta función ha sido también utilizada por Wang y Mendel [WAN-92], [SUD-94], [POM-00], [GON-01] y [RIV-03] para evaluar la capacidad de aproximación de su algoritmo.

$$wm_{11}(x) = sen(2\pi\ x) \qquad x \in [-1,\ 1] \tag{4.8}$$

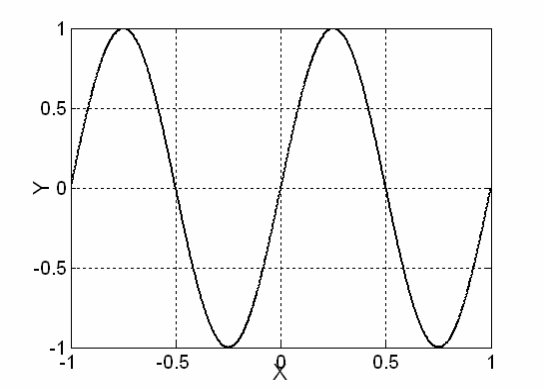


Fig. 4.19 Función objetivo $wm_{11}(X)$

	ECFA						CFA					
m	**NRMSE $_C$**	**DevS**	**NRMSE$_T$**	**DevS**	**NRMSE$_{min}$**	**Tiempo$_C$**	**NRMSE $_C$**	**DevS**	**NRMSE$_T$**	**DevS**	**NRMSE$_{min}$**	**Tiempo$_C$**
4	0.75	0.0	2E-2	0.0	2E-2	0.70	0.7	7E-3	1E-2	7E-3	7E-3	177.9
5	0.70	0.0	1E-2	6E-5	1E-2	1.8	0.70	2E-2	1E-2	7E-3	1E-2	40.5
6	0.12	4E-4	1E-3	2E-5	1E-3	2.8	0.25	2E-2	3E-2	6E-4	2E-2	152.2
7	0.14	9E-2	7E-4	4E-4	5E-4	7.5	0.2	5E-2	3E-3	3E-4	2E-3	270.2
8	0.04	1E-4	4E-4	5E-6	3E-4	4.7	0.17	1E-2	4E-3	2E-4	2E-3	383.5
9	0.05	9E-3	6E-4	2E-5	3E-5	7.4	0.17	2E-2	2E-3	4E-4	2E-3	263.9
10	0.02	2E-3	5E-4	1E-5	3E-5	10.0	0.14	1E-2	2E-3	2E-4	2E-3	221.3

Tabla 4.9.*a*): Comparativa de resultados de los algoritmos ECFA y CFA en el caso de NRBFN

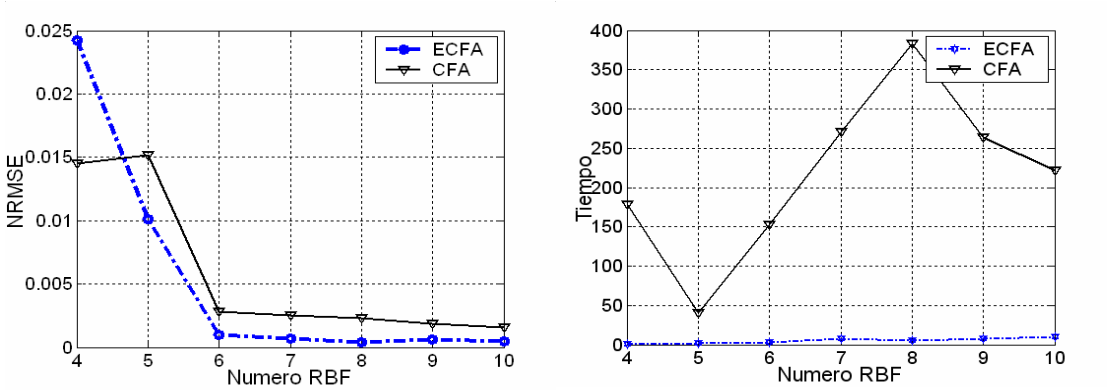


Fig. 4.20 Comparación $NRMSE_T$ y Tiempo de clustering entre ECFA y CFA en el caso NRBFN

	ECFA						CFA					
m	**NRMSE $_C$**	**DevS**	**$NRMSE_T$**	**DevS**	**$NRMSE_{min}$**	**$Tiempo_C$**	**NRMSE $_C$**	**DevS**	**$NRMSE_T$**	**DevS**	**$NRMSE_{min}$**	**$Tiempo_C$**
4	0.58	6E-3	5E-2	1E-2	2E-2	1.2	0.65	2E-2	7E-2	4E-2	4E-2	177.9
5	0.49	2E-2	4E-3	4E-3	3E-3	1.3	0.57	9E-3	5E-2	9E-3	5E-2	40.5
6	0.29	4E-4	5E-3	9E-3	8E-4	8.1	0.17	2E-2	6E-3	2E-3	2E-3	152.2
7	0.15	3E-3	2E-3	5E-4	2E-4	4.8	0.13	1E-2	5E-3	2E-3	3E-3	270.2
8	0.10	1E-3	2E-3	1E-4	3E-4	8.7	0.09	1E-2	5E-3	1E-3	4E-3	383.5
9	0.12	2E-2	1E-3	3E-4	7E-4	17.8	0.09	1E-2	4E-3	1E-3	2E-3	263.9
10	0.10	2E-2	2E-3	7E-4	6E-4	40.9	0.06	4E-3	4E-3	1E-3	3E-3	221.3

Tabla 4.9.*b*): Comparativa de resultados de los algoritmos ECFA y CFA en el caso de RBFN

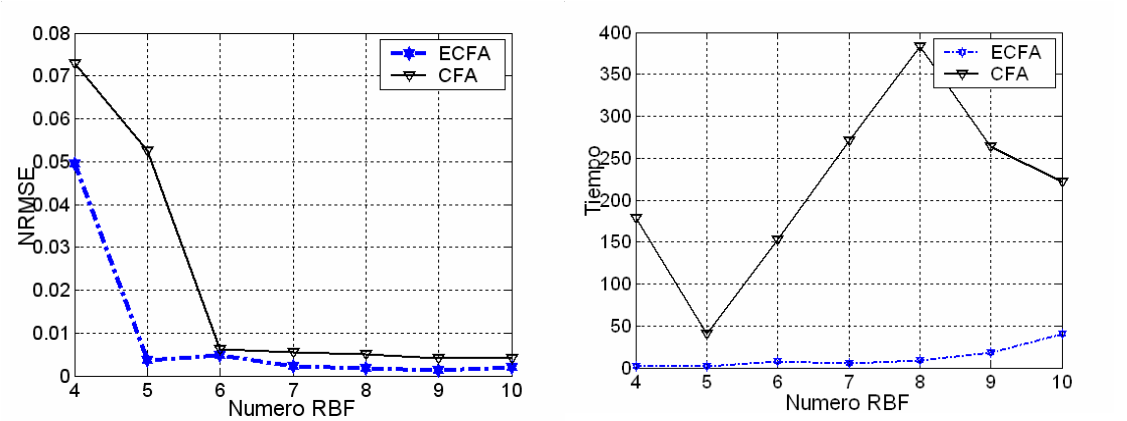


Fig. 4.21 Comparación $NRMSE_T$ y Tiempo de clustering entre ECFA y CFA en el caso RBFN

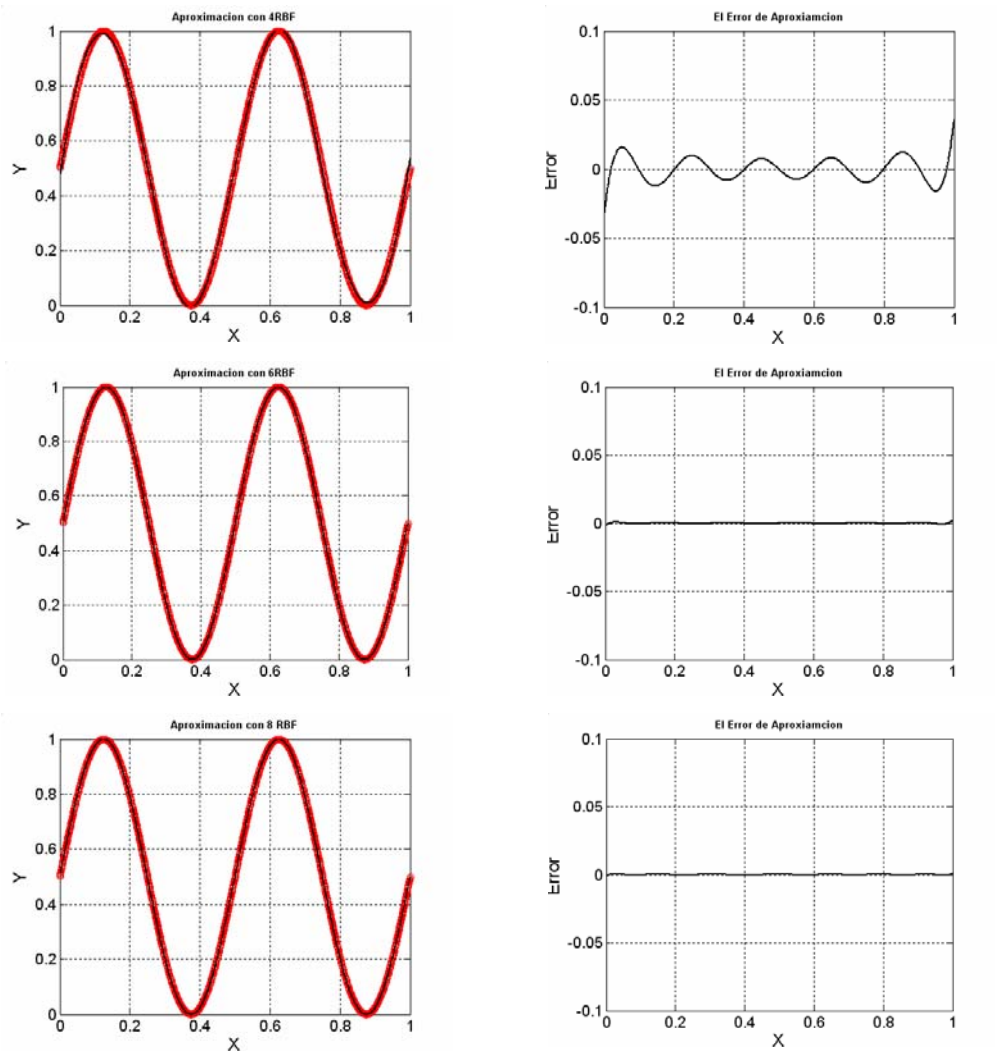


Fig. 4.22 Distintas aproximaciones de la función $wm_{11}(x)$

	ECFA						CFA					
m	**NRMSE $_C$**	**DevS**	**NRMSE$_T$**	**DevS**	**NRMSE$_{min}$**	**Tiempo$_C$**	**NRMSE $_C$**	**DevS**	**NRMSE$_T$**	**DevS**	**NRMSE$_{min}$**	**Tiempo$_C$**
4	0.177	0.0	2E-2	5E-4	1E-2	0.67	0.41	2E-1	2E-2	4E-3	1E-2	177.9
5	0.312	0.00	5E-3	0.0	4E-3	1.9	0.16	4E-2	7E-3	1E-3	7E-3	40.5
6	0.037	3E-4	2E-3	5E-5	9E-4	2.9	0.05	9E-3	4E-3	6E-4	4E-3	152.2
7	0.074	4E-2	1E-3	7E-4	5E-4	6.2	0.03	6E-3	3E-3	5E-4	2E-3	270.2
8	0.011	0.0	3E-3	0.0	3E-3	5.1	0.03	8E-3	3E-3	3E-4	2E-3	383.5
9	0.010	4E-3	1E-3	4E-4	7E-4	6.4	0.03	9E-3	2E-3	3E-4	2E-3	263.9
10	0.006	1E-3	7E-4	2E-4	5E-4	11.9	0.01	2E-3	1E-3	4E-4	1E-3	221.3

Tabla 4.9.*c*): Comparativa de resultados de los algoritmos ECFA y CFA en el caso de NWNN

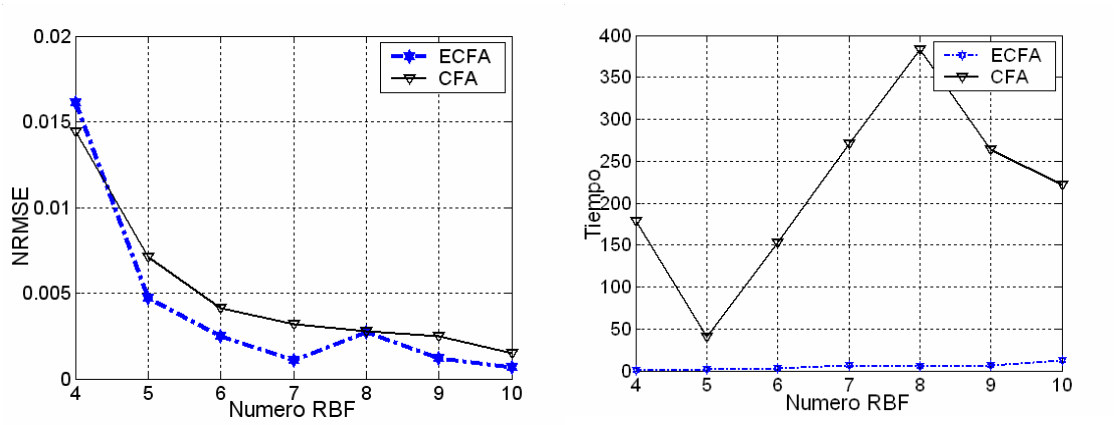


Fig. 4.23 Comparación $NRMSE_T$ y Tiempo de clustering entre ECFA y CFA en el caso NWNN

	ECFA						CFA					
m	**$NRMSE_C$**	**DevS**	**$NRMSE_T$**	**DevS**	**$NRMSE_{min}$**	**$Tiempo_C$**	**$NRMSE_C$**	**DevS**	**$NRMSE_T$**	**DevS**	**$NRMSE_{min}$**	**$Tiempo_C$**
4	0.141	3E-2	5E-3	6E-3	8E-4	3.0	0.180	2E-2	8E-3	6E-3	3E-3	177.9
5	0.108	0.0	3E-3	1E-3	2E-3	1.3	0.110	8E-3	3E-3	4E-4	3E-3	40.5
6	0.032	1E-3	2E-3	9E-4	7E-4	3.1	0.043	8E-3	3E-3	4E-4	3E-3	152.2
7	0.006	2E-3	1E-3	0.0	9E-4	5.0	0.034	4E-3	3E-3	1E-4	2E-3	270.2
8	0.007	1E-3	5E-4	2E-4	3E-4	6.0	0.025	4E-3	2E-3	1E-4	2E-3	383.5
9	0.005	1E-3	4E-4	3E-4	2E-4	4.8	0.019	5E-3	1E-3	1E-4	1E-3	263.9
10	0.002	2E-3	3E-4	1E-4	2E-4	8.0	0.014	2E-3	1E-3	1E-4	1E-3	221.3

Tabla 4.9.*d*): Comparativa de resultados de los algoritmos ECFA y CFA en el caso de WNN

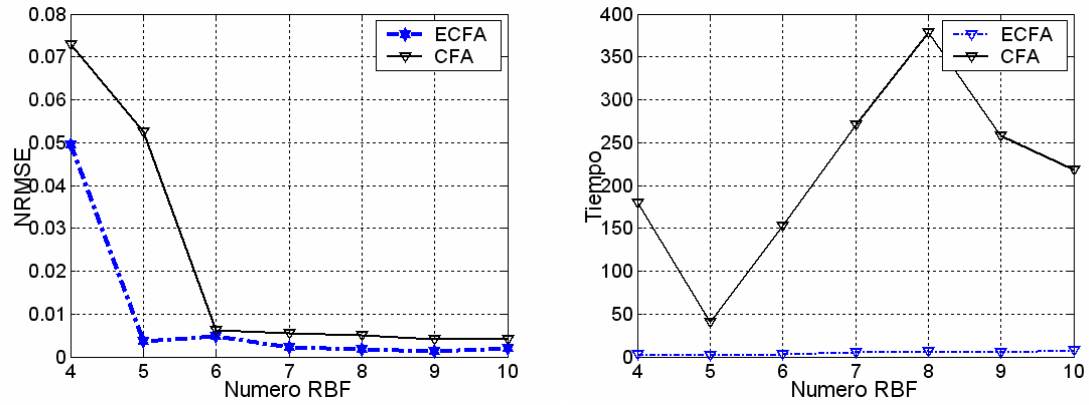


Fig. 4.24 Comparación $NRMSE_T$ y Tiempo de clustering entre ECFA y CFA en el caso WNN

- **Función *dick(x)***

Esta función $dick(x)$ fue inicialmente utilizada por Dickerson y Kosko en [DIC-96]. Actualmente se utiliza para comparar los resultados obtenidos con otros sistemas neuro-difusos. Esta función de define de la siguiente forma:

$$dick(x) = 3x\,(x-1)\,(x-1.9)\;(x+0.7)\,(x+1.8) \qquad x \in [-2.1, 2.1] \tag{4.9}$$

Esta función has sido utilizado por [POM-00], [GON-01] y [RIV-03] para validar la capacidad de aproximación de sus algoritmos.

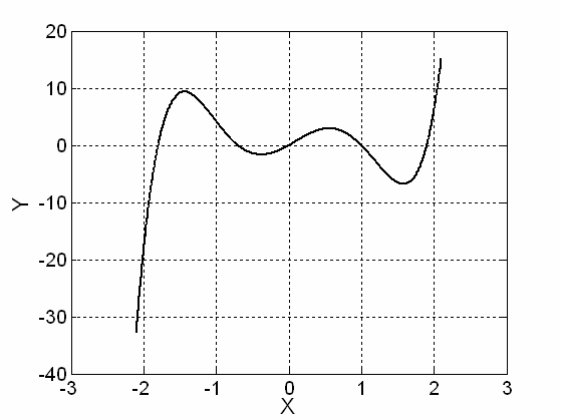


Fig. 4.25 Función objetivo $dick(x)$

	ECFA						CFA					
m	**NRMSE $_C$**	**DevS**	**NRMSE$_T$**	**DevS**	**NRMSE$_{min}$**	**Tiempo$_C$**	**NRMSE $_C$**	**DevS**	**NRMSE$_T$**	**DevS**	**NRMSE$_{min}$**	**Tiempo$_C$**
4	0.89	0.0	3E-2	3E-4	2E-2	1.0	0.92	2E-2	5E-2	1E-2	3E-2	120.3
5	0.87	5E-4	3E-3	1E-3	2E-3	2.1	0.87	9E-3	1E-2	2E-3	1E-3	267.0
6	0.44	0.0	3E-2	5E-4	1E-3	3.3	0.47	1E-2	5E-3	9E-4	4E-3	294.3
7	0.47	5E-2	3E-3	1E-4	1E-3	10.8	0.46	2E-2	5E-3	6E-4	4E-3	214.6
8	0.26	9E-3	3E-3	1E-2	1E-3	5.1	0.35	4E-2	4E-3	7E-4	3E-3	329.1
9	0.23	2E-2	3E-3	1E-3	1E-3	6.5	0.37	1E-2	4E-3	1E-3	2E-3	405.7
10	0.17	9E-3	2E-3	9E-3	1E-3	9.4	0.26	3E-2	3E-3	4E-3	3E-3	346.0

Tabla 4.10.*a*): Comparativa de resultados de los algoritmos ECFA y CFA en el caso de NRBFN.

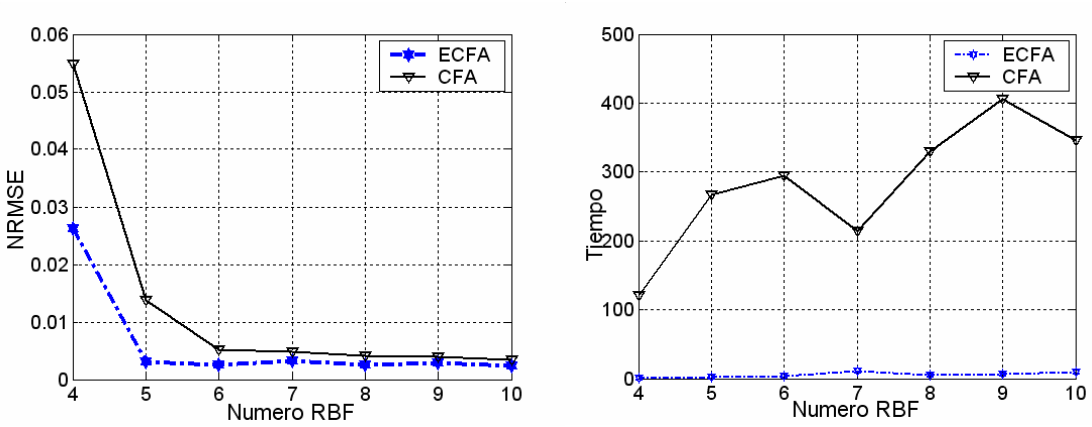


Fig. 4.26 Comparación $NRMSE_T$ y Tiempo de clustering entre ECFA y CFA en el caso NRBFN

	ECFA						CFA					
m	**$NRMSE_C$**	**DevS**	**$NRMSE_T$**	**DevS**	**$NRMSE_{min}$**	**$Tiempo_C$**	**$NRMSE_C$**	**DevS**	**$NRMSE_T$**	**DevS**	**$NRMSE_{min}$**	**$Tiempo_C$**
4	0.89	0.00	8E-1	0.0	8E-1	0.83	0.89	4E-3	5E-2	9E-3	4E-2	120.3
5	0.9	2E-4	9E-2	0.0	9E-2	2.3	0.87	2E-2	5E-2	8E-3	4E-2	267.0
6	0.53	1E-4	3E-3	1E-4	3E-3	2.6	0.58	3E-2	6E-2	8E-3	5E-2	294.3
7	0.54	1E-2	3E-3	1E-3	1E-3	3.5	0.56	4E-2	3E-2	1E-2	3E-2	214.6
8	0.37	4E-4	2E-3	5E-4	9E-4	6.5	0.46	3E-2	2E-2	4E-4	2E-2	329.1
9	0.33	3E-2	3E-3	2E-3	8E-4	7.1	0.39	2E-2	2E-2	4E-3	1E-2	405.7
10	0.28	1E-2	2E-3	4E-4	1E-3	8.41	0.32	1E-2	1E-2	1E-3	1E-2	346.0

Tabla 4.10.*b*): Comparativa de resultados de los algoritmos ECFA y CFA en el caso de RBFN

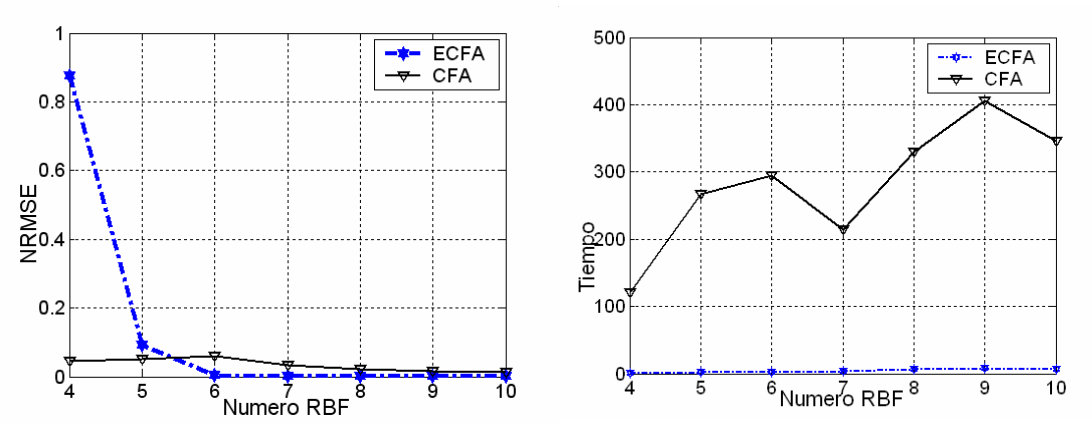


Fig. 4.27 Comparación $NRMSE_T$ y Tiempo de clustering entre ECFA y CFA en el caso RBFN

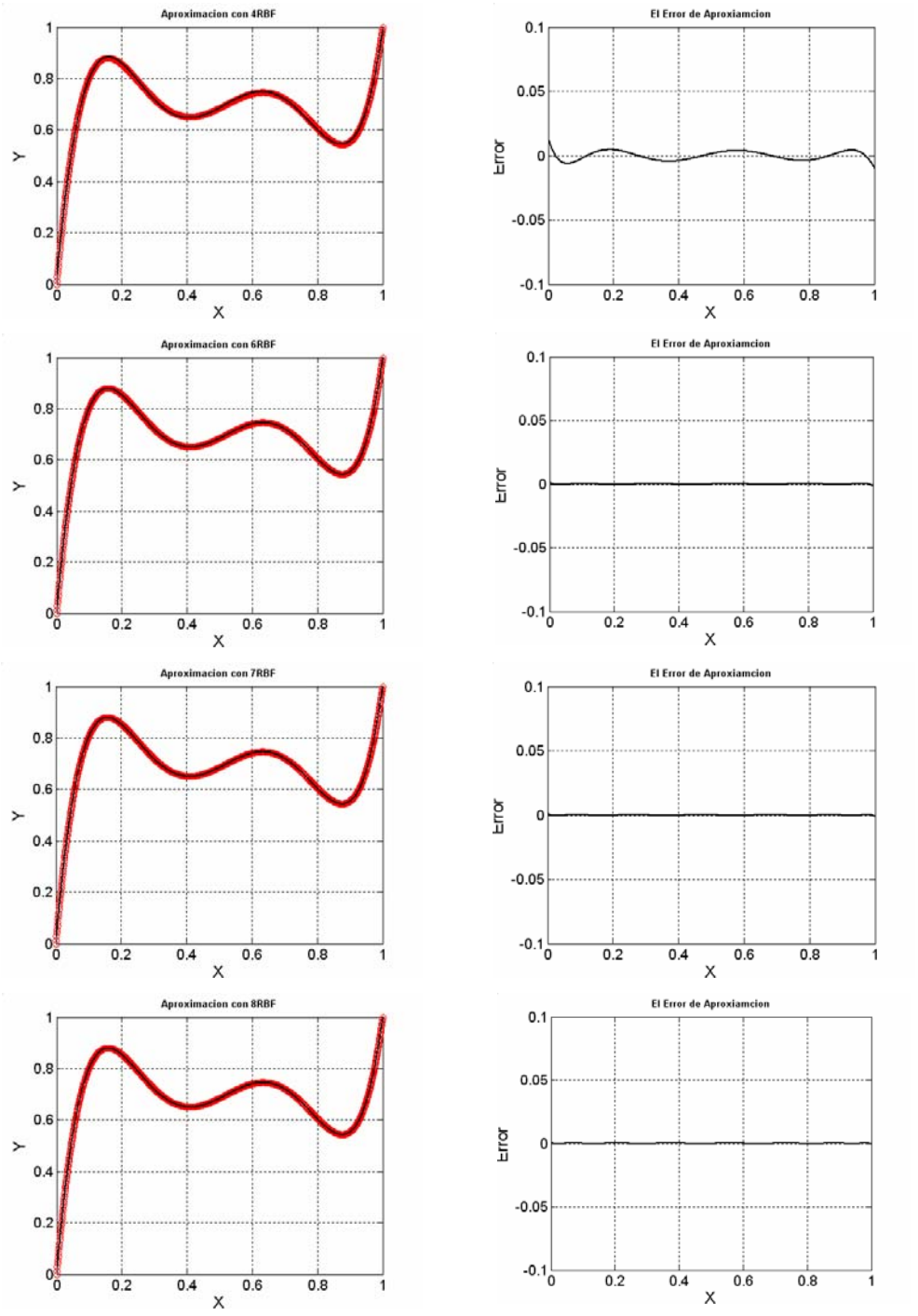


Fig. 4.28 Distintas aproximaciones de la función $dick(x)$

	ECFA						CFA					
m	**$NRME_C$**	**DevS**	**$NRMSE_T$**	**DevS**	**$NRMSE_{min}$**	**$Tiempo_C$**	**NRMSE $_C$**	**DevS**	**$NRMSE_T$**	**DevS**	**$NRMSE_{min}$**	**DevS**
4	0.37	7E-3	1E-2	2E-2	1E-2	1.3	0.36	5E-2	2E-2	4E-3	1E-2	120.3
5	0.30	2E-3	5E-3	4E-4	5E-3	2.2	0.35	5E-2	1E-2	2E-3	1E-2	267.0
6	0.16	4E-4	5E-3	3E-3	2E-3	2.3	0.31	2E-2	9E-3	1E-3	7E-3	294.3
7	0.19	1E-2	3E-3	2E-2	2E-3	10.2	0.25	4E-2	7E-3	2E-3	5E-3	214.6
8	0.06	7E-3	5E-3	2E-4	5E-3	5.0	0.23	3E-2	6E-3	1E-3	4E-3	329.1
9	0.09	3E-2	3E-3	1E-3	2E-3	7.5	0.15	5E-3	5E-3	4E-4	4E-3	405.7
10	0.04	6E-3	3E-3	4E-4	3E-3	10.6	0.16	2E-2	4E-3	1E-3	2E-3	346.0

Tabla 4.10.*c*): Comparativa de resultados de los algoritmos ECFA y CFA en el caso de NWNN

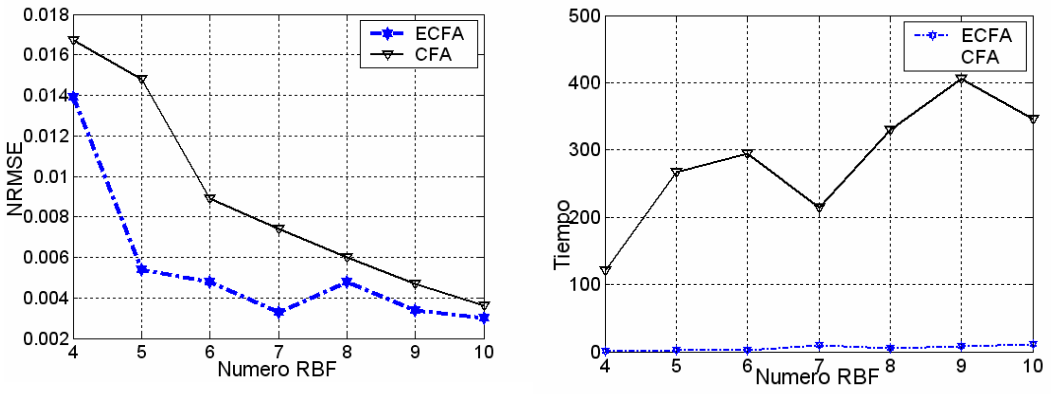


Fig. 4.29 Comparación $NRMSE_T$ y Tiempo de clustering entre ECFA y CFA en el caso NWNN

	ECFA						CFA					
m	**NRMSE $_C$**	**DevS**	**$NRMSE_T$**	**DevS**	**$NRMSE_{min}$**	**$Time_C$**	**NRMSE $_C$**	**DevS**	**$NRMSE_T$**	**DevS**	**$NRMSE_{min}$**	**DevS**
4	0.17	0.0	3E-2	3E-3	2E-2	0.70	0.21	3E-3	3E-2	6E-2	3E-2	120.3
5	0.20	2E-4	3E-3	1E-4	3E-3	1.6	0.25	4E-2	2E-2	5E-3	1E-2	267.0
6	0.07	2E-4	2E-3	5E-4	1E-2	2.2	0.21	4E-2	2E-2	5E-3	2E-2	294.3
7	0.07	2E-3	2E-3	1E-3	1E-2	4.2	0.17	1E-3	1E-2	2E-3	1E-2	214.6
8	0.04	2E-3	2E-3	2E-4	2E-2	5.2	0.10	3E-3	1E-2	1E-3	1E-2	329.1
9	0.04	2E-2	2E-3	1E-4	1E-3	5.3	0.08	3E-3	8E-3	6E-4	7E-3	405.7
10	0.03	3E-3	2E-3	4E-4	1E-3	4.7	0.069	1E-2	6E-3	4E-4	5E-3	346.0

Tabla 4.10.*d*): Comparativa de resultados de los algoritmos ECFA y CFA en el caso de WNN

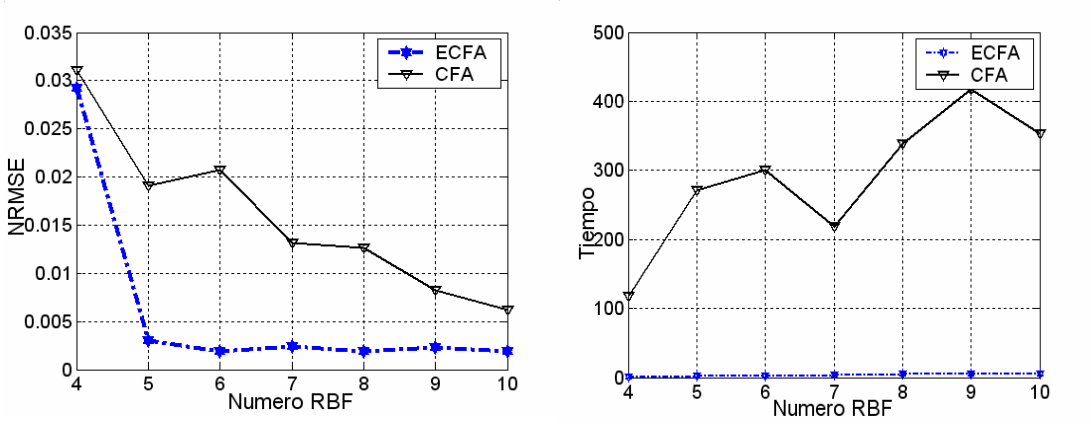


Fig. 4.30 Comparación $NRMSE_T$ y Tiempo de clustering entre ECFA y CFA en el caso WNN

- **Función *nie(x)***

  Esta función $nie(x)$ ha sido utilizado por Nie y Lee [NIE-96] en 1996, donde presentaron tres algoritmos distintos para resolver problemas de aproximación funcional. La Fig. 4.31 muestra la salida objetivo de la función.

$$nie(x) = 3^{-x^2}\, sen(\pi x) \qquad x \in [-3, 3] \tag{4.10}$$

  Esta función has sido utilizada por [POM-00], [GON-01] y [RIV-03] para validar la capacidad de aproximación de sus algoritmos.

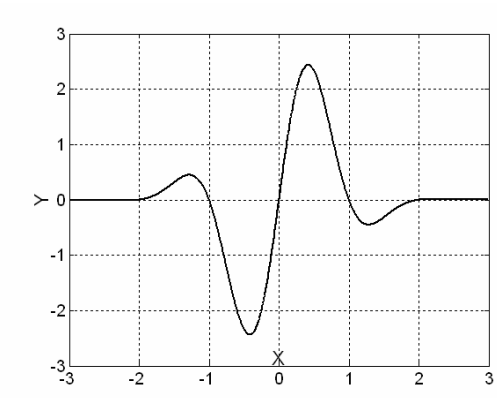


Fig. 4.31 Función objetivo *nie(x)*

	ECFA						CFA					
m	**NRMSE $_C$**	**DevS**	**NRMSE$_T$**	**DevS**	**NRMSE$_{min}$**	**Tiempo$_C$**	**NRMSE $_C$**	**DevS**	**NRMSE$_T$**	**DevS**	**NRMSE$_{min}$**	**DevS**
4	0.94	2E-3	1E-1	1E-4	1E-1	2.5	0.94	1E-2	2E-1	7E-2	1E-1	320.2
5	0.95	3E-4	1E-1	3E-2	1E-1	5.3	0.95	2E-4	1E-2	2E-2	1E-1	15.9
6	0.68	2E-4	7E-3	0.0	7E-3	2.3	0.74	1E-2	4E-2	7E-3	3E-2	170.0
7	0.71	3E-2	1E-2	1E-2	3E-3	5.1	0.71	5E-2	3E-2	6E-3	2E-2	191.4
8	0.27	1E-3	3E-3	6E-4	2E-3	5.3	0.39	3E-2	4E-3	8E-4	4E-3	326.1
9	0.38	4E-3	3E-3	8E-4	2E-3	7.1	0.28	8E-2	2E-3	3E-4	2E-3	307.2
10	0.11	1E-2	1E-3	7E-4	2E-3	8.5	0.18	9E-3	2E-3	2E-4	2E-3	409.5

Tabla 4.11.*a*): Comparativa de resultados de los algoritmos ECFA y CFA en el caso de NRBFN

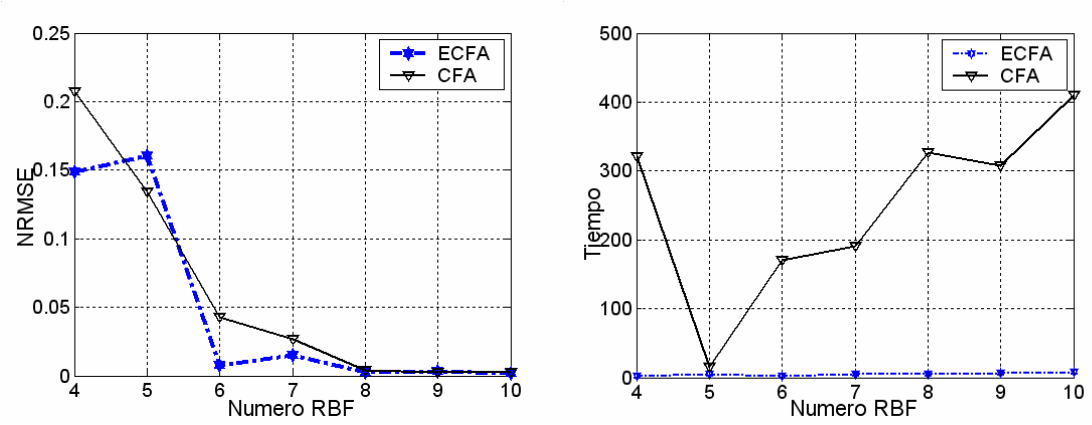


Fig. 4.32 Comparación NRMSE$_T$ y Tiempo de clustering entre ECFA y CFA en el caso NRBFN

	ECFA						CFA					
m	**NRMSE $_C$**	**DevS**	**NRMSE$_T$**	**DevS**	**NRMSE$_{min}$**	**Tiempo$_C$**	**NRMSE $_C$**	**DevS**	**NRMSE$_T$**	**DevS**	**NRMSE$_{min}$**	**Tiempo$_C$**
4	0.86	2E-4	1E-1	1E-3	1E-1	1.8	0.95	3E-3	9E-2	2E-2	6E-2	320.2
5	0.88	6E-2	7E-3	6E-3	5E-3	2.2	0.92	7E-3	3E-2	4E-2	1E-2	15.9
6	0.50	4E-3	2E-3	8E-4	9E-3	3.1	0.69	1E-2	8E-3	4E-3	7E-3	170.0
7	0.63	7E-2	4E-3	2E-3	1E-3	4.6	0.66	6E-2	5E-3	1E-3	3E-3	191.4
8	0.20	7E-2	5E-3	3E-4	1E-3	6.5	0.30	5E-2	4E-3	5E-4	4E-3	326.1
9	0.23	5E-2	3E-3	2E-3	3E-3	7.1	0.28	2E-2	3E-3	9E-4	2E-3	307.2
10	0.16	5E-3	2E-3	3E-4	1E-3	9.8	0.16	8E-3	3E-3	4E-4	3E-3	409.5

Tabla 4.11.*b*): Comparativa de resultados de los algoritmos ECFA y CFA en el caso de RBFN

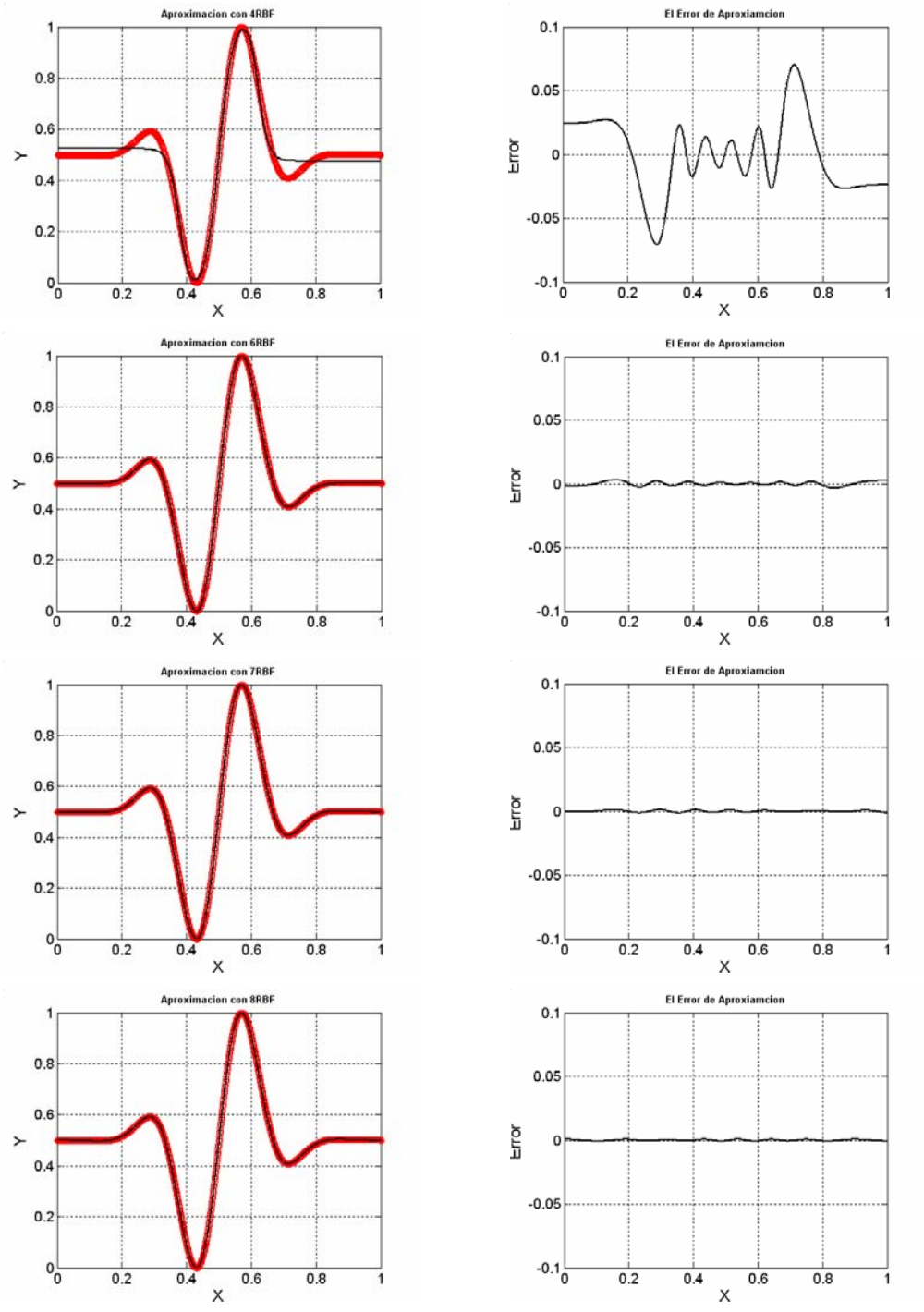


Fig. 4.33 Distintas aproximaciones de la función $nie(x)$

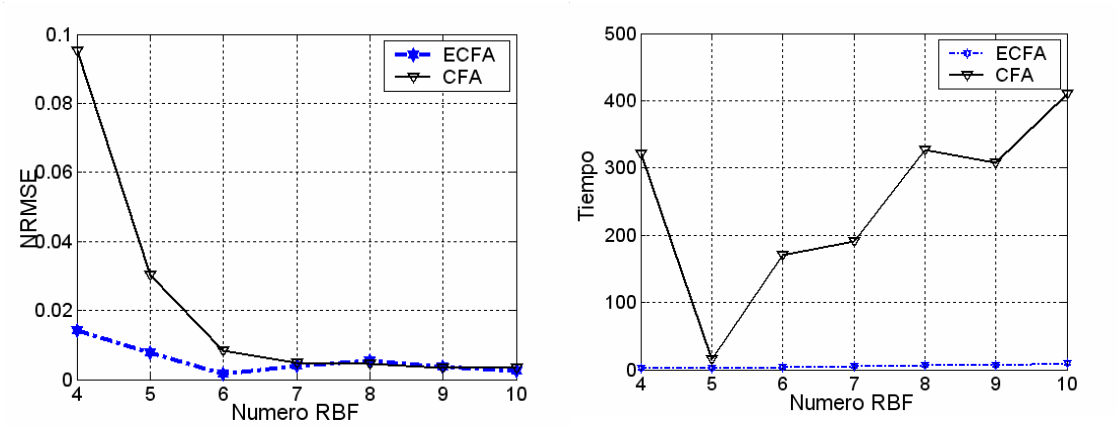


Fig. 4.34 Comparación $NRMSE_T$ y Tiempo de clustering entre ECFA y CFA en el caso RBFN

	ECFA						CFA					
m	**$NRMSE_C$**	**DevS**	**$NRMSE_T$**	**DevS**	**$NRMSE_{min}$**	**$Tiempo_C$**	**$NRMSE_C$**	**DevS**	**$NRMSE_T$**	**DevS**	**$NRMSE_{min}$**	**$Tiempo_C$**
4	0.87	1E-2	9E-2	0.0	9E-2	2.1	0.89	2E-3	2E-2	4E-2	2E-1	320.2
5	0.92	0.0	4E-2	0.0	4E-2	5.5	0.87	2E-2	5E-2	1E-2	3E-2	15.9
6	0.57	2E-3	5E-3	1E-3	4E-3	2.4	0.72	2E-2	2E-2	3E-4	1E-2	170.0
7	0.59	7E-3	4E-3	9E-4	4E-3	6.6	0.54	4E-2	9E-3	4E-4	6E-3	191.4
8	0.19	3E-3	2E-3	4E-4	1E-3	6.1	0.22	3E-2	6E-3	1E-4	4E-3	326.1
9	0.29	1E-3	3E-3	7E-4	2E-3	8.0	0.24	2E-2	4E-3	8E-4	3E-3	307.2
10	0.11	6E-2	1E-3	5E-4	1E-3	9.1	0.18	2E-2	3E-3	5E-4	3E-3	409.5

Tabla 4.11.*c*): Comparativa de resultados de los algoritmos ECFA y CFA en el caso de NWNN

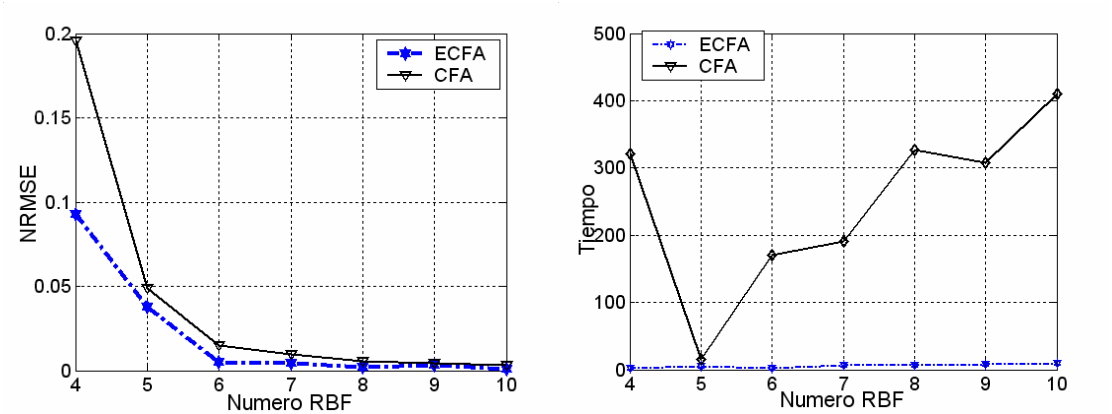


Fig. 4.35 Comparación $NRMSE_T$ y Tiempo de clustering entre ECFA y CFA en el caso NWNN

	ECFA						CFA					
m	**NRMSE $_C$**	**DevS**	**NRMSE$_T$**	**DevS**	**NRMSE$_{min}$**	**Tiempo$_C$**	**NRMSE $_C$**	**DevS**	**NRMSE$_T$**	**DevS**	**NRMSE$_{min}$**	**Tiempo$_C$**
4	0.81	2E-4	1E-2	2E-4	1E-2	6.4	0.84	3E-2	1E-2	2E-4	9E-3	320.2
5	0.83	2E-2	5E-3	4E-4	5E-3	1.7	0.84	3E-4	7E-3	1E-4	6E-3	15.9
6	0.47	4E-4	3E-3	1E-3	3E-3	2.9	0.59	3E-2	4E-3	6E-4	3E-3	170.0
7	0.46	7E-2	2E-3	1E-3	6E-4	4.4	0.56	5E-2	3E-3	5E-4	3E-3	191.4
8	0.15	7E-2	5E-	6E-4	1E-3	5.3	0.26	1E-2	3E-3	5E-4	2E-3	326.1
9	0.21	2E-2	8E-3	1E-4	1E-3	5.0	0.19	4E-2	2E-3	5E-4	2E-3	307.2
10	0.12	4E-2	5E-3	3E-4	1E-3	8.1	0.07	8E-3	2E-3	1E-4	1E-3	409.5

Tabla 4.11.*d*): Comparativa de resultados de los algoritmos ECFA y CFA en el caso de WNN

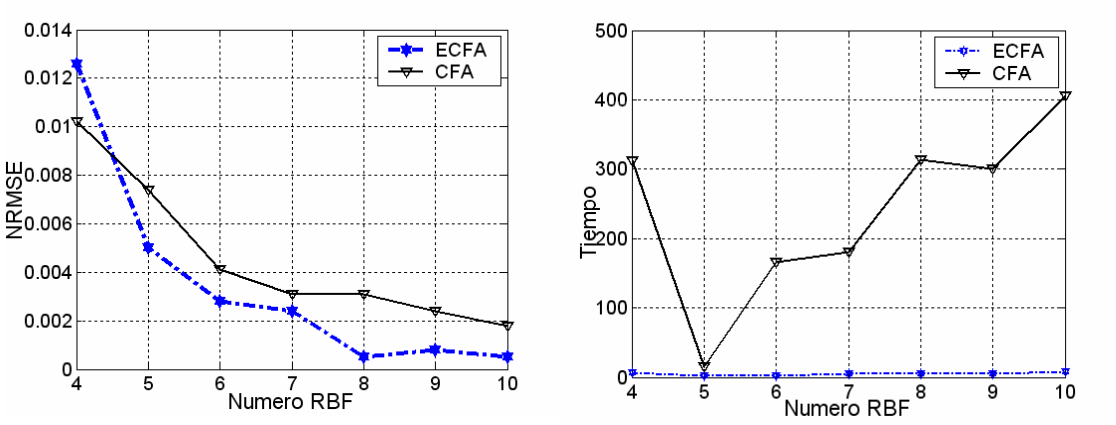


Fig. 4.36 Comparación NRMSE$_T$ y Tiempo de clustering entre ECFA y CFA en el caso WNN

- **Función *dc(x)***

  Esta función $dc(x)$ compara el comportamiento del algoritmo ECFA con otro algoritmo de *clustering* que tiene en cuenta la salida. Esta función se utilizó por [RUN-99] en el algoritmo de estimación de grupos alternante (ACE). La Fig. 4.37 muestra la función:

$$dc(x) = \ 0.2 \ + \ 0.8(x + sen(2\pi x)) \qquad x \in [0\ ,1] \tag{4.11}$$

  Esta función has sido utilizada también por [GON-01], para validar la capacidad de aproximación de su algoritmo CFA.

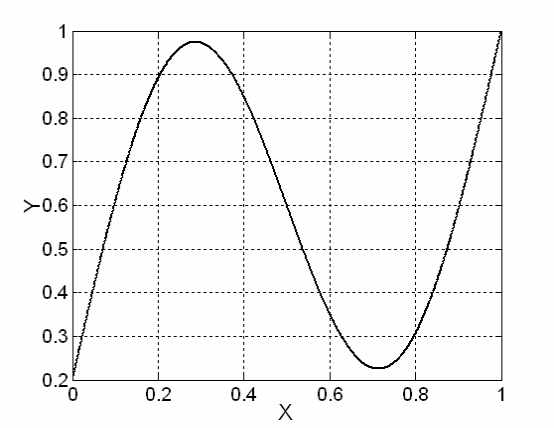


Fig. 4.37 Función objetivo $dc(x)$

	ECFA						CFA					
m	**NRMSE $_C$**	**DevS**	**NRMSE$_T$**	**DevS**	**NRMSE$_{min}$**	**Tiempo$_C$**	**NRMSE $_C$**	**DevS**	**NRMSE$_T$**	**DevS**	**NRMSE$_{min}$**	**Tiempo$_C$**
4	0.05	0.00	4E-4	4E-6	4E-4	1.5	0.40	1E-2	5E-3	5E-3	4E-3	22.8
5	0.14	7E-4	6E-4	2E-5	6E-4	3.1	0.32	2E-2	4E-3	8E-3	3E-3	58.1
6	0.03	2E-3	3E-4	5E-5	3E-4	3.9	0.27	9E-3	3E-3	5E-3	3E-3	141.1
7	0.04	1E-2	7E-4	4E-4	2E-4	11.4	0.30	9E-3	2E-3	2E-3	2E-3	150.3
8	0.02	1E-2	5E-4	3E-4	2E-4	6.1	0.23	1E-2	2E-3	5E-4	1E-3	137.7
9	0.02	4E-3	7E-4	1E-4	5E-4	6.9	0.13	6E-3	1E-3	2E-4	1E-3	53.8
10	0.01	1E-3	5E-4	4E-4	2E-4	12.4	0.12	4E-3	2E-3	3E-4	1E-3	110.9

Tabla 4.12.*a*): Comparativa de resultados de los algoritmos ECFA y CFA en el caso de NRBFN

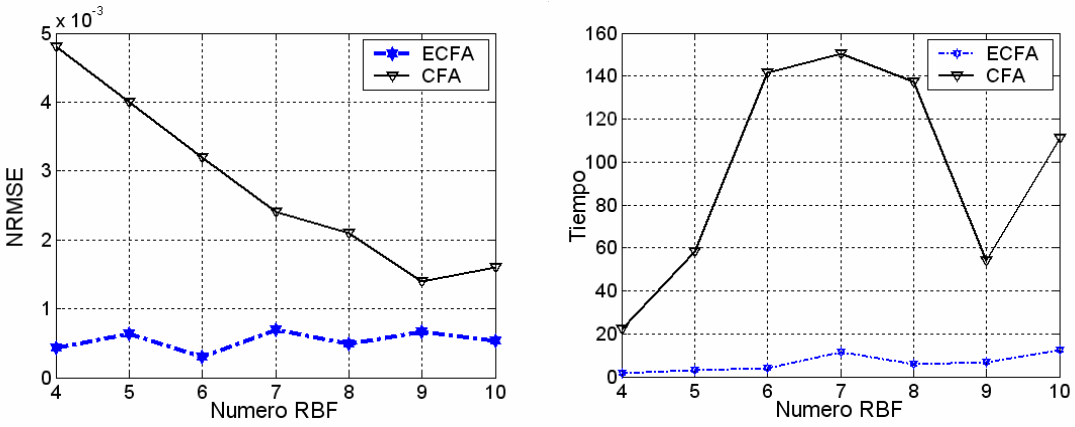


Fig. 4.38 Comparación NRMSE$_T$ y Tiempo de clustering entre ECFA y CFA en el caso NRBFN

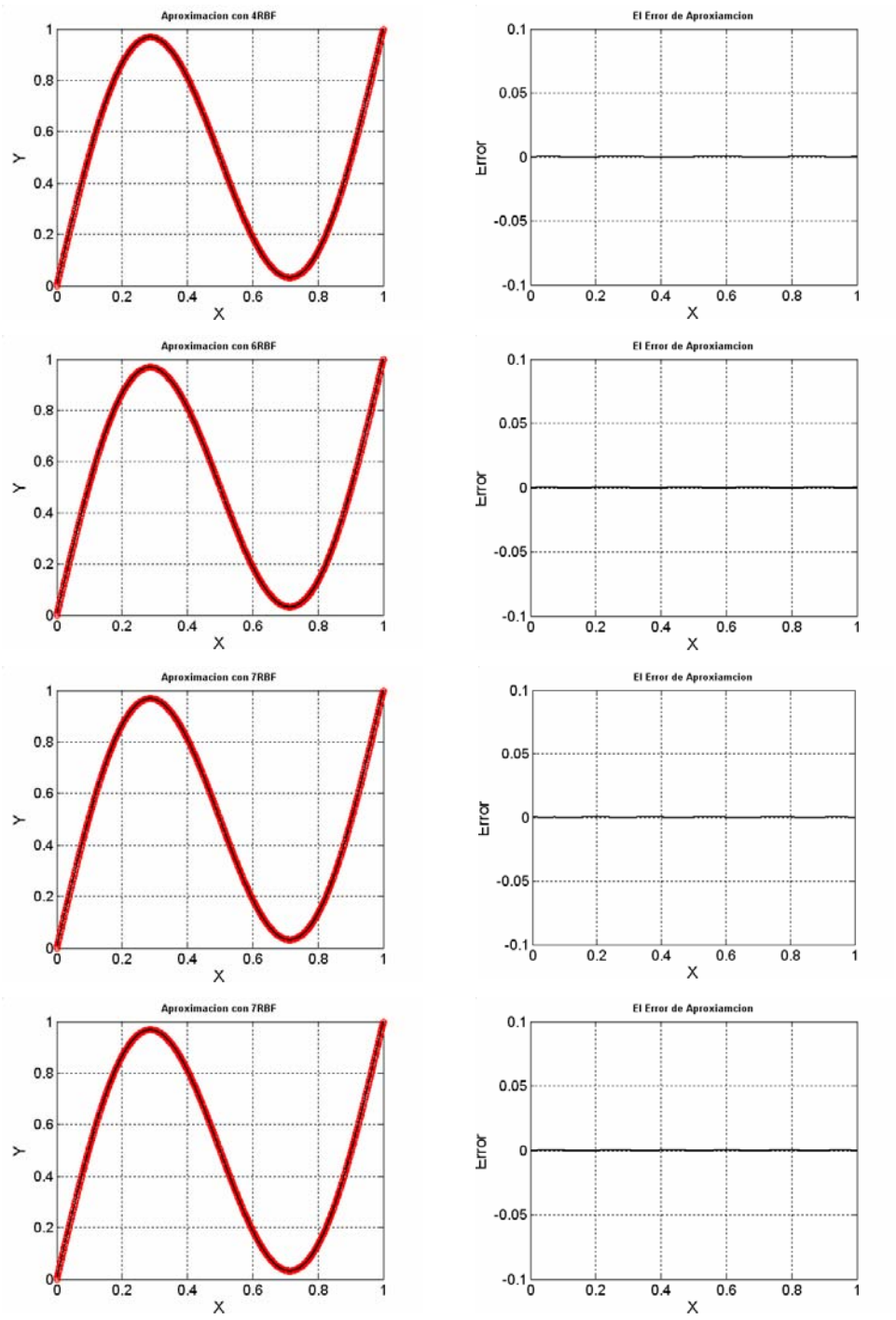


Fig. 4.39 Distintas aproximaciones de la función $dc(x)$

	ECFA						CFA					
m	**NRMSE $_C$**	**DevS**	**NRMSE$_T$**	**DevS**	**NRMSE$_{min}$**	**Tiempo$_C$**	**NRMSE $_C$**	**DevS**	**NRMSE$_T$**	**DevS**	**NRMSE$_{min}$**	**Tiempo$_C$**
4	0.20	2E-2	6E-3	4E-4	2E-4	9.0	0.48	8E-2	5E-3	7E-4	3E-3	22.8
5	0.20	2E-3	1E-3	3E-4	8E-4	0.79	0.45	2E-2	5E-3	4E-4	4E-3	58.1
6	0.11	3E-3	6E-4	5E-4	5E-4	3.6	0.42	5E-2	4E-3	4E-4	3E-3	141.1
7	0.12	2E-3	6E-4	4E-4	4E-4	4.7	0.30	4E-2	4E-3	2E-4	3E-3	150.3
8	0.09	8E-3	2E-3	4E-4	1E-3	5.1	0.24	5E-2	3E-3	1E-4	2E-3	137.7
9	0.08	3E-3	6E-4	4E-4	1E-4	7.2	0.21	2E-2	3E-3	3E-4	2E-3	53.8
10	0.07	2E-3	7E-4	5E-4	3E-4	6.9	0.18	6E-3	2E-3	2E-4	1E-3	110.9

Tabla 4.12.*b*): Comparativa de resultados de los algoritmos ECFA y CFA en el caso de RBFN

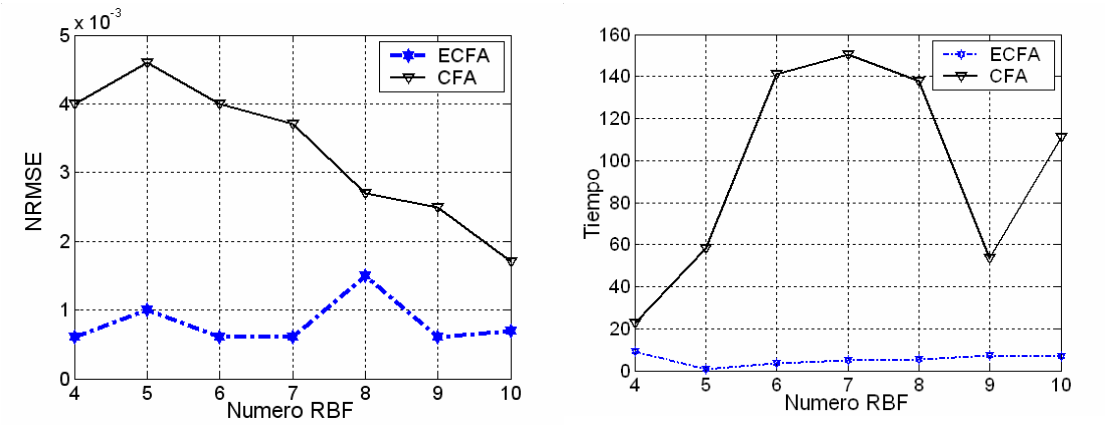


Fig. 4.40 Comparación NRMSE$_T$ y Tiempo de clustering entre ECFA y CFA en el caso RBFN

	ECFA						CFA					
m	**NRMSE $_C$**	**DevS**	**NRMSE$_T$**	**DevS**	**NRMSE$_{min}$**	**Tiempo$_C$**	**NRMSE $_C$**	**DevS**	**NRMSE$_T$**	**DevS**	**NRMSE$_{min}$**	**Tiempo$_C$**
4	0.008	0.0	1E-3	7E-4	1E-3	1.6	0.086	7E-3	5E-3	9E-4	4E-3	22.8
5	0.026	0.0	8E-4	0.0	8E-4	3.1	0.075	1E-3	4E-3	9E-4	3E-3	58.1
6	0.013	1E-2	5E-4	3E-4	3E-4	3.0	0.081	1E-3	4E-3	1E-3	3E-3	141.1
7	0.015	9E-3	6E-4	5E-4	2E-4	11.8	0.068	5E-3	4E-3	8E-4	2E-3	150.3
8	0.006	5E-3	5E-4	3E-4	2E-4	5.8	0.067	5E-3	3E-3	9E-4	1E-3	137.7
9	0.010	9E-3	6E-4	3E-4	4E-4	6.7	0.053	5E-3	2E-3	3E-4	1E-3	53.8
10	0.007	8E-3	4E-4	3E-4	1E-4	13.4	0.035	8E-3	1E-3	1E-3	1E-3	110.9

Tabla 4.12.*c*): Comparativa de resultados de los algoritmos ECFA y CFA en el caso de NWNN

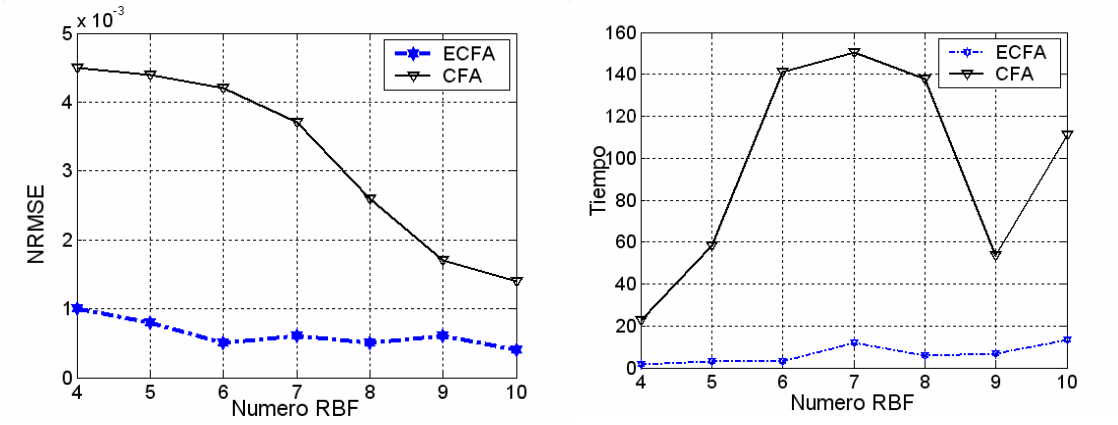


Fig. 4.41 Comparación $NRMSE_T$ y Tiempo de clustering entre ECFA y CFA en el caso NWNN

	ECFA						CFA					
m	**$NRMSE_C$**	**DevS**	**$NRMSE_T$**	**DevS**	**$NRMSE_{min}$**	**$Tiempo_C$**	**$NRMSE_C$**	**DevS**	**$NRMSE_T$**	**DevS**	**$NRMSE_{min}$**	**$Tiempo_C$**
4	0.008	8E-3	7E-4	1E-4	5E-4	8.0	0.034	5E-3	4E-3	6E-5	4E-3	22.8
5	0.001	1E-3	3E-4	3E-4	2E-4	0.89	0.030	4E-3	3E-3	6E-4	2E-3	58.1
6	0.004	4E-3	3E-4	7E-4	3E-4	2.7	0.024	2E-3	2E-3	7E-4	2E-3	141.1
7	0.002	2E-3	9E-5	5E-4	3E-5	4.7	0.014	3E-3	2E-3	1E-4	2E-3	150.3
8	0.001	1E-3	1E-4	4E-4	8E-5	4.6	0.015	2E-3	1E-3	1E-4	1E-3	137.7
9	0.002	2E-3	1E-4	4E-4	1E-4	6.7	0.013	2E-3	1E-3	5E-5	1E-3	53.8
10	0.003	3E-3	2E-4	5E-4	1E-4	9.6	0.014	3E-3	1E-3	1E-4	1E-3	110.9

Tabla 4.12.*d*): Comparativa de resultados de los algoritmos ECFA y CFA en el caso de WNN

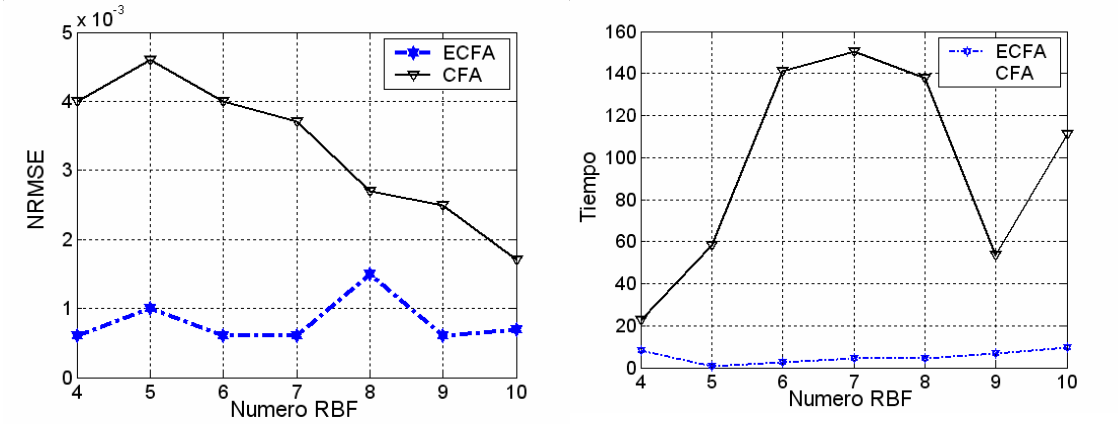


Fig. 4.42 Comparación $NRMSE_T$ y Tiempo de clustering entre ECFA y CFA en el caso WNN

- **Función *pom(x)***

Esta función $pom(x)$ fue utilizada por Pomares en su tesis doctoral [POM-00], para manifestar la influencia de una buena inicialización en su algoritmo. Esta función se describe por la siguiente ecuación:

$$pom(x) = e^{-3x^2} \cdot sen(10\pi\ x) \qquad x \in [0,1] \tag{4.12}$$

La función presenta gran variabilidad en la salida para valores de entrada cercanos de 0 como se muestra en la Fig. 4.43, sin que esta variabilidad se desvanezca por completo para valores altos de x. Un buen algoritmo de *clustering* debería colocar más clusters en la zona donde la variabilidad de la salida es grande, de forma que minimice el error de aproximación.

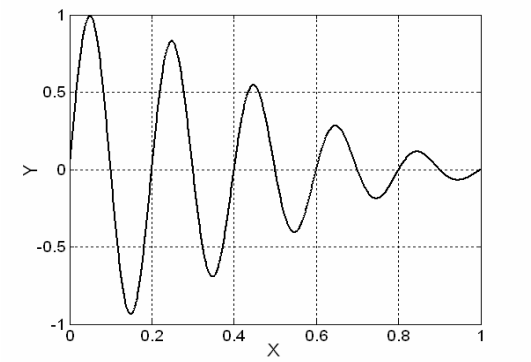


Fig. 4.43 Función objetivo *pom(x)*

	ECFA						CFA					
m	**NRMSE $_C$**	**DevS**	**NRMSE$_T$**	**DevS**	**NRMSE$_{min}$**	**Tiempo$_C$**	**NRMSE $_C$**	**DevS**	**NRMSE$_T$**	**DevS**	**NRMSE$_{min}$**	**Tiempo$_C$**
4	0.97	0.0	0.72	0.0	0.72	2.1	0.97	8E-4	0.72	0.0	0.72	23.4
5	0.96	0.0	0.52	2E-2	0.28	3.3	0.96	8E-4	0.69	5E-2	0.68	9.7
6	0.94	8E-4	0.28	2E-2	0.14	4.8	0.95	3E-3	0.59	6E-2	0.51	70.5
7	0.9	8E-3	0.06	3E-2	0.02	3.8	0.94	6E-3	0.31	3E-2	0.27	123.7
8	0.84	9E-3	0.11	4E-4	0.11	5.5	0.90	3E-2	0.27	3E-2	0.27	110.7
9	0.78	6E-2	0.05	3E-2	0.01	5.2	0.86	3E-2	0.13	2E-2	0.10	121.6
10	0.76	4E-2	0.01	9E-3	0.008	11.4	0.78	2E-2	0.12	6E-2	0.10	143.1

Tabla 4.13.*a*): Comparativa de resultados de los algoritmos ECFA y CFA en el caso NRBFN

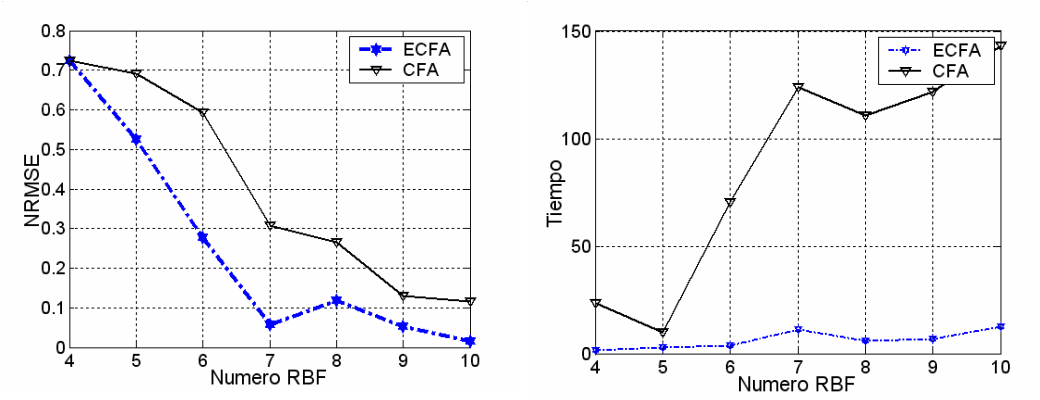


Fig. 4.44 Comparación $NRMSE_T$ y Tiempo de clustering entre ECFA y CFA en el caso NRBFN

	ECFA						CFA					
m	**NRMSE $_C$**	**DevS**	**$NRMSE_T$**	**DevS**	**$NRMSE_{min}$**	**$Tiempo_C$**	**NRMSE $_C$**	**DevS**	**$NRMSE_T$**	**DevS**	**$NRMSE_{min}$**	**$Tiempo_C$**
4	1.07	0.0	0.282	0.0	0.28	1.9	1.0	1E-3	0.591	1E-1	0.28	23.4
5	1.06	2E-3	0.120	0.0	0.12	3.4	1.0	6E-3	0.451	2E-2	0.21	9.7
6	1.02	1E-2	0.075	1E-2	0.01	7.4	1.0	7E-3	0.364	3E-2	0.17	70.5
7	0.92	8E-2	0.062	7E-3	0.01	4.6	0.99	6E-2	0.402	3E-2	0.11	123.7
8	0.76	2E-2	0.016	3E-2	0.006	4.4	0.96	2E-2	0.105	7E-3	0.06	110.7
9	0.51	3E-3	0.003	2E-3	0.001	6.2	0.80	1E-2	0.025	6E-3	0.004	121.6
10	0.54	9E-2	0.004	3E-3	0.004	9.3	0.70	6E-2	0.017	4E-3	0.004	143.1

Tabla 4.13.*c*): Comparativa de resultados de los algoritmos ECFA y CFA en el caso de RBFN

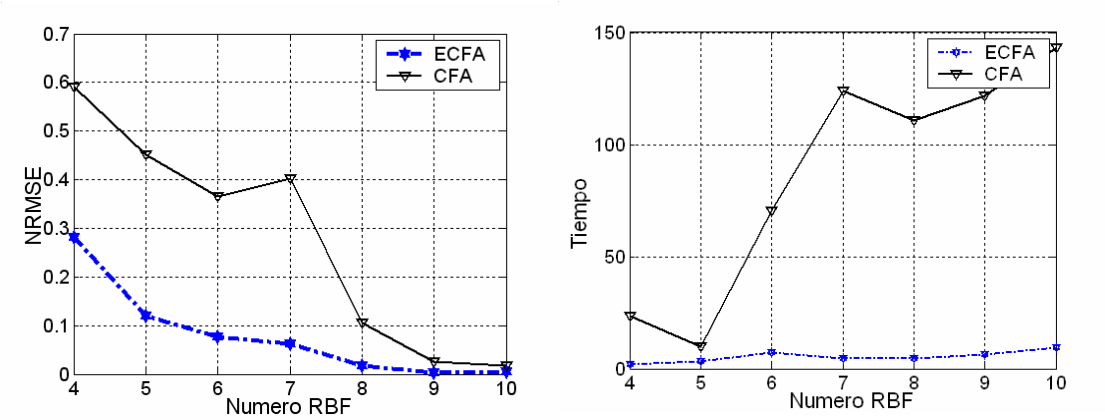


Fig. 4.45 Comparación $NRMSE_T$ y Tiempo de clustering entre ECFA y CFA en el caso RBFN

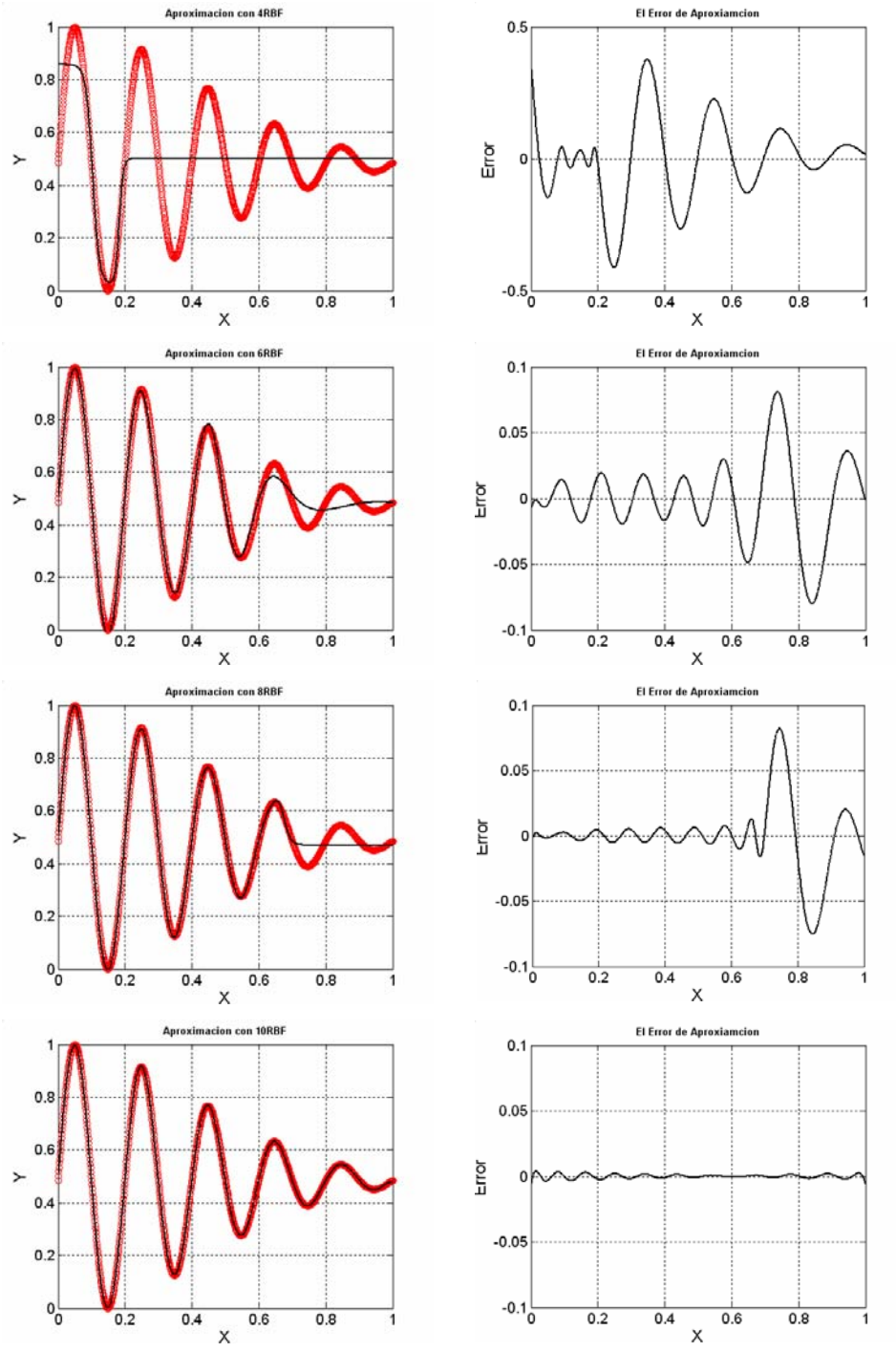


Fig. 4.46 Distintas aproximaciones de la función $pom(x)$

	ECFA						CFA					
m	**NRMSE $_C$**	**DevS**	**NRMSE$_T$**	**DevS**	**NRMSE$_{min}$**	**Tiempo$_C$**	**NRMSE $_C$**	**DevS**	**NRMSE$_T$**	**DevS**	**NRMSE$_{min}$**	**Tiempo$_C$**
4	0.97	0.0	0.551	8E-2	0.49	2.2	0.96	6E-4	0.590	7E-3	0.53	23.4
5	0.94	1E-4	0.485	5E-2	0.45	3.4	0.94	4E-3	0.451	4E-3	0.42	9.7
6	0.94	5E-3	0.419	9E-2	0.33	4.7	0.94	3E-3	0.364	6E-2	0.26	70.5
7	0.90	1E-2	0.030	1E-2	0.02	4.1	0.92	2E-2	0.401	3E-3	0.40	123.7
8	0.82	1E-2	0.021	4E-3	0.02	5.5	0.91	8E-2	0.105	1E-1	0.04	110.7
9	0.82	2E-2	0.011	2E-3	0.009	6.4	0.83	5E-2	0.025	5E-3	0.02	121.6
10	0.64	9E-2	0.006	3E-3	0.006	9.9	0.73	7E-2	0.017	1E-3	0.015	143.1

Tabla 4.13.*c*): Comparativa de resultados de los algoritmos ECFA y CFA en el caso de NWNN

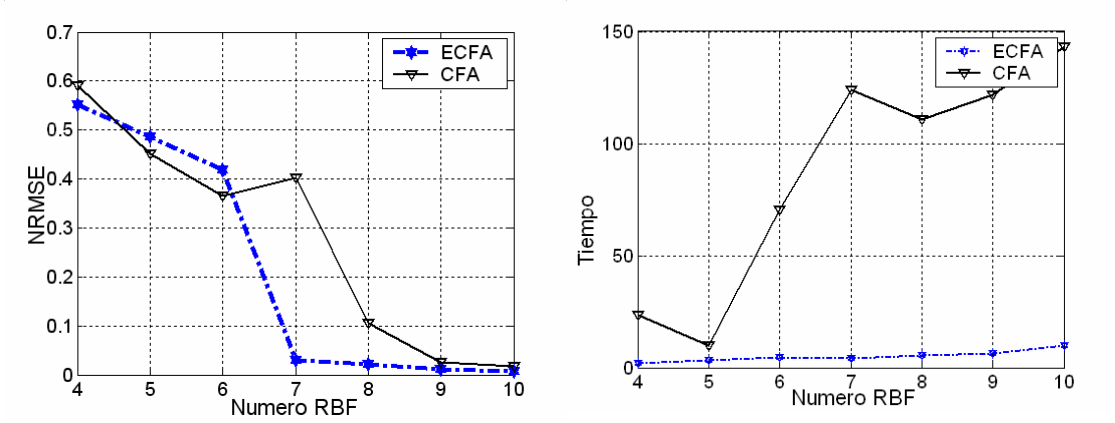


Fig. 4.47 Comparación NRMSE$_T$ y Tiempo de clustering entre ECFA y CFA en el caso NWNN

	ECFA						CFA					
m	**NRMSE $_C$**	**DevS**	**NRMSE$_T$**	**DevS**	**NRMSE$_{min}$**	**Tiempo$_C$**	**NRMSE $_C$**	**DevS**	**NRMSE$_T$**	**DevS**	**NRMSE$_{min}$**	**Tiempo$_C$**
4	0.92	1E-4	0.173	0.0	0.170	1.5	0.92	1E-4	0.200	6E-2	0.175	23.4
5	0.92	3E-3	0.147	1E-2	0.120	2.2	0.92	1E-4	0.246	6E-2	0.174	9.7
6	0.92	6E-3	0.019	1E-2	0.010	5.0	0.90	2E-3	0.134	1E-2	0.123	70.5
7	0.91	5E-3	0.009	8E-4	0.006	4.3	0.90	3E-3	0.098	6E-2	0.071	123.7
8	0.67	2E-2	0.007	2E-3	0.005	4.2	0.85	9E-2	0.041	4E-2	0.026	110.7
9	0.46	1E-2	0.003	1E-3	0.001	7.7	0.75	2E-2	0.015	2E-2	0.012	121.6
10	0.53	3E-2	0.003	7E-4	0.001	9.5	0.54	2E-2	0.015	1E-2	0.013	143.1

Tabla 4.13.d): Comparativa de resultados de los algoritmos ECFA y CFA en el caso de WNN

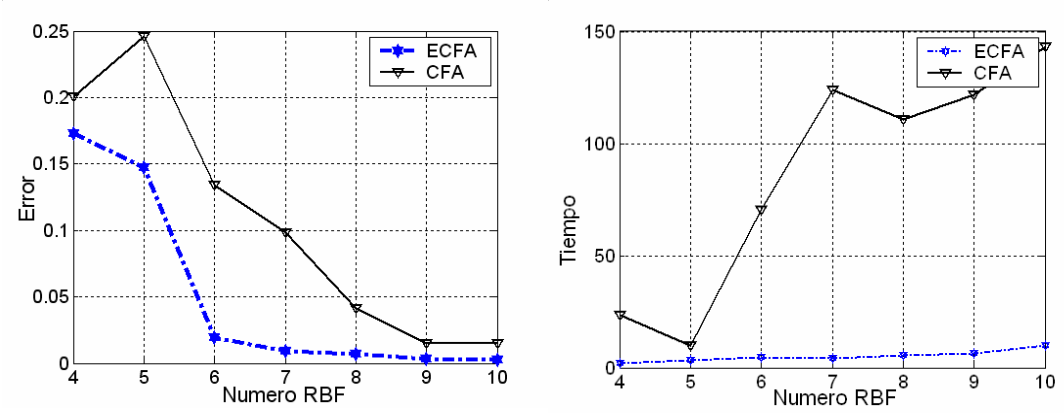


Fig. 4.48 Comparación $NRMSE_T$ y Tiempo de clustering entre ECFA y CFA en el caso WNN

4.4.2 Funciones de dos dimensiones

En esta parte se utilizan diversas funciones en dos dimensiones ampliamente usadas en la bibliografía relacionada con la aproximación funcional. Además de mostrar los resultados que obtiene el algoritmo propuesto ECFA, se compara estos resultados con los resultados obtenidos por el algoritmo CFA aplicado a estas funciones de dos dimensiones, que ya demostró ser el mejor en las comparaciones [GON-01], [GON-02]. La comparación se realiza considerando los valores del error cuadrático medio normalizado (NRMSE). Este error se mide después del proceso de *clustering* como se explica en el Capítulo 3, y después del proceso de optimización local, utilizando algoritmos tradicionales como SVD, Knn y *Levenberg-Marquardt*, para todos los ejemplos. También se compara el tiempo de ejecución para que el algoritmo converja. El número de funciones base (número de clusters) en estos experimentos empieza con número mínimo de 4 clusters y se aumenta de forma ordenada hasta que llega a un número suficiente de funciones base para aproximar una función.

En estas funciones de dos dimensiones se utiliza un conjunto de entrenamiento formado por 441 puntos distribuidos en una rejilla de 21 × 21 celdas en el dominio de entrada. La comparación de las resultados se utiliza el algoritmo CFA [GON-02] con valor de μ_{min} = 0.001 (Capítulo 3, Sección 3.4.3).

- **Función** $g(x_1, x_2)$

  Esta función $g(x_1, x_2)$ se utilizó por [PED-98], en su algoritmo de *clustering* difuso condicional (CFC), y por [GON-02] para comprobar sus resultados en el algoritmo CFA. Esta función se define por esta ecuación:

$$g(x_1, x_2) = [(x_1 - 2)(2x_1 + 1)]/(1 + x_1^2) \cdot [(x_2 - 2)(2x_2 + 1)]/(1 + x_{21}^2) \qquad x_1, x_2 \in [-5, 5] \quad (4.13)$$

  En la Fig. 4.49 se muestra la función $g(x_1, x_2)$, un conjunto de entrenamiento formado por 441 puntos distribuidos en una rejilla de 21 × 21 celdas en el dominio de la entrada.

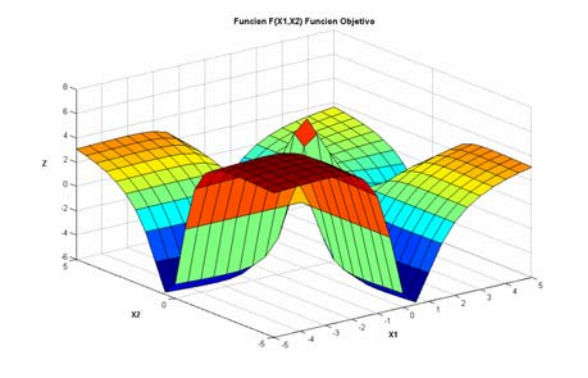


Fig. 4.49 Función objetivo $g(x_1, x_2)$

	ECFA						CFA					
m	**NRMSE $_C$**	**DevS**	**NRMSE$_T$**	**DevS**	**NRMSE$_{min}$**	**Tiempo$_C$**	**NRMSE $_C$**	**DevS**	**NRMSE$_T$**	**DevS**	**NRMSE$_{min}$**	**Tiempo$_C$**
4	0.96	3E-3	0.61	9E-2	0.37	0.5	0.94	1E-2	0.53	7E-2	0.49	59.51
5	0.87	4E-2	0.50	6E-2	0.38	0.7	0.91	4E-2	0.42	6E-2	0.38	23.1
6	0.73	5E-2	0.29	7E-4	0.27	1.5	0.86	9E-3	0.42	6E-2	0.36	61.7
7	0.71	1E-2	0.27	2E-2	0.22	1.7	0.82	4E-2	0.36	2E-2	0.34	36.7
8	0.74	6E-3	0.15	4E-2	0.13	2.0	0.77	4E-2	0.34	4E-2	0.28	34.0
9	0.63	6E-2	0.12	5E-2	0.11	4.4	0.73	1E-2	0.31	6E-2	0.24	26.6
10	0.68	2E-2	0.07	5E-2	0.05	5.7	0.73	2E-2	0.23	3E-2	0.19	24.1

Tabla 4.14.*a*): Comparativa de resultados de los algoritmos ECFA y CFA en el caso NRBFN

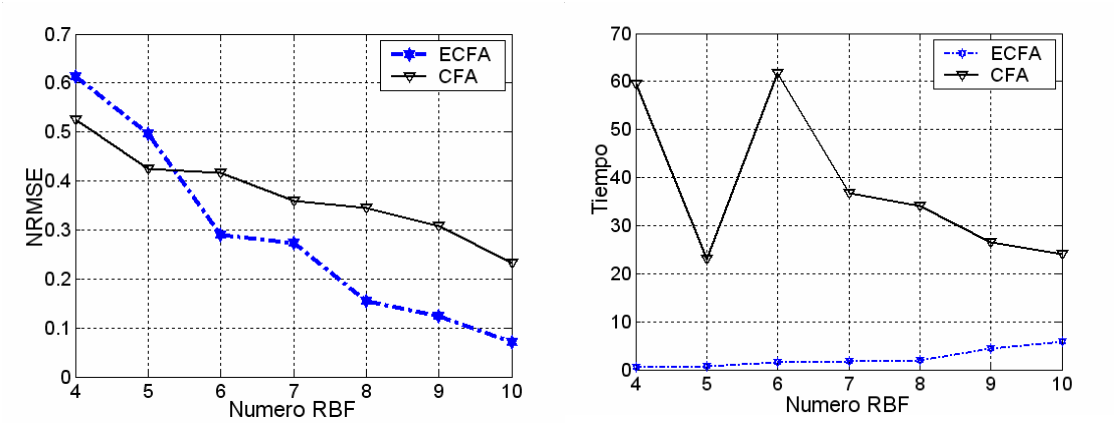


Fig. 4.50 Comparación $NRMSE_T$ y Tiempo de clustering entre ECFA y CFA en el caso NRBFN

	ECFA						CFA					
m	**NRMSE $_C$**	**DevS**	**$NRMSE_T$**	**DevS**	**$NRMSE_{min}$**	**$Tiempo_C$**	**NRMSE $_C$**	**DevS**	**$NRMSE_T$**	**DevS**	**$NRMSE_{min}$**	**$Tiempo_C$**
4	1.1	5E-3	0.65	1E-3	0.60	0.48	1.1	2E-2	0.69	1E-2	0.60	59.51
5	1.0	6E-2	0.56	2E-3	0.48	0.87	1.0	2E-2	0.55	4E-2	0.48	23.1
6	0.80	1E-2	0.46	5E-2	0.46	0.91	0.89	1E-2	0.51	3E-2	0.46	61.7
7	0.73	1E-2	0.39	2E-2	0.36	1.1	0.81	1E-2	0.38	2E-2	0.36	36.7
8	0.69	3E-2	0.34	2E-2	0.32	2.0	0.76	9E-2	0.33	3E-2	0.30	34.0
9	0.66	4E-2	0.28	3E-2	0.28	2.3	0.65	2E-2	0.31	5E-2	0.28	26.6
10	0.64	7E-2	0.24	3E-2	0.22	5.1	0.65	2E-2	0.28	8E-3	0.26	24.1

Tabla 4.14.*b*): Comparativa de resultados de los algoritmos ECFA y CFA en el caso RBFN

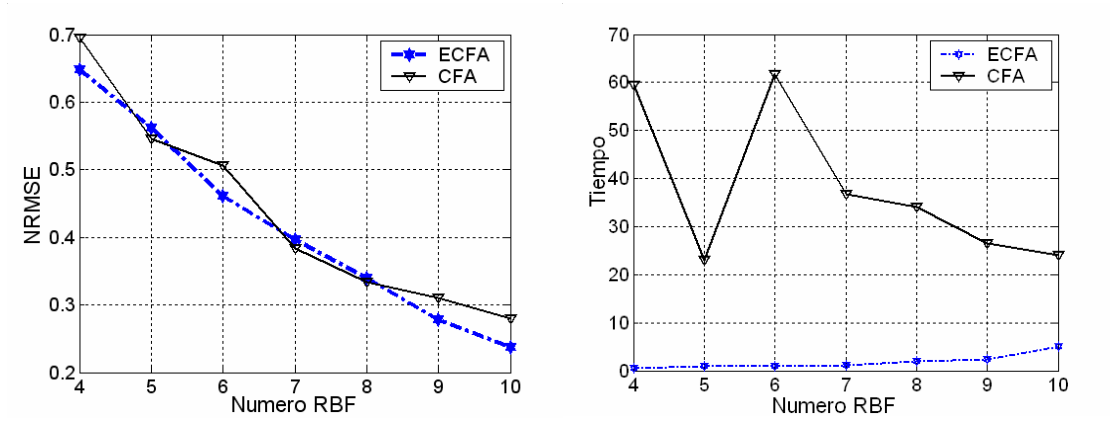


Fig. 4.51 Comparación $NRMSE_T$ y Tiempo de clustering entre ECFA y CFA en el caso RBFN

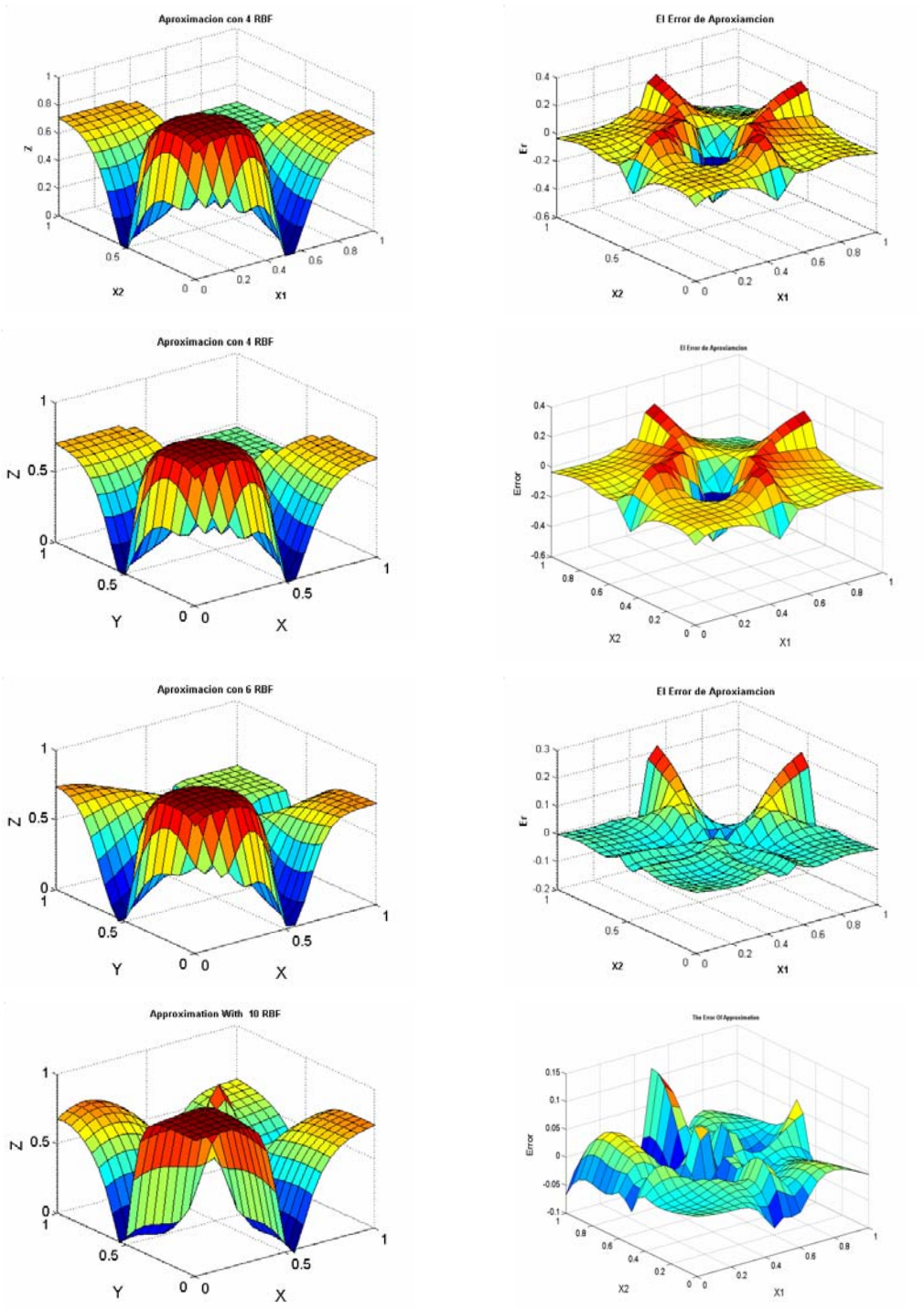


Fig. 4.52 Distintas aproximaciones de la función $g(x_1, x_2)$

	ECFA						CFA					
m	**NRMSE $_C$**	**DevS**	**NRMSE$_T$**	**DevS**	**NRMSE$_{min}$**	**Tiempo$_C$**	**NRMSE $_C$**	**DevS**	**NRMSE$_T$**	**DevS**	**NRMSE$_{min}$**	**Tiempo$_C$**
4	0.91	4E-2	0.46	4E-2	0.42	0.47	0.89	3E-2	0.47	3E-2	0.42	59.51
5	0.88	3E-2	0.37	6E-2	0.35	0.81	0.84	8E-2	0.51	5E-2	0.50	23.1
6	0.76	3E-2	0.33	4E-2	0.31	1.1	0.86	4E-3	0.45	6E-2	0.39	61.7
7	0.70	3E-2	0.25	6E-2	0.23	1.5	0.82	4E-2	0.35	5E-2	0.31	36.7
8	0.73	7E-2	0.21	6E-2	0.05	1.9	0.74	5E-2	0.28	4E-2	0.23	34.0
9	0.64	5E-2	0.12	6E-2	0.10	4.5	0.69	2E-2	0.24	4E-2	0.22	26.6
10	0.62	9E-2	0.05	3E-2	0.04	4.4	0.72	3E-2	0.19	2E-2	0.17	24.1

Tabla 4.14.*c*): Comparativa de resultados de los algoritmos ECFA y CFA en el caso NWNN

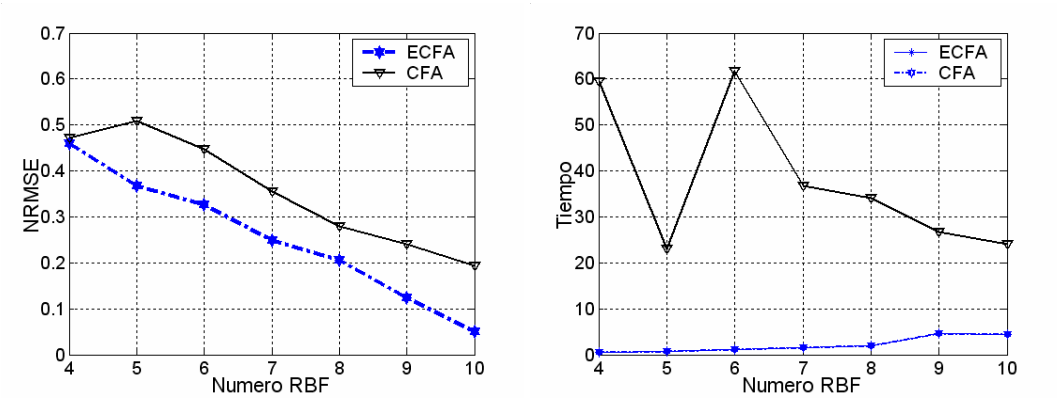


Fig. 4.53 Comparación NRMSE$_T$ y Tiempo de clustering entre ECFA y CFA en el caso NWNN

	ECFA						CFA					
m	**NRMSE $_C$**	**DevS**	**NRMSE$_T$**	**DevS**	**NRMSE$_{min}$**	**Tiempo$_C$**	**NRMSE $_C$**	**DevS**	**NRMSE$_T$**	**DevS**	**NRMSE$_{min}$**	**Tiempo$_C$**
4	0.85	4E-4	0.48	0.0	0.47	0.42	0.86	9E-3	0.48	0.0	0.47	59.51
5	0.83	4E-3	0.44	3E-2	0.42	0.60	0.85	4E-3	0.45	3E-2	0.43	23.1
6	0.75	1E-2	0.39	2E-2	0.36	1.2	0.85	6E-3	0.42	4E-2	0.37	61.7
7	0.66	4E-2	0.33	2E-2	0.31	2.1	0.79	6E-2	0.39	9E-3	0.37	36.7
8	0.63	2E-2	0.30	4E-3	0.27	2.1	0.68	8E-2	0.35	5E-2	0.25	34.0
9	0.62	2E-2	0.24	2E-2	0.20	2.8	0.58	9E-3	0.29	9E-3	0.27	26.6
10	0.60	3E-2	0.22	8E-3	0.20	3.3	0.59	2E-2	0.28	4E-2	0.25	24.1

Tabla 4.14.*d*): Comparativa de resultados de los algoritmos ECFA y CFA en el caso WNN

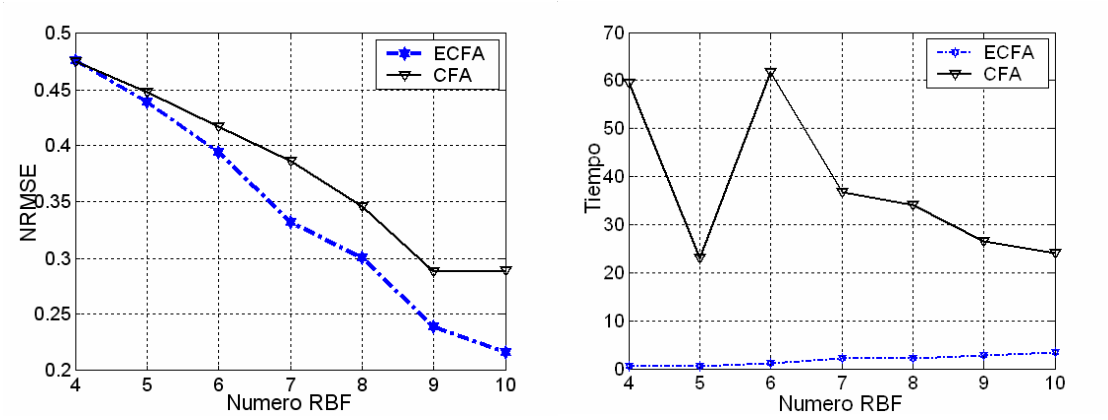


Fig. 4.54 Comparación $NRMSE_T$ y Tiempo de clustering entre ECFA y CFA en el caso NWNN

- **Función** $f_8(x_1, x_2)$

  La Fig. 4.55 muestra la función $f_8(x_1, x_2)$, esta función se define por la ecuación:

$$f_8(x_1, x_2) = sen\left(2\pi\sqrt{x_1^2 + x_2^2}\right), \quad x_1, x_2 \in [-1, 1] \tag{4.14}$$

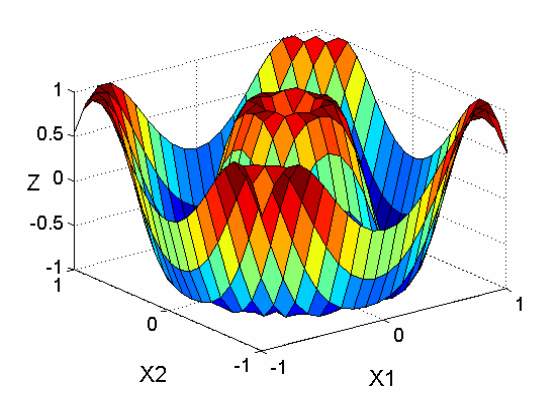


Fig. 4.55 Función objetivo de la función $f_8(x_1, x_2)$

	ECFA						CFA					
m	**NRMSE $_C$**	**DevS**	**NRMSE$_T$**	**DevS**	**NRMSE$_{min}$**	**Tiempo$_C$**	**NRMSE $_C$**	**DevS**	**NRMSE$_T$**	**DevS**	**NRMSE$_{min}$**	**Tiempo$_C$**
4	1.0	0.0	0.91	7E-3	0.90	0.62	0.99	0.0	0.93	4E-3	0.92	13.6
5	0.97	1E-4	0.16	3E-2	0.13	1.3	0.98	1E-3	0.29	6E-2	0.21	12.9
6	0.98	1E-3	0.14	3E-2	0.11	0.97	0.97	3E-3	0.25	5E-2	0.18	17.6
7	0.97	5E-3	0.15	4E-2	0.11	1.6	0.97	1E-4	0.24	4E-2	0.18	32.6
8	0.97	6E-3	0.16	5E-2	0.10	2.3	0.96	2E-4	0.23	2E-2	0.23	14.6
9	0.83	8E-3	0.03	3E-3	0.02	3.7	0.86	3E-2	0.13	2E-2	0.13	35.5
10	0.87	4E-2	0.04	3E-2	0.01	3.7	0.85	1E-2	0.11	1E-2	0.10	89.8

Tabla 4.15.*a*): Comparativa de resultados de los algoritmos ECFA y CFA en el caso NRBFN

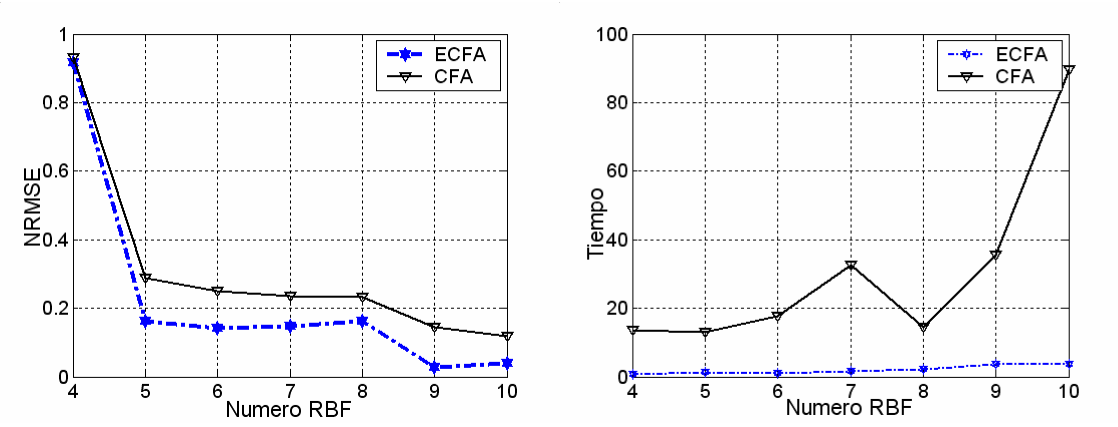


Fig. 4.56 Comparación NRMSE$_T$ y Tiempo de clustering entre ECFA y CFA en el caso NRBFN

	ECFA						CFA					
m	**NRMSE $_C$**	**DevS**	**NRMSE$_T$**	**DevS**	**NRMSE$_{min}$**	**Tiempo$_C$**	**NRMSE $_C$**	**DevS**	**NRMSE$_T$**	**DevS**	**NRMSE$_{min}$**	**Tiempo$_C$**
4	1.05	0.0	0.93	6E-2	0.87	0.34	1.06	4E-4	0.88	1E-2	0.86	13.6
5	1.05	1E-5	0.74	2E-2	0.45	0.70	1.04	7E-4	0.73	1E-2	0.57	12.9
6	1.03	2E-4	0.47	7E-2	0.38	0.80	1.04	9E-4	0.53	8E-2	0.39	17.6
7	1.01	8E-3	0.38	3E-2	0.35	1.7	1.05	2E-3	0.46	7E-2	0.45	32.6
8	1.0	1E-3	0.29	4E-2	0.24	1.5	1.04	7E-2	0.45	8E-2	0.36	14.6
9	0.89	9E-2	0.28	6E-2	0.2	4.5	0.91	4E-2	0.39	5E-2	0.35	35.5
10	0.85	6E-2	0.25	5E-2	0.17	5.5	0.94	3E-2	0.31	5E-2	0.25	89.8

Tabla 4.15.*b*): Comparativa de resultados de los algoritmos ECFA y CFA en el caso RBFN

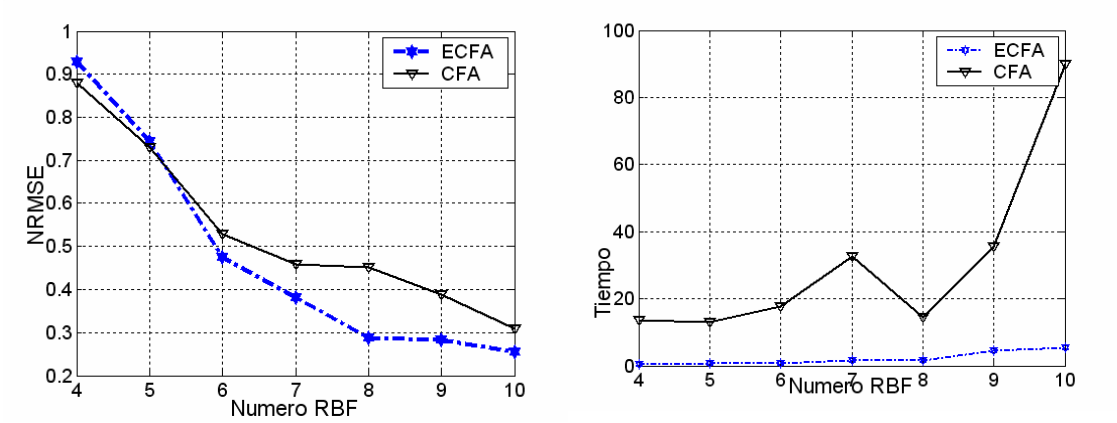


Fig. 4.57 Comparación $NRMSE_T$ y Tiempo de clustering entre ECFA y CFA en el caso RBFN

	ECFA						CFA					
m	**NRMSE $_C$**	**DevS**	**$NRMSE_T$**	**DevS**	**$NRMSE_{min}$**	**$Tiempo_C$**	**NRMSE $_C$**	**DevS**	**$NRMSE_T$**	**DevS**	**$NRMSE_{min}$**	**$Tiempo_C$**
4	0.99	0.0	0.78	7E-3	0.67	0.53	0.99	6E-3	0.50	4E-2	0.47	13.6
5	0.99	1E-3	0.54	2E-2	0.42	0.76	0.98	4E-3	0.46	6E-2	0.38	12.9
6	0.99	1E-2	0.32	2E-2	0.21	1.1	0.98	2E-3	0.31	4E-2	0.26	17.6
7	0.97	2E-3	0.11	3E-3	0.10	0.94	0.97	2E-3	0.33	4E-2	0.30	32.6
8	0.97	1E-2	0.10	2E-2	0.09	2.1	0.96	3E-3	0.27	4E-2	0.21	14.6
9	0.82	2E-3	0.04	3E-2	0.03	2.6	0.86	7E-2	0.14	1E-2	0.12	35.5
10	0.76	2E-2	0.03	1E-2	0.02	3.4	0.79	6E-2	0.11	1E-2	0.09	89.8

Tabla 4.15.*c*): Comparativa de resultados de los algoritmos ECFA y CFA en el caso NWNN

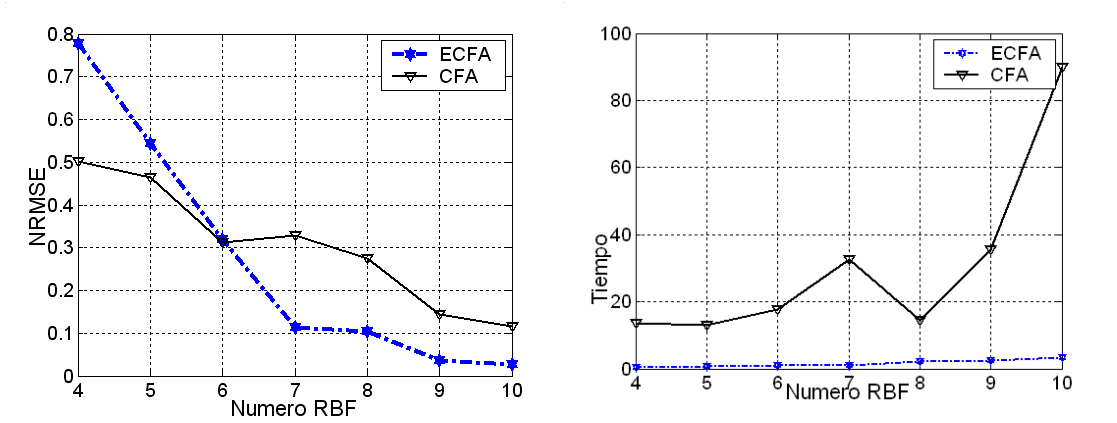


Fig. 4.58 Comparación $NRMSE_T$ y Tiempo de clustering entre ECFA y CFA en el caso NWNN

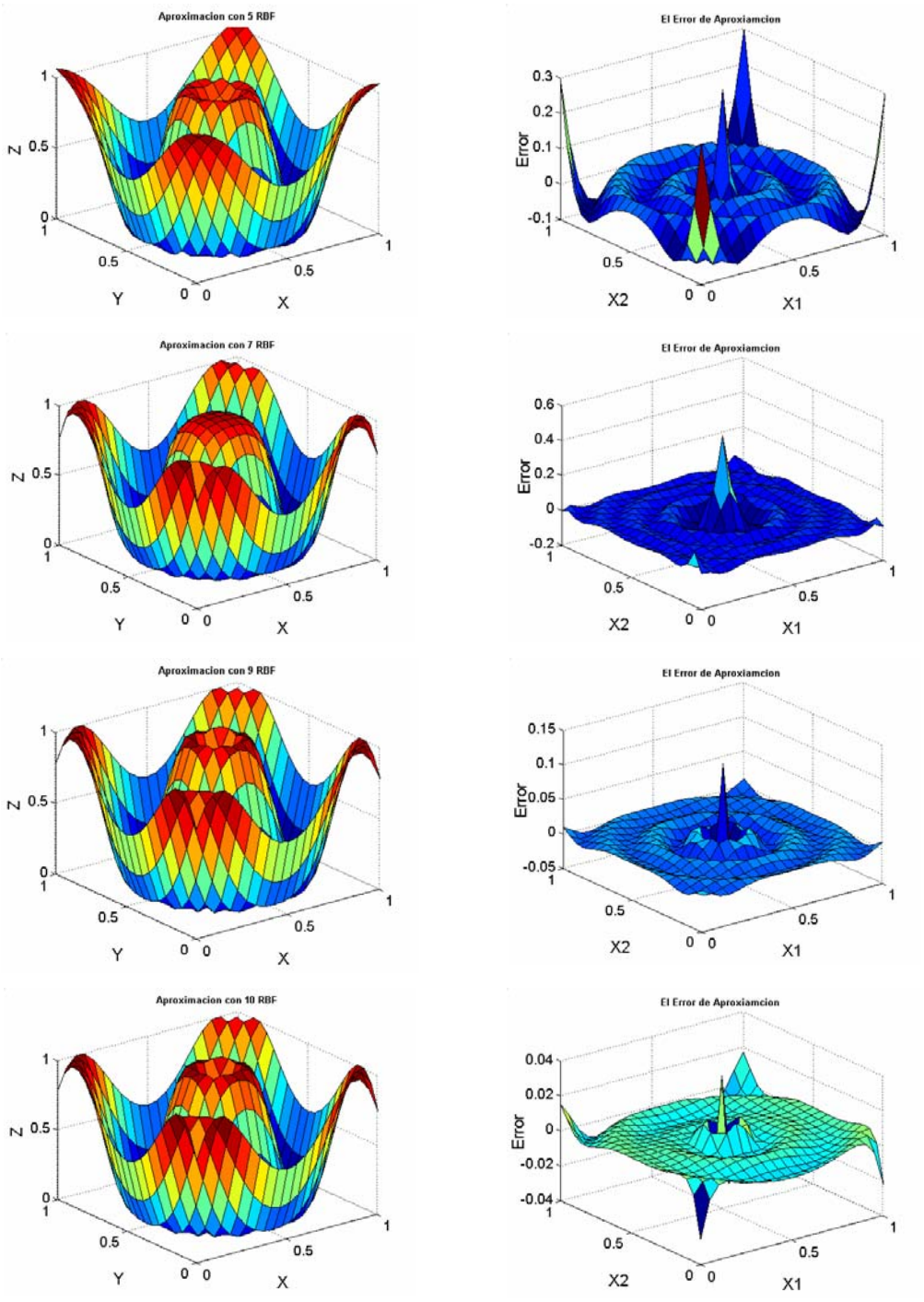


Fig. 4.59 Distintas aproximaciones de la función $f_8(x_1, x_2)$

	ECFA						CFA					
m	**NRMSE $_C$**	**DevS**	**NRMSE$_T$**	**DevS**	**NRMSE$_{min}$**	**Tiempo$_C$**	**NRMSE $_C$**	**DevS**	**NRMSE$_T$**	**DevS**	**NRMSE$_{min}$**	**Tiempo$_C$**
4	0.98	0.0	0.76	5E-2	0.70	0.34	0.98	1E-4	0.78	4E-2	0.76	13.6
5	0.57	2E-4	0.39	0.0	0.39	0.60	0.87	1E-2	0.32	6E-2	0.26	12.9
6	0.91	4E-2	0.41	6E-2	0.37	0.94	0.87	4E-2	0.40	5E-2	0.24	17.6
7	0.68	8E-2	0.28	2E-2	0.25	0.93	0.84	4e-3	0.25	1E-2	0.25	32.6
8	0.67	2E-2	0.27	2E-2	0.24	1.4	0.70	2E-3	0.22	5E-2	0.16	14.6
9	0.49	1E-2	0.25	3E-2	0.22	3.3	0.55	2E-2	0.14	1E-2	0.12	35.5
10	0.50	3E-2	0.14	2E-2	0.11	3.7	0.52	2E-2	0.17	2E-2	0.16	89.8

Tabla 4.15.*d*): Comparativa de resultados de los algoritmos ECFA y CFA en el caso WNN

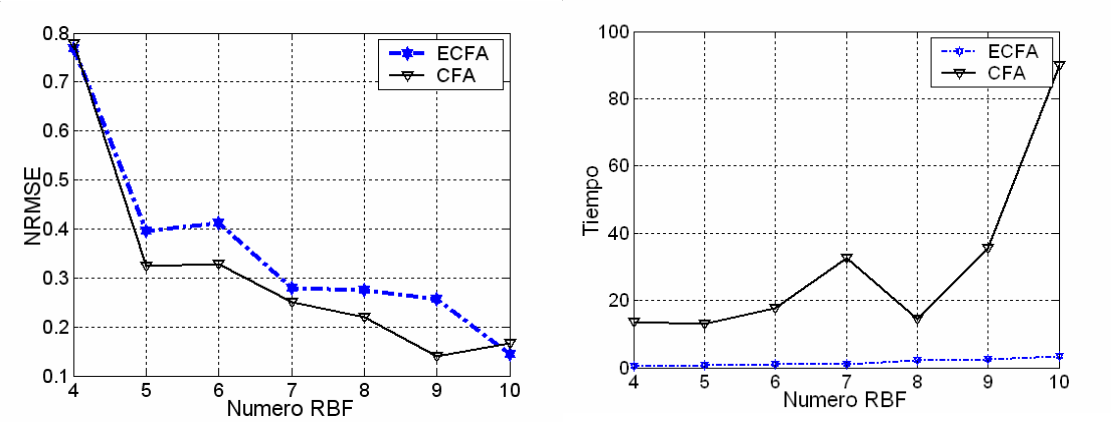


Fig. 4.60 Comparación NRMSE$_T$ y Tiempo de clustering entre ECFA y CFA en el caso WNN

- **Función** $f_1(x_1, x_2)$

  La Fig. 4.61 muestra la función $f_1(x_1, x_2)$, esta función se define por la ecuación:

$$f_1(x_1, x_2) = sen(x_1 x_2) \ , \quad x_1, x_2 \in [-2, 2] \tag{4.15}$$

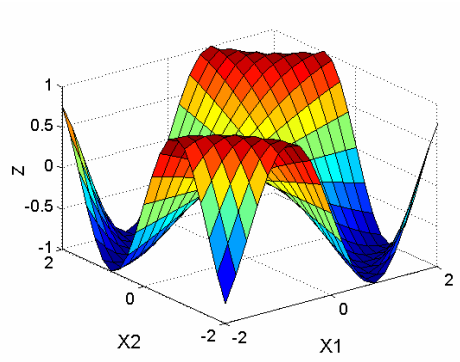


Fig. 4.61 Función objetivo de la función $f_1(x_1, x_2)$

	ECFA						CFA					
m	**NRMSE $_C$**	**DevS**	**NRMSE$_T$**	**DevS**	**NRMSE$_{min}$**	**Tiempo$_C$**	**NRMSE $_C$**	**DevS**	**NRMSE$_T$**	**DevS**	**NRMSE$_{min}$**	**Tiempo$_C$**
4	0.75	0.0	0.48	0.0	0.48	1.3	0.75	2E-4	0.43	2E-4	0.43	14.1
5	0.72	1E-2	0.35	3E-2	0.31	0.61	0.73	3E-3	0.45	2E-2	0.42	13.7
6	0.69	7E-3	0.37	5E-3	0.30	0.80	0.72	7E-3	0.34	3E-2	0.31	18.7
7	0.69	5E-3	0.30	8E-2	0.21	0.99	0.70	2E-3	0.32	4E-2	0.03	25.4
8	0.69	8E-3	0.03	1E-2	0.02	1.9	0.71	7E-3	0.24	3E-2	0.21	26.9
9	0.70	3E-2	0.03	5E-4	0.03	2.5	0.71	2E-2	0.19	2E-2	0.11	34.5
10	0.60	1E-2	0.02	1E-2	0.01	3.2	0.70	5E-3	0.15	3E-2	0.11	33.2

Tabla 4.16.*a*): Comparativa de resultados de los algoritmos ECFA y CFA en el caso NRBFN

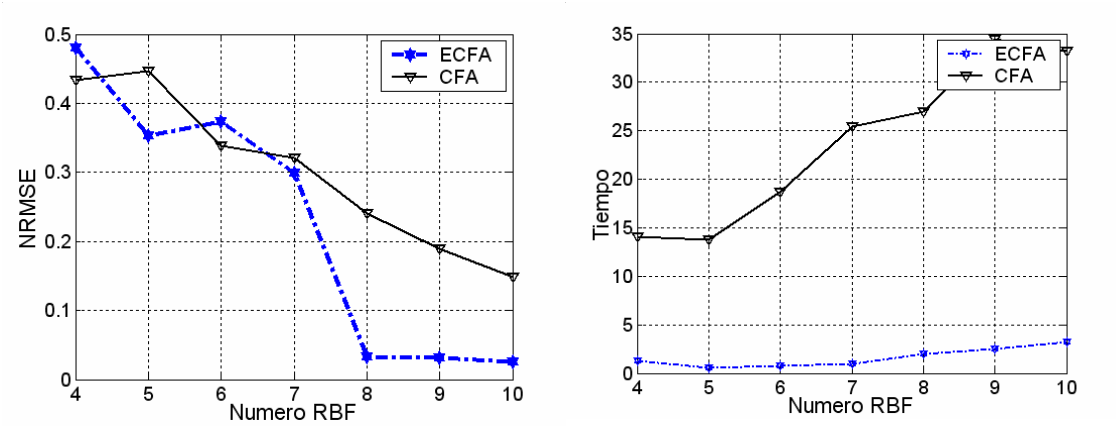


Fig. 4.62 Comparación NRMSE$_T$ y Tiempo de clustering entre ECFA y CFA en el caso NRBFN

	ECFA						CFA					
m	**NRMSE $_C$**	**DevS**	**NRMSE$_T$**	**DevS**	**NRMSE$_{min}$**	**Tiempo$_C$**	**NRMSE $_C$**	**DevS**	**NRMSE$_T$**	**DevS**	**NRMSE$_{min}$**	**Tiempo$_C$**
4	0.71	6E-4	0.36	5E-3	0.35	0.45	0.70	9E-3	0.38	4E-2	0.360.06	14.1
5	0.67	3E-2	0.32	5E-2	0.27	0.60	0.66	1E-2	0.51	6E-2	0.48	13.7
6	0.63	4E-2	0.28	5E-2	0.23	0.78	0.62	9E-3	0.32	9E-2	0.22	18.7
7	0.59	2E-2	0.23	4E-2	0.20	1.1	0.59	1E-2	0.36	5E-2	0.30	25.4
8	0.60	1E-2	0.10	4E-2	0.06	2.2	0.59	9E-3	0.26	8E-2	0.19	26.9
9	0.61	8E-3	0.06	1E-2	0.05	1.9	0.58	8E-3	0.26	3E-2	0.21	34.5
10	0.58	2E-2	0.07	2E-2	0.06	5.9	0.57	6E-3	0.15	2E-2	0.11	33.2

Tabla 4.16.*b*): Comparativa de resultados de los algoritmos ECFA y CFA en el caso RBFN

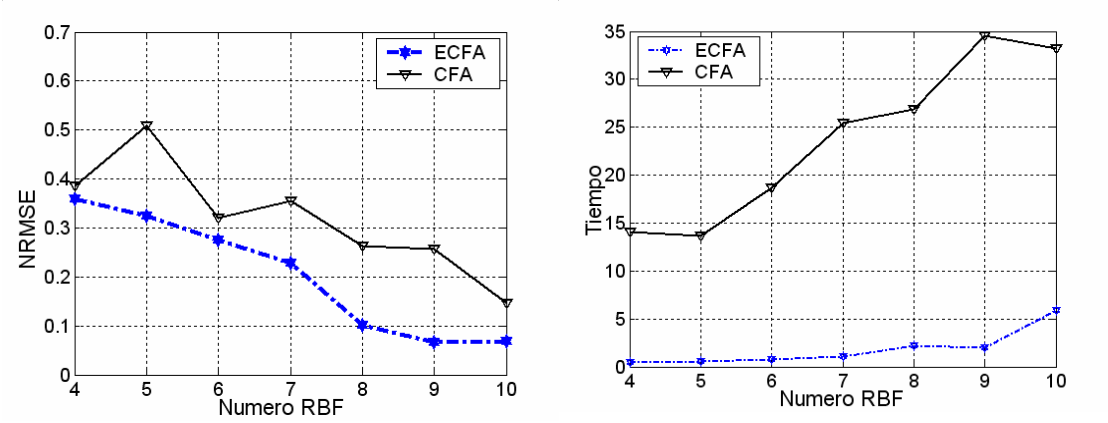


Fig. 4.63 Comparación NRMSE$_T$ y Tiempo de clustering entre ECFA y CFA en el caso RBFN

	ECFA						CFA					
m	**NRMSE $_C$**	**DevS**	**NRMSE$_T$**	**DevS**	**NRMSE$_{min}$**	**Tiempo$_C$**	**NRMSE $_C$**	**DevS**	**NRMSE$_T$**	**DevS**	**NRMSE$_{min}$**	**Tiempo$_C$**
4	0.68	0.0	0.48	0.0	0.46	1.8	0.75	2E-4	0.53	7E-2	0.54	14.1
5	0.62	1E-4	0.42	3E-2	0.25	0.71	0.73	1E-3	0.59	6E-2	0.52	13.7
6	0.59	4E-3	0.35	2E-2	0.12	1.1	0.71	5E-3	0.46	5E-2	0.40	18.7
7	0.62	3E-4	0.06	0.0	0.06	2.1	0.70	2E-3	0.34	4E-2	0.30	25.4
8	0.66	1E-3	0.03	5E-3	0.03	2.5	0.71	2E-3	0.26	3E-2	0.24	26.9
9	0.65	2E-2	0.02	9E-3	0.02	1.9	0.64	7E-3	0.16	4E-2	0.12	34.5
10	0.63	2E-2	0.02	9E-3	0.02	2.5	0.68	7E-2	0.13	2E-2	0.11	33.2

Tabla 4.16.*c*): Comparativa de resultados de los algoritmos ECFA y CFA en el caso NWNN

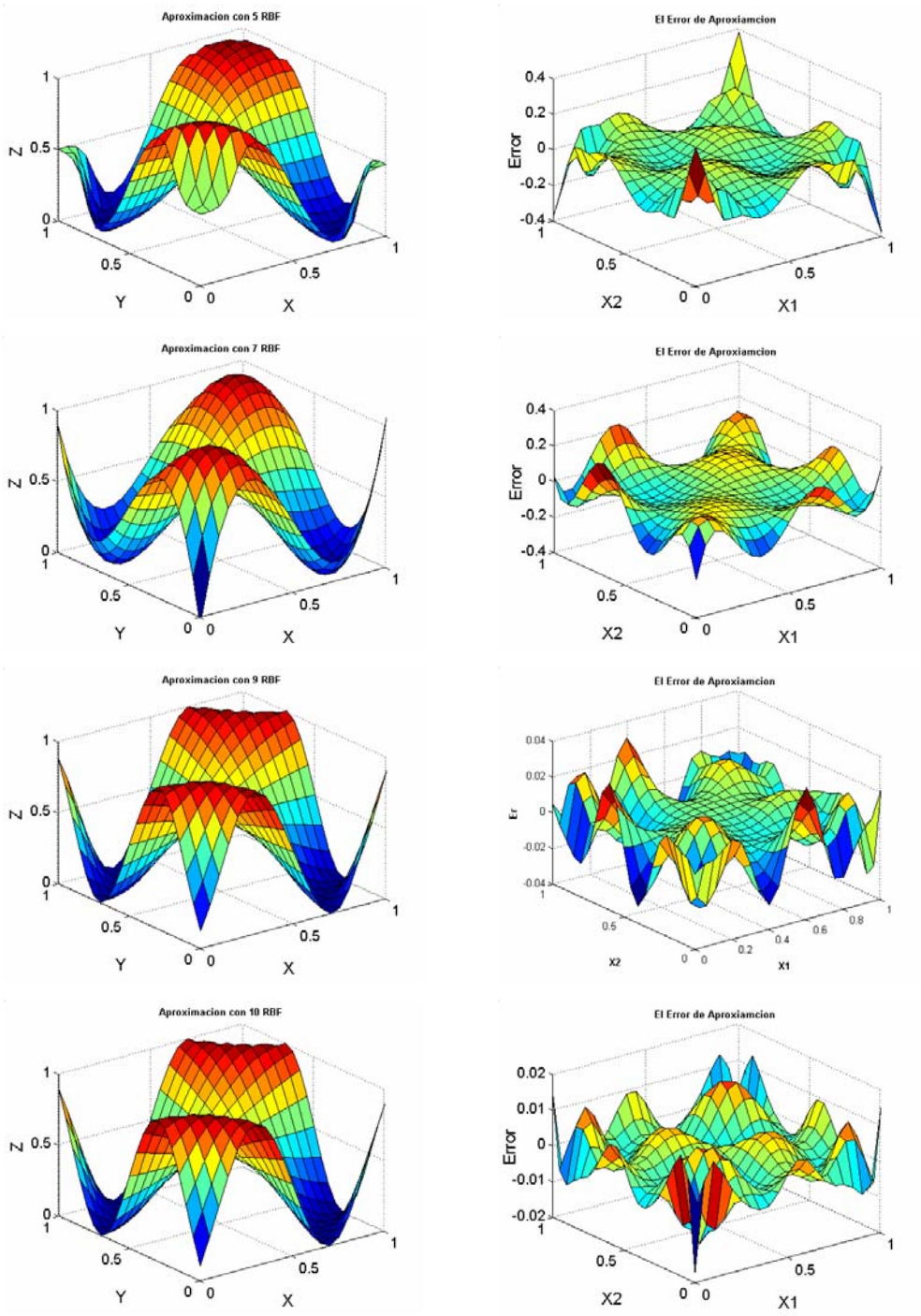


Fig. 4.64 Distintas aproximaciones de la función $f_1(x_1, x_2)$

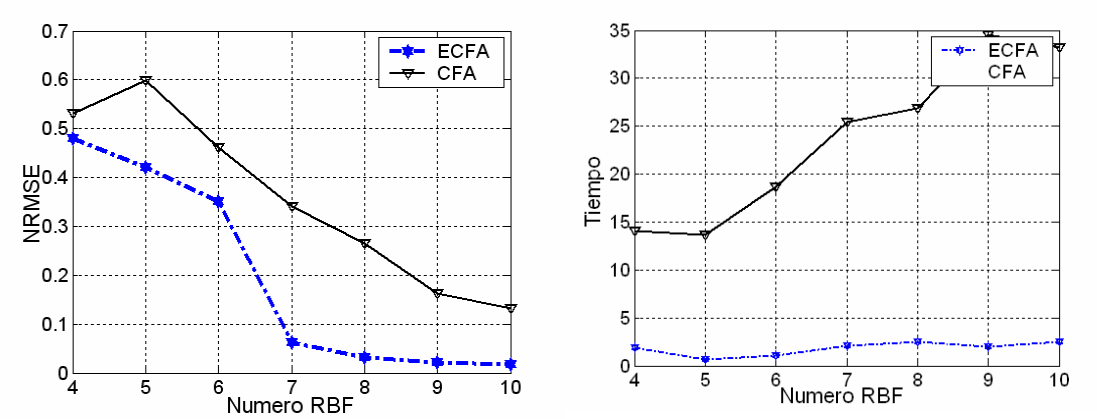


Fig. 4.65 Comparación $NRMSE_T$ y Tiempo de clustering entre ECFA y CFA en el caso NWNN

	ECFA						CFA					
m	**NRMSE $_C$**	**DevS**	**$NRMSE_T$**	**DevS**	**$NRMSE_{min}$**	**$Tiempo_C$**	**NRMSE $_C$**	**DevS**	**$NRMSE_T$**	**DevS**	**$NRMSE_{min}$**	**$Tiempo_C$**
4	0.55	3E-3	0.41	0 .0	0.41	0.45	0.67	1E-3	0.40	0.0	0.40	14.1
5	0.52	4E-4	0.36	0 .0	0.19	0.99	0.62	1E-2	0.33	3E-2	0.30	13.7
6	0.46	3E-2	0.30	1E-2	0.19	5.6	0.59	2E-3	0.27	3E-2	0.23	18.7
7	0.54	8E-3	0.21	1E-2	0.09	15.7	0.58	4E-3	0.26	4E-2	0.21	25.4
8	0.57	3E-3	0.18	9E-2	0.06	1.9	0.58	7E-3	0.26	2E-2	0.24	26.9
9	0.58	4E-3	0.04	5E-3	0.04	1.9	0.58	1E-2	0.16	2E-2	0.14	34.5
10	0.57	2E-2	0.03	6E-3	0.02	2.6	0.58	9E-3	0.15	1E-2	0.14	33.2

Tabla 4.16.*d*): Comparativa de resultados de los algoritmos ECFA y CFA en el caso WNN

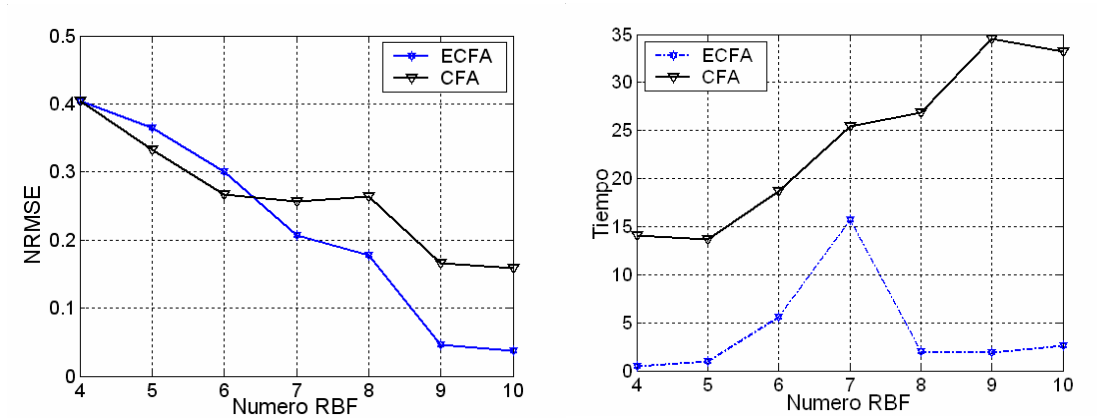


Fig. 4.66 Comparación $NRMSE_T$ y Tiempo de clustering entre ECFA y CFA en el caso WNN

- **Función** $f_5(x_1, x_2)$

  En la figura 4.67 se muestra la función $f_5(x_1, x_2)$, esta función se define por la ecuación:

$$f_5(x_1, x_2) = 42.659(0.1 + x_1(0.05 + x_1^4 - 10x_1^2 x_2^2 + 5x_2^4)), \quad x_1, x_2 \in [-0.5, 0.5] \quad (4.16)$$

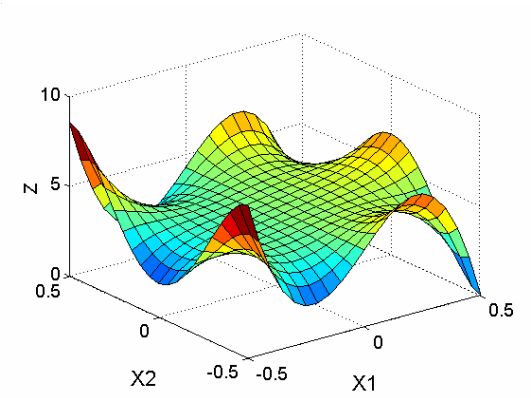


Fig. 4.67 Función objetivo de la función $f_5(x_1, x_2)$

	ECFA						CFA					
m	**NRMSE $_C$**	**DevS**	**NRMSE$_T$**	**DevS**	**NRMSE$_{min}$**	**Tiempo$_C$**	**NRMSE $_C$**	**DevS**	**NRMSE$_T$**	**DevS**	**NRMSE$_{min}$**	**Tiempo$_C$**
4	0.99	0.0	0.97	0.0	0.97	0.57	0.97	0.0	0.43	8E-2	0.40	11.6
5	0.99	9E-4	0.55	2E-2	0.51	0.66	0.99	3E-3	0.45	1E-2	0.40	11.8
6	0.97	3E-2	0.43	3E-2	0.40	0.86	0.99	2E-2	0.34	3E-2	0.31	9.4
7	0.98	5E-3	0.29	4E-2	0.25	1.8	0.94	3E-2	0.32	3E-2	0.30	11.8
8	0.93	1E-2	0.32	2E-2	0.30	2.5	0.91	2E-2	0.24	2E-2	0.21	12.2
9	0.82	4E-2	0.07	2E-2	0.06	3.5	0.92	3E-2	0.19	2E-2	0.14	22.6
10	0.84	2E-2	0.05	2E-2	0.02	5.7	0.88	6E-2	0.15	1E-2	0.13	23.3

Tabla 4.17.*a*): Comparativa de resultados de los algoritmos ECFA y CFA en el caso NRBFN

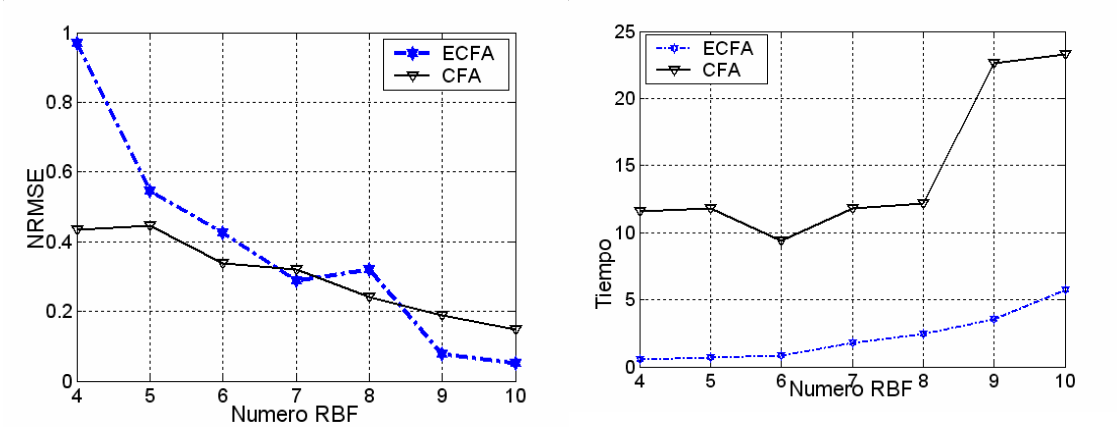


Fig. 4.68 Comparación $NRMSE_T$ y Tiempo de clustering entre ECFA y CFA en el caso NRBFN

	ECFA						CFA					
m	**NRMSE $_C$**	**DevS**	**NRMSE$_T$**	**DevS**	**NRMSE$_{min}$**	**Tiempo$_C$**	**NRMSE $_C$**	**DevS**	**NRMSE$_T$**	**DevS**	**NRMSE$_{min}$**	**Tiempo$_C$**
4	1.07	9E-4	0.72	6E-2	0.70	0.39	1.06	1E-3	0.66	5E-2	0.61	11.6
5	1.06	1E-2	0.59	5E-2	0.53	1.4	1.05	7E-2	0.59	2E-2	0.56	11.8
6	1.02	4E-2	0.44	1E-2	0.32	1.1	1.003	2E-2	0.51	5E-2	0.44	9.4
7	0.95	4E-2	0.19	6E-2	0.12	1.4	0.96	4E-2	0.39	1E-2	0.29	11.8
8	0.93	8E-3	0.13	5E-2	0.07	2.3	0.92	3E-2	0.28	8E-2	0.15	12.2
9	0.91	3E-2	0.10	3E-2	0.07	3.0	0.94	7E-3	0.15	1E-2	0.13	22.6
10	0.87	3E-2	0.09	6E-2	0.03	3.4	0.93	1E-2	0.07	1E-2	0.07	23.3

Tabla 4.17.*b*): Comparativa de resultados de los algoritmos ECFA y CFA en el caso RBFN

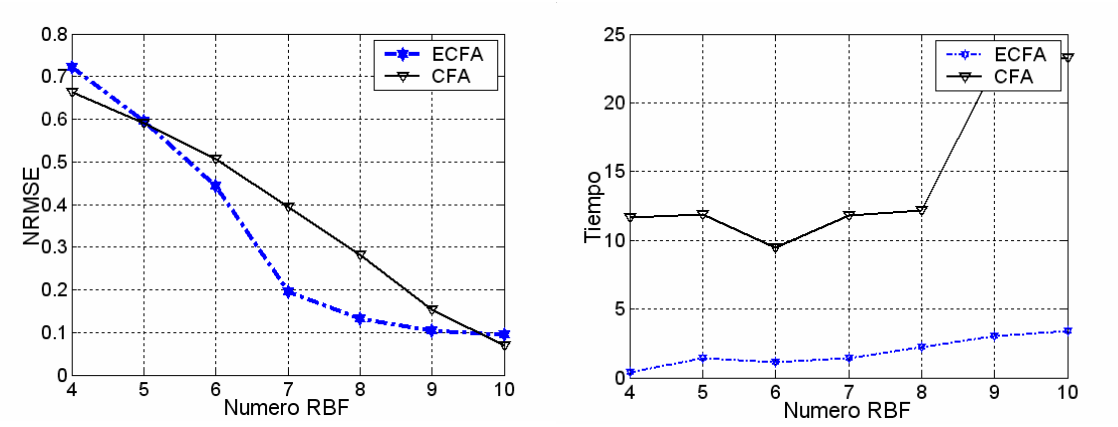


Fig. 4.69 Comparación $NRMSE_T$ y Tiempo de clustering entre ECFA y CFA en el caso RBFN

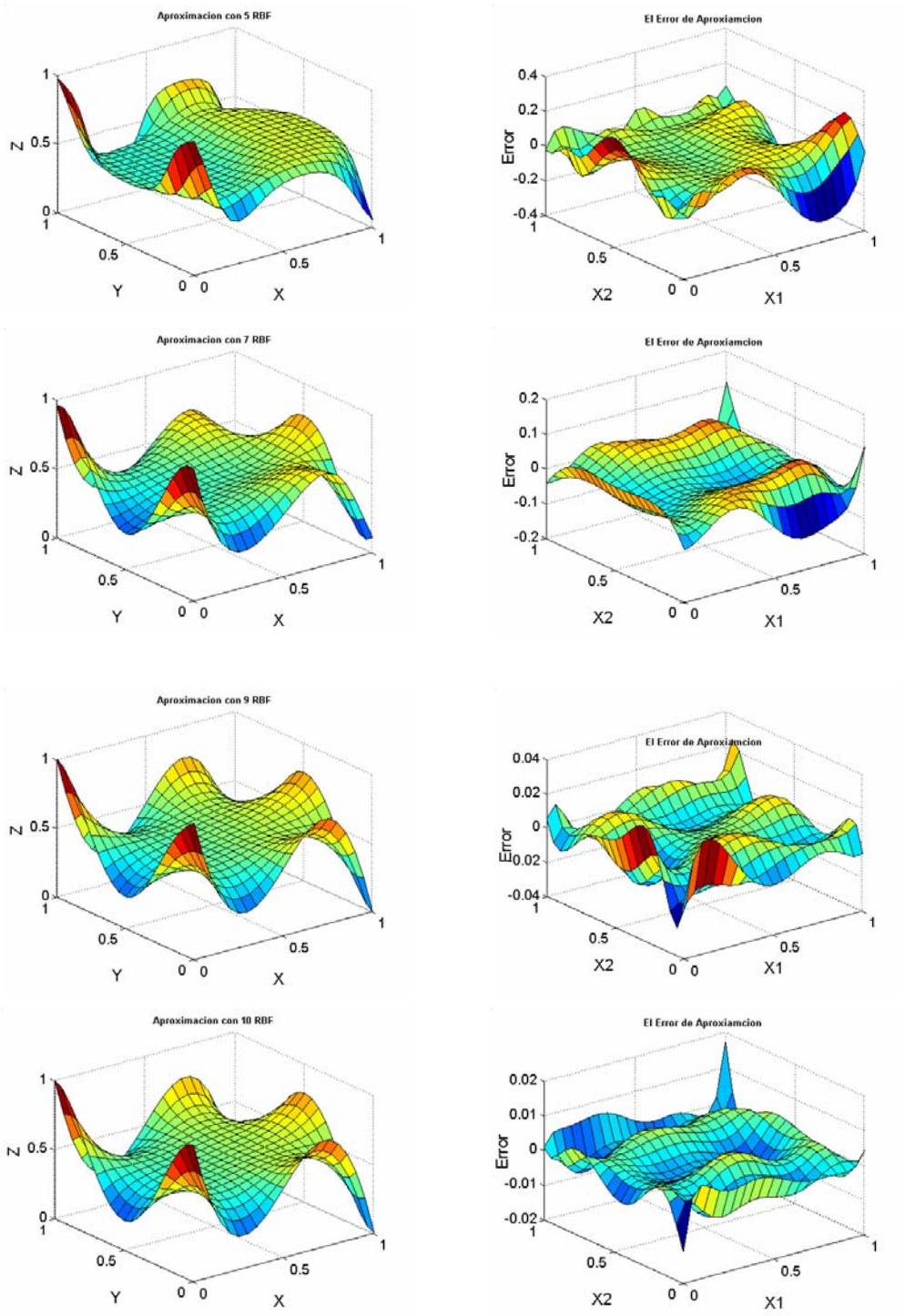


Fig. 4.70 Distintas aproximaciones de la función $f_5(x_1, x_2)$

	ECFA						CFA					
m	**NRMSE $_C$**	**DevS**	**NRMSE$_T$**	**DevS**	**NRMSE$_{min}$**	**Tiempo$_C$**	**NRMSE $_C$**	**DevS**	**NRMSE$_T$**	**DevS**	**NRMSE$_{min}$**	**Tiempo$_C$**
4	0.86	0.0	0.51	1E-2	0.48	0.67	0.88	3E-2	0.71	5E-2	0.62	12.9
5	0.87	6E-3	0.41	1E-2	0.40	0.50	0.89	1E-1	0.58	5E-2	0.50	12.2
6	0.84	4E-2	0.19	6E-2	0.16	0.69	0.83	2E-2	0.29	5E-2	0.25	12.2
7	0.91	3E-2	0.10	1E-2	0.08	1.4	0.78	2E-2	0.28	3E-2	0.25	26.2
8	0.84	8E-4	0.05	3E-3	0.04	2.9	0.76	1E-2	0.25	3E-2	0.23	32.1
9	0.82	3E-2	0.05	1E-2	0.03	2.9	0.79	5E-2	0.15	1E-2	0.14	38.0
10	0.73	1E-2	0.02	1E-2	0.02	3.4	0.77	2E-2	0.14	1E-2	0.13	28.8

Tabla 4.17.*c*): Comparativa de resultados de los algoritmos ECFA y CFA en el caso NWNN

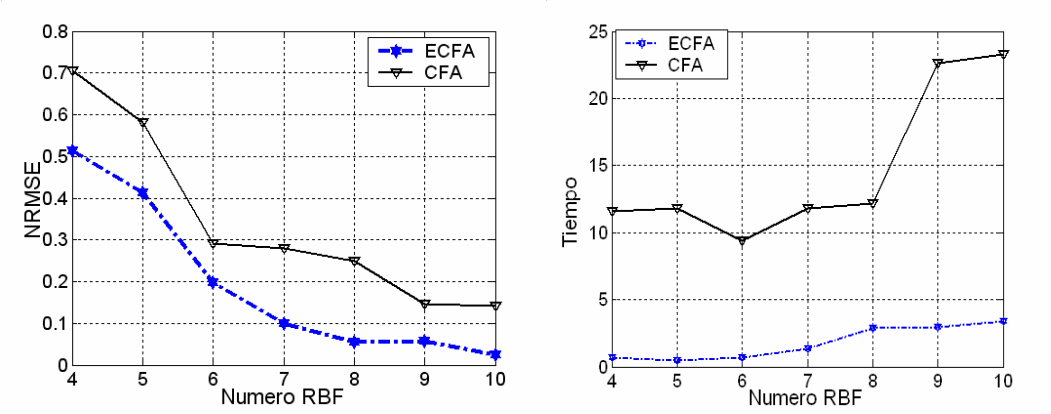


Fig. 4.71 Comparación NRMSE$_T$ y Tiempo de clustering entre ECFA y CFA en el caso NWNN

	ECFA						CFA					
m	**NRMSE $_C$**	**DevS**	**NRMSE$_T$**	**DevS**	**NRMSE$_{min}$**	**Tiempo$_C$**	**NRMSE $_C$**	**DevS**	**NRMSE$_T$**	**DevS**	**NRMSE$_{min}$**	**Tiempo$_C$**
4	0.82	2E-2	0.55	1E-2	0.51	0.59	0.77	0.0	0.47	2E-2	0.29	12.5
5	0.82	3E-2	0.16	5E-2	0.14	0.61	0.78	8E-4	0.59	6E-2	0.58	12.6
6	0.83	4E-3	0.16	3E-2	0.06	0.93	0.78	6E-3	0.26	7E-3	0.25	16.0
7	0.86	3E-2	0.06	2E-2	0.06	1.7	0.79	2E-3	0.26	2E-2	0.24	22.8
8	0.86	2E-2	0.05	3E-2	0.04	2.1	0.79	1E-3	0.26	2E-2	0.24	22.3
9	0.81	2E-2	0.03	7E-3	0.03	2.6	0.78	7E-3	0.16	8E-3	0.15	22.6
10	0.78	1E-2	0.02	7E-3	0.02	5.5	0.77	1E-3	0.14	1E-2	0.13	43.9

Tabla 4.17.*d*): Comparativa de resultados de los algoritmos ECFA y CFA en el caso WNN

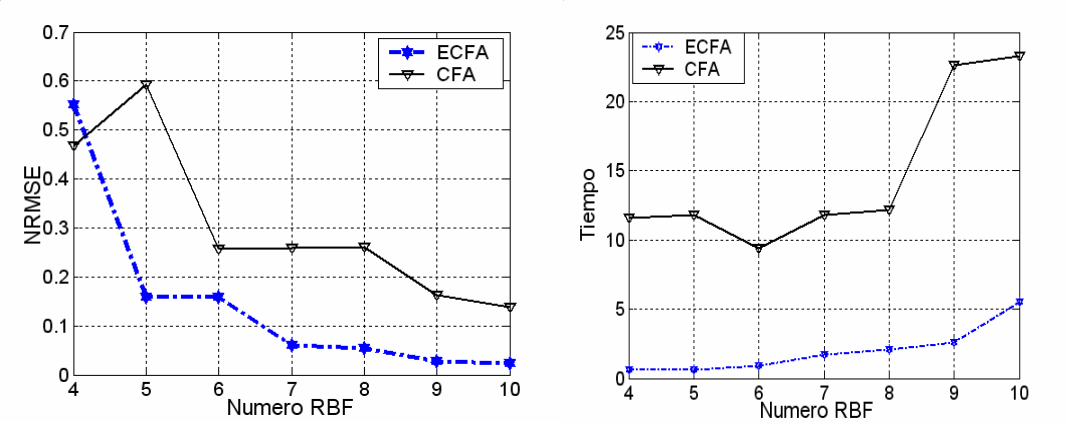


Fig. 4.72 Comparación $NRMSE_T$ y Tiempo de clustering entre ECFA y CFA en el caso WNN

- **Función** $f_6(x_1, x_2)$

En Fig. 4.73 se muestra la función $f_6(x_1, x_2)$, esta función se define por la ecuación:

$$f_6(x_1, x_2) = 1.3356\left[\begin{array}{l} 1.5(1 - x_1) + e^{2x_1 - 1}\,\mathrm{sen}(3\pi(x_1 - 0.6)^2) + \\ e^{3(x_2 - 0.5)}\,\mathrm{sen}(4\pi(x_2 - 0.9)^2) \end{array}\right] \qquad (4.17)$$

$$x_1, x_2 \in [0, 1]$$

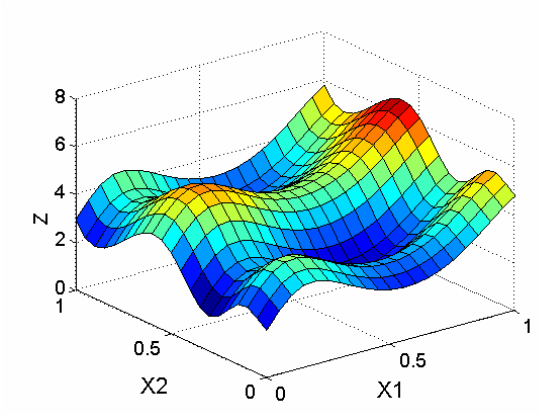


Fig. 4.73 Función objetivo de la función $f_6(x_1, x_2)$

	ECFA						CFA					
m	**NRMSE $_C$**	**DevS**	**NRMSE$_T$**	**DevS**	**NRMSE$_{min}$**	**Tiempo$_C$**	**NRMSE $_C$**	**DevS**	**NRMSE$_T$**	**DevS**	**NRMSE$_{min}$**	**Tiempo$_C$**
4	0.89	6E-3	0.54	5E-2	0.48	0.58	0.91	7E-3	0.56	6E-3	0.55	11.9
5	0.82	3E-2	0.40	6E-2	0.35	1.24	0.90	3E-3	0.37	8E-2	0.32	12.1
6	0.79	4E-2	0.31	3E-2	0.27	1.1	0.81	9E-3	0.37	2E-2	0.34	11.3
7	0.71	3E-2	0.22	7E-2	0.14	1.2	0.80	1E-2	0.27	1E-2	0.25	12.5
8	0.70	2E-2	0.14	4E-2	0.07	2.1	0.76	4E-2	0.20	4E-2	0.16	17.1
9	0.66	6E-2	0.07	2E-2	0.05	2.2	0.70	7E-2	0.16	2E-2	0.15	21.8
10	0.62	6E-2	0.04	1E-2	0.03	3.6	0.68	6E-2	0.14	1E-2	0.14	23.3

Tabla 4.18.*a*): Comparativa de resultados de los algoritmos ECFA y CFA en el caso NRBFN

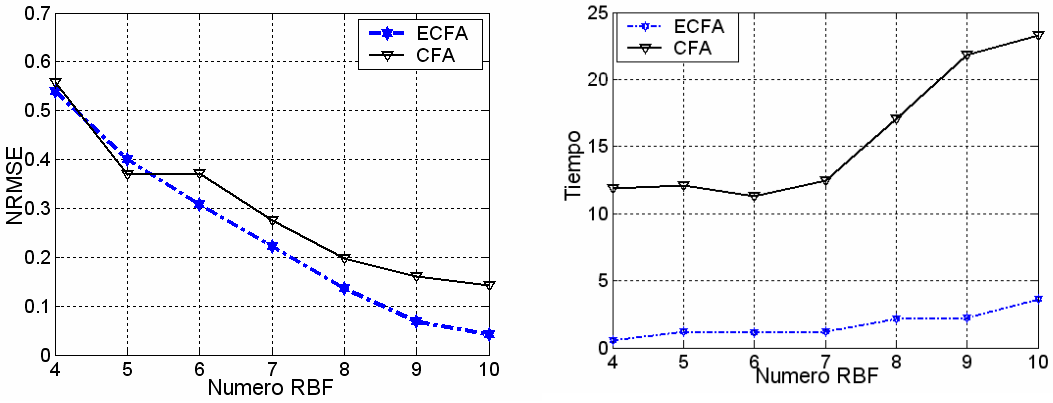


Fig. 4.74 Comparación NRMSE$_T$ y Tiempo de clustering entre ECFA y CFA en el caso NRBFN

	ECFA						CFA					
m	**NRMSE $_C$**	**DevS**	**NRMSE$_T$**	**DevS**	**NRMSE$_{min}$**	**Tiempo$_C$**	**NRMSE $_C$**	**DevS**	**NRMSE$_T$**	**DevS**	**NRMSE$_{min}$**	**Tiempo$_C$**
4	1.06	1E-3	0.60	6E-2	0.41	0.56	1.02	1E-2	0.54	2E-2	0.42	11.9
5	1.04	5E-2	0.40	2E-2	0.37	0.51	0.88	1E-2	0.43	9E-2	0.37	12.1
6	0.83	1E-2	0.34	2E-2	0.28	0.77	0.88	1E-2	0.36	4E-2	0.29	11.3
7	0.78	4E-2	0.26	4E-2	0.20	1.2	0.76	2E-2	0.28	4E-2	0.22	12.5
8	0.77	4E-2	0.18	2E-2	0.12	1.5	0.72	9E-2	0.24	1E-2	0.24	17.1
9	0.77	2E-2	0.16	2E-2	0.13	1.9	0.71	8E-2	0.26	2E-2	0.24	21.8
10	0.71	2E-2	0.09	8E-2	0.07	3.1	0.71	6E-2	0.17	1E-2	0.15	23.3

Tabla 4.18.*b*): Comparativa de resultados de los algoritmos ECFA y CFA en el caso RBFN

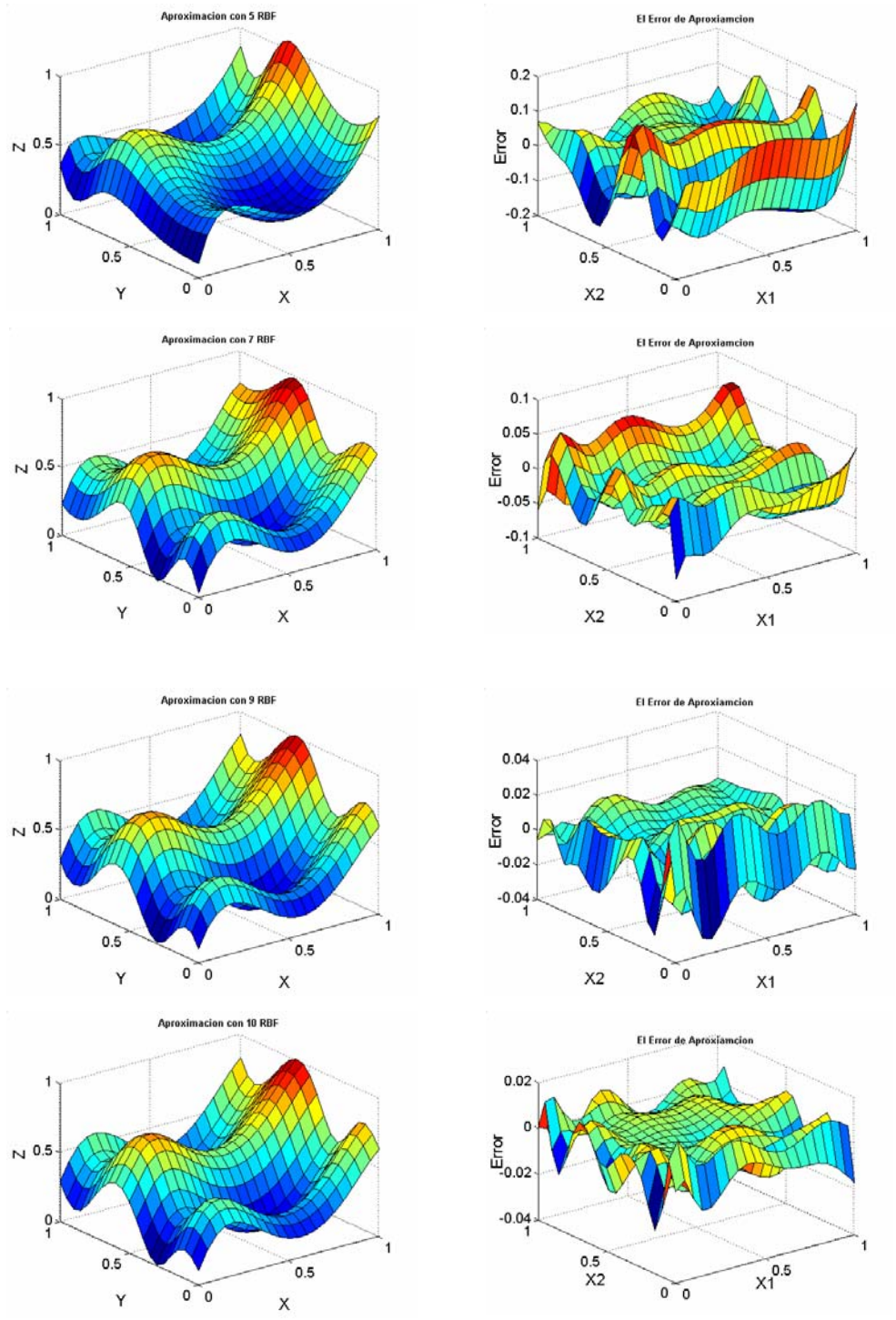


Fig. 4.75 Distintas aproximaciones de la función $f_6(x_1, x_2)$

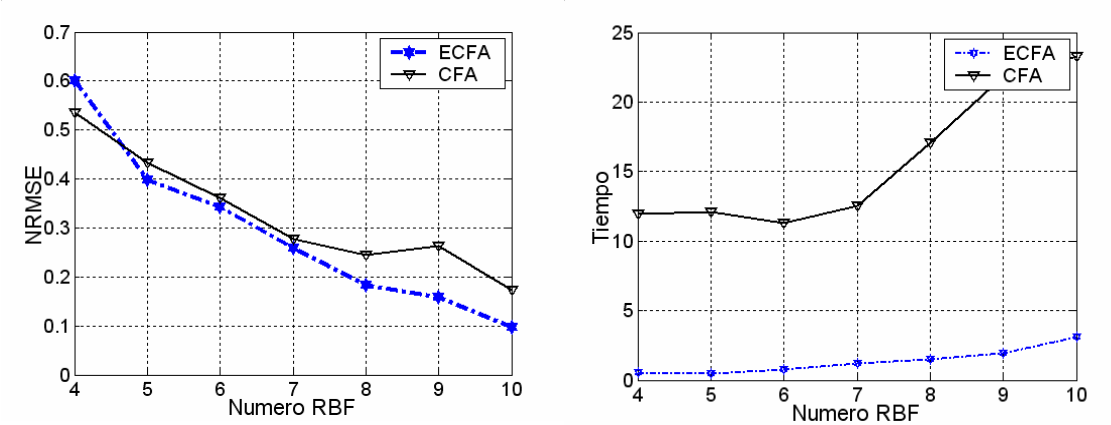


Fig. 4.76 Comparación $NRMSE_T$ y Tiempo de clustering entre ECFA y CFA en el caso RBFN

	ECFA						CFA					
m	**NRMSE $_C$**	**DevS**	**NRMSE$_T$**	**DevS**	**NRMSE$_{min}$**	**Tiempo$_C$**	**NRMSE $_C$**	**DevS**	**NRMSE$_T$**	**DevS**	**NRMSE$_{min}$**	**Tiempo$_C$**
4	0.75	3E-2	0.28	0.0	0.28	0.58	0.87	2E-2	0.35	4E-2	0.34	11.9
5	0.73	1E-2	0.22	6E-3	0.21	1.1	0.65	1E-2	0.34	2E-2	0.31	12.1
6	0.66	4E-2	0.15	8E-2	0.10	1.1	0.63	5E-2	0.30	6E-2	0.27	11.3
7	0.51	5E-2	0.07	2E-2	0.06	1.4	0.59	4E-2	0.25	4E-2	0.24	12.5
8	0.54	6E-2	0.06	2E-2	0.04	2.1	0.55	2E-2	0.20	2E-2	0.18	17.1
9	0.54	7E-2	0.05	9E-3	0.04	2.6	0.55	3E-2	0.15	2E-2	0.13	21.8
10	0.53	1E-2	0.04	5E-3	0.04	4.1	0.54	5E-2	0.14	9E-3	0.14	23.3

Tabla 4.18.*c*): Comparativa de resultados de los algoritmos ECFA y CFA en el caso NWNN

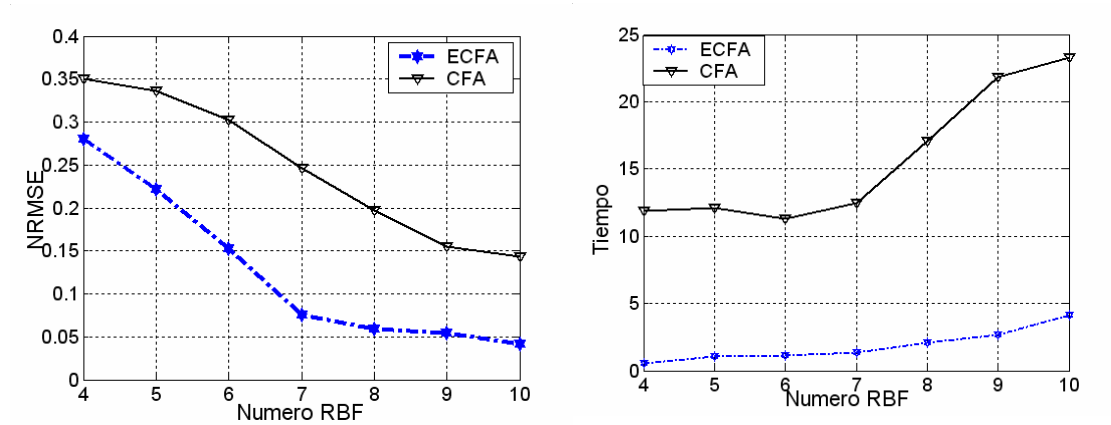


Fig. 4.77 Comparación $NRMSE_T$ y Tiempo de clustering entre ECFA y CFA en el caso NWNN

	ECFA						CFA					
m	**NRMSE $_C$**	**DevS**	**NRMSE$_T$**	**DevS**	**NRMSE$_{min}$**	**Tiempo$_C$**	**NRMSE $_C$**	**DevS**	**NRMSE$_T$**	**DevS**	**NRMSE$_{min}$**	**Tiempo$_C$**
4	0.82	9E-3	0.34	2E-2	0.31	0.46	0.80	7E-3	0.44	6E-2	0.40	11.9
5	0.75	6E-2	0.26	3E-2	0.25	0.50	0.66	1E-2	0.40	6E-2	0.38	12.1
6	0.69	9E-2	0.22	3E-2	0.18	1.1	0.62	9E-2	0.34	8E-3	0.31	11.3
7	0.62	6E-2	0.18	7E-3	0.15	1.3	0.54	1E-2	0.23	2E-2	0.20	12.5
8	0.56	5E-2	0.16	1E-2	0.10	1.7	0.53	2E-2	0.18	2E-2	0.14	17.1
9	0.54	3E-2	0.11	2E-2	0.09	2.3	0.51	1E-2	0.18	1E-2	0.15	21.8
10	0.52	3E-2	0.09	2E-2	0.08	2.8	0.49	1E-2	0.12	1E-2	0.10	23.3

Tabla 4.18.*d*): Comparativa de resultados de los algoritmos ECFA y CFA en el caso WNN

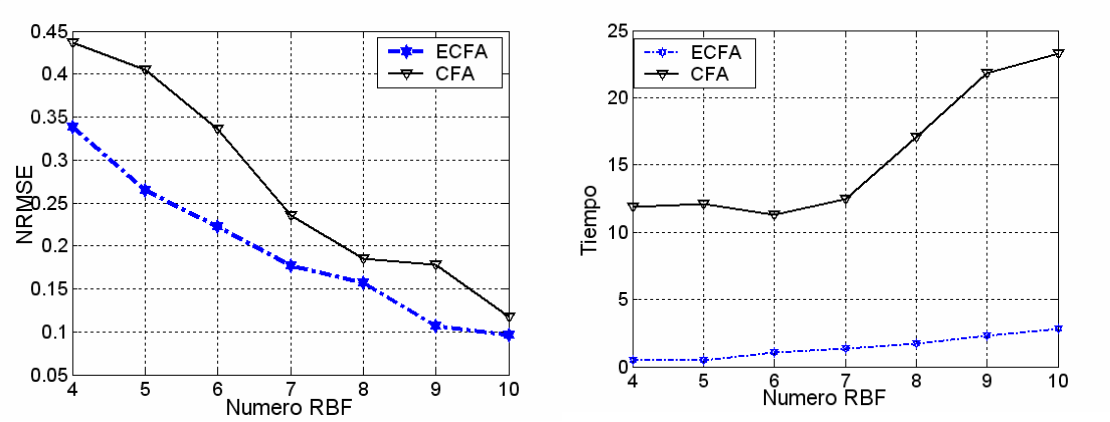


Fig. 4.78 Comparación NRMSE$_T$ y Tiempo de clustering entre ECFA y CFA en el caso WNN

- **Función** $f_7(x_1, x_2)$

  La Fig. 4.79 muestra la función $f_7(x_1, x_2)$, esta función se define por la ecuación:

$$f_7(x_1, x_2) = 1.9\left(1.35 + e^{x_1} sen(13(x_1 - 0.6)^2)\, e^{-x_2}\, sen(7x_2)\right) \quad x_1, x_2 \in [0, 1] \quad (4.18)$$

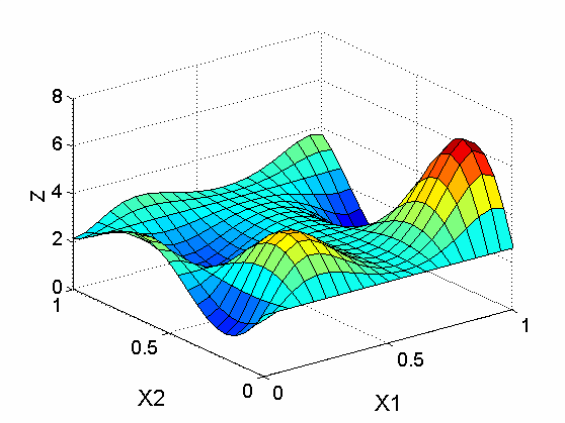


Fig. 4.79 Función objetivo de la función $f_7(x_1, x_2)$

	ECFA						CFA					
m	**NRMSE $_C$**	**DevS**	**NRMSE$_T$**	**DevS**	**NRMSE$_{min}$**	**Tiempo$_C$**	**NRMSE $_C$**	**DevS**	**NRMSE$_T$**	**DevS**	**NRMSE$_{min}$**	**Tiempo$_C$**
4	0.88	1E-4	0.53	5E-2	0.50	0.45	0.88	5E-3	0.43	5E-2	0.37	11.1
5	0.88	2E-3	0.44	6E-2	0.38	0.66	0.88	9E-3	0.48	6E-2	0.37	12.7
6	0.87	2E-2	0.40	8E-3	0.30	0.88	0.87	6E-3	0.44	8E-2	0.34	15.6
7	0.81	5E-2	0.29	8E-3	0.20	0.98	0.87	2E-2	0.34	9E-2	0.27	12.6
8	0.81	5E-2	0.16	2E-2	0.13	2.3	0.83	1E-2	0.27	3E-2	0.22	13.6
9	0.76	3E-2	0.14	3E-3	0.10	2.3	0.82	2E-2	0.19	1E-2	0.17	13.7
10	0.62	8E-3	0.10	2E-3	0.07	3.2	0.83	2E-2	0.17	1E-2	0.15	18.3

Tabla 4.19.*a*): Comparativa de resultados de los algoritmos ECFA y CFA en el caso NRBFN

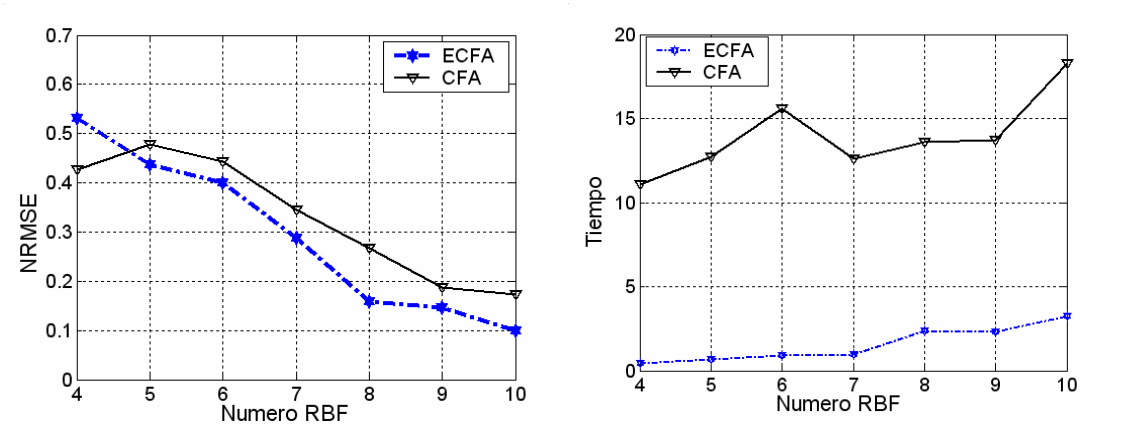


Fig. 4.80 Comparación NRMSE$_T$ y Tiempo de clustering entre ECFA y CFA en el caso NRBFN

	ECFA						CFA					
m	**NRMSE $_C$**	**DevS**	**NRMSE$_T$**	**DevS**	**NRMSE$_{min}$**	**Tiempo$_C$**	**NRMSE $_C$**	**DevS**	**NRMSE$_T$**	**DevS**	**NRMSE$_{min}$**	**Tiempo$_C$**
4	1.01	5E-4	0.47	2E-2	0.43	0.36	1.03	5E-3	0.47	2E-2	0.45	11.1
5	1.01	4E-3	0.48	7E-2	0.35	0.53	1.03	4E-2	0.37	8E-2	0.28	12.7
6	1.00	3E-2	0.27	7E-2	0.17	0.78	0.92	8E-2	0.34	6E-2	0.27	15.6
7	0.78	2E-2	0.19	2E-2	0.16	1.2	0.87	5E-2	0.29	4E-2	0.26	12.6
8	0.81	4E-2	0.16	2E-2	0.14	1.4	0.84	3E-2	0.28	4E-2	0.22	13.6
9	0.76	4E-2	0.14	3E-2	0.10	2.3	0.84	5E-2	0.25	5E-2	0.18	13.7
10	0.68	7E-3	0.10	1E-2	0.08	2.2	0.80	2E-2	0.19	1E-2	0.18	18.3

Tabla 4.19.*b*): Comparativa de resultados de los algoritmos ECFA y CFA en el caso RBFN

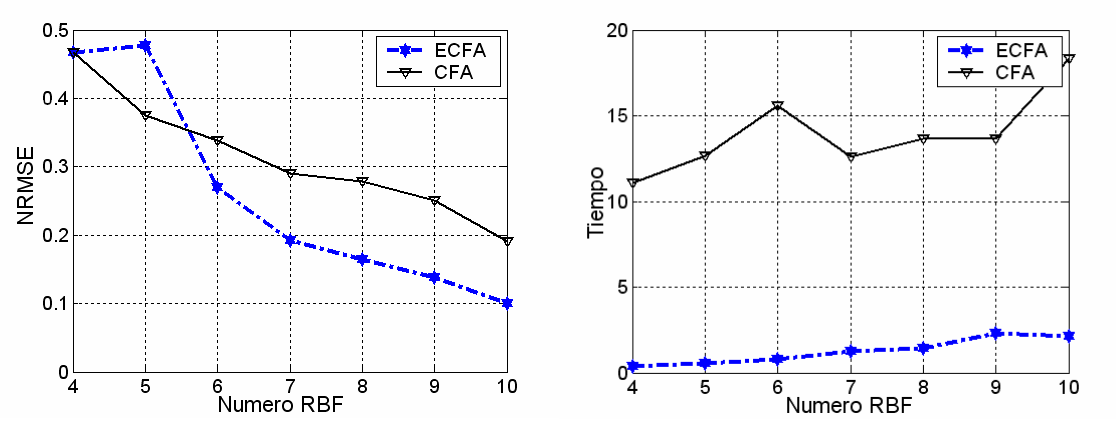


Fig. 4.81 Comparación NRMSE$_T$ y Tiempo de clustering entre ECFA y CFA en el caso RBFN

	ECFA						CFA					
m	**NRMSE $_C$**	**DevS**	**NRMSE$_T$**	**DevS**	**NRMSE$_{min}$**	**Tiempo$_C$**	**NRMSE $_C$**	**DevS**	**NRMSE$_T$**	**DevS**	**NRMSE$_{min}$**	**Tiempo$_C$**
4	0.75	6E-4	0.48	2E-2	0.46	0.61	0.77	2E-2	0.47	2E-2	0.46	11.1
5	0.79	2E-2	0.39	0.0	0.37	0.65	0.84	3E-2	0.44	4E-2	0.43	12.7
6	0.73	6E-2	0.26	2E-2	0.24	0.77	0.83	8E-3	0.37	2E-2	0.33	15.6
7	0.73	4E-2	0.17	8E-2	0.16	1.1	0.78	2E-2	0.36	2E-2	0.34	12.6
8	0.73	5E-2	0.14	3E-2	0.11	1.8	0.75	3E-2	0.24	2E-2	0.23	13.6
9	0.55	4E-2	0.11	9E-2	0.10	2.5	0.70	8E-2	0.25	4E-2	0.20	13.7
10	0.56	4E-2	0.08	1E-2	0.07	2.6	0.68	1E-2	0.18	2E-2	0.15	18.3

Tabla 4.19.*c*): Comparativa de resultados de los algoritmos ECFA y CFA en el caso NWNN

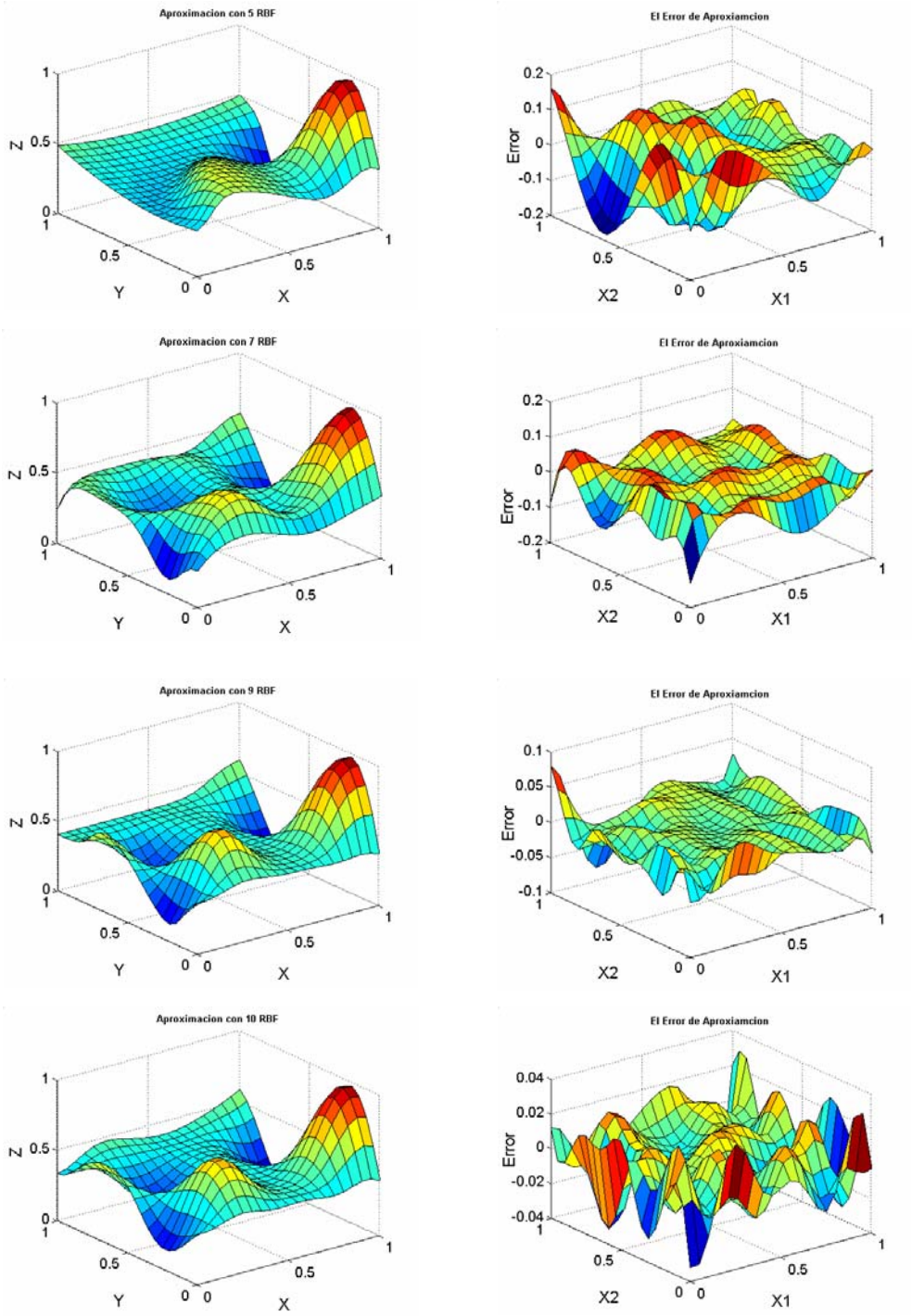


Fig. 4.82 Distintas aproximaciones de la función $f_7(x_1, x_2)$

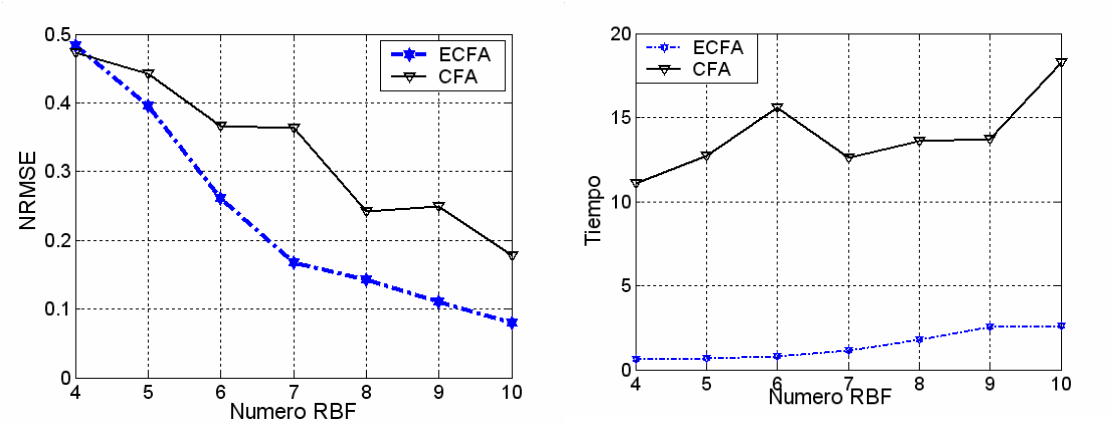


Fig. 4.83 Comparación $NRMSE_T$ y Tiempo de clustering entre ECFA y CFA en el caso NWNN

	ECFA						CFA					
m	**$NRMSE_C$**	**DevS**	**$NRMSE_T$**	**DevS**	**$NRMSE_{min}$**	**$Tiempo_C$**	**$NRMSE_C$**	**DevS**	**$NRMSE_T$**	**DevS**	**$NRMSE_{min}$**	**$Tiempo_C$**
4	0.79	9E-3	0.27	0.0	0.27	0.35	0.81	2E-3	0.44	2E-2	0.42	11.1
5	0.76	2E-3	0.33	2E-3	0.31	0.69	0.80	9E-3	0.46	2E-2	0.45	12.7
6	0.72	3E-2	0.21	5E-2	0.15	1.1	0.78	9E-3	0.35	4E-2	0.30	15.6
7	0.71	2E-2	0.16	5E-2	0.12	1.8	0.76	2E-2	0.33	4E-2	0.29	12.6
8	0.68	1E-2	0.13	2E-2	0.11	2.1	0.72	1E-2	0.26	3E-2	0.23	13.6
9	0.67	2E-2	0.09	4E-2	0.07	2.5	0.69	9E-3	0.12	1E-2	0.10	13.7
10	0.61	3E-2	0.09	1E-2	0.09	3.9	0.63	4E-2	0.16	3E-2	0.11	18.3

Tabla 4.19.*d*): Comparativa de resultados de los algoritmos ECFA y CFA en el caso WNN

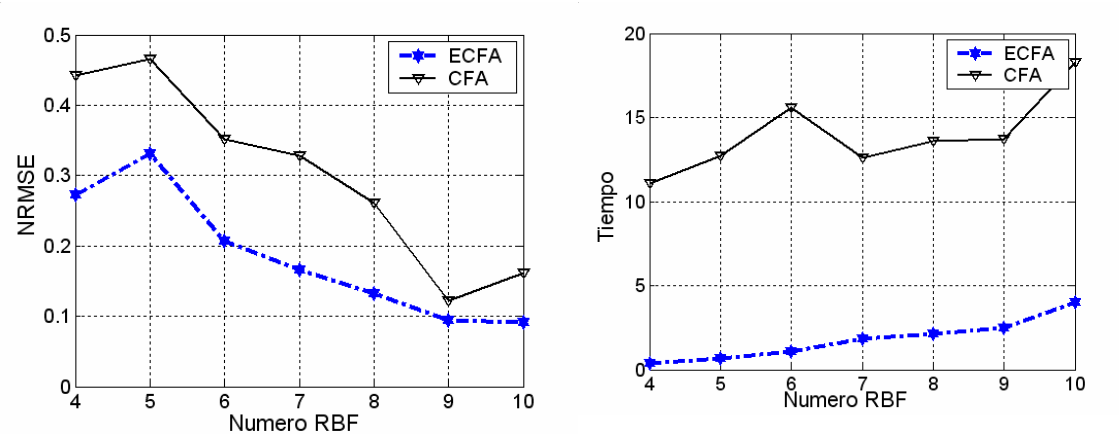


Fig. 4.84 Comparación $NRMSE_T$ y Tiempo de clustering entre ECFA y CFA en el caso WNN

4.5 Análisis de la robustez del algoritmo ECFA ante la introducción de ruido

En experimentos reales, lo normal es que los conjuntos de entrenamiento tengan ruido. Este ruido se produce por las pequeñas perturbaciones que introducen los sensores, en los datos de entrenamiento. Para comprobar cómo funciona el algoritmo de *clustering* propuesto ECFA ante el efecto del ruido se han escogido las funciones *Var(x)* y $g(x_1,x_2)$, utilizadas en los apartados anteriores y se les ha añadido, a los conjuntos de entrenamiento un error aditivo blanco del 5% y del 10% respectivamente. En las figuras 4.85 y 4.86 se representan los conjuntos de entrenamiento con distintas aproximaciones de las dos funciones *Var(x)* y $g(x_1,x_2)$.

- **Función *Var(x)***

  Para está función se utiliza un conjunto de 1000 datos equidistribuidos exentos de ruido para verificar cada una de las configuraciones obtenidos. Se puede comprobar que cuando aumenta el número de funciones base, el error entrenamiento se disminuye. Sin embargo, no ocurre el mismo con el error de test que va decreciendo para redes con número de funciones base inferior a 10 RBF en el caso de una red NRBFN e inferior a 7 en el caso de una red RBFN. La Fig. 4.85 muestra distintas aproximaciones de la función *var(x)* con ruido de 5%.

	NRMSE Training					NRMSE Test				
m	**$NRMSE_C$**	**DevS**	**$NRMSE_T$**	**DevS**	**$NRMSE_{min}$**	**$NRMSE_C$**	**DevS**	**$NRMSE_{Test}$**	**DevS**	**$NRMSE_{min}$**
4	0.96	1E-4	0.55	1E-2	0.47	3.44	1E-3	0.516	1E-3	0.514
5	0.91	6E-3	0.42	5E-2	0.37	2.74	2E-2	0.392	7E-3	0.383
6	0.90	1E-4	0.64	9E-2	0.47	2.04	2E-3	0.346	3E-2	0.316
7	0.88	2 E-2	0.33	8E-2	0.22	1.97	1E-2	0.272	1E-2	0.238
8	0.84	3E-3	0.41	8E-2	0.26	1.54	2E-2	0.210	3E-3	0.208
9	0.79	2E-2	0.25	1E-2	0.13	1.32	5E-2	0.076	7E-2	0.029
10	0.76	1E-2	0.13	3E-3	0.12	1.18	3E-3	0.029	5E-3	0.025
11	0.72	1E-2	0.13	1E-3	0.12	1.06	4E-2	0.032	9E-3	0.021
12	0.69	7E-3	0.12	7E-4	0.12	0.99	2E-2	0.033	4E-3	0.019

Tabla 4.20.*a*): Errores obtenidos Por ECFA para la función *var (x)* con ruido adicional del 5% en el caso NRBFN

	NRMSE Training					NRMSE Test				
m	**NRMSE $_C$**	**DevS**	**NRMSE$_T$**	**DevS**	**NRMSE$_{min}$**	**NRMSE $_C$**	**DevS**	**NRMSE$_{Test}$**	**DevS**	**NRMSE$_{min}$**
4	1.0	2E-3	0.50	4E-2	0.43	3.21	1E-3	0.558	1E-3	0.557
5	0.99	1E-4	0.41	4E-2	0.33	3.06	3E-3	0.425	6E-2	0.351
6	0.98	6E-3	0.17	2E-2	0.16	2.56	2E-2	0.147	3E-2	0.115
7	0.96	3E-2	0.14	1E-2	0.12	2.52	9E-3	0.056	4E-2	0.029
8	0.91	2E-2	0.13	9E-3	0.14	2.01	2E-2	0.057	2E-2	0.039
9	0.81	2E-2	0.14	6E-3	0.13	1.37	8E-2	0.057	2E-2	0.035
10	0.74	3E-2	0.13	2E-3	0.12	1.15	1E-2	0.061	2E-3	0.038
11	0.64	2E-2	0.13	4E-3	0.11	0.923	5E-2	0.062	8E-3	0.022
12	0.60	2E-2	0.12	3E-3	0.11	0.820	3E-2	0.042	4E-3	0.031

Tabla 4.20.*b*): Errores obtenidos Por ECFA para la función *var (x)* con ruido adicional del 5% en el caso RBFN

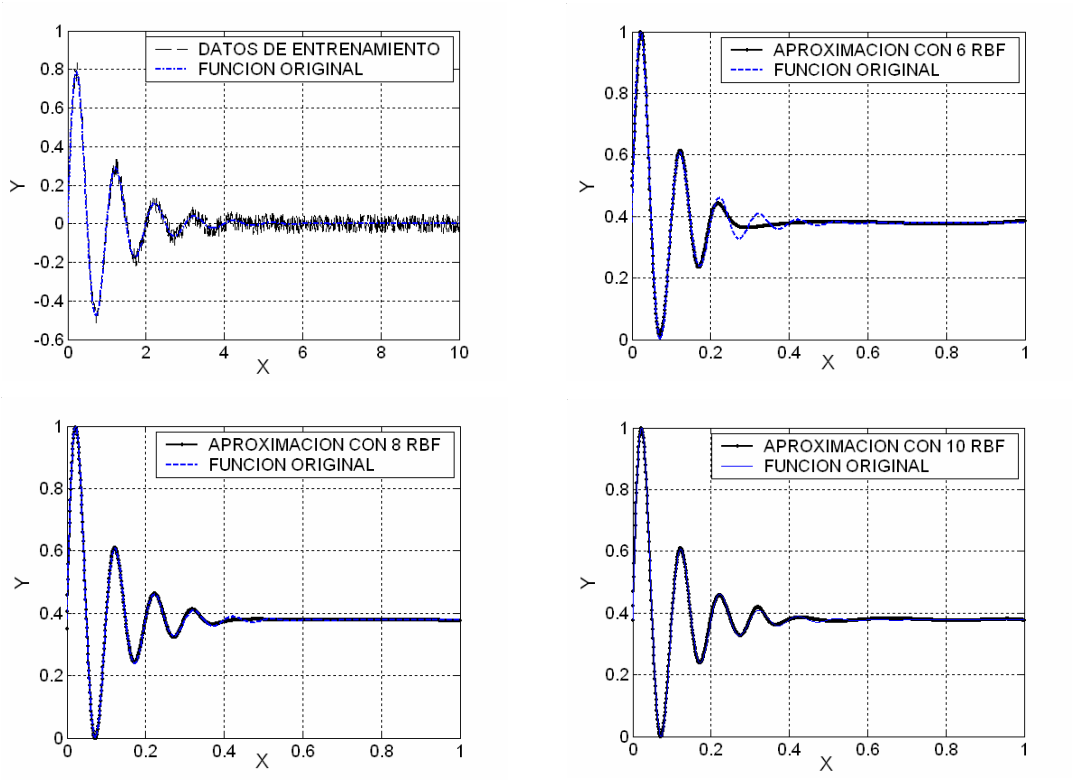


Fig. 4.85 Distintas aproximaciones para la función *var(x)* con un conjunto de entrenamiento al que se le ha añadido 5% de ruido.

- **Función $g(x_1,x_2)$**

  Para está función se utiliza un conjunto de 10000 datos equidistribuidos exentos de ruido para verificar cada una de las configuraciones obtenidos. Se puede comprobar que cuando aumenta el número de funciones base, el error entrenamiento se disminuye. Sin embargo, no ocurre el mismo con el error de test que va decreciendo para redes con número de funciones base inferior a 10 RBF en el caso de una red NRBFN e inferior a 10 en el caso de una red RBFN. La Fig. 4.86 muestra distintas aproximaciones de la función $g(x_1,x_2)$ con ruido de 10%.

	NRMSE Training					NRMSE Test				
m	**$NRMSE_C$**	**DevS**	**$NRMSE_T$**	**DevS**	**$NRMSE_{min}$**	**$NRMSE_C$**	**DevS**	**$NRMSE_{Test}$**	**DevS**	**$NRMSE_{min}$**
4	0.94	1E-2	0.51	4E-2	0.44	2.78	4E-2	0.501	5E-3	0.557
5	0.89	4E-2	0.43	3E-2	0.41	1.92	2E-2	0.448	1E-2	0.351
6	0.86	3E-2	0.36	4E-2	0.30	1.78	9E-2	0.353	5E-2	0.115
7	0.79	3E-2	0.33	5E-2	0.23	1.41	7E-2	0.289	8E-2	0.029
8	0.79	7E-3	0.26	5E-2	0.19	1.47	1E-2	0.200	2E-2	0.039
9	0.74	4E-2	0.20	6E-2	0.11	1.17	1E-2	0.123	5E-2	0.035
10	0.72	4E-2	0.17	3E-2	0.14	1.09	7E-2	0.115	3E-2	0.038
11	0.70	5E-2	0.15	4E-2	0.12	1.06	5E-2	0.128	6E-2	0.022
12	0.66	3E-2	0.14	3E-2	0.11	0.890	8E-2	0.072	2E-2	0.031

Tabla 4.21.*a*): Errores obtenidos por ECFA para la función $g(x_1,x_2)$ con ruido adicional del 10% en el caso NRBFN

	NRMSE Training					NRMSE Test				
m	**$NRMSE_C$**	**DevS**	**$NRMSE_T$**	**DevS**	**$NRMSE_{min}$**	**$NRMSE_C$**	**DevS**	**$NRMSE_{Test}$**	**DevS**	**$NRMSE_{min}$**
4	1.1	1E-2	0.69	1E-2	0.60	2.19	2E-2	0.742	4E-2	0.729
5	0.99	4E-2	0.54	3E-2	0.50	2.25	3E-2	0.596	7E-2	0.519
6	0.87	6E-2	0.50	1E-2	0.39	1.33	2E-2	0.437	3E-2	0.402
7	0.74	3E-2	0.40	2E-2	0.38	1.13	3E-2	0.397	3E-2	0.369
8	0.76	5E-2	0.38	3E-2	0.36	1.16	1E-2	0.396	2E-2	0.273
9	0.69	2E-2	0.32	3E-2	0.25	0.96	5E-2	0.311	3E-2	0.222
10	0.65	5E-2	0.29	3E-2	0.26	1.04	2E-2	0.275	5E-2	0.233
11	0.59	6E-2	0.27	4E-2	0.25	0.725	1E-2	0.281	3E-2	0.248
12	0.57	3E-2	0.24	1E-2	0.21	0.741	2E-2	0.279	1E-2	0.206

Tabla 4.21.*b*): Errores obtenidos por ECFA para la función $g(x_1,x_2)$ con ruido adicional del 10% en el caso RBFN

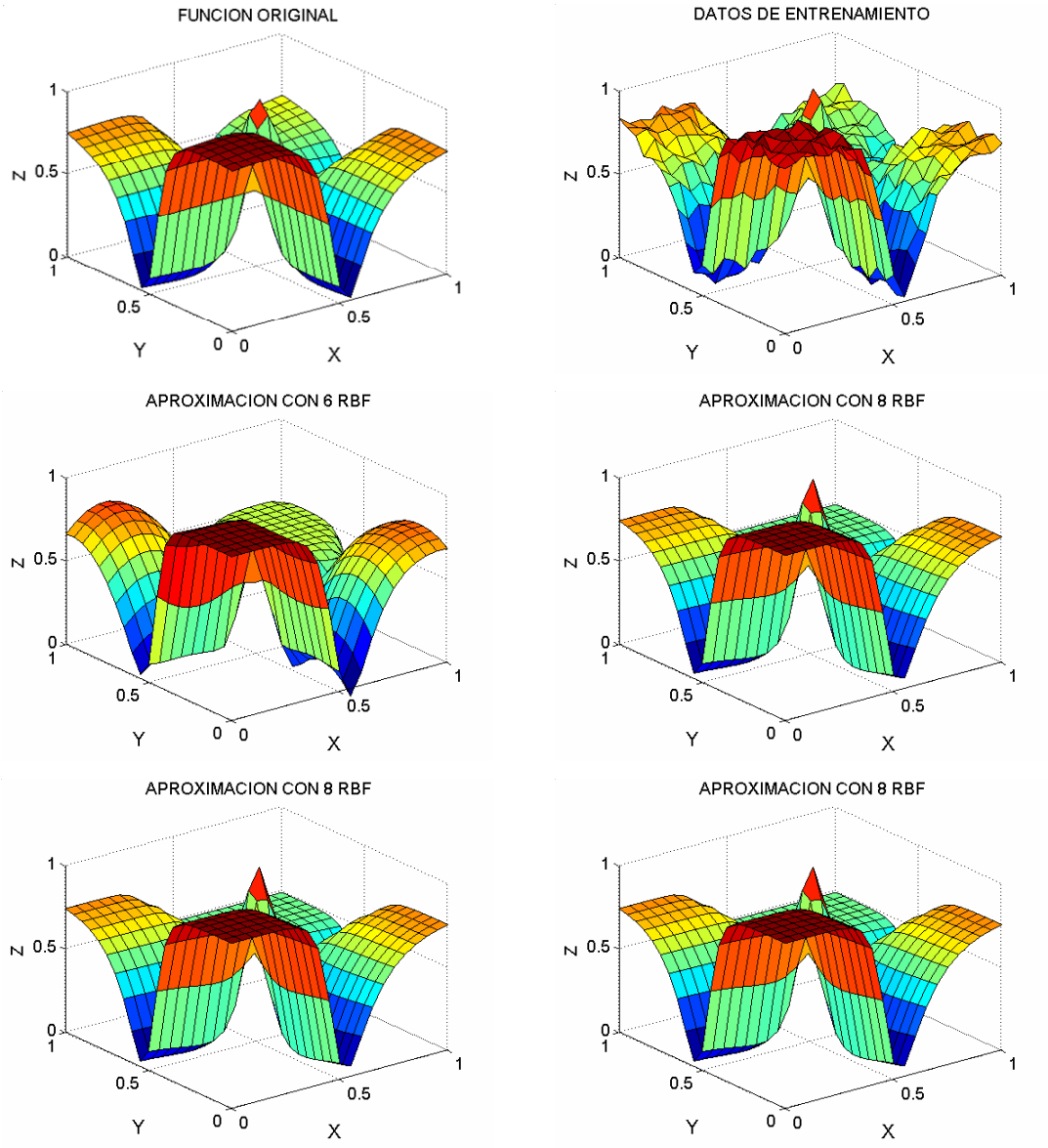


Fig. 4.86 Distintas aproximaciones para la función $g(x_1,x_2)$ con un conjunto de entrenamiento al que se le ha añadido 10% de ruido.

4.6 Efecto del número y la distribución de datos sobre el algoritmo ECFA

Cuando se trata con problemas de aproximación de funciones a partir de un conjunto de vectores de entrada y salida, la cantidad y la calidad de los datos de entrenamiento utilizados es una cuestión importante. Cuando mayor sea el número de datos, mejor estará representada la función original que queremos aproximar. En todas las pruebas

anteriores se ha elegido siempre un número de datos grande, dependiendo en esto de la dimensionalidad de la función de manera que la función estará siempre bien representada. Para realizar una estimación de la influencia del número de datos de entrenamiento en el resultado final, consideremos distintas cantidades de muestras de las funciones *dick(x)* y $f_8(x_1,x_2)$ definidas en las ecuaciones (4.9) y (4.14), y que fueron representados en las figuras 4.25 y 4.55.

- Función *dick(x)*

  Para esta función se ha utilizado número de datos de 50, 100 y 200 datos equidistribuidos en el rango de entrada. En la Tabla 4.22 y Tabla 4.23 se presentan los resultados obtenidos después del proceso de *clustering* y después de proceso de minimización, donde se han utilizado 1000 datos de test.

RBF	50 datos		100 datos		200 datos	
	$NRMSE_T$	$NRMSE_{test}$	$NRMSE_T$	$NRMSE_{test}$	$NRMSE_T$	$NRMSE_{test}$
4	0.8632	1.9651	0.8851	2.0644	0.8883	2.1624
5	0.8239	1.5683	0.8783	1.8252	0.8948	2.0087
6	0.5216	0.7191	0.5340	0.6037	0.5324	0.6165
7	0.4933	0.6864	0.5167	0.5742	0.5638	0.6644
8	0.3373	0.5357	0.3809	0.3900	0.3862	0.4077
9	0.3190	0.5211	0.3493	0.3491	0.3422	0.3515
10	0.2574	0.4855	0.2841	0.2780	0.2901	0.2925

Tabla 4.22: La media del error de entrenamiento y test con distintos números de datos equidistribuidos en el caso de la función *dick(x)* después del proceso de clustering.

RBF	50 datos		100 datos		200 datos	
	$NRMSE_T$	$NRMSE_{test}$	$NRMSE_T$	$NRMSE_{test}$	$NRMSE_T$	$NRMSE_{test}$
4	0.4044	0.6841	0.0380	0.0384	0.1653	0.1641
5	0.0499	0.4399	0.0134	0.0135	0.0787	0.0787
6	0.0060	0.4355	0.0041	0.0042	0.0027	0.0028
7	0.0141	0.4344	0.0066	0.0067	0.0019	0.0019
8	0.0074	0.4352	0.0052	0.0052	0.0048	0.0048
9	0.0035	0.4352	0.0045	0.0045	0.0038	0.0037
10	0.0079	0.4350	0.0076	0.0077	0.0028	0.0028

Tabla 4.23: La media del error de entrenamiento y test con distintos números de datos equidistribuidos en el caso de la función *dick(x)* después del proceso de minimización.

Como se puede comprobar en las tablas anteriores, esta función estaría bien representada con 100 o más datos, pero 50 datos serían insuficientes.

- Función $f_8(x_1,x_2)$

  Para esta función se ha utilizado número de datos de 50, 100 y 200 datos distribuidos de forma homogénera en el rango de entrada. En la Tabla 4.24 y Tabla 4.25 se presentan los resultados obtenidos después del proceso de *clustering* y después de proceso de minimización, donde se han utilizado 10000 datos de test.

RBF	100 datos		200 datos		400 datos	
	$NRMSE_T$	$NRMSE_{test}$	$NRMSE_T$	$NRMSE_{test}$	$NRMSE_T$	$NRMSE_{test}$
4	1.0274	4.7035	1.0298	5.2431	1.0550	1.1371
5	1.0244	6.3597	1.0270	8.4385	1.0345	1.0949
6	1.0384	5.2409	1.0391	6.8721	1.0533	1.0348
7	1.0429	5.7493	1.0327	5.7193	1.0489	1.0792
8	1.0298	4.9188	1.0253	6.7724	1.0365	1.0248
9	0.9898	3.2341	0.8248	2.1626	0.8904	1.0163
10	0.8143	1.6885	0.8603	1.4570	0.8290	1.0263

Tabla 4.24: La media de el error de entrenamiento y test en distintas número de datos equidistribuidos en el caso de la función $f_8(x_1,x_2)$ después del proceso de clustering

RBF	100 datos		200 datos		400 datos	
	$NRMSE_T$	$NRMSE_{test}$	$NRMSE_T$	$NRMSE_{test}$	$NRMSE_T$	$NRMSE_{test}$
4	0.8982	1.8685	0.9360	1.8151	0.8995	0.9467
5	0.3752	0.8273	0.3671	0.6162	0.6147	0.6133
6	0.3708	0.7446	0.4284	0.7036	0.4564	0.5335
7	0.4205	0.8115	0.2551	0.5430	0.3122	0.5518
8	0.2276	0.6663	0.3492	0.5995	0.3073	0.5655
9	0.3165	0.7729	0.3056	0.5940	0.3237	0.3559
10	0.1980	0.6648	0.2909	0.5417	0.2997	0.3259

Tabla 4.25: La media de el error de entrenamiento y test en distintas número de datos equidistribuidos en el caso de la función $f_8(x_1,x_2)$ después del proceso de minimización

De los resultados para la compleja función $f_8(x_1,x_2)$ podemos observar que utilizar un número demasiado pequeño de datos puede conducir a una falsa sensación de buena aproximación. En este caso, se necesitarían al menos unos 400 datos para que la diferencia entre el error de entrenamiento y el de test no difiriera de forma significativa.

4.7 Uso de ECFA para el modelado y predicción de series temporales

Esta sección presenta experimentos que muestran cómo se comporta el algoritmo propuesto de *clustering* para la predicción de valores futuros a partir de información histórica. Se ha usado la ecuación diferencial propuesta por Mackey y Glass [MAC-77]. Este tipo de series temporales se suele utilizar como patrón de test en la mayoría de algoritmos de identificación y optimización de modelos. La serie de Mackey-Glass, concebida inicialmente para el modelado de la concentración de leucocitos en la sangre, se genera a partir de la siguiente ecuación diferencial:

$$\frac{ds(t)}{dt} = \alpha \cdot \frac{s(t-\tau)}{1+s^{10}(t-\tau)} - \beta s(t) \tag{4.19}$$

Los parámetros libres se fijan a $\alpha = 0.2$ y $\beta = 0.1$, lo que convierte la ecuación diferencial anterior en una serie temporal caótica sin un período claramente definido, que no converge ni diverge, y es muy sensible a las condiciones iniciales [MAC-77].

Para encontrar una solución numérica a la ecuación diferencial, se ha aplicado el método de Runge-Kutta de cuatro órdenes como se hizo en [JAN-93]. Los valores iniciales se han fijado a $s(0) = 1.2$ y $s(t) = 0$ cuando $t < 0$, haciéndose $\tau = 17$.

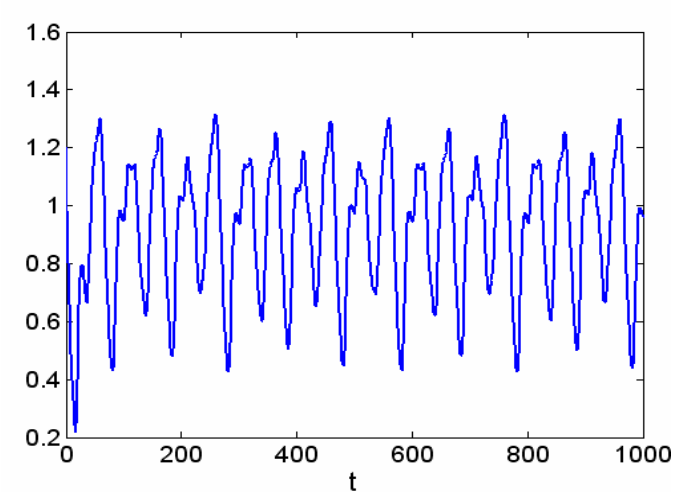


Fig. 4.87 La serie temporal de Mackey-Glass

- Predicción a corto plazo

En esta sección, aplicamos nuestro algoritmo para buscar redes que predigan el valor $s(t+6)$ a partir de valor actual $s(t)$ y de los valores pasados $s(t-6), s(t-12)$, y $s(t-18)$ empleando por tanto vectores de entrenamiento de la forma:

$$\left[s(t-18), s(t-12), s(t-6), s(t);\ s(t+6)\right] \tag{4.20}$$

En los trabajos que utilizan esta serie temporal, se han usado los primeros 500 vectores para entrenar la red, y los 500 vectores siguientes para validarla. Los resultados se realizan con RBFN y *Normalized* RBFN.

RBFs	$NRMSE_T$	$NRMSE_{min}$	DesV	$NRMSE_{Test}$	$NRMSE_{min}$	DesV
2	0.3014	0.3014	0.0	0.2956	0.29564	0
3	0.0672	0.06703	0.0003	0.0668	0.06664	0.0002
4	0.0557	0.03839	0.0152	0.0558	0.03961	0.0142
5	0.0339	0.03093	0.0036	0.0349	0.03222	0.0037
6	0.0191	0.01787	0.0014	0.0205	0.01910	0.0015
7	0.0177	0.01670	0.0013	0.0191	0.01815	0.0012
8	0.0167	0.01421	0.0027	0.0183	0.01600	0.0025
9	0.0138	0.01344	0.0005	0.0157	0.01531	0.0004
10	0.0124	0.01168	0.0010	0.0146	0.01399	0.0008
11	0.0111	0.01079	0.0005	0.0135	0.01318	0.0005
12	0.0098	0.00970	0.0002	0.0124	0.01212	0.0003
13	0.0088	0.00829	0.0006	0.0116	0.01134	0.0004
14	0.0086	0.00828	0.0003	0.0113	0.01115	0.0001
15	0.0083	0.00799	0.0003	0.0113	0.01108	0.0003
16	0.0073	0.00677	0.0006	0.0103	0.00999	0.0005
17	0.0068	0.00665	0.0002	0.0099	0.00972	0.0001
18	0.0065	0.00630	0.0002	0.0099	0.00977	0.0002
19	0.0066	0.00625	0.0003	0.0099	0.00965	0.0003
20	0.0064	0.00623	0.0003	0.0099	0.00964	0.0003

Tabla 4.26: Resultados obtenidos por nuestro algoritmo para la predicción de serie de MG para un valor $s(t+6)$ en el caso de NRBFN

RBFs	$NRMSE_T$	$NRMSE_{min}$	DesV	$NRMSE_{Test}$	$NRMSE_{min}$	DesV
2	0.1282	0.1278	0.0007	0.1276	0.1273	0.0005
3	0.0660	0.0558	0.0089	0.0659	0.0551	0.0094
4	0.0458	0.0405	0.0086	0.0459	0.0411	0.0080
5	0.0371	0.0350	0.0018	0.0377	0.0360	0.0015
6	0.0338	0.0278	0.0053	0.0341	0.0290	0.0044
7	0.0254	0.0246	0.0008	0.0267	0.0259	0.0008
8	0.0225	0.0223	0.0002	0.0239	0.0237	0.0002
9	0.0195	0.0176	0.0019	0.0209	0.0192	0.0017
10	0.0180	0.0176	0.0005	0.0195	0.0191	0.0006
11	0.0146	0.0134	0.0011	0.0163	0.0153	0.0010
12	0.0157	0.0149	0.0009	0.0173	0.0168	0.0009
13	0.0138	0.0129	0.0012	0.0156	0.0147	0.0012
14	0.0126	0.0106	0.0018	0.0145	0.0125	0.0018
15	0.0110	0.0089	0.0026	0.0131	0.0114	0.0023
16	0.0090	0.0084	0.0007	0.0114	0.0109	0.0006
17	0.0093	0.0091	0.0001	0.0116	0.0115	0.0002
18	0.0093	0.0086	0.0006	0.0117	0.0110	0.0007
19	0.0075	0.0070	0.0004	0.0102	0.0099	0.0002
20	0.0083	0.0079	0.0005	0.0108	0.0103	0.0005

Tabla 4.27: Resultados obtenidos por nuestro algoritmo para la predicción de serie de MG para un valor $s(t+6)$ en el caso de RBFN

RBFs	[González 2001]		[Rivas 2003]		Algoritmo Propuesto	
	$NRMSE_{Test}$	DesV	$NRMSE_{Test}$	DesV	$NRMSE_{Test}$	DesV
3	0.0940	0.0134	0.0807	0.0153	0.0551	0.0094
4	0.0662	0.0084	0.0612	0.0095	0.0459	0.0080
6	0.0394	0.0062	0.0474	0.0121	0.0341	0.0044
8	0.0281	0.0052	0.0369	0.0028	0.0239	0.0002
10	0.0243	0.0042	0.0333	0.0043	0.0195	0.0006
12	0.0207	0.0041	0.0277	0.0030	0.0173	0.0009
14	0.0196	0.0031	0.0256	0.0030	0.0145	0.0018
16	0.0159	0.0026	0.0234	0.0014	0.0114	0.0006
18	0.0140	0.0031	0.0230	0.0020	0.0117	0.0007
20	0.0126	0.0025	0.0204	0.0026	0.0108	0.0005

Tabla 4.28: Comparativa de resultados para la predicción de la serie de Mackey-Glass para un valor $s(t+6)$

- Predicción a largo plazo

En esta sección utilizamos el algoritmo propuesto para buscar redes que predigan el valor $s(t+85)$ a partir de valor actual $s(t)$ y de los valores pasados

$s(t-6), s(t-12)$, y $s(t-18)$ empleando por tanto vectores de entrenamiento de la forma:

$$\left[s(t-18), s(t-12), s(t-6), s(t);\ s(t+85)\right] \tag{4.21}$$

En los trabajos que utilizan esta serie temporal, se han usado los primeros 500 vectores para entrenar la red, y los 500 vectores siguientes para validarla. Los resultados se realizan con RBFN y *Normalized* RBFN

RBFs	$NRMSE_T$	$NRMSE_{min}$	DesV	$NRMSE_{Test}$	$NRMSE_{min}$	DesV
2	0.4876	0.4876	0	0.4474	0.4474	0.0048
3	0.4064	0.4035	0.0049	0.3805	0.3777	0.0095
4	0.3707	0.3653	0.0090	0.3520	0.3462	0.0002
5	0.3600	0.3600	0.0001	0.3407	0.3407	0.0158
6	0.3272	0.3160	0.0162	0.3045	0.2930	0.0041
7	0.2743	0.2677	0.0060	0.2574	0.2529	0.0218
8	0.2504	0.2280	0.0206	0.2321	0.2074	0.0168
9	0.2240	0.2049	0.0170	0.2075	0.1881	0.0127
10	0.1812	0.1673	0.0122	0.1687	0.1555	0.0210
11	0.1849	0.1672	0.0181	0.1702	0.1515	0.0174
12	0.1889	0.1711	0.0154	0.1746	0.1546	0.0276
13	0.1698	0.1482	0.0303	0.1555	0.1346	0.0091
14	0.1436	0.1369	0.0077	0.1327	0.1255	0.0225
15	0.1526	0.1220	0.0267	0.1414	0.1156	0.0094
16	0.1207	0.1059	0.0130	0.1121	0.1013	0.0177
17	0.1226	0.1111	0.0200	0.1151	0.1024	0.0332
18	0.1107	0.0697	0.0368	0.1056	0.0679	0.0144
19	0.0955	0.0855	0.0151	0.0884	0.0773	0.0021
20	0.0780	0.0713	0.0081	0.0752	0.0727	0.0048

Tabla 4.29: Resultados obtenidos por nuestro algoritmo para la predicción de serie de MG para un valor $s(t+85)$ en el caso de NRBFN

RBFs	$NRMSE_T$	$NRMSE_{min}$	DesV	$NRMSE_{Test}$	$NRMSE_{min}$	DesV
2	0.4678	0.4668	0.0009	0.4329	0.4275	0.0047
3	0.4337	0.4336	0.0001	0.4014	0.4014	0.0000
4	0.4157	0.3864	0.0256	0.3862	0.3605	0.0224
5	0.4129	0.3921	0.0229	0.3879	0.3675	0.0227
6	0.4009	0.3979	0.0051	0.3774	0.3720	0.0058
7	0.3743	0.3620	0.0205	0.3520	0.3413	0.0165
8	0.3354	0.3135	0.0229	0.3157	0.2949	0.0216
9	0.2912	0.2846	0.0111	0.2753	0.2684	0.0107
10	0.2580	0.2484	0.0144	0.2432	0.2347	0.0130
11	0.2330	0.2062	0.0326	0.2208	0.1942	0.0297
12	0.2322	0.2123	0.0184	0.2185	0.1990	0.0182
13	0.1806	0.1550	0.0233	0.1739	0.1515	0.0215
14	0.1590	0.1384	0.0314	0.1500	0.1316	0.0303
15	0.1584	0.1438	0.0135	0.1470	0.1315	0.0139
16	0.1390	0.1271	0.0123	0.1311	0.1170	0.0144
17	0.1181	0.1058	0.0165	0.1108	0.0992	0.0156
18	0.1286	0.1185	0.0088	0.1220	0.1181	0.0035
19	0.1051	0.0916	0.0126	0.1021	0.0955	0.0070
20	0.1070	0.1011	0.0075	0.0972	0.0918	0.0074

Tabla 4.30: Resultados obtenidos por nuestro algoritmo para la predicción de serie de MG para un valor $s(t+85)$ en el caso de RBFN

RBFs	[González 2001]		[Rivas 2003]		Algoritmo Propuesto	
	$NRMSE_{Test}$	DesV	$NRMSE_{Test}$	DesV	$NRMSE_{Test}$	DesV
3	0.4883	0.0214	0.4545	0.0146	0.3805	0.0095
4	0.4440	0.0218	0.4366	0.0083	0.3520	0.0002
6	0.3606	0.0222	0.3808	0.0250	0.3045	0.0041
8	0.2999	0.0097	0.3050	0.0266	0.2321	0.0168
10	0.2513	0.0246	0.2493	0.0207	0.1687	0.0210
12	0.2138	0.0058	0.2030	0.0191	0.1702	0.0276
14	0.1977	0.0164	0.1675	0.0210	0.1327	0.0225
16	0.1507	0.0193	0.1409	0.0126	0.1121	0.0177
18	0.1297	0.0175	0.1239	0.0048	0.1056	0.0144
20	0.1268	0.0174	0.1133	0.0125	0.0884	0.0048

Tabla 4.31: Comparativa de resultados para la predicción de la serie de Mackey-Glass para un valor $s(t+85)$

4.8 Conclusiones

En este capítulo se han presentado los resultados experimentales obtenidos por el algoritmo propuesto ECFA comparados con algoritmos de *clustering* tradicionales

especialmente comparados con el algoritmo CFA, ambos aplicados a funciones de una, dos y varias dimensiones, con las mismas características del número de datos de entrada. Los resultados experimentales han sido aplicados a cuatro tipos de redes neuronales diferentes: NRBFN, RBFN, NWNN y WNN (ver Capítulo 2, Secciones 2.1 y 2.2).

- De los resultados experiméntales y de la comparación entre nuestro algoritmo ECFA y el algoritmo CFA, ECFA mejora los inconvenientes principales del algoritmo CFA. El largo tiempo de ejecución y la adaptabilidad a un número grande de puntos. Esto se observa en los resultados de una función de cuatro variables y de seis variables.

- De los resultados experimentales y en los cuatro tipos de redes neuronales utilizadas, se muestra que el algoritmo propuesto ECFA tiene mejor resultados en el error NRMSE y menor tiempo de ejecución. Esto se deriva del hecho que ECFA tiene en cuenta el tipo de red aproximadora que se va a utilizar.

- El algoritmo ECFA tiene la capacidad de tratar con funciones ruidosas, como se muestran los resultados, lo cual muestra la robustez del algoritmo propuesto.

- El algoritmo ECFA presenta unos resultados excelentes cuando se aplica al difícil problema del modelado y predicción de series temporales.

CAPÍTULO 5

ESTRUCTURA JERÁRQUICA MULTI-RBFN

Este capítulo propone una nueva técnica que se concentrará en la búsqueda de nuevas arquitecturas para modelar sistemas complejos de aproximación funcional a partir de un conjunto de muestras de E/S, a fin de evitar el aumento exponencial de la complejidad del sistema que es habitual cuando se trata con número alto de variables de entrada. La estructura jerárquica (Multi-RBFN) propuesta está formada por una red de sub-redes de funciones de base radial (Sub-RBFN), con la propiedad de que cada Sub-RBFN puede encargarse de un conjunto de variables de entrada y no de todas.

Para poder utilizar este tipo de arquitecturas se presenta un nuevo método para la selección de las variables de entrada más importantes, que es capaz de decidir cuáles de las variables seleccionadas van solas o juntas a una Sub-RBFN para componer la estructura jerárquica Multi-RBFN más adecuada. Finalmente se propone un método para la optimización conjunta de todos los parámetros involucrados en la red jerárquica, basado en el método de clustering presentado en los capítulos anteriores.

5.1 Introducción

En muchos problemas prácticos de modelado de sistemas es posible medir el valor de muchas señales físicas (variables), pero no es fácil conocer cuáles de estas variables son relevantes y necesarias para solucionar un problema dado [VEH-02]. El desarrollo aleatorio de modelos multi-variantes requiere una muy alta complejidad computacional para aplicaciones industriales y médicas, o cuando el mejor conjunto de las variables de entrada es desconocido [BAC-99]. Los problemas principales con los que nos enfrentamos son que, cuando aumenta la dimensionalidad de las variables de entrada, la complejidad computacional y las exigencias de memoria del modelo aumentarán (en algunos casos exponencialmente) y el aprendizaje será más difícil con entradas innecesarias.

Las redes neuronales pueden ser definidas como unas arquitecturas que contienen masivamente elementos adaptables de procesamiento paralelo interconectados vía estructuras de redes. Los parámetros en las redes neuronales tradicionales por lo general no tienen ningún sentido físico, y los valores iniciales son típicamente seleccionados de forma aleatoria. Es decir, cada parámetro individual puede no estar directamente relacionado con valores de salida o entrada. Por lo tanto, en general, es imposible explicar o señalar el sentido de los parámetros de las redes neuronales. Las redes neuronales tradicionales pueden ser tratadas como cajas negras cuyos parámetros son obtenidos a partir de un conjunto de datos de E/S, mediante métodos de minimización local.

El diseño de las redes neuronales artificiales (ANNs) depende de las aplicaciones elegidas. Las redes neuronales, en general, ofrecen alguna medida de la tolerancia y propiedades de representación distribuidas. Más importante es que poseen capacidades de aprendizaje adaptables de estimar funciones, codificar el conocimiento estructural y deducir relaciones de E/S vía la asociación [TAU-02]. Su fuerza principal es considerar unidades ocultas suficientes para resolver un problema dado. Las RBFNs son aproximadores universales, corresponden a una clase particular de la aproximación funcional que puede ser entrenado usando un conjunto de muestras de E/S. Por otra parte, cuando la estructura de una RBFN es totalmente llana (no jerárquica), una

consecuencia directa es la producción de una red abultada, con un número grande de unidades ocultas que producen una cantidad grande de elementos computacionales que, por su parte, obstaculiza la convergencia del proceso de aprendizaje y aumenta la exigencia de memoria y tiempo. Una eficaz solución es incorporar una estructura jerárquica adecuada para resolver problemas de modelos complejos. Las ideas de modular redes neuronales son esenciales para diseños jerárquicos [TAU-02].

Las estructuras jerárquicas tienen una variedad de utilización en la informática como representaciones que pueden ser formadas, modificadas y por otra parte, manipuladas [KWA-93]. Por ejemplo, en [SOU-02], de Souza propuso un nuevo modelo híbrido neuro-difuso que denominó *quadtree* neuro-difuso jerárquico (HNFQ). Este modelo esta basado en un método de partición recurrente del espacio de entrada denominado *quadtree*. Fukumizu y Amari en [FUK-99] investigaron la estructura jerárquica geométrica de los parámetros del espacio de una red perceptrón de tres capas a fin de mostrar la existencia de mínimos locales. Los autores demostraron que un punto crítico del modelo con h de unidades ocultas siempre da muchos puntos críticos del modelo con $h+1$ de unidades ocultas. También, en [FER-04] presentaron una red jerárquica de función de base radial cuya característica principal está constituida de capas jerárquicas, que contienen una función gausiana en una escala decreciente, insertadas sólo donde el error local se produce sobre un cierto umbral que garantiza un error uniforme residual y la asignación de más unidades con escalas más pequeñas donde los datos contienen frecuencias.

En nuestra hipótesis, las estructuras jerárquicas podrían ser usadas para dirigir la construcción conveniente de un sistema jerárquico denominado Multi-RBFN [AWA-05b] que mejora considerablemente el rendimiento de los problemas de aproximación funcional complejos. Este modelo de estructura jerárquica tiene algunos componentes básicos: la utilización de un método propuesto para el problema de selección de variables de entrada (IVS) para modelos complejos de la aproximación de función que produce la organización adecuada de la estructura jerárquica Multi-RBFN. El método propuesto para la sección de las variables de entrada trata de reconocer las variables más importantes y dirigir estas variables a cada Sub-RBFN en la estructura Multi-RBFN. El número de Sub-RBFNs depende del número seleccionado de variables de entrada y de

cuáles de estas variables deben ir solas o juntas a una Sub-RBFN. Así, depende en esto de la función o del problema que se trata de aproximar. Otro componente es la optimización de los parámetros de esta estructura propuesta, es decir, centros *c*, radios *r*, pesos *w*, y el número de funciones de base radial *RBF* en cada Sub-RBFN.

Una limitación fundamental en el problema de aproximación funcional es que, cuando aumente el número de variables de entrada, el número de parámetros aumentará exponencialmente. Este fenómeno denominado como la maldición de la dimensionalidad [BEN-00] previene la utilización de la mayoría de los sistemas aproximadores convencionales y obliga a buscar soluciones más específicas. La selección de variables de entrada (Input Variable Selection, IVS) está dirigida a la determinación de las variables de entrada que se requieren para un modelo. Su tarea es determinar un subconjunto del conjunto original de las variables de entrada que conducirá a un modelo más eficaz y adecuado para aproximar el conjunto de vectores de E/S. Las ventajas de la selección de variables de entrada consisten en reducir la dimensionalidad del espacio de las variables de entrada y quitar los datos redundantes, irrelevantes o ruidosos. Los efectos inmediatos para estas tareas son la subida de la velocidad de la ejecución del algoritmo de aprendizaje y la mejora significativa de la generalización del modelo obtenido para datos con los que no se ha aprendido.

La selección de variable de entrada puede solucionar problemas que ocurren debido a la selección de mala calidad de entradas como: los aumentos de dimensionalidad de los datos de entrada; la complejidad computacional y el aumento de las exigencias de memoria [BAC-99]. Los procesos de aprendizaje son más difíciles con entradas no requeridas y la exactitud del modelo puede resultar mala con entradas adicionales no requeridas, y la interpretabilidad de modelos complejos más difícil que el de modelos simples [BAC-99].

La selección de variables de entrada ha sido investigada y ha sido aplicada en varios problemas como: la minería de datos (*data mining)* y la extracción del conocimiento *(knowledge extraction)* [FRA-92]. Uno de los métodos más populares que normalmente se utiliza para seleccionar las variables de entrada es el análisis de componentes principales (PCA) [COH-99]. Este método se utiliza en problemas de clasificación para

seleccionar algún subconjunto de las variables originales, de modo que la exactitud de clasificación mantiene el nivel deseado o puede ser hasta mejorado [ONN-01].

Varios autores han trabajado para seleccionar las variables de entrada en los problemas de aproximación de funciones: Pomares, en [POM-02], presentó un método rápido de obtener la estructura de un sistema completo de base de reglas difusas para una exactitud de aproximación específica de los datos de entrenamiento. Este método podía decidir qué variables de entrada deben ser consideradas en el sistema difuso y cuántas funciones de pertenecía (MFs) son necesarias en cada variable de entrada seleccionada a fin de alcanzar el objetivo de aproximación con el número mínimo de parámetros. Vehtari, en [VEH-02], propuso usar probabilidades posteriores y marginales posteriores para averiguar combinaciones de entrada potencialmente útiles que seleccionan un modelo final y evalúan las utilidades esperadas (con cualquier utilidad deseada). Chen, en [CHE-03], propuso un conjunto de entrada y variables de salida, donde una partición difusa asocia conjuntos difusos con cada variable. Hay dos modos de hacerlo: partición de datos independientes y partición de datos dependientes. El antiguo proceso divide el espacio de entrada en una manera predeterminada. Una de las estrategias comúnmente usadas es adjudicar el número fijado de etiquetas lingüísticas a cada variable de entrada. La partición del espacio de salidas, entonces, surge del aprendizaje supervisado.

Como ya hemos comentado anteriormente, en problemas de aproximación de funciones, cuando aumenta el número de variables de entrada, el número de parámetros aumenta exponencialmente. Una consecuencia directa es tener una red abultada, con un número grande de neuronas en la capa oculta que, por su parte, obstaculiza la convergencia del proceso del entrenamiento de la red. La estructura jerárquica de redes de función de base radiales (Multi-RBFN) propuesta divide al principio el problema de la aproximación de función en problemas más pequeños basados en el número de las variables de entrada más importantes que han sido seleccionadas y cuáles de estas variables seleccionadas van solas o juntas en una Sub-RBFN. La estructura final es una serie jerárquica de redes de función de base radiales RBFNs conectadas en paralelo con una salida total que es la suma lineal de todas las salidas de las Sub-RBFNs. Esta topología facilita solucionar problemas que pueden ser combinados para acceder a una solución compleja.

Esta nueva técnica se concentra en la búsqueda de nuevas arquitecturas de cálculo capaces de formar sistemas complejos de RBFNs sin que el aumento del número de variables de entrada tenga que suponer un aumento exponencial en la complejidad del sistema. Nuestro objetivo es encontrar un modelo con el número más pequeño de variables de entrada que tienen, según las estadísticas, o prácticamente al menos, la misma utilidad esperada del modelo completo con todas las entradas disponibles. Propusimos un método bueno de la estimación de las variables de entrada más importantes; este método trata de relacionar cada dimensión de los datos de entrada a la salida objetivo (como una función de una dimensión) y divide los datos de esta dimensión en partes. Para cada una de estas partes se calcula la distancia entre el máximo y el mínimo de los valores de la salida que pertenecen a los datos de cada dimensión en cada parte y el promedio de todas las distancias en todas las partes. Cuando el promedio tiene un valor pequeño, la variable es más importante y tiene que ser seleccionada. Las variables que tienen promedio grande son variables de ruido que no afectan mucho en la función, y deberían ser eliminadas. Cuando selecciona las variables de entrada más importantes, el número de Sub-RBFNs depende del número de estas variables y las cuales van solas o juntas en una Sub-RBFN y esto depende de la función o del modelo que trata de aproximar. El proceso de decidir cuáles de las variables van solas o juntas depende, en general, del cálculo de la varianza de cada variable o conjunto de variables relacionado con la salida objetivo.

Una vez se sepa el número de Sub-RBFNs, se construye la estructura jerárquica Multi-RBFN. En cada Sub-RBFN, que presenta una red de funciones de pase radial (RBFN), se optimiza los parámetros de la red de función de base radial (Centros $\vec{c}_j^s$, Radios r^s, Pesos w_j^s), utilizando algoritmos de clustering para inicializar los valor de los centros $\vec{c}_j^s$ en cada Sub-RBFN. En esta memoria se utiliza un nuevo algoritmo de clustering para inicializar los valores de los centros diseñado para problemas de aproximación funcional presenta en capítulo 3. Para optimizar los valores de los radios r^s en cada Sub-RBFN, se utilizan algoritmos tradicionales como el método de los vecinos más cercanos (*knn*). Cuando los parámetros de centros $\vec{c}_j^s$ y radio r^s de cada Sub-RBFN han

sido inicializados, se utiliza un método de cálculo lineal para encontrar los valores exactos de los pesos w_j^s en todo el sistema jerárquico Multi-RBFN que minimiza la función de coste calculada sobre el conjunto de muestras de E/S.

El objetivo principal de la estructura jerárquica propuesta y de los diversos métodos y algoritmos para su correcta utilización es modelar sistemas complejos en problemas de aproximación de funciones. La metodología propuesta se presenta utilizando dos clases diferentes de resultados; por una parte, presenta la arquitectura adecuada del sistema Multi-RBFN y por otra parte, optimiza los parámetros de esta estructura jerárquica Multi-RBFN aplicada al problema de aproximación funcional a partir de un conjunto de datos de E/S.

5.2 La arquitectura del sistema Multi-RBFN

La mayor parte de las redes neuronales artificiales tienen un máximo de interconexiones entre capas de la red. La estrategia de aproximación usada en RBFNs consiste en aproximar una función desconocida con una combinación lineal de funciones no lineales denominadas funciones base. Las funciones base son funciones radiales es decir, tienen la simetría radial con respecto a un centro. En las RBFNs, cada neurona en la capa oculta recibe todas las variables de entrada de la red. Sin embargo, las interconexiones en la estructura jerárquica Multi-RBFN entre variables de entrada y la capa oculta son muchas más limitadas y localizadas. La ventaja de la estructura jerárquica Multi-RBFN consiste en que el problema se divide en muchos problemas que se conectan en paralelo. Cada problema presenta una RBFN denominada Sub-RBFN. Todas las Sub-RBFN tienen una salida total que es la salida de la estructura jerárquica Multi-RBFN. Esta división del sistema Multi-RBFN limita la cantidad de la información de la capa anterior. En general, construir una estructura jerárquica Multi-RBFN para resolver problemas de aproximación de funciones consiste en tres pasos básicos: la identificación de la estructura (selección de las variables de entrada, división de las variables de entrada seleccionadas al número Sub-RBFNs, el número de Sub-RBFNs depende del número de las variables de entrada seleccionadas y de cuáles de estas variables van solas o juntas en una Sub-RBFN), la estimación de los parámetros de cada Sub-RBFN (centros $\vec{c}^S$, radios r^s, pesos w^S, y el número de funciones radiales

RBF en cada Sub-RBFN) y el cálculo la salida total *f(x)* de la estructuras jerárquicas Multi-RBFN. La función de la salida de la estructura Multi-RBFN es lineal; usamos un método de optimización lineal para encontrar los valores de los pesos w^S que minimizan la función de coste calculada sobre el conjunto de las muestras de E/S. La Fig. 5.1 presenta los principales pasos del algoritmo propuesto.

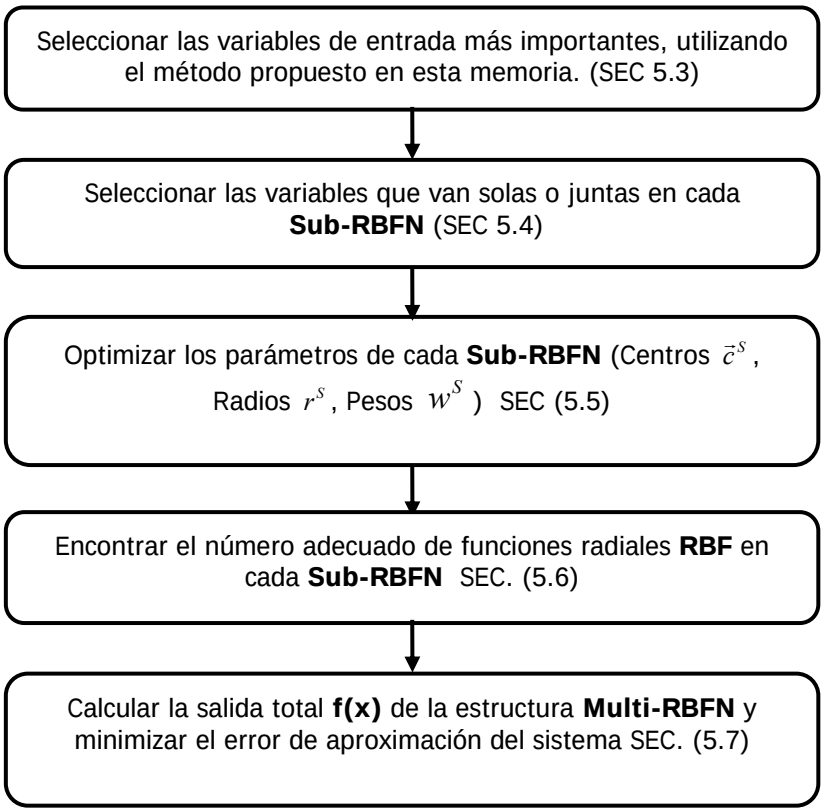


Fig. 5.1 Principales pasos del algoritmo propuesto

En la Figura 5.2 se muestra la estructura Multi-RBFN. Cada uno de los nodos de la figura es una red completa de RBFs (ver Fig. 5.3). Las RBFNs son una clase particular de redes neuronales artificiales, se caracterizan por una función de transferencia en la capa de unidades ocultas que tiene la simetría radial con respecto a un centro. La Fig. 5.3 muestra la arquitectura básica de un RBFN de 3 capas. El comportamiento de esta red y la expresión de su salida ya fueron presentados en la Sección 2.1 del Capítulo 2**.**

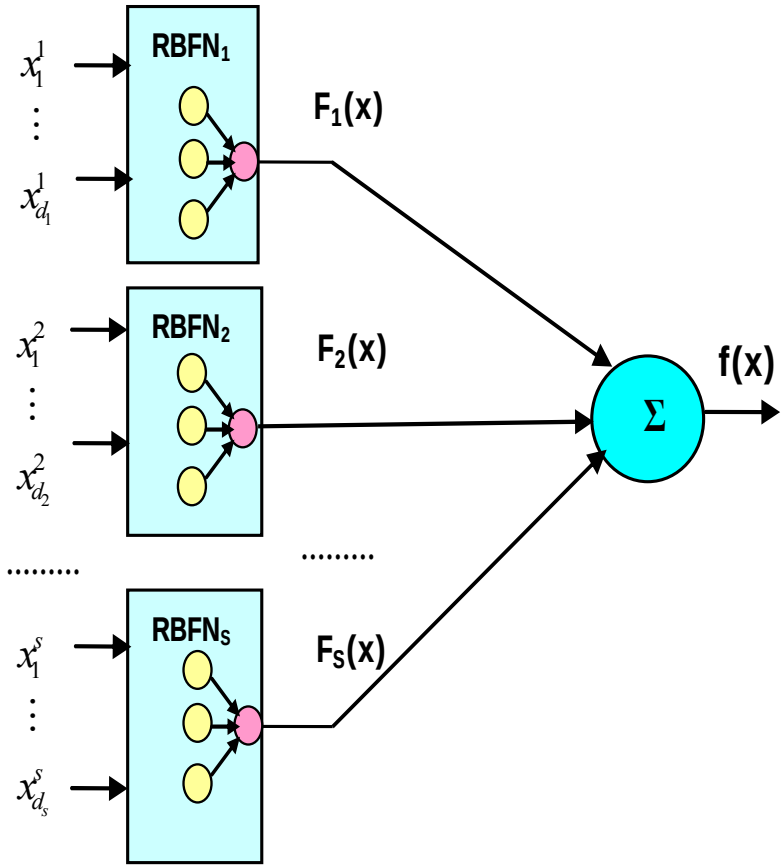


Fig. 5.2 La estructura jerárquica Multi-RBFN.

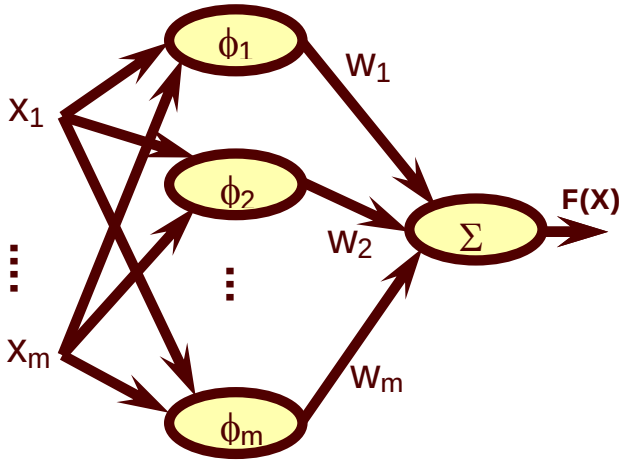


Fig. 5.3 Cada una de las sub-redes es una RBFN

El diseño de la arquitectura Multi-RBFN se muestra en la Fig. 5.2. El aprendizaje es diseñado para aproximar una función o un problema complejo a partir de un conjunto de ejemplos de E/S. La estructura completa se puede ver como una gran RBFN cuya capa oculta está formada por Sub-RBFN que son, a su vez, RBFNs. La conexión de la capa de entrada a la capa oculta depende del número seleccionado de variables de entrada. Nótese que la estructura Multi-RBFN debe ser conocida antes del proceso de la optimización de los parámetros en cada Sub-RBFN. La estructura Multi-RBFN consiste en conjuntos de RBFNs, con la característica de que cada RBFN recibe número de variables de entrada y no todas las variables, y todas las salidas de las RBFNs se suman de forma lineal para dar la salida total del sistema jerárquico Multi-RBFN.

Cada grupo de las variables de entrada se usan como entradas de cada Sub-RBFNs. Estas entradas que han sido seleccionadas dependen de un nuevo método de selección de variables de entrada que se presentará en la sección 5.3. Dada una entrada $\{x_1,\ldots, x_d\}$, donde *d* es el número de las dimensiones del espacio de los datos de entrada (datos de entrada originales), cada Sub-RBFN tiene una cierta probabilidad de tomar una entrada (el número mínimo de las variables de entrada) o cualquier número de entradas entre *1* y el número máximo de entradas *d*, dependiendo esto del número seleccionado de variables y cuáles de estas variables van solas o juntas. Este proceso depende de la función o del problema que pretendemos aproximar. Cada Sub-RBFN recibe variables y realiza el proceso de optimización de los parámetros de cada Sub-RBFN (centros $\vec{c}^{S}$, radios r^{S}). Para inicializar los centros $\vec{c}^{S}$, usamos el algoritmo de clustering presentado en capítulos anteriores [AWA-05a], y otros algoritmos heurísticos para inicializar los radios r^{S}. Cuando los parámetros de centros $\vec{c}^{S}$ y radios r^{S} de cada Sub-RBFN han sido optimizados, se usa un método de optimización lineal para encontrar los valores de los pesos w^{S}, que depende de la salida total del sistema Multi-RBFN, que minimiza la función de coste calculada sobre el conjunto de muestras de E/S.

El cálculo de los pesos w^{S} no depende de cada salida de cada Sub-RBFN $\{F_1(x),\ldots,F_S(x)\}$, sino depende de la salida total $f(x)$ del sistema Multi-RBFN, y debe ser calculado en la forma lineal como en la siguiente expresión:

$$f(\vec{x},\Phi,w)=\sum_{s=1}^{S}\sum_{i=1}^{m_s}\phi_i^s\left(\vec{x}\right)\cdot w_i^s \tag{5.1}$$

Donde $\Phi_i^s=\{\phi_i^s:\ i=1,...,n.\ S=1,...,s\}$ es la matriz de activación del conjunto de funciones base en todas las Sub-RBFNs, w_i^s los pesos asociados de todas las Sub-RBFN y *S* es el número de Sub-RBFNs. El proceso de la optimización lineal de los pesos w_i^s depende a la matriz de la función de activación de la salida total del Multi-RBFN $f(x)$. Este proceso utiliza los métodos directos como la descomposición de valor singular *(SVD)* [CLI-83] para calcular los valores de los pesos w_i^s.

Varias estructuras jerárquicas Multi-RBFN pueden ser obtenidas para cualquier problema dado de un conjunto de variables de entrada. Por ejemplo, para 4 número de variables de entrada seleccionadas $\{x_1,\ldots, x_4\}$, muchas arquitecturas posibles de pueden obtener, la más simple cuando cada variable de entrada forma un subconjunto de entrada, y cada Sub-RBFN recibe la entrada de una variable o cada Sub-RBFN toma entradas de dos o tres variables o cada Sub-RBFN toma todo el número de las variables de entrada.

Asimismo, muchas sub-arquitecturas de la estructura jerárquica Multi-RBFN son posibles con la condición de conservar el número de las variables de entrada seleccionadas y de que una variable no puede irse a dos Sub-RBFN distintas. Nuestro objetivo de elegir la estructura jerárquica Multi-RBFN adecuada depende de la función o del problema que pretendemos aproximar con esta arquitectura. En una red de funciones de base radial que recibe todas las entradas seccionadas como una RBFN típica, como en la Fig. 5.4.*d,* el número total de parámetros es igual a $m\cdot(d+2)$, donde *d* es el número de dimensiones del espacio de entradas y *m* el número de funciones radiales (RBF). Este número que produce la red en la Fig. 5.4.*d* es más grande que cualquier arquitectura de la Fig. 5.4 con el número de funciones radiales igual en todas. La Tabla 5.1 muestra la diferencia en el número de elementos de computación en cada estructura jerárquica Multi-RBFN.

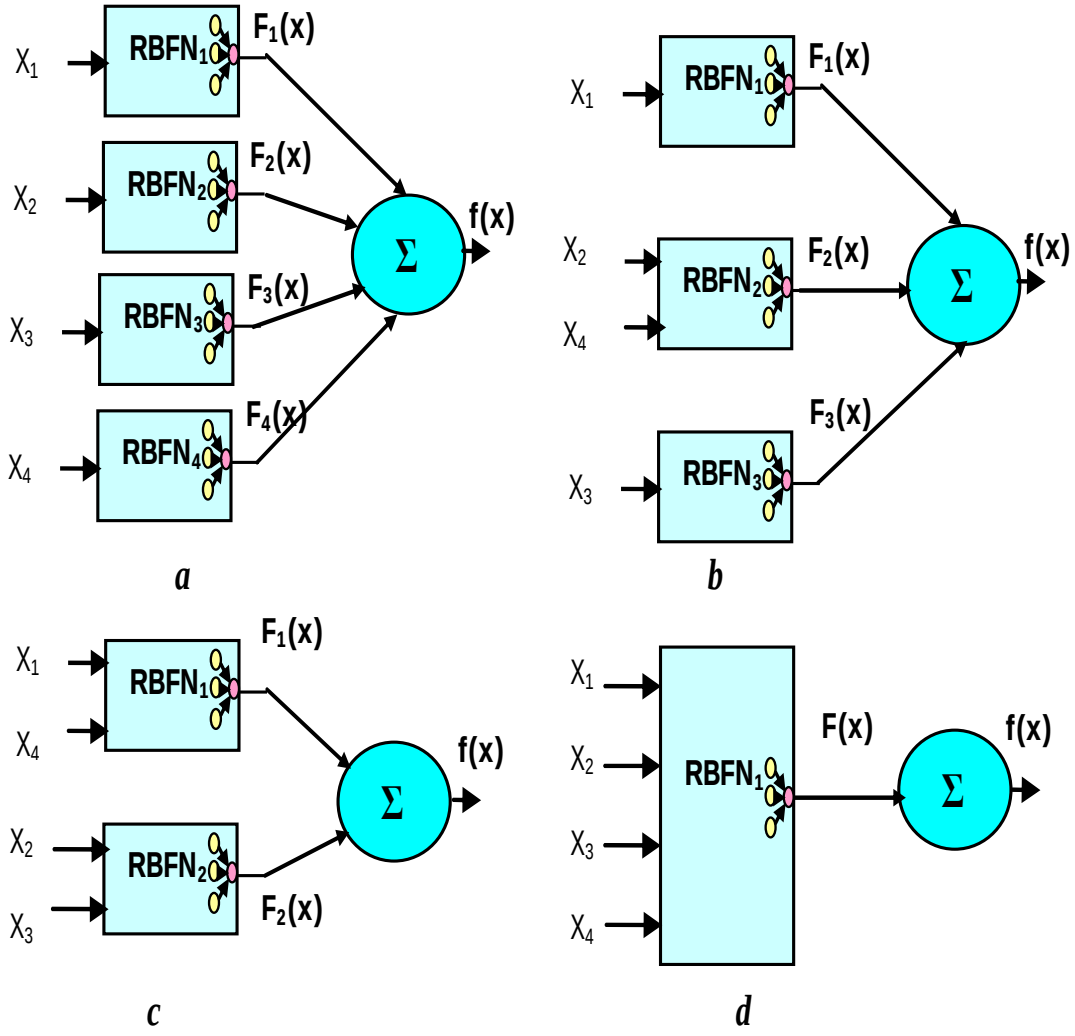


Fig. 5.4 Diferentes Topologías de estructuras jerárquicas Multi-RBFN. **a):** 4 Sub-RBFNs con una variable de entrada para cada Sub-RBFN **b):** 3 Sub-RBFNs con uno y dos variables para cada Sub-RBFN. **c):** 2 Sub-RBFNs con dos variables para cada Sub-RBFN. **d):** una Sub-RBFN con todo el conjunto de variables de entrada

La estructura jerárquica Multi-RBFN propuesta decrece el número de parámetros y produce un decremento de la complejidad del sistema aproximado e incrementa la eferencia del proceso de aproximación de funciones a partir de un conjunto de ejemplos de E/S [AWA-05b].

Nº Figura	Nº Sub-RBFN	Nº de RBF en cada Sub-RBFN	Nº de Variables en cada Sub-RBFN	Nº Parámetros en cada Sub-RBFN	Nº de parámetros en cada Multi-RBFN
5.4.a	4	6	1	18	72
		6	1	18	
		6	1	18	
		6	1	18	
5.4.b	3	6	1	18	84
		12	2	48	
		6	1	18	
5.4. c	2	12	2	48	96
		12	2	48	
5.4 .d	1	24	4	144	144

Tabla 5.1 la diferencia entre las diferentes arquitecturas del sistema Multi-RBFN en el número de elementos de computación.

5.3 Selección de variables de entrada para la estructura Multi-RBFN

La selección de variables de entrada (IVS) trata de reducir la dimensión del espacio de variables de entrada y crear un nuevo conjunto de variables de entrada. Este proceso de identificación y eliminación de tantos datos irrelevantes y redundantes como sean posibles, reduce la dimensionalidad de los datos y permite a los algoritmos de aprendizaje funcionar más rápido y eficazmente. Una limitación fundamental en sistemas de aproximación funcional es que, cuando aumenta el número de variables de entrada, el número de parámetros aumenta exponencialmente. Este fenómeno denominado, la maldición de la dimensionalidad, previene la utilización de la mayoría de los sistemas convencionales aproximadores y obliga a buscar soluciones más específicas.

La maldición de la dimensionalidad [BEL-61] se refiere a la aproximación exponencial del hiper-volumen como una función de dimensionalidad. Las redes de funciones de base radial pueden ser planificadas como correlaciones de un espacio de entrada a un espacio de salida. Una red de funciones de base radial tiene que cubrir o representar

cada parte de su espacio de entrada a fin de saber cómo aquella parte del espacio debería ser trazada sobre un mapa. La cubierta del espacio de entrada toma recursos y en la mayor parte de los casos generales, la cantidad de recursos necesarios es proporcional al hiper-volumen del espacio de entrada. La formulación exacta de recursos y la parte del espacio de entrada debería estar probablemente basada en los conceptos de teoría de información y geometría diferencial [BEN-00]. La maldición de la dimensionalidad provoca redes de funciones de base radial con muchas entradas irrelevantes que se comportan relativamente mal: la dimensión del espacio de entrada es alta, y la red usa casi todos sus recursos para representar partes irrelevantes del espacio. Incluso si tiene un algoritmo de aprendizaje de redes RBF que es capaz de concentrarse en partes importantes del espacio de entrada, cuánto más alta la dimensionalidad del espacio de entrada, más datos pueden ser necesarios para averiguar lo que es importante y lo que no es. IVS afecta a la severidad del problema, así como la selección del modelo de la red neuronal utilizada [JAI-82], [YU-03].

En este trabajo se propone un nuevo método de IVS para el problema de la aproximación de funciones. Este método considera un cálculo simple para seleccionar las variables más importantes. El método trata de relacionar cada dimensión los datos de entrada $\{x_1,\ldots, x_d\}$ con la salida objetivo y, (relacionar cada dimensión con la salida como una función en una dimensión) como en la siguiente expresión:

$$\{(x_1, y), (x_2, y), (x_3, y), \ldots, (x_d, y)\} \tag{5.2}$$

Después se divide los datos en cada dimensión en P partes (el número de las partes depende del número de los datos de entrada n, y cuando el número de los datos de entrada es grande, el número de partes P debe aumentarse). Esta partición se obtiene mediante la siguiente expresión:

$$\left\{P_i^{j-1} \le \left(\vec{x}^k\right)_i < P_i^j\right\} \qquad k = 1,\ldots,n; i = 1,\ldots,d;\ j = 1,\ldots,p \tag{5.3}$$

donde n es el número de datos de E/S. $\left(\vec{x}^k\right)_i$ es el componente i-ésimo del vector de entrada *k-ésimo*.

En el siguiente paso, se asocian los datos de cada dimensión en cada parte *P* a sus datos de salida objetivo correspondientes, mediante la siguiente expresión:

$$\left\{\left(\vec{x}^k\right)_i, y^k\right\} \Big/ P_i^{j-1} \le \left(\vec{x}^k\right)_i < P_i^j \tag{5.4}$$

En el siguiente paso se utiliza el filtro de *kalman* para suavizar los vectores de los máximos y mínimos en cada parte. Sección 5.5.1.1, y se calcula del valor de distancia D_i^j entre el valor máximo y mínimo de la salida objetivo en cada parte:

$$D_i^j = \max(y_j^k) - \min(y_j^k) \quad j = 1,....p \tag{5.5}$$

Para todas las partes se calcula la media de la distancia $\overline{D}$.

$$\overline{D} = \sum_{j=1}^{p} D_i^j \ / \ p \tag{5.6}$$

Cuando el promedio de la media de la distancia $\overline{D}$ tiene valor pequeño, la variable es más importante y tiene que ser seleccionada. Las variables que tienen promedio grande son variables de ruido que no afectan mucho en la salida de la función y deberían ser eliminadas. La Fig. 5.5 presenta el organigrama que tiene la descripción del método propuesto para seleccionar las variables de entrada más importantes.

5.4 Análisis del método IVS

En esta sección se analizara este método. En primer lugar, vamos a estudiar el algoritmo de IVS analizando los casos de que el método selecciona, para su descarte, variables de ruido y cuándo falla en seleccionar variables ruidosas. En segundo lugar estudiamos el efecto del filtro utilizado para suavizar los valores de los máximos y mínimos de las particiones para calcular la media de la distancia de cada variable.

Una vez se ha hecho la partición usando la ecuación (5.3) en cada dimensión de los datos de entrada $\{x_1,..., x_d\}$ con la salida objetivo *y,* se asocian los datos de cada dimensión en cada partición *P* a sus datos de salida objetivo correspondientes, ecuación (5.4). Ahora, utilizamos un filtro de *kalman* para suavizar los vectores de los máximos y mínimos en cada partición (Sección 5.5.1.1), y calculamos el valor de distancia entre el valor máximo y mínimo de la salida objetivo en cada partición, ecuación (5.5), y la

media de la distancia, ecuación (5.6). Si el promedio de la media de la distancia es menor de un valor umbral, la variable es más importante y tiene que ser seleccionada. Las variables que tienen promedio mayor del valor umbral son variables que pueden considerarse de ruido, y deberían ser eliminadas.

Para analizar este método propuesto de IVS, se presenta un ejemplo de datos multi-dimensional. Este ejemplo es una función de 6 dimensiones, con 25000 datos de E/S y con 200 particiones en cada dimensión. Este ejemplo viene dado por la siguiente expresión:

$$f(x_1, x_2, x_3, x_4, x_5, x_6) = sen(2\pi x_1 x_2) + 2x_3.\exp(-5x_4) + 0x_5 + 0.0001x_6 \qquad (5.7)$$

La descripción general del método IVS en la Fig. 5.5 presenta el funcionamiento del método sobre un ejemplo multi-dimensional. Podemos explicar los pasos del método de la siguiente forma:

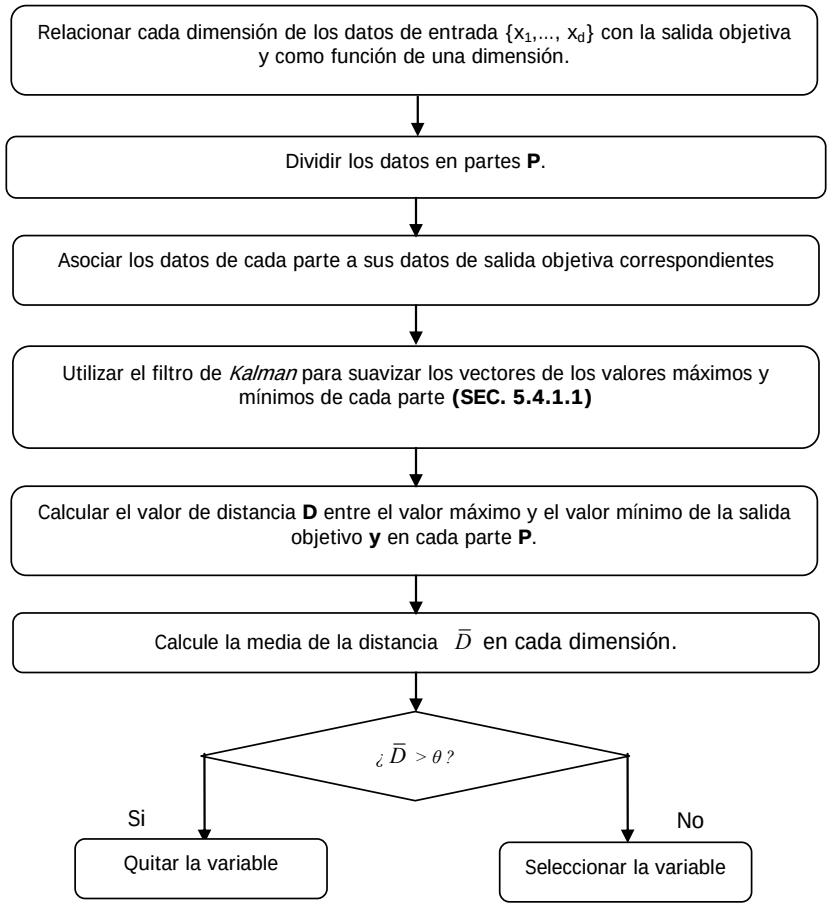


Fig. 5.5 Descripción general del método IVS.

Paso 1: Relacionar cada variable de los datos de entrada $\{x_1,\ldots, x_6\}$ con la salida objetivo *y*. Este paso relaciona cada dimensión con la salida objetivo como función de una dimensión como muestra la Figura 5.7:

$$\{(x_1, y), (x_2, y), (x_3, y), (x_4, y), (x_5, y), (x_6, y)\} \tag{5.8}$$

Paso 2: Dividir los datos de entrada *x* en cada dimensión en 200 partes. La partición se calcula usando la siguiente expresión:

$$\left\{P_i^{j-1} \le \left(\vec{x}^k\right)_i < P_i^j\right\} \qquad k = 1,\ldots,25000; i = 1,...,6;\ j = 1,...,200 \tag{5.9}$$

Paso 3: Asociar los datos de entrada en cada parte x_p a sus datos de salida objetivo correspondientes como en la expresión siguiente:

$$\left\{\left(\vec{x}^k\right)_i, y^k\right\} \Big/ P_i^{j-1} \le \left(\vec{x}^k\right)_i < P_i^j \tag{5.10}$$

Paso 4: Utilizar el filtro de *kalman* para suavizar los vectores de los máximos y mínimos en cada parte (Sección 5.5.1.1). Para mostrar el efecto del proceso de filtrar los vectores de los máximos y mínimos en cada parte, seleccionamos las variables x_1 y x_2 de la función en la ecuación (5.7) y aplicamos el proceso del filtro de *Kalman*. La Fig 5.6 muestra los valores máximos y mínimos antes y después de proceso, para cada una de esas variables.

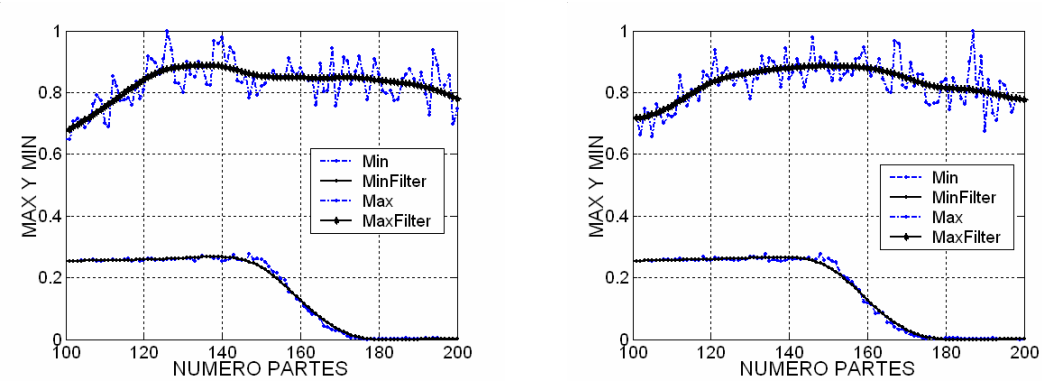


Fig. 5.6 Los máximos y mínimos de las particiones antes y después de utilizar el filtro de *Kalman* .

De la Fig. 5.6 se puede observar que los vectores de los máximos y mínimos después de utilizar el filtro de *Kalman* son más suaves y esto afecta en el cálculo de la media de la distancia y produce un valor umbral de esta distancia más estable.

Paso 5: Calcular el valor de distancia D_i^j entre el valor máximo y el valor mínimo de la los datos de salida objetivo *y*, que pertenecen a cada parte *P*. El valor de la distancia D_i^j se calcula utilizando la siguiente expresión:

$$D_i^j = \max(y_j^k) - \min(y_j^k) \quad j = 1,....200 \tag{5.11}$$

Paso 6: Calcular de la media de la distancia $\bar{D}$ en cada dimensión, este viene dada por la expresión siguiente:

$$\bar{D} = \sum_{j=1}^{200} D_i^j \ / \ 200 \tag{5.12}$$

En el ejemplo actual la distancia media antes y después de filtro de *Kalman* tiene estos valores:

Variable	X_1	X_2	X_3	X_4	X_5	X_6
Con Filtro	0.6652	0.6661	0.7748	0.7755	0.9037	0.9035
Sin Filtro	0.6821	0.6827	0.7832	0.7901	0.9157	0.9148

Paso 7: las dimensiones que tienen un promedio de las distancias $\bar{D}$ menor que el valor umbral θ serán seleccionadas para ser las variables de entrada más importantes, y las que tienen $\bar{D}$ mayor que el valor umbral θ han sido eliminadas del espacio de los datos originales.

Del ejemplo presentado en la Fig. 5.7 podemos ver por qué las variables x_1, x_2, x_3 y x_4 son las variables de entrada más importantes, ya que tienen un valor de distancia media $\bar{D}$ menor del valor umbral $\theta = 0.8$. Por eso, tienen que ser seleccionadas para ser entrada de las Sub-RBFN en el sistema jerárquico Multi-RBFN. Las variables x_5 y x_6 son variables que se pueden considerar ruido y tiene un valor de distancia mayor del valor umbral, por lo que estas variables serán eliminadas. Este método de seleccionar las variables más irrelevantes tiene un cálculo muy simple y con exactitud de selección de las variables de entrada más importantes. El problema que sufre este método es

decidir el valor del umbral θ, que puede ser diferente en diferentes problemas, pero se estabiliza en un valor fijo cuando hay suficientes números de datos.

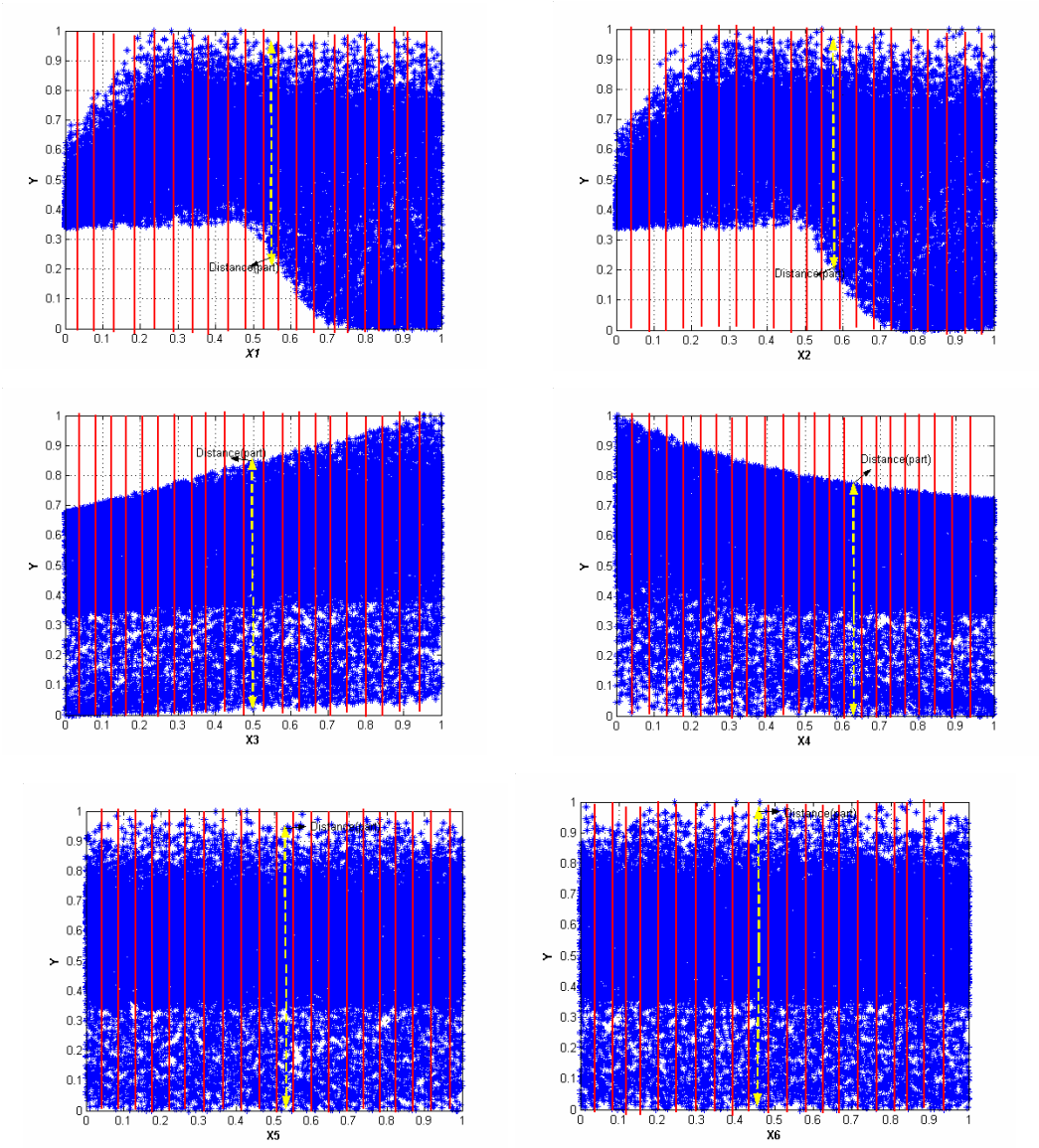


Fig. 5.7 Los salida de todos las dimensiones $\{x_1, x_2, x_3, x_4, x_5, x_6\}$ con la salida objetivo y.

5.5 Selección de las agrupaciones de variables de entrada y del número de Sub-RBFNs

El proceso de seleccionar cuáles de las variables seleccionadas deben ir solas o juntas a una Sub-RBFN es fundamental para decidir la estructura jerárquica del sistema propuesto Multi-RBFN, puesto que este proceso decide cuántas Sub-RBFN deben estar

en la estructura Multi-RBFN y cuáles de las variables seleccionadas deben ir solas o juntas como entradas de las Sub-RBFNs.

Este proceso depende de la función o del problema que se trata de aproximar. Cada sistema funcional viene dado mediante la forma de suma y/o multiplicación y/o división y/o resta entre sus variables, que utilizan funciones matemáticas como exponenciales, seno, coseno y logaritmo, etc. La estructura propuesta Multi-RBFN trata de sumar linealmente la salida de cada Sub-RBFN para tener la salida total del sistema Multi-RBFN. Por esto, las variables que vienen multiplicadas o divididas y no han sido eliminadas por el cálculo de la distancia media van juntas a una Sub-RBFN. Cualquier variable multiplicada o dividida por otra/s variables no produce un gran cambio en el valor de la varianza del variable en el intervalo de los datos, que siempre normalizaremos en el intervalo [0,1]. Las variables que vienen sumadas o restadas a otras variables y no han sido eliminadas por el cálculo de la distancia media van solas a una Sub-RBFN. Cualquier variable sumada o restada por otra/s variables produce un claro cambio en el valor de la varianza de la variable en el intervalo de los datos. En ese caso, la función de salida se podría expresar de la forma: $F(x_1,...,x_d)=F_1(x_i)+F_2(x_1,..,x_{i-1},x_{i+1},...,x_d)$, donde d es el número de variables de entrada. En ese caso, podríamos obtener una representación como la de la figura 5.8.

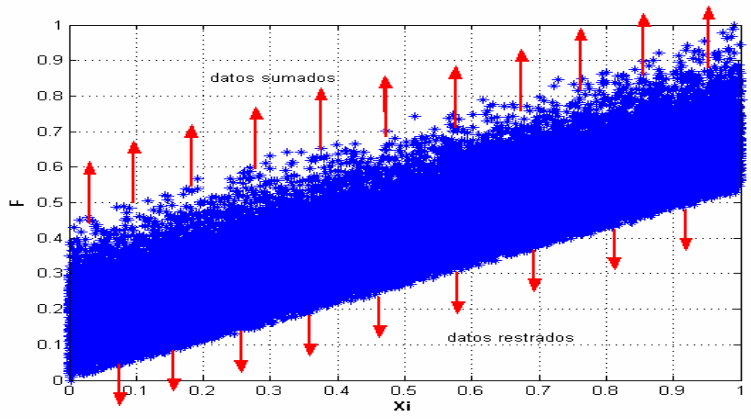


Fig. 5.8 Representación de un variable con la salida de la función

Como muestra la Fig 5.8, cuando la varianza de los datos se mantiene pequeña, la variable o los conjuntos de las variables representadas van a una misma Sub-RBFN.

Pero cuando los datos no pueden mantener la uniformidad, es decir, la varianza tiene valor grande, la variable tiene que estar acompañada por otra u otras variables. De esta forma, se aplica el cálculo de la varianza a todas las combinaciones de todas las variables de los datos de entrada, y selecciona las variables que deben ir solas o juntas a una Sub-RBFN en la estructura jerárquica multi-RBFN, según su valor de la varianza y un valor umbral de está. Para explicar el efecto de las operaciones sobre la varianza, proponemos dos ejemplos de dos funciones distintas

Ejemplo 1: una simple función de tres variables de entrada.

$$f(x) = (x_1 \cdot x_2)^2 + x_3 \tag{5.13}$$

De este ejemplo, la varianza de las variables x_1 y x_2 tiene que tener un valor cercano de forma que la varianza de la variable x_3 tiene que tener un valor de varianza diferente de las dos variables multiplicadas. Fig 5.9.*a* muestra la varianza de las variables con número de datos distintos. De esta figura podemos ver cómo las variables x_1, x_2 tienen un valor de varianza cercano y mucho más grande del valor de varianza que tiene x_3.

Ejemplo 2: una simple función de tres variables de entrada.

$$f(x) = x_1 \cdot x_2 \cdot x_3 \tag{5.14}$$

De este ejemplo la varianza de las variables x_1, x_2 y x_3 tiene que tener un valores cercanos. La Fig5.9.*b* muestra la varianza de las variables con número de datos distintos. De esta figura podemos ver como las variable x_1, x_2 y x_3 tienen valores de varianza cercanos.

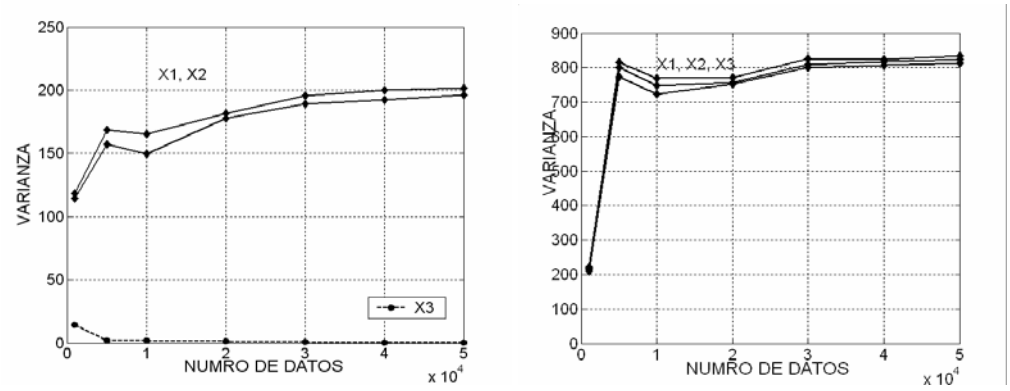


Fig. 5.9: *a)* La varianza de las variables en ejemplo 1. *b)* La varianza de las variables en ejemplo 2.

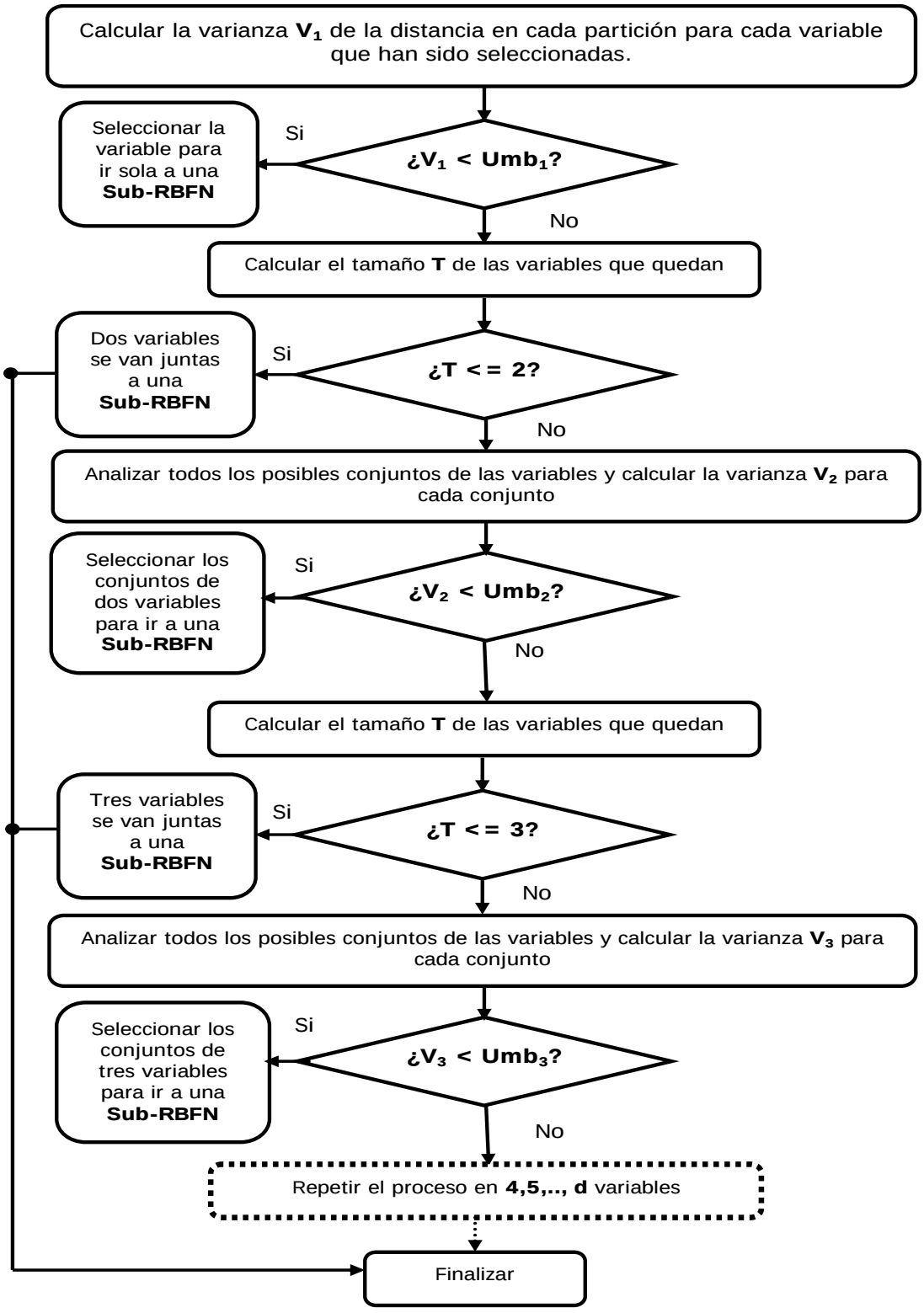


Fig. 5.10 Proceso de seleccionar las variables que van solas o juntas a una Sub-RBFN.

El proceso de seleccionar cuáles de las variables de entrada deben ir solas o juntas a una Sub-RBFN depende del valor de la varianza de la distancia entre el valor máximo y el valor mínimo en cada partición. Las variables que tienen un valor de varianza menor de un determinado valor umbral de la varianza, serán seleccionadas para dirigirse a una Sub-RBFN. La tarea de analizar los datos empieza con cada una de las variables seleccionadas con la salida objetivo, se seleccionan las variables que tengan un valor de varianza menor del propuesto valor umbral como variables que deben ir solas en una Sub-RBFN. Las variables que no han sido seleccionadas en la primera fase se analizan de forma que, se analiza todas las posibilidades de juntar estas variables, realizando cada conjunto posible de dos, tres, cuatro, etc. posibles variables con la salida objetivo. La Fig. 5.10 muestra el proceso de seleccionar cuáles de las variables de entrada van solas o juntas a una Sub-RBFN.

5.5.1 Selección de las variables que van solas a una Sub-RBFN

Este proceso depende del cálculo de la varianza de la distancia entre el punto máximo y el punto mínimo en cada partición de las dimensiones de los datos. Si la varianza tiene un valor menor del valor umbral, esta variable se selecciona como variable que vaya sola a una Sub-RBFN. Las variables que tienen un valor de varianza mayor del umbral se pasan a la siguiente fase como muestra la Fig. 5.10. Antes de hacer el cálculo de la varianza de la distancia entre el punto máximo y el punto mínimo en cada partición de las dimensiones de los datos, se utiliza un filtro para suavizar los valores máximos y mínimos. Existen muchos tipos para filtrar y suavizar los datos, en este algoritmo se utiliza un tipo de filtro denominado el filtro de *kalman* [WEL-04].

5.5.1.1 El filtro de *Kalman*

El filtro de *Kalman* es un conjunto de ecuaciones matemáticas que proveen una solución recursiva eficiente del método de mínimos cuadrados. Esta solución permite calcular un estimador lineal, insesgado y óptimo del estado de un proceso en cada momento del tiempo con base en la información disponible en el momento *t-1*, y actualizar, con la información adicional disponible en el momento *t*, dichas estimaciones. Este filtro es el principal algoritmo para estimar sistemas dinámicos especificados en la forma de estado-espacio (*State-space)* [WEL-04]. El filtro tiene su origen en el documento de

Kalman (1960) donde describe una solución recursiva para el problema del filtrado lineal de datos discretos. El filtro de *Kalman* ha sido objeto de una extensiva investigación y aplicación, particularmente en el área de la navegación autónoma y asistida, en rastreo de misiles y en economía [RAM-03].

El procedimiento de estimación completo es el siguiente: el modelo es formulado en forma de ecuaciones de estado y para un conjunto inicial de parámetros dados, los errores de predicción del modelo son generados por el filtro. Estos son utilizados para evaluar recursivamente la función de verosimilitud hasta maximizarla.

El filtro de *Kalman* tiene como objetivo resolver el problema general de estimar el estado $X \in R^n$ de un proceso controlado en tiempo discreto, el cual es dominado por una ecuación lineal en diferencia estocástica de la siguiente forma

$$X_t = A\,X_{t-1} + w_{t-1} \tag{5.15}$$

con una medida $Z \in R^m$, que es

$$Z_t = HX_t + v_t \tag{5.16}$$

Las variables aleatorias w_t y v_t representan el error del proceso y de la medida respectivamente. Se asume que son independientes entre ellas, que son ruido blanco y con distribución de probabilidad normal

$$p(w) \cong N(0, Q) \tag{5.17}$$

$$p(v) \cong N(0, R) \tag{5.18}$$

En la práctica las matrices de covarianza de la perturbación del proceso (Q) y de la perturbación de la medida (R) podrían cambiar en el tiempo, por simplicidad en general se asumen que son constantes. La matriz A se asume de una dimensión $n \times n$ y relaciona el estado en el periodo previo *t-1* con el estado en el momento *t*. La matriz H de dimensión $m \times n$ relaciona el estado con la medición Z_t. Estas matrices pueden cambiar en el tiempo, pero en general se asumen como constantes.

Las ecuaciones que se utilizan para derivar el filtro de *Kalman* se pueden dividir en dos grupos: las que actualizan el tiempo o ecuaciones de predicción y las que actualizan los datos observados o ecuaciones de actualización. Las del primer grupo son responsables de la proyección del estado al momento *t* tomando como referencia el estado en el momento *t-1* y de la actualización intermedia de la matriz de covarianza del estado. El segundo grupo de ecuaciones son responsables de la retroalimentación, es decir, incorporan nueva información dentro de la estimación anterior con lo cual se llega a una estimación mejorada del estado. Las ecuaciones que actualizan el tiempo pueden también ser pensadas como ecuaciones de pronóstico, mientras que las ecuaciones que incorporan nueva información pueden considerarse como ecuaciones de corrección como se muestra en la Fig.5.11.

Las ecuaciones específicas para la predicción (pronóstico) y la corrección del estado son detalladas de esta forma:

$$\widehat{X}_t^* = A\,\widehat{X}_{t-1} \tag{5.19}$$

$$P^*_t = A\,\widehat{P}_{t-1}\,A^t + Q \tag{5.20}$$

La matriz *A* relaciona el estado en el momento previo *t-1* con el estado al momento actual *t*, esta matriz podría cambiar para los diferentes momentos en el tiempo (*t*). *Q* representa la covarianza de la perturbación aleatoria del proceso que trata de estimar el estado.

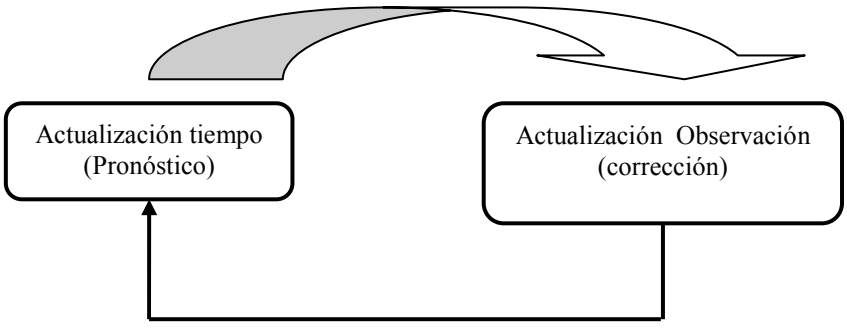


Fig. 5.11: El ciclo del filtro de *Kalman*

Las ecuaciones de corrección del filtro de *Kalman* discreto son:

$$K_t = P_t^* H^T (H P_t^* H^T + R)^{-1} \tag{5.21}$$

$$\hat{X}_t = \hat{X}_t^* + K_t (Z_t - H \hat{X}_t^*) \tag{5.22}$$

$$P_t = (I - K_t H) P_t^*) \tag{5.23}$$

La primera tarea durante la corrección de la proyección del estado es el cálculo de la ganancia de *Kalman*, K_t, (ecuación (5.21)). Este factor de ponderación o ganancia es seleccionado de tal forma que minimice la covarianza del error de la nueva estimación del estado. El siguiente paso es realmente medir el proceso para obtener Z_t y entonces generar una nueva estimación del estado que incorpora la nueva observación como en la ecuación (5.22). El paso final es obtener una nueva estimación de la covarianza del error mediante la ecuación (5.23). Después de cada par de actualizaciones, tanto del tiempo como de la medida, el proceso es repetido tomando como punto de partida las nuevas estimaciones del estado y de la covarianza del error.

Para ver el efecto del proceso de suavizar los máximos y mínimos en cada partición de los datos en la Fig. 5.12 muestra dos ejemplos con valores de máxima y mínimo antes y después del proceso del utilizar el filtro de *kalman.*

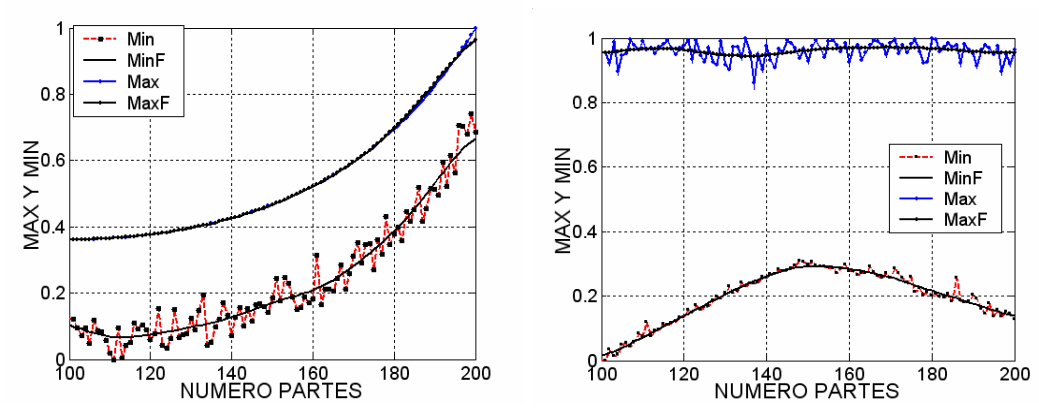


Fig. 5.12: Los máximos y mínimos de las particiones antes y después de utilizar el filtro de *Kalman*.

Se puede observar el cambio en los máximos y mínimos después de utilizar el filtro de *kalman*, que suaviza los máximos y mínimos y produce valores de distancia media y varianza mejores y más estable, lo cual ayuda mucho en estabilizar los valores umbrales de la varianza y la distancia media

5.5.1.2 Estimación del valor umbral de la varianza

Este proceso de estimar el valor umbral de la varianza para seleccionar las entradas que tienen varianza menor del valor umbral en cada fase del análisis de los datos, depende del proceso de analizar las variables en cada fase. Primero se analiza la relación de cada variable con la salida objetivo y se seleccionan las variables que tienen un valor de varianza menor de este valor umbral, como se muestra Fig. 5.10. Las variables que no han sido seleccionadas se pasan al análisis de la siguiente fase, en que se analizan todos los conjuntos posibles de dos variables y se seleccionan los conjuntos de dos variables con valor de varianza menor del valor umbral en esta fase, y de esta misma forma con el resto de las variables en fases de tres cuatro posibles conjuntos de variables. Para estimar los valores umbrales se ha hecho un estudio estadístico de muchas funciones para decidir el valor umbral de la varianza en cada fase.

5.5.1.2.1 Estimación del valor umbral de varianza analizando la relación de cada una de las variables con la salida objetivo

En esta fase se analiza la relación de cada variable de los datos de entrada con la salida objetivo, dividiéndose los datos de cada variable en partes. Se calcula la varianza de la distancia entre el valor máximo y el valor mínimo de cada parte. Después de hacer pruebas a muchas funciones, el valor umbral de la varianza en esta fase tendrá siempre un valor menor de 20, un valor de varianza normalizado como se muestra en las figuras de la 5.13 a la 5.17, con funciones de número de variables diferentes.

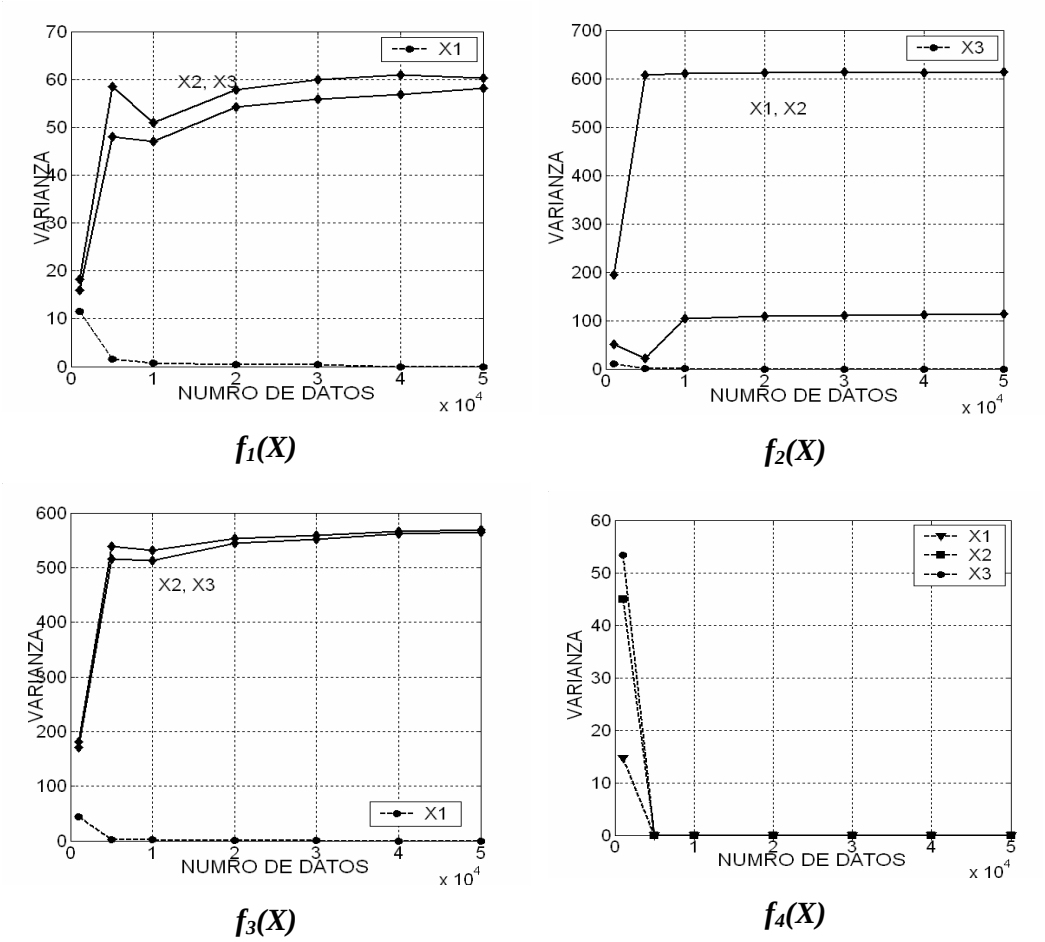
- **Prueba con funciones de 3 variables**

$$f_1(x) = cos(2\pi x_1) + 2\, e^{(-2 x_2 - x_3)}$$

$$f_2(x) = log(1+ (x_3)^2) + 2\, sen(2\pi x_1) \cdot x_2$$

$$f_3(x) = \text{sen}(2\pi\, x_1) + 10\, x_2 \cdot x_3$$

$$f_4(x) = x_3 + 2\, \text{sin}(2\pi\, x_1) + 1.5\, x_2$$



$f_1(X)$ $f_2(X)$

$f_3(X)$ $f_4(X)$

Fig. 5.13: Ejemplos de 3 variables de entrada

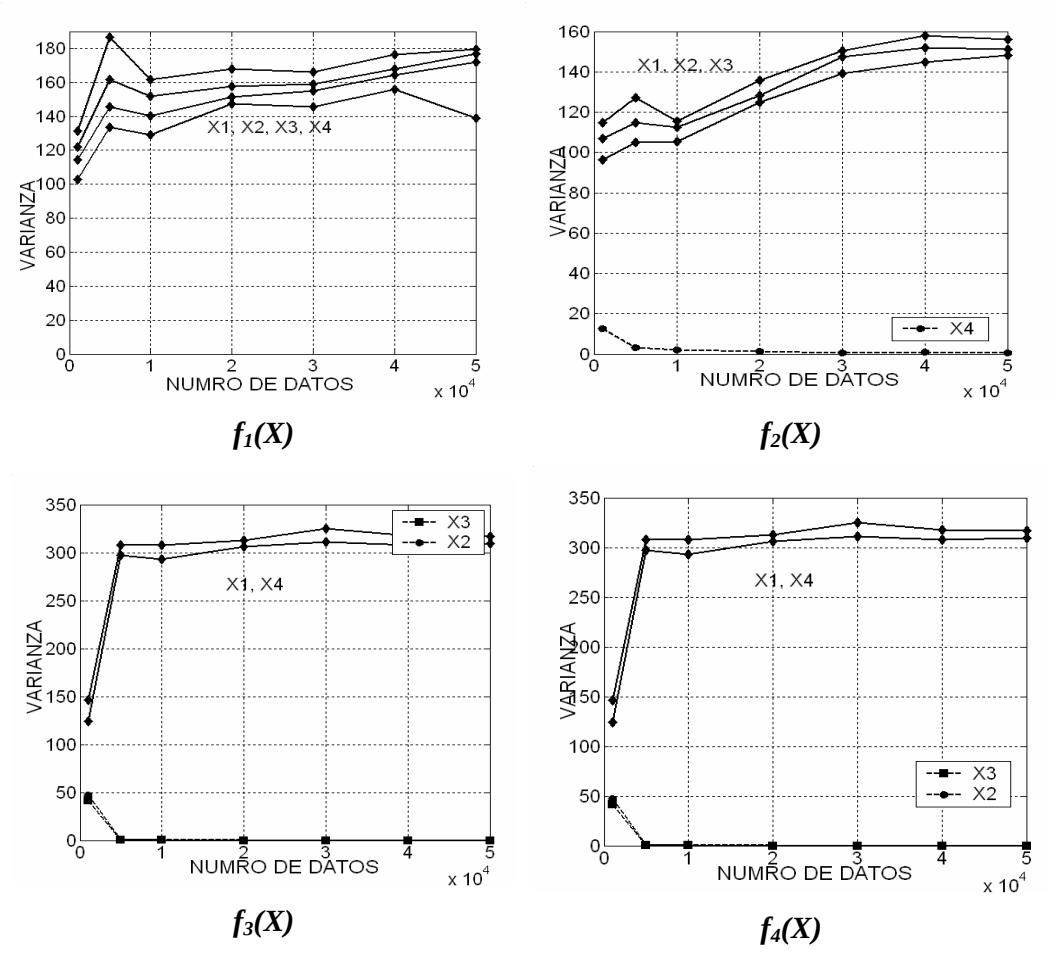
- **Prueba con funciones de 4 variables**

$$f_1(x) = (x_1 \cdot x_2)^2 + x_3 \cdot x_4$$

$$f_2(x) = x_1 \cdot x_2 \cdot x_3 + x_4$$

$$f_3(x) = x_3 + 2\,\sin(2\pi\, x_1 \cdot x_4) + 1.5\, x_2$$

$$f_4(x) = sen(2\,\pi\, x_1) + 10\, x_2 \cdot x_3 + 0\, x_4$$



$f_1(X)$ $f_2(X)$

$f_3(X)$ $f_4(X)$

Fig. 5.14: Ejemplos de 4 variables de entrada

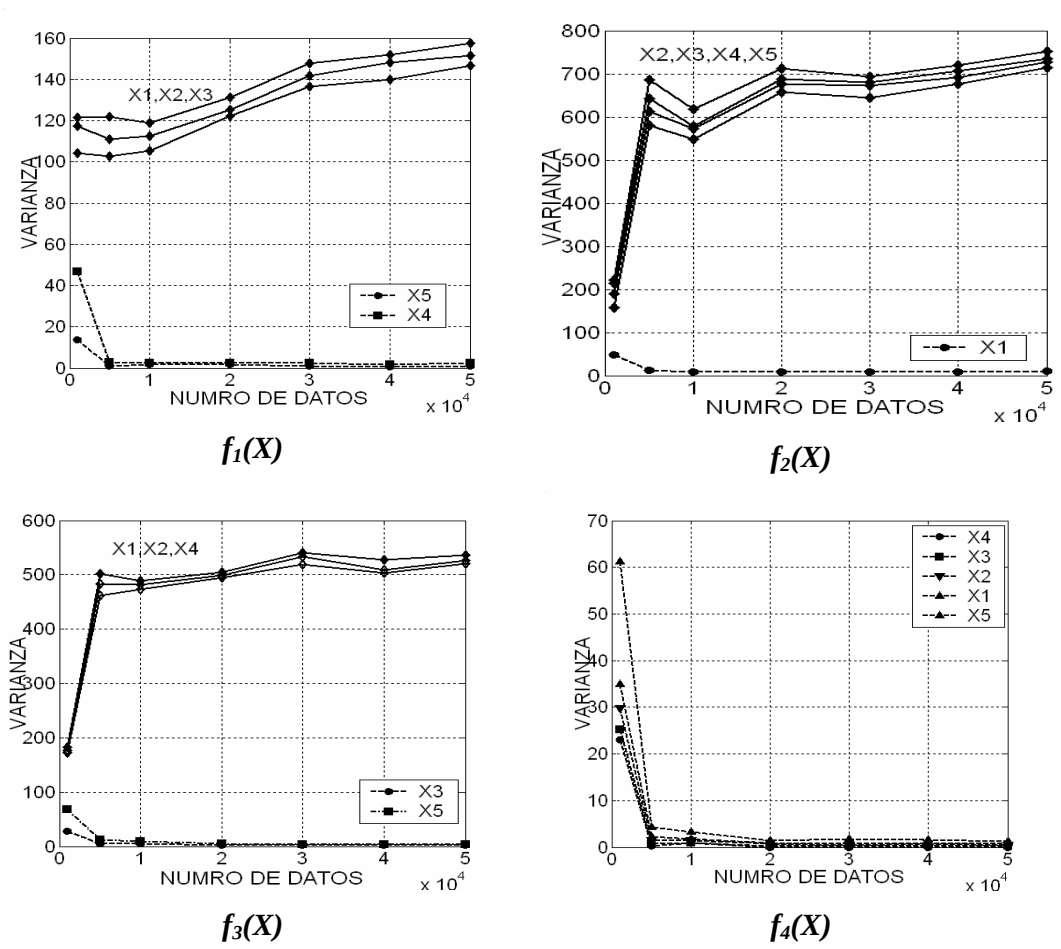
- **Prueba con funciones de 5 variables**

$$f_1(x) = sen(2\pi\, x_1 \cdot x_2) \cdot\, x_3 + e^{(-4\, x_4)}$$

$$f_2(x) = x_5 \cdot x_2 \cdot x_3 \cdot x_4$$

$$f_3(x) = sen(2\, \pi\, x_5) + 10 x_1 \cdot x_4 \cdot x_2$$

$$f_4(x) = log(0.1 + x_2) + 2\, x_4 + x_1 + sen(2\pi x_3)$$



$f_1(X)$ $f_2(X)$

$f_3(X)$ $f_4(X)$

Fig. 5.15: Ejemplos de 5 variables de entrada

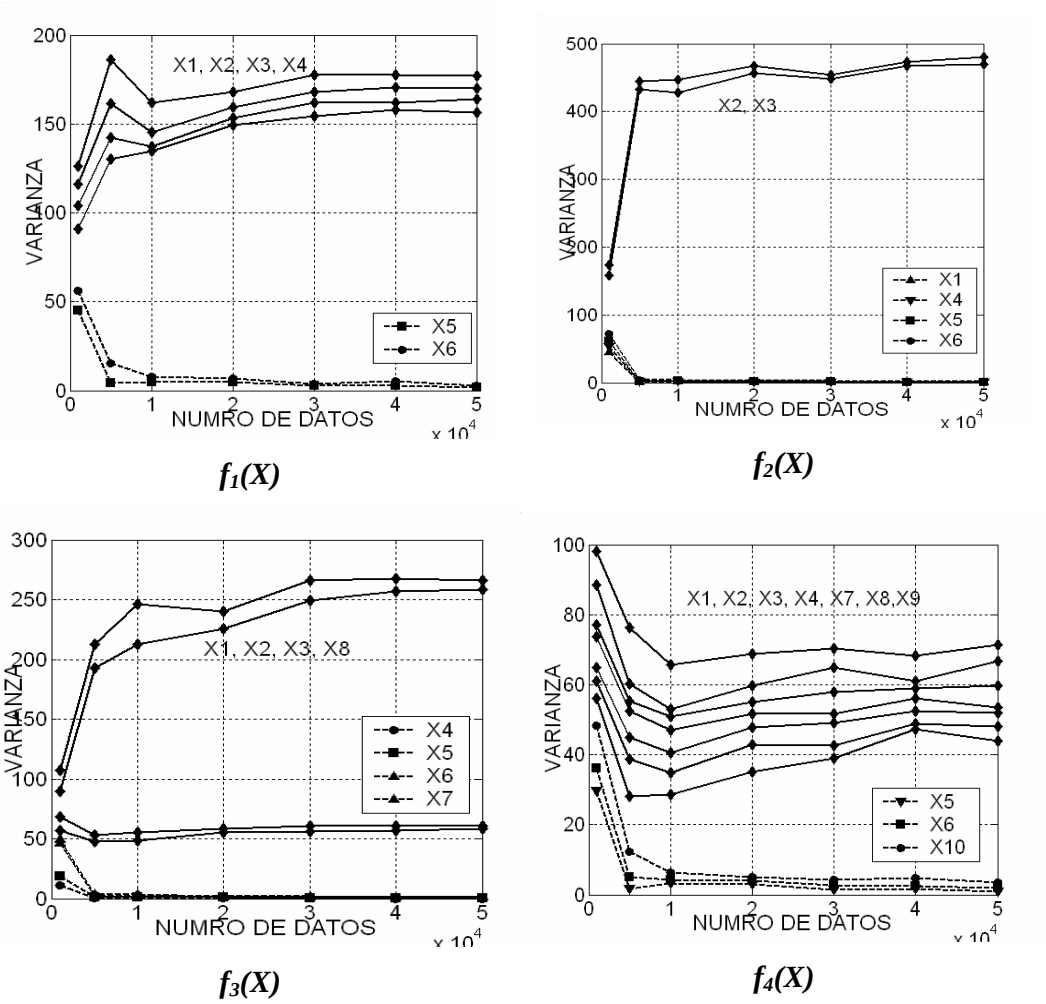
- **Prueba con funciones de 6, 8 y 10 variables**

$$f_1(x) = (x_1 \cdot x_2)^2 + x_3 \cdot x_4 + 0.001(x_5 \cdot x_6)$$

$$f_2(x) = \text{sen}(2\,\pi x_1) + 10\, x_2 \cdot x_3 + 0\, x_4 + (x_6)^2$$

$$f_3(x) = cos(2\,\pi\, x_1 \cdot x_8) + 2\, e^{(-2\, x_2 - x_3)} + 0.0001\, x_4 \cdot x_5 \cdot x_6$$

$$f_4(x) = (x_1 \cdot x_2)^2 + x_3 \cdot x_4 + 0.001\,(x_5 \cdot x_6) - x_7 \cdot x_8 \cdot x_9$$



$f_1(X)$ $f_2(X)$

$f_3(X)$ $f_4(X)$

Fig. 5.16: Ejemplos de 6,8,10 variables de entrada

5.5.1.2.2 Estimación del valor umbral para cada sub-conjunto de dos variables

Este proceso depende del cálculo de la varianza de la distancia entre el punto máximo y el punto mínimo en cada partición de los datos. La partición de los datos en esta fase de los conjuntos de dos variables viene dado de la forma de relacionar los datos de estas dos variables juntas con la salida objetivo y dividir los datos en partición bidimensional. Si la varianza es menor que el valor umbral, este conjunto de dos variables se selecciona como variables que van juntas a una Sub-RBFN. Las variables que tienen valor de varianza mayor del valor umbral en esta fase, se pasan a la siguiente fase como muestra la Fig. 5.10. Antes de hacer el cálculo de la varianza de la distancia entre el punto máximo y el punto mínimo en cada partición de los datos, se utiliza un filtro multi-dimensional para suavizar los valores máximos y mínimos como en la primera fase.

Después de hacer pruebas a muchas funciones, el valor umbral de la varianza en esta fase tendrá valor siempre menor de 60, un valor de varianza normalizado como se muestra las figuras 5.17 y 5.18 con funciones de número de variables diferentes.

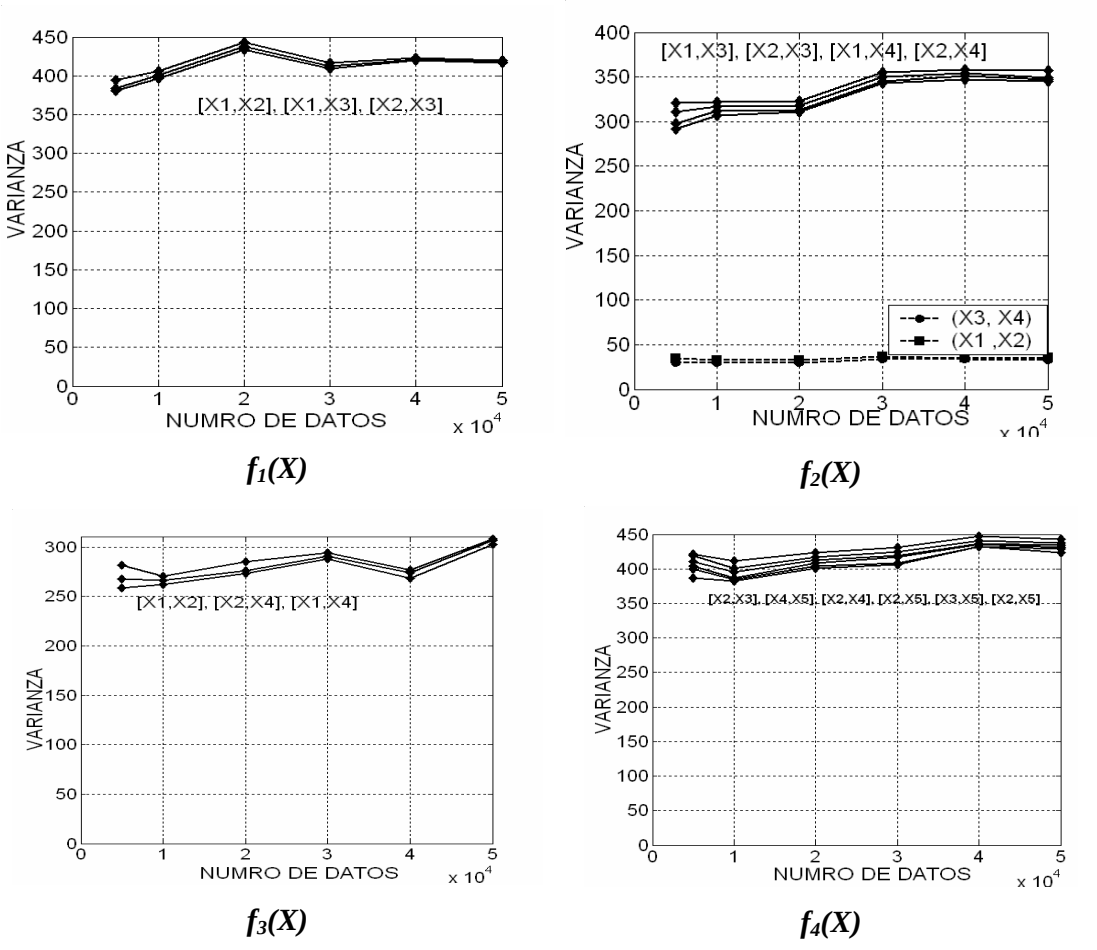
- **Prueba con funciones de 3, 4 y 5 variables**

$$f_1(x) = x_1 \cdot x_2 \cdot x_3$$

$$f_2(x) = (x_1 \cdot x_2)^2 + x_3 \cdot x_4$$

$$f_3(x) = sen(2\ \pi\ x_5) + 10\, x_1 \cdot x_4 \cdot x_2$$

$$f_4(x) = x_5 \cdot x_2 \cdot x_3 \cdot x_4$$



f1(X) ***f2(X)***

f3(X) ***f4(X)***

Fig. 5.17: Ejemplos de 3 variables de entrada

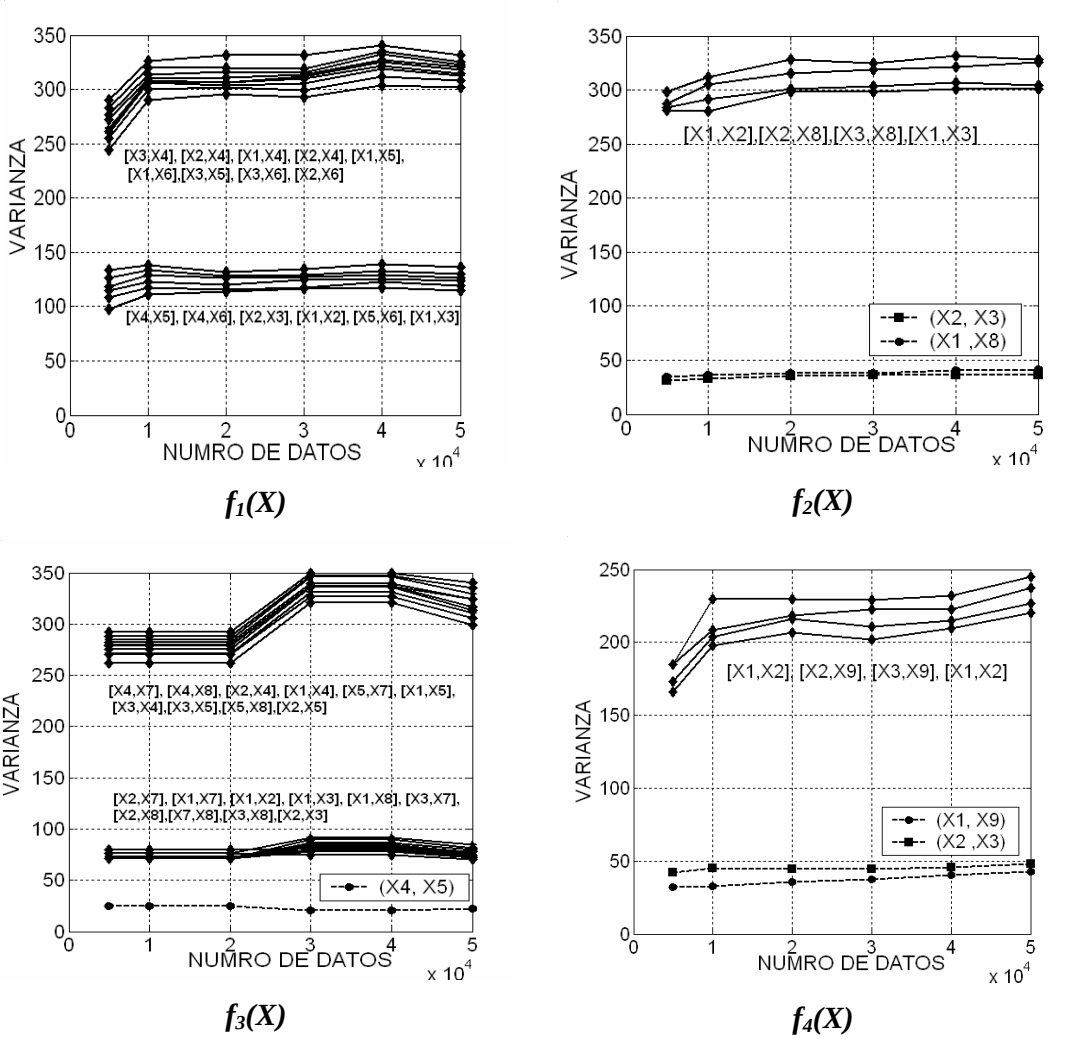
- **Prueba con funciones de 6, 8 y 10 variables**

$$f_1(x) = x_1 \cdot x_2 \cdot x_3 + x_4 \cdot x_5 \cdot x_6$$

$$f_2(x) = cos(2\pi\, x_1 \cdot x_8) + 2\, e^{(-2\, x_2 - x_3)} + 0.0001\, x_4 \cdot x_5 \cdot x_6$$

$$f_3(x) = x_1 \cdot x_2 \cdot x_3 \cdot x_8 \cdot x_7 + x_4 \cdot x_5 - 0.0001\, x_6$$

$$f_4(x) = cos(2\pi x_1 \cdot x_9) + 2\, e^{(-2\, x_2 - x_3)} + 0.0001 x_4 \cdot x_5 \cdot x_6 - x_{10}$$



f1(X) ***f2(X)***

f3(X) ***f4(X)***

Fig. 5.18: Ejemplos de 4 variables de entrada

5.5.1.2.3 Estimación del valor umbral para cada conjunto tres variables

La partición de los datos en esta fase de los conjuntos de tres variables viene dado de la forma de relacionar los datos de estas tres variables juntas con la salida objetivo y dividir los datos en partición cúbica. Si la varianza tiene un valor menor del valor umbral, este conjunto de tres variables se selecciona como tres variables que va juntas a una Sub-RBFN. Las variables que tienen valor de varianza mayor del valor umbral se pasan a la siguiente fase como muestra la Fig.5.10. Antes de hacer el cálculo de la varianza de la distancia entre el punto máximo y el punto mínimo en cada partición de los datos se utiliza un filtro multi-dimensional para suavizar los valores máximos y mínimos. Después de hacer pruebas a muchas funciones, el valor umbral de la varianza en esta fase tendrá valor siempre menor de 60, un valor de varianza normalizado como se muestra en las figuras siguientes con funciones de número de variables diferentes.

De las pruebas anteriores de analizar los datos de una variable o un conjunto de dos o tres variables podemos ver que la varianza siempre tiene un cierto valor cuando una variable o un sub-conjunto de dos o de tres variables tienen que ir a una sola Sub-RBFN. También se puede observar que esos valores umbrales para la selección de las variables que van solas o juntas a una Sub-RBFN, se aciertan cuando el número de los datos es suficiente, esto depende el número de variables utilizadas y la complejidad del problema.

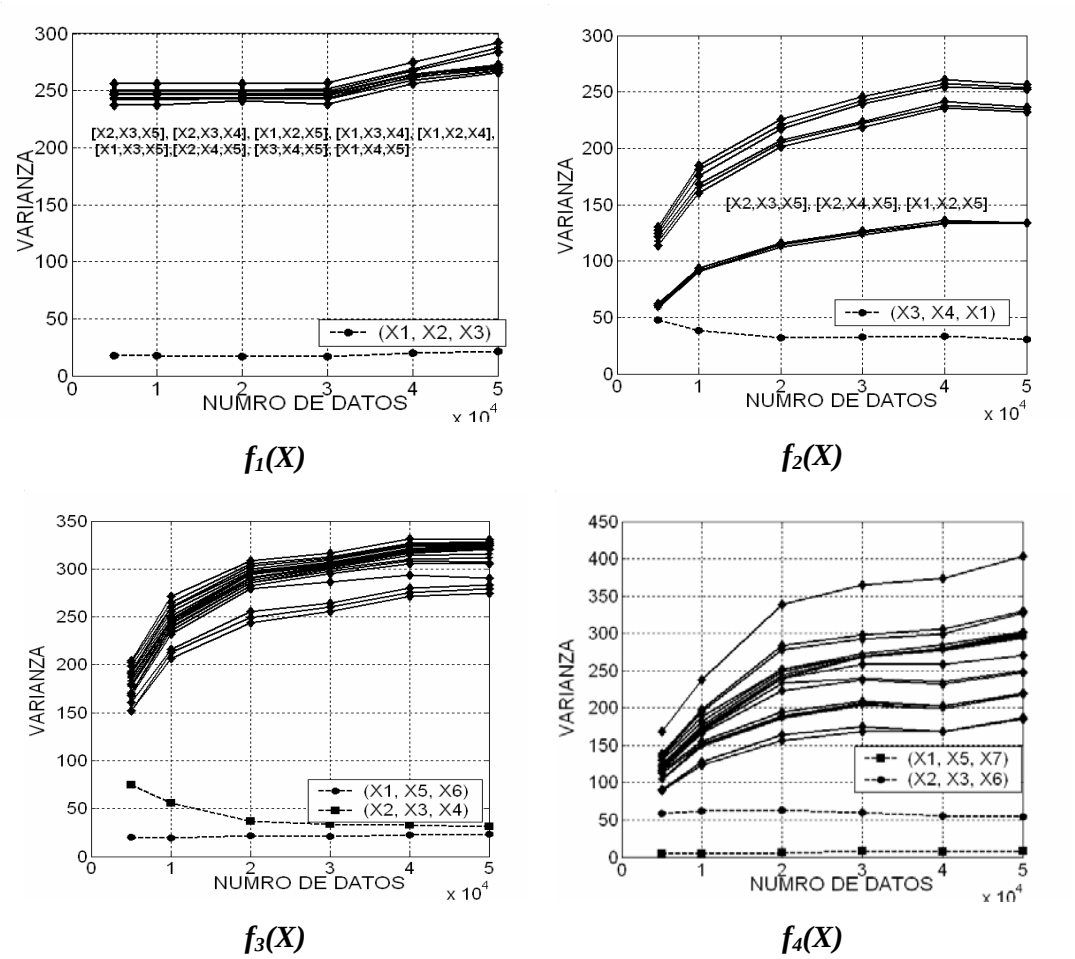
- **Prueba con funciones de 5 , 6 y 8 variables**

$$f_1(x) = 2\, sen(2\pi x_1 \cdot x_2 \cdot x_3\ ) + e^{(-4\, x_4 \cdot x_5\ )}$$

$$f_2(x)\ = x_4 \cdot x_2 + \ x_3 \cdot x_4 \cdot x_1$$

$$f_3(x)\ \ = cos\ (2\ \pi x_1 \cdot x_5 \cdot x_6) + \ 2\, e^{(-2\, x_2 - x_3) \cdot x_4}$$

$$f_4(x)\ = sen(2\ \pi\, x_1 \cdot x_5 \cdot x_7) + 10\, x_2 \cdot x_3 \cdot (x_6)^2 + 0\, x_4 \cdot x_8$$



f1(X) ***f2(X)***

f3(X) ***f4(X)***

Fig. 5.19: Ejemplos de 3 variables de entrada

Para comprobar la capacidad de estos valores umbrales proponemos una función multi-dimensional que tiene posibilidad de que un variable que va sola, dos variables que van juntas, tres variables que van juntas y cuatro variables que van juntas. Esta función artificial viene dada por la siguiente expresión:

$$f(x) = 0.0001x_8 + 2log(1+2x_4) + 2e^{(2x_3 \cdot x_6)} + 7.5\, sen(2\pi x_1 \cdot x_{10} \cdot x_7) + 20\,(x_2 \cdot x_5 \cdot x_{11} \cdot x_9) \qquad (5.24)$$

Cuando se aplica el algoritmo propuesto a esta función tenemos estos valores de varianza y del orden de las variables en la estructura Multi-RBFN:

Varianza	8.2	9.9	46.56	45.2
Variable o sub-conjunto de variables	$\{X_8\}$	$\{X_4\}$	$\{X_3,X_6\}$	$\{X_1,X_7,X_{10}\}$

El algoritmo descarta la variable $\{X_8\}$ como variable de ruido, después la variable $\{X_4\}$ será seleccionada aplicando el valor umbral de la varianza al analizar cada variable por separado (umbral < 20). Las variables $\{X_3, X_6\}$ se seleccionan por tener un valor de varianza menor del valor umbral para conjuntos de de dos variables (umbral < 60). Las variables $\{X_1, X_{10}, X_7\}$ se seleccionan por tener un valor de varianza menor del valor umbral para conjuntos de de tres variables (umbral < 60). Las variables $\{X_2, X_5, X_9, X_{11}\}$ se seleccionan para ir a una Sub-RBFN por que no han sido seleccionadas en ninguna fase. De esta forma se seleccionan todas las variables que van solas o juntas a una Sub-RBFN como muestra la Fig.5.21.

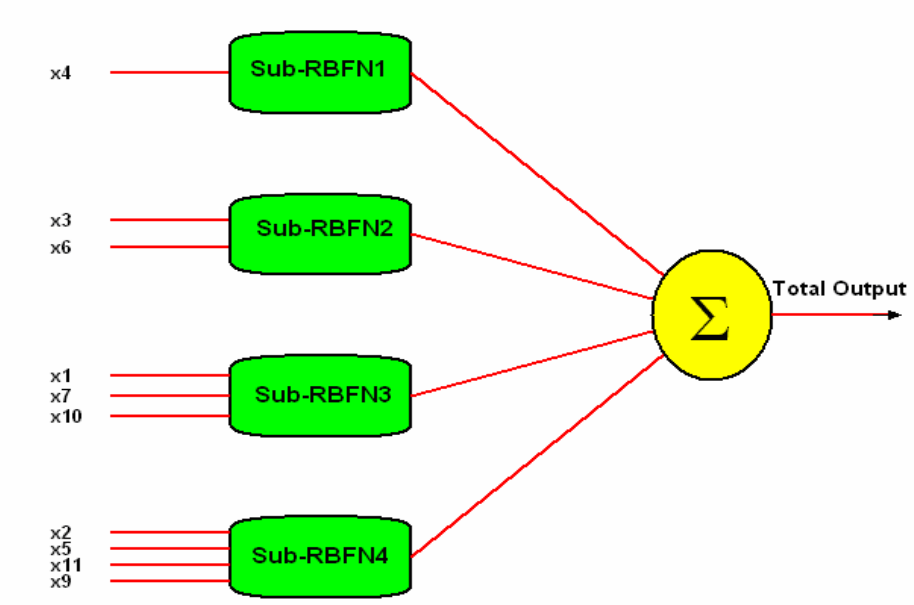


Fig. 5.20 La estructura Multi-RBFN para un ejemplo multi-dimensional dado por la ecuación (5.24)

5.5.2 Efecto del número de datos y el número de particiones

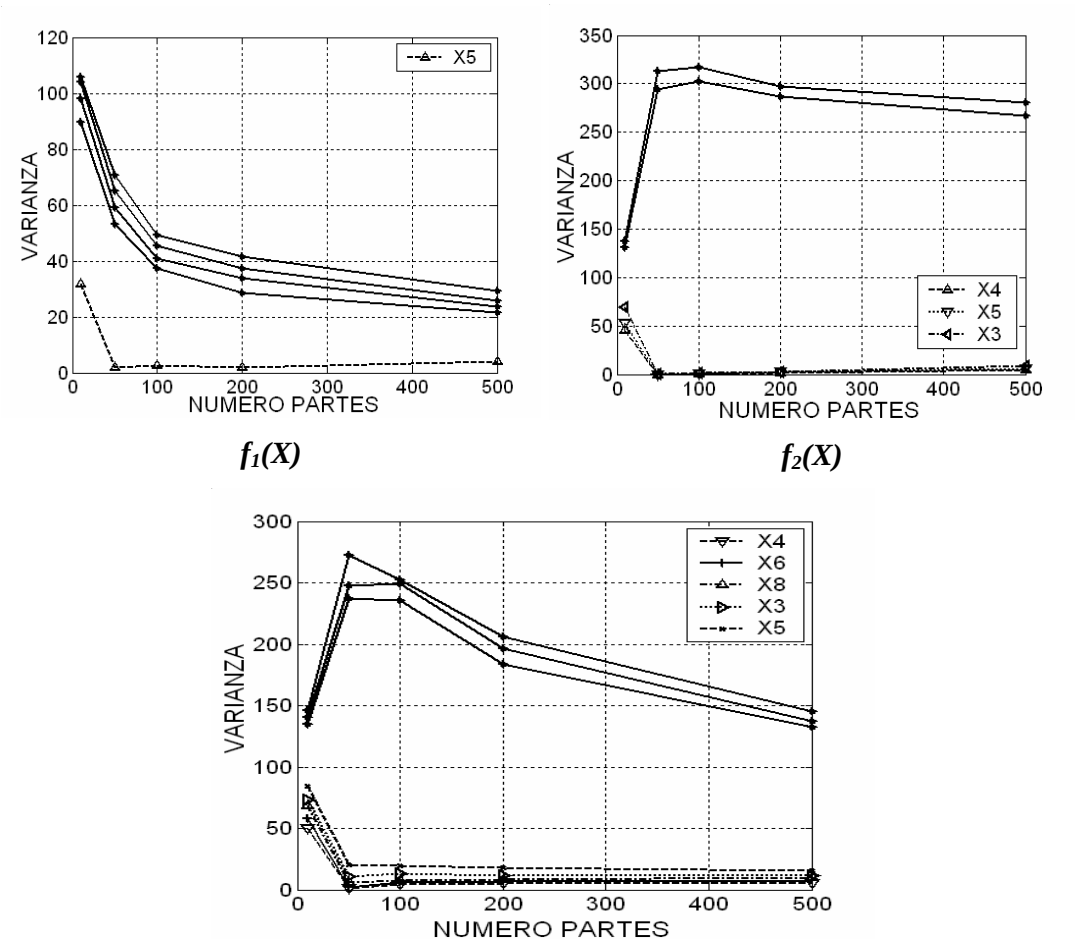
De las pruebas anteriores se puede observar que el algoritmo selecciona las variables que deben ir solas o juntas a una Sub-RBFN solo cuando hay un número de datos suficiente. Por eso, el algoritmo falla en decidir la estructura adecuada del sistema Multi-RBFN cuando el número de datos es pequeño. En el caso del número de particiones se hará pruebas a distintas funciones artificiales para ver el efecto de este parámetro. Estos funciones se generalizan con número de datos de 10000 puntos y con

particiones diferentes *{10, 50, 100, 200 ,500}*. En la 5.21 se muestra el efecto del número de particiones.

$$f_1(x) = x_1 \cdot x_2 + x_3 \cdot x_4 + x_5$$

$$f_2(x) = x_4 + 2\, sen(2\pi\, x_1 \cdot x_2) + 1.5\, x_5$$

$$f_2(x) = x_4 + 2\, sen(2\pi\, x_1 \cdot x_2 \cdot x_7) + 1.5\, x_3 + x_5$$



f1(X) ***f2(X)***

Fig. 5.21: Ejemplos de 3 variables de entrada

De las pruebas en la Fig. 5.21, se puede observar que el número de particiones tiene un efecto en el valor de la varianza. Cuando el número de particiones es adecuado con el número de datos la varianza casi tiene valor constante cuando el número de datos es suficiente. Pero cuando se aumenta el número de particiones la varianza comienza a disminuir acercando a los valores de la varianza del las variables que han sido

seleccionadas para ir solas o juntas a una Sub-RBFN. Por estas razones el número de particiones tiene que ser adecuado con el número de datos de entrada. En todo el sistema propuesto el número de particiones viene dado por la división de los datos de entrada por 100, esto cuando hay un suficiente número de datos.

5.6 Análisis de las agrupaciones de variables de entrada

En este apartado se analiza la manera de seleccionar cuáles de las variables deben ir solas o juntas a una Sub-RBFN, empezando con funciones que tienen una variable que vaya sola, después analizando la posibilidad de que haya un sub-conjunto de dos variables que vayan a una Sub-RBFN, y así sucesivamente.

5.6.1 Análisis del caso cuando una variable va sola en la estructura

La tarea de analizar los datos empieza con cada una de las variables seleccionadas con la salida objetivo, dividendo la variables en particiones y calculando el valor máximo y mínimo en cada partición. Para suavizar estos vectores de los valores máximo y mínimo se utiliza un filtro de Kalman (Sección 5.5.1.1). En este caso la función de salida se podría expresar de la forma: $F(x_1,...,x_d)=F_1(x_i)+F_2(x_1,..,x_{i-1},x_{i+1},...,x_d)$, donde d es el número de variables de entrada. Cuando se analiza estas variables con la salida objetivo (Fig. 5.22.*a*), si sumamos a cada dato de la variable que va sola un valor uniforme dentro de un determinado intervalo cerrado, la variable seguirá manteniendo su uniformidad. y su varianza será estable y pequeña (Fig.5.22.*b*), pero si cambiamos el proceso de suma por multiplicación, la variable pierde la uniformidad (Fig.5.22.*c*) y la varianza tendrá un valor grande. Esto es solo un ejemplo que pone de manifiesto que cuando la varianza es pequeña la variable debe ir sola a una Sub-RBFN.

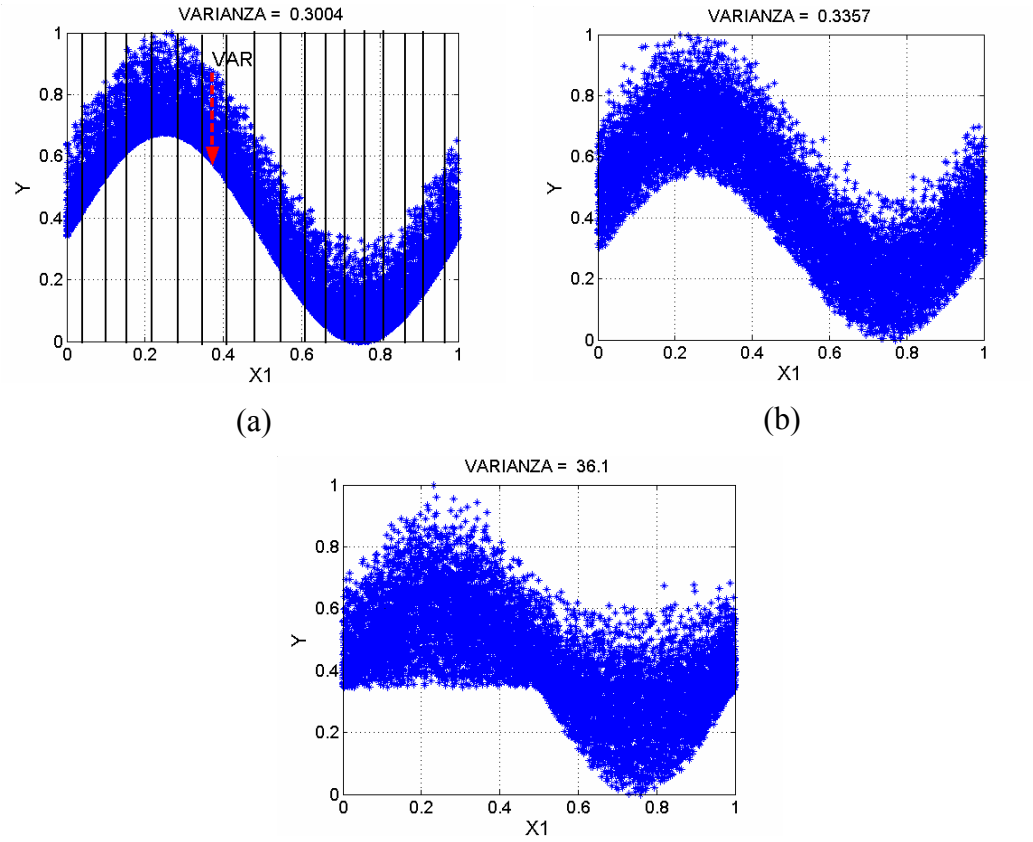


Fig. 5.22: ***a)*** La salida de un variable que va sola ***b)*** La salida de un variable va sola sumada al efecto de otra/s variables ***c)*** La salida de un variable va sola multiplicada por otra/s variables

Como muestra la Fig. 5.22, cuando la varianza de los datos se mantiene pequeña, la variable o los conjuntos de las variables representadas van a una misma Sub-RBFN. Pero cuando los datos no pueden mantener la uniformidad, es decir, la varianza tiene valor grande, la variable tiene que estar acompañada por otra u otras variables. De esta forma, el cálculo de la varianza puede usarse para decidir cuáles de las variables deben ir sola a una Sub-RBFN en la estructura jerárquica multi-RBFN.

5.6.2 Análisis del caso de un sub-conjunto de dos variables que deben ir juntas en la estructura jerárquica Multi-RBFN

La tarea ahora es analizar la relación entre los diferentes sub-conjuntos de dos variables con la salida objetivo. La partición de los datos en esta fase de los conjuntos de dos variables nos ayudará a relacionar los datos de estas dos variables juntas con la salida

objetivo y dividir los datos en partición bidimensional. Se realiza esta partición de cada sub-conjunto de dos variables con la salida objetivo, seleccionando los datos de la salida objetivo que pertenece a los datos de entrada de cada partición. De nuevo, ahora se calcula el valor máximo y mínimo en cada partición y se calcula la varianza para cada sub-conjunto de dos variables, una vez filtrado mediante un filtro bidimensional, tal y como se explica en la sub-sección 5.5.1.2.2. En este caso la función de salida se podría expresar de la forma: $F(x_1,...,x_d)=F_1(x_i,x_j)+F_2(x_1,..,x_{i-1},x_{i+1},...,x_{j-1},x_{j+1},...,x_d)$, donde d es el número de variables de entrada. Cuando se analiza estos sub-conjuntos de dos variables con la salida objetivo (Fig. 5.23.*a*), si sumamos a cada dato del sub-conjuntos de dos variables que van juntas un valor uniforme dentro de un intervalo cerrado, los sub-conjuntos siguen manteniendo la uniformidad con varianza estable (Fig. 5.23.*b*), pero si cambiamos el proceso de suma por multiplicación, los sub-conjuntos pierde la uniformidad (Fig.5.23.*c*) y la varianza tendrá un valor grande y esto justifica que cuando la varianza es pequeña los sub-conjuntos de dos variables deben ir a una Sub-RBFN.

Como muestra la Fig. 5.23, cuando la varianza de los datos se mantiene pequeña, el sub-conjunto formado por las dos variables representadas debe ir a una misma Sub-RBFN. Pero cuando los datos no pueden mantener la uniformidad, es decir, la varianza tiene valor grande (Fig. 5.23.*c*), dicho sub-conjunto de variables tiene que estar acompañado por otra u otras variables. De esta forma, la varianza decide cuáles de los sub-conjuntos de las dos variables debe ir a una Sub-RBFN en la estructura jerárquica multi-RBFN.

Igualmente, esto es extrapolable para el caso de agrupaciones de tres o más variables. La partición de los datos en la fase de los sub-conjuntos de tres variables se realiza para relacionar los datos de estas tres variables juntas con la salida objetivo y dividir los datos en partición cúbica. En este caso la función de salida se podría expresar de la forma: $F(x_1,...,x_d)=F_1(x_i,x_j,x_k)+F_2(x_1,..,x_m,...,x_d)$ con $\mathrm{m}\neq\{i,j,k\}$, donde d es el número de variables de entrada. De la misma forma se analiza las sub-conjuntos de cuatro variables.

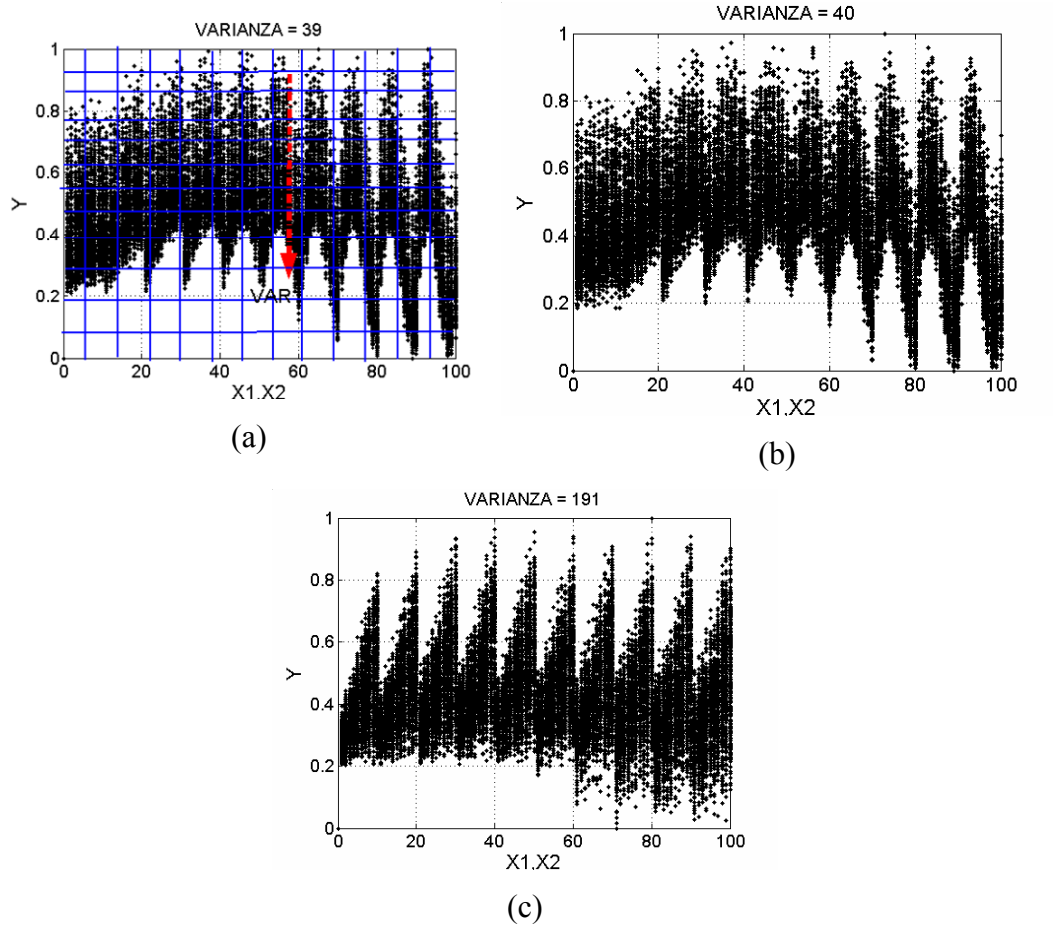


Fig. 5.23: ***a)*** La salida de un sub-conjunto de dos variables que van juntas ***b)*** La salida de un sub-conjunto de dos variables sumada al efecto de otra/s variables ***c)*** La salida de sub-conjunto de dos variables multiplicada por otra/s variables

5.7 Optimización de los parámetros de cada Sub-RBFN (Centros *c*, Radios *r*, Pesos *w*)

Esta sección introduce los algoritmos utilizados para optimizar los parámetros de cada Sub-RBFN en la estructura Multi-RBFN, a fin de encontrar el mínimo global de la función de coste, para un conjunto de datos de E/S. Los algoritmos de aprendizaje deben llevar a cabo los pasos siguientes: la selección del espacio de entrada de los datos, la selección de un punto de partida en el espacio (inicialización) y la búsqueda del mínimo local a partir de dicho punto de partida, y que esperamos que sea el mínimo global. Una RBFN viene completamente especificada mediante los parámetros siguientes: el número *m* de funciones de base radiales, los valores de los centros $\vec{c}$ y de

los radios r, y el conjunto de los pesos de salida w. La inicialización de los parámetros de cada Sub-RBFN en la estructura Multi-RBFN es el tercer paso de nuestro algoritmo en general, después del proceso de la selección de las variables de entrada más importantes y el proceso elegir cuáles de estas variables van solas o juntas a una Sub-RBFN para construir la arquitectura del sistema Multi-RBFN. En esta etapa se realiza una primera determinación de los valores de los parámetros que caracterizan una función base de una Sub-RBFN que es una RBFN típica. Es decir para cada RBF en cada Sub-RBFN, hay que optimizar los valores de centros $\vec{c}^{S}$, radios r^{S} y pesos w^{S}.

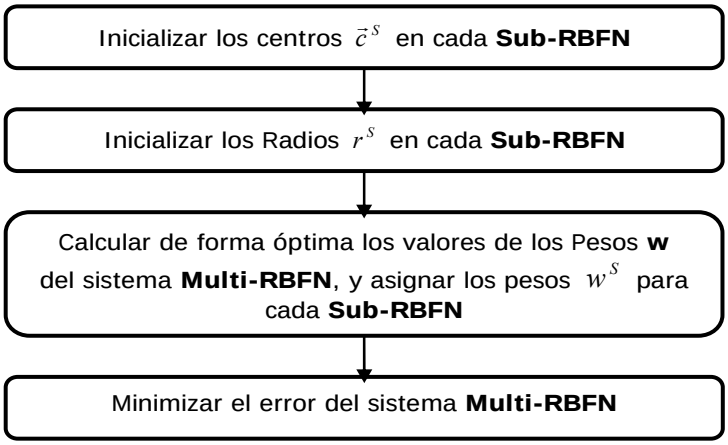


Fig. 5.24 El proceso de optimizar los parámetros de sistema Multi-RBFN

El objetivo es optimizar estos parámetros del sistema Multi-RBFN para aproximar modelos complejos a partir de un conjunto de muestras de E/S. Los centros $\vec{c}^{S}$ y radios r^{S}, que definen la ubicación y forma de las funciones base de la red, influyen de forma no lineal en la salida del sistema Multi-RBFN, por lo que debe utilizar un algoritmo de minimización que los ajusta de forma iterativa hasta encontrar el mínimo local. Los centros $\vec{c}^{S}$ y radios r^{S} de cada Sub-RBFN en el sistema Multi-RBFN se optimizan de forma aislada, es decir, en cada Sub-RBFN estos parámetros se optimizan de forma independiente del resto de parámetros en otros Sub-RBFN. Una vez que se ha fijado un valor para estos parámetros, los pesos w^{S} se optimizan en todo el sistema Multi-RBFN, es decir, los pesos de todas las Sub-RBFN se optimizan juntos dependiendo de la salida total *f(x)* y la matriz de activación Φ_j^s para todo el sistema. Estos pesos se optimizan

usando un algoritmo de resolución de funciones lineales que es, en nuestro caso, el método SVD. La Fig. 5.24 presenta la descripción del proceso de optimizar los parámetros del sistema Multi-RBFN.

5.7.1 Inicialización de los centros $\vec{c}^S$ en cada Sub-RBFN

La técnica de asignar una función radial a cada muestra de los datos de entrada es muy simple, pero produce redes grandes que son ineficaces y sensibles al sobre-ajuste (*over-fitting*), y presenta mal funcionamiento. Una solución parcial a estos problemas es agrupar muestras similares en clusters. A cada cluster corresponde un centroide, el centroide entonces, elegido como el centro de una función de base radial. Cada uno de estos clusters tiene que estar situado en una zona del espacio de entrada en donde haya datos de E/S que lo activen de forma que contribuya a la salida de la red. Una función base que no se activa aumentará la complejidad de la red.

En el algoritmo propuesto en esta memoria utilizamos un nuevo método supervisado de clustering para inicializar los valores de los centros $\vec{c}^S$ en cada Sub-RBFN. Este algoritmo presentado en el Capítulo 3 incorpora la información referente a la salida objetivo para cada vector de entrada del conjunto de entrenamiento, y calcula el error provocado por cada cluster usando una RBFN. El número de clusters aumentará en zonas donde el cluster provoca un error mayor dependiendo del proceso de migración de los clusters que tienen menor error a zonas de clusters que tienen mayor error y un proceso de desplazamiento local que trata de pertenecer los datos al cluster más cercano [AWA-05a]. Para más información sobre este método de clustering, véase el Capítulo 3.

5.7.2 Inicialización los radios r^S en cada Sub-RBFN

Una vez determinados los centros de cada Sub-RBFN, el siguiente paso es fijar los valores de los radios r^S de cada función base, de forma que no se quede ningún punto del espacio de entrada sin cubrir, es decir, de manera que el solapamiento de las zonas de activación de una función base a otra sea lo más ligero posible, para suavizar así la aproximación.

En este algoritmo propuesto utilizamos un algoritmo heurístico de los k vecinos más cercanos (*Knn*). Esta heurística fija el radio de acción de cada función base a un valor

igual a la distancia media de los centros de sus funciones base más cercanos. Utilizamos un solo vecino más cercano, es decir *k=1*.

5.7.3 Cálculo óptimo de los pesos w^S en cada Sub-RBFN

Una vez que los valores de los centros $\vec{c}^S$ y radios r^S de la función de base radial han sido optimizados mediante los métodos anteriores, cada Sub-RBFN pasa a ser un modelo lineal y el conjunto de los pesos w^S depende linealmente de las muestras del conjunto de entrenamiento. En el sistema Multi-RBFN, los pesos *w* del sistema se optimizan dependiendo de la salida total del sistema. El cálculo de dicha salida total *f(x)* es la suma lineal de todas las salidas de las Sub-RBFN *{F_1(x),...,F_S(x)},* como muestra la ecuación (5.1). Este cálculo es muy importante en el sistema Multi-RBFN, porque el cálculo de los pesos óptimos w^S en cada Sub-RBFN depende de la salida total del sistema Multi-RBFN. Así, se calcula la matriz de activación Φ^S para todo el sistema Multi-RBFN depende en esto de los datos de entrada y los valores de los centros $\vec{c}^S$ y radios r^S en todas las Sub-RBFN. La aplicación de un sistema de ecuaciones lineales (que utilizamos en nuestro algoritmo la descomposición en valores singulares 'SVD') para optimizar los pesos *w*, será por la matriz de Φ para todo el sistema Multi-RBFN, y después viene el paso de asignar cada conjunto de pesos w^S para cada Sub-RBFN. Una vez que han sido asignados los pesos w^S a cada Sub-RBFN, se calcula la salida de cada Sub-RBFN y la suma de éstas que es la salida total del sistema Multi-RBFN. Se calculan los pesos w^S entre la capa oculta y la capa de la salida en cada Sub-RBFN. En este caso, el objetivo es minimizar el error entre la salida total del sistema Multi-RBFN y la salida deseada. Una vez que los valores de los centros $\vec{c}^S$ y radios r^S para cada Sub-RBFN han sido fijados mediante el proceso de inicialización, cada Sub-RBFN, será lineal, y la salida total *f(x)* que es la suma de todas las salidas de las Sub-RBFN será lineal y los pesos *w* dependerán del conjunto de entrenamiento. El proceso de aprendizaje está guiado por la minimización de una función del error calculado de esta forma:

$$Er_n^d = \frac{1}{2}\sum_{j=1}^{d}\sum_{i=1}^{n}\left(f(\vec{x}_i^d, \Phi, w) - y_i\right)^2 \tag{5.25}$$

donde $f(\vec{x}_i^d, \Phi, w)$ es la salida total *f(x)* del sistema jerárquico Multi-RBFN, y y_i es la salida deseada.

El objetivo que persigue esta fase es encontrar los pesos óptimos para calcular la salida total y calcular el error. Utilizando la notación matricial, la solución al problema de minimizar el error cuadrático consistirá en encontrar el conjunto óptimo de pesos que hace esto posible se describe el problema mediante la siguiente expresión:

$$\begin{bmatrix} Y_1 \\ Y_2 \\ \vdots \\ Y_n \end{bmatrix} = \begin{bmatrix} w_1^s \\ w_2^s \\ \vdots \\ w_m^s \end{bmatrix} \begin{bmatrix} \varphi_{1\,1}^s & \cdots & \varphi_{1\,m}^s \\ \varphi_{2\,1}^s & \cdots & \varphi_{1\,m}^s \\ \vdots & & \vdots \\ \varphi_{n\,1}^s & \cdots & \varphi_{n\,m}^s \end{bmatrix} \tag{5.26}$$

reduciendo la ecuación (5.26), se obtiene la expresión:

$$Y = w_m^s \ \varphi_m^s \tag{5.27}$$

donde Y es la salida deseada del sistema Multi-RBFN, w_m^s es la matriz de pesos para todas las Sub-RBFN y φ_m^s es la matriz de activación para todas las Sub-RBFN. Para calcular la matriz de los pesos w_m^s se utiliza la siguiente expresión:

$$w_m^s = G \ Y \tag{5.28}$$

donde G es la matriz pseudo-inversa de la matriz de activación φ_m^s. Esta matriz puede ser calculada mediante métodos de resolución de ecuaciones lineales. En este algoritmo utilizamos la descomposición en valores singulares (SVD) para resolver este sistema de ecuaciones lineales del sistema y después asignar los pesos w^S para cada Sub-RBFN para calcular la salida para cada una de éstas.

5.8 Optimización del número de funciones radiales RBF en cada Sub-RBFN

El número de funciones radiales es un parámetro crítico y, según algunos métodos, puede ser fijado a priori o determinado incremental o decrementalmente. De hecho tanto las dimensiones del espacio de parámetros y el tamaño de la familia de aproximadores

dependen del valor de las funciones base que, generalmente, se determinan por prueba y error [VIÑ-03], variando el número de neuronas hasta que se pueda conseguir una red capaz de resolver el problema dado. En los últimos años, ha surgido un gran interés por desarrollar métodos que pudieran determinar el número de neuronas ocultas de forma automática. Otro tipo de metodologías tratan de desarrollar algoritmos incrementales, es decir, algoritmos que comienzan con una neurona oculta y van incrementando nuevas neuronas ocultas.

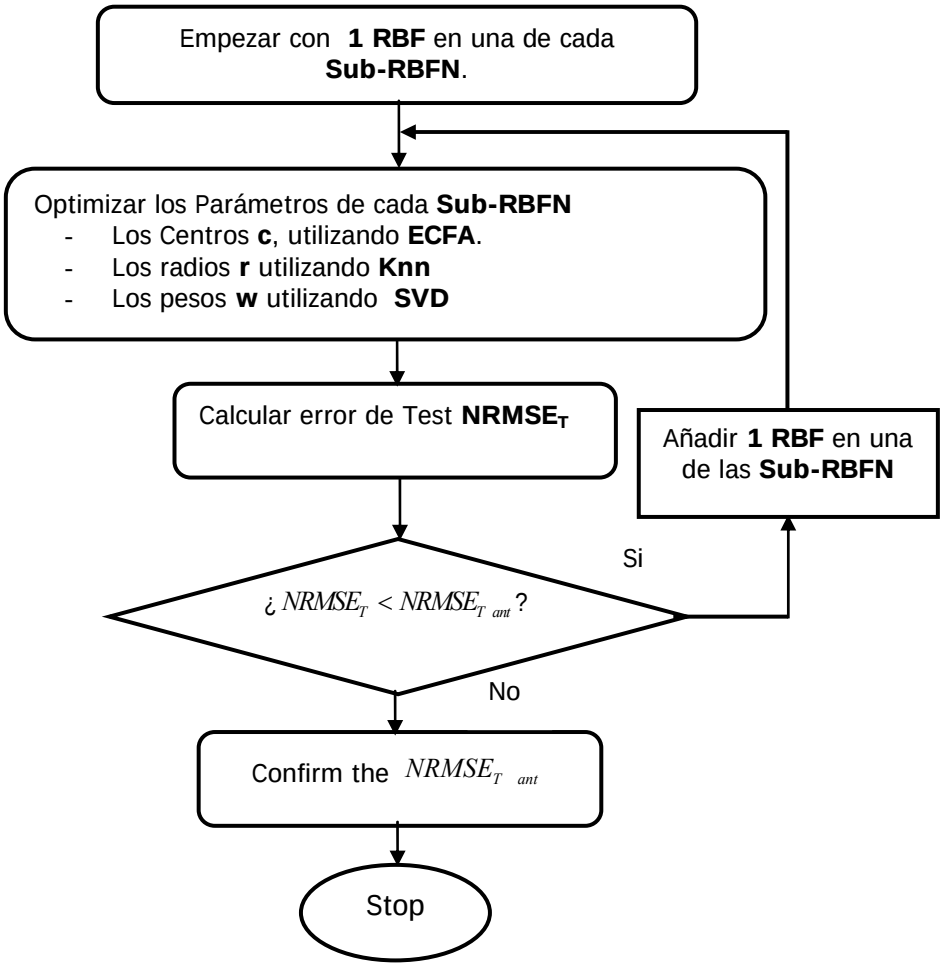


Fig. 5.25 Descripción de la forma de optimizar los parámetros y el número de RBF en el sistema Multi-RBFN

En el sistema jerárquico Multi-RBFN propuesto utilizamos el método incremental para determinar el número de funciones radiales. La manera de utilizar este método está relacionada con el error que produce el sistema Multi-RBFN, es decir, con la salida total

Y_T. La forma de incrementar números RBF empieza con un número bajo de RBF en todas las Sub-RBFN y empieza incrementar cada vez sólo 1 RBF en una de las Sub-RBFN hasta que el error de validación no se mejore en varias iteraciones. Esto finaliza el proceso y produce la arquitectura final con el número óptimo de RBF en cada Sub-RBFN y por tanto en el sistema Multi-RBFN. La Fig. 5.25 presenta la forma de optimizar los parámetros de cada Sub-RBFN y encontrar el número adecuado de funciones radiales en cada Sub-RBFN.

5.9 Minimización conjunta de de la estructura Multi-RBFN

Como resultado de las etapas del algoritmo propuesto para la determinación de la estructura Multi-RBFN, se obtiene la arquitectura del sistema Multi-RBFN que depende de las variables de entrada seleccionadas y cuáles de estas van solas o juntas en una Sub-RBFN y los valores de los parámetros (centros $\vec{c}^{\,S}$, radios r^S y pesos w^S) en cada Sub-RBFN. Estos parámetros iniciales no son más que una configuración inicial adecuada que nos debe conducir al óptimo global, pero todavía está a una cierta distancia de alcanzarlo. En esta etapa se aplica un algoritmo de optimización local para acercarse a dicho óptimo.

El problema de la optimización local y los métodos más utilizados en este campo se han descrito en el Capítulo 2. Este tipo de algoritmos pueden acabar atrapados en un mínimo local, según la configuración de la que parten. Los algoritmos más utilizados para el problema de optimización local son *Steepest Descent* y *Levenberg-Marquardt*. El primer método es el más simple y robusto de los algoritmos de optimización local, pero necesita un mayor número de iteraciones para converger y eso provoca mayor coste computacional. Más expresamente el método de *Levenberg-Marquardt* incorpora información sobre la segunda derivada del error para acelerar la búsqueda cuando la configuración de la red se encuentra cerca del mínimo. Por tanto, será el método utilizado en esta memoria.

5.10 Conclusiones

Este capítulo ha presentado un algoritmo capaz de encontrar arquitecturas para modelar sistemas complejos de aproximación funcional sin que el aumento del número de variables de entrada tenga que suponer un aumento exponencial de la complejidad del sistema aproximador como mostrarán los resultados experimentales en el capítulo siguiente. La estructura jerárquica (Multi-RBFN) presentada consiste en redes de funciones de base radial (Sub-RBFN), con la propiedad de que cada Sub-RBFN se puede encargar un conjunto de variables de entrada y no todas.

El modelo presentado tiene algunos componentes básicos:

- La utilización de un nuevo método para seleccionar las variables de entrada (IVS) más importantes que trata de reducir la dimensión del espacio de variables de entrada y la creación de un nuevo conjunto de variables de entrada.

- La elección de la estructura jerárquica Multi-RBFN adecuada según el número de variables de entrada seccionadas y cuáles de estas variables van solas o juntas en una Sub-RBFN.

- La optimización de los parámetros del sistema Multi-RBFN, para sistemas complejos de aproximación de funciones a partir de un conjunto finito de datos de E/S y la optimización de estos parámetros (número *m* de funciones de base radiales, los centros $\vec{c}_j^s$ y el radio r^s y los pesos w_j^s).

    - El número de funciones radiales se optimiza mediante un algoritmo incremental.

    - Los centros de las funciones radiales se optimizan inicialmente mediante la utilización de un algoritmo nuevo de clustering para el problema de la aproximación de funciones presentado en el Capítulo 3.

- Los radios de las funciones radiales se optimizan inicialmente mediante la utilización de un algoritmo heurístico de los *k* vecinos más cercanos con valor de *k=1*.

- Los pesos de las funciones radiales se calculan mediante un sistema de resolución de ecuaciones lineales. En este algoritmo se utiliza la descomposición en valores singulares (SVD).

El objetivo es encontrar una arquitectura jerárquica del sistema Multi-RBFN adecuado para aproximar un conjunto de vectores de E/S, con las variables de entrada más importantes que han sido seleccionadas, y optimizar los parámetros de la estructura Multi-RBFN para sistemas de aproximación de función de un conjunto de ejemplos de entrada /salida (E/S).

CAPÍTULO 6

RESULTADOS, EVALUACIÓN Y COMPARACIONES DEL SISTEMA MULTI-RBFN

En este capítulo se presentarán ejemplos que muestran la validez del procedimiento presentado en el capítulo anterior para la aproximación de una serie de funciones artificiales a partir de datos de E/S. Este conjunto de funciones comprende funciones bidimensionales de dos, tres, cuatro, cinco, seis y ocho dimensiones. Se evaluará la robustez del algoritmo frente a añadir un cierto nivel de ruido.

Se presenta en este capítulo dos tipos de resultados: la estructura del sistema Multi-RBFN seleccionado por el algoritmo presentado en el capítulo anterior utilizando el método de IVS y cuáles de las variables de entrada deben ir solas o juntas en una Sub-RBFN en el sistema Multi-RBFN, y los resultados de la validez de algoritmo en aproximar funciones a partir de muestras de datos de E/S, comparadas con resultados de un RBFN típica que recibe todas las variables de la función.

6.1 Introducción

En el capítulo anterior se presentó una nueva metodología para aproximar funciones complejas a partir de muestras de E/S. A modo de resumen, dicha metodología constaba en una serie de pasos:

1. Selección de las variables de entrada más importantes de los datos de entrada originales (sección 5.3).

2. Determinación de cuáles de las variables seleccionadas deben ir solas o juntas a una Sub-RBFN para construir el sistema Multi-RBFN (sección 5.5).

3. Optimización de los parámetros del sistema Multi-RBFN para la aproximación de funciones a partir de un conjunto de datos de E/S (sección 5.7).

Los primeros pasos deciden la estructura del sistema Multi-RBFN (cuántas variables de entrada se seleccionan, cuáles de estas variables van solas o juntas a una Sub- RBFN) y el tercer paso optimiza los parámetros de la estructura decidida por los dos primeros pasos para la aproximación de funciones a partir de conjunto de muestras de E/S.

La mayoría de los ejemplos utilizados en este capítulo son funciones artificiales que presentan diversas propiedades para averiguar la validez del algoritmo propuesto en el capítulo anterior. Los resultados de aplicar el algoritmo a diferentes funciones son de dos tipos: la estructura adecuada Multi-RBFN y el error de aproximación. Se analiza también la robustez del algoritmo del mismo frente al número de datos y la presencia de ruido en los mismos.

El preproceso de los datos se centra en seleccionar las variables de entrada más importantes usando nuestro método de IVS, que decide también cuáles de las variables seleccionadas van solas o juntas a una Sub-RBFN [AWA-05b]. El método de IVS funciona correctamente cuando hay un número suficiente de datos que explorar todo el posible dominio de entrada.

En todas las funciones utilizadas se usan datos de entrada/salida. Este número de datos de divide en dos partes próximamente iguales: datos de entrenamiento y datos de

validación. Para el proceso de inicialización de los centros se utiliza un nuevo algoritmo de clustering para aproximación de funciones ECFA presentado en el Capítulo 3 [AWA-05a]. La inicialización de los radios se obtiene mediante el método de los vecinos más cercanos (*knn*) con valor de $k = 1$. La inicialización de los valores de los centros y radios se realiza en cada Sub-RBFN de forma aislada del resto de las Sub-RBFN. El cálculo exacto de los pesos depende de la utilización de SVD para resolver el sistema de ecuaciones lineales resultante. El cálculo de los pesos depende de todas las Sub-RBFN en el sistema Multi-RBFN. El número de funciones radiales utilizadas en cada Sub-RBFN se obtiene por un método incremental, que comienza con un valor inicial pequeño de RBF en todas las Sub-RBFN y añade 1 RBF a una de las Sub-RBFN calculando el error de validación en cada iteración y elige la estructura del número RBF de menor error de validación. El proceso para de añadir RBF cuando no se mejora el error de validación o cuando llegara a un valor umbral de éste.

Los resultados obtenidos por el sistema Multi-RBFN se comparan con resultados de una RBFN clásica que recibe todas las entradas de la función utilizada. De este modo, se evaluará el sistema Multi-RBFN con sus características en decrecer el número de parámetros utilizados y converger de forma más rápida con menor número de elementos, lo cual consigue el principal objetivo en el presente trabajo de la búsqueda de nuevas arquitecturas de cómputo capaces de modelar sistemas complejos de aproximación de funciones, sin que el aumento del número de variables de entrada tenga que suponer un incremento exponencial en la complejidad del sistema.

6.2 Funciones de dos variables

En este apartado se evalúa el algoritmo propuesto utilizando funciones de dos dimensiones. Estas funciones artificiales muestran la capacidad del algoritmo de seleccionar la estructura adecuada del sistema Multi-RBFN y aproximar la función de forma más eficaz. Los conjuntos de entrenamiento y validación que se han utilizado para estas funciones, están formados por 2600 puntos distribuidos en rejilla de 51× 51 celdas en el dominio de entrada. El conjunto de validación esta formado por la mitad del conjunto de los datos.

- Función $f_1(x)$

$$f_1(x) = ((x_1-2)\cdot(2\,x_1+1))\,/(1+(x_1)^2) + ((x_2-2)\cdot(2\,x_2+1))\,/(1+(x_2)^2) \quad x_1, x_2 \in [-5,5] \quad (6.1)$$

El algoritmo propuesto selecciona la arquitectura del sistema Multi-RBFN para la función $f_1(x)$ utilizando el método de IVS que decide también cuáles de estas variables seleccionadas deben ir solas o juntas depende al valor umbral de la varianza como se muestra la Fig.6.1. En la función $f_1(x)$ cada una de las variables debe ir sola a una Sub-RBFN como se muestra la Fig.6.2.*a* Cada nodo en la capa oculta presenta una Sub-RBFN con el número de funciones radiales utilizado. La Fig. 6.2.*b* presenta una RBFN clásica que recibe todas las entradas de la función.

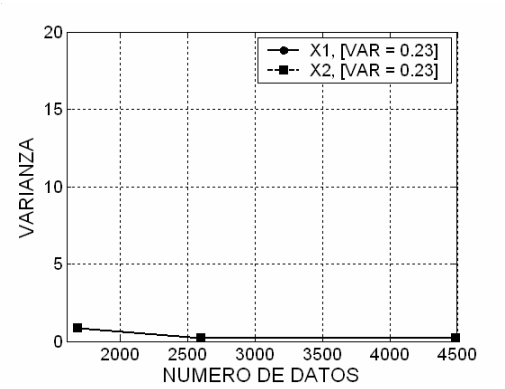
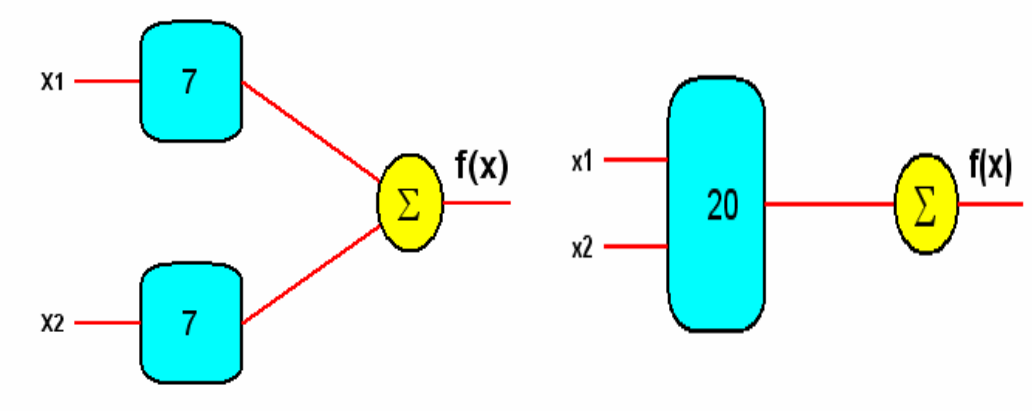


Fig. 6.1: La varianza para cada una de las variables



(a) (b)

Fig. 6.2 ***a)*** Estructura Multi-RBFN seleccionada por el algoritmo. ***b)*** Estructura de una RBFN clásica para la función actual

Una vez aplicado el algoritmo propuesto de IVS, el proceso continúa buscando el número adecuado de funciones radiales en cada Sub-RBFN, optimizándose los parámetros del sistema para cada posible configuración.

Tabla 6.1 muestra los resultados de 5 ejecuciones; el conjunto de funciones radiales utilizadas en las Sub-RBFN {**RBF**} que va considerando el algoritmo (cada vez se añade 1 RBF), **#Parám** es el numero de parámetros computacionales. En la tabla también aparece la media del error cuadrático medio normalizado NRMSE de entrenamiento (**$NRMSE_{Tr}$**) y la media del error cuadrático medio normalizado NRMSE de test (**$NRMSE_{Test}$**). En cada iteración se añade 1 RBF y se calcula el error de validación, y la estructura que produce un valor mínimo de error de validación se selecciona, tal y como se describió en el algoritmo de la Fig. 5.25.

Algoritmo Multi-RBFN						**RBFN Clásica**			
{ RBF }	**# Parám**	**$NRMSE_{Tr}$**	**DesV**	**$NRMSE_{Test}$**	**DesV**	**RBF**	**# Parám**	**$NRMSE_{Tr}$**	**$NRMSE_{Test}$**
{2 1}	12	0.661	6E-3	0.660	4E-3	2	8	0.798	0.798
{1 2}	12	0.657	9E-2	0.662	5E-3	3	12	0.616	0.619
{3 1}	16	0.513	2E-3	0.511	1E-2	4	16	0.552	0.563
{2 2}	16	0.517	9E-3	0.513	1E-3	5	20	0.487	0.489
{4 1}	20	0.480	9E-3	0.483	1 E-3	6	24	0.423	0.418
{3 2}	20	0.484	4E-3	0.482	3E-3	7	28	0.440	0.433
{4 2}	24	0.441	2E-3	0.442	8E-3	8	32	0.393	0.374
{3 3}	24	0.403	2E-3	0.403	2E-3	9	36	0.390	0.371
{4 3}	28	0.355	7E-3	0.354	9E-3	10	40	0.367	0.349
{3 4}	28	0.353	2E-2	0.357	1E-2	11	44	0.329	0.320
{5 3}	32	0.295	1E-2	0.296	9E-3	12	48	0.319	0.306
{4 4}	32	0.306	1E-2	0.319	1E-2	13	52	0.294	0.284
{6 3}	36	0.264	2E-2	0.275	1E-2	14	56	0.277	0.267
{5 4}	36	0.261	2E-2	0.254	6E-3	15	60	0.274	0.265
{6 4}	40	0.207	3E-2	0.231	9E-3	16	64	0.253	0.248
{5 5}	40	0.219	3E-2	0.204	3E-2	17	68	0.245	0.241
{6 5}	44	0.182	7E-3	0.187	9E-3	18	72	0.217	0.220
{5 6}	44	0.184	9E-3	0.191	4E-3	19	76	0.229	0.227
{7 5}	48	0.134	1E-2	0.137	1E-2	20	80	0.203	0.207
{6 6}	48	0.134	2E-2	0.137	2E-2				
{7 6}	52	0.107	1E-2	0.116	9E-3				
{6 7}	52	0.107	1E-2	0.112	9E-3				
{7 7}	56	0.092	7E-3	0.087	6E-3				

Tabla 6.1: NRMSE de entrenamiento y test obtenido por el algoritmo propuesto y por RBFN clásica para la función $f_1(x)$

De los resultados de la Tabla 6.1, se puede observar la capacidad del sistema Multi-RBFN en converger con número de RBF menor del una sistema clásica de una RBFN..

Las funciones $f_2(x)$ y $f_3(x)$ producen las mismas características que la función $f_1(x)$. Estas funciones se presentan a continuación.

- Función $f_2(x)$

$$f_2(x) = 1.3356(1.5(1-x_1)\,.\,e^{(2x_1-1)} \cdot sen(3\pi(x_1-0.6)^2) + e^{(3(x_2-0.5))} \cdot sen(4\pi(x_2-0.9)^2)),\ x_1, x_2 \in [0,1] \quad (6.2)$$

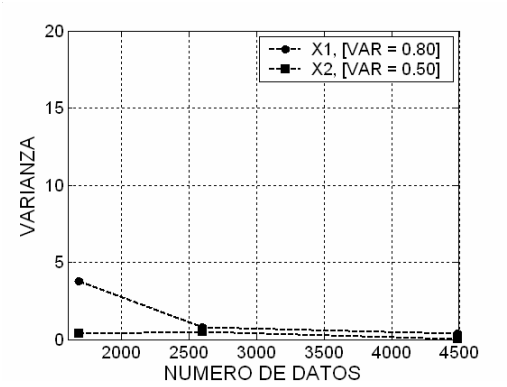
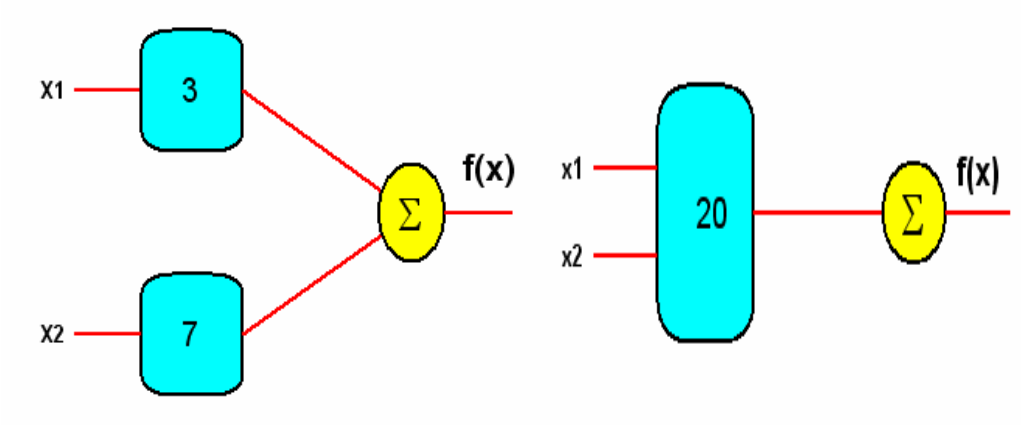


Fig. 6.3 La varianza para cada una de las variables



(a) (b)

Fig. 6.4 ***a)*** Estructura Multi-RBFN seleccionada por el algoritmo. ***b)*** Estructura de una RBFN clásica para la función actual

Algoritmo Multi-RBFN						RBFN Clásica			
{ RBF}	# Parám	$NRMSE_{Tr}$	DesV	$NRMSE_{Test}$	DesV	RBF	# Parám	$NRMSE_{Tr}$	$NRMSE_{Test}$
{2 1}	12	0.800	9E-3	0.808	1E-2	2	8	0.809	0.834
{1 2}	12	0.750	5E-2	0.762	4E-2	3	12	0.796	0.825
{2 2}	16	0.739	7E-2	0.749	5E-2	4	16	0.637	0.668
{1 3}	16	0.671	8E-2	0.687	9E-2	5	20	0.529	0.555
{2 3}	20	0.616	9E-2	0.625	9E-2	6	24	0.597	0.623
{1 4}	20	0.476	8E-2	0.489	1E-2	7	28	0.596	0.618
{2 4}	24	0.432	5E-2	0.446	6E-3	8	32	0.456	0.490
{1 5}	24	0.354	2E-2	0.365	6E-3	9	36	0.329	0.369
{2 5}	28	0.386	1E-2	0.362	2E-2	10	40	0.480	0.519
{1 6}	28	0.349	2E-2	0.352	4E-3	11	44	0.323	0.356
{2 6}	32	0.357	1E-2	0.337	2E-2	12	48	0.222	0.271
{1 7}	32	0.349	1E-2	0.356	1E-2	13	52	0.251	0.303
{3 6}	36	0.175	5E-2	0.180	1E-2	14	56	0.179	0.234
{2 7}	36	0.115	1E-2	0.128	2E-3	15	60	0.191	0.243
{4 6}	40	0.037	2E-3	0.044	2E-3	16	64	0.213	0.264
{3 7}	40	0.034	1E-3	0.040	3E-3	17	68	0.151	0.212
						18	72	0.136	0.198
						19	76	0.122	0.191
						20	80	0.128	0.193

Tabla 6.2: NRMSE de entrenamiento y test obtenido por el algoritmo propuesto y por RBFN clásica para la función $f_2(x)$

- Función $f_3(x)$

$$f_3(x) = 1.9\,(1.35 + e^{x_1}.\,sen(13\cdot(x_1\text{-}0.6)^2) + e^{(-x_2)}\cdot sen(7\,x_2)) \quad x_1, x_2 \in [0,1] \tag{6.3}$$

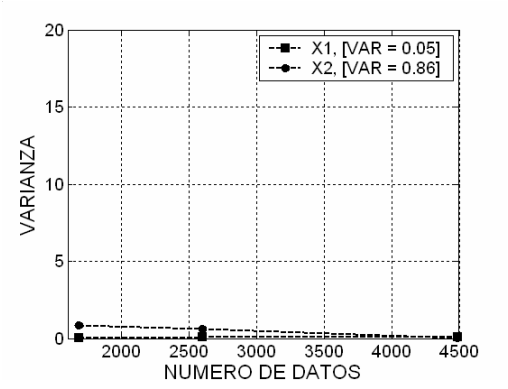
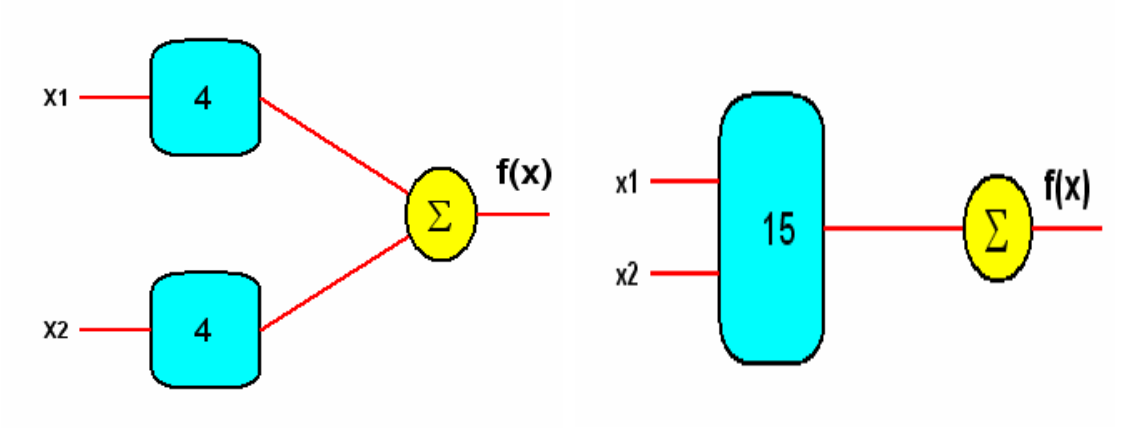


Fig. 6.5 La varianza para cada una de las variables



(a) (b)

Fig. 6.6 ***a)*** Estructura Multi-RBFN seleccionada por el algoritmo. ***b)*** Estructura de una RBFN clásica para la función actual

Algoritmo Multi-RBFN						RBFN Clásica			
{ RBF }	**# Parám**	**$NRMSE_{Tr}$**	**DesV**	**$NRMSE_{Test}$**	**DesV**	**RBF**	**# Parám**	**$NRMSE_{Tr}$**	**$NRMSE_{Test}$**
{2 1}	12	0.546	9E-3	0.543	9E-3	2	8	0.685	0.618
{1 2}	12	0.602	9E-3	0.603	1E-3	3	12	0.508	0.511
{3 1}	16	0.338	2E-3	0.347	1E-3	4	16	0.466	0.468
{2 2}	16	0.302	3E-3	0.319	7E-3	5	20	0.333	0.338
{3 2}	20	0.161	6E-3	0.167	5E-2	6	24	0.367	0.364
{2 3}	20	0.433	1E-2	0.442	1E-2	7	28	0.158	0.154
{4 2}	24	0.248	6E-2	0.261	5E-2	8	32	0.105	0.108
{3 3}	24	0.102	4E-3	0.105	2E-3	9	36	0.107	0.109
{4 3}	28	0.096	1E-3	0.101	1E-3	10	40	0.106	0.109
{3 4}	28	0.101	3E-3	0.104	0.0	11	44	0.103	0.105
{5 3}	32	0.075	1E-3	0.081	1E-3	12	48	0.100	0.105
{4 4}	32	0.074	1E-3	0.079	1E-3	13	52	0.101	0.106
						14	56	0.099	0.102
						15	60	0.091	0.097

Tabla 6.3: NRMSE de entrenamiento y test obtenido por el algoritmo propuesto y por RBFN clásica para la función $f_3(x)$

6.3 Funciones de tres variables

En este apartado se evalúa el algoritmo propuesto utilizando distintas funciones de tres variables. Estas funciones artificiales muestran la capacidad del algoritmo de seleccionar la estructura adecuada del sistema Multi-RBFN. Los conjuntos de entrenamiento y validación que se han utilizado para estas funciones, están formados

por 5000 puntos de forma aleatoria en el dominio de entrada. El conjunto de validación esta formado por la mitad del conjunto de los datos.

- Función $f_4(x)$

$$f_4(x) = sen(2\sqrt{(x_1 \cdot x_1 + x_2 \cdot x_2)}) + (e^{(x_3 \cdot x_3)} + 0.0001) \quad x_1, x_2 \in [0,1] \tag{6.4}$$

El algoritmo propuesto selecciona la arquitectura óptima del sistema Multi-RBFN para la función $f_4(x)$ depende al valor umbral de la varianza como se muestra Fig. 6.14 . En la función $f_4(x)$ cada una de las variables debe ir sola a una Sub-RBFN como se muestra la Fig.6.16.*a*.

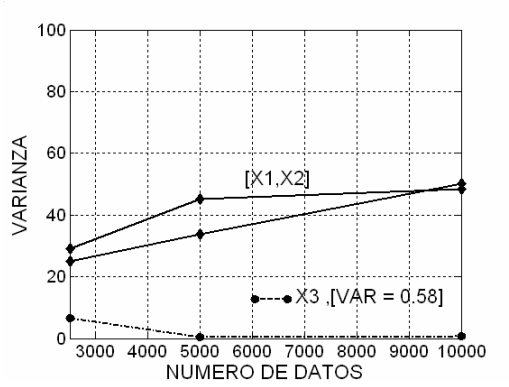
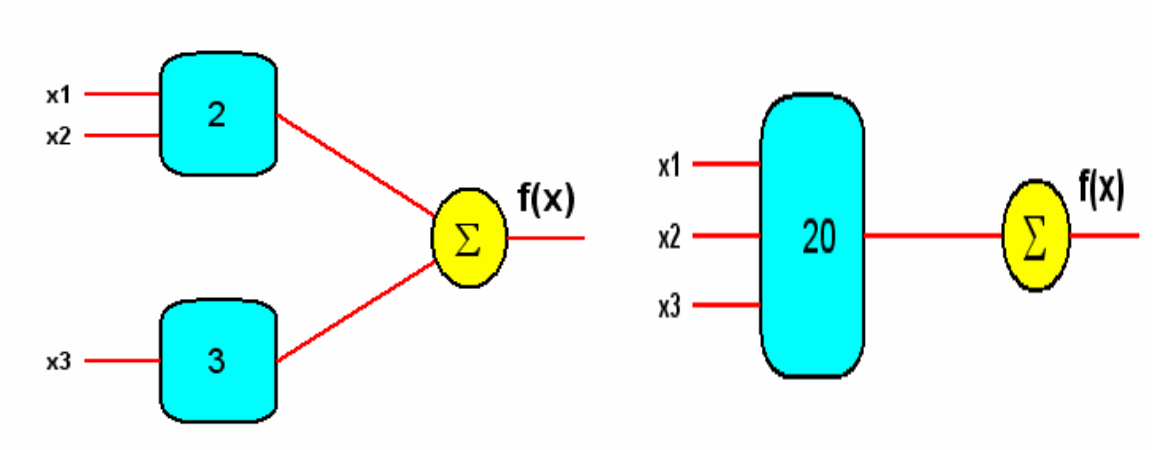


Fig. 6.7 La varianza para cada una de las variables



(a) (b)

Fig. 6.8 ***a)*** Estructura Multi-RBFN seleccionada por el algoritmo. ***b)*** Estructura de una RBFN clásica para la función actual

Algoritmo Multi-RBFN						RBFN Clásica			
{ RBF }	# Parám	$NRMSE_{Tr}$	DesV	$NRMSE_{Test}$	DesV	RBF	# Parám	$NRMSE_{Tr}$	$NRMSE_{Test}$
{2 1}	15	0.145	6E-2	0.179	5E-2	2	10	0.338	0.370
{1 2}	15	0.290	1E-3	0.295	5E-2	3	15	0.155	0.184
{3 1}	20	0.096	1E-2	0.100	2E-2	4	20	0.092	0.114
{2 2}	20	0.105	2E-3	0.109	5E-3	5	25	0.110	0.130
{3 2}	25	0.08	2E-2	0.085	2E-2	6	30	0.095	0.125
{2 3}	25	0.084	2E-3	0.089	5E-3	7	35	0.099	0.130
						8	40	0.099	0.125
						9	45	0.086	0.111
						10	50	0.088	0.114
						11	55	0.091	0.114
						12	60	0.101	0.111
						13	65	0.108	0.109
						14	70	0.106	0.105
						15	75	0.098	0.108
						16	80	0.102	0.101
						17	85	0.082	0.107
						18	90	0.095	0.108
						19	95	0.080	0.107
						20	100	0.099	0.101

Tabla 6.4: NRMSE de entrenamiento y test obtenido por el algoritmo propuesto y por RBFN clásica para la función $f_4(x)$

- Función $f_5(x)$

$$f_6(x) = 3\ log(1+x_3) + sen(2\pi x_2) + 0\, x_1 \quad , x_1, x_2, x_3 \in [0,1] \tag{6.5}$$

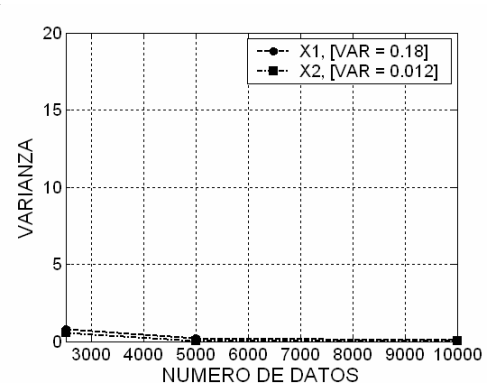
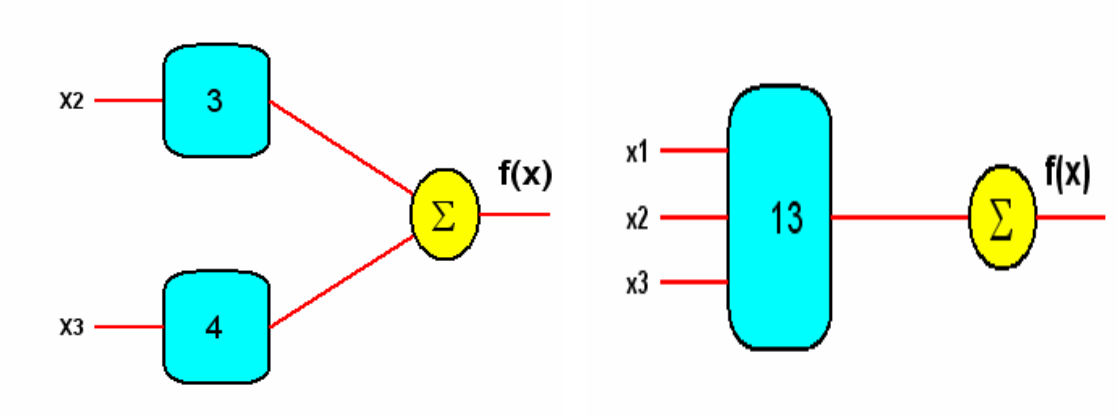


Fig. 6.9 La varianza para cada una de las variables



(a) **(b)**

Fig. 6.10 ***a)*** Estructura Multi-RBFN seleccionada por el algoritmo. ***b)*** Estructura de una RBFN clásica para la función actual

Algoritmo Multi-RBFN						RBFN Clásica			
{ RBF }	# Parám	$NRMSE_{Tr}$	DesV	$NRMSE_{Test}$	DesV	RBF	# Parám	$NRMSE_{Tr}$	$NRMSE_{Test}$
{2 1}	12	0.350	9E-3	0.351	1E-2	2	10	0.440	0.443
{1 2}	12	0.342	5E-3	0.343	8E-3	3	15	0.411	0.417
{2 2}	16	0.363	2E-2	0.362	7E-3	4	20	0.324	0.327
{1 3}	16	0.336	3E-3	0.337	5E-3	5	25	0.291	0.289
{2 3}	20	0.389	6E-3	0.359	5E-3	6	30	0.203	0.207
{1 4}	20	0.329	1E-3	0.326	1E-3	7	35	0.234	0.237
{3 3}	24	0.161	2E-2	0.161	2E-2	8	40	0.178	0.184
{2 4}	24	0.127	1E-2	0.130	9E-3	9	45	0.167	0.174
{4 3}	28	0.068	2E-3	0.060	1E-3	10	50	0.147	0.146
{3 4}	28	0.051	1E-3	0.050	1E-3	11	55	0.115	0.116
						12	60	0.098	0.101
						13	65	0.095	0.095

Tabla 6.5: NRMSE de entrenamiento y test obtenido por el algoritmo propuesto y por RBFN clásica para la función $f_5(x)$

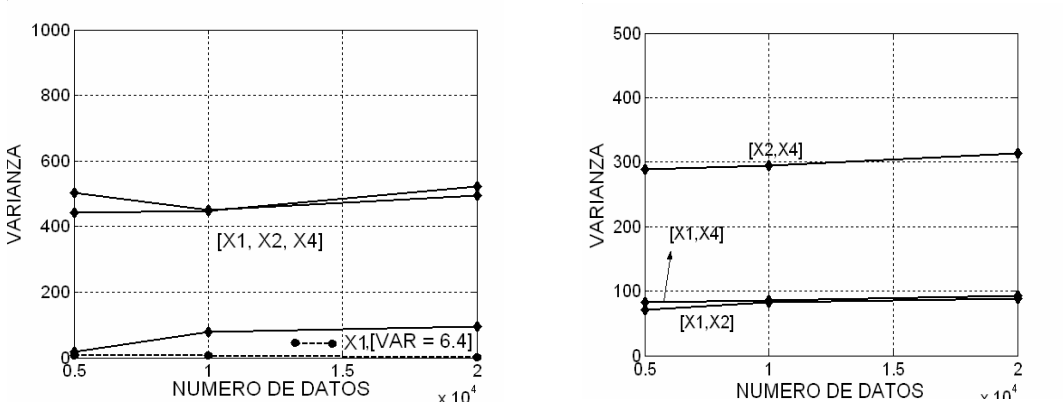
6.4 Funciones de cuatro variables

En este apartado se evalúa el algoritmo propuesto utilizando distintas funciones de cuatro dimensiones. Estas funciones artificiales muestran la capacidad del algoritmo de seleccionar la estructura adecuada del sistema Multi-RBFN. Los conjuntos de entrenamiento y validación que se han utilizado para estas funciones, están formados por 10000 puntos de forma aleatoria en el dominio de entrada. El conjunto de validación esta formado por la mitad del conjunto de los datos.

- Función $f_6(x)$

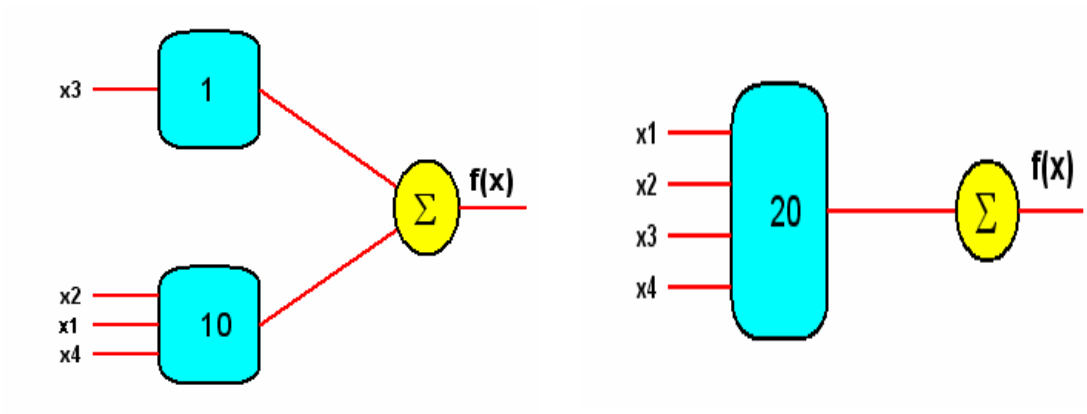
$$f_6(x) = e^{(-2\,x_3)} + 2\,sen(2\pi\,x_2)\,.\,x_1.\,x_4 \quad ,x_1, x_2, x_3, x_4 \in [0,1] \tag{6.6}$$

El algoritmo propuesto selecciona la arquitectura óptima del sistema Multi-RBFN para la función $f_6(x)$ depende al valor umbral de la varianza después de analizar cada variables (Fig.6.11.*a*) y cada sub-conjunto posible de los variables (Fig.6.11.*b*). En la función $f_6(x)$ una de las variables debe ir sola a una Sub-RBFN y el sub-conjunto de resto dirigirá a una Sub-RBFN, como se muestra la Fig.6.12.*a*.



(a) (b)

Fig. 6.11 ***a)*** La varianza para cada una de las variables ***b)*** La varianza para cada sub-conjunto de dos variables



(a) (b)

Fig. 6.12 ***a)*** Estructura Multi-RBFN seleccionada por el algoritmo. ***b)*** Estructura de una RBFN clásica para la función actual

Algoritmo Multi-RBFN						RBFN Clásica			
{ RBF }	**# Parám**	**$NRMSE_{Tr}$**	**DesV**	**$NRMSE_{Test}$**	**DesV**	**RBF**	**# Parám**	**$NRMSE_{Tr}$**	**$NRMSE_{Test}$**
{2 1}	18	0.678	2E-3	0.719	0.0248	2	12	0.632	0.640
{1 2}	18	0.531	3E-2	0.583	0.0618	3	18	0.486	0.485
{2 2}	24	0.483	7E-2	0.533	0.1010	4	24	0.535	0.536
{1 3}	24	0.501	1E-2	0.549	0.0410	5	30	0.466	0.471
{2 3}	30	0.544	5E-2	0.594	0.0023	6	36	0.408	0.420
{1 4}	30	0.433	1E-2	0.492	0.0346	7	42	0.365	0.373
{2 4}	36	0.444	6E-2	0.502	0.0427	8	48	0.378	0.382
{1 5}	36	0.393	7E-2	0.458	0.0139	9	54	0.267	0.272
{2 5}	42	0.400	9E-2	0.462	0.0289	10	60	0.385	0.398
{1 6}	42	0.364	1E-3	0.404	0.0035	11	66	0.331	0.342
{2 6}	48	0.393	3E-2	0.456	0.0791	12	72	0.317	0.330
{1 7}	48	0.258	6E-3	0.307	0.0023	13	78	0.292	0.303
{2 7}	54	0.293	5E-2	0.335	0.0375	14	84	0.285	0.297
{1 8}	54	0.259	3E-2	0.309	0.0173	15	90	0.252	0.261
{2 8}	60	0.237	4E-3	0.330	0.0433	16	96	0.198	0.207
{1 9}	60	0.214	1E-2	0.295	0.0150	17	102	0.177	0.186
{2 9}	66	0.187	5E-3	0.260	0.0023	18	108	0.199	0.207
{1 10}	66	0.188	6E-3	0.213	2E-2	19	114	0.206	0.213
						20	120	0.205	0.205

Tabla 6.6: NRMSE de entrenamiento y test obtenido por el algoritmo propuesto y por RBFN clásica para la función $f_6(x)$

- Función $f_7(x)$

$$f_7(x) = e^{(2\,x_1 .sen(\pi x_4))} + sen(x_2 .x_3) \qquad , x_1, x_2, x_3, x_4 \in [0,1] \tag{6.7}$$

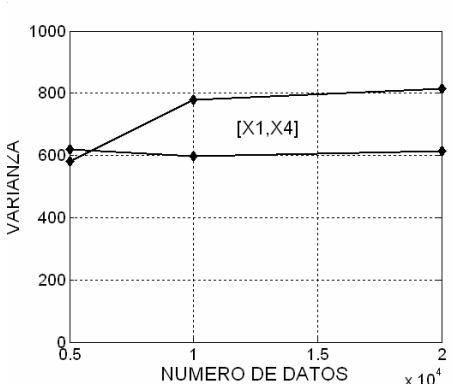
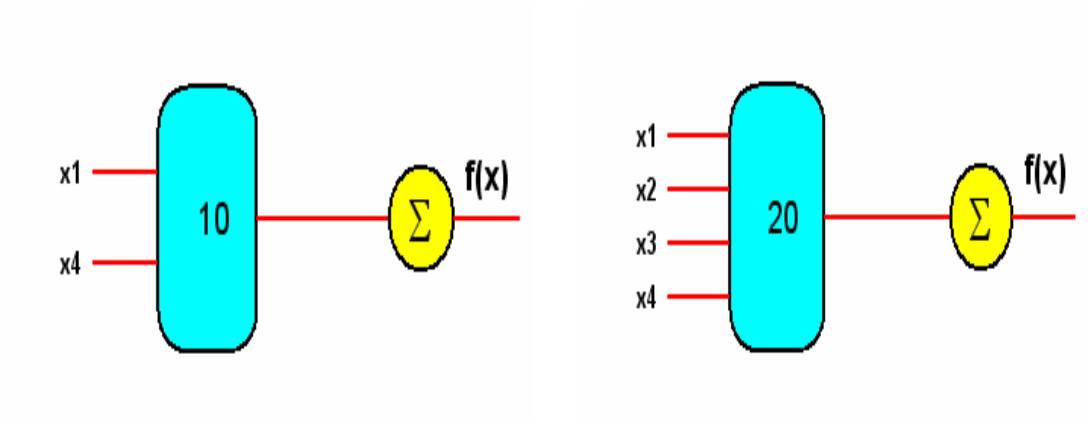


Fig. 6.13 La varianza para cada una de las variables



(a) (b)

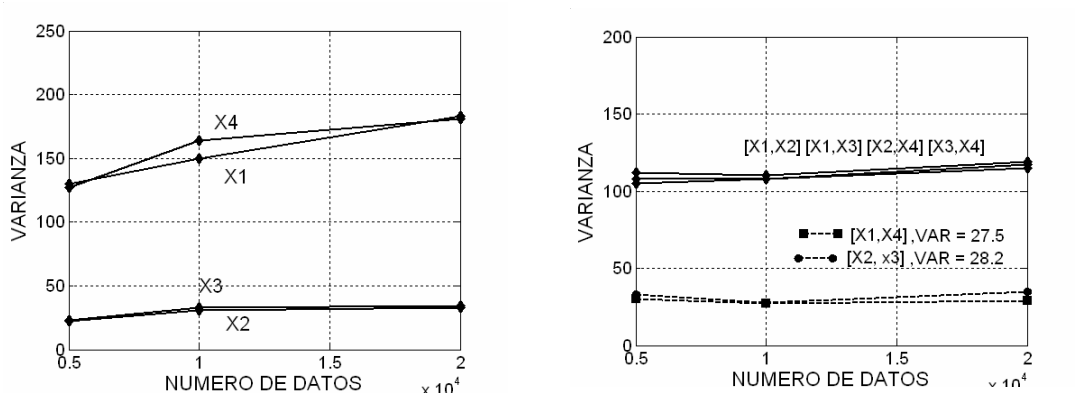
Fig. 6.14 ***a)*** Estructura Multi-RBFN seleccionada por el algoritmo. ***b)*** Estructura de una RBFN clásica para la función actual

Algoritmo Multi-RBFN						RBFN Clásica			
{ RBF }	# Paráme	$NRMSE_{Tr}$	DesV	$NRMSE_{Test}$	DesV	RBF	# Paráme	$NRMSE_{Tr}$	$NRMSE_{Test}$
{2}	8	0.311	3E-2	0.311	3E-2	2	12	0.524	0.533
{3}	12	0.224	5E-2	0.227	4E-2	3	18	0.433	0.448
{4}	16	0.214	5E-2	0.213	5E-2	4	24	0.331	0.336
{5}	20	0.175	2E-3	0.176	7E-3	5	30	0.270	0.278
{6}	24	0.163	2E-3	0.164	5E-3	6	36	0.263	0.272
{7}	28	0.162	4E-3	0.163	9E-3	7	42	0.256	0.263
{8}	32	0.157	6E-4	0.159	6E-3	8	48	0.258	0.267
{9}	36	0.157	4E-3	0.159	5E-3	9	54	0.257	0.267
{10}	40	0.157	2E-3	0.159	4E-3	10	60	0.247	0.254
						11	66	0.250	0.249
						12	72	0.215	0.214
						13	78	0.201	0.204
						14	84	0.181	0.189
						15	90	0.172	0.178
						16	96	0.152	0.148
						17	102	0.142	0.145
						18	108	0.131	0.129
						19	114	0.124	0.127
						20	120	0.117	0.119

Tabla 6.7: NRMSE de entrenamiento y test obtenido por el algoritmo propuesto y por RBFN clásica para la función $f_7(x)$

- Función $f_8(x)$

$$f_8(x) = 4(x_1\text{-}0.5).(x_4\text{-}0.5) + sen(2\pi\sqrt{(x_2)^2 + (x_3)^2} \quad , x_1, x_2, x_3, x_4 \in [0,1] \quad (6.8)$$

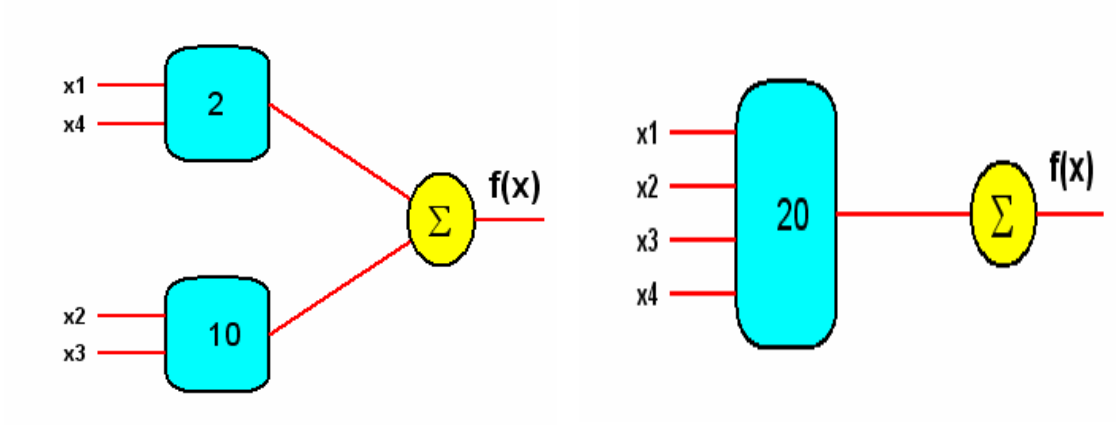


(a) (b)

Fig. 6.15 ***a)*** La varianza para cada una de las variables ***b)*** La varianza para cada sub-conjunto de dos variables

Algoritmo Multi-RBFN						**RBFN Clásica**			
{ RBF }	**# Paráme**	**$NRMSE_{Tr}$**	**DesV**	**$NRMSE_{Test}$**	**DesV**	**RBF**	**# Paráme**	**$NRMSE_{Tr}$**	**$NRMSE_{Test}$**
{2 1}	18	0.784	1E-2	0.792	4E-3	2	12	0.690	0.688
{1 2}	18	0.561	8E-2	0.576	4E-3	3	18	0.573	0.576
{2 2}	24	0.459	1E-2	0.470	2E-2	4	24	0.512	0.519
{1 3}	24	0.509	8E-3	0.522	5E-3	5	30	0.427	0.431
{2 3}	30	0.377	1E-2	0.388	2E-2	6	36	0.411	0.415
{1 4}	30	0.409	1E-2	0.420	1E-2	7	42	0.379	0.383
{3 3}	36	0.357	3E-3	0.367	2E-5	8	48	0.391	0.396
{2 4}	36	0.311	7E-3	0.322	7E-3	9	54	0.372	0.380
{3 4}	42	0.317	6E-2	0.328	5E-3	10	60	0.343	0.346
{2 5}	42	0.173	1E-2	0.186	1E-2	11	66	0.313	0.316
{3 5}	48	0.169	5E-3	0.181	1E-3	12	72	0.313	0.315
{2 6}	48	0.113	7E-3	0.126	9E-3	13	78	0.297	0.301
{3 6}	54	0.120	8E-3	0.131	8E-3	14	84	0.273	0.281
{2 7}	54	0.147	7E-2	0.119	4E-3	15	90	0.277	0.283
{3 7}	60	0.112	2E-3	0.126	2E-3	16	96	0.259	0.267
{2 8}	60	0.101	4E-3	0.114	3E-2	17	102	0.269	0.275
{3 8}	66	0.104	7E-3	0.117	9E-3	18	108	0.190	0.200
{2 9}	66	0.087	3E-3	0.101	8E-3	19	114	0.194	0.202
{3 9}	72	0.093	3E-3	0.107	2E-3	20	120	0.169	0.180
{2 10}	72	0.091	6E-3	0.102	5E-3				

Tabla 6.8: NRMSE de entrenamiento y test obtenido por el algoritmo propuesto y por RBFN clásica para la función $f_8(x)$



(a) (b)

Fig. 6.16 ***a)*** Estructura Multi-RBFN seleccionada por el algoritmo. ***b)*** Estructura de una RBFN clásica para la función actual

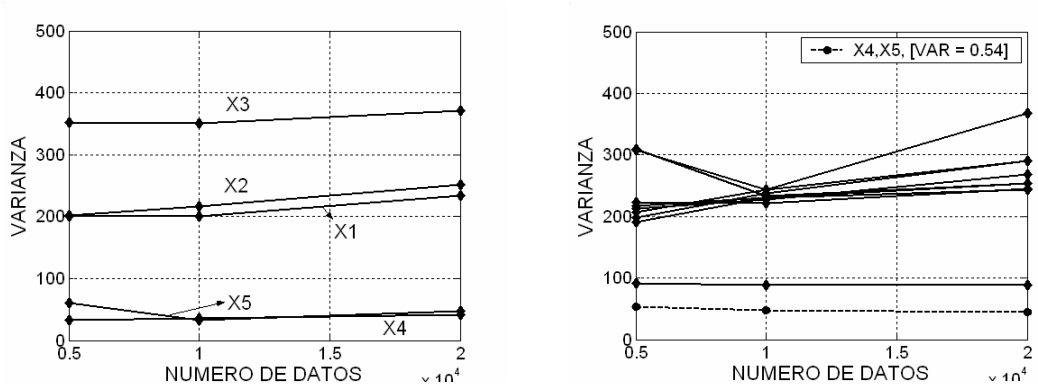
6.5 Funciones de cinco variables

En este apartado se evalúa el algoritmo propuesto utilizando distintas funciones de cenco dimensiones. Estas funciones artificiales muestran la capacidad del algoritmo de seleccionar la estructura adecuada del sistema Multi-RBFN. Los conjuntos de entrenamiento y validación que se han utilizado para estas funciones, están formados por 10000 puntos de forma aleatoria en el dominio de entrada. El conjunto de validación esta formado por la mitad del conjunto de los datos.

- Función $f_9(x)$

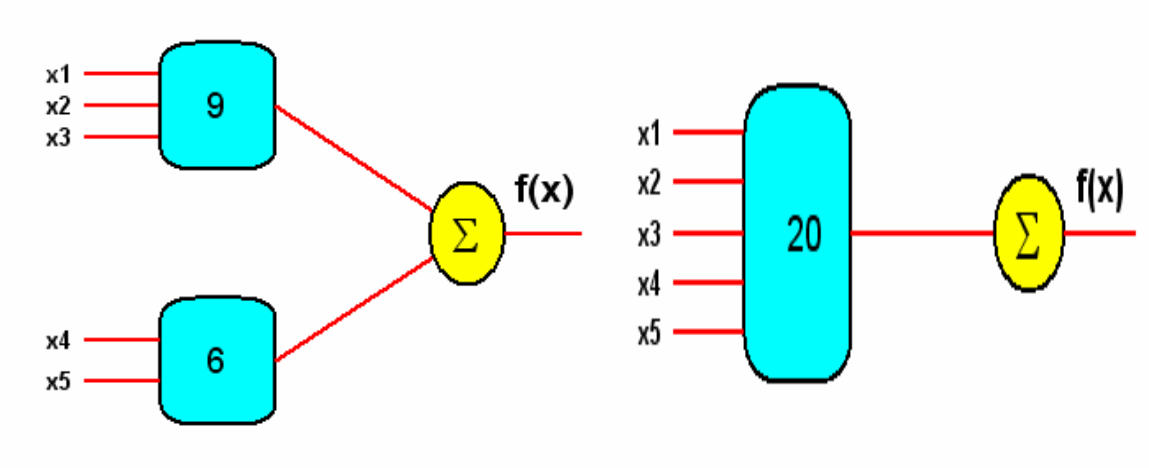
$$f_9(x) = sen(2\ \pi\ x_1.x_2).\ x_3 + e^{(-4\ x_4.x_5)} \quad ,x_1, x_2, x_3, x_4, x_5\ \in\ [0,1] \tag{6.9}$$

El algoritmo propuesto selecciona la arquitectura óptima del sistema Multi-RBFN para la función $f_9(x)$, depende al valor umbral de la varianza después de analizar cada variables (Fig.6.17.*a*) y cada sub-conjunto posible de los variables (Fig.6.17.*b*). En la función $f_9(x)$ un sub-conjunto de dos variables debe ir sola a una Sub-RBFN y el sub-conjunto de resto dirigirá a una Sub-RBFN, como se muestra la Fig.6.33.*a*.



(a) (b)

Fig. 6.17 ***a)*** La varianza para cada una de las variables ***b)*** La varianza para cada sub-conjunto de dos variables



(a) (b)

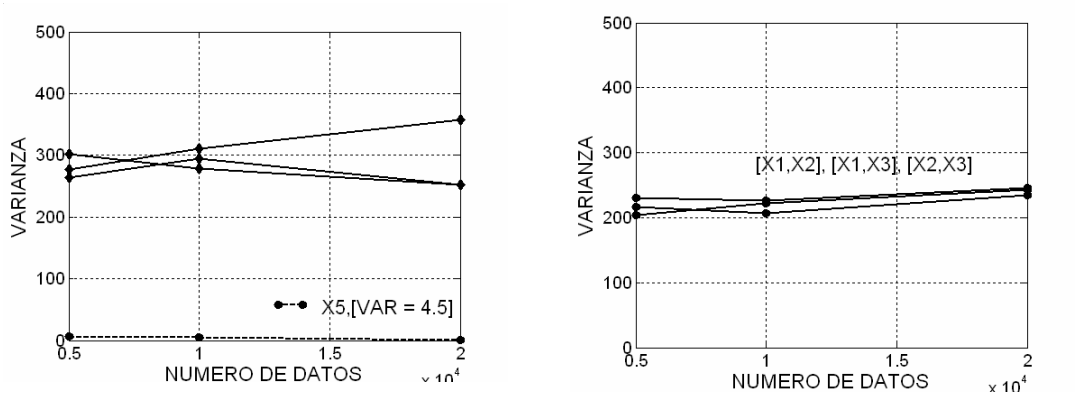
Fig. 6.18 ***a)*** Estructura Multi-RBFN seleccionada por el algoritmo. ***b)*** Estructura de una RBFN clásica para la función actual

Algoritmo Multi-RBFN						RBFN Clásica			
{ RBF }	# Paráme	$NRMSE_{Tr}$	DesV	$NRMSE_{Test}$	DesV	RBF	# Paráme	$NRMSE_{Tr}$	$NRMSE_{Test}$
{2 1}	21	0.593	1E-3	0.610	9E-3	2	14	0.631	0.604
{1 2}	21	0.606	2E-3	0.618	2E-3	3	21	0.593	0.617
{3 1}	28	0.508	5E-3	0.529	5E-2	4	28	0.556	0.577
{2 2}	28	0.556	1E-3	0.573	9E-3	5	35	0.367	0.416
{4 1}	35	0.428	3E-2	0.451	4E-2	6	42	0.333	0.383
{3 2}	35	0.392	0.0	0.412	1E-2	7	49	0.336	0.386
{4 2}	42	0.333	1E-2	0.362	2E-2	8	56	0.319	0.367
{3 3}	42	0.373	2E-3	0.398	9E-3	9	63	0.329	0.374
{5 2}	49	0.300	1E-2	0.331	2E-2	10	70	0.311	0.361
{4 3}	49	0.306	2E-2	0.336	5E-3	11	77	0.309	0.357
{5 3}	56	0.301	8E-3	0.333	3E-3	12	84	0.288	0.336
{4 4}	56	0.286	3E-3	0.320	4E-2	13	91	0.313	0.359
{6 3}	63	0.278	7E-3	0.312	5E-3	14	98	0.289	0.335
{5 4}	63	0.280	4E-3	0.313	1E-2	15	105	0.298	0.343
{7 3}	70	0.272	1E-2	0.305	0.0	16	112	0.303	0.351
{6 4}	70	0.289	1E-2	0.319	3E-3	17	119	0.302	0.347
{8 3}	77	0.254	5E-3	0.288	2E-2	18	126	0.255	0.301
{7 4}	77	0.244	5E-3	0.279	2E-2	19	133	0.233	0.243
{8 4}	84	0.230	9E-3	0.264	2E-2	20	140	0.183	0.190
{7 5}	84	0.265	3E-3	0.298	1E-2				
{9 4}	91	0.240	8E-3	0.274	9E-3				
{8 5}	91	0.224	1E-2	0.260	2E-2				
{9 5}	98	0.210	1E-2	0.246	3E-2				
{8 6}	98	0.227	2E-3	0.266	0.0				
{10 5}	105	0.191	3E-2	0.213	1E-3				
{9 6}	105	0.185	3E-2	0.209	3E-2				

Tabla 6.9: NRMSE de entrenamiento y test obtenido por el algoritmo propuesto y por RBFN clásica para la función $f_9(x)$

- Función $f_{10}(x)$

$$f_{10}(x) = 0\, x_4 + 2\, sen(2\, \pi\, x_1.x_2.x_3) + 1.5\, x_5 \quad , x_1, x_2, x_3, x_4, x_5 \in [0,1] \qquad (6.10)$$

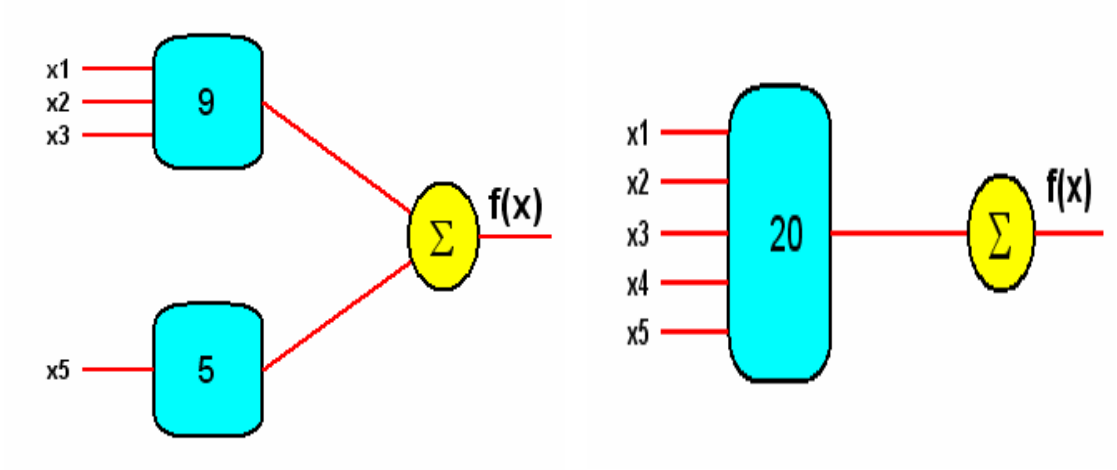


(a) (b)

Fig. 6.19 ***a)*** La varianza para cada una de las variables ***b)*** La varianza para cada sub-conjunto de dos variables

Algoritmo Multi-RBFN						**RBFN Clásica**			
{ RBF }	**# Parám**	**$NRMSE_{Tr}$**	**DevS**	**$NRMSE_{Test}$**	**DevS**	**RBF**	**# Parám**	**$NRMSE_{Tr}$**	**$NRMSE_{Test}$**
{2 1}	18	0.614	1E-2	0.627	7E-3	2	14	0.669	0.674
{1 2}	18	0.642	2E-2	0.655	3E-3	3	21	0.641	0.647
{3 1}	24	0.580	2E-2	0.591	2E-3	4	28	0.599	0.613
{2 2}	24	0.599	1E-2	0.613	4E-3	5	35	0.582	0.594
{4 1}	30	0.476	1E-2	0.488	2E-3	6	42	0.580	0.592
{3 2}	30	0.553	2E-2	0.562	3E-2	7	49	0.304	0.339
{5 1}	36	0.239	1E-2	0.252	5E-2	8	56	0.298	0.332
{4 2}	36	0.392	1E-2	0.402	1E-2	9	63	0.285	0.314
{6 1}	42	0.214	3E-2	0.223	2E-2	10	70	0.293	0.323
{5 2}	42	0.203	1E-3	0.218	4E-3	11	77	0.280	0.308
{6 2}	48	0.197	3E-2	0.206	2E-2	12	84	0.282	0.309
{5 3}	48	0.207	1E-2	0.227	1E-2	13	91	0.280	0.307
{6 3}	54	0.203	2E-2	0.189	4E-2	14	98	0.257	0.283
{5 4}	54	0.209	3E-2	0.224	3E-2	15	105	0.245	0.275
{6 4}	60	0.189	6E-2	0.188	3E-2	16	112	0.243	0.273
{5 5}	60	0.187	3E-2	0.188	3E-2	17	119	0.237	0.267
{7 4}	66	0.181	3E-2	0.184	3E-2	18	126	0.232	0.252
{6 5}	66	0.197	2E-2	0.209	2E-2	19	133	0.218	0.231
{8 4}	72	0.168	2E-2	0.182	2E-2	20	140	0.207	0.217
{7 5}	72	0.182	5E-2	0.192	4E-2				
{9 4}	78	0.157	1E-3	0.174	5E-3				
{8 5}	78	0.164	2E-2	0.180	2E-2				
{10 4}	84	0.148	6E-3	0.166	5E-3				
{9 5}	84	0.143	2E-2	0.166	1E-2				

Tabla 6.10: NRMSE de entrenamiento y test obtenido por el algoritmo propuesto y por RBFN clásica para la función $f_{10}(x)$



(a) (b)

Fig. 6.20 ***a)*** Estructura Multi-RBFN seleccionada por el algoritmo. ***b)*** Estructura de una RBFN clásica para la función actual clásica

6.6 Funciones de seis variables

En este apartado se evalúa el algoritmo propuesto utilizando distintas funciones de cenco dimensiones. Estas funciones artificiales muestran la capacidad del algoritmo de seleccionar la estructura adecuada del sistema Multi-RBFN. Los conjuntos de entrenamiento y validación que se han utilizado para estas funciones, están formados por 10000 puntos de forma aleatoria en el dominio de entrada. El conjunto de validación esta formado por la mitad del conjunto de los datos.

- Función $f_{11}(x)$

$$f_{11}(x) = 10\,sen(\pi x_1 \cdot x_2) + 20\,(x_3\text{-}0.5)^2 + 10\,x_4 + 5x_5 + 0\,x_6 \quad x_1, x_2, x_3, x_4, x_5, x_6 \in [0,1] \quad (6.11)$$

El algoritmo propuesto selecciona la arquitectura óptima del sistema Multi-RBFN para la función $f_{11}(x)$, depende al valor umbral de la varianza después de analizar cada variables (Fig.6.21.*a*). En la función $f_{11}(x)$ unas variables deben ir solas a una Sub-RBFN y el sub-conjunto de resto dirigirá a una Sub-RBFN, como se muestra la Fig.6.22.*a*.

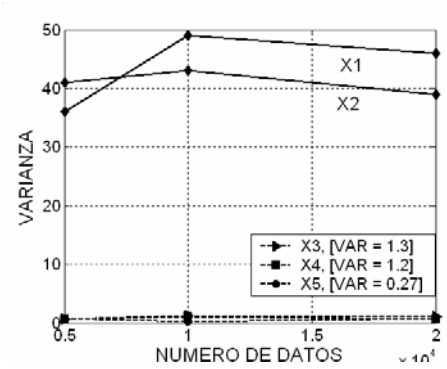


Fig. 6.21 La varianza para cada una de las variables

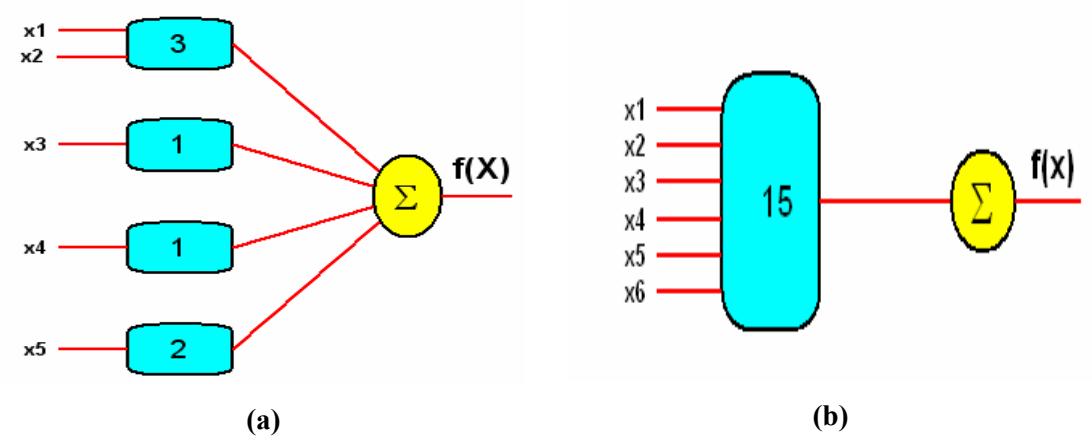


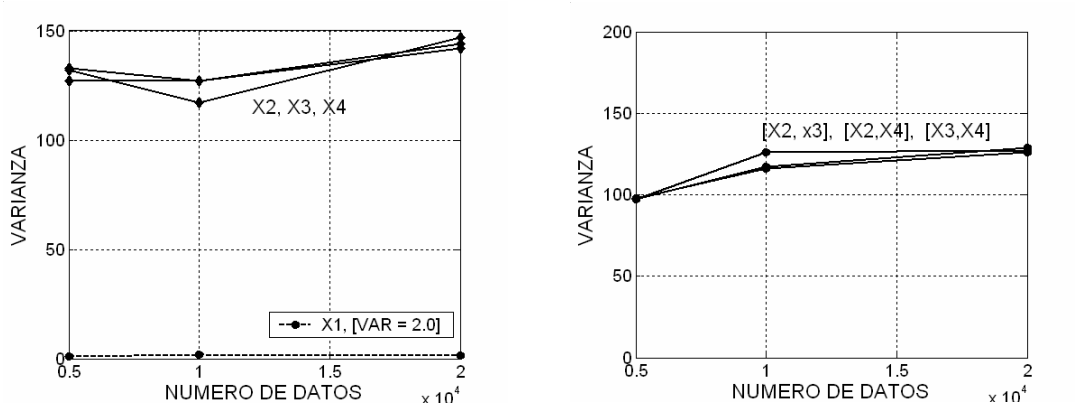
Fig. 6.22 ***a)*** Estructura Multi-RBFN seleccionada por el algoritmo. ***b)*** Estructura de una RBFN clásica para la función actual

Algoritmo Multi-RBFN						**RBFN Clásica**			
{ RBF }	**# Parám**	**NRMSE$_{Tr}$**	**DesV**	**NRMSE$_{Test}$**	**DesV**	**RBF**	**# Parám**	**NRMSE$_{Tr}$**	**NRMSE$_{Test}$**
{2 1 1 1}	35	0.212	2E-3	0.214	1E-4	2	16	0.428	0.437
{1 2 1 1}	35	0.246	6E-3	0.252	4E-4	3	24	0.331	0.328
{1 1 2 1}	35	0.238	1E-2	0.243	5E-3	4	32	0.301	0.305
{1 1 1 2}	35	0.241	1E-2	0.246	6E-4	5	40	0.316	0.316
{3 1 1 1}	42	0.198	2E-1	0.204	1E-4	6	48	0.279	0.278
{2 2 1 1}	42	0.221	9E-3	0.225	1E-2	7	56	0.213	0.214
{2 1 2 1}	42	0.209	2E-3	0.216	6E-3	8	64	0.284	0.284
{2 1 1 2}	42	0.212	1E-3	0.216	1E-4	9	72	0.249	0.252
{4 1 1 1}	49	0.183	1E-2	0.189	8E-3	10	80	0.231	0.237
{3 2 1 1}	49	0.146	8E-2	0.147	3E-2	11	88	0.211	0.219
{3 1 2 1}	49	0.075	5E-3	0.084	3E-3	12	96	0.206	0.212
{3 1 1 2}	49	0.080	3E-3	0.088	2E-3	13	104	0.179	0.190
						14	112	0.153	0.173
						15	120	0.144	0.154

Tabla 6.11: NRMSE de entrenamiento y test obtenido por el algoritmo propuesto y por RBFN clásica para la función $f_{11}(x)$

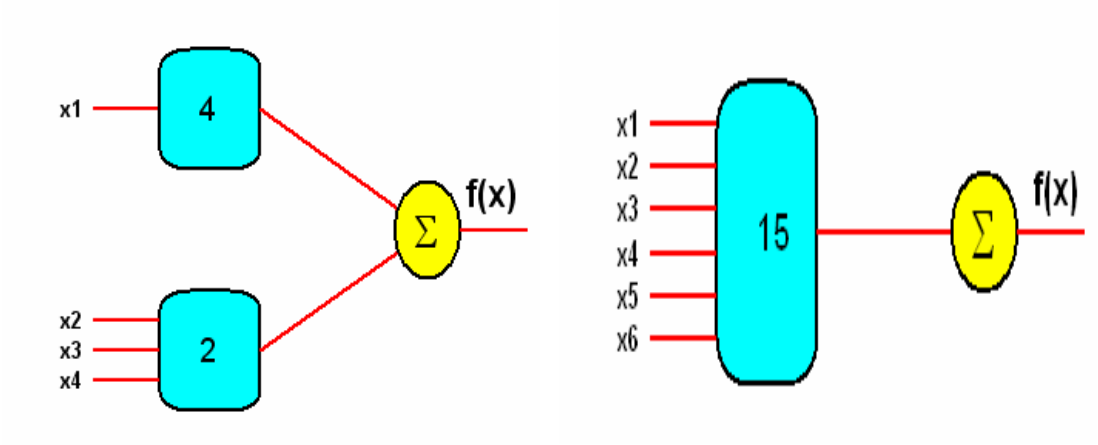
- Función $f_{12}(x)$

$$f_{12}(x) = 5\,sen(2\pi x_1) + 10 x_2 . \cdot x_3 \cdot x_4 + 0((x_6)^2 \cdot x_5) \quad , x_1, x_2, x_3, x_4, x_5, x_6 \in [0,1] \qquad (6.12)$$



(a) (b)

Fig. 6.23 ***a)*** La varianza para cada una de las variables ***b)*** La varianza para cada sub-conjunto de dos variables



(a) (b)

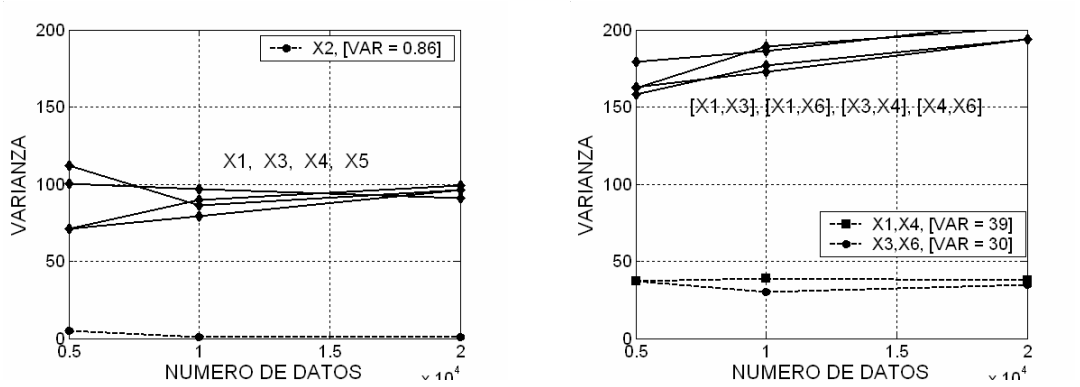
Fig. 6.24 ***a)*** Estructura Multi-RBFN seleccionada por el algoritmo ***b)*** Estructura de una RBFN clásica para el ejemplo actual

Algoritmo Multi-RBFN						RBFN Clásica			
{ RBF }	# Parám	$NRMSE_{Tr}$	DesV	$NRMSE_{Test}$	DesV	RBF	# Parám	$NRMSE_{Tr}$	$NRMSE_{Test}$
{2 1}	12	0.362	1E-2	0.372	1E-2	2	16	0.602	0.601
{1 2}	12	0.164	3E-2	0.186	2E-2	3	24	0.563	0.565
{3 1}	16	0.121	6E-3	0.114	7E-3	4	32	0.540	0.548
{2 2}	16	0.081	9E-3	0.092	2E-2	5	40	0.540	0.546
{4 1}	20	0.081	2E-2	0.092	1E-2	6	48	0.536	0.543
{3 2}	20	0.084	1E-2	0.100	7E-3	7	56	0.496	0.505
{5 1}	24	0.060	2E-2	0.076	2E-2	8	64	0.530	0.540
{4 2}	24	0.068	1E-2	0.084	6E-3	9	72	0.436	0.446
						10	80	0.485	0.496
						11	88	0.327	0.338
						12	96	0.323	0.334
						13	104	0.340	0.350
						14	112	0.318	0.326
						15	120	0.242	0.249

Tabla 6.12: NRMSE de entrenamiento y test obtenido por el algoritmo propuesto y por RBFN clásica para la función $f_{12}(x)$

- Función $f_{13}(x)$

$$f_{13}(x) = 5\,(x_3 \cdot x_6) + 2\,sen(2\pi x_1 \cdot x_4) + 2\,x_2 + 0.0001\,x_5 \quad , x_1, x_2, x_3, x_4, x_5, x_6 \in [0,1] \tag{6.13}$$

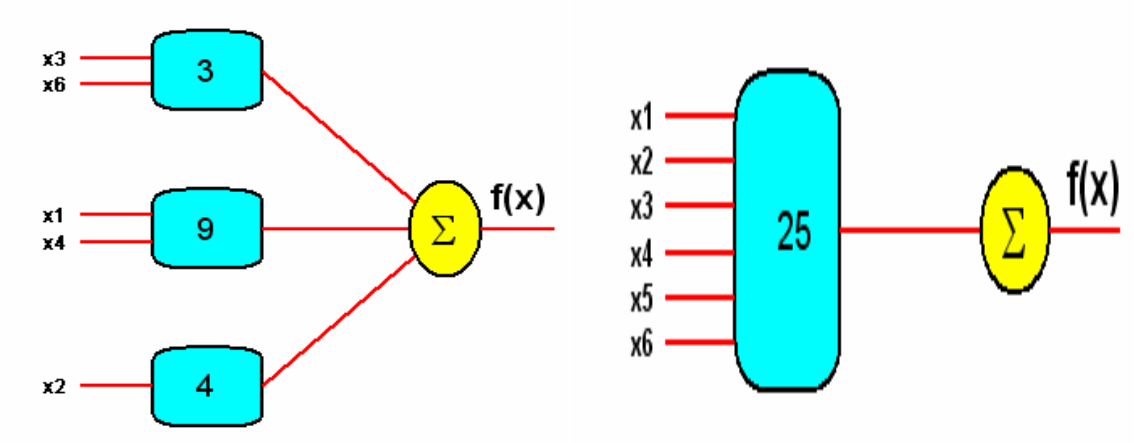


(a) (b)

Fig. 6.25 ***a)*** La varianza para cada una de las variables ***b)*** La varianza para cada sub-conjunto de dos variables

Algoritmo Multi-RBFN						RBFN Clásica			
{ RBF }	# Parám	$NRMSE_{Tr}$	DesV	$NRMSE_{Test}$	DesV	RBF	# Parám	$NRMSE_{Tr}$	$NRMSE_{Test}$
{2 1 1}	28	0.523	1E-2	0.531	6E-3	2	16	0.568	0.581
{1 2 1}	28	0.342	4E-2	0.351	7E-3	3	24	0.522	0.532
{1 1 2}	28	0.523	1E-2	0.531	5E-3	4	32	0.351	0.357
{2 2 1}	35	0.336	8E-3	0.348	7E-3	5	40	0.339	0.349
{1 3 1}	35	0.250	2E-2	0.271	3E-2	6	48	0.321	0.324
{1 2 2}	35	0.333	5E-3	0.345	7E-3	7	56	0.307	0.309
{2 3 1}	42	0.242	2E-2	0.262	3E-2	8	64	0.305	0.310
{1 4 1}	42	0.218	3E-3	0.240	1E-2	9	72	0.304	0.308
{1 3 2}	42	0.237	2E-3	0.260	1E-2	10	80	0.304	0.309
{2 4 1}	49	0.216	3E-3	0.238	1E-2	11	88	0.300	0.303
{1 5 1}	49	0.165	7E-3	0.197	1E-2	12	96	0.260	0.267
{1 4 2}	49	0.207	2E-2	0.230	2E-2	13	104	0.295	0.299
{2 5 1}	56	0.155	9E-2	0.188	2E-2	14	112	0.291	0.295
{1 6 1}	56	0.148	3E-2	0.182	2E-2	15	120	0.232	0.238
{1 5 2}	56	0.145	2E-2	0.178	3E-2	16	128	0.234	0.240
{3 5 1}	63	0.142	8E-3	0.174	2E-2	17	136	0.230	0.235
{2 6 1}	63	0.139	1E-2	0.175	2E-2	18	144	0.276	0.282
{2 5 2}	63	0.151	2E-2	0.183	3E-2	19	152	0.236	0.244
{3 6 1}	70	0.135	5E-3	0.172	2E-2	20	160	0.222	0.226
{2 7 1}	70	0.137	4E-3	0.174	2E-2	21	168	0.232	0.237
{2 6 2}	70	0.138	2E-2	0.175	2E-2	22	176	0.249	0.259
{3 6 2}	77	0.141	2E-2	0.175	3E-2	23	184	0.212	0.222
{2 7 2}	77	0.139	4E-3	0.175	2E-2	24	192	0.228	0.234
{2 6 3}	77	0.137	7E-3	0.172	3E-2	25	200	0.245	0.253
{3 7 2}	84	0.139	1E-2	0.175	1E-2				
{2 8 2}	84	0.136	2E-3	0.172	2E-2				
{2 7 3}	84	0.137	2E-3	0.174	2E-2				
{4 7 2}	91	0.138	6E-3	0.175	2E-2				
{3 8 2}	91	0.136	2E-3	0.172	2E-2				
{3 7 3}	91	0.139	8E-3	0.175	1E-2				
{4 7 3}	98	0.137	3E-3	0.174	2E-2				
{3 8 3}	98	0.134	2E-3	0.170	2E-2				
{3 7 4}	98	0.131	1E-3	0.168	2E-2				
{4 7 4}	105	0.137	2E-3	0.173	2E-2				
{3 8 4}	105	0.132	8E-3	0.169	2E-2				
{3 7 5}	105	0.137	4E-3	0.173	2E-2				
{4 8 4}	112	0.135	4E-3	0.171	2E-2				
{3 9 4}	112	0.127	2E-2	0.166	3E-3				

Tabla 6.13: NRMSE de entrenamiento y test obtenido por el algoritmo propuesto y por RBFN clásica para la función $f_{13}(x)$



(a) (b)

Fig. 6.26 ***a)*** Estructura Multi-RBFN seleccionada por el algoritmo. ***b)*** Estructura de una RBFN clásica para la función actual

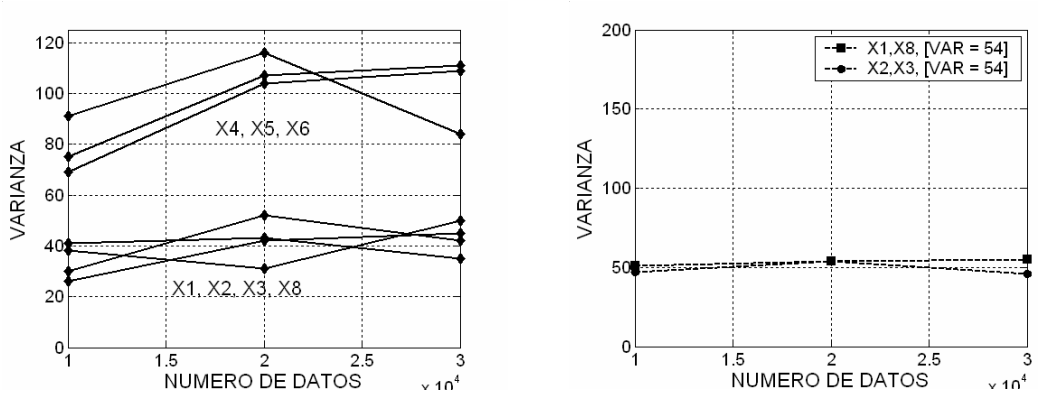
6.7 Función de ocho variables

En este apartado se evalúa el algoritmo propuesto utilizando distintas funciones de cenco dimensiones. Estas funciones artificiales muestran la capacidad del algoritmo de seleccionar la estructura adecuada del sistema Multi-RBFN. Los conjuntos de entrenamiento y validación que se han utilizado para estas funciones, están formados por 20000 puntos de forma aleatoria en el dominio de entrada. El conjunto de validación esta formado por la mitad del conjunto de los datos.

- Función $f_{14}(x)$

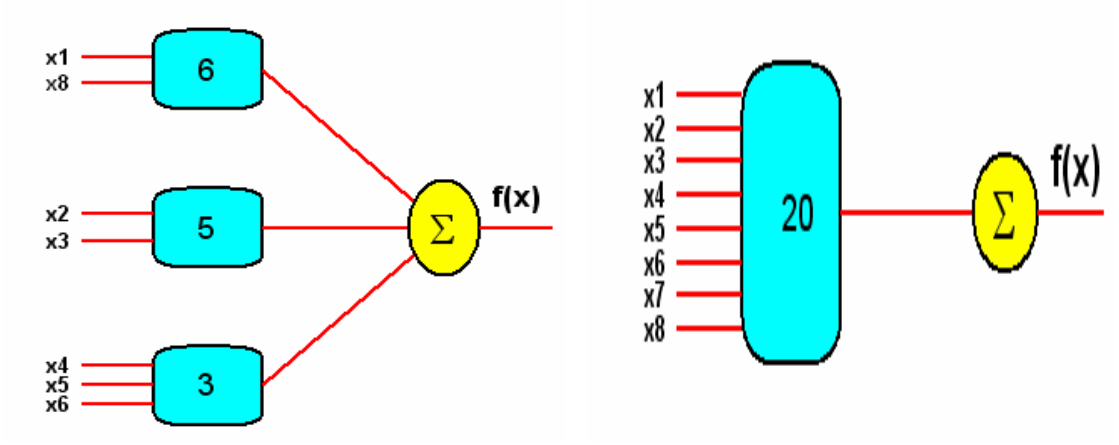
$$f_{14}(x) = 2\cos(2\pi x_1 \cdot x_8) + 6e^{(-2x_2 \cdot x_2)} + 7.5(x_4 \cdot x_5 \cdot x_6) + 0x_7 \quad , x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8 \in [0,1] \quad (6.14)$$

El algoritmo propuesto selecciona la arquitectura óptima del sistema Multi-RBFN para la función $f_{14}(x)$, depende al valor umbral de la varianza después de analizar cada variables (Fig.6.27.*a*), y cada sub-conjunto posible de los variables (Fig. 6.27.*b*). En la función $f_{14}(x)$ dos sub-conjuntos de dos variables deben ir cada uno a una Sub-RBFN y el sub-conjunto de resto dirigirá a una Sub-RBFN, como se muestra la Fig.6.28.*a*.



(a) (b)

Fig. 6.27 ***a)*** La varianza para cada una de las variables ***b)*** La varianza para cada sub-conjunto de dos variables



(a) (b)

Fig. 6.28 ***a)*** Estructura Multi-RBFN seleccionada por el algoritmo. ***b)*** Estructura de una RBFN clásica para la función actual clásica

Algoritmo Multi-RBFN						RBFN Clásica			
{ RBF }	# Parám	$NRMSE_{Tr}$	DesV	$NRMSE_{Test}$	DesV	RBF	# Parám	$NRMSE_{Tr}$	$NRMSE_{Test}$
{2 1 1}	36	0.422	1E-3	0.426	3E-3	2	20	0.459	0.473
{1 2 1}	36	0.451	1E-2	0.457	1E-2	3	30	0.433	0.450
{1 1 2}	36	0.498	6E-3	0.503	9E-3	4	40	0.382	0.393
{3 1 1}	45	0.419	2E-3	0.424	6E-4	5	50	0.397	0.416
{2 2 1}	45	0.260	4E-3	0.268	7E-3	6	60	0.380	0.402
{2 1 2}	45	0.423	6E-4	0.427	2E-3	7	70	0.339	0.354
{3 2 1}	54	0.269	4E-3	0.277	7E-3	8	80	0.349	0.372
{2 3 1}	54	0.261	6E-3	0.269	4E-2	9	90	0.344	0.366
{2 2 2}	54	0.307	3E-2	0.314	3E-2	10	100	0.316	0.341
{3 3 1}	63	0.247	6E-2	0.254	6E-2	11	110	0.328	0.351
{2 4 1}	63	0.264	7E-3	0.273	5E-2	12	120	0.259	0.284
{2 3 2}	63	0.260	5E-2	0.269	4E-3	13	130	0.250	0.262
{3 3 2}	72	0.225	8E-2	0.233	8E-3	14	140	0.241	0.249
{2 4 2}	72	0.261	3E-2	0.269	3E-2	15	150	0.234	0.241
{2 3 3}	72	0.233	5E-2	0.241	5E-2	16	160	0.216	0.231
{3 3 3}	81	0.221	8E-2	0.229	8E-2	17	170	0.208	0.215
{2 4 3}	81	0.207	8E-2	0.215	8E-2	18	180	0.189	0.190
{2 3 4}	81	0.214	8E-2	0.222	6E-2	19	190	0.177	0.185
{3 4 3}	90	0.200	6E-2	0.209	7E-2	20	200	0.157	0.162
{2 5 3}	90	0.207	8E-2	0.215	8E-2				
{2 4 4}	90	0.206	8E-2	0.214	8E-2				
{4 4 3}	99	0.157	6E-2	0.169	6E-2				
{3 5 3}	99	0.179	7E-2	0.190	7E-2				
{3 4 4}	99	0.203	7E-2	0.213	7E-2				
{5 4 3}	108	0.122	5E-3	0.137	1E-2				
{4 5 3}	108	0.150	5E-2	0.163	4E-2				
{4 4 4}	108	0.152	6E-2	0.165	6E-2				
{6 4 3}	117	0.124	7E-3	0.138	2E-2				
{5 5 3}	117	0.116	1E-2	0.132	2E-2				
{5 4 4}	117	0.117	4E-3	0.133	1E-2				
{6 5 3}	126	0.109	1E-2	0.125	2E-2				

Tabla 6.14: NRMSE de entrenamiento y test obtenido por el algoritmo propuesto y por RBFN clásica para la función $f_{14}(x)$

6.8 Comparación con otros métodos propuestos en la bibliografía

En esta sección se realizará una comparación entre los resultados obtenidos por el algoritmo propuesto con otras metodologías propuestas en la bibliografía. Por ello,

evaluaremos el rendimiento del algoritmo propuesto con otros métodos usualmente utilizados para resolver el problema de aproximación funcional.

- Función $f_2(x)$

  Los resultados obtenidos por el algoritmo propuesto para la función *f2(x)* definida en la sección 6.2, ecuación (6.2), se comparan con métodos usualmente utilizados para resolver el problema de aproximación funcional, como el método de "*Projection Pursuit*" (PP) [FRI-81], el método "*Multivariate Adaptive Regression Splines*" (MARS) [FRI-91]. Otro método muy parecido al anterior es la "*Constrained Topological Mapping*" [CHE-91b] y un método mejor para aproximar funciones denominado*"ANN"* [CHE-96]. En la Tabla 6.15 se presenta los resultados obtenidos por estos algoritmos para la función $f_2(x)$ y se compararan con otros métodos [POM-00], [GON-00], [RIV-03].

Algoritmo	***m***	**Test NRMSE**
MLP [CHE-91b]	15	0.096
PP [FRI-81]	-	0.128
CTM [CHE-91b]	-	0.170
MARS [FRI-91]	-	0.063
ECMN [CHE-96]	40	0.008
Pomares [2000]	3 × 5 (TP)	0.278
	4 × 6 (TP)	0.104
	5 × 9 (TP)	0.041
González 2001	5	0.3622 ± 0.0268
	10	0.1343 ± 0.0261
	15	0.0459 ± 0.0096
	21	0.0200 ± 0.0054
	29	0.0143 ± 0.0045
Rivas 2003	5	0.3666 ± 0.0168
	10	0.1108 ± 0.0135
	15	0.0368 ± 0.0092
	21	0.0191 ± 0.0036
	29	0.0147 ± 0.0022
Algoritmo propuesto	{1 4}	0.489 ± 0.0110
	{1 5}	0.365 ± 0.0006
	{1 6}	0.352 ± 0.0004
	{2 7}	0.128 ± 0.002
	{3 7}	0.040 ± 0.0003

Tabla 6.15: Comparativa de distintos algoritmos para la función $f_2(x)$

De los resultados obtenidos en la Tabla 6.15, se puede observar que nuestro algoritmo ha conseguido un muy buen grado de aproximación con una baja complejidad.

- Función $f_{11}(x)$

  Los resultados obtenidos por el algoritmo propuesto para la función $f_{11}(x)$ definida en la sección 6.2, ecuación (6.11), se comparan ahora con el mejor método utilizado por *Cherkassky* en [CHE-96]. Cherkassky seleccionó el método de ANN por una red MLP como el mejor método para aproximar la función $f_{11}(x)$. En la Tabla 6.17 se presenta los resultados obtenidos por el método en [CHE-96] y por el algoritmo propuesto para la función $f_{11}(x)$.

Algoritmo	***m***	**Test NRMSE**
MLP [CHE-96]	40	0.152
Algoritmo propuesto	{2 1 1 1}	0.214
	{3 1 1 1 }	0.204
	{4 1 1 1}	0.189
	{3 2 1 1}	0.147
	{3 1 2 1}	0.084
	{3 1 1 2}	0.088

Tabla 6.16: Comparativa de distintos algoritmos para la función $f_{11}(x)$

- Función $f_{15}(x)$

  Esta función la utilizó Pomares [POM-00] para evaluar su algoritmo de aproximación funcional. El algoritmo propuesto selecciona la estructura del sistema Multi-RBFN, eliminando la variable x_4, como se muestra en la Figura 6.29.

$$f_{15}(x) = 10 sen(\pi x_1 \cdot x_2) + 5 x_3 + 0 x_4 \quad x_1, x_2, x_3, x_4 \in [-1,1] \tag{6.15}$$

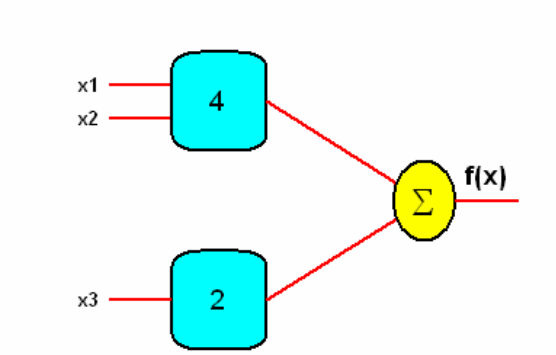


Fig. 6.29 Estructura Multi-RBFN seleccionada por el algoritmo.

Los resultados obtenidos por el algoritmo propuesto se compraran con el método para aproximar funciones presentado en la tesis de Pomares [POM-00].

Algoritmo	*m*	Test NRMSE
Pomares [2000]	1 × 1 × 1 × 1	1.000
	2 × 2 × 1 × 1	0.720
	2 × 2 × 2 × 1	0.553
	3 × 3 × 2 × 1	0.469
	4 × 4 × 2 × 1	0.087
	5 × 5 × 2 × 1	0.067
	6 × 6 × 2 × 1	0.033
Algoritmo propuesto	{1 2}	0.256
	{2 2}	0.213
	{1 3}	0.132
	{2 3}	0.061
	{3 3}	0.059
	{2 4}	0.085

Tabla 6.17: Comparativa de distintos algoritmos para la función $f_{15}(x)$

6.9 Efecto del Ruido

Todos los experimentos se han realizado con conjuntos de entrenamiento ideales. Cuando se trabaja con experimentos reales, lo normal es que los conjuntos de entrenamiento tengan ruido, los sensores suelen añadir pequeñas perturbaciones que quedan incluidas en los propios datos. Para comprobar cómo reacciona el algoritmo propuesto ante estas situaciones se han escogido algunas funciones (una función de dos, tres, cuatro, cinco, seis y ocho variables) y se les ha añadido un error aditivo blanco del

5%, 10% y 20%, para averiguar cuándo falla el algoritmo en seleccionar la estructura Multi-RBFN adecuada.

Los experimentos se han realizado con la misma configuración usada para aproximar estas funciones en las secciones anteriores y se han usado los mismos conjuntos de test para comprobar la capacidad de interpolación de los modelos obtenidos, es decir, conjuntos de test exentos de ruido y que el algoritmo nunca tiene en cuenta durante la fase de aprendizaje.

- Función $f_3(x)$

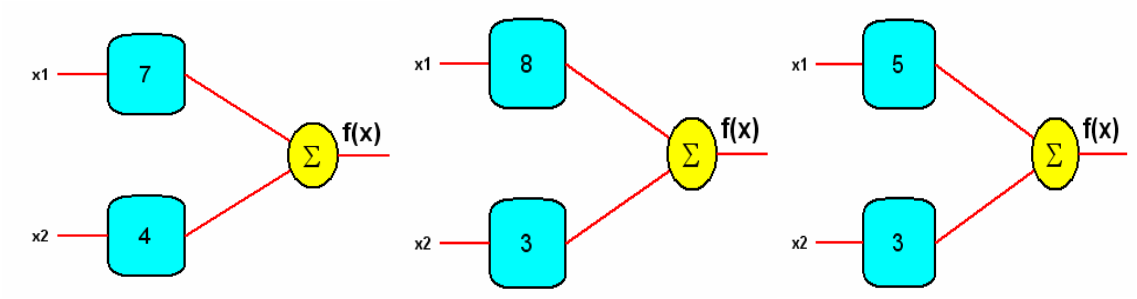
  Los datos de test de la función $f_3(x)$ están formados por 10200 puntos distribuidos en rejilla de 101× 101 celdas en el dominio de entrada.

Añadir 5%				**Añadir 10%**				**Añadir 20%**			
{ RBF }	**$NRMSE_{Tr}$**	**$NRMSE_{Test}$**	**DesV**	**{ RBF }**	**$NRMSE_{Tr}$**	**$NRMSE_{Test}$**	**DesV**	**{ RBF }**	**$NRMSE_{Tr}$**	**$NRMSE_{Test}$**	**DesV**
{2 1}	0.543	0.546	1E-2	{2 1}	0.557	0.551	4E-3	{2 1}	0.581	0.5632	2E-3
{1 2}	0.606	0.602	8E-3	{1 2}	0.612	0.601	4E-3	{1 2}	0.638	0.6132	2E-3
{3 1}	0.339	0.336	3E-2	{3 1}	0.366	0.341	4E-3	{3 1}	0.419	0.358	5E-3
{2 2}	0.308	0.300	2E-3	{2 2}	0.332	0.305	7E-3	{2 2}	0.393	0.323	5E-3
{3 2}	0.144	0.229	4E-3	{3 2}	0.187	0.235	4E-3	{3 2}	0.316	0.212	1E-2
{2 3}	0.263	0.238	4E-3	{2 3}	0.276	0.244	2E-3	{2 3}	0.494	0.375	1E-2
{4 2}	0.230	0.215	2E-3	{4 2}	0.268	0.192	2E-2	{4 2}	0.357	0.251	1E-2
{3 3}	0.186	0.211	1E-2	{3 3}	0.219	0.115	6E-3	{3 3}	0.361	0.275	8E-2
{4 3}	0.175	0.102	2E-3	{4 3}	0.168	0.110	3E-3	{5 2}	0.317	0.215	7E-2
{3 4}	0.122	0.088	1E-2	{3 4}	0.170	0.114	4E-3	{4 3}	0.288	0.168	4E-3
{5 3}	0.106	0.101	6E-3	{5 3}	0.154	0.086	2E-3	{5 3}	0.281	0.157	3E-3
{4 4}	0.117	0.064	9E-3	{4 4}	0.166	0.108	1E-3	{4 4}	0.287	0.167	5E-3
{6 3}	0.085	0.083	2E-2	{6 3}	0.147	0.072	2E-3	{6 3}	0.276	0.162	4E-3
{5 4}	0.094	0.056	8E-3	{5 4}	0.150	0.085	2E-3	{5 4}	0.277	0.165	3E-3
{7 3}	0.082	0.067	1E-2	{7 3}	0.144	0.068	2E-3	{7 3}	0.275	0.165	4E-3
{6 4}	0.083	0.052	2E-3	{6 4}	0.145	0.069	2E-3	{6 4}	0.275	0.166	4E-3
{8 3}	0.077	0.051	9E-3	{8 3}	0.142	0.064	8E-3	{8 3}	0.274	0.164	4E-3
{7 4}	0.082	0.051	8E-3	{7 4}	0.136	0.064	6E-3	{7 4}	0.274	0.163	3E-3

Tabla 6.18: NRMSE de entrenamiento y test obtenido por el algoritmo propuesto con añadir 5%, 10% y 20% de ruido para la función $f_3(x)$

De la Tabla 6.18 se puede observar cómo los errores de entrenamiento y test comienzan a decrecer de forma razonable cuando se añade 5% y 10 % de ruido, pero con el 20% de

ruido la estructura del sistema Multi-RBFN es más sencilla y el error de aproximación para datos de test exentos de ruido es mayor.



(a) (b) (c)

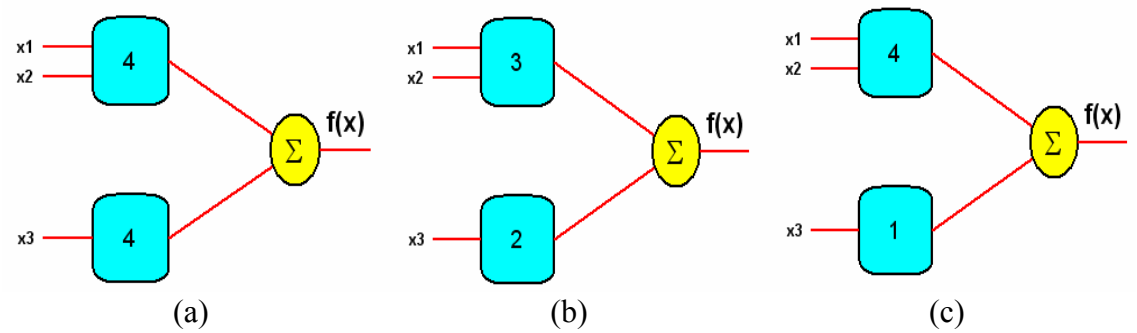
Fig. 6.30 ***a)*** Multi-RBFN Con 5% de ruido ***b)*** Multi-RBFN Con 10% de ruido ***c)*** Multi-RBFN Con 20% de ruido

- Función $f_4(x)$

  Los datos de test de la función $f_4(x)$ están formados por 10000 puntos distribuidos en el dominio de entrada de forma aleatoria.

Añadir 5%				**Añadir 10%**				**Añadir 20%**			
{ RBF }	**$NRMSE_{Tr}$**	**$NRMSE_{Test}$**	**DesV**	**{ RBF }**	**$NRMSE_{Tr}$**	**$NRMSE_{Test}$**	**DesV**	**{ RBF }**	**$NRMSE_{Tr}$**	**$NRMSE_{Test}$**	**DesV**
{2 1}	0.127	0.115	6E-3	{2 1}	0.180	0.126	5E-3	{2 1}	0.299	0.302	5E-3
{1 2}	0.290	0.300	7E-3	{1 2}	0.324	0.304	2E-2	{1 2}	0.400	0.403	5E-3
{3 1}	0.105	0.102	1E.-2	{3 1}	0.164	0.161	5E-2	{3 1}	0.289	0.293	6E-3
{2 2}	0.128	0.115	1E-2	{2 2}	0.179	0.083	7E-2	{2 2}	0.298	0.302	6E-3
{3 2}	0.096	0.086	8E-2	{4 1}	0.159	0.101	2E-2	{4 1}	0.288	0.292	6E-3
{2 3}	0.110	0.099	7E-2	{3 2}	0.159	0.078	8E-3	{3 2}	0.287	0.292	7E-3
{4 2}	0.096	0.084	1E-2	{4 2}	0.161	0.080	6E-3	{4 2}	0.288	0.292	6E-3
{3 3}	0.096	0.081	1E-2	{3 3}	0.158	0.081	4E-3	{3 3}	0.288	0.292	6E-3
{4 3}	0.093	0.074	3E-2					{4 3}	0.287	0.296	1E-2
{3 4}	0.089	0.079	7E-3					{3 4}	0.287	0.296	1E-2
{5 3}	0.101	0.083	3E-2					{5 3}	0.286	0.296	3E-3
{4 4}	0.082	0.065	2E-2					{4 4}	0.286	0.296	6E-3

Tabla 6.19: NRMSE de entrenamiento y test obtenido por el algoritmo propuesto con añadir 5%, 10% y 20% de ruido para la función $f_4(x)$



(a) (b) (c)

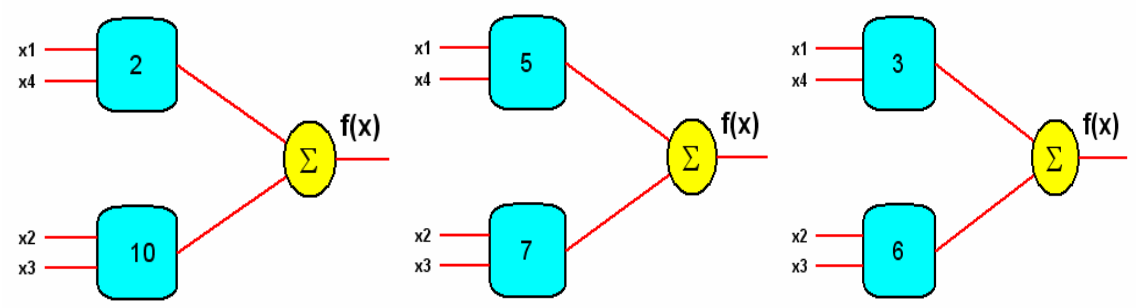
Fig. 6.31 ***a)*** Multi-RBFN Con 5% de ruido ***b)*** Multi-RBFN Con 10% de ruido ***c)*** Multi-RBFN Con 20% de ruido

- Función $f_8(x)$

  Los datos de test de la función $f_8(x)$ están formados por 20000 puntos distribuidos en el dominio de entrada de forma aleatoria.

Añadir 5%				**Añadir 10%**				**Añadir 20%**			
{ RBF }	**$NRMSE_{Tr}$**	**$NRMSE_{Test}$**	**DesV**	**{ RBF }**	**$NRMSE_{Tr}$**	**$NRMSE_{Test}$**	**DesV**	**{ RBF }**	**$NRMSE_{Tr}$**	**$NRMSE_{Test}$**	**DesV**
{2 1}	0.782	0.789	2E-3	{2 1}	0.785	0.796	8E-3	{2 1}	0.803	0.832	2E-3
{1 2}	0.568	0.573	7E-3	{1 2}	0.580	0.585	8E-3	{1 2}	0.607	0.642	6E-3
{2 2}	0.598	0.602	2E-2	{2 2}	0.456	0.408	2E-2	{2 2}	0.627	0.655	4E-2
{1 3}	0.531	0.520	1E-2	{1 3}	0.575	0.571	3E-2	{1 3}	0.604	0.631	5E-3
{2 3}	0.365	0.361	9E-3	{3 2}	0.412	0.412	1E-2	{2 3}	0.438	0.464	6-3
{1 4}	0.552	0.561	1E-2	{2 3}	0.357	0.365	2E-2	{1 4}	0.593	0.630	7E-3
{3 3}	0.364	0.372	7E-3	{3 3}	0.382	0.379	4E-2	{3 3}	0.437	0.464	5E-3
{2 4}	0.312	0.309	2E-2	{2 4}	0.342	0.340	2E-2	{2 4}	0.390	0.416	3E-3
{3 4}	0.301	0.293	4E-3	{3 4}	0.340	0.339	2E-2	{3 4}	0.398	0.423	1E-2
{2 5}	0.190	0.183	1E-2	{2 5}	0.303	0.343	1E-2	{2 5}	0.310	0.346	6E-3
{3 5}	0.185	0.185	8E-3	{4 4}	0.263	0.280	6E-3	{3 5}	0.312	0.347	5E-3
{2 6}	0.135	0.136	1E-2	{3 5}	0.196	0.192	6E-3	{2 6}	0.288	0.333	7E-3
{3 6}	0.138	0.139	1E-2	{4 5}	0.192	0.183	2E-3	{3 6}	0.290	0.335	6E-3
{2 7}	0.132	0.132	1E-2	{3 6}	0.176	0.158	4E-2	{2 7}	0.285	0.336	2E-3
{3 7}	0.133	0.135	8E-3	{4 6}	0.174	0.151	5E-2	{3 7}	0.284	0.340	7E-3
{2 8}	0.126	0.127	1E-2	{3 7}	0.1703	0.154	5E-2				
{3 8}	0.125	0.128	1E-2	{4 7}	0.176	0.114	8E-3				
{2 9}	0.118	0.114	2E-2	{2 8}	0.169	0.113	8E-3				
{3 9}	0.123	0.109	2E-2	{5 7}	0.168	0.103	8E-3				
{2 10}	0.113	0.101	2E-3								

Tabla 6.20: NRMSE de entrenamiento y test obtenido por el algoritmo propuesto con añadir 5%, 10% y 20% de ruido para la función $f_8(x)$

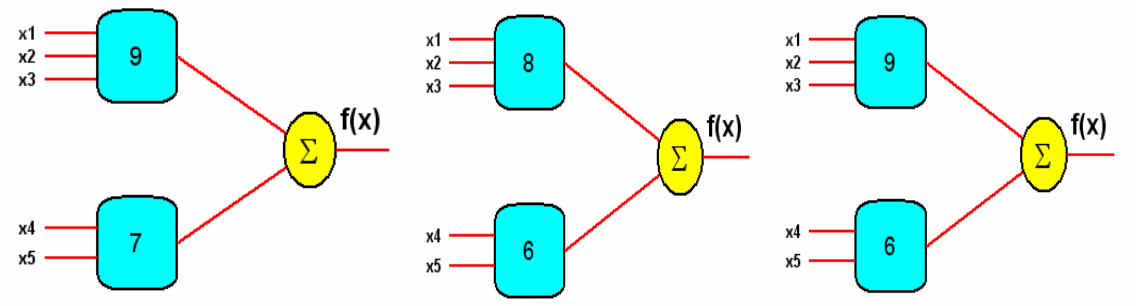


(a) (b) (c)

Fig. 6.32 ***a)*** Multi-RBFN Con 5% de ruido ***b)*** Multi-RBFN Con 10% de ruido ***c)*** Multi-RBFN Con 20% de ruido

- Función $f_9(x)$

  Los datos de test de la función $f_9(x)$ están formados por 20000 puntos distribuidos en el dominio de entrada de forma aleatoria.



(a) (b) (c)

Fig. 6.33 ***a)*** Multi-RBFN Con 5% de ruido ***b)*** Multi-RBFN Con 10% de ruido ***c)*** Multi-RBFN Con 20% de ruido

Añadir 5%				Añadir 10%				Añadir 20%			
{ RBF }	$NRMSE_{Tr}$	$NRMSE_{Test}$	DesV	{ RBF }	$NRMSE_{Tr}$	$NRMSE_{Test}$	DesV	{ RBF }	$NRMSE_{Tr}$	$NRMSE_{Test}$	DesV
{2 1}	0.606	0.796	8E-3	{2 1}	0.608	0.602	4E-3	{2 1}	0.666	0.658	4E-3
{1 2}	0.621	0.585	8E-3	{1 2}	0.614	0.614	6E-3	{1 2}	0.678	0.680	6E-3
{3 1}	0.519	0.408	2E-2	{3 1}	0.521	0.514	1E-3	{3 1}	0.593	0.581	2E-2
{2 2}	0.571	0.571	3E-2	{2 2}	0.564	0.565	3E-3	{2 2}	0.637	0.626	3E-3
{4 1}	0.435	0.442	1E-2	{4 1}	0.436	0.420	9E-3	{4 1}	0.544	0.530	1E-2
{3 2}	0.378	0.435	2E-2	{3 2}	0.437	0.415	2E-2	{3 2}	0.516	0.505	9E-3
{4 2}	0.331	0.419	4E-2	{5 1}	0.448	0.323	2E-2	{4 2}	0.467	0.456	4E-3
{3 3}	0.334	0.410	2E-2	{4 2}	0.340	0.375	2E-2	{3 3}	0.467	0.457	1E-2
{5 2}	0.314	0.389	2E-2	{5 2}	0.364	0.330	1E-2	{5 2}	0.456	0.452	1E-2
{4 3}	0.477	0.393	1E-2	{4 3}	0.356	0.299	3E-3	{4 3}	0.471	0.459	4E-2
{5 3}	0.302	0.380	6E-2	{5 3}	0.313	0.270	2E-2	{6 2}	0.456	0.448	1E-2
{4 4}	0.432	0.392	6E-2	{4 4}	0.371	0.301	6E-3	{5 3}	0.436	0.430	1E-2
{6 3}	0.302	0.383	2E-2	{6 3}	0.321	0.266	2E-2	{6 3}	0.455	0.448	1E-2
{5 4}	0.290	0.358	4E-2	{5 4}	0.303	0.286	2E-2	{5 4}	0.446	0.437	4E-2
{6 4}	0.294	0.351	5E-2	{6 4}	0.316	0.282	2E-2	{6 4}	0.442	0.435	2E-2
{5 5}	0.293	0.354	5E-2	{5 5}	0.308	0.280	1E-2	{5 5}	0.450	0.439	3E-2
{6 5}	0.263	0.314	8E-3	{6 5}	0.303	0.288	2E-2	{7 4}	0.430	0.421	5E-3
{5 6}	0.292	0.313	8E-3	{5 6}	0.315	0.259	2E-3	{6 5}	0.446	0.436	8E-3
{7 5}	0.253	0.302	8E-3	{7 5}	0.279	0.302	6E-3	{8 4}	0.411	0.404	6E-3
{6 6}	0.293	0.302	1E-2	{6 6}	0.311	0.250	1E-2	{7 5}	0.420	0.410	1E-2
{8 5}	0.248	0.296	1E-2	{8 5}	0.281	0.270	5E-3	{9 4}	0.394	0.387	1E-2
{7 6}	0.260	0.289	1E-2	{7 6}	0.286	0.241	4-E3	{8 5}	0.414	0.406	2E-2
{9 5}	0.227	0.273	2E-2	{9 5}	0.269	0.248	5E-3	{10 4}	0.389	0.387	3E-2
{8 6}	0.237	0.252	1E-2	{8 6}	0.267	0.246	7E-3	{9 5}	0.389	0.386	1E-2
{10 5}	0.216	0.272	5E-2					{10 5}	0.384	0.390	1E-2
{9 6}	0.197	0.235	3E-2					{9 6}	0.379	0.394	2E-2
{10 6}	0.174	0.278	2E-2								
{9 7}	0.248	0.279	1E-2								

Tabla 6.21: NRMSE de entrenamiento y test obtenido por el algoritmo propuesto con añadir 5%, 10% y 20% de ruido para la función $f_9(x)$

- Función $f_{11}(x)$

  Los datos de test de la función $f_{11}(x)$ están formados por 20000 puntos distribuidos en el dominio de entrada de forma aleatoria.

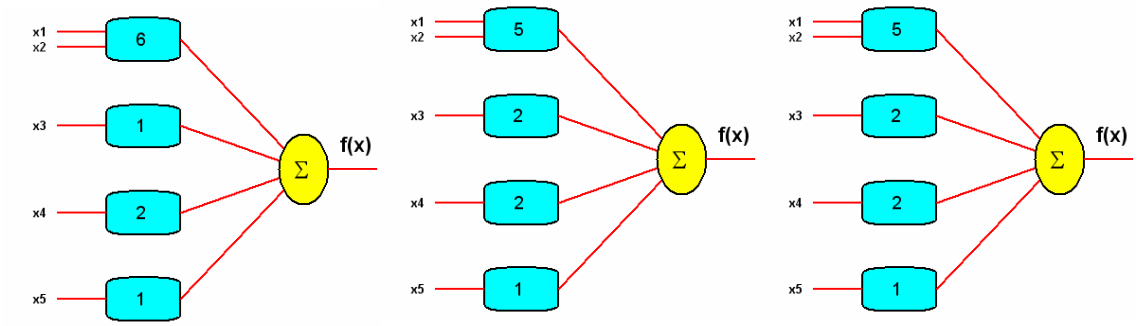


Fig. 6.34 ***a)*** Multi-RBFN Con 5% de ruido ***b)*** Multi-RBFN Con 10% de ruido ***c)*** Multi-RBFN Con 20% de ruido

Añadir 5%				**Añadir 10%**				**Añadir 20%**			
{ RBF }	**$NRMSE_{Tr}$**	**$NRMSE_{Test}$**	**DesV**	**{ RBF }**	**$NRMSE_{Tr}$**	**$NRMSE_{Test}$**	**DesV**	**{ RBF }**	**$NRMSE_{Tr}$**	**$NRMSE_{Test}$**	**DesV**
{2 1 1 1}	0.23	0.213	2E-3	{2 1 1 1}	0.256	0.241	3E-2	{2 1 1 1}	0.364	0.362	7E-3
{1 2 1 1}	0.279	0.254	4E-3	{1 2 1 1}	0.302	0.252	3E-3	{1 2 1 1}	0.390	0.389	7E-3
{1 1 2 1}	0.257	0.243	4E-3	{1 1 2 1}	0.293	0.249	5E-3	{1 1 2 1}	0.381	0.381	2E-3
{1 1 1 2}	0.261	0.246	4E-3	{1 1 1 2}	0.297	0.207	2E-2	{1 1 1 2}	0.382	0.382	6E-3
{3 1 1 1}	0.162	0.104	1E-2	{3 1 1 1}	0.181	0.122	3E-2	{3 1 1 1}	0.313	0.318	4E-3
{2 2 1 1}	0.247	0.230	6E-3	{2 2 1 1}	0.272	0.225	7E-3	{2 2 1 1}	0.378	0.376	2E-3
{2 1 2 1}	0.235	0.216	6E-3	{2 1 2 1}	0.249	0.205	2E-2	{2 1 2 1}	0.362	0.362	3E-3
{2 1 1 2}	0.235	0.218	4E-3	{2 1 1 2}	0.268	0.103	2E-2	{2 1 1 2}	0.361	0.360	4E-3
{4 1 1 1}	0.145	0.194	3E-2	{4 1 1 1}	0.177	0.138	7E-3	{4 1 1 1}	0.313	0.318	1E-3
{3 2 1 1}	0.154	0.191	1E-2	{3 2 1 1}	0.179	0.152	5E-2	{3 2 1 1}	0.312	0.317	1E-3
{3 1 2 1}	0.177	0.153	6E-2	{3 1 2 1}	0.177	0.185	1E-2	{3 1 2 1}	0.312	0.318	9E-3
{3 1 1 2}	0.208	0.178	1E-2	{3 1 1 2}	0.181	0.178	9E-3	{3 1 1 2}	0.312	0.318	7E-3
{5 1 1 1}	0.119	0.106	1E-2	{5 1 1 1}	0.187	0.177	3E-3	{4 2 1 1}	0.312	0.318	4E-3
{4 2 1 1}	0.172	0.085	2E-3	{4 2 1 1}	0.179	0.187	2E-3	{3 3 1 1}	0.312	0.317	4E-3
{4 1 2 1}	0.175	0.079	6E-3	{4 1 2 1}	0.176	0.169	9E-3	{3 2 2 1}	0.309	0.315	2E-3
{4 1 1 2}	0.141	0.082	7E-3	{4 1 1 2}	0.181	0.171	2E-2	{3 2 1 2}	0.312	0.317	1E-2
{6 1 1 1}	0.118	0.093	4E-3	{5 1 2 1}	0.178	0.160	7E-3	{4 2 2 1}	0.307	0.313	6E-3
{5 2 1 1}	0.161	0.082	7E-3	{4 2 2 1}	0.168	0.150	7E-2	{3 3 2 1}	0.309	0.315	4E-3
{5 1 2 1}	0.112	0.078	2E-2	{4 1 3 1}	0.175	0.146	2E-3	{3 2 3 1}	0.346	0.345	4E-3
{5 1 1 2}	0.144	0.073	2E-3	{4 1 2 2}	0.174	0.147	7E-3	{4 1 2 2}	0.309	0.315	9E-3
{6 1 2 1}	0.107	0.073	8E-3	{5 2 2 1}	0.174	0.147	3E-3	{5 2 2 1}	0.301	0.315	8E-3

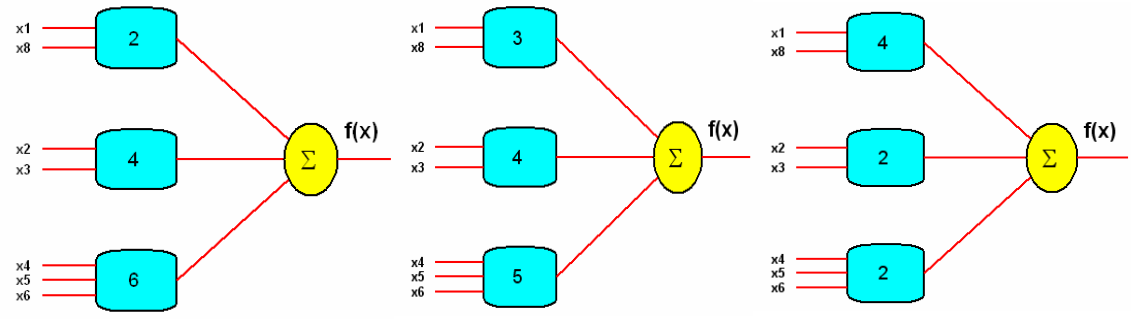
Tabla 6.22: NRMSE de entrenamiento y test obtenido por el algoritmo propuesto con añadir 5%, 10% y 20% de ruido para la función $f_{11}(x)$

- Función $f_{14}(x)$

  Los datos de test de la función $f_{14}(x)$ están formados por 30000 puntos distribuidos en el dominio de entrada de forma aleatoria

Añadir 5%				Añadir 10%				Añadir 20%			
{ RBF }	$NRMSE_{Tr}$	$NRMSE_{Test}$	DesV	{ RBF }	$NRMSE_{Tr}$	$NRMSE_{Test}$	DesV	{ RBF }	$NRMSE_{Tr}$	$NRMSE_{Test}$	DesV
{2 1 1}	0.260	0.437	3E-3	{2 1 1}	0.449	0.479	2E-3	{2 1 1}	0.534	0.555	4E-3
{1 2 1}	0.472	0.468	3E-3	{1 2 1}	0.482	0.511	1E-3	{1 2 1}	0.530	0.553	2E-3
{1 1 2}	0.513	0.512	1E-2	{1 1 2}	0.517	0.545	1E-2	{1 1 2}	0.591	0.610	1E-2
{3 1 1}	0.440	0.443	6E-3	{3 1 1}	0.443	0.473	9E-3	{2 2 1}	0.436	0.462	5E-3
{2 2 1}	0.279	0.283	2E-2	{2 2 1}	0.317	0.352	6E-2	{1 3 1}	0.542	0.564	1E-2
{2 1 2}	0.438	0.438	5E-3	{2 1 2}	0.451	0.481	3E-3	{1 2 2}	0.529	0.552	6E-3
{3 2 1}	0.285	0.288	3E-2	{3 2 1}	0.319	0.355	4E-2	{3 2 1}	0.430	0.458	8E-3
{2 3 1}	0.294	0.298	5E-3	{2 3 1}	0.313	0.350	9E-3	{2 3 1}	0.435	0.461	9E-3
{2 2 2}	0.276	0.280	6E-3	{2 2 2}	0.316	0.351	6E-3	{2 2 2}	0.436	0.462	9E-3
{3 2 2}	0.284	0.288	2E-2	{3 3 1}	0.332	0.367	1E-2	{4 2 1}	0.413	0.443	7E-3
{2 3 2}	0.274	0.278	4E-2	{2 4 1}	0.317	0.353	2E-2	{3 3 1}	0.445	0.471	3E-2
{2 2 3}	0.275	0.279	3E-3	{2 3 2}	0.313	0.350	8E-3	{3 2 2}	0.436	0.462	1E-2
{3 3 2}	0.274	0.279	4E-3	{3 3 2}	0.342	0.376	9E-3	{5 2 1}	0.370	0.406	1E-2
{2 4 2}	0.276	0.280	7E-3	{2 4 2}	0.316	0.351	6E-3	{4 3 1}	0.442	0.467	5E-3
{2 3 3}	0.273	0.276	3E-3	{2 3 3}	0.315	0.351	5E-3	{4 2 2}	0.431	0.467	2E-3
{3 3 3}	0.283	0.286	8E-3	{3 3 3}	0.320	0.356	1E-2				
{2 4 3}	0.275	0.279	2E-2	{2 4 3}	0.315	0.351	1E-2				
{2 3 4}	0.274	0.278	1E-2	{2 3 4}	0.314	0.350	1E-2				
{3 3 4}	0.289	0.293	9E-3	{3 3 4}	0.332	0.367	7E-3				
{2 4 4}	0.275	0.279	6E-3	{2 4 4}	0.315	0.351	6E-3				
{2 3 5}	0.274	0.277	4E-3	{2 3 5}	0.316	0.352	9E-3				
{3 3 5}	0.290	0.294	7E-3	{3 4 4}	0.308	0.345	7E-3				
{2 4 5}	0.275	0.279	3E-2	{2 5 4}	0.310	0.347	1E-2				
{2 3 6}	0.280	0.284	4E-3	{2 4 5}	0.315	0.351	2E-2				
{3 4 5}	0.271	0.275	1E-2	{4 4 4}	0.288	0.328	2E-2				
{2 5 5}	0.271	0.275	5E-3	{3 5 4}	0.308	0.345	1E-2				
{2 4 6}	0.270	0.278	2E-2	{3 4 5}	0.306	0.345	1E-2				

Tabla 6.23: NRMSE de entrenamiento y test obtenido por el algoritmo propuesto con añadir 5%, 10% y 20% de ruido para la función $f_{14}(x)$



(a) (b) (c)

Fig. 6.35 ***a)*** Multi-RBFN Con 5% de ruido ***b)*** Multi-RBFN Con 10% de ruido ***c)*** Multi-RBFN Con 20% de ruido

6.10 Conclusiones

Este capítulo ha presentado los resultados obtenidos por el algoritmo del sistema jerárquico Multi.-RBFN propuesto al aproximar funciones a partir de un conjunto de muestras de entrada y salida. Los resultados obtenidos para seleccionar la arquitectura adecuada y aproximar funciones a partir de ejemplos de E/S se comparan con una red de funciones de base radial clásica que recibe todas las variables de la función que pretende aproximar. Las comparativas revelan que el algoritmo propuesto encuentra mejores aproximaciones, tanto en error de aproximación como en la complejidad (el número de parámetros para definir el problema). La superioridad de los resultados se debe a las distintas características de algoritmo propuesto:

- Las variables de entrada irrelevantes se eliminan en un preproceso antes de optimizar los parámetros del sistema Multi-RBFN lo que produce menor número de elementos computacionales y más eficacia en el proceso de aprendizaje.

- El proceso de decidir cuáles de las variables seleccionadas deben ir solas o juntas a una Sub-RBFN, permite la construcción de un sistema Multi-RBFN que evita un incremento exponencial en la complejidad del sistema.

- El proceso de inicializar los valores de los centros en cada Sub-RBFN, se obtiene por un propuesto algoritmo de clustering que mejora la inicialización con respecto a otros algoritmos de clustering tradicionales.

- Los pesos de la salida de la red se calculan siempre de forma óptima, dependiendo de la salida total del sistema Multi-RBFN.

- La aplicación de un algoritmo de minimización local del error a partir de la configuración inicial que proporciona algoritmo Multi-RBFN permite ajustar los parámetros de cada Sub-RBFN hasta llegar al óptimo local más cercano, que suponemos será el óptimo global.

Por otra parte, se ha comprobado la capacidad del algoritmo para tratar con la adición de un cierto valor de ruido blanco.

De los resultados obtenidos se puede observar que el error de aproximación del conjunto de funciones utilizadas es menor del error obtenido por una RBFN clásica que recibe todas las entradas. Esto debido al decrecimiento de los parámetros utilizados en el algoritmo propuesto. Y como es de esperar, este decrecimiento en el número de parámetros no solo influye en el grado de aproximación alcanzado y en la mejora de la capacidad de generalización de la red ante datos no aprendidos, sino que el tiempo de ejecución total también decrece.

CAPÍTULO 7

CONCLUSIONES

El trabajo de tesis doctoral presentado en esta memoria ha presentado un método computacional formado principalmente por redes de funciones de base radial RBFNs capaz de modelar sistemas complejos de aproximación funcional sin que el aumento del número de variables de entrada tenga que suponer un incremento exponencial en la complejidad del sistema.

Se ha presentado un algoritmo de *clustering* para aproximación de funciones derivado del algoritmo de *clustering* para aproximación funcional *CFA* y el algoritmo de *enhanced LBG*. El algoritmo propuesto trata de aumentar el número de clusters en la zona del espacio de la entrada donde el error de aproximación de la red es mayor, teniendo en cuenta el grado de responsabilidad de cada cluster en este error. El objetivo final es intentar igualar la distribución del error de aproximación a lo largo de cada cluster, con el fin de que esta configuración sea adecuada para abordar un proceso de optimización local posterior. Si un algoritmo de *clustering* puede obtener la igualdad en la distorsión total, significa que cada cluster ha encontrado su lugar correcto en el espacio de entrada y la distribución de los clusters es la mejor.

Esta memoria también ha presentado un sistema jerárquico Multi-RBFN formado por redes de funciones de base radial RBFNs, que aquí hemos denominado Sub-RBFN, con la propiedad de que cada Sub-RBFN se encarga de un conjunto de variables de entrada y no del conjunto completo. De tal forma que se pueda reducir en gran medida, el número de parámetros necesarios para definir esta estructura jerárquica Multi-RBFN.

Se ha presentado un algoritmo que sea capaz de encontrar automáticamente la topología adecuada del sistema jerárquico Multi-RBFN y optimizar sus parámetros para modelar sistemas para resolver el problema de la aproximación funcional a partir de un conjunto de muestras de E/S. Las aplicaciones de esta técnica a algunos problemas de aproximación han sido presentadas.

Para llevar acabo esta tarea, se ha presentado un nuevo método para seleccionar las variables de entrada más importantes (IVS) que decide también cuáles de estas variables deben ir solas o juntas en cada Sub-RBFN. Para tal fin, se ha propuesto un algoritmo capaz de encontrar automáticamente la topología adecuada del sistema jerárquico Multi-RBFN propuesto y optimizar los parámetros del sistema a partir de un conjunto de muestras de E/S de la función a aproximar. Las principales aportaciones y conclusiones obtenidas se resumen a continuación.

1. Se ha presentado un resumen general del desarrollo realizado en esta memoria. Este desarrollo consiste en encontrar nuevas arquitecturas de cómputo capaces de modelar sistemas complejos de aproximación de funciones, sin que el aumento del número de variables de entrada tenga que suponer un incremento exponencial en la complejidad del sistema. Básicamente todos los algoritmos de entrenamiento para RBFNs tratan de encontrar modelos que minimizan dos objetivos: el error de aproximación y la complejidad estructural de la red. El sistema Multi-RBFN permite decrementar el número de elementos computacionales lo que posibilita mejores resultados en el error de aproximación usando una menor complejidad estructural.

2. Se ha analizado la metodología clásica para la creación de redes de funciones de base radiales RBFNs que son el principal elemento computacional del presente

trabajo. Esta metodología consiste básicamente en inicializar los valores de los centros y radios de las funciones base, calcular de forma exacta los pesos de la salida de la red y aplicar algoritmos de minimización del error hasta encontrar el mínimo local más cercano a la configuración inicial. Para cada una de estas tares se han estudiado y analizado las metodologías más utilizadas.

- En el caso de la inicialización de los centros de las funciones base, se ha presentado un nuevo algoritmo de *clustering* para problemas de aproximación de funciones denominada ECFA. Hemos demostrado cómo este algoritmo mejora las inicializaciones de los centros realizado por algoritmos tradicionales.

- Para la inicialización de las amplitudes de cada función base radial, se ha utilizado un método tradicional denominado el método de los vecinos más cercanos (knn). Este método permite que el solapamiento entre las neuronas ocultas sea lo más suave posible.

- Para el cálculo exacto de los pesos de la red, se han analizado distintas métodos de resolución de ecuaciones lineales, entre ellos la descomposición de Cholesky, el método de OLS y el método de SVD. Se ha decidido utilizar el método SVD a largo de este trabajo por su capacidad de facilitar una solución para cualquier sistema de ecuaciones, con la que se obtiene una reducción del error en la salida de la red.

- Para la aplicación de algoritmos de minimización del error, se ha presentado distintos métodos. La inicialización de los parámetros de la RBFN consiste en la dirección adecuada del óptimo global, pero todavía está a una cierta distancia de alcanzarlo. Los algoritmos de minimización del error pueden acabar atrapados en un mínimo local no adecuado, según la configuración de la que partan. El algoritmo que se han utilizado en esta memoria es el de *Levenberg-Marquardt* porque es fácil de implementar e incorpora información sobre la segunda derivada

del error para acelerar la búsqueda cuando la configuración de la red se encuentra cerca del mínimo.

3. Se han analizado los algoritmos de clustering más utilizados. La comparación entre estos algoritmos ha producido la creación de un nuevo método de clustering diseñado para problemas de aproximación de funciones. El algoritmo propuesto trata de aumentar el número de clusters en la zona del espacio de entrada donde el error de aproximación teniendo en cuenta la verdadera salida de la red es mayor, intentando igualar el valor de este error en cada cluster, lo que produce una distribución adecuada de los clusters en el espacio de entrada. Con esta igualdad del error, la distorsión será igual para todos los clusters en el espacio de los datos de entrada, es decir, que cada cluster está en su lugar adecuado y cubre su zona de los datos de entrada. El algoritmo es rápido, con pequeña complejidad computacional y simple de implementar. El algoritmo que se ha presentado puede converger con un número de clusters menor que el de otros algoritmos aplicados al mismo problema. Se puede utilizar este algoritmo para encontrar el número mínimo de centros que satisfacen un problema dado de aproximación de funciones.

4. Se ha propuesto un algoritmo que se concentra en la búsqueda de nuevas arquitecturas de cálculo capaces de formar sistemas complejos de RBFNs sin que el aumento del número de variables de entrada tenga que suponer un aumento exponencial en la complejidad del sistema. Nuestro objetivo es encontrar un modelo con el número más pequeño de variables de entrada que tienen según las estadísticas, o prácticamente al menos, la misma utilidad esperada con el modelo completo con todas las entradas disponibles.

    - Se ha propuesto una metodología eficaz para la estimación de las variables de entrada más importantes; este método trata de relacionar cada dimensión de los datos de entrada con la salida objetivo y divide los datos de esta dimensión en partes. Para cada una de estas partes se calcula la distancia entre el máximo y el mínimo de los valores de la salida que pertenecen a los datos de cada dimensión en cada parte y el promedio de todas las

distancias en todas las partes. Cuando el promedio tiene un valor pequeño, la variable es más importante y tiene que ser seleccionada. Las variables que tienen promedio grande son variables de ruido que no afectan mucho en la función, y deberían ser eliminadas. Dicha metodología también es capaz de seleccionar, entre las variables de entrada más importantes, cuáles van solas o juntas en cada Sub-RBFN. Este proceso depende, en general, del cálculo de la varianza de cada variable o conjunto de variables relacionada con la salida objetivo.

- Una vez obtenida la estructura jerárquica Multi-RBFN, cada Sub-RBFN, es, a su vez, una red de funciones de pase radial (RBFN). En esta memoria se ha presentado también un algoritmo para optimizar los parámetros de cada Sub-RBFN y, por tanto, de la estructura global Multi-RBFN, basado en el algoritmo de clustering ECFA presentado en capítulos anteriores, para inicializar los valores de los centros en cada Sub-RBFN. Para optimizar los valores de los radios en cada Sub-RBFN, se utilizan algoritmos tradicionales como el método de los vecinos más cercanos (*knn*). Cuando los parámetros de centros y radios de cada Sub-RBFN han sido inicializados, se utiliza un método de cálculo lineal para encontrar los valores exactos de los pesos en todo el sistema jerárquico Multi-RBFN que minimiza la función de coste calculada sobre el conjunto de muestras de E/S. Este método se repite de forma incremental, hasta obtener la estructura jerárquica final y el valor de todos sus parámetros.

La superioridad de los resultados se debe a las distintas características de algoritmo propuesto:

- Las variables de entrada irrelevantes se eliminan en un preproceso antes de optimizar los parámetros del sistema Multi-RBFN lo que produce menor número de elementos computacionales y más eficacia en el proceso de aprendizaje.

- El proceso de decidir cuáles de las variables seleccionadas deben ir solas o juntas a una Sub-RBFN, permite la construcción de un sistema Multi-RBFN que evita un incremento exponencial en la complejidad del sistema.

- El proceso de inicializar los valores de los centros en cada Sub-RBFN, se obtiene por un propuesto algoritmo de clustering que mejora la inicialización con respecto a otros algoritmos de clustering tradicionales.

- Los pesos de la salida de la red se calculan siempre de forma óptima, dependiendo de la salida total del sistema Multi-RBFN.

- La aplicación de un algoritmo de minimización local del error a partir de la configuración inicial que proporciona algoritmo Multi-RBFN permite ajustar los parámetros de cada Sub-RBFN hasta llegar al óptimo local más cercano, que suponemos será el óptimo global.

De los resultados obtenidos se puede observar que el error de aproximación del conjunto de funciones utilizadas es menor del error obtenido por una RBFN clásica que recibe todas las entradas. Esto debido al decrecimiento de los parámetros utilizados en el algoritmo propuesto. Y como es de esperar, este decrecimiento en el número de parámetros no solo influye en el grado de aproximación alcanzado y en la mejora de la capacidad de generalización de la red ante datos no aprendidos, sino que el tiempo de ejecución total también decrece.

Finalmente, como continuación de este trabajo, intentaremos considerar la extensión difusa del algoritmo ECFA, en el que cada dato de entrada pueda pertenecer a más de un cluster a la vez, dando lugar este grado de activación a una función de pertenencia propia de un conjunto difuso. Esta extensión aumentaría la complejidad computacional del algoritmo en gran medida aunque es posible que sea compensado por una mejora en los resultados finales, tras el proceso de minimización local. También se intentará estudiar la extensión del

sistema Multi-RBFN a un sistema en el que el nivel superior de la jerarquía sea, a su vez, una nueva red RBFN. Esto complicaría enormemente el tratamiento matemático del sistema global, pero puede que se pueda abordar el problema utilizando herramientas más costosas como son los algoritmos evolutivos.

Bibliografía

[AWA-05a] M.Awad, H.Pomares, F.Rojas, L.J. Herrera, J.González, A. Guillén. "Approximating I/O data using Radial Basis Functions:A new clustering-based approach". IWANN 2005, LNCS 3512, pp. 289 – 296, 2005.© Springer-Verlag Berlin Heidelberg 2005.

[AWA-05b] M.Awad, H.Pomares, I.Rojas, L.J. Herrera, A. Prieto. "Input Variable Selection in Hierarchical RBF Networks". IWANN 2005, LNCS 3512, pp. 280-288, 2005.© Springer-Verlag Berlin Heidelberg 2005.

[BAC-99] A.D.Back and Th.P.Trappenberg. "Input Variable Selection Using Independent component analysis". International Conference on Neural Networks, Washington, 1999.

[BAT-94] R. Battiti, "Using mutual information for selecting features in supervised neural net learning". IEEE Trans. Neural Networks, vol. 5, no. 4, July 1994.

[BEL-61] R.Bellman. "Adaptive Control Processes: A Guided Tour". Princeton University Press. 1961.

[BEN-00] S.Bengio and Y.Bengio. "Taking on the Curse of Dimensionality in Joint Distributions Using Neural Networks" IEEE TRANSACTIONS ON NEURAL NETWORKS, VOL. 11, NO. 3, MAY 2000.

[BEN-02] N.Benoudjit, C.Archambeau, A.Lendasse2, J. Lee, M. Verleysen1. "Width optimization of the Gaussian kernels in Radial Basis Function Networks". ESANN'2002 proceedings - European Symposium on Artificial Neural Networks Bruges (Belgium), d-side publi., ISBN 2-930307- 02-1, pp. 425-432. April 2002.

[BER-92] H.R.Berenji, P.Khedkar, "Learning and tuning fuzzy logic controllers through reinforcements", IEEE Trans. on Neural Networks, Vol.3, No.5, pp. 724-739,1992.

[BEZ-81] J.C.Bezdek, "Pattern Recognition with Fuzzy Objective Function Algorithms", Plenum Press, New York, 1981.

[BEZ-84] J.C.Bezdek, Ehrlich, R., Full, W. "FCM: The Fuzzy cc-means Clustering Algorithm". Computers and Geosciences, 10(2-3), pp.191-203. 1984.

[BLU-92] A.L.Blum and R.L.Rivest. "Training a 3-node neural network is NP-complete. Neural Networks". 5: pp. 117–127, 1992.

[BRO-88] D.S. Broomhead, D.Lowe. "Multivariable Functional Interpolation andAdaptive Networks," Complex Systems, vol. 2, pp. 321-355, 1988.

[CAR-95] B. Carse, A.Pipe, T.Fogarty, T.Hill. “Evolving Radial Basis Function Neural Networks using a Genetic algorithm”. Procs. IEEE Int. Conf. on Evolutionary Computation, Sydney, Australia.vol.1, pp. 300-305, 1995.

[CHA-96] S.V.Chakaravathy and J.Ghosh. “Scale based clustering using a radial basis function network” IEEE Transactions on Neural Networks, 2(5): pp.1250-61, Sept 1996.

[CHE-03] Y. Chen, J. Z Wang.: Kernel machines and additive fuzzy systems: classification and function approximation. pp. 789-795 vol.2. 2003

[CHE-91a] S. Chen, C.F.N.Cowan, and P.M.Grant. “Orthogonal least squares learning for radial basis function networks”. *IEEE Transactions on Neural Networks*, 2(2): pp.302-309, 1991.

[CHE-91b] V. Cherkassky. and H. Lay-Najafy., "Constrained Topological Mapping for Nonparametric Regression Analysis", Neural Networks, vol. 4, pp. 2740,1991.

[CHE-96] V. Cherkassky, D. Gehring, and F. Mulier, Comparison of adaptive methods for function estimation from samples, IEEE Trans. NN 7, 969-984,1996

[CLI-83] L. Cliff. “Visualization of Matrix Singular Value Decomposition”. Mathematics Magazine, pp.161-167. 1983.

[COH-99] I. Cohen Q. T. Xiang S. Zhou Th. S.Huang. “Feature Selection Using Principal Feature Analysis” Beckman Institute for Advanced Science and Technology University of Illinois at Urbana-Champaign, Urbana, IL 61801, 1999.

[COS-96] P.Cosman, R.M.Gray and M.Vetterli. “Vector Quantization of Image Subbands: A Survey.” IEEE Trans. Im. Proc., Vol. 5, No. 2, pp. 202-225, Feb. 1996.

[COV-91] T.M.Cover, and J.A.Thomas. “Elements of Information Theory”. John Wiley & Sons, 1991.

[CRI-00] P.Cristea, R.Tuduce, and A.Cristea. “Time Series Prediction with Wavelet Neural Networks”. Proceedings of IEEE Neural Network Applications in Electrical Engineering, pp. 5-10. 2000.

[DEL-95] B.Delyon, A.Juditsky, and A.Benveniste. “Accuracy Analysis for Wavelet Approximations”. IEEE Trans. Neural Networs, Vol. 6, pp. 332-348, 1995.

[DEN-83] J.Dennis, and R. Schnabel. “Numerical methods for unconstrained optimization and nonlinear Equations”. Englewood Cliffs, NJ. 1983.

[DIC-96] J. A. Dickerson, B. Kosko, "Fuzzy Function Approximation with

Ellipsoidal Rules," IEEE Transactions on Systems, Man, and Cybernetics, Volume 26, Number 4, pp. 542-560, August 1996

[DOA-92] J. Doak. “An evaluation of feature selection methods and their application to computer security”. Technical Report CSE-92-18, Department of Computer Science and Engineering, University of Carlifornia, 1992.

[DUD-73] R.O.Duda and P.E. Hart. “Pattern Classification and Scene Analysis”. New York: pp. 10-43. Wiley .1973.

[FAN-61] R.Fano. “Transmission of information: A statistical theory of communications”. New York: Wiley, 1961.

[FER-04] S. Ferrari, M. Maggioni,. N.A Borghese,.: Multiscale approximation withhierarchical radial basis functions networks. IEEE Trans. Neur. Net., vol.15, no.1, pp.178-188, 2004.

[FLE-64] R. Fletcher., and C.M.Reeves. “Function minimization by conjugate gradients”, Computer Journal vol. 7. pp. 149-154. 1964.

[FON-95] C.M.Fonseca, and P.J.Fleming: “An overview of evolutionary algorithms in multi-objective optimization”. Evolutionary Computation. 1995.

[FRA-92] W. Frawley, G. Piatetsky-Shapiro, and C.Matheus. “Knowledge discovery in databases: An overview”. AI Magazine, pp. 213-228, 1992.

[FRI-81] J.H. Friedman. and W. Stuetzle, "Projection pursuit regression", Journal of the American Statistics Association, Vol. 76, No. 376, pp. 817-823, 1981.

[FRI-91] J.H. Friedman, Multivariate Adaptive Regression Splines, Annals of Statistics, Vol 19, 1-141, 1991.

[FUK-90] K.FUKUNAGA. “*Introduction to Statistical Pattern Recognition”*. Academic Press. 1990.

[FUK-99] K .Fukumizu, S-I. Amari. Local Minima and Plateaus in Hierarchical Structures of Multilayer Perceptrons. Brain Science Institute. The Institute of Physical and Chemical Research (RIKEN) October 22, 1999.

[FUN-01] G.Fung. “A Comprehensive Overview of Basic Clustering Algorithms “. June 22, 2001.

[GER-79] A.Gersho. “Asymptotically Optimal Block Quantization”. IEEE Transaction Information Theory, IT-25 (4), pp.373-380. 1979.

[GER-92] A.GERSHO AND R.M. “Vector Quantization and Signal Compression”.

Kluwer Academic Publishers. 1992.

[GOL-89] G.H.Golub, C.F.V.Loan. “Matrix Computations”. The Johns Hopkins University Press, Baltimore. 1989.

[GOL-96] G.Golub, G.C.van loan. “Matriz computations”. Johns Hopkins university Press, Baltimore, 3rd ed. 1996.

[GON-01] J. Gonzalez. “Identificación y Optimización de redes de Funciones de Base Radiales Para Aproximación funcional”. Tesis Doctoral, Universidad de Granada. 2001.

[GON-02]: J.Gonzalez, I.Rojas, H.Pomares, J.Ortega, A.Prieto. “A new clustering technique for function approximation.” Neural Networks, IEEE .Transactions on, Volume: 13 Issue: Page(s): 132 -142. 1, Jan. 2002.

[GRA-98] T.Graepel. “Statistical physics of clustering algorithms”. Técnica port 171822, FB Physik, Institut fur Theoretische Physic, 1998.

[GRI-84] A. Griffiths, L. A. Robinson, and P. Willett. “Hierarchical agglomerative clustering methods for automatic document classification”. Journal of Documentation, 40(3): pp. 175-205, September 1984.

[GRU-93] F.Gruau. “Genetic Synthesis of Modular Neural Networks”. In S. Forrest, (eds), Genetic Algorithms: Proc. of the 5th International Conference, M. Kaufman, 1993.

[GUR-00] M.T.Gurrea. “ANÁLISIS DE COMPONENTES PRINCIPALES”. Proyecto e- Math Financiado por la Secretaría de Estado de Educación y Universidades (MECD). 2000.

[HAL-99] M.A.Hall. “Correlation-Based Feature Selection for Machine Learning”. PhD thesis, Department of Computer Science, University of Waikato, Hamilton, NewZealand, Apr. 1999.

[HAR-75] J.Hartigan. “*Clustering Algorithms”*. Wiley, New York, NY. 1975

[HAR-90] E.J.Hartman,. j.D.Keeler, and j.M. Kowalski. “Layered Neural Networks with Gaussian Hidden Units as Universal Aproximadores”. Neural Computation.1990.

[HAY-99] S.Haykin. “Neural Networks a Comprehensive Foundation”. New Jersey, Prentice Hall, 1999.

[HER-03] L.J.Herrera, H.Pomares, I.Rojas, O.Valenzuela, M.Awad. “MultiGrid-Based Fuzzy systems for function approximation”. University of Garanada. 2003.

[HER-05a] L.J. Herrera., H. Pomares, I. Rojas, A. Guilén, J.González, and M. Awad

"Clustering-Based TSK Neuro-Fuzzy Model for Function Approximation with Interpretable Sub-models" IWANN 2005, LNCS 3512.© Springer-Verlag Berlin Heidelberg. 2005.

[HER-05b] Herrera.L.J, Pomares.H, Rojas.I, Guilén. A, Awad .M "Analysis of the TaSe-II TSK-type Fuzzy System for function approximation " ECSQARU 2005.

[HOG-91] Z. Hong, "Algebraic feature extraction of image for recognition". Pattern Recognition, Vol.24, pp. 211-219, 1991.

[HOL-75] J. Holland. "Adaptation in Natural and Artificial Systems". University of Michigan Press, 1975.

[JAI-82] A.K. Jain and R .Chandrasekaran. "Dimensionality and sample size consideration in pattern recognition practice". In P.R. Krishaniah and L.N. Kanal, editors, Handbook of Statistics, volume II, pages 835–855. North-Holland, Amsterdam, The Netherlands, 1982.

[JAI-99] A. K. Jain, M. N. Murty, and P. J. Flynn. "Data clustering: a review". *ACM Computing Surveys*, 31(3):264-323, 1999.

[JAN-93] J.R. Jang, ANFIS: Adaptive-network, based fuzzy inference system. IEEE transaction on Systems, Man, and Cybernetics, 23, 3, 665-685. 1993.

[JI-05] Y.Ji, J. Hao, N. Reyhani and A. Lendasse. Direct and Recursive Prediction of Time Series Using Mutual Information Selection". Neural Network Research Centre, Helsinki University of Technology, 2005.

[JOH-94] G.H.John, R.Kohavi, and K.Pfleger. "Irrelevant features and the subset selection problem". In Proceedings of the Eleventh International Conference on Machine learning, pp. 121-129, New Brunswick, NJ, 1994.

[JOL-86] I.T.Joliffe. "Principal Component Analysis", New York: Springer-Verlag, 1986.

[KAR-95] N. B. Karayiannis, P.Pai. "Fuzzy Vector Quantization Algorithms and Their Application in Image Compression". IEEE Transactions on Image Processing, 4(9), pp.1193-1201. 1995.

[KAR-97] N.B.Karayiannis and G.W.Mi. "Growing radial basis neural networks: Merging supervised and unsupervised learning with network growth techniques". IEEE Trans. Neural Networks, vol. 8, pp. 1492-1506, Nov.1997.

[KEC-01] V.kecman., “Learning and Soft Computing: support vector Machines, Neural Networks, Fuzzy Logic models”. ‘A Bradford Book’. pp. 313-363. 2001.

[KIT-87] J. Kittler. “Feature set search algorithm”. In C.H. Chen, editor, Pattern Recognition and Signal Processing, pp. 41-60. Sithof and Noordho, Alphenaan den Rjin, The Netherlands, 1987.

[KLE-80] V.C.Klema. “The Singular Value Decomposition: Its Computation and Some Applications”. IEEE Trans. Automatic Control, Vol. 25, pp164-176, 1980.

[KOH-95] R.Kohavi. “Wrappers for Performance Enhancement and Oblivious Decision Graphs”. PhD thesis, Stanford University, 1995.

[KOH-96] R. Kohavi and G. John. “Wrappers for feature subset selection”. Artificial Intelligence, special issue on relevance, 97 (1–2): pp.273–324, 1996.

[KOH-97] R.Kohavi and G. John. Wrappers for feature subset selection. Artificial Intelligence, 97(1- 2): pp. 273-324, 1997

[KRI-93] P. S.Krishnaprasad and Y.C.Pati. “Analysis and Synthesis of Feed forward Neural Networks Using Discrete Affine Wavelet Transformations.” IEEE Trans. on Neural Networks, vol. 4, no. 1, 1993.

[KWA-93] S. C.Kwasny. B. L.Kalman. N. “Chang Distributed Patterns as HierarchicalStructures”. World Congress on Neural Networks-Portland, OR. 1993

[LEI-04] A. Leino. “Independent Component Analysis: An Overview”. 26th April 2004.

[LEN-03] A.Lendasse, J.Lee, E.Bodt, V.Wertz, M.Verleysen. “APPROXIMATION BY RADIAL BASIS FUNCTION NETWORKS”. Application to Option Pricing. Université catholique de Louvain. 2003.

[LEO-92] J.A. Leonard, M.A. Kramer and L.H. Ungar, Using Radial Basis Functions to Approximate a Function and Its Error Bounds, IEEE Trans. on Neural Networks. 1992.

[LIG-92] W.A.Light. “Some aspects of Radial Basis Function approximation. Approximation Theory, Spline Functions and Applications”. pp.356:163.190, 1992.

[LIN-93] C.T.Lin, C.S.G.Lee. “Reinfrocement structure/parameter learning for neural-network-based fuzzy logic control systems”. Proc of 1993 IEEE Int'l Conf. on Fuzzy Systems. pp.88-93. 1993.

[LIN-99] C.T.Lin, I.F.Chung, “A Reinforcement Neuro-Fuzzy Combiner for

Multi-objective Control". IEEE Trans. Syst., Man and Cyber-Part B-Cybernetics, vol.29, no.6, pp. 726-744, 1999.

[LOW-90] D.Lowe, and A.R.Webb, "Exploiting Prior Knowledge in Network Optimization: An Illustration from Medical Prognosis". Network I, pp. 299-323, 1990.

[MAC-67] J.B.MacQueen. "Some Methods for classification and Analysis of Multivariate Observations". *Proceedings of 5-th Berkeley Symposium on Mathematical Statistics and Probability"*, Berkeley, University of California. Press, 1: pp.281-297, 1967.

[MAC-77] Mackey, M. C. Glass, L. "Oscillation and chaos in physiological control systems," Science 197, pp. 287-289. 1977.

[MAR-63] D. W. Marquardt. "An algorithm for least-squares estimation of non-linear parameters". *SIAM, Journal on Applied mathematics*, vol. 11, n° 2, pp. 431-441. 1963.

[MIC-99] Z.Michalewickz. "Genetic Algorithms + Data Structures = Evolution Programs". Springer-Verlag. 1999.

[MOC-71] M.Mocciardi. "A comparison of seven techniques of choosing subsets of pattern recognition". IEEE Trans. Computers, C-20: pp.1023–1031, 1971.

[MOO-89] J.E.Moody and C.Darken. "Fast learning in networks of locally tuned processing units. Neural Computation". 2(1):281.294, 1989.

[MUS-92] W. Musavi, W. Ahmed, KH Chan, KB Faris , DM Hummels, On the training of radial basis function classifiers, Neural Networks, v.5 n.4, p.595-603, July- Aug. 1992.

[NAR-77] P.Narendra and K.Fukunaga. "A branch and bound algorithm for feature subset selection". IEEE Trans. Computer, C-26(9): pp.917-922, 1977.

[NIE-96] J. H. Nie and T. H. Lee, "Rule-based modeling: Fast construction and optimal manipulation," IEEE Trans. Syst., Man, Cybern. A, vol. 26, pp. 728-738, Nov. 1996.

[NIR-90] M.Niranjan. and F.Fallside. "Neural Networks and Radial Basis Functions in Classifying Static Speech Patterns". Computer Speech and Language 4, pp. 275- 289, 1990.

[ONN-01] V.Onnia. M.Tico. J.Saarinen. "Feature Selection Method Using Neural Network". IEEE. 0: 7803-6725. 2001.

[ORR-95] M.J.L. Orr. "Introduction to Radial Basis Function Networks". Centre for Cognitive Science - University of Edinburgh, 1995.

[PAL-93] K.K Paliwal. And B. S Atal. "Efficient vector quantization of LPC

parameters at 24 bits/frame". IEEE Transactions on speech and audio processing vol.1, no. 1, pp.3-14. 1993.

[PAR-91] J.Park and I.Sandberg. "Universal approximation using radial-basis-function networks". *Neural Computation*. 3(2):246--257, 1991.

[PAR-93] J.Park and I.Sandberg. "Universal approximation using radial-basis-function networks. *Neural Computation*, 5(2): pp.305-316, 1993.

[PED-96] W.Pedrycz. "Conditional fuzzy C-means," Pattern Recognition Lett., vol. 17. pp. 625–632, 1996.

[PED-98] W.Pedrycz. "Conditional fuzzy clustering in the design of radial basis function neural networks". IEEE Trans. Neural Networks, vol. 9, pp. 601–612, 1998.

[POG-87] T.Poggio, F.Girosi. "Networks for approximation and learning". Proceedings of IEEE 78. pp. 1481-1497. 1987

[POM-00] H.Pomares. "Nueva metodología para el diseño automático de sistemas difusos". Tesis Doctoral, universidad de granada, 2000.

[POM-02] H.Pomares.,I.Rojas., J. González., A. Prieto.: Structure Identification in Complete Rule- Based Fuzzy Systems. IEEE Trans. On fuzzy systems, vol. 10, No. 3, June 2002

[POW-77] M.J.D.Powell. "Restart Procedures for the Conjugate Gradient Method". Mathematical Programming, vol. 12, pp. 241-254, 1977.

[POW-84] M.J.D.Powell. "Nonconvex Minimization Calculations and the Conjugate Gradient Method". Lecture Notes in Mathematics, Vol. 1066, pp. 122-141. 1984.

[POW-87] M.Powell. "Radial basis functions for multivariable interpolation: A review". J.C. Mason and M.G. Cox, eds, Algorithms for Approximation, pp.143-167. 1987.

[QUI-86] R. R. Quinlan. "Induction of decision trees". Machine Learning, 1: pp.81-106, 1986.

[QUI-87] J. R. Quinlan. "Simplifying decision trees". International Journal of Man-Machine Studies, 27: pp. 221-234, 1987.

[QUI-93] J. R. Quinlan. "C4.5: Programs for machine learning". Morgan Kaufmann, Los Altos, California, 1993.

[RAM-03] Á. Ramírez. "EL FILTRO DE KALMAN" Documento de trabajo del Banco Central de Costa Rica, elaborado en la División Económica, Departamento de Investigaciones Económicas. 2003.

[REN-88] S. Renals. "Radial basis functions network for speech pattern classification". *Electronics Letters*, 25:. pp.437-439, 1988.

[RIP-96] B.D.Ripley. “Pattern recognition and neural networks”. Cambridge university press, 1996.

[RIV-03] V.Rivas. “Optimización de Redes Neuronales de Funciones Base Radiales Mediante Algoritmos Evolutivos”. Tesis Doctoral, universidad de Granada 2003.

[ROD-02] P. T. Rodríguez-Piñero. “Introducción a los algoritmos genéticos y susaplicaciones Madrid”.2002.

[RUM-86] D.Rumelhart, G.Hinton, R.Williams, “Learning representation by backpropagating errors”, Nature 323. pp. 533-536. 1986.

[RUN-99] T.A.Runkler and J.C.Bezdek, “Alternating cluster estimation: A new tool for clustering and function approximation”. IEEE Trans. Fuzz Syst., vol. 7, pp. 377-393, Aug. 1999.

[RUS-99] M. Russo and G. Patanè. “Improving the LBG Algorithm”. In Lecture Notes in Computer Science. New York: Springer-Verlag, vol. 1606, pp. 621-630. 1999.

[SAH-90] A.Saha, J.Christian, DS.Tang and C.L.Wu. “Oriented non-radial basis functions for image coding and analysis”. Proceedings of IEEE, Neural Information Processing Systems, 3, pp.728-734. 1990.

[SET-97] R. Setiono. “Neural network feature selector”. IEEE Trans. Neural Networks,8(3): pp. 654-662, 1997.

[SMI-02] L.I.Smith. “A tutorial on Principal Components Analysis”. February 26, 2002.

[SOU-02] F.J, de Souza,. Vellasco, M.M.R., Pacheco, M.A.C.: Hierarchical neuro-fuzzyquadtree models. Fuzzy Sets and Systems 130. pp. 189-205. 2002.

[STE-76] S.D.Stearns. “On selecting features for pattern classifiers”. In Proceedings of the 3rd International Conference on Pattern Recognition, pp. 71-75, Coronado, CA, 1976.

[SUD-94] T. Sudkamp and R, J,Hammell.. Interpolation, Completion and LearningFuzzy Rules, IEEE Transactions on Systems, Man and Cybernetics, Vol. 24 No. 2 pp 332-342, February 1994.

[TAU-02] S-Y.Taur.Kung, Sh-H.Lin. “Hierarchical Fuzzy Neural Networks for Pattern Classification”. National Science Council through Grant NSC 88-2213-E-005-012. 2002.

[TUN-02] Sh.Tun. Shu-Ch, Chen. “Function Approximation using Robust Wavelet Neural Networks”. Department of Information Management, National Kaohsiung First University of Science and Technology, Kaohsiung, Taiwan, 2002.

[VAC-98] G. Vachtsevanos and P. Wang, "A Wavelet Network Framework for Diagnostics of Complex Engineered Systems," Proceedings of MARCON 98 Maintenance and Reliability Conference, Knoxville, TN, invited paper. May 12-14, 1998,

[VEH-02] A.Vehtari. and J.Lampinen. “Bayesian Input Variable Selection Using Posterior Probabilities and Expected Utilities”. Report B. ISSN. pp.1457-1404. 2002.

[VIÑ-03] P.Viñuela. I.León. “Redes de Neuronas Artificiales Un Enfoque Práctico”. Pp. 75-101, 2003.

[WAL-91] P. Wallich, “Wavelet theory: An analysis technique that’s creating ripples,” Scientific American, Jan. 1991.

[WEL-04] G. Welchand, G. Bishop. “An Introduction to the Kalman Filter”. Department of Computer Science University of North Carolina at Chapel Hill. April 5, 2004

[WER-74] P.Werbos. “Beyond regression: new tools for prediction and analysis in the behavioural sciences”. PhD thesis, Harvard University. 1974.

[WER-74] P.J.Werbos,. “Beyond regression: new tools for prediction an analysis inbehavioral sciences”. Tesis doctoral no publicada. Harvard University. 1974.

[WHI-96] 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*, vol. 7, no. 4, pp. 869-880, 1996.

[YAN-97] H.Yang. S. Amari.: Adaptive online learning algorithms for blind separation: Maximum entropy and minimum mutual information, Neural Comput., vol. 9 pp.1457-1482. 1997.

[YU-03] S.YU. “Feature Selection and Classifier Ensembles: A Study on Hyperspectral Remote Sensing Data”. PHD Thesis in The University of Antwerp, 2003.

[ZHA-92] Q. Zhang and A.Benveniste. “Wavelet networks”. IEEE Transactions on Neural Networks, vol. 3, no. 6, pp. 889-898, Nov. 1992.

[ZHA-95] J.Zhang, G.G.Walter, Y.Miao, and W.N.W.Lee. “Wavelet Neural Networks for Function Learning”. IEEE Trans. Signal Processing, Vol. 43, pp.1485-1497, June 1995.

[ZHA-97] Q. Zhang, ”Using Wavelet Network in Nonparametric Estimation”, IEEE Trans. Neural Network, Vol. 8, pp.227-236, 1997.